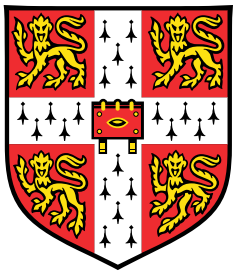
Applications of Geometric Algebra in Mathematical Engineering



Hugo Hadfield

Supervisor: Prof. Joan Lasenby

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Queens' College August 2023

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Hugo Hadfield
August 2023

Acknowledgements

I would like to start by thanking my supervisor, Joan Lasenby. Joan has dedicated huge amounts of her time to my PhD project, proofreading reams and reams of partially thought out ideas, encouraging me to explore new areas of research, and on occasion even coaching me and the SigProc PhD students in circuit training on the Trinity sports pitches. Thank you Joan for all of your help, support and kindness over the years, it really means a lot to me and to all your students.

I would like to thank all of my colleagues in the SigProc Lab and CUED for their friendship. You have all contributed hugely to making my PhD enjoyable and enriching. In particular I would like to thank Fergal Cotter, Jos Van de Westhuizen and Oliver Bonner for being so welcoming when we first joined and to Sam Duffield, Jacob Vorstrup and Alex Grafton for pushing me to improve my times on both 5K runs and the Guardian Quick crossword. Thank you to Eric Wieser for his dry wit and seemly unlimited patience when reviewing my code.

I would also like to thank my various housemates over the years, Ben Young for his patience, excellent cooking and stunning musical performances at Queens' MCR pub quizes, Jonny Morris and Beamish for taking active interests in my collection of pot plants and quilted blankets respectively, and Aoife Hannon for being the very much needed voice of reason in the house. Thank you to Jack Wilde for his impromptu musical performances and teaching me how to handle a soupy puck. I would also like to give special thanks to Tatsiana Ivonchyk, Sam Snyder and Papaya for welcoming me into their home, treating me with kindness, feeding me, encouraging me to finish my thesis and get a job, and allowing me to erect a 2500 litre paddling pool in their garden during the summer months. Thank you to Yasunori Watanabe and Anna Lamstaes for providing expert advice on both thesis writing and red wine pairings, and always being keen to barbecue on any remotely sunny day.

Thank you also to my friends from Queens'. Thank you to Luz Alonso for companionship, for taking me on adventures both around the UK and overseas, and for always laughing along with me through my attempts to learn Spanish. Thank you to Margherita Protasoni for her kindness, support and calm competence in the face of chaos. To all the members of the Queens' College Contract Whist Society I hope to see you at the Chandos one day soon.

I would like to thank the friends I have made since moving to London without whose encouragement this thesis would never have been submitted. Thank you to Ana Pervan for being the ideal desk mate, quiz team member and supportive friend. Thank you to Edgar Tamayo-Cascan and James Chappell for being my companions in exploring local maritime history, in particular the battleships of London. Thank you to Sofía Dudas for ensuring the 9-5 was never dull.

Finally I would also like to thank the Staff of Hot Numbers for making the highest quality sausage rolls available in Cambridge, your dedication to your craft is astounding and it is not exaggeration to say that much of this thesis would not have been possible without your indirect support. I feel blessed to have lived through an era when both kimchi eggs and strawberry Rocko Mountain beans were on the menu.

Abstract

Geometric Algebra (GA) has found success in various areas of the physical sciences and engineering over the last decade but remains relatively underutilised in industry and several key topics in the field remain unexplored. This thesis focuses on the practical applications of Geometric Algebra in various interconnected areas of mathematical engineering. In Part I we explore the properties of the objects resulting from the addition of blades in Conformal Geometric Algebra (CGA) and how we might use these objects in computer graphics and robotics algorithms. In Part II we explore how Screw Theory embeds into CGA, how to use this embedding for simulation of the dynamics of rigid bodies, and how practitioners can leverage the geometric primitives built into CGA to represent and solve constraints in multi-body robotic systems.

Table of contents

1 Introduction and Background **1**

1.1 Introduction 1

1.2 Basics of GA 2

1.2.1 Defining a Specific Subalgebra 3

1.2.2 The Products of GA 3

1.2.3 Reciprocal and Pseudo-reciprocal Frames 5

1.2.4 Linear Multivector Mappings 5

1.2.5 Multivector Reverse and Multivector Inverses 5

1.2.6 Rotors 6

1.3 3D CGA 6

1.3.1 Homogeneous Point Embedding 6

1.3.2 Geometric Primitives 7

1.3.3 Intersections 8

1.3.4 Transformations 8

1.3.5 Duality 8

1.4 3D PGA 9

1.4.1 Homogeneous Point Embedding 9

1.4.2 Geometric Primitives 9

1.4.3 Intersections 9

1.4.4 Transformations 9

1.4.5 Duality 10

1.5 Other GA Frameworks 10

I Computer Graphics, Computer Vision and Visualisation **11**

2 Calculating the rotor between conformal objects **13**

2.1 Introduction 13

2.2 Related Work . . . 14
2.3 Conformal Geometric Algebra . . . 14
2.4 A Rotor between Objects . . . 14
2.4.1 Lines . . . 18
2.4.2 Planes . . . 18
2.4.3 Circles . . . 19
2.4.4 Spheres . . . 19
2.4.5 Point Pairs . . . 20
2.4.6 Lines to Circles: Planes to Spheres . . . 20
2.5 The Non-Uniqueness of the Recovered Rotors . . . 20
2.6 Conclusion . . . 21

3 Direct linear interpolation of geometric objects in conformal geometric algebra 23
3.1 Introduction . . . 23
3.2 Motivation . . . 24
3.3 Linearly interpolating conformal points . . . 24
3.4 Linearly interpolating higher grade conformal objects . . . 25
3.5 Creating a blade from a pure grade multivector . . . 26
3.6 Techniques for understanding interpolant properties . . . 28
3.7 Point pairs . . . 28
3.8 Circles . . . 32
3.9 Lines . . . 35
3.9.1 Screw Theory . . . 35
3.9.2 Bivector representation of a line . . . 37
3.9.3 The bivector representation of a screw . . . 38
3.9.4 Adding dual lines . . . 39
3.9.5 Relationship to object manifold reprojection . . . 40
3.10 Planes . . . 41
3.11 Spheres . . . 42
3.12 Applications . . . 46
3.12.1 Higher order spline interpolation through objects . . . 46
3.12.2 Recursive scene simplification by averaging conformal objects . . . 46
3.12.3 *k*-means clustering of conformal objects . . . 49
3.12.4 Closest point to two non intersecting lines (least squares sense) . . . 50
3.13 Conclusions . . . 52

4 Exploring Novel Surface Representations 53
4.1 Introduction . . . 54
4.2 Conformal Geometric Algebra, CGA . . . 54
4.3 Camera Model and Ray Casting . . . 55
4.4 Ray Geometries for Basic Objects . . . 56
4.4.1 Ray-Object Intersections . . . 56
4.4.2 Extracting Normals and Reflecting Rays . . . 58
4.5 Ray Tracing Evolved Circles . . . 62
4.5.1 Intersection Point of Ray and Interpolated Surface . . . 63
4.5.2 Analytic Form for Normals . . . 67
4.6 Calculating the Derivative of the Object Manifold Projection . . . 70
4.6.1 Closed Form Derivative of the Square Root Operation . . . 71
4.6.2 Closed Form Derivative of the Projector . . . 72
4.7 Ray Tracing Evolved Point Pairs . . . 73
4.7.1 Closed Form Solution for the Intersection of a Ray and an Evolved Point-Pair Surface . . . 73
4.7.2 Bounding Sphere and Normal Calculation . . . 74
4.7.3 Special Cases of Evolved Point-Pairs . . . 75
4.7.4 Triangular Facets from Evolved Point-Pairs . . . 76
4.8 Bézier Curves and Hermite Splines through Geometric Primitives . . . 77
4.8.1 Linear Interpolation as a Linear Bézier Curve . . . 77
4.8.2 Quadratic Bézier Curve . . . 78
4.8.3 Cubic Bézier Curves . . . 78
4.8.4 N^{th} Order Bézier Curve . . . 79
4.8.5 Rational Bézier Curves . . . 79
4.8.6 Hermite Cubic Curves and Splines . . . 80
4.9 Examples of Ray Tracing Simple Objects and Evolved Surfaces . . . 81
4.10 Meshing Evolved Surfaces . . . 83
4.11 Summary and Conclusions . . . 86

5 REFORM 87
5.1 Introduction . . . 88
5.2 Proximity-based matching . . . 88
5.3 Finding the rotor between two sets of matched objects . . . 91
5.4 Iterative matching and rotor estimation . . . 92
5.5 Incorporating sampling . . . 95
5.6 Matching scenes of mixed geometric primitives . . . 99

5.7 Conclusions 99

II Kinematics, Dynamics and Robotics 101

6 Screw Theory in Geometric Algebra for Constrained Rigid Body Dynamics 103
6.1 Introduction 104
6.1.1 Screw Theory 104
6.1.2 CGA 104
6.1.3 PGA 104
6.2 Forces, moments and static equilibrium 105
6.2.1 What is a force? 105
6.2.2 Representations of wrenches in CGA and PGA 108
6.2.3 Forces as dual lines in CGA 109
6.2.4 Forces as lines in PGA 112
6.2.5 Force and moment representations in the GA literature 112
6.3 Screw transformations, instantaneous twists, and the motor manifold 114
6.3.1 Time derivatives of frame transformations 114
6.4 Momentum and inertia 116
6.4.1 Screw momentum 116
6.4.2 Mapping from screw velocity to screw momentum 116
6.4.3 The Screw Inertia Tensor 124
6.4.4 Motor Bivectors as the Principal Screws of Inertia 127
6.5 Unconstrained rigid body dynamics 129
6.6 Constrained dynamics via virtual power 131
6.7 Constrained dynamics by pinned multivectors 133
6.8 Geometric objects as constraints 135
6.8.1 Point constraint 136
6.8.2 Point-pair constraint 136
6.8.3 Direction constraint 136
6.8.4 Flat point constraint 137
6.8.5 Line constraint 137
6.8.6 Circle constraint 138
6.8.7 Plane constraint 138
6.8.8 Sphere constraint 138
6.9 Pinning parametric multivectors paths 139
6.10 Pinning linear functions of parametric multivector paths 140

6.11 Mapping Screw Velocity to Lie Algebra Velocity 140
6.11.1 Exponential Mapping and the Bortz Equation 141
6.11.2 Cayley Mapping 142
6.11.3 Outer Exponential Mapping 143
6.12 Conclusions 145

7 The Kinematics of Multi-body Systems in Geometric Algebra 147
7.1 Introduction 147
7.2 Geometric Algebra 148
7.3 Twists in Kinematic Chains 150
7.4 Geometrically Constrained Kinematic Pairs 153
7.5 The Geometry of Real Joints 154
7.5.1 Spherical Joint 154
7.5.2 Cylindrical Joint 155
7.5.3 Planar Joint 155
7.5.4 Revolute Joint 155
7.5.5 Prismatic Joint 156
7.5.6 Universal Joint 156
7.6 The Kinematic Constraint Matrix and the Jacobian Matrix 157
7.7 Case Study: The Delta Robot 158
7.7.1 Geometry of a Delta Robot 159
7.7.2 Calculating the Robot Pose 160
7.7.3 Full Geometry and Kinematic Constraint Matrix of the Delta Robot 164
7.7.4 From Constraint Matrix to Jacobian Matrices 167
7.7.5 Calculating the Jacobian with Direct Differentiation 169
7.7.6 Comparing Direct Differentiation to Screw Theory 174
7.8 Conclusions and Future Work 175

8 Conclusions 177
8.1 Main Contributions 177
8.2 Future Work 177

References 179

Chapter 1

Introduction and Background

> *Let's think the unthinkable, let's do the undoable. Let us prepare to grapple with the ineffable itself, and see if we may not eff it after all.*
>
> Douglas Adams, Dirk Gently's Holistic Detective Agency

1.1 Introduction

There are many reasons that are given in the literature for studying Geometric Algebra (GA). It is claimed to produce many elegant formulae, it is described as imbuing linear algebra with intuitive geometric meaning and it is claimed to be both computationally efficient and easy to program up. During the course of my PhD I have found many of these claims to be true but from my perspective the real benefit of the GA framework is in 'bootstrapping' the learning of multiple fields. GA is a very broadly applicable framework for expressing problems from various topics and as a result it allows a practitioner to leverage their knowledge from previous domains when coming to a new problem or field. This bootstrapping advantage is one of the main reasons I have enjoyed this PhD, with GA in my toolbox it feels like large swathes of previously inaccessible research are now within reach.

The last few years have seen somewhat of a spike in interest in GA from various disciplines but perhaps most notably from the computer graphics community. Open source GA software has advanced to a point where it is now mature and performant enough to use in real production code environments and there are growing communities of professionals and academics alike designing and implementing GA algorithms on a daily basis.

Despite enjoying the topic I think it is important to emphasise that, despite some grandiose claims in the community, GA is fundamentally one amongst several alternative tools in a mathematical toolbox. As someone new to the field, and to mathematical engineering research in general, there is a danger of myopic referencing and working siloed within the GA literature. I have tried throughout to tie this work into the broader mathematical world but the more research I have read the more my reading list has expanded and, as my supervisor has reminded me, at some point you just have to sit down and write the thesis.

This thesis has been put together from published work of the last few years. It pretty much follows the germination of one idea, namely that adding together geometric primitives in CGA can be a useful, and perhaps intuitive, thing to do. In Part I. we show how this idea came about, its relationship to rotor driven transformations between primitives, some of the properties of the addition of objects, and some of the applications in graphics, vision and visualisation. In Part II. we show how we can use the addition of lines to represent the screws of Screw Theory allowing us to construct a coordinate free basis for, and extend techniques from, that field.

Along the way specific bits of this thesis have been published as stand alone papers:

- Chapter 2 has been published in 'Advances in Applied Clifford Algebras' as 'Calculating the Rotor Between Conformal Objects' [75]
- Chapter 3 has been published in 'Advances in Applied Clifford Algebras' as 'Direct Linear Interpolation of Geometric Objects in Conformal Geometric Algebra' [48]
- Chapter 4 has been published in 'Advances in Applied Clifford Algebras' as 'Exploring Novel Surface Representations via an Experimental Ray-Tracer in CGA' [47]
- Chapter 5 has been published in 'Advances in Applied Clifford Algebras' as 'REFORM: Rotor Estimation From Object Resampling and Matching' [49]
- Chapter 6 has been submitted to 'Advances in Applied Clifford Algebras' as 'Screw Theory in Geometric Algebra for Con- strained Rigid Body Dynamics'
- Chapter 7 has been submitted to 'Advances in Applied Clifford Algebras' as 'The Kinematics of Multi-body Systems in Geometric Algebra'.

1.2 Basics of GA

Geometric Algebras, also known as Clifford Algebras, are a class of algebras that allow operations between objects of mixed grade. Here we will give a quick introduction to the

topic and state some of the required background and results that will be put to use throughout the thesis. As the literature is fairly extensive we would suggest a reader not familiar with the subject, or looking for proofs of some of the more fundamental results, to consult one of the excellent books on the topic [27][29]. As the notation used in the field varies quite widely between research groups we will also remind the reader of our notation at the start of each chapter.

1.2.1 Defining a Specific Subalgebra

Typically when working in GA we choose to work in a specific closed subalgebra. Often we will define this algebra with 3 integers p,q,r which represent the number of orthogonal basis vectors in a set which square to $+1,-1,0$ respectively. To define an algebra we then can write $Cl(p,q,r)$. This notation is often known as the *signature* of the algebra. Algebras with $r \neq 0$ are known as degenerate algebras and have to be handled in specific ways in order to work correctly. We often label the orthogonal basis vectors with numbers or symbols and ascribe specific geometric or problem specific significance to the various members of the set.

1.2.2 The Products of GA

The fundamental product of GA is the **geometric product**. The geometric product is both associative and distributive. The orthonormal basis vectors anticommute with each other and, as we have already specified what they square to, we can compute simplified forms of the products of several vectors. Take for example the algebra $Cl(1,1,0)$, consider two vectors in this space:

$$a = \alpha_1 e_1 + \alpha_2 e_2, \quad b = \beta_1 e_1 + \beta_2 e_2$$

The geometric product of these two vectors is:

$$\begin{aligned} ab &= (\alpha_1 e_1 + \alpha_2 e_2)(\beta_1 e_1 + \beta_2 e_2) \\ &= \alpha_1\beta_1 e_1 e_1 + \alpha_1\beta_2 e_1 e_2 + \alpha_2\beta_1 e_2 e_1 + \alpha_2\beta_2 e_2 e_2 \\ &= \alpha_1\beta_1 - \alpha_2\beta_2 + (\alpha_1\beta_2 - \alpha_2\beta_1) e_1 e_2 \end{aligned}$$

The result of the geometric product of two or more orthogonal vectors is known as a **blade**. We refer to the number of orthogonal vectors multiplied together to create a given blade as the **grade** of the blade. A blade made of the product of h basis vectors is often given a subscript notation:

$$A_h = \prod_{i=1}^{h} e_i$$

A linear combination of blades of different grades is known as a **multivector**. A linear combination of blades of the same grade is known as a pure grade multivector. For an algebra with N orthogonal basis vectors the highest grade blade that can be constructed is also of grade N and is known as the **pseudoscalar** of the algebra and often denoted I.

The second most fundamental operation on a multivector in GA after the geometric product is **grade selection** (also known as grade projection). Grade selection returns only the part of the multivector of a given grade. We often use the $\langle X \rangle_h$ notation to denote selecting grade h from multivector X. With this notation we can define grade selection by considering a multivector X defined as the sum of blades of several grades with scalar coefficients, $X = \sum_g \lambda_g A_g$:

$$\langle X \rangle_h = \langle \sum_g \lambda_g A_g \rangle_h = \lambda_h A_h$$

All other products are defined in terms of the geometric product and grade selection. In this thesis we will refer to the inner product which we will represent by $\cdot$, the outer product that we will represent with $\wedge$, and the commutator product that we will represent with $\times$. The inner product that we will use throughout this thesis is defined as follows:

$$\begin{cases} A_g \cdot B_h = 0 & g = 0 \\ A_g \cdot B_h = 0 & h = 0 \\ A_g \cdot B_h = \langle A_g B_h \rangle_{|g-h|} & \text{otherwise} \end{cases}$$

Note that the choice of the inner product with scalars being 0 or not varies among researchers and software systems. We have chosen this inner product to match the one used in the 'clifford' python package [52]. The outer product has the following standard definition:

$$\begin{cases} A_g \wedge B_h = 0 & g + h > p + q + r \\ A_g \wedge B_h = \langle A_g B_h \rangle_{g+h} & \text{otherwise} \end{cases}$$

In the same way we use the shorthand $\prod_i X_i$ to imply the geometric product of several elements so we use the notation $\bigwedge_i X_i$ to imply the wedge product of several multivectors. The commutator product is typically denoted with a cross, $\times$, and is defined in terms of the geometric product:

$$A \times B = \frac{1}{2}(AB - BA)$$

For more information about these products see Doran and Lasenby [27].

1.2.3 Reciprocal and Pseudo-reciprocal Frames

A particularly useful tool for constructing proofs in GA is the concept of reciprocal frames. Given an ordered set of multivectors, X_i, we define a reciprocal frame of multivectors, X^i, which have the property $X_i \cdot X^i = 1$, for all i. When dealing with degenerate metric algebras (those with $r \neq 0$ in the signature) however we have to use a slightly different construction, the pseudo-reciprocal frame [43]. A pseudo-reciprocal frame P^i has the following property $X_i \wedge P^i = I$, for all i, where I is the pseudoscalar of the algebra. The concept of reciprocal and pseudo-reciprocal frames will come up many times throughout this thesis.

1.2.4 Linear Multivector Mappings

One of the most commonly used types of mapping of a multivector is a linear mapping (also known as a linear function). Linear mappings are exceptionally useful in applied mathematics and are typically represented as matrices with a fixed basis. GA by default does not define a fixed basis for its multivectors, it does however define products that act in a coordinate-free way. If we have a linear mapping of a multivector F we can represent this mapping as a matrix of real coefficients c_{ij} acting on a vector of multivector coefficients for a given basis X_i by the following construction:

$$c_{ij} = F(X_i) \cdot X^j$$

For a degenerate metric algebra we can do the same with a pseudo-reciprocal frame, followed by the selection of the coefficient of the pseudoscalar:

$$c_{ij} I = F(X_i) \wedge P^j$$

1.2.5 Multivector Reverse and Multivector Inverses

The reverse is one of the fundamental operations of GA. The reverse of a multivector A is defined as:

$$\tilde{A} = (\sum_g A_g)\tilde{} = \sum_g (-1)^{g(g-1)/2} A_g$$

Like many operations in GA the reverse is linear and thus distributes over addition:

$$(A+B)\tilde{} = (\sum_g (A_g + B_g))\tilde{} = \tilde{A} + \tilde{B}$$

A topic that has been the subject of much study in recent years is multivector inverses. The goal of multivector inversion is this: for a given multivector, A, to find an multivector, B, such that $AB = 1$, we would then describe B as the inverse of A. Not all multivectors are invertible, however for those that are we can invert them with the matrix based techniques of Perwass [81], the specific closed form solutions of Hitzer [61] or the general closed form solutions of Shirokov [92]. For invertible multivectors in low dimensional algebras these techniques should all give the same results but for the sake of clarity we will state that unless explictly mentioned we use the inverse as described by Shirokov in [92].

1.2.6 Rotors

An aspect of the GA framework that has found widespread use is the inclusion of geometrical transformations as elements of the algebra. The primary mechanism by which we represent geometrical transformations are with a class of multivectors known as **rotors**.

A rotor, often denoted R, in a specific subalgebra can be constructed by the geometric product of an even number of unit 1-vectors from that subalgebra. Whatever signature algebra you use rotors always have certain properties, namely $R\tilde{R} = 1$ and R is made up of a sum of strictly even grade elements. To perform a transformation with a rotor we sandwich a vector, X, with a rotor and the reverse of that rotor.

$$Y = RX\tilde{R}$$

Rotor application is grade preserving and can be extended over the outer product producing the result [27]:

$$\bigwedge_i (RA_i\tilde{R}) = R(\bigwedge_i A_i)\tilde{R}$$

The signature of the algebra determines what types of geometric transformations are available in the rotor group of that algebra.

1.3 3D CGA

1.3.1 Homogeneous Point Embedding

The Conformal Geometric Algebra (CGA) was first described by Hestenes in [56] and has been one of the most popular GA frameworks for solving practical problems ever since. CGA has a signature of $Cl(4,1,0)$, three basis vectors e_1, e_2 and e_3 that square to $+1$ and an additional pair of basis vectors e_+ and e_- that square to $+1$ and -1 respectively. These

additional basis vectors are used to form two null vectors $n_\infty = e_+ + e_-$ and $n_0 = \frac{1}{2}(e_- - e_+)$. These null vectors are then used to define a point embedding via the formula:

$$X = \frac{1}{2}x^2 n_\infty + x + n_0$$

which is sometimes referred to as $X = \text{up}(x)$. An important consequence of this specific point embedding is for a given point X:

$$X^2 = 0$$

and more generally for a point $X = \frac{1}{2}x^2 n_\infty + x + n_0$ and point $Y = \frac{1}{2}y^2 n_\infty + y + n_0$ we get the result:

$$XY = -\frac{1}{2}(x-y)^2$$

The embedding is a homogeneous representation of the point, ie. X and λX for scalar λ both represent the same 3D point. To invert the mapping we therefore first have to normalise the point X such that:

$$\hat{X} = -\frac{X}{X \cdot n_\infty}$$

we can then extract the components of x via the dot product:

$$x = \sum_{i=1}^{3} (\hat{X} \cdot e_i) e_i$$

or via:

$$x = (\hat{X} \wedge E_0) E_0$$

where $E_0 = n_\infty \wedge n_0$. The mapping from a conformal point back to the 3D point it represents is sometimes referred to as $\text{down}(X)$.

1.3.2 Geometric Primitives

One of the primary motivations for shifting to a higher dimensional algebra to represent 3D space is that it allows us to represent a richer set of possible entities. Some of the most immediately useful objects that CGA allows us to represent are the geometric primitives, these primitives are represented by the blades of the algebra and can be constructed with the outer product of several points.

These geometric primitives can also be parameterised in their dual form with the classic parameters that we would associate with a geometric primitive of that type. We will introduce

these primitives for the specific 3D CGA in more detail as we work our way through the thesis but here we will simply mention that they come in two types, primitives formed from the outer product with n_∞, sometimes known as 'flats', and those without n_∞ sometimes known as 'rounds' [29].

1.3.3 Intersections

In many contexts where we are working with geometric primitives we are interested in computing their intersections. In CGA we can use the meet to compute intersections; the meet is represented by the $\vee$ operator. The meet is taken with respect to a subspace, depending on the entities involved and the context in which this meet operation takes place, this subspace is either the full space or the smallest possible subspace that contains the operands of the meet operation. In this thesis we will state explicitly in which subspace the meet takes place.

1.3.4 Transformations

The rotor driven transformations of CGA include all conformal transformations. The full group of conformal transformations has 8 degrees of freedom, but many applications stick to using only the 6 degree of freedom rigid body motion subgroup. Additionally the primitives of CGA support reflection/inversion via the double sided formula:

$$X' = PXP$$

where P is a given blade. The reflection operations are practically useful in many situations and particularly find application in computer graphics.

1.3.5 Duality

The duality operation in CGA is performed by multiplication with the pseudoscalar. In non-degenerate metric algebras such as CGA primitives behave reasonably under rotor transformations whether represented dually or directly, and as such the idea of privileging one representation over another makes very little sense. In practice we tend to use one representation over another when it fits the way in which we choose to parameterise a problem. In this thesis we will switch between dual and direct primitive representations fairly fluidly throughout.

1.4 3D PGA

1.4.1 Homogeneous Point Embedding

The 3D Plane-based (or Projective) Geometric Algebra (PGA), like the 3D CGA uses a homogeneous embedding, representing points as trivectors. PGA has a signature of $Cl(3,0,1)$. The specific embedding used in 3D PGA is:

$$X = J(x + e_0)$$

where $e_i e_i = 1 \forall i \in 1,2,3$ and $e_0 e_0 = 0$. The inverse mapping is:

$$x = J\left(\frac{-X}{\langle X \cdot I_3 \rangle_0} - I_3\right)$$

where J represents the (linear) duality operation of PGA that we will describe shortly.

PGA is an algebra designed to model only the Euclidean motions and has a point embedding that is familiar to graphics programmers. As we will see later in the thesis this point embedding is identical in fact to the embedding of 3D points as 'flat points' in CGA. One of the advantages of embedding points in this way is that normalised points can be interpolated linearly without correction terms.

1.4.2 Geometric Primitives

PGA is a subalgebra of CGA, it is in fact the subalgebra of the flat elements of CGA. This means that it contains points, lines and planes as primitives which have grades 3, 2 and 1 respectively.

1.4.3 Intersections

Intersections in PGA can again be computed with the same collection of operators that we use in CGA. PGA however is a more 'opinionated' algebra, it chooses a default space for its geometric primitives and the intersection between these primitives are always computed with the outer product.

1.4.4 Transformations

The bivectors of PGA are exactly the Euclidean motion generators and as such the rotor driven transformations in PGA are the rigid body motions. This is generally a useful thing,

one of the advantages of PGA is its simplicity; if all you need is flat objects and the Euclidean group there is not much point creating the whole conformal group and then just playing with part of it.

1.4.5 Duality

Duality in degenerate metric algebras is a tricky business. As the pseudoscalar now squares to 0 it is not possible to dualise simply by multiplication with it. Instead we use the concept of pseudo-reciprocal frames to calculate an alternative dual formulation.

For a given basis vector of PGA, X_i, we will select a pseudo-reciprocal frame, X^i, such that $X_i \wedge X^i = I$. We can then construct a linear operator J in the following form:

$$J(A) = \sum_i \langle A \wedge X^i \rangle_I X^i$$

where we use the notation $\langle\,\rangle_I$ to mean taking the coefficient of the pseudoscalar. This linear operator acts as a dual for PGA and, especially when dealing with lines, is a convenient and useful construction.

This dual does however produce one of the more interesting issues with the PGA; in general the dual of the transformation of a primitive by a rotor is not the same as the transformation of the dual of the primitive, ie. $RP^*\tilde{R} \neq (RP\tilde{R})^*$. In general this leads to a selection of one specific space as the primary space for primitves in the PGA, normally the so called plane-based representation of primitves where points are 3-vectors.

1.5 Other GA Frameworks

Several other GA frameworks have been studied over the last few years, and some of them will be mentioned in this thesis. Many of these constructions have been designed to extend CGA in some way to include more geometric primitives and transformations, most notably there has been a push to include higher dimensional primitives such as conics [64] and quadrics [12, 33] as well as a goal of representing richer groups of transformations [31].

Part I
Computer Graphics, Computer Vision and Visualisation

In this Part of the Thesis we develop techniques that may find application in the fields of graphics, vision and visualisation. Graphics, vision and visualisation have long been a promising application area for geometric algebra and indeed much of the present interest in the field is driven from the perspective of computer graphics. Here we link the generation of rotors directly from geometric objects to the idea of direct addition of geometric primitives in CGA. We explore applications of addition of CGA objects in 3D computer vision scenarios and then extend the idea of addition of objects to interpolation and extrapolation and construct and visualise evolved splines and surfaces.

Chapter 2

Calculating the rotor between conformal objects

We shape our tools, and thereafter our tools shape us.

John M. Culkin

2.1 Introduction

In this chapter we will address the problem of recovering covariant transformations between objects – specifically; lines, planes, circles, spheres and point pairs. Using the covariant language of conformal geometric algebra (CGA), we will derive such transformations in a very simple manner. In CGA, rotations, translations, dilations and inversions can be written as a single *rotor*, which is itself an element of the algebra. We will show that the rotor which takes a line to a line (or plane to a plane etc.) can easily be formed and we will investigate the nature of the rotors formed in this way.

If we can recover the rotor between one object and another of the same type, a useable metric which tells us how close one line (plane etc.) is to another, can be a function of how close this rotor is to the identity. Using these ideas, we find that we can define metrics for a number of common problems, specifically recovering the transformation between sets of noisy objects.

2.2 Related Work

Our primary aim in this chapter is to simultaneously estimate the *rotation* and *translation* that takes one object (line to line/circle to circle/plane to plane/sphere to sphere/point-pair to point-pair) to another. There are many methods that estimate rigid body transformations with points [34][98][95][24]. In [62] the authors estimate a general rotor between arbitrary objects using the idea of *carriers* – while interesting, this method lacks simplicity and does not deal directly with the objects themselves.

2.3 Conformal Geometric Algebra

The objects we work with here will be CGA objects unless explicitly stated otherwise. We will use the standard extension of the 3D geometric algebra, where our 5D CGA space is made up of the standard spatial basis vectors $\{e_i\}$ $i = 1,2,3$, plus two additional basis vectors, e and $\bar{e}$ with signatures, $e^2 = 1$, $\bar{e}^2 = -1$. Two *null vectors* can then be defined as: $n_\infty = e + \bar{e}$ and $n_0 = \frac{e-\bar{e}}{2}$. The mapping of a 3D vector x to its conformal representation X is given by $X = F(x) = \frac{1}{2}(x^2 n_\infty + 2x - 2n_0)$. Many of our target applications will be in computer vision, and in investigating algorithms which use more than just points, which is the case with most conventional computer vision algorithms.

2.4 A Rotor between Objects

Suppose we wish to find the rotor (rotation, translation, dilation) which takes an object X_1 to an object X_2 (where X_1 and X_2 are conformal n-blades representing the lines/circles/planes/spheres/point pairs). If we firstly take lines as an example, conventionally we would translate along the common perpendicular and then rotate about the intersection point – which requires a series of non-trivial geometric operations for two arbitrary lines in space. Here we seek a method which will not require reverting to the geometric properties of the lines, but which will give the transformation in terms of the lines themselves – and we wish this method to be valid for all objects. In CGA, let the rotor which takes X_1 to X_2 be R_x, where this comprises both rotation, translation and dilation rotors. We assume both objects are normalised such that $X_1^2 = X_2^2 = \gamma$, where $\gamma = 1$ for lines, circles and point pairs, and $\gamma = -1$ for planes and spheres:

$$X_2 = R_x X_1 \tilde{R}_x$$

Note, that $\tilde{X} = -\gamma X$. We motivate our approach by considering the quantity $(X_1 + X_2)$ which is in some sense the '*average*' object; ie, if we reflect X_1 in $(X_1 + X_2)$, we should get some function of X_2 (we assume for convenience that $X^2 = 1$, ie $\gamma = 1$):

$$\begin{aligned}(X_1 + X_2)X_1(X_1 + X_2) &= (1 + X_2X_1)(X_1 + X_2) \\ &= [2 + (X_1X_2 + X_2X_1)]X_2 \equiv KX_2 \end{aligned} \tag{2.1}$$

So the reflection does indeed produce a multiple, though the multiple is a *scalar plus 4-vector*, of X_2. Since we can write the LHS of equation 2.1 as

$$(X_1 + X_2)X_1(X_1 + X_2) = [(X_1 + X_2)X_1]X_1[X_1(X_1 + X_2)] = (1 + X_2X_1)X_1(1 + X_2X_1)\tilde{}$$

we propose to use the spinor quantity $Z = 1 + \gamma X_2X_1$ to form R_x. As above (but now with γ included) $ZX_1\tilde{Z}$ gives ;

$$Y = ZX_1\tilde{Z} = 2X_2 + \gamma(X_1X_2 + X_2X_1)X_2 = (2 + \gamma M_{12})X_2 = KX_2 \tag{2.2}$$

where $M_{12} = X_1X_2 + X_2X_1$ is the anticommutator of X_1 and X_2. Thus, we see that Z *takes* X_1 *to a multiple of* X_2, where this multiple involves the *anticommutator of the objects*. In general this anticommutator will have scalar and 4-vector parts (the bivector part of X_1X_2 cancels with the bivector part of X_2X_1).

Since all 4-vectors square to give a scalar, we can take $K^* = \langle K\rangle_0 - \langle K\rangle_4$, such that $KK^* = \langle K\rangle_0^2 - \langle K\rangle_4^2$, is a scalar, which we call μ. We show later that μ is always positive. We now multiply both sides of equation 2.2 by K^* to give:

$$\frac{1}{\mu}K^*ZX_1\tilde{Z} = X_2$$

We now look to split up K^* such that $S^2 = K^*$, where $S = \alpha + \beta M_{12} \equiv (\alpha + \beta\langle M_{12}\rangle_0) + \beta\langle M_{12}\rangle_4$ and α and β are scalars. If S takes this form, it is clear that it is both self-reverse and commutes with Z and X_1; we can therefore write

$$\left(\frac{1}{\sqrt{\mu}}SZ\right)X_1\left(\frac{1}{\sqrt{\mu}}SZ\right)^{\sim} = X_2$$

so that $\frac{1}{\sqrt{\mu}}SZ$ is our required rotor and $\mu = K^*K$. To find such an S we can use the square root formula given in [30] or simply equate scalar and 4-vector parts of the equation $S^2 = K^*$. We do the latter first in order to see how the particular form of our *scalar plus 4-vector* behaves and then confirm that it agrees with the formula in [30]:

$$(\alpha + \beta\langle M_{12}\rangle_0)^2 + 2\beta(\alpha + \beta\langle M_{12}\rangle_0)\langle M_{12}\rangle_4 + \beta^2\langle M_{12}\rangle_4^2 = \langle K\rangle_0 - \langle K\rangle_4$$

Since $\langle K\rangle_0 = 2 + \gamma\langle M_{12}\rangle_0$ and $\langle K\rangle_4 = \gamma\langle M_{12}\rangle_4$, we have:

$$\begin{aligned}(\alpha + \beta\langle M_{12}\rangle_0)^2 + \beta^2\langle M_{12}\rangle_4^2 &= \langle K\rangle_0 \\ 2\beta(\alpha + \beta\langle M_{12}\rangle_0)\langle M_{12}\rangle_4 &= -\gamma\langle M_{12}\rangle_4\end{aligned}$$

From equating 4-vector parts we see that $2\beta(\alpha + \beta\langle M_{12}\rangle_0) = -\gamma$ so that, provided $\langle M_{12}\rangle_4 \neq 0$;

$$S = -\frac{\gamma}{2\beta} + \beta\langle M_{12}\rangle_4$$

If $\langle M_{12}\rangle_4 = 0$ we simply have $S = \sqrt{\langle K\rangle_0}$ if $\langle K\rangle_0$ is positive, which it is for lines, planes, circles and point pairs. $\langle K\rangle_0$ can take negative values for some sphere cases. If $\langle M_{12}\rangle_4 \neq 0$ we then find β from the equation which equates scalar parts:

$$\frac{1}{4\beta^2} - \beta^2\lambda = \langle K\rangle_0$$

where $\langle M_{12}\rangle_4^2 \equiv \langle K\rangle_4^2 = -\lambda$, since the 4-vectors always square to give zero or a negative scalar. This is a quadratic in $u = \beta^2$:

$$4\lambda u^2 + 4\langle K\rangle_0 u - 1 = 0 \tag{2.3}$$

with solutions given by:

$$u = \frac{-4\langle K\rangle_0 \pm 4\sqrt{\langle K\rangle_0^2 + \lambda}}{8\lambda}$$

As $\beta^2 = u$ we need the solution which is guaranteed to be positive:

$$\beta^2 = \frac{1}{2\lambda}\left(\sqrt{\langle K\rangle_0^2 + \lambda} - \langle K\rangle_0\right) = \frac{1}{2\lambda}(\sqrt{\mu} - \langle K\rangle_0)$$

Recall $K = 2 + \gamma(X_1X_2 + X_2X_1) = \langle K\rangle_0 + \langle K\rangle_4$, $K^* = \langle K\rangle_0 - \langle K\rangle_4$, $\lambda = -\langle K\rangle_4^2$, $\mu = K^*K = \langle K\rangle_0^2 + \lambda$, so is always positive (as $\lambda \geq 0$). We can now write the explicit form of the rotor as:

1. If $\langle M_{12}\rangle_4 \neq 0$:

$$\boxed{R_x = \frac{1}{\sqrt{\mu}}\left(-\frac{1}{2\beta} + \beta\langle K\rangle_4\right)(1 + \gamma X_2X_1)} \tag{2.4}$$

$$\boxed{\beta^2 = \frac{1}{2\left(\sqrt{\mu} + \langle K\rangle_0\right)}} \tag{2.5}$$

2. If $\langle M_{12}\rangle_4 = 0$ and $\langle K\rangle_0 > 0$

$$\boxed{R_x = \frac{1}{\sqrt{\langle K\rangle_0}}(1 + \gamma X_2X_1)} \tag{2.6}$$

3. If $\langle M_{12}\rangle_4 = 0$ and $\langle K\rangle_0 < 0$,

$$\boxed{R_x = \frac{1}{\sqrt{|\langle K'\rangle_0|}}(1 + \gamma \bar{X}_2X_1)} \tag{2.7}$$

where $\bar{X}_2 = -X_2$ and $K' = 2 + \gamma(X_1\bar{X}_2 + \bar{X}_2X_1)$.

Taking the positive or negative square root for β simply changes the sign of the rotor, which makes no difference to the transformation. These expressions hold for all CGA objects: lines, planes, circles, spheres, point pairs. The following subsection will give the explicit forms for each of these objects and will discuss the third case which can occur for spheres.

Before looking in more detail at the nature of the rotors formed by the process outlined here, we return to equation 2.1 and note that we can now take X_1 to X_2 via a **reflection** in the quantity X_m where

$$X_m = \frac{S}{\sqrt{\mu}}(X_1 + X_2)$$

where S and μ are as given previously, ie $\mu = K^*K$ and S takes the form in equations 2.4,2.6,2.7 depending on the nature of M_{12}. We will see in Chapter 3 that the quantity $\frac{S}{\sqrt{\mu}}$ projects the m-vector obtained from the addition of the two blades X_1 and X_2 onto an m-blade and therefore an object – the object being that in which we reflect X_1 in to get X_2.

We can also confirm the solutions in equations 2.4,2.6,2.7 using the result in [30], where the square root of the *scalar plus 4-vector*, Σ, is given by

$$\sqrt{\Sigma} = \frac{\Sigma \pm [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle \pm [[\Sigma]]}} = \frac{\langle\Sigma\rangle \pm [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle \pm [[\Sigma]]}} + \frac{\langle\Sigma\rangle_4}{\sqrt{2}\sqrt{\langle\Sigma\rangle \pm [[\Sigma]]}}$$

where $[[\Sigma]] = \sqrt{\langle\Sigma\rangle^2 - \langle\Sigma\rangle_4^2}$. Here, our $\Sigma = K^* = \langle K\rangle_0 - \langle K\rangle_4$, so that (taking the solution corresponding to the $+$ sign):

$$[[\Sigma]] = \sqrt{\mu} \quad \text{and} \quad \sqrt{2}\sqrt{\langle\Sigma\rangle + [[\Sigma]]} = \pm\frac{1}{\beta}$$

giving $\sqrt{\Sigma} = -\frac{1}{2\beta} + \beta\langle K\rangle_4$, as required (taking $-\frac{1}{\beta}$).

2.4.1 Lines

Conformal lines take the form $L = A \wedge B \wedge n_\infty$, with A, B being the conformal representations of two points lying on the line, and n_∞ the point at infinity. $\tilde{L} = -L$ and we normalise such that $L^2 = 1$, therefore $\gamma = 1$. For lines, the 4 vector part of the anticommutator takes the form $\beta I_5 n_\infty \equiv \beta I_3 n_\infty$, thus the square of this is always zero, which means $\lambda = 0$ and $\mu = \langle K\rangle_0^2$, which reduces equation 2.3 to $u = 1/(4\langle K\rangle_0)$ and $\beta = \pm 1/(2\sqrt{\langle K\rangle_0}$ [note that it does not matter which sign we take], giving us the simpler form of the rotor as:

$$\boxed{R = \frac{1}{\langle K\rangle_0}\left(\sqrt{\langle K\rangle_0} - \frac{\langle K\rangle_4}{2\sqrt{\langle K\rangle_0}}\right)(1 + L_2L_1) \equiv \frac{1}{\sqrt{\langle K\rangle_0}}\left(1 - \frac{\langle K\rangle_4}{2\langle K\rangle_0}\right)(1 + L_2L_1)} \quad (2.8)$$

2.4.2 Planes

With planes, as with lines, there is no issue of scaling as the objects are infinite. A plane Π is taken to be the conformal 4-blade of the form $A \wedge B \wedge C \wedge n_\infty$, with A, B, C any 3 conformal points lying on the plane. Conformal planes square to a negative number, so we assume that planes are normalised such that $\Pi^2 = -1$, therefore $\gamma = -1$. Note that $\tilde{\Pi} = \Pi$.

For planes the anticommutator is a scalar and it is not hard to show that (for normalised planes) $\langle K\rangle_0$ is always positive. Thus, the form for the rotor in the plane-to-plane case is particularly simple as the $\langle K\rangle_4$ term vanishes:

$$\boxed{R_\Pi = \frac{1}{\sqrt{\langle K\rangle_0}}(1-\Pi_2\Pi_1)} \tag{2.9}$$

where $K = 2-(\Pi_1\Pi_2+\Pi_2\Pi_1)$.

2.4.3 Circles

One might think that the case of circles-to-circles would be more complex, as a transformation which takes one arbitrary circle to another involves a dilation as well as a rotation and translation. However, nothing in the above derivation assumed anything specific about the rotor, and we find that we can use precisely the same formula to move between arbitrary circles.

Let us start with two conformal circles, C_1 and C_2 not necessarily of the same radius. A conformal circle is a 3-blade of the form $P\wedge Q\wedge R$, where P,Q,R lie on the circle. Circles square to a positive scalar, so we will assume that our circles are normalised such that $C^2 = 1$ and therefore $\gamma = 1$. Note that $\tilde{C} = -C$.

The anticommutator, M_{12}, is in general a *scalar plus 4-vector*, so we must use the form given in equations 2.4,2.5 and little simplification is possible:

$$\boxed{R_c = \frac{1}{\sqrt{\mu}}\left(-\frac{1}{2\beta}+\beta\langle K\rangle_4\right)(1+C_2C_1)} \tag{2.10}$$

$$\boxed{\beta^2 = \frac{1}{2\lambda}(\sqrt{\mu}-\langle K\rangle_0)} \tag{2.11}$$

with $K = 2+(C_1C_2+C_2C_1)$, $\mu = K^*K$.

2.4.4 Spheres

We start with two conformal spheres, S_1 and S_2 not necessarily of the same radius. A conformal sphere is a 4-blade of the form $N\wedge P\wedge Q\wedge R$, where N,P,Q,R lie on the sphere. Circles square to a negative scalar, so we will assume that our spheres are normalised such that $S^2 = -1$ and therefore $\gamma = -1$. Note that $\tilde{S} = S$.

As for planes, $\langle K\rangle_4^2$ is zero, so the rotor takes a very simple form:

$$\boxed{R_s = \frac{1}{\sqrt{|\langle K\rangle_0|}}(1-\bar{S}_2S_1)} \tag{2.12}$$

where $K = 2 - (S_1\bar{S}_2 + \bar{S}_2 S_1)$, $\bar{S}_2 = S_2$ if $\langle K \rangle_0 > 0$ and $\bar{S}_2 = -S_2$ if $\langle K \rangle_0 < 0$. $-S_2$ is the *same sphere* as S_2, so in a sense it does not matter whether we take S_1 to S_2 or to $-S_2$ – this additional complexity occurs with spheres as they lack any intrinsic orientation, which is not the case for lines, planes, circles and point pairs.

2.4.5 Point Pairs

In the conformal setting, point pairs take the form $A \wedge B$ where A, B are conformal points – we can think of a point pair as a line segment. For a point pair, P, clearly $\tilde{P} = -P$ and P^2 gives a positive scalar. We will therefore assume that point pairs are normalised so that $P^2 = 1$.

Since the anticommutator will generally have both scalar and 4-vector parts, we again have the general form taken from equations 2.4,2.5:

$$\boxed{R_p = \frac{1}{\sqrt{\mu}}\left(-\frac{1}{2\beta} + \beta\langle K \rangle_4\right)(1 + P_2 P_1)} \tag{2.13}$$

$$\boxed{\beta^2 = \frac{1}{2\lambda}\left(\sqrt{\mu} - \langle K \rangle_0\right)} \tag{2.14}$$

with $K = 2 + (P_1 P_2 + P_2 P_1)$, $\mu = K^* K$.

2.4.6 Lines to Circles: Planes to Spheres

Note that in the previous rotor derivation we assumed X_1 and X_2 were blades of the same grade, but nothing further. Therefore, we should, and indeed do, find that the rotor formulae in equations 2.4-2.7 work for moving between *lines and circles* and between *planes and spheres*.

2.5 The Non-Uniqueness of the Recovered Rotors

Although we have recovered rotors for each case of lines, planes, circles, spheres and point pairs, it is clear that these rotors are not unique. For example, if we transform one line into another, we can then translate along the second line without altering the result. So, a natural question to ask is *exactly what is the transformation we are recovering with the* $1 + X_2 X_1$ *expression*.

To investigate this further we extract the bivector, B, for each recovered rotor, with $R = e^B$, and plot the interpolated objects for each of λ_i, $i = 1,..n$, with $X_i = e^{\lambda_i B} X_1 e^{-\lambda_i B} M_0$, where $X_2 = RX_1\tilde{R}$ and $\lambda_i = i/n$. Figure 2.1 shows these interpolations for each class of object.

2.6 Conclusion

In this chapter we have presented a general framework for extracting the conformal rotor that takes a conformal object of a given grade to another conformal object of the same grade. The technique works for point pairs, lines, circles, planes and spheres. In the process of investigating these rotors we have touched on the form of the object required to reflect one object into another and by visualising intermediate objects we have verified that the rotors take the objects smoothly to each other. Code that implements this rotor extraction algorithm is available in the clifford [52] python package and novel applications of this technique are additionally presented in [35] [48] and [49]. It is also interesting to note that the nature of the quantity X_2X_1 was investigated first in [76], and then in [29], and noted to produce a quantity which was R^2, where R is the rotor taking X_1 to X_2. This has also been used for interpolations between objects in [18]. Here we have given explicit expressions for the rotor itself and investigated the a range of use cases.

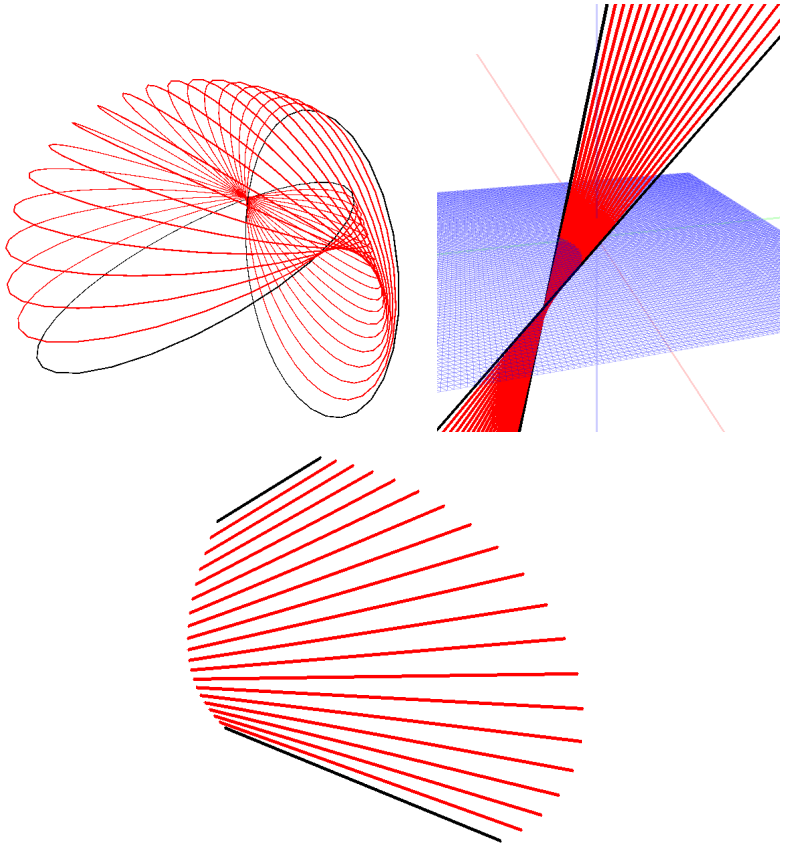


Fig. 2.1 The red objects in each of these images show the interpolations of the rotors formed from pairs of black objects. Here we see that a range of geometric primitives including circles, lines and point pairs are all handled elegantly by the same framework.

Chapter 3

Direct linear interpolation of geometric objects in conformal geometric algebra

I write rhymes with addition and algebra, mental geometry.

Ice T, Mind Over Matter

Abstract

Typically we do not add objects in conformal geometric algebra (CGA), rather we apply operations that preserve grade, usually via rotors, such as rotation, translation, dilation, or via reflection and inversion. However, here we show that direct linear interpolation of conformal geometric objects can be both intuitive and of practical use. We present a method that generates useful interpolations of point pairs, lines, circles, planes and spheres and describe algorithms and proofs of interest for computer vision applications that use this direct averaging of geometric objects.

3.1 Introduction

In this chapter we will look at **adding** CGA objects and adjusting the resulting multivectors to produce *useful* interpolations of the objects. We will present a general technique that is **valid for all geometric objects of grade 2 or above**. This technique uses the decompositions presented in [30].

The objects we work with here will be CGA objects unless explicitly stated otherwise. We will use the standard extension of the 3d geometric algebra, where our 5D CGA space

is made up of the standard spatial basis vectors $\{e_i\}$ $i = 1,2,3$, plus two additional basis vectors, e and $\bar{e}$ with signatures, $e^2 = 1$, $\bar{e}^2 = -1$. Two *null vectors* can therefore be defined as: $n_\infty = e + \bar{e}$ and $n_0 = \frac{e-\bar{e}}{2}$. The mapping of a 3d vector x to its conformal representation X is given by $X = F(x) = \frac{1}{2}(x^2 n_\infty + 2x - 2n_0)$. [1]

3.2 Motivation

In our conformal representation of space the blades of a given grade n represent specific classes of object and lie on a manifold within the overall subspace of grade n. Typically in geometric algebra we traverse this manifold using rotors and reflections. These transformations are incredibly useful and make up the vast majority of operations used in the field of applied geometric algebra. Unfortunately, while being of geometric significance, rotors and reflections are often unintuitive ways of thinking about a problem and traditional algorithms often require significant rehashing to fit within this framework.

For example: Given a cluster of geometric objects we would like to be able to create an 'average' object that lies in some sense in the middle of the bundle. Most geometric algebra approaches to this problem would likely require the explicit design of a geometrically motivated cost function followed by constrained optimisation on the blade manifold either directly or via a parametrisation of rotors over the space.

While this approach has been very effective for a variety of problems, it requires the careful crafting of clever cost functions, consideration of the convexity of the underlying space, efficient implementation of the given optimisation scheme etc. The question we aim to answer here is: what if we just decided to add all the objects together?

3.3 Linearly interpolating conformal points

The result of linear combinations of conformal points is well known [76]. Consider two arbitrary points in 3d space a and b represented as A and B in our conformal model. Linear interpolation of these points followed by our conformal embedding produces a linear interpolation of our conformal points with an additional term in n_∞:

$$F(\alpha a + (1-\alpha)b) = \alpha A + (1-\alpha)B + \alpha(1-\alpha)(A \cdot B)n_\infty \tag{3.1}$$

[1] There are a range of different notations in use in the literature to define the basis of CGA: e is sometimes referred to in other works as e_+, $\bar{e}$ as e_-, n_∞ as e_∞ and an e_o is sometimes defined which is equal to $-n_0$

We can therefore get a useful interpolation of points by taking a direct linear interpolation and simply adding the final $\alpha(1-\alpha)(A\cdot B)n_\infty$ term to the result. If $Y=\alpha A+(1-\alpha)B$, we can recover $Y'=F(\alpha a+(1-\alpha)b)$ via the following formula (assuming $Y'\cdot n_\infty=-1$):

$$Y'=\frac{-Yn_\infty Y}{2(Y\cdot n_\infty)^2}$$

3.4 Linearly interpolating higher grade conformal objects

Objects of grade 2 and above are more difficult to interpolate in a sensible and computationally efficient way. Typically, schemes that have been found are either only valid for certain objects in specific cases [26], or the problem is attacked indirectly via carriers [62] or by forming the rotor between the objects, extracting the corresponding bivector, which is then interpolated [99] and applied to the first object.

It was shown in Chapter 2 that we can represent the mirror object X_m that reflects one object X_1, into another X_2, as the left multiplication of the summation of the blades by a *scalar + 4-vector* factor S:

$$X_m=S(X_1+X_2)=(\beta+\gamma(X_1X_2+X_2X_1))(X_1+X_2) \tag{3.2}$$

where β and γ are scalars and the 4-vector part of S is proportional to the anticommutator of X_1 and X_2.

For the previously known cases in which the linear interpolation of higher grade objects gives a blade, such as with circles [26] and point pairs [29] both with common points, the factor S is a scalar and the object X_m is simply 'half-way' between the objects. We can extend this notion to the cases where the addition of objects is not a blade by using our object X_m, which has been corrected to being a blade, as the half-way object. We can use this idea of the half-way object to recursively subdivide the space between X_1 and X_2 allowing us to create objects that are any fraction of X_1 and X_2. While this technique allows us to generate interpolant objects from any two objects (of the same type), it is nevertheless clumsy to represent fractional interpolant objects via repeated subdivision. This subdivision technique also provides no obvious way of performing an average of many objects. What we would really like is some way of directly dealing with the linear interpolation $\alpha X_1+(1-\alpha)X_2$.

3.5 Creating a blade from a pure grade multivector

Consider the general interpolant, $X'_\alpha = \alpha X_1 + (1-\alpha)X_2$ where X_1 and X_2 are blades of the same grade. We claim that we can project X'_α into object space in a simple and general way. First we will generalise equation (3.2) to the interpolation case:

$$X_\alpha = S(\alpha X_1 + (1-\alpha)X_2) = (\beta_\alpha + \gamma_\alpha(X_1X_2 + X_2X_1))(\alpha X_1 + (1-\alpha)X_2) \tag{3.3}$$

where β_α and γ_α are once again scalars.

Since S is of the form *(scalar + 4-vector)* it is self reverse. Defining $S^- = \langle S\rangle_0 - \langle S\rangle_4$ we get the result that S^-S is a scalar, and can therefore write $X'_\alpha = kS^-X_\alpha$ where k is a scalar and $k = \frac{1}{S^-S}$.

To use this decomposition we need to extract S from X'_α. To do this we can use the methods of Chapter 2, or as follows using the square root operator of Dorst and Valkenburg [30].

Let $SX'_\alpha = X_\alpha$, where X_α is a valid object (squaring to $\pm$ 1). Now define $\Sigma \in Cl^{0,4}_{4,1}$, ie. it only contains 0 and 4 grade coefficients and is an element of the conformal algebra. Then, defining $[[\Sigma]] = \sqrt{\langle\Sigma\rangle_0^2 - \langle\Sigma\rangle_4^2}$, the square root can be found as:

$$\sqrt{\Sigma} = \frac{\Sigma \pm [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 \pm [[\Sigma]]}} = \frac{\langle\Sigma\rangle_0 \pm [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 \pm [[\Sigma]]}} + \frac{\langle\Sigma\rangle_4}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 \pm [[\Sigma]]}} \tag{3.4}$$

To use this method to find S^- we multiply our non-blade object by its own reverse:

$$X'_\alpha\tilde{X}'_\alpha = (kS^-X_\alpha)(kS^-X_\alpha)^\sim = k^2S^-X_\alpha\tilde{X}_\alpha(S^-)^\sim = -k^2S^-(S^-)^\sim = -k^2(S^-)^2 \tag{3.5}$$

This is now in a form where we can apply the above square root formula:

$$kS^- = \sqrt{-X'_\alpha\tilde{X}'_\alpha} \tag{3.6}$$

It now simply remains to isolate X_α via multiplication by kS where $kS = \langle kS^-\rangle_0 - \langle kS^-\rangle_4$. Since $(kS)(kS^-)$ is a scalar, we have

$$X_\alpha = \frac{kS}{(kS)(kS^-)}X'_\alpha \equiv SX'_\alpha \tag{3.7}$$

This result is particularly important as we have identified a way of projecting any pure grade object of the form S^-X (with X a blade) back to the blade manifold. An immediate

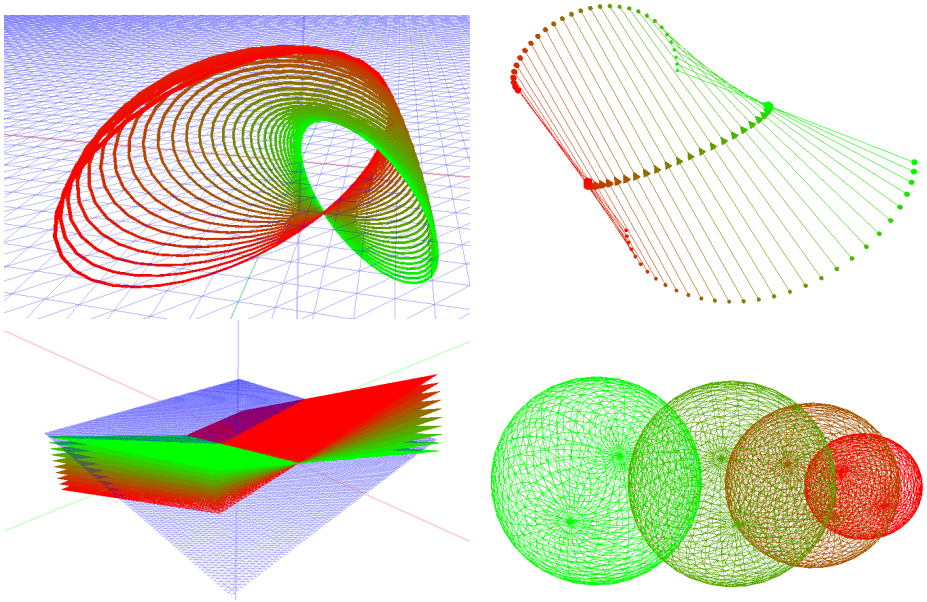


Fig. 3.1 Linear interpolation between different geometric objects.
(a) circles, (b) point pairs, (c) planes, (d) spheres. The pure red and green objects here represent X_1 and X_2 and the intermediate colours show the interpolations between them. Here we are stepping linearly through α between 0 and 1 with the number of steps chosen to show the interpolations as clearly as possible.

application of this is that we can now deal with arbitrary linear combinations of objects, allowing us to smoothly interpolate as well as to average and cluster geometric primitives. Additionally we can correct numerical errors that result from arithmetic operations to give true blades again. Figure 3.1 shows examples of interpolating various geometric objects.

As shown in Chapter 2 this method holds for all the standard normalised conformal objects of grade 2 or above (point pairs, lines, circles, planes, spheres). The direct interpolation method is potentially more computationally efficient than the bivector interpolation method, and its form indicates that it is covariant, ie, for a rotor transformation given by R,

$$R[\alpha X_1 + (1-\alpha)X_2]\tilde{R} = \alpha R X_1 \tilde{R} + (1-\alpha) R X_2 \tilde{R}$$

Unfortunately, we cannot apply this form of interpolation to points as we encounter a problem due to the fact that for a conformal point P, $P\tilde{P} = 0$. However, we saw in equation (3.1) that points can be interpolated very easily using known explicit formulae.

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	0,2	1,3	2,4	3,5	4
2	2	1,3	0,2,4	1,3,5	2,4	3
3	3	2,4	1,3,5	0,2,4	1,3	2
4	4	3,5	2,4	1,3	0,2	1
5	5	4	3	2	1	0

Table 3.1 Resulting grades from the geometric product of pure grade objects. The grade 4 row is highlighted here as it is of specific interest for the form of the S in equation (3.3).

3.6 Techniques for understanding interpolant properties

In order to use the interpolant blades it is useful to get a handle on some of their properties. In several cases it is possible to get good insight into how the interpolant behaves by looking at the interpolant of the dual of the blades, but in others we need to consider the form of the *(scalar + 4-vector)* required to project the interpolant back to the blade manifold. As before we write our blades as:

$$X_\alpha = (\langle S\rangle_0 + \langle S\rangle_4)(\alpha X_1 + (1-\alpha)X_2)$$

From this we immediately see that for the multiplication to be grade preserving we require $\langle S\rangle_4 X_1$ and $\langle S\rangle_4 X_2$ to give only objects of grade n where n is the grade of X_1 and X_2. Table 3.1 shows the resultant grades from the geometric product of pure grade objects and Table 3.2 shows the resultant grades from the inner product. These tables are presented here for reference and will be returned to when dealing with individual grade blades.

3.7 Point pairs

We start with point pairs. Previous work [29] has shown that when an end point is shared between point pairs A and B the interpolant point pairs are also blades and their end points trace out the circumference of the circle formed by the shared point and the additional separate end points. Three points X,Y,Z define a circle $C \propto X \wedge Y \wedge Z$ and a fourth point V lying on the circle will satisfy $V \wedge C = 0$, this allows us to define a check to see if two point pairs are chords of the same circle. Point pairs $A = V \wedge X$, $B = Y \wedge Z$ will satisfy

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	1	2	3	4
2	0	1	0	1	2	3
3	0	2	1	0	1	2
4	0	3	2	1	0	1
5	0	4	3	2	1	0

Table 3.2 Resulting grades from the inner product of pure grade objects. In the case of the inner product of a multivector and a scalar by definition the result is always 0 rather than some other scalar valued function of the scalar and multivector. Here again the grade 4 row is highlighted here as it is of specific interest for the form of the S in equation (3.3).

$A \wedge B = 0$ if they are both chords and any additional chord, W, of the same circle will satisfy $W \wedge A = 0, W \wedge B = 0$ and thus $W \wedge (A+B) = 0$. This leads to:

Theorem 1 *If point pairs A and B are both chords of a common circle C the interpolant point pairs* $T \propto \alpha A + (1-\alpha)B$ *are blades and also have end points lying on C as* $\langle AB \rangle_4 = A \wedge B = 0$ *and* $(\alpha A + (1-\alpha)B) \wedge (A+B) = 0$.

Note, since $\langle AB \rangle_4 = A \wedge B = 0$, the projector S is a scalar. The common circle itself is the 'join' of the two original point pairs and can be computed with the algorithms supplied in Chapter 21 of Dorst, Fontijne and Mann [29]. Figure 3.2 shows two cases of the interpolation of co-planar point pairs that lie on the same circle.

Turning to the more general case of two point pairs in arbitrary positions in space we can get insight into the form of the interpolant by considering the components of the *scalar + 4-vector* projection factor. In the case of the geometric product between grade 4 and grade 2 objects we see from Table 3.1 that we produce both 2 and 4-vector grades. The 2-vector part of the geometric product comes from the inner product between the point pairs and the 4-vector. ie. for point pairs A and B, $\langle S \rangle_4 (\alpha A + (1-\alpha)B) = \langle S \rangle_4 \cdot (\alpha A + (1-\alpha)B)$. For the general case of two point pairs not lying in plane ie. $A \wedge B \neq 0$, we can show that there is only one object that behaves in this way, the sphere $\Sigma \propto A \wedge B$, as it passes through both end points of both point pairs. This is illustrated in Figure 3.3 and suggests that the sphere Σ is intrinsically tied to the form of the interpolant objects. Indeed we can see from the same visualisation that the interpolant C of point pairs A and B always has endpoints lying on the surface of the sphere Σ.

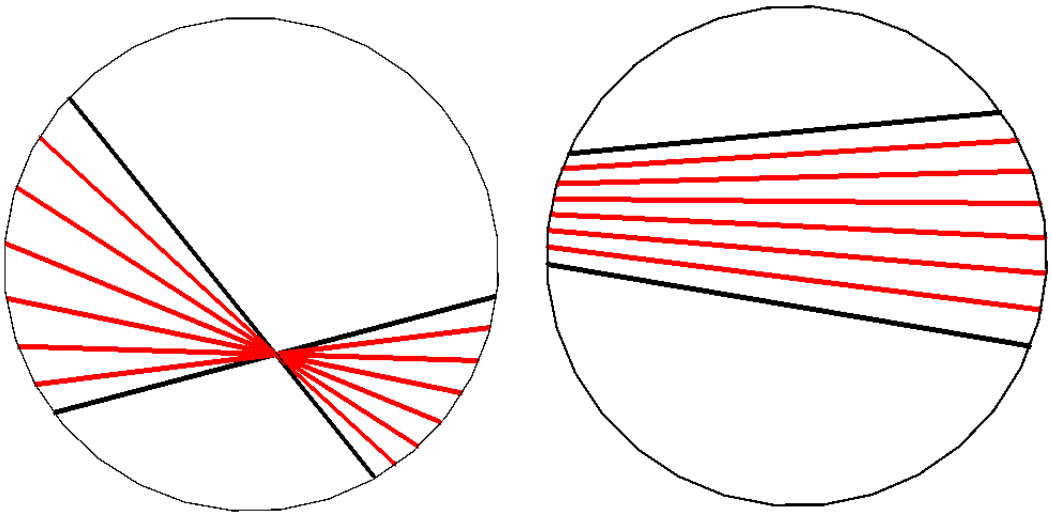


Fig. 3.2 The interpolation of point pairs with endpoints lying in the same plane and on a common circle is a blade and also lies on the same common circle, even in cases in which there are no shared endpoints or intersections. In this figure the red lines are the interpolation of the black lines.

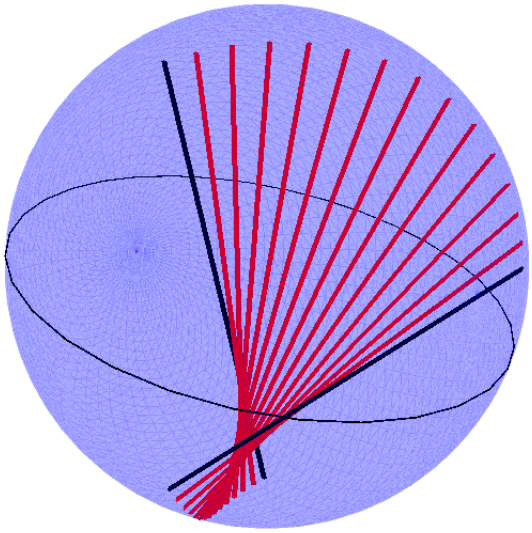


Fig. 3.3 The interpolation in red of point pairs A and B, here shown in black, lie on the surface of the sphere $\propto A \wedge B$, shown in blue with black equator circle.

We can prove this by showing that $C \wedge B$ or $C \wedge A$ also produces the sphere:

$$\Sigma \propto A \wedge B$$

First consider an interpolant object C and its outer product with one of the original objects, B

$$\begin{aligned} C &= (\langle S \rangle_0 + \langle S \rangle_4)(\alpha A + (1-\alpha)B) \\ C \wedge B &= ((\langle S \rangle_0 + \langle S \rangle_4)(\alpha A + (1-\alpha)B)) \wedge B \\ &= \langle S \rangle_0 (\alpha A + (1-\alpha)B) \wedge B + (\langle S \rangle_4 (\alpha A + (1-\alpha)B)) \wedge B \end{aligned}$$

as $B \wedge B = 0$ we see that

$$\alpha \langle S \rangle_0 A \wedge B = \langle S \rangle_0 (\alpha A + (1-\alpha)B) \wedge B$$

Now we just need to prove that $\langle S \rangle_4 ((\alpha A + (1-\alpha)B) \wedge B)$ is a scalar multiple of $A \wedge B$. From equation (3.3) we know that

$$\langle S \rangle_4 \propto \langle AB + BA \rangle_4$$

it is therefore sufficient to prove that:

$$T = (\langle AB + BA \rangle_4 (\alpha A + (1-\alpha)B)) \wedge B \propto A \wedge B$$

We can convert the outer product into a geometric product followed by a projection and thus can write:

$$\begin{aligned} T &= \langle \langle AB + BA \rangle_4 (\alpha A + (1-\alpha)B) B \rangle_4 \\ &= \langle \langle AB + BA \rangle_4 (\alpha AB + (1-\alpha)) \rangle_4 \\ &= (1-\alpha)(\langle AB \rangle_4 + \langle BA \rangle_4) + \alpha \langle (\langle AB \rangle_4 + \langle BA \rangle_4) AB \rangle_4 \end{aligned}$$

As $\langle AB \rangle_4 \equiv \langle BA \rangle_4$ we can write this as:

$$T = 2(1-\alpha)\langle AB \rangle_4 + 2\alpha \langle \langle AB \rangle_4 AB \rangle_4$$

and this can further simplified using the fact that:

$$\langle \langle AB \rangle_4 AB \rangle_4 = \langle \langle AB \rangle_4 \langle AB \rangle_0 \rangle_4$$

As $\langle AB\rangle_0 = A \cdot B$ is a scalar:

$$\begin{aligned} T &= 2(1-\alpha)\langle AB\rangle_4 + 2\alpha\langle AB\rangle_4 (A \cdot B) \\ &= 2((1-\alpha) + \alpha A \cdot B)\langle AB\rangle_4 \end{aligned}$$

As $2((1-\alpha) + \alpha A \cdot B)$ is a scalar we see that the proof is completed

$$2((1-\alpha) + \alpha A \cdot B)\langle AB\rangle_4 \propto \langle AB\rangle_4 \equiv A \wedge B$$

Figure 3.3 shows a graphical representation of the interpolant point pairs lying on the surface of the sphere. To summarise:

Theorem 2 *For non-coplanar point pairs A and B, all interpolant point pairs lie on the surface of the sphere* $\Sigma \propto A \wedge B$.

3.8 Circles

The interpolant of circles has a range of properties that are useful and clearly intrinsically tied to the geometry of spheres and point pairs. Initially we will consider the case of two circles in space that both lie on the surface of a common sphere. In past work it has been shown that circles with two common points interpolate directly without requiring re-projection and the interpolant lies on their common sphere [26] [29]. Here, as with the point pairs, we can show that this is true for a broader class of circles:

Theorem 3 *If circles* C_1 *and* C_2 *together define the caps of a common sphere then* $\langle S\rangle_4 \propto \langle C_1 C_2\rangle_4 = 0$ *where S is of the form shown in equation (3.3) and thus any interpolant object* $C_3 = \alpha C_1 + (1-\alpha)C_2$ *is a blade without requiring re-projection to the blade manifold.*

This can be proved by considering each circle C_i as the intersection of a plane P_i and a sphere Σ_i. Forming this intersection via the dual (where $X^* = XI_5$ and I_5 is the 5D space pseudoscalar as before), we have:

$$\begin{aligned} C_1 &= (\Sigma_1^* \wedge P_1^*)I_5 \\ C_2 &= (\Sigma_2^* \wedge P_2^*)I_5 \\ \langle C_1 C_2\rangle_4 &= -\langle (\Sigma_1^* \wedge P_1^*)(\Sigma_2^* \wedge P_2^*)\rangle_4 \end{aligned}$$

Since $(\Sigma_1^* \wedge P_1^*)$ and $(\Sigma_2^* \wedge P_2^*)$ are both bivectors:

$$\langle C_1 C_2\rangle_4 = -\Sigma_1^* \wedge P_1^* \wedge \Sigma_2^* \wedge P_2^*$$

and so if $\Sigma_2 \propto \Sigma_1$:

$$\langle C_1 C_2 \rangle_4 \propto -\Sigma_1^* \wedge P_1^* \wedge \Sigma_1^* \wedge P_2^* = 0$$

We can additionally find the unique common sphere by finding the join of the circles or by reverting to linear algebra techniques:

Conjecture 1 *If circles C_1 and C_2 together define the caps of a common sphere Σ then $\langle C_1 \Sigma \rangle_3, \langle C_2 \Sigma \rangle_3 = 0$. Σ can be found by the following process:*

First we define:

$$\underline{A} = \begin{bmatrix} \underline{M}_3 \underline{C}_1 \\ \underline{M}_3 \underline{C}_2 \end{bmatrix}$$

where M_3 is the truncated identity matrix that performs selection of grade 3 elements from a vector of coefficients and $\underline{C}_1$ and $\underline{C}_2$ are the matrices that perform the left geometric product of C_1 and C_2 respectively with a vector of coefficients. We can then find the Σ for $\underline{A}\underline{\Sigma} = 0$ where $\underline{\Sigma}$ is a vector of canonical blade coefficients limited to only the 4-vector blades. In the case that C_1 and C_2 are the same radius then $\Sigma \propto (C_1 + C_2)((C_1 + C_2) \wedge n_\infty) I_5$.

The case for circles of the same radius is visualised in Figure 3.4.

It is also the case that the interpolant lies on the surface of the common sphere:

Theorem 4 *If circles C_1 and C_2 together define the caps of a common sphere then all interpolant circles $C_3 = \alpha C_1 + (1-\alpha) C_2$ (which we have shown to be blades) also lie on the surface of the sphere Σ common to both.*

We can prove this by considering the outer product of the interpolant circle with D, an arbitrary point on the common sphere Σ:

$$\Sigma \propto D \wedge C_3 = \alpha D \wedge C_1 + (1-\alpha) D \wedge C_2$$

Figure 3.5 shows an example of this interpolation.

Thus far we have dealt exclusively with circles on a common sphere. In the case in which C_1 and C_2 do not lie on the same sphere we can again look at how the interpolants behave by considering the form of the *(scalar + 4-vector)* that we use to project the interpolant back to the blade manifold. In the case of the geometric product between grade 4 and 3 objects we see from Table 3.1 that we produce both 1 and 3-vector grades, however the 1-vector part of the geometric product comes only from the inner product between the 4-vector and the circles. To maintain grade after the multiplication the 4-vector must therefore be the object

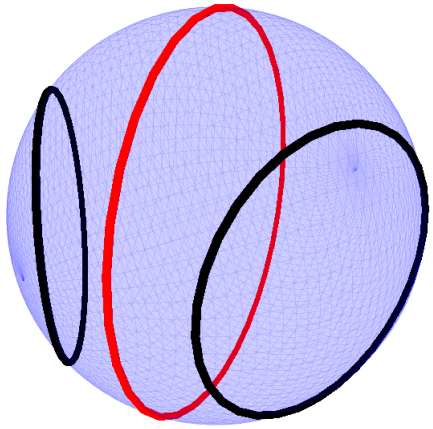


Fig. 3.4 The half way circle $C_1 + C_2$ shown in red in this figure is the equator of the sphere through both C_1 and C_2 if they define a common sphere and have the same radius

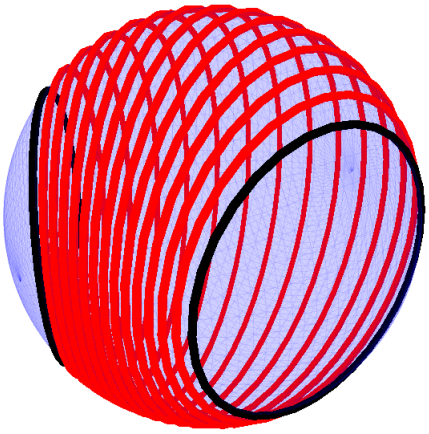


Fig. 3.5 The interpolation of circles C_1 and C_2 is a blade and lies on the surface of a sphere if C_1 and C_2 define a common sphere.

that has an inner product of zero with both circles. This object is the sphere into whose surface both circles plunge orthogonally [29]:

Theorem 5 *If circles C_1 and C_2 together do **not** lie on a common sphere then the 4-vector from our blade projection equation $\langle S\rangle_4 \propto \langle C_1C_2\rangle_4$ is itself a blade and geometrically represents the sphere through which both circles plunge orthogonally. ie. $C_1 \cdot \langle C_1C_2\rangle_4 = 0$. This property means all interpolant circles after projection to the blade manifold also plunge through $\langle C_1C_2\rangle_4$ orthogonally. ie. $C_3 \cdot \langle C_1C_2\rangle_4 = 0$.*

The intersections of the interpolant circles with the sphere $\langle S\rangle_4$ produce a set of point pairs. Intuition would suggest that these point pairs have properties tied to the interpolation of the point pairs generated by the original two circles C_1 and C_2 and indeed we can numerically verify that this is the case:

Conjecture 2 *If circles C_1 and C_2 together do **not** lie on a common sphere then the intersection point pair P_α formed by the meet of the circle interpolant for a given value of α with the orthogonal sphere $\langle C_1C_2\rangle_4$ ie. $P_\alpha \propto C_\alpha \vee \langle C_1C_2\rangle_4$ is the same as the re-projected interpolant Π_α of the point pairs formed from the meet of C_1 and C_2 with $\langle C_1C_2\rangle_4$.*

Figure 3.6 shows the interpolation of two non co-spherical circles as well as the sphere these circles define and the intersection point pairs they generate.

3.9 Lines

When looking at lines we can attempt to use some of the same techniques that we used for circles. First consider the form of $\langle S\rangle_4 \propto \langle L_1L_2\rangle_4$. For lines $\langle L_1L_2\rangle_4 \propto I_5n_\infty$, giving the form of the projection of $(X_1 + X_2)$ as:

$$X_3 = (\mu + \nu I_5 n_\infty)(X_1 + X_2) \tag{3.8}$$

where μ and ν are scalars. While neat, this form I_5n_∞ does not on its own provide information on the properties of the interpolated line. Instead we consider the interpolation of the dual of the lines, and to understand this interpolation we must take a short detour via screw theory.

3.9.1 Screw Theory

Screw theory was developed by Sir Robert Stawell Ball in 1900 in his seminal work ‘A treatise on the theory of screws’ [2]. His original applications were kinematics and one of

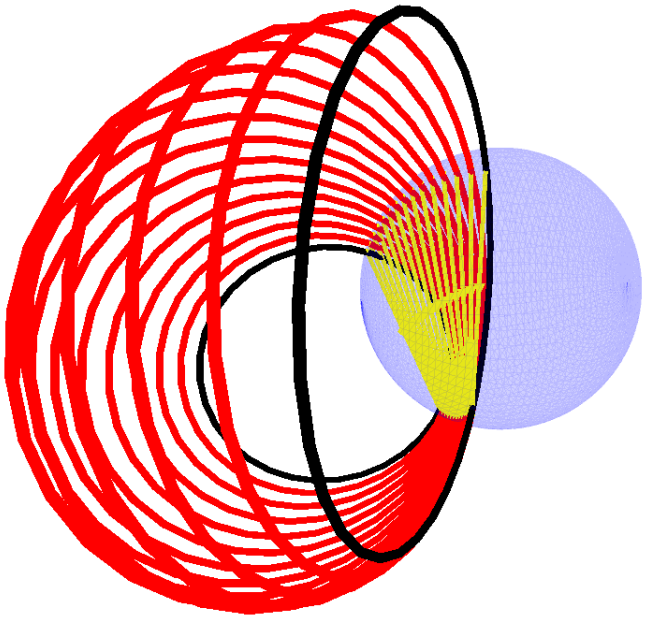


Fig. 3.6 The interpolation of the two black circles which are not spherical caps, intersect orthogonally with a single sphere (shown in blue). The point pairs formed from the two intersection points are shown in yellow.

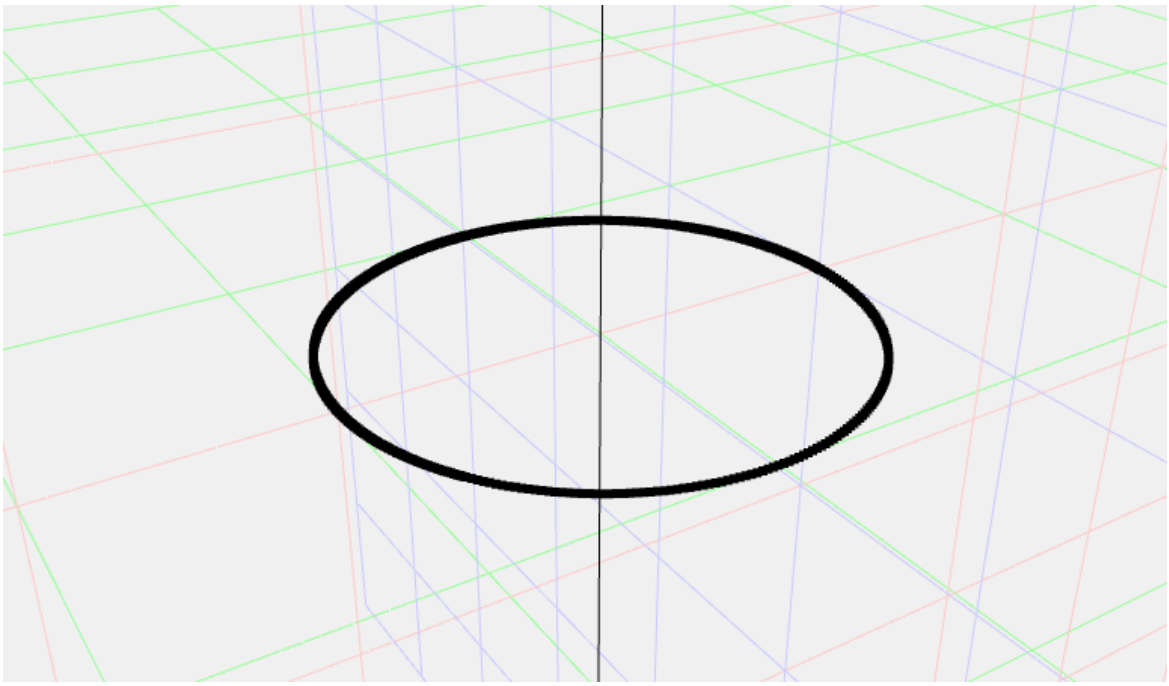


Fig. 3.7 The orbit of a line

the most important theorems in the area, Chasles' theorem, states that the most general rigid body displacement can be described by a screw transformation. More recently screw theory, and the highly related study of dual quaternions, has been applied to robotics, computational geometry and multibody dynamics [67, 80, 68].

Screw transformations consist of a translation along an axis and a rotation around that axis. To parameterise a screw we define the direction of the screw axis via a unit vector $\hat{m}$, a point on the screw axis p and a screw pitch h. The pitch represents how far to move in the direction of the screw axis for each complete revolution about the axis.

3.9.2 Bivector representation of a line

A line in CGA is represented as a 3-vector, or dually as a bivector:

$$L^* = \hat{m}I_3 + (p \wedge \hat{m})I_3 n_\infty \tag{3.9}$$

This bivector formulation is equivalent to the Plücker coordinates of the line.

In [30] the authors describe the orbit of simple bivectors that describe motion. We can visualise the orbit of the dual line bivector by exponentiating the bivector to a rotor and applying it to a test point. Figure 3.7 shows the orbit of the point at the origin about a line. The motion is a circle about the line.

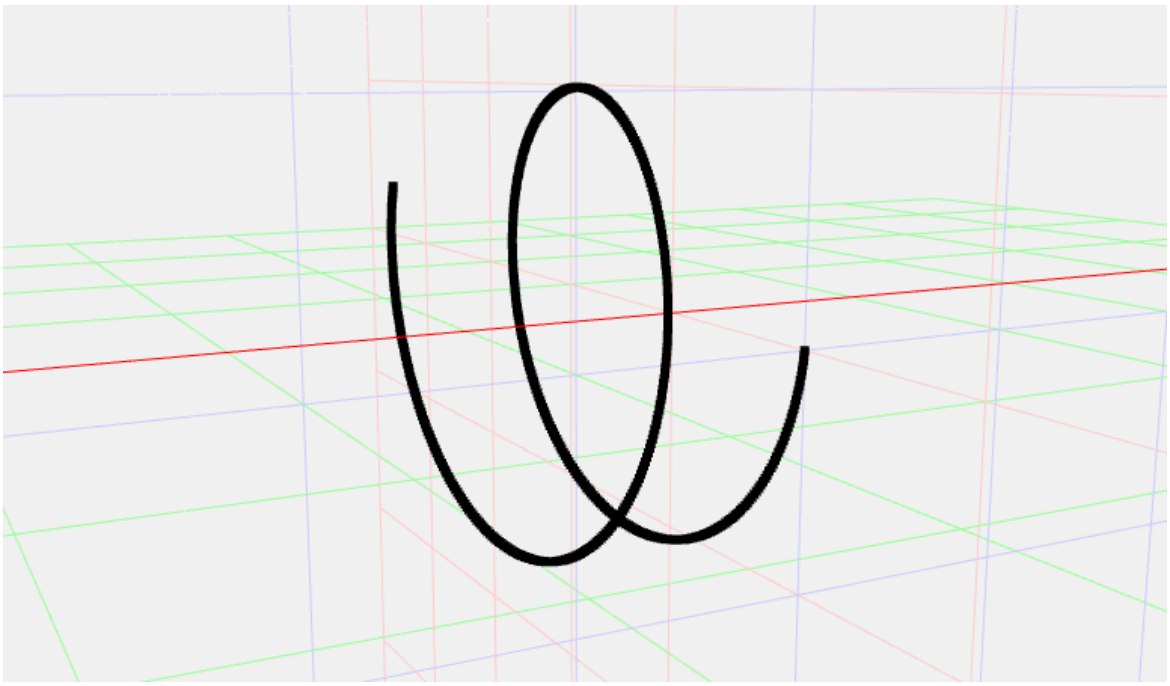


Fig. 3.8 The action of a basic screw formed by the summation of commuting bivectors applied to the point at the origin is shown in black, it forms a screw motion about the screw axis shown in red.

3.9.3 The bivector representation of a screw

To represent a screw we will couple the rotational motion of the dual of a line with a translation in the direction of that same line. The bivector T that transforms along the 3d vector t is:

$$T = tn_\infty \tag{3.10}$$

If $t = h\hat{m}$ where h is a scalar, ie. the translation is in the screw axis direction, the rotors formed from the bivectors in equations (3.9) and (3.10) commute.

It then follows that the rotor formed from the addition of the bivectors in equations (3.9) and (3.10) can be split into the rotor representing translation along the axis and the rotor representing rotation about the axis – as required for a screw. We therefore have a screw, $\mathbb{S}$, whose action on the point at the origin is shown in Figure 3.8.

$$\mathbb{S} = \hat{m}I_3 + (p \wedge \hat{m})I_3 n_\infty + h\hat{m}n_\infty \tag{3.11}$$

Hestenes and Sobczyk [56] p81 gives an expression for decomposing any bivector into two commuting blades. In the case of our screw bivector these blades represent the dual of the screw axis L^* and a translational bivector T in the direction of the screw axis. ie. given a

screw bivector $\mathbb{S}$ we can decompose it as:

$$\mathbb{S} = L^* + T$$

3.9.4 Adding dual lines

The addition of dual lines produces a bivector. Visualising the action of this bivector allows us to see that it is in fact also a screw transformation. Consider the addition of two dual lines:

$$L^*_+ = L^*_1 + L^*_2$$

we can write this elementwise as

$$L^*_+ = m_1 I_3 + \psi_1 n_\infty + m_2 I_3 + \psi_2 n_\infty$$

where ψ_i is $(p_i \wedge m_i) I_3$. We then rearrange to give something proportional to the expression in equation (3.11):

$$L^*_+ = (m_1 + m_2) I_3 + (\psi_1 + \psi_2) n_\infty = m I_3 + (p \wedge m) I_3 n_\infty + h m n_\infty$$

Where clearly $m = m_1 + m_2$. If we divide this by $|m|$ we have the general form of a normalised screw

$$\mathbb{S} = \hat{m} I_3 + (p \wedge \hat{m}) I_3 n_\infty + h \hat{m} n_\infty$$

Gathering like terms, specifically those without an n_∞ component, leads us to the conclusion that our screw axis direction $\hat{m}$ must simply be proportional to the addition of the directions of the two lines. Using this fixed axis direction we can extract the coefficient h (the pitch) of the translation bivector parallel to the screw axis:

$$\begin{aligned} m &= m_1 + m_2 \\ L^*_+ &= m I_3 + (p \wedge m) I_3 n_\infty + h m n_\infty \\ L^*_+ \cdot n_0 &= h m + (p \wedge m) I_3 \\ h &= \frac{(L^*_+ \cdot n_0) \cdot m}{|m|^2} \end{aligned}$$

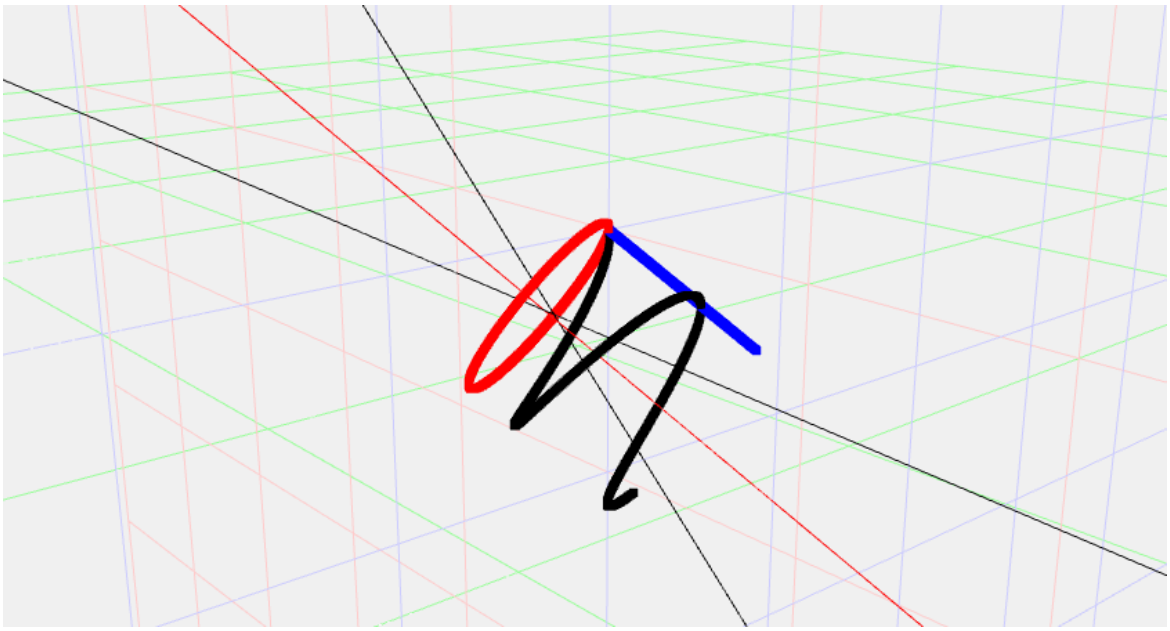


Fig. 3.9 The addition of two dual lines, here shown in black, produces a screw, whose action on the point at the origin is again shown in black. The screw can be decomposed into two commuting bivectors, whose actions are shown in red and blue. The bivector whose action is shown in red is the dual of the screw axis line, also shown in red.

With this coefficient known we now have all the pieces in place for a full decomposition of the dual line addition bivector L_+^*:

$$\begin{aligned} L_+^* &= L_1^* + L_2^* \\ m &= -\langle L_+^* \rangle_{(e_{12},e_{13},e_{23})} I_3 \\ h &= \left(\frac{(L_+^* \cdot n_0) \cdot m}{|m|^2} \right) \\ \mathbb{S} &= L^* + T \\ T &= h\hat{m}n_\infty \\ L^* &= \mathbb{S} - T \end{aligned}$$

Figure 3.9 shows the decomposition of the addition of two lines into its component parts.

3.9.5 Relationship to object manifold reprojection

We can also analyse the screw multiplied by its own reverse, comparing this formulation with our object manifold reprojection to get the projection coefficient S in terms of the screw

parameters:

$$\begin{aligned}
\mathbb{S}\tilde{\mathbb{S}} &= -(L^*)^2 + T\tilde{T} + T\tilde{L^*} + L^*\tilde{T} \\
&= 1 - 2TL^* \\
&= 1 + 2hn_\infty I_3 = 1 + 2hI_5 n_\infty \\
&\equiv (kS^-)^2 \\
(kS^-) &= \mu + \nu I_5 n_\infty \\
(kS^-)^2 &= \mu^2 + 2\mu\nu I_5 n_\infty \\
\mu &= 1,\ \nu = h
\end{aligned}$$

Using this calculated value we can see how the projection coefficient acts on the addition of lines:

$$\begin{aligned}
L^* &\propto (1 - hI_5 n_\infty)(mI_3 + (p \wedge m)I_3 n_\infty + hmn_\infty) \\
&= mI_3 + (p \wedge m)I_3 n_\infty + hmn_\infty - hmn_\infty \\
&= mI_3 + (p \wedge m)I_3 n_\infty
\end{aligned}$$

This is in fact the same line as is formed from the decomposition of the screw bivector into the screw axis bivector and pitch translation bivector. In other words, the addition of lines and reprojection to the line manifold extracts the axis of the screw formed from the addition of their duals. This axis has a direction equal to a linear interpolation of the axes of the original two lines and, as it is a mirror object, passes through the point exactly half way between the lines.

3.10 Planes

All 4-vectors are blades. Thus, for planes and spheres it is impossible to construct an invalid geometric object by addition. For planes we can analyse the form of the interpolant by again looking at the dual of a plane.

The dual of the plane can be written as:

$$P^* = \hat{m} + dn_\infty$$

where $\hat{m}$ is the 3D vector normal to the plane and d is the perpendicular distance of the plane from the origin. Thus the interpolation of duals of two planes can be written as:

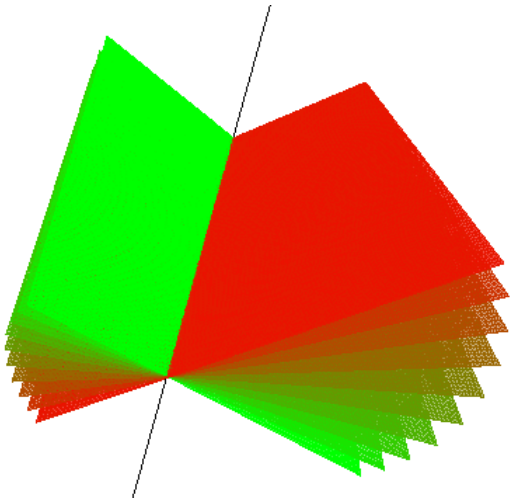


Fig. 3.10 The interpolant of two planes (green to red) always passes through the meet line (black) of the two original planes.

$$\alpha P_1^* + (1-\alpha)P_2^* = \alpha \hat{m}_1 + \alpha d_1 n_\infty + (1-\alpha)\hat{m}_2 + (1-\alpha)d_2 n_\infty$$

which, when we collect like terms, is already in the form of a dual plane P_3^*:

$$P_3^* = \alpha \hat{m}_1 + (1-\alpha)\hat{m}_2 + (\alpha d_1 + (1-\alpha)d_2)n_\infty \tag{3.12}$$

this dual plane has a normal vector that is the interpolation of the normal vectors of the original two planes and has a perpendicular distance from the origin that is also simply an interpolation of the perpendicular distance from the origin of the original two planes. An important feature of this plane interpolation is that, as noted in [13], provided the two planes intersect, the interpolant plane always passes through the line of intersection (the meet) of the two original planes. This is visualised in Figure 3.10. In the case that the planes to not intersect (or more formally are said to intersect at infinity) the interpolation will smoothly translate one plane to the other keeping the normal fixed, the parallel vs anti-parallel cases are explored in Figures 3.11 and 3.12.

3.11 Spheres

The interpolant of spheres has been studied before in [13] and [26]. As with planes, all interpolants of spheres are valid objects as $\langle \Sigma_1 \Sigma_2 \rangle_4 = 0$ and have the property of making contact with the meet of the spheres at all points during the interpolation. We can see the

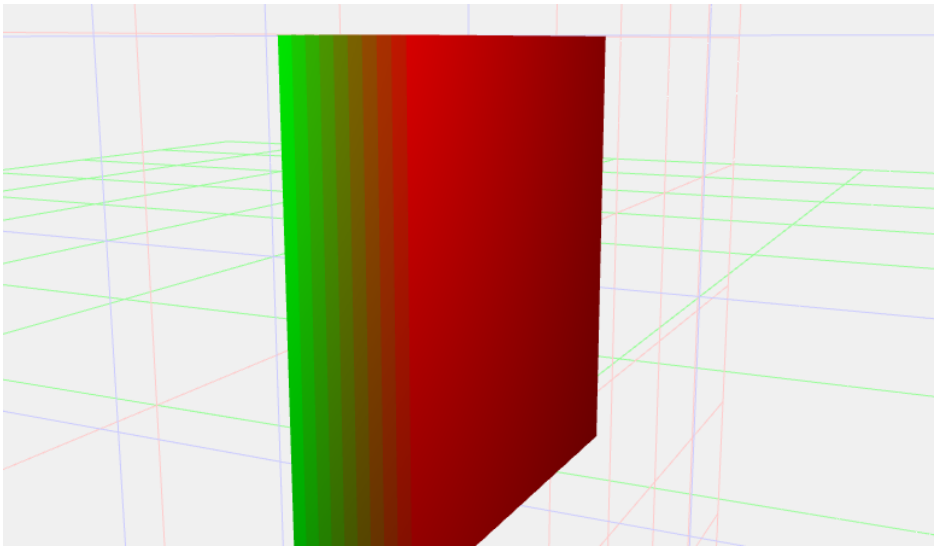


Fig. 3.11 The interpolant of two parallel planes smoothly moves between the start and end points (green to red) while maintaining the direction of the normal.

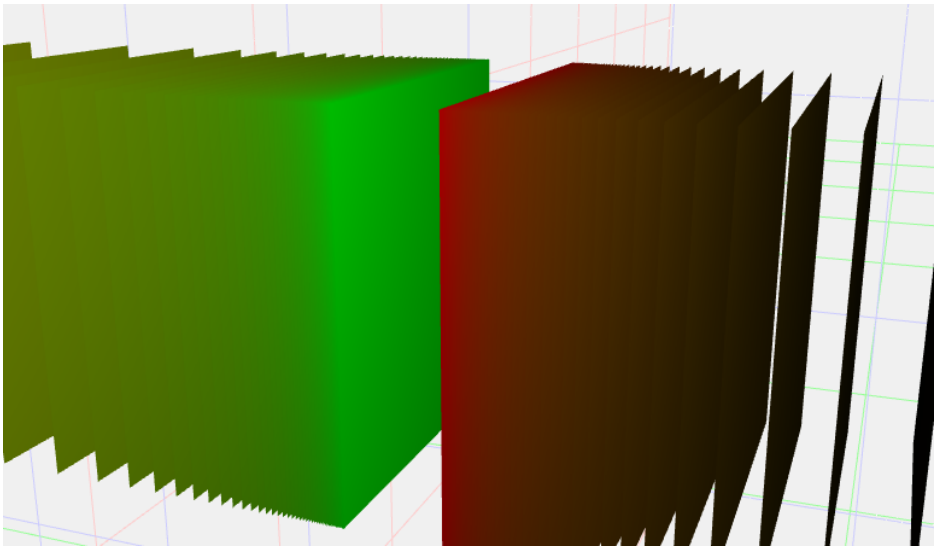


Fig. 3.12 The interpolant of two anti-parallel planes (green to red) must go via infinity due to the sign change. Care must be taken with the orientation of objects when designing algorithms using these interpolations.

form of the interpolant sphere by considering its dual form:

$$I_5\Sigma_3 = \alpha I_5\Sigma_1 + (1-\alpha)I_5\Sigma_2$$

The dual form of a sphere can be decomposed into the sum of the conformal centre point P and negative half the radius squared times n_∞:

$$I_5\Sigma = P - \frac{1}{2}\rho^2 n_\infty$$

the interpolation of the dual of two spheres is therefore

$$I_5\Sigma_3 = \alpha P_1 + (1-\alpha)P_2 - \frac{1}{2}(\alpha\rho_1^2 + (1-\alpha)\rho_2^2)n_\infty$$

For two concentric spheres ie. $P_2 = P_1$ we can therefore see that the interpolation between them will remain centred in the same place and will simply have a radius ρ_3 which varies as $\rho_3^2 = \alpha\rho_1^2 + (1-\alpha)\rho_2^2$.

As we have seen previously, the interpolation of two conformal points P_1 and P_2 is of the form

$$\alpha P_1 + (1-\alpha)P_2 = F(\alpha p_1 + (1-\alpha)p_2) - \alpha(1-\alpha)(P_1 \cdot P_2)n_\infty$$

we can therefore also write the interpolation of two non-concentric spheres as:

$$I_5\Sigma_3 = F(\alpha p_1 + (1-\alpha)p_2) - \alpha(1-\alpha)(P_1 \cdot P_2)n_\infty - \frac{1}{2}(\alpha\rho_1^2 + (1-\alpha)\rho_2^2)n_\infty$$

Collecting like factors shows that the centre point of the interpolated sphere moves linearly along the line joining p_1 and p_2

$$I_5\Sigma_3 = F(\alpha p_1 + (1-\alpha)p_2) - \left(\alpha(1-\alpha)(P_1 \cdot P_2) + \frac{1}{2}(\alpha\rho_1^2 + (1-\alpha)\rho_2^2)\right)n_\infty$$

Furthermore, writing the dot product of points in terms of their euclidean vectors we can see that the radius of the sphere varies along its interpolation path

$$I_5\Sigma_3 = F(\alpha p_1 + (1-\alpha)p_2) - \frac{1}{2}\left(-\alpha(1-\alpha)(p_1 - p_2)^2 + \alpha\rho_1^2 + (1-\alpha)\rho_2^2\right)n_\infty$$

and so the radius ρ_3 varies as

$$\rho_3^2 = -\alpha(1-\alpha)(p_1 - p_2)^2 + \alpha\rho_1^2 + (1-\alpha)\rho_2^2$$

For fixed values of ρ_1 and ρ_2 this implies ρ_3^2 varies as $-(p_1-p_2)^2$ and so the further apart the two spheres are the smaller the radius of the interpolant. To find turning points we differentiate with respect to α

$$\frac{d\rho_3^2}{d\alpha} = (-1+2\alpha)(p_1-p_2)^2 + \rho_1^2 - \rho_2^2$$

setting this to zero yields a single turning point at

$$\alpha = \frac{\rho_2^2 - \rho_1^2 + (p_1-p_2)^2}{2(p_1-p_2)^2}$$

Considering the second derivative

$$\frac{d^2\rho_3^2}{d\alpha^2} = 2(p_1-p_2)^2$$

we see that this is always positive and so the stationary point is a minimum.

For the case that the surfaces of Σ_1 and Σ_2 are just touching we have the condition

$$(p_1-p_2)^2 = (\rho_1+\rho_2)^2 = \rho_1^2 + \rho_2^2 + 2\rho_1\rho_2$$

returning to the first derivative in this case

$$\alpha = \frac{2\rho_2(\rho_1+\rho_2)}{2(\rho_1+\rho_2)^2} = \frac{\rho_2}{(\rho_1+\rho_2)}$$

this value of α is the point at which the centre of the interpolant sphere lies on the surface of both spheres. At this point the squared radius is zero:

$$\rho_3^2 = -\frac{\rho_2}{(\rho_1+\rho_2)}\frac{\rho_1}{(\rho_1+\rho_2)}(\rho_1+\rho_2)^2 + \frac{\rho_2}{(\rho_1+\rho_2)}\rho_1^2 + \frac{\rho_1}{(\rho_1+\rho_2)}\rho_2^2$$

$$\rho_3^2 = \frac{-\rho_2\rho_1(\rho_1+\rho_2) + \rho_2\rho_1^2 + \rho_1\rho_2^2}{(\rho_1+\rho_2)} = 0$$

Pulling the spheres further apart from this point so that they no longer intersect will therefore produce a sphere with negative radius, an *imaginary* sphere. These results are already known [13] and are here included for completeness.

3.12 Applications

The ability to interpolate geometric objects suggests a wide variety of applications in the areas of computer vision and graphics. There are many traditional algorithms in vision that rely solely on point information from images and ignore lines and other, potentially useful, geometric primitives. Many of these algorithms have been non trivial to translate into the framework of CGA due to having to specify transformations between objects explicitly rather than implicitly via the objects themselves. The ability to average geometric objects directly suggests immediate applications in clustering of objects extracted from real data, interpolation to produce surfaces and other areas for problems where we might normally use linear algebra.

3.12.1 Higher order spline interpolation through objects

With the ability to construct arbitrary linear combinations of blades we naturally might wonder about the applications of this to spline generation through control objects. Figure 3.13 shows an example of interpolating through different control objects with different orders of spline. As expected, higher order interpolation produces smoother surfaces through our objects.

3.12.2 Recursive scene simplification by averaging conformal objects

When extracting geometric primitives from triangulated CAD models from point cloud data or from images, there are often many objects that lie close to each other in space. Line segment detectors, for example, will often extract long lines as multiple line segments that need stitching together. We would like a way of simplifying these noisy models by collapsing objects that are close together into a single object. One way to do this is via a recursive filtering algorithm as follows:

1. Set a minimum cost threshold for difference between objects
2. Compute the cost between all objects of the same grade in the scene
3. If all costs are above the threshold then terminate the algorithm
4. Average the two objects with the smallest cost
5. Return to step (2)

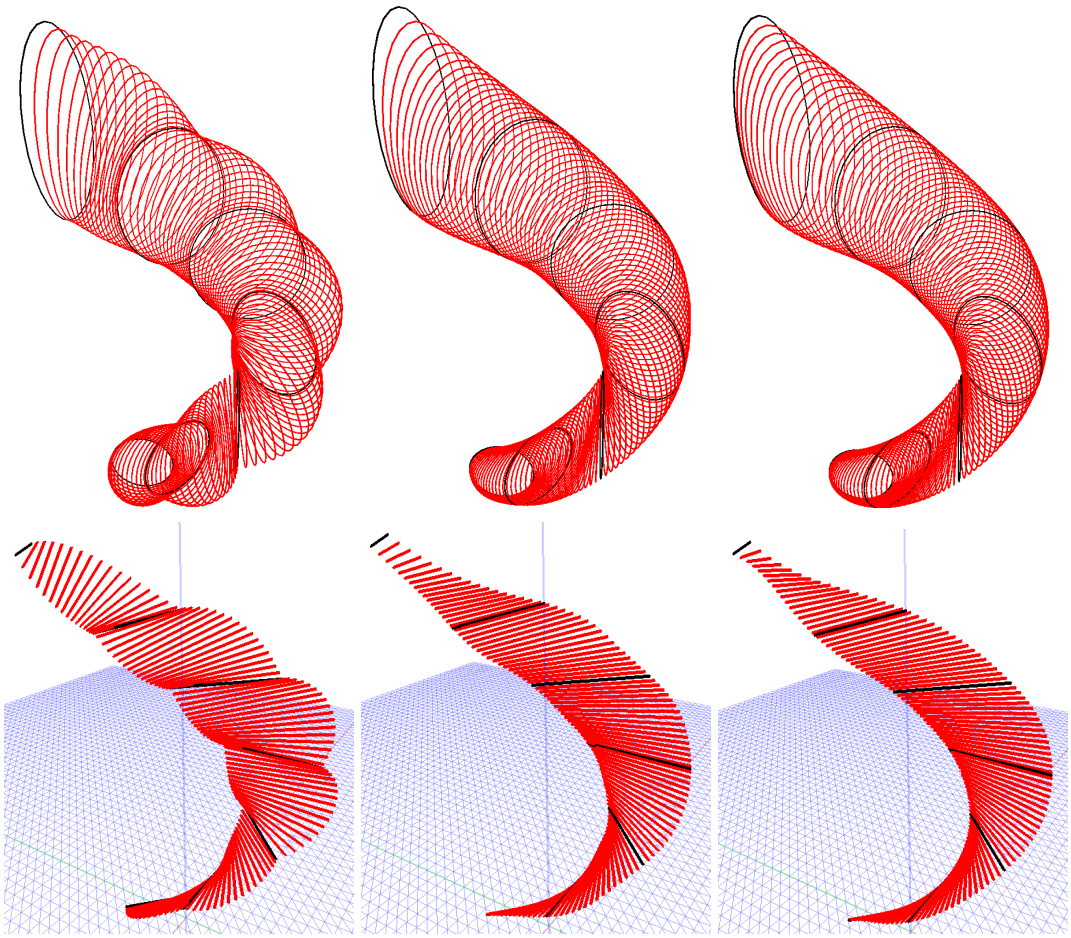


Fig. 3.13 Interpolation through control objects.
Top: Circles. Bottom: Point pairs.
Interpolation type: (a,d) Linear, (b,e) Quadratic, (c,f) Cubic

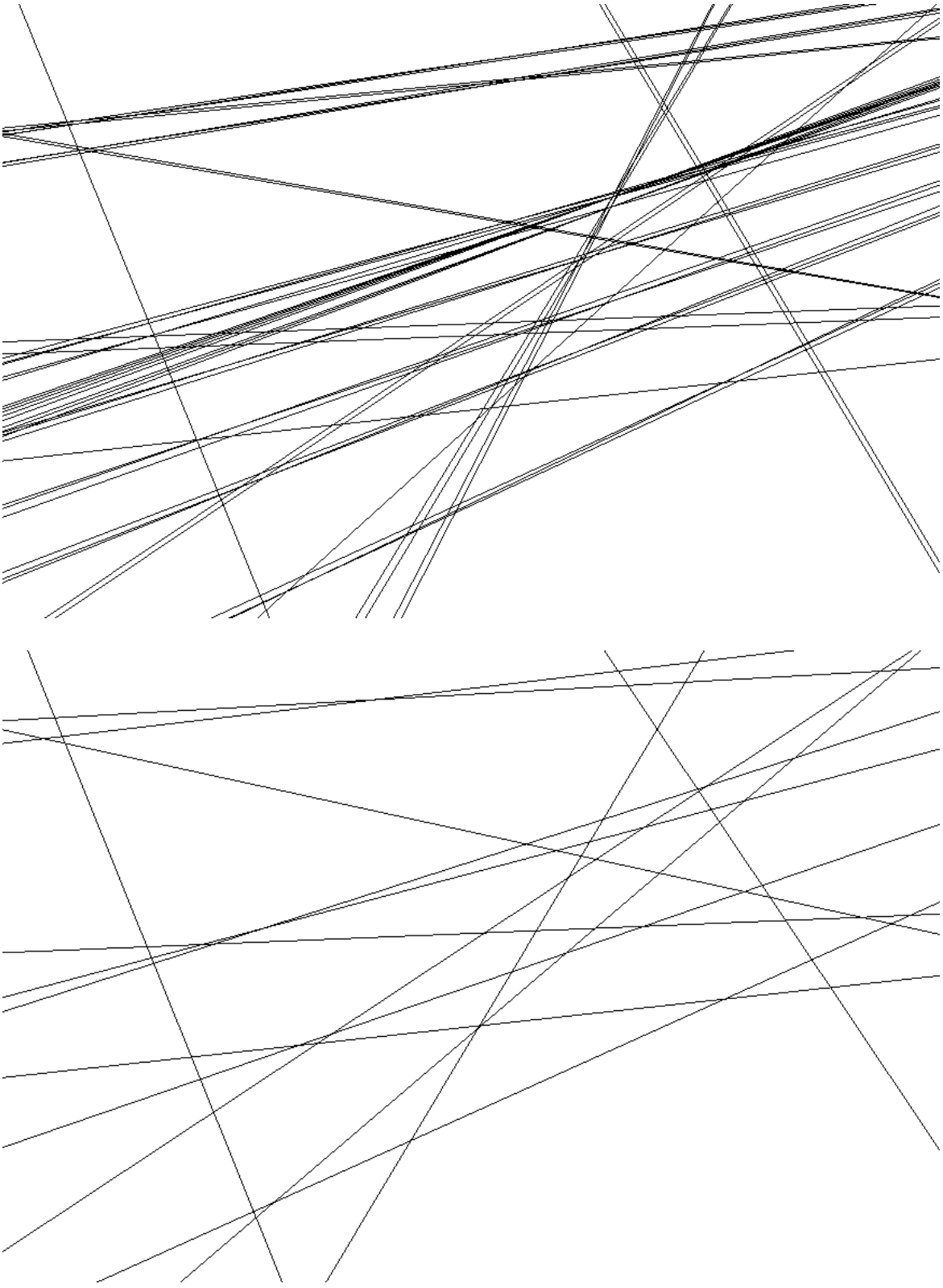


Fig. 3.14 A 3d line model before (above) and after (below) recursive scene simplification.

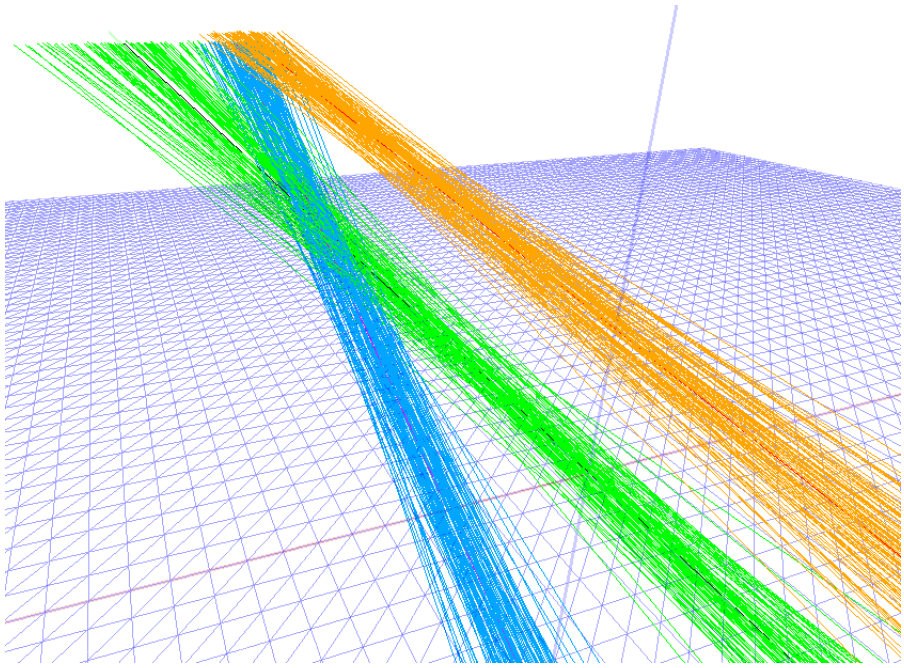


Fig. 3.15 Three clusters of 3d lines correctly segmented by the algorithm.

This leads to a simplified model that retains the core features of the original model. For comparison of objects X_i and X_j we use the cost function C_{ij} for a rotor R_{ij} as defined in [35]:

$$C_{ij} = \langle (R_{ij} - 1)(\tilde{R}_{ij} - 1) \rangle_0 + \langle R_{ij\|} \tilde{R}_{ij\|} \rangle_0 \tag{3.13}$$

where $R_{\|} = R \cdot e$, and gives the component of R having n_∞ as a factor and R_{ij} is the rotor that takes X_i to X_j as described in [75]. An example of this algorithm working on simulated lines is shown in Figure 3.14.

This algorithm is simply one way to perform scene simplification and it has a high computational complexity making it run slowly for large numbers of objects, but is included here as an example of one potential area the averaging of object methodology may be applied to.

3.12.3 k-means clustering of conformal objects

One of the most fundamental and simple clustering algorithms is known as k-means clustering [32]. Consider a 3d scene composed of k geometric objects of a given grade. We have multiple noisy observations for each object and so would like to fit k centroids to these clusters to represent the "true" objects in the world.

The steps for implementing this clustering are given below:

1. Randomly assign k objects to be the initial positions of the cluster centroids, leave all other objects unassigned
2. Assign each object in the scene to the centroid closest under our given cost metric, again we use the cost function given in equation (3.13)
3. If this is not our first iteration and no objects have changed assignment then terminate the algorithm
4. The centroid of each cluster is moved to the mean of the objects assigned to it, where mean is defined as the sum of the objects in the cluster projected back onto the blade manifold
5. Go to step (2)

Figures 3.15 and 3.16 show the successful application of this algorithm on simulated data – each line or circle has been associated with the cluster (indicated by colour) to which it is most likely to belong. One of the key advantages of using the averaging of objects and correction back to a blade for this algorithm is that it is computationally cheap. A typical approach in GA to this kind of problem might involve attempting to find the mean of a given cluster by optimisation of our cost function through a space parameterising our centroid objects. Here we can simply average the objects in each cluster making it feasible to cluster very large numbers of conformal objects quickly.

3.12.4 Closest point to two non intersecting lines (least squares sense)

Consider two non-intersecting non-coplanar lines in 3d space, L_1 and L_2. We wish to find the point P that lies closest to both in a least squares sense. First we will construct two orthogonal intermediary lines $L_+ = S_+(L_1 + L_2)$ and $L_- = S_-(L_1 - L_2)$ where $S(X)$ represents the projection of a 3-vector X back onto the line manifold. L_+ and L_- both lie half way between the two original skew lines but intersect at right angles . The intersection of these lines is the point P that lies half way between the original lines. To extract this point of intersection we can follow the formula given in [76]:

$$Q = (L_- n_0 L_-) n_\infty (L_- n_0 L_-) + L_+ (L_- n_0 L_-) n_\infty (L_- n_0 L_-) L_+$$

$$P = \frac{-Q n_\infty Q}{(Q n_\infty Q) \cdot n_\infty}$$

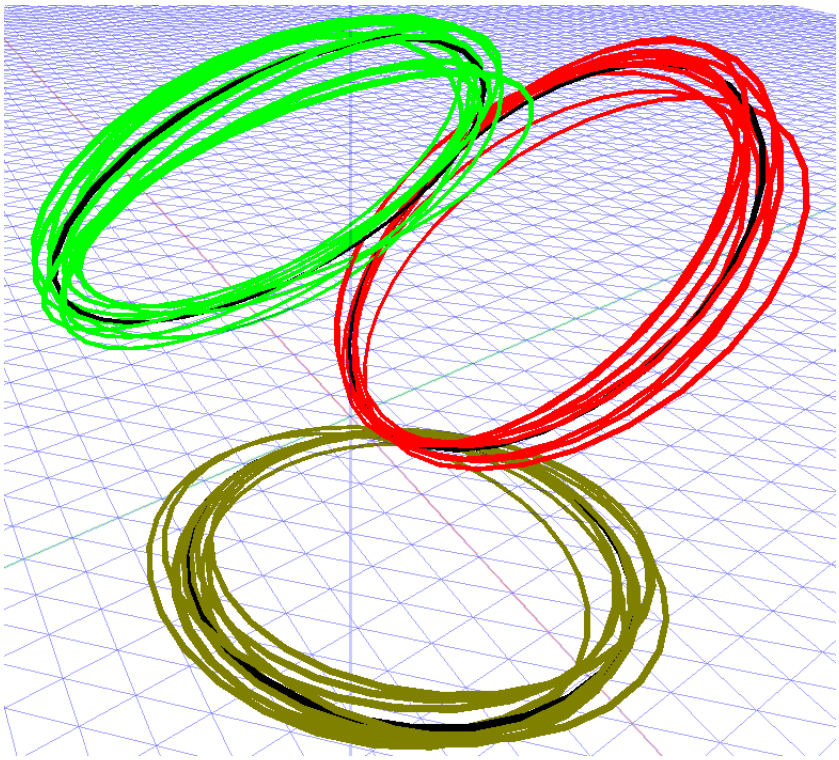


Fig. 3.16 Three clusters of 3d circles correctly segmented by the algorithm. The black circles here are the final computed cluster centroids.

3.13 Conclusions

This chapter has shown how we are able to add multiples of conformal objects by factoring the resulting multivector into a scalar plus 4-vector term and a valid geometric object. We have then investigated the form of this multivector for each grade of conformal object. Using the ideas of interpolating and averaging objects, a range of applications are suggested with relevance in computer vision and computer graphics.

Chapter 4

Exploring Novel Surface Representations via an Experimental Ray-Tracer in CGA

Arithmetic! Algebra! Geometry! Grandiose trinity! Luminous triangle! Whoever has not known you is without sense!

Comte de Lautreamont

Abstract

Conformal Geometric Algebra (CGA) provides a unified representation of both geometric primitives and conformal transformations, and as such holds significant promise in the field of computer graphics. In this chapter we implement a simple ray tracer in CGA with a Blinn-Phong lighting model, before putting it to use to examine ray intersections with surfaces generated from the direct interpolation of geometric primitives. General surfaces formed from these interpolations are rendered using analytic normals. In addition, special cases of point-pair interpolation, which might find use in graphics applications, are described and rendered. A closed form expression is found for the derivative of the square root of a scalar plus 4-vector element with respect to a scalar parameter. This square root derivative is used to construct an expression for the derivative of a pure-grade multivector projected to the blade manifold. The blade manifold projection provides an analytical method for finding the normal line to the interpolated surfaces and its use is shown in lighting calculations for the ray tracer and in generating vertex normals for exporting the evolved surfaces as polygonal meshes.

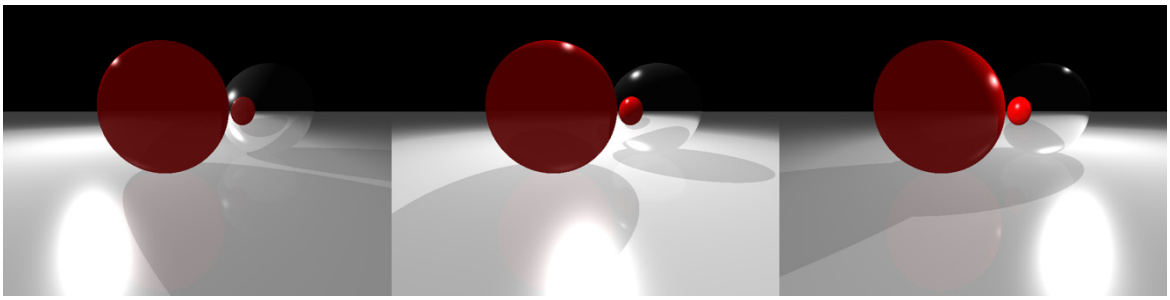


Fig. 4.1 Three images rendered with varying lighting positions. These demonstrate the lighting model, multiple light source capability and recursive tracing of rays for reflections.

4.1 Introduction

Tubular and ribbon surfaces have wide interest in fields such as neuronal modelling and streamline visualisation. The need to represent vast networks of tubular data efficiently and render these surfaces in a visually pleasing way has led to a range of different parametric representations, fitting methods and rendering techniques [83, 3, 84]. Conformal Geometric Algebra (CGA) encodes circles and line-segments, as well as planes, spheres, infinite lines and the geometric transformations between them, as natural elements of an algebra [53, 76, 29]. Given its representational power for curved surfaces and simple encoding of complicated operations, CGA appears to hold great promise in the field of Computer Graphics. Indeed several ray-tracers/path-tracers/sphere-marchers using CGA have been implemented in the past [29, 10, 12, 57, 100, 22, 25]. More recently the design of more intricate surfaces has been investigated with rotors [19] and direct-interpolation of geometric primitives as described in Chapter 3 and published in [48]. In this chapter we will press some of these techniques into use to describe tubes and ribbons as well as to develop the techniques required to render them. Figure 4.1 shows an example of output from the CGA ray-tracer we describe in this chapter.

4.2 Conformal Geometric Algebra, CGA

The ray-tracer used in this chapter is constructed using CGA and all algebraic expressions given will be in terms of elements of this algebra. CGA adds two more basis vectors, e and $\bar{e}$, to the original basis vectors of 3D Euclidean space, giving a complete basis for the 5D space with the following signature: $e_1^2 = e_2^2 = e_3^2 = e^2 = 1$ and $\bar{e}^2 = -1$. These extra basis vectors are used to define two null vectors: $n_\infty = e + \bar{e} \equiv n$ and $n_0 = \frac{\bar{e}-e}{2} \equiv -\frac{\bar{n}}{2}$ – note that the $(n, \bar{n})$ notation was that originally used when Hestenes first introduced this model in [56].

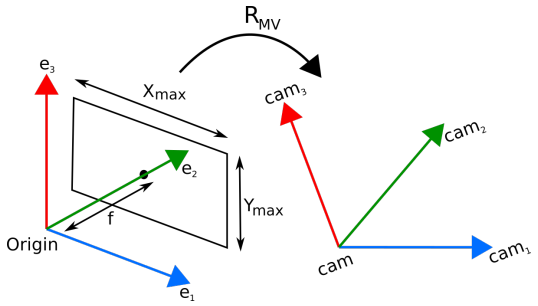


Fig. 4.2 The camera is defined by a focal length, a transformation from the origin, and bounds on the image plane.

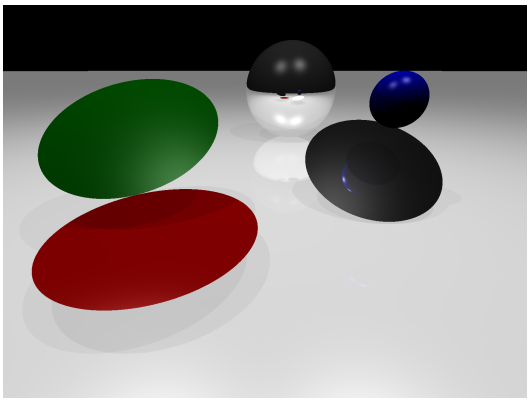


Fig. 4.3 An image from the ray-tracer containing examples of disks, spheres and planes.

The mapping from a 3D vector, x, to its corresponding CGA vector, X, is given by:

$$X = F(x) = \frac{1}{2}\left(x^2 n + 2x - \bar{n}\right) \equiv \frac{1}{2}x^2 n_\infty + x + n_0. \tag{4.1}$$

All vectors formed from such a mapping are null. CGA is chosen for the construction of the ray-tracer since we seek neat expressions for describing intersections, reflections and lighting models, made possible in CGA since rays and scene objects are both elements of the algebra. More background on CGA can be found in [53, 76, 29] as well as in the introduction to this thesis.

4.3 Camera Model and Ray Casting

A pinhole camera model is used with the geometry shown in Figure 4.2. It is defined by a rotor R_{MV} (where MV indicates *model view*) incorporating rotation and translation that takes the camera from the origin to its pose in space, a focal length f and two bounds $x_{\max}$ and $y_{\max}$ on the size of the image plane.

We take $(i, j) = (0, 0)$ to be at the bottom left hand corner of the image. For an image of width w and height h, the world coordinates of the point P_{ij} at the centre of pixel (i, j) are given by:

$$\begin{aligned} W_{i,j} &= F(fe_2 - \frac{x_{\max}}{2}(1-(2i/w))e_1 - \frac{y_{\max}}{2}(1-(2j/h))e_3), \\ P_{ij} &= R_{MV} W_{i,j} \tilde{R}_{MV}. \end{aligned} \tag{4.2}$$

We then generate the ray from the camera centre, L_{ij}, that passes through P_{ij}, via the expression

$$L_{ij} = X_0 \wedge P_{ij} \wedge n_\infty$$

where X_0 is the origin transformed by the model-view rotor R_{MV} to the position of the camera.

4.4 Ray Geometries for Basic Objects

Initially we will start with some basic objects representable as blades in CGA. The ray-tracer will thus initially concentrate on rendering planes, spheres and circles/discs, an example of which is shown in Figures 4.1 and 4.3.

4.4.1 Ray-Object Intersections

In order to compute intersections between blades, the *meet* operator ($\vee$) is used. We will, for the proposes of this chapter, always take the meet with respect to the full 5D space rather than to the join of the blades. Thus, if X is an r-grade blade, Y is an s-grade blade, and the number of basis vectors in the algebra is n, then :

$$X \vee Y = \langle XY \rangle_{2n-r-s} I_5, \tag{4.3}$$

where $\langle Z \rangle_m$ indicates the m-grade component of the multivector Z, and I_5 represents the 5D pseudoscalar of the algebra.

Planes

A plane is a 4-blade and a ray is a 3-blade so the meet gives a 2-blade. If the meet itself is 0, the line lies in the plane. If the meet squared is 0, there is no finite intersection. Otherwise, the intersection point, X, of a line L with a plane Φ, satisfies the following: $L \vee \Phi = \lambda X \wedge n_\infty$ [76], where λ is a scalar. When extracting the 3D intersection point x, we need to account for the sign and magnitude of the line in our extraction, we can do this via the constant of proportionality λ:

$$L \vee \Phi = \lambda X \wedge n_\infty = \lambda x \wedge n_\infty - \lambda n_\infty \wedge n_0. \tag{4.4}$$

Therefore, x can be extracted from the $e_i e$ and $e_i \bar{e}$ coefficients (for $i \in \{1,2,3\}$) by dividing by λ, the $e\bar{e}$ coefficient.

Spheres

Spheres are also 4-blades and so once again, taking the meet with a ray gives a 2-blade, F. With spheres, there can be zero, one or two points of intersection corresponding to the cases where $F^2 < 0$, $F^2 = 0$ and $F^2 > 0$ respectively.

If $F = A \wedge B$ (with A and B null vectors) and $F^2 \geq 0$, the points can be extracted from the point pair/blade, F, by the following formula [76]:

$$\begin{aligned} \sigma_a A &= \left(1 - \frac{F}{\sqrt{-F\tilde{F}}}\right)(F \cdot n_\infty), \quad \sigma_b B = \left(1 + \frac{F}{\sqrt{-F\tilde{F}}}\right)(F \cdot n_\infty), \\ A &= \frac{-\sigma_a A}{(\sigma_a A) \cdot n_\infty}, \qquad\qquad B = \frac{-\sigma_b B}{(\sigma_b B) \cdot n_\infty}, \end{aligned} \tag{4.5}$$

where σ_a, σ_b are scalar constants. If we define $F = A \wedge B$ with F oriented in the same direction as our ray L, then A is the point closest to the origin of the ray, P_0, as long as our sphere is 'in front of' the ray source. To ensure the alignment we can pre-normalise our sphere S via the following expression:

$$S \longrightarrow -\frac{S}{S^* \cdot n_\infty}. \tag{4.6}$$

For any given sphere, its dual can square to a positive or negative number; however, by carrying out the normalisation in equation 4.6, we ensure that all spheres, S, satisfy $S^* \cdot n_\infty = -1$. If the meet of a ray L and a normalised sphere S, is then formed from $L \vee S$, as in equation 4.3, the resulting bivector will be ordered as $A \wedge B$ where A is the point that the ray hits first in its orientation.

In figure 4.4, the meet of the ray (direction as shown) from a point P_0 with the smaller sphere, would result in the point pair $A_2 \wedge B_2$. For the larger sphere in figure 4.4, the point pair resulting from the meet will be $A_1 \wedge B_1$. We extract the points from the point pair and form the distance between these points and P_0 (via taking the inner product). We then see that for the smaller sphere the distance of the first point is less than that of the second point, whereas for the larger sphere, the distance of the first point is larger than that of the second point – which will therefore lead us to label the larger sphere as being 'behind' the point P_0. This allows us to perform bounces only with spheres that are in front of the ray origin point P_0.

Circles/Discs

Circles are 3-blades and so the meet with a ray gives a 1-vector, Y.

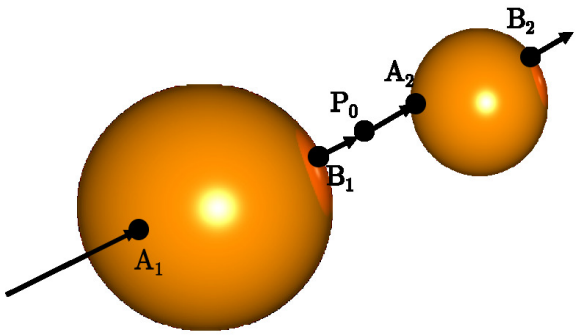


Fig. 4.4 Image showing the positions of intersection points with spheres.

- If Y itself is zero the ray (or line) lies in the plane of the circle and either does not intersect the circle or intersects the circle in one or two points.
- If $Y^2 < 0$, the ray does not lie in the plane of the circle and passes through the circle disc but does not intersect.
- If $Y^2 > 0$ the line does not lie in the plane of the circle and passes outside the circle disc without intersecting.
- If $Y^2 = 0$ (and $Y \neq 0$) the ray intersects the circumference of the circle.

Figure 4.5 shows an example of each case along with a geometric interpretation of the form of the meet. If $Y \neq 0$ and there is an intersection, the plane containing the circle is formed by taking the wedge product between the circle and n_∞, $C \wedge n_\infty$, and the intersection point is then extracted from the ray and this plane.

If $Y = 0$, so that the ray lies in the plane of the circle, we need to work in 2D, so that our 'meet' will result from taking the 2-part of the geometric product and dualising (with respect to the plane of the circle) to give a bivector. If the bivector has negative square there are two intersections at points A and B, so the bivector is $A \wedge B$. If the bivector has positive square, there is no intersection. If the bivector squares to zero there is one intersection at A, and the bivector is $a \wedge n_0$, where $A = F(a)$. In all cases, the intersection points are easily extracted.

4.4.2 Extracting Normals and Reflecting Rays

Extracting the normal to the surface of an object at a ray intersection point, X, and the reflection of that ray at X, are two fundamental building blocks in our ray tracer.

For a plane Φ which intersects with a ray, we can compute the reflection L' of an incident ray L (we assume Φ and L have been normalised such that $\Phi^2 = -1$, $L^2 = 1$) with the plane

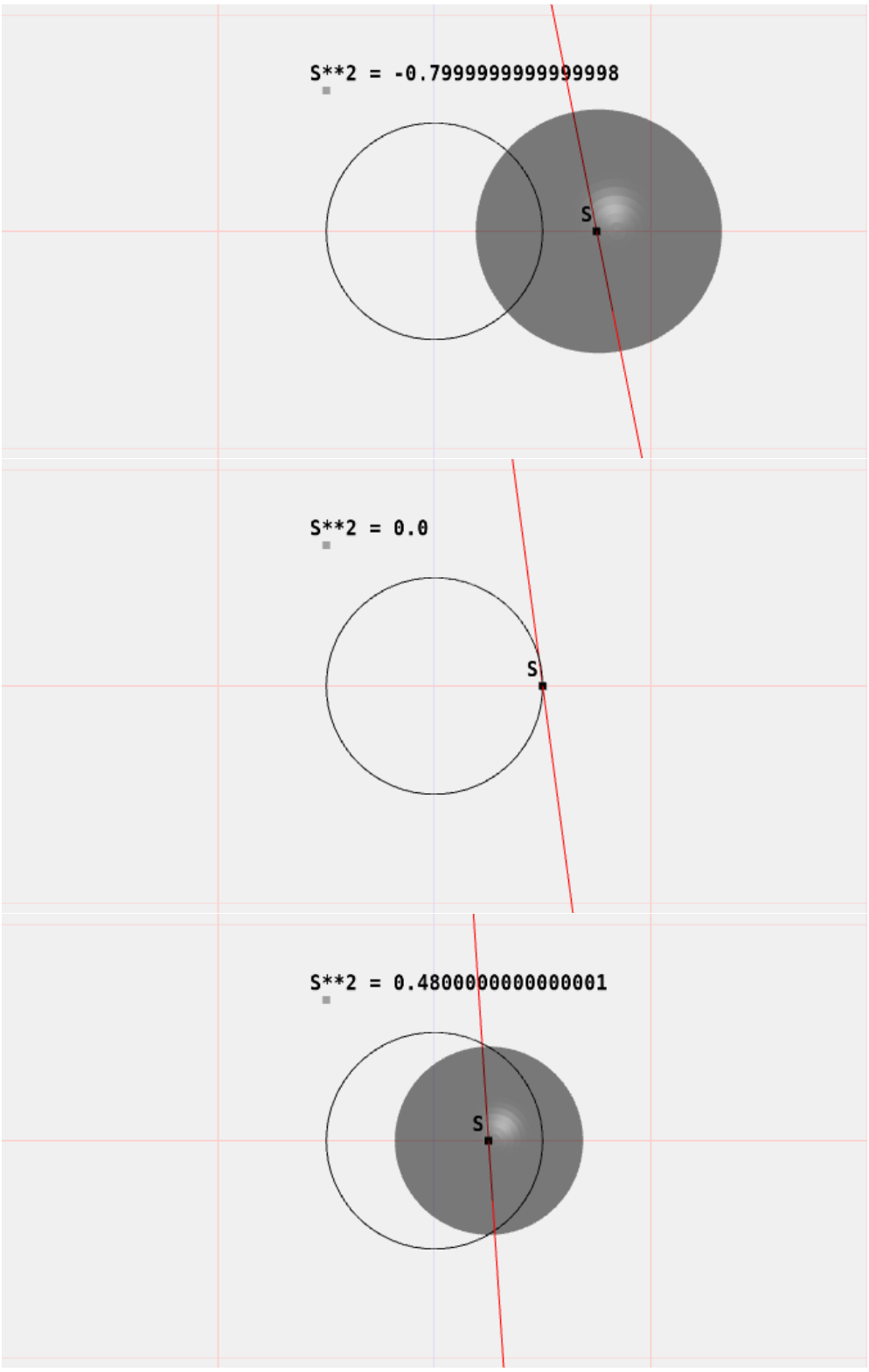


Fig. 4.5 The 5D meet of a ray, L (in red) and a circle C (black) results in a vector Y whose dual is the sphere S (grey), the properties of this sphere vary with the relative positioning of C and L. In the top case the ray passes outside of the circle, S passes orthogonally through both the circle and the ray and squares to a negative number as we would expect from a standard CGA sphere. In the middle case the ray hits the perimeter of the circle and the meet squares to zero, in this case the dual sphere is the special case of zero radius, it represents the intersection point itself. In the bottom case the ray passes inside the circle. Here the sphere squares to a positive scalar implying it is now an imaginary sphere and in fact the circle passes through the sphere's antipodal points.

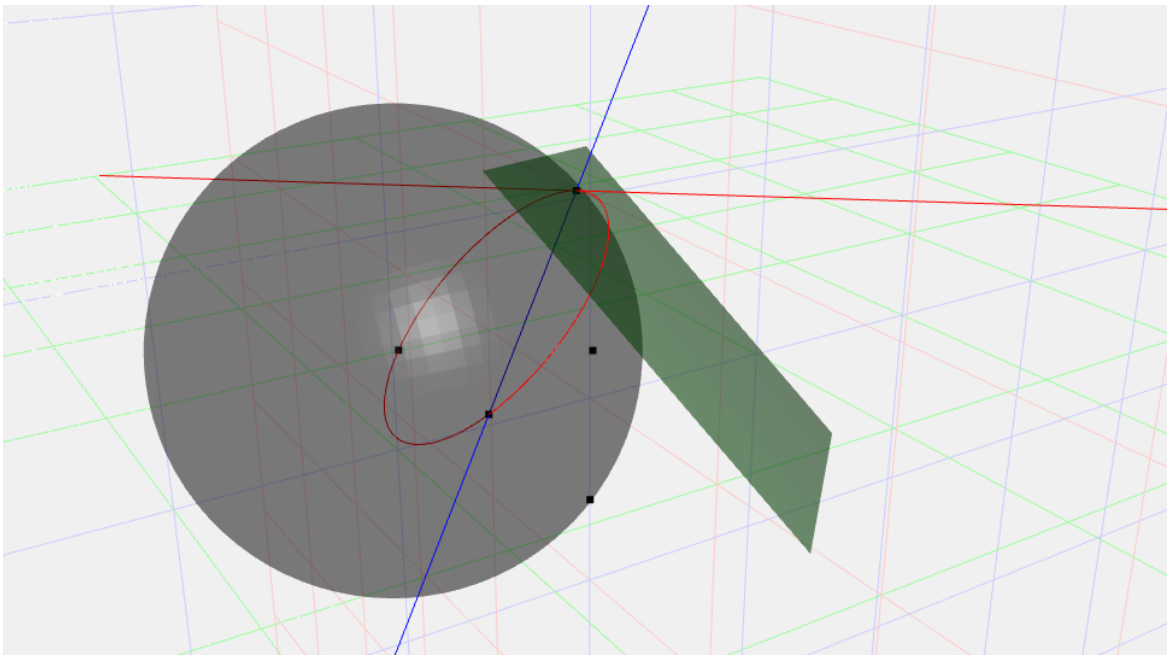


Fig. 4.6 A ray in blue hits a sphere. The tangent plane at the point of impact is shown in green. The inversion of the ray in the sphere produces the red circle. The reflection of the ray in the tangent plane gives the red line. The reflected ray is also the tangent to the red circle at the point of impact.

by simple sandwiching: $L' = \Phi L \Phi$. The resulting line is oriented correctly, passes through the intersection point and $L'^2 = 1$. For the case of a sphere S, we use the following formula from [76]:

$$L' \propto -(X \cdot (SLS)) \wedge n_\infty, \tag{4.7}$$

where X is the first point of intersection. Here, SLS is an example of an *inversion*, where the incoming ray/line, L, is inverted in the sphere to give a *circle* which passes through the two points of intersection and the origin of the sphere (note we only have a meaningful reflection if there are two points of intersection). The tangent line to this circle at the first point of intersection, X, is the reflected ray, L'. Figure 4.6 illustrates this geometrical construction. Note that one can also form the tangent plane at X and reflect L in this plane; this is performed by the following formula:

$$L' \propto \Phi_X L \Phi_X, \quad \Phi_X = (X \cdot S) \wedge n_\infty. \tag{4.8}$$

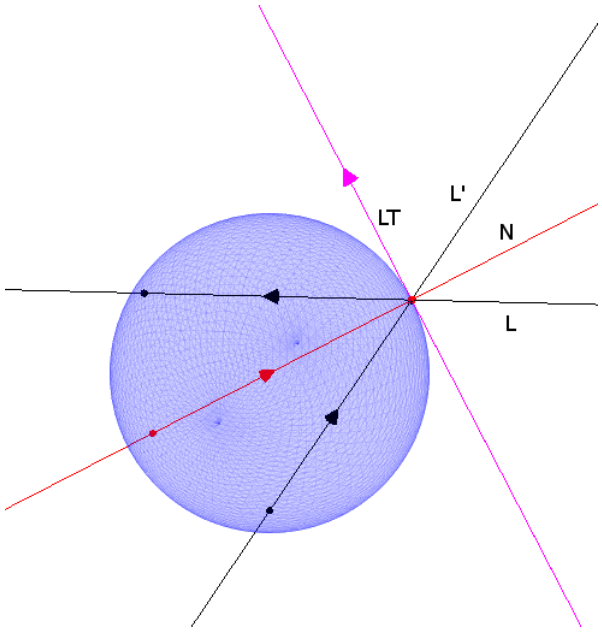


Fig. 4.7 An incident ray L (black) hits a sphere from the right hand side of the figure. The reflected ray L' (also black) scatters to the top right corner of the figure. For L and L' normalised such that $L^2, L'^2 = 1$, the normal N (red) to the surface is proportional to $L' - L$. The tangent line L_T (pink) in the plane containing the incident and reflected rays can be found from $L_T \propto L' + L$.

For a circle/disc C, we first form the plane $C \wedge n_\infty = \Phi$ in which it lies. If the ray intersects the disc (see section 4.4.1), the reflected ray can then be found using the same formula as for the plane reflection case: $L' \propto (C \wedge n_\infty) L (C \wedge n_\infty)$. Note that these expressions specifically give the (correctly oriented) reflected ray that passes through the point of intersection of the incident ray and the object, rather than a parallel ray at the origin.

We end this section with two very useful constructions which we will put to use later in the chapter. Firstly, consider the reflected ray, L', and the incident ray, L, both normalised such that they square to 1. The normal line to the surface of the sphere, N, can be simply found:

$$N \propto (L' - L).$$

The tangent line, T, (in the plane containing incident and reflected rays) can similarly be found from the sum

$$L_T \propto L' + L.$$

Figure 4.7 shows a graphical example of these constructions for the reflection of a ray in a sphere.

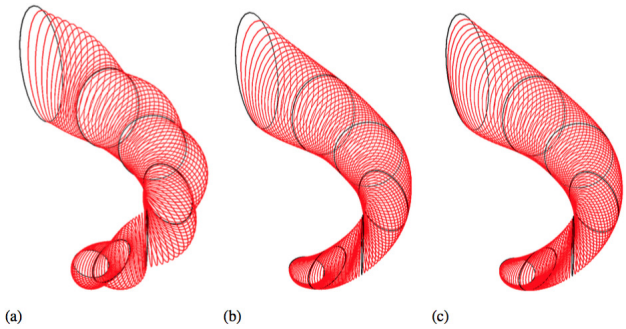


Fig. 4.8 Polynomial interpolation through circular control objects. (a) linear, (b) quadratic, (c) cubic.

4.5 Ray Tracing Evolved Circles

We will now turn to an interesting class of surface that arises from the direct interpolation of CGA circles [48], examples of which are shown in Figure 4.8. In order to generate such a surface, a direct interpolation is first performed between two boundary circles, C_1 and C_2 both of which are normalised such that $C_1^2 = C_2^2 = 1$. Our interpolation is of the form:

$$C_\alpha^{'} = \alpha C_1 + (1-\alpha)C_2, \tag{4.9}$$

where we take α moving between 0 and 1, which moves us from C_2 to C_1. The result of this interpolation is not itself a valid circle and needs to be 'projected' onto a blade via multiplication by a *projector*, which we shall call $\mathscr{S}$. This projector has *only* scalar and 4-vector parts and its construction is detailed in [48] and outlined in the following.

Consider a quantity $\Sigma = \langle\Sigma\rangle_0 + \langle\Sigma\rangle_4$. We then define the quantity $[[\Sigma]] = \sqrt{\langle\Sigma\rangle_0^2 - \langle\Sigma\rangle_4^2}$, and with this the principal square root [30] of the scalar + 4-vector, Σ, can be found as:

$$\sqrt{\Sigma} = \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}} = \frac{\langle\Sigma\rangle_0 + [[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}} + \frac{\langle\Sigma\rangle_4}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}}. \tag{4.10}$$

With this square root we can then form:

$$k\mathscr{S}^- = \sqrt{-C_\alpha^{'}\tilde{C}_\alpha^{'}}, \tag{4.11}$$

where $\mathscr{S}^-$ is $\mathscr{S}$ with the sign of the 4-vector part reversed and $\frac{1}{k} = \mathscr{S}^-\mathscr{S}$. We then construct $k\mathscr{S}$ by reversing the sign of the 4-vector part, ($k\mathscr{S} = \langle k\mathscr{S}^-\rangle_0 - \langle k\mathscr{S}^-\rangle_4$), and use this to

produce the following expression for the projector $\mathscr{S}$ and interpolated circle C_α:

$$C_\alpha = \frac{k\mathscr{S}}{(k\mathscr{S})(k\mathscr{S}^-)} C'_\alpha \equiv \mathscr{S} C'_\alpha, \quad \alpha \in [0,1]. \tag{4.12}$$

Given that these surfaces may find genuine applications in computer graphics and CAD, it is desirable to explore their properties with respect to the ray tracing framework. Specifically, for a given ray and scene object, the geometric constructions of interest for lighting models are the point of intersection between a ray and a surface, and the surface normal at that specific intersection point.

In order to render this surface, we first show how to extract the intersection point with a given ray and then how to construct the surface normal at this point.

4.5.1 Intersection Point of Ray and Interpolated Surface

We saw earlier that the intersection of a ray with a circle produces the 1-vector Y. If $Y = 0$ the ray lies in the plane of the circle and if $Y \neq 0$ and $Y^2 = 0$ there is one intersection. Therefore (in the case where the meet is not zero) to find the intersection point between our interpolated surface and a ray L, we need to find a value of α for which:

$$(C_\alpha \vee L)^2 = 0 \implies \langle C_\alpha L \rangle_4^2 = 0.$$

The system must also be tested for the case of $Y = 0$; if an α exists such that $(C_\alpha \vee L) = 0$, the ray may intersect C_α once, twice or not at all.

Figure 4.9 provides a simple visual illustration of one example of the shape of this curve as a function of α. While this example shown in the figure is particularly smooth, experiments indicate that in the general case this function is not well approximated by low order polynomials.

Non-linear Intersection Point Finder

As it is in general difficult to extract a closed form expression for the solution to $(C_\alpha \vee L)^2 = 0$, it is necessary to design an iterative algorithm to find the roots of the equation. Our implemented algorithm works as follows:

1. Check for intersection with a sphere enclosing the entire surface
2. Calculate the value of $(C_\alpha \vee L)^2$ at N intermediate values of α
3. Record where $(C_\alpha \vee L)^2$ changes sign between successive evaluated values of α

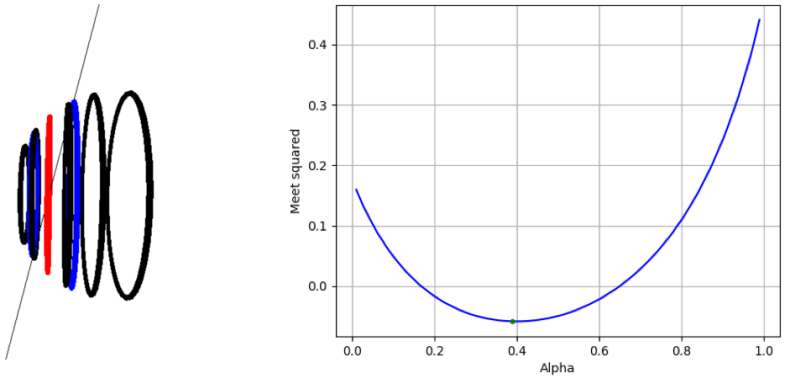


Fig. 4.9 Left: An image showing an example interpolated surface and a ray passing through it, the circles in blue show the circles which have a meet squared of 0 with the incident ray, the red circle shows where the meet squared is minimised. Right: A plot showing the value of the meet squared as a function of α for this case.

4. Locally approximate $(C_\alpha \vee L)^2$ as a quadratic equation in the region of the sign change and solve to get the value of α at the intersection point

Computing the intermediate objects in the surface can be done once per scene and reused for all rays calculated in that scene. To generate the enclosing sphere again we reuse the intermediary objects, in the following way:

1. Given C_1 and C_2, form intermediate circles, C_α, and then calculate the bounding sphere which is given by $S_\alpha = I_5 C_\alpha (C_\alpha \wedge n_\infty)$

2. Construct a sphere that contains all intermediate circle bounding spheres by successive application of a two sphere bounding algorithm

Again the enclosing sphere of the object can be calculated once per scene and used for all rays. Any two sphere bounding algorithm can be used, here we chose the algorithm from [58] which is summarised as follows:

1. Ensure both spheres S_1, S_2 are normalised according to equation 4.6

2. Construct the line L joining the centres of both spheres $L = (S_1 I_5) \wedge (S_2 I_5) \wedge n_\infty$

3. Intersect the line with the first sphere to produce a point pair $L \vee S_1 = F_1 \propto A_1 \wedge B_1$ and extract A_1 using equation (4.5)

4. Intersect the line with the second sphere to produce a point pair $L \vee S_2 = F_2 \propto A_2 \wedge B_2$ and extract B_2 using equation (4.5)

5. Check if $B_2 \cdot (S_1 I_5) > 0$, if so S_1 encloses S_2 and so S_1 is the bounding sphere
6. Check if $A_1 \cdot (S_2 I_5) > 0$, if so S_2 encloses S_1 and so S_2 is the bounding sphere
7. If neither original sphere is enclosed by the other, the new bounding sphere is given by $\frac{1}{2}(A_1 + B_2)I_5$

This iterative algorithm for the most part performs perfectly satisfactorily. When compared against specially constructed test cases for which the intersection points are known it produces negligible error. The main downside to this solution is that it is not mathematically guaranteed to give correct results especially in the case of small numbers of intermediary objects. In practice we can pre-compute large numbers of intermediary objects before rendering, allowing us to get good approximations to the function of interest. Having said that, the more intermediate objects that are created, the more computationally expensive the process is, as the root finder has to evaluate our function at each one for each ray.

'Closed Form' Solution for the Intersection of a Ray and an Evolved Circle Surface

In this section we will use C'_α to be the interpolated circle; however we emphasise that the process outlined here will also hold for the intersection of rays with other evolved objects. The intersection of the ray, L, and the surface, $\mathscr{S}C'_\alpha$ (see equation 4.12) occurs when:

$$(L \vee [\mathscr{S}C'_\alpha])^2 = 0.$$

Writing $\Sigma = -C'_\alpha \tilde{C}'_\alpha$, this can be rewritten as:

$$\left(\frac{L \vee \left[\langle \sqrt{\Sigma} \rangle_0 C'_\alpha - \langle \sqrt{\Sigma} \rangle_4 C'_\alpha \right]}{\left(\langle \sqrt{\Sigma} \rangle_0 - \langle \sqrt{\Sigma} \rangle_4 \right) \sqrt{\Sigma}} \right)^2 = 0. \tag{4.13}$$

The denominator of this expression is never infinite (other than in the uninteresting case of $\Sigma = 0$) and so does not contribute roots. Thus we can write:

$$\left(L \vee [\langle \sqrt{\Sigma} \rangle_0 C'_\alpha] - L \vee [\langle \sqrt{\Sigma} \rangle_4 C'_\alpha] \right)^2 = 0.$$

Now expanding $\sqrt{\Sigma}$ as:

$$\sqrt{\Sigma} = \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle \Sigma \rangle_0 + [[\Sigma]]}}$$

means we can write:

$$\left(L\vee\left[\left\langle\frac{\Sigma+[[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0+[[\Sigma]]}}\right\rangle_0 C'_\alpha\right]-L\vee\left[\left\langle\frac{\Sigma+[[\Sigma]]}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0+[[\Sigma]]}}\right\rangle_4 C'_\alpha\right]\right)^2=0.$$

Again the denominator of the square root function is simply a scalar which is never infinite, thus it contributes no roots and we can write:

$$\left(L\vee\left[\langle\Sigma+[[\Sigma]]\rangle_0 C'_\alpha\right]-L\vee\left[\langle\Sigma+[[\Sigma]]\rangle_4 C'_\alpha\right]\right)^2=0.$$

The quantity $[[\Sigma]]$ is a scalar and so distributing the grade selection operators gives us:

$$\left(L\vee[\langle\Sigma\rangle_0 C'_\alpha]+[[\Sigma]]L\vee C'_\alpha-L\vee\left[\langle\Sigma\rangle_4 C'_\alpha\right]\right)^2=0.$$

Expanding this leads to:

$$\begin{aligned}0=&[L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha)]^2+[[\Sigma]]^2(L\vee C'_\alpha)^2\\&+[[\Sigma]]\left\{(L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha))(L\vee C'_\alpha)\right.\\&\left.+(L\vee C'_\alpha)(L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha))\right\}.\end{aligned}\tag{4.14}$$

To make progress on solving this we recall that $\Sigma=-C'_\alpha\tilde{C'_\alpha}$ and that for our linear interpolation of circles we have defined C'_α as:

$$C'_\alpha=\alpha C_1+(1-\alpha)C_2=\alpha(C_1-C_2)+C_2.$$

As C'_α has terms in α of order 1 we would expect Σ to have terms of order 2. Continuing on this train of thought one might suspect that it is possible to re-write equation (4.14) as a simple polynomial in α. However, it is easy to see that this is not possible due to $[[\Sigma]]$, which is a scalar polynomial in α enclosed entirely in a square root:

$$[[\Sigma]]=\sqrt{\langle\Sigma\rangle_0^2-\langle\Sigma\rangle_4^2}.$$

Thus in order to solve equation (4.14) we need to rearrange:

$$\begin{aligned}&[L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha)]^2+[[\Sigma]]^2(L\vee C'_\alpha)^2\\&\quad=-[[\Sigma]]\left\{L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha)\right\}(L\vee C'_\alpha)\\&\qquad+(L\vee C'_\alpha)\left\{L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha)\right\}.\end{aligned}\tag{4.15}$$

We can then square both sides of the equation, eliminating the square root in the process:

$$\begin{aligned}\left([L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha)]^2+[[\Sigma]]^2(L\vee C'_\alpha)^2\right)^2 \\ =[[\Sigma]]^2\left[\left\{L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha)\right\}(L\vee C'_\alpha)\right. \\ \left.+(L\vee C'_\alpha)\left\{L\vee(\langle\Sigma\rangle_0 C'_\alpha)-L\vee(\langle\Sigma\rangle_4 C'_\alpha)\right\}\right]^2. \qquad (4.16)\end{aligned}$$

Expanding this out will give a polynomial in α – it turns out that this is a scalar polynomial due to the fact that $\langle\Sigma\rangle_4 C'_\alpha$ has only trivector components. This polynomial can then be solved with any numerical polynomial solver such as finding the eigenvalues of the companion matrix [63].

For this case of linear evolution of circles we will get a polynomial of order 12, implying 12 potential roots. In reality 6 of these roots are extraneous, generated by the process of squaring to handle the square root term in $[[\Sigma]]$. Some of the 6 remaining roots may be imaginary, some may be outside of the range $0 \leq \alpha \leq 1$ and some will be spurious roots corresponding to $\mathscr{S} = 0$. To filter out the valid roots we simply take all roots between 0 and 1 and evaluate $(L\vee[\mathscr{S}C'_\alpha])^2$ at these positions, selecting the roots for which $|(L\vee[\mathscr{S}C'_\alpha])^2| < \varepsilon$ for some small ε threshold where $\varepsilon > 0$ (in our experiments $\varepsilon = 10^{-6}$ works satisfactorily).

An interesting point to note here is that we could extend this intersection finding method to C' being higher order functions of α, so long as $(L\vee[\mathscr{S}C'_\alpha])^2 = 0$. Generating such higher order splines through geometric primitives is described in section 4.8.

A Comment on Rendering Speed

For this chapter our raytracer was implemented in Python with the Clifford Library [52]. It is simply an investigative tool used as a framework in which to conduct basic research into the shapes and properties of surfaces as well as the algorithms used to render them. Of course in a production computer graphics environment, a higher performance language such as C/C++/GLSL would be required and the trade off of accuracy for speed with regard to the number of intermediary objects would need to be closely analysed. Such an analysis would require very careful benchmarking and comparison across multiple modern computer architectures and as such is beyond the scope of this chapter.

4.5.2 Analytic Form for Normals

Given the α for which the ray intersects the surface, we have both the interpolated circle, C_α, and the point of intersection X. Using the result from [76], which is also used in equation 4.7,

we extract a tangential line L_C in the plane of the circle at X:

$$L_C = (X \cdot C_\alpha) \wedge n_\infty. \tag{4.17}$$

We would now like an analytic form for the tangent to the surface corresponding to evolving the surface through an increment of α, postulating that this will be orthogonal to L_C: some future work remains to understand how these two tangent vectors are related to the directions of principal curvature. Clearly $\frac{dC(\alpha)}{d\alpha} \equiv \dot{C}_\alpha$ will be a key quantity in deriving this additional tangent vector. A first observation is that the circle and its derivative will be orthogonal to one another, i.e. $\dot{C} \cdot C = 0$, and that the geometric product is minus itself under reversion, i.e. $\dot{C}C = -C\dot{C}$ (note that here, and in what follows, we will drop the α subscript on C). This follows from the fact that $C^2 = C \cdot C = 1$ (our circles are all normalised), so that:

$$\frac{d}{d\alpha}(C \cdot C) = C \cdot \dot{C} + \dot{C} \cdot C = 0, \quad \frac{d}{d\alpha}(CC) = C\dot{C} + \dot{C}C = 0. \tag{4.18}$$

Since $C \cdot \dot{C} = -C \cdot \dot{C}$ and they are both scalars, this tells us $\dot{C} \cdot C = 0$. Using the fact that $\tilde{C} = -C$, we see that $C\dot{C} = -C\dot{C} = -(C\dot{C})\tilde{}$. As there are no 6-vector parts, this indicates that the product can only have bivector parts (this is a standard construct in many areas, the most obvious being rigid body dynamics [27]). Let us call this bivector, Ω_C:

$$\Omega_C = C\dot{C}. \tag{4.19}$$

Using the analogy with rigid body dynamics, we think of this bivector as the *angular velocity bivector* of the circles as they evolve under the parameter α. We note here that a similar construction would be possible for the other main objects that we use in CGA, since they are all normalised to 1 or 0. The null vectors representing points, X, have a constant 'length' due to normalisation, so as with the circles, we can differentiate wrt α to see that X and $\dot{X}$ are orthogonal, ie $X \cdot \dot{X} = 0$. If we were to define the 'velocity', $\dot{X}$ to be the inner product with the angular velocity bivector given in equation 4.19:

$$\dot{X} = X \cdot \Omega_C = X \cdot (C\dot{C}), \tag{4.20}$$

the condition $X \cdot \dot{X} = 0$ is satisfied since $X \cdot (X \cdot B) = (X \wedge X) \cdot B = 0$. Thus, given an X on the surface, lying on a circle with parameter α, the $\dot{X}$ defined above will preserve its length and is, we claim, the tangential direction required. In order to show this, the first thing we must do is establish that if we evolve X according to this rule, generating a quantity we call

$X(\alpha)$, then $X(\alpha)$ should lie on C_α, for all α, i.e.,

$$X(\alpha) \wedge C_\alpha = 0.$$

Differentiating this (and again dropping the subscript α for clarity) and using $\dot{X} = X \cdot (C\dot{C})$, gives

$$\begin{aligned} \dot{X} \wedge C + X \wedge \dot{C} &= 0 \\ \implies \dot{X} \wedge C \equiv \big(X \cdot (C\dot{C})\big) \wedge C &= -X \wedge \dot{C}. \end{aligned}$$

We now expand this expression using standard expansion results ($a \cdot (A_r \wedge B_s) = (a \cdot A_r) \wedge B_s + (-1)^r A_r \wedge (a \cdot B_s)$):

$$\begin{aligned} \big(X \cdot (C\dot{C})\big) \wedge C &= X \cdot \big((C\dot{C}) \wedge C\big) - (C\dot{C}) \wedge (X \cdot C) \\ &= -\frac{1}{2}\langle C\dot{C}(XC + CX)\rangle_4 \\ &= \frac{1}{2}\langle \dot{C}C(XC + CX)\rangle_4. \end{aligned} \tag{4.21}$$

The first term on the right hand side of the first line of this expansion is zero as $(C\dot{C}) \wedge C = \langle C\dot{C}C\rangle_5 = \langle -C^2\dot{C}\rangle_5 = \langle -\dot{C}\rangle_5 = 0$. Since X lies on C and so $X \wedge C = 0$, we see that $XC = CX$ which means that

$$\frac{1}{2}\langle \dot{C}C(XC + CX)\rangle_4 = \langle \dot{C}C^2X\rangle_4 = \langle \dot{C}X\rangle_4 = \dot{C} \wedge X = -X \wedge \dot{C}$$

giving $\dot{X} \wedge C = -X \wedge \dot{C}$ as required, so the proposed evolution is compatible with the constraint.

If we therefore assume that $\dot{X}$ is the direction we want, we can calculate the tangent line in this direction via:

$$L_T = \dot{X} \wedge X \wedge n_\infty. \tag{4.22}$$

The fact that lines L_C and L_T are perpendicular can be verified by showing that the quantity $L_T L_C$ has only a bivector part (see [76] for a discussion of when intersecting lines are orthogonal – if two lines L_1 and L_2 intersect at a point, then $\langle L_1 L_2\rangle_4 = 0$. In addition, if they are orthogonal, $\langle L_1 L_2\rangle_0 = 0$). If this is the case, $L_T L_C$ will reverse to minus itself.

To show this, we need to consider the reverse of $(\dot{X} \wedge X \wedge n_\infty)((X \cdot C) \wedge n_\infty)$. We will need the facts that that $XC = CX$, $\dot{X}C = -C\dot{X}$, $C\dot{C} = -\dot{C}C$, $X\dot{X} = -\dot{X}X$ and $\tilde{C} = -C$. We have shown all of these identities earlier in this section. We also need an additional fact, which is that $\dot{X}$ anticommutes with C. To see this we use another standard result

$(a \wedge (A_r \cdot B_s) = (a \cdot A_r) \cdot B_s + (-1)^r A_r \cdot (a \wedge B_s))$:

$$\dot{X} \cdot C = \left(X \cdot (C\dot{C})\right) \cdot C = X \wedge \left((C\dot{C}) \cdot C\right) - (C\dot{C}) \cdot (X \wedge C). \tag{4.23}$$

The first term on the RHS of this equation is zero as $(C\dot{C}) \cdot C = \langle C\dot{C}C \rangle_1 = \langle -C^2\dot{C} \rangle_1 = 0$, and the second term on the RHS is also zero as X lies on C so $X \wedge C = 0$. Thus $\dot{X} \cdot C = 0$ and $\dot{X}$ therefore anticommutes with C as required.

We are now in a position to expand out $(\dot{X} \wedge X \wedge n_\infty)((X \cdot C) \wedge n_\infty)$:

$$\begin{aligned}
&(\dot{X} \wedge X \wedge n_\infty)((X \cdot C) \wedge n_\infty) \\
&\qquad = \left((\dot{X}X) \wedge n_\infty\right)((XC) \wedge n_\infty) \\
&\qquad = \frac{1}{4}\left[\left(\dot{X}Xn_\infty + n_\infty\dot{X}C\right)(XCn_\infty + n_\infty XC)\right] \\
&\qquad = \frac{1}{4}\left[\dot{X}Xn_\infty XCn_\infty + n_\infty\dot{X}Xn_\infty XC\right].
\end{aligned} \tag{4.24}$$

In the above we have used the facts that $\dot{X} \wedge X = \dot{X}X$, $XC = X \cdot C$ and $X^2 = 0$. Note that the term $Xn_\infty X$ in the final line of equation 4.24 can be written as $2(X \cdot n_\infty)X$ (from the standard reflection formula and the fact that $C^2 = 0$). Reversing the final line of equation 4.24 and using the commutation and anticommutation relations discussed, it is easy to show that the reverse of $L_T L_C$ is indeed minus itself, implying it has only a bivector part, as required, meaning the lines are orthogonal. Note that this result relies crucially on the fact that $\dot{X}$ and C anticommute, which is a good indication that $\dot{X}$ lies in the right direction.

Given these two orthogonal tangent lines L_C and L_T, we can construct the plane tangent to the surface at X by computing the join of the two lines. Or, we can bypass the plane entirely and compute the surface normal line directly as:

$$N = \langle L_T L_C \rangle_2 I_5. \tag{4.25}$$

4.6 Calculating the Derivative of the Object Manifold Projection

To calculate $\dot{C}$ we must differentiate the projection onto the blade manifold of our interpolated object with respect to our evolution parameter α. We will continue to work with circles but note that the process works with the general case where C'_α is any pure-grade multivector which is a function of a scalar parameter α. Let the projection of C'_α onto the blade manifold

be given by:

$$C_\alpha = \mathscr{S} C'_\alpha$$

where $\mathscr{S}$ is our blade projector. The differential of this with respect to α is given by:

$$\frac{\partial C_\alpha}{\partial \alpha} = \frac{\partial \mathscr{S}}{\partial \alpha} C'_\alpha + \mathscr{S} \frac{\partial C'_\alpha}{\partial \alpha}. \tag{4.26}$$

Thus, any closed form expression for the derivative on the manifold will first require a closed form for the derivative of the projector $\frac{\partial \mathscr{S}}{\partial \alpha}$. Recall from equation 4.13 that we can write the projector $\mathscr{S}$ as a function of $\sqrt{\Sigma}$, where $\Sigma = -C'_\alpha \tilde{C}_\alpha$, and so

$$C_\alpha = \frac{\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4}{\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}} C'_\alpha \equiv S C'_\alpha. \tag{4.27}$$

Thus to find an expression for $\frac{\partial \mathscr{S}}{\partial \alpha}$ we will first need one for $\frac{\partial \sqrt{\Sigma}}{\partial \alpha}$.

4.6.1 Closed Form Derivative of the Square Root Operation

The closed form for the derivative of the principal square root function can be found by repeated application of the chain and product rules:

$$\frac{\partial [[\Sigma]]}{\partial \alpha} = \frac{\langle \frac{\partial \Sigma}{\partial \alpha}\rangle_0 \langle\Sigma\rangle_0 + \langle\Sigma\rangle_0 \langle \frac{\partial \Sigma}{\partial \alpha}\rangle_0 - \langle \frac{\partial \Sigma}{\partial \alpha}\rangle_4 \langle\Sigma\rangle_4 - \langle\Sigma\rangle_4 \langle \frac{\partial \Sigma}{\partial \alpha}\rangle_4}{2[[\Sigma]]}, \tag{4.28}$$

where we are using the fact that $\left\langle \frac{\partial \Sigma}{\partial \alpha} \right\rangle_g = \frac{\partial \langle\Sigma\rangle_g}{\partial \alpha}$.

$$\begin{aligned}
\frac{\partial}{\partial \alpha}\left(\frac{1}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}}\right) &= \frac{-1}{2\sqrt{2}}(\langle\Sigma\rangle_0 + [[\Sigma]])^{-\frac{3}{2}}\left(\langle\frac{\partial \Sigma}{\partial \alpha}\rangle_0 + \frac{\partial [[\Sigma]]}{\partial \alpha}\right), \\
\frac{\partial \sqrt{\Sigma}}{\partial \alpha} &= \left(\frac{\partial \Sigma}{\partial \alpha} + \frac{\partial [[\Sigma]]}{\partial \alpha}\right)\left(\frac{1}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}}\right) \\
&\quad + (\Sigma + [[\Sigma]])\frac{\partial}{\partial \alpha}\left(\frac{1}{\sqrt{2}\sqrt{\langle\Sigma\rangle_0 + [[\Sigma]]}}\right).
\end{aligned} \tag{4.29}$$

Thus the derivative of $\sqrt{\Sigma}$ is a function only of Σ and $\frac{\partial \Sigma}{\partial \alpha}$ which in turn can be written in terms of C'_α and $\frac{\partial C'_\alpha}{\partial \alpha}$:

$$\Sigma = -C'_\alpha \tilde{C}'_\alpha, \qquad \frac{\partial \Sigma}{\partial \alpha} = -\frac{\partial C'_\alpha}{\partial \alpha}\tilde{C}'_\alpha - C'_\alpha \frac{\partial \tilde{C}'_\alpha}{\partial \alpha}.$$

4.6.2 Closed Form Derivative of the Projector

With our square root derivative in place we can proceed to finding the derivative of the projector $\mathscr{S}$. Recall, $\mathscr{S}$ is given by:

$$\mathscr{S} = \frac{\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4}{\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}}.$$

We can again differentiate this with repeated applications of the chain and product rule:

$$\frac{\partial\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)}{\partial\alpha} = \left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_0 - \left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_4,$$

$$\frac{\partial\left(\left[\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}\right]^{-1}\right)}{\partial\alpha} = \left[\left(\left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_0 - \left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_4\right)\sqrt{\Sigma} + \left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right] * \left[-\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}\right]^{-2}$$

and so finally we have our closed form expression for the projector derivative:

$$\frac{\partial\mathscr{S}}{\partial\alpha} = \left(\left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_0 - \left\langle\frac{\partial\sqrt{\Sigma}}{\partial\alpha}\right\rangle_4\right)\left[\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}\right]^{-1} + \left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\frac{\partial\left(\left[\left(\langle\sqrt{\Sigma}\rangle_0 - \langle\sqrt{\Sigma}\rangle_4\right)\sqrt{\Sigma}\right]^{-1}\right)}{\partial\alpha}. \tag{4.30}$$

Now consider C'_α to be an interpolated circle of the form $C'_\alpha = C'_\alpha = \alpha C_1 + (1-\alpha)C_2$. The derivative of this with respect to α is a constant:

$$\frac{\partial C'_\alpha}{\partial\alpha} = C_1 - C_2.$$

This derivative is the final piece required for equation (4.26), giving us a completely closed form for $\frac{\partial C_\alpha}{\partial\alpha} \equiv \dot{C}$.

An important point to note here is that this blade projection derivative is grade-agnostic and so can be used for objects other than just evolved circles.

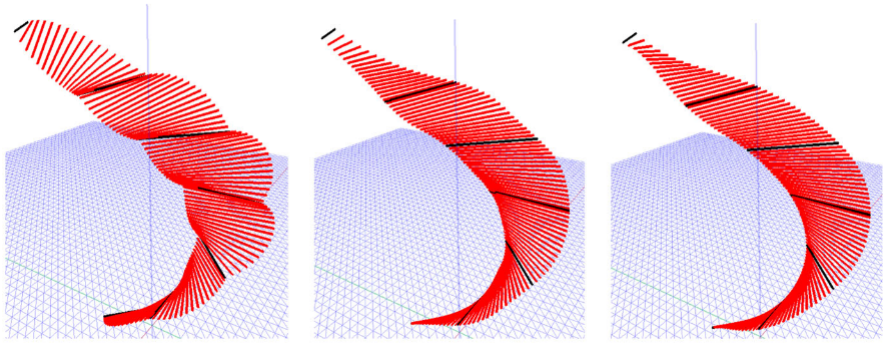


Fig. 4.10 Polynomial interpolation through point-pair control objects. From left to right: linear, quadratic, cubic.

4.7 Ray Tracing Evolved Point Pairs

We will return to the actual ray tracing of circles later (see Figures 4.16, 4.17), but first we turn our attention to point pairs. Due to the mathematical similarities between circles and point-pairs in CGA [29], as well as the practical desire to represent ribbon-like surfaces, we can apply similar ray-tracing methods to surfaces formed from the interpolation of point-pair bivectors representing line segments. If P_1 and P_2 are point-pairs which represent a line segment, we form a surface via:

$$P_\alpha = \mathscr{S} P'_\alpha = \mathscr{S}(\alpha P_1 + (1-\alpha)P_2),$$

where again, $\mathscr{S}$ is a scalar plus 4-vector which maps the interpolated bivector onto a 2-blade. Figure 4.10 gives examples of such surfaces.

4.7.1 Closed Form Solution for the Intersection of a Ray and an Evolved Point-Pair Surface

To find the intersection point of a ray and these surfaces we again form the meet of a ray L and the form of the evolved point-pair P_α. The result is a scalar quantity that can be written as:

$$L \vee P_\alpha \equiv (L^* \wedge P_\alpha^*) I_5.$$

In the case that the ray and the line that passes through both of the points in the point-pair in the surface (also known as the carrier line) intersect, this will give an answer of zero:

$$L \vee P_\alpha = 0.$$

As with the evolved circle surfaces we will attempt to construct this as a simple polynomial in α. We start with:

$$L \vee [\mathscr{S} P'_\alpha] = 0.$$

Expressing $\mathscr{S}$ in terms of Σ where as before, $\Sigma = -P'_\alpha \tilde{P}'_\alpha$, gives

$$L \vee \left[\frac{\langle \sqrt{\Sigma} \rangle_0 - \langle \sqrt{\Sigma} \rangle_4}{\left(\langle \sqrt{\Sigma} \rangle_0 - \langle \sqrt{\Sigma} \rangle_4 \right) \sqrt{\Sigma}} P'_\alpha \right] = 0.$$

As before, the denominator cannot usefully be zero, giving:

$$\begin{gathered} L \vee [\langle \sqrt{\Sigma} \rangle_0 P'_\alpha] - L \vee [\langle \sqrt{\Sigma} \rangle_4 P'_\alpha] = 0, \\ L \vee \left[\left\langle \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle \Sigma \rangle_0 + [[\Sigma]]}} \right\rangle_0 P'_\alpha \right] - L \vee \left[\left\langle \frac{\Sigma + [[\Sigma]]}{\sqrt{2}\sqrt{\langle \Sigma \rangle_0 + [[\Sigma]]}} \right\rangle_4 P'_\alpha \right] = 0, \\ \left(L \vee [\langle \Sigma \rangle_0 P'_\alpha] + [[\Sigma]] L \vee P'_\alpha - L \vee \left[\langle \Sigma \rangle_4 P'_\alpha \right] \right)^2 = 0. \end{gathered}$$

as again, the denominator is not zero.We now take the term containing $[[\Sigma]]$ (a scalar), to the RHS;

$$L \vee [\langle \Sigma \rangle_0 P'_\alpha] - L \vee \left[\langle \Sigma \rangle_4 P'_\alpha \right] = -[[\Sigma]] L \vee P'_\alpha.$$

Squaring then gives us:

$$(L \vee [\langle \Sigma \rangle_0 P'_\alpha] - L \vee \left[\langle \Sigma \rangle_4 P'_\alpha \right])^2 = [[\Sigma]]^2 (L \vee P'_\alpha)^2.$$

allowing us to form a simple scalar polynomial in α (we can see that this produces a scalar equation since $\langle \Sigma \rangle_4 P'_\alpha$ has only bivector parts):

$$(L \vee [\langle \Sigma \rangle_0 P'_\alpha] - L \vee \left[\langle \Sigma \rangle_4 P'_\alpha \right])^2 - [[\Sigma]]^2 (L \vee P'_\alpha)^2 = 0, \tag{4.31}$$

which can again be solved with a fast numerical polynomial solver.

This intersection equation for linearly interpolated point pairs is of order 6, implying there are up to 6 potential hitting points. Again the same process can be used to filter the roots as was done for the roots of the circle intersection equation.

4.7.2 Bounding Sphere and Normal Calculation

We saw earlier that the meet will be zero if the ray hits anywhere along the carrier line of the point-pair $L_C = P \wedge n_\infty$. Assuming the carrier line and ray do meet, the point of intersection

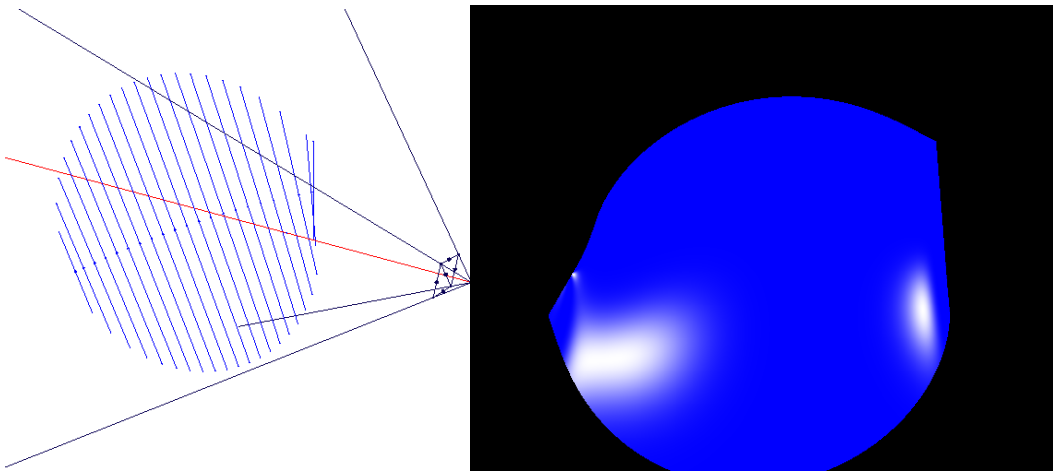


Fig. 4.11 Ray tracing evolved point-pairs. Left: The scene to be rendered, in blue is a representation of the point-pair surface to be rendered, the camera frustum is shown in black, the camera axis is shown in red. Right: The resultant rendered surface of interpolated point-pairs.

can be extracted via the method outlined in the last section of Chapter 3 and published in [48]. Given that the carrier line of P_α (for some α) and the ray intersect at a point X, we can then check if the intersection point is within the bounding sphere $S = P_1 \wedge P_2$ of the surface by ensuring:

$$S^* \cdot X = [(P_1 \wedge P_2)I_5] \cdot X > 0.$$

Since the endpoints of all interpolated point-pairs will lie on the surface of S (see [48]), the above condition ensures there is an intersection with the line segment and not just the carrier line of the point-pair. To find the normal to the point-pair surface we can simply use exactly the same argument, and in fact the same code, as we did before for the evolved circles case but this time extracting L_C as:

$$L_C = P \wedge n_\infty.$$

Figure 4.11 shows an example of rendering a surface composed of interpolated point-pairs.

4.7.3 Special Cases of Evolved Point-Pairs

A special case of the evolved point-pairs occurs when they are co-planar and form chords of a circle, Figure 4.12 shows two examples of this. As proved in [48] this special case results in the 4-vector part of the projector becoming zero implying the interpolation requires no re-projection back to the object manifold, i.e.,

$$P_\alpha = \alpha P_1 + (1-\alpha)P_2.$$

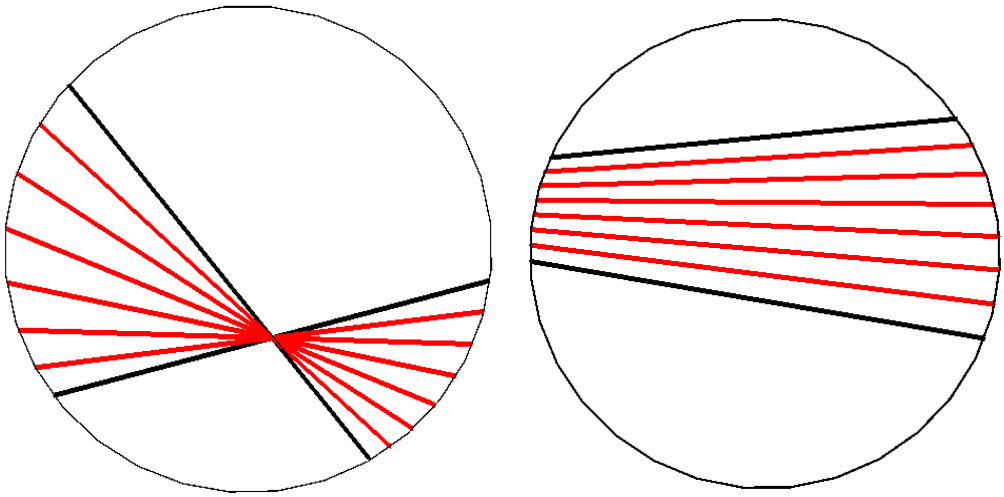


Fig. 4.12 Two examples of point-pair interpolations for which all intermediary objects are blades without requiring projection to the object manifold.

In this case the intersection of the carrier line with the ray can be found by looking for a point at which:

$$P_\alpha \vee L = \alpha P_1 \vee L + (1-\alpha) P_2 \vee L = 0.$$

Re-arranging gives an expression for α:

$$\alpha = \frac{P_2 \vee L}{P_2 \vee L - P_1 \vee L}.$$

If the α derived from the above expression is between 0 and 1 then there is an intersection of the ray with the carrier line.

The disappearing 4-vector part of the projector, which is proportional to $P_1 \wedge P_2$, allows the ray-tracer to detect these cases and thus reduces the computational expense of a ray-surface intersection considerably.

4.7.4 Triangular Facets from Evolved Point-Pairs

The intersection of co-circular point-pairs also allows us to examine the intersection of rays with triangular facets. Consider a ray L and a set of three points A, B, C which together form a triangular facet. First we will form a set of normalised point pairs:

$$P_1 = \frac{A \wedge C}{|A \wedge C|}, \quad P_2 = \frac{A \wedge B}{|A \wedge B|}, \quad P_3 = \frac{C \wedge B}{|C \wedge B|}.$$

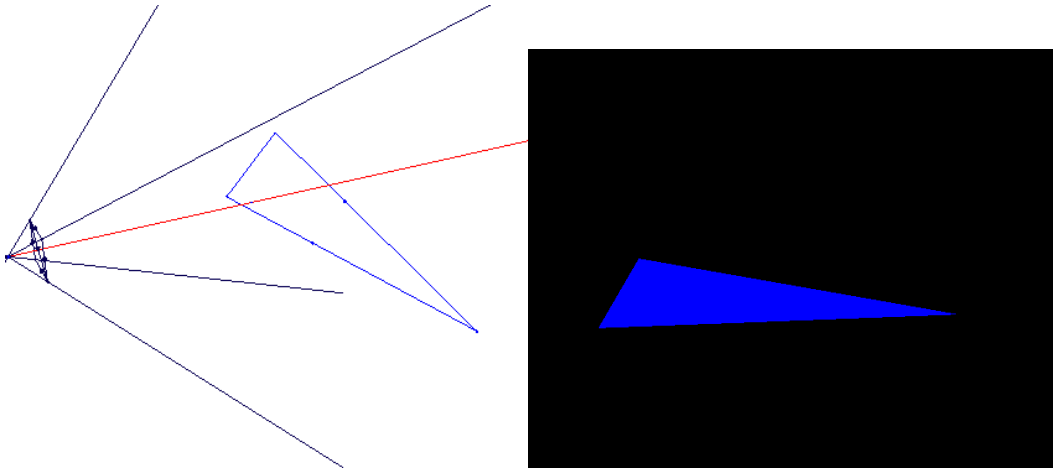


Fig. 4.13 Ray tracing a triangular facet.

We can then check if the ray intersects the facet by computing two scalar quantities

$$\alpha = \frac{P_2 \vee L}{P_2 \vee L - P_1 \vee L}, \qquad \beta = \frac{P_3 \vee L}{P_3 \vee L - P_2 \vee L}.$$

If both α and β are between 0 and 1 then the ray hits the facet. Figure 4.13 shows an example of rendering a triangular facet using this technique. It is of course possible to combine together multiple triangular facets and thus make meshes. Note that the line-facet intersection problem is not new in CGA, an alternative solution is already known via a reciprocal frame construction equivalent to barycentric coordinates and is well demonstrated in the raytracers of [76, 29].

4.8 Bézier Curves and Hermite Splines through Geometric Primitives

So far we have restricted our mathematics to linear interpolation of objects but have hinted that higher order interpolations are possible. A commonly used family of higher order interpolating curves are the Bézier curves [6], which in the cubic case and with specific first order endpoint conditions are known as Hermite curves.

4.8.1 Linear Interpolation as a Linear Bézier Curve

The simplest form of Bézier curve is simply a linear interpolation between two vectors. If we replace the vectors with k-blades and couple with the projection to the blade manifold

we have the exact same linear interpolation, although this time with α going in the other direction. Adopting a notation of C_0 as the first object and C_1 as the second:

$$C'_\alpha = (1-\alpha)C_0 + \alpha C_1, \quad C_\alpha = \mathscr{S} C'_\alpha.$$

In sections 4.5.2 and 4.6, our analysis to extract surface normals was based on having an expression for the derivative of the pure grade multivector as a function of α. For the case of the linear interpolation the solution is constant:

$$\frac{\partial C'_\alpha}{\partial \alpha} = C_1 - C_0.$$

4.8.2 Quadratic Bézier Curve

With three multivectors we can specify a quadratic function of α:

$$C'_\alpha = (1-\alpha)^2 C_0 + 2(1-\alpha)\alpha C_1 + \alpha^2 C_2.$$

This is known as a quadratic Bézier curve. Again we can take derivatives:

$$\frac{\partial C'_\alpha}{\partial \alpha} = 2(1-\alpha)(C_1 - C_0) + 2\alpha(C_2 - C_1).$$

4.8.3 Cubic Bézier Curves

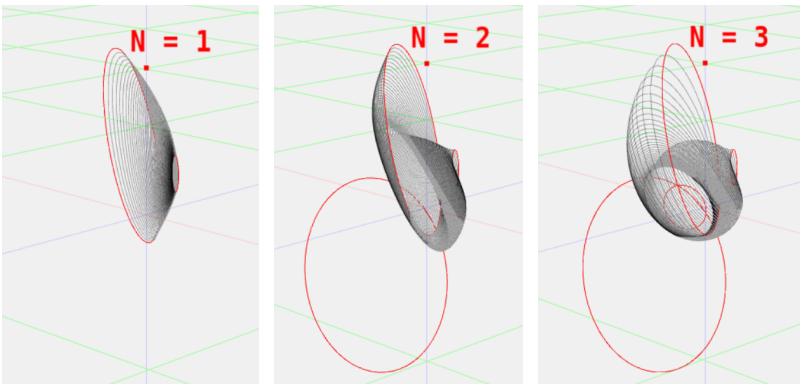
With four control multivectors we get the the most commonly used form of Bézier curve, the cubic Bézier curve:

$$C'_\alpha = (1-\alpha)^3 C_0 + 3(1-\alpha)^2 \alpha C_1 + 3(1-\alpha)\alpha^2 C_2 + \alpha^3 C_3.$$

Again we can take derivatives allowing us to extract surface normals:

$$\frac{\partial C'_\alpha}{\partial \alpha} = 3(1-\alpha)^2(C_1 - C_0) + 6(1-\alpha)\alpha(C_2 - C_1) + 3\alpha^2(C_3 - C_2).$$

Figure 4.14 shows examples of orders 1,2 and 3 Bézier interpolation through circles, along with the control objects used to generate the surface.



(a) Order 1 (b) Order 2 (c) Order 3

Fig. 4.14 Projected multivector Bézier curves of progressively higher order. The surfaces are shown in grey while the control objects are shown in red.

4.8.4 N^{th} Order Bézier Curve

More generally we can say that an N^{th} order multivector Bézier curve is of the form

$$C'_\alpha = \sum_{i=0}^{N} b_{i,N} C_i,$$

where

$$b_{i,N} = \begin{cases} \binom{N}{i} \alpha^i (1-\alpha)^{N-i}, & 0 \leq i \leq N, \\ 0, & \text{otherwise}, \end{cases}$$

are known as the Bernstein polynomials. The derivative of our N^{th} order Bézier curve is:

$$\frac{\partial C'_\alpha}{\partial \alpha} = \sum_{i=0}^{N} \frac{\partial b_{i,N}}{\partial \alpha} C_i \quad \text{where} \quad \frac{\partial b_{i,N}}{\partial \alpha} = N(b_{i-1,N-1} - b_{i,N-1}).$$

If we re-arrange our coefficients the Bézier curve derivative can also be written in the form:

$$\frac{\partial C'_\alpha}{\partial \alpha} = N \sum_{i=0}^{N-1} b_{i,N-1}(C_{i+1} - C_i).$$

4.8.5 Rational Bézier Curves

A **rational** Bézier curve adds weights w_i to the polynomials allowing them to represent a broader class of curves:

$$C'_\alpha = \frac{\sum_{i=0}^{N} b_{i,N} C_i w_i}{\sum_{i=0}^{N} b_{i,N} w_i}.$$

Again, a closed form for their derivatives with respect to α can be calculated:

$$\frac{\partial C'_\alpha}{\partial \alpha} = \frac{1}{[\sum_{i=0}^{N} b_{i,N} w_i]^2} \left(\left[\sum_{i=0}^{N} \frac{\partial b_{i,N}}{\partial \alpha} C_i w_i \right] \left[\sum_{i=0}^{N} b_{i,N} w_i \right] - \left[\sum_{i=0}^{N} b_{i,N} C_i w_i \right] \left[\sum_{i=0}^{N} \frac{\partial b_{i,N}}{\partial \alpha} w_i \right] \right). \quad (4.32)$$

Thus we can additionally represent projected multivector rational Bézier curves and calculate analytic normals to the evolved surfaces formed.

4.8.6 Hermite Cubic Curves and Splines

Hermite cubic curves are another common form of interpolating curve. They are defined by control points, C_i, (where we use the notation C for points, but will see shortly that these can be replaced by objects) and associated tangent vectors, V_i, at each end of the curve, for $\alpha \in [0,1]$:

$$C'_\alpha = (2\alpha^3 - 3\alpha^2 + 1)C_0 + (\alpha^3 - 2\alpha^2 + \alpha)V_0 + (-2\alpha^3 + 3\alpha^2)C_1 + (\alpha^3 - \alpha^2)V_1.$$

The derivative of the curve is:

$$\frac{\partial C'_\alpha}{\partial \alpha} = (6\alpha^2 - 6\alpha)C_0 + (3\alpha^2 - 4\alpha + 1)V_0 + (-6\alpha^2 + 6\alpha)C_1 + (3\alpha^2 - 2\alpha)V_1.$$

Cubic Hermite curves can be converted to cubic Bezier curves and vice-versa. As with Bezier curves, putting multivectors and multivector derivatives instead of the control points and tangents will give us a multivector valued curve.

A very common use of Hermite curves is in the construction of Hermite splines; these are piece-wise constructions in which multiple Hermite curves are placed end to end, sharing tangent vectors and control points at each endpoint. By constructing a curve in this way, a C1 continuous piece-wise curve is designed that passes through the control points exactly.

When moving the spline generation process to the multivector domain we must check whether the blade projection introduces problems with C1 continuity on the manifold. To check C1 continuity we need to evaluate the curve derivative either side of a junction between curves in the spline. Consider the form of the derivative:

$$\frac{\partial C_\alpha}{\partial \alpha} = \frac{\partial \mathscr{S}}{\partial \alpha} C'_\alpha + \mathscr{S} \frac{\partial C'_\alpha}{\partial \alpha}. \quad (4.33)$$

Let us now evaluate this at the endpoint of the n^{th} curve in a piece-wise spline where $\alpha = 1$ and of curve $(n+1)$ where $\alpha = 0$. First we note that on both curves at these points, $\mathscr{S} = 1$ because the curve passes through a blade control object which requires no projection. Additionally we see that by definition of the Hermite spline at that point, the derivative in pure-grade space is shared across both curves as is the control point:

$$\frac{\partial C'_\alpha}{\partial \alpha} = V_{n,n+1}, \quad C'_\alpha = C_{n,n+1},$$

where $V_{n,n+1}$ and $C_{n,n+1}$ are the derivative (or 'tangent') and control objects respectively of the curve that are shared between segments n and $n+1$.

Thus for the derivative to evaluate to the same on either side of the boundary, we only need to check that $\frac{\partial \mathscr{S}}{\partial \alpha}$ is the same either side of the boundary. Considering the equations in section 4.6.1 we can see that $\frac{\partial \mathscr{S}}{\partial \alpha}$ is a function only of C' and $\frac{\partial C'}{\partial \alpha}$ which are both constant across the junction. Thus the curve is C1 continuous on the manifold as required. With assurances that the spline is continuous across the boundaries we are free to chose any means of generating tangents in the pure-grade space that we like.

One such mechanism for generating tangents for Hermite splines comes from Kochanek and Bartels [72]. The Kochanek–Bartels (KB) spline is an interpolating spline with three scalar design parameters t, b, c known as tension, bias and continuity respectively. For given control objects C_i, C_{i+1} the corresponding tangents V_i, V_{i+1} can be calculated using the control objects in the spline C_{i-1} and C_{i+2} which lie previous to, and after, the curve in the order of the spline:

$$V_i = \frac{(1-t)(1+b)(1+c)}{2}(C_i - C_{i-1}) + \frac{(1-t)(1-b)(1-c)}{2}(C_{i+1} - C_i),$$
$$V_{i+1} = \frac{(1-t)(1+b)(1-c)}{2}(C_{i+1} - C_i) + \frac{(1-t)(1-b)(1+c)}{2}(C_{i+2} - C_{i+1}).$$

Setting all three scalar parameters to a value of 0 produces the commonly used Catmull–Rom spline [15]. Figure 4.15 shows an example of a KB spline of multivector geometric primitives.

4.9 Examples of Ray Tracing Simple Objects and Evolved Surfaces

Putting together the material from previous sections we can now raytrace both simple objects and evolved surfaces. Figure 4.3 shows an example of simple objects, spheres, planes and disks being rendered. Figure 4.16 shows an example of an evolved surface being rendered on

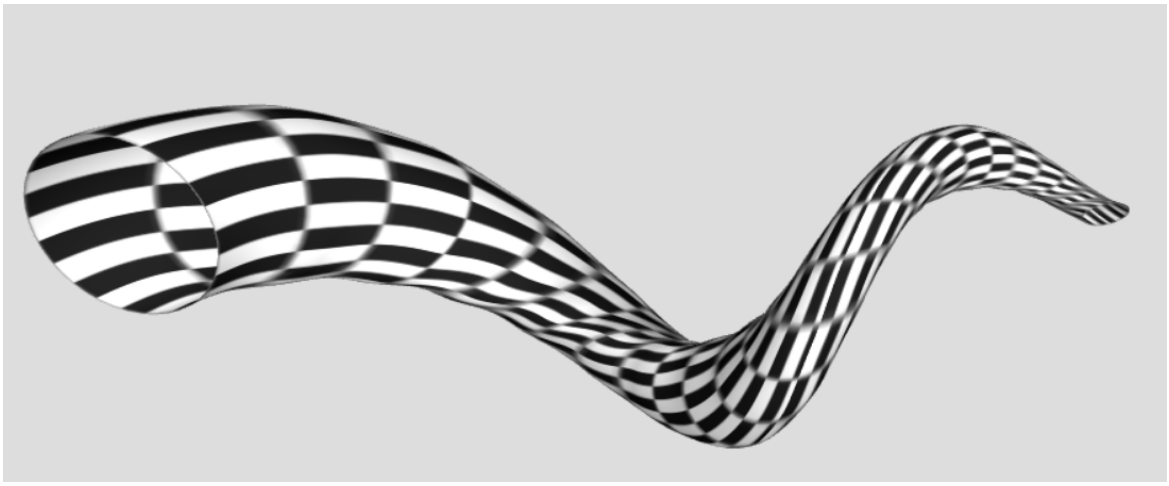


Fig. 4.15 A Kochanek–Bartels spline of evolved circles meshed, textured, and rendered with smooth shading.

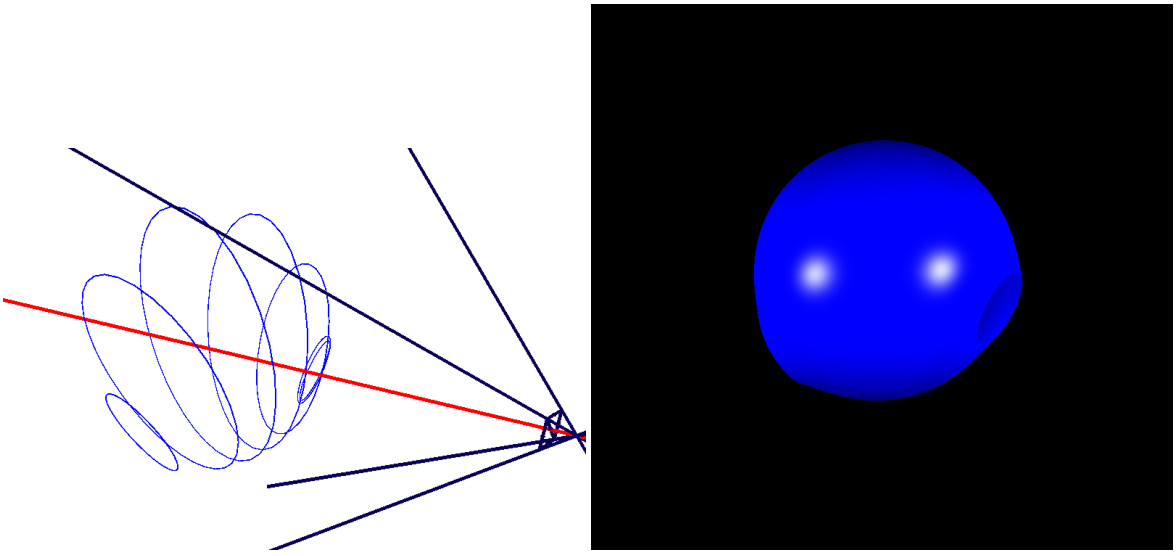


Fig. 4.16 Left: A scene composed of only an evolved surface in blue and a camera. Right: The rendering of the scene from the camera.

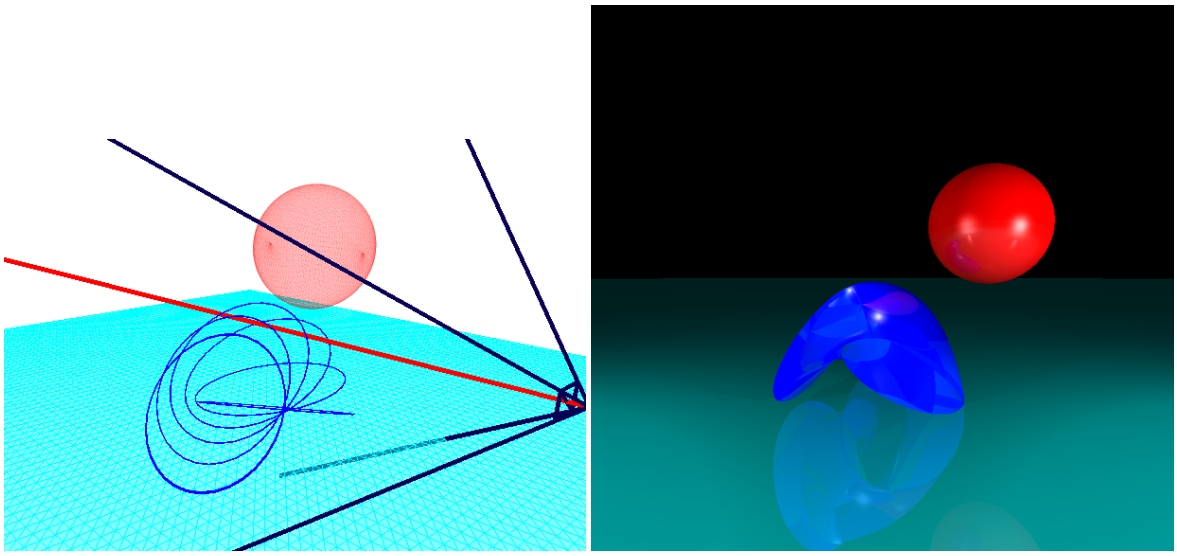


Fig. 4.17 Left: A scene composed of a ground plane in cyan, an evolved surface in blue and a sphere in red. Right: The rendering of the scene from the camera.

its own. The class of surfaces that are able to be generated with the interpolation of circles is large and Figure 4.17 shows a more unusual surface being rendered in a scene with a sphere and a plane.

4.10 Meshing Evolved Surfaces

Most graphics pipelines in modern computers use triangular meshes with some form of interpolation of vertex normals for approximating the look of curved surfaces. In light of this it is clearly desirable to be able to convert from an explicitly parameterised evolved surface to a mesh approximation of that surface.

To produce a mesh approximation we first need to generate a set of points that are in some sense evenly spaced and lie on the surface itself. To do this we will begin by producing a set of evenly spaced points on the first object C_1 and then transform these points along a small step in α to give a second set of points. Continuing in this way we can cover the surface entirely. An appropriate transformation for this task needs to preserve the relative spacing of the points on the objects in order to produce a good quality mesh. TRS (Translation Rotation Scaling) rotors have this property and can map circles to circles, spheres to spheres and point pairs to point pairs (these quantities are sometimes known as *rounds*). A TRS rotor that takes one object C_1, to another, C_2, can be calculated with the following process:

- Calculate T_1 and T_2 the translation rotors that bring C_1 and C_2 respectively to the origin

- Apply T_1 and T_2 to C_1 and C_2 respectively, bringing them to the origin and producing C_1' and C_2'
- Calculate the rotation rotor R_{12} between the blades $C_1' \wedge n_\infty$ and $C_2' \wedge n_\infty$ (if C_1 and C_2 are spheres then we do not need a rotation rotor so set $R_{12} = 1$)
- Calculate the difference in scale between the objects by extracting their relative sizes
- Use the scale to generate a dilation rotor D_{12} that scales C_1' to the same size as C_2'
- Compose the final TRS rotor Z_{12} that takes C_1 to C_2 as:

$$Z_{12} = \tilde{T}_2 D_{12} R_{12} T_1.$$

Armed with our transformation, we now simply need to generate a set of starting points on the first object. First consider the case of evolved circles. We can produce a set of N evenly spaced points on the unit circle in the e_1, e_2 plane by N successive rotations about the origin of a point X lying initially at $X_0 = F(e_1)$ yielding X_n for $n \in 0, \ldots, N$ i.e., for a fixed rotor R_θ:

$$X_n = (R_\theta)^n X_0 (\tilde{R}_\theta)^n$$

where θ is chosen so that $N+1$ uniformly spaced points cover the whole circle. With the TRS rotor Z_{01} that maps from the unit circle at the origin to the first object C_1 it is possible to transform our points to the first object:

$$U_{n1} = Z_{01} X_n \tilde{Z}_{01}.$$

Our process then becomes one of stepping sequentially through α from 1 to 0 in small increments of $\delta\alpha$ and transforming our points along the way. We will use Z_α to refer to the TRS rotor that maps C_α to $C_{\alpha-\delta\alpha}$ and so can write the n^{th} point at α as:

$$U_{n(\alpha-\delta\alpha)} = Z_\alpha U_{n\alpha} \tilde{Z}_\alpha.$$

By doing this we have effectively constructed a mapping from a coordinate system in the 2D plane of α and n to the surface manifold. This is useful as it lets us generate the mesh in the 2D plane of α and n and map the vertex positions directly to 3D.

Modern graphics engines allow users to write shaders that interpolate vertex normals in smart ways, giving the illusion of curved surfaces over flat facets. In our ray tracing experiments we have already identified how to calculate the normal to the evolved surface at any point on the surface provided that α is known at the point. The vertex normals are

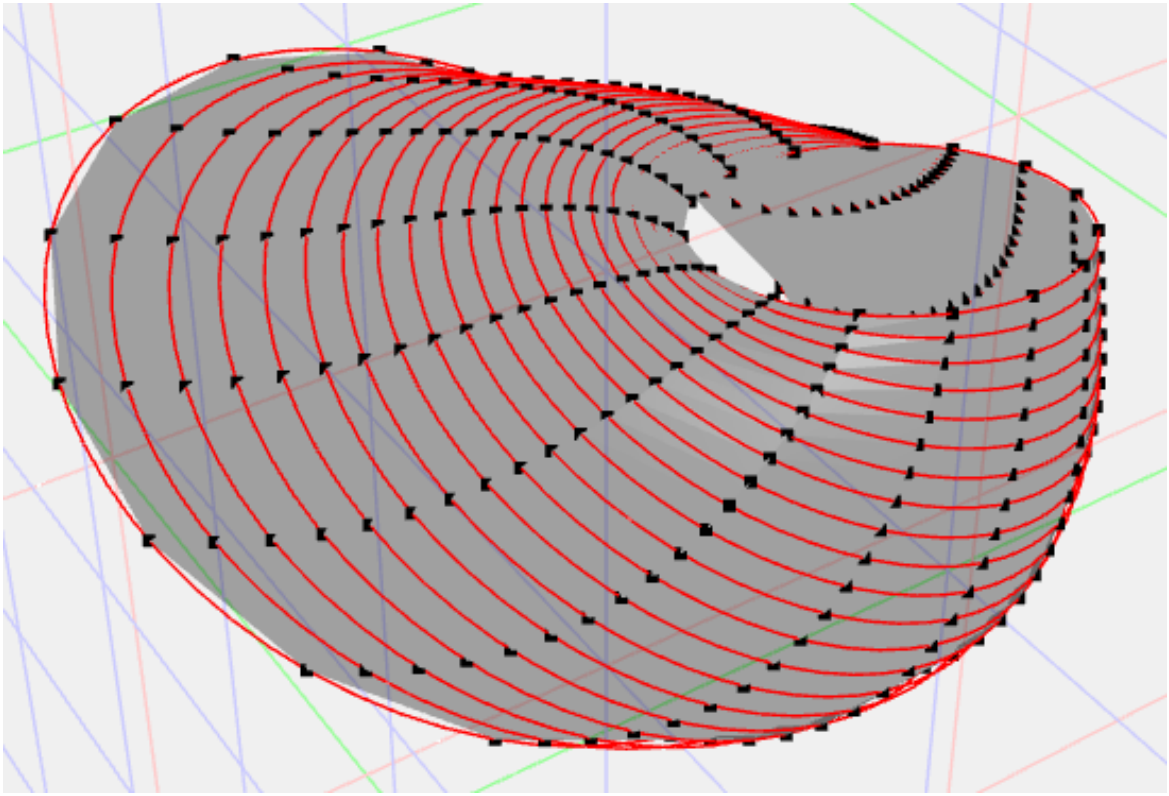


Fig. 4.18 A linear interpolation surface of evolved circles meshed and rendered with flat shading.

calculated using the formulae in the above sections. While Figure 4.18 shows a surface of evolved circles meshed and rendered using flat shading with ganja.js [23], Figure 4.15 shows a tubular KB spline surface, meshed, textured, and shaded with a smooth vertex normal interpolation scheme.

4.11 Summary and Conclusions

In this chapter we have outlined the basic workings of a CGA ray tracer that can render geometric primitives as well as more advanced interpolated surfaces defined by two circles or point-pairs and an *evolution* parameter, α. Integral to ray-tracing these evolved surfaces is the derivation of analytic intersection points and normal vectors.

Chapter 5

REFORM: Rotor Estimation From Object Resampling and Matching

We are stuck with technology when what we really want is just stuff that works.

Douglas Adams, The Salmon of Doubt

Abstract

In this chapter we tackle the problem of correspondence and rotor estimation between models composed of geometric primitives of different types. We frame this problem as searching for the rotor that takes a query model to a reference model. The situations that we consider are those in which our query model: contains additional primitives not present in the reference; is missing primitives that are present in the reference. We will also look at cases in which there are a large number of primitives per model. These are all common issues facing any SLAM-type (Simultaneous Localisation And Mapping) systems. To overcome these problems we introduce an *inter-object rotor magnitude-based matching function* and a *subsampled iterative rotor estimation and matching algorithm*. We title the finished algorithm: Rotor Estimation From Object Resampling and Matching - **REFORM**. REFORM builds on ideas from the RANSAC (RAndom SAmple Consensus) [38] and ICP (Iterative Closest Point) [5][88] algorithms and extends these to multivector correspondence. It is easily parallelisable and designed for good convergence performance with models of real objects.

5.1 Introduction

A fundamental problem in computer vision is the correspondence problem. How do we match features from one image to another? This correspondence problem also appears when dealing with 3D data; given a reference model of an object and a query model of the same object how do we match objects, identify discrepancies and extract the transformation between the models? Our reference might be, for example, a CAD model, and our query model might represent the output of fitting primitives to LIDAR data or structure-from-motion point clouds. Many authors have tackled the problem of rotor estimation between groups of pre-matched geometric objects [35] [98][95][24] and others have applied conformal geometric algebra to 3D registration of point and sphere clouds [71][4]. In this chapter we tackle the problem of registration and rotor estimation for primitives of any grade.

The objects we work with here will be CGA objects unless explicitly stated otherwise. We will use the standard extension of the 3D geometric algebra, where our 5D CGA space is made up of the standard spatial basis vectors $\{e_i\}$ $i = 1,2,3$, plus two additional basis vectors, e and $\bar{e}$ with signatures, $e^2 = 1$, $\bar{e}^2 = -1$. Two *null vectors* can therefore be defined as: $n_\infty = e + \bar{e}$ and $n_0 = \frac{e-\bar{e}}{2}$. The mapping of a 3D vector x to its conformal representation X is given by $X = F(x) = \frac{1}{2}(x^2 n_\infty + 2x - 2n_0)$.

5.2 Proximity-based matching

Our first attempt at matching models made from a collection of geometric objects comes simply from considering their locality in space. For cases in which our query model is a small displacement (where *displacement* here will refer to rotation and translation) from the reference model, we would expect that simply assigning each object in the query model to its *closest* object in the reference model would give us a good number of correct matches.

Several authors have proposed cost functions between objects [98] [95], and while many of these are extremely effective for extracting motors between circles and other round elements, they tend to fail to extract the transformation between parallel lines and planes. To counteract this problem we choose the cost function described in [35] (the properties of this cost function are further explored in [35]).

Consider first two arbitrary objects in 3D space represented as O_i and O_j in our conformal model. As in [75] we will extract the rotor R_{ij} that takes one object O_i to another O_j. Note that the objects will have an orientation (sign), and the rotor extraction will be orientation dependent. Once we have our rotor R between our conformal objects, the next step is to use

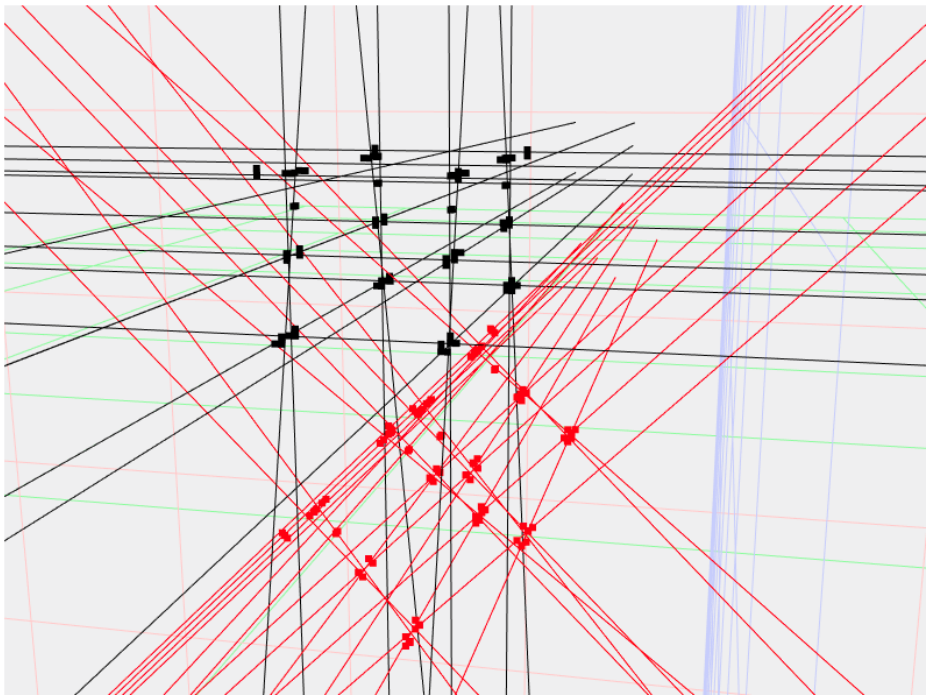


Fig. 5.1 Black: 22 lines extracted from a CAD model of a table. Red: A transformation of the original model.

this rotor to define a cost C as a function of this rotor:

$$C(R) = \langle (R-1)(\tilde{R}-1) \rangle_0 + \langle (R \cdot e)(R \cdot e)\tilde{} \rangle_0 \tag{5.1}$$

where $\langle X \rangle_r$ indicates the r-grade part of X. Equipped with this idea of closeness of objects, for a given i, a query object O_i is assigned to each of the reference objects O_j (i.e. this is done for all j), assuming the model and query sets are spatially close. For each object pair we form the rotor, R_{ij} that takes the query object to the reference object. The minimum cost assignment is then taken as the correct match, M_i, for that query object

$$M_i = \underset{j}{\operatorname{argmin}} \left[C(R_{ij}) \right]$$

Repeating this for all i, we define the total cost of this specific matching by summing the costs of each object-to-object match

$$C_{\text{total}} = \sum_i C(R_{iM_i})$$

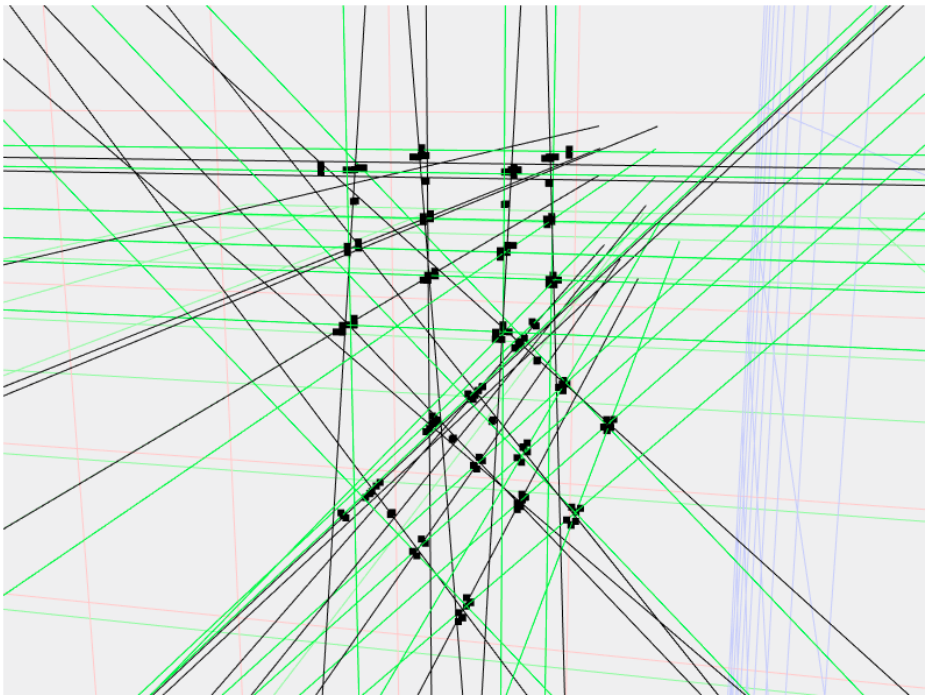


Fig. 5.2 Using a direct proximity match between objects in the example scene, the green lines are correctly matched, the black lines are incorrectly matched. In this case the method produces 11 out of 22 correct matches.

The lower this cost, the better the models are matched. Figure 5.1 shows an example scene constructed of two line-based models extracted from a CAD drawing, one model is in black and the other in red, the vertices of the models are also shown but are not used for matching. Figure 5.2 shows the result of performing proximity matching on the models, the lines in green are correctly matched and those in black are incorrectly matched. In this scene 11 of 22 lines are correctly matched by proximity matching.

5.3 Finding the rotor between two sets of matched objects

Given a set of matches for all object-pairs (under the assumption that the matching is correct) we need a method for finding the rotor between the two sets of objects. One technique for doing this is to optimise over our possible rotors, via minimisation of a cost function. Typically in CGA we parameterise and optimise over rotors in bivector space. Using the above cost metric it is shown in [35] that given correct matching we are able to perform non-linear convex optimisation and produce the correct rotation and displacement rotor. The downside of estimated gradient non-linear convex optimisation methods is that they typically require many cost function evaluations to reach the minimum, and when we have large numbers of objects in each model the optimisation can be very slow.

Here we propose an alternative algorithm, 1, based on directly using the rotors that we calculate between matched objects as part of the proximity matching procedure:

Algorithm 1: Direct rotor estimation algorithm

Result: R_e

$R_e = 1$ // The running estimate of the rotor;

for $j \leftarrow 0$ **to** *max iterations* **do**

  $R_s = 1$ // Keep track of the rotor as we iterate over all the objects;

  for $m \leftarrow 0$ **to** N *matches* **do**

    $U_m = R_e Q_m \tilde{R}_e$ // Transform query object Q_m by the current rotor R_e;

    $R_m =$ rotor between objects(U_m,O_m) // R from transformed object U_m to matched ref object O_m;

    $R_r = \sqrt{R_m}$ // Take the square root of the match rotor;

    $R_e = R_r R_e$ // Update the running estimate to use R_r;

    $R_s = R_r R_s$ // Update the rotor for this iteration;

  end

  if $R_s = 1$ **then**

    return R_e // Terminate when the full loop over the objects comes back on itself;

  end

end

This algorithm does not require the computing of an explicit cost function, it is heuristic driven and has not been proven to converge. In practice however we have found it to perform well. In the case of a fully correct matching, the rotor that is found, for both the non-linear optimisation algorithm and this direct algorithm, is indeed the rotor that takes our query model to our reference model. In the case of a partially incorrect initial matching, the rotor that is produced typically takes the query model closer to the reference model but does not produce the true rotor as shown in Figures 5.3 and 5.4.

5.4 Iterative matching and rotor estimation

Armed with rotor estimation techniques for correctly matched reference and query models we will move to more difficult situations. Consider the general case where the query and reference models are not in close proximity. In this situation we first make an initial guess at the object matches and estimate the rotor between the query and reference models using the methods described in the previous section. If our initial matching was not completely correct we will not estimate the correct rotor between the objects, the resultant rotor will have some error but will likely be relatively close to the true rotor. If we transform our query model by the estimated rotor we can use proximity matching between the transformed query model

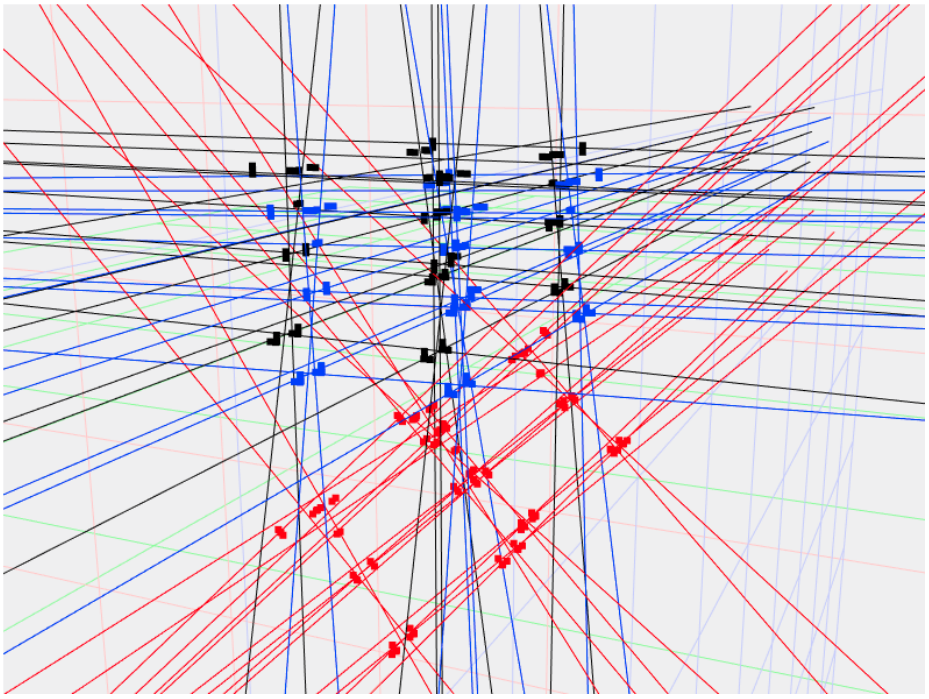


Fig. 5.3 The blue model is the estimated transformation from the set of red lines to the set of black lines (see Figure 5.1) given the starting proximity match using the non linear optimisation method. Given an initial proximity matching (see Figure 5.2), the rotor found by non-linear optimisation still puts the models in close proximity even if the initial matching is not perfect.

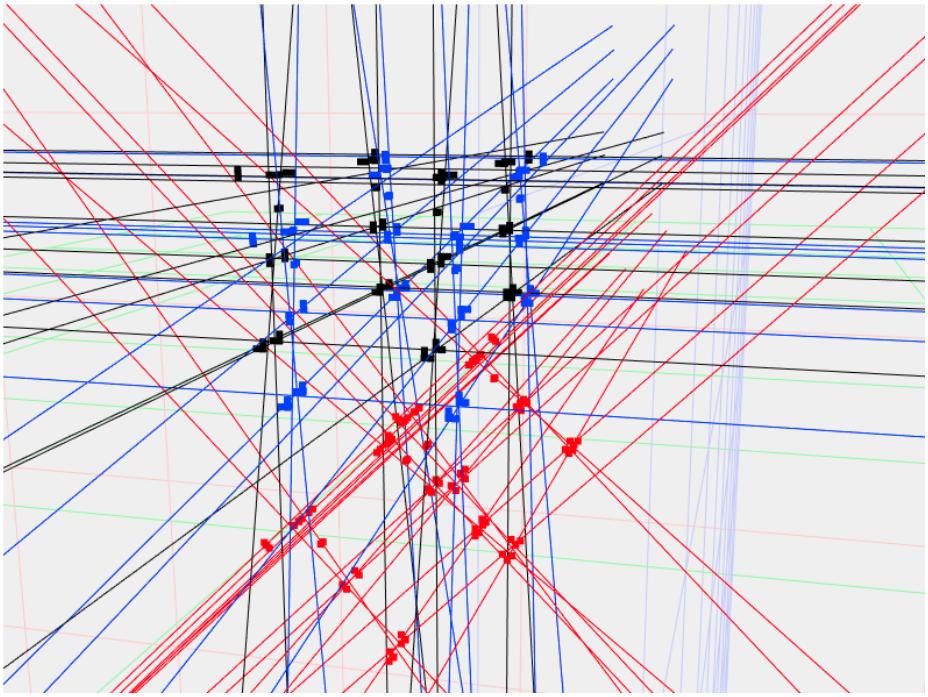


Fig. 5.4 As in Figure 5.3, the blue model is the estimated transformation from the red to the black given the starting proximity match, in this case using the direct rotor estimation algorithm. The direct rotor estimation algorithm in practice produces rotors of similar quality to the non-linear optimiser.

and the reference to get a better set of object matches. The process is then repeated so that the number of incorrect matches decreases with each iteration and the process converges. The iterative algorithm is summarised in the following:

1. Each object in the query model is given a match in the reference model (there are a number of ways of making this initial guess)
2. Calculate the rotor between the models assuming the current matches are correct, this can be done by running an optimisation algorithm to completion or by using the direct method mentioned in the previous section.
3. Transform the query model by applying the rotor calculated in the previous step
4. Each object in the transformed query model is compared to each object in the reference model, the match with the minimum cost according to our chosen cost function is accepted
5. If there is no change in the matches terminate the algorithm otherwise go back to step (2)

This algorithm correctly handles partially incorrect initial matching between models, and iterates towards the answer in relatively few steps. It is also deterministic, each step is a function only of the current state and it has fixed termination criteria that clearly indicate when it has completed. In its current state this algorithm is an extension of the well known Iterative Closest Point (ICP) algorithm [5][88] routinely used for point cloud registration. As with the ICP algorithm, a significant problem arises when we consider cases in which large fractions of the initial matches are incorrect, resulting in convergence to an incorrect set of correspondences. With our geometric algorithm we additionally see local minima arise when models contain many parallel lines or planes and computationally we run into trouble when models contain a very large number of geometric objects. In these cases the algorithm may fail to converge to the true rotor and instead become stuck in a local minimum even though some matches are correct. Real manufactured objects or buildings typically contain many parallel faces and lines and as such we need a way to overcome these limitations. Figure 5.5 shows an example of the previously studied scene stuck in a local minima, in this case there are 17 of 22 lines correctly matched but the algorithm will not progress further.

5.5 Incorporating sampling

To counteract the local minima issue, we modify our procedure to incorporate sampling in a RANSAC-like [38] algorithm. This particular approach is chosen as it is readily adapted to

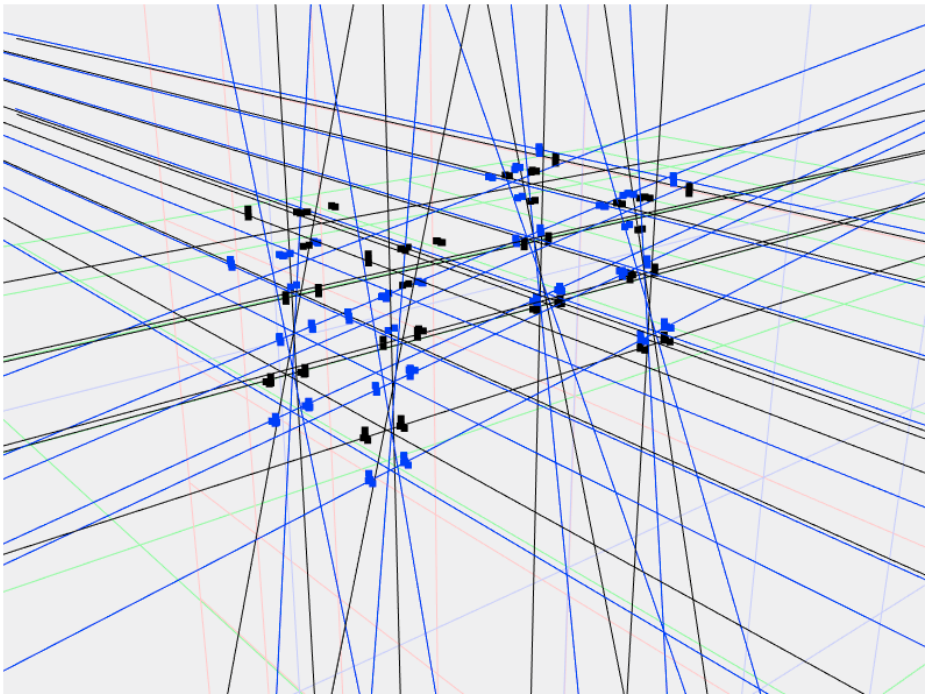


Fig. 5.5 As our model contains a lot of symmetry the iterative algorithm is prone to getting stuck in local minima. As in Figure 5.1 the black model is our reference and the blue is our estimate, the red lines are not shown for clarity. Here the blue model is at the final output of the iterative matching algorithm. 17 of 22 matches are correct but the algorithm is stuck in a local minima.

parallel processing and is well suited to handling large numbers of incorrect matches. After each matching stage in the previous algorithm we randomly and uniformly sample m lots of k matches. Each of these m match sets then propagates through the rotor estimation algorithm and each produces a candidate rotor for the model matching and a cost associated with that rotor for these k matches. The rotor produced by the sample with the minimum cost is then chosen and used to transform the entire query model. This repeats for a fixed number of iterations or until some cost threshold is reached.

The full REFORM algorithm is now summarised as follows:

1. Each object in the query model is given a match in the reference model (there are a number of ways of making this initial guess)
2. Given our matches, randomly select multiple sample subsets
3. For a given sampled subset calculate the rotor that leads to minimum total cost between the subset objects as in equation (5.1)
4. Accept the rotor from the sample that gives the minimum total cost between the subset objects
5. Update our query model position by applying the estimated rotor
6. Each object in the query model is compared to each object in the reference model, the match with the minimum cost is accepted
7. Check termination criteria, go back to step 2.

The disadvantage of moving to a sampling-based model is that we no longer have fixed termination criteria – just because the matches have not changed over multiple sampling and optimisation steps, does not mean they will not change as a result of the next one. On the other hand, the rotor estimation and cost calculation for each sample is independent of every other sample allowing for easy parallelisation. The subsampling also allows the algorithm to jump out of local minima by sampling correct matches whose effect would normally be swamped by the mass of incorrect matches. A CUDA implementation of the algorithm has been written, leveraging the massive parallelisation capability afforded by modern graphics cards and is incorporated in the clifford python package [52].

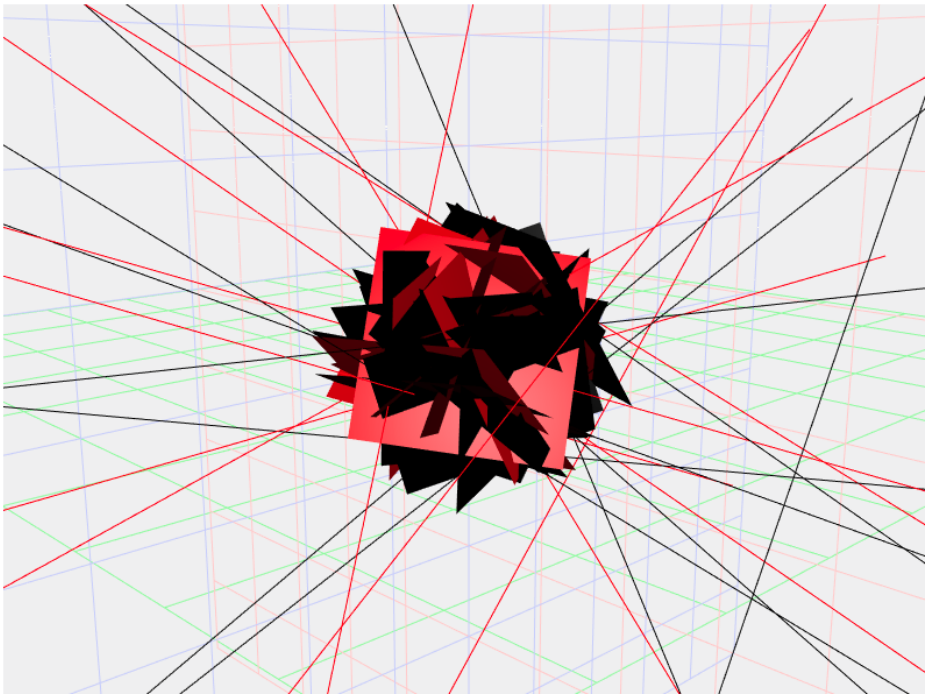


Fig. 5.6 Sets of synthetic random lines and planes in red along with their transformation in black to be matched. REFORM handles both in the same framework and correctly extracts the rotor between them.

5.6 Matching scenes of mixed geometric primitives

3D models of objects are typically constructed from collections of geometric objects, planes, lines and points. While traditional matching techniques typically use points from meshes [41] or points derived from the intersection of planes/lines [8], REFORM allows us to incorporate multiple types of 3D object together into the same matching and rotation/translation estimation framework, Figure 5.6 shows an example of two matching synthetic models composed of both lines and planes, in this example REFORM handles both types of object transparently.

5.7 Conclusions

In this chapter we have presented an algorithm for registering models composed of geometric primitives. This algorithm extends the range of traditional matching and registration algorithms from point cloud only techniques to incorporate higher grade geometric objects. The solution is available in the clifford [52] python package with both CPU and GPU implementations.

Part II
Kinematics, Dynamics and Robotics

In this Part of the Thesis we develop techniques that relate to robotics. Geometric Algebra has found relatively widespread use in the analysis of robots. GA simplifies the intersection of geometric shapes, often used in forward and inverse kinematics, and GA provides a neat framework for the manipulation of Lie algebras and groups, again often useful in forward and inverse kinematics as well as in control problems. Here we specifically look at the embedding of screw theory into Geometric Algebra and how this embedding can be used for dynamics and multi-body kinematics.

Chapter 6

Screw Theory in Geometric Algebra for Constrained Rigid Body Dynamics

We may always depend on it that algebra, which cannot be translated into good English and sound common sense, is bad algebra.

William Kingdon Clifford

Abstract

Screw Theory and Geometric Algebra (GA) are mathematical frameworks that have found wide use in the analysis of robotic mechanisms. Here we consider an embedding of screw theoretic wrenches and twists into the motor bivectors of two common GAs, the Plane-based (also known as Projective) and Conformal Geometric Algebras. We start with statics, considering the representations of forces and moments and how the products of GA map to the products of Screw Theory. Moving on from statics we construct an inertia tensor equivalent based on the concept of the principal screws of inertia and show how to transform this inertia tensor between frames of reference. We then look at the problem of geometrically constrained dynamics in two different ways, first via the familiar concept of virtual work, and secondly via a novel idea of multivector pinning between frames. Finally we consider the problem of integrating screw motions directly in the motor bivector space, describing kinematic equations for several alternative se(3) Lie algebra to Lie group mappings. Our goal in this chapter is to kill two birds with one stone: explicitly work through how Screw Theory embeds into commonly used GAs and use Screw Theoretic ideas to show the similarity

between the various approaches to statics and dynamics in GA. Along the way we will focus heavily on the geometry driving the problems we encounter, with the hope that this will shed light both on both GA and Screw Theory.

6.1 Introduction

6.1.1 Screw Theory

Screw theory was originally developed by Sir Robert Stawell Ball (1840-1913) and described in his seminal text 'A Treatise on the Theory of Screws' [2]. Screw Theory has found wide adoption in the study of 3D mechanisms as it allows a simple, unified treatment of both the rotational and translational motion of rigid bodies. The Screw Theory literature draws on multiple sources of mathematics but Lie theory and Projective Geometry feature heavily. Modern Screw Theory appears in several different forms and is referred to by various different names by different authors. From the 'spatial vector algebra' of Roy Featherstone [36, 37] to the 'Linear Line Complexes' of Helmutt Pottmann [86] the diversity of terminology reflects the diversity of fields in which Screw Theory has found applications and mathematical foundations. This chapter is an attempt to articulate a particularly elegant embedding of Screw Theory into the coordinate free language of Geometric Algebra (GA) and how this novel embedding might allow us to push the boundaries of Screw Theory in new directions.

6.1.2 CGA

Conformal Geometric Algebra (CGA) is a real Clifford algebra with signature $Cl(4,1,0)$. It is very popular for its embedding of conformal transformations (and so also the Euclidean transformations) as well as its blade representations of many geometric primitives. CGA is especially well suited for robotics as the direct circle and sphere intersections appear in many forward and inverse kinematic problems while the motions involved are typically Euclidean [50, 51]. CGA was first suggested by Hestenes [56] and indeed the same author made the first steps in uniting it with Screw Theory [54, 55]. CGA has since been applied to a huge variety of problems, for more information about this algebra the reader is directed to the excellent book of Dorst, Fontijne and Mann [29].

6.1.3 PGA

While CGA has been a very successful idea it is by no means the most computationally efficient way to embed rigid body dynamics into GA. In many applications a practitioner

is willing to lose some of the expressiveness and mathematical niceties of CGA to gain an algebra with fewer elements that nonetheless does what they desire, and crucially, does it computationally **faster**. The Plane-based (or Projective) Geometric Algebra (PGA) aims to strike a balance between efficiency and expressiveness. With a signature of $Cl(3,0,1)$ it contains the so called 'flat' elements of CGA, flat points, direction bivectors, lines, planes and the rigid body rotors, but loses point-pairs, circles, spheres, dilation and inversion rotors as first class citizens of the algebra. The use of the degenerate metric also necessitates the use of an alternate dual map rather than a simple multiplication by the pseudoscalar, in practice this dualisation operation can be implemented by the right complement extended over the canonical basis vectors by linearity. For more information on PGA the reader is directed towards the 2019 SIGGRAPH course by Gunn and De Keninck [46, 45] and Leo Dorst's 'A Guided Tour to the Plane-Based Geometric Algebra PGA' [28].

6.2 Forces, moments and static equilibrium

While this chapter is ostensibly about dynamics we will first cover force representations and static equilibrium to make sure we are developing sensible mathematics before we put things in motion.

For a rigid body to be in static equilibrium certain conditions have to be met. Specifically, there can be no net moment (torque) and no net force acting on it.

6.2.1 What is a force?

We will now do a quick survey of the different force representations, and in doing so, think about what an effective force representation requires.

3D vector representations

In many approaches to mechanics, forces are formalised as a simple 3D vector aligned with the direction of the force and a magnitude equal to the intensity of the force. In order to calculate the resultant force on a rigid body acted upon by many incident forces we take the vector sum of the force vectors acting on the body. To represent a moment in the same formalism we use a 3D vector parallel to the axis of the moment with magnitude equal to its intensity and with the convention that a positive turning force acts counter-clockwise about this axis. To calculate the moment of a force about a specific point on the body we need to combine the incident force with its incident position. If we label the force vector f, its incident point a and the point about which we aim to calculate the moment as p then we can

calculate the moment b with the cross product:

$$b = (a - p) \times f.$$

6D representations

The 3D vector representation is efficient and easy to understand, however it does not encode all the information that exists in our idea of a force. Specifically, it does not inherently include any localisation information about the force. When we want to calculate the moment due to the force about a specific point we need to include the additional information of a point on the line of action of the force. Therefore to be able to use this representation effectively we **actually** require two 3D vectors, one the 3D force vector f and one determining a point a through which the force acts. We can stack these two 3D vectors on top of each other to make a 6D vector which we will call $\mathscr{F}'$:

$$\mathscr{F}' = \begin{bmatrix} f \\ a \end{bmatrix}.$$

This object $\mathscr{F}'$ now contains all the information required to use the force in applications. It is however **not unique**. We could choose multiple values of a all of which lie on the line and they would all give different 6D representations of the force while still fundamentally behaving the same way physically. To solve this problem we need to instead use a representation of the force that is unique. If we change our representation to:

$$\mathscr{F} = \begin{bmatrix} f \\ a \times f \end{bmatrix}$$

then we get such a formulation, as all points on the line of action of the force will have the same result for $a \times f$. This 6D representation is known as the Plücker coordinates of the line [86].

In contrast to a force, a moment is not localised to a specific point or line in the world, in the terminology of physics and engineering this type of non-localised vector is known as a free vector. While respecting their non-localised nature we would like to be able to put the 3D moments in the same 6D box as forces to be able to keep the representations consistent and maybe provide some meaningful operations on them. When putting our 3D moment vector into the 6D box we are left with several different options. One option would be to put the moment vector in the top 3 positions of the 6D vector, such that they align with the f

component of the force, or maybe we should put them in the lower 3 to make them align with the $a \times f$ part of the force representation.

Initially, which choice to make seems non-obvious. To help us gain some insight we consider the problem of two anti-parallel co-planar forces of equal magnitude acting on a rigid body. First, label the forces themselves:

$$\mathscr{F}_1 = \begin{bmatrix} f \\ a_1 \times f \end{bmatrix}, \mathscr{F}_2 = \begin{bmatrix} -f \\ a_2 \times -f \end{bmatrix}.$$

In this situation, as the forces are in opposite directions and of equal magnitude, the total resultant force on the body is $f - f = 0$. But what about the total moment b_t of the forces about a point p on the body?

$$b_t = b_1 + b_2 = (a_1 - p) \times f + (a_2 - p) \times -f.$$

As the cross product is distributive we can expand this out and rearrange it giving:

$$b_t = a_1 \times f - a_2 \times f - p \times f + p \times f,$$

$$b_t = (a_1 - a_2) \times f.$$

This is an interesting result. The moment is independent of the position p on the rigid body that we have taken moments about and is equal simply to the sum of the second half of our 6D force representation. This concept of two anti-parallel forces giving rise to a pure moment is the basis of the term **couple** or **force couple** from mechanical engineering. If we were to write an external moment in the form:

$$\mathscr{B}_e = \begin{bmatrix} 0 \\ b_e \end{bmatrix}.$$

We could exploit the linear independence of the top and bottom of the 6D vector to state our static equilibrium condition as:

$$\sum_i \mathscr{F}_i + \sum_i \mathscr{B}_i = 0.$$

This is a neat result and gives us hope that continuing down the route of 6D representations will lead us to other interesting things.

In fact continuing down this route leads us to the concept of constructing an algebra over these 6D objects, known as the **algebra of screws** and more generally into the territory of

the field known as Screw Theory. The basic element of such a 'screw algebra' is a 6D vector made up of the 6D representation of a force line and the 6D representation of a moment with axis parallel to that force line. Or, more formally,

$$\mathscr{W} = \begin{bmatrix} v \\ u \times v \end{bmatrix} + \begin{bmatrix} 0 \\ hv \end{bmatrix} = \begin{bmatrix} v \\ u \times v + hv \end{bmatrix}$$

where v, u are 3D vectors and h is a scalar. Chasles' theorem states that we can, in fact, decompose any 6D vector into this form. In the language of Screw Theory pure forces and pure moments are both special cases of the more general **wrenches**, which is the name for the 6d representation of a screw representing a combination of force and moment. Screw Theory goes on to define a couple of useful products between screws. The first product is the 'reciprocal scalar product' of screws which, for two screws S and T of the form

$$\mathscr{S} = \begin{bmatrix} A \\ B \end{bmatrix}, \ \mathscr{T} = \begin{bmatrix} C \\ D \end{bmatrix}$$

gives a scalar:

$$\mathscr{S} \circ \mathscr{T} = A \cdot D + B \cdot C$$

where here the $\cdot$ operator is the standard inner product between vectors. The second product we will consider is the screw cross product, for screws S and T the screw cross product gives:

$$\mathscr{S} \times \mathscr{T} = \begin{bmatrix} A \times C \\ A \times D + B \times C \end{bmatrix}$$

where $A \times C$ and $A \times D + B \times C$ are computed with the standard 3D cross product. The result of the cross product of two screws is itself another screw.

6.2.2 Representations of wrenches in CGA and PGA

So far we have dealt entirely with the realm of forces and moments and their representation in 6D screw theory where they are known as wrenches. We have not yet touched on the main topic of the thesis, Geometric Algebra (GA). Our goal in this section will be to describe a direct mapping from our 6D screw theory representation to two of the most commonly used GAs, namely CGA and PGA.

6.2.3 Forces as dual lines in CGA

In the previous section on the screw representation we saw how we could describe a force as a directed line with a magnitude using a 6D vector. Let us now consider how we might go about representing a force line in CGA. As before consider a force line $\mathscr{F}$ expressed as 6D Plücker coordinates:

$$\mathscr{F} = \begin{bmatrix} f \\ a \times f \end{bmatrix}.$$

We could represent this same force in CGA as the outer product of two conformal points and infinity:

$$F = \mathrm{up}(a) \wedge \mathrm{up}(a+f) \wedge n_\infty \tag{6.1}$$

where we use the notation $\mathrm{up}()$ to represent the mapping of a 3DGA point to a conformal point, ie. $X = \mathrm{up}(x) = \frac{1}{2}x^2 n_\infty + x + n_0$. The object F has a magnitude equal to the intensity of the force:

$$F^2 = |f|^2.$$

Now consider the dual form of this CGA line:

$$FI_5 = fI_3 - (a \wedge f)I_3 n_\infty. \tag{6.2}$$

This looks very similar to the 6D Plücker representation, in fact if we consider it closely we can see that it is the same. First, look at the term in front, fI_3. This is the 3D dual to a 3D vector, giving a bivector, specifically the Euclidean bivector orthogonal to f. Now consider the second half of the formula: $-(a \wedge f)I_3 n_\infty$. $-(a \wedge f)I_3$ is the 3D dual to a Euclidean bivector, ie. a vector equal to $a \times f$, just like the lower 3 slots of the 6D screw representation. As $(a \wedge f)I_3$ is a Euclidean vector this makes $-(a \wedge f)I_3 n_\infty$ the form of a CGA 'direction bivector'. An important thing to note here is that the front and back part of this formula are linearly independent, just like in the 6D screw representation, ie.

$$\lambda_1 F_1 I_5 + \lambda_2 F_2 I_5 = (\lambda_1 f_1 + \lambda_2 f_2)I_3 - (\lambda_1 a_1 \wedge f_1 + \lambda_2 a_2 \wedge f_2)I_3 n_\infty.$$

Again as with the 6D representation we can consider two anti-parallel forces and extract the representation of a force couple or moment. Setting $\lambda_1 = \lambda_2$ and $f_1 = f = -f_2$ in the above leaves us with:

$$F_1 I_5 + F_2 I_5 = -(a_1 \wedge f - a_2 \wedge f)I_3 n_\infty$$

which we could re-write as:

$$B = b n_\infty$$

where b is a 3D vector. As previously mentioned this is in the form of a CGA 'direction bivector'. These bivectors have the interesting property of being invariant to the action of translation rotors, effectively making them free vectors in the physics sense. Physically, two anti-parallel forces create a force couple, a pure moment, and so we will take objects of the form bn_∞ to be representations of moments in our scheme. This seems apt as, physically, a pure moment is often thought of as a free vector.

Let us now look at the products of GA. Consider two objects of the form:

$$S = aI_3 - bn_\infty, \;\; T = cI_3 - dn_\infty.$$

The geometric product between them gives:

$$ST = aI_3cI_3 - bn_\infty cI_3 - aI_3dn_\infty$$

$$= -ac - (bc + ad)I_3n_\infty.$$

In general this is a mixed grade object, if we expand out the geometric product between vectors it will give us some insight what these grades signify geometrically:

$$-ac - (bc + ad)I_3n_\infty = [-a \cdot c - a \wedge c] - [(b \cdot c + a \cdot d + b \wedge c + a \wedge d)I_3n_\infty].$$

Now collect the terms by grade:

$$ST = [-a \cdot c] + [-a \wedge c - (b \wedge c + a \wedge d)I_3n_\infty] + [-(b \cdot c + a \cdot d)I_3n_\infty],$$

$$ST = \langle ST \rangle_0 + \langle ST \rangle_2 + \langle ST \rangle_4$$

where

$$\langle ST \rangle_0 = -a \cdot c,$$

$$\langle ST \rangle_2 = -a \wedge c - (b \wedge c + a \wedge d)I_3n_\infty,$$

$$\langle ST \rangle_4 = -(b \cdot c + a \cdot d)I_3n_\infty.$$

From this grade-based breakdown it is quite easy to see how the different parts of the result relate to the various products of Screw Theory. First, the reciprocal product of screws, which in our 6D representation produced a scalar. In our CGA formulation this scalar maps to the coefficient of I_3n_∞ in the result of our geometric product, ie.

$$\langle ST \rangle_4 = -(\mathscr{S} \circ \mathscr{T})I_3n_\infty.$$

As S and T are in fact bivectors we can also write this in terms of the outer product:

$$S \wedge T = -(\mathscr{S} \circ \mathscr{T}) I_3 n_\infty.$$

Next we consider the bivector part of the geometric product result, alongside the cross product of screws:

$$\langle ST \rangle_2 = -a \wedge c - (b \wedge c + a \wedge d) I_3 n_\infty,$$

$$\mathscr{S} \times \mathscr{T} = \begin{bmatrix} A \times C \\ A \times D + B \times C \end{bmatrix}.$$

If we rewrite the GA formula here:

$$\langle ST \rangle_2 = -a \wedge c - (b \wedge c + a \wedge d) I_3 n_\infty = m I_3 - g n_\infty$$

then equating terms gives:

$$m = (a \wedge c) I_3$$

and

$$g = (b \wedge c) I_3 + (a \wedge d) I_3.$$

The form of the standard vector cross product in 3D GA is, for vectors a and b, given by:

$$a \times b = -(a \wedge b) I_3.$$

This means we can represent the cross product of screws via the negative of the bivector part of the geometric product result. In fact, we can write the bivector part of the result in terms of the 'commutator product' of geometric algebra which is also, unsurprisingly perhaps, written using the '$\times$' notation.

$$S \times T = \frac{1}{2}(ST - TS) = \langle ST \rangle_2.$$

For readers well versed in Screw Theory, Clifford/Geometric Algebra and Lie Theory this is a well known result. The commutator product equips the motor bivectors with a Lie bracket just as the cross product equips the screws with one. In fact we can go further into the idea of the motor bivector as an element of $se(3)$ and note that the grade 0 element of the geometric product, equal to the GA dot product between the bivectors, is proportional to the Killing form [79], K, of the Lie algebra ie. $\langle ST \rangle_0 = S \cdot T = -\frac{1}{4} K(\mathscr{S}, \mathscr{T})$.

6.2.4 Forces as lines in PGA

In PGA a line is computed as the intersection of two planes. As by default in PGA we are in a so called 'inner product null space' or IPNS we perform the intersection of these two planes by the outer product. We first break this down component-wise for the intersection of two planes:

$$P_1 = m_1 + d_1 e_0,$$

$$P_2 = m_2 + d_2 e_0,$$

$$L = P_1 \wedge P_2 = (m_1 + d_1 e_0) \wedge (m_2 + d_2 e_0)$$

$$= m_1 \wedge m_2 + (d_2 m_1 - d_1 m_2) \wedge e_0$$

which we can now re-write this in terms that look more familiar:

$$L = m_l I_3 - (a \wedge m_l) I_3 e_0.$$

In this form the line looks very similar to the CGA representation of the line. The line squares to a scalar, there is a section that is a Euclidean bivector and a section that is a null bivector. In fact the only thing that we have changed is the form of the null element. In CGA we typically use the null element n_∞ constructed from the sum of two orthogonal basis vectors, one squaring to $+1$ and one to -1. Here in PGA the null element e_0 is itself a basis vector. For our 6D force line representation we therefore have exactly the same mapping as we did in CGA:

$$F = f I_3 - (a \wedge f) I_3 e_0$$

and so moments appear as:

$$B = b e_0.$$

We would also expect the various products of Screw Theory to still appear as the various grades of the result of a geometric product between two elements of this algebra. Indeed they are the same as the CGA results, but of course replacing $I_3 n_\infty$ with $I_3 e_0$ which is now the pseudo-scalar of the algebra.

6.2.5 Force and moment representations in the GA literature

Different authors have taken different approaches when considering the form of forces in conformal and projective geometric algebra. In Lasenby, Lasenby and Doran's 'Rigid Body Dynamics and Conformal Geometric Algebra' [74] the conformal force representation is

taken to be of the form

$$F = f + \alpha n_\infty$$

where f is the normal Euclidean 3D force vector and α is a scalar multiplier. This force formulation is then wedged with n_∞ and combined with their equation (1.42) to define the equations of motion. When wedged with n_∞ this force takes the form:

$$F \wedge n_\infty = f \wedge n_\infty$$

this is in the form of a direction bivector.

This paper further goes on to specify in equation (1.50) that the form of the moment bivector that a force of this type generates about a point is:

$$M = X \wedge F.$$

We can break this up into its constituent parts as follows:

$$M = \left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \wedge (f + \alpha n_\infty)$$

$$= \frac{1}{2}x^2(n_\infty \wedge f) + (x \wedge f) + (n_0 \wedge f) + \alpha(x \wedge n_\infty) + \alpha(n_0 \wedge n_\infty).$$

This moment formulation is then wedged with n_∞ and combined with their equation (1.42) to define the equations of motion. We now consider, as they do in their equation (1.55), the form of the moment wedged with n_∞:

$$M \wedge n_\infty = \frac{1}{2}x^2(n_\infty \wedge f) \wedge n_\infty + (x \wedge f) \wedge n_\infty + (n_0 \wedge f) \wedge n_\infty + \alpha(x \wedge n_\infty) \wedge n_\infty + \alpha(n_0 \wedge n_\infty) \wedge n_\infty$$

$$= x \wedge f \wedge n_\infty + n_0 \wedge f \wedge n_\infty.$$

Now take the dual of this quantity with respect to 5D pseudoscalar:

$$(M \wedge n_\infty)I_5 = fI_3 - (x \wedge f)I_3 n_\infty.$$

This is the bivector form of a CGA line in the direction of the 3D vector f and passing through the point x.

'Rigid Body Dynamics and Conformal Geometric Algebra' [74] therefore uses a mixture of 1-vectors and bivectors to represent forces but in the end their static equilibrium conditions are in the form of bivectors and trivectors, indeed they are left with something very similar to our formulation in section 6.2.2. The main difference in fact is related to which of the two

orthogonal elements of the wrench formulation we take to be a moment and which to be a line. Lasenby et al. effectively choose the $f \wedge n_\infty$ part to be a force and here we choose the dual line section to be the force.

In fact conceptually our approach here of having lines as forces is the same as Charles Gunn's approach in his paper 'On the Homogeneous Model of Euclidean Geometry' [42] and his PhD Thesis [44]. To make this connection more explicit consider the form of a PGA 'ideal line', used to represent moments in Gunn's formulation:

$$L = e_0(\alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3) = e_0 a$$

where $a = \alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3$. The properties of this object are that it squares to 0, ie. it is null but that the magnitude and direction of the 3D vector a are the magnitude and axis of the moment. This is identical to the $a \wedge n_\infty$ formulation for bivector moments that we described in section 6.2.2.

As discussed in Anthony Lasenby's 'Rigid Body Dynamics in a Constant Curvature Space and the '1D-up' Approach to Conformal Geometric Algebra' [73], both 'On the Homogeneous Model of Euclidean Geometry' [42] and 'Rigid Body Dynamics and Conformal Geometric Algebra' [74] use the same form for rotors and generalised instantaneous velocities, known in the Screw Theory literature as twists.

6.3 Screw transformations, instantaneous twists, and the motor manifold

6.3.1 Time derivatives of frame transformations

Before looking at dynamics in detail, we will define the notation used and state various definitions.

We will consider a world that contains a single rigid body. A frame is rigidly attached to the rigid body and the body moves through space such that a time varying rotor R will transform an arbitrary fixed point X' in the body frame into the corresponding point X in the world frame:

$$X = RX'\tilde{R}.$$

We will take time derivatives. As X' is fixed:

$$\dot{X} = \dot{R}X'\tilde{R} + RX'\dot{\tilde{R}}.$$

Now substitute in $X'\tilde{R} = \tilde{R}X$ and $RX' = XR$:

$$\dot{X} = \dot{R}\tilde{R}X + XR\dot{\tilde{R}}.$$

A rotor by definition has the property that $R\tilde{R} = 1$. If we differentiate this constraint with respect to time

$$\dot{R}\tilde{R} + R\dot{\tilde{R}} = 0$$

which means we can write:

$$\dot{X} = (\dot{R}\tilde{R})X - X(\dot{R}\tilde{R})$$

which is twice the anti-commutator of X and $\dot{R}\tilde{R}$. If X is a 1-vector and as $\dot{R}\tilde{R}$ is a bivector we can write:

$$\dot{X} = -2X \cdot (\dot{R}\tilde{R}).$$

If we now label our bivector quantity:

$$\Omega_w = -2\dot{R}\tilde{R} \tag{6.3}$$

it allows us to write:

$$\dot{X} = -\frac{1}{2}(\Omega_w X - X\Omega_w) = X \times \Omega_w \tag{6.4}$$

where $\times$ represents the commutator product. **Note that in this form with the commutator product no assumptions are made of the grade or other properties of X.** If we do restrict X to 1-vectors however, this allows us to write:

$$\dot{X} = X \cdot \Omega_w.$$

We can further re-arrange equation (6.3) to get an equation for the relationship between the rotor R and its time derivative in terms of this quantity Ω_w:

$$\dot{R} = -\frac{1}{2}\Omega_w R. \tag{6.5}$$

This quantity Ω_w is actually our generalised instantaneous screw velocity, expressed in the world frame. Geometrically it is a screw and we can transform it just like any other screw between frames. We can therefore write $\Omega_w = R\Omega\tilde{R}$ and change equation (6.5) to:

$$\dot{R} = -\frac{1}{2}R\Omega\tilde{R}R = -\frac{1}{2}R\Omega \tag{6.6}$$

where Ω is the velocity bivector expressed in the body frame. For reference the reverse of this quantity is:

$$\dot{\tilde{R}} = -\frac{1}{2}\tilde{\Omega}\tilde{R}.$$

In the screw theory literature Ω is known as a 'twist' or 'velocity screw'. To take further time derivatives we can just use the chain rule:

$$\ddot{R} = -\frac{1}{2}R\dot{\Omega} - \frac{1}{2}\dot{R}\Omega \tag{6.7}$$

and we will also calculate the reverse for reference:

$$\ddot{\tilde{R}} = -\frac{1}{2}\dot{\tilde{\Omega}}\tilde{R} - \frac{1}{2}\tilde{\Omega}\dot{\tilde{R}}.$$

6.4 Momentum and inertia

6.4.1 Screw momentum

In traditional 3D dynamics formulations we specify that the resultant force is the rate of change of linear momentum and the resultant moment is the rate of change of angular momentum. In a screw formulation we can specify that, for a body under the influence of multiple external forces W_i, the resultant wrench W_r is the rate of change of screw momentum Ψ with time:

$$W_r = \sum W_i = \frac{\partial \Psi}{\partial t}.$$

We can, of course, write this whether we are working in the 6D vector space, CGA or PGA.

6.4.2 Mapping from screw velocity to screw momentum

In 3D dynamics we are used to the idea of converting between linear velocity and linear momentum via multiplication or division by the mass of the rigid body. For a body of mass m and linear velocity v_l the linear momentum is simply ρ_l:

$$\rho_l = m v_l.$$

When it comes to angular velocity, v_a, and angular momentum, ρ_a, however we have a more complicated relationship. In fact, for a body centred and axis aligned reference frame, the

two are related by a diagonal matrix known as the inertia tensor that we label here as $\mathscr{M}_a$:

$$\rho_a = \mathscr{M}_a v_a$$

where $\mathscr{M}_a$ can be expressed in the form of γ_i, the second moments of volume:

$$\mathscr{M}_a = m \begin{bmatrix} \gamma_1 & 0 & 0 \\ 0 & \gamma_2 & 0 \\ 0 & 0 & \gamma_3 \end{bmatrix}.$$

While it is clear that our screw equivalent of the inertia tensor should also be a linear function that somehow combines the linear and rotational aspects of the above and maps between screw velocity and screw momentum it is not immediately obvious how we should go about constructing such a function. Let us write the matrix version of this linear function for now as $\mathscr{Q}$ and the GA version as Q:

$$\Psi = Q[\Omega_w].$$

To solve the problem of determining this function we will construct a little thought experiment. Imagine a rigid body that is initially at rest at the origin but which is then acted on by a wrench W. The velocity of objects attached to the frame of the body are given by:

$$\dot{X} = X \times \Omega_w = X \times Q^{-1}[\Psi].$$

By differentiating this equation we can calculate the acceleration of objects in the frame:

$$\ddot{X} = \dot{X} \times Q^{-1}[\Psi] + X \times Q^{-1}[\dot{\Psi}]. \tag{6.8}$$

In this case the body is initially at rest implying $\Omega_w = 0$ allowing us to eliminate the first term and leaving us with:

$$\ddot{X} = X \times Q^{-1}[\dot{\Psi}] = X \times Q^{-1}[W]. \tag{6.9}$$

We therefore have a direct mapping from the wrench acting on the body to the initial acceleration of objects in the frame and we can now use our physical intuition about the world to guide us to a compatible form of Q^{-1} and hence Q.

To calculate the form of this linear function Q in GA we will first have to define what is known in the GA literature as a **reciprocal frame**. An important thing to note here is that this is different to the concept in screw theory of a screw and a twist being reciprocal [40]

which we will come to later when considering virtual work and power. What we mean by a reciprocal frame here is a set of reciprocal bases S^i that are matched to the motor bivectors S_i such that when the GA inner product is taken between the two matched elements the result is 1 but otherwise 0, ie.:

$$S^i \cdot S_j = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

These reciprocal frame constructs are fairly common in the GA literature and are especially useful for this type of problem.

As Q^{-1} is a linear function operating on a motor bivector W in CGA it can be written as:

$$Q^{-1}[W] = \sum_i \left[(S^i \cdot W) \sum_k c_{ik} S_k \right] \tag{6.10}$$

where c_{ki} are scalar coefficients, S^i is the reciprocal frame for the motor bivectors and S_k are the motor bivectors themselves. We will take S_k and S^i as follows:

$$S_k = [e_1 I_3,\ e_2 I_3,\ e_3 I_3,\ e_1 \wedge n_\infty,\ e_2 \wedge n_\infty,\ e_3 \wedge n_\infty],$$

$$S^i = [-e_1 I_3,\ -e_2 I_3,\ -e_3 I_3,\ e_1 \wedge n_0,\ e_2 \wedge n_0,\ e_3 \wedge n_0]$$

and will seek to determine c_{ik}. This formulation is equivalent to a square matrix formulation of Q^{-1} where the coefficients c_{ik} are the elements in the matrix. What we need to do here is to find out exactly what these c_{ik} coefficients have to be.

Firstly we will probe the response of the function to the action of a force passing through the centre of mass of the object, which lies at the origin. We can write such a force as a dual line in CGA as:

$$W = FI_5 = fI_3 - (a \wedge f) I_3 n_\infty$$

where in this case a is 0 as we are at the origin, giving:

$$W = FI_5 = fI_3.$$

The mapping that embeds a 3D point x in CGA is given by:

$$X = \frac{1}{2} x^2 n_\infty + x + n_0$$

which, after differentiation twice leads to:

$$\ddot{X} = \frac{1}{2}(\ddot{x}x + 2\dot{x}\dot{x} + x\ddot{x})n_\infty + \ddot{x}.$$

As the body is initially stationary, $\dot{x}$ is zero and so we have for time $t = 0$:

$$\ddot{X} = \frac{1}{2}(\ddot{x}x + x\ddot{x})n_\infty + \ddot{x} = (\ddot{x} \cdot x)n_\infty + \ddot{x}.$$

For the case of a force of magnitude $|f|$ applied to the centre of mass of a rigid body we would expect all points on the body to accelerate linearly at $\frac{|f|}{m}$ in the direction of f, or more formally $\ddot{x} = \frac{f}{m}$ for all x. We can encode this expectation by substituting $\ddot{X}$ and X into equation (6.9):

$$(\ddot{x} \cdot x)n_\infty + \ddot{x} = \left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \times Q^{-1}[W] \tag{6.11}$$

and we can then apply our specific case:

$$\left(\frac{f}{m} \cdot x\right) n_\infty + \frac{f}{m} = \left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \times \sum_i \left[(S^i \cdot W) \sum_k c_{ik} S_k\right].$$

Looking closely at

$$\sum_i \left[(S^i \cdot W) \sum_k c_{ik} S_k\right]$$

we can see that $W = fI_3$ is a Euclidean bivector and, when dotted with each of the reciprocal frame elements in turn it is non zero only for $[-e_1 I_3, -e_2 I_3, -e_3 I_3]$ in which cases the dot product gives only f_i, the component of the force in the e_i direction. In other words we could write:

$$\sum_i \left[(S^i \cdot W) \sum_k c_{ik} S_k\right] = \sum_j f_j \left[\sum_k c_{jk} S_k\right] = \sum_j \sum_k f_j c_{jk} S_k$$

where $j \in (1,2,3)$ and $f_j = e^j \cdot f$. We can combine this with $\ddot{X} = X \cdot \dot{\Omega}_w$ to give:

$$\left(\frac{f}{m} \cdot x\right) n_\infty + \frac{f}{m} = \left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \cdot \sum_j \sum_k f_j c_{jk} S_k$$

$$\left(\frac{f}{m} \cdot x\right) n_\infty + \frac{f}{m} = \sum_j \sum_k f_j c_{jk} \left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \cdot S_k$$

Let us now examine the terms on the right hand side of the above equation:

$$\left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \cdot S_k = \frac{1}{2}x^2 n_\infty \cdot S_k + x \cdot S_k + n_0 \cdot S_k$$

For $S_k \in e_n \wedge n_\infty, n \in [1,2,3]$ we have the results that $n_0 \cdot S_k = e_n$, $n_\infty \cdot S_k = 0$, $x \cdot (e_n \wedge n_\infty) = x_n n_\infty$. For $S_k \in [e_1 I_3,\ e_2 I_3,\ e_3 I_3]$ we have the results that $n_0 \cdot S_k = 0$, $n_\infty \cdot S_k = 0$.

$$\begin{aligned}\left(\frac{f}{m} \cdot x\right) n_\infty + \frac{f}{m} &= \sum_j \sum_k f_j c_{jk} \left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \cdot S_k \\ &= \sum_j f_j (c_{j1} x \cdot (e_1 I_3) + c_{j2} x \cdot (e_2 I_3) + c_{j3} x \cdot (e_3 I_3) \\ &\quad + c_{j4} e_1 + c_{j5} e_2 + c_{j6} e_3 + c_{j4} x_1 n_\infty + c_{j5} x_2 n_\infty + c_{j6} x_3 n_\infty) \quad (6.12)\end{aligned}$$

Now we can consider the above equation component-wise and extract the required transformation coefficients. Firstly, consider the coefficients of n_∞:

$$\left(\frac{f}{m} \cdot x\right) n_\infty = \sum_j f_j (c_{j4} x_1 n_\infty + c_{j5} x_2 n_\infty + c_{j6} x_3 n_\infty)$$

Expand the left side into a sum of elements of the force vector:

$$\left(\frac{f}{m} \cdot x\right) n_\infty = \sum_j \frac{f_j x_j}{m} n_\infty.$$

Comparing terms we see we have arrived at the following solution:

$$c_{14}, c_{25}, c_{36} = \frac{1}{m}$$

$$c_{24}, c_{34}, c_{15}, c_{35}, c_{16}, c_{26} = 0$$

Now that we have dealt with the n_∞ terms we can return to the Euclidean bivectors:

$$\frac{f}{m} = \sum_j f_j (c_{j1} x \cdot (e_1 I_3) + c_{j2} x \cdot (e_2 I_3) + c_{j3} x \cdot (e_3 I_3) + c_{j4} e_1 + c_{j5} e_2 + c_{j6} e_3).$$

We have just calculated the c_{j4}, c_{j5}, c_{j6} terms in the right hand side of this equation and so by substituting these in we get something of the form:

$$\frac{f}{m} = \frac{f}{m} + \sum_j f_j (c_{j1} x \cdot (e_1 I_3) + c_{j2} x \cdot (e_2 I_3) + c_{j3} x \cdot (e_3 I_3))$$

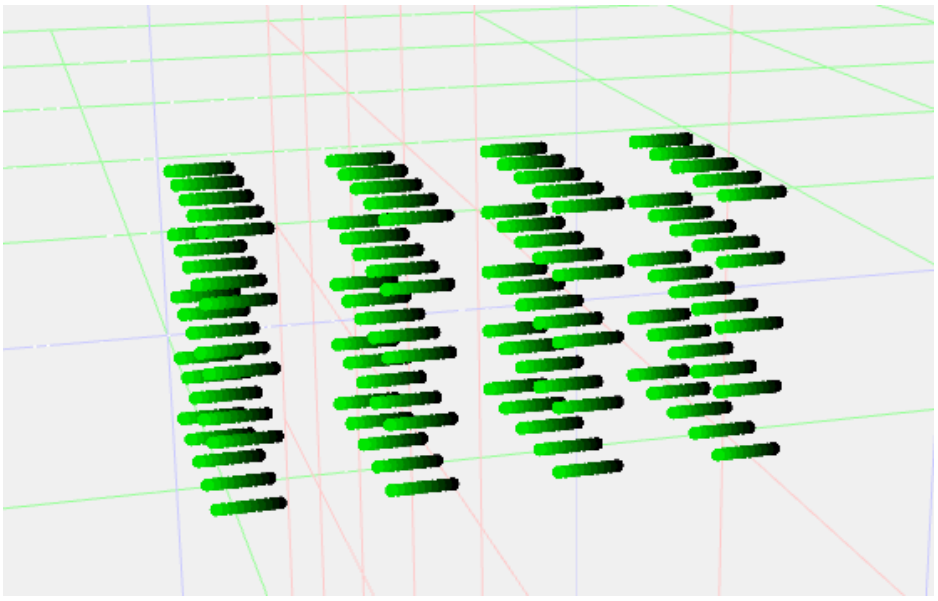


Fig. 6.1 The multivector field generated by the commutator product of a twist and a field of multivector points can be visualised and provides a visual verification of the action of specific types of twist. Here we are visualising a twist in the form of a 'direction bivector' which produces a uniform linear velocity field. This implies that our inertia tensor, or indeed some preprocessing step to the inertia tensor, must map from lines through the origin to direction bivectors, a fact proved in the mathematical content of this section.

which implies that $c_{11}, c_{12}, c_{13}, c_{21}, c_{22}, c_{23}, c_{31}, c_{32}, c_{33} = 0$. A visual representation of the argument made here can be seen in Figure 6.1.

So far we have dealt with all the translational elements of the transformation, however we have only prescribed 18 of the total 36 (6×6) degrees of freedom of the problem. To continue calculating the required form of the transformation we will now analyse the effect of a torque applied to the rigid body. Firstly we will again need the general form of the acceleration of a point due to a wrench:

$$(\ddot{x}\cdot x)n_\infty + \ddot{x} = \left(\frac{1}{2}x^2 n_\infty + x + n_0\right)\cdot \sum_i \sum_k (S^i \cdot W)c_{ik}S_k. \tag{6.13}$$

We will now apply a moment to the rigid body, again at rest at the origin. From standard 3D kinematics we know that if we have a body rotating about its centre of mass and about an axis $\hat{v}_a$ with angular speed $|v_a|$, ie. $v_a = |v_a|\hat{v}_a$ then, provided that the centre of mass has no linear velocity we can calculate the linear velocity of a point on the body as:

$$\dot{x} = v_a \times x = (\mathscr{M}_a^{-1}\rho_a) \times x$$

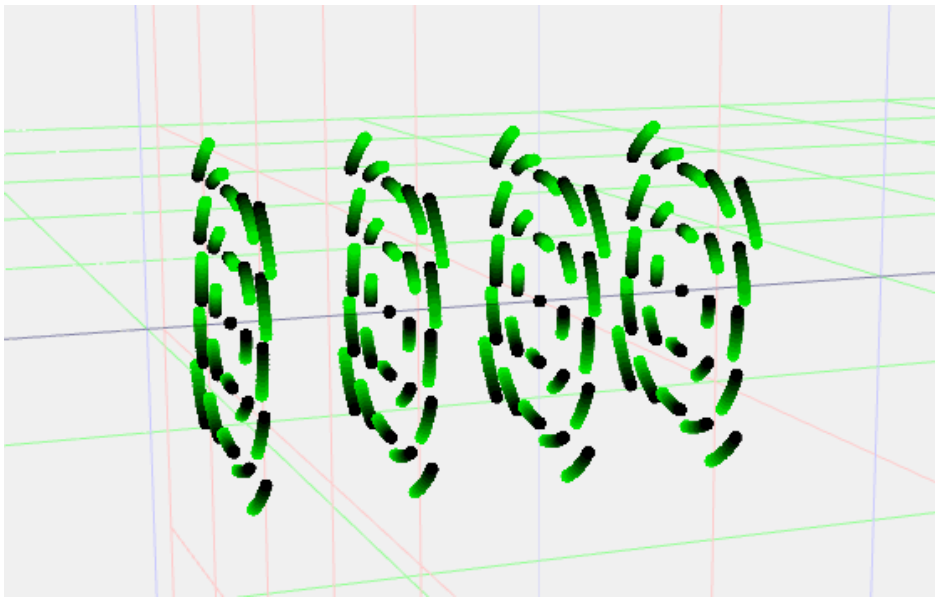


Fig. 6.2 As in Figure 6.1 here we are visualising the multivector field generated by the commutator product of a twist and a field of points. Here the twist that is visualised is a line through the origin. The field that is generated is rotational about the line. Again as with the translational field of Figure 6.1 this implies specific constraints on our inertia tensor and shows the inherent reciprocal nature of momentum screws/wrenches and velocity twists in the Screw Theory formulation of dynamics.

and the acceleration of a point as:

$$\ddot{x} = \dot{v}_a \times x + v_a \times \dot{x}$$

where $\times$ is the traditional cross product operation. In this thought experiment the body is initially at rest allowing us to remove the $\dot{x}$ term and leaving us with:

$$\ddot{x} = \dot{v}_a \times x = (\mathscr{M}_a^{-1}\dot{\rho}_a) \times x = (\mathscr{M}_a^{-1}t) \times x$$

where t is the 3d torque vector. The GA equivalent of the traditional 3d cross product for vectors a and b is $-I_3(a \wedge b)$. Which means we can write:

$$(\mathscr{M}_a^{-1}\rho_a) \times x = -I_3((\mathscr{M}_a^{-1}t) \wedge x).$$

We can represent this same moment in our CGA formulation as the bivector wrench $W = tn_\infty$. Again we look at:

$$\sum_i \sum_k (S^i \cdot W)c_{ik} = \sum_i \sum_k (S^i \cdot (tn_\infty))c_{ik}$$

and from this we note that $(S^i \cdot (tn_\infty)) = 0$ for $S^i \in [-e_n I_3]$ and that $(S^i \cdot (tn_\infty)) = t_{i-3}$ for $i \in [4,5,6]$. For this case we can therefore write equation (6.13) as:

$$\left(\left(-I_3((\mathscr{M}_a^{-1}t) \wedge x)\right) \cdot x\right)n_\infty - I_3((\mathscr{M}_a^{-1}t) \wedge x) = \left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \cdot \sum_{i\in[4,5,6]} \sum_k t_{i-3}c_{ik}S_k$$

The left hand side of this equation collapses simply to $\ddot{x}$ and we bring X inside the summation again on the right:

$$-I_3((\mathscr{M}_a^{-1}t) \wedge x) = \sum_{i\in[4,5,6]} \sum_k t_{i-3}c_{ik}\left(\frac{1}{2}x^2 n_\infty + x + n_0\right) \cdot S_k.$$

We can now use the same results as noted previously to simplify the right hand side of this equation, leading to:

$$\begin{aligned} -I_3((\mathscr{M}_a^{-1}t) \wedge x) = \sum_{i\in[4,5,6]} t_{i-3}(&c_{i1}x \cdot (e_1 I_3) + c_{i2}x \cdot (e_2 I_3) + c_{i3}x \cdot (e_3 I_3) \\ &+ c_{i4}e_1 + c_{i5}e_2 + c_{i6}e_3 + c_{i4}x_1 n_\infty + c_{i5}x_2 n_\infty + c_{i6}x_3 n_\infty) \end{aligned}$$

Equating like terms eliminates all the n_∞ terms, ie. $c_{i4}, c_{i5}, c_{i6} = 0$ for $i \in [4,5,6]$. This leaves us with:

$$-I_3((\mathscr{M}_a^{-1}t) \wedge x) = \sum_{i\in[4,5,6]} t_{i-3}(c_{i1}x \cdot (e_1 I_3) + c_{i2}x \cdot (e_2 I_3) + c_{i3}x \cdot (e_3 I_3)).$$

We can break up the left side of this component-wise:

$$\sum_{i\in[4,5,6]} -I_3((\mathscr{M}_a^{-1}t_{i-3}e_{i-3}) \wedge x) = \sum_{i\in[4,5,6]} t_{i-3}(c_{i1}x \cdot (e_1 I_3) + c_{i2}x \cdot (e_2 I_3) + c_{i3}x \cdot (e_3 I_3))$$

Now assume that our torque vector is aligned with one of the principal axes of the rigid body, say e_1, this implies t_{i-3} is only non-zero for $i = 4$:

$$-I_3((t_1\mathscr{M}_a^{-1}e_1) \wedge x) = t_1(c_{41}x \cdot (e_1 I_3) + c_{42}x \cdot (e_2 I_3) + c_{43}x \cdot (e_3 I_3))$$

$$\frac{t_1}{m\gamma_1}(x \wedge e_1) I_3 = t_1(c_{41}x \cdot (e_1 I_3) + c_{42}x \cdot (e_2 I_3) + c_{43}x \cdot (e_3 I_3))$$

Which is true because $M_a^{-1}e_1 = \frac{1}{m\gamma_1}e_1$. Noting that $(x \wedge e_1) I_3 = x \cdot (e_1 I_3)$ allows us to identify the c parameters: $c_{41} = \frac{1}{m\gamma_1}$, $c_{42} = 0$, $c_{43} = 0$.

Of course our choice of e_1 as the principal axis to which our torque vector aligns was arbitrary, we could equally have chosen one of the other two principal axes, e_2 and e_3 with their respective γ_2 and γ_3. As a result of this symmetry we can directly identify our final c parameters: $c_{51} = 0$, $c_{52} = \frac{1}{m\gamma_2}$, $c_{53} = 0$, $c_{61} = 0$, $c_{62} = 0$, $c_{63} = \frac{1}{m\gamma_3}$.

Figure 6.2 gives a graphical illustration of why this form of mapping is required for rotations and torques.

6.4.3 The Screw Inertia Tensor

For our 6D vector representation had we not just done the maths of the previous section we might have been tempted to stack the v_l and v_a on top of each other to produce a combined inertia tensor like:

$$\Psi = \begin{bmatrix} \rho_l \\ \rho_a \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0\ldots \\ 0 & m & 0 & 0\ldots \\ 0 & 0 & m & 0\ldots \\ 0\ldots & 0\ldots & 0\ldots & \mathscr{M}_a \end{bmatrix} \begin{bmatrix} v_l \\ v_a \end{bmatrix}.$$

This however would ignore the fundamental conceptual differences in the action of the top 3 and bottom 3 elements of the 6D representation as a wrench vs as a twisting motion generator. Instead, we need to use the coefficients c_{ik} of the last section to produce a matrix

that performs a 'flip' in the relative positions of the parts of the vectors while also applying the required scaling along each principal axis. Putting the calculated coefficients in place we can see that the matrix that comes out for Q looks like this:

$$\Psi = \begin{bmatrix} \rho_l \\ \rho_a \end{bmatrix} = \begin{bmatrix} 0... & m & 0 & 0 \\ 0... & 0 & m & 0 \\ 0... & 0 & 0 & m \\ \mathscr{M}_a & 0.. & 0.. & 0.. \end{bmatrix} \begin{bmatrix} v_a \\ v_l \end{bmatrix}.$$

We could also achieve this 'flip' effect via a diagonal matrix and a permutation matrix:

$$\Psi = \begin{bmatrix} \rho_l \\ \rho_a \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0... \\ 0 & m & 0 & 0... \\ 0 & 0 & m & 0... \\ 0... & 0... & 0... & \mathscr{M}_a \end{bmatrix} \begin{bmatrix} 0... & I(3\times 3) \\ I(3\times 3) & 0... \end{bmatrix} \begin{bmatrix} v_a \\ v_l \end{bmatrix} \tag{6.14}$$

where $I(3\times 3)$ is the 3×3 identity matrix.

When writing the inertia tensor in CGA it is convenient to do a little relabelling for ease of reading. The reciprocal frame of the motor bivectors in CGA is as follows:

$$-e_1 I_3,\ -e_2 I_3,\ -e_3 I_3,\ e_1 \wedge n_0,\ e_2 \wedge n_0,\ e_3 \wedge n_0.$$

Which we can break into the two groups:

$$l_i = e_i I_3,\ \ l^i = -e_i I_3$$

$$t_i = e_i \wedge n_\infty,\ \ t^i = e_i \wedge n_0$$

and so with these we can write the inertia tensor:

$$\Psi = Q(\Omega) = m \sum_{i=1}^{i=3} \left[(\Omega \cdot t^i) l_i + \gamma_i (\Omega \cdot l^i) t_i \right]$$

and the inverse inertia tensor:

$$Q^{-1}(\Psi) = \Omega = \frac{1}{m} \sum_{i=1}^{i=3} \left[\frac{1}{\gamma_i} (\Psi \cdot t^i) l_i + (\Psi \cdot l^i) t_i \right].$$

This inertia tensor performs the same kind of 'flip' that we saw for the 6D screw representation. Effectively our inertia tensor does the following mapping:

$$\begin{aligned} e_1 I_3 &\rightarrow e_1 \wedge n_\infty, & e_1 \wedge n_\infty &\rightarrow e_1 I_3 \\ e_2 I_3 &\rightarrow e_2 \wedge n_\infty, & e_2 \wedge n_\infty &\rightarrow e_2 I_3 \\ e_3 I_3 &\rightarrow e_3 \wedge n_\infty, & e_3 \wedge n_\infty &\rightarrow e_3 I_3 \end{aligned}$$

The reciprocal frame construction of the inertia tensor that we have discussed so far works for many algebras but for degenerate metric algebras such as PGA the fact that we have an element squaring to zero means this setup does not work. Instead we need to do something a little different. The degenerate metric approach to reciprocal frames [43] is to consider some blade which we will label x^i that wedges with a given blade of magnitude b_i ie $b_i x_i$ to produce the pseudoscalar with magnitude b_i ie. $b_i x_i \wedge x^i = b_i I$. Clearly, despite us labelling it x^i , this object is not quite the same as the reciprocal frame, although it allows us to perform the same function of coordinate free coefficient selection producing the magnitude b_i as the scalar coefficient of the pseudoscalar. Let us now identify this pseudo-reciprocal frame for the PGA bivectors:

$$\begin{aligned} e_1 \wedge e_2 &\rightarrow -e_3 \wedge e_0, & e_1 \wedge e_0 &\rightarrow -e_2 \wedge e_3 \\ e_1 \wedge e_3 &\rightarrow e_2 \wedge e_0, & e_2 \wedge e_0 &\rightarrow e_1 \wedge e_3 \\ e_2 \wedge e_3 &\rightarrow -e_1 \wedge e_0, & e_3 \wedge e_0 &\rightarrow -e_1 \wedge e_2 \end{aligned}$$

Comparing the PGA pseudo-reciprocal frame mapping with that of our CGA-inertia tensor mapping it is immediately clear that they are equivalent up to a minus sign. We will now define a function to perform this PGA mapping and will call it J. J has the following action:

$$J(b_i x_i) = \langle b_i x_i \wedge x^i \rangle_{e_{1230}} x^i = b_i x^i$$

where the syntax $\langle A \rangle_{e_{1230}}$ returns the scalar coefficient of $e_1 \wedge e_2 \wedge e_3 \wedge e_0$ in A. We can extend this operation to combinations of basis elements by linearity so that for $X = \sum_i b_i x_i$:

$$X^J = J(X) = J\left(\sum_i b_i x_i\right) = \sum_i J(b_i x_i).$$

As with our CGA reciprocal frame let's now write our PGA pseudo-reciprocal frame in two groups:

$$l_i = e_i I_3, \;\; l^i = e_i \wedge e_0$$

$$t_i = e_i \wedge e_0, \;\; t^i = e_i I_3$$

This means we can write our PGA inertia tensor as:

$$\Psi = Q(\Omega) = -m \sum_{i=1}^{i=3} \left[\langle \Omega \wedge l^i \rangle_{e1230} l^i + \gamma_i \langle \Omega \wedge t^i \rangle_{e1230} t^i \right]$$

and its inverse inertia tensor:

$$Q^{-1}(\Psi) = \Omega = -\frac{1}{m} \sum_{i=1}^{i=3} \left[\frac{1}{\gamma_i} \langle \Psi \wedge l^i \rangle_{e1230} l^i + \langle \Psi \wedge t^i \rangle_{e1230} t^i \right].$$

We could also apply the J map first to first 'flip' the input and apply a component-wise scaling A to the result:

$$\Psi = Q(\Omega) = A[J(\Omega)].$$

This J map first formulation is conceptually the same as the screw formulation with a flip permutation matrix as in equation (6.14).

6.4.4 Motor Bivectors as the Principal Screws of Inertia

We can visualise the motor bivectors as a frame of screws attached to the origin. These motor bivectors are a version of the Principal Screws of Inertia, specifically they are the principal screws in a Plücker and Hunt sense as opposed to Ball's original formulation of the principal screws. Effectively what we are doing in the inertia tensor is considering these principal screws as wrenches and analysing the impact of them on the motion of the body.

By considering the bivectors as localised screws we can begin to build intuition about them and their properties. First of all, we will consider how they, and their reciprocal frame, transform under the action of rigid body rotors. The Euclidean bivectors are in the form of a dual line through the origin and are affected by rotors exactly as lines are. The direction type motor bivectors ($e_i \wedge n_\infty$) are, as we mentioned previously, invariant to translation rotors but are affected by rotation rotors. For the general case of the rigid body rotor the direction bivectors therefore are affected only by the rotational aspect of the rotor. This is in keeping with the view of these bivectors as dual lines at infinity, known in the projective geometry world as 'ideal' lines.

The transformation properties of the motor bivectors suggests an inroad on the problem of defining non-axis aligned inertia tensors which often comes up in analysis problems when we wish to transform the frame of a body to be about a known axis of rotation. We can phrase this specific problem as follows. Consider a body with a known, axis aligned, inertia tensor

Q_B and a world frame momentum Ψ. We can get the velocity of the body by transforming the world frame momentum back to the body frame, applying the inverse inertia tensor, and transforming back:

$$\Omega = RQ_B^{-1}(\tilde{R}\Psi R)\tilde{R}.$$

We will wrap this $RQ_B^{-1}(\tilde{R}\Psi R)\tilde{R}$ operation up as an inertia tensor in its own right and label it Q_W^{-1}, we can therefore write:

$$\Omega = Q_W^{-1}(\Psi) = RQ_B^{-1}(\tilde{R}\Psi R)\tilde{R}. \tag{6.15}$$

The problem of defining non-axis aligned inertia tensors is, given we know Q_B^{-1} and R, exactly what form does Q_W^{-1} take?

The form of Q_B^{-1} is:

$$Q_B^{-1}(\Psi_B) = \Omega_B = \frac{1}{m}\sum_{i=1}^{i=3}\left[\frac{1}{\gamma_i}(\Psi_B \cdot t^i)l_i + (\Psi_B \cdot l^i)t_i\right].$$

Based on the fact we can transform screws and their reciprocal frames with rotors as usual, we will guess the form of Q_W^{-1} to be as follows:

$$Q_W^{-1}(\Psi) = \Omega = \frac{1}{m}\sum_{i=1}^{i=3}\left[\frac{1}{\gamma_i}(\Psi \cdot (Rt^i\tilde{R}))(Rl_i\tilde{R}) + (\Psi \cdot (Rl^i\tilde{R}))(Rt_i\tilde{R})\right]. \tag{6.16}$$

Note this is just the same as M_B^{-1} except that we have transformed all of the motor bivectors and reciprocals by R, effectively transforming the frame of principal screws to be centred and aligned with the rigid body but expressed in the world frame. We can check whether our guess is correct as follows. Substitute $\Psi = (R\Psi_B\tilde{R})$:

$$Q_W^{-1}(\Psi) = \frac{1}{m}\sum_{i=1}^{i=3}\left[\frac{1}{\gamma_i}((R\Psi_B\tilde{R}) \cdot (Rt^i\tilde{R}))(Rl_i\tilde{R}) + ((R\Psi_B\tilde{R}) \cdot (Rl^i\tilde{R}))(Rt_i\tilde{R})\right].$$

Noting that $(R\Psi_B\tilde{R}) \cdot (Rt^i\tilde{R}) \equiv \Psi_B \cdot t^i$ and $(R\Psi_B\tilde{R}) \cdot (Rl^i\tilde{R}) \equiv \Psi_B \cdot l^i$ we can write:

$$Q_W^{-1}(\Psi) = \frac{1}{m}\sum_{i=1}^{i=3}\left[\frac{1}{\gamma_i}(\Psi_B \cdot t^i)(Rl_i\tilde{R}) + (\Psi_B \cdot l^i)(Rt_i\tilde{R})\right].$$

We can then take the rotor application outside of the summation by factorisation and we have arrived at the required relation of equation (6.15):

$$\begin{aligned} Q_W^{-1}(\Psi) &= R\left(\frac{1}{m}\sum_{i=1}^{i=3}\left[\frac{1}{\gamma_i}(\Psi_B\cdot t^i)l_i+(\Psi_B\cdot l^i)t_i)\right]\right)\tilde{R} \\ &= RQ_B^{-1}(\Psi_B)\tilde{R} = RQ_B^{-1}(\tilde{R}\Psi R)\tilde{R}. \end{aligned}$$

Of course by allowing the movement of the frame of the inertia tensor with the body it will no longer be constant and we therefore might also want the form of the time derivative for potential applications:

$$\dot{Q_W^{-1}}(\Psi)+Q_W^{-1}(\dot{\Psi}) = \dot{R}Q_B^{-1}(\tilde{R}\Psi R)\tilde{R}+RQ_B^{-1}(\tilde{R}\Psi R)\dot{\tilde{R}}+RQ_B^{-1}(\dot{\tilde{R}}\Psi R+\tilde{R}\Psi\dot{R})\tilde{R}. \quad (6.17)$$

In this subsection we have analysed the principal screws as CGA objects and our analysis of the translation of the inertia tensor was phrased using the transformation of the reciprocal frame. In reality we could equally have done our analysis from the PGA perspective as well, with our pseudo-reciprocal frame taking the place of the reciprocal frame and so our equivalent transformed inverse inertia tensor would appear as:

$$Q_W^{-1}(\Psi) = -\frac{1}{m}\sum_{i=1}^{i=3}\left[\frac{1}{\gamma_i}\langle\Psi\wedge(Rl^i\tilde{R})\rangle_{e1230}(Rl^i\tilde{R})+\langle\Psi\wedge(Rt^i\tilde{R})\rangle_{e1230}(Rt^i\tilde{R})\right].$$

6.5 Unconstrained rigid body dynamics

Equipped with forces, moments, momentum, velocities and inertia tensors we are now at a position where we can formulate the equations of motion and simulate them. We will start by considering the dynamics of an unconstrained rigid body moving under the influence of external forces and moments. We can write the state of our rigid body at a time t as:

$$Y_t = \begin{bmatrix} R_t \\ \Psi_t \end{bmatrix}$$

and its first time derivative is:

$$\dot{Y}_t = \begin{bmatrix} \dot{R}_t \\ \dot{\Psi}_t \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}R_t\Omega_t \\ R_tW_{bt}\tilde{R}_t \end{bmatrix}$$

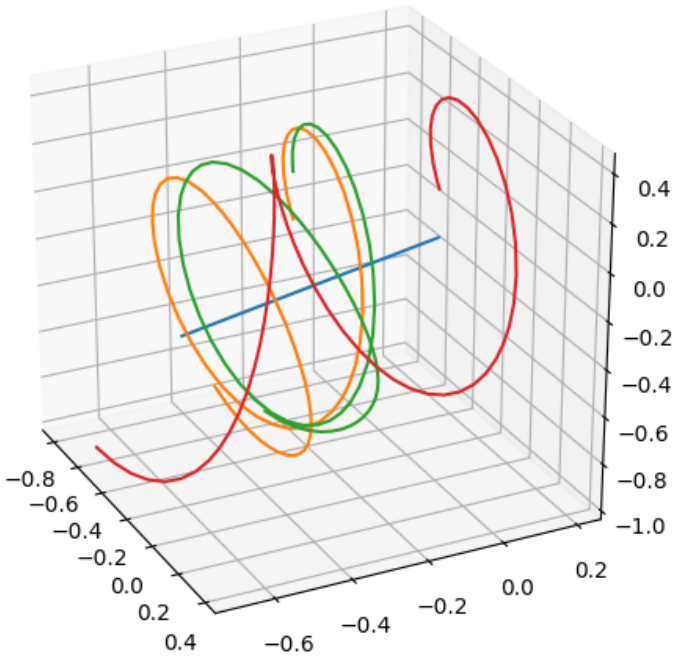


Fig. 6.3 A cuboid is simulated spinning about its 2nd principal axis of inertia while translating linearly. Due to the intermediate axis theorem small instabilities in the rotation build quickly causing rapid flips in orientation. Despite these rapid flips the linear motion of the centre of mass is unaffected. Blue: the path of the centre of mass, Green, Red, Orange: the path of several vertices on the cuboid as it undergoes a flip in orientation.

where Ψ_t is the momentum bivector at time t expressed in the world frame and W_{bt} is the resultant external wrench acting on the body expressed in the body frame. From this point on we will drop the t subscript and simply state that all variables are functions of time. From our discussion in the previous section we know that we can further expand Ω using the inverse inertia tensor Q^{-1}:

$$\Omega = Q^{-1}[\tilde{R}\Psi R].$$

Re-writing the time derivative of the state with this equation for Ω gives:

$$\dot{Y} = \begin{bmatrix} \dot{R} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} -\frac{1}{2}RQ^{-1}[\tilde{R}\Psi R] \\ RW_b\tilde{R} \end{bmatrix}.$$

6.6 Constrained dynamics via virtual power

Unconstrained dynamics, while important, do not allow us to represent all the types of motion that we see in the real world around us. In many practical situations we are faced with the problem of constrained motion. Consider modelling a rigid body that can move dynamically under external forces but is constrained so that one or more points lie on a surface or a situation where a rigid body is constrained such that it can translate but not rotate. These are the types of problem we will attack here.

To impose a constraint on our dynamics model we will use the concept of a **reaction wrench**. The reaction wrench provides a combined external force and moment that acts on the rigid body in addition to the other external wrenches and, in doing so, forces the body to move in a way that respects the constraints. We will write W_b as the sum of external wrenches, S, plus some reaction wrench, F, caused by the constraints. As we already know S, all we need to calculate F is the value of W_b required to keep the constraints valid.

In traditional constrained dynamics work the concepts of virtual work and virtual power are widespread. In the virtual work/virtual power literature constraints are enforced by imagining several independent virtual reaction forces and moments at the constraint position and ensuring that any velocity of the body produces zero power against these forces/moments. In the screw framework that we have developed, the virtual power, P, produced by a virtual world frame wrench, T, when the body moves with a body frame screw velocity Ω is given by:

$$P = \Omega \wedge (\tilde{R}TR)$$

and is of the form

$$P = pI_3 n_\infty$$

where p is a virtual scalar power. Differentiating this gives:

$$\dot{P} = \dot{\Omega} \wedge (\tilde{R}TR) + \Omega \wedge \left(\dot{\tilde{R}}TR + \tilde{R}(\dot{T}R + T\dot{R})\right).$$

We can now substitute in our dynamics equation for $\dot{\Omega}$:

$$\dot{\Omega} = Q^{-1}[\dot{\tilde{R}}\Psi R + W_b + \tilde{R}\Psi\dot{R}],$$

$$\dot{P} = Q^{-1}[\dot{\tilde{R}}\Psi R + W_b + \tilde{R}\Psi\dot{R}] \wedge (\tilde{R}TR) + \Omega \wedge \left(\dot{\tilde{R}}TR + \tilde{R}(\dot{T}R + T\dot{R})\right).$$

Setting the virtual power and rate of change of virtual power to 0 gives us the virtual power condition for our constraint. First:

$$0 = \Omega \wedge (\tilde{R}TR)$$

tells us that the virtual wrench must be parallel to the screw velocity. Setting the rate of change of virtual power to be zero allows us to write:

$$Q^{-1}[\dot{\tilde{R}}\Psi R + W_b + \tilde{R}\Psi\dot{R}] \wedge (\tilde{R}TR) = -\Omega \wedge \left(\dot{\tilde{R}}TR + \tilde{R}(\dot{T}R + T\dot{R})\right)$$

$$Q^{-1}[W_b] \wedge (\tilde{R}TR) + Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] \wedge (\tilde{R}TR) = -\Omega \wedge \left(\dot{\tilde{R}}TR + \tilde{R}(\dot{T}R + T\dot{R})\right)$$

$$Q^{-1}[W_b] \wedge (\tilde{R}TR) = -Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] \wedge (\tilde{R}TR) - \Omega \wedge \left(\dot{\tilde{R}}TR + \tilde{R}(\dot{T}R + T\dot{R})\right)$$

If T is a static constraint we can specify that $\dot{T} = 0$ leaving us with:

$$Q^{-1}[W_b] \wedge (\tilde{R}TR) = -Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] \wedge (\tilde{R}TR) - \Omega \wedge \left(\dot{\tilde{R}}TR + \tilde{R}T\dot{R}\right).$$

Which can again be solved for W_b and hence F.

If we specify the way that T varies with time we can add curved surface constraints. Consider a situation in which a rigid body is constrained such that one point A always touches a sphere centred at point V. Given the point is always touching the sphere we know that T must always be parallel to the line joining A and V, we would therefore write:

$$T = A \wedge V \wedge n_\infty.$$

Taking a time derivative of this we see:

$$\dot{T} = \dot{A} \wedge V \wedge n_\infty.$$

As A is driven by the rotor R, ie:

$$A = RA_0\tilde{R}$$

we get:

$$\dot{A} = \dot{R}A_0\tilde{R} + RA_0\dot{\tilde{R}}$$

and so:

$$\dot{T} = (\dot{R}A_0\tilde{R} + RA_0\dot{\tilde{R}}) \wedge V \wedge n_\infty.$$

We can then directly substitute this into:

$$Q^{-1}[W_b] \wedge (\tilde{R}TR) = -Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] \wedge (\tilde{R}TR) - \Omega \wedge \left(\dot{\tilde{R}}TR + \tilde{R}(\dot{T}R + T\dot{R})\right)$$

and so calculate W_b. To constrain this same point to a circle we would add an additional planar constraint, ie. the point must lie on the plane in which the circle lies and on the sphere of which the circle is the equator.

6.7 Constrained dynamics by pinned multivectors

Consider a geometric primitive represented by multivector U in the body frame and the same geometric primitive represented by multivector V when expressed in the world frame. These two multivectors can be related by the rotor R:

$$V = RU\tilde{R}$$

or equivalently:

$$U = \tilde{R}VR.$$

Taking first and second derivatives gives us the expressions:

$$\dot{U} = \dot{\tilde{R}}VR + \tilde{R}(\dot{V}R + V\dot{R}), \tag{6.18}$$

$$\begin{aligned}\ddot{U} &= \ddot{\tilde{R}}VR + \dot{\tilde{R}}(\dot{V}R + V\dot{R}) + \dot{\tilde{R}}(\dot{V}R + V\dot{R}) + \tilde{R}(\ddot{V}R + 2\dot{V}\dot{R} + V\ddot{R}) \\ &= \ddot{\tilde{R}}VR + 2\dot{\tilde{R}}(\dot{V}R + V\dot{R}) + \tilde{R}\ddot{V}R + 2\tilde{R}\dot{V}\dot{R} + \tilde{R}V\ddot{R}.\end{aligned} \tag{6.19}$$

The next step is to think about what these expressions mean physically. Essentially we have two 'views' of the same object, one in body space and one in world space. For example we can imagine the U is a point attached to our rigid body and V is a point in the world that that point is also attached to. In a sense we are 'pinning' the rigid body to V by its extremity U.

Lets consider first the case that both of these 'views' of the object are fixed, ie. the position and orientation of U cannot change with respect to the coordinate system of the body and the position and orientation of V cannot change with respect to the origin. Mathematically we are stating that $\dot{U},\ddot{U},\dot{V},\ddot{V} = 0$. If we substitute these values into (6.19) for the time derivatives we end up with the following equation:

$$0 = \ddot{\tilde{R}}VR + \tilde{R}V\ddot{R} + 2\dot{\tilde{R}}V\dot{R}. \tag{6.20}$$

This equation is a constraint on the second time derivative of R that will ensure that U and V do not vary with time. We can go a step further here and substitute our expression for $\ddot{R}$ from equation (6.7), leading to:

$$0 = -\frac{1}{2}\dot{\tilde{\Omega}}\tilde{R}VR - \frac{1}{2}\tilde{\Omega}\dot{\tilde{R}}VR - \frac{1}{2}\tilde{R}VR\dot{\Omega} - \frac{1}{2}\tilde{R}V\dot{R}\Omega + 2\dot{\tilde{R}}V\dot{R}.$$

Now if we substitute in $\dot{R} = -\frac{1}{2}R\Omega$ and $\dot{\tilde{R}} = -\frac{1}{2}\tilde{\Omega}\tilde{R}$:

$$0 = -\frac{1}{2}\dot{\tilde{\Omega}}\tilde{R}VR + \frac{1}{4}\tilde{\Omega}\tilde{\Omega}\tilde{R}VR - \frac{1}{2}\tilde{R}VR\dot{\Omega} + \frac{1}{4}\tilde{R}VR\Omega\Omega + \frac{1}{2}\tilde{\Omega}\tilde{R}VR\Omega.$$

Simplify and gather, substituting $U = \tilde{R}VR$:

$$0 = -\frac{1}{2}\dot{\tilde{\Omega}}U - \frac{1}{2}U\dot{\Omega} + \frac{1}{4}\tilde{\Omega}^2U + \frac{1}{4}U\Omega^2 + \frac{1}{2}\tilde{\Omega}U\Omega.$$

Now separate the terms with $\dot{\Omega}$:

$$\frac{1}{2}\dot{\tilde{\Omega}}U + \frac{1}{2}U\dot{\Omega} = \frac{1}{4}\tilde{\Omega}^2U + \frac{1}{4}U\Omega^2 + \frac{1}{2}\tilde{\Omega}U\Omega.$$

As $\Omega,\dot{\Omega}$ are bivectors their reverse is just a negation:

$$-\frac{1}{2}\dot{\Omega}U + \frac{1}{2}U\dot{\Omega} = \frac{1}{4}\Omega^2U + \frac{1}{4}U\Omega^2 + \frac{1}{2}\Omega U\Omega. \tag{6.21}$$

So far, we have done a lot of algebra but so far appear to be no closer to calculating our reaction wrench. If we calculate an expression for $\dot{\Omega}$ however, we start to make headway towards a solution:

$$\dot{\Omega} = Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\dot{\Psi}R + \tilde{R}\Psi\dot{R}]$$

using $W_b = \tilde{R}\dot{\Psi}R$ we can also write:

$$= Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] + Q^{-1}[W_b] \tag{6.22}$$

and so we can now substitute in on the left hand side of equation (6.21) for $\dot{\Omega}$:

$$LHS = -\frac{1}{2}\left(Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] + Q^{-1}[W_b]\right)U + \frac{1}{2}U\left(Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] + Q^{-1}[W_b]\right).$$

Now we separate out the terms with W_b

$$= -\frac{1}{2}Q^{-1}[W_b]U + \frac{1}{2}UQ^{-1}[W_b] + \left(-\frac{1}{2}Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}]U + \frac{1}{2}UQ^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}]\right)$$

and take all terms not containing W_b onto the right side of the equation. We now have something of the form:

$$-\frac{1}{2}Q^{-1}[W_b]U + \frac{1}{2}UQ^{-1}[W_b] = \text{Some function of } R, \Psi, U.$$

We can rewrite this to use the commutator product:

$$(U \times Q^{-1}[W_b]) = \text{Some function of } R, \Psi, U. \tag{6.23}$$

If we now decide to write our total bivector wrench, W_b, as the sum of external wrenches, S, plus some reaction wrench, F, caused by the constraints:

$$(U \times Q^{-1}[F]) = -(U \times Q^{-1}[S]) + \text{Some function of } R, \Psi, U$$

we now have a constraint expression that fixes the reaction wrench F as a function of the state of the system and the forces applied to it.

For a given R, Ψ, U this constraint is linear in F and can be solved for F so long as we provide a correct basis for the constraint wrench. An important point to make here is that this discussion has been entirely algebra agnostic. This framework works equally well for CGA, PGA or indeed many other geometric algebras, a topic that we will return to later on.

6.8 Geometric objects as constraints

Now that we have identified a means of enforcing constraints via pinned geometric primitives let us have a look at exactly what constraints are imposed by specific choices of this pinned multivector for CGA and PGA.

6.8.1 Point constraint

If we want to pin a specific point in our rigid body to a point in the world we can set an invariant point constraint. In this case U and V are both points and, due to the rotational symmetry of a point our reaction wrench F can only support translational forces and no moments. In this case F has 3 degrees of freedom, corresponding to each of the translational principal screws of inertia. This point constraint can be set in both CGA and PGA.

6.8.2 Point-pair constraint

Consider a bivector G of the form:

$$G = A \wedge B$$

where A and B are CGA points. This object is invariant only under the action of rotation rotors about an axis parallel to the line joining A and B. In this case F has 5 degrees of freedom corresponding to 3 translational forces and 2 moments. This point-pair constraint is specific to CGA.

6.8.3 Direction constraint

Consider a so called 'direction' bivector in CGA of the form:

$$D = d \wedge n_\infty$$

where d is a 3D vector. This object is invariant under the action of all translation rotors and is invariant to rotation rotors with axis of rotation parallel to d, ie. rotors of the form:

$$R = e^{-\frac{\theta}{2} d I_3}$$

where I_3 is the pseudoscalar of 3D space. In this case F only has 2 degrees of freedom corresponding to two moments with axes perpendicular to d. For a PGA equivalent of this constraint a bivector of the form:

$$D = d \wedge e_0$$

can be used to achieve the same thing.

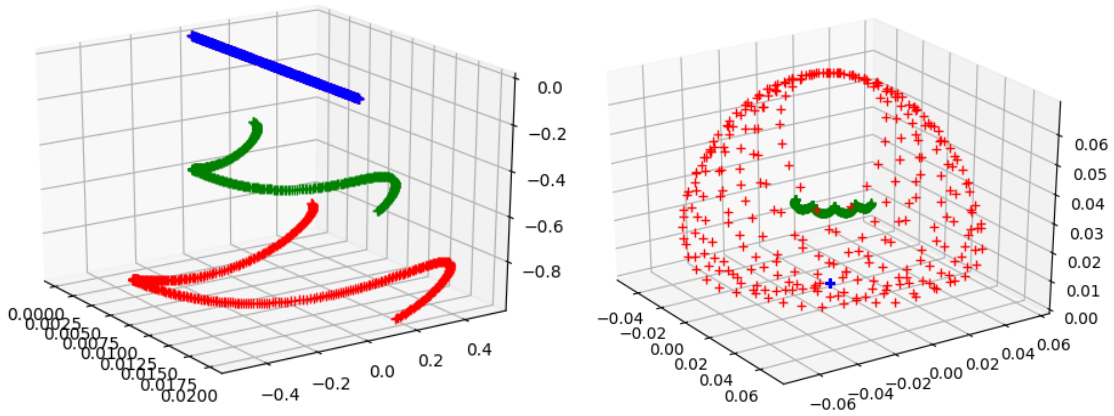


Fig. 6.4 Left: A physical pendulum moves under the effect of gravity and with a starting linear momentum. It is constrained such that a line, coincident with one end of the pendulum shown in blue, is pinned between the body and world reference frames. The symmetry of the line leads to constrained motion along and about the line.
Right: A spinning cone is affected by gravity but is constrained such that its end point, shown in blue, does not move. Precession and nutation are observable in the movement of the centre of mass, shown in green, and a point on the rim of the cone, shown in red.

6.8.4 Flat point constraint

Consider a so called 'flat point' bivector in CGA of the form:

$$D = d \wedge n_\infty - n_\infty \wedge n_0$$

where d is a 3D vector. This object is invariant under the action of all rotation rotors about the point d, but is not invariant to translation. Thus, under the action of the rigid body rotors it behaves like a CGA 1-vector point. For PGA this constraint can again be implemented by a standard PGA point.

6.8.5 Line constraint

A line is invariant to translation along the line and rotation about the line axis. Thus we would expect to be able to support reaction forces orthogonal to the line, and moments with axes orthogonal to the line axis, ie. F has 4 degrees of freedom. In PGA a line is a bivector and is formed by the intersection of two planes, in CGA a line can be represented directly as the wedge of two points and n_∞ or dually as given in equation (6.2). Both the dual and direct CGA form work fine for pinning.

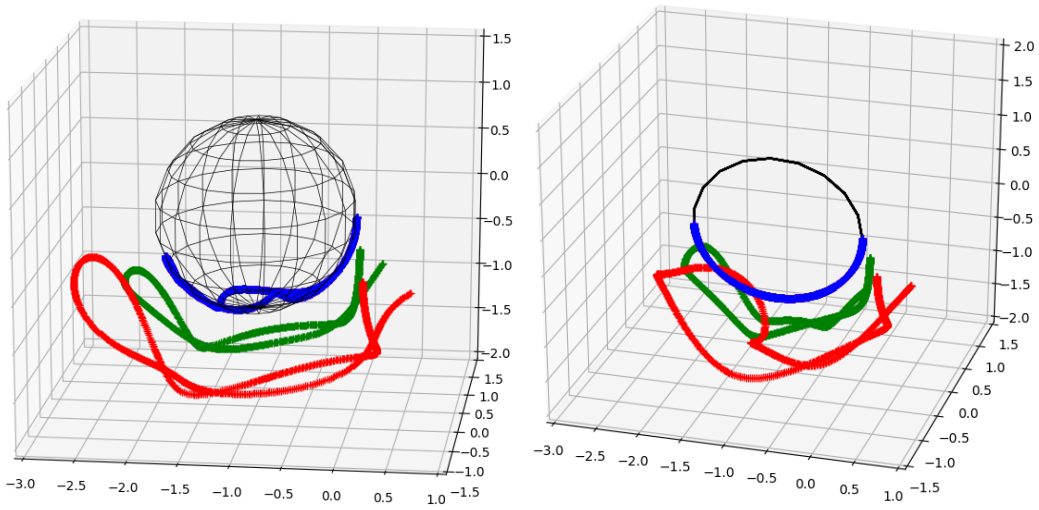


Fig. 6.5 A physical pendulum moves under gravity and is constrained such that one end of it, shown in blue, is always in contact with the surface of an object. The green trace shows the midpoint of the pendulum and the red shows the free end. Left: a sphere. Right: a circle.

6.8.6 Circle constraint

A circle is invariant to rotation about the axis of the circle only. Thus we would expect to be able to support reaction forces in all directions and all moments other than the one with axis parallel to the circle axis, in other words F has 5 degrees of freedom. In retrospect this should be unsurprising as the dual to a 3-vector CGA circle is an imaginary point-pair bivector, and we have already seen that point-pairs have this same form of invariance.

6.8.7 Plane constraint

A plane is invariant to translation in plane and rotation about axes parallel to its normal direction. Thus it can support one direction of force and two directions of moments giving 3 degrees of freedom to F. Both CGA and PGA can use planes for pinning.

6.8.8 Sphere constraint

A sphere is invariant to all rotation about its centre, but not to translation. Thus it acts like a point at the sphere centre and F can support translation reaction forces only giving it 3 degrees of freedom. CGA can represent spheres directly as the outer product of 4 points. In PGA this type of constraint would have to be enforced by pinning a point at the centre of the sphere.

6.9 Pinning parametric multivectors paths

So far in our construction of multivector pinning constraints we have assumed that the objects we are pinning are static in both the world and body frame. When working with constrained dynamics in the real world we often want to pin parts of our rigid body to moving things in the real world, such as a manipulator attached to the moving end-point of a robot, or a flywheel fixed in a moving vehicle. Consider once again equation (6.19):

$$\dot{U} = \dot{\tilde{R}}VR + 2\dot{\tilde{R}}(\dot{V}R + V\dot{R}) + \tilde{R}\dot{V}R + 2\tilde{R}\dot{V}\dot{R} + \tilde{R}V\dot{R}.$$

In the previous section we enforced static multivector constraints by setting $\dot{U}, \ddot{U}, \dot{V}, \ddot{V}$ to zero, rearranging to isolate the $\dot{R}$ terms and solving the resultant linear equation for W_b. Now we will relax the static constraint and consider the cases when U, V are known **time varying** multivector functions, ie when $\dot{U}, \ddot{U}, \dot{V}, \ddot{V} \neq 0$.

First note we can still rearrange to separate terms in $\dot{R}$:

$$\dot{\tilde{R}}VR + \tilde{R}V\dot{R} = -2\dot{\tilde{R}}(\dot{V}R + V\dot{R}) - \tilde{R}\dot{V}R - 2\tilde{R}\dot{V}\dot{R} + \dot{U}.$$

In fact, if we continue as in our previous analysis by breaking up $\dot{R}$ as a function of $\dot{\Omega}$ and extracting W_b:

$$\begin{aligned}
\dot{\tilde{R}}VR + \tilde{R}V\dot{R} &= (-\frac{1}{2}\dot{\tilde{\Omega}}\tilde{R} - \frac{1}{2}\tilde{\Omega}\dot{\tilde{R}})VR + \tilde{R}V(-\frac{1}{2}R\dot{\Omega} - \frac{1}{2}\dot{R}\Omega) \\
&= -\frac{1}{2}\dot{\tilde{\Omega}}\tilde{R}VR - \frac{1}{2}\tilde{R}VR\dot{\Omega} - \frac{1}{2}\tilde{\Omega}\dot{\tilde{R}}VR - \frac{1}{2}\tilde{R}V\dot{R}\Omega \\
&= \dot{\Omega} \times (\tilde{R}VR) - \frac{1}{2}\tilde{\Omega}\dot{\tilde{R}}VR - \frac{1}{2}\tilde{R}V\dot{R}\Omega \\
&= Q^{-1}[W_b] \times (\tilde{R}VR) + Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] \times (\tilde{R}VR) - \frac{1}{2}\tilde{\Omega}\dot{\tilde{R}}VR - \frac{1}{2}\tilde{R}V\dot{R}\Omega \\
&= Q^{-1}[F] \times (\tilde{R}VR) + Q^{-1}[S] \times (\tilde{R}VR) + Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] \times (\tilde{R}VR) - \frac{1}{2}\tilde{\Omega}\dot{\tilde{R}}VR - \frac{1}{2}\tilde{R}V\dot{R}\Omega
\end{aligned}$$

and so:

$$\begin{aligned}
(\tilde{R}VR) \times Q^{-1}[F] = &-\ddot{U} + Q^{-1}[S] \times (\tilde{R}VR) + 2\dot{\tilde{R}}(\dot{V}R + V\dot{R}) + \tilde{R}\ddot{V}R + 2\tilde{R}\dot{V}\dot{R} \\
&+ Q^{-1}[\dot{\tilde{R}}\Psi R + \tilde{R}\Psi\dot{R}] \times (\tilde{R}VR) - \frac{1}{2}\tilde{\Omega}\dot{\tilde{R}}VR - \frac{1}{2}\tilde{R}V\dot{R}\Omega.
\end{aligned}$$

If we substitute $U = \tilde{R}VR$ we have eventually got to a position where:

$$(U \times Q^{-1}[F]) = \text{Some function of } R, \Psi, U, \dot{U}, \ddot{U}, V, \dot{V}, \ddot{V}, S.$$

Again this is a linear function in F and so solvable as long as it is of sufficient rank.

What this means practically is that we can set U and V to follow any desired path we like in their respective spaces and extract the reaction forces and moments acting on the body that are required to keep them pinned to each other.

6.10 Pinning linear functions of parametric multivector paths

In the previous two sections we dealt directly with transformations that pin static multivectors or time varying multivector paths directly to each other in space. In many practical situations what we would really like to pin is a linear function of one multivector to another. For example we could pin the outer product of a point in the body frame and a plane in the world frame to 0, effectively forcing them to be coincident without specifying anything about their relative orientation (unlike in the transformed plane invariant case). Mathematically we can express our linear function constraint as $A[\,]$ and time derivatives as:

$$U = A\left[\tilde{R}VR\right], \quad \dot{U} = A\left[\dot{\tilde{R}}VR + \tilde{R}(\dot{V}R + V\dot{R})\right],$$
$$\ddot{U} = A\left[\ddot{\tilde{R}}VR + 2\dot{\tilde{R}}(\dot{V}R + V\dot{R}) + \tilde{R}\ddot{V}R + 2\tilde{R}\dot{V}\dot{R} + \tilde{R}V\ddot{R}\right].$$

Once again we can rearrange:

$$A\left[\ddot{\tilde{R}}VR + \tilde{R}V\ddot{R}\right] = A\left[-2\dot{\tilde{R}}(\dot{V}R + V\dot{R}) - \tilde{R}\ddot{V}R - 2\tilde{R}\dot{V}\dot{R}\right] + \ddot{U}$$

leading to an equation of the form:

$$A\left[(\tilde{R}VR) \times Q^{-1}[F]\right] = A\left[-((\tilde{R}VR) \times Q^{-1}[S]) + \text{Some function of } R, \Psi, U, \dot{U}, \ddot{U}, V, \dot{V}, \ddot{V}\right] - \ddot{U}.$$

Again this is linear and solvable as before. Figure 6.5 shows the simulation with the Clifford Python library [52] of two cases in which the linear function is the outer product with one end of a physical pendulum.

6.11 Mapping Screw Velocity to Lie Algebra Velocity

Throughout this chapter so far we have represented the derivative of the state of the body on the motor manifold as $\dot{R}$. In practice numerical integration schemes which integrate $\dot{R}$ will undoubtedly accumulate errors and so wander off the motor manifold. Depending on the application this may or may not be a problem [9]. We can, however, nicely side step the problem by directly mapping the velocity screw to a velocity in a suitable se(3) Lie algebra

that generates the rotor R. If we label the generator for the current position and orientation in the Lie algebra as Φ which maps to the current rotor with a function Q then we are interested in finding a function K that does the following:

$$R = Q(\Phi), \quad \dot{\Phi} = K(\Omega, \Phi).$$

We will refer to this function K as the 'kinematic equation' for the given Lie algebra to Lie group mapping and will have a look at its form for a few choices of Q.

6.11.1 Exponential Mapping and the Bortz Equation

One commonly used mapping for SE(3) is the exponential mapping:

$$R = e^{\Phi}.$$

Conveniently, the kinematic equation for the exponential mapping of the motor bivectors has already been derived in [14] and a screw Lie algebra version in [89] and appears again (with a corrected typo) at the end of Section 4.5 in [90]. The result in [89, 90] is derived via idempotents and nilpotents of the adjoint matrix representation of the se(3) Lie algebra but it is readily translatable into our own notation:

$$\begin{aligned}\dot{\Phi} = \Omega_w + \frac{1}{2}\Phi \times \Omega_w + \left(\frac{2}{|\theta|^2} + \frac{|\theta| + 3\sin|\theta|}{4|\theta|(\cos|\theta| - 1)}\right)\Phi \times [\Phi \times \Omega_w] \\ + \left(\frac{1}{|\theta|^4} + \frac{|\theta| + \sin|\theta|}{4|\theta|^3(\cos|\theta| - 1)}\right)\Phi \times [\Phi \times [\Phi \times [\Phi \times \Omega_w]]]\end{aligned} \tag{6.24}$$

where $|\theta| = \sqrt{\Omega_w \cdot \tilde{\Omega}_w}$.

The exponential se(3) kinematic equation is also the subject of Section 5.3 of Liam Candy's PhD thesis [14], and while the set up of the problem is certainly correct we were unable to make their equation 5.41 work in our implementations. Suspecting simply a typo somewhere in their derivations of the derivatives we can calculate the derivatives ourselves and check the final result. We start with the following setup, following mostly along the lines of [14]. Our objective is to calculate $\dot{\Phi}$ as a function of Φ and Ω_b or Ω_w. First, we note that it is possible to write a motor bivector in the form:

$$\Phi = \alpha B + t n_\infty \tag{6.25}$$

where B is a rotation bivector and t is a 3DGA vector. We then follow [14] choosing to define two quantaties:

$$d = 2\langle R\tilde{R}_\alpha\rangle_2 \cdot n_0, \quad \Omega_b = \Omega_\alpha + v n_\infty$$

where:

$$R = \exp\frac{-\Phi}{2}, \quad R_\alpha = \cos\frac{\alpha}{2} - B\sin\frac{\alpha}{2}$$

$$d_\parallel = \frac{1}{2}(d + BdB), \quad d_\perp = \frac{1}{2}(d - BdB)$$

$$t = t_\perp + t_\parallel, \quad t_\perp = d_\perp, \quad t_\parallel = \frac{d_\parallel R_\alpha}{\operatorname{sinc}\frac{\alpha}{2}} = \frac{\alpha d_\parallel R_\alpha}{2\sin\frac{\alpha}{2}}$$

We can then write an expression for $\dot{\Phi}$ as the time derivative of (6.25):

$$\dot{\Phi} = \dot{\alpha}B + \alpha\dot{B} + \dot{t}n_\infty.$$

This is made up of the following components:

$$\dot{\alpha} = -B \cdot \Omega_\alpha, \quad \dot{B} = \frac{1}{2}\cot\frac{\alpha}{2}(\Omega_\alpha + (\Omega_\alpha \cdot B)B) - \frac{1}{2}\langle B\Omega_\alpha\rangle_2$$

$$\dot{t} = \dot{t}_\perp + \dot{t}_\parallel$$

where:

$$\dot{t}_\parallel = \frac{\dot{\alpha}d_\parallel R_\alpha}{2\sin\frac{\alpha}{2}} + \frac{\alpha\dot{d}_\parallel R_\alpha}{2\sin\frac{\alpha}{2}} + \frac{\alpha d_\parallel \dot{R}_\alpha}{2\sin\frac{\alpha}{2}} + (\alpha d_\parallel R_\alpha)\frac{\dot{\alpha}\cos\frac{\alpha}{2}}{2(\cos\alpha - 1)}$$

$$\dot{t}_\perp = \dot{d}_\perp = \frac{1}{2}(\dot{d} - \dot{B}dB - B\dot{d}B - BdB\dot{)}$$

$$\dot{d} = 2\langle\dot{R}\tilde{R}_\alpha + R\dot{\tilde{R}}_\alpha\rangle_2 \cdot n_0, \quad \dot{d}_\parallel = \frac{1}{2}(\dot{d} + \dot{B}dB - B\dot{d}B - BdB\dot{)}$$

$$\dot{R}_\alpha = -\frac{1}{2}R_\alpha\Omega_\alpha, \quad \dot{R} = -\frac{1}{2}R\Omega_b$$

If we restrict ourselves to so(3) these equations will produce the same answer as that of the Bortz equation [7] familiar to practitioners from the field of strapdown inertial navigation.

6.11.2 Cayley Mapping

An alternative mapping to the exponential that is simple and potentially useful is the Cayley mapping [55, 96]. For small rotations and translations the Cayley mapping approximates the exponential however it diverges somewhat as we move further from the origin. Matrix versions of the Cayley map are well known and the kinematic equations for the matrix

versions of this map have been studied before in the aeronautics literature [93]. We have failed to find any previous attempts at the GA version of the kinematic equation however we can reuse much of the logic of the matrix derivation, simply substituting transposes for tildes. Start with the expression for the mapping:

$$R = (1-\Phi)(1+\Phi)^{-1}, \tag{6.26}$$

$$R(1+\Phi) = 1-\Phi.$$

Take the time derivative:

$$\dot{R}(1+\Phi) + R\dot{\Phi} = -\dot{\Phi}.$$

Rearrange:

$$(1+R)\dot{\Phi} = -\dot{R}(1+\Phi) = \frac{1}{2}\Omega_w R(1+\Phi) = \frac{1}{2}\Omega_w(1-\Phi).$$

Now we will seek a reformulation of $1+R$:

$$1+R = (1+\Phi)(1+\Phi)^{-1} + (1-\Phi)(1+\Phi)^{-1}$$

$$= (1+\Phi+1-\Phi)(1+\Phi)^{-1} = 2(1+\Phi)^{-1}.$$

This allows us to write:

$$(1+R)\dot{\Phi} = 2(1+\Phi)^{-1}\dot{\Phi} = \frac{1}{2}\Omega_w(1-\Phi)$$

and so we are left with:

$$\dot{\Phi} = \frac{1}{4}(1+\Phi)\Omega_w(1-\Phi). \tag{6.27}$$

6.11.3 Outer Exponential Mapping

The final mapping that we will consider is the so called 'outer exponential' mapping as presented in [96]. This mapping is defined as taking a Taylor series of the exponential but replacing geometric products with wedge products, for an algebra with maximum grade 5 or below this can be written as:

$$R = \exp_{\wedge}(\Phi) = \frac{1+\Phi+\frac{1}{2}\langle\Phi^2\rangle_4}{\sqrt{1-\langle\Phi^2\rangle}}. \tag{6.28}$$

Again we were unable to find a GA kinematic equation for this mapping in the existing literature and so present our own as follows. First we take a time derivative of the outer

exponential:

$$R = \frac{1+\Phi+\frac{1}{2}\langle\Phi^2\rangle_4}{\sqrt{1-\langle\Phi^2\rangle}} = \left(1+\Phi+\frac{1}{2}\langle\Phi^2\rangle_4\right)\left(1-\langle\Phi^2\rangle\right)^{-\frac{1}{2}}$$

$$\dot{R} = \left(\dot{\Phi}+\frac{1}{2}\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle_4\right)\left(1-\langle\Phi^2\rangle\right)^{-\frac{1}{2}} + \left(1+\Phi+\frac{1}{2}\langle\Phi^2\rangle_4\right)\left(\frac{1}{2}\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle\right)\left(1-\langle\Phi^2\rangle\right)^{-\frac{3}{2}}$$

$$= \frac{\dot{\Phi}+\frac{1}{2}\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle_4}{\left(1-\langle\Phi^2\rangle\right)^{\frac{1}{2}}} + \frac{R\frac{1}{2}\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle}{1-\langle\Phi^2\rangle}.$$

Then, given that in (6.3) we have defined:

$$\Omega_w = -2\dot{R}\tilde{R} = 2R\dot{\tilde{R}}$$

we can therefore write:

$$\Omega_w = \frac{-2\left(\dot{\Phi}+\frac{1}{2}\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle_4\right)\tilde{R}}{\left(1-\langle\Phi^2\rangle\right)^{\frac{1}{2}}} - \frac{\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle}{1-\langle\Phi^2\rangle}.$$

Noticing that:

$$\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle \equiv 2\langle\dot{\Phi}\Phi\rangle$$

$$\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle_4 \equiv 2\langle\dot{\Phi}\Phi\rangle_4$$

and rewriting our equation for Ω_w as:

$$\left(1-\langle\Phi^2\rangle\right)\Omega_w R = -2\left(1-\langle\Phi^2\rangle\right)^{\frac{1}{2}}\left(\dot{\Phi}+\frac{1}{2}\langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle_4\right) - \langle\dot{\Phi}\Phi+\Phi\dot{\Phi}\rangle R$$

gives us:

$$\left(1-\langle\Phi^2\rangle\right)\Omega_w R = -2\left(1-\langle\Phi^2\rangle\right)^{\frac{1}{2}}\left(\dot{\Phi}+\langle\dot{\Phi}\Phi\rangle_4\right) - 2\langle\dot{\Phi}\Phi\rangle R.$$

We can now equate grade 0 elements:

$$\langle\left(1-\langle\Phi^2\rangle\right)\Omega_w R\rangle = -2\langle\dot{\Phi}\Phi\rangle\langle R\rangle$$

and so:

$$\langle\dot{\Phi}\Phi\rangle = \frac{\left(1-\langle\Phi^2\rangle\right)\langle\Omega_w R\rangle}{-2\langle R\rangle}.$$

We can also equate grade 2 elements:

$$\langle\left(1-\langle\Phi^2\rangle\right)\Omega_w R\rangle_2 = -2(1-\langle\Phi^2\rangle)^{\frac{1}{2}}\dot{\Phi} - 2\langle\dot{\Phi}\Phi\rangle\langle R\rangle_2.$$

This, after some rearrangement, leaves us with the form of the kinematic equation for the outer exponential:

$$\begin{aligned}\dot{\Phi} &= \frac{\left(1-\langle\Phi^2\rangle\right)\langle\Omega_w R\rangle_2 + 2\langle\dot{\Phi}\Phi\rangle\langle R\rangle_2}{-2(1-\langle\Phi^2\rangle)^{\frac{1}{2}}} \\ &= \frac{\left(1-\langle\Phi^2\rangle\right)\langle\Omega_w R\rangle_2}{-2(1-\langle\Phi^2\rangle)^{\frac{1}{2}}} + \langle\dot{\Phi}\Phi\rangle\frac{\langle R\rangle_2}{-(1-\langle\Phi^2\rangle)^{\frac{1}{2}}} \\ &= \frac{\left(1-\langle\Phi^2\rangle\right)\langle\Omega_w R\rangle_2}{-2(1-\langle\Phi^2\rangle)^{\frac{1}{2}}} + \frac{\left(1-\langle\Phi^2\rangle\right)\langle\Omega_w R\rangle}{-2\langle R\rangle}\frac{\langle R\rangle_2}{-(1-\langle\Phi^2\rangle)^{\frac{1}{2}}} \\ &= -\frac{1}{2}(1-\langle\Phi^2\rangle)^{\frac{1}{2}}\langle\Omega_w R\rangle_2 + \frac{(1-\langle\Phi^2\rangle)^{\frac{1}{2}}\langle\Omega_w R\rangle\langle R\rangle_2}{2\langle R\rangle}.\end{aligned}$$

We can write this neatly as:

$$\dot{\Phi} = \frac{1}{2}\sqrt{1-\langle\Phi^2\rangle}\left[-\langle\Omega_w R\rangle_2 + \frac{\langle\Omega_w R\rangle\langle R\rangle_2}{\langle R\rangle}\right]. \tag{6.29}$$

6.12 Conclusions

In this chapter we have looked at forces, moments, free and constrained dynamics in both CGA and PGA. As well as considering how to apply virtual power as a constraint mechanism in our GA formulations we have constructed a novel technique for constrained dynamics in GA via the concept of multivector pinning. While in this chapter we have only considered two algebras, CGA and PGA, the techniques are expected to work across the board for algebras with easily representable line elements and motor bivectors. Using other higher dimensional algebras such as Cl(4,4) [31], Cl(8,2) [33] or even Cl(9,6) [12] with this technique in the future should allow for easy configurations of exotic constraints such as pinning dynamic objects to the surface of quadrics.

Chapter 7

The Kinematics of Multi-body Systems in Geometric Algebra

Wealth, beauty and fame are transient. When those are gone, little is left except the need to be useful.

Gene Tierney

Abstract

Screw Theory is a framework for analysing articulated mechanisms and performing statics and dynamics calculations that has found much success in the kinematic analysis of mechanisms. In this chapter we consider the embedding of Screw Theory into another extremely powerful framework for robotics, namely Geometric Algebra (GA). We start by rederiving well known results for the accumulation of twists along kinematic chains within our GA framework before turning our attention to the analysis of kinematic pairs. We derive an elegant representation of kinematic pairs via bilinear functions of basic geometric primitives and use this to describe the most common types of robotic joints. We then address multi-body systems and, using the Delta robot as a case study, we compare the screw theoretic approach to a direct differentiation method for extracting the Jacobians of the system.

7.1 Introduction

Modern manufacturing is increasingly mechanised and automated. The drive for automation has led to a need for advanced modelling capabilities and many modern analysis frameworks

have been developed as a result. Perhaps the most successful of these frameworks for the analysis for 3D mechanisms is known as Screw Theory. Screw Theory is, unsurprisingly perhaps, concerned with the study of 'screws'. In its modern form Screw Theory was first described by Sir Robert Stawell Ball in his 'Treatise on the Theory of Screws' [2], however the mathematical roots that underlie this remarkable field come from Projective Geometry [86] and the study of Lie groups and Lie algebras [89]. More recent proponents of Screw Theory include Hunt [66], Selig [90], Davidson [20], Martins [97, 91], Featherstone [36], Gallardo Alvarado [40], Pottmann [86], Minguzzi [79], Lipkin [77] and indeed many others.

Alongside the development of Screw Theory, and indeed building on some of the same fundamental mathematics, we have seen the rise of the Clifford/Geometric Algebra (GA) as a robotics modelling framework [70, 39, 69, 60, 101, 59, 1, 94]. With modern computing capabilities, the high level description of geometry that these algebras afford allows researchers elegant, concise and coordinate-free descriptions of physically intricate mechanisms and constraints. Many of the modern applications of GA are in the analysis of conformal [30] and Euclidean motions [44], however there have been only a few attempts to properly embed the tools of modern Screw Theory into GA [55, 96]. This chapter, along with the previous one, is an attempt to lay out the overlap between the two fields with language and ideas familiar to both Screw Theory and GA practitioners.

7.2 Geometric Algebra

In this chapter we will, by default, work with Conformal Geometric Alegbra (CGA). CGA adds two more basis vectors, e and $\bar{e}$, to the original basis vectors of 3D Euclidean space, giving a complete basis for the 5D space with the following signature: $e_1^2 = e_2^2 = e_3^2 = e^2 = 1$ and $\bar{e}^2 = -1$. These extra basis vectors are used to define two null vectors: $n_\infty = e + \bar{e} \equiv n$ and $n_0 = \frac{\bar{e}-e}{2} \equiv -\frac{\bar{n}}{2}$ – note that the $(n, \bar{n})$ notation was that originally used when Hestenes first introduced this model in [56]. The mapping from a 3D vector, x, to its corresponding CGA vector, X, is given by:

$$X = F(x) = \frac{1}{2}\left(x^2 n + 2x - \bar{n}\right) \equiv \frac{1}{2}x^2 n_\infty + x + n_0. \tag{7.1}$$

which is often referred to as: $\mathrm{up}(x) = X$. We can invert this mapping quite easily so long as we remember to normalise the CGA point such that $X \cdot n_\infty = -1$:

$$x = F^{-1}(X) = \mathrm{down}(X) = \frac{-(X \wedge E_0)E_0}{X \cdot n_\infty}$$

where $E_0 = n_\infty \wedge n_0$. There are many excellent expositions on CGA in the literature and so we will refrain from a lengthy introduction of all the features of the algebra in this chapter, instead simply preferring to remind the reader of immediately relevant facts about the framework as we go along. If the reader is looking for a more thorough coverage of CGA we would recommend they turn to the excellent book *Geometric Algebra for Computer Science* by Dorst, Fontijne and Mann [29].

One of the important sections of the CGA framework that this chapter will deal with is the set of bivectors that form the generators of rotors that perform Euclidean motion, we will refer to these bivectors as the motor bivectors. The motor bivector basis set contains 6 elements, reflecting the 6 degrees of freedom present in rigid body motion. A common choice for the motor bivector basis in CGA is the following ordered set:

$$m_i \in \{e_1 I_3, e_2 I_3, e_3 I_3, e_1 n_\infty, e_2 n_\infty, e_3 n_\infty\}$$

where I_3 represents the 3D pseudo-scalar $e_1 \wedge e_2 \wedge e_3$. This set would have a corresponding reciprocal frame as follows:

$$m^i \in \{-e_1 I_3, -e_2 I_3, -e_3 I_3, e_1 n_0, e_2 n_0, e_3 n_0\}$$

such that $m_i \cdot m^j = 1$ if $i = j$ and $m_i \cdot m^j = 0$ if $i \neq j$. The full set of motor bivectors is then a linear combination of this motor bivector basis. Introducing 6 scalar parameters γ_i we can write a general motor bivector T as:

$$T = \sum_{i=1}^{6} \gamma_i m_i.$$

An alternative and increasingly popular GA framework to work in is the Plane-based or Projective Geometric Algebra (PGA) [28, 44]. This algebra has signature Cl(3,0,1), meaning it has 3 basis vectors that square to $+1$, 0 basis vectors that square to -1 and one null basis vector that squares to 0. It is a subalgebra of CGA that contains only the 'flat' elements and the Euclidean rotors [73, 65]. Restricting the algebra in this way is useful for certain applications that do not require the round elements of CGA and especially as the reduced dimensionality can produce significant speedups for some numerical packages. PGA also contains the motor bivectors with its null basis vector, e_0 taking the place of n_∞. Due to its degenerate signature PGA cannot use reciprocal frames in the same way as CGA but this restriction can be neatly sidestepped via pseudo-reciprocal frames as described in [43] and explicitly worked through for the motor bivectors in Chapter 6. While PGA is not a direct focus of this specific chapter a sharp eyed reader will note that almost all of the formulae in

the Screw Theory framework developed here will work with no, or only minor, modifications in PGA.

Whichever GA you choose, the motor bivectors, when exponentiated, produce rotors that can implement rigid body motions. In general it is possible to split T into two commuting bivectors, one a generator of rotational motion about a line in the world and one a generator of translational motion along that line. This combination of rotational and translational motion leads to the identification of the motor bivectors as 'screws' and if we were to take the 6 scalar parameters γ_i and arrange them in a 6x1 vector we would get the vector space that is the core subject matter of the field of Screw Theory. This chapter is one of a pair which focus on the embedding of Screw Theory into GA and is the less theoretical of the two, here we will remind readers of any information from the previous chapter when needed but will focus more on the practical realities of using Screw Theory ideas in GA.

7.3 Twists in Kinematic Chains

Consider a rigid body i. The rotor R_i transforms between the world frame and an arbitrary frame fixed to, but not principal axis aligned with, the body. Specifically it transforms objects from the fixed frame to the world frame. The rotor V_i then transforms from a fixed principal axis aligned frame to that arbitrary fixed frame. The combined rotor that transforms directly from the principal axis aligned frame to the world frame is written S_i. Consider another body, j, again with a rotor R_j from fixed frame to world frame, V_j from principal axis aligned fixed frame to arbitrary fixed frame and S_j from principal axis aligned fixed to world frame . Figure 7.1 shows a visual depiction of this. The rotor that transforms between fixed frames of limb i to limb j is labelled R_{ij}:

$$R_{ij}R_i = R_j. \tag{7.2}$$

Using $\sim$ to represent the standard geometric algebra reversion operator we can write:

$$R_{ij} = R_j\tilde{R}_i. \tag{7.3}$$

Now imagine that these bodies form part of a jointed kinematic chain with a sequence of limbs. The position and orientation of each limb relative to the world origin is defined by the rotor S_i, which can in turn be written $S_i = R_iV_i$.

Each of the fixed frames attached to the limbs has a combined rotational and translational velocity in the world frame that we can represent as a motor bivector which we will label Ω_i. These motor bivectors representing combined rotational and translational velocity are known in the Screw Theory literature as a 'velocity screw' or a 'twist'. From standard results we

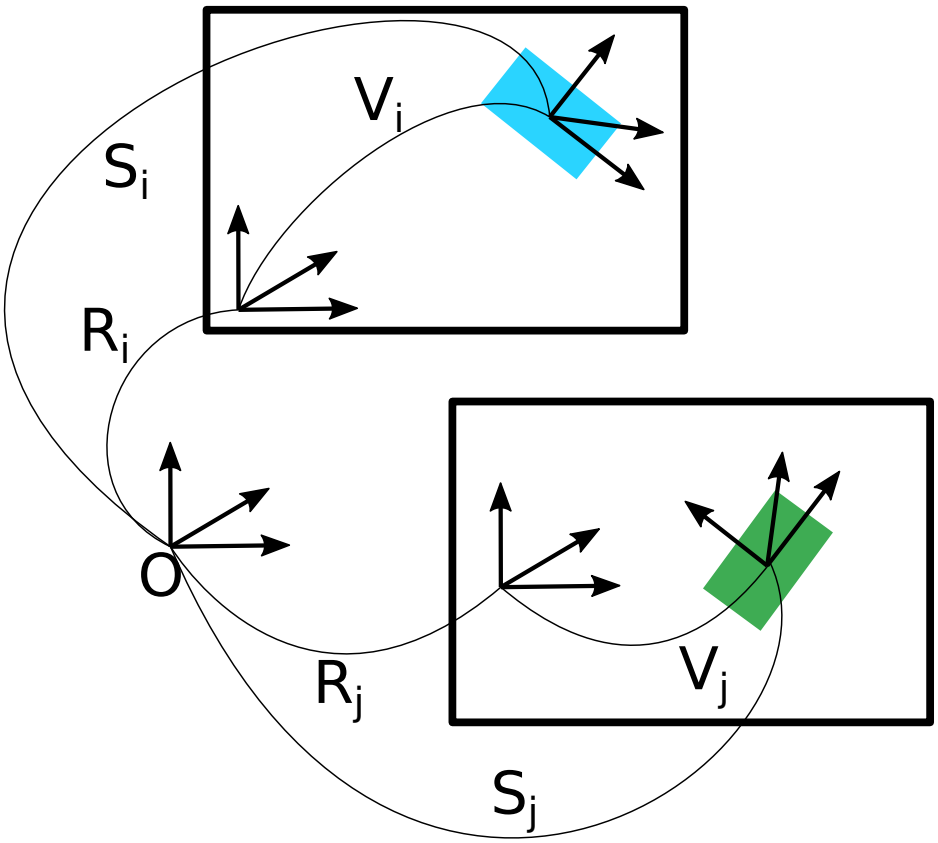


Fig. 7.1 Rigid body i is shown in blue. Rotor V_i transforms from a fixed principal axis aligned frame to an arbitrary frame fixed to the body. Rotor R_i transforms from the arbitrary fixed frame to the world frame and rotor S_i transforms directly from the principal axis aligned body frame to the world frame. Likewise for body j shown in green.

know we can write the time derivative of these rotors as:

$$\dot{S}_i = -\frac{1}{2}\Omega_i S_i. \tag{7.4}$$

We can decompose this rotor into R_i and V_i:

$$\dot{S}_i = -\frac{1}{2}\Omega_i R_i V_i. \tag{7.5}$$

We can also explicitly take the time derivative of S_i, which simplifies as $\dot{V}_i = 0$ due to both body frames being fixed relative to one another:

$$\dot{S}_i = \dot{R}_i V_i + R_i \dot{V}_i = \dot{R}_i V_i. \tag{7.6}$$

This implies:

$$\dot{R}_i V_i = -\frac{1}{2}\Omega_i R_i V_i. \tag{7.7}$$

If we right multiply by $\tilde{V}_i$ we can see that Ω_i is the bivector velocity of both S_i and R_i:

$$\dot{R}_i = -\frac{1}{2}\Omega_i R_i. \tag{7.8}$$

This should not surprise us as both body frames are fixed to one another.

Taking the time derivative of equation (7.2) leads to a formula relating the relative velocity screws of the body in the chain:

$$\dot{R}_{ij} R_i + R_{ij} \dot{R}_i = \dot{R}_j \tag{7.9}$$

$$-\frac{1}{2}\Omega_{ij} R_{ij} R_i - \frac{1}{2} R_{ij} \Omega_i R_i = -\frac{1}{2}\Omega_j R_j \tag{7.10}$$

$$-\frac{1}{2}\Omega_{ij} R_j - \frac{1}{2} R_{ij} \Omega_i \tilde{R}_{ij} R_j = -\frac{1}{2}\Omega_j R_j \tag{7.11}$$

$$\Omega_{ij} + R_{ij} \Omega_i \tilde{R}_{ij} = \Omega_j. \tag{7.12}$$

This is a potentially convenient result but there is a particular case that proves to be of special interest. Our results so far are not a function of the rotors V_i at all, just of R_i. We could choose any decomposition of S_i into R_i and V_i at a given time point and our equations will still be valid. The special case we will concentrate on is to choose to instantaneously align the fixed frames with the world frame, in other words we choose at this instant: $R_i = 1$, $V_i = S_i$

and therefore $R_{ij} = 1$. Under these conditions equation (7.12) simplifies to the following:

$$\Omega_{ij} + \Omega_i = \Omega_j. \tag{7.13}$$

This equation is particularly convenient if we are able to measure the relative velocity screw of one limb with respect to another as we can simply accumulate the relative limb velocities along the chain to arrive at the final global limb velocity screw with respect to the world frame. This is well known in traditional Screw Theory and forms the basis of many practical techniques for analysing robots.

A direct result of the definition of Ω is that the time derivative of a geometric object X is given by the commutator product of the object with Ω:

$$\dot{X} = \frac{1}{2}(X\Omega - \Omega X) = X \times \Omega. \tag{7.14}$$

This is a particularly helpful result as it allows us to define geometry associated with our kinematic chains and calculate how they evolve through time as the kinematic chain moves.

7.4 Geometrically Constrained Kinematic Pairs

Consider a pair of adjacent limbs, i and j, in a kinematic chain. These two limbs are connected by some form of joint and together they are known as a kinematic pair [87]. Exactly how they are constrained to move relative to one another is defined by the type of joint. Let us first consider a specific type of joint, one defined by **shared geometry**.

In a shared geometry joint there exists a piece of geometry, which we will label X, that is fixed relative to both frames in the kinematic pair. Practically what this means is that when acted on by the velocity of i or j this object X must have the same $\dot{X}$, or more formally:

$$\dot{X} = X \times \Omega_i = X \times \Omega_j. \tag{7.15}$$

Due to the linearity of the commutator product we can write this as:

$$0 = X \times (\Omega_i - \Omega_j) = -X \times \Omega_{ij}. \tag{7.16}$$

This is an extremely useful result. The quantity Ω_{ij} is the relative velocity screw of one limb relative to the other and **the restriction of its commutator with X being zero means that the shared geometry of the joint must be invariant to the effect of the relative velocity**.

We can represent many types of joint as shared geometry joints, or combinations of shared geometry joints. In practice however we often want to represent slightly more complex joints compactly. In many commonly used GA frameworks we can represent more advanced types of joint with some form of bilinear mapping between two relevant objects, one in each frame, that we know remains constant for the joint geometry. More formally, for an object X in frame i and an object Y in frame j:

$$B(X,Y) = H \tag{7.17}$$

where H is a multivector that is constant for the joint, and in many cases is simply 0. Taking time derivatives once again gives us a linear constraint on Ω_i and Ω_j:

$$B(\dot{X},Y) + B(X,\dot{Y}) = 0, \tag{7.18}$$

$$B((X \times \Omega_i),Y) + B(X,(Y \times \Omega_j)) = 0. \tag{7.19}$$

So far at no point have we specified what objects X and Y are, nor the form of the bilinear mapping B (7.17). In practice, common forms of B include the outer product:

$$X \wedge Y = 0, \quad (X \times \Omega_i) \wedge Y + X \wedge (Y \times \Omega_j) = 0$$

the inner product:

$$X \cdot Y = 0, \quad (X \times \Omega_i) \cdot Y + X \cdot (Y \times \Omega_j) = 0$$

and the meet taken with respect to a specific fixed subspace:

$$X \vee Y = 0, \quad (X \times \Omega_i) \vee Y + X \vee (Y \times \Omega_j) = 0.$$

7.5 The Geometry of Real Joints

It is all well and good to state some neat theoretical results but it is more useful to robotics practitioners to outline how to describe actual joints in our framework. A good place to start is with the most common types of kinematic pair.

7.5.1 Spherical Joint

A spherical joint, often known as a ball joint, is one in which two bodies are free to move about a single common fixed point. Often in practice these joints are implemented with one body designed to have a spherical cavity (known as the socket) and another with a ball on the

end which is held captive within the socket. To implement a spherical joint within our shared geometry constraint we can use a sphere shared between both bodies in the kinematic pair. The sphere can be of any radius and in many practical applications (and in some GAs which do not naturally embed non-zero radius spheres as objects, such as PGA) it makes sense to use a sphere of zero radius, or, in other words, a point.

7.5.2 Cylindrical Joint

A cylindrical joint is a mechanism that allows two bodies to translate parallel to a shared axis and rotate about the same axis simultaneously. In practice this joint appears often as a cylindrical cuff free to slip over and around a rod. To implement the cylindrical joint within our framework we can use the geometry of a shared line. A line is invariant to rotation about its axis and translation along its axis. In CGA we can use a line either in its trivector form as the direct wedge product of two conformal points and n_∞, $A \wedge B \wedge n_\infty$ or in its bivector form $\hat{m}I_3 - (p \wedge \hat{m})I_3 n_\infty$ which is equivalent to the PGA line formulation $\hat{m}I_3 - (p \wedge \hat{m})I_3 e_0$ where $\hat{m}$ is the direction of the line and p is a point on the line.

7.5.3 Planar Joint

A planar joint is one that allows two bodies to slide over one another in a specific plane of motion and rotate about the normal to that plane. An example of this in the real world would be a wheeled trolley that is free to move in any direction on a flat surface. To implement the planar joint within our framework we can use a shared plane between the kinematic pairs. Again in CGA this can be done with either a 4-vector plane, $A \wedge B \wedge C \wedge n_\infty$, or with a dual 1-vector plane $m + dn_\infty$ which is again the same as the PGA plane $m + de_0$ where m is the normal to the plane and d is the distance of the plane from the origin.

7.5.4 Revolute Joint

A revolute joint, also known as a pin joint or hinge joint, is one that allows rotation about an axis but, unlike the cylindrical joint, does not allow translation parallel to the axis. The revolute joint is one of the most common mechanical joints in use in the wild and so having a convenient mechanism to handle it is important. To implement a revolute joint within our framework with CGA we can use a shared circle. A circle is invariant only to rotation about the axis passing through its centre and normal to the plane in which it lies. The radius of this circle is again arbitrary and choices of zero radius, or radius appropriate to the mechanical proportions of the robot at hand, are likely reasonable. In CGA this circle comes in the form

of a trivector $A \wedge B \wedge C$ or in the form of the bivector dual to that. In CGA a point-pair $P_1 \wedge P_2$ that is aligned with the axis can also be used.

7.5.5 Prismatic Joint

A prismatic joint is one in which two bodies are constrained to move relative to one another parallel to a fixed line without any rotation. The prismatic joint is the first of the classic robotic joints that we cannot represent with a single geometric primitive in 3D conformal GA. Instead we are forced to use some form of compound constraint. There are several options here, two parallel line constraints, a line and a plane constraint, a line and a direction bivector constraint and two planar constraints are all constructs that would work.

7.5.6 Universal Joint

A universal joint is a mechanism which consists of two orthogonal revolute joints with axes that are incident at a fixed point. It is often used to transmit rotational motion between two shafts whose axes are not parallel but are incident.

We can represent a universal joint in our framework as a joint consisting of two equal radius circles. The planes of these circles must lie orthogonal to each other and both circles must have the same centre point. If we choose two circles, X and Y, the orthogonality constraint can be written as $X \cdot Y = \langle XY \rangle_0 = 0$ and the same centre point constraint for circles of the same radius can be written $\langle XY \rangle_4 = 0$ [48]. We can combine these constraints neatly with the anti-commutator product

$$X \bar{\times} Y = \frac{1}{2}(XY + YX).$$

When operating on two circles, the anti-commutator product produces grade 0 and grade 4 elements only. Using this notation and taking derivatives our kinematic constraint can therefore be described as:

$$(X \times \Omega_i) \bar{\times} Y + X \bar{\times} (Y \times \Omega_j) = 0. \tag{7.20}$$

Of course we could have chosen to model this joint with two revolute joints and an additional body to represent the internals of the joint but this would introduce additional and unnecessary variables into our problem definition.

7.6 The Kinematic Constraint Matrix and the Jacobian Matrix

So far we have only looked at individual kinematic pairs, we will now address full multi-body systems. For an articulated robot with N limbs we write the velocity state as:

$$T = \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \dots \\ \Omega_N \end{bmatrix} \tag{7.21}$$

For a system with M constraints, there exists an $M \times N$ matrix C with linear functions as elements that embodies the combination of all the linear constraints on the velocity state. C acts on the velocity state giving a result of M zero valued multivectors:

$$0 = CT. \tag{7.22}$$

We will name C the kinematic constraint matrix and readers familiar with the Screw Theory literature will recognise it as a cousin of Davies' method [21]. We can choose some ordered L dimensional basis in which to represent the multivectors Ω_i in which case T becomes an $NL \times 1$ dimensional vector, C becomes an $ML \times NL$ dimensional matrix and the zero vector is $ML \times 1$ dimensional. To find the various coefficients of C we can simply set each of the coefficients of T to 1 in turn and all the rest to zero and calculate the output of each of our constraints for that input, calculating its output representation in the chosen multivector basis.

Consider a robot with velocity state T, now specify some of the limbs as inputs, with known Ω and some of the limbs as outputs with unknown Ω. We can write this as follows:

$$T = T_k + T_u \tag{7.23}$$

where the subscripts k and u refer to known and unknown Ω respectively. By linearity, applying the constraint matrix produces:

$$0 = CT = CT_k + CT_u,$$

$$CT_u = -CT_k. \tag{7.24}$$

This is in the standard form $Ax = b$ and can be solved with the pseudo-inverse in our chosen basis to produce a valid set of unknown limb velocities given the set of known velocities.

Designating a linear map K from the input degrees of freedom, x, to the known velocities of the system, and a linear map U from the unknown velocities to the output degrees of freedom, y, allows us to calculate a matrix that is equivalent to the Jacobian matrix of the system:

$$T_k = K(x), \quad U(T_u) = y,$$
$$y = (-UC^{+}CK)x \tag{7.25}$$

where C^{+} is the pseudo-inverse of C.

The matrix $C^{+}C$ describes the relationship between the known and unknown twists and, so long as we can form the various components of equation (7.25) we should be able to use this Screw Theory inspired framework to calculate the Jacobian matrix for any robot whose joints can be modelled using the bilinear mappings of Section 7.5.

7.7 Case Study: The Delta Robot

The Delta robot [17, 16] was invented in 1985 by Raymond Clavel at EPFL after being inspired by a visit to a chocolate packing factory [82]. It has since become a particularly popular robot in industrial settings due to its good precision coupled with high speed and acceleration.

The Delta robot is a specific type of robot known as a **parallel manipulator**. Parallel manipulators, also known as parallel robots, are a class of robots that feature end-effectors driven by multiple underactuated parallel kinematic chains [40, 78]. Typically a parallel robot is designed such that all actuators remain fixed to the support structure of the robot thereby minimising the mass of the moving parts of the robot and enabling very fast accelerations. Indeed this goal of high speed/fast acceleration has been the primary driving force in the development of parallel robots for industry and today architectures such as the Delta robot are widespread in many high precision, high throughput manufacturing applications. Parallel robots, while practically very useful, are often significantly more difficult to analyse than their serial cousins due to the end-point position being a function of the configuration of multiple kinematic chains.

Here we will do a case study of the Delta robot, analysing it with Geometric Algebra, calculating Jacobians with our Screw Theory based framework, and then finally comparing our screw theory setup with a direct differentiation approach to get the Jacobians.

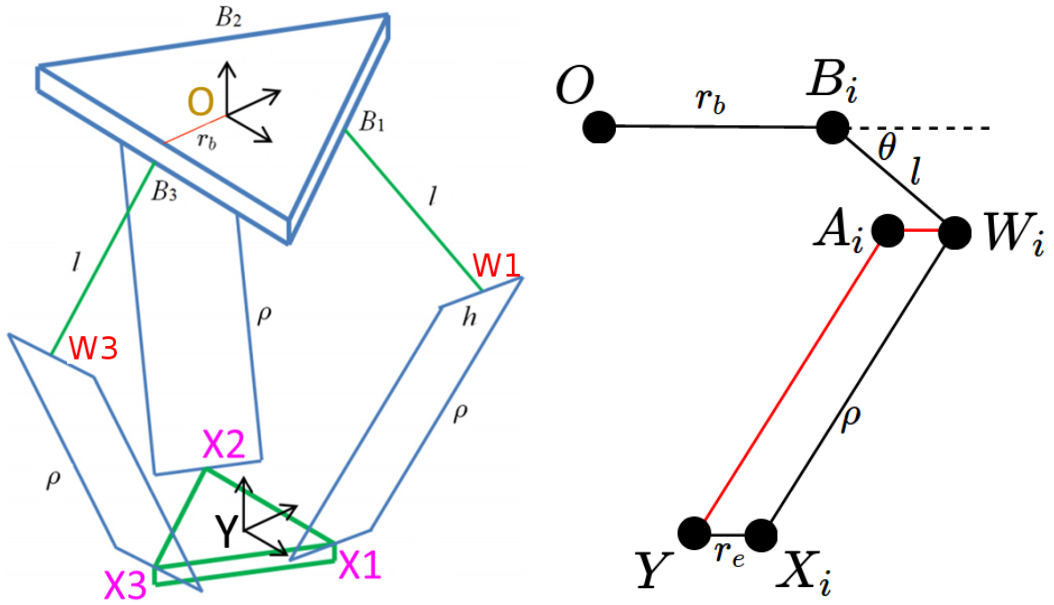


Fig. 7.2 Left: the 3D geometry of the delta robot. Right: The geometry of a single arm in plane.

7.7.1 Geometry of a Delta Robot

Since its inception, there have been many variants of the Delta robot [85]. In this chapter we will assume the simple robot described in this section and illustrated in Figure 7.2. The static part of the robot is a base plate to which three motors are rigidly attached, we will assume a space in which the origin is at the centre of this plate. Each motor shaft is rigidly attached to an 'upper arm' of length l; we will number each upper arm $i \in [1,2,3]$. The connection point of the motor and upper arm will be labelled B_i. The arm can only rotate in plane about the motor axis as the motor shaft and upper arm are rigidly connected. We will refer to the other end of this upper arm as the 'elbow point' and will label it W_i. At the elbow point each arm is rigidly attached to a central point of a horizontal rod we will refer to as the 'elbow rod'. At each end of the elbow rod a ball joint connects to a 'forearm' piece. The two forearm pieces for each arm are the same length and, at the other end from the elbow rod, are connected to a rigid plate that we will refer to as the end-effector plate. The point half-way between where the two forearm rods connect to the end-effector plate is labelled X_i. We will label the point at the centre of the end-effector plate Y. Assuming the robot is infinitely stiff, the end plate is constrained, due to this specific arrangement of the forearms, to always remain parallel to the base plate and to have its in-plane orientation fixed as well. The Delta robot is therefore a purely translational mechanism.

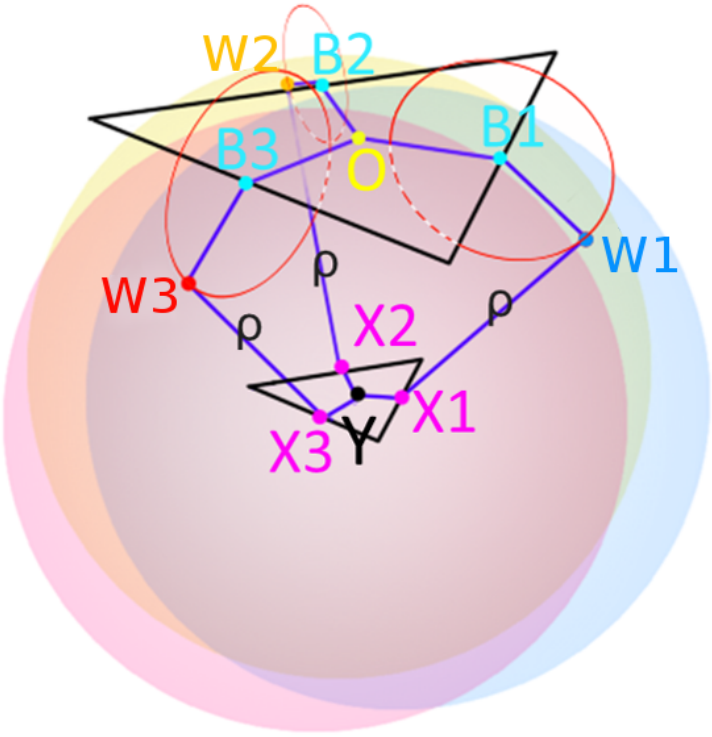


Fig. 7.3 The geometry of the inverse kinematic problem. There are three spheres, one for each arm of the robot, centred at the conformal points X_i. Each sphere intersects with a circle centred at B_i allowing the extraction of the conformal elbow point W_i.

7.7.2 Calculating the Robot Pose

The first task in analysing the Delta robot is to calculate its pose for a given set of inputs. Typically we are attempting to map from the end-effector to the actuators (inverse kinematics) or from the actuators to the end-effector (forward kinematics). Along the way we are interested in the positions of the limbs and joints that define the physical structure of the robot. To solve these pose calculation problems for the Delta robot initially we will use a simplified geometry for each leg.

Inverse Kinematics

The inverse kinematic problem for the Delta robot is summarised as follows: To what angle relative to the base should we move the upper arms given we want the centre of the end-effector plate to be in a specific position in 3D space?

To solve this problem we need to work backwards from the 3D end-effector plate position y to the motor angles θ_i considering the geometry of the robot as we go. Starting at the end-effector plate the 3D points x_i are translationally offset in plane in the direction s_i giving

$x_i = y + r_e s_i$ where r_e is the radius of the end-effector plate. Due to the geometry of the robot the elbow point W_i is constrained to lie on a sphere with radius equal to the length of the forearms ρ centred at this point x_i. We can write the dual form of this sphere as:

$$\Sigma_i^* = X_i - \frac{1}{2}\rho^2 n_\infty.$$

The elbow point is also simultaneously constrained to lie on a circle of radius l centred at the motor shaft to upper-arm joint, B_i. We can represent this circle in its dual form C_i^* in CGA, where C_i is the intersection of a sphere of radius l centred at the position B_i, with dual form $\left(B_i - \frac{1}{2}l^2 n_\infty\right)$, and the plane through the origin, B_i and e_3 which has dual form $I_3(s_i \wedge e_3)$. Here e_3 is the vertical unit vector and for the most common orientation of a delta robot, points vertically downward from the base plate. In CGA we calculate the intersection of objects via the 'meet' operator, as both operands are in their dual form however, here we simply need an outer product:

$$C_i^* = \left(B_i - \frac{1}{2}l^2 n_\infty\right) \wedge (I_3(s_i \wedge e_3)).$$

So long as y is within the reachable volume of the robot there are two possible solutions for this pair of constraints. These two solutions lie at the intersection points of the sphere and circle and the 'meet' operation of CGA provides us with a direct means to calculate these intersection points. The sphere and circle are in the dual form (1 and 2-vectors respectively), and so the point-pair bivector resulting from their meet is calculated as simply their outer product followed by multiplication with the 5D pseudo-scalar, I_5:

$$T_i = (C_i^* \wedge \Sigma_i^*) I_5.$$

The desired individual solution can be extracted from this point-pair object by relying on the oriented nature of the algebra and relying on a projection operator [76]:

$$P_i = \frac{1}{2}\left(1 + \frac{T_i}{\sqrt{T_i^2}}\right),$$
$$W_i = -\tilde{P}_i(T_i \cdot n_\infty).$$

We can then convert from the CGA to the 3D vector point:

$$w_i = \text{down}\,(W_i)$$

and so, with a little trigonometry we can extract the motor angles:

$$\theta_i = \text{atan2}(z_i \cdot e_3, z_i \cdot s_i), \quad z_i = w_i - r_b s_i$$

where r_b is the radius of the base-plate. Figure 7.3 illustrates the geometry of the inverse kinematic problem graphically. As mentioned we have relied on the oriented nature of the algebra to extract the solution of interest from the point-pair. This solution has the elbow position being as far from the e_3 axis as possible and is normally the only feasible position of the elbow in a real Delta robot as the other typically causes self intersection. If the other solution is desired the same projection formulae can be used, simply substituting $\tilde{T}_i$ for T_i.

Forward Kinematics

The forward kinematic problem is, in some sense, the opposite of the inverse kinematic one. Our goal here is to calculate the 3D vector position of the end-effector plate y given the motor angles $\theta_i, i \in [1,2,3]$.

To solve the forward kinematic problem we will consider the robot one arm at a time. For a given arm motor angle θ_i the 3D position of the elbow point w_i can be calculated as:

$$w_i = (r_b + l\cos(\theta_i))s_i + l\sin(\theta_i)e_3.$$

For each arm we will now define a pseudo-elbow point, a_i which is offset horizontally from the true elbow point by the radius of the end effector plate and in the direction of the origin.

$$a_i = (r_b - r_e + l\cos(\theta_i))s_i + l\sin(\theta_i)e_3.$$

The equivalent CGA point is then:

$$A_i = \frac{1}{2}a_i^2 n_\infty + a_i + n_0.$$

Given the geometry of the robot, these pseudo-elbow points all lie a distance equal to the length of the robot's forearms, ρ, from the centre of the end-point plate Y. Geometrically these fixed distance constraints manifest themselves as spheres, which we will label Σ_i, on which the centre of the end-point plate can lie:

$$\Sigma_i^* = A_i - \frac{1}{2}\rho^2 n_\infty.$$

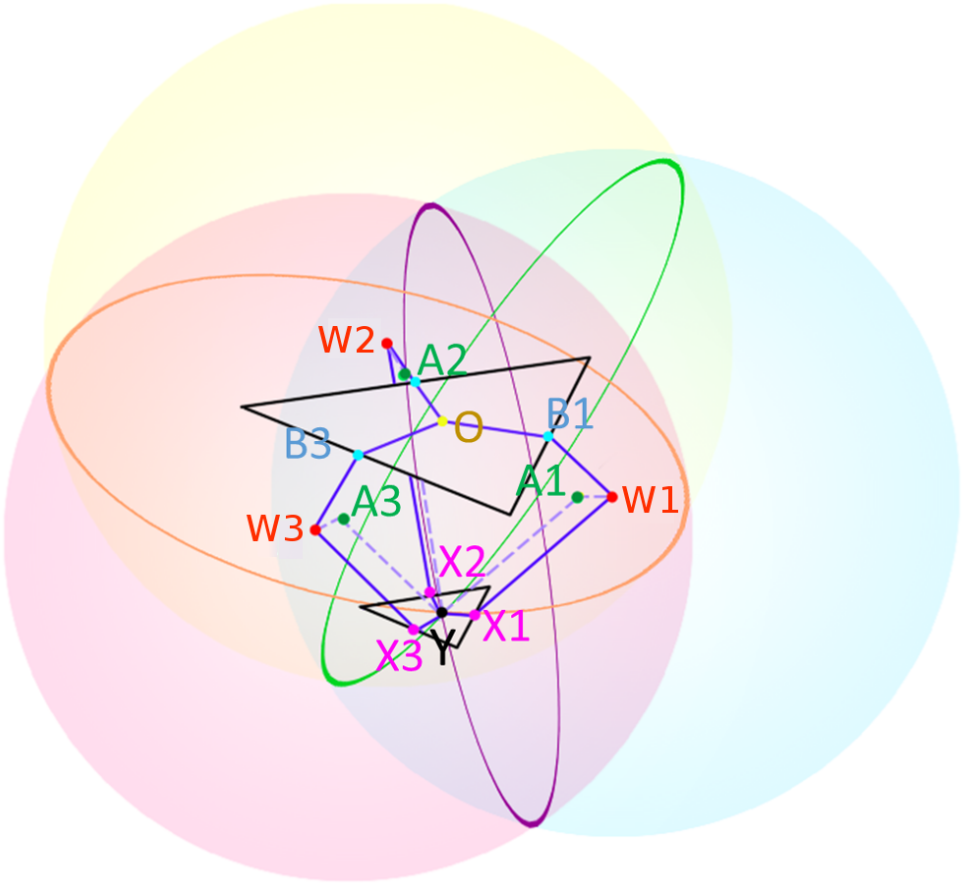


Fig. 7.4 The geometry of the forward kinematic problem. Each motor connects to an upper arm at position B_i. The upper arms end in the elbow point W_i. Each elbow point has an associated pseudo-elbow point A_i and forearm constraint sphere. All three constraint spheres meet at the centre of the end-point plate, Y.

Each arm contributes one constraint sphere and the intersection of the three spheres produces a point-pair, T, that represents the two possible configurations of the end-plate:

$$T = I_5 \bigwedge_{i=1}^{i=3} \Sigma_i^*$$

where the $\bigwedge$ notation implies an outer product of all elements following it.

Practically only one of these possible solutions is feasible, the solution which places Y at a greater position along the e_3 axis. As we have been careful throughout our equations to ensure our signs are correct we can exploit the oriented nature of CGA to extract the 3D

position of this point, y with the use of a single projector:

$$P = \frac{1}{2}\left(1 + \frac{T}{\sqrt{T^2}}\right),$$
$$y = \text{down}\, Y, \quad Y = -\tilde{P}(T \cdot n_\infty).$$

Figure 7.4 illustrates the geometry of the forward kinematic problem.

7.7.3 Full Geometry and Kinematic Constraint Matrix of the Delta Robot

Knowing static kinematic solutions is useful but to do more advanced analysis of the Delta robot mechanism we need to look at velocities. Given we have, up to this point, simplified the pose of the robot in both the forward and inverse cases, we will turn our attention to analysis of the full geometry and possible movements of the limbs. First we will do our analysis with our Screw Theory based framework and then we will compare it with a direct differentiation approach.

Looking at the Delta robot there are two types of joint present. The motor shaft to motor body connection is a revolute joint, the arm to arm connections are spherical joints, and the arm to end-effector platform are also spherical joints. The revolute joints of each motor connection can neatly be represented by a piece of shared geometry, namely a circle with normal parallel to the motor axis. Any radius of circle could be chosen, however as the limbs attached to the motor have a length of l it makes sense for us to use a radius of l for our circle. We will label these circles P_1, P_4, P_7 and we can construct them identically to those in section 7.7.2:

$$P_1 = C_1^* = \left(B_1 - \frac{1}{2}l^2 n_\infty\right) \wedge (I_3(s_1 \wedge e_3)),$$
$$P_4 = C_2^* = \left(B_2 - \frac{1}{2}l^2 n_\infty\right) \wedge (I_3(s_2 \wedge e_3)),$$
$$P_7 = C_3^* = \left(B_3 - \frac{1}{2}l^2 n_\infty\right) \wedge (I_3(s_3 \wedge e_3)).$$

The remaining joints are all spherical/ball joints and can therefore be conveniently represented by shared spheres. Again, the radius of these spheres is irrelevant. In this case we will choose a radius of zero, which makes the shared spheres into shared points. We will label each of the limbs of the Delta robot according to the diagram in Figure 7.6. According to this labelling

scheme we then have the following mapping from simplified to full robot geometry:

$$\begin{aligned}
P_2 &= \mathrm{up}\left(\mathrm{down}\, W_1 + r_t a\right), \\
P_3 &= \mathrm{up}\left(\mathrm{down}\, W_1 - r_t a\right), \\
P_5 &= \mathrm{up}\left(\mathrm{down}\, W_2 + r_t b\right), \\
P_6 &= \mathrm{up}\left(\mathrm{down}\, W_2 - r_t b\right), \\
P_8 &= \mathrm{up}\left(\mathrm{down}\, W_3 + r_t c\right), \\
P_9 &= \mathrm{up}\left(\mathrm{down}\, W_3 - r_t c\right), \\
P_{10} &= \mathrm{up}\left(\mathrm{down}\, X_1 + r_t a\right), \\
P_{11} &= \mathrm{up}\left(\mathrm{down}\, X_1 - r_t a\right), \\
P_{12} &= \mathrm{up}\left(\mathrm{down}\, X_2 - r_t b\right)
\end{aligned}$$

where:

$$r_t = \frac{r_e}{\tan(\pi/6)},$$
$$a = -(s_1 \wedge e_3)I_3, \quad b = -(s_2 \wedge e_3)I_3, \quad c = -(s_3 \wedge e_3)I_3.$$

due to the geometry of the end plate as shown in Figure 7.5.

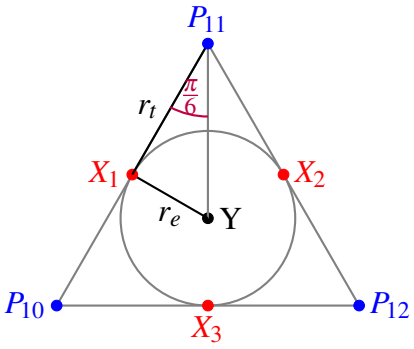


Fig. 7.5 The geometry of the end plate allows us to convert from the simplified end plate geometry X_1, X_2, X_3 to the full end plate geometry P_{10}, P_{11}, P_{12}.

For the motor shaft connections the circles are shared between the fixed world and the arm pieces. As it is fixed, the velocity screw of the world Ω_W is 0. Considering equation 7.16 we can substitute in our circles X_i, for X and 0 for Ω_W:

$$0 = X_i \times (\Omega_W - \Omega_i) = -X_i \times \Omega_i. \tag{7.26}$$

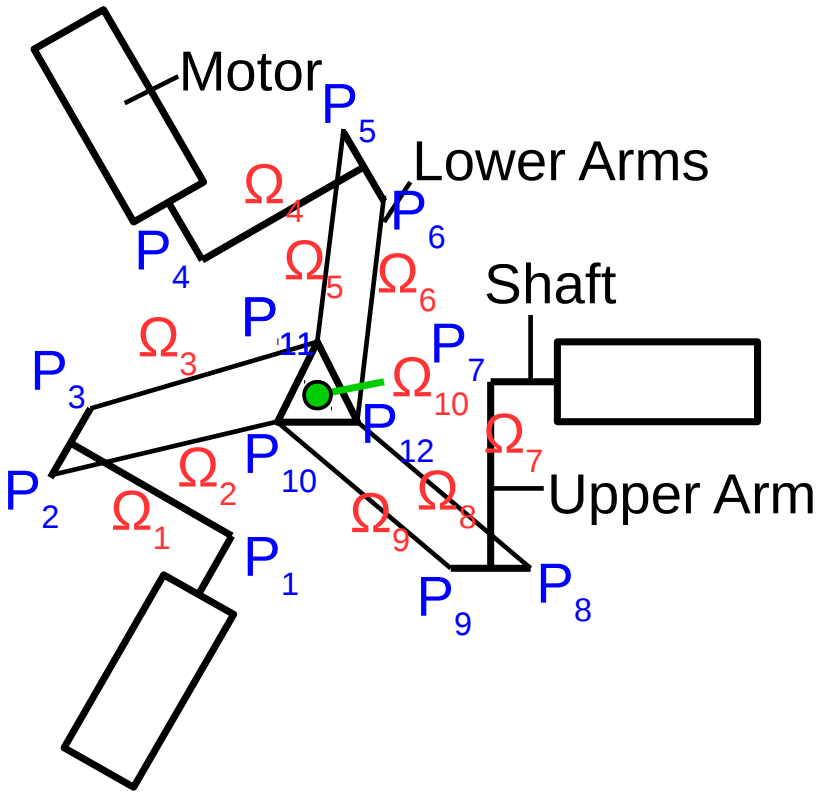


Fig. 7.6 A diagrammatic representation of the Delta robot and the relevant objects and screw quantities.

Putting this together with the shared point ball joints the full set of kinematic constraints for the Delta robot are as follows:

$$
\begin{aligned}
0 &= P_1 \times \Omega_1, \\
0 &= P_4 \times \Omega_4, \\
0 &= P_7 \times \Omega_7, \\
0 &= P_2 \times (\Omega_2 - \Omega_1), \\
0 &= P_3 \times (\Omega_3 - \Omega_1), \\
0 &= P_5 \times (\Omega_5 - \Omega_4), \\
0 &= P_6 \times (\Omega_6 - \Omega_4), \\
0 &= P_8 \times (\Omega_8 - \Omega_7), \\
0 &= P_9 \times (\Omega_9 - \Omega_7), \\
0 &= P_{10} \times (\Omega_{10} - \Omega_2), \\
0 &= P_{11} \times (\Omega_{10} - \Omega_3), \\
0 &= P_{11} \times (\Omega_{10} - \Omega_5), \\
0 &= P_{12} \times (\Omega_{10} - \Omega_6), \\
0 &= P_{12} \times (\Omega_{10} - \Omega_8), \\
0 &= P_{10} \times (\Omega_{10} - \Omega_9).
\end{aligned}
$$

The forward kinematic problem has known input velocities $\Omega_1, \Omega_4, \Omega_7$ and output velocity Ω_{10}. The inverse kinematic problem has known input velocity Ω_{10} and output velocities $\Omega_1, \Omega_4, \Omega_7$.

7.7.4 From Constraint Matrix to Jacobian Matrices

In order to convert from a constraint matrix to a Jacobian matrix we will need to define the linear maps mentioned in Section 7.6, K that maps from the input degrees of freedom to the **known** velocities, and U that maps from the **unknown** velocities to the output degrees of freedom. We will start by defining three normalised lines $\hat{L}_1, \hat{L}_4, \hat{L}_7$ whose dual forms are directly proportional to the twists $\Omega_1, \Omega_4, \Omega_7$. These lines are proportional to the twists as they are the axes about which pivot the limbs attached to the motors. We can form these lines from the dual circles P_1, P_4, P_7:

$$
L_1 = P_1 \wedge n_\infty, \quad L_4 = P_4 \wedge n_\infty, \quad L_7 = P_7 \wedge n_\infty,
$$

$$\hat{L}_1^* = \frac{L_1 I_5}{\sqrt{-L_1\tilde{L}_1}}, \quad \hat{L}_4^* = \frac{L_4 I_5}{\sqrt{-L_4\tilde{L}_4}}, \quad \hat{L}_7^* = \frac{L_7 I_5}{\sqrt{-L_7\tilde{L}_7}}.$$

With these dual lines we can map between the motor angular speeds $\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3$ and the twists $\Omega_1, \Omega_4, \Omega_7$.

$$\Omega_1 = \dot{\theta}_1 \hat{L}_1^*, \quad \Omega_4 = \dot{\theta}_2 \hat{L}_4^*, \quad \Omega_7 = \dot{\theta}_3 \hat{L}_7^*.$$

For the forward kinematic problem we therefore can construct a map K_F as follows:

$$\begin{bmatrix} \Omega_1 \\ \dots \\ \Omega_4 \\ \dots \\ \Omega_7 \\ \dots \end{bmatrix} = \begin{bmatrix} L_1^* & 0 & 0 \\ \dots & \dots & \dots \\ 0 & L_4^* & 0 \\ \dots & \dots & \dots \\ 0 & 0 & L_7^* \\ \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix}$$

and as $(L_1^*)^2 = -1$ for the inverse kinematic case we can construct U_I from negative the transpose of K_F, $U_I = -K_F^T$.

At the end-effector in the forward kinematic problem we need to get the translational effect of Ω_{10} on the central point of the end-plate, we therefore need to calculate $\dot{y}$. Given we have already calculated Y, getting $\dot{y}$ is a relatively straightforward task. First, we form the line $Y \wedge \dot{Y} \wedge n_\infty$ which has the orientation and magnitude of $\dot{y}$, then we extract the 3D direction and magnitude from this line. To extract the 3D element we can use the dual form of the line and extract the euclidean bivector components which when multiplied with the negative 3DGA pseudoscalar gives the direction of the line. Or, symbolically:

$$L^* = (Y \wedge (\Omega_{10} \times Y) \wedge n_\infty) I_5,$$

$$\dot{y} = -(L^* \wedge E_0) E_0 I_3.$$

This is a general method to extract the linear velocity of a point from the CGA point and its velocity screw. In the case of the Delta robot of course we know that the end-plate is constrained to move only translationally and so Ω_{10} will be a purely translational bivector of the form $-\dot{y} \wedge n_\infty$. We can therefore extract $\dot{y}$ directly from Ω_{10}:

$$\dot{y} = n_0 \cdot \Omega_{10}.$$

To turn $\dot{y}$ into individual components we can simply dot it with each basis vector in turn $\dot{y}_{e_1} = \dot{y} \cdot e_1$ etc.

With this in mind we can now construct the known input map for the inverse problem K_I:

$$\begin{bmatrix} \dots \\ \Omega_{10} \\ \dots \end{bmatrix} = \begin{bmatrix} \dots & \dots & \dots \\ -e_1 \wedge n_\infty & -e_2 \wedge n_\infty & -e_3 \wedge n_\infty \\ \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \dot{y}_{e_1} \\ \dot{y}_{e_2} \\ \dot{y}_{e_3} \end{bmatrix}$$

and the unknown output map for the forward problem U_f:

$$\begin{bmatrix} \dot{y}_{e_1} \\ \dot{y}_{e_2} \\ \dot{y}_{e_3} \end{bmatrix} = \begin{bmatrix} \dots & e_1 \cdot (n_0 \cdot [\,]) & \dots \\ \dots & e_2 \cdot (n_0 \cdot [\,]) & \dots \\ \dots & e_3 \cdot (n_0 \cdot [\,]) & \dots \end{bmatrix} \begin{bmatrix} \dots \\ \Omega_{10} \\ \dots \end{bmatrix}$$

7.7.5 Calculating the Jacobian with Direct Differentiation

The Inverse Jacobian

For our direct differentiation method we will start with our simplified inverse kinematic solution and simply differentiate the expressions directly.

First we will write the 3D end-point plate position as a linear combination of basis vectors with coefficients denoted $\alpha_j,\ j \in 1,2,3$:

$$y = \alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3.$$

Our goal is to calculate the partial derivative of each motor angle with respect to each of these α coefficients. Taking partial derivatives of y with respect to one of the α_j coefficients trivially gives:

$$\frac{\partial y}{\partial \alpha_j} = e_j.$$

For now we do not need to worry about **which** α parameter we are taking derivatives with respect to, so we will leave the derivative of the end-point written as $\frac{\partial y}{\partial \alpha}$. Using this notation, our ultimate goal in this section is to find an equation for the partial derivative of a given motor angle θ_i with respect to α, $\frac{\partial \theta_i}{\partial \alpha}$. To find $\frac{\partial \theta_i}{\partial \alpha}$ we select a specific robot arm i and work back through its joints from the end-point.

The first joint position of interest is x_i, we saw in section 7.7.2 that:

$$x_i = y + r_e s_i.$$

Taking partial derivatives gives:

$$\frac{\partial x_i}{\partial \alpha} = \frac{\partial y}{\partial \alpha}.$$

The 3D point x_i can then be represented as the CGA point X_i:

$$X_i = \frac{1}{2} x_i^2 n_\infty + x_i + n_0.$$

The derivative of this CGA point is then easily found:

$$\frac{\partial X_i}{\partial \alpha} = \left(\frac{\partial x_i}{\partial \alpha} \cdot x_i \right) n_\infty + \frac{\partial x_i}{\partial \alpha}.$$

We then form the dual constraint sphere:

$$\Sigma_i^* = X_i - \frac{1}{2} \rho^2 n_\infty$$

which, as the radius is fixed, has partial derivative:

$$\frac{\partial \Sigma_i^*}{\partial \alpha} = \frac{\partial X_i}{\partial \alpha}.$$

As we saw in the previous section, the intersection of this dual constraint sphere Σ_i^* and the dual circle C_i^* centred on the motor shaft produces a point-pair T_i that represents the two possible elbow positions for that arm:

$$\begin{aligned} C_i^* &= \left(B_i - \frac{1}{2} l^2 n_\infty \right) \wedge (I_3 (s_i \wedge e_3)), \\ T_i &= (\Sigma_i^* \wedge C_i^*)^*. \end{aligned}$$

The outer product and dual operations are both linear, which means that taking derivatives is particularly easy here:

$$\frac{\partial T_i}{\partial \alpha} = \left(\frac{\partial \Sigma_i^*}{\partial \alpha} \wedge C_i^* \right)^*.$$

Of course the elbow can only actually be in one position which we can extract via a projection operation:

$$P_i = \frac{1}{2}\left(1 + \frac{T_i}{\sqrt{T_i^2}}\right), \quad \frac{\partial P_i}{\partial \alpha} = \frac{1}{2T_i^2}\left(\sqrt{T_i^2}\frac{\partial T_i}{\partial \alpha} - T_i\frac{\frac{\partial T_i}{\partial \alpha}\cdot T_i}{\sqrt{T_i^2}}\right),$$
$$W_i = -\tilde{P}_i(T_i \cdot n_\infty),$$
$$\frac{\partial W_i}{\partial \alpha} = -\frac{\partial \tilde{P}_i}{\partial \alpha}(T_i \cdot n_\infty) - \tilde{P}_i\left(\frac{\partial T_i}{\partial \alpha}\cdot n_\infty\right).$$

We can then convert from the CGA to the 3D vector point:

$$w_i = \frac{-(W_i \wedge E_0)E_0}{W_i \cdot n_\infty},$$
$$\frac{\partial w_i}{\partial \alpha} = \frac{-(\frac{\partial W_i}{\partial \alpha} \wedge E_0)E_0(W_i \cdot n_\infty) + (W_i \wedge E_0)E_0(\frac{\partial W_i}{\partial \alpha}\cdot n_\infty)}{(W_i \cdot n_\infty)^2}$$

and use this to form the derivative of the motor angles with respect to α:

$$z_i = w_i - r_b s_i, \quad \frac{\partial z_i}{\partial \alpha} = \frac{\partial w_i}{\partial \alpha},$$
$$\theta_i = \text{atan2}(z_i \cdot e_3, z_i \cdot s_i),$$
$$\frac{\partial \theta_i}{\partial \alpha} = \frac{z_i \cdot s_i}{|z_i \cdot s_i|}\frac{(z_i \cdot s_i)\left(\frac{\partial z_i}{\partial \alpha}\cdot e_3\right) - (z_i \cdot e_3)\left(\frac{\partial z_i}{\partial \alpha}\cdot s_i\right)}{z_i^2}.$$

This finally gives us an expression for the derivative of the motor angle with respect to the α of the endpoint. Typically in engineering scenarios we would construct a matrix of the partial derivatives with respect to $\alpha_j,\ j \in 1,2,3$, known as the Jacobian matrix:

$$J^* = \begin{bmatrix} \frac{\partial \theta_1}{\partial \alpha_1} & \frac{\partial \theta_1}{\partial \alpha_2} & \frac{\partial \theta_1}{\partial \alpha_3} \\ \frac{\partial \theta_2}{\partial \alpha_1} & \frac{\partial \theta_2}{\partial \alpha_2} & \frac{\partial \theta_2}{\partial \alpha_3} \\ \frac{\partial \theta_3}{\partial \alpha_1} & \frac{\partial \theta_3}{\partial \alpha_2} & \frac{\partial \theta_3}{\partial \alpha_3} \end{bmatrix}$$

This matrix can then be used to convert an end-point velocity vector to a set of motor velocities:

$$\begin{bmatrix} \frac{\partial \theta_1}{\partial t} \\ \frac{\partial \theta_2}{\partial t} \\ \frac{\partial \theta_3}{\partial t} \end{bmatrix} = J^* \begin{bmatrix} \frac{\partial \alpha_1}{\partial t} \\ \frac{\partial \alpha_2}{\partial t} \\ \frac{\partial \alpha_3}{\partial t} \end{bmatrix}$$

As it is the Jacobian matrix for the inverse kinematic problem, this matrix is specifically labelled the inverse Jacobian matrix.

The Forward Jacobian

Many problems in robotics require us to take derivatives of the forward kinematic equations. Specifically, we need to know the end-point plate velocity as a function of the motor speeds.

Our forward kinematic solution begins with calculating the position of the elbow point for a given arm i:

$$w_i = (r_b + l\cos(\theta_i))s_i + l\sin(\theta_i)e_3, \qquad \frac{\partial w_i}{\partial \theta_i} = -l\sin(\theta_i)s_i + l\cos(\theta_i)e_3.$$

With the elbow point we can then calculate the pseudo-elbow point:

$$a_i = w_i - r_e s_i, \qquad \frac{\partial a_i}{\partial \theta_i} = \frac{\partial w_i}{\partial \theta_i}.$$

We then convert the pseudo-elbow to a CGA point:

$$A_i = \frac{1}{2}a_i^2 n_\infty + a_i + n_0, \qquad \frac{\partial A_i}{\partial \theta_i} = \left(\frac{\partial a_i}{\partial \theta_i}\cdot a_i\right) n_\infty + \frac{\partial a_i}{\partial \theta_i}.$$

The forearm length dual constraint sphere can then be constructed about the pseudo-elbow point

$$\Sigma_i^* = A_i - \frac{1}{2}\rho^2 n_\infty, \qquad \frac{\partial \Sigma_i^*}{\partial \theta_i} = \frac{\partial A_i}{\partial \theta_i}.$$

The intersection of all three constraint spheres, one from each arm, produces the point pair on which the solution lies:

$$T = (\Sigma_1 \vee \Sigma_2 \vee \Sigma_3) \equiv I_5(\Sigma_1^* \wedge \Sigma_2^* \wedge \Sigma_3^*).$$

We can take derivatives of this point-pair with respect to each of the motor angles:

$$\frac{\partial T}{\partial \theta_1} = I_5\left(\frac{\partial \Sigma_1^*}{\partial \theta_1} \wedge \Sigma_2^* \wedge \Sigma_3^*\right), \qquad \frac{\partial T}{\partial \theta_2} = I_5\left(\Sigma_1^* \wedge \frac{\partial \Sigma_2^*}{\partial \theta_2} \wedge \Sigma_3^*\right),$$
$$\frac{\partial T}{\partial \theta_3} = I_5\left(\Sigma_1^* \wedge \Sigma_2^* \wedge \frac{\partial \Sigma_3^*}{\partial \theta_3}\right).$$

We can re-write these derivatives as follows:

$$\frac{\partial T}{\partial \theta_i} = (-1)^{i-1} I_5 \left(\frac{\partial \Sigma_i^*}{\partial \theta_i} \wedge C^* \right), \quad \text{where } C^* = \bigwedge_{j \in 1,2,3 \; j \neq i} \Sigma_j^*. \tag{7.27}$$

Practically, when we take partial derivatives with respect to one θ at a time we are effectively freezing two of the motors in position and moving the third. Geometrically, this process forces the end-point plate to move along a circle formed by the intersection of the two constraint spheres centred at the pseudo-elbow points of the frozen motors.

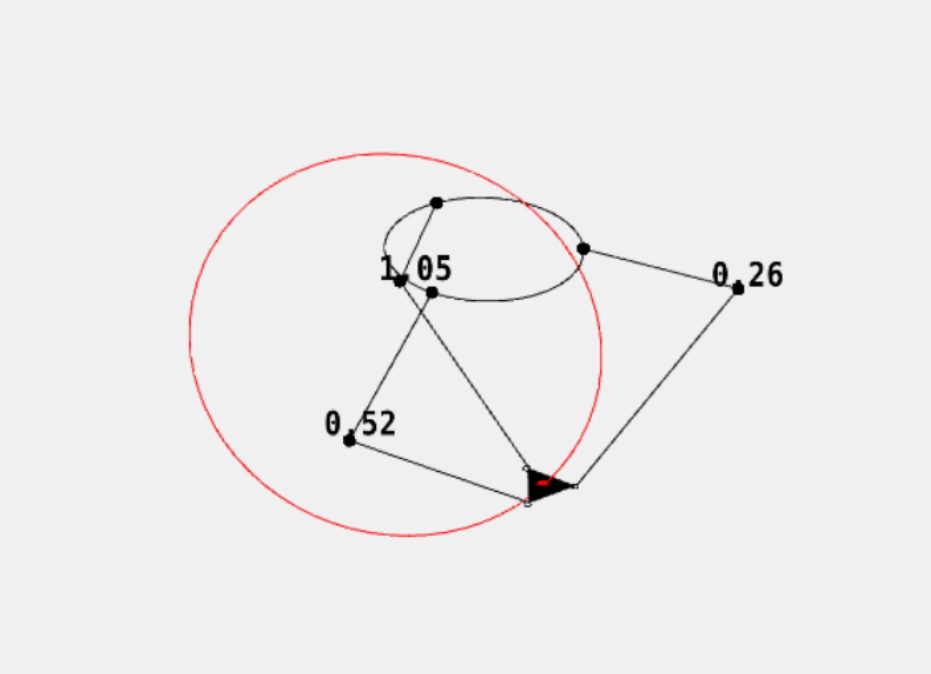


Fig. 7.7 With two limbs frozen the end-point plate is constrained to move such that its centre always lies on the circle (shown in red) formed from the intersection of the other two limbs' constraint spheres. The numbers displayed on the figure are the motor angles in radians.

Figure 7.7 shows the geometric significance of Equation 7.27. To get the end-point plate position we again extract one end of the point-pair T:

$$P = \frac{1}{2}\left(1 + \frac{T}{\sqrt{T^2}}\right), \qquad \frac{\partial P}{\partial \theta_i} = \frac{1}{2T^2}\left(\sqrt{T^2}\frac{\partial T}{\partial \theta_i} - T\frac{\frac{\partial T}{\partial \theta_i} \cdot T}{\sqrt{T^2}}\right),$$

$$Y = -\tilde{P}(T \cdot n_\infty), \quad \frac{\partial Y}{\partial \theta_i} = -\frac{\partial \tilde{P}}{\partial \theta_i}(T \cdot n_\infty) - \tilde{P}\left(\frac{\partial T}{\partial \theta_i} \cdot n_\infty\right).$$

Finally we convert our end-point back to a 3D point:

$$y = \operatorname{down} Y,$$
$$\frac{\partial y}{\partial \theta_i} = \frac{-(\frac{\partial Y}{\partial \theta_i} \wedge E_0) E_0 (Y \cdot n_\infty) + (Y \wedge E_0) E_0 (\frac{\partial Y}{\partial \theta_i} \cdot n_\infty)}{(Y \cdot n_\infty)^2}.$$

We can write the end-point plate position as:

$$y = \alpha_1 e_1 + \alpha_2 e_2 + \alpha_3 e_3, \quad \frac{\partial y}{\partial \theta_i} = \frac{\partial \alpha_1}{\partial \theta_i} e_1 + \frac{\partial \alpha_2}{\partial \theta_i} e_2 + \frac{\partial \alpha_3}{\partial \theta_i} e_3.$$

With $\frac{\partial y}{\partial \theta_i}$ we are therefore in a position to build the forward Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial y}{\partial \theta_1} \cdot e_1 & \frac{\partial y}{\partial \theta_2} \cdot e_1 & \frac{\partial y}{\partial \theta_3} \cdot e_1 \\ \frac{\partial y}{\partial \theta_1} \cdot e_2 & \frac{\partial y}{\partial \theta_2} \cdot e_2 & \frac{\partial y}{\partial \theta_3} \cdot e_2 \\ \frac{\partial y}{\partial \theta_1} \cdot e_3 & \frac{\partial y}{\partial \theta_2} \cdot e_3 & \frac{\partial y}{\partial \theta_3} \cdot e_3 \end{bmatrix}$$

The inverse Jacobian matrix and the forward Jacobian matrix are, as the names suggest, inverse to each other.

$$JJ^* = I.$$

7.7.6 Comparing Direct Differentiation to Screw Theory

Both the direct differentiation method and our Screw Theory inspired method can be shown to give us numerically identical Jacobian matrices, however the screw theoretic approach is significantly easier for a practitioner to use as it does not require explicitly taking derivatives. In this example of the Delta robot we have the unusual luxury of a simple closed form solution for the pose of the robot in both the forward and inverse case and so the direct differentiation method is easy to compute either manually as we have done here or using automatic differentiation. With other robot architectures we may not have this blessing and would have to rely on non-linear optimisation or algebraic geometry methods to find a pose that satisfies the constraints. Table 7.1 shows a brief qualitative comparison of the two methods, as ever in practice the choice of which is best to use will be down to the problem itself and what tools and computational resources are available.

	Direct differentiation	Screw Theory based
Does not require explicit derivatives of pose calculation	✗	✓
Can be implemented directly with automatic differentiation	✓	✗
Encapsulates information about the entire system	✗	✓

Table 7.1 A qualitative comparison of the direct differentiation method vs the Screw Theory based technique for analysing the Jacobians of the system.

7.8 Conclusions and Future Work

In this chapter we have embedded screw theoretic concepts within Geometric Algebra and used this embedding to analyse kinematic pairs and full multi-body systems. The combination of Screw Theory and GA is a particularly potent mix for robotics, allowing clean expressions of geometric constraints and neat representations of kinematic limitations. There are many potential avenues for future work in this area, one promising route would be in the use of higher dimensional Clifford Algebras [11] to represent complex contact surfaces in joints, another might be to expand the allowable motions to include expansion and shear, allowing us to form an extended screw theory for soft robotic modelling. More immediately there is the issue of characterising the computational cost of these Screw Theory based methods. A comparison of the compute speed of these techniques with the direct differentiation method is beyond the scope of this chapter but as it is of practical importance it will likely be a focus of follow up work on this topic.

Chapter 8

Conclusions

8.1 Main Contributions

In this body of work we have made the following contributions:

- We link the problem of finding a valid rotor between two geometric primitives to the direct addition of these primitives
- We introduce a technique for extracting a blade from an arbitrary pure grade multivector
- We show how linear combinations of multivectors can be used to make tubular and ribbon surfaces and give formulae and techniques for intersections of these surfaces with lines and the normal to the surface at any point
- We make explicit the mapping between lines in CGA and PGA and the screws of Screw Theory
- We show how GA can provide Screw Theory with a coordinate free screw representation and show the applications of this in rigid body dynamics
- We show how the GA screw formulation allows us to extend Screw Theory multi-body kinematics to include geometric primitives directly as kinematic constraints within joints

8.2 Future Work

Much of this thesis has been focused on the theory and applications of the **addition** of GA blades representing geometric primitives. We saw in Part I how addition can be used in an

intuitive way in to create intermediary objects that blend between extremal control objects and how these intermediary objects are related to transformations that take one control object to another. In the case of CGA circles and point pairs, while the properties of these interpolant objects are still relatively poorly understood. There may be promising future work that could be done in this area, investigating links between the differential geometry of these surfaces and the form of the interpolant objects.

In the case of the addition of lines we have seen in Part II how linear combinations of these flat elements produces the screws of Screw Theory and the various grades of the geometric product map neatly onto its operators. GA provides a particularly elegant framework for Screw Theory and the unification of geometric primitives, screws motions and covariant products in one algebraic framework and holds great promise for concise descriptions of joint types for multi-body systems. The combination of Screw Theory and Geometric Algebra is underdeveloped, given the wide interest of GA practitioners in robotics.

References

[1] Aristidou, A. (2010). *Tracking and Modelling Motion for Biomechanical Analysis*. PhD thesis, University of Cambridge.

[2] Ball, R. S. (1900). *A treatise on the theory of screws*. Cambridge University Press Cambridge.

[3] Bauer, U. and Polthier, K. (2007). Parametric reconstruction of bent tube surfaces. In *2007 International Conference on Cyberworlds (CW'07)*, page 465–474. IEEE.

[4] Bayro-Corrochano, E. and Rivera-Rovelo, J. (2009). The use of geometric algebra for 3D modeling and registration of medical data. *Journal of Mathematical Imaging and Vision*, 34(1):48–60.

[5] Besl, P. J. and McKay, N. D. (1992). A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256.

[6] Bezier, P. (1986). *The Mathematical Basis of the UNISURF CAD System*. Butterworth-Heinemann, USA.

[7] Bortz, J. E. (1971). A new mathematical formulation for strapdown inertial navigation. *IEEE Transactions on Aerospace and Electronic Systems*, AES-7(1):61–66.

[8] Bosche, F. (2011). Plane-based coarse registration of 3D point clouds with 4d models. *28th International Symposium on Automation and Robotics in Construction*.

[9] Boyle, M. (2017). The integration of angular velocity. *Advances in Applied Clifford Algebras*, 27(3):2345–2374.

[10] Breuils, S., Nozick, V., and Fuchs, L. (2019a). Garamon: A geometric algebra library generator. *Advances in Applied Clifford Algebras*, 29(3).

[11] Breuils, S., Nozick, V., Fuchs, L., and Sugimoto, A. (2019b). Transverse approach to geometric algebra models for manipulating quadratic surfaces. In Gavrilova, M., Chang, J., Thalmann, N. M., Hitzer, E., and Ishikawa, H., editors, *Advances in Computer Graphics*, pages 523–534, Cham. Springer International Publishing.

[12] Breuils, S., Nozick, V., Sugimoto, A., and Hitzer, E. (2018). Quadric conformal geometric algebra of R(9,6). *Advances in Applied Clifford Algebras*, 28(2).

[13] Cameron, J. I. (2007). *Random Disconnected Applications of Geometric Algebra in Computer Graphics and Computer Vision*. PhD thesis, University of Cambridge.

[14] Candy, L. P. (2012). *Kinematics in conformal geometric algebra with applications in strapdown inertial navigation.* PhD thesis, University of Cambridge.

[15] Catmull, E. and Rom, R. (1974). A class of local interpolating splines. In *Computer Aided Geometric Design*, pages 317–326. Academic Press.

[16] Clavel, R. (1990). Device for the movement and positioning of an element in space. United States Patent and Trademark Office, US4976582A.

[17] Clavel, R. (1991). *Conception d'un robot parallèle rapide à 4 degrés de liberté*. PhD thesis, École polytechnique fédérale de Lausanne.

[18] Colapinto, P. (2011). *Spatial Computing with Conformal Geometric Algebra*. PhD thesis, University of California, Santa Barbara.

[19] Colapinto, P. (2017). Composing surfaces with conformal rotors. *Advances in Applied Clifford Algebras*, 27(1):453–474.

[20] Davidson, J. K. and Hunt, K. H. (2004). Robots and screw theory: Applications of kinematics and statics to robotics. *Journal of Mechanical Design*, 126(4):763–764.

[21] Davies, T. H. (1981). Kirchhoff's circulation law applied to multi-loop kinematic chains. *Mechanism and Machine Theory*, 16(3):171–183.

[22] De Keninck, S. (2019). Non-parametric realtime rendering of subspace objects in arbitrary geometric algebras. In Gavrilova, M., Chang, J., Thalmann, N. M., Hitzer, E., and Ishikawa, H., editors, *Advances in Computer Graphics*, Lecture Notes in Computer Science, page 549–555. Springer International Publishing.

[23] De Keninck, S. (2020). ganja.js. Zenodo. https://doi.org/10.5281/ZENODO.3635774.

[24] De Keninck, S. and Dorst, L. (2019). Geometric algebra levenberg-marquardt. *Advances in Computer Graphics. CGI 2019. Lecture Notes in Computer Science*, 11542:511–522.

[25] Deul, C., Burger, M., Hildenbrand, D., and Koch, A. (2009). Raytracing point clouds using geometric algebra. *GraVisMa proceedings*.

[26] Doran, C. (2003). Circle and sphere blending with conformal geometric algebra. *arXiv:cs/0310017*.

[27] Doran, C. and Lasenby, A. (2003). *Geometric Algebra for Physicists*. Cambridge University Press.

[28] Dorst, L. (2020). A guided tour to the plane-based geometric algebra PGA, version 1.15. Bivector.net.

[29] Dorst, L., Fontijne, D., and Mann, S. (2007). *Geometric algebra for computer science: an object-oriented approach to geometry*. Morgan Kaufmann series in computer graphics. Elsevier; Morgan Kaufmann.

[30] Dorst, L. and Valkenburg, R. (2011). Square root and logarithm of rotors in 3D conformal geometric algebra using polar decomposition. In *Guide to Geometric Algebra in Practice*, page 81–104. Springer, London.

[31] Du, J., Goldman, R., and Mann, S. (2017). Modeling 3D geometry in the clifford algebra R(4,4). *Advances in Applied Clifford Algebras*, 27(4):3039–3062.

[32] Duda, R., Hart, P., and Stork, D. (2001). *Pattern Classification*. Wiley Interscience, 2nd edition edition.

[33] Easter, R. B. and Hitzer, E. (2017). Double conformal geometric algebra. *Advances in Applied Clifford Algebras*, 27(3):2175–2199.

[34] Eggert, D., Lorusso, A., and Fisher, R. (1997). Estimating 3D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*, 9(5):272–290.

[35] Eide, E. R. and Lasenby, J. (2018). A novel way of estimating rotors between conformal objects and its applications in computer vision. *AACA: Topical Collection AGACSE 2018, IMECC – UNICAM, Campinas, Brazil*.

[36] Featherstone, R. (2008). *Rigid body dynamics algorithms*. Springer.

[37] Featherstone, R. (2010). A beginner's guide to 6D vectors (part 1). *IEEE Robotics & Automation Magazine*, 17(3):83–94.

[38] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

[39] Fu, Z., Yang, W., and Yang, Z. (2013). Solution of inverse kinematics for 6R robot manipulators with offset wrist based on geometric algebra. *Journal of Mechanisms and Robotics*, 5(3).

[40] Gallardo-Alvarado, J. (2016). *Kinematic Analysis of Parallel Manipulators by Algebraic Screw Theory*. Springer International Publishing.

[41] Gelfand, N., Mitra, N. J., Guibas, L. J., and Pottmann, H. (2005). Robust global registration. *Eurographics Symposiumon Geometry Processing*.

[42] Gunn, C. (2011a). On the homogeneous model of euclidean geometry. *arXiv:1101.4542 [math]*. arXiv: 1101.4542.

[43] Gunn, C. (2020). On-metric alternatives to reciprocal frames. the bivector.net forums. https://discourse.bivector.net/t/non-metric-alternative-to-reciprocal-frame/105/4. Accessed 12th May 2020.

[44] Gunn, C. G. (2011b). *Geometry, Kinematics, and Rigid Body Mechanics in Cayley-Klein Geometries*. PhD thesis, Technical University Berlin.

[45] Gunn, C. G. and De Keninck, S. (2019a). Geometric algebra and computer graphics. *ACM SIGGRAPH 2019 Courses (SIGGRAPH '19). Association for Computing Machinery, New York, NY, USA, Article 12, 1–140*.

[46] Gunn, C. G. and De Keninck, S. (2019b). Siggraph geometric algebra course video.

[47] Hadfield, H., Achawal, S., and Lasenby, J. (2021). Exploring novel surface representations via an experimental ray-tracer in CGA. *Advances in Applied Clifford Algebras*, 31(16).

[48] Hadfield, H. and Lasenby, J. (2019). Direct linear interpolation of geometric objects in conformal geometric algebra. *Advances in Applied Clifford Algebras*, 29(85).

[49] Hadfield, H., Lasenby, J., Ramage, M., and Doran, C. (2019). Reform: Rotor estimation from object resampling and matching. *Advances in Applied Clifford Algebras: Topical Collection AGACSE 2019, IMECC – UNICAM, Campinas, Brazil*.

[50] Hadfield, H., Wei, L., and Lasenby, J. (2020). The forward and inverse kinematics of a Delta robot. In Magnenat-Thalmann, N., Stephanidis, C., Wu, E., Thalmann, D., Sheng, B., Kim, J., Papagiannakis, G., and Gavrilova, M., editors, *Advances in Computer Graphics*, Lecture Notes in Computer Science, page 447–458. Springer International Publishing.

[51] Hadfield, H. and Wieser, E. (2020). Robots, ganja & screw theory. https://www.youtube.com/watch?v=bj9JslblYPU.

[52] Hadfield, H., Wieser, E., Arsenovic, A., Kern, R., and The Pygae Team (2018-Present). www.github.com/pygae/clifford.

[53] Hestenes, D. (2001). *Old Wine in New Bottles: A New Algebraic Framework for Computational Geometry*, pages 3–17. Birkhäuser Boston, Boston, MA.

[54] Hestenes, D. (2010). *New Tools for Computational Geometry and Rejuvenation of Screw Theory*, pages 3–33. Springer London, London.

[55] Hestenes, D. and Fasse, E. D. (2002). *Homogeneous Rigid Body Mechanics with Elastic Coupling*, pages 197–212. Birkhäuser Boston, Boston, MA.

[56] Hestenes, D., Sobczyk, G., and S. Marsh, J. (1985). Clifford algebra to geometric calculus. a unified language for mathematics and physics. *American Journal of Physics*, 53:510–511.

[57] Hildenbrand, D. (2007). *Geometric Computing in Computer Graphics and Robotics using Conformal Geometric Algebra*. PhD thesis, Technische Universität Darmstadt.

[58] Hildenbrand, D. and Hitzer, E. (2008). Analysis of point clouds - using conformal geometric algebra. In *Proceedings of the Third International Conference on Computer Graphics Theory and Applications*, page 99–106. SciTePress - Science and and Technology Publications.

[59] Hildenbrand, D., Hrdina, J., Návrat, A., and Vašík, P. (2019). Local controllability of snake robots based on CRA, theory and practice. *Advances in Applied Clifford Algebras*, 30(1):2.

[60] Hildenbrand, D., Zamora, J., and Bayro-Corrochano, E. (2008). Inverse kinematics computation in computer graphics and robotics using conformal geometric algebra. *Advances in Applied Clifford Algebras*, 18(3–4):699–713.

[61] Hitzer, E. and Sangwine, S. J. (2019). Construction of multivector inverse for clifford algebras over 2m+1-dimensional vector spaces from multivector inverse for clifford algebras over 2m-dimensional vector spaces. *Advances in Applied Clifford Algebras*, 29(2):29.

[62] Hitzer, E., Tachibana, K., Buchholz, S., and Yu, I. (2009). Carrier method for the general evaluation and control of pose, molecular conformation, tracking, and the like. *Advances in Applied Clifford Algebras*, 19(2):339–364.

[63] Horn, R. A. and Johnson, C. R. (2012). *Matrix analysis*. Cambridge University Press, 2nd edition.

[64] Hrdina, J., Návrat, A., and Vašík, P. (2018). Geometric algebra for conics. *Advances in Applied Clifford Algebras*, 28:1–21.

[65] Hrdina, J., Návrat, A., and Vašík, P. (2021). Projective geometric algebra as a subalgebra of conformal geometric algebra. *Advances in Applied Clifford Algebras*, 31(18).

[66] Hunt, K. (1991). Kinematic geometry of mechanisms. *Robotica*, 9(1).

[67] K. Davidson, J., H. Hunt, K., and Pennock, G. (2004). Robots and screw theory: Applications of kinematics and statics to robotics. *Journal of Mechanical Design - J MECH DESIGN*, 126.

[68] Kavan, L., Collins, S., Žára, J., and O'Sullivan, C. (2008). Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph.*, 27(4):105:1–105:23.

[69] Kim, J. S., Jeong, J. H., and Park, J. H. (2015). Inverse kinematics and geometric singularity analysis of a 3-SPS/S redundant motion mechanism using conformal geometric algebra. *Mechanism and Machine Theory*, 90:23–36.

[70] Kleppe, A. L. and Egeland, O. (2016). Inverse kinematics for industrial robots using conformal geometric algebra. *Modeling, Identification and Control: A Norwegian Research Bulletin*, 37(1):63–75.

[71] Kleppe, A. L. and Egeland, O. (2018). A curvature-based descriptor for point cloud alignment using conformal geometric algebra. *Advances in Applied Clifford Algebras*, 28(2):50.

[72] Kochanek, D. H. U. and Bartels, R. H. (1984). Interpolating splines with local tension, continuity, and bias control. *ACM SIGGRAPH Computer Graphics*, 18(3):33–41.

[73] Lasenby, A. (2011). *Rigid Body Dynamics in a Constant Curvature Space and the '1D-up' Approach to Conformal Geometric Algebra*, pages 371–389. Springer London, London.

[74] Lasenby, A., Lasenby, R., and Doran, C. (2011). *Rigid Body Dynamics and Conformal Geometric Algebra*, page 3–24. Springer.

[75] Lasenby, J., Hadfield, H., and Lasenby, A. (2018). Calculating the rotor between conformal objects. *AACA: Topical Collection AGACSE 2018, IMECC – UNICAM, Campinas, Brazil*, 29:102.

[76] Lasenby, A.N., Lasenby, J. Wareham, R.J. (2004). A covariant approach to geometry using geometric algebra,. Cambridge University Engineering Department, Technical Report, *CUED/F-INFENG/TR-483*.

[77] Lipkin, H. (2005). Time derivatives of screws with applications to dynamics and stiffness. *Mechanism and Machine Theory*, 40(3):259–273.

[78] Merlet, J.-P. (2006). *Parallel robots*. Solid mechanics and its applications. Kluwer Academic Publishers, 2nd edition.

[79] Minguzzi, E. (2013). A geometrical introduction to screw theory. *European Journal of Physics*, 34(3):613–632. arXiv: 1201.4497.

[80] Müller, A. (2018). Screw and lie group theory in multibody dynamics. *Multibody System Dynamics*, 42(2):219–248.

[81] Perwass, C. (2009). *Geometric algebra with applications in engineering*. Geometry and computing. Springer.

[82] Pessina, L. (2012). Reymond clavel, creator of the delta robot, reflects on his career - sti - school of engineering. https://sti.epfl.ch/reymond-clavel-creator-of-the-delta-robot-reflects-on-his-career/.

[83] Peternell, M. and Pottmann, H. (1997). Computing rational parametrizations of canal surfaces. *Journal of Symbolic Computation*, 23(2–3):255–266.

[84] Petrovic, V., Fallon, J., and Kuester, F. (2007). Visualizing whole-brain dti tractography with gpu-based tuboids and lod management. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1488–1495.

[85] Pierrot, F., Fournier, A., and Dauchex, P. (1991). Towards a fully-parallel 6 dof robot for high-speed applications. In *1991 IEEE International Conference on Robotics and Automation Proceedings*, page 1288–1293 vol.2.

[86] Pottmann, H. and Wallner, J. (2001). *Computational Line Geometry*. Mathematics and Visualization. Springer-Verlag.

[87] Reuleaux, F. (1876). *Kinematics of Machinery. Translated and Edited by Kennedy, A. B. W.* Macmillan and Company.

[88] Segal, A., Haehnel, D., and Thrun, S. (2009). Generalized-icp. In *Robotics: science and systems*, volume 2, page 435. Seattle, WA.

[89] Selig, J. M. (2004). Lie groups and lie algebras in robotics. In Byrnes, J., editor, *Computational Noncommutative Algebra and Applications*, pages 101–125, Dordrecht. Springer Netherlands.

[90] Selig, J. M. (2005). *Geometric fundamentals of robotics*. Monographs in computer science. Springer, 2nd edition.

[91] Selig, J. M. and Martins, D. (2014). On the line geometry of rigid-body inertia. *Acta Mechanica*, 225(11):3073–3101.

[92] Shirokov, D. S. (2021). On computing the determinant, other characteristic polynomial coefficients, and inverse in clifford algebras of arbitrary dimension. *Computational and Applied Mathematics*, 40:173.

[93] Sinclair, A. J. (2005). *Generalization of rotational mechanics and application to aerospace systems*. Texas A&M University.

[94] Tichý, R. (2020). Inverse kinematics for the industrial robot IRB4400 based on conformal geometric algebra. In Mazal, J., Fagiolini, A., and Vasik, P., editors, *Modelling and Simulation for Autonomous Systems*, Lecture Notes in Computer Science, page 148–161. Springer International Publishing.

[95] Tingelstad, L. and Egeland, O. (2017). Motor estimation using heterogeneous sets of objects in conformal geometric algebra. *Advances in Applied Clifford Algebras*, 27(3):2035–2049.

[96] Tingelstad, L. and Egeland, O. (2018). Motor parameterization. *Advances in Applied Clifford Algebras*, 28(2).

[97] Tischler, C. R., Downing, D. M., Lucas, S. R., and Martins, D. (2000). Rigid-body inertia and screw geometry. *Proceedings of a Symposium Commemorating the Legacy, Works, and Life of Sir Robert Stawell Ball Upon the 100th Anniversary of A Treatise on the Theory of Screws*, page 14.

[98] Valkenburg, R. and Dorst, L. (2011). Estimating motors from a variety of geometric data in 3D conformal geometric algebra. *Guide to Geometric Algebra in Practice*, pages 25–45.

[99] Wareham, R. and Lasenby, J. (2008). Mesh vertex pose and position interpolation using geometric algebra. In Perales, F. J. and Fisher, R. B., editors, *Articulated Motion and Deformable Objects*, Lecture Notes in Computer Science, page 122–131. Springer Berlin Heidelberg.

[100] Wareham, R. J. and Lasenby, J. (2011). Generating fractals using geometric algebra. *Advances in Applied Clifford Algebras*, 21(3):647–659.

[101] Zamora, J. and Bayro-Corrochano, E. (2004). Inverse kinematics, fixation and grasping using conformal geometric algebra. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 4, page 3841–3846.