COLOR EDGE DETECTION HARDWARE BASED ON GEOMETRIC ALGEBRA

Biswajit Mishra, Peter Wilson

Electronic Systems Design, School of Electronics and Computer Science,
University of Southampton, UK, SO17 1BJ
{bm03r,prw}@ecs.soton.ac.uk, Fax +44.2380.592901

Keywords: Rotor Convolution, Geometric Algebra, Hardware, Edge Detection.

Abstract

Modern techniques treat color images as separate monochrome images for processing. Partly, because there is no straightforward generalization of linear filters available for color. However the algorithms yield more accurate results when the correlation among color bands are exploited which shows fundamental difference to process the color images. Earlier work[8] reported the transformation of the color images using Quaternion Fourier Transforms and the realization of a holistic filter based on Quaternion convolution. Here, we discuss the rotor based convolution techniques, a generalization of the previous work, within a new mathematical framework in Geometric Algebra. Based thereupon, a novel hardware architecture is proposed. Experiments show the edge detection with this technique belong to a class of linear vector filter and is holistic in nature. It is tailored for image processing applications, providing an acceptable application performance requirements. The usefulness of the introduced approach was demonstrated by analyzing and implementing a computation intensive edge detection algorithm on this hardware.

1 Introduction

Human visual perception doesn't differentiate color separately but tends to process it as a whole. In this sense, the compact representation of color as vector and operating on these vectors are particularly interesting. This also gives rise to algorithms in useful applications which pose challenges in the realization of nonlinear vector filters. Present day techniques don't treat the color images wholly, but as separate monochrome images. Firstly, because there is no straightforward generalization of linear filters available for color images. Secondly, there is no simple mathematical framework for the linear filters for color images. But when the correlation among the color bands are exploited the algorithms yield more accurate results [10]. hence there is a fundamental difference to apply monochrome images algorithms to color images.

Previous work[9] reported the transformation of the color images using Quaternion Fourier Transforms and the realization of a holistic filter based on Quaternion convolution. This paper discusses the rotor based convolution approach within the mathematical framework in Geometric Algebra, and the linear filter for vector processing. The benefit of using such a mathematical framework is manifold. Firstly the 3-D homogeneous regions are modeled as bivectors (explained in Sec 2) in the graded linear space of Geometric Algebra. Hence extension to the algorithms from the 3-D space becomes easy and the same rules apply for the n-dimensional problems. Secondly, the problem space becomes linear because an n-D space can always be represented as a graded linear space consisting 2^n elements.

Based thereupon, the hardware architecture design is discussed. The focus of the paper is the introduction of a new hardware architecture for application areas involving Image Processing. Its performance and potential are discussed by the implementation of a holistic filter design for an edge detection algorithm in Image Processing application. Color edge detection is used to find the discontinuities along the adjacent regions of a color image. These are useful in several applications involving imaging (medical imaging, detection of microarray images) and vision.

The paper is structured as follows: in section 2 the mathematical framework in Geometric Algebra is discussed. Section 3 provides the concepts of *rotors* in Geometric Algebra. The usefulness of the rotor concept is then discussed in section 4. The usefulness and the reason to develop an application specific hardware architecture is illustrated in section 5 which gives analysis on performance benefits in image processing algorithms operating on color vectors. In this section a very brief review of the the hardware architecture is presented. Finally in Section 6 we conclude with results on a color based edge detection application involving hardware.

2 Geometric Algebra

2.1 Basic Definitions

In this section we discuss the basic definitions in Geometric Algebra. For a more detailed review please see [5][6]. Scalars, vectors, bivectors and trivectors (sweeping a bivector $\mathbf{a} \wedge \mathbf{c}$ along another vector $\mathbf{c}$) represent 0, 1, 2 and 3 dimensional subspaces respectively within Geometric Algebra. The elements of the graded linear vector space is given in the the following (Table 1) which contain all the homogeneous elements for the three dimensional space. In Geometric Algebra it is possible to add quantities of different grades (e.g. 0 grade scalar, 1 grade vector and 2 grade bivector) resulting in a multivector. One can manipulate expressions on multivectors which gives rise to meaningful geometric information.

Element	Grade	basis k-blade	total
0-blade or scalar	0	1	1
1-blade or vectors	1	e_1, e_2, e_3	3
2-blades or bivector	2	e_1e_2, e_2e_3, e_3e_1	3
3-blade or trivector	3	$e_1e_2e_3$	1

Table 1: Homogeneous elements of 3D Geometric Algebra

2.2 Geometric Product

The fundamental building block in Geometric Algebra is called the geometric product. The geometric product for two vectors **a** and **b** is defined as $\mathbf{ab} = \mathbf{a}\cdot\mathbf{b} + \mathbf{a}\wedge\mathbf{b}$. This consists of the inner product and the outer product which gives the information about the magnitude and direction and orientation of the vector. The inner product **a** and **b** results in a scalar quantity and is expressed as $\mathbf{a}\cdot\mathbf{b}$. It conveys the relative direction of the two vectors. The outer product, though shares same properties as the vector product, is fundamentally different, is generalizable to higher dimensions and conveys the orientation information. For example, if **a** and **b** are collinear, then $\mathbf{a}\wedge\mathbf{b} = 0$, the geometric product gives the magnitude of the vectors. If **a** and **b** are perpendicular, then $\mathbf{a}\cdot\mathbf{b} = 0$ and the geometric product gives the orientation of the bivector. The inner product is commutative and the outer product is anticommutative. Hence the geometric product is neither symmetric like the inner product nor antisymmetric like the outer product but is invertible. So the geometric product of **ba** is given by $\mathbf{ba} = \mathbf{b}\cdot\mathbf{a} + \mathbf{b}\wedge\mathbf{a} = \mathbf{a}\cdot\mathbf{b} - \mathbf{a}\wedge\mathbf{b}$. By addition of **ab** and subtraction of **ba**, a more generalized definition of the inner and outer product is obtained. Therefore, the inner product is given by $\mathbf{a}\cdot\mathbf{b} = \frac{1}{2}(\mathbf{ab}+\mathbf{ba})$ and the outer product is given by $\mathbf{a}\wedge\mathbf{b} = \frac{1}{2}(\mathbf{ab}-\mathbf{ba})$. In essence this is the most important element of this algebra and all the other meaningful operations can be derived from this geometric product algebraically. The importance of this product computation is evident in many science and engineering applications [4].

Let $\mathbf{e_1}, \mathbf{e_2}$ and $\mathbf{e_3}$ be the orthonormal vectors for a three dimensional Euclidean space. The relationship $e_ie_j = \delta_{ij}$ leads to a basis calculation in the algebra. The rules of the geometric product calculation which encapsulates the full algebra is given by eqn (1):

$$\mathbf{e_ie_j} = \begin{cases} -\mathbf{e_je_i}, & \text{if } i \neq j \\ \mathbf{1}, & \text{if } i = j \end{cases} \tag{1}$$

In the graded linear space spanned by the **basis vectors** $\mathbf{e_1}, \mathbf{e_2}$ and $\mathbf{e_3}$, the space in $\Re^3$ will have $2^3 = 8$ blades or elements as shown following. The highest grade element is called the pseudoscalar and is denoted by a symbol **i** or **I**.

$$\underbrace{1}_{\text{scalar}},\ \underbrace{e_1, e_2, e_3}_{\text{vector}},\ \underbrace{e_1\wedge e_2, e_2\wedge e_3, e_3\wedge e_1}_{\text{bivector}},\ \underbrace{e_1\wedge e_2\wedge e_3}_{\text{trivector}(\mathbf{I})}$$

.

3 Rotations in the Geometric Algebra 3-D space

As described in the previous section, the pseudoscalar $\mathbf{I} = e_1\wedge e_2\wedge e_3$ squares to -1 and commutes with all the vectors. For example, $e_1e_2 = \mathbf{I}e_3, e_2e_3 = \mathbf{I}e_1, e_3e_1 = \mathbf{I}e_2$. Also, the bivectors rotate the vectors by 90° in their own plane (e.g. $(e_1e_2)e_2 = e_1, (e_2e_3)e_2 = -e_3, (e_3e_1)e_1 = e_3$).

Before we progress to discuss the rotational element in Geometric Algebra and its significance in engineering applications we can agree that any rotation is represented by a pair of reflection. Let us consider any vector **a** being reflected in a plane perpendicular to an unit vector **n**. The reflected vector **a'** is expressed as $\mathbf{a}' = \mathbf{a}_\perp - \mathbf{a}_\parallel$. Where $a_\perp$ is the perpendicular component and $a_\parallel$ is the parallel projection of the vector. Expanding the terms for **a** and **a'** we get,

$$\begin{aligned}\mathbf{a} = \mathbf{n^2a} &= \mathbf{n}(\mathbf{n}\cdot\mathbf{a} + \mathbf{n}\wedge\mathbf{a}) \\ &= (\mathbf{n}\cdot\mathbf{a})\mathbf{n} + \mathbf{n}(\mathbf{n}\wedge\mathbf{a})\end{aligned} \tag{2}$$

Therefore, $a_\parallel = (\mathbf{n}\cdot\mathbf{a})\mathbf{n}$ and $a_\perp = \mathbf{n}(\mathbf{n}\wedge\mathbf{a})$. Hence

$$\begin{aligned}\mathbf{a}' = \mathbf{a}_\perp - \mathbf{a}_\parallel &= \mathbf{n}(\mathbf{n}\wedge\mathbf{a}) - (\mathbf{n}\cdot\mathbf{a})\mathbf{n} \\ &= -(\mathbf{n}\cdot\mathbf{a})\mathbf{n} - (\mathbf{n}\wedge\mathbf{a})\mathbf{n} = -\mathbf{nan}\end{aligned} \tag{3}$$

Hence the resultant reflection of **a** in the plane perpendicular to unit vector **n** is **-nan**. It is observed that the reflection of ***a*** in the plane perpendicular to **n** followed by another reflection in the plane perpendicular to **m** results in another vector given by$-\mathbf{m}(-\mathbf{nan})\mathbf{m} = (\mathbf{mn})\mathbf{a}(\mathbf{nm}) = \mathbf{Ra}\tilde{\mathbf{R}}$. This product **R** is a multivector and is called a rotational element or Rotor and satisfies $\mathbf{R}\tilde{\mathbf{R}} = 1$, where $\tilde{\mathbf{R}}$ is the conjugate of **R**. The equation $\mathbf{Ra}\tilde{\mathbf{R}}$ works for any dimension, any grade and any objects.

In 3D rotations if a rotor $\mathbf{R_1}$ takes a vector **a** to the vector **b** then **b** is defined as $\mathbf{b} = R_1\mathbf{a}\tilde{R_1}$. If another rotor $\mathbf{R_2}$ takes **b** to **c** then **c** is $R_2\mathbf{b}\tilde{R_2}$. Therefore, $\mathbf{c} = (R_2R_1)\mathbf{a}\tilde{R_2R_1}$. This explains that rotors are expressed in a straightforward manner and in this particular case the final rotor **R** is given by $\mathbf{R} = \mathbf{R_2R_1}$.

Using only the bivectors of the algebra it can be shown that the Hamilton's *Quaternions* are a subset of the geometric algebra. If **i**, **j**, **k** are the elements of the *Quaternions* then these can be defined as $\mathbf{i} = \mathbf{I}e_1$, $\mathbf{j} = -\mathbf{I}e_2$ and $\mathbf{k} = \mathbf{I}e_3$. As $(\mathbf{I}e_1)^2 = -1$, $(-\mathbf{I}e_2)^2 = -1$ and $(\mathbf{I}e_3)^2 = -1$ and $(\mathbf{I}e_1)(-\mathbf{I}e_2)(\mathbf{I}e_3) = \mathbf{I}e_1e_2e_3 = -1$ the famous relations by Hamilton $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$ can be recovered.

If $\mathcal{F} = [a_0, a_1, a_2, a_3]$ is an unit quaternion then the one to one mapping between the quaternion and the rotor which performs the same rotation in geometric algebra is given by eqn (4)

$$\mathbf{R} = \underbrace{a_0}_{\text{scalar}} + \underbrace{a_1\mathbf{I}e_1 - a_2\mathbf{I}e_2 + a_3\mathbf{I}e_3}_{\text{bivector}} \tag{4}$$

Therefore taking only the scalar and bivector parts, a general

rotation in 3-D can be written as:

$$\mathbf{R} = exp(-i\frac{\theta}{2}\mathbf{n}) = \cos\frac{\theta}{2} - i\mathbf{n}\sin\frac{\theta}{2} \tag{5}$$

where θ represents a rotation about an axis parallel to unit vector **n** and the rotation axis **n** is given by $n_1e_2e_3 + n_2e_3e_1 + n_3e_1e_2$ which is spanned by the bivector basis.

4 Rotor Edge Detection

The human eye doesn't process different colors and RGB images separately. Rather the evidence suggests that the processing is similar to a continuous vector valued approach in the 3-D Euclidean space. In this regard the bivector representation of color vectors in geometric algebra fits neatly for the 3-D Euclidean space. Then the color information of $(r, g, b)^T$ vector of the color image $c_{m,n}$ can be written as:

$$c_{m,n} = r_{m,n}e_2e_3 + g_{m,n}e_3e_1 + b_{m,n}e_1e_2. \tag{6}$$

where $r_{m,n}$, $g_{m,n}$ and $b_{m,n}$ are the RGB vectors of the image $c_{m,n}$. The edge detection process involves convolving masks $m_L(x, y)$ (for left) and $m_R(x, y)$ (for right) of the size $(2X+1)$ and $(2Y + 1)$ with the image c(m,n) of dimension $(M \times N)$. In rotor based approach as described above the convolution involves geometric product of the vector with the rotor as shown in the following convolution equation:

$$\hat{c}(m,n) = \sum_{x=-X}^{X}\sum_{y=-Y}^{Y} m_L(x,y)\mathbf{c}(m - x\ mod\ M, n - y\ mod\ N)m_R(x,y) \tag{7}$$

The hypercomplex masks for edge detection of the horizontal, vertical edges were introduced by Sangwine[8] and were extended for rotors by Corrochano-Flores[3]. The rotor **R** convolution works exactly the same way as the hypercomplex convolution and operate on the color vectors (eqn6) of the image. The horizontal left and right masks for the rotor convolution are defined in the following eqn8. The vertical masks are obtained by interchanging the rows and columns of the two masks.

$$m_L(left, hor) = \begin{bmatrix} R & R & R \\ 0 & 0 & 0 \\ \tilde{R} & \tilde{R} & \tilde{R} \end{bmatrix} \tag{8}$$

$$m_R(right, hor) = \begin{bmatrix} \tilde{R} & \tilde{R} & \tilde{R} \\ 0 & 0 & 0 \\ R & R & R \end{bmatrix} \tag{9}$$

The rotors are given by

$$\mathbf{R} = se^{n\pi/4} = s(cos(\pi/4) + \mathbf{n}sin(\pi/4)) \tag{10}$$

where *n* is the unit vector and is given by $\mathbf{n} = (e_2e_3 + e_3e_1 + e_1e_2)/\sqrt{3}$ and $s = 1/\sqrt{6}$ is the scale factor. The left and right masks are applied to each of the color pixels which gives the following convolution in the simplified form,

$$\hat{c}(m,n) = \tilde{\mathbf{R}}(c(m,n))\mathbf{R} = \tilde{\mathbf{R}}(c_{m-1,n-1} + c_{m+1,n} + c_{m-1,n+1})R + R(c_{m+1,n-1} + c_{m+1,n} + c_{m+1,n+1})\tilde{\mathbf{R}} = \tilde{\mathbf{R}}c_u\mathbf{R} + \mathbf{R}c_l\tilde{\mathbf{R}} \tag{11}$$

where $\mathbf{c_u}$ and $\mathbf{c_l}$ are the upper and lower rows of the color subimage (fig 1) comprising of different colors. colors. The

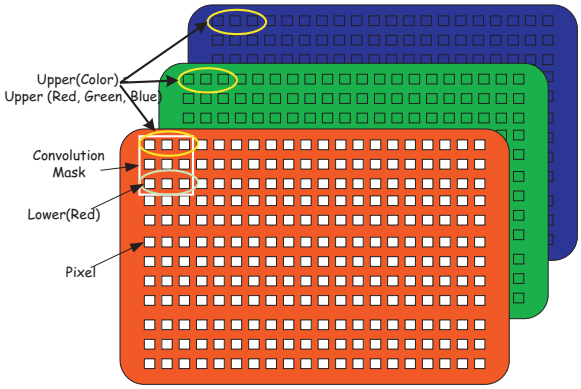


Figure 1: R-G-B plane.

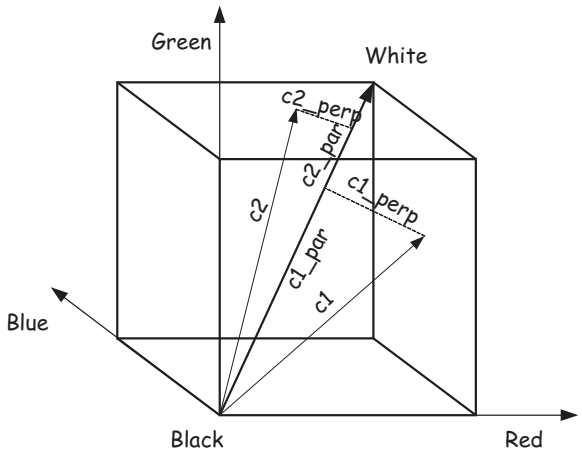


Figure 2: RGB vectors and the Color Cube.

color vector is split into two components. ($c_{\_par}$ or $c_{\parallel}$) is the component parallel to the gray axis and the perpendicular component is ($c_{\_perp}$ or $c_{\perp}$) (fig 2). When the masks are operated on the color vector only the $c_{\perp}$ is affected but the $c_{\parallel}$ is unchanged. After the convolution the perpendicular component is rotated by an amount specified by the rotor **R**. This is quite significant if the colors in the upper and lower rows are homogeneous. The rotor $\mathbf{Ra\tilde{R}}$ would rotate the color vector by the same amount as would the rotor $\mathbf{\tilde{R}aR}$. Hence if the color vectors are homogeneous then both the components would cancel out and the point would fall on somewhere on the gray axis. Then the pixel representing this vector would lie somewhere on gray axis and hence would be

perceived as a gray picture. However if the color components are not homogeneous then the color vector will be rotated by an unequal amount by the two rotors. Thus the resultant vector would lie somewhere in the color cube, far from the gray axis (fig 2). This is evident from the results that we obtained from the three images (see fig 10 12a and 13a) where we applied only the horizontal edge detection masks on the images. More details on this will be discussed in the results section.

Sangwine[8] reported that the Quaternion convolution required the rotations of the 4-D space vectors at an angle $\pi/2$. The same edge detection technique is improved by making the rotation angle dependent on color points with the assumption that the properties of similar and dissimilar regions of the color image remains the same [3]. By doing so the performance improves for the nonhomogeneous regions. This method also suppresses noise caused by shadows. This generalized method based on Sangwine performs the edge detection better than the original Sangwine'smethod.

5 Rotor Edge Detection Hardware

Many image processing algorithms can be decomposed into many parallelized tasks, each task involving operations such as Multiply and Accumulation (MAC). Ideally these systems should also provide the customer with a large variety of processing options to carry out different algorithms. Two major reasons to implement the architecture in hardware is because these algorithms are repetitive in nature and the high degree of regularity of the geometric operations. Hence it presents itself a good candidate for hardware accelerated implementation. Also as shown in Table3 the main computations for the edge detection algorithm is geometric product and additions and multiplications of the vectors which would benefit from such a hardware implementation. With this in mind we developed a new architecture based on the mathematical framework described in the previous section.

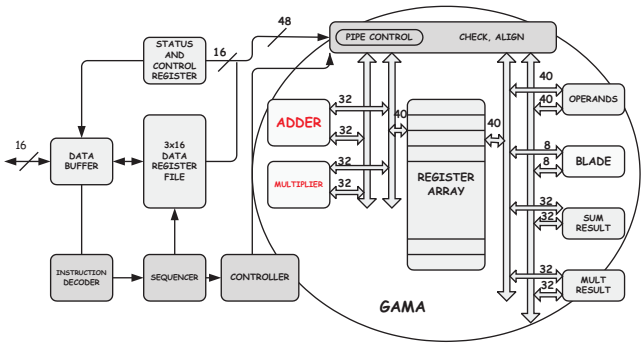


Figure 3: Rotor Edge Detection Hardware Architecture.

The proposed Rotor Edge Detection Hardware architecture consists of an IO interface, control unit, memory unit and a central Geometric Algebra Micro Architecture(GAMA)[2] consisting of adder, multiplier, blade logic, checker and alignment logic, sum and a result register (fig 3). The architecture supports both single and double precision floating point numbers, four rounding modes, and exceptions specified by the IEEE 754 standard [1].The floating point multiplier is a five stage pipeline that produces result on every clock cycle. The shaded portion in the (fig 4) refers to different stages of the pipeline of the floating point multiplier. The floating point adder (fig 5) is more complex than the multiplier. The typical steps for the addition process are: check if any operand to be zero, subtraction of exponents, align the mantissa and add or subtract the two mantissas, adjust the exponent, normalize the result and rounding off the result. The adder is a six stage pipeline (shaded regions of (fig 5)) that produces result on every clock cycle. It is important to state here that our design can process other products of Geometric Algebra with ease. The state machine governing the processing stages of

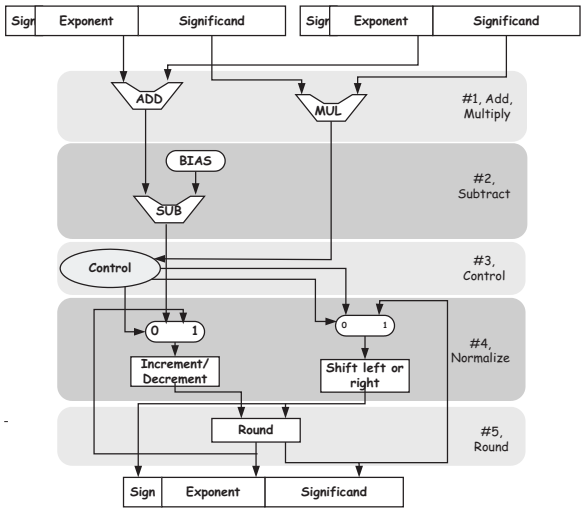


Figure 4: Floating Point Multiplier Architecture.

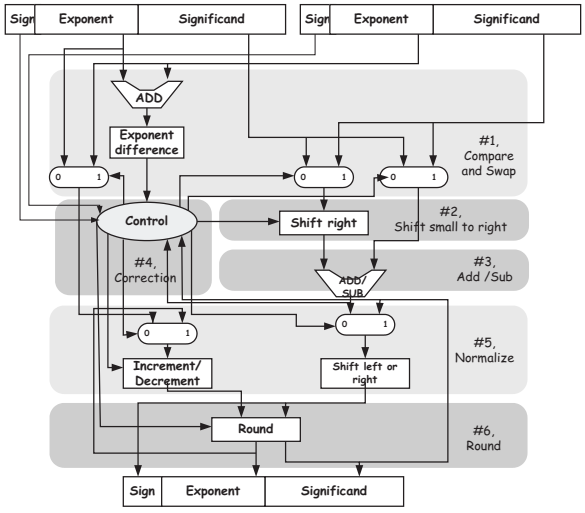


Figure 5: Floating Point Adder Architecture.

Geometric Algebra has six states, idle, clear, load, process,

write and memory dump. Firstly, the idle state waits for the start signal to be high to trigger the state machine. After that, the state machine will come into the clear state where it clears any registers and then to load state to export load as '1' to load input data into the registers. Then it processes the result in required clock cycles based on the control word from the instruction register. Finally, it just drops put the product when the output-enable signal is high in output state. Except the transformation of the load state which is triggered by start signal, the others just proceed to next state after expected number of cycles based on the control word automatically. The long ($\approx$ 320) bit word datapath are coordinated by controller and sequencer unit. The transfer of the data is done in the input and output interface unit. The signals are all registered inputs and outputs to the system. Selection of the data input and output is based on the 16 bit control word. The control bits is used for configuring the processing core for different operations. These control bits are sent by the controller/sequencer which is responsible for defining the data interface and configuring the operators at the correct time and outputting the result to the outside or to the input interface of the core via the IO bus. The control block along with the sequencer ensures effective queueing and stalling to balance the inputs in different stages in the datapath.

Another important element of this architecture is the blade computation. The blade index relationships is already explained for a three dimensional space (eqn1). For example if we want to compute $\mathbf{e}_1$ with $\mathbf{e_2}$, the resultant blade index is $\mathbf{e_1 e_2}$. Similarly if we multiply $\mathbf{e_1 e_2}$ with $\mathbf{e_2}$ then the resultant basis blade index is $\mathbf{e_1}$. This can be implemented by a multiplication table, an approach followed by many software implementation. However accessing a memory in hardware is a slower operation than a simple **EXOR** operation in hardware (fig 6). Determining sign due to blade index is not straightforward due to the invertible nature of the geometric operation. For example the blade index multiplication of $\mathbf{e_1}$ with $\mathbf{e_2}$ gives $\mathbf{e_1 e_2}$ whereas with $\mathbf{e_2}$ and $\mathbf{e_1}$ results in $-\mathbf{e_1 e_2}$. The resulting circuit which is a cascade of **EXOR** gates takes care of the swapping of the blade vector and the **AND** gates compute the number of swaps that the blade element undergoes (see fig 7).

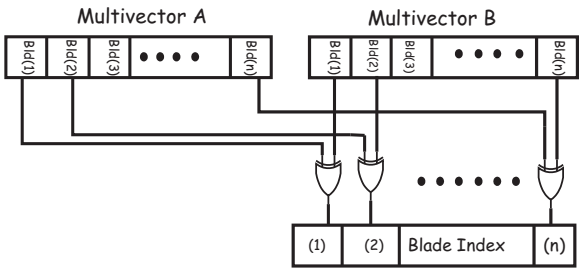


Figure 6: Basis vector computation logic.

The architecture is described using synthesizable VHDL(Very

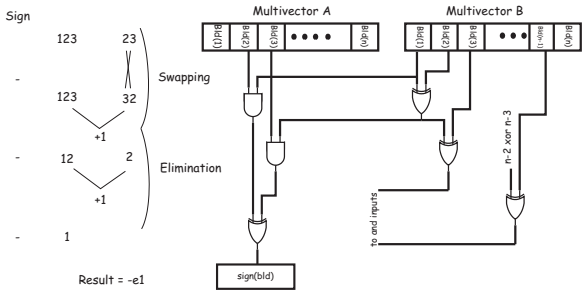


Figure 7: Swapping and sign computation of basis blades.

High Speed Integrated Circuit Hardware Description Language) which makes choosing a different technology not a major decision any more. The EDA tools perform all the translation from VHDL to Silicon. The architecture was synthesized using the ST 0.12 μm standard cell ASIC(Application Specific Integrated Circuit) library with up to 6 layers. The synthesis is done using the synplify ASIC and Cadence Silicon Ensemble tool suite. The synthesis timing and area reports are summarized in the following Table5.

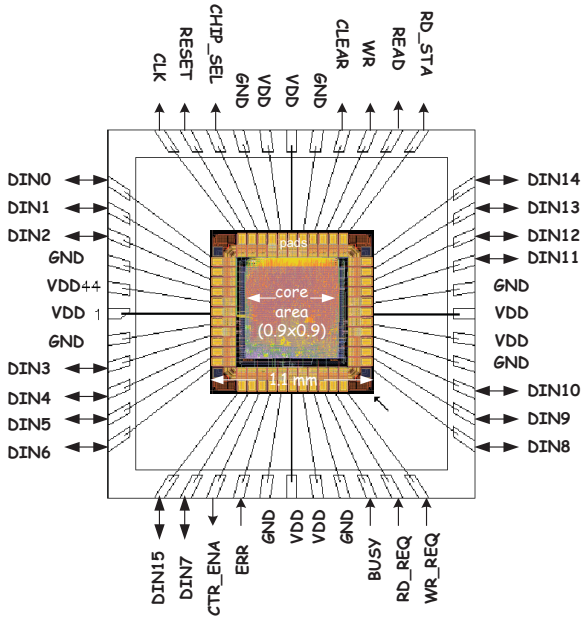


Figure 8: Bonding Diagram of the ASIC.

Design Unit	Rotor Hardware
No of Cells	35355
Area in square microns	813504
No of Equivalent Gates	133361
Clock Frequency	130.0 MHz

Table 2: Area, Clock Frequency of Rotor Hardware .

The core chip area is found to be $0.9mm \times 0.9mm$ and around

$1.1mm \times 1.1mm$ including the IO pads (fig 8). The clock frequency of 130MHz is determined by the longest path in the design.

6 Experimental Results

The experimental setup is as shown in the fig 9. The images were first read in MATLAB and being converted as binaries. These were then fed to the hardware, then the results from the hardware were again converted to the bmp images (see fig 9). Three images (fig 10, 12a and 13a) of different sizes were

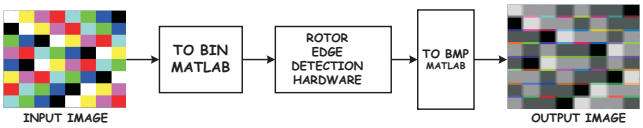


Figure 9: Experimental Setup.

taken and the horizontal masks were applied. The masks are same as the Prewitt operator described in [8] and [3]. Rotor

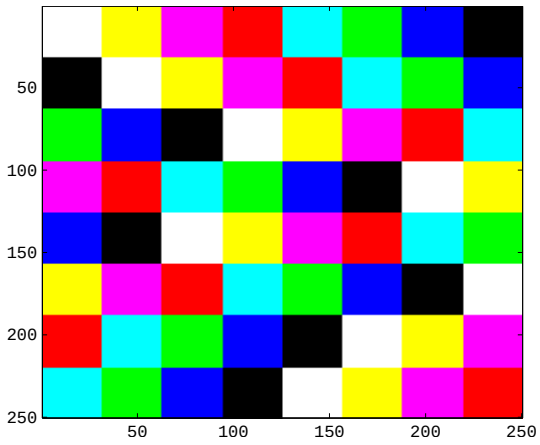


Figure 10: Original Color Block.

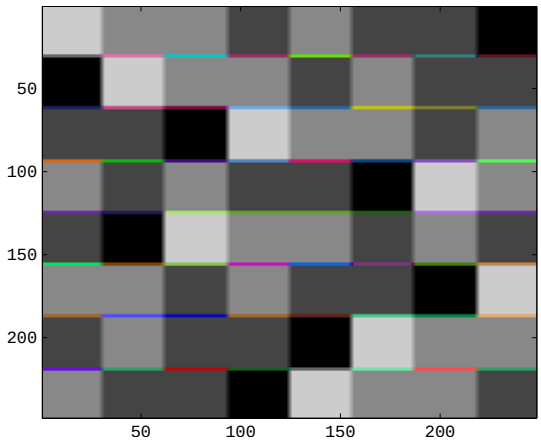
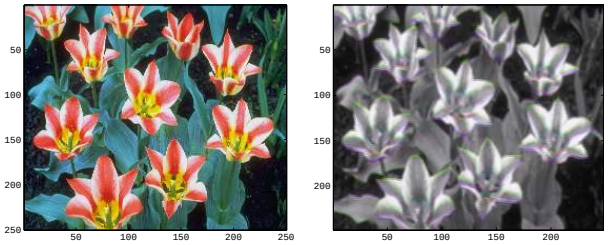


Figure 11: Color Blocks after rotor convolution.

convolution was applied on the test image of the "color blocks" which has an 8×8 array of colored squares (see fig 10). The result of the filtered image is shown in the fig 11. The filtered image has gray areas where the squares had uniform color. But it has colored lines at the edges or where there was a change of color. Also it can be observed that the edges between the black and white blocks remain gray but the edges change to color where there was a color difference across the edge. This is because the $\tilde{\mathbf{R}}\mathbf{c}\mathbf{R}$ rotates the color by $\pi/2$ and $\mathbf{R}\mathbf{c}\tilde{\mathbf{R}}$ by $-\pi/2$. Hence the two color vectors cancel each other due to the rotation operation when they are uniform and fall on the Black and White axis. Otherwise they fall outside the line giving a color to the pixel. This signifies the rotor operation is a shift in hue of the image. The areas where the upper and lower pixels are similar the rotors produce a gray scale image. When these pixels differ in color the rotors produce different colors as they don't cancel in the chromatic sense. Hence the change of direction due to rotation of colors results in different colors on the edges. This type of change can also be observed on the filtered images of tulips (fig 12b) and lenna (fig 13b). The areas where the color change was flat or smoothly varying, the filtered image became chromatic, otherwise colors can be observed in regions like the top edges of the hat of the "lenna" image and the edges of flowers of the "tulip" image.



(a) Tulip Image. (b) Tulip Image after rotor convolution

Figure 12: Outputs of the Tulip Image before(a) and after(b) rotor convolution.

(a) Original Lenna Image. (b) Lenna Image after rotor convolution

Figure 13: Outputs of the Lenna Image before(a) and after(b) rotor convolution.

It can be seen that for an image size of 128×128 pixels the total number of geometric product multiplications adds up to 196608(Table 3, 2^{nd} col). This is because each color pixel is treated as a vector and each convolution operation consists of 12 geometric product multiplications and 4 geometric product additions. However, if we use the linearity of the

rotor operations we can use 4 additions and then perform 4 convolution on these vectors. Furthermore each product in $\mathbf{R}a\tilde{\mathbf{R}}$ takes 28 floating point multiplications and 26 floating point additions. Similar number of operations are needed for the $\tilde{\mathbf{R}}a\mathbf{R}$ computation. Hence the total number of convolution operations equals 56 geometric product multiplications and 52 geometric product additions. However three more addition are required to calculate ($\mathbf{R}a\tilde{\mathbf{R}}$+$\tilde{\mathbf{R}}a\mathbf{R}$) totalling the add operations to 55. Since these products are calculated based on the basis vectors the total number of floating point multiplications and additions result in 917504(Table 3, 4^{th}col) and 901120 respectively (see Table 3, 5^{th}col) and for the 128×128 image. The hardware is designed to handle such specialized computations efficiently. It can be seen that the total time taken for the convolution of a 128×128 image takes 5701781 cycles which amounts to 46 ms (Table 3, 6^{th}col) based on the design running at 130 MHz. The convolution times for different sized images are obtained and is given in the following Table (3).

Image size	GP Mul	GP Add	Float Mul	Float Add	Time (ms)
128×128	196608	65536	917504	901120	46
256×256	786432	262144	3670016	3604480	184
512×512	3145728	1048576	14680064	14417920	735

Table 3: Rotor convolution operation time on hardware for different image sizes on hardware

7 Conclusion

Like the human visual system which does not process the R, G, B color channels separately, this approach fits nicely with the bivector representation for the color vectors. Experiments show that this kind of edge detection is holistic in nature. It is also concluded that the convolution operation with the rotor masks belong to a class of linear vector filters. This linear vector filter can be applied to image or speech signals where vector filtering is of fundamental interest[7].

The paper presented an overview of the convolution operations involving rotors for image processing application. A new rotor hardware was introduced, including its potential for other applications in Vision and Graphics. The hardware architecture is discussed briefly. The architecture is tailored for image processing applications, providing an acceptable application performance requirements. The usefulness of the introduced approach was demonstrated by analyzing and implementing a computation intensive edge detection algorithm on this hardware. Much of the future work will be focussed on theoretical understanding of this linear filter and the frequency response which would be useful in developing further algorithms.

Acknowledgements

BM would like to thank the School of Electronics and Computer Science, University of Southampton for their Support in the form of a University Research Scholarship.

References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[2] B.Mishra and P.Wilson. Hardware implementation of a geometric algebra processor core. In *Proceedings of ACA 2005*. IMACS, Int. Conference on Advancement of Computer Algebra, Nara, Japan, 2005.

[3] E.Bayro-Corrochano and S.Flores. Color edge detection using rotors. In Leo Dorst, Chris Doran, and J. Lasenby, editors, *Applications of Geometric Algebras in Computer Science and Engineering*. Birkhäuser, 2002.

[4] J. Lasenby, W. J. Fitzgerald, A. N. Lasenby, and C. J. L. Doran. New geometric methods for computer vision: An application to structure and motion estimation. *Int. J. Comput. Vision*, 26(3):191–213, 1998.

[5] L.Dorst and S.Mann. Geometric algebra: A computational framework for geometrical applications (1). *IEEE Computer Graphics and Applications*, 22(3):24–31, 2002.

[6] Stephen Mann and Leo Dorst. Geometric algebra: A computational framework for geometrical applications (2). *IEEE Comput. Graph. Appl.*, 22(4):58–67, 2002.

[7] R.Lukac, B.Smolka, K.Martin, K.N.Plataniotis, and A.N.Venetsanopoulos. Vector filtering for colour imaging. *IEEE Signal Processing Magazine*, 22(1):74–86, 2005.

[8] S.J.Sangwine. Colour image edge detector based on quaternion convolution. *IEE Electronics Letters*, 34(10):969–971, 1998.

[9] S.J.Sangwine and T.A.Ell. Colour image filters based on hypercomplex convolution. *IEE Proc Vision Image Signal Processing*, 147(2), 2000.

[10] H.Joel Trussel, Eli Saber, and Michael Vrehl. Colour image processing. *IEEE Signal Processing Magazine*, 22(1):14–22, 2005.