UNIVERSITY OF CALIFORNIA
Santa Barbara

Articulating Space:
Geometric Algebra for Parametric Design – Symmetry, Kinematics, and Curvature

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Media Arts and Technology

by

Pablo Colapinto

Committee in Charge:

Professor Marko Peljhan, Chair

Professor JoAnn Kuchera-Morin

Professor Curtis Roads

March 2016

The dissertation of Pablo Colapinto is approved:

JoAnn Kuchera-Morin

Curtis Roads

Marko Peljhan, Committee Chairperson

December 2015

Articulating Space: Geometric Algebra for Parametric Design —

Symmetry, Kinematics, and Curvature

Copyright © 2016

by

Pablo Colapinto

To family and friends and my partner, Bryn Dudkowski,

many thanks and love.

Acknowledgements

Without the diligent work of others these pages would not be possible. Notably, the author owes an incalculable debt to Professor Leo Dorst, whose publications, solutions, suggestions, and hints have been essential in developing the contained formulations, and whose thoughtful comments proved vital to this document's revision.

The PhD committee has provided critical insight, patient guidance, and comaraderie throughout the preparation of this work. Professor Curtis Roads has been an active listener, encouraging artistic exploration and contributing crucial context and terminology. Professor JoAnn Kuchera-Morin provided endless support, trust, and an empowering vision to advance our shared knowledge. The committee chair, Professor Marko Peljhan, offered a treasure trove of intuition, analysis, and practical application in identifying the problem at hand and our contributions to it.

Attentive readings of and discussions with Joan Lasenby and Eduardo Bayro-Corrochano have proven quite formative as well, and the author would like to extend his thanks to them, as of course to David Hestenes for encouraging us all to "do more." A gracious thanks is due Pierre Dechant, who provided essential training in the basics of reflection groups. Indeed, many members of the geometric algebra community, including Garret Sobczyk, Dietmar Hildenbrand, and Ian Bell, have demonstrated an openness in sharing their expert knowledge.

In addition, this document has directly benefitted from communications with colleagues both at MAT and beyond, including Wesley Smith, who has helped brainstorm and push the GA way, as well as Karl Yerkes, Sölen Kiratli, Peter Flaherty, Charlie Roberts, Lance Putnam, Graham Wakefield, Matthew Wright, Amichi Amar, F. Myles Sciotto, Angus Forbes, August Black, Stephen Pope, the AlloSphere research group, Professor George Legrady, Professor Marcos Novak, Professor Bhaskar Sarkar, Professor Rita Raley, and Professor Alfred Guzzetti.

Research has been funded in part through a Robert W. Deutsch Foundation Fellowship via the AlloSphere Research Facility of the Media Arts and Technology Program at University of California Santa Barbara, and by an Olivia Long Converse Fellowship granted by UCSB's Graduate Division, with additional support provided by The Systemics Lab at MAT. For their financial support, the author would like to thank the AlloSphere director Dr. JoAnn Kuchera-Morin, the Deutsch Foundation President Jane Brown, and the Systemics Lab director Marko Peljhan, as well as the Graduate Division at UCSB and the Media Arts and Technology Program administrative staff.

Of course, any errors or omissions are to be attributed to the author alone.

Curriculum Vitae of Pablo Colapinto

December 2015

EDUCATION

University of California, Santa Barbara 2007-2015
PhD in Media Arts & Technology, December 2015.
MSc in Media Arts & Technology, February 2011.

Harvard University 1996-2000
BA in Visual and Environmental Studies, Magna Cum Laude with Honors.
Mellon Fellow 1998-2000; Detur Book Award 1997.

AWARDS

Robert W. Deutsch Foundation Fellowship	2012-2015
Olivia Long Converse Fellowship	2009-2012
Pew Fellowship in the Arts	2005-2006

PUBLICATIONS

Colapinto, P., Composing Surfaces with Conformal Rotors, In *Proceedings of Applications of Geometric Algebra in Computer Science and Engineering*, Barcelona, 2015.

Colapinto, P. and Peljhan, M., Geometric Algebra for Deployable Design, In *Proceedings of the International Association of Shell and Spatial Structures: Future Visions*, Amsterdam, 2015.

Colapinto, P., Boosted Surfaces: Synthesis of 3D Meshes using Point Pair Generators as Curvature Operators in the Conformal Model, *Advances in Applied Clifford Algebras*, Volume 24, Issue 1, pp 71-78, Springer Basel, 2013.

Colapinto, P., Versor: Spatial Computing with Conformal Geometric Algebra, Thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Media Arts and Technology, University of California, Santa Barbara, 2011.

PRESENTATIONS

Functional Geometry: Producing Pure Spaces, C++ Now Conference, Aspen, Colorado, 2015.

Generic Programming of Generic Spaces: Compile-time Geometric Algebra with C++11, C++ Now Conference, Aspen, Colorado, 2014.

Up and Running with OpenGL. Tutorial, lynda.com, Carpenteria, CA, 2014.

Laberintorios: Espacios sintéticos y la generación de movimientos en álgebra y agua, El Encuentro Transmedia y Narrativas Audiovisuales, Bogotá, 2013.

El Grupo de los Trece, Panel on Latin America and Cybernetics, International Symposium for Electronic Arts, Albuquerque, 2012.

Organic Forms Through Twists and Boosts, International Symposium for Electronic Arts, Istanbul. 2011.

Informaciónes Imaginarias: Arte, Tecnología y la Manipulación de la Realidad, Festival Internacional de la Imagen, Manizales, 2008.

RESEARCH AND TEACHING EXPERIENCE

AlloSphere Research Facility 2010-2015
Graduate Research Fellow *Santa Barbara, CA*

- Collaborate with researchers to develop a fully-immersive 3D environment for artistic and scientific exploration.

University of California 2007-2015
Associate and Assistant Teaching *Santa Barbara, CA*

- Sound and Image. Teaching Associate and Developer of series of multimedia engineering courses for College of Creative Studies.
- Contemporary Cultural Theory and Introduction to Film Theory. Teaching Assistant, Film and Media Department. Fall 2008 and Winter 2009.
- Experiments in Optical Computational Techniques. Teaching Assistant, Media Arts and Technology Program. Spring 2008.

Temple University / Moore College of Art and Design Spring 2002 - 2007
Adjunct Professor *Philadelphia, PA*

- Developed course series on Media Arts Programming for department of Photography and Digital Arts, Moore College of Art and Design. Philadelphia, PA.
- Innovated intermediate level courses on the theory and practice of non-linear video editing in the department of Broadcasting, Telecommunications and Mass Media and the department of Film and Media Arts at Temple University. Philadelphia, PA.

ARTISTIC PRODUCTION

Metropolitan Opera 2011-2013
Visual Effects Post-Production Supervisor *New York City, NY – Lyon, France*

- Directed a team of computer graphics specialists, 3D artists, and software engineers for large-scale projection integrated into production of Wagner's *Parsifal*.

Institute of Contemporary Art 2007
Media Artist *Philadelphia, PA*

- Created *Pass Back a Revolver*, a projection-mapped multimedia installation collaboration with Peter Flaherty, as part of the *Locally Localized Gravity* exhibit. Available at wolftype.com.

Lincoln Center and Japan Society 2004 - 2005
3D Graphics Specialist *New York City*

- Produced 3D video effects for *My Life as a Fairy Tale*. Director: Chen Shi-Zheng, Composer: Stephin Meritt, Producer: Lincoln Center Festival.
- Developed creative projections for *Dogugaeshi*, an innovative puppet theatre. Director: Basil Twist, Producer: Tandem Otter.

TECHNICAL SKILLS

Computer	C++ for Multimedia Engineering, OpenGL, Lua, Python
Mathematics	Geometric Algebras and Spatial Computing
Software	3D Modelling and Video Editing
Languages	English, Spanish, French

Abstract

Articulating Space: Geometric Algebra for Parametric Design – Symmetry, Kinematics, and Curvature

by

Pablo Colapinto

To advance the use of geometric algebra in practice, we develop computational methods for parameterizing spatial structures with the conformal model. Three discrete parameterizations – symmetric, kinematic, and curvilinear – are employed to generate space groups, linkage mechanisms, and rationalized surfaces. In the process we illustrate techniques that directly benefit from the underlying mathematics, and demonstrate how they might be applied to various scenarios. Each technique engages the versor – as opposed to matrix – representation of transformations, which allows for structure-preserving operations on geometric primitives. This covariant methodology facilitates constructive design through geometric reasoning: incidence and movement are expressed in terms of spatial variables such as lines, circles and spheres. In addition to providing a toolset for generating forms and transformations in computer graphics, the resulting expressions could be used in the design and fabrication of machine parts, tensegrity systems, robot manipulators, deployable structures, and freeform architectures. Building upon existing algorithms, these methods participate in the advancement of geometric thinking, developing an intuitive spatial articulation that can be creatively applied across disciplines, ranging from time-based media to mechanical and structural engineering, or reformulated in higher dimensions.

Contents

1 Introduction: Transforming Expression 1

- 1.1 Background . . . 1
- 1.2 Motivations . . . 2
- 1.3 Problem Definition . . . 5
- 1.4 Goals . . . 9
- 1.5 Methodology . . . 12
- 1.6 History . . . 14

2 Geometric Algebra 19

- 2.1 Summary . . . 19
- 2.2 Beginnings . . . 20
  - 2.2.1 Orthogonal Transformations . . . 20
  - 2.2.2 Canonical Basis . . . 21
  - 2.2.3 Basis Blades . . . 23
  - 2.2.4 Geometric Product . . . 24
  - 2.2.5 Duality . . . 27
  - 2.2.6 The Meet . . . 28
- 2.3 Transformations . . . 29
  - 2.3.1 Versors . . . 29
  - 2.3.2 Reflection . . . 30
  - 2.3.3 Projection and Rejection . . . 31
  - 2.3.4 Rotations . . . 32
  - 2.3.5 The Remarkable Exponential . . . 32
  - 2.3.6 Rationalizing with Ratios . . . 35
  - 2.3.7 Example Problem: Preventing Camera Roll . . . 36
  - 2.3.8 Other Metrics . . . 37
- 2.4 Conformal Geometric Algebra . . . 38
  - 2.4.1 Conformal Metric . . . 38
  - 2.4.2 Blades . . . 40
  - 2.4.3 Primitive Elements . . . 43
    - 2.4.3.1 Points as Null Vectors . . . 43
    - 2.4.3.2 Direct Representation . . . 46
    - 2.4.3.3 Dual Representation . . . 47
  - 2.4.4 Transformations . . . 51

2.4.5 Interpolation 54
2.4.5.1 Finding The Logarithm 55
2.4.5.2 Ratios of Rotors 56
2.5 Notation 56
2.6 Example: Constructing Folds with Round Elements 58
2.7 Discussion 61

3 Symmetry: Constructing Space Groups 65
3.1 Summary 65
3.2 Point Groups 68
3.2.1 Specifying root versors from angular constraints *p* and *q* 69
3.2.2 Generating a complete set of transformations from the simple roots **a**, **b** and **c**. 72
3.3 Space Groups 74
3.3.1 Constructing Bravais Lattice Cells from Root Versors 77
3.3.2 Substituting generators with joining constraints for nonsymmorphic space groups 77
3.3.2.1 Glide Reflections 79
3.3.2.2 Screw Displacements 80
3.4 Application: Balanced Structures 83
3.5 Discussion and Future Work 85

4 Kinematics: Constructing Linkage Mechanisms 89
4.1 Summary 89
4.2 Space Frames 91
4.3 Kinematics 92
4.4 Inverse Kinematics 97
4.4.1 Iterative Construction of Revolute k-Chains 100
4.4.2 Analytical Construction of the Bennett Linkage 105
4.5 Application: Deployable Structures 111
4.6 Discussion and Future Work 115

5 Transformations: Constructions through Interpolation of the Bivector Exponential 117
5.1 Summary 117
5.2 The Motor Algebra 118
5.2.1 Exponentiating A Dual Line 118
5.2.2 Logarithm of a motor 120
5.2.3 Interpolation of motors 121
5.2.4 Motor Compositions 122
5.3 Point Pairs as Generators 122
5.3.1 Bending the Line 124
5.3.2 Orbits 127
5.3.3 Orthogonal Orbits 127
5.3.4 The Hopf Fibration 129

5.3.5 Taking a Sphere to a Sphere . . . 132
5.3.6 Taking a Circle to a Circle . . . 133
5.4 Discussion and Future Work . . . 138

6 Curvature: Constructing Cyclidic Nets 139
6.1 Summary . . . 139
6.2 Cyclidic Nets . . . 140
6.3 Principal Patches . . . 141
6.3.1 Outline of the Algorithm . . . 142
6.3.2 Constructing Tangents . . . 144
6.3.3 Triply Orthogonal Coordinates . . . 147
6.3.4 Null Tangents as Coordinate Surface Generators . . . 147
6.3.5 Coordinate Surface Spheres at Four Points on a Circle . . . 149
6.3.6 Rationalization of Coordinate Surfaces with Simple Rotors . . . 151
6.3.7 The Direction of Interpolation . . . 151
6.3.8 Surface Blending . . . 153
6.4 Application: Freeform Structures . . . 155
6.5 Discussion and Future Work . . . 156

7 Conclusion: A Mechanism for Design 161
7.1 Summary . . . 161
7.2 Contribution . . . 163
7.3 Future Work . . . 165
7.4 Final Thoughts . . . 167

A Compile-Time Geometric Algebra with C++11 171
A.1 Summary . . . 171
A.2 Computer Implementations of Geometric Algebra . . . 172
A.3 Implementation Goals . . . 174
A.4 An Example: Generating $\mathscr{G}^3$ . . . 176
A.5 Blade Basics and Constant Expressions . . . 178
A.6 Multivectors and Variadic Templates . . . 181
A.7 Geometric Products and Instruction Lists . . . 184
A.8 Discussion . . . 190

B 2D Crystallographic Space Groups 192

List of Figures

1.1 An example of covariance. 4
1.2 Transformation matrices. 8

2.1 Basis 1-blade in $\mathbb{R}^3$ 24
2.2 Basis 2-blades in $\mathbb{R}^3$ 24
2.3 Basis 3-blade in $\mathbb{R}^3$ 24
2.4 Meet of two blades 29
2.5 Reflection 31
2.6 Rotation 33
2.7 Preventing unwanted camera roll 37
2.8 The horosphere 45
2.9 The null basis and the horocycle 45
2.10 The circle intersection of a sphere and a plane. 48
2.11 The meet of spheres 49
2.12 Point pairs 49
2.13 Factoring rounds 51
2.14 Conformal transformations as orthogonal versors 54
2.15 Degree-4 vertex 58
2.16 Folding motion encoded as a circle 59
2.17 Folding motion constrained by spheres 60
2.18 Miura-Ori crease pattern 61
2.19 Miura-Ori folding motions 62

3.1 The cosine rule from spherical trigonometry 71
3.2 Point Group Symmetries 76
3.3 Glide Reflections 79
3.4 Space groups of holohedral group 42 80
3.5 Space Group screw displacements 82
3.6 Space Group screw displacements 82
3.7 Balanced Forms 84
3.8 Point Group $\bar{3}\bar{3}$ 85
3.9 Tensegrities of Point Group $\bar{3}\bar{3}$ 86
3.10 Tensegrities of Space Group $P\bar{3}\bar{2}$ 86

4.1 An orthonormal frame at point p. 91
4.2 Denavit-Hartenberg kinematic parameters 95

4.3 Forward Kinematics of a k-chain 96
4.4 A robot arm can be modelled as a kinematic chain of frames. 97
4.5 Geometric content of a frame 100
4.6 Geometric encoding of kinematic parameters 101
4.7 The forward reaching step of the FABRIK method for spherical joints constraints 104
4.8 The backward reaching step of the FABRIK method for circular joints constraints 105
4.9 The forward reaching step of the FABRIK method for circular joints constraints 106
4.10 Inverse kinematics of a chain of revolute joints 107
4.11 Spatial movement of a Bennett mechanism 109
4.12 Geometric constraint of a Bennett linkage 109
4.13 Linked Bennett mechanisms 113
4.14 Deployment sequence of a circular truss composed of linked Bennett mechanisms. 113
4.15 Deployment sequence of a branching linked Bennett mechanism 114
4.16 Deployment sequence of a 5-legged linked Bennett mechanism 114

5.1 Anatomy of a dual line 119
5.2 Interpolation of motors 122
5.3 Trilinear interpolation of motor-valued frames 123
5.4 Mesh deformations created through trilinear interpolation of motors 123
5.5 An example using motors as mesh deformations 124
5.6 Coupled motors 125
5.7 Bending a line into a circle 126
5.8 Transversive transformations of a point 127
5.9 Knotted orbits 129
5.10 Hopf fibers as point pair generators 130
5.11 Constructing a Hopf fibration 131
5.12 Twistor meshes representing the 3-sphere Hopf fibration 133
5.13 Isoclinic rotations generated by the ratio of two orthogonal circles 136
5.14 The action of conformal rotors 137

6.1 Concircularity of the principal patch corners 142
6.2 Principal cyclidic patches 142
6.3 Coordinate surface spheres 144
6.4 Triply orthogonal coordinate system 146
6.5 Construction of a principal surface patch 150
6.6 Taking a sphere to a sphere 150
6.7 Two directions of interpolation of a conformal rotor 152
6.8 A patch is a C^1-smooth blend between constant coordinate surfaces 153
6.9 Construction of a circle net to blend between two spheres 154
6.10 Various patches blending two spheres. 154
6.11 C^1 smooth cylidic net surfaces constructed from a series of principal patches. 157
6.12 Surface patches added along a normal to create a volume 157

6.13 Circular nets extended in a third direction to discretize volumes 158
6.14 Discretized volumes used to parametrize the conformal deformation of a sphere. 158

List of Tables

1.1 Geometric transformations. 4

2.1 Basis blades of $\mathscr{G}^3$ 23
2.2 Minkowski Metric of $\mathbb{R}^{1,1}$ 38
2.3 A nondegenerate basis for $\mathbb{R}^{4,1}$. 40
2.4 A degenerate null basis for $\mathbb{R}^{4,1}$. 40
2.5 Basis blades in Conformal $\mathscr{G}^{4,1}$ 42
2.6 Basic elements of 3D conformal geometric algebra 64

3.1 *pq* angular constraints 71
3.2 *pq* point group notation 74
3.3 Holohedral Point Group 22 and its subgroups 75
3.4 Lattice Cells 78
3.5 Screw displacements of the form $P\bar{6}_s\bar{2}$ 81

7.1 Relations between technique, parameter, property, and structure 162

A.1 Binary Representation of Basis Blades in $\mathscr{G}^3$ 178

List of Algorithms

2.1 Preventing Camera Roll 36
3.1 Construct Transformation Sequence 74
4.1 Constrain a Point to a Circle 98
4.2 Calculating Circle of Positions of Next $(i+1)$th Frame 99
4.3 Calculating Rotation of Previous $(i-1)$th Frame 99
4.4 Calculating Circle of Positions of Previous $(i-1)$th Frame 99
4.5 Inverse kinematics of a chain of spherical joints 105
4.6 Inverse kinematics of a chain of revolute joints 106
4.7 Closed-form solution to the Bennett Linkage 110
4.8 Calculation of Joint Angles based on Positions 110
5.1 Technique for constructing a suitable Bivector B 136
6.1 Defining principal patch parameters from two osculating contact spheres. . . 154
6.2 Calculating A 3D Conformal Coordinate 156
A.1 Get Grade of Blade a 179
A.2 Geometric Product of Basis Blades 179
A.3 Check for Sign Flip of Geometric Product of a and b 180
A.4 Check for Inner Product (Left Contraction) 180
A.5 Check for Outer Product 180
A.6 Order Comparison 181
A.7 Insert-Sort 183
A.8 Building Instructions 186
A.9 Check for Existence of Blade in a Multivector 186
A.10 Multivector Return Type from an Instruction List 187
A.11 Sorting Instructions 189
A.12 Executing Products 190

Chapter 1

Introduction: Transforming Expression

1.1 Background

> We still call numbers ‘figures’, i.e. shapes.
>
> – Arthur Koestler, *The Sleepwalkers*[1]

Today, researchers have access to a universal framework for geometric reasoning – one that provides a proper mathematical language of form and transformation with which to share their ideas. **Geometric algebra**, the mathematics applied throughout this document, provides this framework. With its internal algebraic mechanisms geared towards representing primitives such as *circle*, *line* and *plane*, as well as movements such as *rotating, twisting* and *bending*, geometric algebra clarifies spatial concepts across diverse fields of research. This comprehensive model of geometry has been applied to both physical and virtual domains – from quantum physics, electromagnetism, and astrophysics, to robotics, machine learning, and computer graphics – and its potential users include anyone who works with spatial sys-

---

[1] From his text on the history of humanity’s comprehension of the cosmos, Koestler here is referring to the influence of the Pythagoreans, who were possibly the first to understand the geometry behind numerical operations. That numbers are forms themselves is central to understanding synthesis with geometric algebra. See also Garret Sobczyk’s text expanding the “geometric concept of number” [126].

tems.[2] Geometric algebra is the syntax of space, offering its practitioners a coherent[3] analysis of structure, form and movement.

Using this syntax, the development of expressions and processes to generate spatial structures on a computer represents a specific strategy for **parametric design**, where the relationships between geometric entities and operations serve as inputs to a system for configuring form. These structures may remain purely virtual - a flapping wing in a video animation or a plant-like tendril in a digital print – or they may be manufactured and deployed in the real world (as a stent, tent, antenna, bridge or building, etc).

This dissertation illustrates several algorithmic techniques for the synthesis of structures with geometric algebra. Using a computer to model our constructions, we select three parametric systems to synthesize *de novo* – symmetry groups, kinematic mechanisms, and curvilinear surfaces and volumes – and explore how each of these parameterizations enables a wide range of spatial articulation.

1.2 Motivations

> If you want to see, learn how to act.
>
> – Heinz von Foerster

How can we express structure with geometric algebra? To answer this question we must determine what mechanisms the algebra provides, what parameters to structure these mechanisms express, what structures can be designed with these parameters, and what forms these structures can take. As an exploration of a system of processes at play in the mathematical

---

[2] While it has become common to find the term ‘*spatial sciences*’ used specifically in regards to geospatial studies, here we use ‘*spatial systems*’ to include a more general category of study that relies on or advances spatial concepts. That is, any static or dynamic configuration of points, primitives, or operators on a real or simulated manifold, including n-dimensional point clouds, meshes, function spaces, symmetry groups, mechanisms, orbitals, vector fields, architecture, imagery, and organic and inorganic material, etc.

[3] In his preface to *New Foundations for Classical Mechanics* [66], David Hestenes, the central promoter of the unifying modern day geometric methods that are used in this text, posits the need for “externally coherent” methodologies so that scientists can face less noise as they navigate across fields of research.

composition of structure, form, and movement, our question is morphological in nature.

The expressivity of geometric algebra enables constructive design. Constructive design is a methodology for developing techniques and pedagogy to build complex spatial arrangements from simpler forms. Practicing constructive synthesis facilitates deconstructive analysis, helping to build intuitions and methodologies that can be applied to a variety of design problems. With a constructive design system, implementation details enlighten rather than obfuscate applications, providing a mechanism for developing new ways of seeing the real world around us.

Geometric algebra is characterized as expressive because it encodes spatial relationships in a complete, compact, and composable way, exhibiting:

- Universality: the capacity to encode both Euclidean and non-Euclidean geometry.
- Elegance: the tendency to be succinct and coordinate-free.
- Covariance: the property of preserving structures under transformation.

This last component of expressivity – covariance – signals a proper encoding of geometric transformations. In 1872 Felix Klein presented his *Erlangen Program*[4], which classifies geometries in terms of the group of transformations admitted into a space (e.g. projective, affine, conformal, orthogonal – see Figure 1.1), thereby prioritizing process over form as the object of study; geometry is *active*. These transformations – a type of **morphism** – operate on subspaces, and are characterized by the particular geometric properties they leave invariant. Furthermore, our algebraic structures – which include both subspaces and the morphisms themselves – should behave consistently under these transformations: results should not depend on whether we twist the algebraic structure as a whole or each of its components separately (see Figure 1.1). It is this structure-preservation under the process of transformation that we call **covariance**. To design structures in this way – as a series of structure-preserving

---

[4] For an English translation see [84]

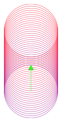
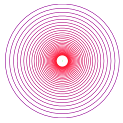
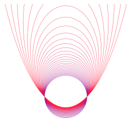
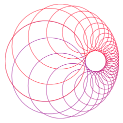
Transformation	**Preserved Property**	
Orthogonal	Distances and Angles	
Affine	Parallelism and Ratio	
Projective	Flat Elements and Cross Ratio	
Conformal	Angles	

Table 1.1: Geometric transformations are covariant morphisms on subspaces that leave certain properties invariant. In this table, the subspace being transformed is a circle.

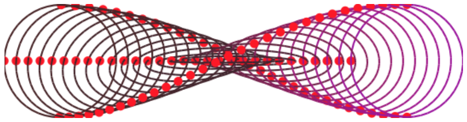


Figure 1.1: One example of covariance is the **outermorphism**: the twisting of the circle through three points is equivalent to the circle through three twisted points. If $R[x]$ is a twisting transformation and $p_a \wedge p_b \wedge p_c$ a circle through three points, then $R[p_a \wedge p_b \wedge p_c] = R[p_a] \wedge R[p_b] \wedge R[p_c]$. As another example, the full geometric product is also preserved under transformation: $R[ab] = R[a]R[b]$ – or, in other words, the rotated product of vectors is equivalent to the product of rotated vectors.

transformations in some metric – allows us to work with space as a trustworthy operator itself, and to hone our geometric concepts knowing that our efforts will be generalizable to other problems formulated in other dimensions. With this added consistency comes access to other fields of research outside our current domain, such as quantum mechanics, optics, or relativity, guided by a single mathematical language of space that can be extended and shared.

1.3 Problem Definition

> The obstacles of achieving a facile relationship of people and things seems to inhere not so much in the structure of things themselves as the structure of our ideas and values.
>
> – Ron Resch, *The Topological Design of Sculptural and Architectural Systems,* 1973.

We have at hand a single mathematical apparatus a student can master to explore the use of geometric reasoning in practice. Yet despite the natural coherence between geometric concept, algebraic representation, and computerized form that geometric algebra provides, familiarity with the apparatus is limited.[5] In particular, the use of geometric algebra as a software tool for computer-aided geometric design is not yet a well-established field of study. As of this writing, none of the leading platforms in Computer-Aided Design (e.g. AutoDesk, SolidWorks, Rhino3D), 3D animation (e.g. Houdini, Maya, 3Ds Max, Blender), or creative coding (e.g. OpenFrameworks, Processing, Touch Designer, Cinder) have geometric algebra packages available. Tools are restricted to the more mathematically inclined users of MATLAB and Mathematica. Appendix A offers an overview of GA software libraries.

The reasons for this lack of adoption in digital design are varied. Some will trace it to the fact that the *vector analysis* of Gibbs and Heaviside won the attention of mathematicians in the early 1900s, and may point to the resistance of learning a new branch of math: much work has already been accomplished to design structures with matrix transformations, Bézier and B-Spline techniques, and algebraic geometry.[6] Others may point to the as-of-yet unfulfilled

---

[5] Attendance numbers at the most recent international conference on Applications of Geometric Algebra in Computer Science and Engineering (as of this writing, AGACSE 2015 in Barcelona) were not much greater than the original in 1999. With little exception (Eduardo Bayro-Corrochano's cybernetics laboratory in Guadalajara is one) there are no research centers dedicated strictly to the advancement of geometric algebras, and practitioners tend to work independently or collaborate by electronic communication. Researchers in the field – from physicists to computer graphics programmers to geometers - frequently lament the lack of adoption of this versatile system for spatial articulation.

[6] As opposed to geometric algebra which works directly with n-dimensional forms, algebraic geometry parameterizes curves and surfaces with polynomials in terms of coordinate coefficients.

need for a *discrete geometric calculus* with which to analyze surfaces on a computer, or that the conformal model itself was not applied to computer graphics until 2001, or that it is difficult to design an efficient implementation.

As a result, most geometric computing software is built without geometric algebra. Existing algorithms are dominated by matrix representations of transformations and coordinate representations of vectors, encouraging an over-reliance on linear algebra and an unnecessary decomposition of geometry into 1D systems. In a presentation entitled *Geometry and the Design of Structures*[7], Bill Baker – the structural engineer behind the tallest[8] building in the world – has suggested the dependance on matrices limits on our ability to design with geometric thinking. Baker flashed a slide of a charging rhinoceros[9] to illustrate his position that computational geometry software, with its under-the-hood matrix formulations, was trampling intuitive design. The disconnect between the geometric reasoning and matrix-based computing stems from the fact that there is nothing inherently geometric about a matrix (excepting its rectilinearity – see Figure 1.2). Meanwhile, in [93](p13), Li points to two deficiencies of expression of the typical Cartesian decomposition of geometry into coordinate systems: the inability for these 1D systems to adequately represent invariance or covariance of the represented geometry under transformations, and the tendency for algorithms to explode in complexity under the hood.

In practice these representation problems – poor structure preservation and unmanaged complexity – mean that manipulating spatial relationships on a computer requires jerry-rigged techniques pulled from a toolbox of non-compatible parts, and practitioners who work with spatial concepts are distanced from the implementation details. Geometric algorithms are heterogeneous combinations of real, complex, matrix, and tensor algebras heavily seasoned with trigonometric functions, mixed in ways dictated more by the quest for code efficiency than by

---

[7] *Proceedings of the International Association of Shell and Spatial Structures*, Amsterdam, 2015.

[8] The Burj Khalifa in Dubai, tallest as of 2015.

[9] His implicit reference was to the powerful Rhinoceros 3D, a popular modelling software tool for architectural geometry.

the spatial concepts they claim to model.[10] Vector algebra (addition, dot and cross product) is used to calculate mechanical forces, matrices (rotation, translation, shearing) for Euclidean transformations and projective geometry (except quaternions for 3D rotations), Plücker coordinates for robotics (twisting and affine transformations of lines), complex numbers are used for electromagnetic wave propagation (polar coordinates, magnitude, and phase), tensor algebras and exterior calculus for differential and surface geometry (divergence and curl), combinatorics for topology, Lie algebras for symmetry groups. While these mechanisms function accurately in their respective applications, instances of them in a computer program do not share membership within one single coherent algebra for construction, and so do not compose[11] well with each other. Implementation details involve extracting particular coordinates from particular mathematical models, resulting in software that is laborious to develop and maintain, and not easily generalizable to other problems or spaces. Nor does knowledge of one ease the learning of another. For instance, a wide divide lies between the mathematics of geometric design used for computer-assisted modelling of a 3D object and the matrix, Lie, or Plücker algebras used for controlling the robotic movements to fabricate it – different branches of mathematics are applied in simulation and fabrication phases of the same object.[12]

David Hestenes has repeatedly pointed out that this *incoherency* of expression guarantees inefficient learning. From the perspective of creative exploration of form, the absence of a geometric algebra toolset limits the digital practitioner to non-constructive methods. An individual problem task – tiling a space, modelling a folding process, constructing a linkage mechanism, designing a doubly-curved shell structure, weaving a pattern – is tackled sepa-

---

[10] One notable exception to this trend is the push for a unifying discrete exterior calculus for computer graphics at CalTech. Another is of course the geometric algebra explained in the current work. There is a continuing struggle within the geometric algebra community, and surely as well within the discrete exterior calculus community, to convince newcomers to spend the time necessary to learn it. Exterior calculus, it should be noted, is a subsystem of the more encompassing geometric calculus.

[11] Functional programming styles, such as found in Haskell or O'Caml languages, help in the design of composable algorithms. They do not themselves answer the question of how to represent geometric concepts.

[12] This schism is mirrored in the arbitrary division of computational geometry into combinatorial (geometry-based) and numeric (algebra-based) methods, a common division in mathematics lamented by Klein himself.

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A = \begin{bmatrix} 1-2a^2 & -2ab & -2ac \\ -2ab & 1-2b^2 & -2bc \\ -2ac & -2bc & 1-2c^2 \end{bmatrix}$$

Figure 1.2: R_x, R_y, and R_z are some typical rotation matrices used in computer graphics. Matrix A can used to multiply a vector over the plane $ax+by+cz=0$. While there are patterns in the matrix form of transformation, there is little geometric content and no reason why anyone should be able to deduce what they do just by looking at them, let alone work out how to construct them without a reference manual. Also, notice the high-amount of redundancy in the number of 0 terms and repetition of the trigonometric entries. This notation is not expressive.

rately and without recourse to borrowing algorithms developed specifically for another task. This lack of constructivity negatively affects potential collaboration. Because no singular strategy for geometric computing informs individual tactics, participation in the design of complex spatial structures is limited to specialists who have mastered particular tasks.

Encouraging participation in an alternative spatial computing practice requires developing its power of raw synthesis – demonstrating how geometric algebra can express complex form with sheer formula. The specific advantages of using GA as an engine to generate *new* structures requires further investigation and publication. Synthesis techniques and software tools that implement them must be developed, so that they may be experimented with. Since geometric algebra has been used successfully to solve wide range of engineering and computer graphics problems, existing algorithms and geometric constructions should be corralled for the distinct purpose of generating structures. Many spatial articulations remain to be expressed. The generic generative geometric algebra we will explore is capable of creating a wealth of expressions from its own internal syntax, and many discoveries await the patient student.

1.4 Goals

The father of the arrow is the thought: how can I expand my reach?

-Paul Klee

This document addresses one requirement for the study of geometric algebra as a tool in computational design: more synthesis techniques are needed. Detailing the process of construction with geometric expressions will increase familiarity with the mathematics, and reduce dependence on matrices and coordinates. The challenge of software implementation for synthesis is addressed in the accompanying code base at

`github.com/wolftype/versor`

and in Appendix A. We noted that another obstacle to computational design with GA is the absence of a fully developed *discrete geometric calculus* with which to work with differential geometry. This is not directly addressed here, however some *tangential* contributions to that effort include the rationalization techniques of Chapter 6.

We propose that extending the practice of geometric algebra will

- *open* new geometries to experimental exploration
- *increase* familiarity with the mathematical system
- *advance* the mathematical system
- *develop* geometric intuition
- *facilitate* transdisciplinary collaboration
- *enable* novel designs of spatial structures

Developing Technique

To illustrate and advance the expressive capabilities of geometric algebra in producing form, we will use the conformal model to synthesize spatial systems. We construct three parameterizations of spatial systems – crystallographic, kinematic, and curvilinear – by way of three different techniques – transformation, constraint, and rationalization. We select these systems because they each require different geometric reasoning tactics and yet can all be expressed within the unifying framework of conformal geometric algebra. Thus while the systems are typically given separate treatment in computational geometry texts, we aim to show that methods developed in the construction of one can be applied to the construction of another. This not only sheds light on the versatility of GA, but in turn uses GA to shed light on the shared geometric concepts that can be used to govern these three types of structure.

Many individual components of the larger algorithms contained herein are present in the literature referenced at the beginning of each chapter, however particular details are worked out here for the first time, especially in terms of procedures and transformation sequences. As we aim to present a self-contained manual explicating various techniques for the synthesis of spatial articulations, in Appendix A our programming implementation details reveals a technique for encoding the core generator of this algebra. The online code base provides this in under 200 kilobytes.

Beyond the screen, the productive nature of our algorithms extends into potential physical manifestations; we parameterize our virtual models using spatial systems that support real-world manufacture – material, robotic, and architectural. We aim to demonstrate a contribution to the field of computer-assisted design, where our algorithmic methods can be embodied as physical structure. The techniques are applied to systems that could result in structures built in the physical world.

Developing Pedagogy

Each step in the sequence of our exploration – from symmetries to movements to surfaces – incorporates additional features of the mathematics, while simultaneously relying on methods delineated during the previous step. The sequence of articulations represents a pedagogy. As we investigate the implementation details of these three parameterizations of 3D structures, we build a general toolset for framing spatial structures in terms of their sub-geometric components. In detailing these methods for spatial synthesis, we help to demarcate a comprehensive strategy for digital practice. This building-block process is similar to Paul Klee's *Pedagogical Sketchbook* [83] or Wassily Kandinsky's *Point and Line to Plain* [82], where primitive shapes are activated into motions in order to develop expressibility.

This focus on movement is critical to learning. A core feature of geometric algebra is the representation of transformation as a type of number called a **rotor** or **spinor**, rather than the generic matrix grids of Figure 1.2. In his introduction to *New Foundations in Classical Mechanics* [66], David Hestenes, who reinvigorated interest in geometric algebra in the 1960s, explains that the use of spinors instead of matrices for calculating transformations is both more efficient computationally and more expressive theoretically – a workflow optimization that serves as a bridge to advanced topics. To leverage this benefit, we must get a better feel for what these rotors do, and so we must see them in action ("if you want to *act,* learn how to *see").*

Our computational approach to geometric design of spatial structures is **constructive**. The constructive computational design is one that is expressive in the following ways:

- **Generic / Complete / Universal**: A wide range of morphologies are shown to be articulated. In addition, the mathematical relationships are expressable, like many good algebras, by words. The methods used not only build on each other, but also inform constructions in other dimensions or non-Euclidean spaces.

- **Geometric / Compact / Elegant**: Symbols represent geometric primitives. Exponentials represent transformations. There is no need for convoluted syntax built with matrices and trigonometric functions. A self-sufficient semantics is the minimum necessary to induce a wealth of resulting methods. Spatial relationships and transformations are built mathematically from the ground up, “from scratch”.

- **Generative / Composable / Covariant**: Algorithms developed for a design are reusable in another. Reliance on geometric primitives such as circles and spheres as components ensures that design parameters are modular and therefore reconfigurable. Transformations are the central mechanism defining form, through a system that honors covariance.

1.5 Methodology

> Our studies should lead to constructive thinking.
>
> – Josef Albers

Our current task is to implement a variety of techniques for the generative synthesis of spatial structures using geometric algebra. While this a text in mathematics applied to computer-assisted design, it is important to emphasize the fact that no previous mathematics is assumed of the reader. Since synthesis technique is our central concern, no mathematical proofs of the formulas developed are provided, and readers interested in such justifications are recommended the list of references in Section 1.6. Instead, here the visualization of form serves to demonstrate functionality, providing *proofs-of-geometric-concept* through construction of images. We will focus on the process of generating structures and suggest some potential uses in real-world applications.

Each Chapter builds upon algorithms found in the literature, providing new details on implementing parametric systems. We outline the mechanics of geometric algebra and the conformal model in Chapter 2. In Chapter 3 we use transformations to detail the implementation of a paper written by David Hestenes and Jeremy Holt on the use of CGA in formulating the

230 3D crystallographic space groups [73], and propose the use of such groups in designing symmetric shapes and tensegrity systems. Moving from static to kinematic forms, Chapter 4 investigates the use of a particular screw transformation called a motor to create linkage mechanisms and simple folding apparatus, introducing precise methods for inverse kinematics of a chain of revolute joints, and offering a novel solution to the design of the canonical Bennett mechanism. Articulated structures are proposed based on networks of such mechanisms. Chapter 5 explores continuous transformations of these motors to create interpolations and twisted deformations of forms, and extends the results to conformal transformations in order to present examples of knotting and fibering, including a unique formulation of the Hopf fibration. This serves as a segue to Chapter 6 where we apply conformal transformations to the building of cyclidic nets, which are piecewise smooth surfaces made from circular arcs. In Appendix A, we outline how this general approach to spatial calculations can itself be generated by implementing its formal structure using C++ template metaprogramming. Implementing geometric algebra software that is both generic and efficient is itself a lesson in computer science.

Video animations of much of the work developed here can be found online at:

`www.vimeo.com/wolftype`

and updated algorithms, code, and any revisions of this document at:

`versor.mat.ucsb.edu`

and

`github.com/wolftype/versor.`

1.6 History

> Geometric Algebra is no less than a universal mathematical language for precisely expressing and reasoning with geometric concepts.
>
> – David Hestenes [68]

Louis Couturat's 1901 treatise *La Logique de Leibniz* makes clear the grand philosopher's 17th century belief in a holistic "geometric calculus" of analysis and synthesis where intuition and precision can operate together to figure the articulation of machines. Writing to his teacher Huygens in 1679, Leibniz states "I think we still need another, properly geometrical linear analysis that will directly express for us *situation*, just as algebra expresses *magnitude,"* that can "represent figures and even machines and movements with characters" such that, as he expounds in a later letter to Marquis de L'Hôpital "calculations in it are true representations of shape and lead directly to constructions." [13] Descartes' analysis falls short of this goal of a metric-indepedent disambiguation of situation (a point in space) from magnitude (a vector). Möbius' barycentric calculus and Hamilton's quaternions were other attempts at enabling operations on geometric elements. Couturat's text eventually points to Hermann Grassmann, the mathematician who, without knowing of Leibniz's vision of a calculus for geometric reasoning, executed it 200 years later.

Grassmann's theory of forms, introduced in [59, 60] as a theory of *extension*, provides the backbone of how to organize spatial concepts mathematically, and serves an elegant way to construct algebraic elements of direction, area, and volume. Grassmann endeavors to explain his motivations in his 1862 re-writing of his misunderstood 1844 treatise, by explaining that his theory is one "which extends and intellectualizes the sensual intuitions of geometry into general, logical concepts..." and "could be said to form the keystone of the entire structure of mathematics." This universal model of space was intended to be profound, emphasizing

---

[13] Chapter 9 addresses Leibniz's quest for geometrical analysis. An 2012 English translation by Donald Rutherford and R. Timothy Moore is available online [30].

the *relational* aspects of the mathematics. In an essay detailing the genealogy of theoretical biology, Gare argues that "what needs to be emphasized is that Grassmann's work continued the tradition of Schelling, Weiss, and his father, Justus Grassmann, the goal of which was to provide the means to grasp the self-formation of nature, including life."[57]

In 1878 William Clifford formulated Grassmann's work on geometric relationships into a *geometric algebra* of transformations [24]. Clifford demonstrated how Grassmann's foundational associative algebra of extensions *induced* William Hamilton's invertible algebra of rotations (the quaternions, which can spin vectors in 3D space). Because of their ability to encapsulate both dimension and transformation, Clifford referred to these mathematical spin structures as *geometric algebras*. Articles by Clifford himself [26, 25] help to round out the philosophy behind spatial reasoning. A summary of the history of the geometry leading to Clifford's geometric algebra can be found in Hestenes [66], Sobczyk [126], and Li [93], and of algebras in general in Van der Waerden [133].

With the notable exceptions of Giuseppe Peano, who re-articulated Grassmann's work in *Geometric Calculus* [108], and Alfred North Whitehead, who presented it as a possible *Universal Algebra [137],* the Grassmann-Clifford geometric system was left relatively unexplored outside the realm of spinor mathematics until David Hestenes found Marcel Riesz's series of lectures delivered in the late 1950's entitled *Clifford Numbers and Spinors* [120]. In reading Reisz, Hestenes discovered that Clifford's spin algebras included transformations that were isomorphic to matrices common in quantum mechanics (namely, the Pauli and Dirac matrices), and used them to develop a geometric algebra of spacetime. In 1966, with the publication of his now classic text on Space-Time Algebra [65], Hestenes awakened interest in the hypercomplex Clifford Algebras by exploring their inherently geometric nature, their utility in describing a range of classical and quantum mechanical phenomena, and their unifying integration of various algebras. Continued treatment developed the system into a full calculus in Hestenes and Sobcyk's text *Clifford Algebra to Geometric Calculus* [75], which was further elaborated into a mechanism for differential geometry in [72, 68, 69, 125].

In 1991, citing previous representations by Lounesto, Ahlfors, Anglès and others, Hestenes published an article [67] clarifying the use of geometric algebra in producing projective and conformal geometric spaces. In 2001, along with Hongbo Li and Alan Rockwood, Hestenes elaborated this work, based on a model developed by a student of Gauss, in order to propose a fully homogenous algebraic model of Euclidean space [70, 74, 95]. These texts demonstrate that adding a Minkowskian metric tensor to a Euclidean subspace enables one to work algebraically with flat and round elements such as lines, circles, planes, and sphere, as well as encoding all *conformal* (angle-preserving) transformations of these subspaces as *orthogonal* transformations in the higher dimension – essentially through compositions of reflections in hyperplanes. Their system encapsulates the geometric properties of a Euclidean space and the covariant morphisms of its subspaces exceptionally well. Thus was born conformal geometric algebra: an expressive mathematical syntax for describing spatial relationships which we explore in Section 2.4.

With this truly Leibnizian mechanism for articulating machines in place, researchers assumed the challenge of working out the full expressivity of the conformal system. Timothy Havel explored molecular modelling with distance geometry [132, 63]. Leo Dorst published texts on its use in rigid body mechanics and wrote a comprehensive textbook with Stephen Mann and Daniel Fontijne on its use in computer science [45]. Eduardo Bayro-Corrochano, a cyberneticist and roboticist, extracted the use of the motor algebra in generating kinematic chains, and used the primitives to construct contraint and control systems for machine learning and computer vision [9, 10, 7]. Richard Wareham, Jonathan Cameron, and Joan Lasenby developed a plethora of useful geometric computing techniques from computer vision to computer graphics [134]. Christian Perwass developed engineering applications along with CluViz, a popular visualization engine of the conformal model [111, 112]. In 2004, Anthony Lasenby outlined uses of the conformal model in conical transformations and alternative geometries[90]. Anthony Lasenby, Chris Doran, and Stephen Gull delved deep into physics applications, including a new gauge theory of gravity [62, 89, 39]. [14]

Within the community of computer graphics practitioners of GA, the conformal model of 3D space is currently the most widely used due to its superior representation of Euclidean transformations and primitives, as well as its *universal* capacity to represent non-Euclidean geometries. Because objects in the algebra transform covariantly, Leo Dorst and his co-authors have repeatedly emphasized the value in using geometric algebra's primitives as building blocks ("You should learn to take advantage of these new possibilities of CGA, by daring to define your shapes as more than simply a mesh of points." [44]p36).

The most common applications of geometric algebra in computer graphics include modelling transformations [40], forward and inverse kinematics for robotics[10, 9, 4, 3], image processing for computer vision [7, 134], raytracing and camera navigation for scene rendering [45, 92], and mesh deformation [13] and point cloud analysis for geometry processing [41, 76, 123]. Such research demonstrates to great effect the advantages of the mathematics when used as an analysis layer over existing shapes and scenarios.

Examples of raw synthesis include the loxodromic Möbius transformations pictured by Dorst, Mann and Fontijne in [45] and the conformal orbits and knots of Dorst and Valkenburg in [46]. Doran has explored the use of linear circle and sphere blending to create curves and surfaces [38]. Krasauskas and Zube develop bezier-like forms and surface patches [87, 86]. Bell has examined surface generation through interpolation of circles on his website [12]. The current author has experimented with deforming primitive surfaces in [28] and [29]. Falcon-Morales and Bayro-Corrochano picture ruled surfaces in [49] and in Chapter 6 of [7]. Sommer, Rosenhahn, and Perwass have categorized the use of screw motions to generate shapes in [127]. Wareham, Cameron, and Lasenby have suggested the interpolation of motors as a potential way to express form in [134] and demonstrated making conics out of the conformal model. Crystallographic space groups have been worked out algebraically

---

[14] All this geometric activity coincided with continued explorations of the analytic component of Clifford Analysis, which extrapolates complex signal processing into the quaternionic domain and beyond. Such applications, while rich in their own right, are not investigated here.

by Hestenes and Holt in [73, 71]. An implementation for examining the transformations themselves was written by Hitzer and Perwass [79].

The results of the above works make it clear that the practice of digital synthesis with geometric algebra is promising field of research that demands more investigation. All these authors articulate the benefits and beauty of geometric algebra exceptionally well. Thanks to the efforts of these researchers and others, we can now begin our practice.

Chapter 2

Geometric Algebra

> The tools of mathematics were invented, not discovered.
>
> – David Hestenes

2.1 Summary

In this section we outline the mathematical construction of geometric algebra, beginning with defining a basis over a vector space and ending with the 5D conformal mapping of 3D Euclidean geometry. The conformal model allows us to express Euclidean primitives and transformations in a compact way. We will restrict ourselves to the key elements of geometric algebra necessary to keep our treatment self-contained, and look at some algebraic proofs just to get a flavor of the articulation. In the process, we will construct dual representations of **primitives** such as points, planes, spheres, lines and circles, and explore how they can combine. We will also generate **transformations** with which to transform those primitives, such as rotations, translations, dilations, and boosts.

Our computer implementation of this system can be found in Appendix A and online at

`github.com/wolftype/versor`

where, because we encode the generic nature of the algebra, our implementation is not limited to the conformal model presented here. A more in-depth look at the algebra and proofs of the constructions can be found in the references of Section 1.6. Other points of entry include the excellent collection of articles discussing the conformal model of geometric algebra in computer science [45, 40, 134, 92, 49, 76].

2.2 Beginnings

2.2.1 Orthogonal Transformations

An **orthogonal transformation** is a linear function f such that for any two vectors a and b

$$a \cdot b = f(a) \cdot f(b) \tag{2.1}$$

preserving the inner product and therefore preserving both distances and angles. In Euclidean spaces these transformations are reflections and rotations and combinations thereof. An **orthogonal group** is the set of all such transformations acting on a vector space, and in geometric algebra this group is generated by vectors themselves. In fact all the transformations detailed throughout this document are orthogonal, and all can be constructed through the linear **geometric product** of vectors. Such transformations are called **versors**. Versors do more than preserve the inner product between elements on which they operate: they preserve the outer product and are therefore **structure-preserving**. Our treatment of synthesis techniques relies on this structure-preserving mechanism for combining and transforming geometric elements under algebraic manipulation, as it allows for flexibility in the design process.[1]

Conformal geometric algebra is so called because its orthogonal group of transforma-

---

[1] The emphasis on building elements and operators through structure-preserving multiplication rather than addition is what some authors use to distinguish the term “geometric algebra” from “Clifford algebra”. See [93] for a more rigorous discussion of the difference.

tions in $n+2$ dimensions is **isomorphic** to the conformal group of transformations in n dimensions. The conformal group includes all transformations that preserve angles – distances need not be preserved. In Euclidean spaces of dimension 3 and above this group is equivalent to the Möbius transformations: reflections in spheres. Essentially, the conformal geometry of 3D Euclidean space is encoded by the structure-preserving versors of a 5D vector space.

We begin by considering Euclidean spaces.

2.2.2 Canonical Basis

An n-dimensional **vector space** $\mathscr{V}^n$ is spanned by n linearly independent elements $\{e_n\}$ which can be defined over a ground field of real numbers $\mathbb{R}$ in which case it is a **real vector space**. Endowing a real vector space with a real-valued **metric** $\langle v, v\rangle = |v|^2$ for $v \in \mathscr{V}^n$ defines a **Euclidean space** $\mathbb{R}^n$. These vector elements are closed under addition and scalar multiplication. Multiplication of the vector elements, however, adds additional algebraic structure to form an associative geometric algebra $\mathscr{G}^n$ with 2^n linear subspace elements.

The added multiplicative structure emerges from a single axiom which defines a metric-preserving quadratic product such that $v^2 = \langle v, v\rangle = |v|^2$ for any vector v. This axiom induces the algebra in the following way: consider $v = \alpha e_1 + \beta e_2$ in $\mathscr{G}^2$, with $\alpha, \beta \in \mathbb{R}$ and a positive metric on its linear elements $\langle e_i, e_i\rangle = 1$ for $i = 1, 2$. The metric product is then

$$\langle v, v\rangle = \alpha^2 + \beta^2 \tag{2.2}$$

and the quadratic product is

$$v^2 = (\alpha e_1 + \beta e_2)^2 = \alpha^2 e_1 e_1 + \beta^2 e_2 e_2 + \alpha\beta(e_1 e_2 + e_2 e_1). \tag{2.3}$$

For equations 2.2 and 2.3 to be equal requires that

$$e_i^2 = \langle e_i, e_i\rangle = 1, \tag{2.4}$$

which we assumed, and that for $i \neq j$

$$e_i e_j = -e_j e_i, \tag{2.5}$$

thus inducing an anti-symmetric non-commutative property. We will find it useful to write

$$e_i e_j = e_{ij} \tag{2.6}$$

and note that such a primary unit element is also a member of the basis of the algebra, where the number of subscripts specifies the **grade**, or dimension, of the element. This building-up of higher grades from lower ones is the cornerstone of **extension** algebras, and the wedge symbol, $\wedge$, specifies such an exterior or **outer** product.

$$e_i \wedge e_j = \begin{cases} e_{ij} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{2.7}$$

The geometric meaning of this algebraic operator is expressed in Figures 2.1, 2.2, and 2.3, where the outer product is associated with increasing dimensions or **spanning** a subspace.

The inner product is associated with decreasing dimensions or **contracting** a subspace. and can be written as

$$e_i \cdot e_j = m_i \delta^i_j \tag{2.8}$$

where δ is the **kronecker-delta** function:

$$\delta^i_j = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \tag{2.9}$$

and where m_i is defined by the metric tensor of the embedding. While the metric is positive for all bases in Euclidean vector spaces $\mathbb{R}^n$, it can also be negative or zero, and this is noted

grade	blades	geometry	interpretation
0	1	scalar	magnitude
1	e_1, e_2, e_3	vector	directed magnitude
2	e_{12}, e_{13}, e_{23}	bivector	directed area
3	e_{123}	trivector	directed volume

Table 2.1: Basis blades of $\mathscr{G}^3$

by the full $\{p, q, r\}$ **signature** of the field $\mathbb{R}^{p,q,r}$, with p bases squaring to 1, q squaring to -1, and r squaring to 0. [2]

For Euclidean spaces $\mathbb{R}^n$ all m_i are equal to 1 and we can write

$$e_i \cdot e_j = \delta^i_j. \tag{2.10}$$

2.2.3 Basis Blades

We call primary unit elements **basis blades** and say the n elements $\{e_i | i = 1, 2...n\}$ which span the space of $\mathscr{V}^n$ have a dimension or **grade** of 1, whereas elements such as e_{12} have a grade of 2. In general an n-dimensional space admits elements of grades $[0, n]$, where the 0-graded element is a 0-dimensional **scalar** value and the n-graded element is an n-dimensional **pseudoscalar**.

In an n-dimensional vector space there are $\binom{n}{k}$ basis blades of grade k, where $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ is the binomial coefficient. For instance, in $\mathscr{G}^3$ there is 1 blade of grade 0, 3 blades of grade 1, 3 blades of grade 2, and 1 of grade 3 (see Figure 2.1).

Altogether, the 2^n elements comprise the complete set of basis blades of $\mathscr{G}^n$. Vectors defined as linear sums of e_i are considered 1-blades and are written in lowercase letters, as in $a = a_1 e_1 + a_2 e_2 + a_3 e_3$. 2-blades can be created by wedging together two 1-blades – indeed, this is the definition of a 2-blade. In general, *a k-blade is an element that can be*

---

[2] The metric can be arbitrary, though we assume the $\{e_i\}$ to be linearly independent and orthogonal. For instance Space Time Algebra uses $\mathbb{R}^{1,3}$, and Conformal Geometric Algebra uses $\mathbb{R}^{4,1}$. Grassmann, who first developed the formal geometric system, used no metric and was primarily interested in *extension* and not *contraction*. This is equivalent to setting $e_i \cdot e_j = 0$ for all $i = j$.

expressed (factorized) as the outer product of k 1-blades. It is important to note – especially when dealing with the logarithms of section 2.4.5 – that in dimensions higher than 3 sums of basis blades can result in forms which, while composed entirely of k-basis blades, are not themselves a k-blade precisely because they are not factorable as the outer product of k 1-blades. The categories of such general forms are typically called bivectors, trivectors, or quadvectors, etc. Thus 2-blades, 3-blades, and 4-blades are a subcategory of these more general types.

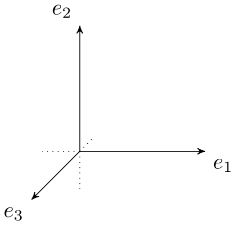


Figure 2.1: Basis 1-blades e_1,e_2, and e_3 in $\mathscr{G}^3$ represent directed magnitudes (e.g. x, y and z). Linear combinations of these basis blades define a vector: $v = \alpha e_1 + \beta e_2 + \gamma e_3$.

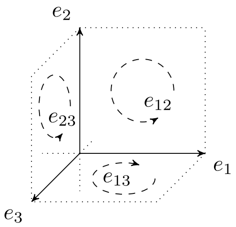


Figure 2.2: Basis 2-blades e_{12},e_{13}, and e_{23} in $\mathscr{G}^3$ represent directed unit areas. Linear combinations of these basis blades define a bivector: $B = \alpha e_{12} + \beta e_{13} + \gamma e_{23}$.

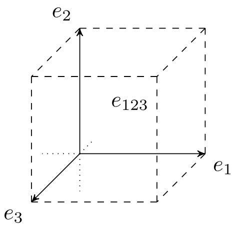


Figure 2.3: The basis trivector e_{123} in $\mathscr{G}^3$ is also known as the pseudoscalar I. As the highest grade, blade I is sometimes referred to as the volume element or tangent space. Multiplying elements by I^{-1} (i.e. dividing out the pseudoscalar) returns their *dual* represention. In $\mathscr{G}^3$ the bivectors and vectors are dual to each other: $BI^{-1} = v$. Typically one writes $B^{\star} = v$. See also Section 2.2.5.

2.2.4 Geometric Product

Let us now apply our above definitions to two different vectors $a = \{a_i e_i\}$ and $b = \{b_i e_i\}$ in $\mathscr{G}^3$:

$$ab = (a_1e_1 + a_2e_2 + a_3e_3) * (b_1e_1 + b_2e_2 + b_3e_3)$$

$$= a_1(b_1e_1e_1 + b_2e_1e_2 + b_3e_1e_3) + a_2(b_1e_2e_1 + b_2e_2e_2 + b_3e_2e_3) + a_3(b_1e_3e_1 + b_2e_3e_2 + b_3e_3e_3)$$

$$= a_1(b_1 + b_2e_{12} + b_3e_{13}) + a_2(-b_1e_{12} + b_2 + b_3e_{23}) + a_3(-b_1e_{13} - b_2e_{23} + b_3)$$

$$= (a_1b_1 + a_2b_2 + a_3b_3) + (a_1b_2 - a_2b_1)e_{12} + (a_1b_3 - a_3b_1)e_{13} + (a_2b_3 - a_3b_2)e_{23}$$

which reveals that our product encapsulates *both* the typical inner ("dot") and outer ("cross") products of vectors. We call this associative quadratic operation, with both commutative and anti-commutative components, the **geometric product**. For vectors a, b, and c:

$$(ab)c = a(bc) \tag{2.11}$$

$$a \wedge b = -b \wedge a \tag{2.12}$$

and we note that it is decomposable into an inner and outer product:

$$ab = a \cdot b + a \wedge b \tag{2.13}$$

The result of this expression has *mixed* grade – a scalar value of grade-0 and a bivector value of grade-2. It is because of the possibility of mixed grade elements that general elements of the algebra are called **multivectors**.

Traditionally, the dot product of two vectors has been known to represent the cosine of the angle between them, while the norm of the cross product is the sine of that angle. This relationship is preserved in the geometric product

$$ab = a \cdot b + a \wedge b = \cos\theta + B\sin\theta \tag{2.14}$$

where $B = a \wedge b / \|a \wedge b\|$ is the normalized, unit-area spanned by a and b, and where θ is the angle between them. This equivalence between the geometric product and the trigonometric functions is critical in formulating a general theory of continuous transformations from the identity, relating the geometric product to the exponential via Euler's formula. This feature is investigated in Section 2.3.5.

With these basis elements and within our additional structure we can extend our n bases to build a combinatoric group of 2^n bases with a variety of grades ranging from $[0,n]$.

Subscript Manipulations

The rules for combining elements are therefore threefold:

1. Concatenate subscripts (e.g. $e_1 * e_2 = e_{12}$)
2. Negate shuffled subscripts (e.g. $e_{21} = -e_{12}$)
3. Reduce doubled subscript to a scalar value[3] (e.g. $e_{11} = e_{33} = 1$)

Thus in $\mathscr{G}^3$ we can write:

$$e_{12} * e_{123} = e_{12123} = -e_{223} = -e_3$$

More generally, the e_{ii} of rule 3 above would be replaced by the tensor metric value m_i (most often 0, 1, or -1).

The concatenation (rule #1) stems from the fact that basis elements can be "wedged" together with the outer product to build higher graded elements. Thus in $\mathscr{G}^3$ where we have three 1-dimensional blades $b = \{e_1, e_2, e_3\}$, we can also define $e_1 \wedge e_2 = e_{12}$ where e_{12} is a basis blade of grade-2. e_{13} and e_{23} are similarly defined and together form a basis $B = \{e_{12}, e_{13}, e_{23}\}$ of **bivectors**. The wedge constructions continue, with $e_1 \wedge e_2 \wedge e_3 = e_{123}$ the grade-3 **trivector** blade.

---

[3] Or, if not a real scalar value, then we reduce to whatever the *ground field* is, as one can define the algebra over a field of other kinds of numbers as long as they form a commutative ring.

$$e_i \wedge e_j = \begin{cases} e_{ij} & i \neq j \\ 0 & i = j \end{cases} \tag{2.15}$$

2.2.5 Duality

Since there is only one n-blade in any n-dimensional space, we call it the **pseudoscalar** and denote it as I.

$$I = \bigwedge_{i=1}^{n} e_i \tag{2.16}$$

The pseudoscalar I has the useful property of **dualizing** any blade that is multiplied by it: turning a blade of grade k into a blade of grade $n-k$ with an equal number of basis elements. This is written in shorthand as a star operator for any multivector member A of the algebra:

$$A^{\star} = AI^{-1} \tag{2.17}$$

where the **inverse** is defined by normalizing the reverse:

$$A^{-1} = \tilde{A}/(A\tilde{A}) \tag{2.18}$$

and the **reverse** is found by reversing the order of subscripts:

$$\widetilde{e_{123}} = e_{321} = -e_{312} = e_{132} = -e_{123}. \tag{2.19}$$

An **undualization** is achieved by *not* inverting the pseudoscalar:

$$A^{-\star} = AI \tag{2.20}$$

which is useful when seeking the multivector to which A acts as dual so that $(A^{-\star})^{\star} = A$ (as, for instance, in Equation 2.25 below).

The reverse is used to define the (reverse) **norm**, defined for a multivector X as:

$$\|X\| = \begin{cases} \sqrt{X \cdot \tilde{X}} & \text{if } X^2 \geq 0 \\ -\sqrt{-X \cdot \tilde{X}} & \text{if } X^2 < 0 \end{cases} \tag{2.21}$$

A few other unary operations are useful: **conjugation** and **involution**. In general for a blade X of grade k

$$\text{Reversion:} X \mapsto \tilde{X} = (-1)^{k(k-1)} X \tag{2.22}$$

$$\text{Involution: } X \mapsto \hat{X} = (-1)^{k} X \tag{2.23}$$

$$\text{Conjugation: } X \mapsto \bar{X} = (-1)^{k(k+1)} X \tag{2.24}$$

2.2.6 The Meet

An essential component of calculations in geometric algebra is that of the **meet** of two blades, which is defined algebraically as the undualized outer product of duals. For bivector elements A and B we can find their coincident axis v by:

$$v = (A^{\star} \wedge B^{\star})^{-\star} \tag{2.25}$$

This simple formulation will be particularly useful once our blades represent more interesting geometric primitives (such as spheres).

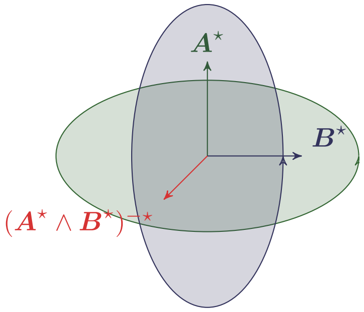


Figure 2.4: Meet of two Blades A and B determined by undualizing the outer product of duals: $(A^\star \wedge B^\star)^{-\star}$

2.3 Transformations

We now investigate the geometric product as a spatial operation. The existence of a geometric product allows us to define an inverse unitary operator on vectors, which in turn enables the notion of **ratios** of vectors. These relationships provide a unified treatment of continuous transformations (as well as the assembling of the apparatus of differential geometry with geometric calculus).

2.3.1 Versors

The definition of the inverse in Equation 2.18 works for any **versor**. A versor is any multivector A that can be expressed as an outer product of n vectors, which we call an n-blade, or any geometric product of such n-blades. Not only are versors invertible, but due to the associativity of the geometric product, their *left* inverse and *right* inverse are the same: $AA^{-1} = A^{-1}A = 1$. Also for A's composed of the product of *even* number of unit 1-blades, the inverse is the reverse: $A^{-1} = \tilde{A}$, a fact which helps optimize computer implementations of the transformations of this section.

2.3.2 Reflection

Given two vectors a and b we can decompose a into a sum of vectors projected onto (parallel to) and rejected from (orthogonal to) b.

$$a = a_{\parallel} + a_{\perp} \tag{2.26}$$

The geometric product of b with a can then be distributed accordingly:

$$\begin{aligned} ba &= b(a_{\parallel} + a_{\perp}) \\ &= b \cdot a_{\parallel} + b \wedge a_{\parallel} + b \cdot a_{\perp} + b \wedge a_{\perp} \\ &= b \cdot a_{\parallel} + b \wedge a_{\perp} \\ &= a_{\parallel} \cdot b - a_{\perp} \wedge b \\ &= (a_{\parallel} - a_{\perp})b \end{aligned}$$

which holds because both $b \wedge a_{\parallel}$ and $b \cdot a_{\perp}$ are 0 by definition. If we multiply both sides by b^{-1}, forming a "sandwich" on the left we get

$$\begin{aligned} bab^{-1} &= (a_{\parallel} - a_{\perp})bb^{-1} \\ &= a_{\parallel} - a_{\perp} \end{aligned}$$

and negating this gives

$$-bab^{-1} = -a_{\parallel} + a_{\perp} \tag{2.27}$$

which, as depicted in Figure 2.5, encodes a **reflection** of a over the bivector plane that b is normal to. Typically we use a b of unit length, such that $b^{-1} = b$ and equation simplifies to $-bab$. We note that this powerful spatial syntax that uses algebraic mechanisms to relate

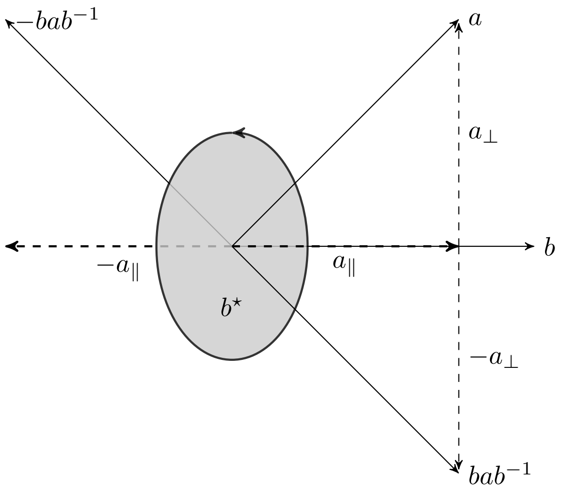


Figure 2.5: Algebraic anatomy of a reflection of a over the bivector $b^\star$ dual to b.

geometric elements stems from our ability to invert a blade.

This ability to model a geometric transformation without any reference to coordinates or dimension gives breath to our algebra. We will soon see that *all* transformations are built this way, using variants of the fundamental sandwich operation. The “sandwich” product $-bab^{-1}$ is referred more formally as the **versor product.**

A further generalization of this reflection operation involves calculating the involution of the element being reflected in order to determine whether a negative is necessary. To reflect a multivector A by a versor X:

$$A \mapsto A^{'} = X\hat{A}X^{-1} \tag{2.28}$$

2.3.3 Projection and Rejection

We were able to use algebraic mechanisms to create a reflection operation in a vector space by acknowledging the decomposition of one vector in terms of another. In particular we use it to define a vector a’s **projection** onto and **rejection** from another blade B.

$$\text{Projection: } (a \cdot B)B^{-1} \tag{2.29}$$

$$\text{Rejection: } (a \wedge B)B^{-1} \tag{2.30}$$

If a is a vector and B a Euclidean bivector, the projection of a onto B is a vector in the plane of B and the rejection is a vector orthogonal to it. If both blades are vectors, denoted a and b, the projection operation then is a scaling of b while the rejection is a scaling and rotation of b by 90 degrees in the $a \wedge b$ plane.

2.3.4 Rotations

If we take the reflection-generating versor product $-bab$ and apply another reflection operation to it of the form $-c(-bab)c = cbabc$ then this composite reflection generates a rotation in the common plane of b and c through twice the angle from b to c.

Because the product cb can be used to generate a rotation, this composition of unit length vectors is type of versor called a **rotor** in the geometric algebra literature[4], and is usually denoted R. As demonstrated in Figure 2.6, a rotor $R = cb$ generates a rotation in the direction of b to c in their common plane and twice their angle. It is applied to some vector a as a versor product:

$$a \mapsto Ra\tilde{R} \tag{2.31}$$

where $\tilde{R}$ is the reverse bc and $R\tilde{R} = 1$. We will often notate the transformation under this versor product in brackets as $a \mapsto R[a] = Ra\tilde{R}$ when convenient.

2.3.5 The Remarkable Exponential

As noted, we say orthogonal transformations like reflections and rotations are versors, and we call the invertible elements used to create them **generators**. Here geometric algebra reveals

---

[4] The term *rotor* dates back to Clifford's original treatment of geometric algebra, who coined it as a conjunction of "rotation vector".

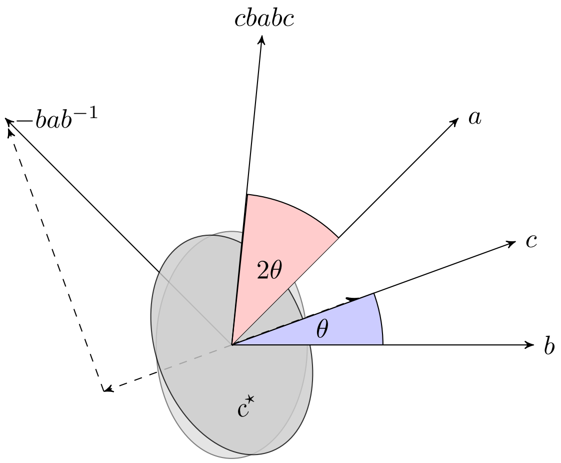


Figure 2.6: A rotation is composed two reflections. Vector a is reflected by b and the result then reflected by c to create $-(c(-bab^{-1})c^{-1})$. The signs cancel, and the expression can be further reduced to $cbabc$ if b and c are unit length. The nature of this double reflection is such that the angle between a and $cbabc$ is exactly twice the angle between b and c. Note that order matters: $bcacb$ would generate a rotation in the opposite direction.

its strong ties to Lie algebras and Lie groups, which uses similar terms. As we will see, a variety of transformations can be constructed, and it helps to categorize them by whether they are composed from an even number or odd number of 1-blades. Throughout the literature, even versors are called rotors or spinors, and they have the ability to be *decomposed* using an extension of the "remarkable formula"[5] of Euler:

$$e^{i\theta} = \cos\theta + i\sin\theta \tag{2.32}$$

where $i^2 = -1$, θ is any real number and e is Euler's number (and is not to be confused with our basis blades e_1 etc). Euler's formula greatly facilitates algebraic manipulations of complex transforms. Recalling that our unit 2-blades (e_{12}, e_{13}, and e_{23} in $\mathscr{G}_3$) also square to -1, we might hope that for a unit 2-blade B:

$$e^{B\theta} = \cos\theta + B\sin\theta \tag{2.33}$$

---

[5] Deemed so by Richard Feynman, Euler's formula is $e^{i\pi} + 1 = 0$, which can be seen to emerge from equation 2.32.

and this is indeed the case. In fact, what we have called the "imaginary" number i can be simply recast as e_{12} in $\mathscr{G}^2$, revealing that a complex number is a type of rotor or spinor, embedded in the even subalgebra of 2 dimensional Euclidean geometric algebra.[6] *Quaternions*, rotation transformations, are another type of rotor or spinor embedded in the even subalgebra of 3-dimensional geometric algebra: given a 2-blade B, equation 2.33 generates a rotation around the axis normal to it. Because B is a directed area plane, we talk about rotations in terms of the plane in which rotation occurs, rather than in terms of the axis. Of course in $\mathscr{G}^3$ these are dual notions however the plane-centric approach enables us to conceptualize rotations in higher dimensions, such as in $\mathscr{G}^{4,1}$.

When applied to an element in the sandwich form of equation 2.31, $e^{B\theta}$ generates a rotation in the direction opposite the orientation of B through twice the angle of θ

$$v \mapsto e^{B\theta} v e^{-B\theta} = R_B v \tilde{R}_B = R_B[v] \tag{2.34}$$

where to emphasize the plane of rotation we will sometimes write R_B. Since in the above equation the resulting rotation will be negative twice the value θ, we often write:

$$v \mapsto e^{-\frac{\theta}{2}B} v e^{\frac{\theta}{2}B} \tag{2.35}$$

to specify a rotation of v by θ in the plane of B.

Furthermore, of relevance when we discuss the conformal model, *any* 2-blade in Euclidean or Minkowski spaces can be exponentiated, and general bivectors can be split into commuting 2-blades $B+$ and $B-$. Any continuous transformation is thereby encoded as a rotation in some hyperplane of these spaces. An example of this decomposition is outlined in Section 5.3.6, as detailed in [46], following a result of [75].

Trigonometry is strictly relegated to this notion of the exponential of a bivector. In the GA literature, pages of classical trigonometric matrix deductions are replaced by these simple

---

[6] This geometric interpretation of i at least partially motivates the title of Gull, Lasenby, and Doran's physics paper *Imaginary Numbers are not Real* [62].

encodings of spin transformations. This plays a particularly helpful role in the kinematics equations of Chapter 4.

2.3.6 Rationalizing with Ratios

We can find the transformation R_{ab} that takes unit vector a to b in their common plane, following section 10.3.2 in [45]. The first thing to note in our geometric numerics is our ability to invert a and therefore to divide it out of b – conceptually this is akin to finding the transformation that takes the real number 2 to 10: we must divide $10/2$ to get the result 5. Consider solving the equation $x5x = 10$; in that case $10/5$ gives us x^2, and the solution is $x = \pm\sqrt{2}$. The versor product applies the rotor *twice*, our division of vectors gives us *twice* the transformation,

Squaring a transformation is akin to applying the transformation twice

$$R_{ab}^2 = \frac{b}{a} = ba \tag{2.36}$$

with the last equality due to $a^{-1} = a$ for a of unit length, and so

$$R_{ab} = \sqrt{ba}. \tag{2.37}$$

Finding the square root of this multivector turns out to be a matter of normalizing $1 + ba$, whose norm is defined as $\sqrt{(1+ab)(1+ba)} = \sqrt{|2(1+\langle ab\rangle)|}$

$$R_{ab} = \frac{1+ba}{\sqrt{|2(1+\langle ab\rangle)|}} \tag{2.38}$$

A "geometric motivation" for this definition of the square root as $|1+R^2|$ can be found in [46], where it is shown that this equation works for more than just Euclidean quaternionic rotors. The square root of any "simple" rotor exponential of a 2-blade can be found this way. See for instance section 5.3.5. This has profound implications when implementing this in code, as Equation 2.38 can be used to generate a rotation in *any* dimension.

2.3.7 Example Problem: Preventing Camera Roll

Let us get a feel for the use of the algebra by applying it to a basic problem in graphics computations. One common task is to orient a local *xyz frame*, such as a virtual camera at location c, towards some point p in space. This is easy enough by using equation 2.38 to find the rotation that takes the global z axis (0,0,1) to some unit target vector v, where $v = (p-c)/\|p-c\|$. However this can induce unwanted *roll* in the resulting camera orientation (see Figure 2.7). Our task is therefore to orient the camera towards our target while *also* keeping the camera's local y axis as vertical as possible. For more about frames please refer to Section 4.2.

Algorithm 2.1 Preventing Camera Roll

---

1. Construct the rotation R_1 that takes e_3 vector to the (unit) target direction v (equation 2.38).
2. Construct the transformed plane xy' by applying rotor R_1 to e_{12}. Also find the transformed vector y' by applying rotor R_1 to e_2 (equation 2.31).
3. Construct the target y_p local "up" direction by projecting e_2 onto transformed xy' bivector (equation 2.29).
4. Construct the rotation R_2 that takes y' to y_p (equation 2.38).
5. Compose the rotation sequence $R = R_2R_1$.

Step 3 of Algorithm 2.1 is the key here, as it takes an ideal "up" direction e_2, and projects it onto the transformed plane xy' representing all possible "up" directions in which any rolling might happen, returning an adjusted local $y$axis that is as vertical as possible. We note that in step 5, the rotation R_1 followed by R_2 is composed as R_2R_1. Transformations are applied in the reverse order in which they are multiplied (consider Figure 2.6, where cb encodes a reflection in b followed by a reflection in c).

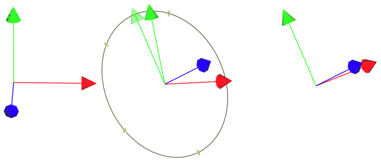


Figure 2.7: A frame is rotated to point its z (blue) axis towards a target, inducing an unwanted roll about the z axis. Absolute y (e_2) is projected onto its transformed $x \wedge y$ plane. The frame is then rotated again to keep its y (green) axis as vertical as possible.

2.3.8 Other Metrics

In addition to $\mathbb{R}^3$ and $\mathbb{R}^{4,1}$(which we explore next in Section 2.4), a few other metrics have been deemed worthy of closer study.

$\mathbb{R}^{1,3}$ Much of the modern day interest in geometric algebra stems from a 1966 publication by David Hestenes called *Space-Time Algebra* [65], which explores spinors in the space of four dimensions to derive a matrix-free approach to quantum theory. As specified by the signature of that space, there is one time dimension which squares to 1 and three space dimensions which square to -1. In addition to this classic text, readers are encouraged to take a look at Gull, Lasenby and Doran [62] for a brief introduction to particle physics with geometric algebra, as well as Doran and Lasenby's textbook [39].

$\mathbb{R}^{3,3}$ Recent explorations into the encoding of projective transformations as versor products have been made by the geometric algebra community, notably in [94] and [42].

$\mathbb{R}^{4,2}$ Lie Sphere Geometry, which we touch upon at the end of Chapter 6, has been proposed as an even more unifying way to deal with 3D surfaces, for instance by Krasauskas in [86] and Bobenko in a non-GA setting in [17].

$\mathbb{R}^{2,4}$ Anthony Lasenby has explored a variety of metrics for cosmology, including this conformalized space-time metric [88, 90].

2.4 Conformal Geometric Algebra

Amongst the most commonly used metrics, $\mathbb{R}^{4,1}$ has garnered the most attention due to a result discovered by Wachter in the 1800s and implemented by Hestenes with the language of geometric algebra: the complete conformal group of transformations in $\mathbb{R}^n$ is encoded by the orthogonal group of $\mathbb{R}^{n+1,1}$. This means orthogonal transformations – versors – in this 5D space preserve *angles* between 3D elements. The preservation of this property signifies a **conformal** transformation and in 3D these are sometimes referred to as Möbius transformations. $\mathbb{R}^{4,1}$ is called the *conformal model* of $\mathbb{R}^3$, and the geometric algebra of this space is known as *conformal geometric algebra*. For more details on this construction please see references in the summary at the beginning of this chapter and in Section 1.6, especially [70, 74, 95].

2.4.1 Conformal Metric

The metric of a space determines the properties of the inner product. Consider a 2-dimensional space where one of the basis blades squares to -1 instead of 1. This is known as a Minkowski metric of signature $\mathbb{R}^{1,1}$, and the basis are often labelled e_- and e_+ (or sometimes alternatively as $\bar{e}$ and e).

$\cdot$	e_+	e_-
e_+	1	0
e_-	0	-1

Table 2.2: Minkowski Metric of $\mathbb{R}^{1,1}$

One can then create a **null** basis by defining two new basis elements n_o and n_∞ in terms of e_+ and e_- such that the new elements square to 0 (see Figure 2.9).

$$n_\infty = e_- - e_+ \qquad\qquad n_o = \frac{e_- + e_+}{2} \tag{2.39}$$

$$e_+ = n_o - \frac{n_\infty}{2} \qquad\qquad e_- = n_o + \frac{n_\infty}{2} \tag{2.40}$$

This is easily verified:

$$\begin{aligned} n_\infty^2 &= (e_- - e_+)^2 = -1 + 1 - e_-e_+ - e_+e_- = 0 \\ n_o^2 &= (\frac{1}{2}(e_- + e_+))^2 = -\frac{1}{4} + \frac{1}{4} = 0 \end{aligned}$$

The weighting is such that their inner product is -1:

$$n_\infty \cdot n_o = (e_- - e_+) \cdot (\frac{1}{2}(e_- + e_+)) = -\frac{1}{2} - \frac{1}{2} = -1$$

which holds regardless of the order.

When added to Euclidean metrics $\mathbb{R}^n$ to create $\mathbb{R}^{n+1,1}$, these characteristics of the null basis of $\mathbb{R}^{1,1}$ enable a rich set of features such as the ability to disambiguate between vectors and points, represent distances of points with the inner product, and represent geometric primitives with the outer product. The inner product table for $\mathbb{R}^{4,1} = \mathbb{R}^3 \oplus \mathbb{R}^{1,1}$ is listed in Table 2.4. This definition of $\mathbb{R}^{4,1}$ entails it being split into Euclidean and Minkowskian metric components.

$\cdot$	e_1	e_2	e_3	e_+	e_-
e_1	1	0	0	0	0
e_2	0	1	0	0	0
e_3	0	0	1	0	0
e_+	0	0	0	1	0
e_-	0	0	0	0	-1

Table 2.3: A nondegenerate basis for $\mathbb{R}^{4,1}$.

$\cdot$	n_o	e_1	e_2	e_3	n_∞
n_o	0	0	0	0	-1
e_1	0	1	0	0	0
e_2	0	0	1	0	0
e_3	0	0	0	1	0
n_∞	-1	0	0	0	0

Table 2.4: A degenerate null basis for $\mathbb{R}^{4,1}$.

2.4.2 Blades

The addition of two new basis blades generates other blades besides scalars, vectors, bivectors, rotors and the pseudoscalar. For example, e_1n_o, e_2n_o, and e_3n_o are now basis elements, as are e_1n_∞, e_2n_∞, and e_3n_∞, and indeed all combinations of the Euclidean subalgebra with its new friends n_o and n_∞. Table 2.5 lists the complete set of 32 basis elements in the conformal model of $\mathbb{R}^{4,1}$ ($2^5 = 32$ basis elements), and the geometric concepts which they encode.

Before we explore these geometric concepts, a few more algebraic details should be pointed out. We write the basis element $E = n_o \wedge n_\infty$ rather than n_on_∞ since the geometric product of these basis elements is of mixed grade:

$$n_on_\infty = n_o \cdot n_\infty + n_o \wedge n_\infty = -1 + E \tag{2.41}$$

Note as well that $E = e_+e_-$ since

$$\begin{aligned} E &= n_o \wedge n_\infty \\ &= \frac{1}{2}(e_- + e_+) \wedge (e_- - e_+) \\ &= \frac{1}{2}(-e_- \wedge e_+ + e_+ \wedge e_-) \\ &= e_+ e_-. \end{aligned} \tag{2.42}$$

Whereas the Euclidean bivector basis e_{ij} give a negative square, some of our new bivector bases, such as $e_i n_\infty$ and $e_i n_o$ square to 0, and $E^2 = 1$:

$$\begin{aligned} E^2 &= (n_o \wedge n_\infty)^2 \\ &= (e_+ e_-)^2 \\ &= e_+ e_- e_+ e_- \\ &= -e_+ e_+ e_- e_- \\ &= 1. \end{aligned} \tag{2.43}$$

That a bivector B can square to a negative, positive, or null value leads to variations of the expansion of e^B from Equation 2.33. We will show in section 2.4.4 (namely, with Equation 2.62) how these variations lead to different types of rotors, by providing a more generalized Euler formula that works in higher dimensions.

For the rest of this document, in order to distinguish between purely Euclidean variables such as $\mathbf{v} = \alpha e_1 + \beta e_2 + \gamma e_3$ and variables of the conformal space such as $\kappa = \alpha e_1 n_o + \beta e_2 n_o + \gamma e_3 n_o$, we write the former in bold, as is the convention in many geometric algebra texts.

grade	basis
0	1
1	$\underbrace{\overbrace{n_o, \underbrace{\overbrace{e_1, e_2, e_3,}^{\text{Euclidean Vector}} \quad n_\infty}_{\text{Dual Plane}}}^{\text{Point}}}_{\text{Dual Sphere}}$
2	$\overbrace{\underbrace{\overbrace{e_{12}, e_{13}, e_{23},}^{\text{Euclidean Bivector}} \overbrace{e_1 n_\infty, e_2 n_\infty, e_3 n_\infty}^{\text{Direction Vector}}}_{\text{Dual Line}}, \overbrace{e_1 n_o, e_2 n_o, e_3 n_o,}^{\text{Tangent Vector}} \quad \underbrace{E = n_o \wedge n_\infty}_{\text{Flat Point Origin}}}^{\text{Point Pair}}$
3	$\overbrace{\overbrace{e_{123}}^{\text{Euclidean Pseudoscalar}} \quad , \overbrace{e_{12} n_o, e_{13} n_o, e_{23} n_o}^{\text{Tangent Bivector}}, \underbrace{\overbrace{e_{12} n_\infty, e_{13} n_\infty, e_{23} n_\infty}^{\text{Direction Bivector}}, e_1 E, e_2 E, e_3 E}_{\text{Line}}}^{\text{Circle}}$
4	$\overbrace{\underbrace{e_{12} E, e_{13} E, e_{23} E, e_{123} n_\infty}_{\text{Plane}}, e_{123} n_o}^{\text{Sphere}}$
5	$I = e_{123} E$

Table 2.5: Basis blades in Conformal $\mathscr{G}^{4,1}$ and most of the geometric concepts they encode. The motivation for the many of the names stem from their behavior under various versor products. Not represented here is a Flat Point not at the origin, which is a blade of grade 2 whose basis is $\{e_1 n_\infty, e_2 n_\infty, e_3 n_\infty, E\}$.

2.4.3 Primitive Elements

The central motivation for this null-space construction is that it allows us to disambiguate between a *point* and a *vector*, and allows us to treat all orthogonal transformations in Euclidean space as versor products – including translations. This is due to the compactification of Euclidean space with an additional invariant basis n_∞. Additionally, we can now distinguish between **flat** subspaces and **round** subspaces of the algebra. Specifically, the conformal model produces a menagerie of familiar geometric elements like lines, circles, planes and spheres in the form of blades, and introduces lesser known species as well such as flat points, imaginary circles, and tangent bivectors. Table 2.5 organizes these elements according to their grade, and helps to illustrate why flat concepts, such as planes and lines, are always subsets of the more general encodings of spheres and circles which contain more information. In the conformal model, flat elements (e.g. lines and planes) are just generalized round elements (e.g. circles and spheres) that pass through infinity. Table 2.6 provides algebraic expressions for most of the useful geometric entities and operators that can be constructed with the conformal model. A useful list of binary operations of the elements can be found in the Appendix to [27].

2.4.3.1 Points as Null Vectors

To begin the construction we map a vector $\mathbf{x}$ in Euclidean space to a point p in conformal space using inverse stereographic projection (see Figure 2.8):

$$\mathbf{x} \mapsto p = n_o + \mathbf{x} + \frac{1}{2}\mathbf{x}^2 n_\infty \tag{2.44}$$

This function of the form $f : \mathbb{R}^3 \to \mathbb{R}^{4,1}$ completes our minimum definition of how to construct the conformal geometric algebra. In this representation points are called **null vectors** because they dot with themselves to return a zero measure: $p \cdot p = 0$. Since the outer product of any blade with itself is always zero, the null-ness of points is often expressed as

$$p^2 = 0 \tag{2.45}$$

This can be demonstrated algebraically:

$$\begin{aligned}
p^2 = p \cdot p &= (n_o + x + \frac{1}{2}x^2 n_\infty) \cdot (n_o + x + \frac{1}{2}x^2 n_\infty) \\
&= n_o^2 + \mathrm{x}^2 + (\frac{1}{2}\mathbf{x}^2 n_\infty)^2 + \frac{1}{2}\mathbf{x}^2(n_o \cdot n_\infty) + \frac{1}{2}\mathbf{x}^2(n_\infty \cdot n_o) \\
&= n_o^2 + \mathrm{x}^2 + (\frac{1}{2}\mathbf{x}^2 n_\infty)^2 - \mathbf{x}^2 \\
&= \mathbf{x}^2 - \mathbf{x}^2 = 0
\end{aligned}$$

which reveals a motivation for weighing[7] the n_∞ by $\frac{1}{2}\mathbf{x}^2$: the new basis blades serve to cancel out the square of the vector $\mathbf{x}$ in the Euclidean preimage.[8] This quadratic mapping defines a **horosphere** (or sometimes called horocycle in lower dimensions, or horoball in higher), which is the paraboloid embedded in $n+1$ dimensions. The term comes from Lobachevsky, who discovered this paraboloid has a natural Euclidean structure that can be used to represent non-Euclidean geometries as well (see Figure 2.8).

These **null points** of equation 2.44 have the useful property of encoding distance geometry: their dot product is minus half their squared distance from each other. If δ is the Euclidean distance between points p_a and p_b then

$$p_a \cdot p_b = -\frac{1}{2}\delta^2. \tag{2.46}$$

Finally, the outer product of points creates other primitives of Euclidean geometry. As Hestenes notes in [70], the ability to make such constructions in Euclidean space eluded

---

[7] Points themselves can be weighted by changing the value of their n_o basis vector. Note that when $n_o \neq 1$, p^2 is no longer 0.

[8] Note that alternative mappings of $\mathbf{x} \mapsto p$, and definitions of n_o and n_∞ exist throughout the literature that still satisfy the requirement that $p^2 = 0$, $n_o^2 = n_\infty^2$, and $n_o n_\infty = n_\infty n_o = -1$.

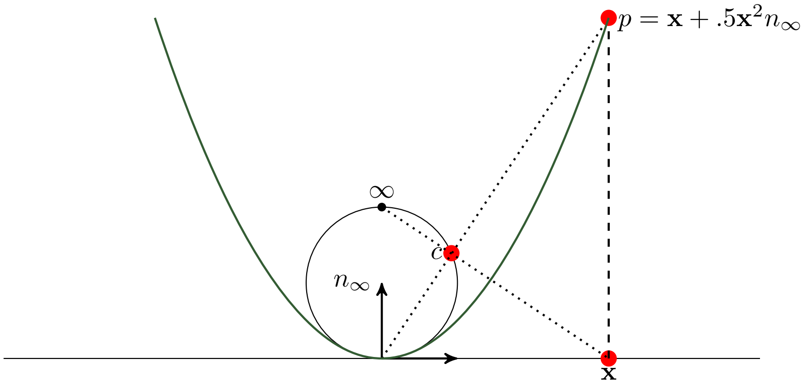


Figure 2.8: The mapping of Euclidean $\mathbb{R}^n$ onto a paraboloid one dimension higher (called the horocycle or horosphere) preserves the conformal structure of the Euclidean space. This can be illustrated geometrically by observing that lines through both $\mathbf{x}$ and p intersect the sphere S^{n+1} at the same location c, with lines drawn to north and south poles respectively. The conformal mapping $\mathbb{R}^n \mapsto S^{n+1}$, defined by extending a line from a point in $\mathbb{R}^n$ to the north pole of S^{n+1}, is called inverse stereographic projection, and is a way of compactifying the Euclidean space by adding a point representing infinity. By adding a basis in the infinity direction to our space we can encode this inverse projection as the point $p = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 n_\infty$. Adding n_0 to this formulation provides a homogeneous representation of the Euclidean subspace.

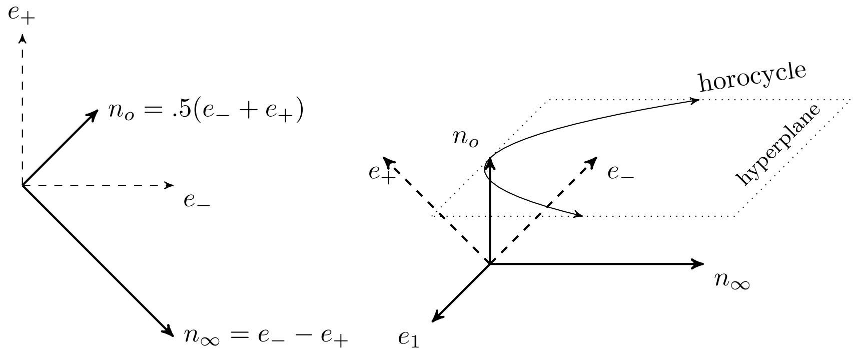


Figure 2.9: The null basis $\{n_o, n_\infty\}$ of $\mathbb{R}^{1,1}$ in terms of $\{e_+, e_-\}$. The horocycle of $\mathbb{R}^{2,1}$ is a paraboloid mapping $\mathbf{x} \mapsto p = n_o + \mathbf{x} + \frac{1}{2}\mathbf{x}^2 n_\infty$.

Grassmann (who did so only in projective space) because he had no null vector representation of points.

2.4.3.2 Direct Representation

Round elements, or n-spheres, can be expressed directly as the outer product of points. A **point pair** can be thought of as a 0-sphere – a sphere on a line with 0-dimensional boundary – and can be created by wedging two points:

$$\kappa = p_a \wedge p_b \tag{2.47}$$

Similarly, a **circle** can be created by wedging together three points:

$$K = p_a \wedge p_b \wedge p_c \tag{2.48}$$

and **sphere** by wedging together four:

$$\Sigma = p_a \wedge p_b \wedge p_c \wedge p_d \tag{2.49}$$

Flat elements can be constructed by wedging any of the above round elements with n_∞, effectively generating a blade that spans infinity. This flat element is called the **carrier** of the round element. Thus two points p_a and p_b, which define a point pair, can be wedged with n_∞ to form a **line** through them:

$$\Lambda = p_a \wedge p_b \wedge n_\infty \tag{2.50}$$

and three points, which define a circle, can be wedged with $n_\infty$to form a **plane** of the circle:

$$\Pi = p_a \wedge p_b \wedge p_c \wedge n_\infty \tag{2.51}$$

while a sphere defined as four points wedges with n_∞ to define the pseudoscalar of the space.

A single point can also be wedged with infinity to form a **flat point**.

These **direct** representations of round and flat elements are sometimes called **tangent forms** or **outer-product-null-space** (OPNS) representations. The latter terminology stems from the fact that a direct element A can be defined as the set of points which do not increase the span if wedged with it. That is $A = \{p | p \wedge A = 0, p^2 = 0\}$.

We next discuss the **dual** or **normal form** or **inner-product-null-space** (IPNS) representations A^*, for which $A^* = \{p | p \cdot A^* = 0, p^2 = 0\}$. The asterisk * signifies dualization with respect to the pseudoscalar of grade-5 (whereas the star * operator is reserved for dualization of the Euclidean subalgebra).

2.4.3.3 Dual Representation

Given a point p mapped from Euclidean vector $\mathbf{v}$ according to equation 2.44, we can subtract from the coefficient of the n_∞ blade an amount proportional to a squared radius. In this way our vectors are no longer null and can also represent spheres. A **dual sphere** σ at point p with radius r is constructed as

$$\sigma = p - \frac{1}{2} r^2 n_\infty \tag{2.52}$$

thus we can consider points as dual spheres with zero radius. In general, the square of normalized spheres returns their squared radius. This can be zero, in the case of points with no radius, or positive, in the case of **real** spheres, or negative, in the case of **imaginary** spheres. An imaginary sphere σ at point p with radius r can be constructed by adding to the n_∞ blade rather than subtracting:

$$\sigma = p + \frac{1}{2} r^2 n_\infty \tag{2.53}$$

A null point p lies on a real dual sphere σ iff $p \cdot \sigma = 0$. This is the dual definition of a sphere. Note that by this definition imaginary spheres contain no points.

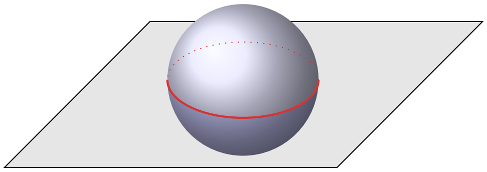


Figure 2.10: The circle intersection of a sphere and a plane.

Subtracting two points p_a and p_b returns the **dual plane** bisecting them. Such a differencing removes the n_0 blade altogether, and we can write the formula for a dual plane π with normal $\mathbf{x}$ at distance δ from the origin as

$$\pi = \mathbf{x} + \delta n_\infty \tag{2.54}$$

A **dual line** is the multivector sum of a Euclidean bivector $\mathbf{B}$ representing its orientation and a support vector $\mathbf{d}n_\infty$ indirectly encoding its distance from the origin:

$$\lambda = \mathbf{B} + \mathbf{d}n_\infty \tag{2.55}$$

A closer analysis of the components of a dual line can be found in Chapter 5, where it is used to generate a motor transformation.

Dual representations are very useful in practice for calculating incidence relationships. Given two dually represented blades, their **coincidence** (or **meet** or **intersection**) is dually represented by their outer product. For instance, a dual sphere and dual plane can be wedged together to find their circle meet (Figure 2.10).

$$K = (\sigma \wedge \pi)^{-*} \tag{2.56}$$

The direct circle (real or imaginary) where the sphere and plane intersect is the undualized outer product of duals. Similarly, two dual spheres can be wedged together to find the

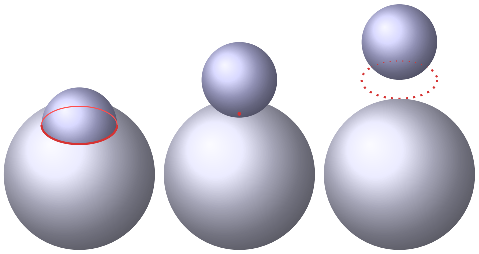


Figure 2.11: The meet (or coincidence) of two dual spheres $K = (\sigma_1 \wedge \sigma_2)^{-*}$is a real circle with positive squared radius if the spheres intersect, an infinitesimal tangent circle with zero radius if the spheres touch, or an imaginary circle with negative squared radius in the case of no intersection. A third sphere can be introduced and wedged with the coincident circle, resulting in a point pair.

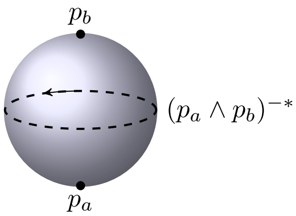


Figure 2.12: The antipodal point pair $\kappa = p_a \wedge p_b$ is dual to an imaginary circle equator: $K^* = \kappa$.

circle where they meet. We can recognize three distinct types of circles by examining the intersection of spheres in Figure 2.11. We see that circles, like spheres, can also be thought of as real or imaginary, depending on whether they have positive or negative squared radius. At zero radius, circles are considered **tangent bivectors**.

The attentive reader may have noticed that since dual spheres contain the same basis elements as points, the outer product of dual spheres is a type of point pair. Since the dual of the outer product of dual spheres is the circle where they meet, then there is a special relationship between point pairs and circles. Indeed, this is the case: a point pair is a dual representation of a circle. The duality of point pairs and circles proves quite useful in subsequent constructions. Wedging two points together creates a 2-blade point pair, which can be thought of as representing two antipodal points on a sphere. Undualizing the point pair, we extract the equatorial imaginary circle of that sphere (see Figure 2.12).

Such geometrically significant relationships led Dorst (in [45]) to give the dual of the meet a special name: the **plunge**. Wedging two dual representations of elements together returns an element that plunges orthogonally into both of them. We will use this property in Section 6.3.8 to find the circle orthogonal to two spheres through some point p.

Given a point pair $\kappa = p_- \wedge p_+$, a useful formula from Chapter 14 of [45] refactorizes the blade back into two null vectors:

$$p_{\mp} = \frac{\kappa \pm \sqrt{|\kappa^2|}}{-n_\infty \cdot \kappa} \tag{2.57}$$

This is useful in extracting information from geometrically defined constraints, as we will see in the next section. For instance, point pairs result when intersecting a line with a sphere. The meet of a line $\Lambda = p_a \wedge p_b \wedge n_\infty$ through a sphere σ is calculated as the dual circle:

$$\kappa = (\Lambda^* \wedge \sigma)^* \tag{2.58}$$

which is a point pair of the form $p_- \wedge p_+$ with the same orientation as the line. The first point along the oriented line that intersects σ is then $\frac{\kappa - \sqrt{|\kappa^2|}}{-n_\infty \cdot \kappa}$. Note that in Equation 2.58 we dualize, rather than undualize, the outer product. This is motivated by a desire for consistency with orientation and justified by the treatment of point pairs as the dual version of circles.

Another very useful expression from Dorst et al [45] is the construction of a round element from a sphere σ and a Euclidean subspace vector or bivector $\mathbf{A}$,

$$\sigma \wedge (\sigma \cdot (-\hat{\mathbf{A}} n_\infty)) \tag{2.59}$$

which gives the direct (outer product null-space) imaginary round element component of σ in the $\mathbf{A}n_\infty$ direction, as shown in Figure 2.13.

Reflections in round elements, or **inversions**, are possible using rounds as versors in Equation 2.28. In fact, finding the center point of a round element can be accomplished by reflecting infinity in it and normalizing the result. An often cheaper way to find the center of a round

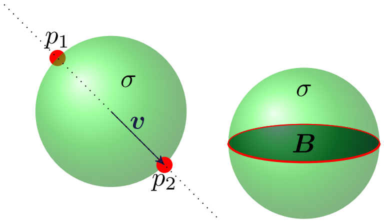


Figure 2.13: Extracting antipodal point pairs and equatorial circles from spheres using equation 2.59. If $\mathbf{v}$ is a Euclidean vector and $\mathbf{B}$ a Euclidean bivector, then $\sigma \wedge (\sigma \cdot (\mathbf{v}n_\infty))$ gives the imaginary point pair $p_1 \wedge p_2$ in the $\mathbf{v}$ direction, while $\sigma \wedge (\sigma \cdot (-\mathbf{B}n_\infty))$ gives the imaginary circle in the plane of $\mathbf{B}$. To create real rounds, we can replace $\sigma = p - \frac{1}{2}\delta^2 n_\infty$ with the imaginary $\sigma' = p + \frac{1}{2}\delta^2 n_\infty$.

is to get the dual sphere surrounding it:

$$\sigma = \kappa/(-n_\infty \cdot \kappa) \tag{2.60}$$

and then to extract the e_1, e_2, e_3 component vector $\mathbf{v}$ by rejecting c from E (using Equation 2.30, or, if using a computer, just selecting the first three coefficients). $\mathbf{v}$ can then be mapped to a null vector $p \mapsto n_o + \mathbf{v} + \frac{1}{2}\mathbf{v}^2 n_\infty$.

Factoring rounds into their carrier and surrounding sphere is often useful for calculating incidence relationships. Given a direct round κ we can specify its carrier line or plane by wedging with infinity, as in equations 2.50 and 2.51. This carrier can then be divided out to find the dual sphere **surround** as:

$$\sigma = \kappa/(\kappa \wedge n_\infty) \tag{2.61}$$

2.4.4 Transformations

A central benefit of the conformal model is its successful representation of **transformation** as a geometric concept. This is primarily demonstrated in its respect of geometry as a study of invariance under transformation, and extended by its use of geometric primitives to generate transformations themselves. As opposed to matrix representation of transformations, from which there is no canonical way to extract its geometric content, the exponential representation of Section 2.3.5 provides clear access to the underlying geometry of a spinor operator through the **logarithm**.

Recall the result of section 2.4.2, which describes how the conformal model admits bivectors that square to -1, 0, and 1. Using the full range of these bivectors in the exponent e^B generates a variety of new rotors. These bivectors are called generators. We investigate the results of using the general bivector point pair as an exponent in Chapters 5 and 6.

Equations 2.14 and 2.33 relate exponentiation of a Euclidean bivector **B** to the Euler formula. For other 2-blades in the conformal model, the rules for exponentiation depend on the sign of the square of the exponent:

$$R = e^{-\frac{\theta}{2}B} = \begin{cases} \cosh(\frac{\theta}{2}) - \sinh(\frac{\theta}{2})B & \text{if } B^2 > 0 \\ \cos(\frac{\theta}{2}) - \sin(\frac{\theta}{2})B & \text{if } B^2 < 0 \\ 1 - \frac{\theta}{2}B & \text{if } B^2 = 0 \end{cases} \tag{2.62}$$

which is a result based on the Taylor expansion of exponential, hyperbolic, and trigonometric functions. Thus if $B = \mathbf{v}n_\infty = \alpha e_1 n_\infty + \beta e_2 n_\infty + \gamma e_3 n_\infty$ is a Euclidean vector **v** wedged with the null-infinity blade, then its exponential is $e^{-\frac{\mathbf{v}}{2}n_\infty} = 1 - \frac{\mathbf{v}}{2}n_\infty$ (this particular example of a rotor is a translator).

Given a unit non-null 2-blade B weighted with θ, the scalar weight θ can be extracted as the norm $||\theta B||$ from Equation 2.21 while B can be found via normalization as $\theta B/||\theta B||$.

In the 5D conformal model, all continuous Euclidean transformations are represented as *even* multivector sums of grades 0,2, and 4. For instance, a rotation of amount θ in the Euclidean bivector plane B, can be constructed from a scalar plus a bivector: $R = \cos\frac{\theta}{2} - \sin\frac{\theta}{2}\mathbf{B}$. This construction is isomorphic to the quaternions, and can be applied to a vector v to generate a rotated vector v' via the versor product:

$$v^{'} = RvR^{-1} \tag{2.63}$$

Other types of rotors (i.e. spinors) exist for all Euclidean transformations, and these are applied in an identical manner as equation 2.63, greatly facilitating programming. For *any* rotor R and any multivector X:

$$X' = RXR^{-1} \tag{2.64}$$

The method of defining rotors through exponentiation of bivectors ensures that the inverse of the resulting rotor is the same as its reverse. Thus equation 2.64 can be written in a manner that is faster to implement:

$$X^{'} = RX\tilde{R} \tag{2.65}$$

A description of the various types of rotors can be found in [40] and [46]. Here we list the ones that can be directly built with equation 2.62:

- A **rotation** of θ radians in the **B** plane around the origin: $R = e^{-\frac{\theta}{2}\mathbf{B}} = \cos\frac{\theta}{2} - \sin\frac{\theta}{2}\mathbf{B}$.
- A **general rotation** of θ radians around the dual line λ: $M = e^{-\frac{\theta}{2}\lambda} = \cos\frac{\theta}{2} - \sin\frac{\theta}{2}\lambda$.
- A **translation** by amount v: $T = e^{-\frac{\mathbf{v}}{2}n_\infty} = 1 - vn_\infty/2$.
- A **dilation** of amount c relative to the origin: $D = e^{-\frac{c}{2}E} = \cosh\frac{c}{2} - \sinh\frac{c}{2}E$.
- A **dilation** of amount c relative to a point p: $D = e^{-\frac{c}{2}p\wedge n_\infty} = \cosh\frac{c}{2} - \sinh\frac{c}{2}p\wedge n_\infty$.
- A **transversion** relative to the origin: $B = e^{-\frac{c}{2}\mathbf{v}n_o} = 1 - vn_o/2$.
- A **conformal transformation** through the point pair κ: $C = e^{-\frac{\kappa}{2}} = ...$ (The expansion depends on whether κ^2 is greater than, less than, or equal to 0. See Equation 2.62).

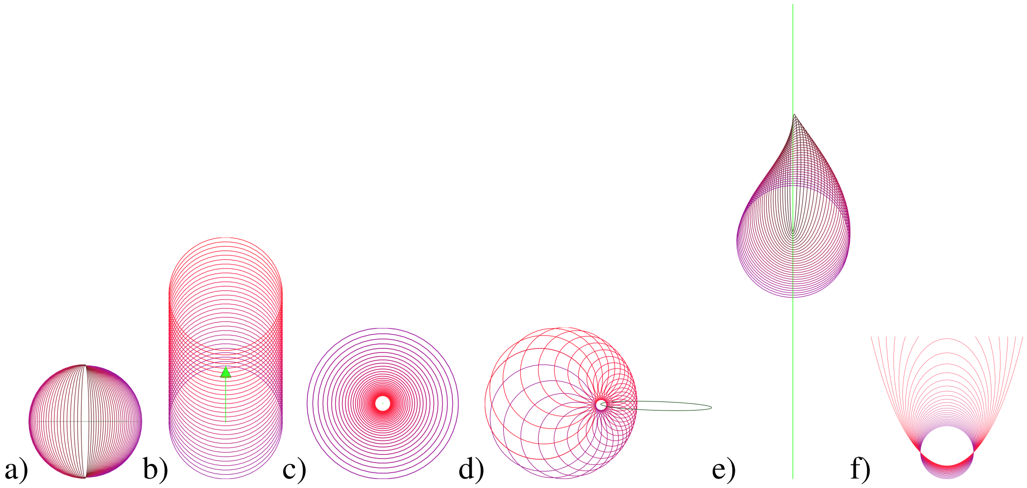


Figure 2.14: The simple continuous 3D conformal transformations representable as orthogonal versors in $\mathbb{R}^{4,1}$ include a) rotations, b) translations, c) dilations, and d) conformal transformations. e) Transformations can be combined, for instance a rotation around a line and translation along it generates a screw motion. f) Projective transformations are not a type of conformal transformation since angles are not preserved. (However, these can be represented in the conformal model through a *conic transform function* [90, 134].) Versors can be used to represent projective transformations in $\mathbb{R}^{3,3}$ [96, 42].

These last two will be more fully explored in Chapters 5 and 6. All of the above **simple** rotors can be combined to form more complex articulations. For instance, a **motor** can be made by composing a translation with a rotation to form a screw motion, $M = TR$, which encodes *any* general rigid-body transformation. Such twisting motors can also be generated directly by the dual line axis, though through a decomposition more involved than equation 2.62. These types of motions are explored in more detail in Chapters 4 and 5.

2.4.5 Interpolation

The general approach to interpolating transformations will be outlined here, and applied in Chapter 5. If we want to find the transformation R that takes a k-blade A to another k-blade B, we first calculate *twice* the transformation as the normalized ratio

$$R^2 = (\frac{B}{A})/\|\frac{B}{A}\|. \tag{2.66}$$

From here we have two options: if we only want that transformation without interpolation, then we directly find the square root using the general method of $R = \sqrt{R^2} = (1+R^2)/\|1+R^2\|$. This requires dividing $1+R^2$ by its norm, defined for a Euclidean 3D rotor in section 2.3.6 and Equation 2.38, and for general rotors in the conformal model in Dorst and Valkenburg [46] and Section 5.3.6.

If, however, we are looking to interpolate the transformation R to create a continous transformation from A to B, then rather than taking the square root we can just take *half* the log of R^2 to get a bivector which we can multiply by some t in the range of $[0,1]$ and then re-exponentiate to get an interpolated transformation rotor R_t. We consider R^2 to be an exponential of some bivector λ over twice the phase θ in the form $R^2 = e^{-2\theta\lambda}$, and therefore since $\log(R^2) = -2\theta\lambda$ we find

$$R_t = e^{-\frac{t}{2}\log(R^2)} \tag{2.67}$$

which is the approach used in continuous transformations of Chapter 5. It remains to investigate the logarithm of a normalized ratio $\frac{B}{A}$.

2.4.5.1 Finding The Logarithm

Given a rotor R=e^B, the general logarithm of a rotor finds its generator B, which is either a 2-blade in which case the rotor is “simple” or a general bivector sum $B = B_+ + B_-$ of orthogonal commuting 2-blades. Given the generator B, it can be linearly weighted (multiplied by a scalar) and then exponentiated to return a different phase of the rotor $R = e^{\phi B}$.

From Dorst and Valkenburg’s text we find that if B is a 2-blade then the rotor R is “simple” and can be written in the closed form of Equation 2.62.

The method for finding the logarithm is:

$$\log(R) = \mathrm{atanh2}(\langle R\rangle_2, \langle R\rangle) \tag{2.68}$$

where brackets $\langle R\rangle_k$ specify the k-grade basis elements of R (with $\langle R\rangle = \langle R\rangle_0$) and where

$$\text{atanh2}(s,c) = \begin{cases} \frac{\text{asinh}(\sqrt{s^2})}{\sqrt{s^2}} s & \text{if } s^2 > 0 \\ s & \text{if } s^2 = 0 \\ \frac{\text{atan2}(\sqrt{-s^2},c)}{\sqrt{-s^2}} s & \text{if } -1 \leq s^2 < 0. \end{cases} \tag{2.69}$$

A notable addition to this last case where $-1 \leq s^2 < 0$ is a modification that enables a rotation in the opposite direction,

$$\frac{\text{-}(\pi\text{-atan2}(\sqrt{-s^2},c))}{\sqrt{-s^2}} s \tag{2.70}$$

which we will use in Chapter 6.

In cases where B is not a 2-blade but a general bivector (such as the dual lines with modified pitch in section 5.2 or the ratio of two circles in Chapter 5), finding the logarithm of R entails finding two orthogonal commuting 2-blades generators B_+ and B_- (see [46] for details). In this more general case the bivector is split into two as $R = e^B = e^{B_+ + B_-} = e^{B_+}e^{B_-} = e^{B_-}e^{B_+}$ before applying equation 2.68 to each B_- and B_+ separately.

2.4.5.2 Ratios of Rotors

Given two rotors R_1 and R_2, the *relative* rotor that takes the first to the second is $R_2R_1^{-1}$, which can be thought of as applying the second to the inverse of the first. Of course this bears a resemblance to the method of determining rotors that take blades to other blades from equation 2.66.

2.5 Notation

We rely on the 2010 Amsterdam convention for notation of the conformal geometric algebra of $\mathbb{R}^{4,1}$, with three Euclidean basis blades $\{e_i\}$, homogenizing origin $n_o = .5(e_- + e)$, and infinity $n_\infty = e_- - e_+$., and null points $p = n_o + x + \frac{1}{2}x^2 n_\infty$. Note that n_o and n_∞ are sometimes identified as e_o and e_∞ or o and ∞ or $\bar{n}$ and n in other texts, and are possibly weighted differently.

Brackets $\langle X \rangle_k$ around a multivector signify a variable that contains only the k-graded elements of X. The absence of a subscript (as in $\langle X \rangle$) signifies only the scalar component, $\langle X \rangle_0$.

We use the generalized dot notation ($A \cdot B$) throughout instead of the contraction ($A \rfloor B$ or $A \lfloor B$) notation found in other texts. This means simply taking the lowest grade blade from the product:

$$A \cdot B = \begin{cases} \langle AB \rangle_{\text{grade}B - \text{grade}A} & \text{if grade}A \leq \text{grade}B \\ \langle AB \rangle_{\text{grade}A - \text{grade}B} & \text{otherwise} \end{cases} \tag{2.71}$$

and noting that if either multivectors are scalar the result is 0.

Following the style of [45], lower case greek letters refer to the dual (*inner-product null space*) representations of geometric elements, and upper case greek letters the direct (*outer-product null space*) representations. Thus where possible in our algorithms we use σ and Σ for dual and direct spheres, λ and Λ for dual and direct lines, π and Π for dual and direct planes, κ for point pairs (dual circles) and null tangent vectors, and K for circles and null tangent bivectors.

I is the 5-blade pseudoscalar of $\mathbb{R}^{4,1}$, with duality in the conformal space by multiplication with I^{-1} is notated with a star, as in $\sigma = \Sigma^*$. Involution of an element Σ is indicated with the hat symbol $\hat{\Sigma}$. We specify rotors as R, rather than in calligraphic font $\mathscr{R}$ as we have done in previous texts, and we notate their application to a generic element X via the versor product mapping of $X \mapsto R[X] = RX\tilde{R}$. Finally in some algorithms we write $[X]_{\text{normalized}}$ to signify dividing out the reverse norm (Equation 2.21) such that $[X]_{\text{normalized}} = X/\|X\|$.

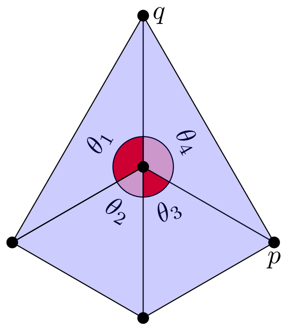


Figure 2.15: A degree-4 vertex with sector angles θ_i. If alternating angles sum to π (i.e. such that $\theta_1 + \theta_3 = \theta_2 + \theta_4 = \pi$) the arrangement represents a flat-foldable and developable crease pattern. Given such a crease pattern, our goal is to determine a legitimate (isometrically constrained) position for p given some movement of q.

2.6 Example: Constructing Folds with Round Elements

Using the encoding of round elements and their intersections described in this chapter, we can generate constructions in 3D space that are similar to the ruler and compass methods of 2D (we could, in fact, do so in any dimension $\mathbb{R}^n$ by conformalizing it to $\mathbb{R}^{n+1,1}$). To demonstrate such constructions, we will model the motion of a simple rigid-foldable origami pattern: isometric folds enabled by a system of creases in a plane. The design of such algorithms is called **computational origami**, and has been developed beyond paper-folding as a tool for architectural and material design [119, 118, 129, 130, 131, 85, 122, 61]. Here, to emphasize the coordinate-free approach of geometric algebra, we introduce a simple method for determining the folding motions around a vertex by intersecting spheres.

Given a degree-4 vertex (a vertex with 4 emanating edges) on a flat surface (i.e. a piece of paper) whose alternating sector angles sum to π (see Figure 2.15), we would like to construct a continuous **isometric** transformation of the vertices into a folded state, such that the edge lengths are preserved at every stage.

We can begin our construction by noting that every edge around a vertex represents a fixed distance, which we encode as a sphere. Determining the folding motion followed by a point around an edge is tantamount to finding a circle of possible positions given two distances from two points: that is, the intersection of two spheres. Figure 2.16 illustrates this concept

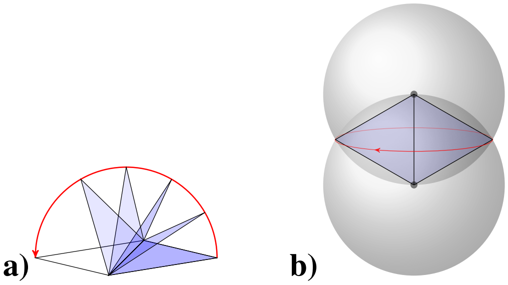


Figure 2.16: Constructing isometric folding configurations with sphere intersections. **a)** A folding motion can be thought of as a circle. **b)** The circular motion computed as the intersection of two spheres, each representing a distance constraint centered at opposites ends of an edge.

of using geometric primitives to construct constraint relationships and possible positions of a mobile structure.

This process can be extended to add a third constraint: finding the intersection of three spheres is tantamount to finding two possible positions (i.e. a point pair) a given distance from three points. Figure 2.17 illustrates adding a third sphere in order to determine possible positions of a point p. The result serves as a purely geometrical, coordinate-free, formulation of what typically requires trigonometric compositions and careful coordinate manipulation. It is a 3D version of the ruler and compass constructions of high school geometry. Noticeably, there is no trigonometry involved when encoded with CGA.

We might wonder whether such simple constructions can extend to more complicated crease patterns. Indeed, degree-4 vertices can be chained together to form larger quadrilateral meshes which open and close with 1 degree-of-freedom, called *Miura-Ori*. Figure 2.18 illustrates a crease pattern upon which we can overlay a network of spheres, and Figure 2.19 some possible folded configurations of such a network.

This is a rather naive “brute-force” approach to finding the allowed configurations of a constrained system, and requires careful sequencing of the meet operations. More work, beyond the scope of this document, would be necessary to model the non-linear actions of origami in general using spheres and circles and point pairs. Such exercises, however, push us to see how far we can go without complex algebraic deductions – i.e. we let the geometry

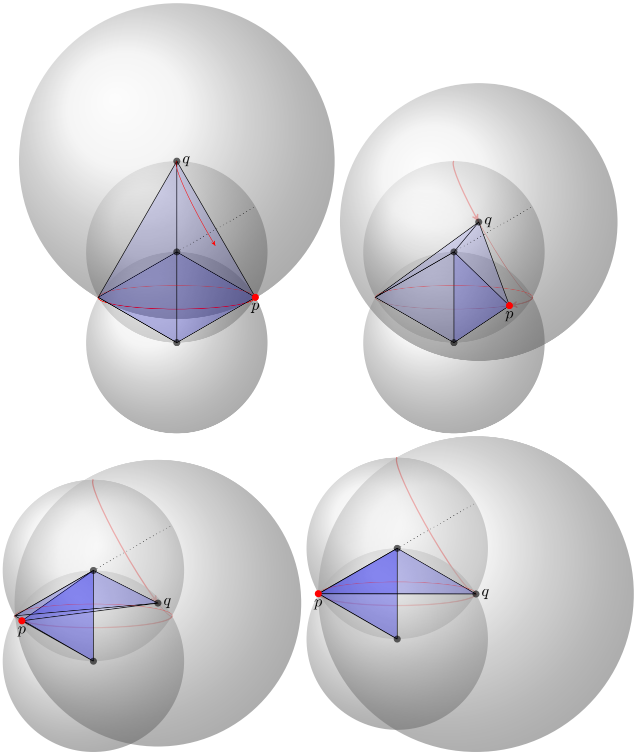


Figure 2.17: A point p (in red) computed as one of two possible intersections of three spheres. As a point q is folded, its associated sphere constraint on p intersects the circle at varying points, effectively calculating the movement of p.

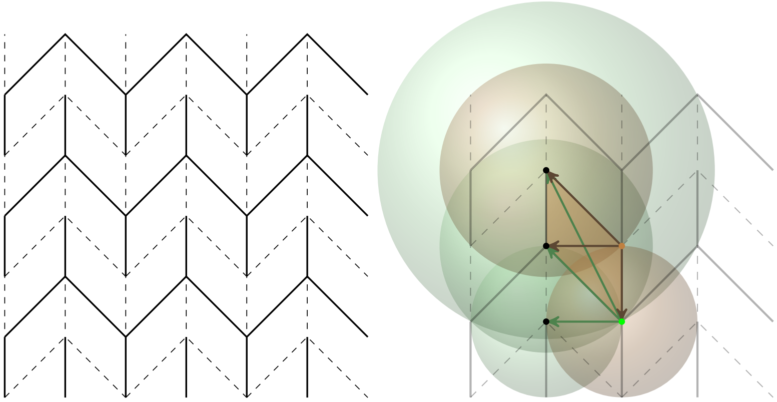


Figure 2.18: Left, a Miura-Ori crease pattern. Right, a construction network generated over the pattern.

do the work – and serve to guide our kinematics work in Chapter 4.

For further reading, Tachi [130, 129] explores form-finding of a rigid-foldable quadrilateral origami structures – *freeform* generalizations of the patterns illustrated here – using trigonometry and Jacobian matrices. These would be interesting to translate into the language of GA.

2.7 Discussion

In this chapter we have presented the mechanics of geometric algebra and the conformal model of $\mathscr{G}^{4,1}$ necessary for a self-contained treatment. We introduced some of the geometric primitives encoded by the model, as well as some of the transformations they induce through exponentiation. The remainder of this thesis will be dedicated to taking a closer look at generating structures from some of these transformations.

To get a taste of the kinds of reasoning basic $\mathscr{G}^3$ enables, we considered the problem of eliminating camera roll by projecting an "absolute" Y direction (represented as e_2) onto the bivector 2-blade of some preliminarily transformed frame of reference, in order to find an as-vertical-as-possible orientation for our camera. To dive into construction techniques facilitated by the conformal representation, we developed a trigonometry-free model of a

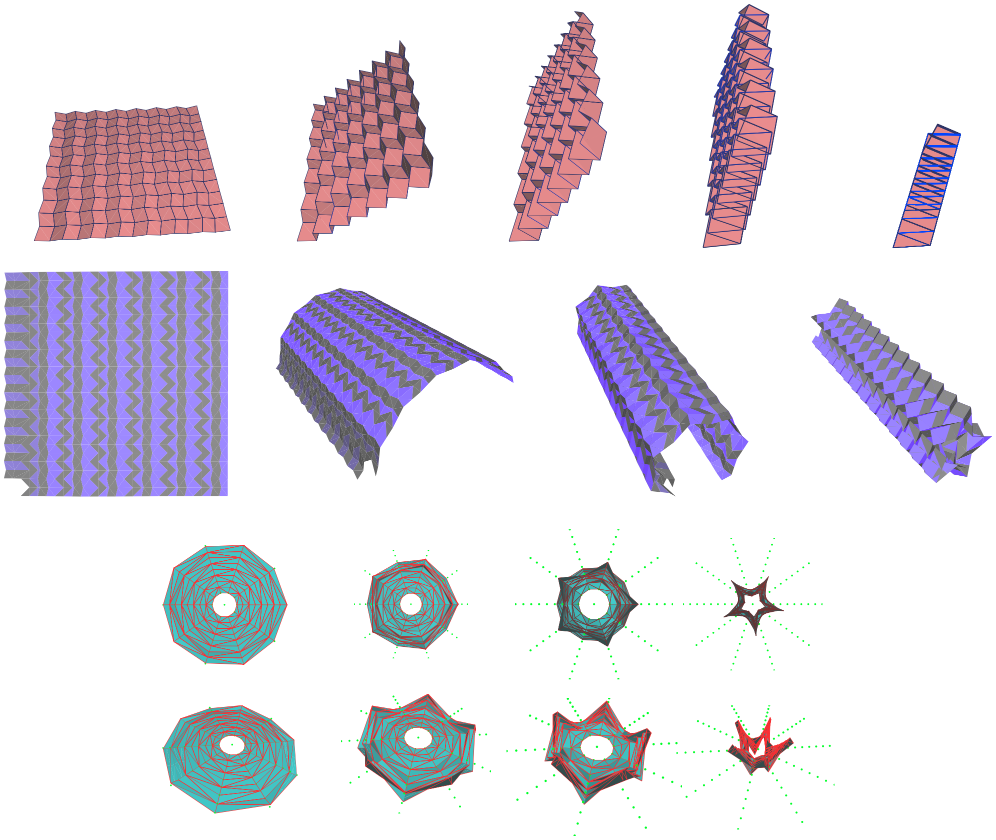


Figure 2.19: Miura-Ori style folding motions generated by applying a series of outer products of three spheres.

flat-foldable crease pattern, and took a look at some of the folded states that can be figured using this intersection-based approach.

These illustrations provide an introduction to the use of geometric primitives such as vectors, bivectors, point pairs, circles and spheres for reasoning with and about space. It reveals that once the spatial system is activated in a design environment, we can quickly start to use it to model intricate articulations without consideration of individual coordinates. With generic coordinate-free expressions such as projection: $(a \cdot B)B^{-1}$ and the meet: $(\sigma_1 \wedge \sigma_2)^{-*}$, we do not have to keep track of coordinates x, y, and z when figuring out a rotation sequence or calculating the positions of a node in a folding pattern. In implementations (see Appendix A), these types of expressions are written in a similar style: e.g. as `(a<=B)/B` and `(s1^s2).undual()`, making it very easy to translate the mathematics written on the page onto the screen.

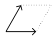
Symbol	Geometric State	Grade(s)	Algebraic Form
α	Scalar	0	α
	Vector	1	$\mathbf{a} = \alpha e_1 + \beta e_2 + \gamma e_3$
	Bivector 2-blade	2	$\mathbf{B} = \mathbf{a} \wedge \mathbf{b}$
	Trivector 3-blade	3	$\mathbf{I}_3 = \mathbf{a} \wedge \mathbf{b} \wedge \mathbf{c}$
	Point	1	$p = n_o + \mathbf{a} + \frac{1}{2}\mathbf{a}^2 n_\infty$
●●	Point Pair	2	$\kappa = p_a \wedge p_b$
○	Circle	3	$K = p_a \wedge p_b \wedge p_c$
	Sphere	4	$\Sigma = p_a \wedge p_b \wedge p_c \wedge p_d$
	Flat Point	2	$\Phi = p \wedge n_\infty$
	Line	3	$\Lambda = p_a \wedge p_b \wedge n_\infty$
	Dual Line	2	$\lambda = \mathbf{B} + \mathbf{d}n_\infty$
	Plane	4	$\Pi = p_a \wedge p_b \wedge p_c \wedge n_\infty$
	Dual Plane	1	$\pi = \mathbf{n} + \delta n_\infty$
	Minkowski Plane	2	$E = n_o \wedge n_\infty$
	Direction Vector	2	$\mathbf{t}n_\infty$
	Direction Bivector	3	$\mathbf{B}n_\infty$
	Direction Trivector	4	$\mathbf{I_3}n_\infty$
	Tangent Vector	2	$n_o\mathbf{t}$
	Tangent Bivector	3	$n_o\mathbf{B}$
	Tangent Trivector	4	$n_o\mathbf{I}_3$
	Rotor	0, 2	$R = e^{-\frac{\theta}{2}\mathbf{B}} = \cos\frac{\theta}{2} - \sin\frac{\theta}{2}\mathbf{B}$
	Translator	0, 2	$T = e^{-\frac{\mathbf{d}}{2}n_\infty} = 1 - \frac{\mathbf{d}}{2}n_\infty$
	Motor (Twist)	0, 2, 4	$M = e^{-\frac{\theta}{2}\lambda} = e^{-\frac{\theta}{2}(\mathbf{B}+\mathbf{d}n_\infty)} = RT$
	Dilator	0, 2	$D = e^{-\frac{c}{2}E} = \cosh\frac{c}{2} - \sinh\frac{c}{2}E$
	Transversor (Boost)	0, 2	$C = e^{-\frac{\theta}{2}\kappa} = \cosh(\frac{\theta}{2}) - \sinh(\frac{\theta}{2})\kappa$

Table 2.6: Basic elements of 3D conformal geometric algebra and their algebraic constructions. Subspaces of the model serve as direct and dual representations of geometric entities such as circles and lines and planes. Bold symbols represent Euclidean elements, with lowercase letters representing 1-blade vectors as is the custom. With $e_-^2 = -1$ and $e_+^2 = 1$, then $n_o = (e_- - e_+)/2$ and $n_\infty = e_- + e_+$ represent the point at the origin and the point at infinity, respectively.

Chapter 3

Symmetry: Constructing Space Groups

> “The universe is asymmetric and I am persuaded that life, as it is known to us, is a direct result of the asymmetry of the universe or of its indirect consequences.”
> – Louis Pasteur
>
> “The heart of mammals is an asymmetric screw.” – Hermann Weyl

3.1 Summary

As pointed out by Hitzer in [78], Hermann Grassmann viewed crystal structures as a particularly appropriate application for his algebra of forms. Symmetry groups are well-represented by versor multiplication, and the translation symmetries needed to describe space group tessellations became encodable as versors with the advent of the conformal model. Illuminating the expressive powers of the algebra, David Hestenes has shown the simplicity of an algebraic representation of all 230 crystallographic space groups. In 2002, Hestenes published *Point Groups and Space Groups in Geometric Algebra* [71], and again more fully in 2007, along with Jeremy Holt, *The Crystallographic Space Groups in Geometric Algebra* in the Journal of Mathematical Physics [73]. These elegant papers demonstrated the ability to encode the complex symmetries of 3D tessellations within the mechanism of conformal geometric algebra. Since then, Dechant has more fully systemized the versor representation of both crystallographic and non-crystallographic groups [34], and applied it to the classification of viruses. Applying the algebraic work to the visualization of symmetry groups, Hitzer and Perwass created the *Space Group Visualizer* [SGV], a stand-alone software for exploring the symmetries embedded within the 230 three-dimensional crystallographic space groups.

In order to leverage the results of Chapter 2, it will be helpful to establish how geometric algebra can be used as a language of symmetry. Beyond visualizing symmetry itself, we would like to develop techniques for its use as a synthetic engine, in order to provide a general mechanism for creating structures based on these symmetries. In [98], Liu, Hel-Or, Kaplan, and Van Gool point to four motivations for developing a computational representation of symmetry: it is *ubiquitous* in real and virtual worlds, it is *essential* in perception and behavior, it is *compact* when expressing form through transformation, and it is *aestho-physiological* in developing design.[1] In addition to its role in the morphology of life itself,[2] symmetries are what we use to build stable structures, to design deployable forms, to express fashion, and to recognize forms.

To enable the construction of designs based on these symmetries, we detail how to generate these groups from scratch, and how to apply the resulting symmetries to any **motif**. Our main contribution is the detailing of how to parameterize the generation of roots and transformation operators in such a way that the resulting operations can not only be observed but also *applied* to arbitrary shapes as a constructive design technique. While the existing texts

---

[1] In the cited text these reasons are actually used to argue for the need for computational symmetry analysis techniques in advancing machine intelligence and computer vision. Using a computer to help in the classification of symmetric systems could enhance computer vision techniques. The ability to *synthesize* symmetric models will aid our ability to *analyze* real-world incarnations.

[2] The famed historian of the life sciences, Georges Canguilhem, writes in [21] that when Louis Pasteur discovered bacteria, he was not looking for living organisms but looking at the geometric properties of crystals (tartrate and paratartrate). Experimenting with their polarizing effects on light, he found mold was consuming only the crystals of a certain handedness, thereby changing the polarity of the light that passed through. Life itself was breaking the symmetry, selecting only left-handed molecules, leaving their enantiomorphs alone. One of the characteristics of living material is this sensitivity to spatial orientation. Certain molecules can heal or hurt us, depending only on their chirality. Hermann Weyl picks up this symmetry-breaking theme of life in *Symmetry* [136], telling a similar story about Pasteur and citing lectures by F.M. Jaeger [81]. Canguilhem cites François Dagognet [33] on this topic.

explain the algebraic logic behind the formulations, here we more precisely detail the algorithmic steps necessary to implement a construction based on crystallographic space groups. We begin with the very fundamental step of finding three root versors given three angular constraints.

We will explore each of the following in turn, but it will be useful to list the various **operative** ingredients of the construction here:

1. Root **versors** specified by the angles between them. These are unit-length variables $\mathbf{a}$,$\mathbf{b}$, and $\mathbf{c}$.
2. Transformation **generators** composed of the root versors. These include reflections, rotations and roto-reflections.
3. **Transformations** composed of all unique sequences of the generators.
4. Translational symmetry **edge vectors** which are generated from the root versors using Table 3.4. To distinguish these vectors from the root versors, we write these as $\underline{\mathbf{a}}$, $\underline{\mathbf{b}}$, and $\underline{\mathbf{c}}$.[3]
5. **Lattice** cells which are specified by the symmetry edge vectors.
6. **Translators** based on linear sums of the symmetry edge vectors.
7. **Joining constraints** that specify translators to be multiplied by generators in order to create glides and screws.

In the end, our crystallographic space groups are composed of a finite set of transformations which we can apply to any multivector input motif in order to generate a 3D pattern.

---

[3] This notation differs from the a, b, and c of [73] because these are different vectors. Hestenes and Holt use a given lattice cell to define the "symmetry vectors" which are root versors of non-unit length. They then extract cell edges as linear sums of these. This has the advantage of allowing them to demonstrate that the generators can be written completely in terms of the symmetry vectors. However, we find it more constructive from an implementation perspective to let the root versors be generated from angular constraint criteria. These versors remain unit length and are then used to generate cell edges, which we call $\underline{\mathbf{a}}$, $\underline{\mathbf{b}}$, and $\underline{\mathbf{c}}$. Working directly with cell edges greatly simplifies implementation of space groups, and has no negative impact on algebraic elegance.

3.2 Point Groups

A **point group** is the complete set of **transformations** composed of all unique combinations of reflection and rotation **generators** about some common center point. This point remains invariant under the transformations. Using geometric algebra, such operators are constructed by up to *n* number of **root versors** where *n* is the dimension of the space. These **versors**, unit-length *n*-dimensional vectors which can act independently to generate **reflections** or which can act in composition to generate **rotations** and rotation-reflections (**roto-reflections**). Given up to *n* root versors and a specification as to which act independently to reflect and which in composition to rotate or roto-reflect, we can generate the a point group. The set of transformations, and thus the point group, could be infinite, but here we are interested in finite point groups, related by a finite number of operations.

Various notations exist for describing point groups, and typically characterize either the list of transformations themselves (International or Hermann-Mauguin) or else the generators or versors of the transformations (Coxeter or Geometric), or else the overall symmetry of the group they generate (Schönflies). We adopt the geometric convention of [73], which is similar to Coxeter notation for specifying the *angular* relationship between the **versors** of the group. In three dimensions, we consider three versors **a**,**b**, and **c**, whose directions are parameterized by two numbers p and q which indicate the angles between **a** and **b** and between **b** and **c**, respectively. **a** and **c** are always orthogonal. We use a bar, as in $\bar{p}$, to specify when the versors **a** and **b** are to be used not as independent reflection generators but rather in composition as in **ab**, to form a rotation generator.

The overview of the methodology delineated in this section is as follows:

1. Specify root **versors** based on pq values.

2. Construct base **generators** based on bar notation.

3. Seed **transformation** sequence.

4. Apply sequence to a **motif**.

3.2.1 Specifying root versors from angular constraints p and q

The 3D crystallographic space groups are so well-studied, that the very first step regarding *how* to actually compute the root vectors given angular constraints p and q is typically omitted in the literature: researchers can just look them up in a table, or else extract them from the lattices in which they are known to fit. Because our current goal is to develop constructive techniques, we would like to be able generate the roots from scratch, following a geometric approach, saving the consideration of additional structure such as lattices only when necessary later on.

As mentioned in equation 18 of [73] the relations governing the angular constraints p and q represent rations of π. This is entailed by the condition that, for **a**, **b**, and **c** of unit length

$$(\mathbf{ab})^p = (\mathbf{bc})^q = (\mathbf{ca})^2 = -1 \tag{3.1}$$

which specifies $\frac{\pi}{p}$ radians between **a** and **b**, $\frac{\pi}{q}$ radians between **b** and **c**, and $\frac{\pi}{2}$ radians between **c** and **a**. Because **c** and **a** are always orthogonal ($\frac{\pi}{2}$ radians apart) we can set $\mathbf{a} = e_1$, $\mathbf{c} = e_2$ and then need only a procedure to find the unknown **b**. We do so by treating **b** as the intersection of two bivector planes, each satisfying a particular angular constraint.

We find these planes via the cosine rule identity from spherical trigonometry, which relates the $\frac{\pi}{p}$, $\frac{\pi}{q}$, and $\frac{\pi}{2}$ vector angles between **a**, **b** and **c** to the dihedral angles between planes $\mathbf{a}\wedge\mathbf{b}$, $\mathbf{b}\wedge\mathbf{c}$, and $\mathbf{a}\wedge\mathbf{c}$ (see Figure 3.1). For vector angles α, β, and γ, and corresponding dihedral bivector angles A, B and C, the cosine law for angles[4] is

$$\cos\alpha = \cos\beta\cos\gamma + \sin\beta\sin\gamma\cos A \tag{3.2}$$

---

[4] In Appendix A of [66], Hestenes gives a full treatment of spherical trigonometry in terms of rotors, and a derivation of the cosine law for angles.

and

$$\cos\gamma = \cos\beta\cos\alpha + \sin\beta\sin\alpha\cos C. \tag{3.3}$$

Since β is constrained to $\frac{\pi}{2}$ these equations reduce to

$$\cos\alpha = \sin\gamma\cos A \tag{3.4}$$

$$\cos\gamma = \sin\alpha\cos C \tag{3.5}$$

and solving for A and C gives

$$A = \mathrm{acos}(\frac{\cos\alpha}{\sin\gamma}) \tag{3.6}$$

$$C = \mathrm{acos}(\frac{\cos\gamma}{\sin\alpha}) \tag{3.7}$$

Setting $\gamma = \frac{\pi}{p}$ and $\alpha = \frac{\pi}{q}$ we find the angle A to rotate the bivector plane $\mathbf{a} \wedge \mathbf{c}$ about the $\mathbf{a}$ axis (i.e. the $\mathbf{a}^{-\star}$ plane) so that it contains $\mathbf{b}$. Likewise, we find the angle C to rotate $\mathbf{a} \wedge \mathbf{c}$ about the $\mathbf{c}$ axis (i.e. the $\mathbf{c}^{-\star}$plane) so that it too contains $\mathbf{b}$. Intersecting these rotated planes to find $\mathbf{b}$ involves undualizing the outer product of their duals:

$$\mathbf{b} = (R_C[\mathbf{a} \wedge \mathbf{c}]^\star \wedge R_A[\mathbf{a} \wedge \mathbf{c}]^\star)^{-\star} \tag{3.8}$$

which is then normalized to unit length. Note that setting $\mathbf{a} = e_1$ and $\mathbf{c} = e_3$ requires assigning a negative value to A to ensure we rotate $\mathbf{a} \wedge \mathbf{c}$ in the correct direction (namely, $\mathbf{a}^\star$ or$-e_{23}$), so that $R_A = e^{-(-\frac{A}{2}e_{23})} = e^{\frac{A}{2}e_{23}}$ and $R_C = e^{-\frac{C}{2}e_{12}}$.

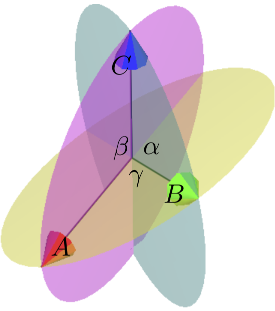


Figure 3.1: The cosine rule from spherical trigonometry relates angles between vectors to angles between the planes they span. Given three angles α, β and γ we can find the three angles A, B and C. In our construction, we can then define planes $\mathbf{a} \wedge \mathbf{b}$ and $\mathbf{b} \wedge \mathbf{c}$ as rotations of the $\mathbf{a} \wedge \mathbf{c}$ plane about the $\mathbf{a}$ and $\mathbf{c}$ axes by angles A and C respectively. We construct the $\mathbf{b}$ vector (in green) by intersecting these planes.

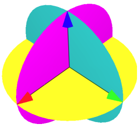
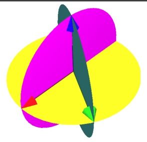
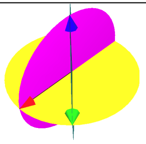
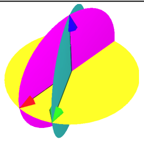
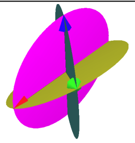
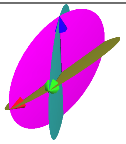
pq	22	32	42	62	33	43
a, *b*, *c*						

Table 3.1: *pq* angular constraint symbols and their corresponding vector generators. $\mathbf{a}$ is depicted in red, $\mathbf{b}$ in green and $\mathbf{c}$ in blue, $\mathbf{a} \wedge \mathbf{b}$ in yellow, $\mathbf{a} \wedge \mathbf{c}$ in purple and $\mathbf{b} \wedge \mathbf{c}$ in cyan. The angle between $\mathbf{a}$ and $\mathbf{b}$ is $\frac{\pi}{p}$ radians, and between $\mathbf{b}$ and $\mathbf{c}$ is $\frac{\pi}{q}$ radians, while $\mathbf{a}$ and $\mathbf{c}$ are always $\frac{\pi}{2}$ radians apart.

3.2.2 Generating a complete set of transformations from the simple roots a, b and c.

Given our three unit length roots **a**, **b**, and **c**, our next task is to find all the transformations they generate. This will amount to inputing a random "seed" vector and building a list of all the ways it can uniquely transform. With such a list, we can apply our transformations to any input, or replace transformations with joining constraints when we develop space groups. First, let us investigate how to encode the *pq* bar notation in an implementation.

We have three vectors **a**, **b**, and **c** which can be used as reflection operators, or **pinors**, and three rotors **ab**, **bc**, and **ac** which can be used as rotation operators, or **spinors**. Finally, we have a roto-reflection operation **abc**, which is a rotation by **bc** followed by a reflection in **a**. Order matters; a different roto-reflection is encoded by **bac**. In practice, however, we will only need to use **abc** to generate all unique space groups.

The bars over the *pq* notation (for instance, $\bar{4}2$) specifies which reflections, rotations, and roto-reflections are active in a given system. Table 3.2, partially duplicated from [73], explains the notation, as well as our own method for encoding this information in a computer implementation as integers representing the `type` of transformation (reflection, rotation, or roto-reflection), and the `index` into our lists of up to three transformations for each type ($\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$, $\{\mathbf{ab}, \mathbf{bc}, \mathbf{ca}\}$, or $\{\mathbf{abc}, \mathbf{bac}\}$).

Algorithm 3.1 demonstrates in pseudocode the following description: applying these generators to some input vector transforms it up to three ways. We call these transformed vector our list of *results*. For each of these transformed vectors, we again apply the generators and add any new results to our list. We repeat this process, applying our generators to our last results and adding new results until no new results are generated. For each new result, we note the `{type, index}` encoding used to generate it and, crucially, the index of the list that was used as the input to generate the new result. Thus, to keep track of which result in our list was used as an input to which transformation, we must add a third `input` parameter.

This seeding procedure constructs a *sequence* of operation instructions which can be applied to any other input to generate a symmetry pattern. This is sufficient for an implementation and examples of the result of applying this to a motif can be seen in Figure 3.2. Sometimes, however, it is desired to have at one's disposal the full set of transformation operations without resorting to a sequential ordering. In such a case it is straightforward to examine the sequence and construct from it the full set of absolute transformations: for every operation in the sequence, if the `input` is $n = 0$, then the base generator specified by the `type` and `index` is sufficient. Otherwise, we multiply the base generator by the nth operation in the sequence. This is a recursive scheme that traces the operations back to the original input, concatenating operations in the process. For instance, for point group 22, the full set of unique generated transformations is $\{1, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{ab}, \mathbf{ac}, \mathbf{bc}, \mathbf{abc}\}$ where 1 is the identity. Point group $\bar{2}2$ generates only four transformations, $\{1, \mathbf{ab}, \mathbf{c}, \mathbf{cab} = \mathbf{abc}\}$, as does point group $\bar{2}\bar{2}$, with $\{1, \mathbf{ab}, \mathbf{bc}, \mathbf{abbc} = \mathbf{ac}\}$, while point group $\overline{22}$ generates just two, $\{1, \mathbf{abc}\}$. Table 3.3 illustrates the differences between these four crytallographic groups.

The geometric approach to generating these transformations avoids the use of matrices, allowing us to work directly with the root versors. One advantage of this approach is that we can use the same unit-length versors in the next step to create our symmetry edge vectors with which we build a translation lattice. Another is that we can use the transformation operations on any element of the algebra, not just other vectors.

Symbol	Generators	Encoding = `{{type, index},...}`
$p = 1$	**a**	$\{\{0,0\}\}$
$p \neq 1$	**a**,**b**	$\{\{0,0\},\{0,1\}\}$
$\bar{p}$	**ab**	$\{\{1,0\}\}$
pq	**a**,**b**, **c**	$\{\{0,0\},\{0,1\},\{0,2\}\}$
$\bar{p}q$	**ab**,**c**	$\{\{1,0\},\{0,2\}\}$
$p\bar{q}$	**a**,**bc**	$\{\{0,0\},\{1,1\}\}$
$\bar{p}\bar{q}$	**ab**,**bc**	$\{\{1,0\},\{1,1\}\}$
$\overline{pq}$	**abc**	$\{\{2,0\}\}$

Table 3.2: The first two columns pair *pq* bar notation with the generators they specify, and are duplicated from [73]. We have added a third column detailing how one can encode the basic generators in an implementation as a list of {type, index} instructions. The `type` integer specifies reflections (0), rotations (1), or roto-reflections (2). The `index` specifies **a**, **b** or **c** in the case of `type=0`, **ab** or **bc** in the case of `type=1`, and **abc** or **bac** in the case of `type=2`.

Algorithm 3.1 Construct Transformation Sequence

```
procedure SEED
    $results \leftarrow \{$random input vector$\}$
    $opseq \leftarrow \{\}$
    while new results have been found do
        for $x \in results$ do
            for $g \in generators$ do
                $tx \leftarrow g[x]$
                if $tx$ is a new result then
                    $results = results + tx$
                    $opseq = opseq + \{type(g), index(g), index(x)\}$
    return $opseq$
end procedure
```

3.3 Space Groups

A **space group** is a set of transformations duplicated, or “hung” on every point on a **lattice cell**. In practice, this is achieved by applying all transformations to a motif and then translating the result to all points on the lattice cell. The next steps in our construction are therefore to:

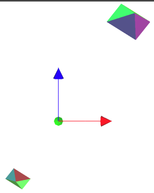
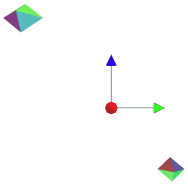
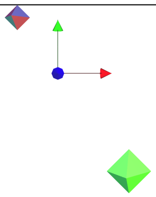
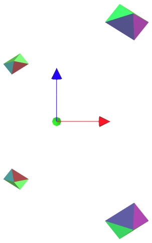
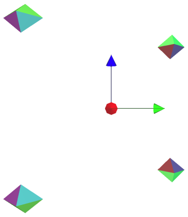
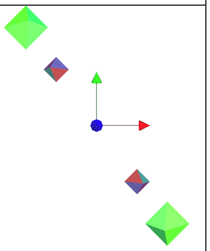
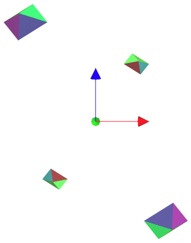
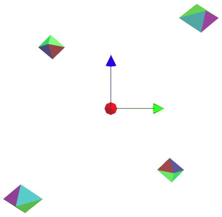
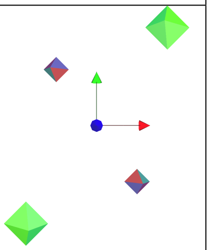
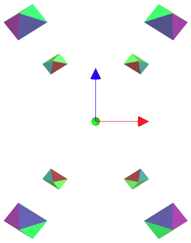
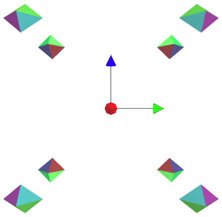
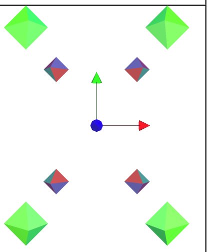
Geometric	Int’l	Front View	Side View	Top View
$\overline{22}$	$\bar{1}$			
$\{\mathbf{abc}\} \to \{1, \mathbf{abc}\}$				
$\bar{2}2$	$\frac{2}{m}$			
$\{\mathbf{ab}, \mathbf{c}\} \to \{1, \mathbf{ab}, \mathbf{c}, \mathbf{abc}\}$				
$\bar{2}\bar{2}$	222			
$\{\mathbf{ab}, \mathbf{bc}\} \to \{1, \mathbf{ab}, \mathbf{bc}, \mathbf{ac}\}$				
22	*mmm*			
$\{\mathbf{a}, \mathbf{b}, \mathbf{c}\} \to \{1, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{ab}, \mathbf{ac}, \mathbf{bc}, \mathbf{abc}\}$				

Table 3.3: The four different crystal symmetries of $p = q = 2$, along with their primary generators and the closed group of transformations they induce. The maximal symmetry, 22, is the holohedral group of which the others are subgroups.

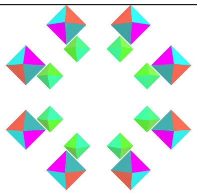
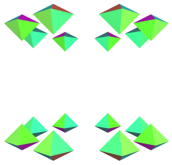
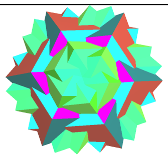
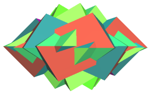
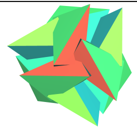
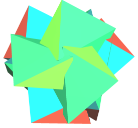
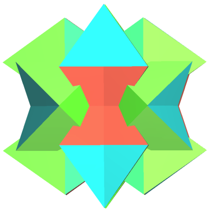
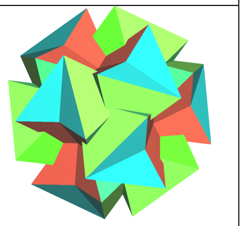
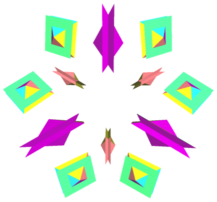
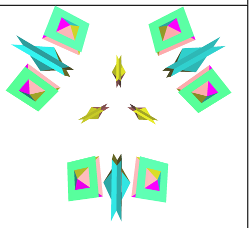
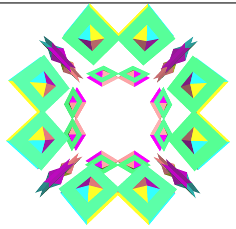
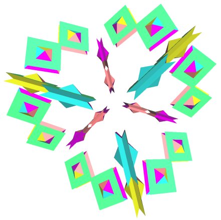
Geometric	Int'l	p	q
42	$\frac{4}{mmm}$	 4	 2
$\bar{6}\bar{2}$	622	 $\bar{6}$	 $\bar{2}$
$\bar{3}\bar{3}$	23	 $\bar{3}$	 $\bar{3}$
$4\bar{3}$	$m3$	 4	 $\bar{3}$
33	$\bar{4}3m$	 3	 3
43	$m3m$	 4	 3

Figure 3.2: Symmetries generated by applying a sequence of operations to an input motif. Notation matches Table 2 of [73].

1. Construct translational **symmetry edge vectors** based on root versors.
2. Construct **lattice cells.**
3. Add **joining constraints** that replace reflection and rotation generators with glides and screws (respectively).

3.3.1 Constructing Bravais Lattice Cells from Root Versors

With our point group of versors and their particular generators defined, the next step in the construction is to define the **symmetry edge vectors** to use for generating translational symmetries in a 3D lattice. While Hestenes and Holt define symmetry vectors a,b, and c so that they are rescaled versions of unit-length root versors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$, in our construction we use symmetry edge vectors $\underline{\mathbf{a}}$, $\underline{\mathbf{b}}$, and $\underline{\mathbf{c}}$ so that they lie along the edges of the lattice cell. This makes subsequent formulations on a computer less tedious (see Footnote 3). Table 3.4 specifies the relationship between the versors and the symmetry edge vectors, clarifying how to construct vectors from versors, as well as which are free to rotate or scale. These relationships define the **primitive** cells for each crystal **system**, where copies of the motif are "hung" on each corner of the cell. Other non-primitive cells specify additional points on which to hang a motif, for instance in the middle of it in the case of **body** cells, or in the middle of a face in the case of **face** cells (we will only consider Primitive cells in this text). Certain cells admit certain symmetries, and the combination of a point group with a lattice on which it is hung defines a **symmorphic** space group.

3.3.2 Substituting generators with joining constraints for nonsymmorphic space groups

The final step in our construction is the multiplication of reflection and rotation generators with **translators** based on linear sums of symmetry edge vectors. This results in two new

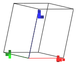
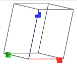
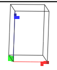
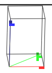
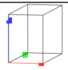
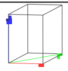
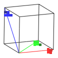
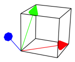
System	Cell	Edge vectors
Triclinic $\bar{1}, \overline{22}$		$\{\underline{\mathbf{a}} = \alpha a, \underline{\mathbf{b}} = \beta R_{\mathbf{a}\wedge\mathbf{b}}[b], \underline{\mathbf{c}} = \gamma R_{\mathbf{a}\wedge\mathbf{c}}[c]\}$
Monoclinic $\bar{2}, 1, \bar{2}2$		$\{\underline{\mathbf{a}} = \alpha a, \underline{\mathbf{b}} = \beta R_{\mathbf{a}\wedge\mathbf{b}}[b], \underline{\mathbf{c}} = \gamma c\}$
Orthorhombic $\bar{2}\bar{2}, 2, 22$		$\{\underline{\mathbf{a}} = \alpha a, \underline{\mathbf{b}} = \beta b, \underline{\mathbf{c}} = \gamma c\}$
Tetragonal $\bar{4}, \bar{4}2, \bar{4}2, \bar{4}\bar{2}, 4, 4\bar{2}, 42$		$\{\underline{\mathbf{a}} = \alpha \mathbf{a}, \underline{\mathbf{b}} = -\mathbf{b}\underline{\mathbf{a}}\mathbf{b}, \underline{\mathbf{c}} = \gamma \mathbf{c}\}$
Trigonal $3, \bar{3}, \overline{62}, \bar{3}\bar{2}, 3, 6\bar{2}$		$\{\underline{\mathbf{a}} = \alpha a, \underline{\mathbf{b}} = \alpha b, \underline{\mathbf{c}} = \gamma c\}$
Hexagonal $\bar{6}, \bar{3}2, \bar{6}2, \bar{6}\bar{2}, 6, 32, 62$		$\{\underline{\mathbf{a}} = \alpha \mathbf{a}, \underline{\mathbf{b}} = -\mathbf{b}\underline{\mathbf{a}}\mathbf{b}, \underline{\mathbf{c}} = \gamma \mathbf{c}\}$
Cubic $(4,3)$ $\bar{4}\bar{3}, 4\bar{3}, 43$		$\{\underline{\mathbf{a}} = \alpha \mathbf{a}, b = -\mathbf{b}\underline{\mathbf{a}}\mathbf{b}, \underline{\mathbf{c}} = \alpha(\mathbf{a}\wedge\mathbf{b})^{\star}\}$
Cubic $(3,3)$ $\bar{3}\bar{3}, 33$		$\{\underline{\mathbf{a}} = (a\text{-}c)/2, \underline{\mathbf{b}} = (a\text{+}c)/2, \underline{\mathbf{c}} = (2b - a - c)/2\}$

Table 3.4: **Primitive** Lattice Cells and the symmetry **edge vectors** $\underline{\mathbf{a}}$, $\underline{\mathbf{b}}$, and $\underline{\mathbf{c}}$ which define them. The notation details how to define the symmetry edge vectors in terms of unit length versors $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$. For instance, the Orthorhombic cell symmetry edge vectors are just linearly scaled version of the versors, where α, β, and γ are scalar values. Triclinic cells symmetry edge vectors $\underline{\mathbf{b}}$ and $\underline{\mathbf{c}}$ are free to rotate in the respective planes they form with $\underline{\mathbf{a}}$, where $R_{\mathbf{a}\wedge\mathbf{b}}$ and $R_{\mathbf{a}\wedge\mathbf{c}}$ represent arbitrary rotations in those planes. In the cubic cell, $(\mathbf{a}\wedge\mathbf{b})^{\star}$ is the vector normal to the bivector $\mathbf{a}\wedge\mathbf{b}$.

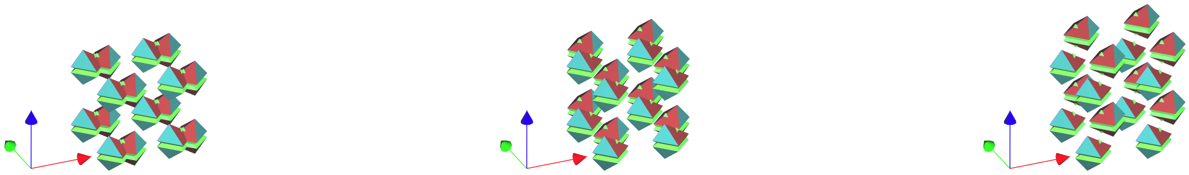


Figure 3.3: Left, the holohedral group 22 hung on a primitive lattice. Middle, reflection over **a** is converted into an axial-b glide. Right, reflection over **a** converted into an axial c-glide.

transformations: **glide** reflections (a translation followed by a reflection) and **screw** displacements (a translation followed by a rotation).

3.3.2.1 Glide Reflections

Three kinds of glides are specified, **axial** glides along the symmetry edge vectors **a**, **b**, or **c**, **diagonal** glides along the diagonal of the cell body or one of its faces (here we call these two "diagonal-face" and "diagonal-body"), and **diamond** glides halfway along a diagonal.

Axial glides are along the edge of the lattice, they are only found in lattices for which the edge is *necessarily* perpendicular to the reflection it is replacing. For instance, there are no axial b-glides replacing the **a** generator in triclinic, monoclinic, trigonal or hexagonal lattices, because the **b** edge is not perpendicular to the **a** versor in these systems.

The same absence is noted for diagonal glides with respect to the **a** generator. Additionally, almost all diagonal glides replacing the **a** generator are diagonal-face glides – the one notable exception is a single diagonal-body glide in the holohedral group 33 where **a** does not lie along an edge vector but rather itself cuts diagonally across a face.

Similar restrictions apply to glides replacing the **b** generator, for which axial a-glides and and diagonal-face glides only exist in orthorhombic systems, with diagonal-body glides limited to cubic systems. Diagonal glides replacing the **c** generator are across the face for orthorhombic and tetragonal systems, and across the body in cubic systems.

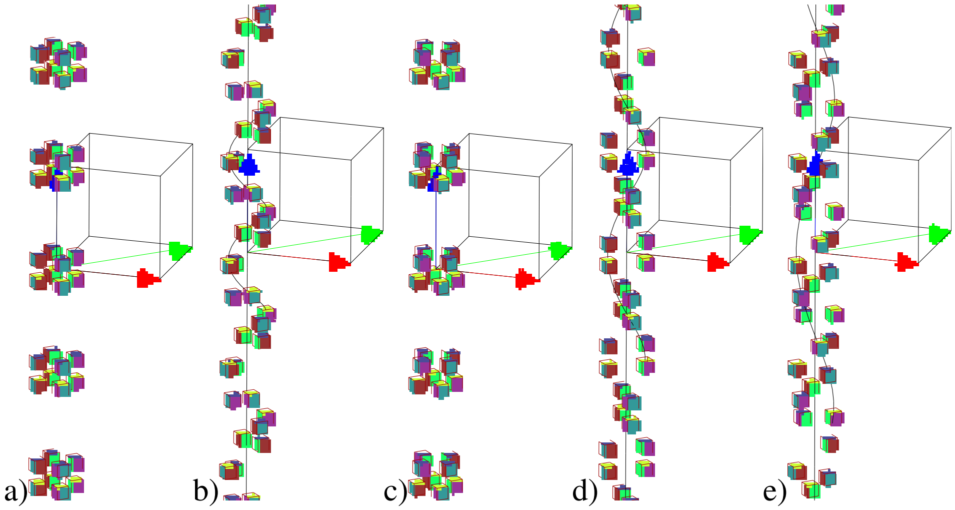


Figure 3.4: Space groups a) $P\bar{4}2$ (int'l $P4/m$), b) $P\bar{4}_1\bar{2}\bar{2}$ (int'l $P4_122$), c) $P\bar{4}\bar{2}$ (int'l $P42$), d) $P\bar{4}_22$ (int'l $P4_2/m$), and e) $P\bar{4}_3\bar{2}\bar{2}$ (int'l $P4_322$)

3.3.2.2 Screw Displacements

Whereas the above glide reflections replace reflection generators with translations followed by reflections, screw displacements replace rotation generators with translations followed by rotations. Like glides, the translation portion can be generated by linear combinations of the symmetry vectors. These directions come in lengths that are rations of the rotation parameters p and/or q.

Unlike glides, the translation component of screw displacements are always constructed along the edge of the cell, with the added caveat that sometimes the resulting screw axis is itself again translated along another edge. For instance, in space group $P\bar{2}_1\bar{2}_1\bar{2}_1$ (int'l $P2_12_12_1$), depicted in Figure , the screw axis of $bcT_{\underline{\mathbf{a}}}^{1/2}$ has been translated along the $\underline{\mathbf{b}}$ edge. The resulting generator is $T_{\underline{\mathbf{b}}}^{1/4}[bcT_{\underline{\mathbf{a}}}^{1/2}]$.

For screw displacements replacing the ab rotation, the screw translation vector is usually along the $\underline{\mathbf{c}}$ edge of the cell. For instance $F4_132$ (geom. $F\bar{4}_1\bar{3}\bar{2}$), the ab rotation is multiplied by a translation along the $\underline{\mathbf{c}}$ edge and the resulting screw transformation is shifted along the $\underline{\mathbf{b}}$ edge in the *negative* direction to create the generator $T_{\underline{\mathbf{b}}}^{-1/4}[abT_{\underline{\mathbf{c}}}^{3/4}]$.

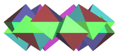
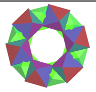
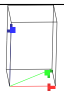
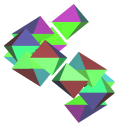
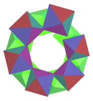
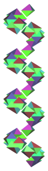
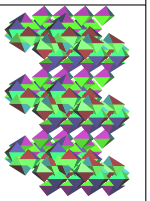
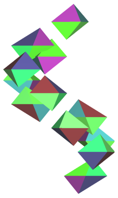
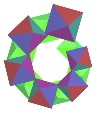
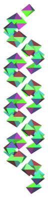
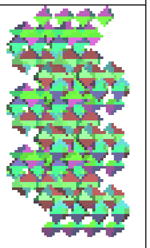
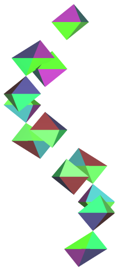
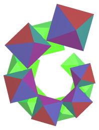
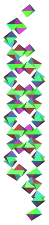
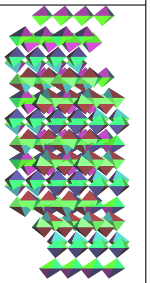
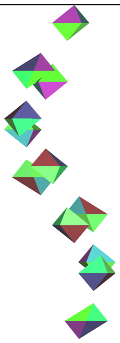
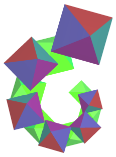
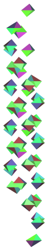
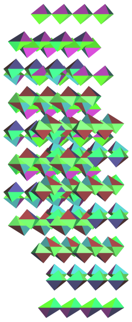
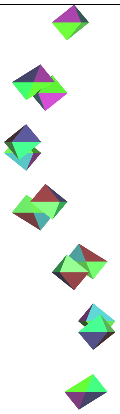
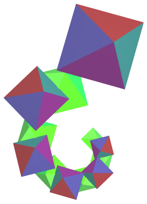
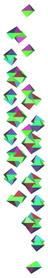
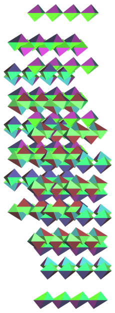
s	Int'l	Geo	Point Group	Point Group (Top View)		
0	$P622$	$P\bar{6}\bar{2}$				
1	$P6_122$	$P\bar{6}_1\bar{2}$				
2	$P6_222$	$P\bar{6}_2\bar{2}$				
3	$P6_322$	$P\bar{6}_3\bar{2}$				
4	$P6_422$	$P\bar{6}_4\bar{2}$				
5	$P6_522$	$P\bar{6}_5\bar{2}$				

Table 3.5: Screw displacements of the form $P\bar{6}_s\bar{2}$ with translation component $T_{\underline{\mathbf{c}}}^{s/6}$ generated from $s/6$th of the $\underline{\mathbf{c}}$ edge vector.

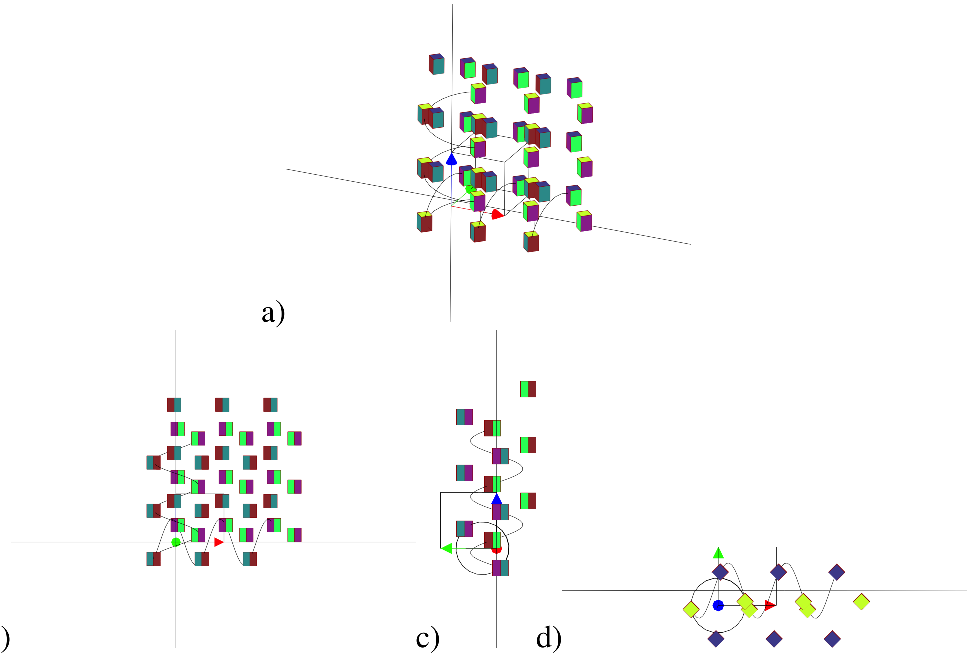


Figure 3.5: Screw displacements for space group $P\bar{2}_1\bar{2}_1\bar{2}_1$ (int'l $P2_12_12_1$). b) Front view, c) side view, d) top view.

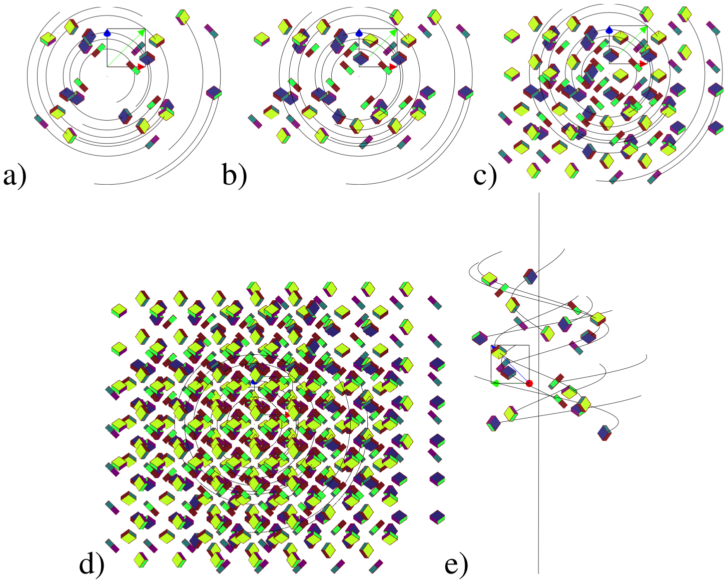


Figure 3.6: Space group $P\bar{4}_3\bar{3}\bar{2}$ (int'l $P4_332$). a) Top view of the $\bar{4}\bar{3}$ point group whose *ab* rotation generator has been replaced by a screw along the $\underline{\mathbf{c}}$ edge vector. For each transformation of the motif, we draw the screw movement as a black arc. Before repeated translation to other points on the lattice, certain symmetries are missing, notably wherever the circular black screw lines end. b) - d) These missing symmetries begin to appear during translation of the motif to other points on the lattice. e) Side view demonstrating that the $\underline{\mathbf{c}}$ screw axis has been translated in the *negative* $\underline{\mathbf{b}}$ direction.

3.4 Application: Balanced Structures

The generic covariant nature of geometric algebra ensures that we can use our space groups to transform more than just points. In Chapter 5 we will discuss continuous transformations generated by bivector elements of the algebra. These bivector elements can be used as a motifs to be transformed by a symmetry group. The resulting set of bivector elements can be used to warp some input mesh in a symmetric way. For instance, given a set of n point pairs $\{\kappa_i\}$ created by transforming an input point pair by a symmetry group, we can transform some point p as $p^{'} = C[p]$ where

$$C = e^{-\sum_{i=0}^{n} d_i \kappa_i} \tag{3.9}$$

and d_i is the squared distance between p and κ_i point pair element of the set. That is, we create a single transformation as a linear sum of bivectors. For more details on this linear weighting scheme see also our previous work on the warping of surfaces using point pairs in [28]. Figure 3.7 illustrates some of the shapes that can be synthesized in this fashion. Potential applications of these types of structures include machine parts such as blades and gears, where symmetry is often a defining parameter.

More generally, symmetry groups are a cornerstone in the study of stability in structural systems. For instance, a relationship between symmetry and engineering can be found in Buckminster Fuller's *Synergetics* [55], where (amongst many other contributions) he elaborates the notion of tension-and-compression arrangement of struts and cables exhibiting equilibrium of forces (the origin of this design technique is sometimes credited to Kenneth Snelson). Fuller terms this type of structure a **tensegrity**, short for tension-integrity, and describes it as an "island of compression in a sea of tension." We will not delve too deeply into the organizing principles of such systems, except to note that they are often symmetric, and so explore how one might apply the above symmetry constructions in an study of them. In the design of structural systems, using these symmetry groups can facilitate the engineer's

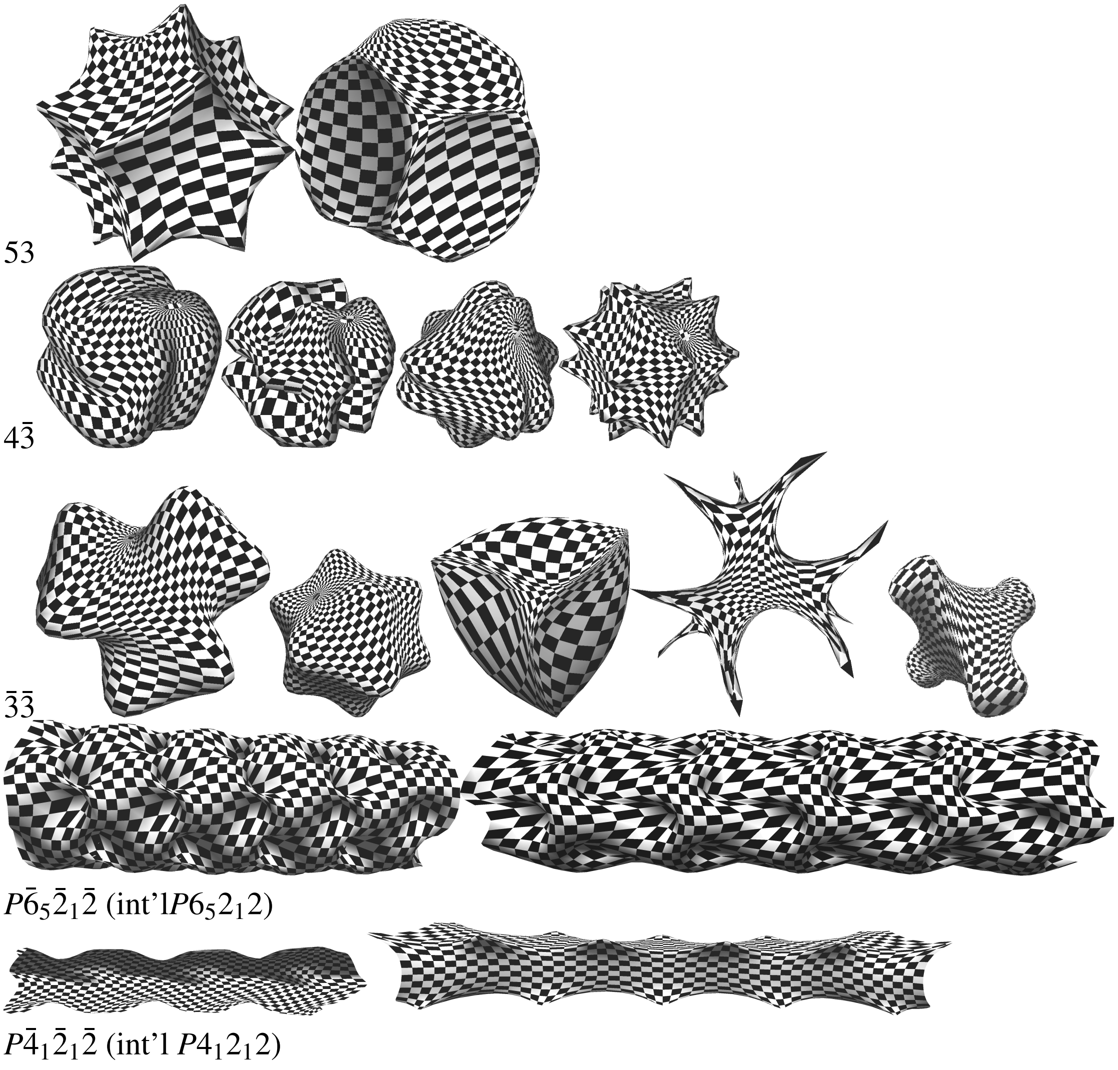

53
$4\bar{3}$
$\bar{3}\bar{3}$
$P\bar{6}_5\bar{2}_1\bar{2}$ (int'l$P6_52_12$)
$P\bar{4}_1\bar{2}_1\bar{2}$ (int'l $P4_12_12$)

Figure 3.7: Balanced shapes generated by applying equation 3.9 to each point on the surface of a sphere or cylinder. Each point is transformed by a rotor C, where C is generated by the exponential of a linear sum of distance-weighted 2-blades $\{\kappa_i\}$, and where the 2-blade set is itself generated by the action of a symmetry group.

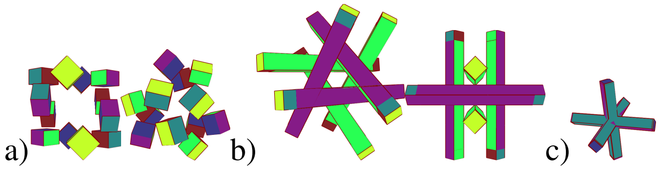


Figure 3.8: The point group $\bar{3}\bar{3}$ acting on a motif. In a) the motif is in general position creating 12 distinct transformations, while in b) the motif is in special position, creating 6 unique elements in an icoshedral form. c) Further collapsing of the motif would result in octohedral symmetry, revealing the motivation for considering b) an expanded version of c).

orchestration of tensile and compressive forces, and it appears feasible that each of the 230 different space groups can be used to design a tensegrity structure.

Particularly, in [117], Pugh explores some of the symmetry groups underpinning tensegrity systems. To explore how synthesis with symmetry groups could be used to model these systems, let us consider a well-known tensegrity system which Pugh calles the "expanded octohedron". We will model it using the point group $\bar{3}\bar{3}$, in the Cubic system, variations of which are shown in Figure 3.8. If we apply the symmetries of $\bar{3}\bar{3}$ onto a bar we can generate the struts and then consider the connection of one bar to another as a single cable which itself can also be transformed. The result of this exercise is shown in Figure 3.9. Similarly, in Figure 3.10, we use the Space Group $P\bar{3}\bar{2}$ to generate possible configurations of a tensegrity truss.

One might suspect that every point group symmetry has some corresponding tensegrity system it can help generate, and indeed all platonic and archimedean solids seem to admit a deconstruction of this sort. It remains to be explored how the full range of space group symmetries and their joining constraints might help in construction of these relations.

3.5 Discussion and Future Work

Following the specifications delineated by Hestenes and Holt in [73], we have detailed a method for constructing the 3D crystallographic space groups "from scratch". Our proposed simple modification to their symmetry vector notation has helped to clarify the translation

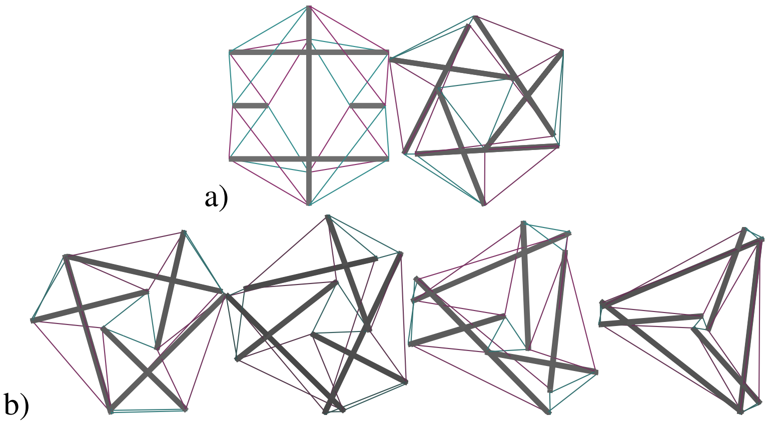


Figure 3.9: The “Expanded Octohedron” tensegrity figure (see Pugh [117]). The transformations of point group $\bar{3}\bar{3}$ are applied to a generating tensegrity strut (thick line) in special position to create 6 total struts. Two different connecting cables (one blue and one purple) between resulting struts are then also transformed by the same group for a total of 24 cables. b) Manipulation of the orientation of the generating strut creates potential variations of the figure. These require further analysis to measure any resulting instabilities.

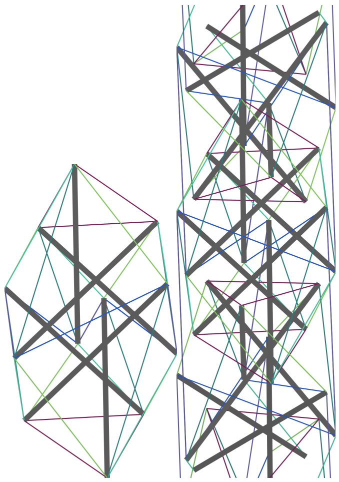


Figure 3.10: Potential tensegrities using $P\bar{3}\bar{2}$. Left, a point group model. Right, a space group tower model.

vectors in terms of the cell edges. We demonstrated the calculation of root vectors using the cosine angle law of spherical trigonometry, and developed a strategy for seeding the resulting symmetry generators in order to construct an operation sequence that could be easily adjusted with joining constraints such a glides and screws. We have made explicit through visualizations the spatial relationships entailed by these symmetries. We have enabled the construction of a system that can take any number of points as a motif and transform it according to the rules of the group. This formalism has granted us a quick way to generate 230 unique spatial configurations: an internationally standardized dictionary of arrangements.

The covariance of geometric algebra allows us to transform any motif, and we have explored one advantage of this in Section 3.4, where we generated groups of point pairs which could then be applied to some input mesh. The resulting deformations can be used to model surfaces that would otherwise be difficult to rationalize, some of potential use in machine design such as fan blades or gears. More details on the use of point pairs as generators of transformations can be found in Chapter 5.

Using symmetry groups to model some basic tensegrity systems has opened a way to generative exploration of other arrangements of tension and compressive forces. We study only the geometry here, and not the dynamics, so an important next step will be to try to add simulation of forces into our computer models and fabricate some of the models we synthesize to study them in real (physical) space. Additionally, we have only begun to explore the use of the point group and space group symmetries to generate these systems, and have not considered the full range of possibilities of all the space groups. These should be studied further for their contribution to the construction of tensegrity space structures.

Both non-Euclidean as well as higher dimensional reflection groups can be explored by extending these methods. In [34] Pierre Dechant has demonstrated how any 3D reflection group in $\mathscr{G}^3$, better known as a *root system*, induces a root system in 4D through the geometric product of its roots in what he calls a “spinor construction”. Dechant shows that even all *exceptional* root systems in 4D – which have no analogue in other dimensions – can be induced in this way, including the mysterious exceptional E_8 group.

Chapter 4

Kinematics: Constructing Linkage Mechanisms

> It is in the nature of man that he makes music. It is in the nature of man that he makes machinery too.
>
> – Jack Phillips, *Freedom in Machinery*
>
> Machines are nothing but certain figures, so I can describe them with these characters, and I can explain the change of situation that can occur in them, that is, their movement.
>
> – Leibniz

4.1 Summary

In the previous chapter on crystallographic space groups, the last joining constraints we considered – screw displacements – were constructed by multiplying a rotation in a plane by a translation partway along an edge or diagonal normal to the plane. In this chapter, we will explore this type of screw transformation more carefully, through the use of **motors** as multipliers for **forward kinematics** as outlined in [9]. We will introduce mathematical formulations for procedures solving two different **inverse kinematics** problems: in Section 4.4.1 we detail equations for an *iterative* solution to a rotationally constrained chain of revolute joints based on the FABRIK (Forward-And-Backward-Reaching-Inverse-Kinematics) approach outlined in [1, 4, 3], and in Section 4.4.2 we offer a *closed-form* solution to the Bennett linkage mechanism, drawing upon the work of Bayro-Corrochano [6] in designing geometric constraint-based systems of control, of which Hildenbrand [76] has a good description. We then generate some structures that can be designed from neworked linkage mechanisms as suggested by You and Chen in [138].

In Section 2.6 we introduced the modelling of constrained movement via the geometric intersections of rounds, building on the reader's intuition of compass and ruler constructions. Since rigid (*isometric*) structural constraints are distance-based, they can be represented as spheres, and possible solutions can be encoded by their intersections. Here we further explicate how relationships between points, circles, spheres, lines and planes that are constructed with the closed-form implicit equations of geometric algebra can be used to model structures that move. This provides a system for articulating forms using geometric primitives as constraint parameters, greatly facilitating intuitive design of kinematic structures such as are used in mechanical linkages and robotics.

William Clifford's interest in motion led to his study of biquaternions, now called dual quaternions, which can generate twisting motions. These are isomorphic to Plücker coordinates. Since Hestenes' treatment on classical mechanics [66], the use of Clifford algebras for rigid body motion analysis was explored by Bayro-Corrochano and Lasenby in [11], and explorations of kinematics using Euclidean GA can be found in Lasenby et al in [91], and with McRobie in [101]. The kinematic design of robot manipulators with primitives was addressed by Bayro-Corrochano and Kähler in 2000 [9], where, outlining previous methods, the authors develop the **motor algebra** for forward kinematics, and a method of using flat geometric primitives (lines and planes) as constraints in the solution of inverse kinematic equations. In subsequent formulations [8, 6] Bayro-Corrochano extended the motor-based theory into the

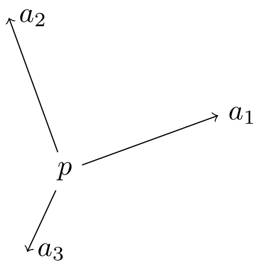


Figure 4.1: An orthonormal frame at point p.

conformal algebra in order to include round geometric primitives as constraints which could be intersected, and to incorporate computer vision analysis into the technique. The result is a unifying approach to **perception-action** systems, often applied to humanoid locamotion. For instance, Aristidou and Lasenby use their FABRIK constraint-based solver to model a hand and a human body, and Rosenhahn and Klette use CGA to analyze poses of a human model in [121].

4.2 Space Frames

In 3D graphics and CAD software, the combination of **position** and **orientation** is sometimes called a **pose**. Here, to help ground our work in the physics from which it came, we call such a point in space with orthonormal axes a **frame** (as in a *frame of reference*).[1]

Frames are parameterized by a position and orientation in the following way. With CGA, we use a Euclidean rotor $R = e^{-\frac{1}{2}\theta\mathbf{B}}$ to encode 3D orientation relative to $R = 1$ and a null point p to encode position. The Euclidean vector component $\mathbf{v}$ of p (where $p = n_o + \mathbf{v} + \frac{1}{2}\mathbf{v}^2 n_\infty$) can be used to construct a **translator** (Section 2.4.4) of the form $T = e^{-\frac{1}{2}\mathbf{v}n_\infty} = 1 - \frac{1}{2}\mathbf{v}n_\infty$. We can concatenate this description into a single multivector **motor**, which is a rotation followed by a translation, $M = e^{-\frac{1}{2}\mathbf{v}n_\infty} e^{-\frac{1}{2}\theta\mathbf{B}} = TR$, which is like the "screw displacement" we generated in the previous chapter except that now the translation direction $\mathbf{v}$ is not necessarily perpendicular to the plane of rotation $\mathbf{B}$. When generating screw displacements in the previ-

---

[1] The local axes of frames need not be orthonormal, though throughout this text we consider them to be.

ous chapter, we translated first and then rotated (in the form RT), but in fact the order was irrelevant because we were translating in a direction orthogonal to the plane of rotation and therefore **B** and **v** commuted. In Chapter 5 we will investigate how any motor $M = TR$ can be re-composed as the exponential of some dual line $M = e^{-\lambda}$, where the exponent represents the **dual line** of motion, also known as a **screw** or a **twist**, around which and along which the transformation occurs (see Figure 5.1 in Chapter 5 for an analysis of this representation of a line).

In this current chapter we restrict ourselves to building motors by multiplying two simple rotors, to encode a rotation followed by a translation: $M = TR$. When we refer to an frame's local $\mathbf{y}$ axis vector or $\mathbf{x} \wedge \mathbf{y}$ plane these are defined by applying the frame's orientation rotor R to the basis elements e_2 and e_{12} (respectively), such that $\mathbf{y} = R[e_2]$ and $\mathbf{x} \wedge \mathbf{y} = R[e_{12}]$, with $R[X]$ shorthand notation for the versor product $RX\tilde{R}$. In contrast to the Euclidean element $\mathbf{x} \wedge \mathbf{y}$ which is bound to the origin, we refer to the frame's homogeneous plane, translated in world space as Π_{xy}.

For a frame at the origin M_o, we say that $||\mathbf{v}|| = 0$ and $R = 1$. Applying M to any element X as $X^{'} \mapsto M[X]$ translates and rotates X into the **world space** of the reference frame. Conversely, applying M^{-1} to any element $X^{'}$ as $X \mapsto M^{-1}[X^{'}]$ inverts the reference frame M and returns an X in **object space** relative to the origin.

4.3 Kinematics

Rigid-body kinematics gains a full-featured friend in the conformal model. As we have seen in Section 2.4.4, an essential subalgebra, called the **motor algebra**, emerges in which translations and rotations are both multiplicative operators. We have explored the power of this algebraic mechanism in articulating (expressing) the relations of crystallographic space groups in Chapter 3, and now we will use it to relate the articulations (movements) of mechanisms. The subalgebra is isomorphic to **dual-quaternions** which have found favor within the robotic community, yet it enjoys the additional advantage of being able to operate on all the geometric numbers described in section 2.4.3 and listed in Table 2.6.

A **linkage** is an **open** (one end position fixed) or **closed** (both end positions fixed) **kinematic chain** of frames or combination of linkages whose movement constitutes a **mechanism**. Each **link** in a chain represents a relative transformation from one **joint** to the next, and the joints are categorized according to their allowed motion. According to Chasles' theorem, all motions can be thought of as a rotation around and translation along some screw axis, and joints are categorized into **lower** and **higher** pairs according to whether that screw axis remains fixed in relation to the links it couples. The lower pairs, in which the axis is fixed, can be revolute (R), cylindrical (C), helical (H), or prismatic (P).

Throughout this Chapter we will consider only **revolute** joints which rotate in their frame's $\mathbf{x} \wedge \mathbf{y}$ plane with one degree of freedom which we label θ. In the case of open chains, such as robot arm **manipulators** which do not form a loop, the individual joint rotations θ_i are not constrained by the others (except to the extent that the result avoids self-collision). Such manipulators are explored in Section 4.4.1. In the case of certain closed loops, such as the Bennett mechanism of Section 4.4.2, joint rotations are not independent and the arrangement is constrained or overconstrained.

In 1955 Denavit and Hartenberg [36] formalized the scalar parameters necessary to describe the configuration of a linkage mechanism based on matrix multiplication. In conformal geometric algebra we use the same parameters, but encode them as motors rather than matrices. Figure 4.2 illustrates the following parameterizations:

- α_i: the **skew** angle between joint j_i and joint j_{i+1}
- a_i: the **link** distance between joint j_i and joint j_{i+1}
- r_i : the translational **offset** of joint j_i along its own axis of rotation.[2]

---

[2] In many kinematics texts this offset is actually written R_i but we use lowercase to avoid confusion with our rotor notation.

These **geometric parameters** standardize the description of a k-chain and are fixed for revolute-only mechanisms. Additionally, to configure the particular state of such a chain we must also specify a **free** parameter:

- θ_i: the **rotation** of joint j_i about its z axis of rotation.

In the motor representation, this final free parameter is encoded as a relative local "in-socket" joint transformation about the e_{12} axis in object space as $R_\theta = e^{-\frac{1}{2}\theta e_{12}}$ whereas the first three parameters above are encoded as a relative link between joints in object space. We encode the link parameters as a single motor in object space in the following way:

- The skew parameter α_i is encoded as a rotation around e_{13} as $R_{\alpha_i} = e^{-\frac{1}{2}\alpha_i e_{13}}$.
- The link distance a_i is encoded as a direction along e_2 as $d_{a_i} = a_i e_2 n_\infty$.
- The translational offset r_i is a translation along e_3 axis as $d_{r_i} = r_i e_3 n_\infty$.

These encodings of α_i, a_i, and r_i combine to form a single relative link motor in object space

$$M_{L_i} = T_{d_{a_i}+d_{r_i}} R_{\alpha_i} \tag{4.1}$$

used to jump from M_i to M_{i+1}.

A frame in world space is computed by **composing** transformations in a method analogous to matrix method. Given our four parameters, to calculate the absolute (world space) position and orientation of the ith frame we calculate the motor transformation M_i that gets us there relative to the origin:

$$M_i = M_{i-1} M_{L_{i-1}} M_{J_i} \tag{4.2}$$

where $M_{L_{i-1}}$, defined in equation 4.1, is the relative link transformation from the previous frame and M_{J_i} is the local joint transformation R_{θ_i} at i. This **forward kinematic** composition specifies that we first consider the local joint rotation, followed by the link from the previous

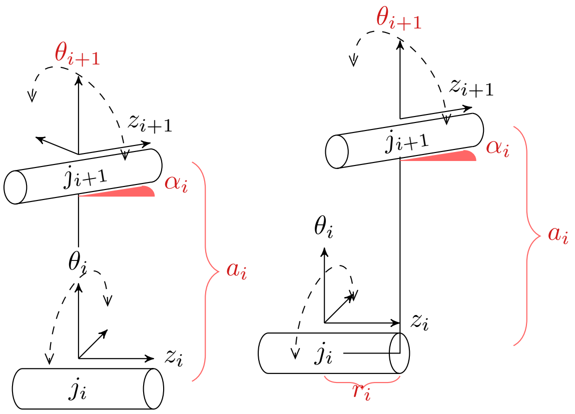


Figure 4.2: Key parameters describing a chain of revolute joints. θ_i the angle of rotation of joint j_i is a free parameter, whereas a_i the link length and α_i the link skew between joints j_i and j_{i+1} is typically a fixed geometric property of the mechanism. The right image illustrates a third fixed property r_i which specifies the offset position of joint j_i along its z axis of rotation.

frame, and finally the previous frame's absolute position and orientation in world space. As in matrix transformations, and mathematical function composition $(f \circ g)$, the *last* transformation in the equation is the *first* one to be calculated (see Figure 4.3). Note that links and joints are sufficiently encoded in object space using our basis elements, e_2, e_3, e_{12} and e_{13}. And that frames M_i are found through composition of these relative transformations.

To summarize:

- A **joint** M_J is an in-place transformation. In the case of *revolute* joints, this is encoded in object space as a rotation by θ in the e_{12} plane.

- A **link** M_L is the relative transformation required from joint j_i to the next joint j_{i+1}. This is encoded in object space as a skew rotation α about the e_{13} plane followed by a translation $d = ae_2 + re_3$. Note that this transformation jumps from joint j_i to the location and orientation of j_{i+1} before the in-socket θ_{i+1} transformation has been applied.

- A **frame** M is the calculated, absolute**,** world space position and orientation relative to the origin. In forward kinematics (Equation 4.2 and Figure 4.3) this is the result of

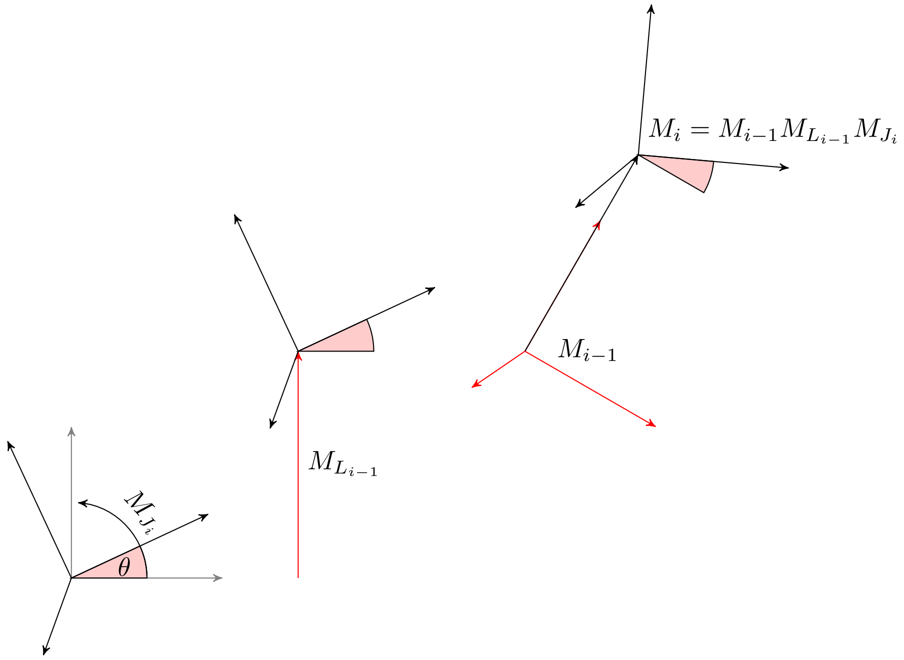


Figure 4.3: Forward kinematics of a k-chain series of frames M. From left to right, calculation of the absolute position and orientation of frame M_i requires calculating its local in-socket transformation M_{J_i}, followed by its link displacement $M_{L_{i-1}}$ relative to the the previous frame, and finally the previous frame itself, M_{i-1}. This sequence of operations is composed as $M_i = M_{i-1}M_{L_{i-1}}M_{J_i}$.

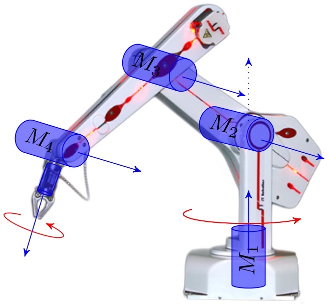


Figure 4.4: A robot arm can be modelled as a kinematic chain of frames.

compounding the transformations of all previous frames.

- A k**-chain** is a series of frames numbered $\{M_0, ..., M_{k-1}\}$.

Thus, in a representation of a revolute chain, each frame M_i has a joint M_{J_i} specifying the in-socket rotation and a link M_{L_i} specifying the transformation to the next frame.

4.4 Inverse Kinematics

A common problem in robotics and mechanism design and modelling involves knowing both a **target** position for the **end effector** – the last frame in the kinematic chain – and a **base** position for the first frame, and then determining the joint angles for all frames to satisfy that criteria. Solutions to this **inverse kinematics** question are typically either analytical, closed form and exact, or else iterative and error-reducing. In this section we present examples for both approaches, the first an iterative solution to an open chain consisting of an arbitrary number of revolute joints, and the second an analytical solution for determining the configuration of a Bennett linkage closed 4-bar revolute skew mechanism.

In our geometric formulations, we will find it necessary to add the ability to extract geometry from the space frame we constructed in Section 4.2 as a position p and orientation R, and combined motor representation $M = TR$. We will call σ the sphere at p with radius equal to the link length $||d_a + d_r||$ (see Figure 4.6). This sphere represents the distance constraint of frame M_i on frame M_{i+1}. The plane of rotation Π is the $e_{12}n_on_\infty$ blade transformed by the frame's motor representation as $\Pi = M[e_{12}n_on_\infty]$. Similarly, it is useful to cast the link offset direction re_3n_∞ in the frame of reference as $r_M = R[re_3n_\infty]$. This offset together with the plane adds an additional constraint on the position of M_{i+1}, namely $T_{r_M}[\Pi]$. Note that this can just as easily be written as $M[T_r[e_{12}n_on_\infty]]$ which in essence translates the $e_{12}n_on_\infty$ plane along the e_3 offset before transforming it into the frame of reference. As shown in Figure 4.6, the distance constraint σ along with the dual of the plane constraint $T_{r_M}[\Pi]^*$ can be wedged together to determine a circle of possible positions for the M_{i+1} frame. Algorithm 4.2 details this construction, and Algorithm 4.1 expresses a method for constraining a point to lie on a circle.

When calculating the circle of possible positions for the previous M_{i-1} frame, care is required when accounting for skew parameters α, in order to ensure constructions are built in the correct frame of reference. Similarly, when calculating joint angles, it is necessary to consider the contribution of the link offset and distance parameters r and a. With these details in mind, Algorithms 4.3 and 4.4 offer methods for extracting the circle of possible positions of the next or previous frame in a chain of revolute joints. These work for arbitrary skews and offsets. It should be noted that the algorithm we provide for finding the orientation of the previous frame takes into consideration its location.

Algorithm 4.1 Constrain a Point to a Circle

```
procedure CONSTRAINPOINTTOCIRCLE(Point $p$, Circle $K$)
    $\Pi \leftarrow K \wedge n_\infty$                    ▷ Carrier plane of circle (Equation 2.51).
    $\sigma \leftarrow K/\Pi$                             ▷ Surround sphere of circle (Equation 2.61).
    $p' \leftarrow (p \cdot \Pi)/\Pi$                     ▷ Projection of $p$ onto carrier plane.
    $\Lambda \leftarrow \sigma \wedge p' \wedge n_\infty$  ▷ Line through center of surround and $p'$.
    $\kappa \leftarrow (\Lambda^* \wedge \sigma)^*$       ▷ Point Pair meet of surround and line (Equation 2.58).
    return $\frac{\kappa + \sqrt{|\kappa^2|}}{-n_\infty \cdot \kappa}$   ▷ Point on $K$ closest to $p$ (Equation 2.57).
end procedure
```

Algorithm 4.2 Calculating Circle of Positions of Next $(i+1)$th Frame

```
procedure NEXTCIRCLE
    $p \leftarrow$ Position of $i$th Frame
    $\mathbf{v} \leftarrow d_a + d_r$                    ▷ Vector of link offset and length.
    $\sigma \leftarrow p - \frac{1}{2}\|\mathbf{v}\|^2 n_\infty$                    ▷ Sphere at $p$ with radius equal to $\|\mathbf{v}\|$
    $reject \leftarrow (\mathbf{v} \wedge e_{12})/e_{12}$                    ▷ Rejection of link vector from plane of rotation.
    $rotRej \leftarrow R_i[reject]$                    ▷ Rejection Rotated into Frame's Coordinates.
    $\Pi \leftarrow T_{rotRej}[\Pi_{xy}]$                    ▷ Translation of $\Pi_{xy}$ plane along rejection.
    return $K_{i+1} \leftarrow (\sigma \wedge \Pi^*)^{-*}$                    ▷ Meet of Plane of Rotation and Distance Sphere.
end procedure
```

Algorithm 4.3 Calculating Rotation of Previous $(i-1)$th Frame

```
procedure PREVROTOR
    $\mathbf{v} \leftarrow R_i^{-1}[p_i - p_{i-1}]$                    ▷ Direction from previous frame (relative to previous frame).
    $\mathbf{link} \leftarrow R_{\alpha_{i-1}}^{-1}[d_{a_{i-1}} + d_{r_{i-1}}]$                    ▷ Link vector (relative to previous frame).
    $\mathbf{proj_v} \leftarrow (\mathbf{v} \cdot e_{12})/e_{12}$                    ▷ Projection of $\mathbf{v}$ onto $e_{12}$ blade.
    $\mathbf{proj_{link}} \leftarrow (\mathbf{link} \cdot e_{12})/e_{12}$                    ▷ Projection of $\mathbf{link}$ onto $e_{12}$ blade.
    $R_{\text{rel}} \leftarrow \sqrt{\frac{[\mathbf{proj_v}]\text{normalized}}{[\mathbf{proj_{link}}]\text{normalized}}}$                    ▷ Rotor taking link vector to direction vector
    return $R_i R_{\text{rel}} R_{\alpha_{i-1}}^{-1}$
end procedure
```

Algorithm 4.4 Calculating Circle of Positions of Previous $(i-1)$th Frame

```
procedure PREVCIRCLE
    $\mathbf{v} \leftarrow d_{a_{i-1}} + d_{r_{i-1}}$                    ▷ Vector of previous link offset and length.
    $\sigma \leftarrow p - \frac{1}{2}\|\mathbf{v}\|^2 n_\infty$                    ▷ Sphere at $p$ with radius equal to $\|\mathbf{v}\|$
    $\mathbf{reject} \leftarrow (\mathbf{v} \wedge e_{12})/e_{12}$                    ▷ Rejection of link vector from $e_{12}$ blade.
    $R \leftarrow$ PrevRotor()                    ▷ Previous rotation from Algorithm 4.3.
    $\mathbf{n} \leftarrow R[e_3]$                    ▷ $\mathbf{z}$ axis of rotation in previous Frame of reference.
    $\pi \leftarrow T_{p_i - R[\mathbf{reject}]}[\mathbf{n}]$                    ▷ axis translated by position and link rejection.
    return $K \leftarrow (\sigma \wedge \pi)^{-*}$                    ▷ Meet of Plane of Rotation and Distance Sphere.
end procedure
```

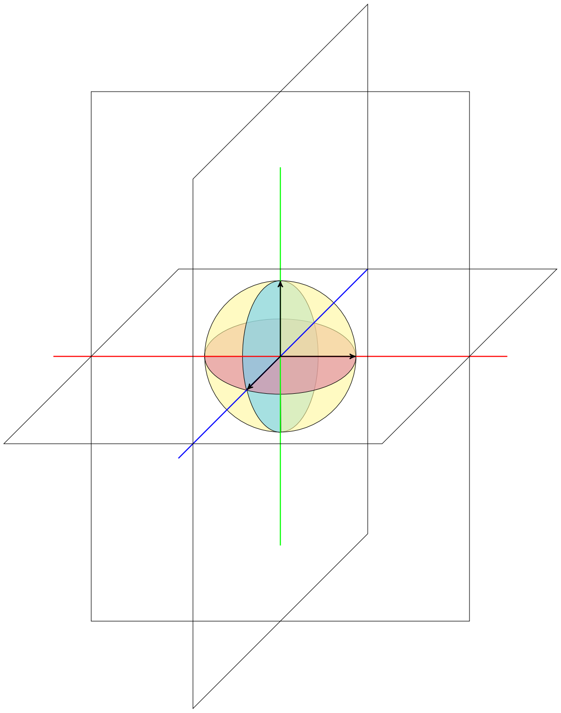


Figure 4.5: Geometric content of a frame of reference includes the orthogonal planes, line axes, circles of rotation, and surrounding sphere.

4.4.1 Iterative Construction of Revolute k-Chains

We will use the geometric construction of revolute constraints as the basis for implementing a version of a conceptually simple iterative technique called Forward-And-Backward-Reaching-Inverse-Kinematics [FABRIK]. FABRIK was proposed by Aristidou and Lasenby in [1] as a method for motion capture – a way of representing a configuration of joint positions of a chain given some set of 3D positions. In a subsequent work they modelled a human hand and a human skeleton. The approach has recently gathered attention in the robotics community for its intuitive construction and flexibility in solving a range of problems, and can be used to constrain the positions of an open or closed kinematic chain using coincidence relationships of round elements such as circles and spheres. Here we will detail an iterative solver for an open chain with an arbitrary number of revolute joints using this technique. The basic principle behind the technique is to satisfy constraints by reaching forward toward some target, and then re-solve them by reaching backwards to the root of the chain, and then

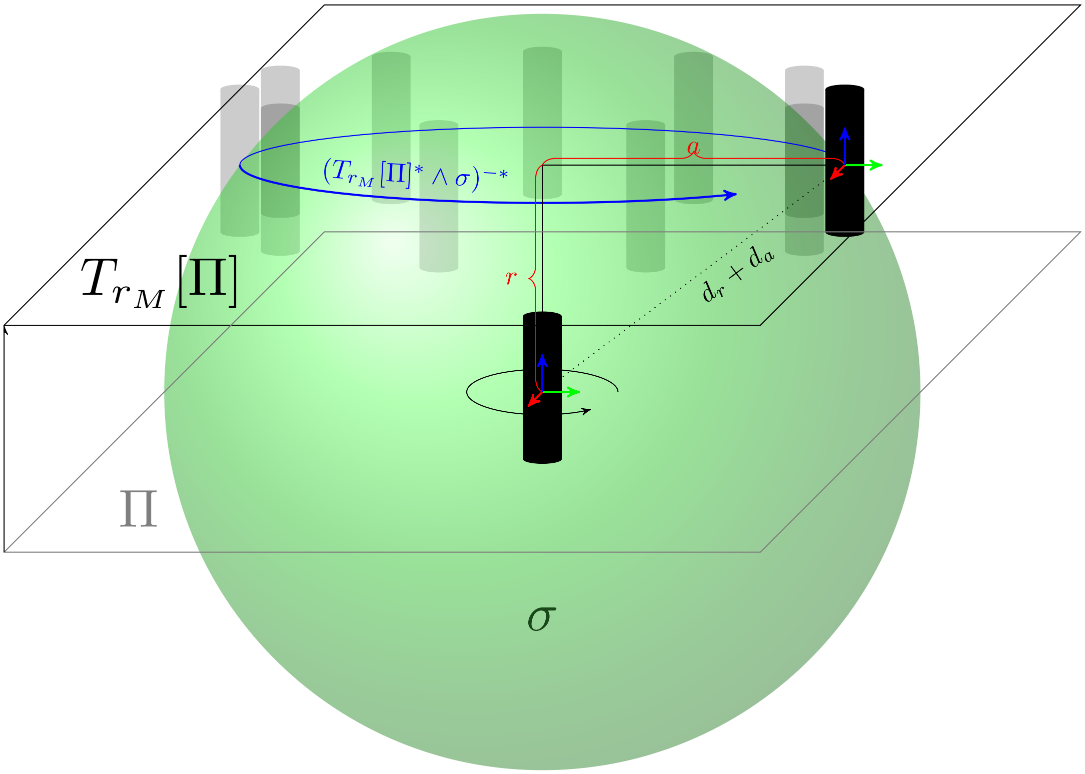


Figure 4.6: Given the ith frame in chain of revolute joints, we can use its a_i and r_i link parameters to construct geometric elements which encode the possible positions of the $(i+1)$th frame. Here the rotation plane is Π which can be translated along the offset r and then intersected with the surrounding sphere σ to create the circle of possible positions of the $(i+1)$th frame. Note that in this figure the skew parameter, α_i, is 0. See Algorithms 4.2,4.3, and 4.4 for more details on how to extract this circle from link parameters.

repeating the process until the end effector is within some error threshold of the target. Note that there is no guarantee of convergence, though in practice its users report stable results. Our algorithm is similar in concept to Algorithms 2 and 3 of [1], though whereas those descriptions model bone-like chains, here we contribute specific CGA mathematical equations for k-chains in terms of Denavit-Hartenberg parameters visualized in Figure 4.6.

Basic Forward and Backward Approach with Spherical k-chains Before we tackle this problem, it will help to review the basic technique of the FABRIK method by assuming that all joints in our chain are universal ball joints, and therefore are constrained only by spheres, not planes. Algorithm 4.5 and Figure 4.7 detail this most basic method: we fix the last frame position, p_n, to the desired target and "reach forwards" to constrain the p_{i-1} position to lie on the sphere of radius a_{i-1} at p_i. We iterate to the root of the chain. Then, we fix the first frame position p_0 to the desired base and "reach backwards" to constrain the p_{i+1} position to lie on the sphere of radius a_i at p_i. We iterate towards the end effector. Note that the terminology can seem counterintuitive: "reaching forwards" entails iterating backwards from the end effector, while "reaching backwards" entails iterating forward from the base of the chain.

We can define these spheres using the definition of a sphere at point p_i with radius r from Equation 2.52, and calculate the point on the sphere closest to p_{i+1} by creating a line through p_i and p_{i+1} and intersecting it with the sphere. We then use Equation 2.57 to extract the point closest to p_{i+1} from the resulting point pair intersection: for a line $\Lambda = p_i \wedge p_{i+1} \wedge n_\infty$ intersecting a sphere σ centered at p_i we can find this point by extracting the second point in the resulting point pair intersection κ from Equation 2.58, namely: $\frac{\kappa+\sqrt{|\kappa^2|}}{-n_\infty \cdot \kappa}$.

This particular distance constraint could easily be written with typical vector math, and the mechanics of geometric algebra may seem redundant here. The real power lies in adding additional constraints, such as only allowing the joints to rotate in a single plane.

Backward iteration of Revolute k-Chains We have seen (Section 2.6) that a rotation in a plane of fixed distance a can be encoded as a circle, and so we can use the circle as a geometric constraint. Given a point p_{i+1} and a circle K at p_i we can find the point on K closest to p_{i+1}. Aristidou and Lasenby provide a method that uses the midpoint of the reflection of p_{i+1} in the carrier plane Π of K, where $\Pi = K \wedge n_\infty$. In Algorithm 4.1 we simply use the projection of p_{i+1} onto this carrier plane of K. Whichever method is used, that projected point, call it p_{i+1}^{proj}, can then be used to construct a line that is intersected with a sphere at p_i with radius a_i as in the spherical approach.

We note that since in the case of inverse kinematics chains the circle K is derived from its known center point q, orientation R, and radius r, an even more computationally efficient option to constrain p_{i+1} is to project the Euclidean vector $\mathbf{d}$ between points p_{i+1} and p_i onto the Euclidean 2-blade $\mathbf{x} \wedge \mathbf{y}_i = R[e_{12}]$ of the frame at p_i:

$$\mathbf{v} = ((p_{i+1} - p_i) \cdot (\mathbf{x} \wedge \mathbf{y}_i))/(\mathbf{x} \wedge \mathbf{y}_i).$$

We can then find a point pair by a construction using (normalized) Euclidean vector $\mathbf{v}$ and the sphere σ surrounding the circle of possible positions of p_{i+1}:

$$\kappa = \sigma \wedge (\sigma \cdot (\mathbf{v} n_\infty))$$

which is based on Equation 2.59 and illustrated in Figure 2.13. Extracting of the point closest to p is then straightforward using Equation 2.57. In the case of links with an offset parameter, this sphere σ is not necessarily centered at p_i.

Unlike the spherical case, where in-joint rotations can be determined after all positions have been determined, at each step of the revolute algorithm we must also set the rotor orientation R_i of each frame in addition to the position p_i. When reaching backwards to satisfy the base position constraint, the joint orientation is found by projecting the target direction onto the plane of rotation. When reaching forwards we use Algorithm 4.3, which compares

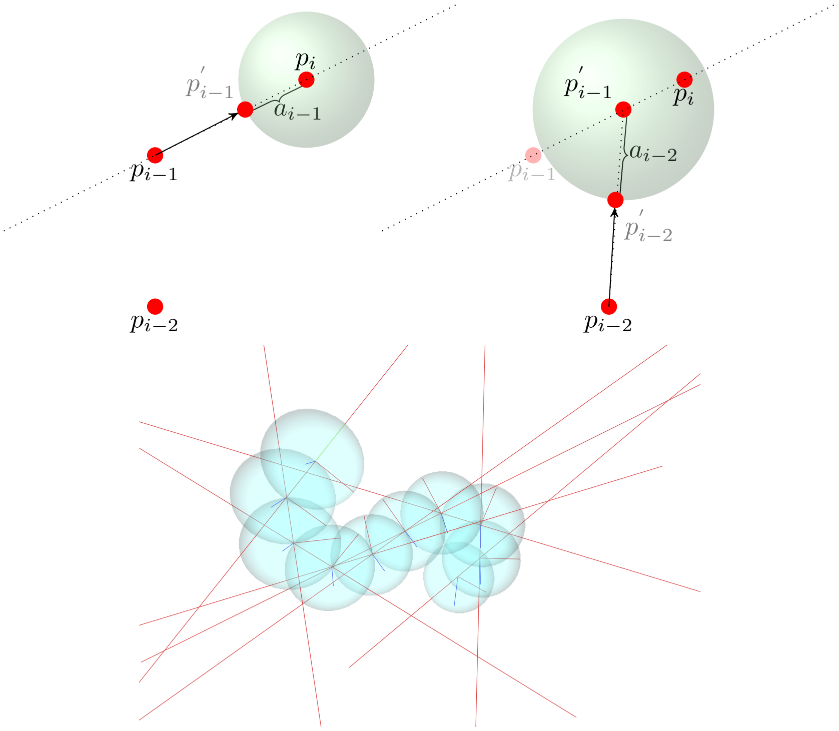


Figure 4.7: The forward reaching step of the FABRIK method for spherical joints constrains frames at p_{i-1} to lie within the a_{i-1} radius of p_i. Algorithm 4.5 details the process used to calculate this geometrically as the intersection of the line through p_{i-1} and p_i and the sphere at p_i with radius a_{i-1}. Using this method, a chain of ball joints can be represented as a series of forward-constraining-spheres centered at i going through $i-1$, and a series of backward-constraining-spheres centered at i going through $i+1$.

the projection of the link vector and target vector, each transformed to be in the frame of reference of the $i-1$th frame.

Because the calculations depend on the current orientation of each frame, it can be helpful to slowly rotate the target end effector about its **y** axis in order to find a suitable solution, thereby changing the **z** axis direction and relaxing the link constraint. This method facilitates reaching a specified target position of the end effector (but not a target orientation).

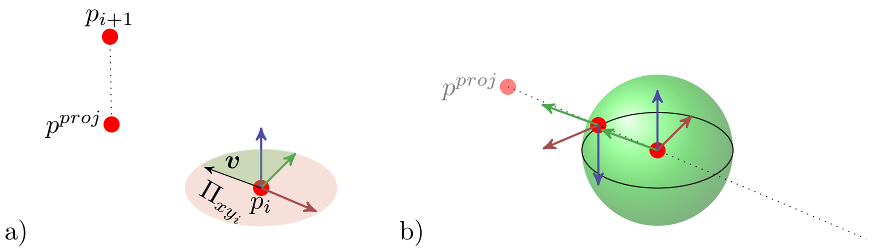


Figure 4.8: An implementation of the backward reaching step for chains composed of revolute joints from Algorithm 4.6. Each p_{i+1} frame position is constrained to lie on the circle defined by the R_i frame Π_{xy} orientation. We project p_{i+1} onto the Π_{xy} plane of the previous frame, and then calculate the the line through that projected point and the p_i position. We intersect this line with the sphere at p_i with radius a_i.

Algorithm 4.5 Inverse kinematics of a chain of **spherical** joints

```
procedure SPHERICAL FABRIK(Base,Target)
    while $|p_n \cdot \text{target}| > \text{error}$ do                ▷ While end frame is too far from target
        Point: $p_n \leftarrow$ Target                                  ▷ Place end frame at Target Position
        for $p_i \in$ Frames (Starting at $p_n$) do                      ▷ Forward Reaching Iteration Starting at $p_n$
            $\sigma \leftarrow p_i - \frac{1}{2}a_{i-1}^2 n_\infty$                           ▷ Possible positions of $p_{i-1}$
            $\lambda \leftarrow p_i \wedge p_{i-1} \wedge n_\infty$                         ▷ Line through $p_i$and current $p_{i-1}$
            $\kappa \leftarrow (\lambda^* \wedge \sigma)^*$                              ▷ Point Pair intersection of $\lambda$ and $\sigma$
            $p_{i-1} \leftarrow \frac{\kappa+\sqrt{|\kappa^2|}}{-n_\infty \cdot \kappa}$                   ▷ Point of $\kappa$ closest to current $p_{i-1}$
        $p_0 \leftarrow$ Base                                            ▷ Place first frame at Base Position
        for $p_i \in$ Frames (Starting at $p_0$ ) do                     ▷ Backward Reaching Iteration Starting at $p_0$
            $\sigma \leftarrow p_i - \frac{1}{2}a_i^2 n_\infty$                             ▷ Sphere at $p_i$ with radius $a_i$
            $\lambda \leftarrow p_i \wedge p_{i+1} \wedge n_\infty$                         ▷ Line through $p_i$ and current $p_{i+1}$
            $\kappa \leftarrow (\lambda^* \wedge \sigma)^*$                              ▷ Point Pair intersection of $\lambda$ and $\sigma$
            $p_{i+1} \leftarrow \frac{\kappa+\sqrt{|\kappa^2|}}{-n_\infty \cdot \kappa}$                   ▷ Point of $\kappa$ closest to current $p_{i+1}$
    Calculate Joint Rotations Based on Positions (Algorithm 4.8)
end procedure
```

4.4.2 Analytical Construction of the Bennett Linkage

Certain cases of closed chains, where the last frame is linked to the first, are **overconstrained.** An overconstrained mechanism is one that “paradoxically” fails to pass the Grübler-Kutzbach mobility equation, and yet still moves with at least one degree of freedom. For a spatial mechanism the mobility equation is:

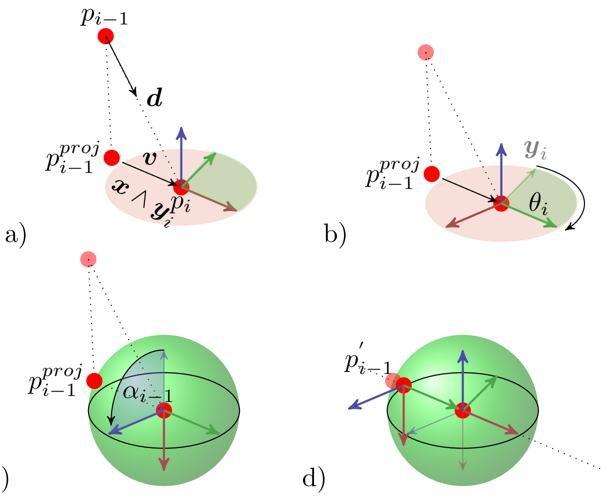


Figure 4.9: An implementation of the forward reaching step for revolute joints from Algorithm 4.6. a) Given a frame at p_i in position R_i, the unit vector $\mathbf{v}$ is the difference between p_i and the projection of p_{i-1} onto the frame's Π_{xy} plane. $\mathbf{v}$ can also be more efficiently formulated as the projection of of the vector $\mathbf{d}$ from p_{i-1} to p_i onto the Euclidean $\mathbf{x} \wedge \mathbf{y}_i$ plane. b) Frame is in position $R_{\theta_i}^{-1} R_i$ where R_{θ_i} is the rotation that takes $\mathbf{v}$ to $\mathbf{y}_i$ c) Frame is in postion $R_{\theta_i}^{-1} R_i R_{\alpha_{i-1}}^{-1}$. d) Frame has been placed at a position derived from the sphere σ with radius a_{i-1} and Euclidean vector $\mathbf{v}$ using Equation 2.59.

Algorithm 4.6 Inverse kinematics of a chain of **revolute** joints (See Figures 4.8 and 4.9)

```
procedure CIRCULAR FABRIK(Base,Target)
    while $|p_n \cdot \text{Target}| > \text{error}$ do                ▷ While end frame is too far from target
        $p_n \leftarrow$ Target                                        ▷ Place end frame at target
        for $p_i \in$ Frames (Starting at $p_n$) do                    ▷ Forward-reaching Step
            $R_{i-1} \leftarrow$ PrevRotor()          ▷ Set rotation of previous Frame (Algorithm 4.3).
            $K_{i-1} \leftarrow$ PrevCircle()   ▷ Circle of positions of previous Frame (Algorithm 4.4).
            $p_{i-1} \leftarrow$ ConstrainPointToCircle($p_{i-1}, K_{i-1}$)          ▷ Algorithm 4.1.
        $p_0 \leftarrow$ Base                                          ▷ Place first frame at Base Position
        $R_t \leftarrow 1$                                             ▷ Set Rotation to identity
        for $p_i \in$ Frames (Starting at $p_0$ ) do                   ▷ Backward-reaching step
            $K_{i+1} \leftarrow$ NextCircle()        ▷ Circle of positions of next Frame (Algorithm 4.2).
            $p_{i+1} \leftarrow$ ConstrainPointToCircle($p_{i+1}, K_{i+1}$)          ▷ Algorithm 4.1.
            $\mathbf{v} \leftarrow p_{i+1} - p_i$                                  ▷ Vector from $p_i$ to $p_{i+1}$
            $\mathbf{v}' \leftarrow [(\mathbf{v} \cdot (\mathbf{x} \wedge \mathbf{y}_i))/(\mathbf{x} \wedge \mathbf{y}_i)]_{\text{normalized}}$   ▷ Vector projected onto frame's $\mathbf{x} \wedge \mathbf{y}$ plane.
            $\mathbf{v}'' \leftarrow R_t^{-1}[\mathbf{v}']$                          ▷ Vector relative to origin
            $R_{\theta_i} \leftarrow \sqrt{\frac{\mathbf{v}''}{e_2}}$                    ▷ $i$th Joint rotation takes $e_2$ to $\mathbf{v}''$
            $R_i \leftarrow R_t R_{\theta_i}$                                  ▷ Final Rotation of $i$th Frame
            $R_t \leftarrow R_i R_{\alpha_i}$                       ▷ Preliminary Rotation of $i$th $+ 1$ Frame
end procedure
```

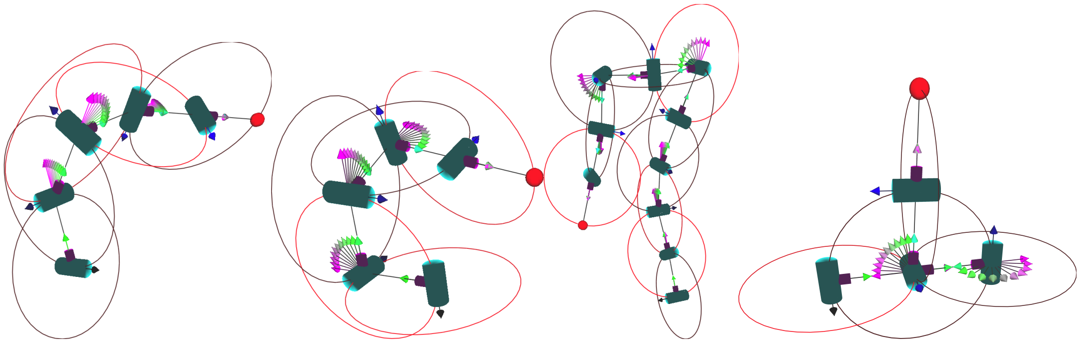


Figure 4.10: Algorithm 4.6 applied to a revolute chain, with $\alpha_i = \frac{\pi}{2}$. The arrow fans represent in-socket joint rotations, and the circles illustrate the possible positions of the subsequent joints. On the right is an illegal configuration demonstrating the need for additional collision testing.

$$\text{Degrees of Freedom} = 6(n-1) - \sum_{i=1}^{5}(6-i)p_i \tag{4.3}$$

which gives the **degrees of freedom** of a mechanism with n links, where p_i are the number of joints with i degrees of freedom. In, [114] Phillips discusses the role overconstrained mechanisms have in engineering: primarily adding strength and stiffness (and yet requiring more accurate construction). In [130], Tachi names two advantages of overconstrained *quadrilateral* meshes (such as discussed in Section 2.6) – their ability to be actuated with a single motor and their redundant – i.e. robust – construction. As explored by You and Chen in [138, 22], overconstrained linkages can be networked together to make structures that deploy with one degree-of-freedom – we experiment with the design of such mechanisms in the next Section. Since such mechanisms can be modelled geometrically, we use geometric algebra to help with a closed-form, analytical, construction. Here we explore the Bennett 4-bar linkage, in Phillips' words the "most famous and mysterious overconstrained yet mobile loop of them all."

Bennett proposed an overconstrained 4-bar skew mechanism in 1903 [14], a few years after Bricard introduced a variety of six-bar linkages. In 1931 Myard published a *Contribution à la géométrie des systèmes articulés*, which included 5-bar overconstrained spatial mechanisms. The kinematics of these overconstrained linkages have been greatly investigated but they have been relatively underused in design. An examination of these and other linkages can be found in Phillips [114], along with methods for designing more complicated mechanisms from the Bennett linkage mechanism. See also Baker, e.g. on Bricard linkages in [5].

Our contribution here is to demonstrate how we can construct a model of one such mechanism using only our geometric intuition and what we have covered in this text thus far. In particular, we greatly simplify the algorithm for specifying positions of a Bennett mechanism, alleviating the burden of working out elaborate trigonometric equations and instead solving constraint requirements through closed-form geometric constructions. Additionally, we avoid the need to recalculate our formulation in the case of a change of geometric parameter such as the offset r. We synthesize the mechanism using the motor algebra and intersections of spheres and planes. We will first specify the parameters of the linkage: the relative transformations from joint to joint, then use those constraints to calculate the final absolute positions of the frames, and only then calculate each joint's local in-socket transformation.

The Bennett linkage is a 4-bar skew spatial mechanism connected by 4 revolute joints. Plugging in $n = 4$ and $i = 1$ (revolute joints have one degree-of-freedom), the mobility equation for a Bennett linkage returns $6(4-1)-(5*4) = -2$. Therefore in theory it cannot move, however its geometric parameters actually allow it one degree of freedom. In relation to Figure 4.2, in a standard Bennett linkage there is no offset r, and the two main components to parameterizing a Bennett linkage are the relative skew angles α_i and link lengths a_i. A single equation serves to restrict the mechanism's parameters:

$$sin(\alpha_2) = -sin(\alpha_1) * \frac{a_2}{a_1} \tag{4.4}$$

and alternating link parameters are equal such that $\alpha_{3=}\alpha_1$, $\alpha_4 = \alpha_2$, $a_3 = a_1$, and $a_4 = a_2$. This equation be understood to mean that consecutive links skew the plane of rotation in

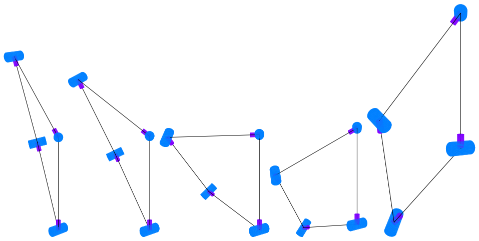


Figure 4.11: Spatial movement of a Bennett mechanism. In this case a_i are all equal and $\alpha_2 = -\alpha_1$. Revolute joints are depicted in blue.

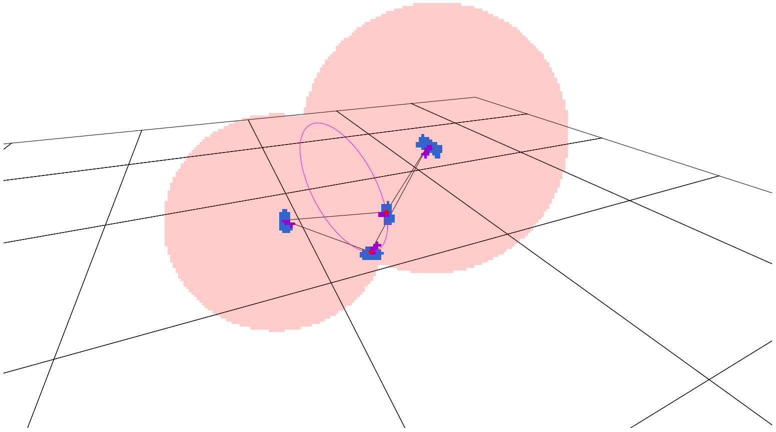


Figure 4.12: With parameters α_i and a_i specified according to the single equation 4.4, the constituting relations of a Bennett mechanism can be determined using the intersection of two spheres and a plane. This is a novel, closed-form, geometric solution that avoids the need to solve Denavit-Hartenberg closure equations with complex trigonometric relationships.

opposite directions by a factor porportional to the ratio of link lengths. In the case that $\alpha_i = 0$, the mechanism becomes a planar mechanism. Note as well that in the case that $a_1 = a_2$ all link lengths are equal and the equation simplifies to $\alpha_2 = -\alpha_1$. Figure 4.11 illustrates the movement of such a mechanism.

Solving the configuration of a Bennett mechanism consists of determining, for a given input angle state θ_1, all the other angles θ_2, θ_3, and θ_4 and the frame positions of that state. Our deductive, geometric method is constructed in Algorithm 4.7 and illustrated in Figure 4.12. We first find the positions of all frames and then calculate the joint angles necessary to achieve those positions.

Algorithm 4.7 Closed-form solution to the Bennett Linkage

procedure BENNETT

1. Given a frame M_1 at the origin with joint rotation θ_1 calculate a temporary position and orientation of M_2 as $M_{2_{temp}} = M_{J_1} M_{L_1}$. Note that we do not yet know the joint angle at M_2, but that we do know the position of M_2 as well as the *plane of rotation* in which θ_2 will act.
2. Calculate the position and orientation of M_4 as $M_4 = M_{L_4}^{-1}$, that is, as the inverse of the relative transformation of the final link. Note that while we have not yet calculated the specific joint angle θ_4, we *have* determined the absolute (world space) position and orientation of M_4 which includes the joint angle.
3. With the positions of frame M_2 and M_4 thus determined, calculate the circle of possible positions for frame M_3 as the intersection of the two spheres, one centered at M_2 with a radius of a_2 and the other centered at M_4 with a radius of a_3.
4. Intersect this circle with the plane of rotation of $M_{2_{temp}}$. This returns two points; pick one. (Note one of the points may be the position of frame M_1).
5. With all positions now specified, we can calculate the in-socket joint angles R_{θ_i} using Algorithm 4.8.

end procedure

Algorithm 4.8 Calculation of Joint Angles based on Positions

```
procedure CALCJOINTS
    $R_t \leftarrow 1$                                                          ▷ A Rotor set to Identity
    for $p_i \in$ Frames do
        $R \leftarrow R_t$                                                      ▷ Rotor at ith Frame
        $\mathbf{v} \leftarrow (p_{i+1} - p_i)_{\text{normalized}}$             ▷ The unit target vector
        $\mathbf{proj} \leftarrow (\mathbf{v} \cdot (\mathbf{x} \wedge \mathbf{y}_i)) / (\mathbf{x} \wedge \mathbf{y}_i)$    ▷ Projection onto frame's x ∧ y plane.
        $\mathbf{target} \leftarrow R_t^{-1}[\mathbf{proj}]$                    ▷ The target vector relative to origin
        $R_{\theta_i} \leftarrow \sqrt{\frac{\mathbf{target}}{e_2}}$             ▷ In-Socket Joint rotation (equation 2.38)
        $R_t \leftarrow R_t R_{\theta_i} R_{\alpha_i}$                          ▷ Rotor of next Frame
end procedure
```

4.5 Application: Deployable Structures

In the literature, the most common uses of CGA kinematic methods encompass the perception-action cycle: image and motion capture and robotic articulation. We have described a method for determining joint positions and rotations of a chain of revolute joints given some target end effector position. Executing this model in real-time on a computer can be used to control the motions of a real-world robot manipulator such as a grasping arm, or conversely to control movements of a virtual simulation by following motion captured points. A few other potential applications present themselves as well:

- **Computational origami**. As suggested in Section 2.6, a crease pattern on paper can be considered a network of revolute joints. By creating loops of revolute constraints it may be possible to increase proficiency in modelling the movements of foldable surfaces, perhaps by coupling the results with a relaxation step.

- **Protein Folding**. Kinematic models of proteins are sometimes used to try to find their tertiary structures – possible low energy configurations of the molecules. These chains of molecules are typically represented as chains of revolute joints (the backbone), with additional revolute side chains. Representing these chains as a series of forward and backward reaching circles may offer new computational advantages.

- **Structural Engineering.** Stadium roofs, arterial stents, satellite dishes, and mobile habitats, are all examples of structures which are sometimes designed to transform between two distinct configurations: compactly packaged folded state and an efficiently opened operational unfolded state.

These categories of articulating spaces are sometimes called **deployable structures**. While the term tends to refer to aerospace applications, where the need for transportable payloads which can be opened automatically is prevalent, the same geometries of folding can be applied to the design of expandable volumes at any scale. Deployable structures are also frequently found in nature, including twisting and untwisting DNA strands, unfurling moth tongues, opening insect wings, and inflating lungs [20, 85]. Indeed, Miura's pattern from Section 2.6 is said to have been inspired by the herringbone pattern of an unfolding hornbeam leaf [103].

Spatial mechanisms such as the Bennett mechanism, in which the axes of the revolute joints are not parallel, are more complicated than planar mechanisms, in which the movement is restricted to the plane.[3] As You and Chen explain in *Motion Structures*, the vast majority of linkage designs on this planet are *planar* mechanisms, meaning they do not articulate in 3D space; their paths of motion are restricted to a 2D plane. 3D articulations can then be constructed by combining articulations in two directions, as in the case of the Hoberman sphere. Similarly, Guest, in his 1994 PhD thesis on the deployment of cylinders and membranes, explains that most space structures are 1 or 2 dimensional – either articulating beams or struts, or else solar sails or arrays and antennae.[4]

In [22], Chen and You discuss the under-utilized 4-bar, 5-bar, and 6-bar spatial linkage mechanisms, of which there are 15 (and growing), and propose their increased use in deployable structure design. In particular, in their book *Motion Structures*, You and Chen investigate the many forms that can be created using a network of Bennett linkages [138], some examples of which we will generate in this section. The Bennett mechanism can be linked to form networks of motion, where the movement is itself the target of realization.

The use of closed loop linkages for the construction of deployable structures was first investigated by Gan and Pellegrino in [56], though as they mention, they found previous work in Crawford, R. F., Hedgepeth, J. M. and Preiswerk, P. R. (1973). In [109], Pellegrino and Vincent consider the problem of packaging flat membranes.

---

[3] Configurations of planar linkages are investigated by Demaine in his dissertation [35].

[4] In aerospace, the complexity of designing 3 dimensional deployable forms is canonized by the challenge of building *parabolic reflectors*. Satellite dishes must *minimize* their volumes in the folded state and *maximize* their precision in the unfolded state. Guest credits Huso, Lanford, and Scheel with developing some early contributions to the notion of wrapping a flat membrane around a central hub. In the Space Structures Laboratory at the California Institute of Technology, Sergo Pellegrino has led research investigating various deployment mechanisms for microgravity conditions.

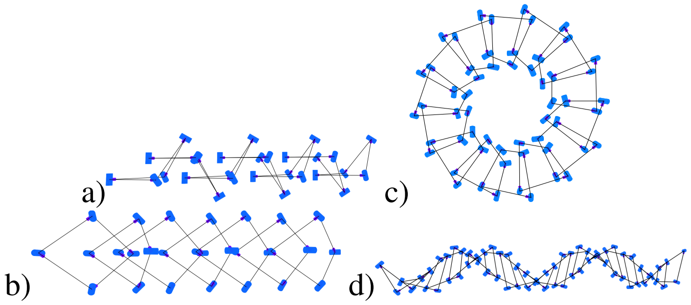


Figure 4.13: Linked Bennett mechanisms after You and Chen [138], which move with one degree-of-freedom. a) Profile view. b) Top View. c) By shifting the connection point between consecutive linkages, the configuration can be used to generate a circular truss. d) A twisted truss.

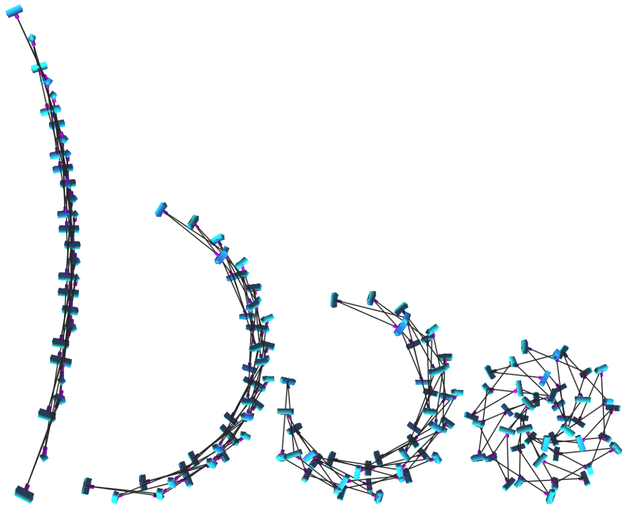


Figure 4.14: Deployment sequence of a circular truss composed of linked Bennett mechanisms.

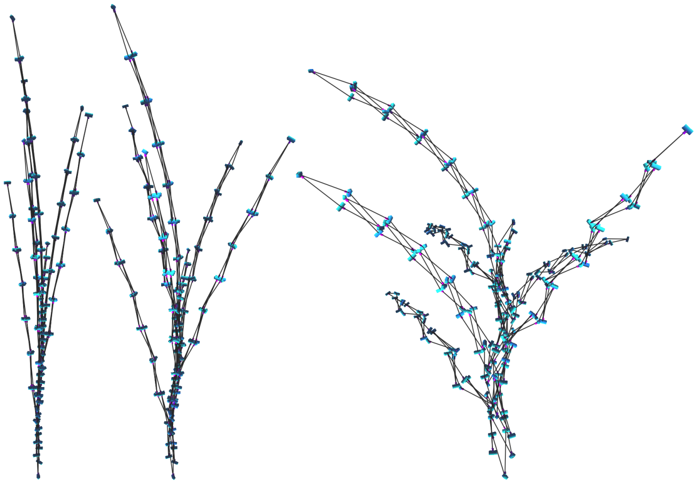


Figure 4.15: Transformation sequence of a linked Bennett mechanism, twisted in one direction, and circular in the other.

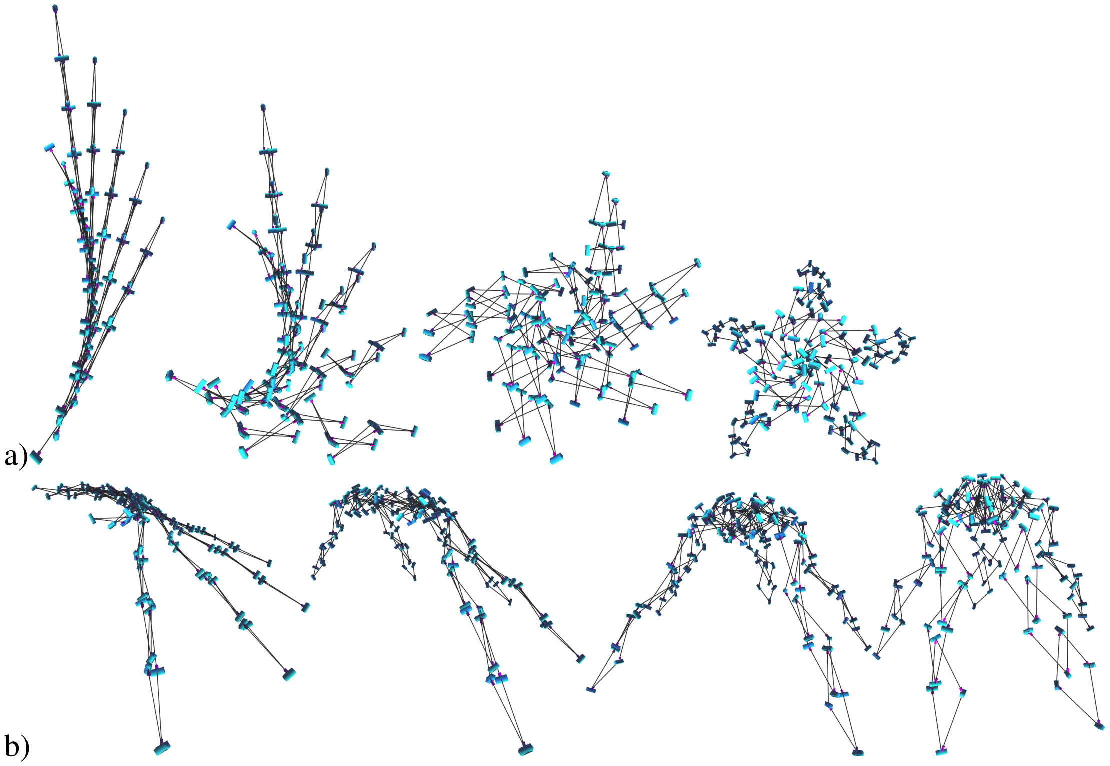


Figure 4.16: Deployment sequence of a proposed structure constructed with linked Bennett mechanisms: circular in one direction and twisted in the other. a) Top and b) side view.

4.6 Discussion and Future Work

We have demonstrated how the motor algebra of the conformal model can be used in conjunction with the coincidence of round elements to construct complex articulations that typically do not receive such unified treatment. By working within a homogeneous representation of 3D Euclidean space that admits and organizes all its isometric transformations, we were able to avoid notational complications that emerge from compounding trigonometric functions. It is apparent from these initial studies that the use of spheres to represent isometric constraints can be employed to find legitimate configuration spaces of overconstrained systems. It remains to be shown whether these construction methods are more precise.

We have been able to apply the Denavit-Hartenberg parameters for linkage mechanisms [36], while avoiding their matrix formulations, and detailed a strategy for encoding these parameters in terms of motors based on the work of Bayro-Corrochano. Illustrating the geometric representation of constraints they elicit, we applied this representation to an implementation of the approach suggested by the FABRIK method of Aristidou and Lasenby, and constructed a detailed algorithm for the inverse kinematics of a chain consisting of an arbitrary number of revolute joints (collision detection between joints was not addressed). This also required specifying expressions for extracting the circle of revolution given a set of link parameters. In contrast to this iterative approach, we then constructed a closed-form algorithm for the modelling of Bennett linkages, and then, inspired by You and Chen [138], we presented some forms that can be made by linking them together.

The potential applications of this work combine the motor algebra with the geometric construction capabilities in order to fabricate objects. It would be interesting to see gear systems and machinery in general designed with the constructivist approach applied here. A good resource for finding more linkage problems to cast into GA terms is Phillips [114]. To accomplish this, collision testing algorithms need to be developed to prevent self-crossing during both forward and inverse kinematic solutions. To further the treatment of kinematics, investigations into the modelling of *higher pair* joints are necessary, where the motion screws are not invariant with respect to the points of contact between links. For these contact-based models, more details are required in order to fully express the geometric constraints at play in terms of geometric algebra.

Chapter 5

Transformations: Constructions through Interpolation of the Bivector Exponential

> Jo sóc geòmetra, que vol dir sintètic.[1]
>
> -Antonio Gaudí

5.1 Summary

Here we more closely consider the synthesis techniques allowed by the representation of transformations as the exponentiation of bivectors. Continuing from the previous chapter, we begin with motors as exponentials of dual lines, and the interpolations that this representation allows. We will demonstrate the use of dual lines to build trilinearly interpolated twist **deformation** fields and then extend these methods into exponentiation of point pairs in preparation for the final Chapter. Using point pairs as exponentials, we introduce the action of rotation around a circle, for which the screws and twists serve as a special case. We show how this can be formulated into toroidal knots following the work of Dorst and Valkenburg in [46], and find a connection between that formulation and the Hopf fibration.

---

[1] I am a geometer, that is to say synthetic.

In [135], Wareham and Lasenby propose the use of linear and quadratic interpolations of bivector-valued rotor logarithms to blend frame positions, a subject also explored in [134], where the method is applied to deformation of a mesh. Those papers also contain algorithms to exponentialize dual line bivector "twists" into motors and conversely to find the logarithm. Belón extends those results to create more versatile mesh deformations in [13]. Sommer, Rosenhahn, and Perwass formulate "coupled twists" in [127] to synthesize forms. This is taken up by Perwass more completely in [111]. In [46], Dorst and Valkenburg give a rigorous treatment of the general bivector logarithm and reveal how it can be used to create **conformal orbits**, including knots. In [29], we take up the task of using these rotors to generate surfaces, a development we explore more fully in the next Chapter. Additional explorations into circle blending can be found on Ian Bell's website [12].

5.2 The Motor Algebra

Multiplying a Euclidean rotor by a translator returns an even-graded element with grades 0, 2 and 4. As we have discussed, even rotors in the conformal model can be represented as the exponential of some bivector $R = e^{-\frac{\theta}{2}\mathbf{B}}$. In the case of motors, $M = e^{-\frac{\lambda}{2}}$, λ is a **dual line.** In Plücker coordinate systems, the moment is defined as the cross product between two points on the line and is scaled by a factor proportional to the distance between them.

Figure 5.1 details the construction of a dual line twist generator, manipulating the pitch of the screw by adding a portion of the line's direction to its moment.

5.2.1 Exponentiating A Dual Line

Given a line λ weighted by a period θ, whose direction component is offset by a pitch ϕ, how do we formulate $e^{-\lambda}$? A general closed-form exponentiation method for *all* bivectors in the conformal model is provided by Dorst and Valkenburg in [46], and an alternative specifically for dual lines by in Wareham, Cameron and Lasenby [134, 135] (these texts also provide a

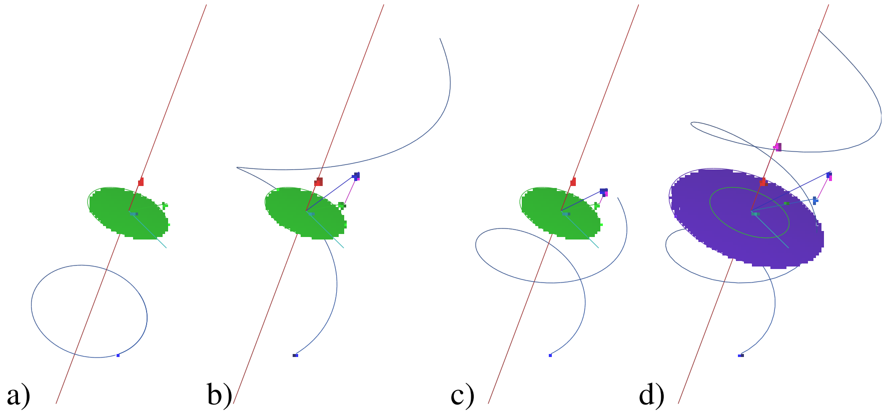


Figure 5.1: A dual line $\lambda = \mathbf{B} + \mathbf{d}n_\infty$ is encoded as a sum of a Euclidean bivector and a support direction or **moment**, its exponential $e^{-\lambda}$ is a **motor** which generates a **general rotation** in the direction of the orientation of **B**, depicted as the blue circle orbit. The bivector **B** encodes the direction of the line as an oriented area, here rendered as a green disk. The dual of that bivector, $\mathbf{B}^\star$, is the red vector which points in the direction of the line. The direction $\mathbf{d}n_\infty$, rendered as a green vector, encodes the line's location relative to from the origin in the following way: the inner product $\mathbf{d} \cdot \mathbf{B}$ is the vector distance from the origin, rendered here as a blue vector. b) By adding some proportion of $\mathbf{B}^\star n_\infty$ to $\mathbf{d}n_\infty$, we can add a **pitch** to the transformation generated by e^λ, turning a general rotation into a **screw**. c) Reducing the pitch proportion creates a tighter screw motion. d) Increasing the weight of **B** changes the **period** of the transformation, that is the rate at which it rotates around the line.

logarithm – a method for extracting the dual line λ from the motor $e^{-\lambda}$). In both texts, the exponentiation process requires treating these two motions separately, essentially **decomposing** them.

The general solution to bivector exponentials provided by Dorst and Valkenburg is discussed in section 5.3.6. Here we note the results of Wareham, Cameron, and Lasenby: given a dual line of the form $\lambda = \theta\mathbf{B} + \mathbf{d}n_\infty$(see Figure 5.1) where θ is a scalar and $\mathbf{B}$ a unit Euclidean bivector, an exponentiation decomposes $\mathbf{d}$ into parts parallel and perpendicular to $\mathbf{B}$, specifically:

$$e^{\lambda} = (\cos\theta + \sin\theta\mathbf{B})(1 + \mathbf{d}_{\perp}n_\infty) + \mathrm{sinc}(\theta)\mathbf{d}_{\|}n_\infty \tag{5.1}$$

with $\mathrm{sinc}(x) = \sin(x)/x$ and $\mathrm{sinc}(0) = 1$. Please see their text ([134]) for details.

5.2.2 Logarithm of a motor

Theorem 1 of [134] offers proof of a mechanism for extracting the dual line generator λ from a motor e^{λ}. This generator can then be linearly weighted and re-exponentiated (with Equation 5.1) to create an interpolated twisting motion. In practice it is helpful to remember to normalize the motor before finding its logarithm – given a **motor** $M = TR$ where R is the quaternionic rotor orientation and $T = 1 + \frac{\mathbf{v}n_\infty}{2}$ is the translator relative to the origin, normalize the result $M \mapsto M_{\text{normalized}}$ using the reverse norm: $M = \frac{M}{\|M\|}$ where $\|M\| = \pm\sqrt{M\tilde{M}}$ as defined in Equation 2.21. With such a normalized motor we can apply the logarithm from [134], replicated here for convenience:

$$\log(M) = ab + c_{\perp}n_\infty + c_{\|}n_\infty \tag{5.2}$$

where

$$
\begin{aligned}
\|ab\| &= \cos^{-1}(\langle M \rangle) \\
s &= \mathrm{sinc}(\|ab\|) \\
sq &= \|ab\|^2 * s \\
ab &= (\langle M \rangle_2 n_\infty) \cdot n_o)/s \\
c_\perp n_\infty &= -ab\langle R \rangle_4/sq \\
c_\| n_\infty &= -ab\langle ab\langle R \rangle_2 \rangle_2/sq.
\end{aligned}
$$

5.2.3 Interpolation of motors

The dual line logarithm of a motor $M_1 = e^{\lambda_1}$ is a linearly interpolatable 2-blade λ_1, and accordingly a common use of the logarithm of motors is to interpolate between two frame positions M_1 and M_2 (see Figure 5.2). This provides a way to continuously transform one frame into another through a combination of rotational and translational interpolation. There are two methods to do this. We can find the transformation that takes M_1 to M_2 as $M = M_2 M_1^{-1}$ and then formulate an interpolated value for scalar values $t \in [0,1]$ as $M_t = e^{t\,\log(M)} M_1$. Alternatively, we can formulate M_t through linear interpolation of the individual logarithms as

$$M_t = e^{((1-t)\lambda_1 + t\lambda_2)}, t \in [0,1] \tag{5.3}$$

where $\lambda_1 = \log(M_1)$ and $\lambda_2 = \log(M_2)$. These methods give different results. The latter method is useful when incorporating more than two motors into the results. For instance, as illustrated in [134] and here in Figure 5.4, we can create a mesh deformation field through **trilinear** interpolation of a volume. Given control frames $M_{i,j,k\in\{0,1\}}$ at the eight corners of a rectangular box, we can define a frame $M_{t,u,v\in[0,1]}$ in between as shown in Figure 5.3

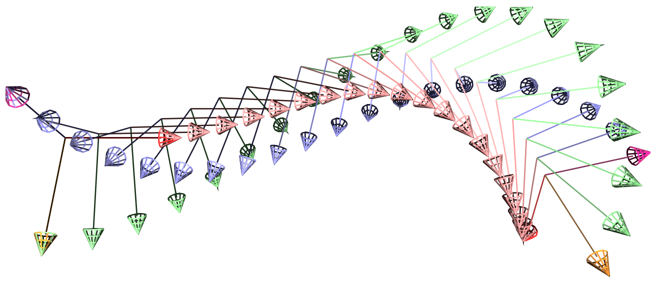


Figure 5.2: Interpolation of motors using equation 5.3.

A new extension of this concept is given form in Chapter 6, when we interpolate coordinate surface spheres.

5.2.4 Motor Compositions

In [127], Sommer, Rosenhahn and Perwass explore how two motor generating twists λ_1 and λ_2 can be exponentiated at different constant rates and applied to a point to create various shapes. We briefly experiment with this method (which they call "coupled motors") in Figure 5.6, in order to introduce more general results in the next section.

5.3 Point Pairs as Generators

In Chapter 6 we will explore some of the implications of a powerful formalism unique to the conformal model of geometric algebra: conformal rotors, which requires a deeper understanding of the general bivector exponential of a point pair. Point pairs contain all the 2-dimensional elements of our 5-dimensional algebra and so can be thought of as the most general exponent. Let us take a closer look at the special conformal transformations produced by exponentiation of this bivector element κ. Further details on the section that follows can be garnered from a close reading of the excellent text by Dorst and Valkenburg on this subject [46], as well as Dorst's follow up text [43].

The simple conformal rotors $C = e^{\kappa}$ explored in this section are sometimes called **transversions** or **boosts** or **Möbius transformations**. Unlike Euclidean isometries, these are **special**

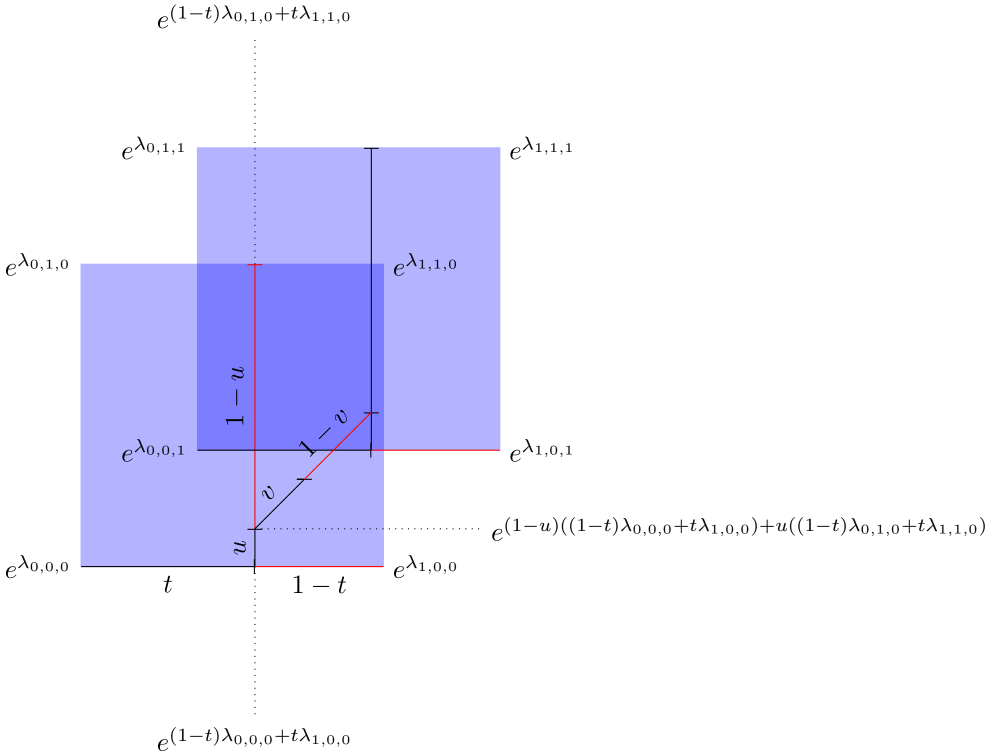


Figure 5.3: Trilinear interpolation of motor-valued frames at the eight corners of a rectangular box through linear sums of their logarithms, as proposed in [134].

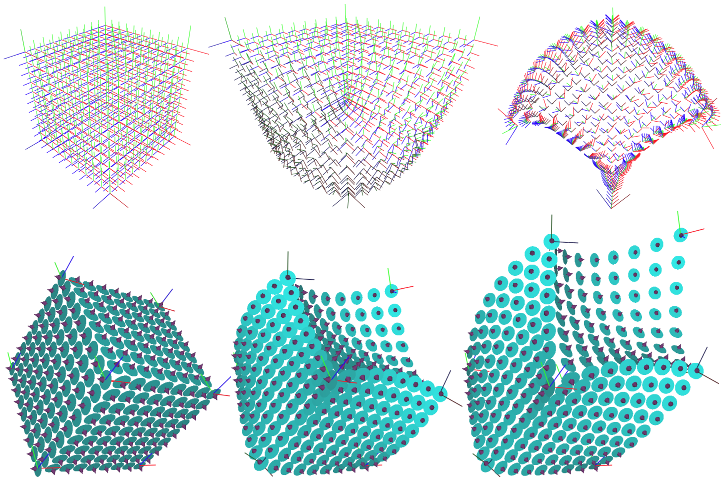


Figure 5.4: Mesh deformations created through trilinear interpolation of motors. 8 control frames are positioned on the corner of a cube and their logarithms affinely combined through trilinear interpolation.

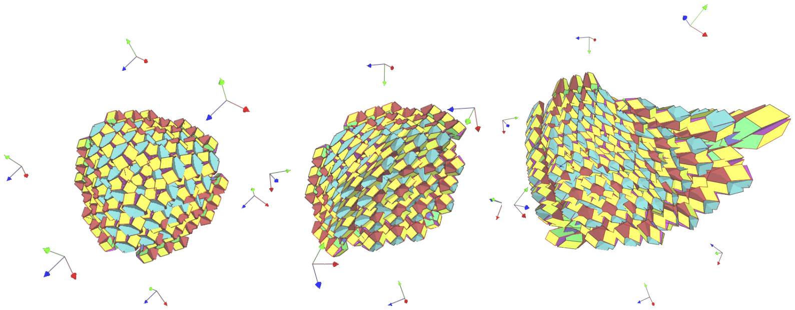


Figure 5.5: An example using motors as mesh deformations applied to one of the crystal systems from Chapter 3. Each point p in the crystal structure is given a coordinate $t,u,v \in [0,1]$ within the rectangular box formed by eight control frames. A deformed position $p^{'}$ is then calculated as $M_{t,u,v}[n_o]$, that is, the interpolated motor at t,u,v applied to the point at the origin.

conformal spinor operators which can straighten round geometric elements and bend straight ones, while maintaining invariance of local angles. A component of **inversive geometry**, conformal rotors can be considered a double reflection in two spheres with a common point, much the same way a rotation can be considered a double reflection in two planes with a common line. Operating at the origin, these rotors can also be constructed through concatenation of a sequence of operations: inversion in the unit sphere, followed by a translation, followed by another inversion in the unit sphere. They can also be used to realize the **Lorentz group** of transformations and thus to model the symmetries of relativistic physics. In this Section and the next Chapter, we use them as a bending curvature operators that can generate orbits and surfaces.

5.3.1 Bending the Line

Given a tangent vector $\mathbf{v}n_o$ at the origin with basis $\{e_1n_o, e_2n_o, e_3n_o\}$, it can be translated by $T = e^{-\frac{1}{2}\mathbf{t}n_\infty} = 1 - \frac{1}{2}\mathbf{t}n_\infty$ as $\kappa \mapsto T[\mathbf{v}n_o]$ which can be interpreted as a point pair with zero radius, encoding only a position, orientation and a weight. The action of such a null 2-blade, when exponentiated as $C = e^{-\frac{1}{2}\kappa}$ is what we now investigate.

As a conformal deformation which preserves angles but not distances, the point pair exponential C can be used as a curvature operator. Given a normalized line Λ, we can bend it

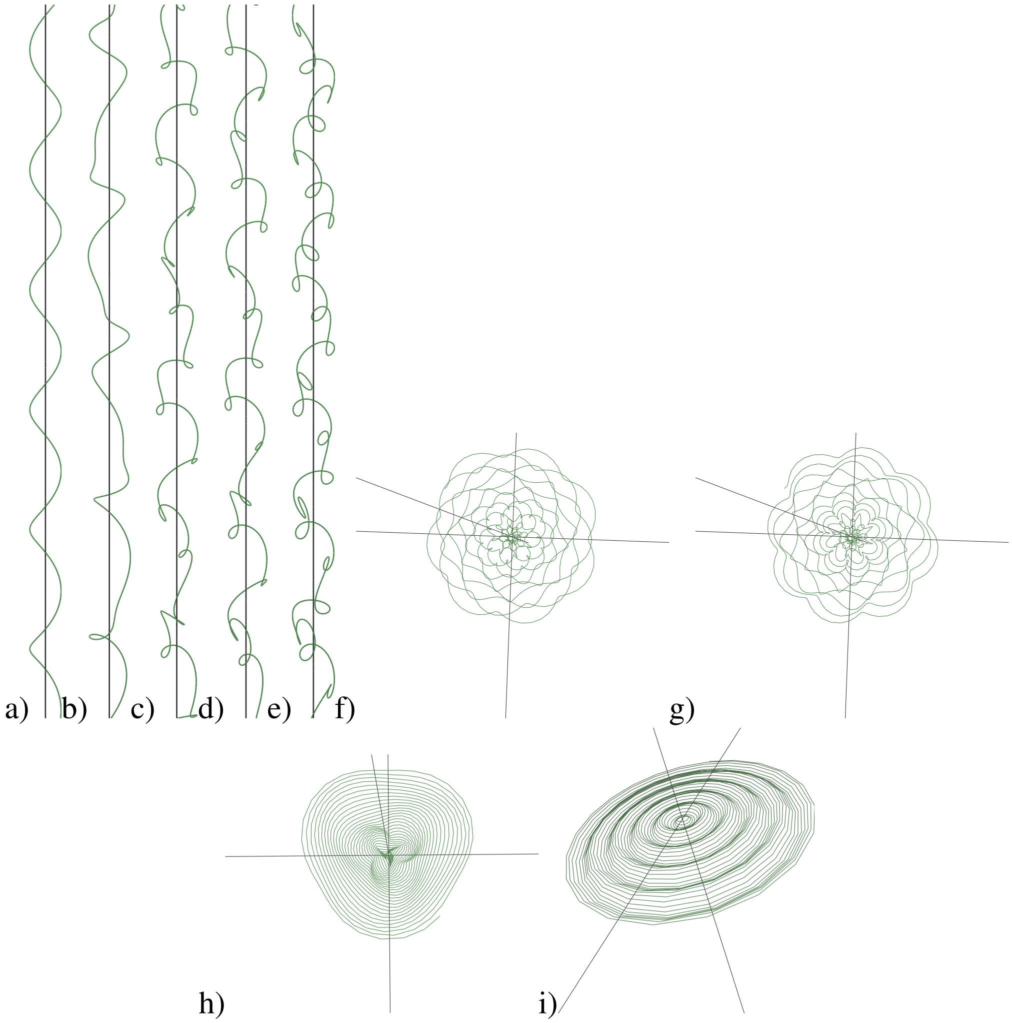


Figure 5.6: Trajectories of a initial point $p^{'} \mapsto M_t[p]$ created by applying two concatenated motors of the form $M_t = e^{-\frac{1}{2}t(c_2\lambda_2)}e^{-\frac{1}{2}t(c_1\lambda_1)}$, where t is in the range $[0,1]$, c_1 and c_2 are constants, and λ_1 and λ_2 are orthogonal dual lines. In figures a) - e) c_2 is held constant while values of c_1 increase from left to right. λ_1 is a pure rotation (no pitch along the screw axis). In figures f) - i) a third motor is concatenated, representing a screw motion in a third orthogonal direction.

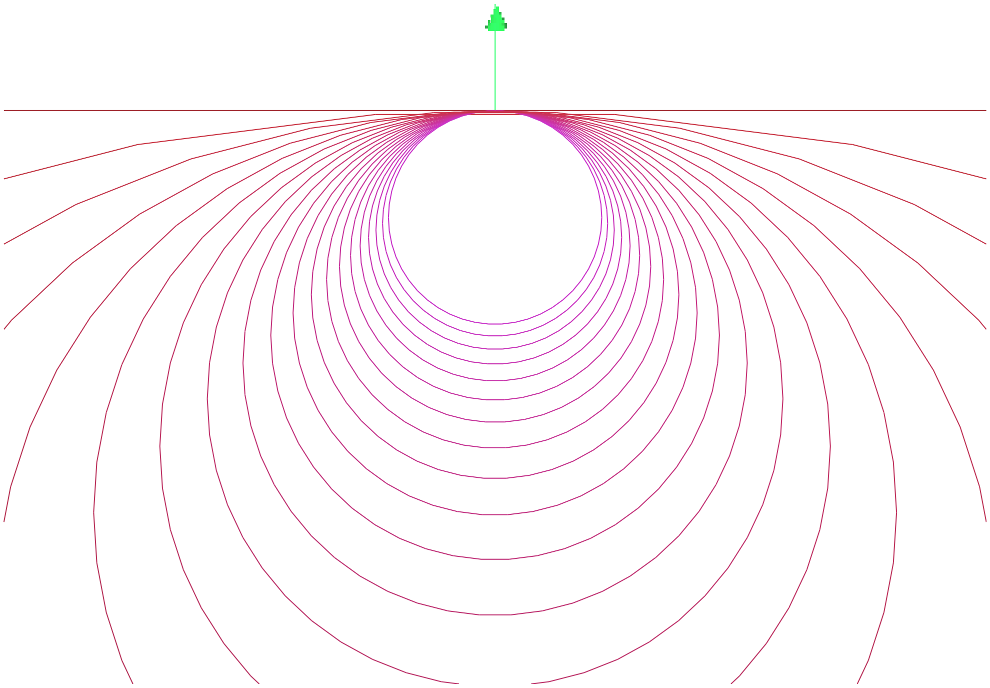


Figure 5.7: Continuous bending of a line into a circle of radius r using a rotor of the form $1 - \frac{1}{2r}\kappa$ (Equation 5.4), where κ is a null point pair (in green) generated by translating a tangent vector orthogonal to the line to a point on the line. The counter-rotor, $1 + \frac{1}{2r}\kappa$, would turn a circle of radius r into a line.

into a circle of radius r and curvature $c = \frac{1}{r}$ at point p by applying a boosting rotor of the form:

$$C = 1 - \frac{c}{2}\kappa \tag{5.4}$$

where κ is a zero-sized (null) point pair created by first translating a unit tangent vector orthogonal to the line to the point p on the line, and then weighting it by the target curvature. By adding rather than subtracting the bivector part of equation 5.4, we can create a transformation that takes a circle with curvature c and straightens it into a line. The $p-$translated and $\frac{c}{2}$-weighted tangent κ generates a c curvature operator at p. The effects of interpolating this operator or applying it to a mesh can be seen in the forms of Figure 3.7 and in our previous work in [28].

In section 5.3.6 we detail a formulation for taking any circle to any line.

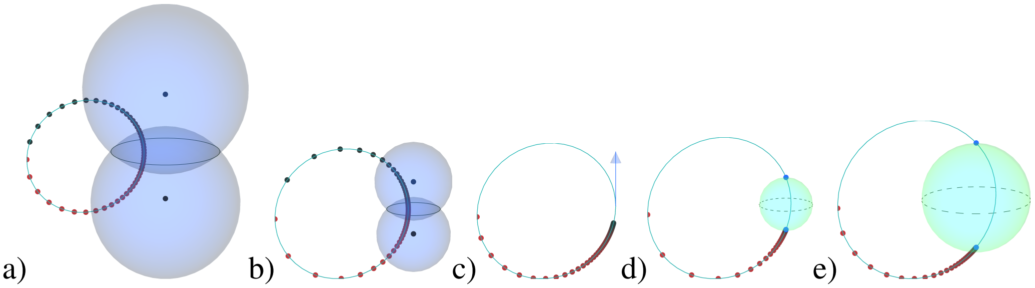


Figure 5.8: Conformal transformations of a point through transversion, sometimes called a boost. a) Toroidal rotation of a point p around a real circle K is achieved as a mapping $p^{'} \mapsto f_{\text{center}}[C_t[p]]$ where $C_t = e^{-\frac{t}{2}\kappa}$ with $\kappa = K^*$ and $t \in [0, 2\pi]$ and f_{center} signifying a normalization by taking the center of the result using Equation 2.60. Here we see the dual of the real circle K is the pair of spheres κ which meet at K. In b) and c), as the radius of K shrinks to zero, so does the radius of its dual κ. At radius 0, κ is a tangent vector and the action of $C_t[p]$ is no longer modulo 2π, but rather brings p to the limit as $t \rightarrow \infty$. d) and e) as the radius of K becomes negative, κ represents the poles of the sphere which surrounds the imaginary circle K. In this case, the action of $C[p]$ acts as a sink which brings p to the south pole of κ. In all cases the transformation is along a circle $p \wedge \kappa$ called the **orbit** of p, as discussed in [46].

5.3.2 Orbits

We have seen that a line Λ, or rather its dual representation λ, can be used to generate a rotation around it. Since lines are specialized versions of circles, we might imagine that a circle K, or rather its dual representation as point pair κ (see Figure 2.12 in Section 2.4.3), can be used to generate a toroidal rotation around it. Indeed this is case. In Figure 5.8 we identify a circle K, its dual $\kappa = K^*$, and the action of the exponential $C = e^{-\frac{c}{2}\kappa}$ on a point p along the circular orbit $p \wedge \kappa$. Note the need for a normalization of the mapping $p^{'}$, which can be easily done in an implementation by extracting the center point of the result using Equation 2.60.

5.3.3 Orthogonal Orbits

The previous orbits are **simple** in that the exponent is a 2-blade (remember, a blade is an element that can be expressed as the outer product of vectors). As with dual lines, whose directions can be summed to create a pitch along the axis of rotation (as illustrated in Figure

5.1), so can point pairs be summed with other point pairs to create a secondary rotation. The format of these orthogonal components is not as easy to dissect as it is in the case of dual lines, and more care needs to be taken in choosing them (see [46]). The sum B of any two point pairs κ_a and κ_b can be refactorized into a sum of two **commuting** point pairs κ_1 and κ_2 such that $\kappa_a + \kappa_b = B = \kappa_1 + \kappa_2$ and

$$\kappa_1 \kappa_2 = \kappa_2 \kappa_1. \tag{5.5}$$

The commutativity allows one to calculate e^B for a general bivector B as the composition of two simple transformations $e^B = e^{\kappa_1 + \kappa_2} = e^{\kappa_1} e^{\kappa_2}$. In Appendix C of [43] Dorst proves that such commuting 2-blade generators κ_1 and κ_2 are necessarily orthogonal to each other (provided $\kappa_1 \wedge \kappa_2 \neq 0$):

$$\kappa_1 \cdot \kappa_2 = 0 \tag{5.6}$$

and, critically, that so are therefore the orbits around both for any given point x. We leverage this orthogonality of orbits in the next Chapter to navigate curved coordinate systems. Specific details for splitting any general bivector into two commuting point pairs can be found in [46], and we will look at what transformations this enables in a moment.

Before detailing the split of a bivector, let us first consider some geometric configurations of well-chosen point pairs which already naturally satisfy the commutation criterion. Given a circle K and its point pair dual κ, its axis can be represented as a dual line $\lambda = n_\infty \cdot K$ which commutes with κ (dual lines represent a subset of point pairs, see Table 2.5). Thus weighted combinations of κ and λ can create orbits that combine a motion around the circle as in Figure 5.8a and around the dual line axis of the circle as in Figure 5.1a. If these weights are harmonically related, then the orbit will close into a knotted formation, as is illustrated in Figure 5.9.

For reasons that will become apparent soon, let us call the circle and its axis, whose dual

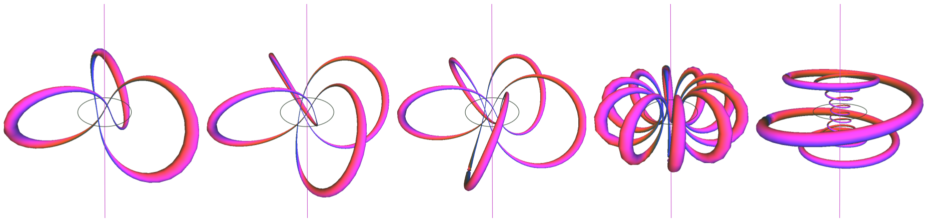


Figure 5.9: Knotted orbits composed of a simultaneous rotation around a circle and its axis. The generators, $\kappa_1 = K^*$ and $\kappa_2 = n_\infty \cdot K$, are orthogonal and commute, and the compound conformal rotor $C_\theta = e^{-\frac{\theta}{2}c_2\kappa_2}e^{-\frac{\theta}{2}c_1\kappa_1}$ with $\theta \in [0, 2\pi]$ and scalar constants c_1, c_2 generates the full orbit when applied to some input p. If the range $[0, 2\pi]$ is divided into n steps then a single small simple rotor $C = e^{-\frac{\pi}{n}(c_1\kappa_1 + c_2\kappa_2)}$ can be applied iteratively n times to some input p. Again we normalize the result by finding its center using equation 2.60. In the figures above, the coprime harmonic constants$\{c_1, c_2\}$ represent the ratio of windings arounding each $\{$circle, axis$\}$ fiber. These are, from left to right, $\{3,2\}$, $\{4,3\}$, $\{5,3\}$, $\{10,1\}$, and $\{1,10\}$.

representations generate the rotation, **fibers**. What other pairs of fibers provide this naturally commuting split? Based on our own visual experiments with a computer, it appears that we can use a mapping of the 2-sphere (a "regular" sphere in 3D space) to the 3-sphere (a sphere in 4D space) called the **Hopf fibration** to create antipodal linked fiber pairs suitable for generating knotted orbits (i.e. the fibers commute).[2]

5.3.4 The Hopf Fibration

The Hopf fibration is a mechanism for representing a **3-sphere**, which is a sphere in 4 dimensions, as a bundle of circle fibers in 3 dimensions. The topology crops up in various branches of mathematics and physics, including quantum mechanics. The mapping takes each point on the surface of a 2-sphere – which is our typical sphere in 3 dimensions and which we will call the **preimage** – to a circle, thereby extending each point by one dimension. The circle and axis from Figure 5.9 can be thought as fiber mappings of the points at the south pole and north pole of our preimage, respectively. Points along the longitudinal lines connecting the poles represent a continuous transformation of the circle to its axis. By this mapping, the fibers of any two **antipodal** points on the 2-sphere preimage are orthogonal commuting blades.

---

[2] Etienne Ghys and Jos Leys provide an interesting visualization of modular flows in higher dimensions online at [58], which investigates the connection between the Hopf fibration and knots.

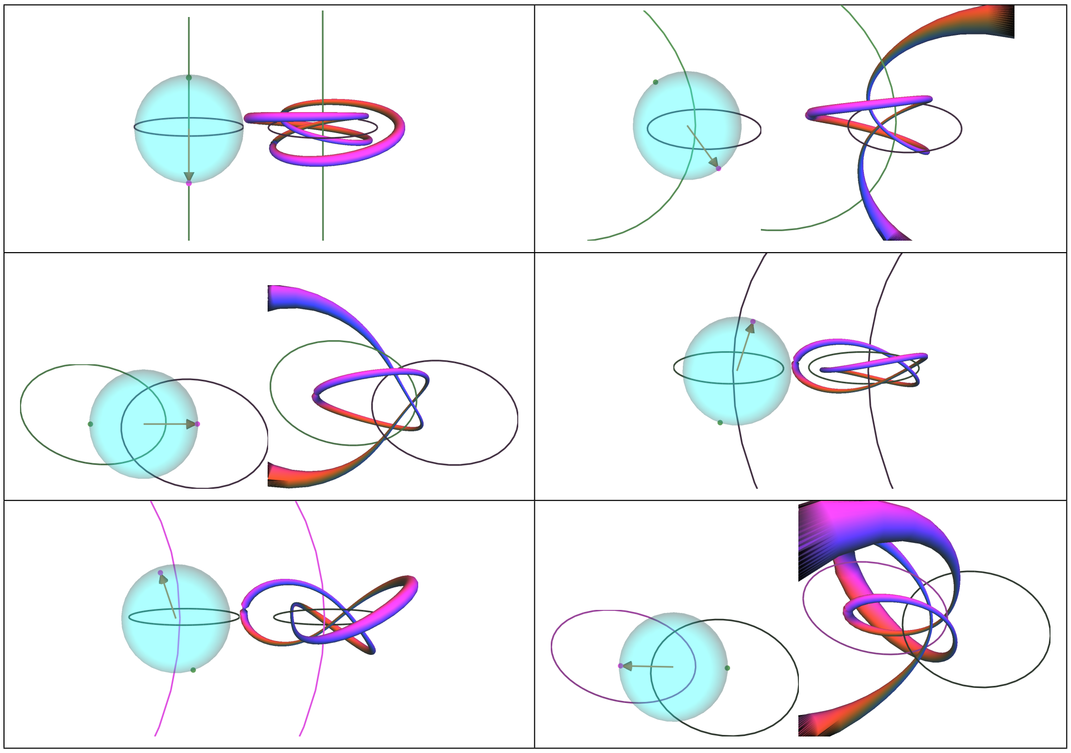


Figure 5.10: Antipodal points on a sphere are mapped to two Hopf fibers which are then used to generate a compound conformal orbit of some input point. As the vector specifying the antipodal points rotates, the resulting fibers create **homotopic** variations of the knotted trajectory of the point: the knot remains a $\{3,2\}$ torus knot and never crosses itself. Videos demonstrating this commutation property, which allows us to keep a knot invariant as we smoothly transform the fibers around which it winds, can be viewed online at https://vimeo.com/wolftype/videos.

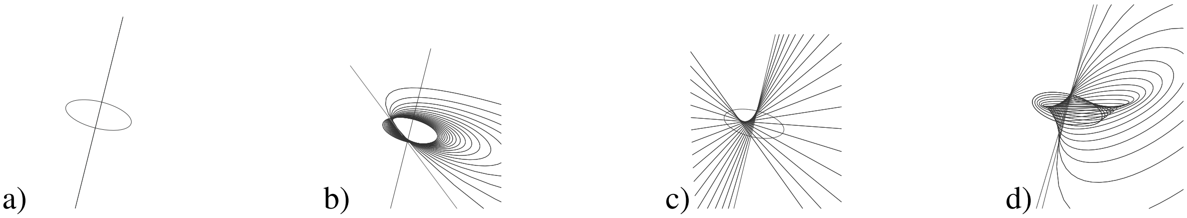


Figure 5.11: The initial components of the Hopf fibration are a circle and its axis representing fibers over the south and north poles of the 2-sphere. b) The circle is boosted into a line. c) The line is twisted into the axis. d) Composition of the boost and twist make a fiber bundle over a meridian of the 2-sphere.

A Boost with a Twist

Generating the transformations of the circle fibers in Figure 5.10 requires a composition of a bending operation, which straightens the circle, and a twisting, which rotates it by 90 degrees. Here we show how the canonical **Hopf fibration** map can be constructed by a boost followed by a twist, each parameterized on spherical coordinates θ, ϕ. The fibration maps the 2-sphere to the 3-sphere by taking each point on the 2-sphere to a circle fiber. If we map the point at the south pole to a circle, then the antipodal point at the north pole is mapped to the axis of that circle. Thus we start by noting that the fibration of a longitudinal line from pole to pole can be generated by a rotor which takes a circle to its own axis. To construct this transformation, we compose a boost which straightens the circle into a line, and a twist which screws that line into the axis. The combined process of boosting and twisting is seen in Figure 5.11.

Explicitly, to map the point $p_{\theta\phi}$ with spherical coordinates θ, ϕ (in radians) on a 2-sphere to a circle fiber of a 3-sphere, we start with a unit circle K at the south pole (where $\phi = -\frac{\pi}{2}$) and a base axis $\lambda = -n_\infty \cdot K$ at the north pole (where $\phi = \frac{\pi}{2}$). We then define the transversion $C_{\theta,\phi}$ which takes K to a line using Equation 5.4:

$$C_{\theta,\phi} = 1 + k_\phi n_o v_\theta \tag{5.7}$$

where $k_\phi = \frac{1}{2} + \frac{\phi}{\pi}$ and scales in the range $[0, 1]$ and where $\mathbf{v}_\theta$ is a unit vector in the plane of the circle. At $\phi = \frac{\pi}{2}$, C takes the circle to a line:

$$\Lambda_\theta = C_{\theta,\frac{\pi}{2}}[K] \tag{5.8}$$

The corresponding twist motor $M_{\theta,\phi}$ can then be generated by finding the ratio of the initial axis Λ with the line Λ_θ.

$$M_{\theta,\phi} = e^{k\phi log(\frac{\Lambda}{\Lambda_\theta})} \tag{5.9}$$

The full transformation rotor K^H can then be defined:

$$C^H_{\theta,\phi} = M_{\theta,\phi} C_{\theta,\phi},\ \theta \in [0, 2\pi],\ \phi \in [-\frac{\pi}{2}, \frac{\pi}{2}] \tag{5.10}$$

Note that as a product of a twist and a boost, C^H is not a simple boost composed of just a scalar and bivector, but now also contains the quadvector $e_{12}E$, $e_{13}E$, $e_{23}E$, $e_{123}n_o$, and $e_{123}n_\infty$.

The explicit formulation in Equation 5.10 of a conformal rotor C which takes a circle to its own axis can also be calculated by taking the square root of their ratio and then using the bivector split approach developed by Dorst and Valkenburg in [46]. The construction is more involved than typical rotor ratios, however, since the ratio of circles is not a simple rotor but rather contains a quadvector component. In Section 5.3.6 we take a brief look at this formulation.

5.3.5 Taking a Sphere to a Sphere

In the next chapter, given two spheres, σ_1 and σ_2, we will find the conformal rotor C that takes one to the other. The absolute transformation can be calculated as $\sqrt{[\frac{\sigma_2}{\sigma_1}]_{\text{normalized}}}$ using equation 2.38. In practice, however, we are usually more interested in constructing a continuous transformation, which requires finding the point pair κ which we can weigh in the range of $[0,1]$ and then re-exponentiate. We can calculate κ as *half* the log of normalized

Figure 5.12: Twistor meshes representing the 3-sphere Hopf fibration generated by application of the transformation rotor $C^H_{\theta,\phi}$ onto a base circle.

$\frac{\sigma_2}{\sigma_1}$ using equation 2.68: $\kappa = \frac{1}{2}\log([\frac{\sigma_2}{\sigma_1}]_{\text{normalized}})$. Thereafter, the rotor $C_t = e^{-\frac{t}{2}\kappa}, t \in [0,1]$ can be used to take one sphere to another.

5.3.6 Taking a Circle to a Circle

In the very general case of circle-to-circle transformations (which, of course, includes circle-to-line transformations) we have found greater control is possible when calculating the logarithm of the square root (that is, as opposed to calculating half the logarithm). The main reason for this supposition is that deliberate steps can be taken when calculating the square root to account for the special case when the two circles are orthogonal to each other: specifically, we detail how to pick a particular direction of interpolation when more than one direction, or an infinite number, are possible.

Section 5.2.2 in Dorst and Valkenburg's text details how to take the square root of such a general ratio, and sections 5.3.3 and 5.3.4 of their text explain the use of the exterior derivative of its action to extract the logarithm and then split it into two commuting bivectors. Rather than duplicate those somewhat intricate equations here, the reader is encouraged to examine that reference. Here we note some subtleties when using those formulations, giving particular focus to the case of orthogonally linked circles.

We will note here some very useful results of their work in finding square roots and logarithms of general bivector exponentials in $\mathbb{R}^{4,1}$, which can be applied to the ratio of circles. First, Equation 5.4 of their text demonstrates that the square root of such a ratio can be expressed as

$$\sqrt{C} = (1+C)\frac{1+\langle C\rangle - \langle C\rangle_4}{2((1+\langle C\rangle)^2 - \langle C\rangle_4^2)}\frac{1+\langle C\rangle + \langle C\rangle_4 + \sqrt{(1+\langle C\rangle)^2 - \langle C\rangle_4^2}}{\sqrt{1+\langle C\rangle + \sqrt{(1+\langle C\rangle)^2 - \langle C\rangle_4^2}}} \tag{5.11}$$

with a special case (for the instance when C is the ratio of two orthogonal circles) that is expressed in Equation 5.16.

Let us now say we have a rotor C that is the square root of some normalized rotor ratio $C = \sqrt{[\frac{K_b}{K_a}]_{\text{normalized}}}$. Following a result from [75], Dorst and Valkenburg show we can find half the *curl* (exterior derivative) of this rotor C and then split it into commuting 2-blades.

$$F_{\pm} = \frac{1}{2}F(1 \pm \frac{\|F\|^2}{F^2}) \tag{5.12}$$

where

$$F = 2(\langle C\rangle_4 - \langle C\rangle)\langle C\rangle_2. \tag{5.13}$$

and

$$\|F\| = \sqrt[4]{(2F^2 - F^2)F^2} \tag{5.14}$$

Equation 5.12 splits the curl into two commuting 2-blades of the form $F = S_+ + S_- = \sinh(B_+) + \sinh(B_-)$ that can each be fed into the first argument of Equation 2.68 to find the logarithms. The second argument to Equation 2.68 can be extracted using equation 5.26 of [46]:

$$\cosh(B_{\pm}) = \begin{cases} -\langle\langle C^2\rangle_2 / S_{\mp} & \text{if } S_{\mp}^2 \neq 0 \\ \langle C^2\rangle & \text{if } S_{\mp}^2 = 0. \end{cases} \tag{5.15}$$

Finally, we note that Equation 5.12 requires defining the inverse of F^2, which is a scalar plus a quadvector, and the usual inversion of versors from Equation 2.18 does not apply here. Still, the inversion takes a simple form: Dorst has observed[3] that the inverse of such a multivector X is $X^{-1} = (\langle X\rangle - \langle X\rangle_4)/(\langle X\rangle^2 - \langle X\rangle_4^2)$.

Orthogonal Circles

Many interesting forms can be constructed from the interpolation of two orthogonal circles, which is a special case that requires care when implementing the algorithms of [46].

Given two orthogonal circles K_1 and K_2, their normalized ratio

$$C = (\frac{K_2}{K_1})/\|(\frac{K_2}{K_1})\|$$

represents twice the transformation that takes K_1 to K_2. Its square root can be found using equation 5.5 of [46]:

$$\sqrt{C} = (\frac{1+C}{2} + B\frac{1-C}{2}) \tag{5.16}$$

with the specific condition that B, a 2-blade, squares to -1 and commutes with $\langle C\rangle_4$ (the quadvector portion of C). In an implementation, we must create such a B and in Algorithm 5.1 we detail a method for construction which enables control over the direction of transformation through selection of a unit tangent vector. Figure 5.13 illustrates this method.

---

[3] Personal correspondence.

Algorithm 5.1 Technique for constructing a suitable Bivector B for Equation 5.16, to be applied when calculating the square root of a rotor ratio C if $(1+C_0)^2 = C_4^2$.

```
procedure CONSTRUCT B FOR ORTHOGONAL CIRCLES
    $C \leftarrow [K_2/K_1]_{\text{normalized}}$                ▷ The normalized rotor ratio of two circles.
    $\Sigma \leftarrow C_4$                                      ▷ The quadvector portion of $C$.
    if $(1+C_0)^2 = C_4^2$ then                                  ▷ Orthogonality Condition.
        $\sigma_1 \leftarrow K_1/(-n_\infty \cdot K_1)$          ▷ The imaginary dual sphere surrounding circle $K_1$.
        $\sigma_2 \leftarrow K_2/(-n_\infty \cdot K_2)$          ▷ The imaginary dual sphere surrounding circle $K_2$.
        $\mathbf{v} \leftarrow [\sigma_2 - \sigma_1]_{\text{normalized}}$   ▷ The unit vector between circles.
        $\kappa \leftarrow T_{\sigma_1}[\mathbf{v}n_o]$          ▷ Tangent vector translated to center of circle $K_1$.
        $B \leftarrow [(\kappa \wedge \Sigma^*)^*]_{\text{normalized}}$     ▷ $B$ is the normalized bivector meet of $\kappa$ and $\Sigma$.
end procedure
```

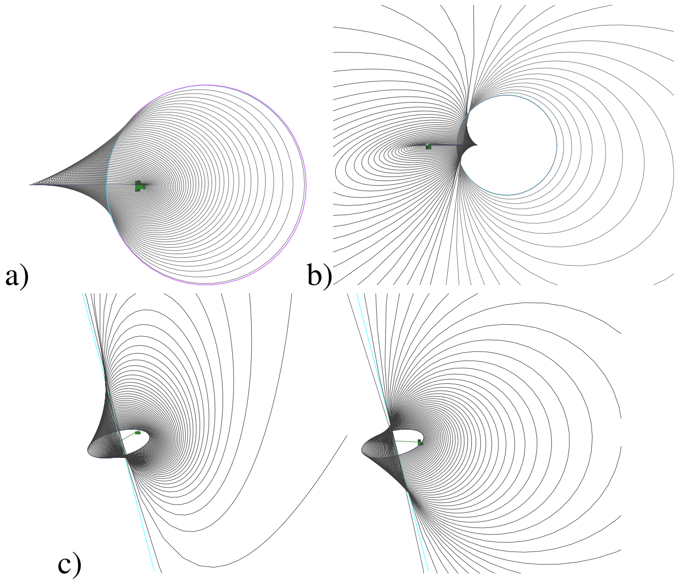


Figure 5.13: Isoclinic rotations generated by the ratio of two orthogonal circles. a) and b) illustrate two different directions of interpolation based on two different tangent vectors $\mathbf{v}n_o$ (represented as arrows) of Algorithm 5.1. c) In the case of a circle and its axis, the vector $\mathbf{v}$ must be chosen using some other method, such as selecting a particular angle θ.

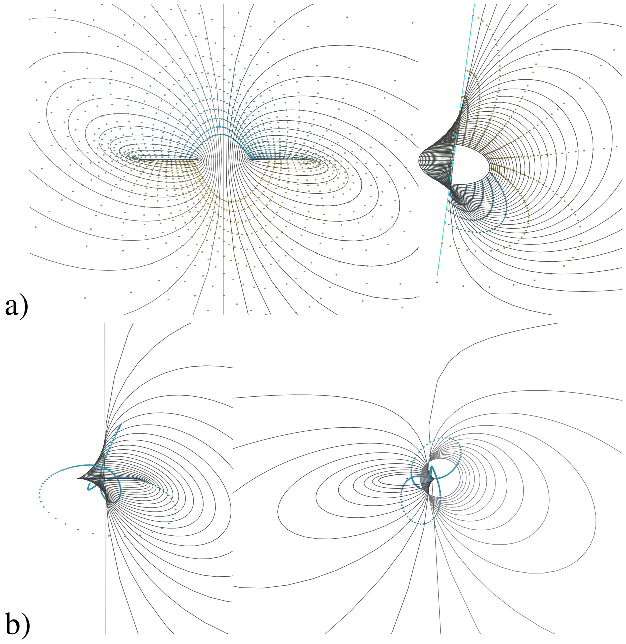


Figure 5.14: a)The action of conformal rotors C take points of a circle K_2 to corresponding points on circle K_2. b) If the circles are linked and orthogonal, the action is an isoclinic Clifford rotation which can induce knotted orbits through harmonic weighting of the 2-blades that emerge from the bivector decomposition.

Modular Characteristics

Given **orthogonally linked fibers** K_a and K_b and an **orbit** generated by the split of normalized $\sqrt{\frac{K_b}{K_a}}$, with each component of the split assigned a **P** and **Q** weighting:

- If K_a and K_b are **linked**, the induced orbital flow has no singularities
- if K_a and K_b are **linked** and **orthogonal**, the orbit is periodic (knotted)
- To discretize a PQ orbit into n steps, weigh by $4n/P$ and $4n/Q$
- At n/P and n/Q, a point on K_a is taken to a point on K_b
- At $2n/P$ and $2n/Q$, a point on K_a is taken through K_b and back to K_a
- At $3n/P$ and $3n/Q$, a point on K_a is taken through K_b, through K_a, and back to K_b

This happens for certain harmonic ratios of P and Q. We have found these characteristics hold through geometric experimentation on a computer, but do not provide proofs of these statements. They do, however, hint at the connection between spinors in conformal geometric algebra, knotted orbits, and the **modular group**.

5.4 Discussion and Future Work

The wealth of transformations available of the form e^B – with B some bivector in the conformal model – is a long cry from the bare bones simplicity of complex numbers and quaternions. These rotors can be compounded to generate not just proper Euclidean transformations – rotation, translation, and scaling – but also the conformal transformations in both simple (with B a 2-blade) and compound (with B a general bivector) form.

From the perspective of raw synthesis techniques, the linear transformations of dual lines offer a mechanism for mesh warping that avoids kinking behavior. While much work has been executed with respect to the motor algebra and representation of shapes and their deformations using twists amd motors, the use of conformal transformations for deformation and synthesis requires greater attention. It is evident that conformal transformations enable powerful articulations, producing continuous mappings of generalized circles and knotted orbits. Following [46], we have specified algorithms for controlling isoclinic rotations, and uncovered a relationship between antipodal Hopf fibers and commuting point pairs. It would serve us well to investigate this connection further in order to explore potential applications in modelling quantum behavior. The Hopf fibration is usually defined in terms of two complex numbers or quaternions, and understanding its structure as an isoclinic transformation which bends and twists a circle into its own axis may provide insight into how to visualize or navigate this manifold. Developing an algorithm which maps into our geometric representation to the standard methods will help make contact with the rich literature on its topology. For now, to build our intuition, in Chapter 6 we will introduce a novel use of these expressive conformal operators for the generation of curved surfaces and volumes.

Chapter 6

Curvature: Constructing Cyclidic Nets

> There are no cubical heads, eggs, nuts, or planets.
>
> – Buckminster Fuller

6.1 Summary

Conformal immersions are harmonic and numerically stable surfaces whose tangents scale isometrically, providing many elegant geometric properties of use in design. This chapter maps these forms within the context of discrete differential geometry in order to outline an approach to synthesizing curved surfaces with applications in architectural geometry, machining, and computer graphics. Inspired by Dorst and Valkenburg's "Square Root and Logarithm of Rotors" paper [46], we reformulate the rationalization of cyclidic nets, piecewise smooth surfaces characterized and controlled directly by their tangents.

In their remarkable paper, Dorst and Valkenburg rigorously *decompose* general rotations in the 3D conformal model of geometric algebra into commuting point pair generators, revealing much of the beauty in the algebraic machinery that generates any structure-preserving 3D motion. This is further explored in a more recent text by Dorst [43]. To explore some of the implications of these works, we outline a parametric approach to design which *composes* surfaces from out of that decomposition, suggesting the central role of conformal rotors in the establishment of a discrete calculus using the geometric algebra mechanism.

The benefits of conformal mappings in design are known in various fields. In architectural geometry, discretization of conformal mappings allows the design of doubly-curved[1] forms with torque-free nodes and consistent offsets [16]. In computer graphics, conformal mappings have been used to deform meshes intuitively and in a way which preserves details and textures [32, 31].

Leveraging Dorst and Valkenburg's analysis of general conformal rotors as commuting (and orthogonal) generators, we use Möbius geometry to reformulate the discrete cyclidic nets described by Bobenko and Huhnen-Venedey in [17], who use Lie Sphere Geometry (in the space $\mathbb{R}^{4,2}$) to build on the original rationalization of Martin in [99] based on Dupin's cyclides [48].

We find that the conformal rotor construction provides a straightforward mechanism for linearly rationalizing curved surfaces within the already rich context of conformal geometric algebra.

6.2 Cyclidic Nets

Cyclidic nets are a parameterization of continuously smooth surfaces introduced by R. Martin in his 1983 thesis which enable the digital designer to construct surfaces by specifying curvature directly at tangents. Motivated by a desire to develop a technique to form firmly grounded in geometry, Martin showed that these biquadratic meshes are piecewise smooth patches of Dupin cyclides, and thereby possess a simple algebraic representation. Built with families of generalized circles, discrete differential geometry over these surfaces exhibits excellent convergence to the continuous case. As a result, rationalization and efficient eval-

---

[1] Buckminster Fuller argues the central role of curvature in generating stable forms. In section 106.10 of Synergetics he writes “Compound curvature, or sphericity, gives the greatest strength with the least material. It is no aesthetic accident that nature encased our brains and regenerative organs in compoundly curvilinear structures. There are no cubical heads, eggs, nuts, or planets.”

uation of these surfaces – and their general form as Darboux cyclides – has been the subject of much research [52, 116]. Some modern techniques, such as developed by Bobenko and Huhnen in [17], rely on *Lie sphere geometry*, a projective model of contact-based geometric constructions that encapsulates the conformal model of geometric algebra. Others, such as Krasauskas and Zubé in [87], use geometric algebra to construct generic Darboux cyclidic patches using bilinear interpolation of quaternion-weighted Bézier arcs. In [86], Krasauskas uses Laguerre geometry, another subset of Lie sphere geometry, to create surface patches.

In other work specific to GA and CGA, Doran explores circle and sphere blending through direct linear interpolation of the primitive elements (that is, without rotors, conformal maps, or logarithms) [38], and Druoton et al explore constructions of Dupin cyclides by generating the full families of tangent spheres in [47].

Our current exploration differs from these others in its reliance on orthogonally-composed conformal rotors and their logarithms to rationalize blends between coordinate surface elements. We emphasize the use of sphere-to-sphere conformal transformations to discretize these coordinate surfaces, allowing ratios of basic geometric primitives to guide our rationalization. We provide a more controllable and general result than our experiments in [28], where we examined the use of interpolated null tangents to create “boosted surfaces”.

Our technique is grounded in a geometric (spherical) discretization of curvilinear coordinate systems, and our mappings apply as easily to tangents and normals as they do to points, representing a novel approach to constructing a general operator on a surface.

6.3 Principal Patches

Throughout the literature on parametric surface design with Dupin cyclides, a **principal patch** is created through bilinear rationalization of four points on a circle (Figures 6.1 and 6.2). Given these four concircular points and the orientation of a frame on one of them, the patch surface is determined. Surfaces constructed out of such patches are called **cyclidic nets**.

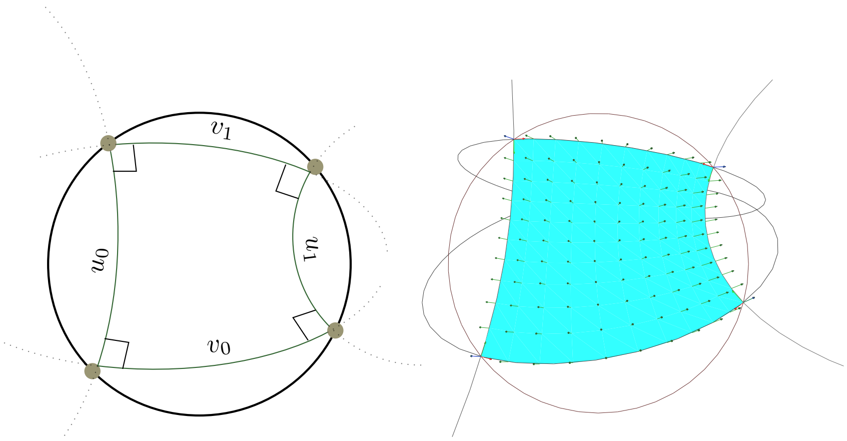


Figure 6.1: A four-sided principal patch is constructed from four concircular points with orthogonal corners. Principal contact edges lie along circles.

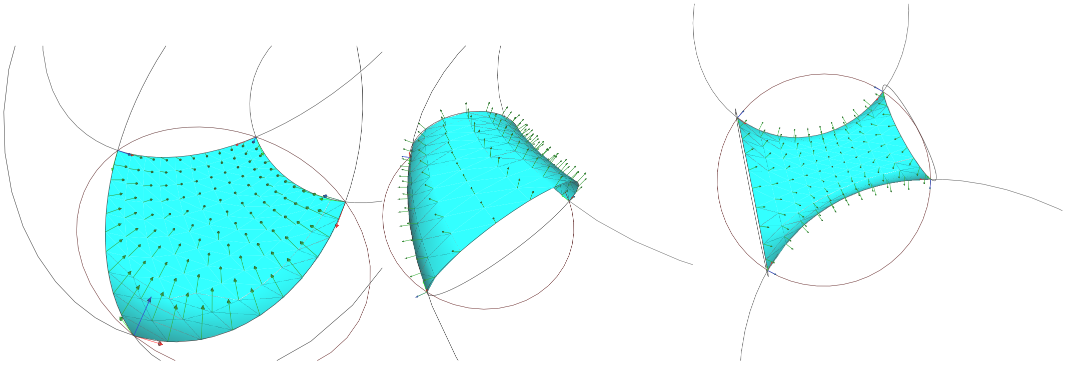


Figure 6.2: Given four points on a circle, one frame of orientation at one point (i.e. 3 degrees of freedom) is sufficient to fully characterize the patch.

We seek here to develop a system for constructing and discretizing these patches.

6.3.1 Outline of the Algorithm

To rationalize a principal patch – that is, to evaluate the position of a point p at coordinate (u, v) – we seek a particular conformal mapping of a 2-manifold into 3D $f : M \to \mathbb{R}^3$. In what follows, we will formulate this map in terms of two commuting and orthogonal transformation generators, κ_u and κ_v, each one a 2-blade point pair in the conformal geometric algebra of $\mathbb{R}^{4,1}$.

Labelling each side of the patch u_0, u_1, v_0, and v_1 (Figure 6.1) maps our edges to a 2-dimensional (u, v)-coordinate system in the range of $[0, 1]$ where the subscript denotes a constant coordinate value, e.g. $u_0 : u = 0$. One transformation generator, κ_u, will be responsible for interpolating u_0 to u_1 and the other, κ_v, will be responsible for interpolating v_0 to v_1.

Each edge is part of a **coordinate curve** circle K^{u_0},K^{u_1},K^{v_0}, and K^{v_1} which, as we will see, are themselves each part of larger **coordinate surface** spheres σ^{u_0},σ^{u_1},σ^{v_0}, and σ^{v_1} (see Figure 6.4). We define κ_u and κ_v as the principal logarithms of conformal rotors C_u and C_v, which we define as the square root of the normalized ratio of coordinate surface spheres. Thus, $C_u = \sqrt{\frac{\sigma^{u_1}/\sigma^{u_0}}{\|\sigma^{u_1}/\sigma^{u_0}\|}}$ and $C_v = \sqrt{\frac{\sigma^{v_1}/\sigma^{v_0}}{\|\sigma^{v_1}/\sigma^{v_0}\|}}$. These rotors encode the transformation that take one constant-coordinate surface (σ^{u_0}) to another (σ^{u_1}).

As discussed in Section 2.4.5, because we will be executing a continuous transformation, rather than calculate the square root of normalized ratios, we calculate half the logarithm of normalized ratios. For instance for κ_u with $C_u^2 = (\sigma^{u_1}/\sigma^{u_0})/(\|\sigma^{u_1}/\sigma^{u_0}\|)$:

$$\kappa_u = \frac{1}{2}\log(C_u^2) \tag{6.1}$$

and

$$\kappa_v = \frac{1}{2}\log(C_v^2). \tag{6.2}$$

These logarithms are 2-blade point pairs which can be linearly weighted and then exponentiated to interpolate the transformation. For u_x evaluated in the range [0,1]:

$$C_{u_x} = e^{x\kappa_u} \tag{6.3}$$

and

$$C_{v_y} = e^{y\kappa_v} \tag{6.4}$$

In Figure 6.3 we draw the 2-blades κ_u and κ_v in their undualized representation as circles.

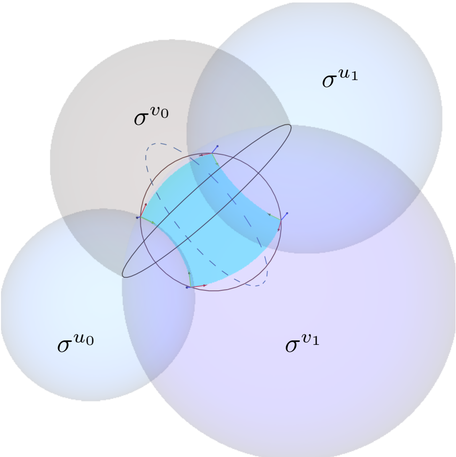


Figure 6.3: To construct a cyclidic patch, we generate rotors from spheres tangent to the circular edges of the patch. Each sphere represents a *constant coordinate surface*. Individual principal 'simple' rotors C_u and C_v in the u and v directions of a patch are the *square root of the normalized ratio of coordinate surface spheres*. The logarithms of these rotors are point pairs. Here we render their undualized representation as circles, both real (in the case the principal coordinate surface spheres intersect) and imaginary (when they do not).

The geometric constraints on the concircular system ensures that the point pair generators commute, a condition which allows us to compose them into a conformal rotor:

$$C_{u_x v_y} = C_{v_y} C_{u_x} \tag{6.5}$$

with u_x and v_y each evaluated in the range of [0,1]. These rotors are applied to a point (or other geometric primitive) at one corner of the patch, $p = p^{(0,0)}$, at (u_0, v_0) in order to transform it to a point $p^{(x,y)}$ at (u_x, v_y). Thus our mapping is:

$$f : (x,y) \mapsto C_{u_x v_y}[p]. \tag{6.6}$$

6.3.2 Constructing Tangents

To better seat our treatment within the larger construction of a geometric calculus, we will emphasize the direct use of tangent elements accessible in the conformal geometric algebra. This also enables us to leverage the findings of Bobenko and Huhnen-Venedey in their use of

Lie geometry, based as it is on **contact** elements. We therefore make explicit use of several basic operations in CGA relating tangent elements to round elements.[2] The first,

$$K = p \rfloor \hat{\Sigma} \tag{6.7}$$

is the extraction of the tangent bivector of a direct sphere Σ at a point p (assuming p lies on Σ) where the hat symbol $\hat{}$ signifies an involution (Equation 2.23 – for a k-graded element: $\hat{X}_k = -1^k X_k$). For a sphere Σ, Equation 6.7 returns a null-valued (zero radius) circle. For a circle K the same expression returns a null-valued point pair:

$$\kappa = p \rfloor \hat{K} \tag{6.8}$$

As a corollary we can build round elements from a tangent and a point:

$$K = p \wedge \kappa \tag{6.9}$$

which is the direct construction of a circle from a point p on it and tangent vector element κ along it. Similarly,

$$\Sigma = p \wedge K \tag{6.10}$$

is the direct construction of a sphere from a point p and tangent bivector element K. We also find it a useful to remember that homogeneous tangent vector elements κ and K are easily constructed by translation of a tangent vector and bivector at the origin, and that geometrically speaking they are null point pairs and null circles, possessing a weight and an well-defined orientation but no radius.

---

[2] Many components of these algorithms can be found in the essential table 14.1 on page 407 of [45], and in section 15.2 of that text.

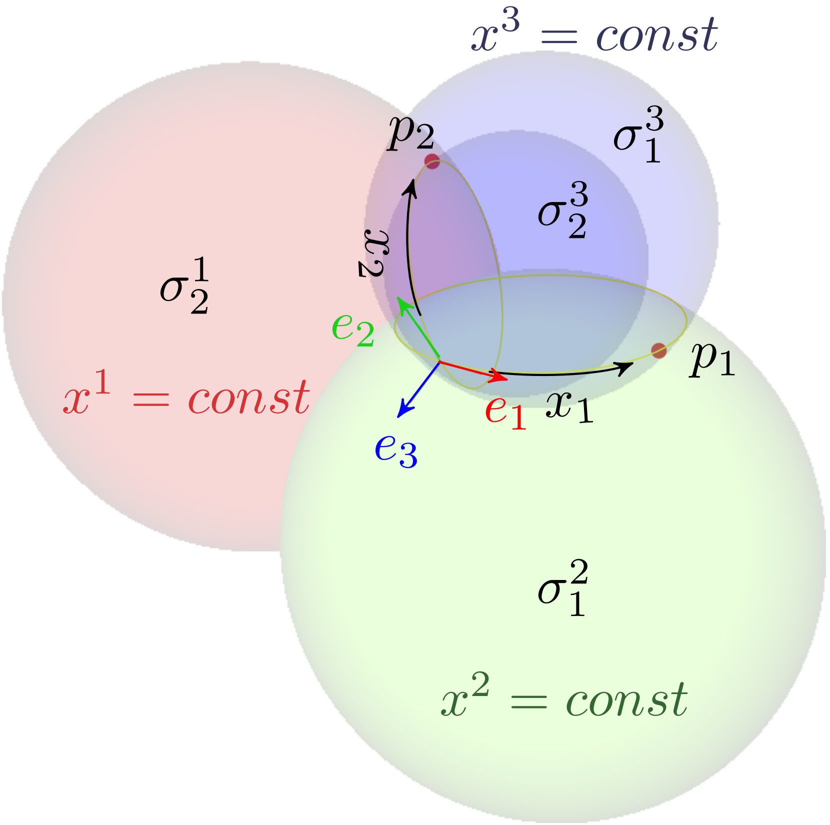


Figure 6.4: To discretize a *triply orthogonal curvilinear coordinate system* we can use a *6-sphere coordinate system* as a local unit (for more on different types of coordinate systems, see Moon and Spencer [104]). Consider an orthogonal frame of *coordinate axes* $\{e_k\}$ encoded by null tangents at p with $k = 1,2,3$. *Coordinate surfaces* $\{\sigma_j^k\}$ are represented by spheres tangent to the frame with normal e_k. Given scalar coordinate weights $\{x^k\}$ assigned to the $\{e_k\}$ each σ_j^k sphere represents a surface of constant x^k as we travel along the e_j direction. We use superscripts to denote the coordinate that is held constant, and subscripts to denote the tangent direction along which we travel. To create circular *coordinate curves*, we specify two constant coordinate surface spheres ($x^3 = const$, in blue) with e_3 as a normal, each one corresponding to a different orthogonal tangent direction denoted in the subscript. The coordinate curves x_1 and x_2 are circle intersections of the (blue) contact surfaces σ_1^3 and σ_2^3 with σ_1^2 and σ_2^1 respectively, a relation known as Dupin's theorem. A patch construction could proceed by picking two points p_1 and p_2 along these curves to define a circle with p which we can now interpret as the foundation of the principal cyclidic patch of Figure 6.1. Then, after picking a fourth concircular point we can identify the local coordinate surfaces at all points using the iterative mechanisms of Section 6.3.5. Not figured are the two coordinate surfaces σ_3^1 and σ_3^2 of constant x^1 and x^2, respectively, along the e_3 direction. Using this third direction to discretize volumes is explored in Section 6.4.

6.3.3 Triply Orthogonal Coordinates

As Figure 6.4 demonstrates, we can begin our synthetic construction with the notion of a triply orthogonal local frame $\{e_k | k \in 1,2,3\}$ at p. We label σ^i_j the principal contact sphere with normal e_i at p along direction e_j, which should be understood as representing a constant coordinate surface along the e_j direction. The raising of the index associates our construction with Hestenes' notation for curvilinear coordinate systems in [69].[3] In the case that it is unambiguous in which direction we are moving, we drop the lower index. In our notation for cyclidic nets of the previous section, we omit the subscript since it is clear in which direction we are moving, and instead specify only the coordinate that is held constant. The full notation for σ^{u_0} would read $\sigma^{u_0}_v$ to signify a surface of constant coordinate ($u = 0$) as we move the in the v direction along the patch.

The coordinate curve in the e_i direction is defined as the union of constant coordinate surfaces $\bigwedge \sigma^j_i, j \neq i$. Note that in this sense each surface σ^j_i encodes a partial derivative $\frac{\partial j}{\partial i}$ and each coordinate curve circle can be thought of as the undualized *exterior product of partial derivatives* which contribute to its definition (see Footnote 3).

6.3.4 Null Tangents as Coordinate Surface Generators

To emphasize the fact that our frame is a null **tangent frame** at a point in space, we write $\{\kappa_k\}$ to signify the tangent vectors $\{e_k\}$ have been translated to point p:

$$\kappa_k = T_p[n_o \mathbf{v}_k] \tag{6.11}$$

---

[3] In [69], Hestenes proposes the "tangential derivative as the most fundamental of all concepts of derivative", and explains that the relation between a tangential frame $\{e_k\}$ on a manifold and its inverse *reciprocal* frame $\{e^k\}$ reveals that all "*the coordinate curves are intersections of coordinate surfaces*" (p.34, emphasis in original). Intriguingly, this relationship is precisely Dupin's theorem, and suggests to us that the rationalization of cyclidic transformations using the mappings of orthogonal conformal rotors could provide clues as to how to discretize differentiable manifolds in general in the conformal model – namely, by treating *reciprocal tangent frames as normals of coordinate surface spheres*.

where T_p is a translation rotor and $\mathbf{v}_k$ is the Euclidean vector of our local e_k. One can create constant coordinate surfaces by applying a simple conformal rotor $e^{-\frac{c}{2}\kappa_k}$ to a plane through p with e_k normal, thereby bending the plane into a sphere through p. As we discussed in Section 5.3.1 and in a previous text [28], such a curvature generator is constructed by translating a tangent vector from the origin to p, thereby creating a null point pair κ which squares to 0. Our coordinate surface generating rotor is

$$C = 1 - \frac{c}{2}\kappa, \tag{6.12}$$

a simple result which stems directly from following the rules for exponentiation of a 2-blade in Equation 2.62 which states that

$$C = e^{-\frac{c}{2}\kappa} = 1 - \frac{c}{2}\kappa \tag{6.13}$$

when $\kappa\kappa = 0$. In these equations, $c = \frac{1}{r}$ is a scalar-valued weight that specifies the curvature in the κ direction, where r is the radius of curvature (see Figure 5.7). In a triply orthogonal coordinate system, as we travel along one direction, there will be *two* constant coordinate surfaces, one for each other direction. For each tangent vector κ_k we can sum curvatures in the other two κ_i and κ_j directions, essentially adding partial derivatives in the exponent:

$$C_k = e^{-(c_i\kappa_i + c_j\kappa_j)} \tag{6.14}$$

Applied to a point $T_k[p]$ that has been translated along κ_k, this rotor specifies a new point p_k that has travelled along a curve in both κ_i and κ_j directions:

$$p_k = C_k T_k[p] \tag{6.15}$$

6.3.5 Coordinate Surface Spheres at Four Points on a Circle

If the p_1 and p_2 along the coordinate curve are already known, then the coordinate surfaces can be constructed directly by application of Equation 6.10. This happens in practice when we are given four concircular points $\{p^{(i,j)}\}$ and a starting frame at one of them. Figures 6.1, 6.2 and 6.8 were in fact generated this way. κ_2 of tangent frame $\{\kappa_k\}$ at $p^{(0,0)}$, notated $\kappa_2^{(0,0)}$, is dual to a tangent bivector $K_2^{(0,0)}$ which wedged with the point $p^{(1,0)}$, creates a coordinate surface sphere σ_1^2 on which another tangent vector $\kappa_2^{(1,0)}$ at $p^{(1,0)}$ we can extract by application of Equation 6.7:

$$\kappa_2^{(1,0)} = (p^{(1,0)} \cdot ((\kappa_2^{(0,0)})^{-*} \wedge p^{(1,0)}))^*.$$

The entire tangent frame at $p_{(1,0)}$ can be similarly derived,

$$\kappa_i^{(1,0)} = (p^{(1,0)} \cdot ((\kappa_i^{(0,0)})^{-*} \wedge p^{(1,0)})^*$$

as can subsequent points $p^{(1,1)}$ and $p^{(0,1)}$ and their associated local coordinate surfaces. If the points are numbered sequentially counterclockwise around the circle we have

$$\kappa_i^n = (p^n \cdot ((\kappa_i^{n-1})^{-*} \wedge p^n)^* \tag{6.16}$$

More compactly we can manipulate the tangent bivectors K_i^n themselves,

$$K_i^n = p^n \cdot (K_i^{n-1} \wedge p^n). \tag{6.17}$$

We visualize their dualized representation as tangent vectors in Figure 6.5.

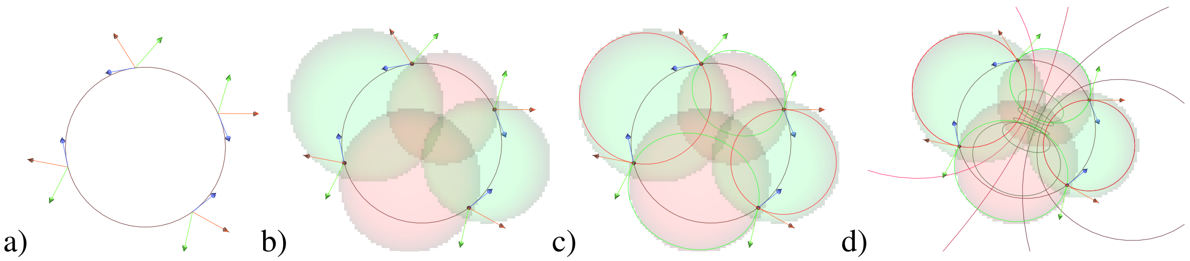


Figure 6.5: a) Given four concircular points and one frame at one point, all other frames are calculated by applying Equation 6.17 counter-clockwise from the known frame. We thereby extract tangent bivectors from coordinate surfaces generated at each point. Here we draw the normalized dual of the resulting tangent bivectors. Note that consecutive tangent frames are reflections of each other. b) To generate a patch, we interpolate between two coordinate surfaces bilinearly. c) Coordinate curves along the edges of our patch are circles. d) The rotor ratio of coordinate surfaces applied directly to the circular edge curves begins to suggest a surface.

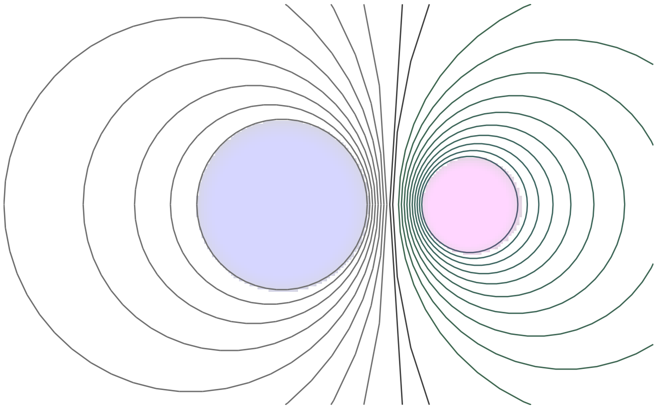


Figure 6.6: Taking a sphere to a sphere by exponentiation of the linearly weighted log of a simple rotor. Here we draw the circle cross section of the sphere during its transformation.

6.3.6 Rationalization of Coordinate Surfaces with Simple Rotors

With the system of principal contact elements in place with Equation 6.17, let us explore their **rationalization**, a method of discretizing the transformation of curvatures. In the intuitively quantitative world of geometric algebra, we must calculate **ratios** of elements in order to rationalize their transformation. Thus we first calculate the ratio of spheres, for example to rationalize the u coordinate surfaces we find the ratio:

$$C_s = \frac{\sigma^{u_1}}{\sigma^{u_0}} \tag{6.18}$$

which gives us a scalar and point pair bivector. We normalize this product by dividing out the reverse norm: $C_n = C_s / \|C_s\|$, where the reverse norm is defined as in Equation 2.21.

Our normalized rotor C_n represents *twice* the transformation that takes σ^{u_0} to σ^{u_1}. Thus, we take half its logarithm as in Equation 6.1, and this logarithm we can weigh linearly and then re-exponentiate using Equation 6.13. The logarithm itself is expressed in Equation 2.69.

We calculate κ_v the same way, and now have the components necessary to plug into Equation 6.6. Below we use Equation 6.20 – which is a simple modification to Equation 2.69 – to account for two choices of direction of motion around the orbit.

6.3.7 The Direction of Interpolation

Given two *intersecting* spheres, the conformal transformation between them can occur clockwise or counterclockwise. As in any rotation, the direction in which the transformation occurs determines the points that are evaluated along the way. Figure 6.7 illustrates this difference, and presents the need for additional measures to ensure the correct orientation of rationalization.

The direction is determined by the log function. The default direction uses

$$\frac{\operatorname{atan2}(\sqrt{-s^2}, c)}{\sqrt{-s^2}} s \tag{6.19}$$

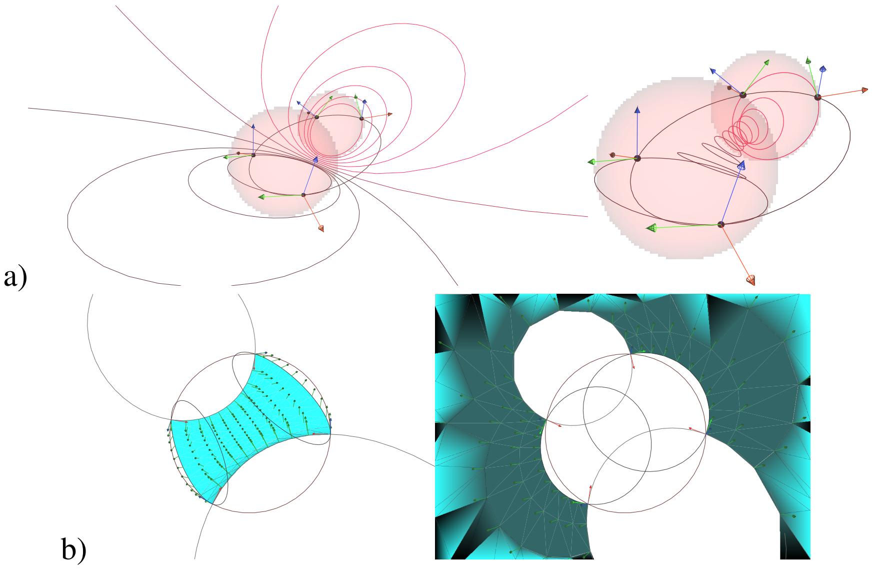


Figure 6.7: Given two intersecting spheres, there are two directions that one sphere can conformally transform into the other. a) Circles undergoing transformation in both directions, b) a surface generated in two alternate directions. Additional measures (such as Equation 6.21) are needed to determine the correct direction and to determine the full interval of a given direction.

whereas an alternative uses a modified weighting in the opposite direction:

$$\frac{-(\pi - \operatorname{atan2}(\sqrt{-s^2}, c))}{\sqrt{-s^2}} s \tag{6.20}$$

We find through experiment on a computer that using Equation 6.20 is necessary for calculating κ_u (resp., κ_v) precisely when our initial corner point $p^{(0,0)}$ has a negative dot product with the opposing coordinate surface sphere σ^{u_1} (resp., σ^{v_1}). Calling our alternative $\log_{alt}$ we have

$$\kappa_{u,v} = \begin{cases} \log_{alt}(C_{u,v}) & \text{if } p^{(0,0)} \cdot \sigma^{u_1,v_1} < 0 \\ \log(C_{u,v}) & \text{otherwise.} \end{cases} \tag{6.21}$$

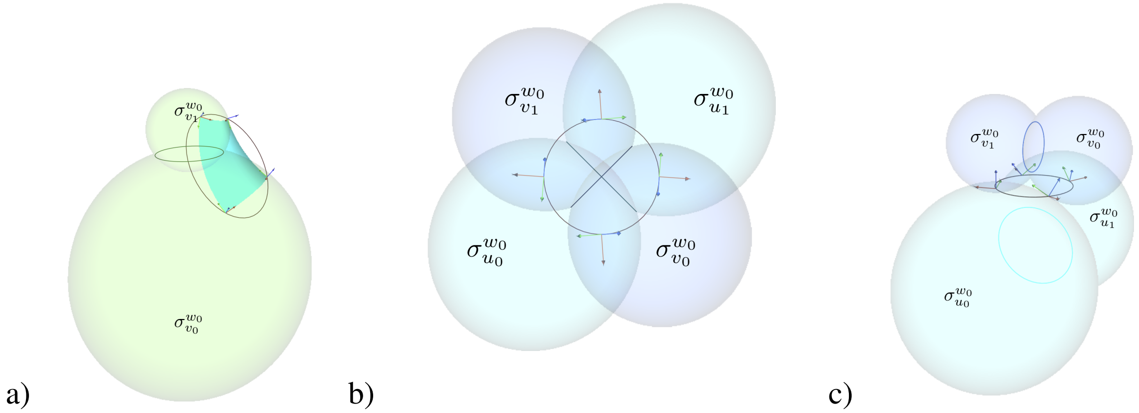


Figure 6.8: a) A patch is a C^1-smooth blend between coordinate surfaces of constant coordinate w_0, that is as a constant along the normal axis. Here $\sigma^{w_0}_{v_0}$ (resp. $\sigma^{w_0}_{v_1}$) represents the principal contact sphere change in the normal along the v_0 (resp. v_1) edges. b) and c) Two views of principal contact spheres tangent to the surface frames.

6.3.8 Surface Blending

Thus far (with the exception of Figure 6.4) we have primarily visualized coordinate surfaces in two directions: σ^u_v and σ^v_u. Here we discuss the third coordinate surface direction σ^w which can be constructed with respect to both u and v as σ^w_u and σ^w_v. These surfaces are the **osculating** spheres tangent to the uv cyclidic surface. We use the notation σ^{w_0} on these surfaces to indicate that uv surface itself lies on the constant $(w_0 = 0)$ in the direction normal to the surface. Given two contact surface spheres, two points on one of the surfaces is sufficient to identify the four concicular points from which a blending patch can be constructed, essentially one that interpolates between the two sphere surfaces (Figure 6.8a).

Not *any* circle through these two points can carry this patch – Algorithm 6.1 details a simple construction for finding a patch that blends between the two osculating contact spheres. Figure 6.9 illustrates the method, and Figure 6.10 depicts various results. This important technique reveals that four points will be concircular if they are constructed with the orthogonal plunge.

Algorithm 6.1 Defining principal patch parameters from two osculating contact spheres.

```
1. Pick a point $p$ on one of the spheres $\sigma_{v_1}$.
2. Find the plunge $K$ through p orthogonal to both spheres: $K = p \wedge \sigma_{v_0} \wedge \sigma_{v_1}$.
3. Find the first intersection point of $K$ with $\sigma_{v_0}$.
4. Repeat for another point on $\sigma_{v_1}$.
5. The tangent normal to $\sigma_{v_1}$ at $p$ defines $\kappa_3$. Pick a rotation angle about this axis to define
   $\kappa_1$ and $\kappa_2$.
6. Define $\{\kappa_k\}$ for each frame using Equations 6.16 and 6.17.
```

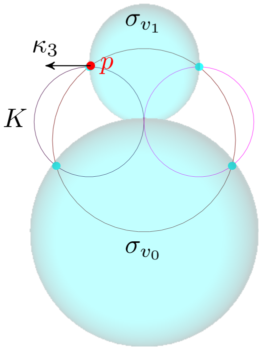


Figure 6.9: Construction of a circle net to blend between two spheres. Given a point p on sphere σ_{v_1}, we calculate the orthogonal plunge K by wedging p with both spheres: $K = p \wedge \sigma_{v_0} \wedge \sigma_{v_1}$. We then find the intersection of K with σ_{v_0}. We repeat for a second point on σ_{v_1}. All four points thus defined will be concircular. κ_3 is the tangent normal to the surface patch at p.

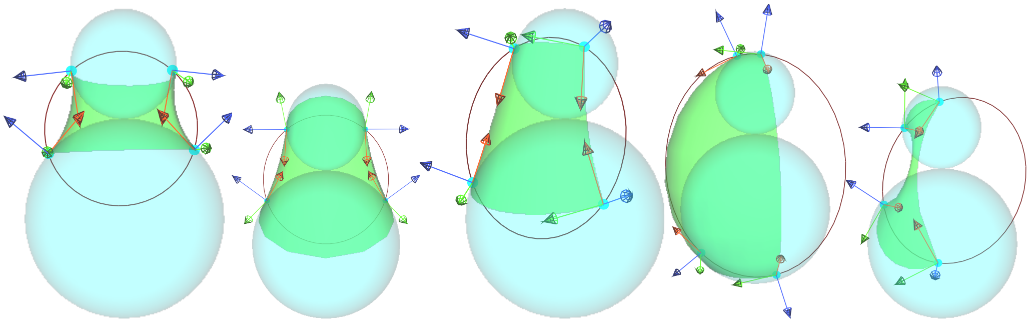


Figure 6.10: Various patches blending two spheres.

6.4 Application: Freeform Structures

Much of the interest in cyclidic patches comes from their structural properties. In the introduction to this Chapter we discussed some of these benefits. First, as the mappings are conformal, all local angles are preserved, which eliminates shearing. This results in torsion-free nodes which facilitates physical construction [16, 97]. In computer graphics, this same property helps preserve texture and mesh detail under deformation [31]. Second, as patches are built with circles rather than lines, they provide curvature discretizations which converge more quickly to the continuous case. Third, both of the first two properties are shared by offset surfaces along the normals, and are in fact generalizable to an n-D smooth orthogonal coordinate system for working with differential geometry [17].

As a result, parameterizing surfaces and volumes with an orthogonal curvilinear system can facilitate their deformation and construction. Such designs can come to fruition as freeform architectures covered in flat glass, where offsetting along normals to the surface is critical, and where the ability to manufacture a single 90 degree node reduces costs. The ability to blend between shapes in a precise way is also useful in machine-milling of small parts, where oddly shaped curved pieces are often necessary [52, 86]. Conformal lattices can provide a robust way to animate virtual forms.

C^1 smooth surfaces are built from a series of patches in Figure 6.11. A new patch is added to an existing edge by specifying an additional tangent sphere. The two additional tangent frames necessary to define the second patch are then found using the procedure outlined in Algorithm 6.1. This process can also occur in a direction *orthogonal* to the first patch, as shown in Figures 6.12 and 6.13. In this way a 3D volume can be described, as demonstrated by Bobenko and Huhnen-Venedey in [17], where it is shown that three patches are sufficient to define a six-sided volume, or **hexahedron**. Here we show how such a system can be used to warp a mesh in a way similar to the trilinear interpolation of motors demonstrated in Figure 5.4.

Given a hexahedron of patches arranged as six sides of a warped cube, the technique for specifying a point at coordinate (x, y, z) with $x, y, z \in [0, 1]$ requires transforming the normal direction σ^w coordinate surfaces on opposite sides of the cube and then finding a new rotor from their ratio. The complete steps are enumerated in Algorithm 6.2 and its application to finding the transformed positions of a warped sphere in Figure 6.14.

Algorithm 6.2 Calculating A 3D Conformal Coordinate

---

1. Given a hexahedron (six-sides) of surface patches, pick the two opposing patches at $w = 0$ and $w = 1$.
2. Calculate the C_{u_x} rotor of each of the two opposing patches using Equation 6.3.
3. Apply these rotors to the constant coordinate osculating surfaces $\sigma_{u_0}^{w_0}$ and $\sigma_{u_0}^{w_1}$ of each corresponding patch. Call these transformed two spheres $\sigma_{u_x}^{w_0}$ and $\sigma_{u_x}^{w_1}$.
4. Generate a transformation from the ratio of these two transformed surfaces as $C_{w_z} = e^{-\frac{1}{2}z\log([\sigma_{u_x}^{w_1}/\sigma_{u_x}^{w_0}]_{\text{normalized}})}$.
5. Multiply this transformation by $C_{u,v}$, the bilinear rotor of the $w = 0$ patch, found using Equation 6.5.
6. The result, $C_{w_z}C_{u_x,v_y}$ can be applied to the point $p^{(0,0,0)}$ to find its transformed position in the 3D conformal coordinate grid.

6.5 Discussion and Future Work

We have introduced a new method of rationalizing cyclidic nets by composing conformal transformations from orthogonal coordinate surface pairs. Our technique emerges from the orthogonal decomposition of general conformal transformations uncovered by Dorst and Valkenburg in [46] and detailed further in [43]. We use concircularity as a constraint to ensure our bivector point pairs are well-chosen. Constructing spheres with the outer product of points and bivector tangents, we touch base with contact geometry and the curvilinear coordinates explored by Bobenko and Huhnen-Venedy in [17]. To develop continuous blending

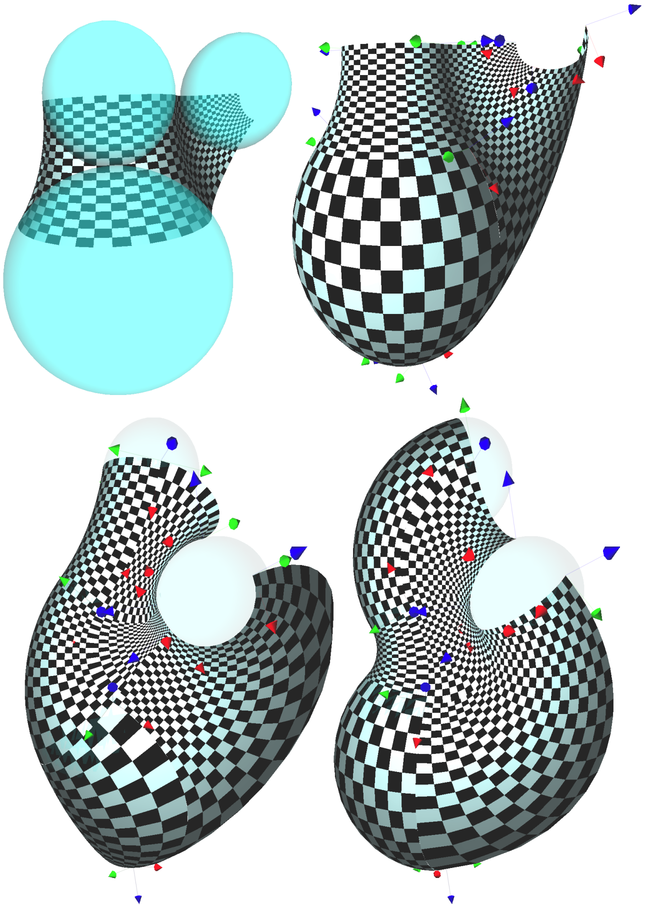


Figure 6.11: C^1 smooth cylidic net surfaces constructed from a series of principal patches.

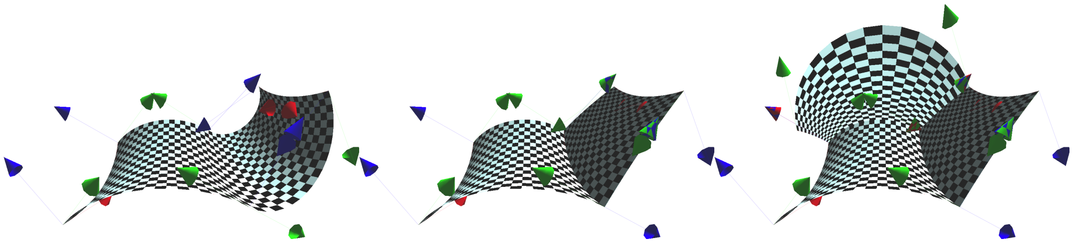


Figure 6.12: Smooth surfaces can break a corner orthogonally to form nets along a normal. Three sides are sufficient to uniquely define the remaining three necessary to define a hexahedron volume.

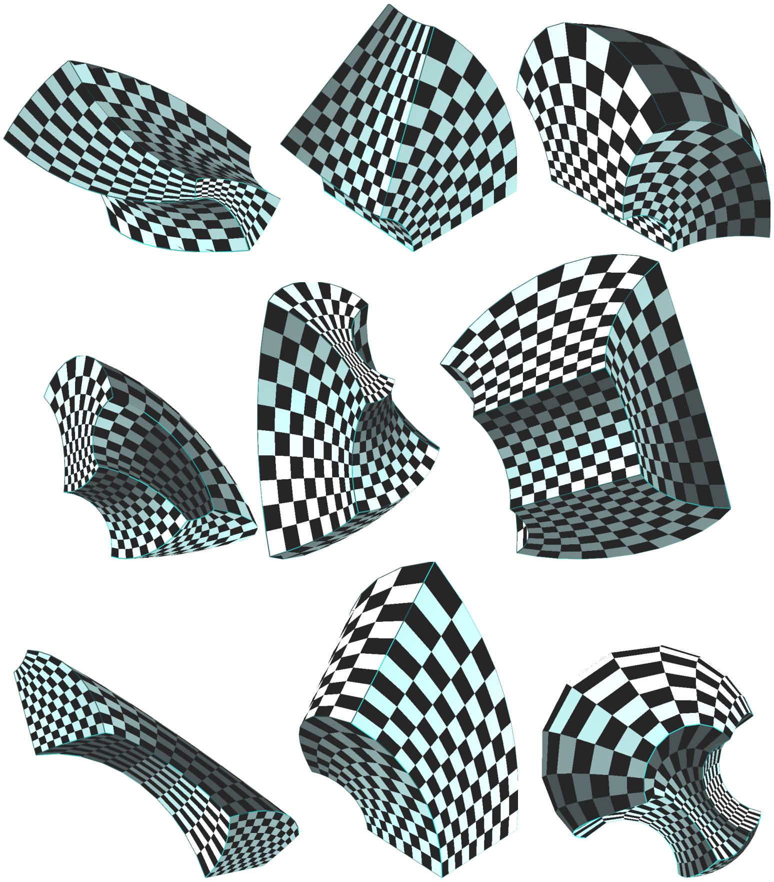


Figure 6.13: Circular nets can be extended in a third direction to discretize volumes.

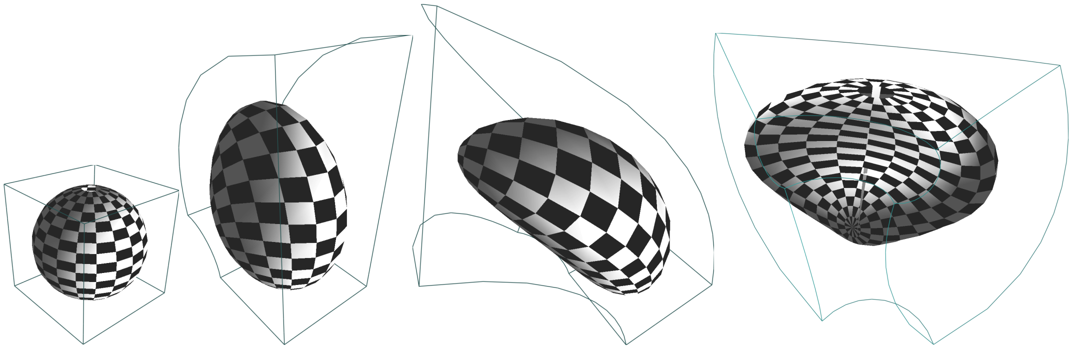


Figure 6.14: Discretized volumes used to parametrize the conformal deformation of a sphere.

techniques we rely on the orthogonal plunge of circle through two coordinate surface spheres. In contrast to matrix-based rationalization techniques, the incorporation of spinor theory into geometric algebra allows us to generate transformations through ratios of the coordinate surfaces themselves. To help with this, we use a local 6-sphere coordinate system to encode curvature in every possible direction.

By using geometric algebra we are able to discretize smooth shapes in a way that preserves the two goals of discrete differential geometry proposed by Bobenko and Suris in [18]: the *transformation group principle* (wherein discretizations and their smooth counterparts transform the same way) and the *consistency principle* (whereby constructions can be extended to higher dimensions). Because of the structure-preserving nature of conformal transformations within the CGA mechanism, our composed transformations are as capable of operating on tangents and normals as well as points, greatly simplifying basic calculations in differential geometry. We suspect that approaching differential geometry by careful study of integration of conformal mappings will help in developing the discrete geometric calculus within the conformal model. In our treatment, our frames are already orthogonal – a condition which can be applied to non-orthogonal frames using the reciprocal construction of geometric calculus. Hestenes' writings are, as usual, a good place to start this mapping [68, 72, 75] as is Sobczyk's simplicial calculus [125] and the rich literature on discrete exterior calculus [31, 32]. We would like to more carefully consider the relationship between the rotors that transforms these spheres across a surface patch and the *shape tensor* and *shape bivector* or *curl*, to better pin-point the pair generators that most clearly and generally match Hestenes' definition of the shape bivector as the "angular velocity of the pseudoscalar as it slides along the manifold" [75]. Explicating such relationships will give more space for a discrete calculus to form.

A good next step in our formulation will be to analyze an input simplicial surface and try to find its closest conformal representation through piecewise integration of cyclidic patches. It is possible that, in order to fully articulate a discete differential geometry, we decide to move into the space of $\mathbb{R}^{4,2}$ to allow for geometries other than Möbius, in particular those of *Laguerre*. In [18], Bobenko and Suris construct discretizations using Lie Sphere Geometry, of which the Möbius geometry of $\mathbb{R}^{n+1,1}$ is one subset and Laguerre geometry the other. In [97], Liu et al use a Laguerre-based approach to discretization based on *conical* rather than circular quad meshes. Recently, Krasauskas has used Laguerre transformations in the geometric algebra model of $\mathbb{R}^{4,2}$ to solve hole-filling problems [86]. As opposed to the Möbius transformations which preserve points (spheres of zero radius), transformations Laguerre geometry preserves hyperplanes (spheres of infinite radius). Their combination, Lie Geometry, preserves *oriented contact*. The surfaces that can be carved out in these other geometries include Darboux cyclides, more general versions of the Dupin cyclides depicted here. Thus having seen the power of rationalization using Möbius transforms, and the demonstrations of Laguerre transformations in the aforementioned texts, we might consider moving to the more encompassing Lie Sphere Geometry. Leo Dorst has also suggested potential advantages to working with the Lie Sphere system of oriented contact, for instance for matching broken pieces of pottery. Luckily, with the universal geometric algebra, one could conduct such experiments by extending the constructive methods developed here.

Chapter 7

Conclusion: A Mechanism for Design

7.1 Summary

The current work began as a simple question – how can we design structures with geometric algebra? The investigation of this question has unfolded into an operational strategy for spatial composition on a computer: a synthesis of form through transformation. Felix Klein's identification of transformation as the central characteristic defining a space is honored by the model of space adopted here. Geometric algebra allows us to *practice the processes that constitute spatial systems*. With enough practice, these processes can lead to a unified methodology for structural design.

To advance this methodology, we have developed techniques for encoding three properties of spatial structures – symmetric, kinematic, and curvilinear. Examining transformation (reflecting, folding, twisting, bending, and knotting), distance constraints (round incidence relationships), and rationalization (logarithms of ratios) we have implemented a collection of constructive geometric design techniques with a single unifying spatial computing engine. This relationship between technique, parameter, property, and structure is outlined in Table 7.1.

From a single axiom relating the dot product to a quadratic product, we modelled a train

Technique	Transformation	Incidence	Tensor
Parameter	Groups	Constraint	Differential
Property	Symmetry	Kinematics	Curvature
Structure	Balanced	Deployable	Freeform

Table 7.1: The relations between techniques, parameters, properties, and structures that have been developed in this text.

of thought, first establishing relationship between the geometric product and the associated algebraic concept of inversion and the fundamental spatial concept of reflection. We then explored how – from from that algebraic-spatial connection– the catalysis of mathematical invention generates a coherent system of relationships between geometric numbers. With these relationships thus crystallized, we explored how they can be used to place constraints on each other in order to articulate position and orientation in moving structures. Finally, to demonstrate the elegance achieved, we offered a direct method for rationalizing curved surfaces via a tensor product of sphere ratios, and illustrated a method for conformally warping a mesh using this discretization of curvature.

The three spatial systems selected were not chosen at random. From start to finish we generated a pedagogy for practice: the notion of a screw displacement as a rotation and a translation discussed at the end of Chapter 3 is used to model kinematics in Chapter 4. The logarithm of such motors are used to introduce a method interpolation in Chapter 4, and the application of such interpolation methods onto general bivectors reveals further forms. In Chapter 6 these are regimented into the deliberate construction of curvilinear surfaces and volumes. A single mathematical model of 3D space has proven adequate for designing a range of spatial configurations. We have not only shown how individual expressions can be used to generate structures, but the processes by which such expressions can be extrapolated. Using a comprehensive language of space, our processes can evolve intuitively; further forms suggest themselves from current manipulations.

Within each Chapter, we argued that the production of geometric expressions enables a novel way to design structures. Applications of our synthesis techniques include machine parts, tensegrity systems, deployable structures, and freeform architecture, bringing the theoretical underpinnings of our models one step closer to real-world manufacture. By visualizing this language of space, we have encouraged the adoption of these algebraic methods by digital practitioners. We find the expressive powers of the algebra make it an essential tool for those who seek to develop their spatial thinking.

7.2 Contribution

By restricting ourselves to examining the generative prowess of conformal geometric algebra model of 3D space, we have pushed the range of forms that have been synthesized from scratch within this system. We have used constructive techniques – small reusable algebraic expressions – to demonstrate how a full-featured articulation of space can be developed from a single framework. Without recourse to mathematical proofs, we have contributed a *proof-of-geometric-concept* of the role of geometric algebra in design practice: illustration of the raw power of these expressions – when used in combination – to generate complex forms.

This work can be characterized as a manual for practicing geometric algebra. It is precisely the ability to implement symmetry groups, linkage mechanisms, robotic chains, and rationalized curvilinear surfaces which we offer as crystal-clear evidence of the algebra's ability to communicate form. Each synthesis technique has relied heavily on the previous work of others, both inside and outside the geometric algebra community, and by sheer necessity we have built upon known algorithms by unearthing details critical to their use in construction. Throughout we have consistently returned to the groundbreaking textbook by Dorst, Mann, and Fontijne [45], which provides much of the syntax of geometric relationships (for instance, the orthogonal *plunge* of Chapter 6). Through synthesis, we have harnessed the basic operations of an articulate system into a comprehensive language.

In Chapter 3, we implemented crystal systems, and focussed on transducing the transformations expressed in Hestenes and Holt in [73] into visual forms, in order to grow comfortable with using symmetry groups as generators of pattern. In doing so, we not only clarified how such a system can be implemented procedurally on a computer (see, for instance, the Space Group Visualizer of Hitzer and Perwass [79]), but also discovered a unique way to experiment with the geometrical design of closed surfaces and tensegrity systems.

In Chapter 4, we implemented kinematic systems based on the motor algebra approach outlined by Bayro-Corrochano and Kähler in [10]. Applying the FABRIK method of Aristidou and Lasenby in [4] and Aristidou, Chrysanthou, and Lasenby in [3], we detailed our own specific technique for calculating the inverse kinematics of a chain of revolute joints, where care must be taken when constructing rotation sequences. As an example of a closed-form solution, we presented a novel technique for the construction of a Bennett overconstrained mechanism, and chained these linkages together based on the work of You and Chen in [138]. We then used our new formulas to propose a few designs for articulating structures.

In Chapter 5 we detailed the twist interpolations explicated by Wareham, Cameron and Lasenby in [134] and the conformal knotted orbits of Dorst and Valkenburg in [46], combining them to model a Hopf fibration and discovering unique commutative properties of antipodal fibers in the process. This abstract play emerges as an exercise in order to prepare for Chapter 6 where we developed a novel technique for rationalizing surfaces with ratios of spheres. We experimented with this system to carve a few surfaces and volumes, and demonstrated its use in defining conformally warped coordinate systems of use in animation and architecture. We note that the technique seems extendable to other dimensions.

It is clear that working with a proper language of space significantly reduces time-to-discovery by ensuring that any geometric experiment can inform a subsequent one. In order to implement the formulas in this document, it was necessary to write a conformal geometric algebra library, the implementation of which is outlined in Appendix 2. Two core features of this library are a) its generic nature – *any* arbitrary metric is supported and b) its terse formalism: the expressions are compact. Thus a software library built specifically for the conformal model can in fact be used to explore other models as well. The tool we built is yet another example of the *constructive strategy to design* enabled by geometric algebra, where the implementation of a solution to a specific problem induces a general solution to other as-yet unspecified problems.

7.3 Future Work

Geometric algebra opens the channels of transdisciplinary communication by creating a space for researchers to extend and share their algorithmic methods. The presence of a unified framework for synthesizing spatial configurations helps us draw relationships between seemingly disparate fields of geometric computing. By further activating the algebra in the realm of digital design practice, we have opened participation in the development of spatial research to visual thinkers. An important step will be collaboration between researchers from computer graphics, materials science, robotics, and structural engineering in order to explore the use of these methods in practical applications beyond the proof-of-concepts presented here.

Deployable Structures and Origami In Section 2.6 we briefly discussed the use of round elements as distance constraint parameters in isometric deformations of the plane. Exploration of this topic, perhaps by pairing geometric methods with other distance-constraint solvers such as Verlet integration, could advance complex configurations such as found in curved-crease origami or protein folding. The design of a foldable parabolic dish for satellite deployment, or a collapsible stent for arterial deployment could benefit from combining the crystallographic studies with such constraint systems.

Software: Plugins and GPU programming `Versor`, the software library written to support the geometric investigations of this thesis, should be extended as a plugin module to other software packages. Such integration will help gather insight from the artistic community in the design of spatial structures, and enable the communication of these parameters to other researchers. Thus it will prove useful to integrate the algorithms developed here into plugins for commonly used 3D modelling platforms such as Maya, Cinema4D, Houdini, Autodesk CAD, Rhino (and Grasshopper), Catia, as well as modules for creative coding such as OpenFrameworks and Processing. An online editor written in Javascript would help encourage researchers to play with geometric concepts. Python scripts will help scientists accept its utility in encoding their transformations.

Fabrication Realizing the formulations on these pages in physical form is a way to test the durability of the reasoning method in constructing actual forms, and the adaptability for iterating when things go awry in practice. With its exquisite articulation of space, transformation, and invariance, entire pipelines from design to manufacture can be written in the language of geometric algebra. Architects can explore geometric positions of spheres and circles, and structural engineers can probe their work to examine the stresses from within the same software framework. No more are design and construction separated by mathematics during different phases of manufacturing, but rather holistically treated through one continuous exploration of form.

Discrete Differential Geometry Geometric algebra has been developed into a rich calculus and in particular a new approach to differential geometry [75, 72, 68]. To date, only a single paper (Sobczyk [125]) has tackled the problem of discretization of this calculus, which is needed in order to fully develop computer-based techniques for mesh manipulation such as smoothing of meshing of point cloud data. We have helped pave a way forward in the use of the conformal model for surface and volume rationalization of Chapter 6. A necessary next step is to analyze a given mesh and find its conformally rationalized parameterization.

Generative Algorithms Once implemented, CGA makes it possible to write programs which experiment with different geometric traits, access various intersection points, conduct transformations, and then recalculate results based on some geometrically defined fitness function such as orthogonality of circles or coplanarity of points. In Section 4.4 we used intersections of round elements to satisfy distance based constraints, and it is feasible to extend this method to optimize for even more constaints, an idea presented by Aristidou, Chrysanthou, and Lasenby in [3]. Details of specific implementation strategies are an important next step to such a constraint network. Then, wedding these geometric constraints with adaptive algorithms could help evolve more articulate movements such as are needed for advanced design of machines.

Geometric Signal Processing / Geometric Audio As numbers are now imbued with geometric content, any filtering or processing of them using techniques learned in digital signal processing could be applied to the manipulations of forms. Conversely, the geometric manipulations of physical relationships could offer new audio synthesis techniques. This will require pairing Clifford analysis, the field devoted to the study of higher dimensional signal processing, with more user-friendly audio signal processing, the common element to both being the bivector exponential e^B. Programming languages that help in composition of signals, such as functional languages, may prove useful here. A collection of software that uses the functional paradigm to generate structure can be found in Footnote 1.

Topology and Braid Groups The natural transformation of knotted orbits of Section 5.3.3 suggests a potential contribution to the field of topological quantum computing, where information is recorded based on braid groups. The various types of conformal orbits also seem to be of use in continuous deformation of meshes, perhaps serving as continuous topological operators which can add or remove holes in a surface.

7.4 Final Thoughts

A graduate student asks her professor a simple question:

> Student: *How do you produce a rotation in 3D? I hear there are issues with*

matrices.

Professor: *Yes. Matrices can introduce "gimbal lock". It is best to use quaternions.*

This is the commonly agreed upon correct answer. Unfortunately, what happens next ignores an opportunity to fully define the production of proper transformations on a computer. The *quaternion* is variously characterized as a sort of complex number of higher order, sometimes considered a 4-dimensional vector, constrained by identities that seem magically inspired,[1] and typically implemented by copying someone else's code. This most assured recipe for rotating objects is implemented by graphics programmers without clear understanding of why their code works. Yet these esoteric "quaternions" are the mathematical objects we use to construct *simple* 3D rotations around the origin.

Student: *Is it an entity or an operator? In what dimension does it exist? How do we implement it?*

Professor: *Sounds like a good homework assignment.*

Rather than gloss over the intricacies of how transformations work in computer graphics, let us examine them more carefully. If we master them, what can we do?

For instance, the quaternion can now itself be understood as the composition of a more basic operation – the reflection – and is a type of *rotor* or *spinor* released from the origin. It is just one of a number of transformations we can construct from the basic combinatoric system called geometric algebra. The details of how a rotor is generated and applied are based on axioms that guide rotations in any dimension. In learning about rotors in general, we can learn about symmetry groups (a way of generating patterns), rigid body transformations (a way of generating mechanisms) and rationalization (a way of generating surfaces).

---

[1] The story told is that William Hamilton, working on an algebraic mechanism that could be used to multiple 3D vectors together the way complex numbers can in 2D, discovered the formulations $i^2 = j^2 = k^2 = ijk = -1$ while crossing the Brougham bridge in Ireland.

A good model of space can do more than calculate space itself. In 1898 Henri Poincaré wrote *On the Foundations of Geometry* [115], concluding that "space is not a form of our sensibility; it is an instrument which serves us not to represent things to ourselves, but to reason upon things." The idea that space itself is an instrument to reason finds resonance in the history of computing, where geometry has provided the backbone to analogue calculators.[2] This correlation continues in the digital domain: in his text *Calculating Space,* Konrad Zuse, the pioneer computer scientist, arranges a system of "digital particles" to model differential equations.[3] The arrangement of Zuse's digital particles affect how information is exchanged between them, creating patterns in the resulting process:[4] by this line of thinking, space in particular can facilitate calculation in general. Zuse's work points to an important principle: spatial logic is a cornerstone of computation.

The title of this dissertation is a tribute to Zuse's *Calculating Space*. In our case, *Articulating Space* entails both expressions *of* geometric concepts – as in articulating the concept of a imaginary circle – and expressions *with* geometric concepts – as in an articulating robot arm. Such a double articulation uses geometric primitives to provide details of relationships and movements, positioning space itself as both subject being articulated and object doing the articulating. In this way our spatial intuition gains agency; by using geometry in algebra, we are empowered to combine what we already know with what we can imagine in order to

---

[2] The instrument of space is used by computer scientists who rely on geometric reasoning to drive their calculating machinery. Charles Babbage used a network of differential gears to making a "thinking machine". Alfred Kempe found linkage mechanisms could be used to generate any algebraic curves. More recently, Alexei Kitaev used braids to model topological quantum computing. It is little surprise that a spatial problem is used to motivate Alonzo Church's historic essay on computability itself. Church's treatment, *An Unsolvable Problem of Elementary Number Theory* [23], begins with the example problem of determining whether two shapes are similar to each other – that is, whether two simplicial manifolds are topologically invariant under homeomorphism. Spatial concepts pervade the language of calculation: to *compute* is, etymologically speaking, to *put together*. Similarly, to *complicate* is to *fold together*.

[3] Crucially, space is the object and not the subject. A more accurate translation of the title of Zuse's work, *Rechnender Raum*, might be *Space, that is Calculating.*

[4] We have not developed such automata for simulating complex behaviors in this text, but the fundamental mathematical structures we did study are themselves "digital particles" which could augment Zuse's representations with significant geometric meaning. For other classic sources on digitization of actions, see Braitenburg's *Vehicles*: *Experiments in Synthetic Psychology*, which produces teleological behavior from basic orientation protocols, and *A New Kind of Science,* in which Wolfram achieves nearly biological behavior through deterministic local interactions.

create constructive tools for discovery.

We have endeavored to use space as an instrument with which to conduct research, providing details of several synthesis tactics within a larger strategic framework for spatial computing. By tethering our methodology to the axioms of a geometric algebra, we have ensured that all algorithmic inventions can inform the next problem we tackle. Many discoveries await the explorer of this system of articulating spaces. We hope it will inspire your own innovative use of forms in your formulas.

Appendix A

Compile-Time Geometric Algebra with C++11

A.1 Summary

Algorithms in this document have been implemented using `Versor`, a generic lightweight C++ library for visualizing geometric algebra. This appendix outlines a strategy for compiling high performance geometric algebras through the functional programming idioms of C++11 template metaprogramming [TMP]. In particular, we provide techniques for listing, sorting, and evaluating combinatoric instructions at compile time.[1]

---

[1] While C++ is inherently imperative, its templating framework is inherently functional. Extending this functional paradigm should be further explored in developing shape grammars. We list here, for reference, a short compendium of explorations of 3D modelling in the functional style. In an article entitled *Functional Geometry* [64], Henderson demonstrates the use of recursion in Scheme language to generate the geometric tessellations of MC Escher. The approach is developed by Mairson in to outline a constructive approach to the design of musical instruments, based on the research of the luthier François Denis [37]. Cousineau and Mauny devote a chapter in [100]to functional programming of drawings, using the language CAML to output the imperative language PostScript. In *Functional Differential Geometry* [128], Sussman and Wisdom propose a functional notation to define the classical calculus of manifolds using Scheme, and provide an example of modelling electrodynamics. Beynon specifies a *definitive* programming style for Computer-Assisted Design, which is similar to functional programming but with assignable variables (In this sense definitive programming is similar to Lisp programming paradigm)[15]. PLaSM [106] is a functional language extension to FL, a functional language developed by the IBM research group. PLaSM is specifically geared towards Computer Aided Design of architecture and outputs VRML (Virtual Reality Modeling Language). *Formian* is a language developed by Nooshin that implements *formex algebra*, a method he developed for "configuration processing" which manipulates polyhedra and finite elements [105]. Individual *formices* are graded dimensional elements which

Designing a software library that is both generic enough to leverage the vast expressivity of this combinatoric system and efficient enough to be used as a practical substitute for more commonly used techniques is a challenge. Current optimization strategies typically employ multi-stage solutions, generating code in one language and optimizing in another. With the end-user in mind, we aim to streamline the optimization process, reduce compilation time, and ease cross-platform use, while enabling fluency in n-dimensional vector spaces with arbitrary metrics such as $\mathbb{R}^{4,1}$ or $\mathbb{R}^{3,3}$. To accomplish these goals, we created a lightweight library that can compile quickly across a range of computers and devices. We leverage C++11 variadic templates and constant expressions to build a system of type creation that defines how any two multivectors combinatorically form new ones. In detailing the system, we look at compile-time list processing, efficient sorting procedures, and advanced type creation. The resulting 200kb of code generates inlined instructions of arbitrary metrics in arbitrary dimensions at compile time and can run real-time graphics applications on embedded systems. Accompanying documentation and code, tested on unix-like operating systems, is available at

`versor.mat.ucsb.edu` and `github.com/wolftype/versor`.

A.2 Computer Implementations of Geometric Algebra

The general nature of spatial expressivity is powerful when developing abstract thinking and building deductive logical systems. But that same genericity is a serious obstacle for implementing code, which is often most efficient when specialized. Indeed, the trade-off between universal applicability and computational efficiency is so common in programming languages that the very implementation of GA can serve as a useful way to test a language's ability to negotiate this conflict. In [76], Hildenbrand, Fontijne, Perwass and Dorst list four difficul-

---

can be composed together. The use of functional programming in robotic *control* has received much attention with the advent of functional reactive programming [FRP], a method used to organize *events* and *behaviors* in continuous and discrete systems[80, 110].

ties in developing an efficient GA computational system: the need for a large number of specialized types, the number of basic operations that must be coded between these types, the arbitrariness of the metric, and the exponential increase in combinatoric complexity with each increasing dimension. As a result of these difficulties, software solutions tend to be either slow to run, awkward to distribute, or slow to compile. There is little doubt that a major obstacle in wide-spread adoption of Geometric Algebra is this lack of a coherent software solution.

The most inefficient methods, such as GABLE for MatLab, store each geometric element in n-dimensional space in a $2^n \times 2^n$ matrix. Some faster implementations, such as the C++-based CluCalc [112], still keep track of which indices into these matrices are being used for any particular element of the algebra, causing run-time overhead. Multi-stage programming techniques provide further speed ups to the sparse matrix representation: for instance, Gaalop [77] is a precompiler which optimizes Clu scripts to run on GPUs. Generative approaches, such as Gaigen [51], construct implementation generators which output optimized code in multiple programming languages. This was used to create the GA Sandbox viewer [50]. The first version of Versor similarly used lua code generation to produce C header files which were then linked into a dynamic library [27]. The main difficulty in the code generation approach is that experimentation across metrics requires consistent generation of new code which makes it difficult to package and deploy a lightweight comprehensive system. These multi-stage compilations and Domain Specific Languages (DSL) invariably add an extra step to programming, either requiring the programmer to learn a new scripting language, or implement a second phase of code generation before entering the realm of the algebra. Since GA is relatively new compared to other mathematical domains, these are extra barriers to access best eliminated in order to encourage more widespread, immediate, computational experimentation.

A third approach – the one detailed here – leverages the functional-style metaprogramming capabilities of C++ to instruct the compiler to generate efficient code at compile time.

The software library that most closely resembles the one designed here is Gaalet [124]. The main differences are that Gaalet creates expression templates, which adds significantly to compile time, and does not include an optimized version of the conformal model, which is the most useful for the work developed in this thesis. Jaap Suter's *Clifford* library was also a metatemplated approach, though it seems to be no longer supported. Robert Valkenburg is also known to have experimented with expression-templating techniques using the Boost::Proto library for creating domain specific languages, though no publication of this work exists to date. Versor contains more than just an implementation of geometric algebra: it includes numerous classes and wrappers around the types of structure-forming and space-articulating algorithms included here.

Formal explanations of how the functional paradigm of metaprogramming maps to an efficient implementation of geometric algebra application using templates has yet to be written. However, purely functional implementations do exist as well. In [54] Fuchs and Théry examine a functional implementation of Geometric Algebra using binary trees, though it is unclear how to experiment with alternative metrics in that formulation. Modules for Mathematica [107, 2], and a symbolic parser for SymPy [19] are also inherently functional.

Finally, hardware accelerated approaches, which leverage FPGAs and embedded processors, have begun emerging as well [102, 53, 113].

A.3 Implementation Goals

We seek to ease the access to efficient GA by implementing a single-stage *pure* C++ templating alternative to the multi-stage approach. We submit that doing so facilitates development of generic GA code, including higher dimensional constructs and alternative geometries. We address each of the difficulties above in a lightweight header-only library of about 200kb. No domain specific languages must be learned, and no heavy reliance on standard template libraries. This allows us to investigate what is possible within native syntax and ensures portability of use across any machine with an adequate C++11 compiler. Integration with any language that has a C-like api of course remains possible: using the Clang compiler, we can compile to intermediate representation and translate to other languages from there.

Advantages to keeping everything in one C++ language environment with minimal use of standard template libraries include:

1. **Portable** – software deployment is easier when no secondary language support is needed, anyone with gcc4.7 or clang3.2 or above can compile with a simple Makefile. This library can run natively on the RaspberryPi, for instance, or a smartphone. No new scripting language needs to be learned.

2. **Stable** - the strongly-typed templating system can help reduce run-time errors since the compilation phase is more strict.

3. **Integrable** with other C++ libraries - multimedia programming, for instance, is an increasingly more common application of GA and the ability to include other templated libraries in one project enables the algebra to be used in other contexts, such as audio programming.

4. **Translatable** - using Clang and LLVM, it is possible to leverage the universality of C-like languages to compile the library to used as an extension in any other language.

5. **Compilable** - Restriction of other standard template libraries enables fine-tuning of tuple-like classes for domain specific compilation

We face two major disadvantages that we hope to work towards eliminating or minimizing, but that are common to generic programming with C++.

1. Template generation in C++ is still slow relative to interpreted languages like lua. While we can significantly decrease compile times by creating a library of pre-instantiated types, the template-depth limits to recursion remain obstacles when compiling algebras for dimensions higher than 10.

2. Template programming techniques employed here require experts to parse and contribute – the guts of the code is not inherently clean or as readable as a DSL might be.

A.4 An Example: Generating $\mathscr{G}^3$

Before examining the inside of our template library, let us start with an example to show how one might instantiate $\mathscr{G}^3$ using the `Versor` library.

$\mathscr{G}^3$ is the algebra of Euclidean $\mathbb{R}^3$, the common space of vectors and quaternions with which most programmers are familiar. To instantiate a 3-dimensional vector, we use the `NEVec<>` class, which is an N-dimensional Euclidean vector templated on N, the dimension:

```
typedef NEVec<3> Vec3;

Vec3 a(1,2,3);
Vec3 b(4,5,6);

auto ip = a <= b;
auto op = a ^ b;
auto gp = a * b;

ip.print(); op.print(); gp.print();
```

which prints out the inner product:

```
blades:
s 000

values:
32.000000
```

where `s` is a scalar value and `000` delineates the 0-grade of the result. Next is printed the outer product:

```
blades:
e12 011
e13 101
e23 110

values:
-3.000000 -6.000000 -3.000000
```

where the bit lists here show us which dimensions are present in each blade. Finally we fetch the geometric product (a combination of inner and outer products):

```
blades:
s 000
e12 011
e13 101
e23 110

values:
32.000000 -3.000000 -6.000000 -3.000000
```

The final type is that of a rotor, or quaternion. While the actual instructions for calculating these results are generated at compile time, we can of course still print them out. For instance, the comma-separated instructions for generating the Euclidean outer product of two 3D vectors can be fetched:

```
EOProd< Vec3, Vec3>::DO().print();
```

which prints out:

```
a[0] * b[1] /*e12*/ -a[1] * b[0] /*e12*/,
a[0] * b[2] /*e13*/ -a[2] * b[0] /*e13*/,
a[1] * b[2] /*e23*/ -a[2] * b[1] /*e23*/
```

which shows us how the outer product is computed by indexing the data of two inputs a and b. Whereas in typical algebraic code generating applications, these instructions are printed into headers which are then compiled, our method generates them as type-specific execution lists at compile-time.

Of course we are not restricted to 3 dimensions. For instance we can print the instructions generated by the compiler for calculating the outer product of two 4-dimensional Vectors:

```
EOProd< NEVec<4>, NEVec<4> >::DO().print();
```

which prints out:

```
a[0] * b[1] /*e12*/ -a[1] * b[0] /*e12*/,
a[0] * b[2] /*e13*/ -a[2] * b[0] /*e13*/,
a[1] * b[2] /*e23*/ -a[2] * b[1] /*e23*/,
a[0] * b[3] /*e14*/ -a[3] * b[0] /*e14*/,
a[1] * b[3] /*e24*/ -a[3] * b[1] /*e24*/,
a[2] * b[3] /*e34*/ -a[3] * b[2] /*e34*/
```

blade	**bits**	**grade**
α	`000`	0
e_1	`001`	1
e_2	`010`	1
e_3	`100`	1
e_{12}	`011`	2
e_{13}	`101`	2
e_{23}	`110`	2
e_{123}	`111`	3

Table A.1: Binary Representation of Basis Blades in $\mathscr{G}^3$

A.5 Blade Basics and Constant Expressions

Our implementation integrates three basic techniques:

1. Short integers are used to represent blades, following the convention outlined in [45] and described in table A.1.
2. Constant expressions are used for manipulating the combinations of particular blades.
3. Variadic templates are used for defining Multivectors

Table A.1 depicts the blade representations for $\mathscr{G}^3$ outlined in [45]. The number of "on" bits determines the grade of the blade and the position of the bits encodes the particular bases present.

Since we do not plan to go above 16 dimensional representations, in our implementation we typedef this as a short integer:

```
typedef short bit_type;
```

Using this bit representation, Dorst, Mann, and Fontijne have shown that the geometric product between two blades can be calculated as an *xor* operation, along with some bit twiddling to determine whether a negation emerges as well as a result of anti-commutativity. Similar bit operations enable calculation of whether inner or outer products should "count", that is, whether they result in a new blade or 0. Their simple methods for determining these true or false statements are written below in algorithms A.2,A.3,A.4, and A.5.

Since most of this algorithms require knowing the grade of the blade, we first examine a recursive algorithm calculates the number of "on" bits at compile time:

Algorithm A.1 Get Grade of Blade *a*

```
function GRADE($a$, $c = 0$)
    if $a > 0$ then
        if $a\&1$ then
            return GRADE($a \gg 1$, $c + 1$)                ▷ $\gg$:bitshift right
        else
            return GRADE($a \gg 1$,$c$)
    else
        return $c$
end function
```

In C++11 compile-time constant expressions we code this using ternary operators:

```
constexpr short grade (short a, short c = 0){
  return a>0 ? (
    a&1 ?
      grade( a>>1, c+1 ) : grade( a>>1, c )
      )
      : c;
 }
```

In pseudocode we can write the algorithms of Dorst et al, such as the xor product and the signflip.

Algorithm A.2 Geometric Product of Basis Blades

```
function PRODUCT(A, B)
    $r \leftarrow A \oplus B$                               ▷ $\oplus$: bitwise XOR
    return $r$
end function
```

Algorithm A.3 Check for Sign Flip of Geometric Product of *a* and *b*

```
function SIGN($a$,$b$,$c$)
    if $(a \gg 1) > 0$ then
        return SIGN($a \gg 1$, $b$, $c$+GRADE($a \gg 1 \& b$))
    else
        if $c \& 1$ then
            return true
        else
            return false
end function
```

The INNER() and OUTER() functions check whether those products should be calculated.

Algorithm A.4 Check for Inner Product (Left Contraction)

```
function INNER($a$,$b$)
    $ga \leftarrow$ GRADE($a$), $gb \leftarrow$ GRADE($b$), $gab \leftarrow$ GRADE($a \wedge b$)
    if $ga > gb$ or $gab \neq gb - ga$ then
        return true
    else
        return false
end function
```

Algorithm A.5 Check for Outer Product

```
function OUTER($a$,$b$)
    return not $a \& b$
end function
```

We use constant expressions to evaluate at compile time whether or not the inner or outer products should be calculated:

```
constexpr bool inner(short a, short b) {
  return !(
    (grade(a) > grade(b)) ||
    (grade(a ^ b) != (grade(b) - grade(a))) );
}
```

```
constexpr bool outer(short a, short b){
  return !( a & b );
}
```

and use a similar method for programming the sign flip check.

Algorithm A.6 Order Comparison

```
function COMPARE($a$,$b$)
    $ga \leftarrow$ GRADE($a$)
    $gb \leftarrow$ GRADE($b$)
    if $a == b$ then
        return $a < b$
    else
        return $ga < gb$
end function
```

A.6 Multivectors and Variadic Templates

We define a Multivector as a templated list of shorts. First we define the generic empty set:

```
template<short ... XS>
struct MV{
  static const int Num = 0;
};
```

where the "`...`" represents a **variadic** template of variable arguments. Note that it is somewhat counterintuitive that the empty set is the generic template. The specialized template, below, is instantiated when there is at least one basis.

```
typedef float VT;

template<TT X, TT ... XS>
struct MV<X, XS...>{

   static const TT HEAD = X;
   typedef MV<XS...> TAIL;

   static const int Num = sizeof...(XS) + 1;
   VT val[Num];

   template<typename ... Args>
   constexpr explicit MV(Args...v) :
   val{ static_cast<VT>(v)...} {}
}
```

Thus when we call `MV<>()` we instantiate an empty set and when we call `MV<1,2,4>()` we instantiate a 3D vector with binary basis: `001,010,100`. Note the similarity to functional programming languages like Haskell or O'Caml where one works with a "HEAD" (in our case `x` and a "TAIL" (in our case `...XS`). The TAIL is recursively instantiates subtypes until `XS...` is empty, at which point the empty set is returned.

We can combine two such lists together using a concatenation operation called `Cat`.

```
template<class A, class B>
struct Cat{
  typedef MV<> Type;
};

template<TT ... XS, TT ... YS>
struct Cat< MV<XS...>, MV<YS...> > {
  typedef MV<XS..., YS...> Type;
};
```

The `Cat` template leverages the simplicity of variadic templates to manage type construction. Our approach adopts a functional style for managing more complex type creation that emerges through the compile-time evaluation of basic basis operations. Each operation on our multivector (`MV`) type is a templated struct with a limiting special case. Critical to this is the employment of a `Either` type constructor, which defines a type A if the first templated parameter is true, and otherwise returns type B.

```
template<bool, class A, class B>
struct Either{
  typedef A Type;
};

template<class A, class B>
struct Either<false,A,B>{
  typedef B Type;
};
```

As an example of the use of the COMPARE constant expression function, the CAT type constructor, and the EITHER type constructor, let us consider a procedure that inserts a basis blade (represented by a short) into a multivector (represented by a variadic template of shorts). It is a functional sorting procedure.

Algorithm A.7 Insert-Sort

```
procedure INSERT($a$, R, F = MV<>)
    $isLessThan \leftarrow$ COMPARE($a$, R::HEAD)
    $catF \leftarrow$ CAT(CAT($F$,MV<$a$>), $R$)
    $catR \leftarrow$ CAT($F$,MV<R::HEAD>)
    return $Type \leftarrow$ EITHER($isLessThan$, $catF$, INSERT($a$, R::TAIL, $catR$)
end procedure
```

We begin with a basis blade *a*, a multivector list of blades `R`, and an empty set `F`. We use our COMPARE function to evaluate whether *a* should be considered "less than" or "greater than" the first basis of the multivector, `R::HEAD`. Two options are available depending on this compile-time compare: we can either pop the blade in the front, or we can iterate the function, considering this time the tail of `R` and setting the head as `F` for future use. We keep track of this head as it increases with each iteration. Note that we will need to add another conditional branch to check to see if it is equal to `R::HEAD`. If it is equal, that means it already exists in our list *R* and so we do not insert *a*. This is accomplished during the REDUCE method discussed below.

```
template <int A, class Rest, class First=MV<> >
struct Insert{
  typedef typename Either<

    compare<A, Rest::HEAD>(),

    typename Cat<
      typename Cat<
        First,
        MV<A>
      >::Type,
      Rest
    >::Type,

    typename Insert<
       A,
       typename Rest::TAIL,
       typename Cat<
          First,
          MV<Rest::HEAD>
       >::Type
    >::Type

  >::Type Type;
};

template <int A, class First>
struct Insert<A, MV<>, First>{
  typedef typename Cat< First, MV<A> >::Type Type;
};
```

Because C++ template instantiations are eager and slow, this insert-and-sort technique is a rather naive implementation. In our actual implementation we use additional branching to provide opportunities to exit the recursive loop early. For instance, the INSERT metafunction calls an INSERTIMPL metafunction which has different instantiations depending on whether the blade already exists in the list to which it is being added.

A.7 Geometric Products and Instruction Lists

As we have discussed, geometric algebra defines three basic binary operators between any two multivectors A and B: the inner, outer, and geometric products. We consider first the geometric product.

The geometric product between two multivectors is a distributed multiplication: for each blade a_i in A, calculate its product with each blade b_i in B. As we have seen, we represent the new blades that emerge from this multiplication with the xor operator, and we assign a weight to it that is simply the product of the two blades - negating the weight if the anti-commutative property of the algebra requires us to do so.

We must generate a list of instructions at compile-time which can then be executed to evaluate the weighted basis of a new multivector at compile-time (or run-time). That is, the return type and combinatoric rules for how the basis blades combine must be evaluated at compile-time, but the actual weights can be evaluated at run-time.

To store this information at compile-time, we define an `Inst` struct that can be used to calculate one element. It is fed a boolean value used to determine whether or not to negate the product, two basis blades, and two indices. Most importantly, it stores the result of the xor product of two input basis blades, a boolean for whether or not the inner and outer products should be evaluated. Its `Exec()` method returns the product of two indices of two particular multivector arguments.

```
template<bool F, short A, short B, int IDXA, int IDXB>
struct Inst{
  static const short Res = A ^ B;
  static const bool IP = inner(A,B);
  static const bool OP = outer(A,B);
  static const int idxA = IDXA;
  static const int idxB = IDXB;

  template<class TA, class TB>
  static constexpr
  VT Exec( const TA& a, const TB& b){
    return a[idxA] * b[idxB];
  }
};
```

A slightly different specialized version of the struct is instantiated when the SIGNFLIP check returns true:

```
template<bool F, short A, short B, int IDXA, int IDXB>
struct Inst{
  ...//as above

  template<class TA, class TB>
  static constexpr
  VT Exec( const TA& a, const TB& b){
    return -a[idxA] * b[idxB];
  }
};
```

These instructions are generated when two multivectors are multiplied together. The algorithm below demonstrates this process, whereby a main method, GP, calls a subroutine, SUBGP, inorder to recursively iterate through two lists distributively.

Algorithm A.8 Building Instructions

```
procedure SubGP($a$,B)
    if B=$\emptyset$ then
        return $\emptyset$
    else
        $inst \leftarrow$ INST($a$, B::HEAD)
        return CAT($inst$, SubGP($a$, B::TAIL) )
end procedure
procedure GP(A,B)
    if A=$\emptyset$ then
        return $\emptyset$
    else
        return CAT( SubGP(A::HEAD,B), GP(A::TAIL,B )
end procedure
```

Inner and Outer product instructions are similarly built, with the exception that we use the EITHER type constructor to only add in blades that should count towards the result. No such check is necessary with the geometric product.

In order to save compile-time, we use the calculations contained in the instruction structure in order to determine return type of two multivectors. The product of two blades is contained in the `Res` static member, and that is what we want to gather, reduce, and sort.

With our functional approach we first define a recursive template for determining whether a blade exists in a multivector.

Algorithm A.9 Check for Existence of Blade in a Multivector

```
procedure EXISTS($a$,M)
    if M $= \emptyset$ then return  false
    else
        if $a$ ==M::HEAD then
            return  true
        else
            return EXISTS($a$, M::TAIL)
end procedure
```

```
template<short N, class M>
struct Exists{
  static constexpr bool Call() {
    return M::HEAD == N ?
      true :
      Exists<N, typename M::TAIL>::Call();
  }
};

template<TT N>
struct Exists< N, MV<> >{
  static constexpr bool Call() { return false; }
};
```

To determine the return type from a list of instructions, we reduce the output to include only the *different* types. We employ another technique from functional programming, wherein we start at the end with an empty set, and recursively add in non-existing blades starting from from the tail of the instruction list. Thus we define the end case first, recursively adding in the tail if whether the last element is a member.

Algorithm A.10 Multivector Return Type from an Instruction List

```
procedure REDUCE(I)
    $M \leftarrow$ REDUCE(I::TAIL)                                ▷ Recursion
    $A \leftarrow$ INSERT(I::HEAD, M)
    $Type \leftarrow$ EITHER( EXISTS(I::HEAD::Res, M), M, A)
    return $Type$
end procedure
```

```
template<class X>
struct Reduce{
  typedef typename Reduce<typename X::TAIL>::Type M;
  typedef typename Maybe<
    Exists< X::HEAD::Res, M>::Call() ,
    M,
    typename Insert<
      X::HEAD::Res, M
    >::Type
  >::Type Type;
};

template<>
struct Reduce<XList<> >{
  typedef MV<> Type;
};
```

We have not explained how we have been storing these instructions as they were generated for the GP procedure: we define another type of variadic template, an `XList<typename ... XS>` which is like a `std::tuple` except that it includes methods for executing instructions in the template parameter packs, and instantiating types using those instructions.

Similar to our `MV` type, we first define the empty set:

```
template< typename ... XS >
struct XList{

   template<class A, class B>
   static constexpr
   VT Exec(const A& a, const B& b){
    return 0;
   }

  template<class R, class A, class B>
  static constexpr MV<> Make(const A& a, const B& b){
    return MV<>();
  }

};
```

and then specialize the template for when there is at least one template parameter:

```
template< typename X, typename ... XS >
struct XList<X,XS...>{

  typedef X HEAD;
  typedef XList<XS...> TAIL;

  template<class A, class B>
  static constexpr
  VT Exec(const A& a, const B& b){
    return X::Exec(a,b) + TAIL::Exec(a,b);
  }

  template<class R, class A, class B>
  static constexpr R Make(const A& a, const B& b){
    return R(X::Exec(a,b), XS::Exec(a,b)...);
  }
};
```

Upon instantiating an operation, a geometric product is calculated by first generating an individual instruction for each product of each basis of multivector *A* with each basis of multivector *B*. We then sort these instructions by return type and create a list of instructions for each blade in the return multivector. We gather these lists into one list, and the result is a *list of lists* of instructions: when two multivectors are multiplied together, the instruction list executes each of its children instruction lists, which in turn sum each of its children instructions together to form one computation for one blade.

To accomplish this organization, we need to sort the instructions into each element in the return type that it will contribute to. For each element in the return type, we build a list of instructions.

Algorithm A.11 Sorting Instructions

```
procedure FIND(()$a$,A)
    if A=$\emptyset$ then
        return $\emptyset$
    else
        next ← FIND($a$,A::TAIL)
        cat ← CAT(A::HEAD, next)
        list ← EITHER($a$ ==A::HEAD::Res, cat, next )
end procedure
procedure INDEX(A,B)
    if B=$\emptyset$ then
        return $\emptyset$
    else
        one ← FIND(B::HEAD,A)
        list ← CAT(one, INDEX(A,B::TAIL) )
end procedure
```

```
template< int N, class A >
struct FindAll {
   typedef typename
   FindAll< N, typename A::TAIL>::Type Next;

   typedef typename
   Either< A::HEAD::Res == N,
      typename XCat<
        XList< typename A::HEAD >,
        Next
      >::Type,
      Next
   >::Type Type;
};

template< int N >
struct FindAll< N, XList<> >{
  typedef XList<> Type;
};


template< class I, class R >
struct Index{

  typedef typename FindAll<R::HEAD, I>::Type One;
  typedef typename
  XCat<
    XList< One > ,
    typename Index < I, typename R::TAIL >::Type
    >::Type Type;

};
template< class I>
struct Index< I, MV<> > {
   typedef XList<> Type;
};
```

Finally, we are able to execute our products using the sorted list of instructions for doing so.

Algorithm A.12 Executing Products

```
procedure MAKEGP(A,B)
    inst ← INST(A,B)
    R ← REDUCE(inst)
    list ← INDEX(inst,R)
    return list(A,B)
end procedure
```

A.8 Discussion

Geometric algebra, a hypercomplex algebra of exponential increase in types, is a prime candidate for the exploration of C++11 tools such as variadic templates and parameter unpacking, which provide the developer with a novel tool for implementing efficient combinatoric instructions at compile-time. We have outlined here the process for pure type construction of Geometric algebra multivectors. Changing or splitting the underlying metric, as is necessary in the conformal model [45] is also present in the online code base. This code has been used to model the problems presented in this dissertation.

We hope that by providing the programming community with a lightweight templated header-only library, geometric algebra will be adopted by a larger group of people. Likewise, we hope that by introducing the geometric algebra community to the prowess of C++11 template metaprogramming, we can begin to search for more single-language solutions to the complex problem of programming hypercomplexity. It is also expected that colleagues will find ways for making the cpu code more efficient, or make use of other C++ idioms than we have explored here.

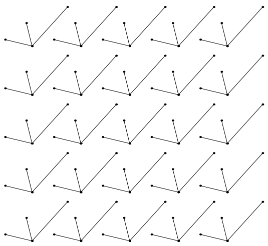
If implemented generically, geometric algebra can narrow the gap between the conceptual model of space and its digital encoding on a computer. There is a subtle friction between the structures represented by a computer (a circle for instance) and the structure of the representations themselves (three coordinates, an orientation and a radius), and any formal investigation into the *structure of a mathematics of structure* is a chance to find their fault-line. A good model of space preserves structures across this divide, allowing experiments on a computer to inform our geometric intuition.

Appendix B

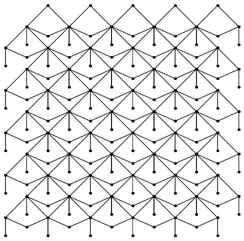
2D Crystallographic Space Groups

Oblique

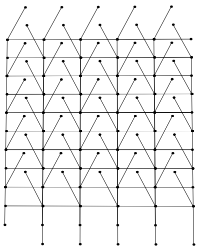
Rectangular



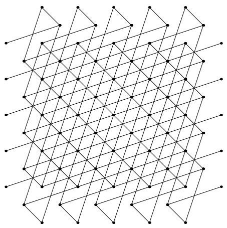
p1



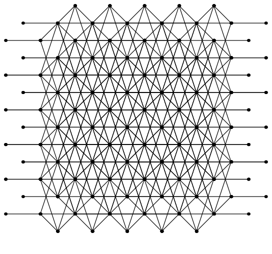
cm



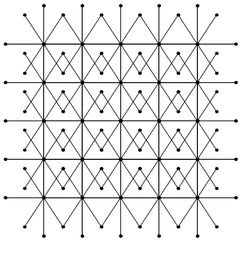
pg



p2

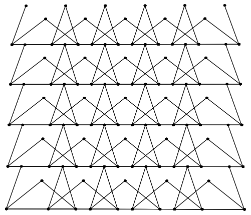


cmm

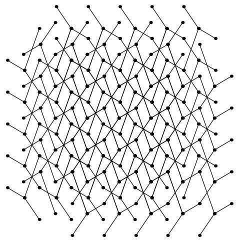


pmm

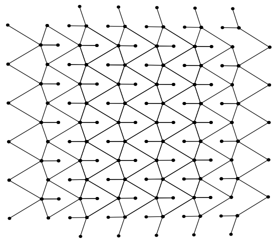
Rectangular



pm

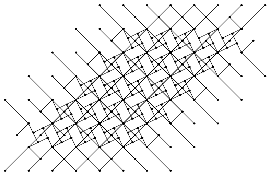


pgg

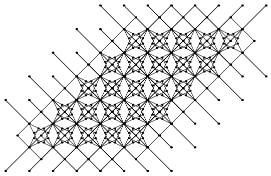


pmg

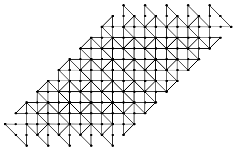
Square



p4

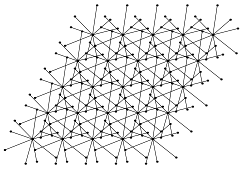


p4m

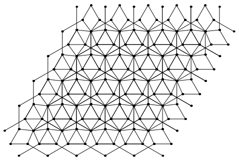


p4g

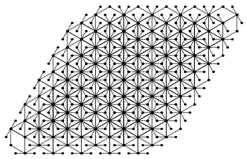
Trigonal



p3

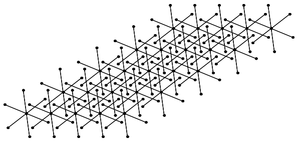


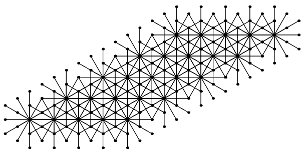
p3m1



p31m

Hexagonal





p6	p6m

Bibliography

[1] Joan Lasenby Andreas Aristidou. Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver. Technical Report CUED/F-INFENG/TR-632, Cambridge University Engineering Department, 2009.

[2] G. Aragon-Camarasa, G. Aragon-Gonzalez, J. L. Aragon, and M. A. Rodriguez-Andrade. Clifford algebra with Mathematica, 2008. Accessed online at http://arxiv.org/abs/0810.2412.

[3] Andreas Aristidou, Yiorgos Chrysanthou, and Joan Lasenby. Extending FABRIK with model constraints. *Computer Animation and Virtual Worlds*, 2015.

[4] Andreas Aristidou and Joan Lasenby. FABRIK: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73(5):243–260, 2011.

[5] J. Eddie Baker. An analysis of the bricard linkages. *Mechanism and Machine Theory*, 15(4):267–286, 1980.

[6] Eduardo Bayro-Corrochano. Robot perception and action using conformal geometric algebra. In *Handbook of Geometric Computing*, pages 405–458. Springer Berlin Heidelberg, 2005.

[7] Eduardo Bayro-Corrochano. *Geometric Computing for Wavelet Transforms, Robot Vision, Learning, Control and Action*. Springer-Verlag, 2010.

[8] Eduardo Bayro-Corrochano and Luis Eduardo Falcón. Geometric algebra of points, lines, planes and spheres for computer vision and robotics. *Robotica*, 23(06):755–770, 11 2005.

[9] Eduardo Bayro-Corrochano and Detlef Kähler. Motor algebra approach for computing the kinematics of robot manipulators. *Journal of Robotic Systems*, 17(9):495 – 516, 2000.

[10] Eduardo Bayro-Corrochano and Detlef Kähler. Kinematics of robot manipulators in the motor algebra. In Gerald Sommer, editor, *Geometric Computing with Clifford Algebras*, pages 471–488. Springer Berlin Heidelberg, 2001.

[11] Eduardo Bayro-Corrochano and Joan Lasenby. Object modelling and motion analysis using clifford algebra. In *Proceedings of Europe-China Workshop on Geometric Modeling and Invariants for Computer Vision, Ed. Roger Mohr and Wu Chengke*, pages 143–149, 1995.

[12] Ian Bell. Multivector methods website, Accessed 2015. `http://www.iancgbell.clara.net/maths/geodes.htm#A1`.

[13] M.C. Lopez Belon. Applications of conformal geometric algebra in mesh deformation. *Brazilian Symposium of Computer Graphic and Image Processing*, pages 39–46, 2013.

[14] G. T. Bennett. A new (four-piece skew) mechanism. *Engineering*, 76:777–778, 1903.

[15] M. Beynon and A. Cartwright. Intelligent CAD systems II: Implementational issues. chapter A Definitive Programming Approach to the Implementation of CAD Software, pages 126–145. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[16] Pengbo Bo, Helmut Pottmann, Martin Kilian, Wenping Wang, and Johannes Wallner. Circular arc structures. *ACM Trans. Graph.*, 30(4):101:1–101:12, July 2011.

[17] Alexander I. Bobenko and Emanuel Huhnen-Venedey. Curvature line parametrized surfaces and orthogonal coordinate systems: discretization with dupin cyclides. *Geometriae Dedicata*, 159(1):207–237, 2012.

[18] Alexander I Bobenko and Yury B Suris. On organizing principles of discrete differential geometry. Geometry of spheres. *Russian Mathematical Surveys*, 62(1):1, 2007.

[19] Alan Bromborsky. Sympy geometric algebra module homepage, 2013. Accessed online at `http://docs.sympy.org/0.7.0/modules/galgebra/GA/GAsympy.html`.

[20] C.R. Calladine. Deployable structures: What can we learn from biological structures? In S. Pellegrino and S.D. Guest, editors, *IUTAM-IASS Symposium on Deployable Structures: Theory and Applications*, volume 80 of *Solid Mechanics and Its Applications*, pages 63–76. Springer Netherlands, 2000.

[21] Georges Canguilhem. *Ideology and rationality in the history of the life sciences*. MIT Press, Cambridge, Mass., 1988.

[22] Yan Chen and Zhong You. Spatial overconstrained linkages – the lost jade. In Teun Koetsier and Marco Ceccarelli, editors, *Explorations in the History of Machines and Mechanisms*, volume 15 of *History of Mechanism and Machine Science*, pages 535–550. Springer Netherlands, 2012.

[23] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):pp. 345–363, 1936.

[24] William Clifford. Applications of Grassmann's extensive algebra. *American Journal of Mathematics (The Johns Hopkins University press)*, 1(4):350–358, 1878.

[25] William Kingdon Clifford. *Seeing and thinking*. Nature series. Macmillan and co., London,, 1879.

[26] William Kingdon Clifford. *The common sense of the exact sciences*. D. Appleton and company, New York,, 1885.

[27] Pablo Colapinto. Versor: Spatial computing with conformal geometric algebra. Master's thesis, University of California at Santa Barbara, 2011. Available at `http://versor.mat.ucsb.edu`.

[28] Pablo Colapinto. Boosted surfaces: Synthesis of meshes using point pair generators as curvature operators in the 3d conformal model. *Advances in Applied Clifford Algebras*, 24(1):71–88, 2014.

[29] Pablo Colapinto. Composing surfaces with conformal rotors. In *Proceedings of Applications of Geometric Algebra in Computer Science and Engineering*, pages 109–119, Barcelona, 2015.

[30] Louis Couturat. *The Logic of Leibniz*. Translated by Donald Rutherford and R. Timothy Monroe,, 1901 / 2012. Accessed online at http://philosophyfaculty.ucsd.edu/faculty/rutherford/Leibniz/couturatcontents.php.

[31] Keenan Crane, Ulrich Pinkall, and Peter Schröder. Spin transformations of discrete surfaces. *ACM Trans. Graph.*, 30(4):104, 2011.

[32] Keenan Crane, Ulrich Pinkall, and Peter Schröder. Robust fairing via conformal curvature flow. *ACM Trans. Graph.*, 32(4):61, 2013.

[33] Francois Dagognet. *Methodes et doctrine dans l'oeuvre de Pasteur*. Presses universitaires de France, Paris, 1967.

[34] Pierre-Philippe Dechant. A clifford algebraic framework for coxeter group theoretic computations. *Advances in Applied Clifford Algebras*, 24(1):89–108, 2014.

[35] Erik D. Demaine. Folding and unfolding. In *in Computational Geometry. 2004. Monograph in preparation*, pages 167–211. Cambridge University Press, 2001.

[36] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans. ASME E, Journal of Applied Mechanics*, 22:215–221, June 1955.

[37] Francois Denis. *Traité de Lutherie: The Violin and the Art of Measurement*. Aladfi, 2006.

[38] Chris Doran. Circle and sphere blending with conformal geometric algebra, 2003. Accessed online at http://arxiv.org/abs/cs/0310017.

[39] Chris Doran and Anthony Lasenby. *Geometric algebra for physicists*. Cambridge University Press, Cambridge; New York, 2007.

[40] Leo Dorst. Tutorial appendix: Structure preserving representation of Euclidean motions through conformal geometric algebra. In Leo Dorst and Joan Lasenby, editors, *Guide to Geometric Algebra in Practice*, pages 435–453. Springer, 2011.

[41] Leo Dorst. Total least squares fitting of k-spheres in n-d euclidean space using an (n+2)-d isometric representation. *Journal of Mathematical Imaging and Vision*, 50(3):214–234, 2014.

[42] Leo Dorst. 3d oriented projective geometry through versors of $\mathbb{R}^{3,3}$. *Advances in Applied Clifford Algebras*, pages 1–36, 2015.

[43] Leo Dorst. The construction of 3d conformal motions. *Accepted to Mathematics in Computer Science, Special Issue on Geometric Computation*, 2016.

[44] Leo Dorst and Daniel Fontijne. 3d Euclidean geometry through conformal geometric algebra (a GAViewer tutorial). 2005. Accessed online at http://www.science.uva.nl/research/ias/ga/.

[45] Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric algebra for computer science an object-oriented approach to geometry*. Elsevier ; Morgan Kaufmann, Amsterdam; San Francisco, 2007.

[46] Leo Dorst and Robert Valkenburg. Square root and logarithm of rotors in 3d conformal geometric algebra using polar decomposition. In Leo Dorst and Joan Lasenby, editors, *Guide to Geometric Algebra in Practice*, pages 81–104. Springer, 2011.

[47] Lucie Druoton, Laurent Fuchs, Lionel Garnier, and Rémi Langevin. The non-degenerate Dupin cyclides in the space of spheres using geometric algebra. *Advances in Applied Clifford Algebras*, 24(2):515–532, 2014.

[48] Charles Dupin. *Applications de géométrie de mécanique à la marine, aux ponts et chaussées; pour faire suite aux développemens de géométrie*. Paris, 1822.

[49] Luis Falcon-Morales and Eduardo Bayro-Corrochano. 3d-curves and ruled surfaces for graphics and robotics using conformal geometric computing. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2627–2632, May 2006.

[50] Daniel Fontijne. GA sandbox homepage, 2006. Accessed online at `http://sourceforge.net/projects/gasandbox/`.

[51] Daniel Fontijne, Leo Dorst, and Tim Bouma. The gaigen home page, 2001. Accessed online at `http://www.science.uva.nl/research/ias/ga/gaigen/content_gaigen.html`.

[52] Sebti Foufou and Lionel Garnier. Dupin cyclide blends between quadric surfaces for shape modeling. *Comput. Graph. Forum*, 23(3):321–330, 2004.

[53] Silvia Franchini, Antonio Gentile, Filippo Sorbello, Giorgio Vassallo, and Salvatore Vitabile. An embedded, FPGA-based computer graphics coprocessor with native geometric algebra support. *Integration, the {VLSI} Journal*, 42(3):346 – 355, 2009. Special Section on {DCIS2006}.

[54] Laurent Fuchs and Laurent Thery. Implementing geometric algebra products with binary trees. *Advances in Applied Clifford Algebras*, 24(2):589–611, 2014.

[55] Buckminster Fuller. *Synergetics : explorations in the geometry of thinking*. Macmillan, New York, 1975.

[56] Wei W. Gan and Sergio Pellegrino. Closed-loop deployable structures. *In Proc. 44th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, April 2003.

[57] Arran Gare. Overcoming the Newtonian paradigm: the unfinished project of theoretical biology from a schellingian perspective. *Prog Biophys Mol Biol*, 113(1):5–24, September 2013.

[58] Etienne Ghys and Jos Leys. Lorenz and modular flows: a visual introduction. `http://www.josleys.com/articles/ams_article/Lorenz3.htm`, Online, Accessed October 2015.

[59] Hermann Grassmann. *Die lineale Ausdehnungslehre ein neuer Zweig der Mathematik, dargestellt und durch Anwendungen auf die übrigen Zweige der Mathematik, wie auch auf die Statik, Mechanik, die Lehre vom Magnetismus und die Krystallonomie erläutert*. O. Wigand, Leipzig, 1844.

[60] Hermann Grassmann. *Die Ausdehnungslehre*. Enslin, Berlin, 1862.

[61] Simon D. Guest and Sergio Pellegrino. The folding of triangulated cylinders, part I: Geometric considerations. *Journal of Applied Mechanics-transactions of The Asme - J APPL MECH*, 61(4), 1994.

[62] Stephen Gull, Anthony Lasenby, and Chris Doran. Imaginary numbers are not real - the geometric algebra of spacetime. *Foundations of Physics*, 23(9):1175–1201, 1993.

[63] Timothy F. Havel and Igor Najfeld. Applications of geometric algebra to the theory of molecular conformation part 2. The local deformation problem. *Journal of Molecular Structure*, 336(2-3):175 – 189, 1995. Topological aspects of molecular structures.

[64] Peter Henderson. Functional geometry. In *Symposium on LISP and Functional Programming*, pages 179–187, 1982.

[65] David Hestenes. *Space-Time Algebra*. Gordon and Breach, New York, 1966.

[66] David Hestenes. *New foundations for classical mechanics*. Reidel ; Sold and distributed in the U.S.A. and Canada by Kluwer Academic Publishers, Dordrecht; Boston; Hingham, MA, 1986.

[67] David Hestenes. The design of linear algebra and geometry. *Acta Applicandae Mathematica*, 23(1):65–93, 1991.

[68] David Hestenes. Differential forms in geometric calculus. In F. Brackx, R. Delanghe, and H. Serras, editors, *Clifford Algebras and their Applications in Mathematical Physics*, volume 55 of *Fundamental Theories of Physics*, pages 269–285. Springer Netherlands, 1993.

[69] David Hestenes. Geometric calculus, 1998. Available at `http://geocalc.clas.asu.edu/pdf/NFMPchapt2.pdf`.

[70] David Hestenes. *Old Wine in New Bottles: a new algebraic framework for computational geometry*, volume Geometric Algebra with Applications in Science and Engineering, page 16. Birkhäuser, Boston, 2001.

[71] David Hestenes. Point groups and space groups in geometric algebra. In Leo Dorst, Chris Doran, and Joan Lasenby, editors, *Applications of Geometric Algebra in Computer Science and Engineering*, pages 3–34. Birkhäuser Boston, 2002.

[72] David Hestenes. The shape of differential geometry in geometric calculus. In Leo Dorst and Joan Lasenby, editors, *Guide to Geometric Algebra in Practice*, pages 393–410. Springer, 2011.

[73] David Hestenes and Jeremy W. Holt. Crystallographic space groups in geometric algebra. *Journal of Mathematical Physics*, 48(2):–, 2007.

[74] David Hestenes, Hongbo Li, and Alyn Rockwood. New algebraic tools for classical geometry. In G. Sommer, editor, *Geometric Computing with Clifford Algebras*, chapter New Algebraic Tools for Classical Geometry, pages 3–26. Springer-Verlag, London, UK, UK, 2001.

[75] David Hestenes and Garret Sobczyk. *Clifford algebra to geometric calculus : a unified language for mathematics and physics*. D. Reidel ; Distributed in the U.S.A. and Canada by Kluwer Academic Publishers, Dordrecht; Boston; Hingham, MA, U.S.A., 1984.

[76] Dietmar Hildenbrand, Daniel Fontijne, Christian Perwass, and Leo Dorst. Geometric algebra and its application to computer graphics. In *Eurographics conference, Grenoble*, 2004.

[77] Dietmar Hildenbrand, Joachim Pitt, and Andreas Koch. Gaalop - high performance parallel computing based on conformal geometric algebra. In Eduardo Bayro-Corrochano and Gerik Scheuermann, editors, *Geometric Algebra Computing*, pages 477–494. Springer, 2010.

[78] Eckhard Hitzer. New views of crystal symmetry guided by profound admiration of the extraordinary works of Grassmann and Clifford. In Hans-Joachim Petsche, Albert C. Lewis, Jörg Liesen, and Steve Russ, editors, *From Past to Future: Grassmann's Work in Context*, pages 413–422. Springer Basel, 2011.

[79] Eckhard Hitzer and Christian Perwass. Interactive 3d space group visualization with clucalc and the clifford geometric algebra description of space groups. *Advances in Applied Clifford Algebras*, 20(3):631–658, 2010.

[80] Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson. Arrows, robots, and functional reactive programming. In Johan Jeuring and Simon L. Peyton Jones, editors, *Advanced Functional Programming*, volume 2638 of *Lecture Notes in Computer Science*, pages 159–187. Springer Berlin Heidelberg, 2003.

[81] F. M. Jaeger. *Lectures on the principle of symmetry and its applications in all natural sciences*. Elsevier, Amsterdam, 2d, augmented ed., with 173 diagrams and 3 portraits. edition, 1920.

[82] Wassily Kandinsky and Hilla Rebay. *Point and line to plane*. Dover Publications, New York, 1979.

[83] Paul Klee. *Pedagogical sketchbook*. Faber and Faber, London; Boston, 1968.

[84] Felix C. Klein. A comparative review of recent researches in geometry, 1872. English Translation accessed online at http://arxiv.org/abs/0807.3161.

[85] Hidetoshi Kobayashi, Biruta Kresling, and Julian F.V. Vincent. The geometry of unfolding tree leaves. *Proceedings: Biological Sciences*, 265(1391):pp. 147–154, 1998.

[86] Rimas Krasauskas. Unifying theory of pythagorean-normal surfaces. In *Proceedings of the conference on Applied Geometric Algebra in Computer Science and Engineering*, page 205, Barcelona, 2015.

[87] Rimvydas Krasauskas and Severinas Zube. Bézier-like parametrizations of spheres and cyclides using geometric algebra. In K Guerlebeck, editor, *Proceedings of 9th International Conference on Clifford Algebras and their Applications in Mathematical Physics*, Weimar, Germany, 2011.

[88] Anthony Lasenby. Conformal geometry and the universe. Accessed online www.mrao.cam.ac.uk/ anthony/anl.pdf, 2003.

[89] Anthony Lasenby, Chris Doran, and Stephen Gull. Gravity, gauge theories and geometric algebra. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 356(1737):487–582, 1998.

[90] Anthony N. Lasenby. Recent applications of conformal geometric algebra. In Hongbo Li, Peter J. Olver, and Gerald Sommer, editors, *IWMM/GIAE*, volume 3519 of *Lecture Notes in Computer Science*, pages 298–328. Springer, 2004.

[91] Joan Lasenby, Sahan Gamage, and Maurice Ringer. Modelling motion: Tracking, analysis and inverse kinematics. In Gerald Sommer and YehoshuaY. Zeevi, editors, *Algebraic Frames for the Perception-Action Cycle*, volume 1888 of *Lecture Notes in Computer Science*, pages 104–114. Springer Berlin Heidelberg, 2000.

[92] Joan Lasenby, Anthony Lasenby, and Richard Wareham. A covariant approach to geometry using geometric algebra. Technical Report CUED/F-INFENG/TR-483, 2004.

[93] Hongbo Li. *Invariant Algebras and Geometric Reasoning*. World Scientific, Singapore, 2008.

[94] Hongbo Li, Lei Dong, Changpeng Shao, and Lei Huang. Elements of line geometry with geometric algebra. In *Proceedings of the conference on Applied Geometric Algebra in Computer Science and Engineering*, pages 195–204, 2015.

[95] Hongbo Li, David Hestenes, and Alyn Rockwood. Generalized homogeneous coordinates for computational geometry. In Gerald Sommer, editor, *Geometric Computing with Clifford Algebras*, pages 27–59. Springer Berlin Heidelberg, 2001.

[96] Hongbo Li, Lei Huang, Changpeng Shao, and Lei Dong. Three-dimensional projective geometry with geometric algebra. *CoRR*, abs/1507.06634, 2015.

[97] Yang Liu, Helmut Pottmann, Johannes Wallner, Yong-Liang Yang, and Wenping Wang. Geometric modeling with conical meshes and developable surfaces. *ACM Trans. Graph.*, 25(3):681–689, July 2006.

[98] Yanxi Liu, Hagit Hel-Or, Craig S. Kaplan, and Luc Van Gool. Computational symmetry in computer vision and computer graphics. *Foundations and Trends in Computer Graphics and Vision*, 5(1-2):1–195, 2009.

[99] Ralph Robert Martin. *Principal Patches for Computational Geometry*. PhD thesis, Pembroke College, Cambridge University, 1983.

[100] Michel Mauny. *The functional approach to programming*. Cambridge University Press, Cambridge, U.K.; New York, NY, USA, 1998.

[101] F.A. McRobie and J. Lasenby. The kinematics of large rotations using clifford algebra. In Sergio Pellegrino and Simon D. Guest, editors, *IUTAM-IASS Symposium on Deployable Structures: Theory and Applications*, volume 80 of *Solid Mechanics and Its Applications*, pages 271–280. Springer Netherlands, 2000.

[102] Biswajit Mishra, Mittu Kochery, Peter Wilson, and Reuben Wilcock. A novel signal processing coprocessor for n-dimensional geometric algebra applications. *Circuit. Syst.*, 5:274–279, 2014.

[103] K. Miura and Uchū Kagaku Kenkyūjo (Japan). *Method of Packaging and Deployment of Large Membranes in Space*. Institute of Space and Astronautical Science report. Institute of Space and Astronautical Sciences, 1985.

[104] Parry Hiram Moon and Domina Eberle Spencer. *Field theory handbook : including coordinate systems, differential equations, and their solutions*. Springer-Verlag, Berlin; New York, 1988.

[105] H. Nooshin, P.L. Disney, and O.C. Champion. Formian 2 and a formian function for processing polyhedric configurations. Technical report, NASA Ames Research Center, Moffett Field,CA United States, 1996.

[106] Alberto Paoluzzi, Valerio Pascucci, Michele Vicentino, Claudio Baldazzi, and Simone Portuesi. *Geometric Programming for Computer-Aided Design*. Wiley-Blackwell, West Sussex, England, March 2003.

[107] Josep M. Parra and Llorenc Roselló. General Clifford algebra and related differential geometry calculations with Mathematica. In Rafal Ablamowicz, JosepM. Parra, and Pertti Lounesto, editors, *Clifford Algebras with Numeric and Symbolic Computations*, pages 57–67. Birkhäuser Boston, 1996.

[108] Giuseppe Peano. *Geometric calculus : according to the Ausdehnungslehre of H. Grassmann*. Birkhäuser, Boston, 2000.

[109] Sergio Pellegrino and Julian F.V. Vincent. How to fold a membrane. In Sergio Pellegrino, editor, *Deployable Structures*, volume 412 of *International Centre for Mechanical Sciences*, pages 59–75. Springer Vienna, 2001.

[110] Izzet Pembeci, Henrik Nilsson, and Gregory Hager. Functional reactive robotics: An exercise in principled integration of domain-specific languages. In *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP '02, pages 168–179, New York, NY, USA, 2002. ACM.

[111] Christian Perwass. *Geometric algebra with applications in engineering*. Springer, Berlin, 2009.

[112] Christian Perwass. Clucalc homepage, 2010. Accessed online at `http://www.clucalc.info/`.

[113] Christian Perwass, Christian Gebken, and Gerald Sommer. Implementation of a clifford algebra co-processor design on a field programmable gate array. In Rafal Ablamowicz, editor, *Clifford Algebras*, volume 34 of *Progress in Mathematical Physics*, pages 561–575. Birkhäuser Boston, 2004.

[114] Jack Phillips. *Freedom in machinery*. Cambridge University Press, Cambridge, 2006.

[115] Henri Poincaré. On the foundations of geometry. *The Monist*, 9:1–43, 1898.

[116] Helmut Pottmann, Ling Shi, and Mikhail Skopenkov. Darboux cyclides and webs from circles. *Computer Aided Geometric Design*, 29(1):77 – 97, 2012.

[117] Anthony Pugh. *An Introduction to Tensegrity*. Dome series. University of California Press, 1976.

[118] Ronald D. Resch. The topological design of sculptural and architectural systems. In *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, AFIPS '73, pages 643–650, New York, NY, USA, 1973. ACM.

[119] Ronald D. Resch and Hank Christiansen. The design and analysis of kinematic folded plate systems. *Proceedings of IASS Symposium on Folded Plates and Prismatic Structures*, 1970.

[120] Marcel Riesz. *Clifford numbers and spinors : with Riesz's private lectures to E. Folke Bolinder and a historical review by Pertti Lounesto*. Kluwer Academic, Dordrecht; Boston, 1993.

[121] Bodo Rosenhahn and Reinhard Klette. Geometric algebra for pose estimation and surface morphing in human motion estimation. In Reinhard Klette and Joviša Žunić, editors, *Combinatorial Image Analysis*, volume 3322 of *Lecture Notes in Computer Science*, pages 583–596. Springer Berlin Heidelberg, 2005.

[122] Mark Schenk and Simon D. Guest. Geometry of miura-folded metamaterials. *Proceedings of the National Academy of Sciences*, 110(9):3276–3281, 2013.

[123] Helmut Seibert, Dietmar Hildenbrand, Meike Becker, and et al. *Estimation of curvatures in point sets based on geometric algebra*. 2010.

[124] Florian Seybold. The Gaalet homepage, 2010. Accessed online at `http://sourceforge.net/projects/gaalet/`.

[125] Garret Sobczyk. Simplicial calculus with geometric algebra. In A. Micali, R. Boudet, and J. Helmstetter, editors, *Clifford Algebras and their Applications in Mathematical Physics*, volume 47 of *Fundamental Theories of Physics*, pages 279–292. Springer Netherlands, 1992.

[126] Garret Sobczyk. *New Foundations in Mathematics: The Geometric Concept of Number*. Birkhäuser/Springer, New York, 2013.

[127] Gerald Sommer, Bodo Rosenhahn, and Christian Perwass. The twist representation of free-form objects. In Reinhard Klette, Ryszard Kozera, Lyle Noakes, and Joachim Weickert, editors, *Geometric Properties for Incomplete data*, volume 31, pages 3–22. Springer Netherlands, 2006.

[128] Gerald Jay Sussman and Will Farr. *Functional differential geometry*. MIT Press, Cambridge, Mass., 2013.

[129] Tomohiro Tachi. Generalization of rigid foldable quadrilateral mesh origami. In *Proceedings of International Association for Shell and Spatial Structures Symposium*, Valencia,Spain, 2009.

[130] Tomohiro Tachi. Geometric considerations for the design of rigid origami structures. In *Proceedings of the International Association for Shell and Spatial Structures Symposium*, Shanghai, China, 2010.

[131] Tomohiro Tachi. Freeform origami tessellations by generalizing Resch's patterns. *Journal of Mechanical Design*, 135(11), 2013.

[132] Igor Najfeld Timothy F. Havel. Applications of geometric algebra to the theory of molecular conformation part 1. the optimum alignment problem. *Journal of Molecular Structure*, 308:241–262, 1994.

[133] Bartel Leenert van der Waerden. *A history of algebra : from al-Khwarizmi to Emmy Noether*. Springer-Verlag, Berlin; New York, 1985.

[134] Rich Wareham, Jonathan Cameron, and Joan Lasenby. Applications of conformal geometric algebra in computer vision and graphics. In Hongbo Li, Peter J. Olver, and Gerald Sommer, editors, *IWMM/GIAE*, volume 3519 of *Lecture Notes in Computer Science*, pages 329–349. Springer, 2004.

[135] Richard J. Wareham and Joan Lasenby. Rigid-body pose and position interpolation using geometric algebra. *Submitted to ACM Transactions on Graphics*, 2004.

[136] Hermann Weyl. *Symmetry*. Princeton University Press, Princeton, 1952.

[137] Alfred N. Whitehead. *A Treatise on Universal Algebra: With Applications ..* A Treatise on Universal Algebra: With Applications. The University Press, 1898.

[138] Zhong You and Yan Chen. *Motion structures : deployable structural assemblies of mechanisms*. Spon Press, London; New York, 2012.