Algorithmic structure for geometric algebra operators and application to quadric surfaces

Stéphane Breuils

UNIVERSITÉ PARIS-EST

ÉCOLE DOCTORALE MSTIC

THÈSE DE DOCTORAT EN INFORMATIQUE

Structure algorithmique pour les opérateurs d'Algèbres Géométriques et application aux surfaces quadriques

---

Auteur:

Stéphane Breuils

Encadrants:

Dr. Vincent NOZICK

Dr. Laurent FUCHS

Rapporteurs:

Prof. Dominique MICHELUCCI

Prof. Pascal SCHRECK

Examinateurs:

Dr. Leo DORST

Prof. Joan LASENBY

Prof. Raphaëlle CHAINE

Soutenue le 17 Décembre 2018

Thèse effectuée au Laboratoire d'Informatique Gaspard-Monge, équipe A3SI,
dans les locaux d'ESIEE Paris

LIGM, UMR 8049
Cité Descartes,
Bâtiment Copernic-5, bd Descartes
Champs-sur-Marne
77 454 Marne-la-Vallée Cedex 2

École doctorale Paris-Est
Cité Descartes,
6-8 av. Blaise Pascal
Champs-sur-Marne,
77 455 Marne-la-Vallée Cedex 2

Abstract

Geometric Algebra is considered as a very intuitive tool to deal with geometric problems and it appears to be increasingly efficient and useful to deal with computer graphics problems. The Conformal Geometric Algebra includes circles, spheres, planes and lines as algebraic objects, and intersections between these objects are also algebraic objects. More complex objects such as conics, quadric surfaces can also be expressed and be manipulated using an extension of the conformal Geometric Algebra. However due to the high dimension of their representations in Geometric Algebra, implementations of Geometric Algebra that are currently available do not allow efficient realizations of these objects.

In this thesis, we first present a Geometric Algebra implementation dedicated for both low and high dimensions. The proposed method is a hybrid solution that includes precomputed code with fast execution for low dimensional vector space, which is somehow equivalent to the state-of-the-art method. For high dimensional vector spaces, we propose runtime computations with low memory requirement. For these high dimensional vector spaces, we introduce new recursive scheme, and we prove that associated algorithms are efficient both in terms of computational and memory complexity. Furthermore, some rules are defined to select the most appropriate choice, according to the dimension of algebra and the type of multivectors involved in the product. We will show that the resulting implementation is well suited for high dimensional spaces (e.g. algebra of dimension 15) as well as for lower dimensional spaces.

The next part presents an efficient representation of quadric surfaces using Geometric Algebra. We define a novel Geometric Algebra framework, the Geometric Algebra of $\mathbb{R}^{9,6}$ to deal with quadric surfaces where an arbitrary quadric surface is constructed by merely the outer product of nine points. We show that the proposed framework enables us not only to intuitively represent quadric surfaces but also to construct objects using Conformal Geometric Algebra. In the proposed framework, the computation of intersection of quadric surfaces, the normal vector, and the tangent plane of a quadric surface are provided. Finally, a computational framework of the quadric surfaces will be presented with the main operations required in computer graphics.

Keywords: Geometric Algebra, Algorithmic structure, Trees, Quadric surfaces

Résumé

L'algèbre géométrique est un outil permettant de représenter et manipuler les objets géométriques de manière générique, efficace et intuitive. A titre d'exemple, l'Algèbre Géométrique Conforme (*CGA*), permet de représenter des cercles, des sphères, des plans et des droites comme des objets algébriques. Les intersections entre ces objets sont inclus dans la même algèbre. Il est possible d'exprimer et de traiter des objets géométriques plus complexes comme des coniques, des surfaces quadriques en utilisant une extension de *CGA*. Cependant due à leur représentation requérant un espace vectoriel de haute dimension, les implantations de l'algèbre géométrique, actuellement disponible, n'autorisent pas une utilisation efficace de ces objets.

Dans ce manuscrit, nous présentons tout d'abord une implantation de l'algèbre géométrique dédiée aux espaces vectoriels aussi bien basses que hautes dimensions. L'approche suivie est basée sur une solution hybride de code pré-calculé en vue d'une exécution rapide pour des espaces vectoriels de basses dimensions, ce qui est similaire aux approches de l'état de l'art. Pour des espaces vectoriels de haute dimension, nous proposons des méthodes de calculs ne nécessitant que peu de mémoire. Pour ces espaces, nous introduisons un formalisme récursif et prouvons que les algorithmes associés sont efficaces en terme de complexité de calcul et complexité de mémoire. Par ailleurs, des règles sont définies pour sélectionner la méthode la plus appropriée. Ces règles sont basées sur la dimension de l'espace vectoriel considéré. Nous montrons que l'implantation obtenue est bien adaptée pour les espaces vectoriels de hautes dimensions (espace vectoriel de dimension 15) et ceux de basses dimensions.

La dernière partie est dédiée à une représentation efficace des surfaces quadriques en utilisant l'algèbre géométrique. Nous étudions un nouveau modèle en algèbre géométrique de l'espace vectoriel $\mathbb{R}^{9,6}$ pour manipuler les surfaces quadriques. Dans ce modèle, une surface quadrique est construite par l'intermédiaire de neuf points. Nous montrerons que ce modèle permet non seulement de représenter de manière intuitive des surfaces quadriques mais aussi de construire des objets en utilisant les définitions de *CGA*. Nous présentons le calcul de l'intersection de surfaces quadriques, du vecteur normal, du plan tangent à une surface en un point de cette surface. Enfin, un modèle complet de traitement des surfaces quadriques est détaillé.

Mots clés: Algèbre Géométrique, Structure algorithmique, Arbres, Surfaces quadriques

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisors Vincent Nozick and Laurent Fuchs for the continuous support of my Ph.D study and research, for their patience and motivation. I am very grateful to Vincent for his guidance and great help in all the time of research and writing of this thesis and that both in France and Japan. I am also very thankful to Leo Dorst for the very insightful meeting that we had in France.

I would like to thank my thesis committee: Raphaëlle Chaine, Dominique Michelucci, Pascal Schreck, Leo Dorst, Joan Lasenby for their encouragement, comments, reviews, and very interesting questions.

I am also very grateful to Akihiro Sugimoto for hosting me at the National Institute of Informatics and for the very helpful weekly meetings that we had during my stay in Japan. I wanted also thank Yukiko Kenmochi for her help and advices during this thesis.

I am grateful to the support received through the collaborative work with Eckhard Hitzer at ICU in Tokyo. I would like to express my appreciation to my co-authors, including Dietmar Hildenbrand, Werner Benger and Christian Steinmetz. I am also thankful to the Japan Society for the Promotion of Science (JSPS) for supporting me in Tokyo. In this context, I thank my fellow labmates at the JFLI and NII in Tokyo: Marc Bruyère, Fumiki Sekiya, Benjamin Renoust, Thomas Silverston, Megumi Kaneko, Diego Thomas and Jean-François Baffier with whom I would be delighted to work with.

I thank also my fellow PhD candidates at ESIEE Paris: Clara Jaquet, Julie Robic, Thibault Julliand, Bruno Jartoux, Odyssée Merveille, Elodie Puybareau, Diane Genest, Deise Santana Maia, Eloise Grossiord, Karla Otiniano, Ketan Bacchuwar, Thanh Xuan Nguyen.
Furthermore, I wanted to thank Bruno Jartoux with whom I could share the preparation of a complete lecture at IMAC.

I have special friends to thank. Going back to my ESIEE Paris days, I want to thank Gilles, Maxime, Amaury and Florian. In Japan, I want also to thank Asa for the very interesting discussions that we had at the end of this thesis.

Last but not the least, I would like to warmly thank my family for their continuous and unparalleled love, thank you for your great help throughout the time of my research. Thank you for encouraging and inspiring me in all of my pursuits. Finally, this thesis is dedicated to my brother Guillaume Breuils.

Contents

Abstract v

Résumé vii

Acknowledgements ix

List of Figures xv

List of Tables xvii

Introduction 1

Introduction (in French) 5

1 Construction of Geometric Algebra 11
1.1 The battle of Vector Algebra . . . 12
1.2 Vectors of Geometric Algebra . . . 13
1.3 First major product of Geometric Algebra . . . 14
1.3.1 2-blades . . . 15
1.3.2 3-blades . . . 16
1.3.3 Higher dimensional subspaces . . . 17
1.4 Inner product . . . 19
1.4.1 Metric vector space . . . 19
1.4.2 Between blades of different grade . . . 20
1.4.3 Reverse and squared norm of blades . . . 21
1.5 Geometric product and Geometric Algebra . . . 22
1.6 Complex numbers and Hamilton algebra included . . . 23
1.7 Pauli matrices included . . . 24
1.8 Inverse and dual . . . 26
1.8.1 Intersections . . . 27
1.9 Products from geometric product . . . 27
1.10 Transformations . . . 29
1.10.1 Reflections and rotations . . . 29
1.10.2 Versors . . . 30
1.11 Conformal Geometric Algebra . . . 31
1.11.1 basis and metric . . . 31
1.11.2 Point of CGA . . . 32
1.11.3 CGA objects . . . 32
1.11.4 Intersection in CGA . . . 34
1.11.5 Transformations . . . 35
1.11.6 Properties of CGA objects . . . 36

I Implementation of Geometric Algebra 37

2 Design 39
2.1 Introduction . . . 39
2.1.1 State of the art . . . 39
2.1.1.1 Code generators . . . 40
2.1.1.2 Library generators . . . 41

2.1.2 Expression of needs . . . 41
2.1.3 Library generator . . . 42
2.1.3.1 Efficient . . . 42
2.1.3.2 User friendly . . . 43
2.2 What to store . . . 43
2.2.1 Requirements . . . 43
2.3 Multivector and linked list . . . 44
2.4 Multivector and arrays . . . 44
2.5 Per-grade data structure . . . 44
2.6 What to compute? . . . 46
2.7 Naive methods to compute the products . . . 47
2.7.1 Inside the products . . . 47
2.7.2 Table based methods . . . 47
2.7.3 Function based methods . . . 48
2.7.4 Complexity issue . . . 49

3 Products in Low dimensional space **51**
3.1 Per grade products . . . 51
3.2 SIMD instructions . . . 52
3.3 Dual computation . . . 53

4 Products in high dimensional space **55**
4.1 Recursive scheme . . . 55
4.1.1 Definitions and Notations . . . 55
4.1.2 Recursive form of Geometric Algebra . . . 56
4.1.2.1 Binary Trees and Multivectors . . . 56
4.1.2.2 Outer product . . . 57
4.2 Geometric Algebra operators as a recursive construction of lists . . . 58
4.2.1 Outer product as a recursive construction of lists . . . 58
4.2.2 Binary trees labelling . . . 58
4.2.2.1 From trees to lists . . . 59
4.2.2.2 Construction of *Alist* . . . 61
4.2.3 Complexity of this method . . . 64
4.2.4 Sign computation . . . 64
4.2.5 Geometric product as a recursive construction of lists . . . 66
4.2.5.1 Euclidean space . . . 66
4.2.5.2 Iterative construction of *Alist* and *Blist* . . . 67
4.2.5.3 Non-Euclidean space . . . 70
4.2.6 Discussions . . . 71
4.3 Automata speed up . . . 72
4.3.1 Recursive method revisited . . . 73
4.3.2 Discussion . . . 74
4.4 Prefix tree approach . . . 75
4.4.1 Multivectors and prefix trees . . . 75
4.4.2 Anti-commutativity recursive operator . . . 85
4.4.2.1 Recursive formula . . . 85
4.4.3 Outer product . . . 87
4.4.3.1 Recursive formula . . . 87
4.4.4 Complexity . . . 89
4.4.5 GA products using metric . . . 90
4.4.5.1 Left, right contractions and inner product . . . 90
4.4.5.2 Geometric product . . . 91
4.4.6 Complexity . . . 93
4.4.7 Dual and prefix tree . . . 95
4.4.8 Products with dual multivectors . . . 96
4.4.9 Discussion . . . 97
4.4.10 Hybridization . . . 98
4.5 Non orthogonal metric . . . 98

4.5.1 Numerical robustness preprocessing . . . 99
4.5.2 Computing transformation matrices . . . 99
4.5.2.1 Algorithm . . . 100
4.5.2.2 Data structure to use . . . 101

5 Garamon **103**
5.1 Resulting library . . . 103
5.2 Performances . . . 105
5.2.1 High dimensions . . . 106
5.2.2 Speed computation . . . 107
5.2.3 Memory consumption . . . 107
5.3 Partial conclusion . . . 108

II Geometric algebra and quadrics **109**

6 Introduction **113**
6.1 Contributions . . . 114

7 Models of quadric surfaces with Geometric Algebra **115**
7.1 Complexity estimation model . . . 116
7.2 DCGA of $\mathbb{G}_{8,2}$ . . . 116
7.3 DPGA of $\mathbb{G}_{4,4}$ . . . 122

8 Quadric conformal Geometric Algebra **127**
8.1 QCGA definition . . . 127
8.1.1 QCGA basis and metric . . . 127
8.1.2 Point in QCGA . . . 129
8.1.3 Normalization . . . 130
8.1.4 Embedded of CGA points . . . 131
8.2 Round objects of QCGA . . . 132
8.2.1 Sphere . . . 132
8.2.1.1 Primal representation of a sphere . . . 132
8.2.1.2 Dual representation of a sphere . . . 134
8.3 Flat objects . . . 135
8.3.1 Plane . . . 135
8.3.1.1 Primal representation of a plane . . . 135
8.3.1.2 Dual representation of a plane . . . 136
8.3.2 Line . . . 137
8.4 Quadric surfaces . . . 138
8.4.1 Primal representation of a quadric surface . . . 139
8.4.2 Representation of a primal axis-aligned quadric surface . . . 140
8.4.3 Representation of a dual quadric surface . . . 144
8.4.4 Some examples . . . 145
8.5 Normals and tangents . . . 149
8.6 Intersections . . . 151
8.6.1 Quadric-Line intersection . . . 151
8.7 Transformations . . . 152
8.7.1 Translation . . . 152
8.7.2 Other transformations . . . 154
8.8 Discussion . . . 155
8.8.1 Limitations . . . 155
8.8.2 Implementations . . . 156
8.8.3 Complexity of some major operations of QCGA . . . 157

9 Mapping **159**
9.1 DCGA reciprocal operators 159
9.2 DPGA reciprocal operators 160
9.3 QCGA reciprocal operators 161
9.4 Test 161
9.5 Potential computational framework 162

A Garamon sample **169**

References **175**

List of Figures

1.1 Some of the main contributors to the vector algebras . . . 13
1.2 Geometric meaning of the outer product between two vectors. . . . 14
1.3 Geometric meaning of the outer product between three vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$ of $\mathbb{R}^3$. . . . 16
1.4 Relation between k-vectors and k-blades . . . 17
1.5 Reflection to rotation (from gaviewer [35]) . . . 30
1.6 definition of some geometric primitives from control points . . . 33
1.7 Two intersection are computed, one between the circle and the sphere and the other between the sphere and the plane. The intersections exist in (A) but do not in (B). But in both cases, this results in a Geometric Algebra entity, and we can interpret the results. . . . 35

2.1 Data structure, example with $\mathbf{x} = 3\mathbf{e}_2 + 2\mathbf{e}_{14}$ in a 4-dimensional vector space. The column on the left denotes the grade . . . 45
2.2 Data structure, example with the initialization of a sphere of CGA (5-dimensional vector space). This sphere has a radius of 2 and its center point at coordinates $(1, 2, 3)$. Note that there are no useless zeros stored . . . 45

4.1 Binary tree of $\mathfrak{a}$, representing a multivector in a 2^3-dimensional space. There is a mapping between the coefficients a_i of the multivector and the leaves of the tree. . . . 57
4.2 Labelling of a binary tree $\mathfrak{c}$ in a 3-dimensional space. . . . 59
4.3 Recursive definition of *Alist* pushed down to the leaves. . . . 60
4.4 *Alist* development in a 3—dimensional space. . . . 68
4.5 *Blist* development in a 3-dimensional space. . . . 68
4.6 Construction of *Rlist* and *Alist* for the node 101. . . . 70
4.7 Automaton of the set of products of the outer product from an recursion level to the next. . . . 73
4.8 Prefix tree data structure. Each node of the tree is labelled by a binary index representing a basis vector of the algebra, here in a 3-dimensional vector space. . . . 76
4.9 The labelling of the prefix tree at one depth . . . 77
4.10 Multivector prefix tree representation. . . . 77
4.11 Tree structure for some resulting multivectors of grade 4 (A), grade 3 (B), grade 2 (C), grade 1 (D) in a 4-dimensional vector space. Useless branches are depicted in green dashed arrows above the targeted multivector and in blue below. The targeted nodes are surrounded by a black rectangle. We can remark that over 15 theoretic traversals, 11 useless traversals are ignored in (A), 6 useless traversals are ignored in (B), 6 useless traversals are ignored in (C) and 11 useless traversals are ignored in (D). . . . 82
4.12 Primal form of a tree data structure of an Euclidean 3 dimensional vector space, and its dual counterpart in red . . . 95
4.13 The Data structure shown in Section 2.2.1, this structure includes the coefficients to traverse any dual multivector. . . . 97

5.1 Garamon library directory tree . . . 106

8.1 Result of one paraboloid from six points . . . 142
8.2 Result of one hyperbolic paraboloid from six points . . . 142
8.3 Construction of three cylinders along (Ox, Oy, Oz). Each cylinder is constructed from 5 points and has the same diameter . . . 143
8.4 Construction of one elliptic cylinder from five points . . . 144

8.5 Construction of an axis-aligned prolate spheroid from 5 points 145
8.6 Construction of an axis-aligned ellipsoid. 145
8.7 Construction of a generalized cylinder from nine points. 146
8.8 Construction of a dual hyperboloid of one sheet. 147
8.9 Construction of a axis-aligned pair of planes. 147
8.10 Example of our construction of QCGA objects. From left to right: a dual hyperboloid built from its equation, an ellipsoid built from its control points (in yellow), the intersection between two cylinders, and a hyperboloid with a sphere (the last one was computed with our ray-tracer). 156

List of Tables

1.1 Structure of a general entity of Geometric Algebra 18
1.2 Inner product between QCGA basis vectors. 31
1.3 Transformations handled in CGA 35

4.1 Outer product table in a 3-dimensional space. 59
4.2 *Alist* computed at the node c_{101}. 62
4.3 Geometric product table in a 3-dimensional space. 67

5.1 Memory requirement (in MB) to store 5.10^4 random vectors. 107
5.2 Memory requirement (in MB) to store 5.10^4 random bivectors. 108

7.1 Inner product between DCGA basis vectors. 117
7.2 Computational features in number of Geometric Algebra operations for DCGA . . 122
7.3 Inner product between DPGA basis vectors. 122
7.4 Computational features in number of Geometric Algebra operations for DPGA . . 125

8.1 Inner product between QCGA basis vectors. The dots denote some inner products that result to 0. 128
8.2 Definition of dual CGA objects using QCGA. 148
8.3 Definition of dual **axis-aligned** quadric surfaces using QCGA. 148
8.4 Definition of primal geometric objects using QCGA. 148
8.5 Definition of some primal **axis-aligned** quadric surfaces using QCGA. 149
8.6 Comparison of properties between DPGA, DCGA, and QCGA. The symbol • stands for "capable", ∘ for "incapable" and ⊘ for "unknown". 155
8.7 Comparison of the computational features between QCGA, DCGA, DPGA in number of Geometric Algebra operations 158

Introduction

Geometric Algebra provides useful and, more importantly, intuitively understandable tools to represent, construct and manipulate geometric objects. Intensively explored by physicists, Geometric Algebra has been applied in classical mechanics, quantum mechanics and electromagnetism [19, 44, 48, 58, 40, 58, 79, 53]. Geometric Algebra has also found some interesting applications in data manipulation for Geographic Information Systems (GIS), see [63, 86, 84, 10]. Furthermore, it turns out that Geometric Algebra can be applied even in computer graphics, either to basic geometric primitive manipulations [83, 57, 51, 75, 50, 81, 47] or to more complex illumination processes as in [69] where spherical harmonics are substituted by Geometric Algebra entities. Other works [27] and [25] show that Geometric Algebra can be used to express and transform some more complex objects such as conics and quadrics. These works on computer graphics can sometime be extended to some applications in virtual reality for object collision detection [75]. Even if Geometric Algebra literature lacks of least square estimations, it can also still find application in computer vision, particularly in the representation of camera model [80, 55]. Geometric Algebra can also find some more unexpected applications such as in deep learning with "Clifford neurons" [9, 52, 62].

There already exist numerous Geometric Algebra implementations, and most of the programming languages or famous mathematical frameworks can find a Geometric Algebra library well suited for a comfortable use. However, very few of these libraries can handle computations in high dimensional vector spaces, i.e. dimension where the memory storage of the multivector becomes problematic, usually dimension 12. In addition, more and more Geometric Algebra applications focus on high dimensional spaces. It started with Conformal Geometric Algebra (CGA) which is certainly one of the most studied [73]. CGA is built from 5 dimensional vector space, and includes $2^5 = 32$ basis vectors. Although some people consider 32 components per multivector to be already high, some studies are conducted to explore even higher dimensions. Easter and Hitzer [27] represent some quartics and quadrics 3-d shapes using a double conformal geometry of $\mathbb{R}^3$. Extending this process to a triple conformal geometry would lead to a 15 dimensional algebra containing $2^{15} = 32,768$ elements. For such geometry, the memory requirement

for optimized libraries explodes far beyond consumer grade hardware capabilities. More regular approaches lead to very long processing time.

Contributions

These issues are the initial motivation to create a new library called *Garamon* (Geometric Algebra Recursive and Adaptive Monster), a library generator written in C++ programming language and producing specialized Geometric Algebra (GA) libraries also in C++. We propose a flexible, portable, and computationally efficient Geometric Algebra implementation dedicated for both low and high dimensions. More precisely, the proposed method combines pre-computation of products of Geometric Algebra for low dimensions with a method using efficient recursive formulas for high dimensions. The proposed approach includes new computationally efficient Geometric Algebra recursive scheme. The first method is based on binary trees and is specifically dedicated to parallel algorithm. The performance of the method is evaluated in terms of time complexity. This leaded to the publication of [5]. The second method is based on automaton structure. Again, a complexity study both in terms of time and memory is presented. Our paper [6] detailed the integration of this hybrid method as a plug-in into Gaalop [12], which is a very advanced optimizing code generator. Finally, the third method is based on prefix tree structure. We explain

Thanks to the fact that the resulting implementation allow comfortable use of Geometric Algebra for high dimensional space, Garamon allowed us to perform some tests of high dimensional model. These tests were the base for the development of our proposed Geometric Algebra framework.

This framework, referred to as Quadric Conformal Geometric Algebra (QCGA), is an extension of CGA, specifically dedicated to quadric surfaces. Through generalizing the conic construction in $\mathbb{R}^2$ by Perwass [73], we prove that QCGA can construct quadric surfaces using either control points or implicit equations. Moreover, QCGA can compute the intersection of quadric surfaces, the surface tangent, and normal vectors for a quadric surface point. QCGA can be the support for computing transformations. This contribution led to the publication of [7]. Finally, we propose and detail a computational framework of the quadric surfaces using geometric Algebra.

In practice, this PhD. manuscript is divided into three main parts. The first part is a brief introduction to Geometric Algebra where we define the most important notions starting with Grasmmann algebra to finish with Clifford Algebra. Then the second part is dedicated to the proposed library of Geometric Algebra. Finally, the third part focuses on the representation and manipulation of quadric surfaces using Geometric Algebra.

Introduction (in French)

L'algèbre géométrique est un outil mathématique de représentation et de manipulation d'objets géométriques initialement conceptualisé par Hermann Grassmann en 1844. À la même époque, Wiliam Rowan Hamilton (1805-1865) développait l'algèbre des quaternions utilisée pour les rotations 3D. Enfin William Kingdon Clifford (1845-1879) produisit une algèbre permettant d'englober toutes les algèbres (Grassmann, quaternion, ...). Leurs travaux n'ont été repris seulement qu'à la fin du XX^{e} siècle grâce à l'avènement de l'informatique. David Hestenes a remis au goût du jour cette algèbre en résolvant des problèmes de mécanique quantique avec l'algèbre géométrique. Au début du XXI^{e} siècle, l'algèbre géométrique s'est développée, et ainsi d'autres intéressantes applications sont apparues, notamment dans la manipulation de données pour les systèmes d'information géographique (GIS), voir [63, 86, 84]. Malgré le fait que la littérature en algèbre géométrique manque de contenu pour l'estimation au sens des moindres carrés, des applications en vision par ordinateur ont émergé. La représentation des modèles d'appareil photographique par l'algèbre géométrique occupe une grande partie de ces applications, par exemple [80, 55]. Enfin, d'autres applications récentes s'intéressent à l'apprentissage profond par l'utilisation de "Clifford neurons" [9]. Pour un historique plus complet et plus d'applications, se référer à [23] ou [55].

Tout comme l'algèbre linéaire usuellement utilisée pour représenter les objets géométriques, cette algèbre est définie à partir d'un espace vectoriel. Sa spécificité est de pouvoir représenter des objets géométriques par des sous-espaces vectoriels engendrés par des opérateurs de l'algèbre.

Voici quelques exemples d'opérations typiques que l'on peut rencontrer en algèbre géométrique. Dans l'espace usuel de dimension 3, considérons 4 points représentés par 4 vecteurs $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ de l'algèbre, il est possible de calculer la sphère $\mathbf{S}$ passant par ces 4 points de la manière suivante:

$$\mathbf{S} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \tag{0.0.1}$$

Avec le même opérateur et à partir de trois points dont les vecteurs sont $\mathbf{x}_5, \mathbf{x}_6, \mathbf{x}_7$ et un point à l'infini de vecteur $\mathbf{e}_\infty$ (similaire à la coordonnée homogène en géométrie projective), on détermine

le plan Π passant par ces trois points par la formule suivante:

$$\boldsymbol{\Pi} = \mathbf{x}_5 \wedge \mathbf{x}_6 \wedge \mathbf{x}_7 \wedge \mathbf{e}_\infty \tag{0.0.2}$$

Enfin avec un deuxième opérateur étant une extension du produit scalaire de l'algèbre linéaire appelé *contraction* et noté $\rfloor$, on calcule le cercle $\mathbf{C}$ obtenu par l'intersection entre le plan $\boldsymbol{\Pi}$ et la sphère $\mathbf{S}$:

$$\mathbf{C} = \boldsymbol{\Pi} \rfloor \mathbf{S} \tag{0.0.3}$$

Enfin, une droite est simplement obtenue par 2 points $\mathbf{x}_7, \mathbf{x}_8$ et un point à l'infini:

$$\mathbf{L} = \mathbf{x}_7 \wedge \mathbf{x}_8 \wedge \mathbf{e}_\infty \tag{0.0.4}$$

mais aussi par l'intersection de deux plans $\boldsymbol{\Pi}_1, \boldsymbol{\Pi}_2$:

$$\mathbf{L} = \boldsymbol{\Pi}_1 \rfloor \boldsymbol{\Pi}_2 \tag{0.0.5}$$

Ces exemples illustrent la capacité de compacité des expressions de l'algèbre géométrique pour la définition de primitives géométriques, points, sphères, plans, droites, cercles. Par ailleurs, les calculs effectifs produits par ces opérateurs sont, à l'image du produit vectoriel de $\mathbb{R}^3$, extrêmement compacts. Tout ceci fait de l'algèbre géométrique un outil pratique et efficace pour la résolution des problèmes géométriques.

L'expression et le traitement d'objets plus complexe, telle que les coniques, les surfaces quadriques est également possible en utilisant une extension de l'algèbre utilisée pour les sphères. Cependant due à leur représentation requérant un espace vectoriel de haute dimension, les implantations actuels de l'algèbre géométrique n'autorisent pas une utilisation efficace de ces objets.

En effet, il existe un certain nombre d'implantations d'algèbre géométrique qui couvrent un large spectre de langage de programmation et d'applications. La plupart des modèles d'algèbre géométrique peut être traitée de manière confortable grâce aux implantations existantes. Cependant, très peu de ces librairies sont capables de traiter des calculs nécessitant des espaces vectoriels de hautes dimensions, c.a.d. supérieures à 12. La taille des sous-espaces générés par ces espaces vectoriels de hautes dimensions induit des calculs très couteux. Les calculs effectués deviennent ainsi lents.

Plus précisément, les implantations existantes peuvent être classifiées en deux groupes. Le premier groupe correspond aux générateurs de code. Ces générateurs de code optimisent l'algèbre géométrique par génération de code bas niveau et effectuent des optimisations symboliques en terme d'algèbre. Dans cette catégorie, Gaalop [12] est une des librairies les plus avancée de cette catégorie. Cette librairie utilise des opérations symboliques écrites en CLUCalc [72] et peut produire un code optimisé soit en C++, OpenCL, CUDA or LaTeX. Les optimisations choisies résultent en un code optimisé pour une tâche spécifique. Cependant ces optimisations ne sont pas bien adaptées pour des cas où les calculs ne sont pas décidés à l'avance. Par ailleurs, le code source généré doit être intégré manuellement dans l'application de l'utilisateur, ce qui fait perdre en flexibilité.

Le deuxième groupe correspond aux générateurs de librairies définis à partir d'une spécification de l'algèbre. Dans ce type d'implantation, le produit entre les objets de l'algèbre géométrique ayant une structure particulière est défini à l'avance. Gaigen [23, 33], présenté par Daniel Fontijne, a été la première et la plus aboutie dans cette catégorie. Gaigen signifie Geometric Algebra Implementation GENerator et cette librairie peut générer du code source C++,C,C# et Java, implantant une algèbre géométrique de dimension de l'espace vectoriel et de métrique fixées. Les objets de cette algèbre n'ayant pas une structure bien définie sont associés à des classes générales présentant de faibles niveaux d'optimisation. Finalement, ce générateur de librairie est limité en terme de dimension de l'espace vectoriel. Plus précisément, les optimisations limitent la librairie d'être utilisée pour des algèbres dont la dimension de l'espace vectoriel dépasse 10.

Toutes ces approches présentent d'intéressantes propriétés. Cependant, des améliorations doivent être apportées pour faire en sorte que ces librairies présentent une facilité d'utilisation, mais aussi de meilleures performances en terme de mémoire et de temps de calcul.

Ces derniers points ont été notre motivation initiale pour créer un générateur de librairies d'algèbre géométrique, appelée *Garamon* (Geometric Algebra Recursive and Adaptive Monster). Nos besoins pour cette librairie ont été tout d'abord d'être capable de supporter le traitement et la représentation de surfaces quadriques en algèbre géométrique. Par ailleurs, cette librairie doit présenter un confort d'utilisation permettant d'effectuer des opérations dans des algèbres à espaces vectoriels aussi bien basses que hautes dimensions. La première partie de cette thèse est dédiée à la présentation de cette implantation. Plus précisément, nous proposons, dans cette partie, une implantation performante en terme de mémoire et de temps de calcul.

Dans cette partie, nous proposons tout d'abord une structure de données encodant la structure de l'algèbre géométrique. En effet, pour un espace vectoriel de dimension d, les informations qui peuvent être stockées pour représenter les éléments de l'algèbre linéaire (vecteurs et matrices) diffèrent des de ceux utilisés en algèbre géométrique. En algèbre linéaire, la complexité en algèbre linéaire est en $\mathcal{O}(d^2)$ alors que cette complexité est de $\mathcal{O}(2^d)$ pour l'algèbre géométrique. Cette différence influence leur respective structure de données. Par conséquent, les implantations d'algèbre linéaire représentent fréquemment **toutes** les données composant un vecteur ou une matrice. En contraste, les implantations de l'algèbre géométrique essaient de faire en sorte de ne stocker que les éléments non nuls. Une façon de prendre en compte cette contrainte passe par l'utilisation d'une liste chaînée. Nous proposons une structure de données à base de dictionnaire et permettant d'obtenir une complexité proportionnelle à la taille d'un sous espace de l'algèbre et plus en 2^d.

Il est ensuite question de l'implantation des opérateurs de l'algèbre géométrique. Nous proposons une méthode hybride consistant à effectuer les pré-calculs en basse dimension et à utiliser des méthodes efficaces de calculs pour des espaces vectoriels de haute dimension. Pour ces espaces de basses dimensions, cela consiste à utiliser des fonctions contenant tous les calculs à effectuer pour tout type d'objets de l'algèbre. Pour des espaces vectoriels de haute dimension, nous proposons de nouvelles méthodes récursives à base d'arbre. Plus précisément, les méthodes de calcul des produits de l'algèbre géométrique ont une complexité de calcul en $\mathcal{O}(d \times 4^d)$. Ce qui pose problème pour des algèbres de hautes dimensions. Pour résoudre ce problème, nous présentons plusieurs méthodes de calcul permettant de réduire cette complexité à $\mathcal{O}(3^d)$. Nous montrerons que la méthode choisie permet d'effectuer dans le pire des cas exactement 3^d opérations. Ceci fait de notre méthode, une approche adaptée aux hautes dimensions. Par ailleurs, une nouvelle méthode de calcul dans les bases non orthogonales sera explicitée. Cette méthode permet de solutionner les problèmes de robustesse induit par un changement de base en algèbre géométrique.

Nous présenterons la méthode permettant la transition entre l'approche pour les espaces vectoriels de basse dimension et la méthode pour les espaces vectoriels de haute dimension. Brièvement, le critère se base sur la taille des sous-espaces générés impliqués dans le produit considéré. Finalement, cette dernière implantation nous a ouvert de nouvelles perspectives, tout particulièrement dans l'étude d'un modèle de représentation à haute dimension des surfaces quadriques en algèbre géométrique.

Ce modèle sera l'objet de la seconde partie de ce manuscrit de thèse. Tout d'abord, l'algèbre géométrique de l'espace vectoriel $\mathbb{R}^{p,q}$ est symbolisé par $\mathbb{G}_{p,q}$, où p est le nombre de vecteurs de base dont le carré vaut 1, et q est celui dont le carré vaut -1. Différents modèles d'algèbre géométrique sont dédiées à la représentation de coniques. $\mathbb{G}_{5,3}$ [73] permet d'exprimer de manière élégante les coniques de $\mathbb{R}^2$. Les coniques de $\mathbb{R}^2$ peuvent aussi être exprimé dans $\mathbb{G}_{6,2}$, voir [27]. Il existe également différens modèles d'algèbres géométriques dédiés à la représentation de surfaces quadriques. Un premier modèle développé par Zamora [85] autorise la construction de quadriques à partir de points de contrôle. Cependant cette dernière approche ne traite que les quadriques centrées et alignées. Se représentation n'inclut donc pas toute forme de quadriques, appelées quadriques générales par la suite.

Il existe deux approches générales permettant de traiter les quadriques générales. Premièrement, DCGA [27] (Double Conformal Geometric Algebra) avec $\mathbb{G}_{8,2}$, défini par Easter et Hitzer, construit les surfaces quadriques et certaines quartiques à partir des coefficients de leur forme implicite. Ce modèle permet de représenter des surfaces et d'effectuer des transformations sur ces objets. Cependant, ce modèle ne permet pas de calculer l'intersection entre deux quadriques générales et donc ne construit pas de quadriques à partir de points de contrôles.

La deuxième approche a été introduite par Parkin [70] puis développée par Goldman et al. [38] et enfin finalisée par Du et al. [26] utilise l'algèbre $\mathbb{G}_{4,4}$. Cette algèbre consiste en une duplication de la géométrie projective de $\mathbb{R}^3$ et est donc nommé Double Projective Geometric Algebra, appelée dans cette thèse DPGA. DPGA permet de traiter les intersections entre quadriques et les coniques. Cette approche permet également d'effectuer des transformations sur des quadriques. Cependant, elle ne permet pas de construire des quadriques à partir de points de contrôle.

Nous présentons dans cette partie Quadric Conformal Geometric Algebra (QCGA). Brièvement, il s'agit d'une extension de CGA, présenté dans [21]. Par une généralisation de [73], nous prouverons que QCGA est capable de construire les quadriques générales à partir de points de contrôle mais également à partir de la forme implicite de la quadrique. Nous proposerons une méthode permettant de calculer l'intersection de quadriques. Par ailleurs, nous prouverons qu'il est possible de représenter un plan tangent, un vecteur normal à une quadrique générale à partir d'un point de cette quadrique. Une partie sera dédiée à la représentation des transformations dans QCGA, typiquement nous montrerons que les translations peuvent être représentées de manière intuitive dans QCGA. Nous prouverons finalement que le modèle proposé inclut les

approches de l'état de l'art, ce qui permettra d'aboutir à un modèle général efficace de représentation des surfaces quadriques.

Chapter 1

Construction of Geometric Algebra

In applications such as computer vision and computer graphics, a typical operation consists in having representation of geometric objects and intersect these objects. In Linear Algebra, one has a list of operations to deal with these kinds of problems, that are performed in an efficient way. However, they are some limitations that come from the fact that the solution is expressed with respect to coordinates and "low level" operations. For example, the intersection of two lines involves the representation of lines. A line l is expressed as the coordinates (x, y, z) that satisfy:

$$\begin{aligned} x &= x_1 + t(x_1 - x_2) \\ y &= y_1 + t(y_1 - y_2) \\ z &= z_1 + t(z_1 - z_2) \end{aligned} \tag{1.0.1}$$

where, (x_1, y_1, z_1) and (x_2, y_2, z_2) are the coordinates of two points that lie on the considered line. Then the intersection of two lines consists in solving a linear system leading to a list of operations. This is completely sufficient to just have this list of operations. However, this becomes unintuitive and cumbersome, when one has to deal with geometric problems that require high level of abstractions. In fact, the geometric natures of the operations and representations are lost. This restrains the user of the algebra to think the problems in terms of coordinates operations instead of objects.

In contrast, Geometric Algebra is designed to be generic and to offer operators that are able to compute the considered geometric operations for any geometric objects of the algebra and in any dimension. As an example, in an algebra, with only one operator called outer product, noted as $\wedge$, and the entity representing a point, it is possible to construct in a very intuitive a set of geometric objects. For example, a line can be expressed as:

$$\mathbf{l} = \mathbf{x}_1 \wedge \mathbf{x}_2 \tag{1.0.2}$$

And the intersection between two lines involves the same operator and in a very compact way. This intersection operation is not limited to lines but also to any geometric objects of this algebra. Moreover, the operators involved in the computation of the intersection remain the same whatever the geometric object is. This leads to very intuitive and ”geometric object" thinking oriented Algebra.

1.1 The battle of Vector Algebra

Despite these powerful features, this Geometric Algebra did not develop as expected. This section gives some historical elements to understand why. More precisely, we focus on the reason why so few researchers know Geometric Algebra compared to the Vector Algebra, please refer to Figure 1.1 for a quick view of some of the main contributors of Vector Algebra.

In the history of Vector Algebra, William Rowan Hamilton extended the complex numbers to 3-dimensional space and produced the quaternions in 1843 [42]. These quaternions succeeded in representing 3D rotations. Meanwhile, raised Hermann Günther Grassmann extensive algebra [39] seen as the first contributor to the Geometric Algebra. The key point of his research is to develop an algebra that could be used to as “a universal instrument for geometric research”, as stated in [43]. He developed a high dimensional vector algebra whose entities and products have geometric interpretations. But partly due to the fact that the major paper was full of philosophical statements, Grassmann extensive algebra did not develop that much.

Then, Josiah Willard Gibbs defined the 3—dimensional space vector along with two products, namely the cross product and the dot product. These products have the advantage to be simple to use. These products also helped to express the four differential equations which were published in a developed form in 1873 [66]. This tended to develop much more Gibbs work than Geometric Algebra.

However, with the arrival of special relativity, physicists needed a framework that handles 4-dimensional space. This leaded to the work of Minkowski algebra [67], which can also be easily derived from Geometric Algebra.

In 1878, William Kingdon Clifford unified the work of Grassmann and Hamilton into a whole framework, namely Geometric Algebra. However, the development of this algebra stopped. Indeed, Clifford died at the age of 34 which is only 1 year after the publication of his paper on

FIGURE 1.1: Some of the main contributors to the vector algebras

Applications of Grassmann's Extensive Algebra [13]. Despite this, some other algebras were created and can be derived from Geometric Algebra, for example Pauli algebra, Dirac algebra [18].

Geometric Algebra stayed almost undeveloped until the 1960s. David Hestenes rescued Geometric Algebra for physics applications, as relativity [46] and in classical mechanics [45]. Geometric Algebra is now in the process of catching up in engineering, video game and physics compared to Gibbs vector.

One might consider for example [11, 15] for a more detailed history. Let us now take a further look at Geometric Algebra.

1.2 Vectors of Geometric Algebra

Geometric Algebra is an algebra over a field. We will only consider the real field, $\mathbb{R}$. Vectors of Geometric Algebra are denoted with bold font lower case letter, e.g. $\mathbf{a}, \mathbf{b}$ are two vectors. Gibbs vectors are usually denoted as column of numbers. Whereas, it is more convenient to note vectors of Geometric Algebra as linear combination according to some basis vectors (e.g.$\mathbf{e}_1$, $\mathbf{e}_2$). Coefficients of vectors are symbolized with italic lower-case letters(a_1, x, y^2, $\cdots$). For example, a

3-dimensional vector $\mathbf{a} \in \mathbb{R}^3$ of Geometric Algebra:

$$\mathbf{a} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3 \tag{1.2.1}$$

For any d—dimensional space, a vector is thus defined as follows:

$$\mathbf{a} = \sum_{i=1}^{d} a_i\mathbf{e}_i \tag{1.2.2}$$

As we consider a vector space, we can compute the multiplication of a vector by a scalar. We represent scalars using lower-case default text font (constant scalar r). The scalar multiplication is distributive thus $\forall \mathbf{a} \in \mathbb{R}^d$ and $\forall \mathrm{r} \in \mathbb{R}$:

$$\mathrm{r}\mathbf{a} = \mathrm{r}\sum_{i=1}^{d} a_i\mathbf{e}_i = \sum_{i=1}^{d} \mathrm{r}a_i\mathbf{e}_i \tag{1.2.3}$$

Finally the addition of two vectors can also be computed. Still, with $\mathbf{a} \in \mathbb{R}^d$ and $\mathbf{b} \in \mathbb{R}^d$, we get:

$$\mathbf{a} + \mathbf{b} = \sum_{i=1}^{d} (a_i + b_i)\mathbf{e}_i \tag{1.2.4}$$

1.3 First major product of Geometric Algebra

There are several products of Geometric Algebra and one of the major products of Geometric Algebra is the outer product, also called wedge product. The outer product of two vectors $\mathbf{a}$ and $\mathbf{b}$ is written as $\mathbf{a} \wedge \mathbf{b}$ and read as $\mathbf{a}$ "wedge" $\mathbf{b}$. This product has a geometric meaning. Indeed, the outer product of $\mathbf{a} \in \mathbb{R}^d$ and $\mathbf{b} \in \mathbb{R}^d$ expresses the oriented surface spanned by $\mathbf{a}$ and $\mathbf{b}$. An example is shown in Figure 1.2.

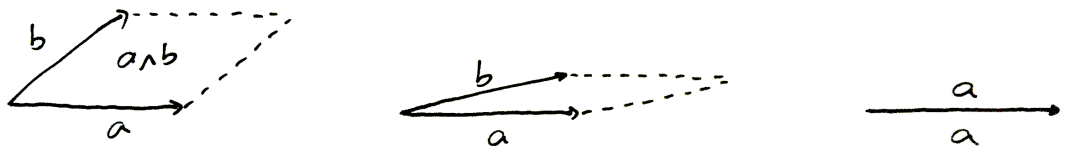


FIGURE 1.2: Geometric meaning of the outer product between two vectors.

The main properties of the outer product are as follows:

scalar product:	$\mathrm{r} \wedge \mathbf{a} = \mathrm{r}\mathbf{a}$	$\forall \mathbf{a} \in \mathbb{R}^d, \mathrm{r} \in \mathbb{R}$
associativity:	$(\mathbf{a} \wedge \mathbf{b}) \wedge \mathbf{c} = \mathbf{a} \wedge (\mathbf{b} \wedge \mathbf{c})$	$\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^d$
anti-commutativity:	$\mathbf{a} \wedge \mathbf{b} = -\mathbf{b} \wedge \mathbf{a}$	$\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^d$
	$\mathbf{a} \wedge \mathbf{a} = 0$	$\forall \mathbf{a} \in \mathbb{R}^d$
distributivity:	$\mathbf{a} \wedge (\mathbf{b} + \mathbf{c}) = (\mathbf{a} \wedge \mathbf{b}) + (\mathbf{a} \wedge \mathbf{c})$	$\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^d$
scaling:	$\mathbf{a} \wedge (\mathrm{r}\mathbf{b}) = \mathrm{r}(\mathbf{a} \wedge \mathbf{b})$	$\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^d, \mathrm{r} \in \mathbb{R}$

Note that the anti-commutativity property is an essential property. Furthermore, this property is geometrically intuitive. Indeed, the resulting oriented area formed by $\mathbf{a}$ and itself is zero (see Figure 1.2) and the oriented area spanned by $\mathbf{a}$ and $\mathbf{b}$ is the opposite of the oriented area spanned by $\mathbf{b}$ and $\mathbf{a}$.

1.3.1 2-blades

The outer product of two vectors forms a subspace, called a 2-blade. This 2-blade is expressed in a new basis, namely the basis of 2-blades or bivectors. For example in $\mathbb{R}^2$, the outer product between two vectors $\mathbf{a}$ and $\mathbf{b}$ is expressed as:

$$\begin{aligned}
\mathbf{a} \wedge \mathbf{b} &= (a_1\mathbf{e}_1 + a_2\mathbf{e}_2) \wedge (b_1\mathbf{e}_1 + b_2\mathbf{e}_2) \\
&= (a_1b_1)(\mathbf{e}_1 \wedge \mathbf{e}_1) + (a_1b_2)(\mathbf{e}_1 \wedge \mathbf{e}_2) \\
&\qquad + (a_2b_1)(\mathbf{e}_2 \wedge \mathbf{e}_1) + (a_2b_2)(\mathbf{e}_2 \wedge \mathbf{e}_2) \\
&= (a_1b_2 - a_2b_1)(\mathbf{e}_1 \wedge \mathbf{e}_2) \\
&= (a_1b_2 - a_2b_1)\mathbf{e}_{12}
\end{aligned} \tag{1.3.1}$$

For sake of clarity, one notes the basis $\mathbf{e}_1 \wedge \mathbf{e}_2$ as $\mathbf{e}_{12}$. This 2-blade $\mathbf{e}_{12}$ forms the basis of 2-blades in $\mathbb{R}^2$. In $\mathbb{R}^d$, we define the pseudo-scalar as $\mathbf{I} = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \cdots \wedge \mathbf{e}_d = \mathbf{e}_{12\ldots d}$. Therefore, in $\mathbb{R}^2$, the 2-blade $\mathbf{e}_{12}$ is also $\mathbf{I}$.

In $\mathbb{R}^3$, the outer product between two vectors is as follows:

$$\begin{aligned}
\mathbf{a} \wedge \mathbf{b} &= (a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3) \wedge (b_1\mathbf{e}_1 + b_2\mathbf{e}_2 + b_3\mathbf{e}_3) \\
&= (a_1b_2 - a_2b_1)\mathbf{e}_{12} + (a_1b_3 - a_3b_1)\mathbf{e}_{13} + (a_2b_3 - a_3b_2)\mathbf{e}_{23}
\end{aligned} \tag{1.3.2}$$

The spanned 2-blades basis is now $\mathbf{e}_{12}, \mathbf{e}_{13}, \mathbf{e}_{23}$.

Furthermore, by a slight rewriting of the above equation as:

$$\mathbf{a} \wedge \mathbf{b} = (a_1b_2 - a_2b_1)\mathbf{e}_{12} + (a_3b_1 - a_1b_3)\mathbf{e}_{31} + (a_2b_3 - a_3b_2)\mathbf{e}_{23} \tag{1.3.3}$$

This formula looks like the cross product, indeed the cross product between two Gibbs vectors is:

$$\mathbf{a} \times \mathbf{b} = (a_1b_2 - a_2b_1)\,\mathbf{e}_3 + (a_3b_1 - a_1b_3)\,\mathbf{e}_2 + (a_2b_3 - a_3b_2)\,\mathbf{e}_1 \tag{1.3.4}$$

However the outer product is:

- actually defined in any dimension
- associative $(\mathbf{a} \wedge \mathbf{b}) \wedge \mathbf{c} = \mathbf{a} \wedge (\mathbf{b} \wedge \mathbf{c})$

1.3.2 3-blades

In $\mathbb{R}^3$, it is also possible to build higher dimensional subspace, namely three-dimensional subspace. It is formed by the outer product of 3 vectors, e.g. the outer product of $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^3$ is as:

$$\mathbf{a} \wedge \mathbf{b} \wedge \mathbf{c} = \Big(c_3(a_1b_2 - a_2b_1) + c_2(a_1b_3 - a_3b_1) + c_1(a_2b_3 - a_3b_2)\Big)\,\mathbf{e_{123}} \tag{1.3.5}$$

The geometric meaning of this outer product between three vectors is the oriented volume spanned by the three vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$, as shown in Figure 1.3.

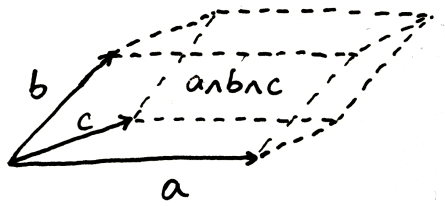


FIGURE 1.3: Geometric meaning of the outer product between three vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}$ of $\mathbb{R}^3$.

Note that the outer product between three vectors of $\mathbb{R}^3$ has only one component. Thus, by regrouping the basis subspaces spanned along with scalar component, we get the new basis:

$$\Big(\underbrace{\mathbf{1}}_{\text{scalars}}, \underbrace{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3}_{\text{vector space}}, \underbrace{\mathbf{e}_{12}, \mathbf{e}_{13}, \mathbf{e}_{23}}_{\text{bivector space}}, \underbrace{\mathbf{e}_{123}}_{\text{trivector space}}\Big) \tag{1.3.6}$$

Let us also introduce two other very useful and powerful properties of the outer product. By considering 3 vectors $\mathbf{x}, \mathbf{a}, \mathbf{b}$, we can verify that:

$$\begin{aligned} \mathbf{x} \wedge \mathbf{a} = 0 \quad &\Leftrightarrow \quad \mathbf{x} = r\mathbf{a} \qquad r \in \mathbb{R} \\ \mathbf{x} \wedge \mathbf{a} \wedge \mathbf{b} = 0 \quad &\Leftrightarrow \quad \mathbf{x} = r\mathbf{a} + s\mathbf{b} \quad r, s \in \mathbb{R} \end{aligned} \tag{1.3.7}$$

1.3.3 Higher dimensional subspaces

For higher dimensional spaces, higher graded subspaces are spanned. This leads to the notion of k-blades. An entity of the algebra is a k-blade (by bold capital letters) if and only if it is factorizable into the outer product of k vectors, for example:

$$\mathbf{A} = \mathbf{e}_{12} + \mathbf{e}_{13} - \mathbf{e}_{23} \tag{1.3.8}$$

Indeed, a factorization of this entity into outer product of vectors is found as follows:

$$\mathbf{a} = (\mathbf{e}_1 + \mathbf{e}_3) \wedge (\mathbf{e}_2 + \mathbf{e}_3) \tag{1.3.9}$$

In contrast, the entity:

$$B = 2\mathbf{e}_{12} + 3\mathbf{e}_{34} \tag{1.3.10}$$

in a 4-dimensional vector space, cannot be factorized as the outer product of two vectors. These entities are called k-vector (by nonbold capital letters). The set of k-blades is included in the set of k-vectors. This relation is illustrated in Figure 1.4.

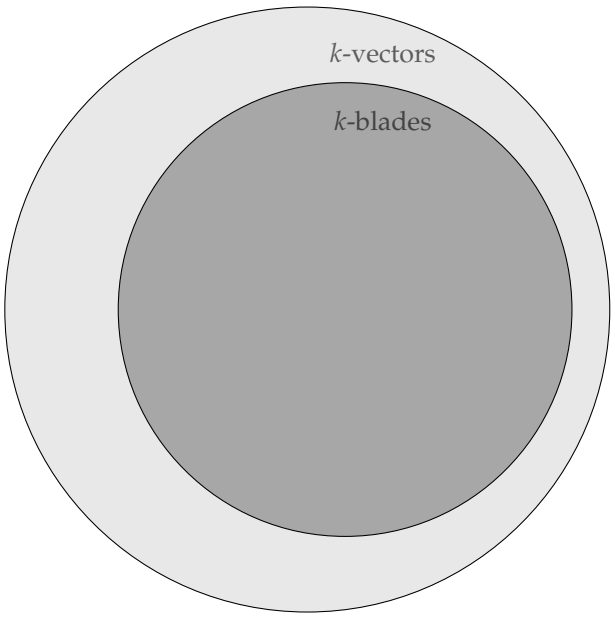


FIGURE 1.4: Relation between k-vectors and k-blades

In the term k-blade and k-vector, k is called the **grade**. As an example, the construction of the basis of the algebra of 4-dimensional vector along with the grade information are as follows:

$(\mathbf{1})$	grade 0
$(\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}, \mathbf{e_4})$	grade 1
$(\mathbf{e_{12}}, \mathbf{e_{13}}, \mathbf{e_{14}}, \mathbf{e_{23}}, \mathbf{e_{24}}, \mathbf{e_{34}})$	grade 2
$(\mathbf{e_{123}}, \mathbf{e_{124}}, \mathbf{e_{134}}, \mathbf{e_{234}})$	grade 3
$(\mathbf{e_{1234}})$	grade 4

For any k-dimensional subspace of a d-dimensional vector space, it is possible to compute the number of k-basis blade as:

$$\binom{d}{k} = \frac{d!}{(d-k)!d!} \tag{1.3.11}$$

Finally, using the binomial theorem, the total number of basis blades is computed as:

$$\sum_{i=0}^{d} \binom{d}{i} = 2^d \tag{1.3.12}$$

The table 1.1 shows some values of the total number of basis for some vector space dimension. Finally, we can form a sum of basis blades in a new vector space whose dimension is 2^d. For

Dimension	Number of independent vectors of each k-basis blade	total
3	1 3 3 1	8
4	**1 4 6 4 1**	16
5	1 5 10 10 5 1	32
⋮	⋮	⋮
10	1 10 45 120 210 252 210 120 45 10 1	1024
⋮	⋮	⋮

TABLE 1.1: Structure of a general entity of Geometric Algebra

example, it is possible to define a *multivector* **a** of a 3-dimensional vector space as:

$$A = \mathrm{r} + a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3 + a_{12}\mathbf{e}_{12} + a_{13}\mathbf{e}_{13} + a_{23}\mathbf{e}_{23} + a_{123}\mathbf{e}_{123} \tag{1.3.13}$$

where $\mathrm{r}, a_1, a_2, a_3, a_{12}, a_{13}, a_{23}, a_{123} \in \mathbb{R}$

1.4 Inner product

1.4.1 Metric vector space

To the algebra is assigned an inner product. The inner product between $\mathbf{a}$ and $\mathbf{b}$, is denoted as $\mathbf{a} \cdot \mathbf{b}$. The inner product which maps two vectors to a scalar, that is to say $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, is also called the scalar product. The properties of this product are:

Commutativity:	$\mathbf{a} \cdot \mathbf{b} = \mathbf{b} \cdot \mathbf{a}$	$\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^d$
Linearity:	$\mathbf{a} \cdot (\mathrm{r}\mathbf{b} + \mathrm{s}\mathbf{c}) = \mathrm{r}(\mathbf{a} \cdot \mathbf{b}) + \mathrm{s}(\mathbf{a} \cdot \mathbf{c})$	$\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^d, \forall \mathrm{r}, \mathrm{s} \in \mathbb{R}$

For two Euclidean vector $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$, the inner product $\mathbf{a} \cdot \mathbf{b}$ is as follows:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= (a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3) \cdot (b_1\mathbf{e}_1 + b_2\mathbf{e}_2 + b_3\mathbf{e}_3) \\ &= (a_1b_1)(\mathbf{e}_1 \cdot \mathbf{e}_1) + (a_2b_2)(\mathbf{e}_2 \cdot \mathbf{e}_2) + (a_3b_3)(\mathbf{e}_3 \cdot \mathbf{e}_3) \\ &= (a_1b_1) + (a_2b_2) + (a_3b_3) \end{aligned} \tag{1.4.1}$$

Note that the inner product of two Euclidean vectors $\mathbf{e}_i \cdot \mathbf{e}_j = 0$, if $i \neq j$. Using this inner product, it is possible to define the signature of the vector space. Note that we do not assume positive definiteness of the inner product. For a vector space, the signature is defined as the triplet (p, q, r). More precisely, this means that:

$$\mathbf{e}_i \cdot \mathbf{e}_i = \begin{cases} +1 & \text{for } i = 1, \cdots, p \\ -1 & \text{for } i = p+1, \cdots, p+q \\ 0 & \text{for } i = p+q+1, \cdots, p+q+r \end{cases} \tag{1.4.2}$$

We will use the common notation of $\mathbb{R}^{p,q,r}$ for a vector space with the signature (p, q, r). For example, we can define the Minkowski space as the vector space $\mathbb{R}^{3,1,0}$ or equivalently $\mathbb{R}^{1,3,0}$. Note that this is defined up to change of basis.

To express the inner product of vectors, we will also use the metric matrix defining $\mathbf{e}_i \cdot \mathbf{e}_j, \forall (i, j) \in [1, d]^2$. The simplest metric matrix is the Euclidean metric, it is simply a identity matrix whose rank is the dimension of the vector space. A metric matrix that is used especially in relativity is the metric matrix associated with the vector space $\mathbb{R}^{1,3}$. This is the diagonal metric matrix given

as follows:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{1.4.3}$$

1.4.2 Between blades of different grade

We defined the inner product between vectors. This section aims at defining it also between blades of different grades.

1-blade · m-blade: The inner product of a vector $\mathbf{a}$ with a m-blade $\mathbf{B}$ in a d-dimensional space ($d \geq l$) is defined as:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{B} &= \mathbf{a} \cdot (\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \cdots \wedge \mathbf{b}_m) \\ &= (\mathbf{a} \cdot \mathbf{b}_1)\mathbf{b}_2 \wedge \cdots \wedge \mathbf{b}_m - (\mathbf{a} \cdot \mathbf{b}_2)\mathbf{b}_1 \wedge \mathbf{b}_3 \cdots \wedge \mathbf{b}_m + \cdots \\ &\quad + (-1)^{m+1}(\mathbf{a} \cdot \mathbf{b}_m)\mathbf{b}_1 \wedge \cdots \wedge \mathbf{b}_{m-2} \wedge \mathbf{b}_{m-1} \end{aligned} \tag{1.4.4}$$

Note that the grade of the result is $m - 1$.

m-blade · 1-blade: The inner product of two vectors is commutative. By definition, the inner product between a vector and a m-blade results in:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{B} &= \mathbf{a} \cdot (\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \cdots \wedge \mathbf{b}_m) \\ &= (-1)^{m-1}(\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \cdots \wedge \mathbf{b}_m) \cdot \mathbf{a} \\ &= (-1)^{m-1}\mathbf{B} \cdot \mathbf{a} \end{aligned} \tag{1.4.5}$$

k-blade · m-blade: Now, we consider the inner product between a k-blade $\mathbf{A}$ and m-blade $\mathbf{B}$. $\mathbf{A} \cdot \mathbf{B}$ is defined as follows:

$$\begin{aligned} \mathbf{A} \cdot \mathbf{B} &= (\mathbf{a}_1 \wedge \mathbf{a}_2 \wedge \cdots \wedge \mathbf{a}_k) \cdot (\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \cdots \wedge \mathbf{b}_m) \\ &= \Big(\mathbf{a}_1 \cdot \Big(\mathbf{a}_2 \cdot \Big(\cdots \big(\mathbf{a}_k \cdot (\mathbf{b}_1 \wedge \mathbf{b}_2 \wedge \cdots \wedge \mathbf{b}_m)\big)\Big)\Big)\Big) \end{aligned} \tag{1.4.6}$$

Thus, the grade of the result is $|m - k|$. Note that when $m = k$, the result of the inner product is a scalar.

***m*-blade · *k*-blade:** Using the same blade $\mathbf{A}$ and $\mathbf{B}$, we also deduce:

$$\begin{aligned}\mathbf{A}\cdot\mathbf{B} &= (\mathbf{a}_1\wedge\mathbf{a}_2\wedge\cdots\wedge\mathbf{a}_k)\cdot(\mathbf{b}_1\wedge\mathbf{b}_2\wedge\cdots\wedge\mathbf{b}_m)\\ &= (-1)^{k(m+1)}(\mathbf{b}_1\wedge\mathbf{b}_2\wedge\cdots\wedge\mathbf{b}_m)\cdot(\mathbf{a}_1\wedge\mathbf{a}_2\wedge\cdots\wedge\mathbf{a}_k)\\ &= (-1)^{k(m+1)}(\mathbf{B}\cdot\mathbf{A})\end{aligned} \tag{1.4.7}$$

1.4.3 Reverse and squared norm of blades

The reverse of a *k*-blade $\mathbf{A}$, is noted as $\widetilde{\mathbf{A}}$. As $\mathbf{A}$ is a *k*-blade, it can then be written as:

$$\mathbf{A} = \mathbf{a}_1\wedge\mathbf{a}_2\wedge\cdots\wedge\mathbf{a}_k \tag{1.4.8}$$

Then the reverse of $\mathbf{A}$ can be defined as:

$$\begin{aligned}\widetilde{\mathbf{A}} &= \mathbf{a}_k\wedge\mathbf{a}_{k-1}\wedge\cdots\wedge\mathbf{a}_1\\ &= (-1)^{k(k-1)/2}(\mathbf{a}_1\wedge\mathbf{a}_2\wedge\cdots\wedge\mathbf{a}_k)\\ &= (-1)^{k(k-1)/2}\mathbf{A}\end{aligned} \tag{1.4.9}$$

Note the reverse operation does not change the grade of blade. Finally, the squared norm of a *k*-blade $\mathbf{A}$ is defined as:

$$\begin{aligned}\|\mathbf{A}\|^2 &= \mathbf{A}\cdot\widetilde{\mathbf{A}}\\ &= (\mathbf{a}_1\wedge\mathbf{a}_2\wedge\cdots\wedge\mathbf{a}_k)\cdot(\mathbf{a}_k\wedge\mathbf{a}_{k-1}\wedge\cdots\wedge\mathbf{a}_1)\end{aligned} \tag{1.4.10}$$

Note that the result is a scalar. As an example, let us compute the norm of the blade $\mathbf{e}_{124}$ in the Euclidean vector space $\mathbf{R}^{4,0,0}$:

$$\|\mathbf{e}_{124}\|^2 = \mathbf{e}_{124}\cdot\widetilde{\mathbf{e}_{124}} \tag{1.4.11}$$

By application the definition of the reverse, we get:

$$\|\mathbf{e}_{124}\|^2 = \mathbf{e}_{124}\cdot\mathbf{e}_{421} \tag{1.4.12}$$

Then, applying the inner product definition between two blades results in:

$$\begin{aligned}
\|\mathbf{e}_{124}\|^2 &= \mathbf{e}_{124} \cdot \mathbf{e}_{421} \\
&= \mathbf{e}_1 \cdot \Big(\mathbf{e}_2 \cdot \Big(\mathbf{e}_4 \cdot (\mathbf{e}_4 \wedge \mathbf{e}_2 \wedge \mathbf{e}_1) \Big) \Big) \\
&= \mathbf{e}_1 \cdot \Big(\mathbf{e}_2 \cdot \Big((\mathbf{e}_4 \cdot \mathbf{e}_4)\mathbf{e}_2 \wedge \mathbf{e}_1) - (\mathbf{e}_4 \cdot \mathbf{e}_2)\mathbf{e}_4 \wedge \mathbf{e}_1 \\
&\quad + (\mathbf{e}_4 \cdot \mathbf{e}_1)\mathbf{e}_4 \wedge \mathbf{e}_2) \Big) \Big) \\
&= \mathbf{e}_1 \cdot \Big(\mathbf{e}_2 \cdot \Big(\mathbf{e}_2 \wedge \mathbf{e}_1 \Big) \Big)
\end{aligned} \tag{1.4.13}$$

By reitering the same algorithm, we get:

$$\begin{aligned}
\|\mathbf{e}_{124}\|^2 &= \mathbf{e}_1 \cdot \Big(\mathbf{e}_2 \cdot \Big(\mathbf{e}_2 \wedge \mathbf{e}_1 \Big) \Big) \\
&= \mathbf{e}_1 \cdot \Big((\mathbf{e}_2 \cdot \mathbf{e}_2)\mathbf{e}_1 - (\mathbf{e}_2 \cdot \mathbf{e}_1)\mathbf{e}_2 \Big) \Big) \\
&= \Big(\mathbf{e}_1 \cdot \mathbf{e}_1 \Big) \\
&= 1
\end{aligned} \tag{1.4.14}$$

1.5 Geometric product and Geometric Algebra

Up to now, we introduced the Grassmann's outer product as well as the metric vector space $\mathbb{R}^{p,q,r}$ and the inner product between blades. We are now left to define the Geometric Algebra $\mathbb{G}_{p,q,r}$ ($d = p + q + r$). We use the vector space $\mathbb{R}^{p,q,r}$ along with a geometric product according to the multiplication rule:

$$\mathbf{e}_i \mathbf{e}_j = \mathbf{e}_i \cdot \mathbf{e}_j + \mathbf{e}_i \wedge \mathbf{e}_j \quad i, j = 1, \cdots, d \tag{1.5.1}$$

The geometric product between two vectors is a multivector and is the sum of a scalar and bivector.

It is convenient for the following to use the notation of the grade projection of a multivector $\mathbf{a}$ as $\langle \mathbf{a} \rangle_k$. This denotes the grade k part of $\mathbf{a}$. Then, the geometric product of two vectors $\mathbf{a}$ and $\mathbf{b}$ can be defined as:

$$\mathbf{ab} = \langle \mathbf{ab} \rangle_0 + \langle \mathbf{ab} \rangle_2 \tag{1.5.2}$$

Note that this definition only holds when $\mathbf{a}$ and $\mathbf{b}$ are vectors.

The properties of this geometric product are as follows:

Associativity	$\mathbf{a}(\mathbf{bc}) = (\mathbf{ab})\mathbf{c}$	$\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^d$
Distributivity over +	$\mathbf{a}(\mathbf{b} + \mathbf{c}) = (\mathbf{ab}) + (\mathbf{ac})$	$\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^d$
Linearity	$\mathbf{a}(r\mathbf{b} + s\mathbf{c}) = r(\mathbf{ab}) + s(\mathbf{ac})$	$\forall \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^d, \forall r, s \in \mathbb{R}$

Note that this product is not necessarily commutative. Below are some geometric products that are worth mentioning when we use orthogonal basis vectors:

$$\begin{aligned} \mathbf{e}_i\mathbf{e}_i &= \mathbf{e}_i \cdot \mathbf{e}_i \\ \mathbf{e}_i\mathbf{e}_j &= \mathbf{e}_i \wedge \mathbf{e}_j = \mathbf{e}_{ij} \qquad \text{for } i \neq j \\ \mathbf{e}_i\mathbf{e}_j &= -\mathbf{e}_j\mathbf{e}_i \qquad \text{for } i \neq j \end{aligned} \tag{1.5.3}$$

Note that these results are verified for any metric vector space $\mathbb{R}^{p,q,r}$.

Commutator product

Formula 1.5.3 slightly changes when one considers the geometric product between a bivector A and a k-vector ($k > 1$) B:

$$AB = A \cdot B + A \wedge B + A \times B \tag{1.5.4}$$

Where $A \times B$ denotes the commutator product between A and B.

1.6 Complex numbers and Hamilton algebra included

In the Geometric Algebra $\mathbb{G}_{2,0,0}$, the only bivector of this algebra raised to the power of 2 is defined as:

$$(\mathbf{e}_{12})^2 = \mathbf{e}_{12}\mathbf{e}_{12} = -1 \tag{1.6.1}$$

Furthermore, let us consider the even components of this algebra. The basis vectors associated is:

$$\mathbf{1}, \mathbf{e}_{12} \tag{1.6.2}$$

A multivector A can be expressed as:

$$A = a_r + a_i\mathbf{e}_{12} \tag{1.6.3}$$

And the geometric product between two multivectors A and B are determined as:

$$\begin{aligned} AB &= (a_r + a_i\mathbf{e}_{12})(b_r + b_i\mathbf{e}_{12}) \\ &= (a_r b_r - a_i b_i) + (a_r a_i + a_i a_r)\mathbf{e}_{12} \end{aligned} \tag{1.6.4}$$

The result is also part of the even components of $\mathbb{G}_{2,0,0}$. This results in a sub-algebra of $\mathbb{G}_{2,0,0}$ and this sub-algebra, as an algebra, is isomorphic to the complex number $\mathbb{C}$. Indeed, $\mathbf{e}_{12}$ has the properties of $\mathbf{i}$ in $\mathbb{C}$.

To go further, when one consider the Geometric Algebra $\mathbb{G}_{3,0,0}$ (also noted $\mathbb{G}_3$), then the geometric product of bivectors is:

$$\begin{aligned}(\mathbf{e}_1\mathbf{e}_2)(\mathbf{e}_1\mathbf{e}_2) &= -1\\ (\mathbf{e}_1\mathbf{e}_3)(\mathbf{e}_1\mathbf{e}_3) &= -1\\ (\mathbf{e}_2\mathbf{e}_3)(\mathbf{e}_2\mathbf{e}_3) &= -1\end{aligned} \tag{1.6.5}$$

We can also compute the geometric products:

$$\begin{aligned}(\mathbf{e}_{23}\mathbf{e}_{31}) &= -\mathbf{e}_{12}\\ (\mathbf{e}_{31}\mathbf{e}_{12}) &= -\mathbf{e}_{23}\\ (\mathbf{e}_{31}\mathbf{e}_{12}) &= -\mathbf{e}_{13}\end{aligned} \tag{1.6.6}$$

Thus, the extraction of the even sub-algebra of $\mathbb{G}_{3,0,0}$ with the basis:

$$\mathbf{1}, -\mathbf{e}_{12}, -\mathbf{e}_{13}, -\mathbf{e}_{23} \tag{1.6.7}$$

is isomorphic to the quaternions $\mathbb{H}$ with the basis:

$$\mathbf{1}, \mathbf{i}, \mathbf{j}, \mathbf{k} \tag{1.6.8}$$

1.7 Pauli matrices included

We stay with the Geometric Algebra $\mathbb{G}_3$ ($\mathbb{G}_{3,0,0}$) and the same basis as before. Let us of flush out our list of properties of $\mathbb{G}_3$ by rewriting:

$$\begin{aligned}2\mathbf{e}_{123}\mathbf{e}_3 &= 2\mathbf{e}_{12}\\ &= (\mathbf{e}_1\mathbf{e}_2 - \mathbf{e}_2\mathbf{e}_1)\end{aligned} \tag{1.7.1}$$

Repeating the same computation for the remaining vectors, we get:

$$\begin{aligned}2\mathbf{e}_{123}\mathbf{e}_2 &= 2\mathbf{e}_{31}\\ &= (\mathbf{e}_3\mathbf{e}_1 - \mathbf{e}_1\mathbf{e}_3)\end{aligned} \tag{1.7.2}$$

Finally:

$$\begin{aligned} 2\mathbf{e}_{123}\mathbf{e}_1 &= 2\mathbf{e}_{23} \\ &= (\mathbf{e}_2\mathbf{e}_3 - \mathbf{e}_3\mathbf{e}_2) \end{aligned} \tag{1.7.3}$$

Using the pseudoscalar $\mathbf{I} = \mathbf{e}_{123}$ of $\mathbb{G}_3$, we obtain the three equalities:

$$\begin{aligned} \mathbf{I}\mathbf{e}_1 &= \mathbf{e}_{23} \\ \mathbf{I}\mathbf{e}_2 &= \mathbf{e}_{31} \\ \mathbf{I}\mathbf{e}_3 &= \mathbf{e}_{12} \end{aligned} \tag{1.7.4}$$

If we now consider two basis vector $\mathbf{e}_i$ and $\mathbf{e}_j$, the inner product $\mathbf{e}_i \cdot \mathbf{e}_j$ can be defined as follows:

$$\mathbf{e}_i \cdot \mathbf{e}_j = \delta_{i,j} \tag{1.7.5}$$

where, $\delta_{i,j}$ is the Kronecker symbol, for $i = j, \delta_{i,j} = 1$, and if $i \neq j, \delta_{i,j} = 0$. Furthermore, for three equalities previously defined, we get:

$$\mathbf{e}_i \wedge \mathbf{e}_j = \mathbf{I}\epsilon_{ijk}\mathbf{e}_k \tag{1.7.6}$$

Where ϵ_{ijk} denotes the Levi-Civita symbol in 3-dimensional space defined as:

$$\epsilon_{ijk} = \begin{cases} +1 & \text{if } (i,j,k) \text{ is } (1,2,3) \text{ or } (2,3,1) \text{ or } (3,1,2) \\ -1 & \text{if } (i,j,k) \text{ is } (3,2,1) \text{ or } (1,3,2) \text{ or } (2,1,3) \\ 0 & \text{for } i = j, \text{ or } j = k, \text{ or } k = i \end{cases} \tag{1.7.7}$$

Finally, the geometric product $\mathbf{e}_i\mathbf{e}_j$ yields:

$$\mathbf{e}_i\mathbf{e}_j = \delta_{i,j} + \mathbf{I}\epsilon_{ijk}\mathbf{e}_k \tag{1.7.8}$$

If we rename all the $\mathbf{e}$ into σ in the equations previously defined, we verify that σ satisfy the Pauli matrices properties, see [47].

1.8 Inverse and dual

It is possible to invert a blade using the geometric product. The definition of the inverse of a k—blade $\mathbf{a}$ is:

$$\mathbf{A}^{-1} = \frac{\widetilde{\mathbf{A}}}{\|\mathbf{A}\|^2} \tag{1.8.1}$$

Let us check this definition with a very simple example, namely if $\mathbf{a}$ is a vector then:

$$\mathbf{a}^{-1} = \frac{\mathbf{a}}{\mathbf{a}\cdot\mathbf{a}} \tag{1.8.2}$$

Then, if $\mathbf{a}\cdot\mathbf{a} \neq 0$, the computation of $\mathbf{a}^{-1}\mathbf{a}$ results in:

$$\begin{aligned} \mathbf{a}^{-1}\mathbf{a} &= \frac{1}{\mathbf{a}\cdot\mathbf{a}}\mathbf{a}\mathbf{a} \\ &= \frac{1}{\mathbf{a}\cdot\mathbf{a}}(\mathbf{a}\cdot\mathbf{a} + \mathbf{a}\wedge\mathbf{a}) \\ &= \frac{1}{\mathbf{a}\cdot\mathbf{a}}(\mathbf{a}\cdot\mathbf{a} + 0) \\ &= 1 \end{aligned} \tag{1.8.3}$$

Using the inverse, it is possible to define the dual of a k-blade $\mathbf{A}$. It is denoted as $\mathbf{A}^*$ and defined as:

$$\mathbf{A}^* = \mathbf{A}\mathbf{I}^{-1} = \frac{\mathbf{A}\widetilde{\mathbf{I}}}{\mathbf{I}\cdot\widetilde{\mathbf{I}}} \tag{1.8.4}$$

Where, $\mathbf{I} = \mathbf{e}_{12\cdots d}$ is the pseudoscalar of the considered Geometric Algebra. Intuitively, this corresponds to the basis blades contained in $\mathbf{A}$ but not contained in the pseudoscalar $\mathbf{I}$

As an example, let us compute the dual of the vector $\mathbf{a} = 2\mathbf{e_1} + 3\mathbf{e_2}$ in $\mathbb{G}_3$. Tn this algebra, $\mathbf{I} = \mathbf{e}_{123}$ and:

$$\begin{aligned} \mathbf{I}\cdot\widetilde{\mathbf{I}} &= \mathbf{e}_{123}\cdot\widetilde{\mathbf{e}_{123}} \\ &= \mathbf{e}_{123}\cdot\mathbf{e}_{321} \\ &= 1 \end{aligned} \tag{1.8.5}$$

Using the definition of the geometric product, the computation of the dual results in:

$$\mathbf{a}^* = (2\mathbf{e_1} + 3\mathbf{e_2})\cdot\mathbf{e}_{321} \tag{1.8.6}$$

From the linearity of the inner product, we get:

$$\begin{aligned} \mathbf{a}^* &= (2\mathbf{e_1}\cdot\mathbf{e}_{321}) + (3\mathbf{e_2}\cdot\mathbf{e}_{321}) \\ &= -2\mathbf{e}_{23} + 3\mathbf{e}_{13} \end{aligned} \tag{1.8.7}$$

From [48], we have the useful duality between outer and inner products of non-scalar blades $\mathbf{A}$ and $\mathbf{B}$ in Geometric Algebra:

$$\begin{aligned}(\mathbf{A} \wedge \mathbf{B})^* &= \mathbf{A} \cdot \mathbf{B}^*, \\ \mathbf{A} \wedge (\mathbf{B}^*) &= (\mathbf{A} \cdot \mathbf{B})^*, \\ \mathbf{A} \wedge (\mathbf{B}\mathbf{I}) &= (\mathbf{A} \cdot \mathbf{B})\mathbf{I},\end{aligned} \tag{1.8.8}$$

which indicates the two major properties, namely:

$$\begin{cases} \mathbf{A} \wedge \mathbf{B} = 0 & \Leftrightarrow \quad \mathbf{A} \cdot \mathbf{B}^* = 0, \\ \mathbf{A} \cdot \mathbf{B} = 0 & \Leftrightarrow \quad \mathbf{A} \wedge \mathbf{B}^* = 0. \end{cases} \tag{1.8.9}$$

1.8.1 Intersections

Let us consider two blades corresponding to dual objects $\mathbf{A}^*$ and $\mathbf{B}^*$. Assuming that the two objects are linearly independent, i.e., $\mathbf{A}^*$ and $\mathbf{B}^*$ are linearly independent, we consider the outer product $\mathbf{C}^*$ of these two objects:

$$\mathbf{C}^* = \mathbf{A}^* \wedge \mathbf{B}^*. \tag{1.8.10}$$

If an entity $\mathbf{x}$ lies on $\mathbf{C}^*$, then

$$\mathbf{x} \cdot \mathbf{C}^* = \mathbf{x} \cdot (\mathbf{A}^* \wedge \mathbf{B}^*) = 0. \tag{1.8.11}$$

The inner product definition develops (1.8.11) as follows:

$$\mathbf{x} \cdot \mathbf{C}^* = (\mathbf{x} \cdot \mathbf{A}^*)\mathbf{B}^* - (\mathbf{x} \cdot \mathbf{B}^*)\mathbf{a}^* = 0. \tag{1.8.12}$$

The assumption of linear independence between $\mathbf{A}^*$ and $\mathbf{B}^*$ implies that (1.8.12) holds if and only if $\mathbf{x} \cdot \mathbf{A}^* = 0$ and $\mathbf{x} \cdot \mathbf{B}^* = 0$, i.e. the entity $\mathbf{x}$ lies on both entities represented by $\mathbf{A}^*$ and $\mathbf{B}^*$. Thus, $\mathbf{C}^* = \mathbf{A}^* \wedge \mathbf{B}^*$ represents the intersection of the linearly independent objects $\mathbf{A}^*$ and $\mathbf{B}^*$, and an entity $\mathbf{x}$ lies on this intersection if and only if $\mathbf{x} \cdot \mathbf{C}^* = 0$. This is a major property of Geometric Algebra that is very useful to define intersections between geometric objects.

1.9 Products from geometric product

All the products of Geometric Algebra can be defined from the geometric product. Below are some products that are worth mentioning and defining from the geometric product between a k-blade $\mathbf{A}$ and a l-blade $\mathbf{B}$ (more explanations at [22]).

Left contraction

It is noted as $\rfloor$ and defined as:

$$\mathbf{A} \rfloor \mathbf{B} = \begin{cases} \langle \mathbf{AB} \rangle_{l-k} & \text{if } l \geq k \\ 0 & \text{else} \end{cases} \tag{1.9.1}$$

Right contraction

It is noted as $\lfloor$ and defined as:

$$\mathbf{A} \lfloor \mathbf{B} = \begin{cases} \langle \mathbf{AB} \rangle_{k-l} & \text{if } l \leq k \\ 0 & \text{else} \end{cases} \tag{1.9.2}$$

Scalar product

It is noted as a $*$ and defined as:

$$\mathbf{A} * \mathbf{B} = \langle \mathbf{AB} \rangle_0 \tag{1.9.3}$$

Or equivalently as:

$$\mathbf{A} * \mathbf{B} = \begin{cases} \langle \mathbf{AB} \rangle_0 & \text{if } k = l \\ 0 & \text{else} \end{cases} \tag{1.9.4}$$

Fat dot product

It is noted as $\bullet$ and defined as:

$$\mathbf{A} \bullet \mathbf{B} = \langle \mathbf{AB} \rangle_{|k-l|} \tag{1.9.5}$$

Hestenes inner product

It is noted as a $\bullet_H$

$$\mathbf{A} \bullet_H \mathbf{B} = \begin{cases} \langle \mathbf{AB} \rangle_{|k-l|} & \text{if } k, l \neq 0 \\ 0 & \text{else} \end{cases} \tag{1.9.6}$$

Below is a useful relation between these inner products, see [23]:

$$\mathbf{A} \bullet \mathbf{B} + \mathbf{A} * \mathbf{B} = \mathbf{A} \lfloor \mathbf{B} + \mathbf{A} \rfloor \mathbf{B} \tag{1.9.7}$$

Some other useful relations can be extracted for different configurations (also [23]) of k and l, typically:

For $k < l$	$\mathbf{A} \rfloor \mathbf{B} = \mathbf{A} \bullet \mathbf{B}$ $\mathbf{A} \lfloor \mathbf{B} = 0$ $\mathbf{A} * \mathbf{B} = 0$
For $k = l$	$\mathbf{A} \rfloor \mathbf{B} = \mathbf{A} * \mathbf{B}$ $\mathbf{A} \lfloor \mathbf{B} = \mathbf{A} * \mathbf{B}$ $\mathbf{A} * \mathbf{B} = \mathbf{A} \bullet \mathbf{B}$
For $k > l$	$\mathbf{A} \rfloor \mathbf{B} = 0$ $\mathbf{A} \lfloor \mathbf{B} = \mathbf{A} \bullet \mathbf{B}$ $\mathbf{A} * \mathbf{B} = 0$

For the following, the use of the dot $\cdot$ for the inner product will denote the computation of the **fat dot product**, see Equation 1.9.5.

1.10 Transformations

1.10.1 Reflections and rotations

The projection of one vector $\mathbf{x}$ onto a another vector $\mathbf{a}$ using the inner product is defined as follows:

$$\mathbf{x}_{\parallel} = (\mathbf{x} \cdot \mathbf{a})\mathbf{a}^{-1} \tag{1.10.1}$$

The rejection of the same vector onto $\mathbf{a}$ is:

$$\mathbf{x}_{\perp} = \mathbf{x} - \mathbf{x}_{\parallel} = (\mathbf{x} \wedge \mathbf{a})\mathbf{a}^{-1} \tag{1.10.2}$$

These definitions of projections and rejection are the support to define reflections and so more advanced transformations. The computation of the reflection of $\mathbf{x}$ onto the vector $\mathbf{a}$ merely consists in computing $\mathbf{x}_{\parallel} - \mathbf{x}_{\perp}$. This yields:

$$\mathbf{x}_{\parallel} - \mathbf{x}_{\perp} = (\mathbf{x} \cdot \mathbf{a})\mathbf{a}^{-1} - (\mathbf{x} \wedge \mathbf{a})\mathbf{a}^{-1} \tag{1.10.3}$$

Due to the anti-commutativity of the outer product, the commutativity of the inner product over the vectors and the distributivity of the geometric product, we get:

$$\mathbf{x}_{\parallel} - \mathbf{x}_{\perp} = (\mathbf{a} \cdot \mathbf{x} + \mathbf{x} \wedge \mathbf{a})\mathbf{a}^{-1} \tag{1.10.4}$$

Finally, this yields:

$$\mathbf{x}_{\parallel} - \mathbf{x}_{\perp} = \mathbf{a}\mathbf{x}\mathbf{a}^{-1} \tag{1.10.5}$$

This formula is the definition of the reflection. Now if we choose another vector $\mathbf{b}$ (see Figure 1.5) and if we compute the reflection of the reflected entity $\mathbf{a}\mathbf{x}\mathbf{a}^{-1}$, the result is as follows:

$$\mathbf{b}(\mathbf{a}\mathbf{x}\mathbf{a}^{-1})\mathbf{b}^{-1} \tag{1.10.6}$$

Since the geometric product is associative, the above formula can be rewritten as follows:

$$(\mathbf{b}\mathbf{a})\mathbf{x}(\mathbf{a}^{-1}\mathbf{b}^{-1}) = (\mathbf{b}\mathbf{a})\mathbf{x}(\mathbf{b}\mathbf{a})^{-1} \tag{1.10.7}$$

This results in the rotation of $\mathbf{x}$ along the vector normal to the plane spanned by $\mathbf{a}$ and $\mathbf{b}$ and the rotation angle is twice (mod 2π) the angle between $\mathbf{a}$ and $\mathbf{b}$. More generally, the product of an even number of reflections is a rotation, leading to the definition of a versor.

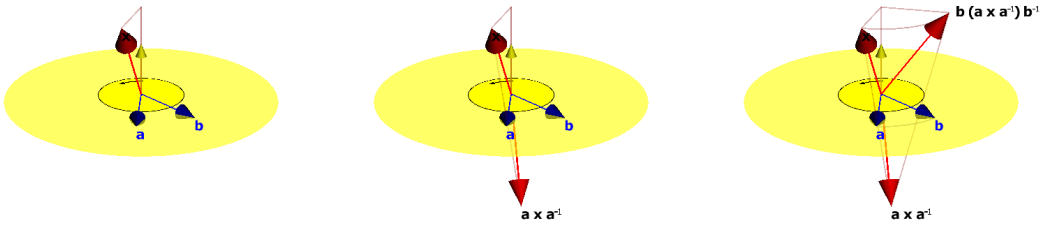


FIGURE 1.5: Reflection to rotation (from gaviewer [35])

1.10.2 Versors

In a Geometric Algebra $\mathbb{G}_{p,q,r}$, a versor is defined as a product expression composed of invertible vectors, it is factorizable under the geometric product as follows:

$$V = \mathbf{v}_1\mathbf{v}_2\cdots\mathbf{v}_k \tag{1.10.8}$$

The conjugate of a versor is defined as:

$$\begin{aligned} V^{\dagger} &= (-1)^k V^{-1} \\ &= (-1)^k \mathbf{v}_k^{-1}\mathbf{v}_{k-1}^{-1}\cdots\mathbf{v}_1^{-1} \end{aligned} \tag{1.10.9}$$

Transformations of geometric objects can be performed as follows:

$$V(\cdots)V^{\dagger} \tag{1.10.10}$$

Note that a property of this product is that it preserves the outer product, namely:

$$\mathbf{a}'_1 \wedge \mathbf{a}'_2 \wedge \cdots \wedge \mathbf{a}'_m = (-1)^k V(\mathbf{a}'_1 \wedge \mathbf{a}'_2 \wedge \cdots \wedge \mathbf{a}'_m)V^\dagger \tag{1.10.11}$$

Some examples of transformations using versors will be shown in the next section.

1.11 Conformal Geometric Algebra

Up to now, we defined the main operators of Geometric Algebra. This section introduces a powerful and intuitive application called Conformal Geometric Algebra, noted in short CGA.

1.11.1 basis and metric

CGA of $\mathbb{R}^3$ is $\mathbb{G}_{4,1}$ thus a 5-dimensional vector space. The base vectors of the space are basically divided into three groups: $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ (corresponding to the Euclidean vectors in $\mathbb{R}^3$), but also $\{\mathbf{e}_o\}$, and $\{\mathbf{e}_\infty\}$. The inner products between them are defined in Table 1.2.

TABLE 1.2: Inner product between QCGA basis vectors.

	$\mathbf{e_0}$	$\mathbf{e_1}$	$\mathbf{e_2}$	$\mathbf{e_3}$	$\mathbf{e}_\infty$
$\mathbf{e_0}$	0	0	0	0	-1
$\mathbf{e_1}$	0	1	0	0	0
$\mathbf{e_2}$	0	0	1	0	0
$\mathbf{e_3}$	0	0	0	1	0
$\mathbf{e}_\infty$	-1	0	0	0	0

The basis previously defined was generated with the Geometric Algebra $\mathbb{G}_{4,1}$. A generator of this Geometric Algebra can be first the Euclidean basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ along with the basis vector $\mathbf{e}_+$ which squared to $+1$ and the basis vector $\mathbf{e}_-$ which squared to -1. This basis corresponds to the diagonal metric matrix. The transformation from the defined basis to the diagonal metric basis can be defined as follows:

$$\begin{cases} \mathbf{e}_\infty &= \mathbf{e}_+ + \mathbf{e}_- \\ \mathbf{e}_0 &= \frac{1}{2}(\mathbf{e}_- - \mathbf{e}_+) \end{cases} \tag{1.11.1}$$

The dual definition of Equation 1.8.4 applied to this Geometric Algebra are now computed. First, the pseudo-scalar is the 5-blade:

$$\mathbf{I} = \mathbf{e}_0 \wedge \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3 \wedge \mathbf{e}_\infty \tag{1.11.2}$$

Its inverse:

$$\mathbf{I}^{-1} = -\mathbf{I} \tag{1.11.3}$$

The dual a vector $\mathbf{a}$ is thus:

$$\mathbf{a}^* = -\mathbf{a}\mathbf{I} \tag{1.11.4}$$

1.11.2 Point of CGA

In this space, it is possible to define a point whose Euclidean vector is $\mathbf{x}_\epsilon = x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3$ as:

$$\mathbf{x} = \mathbf{e}_0 + \mathbf{x}_\epsilon + \frac{1}{2}\left\|\mathbf{x}_\epsilon\right\|^2 \mathbf{e}_\infty \tag{1.11.5}$$

The major property is the inner product between two points $\mathbf{x}_1$ and $\mathbf{x}_2$.

$$\mathbf{x}_1 \cdot \mathbf{x}_2 = \frac{1}{2}\left\|\mathbf{x}_{\epsilon 2} - \mathbf{x}_{\epsilon 1}\right\|^2 \tag{1.11.6}$$

This corresponds to squared Euclidean distance between the two points. This is very useful to define objects.

1.11.3 CGA objects

Firstly the objects defined in CGA include circles, lines, spheres, planes and each defined object can be intuitively defined. Firstly a circle is merely constructed as the outer product of 3 points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ as:

$$\mathbf{C} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_3 \tag{1.11.7}$$

Then sending one of the points of $\mathbf{c}$ at infinity flatten the circle and this yields a line $\mathbf{l}$. The definition of the line in CGA is not more complicated than replacing one of the point in $\mathbf{c}$ by the point at infinity $\mathbf{e}_\infty$, namely:

$$\mathbf{L} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{e}_\infty \tag{1.11.8}$$

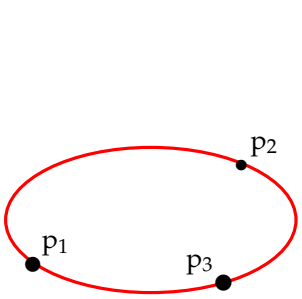
Then, by incrementing the dimensions of objects by 1, a sphere $\mathbf{S}$ can be obtained by the outer product of four points.

$$\mathbf{S} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_3 \wedge \mathbf{p}_4 \tag{1.11.9}$$

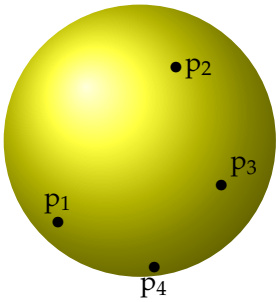
Finally, sending one of the point at infinity will flatten the sphere and results in a plane $\boldsymbol{\pi}$ defined as:

$$\boldsymbol{\Pi} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_3 \wedge \mathbf{e}_\infty \tag{1.11.10}$$

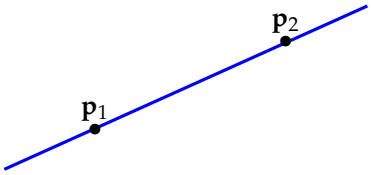
This leads to very efficient and intuitive way to define geometric objects using Geometric Algebra. The Figure 1.6 describes the definition of some geometric objects from control points with CGA.



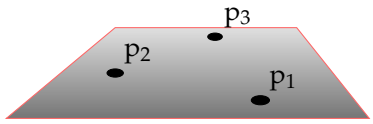
Circle: $\mathbf{C} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_3$



Sphere: $\mathbf{S} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_3 \wedge \mathbf{p}_4$



Line: $\mathbf{L} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{e}_\infty$



Plane $\boldsymbol{\Pi} = \mathbf{p}_1 \wedge \mathbf{p}_2 \wedge \mathbf{p}_3 \wedge \mathbf{e}_\infty$

FIGURE 1.6: definition of some geometric primitives from control points

Note that a sphere $\mathbf{S}$ may also be dually represented using its radius r and centre point $\mathbf{x}_c$ as follows:

$$\mathbf{S}^* = \mathbf{x}_c - \frac{1}{2}\mathrm{r}^2\mathbf{e}_\infty, \tag{1.11.11}$$

Then, a point $\mathbf{x}$ lies on the dual sphere if and only if $\mathbf{x} \cdot \mathbf{s}^* = 0$. By developing this latter equation, we find:

$$\begin{aligned} & \mathbf{x} \cdot \mathbf{S}^* && = 0 \\ \Leftrightarrow \quad & \mathbf{x} \cdot \mathbf{x}_c - \tfrac{1}{2}\mathrm{r}^2\mathbf{x} \cdot \mathbf{e}_\infty && = 0 \\ \Leftrightarrow \quad & \mathbf{x} \cdot \mathbf{x}_c + \tfrac{1}{2}\mathrm{r}^2 && = 0 \\ \Leftrightarrow \quad & -\tfrac{1}{2}\left\|\mathbf{x}_\epsilon - \mathbf{x}_{c\epsilon}\right\|^2 + \tfrac{1}{2}\mathrm{r}^2 && = 0 \\ \Leftrightarrow \quad & \left\|\mathbf{x}_\epsilon - \mathbf{x}_{c\epsilon}\right\|^2 && = \mathrm{r}^2 \end{aligned} \tag{1.11.12}$$

Thus, corresponding to the equation of a sphere whose centre point is $\mathbf{x}_c$ and radius r.

A circle can also be obtained by the intersection of two spheres. Finally, a point pair can be obtained as the outer product of two points.

A plane can also be obtained dually as.

$$\boldsymbol{\Pi}^* = \mathbf{n} + \mathrm{h}\mathbf{e}_\infty \tag{1.11.13}$$

In a similar way, a point $\mathbf{x}$ lies on the dual plane if and only if $\mathbf{x} \cdot \boldsymbol{\Pi}^* = 0$. By developing this latter equation, we find:

$$\begin{aligned} & \mathbf{x} \cdot \boldsymbol{\Pi}^* && = 0 \\ \Leftrightarrow \quad & \mathbf{x} \cdot (\mathbf{n} + \mathrm{h}\mathbf{e}_\infty) && = 0 \\ \Leftrightarrow \quad & \mathbf{x} \cdot \mathbf{n} - \mathrm{h} && = 0 \end{aligned} \tag{1.11.14}$$

The above equation corresponds to the Hessian form of the plane of normal $\mathbf{n}$ with orthogonal distance h from the origin.

1.11.4 Intersection in CGA

The major point with the intersection in CGA is that the algebraic object resulting from the intersection exists even for two objects whose intersection is empty. Furthermore, some properties can even be interpreted from this object. This case is illustrated in the two figures of 1.7.

When we consider two objects that span the whole space, there even exists some very efficient way to know whether one sphere $\mathbf{S}^*$ and object $\mathbf{A}^*$ intersect. This consists in computing the intersection as follows:

$$\mathbf{C} = \mathbf{S}^* \wedge \mathbf{A}^* \tag{1.11.15}$$

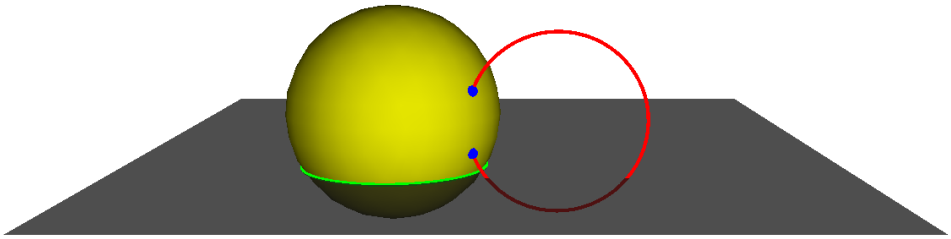
Then check the radius of the circle using:

$$\left(\frac{\mathbf{C}}{\mathbf{C} \cdot \mathbf{e}_\infty}\right)^2 \overset{?}{\geq} 0 \tag{1.11.16}$$

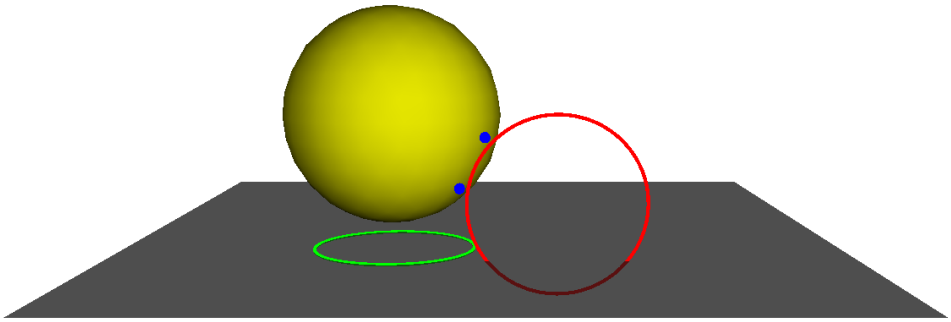
If the radius is higher than zero then this means that the intersection is real. This is a very simple and efficient way to know if two objects intersect.
When we consider two objects that do not span the whole space, the dual in (1.11.15) has to be defined with respect to the spanned subspace. For example, in the space, the join of two parallel lines or two intersecting lines $\mathbf{L}_1$,$\mathbf{L}_2$ is the plane $\mathbf{\Pi}$ that is spanned by these two lines. Then, the definition of the dual of the line $\mathbf{L}_1$ (and $\mathbf{L}_2$) is taken with respect to the spanned plane as follows:

$$\mathbf{L}_1^* = \mathbf{L}_1 \Big(join(\mathbf{L}_1, \mathbf{L}_2) \Big)^{-1}, \tag{1.11.17}$$

and not the pseudoscalar $\mathbf{I}$. For more details, please refer to [23].



(A)



(B)

FIGURE 1.7: Two intersection are computed, one between the circle and the sphere and the other between the sphere and the plane. The intersections exist in (A) but do not in (B). But in both cases, this results in a Geometric Algebra entity, and we can interpret the results.

1.11.5 Transformations

The last section shows definition of a versor. Table 1.3 defines and summarizes all the transformations that can be performed using CGA versors.

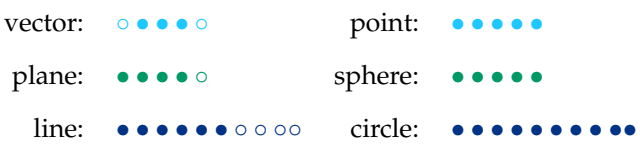
TABLE 1.3: Transformations handled in CGA

reflector	reflexion by plane $\mathbf{\Pi}$ with unit normal $\mathbf{n}$	$N = \mathbf{\Pi}$ $N^\dagger = -\mathbf{\Pi}$
translator	translation by $\mathbf{t}$	$T_\mathbf{t} = 1 - \frac{1}{2}\mathbf{t}e_\infty$ $T_\mathbf{t}^\dagger = 1 + \frac{1}{2}\mathbf{t}e_\infty$
rotor	rotation in the plane $\boldsymbol{\tau}$ of angle θ	$R = \cos\frac{\theta}{2} + \boldsymbol{\tau}\sin\frac{\theta}{2}$ $R^\dagger = \cos\frac{\theta}{2} - \boldsymbol{\tau}\sin\frac{\Omega}{2}$
dilator	dilation by $e^{\frac{\gamma}{2}}$ around the origin	$D = \cosh\frac{\gamma}{2} + \mathbf{e_{0\infty}}\sin\frac{\gamma}{2}$ $D^\dagger = \cosh\frac{\gamma}{2} - \mathbf{e_{0\infty}}\sin\frac{\gamma}{2}$
motor	rotation R followed by translation $\mathbf{t}$	$M = T_\mathbf{t}R$ $M^\dagger = R^{-1}T_\mathbf{t}^{-1}$

1.11.6 Properties of CGA objects

As a transition with the next chapter, let us look further to the multivector structure of the geometric objects defined previously. Let us consider a multivector $\mathbf{a}$ in $\mathbb{G}_{4,1}$. This multivector may have up to 32 non-zero elements and may be organized as follows:

However, the multivector representing geometric objects are much sparser. Below is the real occupation of the more common geometric objects in CGA:



We remark the lengthiest object has only 10 elements over 32 for a full multivector. It is also worth mentioning that each object has a fixed grade, they are called homogeneous multivectors. Furthermore, we observe that within the subspace of grade k, all the components of the homogeneous multivector representing the object of grade k may be non-null. For example, the grade of a circle is 3 and all the basis blades of grade 3 may be non-null. A similar observation can be done for spheres and points.

Part I

Implementation of Geometric Algebra

Chapter 2

Design

2.1 Introduction

There exist various implementations of Geometric Algebra, and most of the programming languages or famous mathematical frameworks can find a Geometric Algebra library well suited for a comfortable use. However, very few of these libraries can handle computations in high dimensional vector spaces, i.e. dimension where the memory storage of the multivector becomes problematic, usually dimension 12. In addition, more and more Geometric Algebra applications focus on high dimensional spaces. It started with Conformal Geometric Algebra (CGA) which is certainly one of the most studied [73]. CGA is built from 5 dimensional vector space, and includes $2^5 = 32$ basis vectors. Although some people consider 32 components per multivector to be already high, some studies are conducted to explore even higher dimensions. Easter and Hitzer [27] represent some quartics and quadrics 3-d shapes using a double conformal geometry of $\mathbb{R}^3$. Extending this process to a triple conformal geometry would lead to a 15—dimensional algebra containing $2^{15} = 32,768$ elements. For such geometry, the memory requirement for optimized libraries explodes far beyond consumer grade hardware capabilities. More regular approaches lead to very long processing time.

2.1.1 State of the art

In the wide range of Geometric Algebra implementations, we find first specialized software dedicated to 3D visualization for scientific and/or educational aims. For performing some tests using Matlab, Stephen Mann and Leo Dorst et al. developed a Geometric Algebra package [64]. GAViewer [35] developed by Fontijne et al. allows interactive representation and manipulation of objects in the Conformal Geometric Algebra. CLUCalc [72], conceived and written by Christian Perwass handles more general Geometric Algebras. Some other libraries are computer algebra software specifically designed for symbolic computations, such as GAlgebra [8]

in Python,GALua [71] for Lua, or the library CLIFFORD [2, 1] developed by Rafal Ablamowicz and Bertfried Fauser, based on Maple. Steve Sangwine and Eckhard Hitzer developed a Multivector Toolbox [76] for Matlab. Maxima also finds its Clifford Algebra implementation [74] presented by Dimiter Prodanov. We also find some Geometric Algebra implementation specifically dedicated to specialized Geometric Algebra, these are developed to perform numerical computations. Among these, GluCat [60] is using real matrix representation for Clifford algebra [59] and a dedicated version of fast Fourier transform to improve Clifford product. GluCat was benchmarked and found its performance to be similar to CLU [32]. Versor [14] developed by Pablo Colapinto is a very advanced C++ library. It uses C++ meta-programming techniques like expression templates to define types representing expressions to be computed at compile time. Thus, expressions are computed only when needed to produce efficient code. Finally, Gaalet [78] proposed by Florian Seybold, standing for Geometric Algebra ALgorithms Expression Template, is also a C++ library and uses expression templates and meta-programming techniques.

All these libraries present a comfortable use in their dedicated framework or programming language and are very well suited for experimentations in Geometric Algebra. Some other libraries expressly focus on computation and memory performance for different algebras. These implementations can be classified into two groups, namely code generators and library generators.

2.1.1.1 Code generators

Code generators optimize Geometric Algebra by means of low-level source code generation and some symbolic optimizations in terms of algebra. One of the most advanced program in this category is Gaalop [49, 12] developed by Dietmar Hildenbrand et al. Gaalop, standing for Geometric Algebra Algorithms Optimizer. It supposes that a program is written in CLUCalc. A first compilation stage is performed to produce C++, C++ AMP (Accelerated Massive Parallelism), OpenCL, CUDA, CLUCalc or LaTeX optimized output. Then a second compilation stage is required by the output language compiler to produce the target object. GMac [30, 31] developed by Ahmad Hosny Eid is the other optimized library for Geometric Algebra. GMac stands for Geometric MACro, and is closely coupled with .NET languages like C++, C#, VB.NET, F#, and IronPython. GMac presents advanced conception and implementation. These compilation optimizations produce a very optimized code dedicated for a specific task, however they are not well suited for runtime computation where the operations are not decided in advance. Furthermore, the generated code has to be manually integrated into the application of the user which can become annoying when investigating some algebra equations.

2.1.1.2 Library generators

A more flexible way to use Geometric Algebra consists in generating libraries defined from an algebra specification. In such kind of implementation, the product between some multivectors with defined structure are optimized. This is the case for Gaigen [23, 33] presented by Daniel Fontijne. Gaigen stands for Geometric Algebra Implementation GENerator and can produce C++, C, C# and Java source code which implements a Geometric Algebra with a specified dimension and metric. The Geometric Algebra products are computed at run time using high complexity products. Exotic multivectors (part of non-homogeneous multivectors) are computed with a general class that presents a much lower optimization level. Finally, this generator is limited in terms of vector space dimension. More precisely, the optimizations limit the library to be used for algebra whose vector space dimensions are higher than 10.

All these approaches present some interesting properties, however some improvements can be achieved to make such libraries easier to use, to have better memory requirements and to range over wider dimension spaces for applications of [3, 4] for example. These points are the initial motivation to create the proposed library, called *Garamon*.

2.1.2 Expression of needs

In order to construct a generic, efficient library, we consider the needs to be expressed according to both theoretic and practical criteria as follows:

1. Flexibility:
   (a) Be able to use different metrics and basis without code changes,
   (b) Be able to use simultaneously different algebras without code changes,
2. Computationally efficient:
   (a) Efficient for the computations in a wide range of Geometric Algebra vector space dimensions,
3. Memory efficient
   (a) The consumption of the data structures has to be as low as possible,
   (b) The binary file size also has to be low,
4. Numerically stable
   (a) The implementation has to be numerically robust, including for the change of basis,

5. Portability:

   (a) has to be multi-platform,

   This includes Windows, Linux, and MacOs operating system.

6. Installability:

   (a) The implementation has to be easily installable.

   (b) The number of dependency of the implementation has to be as low as possible.

7. Readability

   (a) has to have a clear and concise structure

   (b) has to be well documented and commented

These needs are our initial motivations to create *Garamon*.

2.1.3 Library generator

Garamon is a C++ template library generator dedicated to Geometric Algebra. The generator itself runs in C++ and generates optimized C++ code. These generated GA libraries are dedicated to fulfilling the needs previously described.

The generated libraries are built from a short configuration file describing the targeted algebra. This configuration file specifies the algebra signature, the name of the basis vectors and some optimization options. This file is restricted to the minimum information such it can be filled very easily.

2.1.3.1 Efficient

The generated GA libraries handles both “low dimensional” (base vector space of dimension roughly up to 10) and “high dimensional”, with a hard-coded limit to dimension 31. The “low dimensional” operations are precomputed, whereas the “high dimension” computations run on a new recursive scheme based on a prefix tree multivector representation. This prefix tree representation presents some very effective optimization in term of time complexity, as well as the property to encode easily the dual of the considered multivector. The transition from “low dimensional” to “high dimensional” is smooth, such that “high dimensional” GA libraries still include some precomputed instructions for some products.

2.1.3.2 User friendly

The generated libraries are dedicated to be very easy to install and to use. They are multi-platform, run and compile with only one dependency, i.e. the header only library `eigen` [41]. Any generated library contains its own dedicated installation file (cmake), as well as a dedicated sample code. The generated libraries handle any arbitrary Geometric Algebra signature, such that the user does not have to care about basis change. The embedded basis change takes a special care about numerical stability. Moreover, since all the generated libraries are identified by a namespace, multiple GA libraries can be used together.

2.2 What to store

One of the main way to meet the memory and efficiency needs 3a and 2a is by a proper use of the multivector data structure.

2.2.1 Requirements

For a d dimensional vector space, the potential amount of information that could be stored to represent fundamental elements of linear algebra (vectors and matrices) strongly differs from the information represented in GA (multivectors). For linear algebra, it is of order $O(d^2)$ whereas for it is of order $O(2^d)$ for GA, as seen in Equation 1.3.12. This difference usually influences their respective implementation. Hence, linear algebra implementations are frequently expressing and storing all the data composing a vector or a matrix (except if they are known to be sparse) whereas GA implementations are mostly trying to only store non-zero elements. More precisely, as seen in the last chapter, most object represented with Geometric Algebra are blades. Indeed, each object occupies most of the elements of the k-vector. Thus, an efficient data structure used for Geometric Algebra has to take advantage of such properties. This means that an efficient multivector data structure has to have efficient storage for each homogeneous multivector. Efficient means:

① fast access time to a homogeneous multivector,

② fast access time to each element of a homogeneous multivector (aligned data),

③ low memory consumption for the storage of multivectors.

2.3 Multivector and linked list

One way to implement these constraints is to use a linked list of non-zero elements as in many Geometric Algebra implementations [34, 78, 14]. This data structure is well suited for low number of elements since the memory is allocated for only non-zero elements. However, this latter data structure presents some major drawbacks. First, the access time to an element is in $\mathcal{O}(n)$ or linear to the number of elements, thus the constraints ① and ② are not fulfilled. Second, for a full multivector, the required memory is twice the price of the data itself (includes data and the pointer).

2.4 Multivector and arrays

In contrast, some other implementations like the code generator of Gaalop [12] use full size static arrays (2^d elements) to store multivectors in the generation process. This structure has the great advantage to have $\mathcal{O}(1)$ access time to one element (constant time), meaning that the both access time to a homogeneous multivector and to an element of a homogeneous multivector are fast. The major drawback of this structure is in terms of memory consumption. For any kind of multivector, 2^d elements will be allocated in memory. For example, if the considered object is a point. The multivector associated to this point in CGA has at most 5 non-null elements over a total of $2^5 = 32$ elements. This means that 27 elements, or roughly 84% of the elements will be useless. In computer graphics or image processing or in digital geometry, one might want to store and use a cloud of points of approximately 10^6 points. In this case, this number of useless zero becomes 27 millions of useless elements which can be costly in terms of memory. Thus, the constraint ③ is not met.

2.5 Per-grade data structure

We follow a different approach by storing multivector elements by grade. A multivector is considered as a set of arrays, all dedicated to a specific grade. This set contains only arrays related to grades explicitly expressed by the represented multivector, but still, an array may contain zero values, as depicted in figure 2.1. Again, this choice is motivated by the fact that most GA entities consist in homogeneous multivectors. In this situation, the array dedicated to the specific grade of an arbitrary object is likely to be full and thus much more effective than a linked list. In fact, for such a structure, the time access to a **homogeneous** vector is in $\mathcal{O}\Big(\log(d)\Big)$ and the time access to an element of this vector is in $\mathcal{O}(1)$. Finally, only homogeneous vector elements are allocated.

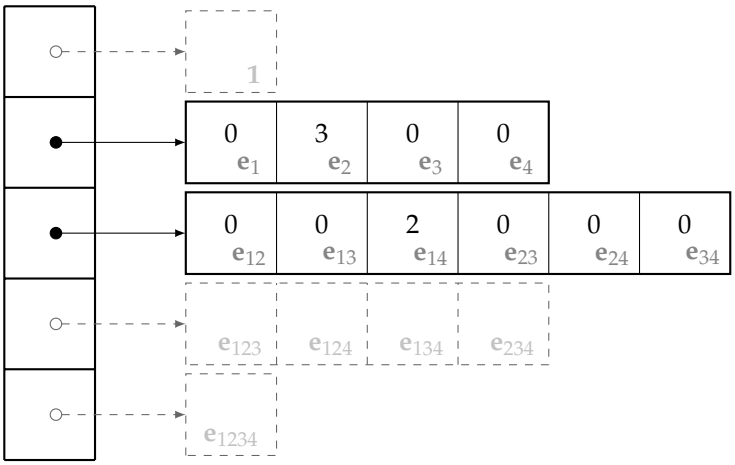


FIGURE 2.1: Data structure, example with $\mathbf{x} = 3\mathbf{e}_2 + 2\mathbf{e}_{14}$ in a 4-dimensional vector space. The column on the left denotes the grade

Hence, this structure meet the three constraints of the data structure ①, ② and ③. Another example of our data structure for the definition of a sphere in CGA is shown in Figure 2.2

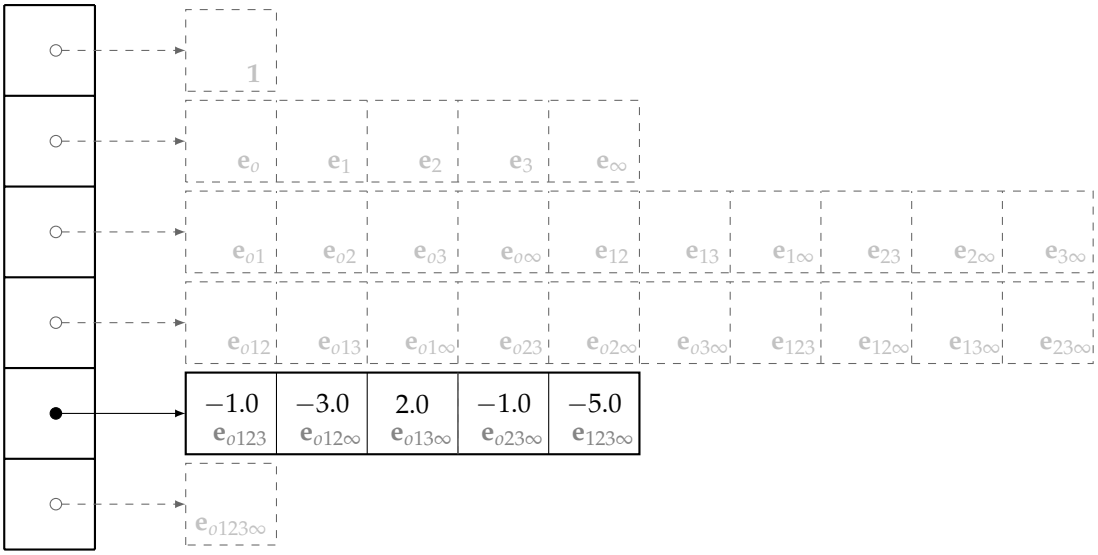


FIGURE 2.2: Data structure, example with the initialization of a sphere of CGA (5-dimensional vector space). This sphere has a radius of 2 and its center point at coordinates $(1, 2, 3)$. Note that there are no useless zeros stored

In practice, storing this GA elements as per grade arrays is also motivated by some code optimization using SIMD registers. In that case, even storing some zero often does not affect the computation speed since the SIMD registers process several multivector product operations simultaneously.

2.6 What to compute?

As mentioned in section 2.2.1, GA implementation aims to avoid to store zero data. In practice, GA implementations of products also tend to avoid manipulation of zeros. To be more exhaustive, we define 3 types of zeros that can be encountered in GA operations. Let consider the example:

$$\mathbf{C} = \mathbf{a} \wedge \mathbf{b}$$

with

$$\mathbf{a} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 3 & 2 & 0 & 0 & 0 & 0 \\ \hline 1 & \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 & \mathbf{e}_{12} & \mathbf{e}_{13} & \mathbf{e}_{23} & \mathbf{e}_{123} \end{array}$$

$$\mathbf{b} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 4 & 2 & 1 & 0 \\ \hline 1 & \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 & \mathbf{e}_{12} & \mathbf{e}_{13} & \mathbf{e}_{23} & \mathbf{e}_{123} \end{array}$$

$$\mathbf{C} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ \hline 1 & \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 & \mathbf{e}_{12} & \mathbf{e}_{13} & \mathbf{e}_{23} & \mathbf{e}_{123} \end{array}$$

The naive double loop over the elements of **a** and **b** would encounter different types of zeros. Considering that GA objects usually have limited different grade elements, these zero can be listed as:

1. **structural zero**: when an operation leads to zero due to the nature of the product independently of the coefficients (i.e. $\underbrace{3\mathbf{e}_2}_{\mathbf{a}} \wedge \underbrace{4\mathbf{e}_{12}}_{\mathbf{b}}$).
2. **object zero**: when **a** and **b** are homogeneous GA objects, computing products with elements of grade not related to the object is useless (i.e. $\underbrace{3\mathbf{e}_2}_{\mathbf{a}} \wedge \underbrace{0\mathbf{e}_1}_{\mathbf{b}}$).
3. **data zero**: when not all the components of grade k are used to express a GA object of grade k (i.e. $\underbrace{0\mathbf{e}_1}_{\mathbf{a}} \wedge \underbrace{1\mathbf{e}_{23}}_{\mathbf{b}}$).
4. **computational zero**: when an element should theoretically be zero but is numerically non-zero due to numerical errors.

In any cases, "computational zeros" are very difficult to handle. Indeed, a program will hardly identify if a value $x = 10^{-9}$ is an expectable value in the considered problem or a numerical error (like in `x = 0.1f - 0.1L` $\simeq 10^{-9}$ in C/C++ language). In list above, the most interesting useless product is the "structural zeros" that seems unavoidable. How to avoid this product without checking the compatibility of the two basis vectors involved in the operation? The answer is a consequent part of this thesis.

2.7 Naive methods to compute the products

2.7.1 Inside the products

The geometric product, the inner product and the outer product between 2 multivectors **a** and **b** are distributive over the addition and can conceptually be computed by iterating over the components of both **a** and **b**. For this conciseness purpose, we limit our description to the outer product, but the overall method remains true for the geometric product and the inner product. As we saw in the previous chapter, the outer product $\mathbf{C} = \mathbf{a} \wedge \mathbf{b}$ can be expressed as:

$$\mathbf{C} = \sum_{k=0}^{2^d-1} c_k \mathbf{E}_k = \left(\sum_{i=0}^{2^d-1} a_i \mathbf{E}_i \right) \wedge \left(\sum_{j=0}^{2^d-1} b_j \mathbf{E}_j \right) \tag{2.7.1}$$

where $\mathbf{E}_i$ refers to a basis vector, i.e $\mathbf{E}_i \in \{\mathbf{1}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_{12}, ...\}$. Each pair $\{a_i, b_j\}$ leads to a computed element that contributes to the final value of **c**, and should be assigned to the adequate basis vector of **c**, namely $\mathbf{E}_k = \pm\, \mathbf{E}_i \wedge \mathbf{E}_j$. This assignment can be decided either by a precomputed table that requires some memory space or by a function that requires some additional processing time.

2.7.2 Table based methods

An efficient way to compute the Geometric Algebra products is to use precomputed 2D-tables indicating the result of the products between any basis blades. This approach is used by a plugin of Gaalop. A product between two multivectors **a** and **b** consists of a loop over each basis blades present in the two multivectors. For each couple $\{a_i, b_j\}$, the program refers to a table to know where to put the result. Although this approach is simple and effective, it includes some useless operations. Indeed, the outer product between dependent basis blades (e.g. $\mathbf{e}_1 \wedge \mathbf{e}_{12}$) requires a table access and are discarded if the table says so. The pseudo-code of this approach is presented on Algorithm 1.

The two main drawbacks of this approach are:

- the memory consumption of the table that grows exponentially with d
- structural zero are not avoided

Each basis blade product is a constant time algorithm. However, as the size of the multivector components increases, the cost of this structural zero will become costly.

Concerning the memory consumption, a table should include all the basis vectors of the algebra

Algorithm 1: Product using the table based method.

Input: multivectors: A and B,
Table: T and S,
Dimension of the algebra: d
Output: resulting multivector: **C**

```
1 for i from 0 to 2^d − 1 do
2     if A[i] non-zero blade then
3         for j from 0 to 2^d − 1 do
4             if B[j] non-zero blade then
5                 index ← T[i, j]
6                 if index ≠ null then                    // i.e. a e_i ∧ b e_j ≠ 0
7                     sign ← S[i, j]
8                     C[index] += sign × A[i] × B[j]
```

in the table, leading to a $2^d \times 2^d = 4^d$ size table, where d is the dimension of the algebra. Each element of the table is composed of an index on where to put the result and a sign (sometimes, the result is negative). Moreover, Geometric Algebra involves at least 3 products (geometric, outer and inner products), leading to at least 3 tables. If we assume that the sign and index are stored in a signed integer of 6 bits, then the size of the tables would be $2^5 \times 2^5 \times 6 \times 3 \approx 18$ Kbits. However, this would be a lower bound. Indeed, a geometric product between two basis blades might result in a sum of basis blades, as encountered in Conformal Geometric Algebra. Thus, in practice, the tables contain a structure element that may enclose more than one index and sign. In order to estimate the precise memory requirement of the table-based methods, we ran a profiler on Gaalop. In a 5-dimensional space, the table-based approach of Gaalop requires 1.1 MB of memory and requires 710 MB for dimension 10. According to the progression of a $O(4^d)$ memory complexity, we can infer that for a dimension 11, the table size will occupy more than 2 GB memory and for a dimension 15, it would be around 360 GB. This memory requirement becomes a critical issue for high dimensional spaces.

2.7.3 Function based methods

During the product between **a** and **b**, the access to a table can be replaced by a call of a function that specifies where to put the result of a product between an element of **a** and an element of **b**. In practice, this approach just changes lines 5 and lines 7 of Algorithm 1. Gaigen [23, 33] propose this kind of functions, based on logical operations on a binary representation of the basis vectors. Another function, also based on binary operators, is required to specify the sign of a product contribution. The pseudo-code of the computation of the sign is shown on Algorithm 2 as presented in [32].

Algorithm 2: Sign computation knowing the binary representation of the two considered blades.

Input: binary representation of blades: **A** and **B**,
Dimension of the algebra: d
Output: resulting sign: $sign$

```
1  sign ← 1
2  A ← A >> 1                                   // corresponds to a bit shifting
3  while A ≠ 1 do
4  |  sign ← sign+ hammingWeight(A&B)
5  |  A ← A >> 1
6  if sign&1 = 0 then
7  |  sign ← 1
8  else
9  |  sign ← −1
10 return sign
```

In one hand, the function `hammingWeight()` of Algorithm 2 computes the Hamming weight (number of 1-bits in its argument). This function can be computed in constant time without the storage of a huge size table, see [29]. Moreover, the loop over all non zero elements of **A** has a complexity proportional to the dimension, $\mathcal{O}(d)$.

Algorithm 1 shows that the sign computation is repeted in the worst case 4^d. Thus, the complexity of such algorithm is in $\mathcal{O}(d \times 4^d)$.

2.7.4 Complexity issue

To conclude, these algorithms are computationally expensive for both low and high dimensions. This latter point suggests to explicitly compute the product for any kind of grades of the two multivectors and use these products when required. This way, the sign along with the resulting blades would not have to be recomputed. Again, this holds for low dimensions but not for high dimensions.

Chapter 3

Products in Low dimensional space

In low dimensional vector spaces, it is neither necessary to use tables nor any logical operations that are computationally expensive.

3.1 Per grade products

Considering the per grade data structure defined in section 2.2.1, the most efficient way to process any product is to pre-compute it in advance. Since the outer, inner and geometric products are distributive over the addition, each "per grade product" can be extracted and computed independently. Let $\langle X \rangle_k$ be the part of the multivector X of grade k, and $D_X = \{\langle X \rangle_i \neq 0, i \in 0, \cdots, d\}$ be the set of all k-vector $\langle X \rangle_k$ of any grade present in $\mathbf{x}$. Then, most of the products $\odot$ between the multivectors A and B can be computed by the double loop algorithm as presented in algorithm 3, (geometric product is a special case). Note that the function $\texttt{find\_grade}(\odot, k_A, k_B)$ is a constant time algorithm and merely consists in computing:

- $k_A + k_B$ for the outer product,
- $k_B - k_A$ for the left contraction,
- $k_A - k_B$ for the right contraction,
- $|k_A - k_B|$ for the inner product,

Algorithm 3: Per grade loop

input : multivectors A and and B,
a product $\odot$ distributive over the addition
output: multivector: $C = A \odot B$

```
1 foreach k-vector ⟨A⟩_{k_A} ∈ D_A do
2     foreach k-vector ⟨B⟩_{k_B} ∈ D_B do
3         k_c = find_grade(⊙, k_A, k_B)
4         ⟨C⟩_{k_C} = product_k_A_k_B(⊙, A, B)
5 return C
```

In practice, these two loops are likely to contain only one call, in the case where A and B are homogeneous multivectors. Moreover, each product called `product_kA_kB` in the Algorithm can be precomputed in advance, according to the specified GA signature. An example of the generated codes for the outer product can be found in A.

As for the geometric product, Algorithm is 3 is slightly different. In fact, the grade of the geometric product between one multivector whose grade is k_A and another whose grade is k_B, ranges from $|k_A - k_B|$ (inner product) to $k_A + k_B$ (outer product). Thus, a loop over possible grades of C is added to the Algorithm 3. The resulting pseudo-code is shown in Algorithm 4

Algorithm 4: Per grade geometric product

input : multivectors A and and B
output: multivector: $C = A \odot B$
foreach k-vector $\langle A \rangle_{k_A} \in D_A$ **do**
  foreach k-vector $\langle B \rangle_{k_B} \in D_B$ **do**
    for $k_C = |k_A - k_B|, |k_A - k_B| + 2, \cdots, k_A + k_B - 2, k_A + k_B$ **do**
      $\langle C \rangle_{k_C} =$ `geometric_product_kC_kA_kB`(A, B)
return C

3.2 SIMD instructions

As presented in Algorithm 3, a multivector product is divided in sub-products of homogeneous grade. Each sub-product $\langle C \rangle_{k_C} = \langle A \rangle_{k_A} \odot \langle B \rangle_{k_B}$ consists in a list of atomic basic contributions of the form c_l+=$w a_m b_n$ where c_l, w, a_m and b_m are elements of the base field used to build the algebra. These instructions are easily converted to C++ code but the conversion to SIMD is not straightforward since some low level memory constraints should be considered. First, "writing" a new value in a variable is more costly than just reading it. Second, dealing with consecutive array elements should be highly preferred. Moreover, a specific care must be given to data cache transfer minimization.

For a given product, all the atomic instructions c_l+=$w a_m b_n$ are sorted according to first the resulting k-vector element index l, then if necessary according to the left and right operand index n and m. In the current version of Garamon, the SIMD instructions uses only `mavx` intrinsics and avoid `mavx2` functions such that it can run on almost any computer. In case of incompatibility, the SIMD instructions can just be disabled such the program automatically use instead the default C++ functions.

In practice, the SIMD implementations of GA products are actually not as impressive as expected and leads to roughly the same performances as the regular C++ version compiled with `-O2` option. However, further investigations may lead to a more consequent speed up.

3.3 Dual computation

For any full-rank GA signature, the dual multivector computation can be optimized in advance. By definition the dual of a multivector is given by:

$$A^* = A \cdot \mathbf{I}^{-1} = \frac{A \cdot \widetilde{\mathbf{I}}}{\mathbf{I} \cdot \widetilde{\mathbf{I}}} \tag{3.3.1}$$

This expression requires the computation of two inner products, a reverse and a scalar division, that can be precomputed in advance. Due to the symmetry property of the Pascal's triangle, a k-vector A and its dual A^* both have the same number of elements. In the array based data structure of section 2.2.1, computing the dual of a k-vector thus just consists in changing the "grade label" of the corresponding array from k to $d - k$ (for a vector space of dimension d), permuting some array elements and eventually multiplying them by some constant according to the metric of the algebra. Extending this method to a multivector means to apply it to all non-null blades of the multivector. Concerning the implementation, the dual is merely computed by pre-computing both an array that stores the required permutation for each array and a vector that stores the coefficients to apply to each resulting array.

Chapter 4

Products in high dimensional space

Section 2.7.1 describes the state of the art of common optimizations of GA products used in most of the GA libraries. These products fail in high dimension space, due to memory overload or to complexity issue. Indeed, using tables is too memory expensive ($\mathcal{O}(4^d)$) and using bits based algorithms is too computationally expensive ($\mathcal{O}(d \times 4^d)$). In order to meet the need of computational efficiency 2a, we are seeking for an approach which has:

- low memory complexity
- low computational complexity

In this thesis, we propose some approaches based on [36] developed by Fuchs and Théry. In this latter approach, each product is explicitly defined using recursive definitions. The approach of Fuchs and Théry leads to efficient computation of products. However, this method does not handle parallel algorithm implementations. The next section presents a first approach to fix this latter issue.

4.1 Recursive scheme

The first method that we proposed consists in restructuring the recursive functions of [36] such that the products can be specialized according to the grade or a particular coefficient of the resulting multivector. The proposed approach yields to parallel algorithms. The next sections start by explaining the approach followed [36] and finish by presenting our first proposed approach.

4.1.1 Definitions and Notations

We start with the notations used for the sequel. Similarly to previous sections lower-case bold letters will denote basis blades and multivectors (multivector $\mathbf{A}$). Lower-case letters refer to multivector coordinates. For example, a_i is the i^{th} coordinate of the multivector $\mathbf{A}$. Lower-case

and Frakture letters denote multivector expressed over a tree structure. For example, $\mathfrak{a}$ represents a multivector over a tree structure. This notion is presented in the next subsection. Finally, the vector space dimension is denoted by 2^d, where d is the number of basis blades $\mathbf{e}_i$ of grade 1.

4.1.2 Recursive form of Geometric Algebra

Fuchs and Théry [36] presented a recursive formalism to deal with Geometric Algebra multivector and operators. As our approach is based on this recursive formalism, it is worth presenting this framework. Let us first present the recursive construction of multivector.

4.1.2.1 Binary Trees and Multivectors

A multivector A is represented as a sum of basis blades weighted by coefficients a_i, i.e.:

$$A = \sum_{i=0}^{2^d-1} a_i \mathbf{E}_i \tag{4.1.1}$$

where $\mathbf{E}_i$ denotes a basis blade whose grade ranges from 0 to d (i.e. $\mathbf{E}_7 = \mathbf{e}_{123}$ with $d = 3$).

The approach [36] proposed a recursive representation of multivectors over binary trees. This binary tree is recursively defined as follows:

$$\mathfrak{a}^n = (\mathfrak{a}_1^{n+1}, \mathfrak{a}_0^{n+1})^n \tag{4.1.2}$$

where n is the depth of recursion. This depth may vary from 0 (root level) to d (leaf level). In this representation, each node of the binary tree $\mathfrak{a}$ is considered as multivector which contributes to the construction of the multivector $\mathbf{a}$. Each depth of the binary tree corresponds to a basis vector (of grade 1). This basis vector contributes to the construction of the multivector of upper grades. A binary tree $(\mathfrak{a}_1^{n+1}, \mathfrak{a}_0^{n+1})^n$ is thus interpreted as $(\mathbf{e}_n \wedge \mathfrak{a}_1) + \mathfrak{a}_0$ at each depth n, that is to say $\mathfrak{a}^n$ is divided into two sub-trees, namely the tree $\mathfrak{a}_1$ containing $\mathbf{e}_n$ and $\mathfrak{a}_0$ which does not, as shown on Figure 4.1. Considering that a leaf represents a basis blade, a multivector is defined by the mapping between these leaves and the multivector coefficients (see coefficients a_i on Figure 4.1). For the following, we define a node $\mathfrak{a}^n$ as a multivector composed of basis blades whose maximum grade is n.

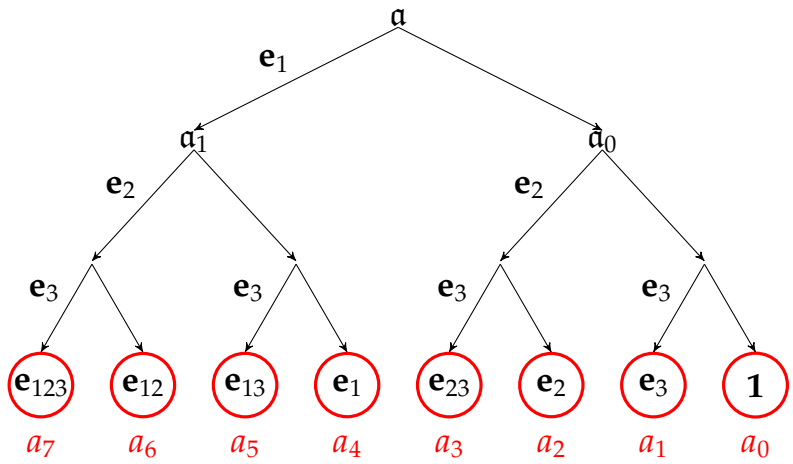


FIGURE 4.1: Binary tree of $\mathfrak{a}$, representing a multivector in a 2^3-dimensional space. There is a mapping between the coefficients a_i of the multivector and the leaves of the tree.

4.1.2.2 Outer product

In this section, we consider the computation of the outer product in any d-dimensional space. Let $\mathbf{c} = \mathbf{a} \wedge \mathbf{b}$ be the outer product between two multivectors $\mathbf{a}$ and $\mathbf{b}$, then we have:

$$C = \sum_{k=0}^{2^d-1} c_k \mathbf{E}_k = \sum_{i=0}^{2^d-1} a_i \mathbf{E}_i \wedge \sum_{j=0}^{2^d-1} b_j \mathbf{E}_j \tag{4.1.3}$$

The distributivity of the outer product leads to:

$$C = \sum_{i=0}^{2^d-1} \sum_{j=0}^{2^d-1} a_i b_j \, \mathbf{E}_i \wedge \mathbf{E}_j \tag{4.1.4}$$

Finally a coefficient c_k of $\mathbf{c}$ can be expressed as:

$$c_k = \sum_{\substack{i,j \text{ such that} \\ \mathbf{E}_i \wedge \mathbf{E}_j = s_{ij} \mathbf{E}_k}} s_{ij} \, a_i b_j \tag{4.1.5}$$

where s_{ij} is -1 or 1 according to parity of the number of permutations required to change $\mathbf{E}_i \wedge \mathbf{E}_j$ in $\mathbf{E}_k$. In order to extract each coefficient c_k, one may think of computing each product $\mathbf{E}_i \wedge \mathbf{E}_j$, as performed by the non-specialized form of Gaigen. In this case, some products will lead to useless operations where $\mathbf{E}_i \wedge \mathbf{E}_j = 0$, for example where $i = j$. General multivectors $\mathbf{a}$ and $\mathbf{b}$ may have up to 2^d non-zero coefficients, leading to $2^d \times 2^d = 4^d$ products. This number of operations can be reduced by avoiding useless products.

In [36], each product is explicitly defined using a recursive definition over binary trees. The outer product between $\mathfrak{a}$ and $\mathfrak{b}$ is defined as follows:

$$\begin{aligned} &\mathfrak{a}^n \wedge \mathfrak{b}^n = (\mathfrak{a}_1^{n+1}, \mathfrak{a}_0^{n+1})^n \wedge (\mathfrak{b}_1^{n+1}, \mathfrak{b}_0^{n+1})^n \\ &\text{if } n < d \text{ , } \mathfrak{a}^n \wedge \mathfrak{b}^n = \left(\mathfrak{a}_1^{n+1} \wedge \mathfrak{b}_0^{n+1} + \overline{\mathfrak{a}_0}^{n+1} \wedge \mathfrak{b}_1^{n+1}, \mathfrak{a}_0^{n+1} \wedge \mathfrak{b}_0^{n+1}\right)^n \\ &\text{if } n = d \text{ , } \mathfrak{a}^n \wedge \mathfrak{b}^n = \mathfrak{a}^d \wedge \mathfrak{b}^d \end{aligned} \tag{4.1.6}$$

where $\overline{\mathfrak{a}}$ expresses the anti-commutativity of the outer product. The outer product of two multivectors consists in developing the hereabove formula from the root to the leaves. From this formula, we may derive that for each depth n, the number of recursive calls is multiplied by 3. Thus, the total number of recursive calls for a d-dimensional space is:

$$\sum_{i=1}^{d} 3^i = \frac{3}{2}(3^d - 1) \tag{4.1.7}$$

Therefore, the number of recursive calls is thus in $\mathcal{O}(3^d)$ which is less than $\mathcal{O}(d4^d)$ corresponding to the complexity of the naive method.

4.2 Geometric Algebra operators as a recursive construction of lists

4.2.1 Outer product as a recursive construction of lists

To reduce the number of products of Equation (4.1.4) and to use parallel optimizations, we present a rewriting of the recursive functions. The contributions of our method are:

- to extract all products $a_i b_j$ for a considered coefficient $\mathfrak{c}_k$, and the sign associated with each product,
- to determine the products involved in $\mathfrak{c}$ knowing the grade of $\mathfrak{a}$ and $\mathfrak{b}$.

4.2.2 Binary trees labelling

We first define a label for each node of the binary tree, derived from Huffman labelling, as illustrated in Figure 4.2. The label of a leaf provides the path from this leaf to the root. For example,

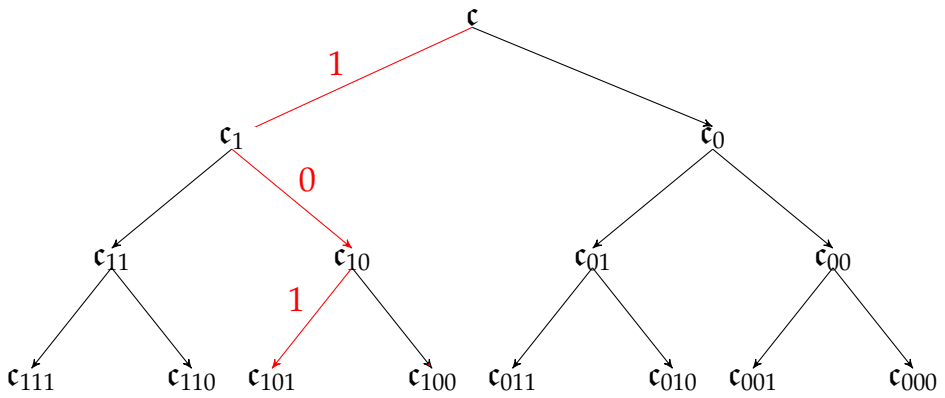


FIGURE 4.2: Labelling of a binary tree $\mathfrak{c}$ in a 3-dimensional space.

the path that connects the leaf labelled 101 to the root is (left, right, left), as shown in red Figure 4.2.

4.2.2.1 From trees to lists

We know, from Equation 4.1.5, that each leaf c_k is expressed as a sum of products $a_i b_j$. In order to compute these products, we want to identify the lists a_i and b_j involved in the computation of each coefficient c_k. For example, the computation of c_5 will be based on the extraction of the list (a_5, a_4, a_1, a_0) and (b_0, b_1, b_4, b_5), see Table 4.1.

TABLE 4.1: Outer product table in a 3-dimensional space.

$\mathbf{e}_{123}$	$\mathbf{e}_{12}$	$\mathbf{e}_{13}$	$\mathbf{e}_1$	$\mathbf{e}_{23}$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{1}$
c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
a_7b_0	a_6b_0	a_5b_0	a_4b_0	a_3b_0	a_2b_0	a_1b_0	a_0b_0
a_6b_1	a_4b_2	a_4b_1	a_0b_4	a_2b_1	a_0b_2	a_0b_1	
$-a_5b_2$	$-a_2b_4$	$-a_1b_4$		$-a_1b_2$			
a_4b_3	a_0b_6	a_0b_5		a_0b_3			
a_3b_4							
$-a_2b_5$							
a_1b_6							
a_0b_7							

In order to identify these products for any dimension and for any coefficient c_k, we transform the recursive functions. More precisely, instead of reducing a set of products to a base case (recursive function), we start with a base case (root node), then we build some sequences forward from the base case. To achieve this, we use the labelling on Equation 4.1.6 to identify each node by a word u. A binary tree $\mathfrak{a}^n$ can be expressed as $\mathfrak{a}^n_u$, with its left child rewritten as $\mathfrak{a}^{n+1}_{u1}$ and its right child as $\mathfrak{a}^{n+1}_{u0}$.

Using this labelling, we can also rearrange the recursive definition of the outer product (Equation 4.1.6) as follows:

$$\begin{aligned}
&\mathfrak{a}_u^n \wedge \mathfrak{b}_v^n = (\mathfrak{a}_{u1}^{n+1}, \mathfrak{a}_{u0}^{n+1})^n \wedge (\mathfrak{b}_{v1}^{n+1}, \mathfrak{b}_{v0}^{n+1})^n \\
&\text{if } n < d \text{ , } \mathfrak{a}_u^n \wedge \mathfrak{b}_v^n = \left(\mathfrak{a}_{u1}^{n+1} \wedge \mathfrak{b}_{v0}^{n+1} + \overline{\mathfrak{a}_{u0}}^{n+1} \wedge \mathfrak{b}_{v1}^{n+1} \text{ , } \mathfrak{a}_{u0}^{n+1} \wedge \mathfrak{b}_{v0}^{n+1}\right)^n \\
&\text{if } n = d \text{ , } \mathfrak{a}_u^n \wedge \mathfrak{b}_v^n = \mathfrak{a}_u^d \wedge \mathfrak{b}_v^d
\end{aligned} \tag{4.2.1}$$

In order to identify the components a_i and b_j involved in the computation of each coefficient c_k, we extract a construction of labels of $\mathfrak{a}$ and $\mathfrak{b}$ from Equation (4.2.1). Let *Alist* and *Blist* be these recursive constructions for labels of $\mathfrak{a}$ and $\mathfrak{b}$ respectively. In the following, we will only consider the construction of labels of $\mathfrak{a}$. We can prove by induction that labels of $\mathfrak{b}$ are the labels of $\mathfrak{a}$ in reverse order. The recursive definition of this *Alist* is the following:

$$\begin{aligned}
&\text{if } n < d \text{ , } \{Alist^n\} = (\{Alist(1)^{n+1}, Alist(0)^{n+1}\}, \{Alist(0)^{n+1}\}) \\
&\text{if } n = d \text{ , } \{Alist^n\} = \{Alist\}
\end{aligned} \tag{4.2.2}$$

where $\{.\ ,\ .\}$ denotes the merge of two lists of labels, and $Alist(r)$ indicates that the elements of the list are suffixed by the letter r. The resulting tree of dimension 2^3 is depicted on Figure 4.3.

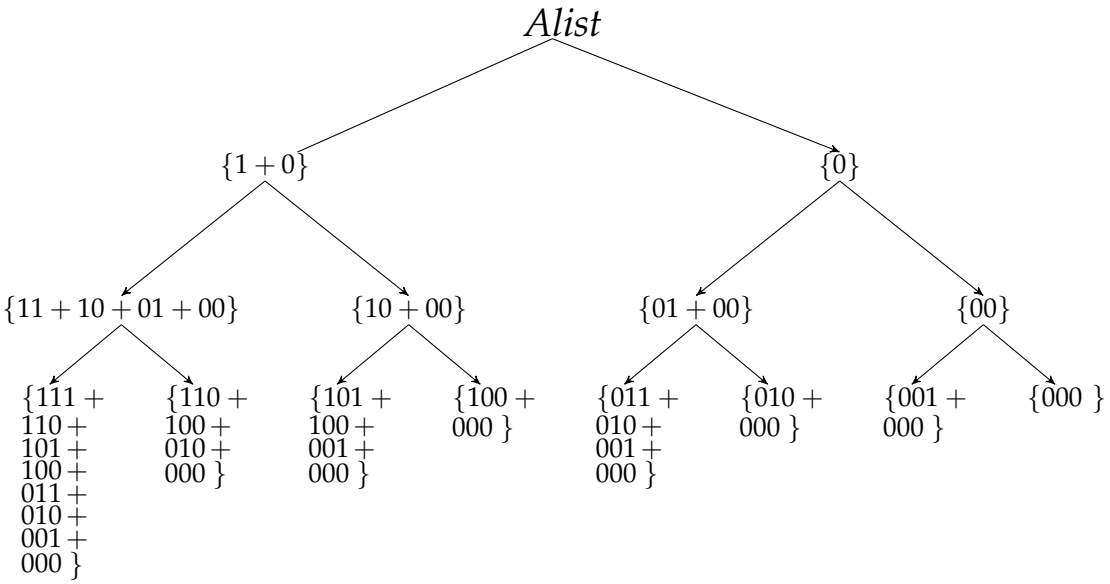


FIGURE 4.3: Recursive definition of *Alist* pushed down to the leaves.

4.2.2.2 Construction of *Alist*

The recursive construction of the *Alist* provides the number p of products associated to a label u of $\mathfrak{c}$:

$$p = 2^{h(u)} \tag{4.2.3}$$

where $h(u)$ denotes the Hamming weights of u (i.e. the number of ones in a binary word u).

We now introduce an approach to determine the evolution of the labels of $\mathfrak{a}$ involved in the computation of each leaf $\mathfrak{c}$. The recursive function (4.2.2) shows that each left sub-tree is both suffixed by 0 and 1, meaning a duplication of the number of elements of the list, whereas the right sub-tree is only suffixed by 0. From this result, an algorithm to construct *Alist* at only one level is extracted. In this algorithm, *Alist* is represented by a list of binary sequences. A binary sequence is an integer representing a label. Algorithm 5 gives pseudo-code for the method.

Algorithm 5: Outer product recursive construction of the list of contributions of $\mathfrak{a}$.

```
1  Function oneLevelOuterProduct
       Input: list: list of binary sequences
              b: bit of a label
2      tmpList ← { }
3      if b == 1 then
4          tmpList.push(addBitList(list, 1, 0))
5      else
6          tmpList.push(addBitList(list, 0))
7      return tmpList
8
9  Function addBitList(list, bit)
10     listRes ← { }
11     foreach label u of list do
12         listRes.push(concat(u, bit))
13     return listRes
14
   // Overload the function above
15 Function addBitList(list, bitA, bitB)
16     listRes ← { }
17     foreach label u of list do
18         listRes.push(concat(u, bitA))
19         listRes.push(concat(u, bitB))
20     return listRes
```

As an example, let us consider the *Alist* for the node $\mathfrak{c}_{101}$. This construction is equivalent to computing each product $a_i b_j$ for the leaf c_5 (for example $5_{(10)} = 101_{(2)}$). The computation of the *Alist* for the coefficient $\mathfrak{c}_{101}$ is described Table 4.2. In this table, the binary words $(000, 001, 100, 101)$ correspond to the coefficients (a_0, a_1, a_4, a_5) in table 4.1.

TABLE 4.2: *Alist* computed at the node $\mathfrak{c}_{101}$.

bit	subtree	*Alist*
Beginning	root	ϵ
1	left subtree	$(0, 1)$
	$\downarrow$	
0	right subtree	$(00, 10)$
	$\downarrow$	
1	left subtree	$(000, 001, 100, 101)$

Algorithm 6: Outer product corresponding to a coefficient $\mathfrak{c}_k$

Data:
$\mathfrak{a}, \mathfrak{b}$: multivectors
Alist: sequence of binary words
k: label of a coefficient of the multivector $\mathfrak{c}$
d: dimension
Result: $\mathfrak{c}_k$: k^{th} coefficient of the multivector $\mathfrak{c} = \mathfrak{a} \wedge \mathfrak{b}$

```
Alist ← { }
Sign ← { }
SignOuter(Sign, 1, 1, d)                          // refer to Algorithm 8
foreach bit k_i of k do
    Alist ← oneLevelOuterProduct(Alist, k_i)
c_k ← 0
i ← 0
foreach label u of Alist do
    c_k ← c_k + Sign[i] · a_u · b_reverse(u,d)
    i ← i + 1
```

Algorithm 6 shows the pseudo-code of our method for a considered coefficient of $\mathfrak{c}$. The function `reverse`$(binaryWord, dimension)$ computes the operation $(2^d - 1 - binaryWord)$, in order to compute the *Blist* coefficients from a *Alist*. Thus, Algorithm 6 enables us to compute any coefficients of $\mathfrak{c}$ independently. Therefore, the algorithm can be used to compute different coefficients of $\mathfrak{c}$ in parallel.

The latter algorithm can be further improved when the grade of the two multivectors are known. Let M be the grade of $\mathfrak{a}$, N the grade of $\mathfrak{b}$ and L the grade of $\mathfrak{a} \wedge \mathfrak{b}$. Then the grade of $\mathfrak{a} \wedge \mathfrak{b}$ is $L = M + N$. Thus, the computation of the outer product is equivalent to identifying and computing each coefficient $\mathfrak{c}_k$ whose grade is L. The labelling of the binary tree enables to efficiently extract the leaves of $\mathfrak{c}$ whose grade is L. An algorithm is produced and consists in traversing the binary tree of $\mathfrak{c}$. At each depth of this tree, if the grade of the label of $\mathfrak{c}$ is L then the products at this label can be computed and the children of this node don't have to be traversed. This enables us to efficiently compute the products. The algorithm used is shown in Algorithm 7. This algorithm is used to implement a per-grade specialization of our implementation.

Finally, note that the considered label is enough to compute the *Alist* elements. Firstly, the binary

Algorithm 7: Outer product of two multivectors, with a known resulting grade

Data:
Sign: Sequence of signs
$\mathfrak{a}, \mathfrak{b}$: multivectors in tree form

1 **Function** `perGradeOuterP`
  Input:
    Alist: sequence of binary words
    $\mathfrak{c}_k$: node of $\mathfrak{c} = \mathfrak{a} \wedge \mathfrak{b}$
    L: grade of $\mathfrak{c}$
    d: dimension
    $lPathNum$: number of ones in the label of the node of $\mathfrak{c}$
    depth: depth of the current node
  Result: $\mathfrak{c} = \mathfrak{a} \wedge \mathfrak{b}$
  `// lPathNum represents the grade of the node`
2 **if** $lPathNum == L$ **then**
3    $i \leftarrow 0$
4   **foreach** label u of *Alist* **do**
5      $v \leftarrow 2^{d-depth} \cdot u$ `// from node to leaf`
6      $\mathfrak{c}_k \leftarrow \mathfrak{c}_k + Sign[i] \cdot \mathfrak{a}_v \cdot \mathfrak{b}_{\texttt{reverse}(v,d)}$
7      $i \leftarrow i + 1$
8 **else**
9    $list \leftarrow \{\ \}$
10   $depth \leftarrow depth + 1$
11   **if** $\mathfrak{c}_k$ *is not a leaf* **then**
12     $list \leftarrow$ `oneLevelOuterProduct`(*Alist*,1)
13   `perGradeOuterP`(*list*, $\mathfrak{c}_{k1}$, L, d, $lPathNum + 1$,*depth*)
14     $list \leftarrow$ `oneLevelOuterProduct`(*Alist*,0)
15   `perGradeOuterP`(*list*, $\mathfrak{c}_{k0}$, L, d, $lPathNum$, depth)

0 is in this list. Then the other elements can be defined by the list of binary words whose length is d such that the logical AND operator between this binary word and the label is non-zero. This method is derived from the Hamming expression of Equation (4.2.3).

4.2.3 Complexity of this method

The performance of our method is estimated from Algorithm 6. The cost of the function *oneLevelOuterProduct*() in Algorithm 6 is linear to the size of the list and more precisely proportional to the hamming weight of the coefficient of the node. Thus, this operation is repeated d times. The latter computation is repeated 2^d times in the worst case (2^d non-zero coefficients) which leads to a complexity proportional to 3^d. The performance of this approach is thus asymptotically equivalent to the cost of the previous approach [36].

4.2.4 Sign computation

The sign of each product $a_u b_v$ is now computed. This computation might be the most time-consuming part of the outer product. The first method that we explored consisted in a "convolution" between the considered label in *Alist* and in *Blist* in a similar way to [23]. The convolution consists of right-shifting each bit of the label in *Alist* until the label is zero. At each iteration, we count the number of ones in common between the shifted label and the other label. The sign is obtained by raising -1 to the power of the number of ones.

The performance of this approach is estimated by computing the total number of right-shifting in $\mathfrak{a} \wedge \mathfrak{b}$. For a label a in *Alist*, the maximum number of right-shifting is $\lfloor log_2(a) \rfloor$. As mentioned in the previous subsection, the occurrence of a label a in the computation of the outer product is given by $2^{d-h(a)}$, with h the hamming weight of a. Instead of computing the leading formula, we give an upper bound to the number of total right-shifting. From the Equation 4.1.1, an upper bound to the coefficient is 2^d. Therefore, for any label $a \geq 1$, the upper bound of the maximum number of right-shifting is $\lfloor log_2(2^d) \rfloor = d$. Thus, the number of right-shifting for the total number of occurrences of the label a is bounded as follows:

$$2^{d-h(a)} \cdot \lfloor log_2(a) \rfloor \leq d \cdot 2^{d-h(a)} \tag{4.2.4}$$

The total number of right-shifting is obtained by summing over the number of coefficients in a multivector:

$$\sum_{a=1}^{2^d-1} 2^{d-h(a)} \cdot \lfloor log_2(a) \rfloor \leq \sum_{a=1}^{2^d-1} d \cdot 2^{d-h(a)} \tag{4.2.5}$$

Due to the linearity of the summation, we can rearrange the upper bound by extracting the dimension the following way:

$$\sum_{a=1}^{2^d-1} 2^{d-h(a)} \cdot \lfloor log_2(a) \rfloor \leq d \cdot \sum_{a=1}^{2^d-1} 2^{d-h(a)} \tag{4.2.6}$$

Here, the Hamming weight $h(a)$ is ranging from 1 to d. The number of coefficients whose hamming weight is k, is $\binom{n}{k}$. Hence, the upper bound can be rewritten as $\sum_{k=1}^{d} \binom{n}{k} 2^k$. From the binomial theorem, the upper bound is thus proportional to $d \cdot 3^d$.

We now introduce our method to reduce this number of arithmetic operations. The key point of the method lies in the fact that the sequence of signs over the binary tree remains the same for any coefficient $\mathfrak{c}$. This is explained by the structure of the recursive definition of the sign explained in [36]. The recursive definition of the sign is as follows:

$$\begin{aligned} &\text{if } n < d \text{ , } \overline{\mathfrak{a}_u}^n = \left(-\overline{\mathfrak{a}_{u1}}^{n+1}, \overline{\mathfrak{a}_{u0}}^{n+1} \right) \\ &\text{if } n = d \text{ , } \overline{\mathfrak{a}}^n = \mathfrak{a} \end{aligned} \tag{4.2.7}$$

This formula combined with the recursive definition of the outer product *Alist* is the following:

$$\begin{aligned} &\text{if } n < d \text{ , } \{Alist^n\} = \\ &\qquad \left(\{Alist(1)^{n+1}, \overline{Alist(0)}^{n+1} \}, \{Alist(0)^{n+1}\} \right) \\ &\text{if } n = d \text{ , } \{Alist^n\} = \{Alist\} \end{aligned} \tag{4.2.8}$$

From Equation (4.2.8), the sequence of signs is left unchanged for each right sub-tree. Therefore, the sequence of signs is completely determined by the sequence of signs of the far-left leaf of $\mathfrak{c}$.

Thus, we only have to compute the sequence of signs for the far-left leaf and store the sequence. Then for each leaf, the outer product algorithm goes through the elements of this sequence. Algorithm 8 gives pseudo-code for our method.

Algorithm 8: Computation of the sequence of signs

```
1 Function SignOuter
    Input:
      Sign: list to store the resulting signs
      currentSign: sign
      comp: complementary operator
      depth: depth of the current node
      d: dimension
2   if depth == d then
3     Sign.push(currentSign)
4   else
5     SignOuter(comp × currentSign, comp, depth + 1, d)
6     SignOuter(currentSign, −comp, depth + 1, d)
```

In this algorithm, the variable *comp* enables the flip of sign. From this algorithm, the number of operations to be performed is 2^d for a d-dimensional space. Therefore, using this algorithm in the computation of the outer product takes 3^d proportional time, thus in $\mathcal{O}(3^d)$.

4.2.5 Geometric product as a recursive construction of lists

4.2.5.1 Euclidean space

We now consider the computation of the geometric product of two multivectors $\mathfrak{a}$ and $\mathfrak{b}$ in an Euclidean space. We first describe the geometric product in an Euclidean space and then in an non-Euclidean space. We denote the geometric product between $\mathfrak{a}$ and $\mathfrak{b}$ by $\mathfrak{c} = \mathfrak{a} * \mathfrak{b}$. The overall method is equivalent to the method described in Section 4.2.2.1. The labelled recursive equation of the geometric product gives:

$$\begin{aligned}
&\mathfrak{a}_u^n * \mathfrak{b}_v^n = (\mathfrak{a}_{u1}^{n+1}, \mathfrak{a}_{u0}^{n+1})^n * (\mathfrak{b}_{v1}^{n+1}, \mathfrak{b}_{v0}^{n+1})^n \\
&\text{if } n < d\ ,\ \mathfrak{a}_u^n * \mathfrak{b}_v^n = \\
&\qquad (\mathfrak{a}_{u1}^{n+1} * \mathfrak{b}_{v0}^{n+1} + \overline{\mathfrak{a}}_{u0}^{n+1} * \mathfrak{b}_{v1}^{n+1}\ ,\ \overline{\mathfrak{a}}_{u1}^{n+1} * \mathfrak{b}_{v1}^{n+1} + \mathfrak{a}_{u0}^{n+1} * \mathfrak{b}_{v0}^{n+1})^n \\
&\text{if } n = d\ ,\ \mathfrak{a}_u^n * \mathfrak{b}_v^n = \ \mathfrak{a}_u^d * \mathfrak{b}_v^d
\end{aligned} \tag{4.2.9}$$

The development of this recursive formula in a 3-dimensional space is presented in Table 4.3, where the sign of each product is computed with a equivalent method used for the outer product, see Algorithm 9.

As for the outer product, we extract a recursive construction of the set of labels of $\mathfrak{a}$ and $\mathfrak{b}$. *Alist* and *Blist* are again these recursive constructions of labels. The recursive definitions of *Alist* and

TABLE 4.3: Geometric product table in a 3-dimensional space.

$\mathbf{e}_{123}$	$\mathbf{e}_{12}$	$\mathbf{e}_{13}$	$\mathbf{e}_1$	$\mathbf{e}_{23}$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{1}$
c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
a_7b_0	a_7b_1	$-a_7b_2$	$-a_7b_3$	a_7b_4	a_7b_5	$-a_7b_6$	$-a_7b_7$
a_6b_1	a_6b_0	a_6b_3	a_6b_2	$-a_6b_5$	$-a_6b_4$	$-a_6b_7$	$-a_6b_6$
$-a_5b_2$	$-a_5b_3$	a_5b_0	a_5b_1	a_5b_6	a_5b_7	$-a_5b_4$	$-a_5b_5$
a_4b_3	a_4b_2	a_4b_1	a_4b_0	a_4b_7	a_4b_6	a_4b_5	a_4b_4
a_3b_4	a_3b_5	$-a_3b_6$	$-a_3b_7$	a_3b_0	a_3b_1	$-a_3b_2$	$-a_3b_3$
$-a_2b_5$	$-a_2b_4$	$-a_2b_7$	$-a_2b_6$	a_2b_1	a_2b_0	a_2b_3	a_2b_2
a_1b_6	a_1b_7	$-a_1b_4$	$-a_1b_5$	$-a_1b_2$	$-a_1b_3$	a_1b_0	a_1b_1
a_0b_7	a_0b_6	a_0b_5	a_0b_4	a_0b_3	a_0b_2	a_0b_1	a_0b_0

Alist are the following:

$$\begin{aligned}&\text{If } n < d\ ,\ \{Alist^n\} = \\ &\quad (\{Alist(1)^{n+1}, Alist(0)^{n+1}\}, \{Alist(0)^{n+1}, Alist(1)^{n+1}\}) \\ &\text{If } n = d\ ,\ \{Alist^n\} = \{Alist\}\end{aligned} \tag{4.2.10}$$

$$\begin{aligned}&\text{If } n < d\ ,\ \{Blist^n\} = \\ &\quad (\{Blist(0)^{n+1}, Blist(1)^{n+1}\}, \{Blist(0)^{n+1}, Blist(1)^{n+1}\}) \\ &\text{If } n = d\ ,\ \{Blist^n\} = \{Blist\}\end{aligned} \tag{4.2.11}$$

4.2.5.2 Iterative construction of *Alist* and *Blist*

In order to extract a construction, an analysis of these recursive definitions is performed. Figures 4.4 and 4.5 shows the development of these recursive formula in a 3-dimensional space.

Let us now analyze these formulas. Firstly, for any recursion level on *Alist* and *Blist*, the number of labels is multiplied by 2. Hence, the number p of products for a d—dimensional space is the following:

$$p = 2^d \tag{4.2.12}$$

Moreover, we observe that each sub-tree is suffixed by the same binary word. We also observe, and we can show that the obtained labels are composed of the binary words whose length n is ranging from $(\underbrace{00\cdots00}_{n})$ to $(\underbrace{11\cdots11}_{n})$ that is to say from 0 to $2^d - 1$.

Finally, the construction of labels of *Alist* is extracted from Equation (4.2.10). We first define a

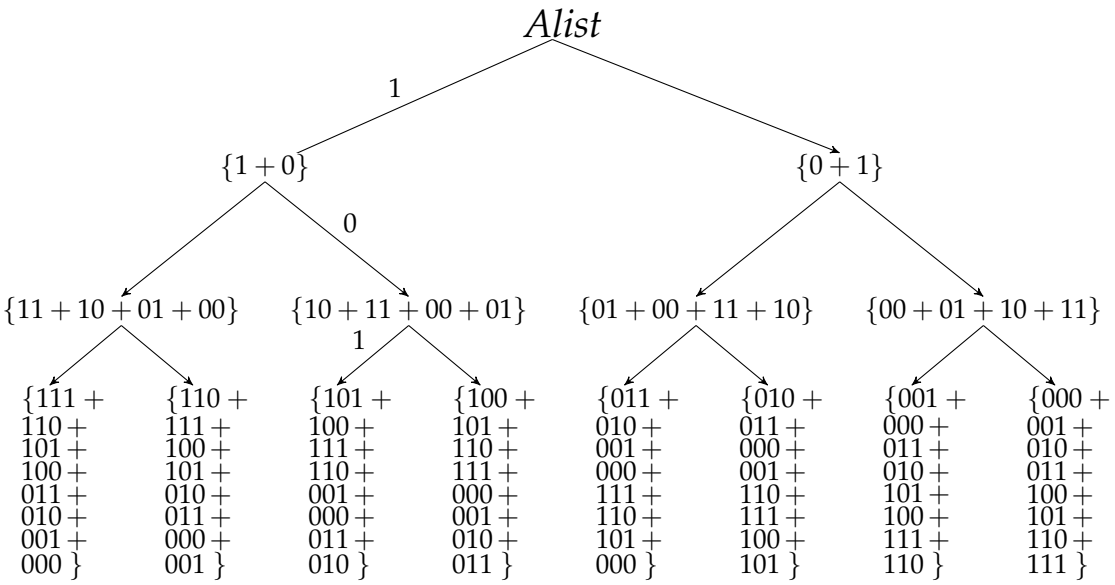


FIGURE 4.4: *Alist* development in a 3—dimensional space.

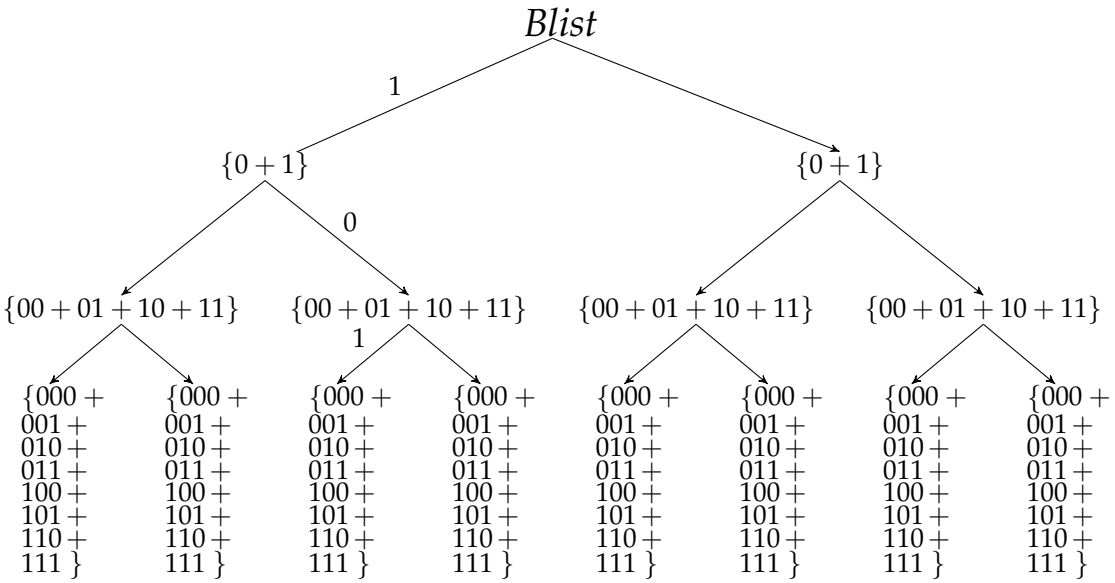


FIGURE 4.5: *Blist* development in a 3-dimensional space.

list of labels *Rlist*, composed of consecutive binary words whose length is n and ranging from $(\underbrace{00\cdots00}_{n})$ to $(\underbrace{11\cdots11}_{n})$. Therefore, the labels of *Alist* are determined by the bitwise XOR logical operation ($\oplus$ operator) between elements of *Rlist* and the binary word of the considered node. For example, the construction of *Alist* for the node 101 is presented in Figure 4.6.

The method to construct the geometric product leads to Algorithm 9. The computation of the sign is adapted from Algorithm 8 and shown in Algorithm 9.

Algorithm 9: Geometric product corresponding to a coefficient $\mathfrak{c}_k$

Data: $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$: multivectors,
Alist, *Blist*: sequences of binary words,
k: label of a coefficient of the multivector $\mathfrak{c}$,
d: dimension
Result: $\mathfrak{c}_k$: k^{th} coefficient of the multivector $\mathfrak{c} = \mathfrak{a} * \mathfrak{b}$

```
1  Alist ← { }
2  Blist ← { }
3  for binaryWords from 0 to 2^d − 1 do
4      Alist.push(binaryWords ⊕ k)
5      Blist.push(binaryWords)
6  Sign ← SignGeo(1, 0, k, d, 1)  c_k ← 0
7  i ← 0
8  foreach label u, v of Alist and Blist do
9      c_k ← c_k + Sign[i] · a_u · b_v
10     i ← i + 1
11
12 Function SignGeo
       Input:
           CurrentSign: sign
           level:Integer
           k:coefficient
           d: dimension
           comp: Integer
       Result: Sign: sequence of signs
13     if level==d then
14         Sign = Sign.push(CurrentSign)
15     else
16         b ← level^th bit of k
17         if b==1 then
18             SignGeo(comp ∗ currentSign, level + 1, k, d , comp)
19             SignGeo(currentSign, level + 1, k, d , −comp)
20         else
21             SignGeo(comp ∗ currentSign, level + 1, k, d , −comp)
22             SignGeo(currentSign, level + 1, k, d , comp)
```

Rlist	k	*Alist*
000	101	$101 \oplus 000 = 101$
001	101	$101 \oplus 001 = 100$
010	101	$101 \oplus 010 = 111$
011	101	$101 \oplus 011 = 110$
100	101	$101 \oplus 100 = 001$
101	101	$101 \oplus 101 = 000$
110	101	$101 \oplus 110 = 011$
111	101	$101 \oplus 111 = 010$

FIGURE 4.6: Construction of *Rlist* and *Alist* for the node 101.

4.2.5.3 Non-Euclidean space

In this section, we show the construction of the geometric product for a non-Euclidean space. We assume that the basis used is orthogonal. If non-orthogonal basis is needed, then a change of basis can be performed, similarly to what is explained in [23]. In order to construct the geometric product, the quadratic form ϕ is required. The representation of the quadratic used in the sequel is picked up from [36]. The values of this quadratic form are represented with d-tuple. We aim now at determining the construction of the geometric product with this quadratic form. In [36], Fuchs and Théry define the geometric product with the quadratic form as follows:

$$\begin{aligned}
&\mathfrak{a}_u^n * \mathfrak{b}_v^n = (\mathfrak{a}_{u1}^{n+1}, \mathfrak{a}_{u0}^{n+1})^n * (\mathfrak{b}_{v1}^{n+1}, \mathfrak{b}_{v0}^{n+1})^n \\
&\text{if } n < d \text{ , } \mathfrak{a}_u^n * \mathfrak{b}_v^n = \\
&\quad (\mathfrak{a}_{u1}^{n+1} * \mathfrak{b}_{v0}^{n+1} + \overline{\mathfrak{a}}_{u0}^{n+1} * \mathfrak{b}_{v1}^{n+1} \text{ , } \mathfrak{a}_{u0}^{n+1} * \mathfrak{b}_{v0}^{n+1} + \phi_{n+1}\overline{\mathfrak{a}}_{u1}^{n+1} * \mathfrak{b}_{v1}^{n+1})^n \\
&\text{if } n = d \text{ , } \mathfrak{a}_u^n * \mathfrak{b}_v^n = \mathfrak{a}_u^d * \mathfrak{b}_v^d
\end{aligned} \tag{4.2.13}$$

where $\phi_{n+1} = \phi(\mathbf{e}_{n+1}, \mathbf{e}_{n+1})$ denotes the $(n+1)^{th}$ element of this quadratic form, and represents the squared of $\mathbf{e}_{n+1}$.

The sequence of labels of $\mathfrak{a}$ and $\mathfrak{b}$ remain the same. Therefore, we determine the sequence of ϕ in the binary tree. Let *ϕlist* denote this construction. This construction is computed with a list of elements of ϕ in an equivalent way as the sequence of labels. It is extracted from Equation (4.2.13)

and defined as follows:

$$\begin{aligned} &\text{If } n < d \ , \ \{\phi list^n\} = \\ &\quad (\{\phi list^{n+1}, \phi list^{n+1} \ \}, \{\phi list^{n+1}, \phi^{n+1}.\phi list(1)^{n+1} \ \}) \\ &\text{If } n = d \ , \ \{\phi list^n\} = \ \{\phi list\} \end{aligned} \tag{4.2.14}$$

Algorithm 10 shows the pseudo-code for the construction of this sequence. This construction is inserted in the implementation of the geometric product.

Algorithm 10: Geometric product corresponding to a coefficient $\mathfrak{c}_k$

```
1  Function quadraticForm
       Input:
           currentCoef: coefficient
           metric: list of coefficients
           level: Integer
           k: coefficient
           d: dimension
       Result: seq: sequence of metric elements for a leaf c_k
2      if level==d then
3          seq.push(CurrentCoef)
4      else
5          φ ← level^th element of metric
6          b ← level^th bit of k
7          if b==1 then
8              quadraticForm(currentCoef, metric, level + 1, k , d)
9              quadraticForm(currentCoef, metric, level + 1, k , d)
10         else
11             quadraticForm(φ * currentCoef, metric, level + 1, k , d)
12             quadraticForm(currentCoef, metric, level + 1, k , d)
```

4.2.6 Discussions

These sections presented our first method to compute Geometric Algebra products, defined in any dimension. We show that the proposed method has algorithms to compute elements in parallel. These algorithms have better complexity than state-of-the-art Geometric Algebra methods, namely in $\mathcal{O}(3^d)$ instead of $\mathcal{O}(d \times 4^d)$. However, there are major drawbacks.

Firstly, some recursive calls are useless in the construction of these lists. This is due to the fact that the binary tree and the binary tree list do not encode the graded structure of the Geometric Algebra very well. In fact, at the same depth of the tree, one might have nodes whose grade

ranges from 0 to the dimension d.

Secondly, these products use list and new data structures. To use these lists, memory allocations have to be performed. Furthermore, these memory allocations are in $\mathcal{O}(2^d)$.

To fix this second issue, we propose a second approach presented in the next paragraph.

4.3 Automata speed up

The multivectors product takes the following form:

$$\mathfrak{a} \wedge \mathfrak{b} = \left(\mathfrak{a}_1, \mathfrak{a}_0\right) \wedge \left(\mathfrak{b}_1, \mathfrak{b}_0\right) \tag{4.3.1}$$

where the subscript $\mathfrak{a}_0$ or $\mathfrak{a}_1$ respectively refers to the right and left sub-tree of $\mathfrak{a}$. The recursive outer product starts with the iteration $n = 0$ and is defined as:

$$\begin{aligned} &\text{if } n < d,\ \mathfrak{a}^n \wedge \mathfrak{b}^n = \left(\mathfrak{a}_1^{n+1} \wedge \mathfrak{b}_0^{n+1} + \overline{\mathfrak{a}_0}^{n+1} \wedge \mathfrak{b}_1^{n+1}\,, \mathfrak{a}_0^{n+1} \wedge \mathfrak{b}_0^{n+1}\right)^n \\ &\text{if } n = d,\ \mathfrak{a}^n \wedge \mathfrak{b}^n = \mathfrak{a}^d \wedge \mathfrak{b}^d \end{aligned} \tag{4.3.2}$$

where $\overline{\mathfrak{a}}$ expresses the anti-commutativity of the outer product. As previously showed , this method avoids useless products like outer product between dependent basis blade, resulting in a complexity of $\mathcal{O}(3^d)$ instead of $\mathcal{O}(d4^d)$. This complexity enhancement has a huge effect in high dimensional spaces.

To handle sparse multivectors, Fuchs and Théry proposed to use unbalanced binary trees where the subtrees leading to leaves with coefficient set to 0 are removed. A recursive call leading to nodes without sub-trees is immediately discarded, as presented in Algorithm 11.

Algorithm 11: Recursive outer product (unbalanced binary tree)

```
1  Function wedge
       Input: C: resulting binary tree, A, B: binary tree
2      if A is a leaf then
3          C += A × B                                  // (× is the product of scalars)
4      else
5          if A.hasLeftChild() and B.hasRightChild() then
6              wedge(C.leftChild,A.leftChild,B.rightChild)
7          if A.hasRightChild() and B.hasLeftChild() then
8              wedge(C.leftChild,A.rightChild,B.leftChild)
9          if A.hasRightChild() and B.hasRightChild() then
10             wedge(C.rightChild,A.rightChild,B.rightChild)
```

Note that for clarity purpose, Algorithm 11 omits sign computations.

4.3.1 Recursive method revisited

This section focuses on the recursive method and more specifically on how to minimize the number of conditionals in the recursive products. Indeed, a recursive call from a recursion level to the next involves some tests on the existence of sub-trees, i.e. line 5, 7 and 9 of Algorithm 11 for the outer product. We express this set of conditions as a deterministic finite automaton with boolean transitions according to the existence of left subtree $\mathfrak{a}_1$ and right subtree $\mathfrak{a}_0$ of the current node $\mathfrak{a}$ (respectively for $\mathfrak{b}$). As an example, the automaton of the outer product is depicted in Figure 4.7. To each final state of the automaton, or equivalently to each element of $\{5,6,8,9,10\}$, of the automaton in Figure 4.7, is associated a subset of the set of products of equation (4.3.2), except for state 5 where no product is required. Let us list this subset of products for each final state. As

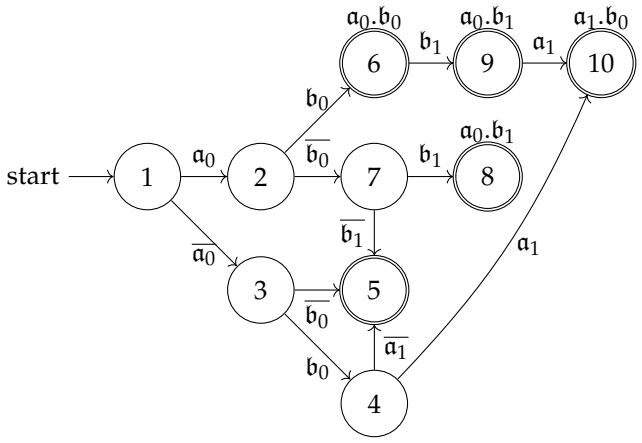


FIGURE 4.7: Automaton of the set of products of the outer product from an recursion level to the next.

previously stated, no products are required for the final state 5. The product $\mathfrak{a}_0 \wedge \mathfrak{b}_0$ is required for the states $\{6,9,10\}$. We need to compute $\mathfrak{a}_0 \wedge \mathfrak{b}_1$ for the states $\{8,9,10\}$. Finally, the product $\mathfrak{a}_1 \wedge \mathfrak{b}_0$ is needed for the state $\{10\}$.

We remark that in the worst case four conditionals are required but less in the average case. For example, if we consider that $\mathfrak{a}_0$ and $\mathfrak{b}_0$ are non-null binary tree whereas $\mathfrak{a}_1$ and $\mathfrak{b}_1$ are null binary sub-trees, then the control flow will be formed of two conditionals instead of four.

4.3.2 Discussion

Compared to the first presented method, this approach does not require the storage of any lists. Moreover, the complexity obtained is also in $\mathcal{O}(3^d)$. However, this approach presented drawbacks:

- as seen in Algorithm 11, the two multivectors have to be binary trees. Hence, the data structure used for the multivectors have to be binary trees. To be consistent with the presented data structure, this would require converting back and forth between binary tree and per-grade data structure. This latter computation is computationally expensive, more precisely in $\mathcal{O}(2^d)$.
- furthermore, the Algorithm 11 does not depend on the structure of the multivector (grade of each multivector).

In order to fix these two issues and the issues presented at the end the last section, we proposed new algorithms presented in the next paragraphs.

4.4 Prefix tree approach

In the previous sections, we first presented some parallel algorithms to compute the products of Geometric Algebra. We then detailed how a binary tree can represent efficiently multivector components and lead to a recursive formulation of the products used in GA for high dimensions.

In the following sections, we introduce a variation of this formulation, using a prefix tree [16] that presents some interesting properties leading to very efficient optimization in recursive GA products. Moreover, this prefix tree formulation also includes a natural dual multivector representation well suited to an efficient dual computation algorithm, particularly useful for high dimensions.

To summarize, our purposes for this prefix tree are

- to have complexity in $\mathcal{O}(3^d)$ for the outer product and $\mathcal{O}(4^d)$ for the geometric product (instead of $\mathcal{O}(d4^d)$ for the state-of-the-art method,
- to not require any additional data structure to store the products in order to be memory efficient,

In the worst case,we proved in the last section that the number of recursive calls of the binary tree products is:

$$\sum_{i=1}^{d} 3^i = \frac{3}{2}\left(3^d - 1\right) \tag{4.4.1}$$

Due to the complexity, this approach avoids structural zero. However, it failed to efficiently handling object zero. To fix this issue, we propose new recursive products over the prefix tree structure previously defined. In practice, we aim at having 3^d recursive calls in the worst case instead of $\frac{3}{2}\left(3^d - 1\right)$.

4.4.1 Multivectors and prefix trees

As previously explained, a multivector can be represented by a binary tree. This approach can be modified to represent a multivector with a prefix tree, where each node is associated to a basis vector. More precisely, the nodes of depth k of the tree represent all the basis vector of grade k. Thus, the root node, denoted by $\mathbf{1}$, represents the scalar part of a multivector, the children of the root node correspond to the vectors contributions to the multivector, the children of those nodes are the bivectors contributions to the multivector, and so on. Moreover, by nature of the

prefix tree structure, each label of basis vector of the tree is prefixed by the labels its parents, as illustrated in figure 4.8, where $\mathbf{e}_{ijk}$ stands for $\mathbf{e}_i \wedge \mathbf{e}_j \wedge \mathbf{e}_k$. Note that the scalar $\mathbf{1}$ is considered as the prefix of all the other nodes.

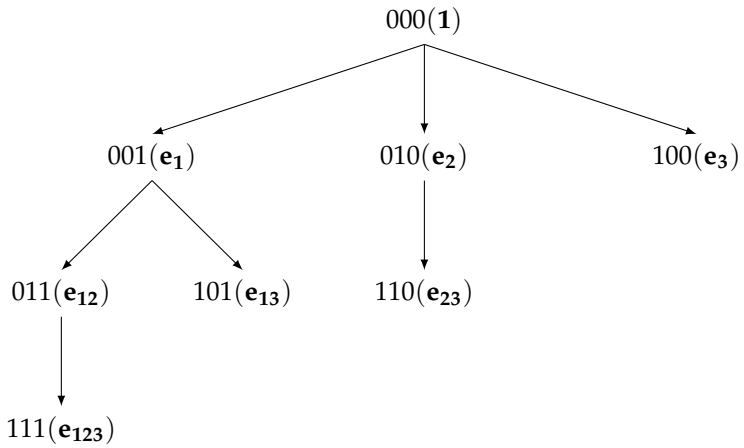


FIGURE 4.8: Prefix tree data structure. Each node of the tree is labelled by a binary index representing a basis vector of the algebra, here in a 3-dimensional vector space.

Each node of the tree is labelled by a binary word encoding the basis vector representing the node. This binary labelling, where each bit represents a basis vector, is used in numerous GA implementations [30, 12, 34]. Traversing a prefix tree representing a multivector can be achieved using an index called `msb` (most significant blade). This index represents the binary label of the basis vector (of grade 1) following (canonical ordering of the basis) the last basis vector encountered from the reached node. Thus, this index actually contains only a single bit to 1. Consequently, the label of a child of a node with binary identifier `u` and index `msb` is computed by:

$$\text{child\_label}(\texttt{u}, \texttt{msb}) = \texttt{u} + \texttt{msb} \tag{4.4.2}$$

where $+$ is the binary addition. In this formulation, the contribution of `msb` corresponds to the most significant bit of child_label(`label`,`msb`). Finally, note that the most significant bit of the resulting label corresponds to the position of the 1-bit of `msb`.

Then, labels of the siblings of this child can be easily computed by means of left-shifting of `msb`. This labelling is shown Figure 4.9.

We would like to express the operations of the algebra using this prefix tree representation. This requires expressing two multivectors with the prefix tree representation. Just like we expressed

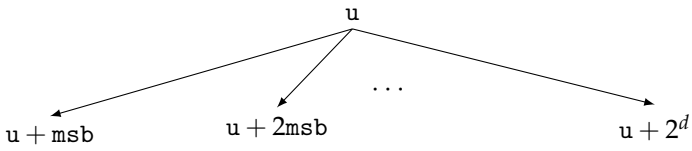


FIGURE 4.9: The labelling of the prefix tree at one depth

a multivector as a binary tree with recursive formalism, we seek for an expression of the Geometric Algebra with prefix tree. We consider a multivector $\mathbf{a}$ in $\mathbb{G}_{p,q}$ where $p+q=d$, we note a component of $\mathbf{a}$ at index $\mathtt{u}$ in the prefix tree formulation as $a_{\mathtt{u}}$. The children of $a_{\mathtt{u}}$ in the prefix tree formulation are shown in Figure 4.10.

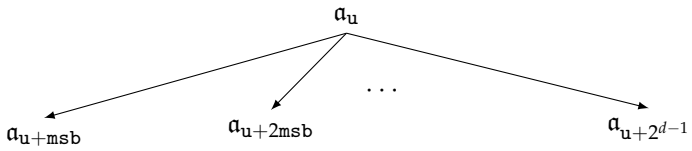


FIGURE 4.10: Multivector prefix tree representation.

For clarity purpose, we will also use the compact notation as follows:

$$\left(a_{\mathtt{u}}, \left(a_{\mathtt{u}+\mathtt{msb}}, a_{\mathtt{u}+2\mathtt{msb}}, \cdots, a_{\mathtt{u}+2^{d-1}}\right)\right), \tag{4.4.3}$$

Definition 4.4.1.

Let $a_{\mathtt{u}}$ be a node of the prefix tree. Let us assume that the basis vector associated with $\mathtt{msb}$ is $\mathbf{e}_i$, $i \in 1, \cdots, d$. Then, the interpretation of the prefix tree $\mathbf{a}_{\mathtt{u}}$ into Geometric Algebra multivector is:

$$a_{\mathtt{u}} + \mathbf{e}_i \wedge a_{\mathtt{u}+\mathtt{msb}} + \mathbf{e}_{i+1} \wedge a_{\mathtt{u}+2\mathtt{msb}} + \cdots + \mathbf{e}_d \wedge a_{\mathtt{u}+2^{d-1}} \tag{4.4.4}$$

As an example, the compact notation of the prefix tree in a 3-dimensional vector space and at one recursion depth results in:

$$\left(a_{000}, \left(a_{001}, a_{010}, a_{100}\right)\right) \tag{4.4.5}$$

The development after a recursive depth of $d=3$ yields:

$$\left(a_{000}, \left(a_{001}, \left(a_{011}, \left(a_{111}\right), a_{101}\right), \left(a_{010}, \left(a_{110}\right)\right), a_{100}\right)\right) \tag{4.4.6}$$

Using the interpretation of the tree given in Equation (4.4.4) results in:

$$\begin{aligned} & \mathfrak{a}_{000} + \mathbf{e}_1 \wedge \Big(\mathfrak{a}_{001} + \mathbf{e}_2 \wedge \big(\mathfrak{a}_{011} + \mathbf{e}_3 \wedge (\mathfrak{a}_{111})\big) + \mathbf{e}_3 \wedge \mathfrak{a}_{101}\Big) + \mathbf{e}_2 \wedge \Big(\mathfrak{a}_{010} + \mathbf{e}_3 \wedge \mathfrak{a}_{110}\Big) \\ & + \mathbf{e}_3 \wedge \mathfrak{a}_{100} \\ = \; & \mathfrak{a}_{000} + \mathfrak{a}_{001}\mathbf{e}_1 + \mathfrak{a}_{011}\mathbf{e}_1 \wedge \mathbf{e}_2 + \mathfrak{a}_{111}\mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3 + \mathfrak{a}_{101}\mathbf{e}_1 \wedge \mathbf{e}_3 + \mathfrak{a}_{010}\mathbf{e}_2 + \mathfrak{a}_{110}\mathbf{e}_2 \wedge \mathbf{e}_3 \\ & + \mathfrak{a}_{100}\mathbf{e}_3 \end{aligned} \tag{4.4.7}$$

This corresponds to a general multivector in a 3-dimensional vector space.

Definition 4.4.1 is the support for us to define a mapping between the prefix tree and the binary tree frameworks defined bellow.

Definition 4.4.2. Let ψ be the mapping associating a prefix tree multivector:

$$\Big(\mathfrak{a}_{\mathtt{u}}, \big(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^d-1}\big)\Big) \tag{4.4.8}$$

to its counterpart in the binary tree framework as the labelled binary tree:

$$\Big(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}, \Big(\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}, \big(\cdots(\mathfrak{a}_{\mathtt{uu}+2^d-1}, \mathfrak{a}_{\mathtt{u}})\big)\Big)\Big) \tag{4.4.9}$$

Proposition 4.4.3. *The interpretation of the binary tree given in Equation 4.4.9 results in interpretation of Equation 4.4.4.*

Proof. Let us consider the pair:

$$\Big(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{a}\Big), \tag{4.4.10}$$

where

$$\mathfrak{a} = \Big(\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}, \big(\cdots(\mathfrak{a}_{\mathtt{uu}+2^d-1}, \mathfrak{a}_{\mathtt{u}})\cdots\big)\Big) \tag{4.4.11}$$

Assumption of Definition 4.4.1 denotes that the basis vector associated with $\mathtt{msb}$ is $_i$, $i \in 1, \cdots, d$. The interpretation of Section 3 of [36] means that the pair composed of $\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}$ as a left sub-tree can be written as:

$$\mathbf{e}_i \wedge \mathfrak{a}_{\mathtt{u}+\mathtt{msb}} + \mathfrak{a} \tag{4.4.12}$$

By reitering the same computation for the nested pairs in $\mathfrak{x}$ yields:

$$\mathbf{e}_i \wedge \mathfrak{a}_{\mathtt{u}+\mathtt{msb}} + \mathbf{e}_{i+1} \wedge \mathfrak{a}_{\mathtt{u}+2\mathtt{msb}} + \cdots + \mathbf{e}_d \wedge \mathfrak{a}_{\mathtt{u}+2^d-1} + \mathfrak{a}_{\mathtt{u}} \tag{4.4.13}$$

This corresponds to Equation 4.4.4. □

We are now left to recursively define the vector space operations by starting with the definition of the recursive formula of the addition between two prefix tree multivector.

Proposition 4.4.4. *Let us consider two multivectors* $\mathbf{a}$ *and* $\mathbf{b}$ *whose recursive construction are respectively:*

$$\Big(\mathfrak{a}_{\mathtt{u}}, (\mathfrak{a}_{\mathtt{u+msb}}, \mathfrak{a}_{\mathtt{u+2msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^{d-1}})\Big), \tag{4.4.14}$$

and

$$\Big(\mathfrak{b}_{\mathtt{u}}, (\mathfrak{b}_{\mathtt{u+msb}}, \mathfrak{b}_{\mathtt{u+2msb}}, \cdots, \mathfrak{b}_{\mathtt{u}+2^{d-1}})\Big). \tag{4.4.15}$$

The recursive construction of the addition of these two multivectors can be computed as:

$$\begin{aligned} & \Big(\mathfrak{c}_{\mathtt{u}}, (\mathfrak{c}_{\mathtt{u+msb}}, \mathfrak{c}_{\mathtt{u+2msb}}, \cdots, \mathfrak{c}_{\mathtt{u}+2^{d-1}})\Big) \\ = \; & \Big(\mathfrak{a}_{\mathtt{u}}, (\mathfrak{a}_{\mathtt{u+msb}}, \mathfrak{a}_{\mathtt{u+2msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^{d-1}})\Big) + \Big(\mathfrak{b}_{\mathtt{u}}, (\mathfrak{b}_{\mathtt{u+msb}}, \mathfrak{b}_{\mathtt{u+2msb}}, \cdots, \mathfrak{b}_{\mathtt{u}+2^{d-1}})\Big) \\ = \; & \Big(\mathfrak{a}_{\mathtt{u}} + \mathfrak{b}_{\mathtt{u}}, (\mathfrak{a}_{\mathtt{u+msb}} + \mathfrak{b}_{\mathtt{u+msb}}, \mathfrak{a}_{\mathtt{u+2msb}} + \mathfrak{b}_{\mathtt{u+2msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^{d-1}} + \mathfrak{b}_{\mathtt{u}+2^{d-1}})\Big) \end{aligned} \tag{4.4.16}$$

Proof. We use the interpretation of the prefix tree defined in Equation 4.4.4. The operation to be performed is:

$$\begin{aligned} & \mathfrak{a}_{\mathtt{u}} + \mathbf{e}_i \wedge \mathfrak{a}_{\mathtt{u+msb}} + \mathbf{e}_{i+1} \wedge \mathfrak{a}_{\mathtt{u+2msb}} + \cdots + \mathbf{e}_d \wedge \mathfrak{a}_{\mathtt{u}+2^{d-1}} + \mathfrak{b}_{\mathtt{u}} + \mathbf{e}_i \wedge \mathfrak{b}_{\mathtt{u+msb}} \\ + \; & \mathbf{e}_{i+1} \wedge \mathfrak{b}_{\mathtt{u+2msb}} + \cdots + \mathbf{e}_d \wedge \mathfrak{b}_{\mathtt{u}+2^{d-1}} \end{aligned} \tag{4.4.17}$$

The distributive property of the outer product yields:

$$\begin{aligned} & \mathfrak{a}_{\mathtt{u}} + \mathfrak{b}_{\mathtt{u}} + \mathbf{e}_i \wedge (\mathfrak{a}_{\mathtt{u+msb}} + \mathfrak{b}_{\mathtt{u+msb}}) + \mathbf{e}_{i+1} \wedge (\mathfrak{a}_{\mathtt{u+2msb}} + \mathfrak{b}_{\mathtt{u+2msb}}) + \cdots + \mathbf{e}_d \wedge (\mathfrak{a}_{\mathtt{u}+2^{d-1}} + \mathbf{e}_i \wedge \mathfrak{b}_{\mathtt{u+msb}} \\ + \; & \mathbf{e}_{i+1} \wedge \mathfrak{b}_{\mathtt{u+2msb}} + \cdots + \mathbf{e}_d \wedge \mathfrak{b}_{\mathtt{u}+2^{d-1}} \end{aligned} \tag{4.4.18}$$

Finally, by identification, the above formula is the interpretation of the prefix tree multivector:

$$\Big(\mathfrak{a}_{\mathtt{u}} + \mathfrak{b}_{\mathtt{u}}, (\mathfrak{a}_{\mathtt{u+msb}} + \mathfrak{b}_{\mathtt{u+msb}}, \mathfrak{a}_{\mathtt{u+2msb}} + \mathfrak{b}_{\mathtt{u+2msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^{d-1}} + \mathfrak{b}_{\mathtt{u}+2^{d-1}})\Big) \tag{4.4.19}$$

□

Proposition 4.4.5. *The recursive construction of the multiplication of one multivector* $\mathbf{a}$ *by a scalar* $\lambda \in \mathbb{R}$ *can be computed as:*

$$\begin{aligned} & \Big(\mathfrak{c}_{\mathtt{u}}, (\mathfrak{c}_{\mathtt{u+msb}}, \mathfrak{c}_{\mathtt{u+2msb}}, \cdots, \mathfrak{c}_{\mathtt{u}+2^{d-1}})\Big) \\ = \; & \lambda\Big(\mathfrak{a}_{\mathtt{u}}, (\mathfrak{a}_{\mathtt{u+msb}}, \mathfrak{a}_{\mathtt{u+2msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^{d-1}})\Big) \\ = \; & \Big(\lambda\mathfrak{a}_{\mathtt{u}}, (\lambda\mathfrak{a}_{\mathtt{u+msb}}, \lambda\mathfrak{a}_{\mathtt{u+2msb}}, \cdots, \lambda\mathfrak{a}_{\mathtt{u}+2^{d-1}})\Big) \end{aligned} \tag{4.4.20}$$

Proof. Once more, we use the interpretation of the prefix tree defined in Equation 4.4.4. The operation to be performed is:

$$\lambda\left(\mathfrak{a}_{\mathtt{u}} + \mathbf{e}_i \wedge \mathfrak{a}_{\mathtt{u}+\mathtt{msb}} + \mathbf{e}_{i+1} \wedge \mathfrak{a}_{\mathtt{u}+2\mathtt{msb}} + \cdots + \mathbf{e}_d \wedge \mathfrak{a}_{\mathtt{u}+2^{d-1}}\right) \tag{4.4.21}$$

The distributive property of both the scalar multiplication and the outer product yields:

$$\lambda\mathfrak{a}_{\mathtt{u}} + \mathbf{e}_i \wedge \lambda\mathfrak{a}_{\mathtt{u}+\mathtt{msb}} + \mathbf{e}_{i+1} \wedge \lambda\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}} + \cdots + \mathbf{e}_d \wedge \lambda\mathfrak{a}_{\mathtt{u}+2^{d-1}} \tag{4.4.22}$$

Thus, by identification, this results in:

$$\left(\lambda\mathfrak{a}_{\mathtt{u}}, (\lambda\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}, \lambda\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \lambda\mathfrak{a}_{\mathtt{u}+2^{d-1}})\right) \tag{4.4.23}$$

□

We can note that this tree representation is not very well suited for an efficient data storage due to the difficulty to cut useless parts of the tree. Therefore, we include a mapping from the tree representation to the array structure defined in section 2.2.1. This mapping consists in two precomputed lookup tables that extract both the grade and position on the array of a given label. Note that the size of this lookup table is 2^d.

In terms of pseudo-code, the traversal of this structure is defined in the Algorithm 12.

Algorithm 12: Recursive traversing of a multivector **a** whose grade is k_a

```
1 Function traverse
      Input: a: the multivector to be traversed,
             f: function,
             k_a: the grade of the multivector.
             label_a: the recursive position on each tree.
2     if grade(label_c) == k_c then                    // end of recursion
3         f[label_a]
4     else                                             // recursive calls
5         msb_a = labelToMsb(label_a)
6         foreach msb such that gradeKReachable(k_a, msb)== true do
7             label = label_a + msb
8             if gradeKReachable(k_a, msb) then
9                 traverse(a, k_a, label_a + msb)
```

In this pseudo-code, `labelToMsb(label)` computes the most significant bit from the considered label. The function `gradeKReachable(grade,msb)` indicates whether at least a child of a node reached by reading the basis vector '`msb`' can reach the grade '`grade`'. This function is used to

avoid several recursive calls, as shown on Figure 4.11. Furthermore, as the dimension grows, this number of useless recursive calls grows exponentially.

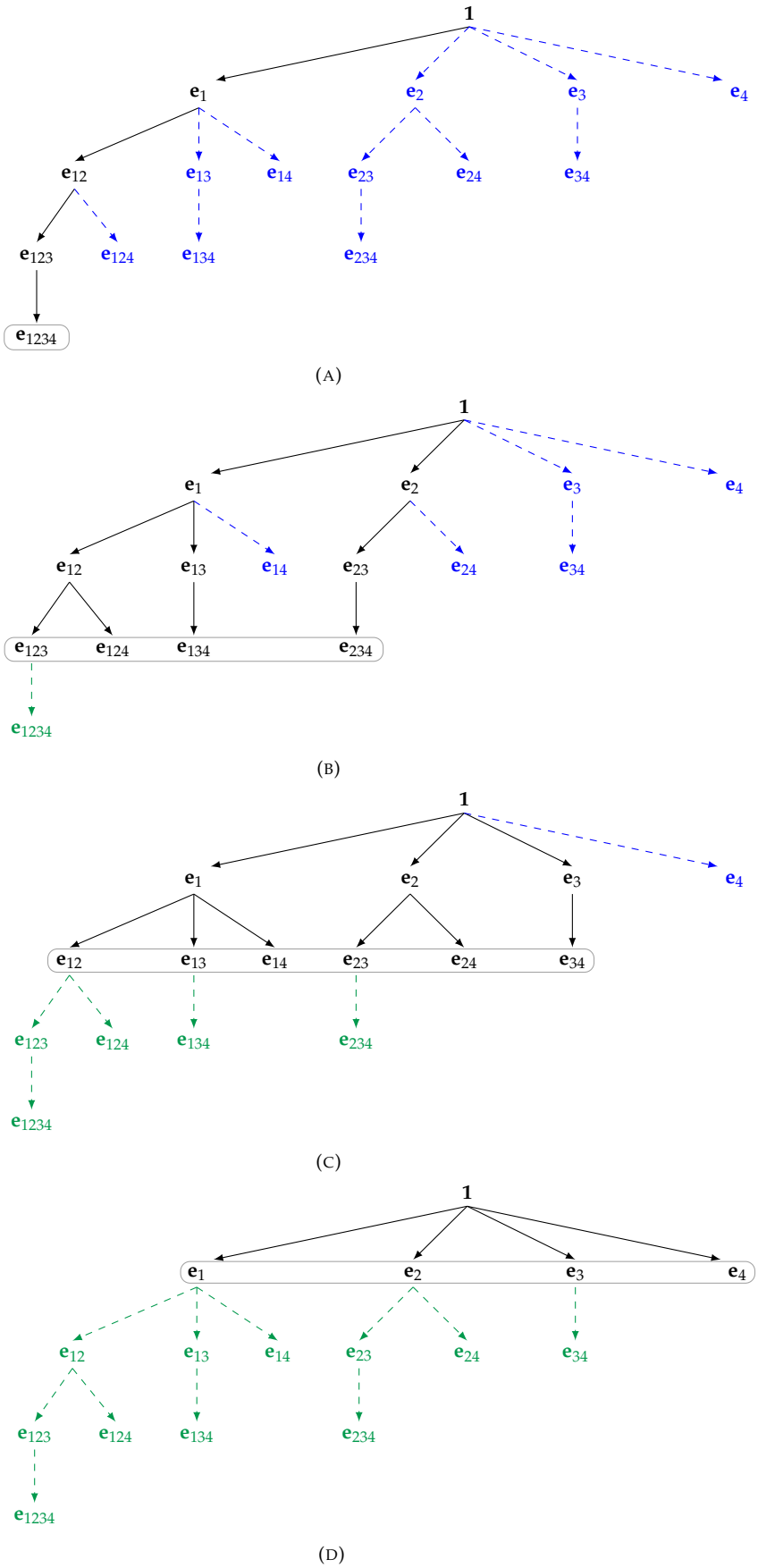


FIGURE 4.11: Tree structure for some resulting multivectors of grade 4 (A), grade 3 (B), grade 2 (C), grade 1 (D) in a 4-dimensional vector space. Useless branches are depicted in green dashed arrows above the targeted multivector and in blue below. The targeted nodes are surrounded by a black rectangle. We can remark that over 15 theoretic traversals, 11 useless traversals are ignored in (A), 6 useless traversals are ignored in (B), 6 useless traversals are ignored in (C) and 11 useless traversals are ignored in (D).

Let us consider:

- a label called *label* that is to say a node of the prefix tree
- a label called *msb* that represents the index of the last traversed basis vector
- an integer k representing the considered grade (might be different from the grade of *label*)

By definition, at each recursive depth, the grade is incremented. Furthermore, if $\texttt{label} + msb$ corresponds to the first child of the current node. After 2 recursive calls, the label of the new node is lower bounded by (left-most child):

$$\texttt{label} + msb + 2msb, \tag{4.4.24}$$

and the grade was increased. The grade of the label is $grade(label) + 2$ and the index of the last traversed vector is simply $2msb$. After 3 recursive calls, this label is lower bounded by

$$\texttt{label} + msb + 2msb + 4msb \tag{4.4.25}$$

and the grade was again increased. The grade of the label is $grade(label) + 3$ and the index of the last traversed vector is $4msb$. There seems to be a general formula for the lower bound of the label of the children after l recursive calls as:

$$\texttt{label} + \sum_{i=0}^{l-1} 2^i msb \tag{4.4.26}$$

This can be also rewritten as:

$$\texttt{label} + msb \sum_{i=0}^{n-1} 2^i, n \in [1, d] \tag{4.4.27}$$

The grade of the label seems to be $grade(label) + n$ and the index of the last traversed vector is $2^{n-1}msb$.

Let us prove it by induction.

The base case holds. In fact, after one recursive call the child of the label has label:

$$\texttt{label} + msb = \texttt{label} + msb \sum_{i=0}^{0} 2^i \tag{4.4.28}$$

This corresponds to the first child thus to a lower bound. The grade of the label is $grade(label) + 1$ and the index of the last traversed vector is *msb*, as seen in the definition.

Let us assume that the formula 4.4.27 holds for a number of recursive calls noted m. Thus, the

label of the children of the current node is lower bounded by:

$$\texttt{label} + msb \sum_{i=0}^{m-1} 2^i \tag{4.4.29}$$

By definition of the child of a node, the last traversed vector is $2^m msb$. Thus the left-most child of the node after m recursive traversals is defined as:

$$\texttt{label} + msb \sum_{i=0}^{m-1} 2^i + 2^m msb \tag{4.4.30}$$

This can be rewritten as:

$$\texttt{label} + msb \sum_{i=0}^{(m+1)-1} 2^i \tag{4.4.31}$$

As the grade after m recursive calls is $\texttt{grade}(label) + m$ and we computed the child of this label thus the grade was incremented and the grade of the leading node is $\texttt{grade}(label) + m + 1$.
Thus, the formula 4.4.27 holds for a $m + 1$ recursive calls. Finally, by induction, this formula holds for any number of recursive calls.

Furthermore, the term $\sum_{i=0}^{(m+1)-1} 2^i$ is known to be geometric series whose first term is 1 and its common ratio is 2. The general formula of such geometric series is given as:

$$\begin{aligned} \textstyle\sum_{i=0}^{(m+1)-1} 2^i &= \frac{1-2^n}{1-2} \\ &= 2^n - 1 \end{aligned} \tag{4.4.32}$$

Finally, the formula 4.4.27 can be rewritten as:

$$\texttt{label} + msb(2^n - 1) \tag{4.4.33}$$

Note that this computation can be computed in a constant time $\mathcal{O}(1)$.
Furthermore, we know the upper bound of the labels in a d-dimensional vector space. This simply corresponds to the label $11\cdots1$ or equivalently to the pseudo-scalar. Thus, in a d-dimensional space, all labels are upper bounded by the label:

$$2^d \tag{4.4.34}$$

This means that if the lower bound $\texttt{label} + msb(2^n - 1)$ exceeds this upper bound of the labels then this is not a label of the vector space, meaning that we have a constant time function to know

whether after n-recursive, there exists a child of the current node in the vector space labels. Furthermore, a node *label* whose grade is $grade(label)$ will require exactly $k - grade(label)$ to reach a node whose grade is k. And this will require exactly, $k - grade(label)$ recursive calls. This latter result along with the upper bound of formula 4.4.34 and the lower bound formula 4.4.33 yields to the following inequality:

$$\texttt{label} + msb(2^{k-grade(label)} - 1) < 2^d \tag{4.4.35}$$

Formula 4.4.35 is a way to know whether there exists a child of `label` with k as a grade which is reachable. The associated function $\texttt{gradeKReachable}(k, \texttt{label}, \texttt{msb})$ is defined by Algorithm 13.

Algorithm 13: check whether it exists one child of the node `label` whose grade is k.

```
1 Function gradeKReachable
      Input: label: the recursive position
             msb: a label of the last traversed vector
             k: the considered grade.
2     labelChildK ← label + msb(2^{k−grade(label)} − 1)
3     return labelChildK < 2^d
```

With this Algorithm, some branches of the trees can be left unvisited. The formula can be computed in constant time. Indeed, the computation of $2^{k-grade(label)} - 1$ only requires bit shifting (constant time), one integer multiplication and 3 additions. And when the dimension increases, this number of operations remains the same thus the algorithm is constant time.
Furthermore the decision only depends on grades and label, and does not require any allocation thus is also constant in terms of memory complexity.

4.4.2 Anti-commutativity recursive operator

In order to efficiently compute permutation required for some Geometric Algebra operators, we define the anti-commutativity operator denoted as an overline. This operator is recursively defined over the prefix tree.

4.4.2.1 Recursive formula

Proposition 4.4.6. *Let us consider one multivector* **a** *whose recursive construction is:*

$$\left(\mathfrak{a}_{\mathtt{u}}, \left(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^{d-1}}\right)\right), \tag{4.4.36}$$

The recursive construction of the anticommutativity of this multivector can be computed as:

$$\begin{aligned} & \left(\mathfrak{c}_{\mathtt{u}}, \left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{c}_{\mathtt{u}+2^{d-1}}\right)\right) \\ = & \left(\overline{\mathfrak{a}_{\mathtt{u}}}, \left(\overline{\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}}, \overline{\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}}, \cdots, \overline{\mathfrak{a}_{\mathtt{u}+2^{d-1}}}\right)\right) \\ = & \left(\overline{\mathfrak{a}_{\mathtt{u}}}, \left(-\overline{\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}}, -\overline{\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}}, \cdots, -\overline{\mathfrak{a}_{\mathtt{u}+2^{d-1}}}\right)\right) \end{aligned} \tag{4.4.37}$$

Proof. The method followed here consists in proving that the commutative diagram of the anti-commutative operator shown in (4.4.38) holds.

$$\begin{array}{ccc} \left(\mathfrak{c}_{\mathtt{u}}, \left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{c}_{\mathtt{u}+2^{d-1}}\right)\right) & \xrightarrow{\psi} & \left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \left(\mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}, \left(\cdots\left(\mathfrak{c}_{\mathtt{uu}+2^{d-1}}, \mathfrak{c}_{\mathtt{u}}\right)\right)\right)\right) \\ \downarrow - & & \downarrow - \\ \left(\overline{\mathfrak{c}_{\mathtt{u}}}, \left(\overline{\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}}, \overline{\mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}}, \cdots, \overline{\mathfrak{c}_{\mathtt{u}+2^{d-1}}}\right)\right) & \xrightarrow{\psi} & \left(\overline{\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}}, \left(\overline{\mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}}, \left(\cdots\left(\overline{\mathfrak{c}_{\mathtt{uu}+2^{d-1}}}, \overline{\mathfrak{c}_{\mathtt{u}}}\right)\right)\right)\right) \end{array} \tag{4.4.38}$$

On one hand, mapping the prefix tree $\mathfrak{c}$ of Equation (4.4.37) in the binary tree using ψ of Equation (4.4.9) results in:

$$\begin{aligned} & \left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \left(\mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}, \left(\cdots\left(\mathfrak{c}_{\mathtt{uu}+2^{d-1}}, \mathfrak{c}_{\mathtt{u}}\right)\right)\right)\right) \\ = & \left(-\overline{\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}}, \left(-\overline{\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}}, \left(\cdots\left(-\overline{\mathfrak{a}_{\mathtt{u}+2^{d-1}}}, \overline{\mathfrak{a}_{\mathtt{u}}}\right)\right)\right)\right) \end{aligned} \tag{4.4.39}$$

On the other hand, the recursive formula of the anti-commutativity defined in [36] as:

$$\begin{aligned} & \left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{c}_{\mathtt{u}}\right) \\ = & \left(-\overline{\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}}, \overline{\mathfrak{a}_{\mathtt{u}}}\right) \end{aligned} \tag{4.4.40}$$

After developing this formula at one recursion depth, this formula becomes:

$$\begin{aligned} & \left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \left(\mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}, \mathfrak{c}_{\mathtt{u}}\right)\right) \\ = & \left(-\overline{\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}}, -\overline{\left(\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}, \overline{\mathfrak{a}_{\mathtt{u}}}\right)}\right) \end{aligned} \tag{4.4.41}$$

After further developing this formula, we find:

$$\begin{aligned} & \left(\mathtt{c}_{\mathtt{u}+\mathtt{msb}}, \left(\mathtt{c}_{\mathtt{u}+2\mathtt{msb}}, \left(\cdots(\mathtt{c}_{\mathtt{u}\mathtt{u}+2^{d-1}}, \mathtt{c}_{\mathtt{u}})\right)\right)\right) \\ = & \left(-\overline{\mathtt{a}_{\mathtt{u}+\mathtt{msb}}}, \left(-\overline{\mathtt{a}_{\mathtt{u}+2\mathtt{msb}}}, \left(\cdots-\overline{\mathtt{a}_{\mathtt{u}+2^{d-1}}}, \overline{\mathtt{a}_{\mathtt{u}}}\right)\right)\right) \end{aligned} \tag{4.4.42}$$

Equations (4.4.42) and (4.4.39) are equivalent, thus the commutative diagram holds. □

4.4.3 Outer product

In the following, we consider the outer product $C = A \wedge B$. The grades of A, B, C are respectively k_A, k_B, k_C. The outer products can be computed recursively in efficiently traversing the prefix tree of A, B and C.

4.4.3.1 Recursive formula

Proposition 4.4.7. *Let us consider two multivectors A and B whose recursive constructions are respectively:*

$$\left(\mathtt{a}_{\mathtt{u}}, (\mathtt{a}_{\mathtt{u}+\mathtt{msb}}, \mathtt{a}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathtt{a}_{\mathtt{u}+2^{d-1}})\right), \tag{4.4.43}$$

and

$$\left(\mathtt{b}_{\mathtt{u}}, (\mathtt{b}_{\mathtt{u}+\mathtt{msb}}, \mathtt{b}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathtt{b}_{\mathtt{u}+2^{d-1}})\right). \tag{4.4.44}$$

The recursive construction of the outer product of these two multivectors can be computed as:

$$\begin{aligned} & \left(\mathtt{c}_{\mathtt{u}}, (\mathtt{c}_{\mathtt{u}+\mathtt{msb}}, \mathtt{c}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathtt{c}_{\mathtt{u}+2^{d-1}})\right) \\ = & \left(\mathtt{a}_{\mathtt{u}}, (\mathtt{a}_{\mathtt{u}+\mathtt{msb}}, \mathtt{a}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathtt{a}_{\mathtt{u}+2^{d-1}})\right) \wedge \left(\mathtt{b}_{\mathtt{u}}, (\mathtt{b}_{\mathtt{u}+\mathtt{msb}}, \mathtt{b}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathtt{b}_{\mathtt{u}+2^{d-1}})\right) \\ = & \Big(\mathtt{a}_{\mathtt{u}} \wedge \mathtt{b}_{\mathtt{u}}, (\mathtt{a}_{\mathtt{u}+\mathtt{msb}} \wedge \mathtt{b}_{\mathtt{u}} + \overline{\mathtt{a}_{\mathtt{u}}} \wedge \mathtt{b}_{\mathtt{u}+\mathtt{msb}}, \mathtt{a}_{\mathtt{u}+2\mathtt{msb}} \wedge \mathtt{b}_{\mathtt{u}} + \overline{\mathtt{a}_{\mathtt{u}}} \wedge \mathtt{b}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \\ & \mathtt{a}_{\mathtt{u}+2^{d-1}} \wedge \mathtt{b}_{\mathtt{u}} + \overline{\mathtt{a}_{\mathtt{u}}} \wedge \mathtt{b}_{\mathtt{u}+2^{d-1}})\Big) \end{aligned} \tag{4.4.45}$$

Proof. Again, we aim at proving that commutative diagram of the outer product holds. On the

one hand, mapping the prefix tree $\mathfrak{c}$ of Equation (4.4.45) in the binary tree using ψ of Equation (4.4.9) results in:

$$\begin{aligned} & \left(\mathfrak{c}_{u+msb}, \left(\mathfrak{c}_{u+2msb}, \left(\cdots (\mathfrak{c}_{uu+2^{d-1}}, \mathfrak{c}_u)\right)\right)\right) \\ = & \left(\mathfrak{a}_{u+msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+msb}, \left(\mathfrak{a}_{u+2msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+2msb}, \left(\cdots\right.\right.\right. \\ & \left.\left.\left.(\mathfrak{a}_{u+2^{d-1}} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+2^{d-1}}, \mathfrak{a}_u \wedge \mathfrak{b}_u)\right)\right)\right) \end{aligned} \tag{4.4.46}$$

On the other hand, the recursive formula of the outer product between two multivectors in the binary tree framework is the pair:

$$\left(\mathfrak{c}_{u+msb}, \mathfrak{c}_u\right) = \left(\mathfrak{a}_{u+msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+msb}, \mathfrak{a}_u \wedge \mathfrak{b}_u\right) \tag{4.4.47}$$

After developing this formula at one recursion depth, this formula becomes:

$$\begin{aligned} & \left(\mathfrak{c}_{u+msb}, \left(\mathfrak{c}_{u+2msb}, \mathfrak{c}_u\right)\right) \\ = & \left(\mathfrak{a}_{u+msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+msb}, \left(\mathfrak{a}_{u+2msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+2msb}, \mathfrak{a}_u \wedge \mathfrak{b}_u\right)\right) \end{aligned} \tag{4.4.48}$$

After further developing this formula, we find:

$$\begin{aligned} & \left(\mathfrak{c}_{u+msb}, \left(\mathfrak{c}_{u+2msb}, \left(\cdots (\mathfrak{c}_{uu+2^{d-1}}, \mathfrak{c}_u)\right)\right)\right) \\ = & \left(\mathfrak{a}_{u+msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+msb}, \left(\mathfrak{a}_{u+2msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+2msb}, \left(\cdots\right.\right.\right. \\ & \left.\left.\left.(\mathfrak{a}_{u+2^{d-1}} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+2^{d-1}}, \mathfrak{a}_u \wedge \mathfrak{b}_u)\right)\right)\right) \end{aligned} \tag{4.4.49}$$

Equations (4.4.49) and (4.4.46) are equivalent, thus the commutative diagram holds. □

This recursive formula is the base to develop the pseudo-code of the outer product. As a reminder, the recursive formula is defined as:

$$\begin{aligned} & \left(\mathfrak{c}_u, (\mathfrak{c}_{u+msb}, \mathfrak{c}_{u+2msb}, \cdots, \mathfrak{c}_{u+2^{d-1}})\right) \\ = & \left(\mathfrak{a}_u \wedge \mathfrak{b}_u, (\mathfrak{a}_{u+msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+msb}, \mathfrak{a}_{u+2msb} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+2msb}, \cdots,\right. \\ & \left.\mathfrak{a}_{u+2^{d-1}} \wedge \mathfrak{b}_u + \overline{\mathfrak{a}_u} \wedge \mathfrak{b}_{u+2^{d-1}})\right) \end{aligned} \tag{4.4.50}$$

We highlight the main parts of the recursive formula and their equivalent in the pseudo-code shown in Algorithm 14.

Algorithm 14: Recursive outer product $C = A \wedge B$

```
1  Function outer
       Input: A, B: two multivectors,
              C: resulting multivector,
              k_A, k_B and k_C: the respective grade of each multivector.
              label_A, label_B, label_c: the recursive position on each tree.
              sign: a recursive sign index.
              complement: is at -1 when a flip of sign has to be performed at the next depth.
2      if grade(label_c) == k_c then                                   // end of recursion
3          c[label_C] += sign × A[label_A] × B[label_B]
4      else                                                            // recursive calls
5          msb_A = labelToMsb(label_A)
6          msb_B = labelToMsb(label_B)
7          msb_C = labelToMsb(label_C)
8          foreach msb such that gradeKReachable(k_C, msb, label_C) == true do
9              label = label_C + msb
10             if gradeKReachable(k_A, msb, label_a) then
11                 outer(A, B, C, k_A, k_B, k_C, label_A + msb, label_B, label, sign ×
                   complement, complement)
12             if gradeKReachable(k_B, msb, label_B) then
13                 outer(A, B, C, k_A, k_B, k_C, label_a, label_b + msb, label, sign, −complement)
```

4.4.4 Complexity

This paragraph to study the complexity of the outer product. At each depth of the tree, the number of recursive calls is multiplied by 2. Furthermore, to a considered depth corresponds the same grade and thus the same number of recursive calls. Finally, the number of nodes of the same grade k is given by $\binom{d}{k}$ and in the worst case the depth may vary from 0 to d. Thus, the number of recursive calls is upper bounded by:

$$\sum_{i=0}^{d} \binom{d}{k} 2^i \tag{4.4.51}$$

From the binomial theorem this formula can be rewritten as:

$$\sum_{i=0}^{d} \binom{d}{k} 2^i = 3^d \tag{4.4.52}$$

Thus the number of recursive calls is upper bounded by 3^d which is lower than the number of recursive calls obtained with the two previous methods presented up to now in:

$$\sum_{i=1}^{d} 3^i = \frac{3}{2}(3^d - 1) \tag{4.4.53}$$

4.4.5 GA products using metric

4.4.5.1 Left, right contractions and inner product

We defined the outer products with recursive products over the prefix tree, but we did not yet define products and algorithms that depend on the metric. To achieve this, we assume that a diagonal metric *diagMetric* is stored as a vector whose size is the dimension d. This vector is such that:

$$\begin{aligned} diagMetric(0) &= \mathbf{e}_1 \cdot \mathbf{e}_1 \\ diagMetric(1) &= \mathbf{e}_2 \cdot \mathbf{e}_2 \\ &\vdots \\ diagMetric(d-1) &= \mathbf{e}_d \cdot \mathbf{e}_d \end{aligned} \tag{4.4.54}$$

This means that we add another parameter to the recursive functions called $metricCoefficient$. We assume also that *diagMetric* is given. The definition of the left contraction is shown in Algorithm 15.

Algorithm 15: Recursive left contraction product $C = A \rfloor B$

```
1  Function leftcont
       Input: A, B: two multivectors,
              C: resulting multivector,
              k_A, k_B and k_C: the respective grade of each multivector.
              label_A, label_B, label_C: the recursive position on each tree.
              sign: a recursive sign index.
2             complement: is at -1 when a flip of sign has to be performed at the next depth.
3             metricCoefficient: coefficient for handling the metric in the recursive
               formula.
4      if grade(label_B) == k_B then                                  // end of recursion
5          C[label_C] += metricCoefficient × sign × A[label_a] × B[label_B]
6      else                                                           // recursive calls
7          msb_A = labelToMsb(label_A)
8          msb_B = labelToMsb(label_B)
9          msb_C = labelToMsb(label_C)
10         foreach msb such that gradeKReachable(k_B, msb, label_B)== true do
11             label = label_B + msb
12             if gradeKReachable(k_A, msb, label_A) then
13                 leftcont(A, B, C, k_A, k_B, k_C, label_A + msb, label, label_C, sign ×
                    complement, −complement, metricCoefficientdiagMetric(grade(label_B)))
14             if gradeKReachable(k_c, msb, label_c) then
15                 leftcont(A, B, C, k_A, k_B, k_C, label_A, label, label_C +
                    msb, sign, −complement, metricCoefficient))
```

The right contraction is simply obtained due to its similarities with the left contraction. The resulting pseudo-code is shown 16

Algorithm 16: Recursive right contraction product $C = A \lfloor B$

```
1  Function rightcont
       Input: A, B: two multivectors,
              C: resulting multivector,
              k_A, k_B and k_Cs: the respective grade of each multivector.
              label_A, label_B, label_C: the recursive position on each tree.
              sign: a recursive sign index.
2             complement: is at -1 when a flip of sign has to be performed at the next depth.
3             metricCoefficient: coefficient for handling the metric in the recursive
               formula.
4      if grade(label_B) == k_B then                                  // end of recursion
5          C[label_C] += metricCoefficient × sign × A[label_A] × B[label_B]
6      else                                                           // recursive calls
7          msb_A = labelToMsb(label_A)
8          msb_B = labelToMsb(label_B)
9          msb_C = labelToMsb(label_C)
10         foreach msb such that gradeKReachable(k_A, msb, label_A) == true do
11             label = label_A + msb
12             if gradeKReachable(k_B, msb, label_B) then
13                 rightcont(A, B, C, k_A, k_B, k_C, label, label_C + msb, label_C, sign ×
                    complement, −complement, metricCoefficient diagMetric(grade(label_C)))
14             if gradeKReachable(k_C, msb, label_C) then
15                 rightcont(A, B, C, k_A, k_B, k_C, label, label_B, label_C +
                    msb, sign, −complement, metricCoefficient))
```

Knowing both the left and the right contractions, it is not difficult to deduce the inner product.

4.4.5.2 Geometric product

The last product to define is the major one, namely the geometric product. In a similar way as for the outer product, we define and try to prove the recursive formula of the product.

4.4.5.2.1 Recursive products

Proposition 4.4.8. *Let us consider two multivectors A and B whose recursive constructions are respectively:*

$$\left(\mathfrak{a}_{\mathtt{u}}, \left(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^{d-1}}\right)\right), \tag{4.4.55}$$

and

$$\left(\mathfrak{b}_{\mathtt{u}}, \left(\mathfrak{b}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{b}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{b}_{\mathtt{u}+2^{d-1}}\right)\right). \tag{4.4.56}$$

The recursive construction of the geometric product of these two multivectors can be computed as:

$$\begin{aligned}
&\left(\mathfrak{c}_{\mathtt{u}}, \left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{c}_{\mathtt{u}+2^{d-1}}\right)\right) \\
= &\left(\mathfrak{a}_{\mathtt{u}}, \left(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{a}_{\mathtt{u}+2^{d-1}}\right)\right) * \left(\mathfrak{b}_{\mathtt{u}}, \left(\mathfrak{b}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{b}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \mathfrak{b}_{\mathtt{u}+2^{d-1}}\right)\right) \\
= &\Big(\mathfrak{a}_{\mathtt{u}} * \mathfrak{b}_{\mathtt{u}} + \mathit{diagMetric}(\mathtt{msb})\overline{\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}} * \mathfrak{b}_{\mathtt{u}+\mathtt{msb}} + \mathit{diagMetric}(2\mathtt{msb})\overline{\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}} * \mathfrak{b}_{\mathtt{u}+2\mathtt{msb}} + \cdots \\
&+\mathit{diagMetric}(2^{d-1})\overline{\mathfrak{a}_{\mathtt{u}+2^{d-1}}} * \mathfrak{b}_{\mathtt{u}+2^{d-1}}, (\mathfrak{a}_{\mathtt{u}+\mathtt{msb}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{a}_{\mathtt{u}+2\mathtt{msb}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+2\mathtt{msb}}, \cdots, \\
&\mathfrak{a}_{\mathtt{u}+2^{d-1}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+2^{d-1}})\Big)
\end{aligned} \tag{4.4.57}$$

Note that as in the presentation of the geometric product over binary tree, we use the operator $*$ to denote the geometric product.

Proof. We follow a similar approach as with the recursive outer product. On one hand, mapping the prefix tree $\mathfrak{c}$ of Equation (4.4.57) in the binary tree using ψ of Equation (4.4.9) results in:

$$\begin{aligned}
&\left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \left(\mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}, \left(\cdots(\mathfrak{c}_{\mathtt{uu}+2^{d-1}}, \mathfrak{c}_{\mathtt{u}})\right)\right)\right) \\
= &\Big(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+\mathtt{msb}}, \Big(\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+2\mathtt{msb}}, \big(\cdots \\
&(\mathfrak{a}_{\mathtt{u}+2^{d-1}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+2^{d-1}}, \mathfrak{a}_{\mathtt{u}} * \mathfrak{b}_{\mathtt{u}} + \mathit{diagMetric}(\mathtt{msb})\overline{\mathfrak{a}_{\mathtt{u}+\mathtt{msb}}} * \mathfrak{b}_{\mathtt{u}+\mathtt{msb}} \\
&+\mathit{diagMetric}(2\mathtt{msb})\overline{\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}}} * \mathfrak{b}_{\mathtt{u}+2\mathtt{msb}} + \cdots + \mathit{diagMetric}(2^{d-1})\overline{\mathfrak{a}_{\mathtt{u}+2^{d-1}}} * \mathfrak{b}_{\mathtt{u}+2^{d-1}})\big)\Big)\Big)
\end{aligned} \tag{4.4.58}$$

On the other hand, the recursive formula of the geometric product between two multivectors in the binary tree framework can be the pair:

$$\begin{aligned}
&\left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{c}_{\mathtt{u}}\right) \\
= &\left(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+\mathtt{msb}}, \mathfrak{a}_{\mathtt{u}} * \mathfrak{b}_{\mathtt{u}} + \mathit{diagMetric}(\mathtt{msb})\overline{\mathfrak{a}_{\mathtt{u}} + \mathtt{msb}} * \mathfrak{b}_{\mathtt{u}+\mathtt{msb}}\right)
\end{aligned} \tag{4.4.59}$$

After developing this formula at one recursion depth and having in mind the recursive structure of the binary tree, the above formula becomes:

$$\begin{aligned}
&\left(\mathfrak{c}_{\mathtt{u}+\mathtt{msb}}, \left(\mathfrak{c}_{\mathtt{u}+2\mathtt{msb}}, \mathfrak{c}_{\mathtt{u}}\right)\right) \\
= &\Big(\mathfrak{a}_{\mathtt{u}+\mathtt{msb}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+\mathtt{msb}}, \Big(\mathfrak{a}_{\mathtt{u}+2\mathtt{msb}} * \mathfrak{b}_{\mathtt{u}} + \overline{\mathfrak{a}_{\mathtt{u}}} * \mathfrak{b}_{\mathtt{u}+2\mathtt{msb}}, \mathfrak{a}_{\mathtt{u}} * \mathfrak{b}_{\mathtt{u}} \\
&+\mathit{diagMetric}(\mathtt{msb})\overline{\mathfrak{a}_{\mathtt{u}} + \mathtt{msb}} * \mathfrak{b}_{\mathtt{u}+\mathtt{msb}} + \mathit{diagMetric}(2\mathtt{msb})\overline{\mathfrak{a}_{\mathtt{u}} + 2\mathtt{msb}} * \mathfrak{b}_{\mathtt{u}+2\mathtt{msb}}\Big)\Big)
\end{aligned} \tag{4.4.60}$$

After further developing this formula, we find:

$$
\begin{aligned}
& \left(\mathfrak{c}_{u+msb}, \left(\mathfrak{c}_{u+2msb}, \left(\cdots\left(\mathfrak{c}_{uu+2^{d-1}}, \mathfrak{c}_u\right)\right)\right)\right) \\
= & \left(\mathfrak{a}_{u+msb} * \mathfrak{b}_u + \overline{\mathfrak{a}_u} * \mathfrak{b}_{u+msb}, \left(\mathfrak{a}_{u+2msb} * \mathfrak{b}_u + \overline{\mathfrak{a}_u} * \mathfrak{b}_{u+2msb}, \left(\cdots\right.\right.\right. \\
& \left.\left.\left.(\mathfrak{a}_{u+2^{d-1}} * \mathfrak{b}_u + \overline{\mathfrak{a}_u} * \mathfrak{b}_{u+2^{d-1}}, \mathfrak{a}_u * \mathfrak{b}_u + diagMetric(msb)\overline{\mathfrak{a}_{u+msb}} * \mathfrak{b}_{u+msb}\right.\right.\right. \\
& \left.\left.\left. + diagMetric(2msb)\overline{\mathfrak{a}_{u+2msb}} * \mathfrak{b}_{u+2msb} + \cdots + diagMetric(2^{d-1})\overline{\mathfrak{a}_{u+2^{d-1}}} * \mathfrak{b}_{u+2^d}\right)\right)\right)
\end{aligned}
\tag{4.4.61}
$$

Equations (4.4.61) and (4.4.58) are equivalent, thus the commutative diagram holds for the geometric product. □

4.4.5.2.2 Algorithm: This recursive formula is the base to develop the pseudo-code of the geometric product. As a reminder, the recursive formula is defined as:

$$
\begin{aligned}
& \left(\mathfrak{c}_u, (\mathfrak{c}_{u+msb}, \mathfrak{c}_{u+2msb}, \cdots, \mathfrak{c}_{u+2^{d-1}})\right) \\
= & \left(\mathfrak{a}_u * \mathfrak{b}_u + diagMetric(msb)\overline{\mathfrak{a}_{u+msb}} * \mathfrak{b}_{u+msb} + diagMetric(2msb)\overline{\mathfrak{a}_{u+2msb}} * \mathfrak{b}_{u+2msb} + \cdots\right. \\
& + diagMetric(2^{d-1})\overline{\mathfrak{a}_{u+2^{d-1}}} * \mathfrak{b}_{u+2^{d-1}}, (\mathfrak{a}_{u+msb} * \mathfrak{b}_u + \overline{\mathfrak{a}_u} * \mathfrak{b}_{u+msb}, \mathfrak{a}_{u+2msb} * \mathfrak{b}_u + \overline{\mathfrak{a}_u} * \mathfrak{b}_{u+2msb}, \cdots, \\
& \left.\mathfrak{a}_{u+2^{d-1}} * \mathfrak{b}_u + \overline{\mathfrak{a}_u} * \mathfrak{b}_{u+2^{d-1}}\right)\Big)
\end{aligned}
\tag{4.4.62}
$$

We highlight the main parts of the recursive formula and their equivalent in the pseudo-code shown in Algorithm 17.

4.4.6 Complexity

Concerning the left and right contractions, the number of recursive calls is the same as the recursive outer product. Hence, these two recursive products require 3^d recursive calls in the worst case. Again, this is a lower complexity than state of the art methods. As for the geometric product, we note that the computation of the sign is performed by a constant time operation. By an operation similar to the computation of the complexity of the outer product, we prove that the number of recursive calls of the geometric product is 4^d. Again, the number of recursive calls is lower than for the two other recursive methods.

Algorithm 17: Recursive geometric product $C = AB$

```
1  Function geometric
       Input: a, b: two multivectors,
              c: resulting multivector,
              k_a, k_b and k_c: the respective grade of each multivector.
              label_a, label_b, label_c: the recursive position on each tree.
              sign: a recursive sign index.
              complement: is at -1 when a flip of sign has to be performed at the next depth.
              metricCoefficient: coefficient for handling the metric in the recursive
              formula.
              depth: current depth in the prefix tree structure.
2      if grade(label_B) == k_B and grade(label_A) == k_A then        // end of recursion
3          C[label_C] += metricCoefficient × sign × A[label_A] × B[label_B]
4      else
5          msb_A = labelToMsb(label_A)
6          msb_B = labelToMsb(label_B)
7          msb_C = labelToMsb(label_C)
8          for i in 2^depth, 2^(depth+1), ··· , 2^(d−1) do
9              if gradeKReachable(k_B, i, label_B) then
10                 if gradeKReachable(k_A, i, label_A) then
11                     geometric(A, B, C, k_A, k_B, k_C, label_A + i, label_B + i, label_C, sign ×
                         complement, −complement, metricCoefficient ×
                         diagMetric(i), depth + 1))
12             if gradeKReachable(k_A, i, label_A) then
13                 geometric(A, B, C, k_A, k_B, k_C, label, label_B, label_C + msb, sign ×
                     complement, complement, metricCoefficient, depth + 1))
14             if gradeKReachable(k_B, i, label_B) then
15                 geometric(A, B, C, k_A, k_B, k_C, label_A, label_B + i, label_C +
                     i, sign, −complement, metricCoefficient), depth + 1)
```

4.4.7 Dual and prefix tree

Given a multivector A, the nodes at depth k of the prefix tree represent the components of A of grade k. Hence, the root of the tree is always the scalar component, and the deepest leave corresponds to the pseudoscalar component of A. It is noteworthy to observe that this formulation also implicitly describes the dual A^* of A by reading the tree from the pseudoscalar leave to the scalar root, as shown in figure 4.12. This dual "upside down" representation of the prefix tree

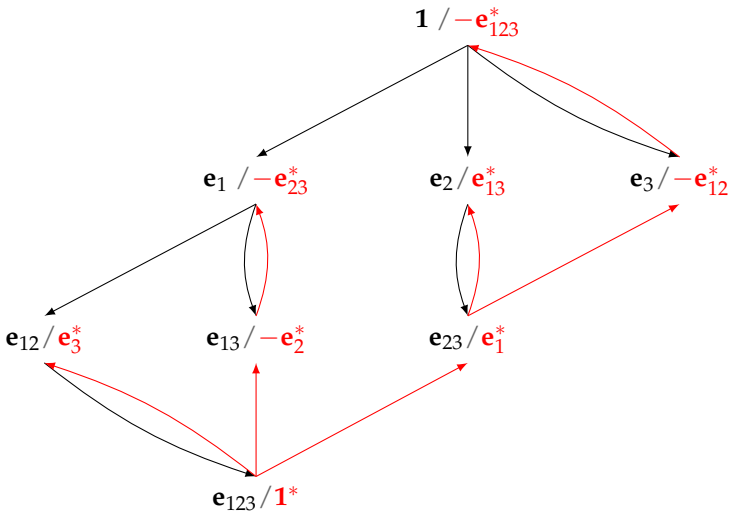


FIGURE 4.12: Primal form of a tree data structure of an Euclidean 3 dimensional vector space, and its dual counterpart in red

involves some basis sign changes adjustments. Moreover, some nodes of the dual prefix tree are affected by the metric of the specified algebra. To include these coefficient changes, both the sign and the metric coefficients can be stored in a single array of size 2^d, where d is the dimension of the vector space supporting the algebra.

In practice, the dual tree traversal requires a function to indicate the children of a given node. This function corresponds to a dual version of Eq. (4.4.2) and is derived from the binary labelling of the nodes, by:

$$\text{dual\_child\_label}(\texttt{label}, \texttt{msb}) = \texttt{label} - \texttt{msb} \tag{4.4.63}$$

Note that the label of the root is now the binary label $(\texttt{1} << \texttt{d}) - \texttt{1}$ where $<<$ is the left shift operator that shifts on the left the digits of a label.
Then, the pseudo-code of the traversal of such dual tree is shown in Algorithm 18.

The dual and primal prefix tree representations are the support of an efficient recursive expression of GA products, coupled with the per grade data structure of section 2.2.1. As for the primal prefix tree, the dual prefix tree is just a support for the recursive products, the data are staying

stored into the "per grade" data structure of section 2.2.1. The main goal of this dual prefix tree is to compute some products between dual multivectors without computing the costly multivector dualization.

Algorithm 18: Recursive traversing of the dual of a multivector A whose grade is k_A

```
1 Function traverse
    Input: A: the multivector to be traversed,
           f: function,
           k_A: the grade of the multivector.
           label_A: the recursive position on each tree.
2   if grade(label_C) == (d − k_C) then                    // end of recursion
3     f[label_A]
4   else                                                   // recursive calls
5     msb_A = labelToMsb(label_A)
6     foreach msb such that gradeKReachable(k_A,msb)== true do
7       label = label_A − msb
8       if gradeKReachable(k_A,msb) then
9         traverse(A, k_A, label)
```

4.4.8 Products with dual multivectors

A recursive product where one or both of the operands are dual multivectors can be optimized by an extension of Algorithm 14 adapted to the dual tree defined in Section 4.4.7. In a certain sense, it is like if the recursive product algorithm is dualized instead of the multivectors. In this situation, potential costly dualizations can be avoided.

In more details, let us consider the following product:

$$C = A \wedge B^* \tag{4.4.64}$$

First we assume that a coefficient is associated to each element of the per grade data structure of 2.2.1. This coefficient contains the basis coefficients changes adjustments. As an example, the Figure 4.13 shows the coefficients that are stored using C3GA: We call this structure *dualCoefficients*. Then the outer product between A and B^* is based on the following:

$$C = A \wedge B^* = (A \cdot \mathit{dualCoefficients} \cdot B)^* \tag{4.4.65}$$

dualCoefficients is here due to the fact that

$$\begin{aligned}(B^*)^* &= (\pm B \cdot \mathbf{I})^* \\ &= (B \cdot \mathbf{I}^{-1}) \cdot \mathbf{I}^{-1} \\ &= B \cdot \mathbf{I}^{-2}\end{aligned} \tag{4.4.66}$$

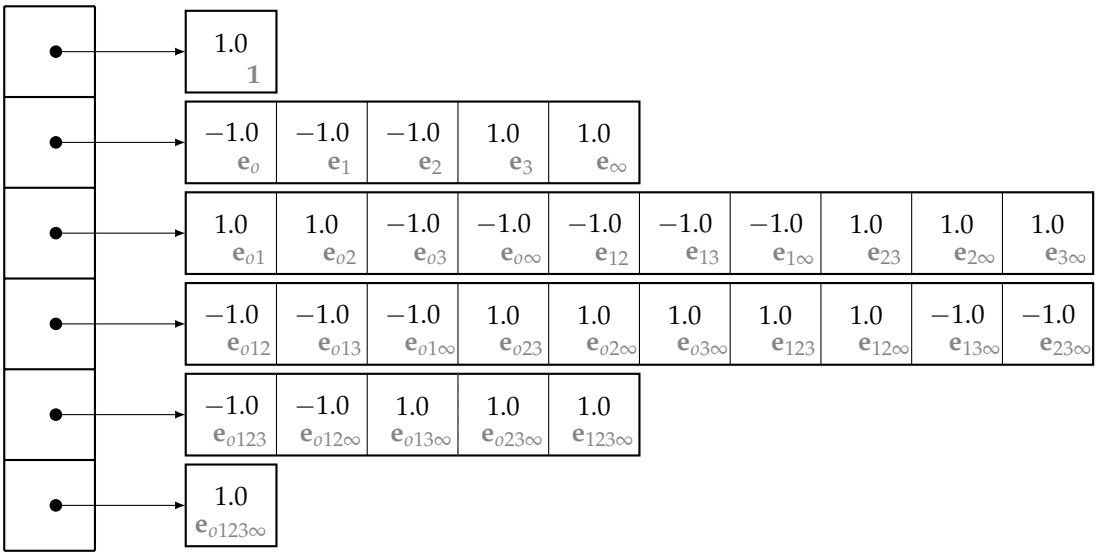


FIGURE 4.13: The Data structure shown in Section 2.2.1, this structure includes the coefficients to traverse any dual multivector.

thus *dualCoefficients* embeds this coefficient.

We already have a recursive method to compute the inner product as well as a method to traverse the dual prefix tree of the resulting multivector C using the structure *dualCoefficients*. The resulting pseudo-code is shown in Algorithm 19 when the grade of A is lower than the grade of B. When the grade of B is higher than the grade of A then the algorithm is very similar (extracted from the right contraction algorithm).

4.4.9 Discussion

We presented a new approach based on recursive products over prefix tree for high dimensional space. This prefix tree has a per-grade structure. Thus, the recursive products well encode the per-grade structure of the data structure presented in the previous section. This latter approach has the lowest number of recursive calls compared with binary tree recursive products. The products neither require memory allocation of costly lists nor the computation of a costly function to interface with the data structure. Thus, this is a good support to meet requirements 2a,3a, 3b.

Up to now, we described the approach followed for high dimensional spaces and the approach for low dimensional space. We explain now the transition from the low to the high dimensional called Hybridization.

Algorithm 19: Recursive outer product between a primal multivector and the dual of another multivector: $C = A \wedge B^*$

```
1  Function outerPrimalDual
       Input: A, B: two multivectors,
              C: resulting multivector,
              k_A, k_B and k_C: the respective grade of each multivector.
              label_A, label_B, label_C: the recursive position on each tree.
              sign: a recursive sign index.
2             complement: is at -1 when a flip of sign has to be performed at the next depth.
3             metricCoefficient: coefficient for handling the metric in the recursive
               formula.
4      if grade(label_B) == k_B then                                  // end of recursion
5          dualCoefficients(label_C) × C[label_C] += 
            dualCoefficients × sign × A[label_C] × B[label_B]
6      else                                                           // recursive calls
7          msb_A = labelToMsb(label_A)
8          msb_B = labelToMsb(label_B)
9          msb_C = labelToMsb(label_C)
10         foreach msb such that gradeKReachable(k_B, msb, label_B)== true do
11             label = label_B + msb
12             if gradeKReachable(k_A, msb, label_A) then
13                 outerPrimalDual(A, B, C, k_A, k_B, k_C, label_A + msb, label, label_C, sign ×
                    complement, −complement, metricCoefficient diagMetric(grade(label_B)))
14             if gradeKReachable(k_C, msb, label_C) then
15                 outerPrimalDual(A, B, C, k_A, k_B, k_C, label_A, label, label_C −
                    msb, sign, −complement, metricCoefficient)
```

4.4.10 Hybridization

For high dimension GA, the generated libraries include a soft transition between precomputed products and recursive products. The criteria for a product to be implemented either with precomputed functions or recursively is defined by a user defined threshold on the size of the two k-vectors involved in the product. With this approach, a GA library over a 10 dimension vector space can entirely be implemented in precomputed functions and a GA library over a 15 dimension vector space will have at least the products of vectors implemented with precomputed functions.

4.5 Non orthogonal metric

For ergonomic purposes, any optional basis changes required by an arbitrary metric are automatically handled by the generated library. This basis change is included in the precomputed functions during the pre-computation process and is explicitly computed for the recursive products before and after the recursive calls. The library generator first checks if the metric is a valid symmetric matrix. If the matrix is identity, all the generated products are left unchanged. If the

matrix is a diagonal matrix (but not identity), the metric coefficients are inserted in the products. In any other cases, we follow a similar approach as [31] and proceed to a basis change.

4.5.1 Numerical robustness preprocessing

However we also add some numerical robustness preprocessing. As an example, let us consider the Conformal Geometric Algebra of $\mathbb{R}^2$ with metric M and its eigen decomposition $M = PDP^{-1}$:

$$M = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.707 & 0.707 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ -0.707 & 0.707 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.707 & 0 & 0 & -0.707 \\ 0.707 & 0 & 0 & 0.707 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.5.1}$$

For such very common metrics, an eigen decomposition leads to square roots in the eigen vector components. For a better numerical robustness, we automatically up-scale the matrix P such that it is composed of integers and downscale accordingly its inverse P^{-1}:

$$M = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 & -0.5 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.5.2}$$

Then, all the components of the resulting matrices are subject to a numerical clean up, by adjusting each value to the nearest integer, inverse power of two or decimal. Thus, this clean up removes the numerical errors generated by the eigen decomposition and is validated if the resulting decomposition still results in the original metric. In all the GA we encountered, this process removes all the numerical approximations. This latter result is a good support for fulfilling the need 4a.

4.5.2 Computing transformation matrices

The final stage consists in the generation of both transformations and inverse transformation matrices for any grade of the algebra. In practice, these transformation matrices are very sparse and are stored in the efficient `eigen` sparse matrices [41]. The algorithm followed to achieve this is explained in the following section.

4.5.2.1 Algorithm

We explain the computation of the transformation matrix P_k that maps any k—basis vector in the non-orthogonal basis to the k-basis vector in the orthogonal basis. First, let us consider the orthogonal basis $(\mathbf{e}_1, \cdots, \mathbf{e}_d)^\top$ and the non-orthogonal basis as $(\mathbf{n}_1, \cdots, \mathbf{n}_d)^\top$. We assume that $P(i)$ denotes the i-th line of P and p_{ij} represents the element of the i-th line and j-th column of P. The transformation matrix P maps these basis vectors as follows:

$$\begin{pmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \\ \mathbf{e}_d \end{pmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1d} \\ p_{21} & p_{22} & \cdots & p_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ p_{d1} & p_{d2} & \cdots & p_{dd} \end{bmatrix} \begin{pmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \vdots \\ \mathbf{n}_d \end{pmatrix} = \begin{bmatrix} P(1) \\ P(2) \\ \vdots \\ P(d) \end{bmatrix} \begin{pmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \vdots \\ \mathbf{n}_d \end{pmatrix}, \tag{4.5.3}$$

Thus, by definition:

$$\mathbf{e}_i = P(i) \begin{pmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \vdots \\ \mathbf{n}_d \end{pmatrix}, \forall i \in [1, d], \tag{4.5.4}$$

For a k—basis blade defined as:

$$\mathbf{e}_{pqr\cdots u} = \mathbf{e}_p \wedge \mathbf{e}_q \wedge \mathbf{e}_r \wedge \cdots \wedge \mathbf{e}_u, \tag{4.5.5}$$

Using formula 4.5.4, this yields:

$$\mathbf{e}_{pqr\cdots u} = P(p) \begin{pmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \vdots \\ \mathbf{n}_d \end{pmatrix} \wedge P(q) \begin{pmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \vdots \\ \mathbf{n}_d \end{pmatrix} \wedge P(r) \begin{pmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \vdots \\ \mathbf{n}_d \end{pmatrix} \wedge \cdots \wedge P(u) \begin{pmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \vdots \\ \mathbf{n}_d \end{pmatrix}, \tag{4.5.6}$$

Hence, the main point of determining the transformation matrices lies in computing the outer product between lines of the vector transformation matrix. The resulting pseudo-code is shown in Algorithm 20.

Algorithm 20: Compute the k-vector transformation matrix from the vector transformation matrix P

Function `computeKvectorTransformationMatrix`
  Input: P: the vector transformation matrix,
    d: vector space dimension,
    k: the grade of the transformation matrix to be computed.
  $P_k \leftarrow$transformation matrix
  foreach $\mathbf{e}_{pqr\cdots u}, idx$ **in** *k-basis blades* **do** // idx:index of the blade in the k basis blades
    $P_{<k>}(idx) \leftarrow$ vector whose dimension is $\binom{d}{k}$
    $mv \leftarrow P(p)(\mathbf{n}_1, \mathbf{n}_2, \cdots, \mathbf{n}_d)^\top$
    foreach $\mathbf{e}_v$ **in** $\mathbf{e}_{qr\cdots u}$ **do**
      $mv \leftarrow mv \wedge \left(P(v)(\mathbf{n}_1, \mathbf{n}_2, \cdots, \mathbf{n}_d)^\top\right)$
    $P_k(idx) \leftarrow mv$
  return $P_{<k>}$

4.5.2.2 Data structure to use

As stated in the last section, the obtained transformation matrices are sparse. An example of the transformation matrix $P_{<3>}$ of 3-basis blades is shown below:

$$P_{<3>} = \begin{bmatrix} 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.5.7}$$

We remark that for over a total of $10 \times 10 = 100$ elements, only 16 elements are non-null, giving to this matrix a sparsity score of $\frac{84}{100} = 0.84$. This score tends to be stable for extension of conformal Geometric Algebra in higher dimension. That is why, we use sparse matrix data structure of *Eigen*.

Note also that the inverse transformation matrices can be computed using the same process as with the transformation matrices. The only difference will be in the vector transformation matrix used as input. In this case, we will use the inverse of the vector transformation matrix. All the remaining algorithm remains the same.

Finally, in the case of non-full-rank metrics, the process remains unchanged, however the dual functions are not generated.

Chapter 5

Garamon

5.1 Resulting library

The resulting implementation is a C++ template library generator dedicated to Geometric Algebra. The generator itself runs in C++ and generate optimized C++ code. These generated GA libraries are dedicated to be user-friendly and efficient both in term of computation speed and memory consumption.

The generated libraries are built from a short configuration file describing the targeted algebra. This configuration file specifies the algebra signature, the name of the basis vectors and some optimization options. This file is restricted to the minimum information such it can be filled very easily. The Listing 5.1 is an example of such configuration file for CGA.

LISTING 5.1: Configuration file of C3GA using Garamon

```
<namespace>
c3ga
</namespace>
# dimension of the algebra vector space (grade 1)
<dimension>
5
</dimension>

# inner product of basis vectors (grade 1)
# It should be a symmetric matrix.
# delimiter are spaces
<metric>
 0 0 0 0 -1
 0 1 0 0 0
 0 0 1 0 0
 0 0 0 1 0
-1 0 0 0 0
</metric>

# name of each basis vector (grade 1). They will be prefixed by the symbol
# 'e' in the library.
```

```
# supported characters : letters and numbers that are compatible with C++
# variable naming.
# for hig dimensions, add a '_' at the end of the name to avoid ambiguities
# (e114 : 11,4 or 1,14 ? => e11_4_)
# example: "0 1 2 3 i" will generate e0 e1 e2 e3 ei e01 e02 ... e0123i
<basis vector name>
0 1 2 3 i
</basis vector name>
```

Any generated library contains its own dedicated installation file (cmake), as well as a dedicated sample code. The generated libraries handle any arbitrary Geometric Algebra signature, such that the user do not have to care about basis change. The Listing A.1 shows a example of some operations in CGA using our library:

LISTING 5.2: Code example in CGA with Garamon

```
#include <iostream>
#include <cstdlib>
#include <c3ga/Mvec.hpp>

int main(){

    c3ga::Mvec<double> mv1;
    mv1[c3ga::scalar] = 1.0;
    mv1[c3ga::E0] = 42.0;
    std::cout << "mv1 : " << mv1 << std::endl;

    c3ga::Mvec<double> mv2;
    mv2[c3ga::E0] = 1.0;
    mv2[c3ga::E1] = 2.0;
    mv2 += c3ga::I<double>() + 2*c3ga::e01<double>();
    std::cout << "mv2 : " << mv2 << std::endl << std::endl;

    // some products
    std::cout << "outer product : " << (mv1 ^ mv2) << std::endl;
    std::cout << "inner product : " << (mv1 | mv2) << std::endl;
    std::cout << "geometric product : " << (mv1 * mv2) << std::endl;
    std::cout << "left contractiont : " << (mv1 < mv2) << std::endl;
    std::cout << "right contractiont: " << (mv1 > mv2) << std::endl;
    std::cout << std::endl;

    // some tools
    std::cout << "grade : " << mv1.grade() << std::endl;
    std::cout << "norm : " << mv1.norm() << std::endl;
    mv1.clear();
    if(mv1.isEmpty()) std::cout << "mv1 is empty: ok" << std::endl;

    return EXIT_SUCCESS;
}
```

Finally, since all the generated libraries are identified by a namespace, multiple GA libraries can be used together as shown in Listing 5.3.

LISTING 5.3: Code example with Garamon using both space time algebra and algebra of a 3-dimensional vector space

```
#include <iostream>
#include <cstdlib>
#include <st3ga/Mvec.hpp>
#include <e3ga/Mvec.hpp>

int main(){

    st3ga::Mvec<double> mv1 = ...;
    e3ga::Mvec<double> mv2 = ...;

    return EXIT_SUCCESS;
}
```

For example, one can think of checking that space-time bivectors of the space-time algebra are isomorphic to vectors of a Euclidean space Geometric Algebra.
This latter thing yields a very flexible implementation. Indeed, flexibility requirements 1b, 1a are fulfilled.

The organization of the generated library is simple. The generated code contains only header files, including a Multivector class Mvec, as well as header files to compute the products of Geometric Algebra. Other header files are used for the storage of useful constants, namely Constants.hpp. These classes' configuration are shown Figure 5.1

Additional functionalities, including an implementation of the considered library can be seen in the folder `sample/`. In this folder, one may find demonstration of some of the main functionalities of the considered library as well as some tests. The folder `doc/` contains all the files to generate documentation of the library. Finally, note that the libraries are open source and the MIT license. All this functionalities enable to fulfil the reading requirements 7a and 7b.

5.2 Performances

We conducted some tests on high quality consumer grade hardware over several platforms (Ubuntu-16.04, MacOs-10.12 and Windows-10), with gcc-5.4, clang-9.0 and MinGW-7.2 compilers. The compilers just need to be compatible with C++14. These tests mainly concern the speed of the products, the size of the binary file, the size of the stored data and the dimension range. To get a better understanding of the results, we compared Garamon with some of the most efficient existing GA libraries in C++, Gaigen [34] and Versor [14].

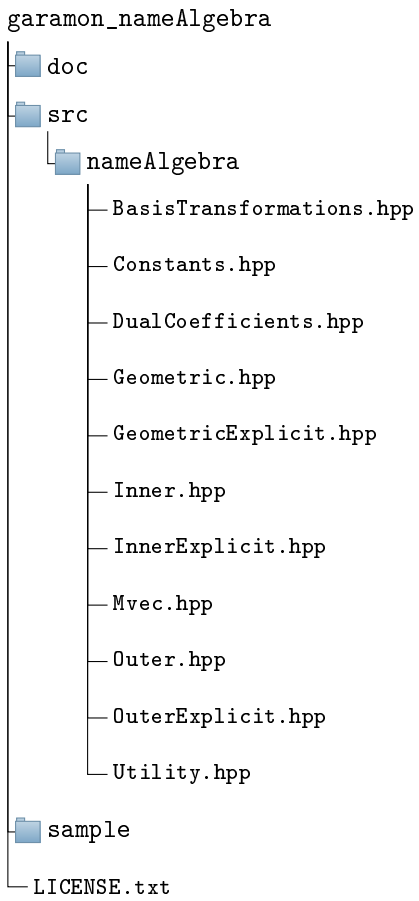


FIGURE 5.1: Garamon library directory tree

5.2.1 High dimensions

In this section, the term dimension d refers to the dimension of the vector space used to build a GA composed of 2^d elements. As stated in [34], the maximum dimension supported by Gaigen is dimension 12. The tests we conducted on Versor showed that a single vector product could run in an Euclidean GA at most in dimension 10, due to compilation memory overloads. This maximum dimension falls to dimension 7 when the program tested involves various grades of k-vectors and various associated products.

Garamon is designed to be compatible with high dimension algebras. Due to some technical choices, Garamon has a hard-coded limit of dimension 31. However, in practice, while generating a library based on an Euclidean algebra of dimension 20 takes few seconds, generating a library based on a conformal vector space (including basis changes) of the same dimension 20 may requires hours.

For practical applications, we conducted some tests on both Double Projective Geometric Algebra of $\mathbb{R}^{4,4}$ [26] and Triple Conformal Geometric Algebra of $\mathbb{R}^{9,3}$ [28]. For higher dimensional algebra, we tested Garamon on the Quadric Conformal Geometric Algebra [7] built over a 15—dimensional vector space for real-time applications.

5.2.2 Speed computation

The speed computation tests were conducted on basic operations like outer products $C = A \wedge B$, inner products $C = A \cdot B$, or some combinations $D = (A \wedge B) \cdot C$. For more complex operations, we expect Gaalop [12] to provide some efficient code reduction such it becomes the best solution every time.

For Gaalop, we followed its standard usage and generated a set of functions with general signature like "`void myProduct(A,B,C)`", that are clearly efficient since no memory allocation nor memory copy are required. However, these functions are far from easy to use when combining several products. For the other tested libraries, we used the already defined functions, such as $\mathtt{C} = \mathtt{A} \wedge \mathtt{B}$. In most of the implementations, these operations require a memory allocation to locally store the result, and a copy to the final variable.

For each tested libraries, the speed performance can vary according to the platform, the compiler and the algebra dimension. However, the trend of these benchmarks tends to show that Gaalop and Versor are almost every time the fastest. Garamon presents the same performances as Gaigen, and surprisingly performs sometimes better that Gaalop on products such as $D = (A \wedge B) \cdot C$.

The code profiling shows that a large part of the product in Garamon is actually used for the memory allocation. This situation is especially true when the result of a product has several grades, like is some geometric products where the memory allocation is performed for all independent grades and not once, like in Gaigen. The memory handling of Garamon, however, presents some good property when manipulating a large amount of data, as described in the section 5.2.3. Finally, in a future version of Garamon, we may consider including meta-programming, like in Versor, to speed up some computations in low dimensions.

5.2.3 Memory consumption

The data memory consumption tests were conducted by generating both a large number of random vectors and bivectors. Let d be the dimension of the vector space supporting the algebra, Table 5.1 and 5.2 show that the per-grade arrays has a memory storage roughly linear in d when the full multivector has a memory complexity of $O(2^d)$.

TABLE 5.1: Memory requirement (in MB) to store 5.10^4 random vectors.

dimension	5	6	7	8	10	15
Gaigen	12.8	25.6	51.2	102.4	409.6	–
Versor	4.6	5.0	5.5	6.3	–	–
Garamon	2.1	2.5	2.9	3.4	4.3	6.4

TABLE 5.2: Memory requirement (in MB) to store 5.10^4 random bivectors.

dimension	5	6	7	8	10	15
Gaigen	12.8	25.6	51.20	102.4	409.6	–
Versor	7.9	11.8	16.6	22.1	–	–
Garamon	5.3	7.9	11.2	15.3	24.7	57.6

In term of binary file size and as an indication, Garamon weights 2.6 MB in dimension 5 and 7.8MB in dimension 18. This binary includes a pre-compiled version for `float` and `double`.

5.3 Partial conclusion

We presented a Geometric Algebra library generator synthesizing C++ libraries implementing Geometric Algebras of low and high dimensions for any arbitrary metrics. Our purpose is to be as user-friendly as possible, without too much computation speed repercussions, and to have a good behaviour in term of memory consumption. The libraries are generated from a simple specification file. The "per grade" data structure used in Garamon is an efficient compromise between data storage, computation efficiency and user-friendliness. According to the base vector space dimension, the generated specialized libraries are implemented either with full precomputed operations or also based on a new recursive scheme following a prefix tree multivector representation for higher dimensions. An "upside down" reading of the prefix tree leads to recursive products of the dual mutivector without any explicit dualization. Finally, Garamon can handle any arbitrary algebra signatures with a numerically robust basis change implementation.

We consider the resulting library as an efficient tool to easily test and investigate GA algorithms. Furthermore, due to both the flexibility and its computation efficiency, we could investigate and test a new framework to represent and manipulate quadric surfaces in a 15-dimensional vector space, namely Quadric Conformal Geometric Algebra. This new model is presented in the next chapter. All these features of Garamon ensure that all the needs expressed at the beginning of this chapter are fulfilled.

Part II

Geometric algebra and quadrics

The development of the library *Garamon* opened some lock, such that one could easily explore some computations in vector spaces whose dimension is higher than 12. We decided to investigate new models to represent and manipulate quadric surfaces. Some Geometric Algebra like the well-defined Conformal Geometric Algebra (CGA) constructs lines, circles, planes, and spheres from control points just by using the outer product. As an example, a sphere in CGA is expressed as the Geometric Algebra element $\mathbf{S}$ and 4 control points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ as:

$$\mathbf{S} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \tag{5.3.1}$$

There exist some Geometric Algebras to handle quadric surfaces, however, none of them build general quadric surfaces from control points, in a similar way as Equation 5.3.1. In this part, we present a novel Geometric Algebra framework, the Geometric Algebra of $\mathbb{R}^{9,6}$, to deal with quadric surfaces where an arbitrary quadric surface is constructed by the outer product of points. The proposed framework enables us not only to intuitively represent quadric surfaces but also to construct objects using Conformal Geometric Algebra. Our proposed framework also provides the computation of the intersection of quadric surfaces, the normal vector, and the tangent plane of a quadric surface. Models of Versor are discussed. Although the use of such high dimensional space might appear computationally inefficient compared to other Geometric Algebra framework, we prove that some operations are computationally more efficient using the algebra of $\mathbb{R}^{9,6}$ than others in lower dimension. We go further in that direction by proposing a mapping between the main Geometric Algebra models that makes computationally efficient the main computer graphics operations required for quadrics.

Chapter 6

Introduction

Following the Conformal Geometric Algebra (CGA) of $\mathbb{G}_{4,1}$*, and its well-defined properties [23], the Geometric Algebra community recently started to explore new frameworks that work in higher dimensional spaces. The motivation for this direction is to investigate the representation and manipulation of wider range of objects in an intuitive way. In particular, $\mathbb{G}_{3,3}$ can be the support to define either a 3D projective geometry, as introduced by Dorst [21] or line geometry defined by Klawitter [56]). Conics in $\mathbb{R}^2$ can be represented by the conic space defined by Perwass [73] with $\mathbb{G}_{5,3}$. Conics in $\mathbb{R}^2$ may also be defined by the Double Conformal Geometric Algebra (DCGA) with $\mathbb{G}_{6,2}$ introduced by Easter and Hitzer [27].

The possibility to construct surfaces as outer products of points pays particular attention. In Geometric Algebra, this construction requires even higher dimensional vector space than for conics. Briefly, a general quadric has 10 coefficients and 9 degrees of freedom. Therefore, 9 points are enough to construct a general quadric surface. If we assume that a point is represented as a 1-vector, then the construction of a general quadrics $\mathbf{Q}$ using control points similarly as CGA is constrained by the facts that one has to:

1. compute outer product between these 9 points,
2. check if a point lies in the resulting entity, meaning $\mathbf{x} \wedge \mathbf{Q} = 0$.

Constraint 1 implies that 9-dimensional subspace has to be part of the algebra. Constraint 2 adds one more subspace. It means that the dimension of the vector space is at least a 10-dimensional space. Furthermore, one might want that the inner product between two points represents the distance between the two points. This adds again extra null-basis vectors. This leads to a high dimensional vector space.

Two major problems arise with these high dimensional vector spaces. These problems are inherent to the fact that the dimension of the algebra generated by any d-dimensional vector spaces

*As a reminder the Geometric Algebra of $\mathbb{R}^{p,q}$ is denoted by $\mathbb{G}_{p,q}$ where p is the number of basis vectors squared to $+1$ and q is that of basis vectors square to -1

grows exponentially as they have 2^d basis elements. The first problem is the lack of implementation that can be used to handle such algebras. This latter point was discussed in the previous part and results in an efficient implementation of high dimensional vector space. The second problem lies in the fact that the use of an algebra with 2^d, $d > 10$ elements results in **at least** 1024 basis elements. Represent a general quadric with only 10 coefficients in a 1024-dimensional basis vectors does not seem very efficient.

6.1 Contributions

This is our motivation for the development of a new model referred as Quadric Conformal Geometric Algebra (QCGA). This model is an extension of CGA, specifically dedicated to quadric surfaces. The idea is to represent the objects in low dimensional subspaces of the algebra. QCGA is capable of constructing quadric surfaces either using control points or implicit equations. Moreover, QCGA can compute the intersection of quadric surfaces, as well as, the surface tangent and normal vectors at a point that lies in the quadric surface. We prove that some operations of this model are computationally more efficient using QCGA than other frameworks of lower dimensions. We go further in that direction by proposing a mapping between the main Geometric Algebra models that makes computationally efficient operations required for the manipulation of quadrics.

This second part of the thesis is organized as follows. The first chapter presents the main state-of-the-art methods to deals with quadric surfaces in Geometric Algebra. In addition, a study of complexity is carried to compare the different frameworks. Then, we present QCGA, a study of complexity is also performed. Finally, a new efficient framework is presented that is well suited for the representation and manipulation of quadric surfaces.

Chapter 7

Models of quadric surfaces with Geometric Algebra

This chapter presents the main Geometric Algebra models to represent and manipulate quadric surfaces. Furthermore, we aim at determining the most efficient Geometric Algebra model for each of the main operations required in computer graphics. The purpose is to propose a Geometric Algebra model that allows to efficiently handle quadrics.
The considered operations over the surfaces are:

- checking if a point lies in a quadric surface,
- intersecting quadric and line,
- computing the normal vector (and tangent plane) of a surface at a given point.

One of the applications of such operations is to compute precise visualizations using ray-tracer [37].

A first framework to handle quadric surfaces was introduced thanks to the pioneering work of Zamora [85]. This framework constructs a quadric surface from control points in $\mathbb{G}_{6,3}$. In this model, an axis-aligned quadric $\mathbf{q}$ can be defined as:

$$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{x}_6 \tag{7.0.1}$$

The major drawback of this model is that it supports only axis-aligned quadric surfaces. Due to this fact, we will not further detail this model.

There exist three main Geometric Algebra frameworks to manipulate general quadric surfaces. First, DCGA (Double Conformal Geometric Algebra) with $\mathbb{G}_{8,2}$ defined by Easter and Hitzer [27]. Second, a framework of $\mathbb{G}_{4,4}$ as firstly introduced by Parkin [70] and developed further by Du et al. [26]. The last one is our contribution, introduced in [7] and is a model of $\mathbb{G}_{9,6}$.

7.1 Complexity estimation model

In order to compare the operations for these different frameworks, we need a computational model. This requires to be able to determine the complexity of operations in each framework. Using the complexity of the operations explained in the last part of this manuscript is well suited when one wants to compare different methods to compute the products. Instead, our complexity estimation is made through two simplifying assumptions.

First, let us consider the outer product between one homogeneous multivector whose number of components is u and another homogeneous multivector whose number of components is v, $u, v \in \mathbb{N}$. We assume that an upper bound to the number of required products is at most uv products, as shown in Equation 2.7.1.

The general Equation 1.4.6 is our base for the second assumption. Furthermore, we need to use this formula for inner products between 1-vector and 2-vector as well as inner product between two 1-vector. The first multivector has u non-zero components, and the second has v non-zero components. Then using Equation 1.4.6, the inner product between two 1—vectors will result in uv products. Whereas, the inner product between 1—vectors and 2-vectors requires two inner products for each pair of components of the two multivectors. Thus, this inner product requires at most $2uv$ products. These models will be used for determining the complexity of each operation. Let us now explain in more details these models, we start with DCGA.

7.2 DCGA of $\mathbb{G}_{8,2}$

DCGA was presented by Hitzer and Easter [27] and aims at having entities representing both quartic surfaces and quadric surfaces.

Basis and metric

In more details, the DCGA $\mathbb{G}_{8,2}$ is defined over a 10-dimensional vector space. The base vectors of the space are basically divided into two groups: $\{\mathbf{e}_{o1}, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_{\infty 1}\}$, corresponding to the CGA vectors defined in Chapter 1, and a copy of this basis $\{\mathbf{e}_{o2}, \mathbf{e}_4, \mathbf{e}_5, \mathbf{e}_6, \mathbf{e}_{\infty 2}\}$. The inner products between them are defined in Table 7.1. Note that we highlighted in grey the two models of CGA that are included in DCGA.

Point of DCGA

A point of DCGA whose Euclidean coordinates are (x, y, z) is defined as the outer product of two

TABLE 7.1: Inner product between DCGA basis vectors.

	$\mathbf{e}_{o1}$	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_{\infty 1}$	$\mathbf{e}_{o2}$	$\mathbf{e}_4$	$\mathbf{e}_5$	$\mathbf{e}_6$	$\mathbf{e}_{\infty 2}$
$\mathbf{e}_{o1}$	0	0	0	0	−1	0	0	0	0	0
$\mathbf{e}_1$	0	1	0	0	0	0	0	0	0	0
$\mathbf{e}_2$	0	0	1	0	0	0	0	0	0	0
$\mathbf{e}_3$	0	0	0	1	0	0	0	0	0	0
$\mathbf{e}_{\infty 1}$	−1	0	0	0	0	0	0	0	0	0
$\mathbf{e}_{o2}$	0	0	0	0	0	0	0	0	0	−1
$\mathbf{e}_4$	0	0	0	0	0	0	1	0	0	0
$\mathbf{e}_5$	0	0	0	0	0	0	0	1	0	0
$\mathbf{e}_6$	0	0	0	0	0	0	0	0	1	0
$\mathbf{e}_{\infty 2}$	0	0	0	0	0	−1	0	0	0	0

CGA points with coordinates (x, y, z). By defining this two points $\mathbf{x}_1$ and $\mathbf{x}_2$ as:

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{e}_{o1} + x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3 + \tfrac{1}{2}(x^2+y^2+z^2)\mathbf{e}_{\infty 1} \\ \mathbf{x}_2 &= \mathbf{e}_{o2} + x\mathbf{e}_4 + y\mathbf{e}_5 + z\mathbf{e}_6 + \tfrac{1}{2}(x^2+y^2+z^2)\mathbf{e}_{\infty 2} \end{aligned} \tag{7.2.1}$$

The embedding of a DCGA point is defined as follows:

$$\mathbf{X} = \mathbf{x}_1 \wedge \mathbf{x}_2 \tag{7.2.2}$$

The development of this operation results in:

$$\begin{aligned} \mathbf{X} \;=\; & x^2\mathbf{e}_1\wedge\mathbf{e}_4 + y^2\mathbf{e}_2\wedge\mathbf{e}_5 + z^2\mathbf{e}_3\wedge\mathbf{e}_6 + xy(\mathbf{e}_1\wedge\mathbf{e}_5 + \mathbf{e}_2\wedge\mathbf{e}_4) \\ & + xz(\mathbf{e}_1\wedge\mathbf{e}_6 + \mathbf{e}_3\wedge\mathbf{e}_4) + yz(\mathbf{e}_2\wedge\mathbf{e}_6 + \mathbf{e}_3\wedge\mathbf{e}_5) \\ & + x(\mathbf{e}_{o1}\wedge\mathbf{e}_4 + \mathbf{e}_1\wedge\mathbf{e}_{o2}) + y(\mathbf{e}_{o1}\wedge\mathbf{e}_5 + \mathbf{e}_2\wedge\mathbf{e}_{o2}) + z(\mathbf{e}_{o1}\wedge\mathbf{e}_6 + \mathbf{e}_3\wedge\mathbf{e}_{o2}) \\ & + \mathbf{e}_{o1}\wedge\mathbf{e}_{o2} + \tfrac{1}{2}(x^2+y^2+z^2)(\mathbf{e}_{o1}\wedge\mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1}\wedge\mathbf{e}_{o2}) \\ & + \tfrac{1}{2}x(x^2+y^2+z^2)(\mathbf{e}_1\wedge\mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1}\wedge\mathbf{e}_4) + \tfrac{1}{2}y(x^2+y^2+z^2)(\mathbf{e}_2\wedge\mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1}\wedge\mathbf{e}_5) \\ & + \tfrac{1}{2}z(x^2+y^2+z^2)(\mathbf{e}_3\wedge\mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1}\wedge\mathbf{e}_6) + \tfrac{1}{4}(x^2+y^2+z^2)^2\mathbf{e}_{\infty 1}\wedge\mathbf{e}_{\infty 2} \end{aligned} \tag{7.2.3}$$

This high number of components is due to the fact that the representation of a point of DCGA was designed to not only define quadric surfaces but also quartic surfaces. To illustrate this, we highlight the components that contribute to construct quadrics in red. The other components are dedicated to the construction of quartics.

Quadrics

A general quadric merely consists in defining some operators that extract the components of $\mathbf{X}$. For a general quadric defined as:

$$ax^2 + by^2 + cz^2 + dxy + eyz + fzx + gx + hy + iz + j = 0 \tag{7.2.4}$$

This means that 4 operators are defined for the quadratic part:

$$\begin{aligned} \mathbf{T}_{x^2} &= \mathbf{e}_4 \wedge \mathbf{e}_1 \quad \mathbf{T}_{y^2} = \mathbf{e}_5 \wedge \mathbf{e}_2 \\ \mathbf{T}_{z^2} &= \mathbf{e}_6 \wedge \mathbf{e}_3 \quad \mathbf{T}_1 = -\mathbf{e}_{\infty 1} \wedge \mathbf{e}_{\infty 2} \end{aligned} \tag{7.2.5}$$

along with the 3 operators for the linear part:

$$\begin{aligned} \mathbf{T}_x &= \tfrac{1}{2}\Big(\mathbf{e}_1 \wedge \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1} \wedge \mathbf{e}_4\Big) \\ \mathbf{T}_y &= \tfrac{1}{2}\Big(\mathbf{e}_2 \wedge \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1} \wedge \mathbf{e}_5\Big) \\ \mathbf{T}_z &= \tfrac{1}{2}\Big(\mathbf{e}_3 \wedge \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1} \wedge \mathbf{e}_6\Big) \end{aligned} \tag{7.2.6}$$

and 3 operators for the cross terms:

$$\begin{aligned} \mathbf{T}_{xy} &= \tfrac{1}{2}\Big(\mathbf{e}_5 \wedge \mathbf{e}_1 + \mathbf{e}_4 \wedge \mathbf{e}_2\Big) \\ \mathbf{T}_{xz} &= \tfrac{1}{2}\Big(\mathbf{e}_6 \wedge \mathbf{e}_1 + \mathbf{e}_4 \wedge \mathbf{e}_3\Big) \\ \mathbf{T}_{yz} &= \tfrac{1}{2}\Big(\mathbf{e}_5 \wedge \mathbf{e}_3 + \mathbf{e}_6 \wedge \mathbf{e}_2\Big) \end{aligned} \tag{7.2.7}$$

Then, for example,

$$\mathbf{T}_{yz} \cdot \mathbf{X} = yz \tag{7.2.8}$$

A general quadric is defined as the bivector $\mathbf{Q}_{DCGA}$ with the following formula:

$$\mathbf{Q}_{DCGA} = \mathrm{a}\mathbf{T}_{x^2} + \mathrm{b}\mathbf{T}_{y^2} + \mathrm{c}\mathbf{T}_{z^2} + \mathrm{d}\mathbf{T}_{xy} + \mathrm{e}\mathbf{T}_{yz} + \mathrm{f}\mathbf{T}_{xz} + \mathrm{g}\mathbf{T}_x + \mathrm{h}\mathbf{T}_y + \mathrm{i}\mathbf{T}_z + \mathrm{j}\mathbf{T}_1 \tag{7.2.9}$$

Finally, we check that a point $\mathbf{x}$ is in a quadric if and only if:

$$\mathbf{q}_{DCGA} \cdot \mathbf{X} = 0 \tag{7.2.10}$$

DCGA not only supports the definition of general quadrics but also some quartic surfaces like Torus, cyclides (Dupin cyclides...).

Quadrics intersection

Plane tangent to a quadric

The tangent plane was defined using differential operators in DCGA. Let us consider a point $\mathbf{X}$

whose Euclidean coordinates are (x, y, z) and a DCGA quadric $\mathbf{Q}_{DCGA}$ defined as:

$$\mathbf{Q}_{DCGA} = \mathrm{a}\mathbf{T}_{x^2} + \mathrm{b}\mathbf{T}_{y^2} + \mathrm{c}\mathbf{T}_{z^2} + \mathrm{d}\mathbf{T}_{xy} + \mathrm{e}\mathbf{T}_{yz} + \mathrm{f}\mathbf{T}_{xz} + \mathrm{g}\mathbf{T}_{x} + \mathrm{h}\mathbf{T}_{y} + \mathrm{i}\mathbf{T}_{z} + \mathrm{j}\mathbf{T}_{1} \tag{7.2.11}$$

The differential operators along the axis are defined as:

$$\begin{aligned} \mathbf{D}_x &= (\mathbf{e}_1 \wedge \mathbf{e}_{\infty 1} + \mathbf{e}_4 \wedge \mathbf{e}_{\infty 2}) \\ \mathbf{D}_y &= (\mathbf{e}_2 \wedge \mathbf{e}_{\infty 1} + \mathbf{e}_5 \wedge \mathbf{e}_{\infty 2}) \\ \mathbf{D}_z &= (\mathbf{e}_3 \wedge \mathbf{e}_{\infty 1} + \mathbf{e}_6 \wedge \mathbf{e}_{\infty 2}) \end{aligned} \tag{7.2.12}$$

Then, using the commutator product, noted as $\times$ and defined in Chapter 1, the following properties hold:

$$\begin{aligned} \mathbf{D}_x \times \mathbf{Q}_{DCGA} &= 2\mathrm{a}\mathbf{T}_x + \mathrm{d}\mathbf{T}_y + \mathrm{e}\mathbf{T}_z + \mathrm{g}\mathbf{T}_1 \\ \mathbf{D}_y \times \mathbf{Q}_{DCGA} &= 2\mathrm{b}\mathbf{T}_y + \mathrm{d}\mathbf{T}_x + \mathrm{f}\mathbf{T}_z + \mathrm{h}\mathbf{T}_1 \\ \mathbf{D}_z \times \mathbf{Q}_{DCGA} &= 2\mathrm{c}\mathbf{T}_z + \mathrm{e}\mathbf{T}_x + \mathrm{f}\mathbf{T}_y + \mathrm{i}\mathbf{T}_1 \end{aligned} \tag{7.2.13}$$

This latter formula defines the normal vector to the quadric surface at any point of the surface. It is computed using the normal vector $\mathbf{n}_1$ defined in the first copy CGA and $\mathbf{n}_2$ along the second copy of the CGA basis vectors. $\mathbf{n}_1$ at the considered point can be defined as follows:

$$\mathbf{n}_1 = ((\mathbf{D}_x \times \mathbf{Q}_{DCGA}) \cdot \mathbf{X})\mathbf{e}_1 + ((\mathbf{D}_y \times \mathbf{Q}_{DCGA}) \cdot \mathbf{X})\mathbf{e}_2 + ((\mathbf{D}_z \times \mathbf{Q}_{DCGA}) \cdot \mathbf{X})\mathbf{e}_3 \tag{7.2.14}$$

Similarly, the normal vector $\mathbf{n}_2$ is:

$$\mathbf{n}_2 = ((\mathbf{D}_x \times \mathbf{Q}_{DCGA}) \cdot \mathbf{X})\mathbf{e}_4 + ((\mathbf{D}_y \times \mathbf{Q}_{DCGA}) \cdot \mathbf{X})\mathbf{e}_5 + ((\mathbf{D}_z \times \mathbf{Q}_{DCGA}) \cdot \mathbf{X})\mathbf{e}_6 \tag{7.2.15}$$

Now, the definition of the plane from normal vector is:

$$\boldsymbol{\pi} = (\mathbf{n}_1 + \mathrm{d}\mathbf{e}_{\infty 1}) \wedge (\mathbf{n}_2 + \mathrm{d}\mathbf{e}_{\infty 2}) \tag{7.2.16}$$

where d represents the orthogonal distance between the plane and the origin. Finally, the computation of the orthogonal distance can be simply performed as follows:

$$\mathrm{d} = \mathbf{n}_1 \cdot \mathbf{x}_1 \tag{7.2.17}$$

Where $\mathbf{x}_1$ is the point used to form the DCGA point $\mathbf{x}$.

Quadric-line intersection

DCGA also supports the construction of the intersection $\mathbf{p}_p$ of a quadric $\mathbf{q}_{DCGA}$ and a line $\mathbf{l}$. A line in DCGA can be defined as:

$$\mathbf{L} = \mathbf{l}_1 \wedge \mathbf{l}_2 \tag{7.2.18}$$

Where these two entities $\mathbf{l}_1$ and $\mathbf{l}_2$ can be expressed using the direction of the unit vector in CGA1 $\mathbf{d_1}$ and CGA2 $\mathbf{d}_1$ and a point of this line expressed in CGA1 $\mathbf{x}_1$ and CGA2 $\mathbf{x}_2$ as:

$$\mathbf{l}_1 = \mathbf{d}_1\mathbf{I}_\epsilon^{-1} - (\mathbf{x}_1 \cdot (\mathbf{d}_1\mathbf{I}_{\epsilon 1}^{-1})) \tag{7.2.19}$$

and:

$$\mathbf{l}_2 = \mathbf{d}_2\mathbf{I}_\epsilon^{-1} - (\mathbf{x}_2 \cdot (\mathbf{d}_2\mathbf{I}_{\epsilon 1}^{-1})) \tag{7.2.20}$$

Both $\mathbf{d}_1\mathbf{I}_{\epsilon 1}$ and $\mathbf{d}_2\mathbf{I}_{\epsilon 1}$ are 2-vectors, therefore, $\mathbf{l}$ is a 4-vector. Note that a line can similarly be obtained by the intersection of two DCGA planes as:

$$\mathbf{l} = \pi_1 \wedge \pi_2 \tag{7.2.21}$$

Finally, the intersection is computed as:

$$\mathbf{p}_p = \mathbf{q}_{DCGA} \wedge \mathbf{l} \tag{7.2.22}$$

Complexity of some major operations of DCGA

Let us first evaluate the computational cost of checking whether a point is on a quadric using the model of 7.1. $\mathbf{q}_{DCGA}$ has a total of 10 basis bivector components. For each basis bivector, at most 3 inner products (bivector $\wedge$ bivector) are performed, please refer to the formula of the inner of Chapter 1. Finally, the number of point component is 25. Thus, the product $\mathbf{q}_{DCGA} \cdot \mathbf{x}$ require $25 \times 3 \times 10 = 750$ products.

The cost of the computation of the tangent plane to a quadric corresponds to the cost of, first, the normal vector $\mathbf{n}_1$ and second the tangent plane. Firstly, the normal vector is defined as:

$$\mathbf{n}_1 = ((\mathbf{D}_x \times \mathbf{q}_{DCGA}) \cdot \mathbf{x})\mathbf{e}_1 + ((\mathbf{D}_y \times \mathbf{q}_{DCGA}) \cdot \mathbf{x})\mathbf{e}_2 + ((\mathbf{D}_z \times \mathbf{q}_{DCGA}) \cdot \mathbf{x})\mathbf{e}_3 \tag{7.2.23}$$

Equation (7.2.13) defined $\mathbf{D}_x, \mathbf{D}_y, \mathbf{D}_z$, and the commutator product of these operators with the quadric results in a 7-component bivector. Indeed, the extractions operators $\mathbf{T}_x, \mathbf{T}_y, \mathbf{T}_z$ are 2-components operator while $\mathbf{T}_1$ is a single component extraction operator. Each inner product

with $\mathbf{x}$ then has a computational cost of $7 \times 25 = 175$ products. This latter computation is repeated for each axis thus this results in $175 \times 3 = 525$ products.

Second, the tangent plane is obtained by using the normal vector and the orthogonal distance from the origin. The orthogonal distance is computed as:

$$\mathrm{d} = \mathbf{n}_1 \cdot \mathbf{x}_1 \tag{7.2.24}$$

Thus, this requires the computation of 3 inner products. Finally, the tangent plane is the results of the outer product:

$$\boldsymbol{\Pi} = (\mathbf{n}_1 + \mathrm{d}\mathbf{e}_{\infty 1}) \wedge (\mathbf{n}_2 + \mathrm{d}\mathbf{e}_{\infty 2}) \tag{7.2.25}$$

Both $(\mathbf{n}_1 + \mathrm{d}\mathbf{e}_{\infty 1})$ and $(\mathbf{n}_2 + \mathrm{d}\mathbf{e}_{\infty 2})$ are 4-components 1-vector. Thus, the computational cost of the outer product is $4 \times 4 = 16$. Hence, the total cost of the computation of the tangent plane is $525 + 16 = 541$ products.

The cost of the computation of the intersection between a quadric and a line consists in evaluating the cost of the outer product between a DCGA line $\mathbf{L}$ and the quadric of DCGA $\mathbf{Q}_{DCGA}$. In the previous section, we defined a line $\mathbf{L}$ as the 4-vector entity obtained by the outer product of the planes as follows:

$$\mathbf{L} = \boldsymbol{\Pi}_1 \wedge \boldsymbol{\Pi}_2 \tag{7.2.26}$$

A plane in DCGA is obtained as the outer product of two CGA planes whose number of components is 4. The result of the outer product between the two CGA planes may have non-zero components along the following components:

$$(\mathbf{e}_{14}, \mathbf{e}_{15}, \mathbf{e}_{16}, \mathbf{e}_{1\infty 2}, \mathbf{e}_{24}, \mathbf{e}_{25}, \mathbf{e}_{26}, \mathbf{e}_{2\infty 2}, \mathbf{e}_{34}, \mathbf{e}_{35}, \mathbf{e}_{36}, \mathbf{e}_{3\infty 2}, \mathbf{e}_{\infty 14}, \mathbf{e}_{\infty 15}, \mathbf{e}_{\infty 16}, \mathbf{e}_{\infty 14}, \mathbf{e}_{\infty 1\infty 2}) \tag{7.2.27}$$

Then, computing the outer product between two planes may have some results along the following basis quadvectors:

$$\begin{aligned}(&\mathbf{e}_{1245}, \mathbf{e}_{1246}, \mathbf{e}_{124\infty 2}, \mathbf{e}_{1256}, \mathbf{e}_{125\infty 2}, \mathbf{e}_{126\infty 2}, \mathbf{e}_{1345}, \mathbf{e}_{1346}, \mathbf{e}_{134\infty 2}, \mathbf{e}_{1356}, \mathbf{e}_{135\infty 2},\\ &\mathbf{e}_{136\infty 2}, \mathbf{e}_{1\infty 145}, \mathbf{e}_{1\infty 146}, \mathbf{e}_{1\infty 14\infty 2}, \mathbf{e}_{1\infty 156}, \mathbf{e}_{1\infty 15\infty 2}, \mathbf{e}_{1\infty 16\infty 2}, \mathbf{e}_{2345}, \mathbf{e}_{2346}, \mathbf{e}_{234\infty 2},\\ &\mathbf{e}_{2356}, \mathbf{e}_{235\infty 2}, \mathbf{e}_{236\infty 2}, \mathbf{e}_{2\infty 145}, \mathbf{e}_{2\infty 146}, \mathbf{e}_{2\infty 14\infty 2}, \mathbf{e}_{2\infty 156}, \mathbf{e}_{2\infty 15\infty 2}, \mathbf{e}_{2\infty 16\infty 2}, \mathbf{e}_{3\infty 145},\\ &\mathbf{e}_{3\infty 146}, \mathbf{e}_{3\infty 14\infty 2}, \mathbf{e}_{3\infty 156}, \mathbf{e}_{3\infty 15\infty 2}, \mathbf{e}_{3\infty 16\infty 2})\end{aligned} \tag{7.2.28}$$

This latter 4-vector has thus 36 components. Then, the outer product between this quadvector $\mathbf{l}$ and the quadric can be performed as:

$$\mathbf{Q}_{DCGA} \wedge \mathbf{L} \tag{7.2.29}$$

As the number of components of $\mathbf{Q}_{DCGA}$ is 25 and the number of components of $\mathbf{L}$ is 36. Then the cost of the outer product is $25 \times 36 = 900$ products. The following table summarizes the computational cost of the three features computed so far for DCGA.

TABLE 7.2: Computational features in number of Geometric Algebra operations for DCGA

Feature	**DCGA**
point is on a quadric	725
tangent plane	541
quadric line intersection	900

7.3 DPGA of $\mathbb{G}_{4,4}$

DPGA was adapted from the approach of Parkin [70] in 2012 and firstly introduced in 2015 by Goldman and Mann [38] and further developed by Du and Goldman and Mann [26].

Basis and metric

DPGA $\mathbb{G}_{4,4}$ is defined over a 8-dimensional vector space. In a similar way to DCGA, the base vectors of the space are divided into two groups: $\{\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3\}$ (corresponding to the projective Geometric Algebra vectors), and a copy of this basis $\{\mathbf{w}_0^*, \mathbf{w}_1^*, \mathbf{w}_2^*, \mathbf{w}_3^*\}$ such that $\mathbf{w}_i\mathbf{w}_i^* = 0.5 + \mathbf{w}_i \wedge \mathbf{w}_i^*, \forall i \in \{0,1,2,3\}$. To have more details, we show the inner products between any basis vectors in Table 7.3.

TABLE 7.3: Inner product between DPGA basis vectors.

	$\mathbf{w_0}$	$\mathbf{w_1}$	$\mathbf{w_2}$	$\mathbf{w_3}$	$\mathbf{w_0^*}$	$\mathbf{w_1^*}$	$\mathbf{w_2^*}$	$\mathbf{w_3^*}$
$\mathbf{w_0}$	0	0	0	0	0.5	0	0	0
$\mathbf{w_1}$	0	0	0	0	0	0.5	0	0
$\mathbf{w_2}$	0	0	0	0	0	0	0.5	0
$\mathbf{w_3}$	0	0	0	0	0	0	0	0.5
$\mathbf{w_0^*}$	0.5	0	0	0	0	0	0	0
$\mathbf{w_1^*}$	0	0.5	0	0	0	0	0	0
$\mathbf{w_2^*}$	0	0	0.5	0	0	0	0	0
$\mathbf{w_3^*}$	0	0	0	0.5	0	0	0	0

Point of DPGA

In DPGA, the entity representing a point whose Euclidean coordinates are (x, y, z) has two definitions, namely a primal and dual. Both definitions are the base to construct quadrics by means of sandwiching product. The definitions of the points are:

$$\begin{aligned} \mathbf{p} &= x\mathbf{w}_0 + y\mathbf{w}_1 + z\mathbf{w}_2 + w\mathbf{w}_3 \\ \mathbf{p}^* &= x\mathbf{w}_0^* + y\mathbf{w}_1^* + z\mathbf{w}_2^* + w\mathbf{w}_3^* \end{aligned} \tag{7.3.1}$$

Note that the dual definition denotes the fact that:

$$\mathbf{w}_i \cdot \mathbf{w}_j^* = \frac{1}{2}\delta_{i,j} \; \forall i, j = 0, \cdots 3 \tag{7.3.2}$$

Where $\delta_{i,j} = 1$ if $i = j$, else 0. This corresponds to the condition of the dual stated in Section 11 of [20].

Quadrics

Again, for a general quadric defined as:

$$\mathrm{a}x^2 + \mathrm{b}y^2 + \mathrm{c}z^2 + \mathrm{d}xy + \mathrm{e}yz + \mathrm{f}zx + \mathrm{g}x + \mathrm{h}y + \mathrm{i}z + \mathrm{j} = 0 \tag{7.3.3}$$

A quadric in DPGA is the bivector Q_{DPGA} defined as follows:

$$\begin{aligned} Q_{DPGA} =& 4\mathrm{a}\mathbf{w}_0^* \wedge \mathbf{w}_0 + 4\mathrm{b}\mathbf{w}_1^* \wedge \mathbf{w}_1 + 4\mathrm{c}\mathbf{w}_2^* \wedge \mathbf{w}_2 + 4\mathrm{j}\mathbf{w}_3^* \wedge \mathbf{w}_3 + 2\mathrm{d}(\mathbf{w}_0^* \wedge \mathbf{w}_1 + \mathbf{w}_1^* \wedge \mathbf{w}_0) \\ &+ 2\mathrm{e}(\mathbf{w}_0^* \wedge \mathbf{w}_2 + \mathbf{w}_2^* \wedge \mathbf{w}_0) + 2\mathrm{f}(\mathbf{w}_1^* \wedge \mathbf{w}_2 + \mathbf{w}_2^* \wedge \mathbf{w}_1) + 2\mathrm{g}(\mathbf{w}_0^* \wedge \mathbf{w}_3 + \mathbf{w}_3^* \wedge \mathbf{w}_0) \\ &+ 2\mathrm{h}(\mathbf{w}_1^* \wedge \mathbf{w}_3 + \mathbf{w}_3^* \wedge \mathbf{w}_1) + 2\mathrm{i}(\mathbf{w}_2^* \wedge \mathbf{w}_3 + \mathbf{w}_3^* \wedge \mathbf{w}_2) \end{aligned} \tag{7.3.4}$$

Finally, a point (x, y, z) is in the quadric $\mathbf{Q}_{DPGA}$ if and only if

$$\mathbf{p} \cdot \mathbf{Q}_{DPGA} \cdot \mathbf{p}^* = 0 \tag{7.3.5}$$

Let us call $\mathbf{f}_{DPGA} = \mathbf{p} \cdot \mathbf{Q}_{DPGA} \cdot \mathbf{p}^*$. Then to investigate numerical properties of the quadric computation, we develop the formula $\mathbf{p} \cdot \mathbf{Q}_{DPGA} \cdot \mathbf{p}^*$:

$$\begin{aligned} \mathbf{f}_{DPGA} &= \mathbf{p} \cdot \mathbf{q}_{DPGA} \cdot \mathbf{p}^* \\ &= \Big(2\mathrm{a}x\mathbf{w}_0 + \mathrm{d}x\mathbf{w}_1 + \mathrm{e}x\mathbf{w}_2 + \mathrm{g}x\mathbf{w}_3 + 2\mathrm{b}y\mathbf{w}_1 + \mathrm{d}y\mathbf{w}_0 + \mathrm{f}y\mathbf{w}_2 + \mathrm{h}y\mathbf{w}_3 \\ &\quad + 2\mathrm{c}z\mathbf{w}_2 + \mathrm{e}z\mathbf{w}_0 + \mathrm{f}z\mathbf{w}_1 + \mathrm{i}z\mathbf{w}_3 + 2\mathrm{j}\mathbf{w}_3 + \mathrm{g}\mathbf{w}_0 + \mathrm{h}\mathbf{w}_1 + \mathrm{i}\mathbf{w}_2\Big) \cdot \mathbf{p}^* \\ &= \Big((2\mathrm{a}x + \mathrm{d}y + \mathrm{e}z + \mathrm{g})\mathbf{w}_0 + (2\mathrm{b}y + \mathrm{d}x + \mathrm{f}z + \mathrm{h})\mathbf{w}_1 \\ &\quad + (2\mathrm{c}z + \mathrm{e}x + \mathrm{f}y + \mathrm{i})\mathbf{w}_2 + (\mathrm{i}z + \mathrm{g}x + \mathrm{h}y + 2\mathrm{j})\mathbf{w}_3)\Big) \cdot \mathbf{p}^* \end{aligned} \tag{7.3.6}$$

The last inner product results in:

$$\begin{aligned}\mathbf{f}_{DPGA} &= \mathbf{p}\cdot\mathbf{q}_{DPGA}\cdot\mathbf{p}^* \\ &= \mathrm{a}x^2+0.5\mathrm{d}xy+0.5\mathrm{e}xz+0.5\mathrm{g}x+\mathrm{b}y^2+0.5\mathrm{d}xy+0.5\mathrm{f}yz+0.5\mathrm{h}y \\ &\quad +\mathrm{c}z^2+0.5\mathrm{e}xz+0.5\mathrm{f}yz+0.5\mathrm{i}z+0.5\mathrm{i}z+0.5\mathrm{g}x+0.5\mathrm{h}y+\mathrm{j} \\ &= \mathrm{a}x^2+\mathrm{b}y^2+\mathrm{c}z^2+\mathrm{d}xy+\mathrm{e}xz+\mathrm{f}yz+\mathrm{g}x+\mathrm{h}y+\mathrm{i}z+\mathrm{j}\end{aligned} \tag{7.3.7}$$

This latter development is the base to determine the number of operations in the computation of $\mathbf{p}\cdot\mathbf{Q}_{DPGA}\cdot\mathbf{p}^*$.

Plane tangent to a quadric

In a similar way as CGA, DPGA supports the computation of the tangent plane π^* to a quadric $\mathbf{Q}_{DPGA}$ at a dual point $\mathbf{p}^*$ as follows:

$$\pi^* = \mathbf{Q}_{DPGA}\cdot\mathbf{p}^* \tag{7.3.8}$$

Quadric-line intersection

DPGA also supports the quadric-line intersection. Given a line defined as the primal and dual $\mathbf{L}=\mathbf{x}_1\wedge\mathbf{x}_2$ and $\mathbf{L}^*=\mathbf{x}_1^*\wedge\mathbf{x}_2^*$ and the quadric bivector $\mathbf{Q}_{DPGA}$. The intersection bivector called $\mathbf{P}_p$ is defined as follows:

$$\mathbf{P}_p = (\mathbf{l}^*\wedge\mathbf{q}_{DPGA}\wedge\mathbf{l})\cdot\mathbf{I} \tag{7.3.9}$$

where $\mathbf{I}$ is the pseudoscalar of $\mathbb{G}_{4,4}$ defined as:

$$\mathbf{I}=\mathbf{w}_0\wedge\mathbf{w}_1\wedge\mathbf{w}_2\wedge\mathbf{w}_3\wedge\mathbf{w}_0^*\wedge\mathbf{w}_1^*\wedge\mathbf{w}_2^*\wedge\mathbf{w}_3^* \tag{7.3.10}$$

Complexity of some major operations of DPGA

$\mathbf{Q}_{DPGA}$ has a total of 16 basis bivector components. For each basis bivector, 2 inner products are performed. Thus, the first product $\mathbf{p}\cdot\mathbf{Q}_{DCGA}$ will require $4\times2\times16=128$ inner products. As previously seen, the resulting entity is a vector with 4 components. Hence, the second inner product requires $4\times4=16$ products. This results in 144 products.

Let us now evaluate the cost of the intersection between a quadric $\mathbf{q}_{DPGA}$ and a line $\mathbf{L}$ and $\mathbf{L}^*$. The line $\mathbf{L}^*$ is obtained by the outer product of two points $\mathbf{x}_1$ and $\mathbf{x}_2$ whose number of components is

4. Thus, a line $\mathbf{L}$ has 6 components along the bivector basis:

$$(\mathbf{w}_0 \wedge \mathbf{w}_1, \mathbf{w}_0 \wedge \mathbf{w}_2, \mathbf{w}_0 \wedge \mathbf{w}_3, \mathbf{w}_1 \wedge \mathbf{w}_2, \mathbf{w}_1 \wedge \mathbf{w}_3, \mathbf{w}_2 \wedge \mathbf{w}_3) \tag{7.3.11}$$

and a line $\mathbf{L}^*$ has the following bivector basis components:

$$(\mathbf{w}_0^* \wedge \mathbf{w}_1^*, \mathbf{w}_0^* \wedge \mathbf{w}_2^*, \mathbf{w}_0^* \wedge \mathbf{w}_3^*, \mathbf{w}_1^* \wedge \mathbf{w}_2^*, \mathbf{w}_1^* \wedge \mathbf{w}_3^*, \mathbf{w}_2^* \wedge \mathbf{w}_3^*) \tag{7.3.12}$$

The number of components of the quadric is 16 and the number of components of the line is 6. Then, the computational cost of the outer product $\mathbf{l}^* \wedge \mathbf{Q}_{DPGA}$ is $6 \times 16 = 96$ outer products. The result is a 4-vector and may have components along the quadvector basis:

$$\begin{aligned}
&(\mathbf{w}_0^* \wedge \mathbf{w}_1^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_0, \mathbf{w}_0^* \wedge \mathbf{w}_1^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_1, \mathbf{w}_0^* \wedge \mathbf{w}_1^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_2, \mathbf{w}_0^* \wedge \mathbf{w}_1^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3 \\
&\mathbf{w}_0^* \wedge \mathbf{w}_1^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_0, \mathbf{w}_0^* \wedge \mathbf{w}_1^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_1, \mathbf{w}_0^* \wedge \mathbf{w}_1^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_2, \mathbf{w}_0^* \wedge \mathbf{w}_1^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_3 \\
&\mathbf{w}_0^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_0, \mathbf{w}_0^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_1, \mathbf{w}_0^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_2, \mathbf{w}_0^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_3 \\
&\mathbf{w}_1^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_0, \mathbf{w}_1^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_1, \mathbf{w}_1^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_2, \mathbf{w}_1^* \wedge \mathbf{w}_2^* \wedge \mathbf{w}_3^* \wedge \mathbf{w}_3)
\end{aligned} \tag{7.3.13}$$

Thus, the resulting entity has 16 components. Furthermore, the line $\mathbf{l}$ has 6 components. Hence, the cost of the final outer product is $16 \times 6 = 96$ outer products. Finally, the total operation cost is thus $96 + 96 = 192$ products.

The cost of the computation of the tangent plane at a point $\mathbf{p}$ is the cost of the following product:

$$\pi^* = \mathbf{Q}_{DPGA} \cdot \mathbf{p}^* \tag{7.3.14}$$

Considering the fact that the number of components of $\mathbf{p}^*$ is 4 and the number of components of $\mathbf{Q}_{DPGA}$ is 16. Then the computational cost of the computation of the tangent plane is $16 \times 4 = 64$ products.

The following table summarizes the computational cost of the three features computed so far for DPGA.

TABLE 7.4: Computational features in number of Geometric Algebra operations for DPGA

Feature	**DPGA**
point is on a quadric	144
tangent plane	64
quadric line intersection	192

Chapter 8

Quadric conformal Geometric Algebra

Our proposed framework, referred to as Quadric Conformal Geometric Algebra (QCGA) hereafter, is an extension of CGA, specifically dedicated to quadric surfaces. Through generalizing the conic construction in $\mathbb{R}^2$ by Perwass [73], QCGA is capable of constructing quadric surfaces using either control points or implicit equations. Moreover, QCGA can compute the intersection of quadric surfaces, the surface tangent, and normal vectors for a quadric surface point.

8.1 QCGA definition

This section introduces QCGA. We specify its basis vectors and give the definition of a point.

8.1.1 QCGA basis and metric

A first advantage of this algebra is to construct general quadric as the outer product of control points. Algebraically, a quadric surface requires at least a 10-dimensional space as stated in Section 6. In a similar way as CGA, we add some null basis vectors such that the blades and operators of this algebra have geometrical meanings. The result is QCGA $\mathbb{G}_{9,6}$, defined over a 15-dimensional vector space. The base vectors of the space are divided into three groups: $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$, corresponding to the Euclidean vectors in $\mathbb{R}^3$, $\{\mathbf{e}_{o1}, \mathbf{e}_{o2}, \mathbf{e}_{o3}, \mathbf{e}_{o4}, \mathbf{e}_{o5}, \mathbf{e}_{o6}\}$, and $\{\mathbf{e}_{\infty 1}, \mathbf{e}_{\infty 2}, \mathbf{e}_{\infty 3}, \mathbf{e}_{\infty 4}, \mathbf{e}_{\infty 5}, \mathbf{e}_{\infty 6}\}$. These basis vectors are the support for bringing geometric intuitions to the algebra. The inner products between them are defined in Table 8.1.

As stated in Section 4.5, a diagonal metric matrix is useful in order to implement the algebra. The algebra $\mathbb{G}_{9,6}$ generated by the Euclidean basis $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$, and 6 basis vectors $\{\mathbf{e}_{+1}, \mathbf{e}_{+2}, \mathbf{e}_{+3}, \mathbf{e}_{+4}, \mathbf{e}_{+5}, \mathbf{e}_{+6}\}$ each of which is squared to $+1$ along with six other basis vectors $\{\mathbf{e}_{-1}, \mathbf{e}_{-2}, \mathbf{e}_{-1}, \mathbf{e}_{-4}, \mathbf{e}_{-5}, \mathbf{e}_{-6}\}$ each of which is squared to -1 corresponds to a diagonal metric matrix. The

TABLE 8.1: Inner product between QCGA basis vectors. The dots denote some inner products that result to 0.

	$\mathbf{e}_1$	$\mathbf{e}_2$	$\mathbf{e}_3$	$\mathbf{e}_{o1}$	$\mathbf{e}_{\infty 1}$	$\mathbf{e}_{o2}$	$\mathbf{e}_{\infty 2}$	$\mathbf{e}_{o3}$	$\mathbf{e}_{\infty 3}$	$\mathbf{e}_{o4}$	$\mathbf{e}_{\infty 4}$	$\mathbf{e}_{o5}$	$\mathbf{e}_{\infty 5}$	$\mathbf{e}_{o6}$	$\mathbf{e}_{\infty 6}$
$\mathbf{e}_1$	1	0	0	·	·	·	·	·	·	·	·	·	·	·	·
$\mathbf{e}_2$	0	1	0	·	·	·	·	·	·	·	·	·	·	·	·
$\mathbf{e}_3$	0	0	1	·	·	·	·	·	·	·	·	·	·	·	·
$\mathbf{e}_{o1}$	·	·	·	0	−1	·	·	·	·	·	·	·	·	·	·
$\mathbf{e}_{\infty 1}$	·	·	·	−1	0	·	·	·	·	·	·	·	·	·	·
$\mathbf{e}_{o2}$	·	·	·	·	·	0	−1	·	·	·	·	·	·	·	·
$\mathbf{e}_{\infty 2}$	·	·	·	·	·	−1	0	·	·	·	·	·	·	·	·
$\mathbf{e}_{o3}$	·	·	·	·	·	·	·	0	−1	·	·	·	·	·	·
$\mathbf{e}_{\infty 3}$	·	·	·	·	·	·	·	−1	0	·	·	·	·	·	·
$\mathbf{e}_{o4}$	·	·	·	·	·	·	·	·	·	0	−1	·	·	·	·
$\mathbf{e}_{\infty 4}$	·	·	·	·	·	·	·	·	·	−1	0	·	·	·	·
$\mathbf{e}_{o5}$	·	·	·	·	·	·	·	·	·	·	·	0	−1	·	·
$\mathbf{e}_{\infty 5}$	·	·	·	·	·	·	·	·	·	·	·	−1	0	·	·
$\mathbf{e}_{o6}$	·	·	·	·	·	·	·	·	·	·	·	·	·	0	−1
$\mathbf{e}_{\infty 6}$	·	·	·	·	·	·	·	·	·	·	·	·	·	−1	0

transformation from the original basis to this diagonal metric can be defined as follows:

$$\mathbf{e}_{\infty i} = \mathbf{e}_{+i} + \mathbf{e}_{-i} \quad \mathbf{e}_{oi} = \frac{1}{2}(\mathbf{e}_{-i} - \mathbf{e}_{+i}) \quad i \in \{1, \cdots, 6\} \tag{8.1.1}$$

To simplify the computations, we define the null basis vector $\mathbf{e}_\infty$ representing the point at infinity:

$$\mathbf{e}_\infty = \tfrac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}) \tag{8.1.2}$$

as well as the null basis vectors:

$$\mathbf{e}_o = \mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3} \tag{8.1.3}$$

note that $\mathbf{e}_\infty \cdot \mathbf{e}_o = -1$ and $\mathbf{e}_\infty^2 = \mathbf{e}_o^2 = 0$.

To define geometric primitives, we introduce the following 6-blades:

$$\begin{aligned} \mathbf{I}_\infty &= \mathbf{e}_{\infty 1} \wedge \mathbf{e}_{\infty 2} \wedge \mathbf{e}_{\infty 3} \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6}, \\ \mathbf{I}_o &= \mathbf{e}_{o1} \wedge \mathbf{e}_{o2} \wedge \mathbf{e}_{o3} \wedge \mathbf{e}_{o4} \wedge \mathbf{e}_{o5} \wedge \mathbf{e}_{o6} \end{aligned} \tag{8.1.4}$$

the following 5-blades:

$$\begin{aligned} \mathbf{I}_\infty^{\triangleright} &= (\mathbf{e}_{\infty 1} - \mathbf{e}_{\infty 2}) \wedge (\mathbf{e}_{\infty 2} - \mathbf{e}_{\infty 3}) \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \\ \mathbf{I}_o^{\triangleright} &= (\mathbf{e}_{o1} - \mathbf{e}_{o2}) \wedge (\mathbf{e}_{o2} - \mathbf{e}_{o3}) \wedge \mathbf{e}_{o4} \wedge \mathbf{e}_{o5} \wedge \mathbf{e}_{o6} \end{aligned} \tag{8.1.5}$$

These blades have very useful properties that will be highlighted in the sequel of this chapter. We also define the pseudo-scalar of $\mathbb{R}^3$:

$$\mathbf{I}_\epsilon = \mathbf{e}_1 \wedge \mathbf{e}_2 \wedge \mathbf{e}_3 \tag{8.1.6}$$

and the pseudo-scalar:

$$\mathbf{I} = \mathbf{I}_\epsilon \wedge \mathbf{I}_\infty \wedge \mathbf{I}_o \tag{8.1.7}$$

The inverse of the pseudo-scalar results in:

$$\mathbf{I}^{-1} = -\mathbf{I} \tag{8.1.8}$$

As stated in the first chapter, the dual of a multivector indicates division by the pseudo-scalar, e.g., $\mathbf{A}^* = -\mathbf{A}\mathbf{I}$, $\mathbf{A} = \mathbf{A}^*\mathbf{I}$. From [48] and in Equation (1.19) of [54], we have the useful duality between outer and inner products of non-scalar blades $\mathbf{A}$ and $\mathbf{B}$ in Geometric Algebra:

$$(\mathbf{A} \wedge \mathbf{B})^* = \mathbf{A} \cdot \mathbf{B}^*, \quad \mathbf{A} \wedge (\mathbf{B}^*) = (\mathbf{A} \cdot \mathbf{B})^*, \quad \mathbf{A} \wedge (\mathbf{B}\mathbf{I}) = (\mathbf{A} \cdot \mathbf{B})\mathbf{I} \tag{8.1.9}$$

which indicates that

$$\mathbf{A} \wedge \mathbf{B} = 0 \;\Leftrightarrow\; \mathbf{a} \cdot \mathbf{b}^* = 0, \quad \mathbf{A} \cdot \mathbf{B} = 0 \;\Leftrightarrow\; \mathbf{A} \wedge \mathbf{B}^* = 0 \tag{8.1.10}$$

8.1.2 Point in QCGA

The construction of the Geometric Algebra element representing a point $\mathbf{x}$ of QCGA is specifically designed to both define quadric surfaces and reproduce the convenient geometric properties of a CGA point. Namely, the element corresponding to the Euclidean point $\mathbf{x}_\epsilon = x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3 \in \mathbb{R}^3$ is defined as

$$\mathbf{x} = \mathbf{x}_\epsilon + \tfrac{1}{2}(x^2\mathbf{e}_{\infty 1} + y^2\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}) + xy\mathbf{e}_{\infty 4} + xz\mathbf{e}_{\infty 5} + yz\mathbf{e}_{\infty 6} + \mathbf{e}_o \tag{8.1.11}$$

The null vectors $\mathbf{e}_{o4}, \mathbf{e}_{o5}, \mathbf{e}_{o6}$ are not present in the definition of the point. This is to keep the convenient properties of CGA points, namely, the inner product between two points is proportional

to the squared distance between them. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two points, their inner product is

$$\begin{aligned}\mathbf{x}_1 \cdot \mathbf{x}_2 =& \Big(x_1\mathbf{e}_1 + y_1\mathbf{e}_2 + z_1\mathbf{e}_3 + \frac{1}{2}x_1^2\mathbf{e}_{\infty 1} + \frac{1}{2}y_1^2\mathbf{e}_{\infty 2} + \frac{1}{2}z_1^2\mathbf{e}_{\infty 3} + \\ & x_1y_1\mathbf{e}_{\infty 4} + x_1z_1\mathbf{e}_{\infty 5} + y_1z_1\mathbf{e}_{\infty 6} + \mathbf{e}_o\Big)\cdot \\ & \Big(x_2\mathbf{e}_1 + y_2\mathbf{e}_2 + z_2\mathbf{e}_3 + \frac{1}{2}x_2^2\mathbf{e}_{\infty 1} + \frac{1}{2}y_2^2\mathbf{e}_{\infty 2} + \frac{1}{2}z_2^2\mathbf{e}_{\infty 3} + \\ & x_2y_2\mathbf{e}_{\infty 4} + x_2z_2\mathbf{e}_{\infty 5} + y_2z_2\mathbf{e}_{\infty 6} + \mathbf{e}_o\Big)\end{aligned} \tag{8.1.12}$$

from which together with Table 8.1, it follows that

$$\begin{aligned}\mathbf{x}_1 \cdot \mathbf{x}_2 &= \Big(x_1x_2 + y_1y_2 + z_1z_2 - \frac{1}{2}x_1^2 - \frac{1}{2}x_2^2 - \frac{1}{2}y_1^2 - \frac{1}{2}y_2^2 - \frac{1}{2}z_1^2 - \frac{1}{2}z_2^2\Big) \\ &= -\frac{1}{2}\left\|\mathbf{x}_{1\epsilon} - \mathbf{x}_{2\epsilon}\right\|^2\end{aligned} \tag{8.1.13}$$

We see that the inner product is equivalent to the minus half of the squared Euclidean distance between $\mathbf{x}_1$ and $\mathbf{x}_2$.

8.1.3 Normalization

It is worth mentioning that the properties previously obtained requires that the three components $\mathbf{e}_{o1}, \mathbf{e}_{o2}, \mathbf{e}_{o3}$ are normalized, or equivalently the $\mathbf{e}_o$ component has to be normalized. This also means that any scaled point has to represent the same point.

To fulfil this requirement, we define the normalization of the entity representing points.

Proposition 8.1.1. *For QCGA point* $\mathbf{x}$*, the normalization is merely computed through an averaging of* $\mathbf{e}_{o1}, \mathbf{e}_{o2}, \mathbf{e}_{o3}$ *components thus of* $\mathbf{e}_o$ *component, namely as:*

$$-\frac{\mathbf{x}}{\mathbf{x}\cdot\mathbf{e}_\infty} \tag{8.1.14}$$

Proof. A scale α on $\mathbf{x}$ acts the same way on all null basis vectors of $\mathbf{x}$:

$$\begin{aligned}\alpha\mathbf{x} = \alpha\mathbf{x}_\epsilon + \tfrac{1}{2}\alpha(x^2\mathbf{e}_{\infty 1} + y^2\mathrm{w}\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}) + xy\alpha\mathbf{e}_{\infty 4} + xz\alpha\mathbf{e}_{\infty 5} + yz\alpha\mathbf{e}_{\infty 6} \\ + \alpha\mathbf{e}_{o1} + \alpha\mathbf{e}_{o2} + \alpha\mathbf{e}_{o3}\end{aligned} \tag{8.1.15}$$

The metric of QCGA indicates (see Table 8.1):

$$\begin{aligned} \alpha\mathbf{x}\cdot\mathbf{e}_{\infty 1} &= -\alpha \\ \alpha\mathbf{x}\cdot\mathbf{e}_{\infty 2} &= -\alpha \\ \alpha\mathbf{x}\cdot\mathbf{e}_{\infty 3} &= -\alpha \end{aligned} \tag{8.1.16}$$

Thus, if $\alpha \neq 0$:

$$\begin{aligned} \frac{-3\alpha\mathbf{x}}{\alpha\mathbf{x}\cdot(\mathbf{e}_{\infty 1}+\mathbf{e}_{\infty 2}+\mathbf{e}_{\infty 3})}\cdot\mathbf{e}_{\infty 1} &= -\tfrac{-3\alpha\mathbf{x}}{-3\alpha}\cdot\mathbf{e}_{\infty 1} \\ &= \mathbf{x}\cdot\mathbf{e}_{\infty 1} = -1 \end{aligned} \tag{8.1.17}$$

A similar result is obtained with $\mathbf{e}_{\infty 2}$ and $\mathbf{e}_{\infty 3}$:

$$\frac{-3\alpha\mathbf{x}}{\alpha\mathbf{x}\cdot(\mathbf{e}_{\infty 1}+\mathbf{e}_{\infty 2}+\mathbf{e}_{\infty 3})}\cdot\mathbf{e}_{\infty 2} = \mathbf{x}\cdot\mathbf{e}_{\infty 2} = -1 \tag{8.1.18}$$

$$\frac{-3\alpha\mathbf{x}}{\alpha\mathbf{x}\cdot(\mathbf{e}_{\infty 1}+\mathbf{e}_{\infty 2}+\mathbf{e}_{\infty 3})}\cdot\mathbf{e}_{\infty 3} = \mathbf{x}\cdot\mathbf{e}_{\infty 3} = -1 \tag{8.1.19}$$

□

Thus, we checked that for any scaled points $\mathbf{x}_1, \mathbf{x}_2$:

$$\frac{\mathbf{x}_1}{\mathbf{x}_1\cdot\mathbf{e}_\infty}\cdot\frac{\mathbf{x}_2}{\mathbf{x}_2\cdot\mathbf{e}_\infty} = -\frac{1}{2}\left\|\mathbf{x}_{1\epsilon}-\mathbf{x}_{2\epsilon}\right\|^2 \tag{8.1.20}$$

8.1.4 Embedded of CGA points

Proposition 8.1.2. *CGA points are included in the QCGA points. Indeed, the outer product between a QCGA point* $\mathbf{x}$ *with* $\mathbf{I}_\infty^\triangleright$ *and* $\mathbf{I}_o^\triangleright$ *can be rewritten as an equivalent of the CGA point. In other words:*

$$\mathbf{x}\wedge\mathbf{I}_\infty^\triangleright\wedge\mathbf{I}_o^\triangleright = \left(\mathbf{x}_\epsilon+\tfrac{1}{2}\mathbf{x}_\epsilon^2\mathbf{e}_\infty+\mathbf{e}_o\right)\wedge\mathbf{I}_\infty^\triangleright\wedge\mathbf{I}_o^\triangleright \tag{8.1.21}$$

Proof. First the outer product between $\mathbf{x}$ and $\mathbf{I}_\infty^\triangleright$ removes all the $\mathbf{e}_{\infty 4}, \mathbf{e}_{\infty 5}, \mathbf{e}_{\infty 6}$ components, namely:

$$\mathbf{x}\wedge\mathbf{I}_\infty^\triangleright\wedge\mathbf{I}_o^\triangleright = \left(\mathbf{x}_\epsilon+\tfrac{1}{2}\left(x^2\mathbf{e}_{\infty 1}+y^2\mathbf{e}_{\infty 2}+z^2\mathbf{e}_{\infty 3}\right)+\mathbf{e}_o\right)\wedge\mathbf{I}_\infty^\triangleright\wedge\mathbf{I}_o^\triangleright \tag{8.1.22}$$

Then, on one hand, focusing on the $\frac{1}{2}\left(x^2\mathbf{e}_{\infty 1}+y^2\mathbf{e}_{\infty 2}+z^2\mathbf{e}_{\infty 3}\right)\wedge\mathbf{I}_\infty^\triangleright\wedge\mathbf{I}_o^\triangleright$ part, results in:

$$\begin{aligned} \tfrac{1}{2}\left(x^2\mathbf{e}_{\infty 1}+y^2\mathbf{e}_{\infty 2}+z^2\mathbf{e}_{\infty 3}\right)\wedge\mathbf{I}_\infty^\triangleright\wedge\mathbf{I}_o^\triangleright = \; & \tfrac{1}{2}\left(x^2+y^2+z^2\right)\mathbf{e}_{\infty 1}\wedge\mathbf{e}_{\infty 2} \\ & \wedge\mathbf{e}_{\infty 3}\wedge\mathbf{e}_{\infty 4}\wedge\mathbf{e}_{\infty 5}\wedge\mathbf{e}_{\infty 6}\wedge\mathbf{I}_o^\triangleright \end{aligned} \tag{8.1.23}$$

On the other hand:

$$\begin{aligned}\tfrac{1}{2}\left(x^2+y^2+z^2\right)\tfrac{1}{3}\left(\mathbf{e}_{\infty 1}+\mathbf{e}_{\infty 2}+\mathbf{e}_{\infty 3}\right)\wedge\mathbf{I}_\infty^{\triangleright}\wedge\mathbf{I}_o^{\triangleright} = \quad & \tfrac{1}{2}\left(x^2+y^2+z^2\right)\mathbf{e}_{\infty 1}\wedge\mathbf{e}_{\infty 2} \\ & \wedge\mathbf{e}_{\infty 3}\wedge\mathbf{e}_{\infty 4}\wedge\mathbf{e}_{\infty 5}\wedge\mathbf{e}_{\infty 6}\wedge\mathbf{I}_o^{\triangleright}\end{aligned} \tag{8.1.24}$$

Thus, this results in:

$$\mathbf{x}\wedge\mathbf{I}_\infty^{\triangleright}\wedge\mathbf{I}_o^{\triangleright} \quad = \left(\mathbf{x}_\epsilon+\tfrac{1}{2}\mathbf{x}_\epsilon^2\mathbf{e}_\infty+\mathbf{e}_o\right)\wedge\mathbf{I}_\infty^{\triangleright}\wedge\mathbf{I}_o^{\triangleright} \tag{8.1.25}$$

□

Finally, the outer product between a point of QCGA with both $\mathbf{I}_\infty^{\triangleright}$ and $\mathbf{I}_o^{\triangleright}$ can be replaced by an equivalent of a CGA point $\mathbf{x}_C$ as:

$$\mathbf{x}_C = \left(\mathbf{x}_\epsilon+\tfrac{1}{2}\mathbf{x}_\epsilon^2\mathbf{e}_\infty+\mathbf{e}_o\right)\wedge\mathbf{I}_\infty^{\triangleright}\wedge\mathbf{I}_o^{\triangleright}\left(\mathbf{I}_\infty^{\triangleright}\wedge\mathbf{I}_o^{\triangleright}\right)^{-1} \tag{8.1.26}$$

Note that the subscript C symbolizes the fact that it is an equivalent of a **C**GA point.

8.2 Round objects of QCGA

As seen a CGA point is well embedded in QCGA, thus the objects defined in CGA are also defined in QCGA. We define round objects in an equivalent way as in CGA using points $\mathbf{x}_C$ in the primal.

8.2.1 Sphere

8.2.1.1 Primal representation of a sphere

As in CGA, we define a sphere **S** using four points as the following 14-blade:

$$\mathbf{S} = \mathbf{x}_1\wedge\mathbf{x}_2\wedge\mathbf{x}_3\wedge\mathbf{x}_4\wedge\mathbf{I}_\infty^{\triangleright}\wedge\mathbf{I}_o^{\triangleright} \tag{8.2.1}$$

Using Equation (8.1.21), this can be rewritten as:

$$\mathbf{S} = \mathbf{x}_{C1}\wedge\mathbf{x}_{C2}\wedge\mathbf{x}_{C3}\wedge\mathbf{x}_{C4}\wedge\mathbf{I}_\infty^{\triangleright}\wedge\mathbf{I}_o^{\triangleright} \tag{8.2.2}$$

The development of the above Equation with respect to $\mathbf{x}_{C4}$ yields:

$$\mathbf{S} = \mathbf{x}_{C1} \wedge \mathbf{x}_{C2} \wedge \mathbf{x}_{C3} \wedge \left(\frac{1}{2}(x_{4\epsilon}^2 \mathbf{e}_\infty)\mathbf{I}_\infty^\triangleright \wedge \mathbf{I}_o^\triangleright - 3\mathbf{I}_\infty^\triangleright \wedge \mathbf{I}_o + \mathbf{x}_{4\epsilon}\mathbf{I}_\infty^\triangleright \wedge \mathbf{I}_o^\triangleright\right) \tag{8.2.3}$$

Then, the development with $\mathbf{x}_{C3}$ gives:

$$\begin{aligned}\mathbf{S} =& \mathbf{x}_{C1} \wedge \mathbf{x}_{C2} \wedge \Big(\mathbf{x}_{3\epsilon} \wedge \mathbf{x}_{4\epsilon}\mathbf{I}_o^\triangleright \wedge \mathbf{I}_\infty^\triangleright + 3(\mathbf{x}_{3\epsilon} - \mathbf{x}_{4\epsilon})\mathbf{I}_o \wedge \mathbf{I}_\infty^\triangleright + \\ & \frac{1}{2}\left\|\mathbf{x}_{4\epsilon}\right\|^2 \mathbf{x}_{3\epsilon}\mathbf{I}_\infty \wedge \mathbf{I}_o^\triangleright - \frac{1}{2}\left\|\mathbf{x}_{3\epsilon}\right\|^2 \mathbf{x}_{4\epsilon}\mathbf{I}_\infty \wedge \mathbf{I}_o^\triangleright + \tfrac{3}{2}(\left\|\mathbf{x}_{4\epsilon}\right\|^2 - \left\|\mathbf{x}_{3\epsilon}\right\|^2)\mathbf{I}_o \wedge \mathbf{I}_\infty\Big)\end{aligned} \tag{8.2.4}$$

Again, we remark that the resulting entity has similarities with point pair of CGA. More precisely, let $\mathbf{c}_\epsilon$ be the Euclidean midpoint between the two entities $\mathbf{x}_3$ and $\mathbf{x}_4$, $\mathbf{d}_\epsilon$ be the unit vector from $\mathbf{x}_3$ to $\mathbf{x}_4$, and r be the half of the Euclidean distance between the two points in the exactly same fashion as Hitzer et al in [54], namely:

$$2\mathrm{r} = \left|\mathbf{x}_{3\epsilon} - \mathbf{x}_{4\epsilon}\right|, \quad \mathbf{d}_\epsilon = \frac{\mathbf{x}_{3\epsilon} - \mathbf{x}_{4\epsilon}}{2\mathrm{r}}, \quad \mathbf{c}_\epsilon = \frac{\mathbf{x}_{3\epsilon} + \mathbf{x}_{4\epsilon}}{2}. \tag{8.2.5}$$

Then, (8.2.4) is rewritten by

$$\begin{aligned}\mathbf{S} =& \mathbf{x}_{C1} \wedge \mathbf{x}_{C2} \wedge \\ & 2\mathrm{r}\Big(\mathbf{d}_\epsilon \wedge \mathbf{c}_\epsilon \mathbf{I}_o^\triangleright \wedge \mathbf{I}_\infty^\triangleright + 3\mathbf{d}_\epsilon \mathbf{I}_o \wedge \mathbf{I}_\infty^\triangleright + \frac{1}{2}\big((\mathbf{c}_\epsilon^2 + \mathrm{r}^2)\mathbf{d}_\epsilon - 2\mathbf{c}_\epsilon \mathbf{c}_\epsilon \cdot \mathbf{d}_\epsilon\big)\mathbf{I}_\infty \wedge \mathbf{I}_o^\triangleright\Big)\end{aligned} \tag{8.2.6}$$

The bottom part corresponds to a point pair, as defined in [54], that belongs to the round object family. Applying the same development to the two points $\mathbf{x}_1$ and $\mathbf{x}_2$ still results in round objects:

$$\mathbf{S} = -\frac{1}{6}\Big(\left\|\mathbf{x}_{c\epsilon}\right\|^2 - \mathrm{r}^2\Big)\mathbf{I}_\epsilon \wedge \mathbf{I}_\infty \wedge \mathbf{I}_o^\triangleright + \mathbf{e}_{123} \wedge \mathbf{I}_\infty^\triangleright \wedge \mathbf{I}_o + (\mathbf{x}_{c\epsilon}\mathbf{I}_\epsilon) \wedge \mathbf{I}_\infty \wedge \mathbf{I}_o. \tag{8.2.7}$$

Note that $\mathbf{x}_{c\epsilon}$ corresponds to the center of the sphere and r to its radius. It can be further simplified into:

$$\mathbf{S} = (\mathbf{x}_C - \frac{1}{2}\mathrm{r}^2\mathbf{e}_\infty)\mathbf{I} \tag{8.2.8}$$

which is dualized to

$$\mathbf{S}^* = \mathbf{x}_C - \frac{1}{2}\mathrm{r}^2\mathbf{e}_\infty \tag{8.2.9}$$

Proposition 8.2.1. *A point* $\mathbf{x}$ *lies on the sphere* $\mathbf{s}$ *iff* $\mathbf{x} \wedge \mathbf{S} = 0$.

Proof. Since the components $\mathbf{e}_{\infty 4}, \mathbf{e}_{\infty 5}$ and $\mathbf{e}_{\infty 6}$ of $\mathbf{x}$ are removed by the outer product with $\mathbf{s}$ in (8.2.4), we ignore them to obtain

$$\begin{aligned} \mathbf{x} \wedge \mathbf{S} &= \mathbf{x} \wedge (\mathbf{S}^* \mathbf{I}) = \mathbf{x} \cdot \mathbf{S}^* \mathbf{I} \\ &= (\mathbf{x}_\epsilon + \mathbf{e}_o + \frac{1}{2}x^2\mathbf{e}_{\infty 1} + \frac{1}{2}y^2\mathbf{e}_{\infty 2} + \frac{1}{2}z^2\mathbf{e}_{\infty 3}) \cdot (\mathbf{x'}_c - \frac{1}{2}\mathrm{r}^2\mathbf{e}_\infty)\mathbf{I}, \end{aligned} \tag{8.2.10}$$

which is rewritten by

$$\begin{aligned} \mathbf{x} \wedge \mathbf{s} \quad = \quad & \Big(x\mathrm{x}_c + y\mathrm{y}_c + z\mathrm{z}_c - \left(\tfrac{1}{2}\mathrm{x}_c^2 - \tfrac{1}{6}\mathrm{r}^2\right) - \left(\tfrac{1}{2}\mathrm{y}_c^2 - \tfrac{1}{6}\mathrm{r}^2\right) - \left(\tfrac{1}{2}\mathrm{z}_c^2 - \tfrac{1}{6}\mathrm{r}^2\right) \\ & -\tfrac{1}{2}x^2 - \tfrac{1}{2}y^2 - \tfrac{1}{2}z^2 \Big) \mathbf{I} = 0 \end{aligned} \tag{8.2.11}$$

Which is equivalent to

$$\begin{aligned} & x\mathrm{x}_c + y\mathrm{y}_c + z\mathrm{z}_c - \tfrac{1}{2}\mathrm{x}_c^2 - \tfrac{1}{2}\mathrm{y}_c^2 - \tfrac{1}{2}\mathrm{z}_c^2 && = -\tfrac{1}{2}\mathrm{r}^2 \\ \Leftrightarrow \quad & -\tfrac{1}{2}(x - \mathrm{x}_c)^2 + (y - \mathrm{y}_c)^2 + (z - \mathrm{z}_c)^2 && = \tfrac{1}{2}\mathrm{r}^2 \end{aligned} \tag{8.2.12}$$

Thus resulting in the implicit equation of a sphere as:

$$(x - \mathrm{x}_c)^2 + (y - \mathrm{y}_c)^2 + (z - \mathrm{z}_c)^2 = \mathrm{r}^2 \tag{8.2.13}$$

□

8.2.1.2 Dual representation of a sphere

The dualization of the primal sphere $\mathbf{s}$ gives:

$$\mathbf{S}^* = \mathbf{x'}_c - \frac{1}{2}\mathrm{r}^2\mathbf{e}_\infty \tag{8.2.14}$$

Proposition 8.2.2. *A point* $\mathbf{x}$ *lies on the dual sphere* $\mathbf{S}^*$ *iff* $\mathbf{x} \cdot \mathbf{s}^* = 0$.

Proof. Consequence of the relations between the outer, inner and dualization as explained in Equation (8.1.10). □

Now that we studied the round objects, let us expose the definition of the plane, the line, and the sphere to show the similarities between these flat objects in CGA and their counterparts in QCGA.

8.3 Flat objects

As defined in Chapter 1, flat objects can be seen as round objects with one point sent at infinity, thus replaced by the $\mathbf{e}_\infty$. This also holds for QCGA.

Proposition 8.3.1. *To achieve this in QCGA, we use the fact that:*

$$\mathbf{e}_\infty \wedge \mathbf{I}_\infty^{\triangleright} \wedge \mathbf{I}_o^{\triangleright} = \mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright} \tag{8.3.1}$$

Proof. Using the definition (8.1.2) of $\mathbf{e}_\infty$ and the distributivity of the outer product results in:

$$\begin{aligned} \mathbf{e}_\infty \wedge \mathbf{I}_\infty^{\triangleright} \wedge \mathbf{I}_o^{\triangleright} &= \tfrac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}) \wedge \mathbf{I}_\infty^{\triangleright} \wedge \mathbf{I}_o^{\triangleright} \\ &= \tfrac{1}{3}(3\mathbf{e}_{\infty 1} \wedge \mathbf{e}_{\infty 2} \wedge \mathbf{e}_{\infty 3}) \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \wedge \mathbf{I}_o^{\triangleright} \\ &= \mathbf{e}_{\infty 1} \wedge \mathbf{e}_{\infty 2} \wedge \mathbf{e}_{\infty 3} \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \wedge \mathbf{I}_o^{\triangleright} \\ &= \mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright} \end{aligned} \tag{8.3.2}$$

□

8.3.1 Plane

8.3.1.1 Primal representation of a plane

A plane $\boldsymbol{\pi}$ in QCGA is computed using outer product from 3 linearly independent points $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ on the plane and a point at infinity as:

$$\boldsymbol{\Pi} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{e}_\infty \wedge \mathbf{I}_\infty^{\triangleright} \wedge \mathbf{I}_o^{\triangleright} \tag{8.3.3}$$

This construction will be reproduced for the other elements representing geometric objects.

The multivector $\boldsymbol{\Pi}$ corresponds to the primal form of a plane in QCGA, with grade 14 and composed by six components. The last three components have the same coefficient and thus can be combined, resulting in a form defined with four coefficients x_n, y_n, z_n and h. These coefficients have geometric meanings that are now highlighted. The development of the formula then results in:

$$\begin{aligned} \boldsymbol{\Pi} = &\left(\mathrm{x}_n\mathbf{e}_{23} - \mathrm{y}_n\mathbf{e}_{13} + \mathrm{z}_n\mathbf{e}_{12}\right)\mathbf{I}_\infty \wedge \mathbf{I}_o \\ &- \frac{\mathrm{h}}{3}\mathbf{e}_{123}\mathbf{I}_\infty \wedge \left(\mathbf{e}_{o2o3} - \mathbf{e}_{o1o3} + \mathbf{e}_{o1o2}\right) \wedge \mathbf{e}_{o4o5o6}. \end{aligned} \tag{8.3.4}$$

Proposition 8.3.2. *A point* $\mathbf{x}$ *with Euclidean coordinates* $\mathbf{x}_\epsilon$ *lies on the plane* $\boldsymbol{\Pi}$ *iff* $\mathbf{x} \wedge \boldsymbol{\Pi} = 0$.

Proof.

$$\begin{aligned}\mathbf{x} \wedge \boldsymbol{\Pi} = &\Big(x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3 + \frac{1}{2}x^2\mathbf{e}_{\infty 1} + \frac{1}{2}y^2\mathbf{e}_{\infty 2} + \frac{1}{2}z^2\mathbf{e}_{\infty 3} \\ &+ xy\mathbf{e}_{\infty 4} + xz\mathbf{e}_{\infty 5} + yz\mathbf{e}_{\infty 6} + \mathbf{e}_o\Big) \\ \wedge &\Big((\mathrm{x}_n\mathbf{e}_{23} - \mathrm{y}_n\mathbf{e}_{13} + \mathrm{z}_n\mathbf{e}_{12})\mathbf{I}_\infty \wedge \mathbf{I}_o \\ &- \frac{\mathrm{h}}{3}\mathbf{e}_{123}\mathbf{I}_\infty \wedge (\mathbf{e}_{o2o3} - \mathbf{e}_{o1o3} + \mathbf{e}_{o1o2}) \wedge \mathbf{e}_{o4o5o6}\Big).\end{aligned} \tag{8.3.5}$$

Using the distributivity and the anti-commutativity of the outer product, we obtain

$$\begin{aligned}\mathbf{x} \wedge \boldsymbol{\Pi} &= \left(x\mathrm{x}_n + y\mathrm{y}_n + z\mathrm{z}_n - \frac{1}{3}\mathrm{h}(1+1+1)\right)\mathbf{I} \\ &= \left(x\mathrm{x}_n + y\mathrm{y}_n + z\mathrm{z}_n - \mathrm{h}\right)\mathbf{I} \\ &= \left(\mathbf{x}_\epsilon \cdot \mathbf{n}_\epsilon - \mathrm{h}\right)\mathbf{I},\end{aligned} \tag{8.3.6}$$

which corresponds to the Hessian form of the plane of Euclidean normal $\mathbf{n}_\epsilon = \mathrm{x}_n\mathbf{e}_1 + \mathrm{y}_n\mathbf{e}_2 + \mathrm{z}_n\mathbf{e}_3$ with orthogonal distance h from the origin. □

8.3.1.2 Dual representation of a plane

The dualization of the primal form of the plane is

$$\boldsymbol{\Pi}^* = \mathbf{n}_\epsilon + \frac{1}{3}\mathrm{h}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}) \tag{8.3.7}$$

Proposition 8.3.3. *A point* $\mathbf{x}$ *with Euclidean coordinates* $\mathbf{x}_\epsilon$ *lies on the dual plane* $\boldsymbol{\Pi}^*$ *iff* $\mathbf{x} \cdot \boldsymbol{\Pi}^* = 0$.

Proof. Consequence of the relations between the outer, inner and dualization as explained in Equation (8.1.10). □

Because of (8.1.13), a plane can be also obtained as the bisection plane from the difference of two points $\mathbf{x}_1$ and $\mathbf{x}_2$ in a similar way as in CGA.

Proposition 8.3.4. *The dual plane*

$$\boldsymbol{\Pi}^* = \mathbf{x}_1 - \mathbf{x}_2 \tag{8.3.8}$$

is the dual orthogonal bisecting plane between the points $\mathbf{x}_1$ *and* $\mathbf{x}_2$.

Proof. From Proposition 8.3.3, a point $\mathbf{x}$ on $\boldsymbol{\Pi}^*$ satisfies $\mathbf{x} \cdot \boldsymbol{\Pi}^* = 0$.

$$\mathbf{x} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = \mathbf{x} \cdot \mathbf{x}_1 - \mathbf{x} \cdot \mathbf{x}_2 = 0 \tag{8.3.9}$$

As seen in (8.1.13), the inner product between two points results in the squared Euclidean distance between the two points. We thus have

$$\mathbf{x} \cdot (\mathbf{x}_1 - \mathbf{x}_2) = 0 \quad \Leftrightarrow \quad \|\mathbf{x}_\epsilon - \mathbf{x}_{1\epsilon}\|^2 = \|\mathbf{x}_\epsilon - \mathbf{x}_{2\epsilon}\|^2 \tag{8.3.10}$$

This corresponds to the equation of the orthogonal bisecting dual plane between $\mathbf{x}_{1\epsilon}$ and $\mathbf{x}_{2\epsilon}$. □

8.3.2 Line

Primal representation of a line

A primal line $\mathbf{L}$ is a 13-vector constructed from two linearly independent points $\mathbf{x}_1$ and $\mathbf{x}_2$ and a point at infinity in a similar way as in CGA:

$$\mathbf{L} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{e}_\infty \wedge \mathbf{I}_\infty^{\triangleright} \wedge \mathbf{I}_o^{\triangleright} \tag{8.3.11}$$

Equations (8.3.1) and (8.1.21) yields:

$$\mathbf{l} = \mathbf{x}_{C1} \wedge \mathbf{x}_{C2} \wedge \mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright} \tag{8.3.12}$$

Furthermore, the outer product between the $6-$vector $\mathbf{I}_\infty$ and the two points $\mathbf{x}_{C1}$ and $\mathbf{x}_{C2}$ remove all their $\mathbf{e}_{\infty i}$ components ($i \in \{1, \cdots, 6\}$). For the clarity purpose, (8.3.12) is simplified "in advance" as:

$$\begin{aligned}
\mathbf{L} &= \mathbf{x}_{C1} \wedge \mathbf{x}_{C2} \wedge \mathbf{I}_\infty \wedge (\mathbf{e}_{o1} - \mathbf{e}_{o2}) \wedge (\mathbf{e}_{o2} - \mathbf{e}_{o3}) \wedge \mathbf{e}_{o4o5o6} \\
&= \mathbf{x}_{C1} \wedge \Big(\mathbf{x}_{\epsilon 2} \wedge (\mathbf{e}_{o1} - \mathbf{e}_{o2}) \wedge (\mathbf{e}_{o2} - \mathbf{e}_{o3}) + 3\mathbf{e}_{o1} \wedge \mathbf{e}_{o2} \wedge \mathbf{e}_{o3}\Big) \wedge \mathbf{I}_\infty \wedge \mathbf{e}_{o4} \wedge \mathbf{e}_{o5} \wedge \mathbf{e}_{o6} \\
&= \big(3\mathbf{e}_{o1} \wedge \mathbf{e}_{o2} \wedge \mathbf{e}_{o3} \wedge (\mathbf{x}_{\epsilon 2} - \mathbf{x}_{\epsilon 1}) + \mathbf{x}_{\epsilon 1} \wedge \mathbf{x}_{\epsilon 2} \wedge (\mathbf{e}_{o1} - \mathbf{e}_{o2}) \wedge (\mathbf{e}_{o2} - \mathbf{e}_{o3})\big) \\
&\quad \wedge \mathbf{I}_\infty \wedge \mathbf{e}_{o4} \wedge \mathbf{e}_{o5} \wedge \mathbf{e}_{o6}
\end{aligned} \tag{8.3.13}$$

Letting $\mathbf{m} = \mathbf{x}_{\epsilon 2} - \mathbf{x}_{\epsilon 1}$ and $\mathbf{n} = \mathbf{x}_{\epsilon 1} \wedge \mathbf{x}_{\epsilon 2}$ gives

$$\begin{aligned}
\mathbf{L} &= \big(3\mathbf{e}_{o1} \wedge \mathbf{e}_{o2} \wedge \mathbf{e}_{o3} \wedge \mathbf{m} + \mathbf{n} \wedge (\mathbf{e}_{o1o2} - \mathbf{e}_{o1o3} + \mathbf{e}_{o2o3})\big) \wedge \mathbf{I}_\infty \wedge \mathbf{e}_{o4o5o6} \\
&= -3\,\mathbf{m}\,\mathbf{I}_\infty \wedge \mathbf{I}_o + \mathbf{n}\,\mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright}
\end{aligned} \tag{8.3.14}$$

Note that $\mathbf{m}$ and $\mathbf{n}$ correspond to the 6 Plücker coefficients of a line in $\mathbb{R}^3$. More precisely, $\mathbf{m}$ is the support vector of the line and $\mathbf{n}$ is its moment.

Proposition 8.3.5. *A point* $\mathbf{x}$ *with Euclidean coordinates* $\mathbf{x}_\epsilon$ *lies on the line* $\mathbf{L}$ *iff* $\mathbf{x} \wedge \mathbf{L} = 0$.

Proof.

$$\begin{aligned}\mathbf{x} \wedge \mathbf{L} &= \mathbf{x}_C \wedge (-3\,\mathbf{m}\,\mathbf{I}_\infty \wedge \mathbf{I}_o + \mathbf{n}\,\mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright}) \\ &= -3\mathbf{x}_\epsilon \wedge \mathbf{m}\,\mathbf{I}_\infty \wedge \mathbf{I}_o + \mathbf{x}_\epsilon \wedge \mathbf{n}\,\mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright} + \mathbf{n}\,\mathbf{I}_\infty \wedge \left((\mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3}) \wedge \mathbf{I}_o^{\triangleright}\right) \\ &= -3(\mathbf{x}_\epsilon \wedge \mathbf{m} - \mathbf{n})\,\mathbf{I}_\infty \wedge \mathbf{I}_o + \mathbf{x}_\epsilon \wedge \mathbf{n}\,\mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright} \end{aligned} \tag{8.3.15}$$

The 6-blade $\mathbf{I}_\infty \wedge \mathbf{I}_o$ and the 5-blade $\mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright}$ are linearly independent. Therefore, $\mathbf{x} \wedge \mathbf{L} = 0$ yields

$$\mathbf{x} \wedge \mathbf{L} = 0 \Leftrightarrow \begin{cases} \mathbf{x}_\epsilon \wedge \mathbf{m} = \mathbf{n} \\ \mathbf{x}_\epsilon \wedge \mathbf{n} = 0 \end{cases} \tag{8.3.16}$$

As $\mathbf{x}_\epsilon, \mathbf{m}$ and $\mathbf{n}$ are Euclidean entities, (8.3.16) corresponds to the Plücker equations of a line as mentioned in [55]. □

Dual representation of a line

Dualizing the entity $\mathbf{l}$ consists in computing with duals:

$$\begin{aligned}\mathbf{L}^* &= (-3\,\mathbf{m}\,\mathbf{I}_\infty \wedge \mathbf{I}_o + \mathbf{n}\,\mathbf{I}_\infty \wedge \mathbf{I}_o^{\triangleright})(-\mathbf{I}) \\ &= 3\,\mathbf{m}\,\mathbf{I}_\epsilon + (\mathbf{e}_{\infty 3} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1}) \wedge \mathbf{n}\,\mathbf{I}_\epsilon \end{aligned} \tag{8.3.17}$$

Proposition 8.3.6. *A point* $\mathbf{x}$ *lies on the dual line* $\mathbf{L}^*$ *iff* $\mathbf{x} \cdot \mathbf{L}^* = 0$.

Proof. Consequence of the relations between the outer, inner and dualization as explained in Equation (8.1.10). □

Note that a dual line $\mathbf{L}^*$ is also constructed by the intersection of two dual planes as follows:

$$\mathbf{L}^* = \boldsymbol{\Pi}_1^* \wedge \boldsymbol{\Pi}_2^* \tag{8.3.18}$$

8.4 Quadric surfaces

This section describes how QCGA handles the quadric surface. All the QCGA objects defined in Section 8.2 are associated with basis blades in a similar way as CGA with flat point. Going further in this direction and replacing null basis blades by points results in other geometric objects. We can then obtain general quadric surfaces.

8.4.1 Primal representation of a quadric surface

The implicit formula of a quadric surface in $\mathbb{R}^3$ is

$$F(x,y,z) = \mathrm{a}x^2 + \mathrm{b}y^2 + \mathrm{c}z^2 + \mathrm{d}xy + \mathrm{e}xz + \mathrm{f}yz + \mathrm{g}x + \mathrm{h}y + \mathrm{i}z + \mathrm{j} = 0 \tag{8.4.1}$$

A quadric surface is constructed as the outer product of points. This consists in replacing the blade $\mathbf{I}_\infty^{\triangleright}$ in Equation 8.2.3 by 5 points. In other words, it consists in computing the outer product of 9 points and filling the 6 remaining vectors with null basis vectors as follows:

$$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \cdots \wedge \mathbf{x}_9 \wedge \mathbf{I}_o^{\triangleright} \tag{8.4.2}$$

The multivector $\mathbf{q}$ corresponds to the primal form of a quadric surface with grade 14 and 12 components. Again 3 of these components have the same coefficient and can be combined into the form defined by 10 coefficients a, b, ..., j:

$$\begin{aligned}
\mathbf{Q} &= \mathbf{e}_{123}\Big(\big(2\mathrm{a}\mathbf{e}_{o1} + 2\mathrm{b}\mathbf{e}_{o2} + 2\mathrm{c}\mathbf{e}_{o3} + \mathrm{d}\mathbf{e}_{o4} + \mathrm{e}\mathbf{e}_{o5} + \mathrm{f}\mathbf{e}_{o6}\big)\cdot\mathbf{I}_\infty\Big)\wedge\mathbf{I}_o \\
&\quad + \big(\mathrm{g}\mathbf{e}_1 + \mathrm{h}\mathbf{e}_2 + \mathrm{i}\mathbf{e}_3\big)\mathbf{e}_{123}\,\mathbf{I}_\infty\wedge\mathbf{I}_o + \frac{\mathrm{j}}{3}\,\mathbf{e}_{123}\,\mathbf{I}_\infty\wedge\big((\mathbf{e}_{\infty1}+\mathbf{e}_{\infty2}+\mathbf{e}_{\infty3})\cdot\mathbf{I}_o\big) \\
&= \Big(-\big(2\mathrm{a}\mathbf{e}_{o1} + 2\mathrm{b}\mathbf{e}_{o2} + 2\mathrm{c}\mathbf{e}_{o3} + \mathrm{d}\mathbf{e}_{o4} + \mathrm{e}\mathbf{e}_{o5} + \mathrm{f}\mathbf{e}_{o6}\big) \\
&\quad + \big(\mathrm{g}\mathbf{e}_1 + \mathrm{h}\mathbf{e}_2 + \mathrm{i}\mathbf{e}_3\big) - \frac{\mathrm{j}}{3}\big(\mathbf{e}_{\infty1}+\mathbf{e}_{\infty2}+\mathbf{e}_{\infty3}\big)\Big)\,\mathbf{I} = \mathbf{q}^*\mathbf{I},
\end{aligned} \tag{8.4.3}$$

where in the second equality we used the duality property. The expression for the dual vector is therefore

$$\begin{aligned}
\mathbf{Q}^* &= -\big(2\mathrm{a}\mathbf{e}_{o1} + 2\mathrm{b}\mathbf{e}_{o2} + 2\mathrm{c}\mathbf{e}_{o3} + \mathrm{d}\mathbf{e}_{o4} + \mathrm{e}\mathbf{e}_{o5} + \mathrm{f}\mathbf{e}_{o6}\big) \\
&\quad + \big(\mathrm{g}\mathbf{e}_1 + \mathrm{h}\mathbf{e}_2 + \mathrm{i}\mathbf{e}_3\big) - \frac{\mathrm{j}}{3}\big(\mathbf{e}_{\infty1}+\mathbf{e}_{\infty2}+\mathbf{e}_{\infty3}\big).
\end{aligned} \tag{8.4.4}$$

Proposition 8.4.1. *A point* $\mathbf{x}$ *lies on the quadric surface* $\mathbf{Q}$ *iff* $\mathbf{x}\wedge\mathbf{Q} = 0$.

Proof.

$$\begin{aligned}\mathbf{x} \wedge \mathbf{Q} = \mathbf{x} \wedge (\mathbf{Q}^* \mathbf{I}) &= (\mathbf{x} \cdot \mathbf{Q}^*)\mathbf{I} \\ &= \Big(\mathbf{x} \cdot -\big(2a\mathbf{e}_{o1} + 2b\mathbf{e}_{o2} + 2c\mathbf{e}_{o3} + d\mathbf{e}_{o4} + e\mathbf{e}_{o5} + f\mathbf{e}_{o6}\big) \\ &\qquad + \big(g\mathbf{e}_1 + h\mathbf{e}_2 + i\mathbf{e}_3\big) - \frac{j}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})\Big)\mathbf{I} \\ &= \big(ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j\big)\mathbf{I}. \end{aligned} \tag{8.4.5}$$

This corresponds to the formula representing a general quadric surface. □

8.4.2 Representation of a primal axis-aligned quadric surface

Up to now, we defined general quadrics using the outer product of 9 points. For simplicity purpose, one might prefer to define axis-aligned quadrics from points. The equation of an axis-aligned quadric is as follows:

$$F(x, y, z) = ax^2 + by^2 + cz^2 + gx + hy + iz + j = 0. \tag{8.4.6}$$

On the one hand, this equation has 7 coefficients and 6 degrees of freedom. An axis-aligned quadric can then be constructed by computing the outer product of 6 points.

On the other hand, one has to remove the cross terms xy, xz, yz in the representation of points to satisfy Equation 8.4.6. To achieve this, our solution is to compute the outer product with $\mathbf{e}_{\infty 4}, \mathbf{e}_{\infty 5}, \mathbf{e}_{\infty 6}$. Indeed, one can remark that any point $\mathbf{x}$ satisfy:

$$\mathbf{x} \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} = \Big(\mathbf{e}_o + \mathbf{x}_\epsilon + \tfrac{1}{2}\big(x^2\mathbf{e}_{\infty 1} + y^2\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}\big)\Big) \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \tag{8.4.7}$$

Thus, one can consider that an axis aligned quadric is somehow a general quadric where 3 points are sent at infinity the following way (highlighted in red):

$$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{x}_6 \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \wedge \mathbf{I}_o^{\triangleright} \tag{8.4.8}$$

This multivector $\mathbf{Q}$ corresponds to the primal form of a quadric surface with grade 14, outer product of 6 points with 3 $\mathbf{e}_\infty$ basis vectors and the 5-vector $\mathbf{I}_o^{\triangleright}$ and this quadric has 9 components. For the same reason as in the construction of the general quadric, we can combine three components which have the same coefficient. Furthermore, computing the outer product with the basis vectors $\mathbf{e}_{\infty 4}, \mathbf{e}_{\infty 5}, \mathbf{e}_{\infty 6}$ removes the components $\mathbf{e}_{\infty 4}$, $\mathbf{e}_{\infty 5}$, $\mathbf{e}_{\infty 6}$ of each singular point. By

combining the outer product of such points with null basis vectors results in the form defined by the 7 coefficients a, b, c, g, h, i, j as:

$$\begin{aligned}\mathbf{Q} &= \mathbf{e}_{123}\Big((2a\mathbf{e}_{o1} + 2b\mathbf{e}_{o2} + 2c\mathbf{e}_{o3}) \cdot \mathbf{I}_\infty\Big) \wedge \mathbf{I}_o \\ &\quad + (g\mathbf{e}_1 + h\mathbf{e}_2 + i\mathbf{e}_3)\mathbf{e}_{123}\mathbf{I}_\infty \wedge \mathbf{I}_o + \tfrac{j}{3}\mathbf{I}_\infty \wedge \big((\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}) \cdot \mathbf{I}_o\big)\end{aligned} \tag{8.4.9}$$

Proposition 8.4.2. *A point* $\mathbf{x}$ *lies on a axis-aligned quadric* $\mathbf{Q}$ *iff.* $\mathbf{x} \wedge \mathbf{Q} = 0$*:*

Proof.

$$\begin{aligned}\mathbf{x} \wedge \mathbf{Q} = \mathbf{x} \wedge (\mathbf{Q}^*\mathbf{I}) &= (\mathbf{x} \cdot \mathbf{Q}^*)\mathbf{I} \\ &= \Big(\mathbf{x} \cdot \Big(- (2a\mathbf{e}_{o1} + 2b\mathbf{e}_{o2} + 2c\mathbf{e}_{o3}) \\ &\qquad + (g\mathbf{e}_1 + h\mathbf{e}_2 + i\mathbf{e}_3) - \frac{j}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})\Big)\Big)\mathbf{I} \\ &= (ax^2 + by^2 + cz^2 + gx + hy + iz + j)\mathbf{I}.\end{aligned} \tag{8.4.10}$$

This corresponds to the formula representing an axis-aligned quadric surface. □

Then, it is very easy to construct axis aligned quadrics by properly choosing the points that lie on the chosen axis-aligned quadric. The next sections present some examples of chosen axis-aligned quadric, with some chosen points that lie on these quadrics.

Representation of a primal axis-aligned paraboloid

We can construct the axis-aligned ellipsoid using six points that lie on it. For example the points:

$$\begin{aligned}&\mathbf{x}_1(0.0, 0.0, 0.0), \quad \mathbf{x}_2(-0.39, 0.1, 0.33), \quad \mathbf{x}_3(0.0, -0.41, 0.5), \\ &\mathbf{x}_4(0.0, 0.23, 0.17), \quad \mathbf{x}_5(0.47, 0.0, 0.45), \quad \mathbf{x}_6(0.29, -0.27, 0.4)\end{aligned}$$

lie on a axis-aligned paraboloid. The result of Equation 8.4.8 applied to these points is shown Figure 8.4

Representation of a primal axis-aligned hyperbolic paraboloid

Using the same equation, and replacing the points by some that lie on an axis-aligned hyperbolic paraboloid:

$$\begin{aligned}&\mathbf{x}_1(0.0, 0.0, 0.0), \quad \mathbf{x}_2(0.45, -0.01, 0.2), \quad \mathbf{x}_3(0.34, -0.37, -0.17), \\ &\mathbf{x}_4(-0.47, -0.18, 0.15), \quad \mathbf{x}_5(-0.36, 0.12, 0.1), \quad \mathbf{x}_6(0.18, 0.13, 0.0)\end{aligned}$$

results in the axis-aligned hyperbolic paraboloid shown in Figure 8.2

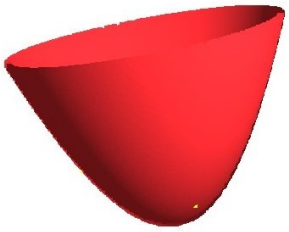


FIGURE 8.1: Result of one paraboloid from six points

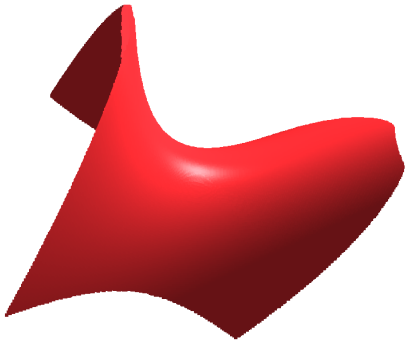


FIGURE 8.2: Result of one hyperbolic paraboloid from six points

Representation of a primal axis-aligned cylinder

An axis-aligned cylinder is an axis-aligned quadric where one of the squared components is removed with respect to the axis of the cylinder.
On one hand, this supposes that an axis-aligned cylinder can be constructed using only 5 points. On the other hand, this means that the considered component of each point taken to construct the quadric has to be removed. In a QCGA point, the squared components lie in the $\mathbf{e}_{\infty 1}$, $\mathbf{e}_{\infty 2}$, $\mathbf{e}_{\infty 3}$ components.

Thus, replacing one point of Equation (8.4.8) by the point at infinity with respect to the right axis define the wanted axis-aligned cylinder. For example, one can define an axis-aligned cylinder along the z-axis from only 5 points. From Equation (8.4.8), this means replacing a point of the above formula by a point at infinity, namely $\mathbf{e}_{\infty 3}$ for a z-axis aligned centered cylinder. For example, with the five following points:

$$\mathbf{x}_1(-0.2, 0.1, 0.3), \quad \mathbf{x}_2(0.4, 0.1, 0.2), \quad \mathbf{x}_3(0.1, 0.4, 0.1),$$

$$\mathbf{x}_4(0.1, -0.2, 0.4), \quad \mathbf{x}_5(0.1, -0.2, -0.4),$$

and the outer product between these points as:

$$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{e}_{\infty 3} \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \wedge \mathbf{I}_o^{\triangleright} \tag{8.4.11}$$

The cylinder whose axis is the (Oy) axis can be constructed in as the above equation by replacing $\mathbf{e}_{\infty 3}$ by $\mathbf{e}_{\infty 2}$ as:

$$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{e}_{\infty 2} \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \wedge \mathbf{I}_o^{\triangleright} \tag{8.4.12}$$

And finally, the cylinder whose axis is the (Ox) axis is obtained by replacing $\mathbf{e}_{\infty 3}$ by $\mathbf{e}_{\infty 1}$ as:

$$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{e}_{\infty 1} \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \wedge \mathbf{I}_o^{\triangleright} \tag{8.4.13}$$

The Figure 8.3 shows 3 cylinders, one along (Ox),another along (Oy) and the third one along (Oz).

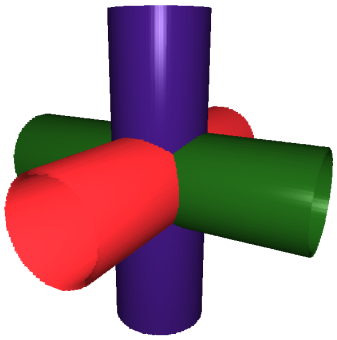


FIGURE 8.3: Construction of three cylinders along (Ox, Oy, Oz). Each cylinder is constructed from 5 points and has the same diameter

Representation of a primal axis-aligned elliptic cylinder

As 5 points are enough to uniquely define a cylinder, 5 points define also an axis-aligned elliptic cylinder whose axis is given by the null basis vector replacing the sixth point. An axis-aligned elliptic cylinder whose axis is (Oz) can be defined with the five points lying of it:

$$\mathbf{x}_1(-0.44, 0.0, 0.0), \quad \mathbf{x}_2(0.0, -0.2, 0.0), \quad \mathbf{x}_3(0.3, 0.15, 0.15),$$
$$\mathbf{x}_4(0.0, 0.2, 0.3), \quad \mathbf{x}_5(0.44, 0.0, 0.4).$$

The result is represented in Figure 8.4.

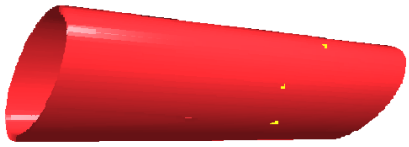


FIGURE 8.4: Construction of one elliptic cylinder from five points

Representation of a primal axis-aligned spheroid

A spheroid is characterized as an ellipsoid having two of its axes whose length is equal. Again, this property supposes that an axis-aligned spheroid can be constructed from 5 points. Furthermore, one has to constrain each point such that the squared components along the two axis has equal length. This is achieved by the outer of the points and the vector $\mathbf{e}_{\infty i} - \mathbf{e}_{\infty j}$, i and j define the two equal length axis. This 1-vector can be geometrically seen as the bisecting plane at infinity along the two considered axis, see (8.3.8), leading to some new geometric interpretations in the algebra. Thus we can construct a spheroid, having equal Ox and Oy axis, as:

$$\mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge (\mathbf{e}_{\infty 1} - \mathbf{e}_{\infty 2}) \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \wedge \mathbf{I}_o^{\triangleright} \tag{8.4.14}$$

Note that the sixth point is replaced by $(\mathbf{e}_{\infty 1} - \mathbf{e}_{\infty 2})$. As an example, we constructed the axis-aligned prolate (elongated in the z-axis) spheroid passing by the five following points that lie on the prolate spheroid:

$$\mathbf{x}_1(-0.26, 0.0, 0.0), \quad \mathbf{x}_2(0.03, 0.22, 0.24), \quad \mathbf{x}_3(-0.2, -0.1, -0.23),$$
$$\mathbf{x}_4(0.0, 0.26, 0.0), \quad \mathbf{x}_5(0.0, 0.0, 0.45)$$

The resulting surface is shown Figure 8.5.

Note that the construction of all these surfaces is available using the plugin *qc3ga.hpp* on the github repository presented last chapter.

8.4.3 Representation of a dual quadric surface

The dualization of a primal quadric surface leads to the 1-vector dual quadric surface $\mathbf{Q}^*$ of (9.4.5).

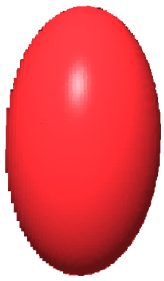


FIGURE 8.5: Construction of an axis-aligned prolate spheroid from 5 points

Proposition 8.4.3. *A point* $\mathbf{x}$ *lies on the dual quadric surface* $\mathbf{Q}^*$ *iff* $\mathbf{x} \cdot \mathbf{Q}^* = 0$.

Proof. Consequence of (8.1.10). □

The interesting thing is that this dualization enables to define axis-aligned quadric surfaces merely using their implicit equations.

8.4.4 Some examples

This subsection presents the construction of some specific quadrics.

Representation of a dual axis-aligned ellipsoid

First, an axis-aligned ellispoid can be computed as follows:

$$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{b^2}\mathbf{e}_{o2} - \frac{2}{c^2}\mathbf{e}_{o3} - \frac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}) \tag{8.4.15}$$

where a, b, c are the semi axis parameters of the ellipsoid. This construction is illustrated in Figure 8.6.

FIGURE 8.6: Construction of an axis-aligned ellipsoid.

Representation of a dual axis-aligned cylinder

Another example of quadric is a cylinder. Axis-aligned cylinder are easily defined. A cylinder whose axis is (Oz) and whose radius is r is defined as follows:

$$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{b^2}\mathbf{e}_{o2} - \frac{r^2}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}) \tag{8.4.16}$$

Note that non-axis aligned can be constructed as the outer product of 9 points as shown in figure 8.7.

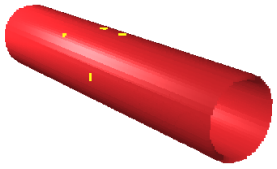


FIGURE 8.7: Construction of a generalized cylinder from nine points.

Representation of a dual axis-aligned hyperbolic paraboloid

Another example of axis-aligned quadric is the hyperbolic paraboloid, also called saddle. It can be defined as:

$$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} + \frac{2}{b^2}\mathbf{e}_{o2} - \mathbf{e}_3 \tag{8.4.17}$$

An axis-aligned cone can be computed with qcga as follows:

$$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{b^2}\mathbf{e}_{o2} + \frac{2}{c^2}\mathbf{e}_{o3} \tag{8.4.18}$$

Representation of a dual axis-aligned hyperboloid

An axis-aligned hyperboloid of one sheet can be constructed as follows:

$$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{a^2}\mathbf{e}_{o2} + \frac{2}{c^2}\mathbf{e}_{o3} + \frac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}) \tag{8.4.19}$$

An example of such quadric is shown in Figure 8.8 The definition of an axis-aligned hyperboloid of two sheets is the following:

$$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{a^2}\mathbf{e}_{o2} + \frac{2}{c^2}\mathbf{e}_{o2} - \frac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}) \tag{8.4.20}$$

FIGURE 8.8: Construction of a dual hyperboloid of one sheet.

Representation of a dual axis-aligned elliptic paraboloid

An axis-aligned elliptic paraboloid can be defined as:

$$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{b^2}\mathbf{e}_{o2} + \frac{2}{c}\mathbf{e}_{o3} \tag{8.4.21}$$

Representation of a dual axis-aligned degenerate quadrics

As previously seen, degenerate quadrics can also be defined. For example, a pair of planes can be defined as:

$$\mathbf{Q}^* = -2\mathbf{e}_{o1} + 2\mathbf{e}_{o2} \tag{8.4.22}$$

An illustration of pair of planes using QCGA is shown on Figure 8.9 The tables 8.2 and 8.3

FIGURE 8.9: Construction of a axis-aligned pair of planes.

summarize the dual definition of CGA objects, as well as axis-aligned quadrics and degenerate quadrics.

The table 8.4 details the main class of objects that can be handled using QCGA.

TABLE 8.2: Definition of dual CGA objects using QCGA.

Geometric objects	Dual definition
Sphere	$\mathbf{S}^* = \mathbf{x'}_c - \frac{1}{6}\mathrm{r}^2(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})$
Plane	$\boldsymbol{\Pi}^* = \mathbf{n}_\epsilon + \frac{1}{3}\mathrm{h}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})\mathbf{n}_\epsilon + \frac{1}{3}\mathrm{h}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})$
Line	$\mathbf{L}^* = 3\,\mathbf{m}\,\mathbf{I}_\epsilon + (\mathbf{e}_{\infty 3} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1}) \wedge \mathbf{n}\,\mathbf{I}_\epsilon$ $\mathbf{L}^* = \boldsymbol{\Pi}_1^* \wedge \boldsymbol{\Pi}_2^*$
Circle	$\mathbf{C}^* = \mathbf{S}_1^* \wedge \mathbf{S}_2^*$ $\mathbf{C}^* = \mathbf{S}^* \wedge \boldsymbol{\Pi}$
Point pair	$\mathbf{P}_p^* = \mathbf{S}^* \wedge \mathbf{L}^*$ $\mathbf{P}_p^* = \mathbf{C}_1^* \wedge \mathbf{C}_2^*$

TABLE 8.3: Definition of dual **axis-aligned** quadric surfaces using QCGA.

Geometric objects	Dual definition
Ellipsoids	$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} + \frac{2}{b^2}\mathbf{e}_{o2} - \frac{2}{c^2}\mathbf{e}_{o3} - \frac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})$
Cones	$\mathbf{Q}^* = -2a\mathbf{e}_{o1} - 2b\mathbf{e}_{o2} + 2c\mathrm{k}\mathbf{e}_{o3},\ \mathrm{k} \in \mathbb{R}$
Paraboloids	$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} + \frac{2}{b^2}\mathbf{e}_{o2} - \frac{2}{c^2}\mathbf{e}_{o3} - \frac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})$
Cylinders	$\mathbf{Q}^* = -2\mathbf{e}_{o1} - 2\mathbf{e}_{o2} - \frac{\mathrm{r}^2}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})$
Elliptic paraboloids	$\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{b^2}\mathbf{e}_{o2} + \frac{2}{c}\mathbf{e}_{o3}$
Hyperboloids one sheet two sheets	 $\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{a^2}\mathbf{e}_{o2} + \frac{2}{c^2}\mathbf{e}_{o2} + \frac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})$ $\mathbf{Q}^* = -\frac{2}{a^2}\mathbf{e}_{o1} - \frac{2}{a^2}\mathbf{e}_{o2} + \frac{2}{c^2}\mathbf{e}_{o2} - \frac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})$
Parallel planes	$\mathbf{Q}^* = -2\mathbf{e}_{o1} + 2\mathbf{e}_{o2}$

TABLE 8.4: Definition of primal geometric objects using QCGA.

Round objects (Sphere,circle)	$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \cdots \wedge \mathbf{I}_\infty^\triangleright \wedge \mathbf{I}_o^\triangleright$
Flat objects (plane,line,$\cdots$)	$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \cdots \wedge \mathbf{e}_\infty \wedge \mathbf{I}_\infty^\triangleright \wedge \mathbf{I}_o^\triangleright$
Axis aligned quadrics	$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{x}_6 \wedge \mathbf{e}_{\infty 4} \wedge \mathbf{e}_{\infty 5} \wedge \mathbf{e}_{\infty 6} \wedge \mathbf{I}_o^\triangleright$
General quadrics	$\mathbf{q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{x}_6 \wedge \mathbf{x}_7 \wedge \mathbf{x}_8 \wedge \mathbf{x}_9 \wedge \mathbf{I}_o^\triangleright$

Table 8.5 summarizes some primal definitions of QCGA axis-aligned objects with points.

TABLE 8.5: Definition of some primal **axis-aligned** quadric surfaces using QCGA.

Ellipsoids Paraboloids Hyperbolic paraboloids	$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{x}_6 \wedge \mathbf{e}_{\infty 4 \infty 5 \infty 6} \wedge \mathbf{I}_o^{\triangleright}$
Spheroids	$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge (\mathbf{e}_{\infty 1} - \mathbf{e}_{\infty 2}) \wedge \mathbf{e}_{\infty 4 \infty 5 \infty 6} \wedge \mathbf{I}_o^{\triangleright}$
Cylinders Elliptic cylinders	$\mathbf{Q} = \mathbf{x}_1 \wedge \mathbf{x}_2 \wedge \mathbf{x}_3 \wedge \mathbf{x}_4 \wedge \mathbf{x}_5 \wedge \mathbf{e}_{\infty 3} \wedge \mathbf{e}_{\infty 4 \infty 5 \infty 6} \wedge \mathbf{I}_o^{\triangleright}$

8.5 Normals and tangents

This section presents the computation of the normal Euclidean vector $\mathbf{n}_e$ and the tangent plane $\boldsymbol{\Pi}^*$ of a point $\mathbf{x}$ (associated to the Euclidean point $\mathbf{x}_e = x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3$) on a dual quadric surface $\mathbf{Q}^*$. The implicit formula of the dual quadric surface is considered as the following scalar field:

$$F(x,y,z) = \mathbf{x} \cdot \mathbf{Q}^*. \tag{8.5.1}$$

The normal vector $\mathbf{n}_e$ of a point $\mathbf{x}$ is computed as the gradient of the implicit surface (scalar field) at $\mathbf{x}$:

$$\mathbf{n}_e = \nabla F(x,y,z) = \frac{\partial F(x,y,z)}{\partial x}\mathbf{e}_1 + \frac{\partial F(x,y,z)}{\partial y}\mathbf{e}_2 + \frac{\partial F(x,y,z)}{\partial z}\mathbf{e}_3. \tag{8.5.2}$$

Since the partial derivative with respect to the x component is defined by

$$\frac{\partial F(x,y,z)}{\partial x} = \lim_{h \mapsto 0} \frac{F(x+h,y,z) - F(x,y,z)}{h}, \tag{8.5.3}$$

we have

$$\frac{\partial F(x,y,z)}{\partial x} = \lim_{h \mapsto 0} \frac{\mathbf{x}_h \cdot \mathbf{Q}^* - \mathbf{x} \cdot \mathbf{Q}^*}{h} = \left(\lim_{h \mapsto 0} \frac{\mathbf{x}_2 - \mathbf{x}}{h} \right) \cdot \mathbf{Q}^*, \tag{8.5.4}$$

where $\mathbf{x}_h$ is the point obtained by translating $\mathbf{x}$ along the x-axis by the value h. Note that $\mathbf{x}_h - \mathbf{x}$ represents the dual orthogonal bisecting plane spanned by $\mathbf{x}_h$ and $\mathbf{x}$ (see Proposition 8.3.4). The computation of $\mathbf{x}_h - \mathbf{x}$ results in:

$$\begin{aligned} \mathbf{x}_h - \mathbf{x} &= \Big(\mathbf{e}_o + \tfrac{1}{2}\big((x+h)^2\mathbf{e}_{\infty 1} + y^2\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}\big) + (x+h)y\mathbf{e}_{\infty 4} + (x+h)z\mathbf{e}_{\infty 5} \\ &\quad + yz\mathbf{e}_{\infty 6} + (x+h)\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3\Big) - \Big(\mathbf{e}_o + \tfrac{1}{2}\big(x^2\mathbf{e}_{\infty 1} + y^2\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}\big) + xy\mathbf{e}_{\infty 4} \\ &\quad + xz\mathbf{e}_{\infty 5} + yz\mathbf{e}_{\infty 6} + x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3\Big) \\ &= \tfrac{1}{2}\big(2xh + h^2\big)\mathbf{e}_{\infty 1} + hy\mathbf{e}_{\infty 4} + hz\mathbf{e}_{\infty 5} + h\mathbf{e}_1 \end{aligned} \tag{8.5.5}$$

Dividing by h yields:

$$\mathbf{x}_h - \mathbf{x} \quad = \quad \tfrac{1}{2}\big(2x + h\big)\mathbf{e}_{\infty 1} + y\mathbf{e}_{\infty 4} + z\mathbf{e}_{\infty 5} + \mathbf{e}_1 \tag{8.5.6}$$

Finally, the limit results in:

$$\begin{aligned} \lim_{h \mapsto 0} \frac{\mathbf{x}_h - \mathbf{x}}{h} &= \lim_{h \mapsto 0} \frac{\frac{1}{2}\big(2x + h\big)\mathbf{e}_{\infty 1} + y\mathbf{e}_{\infty 4} + z\mathbf{e}_{\infty 5} + \mathbf{e}_1}{h} \\ &= \tfrac{1}{2}\big(2x\big)\mathbf{e}_{\infty 1} + y\mathbf{e}_{\infty 4} + z\mathbf{e}_{\infty 5} + \mathbf{e}_1 \\ &= x\mathbf{e}_{\infty 1} + y\mathbf{e}_{\infty 4} + z\mathbf{e}_{\infty 5} + \mathbf{e}_1 \end{aligned} \tag{8.5.7}$$

Thus, we have

$$\begin{aligned} \lim_{h \mapsto 0} \frac{\mathbf{x}_h - \mathbf{x}}{h} &= x\mathbf{e}_{\infty 1} + y\mathbf{e}_{\infty 4} + z\mathbf{e}_{\infty 5} + \mathbf{e}_1 \\ &= (\mathbf{x} \cdot \mathbf{e}_1)\mathbf{e}_{\infty 1} + (\mathbf{x} \cdot \mathbf{e}_2)\mathbf{e}_{\infty 4} + (\mathbf{x} \cdot \mathbf{e}_3)\mathbf{e}_{\infty 5} + \mathbf{e}_1. \end{aligned} \tag{8.5.8}$$

This argument can be applied to the partial derivative with respect to the y and z components. Therefore, we obtain:

$$\begin{aligned} \mathbf{n}_\epsilon = &\Big(\big((\mathbf{x} \cdot \mathbf{e}_1)\mathbf{e}_{\infty 1} + (\mathbf{x} \cdot \mathbf{e}_2)\mathbf{e}_{\infty 4} + (\mathbf{x} \cdot \mathbf{e}_3)\mathbf{e}_{\infty 5} + \mathbf{e}_1\big) \cdot \mathbf{Q}^*\Big)\mathbf{e}_1 + \\ &\Big(\big((\mathbf{x} \cdot \mathbf{e}_2)\mathbf{e}_{\infty 2} + (\mathbf{x} \cdot \mathbf{e}_1)\mathbf{e}_{\infty 4} + (\mathbf{x} \cdot \mathbf{e}_3)\mathbf{e}_{\infty 6} + \mathbf{e}_2\big) \cdot \mathbf{q}^*\Big)\mathbf{e}_2 + \\ &\Big(\big((\mathbf{x} \cdot \mathbf{e}_3)\mathbf{e}_{\infty 3} + (\mathbf{x} \cdot \mathbf{e}_1)\mathbf{e}_{\infty 5} + (\mathbf{x} \cdot \mathbf{e}_2)\mathbf{e}_{\infty 6} + \mathbf{e}_3\big) \cdot \mathbf{Q}^*\Big)\mathbf{e}_3. \end{aligned} \tag{8.5.9}$$

On the other hand, the tangent plane at a surface point $\mathbf{x}$ can be computed from the Euclidean normal vector $\mathbf{n}_\epsilon$ and the point $\mathbf{x}$. Since the plane orthogonal distance from the origin is h, the tangent plane $\boldsymbol{\pi}^*$ is obtained by

$$\boldsymbol{\Pi}^* = \mathbf{n}_\epsilon + \frac{1}{3}\big(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}\big)h. \tag{8.5.10}$$

Note that h can be very easily be computed with the following formula:

$$h = \mathbf{x}_\epsilon \cdot \mathbf{n}_\epsilon. \tag{8.5.11}$$

Finally, we get the full definition of the tangent plane as follows:

$$\boldsymbol{\Pi}^* = \mathbf{n}_\epsilon + \frac{1}{3}\big(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3}\big)(\mathbf{x}_\epsilon \cdot \mathbf{n}_\epsilon). \tag{8.5.12}$$

8.6 Intersections

Let us consider two geometric objects corresponding to dual quadrics* $\mathbf{a}^*$ and $\mathbf{b}^*$. Assuming that the two objects are linearly independent, i.e., $\mathbf{A}^*$ and $\mathbf{B}^*$ are linearly independent, we consider the outer product $\mathbf{C}^*$ of these two objects:

$$\mathbf{C}^* = \mathbf{a}^* \wedge \mathbf{b}^*. \tag{8.6.1}$$

If a point $\mathbf{x}$ lies on $\mathbf{C}^*$, then

$$\mathbf{x} \cdot \mathbf{C}^* = \mathbf{x} \cdot (\mathbf{A}^* \wedge \mathbf{B}^*) = 0. \tag{8.6.2}$$

The inner product definition develops (8.6.2) as follows:

$$\mathbf{x} \cdot \mathbf{C}^* = (\mathbf{x} \cdot \mathbf{A}^*)\mathbf{B}^* - (\mathbf{x} \cdot \mathbf{B}^*)\mathbf{A}^* = 0. \tag{8.6.3}$$

Our assumption of linear independence between $\mathbf{a}^*$ and $\mathbf{b}^*$ indicates that (8.6.3) holds if and only if $\mathbf{x} \cdot \mathbf{A}^* = 0$ and $\mathbf{x} \cdot \mathbf{B}^* = 0$, i.e. the point $\mathbf{x}$ lies on both quadrics. Thus, $\mathbf{C}^* = \mathbf{A}^* \wedge \mathbf{B}^*$ represents the intersection of the linearly independent quadrics $\mathbf{A}^*$ and $\mathbf{B}^*$, and a point $\mathbf{x}$ lies on this intersection if and only if $\mathbf{x} \cdot \mathbf{C}^* = 0$.

8.6.1 Quadric-Line intersection

In computer graphics, making a Geometric Algebra compatible with a raytracer requires only to be able to compute a surface normal and a line to object intersection. This section defines the line to quadric intersection.

Similarly to (8.6.1), the intersection $\mathbf{x}_\pm$ between a dual line $\mathbf{L}^*$ and a dual quadric $\mathbf{Q}^*$ is computed by $\mathbf{L}^* \wedge \mathbf{Q}^*$. Any point $\mathbf{x}$ lying on the line $\mathbf{L}$ defined by two points $\mathbf{x}_1$ and $\mathbf{x}_2$ can be represented by the parametric formula $\mathbf{x}_\epsilon = \alpha(\mathbf{x}_{1\epsilon} - \mathbf{x}_{2\epsilon}) + \mathbf{x}_{2\epsilon} = \alpha\mathbf{m}_\epsilon + \mathbf{x}_{2\epsilon}$ (see Section 8.2). Note that $\mathbf{m}_\epsilon$ is computed using only the inner product between the dual line $\mathbf{L}^*$ and the null basis vectors. Any point $\mathbf{x}_2 \in \mathbf{L}$ can be used, especially the closest point of $\mathbf{L}$ from the origin, i.e. $\mathbf{x}_{2\epsilon} = (\mathbf{n}_\epsilon \times \mathbf{m}_\epsilon)/(\mathbf{m}_\epsilon \cdot \mathbf{m}_\epsilon)$. Accordingly, computing the intersection between the dual line $\mathbf{L}^*$ and the dual quadric $\mathbf{Q}^*$ becomes equivalent to finding α such that $\mathbf{x}$ lies on the dual quadric, i.e., $\mathbf{x} \cdot \mathbf{Q}^* = 0$, leading to a second degree equation in α, as shown in (8.4.1). In this situation, the problem is reduced to computing the roots of this equation. However, we have to consider four cases: the case where the line is tangent to the quadric, the case where the intersection is empty, the case where the line intersects the quadric into two points, and the case where one of the two

*The term "quadric" (without being followed by surface) encompasses quadric surfaces and conic sections.

points exists at infinity. To identify each case, we use the discriminant δ defined as:

$$\delta = \beta^2 - 4(\mathbf{x} \cdot \mathbf{Q}^*) \sum_{i=1}^{6} (\mathbf{m} \cdot \mathbf{e}_{oi})(\mathbf{Q}^* \cdot \mathbf{e}_{\infty i}), \tag{8.6.4}$$

where

$$\begin{aligned}\beta =& 2\mathbf{m} \cdot (\mathrm{a}(x_{2\epsilon} \cdot \mathbf{e}_1)\mathbf{e}_1 + \mathrm{b}(x_{2\epsilon} \cdot \mathbf{e}_2)\mathbf{e}_2 + \mathrm{c}(x_{2\epsilon} \cdot \mathbf{e}_3)\mathbf{e}_3) + \\ & \mathrm{d}\big((\mathbf{m} \wedge \mathbf{e}_1) \cdot (\mathbf{x}_{2\epsilon} \wedge \mathbf{e}_2) + (\mathbf{x}_{2\epsilon} \wedge \mathbf{e}_1) \cdot (\mathbf{m} \wedge \mathbf{e}_2)\big) + \\ & \mathrm{e}\big((\mathbf{m} \wedge \mathbf{e}_1) \cdot (\mathbf{x}_{2\epsilon} \wedge \mathbf{e}_3) + (\mathbf{x}_{2\epsilon} \wedge \mathbf{e}_1) \cdot (\mathbf{m} \wedge \mathbf{e}_3)\big) + \\ & \mathrm{f}\big((\mathbf{m} \wedge \mathbf{e}_2) \cdot (\mathbf{x}_{2\epsilon} \wedge \mathbf{e}_3) + (\mathbf{x}_{2\epsilon} \wedge \mathbf{e}_2) \cdot (\mathbf{m} \wedge \mathbf{e}_3)\big) + \mathbf{Q}^* \cdot \mathbf{m}_\epsilon.\end{aligned} \tag{8.6.5}$$

If $\delta < 0$, the line does not intersect the quadric (the solutions are complex). If $\delta = 0$, the line and the quadric are tangent. If $\delta > 0$ and $\sum_{i=1}^{6}(\mathbf{m} \cdot \mathbf{e}_{oi})(\mathbf{q}^* \cdot \mathbf{e}_{\infty i}) = 0$, we have only one intersection point (linear equation). Otherwise, we have two different intersection points $\mathbf{x}_\pm$ computed by

$$\mathbf{x}_\pm = \mathbf{m}(-\mathrm{b} \pm \sqrt{\delta}) / \Big(2 \sum_{i=1}^{6} (\mathbf{m} \cdot \mathbf{e}_{oi})(\mathbf{Q}^* \cdot \mathbf{e}_{\infty i})\Big) + \mathbf{x}_{2\epsilon}. \tag{8.6.6}$$

8.7 Transformations

8.7.1 Translation

We consider $3D$ projective transformations acting on points and quadrics. We first focus on translation of points $\mathbf{x}$ and quadrics $\mathbf{Q}^*$ over :

$$\mathbf{t} = \tau \mathbf{e}_1, \tau \in \mathbb{R} \tag{8.7.1}$$

The versor $\mathbf{V}_\tau$ for the translation $\mathbf{t}$ is:

$$V_\tau = \Big(1 - \frac{1}{2}\tau \mathbf{e}_3 \wedge \mathbf{e}_{\infty 5}\Big)\Big(1 - \frac{1}{2}\tau \mathbf{e}_2 \wedge \mathbf{e}_{\infty 4}\Big)\Big(1 - \frac{1}{2}\tau \mathbf{e}_1 \wedge \mathbf{e}_{\infty 1}\Big) \tag{8.7.2}$$

As a geometric product of versor, this entity is also a versor and its inverse is:

$$V\tau^{-1} = \Big(1 + \frac{1}{2}\tau \mathbf{e}_1 \wedge \mathbf{e}_{\infty 1}\Big)\Big(1 + \frac{1}{2}\tau \mathbf{e}_2 \wedge \mathbf{e}_{\infty 4}\Big)\Big(1 + \frac{1}{2}\tau \mathbf{e}_3 \wedge \mathbf{e}_{\infty 5}\Big) \tag{8.7.3}$$

Proposition 8.7.1. *The translation of* $\mathbf{x}$ *is given as follows:*

$$\mathbf{x}_\tau = V_\tau \mathbf{x} V_\tau^{-1} \tag{8.7.4}$$

This rotor has to translate the point at the origin defined as:

$$\mathbf{o} = \mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3} \tag{8.7.5}$$

to the point:

$$\mathbf{o}_\tau = \mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3} + \frac{1}{2}\tau^2\mathbf{e}_{\infty 1} + \tau\mathbf{e}_1 \tag{8.7.6}$$

Using the distributivity of the geometric product results in:

$$\begin{aligned} \mathbf{o}_\tau &= V_\tau \mathbf{o} V_\tau^{-1} \\ &= V_\tau \mathbf{e}_{o1} V_\tau^{-1} + V_\tau \mathbf{e}_{o2} \mathbf{V}_\tau^{-1} + V_\tau \mathbf{e}_{o3} V_\tau^{-1} \end{aligned} \tag{8.7.7}$$

To develop this formula, we use the following property:

$$\begin{aligned} & \left(1 - \tfrac{1}{2}\tau\mathbf{e}_j \wedge \mathbf{e}_{\infty k}\right)\mathbf{e}_{oi}\left(1 + \tfrac{1}{2}\tau\mathbf{e}_j \wedge \mathbf{e}_{\infty k}\right) \\ = \;& \mathbf{e}_{oi} + \tfrac{1}{2}\tau\left(\mathbf{e}_{oi}\mathbf{e}_j \wedge \mathbf{e}_{\infty k} - \mathbf{e}_j \wedge \mathbf{e}_{\infty k}\mathbf{e}_{oi}\right) - \tfrac{1}{4}\tau^2\left(\mathbf{e}_j \wedge \mathbf{e}_{\infty k}\mathbf{e}_{oi}\mathbf{e}_j \wedge \mathbf{e}_{\infty k}\right) \\ = \;& \begin{cases} \mathbf{e}_{oi} & \text{If } i \neq k, \forall j \in \{1,2,3\}\,, \\ \mathbf{e}_{oi} + \tau\mathbf{e}_j + \frac{1}{2}\tau^2\mathbf{e}_{\infty k} & \text{If } i = k, \forall j \in \{1,2,3\}\,. \end{cases} \end{aligned} \tag{8.7.8}$$

along with the following more general property:

$$\begin{aligned} & \left(1 - \tfrac{1}{2}\tau\mathbf{e}_j \wedge \mathbf{e}_{\infty k}\right)\mathbf{e}_{u}\left(1 + \tfrac{1}{2}\tau\mathbf{e}_j \wedge \mathbf{e}_{\infty k}\right) \\ = \;& \mathbf{e}_{u} + \tfrac{1}{2}\tau\left(\mathbf{e}_{u}\mathbf{e}_j \wedge \mathbf{e}_{\infty k} - \mathbf{e}_j \wedge \mathbf{e}_{\infty k}\mathbf{e}_{u}\right) - \tfrac{1}{4}\tau^2\left(\mathbf{e}_j \wedge \mathbf{e}_{\infty k}\mathbf{e}_{u}\mathbf{e}_j \wedge \mathbf{e}_{\infty k}\right) \\ = \;& \begin{cases} \mathbf{e}_{u} & \text{If } \mathbf{e}_u \cdot \mathbf{e}_j = 0 \text{ and } \mathbf{e}_u \cdot \mathbf{e}_{\infty k} = 0, \\ \mathbf{e}_{u} + \tau\mathbf{e}_{\infty k} & \text{If } \mathbf{e}_u \cdot \mathbf{e}_j \neq 0, \\ \mathbf{e}_{u} + \tau\mathbf{e}_j + \frac{1}{2}\tau^2\mathbf{e}_{\infty k} & \text{If } \mathbf{e}_u \cdot \mathbf{e}_{\infty k} \neq 0. \end{cases} \end{aligned} \tag{8.7.9}$$

Applying 8.7.9 in 8.7.7 results in:

$$\begin{aligned} \mathbf{o}_\tau &= V_\tau \mathbf{e}_{o1} V_\tau^{-1} + \mathbf{e}_{o2} + \mathbf{e}_{o3} \\ &= \mathbf{e}_{o1} + \tau\mathbf{e}_1 + \tfrac{1}{2}\tau^2\mathbf{e}_{\infty 1} + \mathbf{e}_{o2} + \mathbf{e}_{o3} \end{aligned} \tag{8.7.10}$$

Thus corresponding to the point at the origin translated by the vector $\mathbf{t} = \tau\mathbf{e}_1$.

Now for general point:

$$\mathbf{x} = \mathbf{x}_\epsilon + \tfrac{1}{2}(x^2\mathbf{e}_{\infty 1} + y^2\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}) + xy\mathbf{e}_{\infty 4} + xz\mathbf{e}_{\infty 5} + yz\mathbf{e}_{\infty 6} + \mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3}. \tag{8.7.11}$$

We prove that the versor V_τ translates $\mathbf{x}$ with a vector $\mathbf{t} = \tau\mathbf{e}_1$. Once more, we use the distributivity of the geometric product over the addition resulting in:

$$\begin{aligned}\mathbf{x}_\tau =& V_\tau \mathbf{x}_\epsilon V_\tau^{-1} + \tfrac{1}{2}(x^2 V_\tau \mathbf{e}_{\infty 1} V_\tau^{-1} + y^2 V_\tau \mathbf{e}_{\infty 2} V_\tau^{-1} + z^2 V_\tau \mathbf{e}_{\infty 3} V_\tau^{-1}) + xy V_\tau \mathbf{e}_{\infty 4} V_\tau^{-1} \\ &+ xz V_\tau \mathbf{e}_{\infty 5} V_\tau^{-1} + yz V_\tau \mathbf{e}_{\infty 6} V_\tau^{-1} + V_\tau(\mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3}) V_\tau^{-1}\end{aligned} \tag{8.7.12}$$

Using 8.7.9, the product of the Euclidean part of $\mathbf{x}$ with the versor results in:

$$V_\tau \mathbf{x}_\epsilon V_\tau^{-1} \quad = (\mathbf{x}_\epsilon + x\tau\mathbf{e}_{\infty 1} + y\tau\mathbf{e}_{\infty 4} + z\tau\mathbf{e}_{\infty 5}) \tag{8.7.13}$$

In a similar way, 8.7.9 yields:

$$V_\tau(\mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3}) V_\tau^{-1} = \mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3} + \tau\mathbf{e}_1 + \tfrac{1}{2}\tau^2\mathbf{e}_{\infty 1} \tag{8.7.14}$$

Still using 8.7.9, the versor product $V_\tau \mathbf{e}_{\infty i} V_\tau^{-1} = \mathbf{e}_{\infty i}$.

Finally, the transformed point is given as:

$$\begin{aligned}\mathbf{x}_\tau =& \mathbf{x}_\epsilon + x\tau\mathbf{e}_{\infty 1} + y\tau\mathbf{e}_{\infty 4} + z\tau\mathbf{e}_{\infty 5} + \tfrac{1}{2}(x^2\mathbf{e}_{\infty 1} + y^2\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}) + xy\mathbf{e}_{\infty 4} \\ &+ xz\mathbf{e}_{\infty 5} + yz\mathbf{e}_{\infty 6} + \mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3} + \tau\mathbf{e}_1 + \tfrac{1}{2}\tau^2\mathbf{e}_{\infty 1} \\ =& (x+\tau)\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3 + \tfrac{1}{2}\Big((x^2 + 2x\tau + \tau^2)\mathbf{e}_{\infty 1} + y^2\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}\Big) \\ &+ (xy + y\tau)\mathbf{e}_{\infty 4} + (xz + z\tau)\mathbf{e}_{\infty 5} + yz\mathbf{e}_{\infty 6} + \mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3} \\ =& (x+\tau)\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3 + \tfrac{1}{2}\Big((x+\tau)^2\mathbf{e}_{\infty 1} + y^2\mathbf{e}_{\infty 2} + z^2\mathbf{e}_{\infty 3}\Big) \\ &+ (x+\tau)y\mathbf{e}_{\infty 4} + (x+\tau)z\mathbf{e}_{\infty 5} + yz\mathbf{e}_{\infty 6} + \mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3}\end{aligned} \tag{8.7.15}$$

This latter entity is the point $\mathbf{x}$ translated by the vector $\tau\mathbf{e}_1$.

Note that this can be reproduced for the translation vector $\mathbf{t} = \tau\mathbf{e}_2$ and $\mathbf{t} = \tau\mathbf{e}_3$ resulting in translation over any axis.

8.7.2 Other transformations

The translation is found however, we still have to find versors for the remaining transformations.

If we consider the situation where we want to handle transformations of QCGA lines, spheres, circles, planes, then a solution is merely to use CGA versors. Indeed, due to the property of rotors and the fact that a CGA point is embedded in QCGA then standard CGA rotors can transform all embedded round and flat objects of QCGA.

To find versors that transform non-CGA entities, we want to find the reflections at non-parallel planes and over concentric spheres. To achieve this, we would investigate in more details the versors that involves terms similar to Equation (8.1.5). More generally, we would also study the possible versors that could be computed using QCGA. We aim at finding also projective transformations.

8.8 Discussion

8.8.1 Limitations

The construction of quadric surfaces by the outer product of points presented in Sections 8.2 and 8.4 is a distinguished property of QCGA that is missing in DPGA and DCGA. However, QCGA also faces some limitations. Some properties possessed by different frameworks for Geometric Algebra are summarized in Table 8.6.

TABLE 8.6: Comparison of properties between DPGA, DCGA, and QCGA. The symbol • stands for "capable", ○ for "incapable" and ⊘ for "unknown".

Framework	DPGA [26]	DCGA [27]	**QCGA**
dimensions	8	10	15
construction from outer product of points	○	○	•
quadrics intersection	○	○	•
quadric plane intersection	•	•	•
versors	•	•	⊘
Darboux cyclides	○	•	○

First, we have not yet proved whether objects in QCGA can be transformed using versors. We computed the versor of the translation however not yet the versor of the rotation nor for non-isotropic scale. In contrast, DPGA and DCGA are known to be capable of transforming objects.

Second, the 15 dimensions of the QCGA vector space have $2^{15} (\simeq 32,000)$ elements for a full multivector. Though it is elegant to construct quadrics as the outer product of 9 nine points, some components are likely to be multiplied at the power of 9 thus this requires some numerical care in computation.

8.8.2 Implementations

There exist many implementations of Geometric Algebra; however, very few can handle dimensions higher than 8 or 10. This is because higher dimensions bring a large number of elements of the multivector used, resulting in expensive computation; in many cases, the computation becomes impossible in practice. QCGA has 15 dimensions and hence requires some specific care during the computation.

We conducted our tests with *Garamon* which was presented in the Section 2.1.3. We remark that most of the products involved in our tests were the outer products between 14-vectors and 1-vectors, revealing one of the less time-consuming products of QCGA. Indeed, QCGA with dimension of 15 has 2^{15} elements and this number is 1,000 times as large as that of elements for CGA with dimension of 5 (CGA with dimension of 5 is needed for the equivalent operations with QCGA with dimension of 15). The computational time required for QCGA, however, did not achieve 1,000 times but only 70 times of that for CGA. This means that the computation of QCGA runs in reasonable time on the enhanced version of Breuils et al. [5, 6]. Figure 8.10 depicts a few examples generated with our OpenGL renderer based on the outer product null-space voxels and our ray-tracer.

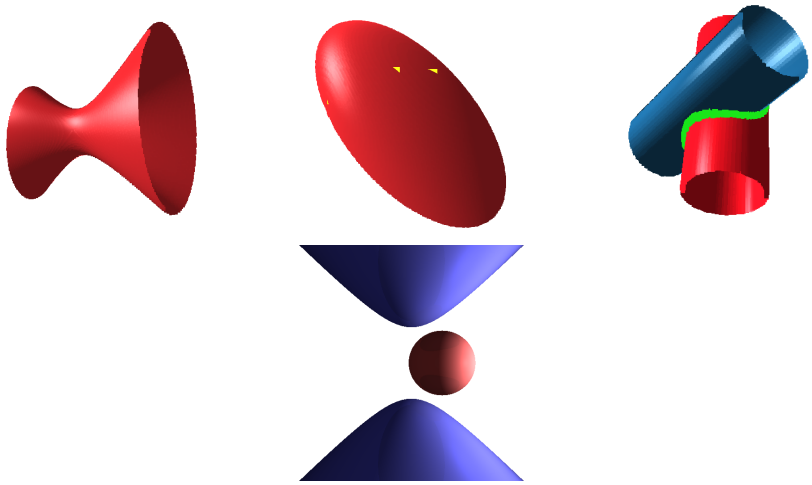


FIGURE 8.10: Example of our construction of QCGA objects. From left to right: a dual hyperboloid built from its equation, an ellipsoid built from its control points (in yellow), the intersection between two cylinders, and a hyperboloid with a sphere (the last one was computed with our ray-tracer).

8.8.3 Complexity of some major operations of QCGA

Let us evaluate the computational cost of checking whether a point is on a quadric. $\mathbf{q}^*$ has a total of 12 basis vector components. For each basis vector, at most 1 inner product is performed, please refer to the Equation (1.4.6). Finally, the number of point component is 12. Thus, the product $\mathbf{x} \cdot \mathbf{q}^*$ requires at most $12 \times 12 = 144$ products.

The computation of the tangent plane is performed by first the computation of the normal vector as:

$$\begin{aligned}\mathbf{n}_\epsilon = &\Big(((\mathbf{x}\cdot\mathbf{e}_1)\mathbf{e}_{\infty 1} + (\mathbf{x}\cdot\mathbf{e}_2)\mathbf{e}_{\infty 4} + (\mathbf{x}\cdot\mathbf{e}_3)\mathbf{e}_{\infty 5} + \mathbf{e}_1)\cdot\mathbf{Q}^*\Big)\mathbf{e}_1 + \\ &\Big(((\mathbf{x}\cdot\mathbf{e}_2)\mathbf{e}_{\infty 2} + (\mathbf{x}\cdot\mathbf{e}_1)\mathbf{e}_{\infty 4} + (\mathbf{x}\cdot\mathbf{e}_3)\mathbf{e}_{\infty 6} + \mathbf{e}_2)\cdot\mathbf{Q}^*\Big)\mathbf{e}_2 + \\ &\Big(((\mathbf{x}\cdot\mathbf{e}_3)\mathbf{e}_{\infty 3} + (\mathbf{x}\cdot\mathbf{e}_1)\mathbf{e}_{\infty 5} + (\mathbf{x}\cdot\mathbf{e}_2)\mathbf{e}_{\infty 6} + \mathbf{e}_3)\cdot\mathbf{Q}^*\Big)\mathbf{e}_3. \end{aligned} \tag{8.8.1}$$

This computation required the inner product between a vector with 12 components and another vector with 4 components. This computation is repeated for each Euclidean basis vector thus the computation of the normal vector requires $3 \times 4 \times 12 = 144$ inner products.

Then, the tangent plane is computed using the normal vector as follows:

$$\boldsymbol{\Pi}^* = \mathbf{n}_\epsilon + \frac{1}{3}(\mathbf{e}_{\infty 1} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 3})\sqrt{-2(\mathbf{e}_{o1} + \mathbf{e}_{o2} + \mathbf{e}_{o3})\cdot\mathbf{x}}. \tag{8.8.2}$$

This computation requires the computation of an inner product of a vector with 3 components $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ with a 12 component-vector. This means $12 \times 3 = 36$ products. Thus, the total number of inner products required in the computation of the tangent plane is $144 + 36 = 180$ products.

The final computational feature is the quadric-line intersection. In QCGA, this simply consists in computing the outer product:

$$\mathbf{C}^* = \mathbf{Q}^* \wedge \mathbf{L}^* \tag{8.8.3}$$

The number of components of $\mathbf{Q}^*$ is 12 as already seen. In QCGA, we defined a line with the 6 Plücker coefficients as:

$$\mathbf{L}^* = 3\,\mathbf{m}\,\mathbf{I}_\epsilon + (\mathbf{e}_{\infty 3} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1}) \wedge \mathbf{n}\,\mathbf{I}_\epsilon. \tag{8.8.4}$$

The number of components of both $\mathbf{m}$ and $\mathbf{n}$ is 3. The outer product $(\mathbf{e}_{\infty 3} + \mathbf{e}_{\infty 2} + \mathbf{e}_{\infty 1}) \wedge \mathbf{n}\,\mathbf{I}_\epsilon$ yields a copy of the 3 components of $\mathbf{n}$ along $\mathbf{e}_{\infty 1}, \mathbf{e}_{\infty 2}, \mathbf{e}_{\infty 3}$ basis vectors. Thus, the number of components of $\mathbf{l}^*$ is $3 \times 3 + 3 = 12$. Finally, the cost of the outer product between $\mathbf{Q}^*$ and $\mathbf{L}^*$ is

$12 \times 12 = 144$ products.

The table 8.7 summarizes the computational features of the proposed framework compared to DPGA and DCGA. We remark that the computation of the tangent plane is more efficient using

TABLE 8.7: Comparison of the computational features between QCGA, DCGA, DPGA in number of Geometric Algebra operations

Feature	DPGA	DCGA	**QCGA**
point is on a quadric	**144**	750	**144**
tangent plane	**64**	541	180
quadric line intersection	192	300	**144**

DPGA whereas the intersection between a quadric and a line requires less computations using QCGA. Furthermore, some versors are not defined in some models.

Now, in order to obtain an efficient model of quadrics using Geometric Algebra, that:

- is efficient for the main computer graphics operations,
- inherits the elegance of CGA (objects can be constructed as the outer product of points),
- is capable of transforming any general quadrics,

we propose a mapping between the three models. It is merely based on the fact that it is not hard to extract the components of the quadrics in the three models. The following chapter focusses on this mapping.

Chapter 9

Mapping

As a practical application, it might be interesting to construct a quadric from 9 points then rotating this quadric. For the moment, QCGA is the only approach, in Geometric Algebra, that can construct quadric from 9 points, but it does not yet support all the transformations. Furthermore, the last chapter shows that some operations are worth doing in a certain framework. These points are our motivation for defining new operators that convert quadrics surfaces between the three presented frameworks.

The key idea is that for any entities representing quadric surface in QCGA,DCGA and DPGA, it is possible to convert the entity such that all the coefficients of the quadrics:

$$\mathrm{a}x^2 + \mathrm{b}y^2 + \mathrm{c}z^2 + \mathrm{d}xy + \mathrm{e}yz + \mathrm{f}zx + \mathrm{g}x + \mathrm{h}y + \mathrm{i}z + \mathrm{j} = 0, \tag{9.0.1}$$

can be extracted easily.

9.1 DCGA reciprocal operators

This means defining reciprocal operators for DCGA:

$$\begin{aligned} \mathbf{T}^{x^2} &= \mathbf{e}_1 \wedge \mathbf{e}_4 \quad \mathbf{T}_{y^2} = \mathbf{e}_2 \wedge \mathbf{e}_5 \\ \mathbf{T}^{z^2} &= \mathbf{e}_3 \wedge \mathbf{e}_6 \quad \mathbf{T}_1 = \mathbf{e}_{o1} \wedge \mathbf{e}_{o2} \end{aligned} \tag{9.1.1}$$

along with the 6 following:

$$\begin{aligned}
\mathbf{T}^{x} &= \left(\mathbf{e}_1 \wedge \mathbf{e}_{o2} + \mathbf{e}_{o1} \wedge \mathbf{e}_4\right) \\
\mathbf{T}^{y} &= \left(\mathbf{e}_2 \wedge \mathbf{e}_{o2} + \mathbf{e}_{o1} \wedge \mathbf{e}_5\right) \\
\mathbf{T}^{z} &= \left(\mathbf{e}_3 \wedge \mathbf{e}_{o2} + \mathbf{e}_{o1} \wedge \mathbf{e}_6\right) \\
\mathbf{T}^{xy} &= \left(\mathbf{e}_1 \wedge \mathbf{e}_5 + \mathbf{e}_2 \wedge \mathbf{e}_4\right) \\
\mathbf{T}^{xz} &= \left(\mathbf{e}_1 \wedge \mathbf{e}_6 + \mathbf{e}_3 \wedge \mathbf{e}_4\right) \\
\mathbf{T}^{yz} &= \left(\mathbf{e}_3 \wedge \mathbf{e}_5 + \mathbf{e}_2 \wedge \mathbf{e}_6\right)
\end{aligned} \tag{9.1.2}$$

These reciprocal operators verify the following properties:

$$\begin{aligned}
&\mathbf{T}^{x^2} \cdot \mathbf{T}_{x^2} = 1,\ \mathbf{T}^{y^2} \cdot \mathbf{T}_{y^2} = 1,\ \mathbf{T}^{z^2} \cdot \mathbf{T}_{z^2} = 1,\ \mathbf{T}^{xy} \cdot \mathbf{T}_{xy} = 1,\ \mathbf{T}^{xz} \cdot \mathbf{T}_{xz} = 1, \\
&\mathbf{T}^{yz} \cdot \mathbf{T}_{yz} = 1,\ \mathbf{T}^{x} \cdot \mathbf{T}_{x} = 1,\ \mathbf{T}^{y} \cdot \mathbf{T}_{y} = 1,\ \mathbf{T}^{z} \cdot \mathbf{T}_{z} = 1,\ \mathbf{T}^{1} \cdot \mathbf{T}_{1} = 1
\end{aligned} \tag{9.1.3}$$

Then, given $\mathbf{q}_{DCGA}$ the entity representing a quadric of DCGA, any coefficients of this quadric (9.0.1) can be extracted as:

$$\begin{aligned}
&\mathbf{T}^{x^2} \cdot \mathbf{Q}_{DCGA} = \mathrm{a},\ \mathbf{T}^{y^2} \cdot \mathbf{Q}_{DCGA} = \mathrm{b},\ \mathbf{T}^{z^2} \cdot \mathbf{Q}_{DCGA} = \mathrm{c},\ \mathbf{T}^{xy} \cdot \mathbf{Q}_{DCGA} = \mathrm{d},\ \mathbf{T}^{xz} \cdot \mathbf{Q}_{DCGA} = \mathrm{e}, \\
&\mathbf{T}^{yz} \cdot \mathbf{Q}_{DCGA} = \mathrm{f},\ \mathbf{T}^{x} \cdot \mathbf{Q}_{DCGA} = \mathrm{g},\ \mathbf{T}^{y} \cdot \mathbf{Q}_{DCGA} = \mathrm{h},\ \mathbf{T}^{z} \cdot \mathbf{Q}_{DCGA} = \mathrm{i},\ \mathbf{T}^{1} \cdot \mathbf{Q}_{DCGA} = \mathrm{j}
\end{aligned} \tag{9.1.4}$$

9.2 DPGA reciprocal operators

In a similar way, let us note $\mathbf{W}$ reciprocal operators for DPGA

$$\begin{aligned}
\mathbf{W}^{x^2} &= \mathbf{w}_0^* \wedge \mathbf{w}_0 & \mathbf{W}^{y^2} &= \mathbf{w}_1^* \wedge \mathbf{w}_1 \\
\mathbf{W}^{z^2} &= \mathbf{w}_2^* \wedge \mathbf{w}_2 & \mathbf{W}^{xy} &= 2\mathbf{w}_1^* \wedge \mathbf{w}_0 \\
\mathbf{W}^{xz} &= 2\mathbf{w}_2^* \wedge \mathbf{w}_0 & \mathbf{W}^{yz} &= 2\mathbf{w}_2^* \wedge \mathbf{w}_1 \\
\mathbf{W}^{x} &= 2\mathbf{w}_3^* \wedge \mathbf{w}_0 & \mathbf{W}^{y} &= 2\mathbf{w}_3^* \wedge \mathbf{w}_1 \\
\mathbf{W}^{z} &= 2\mathbf{w}_3^* \wedge \mathbf{w}_2 & \mathbf{W}^{1} &= \mathbf{W}_3^* \wedge \mathbf{w}_3
\end{aligned} \tag{9.2.1}$$

Again, the following properties hold:

$$\begin{aligned}
&\mathbf{W}^{x^2} \cdot \mathbf{W}_{x^2} = 1,\ \mathbf{W}^{y^2} \cdot \mathbf{W}_{y^2} = 1,\ \mathbf{W}^{z^2} \cdot \mathbf{W}_{z^2} = 1,\ \mathbf{W}^{xy} \cdot \mathbf{W}_{xy} = 1,\ \mathbf{W}^{xz} \cdot \mathbf{W}_{xz} = 1, \\
&\mathbf{W}^{yz} \cdot \mathbf{W}_{yz} = 1,\ \mathbf{W}^{x} \cdot \mathbf{W}_{x} = 1,\ \mathbf{W}^{y} \cdot \mathbf{W}_{y} = 1,\ \mathbf{W}^{z} \cdot \mathbf{W}_{z} = 1,\ \mathbf{W}^{1} \cdot \mathbf{W}_{1} = 1
\end{aligned} \tag{9.2.2}$$

Then, given $\mathbf{q}_{DPGA}$ the entity representing a quadric of DPGA, any coefficients of this quadric (9.0.1) can be extracted as:

$$\begin{aligned}&\mathbf{W}^{x^2}\cdot\mathbf{Q}_{DPGA}=\mathrm{a},\ \mathbf{W}^{y^2}\cdot\mathbf{Q}_{DPGA}=\mathrm{b},\ \mathbf{W}^{z^2}\cdot\mathbf{Q}_{DPGA}=\mathrm{c},\ \mathbf{W}^{xy}\cdot\mathbf{Q}_{DPGA}=\mathrm{d},\ \mathbf{W}^{xz}\cdot\mathbf{Q}_{DPGA}=\mathrm{e},\\&\mathbf{W}^{yz}\cdot\mathbf{Q}_{DPGA}=\mathrm{f},\ \mathbf{W}^{x}\cdot\mathbf{Q}_{DPGA}=\mathrm{g},\ \mathbf{W}^{y}\cdot\mathbf{Q}_{DPGA}=\mathrm{h},\ \mathbf{W}^{z}\cdot\mathbf{Q}_{DPGA}=\mathrm{i},\ \mathbf{W}^{1}\cdot\mathbf{Q}_{DPGA}=\mathrm{j}\end{aligned} \tag{9.2.3}$$

9.3 QCGA reciprocal operators

For QCGA, quadrics can be either representing using the primal form or the dual form. We define the reciprocal operators for the dual form. Indeed, if one considers the primal form, then this would consist in computing the dual of the primal and apply the following reciprocal operators:

$$\begin{aligned}\mathbf{Q}^{x^2}&=\tfrac{1}{2}\mathbf{e}_{\infty 1} & \mathbf{Q}^{y^2}&=\tfrac{1}{2}\mathbf{e}_{\infty 2}\\ \mathbf{Q}^{z^2}&=\tfrac{1}{2}\mathbf{e}_{\infty 3} & \mathbf{Q}^{xy}&=\mathbf{e}_{\infty 4}\\ \mathbf{Q}^{xz}&=\mathbf{e}_{\infty 5} & \mathbf{Q}^{yz}&=\mathbf{e}_{\infty 6}\\ \mathbf{Q}^{x}&=\mathbf{e}_{1} & \mathbf{Q}^{y}&=\mathbf{e}_{2}\\ \mathbf{Q}^{z}&=\mathbf{e}_{3} & \mathbf{Q}^{1}&=\mathbf{e}_{o1}+\mathbf{e}_{o2}+\mathbf{e}_{o3}\end{aligned} \tag{9.3.1}$$

Given a general quadric $\mathbf{q}^*$ whose coefficients are $(\mathrm{a},\mathrm{b},\mathrm{c},\cdots,\mathrm{j})$, the properties of these operators are as follows:

$$\begin{aligned}&\mathbf{Q}^{x^2}\cdot\mathbf{Q}^*=\mathrm{a},\ \mathbf{Q}^{y^2}\cdot\mathbf{Q}^*=\mathrm{b},\ \mathbf{Q}^{z^2}\cdot\mathbf{Q}^*=\mathrm{c},\ \mathbf{Q}^{xy}\cdot\mathbf{Q}^*=\mathrm{d},\ \mathbf{Q}^{xz}\cdot\mathbf{Q}^*=\mathrm{e},\\&\mathbf{Q}^{yz}\cdot\mathbf{Q}^*=\mathrm{f},\ \mathbf{Q}^{x}\cdot\mathbf{Q}^*=\mathrm{g},\ \mathbf{Q}^{y}\cdot\mathbf{Q}^*=\mathrm{h},\ \mathbf{Q}^{z}\cdot\mathbf{Q}^*=\mathrm{i},\ \mathbf{Q}^{1}\cdot\mathbf{Q}^*=\mathrm{j}\end{aligned} \tag{9.3.2}$$

9.4 Test

We tested this approach by defining an ellipsoid from 9 points using QCGA. Then we rotate it using DPGA and back-convert the rotated ellipsoid into QCGA framework. In terms of Geometric Algebra computations, first we compute the quadric:

$$\mathbf{Q}^* = (\mathbf{x}_1\wedge\mathbf{x}_2\wedge\cdots\wedge\mathbf{x}_9\wedge\mathbf{I}_o^{\triangleright})^* \tag{9.4.1}$$

Then, we apply the extraction operators of QCGA to convert the QCGA quadric to DPGA quadric.

$$\begin{aligned}\mathbf{Q}_{DPGA} =& 4(\mathbf{Q}^{x^2}\cdot\mathbf{q}^*)\mathbf{w}_0^*\wedge\mathbf{w}_0 + 4(\mathbf{Q}^{y^2}\cdot\mathbf{q}^*)\mathbf{w}_1^*\wedge\mathbf{w}_1 + 4(\mathbf{Q}^{z^2}\cdot\mathbf{q}^*)\mathbf{w}_2^*\wedge\mathbf{w}_2\\ &+4(\mathbf{Q}^{1}\cdot\mathbf{q}^*)\mathbf{w}_3^*\wedge\mathbf{w}_3 + 2(\mathbf{Q}^{xy}\cdot\mathbf{q}^*)(\mathbf{w}_0^*\wedge\mathbf{w}_1 + \mathbf{w}_1^*\wedge\mathbf{w}_0)\\ &+2(\mathbf{Q}^{xz}\cdot\mathbf{q}^*)(\mathbf{w}_0^*\wedge\mathbf{w}_2 + \mathbf{w}_2^*\wedge\mathbf{w}_0) + 2(\mathbf{Q}^{yz}\cdot\mathbf{q}^*)(\mathbf{w}_1^*\wedge\mathbf{w}_2 + \mathbf{w}_2^*\wedge\mathbf{w}_1)\\ &+2(\mathbf{Q}^{x}\cdot\mathbf{q}^*)(\mathbf{w}_0^*\wedge\mathbf{w}_3 + \mathbf{w}_3^*\wedge\mathbf{w}_0) + 2(\mathbf{Q}^{y}\cdot\mathbf{q}^*)(\mathbf{w}_1^*\wedge\mathbf{w}_3 + \mathbf{w}_3^*\wedge\mathbf{w}_1)\\ &+2(\mathbf{Q}^{z}\cdot\mathbf{q}^*)(\mathbf{w}_2^*\wedge\mathbf{w}_3 + \mathbf{w}_3^*\wedge\mathbf{w}_2)\end{aligned} \tag{9.4.2}$$

The rotation is now performed as follows:

$$\mathbf{Q}_{DPGA} = R\mathbf{Q}_{DPGA}R^{-1} \tag{9.4.3}$$

The rotor R is defined as:

$$R = \exp(\frac{1}{2}\theta\mathbf{w}_i\mathbf{w}_j^*) \tag{9.4.4}$$

where $i \neq j$

The final step is to back-convert the resulting quadric to the QCGA framework. It is merely computed using the QCGA extraction operators as follows:

$$\begin{aligned}\mathbf{Q}^* = -\big(&2(\mathbf{W}^{x^2}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_{o1} + 2(\mathbf{W}^{y^2}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_{o2} + 2(\mathbf{W}^{z^2}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_{o3}\\ &+ (\mathbf{W}^{xy}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_{o4} + (\mathbf{W}^{xz}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_{o5} + (\mathbf{W}^{yz}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_{o6}\big)\\ &+ \big((\mathbf{W}^{x}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_1 + (\mathbf{W}^{y}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_2 + (\mathbf{W}^{z}\cdot\mathbf{Q}_{DPGA})\mathbf{e}_3\big)\\ &- \frac{(\mathbf{W}^{1}\cdot\mathbf{Q}_{DPGA})}{3}(\mathbf{e}_{\infty1} + \mathbf{e}_{\infty2} + \mathbf{e}_{\infty3}).\end{aligned} \tag{9.4.5}$$

Note that the program can be found in the plugin folder of the git repository previously shown.

9.5 Potential computational framework

In order to implement and convert easily quadrics from different frameworks, it is not necessary to generate each framework independently. This is equivalent with seeking for a framework that encompasses the other framework.

First, $\mathbb{G}_{8,2}$ can not be the support for such a mapping due to the fact that it does not embed $\mathbb{G}_{4,4}$. Whereas $\mathbb{G}_{9,6}$ include both $\mathbb{G}_{4,4}$ and $\mathbb{G}_{8,2}$. The next paragraphs show the way we embed the two frameworks using QCGA.

As for DCGA and if we consider the vector space composed of first the Euclidean vector space

$\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ along with the null basis vector space $\{\mathbf{e}_{o1}, \mathbf{e}_{o2}, \mathbf{e}_{o3}, \mathbf{e}_{o4}, \mathbf{e}_{o5}\}$ and finally $\{\mathbf{e}_{\infty 1}, \mathbf{e}_{\infty 2}, \mathbf{e}_{\infty 3}, \mathbf{e}_{\infty 4}, \mathbf{e}_{\infty 5}\}$. Note that these are QCGA basis vectors. Then using the following basis transformation:

$$\begin{aligned} \mathbf{e}_4 &= \mathbf{e}_{o3} - \tfrac{1}{2}\mathbf{e}_{\infty 3}, \\ \mathbf{e}_5 &= \mathbf{e}_{o4} - \tfrac{1}{2}\mathbf{e}_{\infty 4}, \\ \mathbf{e}_6 &= \mathbf{e}_{o5} - \tfrac{1}{2}\mathbf{e}_{\infty 5}, \end{aligned} \tag{9.5.1}$$

and using the new $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_{o1}, \mathbf{e}_{\infty 1}, \mathbf{e}_{o2}, \mathbf{e}_{\infty 2}, \mathbf{e}_4, \mathbf{e}_5, \mathbf{e}_6\}$ basis, it is possible to compute DCGA points (x, y, z) as:

$$\begin{aligned} \mathbf{P}_{DCGA} = \quad & (\mathbf{e}_{o1} + x\mathbf{e}_1 + y\mathbf{e}_2 + z\mathbf{e}_3 + \tfrac{1}{2}(x^2+y^2+z^2)\mathbf{e}_{\infty 1}) \\ & \wedge (\mathbf{e}_{o2} + x\mathbf{e}_4 + y\mathbf{e}_5 + z\mathbf{e}_6 + \tfrac{1}{2}(x^2+y^2+z^2)\mathbf{e}_{\infty 2}) \end{aligned} \tag{9.5.2}$$

Concerning DPGA, if we consider the vector space group $\{\mathbf{e}_{o1}, \mathbf{e}_{o2}, \mathbf{e}_{o3}, \mathbf{e}_{o4}, \mathbf{e}_{\infty 1}, \mathbf{e}_{\infty 2}, \mathbf{e}_{\infty 3}, \mathbf{e}_{\infty 4}\}$ of QCGA. Then it is possible to define the new DPGA basis $\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ as follows:

$$\begin{aligned} \mathbf{w}_i &= -\mathbf{e}_{oj} \\ \mathbf{w}_i^* &= \tfrac{1}{2}\mathbf{e}_{\infty j} \end{aligned} \tag{9.5.3}$$

where $i = 0, 1, 2, 3$ and $j = i + 1$. Then, the following properties hold:

$$\begin{aligned} \mathbf{w}_i \cdot \mathbf{w}_i^* &= 0.5 \\ \mathbf{w}_i \cdot \mathbf{w}_i &= 0 \end{aligned} \tag{9.5.4}$$

The resulting metric corresponds to the metric shown in Table 7.3 and a DPGA point can be simply defined as follows:

$$\begin{aligned} \mathbf{p}_{DPGA} &= x\mathbf{w}_0 + y\mathbf{w}_1 + z\mathbf{w}_2 + w\mathbf{w}_3 \\ \mathbf{p}^*_{DPGA} &= x\mathbf{w}_0^* + y\mathbf{w}_1^* + z\mathbf{w}_2^* + w\mathbf{w}_3^* \end{aligned} \tag{9.5.5}$$

Using this method, all the computations can be performed using the same algebra, namely $\mathbb{G}_{9,6}$.

Conclusion and perspectives

This part concludes this thesis. We first summarize our main contribution and finish with our future works. This thesis includes both practical and theoretic works.

As practical work, we proposed a new Geometric Algebra implementation called *Garamon*, a library generator written in C++ programming language and producing specialized Geometric Algebra (GA) libraries also in C++. This library was designed to be a flexible, portable, and computationally efficient Geometric Algebra implementation dedicated for both low and high dimensions. The proposed approach includes new computationally efficient Geometric Algebra algorithms. We showed that these resulting algorithms have lower complexity than the state of the methods. More precisely, we proposed a recursive formalism such that the outer product and inner product can be computed in the worst case in 3^d products instead of 4^d products for the state-of-the-art methods. Experimental results showed the memory and computational performances of the library.

The resulting library opened some lock, such that one could easily explore some computations in vector spaces whose dimension is higher than 12. We thus decided to investigate new models to represent and manipulate quadric surfaces.

This lead to a new Geometric Algebra framework, Quadric Conformal Geometric Algebra (QCGA), that handles the construction of quadric surfaces using the implicit equation and/or control points. We showed that QCGA naturally includes CGA objects and generalizes some dedicated constructions and naturally embed other framework.

The intersection between objects in the QCGA space is merely computed using the outer product. This manuscript also details the computation of the tangent plane and the normal vector at a point on a quadric surface. Although, the dimension of the vector space is considered as high, the representation of geometric objects use only low dimensional subspaces of the algebra. We even proved that some operations of this model are computationally more efficient using QCGA

than other frameworks of lower dimensions. Therefore, QCGA can be used for numerical computations in applications such as computer graphics. We proved that QCGA can be the support for computing translations.

Finally, we proposed a mapping between the main Geometric Algebra models that makes computationally efficient operations required for the manipulation of quadrics in Geometric Algebra.

Future works

As future works, we plan to first understand what the transformations look like in the Quadric Conformal Geometric Algebra, as discussed with Joan Lasenby and Leo Dorst. In particular, we want to know if whether versors for all the projective transformations can be defined. This would include rotations, shears, non-isotropic scales. We would also investigate the distance computation in this framework.

The resulting framework could be the base for a discrete geometry model using Geometric Algebra, see [68, 77] and we could try to apply it to [24]. We want also to study in more details the intersection of quadrics. Ideally, we would like to show that there is a Geometric Algebra object for each kind of intersection. Other interesting works that we would like to perform is the extension of QCGA for cubics and quartics. More generally, we would study the intuitive expression and manipulation of surfaces using such high dimensional spaces. Again, this would mean extending the dimension of the vector space however we would still use low dimensional subspaces of the algebra. In a similar way, a complexity study would be developed leading to a general framework that could be used a Geometric Algebra model to handle surfaces. A comparison with linear algebra methods would be worth performing both in terms of computational and memory complexity. Finally, we would like to develop a certification of the construction of such surfaces. To achieve this, [65] along with [36] would be our base.

As for Garamon, it is in process to fine tune some optimization and to integrate a meta-programming approach. Moreover, a binding with Python programming language is among works in progress. The idea of this binding is to facilitate the prototyping work of implementing new approaches using high dimensional Geometric Algebras. Finally, we aim at extending Garamon for Geographical Information Science and very high dimensional space. We would develop new algorithms to compute only required products at runtime. To achieve this, cache-oblivious [17] of Geometric Algebra operators will be developed. These algorithms will be based on the prefix

tree approach. In order to compute as low products as possible we would also base our approach on stochastic acceptance [61] of the products of Geometric Algebra with respects to the products computed by the user. Another interesting work that would be performed is the formalization of Clifford Algebra in COQ using the resulting approach based on [36]. These results along with our approach of Geometric Algebra on surfaces could be the base for the certification of surfaces manipulation [82] using Geometric Algebra.

Appendix A

Garamon sample

This appendix presents the functions generated for the computation of any outer products in the Conformal Geometric Algebra space.

LISTING A.1: Code example in CGA with Garamon

```
// Copyright (c) 2018 by University Paris-Est Marne-la-Vallee
// OuterExplicit.hpp
// This file is part of the Garamon for c3ga.
// Authors: Stephane Breuils and Vincent Nozick
// Contact: vincent.nozick@u-pem.fr
//
// Licence MIT
// A a copy of the MIT License is given along with this program

/// \file OuterExplicit.hpp
/// \author Stephane Breuils, Vincent Nozick
/// \brief Explicit precomputed per grades outer product.


#ifndef C3GA_OUTER_PRODUCT_EXPLICIT_HPP__
#define C3GA_OUTER_PRODUCT_EXPLICIT_HPP__
#pragma once

#include <Eigen/Core>

#include "c3ga/Mvec.hpp"
#include "c3ga/Outer.hpp"



/*!
 * @namespace c3ga
 */
namespace c3ga {
	template<typename T> class Mvec;

	/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 0) and mv2 (grade 0).
	/// \tparam the type of value that we manipulate, either float or double or something.
	/// \param mv1 - the first homogeneous multivector of grade 0 represented as an Eigen::VectorXd
	/// \param mv2 - the second homogeneous multivector of grade 0 represented as a Eigen::VectorXd
	/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 0
	template<typename T>
	void outer_0_0(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
		mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(0);
	}

	/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 0) and mv2 (grade 1).
	/// \tparam the type of value that we manipulate, either float or double or something.
	/// \param mv1 - the first homogeneous multivector of grade 0 represented as an Eigen::VectorXd
```

```
/// \param mv2 - the second homogeneous multivector of grade 1 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 1
template<typename T>
void outer_0_1(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
	mv3 += mv1.coeff(0)*mv2;
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 0) and mv2 (grade 2).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 0 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 2 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 2
template<typename T>
void outer_0_2(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
	mv3 += mv1.coeff(0)*mv2;
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 0) and mv2 (grade 3).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 0 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 3 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 3
template<typename T>
void outer_0_3(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
	mv3 += mv1.coeff(0)*mv2;
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 0) and mv2 (grade 4).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 0 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 4 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 4
template<typename T>
void outer_0_4(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
	mv3 += mv1.coeff(0)*mv2;
}


/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 0) and mv2 (grade 5).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 0 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 5 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 5
template<typename T>
void outer_0_5(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
	mv3 += mv1.coeff(0)*mv2;
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 1) and mv2 (grade 0).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 1 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 0 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 1
template<typename T>
void outer_1_0(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
	mv3 += mv1*mv2.coeff(0);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 1) and mv2 (grade 1).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 1 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 1 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 2
template<typename T>
void outer_1_1(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
	mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(1) - mv1.coeff(1)*mv2.coeff(0);
	mv3.coeffRef(1) += mv1.coeff(0)*mv2.coeff(2) - mv1.coeff(2)*mv2.coeff(0);
	mv3.coeffRef(2) += mv1.coeff(0)*mv2.coeff(3) - mv1.coeff(3)*mv2.coeff(0);
	mv3.coeffRef(3) += mv1.coeff(0)*mv2.coeff(4) - mv1.coeff(4)*mv2.coeff(0);
```

```
        mv3.coeffRef(4) += mv1.coeff(1)*mv2.coeff(2) - mv1.coeff(2)*mv2.coeff(1);
        mv3.coeffRef(5) += mv1.coeff(1)*mv2.coeff(3) - mv1.coeff(3)*mv2.coeff(1);
        mv3.coeffRef(6) += mv1.coeff(1)*mv2.coeff(4) - mv1.coeff(4)*mv2.coeff(1);
        mv3.coeffRef(7) += mv1.coeff(2)*mv2.coeff(3) - mv1.coeff(3)*mv2.coeff(2);
        mv3.coeffRef(8) += mv1.coeff(2)*mv2.coeff(4) - mv1.coeff(4)*mv2.coeff(2);
        mv3.coeffRef(9) += mv1.coeff(3)*mv2.coeff(4) - mv1.coeff(4)*mv2.coeff(3);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 1) and mv2 (grade 2).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 1 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 2 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 3
template<typename T>
void outer_1_2(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
        mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(4) - mv1.coeff(1)*mv2.coeff(1) + mv1.coeff(2)*mv2.coeff(0);
        mv3.coeffRef(1) += mv1.coeff(0)*mv2.coeff(5) - mv1.coeff(1)*mv2.coeff(2) + mv1.coeff(3)*mv2.coeff(0);
        mv3.coeffRef(2) += mv1.coeff(0)*mv2.coeff(6) - mv1.coeff(1)*mv2.coeff(3) + mv1.coeff(4)*mv2.coeff(0);
        mv3.coeffRef(3) += mv1.coeff(0)*mv2.coeff(7) - mv1.coeff(2)*mv2.coeff(2) + mv1.coeff(3)*mv2.coeff(1);
        mv3.coeffRef(4) += mv1.coeff(0)*mv2.coeff(8) - mv1.coeff(2)*mv2.coeff(3) + mv1.coeff(4)*mv2.coeff(1);
        mv3.coeffRef(5) += mv1.coeff(0)*mv2.coeff(9) - mv1.coeff(3)*mv2.coeff(3) + mv1.coeff(4)*mv2.coeff(2);
        mv3.coeffRef(6) += mv1.coeff(1)*mv2.coeff(7) - mv1.coeff(2)*mv2.coeff(5) + mv1.coeff(3)*mv2.coeff(4);
        mv3.coeffRef(7) += mv1.coeff(1)*mv2.coeff(8) - mv1.coeff(2)*mv2.coeff(6) + mv1.coeff(4)*mv2.coeff(4);
        mv3.coeffRef(8) += mv1.coeff(1)*mv2.coeff(9) - mv1.coeff(3)*mv2.coeff(6) + mv1.coeff(4)*mv2.coeff(5);
        mv3.coeffRef(9) += mv1.coeff(2)*mv2.coeff(9) - mv1.coeff(3)*mv2.coeff(8) + mv1.coeff(4)*mv2.coeff(7);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 1) and mv2 (grade 3).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 1 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 3 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 4
template<typename T>
void outer_1_3(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
        mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(6) - mv1.coeff(1)*mv2.coeff(3) + mv1.coeff(2)*mv2.coeff(1) - mv1.coeff(3)*mv2.coeff(0);
        mv3.coeffRef(1) += mv1.coeff(0)*mv2.coeff(7) - mv1.coeff(1)*mv2.coeff(4) + mv1.coeff(2)*mv2.coeff(2) - mv1.coeff(4)*mv2.coeff(0);
        mv3.coeffRef(2) += mv1.coeff(0)*mv2.coeff(8) - mv1.coeff(1)*mv2.coeff(5) + mv1.coeff(3)*mv2.coeff(2) - mv1.coeff(4)*mv2.coeff(1);
        mv3.coeffRef(3) += mv1.coeff(0)*mv2.coeff(9) - mv1.coeff(2)*mv2.coeff(5) + mv1.coeff(3)*mv2.coeff(4) - mv1.coeff(4)*mv2.coeff(3);
        mv3.coeffRef(4) += mv1.coeff(1)*mv2.coeff(9) - mv1.coeff(2)*mv2.coeff(8) + mv1.coeff(3)*mv2.coeff(7) - mv1.coeff(4)*mv2.coeff(6);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 1) and mv2 (grade 4).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 1 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 4 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 5
template<typename T>
void outer_1_4(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
        mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(4) - mv1.coeff(1)*mv2.coeff(3) + mv1.coeff(2)*mv2.coeff(2) - mv1.coeff(3)*mv2.coeff(1)
                         + mv1.coeff(4)*mv2.coeff(0);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 2) and mv2 (grade 0).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 2 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 0 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 2
template<typename T>
void outer_2_0(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
        mv3 += mv1*mv2.coeff(0);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 2) and mv2 (grade 1).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 2 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 1 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 3
template<typename T>
void outer_2_1(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
```

```
        mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(2) - mv1.coeff(1)*mv2.coeff(1) + mv1.coeff(4)*mv2.coeff(0);
        mv3.coeffRef(1) += mv1.coeff(0)*mv2.coeff(3) - mv1.coeff(2)*mv2.coeff(1) + mv1.coeff(5)*mv2.coeff(0);
        mv3.coeffRef(2) += mv1.coeff(0)*mv2.coeff(4) - mv1.coeff(3)*mv2.coeff(1) + mv1.coeff(6)*mv2.coeff(0);
        mv3.coeffRef(3) += mv1.coeff(1)*mv2.coeff(3) - mv1.coeff(2)*mv2.coeff(2) + mv1.coeff(7)*mv2.coeff(0);
        mv3.coeffRef(4) += mv1.coeff(1)*mv2.coeff(4) - mv1.coeff(3)*mv2.coeff(2) + mv1.coeff(8)*mv2.coeff(0);
        mv3.coeffRef(5) += mv1.coeff(2)*mv2.coeff(4) - mv1.coeff(3)*mv2.coeff(3) + mv1.coeff(9)*mv2.coeff(0);
        mv3.coeffRef(6) += mv1.coeff(4)*mv2.coeff(3) - mv1.coeff(5)*mv2.coeff(2) + mv1.coeff(7)*mv2.coeff(1);
        mv3.coeffRef(7) += mv1.coeff(4)*mv2.coeff(4) - mv1.coeff(6)*mv2.coeff(2) + mv1.coeff(8)*mv2.coeff(1);
        mv3.coeffRef(8) += mv1.coeff(5)*mv2.coeff(4) - mv1.coeff(6)*mv2.coeff(3) + mv1.coeff(9)*mv2.coeff(1);
        mv3.coeffRef(9) += mv1.coeff(7)*mv2.coeff(4) - mv1.coeff(8)*mv2.coeff(3) + mv1.coeff(9)*mv2.coeff(2);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 2) and mv2 (grade 2).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 2 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 2 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 4
template<typename T>
void outer_2_2(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
        mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(7) - mv1.coeff(1)*mv2.coeff(5) + mv1.coeff(2)*mv2.coeff(4) + mv1.coeff(4)*mv2.coeff(2)
                         - mv1.coeff(5)*mv2.coeff(1) + mv1.coeff(7)*mv2.coeff(0);
        mv3.coeffRef(1) += mv1.coeff(0)*mv2.coeff(8) - mv1.coeff(1)*mv2.coeff(6) + mv1.coeff(3)*mv2.coeff(4) + mv1.coeff(4)*mv2.coeff(3)
                         - mv1.coeff(6)*mv2.coeff(1) + mv1.coeff(8)*mv2.coeff(0);
        mv3.coeffRef(2) += mv1.coeff(0)*mv2.coeff(9) - mv1.coeff(2)*mv2.coeff(6) + mv1.coeff(3)*mv2.coeff(5) + mv1.coeff(5)*mv2.coeff(3)
                         - mv1.coeff(6)*mv2.coeff(2) + mv1.coeff(9)*mv2.coeff(0);
        mv3.coeffRef(3) += mv1.coeff(1)*mv2.coeff(9) - mv1.coeff(2)*mv2.coeff(8) + mv1.coeff(3)*mv2.coeff(7) + mv1.coeff(7)*mv2.coeff(3)
                         - mv1.coeff(8)*mv2.coeff(2) + mv1.coeff(9)*mv2.coeff(1);
        mv3.coeffRef(4) += mv1.coeff(4)*mv2.coeff(9) - mv1.coeff(5)*mv2.coeff(8) + mv1.coeff(6)*mv2.coeff(7) + mv1.coeff(7)*mv2.coeff(6)
                         - mv1.coeff(8)*mv2.coeff(5) + mv1.coeff(9)*mv2.coeff(4);
}


/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 2) and mv2 (grade 3).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 2 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 3 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 5
template<typename T>
void outer_2_3(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
        mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(9) - mv1.coeff(1)*mv2.coeff(8) + mv1.coeff(2)*mv2.coeff(7) - mv1.coeff(3)*mv2.coeff(6)
                         + mv1.coeff(4)*mv2.coeff(5) - mv1.coeff(5)*mv2.coeff(4) + mv1.coeff(6)*mv2.coeff(3) + mv1.coeff(7)*mv2.coeff(2)
                         - mv1.coeff(8)*mv2.coeff(1) + mv1.coeff(9)*mv2.coeff(0);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 3) and mv2 (grade 0).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 3 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 0 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 3
template<typename T>
void outer_3_0(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
        mv3 += mv1*mv2.coeff(0);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 3) and mv2 (grade 1).
/// \tparam the type of value that we manipulate, either float or double or something.
/// \param mv1 - the first homogeneous multivector of grade 3 represented as an Eigen::VectorXd
/// \param mv2 - the second homogeneous multivector of grade 1 represented as a Eigen::VectorXd
/// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 4
template<typename T>
void outer_3_1(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
        mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(3) - mv1.coeff(1)*mv2.coeff(2) + mv1.coeff(3)*mv2.coeff(1) - mv1.coeff(6)*mv2.coeff(0);
        mv3.coeffRef(1) += mv1.coeff(0)*mv2.coeff(4) - mv1.coeff(2)*mv2.coeff(2) + mv1.coeff(4)*mv2.coeff(1) - mv1.coeff(7)*mv2.coeff(0);
        mv3.coeffRef(2) += mv1.coeff(1)*mv2.coeff(4) - mv1.coeff(2)*mv2.coeff(3) + mv1.coeff(5)*mv2.coeff(1) - mv1.coeff(8)*mv2.coeff(0);
        mv3.coeffRef(3) += mv1.coeff(3)*mv2.coeff(4) - mv1.coeff(4)*mv2.coeff(3) + mv1.coeff(5)*mv2.coeff(2) - mv1.coeff(9)*mv2.coeff(0);
        mv3.coeffRef(4) += mv1.coeff(6)*mv2.coeff(4) - mv1.coeff(7)*mv2.coeff(3) + mv1.coeff(8)*mv2.coeff(2) - mv1.coeff(9)*mv2.coeff(1);
}

/// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 3) and mv2 (grade 2).
```

```
        /// \tparam the type of value that we manipulate, either float or double or something.
        /// \param mv1 - the first homogeneous multivector of grade 3 represented as an Eigen::VectorXd
        /// \param mv2 - the second homogeneous multivector of grade 2 represented as a Eigen::VectorXd
        /// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 5
        template<typename T>
        void outer_3_2(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
                mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(9) - mv1.coeff(1)*mv2.coeff(8) + mv1.coeff(2)*mv2.coeff(7) + mv1.coeff(3)*mv2.coeff(6)
                                - mv1.coeff(4)*mv2.coeff(5) + mv1.coeff(5)*mv2.coeff(4) - mv1.coeff(6)*mv2.coeff(3) + mv1.coeff(7)*mv2.coeff(2)
                                - mv1.coeff(8)*mv2.coeff(1) + mv1.coeff(9)*mv2.coeff(0);
        }

        /// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 4) and mv2 (grade 0).
        /// \tparam the type of value that we manipulate, either float or double or something.
        /// \param mv1 - the first homogeneous multivector of grade 4 represented as an Eigen::VectorXd
        /// \param mv2 - the second homogeneous multivector of grade 0 represented as a Eigen::VectorXd
        /// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 4
        template<typename T>
        void outer_4_0(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
                mv3 += mv1*mv2.coeff(0);
        }

        /// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 4) and mv2 (grade 1).
        /// \tparam the type of value that we manipulate, either float or double or something.
        /// \param mv1 - the first homogeneous multivector of grade 4 represented as an Eigen::VectorXd
        /// \param mv2 - the second homogeneous multivector of grade 1 represented as a Eigen::VectorXd
        /// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 5
        template<typename T>
        void outer_4_1(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
                mv3.coeffRef(0) += mv1.coeff(0)*mv2.coeff(4) - mv1.coeff(1)*mv2.coeff(3) + mv1.coeff(2)*mv2.coeff(2) - mv1.coeff(3)*mv2.coeff(1)
                                + mv1.coeff(4)*mv2.coeff(0);
        }

        /// \brief Compute the outer product between two homogeneous multivectors mv1 (grade 5) and mv2 (grade 0).
        /// \tparam the type of value that we manipulate, either float or double or something.
        /// \param mv1 - the first homogeneous multivector of grade 5 represented as an Eigen::VectorXd
        /// \param mv2 - the second homogeneous multivector of grade 0 represented as a Eigen::VectorXd
        /// \param mv3 - the result of mv1^mv2, which is also a homogeneous multivector of grade 5
        template<typename T>
        void outer_5_0(const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv1, const Eigen::Matrix<T, Eigen::Dynamic, 1>& mv2, Eigen::Matrix<T, Eigen::Dynamic, 1>& mv3){
                mv3 += mv1*mv2.coeff(0);
        }


    template<typename T>
        std::array<std::array<std::function<void(const Eigen::Matrix<T,Eigen::Dynamic,1> &,
            const Eigen::Matrix<T, Eigen::Dynamic,1>&, Eigen::Matrix<T,Eigen::Dynamic,1>&)>,6>,6> outerFunctionsContainer={{
                {{outer_0_0<T>,outer_0_1<T>,outer_0_2<T>,outer_0_3<T>,outer_0_4<T>,outer_0_5<T>}},
                {{outer_1_0<T>,outer_1_1<T>,outer_1_2<T>,outer_1_3<T>,outer_1_4<T>,{}}},
                {{outer_2_0<T>,outer_2_1<T>,outer_2_2<T>,outer_2_3<T>,{},{}}},
                {{outer_3_0<T>,outer_3_1<T>,outer_3_2<T>,{},{},{}}},
                {{outer_4_0<T>,outer_4_1<T>,{},{},{},{}}},
                {{outer_5_0<T>,{},{},{},{},{}}}
        }};
}/// End of Namespace
#endif // C3GA_OUTER_PRODUCT_EXPLICIT_HPP__
```

References

[1] Rafał Abłamowicz and Bertfried Fauser. "On Parallelizing the Clifford Algebra Product for CLIFFORD". In: *Advances in Applied Clifford Algebras* 24.2 (2014), pp. 553–567. ISSN: 1661-4909. DOI: 10.1007/s00006-014-0445-5. URL: http://dx.doi.org/10.1007/s00006-014-0445-5.

[2] Rafał Abłamowicz and Bertfried Fauser. "Using periodicity theorems for computations in higher dimensional Clifford algebras". In: *Advances in Applied Clifford Algebras* 24.2 (2014), pp. 569–587.

[3] Werner Benger and Wolgang Dobler. "Massive Geometric Algebra: Visions for C++ implementations of geometric algebra to scale into the big data era". In: *Advances in Applied Clifford Algebras* 27 (2017), pp. 2153–2174.

[4] Werner Benger, René Heinzl, Dietmar Hildenbrand, Tino Weinkauf, Holger Theisel, and David Tschumperlé. "Differential Methods for Multidimensional Visual Data Analysis". In: *Handbook of Mathematical Methods in Imaging* (2014), pp. 1–56.

[5] Stéphane Breuils, Vincent Nozick, and Laurent Fuchs. "A Geometric Algebra Implementation using Binary Tree". In: *Advances in Applied Clifford Algebras* 27.3 (2017), pp. 2133–2151. ISSN: 1661-4909. DOI: 10.1007/s00006-017-0770-6. URL: https://doi.org/10.1007/s00006-017-0770-6.

[6] Stéphane Breuils, Vincent Nozick, Laurent Fuchs, Dietmar Hildenbrand, Werner Benger, and Christian Steinmetz. "A Hybrid Approach for Computing Products of High-dimensional Geometric Algebras". In: *Proceedings of the Computer Graphics International Conference, ENGAGE*. CGI '17. Hiyoshi, Japan: ACM, 2017, 43:1–43:6. ISBN: 978-1-4503-5228-4. DOI: 10.1145/3095140.3097284. URL: http://doi.acm.org/10.1145/3095140.3097284.

[7] Stéphane Breuils, Vincent Nozick, Akihiro Sugimoto, and Eckhard Hitzer. "Quadric Conformal Geometric Algebra of $\mathbb{R}^{9,6}$". In: *Advances in Applied Clifford Algebras* 28.2 (2018), p. 35. ISSN: 1661-4909. DOI: 10.1007/s00006-018-0851-1. URL: https://doi.org/10.1007/s00006-018-0851-1.

[8] Alan Bromborsky. *GAlgebra*. https://github.com/brombo/galgebra. Accessed: 2017-04-06.

[9] Sven Buchholz, Kanta Tachibana, and Eckhard MS Hitzer. "Optimal learning rates for Clifford neurons". In: *International conference on artificial neural networks*. Springer. 2007, pp. 864–873.

[10] Fabien Cellier, Pierre-Marie Gandoin, Raphaëlle Chaine, Aurélien Barbier-Accary, and Samir Akkouche. "Simplification and streaming of GIS terrain for web clients". In: *Proceedings of the 17th International Conference on 3D Web Technology*. ACM. 2012, pp. 73–81.

[11] James M Chappell, Azhar Iqbal, John G Hartnett, and Derek Abbott. "The vector algebra war: a historical perspective". In: *IEEE Access* 4 (2016), pp. 1997–2004.

[12] Patrick Charrier, Mariusz Klimek, Christian Steinmetz, and Dietmar Hildenbrand. "Geometric algebra enhanced precompiler for C++, OpenCL and Mathematica's OpenCLLink". In: *Advances in Applied Clifford Algebras* 24.2 (2014), pp. 613–630.

[13] Professor Clifford. "Applications of Grassmann's Extensive Algebra". In: *American Journal of Mathematics* 1.4 (1878), pp. 350–358. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2369379.

[14] Pablo Colapinto. "Spatial computing with conformal geometric algebra". PhD thesis. University of California Santa Barbara, 2011.

[15] Michael J Crowe. *A history of vector analysis: The evolution of the idea of a vectorial system*. Courier Corporation, 1994.

[16] René De La Briandais. "File Searching Using Variable Length Keys". In: *Papers Presented at the March 3-5, 1959, Western Joint Computer Conference*. IRE-AIEE-ACM '59 (Western). New York, NY, USA: ACM, 1959, pp. 295–298.

[17] Erik D Demaine. "Cache-oblivious algorithms and data structures". In: *Lecture Notes from the EEF Summer School on Massive Data Sets* 8.4 (2002), pp. 1–249.

[18] Paul Adrien Maurice Dirac. "The quantum theory of the electron". In: *Proc. R. Soc. Lond. A* 117.778 (1928), pp. 610–624.

[19] Chris Doran and Anthony Lasenby. *Geometric Algebra for Physicists*. Cambridge University Press, 2003. DOI: 10.1017/CBO9780511807497.

[20] Chris Doran, David Hestenes, Frank Sommen, and Nadine Van Acker. "Lie groups as spin groups". In: *Journal of Mathematical Physics* 34.8 (1993), pp. 3642–3669.

[21] Leo Dorst. "3D Oriented Projective Geometry Through Versors of $\mathbb{R}^{3,3}$". In: *Advances in Applied Clifford Algebras* 26.4 (2016), pp. 1137–1172.

[22] Leo Dorst. "The inner products of geometric algebra". In: *Applications of Geometric Algebra in Computer Science and Engineering*. Springer, 2002, pp. 35–46.

[23] Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry*. Morgan Kaufmann, 2007.

[24] Leo Dorst and Rein Van Den Boomgaard. "An analytical theory of mathematical morphology". In: *Mathematical Morphology and its Applications to Signal Processing*. 1993, pp. 245–250.

[25] Lucie Druoton, Laurent Fuchs, Lionel Garnier, and Rémi Langevin. "The Non-Degenerate Dupin Cyclides in the Space of Spheres Using Geometric Algebra". In: *Advances in Applied Clifford Algebras* 24.2 (2014), pp. 515–532. ISSN: 1661-4909. DOI: 10.1007/s00006-014-0453-5. URL: http://dx.doi.org/10.1007/s00006-014-0453-5.

[26] Juan Du, Ron Goldman, and Stephen Mann. "Modeling 3D Geometry in the Clifford Algebra $\mathbb{R}^{4,4}$". In: *Advances in Applied Clifford Algebras* 27.4 (2017), pp. 3039–3062. ISSN: 1661-4909. DOI: 10.1007/s00006-017-0798-7. URL: https://doi.org/10.1007/s00006-017-0798-7.

[27] Easter, Robert Benjamin and Hitzer, Eckhard. "Double conformal geometric algebra". In: *Advances in Applied Clifford Algebras* 27.3 (2017), pp. 2175–2199.

[28] Easter, Robert Benjamin and Hitzer, Eckhard. "Triple conformal geometric algebra for cubic plane curves". In: *Mathematical Methods in the Applied Sciences* (2017). mma.4597. ISSN: 1099-1476. DOI: 10.1002/mma.4597. URL: http://dx.doi.org/10.1002/mma.4597.

[29] Y Edel and A Klein. "Population count in arrays". In: *Submitted, available online http://cage.ugent.be/~klein/popc.html* (2009), p. 95.

[30] Ahmad Hosney Awad Eid. "Optimized Automatic Code Generation for Geometric Algebra Based Algorithms with Ray Tracing Application". In: *arXiv preprint arXiv:1607.04767* (2016).

[31] Ahmad Hosny Eid. "An Extended Implementation Framework for Geometric Algebra Operations on Systems of Coordinate Frames of Arbitrary Signature". In: *Advances in Applied Clifford Algebras* 28.1 (2018), p. 16. ISSN: 1661-4909. DOI: 10.1007/s00006-018-0827-1. URL: https://doi.org/10.1007/s00006-018-0827-1.

[32] Daniel Fontijne. "Efficient Implementation of Geometric Algebra". PhD thesis. University of Amsterdam, 2007.

[33] Daniel Fontijne. "Gaigen 2:: a geometric algebra implementation generator". In: *Proceedings of the 5th international conference on Generative programming and component engineering*. ACM. 2006, pp. 141–150.

[34] Daniel Fontijne. *Gaigen 2.5 User Manual*. https://sourceforge.net/projects/g25/.

[35] Daniel Fontijne, Leo Dorst, Tim Bouma, and Stephen Mann. "GAviewer, interactive visualization software for geometric algebra". In: *URL: http://www.geometricalgebra.net/downloads.html* (2010).

[36] Laurent Fuchs and Laurent Théry. "Implementing geometric algebra products with binary trees". In: *Advances in Applied Clifford Algebras* 24.2 (2014), pp. 589–611.

[37] Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989.

[38] Ron Goldman and Stephen Mann. "R(4, 4) As a Computational Framework for 3-Dimensional Computer Graphics". In: *Advances in Applied Clifford Algebras* 25.1 (2015), pp. 113–149. ISSN: 1661-4909. DOI: 10.1007/s00006-014-0480-2. URL: https://doi.org/10.1007/s00006-014-0480-2.

[39] Hermann Grassmann. "Die lineale Ausdehungslehre: ein neuer Zweig der Mathematik, dargestellt und durch Anwendungen auf die übrigen Zweige der Mathematik, wie auch die Statik, Mechanik, die Lehre von Magnetismus und der Krystallonomie erläutert". In: *Wigand, Leipzig* (1844).

[40] Alastair L Gregory, Joan Lasenby, and Anurag Agarwal. "The elastic theory of shells using geometric algebra". In: *Royal Society open science* 4.3 (2017), p. 170065.

[41] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[42] William Rowan Hamilton. "XI. On quaternions; or on a new system of imaginaries in algebra". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 33.219 (1848), pp. 58–60.

[43] David Hestenes. "Grassmann's vision". In: *Hermann Günther Graßmann (1809–1877): Visionary Mathematician, Scientist and Neohumanist Scholar*. Springer, 1996, pp. 243–254.

[44] David Hestenes. *New foundations for classical mechanics*. Vol. 15. Springer Science & Business Media, 2012.

[45] David Hestenes. *Oersted Medal Lecture 2002: Reforming the mathematical language of physics*. 2003.

[46] David Hestenes. *Space-time algebra*. Vol. 1. 6. Springer, 1966.

[47] David Hestenes. "The zitterbewegung interpretation of quantum mechanics". In: *Foundations of Physics* 20.10 (1990), pp. 1213–1232. ISSN: 1572-9516. DOI: 10.1007/BF01889466. URL: https://doi.org/10.1007/BF01889466.

[48] David Hestenes and Garret Sobczyk. *Clifford algebra to geometric calculus: a unified language for mathematics and physics*. Vol. 5. Springer Science & Business Media, 2012.

[49] Dietmar Hildenbrand. *Foundations of Geometric Algebra Computing*. Springer, 2013.

[50] Dietmar Hildenbrand, Werner Benger, and Yu Zhaoyuan. "Analyzing the inner product of 2 circles with Gaalop". In: *Mathematical Methods in the Applied Sciences* 41.11 (2018), pp. 4049–4062.

[51] Dietmar Hildenbrand, Christian Perwass, Leo Dorst, and Daniel Fontijne. "Geometric Algebra and its Application to Computer Graphics". In: *Eurographics 2004 - Tutorials*. Eurographics Association, 2004. DOI: 10.2312/egt.20041032.

[52] Eckhard Hitzer. "Geometric operations implemented by conformal geometric algebra neural nodes". In: *arXiv preprint arXiv:1306.1358* (2013).

[53] Eckhard Hitzer. "Relativistic physics as application of geometric algebra". In: *Proceedings of the International Conference on Relativity*. 2005, pp. 71–90.

[54] Eckhard Hitzer, Kanta Tachibana, Sven Buchholz, and Isseki Yu. "Carrier Method for the General Evaluation and Control of Pose, Molecular Conformation, Tracking, and the Like". In: *Advances in Applied Clifford Algebras* 19.2 (2009), pp. 339–364. ISSN: 1661-4909. DOI: 10.1007/s00006-009-0160-9. URL: https://doi.org/10.1007/s00006-009-0160-9.

[55] Kenichi Kanatani. *Understanding Geometric Algebra: Hamilton, Grassmann, and Clifford for Computer Vision and Graphics*. Natick, MA, USA: A. K. Peters, Ltd., 2015. ISBN: 1482259508, 9781482259506.

[56] Daniel Klawitter. "A Clifford algebraic approach to line geometry". In: *Advances in Applied Clifford Algebras* 24.3 (2014), pp. 713–736.

[57] A.N. Lasenby J. Lasenby and R.J. Wareham. *A Covariant Approach to Geometry and its Applications in Computer Graphics*. Tech. rep. Cambridge University Engineering Dept., 2002.

[58] Joan Lasenby and Adam Stevenson. "Using geometric algebra for optical motion capture". In: *Geometric Algebra with Applications in Science and Engineering*. Springer, 2001, pp. 147–169.

[59] Paul Leopardi. "A generalized FFT for Clifford algebras". In: *Bulletin of Belgian Mathematical Society* 11 (2004), pp. 663–688.

[60] Paul Leopardi. *GluCat: Generic library of universal Clifford algebra templates*. http://glucat.sourceforge.net/.

[61] Adam Lipowski and Dorota Lipowska. "Roulette-wheel selection via stochastic acceptance". In: *Physica A: Statistical Mechanics and its Applications* 391.6 (2012), pp. 2193–2196.

[62] Yang Liu, Pei Xu, Jianquan Lu, and Jinling Liang. "Global stability of Clifford-valued recurrent neural networks with time delays". In: *Nonlinear Dynamics* 84.2 (2016), pp. 767–777. ISSN: 1573-269X. DOI: 10.1007/s11071-015-2526-y. URL: https://doi.org/10.1007/s11071-015-2526-y.

[63] Wen Luo, Yong Hu, Zhaoyuan Yu, Linwang Yuan, and Guonian Lü. "A Hierarchical Representation and Computation Scheme of Arbitrary-dimensional Geometrical Primitives Based on CGA". In: *Advances in Applied Clifford Algebras* 27.3 (2017), pp. 1977–1995. ISSN: 1661-4909. DOI: 10.1007/s00006-016-0697-3. URL: https://doi.org/10.1007/s00006-016-0697-3.

[64] Stephen Mann, Leo Dorst, Tim Bouma, et al. "The making of a geometric algebra package in Matlab". In: *Univerrsity of Waterloo Research Report CS-99-27* (1999).

[65] Vesna Marinković, Predrag Janičić, and Pascal Schreck. "Computer theorem proving for verifiable solving of geometric construction problems". In: *International Workshop on Automated Deduction in Geometry*. Springer. 2014, pp. 72–93.

[66] James Clerk Maxwell. *A treatise on electricity and magnetism*. Vol. 1. Clarendon press, 1881.

[67] Hermann Minkowski. "Die Grundgleichungen für die elektromagnetischen Vorgänge in bewegten Körpern". In: *Mathematische Annalen* 68.4 (1910), pp. 472–525.

[68] Phuc Ngo, Nicolas Passat, Yukiko Kenmochi, and Isabelle Debled-Rennesson. "Geometric preservation of 2D digital objects under rigid motions". In: *Journal of Mathematical Imaging and Vision* (2018). DOI: 10.1007/s10851-018-0842-9. URL: https://hal.univ-reims.fr/hal-01695370.

[69] Margarita Papaefthymiou and George Papagiannakis. "Real-time rendering under distant illumination with conformal geometric algebra". In: *Mathematical Methods in the Applied Sciences* (2017).

[70] Spencer T Parkin. "A model for quadric surfaces using geometric algebra". In: *Unpublished, October* (2012).

[71] Spencer T Parkin. *GALua*. https://github.com/spencerparkin/GALua. Accessed: 2017-04-06.

[72] Christian Perwass. *CLUCal/CLUViz Interactive Visualization [online]*. http://www.clucalc.info/. 2010. (Visited on 2010).

[73] Christian Perwass. *Geometric algebra with applications in engineering*. Vol. 4. Geometry and Computing. Springer, 2009. ISBN: 9783540890676 354089067X.

[74] Dimiter Prodanov. "Clifford Algebra Implementation in Maxima". In: *Alterman Conference on Geometric Algebra and Summer School on Kahler Calculus* (2016).

[75] Eduardo Roa, Víctor Theoktisto, Marta Fairén, and Isabel Navazo. "GPU collision detection in conformal geometric space". In: *V Ibero-American symposium in computer graphics SIACG*. 2011, pp. 153–157.

[76] Stephen J. Sangwine and Eckhard Hitzer. "Clifford Multivector Toolbox (for MATLAB)". In: *Advances in Applied Clifford Algebras* 27.1 (2017), pp. 539–558. ISSN: 1661-4909. DOI: 10.1007/s00006-016-0666-x. URL: http://dx.doi.org/10.1007/s00006-016-0666-x.

[77] Fumiki Sekiya and Akihiro Sugimoto. "On properties of analytical approximation for discretizing 2D curves and 3D surfaces". In: *Mathematical Morphology-Theory and Applications* 2.1 (2017), pp. 25–34.

[78] Florian Seybold and U Wössner. "Gaalet-a C++ expression template library for implementing geometric algebra". In: *6th High-End Visualization Workshop*. 2010.

[79] Alexander Soiguine. "Anyons in three dimensions with geometric algebra". In: *arXiv preprint arXiv:1607.03413* (2016).

[80] Gerald Sommer. *Geometric computing with Clifford algebras: theoretical foundations and applications in computer vision and robotics*. Springer Science & Business Media, 2013.

[81] Jaap Suter. "Geometric algebra primer". In: *Self-published on personal website: http://www. jaapsuter. com/paper/ga_primer. pdf* (2003).

[82] George Tzoumas, Dominique Michelucci, and Sebti Foufou. "Extending CSG with projections: Towards formally certified geometric modeling". In: *Computer-Aided Design* 66 (2015), pp. 45–54.

[83] John Vince. *Geometric algebra for computer graphics*. Springer Science & Business Media, 2008.

[84] Linwang Yuan, Zhaoyuan Yu, Wen Luo, Jiyi Zhang, and Yong Hu. "Clifford algebra method for network expression, computation, and algorithm construction". In: *Mathematical Methods in the Applied Sciences* 37.10 (2014), pp. 1428–1435.

[85] Julio Zamora-Esquivel. "G 6,3 Geometric Algebra; Description and Implementation". In: *Advances in Applied Clifford Algebras* 24.2 (2014), pp. 493–514. ISSN: 1661-4909. DOI: 10.1007/s00006-014-0442-8. URL: https://doi.org/10.1007/s00006-014-0442-8.

[86] Shuai Zhu, Shuai Yuan, Dongshuang Li, Wen Luo, Linwang Yuan, and Zhaoyuan Yu. "MVTree for Hierarchical Network Representation Based on Geometric Algebra Subspace". In: *Advances in Applied Clifford Algebras* 28.2 (2018), p. 39. ISSN: 1661-4909. DOI: 10.1007/s00006-018-0855-x. URL: https://doi.org/10.1007/s00006-018-0855-x.