A 33MHz 16-Bit Gradient Calculator for Real-Time Volume Imaging

Martin Margala, Nelson G. Durdle, Scott Juskiw, [1]
V. James Raso, and Doug L. Hill [2]

1. Electrical Engineering Department, The University of Alberta
Edmonton, Alberta, Canada T6G 2G7
2. Rehabilitaion Engineering Department, Glenrose Rehabilitation Hospital
Edmonton, Alberta, Canada T5G 0B7

Abstract

This paper describes a gradient calculator which forms an important part of a shading processor being developed for a high resolution high performance real-time general purpose volume imaging system. The proposed architecture overcomes current image resolution and frame-rate limitations through the use of custom high-speed processors. The gradient calculator evaluates three arithmetic operations: a square and add operation, square-root, and three division operations. Input-output delay time is 30 ns with an accuracy of ±0.78%. The algorithms and implementation in silicon are described in detail.

1. Introduction

A Volume Imaging System (VIS) is a powerful tool in real-time manipulation of complex 3D data sets. Applications that require such service are increasing. The data can be of medical, geophysical, aerospace or other origins. Medical treatment planning sytems using high speed graphics workstations can provide powerful clinical tools for diagnostic evaluation of radiation therapy, for control in computer assisted surgery, for laser surgery simulations, dental diagnostics or for facial reconstructions [1,2]. Geophysical imaging systems use rendering and displaying capabilities to visualize oil and gas deposits. Imaging systems are used in the airline industry to localize possible leaks and defects in metal structures. Such facilities would achieve maximum information extraction through high resolution reconstructions of examined structures. An interactive user interface requires real-time image update and continuous feedback to operator actions. This necessitates updates at the rate of 30 frames per second. The majority of existing 3D imaging systems employ general purpose workstations to manipulate complex data sets compiled from computed tomography, magnetic resonance images, or other scanned images. The processed data are three-dimensional density volumes of 256^3 or 512^3 generated as a stack of slices. This amount of data creates major problems for software-oriented 3D reconstruction techniques. Software based systems are too slow to provide real-time user feedback. Development of reconstruction methods using high speed hardware has been actively pursued [3-9]. Traditional computer graphics software approaches, even those using hardware assist, cannot convey the detailed information contained in volume data sets in real-time. A voxel-based architecture GODPA (generalized object display process architecture), a derivative of the Voxel Processor [3,4] has been implemented as a prototype system of 64^3 voxels, with a 16-frames-per-second image update rate. Jackel and Strasser [5] described the PARCUM II system, which is used for handling solids reconstructed from CT scans. An emulation of this system generated medical images in 38-110 seconds from data sets with 256^3 elements. Two projects have been proposed from Keio University in Japan. One is a prototype system SCOPE (solid and colored object projection environment) [6] and the other system—which is very similar to GODPA—is the $3DP^4$ architecture [7]. $3DP^4$ has been simulated in software, and a 256^3 hardware implementation is estimated to have a throughput of up to 10 frames per second. A commercial image system, the Insight system of Phoenix Data Systems [8], combines hardware and software. It can display complex objects at a "near real-time" rate of about one frame per second. Kaufman and Bakalash [9] developed the CUBE architecture and constructed a prototype 16^3 system. All of these systems exhibit either low frame rates or low resolution.

Our objective is to design a system which overcomes these limitations with a high performance architecture which will render and display high resolution volume images from 256^3 and 512^3 data sets in real-time, that is at least 30 frames per second.

The objective of this paper is to present algorithms that yield an efficient, fast architecture for computing the normalized gradient magnitude and the normalized gradient direction.

2. Gradient Calculation

This paper describes a gradient calculator which forms a subunit of the imaging system. The volume data received from computed tomographs or magnetic resonance scans must be shaded and composited to produce a 3D image of the observed structure. The shading processor performs several arithmetic and logic operations. The first and most complex operation is the calculation of a normalized gradient direction and normalized gradient magnitude. The gradient parameters define the surface information of a single 3D volumetric element called a *voxel*.

In a volumetric data set, each *voxel* has 6 face-connected neighbours, 12 edge-connected neighbours, and 8 corner-connected neighbours. Gradients can be calculated using any subset of these neighbours. The simplest case uses the face-connected neighbours. For example, the gradient of point (i, j, k) :
in the *X* direction, is given by

$$\frac{S(i+1,j,k)-S(i-1,j,k)}{\Delta x}, \qquad (1a$$

in the Y direction, by

$$\frac{S(i,j+1,k)-S(i,j-1,k)}{\Delta y}, \text{ and} \qquad (1b$$

in the Z direction, by

$$\frac{S(i,j,k+1)-S(i,j,k-1)}{\Delta z}. \qquad (1c$$

The gradient magnitude is given by :

$$|Gijk|=\sqrt{G(x)ijk^2+G(y)ijk^2+G(z)ijk^2} \qquad (2$$

with the direction defined by :

$$Gijk=[\frac{G(x)ijk}{|Gijk|},\frac{G(y)ijk}{|Gijk|},\frac{G(z)ijk}{|Gijk|}]$$
$$=[g(x)ijk,g(y)ijk,g(z)ijk] \qquad (3$$

In an isometric volume Δx, Δy, Δz are constant and equal. It is assumed that the sampling distance is equal to one unit.. Therefore, the components $G(x),G(y),G(z)$ simplify to:

$$G(x)ijk=S(i+1,j,k)-S(i-1,j,k), \qquad (4a$$
$$G(y)ijk=S(i,j+1,k)-S(i,j-1,k), \qquad (4b$$
$$G(z)ijk=S(i,j,k+1)-S(i,j,k-1), \qquad (4c$$

where for 8-bit data:

$$0\le G(x)ijk\le 255, \qquad (4d$$
$$0\le G(y)ijk\le 255, \qquad (4e$$
$$0\le G(z)ijk\le 255, \qquad (4f$$

It is then obvious that :

$$0\le|Gijk|\le\sqrt{195075}, \qquad (5a$$

and:

$$0\le g(x)ijk\le 1, \qquad (5b$$
$$0\le g(y)ijk\le 1, \qquad (5c$$
$$0\le g(z)ijk\le 1, \qquad (5d$$

To maintain real-time performance of the imaging system the gradient calculator must operate in less than 100 ns with the result error not exceeding 5%. Errors above this limit are detectable by human eye. With these requirements it takes 8 imaging modules including gradient calculators to process 256^3 data, 64 to process 512^3 data and 512 to process 1024^3 data.

From the equations above, the calculation of the gradient magnitude and normalized gradient direction involves a square and add operation, a square-root operation (eq.2), and three division operations (eq.3). There is no off-the-shelf circuit or subsystem available to perform these operations in an optimal way for 8 or 16-bit fixed point numbers.

3. Developed Architecture

The gradient calculator has three inputs (gradient components, eq. 4a to 4c) and four outputs (3 normalized gradients, eq. 3 and normalized gradient magnitude eq. 2). The block diagram of the gradient calculator is presented in Figure 1. This architecture performs three operations: ***square and add in parallel, square-root,*** and ***division.***

3.1. Square and Add

The first term to be calculated in obtaining the gradient magnitude is the sum of the squares of the inputs. Dadda [10] showed an implementation of squarers in serial form. His regular multiplier array consisting of factors $x_i x_j$ was reduced because $x_i x_j = x_j x_i$, therefore, the partial products above the anti-diagonal (in anti-diagonal is x_i^2) are equivalent to partial

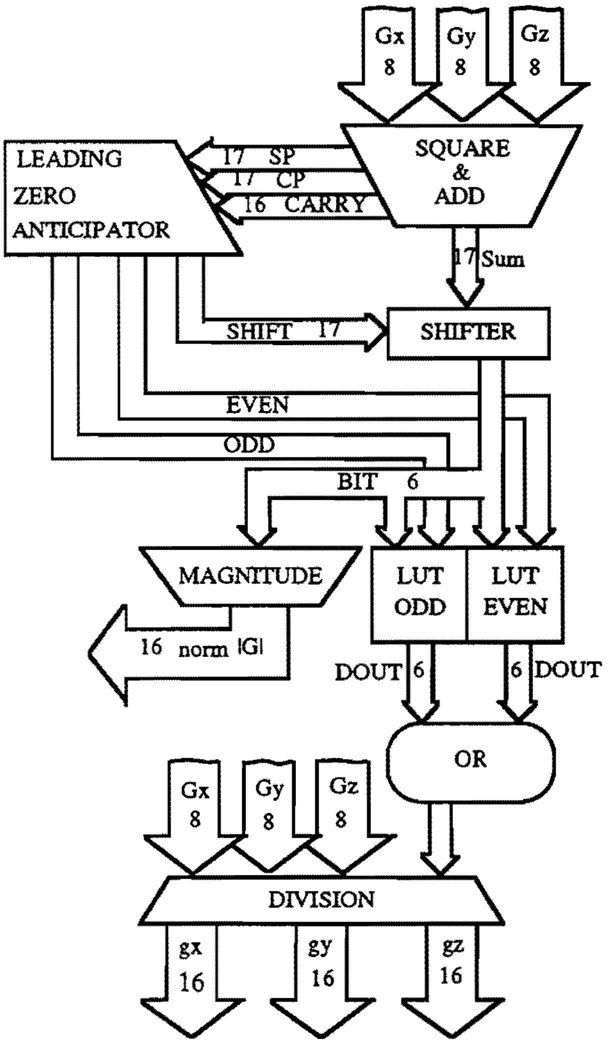

Figure 1. Block Diagram of the Gradient Calculator.

products below (Figure 2). The regular partial product array can thus be replaced by a reduced equivalent array comprising the anti-diagonal and one of the parts shifted by one position to the left (Figure 3). This significantly reduces the number of partial products generated by the multiplier. If this scheme is implemented with a high speed enhanced multiplier structure [11], square and add operations can be performed in parallel.. The multiplier-array consists of 3 parts, each part squares G_x,

G_y, and G_z separately. Joining squarers into one array results in the adding operation being evaluated concurrently with the squaring (Figure 4). The longest path requires the addition of 15 products (column S9). This array is also known as a partial product generator (PPG). In the next step a carry-save adder array (CSA) is used to reduce the number of products (Figure 5). The final step is a carry-propagate adder (CPA). *Carry-save adders (CSA)* are used for fast accumulation of partial product terms [11,12]. Speed and ease of use have made 4:2 compressors the reduction circuit of choice in a number of multiplier designs [13,14,15]. A carry-propagate adder adds carry and sum and produces the result. It consists of an array of 4-bit Manchaster chains (Figure 5). The result is 18 bits long, but only 17 bits are used for futher calculations.

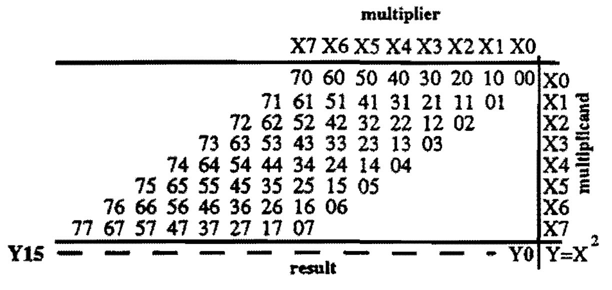

Figure 2. Regular multiplication array.

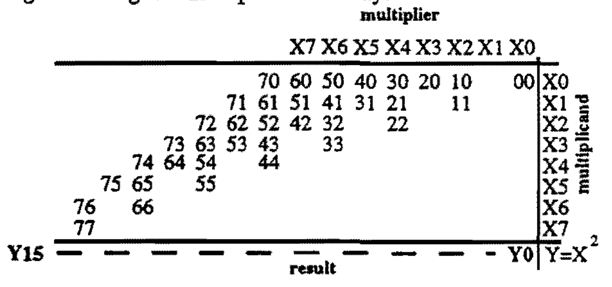

Figure 3. Simplified multiplication array for squarers.

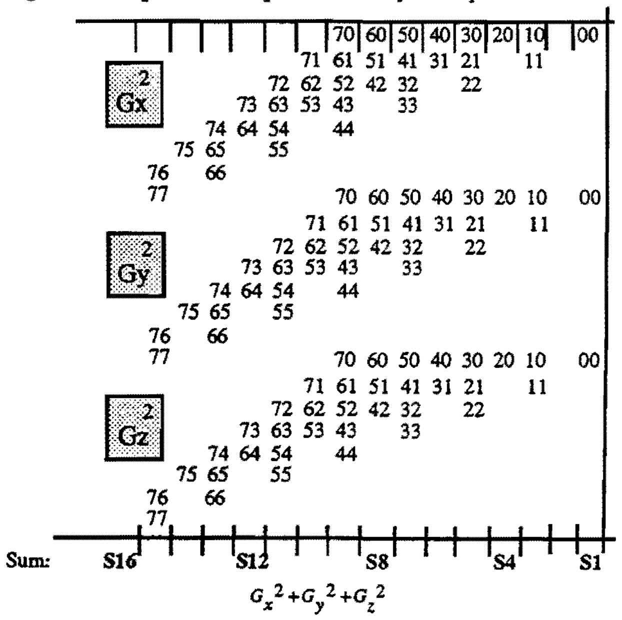

Figure 4. Multiplier array with square and add operation.

In this application 3:2 counter circuits and 4:2 compressor circuits were used in three stages to obtain the best performance. To avoid long interconnects that decrease the overall delay, a *Regularly Structured Tree (RST)* approach [16] for multiplication was used. A recursive algorithm to generates a *sum* and *carry* term for the CPA and provides a regular layout structure. The advantages are reduced wiring length and increased density of transistors resulting in a design which is both area and time efficient.

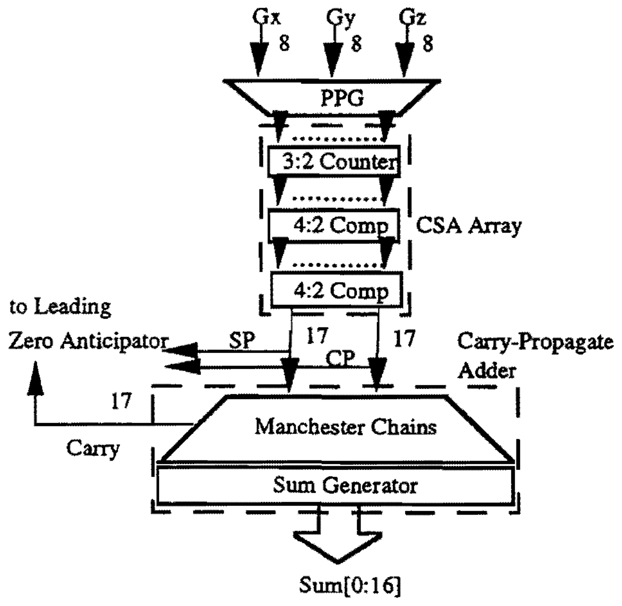

Figure 5. Square & Add Circuit.

3.2. Square-root

Square-root has been considered one of the most important arithmetic functions since the early stages of computer development [17]. It's wide range of applications have forced designers to look for an optimal algorithms and simple implementations. With image processing applications coming forward, there has been further development on relatively high-speed square-rooting techniques [18]. The most widely used method for evaluating the square-root of a number is the Newton-Raphson iteration technique. However, this method requires a division operation, which makes the implementation more complex and slows down the overall performance.

The gradient calculator uses a simplified version of Hashemian's square-root algorithm. Hashemian's technique [19] is very fast, because it involves no division operation except division by 2 (bit-shift operation). The algorithm itself consists of two parts. In the first part an estimate is obtained with an average accuracy 1.7% (±0.85%) and in the worst case up to 6%. The procedure in the second part modifies the initial estimate iteratively, until an exact root is evaluated. The gradient calculator precisely follows part one for obtaining a first estimate. The algorithm considers that every given integer A has $2m$ number of effective bits (the MSB is always "1"), which is an even number of bits. If the number of bits is odd (i.e. 101 - has 3 effective bits) then one left shift is performed. After this the procedure has two steps:

a) apply $m+1$ right shifts to integer A, and
b) insert a "1" bit to the left of the most significant bit (MSB) of the result in step a).

The advantage of this implementation is in its concurrency (Figure 6). The leading-zero-anticipator (LZA) locates the position of the MSB in the summand $(G_x^2+G_y^2+G_z^2)$, calculates the number of effective bits and generates a pair of control signals (EVEN/SHIFT or ODD/SHIFT). These signals are activated in the same time as the summand from the Square & Add circuit is finalized. The result is then shifted by an appropriate number of bits.

The simplification is in the implementation of the second part of the algorithm. The gradient calculator evaluates only the first step of the iteration. It doesn't follow the Hashemian's algorithm rather it uses look-up-tables to compute the next operation. There are two look-up-tables used for the first iteration step, one for even and another for odd numbers with values that compensate for an error of the initial estimate. The initial estimate is used as an index to these look-up-tables. The index then retrieves a value $1/(|G|+g_{err})$. The value g_{err}, a ***compensating divisor*** , brings the overall accuracy for the normalized gradient direction to ±0.78%. The MAGNITUDE circuit is an enhanced multiplier similar to that constructed for the Square and Add circuit. It modifies the error for gradient magnitude and normalizes the result in the same time. The accuracy of the $norm|G|$ is ±0.85%.

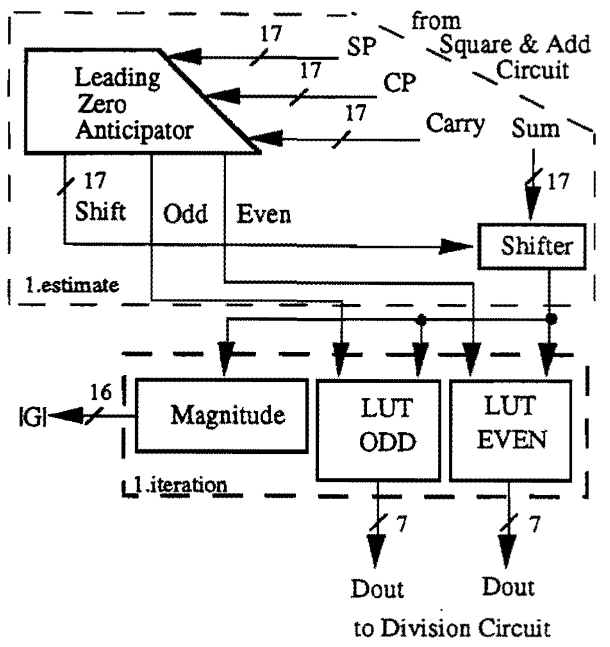

Figure 6. Square-Root Circuitry.

3.3. Division

The iteration block illustrated in Figure 6 is multifunctional with iteration, division, and normalization operations overlaping. Division is the most complex arithmetic function to implement. Several methods deal with the division of integer and real numbers. One approach introduced by Alverson [20] uses reciprocals. It was designed for 64-bit integer numbers. With reciprocal approximations, integer division can be synthesized from a multiply followed by a shift. However, if reciprocal values are not carefully selected, the quotient obtained often incurs "off-by-one errors", requiring a correction step. The reciprocal computation is sufficiently fast with one look-up-table and five multiplies. Wong and Flynn presented two methods for division of integer binary numbers, a basic and an advanced method [21]. Both of them are based on a look-up-table and Taylor series of approximations of the reciprocal. The basic method considers 3 binary representations X as a dividend, Y as a divisor, and Q as a seeking quotient. Binary numbers X and Y are first left-shifted, normalized, X by *j-bits*, and Y by *k-bits*. Several stages of the algorithm follow including an iteration. The gradient calculator uses the simplified basic method. It calculates only one equation $Q=X_h*(1/Y_h)*(1/2^{(j-k)})$, where X_h and Y_h are normalized binary representations of X and Y. In the look-up-tables (Figure 4) the values of $1/Y_h$ are stored. These values are selected by the first estimate of the square-root (index to LUT) as described in the previous section. The gradient inputs G_x, G_y, G_z represent the X_h binary number. The reciprocal values were examined for all possible values of gradient magnitude $|G|$ which range from 1 to $\sqrt{195075}$ and could be matched to 64 reciprocal values. Therefore, the size of the look-up-tables is *64x6bits*. The DIVISION cicuit takes $1/Y_h$ from a selected LUT, multiplies it by a gradient and shifts the product in the multiplier array concurrently. The final result is a 16 bit long normalized gradient direction. The DIVISION circuit is constructed as a high speed multiplier, similar to the one used to square and add. The multiplier differs in that it is implemented with 4:2 compressor circuits in all stages of the CSA.

4. Simulations and Results

In the gradient calculator, the dominant architecture is a custom high speed multiplier. There are three in the design: S&A circuit, DIVISION circuit and MAGNITUDE circuit. The performance of these three structures directly affects the performance and reliability of the entire architecture. The gradient calculator was simulated hierarchically for speed and logic correctness; starting from the basic cells up to the major circuits. The processing time of each part of the gradient calculator was determined using HSPICE simulation tools in the Cadence environment. Cadence CAD package was used throughout the entire design process. Logic evaluation was done with Cadence Verilog-XL tools. Emphasis is on the results obtained from the simulation of the dynamic 1.2μ CMOS 4:2 compressor circuit. The compression time was comparable to the design of Mori, et.al. (13) , using 0.5μ CMOS technology, (Table 1). The dynamic 4:2 compressor is shown in Figure 7 and the static 4:2 compressor is presented in Figure 8.

The total number of transistors used in the gradient calculator is 17716 therefore it is considered LSI type circuit.. Table 2 shows the number of transistors used in each circuit and their delays in nanoseconds.

The gradient calculator was originally designed using 1.2u static CMOS technology. It requires 30ns processing time. Many essential parts of the multiplier structures were redesigned using dynamic logic. These improvements are being simulated to guarantee desired functionality. New layout schemes are being considered to further increase compactibility and speed. Our main objective is to construct the entire imaging system using 0.8u BiCMOS technology which will be available to us in the second half of 1994. This technology will increase speed by up to an additional 33%. This will give more flexibility in other parts of the imaging system where changes in the design bring no further improvement and become a bottleneck for the system and it will bring a significant reduction in necessary hardware and thereby overall reduction in costs.

5. Conclusion

Using the algorithms described in this paper, a gradient calculator was constructed that requires 30 ns processing time and produces results with an accuracy of ±0.78% (for the normalized gradient directions) and ±0.85% (for the normalized gradient magnitude). With currently available fabrication technology, the gradient calculator exceeds the speed requirements to achieve real-time volume imaging.

Acknowledgments

This research was partially supported by Alberta Microelectronic Centre. We also acknowledge technical support from Canadian Microelectronic Corporation.

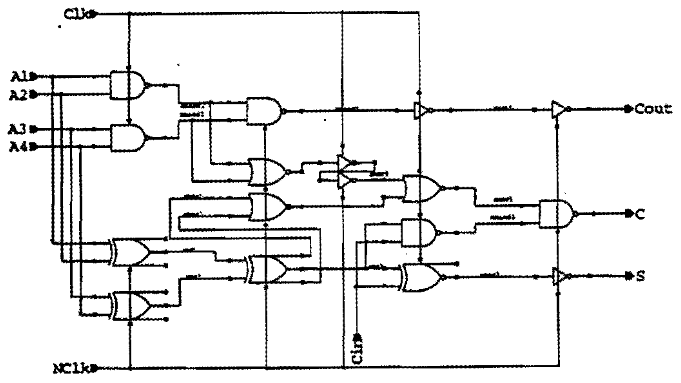

Figure 7. Dynamic 4:2 compressor circuit.

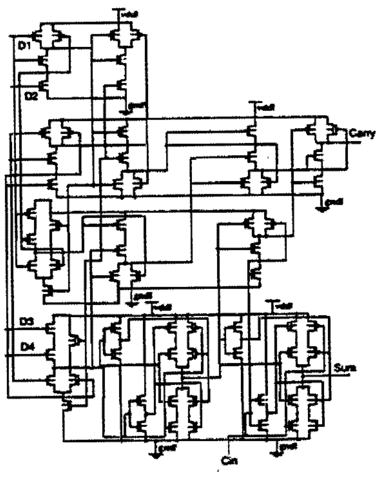

Figure 8. Static 4:2 compressor circuit.

Design	CMOS logic	Technology	Delay(ns)
Design in (13)	pseudo	0.5u	1.2
Current design	static	1.2u	3.0
New design	domino	1.2u	1.3

Table 1. Comparison of the 4:2 compressor circuits.

Circuits	Numberof transistors	Delay(ns)
Square & Add	5566	11.54
LZA	1216	8.5
Shifter	111	0.5
LUT EVEN	1589	0.840
LUT ODD	1477	0.927
OR	42	0.8
Division	3653	11.25
Magnitude	4062	11.37

Table 2. Number of transistors and processing time of each circuit.

References

[1] Brewster, L.J., Triveldi, S.S., Tuy, H.K., and Udupa, J.K., *Interactive Surgical Planning,* IEEE Comp. Graph. and Appl., vol.4, No.3, March 1984, pp.31-40.

[2] Kall, B.A., Kelly, P.J., and Goerss, S.J., *Comprehensive Data Collection Treatment Planning and Interactive Surgery,* In : Medical Imaging, SPIE, vol.767, 1987, pp.509-514.

[3] Goldwasser, S.M., *A Generalized Object Display Processor Architecture,* IEEE Comp. Graph. & Appl., vol.4, No.10, Oct. 1984, pp.43-55.

[4] Goldwasser, S.M., Reynolds, R.A., Bapty, T., Baraff, D., Summers, J., Talton, D.A., and Walsh, E., *Physician's Workstation with Real-Time Performance,* IEEE Comp. Graph. & Appl., Dec. 1985, pp. 44-57.

[5] Jackel, D., and Strasser, W., *Reconstructing Solids From Tomographic Scans-The PARCUM II System,* In:Advances in Computer Graphics Hardware II, Kuijk, A.A.M., and Strasser, W., (Eds.), Springer-Verlag, Berlin, 1988, pp.209-227.

[6] Uchiki, T., and Tokoro, M., *Solid and Colored Object Environment,* Transactional Institution of Electronics and Commercial Engineers of Japan, vol.68-D, No.4, April 1985.

[7] Ohashi, T., Uchiki, T., and Tokoro, M., *A Three-Dimensional Shaded Display Method for Voxel-Based Representation,* Eurographics 85, Amsterdam, September 1985, pp.221-232.

[8] Meagher, D.J., *Interactive Solids Processing for Medical Analysis and Planning,* Proceedings NCGA 84, NCGA, Fairfax, Va., 1984, pp.96-106.

[9] Kaufman, A., and Bakalash, R., *Memory and Processing Architecture for 3D Voxel Based Imagery,* IEEE Computer Graphics & Applications, November 1988, pp.10-23.

[10] Dadda, L., *Squares for Binary Numbers in Serial Form,* IEEE 1985 Symposium on Computer Arithmetic, 1985, pp.173-180.

[11] Juskiw, S., Durdle, N.G., Raso, V.J., Hill, D.L., *High Speed Image Composition with Enhanced Multiplier Structure,* unpublished.

[12] Wallace, C.S., *A Suggestion for Fast Multiplier,* IEEE Transactions on Electronic Computers, vol. EC-13, February 1964, pp.14-17.

[13] Mori, J., Nagamatsu, M., Hirano, K., Tanaka, S., Noda, M., Toyoshima, Y., Hashimoto, K., Hayashida, H., and Maeguchi, K., *A 10ns 54x54-b Parallel Structured Full Array Multiplier with 0.5-um CMOS Technology,* IEEE Journal of Solid-State Circuits, vol.26, No.4, April 1991, pp.600-606.

[14] Goto, G., Sato, T., Nakajima, M., and Sukemura, T., *A 54x54-b Regularly Structured Tree Multiplier,* IEEE Journal of Solid-State Circuits, vol.27, No.9, September 1992, pp.1229-1236.

[15] Santoro, M.R., and Horowitz, M.A., *SPIM: A Pipelined 64x64-bit Iterative Multiplier,* IEEE Journal of Solid-State Circuits, vol.24, No.2, April 1989, pp.487-493.

[16] Sharma, R., *Area-Time Efficient Arithmetic Elements for VLSI Systems,* at 8-th Symposium on Computer Arithmetic, IEEE, 1987, pp.57-62.

[17] Flores, I., *The Logic of Computer Arithmetic,* Prentice-Hall, Inc., 1963.

[18] Montuschi, P., and Ciminiera, L., *Simple Radix 2 Division and Square-Root with Skipping of Some Addtion Steps,* at 10-th Symposium on Computer Arithmetic, IEEE, 1991, pp.202-209.

[19] Hashemian, R., *Square-Rooting Algorithms for Integer and Floating-Point Numbers,* IEEE Transactions on Computers, vol.39, No.8, August 1990, pp.1025-1029.

[20] Alverson, R., *Integer Division Using Reciprocals,*at 10th Symp. on Comp.Arith., IEEE, 1991, pp.186-190.

[21] Wong, D., and Flynn, M., *Fast Division Using Accurate Quotient Approximations to Reduce the Number of I terations,* IEEE Trans. on Comp., vol:41, No.8, Aug. 1992, pp.981-995.