Geometric Computing in Computer Graphics and Robotics using Conformal Geometric Algebra

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

von

Dipl.-Inform. Dietmar Hildenbrand

aus Freudenberg/Main

Referenten der Arbeit: *Prof. Dr.-Ing. Marc Alexa*, Technische Universität Berlin
Prof. Dr. techn. Dieter W. Fellner, Technische Universität Darmstadt
Prof. Dr.-Ing. Dr.-Ing. E.h. Wolfgang Straßer, Universität Tübingen

Tag der Einreichung: 18.10.2006
Tag der mündlichen Prüfung: 13.12.2006

D17
Darmstädter Dissertation 2006

Contents

List of Figures

List of Tables

Abstract

In computer graphics and robotics a lot of different mathematical systems like vector algebra, homogenous coordinates, quaternions or dual quaternions are used for different applications. Now it seems that a change of paradigm is lying ahead of us based on Conformal Geometric Algebra unifying all of these different approaches in one mathematical system.

Conformal Geometric Algebra is a very powerful mathematical framework. Due to its geometric intuitiveness, compactness and simplicity it is easy to develop new algorithms. Algorithms based on Conformal Geometric Algebra lead to enhanced quality, a reduction of development time, better understandable and better maintainable solutions. Often a clear structure and greater elegance results in lower runtime performance. However, it will be shown that algorithms based on Conformal Geometric Algebra can even be faster than conventional algorithms.

The main contribution of this thesis is the geometrically intuitive and - nevertheless - efficient algorithm for a computer animation application, namely an inverse kinematics algorithm for a virtual character. This algorithm is based on an embedding of quaternions in Conformal Geometric Algebra. For performance reasons two optimization approaches are used in a way to make the application now three times faster than the conventional solution.

With these results we are convinced that **Geometric Computing using Conformal Geometric Algebra** will become more and more fruitful in a great variety of applications in computer graphics and robotics.

Deutsche Zusammenfassung

0.1 Motivation

In der Computergraphik und in der Robotik werden eine ganze Reihe von unterschiedlichen mathematischen Systemen eingesetzt wie Vektoralgebra, homogene Koordinaten, Quaternionen und duale Quaternionen. Jetzt scheint ein Paradigmenwechsel vor uns zu liegen, der auf konformer geometrischer Algebra basiert, die in der Lage ist, diese unterschiedlichen Ansätze in einem mathematischen System zu vereinheitlichen.

Die konforme geometrische Algebra ist ein sehr mächtiges mathematisches Werkzeug. Dank ihrer geometrischen Intuitivität und Kompaktheit ist es einfach, mit ihr neue Algorithmen zu entwickeln. Algorithmen, die auf konformer geometrischer Algebra beruhen, führen zu erhöhter Qualität, einer Reduzierung von Entwicklungszeit, besser verständlichen und besser wartbaren Lösungen. Oft ist es jedoch so, dass eine klare Struktur und größere Eleganz zu einer geringeren Laufzeit führen. Demgegenüber kann aber in dieser Arbeit gezeigt werden, dass Algorithmen in konformer geometrischer Algebra sogar schneller sein können als herkömmliche Algorithmen.

0.2 Beitrag

Die Beiträge dieser Arbeit lassen sich wie folgt zusammenfassen:

- Untersuchung von Quaternionen und anderen bekannter Algebren in Hinsicht auf ihre direkte Verwendbarkeit in Algorithmen der konformen geometrischen Algebra
- Untersuchung der Rolle des Unendlichen in konformer geometrischer Algebra für ein besseres Verständnis ihrer geometrischen Basis-Objekte und -Operatoren
- Algorithmen zur Approximation von Punktmengen mit Hilfe von den Basisobjekten Kugel und Ebene der konformen geometrischen Algebra
- Nachweis der "Rapid Prototyping"-Möglichkeit zur visuellen und interaktiven Entwicklung und zum Test von Algorithmen am Beispiel der inversen Kinematik sowie des Greifprozesses eines Roboters
- Entwicklung und Nachweis der Effizienz eines Algorithmus auf Basis der konformen geometrischen Algebra am Beispiel der inversen Kinematik des Arms eines virtuellen Charakters

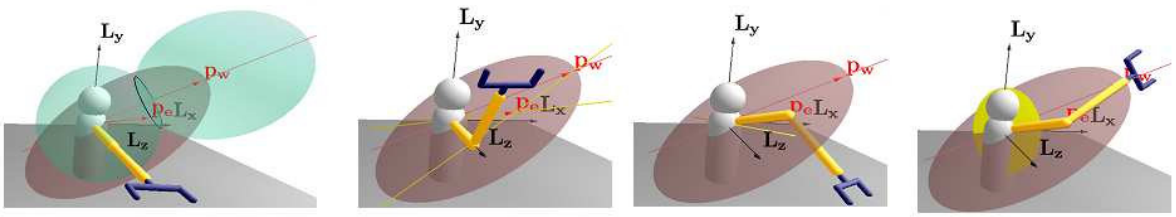

Figure 1: visuelle Entwicklung eines inverse Kinematik Algorithmus

Der Hauptbeitrag dieser Arbeit ist ein geometrisch intuitiver und dennoch effizienter Algorithmus für eine Computeranimationsanwendung, und zwar die inverse Kinematik des Arms eines virtuellen Charakters. Dieser Algorithmus basiert auf der Einbettung von Quaternionen in konformer geometrischer Algebra, da zur Interpolation der Bewegung des Arms Quaternionen verwendet werden. Aus Performancegründen werden zwei Optimierungsansätze genutzt, die dazu führen, dass die Anwendung nun drei mal schneller ist als die herkömmliche Lösung.

Figure 1 gibt einen Eindruck von der visuellen Art und Weise, in der der Algorithmus in der intuitiven Sprache der konformen geometrischen Algebra entwickelt werden kann. Diese Algebra erlaubt das einfache Rechnen mit geometrischen Objekten wie Kugeln und Ebenen, was mit Hilfe der Software CLUCalc in einem "Rapid Prototyping"-Prozess Schritt für Schritt umgesetzt werden kann. Ausgehend von der Berechnung des Ellenbogenpunktes wird der Winkel am Ellenbogen berechnet, der Arm zum Ellenbogenpunkt rotiert und letztendlich das Handgelenk zur Zielposition bewegt. Grundoperationen

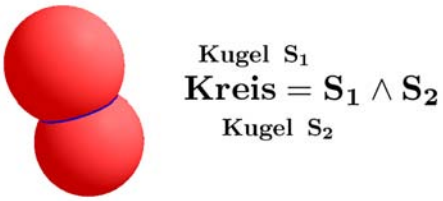

Figure 2: Schnitt zweier Kugeln S_1 und S_2

sind dabei Schnitte von geometrischen Objekten. Eine Kugel ist beispielsweise durch das folgende algebraische Objekt repräsentiert

$$S = P - \frac{1}{2}r^2 e_\infty. \tag{1}$$

Dieser Ausdruck beschreibt die Kugel mit ihrem Mittelpunkt P, ihrem Radius r und e_∞, was dem Punkt im Unendlichen entspricht. Um einen Kreis Z zu repräsentieren, müssen zwei Kugeln S_1 und S_2 miteinander geschnitten werden. Dies geschieht mit der folgenden algebraischen Operation

$$Z = S_1 \wedge S_2 \tag{2}$$

auf der Basis des elementaren äußeren Produkts.

Die Performancevorteile resultieren einerseits daraus, dass auf der Ebene des Algorithmus direkt mit Quaternionen gerechnet wird, während bei konventionellen Ansätzen noch eine Übersetzung von Transformationsmatrizen zu Quaternionen erforderlich ist. Auf der anderen Seite werden zur Implementierung alternativ der Codegenerator Gaigen 2 beziehungsweise die Mathematik-Software Maple eingesetzt. Die beiden Ansätze liefern vergleichbare Resultate. Während Gaigen 2 den Vorteil hat, dass die Eleganz der mathematischen Sprache der konformen geometrischen Algebra erhalten bleibt, hat die Maple-Lösung den Vorteil, dass der Algorithmus mit einem Standard-Compiler implementiert werden kann.

0.3 Zukunftsperspektiven

Mit den erreichten Ergebnissen wird "**Geometric Computing auf der Basis von konformer geometrischer Algebra**" zukünftig immer stärker Einzug halten in die unterschiedlichsten Anwendungen der Computergraphik und der Robotik. Eine beträchtliche Verbesserung der Laufzeit dürfte sich aus der Verwendung einer Hardware ergeben, die sich speziell auf Berechnungen der Geometrischen Algebra konzentriert. Hierzu wird eine Zusammenarbeit mit Forschern der Universität von Southampton angestrebt, die bereits einen Prototypen für ein ASIC entwickelt haben. Computerspiel-Engines bieten sich als ein Anwendungsgebiet an. Weitere Forschungen auf den Gebieten der Kinematik und der Dynamik können die Robotik sowie die Computeranimationen bereichern, beispielsweise auf dem Gebiet von "Virtual Kinematics", einem System zum virtuellen Zusammenbau und Test von beliebigen Kinematiken.

Acknowledgements

First of all I would like to thank my referees Prof. Marc Alexa, Prof. Dieter Fellner and Prof. Wolfgang Straßer for their support of my thesis. Special thanks to my advisor Prof. Marc Alexa pointing my attention to Conformal Geometric Algebra as well as always emphasizing the importance of efficiency for the practical use of the described technology. I would like to thank Prof. José Luis Encarnação for giving me the chance to work in his institute and for his support of this topic, especially in establishing a geometric computing working group in the Research Center of Excellence for Computer Graphics in Darmstadt. Many thanks to Prof. Eduardo Bayro-Corrochano and Julio Zamora-Esquivel for their cooperation concerning kinematics algorithms based on Conformal Geometric Algebra. For the best of my knowledge they are the first researchers applying kinematics algorithms to Conformal Geometric Algebra. I would also like to thank Dr. Christian Perwass for his support with CLUCalc. It is a pleasure for me cooperating with him. Many thanks to Dr. Leo Dorst and Daniel Fontijne for our successful cooperation. With the help of their tool Gaigen 2 we are now able to implement algorithms faster than conventional algorithms. Furthermore, I would like to thank Roland Martin and Dr. Eckhard Hitzer for their reviews of parts of this document and to Thomas Kalbe [45], Yusheng Wang [83], Haidan Zhang [91], Jun Zhao [92] and Holger Griesheimer [29] working with me as master students. Finally, I would like to thank my wife Carola for her interest shown in my work and her support during the last months in particular for proofreading of parts of this dissertation.

Chapter 1

Introduction

This chapter motivates the topic of this thesis and outlines the contribution and the structure of the document.

1.1 Motivation

Early in the development of computer graphics and robotics it was realized that projective geometry is very well suitable to represent transformations. Now we can realize that another change of paradigm is lying ahead of us which is based on the so-called **Conformal Geometric Algebra** [1] . In computer graphics and robotics a couple of diverse mathematical approaches like vector algebra, trigonometry, homogenous coordinates, quaternions or dual quaternions are used for different applications. This is why people, working in this area, have to learn and understand diverse approaches and how to translate between them, for instance between rotation matrices and quaternions. Conformal Geometric Algebra is able to unify all of these different approaches in one easy to understand mathematical system.

While points and vectors are normally used as basic geometric entities, Conformal Geometric Algebra provides a wider variety of basic geometric entities to compute with, namely points, spheres, planes, circles, lines and point pairs. A sphere is simply represented by the algebraic object

$$S = P - \frac{1}{2}r^2 e_\infty \tag{1.1}$$

based on its center point P, its radius r and e_∞ representing the point at infinity.

[1] Note that we capitalize "Conformal Geometric Algebra", since we want to distinguish this particular algebra from other algebras that describe geometry.

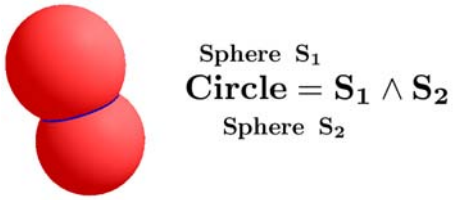

Figure 1.1: Intersection of two spheres

To represent a circle we only have to intersect two spheres, which can be done with the basic algebraic operation

$$Z = S_1 \wedge S_2 \tag{1.2}$$

See table 2.5 on page 19 for a list of the conformal geometric entities as well as their representations.

Spheres as basic computational entities can be used very advantageously in a lot of applications. For instance, robots or virtual characters are typically composed of rigid links connected to each other at joints. The possible positions of the end of each link, assuming complete rotational freedom at the joints, is of course a sphere. In applications like collision detection it is helpful to have an easy possibility to check against bounding spheres. Another interesting task with spheres as basic entities is for instance the approximation of points with the help of spheres.

At the Eurographics conference 2004 we presented a tutorial on the application of Geometric Algebra in computer graphics. We got a lot of positive feedback, especially

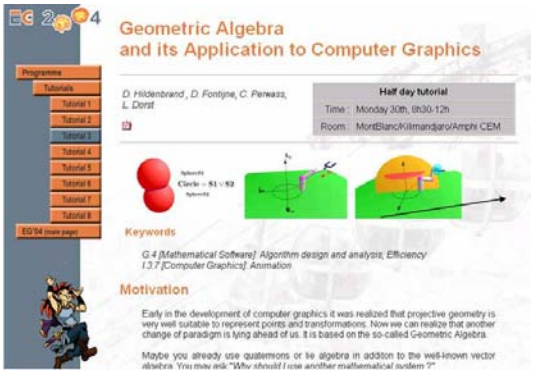

Figure 1.2: Tutorial at the Eurographics conference 2004

concerning the elegance and the geometric intuitiveness of this technology. However,

there was also a problem in terms of the runtime performance. Now we are pleased to present two approaches in order to implement algorithms very efficiently.

1.2 Contribution

The contributions of this thesis are specifically:

- Investigation of quaternions and other algebras with regard to the direct usage for algorithms in Conformal Geometric Algebra
- Analysis of the role of infinity in Conformal Geometric Algebra for a better understanding of its geometric entities and operators
- Algorithms for the approximation of point sets with the help of the basic Conformal Geometric Algebra objects sphere and plane
- Demonstration of the rapid prototyping functionality of Conformal Geometric Algebra for a visual and interactive development and test of algorithms with the help of kinematics applications of a robot
- Algorithm and proof of the efficience of an inverse kinematics application of a virtual character based on Conformal Geometric Algebra

Figure 1.3: Real-time inverse kinematics in the project "Virtual human" ([82])

The main contribution of this thesis is the efficient solution for the inverse kinematics of a human-arm-like model of a virtual character based on Conformal Geometric Algebra. We developed a new algorithm and implemented it in a virtual reality system (see figure 1.3). It is running in real-time in the project "Virtual human" ([82]). The

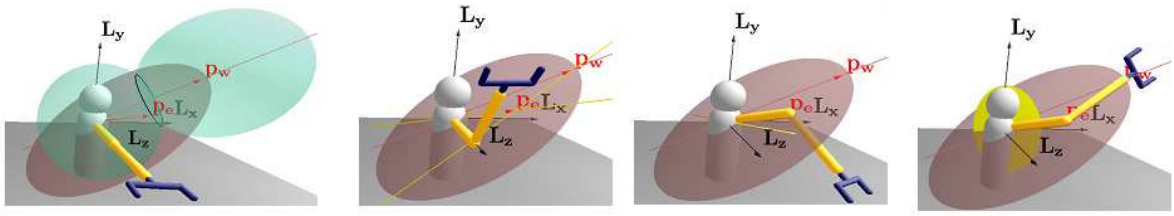

Figure 1.4: Visual development and test of an inverse kinematics algorithm

results are also applicable to robotics applications. Please refer to figure 1.4 for an impression of how visual this algorithm could be developed and tested based on the elegant and geometrically intuitive language of Conformal Geometric Algebra allowing to easily compute with geometric entities like spheres and planes. The software tool CLUCalc supports the rapid prototyping of algorithms step by step. Starting from the computation of the elbow point we compute the angle at the elbow, rotate the arm to the elbow point and finally rotate the wrist to the target position. The algorithm is based on our inverse kinematics algorithm for a virtual character in [36], which we further improved in [39]. For performance reasons we finally succeeded in [38] to improve the algorithm and its performance based on two optimization approaches in a way that we are now faster than the conventional algorithm.

Our inverse kinematics algorithm is based on the embedding of quaternions in Conformal Geometric Algebra. This is why we investigated how quaternions and some other algebras are identified and handled in Conformal Geometric Algebra. For a lot of algorithms a deeper understanding of the basic entities is helpful. A key role of our investigation plays infinity. We will see for instance how a plane is created by infinitely increasing the radius of a sphere. These planes and spheres can be used advantageously for the approximation of point sets by these objects ([35]). The above mentioned rapid prototyping of algorithms will be shown based on two robotics applications, namely the inverse kinematics of a simple robot (see [35]) as well as a grasping approach for a robot (see [39]). We introduced into this way of developing algorithms in [64] and [37].

1.3 Outline

This thesis is organized as follows:

- In Chapter 2 the state-of-the-art of Geometric Algebra and its applications in computer graphics and robotics is presented. After a short description of the history of Geometric Algebra, its properties, foundations and main products, we take a look at three specific algebras. Particularly the representation of geometric entities and the handling of transformations and motions in Conformal Geometric Algebra are shown. Furthermore the application of Conformal Geometric Algebra in computer graphics and robotics and the application of inverse kinematics of virtual characters are reviewed. We also present some helpful tools for the visual and interactive development of algorithms as well as their efficient implementation.

- Chapter 3 investigates the embedding of different mathematical systems that can be regarded as part of Conformal Geometric Algebra. We especially look at the handling of quaternions that we need for our algorithm in chapter 7. It is well-known that quaternions are part of the 3D Euclidean Geometric Algebra. Our goal is to identify and analyze them in Conformal Geometric Algebra and to realize what their geometric meaning is. In addition to quaternions we will also look at the identification of other mathematical systems like complex numbers, Plücker coordinates and dual quaternions.

- Chapter 4 investigates the role of infinity in Conformal Geometric Algebra. In order to better understand the geometric objects, that we are mainly using in our algorithms, we investigate the limits between them, for instance the transition between a sphere and a plane, how infinity is represented in Conformal Geometric Algebra and how translators are a limit of rotors.

- One big advantage of Conformal Geometric Algebra is its easy handling of objects like spheres. In computer graphics a lot of problems are related to this kind of objects. Chapter 5 presents the approximation of points with the help of spheres and/or planes as one important example.

- Chapter 6 presents the rapid prototyping of two robotics applications based on the software tool CLUCalc together with a link to the corresponding sources. Rapid prototyping means the interactive and visual development as well as the test of the algorithms. At first, we present step by step an algorithm for the inverse kinematics of a simple robot. The geometrically intuitive operations of Conformal Geometric Algebra make it easy to compute the joint angles of this robot to be set

in order for the robot to reach its new position. The second application, grasping of an object, shows how easy it is to develop and to visualize the algorithm of moving a gripper to an object.

- Chapter 7 presents our inverse kinematics algorithm of a human-arm-like model based on Conformal Geometric Algebra. Based on two optimization approaches for the implementation of this computer animation application, we present solutions that are three times faster than the conventional solution.
- We conclude in chapter 8 with a summary of the benefits of applying Conformal Geometric Algebra .
- Chapter 9 presents an outline of future work, namely virtual kinematics, dynamics, game engines and hardware support.

Chapter 2

State-of-the-art

In this chapter a brief introduction to Geometric Algebra and its applications in computer graphics and robotics is given. After a short description of the history of Geometric Algebra, its properties, foundations and main products, we take a look at three specific algebras: the 3D Euclidean Geometric Algebra, the 4D Projective Geometric Algebra and the 5D Conformal Geometric Algebra. Particularly the representation of geometric entities and the handling of transformations and motions in Conformal Geometric Algebra are shown. We focus on the geometric meaning of the algebra. References to more mathematical documents are given.

Furthermore the state-of-the-art of the application of Conformal Geometric Algebra in computer graphics and robotics as well as the state-of-the-art of inverse kinematics of virtual characters is reviewed. We also present some very useful tools for the visual and interactive development of algorithms as well as their efficient implementation. A more detailed discussion of specific papers will be given in the subsequent chapters.

2.1 History of Geometric Algebra

The foundation of Geometric Algebra was already laid in 1844 by the German high school teacher Hermann Grassmann. William K. Clifford (1845-1879) introduced what we now call Geometric or Clifford Algebra in a paper entitled "On the classification of geometric algebras" [17]. He realized that Grassmann's extensive algebra and Hamilton's quaternions can be brought into the same algebra by a slight change of the exterior product. With this new product, called geometric product, the multiplication rules of the quaternions follow directly from combinations of basis vectors (more details later). However, due to the early death of Clifford, the vector calculus of Gibbs dominated

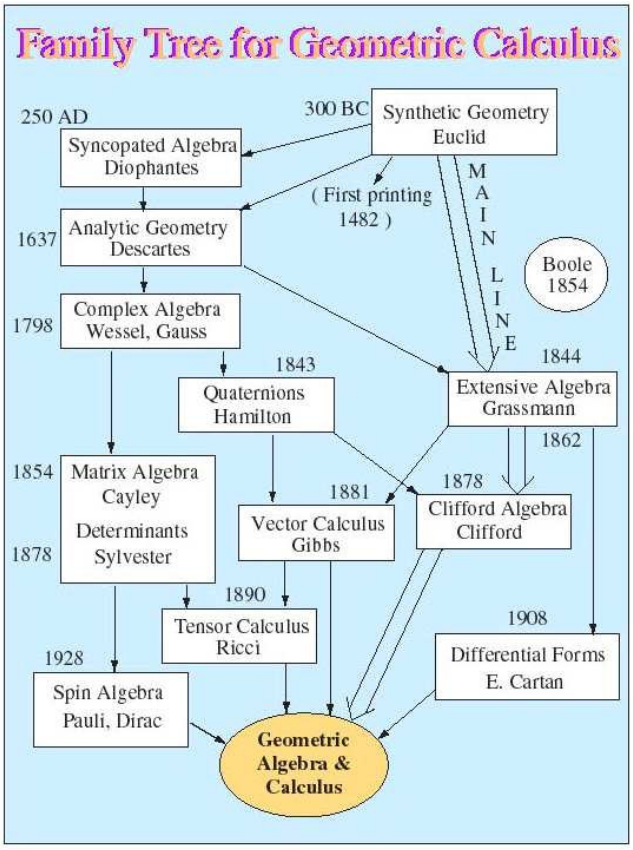

Figure 2.1: History of Geometric Algebra and Calculus [33]

most of the 20th century, and not the Geometric Algebra. Please refer to a family tree of mathematical systems according to David Hestenes in figure 2.1.

Geometric Algebra has found its way into many areas of science, since David Hestenes treated the subject in the '1960s. In particular, his aim was to find a unified language for mathematics, and he went about to show the advantages that could be gained by using Geometric Algebra in many areas of physics and geometry [32, 34].

Early in our century David Hestenes et al. succeeded in introducing the (5D) Conformal Geometric Algebra (see [51] and [31]). It is an extension of the 4D Projective Geometric Algebra. Alyn Rockwood, David Hestenes and Hongbo Li hold a US patent for the use of the conformal model ([69]). There are no restrictions for academic research and educational use. However, the patent requires a license agreement with the inventors for commercial products in the US.

For details on the algebra please refer for instance to the books from Hestenes [30], Sommer [76] and Doran/Lasenby [19]. Some Geometric Algebra tutorials can be found in [37, 64, 21, 22, 57, 41, 40, 54, 42].

2.2 Properties of Geometric Algebra

Geometric Algebra promises to stimulate new methods and insights in all areas of science dealing with geometric properties. It treats geometric objects and operators on these objects in one algebra. Furthermore it allows for simple, compact, coordinate-free and dimensionally fluid formulations.

Geometric Intuitiveness

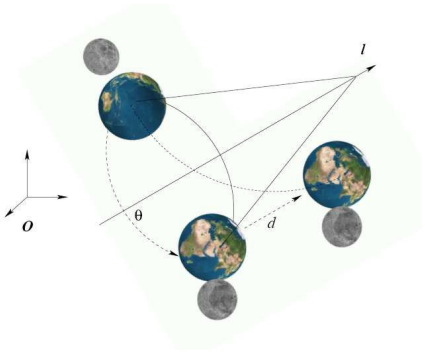

Figure 2.2: Rigid body motion [70]

A very favourable feature of Conformal Geometric Algebra is its geometric intuitiveness. As shown in section 1.1, spheres and circles are both algebraic objects with a geometric meaning. To represent a circle you only have to take two spheres and to intersect them with the help of their outer product. Another beneficial example are rigid body motions. They are described with the help of an operator including the relevant geometric parameters, namely the rotation axis, the angle of rotation and the displacement (see section 2.9.3 for details).

Unification

According to table 2.4 on page 17 Conformal Geometric Algebra is based on 32 algebraic entities, called blades. For details please refer to section 2.3. A lot of mathematical

systems use subsets of these 32 blades. In table 3.2 on page 38 you can see for instance the four blades for quaternions representing the scalar part and the three imaginary parts of quaternions. While we usually know quaternions as a special number system, somehow mysteriously handy for computations with rotations, we will see in section 3.2, that quaternions in Conformal Geometric Algebra have an intuitive meaning of rotation axis and angles.

Furthermore Conformal Geometric Algebra includes also a lot of other mathematical systems like vector algebra, projective geometry, complex numbers, Plücker coordinates and dual quaternions. Please find details in chapter 3.

Elegance

The elegance of Geometric Algebra is well demonstrated in a ray tracing application developed at the university of Amsterdam: 5 models of geometry are compared in this standard graphics application. Please take note of the paper [27] and the web page [26] for your information and download.

In particular kinematics algorithms can be expressed in an elegant way in Conformal Geometric Algebra as will be shown in the chapters 6 and 7.

Low symbolic complexity

Expressions in Geometric Algebra normally have low complexity. As will be shown in section 5.1, the inner product of two vectors $A \cdot B$ can be used for different tasks like

- the Euclidean distance between two points
- the distance between one point and one plane
- the decision whether a point is inside or outside of a sphere

2.3 Foundations of Geometric Algebra

Blades are the basic computational elements and the basic geometric entities of the Geometric Algebra. A n-dimensional Geometric Algebra consists of blades with **grades** 0, 1, 2 .. n, whereby a scalar is a **0-blade** (blade of grade 0) and the **1-blades** are the basis vectors (e_1, e_2, e_3 for the 3D Euclidean Geometric Algebra). The **2-blades** are blades spanned by two 1-blades, and so on. There exists only one element of maximum grade n. It is therefore also called the pseudoscalar. Please find a list of all the

- 8 blades of 3D Euclidean Geometric Algebra in table 2.2 on page 14
- 16 blades of 4D Projective Geometric Algebra in table 2.3 on page 16
- 32 blades of 5D Conformal Geometric Algebra in table 2.4 on page 17

A linear combination of k-blades is called a k-vector (also called vectors, bivectors, trivectors ...). Furthermore, a linear combination of blades with different grades is called a multivector. Multivectors are the general elements of a Geometric Algebra.

2.4 The products of Geometric Algebra

Table 2.1: Notations of Conformal Geometric Algebra

notation	**meaning**	**alternative**
AB	geometric product of A and B	
$A \wedge B$	outer product of A and B	A^B
$A \cdot B$	inner product of A and B	A.B, LC(A,B)
A^*	dual of A	dual(A)
A^{-1}	inverse of A	1/A
$\tilde{A}$	reverse of A	
e_0	conformal origin	e0
e_∞	conformal infinity	einf

The three most often used products of Geometric Algebra are the **outer**, the **inner** and the **geometric** product. In table 2.1 the notations of these products are listed. We will use the outer product mainly for the construction and intersection of geometric objects while the inner product will be used for the computation of angles and distances. The geometric product will be used mainly for the description of transformations.

2.4.1 The outer product

Geometric Algebra provides an outer product $\wedge$ with the following properties

	Property	Meaning
1.	anti-symmetry	$u \wedge v = -(v \wedge u)$
2.	linearity	$u \wedge (v + w) = u \wedge v + u \wedge w$
3.	associativity	$u \wedge (v \wedge w) = (u \wedge v) \wedge w$

The outer product of parallel vectors is 0.

$$a \wedge a = -(a \wedge a) = 0 \tag{2.1}$$

This is the reason why the outer product can be used as a measure for parallelness.
Computation example:

$$b = (e_1 + e_2) \wedge (e_1 - e_2)$$

can be transformed based on linearity to

$$b = (e_1 \wedge e_1) - (e_1 \wedge e_2) + (e_2 \wedge e_1) - (e_2 \wedge e_2)$$

since $a \wedge a = 0$

$$b = -(e_1 \wedge e_2) + (e_2 \wedge e_1)$$

because of anti-symmetry

$$b = -(e_1 \wedge e_2) - (e_1 \wedge e_2)$$

or

$$b = -2(e_1 \wedge e_2)$$

2.4.2 The inner product

For the 3D Euclidean space, the inner product of 2 vectors is the same as the well known Euclidean scalar product of 2 vectors. For perpendicular vectors the inner product is 0, for instance $e_1 \cdot e_2 = 0$. In Geometric Algebra, the inner product is not only defined for vectors. The inner product is grade decreasing, e. g. the result of the inner product of an element with grade 2 and grade 1 is an element of grade 2-1 =1. Please refer to [64] and [37] for a mathematical treatment.

Please find detailed information on the geometric meaning of the inner product in Conformal Geometric Algebra in section 5.1 (inner product and distances) and in section 6.1 (inner product and angles).

2.4.3 The geometric product

The geometric product is an amazingly powerful operation. It has a lot of geometric meaning whereby the easy handling of transformations is the most important one. The geometric product is a combination of the outer product and the inner product. The geometric product of u and v is denoted by uv. For vectors u and v the geometric product uv is defined as

$$uv = u \wedge v + u \cdot v \tag{2.2}$$

We derive for the inner and the outer product

$$u \cdot v = \frac{1}{2}(uv + vu) \tag{2.3}$$

$$u \wedge v = \frac{1}{2}(uv - vu) \tag{2.4}$$

Computation examples:
What is the square of a vector ?

$$a^2 = aa = \underbrace{a \wedge a}_{0} + a \cdot a = a \cdot a$$

for example

$$e_1^2 = e_1 \cdot e_1 = 1$$

The geometric product is not only defined for vectors but for all kind of multivectors. Let us for example calculate the geometric product of 2 bivectors $u = e_1 \wedge e_2$ and $v = (e_1 + e_2) \wedge e_3$

$$uv = (e_1 \wedge e_2)((e_1 + e_2) \wedge e_3)$$

since $e_1 e_2 = e_1 \wedge e_2 + \underbrace{e_1 \cdot e_2}_{0} = e_1 \wedge e_2$

$$uv = (e_1 e_2)(e_1 \wedge e_3 + e_2 \wedge e_3)$$

$$= e_1 e_2 (e_1 e_3 + e_2 e_3)$$

$$= e_1 e_2 e_1 e_3 + e_1 \underbrace{e_2 e_2}_{1} e_3$$

since $e_1 e_2 = e_1 \wedge e_2 = -e_2 \wedge e_1 = -e_2 e_1$

$$uv = -e_2 e_1 e_1 e_3 + e_1 e_3$$

$$= -e_2 e_3 + e_1 e_3$$

$$= -(e_2 \wedge e_3) + e_1 \wedge e_3$$

Table 2.2: The 8 blades of the 3D Euclidean Geometric Algebra

grade	**term**	**blades**	**nr.**
0	scalar	1	1
1	vector	e_1, e_2, e_3	3
2	bivector	$e_1 \wedge e_2,\ \ e_1 \wedge e_3,\ \ e_2 \wedge e_3$	3
3	pseudoscalar	$e_1 \wedge e_2 \wedge e_3$	1

The **inverse** of a blade A is defined by

$$AA^{-1} = 1$$

The inverse of a vector v is

$$v^{-1} = \frac{v}{v \cdot v}$$

because

$$v\frac{v}{v \cdot v} = \frac{v \cdot v}{v \cdot v} = 1$$

Divisions by algebraic objects are possible due to the fact that the geometric product is invertible. The **dual** of an algebraic object is calculated with the help of its division by the pseudoscalar. The **reverse** is an operator simply reversing the order of vectors in a blade. The notations of these operations are listed in table 2.1 on page 11. Please refer to [64] and [37] for mathematical details.

2.5 Euclidean Geometric Algebra

Euclidean Geometric Algebra includes the well-known vector algebra. It is dealing with the 3 Euclidean basis vectors e_1, e_2, e_3. Linear combinations of these basis vectors can be interpreted as 3D vectors or 3D points (please find a list of all the 8 blades of the 3D Euclidean Geometric Algebra in table 2.2). The **scalar product** is identical with the inner product. The Euclidean **cross product** of the two Euclidean vectors $\mathbf{u}$ and $\mathbf{v}$ can also be written in Geometric Algebra form as

$$\mathbf{u} \times \mathbf{v} = -(\mathbf{u} \wedge \mathbf{v})e_{123} \tag{2.5}$$

with $e_{123} = e_1 \wedge e_2 \wedge e_3$ as the Euclidean pseudoscalar (blade with grade 3).
In order to prove this equation, we would like to calculate at first an expression for the outer product of the vectors $\mathbf{u}$ and $\mathbf{v}$

$$\mathbf{u} \wedge \mathbf{v} = (u_1 e_1 + u_2 e_2 + u_3 e_3) \wedge (v_1 e_1 + v_2 e_2 + v_3 e_3)$$

$$= u_1 v_2 (e_1 \wedge e_2) + u_1 v_3 (e_1 \wedge e_3) + u_2 v_1 (e_2 \wedge e_1) + u_2 v_3 (e_2 \wedge e_3) + u_3 v_1 (e_3 \wedge e_1) + u_3 v_2 (e_3 \wedge e_2)$$

$$= u_1 v_2 (e_1 \wedge e_2) + u_1 v_3 (e_1 \wedge e_3) - u_2 v_1 (e_1 \wedge e_2) + u_2 v_3 (e_2 \wedge e_3) - u_3 v_1 (e_1 \wedge e_3) - u_3 v_2 (e_2 \wedge e_3)$$

leading to the following equation for the outer product of the vectors $\mathbf{u}$ and $\mathbf{v}$:

$$\mathbf{u} \wedge \mathbf{v} = (u_1 v_2 - u_2 v_1)(e_1 \wedge e_2) + (u_1 v_3 - u_3 v_1)(e_1 \wedge e_3) + (u_2 v_3 - u_3 v_2)(e_2 \wedge e_3) \quad (2.6)$$

Let us now compute the expression $-(\mathbf{u} \wedge \mathbf{v}) e_{123}$

$$-(\mathbf{u} \wedge \mathbf{v}) e_{123} = -((u_1 v_2 - u_2 v_1) e_1 e_2 + (u_1 v_3 - u_3 v_1) e_1 e_3 + (u_2 v_3 - u_3 v_2) e_2 e_3) e_{123}$$

$$= (u_1 v_2 - u_2 v_1) e_2 e_1 e_1 e_2 e_3 + (u_1 v_3 - u_3 v_1) e_3 e_1 e_1 e_2 e_3 + (u_2 v_3 - u_3 v_2) e_3 e_2 e_1 e_2 e_3$$

$$= (u_1 v_2 - u_2 v_1) e_2 e_2 e_3 + (u_1 v_3 - u_3 v_1) e_3 e_2 e_3 - (u_2 v_3 - u_3 v_2) e_3 e_1 e_2 e_2 e_3$$

$$= (u_1 v_2 - u_2 v_1) e_3 - (u_1 v_3 - u_3 v_1) e_2 e_3 e_3 - (u_2 v_3 - u_3 v_2) e_3 e_1 e_3$$

$$= (u_1 v_2 - u_2 v_1) e_3 - (u_1 v_3 - u_3 v_1) e_2 + (u_2 v_3 - u_3 v_2) e_3 e_3 e_1$$

$$= (u_1 v_2 - u_2 v_1) e_3 - (u_1 v_3 - u_3 v_1) e_2 + (u_2 v_3 - u_3 v_2) e_1$$

leading to the equation

$$-(\mathbf{u} \wedge \mathbf{v}) e_{123} = (u_2 v_3 - u_3 v_2) e_1 - (u_1 v_3 - u_3 v_1) e_2 + (u_1 v_2 - u_2 v_1) e_3 \quad (2.7)$$

with the right side equal to the definition of the cross product

$$\mathbf{u} \times \mathbf{v} = (u_2 v_3 - u_3 v_2) e_1 - (u_1 v_3 - u_3 v_1) e_2 + (u_1 v_2 - u_2 v_1) e_3 \quad (2.8)$$

Table 2.3: The 16 blades of the 4D Projective Geometric Algebra

grade	**term**	**blades**	**nr.**
0	scalar	1	1
1	vector	e_1, e_2, e_3, e_0	4
2	bivector	$e_1 \wedge e_2,\ \ e_1 \wedge e_3,\ \ e_2 \wedge e_3,$ $e_1 \wedge e_0,\ \ e_2 \wedge e_0,\ \ e_3 \wedge e_0$	6
3	trivector	$e_1 \wedge e_2 \wedge e_3,\ \ e_1 \wedge e_2 \wedge e_0,$ $e_1 \wedge e_3 \wedge e_0,\ \ e_2 \wedge e_3 \wedge e_0$	4
4	pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_0$	1

2.6 Projective Geometric Algebra

Projective Geometric Algebra is a 4D Geometric Algebra. Please refer to the list of its 16 blades in table 2.3. Based on this algebra projective geometry is a part of Conformal Geometric Algebra. The **inhomogenous** point

$$\mathbf{x} = x_1 e_1 + x_2 e_2 + x_3 e_3 \tag{2.9}$$

is transformed to a **homogenous** point via

$$X = \mathbf{x} + e_0 \tag{2.10}$$

Please note that based on this mapping the origin is mapped to e_0.
Vice-versa, an arbitrary homogenous point

$$X = w x_1 e_1 + w x_2 e_2 + w x_3 e_3 + w e_0,\ \ w \neq 0 \tag{2.11}$$

is at first scaled into the hyperplane

$$X' = x_1 e_1 + x_2 e_2 + x_3 e_3 + e_0 \tag{2.12}$$

and afterwards projected to the inhomogenous point $\mathbf{x} = x_1 e_1 + x_2 e_2 + x_3 e_3$.
Please find further details for instance in [64].

2.7 The Conformal Geometric Algebra

In this document we focus on the 5D Conformal Geometric Algebra. One advantage of this algebra is that points, spheres and planes are easily represented as vectors (grade 1

Table 2.4: The 32 blades of the 5D Conformal Geometric Algebra

grade	**term**	**blades**	**nr.**
0	scalar	1	1
1	vector	$e_1, e_2, e_3, e_0, e_\infty$	5
2	bivector	$e_1 \wedge e_2,\ e_1 \wedge e_3,\ e_2 \wedge e_3,$ $e_1 \wedge e_\infty,\ e_2 \wedge e_\infty,\ e_3 \wedge e_\infty,$ $e_1 \wedge e_0,\ e_2 \wedge e_0,\ e_3 \wedge e_0,$ $e_0 \wedge e_\infty$	10
3	trivector	$e_1 \wedge e_2 \wedge e_3,\ e_1 \wedge e_2 \wedge e_0,\ e_1 \wedge e_2 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0,\ e_1 \wedge e_3 \wedge e_\infty,\ e_1 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0,\ e_2 \wedge e_3 \wedge e_\infty,\ e_2 \wedge e_0 \wedge e_\infty,$ $e_3 \wedge e_0 \wedge e_\infty$	10
4	quadvector	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty,$ $e_1 \wedge e_2 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_2 \wedge e_0 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	5
5	pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	1

blades). "Conformal" comes from the fact that it handles the conformal transformations easily. These transformations leave angles invariant (see details for instance in [19]).

The Conformal Geometric Algebra uses 2 additional basis vectors e_+, e_- with positive signature (e_+) and negative signature (e_-).

$$e_+^2 = 1 \qquad e_-^2 = -1 \qquad e_+ \cdot e_- = 0 \tag{2.13}$$

Another basis e_o, e_∞ with the following geometric meaning

- e_0 representing the 3D origin
- e_∞ representing infinity

can be defined with the relations

$$e_o = \frac{1}{2}(e_- - e_+) \qquad e_\infty = e_- + e_+ \tag{2.14}$$

These new basis vectors are null vectors

$$e_o^2 = e_\infty^2 = 0 \tag{2.15}$$

Their inner product results in

$$e_\infty \cdot e_o = -1 \tag{2.16}$$

since

$$(e_- + e_+) \cdot \frac{1}{2}(e_- - e_+) = \frac{1}{2}(\underbrace{e_- \cdot e_-}_{-1} - \underbrace{e_- \cdot e_+}_{0} + \underbrace{e_+ \cdot e_-}_{0} - \underbrace{e_+ \cdot e_+}_{1})$$

and their geometric product in

$$e_\infty e_o = e_\infty \wedge e_o + e_\infty \cdot e_o = e_\infty \wedge e_o - 1 \tag{2.17}$$

or

$$e_o e_\infty = e_o \wedge e_\infty + e_\infty \cdot e_o = -e_\infty \wedge e_o - 1 \tag{2.18}$$

The outer product $e_\infty \wedge e_o$ is often abbreviated by E.

Please find a list of all the 32 blades of the 5D Conformal Geometric Algebra in table 2.4.

2.8 The basic geometric entities

Conformal Geometric Algebra provides a great variety of basic geometric entities to compute with, namely points, spheres, planes, circles, lines and point pairs as listed in table 2.5. They have two algebraic representations: 'standard' and 'direct'. These representations are duals of each other (the superscription of "$*$" denotes the dualization operator).

In table 2.5 $\mathbf{x}$ and $\mathbf{n}$ are marked bold to indicate that they represent 3D entities by linear combinations of the 3D basis vectors e_1, e_2 and e_3.

$$\mathbf{x} = x_1 e_1 + x_2 e_2 + x_3 e_3 \tag{2.19}$$

The $\{S_i\}$ represent different spheres and the $\{\pi_i\}$ represent different planes. In the direct representation the outer product '$\wedge$' indicates the construction of geometric objects with the help of points $\{P_i\}$ that lie on it. A sphere is for instance defined by 4 points $(P_1 \wedge P_2 \wedge P_3 \wedge P_4)$ determining this sphere.
In the standard representation the meaning of the outer product is the intersection of geometric entities. A circle is for instance defined by the intersection of two spheres $S_1 \wedge S_2$ (please refer to figure 1.1 on page 2).

Table 2.5: List of the conformal geometric entities

entity	**standard representation**	**direct representation**
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	
Sphere	$S = P - \frac{1}{2}r^2 e_\infty$	$S^* = P_1 \wedge P_2 \wedge P_3 \wedge P_4$
Plane	$\pi = \mathbf{n} + d e_\infty$	$\pi^* = P_1 \wedge P_2 \wedge P_3 \wedge e_\infty$
Circle	$Z = S_1 \wedge S_2$	$Z^* = P_1 \wedge P_2 \wedge P_3$
Line	$L = \pi_1 \wedge \pi_2$	$L^* = P_1 \wedge P_2 \wedge e_\infty$
Point Pair	$Pp = S_1 \wedge S_2 \wedge S_3$	$Pp^* = P_1 \wedge P_2$

2.8.1 Points

In order to represent points in 5D conformal space, the original 3D point $\mathbf{x}$ is projectively extended to a 5D vector by linear combinations of the 5D basis vectors e_1, e_2, e_3, e_∞ and e_0 according to the equation

$$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0, \tag{2.20}$$

whereby $\mathbf{x}^2$ is the well-known scalar product

$$\mathbf{x}^2 = x_1^2 + x_2^2 + x_3^2. \tag{2.21}$$

E. g. for the 3D origin (0,0,0) we get

$$P(0,0,0) = e_0 \tag{2.22}$$

or for the 3D point (0,1,0)

$$P_y = P(0,1,0) = e_2 + \frac{1}{2}e_\infty + e_0. \tag{2.23}$$

2.8.2 Spheres

A sphere is on the one hand represented with the help of its center point P and its radius r

$$S = P - \frac{1}{2}r^2 e_\infty. \tag{2.24}$$

Note that the representation of a point is simply a sphere with radius zero.
A sphere can on the other hand be represented with the help of 4 points that lie on it

$$S^* = P_1 \wedge P_2 \wedge P_3 \wedge P_4. \tag{2.25}$$

2.8.3 Planes

A plane is defined by

$$\pi = \mathbf{n} + de_\infty \tag{2.26}$$

whereby $\mathbf{n}$ refers to the 3D normal vector of the plane π and d is the distance to the origin. A plane can also be defined with the help of three points that lie on it and the point at infinity

$$\pi^* = P_1 \wedge P_2 \wedge P_3 \wedge e_\infty. \tag{2.27}$$

Would you please notice that a plane is a sphere with infinite radius (details in chapter 4).

2.8.4 Circles

A circle is defined by the intersection of two spheres

$$Z = S_1 \wedge S_2 \tag{2.28}$$

or with the help of three points that lie on it

$$Z^* = P_1 \wedge P_2 \wedge P_3. \tag{2.29}$$

2.8.5 Lines

A line is defined by the intersection of two planes

$$L = \pi_1 \wedge \pi_2 \tag{2.30}$$

or with the help of two points that lie on it and the point at infinity

$$L^* = P_1 \wedge P_2 \wedge e_\infty. \tag{2.31}$$

For example the y-axis L_y can be described by

$$L_y^* = e_0 \wedge P_y \wedge e_\infty \tag{2.32}$$

whereby e_0 represents the origin (see equation (2.22)) and P_y is the point of equation (2.23). Please note that a line can be regarded as a circle with infinite radius.

2.8.6 Point Pairs

A point pair is defined by the intersection of three spheres

$$Pp = S_1 \wedge S_2 \wedge S_3 \tag{2.33}$$

or directly with the help of the two points

$$Pp^* = P_1 \wedge P_2. \tag{2.34}$$

2.9 Transformations and motions

Transformations and motions are easily described in Conformal Geometric Algebra based on algebraic objects.

2.9.1 Transformations

All kind of transformations of an object o are done in Conformal Geometric Algebra with the help of the following geometric product

$$o_{transformed} = VoV\tilde{} \tag{2.35}$$

with V being a so-called **versor** and with $\tilde{V}$ as its reverse.

Rotor

The operator

$$R = e^{-\frac{\phi}{2}L} \tag{2.36}$$

describes a so-called **rotor** .
L is the rotation axis represented by a normalized bivector and ϕ is the rotation angle around this axis. Please note that L can be an arbitrary line and not only going through the origin.
R can also be written as

$$R = cos(\frac{\phi}{2}) - Lsin(\frac{\phi}{2}). \tag{2.37}$$

The rotation of a geometric object o is performed with the help of the operation

$$o_{rotated} = RoR\tilde{}.$$

Translator

In Conformal Geometric Algebra, a translation can be expressed in a multiplicative way with the help of a **translator** T defined by

$$T = e^{-\frac{1}{2}\mathbf{t}e_\infty} \tag{2.38}$$

whereby $\mathbf{t}$ is a vector

$$\mathbf{t} = t_1e_1 + t_2e_2 + t_3e_3.$$

The application of the Taylor series

$$T = e^{-\frac{1}{2}\mathbf{t}e_\infty} = 1 + \frac{-\frac{1}{2}\mathbf{t}e_\infty}{1!} + \frac{(-\frac{1}{2}\mathbf{t}e_\infty)^2}{2!} + \frac{(-\frac{1}{2}\mathbf{t}e_\infty)^3}{3!}...$$

and of the property $(e_\infty)^2 = 0$ result in the translator

$$T = 1 - \frac{1}{2}\mathbf{t}e_\infty. \tag{2.39}$$

Please find some more detailed information in [70].

Example:

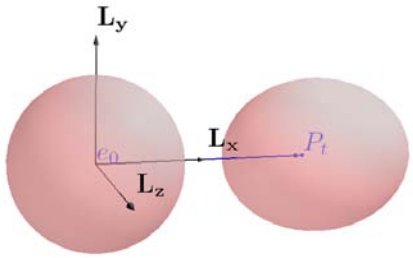

Figure 2.3: Translation of a sphere

Versors do not only transform points but they are able to transform complete geometric objects. Let us for instance translate the sphere

$$S = -e_\infty + e_0 \tag{2.40}$$

with its center at the origin (see figure 2.3) in x-direction with the translation vector

$$\mathbf{t} = 4e_1. \tag{2.41}$$

The translator has the form

$$T = 1 - 2e_1 e_\infty \tag{2.42}$$

with its reverse

$$\tilde{T} = 1 + 2e_1 e_\infty. \tag{2.43}$$

The translated sphere can now be computed as the versor product

$$S_{translated} = T S \tilde{T} \tag{2.44}$$

$$= (1 - 2e_1 e_\infty)(-e_\infty + e_0)(1 + 2e_1 e_\infty)$$

$$= (1 - 2e_1 e_\infty)(-e_\infty - 2e_\infty e_1 e_\infty + e_0 + 2e_0 e_1 e_\infty)$$

$$= (1 - 2e_1 e_\infty)(-e_\infty + e_0 - 2e_1 e_0 e_\infty)$$

$$= -e_\infty + e_0 - 2e_1 e_0 e_\infty + 2e_1 e_\infty e_\infty - 2e_1 e_\infty e_0 + 4e_1 e_\infty e_1 e_0 e_\infty$$

$$= -e_\infty + e_0 - 2e_1 e_0 e_\infty + 2e_1 \underbrace{e_\infty e_\infty}_{0} - 2e_1 e_\infty e_0 + 4e_1 e_\infty e_1 e_0 e_\infty$$

$$= -e_\infty + e_0 - 2e_1 (\underbrace{e_0 e_\infty + e_\infty e_0}_{-2}) + 4e_1 e_\infty e_1 e_0 e_\infty$$

$$= 4e_1 - e_\infty + e_0 + 4 \underbrace{e_1 e_\infty e_1}_{-e_\infty} e_0 e_\infty$$

$$= 4e_1 - e_\infty + e_0 - 4e_\infty \underbrace{e_0 e_\infty}_{-e_\infty \wedge e_0 - 1}$$

$$= 4e_1 - e_\infty + e_0 - 4 \underbrace{e_\infty (-e_\infty \wedge e_0 - 1)}_{-2e_\infty}$$

resulting in

$$S_{translated} = 4e_1 + 7e_\infty + e_0. \tag{2.45}$$

This is a sphere with the same radius but with the translated center point

$$P_t = \mathbf{t} + \frac{1}{2}\mathbf{t}^2 e_\infty + e_0. \tag{2.46}$$

2.9.2 Rigid body motion

A motion in 3D includes both a rotation and a translation. In Conformal Geometric Algebra a rigid body motion is described by one operator M, a so-called **motor**

$$M = RT \tag{2.47}$$

with R being a rotor and T being a translator(see section 2.9.1).
A rigid body motion of an object o is described by

$$o_{rigid\_body\_motion} = Mo\tilde{M}.$$

2.9.3 Screw motion

An alternative description of a rigid body motion is a screw motion that is very suitable for the interpolation of motions. A screw motion describes a rigid body motion in a compact form including both a rotation and translation in the direction of the rotation axis. The screw motion of an object o is described by

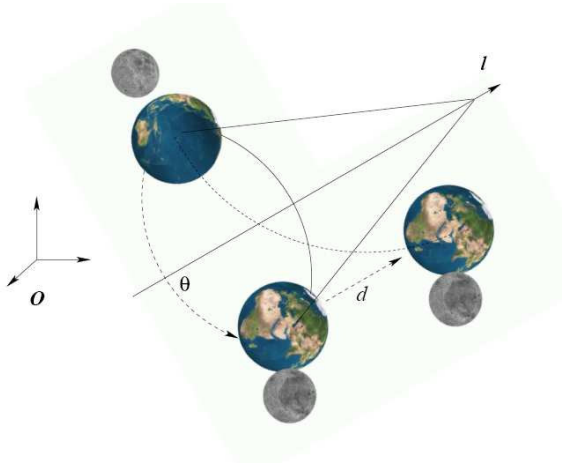

Figure 2.4: Screw motion along l [70]

$$o_{screw\_motion} = Mo\tilde{M}$$

with the motor

$$M = e^{-\frac{\theta}{2}(L+e_\infty \mathbf{m})} \tag{2.48}$$

whereby

- L is a bivector representing an axis through the origin,
- θ is the angle of rotation,
- $\mathbf{m}$ is a 3D vector.

If L is zero, a pure translation is described.
If $\mathbf{m}$ is zero, a pure rotation is described.
Another form of the motor is

$$M = e^{-\frac{\theta}{2}(l+e_\infty \mathbf{d})} \tag{2.49}$$

whereby (see figure 2.4)

- $l = L + e_\infty \mathbf{m}^\perp$ is a bivector representing an arbitrary axis,
- θ is the rotation angle,
- $\mathbf{d} = \mathbf{m}^\parallel$ is a vector parallel to the axis l.

For details please refer for instance to [70].

2.9.4 Interpolation of transformations

The exponent of a motor representing a screw motion is called a **twist**. We assume two transformations described by the two twists W_1 and W_2:

$$W_1 = -\frac{\theta_1}{2}(L_1 + e_\infty \mathbf{d_1})$$

$$W_2 = -\frac{\theta_2}{2}(L_2 + e_\infty \mathbf{d_2})$$

Interpolations between these two transformations can be described by interpolating their twists, e. g. in a linear manner

$$W(t) = (1-t) * W_1 + t * W_2 \tag{2.50}$$

with the resulting motor

$$M(t) = e^{W(t)}. \tag{2.51}$$

For $t \in [0..1]$ we get

$$M(0) = e^{W_1}, M(1) = e^{W_2}.$$

This kind of interpolation of transformations is comparable to Alexa [4]. Please find more detailed information and some other interpolation approaches in [84].

2.9.5 Relations to quaternions and dual quaternions

There are strong relations between transformations in Conformal Geometric Algebra and quaternions and dual quaternions.

In section 3.2 we will see that quaternions are in principle rotors with a rotation axis through the origin.

In section 3.5 we will see that dual quaternions are in principle versors describing for instance rotations with an arbitrary rotation axis. This is shown based on the equivalence of the versor $TR\tilde{T}$ (describing a translation to the origin, a rotation at the origin and a translation back) and a dual quaternion. This operation is especially helpful to describe the movement of a robot at a revolute joint.

2.10 Conformal Geometric Algebra in computer graphics

Please find hereafter some research groups working with Conformal Geometric Algebra in the field of computer graphics.

Since about one decade, many researchers at the **University of Cambridge, UK** have shown that applying Geometric Algebra in their field of research is very advantageous. They started with projects more related to computer vision. In the meantime the Cambridge engineering department as well as a company with university background are dealing with typical computer graphics applications like mesh deformation and lighting. Lasenby et al. and Perwass et al. present some applications dealing with structure and motion estimation as well as with the trifocal tensor in the articles [44], [48] and [67, 68, 66]. Some computer graphics articles using Geometric Algebra are presented by Cameron et al. [16] and Wareham et al. [84], [85]. They use Geometric Algebra for applications like rigid-body pose and position interpolation, mesh deformation and catadioptric cameras. Geomerics [1] is a new start-up company in Cambridge specializing in simulation software for physics and lighting which just presented its new technology allowing real-time radiosity in videogames utilizing commodity graphics processing hardware. The technology is Precomputed Radiance Transfer (PRT) based on Geometric Algebra wavelet technology.

Dorst et al. at the **University of Amsterdam**, the Netherlands, are applying their fundamental research on Geometric Algebra [23, 21, 22, 53, 54] to computer graphics. Zaharia et al. investigated modeling and visualization of 3D polygonal mesh surfaces using Geometric Algebra [88]. Currently D. Fontijne is primarily focusing on the efficient implementation of Geometric Algebra. He investigated the performance and elegance

of five models of 3D Euclidean geometry in a ray tracing application [27] and developed a code generator for Geometric Algebras [25]. There is a book with applications of Geometric Algebra edited by Dorst et al. [20]. A new book will be published soon [49].

At the **MPI Saarbruecken**, Germany, researchers are dealing with pose estimation and marker-less motion capture, e.g. Rosenhahn et al. [8], [7] and Kersting et al. [46].

The first time Geometric Algebra was introduced to a wider Computer Graphics audience, was probably at the SIGGRAPH conferences 2000 and 2001 (see [57]).

2.11 Conformal Geometric Algebra in robotics

Please find as follows research groups working with Conformal Geometric Algebra in the field of robotics.

Bayro-Corrochano et al. from **Guadalajara, Mexico** are primarily dealing with the application of Geometric Algebra in the field of robotics and computer vision. Some of their kinematics algorithms can be found in [11] for the 4D motor algebra and in the Conformal Geometric Algebra papers [13, 14] dealing with inverse kinematics, fixation and grasping as well as with kinematics and differential kinematics of binocular robot heads. Books from Bayro-Corrochano et al. with Geometric Algebra applications are for instance [10] and [12].

At the **University of Kiel**, Germany, Sommer et al. are applying Geometric Algebra to robot vision [77], e.g. Rosenhahn et al. concerning pose estimation [70, 71] and Sommer et al. regarding the twist representation of free-form objects [78]. Perwass et al. are applying Conformal Geometric Algebra to uncertain geometry with circles, spheres and conics [61], to geometry and kinematics with uncertain data [63] or concerning the inversion camera model [65] . There is a book with applications of Geometric Algebra edited by Sommer [76].

2.12 Inverse kinematics of a human-arm-like model

For the animation of humanoid models, inverse kinematics (IK) solutions are important as a basic building block for path planning. The standard model for arms (and also legs) is a seven degrees of freedom (DOF) kinematic chain.

The current standard tool for solving the inverse problem of mapping from a given end effector state to the configuration space $\{\theta_i\}$ is due to Tolani, Goswami, and Badler [80]. They also discuss in detail less favorable, optimization-based solutions. The importance of their algorithm in computer graphics and animation can be seen from the

large number of uses and citations of their work (just to give a few recent examples [74, 75, 9, 72]). Inverse kinematics in general is treated in standard text books like [73]. There are text books with a focus on human characters like [87].

Research dealing with motion of humanoids or virtual characters is often based on Clifford Algebra, especially on dual quaternions (Perez et al. [59], [58], [81]). We will see in chapter 3 that dual quaternions can be regarded as part of Conformal Geometric Algebra.

2.13 Tools for Conformal Geometric Algebra

This section describes the tools we are using for the development and the implementation of algorithms.

Due to its geometric intuitiveness Conformal Geometric Algebra can be advantageously supported by tools. There are packages for the symbolic computer algebra systems Maple [5, 6] and Mathematica [15], a package for the numerical mathematics program MatLab called GABLE [54], the C++ software library generator Gaigen [25], the C++ software library GluCat [50], the Java library Clados [18], GAP [89], CLUCalc [60], NKlein [24], Clifford [79], Gaigen 2 and a stand alone program called CLICAL [52], to name just a few.

We decided to use the software tool CLUCalc for the interactive and visual development. For the efficient implementation of the algorithms we use the code generator Gaigen 2 as well as the library Cliffordlib for Maple. The screenshot of figure 2.5 shows how the computations of section 1.1 can be performed in an interactive and visual way. With the help of the editor window you are able to easily edit your formulas and in the visualization window you are able to see the spheres and the circle as directly visualized results. Once developed and verified, the algorithms have to be implemented on the target platform. This can be done very efficiently using a code generator for Geometric Algebra computations or an approach based on Maple. In a nutshell, the development process based on these three tools is characterized by

- the interactive and visual development and
- the efficient implementation

of geometrically intuitive algorithms based on Conformal Geometric Algebra.

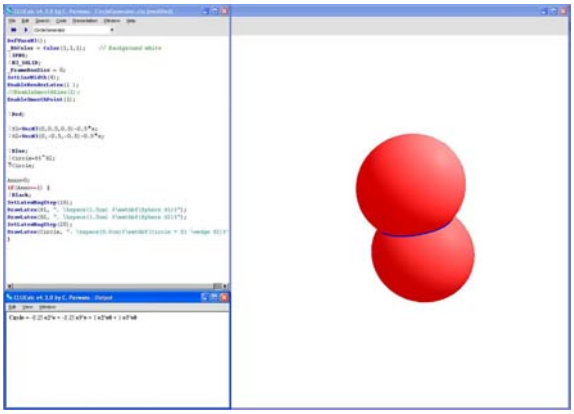

Figure 2.5: Interactive and visual development of algorithms

2.13.1 CLUCalc

We use the **CLUCalc** software to interactively **compute with Geometric Algebra** and to **visualize the results** of these computations. CLUCalc is freely available for download at [60]. With the help of the CLUCalc Software you are able to edit and run Scripts called **CLUScripts**. A screenshot of CLUCalc can be seen in figure 2.5. CLUCalc provides the following three windows

- editor window
- visualization window
- output window

There is almost a one to one correspondence between formulas and code as for example for the following computations of the example in figure 2.5.
The formulas

$$S_1 = P_1 - \frac{1}{2} r_1^2 e_\infty,$$
$$S_2 = P_2 - \frac{1}{2} r_2^2 e_\infty$$

and

$$z = S_1 \wedge S_2$$

are coded in CLUCalc as follows

```
:s1 = p1 - 0.5*r1*r1*einf;
:s2 = p2 - 0.5*r2*r2*einf;
:z = s1^s2;
```

Please find as follows a CLUScript describing motion according to section 2.9. In this example we rotate the blue sphere *Earth* around the yellow sphere *Sun* located at the origin.

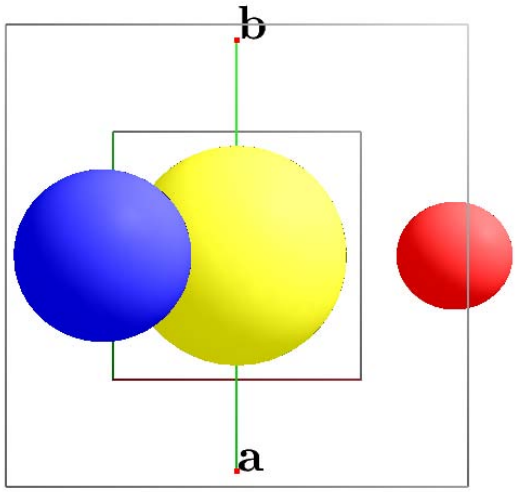

Figure 2.6: Visualization of a CLUScript describing motion

```
_DoAnimate = 1;
```

This script is animated (for details please refer to the Online help of the CLU software). The sphere *Earth* is continuously rotated according to a continuously changing angle. This angle is computed depending on the elapsed time.

```
DefVarsN3();
angle = ((Time * 45) % 360) * RadPerDeg;

SetMode(N3_IPNS, N3_SOLID);
:Red;
:a = VecN3(0,-2,0);
```

```
:b=VecN3(0,2,0);

:Green;
axis = *(a^b^einf);
:axis;
axis=axis/abs(axis);
?axis;
```

DefVarsN3(); indicates that we are working in the 5-dimensional conformal space N3.
:Red; means that the succeeding geometric objects will be drawn in red.
:a = VecN3(0,-2,0); assigns the 5-dimensional representation of a 3-dimensional point to the variable a according to table 2.5 on page 19. The leading colon means that this geometric object is not only computed, but also visualized.
With the help of **axis = *(a ∧ b ∧ einf);** a bivector representing a line $axis$ is computed.
According to table 2.5 on page 19 the dual representation of a line is the outer product of 2 points and e_∞, the point at infinity (indicated in CLUCalc by the predefined value **einf**).
The resulting bivector after dualization (indicated in CLUCalc by a leading "*") is normalized with the help of the *abs*-function, visualized and printed.

```
:Yellow;
:Sun = e0 -0.5*einf;

:Red;
:Earth =VecN3(2,0,0)-0.125*einf;

?R = exp(-angle/2*axis);

:Blue;
:R * Earth * ~R;
```

Sun is centered at the origin e_0 with radius $r = 1$ (see table 2.5 on page 19). It is drawn as a yellow sphere.
The red sphere is used as the basis sphere for the rotation of $Earth$. It is located out of the origin with half the radius of Sun.
The blue sphere representing the earth is rotated with the help of the product $REarth\tilde{R}$ (see section 2.9.1). The rotation operator depends on the fixed $axis$ and the continuously

changing *angle*. For details regarding CLUScript please refer to the CLUCalc online help [60].

The inverse kinematics algorithm, described in section 6.2, as well as the grasping algorithm of section 6.3 are implemented using CLUCalc. The corresponding CLUScripts can be downloaded from the homepage

`http://www.gris.informatik.tu-darmstadt.de/~dhilden/`

These example scripts show how easy it is to develop algorithms based on Conformal Geometric Algebra.

2.13.2 Gaigen 2

With the help of the Gaigen toolkit we are able to implement algorithms and integrate them inside our target platform. Gaigen is a C++ code generator for geometric algebras of different dimensions. Please refer to [25] for information and download.

If runtime efficiency is a big issue we are able to use the next generation Gaigen 2 toolkit using some additional techniques in order to optimize the resulting C++ code. The philosophy behind Gaigen 2 is based on two ideas : generative programming and specializing for the structure of geometric algebra. Gaigen 2 takes a succinct specification of a geometric algebra and transforms it into an implementation. The resulting implementation is very similar to what someone would program by hand and can be directly linked to an application.

In many types of programs, each variable does not need a linear combination of all the 32 blades of Conformal Geometric Algebra (see table 2.4 on page 17), but has a fixed 'specialized' multivector type. The inverse kinematics algorithms of this document for instance use variables with multivector types like *line* and *sphere*. If the Geometric Algebra implementation could work directly with these leaner specialized multivector types, performance would be greatly increased. As an implementation of this insight, Gaigen 2 allows the user to define specialized types along with the algebra specification and generates classes for each of them. These specialized multivector classes require much less storage than the generic multivector, but as a result, they are of course unable to store an arbitrary multivector type. For example, a *line* variable can not be stored in a *sphere* variable.

Please refer to chapter 7 for some results of an inverse kinematics application optimized based on Gaigen 2. The big advantage of this solution is that one can think in geometry, and directly program in geometric elements.

2.13.3 Maple with Cliffordlib

With the help of Maple we are able to compute symbolically [55] and to generate very efficient algorithms. In order to deal with the computation of geometric algebra, we use a library called Cliffordlib, developed by Rafal Ablamowicz and Bertfried Fauser. For download and installation hints please refer to [6, 83]. The most important operations of the Clifford package are presented in table 2.6. For the inner product we use the left contraction (LC) operation. Besides these main operations we also need some methods

Notation	Meaning
`a &c b`	geometric product
`a &w b`	outer product
`LC(a,b)`	inner product
`-(a) &c e12345`	dualization
`reversion()`	reversion

Table 2.6: Notation of Geometric Algebra operations in Maple

like `scalarpart()` or `vectorpart()` for extracting the scalar or the vector part of a multivector.

In order to use Conformal Geometric Algebra computations we have to load the Clifford package, set the metric of Clifford algebra, set aliases to basic blades (optional) and define e_0 and e_∞ as shown in the following Maple listing:

```
> with(Clifford);
> B:=linalg[diag](1, 1, 1, 1, -1);
> eval(makealiases(5, "ordered"));
> e0:=-0.5*e4+0.5*e5;
> einf:=e4+e5;
```

There is the possibility to write user specific functions like the following often needed functions for the computation of the conformal representation of a 3D point as well as for the computation of the dual.

```
> conformal := proc(x)
>   local conf;
>   global einf, e0;
>   conf := x + 1/2 * x &c x &c einf + e0;
> RETURN(conf);
> end:
```

```
> dual := proc(x)
>   local dual;
>   global e12345;
>   dual := - x &c e12345;
> RETURN(dual);
> end:
```

Please refer to chapter 3 for Maple computations according to the embedding of quaternions and other algebras. See chapter 7 for some results of an inverse kinematics application optimized based on Maple. The big advantage of this solution is that you are able to implement it with the help of standard compilers without the need of additional libraries.

Chapter 3

Embedding of quaternions and other algebras

Vector algebra (see table 2.2 on page 14) and projective geometry (see table 2.3 on page 16) can be regarded as part of Conformal Geometric Algebra.

It is also known that for instance quaternions are part of the 3D Euclidean Geometric Algebra. The goal of this chapter is to identify and analyze the embedding of some mathematical systems in Conformal Geometric Algebra and what their geometric meaning is. Besides the embedding of quaternions in Conformal Geometric Algebra (see the identification in table 3.2 on page 38) we will see the identification of other mathematical systems like complex numbers (see table 3.1 on page 36), Plücker coordinates (see table 3.3 on page 43) and dual quaternions (see table 3.5 on page 48).

3.1 Complex numbers

Complex numbers are linear combinations of scalars and the imaginary unit i squaring to -1. This imaginary unit can be identified in Conformal Geometric Algebra as one of the 2-blades $e_1 \wedge e_2$, $e_1 \wedge e_3$ and $e_2 \wedge e_3$, spanned by the three Euclidean basis vectors e_1, e_2, e_3. We select

$$i = e_3 \wedge e_2 = -e_2 \wedge e_3 \tag{3.1}$$

in order to be in line with the imaginary units of the quaternions as described in section 3.2.

$$i^2 = (e_3 \wedge e_2)^2 = e_3 e_2 \underbrace{e_3 e_2}_{-e_2 e_3} = -e_3 \underbrace{e_2 e_2}_{1} e_3 = -\underbrace{e_3 e_3}_{1} = -1 \tag{3.2}$$

Table 3.1: Complex numbers in Conformal Geometric Algebra

grade	**term**	**blades**	**nr.**
0	scalar	$\overbrace{\mathbf{1}}$	1
1	vector	$e_1, e_2, e_3, e_0, e_\infty$	5
2	bivector	$e_1 \wedge e_2,\ \ e_1 \wedge e_3,\ \ \overbrace{e_2 \wedge e_3}^{-\mathbf{i}},$ $e_1 \wedge e_\infty,\ \ e_2 \wedge e_\infty,\ \ e_3 \wedge e_\infty,$ $e_1 \wedge e_0,\ \ e_2 \wedge e_0,\ \ e_3 \wedge e_0,$ $e_0 \wedge e_\infty$	10
3	trivector	$e_1 \wedge e_2 \wedge e_3,\ \ e_1 \wedge e_2 \wedge e_0,\ \ e_1 \wedge e_2 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0,\ \ e_1 \wedge e_3 \wedge e_\infty,\ \ e_1 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0,\ \ e_2 \wedge e_3 \wedge e_\infty,\ \ e_2 \wedge e_0 \wedge e_\infty,$ $e_3 \wedge e_0 \wedge e_\infty$	10
4	quadvector	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty,$ $e_1 \wedge e_2 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_2 \wedge e_0 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	5
5	pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	1

From the 32 blades of Conformal Geometric Algebra only two blades are needed for complex numbers, namely the scalar and one bivector as indicated in table 3.1. From a geometric point of view the imaginary unit i is representing a line along the x-axis: Planes are represented according to equation (2.26) by their 3D normal vector and their distance to the origin. e_3 represents a plane through the origin in the direction of the x-axis and the y-axis. A line is represented as the intersection of two planes (see equation 2.30). Therefore $i = e_3 \wedge e_2$ represents the intersection of the two planes represented by its normal vectors e_3 and e_2. This results in the line along the x-axis.
Let us now investigate a rotation with angle ϕ around the line represented by the imaginary unit $i = e_3 \wedge e_2$. According to equation 2.36 this rotation can be described by the rotor R

$$R = e^{-\frac{\phi}{2} e_3 \wedge e_2}$$

With the help of the Taylor series we write

$$R = 1 + \frac{-e_3 \wedge e_2 \frac{\phi}{2}}{1!} + \frac{(-e_3 \wedge e_2 \frac{\phi}{2})^2}{2!} + \frac{(-e_3 \wedge e_2 \frac{\phi}{2})^3}{3!} + \frac{(-e_3 \wedge e_2 \frac{\phi}{2})^4}{4!} + \frac{(-e_3 \wedge e_2 \frac{\phi}{2})^5}{5!} + \frac{(-e_3 \wedge e_2 \frac{\phi}{2})^6}{6!} ...$$

or

$$R = 1 - \frac{e_3 \wedge e_2 \frac{\phi}{2}}{1!} + \frac{(e_3 \wedge e_2 \frac{\phi}{2})^2}{2!} - \frac{(e_3 \wedge e_2 \frac{\phi}{2})^3}{3!} + \frac{(e_3 \wedge e_2 \frac{\phi}{2})^4}{4!} - \frac{(e_3 \wedge e_2 \frac{\phi}{2})^5}{5!} + \frac{(e_3 \wedge e_2 \frac{\phi}{2})^6}{6!} ...$$

or according to equation (3.2)

$$R = 1 - \frac{(\frac{\phi}{2})^2}{2!} + \frac{(\frac{\phi}{2})^4}{4!} - \frac{(\frac{\phi}{2})^6}{6!} ..... - e_3 \wedge e_2 \frac{\frac{\phi}{2}}{1!} + e_3 \wedge e_2 \frac{(\frac{\phi}{2})^3}{3!} - e_3 \wedge e_2 \frac{(\frac{\phi}{2})^5}{5!} .....$$

and therefore

$$R = \cos(\frac{\phi}{2}) - e_3 \wedge e_2 \sin(\frac{\phi}{2}) \tag{3.3}$$

or

$$R = \cos(-\frac{\phi}{2}) + e_3 \wedge e_2 \sin(-\frac{\phi}{2}) \tag{3.4}$$

This means that a complex number identified by the imaginary unit $i = e_3 \wedge e_2$ represents a rotation around the x-axis. Identifying the imaginary unit with one of the other bivectors squaring to -1 would mean to rotate around the y-axis or the z-axis.

3.2 Quaternions

There is a lot of literature about quaternions represented in Euclidean Geometric Algebra (for instance [30], [48], [54], [64] and[43]). In this thesis, we will see how they are embedded in the Conformal Geometric Algebra in a very intuitive way. The main observation will be that an arbitrary line through the origin represents the rotation axis for a quaternion if we use the following definitions for the imaginary units

$$i = e_3 \wedge e_2, \tag{3.5}$$

$$j = e_1 \wedge e_3, \tag{3.6}$$

$$k = e_2 \wedge e_1. \tag{3.7}$$

In table 3.2 you can see the four blades for quaternions representing the scalar part and the three imaginary parts of quaternions.
Let $\mathbf{v}$ be an arbitrary *normalized Euclidean 3D vector*

$$\mathbf{v} = v_1 e_1 + v_2 e_2 + v_3 e_3. \tag{3.8}$$

Table 3.2: Quaternions in Conformal Geometric Algebra

grade	**term**	**blades**	**nr.**
0	scalar	$\overbrace{\mathbf{1}}$	1
1	vector	$e_1, e_2, e_3, e_0, e_\infty$	5
2	bivector	$\overbrace{\underbrace{e_1 \wedge e_2}_{-\mathbf{k}},\ \underbrace{e_1 \wedge e_3}_{\mathbf{j}},\ \underbrace{e_2 \wedge e_3}_{-\mathbf{i}}}$, $e_1 \wedge e_\infty,\ e_2 \wedge e_\infty,\ e_3 \wedge e_\infty,$ $e_1 \wedge e_0,\ e_2 \wedge e_0,\ e_3 \wedge e_0,$ $e_0 \wedge e_\infty$	10
3	trivector	$e_1 \wedge e_2 \wedge e_3,\ e_1 \wedge e_2 \wedge e_0,\ e_1 \wedge e_2 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0,\ e_1 \wedge e_3 \wedge e_\infty,\ e_1 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0,\ e_2 \wedge e_3 \wedge e_\infty,\ e_2 \wedge e_0 \wedge e_\infty,$ $e_3 \wedge e_0 \wedge e_\infty$	10
4	quadvector	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty,$ $e_1 \wedge e_2 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_2 \wedge e_0 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	5
5	pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	1

The conformal representation of the Euclidean point (v_1, v_2, v_3) according to table 2.5 on page 19 is

$$P = \mathbf{v} + \frac{1}{2}\mathbf{v}^2 e_\infty + e_o, \tag{3.9}$$

According to table 2.5 on page 19, the line through the origin e_o and the point a is described by their outer product with the point at infinity e_∞

$$L^* = e_o \wedge P \wedge e_\infty. \tag{3.10}$$

The dualization calculation leads to the standard representation of the line (see figure 3.1 with a screenshot of the Maple computations)

$$L = v_1(e_3 \wedge e_2) + v_2(e_1 \wedge e_3) + v_3(e_2 \wedge e_1). \tag{3.11}$$

or

$$L = v_1 i + v_2 j + v_3 k \tag{3.12}$$

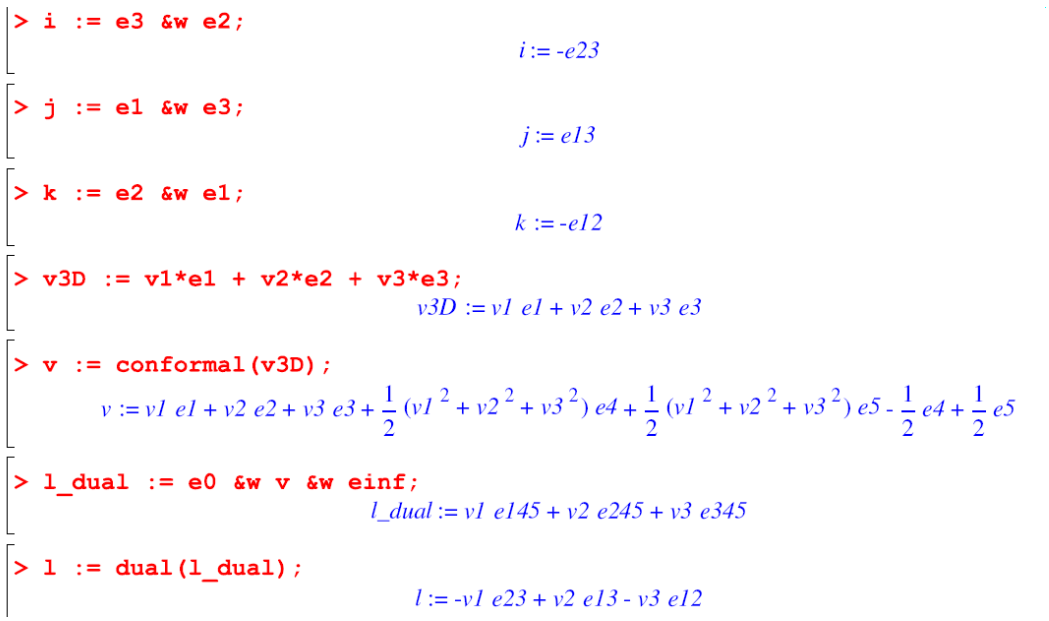

Figure 3.1: Quaternion computations in Maple

We will see in the following sections that a rotation around this axis L with an angle of ϕ can be computed as the following quaternion:

$$Q = \cos(\frac{\phi}{2}) + L\sin(\frac{\phi}{2}). \tag{3.13}$$

3.2.1 The imaginary units

For the imaginary units as defined in the equations (3.5), (3.6) and(3.7) we are able to derive the following properties :

$$i^2 = (e_3 \wedge e_2)^2 = e_3e_2\underbrace{e_3e_2}_{-e_2e_3} = -e_3\underbrace{e_2e_2}_{1}e_3 = -\underbrace{e_3e_3}_{1} = -1$$
$$j^2 = (e_1 \wedge e_3)^2 = e_1e_3\underbrace{e_1e_3}_{-e_3e_1} = -e_1\underbrace{e_3e_3}_{1}e_1 = -\underbrace{e_1e_1}_{1} = -1$$
$$k^2 = (-e_1 \wedge e_2)^2 = e_1e_2\underbrace{e_1e_2}_{-e_2e_1} = -e_1\underbrace{e_2e_2}_{1}e_1 = -\underbrace{e_1e_1}_{1} = -1$$

For the multiplication of i and j we get

$$ij = (e_3 \wedge e_2)(e_1 \wedge e_3) = e_3e_2e_1e_3 = e_2e_3e_3e_1 = e_2 \wedge e_1 = k$$

Accordingly

$$jk = i$$

$$ki = j$$

and

$$ijk = ii = -1$$

We recognize that the three imaginary units i, j, k are represented as the 3 axes in Conformal Geometric Algebra. As for the imaginary unit of section 3.1 i is representing the x-axis as well as j and k are representing the y-axis and the z-axis.

```
> q1_pure := v11*i +v12*j + v13*k;
                    q1_pure := -v11 e23 + v12 e13 - v13 e12

> q2_pure := v21*i +v22*j + v23*k;
                    q2_pure := -v21 e23 + v22 e13 - v23 e12

> Q_pure := q1_pure &c q2_pure;
   Q_pure := -(v11 v21 + v12 v22 + v13 v23) Id + (v13 v21 - v11 v23) e13 - (v12 v23 - v13 v22) e23
         + (v12 v21 - v11 v22) e12

> Test := cross(v11,v12,v13,v21,v22,v23) &c e123;
         Test := -(v13 v21 - v11 v23) e13 + (v12 v23 - v13 v22) e23 - (v12 v21 - v11 v22) e12

> Test2:= Q_pure+Test;
                    Test2 := -(v11 v21 + v12 v22 + v13 v23) Id

> q1 :=  s1 + q1_pure;
                    q1 := s1 - v11 e23 + v12 e13 - v13 e12
```

Figure 3.2: Quaternion product computations in Maple

3.2.2 Pure quaternions and their geometric product

A pure quaternion Q_1 has no scalar part. Its Conformal Geometric Algebra form is

$$Q_1 = v_{11}(e_3 \wedge e_2) + v_{12}(e_1 \wedge e_3) + v_{13}(e_2 \wedge e_1) \tag{3.14}$$

or using the definitions (3.5), (3.6) and (3.7)

$$Q_1 = v_{11}i + v_{12}j + v_{13}k \tag{3.15}$$

with (v_{11}, v_{12}, v_{13}) being a normalized 3D vector.

The geometric product of a pure quaternion Q_1 and a pure quaternion Q_2

$$Q_2 = v_{21}(e_3 \wedge e_2) + v_{22}(e_1 \wedge e_3) + v_{23}(e_2 \wedge e_1)$$

is according to the Maple evaluation of figure 3.2

$$Q_1 Q_2 = -(v_{11}v_{12} + v_{12}v_{22} + v_{13}v_{23}) \tag{3.16}$$

$$+(v_{12}v_{23} - v_{13}v_{22})(e_3 \wedge e_2) + (v_{13}v_{12} - v_{11}v_{23})(e_1 \wedge e_3) + (v_{11}v_{22} - v_{12}v_{12})(e_2 \wedge e_1)$$

which is equivalent to the product of quaternions

$$Q_1 Q_2 = -(v_{11}v_{12}+v_{12}v_{22}+v_{13}v_{23})+(v_{12}v_{23}-v_{13}v_{22})i+(v_{13}v_{12}-v_{11}v_{23})j+(v_{11}v_{22}-v_{12}v_{12})k \tag{3.17}$$

Note : The square of a pure quaternion therefore is

$$Q_1^2 = -(v_{11}v_{11} + v_{12}v_{12} + v_{13}v_{13}) = -1 \tag{3.18}$$

3.2.3 Rotations based on unit quaternions

Rotations based on quaternions are restricted to rotations with a rotation axis going through the origin. They can be defined as

$$Q = e^{\frac{\phi}{2}L} \tag{3.19}$$

with

$$L = v_1 i + v_2 j + v_3 k$$

representing a normalized line through the origin according to the Euclidean direction vector of equation (3.8).
This leads to the well-known definition of general quaternions

$$Q = \cos(\frac{\phi}{2}) + L\sin(\frac{\phi}{2}) \tag{3.20}$$

Note :
With the help of the Taylor series and the property $L^2 = -1$ (see equation (3.18))

$$Q = e^{\frac{\phi}{2}L}$$

$$= 1 + \frac{L\frac{\phi}{2}}{1!} + \frac{(L\frac{\phi}{2})^2}{2!} + \frac{(L\frac{\phi}{2})^3}{3!} + \frac{(L\frac{\phi}{2})^4}{4!} + \frac{(L\frac{\phi}{2})^5}{5!} + \frac{(L\frac{\phi}{2})^6}{6!}...$$

$$= 1 - \frac{(\frac{\phi}{2})^2}{2!} + \frac{(\frac{\phi}{2})^4}{4!} - \frac{(\frac{\phi}{2})^6}{6!}...$$
$$+L\frac{\frac{\phi}{2}}{1!} - L\frac{(\frac{\phi}{2})^3}{3!} + L\frac{(\frac{\phi}{2})^5}{5!}...$$
$$= \cos(\frac{\phi}{2}) + L\sin(\frac{\phi}{2})$$

Would you please notice that there is only a slight difference between the quaternion of equation (3.20) and the rotor of section 2.9.1. The sign difference indicates a rotation in different directions.

In Conformal Geometric Algebra we rotate an object o with the help of the operation

$$o_{rotated} = Qo\tilde{Q} \tag{3.21}$$

with $\tilde{Q}$ being the reverse of Q,

$$\tilde{Q} = \cos(\frac{\phi}{2}) - L\sin(\frac{\phi}{2}) \tag{3.22}$$

which is also indicated as **conjugate** of a quaternion.

Please note that for $\phi = \pi$ the quaternion Q

$$Q = v_1 i + v_2 j + v_3 k \tag{3.23}$$

represents a line through the origin and the 3D point represented by the normalized 3D vector (v_1, v_2, v_3) as well as a rotation with angle $\phi = \pi$ about this line. We will use this property advantageously in the inverse kinematics algorithm of chapter 7.

Table 3.3: Plücker coordinates in Conformal Geometric Algebra

grade	**term**	**blades**	**nr.**
0	scalar	1	1
1	vector	$e_1, e_2, e_3, e_0, e_\infty$	5
2	bivector	$\overbrace{\mathbf{e_1} \wedge \mathbf{e_2},\ \mathbf{e_1} \wedge \mathbf{e_3},\ \mathbf{e_2} \wedge \mathbf{e_3}}$, $\overbrace{\mathbf{e_1} \wedge \mathbf{e}_\infty,\ \mathbf{e_2} \wedge \mathbf{e}_\infty,\ \mathbf{e_3} \wedge \mathbf{e}_\infty}$, $e_1 \wedge e_0,\ e_2 \wedge e_0,\ e_3 \wedge e_0,$ $e_0 \wedge e_\infty$	10
3	trivector	$e_1 \wedge e_2 \wedge e_3,\ e_1 \wedge e_2 \wedge e_0,\ e_1 \wedge e_2 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0,\ e_1 \wedge e_3 \wedge e_\infty,\ e_1 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0,\ e_2 \wedge e_3 \wedge e_\infty,\ e_2 \wedge e_0 \wedge e_\infty,$ $e_3 \wedge e_0 \wedge e_\infty$	10
4	quadvector	$e_1 \wedge e_2 \wedge e_3 \wedge e_\infty,$ $e_1 \wedge e_2 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_2 \wedge e_0 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	5
5	pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	1

3.3 Plücker coordinates

We will see in this section that Plücker coordinates can be identified in Conformal Geometric Algebra based on the six 2-blades indicated in table 3.3.
Since Plücker coordinates describe an arbitrary line we first of all compute the line representation. According to the Maple program of figure 3.3 this results in

$$L = -(a_3 - b_3)e_1 \wedge e_2 + (a_2 - b_2)e_1 \wedge e_3 - (a_1 - b_1)e_2 \wedge e_3 \tag{3.24}$$
$$+(a_2b_3 - a_3b_2)e_1 \wedge e_\infty - (a_1b_3 - a_3b_1)e_2 \wedge e_\infty + (a_1b_2 - a_2b_1)e_3 \wedge e_\infty$$

Another form is

$$L = (a_1 - b_1)i + (a_2 - b_2)j + (a_3 - b_3)k \tag{3.25}$$
$$+(a_2b_3 - a_3b_2)e_1 \wedge e_\infty - (a_1b_3 - a_3b_1)e_2 \wedge e_\infty + (a_1b_2 - a_2b_1)e_3 \wedge e_\infty$$

or

$$L = (\mathbf{b} - \mathbf{a})e_{123} + [\mathbf{a} \times \mathbf{b}] \wedge e_\infty \tag{3.26}$$

```
a := a1 * e1 + a2*e2 + a3 * e3;
```
$a := a1\ e1 + a2\ e2 + a3\ e3$

```
b := b1 * e1 + b2*e2 + b3 * e3;
```
$b := b1\ e1 + b2\ e2 + b3\ e3$

```
Line := dual ( conformal(b) &w conformal(a) &w einf);
```
$Line := (a2 - b2)\ e13 - (a1 - b1)\ e23 - (a3 - b3)\ e12 - (-b1\ a3 + b3\ a1)\ e24 + (b2\ a1 - b1\ a2)\ e34 + (-b2\ a3 + b3\ a2)\ e15 + (-b2\ a3 + b3\ a2)\ e14 - (-b1\ a3 + b3\ a1)\ e25 + (b2\ a1 - b1\ a2)\ e35$

```
L := (b-a) &c e123 + cross (a1,a2,a3,b1,b2,b3) &w einf;
```
$L := (a2 - b2)\ e13 - (a1 - b1)\ e23 - (a3 - b3)\ e12 + (-b2\ a3 + b3\ a2)\ e14 + (b1\ a3 - b3\ a1)\ e24 + (b2\ a1 - b1\ a2)\ e34 + (-b2\ a3 + b3\ a2)\ e15 + (b1\ a3 - b3\ a1)\ e25 + (b2\ a1 - b1\ a2)\ e35$

Figure 3.3: Plücker computations in Maple

resulting in

$$L = \mathbf{u}e_{123} + \mathbf{m} \wedge e_\infty \tag{3.27}$$

with $\mathbf{u} = \mathbf{b} - \mathbf{a}$ as Euclidean direction vector and $\mathbf{m} = \mathbf{a} \times \mathbf{b}$ as the moment vector. The corresponding six Plücker coordinates are

$$(\mathbf{u} : \mathbf{m}) = (u_1 : u_2 : u_3 : m_1 : m_2 : m_3). \tag{3.28}$$

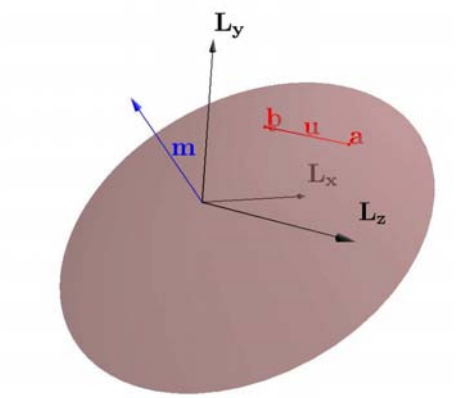

Figure 3.4: Plücker coordinates

Would you please notice that based on the above equations the pair $(\mathbf{u} : \mathbf{m})$ uniquely determines the line L up to a common (nonzero) scalar multiple which depends on the distance between $\mathbf{a}$ and $\mathbf{b}$. That is, the coordinates may be considered homogenous coordinates for L, in the sense that all pairs $(\lambda\mathbf{u} : \lambda\mathbf{m})$, for $\lambda \neq 0$ can be produced by points on L and only L, and any such pair determines a unique line as long as $\mathbf{u}$ is not zero and $\mathbf{u} \cdot \mathbf{m} = 0$ (see [2]).

The cross product $\mathbf{m} = \mathbf{a} \times \mathbf{b}$ can also be written as $\mathbf{m} = -(\mathbf{a} \wedge \mathbf{b})e_{123}$ according to equation (2.5).
$\mathbf{u}e_{123}$ equals to $u_1e_{23} - u_2e_{13} + u_3e_{12}$ since

$$(u_1e_1 + u_2e_2 + u_3e_3)e_1e_2e_3$$

$$= u_1 \underbrace{e_1e_1}_{1} e_2e_3 + u_2 \underbrace{e_2e_1}_{-e_1e_2} e_2e_3 + u_3 \underbrace{e_3e_1}_{-e_1e_3} \underbrace{e_2e_3}_{-e_3e_2}$$

$$= u_1e_2e_3 - u_2e_1e_2e_2e_3 + u_3e_1e_3e_3e_2$$

$$= u_1e_2e_3 - u_2e_1e_3 + u_3e_1e_2$$

According to the Maple computations of figure 3.5

```
> a := a1 * e1 + a2*e2 + a3 * e3;
                        a := a1 e1 + a2 e2 + a3 e3

> b := b1 * e1 + b2*e2 + b3 * e3;
                        b := b1 e1 + b2 e2 + b3 e3

> Line := dual ( conformal(b) &w conformal(a) &w einf);
Line := (a2 - b2) e13 - (a1 - b1) e23 - (a3 - b3) e12 - (-a3 b1 + a1 b3) e24 + (-a2 b1 + a1 b2) e34 + (-a3 b2 + a2 b3) e15 + (-a3 b2 + a2 b3) e14 - (-a3 b1 + a1 b3) e25
     + (-a2 b1 + a1 b2) e35

> L0 := (b-a) &c e123;
                        L0 := (a2 - b2) e13 - (a1 - b1) e23 - (a3 - b3) e12

> P:=conformal(a);
                        P := a1 e1 + a2 e2 + a3 e3 + 1/2 (a1^2 + a2^2 + a3^2) e4 + 1/2 (a1^2 + a2^2 + a3^2) e5 - 1/2 e4 + 1/2 e5

> pi_m := dual(P &w dual(L0));
                        pi_m := (-a3 b2 + a2 b3) e1 - (-a3 b1 + a1 b3) e2 + (-a2 b1 + a1 b2) e3

> NewLine := L0 + pi_m &w einf;
NewLine := (a2 - b2) e13 - (a1 - b1) e23 - (a3 - b3) e12 + (-a3 b2 + a2 b3) e14 + (a3 b1 - a1 b3) e24 + (-a2 b1 + a1 b2) e34 + (-a3 b2 + a2 b3) e15 + (a3 b1 - a1 b3) e25
     + (-a2 b1 + a1 b2) e35
```

Figure 3.5: Plücker computations in Maple

a line can also be described as

$$L = L_0 + \pi_m \wedge e_\infty \tag{3.29}$$

with L_0 as a parallel line through the origin and the moment plane

$$\pi_m = (P \wedge L_0^*)^* \tag{3.30}$$

spanned by L_0 and an arbitrary conformal point P on the line.

Example:
For the two Euclidean points $\mathbf{a} = (1, 0, 1)$ and $\mathbf{b} = (1, 0, 0)$ we get

$$\mathbf{u} = -e_3$$

and

$$\mathbf{m} = e_2$$

leading to the line

$$L = -e_3 e_{123} + e_2 \wedge e_\infty = -e_{12} + e_2 \wedge e_\infty \tag{3.31}$$

with the six Plücker coordinates (0 : 0 : -1 : 0 : 1 : 0)

Table 3.4: Dual numbers in Conformal Geometric Algebra

grade	**term**	**blades**	**nr.**
0	scalar	$\overbrace{\mathbf{1}}$	1
1	vector	$e_1, e_2, e_3, e_0, e_\infty$	5
2	bivector	$e_1 \wedge e_2,\ e_1 \wedge e_3,\ e_2 \wedge e_3,$ $e_1 \wedge e_\infty,\ e_2 \wedge e_\infty,\ e_3 \wedge e_\infty,$ $e_1 \wedge e_0,\ e_2 \wedge e_0,\ e_3 \wedge e_0,$ $e_0 \wedge e_\infty$	10
3	trivector	$e_1 \wedge e_2 \wedge e_3,\ e_1 \wedge e_2 \wedge e_0,\ e_1 \wedge e_2 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0,\ e_1 \wedge e_3 \wedge e_\infty,\ e_1 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0,\ e_2 \wedge e_3 \wedge e_\infty,\ e_2 \wedge e_0 \wedge e_\infty,$ $e_3 \wedge e_0 \wedge e_\infty$	10
4	quadvector	$\overbrace{\mathbf{e_1 \wedge e_2 \wedge e_3 \wedge e_\infty}}^{\epsilon},$ $e_1 \wedge e_2 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_2 \wedge e_0 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0 \wedge \epsilon,$ $e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	5
5	pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	1

3.4 Dual numbers

A dual number is a number $x + \epsilon y$, whereby x,y are scalars and ϵ is a unit with the property that $\epsilon^2 = 0$. In Conformal Geometric Algebra there are a lot of blades with this property of squaring to zero, since $e_0^2 = e_\infty^2 = 0$. We use

$$\epsilon = e_1 \wedge e_2 \wedge e_3 \wedge e_\infty \tag{3.32}$$

as used in [45] that fulfills this property for dual numbers as well as for dual quaternions (see section 3.5). This is why we can write a dual number in Conformal Geometric Algebra as follows

$$x + \epsilon y = x + e_1 \wedge e_2 \wedge e_3 \wedge e_\infty y \tag{3.33}$$

Table 3.4 shows the subset of the 32 blades of Conformal Geometric algebra needed for dual numbers, namely the scalar and ϵ.

Table 3.5: Dual quaternions in Conformal Geometric Algebra

grade	**term**	**blades**	**nr.**
0	scalar	$\overbrace{\mathbf{1}}$	1
1	vector	$e_1, e_2, e_3, e_0, e_\infty$	5
2	bivector	$\overbrace{\underbrace{e_1 \wedge e_2}_{-\mathbf{k}},\ \underbrace{e_1 \wedge e_3}_{\mathbf{j}},\ \underbrace{e_2 \wedge e_3}_{-\mathbf{i}}}$, $\overbrace{\mathbf{e_1} \wedge \mathbf{e}_\infty,\ \mathbf{e_2} \wedge \mathbf{e}_\infty,\ \mathbf{e_3} \wedge \mathbf{e}_\infty}$, $e_1 \wedge e_0,\ e_2 \wedge e_0,\ e_3 \wedge e_0,$ $e_0 \wedge e_\infty$	10
3	trivector	$e_1 \wedge e_2 \wedge e_3,\ e_1 \wedge e_2 \wedge e_0,\ e_1 \wedge e_2 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0,\ e_1 \wedge e_3 \wedge e_\infty,\ e_1 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0,\ e_2 \wedge e_3 \wedge e_\infty,\ e_2 \wedge e_0 \wedge e_\infty,$ $e_3 \wedge e_0 \wedge e_\infty$	10
4	quadvector	$\overbrace{\mathbf{e_1} \wedge \mathbf{e_2} \wedge \mathbf{e_3} \wedge \mathbf{e}_\infty}^{\epsilon}$, $e_1 \wedge e_2 \wedge e_3 \wedge e_0,$ $e_1 \wedge e_2 \wedge e_0 \wedge e_\infty,$ $e_1 \wedge e_3 \wedge e_0 \wedge e_\infty,$ $e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	5
5	pseudoscalar	$e_1 \wedge e_2 \wedge e_3 \wedge e_0 \wedge e_\infty$	1

3.5 Dual quaternions

A dual quaternion is defined by

$$Q = Q_1 + \epsilon Q_2 \tag{3.34}$$

with the quaternions

$$Q_i = s_i + v_{i1} i + v_{i2} j + v_{i3} k \tag{3.35}$$

and ϵ is a unit with the property that $\epsilon^2 = 0$.

We will see in this section that dual quaternions - as often used in robotics - can be identified in Conformal Geometric Algebra based on the six 2-blades used for Plücker coordinates as well as on scalars and on one 4-blade ($\epsilon = e_1 e_2 e_3 e_\infty$) as indicated in table 3.5.

We will also see in the following that there is an geometrically intuitive relation between quaternions and a versor describing for instance a rotation around an arbitrary rotation axis.

The quaternions can be written in Conformal Geometric Algebra form as

$$Q_i = s_i + v_{i1}(e_3 \wedge e_2) + v_{i2}(e_1 \wedge e_3) + v_{i3}(e_2 \wedge e_1) \tag{3.36}$$

With equation (3.32) we can write a dual quaternion in Conformal Geometric Algebra

```
> epsilon := e1 &w e2 &w e3 &w einf;
                                    ε := e1234 + e1235

> Q1 := s1 + v11*i+v12*j +v13*k;
                          Q1 := s1 - v11 e23 + v12 e13 - v13 e12

> Q2 := s2 + v21*i+v22*j +v23*k;
                          Q2 := s2 - v21 e23 + v22 e13 - v23 e12

> Q := Q1 + epsilon &c Q2;
   Q := s1 - v11 e23 + v12 e13 - v13 e12 + v22 e24 + s2 e1234 + s2 e1235 + v23 e34 + v21 e15 + v21 e14
        + v22 e25 + v23 e35

> QForm := s1 - v11*e23+v12*e13-v13*e12+s2*(e123 &w einf)+v22*(e2 &w
  einf)+v23*(e3 &w einf)+v21*(e1 &w einf);
    QForm := s1 - v11 e23 + v12 e13 - v13 e12 + s2 (e1234 + e1235) + v22 (e24 + e25) + v23 (e34 + e35)
         + v21 (e14 + e15)
```

Figure 3.6: Dual quaternion computations in Maple

as follows

$$Q = Q_1 + \epsilon Q_2 = Q_1 + e_1 \wedge e_2 \wedge e_3 \wedge e_\infty Q_2 \tag{3.37}$$

This can be written in the form of a linear combination of the above mentioned 8 blades.

$$Q = s_1 - v_{11}(e_2 \wedge e_3) + v_{12}(e_1 \wedge e_3) - v_{13}(e_1 \wedge e_2) \tag{3.38}$$

$$+s_2(e_1 \wedge e_2 \wedge e_3 \wedge e_\infty) + v_{21}(e_1 \wedge e_\infty) + v_{22}(e_2 \wedge e_\infty) + v_{23}(e_3 \wedge e_\infty)$$

We will see now that there is a geometric relation between dual quaternions and a versor describing a rotation with arbitrary rotation axis. This operation is especially helpful to describe the movement of robot at a revolute joint.

An arbitrary rotation according to the quaternion Q_1 can be described based on a translation to the origin, the rotation Q_1 at the origin and a translation back as expressed in the versor product

$$TQ_1\tilde{T}. \tag{3.39}$$

```
> t := t1 * e1 + t2*e2 + t3*e3;
                    t := t1 e1 + t2 e2 + t3 e3

> T := 1 - 0.5 * t &c einf;
        T := 1 - 0.5 t2 e24 - 0.5 t3 e34 - 0.5 t1 e15 - 0.5 t1 e14 - 0.5 t2 e25 - 0.5 t3 e35

> T_rev := 1 + 0.5 * t &c einf;
      T_rev := 1 + 0.5 t2 e24 + 0.5 t3 e34 + 0.5 t1 e15 + 0.5 t1 e14 + 0.5 t2 e25 + 0.5 t3 e35

> Versor := T &c Q1 &c T_rev;
  Versor := s1 Id + v12 e13 - v11 e23 - v13 e12 + (-1. t3 v11 + t1 v13) e24 - 1. (-1. t2 v11 + t1 v12) e34
        - 1. (t2 v13 - 1. t3 v12) e15 - 1. (t2 v13 - 1. t3 v12) e14 + (-1. t3 v11 + t1 v13) e25
        - 1. (-1. t2 v11 + t1 v12) e35

> Test := simplify(Versor - cross(v11,v12,v13,t1,t2,t3)  &w einf);
                    Test := s1 Id + v12 e13 - 1. v11 e23 - 1. v13 e12
```

Figure 3.7: Dual quaternion computations in Maple

This equals to

$$V = s_1 - v_{11}(e_2 \wedge e_3) + v_{12}(e_1 \wedge e_3) - v_{13}(e_1 \wedge e_2) \tag{3.40}$$
$$+(-t_2 v_{13} + t_3 v_{12})(e_1 \wedge e_\infty) + (t_1 v_{13} - t_3 v_{11})(e_2 \wedge e_\infty) + (t_2 v_{11} - t_1 v_{12})(e_3 \wedge e_\infty)$$

This means that a relation between versors and dual quaternions is

$$s_2 = 0 \tag{3.41}$$

$$v_{21} = -t_2 v_{13} + t_3 v_{12} \tag{3.42}$$

$$v_{22} = t_1 v_{13} - t_3 v_{11} \tag{3.43}$$

$$v_{23} = t_2 v_{11} - t_1 v_{12} \tag{3.44}$$

In a nutshell, there is the following relation between dual quaternions and the versor $TQ_1\tilde{T}$:

$$TQ_1\tilde{T} = Q_1 + \epsilon Q_2 \tag{3.45}$$

with the cross product

$$\mathbf{v_2} = \mathbf{v_1} \times \mathbf{t}$$

Please note that a normalization according to $\mathbf{v_2}$ is needed in order to handle correct quaternions.

Chapter 4

The role of infinity

A key role for a deeper understanding of the basic entities of Conformal Geometric Algebra plays infinity. Here we see how a plane is created by infinitely increasing the radius of a sphere, how infinity is represented in Conformal Geometric Algebra and how translators are a limit of rotors.

4.1 Infinity

In section 2.20 we could see that the conformal basis vector e_0 represents the origin of the Euclidean space (see equation 2.22).
But, what about e_∞ ? We will see in this section that e_∞ can be interpreted either as a sphere with infinite radius or as a point or a plane at infinity.

4.1.1 Sphere with infinite radius

Let us at first look at a sphere with center point at the origin. According to equation 2.24 this origin sphere ($p = e_0$) is represented as

$$S = -\frac{1}{2}r^2 e_\infty + e_0 \tag{4.1}$$

Another homogenous representation of this origin sphere is for instance its product with the scalar $-\frac{2}{r^2}$

$$S' = -\frac{2}{r^2}S = e_\infty - \frac{2}{r^2}e_0 \tag{4.2}$$

Based on this formula and on the fact that S and S' are representing the same sphere we can easily see that an origin sphere with infinite radius is represented by e_∞

$$\lim_{r\to\infty} S' = e_\infty$$

It can be shown that this is true not only for an origin sphere but also for a sphere with an arbitrary center point.

4.1.2 Point at infinity

Let us now assume an arbitrary Euclidean point $\mathbf{x}$ (not equal to the origin) represented by the conformal vector P

$$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$$

with the Euclidean normal vector $\mathbf{n}$ in the direction of $\mathbf{x}$

$$\mathbf{x} = t\mathbf{n},\ t > 0,\ \mathbf{n}^2 = 1$$

Another homogenous representation of this point is for instance its product with the

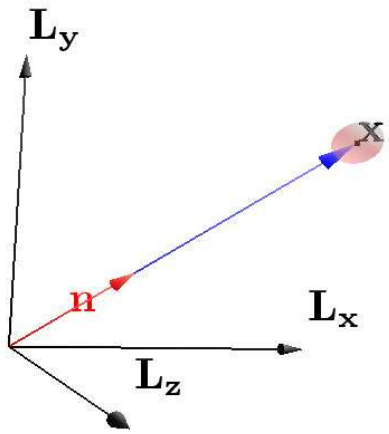

Figure 4.1: The point at infinity

scalar $\frac{2}{\mathbf{x}^2}$

$$P' = \frac{2}{\mathbf{x}^2}(\mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0)$$

$$P' = \frac{2}{\mathbf{x}^2}\mathbf{x} + e_\infty + \frac{2}{\mathbf{x}^2}e_0$$

We use that form in order to compute the limit $\lim_{t\to\infty} P'$ for increasing $\mathbf{x}$. Since $\mathbf{x} = t\mathbf{n}$ we get

$$P' = \frac{2}{t^2\mathbf{n}^2}t\mathbf{n} + e_\infty + \frac{2}{t^2\mathbf{n}^2}e_0$$

and since $\mathbf{n}^2 = 1$

$$P' = \frac{2}{t}\mathbf{n} + e_\infty + \frac{2}{t^2}e_0$$

Based on this formula and on the fact that P and P' are representing the same Euclidean point we can easily see that the point at infinity for each direction vector $\mathbf{n}$ is represented by e_∞

$$\lim_{t\to\infty} P' = e_\infty$$

4.1.3 Plane with infinite distance to the origin

Let us consider a plane with an arbitrary distance $d \neq 0$ to the origin. According to equation 2.26 this plane is represented as

$$\pi = \mathbf{n} + de_\infty \tag{4.3}$$

with the 3D normal vector $\mathbf{n}$. Another homogenous representation of this plane is for instance its product with the scalar $\frac{1}{d}$

$$\pi' = \frac{1}{d}\mathbf{n} + e_\infty \tag{4.4}$$

Based on this formula and on the fact that π and π' are representing the same plane we can easily see that a plane with infinite distance to the origin is represented by e_∞

$$\lim_{d\to\infty} \pi' = e_\infty$$

4.2 Planes as a limit of spheres

Spheres and planes, both, are vectors in Conformal Geometric Algebra. In this section, we will see how a sphere S

$$S = \mathbf{s} + \frac{1}{2}(\mathbf{s}^2 - r^2)e_\infty + e_0 \tag{4.5}$$

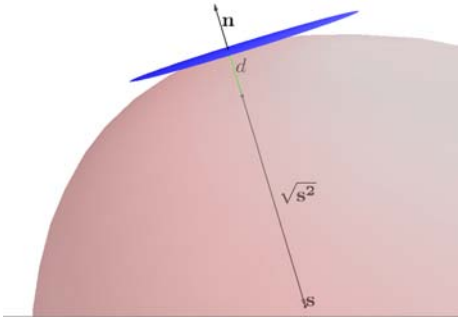

Figure 4.2: Spheres and planes

with Euclidean center point $\mathbf{s}$ and radius r degenerates to a plane as the result of a limit process.
Depending on whether the origin lies inside or outside of the sphere the minimum distance from the origin to the sphere is

$$d = r \pm \sqrt{\mathbf{s}^2} \tag{4.6}$$

If the origin lies inside of the sphere

$$r = \sqrt{\mathbf{s}^2} + d \tag{4.7}$$

or

$$r^2 = \mathbf{s}^2 + 2d\sqrt{\mathbf{s}^2} + d^2 \tag{4.8}$$

and the sphere can be written as

$$S = \mathbf{s} + \frac{1}{2}(\mathbf{s}^2 - \mathbf{s}^2 - 2d\sqrt{\mathbf{s}^2} - d^2)e_\infty + e_0 \tag{4.9}$$

or

$$S = \mathbf{s} + \frac{1}{2}(-2d\sqrt{\mathbf{s}^2} - d^2)e_\infty + e_0 \tag{4.10}$$

With $\mathbf{s} = -t\mathbf{n}$ and $\mathbf{s}^2 = t^2\mathbf{n}^2$ we get

$$S = -t\mathbf{n} - \frac{1}{2}(2td\sqrt{\mathbf{n}^2} + d^2)e_\infty + e_0 \tag{4.11}$$

or

$$-\frac{S}{t} = \mathbf{n} + \frac{1}{2}(2d\sqrt{\mathbf{n}^2} + \frac{d^2}{t})e_\infty - \frac{e_0}{t} \tag{4.12}$$

$$\lim_{t\to\infty} -\frac{S}{t} = \mathbf{n} + \lim_{t\to\infty} \frac{1}{2}(2d + \frac{d^2}{t})e_\infty - \lim_{t\to\infty} \frac{e_0}{t} \tag{4.13}$$

$$\lim_{t\to\infty} -\frac{S}{t} = \mathbf{n} + de_\infty \tag{4.14}$$

which is a representation of a plane.

4.3 Translators as a limit of rotors

Here, we will see as an example how a specific translator results from a limit process of a specific rotor. We derive the rotor from the line of equation (3.31) resulting from the Plücker coordinates based on the two Euclidean points $\mathbf{a} = e_1 + e_3$ and $\mathbf{b} = e_1$. This

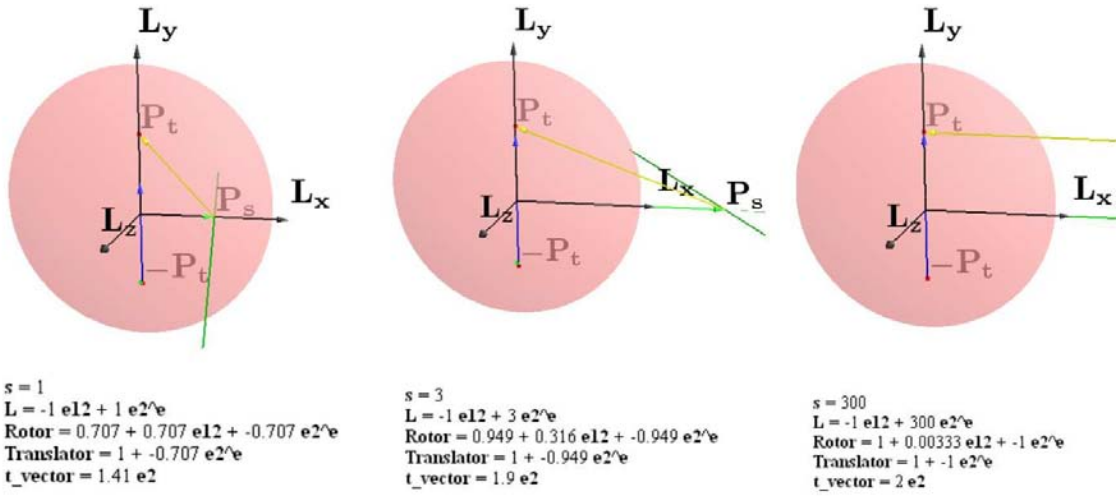

Figure 4.3: From Rotation to Translation

line is equipped with a parameter s as follows

$$L = -e_1 \wedge e_2 + se_2 \wedge e_\infty \tag{4.15}$$

in order to translate it with increasing s in x-direction.

Let us consider a rotation with angle α around this line L from point $-P_t$ to point P_t (see figure 4.3). Based on this rotation, described by the following rotor (see section 2.9.1)

$$R = \cos\frac{\alpha}{2} - \sin\frac{\alpha}{2}L \tag{4.16}$$

we will compute the translation from point $-P_t$ to the point P_t with parameter $t > 0$ translating P_t from the origin in y-direction. The sin and cos of half the angle can be expressed as follows

$$\sin\frac{\alpha}{2} = \frac{t}{\sqrt{t^2+s^2}} = \sqrt{\frac{1}{1+(\frac{s}{t})^2}} \tag{4.17}$$

$$\cos\frac{\alpha}{2} = \frac{s}{\sqrt{t^2+s^2}} = \sqrt{\frac{1}{1+(\frac{t}{s})^2}} \tag{4.18}$$

since the distance between e_0 and P_s is s and the distance between e_0 and P_t is t. Based on these observations we are able to describe the rotation as follows

$$R = \sqrt{\frac{1}{1+(\frac{t}{s})^2}} - \sqrt{\frac{1}{1+(\frac{s}{t})^2}}(-e_1 \wedge e_2 + se_2 \wedge e_\infty) \tag{4.19}$$

or

$$R = \sqrt{\frac{1}{1+(\frac{t}{s})^2}} + \sqrt{\frac{1}{1+(\frac{s}{t})^2}}(e_1 \wedge e_2) - \sqrt{\frac{s^2}{1+(\frac{s}{t})^2}}(e_2 \wedge e_\infty) \tag{4.20}$$

or

$$R = \sqrt{\frac{1}{1+(\frac{t}{s})^2}} + \sqrt{\frac{1}{1+(\frac{s}{t})^2}}(e_1 \wedge e_2) - \sqrt{\frac{1}{\frac{1}{s^2}+\frac{1}{t^2}}}(e_2 \wedge e_\infty) \tag{4.21}$$

or

$$R = \sqrt{\frac{1}{1+(\frac{t}{s})^2}} + \sqrt{\frac{1}{1+(\frac{s}{t})^2}}(e_1 \wedge e_2) - t\sqrt{\frac{1}{1+(\frac{t}{s})^2}}(e_2 \wedge e_\infty) \tag{4.22}$$

For $s \to \infty$ the resulting translator is

$$T = \lim_{s\to\infty} R = 1 - t(e_2 \wedge e_\infty) = 1 - te_2 \wedge e_\infty \tag{4.23}$$

According to equation 2.39 this corresponds to a translation with the translation vector

$$\mathbf{t} = 2te_2 \tag{4.24}$$

that we would expect for a translation from point $-P_t$ to point P_t. Figure 4.3 visualizes this limit process for increasing parameter s. While the green line L is moving to infinity, the translation vector approaches more and more to $2e_2$ (for $t = 1$).

Chapter 5

Approximation algorithms based on Conformal Geometric Algebra

In Conformal Geometric Algebra points, planes and spheres are represented as vectors. Table 5.1 summarizes and details results of chapter 2. This kind of representation makes

Table 5.1: Geometric meaning of the conformal vectors

Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	$= x_1 e_1 + x_2 e_2 + x_3 e_3 + (x_1^2 + x_2^2 + x_3^2)e_\infty + e_0$
Sphere	$S = P - \frac{1}{2}r^2 e_\infty$	$= s_1 e_1 + s_2 e_2 + s_3 e_3 + \frac{1}{2}(s_1^2 + s_2^2 + s_3^2 - r^2)e_\infty + e_0$
Plane	$\pi = \mathbf{n} + d e_\infty$	$= n_1 e_1 + n_2 e_2 + n_3 e_3 + d e_\infty$

it easy to perform tasks like grasping of objects (see for instance [91]) or ray tracing ([27]). In this chapter we solve the approximation of points with the help of a sphere or a plane. For this application we need at first an investigation of the inner product of conformal vectors and its geometric meaning. Table 5.2 summarizes the geometric

Table 5.2: The geometric meaning of the inner product of conformal vectors U and V

$U \cdot V$	**plane**	**sphere**	**point**
plane	angle between planes	Euclidean distance to center	Euclidean distance
sphere	Euclidean distance to center	distance measure	distance measure
point	Euclidean distance	distance measure	Euclidean distance

meaning of the inner product of conformal vectors U and V as outlined in the section 5.1 for distances and in section 6.1 for angles.

5.1 The inner product and distances

The inner product of vectors in Conformal Geometric Algebra results in a scalar and can be used as a measure for distances between basic objects. In this section, we will see, that the inner product $P \cdot S$ of two vectors P and S can be used for tasks like

- the Euclidean distance between two points
- the distance between one point and one plane
- the decision whether a point is inside or outside of a sphere.

5.1.1 Vectors in Conformal Geometric Algebra

A vector in Conformal Geometric Algebra can be written as

$$V = v_1 e_1 + v_2 e_2 + v_3 e_3 + v_4 e_\infty + v_5 e_0 \tag{5.1}$$

The meaning of the two additional coordinates e_0 and e_∞ is as follows :

	$\mathbf{v_5 = 0}$	$\mathbf{v_5 \neq 0}$
$\mathbf{v_4 = 0}$	plane through origin	sphere/point through origin
$\mathbf{v_4 \neq 0}$	plane	sphere/point

The multiplication with a constant $k \neq 0$ leads always to the same geometric object (same as for homogenous objects in projective space).
Division by $v_5 \neq 0$ leads to

$$S = s_1 e_1 + s_2 e_2 + s_3 e_3 + s_4 e_\infty + e_0 \tag{5.2}$$

representing a sphere S with center point $\mathbf{s}$ and radius r

$$S = \mathbf{s} + s_4 e_\infty + e_0 \tag{5.3}$$

with

$$s_4 = \frac{1}{2}(\mathbf{s}^2 - r^2) = \frac{1}{2}(s_1^2 + s_2^2 + s_3^2 - r^2)$$

Points are degenerate spheres with radius $r = 0$.

$$P = \mathbf{s} + \frac{1}{2}\mathbf{s}^2 e_\infty + e_0 \tag{5.4}$$

Note : when inserting this formula in equation (2.24) the result corresponds to the equation (5.3).
Planes are degenerate spheres with infinite radius. They are represented as a vector with $v_5 = 0$

$$V = v_1 e_1 + v_2 e_2 + v_3 e_3 + v_4 e_\infty = \mathbf{v} + v_4 e_\infty \tag{5.5}$$

This corresponds to the equation (2.26) if we transform it to an expression with normal vector by dividing with $|\mathbf{v}| = \sqrt{v_1^2 + v_2^2 + v_3^2} \neq 0$

$$\pi = \frac{\mathbf{v}}{|\mathbf{v}|} + \frac{v_4}{|\mathbf{v}|} e_\infty. \tag{5.6}$$

5.1.2 The inner product of vectors

The inner product of 3D vectors corresponds to the well-known scalar product. The 3D basis vectors e_1, e_2, e_3 square to 1

$$e_1^2 = e_2^2 = e_3^2 = 1 \tag{5.7}$$

For instance, the length of the unit normal vector of equation (2.26)

$$\mathbf{n} = n_1 e_1 + n_2 e_2 + n_3 e_3 \tag{5.8}$$

results in

$$|\mathbf{n}| = \sqrt{n_1^2 + n_2^2 + n_3^2} = 1 \tag{5.9}$$

Because of the specific metric of the Conformal space, the additional basis vectors e_o^2, e_∞^2 square to 0 and their inner product results in $e_\infty \cdot e_o = -1$ (see section 2.7).

Based on these specific properties the inner product between a Conformal vector U and a Conformal vector V is defined by

$$U \cdot V = (\mathbf{u} + u_4 e_\infty + u_5 e_o) \cdot (\mathbf{v} + v_4 e_\infty + v_5 e_o)$$

Let us now translate the inner product to an expression in Euclidean space

$$U \cdot V = \mathbf{u} \cdot \mathbf{v} + v_4 \underbrace{\mathbf{u} \cdot e_\infty}_{0} + v_5 \underbrace{\mathbf{u} \cdot e_o}_{0}$$

$$+u_4\underbrace{e_\infty \cdot \mathbf{v}}_{0}+u_4v_4\underbrace{e_\infty^2}_{0}+u_4v_5\underbrace{e_\infty \cdot e_o}_{-1}$$
$$+u_5\underbrace{e_o \cdot \mathbf{v}}_{0}+u_5v_4\underbrace{e_o \cdot e_\infty}_{-1}+u_5v_5\underbrace{e_o^2}_{0}$$

Based on the fact that a 3D vector is perpendicular to the additional basis vectors this results in

$$U \cdot V = \mathbf{u} \cdot \mathbf{v} - u_5v_4 - u_4v_5 \tag{5.10}$$

or

$$U \cdot V = u_1v_1 + u_2v_2 + u_3v_3 - u_5v_4 - u_4v_5$$

Based on this observation we will now investigate the inner product between points, spheres and planes.

5.1.3 Distance between points

In the case of U and V being points we get

$$u_4 = \frac{1}{2}\mathbf{p}^2, \;\; u_5 = 1$$

$$v_4 = \frac{1}{2}\mathbf{s}^2, \;\; v_5 = 1$$

The inner product of these points is according to equation (5.10)

$$U \cdot V = \mathbf{p} \cdot \mathbf{s} - \frac{1}{2}\mathbf{s}^2 - \frac{1}{2}\mathbf{p}^2$$

$$= p_1s_1 + p_2s_2 + p_3s_3 - \frac{1}{2}(s_1^2 + s_2^2 + s_3^2) - \frac{1}{2}(p_1^2 + p_2^2 + p_3^2)$$

$$= -\frac{1}{2}(s_1^2 + s_2^2 + s_3^2 + p_1^2 + p_2^2 + p_3^2 - 2p_1s_1 - 2p_2s_2 - 2p_3s_3)$$

$$= -\frac{1}{2}((s_1 - p_1)^2 + (s_2 - p_2)^2 + (s_3 - p_3)^2)$$

$$= -\frac{1}{2}(\mathbf{s} - \mathbf{p})^2$$

We recognize that the square of the Euclidean distance of the inhomogenous points corresponds to the inner product of the homogenous points multiplied by -2.

$$(\mathbf{s} - \mathbf{p})^2 = -2(U \cdot V) \tag{5.11}$$

5.1.4 Distance between points and planes

For a vector U representing a point we get

$$u_4 = \frac{1}{2}\mathbf{p}^2, \;\; u_5 = 1$$

For a vector V representing a plane with normal vector $\mathbf{n}$ and distance d we get

$$\mathbf{v} = \mathbf{n}, \;\; v_4 = d, \;\; v_5 = 0$$

The inner product of point and plane is according to equation (5.10)

$$U \cdot V = P \cdot \pi = \mathbf{p} \cdot \mathbf{n} - d \tag{5.12}$$

representing the Euclidean distance of a point and a plane. Note that the scalar product $\mathbf{p} \cdot \mathbf{n}$ describes the distance of a parallel plane to the origin. Its difference with d results in the Euclidean distance from the plane to the point.

5.1.5 Distance between planes and spheres

For a vector U representing a plane with normal vector $\mathbf{n}$ and distance d we get

$$\mathbf{u} = \mathbf{n}, \;\; u_4 = d, \;\; u_5 = 0$$

For a vector V representing a sphere we get

$$v_4 = \frac{1}{2}(\mathbf{s}^2 - r^2), \;\; v_5 = 1$$

The inner product of point and plane is according to equation (5.10)

$$U \cdot V = \pi \cdot S = \mathbf{n} \cdot \mathbf{s} - d \tag{5.13}$$

representing the Euclidean distance of the center point of the sphere and the plane (see section 5.1.4).

5.1.6 Distance between two spheres

We will now compute the inner product of two spheres.

For two vectors S_1 and S_2 representing two spheres we get

$$u_4 = \frac{1}{2}(\mathbf{s_1}^2 - r_1^2), \qquad u_5 = 1$$

For a vector S representing a sphere we get

$$v_4 = \frac{1}{2}(\mathbf{s_2}^2 - r_2^2), \qquad v_5 = 1$$

The inner product of the two spheres is according to equation (5.10)

$$\begin{aligned} S_1 \cdot S_2 &= \mathbf{s_1} \cdot \mathbf{s_2} - \frac{1}{2}(\mathbf{s_2}^2 - r_2^2) - \frac{1}{2}(\mathbf{s_1}^2 - r_1^2) \\ &= \mathbf{s_1} \cdot \mathbf{s_2} - \frac{1}{2}\mathbf{s_2}^2 + \frac{1}{2}r_2^2 - \frac{1}{2}\mathbf{s_1}^2 + \frac{1}{2}r_1^2 \\ &= \frac{1}{2}r_1^2 + \frac{1}{2}r_2^2 - \frac{1}{2}(\mathbf{s_2}^2 - 2\mathbf{s_1} \cdot \mathbf{s_2} + \mathbf{s_1}^2) \\ &= \frac{1}{2}(r_1^2 + r_2^2) - \frac{1}{2}(\mathbf{s_2} - \mathbf{s_1})^2 \end{aligned}$$

We get

$$2(S_1 \cdot S_2) = r_1^2 + r_2^2 - (\mathbf{s_2} - \mathbf{s_1})^2 \tag{5.14}$$

This means that twice the inner product of two spheres equals the sum of the square of the radii minus the square of the Euclidean distance of the sphere centers.

5.1.7 Is a point inside or outside of a sphere ?

We will see now that the inner product of a point and a sphere can be used for the decision of whether a point is inside or outside of a sphere.

For a vector P representing a point (sphere with radius 0) and a vector S representing a sphere with radius r we get according to equation (5.14)

$$2(P \cdot S) = r^2 - (\mathbf{s} - \mathbf{p})^2 \tag{5.15}$$

That is equal to the square of the radius minus the square of the distance between the point and the center point of the sphere.
Based on this observation we can see that

$P \cdot S > 0$: $\mathbf{p}$ is inside of the sphere
$P \cdot S = 0$: $\mathbf{p}$ is on the sphere
$P \cdot S < 0$: $\mathbf{p}$ is outside of the sphere

5.2 Approximation of points with the help of a sphere

In this section a point set $\mathbf{p_i} \in \mathbb{R}^3,\ i \in \{1, ..., n\}$ will be approximated with the help of a sphere. The inhomogenous points $\mathbf{p_i}$ are represented as

$$P_i = \mathbf{p_i} + \frac{1}{2}\mathbf{p_i}^2 e + e_0 \tag{5.16}$$

and the sphere S with inhomogenous center point $\mathbf{s}$ and radius r is represented as

$$S = \mathbf{s} + s_4 e + e_0 \tag{5.17}$$

with

$$s_4 = \frac{1}{2}(s_1^2 + s_2^2 + s_3^2 - r_2^2)$$

5.2.1 Approach

In order to solve the approximation problem we

- define a distance measure between point and sphere with the help of the inner product.
- make a least squares approach to minimize the squares of the distances between the points and the sphere.
- solve the resulting linear system of equations.

5.2.2 Distance measure

In Conformal Geometric Algebra points and spheres are represented as vectors. This is why we use the inner product between points and spheres as a distance measure.
The inner product between a point P_i and the sphere S is defined by

$$P_i \cdot S = (\mathbf{p_i} + \frac{1}{2}\mathbf{p_i}^2 e + e_0) \cdot (\mathbf{s} + s_4 e + e_0) \tag{5.18}$$

According to equation (5.10) this results in

$$P_i \cdot S = \mathbf{p_i} \cdot \mathbf{s} - \frac{1}{2}\mathbf{p_i}^2 - s_4$$

or

$$P_i \cdot S = w_{i,1}s_1 + w_{i,2}s_2 + w_{i,3}s_3 + w_{i,4}s_4 + w_{i,5} = \sum_{j=1}^{4}(w_{i,j}s_j) + w_{i,5} \tag{5.19}$$

with

$$w_{i,k} = \begin{cases} p_{i,k} & : \quad k \in \{1,2,3\} \\ -1 & : \quad k = 4 \\ -\frac{1}{2}\mathbf{p_i}^2 & : \quad k = 5 \end{cases}$$

5.2.3 Least squares approach

In the least-squares sense we consider the minimum of the squares of the distances between all the points and the sphere

$$min \sum_{i=1}^{n} (P_i \cdot S)^2 \tag{5.20}$$

In order to obtain the minimum we have the following 4 necessary conditions

$$\forall k \in \{1..4\} : \frac{\partial(\sum_{i=1}^{n}(P_i \cdot S)^2)}{\partial s_k} = \sum_{i=1}^{n} \frac{\partial(P_i \cdot S)^2}{\partial s_k} = 0 \tag{5.21}$$

With the help of

$$\frac{\partial(P_i \cdot S)^2}{\partial s_k} = 2(P_i \cdot S) \cdot \frac{\partial(P_i \cdot S)}{\partial s_k}$$

and

$$\frac{\partial(P_i \cdot S)}{\partial s_k} = \frac{\partial(\sum_{j=1}^{4}(w_{i,j}s_j) + w_{i,5})}{\partial s_k} = w_{i,k}$$

we obtain

$$\forall k \in \{1..4\} : \frac{\partial(P_i \cdot S)^2}{\partial s_k} = 2\sum_{i=1}^{n}(\sum_{j=1}^{4}(w_{i,j}s_j w_{i,k}) + w_{i,5}w_{i,k}) = 0$$

or

$$\forall k \in \{1..4\} : \sum_{i=1}^{n}\sum_{j=1}^{4}(w_{i,j}w_{i,k}s_j) = -\sum_{i=1}^{n}(w_{i,5}w_{i,k})$$

which is the same as

$$\forall k \in \{1..4\} : \sum_{j=1}^{4}\sum_{i=1}^{n}(w_{i,j}w_{i,k}s_j) = -\sum_{i=1}^{n}(w_{i,5}w_{i,k})$$

or

$$\forall k \in \{1..4\} : \sum_{j=1}^{4} s_j \sum_{i=1}^{n}(w_{i,j}w_{i,k}) = -\sum_{i=1}^{n}(w_{i,5}w_{i,k})$$

The result of the least squares approach is as follows :

$$\begin{pmatrix} \sum_{i=1}^{n} p_{i,1}p_{i,1} & \sum_{i=1}^{n} p_{i,2}p_{i,1} & \sum_{i=1}^{n} p_{i,3}p_{i,1} & -\sum_{i=1}^{n} p_{i,1} \\ \sum_{i=1}^{n} p_{i,1}p_{i,2} & \sum_{i=1}^{n} p_{i,2}p_{i,2} & \sum_{i=1}^{n} p_{i,3}p_{i,2} & -\sum_{i=1}^{n} p_{i,2} \\ \sum_{i=1}^{n} p_{i,1}p_{i,3} & \sum_{i=1}^{n} p_{i,2}p_{i,3} & \sum_{i=1}^{n} p_{i,3}p_{i,3} & -\sum_{i=1}^{n} p_{i,3} \\ -\sum_{i=1}^{n} p_{i,1} & -\sum_{i=1}^{n} p_{i,2} & -\sum_{i=1}^{n} p_{i,3} & \sum_{i=1}^{n} 1 \end{pmatrix} \cdot s = \begin{pmatrix} \frac{1}{2}\sum_{i=1}^{n} \mathbf{p_i}^2 p_{i,1} \\ \frac{1}{2}\sum_{i=1}^{n} \mathbf{p_i}^2 p_{i,2} \\ \frac{1}{2}\sum_{i=1}^{n} \mathbf{p_i}^2 p_{i,3} \\ -\frac{1}{2}\sum_{i=1}^{n} \mathbf{p_i}^2 \end{pmatrix} \tag{5.22}$$

with $p_{i,1}, p_{i,2}, p_{i,3}$ as inhomogenous coordinates of the points $\mathbf{p}_i$. The result $s = (s_1, s_2, s_3, s_4)$ represents the center point of the sphere (s_1, s_2, s_3) and its radius in terms of $r^2 = s_1^2 + s_2^2 + s_3^2 - 2s_4$

5.3 Approximation of points with the help of planes or spheres

In this section, a point set $\mathbf{p_i} \in \mathbb{R}^3,\ i \in \{1, ..., n\}$ will be approximated with the help of the best approximation plane or sphere.

Plane and sphere in conformal space are vectors of the form

$$S = s_1e_1 + s_2e_2 + s_3e_3 + s_4e_\infty + s_5e_0 \tag{5.23}$$

while the points $\mathbf{p_i}$ are specific vectors of the form

$$P_i = \mathbf{p_i} + \frac{1}{2}\mathbf{p_i}^2 e_\infty + e_0 \tag{5.24}$$

5.3.1 Approach

In order to solve the approximation problem we

- use the distance measure of the previous section between point and sphere/plane with the help of the inner product.
- make a least squares approach to minimize the squares of the distances between the points and the sphere/plane.
- solve the resulting eigenvalue problem.

5.3.2 Distance measure

A distance measure between a point P_i and the sphere/plane S can be defined in Conformal Geometric Algebra with the help of their inner product

$$P_i \cdot S = (\mathbf{p_i} + \frac{1}{2}\mathbf{p_i}^2 e_\infty + e_0) \cdot (\mathbf{s} + s_4 e_\infty + s_5 e_0) \tag{5.25}$$

According to equation (5.10) this results in

$$P_i \cdot S = \mathbf{p_i} \cdot \mathbf{s} - s_4 - \frac{1}{2} s_5 \mathbf{p_i}^2$$

or

$$P_i \cdot S = \sum_{j=1}^{5} w_{i,j} s_j \tag{5.26}$$

with

$$w_{i,k} = \begin{cases} p_{i,k} & : \quad k \in \{1,2,3\} \\ -1 & : \quad k = 4 \\ -\frac{1}{2}\mathbf{p_i}^2 & : \quad k = 5 \end{cases}$$

5.3.3 Least squares approach

In the least-squares sense we consider the minimum of the squares of the distances between all the points and the plane/sphere

$$min \sum_{i=1}^{n} (P_i \cdot S)^2 \tag{5.27}$$

In order to obtain the minimum this can be rewritten in bilinear form to

$$min(s^T Bs) \tag{5.28}$$

with

$$s^T = (s_1, s_2, s_3, s_4, s_5)$$

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} & b_{4,5} \\ b_{5,1} & b_{5,2} & b_{5,3} & b_{5,4} & b_{5,5} \end{pmatrix}$$

$$b_{j,k} = \sum_{i=1}^{n} w_{i,j} w_{i,k}$$

The matrix B is symmetric since $b_{j,k} = b_{k,j}$. Without loss of generality we consider only normalized results $s^T s = 1$. A conventional approach to such a constrained optimization problem is to introduce the Lagrangian

$$L = s^T Bs - \lambda s^T s,$$

$$s^T s = 1,$$

$$B^T = B$$

Necessary conditions for a minimum are

$$0 = \nabla L = 2 \cdot (Bs - \lambda s) = 0$$

$$\rightarrow Bs = \lambda s$$

The solution of the minimization problem is given as the Eigenvector of B that corresponds to the smallest Eigenvalue. (see [28] for details)

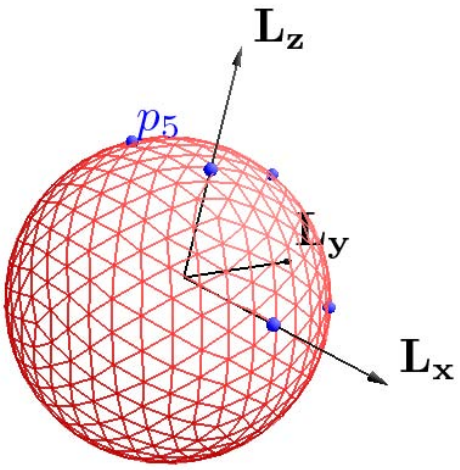

Figure 5.1: Fit of a sphere

5.3.4 Example

Let us have a look on an example with 5 points.

Point	x	y	z
$\mathbf{p_1}$	1	0	0
$\mathbf{p_2}$	1	1	0
$\mathbf{p_3}$	0	0	1
$\mathbf{p_4}$	0	1	1
$\mathbf{p_5}$	-1	0	1

The least squares calculation results in

$$S = -0.301511e_1 + 0.301511e_2 - 0.301511e_3$$

$$-0.603023e_\infty + 0.603023e_0$$

Another representation of this object is

$$S = -\frac{1}{2}e_1 + \frac{1}{2}e_2 - \frac{1}{2}e_3 - e_\infty + e_0$$

This corresponds to a sphere with the center point $\mathbf{s} = (0.5, 0.5, -0.5)$ and the square of the radius $r^2 = 2.75$ (see figure 5.1).

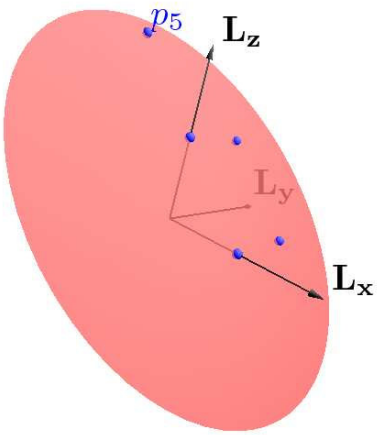

Figure 5.2: Fit of a plane

Let us now change the fifth point in order that all the points are within one plane.

Point	x	y	z
$\mathbf{p_1}$	1	0	0
$\mathbf{p_2}$	1	1	0
$\mathbf{p_3}$	0	0	1
$\mathbf{p_4}$	0	1	1
$\mathbf{p_5}$	-1	0	2

Now, the result is

$$S = 0.57735e_1 + 0.57735e_3 + 0.57735e_\infty$$

representing a plane according to figure 5.2.

Chapter 6

Rapid prototyping of robotics algorithms

Conformal Geometric Algebra together with the tool CLUCalc support the rapid prototyping of algorithms, namely the interactive and visual development as well as the test of the algorithms. This chapter shows the rapid prototyping of two robotics algorithms. The algorithms are derived from two real applications of Cinvestav, Guadaljara for the grasping of the robot Geometer shown in figure 6.7 on page 77. At first, we visualize the inverse kinematics of the robot (see [35]). Second, we visualize the grasping algorithm of the robot (see [39]). The visualization is done using CLUCalc (see section 2.13.1). The corresponding CLUScripts can be downloaded from the homepage

`http://www.gris.informatik.tu-darmstadt.de/~dhilden/`

These scripts demonstrate the easy way of developing algorithms based on CLUCalc. The figures of this chapter show the visualization of the main steps of the rapid prototyping process.

6.1 The inner product and angles

Kinematics calculations often need the computation of angles. This section presents relations between the inner product of geometric objects and their angles.

Angles between two objects o_1, o_2 like two lines or two planes can be computed using the inner product of the normalized direct representation of the objects.

$$cos(\theta) = \frac{o_1^* \cdot o_2^*}{|o_1^*|\,|o_2^*|} \tag{6.1}$$

or

$$\theta = \angle(o_1, o_2) = \arccos \frac{o_1^* \cdot o_2^*}{|o_1^*|\,|o_2^*|} \tag{6.2}$$

Please refer to [19] for more details.

Let us derive as one example an exression for the angle between two planes based on the observation of equation (5.10). For a vector π_1 representing a plane with normal vector $\mathbf{n_1}$ and distance d_1 we get

$$\mathbf{u} = \mathbf{n_1},\ \ u_4 = d_1,\ \ u_5 = 0$$

For a vector π_2 representing another plane we get

$$\mathbf{v} = \mathbf{n_2},\ \ v_4 = d_2,\ \ v_5 = 0$$

The inner product of the two planes is

$$\pi_1 \cdot \pi_2 = \mathbf{n_1} \cdot \mathbf{n_2} \tag{6.3}$$

representing the scalar product of the two normals of the planes.
Based on this observation the angle θ between two planes can be computed as follows

$$cos(\theta) = \pi_1 \cdot \pi_2 \tag{6.4}$$

This corresponds to equation (6.2) taking into account that the planes are normalized and that the dualization operation only switches between the two possible angles between planes.

6.2 Inverse kinematics application

Objects like robots or virtual humans can be modeled as a set of rigid links connected together at various joints. These objects are described as kinematic chains.
The robot of figure 6.1 consists of 3 links and one gripper

- the 3 joint points are called P_0, P_1 and P_2
- the 3 link distances are called d_1, d_2 and d_3
- the distance from the last joint P_2 to the 'T' intersection of the gripper is called d_4

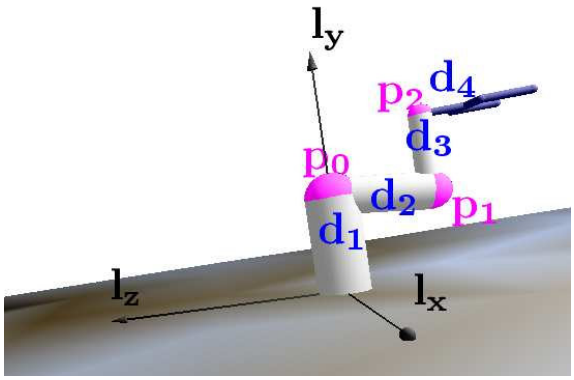

Figure 6.1: Kinematic chain of the example robot

It has 5 degrees of freedom (DOF) by means of the following 5 joint angles θ_1 .. θ_5 :

- θ_1 : rotate robot (around L_y)
- $\theta_2, \theta_3, \theta_4$: acting in plane π_1 (blue plane in figure 6.3)
- θ_5 : rotate gripper

whereby the plane π_1 is defined by the origin e_0, the point P_y (see equation (2.23)) on the y-axis and the target point P_t (see figure 6.2).

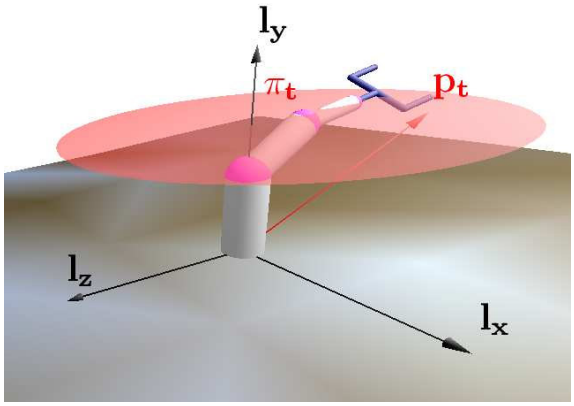

Figure 6.2: Target point and gripper plane

According to equation (2.27) we get

$$\pi_1^* = e_0 \wedge P_y \wedge P_t \wedge e_\infty \tag{6.5}$$

Our goal is to find the joint angles in terms of a target position P_t and an orientation of the gripper plane π_t (the red plane in figure 6.2).
In Conformal Geometric Algebra, this inverse kinematics problem can be solved in a geometrically very intuitive way due to its easy handling of intersections of spheres, circles and planes etc.
Our approach is based on the papers [13] and [36].

For ease of use we define the gripper plane π_t as parallel to the ground plane. Since a plane can be described using equation (2.26) we get

$$\pi_t = e_2 + P_{t,y}\, e_\infty \tag{6.6}$$

whereby $P_{t,y}$ is the y-coordinate of the target point P_t.

In the following steps we will at first calculate the 3 locations P_0, P_1, P_2. Based on these points we will be able to calculate the 5 joint angles θ_1 .. θ_5.

6.2.1 Computation of P_0

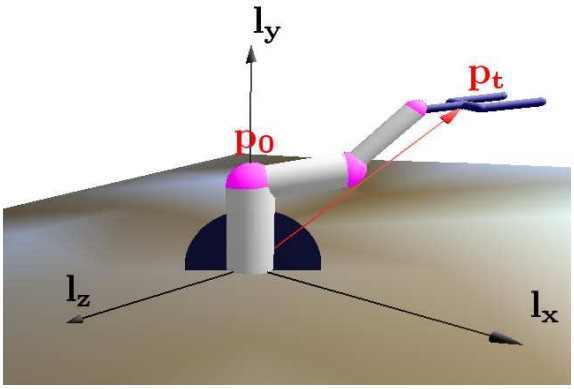

Figure 6.3: Computation of P_0

In the first step the point P_0 is calculated. Its 3D representation is $(0, d_1, 0)$.
Using equation (2.20) we get

$$P_0 = d_1 e_2 + \frac{1}{2} d_1^2 e_\infty + e_0 \tag{6.7}$$

.

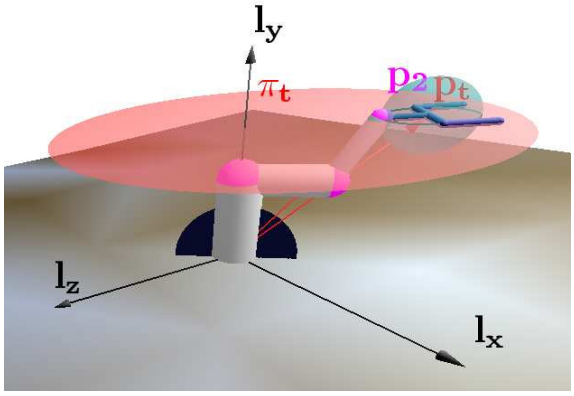

Figure 6.4: Computation of P_2

6.2.2 Computation of P_2

In the second step point P_2 is calculated. It is the joint location of the last link of the robot. This means that it has to lie on the sphere S_t with the center point P_t and with the length d_4 of the displacement between P_t and P_2 as radius. Using equation (2.24) we get

$$S_t = P_t - \frac{1}{2} d_4^2 e_\infty \tag{6.8}$$

Since the gripper also has to lie in the orientation plane π_t, we have to intersect it with S_t. The result is the circle Z_t (see equation (2.28))

$$Z_t = S_t \wedge \pi_t. \tag{6.9}$$

Please remember that a plane is simply a sphere with infinite radius.

Since P_2 also has to lie in the plane π_1, the intersection of π_1 with the circle Z_t results in a point pair (see equation (2.33))

$$Pp_2 = Z_t \wedge \pi_1. \tag{6.10}$$

From the mechanics point of view, only one of these two points is applicable which we choose as our point P_2. We use the following formula for extracting the two points of a point pair Pp (see [26])

$$P_\pm = \frac{\pm\sqrt{Pp* \cdot Pp*} + Pp*}{e_\infty \cdot Pp*} \tag{6.11}$$

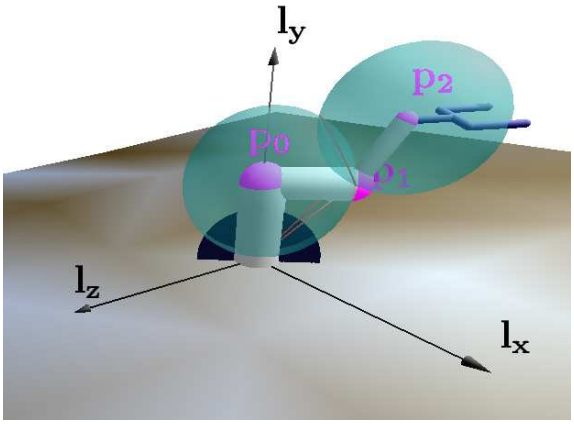

Figure 6.5: Computation of P_1

6.2.3 Computation of P_1

In the third step the point P_1 is calculated.

Computing this point is usually a difficult task because it is the intersection of two circles. However, using Conformal Geometric Algebra we can determine it by intersecting the spheres S_1 and S_2 with the plane π_1

$$S_1 = P_0 - \frac{1}{2}d_2^2 e_\infty, \tag{6.12}$$

$$S_2 = P_2 - \frac{1}{2}d_3^2 e_\infty, \tag{6.13}$$

and

$$Pp_1 = S_1 \wedge S_2 \wedge \pi_1. \tag{6.14}$$

Again, we have to choose one point from the resulting point pair.

As an example for the CLUCalc implementation, please find as follows the CLUCalc code of this step :

```
s1 = p0 - 0.5*d2*d2*einf;
s2 = p2 - 0.5*d3*d3*einf;
Pp1 = s1^s2^PI1;
// choose one of the two points
p1 = DissectSecond(*Pp1);
```

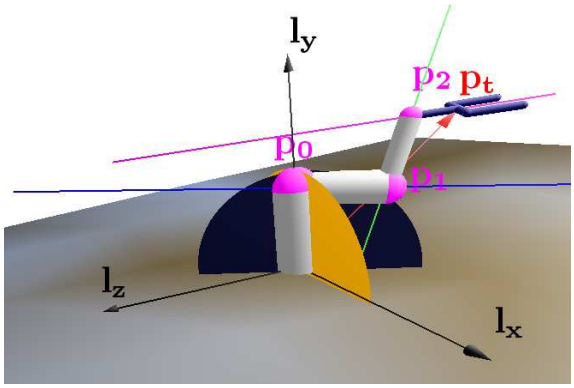

Figure 6.6: Visualization of step 4

6.2.4 Computation of the joint angles

At first, all the auxiliary planes and lines, that are needed for the computation of the angles of the joints are calculated. We need

- the plane π_2 (orange in figure 6.6) spanned by the x-axis and the y-axis. Since the z-axis is perpendicular to this plane, we get

$$\pi_2 = e_3 \tag{6.15}$$

- the (blue) line L_1 through P_0 and P_1

$$L_1^* = P_0 \wedge P_1 \wedge e_\infty \tag{6.16}$$

- the (green) line L_2 through P_1 and P_2

$$L_2^* = P_1 \wedge P_2 \wedge e_\infty \tag{6.17}$$

- the (magenta) line L_3 through P_2 and P_t

$$L_3^* = P_2 \wedge P_t \wedge e_\infty \tag{6.18}$$

Now, we are able to compute all the joint angles

$$\theta_1 = \angle(\pi_1, \pi_2) \tag{6.19}$$

$$\theta_2 = \angle(L_1, L_y) \tag{6.20}$$

$$\theta_3 = \angle(L_1, L_2) \tag{6.21}$$

$$\theta_4 = \angle(L_2, L_3) \tag{6.22}$$

using the equation (6.2) with o_1, o_2 being either two lines or two planes.
In our simplified example

$$\theta_5 = 0 \tag{6.23}$$

since the gripper should be parallel to the ground plane.

6.3 Grasping an object

This algorithm was derived from the algorithm for the grasping process of the robot Geometer (figure 6.7) at Cinvestav, Guadalajara (see [39]). Here we present its rapid prototyping using CLUCalc. The goal of our grasping algorithm is to move a gripper to

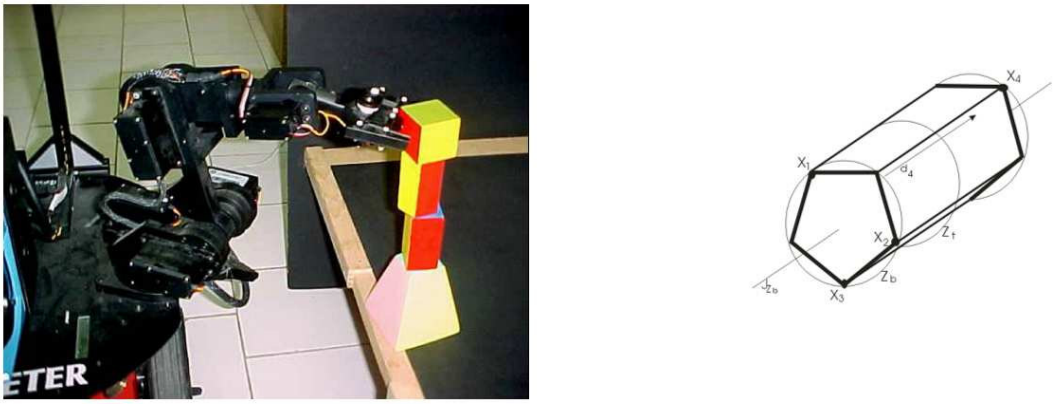

Figure 6.7: Robot Geometer grasping an object

an object, both gripper and object described by a circle.
At first we compute the grasping circle Z_t of the object, second the gripper circle Z_h is estimated, third the translation and rotation for the movement is computed. The figures of this section show the visualization of the main steps of the rapid prototyping of this algorithm.

6.3.1 Assign points

First of all, we need 4 points identifying the object to be grasped. In the real application they are taken from a calibrated stereo pair of images of the object. In order to assign the four points we at first compute the distances between all the 4 points and the plane spanned by the 3 other points. The point with the greatest distance d_a will be called apex point x_a (see CLUCalc visualization in figure 6.8). The other 3 points are called base points $x_{b_1}, x_{b_2}, x_{b_3}$.

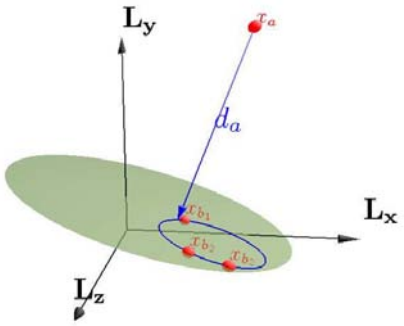

Figure 6.8: Assign points

6.3.2 Compute grasping circle Z_t

To compute the grasping plane π_t we compute the base circle

$$Z_b^* = x_{b_1} \wedge x_{b_2} \wedge x_{b_3} \tag{6.24}$$

based on the outer product of the three base points (see table 2.5 on page 19). The object might be grasped in the middle of it. That means, we translate the circle Z_b in the direction and magnitude of $\frac{d_a}{2}$.
With the help of the translator

$$T = 1 + \frac{1}{4} d_a e_\infty \tag{6.25}$$

we compute the grasping circle Z_t (see CLUCalc visualization in figure 6.9)

$$Z_t = T Z_b \tilde{T} \tag{6.26}$$

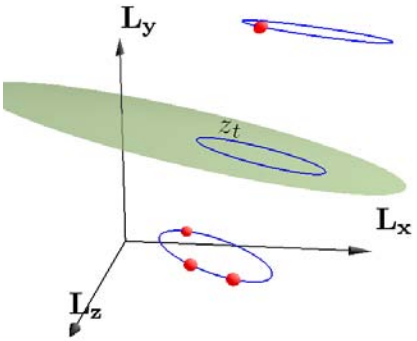

Figure 6.9: Grasping circle Z_t

6.3.3 Gripper circle

While the gripper circle of our simulation is computed by default (see CLUCalc visualization in figure 6.11), the gripper circle of the real robot is estimated by tracking its metallic screws. Figure 6.10 shows the position of the point P_h. We create a sphere

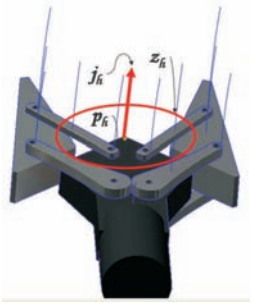

Figure 6.10: Gripper

with center point P_h and radius r (equal to the middle of the aperture of the gripper) according to table 2.5 on page 19

$$S_h = P_h - \frac{1}{2} r^2 e_\infty \tag{6.27}$$

Now tracking two additional points a and b on the gripper we create the plane π_h.

$$\pi_h^* = P_h \wedge a \wedge b \wedge e_\infty \tag{6.28}$$

(please notice that a plane is created with the help of the outer product of three points and the point at infinity).
Finally we calculate the gripper circle as the intersection of sphere and plane

$$Z_h = S_h \wedge \pi_h \tag{6.29}$$

6.3.4 Estimation of translation and rotation

Now, we compute the transformation needed to move the gripper circle to the grasping circle.

The translation axis is computed easily having the centers of the circles (the center point P of a circle Z can be easily computed with the help of a sandwich product $P = Ze_\infty Z$).

$$l_T^* = P_h \wedge P_t \wedge e_\infty \tag{6.30}$$

The distance d between the circles is $d = |l_T^*|$. The rotation axis is computed using the

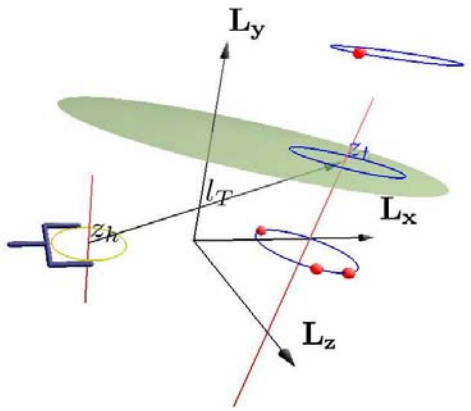

Figure 6.11: Gripper circle Z_h, grasping circle Z_t and their axes L_h and L_t

axes of each circle L_h^* and L_t^* (see the red lines in figure 6.11)

$$L_h^* = Z_h \wedge e_\infty, \tag{6.31}$$
$$L_t^* = Z_t \wedge e_\infty. \tag{6.32}$$

That axes L_h and L_t yield in the plane π_{th}^* given by

$$\pi_{th}^* = L_t^* \wedge (L_h^* E), \qquad E = e_0 \wedge e_\infty \tag{6.33}$$

therefore the rotation axis is

$$L_r^* = P_h \wedge \pi_{th} \wedge e_\infty. \tag{6.34}$$

The angle θ between the two circles can be computed based on the inner product of the two lines L_h^* and L_t^* (see CLUCalc visualization in figure 6.11).

$$\cos(\theta) = \frac{L_t^* \cdot L_h^*}{|L_t^*||L_h^*|} \tag{6.35}$$

Once that we estimated the rotation and translation axes (see section 2.9.1),

$$R = e^{-\frac{1}{2}\Delta\theta L_r} \tag{6.36}$$

$$T = 1 + \frac{1}{2}\Delta d L_t e_\infty \tag{6.37}$$

we are able to move the gripper circle Z_h step by step (z_h') towards the grasping circle Z_t. (see CLUCalc visualization in figure 6.12).

$$z_h' = TRZ_h\widetilde{R}\widetilde{T}. \tag{6.38}$$

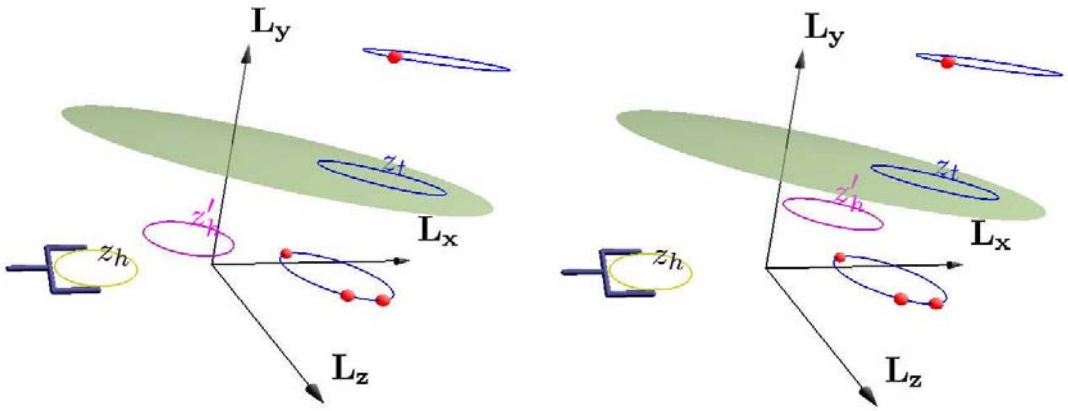

Figure 6.12: Moving gripper circle Z_h towards the grasping circle Z_t

Chapter 7

Efficient inverse kinematics in Conformal Geometric Algebra

In this chapter, we present the efficient implementation of the inverse kinematics of a human-arm-like model. For the animation of humanoid models, inverse kinematics (IK) solutions are important as a basic building block for path planning. The standard model for arms (and also legs) is a seven 7 DOF kinematic chain, with 3 degrees of freedom $(\theta_1, \theta_2, \theta_3)$ at the shoulder, 1 degree of freedom at the elbow θ_4 and 3 degrees of freedom at the wrist $(\theta_5, \theta_6, \theta_7)$.

The current standard tool for solving the inverse problem of mapping from a given end effector state to the configuration space $\{\theta_i\}$ is due to Tolani, Goswami, and Badler [80]. Our analytic solution to the inverse problem in Conformal Geometric Algebra is an improvement of [36], and its derivation is considerably simpler than in affine or projective geometry. Perhaps more importantly for the prospective user, our approach also turns out to be faster, when implemented using Gaigen 2 or optimized based on Maple.

7.1 Optimizations based on quaternions

Normally, the goal of an inverse kinematics algorithm is the computation of the joint angles (see section 6.2). But, in our application the goal is to compute quaternions in order to perform SLERP operations for the motion of the arm (see [86] for details on this motion interpolation procedure). In our former approach for the application of this chapter [36] we at first computed the angles and then the corresponding quaternions.

But, in section 3.2 we realized that quaternions can be directly handled in a Confor-

mal Geometric Algebra algorithm. This is the reason why we improved the algorithm to directly and efficiently compute the needed quaternions.

7.1.1 Direct computation of quaternions

For efficiency reasons, our inverse kinematics approach directly uses quaternions in the algorithm. This is why we avoid the effort of translating between different mathematical systems like transformation matrices and quaternions. Here, we directly compute a quaternion that rotates an object from one point P_1 to another point P_2, both points having the same distance to the origin.

At first, we calculate the middle line L_m between the two points through the origin. In Conformal Geometric Algebra, a middle plane of two points is described by their difference (see [64])

$$\pi_m = P_1 - P_2. \tag{7.1}$$

We calculate the middle line with the help of the intersection of this plane and the plane through the origin and the points P_1 and P_2

$$\pi_e^* = e_o \wedge P_1 \wedge P_2 \wedge e_\infty, \tag{7.2}$$

and get

$$L_m = \pi_e \wedge \pi_m. \tag{7.3}$$

Second, in order to rotate from P_1 to P_2 we have to rotate around the middle line

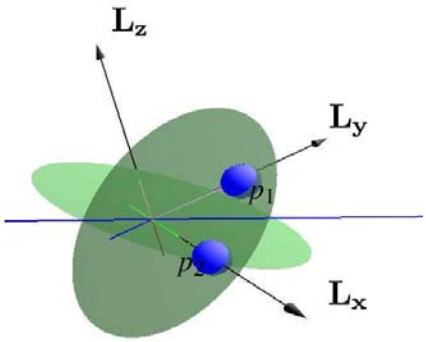

Figure 7.1: Rotation based on the line between two points through the origin

with radius π. This results in a quaternion identical with the normalized line (see

section 3.2.3)

$$Q = \frac{L_m}{|L_m|}. \tag{7.4}$$

In figure 7.1 the two points P_1 and P_2 are indicated by two blue spheres. They can be transformed into each other based on a rotation around the blue middle line.

7.1.2 Efficient computation of quaternions

For efficiency reasons we use an approach to calculate quaternions without the need of using trigonometric functions. According to equation (3.13) a quaternion describing a rotation can be computed with the help of half of an angle and a normalized rotations axis. For example, if $L = i = e_3 \wedge e_2$, the resulting quaternion

$$\begin{aligned} Q &= \cos(\frac{\phi}{2}) + i\sin(\frac{\phi}{2}) \\ &= \cos(\frac{\phi}{2}) + (\mathbf{e_3} \wedge \mathbf{e_2})\sin(\frac{\phi}{2}) \end{aligned}$$

represents a rotation around the x-axis. The angle between two lines or two planes is defined according to section 6.1 as follows:

$$\cos(\theta) = \frac{o_1^* \cdot o_2^*}{|o_1^*|\,|o_2^*|}, \tag{7.5}$$

We already know the cosine of the angle. This is why we are able to compute the quaternion in a more direct way using the following two properties of the trigonometric functions

$$\cos(\frac{\phi}{2}) = \pm\sqrt{\frac{1+\cos(\phi)}{2}} \tag{7.6}$$

and

$$\sin(\frac{\phi}{2}) = \pm\sqrt{\frac{1-\cos(\phi)}{2}}, \tag{7.7}$$

leading to the formulas

$$\cos(\frac{\phi}{2}) = \pm\sqrt{\frac{1+\frac{o_1^* \cdot o_2^*}{|o_1^*||o_2^*|}}{2}} \tag{7.8}$$

and

$$\sin(\frac{\phi}{2}) = \pm\sqrt{\frac{1-\frac{o_1^* \cdot o_2^*}{|o_1^*||o_2^*|}}{2}}. \tag{7.9}$$

The signs of these formulas depend on the application.

Table 7.1: Input/output parameters of the inverse kinematics algorithm

parameter	**meaning**
P_w	target point of wrist
ϕ	swivel angle
d_1, d_2	length of the forearm and the upper arm
Q_s	shoulder quaternion
Q_e	elbow quaternion

7.2 The inverse kinematics algorithm

Here, we describe the inverse kinematics of the human-arm-like model step by step. Our goal is to reach the chosen point P_w with the wrist. An arbitrary orientation of the gripper is not investigated in this document. Additional input parameters of the algorithm are the lengths of the arm as well as the swivel angle. Results of the algorithm are the quaternions at the shoulder as well as at the elbow. Please find a summary of the input and output parameters of the algorithm in table 7.1.

7.2.1 Compute the swivel plane

According to [80] we use the swivel angle ϕ as one free degree of redundancy.

The swivel plane is the plane rotated by ϕ around the line L_{sw} through shoulder (at the origin) and P_w (see figure 7.2).

$$L_{sw} = (e_o \wedge P_w \wedge e_\infty)^* \tag{7.10}$$

Note that the direct representation of a line is defined with the help of two points and the point at infinity. The quaternion Q_{swivel} is defined according to equation (3.13)

$$Q_{swivel} = \cos(\frac{\phi}{2}) + \frac{L_{sw}}{|L_{sw}|}\sin(\frac{\phi}{2}) \tag{7.11}$$

Initially, the swivel plane is defined with the help of the origin, the point P_w, the point P_z ($\mathbf{x} = e_3$)

$$P_z = e_3 + \frac{1}{2}e_\infty + e_o \tag{7.12}$$

and the point at infinity (see table 2.5 on page 19)

$$\pi_{swivel} = (e_o \wedge P_z \wedge P_w \wedge e_\infty)^*. \tag{7.13}$$

Its final rotated location is

$$\pi_{swivel} = Q_{swivel}\, \pi_{swivel}\, \tilde{Q}_{swivel} \tag{7.14}$$

For details on computing rotations in Conformal Geometric Algebra please refer to section 2.9.1.

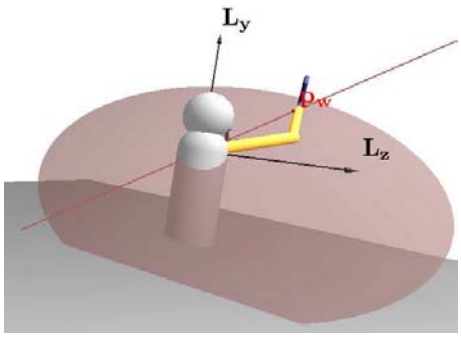

Figure 7.2: Swivel plane

7.2.2 The elbow point P_e

With the help of the two spheres

$$\begin{aligned} S_1 &= P_w - \frac{1}{2} d_2^2 e_\infty \\ S_2 &= e_o - \frac{1}{2} d_1^2 e_\infty \end{aligned} \tag{7.15}$$

with center points P_w and e_o and radii d_2, d_1 we are able to compute the circle determining all the possible locations of the elbow as the intersection of the spheres (see table 2.5 on page 19).

$$Z_e = S_1 \wedge S_2 \tag{7.16}$$

The intersection with the swivel plane delivers the point pair.

$$Pp = Z_e \wedge \pi_{swivel} \tag{7.17}$$

and we decide for one of the two possible elbow points and call it P_e. Please refer to the formula 6.11 for extracting points of a point pairs. Figure 7.3 shows the elbow point P_e as the intersection of the swivel plane with the two spheres at the shoulder and at the target point P_w.

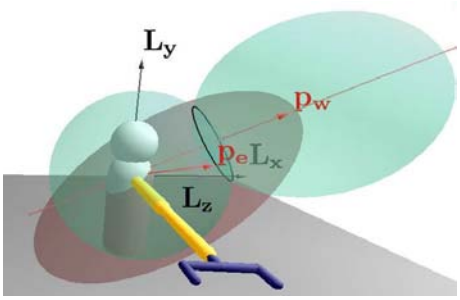

Figure 7.3: Compute the elbow point

7.2.3 Calculate the elbow quaternion Q_e

The elbow angle θ_4 is computed with the help of the line L_{se} through the shoulder and the elbow

$$L_{se} = (e_o \wedge P_e \wedge e_\infty)^* \tag{7.18}$$

and the line L_{ew} through the shoulder and the wrist

$$L_{ew} = (P_e \wedge P_w \wedge e_\infty)^*. \tag{7.19}$$

Based on these two lines we are able to compute the angle between them ($c_i = \cos(\theta_i)$)

$$c_4 = \cos(\theta_4) = \frac{L^*_{se} \cdot L^*_{ew}}{|L^*_{se}|\,|L^*_{ew}|} \tag{7.20}$$

according to equation (7.5).

Now, we are able to compute the quaternion Q_e according to equation (3.13)

$$Q_e = \cos(\theta_4/2) + \sin(\theta_4/2)i. \tag{7.21}$$

It represents a rotation around the local x-axis with the angle θ_4. The optimized version of this quaternion is

$$Q_e = \sqrt{\frac{1+c_4}{2}} - \sqrt{\frac{1-c_4}{2}}i, \tag{7.22}$$

according to (7.8) and (7.9). This quaternion rotates the upper arm corresponding to the angle between the two yellow lines as shown in figure 7.4.

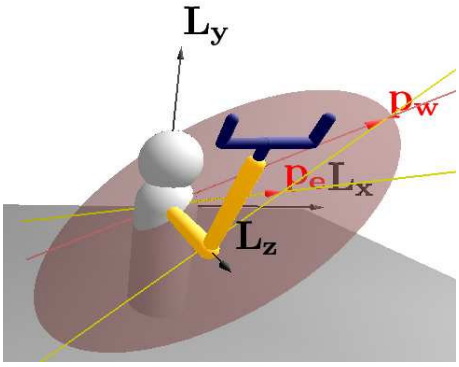

Figure 7.4: Use the elbow quaternion

7.2.4 Rotate to the elbow position

At first we calculate the middle line L_m through the origin within the same distance from the points P_e and P_{ze} (see section 7.1.1)

$$P_{ze} = d_1 e_3 + \frac{1}{2} d_1^2 e_\infty + e_o. \tag{7.23}$$

We will need this line L_m in the next step in order to rotate around this line.
To compute L_m, we use the middle plane (difference of the two points P_e and P_{ze})

$$\pi_m = P_{ze} - P_e \tag{7.24}$$

and the plane through the origin and the points P_e and P_{ze}

$$\pi_e^* = e_o \wedge P_{ze} \wedge P_e \wedge e_\infty \tag{7.25}$$

and intersect them

$$L_m = \pi_e \wedge \pi_m. \tag{7.26}$$

In order to rotate the elbow towards our already computed point P_e we have to rotate around the middle line of the previous step with angle π. This results in a quaternion identical with the normalized middle line (see equation 3.13).

$$Q_{12} = \frac{L_m}{|L_m|}. \tag{7.27}$$

Figure 7.5 shows this rotation from the z-axis L_z to the elbow point with the help of the yellow middle line.

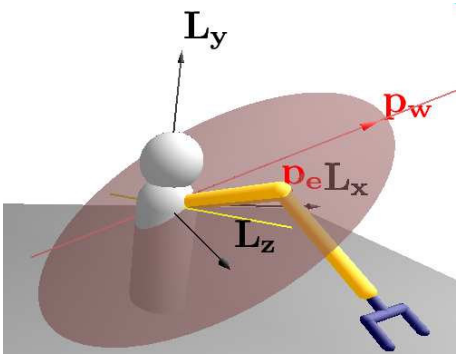

Figure 7.5: Rotate to the elbow position

7.2.5 Rotate to the wrist location

The angle θ_3 (and the resulting quaternion Q_3) is computed with the help of the y-z-plane rotated by the quaternion Q_{12} and the swivel plane. The plane in y and z direction (with normal vector e_1 and zero distance to the origin), is computed by

$$\pi_{yz} = e_1. \tag{7.28}$$

The rotated plane π_{yz2} results in

$$\pi_{yz2} = Q_{12}\, \pi_{yz}\, \tilde{Q}_{12}. \tag{7.29}$$

Based on these two planes we are able to compute the angle between them

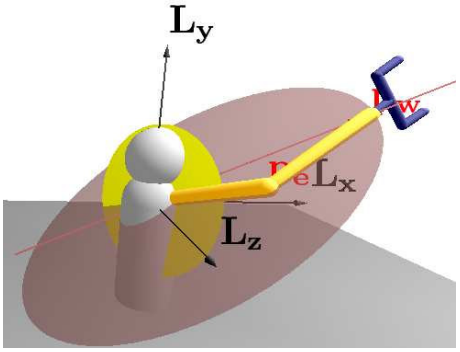

Figure 7.6: Rotate to the wrist location

$$c_3 = \cos(\theta_3) = \frac{\pi^*_{yz2} \cdot \pi^*_{swivel}}{\left|\pi^*_{yz2}\right| \left|\pi^*_{swivel}\right|} \tag{7.30}$$

according to equation (7.5) and we get the quaternion

$$Q_3 = \cos(\theta_3/2) + \sin(\theta_3/2)k. \tag{7.31}$$

It represents a rotation around the local z-axis with the angle θ_3. The optimized version of this quaternion is

$$Q_3 = \pm\sqrt{\frac{1+c_3}{2}} + \sqrt{\frac{1-c_3}{2}}k, \tag{7.32}$$

according to (7.8) and (7.9).
Note: the sign of this quaternion depends on which side of the plane π_{yz2} the point P_w is lying. This can be easily computed with the help of the inner product

$$\pi_{yz2} \cdot P_w.$$

This quaternion rotates the arm to the wrist location as shown in figure 7.6.

The resulting quaternion for the shoulder rotation can now be computed as the product of Q_{12} and Q_3

$$Q_s = Q_{12}Q_3. \tag{7.33}$$

Together with the elbow quaternion Q_e, we have all the information needed for the interpolation based on SLERP (see [86] for details on this motion interpolation procedure) in order to reach the target.

7.3 Runtime optimization approaches

According to the development process proposed in section 2.13, at first we developed and simulated our algorithm visually based on CLUCalc, developed by Christian Perwass.

Then, we had to implement it on the target platform, the virtual reality system Avalon ([90]) written in C++. Previously, inverse kinematics was implemented using IKAN [80], a widely used C++ library.

When implementing our algorithm at first with Gaigen, the runtime benchmarks were worse than the IKAN implementation. But when using our optimization techniques our approaches outperformed the IKAN implementation clearly. We present two different optimization approaches with different advantages, one is based on Maple, the other one is based on the code generator Gaigen 2.

Table 7.2: Input/output parameters of the inverse kinematics algorithm

parameter	**Gaigen**	**meaning**
P_w	`pw`	target point of wrist
ϕ	`Sangle`	swivel angle
d_1, d_2	`d1, d2`	length of the forearm and the upper arm
Q_s	q_s	shoulder quaternion
Q_e	q_e	elbow quaternion

7.3.1 Optimizations with Gaigen 2

The specific technique of Gaigen 2 is described in section 2.13.2 on page 32. The following detailed description of the inverse kinematics algorithm is offered as an explicit example of how one can think in geometry, and directly program in geometric elements. We present the Gaigen 2 code for the above inverse kinematics algorithm.

The goal of our inverse kinematics algorithm is to compute the output parameters Q_s and Q_e based on the input parameters (see table 7.2).

compute the elbow point P_e

Please find the Gaigen 2 implementation of the equations 7.15 to 7.17 as follows :

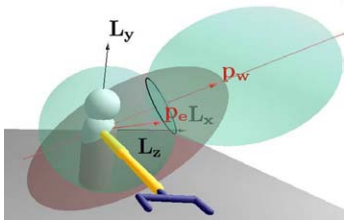

Figure 7.7: Compute the elbow point

```
Sphere s1 = _Sphere(pw - 0.5f*d2*d2*einf); // einf means e∞

originSphere s2 = _originSphere(e0 - 0.5f*d1*d1*einf);

Circle Z_e = _Circle(s1^s2);
```

With the help of the two spheres s1, s2 with center points P_w (target point of the wrist) and e_o (shoulder located at the origin) and radii d_2, d_1 we are able to compute the circle determining all the possible locations of the elbow as the intersection of the spheres.

For the needed geometric objects we use the following specializations:
Both, the target point P_w as well as the sphere S_1 are assigned to a multivector type called `Sphere` (please remember that a point in Conformal Geometric Algebra is simply a sphere with 0 radius). Type `Sphere` is defined as follows:

```
specialization: blade Sphere(e0=1, e1, e2, e3, einf);
```

Type `Sphere` means a linear combination of basis blades with the coefficient of e_0 being 1. For the second sphere S_2 with center at the origin e_0, we use the type `originSphere`:

```
specialization: blade originSphere(e0, einf);
```

since `e0-0.5f* d1* d1*einf` needs only the blades `e0` and `einf`. The result of the intersection of the spheres $Z_e = S_1 \wedge S_2$ is of type `Circle` defined as follows:

```
specialization: blade Circle(e0^e1, e0^e2, e0^e3, e1^einf,
e2^einf, e3^einf, e0^einf);
```

Please find the corresponding Gaigen 2 implementation for the **computation of the swivel plane** as follows :

```
originLine l_sw = _originLine(dual(e0^pw^einf));

originPlane SwivelPlane = _originPlane(dual(pz^pw^e0^einf));
```

Please notice that `pz` is defined as constant equal to e_3.

```
quaternion SwivelRot = _quaternion(cos( SAngle/2)
+sin( SAngle/2) * (l_sw*(1.0f/_float(GAnorm(l_sw)))));

SwivelPlane = _originPlane(SwivelRot*SwivelPlane*reverse(SwivelRot));

pointPair pp2 = _pointPair(dual(Z_e^SwivelPlane));

Sphere p_e = _Sphere(
-(-sqrt(_float(GAnorm(lcont(pp2,pp2))))+pp2)*
(inverse(lcont(einf,pp2))));
```

Line L_{sw} goes through the origin, its algebraic representation is an Euclidean bivector. We define `originLine` as follows:

```
specialization: blade originLine(e1^e2, e2^e3, e3^e1);
```

The plane π_{swivel} is a linear combination of the blades defined by the type `originPlane`:

```
specialization: blade originPlane(e1, e2, e3);
```

We do not define it as `Plane`, because the distance between π_{swivel} and origin is 0, that means the e_∞ part can be omitted. For contrast, we give the definition of an arbitrary `Plane` below:

```
specialization: blade Plane(e1, e2, e3, einf);
```

The quaternion Q_{swivel} matches the definition of `quaternion`,

```
specialization: versor quaternion(1.0, e1^e2, e2^e3, e3^e1);
```

point pair pp_2 matches the definition of `pointPair`, and the point P_e is assigned to `Sphere` naturally.

Computation of the elbow quaternion Q_e

This step is based on section 7.2.3. Please find the Gaigen 2 implementation as follows:

```
originLine l_se = _originLine(dual(e0^pe^einf));
Line l_ew = _Line(dual((p_e^pw^einf)));
```

Line L_{se} goes through the origin, so it is of type `originLine`:

```
specialization: blade originLine(e1^e2, e2^e3, e3^e1);
```

Line L_{ew} does not go through the origin. We define the type `Line` as follows:

```
specialization: blade Line(e1^e2, e1^e3, e2^e3,
e1^einf,e2^einf,e3^einf);
```

```
mv::Float cosi = (-_Float(scp(l_se,l_ew)))/( d1* d2);
mv::Float cos_2 = sqrt((1+cosi)/2.0);
mv::Float sin_2 = sqrt((1-cosi)/2.0);
quaternion_i q_e = _quaternion_i(cos_2+sin_2*quati);
```

Here please pay attention to the difference between L_{se} and L_{ew}. `quati` is defined as a constant for the quaternion i. Quaternion Q_e describes a rotation around the x axis only using $e_3 \wedge e_2$ as blade. This matches the definition of `quaternion_i`:

```
specialization: versor quaternion_i(1.0, e3^e2);
```

Rotate to the elbow position

Please find the Gaigen 2 implementation for the computation of Q_{12} as follows :

```
Sphere p_ze = _Sphere( d1*e3);
originPlane pi_m = _originPlane(p_ze-p_e);
originPlane pi_e = _originPlane(dual(p_zeˆp_eˆe0ˆeinf));
originLine l_m = _originLine(pi_eˆpi_m);
pureQuaternion q_12 =_pureQuaternion
    (l_m*(1.0f/_float(GAnorm(l_m))));
```

Point P_{ze} is assigned to type `Sphere`. Plane π_m and π_e are NOT assigned to type `Plane`, because they go through the origin (the coefficient of e_∞ is 0). This matches the definition of multivector type `originPlane`. The intersection line L_m goes through the origin, so it is assigned to type `originLine`. Quaternion Q_{12} has no scalar part, so it is assigned to type `pureQuaternion`:

```
specialization: blade pureQuaternion (e3ˆe2, e1ˆe3, e2ˆe1);
```

Rotate to the wrist location

Table 7.3: Computation of the shoulder quaternion

$$\begin{array}{|l|}\hline \pi_{yz2} = Q_{12} * e_1 * \tilde{Q_{12}} \\ c_3 = \frac{\pi_{yz2} \cdot \pi_{swivel}}{|\pi_{yz2}||\pi_{swivel}|} \\ sign = \pi_{yz2} \cdot P_w \\ Q_3 = \sqrt{\frac{1+c_3}{2}} + \sqrt{\frac{1-c_3}{2}} k \\ Q_s = Q_{12} * Q_3 \\ \hline \end{array}$$

Please find the Gaigen 2 implementation according to table 7.3 as follows (in the first formula we use directly e_1 instead of π_{yz}):

```
originPlane plane_yz2 = _originPlane(q_12*e1*reverse(q_12));
cosi=computeCos(plane_yz2,SwivelPlane);
mv:: Float sign= _float(scp(plane_yz2,pw));
sign = sign/abs(sign);
cos_2 = sqrt((1+cosi)/2.0)*sign;
sin_2 = sqrt((1-cosi)/2.0);
quaternion_k q_3 = _quaternion_k(cos_2+sin_2*quatk);
q_s = q_12 * q_3;
```

Plane π_{yz2} is NOT assigned to type `Plane`, because it has the distance 0 to the origin. So the coefficient of e_∞ is 0. This matches the definition of multivector type `originPlane`. Quaternion Q_3 describes a rotation around the z axis, so it is assigned to type `quaternion_k`:

```
specialization: versor quaternion_k(1.0, e2^e1);
```

and quaternion Q_s is assigned to the quaternion type.

The above mentioned versor multiplication rotating the plane π_{yz} with the help of a quaternion Q_{12} can be optimized as follows . The C++ code for this equation reads:

```
Plane plane_yz2 = applyVersor(q_12, e1);
```

whereby **q_12** is a quaternion and **e1** is a constant.

The final result is then emitted as the following optimized C++ function:

```
inline Plane applyVersor(
    const quaternion& V, const __e1_ct__& X)
{
  return Plane(
    V.c[0]*V.c[0] - V.c[1]*V.c[1] +
    V.c[2]*V.c[2] - V.c[3]*V.c[3] ,
    -2*V.c[0]*V.c[1] + 2*V.c[2]*V.c[3],
     2*V.c[2]*V.c[1] + 2*V.c[0]*V.c[3]);
}
```

7.3.2 Optimizations with Maple

We use Maple in order to get the most elementary relationship between the input and output parameters of our inverse kinematics algorithm (see table 7.4). The specific technique is described in section 2.13.3 on page 33.

Inverse kinematics algorithm in Maple

The goal of our inverse kinematics algorithm is to compute the output parameters Q_s and Q_e based on the input parameters (see table 7.4). In this section we present the Maple code of the inverse kinematics algorithm :

- **Compute the elbow point** P_e

Table 7.4: Input/output parameters of the inverse kinematics algorithm

parameter	**Maple**	**meaning**
P_w	pw(pwx, pwy, pwz)	target point of wrist
ϕ	sangle	swivel angle
d_1, d_2	d1, d2	length of the forearm and the upper arm
Q_s	qs	shoulder quaternion
Q_e	qe	elbow quaternion

```
> pw:=pwx*e1+pwy*e2+pwz*e3+0.5*
      (pwx^2+pwy^2+pwz^2)*einf+e0;
> S1:=pw-0.5*d2*d2*einf;
> S2:=e0-0.5*d1*d1*einf;
> Z_e:=S1 &w S2;
> // now compute the swivel plane ...
> l_sw:=-(e0 &w pw &w einf)&c e12345; // dualization operation
> pi_swivel:=-(pz &w pw &w e0 &w einf)
            &c e12345;
> norm_l_sw:=sqrt(l_sw &c reversion(l_sw));
> q_swivel:=cos(sangle/2)+sin(sangle/2)
           *(l_sw / norm_l_sw);
> pi_swivel:=q_swivel &c pi_swivel
            &c reversion(q_swivel);
> PP:=-(Z_e &w pi_swivel) &c e12345;
> PP:=vectorpart(PP,2);
> einf_PP:=LC(einf,PP);
> norm_einf_PP:=einf_PP &c
                reversion(einf_PP);
> inv_einf_PP:=einf_PP/norm_einf_PP;
> p_e:=-(-sqrt(scalarpart(LC(PP,PP)))
      +PP) &c inv_einf_PP;
> p_e:=vectorpart(p_e,1);
```

- **Compute the quaternion Q_e at the elbow joint**

```
> l_se:=-(e0 &w p_e &w einf)&c e12345;
```

```
> l_ew:=-(p_e &w pw &w einf)&c e12345;
> c4:=-LC(l_se,l_ew)/(d1*d2)/Id;
> qe:=sqrt((1+c4)/2)+sqrt((1-c4)/2)*(-qi);
```

- **Rotate to the elbow position**

```
> p_ze:=d1*e3+0.5*d1^2*einf+e0;
> pi_m:= p_ze-p_e;
> pi_e:=-(p_ze &w p_e &w e0 &w einf)&c e12345;
> l_m:=pi_e &w pi_m;
> q12:=l_m/(sqrt(l_m &c reversion(l_m)));
```

- **Rotate to the wrist location**
  We compute the quaternion Q_s at the shoulder joint. It will let the robot wrist reach the given target P_w:

```
> pi_yz2:=q_12 &c e1 &c reversion(q_12);
> _sign:=scalarpart(LC(pw,pi_yz2));
> _sign:=_sign/abs(_sign);
> norm_pi_swivel:=sqrt(pi_swivel &c
                  reversion(pi_swivel));
> c3:=scalarpart(-LC(pi_yz2,pi_swivel))
      /(norm_pi_swivel);
> q3:=sqrt((1+c3)/2)+sqrt((1-c3)/2)
      *_sign*qk;
> qs:=q12 &c q3;
```

  The quaternions Q_s and Q_e are the required results of our algorithm.

Optimized inverse kinematics algorithm

With the help of Maple our geometric algebra formulas are simplified and combined to very efficient expressions because of the symbolic computation feature of Maple. For instance, for the first lines of the algorithm of section 7.3.2 we get a result as follows:

```
Z_e = 0.5*(1-d1^2)*(pwx*e15+pwy*e25+pwz*e35)-
      0.5*(1+d1^2)*(pwx*e14+pwy*e24+pwz*e34)+
      0.5*e45*(pwx^2+pwy^2+pwz^2+d1^2-d2^2)
```

with only some simple multiplications and additions.

$$|L_{sw}| = \sqrt{p_{w_x}^2 + p_{w_y}^2 + p_{w_z}^2}. \tag{7.34}$$

The coefficients of the swivel plane are:

$$\begin{aligned}
\pi_{swivelx} =& (2\cos\frac{\phi}{2}\sin\frac{\phi}{2}p_{w_z}p_{w_x} - p_{w_y}|L_{sw}| + \\
& 2p_{w_y}|L_{sw}|\cos\frac{\phi}{2}^2)/|L_{sw}| \\
\pi_{swively} =& (2\cos\frac{\phi}{2}\sin\frac{\phi}{2}p_{w_z}p_{w_y} + p_{w_x}|L_{sw}| - \\
& 2p_{w_x}|L_{sw}|\cos\frac{\phi}{2}^2)/|L_{sw}| \\
\pi_{swivelz} =& \frac{-2\sin\frac{\phi}{2}\cos\frac{\phi}{2}(p_{w_x}^2 + p_{w_y}^2)}{|L_{sw}|}
\end{aligned} \tag{7.35}$$

The coefficients of the point pair PP

$$\begin{aligned}
PP_i &= \frac{1}{2}\pi_{swivelx}(p_{w_x}^2 + p_{w_z}^2 + p_{w_y}^2 + d_1^2 - d_2^2) \\
PP_j &= \frac{1}{2}\pi_{swively}(p_{w_x}^2 + p_{w_z}^2 + p_{w_y}^2 + d_1^2 - d_2^2) \\
PP_k &= \frac{1}{2}\pi_{swivelz}(p_{w_x}^2 + p_{w_z}^2 + p_{w_y}^2 + d_1^2 - d_2^2) \\
PP_{14} &= \frac{1}{2}(1 - d_1^2)(p_{w_y}\pi_{swivelz} - p_{w_z}\pi_{swively}) \\
PP_{15} &= \frac{1}{2}(1 + d_1^2)(p_{w_z}\pi_{swively} - p_{w_y}\pi_{swivelz}) \\
PP_{24} &= \frac{1}{2}(1 - d_1^2)(p_{w_z}\pi_{swivelx} - p_{w_x}\pi_{swivelz}) \\
PP_{25} &= \frac{1}{2}(1 + d_1^2)(p_{w_x}\pi_{swivelz} - p_{w_z}\pi_{swivelx}) \\
PP_{34} &= \frac{1}{2}(d_1^2 - 1)(p_{w_y}\pi_{swivelx} - p_{w_x}\pi_{swively}) \\
PP_{35} &= \frac{1}{2}(1 + d_1^2)(p_{w_y}\pi_{swivelx} - p_{w_x}\pi_{swively})
\end{aligned} \tag{7.36}$$

Extract the elbow point P_e from the point pair PP:

$$\begin{aligned}
einf_PP =&(PP_{35} - PP_{34})^2 + (PP_{14} - PP_{15})^2 \\
&+ (PP_{25} - PP_{24})^2 \\
tmp_1 =&- PP_i^2 - PP_j^2 - PP_k^2 - PP_{14}^2 + \\
&PP_{15}^2 - PP_{24}^2 + PP_{25}^2 - PP_{34}^2 + PP_{35}^2 \\
tmp_{sqrt} =&\sqrt{tmp_1} \\
p_{ex} =&(PP_j(PP_{34} - PP_{35}) + PP_k(PP_{25} - PP_{24} \\
&+ tmp_{sqrt}(PP_{15} - PP_{14}))/einf_PP \\
p_{ey} =&(PP_i(PP_{35} - PP_{34}) + PP_k(PP_{14} - PP_{15}) \\
&+ tmp_{sqrt}(PP_{25} - PP_{24}))/einf_PP \\
p_{ez} =&(PP_j(PP_{15} - PP_{14}) + PP_i(PP_{24} - PP_{25}) \\
&+ tmp_{sqrt}(PP_{35} - PP_{34}))/einf_PP
\end{aligned} \tag{7.37}$$

The quaternion Q_e of the rotation at the elbow joint:

$$\begin{aligned}
Q_e =&\sqrt{\frac{1 + \frac{p_{e_x}^2 - p_{e_x}p_{w_x} + p_{e_y}^2 - p_{e_y}p_{w_y} - p_{e_z}p_{w_z} + p_{e_z}^2}{d_1 d_2}}{2}} + \\
&\sqrt{\frac{1 - \frac{p_{e_x}^2 - p_{e_x}p_{w_x} + p_{e_y}^2 - p_{e_y}p_{w_y} - p_{e_z}p_{w_z} + p_{e_z}^2}{d_1 d_2}}{2}} e_{23}
\end{aligned} \tag{7.38}$$

The result for the quaternion Q_{12} is:

$$\begin{aligned}
tmp_2 =&d_1^4 p_{e_y}^2 - 2d_1^3 p_{e_y}^2 p_{e_z} + d_1^2 p_{e_y}^2 p_{e_z}^2 + \\
&d_1^4 p_{e_x}^2 - 2d_1^3 p_{e_x}^2 p_{e_z} + d_1^2 p_{e_x}^2 p_{e_z}^2 + \\
&d_1^2 p_{e_x}^4 + 2d_1^2 p_{e_x}^2 p_{e_y}^2 + d_1^2 p_{e_y}^4 \\
|L_m| =&\sqrt{tmp_2} \\
q_{12_i} =&\frac{d_1 p_{e_x}(d_1 - p_{e_z})}{|L_m|} \\
q_{12_j} =&\frac{d_1 p_{e_y}(d_1 - p_{e_z})}{|L_m|} \\
q_{12_k} =&\frac{d_1(p_{e_x}^2 + p_{e_y}^2)}{|L_m|}
\end{aligned} \tag{7.39}$$

The last rotation at the shoulder joint is c_3:

$$\begin{aligned}
c_3 =&(\pi_{swivelx}({q_{12}}_k^2 + {q_{12}}_j^2 - {q_{12}}_i^2) - \\
&2q_{12_i}(q_{12_j}\pi_{swively} + q_{12_k}\pi_{swivelz})) \\
&/\sqrt{{\pi_{swivel}}_x^2 + {\pi_{swivel}}_y^2 + {\pi_{swivel}}_z^2} \\
sign =&p_{wx}({q_{12}}_i^2 - {q_{12}}_k^2 - {q_{12}}_j^2) + \\
&2q_{12_i}(q_{12_k}p_{wz} + q_{12_j}p_{wy}) \\
sign =&\frac{sign}{|sign|} \\
q_{3_{scalar}} =&\sqrt{\frac{1+c_3}{2}} \\
q_{3_k} =&\sqrt{\frac{1-c_3}{2}}sign
\end{aligned} \tag{7.40}$$

The final result of Q_s is:

$$\begin{aligned}
Q_s =& - q_{12_k} \cdot q_{3_k} - \\
&(q_{12_i} \cdot q_{3_{scalar}} + q_{12_j} \cdot q_{3_k})e_{23} - \\
&(q_{12_i} \cdot q_{3_k} - q_{12_j} \cdot q_{3_{scalar}})e_{13} - \\
&(q_{12_k} \cdot q_{3_{scalar}})e_{12}
\end{aligned} \tag{7.41}$$

Based on these results of the Maple optimization process we can code them into C/C++ easily. For better computation efficiency some frequently used given variables should be defined as constant and some repeatedly computed expressions should be assigned to help variables.

7.4 Results

At first we developed and simulated our algorithm on a high level based on CLUCalc (see [60]), developed by Christian Perwass. Then, we had to implement it on the target platform Avalon ([90]), a virtual reality system written in C++ using Visual Studio.NET 2003. Previously, inverse kinematics was implemented using IKAN [80], a widely used C++ library.

Our Maple approach outperformed the IKAN implementation clearly. It turned out to be about 3.3 times faster. Based on this approach we are able to design and test our

algorithms on a high level. When we are satisfied with our algorithm we are able to transfer it to C/C++ without the need of additional libraries.

Also our Gaigen 2 approach outperformed IKAN in a similar way. The Conformal Geometric Algebra based algorithm is 43 % faster than IKAN, and even 240 % faster when the conversion from matrices to quaternions is taken into account. Based on the Gaigen 2 approach we are able to implement our algorithms in a way that still reflects the elegant features of Conformal Geometric Algebra.

In a nutshell, we do not only provide a rapid prototyping approach based on elementary objects but also highly efficient implementations.

Chapter 8

Conclusion

The main result of this thesis is that **"Geometric Computing using Conformal Geometric Algebra"** is not only leading to elegant and geometrically intuitive algorithms but that these applications can also be implemented very efficiently. Please find hereafter the benefits in more detail:

- We investigated quaternions with regard to the direct usage for algorithms in Conformal Geometric Algebra. In the computer animation algorithm of chapter 7 we could use them directly embedded in the inverse kinematics algorithm. This is the reason why we could avoid to translate between transformation matrices and quaternions. In chapter 3 we furthermore showed that Conformal Geometric Algebra is able to unify a lot of additional mathematical systems like imaginary numbers, Plücker coordinates, dual numbers and dual quaternions. From the knowledge point of view, you have to learn only one mathematical system, you do not have to learn translations between different systems and you are able to get more intuitively new insights in all areas of engineering.

- We analyzed the role of infinity in Conformal Geometric Algebra for a better understanding of its geometric entities like spheres and planes. This analysis in chapter 4 is also helpful for a better understanding of the nature of Conformal Geometric Algebra operators like rotors and translators.

- The just mentioned basic objects sphere and plane could be used advantageously for the approximation of point sets by these objects in chapter 5. Very helpful was the application of the inner product of Conformal Geometric Algebra as a measure of distance.

- Chapter 6 showed the rapid prototyping possibility of algorithms based on Conformal Geometric Algebra with the help of the inverse kinematics and the grasping process of a robot. The interactive and visual approach based on CLUCalc is able to lead to a remarkable reduction of development time for new algorithms. These algorithms are very intuitive and compact because of easy computations of geometric objects like spheres or circles as well as because of the easy handling of transformations.

- Chapter 7 presented an inverse kinematics algorithm in Conformal Geometric Algebra and the proof of its efficience. Often a clear structure and greater elegance result in lower runtime performance. However, we could show that our inverse kinematics application of a virtual character based on Conformal Geometric Algebra is now even faster than conventional algorithms. Because of their compactness Conformal Geometric Algebra algorithms are easy to implement and easy to process in Gaigen 2 and Maple. The two approaches have different advantages: Based on the Gaigen 2 approach we are able to implement our algorithms in a way that still reflects the elegant features of Conformal Geometric Algebra. Based on the Maple approach we are able to implement them with the help of our standard compilers without the need of additional libraries.

All of these properties lead to enhanced quality of the algorithms that are also easier to understand and better to maintain. With these results, we are convinced that **Geometric Computing based on Conformal Geometric Algebra** will become more and more widely accepted in a great variety of applications in computer graphics and robotics.

Chapter 9

Future work

There is a wide range of possibilities for new applications of **Geometric Computing based on Conformal Geometric Algebra**. In order to further improve the runtime performance of Conformal Geometric Algebra solutions we intend to evaluate hardware solutions. From the application point of view, we will mainly focus on virtual kinematics, on game engines as well as on dynamics for robotics and computer animations.

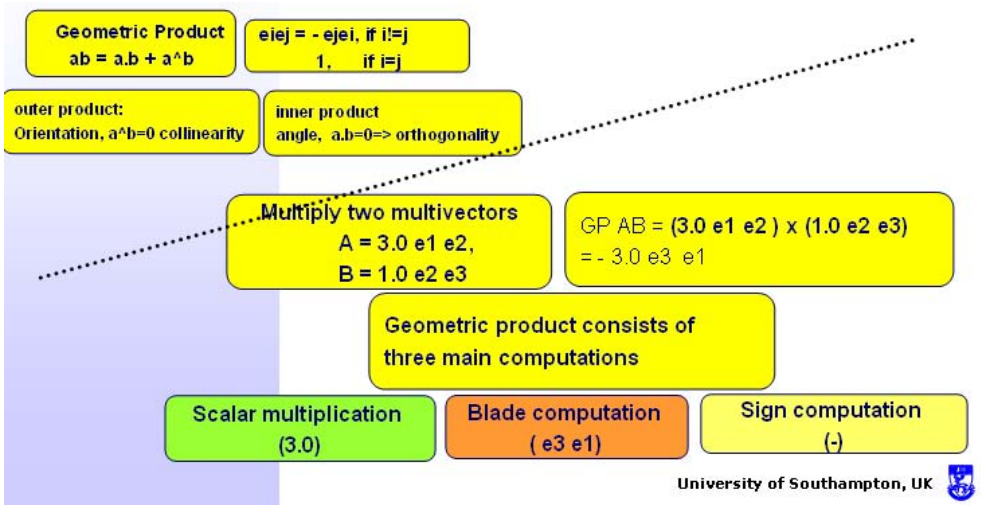

Figure 9.1: Architecture of a Geometric Algebra hardware

9.1 Hardware solution

From the efficiency point of view, we intend to investigate hardware solutions for Geometric Algebra computations. Currently there are three hardware solutions, one from Germany ([62]), one from Italy ([3]) and one from UK ([56]). The hardware solution

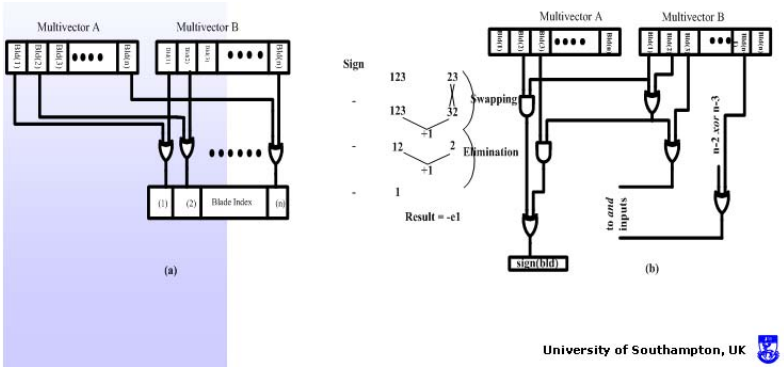

Figure 9.2: Multivector hardware representation

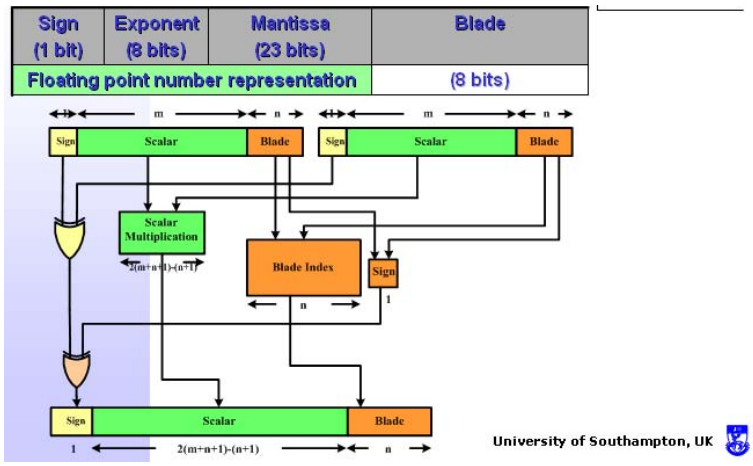

Figure 9.3: Multivector hardware representation details

from the University of Southampton, UK seems to be the most interesting one since this is the only floating point solution. The researchers from Southampton have built a scalable n-dimensional(n=8) geometric algebra processor core architecture realizable using both ASIC (Application Specific Integrated Circuit) and FPGA(Field Programmable

Gate Array) platform. The FPGA platform is an obvious choice for evaluating algorithms and implementations. However, the over-all goal is performance and FPGAs just do not deliver fast enough throughput generally to improve over conventional dedicated hardware solutions in particular for computer graphics applications. Therefore, we - as also the developers do - believe that a dedicated ASIC platform is necessary to demonstrate a credible practical alternative to classical implementations.

The mathematical representation of the architecture is shown in figure 9.1. The geometric product consists of three main computations, a scalar multiplication, a blade computation and a sign computation.

Figure 9.2 shows the representation of a multivector consisting of a linear combination of components as shown in figure 9.3 consisting of a sign, an exponent, a mantissa and a blade. Their multiplication is done for the sign, the scalar part and the blade part separately. The sign computation is based on swapping of basic blades.

9.2 Virtual kinematics

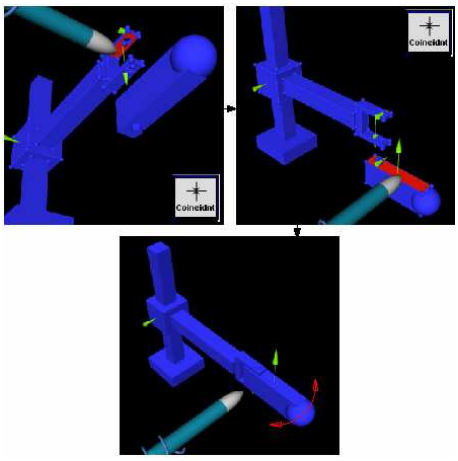

Figure 9.4: Virtual kinematics

From the kinematics point of view we will focus on so-called virtual kinematics [47]. With this system you are able to virtually assemble and simulate all kind of kinematic mechanisms like robots. The described properties of Conformal Geometric Algebra can be used in order to design a new general approach for inverse kinematics algorithms. A wide class of kinematic chains can be handled by a single general inverse kinematics algorithm. Starting from a general description of a wide class of kinematic mechanisms,

general solutions are to be developed based on this method. This model will make it easy to extend the class of kinematic mechanisms. You only have to maintain one algorithm for many kinematic mechanisms that can be globally optimized.

9.3 Game engines

We expect a lot of advantages for game engines using Conformal Geometric Algebra. The main question concerning game engines is: what kind of game engine functionality can benefit the most of Conformal Geometric Algebra? Benefits could be easy programming, the unified mathematical system, robustness, runtime performance etc. Good candidates are collision detection and kinematics.

9.4 Dynamics

The physically correct handling of dynamics is especially important for robotics but also for computer animations and simulations. First results show that also from the differential kinematics point of view, we are able to work very intuitively with Conformal Geometric Algebra. While we are able to describe both spheres and planes as vectors we are additionally able to describe rotations and translations as well as rotational and linear velocity with the help of unified algebraic expressions. The expression for the Jacobian immediately relates to geometric entities like lines for the rotation axis of the rotational velocity and planes with the well known velocity vector as a normal. For the

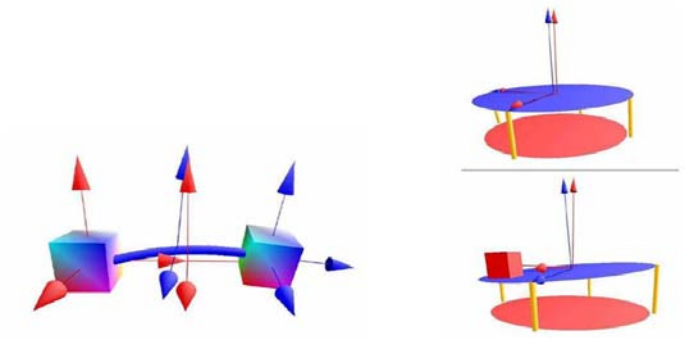

Figure 9.5: Elasticity based on Conformal Geometric Algebra

handling of elasticity we will benefit from the master thesis [45]. See figure 9.5 for some simulation results.

Bibliography

[1] The homepage of geomerics ltd. HTML document http://www.geomerics.com.

[2] Plucker coordinates. HTML document http://en.wikipedia.org.

[3] F. Sorbello G. Vassallo S. Vitabile A. Gentile, S. Segreto and V. Vullo. Cliffosor, an innovative fpga-based architecture for geometric algebra. In *ERSA 2005*, pages 211–217, 2005.

[4] M. Alexa. Linear combination of transformations. In *Proceedings of ACM SIGGRAPH 2002*, 2002.

[5] R. Abłamowicz. Clifford algebra computations with maple. In W. E. Baylis, editor, *Clifford (Geometric) Algebras*, pages 463–501. Birkhäuser, Boston, 1996.

[6] R. Abłamowicz and B. Fauser. The homepage of the package cliffordlib. HTML document http://math.tntech.edu/rafal/cliff9/, 2005. Last revised: September 17, 2005.

[7] D. Cremers B. Rosenhahn, T. Brox and H.-P. Seidel. A comparison of shape matching methods for contour based pose estimation. In *11th International Workshop on Combinatorial Image Analysis*, 2006.

[8] U. Kersting D. Smith J. Gurney R. Klette B. Rosenhahn, T. Brox. A system of marker-less human motion estimation. *Kuenstliche Intelligenz(KI)*, 2006.

[9] Paolo Baerlocher and Ronan Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer*, 20(6):402–417, 2004.

[10] E. Bayro-Corrochano. Robot perception and action using conformal geometry. In E. Bayro-Corrochano, editor, *the Handbook of Geometric Computing. Applications in Pattern Recognition, Computer Vision, Neurocomputing and Robotics*, chapter 13, pages 405–458. Springer Verlag, Heidelberg, 2005.

[11] E. Bayro-Corrochano, K. Daniilidis, and G. Sommer. Motor algebra for 3d kinematics : The case of the hand-eye calibration. *Journal of Mathematical Imaging and Vision*, 13:79–99, 2000.

[12] E. Bayro-Corrochano and G. Sobczyk, editors. *Geometric Algebra with Applications in Science and Engineering*. Birkhäuser, 2001.

[13] E. Bayro-Corrochano and J. Zamora-Esquivel. Inverse kinematics, fixation and grasping using conformal geometric algebra. In *IROS 2004, September 2004, Sendai, Japan*, 2004.

[14] E. Bayro-Corrochano and J. Zamora-Esquivel. Kinematics and differential kinematics of binocular robot heads. In *proceedings of ICRA conference, Orlando, USA*, 2006.

[15] J. Browne. The grassmannalgebra book home page. HTML document, 2002. Last visited 15. Sept. 2003.

[16] Jonathan Cameron and Joan Lasenby. Oriented conformal geometric algebra. *Proceedings of ICCA7*, 2005.

[17] William K. Clifford. On the classification of geometric algebras. In R. Tucker, editor, *Mathematical Papers*, pages 397–401. Macmillian, London, 1882.

[18] A. Differ. The Clados home page. HTML document, 2002. Last visited 15. Sept. 2003.

[19] Chris Doran and Anthony Lasenby. *Geometric Algebra for Physicists*. Cambridge University Press, 2003.

[20] L. Dorst, C. Doran, and J. Lasenby, editors. *Applications of Geometric Algebra in Computer Science and Engineering*. Birkhäuser, 2002.

[21] L. Dorst and D. Fontijne. 3d euclidean geometry through conformal geometric algebra (a gaviewer tutorial). *available from http://www.science.uva.nl/ga*, 2003.

[22] L. Dorst and S. Mann. Geometric algebra: a computational framework for geometrical applications (part i: algebra). *Computer Graphics and Application*, 22(3):24–31, May/June 2002.

[23] Leo Dorst. Honing geometric algebra for its use in the computer sciences. In G. Sommer, editor, *Geometric Computing with Clifford Algebra*. Springer-Verlag, 2001.

[24] P. Fleckenstein. C++ template classes for geometric algebras. *Available at http://www.nklein.com/products/geoma*.

[25] D. Fontijne, T. Bouma, and L. Dorst. Gaigen: A geometric algebra implementation generator. *Available at http://www.science.uva.nl/ga/gaigen*.

[26] D. Fontijne and L. Dorst. Performance and elegance of 5 models of geometry in a ray tracing application. *Software and other downloads available at http://www.science.uva.nl/∼fontijne/raytracer*, 2002.

[27] D. Fontijne and L. Dorst. Modeling 3D euclidean geometry. *IEEE Computer Graphics and Applications*, 23(2):68–78, MarchApril 2003.

[28] C. F. van Loan G. H. Golub. *Matrix Computations*. The Johns Hopkins University Press, Baltimore and London, 1996.

[29] Holger Griesheimer. Inverse kinematik auf basis von conformaler geometrischer algebra. Master's thesis, FH Darmstadt, 2005.

[30] D. Hestenes. *New Foundations for Classical Mechanics*. Dordrecht, 1986.

[31] D. Hestenes. Old wine in new bottles : A new algebraic framework for computational geometry. In E. Bayro-Corrochano and G. Sobczyk, editors, *Geometric Algebra with Applications in Science and Engineering*. Birkhäuser, 2001.

[32] D. Hestenes and G. Sobczyk. *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. Dordrecht, 1984.

[33] David Hestenes. The Geometric Calculus home page. HTML document http://modelingnts.la.asu.edu/.

[34] David Hestenes and Renatus Ziegler. Projective Geometry with Clifford Algebra. *Acta Applicandae Mathematicae*, 23:25–63, 1991.

[35] D. Hildenbrand. Geometric computing in computer graphics using conformal geometric algebra. *Computers & Graphics*, 29(5):802–810, 2005.

[36] D. Hildenbrand, E. Bayro-Corrochano, and J. Zamora-Esquivel. Advanced geometric approach for graphics and visual guided robot object manipulation. In *proceedings of ICRA conference, Barcelona, Spain*, 2005.

[37] D. Hildenbrand, D. Fontijne, C. Perwass, and L. Dorst. Tutorial geometric algebra and its application to computer graphics. In *Eurographics conference Grenoble*, 2004.

[38] D. Hildenbrand, D. Fontijne, Yusheng Wang, M. Alexa, and L. Dorst. Competitive runtime performance for inverse kinematics algorithms using conformal geometric algebra. In *Eurographics conference Vienna*, 2006.

[39] D. Hildenbrand, J. Zamora-Esquivel, and E. Bayro-Corrochano. Inverse kinematics computation in computer graphics and robotics using conformal geometric algebra. In *ICCA7, 7th International Conference on Clifford Algebras and their Applications*, 2005.

[40] E. Hitzer. Interactive and animated geometric algebra with cinderella. http://sinai.mech.fukui-u.ac.jp/gcj/software/GAcindy/GAcindy.htm.

[41] E. Hitzer. Euclidean geometric objects in the clifford geometric algebra of Origin, 3-Space, Infinity. In *Bulletin of the Belgian Mathematical Society - Simon Stevin*, 2004.

[42] Martin Erik Horn. Grass, mann! das clifford-kinder-rechenbuch. In Arne Oberlaender Volkhard Nordmeier, editor, *Didaktik der Physik der DPG, Beitraege zur Fruehjahrstagung Duesseldorf 2004 (Beitrag 8.5), Tagungs-CD des Fachverbands, ISBN 3-86541-066-9, LOB - Lehmanns Media, Berlin 2004.*, 2004.

[43] Martin Erik Horn. Quaternionen und geometrische algebra. In *Didaktik der Physik der DPG, Beitraege zur Fruehjahrstagung Kassel 2006*, 2006.

[44] A. Lasenby J. Lasenby, E. Bayro-Corrochano and G. Sommer. A new methodology for computing invariants in computer vision. In *Proceedings of ICPR 96*, 1996.

[45] Thomas Kalbe. Beschreibung der dynamik elastisch gekoppelter koerper in konformaler geometrischer algebra. Master's thesis, TU Darmstadt, 2006.

[46] U. Kersting, B. Rosenhahn, H.-P. Seidel, and R. Klette. Tracking human motion without markers - opportunities for field testing in sports. In *5th World Congress of Biomechanics, Munich*, 2006.

[47] Virtual Kinematics. Virtual engineering project home page. HTML document http://www.igd.fraunhofer.de/igd-a2/projects/Kinematics/index.html.

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

[49] S. Mann L.Dorst, D. Fontijne and Morgan Kaufman. *Geometric Algebra for Computer Science, An Object-Oriented Approach to Geometry*. Morgan Kaufman, 2005.

[50] P. Leopardi. The GluCat home page. HTML document, 2002. Last visited 15. Sept. 2003.

[51] H. Li, D. Hestenes, and A. Rockwood. Generalized homogeneous coordinates for computational geometry. In G. Sommer, editor, *Geometric Computing with Clifford Algebra*, pages 27–59. Springer-Verlag, 2001.

[52] P. Lounesto. The CLICAL home page. HTML document, 1987. Last visited 15. Sept. 2003.

[53] S. Mann and L. Dorst. Geometric algebra: a computational framework for geometrical applications (part ii: applications). *Computer Graphics and Application*, 22(4):58–67, July/August 2002.

[54] S. Mann, L. Dorst, and T. Bouma. The making of GABLE, a geometric algebra learning environment in matlab. pages 491–511, 2001.

[55] The homepage of maple. http://www.maplesoft.com/products/maple. 615 Kumpf Drive, Waterloo, Ontario, Canada N2V 1K8.

[56] B. Mishra and P. Wilson. Hardware implementation of a geometric algebra processor core. In *Proceedings of IMACS International Conference on Applications of Computer Algebra (in press), Nara, Japan*, 2005.

[57] Naeve and A. Rockwood. Course 53 geometric algebra. In *Siggraph conference Los Angeles*, 2001.

[58] Alba Perez and M. McCarthy. Sizing a serial chain to fit a task trajectory using clifford algebra exponentials. In *proceedings of ICRA conference, Barcelona*, 2005.

[59] Alba Perez, M. McCarthy, and B. Bennet. Dual quaternion synthesis of constrained robots. In *proceedings of Advances in Robot Kinematics*, 2002.

[60] C. Perwass. The CLU home page. HTML document http://www.clucalc.info, 2005.

[61] C. Perwass and W. Förstner. Uncertain geometry with circles, spheres and conics. In R. Klette, R. Kozera, L. Noakes, and J. Weickert, editors, *Geometric Properties from Incomplete Data*, volume 31 of *Computational Imaging and Vision*, pages 23–41. Springer-Verlag, 2006.

[62] C. Perwass, C. Gebken, and G. Sommer. Implementation of a clifford algebra co-processor design on a field programmable gate array. In R. Ablamowicz, editor, *CLIFFORD ALGEBRAS: Application to Mathematics, Physics, and Engineering*, Progress in Mathematical Physics, pages 561–575. 6th Int. Conf. on Clifford Algebras and Applications, Cookeville, TN, Birkhäuser, Boston, 2003.

[63] C. Perwass, C. Gebken, and G. Sommer. Geometry and kinematics with uncertain data. In A. Leonardis, H. Bischof, and A. Pinz, editors, *9th European Conference on Computer Vision, ECCV 2006, May 2006, Graz, Austria*, number 3951 in LNCS, pages 225–237. Springer-Verlag, Berlin Heidelberg, 2006.

[64] C. Perwass and D. Hildenbrand. Aspects of geometric algebra in euclidean, projective and conformal space. Technical report, University of Kiel, 2004.

[65] C. Perwass and G. Sommer. The inversion camera model. In *28. Symposium für Mustererkennung, DAGM 2006, Berlin, 12.-14.09.2006*. Springer-Verlag, Berlin, Heidelberg, 2006.

[66] C.B.U. Perwass. *Applications of Geometric Algebra in Computer Vision*. PhD thesis, Cambridge Universtiy, 2000.

[67] C.B.U. Perwass and J. Lasenby. A Geometric Analysis of the Trifocal Tensor. In R. Kakarala R. Klette, G. Gimel'farb, editor, *Image and Vision Computing New Zealand, IVCNZ'98, Proceedings*, pages 157–162. The University of Auckland, 1998.

[68] C.B.U. Perwass and J. Lasenby. A Unified Description of Multiple View Geometry. In G. Sommer, editor, *Geometric Computing with Clifford Algebra*. Springer-Verlag, 2001.

[69] A. Rockwood, H. Li, and D. Hestenes. United states patent no. 6,853,964: System for encoding and manipulating models of objects. 2005.

[70] B. Rosenhahn. *Pose Estimation Revisited*. PhD thesis, Inst. f. Informatik u. Prakt. Mathematik der Christian-Albrechts-Universität zu Kiel, 2003.

[71] B. Rosenhahn and G. Sommer. Pose estimation in conformal geometric algebra. *Journal of Mathematical Imaging and Vision*, 22:27–70, 2005.

[72] Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics*, 23(3):514–521, August 2004.

[73] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators*. Springer, 2000.

[74] Hyun Joon Shin, Jehee Lee, Michael Gleicher, and Sung Yong Shin. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics*, 20(2):67–94, April 2001.

[75] Cristian Sminchisescu and Bill Triggs. Kinematic jump processes for monocular 3d human tracking. In *2003 Conference on Computer Vision and Pattern Recognition (CVPR 2003)*, pages 69–76, June 2003.

[76] G. Sommer, editor. *Geometric Computing with Clifford Algebra*. Springer Verlag, 2001.

[77] G. Sommer. Applications of geometric algebra in robot vision. In H. Li, P.J. Olver, and G. Sommer, editors, *Computer Algebra and Geometric Algebra with Applications*, volume 3519 of *LNCS*, pages 258–277. 6th International Workshop IWMM 2004, Shanghai, China and International Workshop GIAE 2004, Xian, China, Springer-Verlag, Berlin Heidelberg, 2005.

[78] G. Sommer, B. Rosenhahn, and C. Perwass. The twist representation of free-form objects. In *Geometric Properties from Incomplete Data*, volume 31 of *Computational Imaging and Vision*, pages 3–22. Springer-Verlag, 2006.

[79] J. Suter. Clifford. *Used to be available at http://www.jaapsuter.com*, 2003.

[80] Deepak Tolani, Ambarish Goswami, and Norman I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, 62(5):353–388, September 2000.

[81] Maria Cruz Villa Uriol, Alba Perez, and Falko Kuester. Humanoid synthesis using clifford algebra. In *proceedings of ICRA conference, Orlando, USA*, 2006.

[82] VHuman. The Virtual Human Project home page. HTML document www.virtualhuman.de.

[83] Yusheng Wang. Algorithm and performance of the grasping movement of a human-arm-like chain based on conformal geometric algebra. Master's thesis, TU Darmstadt, 2006.

[84] Rich Wareham, Jonathan Cameron, and Joan Lasenby. Applications of conformal geometric algebra in computer vision and graphics. *Lecture Notes in Computer Science*, 3519:329–349, June 2005.

[85] Rich Wareham and Joan Lasenby. Applications of conformal geometric algebra in computer vision and graphics. *submitted to ACM Transactions on Graphics*, 2004.

[86] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques*. Addison-Wesley, 1992.

[87] Katsu Yamane. *Simulating and Generating Motions of Human Figures*. Springer, 2004.

[88] M. Zaharia and L. Dorst. Modeling and visualization of 3d polygonal mesh surfaces using geometric algebra. *Computers & Graphics*, 29(5):802–810, 2003.

[89] M.D. Zaharia and L. Dorst. Interface specification and implementation internals of a program module for geometric algebra. *Journal of Logic and Algebraic Programming*, 2003.

[90] ZGDV. The Avalon home page. HTML document http://www.zgdv.de/avalon/.

[91] Haidan Zhang. Grasping strategies for a virtual barrett hand based on conformal geometric algebra. Master's thesis, TU Darmstadt, 2006.

[92] Jun Zhao. Motion optimization of kinematic chains based on dynamics parameters in conformal geometric algebra. Master's thesis, TU Darmstadt, 2005.

Appendix A

Akademischer Werdegang

Dietmar Hildenbrand hat von Okt. 1981 bis Apr. 1986 an der TH Darmstadt Informatik mit Abschluß Diplom studiert. Danach hat er verschiedene Positionen in der Industrie ausgefüllt speziell im Bereich der Leitung von Software-Projekten.

Seit Juli 2002 ist er wissenschaftlicher Mitarbeiter im Fachgebiet Graphisch interaktive Systeme der TU Darmstadt und seit Jan. 2006 Geschäftsführer des interdisziplinären Forschungsschwerpunkts Graphische Datenverarbeitung der TU Darmstadt.