Functional Programming of Geometric Algebra and its Ray-Tracing Application

Charneau S., Aveneau L., Fuchs L.
SIC, CNRS FRE 2731, SP2MI
Bd Marie et Pierre Curie, BP 30179
86962 Futuroscope Chasseneuil Cedex – France
{charneau,aveneau,fuchs}@sic.univ-poitiers.fr

ABSTRACT

In computer graphics, geometric algebra provides a formal way to do complex geometric calculations. We propose an implementation of the geometric algebra by using a functional programming language. To emphasize its efficiency, we compare it with a well known geometric algebra implementation in an application to a ray tracer.

Keywords
geometric algebra, geometric algorithms, functional programming, imperative programming

1. INTRODUCTION

Geometric Algebra initially comes from Clifford Algebra. It has been studied to develop a mathematical system designed for a universal geometric calculus [Hes86]. The possibilities of this system are large. First, it embeds some schemes like vector algebra and quaternions in a unique system. Then it permits to represent geometric concepts by symbolic terms. Finally, specifications of operations are coordinate free, and can be easily generalised in all dimensions.

The use of the geometric algebra in computer science appears very promising, but its implementation causes some problems [PHF04], which require some cares about the data structures definition.

We present here an implementation of geometric algebra with the functional programming language Objective Caml ; we then show the results of its comparison to Gaigen [BFD03], the most efficient geometric algebra implementation that exists [PHF04].

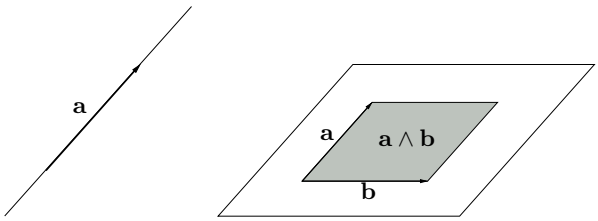
2. GEOMETRIC ALGEBRA

A geometric algebra is an associative algebra generated by a real vector space, in which a vector squares to a scalar [Hes86]. Given four vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$, we can define a geometric product by :

$$\begin{aligned} \mathbf{a}(\mathbf{bc}) &= (\mathbf{ab})\mathbf{c} \\ (\mathbf{a}+\mathbf{b})(\mathbf{c}+\mathbf{d}) &= \mathbf{ac}+\mathbf{ad}+\mathbf{bc}+\mathbf{bd} \\ \mathbf{a}^2 &= \epsilon_{\mathbf{a}}|\mathbf{a}|^2 \end{aligned}$$

where $|\mathbf{a}|$ is the magnitude of $\mathbf{a}$ and $\epsilon_{\mathbf{a}} \in \{-1, 0, 1\}$ is called the signature of $\mathbf{a}$.

It is convenient to decompose this product into its symmetric and antisymmetric parts by $\mathbf{ab} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathbf{b}$, where $\mathbf{a} \cdot \mathbf{b}$ is a scalar and $\mathbf{a} \wedge \mathbf{b}$ is a new entity called a bivector, or 2-vector, geometrically interpreted by an oriented plane segment in the same way that a vector is an oriented line segment (see figure 1).



(a) vector, gives a unique line

(b) bivector, gives a unique plane

Figure 1: Interpretation of k-vectors

With a n dimensional euclidean space $\mathcal{E}^n$, the geometric product leads to a 2^n dimensional linear space $\mathcal{G}_n$ of multivectors constructed by a direct sum of $n+1$ linear spaces $\mathcal{G}_n^k$ of k-vectors, $k \in [0; n]$, where k is called the *grade*. 0-vectors are scalars and 1-vectors are isomorph

WSCG 2005 Posters proceedings, ISBN 80-903100-8-7
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.

to vectors of $\mathcal{E}^n$. A k-vector $\mathbf{v}_1 \wedge \mathbf{v}_2 \cdots \wedge \mathbf{v}_k$ is geometrically interpreted by an oriented segment of the k-subspace of $\mathcal{E}^n$ generated by the k vectors $\mathbf{v}_1, \ldots \mathbf{v}_k$.

3. A FUNCTIONAL GEOMETRIC ALGEBRA IMPLEMENTATION

For our implementation, we use the functional language Objective Caml [INRIA], and we include optimizations equivalent to those found in the Gaigen implementations.
Gaigen [BFD03] is a C++ geometric algebra implementation generator. It includes some optimisations on the data structures and the operations, that consist in representing a multivector only by its non-null grade parts and in making specific treatments on them. However, doing that with an imperative language makes the data structures and their handling complex.

Interests for a functional language

The interests for a functional language are due to its links with mathematical and formal models which make the programming more efficient. Likewise, these languages were developed to handle formal expressions, which is precisely what we do in geometric algebra. Moreover Objective Caml allows *pattern matching* oriented programming, a kind of term unification system which highly facilitates term handling.

Our Implementation

We have implemented two kinds of O'Caml modules. The first one is a module that represents the geometric algebra of a given euclidean space. On this pattern, three modules were developed, for three algebras frequently used for 3 dimensional euclidean geometry. These modules integrate the same optimisations on the represented data as the Gaigen ones.
For the second implementation, we have parameterized the module by a space (the latter being specified in an other module), thanks to which every algebra can be "generated" and used, whatever the space used.

Results and comparisons

Our O'Caml implementation represent several advantages over the Gaigen libraries. First, from the point of view of the programming :

- the code is shorter and more readable,
- the data structure is simpler to define and to handle,
- we do not need intermediate data structures to control the content of a multivector, contrary to Gaigen,
- the realization of a generic module is made easier.

All these advantages simplify the development which becomes consequently faster.
Then, from the efficiency point of view, we have compared the different implementations for the calculation of rays intersections, in a home made C ray tracer.

Figure 2 shows the results we have obtained on a scene that contains about $20\ 000$ triangles, for a 640×480 pixels image size and 4 rays per pixels. The times indicated only take into account the time spent in the different geometric algebra libraries. The difference between the remained times is due to the interface between O'Caml and C. Then one can see that we are more than 2.5 times faster than the Gaigen libraries.

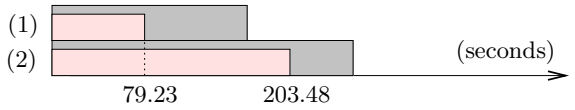


Figure 2: Rendering times for each implementation, (1) : non generic O'Caml module (2) : Gaigen optimized library

4. CONCLUSION

We have demonstrated that functional languages are more adapted to implement geometric algebra than imperative languages. Thereby, the idea of an implementation generator as Gaigen remains interesting. Our generic parameterized module is a first state for such a goal. Indeed, based on a pseudo euclidean space, a non generic optimized module should be generated simply by integrating into it the result of each product for every combinaison of k-vectors in terms of product and sum of coordinates. The determination of such results can easily be done with the generic parameterized module. Moreover, by taking more care about the data structure and its handling, and less care about the generated code (length and redundancy), we would be able to generate libraries even more optimized.
Another possible use of functional languages consists in, for a given operation on multivectors, reducing the induced term by *lazy evaluation*. For this reduction of terms, functional languages again appear particularly handy. This should minimize the calculations on coordinates as well as the errors on these floating points calculations and the time to perform them.

5. REFERENCES

[BFD03] T. Bouma D. Fontijne, L. Dorst. Gaigen. 2003, University of Amsterdam. http://www.science.uva.nl/ga/gaigen/index.html.

[Hes86] D. Hestenes. A unified language for mathematics and physics. *Clifford Algebras and their Applications in Mathematical Physics*, chapter 1, pages 1–23. Kluwer Academic Publishers, 1986.

[INRIA] Institut National de Recherche en Informatique et en Automatique. The O'Caml language. http://www.ocaml.org/.

[PHF04] C. Perwasss D. Hildenbrand, D. Fontijne and L. Dorst. Geometric algebra and its application to computer graphics. Tutorial 3, 25^{th} Annual Conference of the EACG, September 2004.