Geometric Algebra and its Application to Computer Graphics

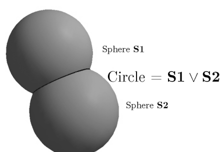
D. Hildenbrand[1] , D. Fontijne[2] , C. Perwass[3] and L. Dorst[2]

[1]Interactive Graphics Systems Group, TU Darmstadt, Germany
[2]Informatics Institute, University of Amsterdam, The Netherlands
[3] Institute of Computer Science, University Kiel, Germany

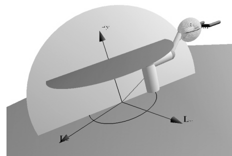
Abstract

Early in the development of computer graphics it was realized that projective geometry is suited quite well to represent points and transformations. Now, maybe another change of paradigm is lying ahead of us based on ***Geometric Algebra****. If you already use quaternions or Lie algebra in additon to the well-known vector algebra, then you may already be familiar with some of the algebraic ideas that will be presented in this tutorial. In fact, quaternions can be represented by Geometric Algebra, next to a number of other algebras like complex numbers, dual-quaternions, Grassmann algebra and Grassmann-Cayley algebra. In this half day tutorial we will emphasize that Geometric Algebra*

- *is a unified language for a lot of mathematical systems used in Computer Graphics,*
- *can be used in an easy and geometrically intuitive way in Computer Graphics.*



We will focus on the (5D) ***Conformal Geometric Algebra****. It is an extension of the 4D projective geometric algebra. For example, spheres and circles are simply represented by algebraic objects. To represent a circle you only have to intersect two spheres (or a sphere and a plane), which can be done with a basic algebraic operation. Alternatively you can simply combine three points (using another product in the algebra) to obtain the circle through these three points.*



Next to the construction of algebraic entities, kinematics can also be expressed in Geometric Algebra. For example, the inverse kinematics of a robot can be computed in an easy way. The geometrically intuitive operations of Geometric Algebra make it easy to compute the joint angles of a robot which need to be set in order for the robot to reach its goal.

Categories and Subject Descriptors (according to ACM CCS): G.4 [Mathematical Software]: Algorithm design and analysis, Efficiency I.3.7 [Computer Graphics]: Animation

© The Eurographics Association 2004.

Contents

Part 1: Overview

1 Outline

1.1 Agenda

2 Introduction

3 History of Geometric Algebra

4 Properties of Geometric Algebra

4.1 Geometric Intuitivity

4.2 Unification

4.3 Low Symbolic Complexity

5 Foundations of Conformal Geometric Algebra

5.1 Blades and Products

5.2 The Blades of the Conformal Geometric Algebra

Part 2: The Blades of the Conformal Model

6 Warming up to CGA

6.1 Points

6.2 Complementation: the dual

6.3 Intersection

7 Elementary objects

7.1 Rounds and flats

7.2 Tangent blades

7.3 Free blades (attitudes)

7.4 That's all

8 A visual explanation

9 Practicalities

9.1 Parameters

9.2 Example: Dissecting a Point Pair

9.3 Angles between flats

10 Construction by containment and orthogonality

11 Summary and conclusion

Part 3: A Mathematical Introduction toGeometric Algebra

12 Introduction to Geometric Algebra

12.1 The Outer Product

12.2 The Outer Product Null Space

© The Eurographics Association 2004.

12.3 Magnitude of Blades
12.4 The Inner Product
12.5 The Inverse of a Blade
12.6 Geometric Interpretation of Inner Product
12.7 The Inner Product Null Space
12.8 The Dual
12.9 Geometric Interpretation of the IPNS
12.10 The Meet Operation
12.11 The Geometric Product
12.12 Reflection
12.13 Rotation
13 Geometry
13.1 The Setup
13.2 Geometric Algebra on $\mathbb{PE}^n$
13.3 The Euclidean OPNS
13.4 The Euclidean IPNS
13.5 The Pinhole Camera Model
13.6 Reflections in Projective Space
13.7 Rotations in projective space

Part 4: Animation and Motion
14 CLUCalc example RotationAxis
15 Transformations
15.1 Rotors
15.2 Translators
15.3 Rigid Body Motion
16 Interpolation of motions
16.1 Screw Motion
16.2 Interpolation of twists
17 Kinematic chains
18 Inverse Kinematics
18.1 Step 1
18.2 Step 2
18.3 Step 3
18.4 Step 4
18.5 Computation of the joint angles
19 Dynamics

© The Eurographics Association 2004.

Part 5: Implementation and Performance of Geometric Algebra

20 Introduction

21 Issues in Efficiently Implementing a Numerical Geometric Algebra Package

22 Approaches to Implementing a Numerical Geometric Algebra Package

22.1 Matrix based

22.2 Run-time Configurable

22.3 Generative Programming

22.4 Direct Integration into Programming Language

22.5 Summary Pro / Cons of the approaches

23 Geometric Algebra Hardware

24 Performance and Elegance of Five Models of Geometry in a Ray Tracing Application

25 Conclusion

References

26 Biography

26.1 L. Dorst

26.2 D. Fontijne

26.3 D. Hildenbrand

26.4 C. Perwass

© The Eurographics Association 2004.

PART ONE

Overview

1. Outline

In this tutorial we will give an overview of Geometric Algebra and its application to Computer Graphics. First of all, we want to motivate the topic and give insights into some applications. In particular, the **Conformal Geometric Algebra** with its so-called 'conformal model' of 3-dimensional Euclidean geometry will be introduced. In this model, Euclidean objects and their interactions will be explored and visualized interactively.

With help of the conformal model we will describe **animations and motions**. It will be shown how it can be used quite advantageously to treat this kind of Computer Graphics application. We will give some basic visual examples and describe rigid body motions and their interpolations. We will focus on the inverse kinematics and dynamics of kinematic chains in order to describe motions of robots and human figures.

At the university of Amsterdam a **ray tracer** was developed in order to compare different geometric approaches from the **implementation and performance** point of view. Compared to linear algebra, the richer mathematical language of GA leads to more work for implementing the algebra, but **less work for implementing the application**. We discuss the issues in implementing a numerical Geometric Algebra package for a language like C++. We compare various existing implementations and look at their performance. We conclude with future implementation methods like SIMD hardware suitable for GA and generative programming.

During the tutorial only the **most fundamental mathematical aspects of Geometric Algebra** will be presented. This is possible, since most aspects of Geometric Algebra can be understood with geometric intuition. The actual mathematical 'inner workings' of the algebra will be detailed in an accompanying script that also contains many visual examples from the presentations.

The tutorial will be rounded off by an outlook into possible future applications of Geometric Algebra in Computer Graphics.

1.1. Agenda

- Motivation
  - some nice properties of GA (Dietmar Hildenbrand)
  - some applications
- Conformal model Tutorial (Daniel Fontijne)
  - Spanning rounds and flats from points
  - Dualization, intersection
  - The blades of the conformal model
  - Language: orthogonal specification
  - Towards versors: objects as operators
- Introduction to the mathematical foundations (Christian Perwass)
  - main characteristics of the algebra
  - how the algebra represents geometry
  - algebraic operations for reflection, rotation, intersection and inversion
- Animation and Motion (Dietmar Hildenbrand)
  - basic visual samples of transformations
  - rigid body motion
  - interpolation of motions
  - (forward and inverse) kinematics
  - dynamics
- Implementation and Performance (Daniel Fontijne)
  - issues in implementing Geometric Algebra
  - pros, cons, performance of various implementations
  - future directions: hardware, generative programming
- Summary and Future Prospects (Dietmar Hildenbrand)

2. Introduction

In this tutorial we will emphasize the advantages of a new mathematical system for all areas of Computer Graphics. Early in the development of Computer Graphics it was realized that projective geometry is suited quite well to represent points and transformations. This is done in 4D with help of an additional (homogenous) coordinate using 4D vectors to represent points and 4x4 matrices to express transformations. Now, maybe another change of paradigm is lying ahead of us. Mainly at the SIGGRAPH 2000 and 2001 a new unified mathematical language was introduced to the graphics community, the **Geometric Algebra**. In particular the Geometric Algebra of 5D conformal space unifies a lot of mathematical systems that are used in the graphics community like quaternions to represent rotations and Lie algebra for the description of rigid body motions. It is hoped that

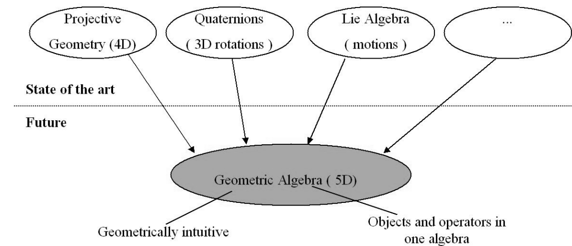


Figure 1: *Geometric Algebra in Computer Graphics*

© The Eurographics Association 2004.

our presentation of Geometric Algebra will stimulate a lot of new insights in all areas of Computer Graphics.

3. History of Geometric Algebra

William K. Clifford (1845-1879) introduced what we now call Geometric or Clifford Algebra, in a paper entitled "On the classification of geometric algebras," [Cli82]. He realized (as Grassmann did) that Grassmann's exterior algebra and Hamilton's quaternions can be brought into the same algebra by a slight change of the exterior product. With this new product, which we will call the geometric product, the multiplication rules of the quaternions follow directly from combinations of basis vectors (more details later), while Grassmann's exterior algebra is not lost. Furthermore, complex numbers and the Pauli matrices, as used in Quantum mechanics, have also a natural representation in Clifford algebra. However, due to the early death of Clifford, the vector analysis of Gibbs and Heaviside dominated most of the 20th century, and not the Geometric Algebra.

Geometric Algebra has found its way into many areas of science, since David Hestenes treated the subject in the '60s. 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 [HS84, HZ91]. Many other researchers followed and showed that applying Geometric Algebra in their field of research can be advantageous, e.g. [LFLD98, PL01, Dor01, LHR01].

The first time Geometric Algebra was introduced to a wider Computer Graphics audience, was probably at the SIGGRAPH conferences 2000 and 2001. These tutorials were very well received by the community. Since then, many people in the graphics community became interested in using Geometric Algebra. This convinced us that a tutorial in which we present the fundamental ideas behind Geometric Algebra, showing new applications and giving ideas for future research, based on our experiences with the subject matter, would be very useful for the community.

4. Properties of Geometric Algebra

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

4.1. Geometric Intuitivity

A very nice feature of Geometric Algebra is its geometric intuitivity. For example, 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 help of their outer product. Another nice example are rigid body motions. They are described with help of an operator including the relevant geometric parameters like the rotation axis, the angle of rotation and the displacement.

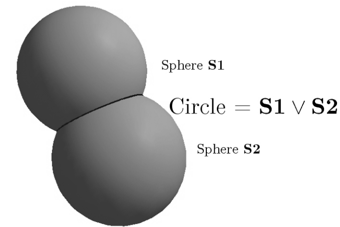


Figure 2: *Circle as intersection of two spheres*

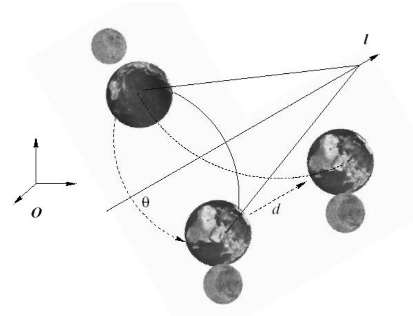


Figure 3: *Screw motion along l*

4.2. Unification

As mentioned before, in Geometric Algebra **quaternions** can be represented. Furthermore, this algebra includes also a lot of other mathematical systems like vector algebra, Grassmann algebra, complex numbers and quaternions.

4.2.1. Performance and elegance in a ray tracing application

At the university of Amsterdam a ray tracer was developed in order to compare different geometric approaches to this standard graphics application. For details see section 5 or refer to [FD02].

4.3. Low Symbolic Complexity

Expressions in Geometric Algebra normally have low complexity. As will be shown in this tutorial the inner product of two vectors $P \cdot S$ can be used for different tasks like

- the Euclidean distance between two points
- the distance between one point and one plane

© The Eurographics Association 2004.

Table 1: *List of the conformal geometric entities (IPNS).* † *This representation of a point is the same in the IPNS and OPNS.*

entity	representation	grade
Point†	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	1
Sphere	$s = P - \frac{1}{2}r^2 e_\infty$	1
Plane	$\pi = \mathbf{n} + d e_\infty$	1
Circle	$z = s_1 \wedge s_2$	2
Line	$l = \pi_1 \wedge \pi_1$	2
Point Pair	$q = s_1 \wedge s_2 \wedge s_3$	3
Point	$p = s_1 \wedge s_2 \wedge s_3 \wedge s_4$	4

- the decision whether a point is inside or outside of a sphere

5. Foundations of Conformal Geometric Algebra

5.1. Blades and Products

The three most often used products of Geometric Algebra are the **outer**, the **inner** and the **geometric** product. In table 3 the notation of these products is listed. **Blades** are the basic computational elements and the basic geometric entities of the Geometric Algebra. For example, the Geometric Algebra of the Euclidean 3D space consists of blades with **grades** 0, 1, 2 and 3, whereby a scalar is a **0-blade** (blade of grade 0), the **1-blades** are the three base vectors e_1, e_2, e_3, the **2-blades** are plane elements spanned by three base vectors and the **3-blade** represents the whole space. There exists only one element of grade three in the Geometric Algebra of 3D Euclidean space. It is therefore also called the pseudoscalar. A linear combination of k-blades is called a **k-vector** (also called vectors, bivectors, trivectors ...). Furthermore, a linear combination of blades of different grades is called a multivector. Multivectors are the general elements of a Geometric Algebra.

5.2. The Blades of the Conformal Geometric Algebra

Compared to the above mentioned 3D Euclidean Geometric Algebra, the 5D Conformal Geometric Algebra provides a greater variety of basic geometric entities to compute with.

Table 1 lists the conformal geometric entities with respect to the inner product null space **IPNS**. In this table $\mathbf{x}$ and $\mathbf{n}$ are marked bold since they represent 3D entities. The $\{s_i\}$ represent different spheres and the $\{\pi_i\}$ different planes. A sphere is represented with help of its center point P and its radius r. Note that the representation of a point as in the first row of this table in terms of the IPNS, is simply a sphere with radius zero. Similarly, a plane represented in the IPNS is a sphere with infinite radius. The representation of a point as in the last row of the table, is given by the intersection of four spheres, which can at most intersect in a single point. That is, not every outer product of four spheres represents a point.

Table 2: *List of the conformal geometric entities (OPNS)*

entity	representation	grade
Sphere	$S = x_1 \wedge x_2 \wedge x_3 \wedge x_4$	4
Plane	$\Pi = x_1 \wedge x_2 \wedge x_3 \wedge e_\infty$	4
Circle	$Z = x_1 \wedge x_2 \wedge x_3$	3
Line	$L = x_1 \wedge x_2 \wedge e_\infty$	3
Point Pair	$Q = x_1 \wedge x_2$	2
Point	$P = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	1

Table 3: *Notations*

notation	meaning	alternative
AB	geometric product of A and B	
$A \wedge B$	outer product of A and B	
$A \cdot B$	inner product of A and B	A.B
A^*	dual of A	dual(A)
A^{-1}	inverse of A	
$\tilde{V}$	reverse of V	
$A \vee B$	meet of A and B	
e_0	conformal origin	e0
e_∞	conformal infinity	einf
$< A >_r$	grade selection of a multivector	
$A_{<r>}$	blade of grade r	

Table 2 lists the conformal geometric entities with respect to the outer product null space **OPNS**. A sphere is represented with help of 4 points that lie on it. The IPNS and OPNS representations are **dual** to each other. It depends on the application which representation is more convenient to use.

Remark: At times we will refer to the OPNS representation as the 'standard' representation, and to the IPNS representation as the 'dual' representation.

© The Eurographics Association 2004.

PART TWO

The Blades of the Conformal Model

Daniel Fontijne & Leo Dorst

This tutorial introduces the blades of the conformal model of 3D Euclidean geometry, to date the most powerful way of using Geometric Algebra for Euclidean computations. We use GAViewer, a package for computation and visualization of the elements of this model, to establish the correspondence between geometric intuition and algebraic specification in this model.

This tutorial follows the presentation given at Eurographics 2004 and is a selection of our more detailed tutorial *3D Euclidean Geometry through Conformal Geometric Algebra (a GAViewer tutorial)*, which can be found on our site `http://www.science.uva.nl/ga` .

For a full understanding of the material, some basic knowledge of Geometric Algebra is assumed; it can be attained by reading part 3 of this tutorial, our more basic GABLE+ tutorial, or reading tutorial papers such as [DM02] and [MD02]. But for now, let's just have a look at what's possible with Conformal Geometric Algebra.

6. Warming up to CGA

CGA (Conformal Geometric Algebra) is a very convenient model to do Euclidean geometry. It is built upon a representation of points and (dual) spheres. Internally, these are represented as vectors, but you do not need to know that to work with them. Also, no operations depend upon an origin, or need to be specified in terms of coordinates relative to that origin. In this sense, CGA is coordinate-free.

In this chapter, we get a feeling for its ease and possibilities, by playing around with the basic concepts. In chapter 7, we go into why this is possible and how it works – for that, you will need to know some Geometric Algebra (or Clifford algebra) – but for now you can get away with pattern matching.

You should first download GAViewer from `http://www.science.uva.nl/ga/viewer` . Unpack it and start the executable. Then download `eg04_cga_blades.tar.gz` which should be available from the same page.

Since GAViewer supports several models, we need to set ourselves up in the 3D Conformal Geometric Algebra. This is most simply done by loading in the entire directory `eg04_cga_blades` for this tutorial. That will load the necessary functions, and perform initialization. So, under the File menu, select `File→Load .g directory` and point it to where you have unpacked the software.

modifier	LeftMouse	MiddleMouse	RightMouse
-none-	Rotate	Translate	Zoom
Ctrl	Select	Select	Translate

Table 4: Mouse actions.

6.1. Points

First, we have a way to specify a (typical) point in our display. It is represented by the vector 'e_0'

```
  a = e0
a = 1.00*e0
```

This creates a point. By Ctrl-RightMouse-Drag you can move it a bit (i.e. press the Ctrl key and the right mouse button at the same time while you are over the point, this selects it; then drag the mouse to move it - see also table 4). You can ask for the representation of this shifted point by typing '`a`', which will be something like:

```
  a
a = 1.00*e1 + 0.50*e2 + 0.00*e3 +
    1.00*e0 + 0.625*einf
```

So the point `a` has changed, and you see that in general there are 5 coefficients to the specification of a point. The coefficient of e_0 is an overall weight, the coefficients of $e1$, $e2$, $e3$ are its (weighted) position relative to e_0, and we will explain in section 7.1 that the coefficient of e_∞ is proportional to half the modulus squared of the displacement vector. The $\{e1, e2, e3\}$ part is therefore how you would represent a point by its location vector in the regular representation of Euclidean space, the 'e_0' part extends that to the representation of a point in the homogeneous model, and the 'e_∞' part makes this into the new 'conformal' representation. This chapter will let you play around with such points to show that the conformal part really helps to make a very nice and compact description of Euclidean geometry.

Let us create some more points `b`, `c`, `d`:

```
  b = c = d = e0
```

and you can drag them to any place by Ctrl-RightMouse-Drag (repeatedly do Ctrl-RightMouse to cycle through different objects at the cursor location and note that the information bar at the bottom shows which one you have selected). You should tilt the viewing plane with LeftMouse-Drag to move the points to arbitrary 3D positions (if you don't, they will be in the same plane). We don't really care about their coordinates, in GA we never need them to describe the actual geometry, in the sense of the relationships and operations of objects.

Now you make the sphere 'spanned' by these four points simply by typing:

```
  sphere = a^b^c^d
```

© The Eurographics Association 2004.

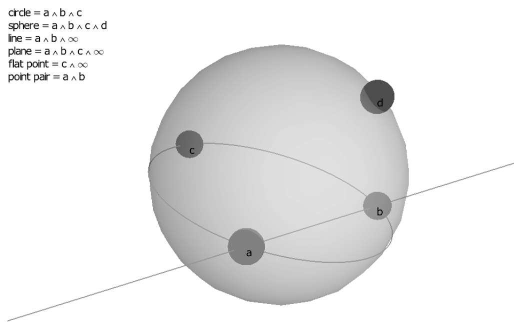


Figure 4: *Spanning rounds and flats in the conformal model.*

(Ignore the printed sphere coordinates, for now.) So this '$\wedge$' can span things. It is called the *outer product*. Using it on 3 points produces a circle:

```
circle = a^b^c
```

and doing it on 2 points makes a point pair:

```
ppair = a^b
```

(A point pair is blue, whereas individual points are red) But it would really be nice to have all these objects change as we drag the points around, to convince ourselves that it always works. Let's do that by making the definitions 'dynamic':

```
dynamic{ sphere = a^b^c^d ,},
dynamic{ circle = a^b^c ,},
dynamic{ ppair  = a^b ,},
```

(Note the curly brackets, and the extra , just before the final } ! If you make a mistake, you have to remove the dynamic statements using `Dynamic→View Dynamic Statements` and remove it by ticking the box.) (You can use the 'up arrow' to go back to earlier statements, edit them, and re-enter them by pressing Enter.)

All these objects have a sense of orientation, and you can show this by selecting them, and then check 'draw orientation' in the Controls panel, which you can pop up by selecting it in the `View→Controls` menu. In this way, you should be able to see that the circle $a \wedge b \wedge c$ has the opposite orientation of $a \wedge c \wedge b$, but the same as $c \wedge a \wedge b$. Just drag the points around along the circle (make sure you look at it from straight above) and see the orientation change as one point passes another. (If you find it hard to see the 'barbs' denoting the orientation, de-select 'draw weight', and they'll be drawn at a standard length.)

Algebraically, this means that the outer product $\wedge$ is antisymmetric in any two of its arguments. It is also associative (so that defining it for 2 arguments extends to any number of arguments), so we do not need to write $(a \wedge b) \wedge c$ or $a \wedge (b \wedge c)$ – these are both equal to $a \wedge b \wedge c$.

Switch on the orientation display of the sphere, and note that it also changes orientation as you swap points. For the sphere, this happens when one of the points moves through the plane determined by the three others.

When you move the points around, you find that in some configurations, the circle almost becomes a straight line, and the sphere a plane, but that it is very hard to get that exact. There is, of course, also an explicit representation for these entities. Such 'flat' spaces go through the point at infinity, and in the conformal model, that is a regular point of the algebra. We have denoted it as '$\mathtt{e}_\infty$'.

So to form a line and a plane, do:

```
dynamic{ line = a^b^einf,},
dynamic{ plane = a^b^c^einf,},
```

As you see, the circle $C = a \wedge b \wedge c$ lies in the plane $C \wedge \mathtt{e}_\infty = a \wedge b \wedge c \wedge \mathtt{e}_\infty$, so we are beginning to glimpse a geometrically significant algebra. We call the elements in this algebra *blades*: a circle is a 3-blade, a point a 1-blade, et cetera. The number of points used to make it is called the *grade* of the blade.

6.2. Complementation: the dual

Another important operation in the construction of elementary Euclidean objects is *dualization*. It is a 'complementary' representation of an object. It can be used to describe an object not so much specifying the points that are on it, but the points that are orthogonal to its complement.

If x is a point on a circle C (for instance constructed from other points as $C = a \wedge b \wedge c$), then we have $x \wedge C = 0$ since yet another point on the circle does not help to span a sphere or plane.

For the dual representation of C, i.e. $D \equiv \mathbf{dual}(C)$, the points on C are characterized as $x \cdot D = 0$, with $\cdot$ the geometrical inner product.

Of course we still think about a dual object in very much the same way as about the original object. So in GAViewer we plot it the same, but in a different color.

```
sphere = a^b^c^d,
dsphere = dual(sphere),
```

We draw direct spheres (or planes) yellow, and dual spheres (or planes) red.

Dualization makes some operations much more easy to specify, and is a powerful tool – we'll use it immediately in the next section.

6.3. Intersection

Apart from spanning using the $\wedge$, making object of higher dimensions, we can also intersect objects. The intersection of A and B is generally given by the operation

$$A \cap B = \mathbf{dual}(B) \cdot A,$$

© The Eurographics Association 2004.

at least if they are 'in general position'. You can think of this as removing (through the inner product) 'everything that is not *B*' from *A*.

But it is easier to remain in a dual representation, for

$$\mathbf{dual}(A \cap B) = \mathbf{dual}(B) \wedge \mathbf{dual}(A),$$

so basically intersection is an outer product in a dual representation. $\mathbf{dual}(B) \wedge \mathbf{dual}(A)$ can intuitively be understood as computing the union of 'everything that is not *B*' and 'everything that is not *A*'. Then the dual of that must be what *B* and *A* have in common. We will get back to understanding this later in section 10, for now let's just play with it.

The above suggests that we first construct the objects we want using the outer product $\wedge$; then dualize them all using $\mathbf{dual}()$; then use $\wedge$ intersect them in this dual representation. We may never choose to go back to the direct representation after this.

So make dual representations of our objects that we still have in the view:

```
dynamic{ dsphere = dual(sphere) ;},
dynamic{ dcircle = dual(circle) ;},
dynamic{ dplane  = dual(plane) ;},
dynamic{ dline   = dual(line) ;},
```

The ; before the final curly bracket is an instruction not to draw the object constructed – if you want to see them, change it to a comma (but what you see for the circle may surprise you).

Now let's form another dual sphere and show the intersections with the objects defined. Ignore how this dual sphere is made for now, we just mention that it is at e_0 and has unit radius.

```
dA = e0-einf/2,
```

Now for the intersections. Before we make them, we simplify the situation a bit by putting the points in more standard positions relative to e_0. (To prevent mistakes, you could also run the whole demonstration by typing

```
DEMOintersect();
```

which also gives you some handy labels for the quantities to aid in dragging them - just do Ctrl-RightMouse-Drag on the label. It builds things up gradually, just keep pressing Enter while you see the `DEMOintersect` prompt.) To draw things properly, we undualize them, though for continued computations this is not really necessary.

```
a=pt(0), b=pt(e1), c=pt(e2), d=pt(e3),
dynamic{ dAsphere = -dual( dA^dsphere ) ,}
dynamic{ dAcircle = -dual( dA^dcircle ) ,}
dynamic{ dAplane  = -dual( dA^dplane ) ,}
```

The function $\mathbf{pt}()$ creates a point that is the point e_0 shifted over the vector specified. As you move the dual sphere `dA` around (Ctrl-RightMouse-Drag), you see the intersections change. Note that some are peculiar: apparently, two spheres always intersect in a circle, though this circle is not always on the spheres! Such strange intersecting circles are in fact imaginary (in the sense that their radius has negative square), and we have drawn them dashed to show that they are unusual.

Now move `dA` around again, in different planes (by tilting the view using LeftMouse-Drag), and note how intersections change. They always exist, for the system is 'closed' due to the inclusion of e_∞. If you're precise and capable of placing some elements such that they touch or are parallel, you'll see some strange objects which we'll discuss in at the end of chapter 7.

If you'd like to play some more, `DEMOincidence();` is similar.

7. Elementary objects

We can now unveil how CGA works, by going through some of the details of the representation. To keep the explanation down to earth, we will occasionally refer to a coordinate representation. Although coordinates are not required to specify the operations of Geometric Algebra, they are of course still useful to specify its objects. We also give all of the kinds of objects that appear when we combine the span and dual operations (i.e. all intersections of spanned objects). After this section, you will be able to define lines, planes, etcetera simply, and with the precise locational and directional properties you desire.

7.1. Rounds and flats

You can best start this section with a clean display. Type

```
clearall();
```

This removes all objects and dynamics and clears the console.

We start with a point. The prototypical point is 'e_0', the point at the (arbitrary) origin of our 3D space. Let us again pay attention to the representation of points. Make `a = e0` and move it around with Ctrl-RightMouse-Drag to a new location. Then ask for its new representation, which will be something like:

```
  a
a = 1.00*e1 + 0.50*e2 + 0.00*e3 +
    1.00*e0 + 0.625*einf
```

The general expression of a point at a location relative to e_0 given by position vector the $\mathbf{p}$ (specified on the Euclidean basis $\{\mathrm{e}1, \mathrm{e}2, \mathrm{e}3\}$ is:

$$p = \alpha\,\mathbf{pt}(\mathbf{p}) \equiv \alpha\,(\mathrm{e}_0 + \mathbf{p} + \tfrac{1}{2}(\mathbf{p}\cdot\mathbf{p})\,\mathrm{e}_\infty))$$

(where $\mathbf{p}\cdot\mathbf{p}$ is of course the squared norm of $\mathbf{p}$, a scalar, and α is a scalar). Thus the function $\mathbf{pt}()$ maps a Euclidean vector to the CGA representation of a point at that relative location. For instance:

© The Eurographics Association 2004.

```
p = pt(e1 + 2 e2)
```

These points are the basic elements of CGA. The inner product of CGA has been defined so that the basis elements of $\{e1, e2, e3, e_0, e_\infty\}$ have the following inner products:

$$\begin{aligned} e1 \cdot e1 = e2 \cdot e2 = e3 \cdot e3 &= 1 \\ e_0 \cdot e_\infty &= -1 \end{aligned}$$

All other inner products between basis vectors are 0. Note the special nature of e_0 and e_∞: they are *null vectors* (i.e. their norm is zero), but in a sense each other's negative inverse under the inner product (since $e_0 \cdot e_\infty = -1$). We sometimes call e_0 and e_∞ *reciprocal null vectors*.

Why it has been set up in this way you find out when you compute the inner product between two normalized point representatives (for which the weight α equals 1):

$$\begin{aligned} \mathbf{pt}(\mathbf{p}) \cdot \mathbf{pt}(\mathbf{q}) &= (e_0 + \mathbf{p} + \tfrac{1}{2}\mathbf{p}^2 e_\infty) \cdot (e_0 + \mathbf{q} + \tfrac{1}{2}\mathbf{q}^2 e_\infty) \\ &= -\tfrac{1}{2}(\mathbf{p} - \mathbf{q}) \cdot (\mathbf{p} - \mathbf{q}) \\ &\equiv -\tfrac{1}{2}\, d_E^2(\mathbf{pt}(\mathbf{p}), \mathbf{pt}(\mathbf{q})) \end{aligned}$$

The inner product of two normalized points gives the square of the Euclidean distance! Since normalization is achieved simply through division: $\mathbf{pt}(\mathbf{p}) \rightarrow \mathbf{pt}(\mathbf{p})/(-e_\infty \cdot \mathbf{pt}(\mathbf{p}))$, we have for the vector representatives p and q of two points $\mathcal{P}$ and $\mathcal{Q}$:

$$\frac{p}{-e_\infty \cdot p} \cdot \frac{q}{-e_\infty \cdot q} = -\tfrac{1}{2}\, d_E^2(\mathcal{P}, \mathcal{Q})$$

The Euclidean distance measure is therefore deeply embedded into the algebra, and this means that all algebraic constructions incorporate it. (In the more classical approaches, we have to impose it explicitly by more cumbersome constructions.) Note that point representatives are null vectors:

$$\mathbf{pt}(\mathbf{x}) \cdot \mathbf{pt}(\mathbf{x}) = 0, \quad \text{for any } \mathbf{x}$$

A sphere with center c and radius ρ can by constructed by analyzing the demand:

$$x \cdot c = -\tfrac{1}{2}\rho^2$$

We want to rewrite this to something involving x explicitly. We do this using $e_\infty \cdot x = -1$, true for a normalized point x. This gives:

$$x \cdot (c - \tfrac{1}{2}\rho^2 e_\infty) = 0,$$

so that

$$s = c - \tfrac{1}{2}\rho^2 e_\infty$$

is the dual representation of a sphere with center c and radius ρ^2. Since s is a vector in CGA, we see that general vectors 'are' (weighted) dual spheres.

We will often make dual spheres at e_0, which are simply

```
e0 - einf * r * r / 2,
```

So you often see us use the dual unit sphere at the origin: $e_0 - e_\infty/2$.

As you see, you can also make a sphere with a radius whose square is negative, for instance:

```
e0 + einf
```

We will call those 'imaginary spheres', and automatically stipple them.

The squared radius of a sphere can be found as the square of its dual (using the geometric product or the inner product):

$$s^2 = (c - \tfrac{1}{2}\rho^2 e_\infty)^2 = \rho^2$$

(if you want to type this in, do something like `ds = pt(e1) - einf/20; rhosquared = ds ds,`). In this view, the points $\mathbf{pt}(\mathbf{x})$ are dual spheres with radius zero, since $\mathbf{pt}(\mathbf{p})^2 = 0$ for any $\mathbf{p}$. This will give us a nicely consistent semantics in section 10.

To get the actual sphere corresponding to `e0-einf/2`, just *un*dualize it:

```
-dual(e0-einf/2)
```

The difference in display is that a dual sphere is red (since it is an object of grade 1), whereas a direct sphere is yellow (being of grade 4).

A plane is also a grade 4 object, and in fact merely a sphere that also contains the point e_∞. Dually, it looks like

$$\mathbf{n} + d\, e_\infty$$

where $\mathbf{n}$ is the unit normal vector denoting its attitude, and d the distance to the origin. You may verify that this indeed represents the plane, by showing that:

$$\mathbf{pt}(\mathbf{x}) \cdot (\mathbf{n} + d\, e_\infty) = 0 \iff \mathbf{x} \cdot \mathbf{n} = d,$$

which is the usual 'Hesse normal equation' of a plane.

A line can be made in various ways. You can do

```
a^b^einf,
```

showing that a line is determined by two points plus the point at infinity. You can also intersect two planes, for instance the planes dually represented by $e1$ and $(e2+e_\infty)$:

```
dual( e1 ^(e2+einf) )
```

Or you can specify it by a point and a direction. For a line through the point b, this is:

```
b^e1^einf
```

for a line in the e1-direction. Note that we need to put the special point e_∞ in as well (any line passes through infinity).

Since a line is a grade 3 object, a dual line is of grade 2. We have incorporated some tests that recognize these particular grade 2 objects and draw them as the line they represent – but in blue, as befits a grade 2 object.

© The Eurographics Association 2004.

The combination of dual spheres and dual planes allows specification of dual circles rather easily, since the wedge is their dual intersection. So to specify a circle with radius 1 around the origin in the $e2 \wedge e3$ plane, simply type:

```
dual( (e0-einf/2)^e1 )
```

Note what happens when you leave out the dual:

```
(e0-einf/2)^e1
```

This is an imaginary point pair, 'orthogonal' to the circle. We cannot automatically interpret this for you and draw it as a circle, since point pairs (even imaginary point pairs) are also legitimate objects. In fact, they are 1-dimensional spheres, the set of points of a line which have equal squared distance to a given point also on that line (the latter being the 'center' of the 1-dimensional sphere).

We will call planes and lines *flats*, and spheres, circles and point pairs *rounds*. A flat is a round containing the point at infinity 'e_∞'. As we will see later, this means that it does not have a *size* in the way that spheres and circles do. Both flats and rounds have a *weight* (or if you prefer, a *density*), which you can see in the Controls panel by selecting the object (Ctrl-RightMouse).

7.2. Tangent blades

With all these objects, you might think we are complete: what more can there be when you intersect round and flat things except other round and flat things? However, there are several surprises. See what happens if you intersect a sphere with one of its tangent planes:

```
-dual(  (e0-einf/2)^(e1+einf) )
```

Your display shows a disk, which the information bar in the Controls panel tells you is a 'tangent bivector'. It is what the sphere and the plane have in common at their point of intersection, which is slightly more than merely the point. It is grade 3, and you can think of it as an infinitesimal circle in a well-defined plane. We knew of no better way to draw it than this disk.

To make such a tangent object at the origin, type:

```
e0^e1^e2
```

but beware: a tangent bivector at a point c is *not* made using the construction $c \wedge e1 \wedge e2$. Of course there are also tangent *vectors*, and you can make one at e_0 by

```
e0^e3
```

7.3. Free blades (attitudes)

And still, this does not exhaust the possibilities of basic object classes. Let us intersect two parallel planes:

```
-dual( e1^(e1+einf)),
```

and you find that the answer is

```
-e2^e3^einf
```

which is a form we have not seen before. It is a 2-dimensional direction element, which we draw stippled at the origin. Try to drag it away: you can't. This is a translation invariant element of CGA, and eminently suitable to be called an 'attitude': it has no position, only an attitude (or orientation, if you will).

We call this a *free bivector*, because it has no position. We could have drawn it anywhere, or as a 'bivector field'. We decided to draw it dashed at the origin and make it immovable, but this does *not* mean that it resides there – in fact, it resides nowhere at all, it merely has an attitude. It certainly does not reside at the origin, which is arbitrary anyway – but we had to do something. Fortunately, you will find that these elements hardly occur usefully by themselves, but mostly as elements in the construction of more easily interpretable objects.

Similarly, the *free vector*

```
e1^einf
```

is drawn dashed at the origin; mind that it is different from $e_0 \wedge e1$ (which is drawn solid). It is a one-dimensional attitude, i.e. a direction vector. We now see that a line is in fact made as the outer product of a point and an attiude:

```
a^(e1^einf)
```

which corresponds to the idea of a location/direction pair, algebraically composed. You can drop the brackets since the outer product is associative, and change the order (giving a minus sign each time you swap two vectors) – that retrieves the representation we have seen before.

7.4. That's all

The above really does exhaust the objects that can be made by repeated application of outer product and dualization applied to vectors and hence as the 'closure' of the spans of points and their intersection. As you see, we have the classical repertoire of elements you use in linear algebra, and then some more: spheres and tangents, free elements. All these are precisely related algebraically. None of these is what we call a 'vector' classically – that has in fact become an imprecise usage, since a 'normal vector' a 'direction vector' and a 'position vector' are all different (they react differently when the origin is moved, or when the space is transformed). We should reserve the term 'vector' for an element of the modeling algebra, rather than for the elements of geometry.

CGA enables precise definitions for each vector-based concept:

- 'normal vector' **n** (best seen as a dual plane **n**, which is then automatically extended by translation to encode for its location, see below)
- 'direction vector' **v** (best seen as the attitude $\mathbf{v} \wedge e_\infty$)

© The Eurographics Association 2004.

- 'tangent vector' $\mathbf{t}$ (which is $e_0 \wedge \mathbf{t}$, translated to the desired location)
- 'position vector' $\mathbf{p}$ (this corresponds to $e_0 \wedge \mathbf{p} \wedge e_\infty$, a line element from e_0 to $\mathbf{pt}(\mathbf{p})$, although this is freely shiftable along the line; it may be better to see a position vector as a direction vector to be used from e_0)

Each of these have well-defined properties and all are properly related within the unified framework.

8. A visual explanation

We can also show you more visually why this surprising characterization of rounds by blades works. In this chapter, we 'pop up' the e_∞-dimension graphically, by using the 3D CGA as a specification language for OpenGL commands, but we will necessarily show you only the CGA for a 2-dimensional Euclidean geometry. For us, this depiction helped to take quite a bit of the magic out (though not affecting the poetry of the procedure).

We have seen that a point at $\mathbf{x}$ is represented as:

$$\mathbf{pt}(\mathbf{x}) = e_0 + \mathbf{x} + \tfrac{1}{2}\mathbf{x}^2\, e_\infty$$

In 2D, this requires a 4D space with a basis like $\{e_0, e1, e2, e_\infty\}$. This would seem hard to visualize. However, the e_0-dimension works very much like the extra dimension in homogeneous coordinates: it allows you to talk about 'offset linear subspaces', linear subspaces that are shifted out of the origin (you can run `DEMOhomogeneous();` to remind yourself of this). So because of the e_0-term, we are allowed to draw planes, lines, et cetera that do not need to go through the origin. If you accept that, we do not need to draw this dimension explicitly, we can just use this freedom and know that such things are blades because of the e_0-dimension.

The e_∞-dimension is new, and much more interesting. If we draw the Euclidean 2-space as the $e1 \wedge e2$-plane, then there is apparently a *paraboloid* $\frac{1}{2}\mathbf{x}^2$ in the e_∞-direction that we should get to know better.

Just execute the command

```
DEMOc2ga();
```

By hitting return, it will execute various stages of our visualization. For now, stop at the step where it says: `DEMOc2ga initialized »` . (If you hit too far, just keep doing it till your prompt returns to the regular prompt, then restart `DEMOc2ga()`.) You see the 2-dimensional Euclidean space laid out in white, and the e_∞-paraboloid indicated vertically above it. The sliders in the bottom right of your window allow you to play with `pan` and `tilt` for better views.

Now we can play around. Let us first interpret a point x – actually, we use flat points like $x \wedge e_\infty$. Hit return till you get to: `DEMOc2ga visualization of x »` . As you move the red vector x around (by dragging its point), you see a yellowish plane move with it. This plane is the $\mathbf{dual}(x)$ (in the metric of 2D CGA): it consists of all the vectors perpendicular to the vector x (in this metric). (It doesn't *look* perpendicular, but that is because we are watching with Euclidean eyes.)

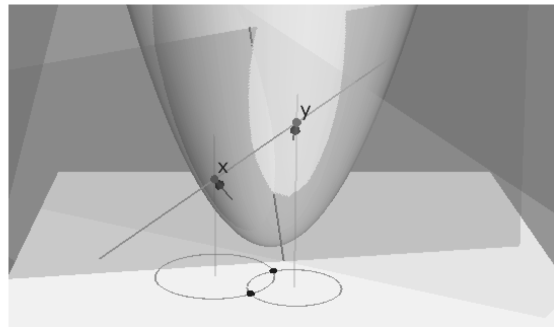


Figure 5: Visualization of the intersection of circles in the conformal model.

If $\mathbf{x}$ is on the paraboloid, this plane is the tangent to the paraboloid at that point. (Confirm this for yourself, if necessary by changing your viewpoint.) How would we express this? Well, in homogeneous coordinates x is on a plane P iff $x \wedge P = 0$. Or, if we have a dual representation of the plane, $p = \mathbf{dual}(P)$, then x is in the plane iff $x \cdot p = 0$. You will remember that the metric of CGA is set up in such a way that $x \cdot x = 0$, and the motivation for that was that a point represented by x has distance 0 to itself in the Euclidean metric. We now see that we can also read this as: if the vector x represents a Euclidean point, then x is on the plane dually represented by x in CGA. This is all consistent, for the parabola is given by the CGA metric, which in turn was designed to make the inner product of points be related to the (squared) Euclidean distance.

In the 2D CGA metric, the duality of a point to a plane works in a matter that you may discover by moving the point around: project the point x onto the parabola by a (vague red) line perpendicular to the Euclidean 2-space. The dual plane for x will be parallel to the tangent plane at this intersection point, but as far above the paraboloid as x is below it (or vice versa).

You see that the plane intersects the paraboloid in an ellipse (if you are at the proper side of it), and that we have drawn a circle in the 2D Euclidean plane as its projection. Apparently, there is a direct correspondence between the dual of a vector (the plane) and a Euclidean circle. But we know that there is. Let c be the representation of a Euclidean point – so c is on the paraboloid. Now subtract $\frac{1}{2}\rho^2 e_\infty$ from c, which gives the vector

$$s = c - \tfrac{1}{2}\rho^2\, e_\infty$$

This is the dual representation of a sphere (and in 2D, a sphere is a circle). But it is also a general vector of the form

© The Eurographics Association 2004.

we have just moved around. If we enquire which actual Euclidean points are on this set, we have to enquire for which vectors x, which satisfy $\mathrm{x} \cdot \mathrm{x} = 0$, the equation $\mathrm{x} \cdot \mathrm{s} = 0$ holds. The former demand is: x lies on the paraboloid, and the latter: x lies on the plane dual to s. Together you can work it out as:

$$0 = \mathrm{x} \cdot (\mathrm{c} - \tfrac{1}{2}\rho^2\, \mathrm{e}_\infty) = -\tfrac{1}{2} d_E^2(\mathrm{x}, \mathrm{c}) + \tfrac{1}{2}\rho^2$$

(using $\mathrm{x} \cdot \mathrm{e}_\infty = -1$, true for normalized points). So indeed these are the points x that have squared distance ρ^2 from c.

As you move the point s 'inside' the parabola, the dual plane lies outside it, and it seems there is no intersection. Actually, the intersection is imaginary, leading to a sphere with negative radius squared, but we hope that the interactive depiction provides the confidence that this is all completely regular.

There is a small artifact of our depiction: if you would have x precisely *on* the paraboloid, the circle should degenerate to a 2D CGA point. But in our depiction (which fakes this using 3D CGA geometry), it actually becomes a 3D CGA tangent bivector. Try it by defining `x = pt(e1+e3/2)`.

Now hit return again in the demo, to get to the prompt "`DEMOc2ga visualization of x and y »` ". The construction shows how the intersection of two spheres (circles in 2D) is done in CGA, and we explain it as follows.

A point pair is a 1D sphere. We know that the 2D CGA model would represent this as the outer product of two vectors $\mathrm{x} \wedge \mathrm{y}$, i.e. as a line in this homogeneous depiction. For two vectors representing points, this is easy enough: the vectors lie on the parabola, and so the intersection of the line with the parabola precisely retrieves the points. If the vectors x and y are off the paraboloid, they represent dual circles. Then their outer product dually represents the intersection of those circles. Undualizing should then provide the direct representation of this intersection, i.e. a point pair.

In the terminology of our visualization: make the planes corresponding to the vectors x and y, and intersect them to form a line ℓ. That is the representation of the intersection of the circles. To find the point pair, ask which points lie in the set ℓ; you do that by intersecting ℓ with the paraboloid. If it really intersects, the point pair is real, and otherwise imaginary (in a location that is rather counterintuitive but can be explained with some effort – look for hyperbolas, if you must...).

Move x and y around from the initial situation to get a feeling for how it is all connected. And you may enjoy typing the following to try a situation with two tangent circles (zoom, pan, tilt if necessary):

```
x = pt(-1.5 e2 + e3),
y = pt(-2 e2 + 1.5 e3),
```

Here is the take-home message of all this visualization:

> *In 2D CGA, 'intersecting circles' is identical to 'intersecting homogeneous planes' in 1 more dimension, which in yet 1 more dimension is identical to 'intersecting subspaces through the origin' – which is easy to do. So intersecting circles is easy – and so is intersecting general rounds in nD.*

Since spheres are important to Euclidean geometry (planes and lines et cetera are merely affine, not Euclidean), this is a relevant trick. We hope you now understand slightly better where those two extra dimensions come from.

9. Practicalities

This section treats some practicalities that might be among the first problems that a computer graphics practitioner todo would have to tackle to write a real application using conformal Geometric Algebra.

9.1. Parameters

If we want to draw the blades from the conformal model, we need to break them down into parameters such as location and attitude that we can send to OpenGL. Having these parameters can also be handy for other parameters, such as breaking a point pair up into to separate points.

In principle, computing parameters of the various kinds of blades is not too hard. For instance, the square of a normalized dual sphere gives you its radius squared, for

$$(\mathrm{e}_0 - \tfrac{1}{2}\mathrm{e}_\infty\,\rho^2)^2 = -\tfrac{1}{2}(\mathrm{e}_0\,\mathrm{e}_\infty + \mathrm{e}_\infty\,\mathrm{e}_0)\,\rho^2 = \rho^2$$

For the direct representation of a round there may be extra signs (depending on its grade), and for a general formula you need to normalize first. Tangents have size 0, and for the flats and attitudes, size is not an issue, they don't really have any.

Instead, attitudes and flats only have a *weight*, an overall multiplicative factor relative to unity. The weights of $2\mathrm{e}1$, of $2\mathrm{e}1 \wedge \mathrm{e}_\infty$, of $2\mathrm{e}_0 \wedge \mathrm{e}1 \wedge \mathrm{e}_\infty$ are all 2, but so is the weight of the tangent $2\mathrm{e}_0 \wedge \mathrm{e}1$ and the dual round $2\mathrm{e}_0 - \mathrm{e}_\infty$. So tangents and rounds have weights too. In some cases, these weights have a traditional way of displaying: a vector of weight 2 can be depicted as having length 2, and a tangent bivector of weight 2 as an area element of 2 areal units. But we have not decided how to depict a sphere of weight 2, and if we would draw points (i.e. dual spheres of zero radius) as different sizes, they would easily become confused with spheres. So for some objects, you will just have to monitor the weight, for instance using the 'Controls' panel.

The set of functions defined in `conformal_blade_parameters.g` defines the various blade parameters as the functions. The actual computations are indicated in table 5.

The *location* of a blade could be the Euclidean coordinates of some relevant point. For a round, this is naturally the

© The Eurographics Association 2004.

class	attitude	flat	dual flat	tangent	round
form	$\infty\mathbf{E}$	$p\wedge(\infty\mathbf{E}) = \mathsf{T}_{\mathbf{p}}(o\wedge(\infty\mathbf{E}))$	$p\cdot(\infty\mathbf{E}) = \mathsf{T}_{\mathbf{p}}(\mathbf{E})$	$p\wedge(p\cdot(\infty\mathbf{E})) = \mathsf{T}_{\mathbf{p}}(o\mathbf{E})$	$v\wedge(v\cdot(\infty\mathbf{E})) = \mathsf{T}_{\mathbf{p}}((o+\alpha\infty)\mathbf{E})$
condition	$\infty\wedge X = 0$ $\infty\cdot X = 0$	$\infty\wedge X = 0$ $\infty\cdot X \neq 0$	$\infty\wedge X \neq 0$ $\infty\cdot X = 0$	$\infty\wedge X \neq 0$ $\infty\cdot X \neq 0$ $X^2 = 0$	$\infty\wedge X \neq 0$ $\infty\cdot X \neq 0$ $X^2 \neq 0$
attitude	X	$\infty\cdot X$	$\infty\wedge X$	$\infty\wedge(\infty\cdot X)$	$\infty\wedge(\infty\cdot X)$
location	none	$(q\cdot X)/X$	$(q\wedge X)/X$	$\frac{X}{\infty\cdot X}$	$\frac{X}{\infty\cdot X}$ or $\frac{1}{2}\frac{X\infty X}{(\infty\cdot X)^2}$
sq. weight	$(q\cdot \mathrm{att}(X))^2$	$(q\cdot \mathrm{att}(X))^2$	$(q\cdot \mathrm{att}(X))^2$	$(q\cdot \mathrm{att}(X))^2$	$(q\cdot \mathrm{att}(X))^2$
sq. size	none	none	none	0	$\alpha = -\frac{X\widehat{X}}{2(\infty\cdot X)^2}$
inverse	none	$p\wedge(\infty\mathbf{E}^{-1})$	$p\cdot(\infty\mathbf{E}^{-1})$	none	$\frac{1}{2}\mathsf{T}_{\mathbf{p}}((o/\alpha+\infty)\widehat{\mathbf{E}}^{-1})$

Table 5: All non-zero blades in the conformal model of Euclidean geometry, and their parameters. For a round, the squared size α equals radius squared, for a dual round it is minus the radius squared. Locations are denoted by dual spheres. The points q are probes to give locations closest to q, one can just use $o = \mathrm{e}_0$. We denote $\mathrm{e}_\infty = \infty$, and $\widehat{X}$ is the grade inversion $(-1)^x X$ with $x = \mathrm{grade}(X)$, while $\widetilde{X}$ is the reversion $(-1)^{x(x-1)/2}X$. For more information see test [Dor03].

center, but for planes and lines such a point is not uniquely indicated in a coordinate free manner. We can either take the point closest to the origin (obviously not coordinate-free) or closest to some given point `q`. The formulas in the table actually produce a normalized dual sphere as the location; this is often enough, or you can take its Euclidean part as the Euclidean location vector (usable in a translation versor `tv()`), or compute the center by reflection e_∞ into the round X of grade k by:

$$c = -\frac{1}{2}\frac{X\,\mathrm{e}_\infty\,X}{(\mathrm{e}_\infty\cdot X)^2}$$

All these class-dependent functions have been collected in `conformal_blade_parameters.g`, but in a rather coded way for fast usage. The most useful are:

```
function attitude(X)
  // gives attitude of X
function location(X)
  // location of X
function sq_weight(X)
  // squared weight of X
function sq_size(X)
  // squared size, radius is +/- 2 sq_size
```

9.2. Example: Dissecting a Point Pair

At times, it may be necessary to split a point pair up into two separate points, for example, to draw it on the screen. We can use the parameter functions from above to achieve that. The following GAViewer code assume `pp` is a normalized point pair:

```
// get center & attitude of point pair 'pp':
  L = location(pp),
  A = attitude(pp),
// compute translation versor...
// ...in direction of A:
  T = exp(0.25 A);

// translate 'L' over 'T' one way:
  p1 = T * L * inverse(T),
// translate 'L' over 'T' the other way:
  p2 = inverse(T) * L * T
```

In general, `p1` and `p2` will be dual spheres. They can be turned into regular points adding their radius times $0.5\mathrm{e}_\infty$:

```
  p1 = p1 + 0.5 p1.p1 einf,
  p2 = p2 + 0.5 p2.p2 einf,
```

Translation versors such as `T` will be treated in part 4 of this tutorial.

A more careful analysis can reduce this method of dissecting a point pair to the following, more efficient equation: $\{p_1, p_2\} =$

$$\frac{pp\cdot(\mathrm{e}_\infty\wedge pp) - (pp\cdot pp)\,\mathrm{e}_\infty \pm \sqrt{pp\cdot pp}\,(\mathrm{e}_\infty\cdot pp)}{(\mathrm{e}_\infty\cdot pp)^2}$$

where the division by $(\mathrm{e}_\infty\cdot pp)^2$ is used only to normalize the points.

9.3. Angles between flats

A problem that occurs often in computer graphics is how to compute the angle between a pair of lines or a pair of planes.

© The Eurographics Association 2004.

These angles can be computed in the same way as with regular vectors. Say that the two normalized lines/planes are called x1 and x2, we compute cosine of the angle between them as

```
ca = x1 . x2
```

To see why this is true, first write:

$$\begin{aligned} x_1 &= p_{x_1} \wedge E_{x_1} \wedge e_\infty \\ x_2 &= p_{x_2} \wedge E_{x_2} \wedge e_\infty \end{aligned}$$

By this we mean what each x_i can be factored as the outer product of a point p_i, a purely Euclidean element E_i and the point at infinity. We can then proceed to derive:

$$\begin{aligned} x_1 \cdot x_2 &= (p_{x_1} \wedge E_{x_1} \wedge e_\infty) \cdot (p_{x_2} \wedge E_{x_2} \wedge e_\infty) \\ &= p_{x_1} \cdot (E_{x_1} \cdot (e_\infty \cdot (p_{x_2} \wedge E_{x_2} \wedge e_\infty))) \\ &= -p_{x_1} \cdot (E_{x_1} \cdot (E_{x_2} \wedge e_\infty)) \\ &= -p_{x_1} \cdot (E_{x_1} \cdot E_{x_2} e_\infty) \\ &= E_{x_1} \cdot E_{x_2} \end{aligned}$$

So $x_1 \cdot x_2$ just computes the inner product of the Euclidean part of both blades. If the blades are not normalized, then we should divide the result by $\sqrt{x_1 \cdot x_1}\sqrt{x_2 \cdot x_2}$; again, just as with regular vectors. Angles between other (non-flat) primitives can be computed with similar formulas, although computing the angle between primitives that are not of the same grade is more involved.

10. Construction by containment and orthogonality

We have constructed elements by spanning, or by intersection of spanned quantities. There are many geometrical problems in which this is enough, but CGA also allows direct specification of objects with different partial data. When we explore the rules involved, we seem to uncover a new and compact language for Euclidean geometry. In this section, we give you a feeling for this very new subject, attempting to develop algebraic rules and geometric intuition in tandem.

Let us see how we could specify a sphere of which we know the center c and one point p on it. Recall that the representation of a dual sphere with center c was: $s = c - \frac{1}{2}\rho^2 e_\infty$. If we know a point p on it, we must have $p \cdot c = -\frac{1}{2}\rho^2$. Rearranging terms (using the distributive law of inner product over outer product, as well as $p \cdot e_\infty = -1$) we find that the dual representation is:

$$c + (p \cdot c) e_\infty = p \cdot (c \wedge e_\infty)$$

This is immediately converted into GAViewer commands:

```
clearall(); // clear screen, dynamics
p = e0, c = e0, label(p);
dynamic{ s = p.(c^einf), }
```

Drag p and/or c to see the result. Notice that the sphere is drawn in red, since it is a dual sphere. Note also that $c \wedge e_\infty$ is a 'flat point' – we will get back to the geometrical intuition behind this equation below.

Key to the correspondence of geometrical intuition and algebraic expressions are two rules, involving 'being part of' and 'being perpendicular to'. Both can be given in direct form, and in dual form, and all four together provide our framework. We state them without proof. (In each of these expressions, the inner product is the default in our software: the contraction inner product.) We use the notation $\cdot^*$ to denote dualization, for easy reading of the formulas.

- *containment:* for vector **x** and blade **A** (with grade at least 1)

$$\mathbf{x} \in \mathbf{A} \iff \mathbf{x} \wedge \mathbf{A} = 0 = \mathbf{x} \cdot \mathbf{A}^*$$

- *perpendicularity* for blade **A** and blade **B** (with $\text{grade}(\mathbf{A}) \leq \text{grade}(\mathbf{B})$)

$$\mathbf{A} \perp \mathbf{B} \iff \mathbf{A} \cdot \mathbf{B} = 0 = \mathbf{A} \wedge \mathbf{B}^*$$

Let us play with that. Suppose we have three spheres A, B, C, and are looking for the GA object X that is perpendicular to each. We therefore need to satisfy $X \cdot A = 0, X \cdot B = 0, X \cdot C = 0$. Dualizing this, we get $X \wedge A^* = X \wedge B^* = X \wedge C^* = 0$. The simplest object satisfying this is:

$$X = A^* \wedge B^* \wedge C^*$$

Done. Let's show it.

```
clearall();
a = e0-einf/4, b = e0-einf/2, c = e0-einf,
dynamic{ X= a^b^c,}
```

Drag a, b, c around, and be convinced.

The outcome is interesting. The *direct* representation of the object perpendicular to other objects is the outer product of their *duals*. Shrinking the spheres to points, you see that you get a circle through them. So in this interpretation, *points are small dual spheres*, and to 'pass through' a point means to cut its corresponding *direct* sphere perpendicularly. This neatly unifies the point description with the spheres in one consistent scheme. This is a subtle point, and it pays to pause here a moment to let it sink it and make it your own.

Now let us revisit the object $p \cdot (c \wedge e_\infty)$. It was a dual sphere through the point p, with center c. Dualizing this, we see that it is the direct object

$$p \wedge (c \wedge e_\infty)^*.$$

With what we have just learned we see that this indeed contains p, and that we can think of it as being perpendicular to the flat point $c \wedge e_\infty$. Since the result has to be the sphere, this suggests the intuitive picture of Figure 6: a flat point has 'hairs' extending to infinity, and our object cuts them orthogonally. These hairs therefore help to construct an object consisting of points equidistant to c.

At the right of the figure, we see a similar explanation for the construction of the midplane between two points p and q, which is $q - p$, or written multiplicatively and dualized:

$$(q-p)^* = (e_\infty \cdot (p \wedge q))^* = e_\infty \wedge (p \wedge q)^*.$$

© The Eurographics Association 2004.

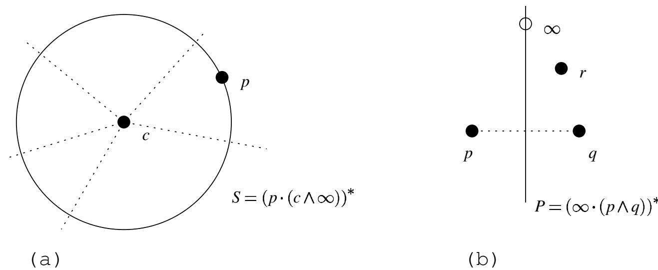


Figure 6: The specification of dual spheres and planes, see text.

Therefore the direct representation contains $\mathtt{e}_\infty$ and cuts $p \wedge q$ orthogonally, a fair description of the midplane.

If we replace $\mathtt{e}_\infty$ with a finite point r, we get $r \cdot (p \wedge q)$. Please explore its meaning yourself, and verify your insights using a small implementation.

With what we have learned, we can also interpret an object like

$$\mathtt{p} \wedge \mathtt{e1} \wedge \mathtt{e}_\infty.$$

It contains the points $\mathtt{p}$ and $\mathtt{e}_\infty$, and should be orthogonal to $\mathtt{e1}^*$, which is the $(\mathtt{e2} \wedge \mathtt{e3})$-plane through the origin. Obviously this is the line through $\mathtt{p}$ in the $\mathtt{e1}$ direction.

Now you can play with `DEMOortho();` and get some more intuition for the construction of such blades. It constructs the circle through p perpendicular to the planes dually characterized by e1 and e2, you would simply type:

```
dp1 = e1,
dp2 = e2,
p = e0,
dynamic{c = p^dp1^dp2,},
```

In the setup in the demo, this gives a 2-tangent, and as you move p a little you see that that is in a sense an infinitesimal circle. Then `DEMOortho();` proceeds to construct more elements (see its legenda in the upper left), which you should try to understand.

Let's get back to the incidence operation $A \cap B = -\mathbf{dual}(\mathbf{dual}B \wedge \mathbf{dual}A)$ from section 6.3. We now recognize the outcome as the dual of a blade that is the composed (by spanning) of elements perpendicular to both A and B. The result is therefore in both A and B.

We can use our new insights to show the relationship between the direct representation of a sphere as the outer product of four points $S = a \wedge b \wedge c \wedge d$, and the dual representation by a center m and a point a on it which is $s = a \cdot (m \wedge \mathtt{e}_\infty)$. This is illustrated in `DEMOspheres();`, which you may run in tandem with the algebraic explanation below.

We realize that the center should be the intersection of the midplanes of three points pairs. These midplanes are dually represented as $b-a$, $c-a$ and $d-a$, and the dual of their intersection is their outer product. However, this is not merely the center m, since $\mathtt{e}_\infty$ is also on all planes. Therefore:

$$(m \wedge \infty)^* = \alpha\,(b-a) \wedge (c-a) \wedge (d-a)$$

(which α some proportionality constant) This helps use relate the two immediately. The dual representation gives $0 = x \cdot s$, and we dualize this and rearrange:

$$\begin{aligned} 0 &= (x \cdot (a \cdot (m \wedge \infty)))^* \\ &= x \wedge (a \cdot (m \wedge \infty))^* \\ &= x \wedge \left(a \wedge (m \wedge \infty)^*\right) \\ &= \alpha\, x \wedge (a \wedge (b-a) \wedge (c-a) \wedge (d-a)) \\ &= \alpha\, x \wedge (a \wedge b \wedge c \wedge d) \end{aligned}$$

This is of the form $0 = x \wedge S$, so we have found the direct representation. Done! And this also shows why the representation space for 3D Euclidean geometry is 5-dimensional: the dual of a vector is a 4-blade.

It is rather satisfying that such geometrically involved computations can be done so simply in CGA, without even introducing coordinates!

11. Summary and conclusion

This concludes our cursory exploration of the blades of the conformal model of Euclidean geometry. As you have seen, many familiar but not-so-well defined elements from practical geometry find a home here, and are enriched by their relationship with other elements.

We have not touched upon operations and the corresponding calculus at all, and have done very little that could be seen as an 'application'. Our main goal was to give you a good understanding of the basics: why it works, and what makes it different from more classical methods.

If you want to learn more, you can try the more detailed tutorial at our website `http://www.science.uva.nl/ga/` of which tutorial this was just a selection. On our website, you can also find GABLE, a gentle introduction to 3D Geometric Algebra, and other tutorial papers.

© The Eurographics Association 2004.

PART THREE

A Mathematical Introduction to Geometric Algebra

Christian Perwass

This text is meant to be a script of a tutorial on Geometric Algebra. It is therefore not complete in the description of the algebra and neither completely rigorous. The reader is also not likely to be able to perform arbitrary calculations with Clifford algebra after reading this script. The goal of this text is to give the reader a feeling for what Clifford algebra is about and how it may be used. It is attempted to convey the basic ideas behind the use of Clifford algebra in the description of geometry in Euclidean and projective space. A more detailed introduction in the same style as this text can be found in [PH03].

There are also many other introductions to Clifford and Geometric Algebra and its applications in Euclidean, projective and conformal space. Some of these are [HS84, Hes86, HZ91, GLD93, Lou97, Rie93, GM91] and [Por95, LFLD98, Dor01, Mac99, Per00, MDB01]. A collection of papers discussing in particular the conformal space in detail and applications of Geometric Algebra in Computer Vision may be found in the book *Geometric Computing with Clifford Algebra* [Som01].

You can also try out the mathematics discussed in this text using **CLUCalc**. CLUCalc is a user friendly software tool to calculate with and visualize Geometric Algebra. It is available for download from [Per02]. In CLUCalc you can type your equations in a simple script language, called **CLUScript** and visualize the results immediately with OpenGL graphics. The program comes with a manual in HTML form and a number of example scripts. There is also an online version of the manual under:

`http://www.perwass.de/CLU/CLUCalcDoc/`

CLUCalc should serve as a good accompaniment to this script, helping you to understand the concepts behind Geometric Algebra visually.

CLUCalc is of course not the only software available that deals with Clifford or Geometric Algebra. Many software packages have been developed, because the numerical evaluation of Clifford algebra equations becomes more and more important as Clifford algebra becomes more prominent in applied fields like computer vision, computer graphics and robotics [LFLD98, Sel96, LL98, PL01, Dor01, Som01, DDL02]. There are packages for the symbolic computer algebra systems Maple [amo96, aF02] and Mathematica [Bro02], a package for the numerical mathematics program MatLab called GABLE [MDB01], the C++ software library generator Gaigen [FBD01], the C++ software library GluCat [Leo02], the Java library Clados [Dif02] and a stand alone program called CLICAL [Lou87], to name just a few.

By the way, CLUCalc was also used to create all of the 2d and 3d graphics in this section. You can use it for the same purpose, illustrating your publications or web-pages, from the version 3.0 onwards. Some other features of the current version CLUCalc v4.0.0 are:

- render and display LaTeX text and formulas to annotate your graphics, or to create slides for presentations,
- prepare presentations with user interactive 3D-graphics included in your slides,
- draw 2D-surfaces, including the surface generated by a set of circles,
- do structured programming with if-clauses and loops,
- do error propagation in Geometric Algebra,
- construct and visualize 2D-conic sections in a Geometric Algebra,
- and much more...

If you want to know more details, go to `www.clucalc.info` or simply send an email to `help@clucalc.info`.

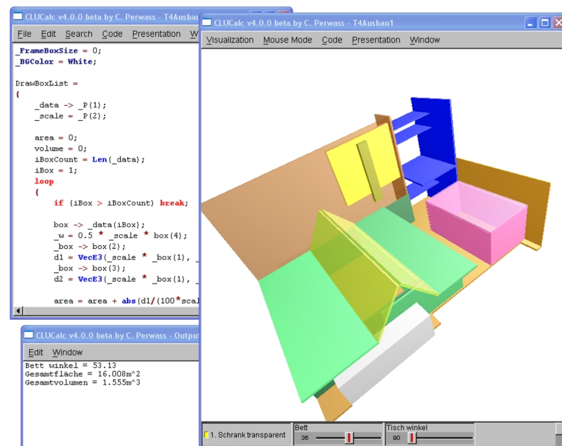


Figure 7: *A screenshot of CLUCalc v4.0.*

12. Introduction to Geometric Algebra

In this section we discuss the geometric interpretation of algebraic entities, since it is hoped that the reader's geometric intuition will further the understanding. A discussion of the algebra in a purely mathematical sense will not be given here. The reader should refer to [GM91, Por95] for an in depth discussion of the algebra from a purely mathematical point of view.

© The Eurographics Association 2004.

In this introduction we will neglect many algebraic aspects and introduce Geometric Algebra as an extension of the standard vector algebra. The actual algebra product is called "*geometric product*", but we will not start this discourse by discussing this product. Instead, we start by introducing the "*inner product*" and "*outer product*", which can be regarded as special "parts" of the geometric product. This "top-down" approach is hoped to show the applicability of the mathematics before giving a lot of details that may confuse the reader.

In the following the terms "*scalar product*" and "*inner product*" will be used quite often, and it is important to understand that in this text these two terms refer to quite different operations. Depending on which books you have read before, you may be used to employing these terms interchangeably. Here, a scalar product is a product which results in a scalar - no more, no less. This scalar is in general an element of $\mathbb{R}$, in particular it may also be zero or negative. This may, for example, occur if the basis of the vector space we are working in is not Euclidean. This will in fact turn up in conformal space.

The operation termed "*inner product*" here, may coincide with the scalar product, but represents in general an algebraic operation which does not result in a scalar. This will be explained further in section 12.4. One may also say that the scalar product is a "metric" operation, since it depends on a metric, while the inner product is an algebraic operation, which can also be executed without the knowledge of a metric.

So let's start with a 3d Euclidean vector space denoted by $\mathbb{E}^3$. We will use the coordinate representation $\mathbb{R}^3$ for $\mathbb{E}^3$. We assume that the standard scalar product is defined on $\mathbb{E}^3$. It will be denoted by $*$. Furthermore, the usual vector cross product exists on $\mathbb{E}^3$ and will be written as $\times$. Recall that the scalar product gives the length of the component two vectors have in common. The vector cross product, on the other hand, results in a vector perpendicular to both of the initial vectors. For example, let $\mathbf{a},\mathbf{b},\mathbf{c} \in \mathbb{E}^3$, then

$$\mathbf{a}*\mathbf{b} \in \mathbb{R} \quad \text{and} \quad \mathbf{a}\times\mathbf{b} \in \mathbb{E}^3.$$

Furthermore,

$$\mathbf{c} = \mathbf{a}\times\mathbf{b} \;\Rightarrow\; \mathbf{c}\perp\mathbf{a} \text{ and } \mathbf{c}\perp\mathbf{b}.$$

A plane in $\mathbb{E}^3$ is typically represented by its normal and an offset vector. Given two vectors that are to span a plane, the vector cross product can be used to find the plane's normal. However, this only works in 3d. In higher dimensions the (standard) vector cross product of two vectors is not defined.Nevertheless, we may be interested in describing the two dimensional subspace spanned by two vectors also in a n-dimensional vector space.

12.1. The Outer Product

Without explaining exactly what it is, we can define a Geometric Algebra on $\mathbb{R}^n$, $\mathcal{G}(\mathbb{R}^n)$ or simply $\mathcal{G}_n$ if it is clear that we are forming the Geometric Algebra over the reals. The latter will in fact be the case for the whole of this text.

The outer product is an operation defined within this algebra and is denoted by $\wedge$. Here are the properties of the outer product of vectors. Let $\mathbf{a},\mathbf{b},\mathbf{c} \in \mathbb{E}^n$.

$$\begin{aligned} \mathbf{a}\wedge\mathbf{b} &= -\mathbf{b}\wedge\mathbf{a} \\ (\mathbf{a}\wedge\mathbf{b})\wedge\mathbf{c} &= \mathbf{a}\wedge(\mathbf{b}\wedge\mathbf{c}) \\ \mathbf{a}\wedge(\mathbf{b}+\mathbf{c}) &= (\mathbf{a}\wedge\mathbf{b})+(\mathbf{a}\wedge\mathbf{c}). \end{aligned} \tag{1}$$

Another important property is

$$\mathbf{a}\wedge\mathbf{b} = 0 \iff \mathbf{a} \text{ and } \mathbf{b} \text{ are linearly dependent.} \tag{2}$$

Let $\{\mathbf{a}_1,\ldots,\mathbf{a}_k\} \subset \mathbb{R}^n$ be $k \leq n$ mutually linearly independent vectors. Then

$$(\mathbf{a}_1\wedge\mathbf{a}_2\wedge\ldots\wedge\mathbf{a}_k)\wedge\mathbf{b} = 0, \tag{3}$$

if and only if $\mathbf{b}$ is linearly dependent on $\{\mathbf{a}_1,\ldots,\mathbf{a}_k\}$. The outer product of k vectors is called a k-*blade* and is denoted by

$$A_{\langle k\rangle} = \mathbf{a}_1\wedge\mathbf{a}_2\wedge\ldots\wedge\mathbf{a}_k =: \bigwedge_{i=1}^{k} \mathbf{a}_i.$$

The *grade* of a blade is simply the number of vectors that "wedged" together give the blade. Hence, the outer product of k linearly independent vectors gives a blade of grade k, a k-blade.

12.2. The Outer Product Null Space

In Geometric Algebra, blades, as defined above, are given a geometric interpretation. This is based on their interpretation as linear subspaces. For example, given a vector $\mathbf{a} \in \mathbb{R}^n$, we can define a function $\mathcal{O}_\mathbf{a}$ as

$$\mathcal{O}_\mathbf{a} : \mathbf{x} \in \mathbb{R}^n \mapsto \mathbf{x}\wedge\mathbf{a} \in \mathcal{G}(\mathbb{R}^n).$$

The kernel of this function is the set of vectors in $\mathbb{R}^n$ that $\mathcal{O}_\mathbf{a}$ maps to zero. This kernel will be called the *outer product null space* (OPNS) of $\mathbf{a}$ and denoted by $\mathbb{NO}(\mathbf{a})$. That is,

$$\text{kern}\ \mathcal{O}_\mathbf{a} = \mathbb{NO}(\mathbf{a}) := \left\{\mathbf{x}\in\mathbb{R}^n \,:\, \mathbf{x}\wedge\mathbf{a} = 0 \in \mathcal{G}(\mathbb{R}^n)\right\}. \tag{4}$$

We already know that $\mathbf{x}\wedge\mathbf{a}$ is zero if and only if $\mathbf{x}$ is linearly dependent on $\mathbf{a}$. Therefore, $\mathbb{NO}(\mathbf{a})$ can also be given in terms of $\mathbf{a}$ as

$$\mathbb{NO}(\mathbf{a}) = \left\{\alpha\mathbf{a} \,:\, \alpha\in\mathbb{R}\right\},$$

which means that the OPNS of $\mathbf{a}$ is a *line through the origin with the direction of* $\mathbf{a}$. In Geometric Algebra it is therefore said that a vector in $\mathbb{E}^n$ represents a line.

Given a 2-blade $\mathbf{a}\wedge\mathbf{b} \in \mathcal{G}(\mathbb{R}^n)$, where $\mathbf{a},\mathbf{b}\in\mathbb{R}^n$, a function $\mathcal{O}_{\mathbf{a}\wedge\mathbf{b}}$ can be defined as

$$\mathcal{O}_{\mathbf{a}\wedge\mathbf{b}} : \mathbf{x}\in\mathbb{R}^n \mapsto \mathbf{x}\wedge\mathbf{a}\wedge\mathbf{b} \in \mathcal{G}(\mathbb{R}^n).$$

© The Eurographics Association 2004.

The kernel of this function is

$$\text{kern}\, \mathcal{O}_{\mathbf{a}\wedge\mathbf{b}} = \mathbb{NO}(\mathbf{a}\wedge\mathbf{b}) := \left\{\mathbf{x}\in\mathbb{R}^n \,:\, \mathbf{x}\wedge\mathbf{a}\wedge\mathbf{b} = 0 \in \mathcal{G}(\mathbb{R}^n)\right\}. \quad (5)$$

As before, it follows that the OPNS of $\mathbf{a}\wedge\mathbf{b}$ can be parameterized as follows

$$\mathbb{NO}(\mathbf{a}\wedge\mathbf{b}) = \left\{\alpha\mathbf{a}+\beta\mathbf{b} \,:\, (\alpha,\beta)\in\mathbb{R}^2\right\}.$$

Hence, $\mathbf{a}\wedge\mathbf{b}$ is said to represent the two-dimensional subspace of $\mathbb{R}^n$ spanned by $\mathbf{a}$ and $\mathbf{b}$, ie a plane through the origin. In general the OPNS of some k-blade $A_{\langle k\rangle}\in\mathcal{G}(\mathbb{R}^n)$ is a k-dimensional linear subspace of $\mathbb{R}^n$.

$$\mathbb{NO}(A_{\langle k\rangle}) := \left\{\mathbf{x}\in\mathbb{R}^n \,:\, \mathbf{x}\wedge A_{\langle k\rangle} = 0\right\}.$$

Consider again the three-dimensional Euclidean space $\mathbb{E}^3$ with $\mathbf{a},\mathbf{b},\mathbf{c}\in\mathbb{E}^3$ three mutually linearly independent vectors. Hence, $\{\mathbf{a},\mathbf{b},\mathbf{c}\}$ form a basis of $\mathbb{E}^3$. Then

$$\begin{aligned}\mathbb{NO}(\mathbf{a}\wedge\mathbf{b}\wedge\mathbf{c}) &:= \left\{\mathbf{x}\in\mathbb{E}^3 \,:\, \mathbf{x}\wedge\mathbf{a}\wedge\mathbf{b}\wedge\mathbf{c} = 0\in\mathcal{G}(\mathbb{R}^3)\right\}\\ &= \left\{\alpha\mathbf{a}+\beta\mathbf{b}+\gamma\mathbf{c}\in\mathbb{E}^3 \,:\, (\alpha,\beta,\gamma)\in\mathbb{R}^3\right\}.\end{aligned}$$

Therefore, the OPNS of $\mathbf{a}\wedge\mathbf{b}\wedge\mathbf{c}$ is the whole space $\mathbb{E}^3$. Since the OPNS of the outer product of any basis of $\mathbb{E}^3$ is the whole space $\mathbb{E}^3$, the blades created from different bases have to be similar. In fact, they only differ by a scalar factor. A blade of grade n in some $\mathcal{G}(\mathbb{R}^n)$ is called a *pseudoscalar*. "Pseudoscalar" because all pseudoscalars only differ by a scalar factor, just like the scalar element $1\in\mathcal{G}(\mathbb{R}^n)$.

> **Aside.** Note that the fact that $\mathbb{NO}\big(A_{\langle n\rangle}\in\mathcal{G}(\mathbb{R}^n)\big) = \mathbb{R}^n$, implies that no blades of grade higher than n can exist in $\mathcal{G}(\mathbb{R}^n)$.

12.3. Magnitude of Blades

On the Euclidean space $\mathbb{E}^n$ the norm typically used is the L_2 norm. This is defined in terms of the scalar product. Let $\mathbf{a}\in\mathbb{E}^n$, then

$$\|\mathbf{a}\|_2 := \sqrt{\mathbf{a}*\mathbf{a}}. \quad (6)$$

This norm can also be extended to blades in $\mathcal{G}(\mathbb{E}^n)$. We will not give a proper derivation here, but try to motivate the definition. In the following we will also use $\|.\|$ instead $\|.\|_2$ for brevity. Let $\mathbf{a},\mathbf{b}\in\mathbb{R}^3$ and denote by $\mathbf{b}^\perp$ and $\mathbf{b}^\parallel$ the parts of $\mathbf{b}=\mathbf{b}^\perp+\mathbf{b}^\parallel$ that are perpendicular and parallel to $\mathbf{a}$, respectively. Then

$$\begin{aligned}\mathbf{a}\wedge\mathbf{b} &= \mathbf{a}\wedge(\mathbf{b}^\perp+\mathbf{b}^\parallel)\\ &= \mathbf{a}\wedge\mathbf{b}^\perp + \underbrace{\mathbf{a}\wedge\mathbf{b}^\parallel}_{=0}\\ &= \mathbf{a}\wedge\mathbf{b}^\perp.\end{aligned} \quad (7)$$

Similarly, for any k-blade $A_{\langle k\rangle} = \bigwedge_{i=1}^k \mathbf{a}_i$, we can find a set of k mutually orthogonal vectors $\{\mathbf{a}'_1,\ldots,\mathbf{a}'_k\}$, such that

$$A_{\langle k\rangle} = A'_{\langle k\rangle} := \bigwedge_{i=1}^k \mathbf{a}'_i.$$

Now, it may be shown that

$$\|A_{\langle k\rangle}\| = \|A'_{\langle k\rangle}\| = \sqrt{\prod_{i=1}^k (\mathbf{a}'_i)^2} = \prod_{i=1}^k \|\mathbf{a}'_i\|, \quad (8)$$

with $k>0$. Since the $\{\mathbf{a}'_i\}$ are mutually orthogonal, the norm or magnitude of $A_{\langle k\rangle}$ is the "volume" spanned by them. For $k=1$ this reduces to the norm of a vector.

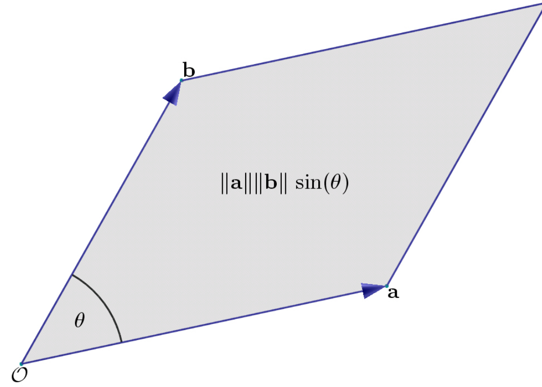


Figure 8: *Area of bivector.*

An illustrative example is the norm of a 2-blade (also called *bivector*). The bivector $\mathbf{a}\wedge\mathbf{b}\in\mathcal{G}(\mathbb{R}^n)$ may also be written as $\mathbf{a}\wedge\mathbf{b}^\perp$, where $\mathbf{b}^\perp$ is the component of $\mathbf{b}$ that is perpendicular to $\mathbf{a}$. Then $\|\mathbf{b}^\perp\| = \sin\theta\,\|\mathbf{b}\|$, with $\theta = \angle(\mathbf{a},\mathbf{b})$. Therefore,

$$\|\mathbf{a}\wedge\mathbf{b}\| = \|\mathbf{a}\wedge\mathbf{b}^\perp\| = \|\mathbf{a}\|\,\|\mathbf{b}\|\,\sin\theta,$$

which is the area of the parallelogram spanned by $\mathbf{a}$ and $\mathbf{b}$.

Now consider a $n\times k$ matrix A, whose columns are the $\{\mathbf{a}_i\}_{i=1}^k\subset\mathbb{R}^n$. This will be written as $\mathsf{A}=[\mathbf{a}_1,\ldots,\mathbf{a}_k]$. We could now define the norm of such a matrix to be the "volume" of the parallelepiped spanned by its column vectors. This would then be in accordance with the norm of a blade of these vectors. In fact, for a matrix $\mathsf{B}=[\mathbf{b}_1,\ldots,\mathbf{b}_n]$, where the $\{\mathbf{b}_i\}_{i=1}^n\subset\mathbb{R}^n$ are a basis of $\mathbb{R}^n$, the determinant of B, $\det(\mathsf{B})$ does give the volume of the parallelepiped spanned by the $\{\mathbf{b}_i\}_{i=1}^n$. Therefore, in this case,

$$\|\,\mathbf{b}_1\wedge\ldots\wedge\mathbf{b}_n\,\| = \det([\mathbf{b}_1,\ldots,\mathbf{b}_n]).$$

The unit pseudoscalar of some $\mathcal{G}(\mathbb{R}^n)$, is a blade of grade n with magnitude 1 and is usually denoted by I. Therefore, for example,

$$\mathbf{b}_1\wedge\ldots\wedge\mathbf{b}_n = \|\,\mathbf{b}_1\wedge\ldots\wedge\mathbf{b}_n\,\|\,I = \det([\mathbf{b}_1,\ldots,\mathbf{b}_n])\,I.$$

12.4. The Inner Product

Another important operation in Geometric Algebra is the *inner product*. The inner product will be denoted by $\cdot$. For vectors $\mathbf{a},\mathbf{b}\in\mathbb{R}^n$, their inner product is just the same as their

© The Eurographics Association 2004.

scalar product, ie

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a} * \mathbf{b}.$$

This may be called the "metric" property of the inner product, since the result of the scalar product of two vectors depends on the metric of the vector space they lie in. However, the inner product also has some purely algebraic properties for elements in $\mathcal{G}(\mathbb{R}^n)$, which are independent of the metric of the vector space $\mathbb{R}^n$. In the following a number of these properties are stated without proof.

Let $\mathbf{a},\mathbf{b},\mathbf{c} \in \mathbb{R}^n$, then the bivector $\mathbf{b}\wedge\mathbf{c} \in \mathcal{G}(\mathbb{R}^n)$. The inner product of $\mathbf{a}$ with this bivector gives,

$$\mathbf{a} \cdot (\mathbf{b}\wedge\mathbf{c}) = (\mathbf{a}\cdot\mathbf{b})\,\mathbf{c} - (\mathbf{a}\cdot\mathbf{c})\,\mathbf{b}. \tag{9}$$

Since $(\mathbf{a}\cdot\mathbf{b})$ and $(\mathbf{a}\cdot\mathbf{c})$ are scalars, we see that the inner product of a vector with a bivector results in a vector. More generally it may be shown that for $k \geq 1$

$$\begin{aligned}
\mathbf{x}\cdot A_{\langle k\rangle} \;=\; & \;(\mathbf{x}\cdot\mathbf{a}_1)\,(\mathbf{a}_2\wedge\mathbf{a}_3\wedge\mathbf{a}_4\wedge\ldots\wedge\mathbf{a}_k)\\
& -\;(\mathbf{x}\cdot\mathbf{a}_2)\,(\mathbf{a}_1\wedge\mathbf{a}_3\wedge\mathbf{a}_4\wedge\ldots\wedge\mathbf{a}_k)\\
& +\;(\mathbf{x}\cdot\mathbf{a}_3)\,(\mathbf{a}_1\wedge\mathbf{a}_2\wedge\mathbf{a}_4\wedge\ldots\wedge\mathbf{a}_k)\\
& -\;\text{etc.}\\
=\; & \sum_{i=1}^{k}(-1)^{(i+1)}\,(\mathbf{x}\cdot\mathbf{a}_i)\,\big[A_{\langle k\rangle}\setminus\mathbf{a}_i\big],
\end{aligned} \tag{10}$$

where $[A_{\langle k\rangle}\setminus\mathbf{a}_i]$ denotes the blade $A_{\langle k\rangle}$ without the vector $\mathbf{a}_i$. Here the inner product of a vector with a k-blade results in a $(k-1)$-blade. An example of another important rule is this

$$(\mathbf{a}\wedge\mathbf{b})\cdot A_{\langle k\rangle} = \mathbf{a}\cdot\big(\mathbf{b}\cdot A_{\langle k\rangle}\big), \tag{11}$$

with $k \geq 2$. More generally, the inner product of blades $A_{\langle k\rangle}, B_{\langle l\rangle} \in \mathcal{G}(\mathbb{R}^n)$, with $0 < k \leq l \leq n$, can be expanded as

$$A_{\langle k\rangle}\cdot B_{\langle l\rangle} = \mathbf{a}_1\cdot\Big(\mathbf{a}_2\cdot\big(\ldots\cdot(\mathbf{a}_k\cdot B_{\langle l\rangle})\big)\Big). \tag{12}$$

Hence, the result of this inner product is a $(l-k)$-blade.

In comparison to the outer product we see that the inner and the outer product are antagonists: while the outer product increases the grade of a blade, the inner product reduces it.

12.5. The Inverse of a Blade

Similar to the formula for vectors, the inverse of a blade $A_{\langle k\rangle} \in \mathcal{G}(\mathbb{R}^n)$, $k \leq n$, is in general given by

$$A_{\langle k\rangle}^{-1} := \frac{\widetilde{A}_{\langle k\rangle}}{\|A_{\langle k\rangle}\|^2},$$

as long as $\|A_{\langle k\rangle}\| \neq 0$. Note that the magnitude of a blade can in fact become zero in Minkowski spaces. Using this formula it may indeed be shown that

$$A_{\langle k\rangle}\cdot A_{\langle k\rangle}^{-1} = A_{\langle k\rangle}^{-1}\cdot A_{\langle k\rangle} = 1.$$

The symbol $\widetilde{A}_{\langle k\rangle}$ denotes the *reverse* of a blade. The reverse is an operator that simply reverses the order of vectors in a blade. For example, if $A_{\langle k\rangle} = \bigwedge_{i=1}^{k}\mathbf{a}_i$ then

$$\widetilde{A}_{\langle k\rangle} = \bigwedge_{i=k}^{1}\mathbf{a}_i = \mathbf{a}_k\wedge\mathbf{a}_{k-1}\wedge\ldots\wedge\mathbf{a}_1. \tag{13}$$

Since the outer product is associative and anti-commutative, the reordering of vectors in a blade can only change the blade's sign. For the reverse we find in particular

$$\widetilde{A}_{\langle k\rangle} = (-1)^{k(k-1)/2}\,A_{\langle k\rangle}. \tag{14}$$

So, why do we need the reverse in the definition of the inverse of a blade? The answer is, that the reverse takes care of a sign that is introduced due to the grade of a blade. As an example consider the orthonormal basis $\{\mathsf{e}_i\}$ of $\mathbb{R}^n$. From equations (12) and (10) it follows that

$$\begin{aligned}
(\mathsf{e}_1\wedge\mathsf{e}_2)\cdot(\mathsf{e}_1\wedge\mathsf{e}_2) &= \mathsf{e}_1\cdot\big((\mathsf{e}_2\cdot\mathsf{e}_1)\mathsf{e}_2-(\mathsf{e}_2\cdot\mathsf{e}_2)\mathsf{e}_1\big)\\
&= \mathsf{e}_1\cdot\big(-\mathsf{e}_1\big)\\
&= -1.
\end{aligned}$$

On the other hand, obviously $\mathsf{e}_1\cdot\mathsf{e}_1 = 1$. That is, depending on the grade of a blade (a vector being a blade of grade 1), an additional sign is introduced or not. This is fixed by the reverse. Given any blade $A_{\langle k\rangle} \in \mathcal{G}(\mathbb{R}^n)$, then

$$A_{\langle k\rangle}\cdot\widetilde{A}_{\langle k\rangle} = \|A_{\langle k\rangle}\|^2,$$

whereas

$$A_{\langle k\rangle}\cdot A_{\langle k\rangle} = (-1)^{k(k-1)/2}\,\|A_{\langle k\rangle}\|^2. \tag{15}$$

12.6. Geometric Interpretation of Inner Product

We can already get an idea of what is happening by looking at the Geometric Algebra of $\mathbb{R}^2$, $\mathcal{G}(\mathbb{R}^2)$ with orthonormal basis $\{\mathsf{e}_1,\mathsf{e}_2\}$. The outer product $\mathsf{e}_1\wedge\mathsf{e}_2$ spans the whole space, ie a plane. Now let's look at the inner product of e_1 with this bivector.

$$\mathsf{e}_1\cdot(\mathsf{e}_1\wedge\mathsf{e}_2) = (\mathsf{e}_1\cdot\mathsf{e}_1)\,\mathsf{e}_2 - (\mathsf{e}_1\cdot\mathsf{e}_2)\,\mathsf{e}_1 = \mathsf{e}_2. \tag{16}$$

This may be interpreted as "subtracting" the subspace represented by e_1 from the subspace represented by $\mathsf{e}_1\wedge\mathsf{e}_2$. What is left after the subtraction is, of course, perpendicular to e_1.

More generally, let $\mathbf{x},\mathbf{y},\mathbf{a},\mathbf{b} \in \mathbb{R}^n$ and let

$$\mathbf{y} = \mathbf{x}\cdot(\mathbf{a}\wedge\mathbf{b}) = (\mathbf{x}\cdot\mathbf{a})\,\mathbf{b} - (\mathbf{x}\cdot\mathbf{b})\,\mathbf{a}.$$

Now we find that

$$\begin{aligned}
\mathbf{x}\cdot\mathbf{y} &= \mathbf{x}\cdot\big[(\mathbf{x}\cdot\mathbf{a})\,\mathbf{b} - (\mathbf{x}\cdot\mathbf{b})\,\mathbf{a}\big]\\
&= (\mathbf{x}\cdot\mathbf{a})\,(\mathbf{x}\cdot\mathbf{b}) - (\mathbf{x}\cdot\mathbf{b})\,(\mathbf{x}\cdot\mathbf{a})\\
&= 0.
\end{aligned}$$

That is, $\mathbf{x}$ is perpendicular to $\mathbf{y}$, which again implies that the inner product $\mathbf{x}\cdot(\mathbf{a}\wedge\mathbf{b})$ "subtracted" the subspace represented by $\mathbf{x}$ from the subspace represented by $\mathbf{a}\wedge\mathbf{b}$. This can also be illustrated quite nicely in $\mathbb{E}^3$.

© The Eurographics Association 2004.

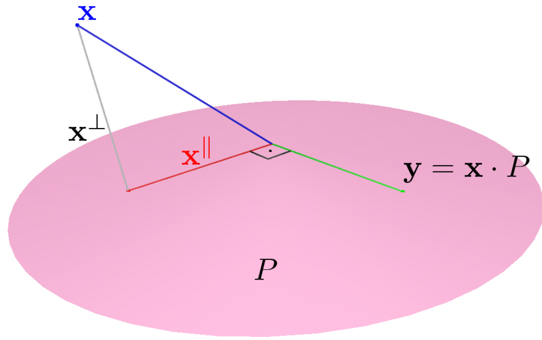


Figure 9: *Inner product of vector and bivector.*

Let P denote the bivector $\mathbf{a} \wedge \mathbf{b} \in \mathcal{G}(\mathbb{R}^3)$. In $\mathbb{E}^3$ this bivector represents a plane through the origin, as shown in figure 9. A vector $\mathbf{x} \in \mathbb{R}^3$ will in general have a component parallel to P, $\mathbf{x}^{\parallel}$, and a component perpendicular to P, $\mathbf{x}^{\perp}$, such that $\mathbf{x} = \mathbf{x}^{\parallel} + \mathbf{x}^{\perp}$. Therefore,

$$\mathbf{y} := \mathbf{x} \cdot P = (\mathbf{x}^{\parallel} + \mathbf{x}^{\perp}) \cdot P = \mathbf{x}^{\parallel} \cdot P.$$

The inner product $\mathbf{x}^{\parallel} \cdot P$ now "subtracts" the subspace represented by $\mathbf{x}^{\parallel}$ from the subspace represented by P, which results in a vector that lies in P and is perpendicular to $\mathbf{x}$, as shown in figure 9.

12.7. The Inner Product Null Space

Just as for the outer product, we can also define the null space of blades with respect to the inner product. The *inner product null space* (IPNS) of a blade $A_{\langle k \rangle} \in \mathcal{G}(\mathbb{R}^n)$, denoted by $\mathbb{NI}(A_{\langle k \rangle})$, is the kernel of the function $\mathcal{I}_{A_{\langle k \rangle}}$ defined as

$$\mathcal{I}_{A_{\langle k \rangle}} : \mathbf{x} \in \mathbb{R}^n \mapsto \mathbf{x} \cdot A_{\langle k \rangle} \in \mathcal{G}(\mathbb{R}^n), \tag{17}$$

and thus

$$\mathbb{NI}(A_{\langle k \rangle}) := \left\{ \mathbf{x} \in \mathbb{R}^n : \mathcal{I}_{A_{\langle k \rangle}}(\mathbf{x}) = 0 \in \mathcal{G}(\mathbb{R}^n) \right\}. \tag{18}$$

For example, consider a vector $\mathbf{a} \in \mathbb{R}^3$, then $\mathbb{NI}(\mathbf{a})$ is given by

$$\mathbb{NI}(\mathbf{a}) := \left\{ \mathbf{x} \in \mathbb{R}^3 : \mathbf{x} \cdot \mathbf{a} = 0 \right\}.$$

That is, all vectors that are perpendicular to $\mathbf{a}$ belong to its IPNS. In $\mathbb{R}^3$ the IPNS of $\mathbf{a}$ is therefore a plane of which $\mathbf{a}$ is the *normal*. Earlier we already saw that the OPNS of a bivector represents a plane. This implies that there has to be some kind of relationship between the IPNS of a vector in $\mathbb{R}^3$ and the OPNS of a bivector in $\mathcal{G}(\mathbb{R}^3)$.

12.8. The Dual

Let $\{\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3\}$ denote again an orthonormal basis of $\mathbb{R}^3$. The IPNS of e_1 is the set of all vectors that are perpendicular to e_1. Hence,

$$\mathbb{NI}(\mathsf{e}_1) = \left\{ \alpha \mathsf{e}_2 + \beta \mathsf{e}_3 : (\alpha, \beta) \in \mathbb{R}^2 \right\},$$

the plane spanned by e_2 and e_3. However, we know that this is also the OPNS of $\mathsf{e}_2 \wedge \mathsf{e}_3$,

$$\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3) = \left\{ \alpha \mathsf{e}_2 + \beta \mathsf{e}_3 : (\alpha, \beta) \in \mathbb{R}^2 \right\}.$$

We may therefore ask whether there is a relation between the concepts of the IPNS and the OPNS. Such a relation does indeed exist and it is called *duality*. In the following we will see how this comes about.

Before we start with the actual calculations, we will introduce two set operations for sets of vectors that will become quite useful. The first is the direct sum of two sets of vectors denoted by $\oplus$. Given two sets $\mathbb{A} := \{\mathbf{a}_i\}_{i=1}^k \subset \mathbb{R}^n$ and $\mathbb{B} := \{\mathbf{b}_i\}_{i=1}^l \subset \mathbb{R}^n$ their direct sum is

$$\mathbb{A} \oplus \mathbb{B} := \left\{ \mathbf{a}_i + \mathbf{b}_j \in \mathbb{R}^n : 0 < i \leq k, 0 < j \leq l \right\}. \tag{19}$$

In particular this means for two infinite sets, ie one dimensional subspaces

$$\mathbb{A} := \left\{ \alpha \mathbf{a} \in \mathbb{R}^n : \alpha \in \mathbb{R} \right\}, \quad \text{and} \quad \mathbb{B} := \left\{ \beta \mathbf{b} \in \mathbb{R}^n : \beta \in \mathbb{R} \right\},$$

that their direct sum is the set of all linear combinations of the elements of $\mathbb{A}$ and $\mathbb{B}$. That is,

$$\mathbb{A} \oplus \mathbb{B} = \left\{ \alpha \mathbf{a} + \beta \mathbf{b} \in \mathbb{R}^n : (\alpha, \beta) \in \mathbb{R}^2 \right\}.$$

In this spirit it makes sense also to define a "direct subtraction" between two such sets as

$$\mathbb{A} \ominus \mathbb{B} := \left\{ \mathbf{x} \in \mathbb{A} : \mathbf{x} * \mathbf{y} = 0 \; \forall \mathbf{y} \in \mathbb{B} \right\}, \tag{20}$$

where we assume that a scalar product is defined on the elements of $\mathbb{A}$ and $\mathbb{B}$. Hence, the direct subtraction removes the linear dependence on elements of $\mathbb{B}$ from the elements of $\mathbb{A}$. Note that this is more than just to remove the elements of $\mathbb{B}$ from $\mathbb{A}$.

Now let us return to the question of duality. First of all note that the OPNS of e_1 is simply

$$\mathbb{NO}(\mathsf{e}_1) = \left\{ \alpha \mathsf{e}_1 : \alpha \in \mathbb{R} \right\},$$

a line through the origin with direction e_1. The direct sum of $\mathbb{NO}(\mathsf{e}_1)$ and $\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3)$ is the whole space $\mathbb{R}^3$,

$$\mathbb{NO}(\mathsf{e}_1) \oplus \mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3) = \left\{ \alpha \mathsf{e}_1 + \beta \mathsf{e}_2 + \gamma \mathsf{e}_3 : (\alpha, \beta, \gamma) \in \mathbb{R}^3 \right\} \equiv \mathbb{R}^3.$$

and, in particular, "removing" the linear dependence on $\mathbb{NO}(\mathsf{e}_1)$ from $\mathbb{R}^3$ gives
$\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3)$,

$$\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3) = \mathbb{R}^3 \ominus \mathbb{NO}(\mathsf{e}_1).$$

© The Eurographics Association 2004.

With respect to $\mathbb{R}^3$, $\mathbb{NO}(\mathsf{e}_1)$ may therefore be called the *complement* set to $\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3)$. Furthermore,

$$\mathbb{NI}(\mathsf{e}_1) = \mathbb{R}^3 \ominus \mathbb{NO}(\mathsf{e}_1).$$

The question now is: can we find an operation in $\mathcal{G}(\mathbb{R}^n)$ which transforms any blade $A_{\langle k \rangle} \in \mathcal{G}(\mathbb{R}^n)$ into a complementary blade $B_{\langle n-k \rangle} \in \mathcal{G}(\mathbb{R}^n)$, such that

$$\mathbb{NO}(A_{\langle k \rangle}) = \mathbb{R}^n \ominus \mathbb{NO}(B_{\langle n-k \rangle}).$$

Such an operation does indeed exist and is called the *dual*. The dual of a multivector $A \in \mathcal{G}(\mathbb{R}^n)$ is written A^* and is defined as

$$A^* := A \cdot I^{-1}, \tag{21}$$

where I^{-1} is the inverse unit pseudoscalar of $\mathcal{G}(\mathbb{R}^n)$. It is a nice feature of Geometric Algebra that the dual can be given as a standard product with a particular element of the algebra. However, this has also the drawback that the dual of the dual of a multivector may introduce an additional sign. That is,

$$(A^*)^* = (A \cdot I^{-1}) \cdot I^{-1} = A(I^{-1} \cdot I^{-1}).$$

Why the last step in this equation works will be shown later on in equation (31), page 26. If we believe this equation for the moment, then it shows that an additional sign is introduced whenever $I^{-1} \cdot I^{-1} = -1$. Since I^{-1} is a n-blade in $\mathcal{G}(\mathbb{R}^n)$ we know from equations (14) and (15) that

$$I^{-1} \cdot I^{-1} = (-1)^{k(k-1)/2} \, \|I^{-1}\|^2 = (-1)^{k(k-1)/2}.$$

With respect to the orthonormal basis $\{\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3\}$ of $\mathbb{R}^3$, the dual operation has the following effect. Consider again the bivector $\mathsf{e}_2 \wedge \mathsf{e}_3$ which represents the plane spanned by e_1 and e_2 in its OPNS. The unit pseudoscalar of $\mathbb{R}^3$ and its inverse may be given as

$$I = \mathsf{e}_1 \wedge \mathsf{e}_2 \wedge \mathsf{e}_3 \quad \text{and} \quad I^{-1} = \widetilde{I} = \mathsf{e}_3 \wedge \mathsf{e}_2 \wedge \mathsf{e}_1 = -I.$$

Now, the dual of $\mathsf{e}_2 \wedge \mathsf{e}_3$ is

$$\begin{aligned} (\mathsf{e}_2 \wedge \mathsf{e}_3)^* &= (\mathsf{e}_2 \wedge \mathsf{e}_3) \cdot I^{-1} \\ &= (\mathsf{e}_2 \wedge \mathsf{e}_3) \cdot (\mathsf{e}_3 \wedge \mathsf{e}_2 \wedge \mathsf{e}_1) \\ &= \mathsf{e}_2 \cdot \big(\mathsf{e}_3 \cdot (\mathsf{e}_3 \wedge \mathsf{e}_2 \wedge \mathsf{e}_1)\big), \end{aligned}$$

where we used equation (12). We first evaluate the term within the outer brackets using equation (10).

$$\begin{aligned} \mathsf{e}_3 \cdot (\mathsf{e}_3 \wedge \mathsf{e}_2 \wedge \mathsf{e}_1) &= (\mathsf{e}_3 \cdot \mathsf{e}_3)(\mathsf{e}_2 \wedge \mathsf{e}_1) - (\mathsf{e}_3 \cdot \mathsf{e}_2)(\mathsf{e}_3 \wedge \mathsf{e}_1) \\ &\quad + (\mathsf{e}_3 \cdot \mathsf{e}_1)(\mathsf{e}_3 \wedge \mathsf{e}_2) \\ &= \mathsf{e}_2 \wedge \mathsf{e}_1. \end{aligned}$$

Therefore,

$$\begin{aligned} (\mathsf{e}_2 \wedge \mathsf{e}_3)^* &= \mathsf{e}_2 \cdot (\mathsf{e}_2 \wedge \mathsf{e}_1) \\ &= (\mathsf{e}_2 \cdot \mathsf{e}_2)\mathsf{e}_1 - (\mathsf{e}_2 \cdot \mathsf{e}_1)\mathsf{e}_2 \\ &= \mathsf{e}_1. \end{aligned}$$

This is a nice example to see that the dual of a blade gives a blade complementing the whole space. In this case

$$(\mathsf{e}_2 \wedge \mathsf{e}_3) \wedge (\mathsf{e}_2 \wedge \mathsf{e}_3)^* = I,$$

the unit pseudoscalar. With respect to the OPNS we have

$$\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3) \oplus \mathbb{NO}\big((\mathsf{e}_2 \wedge \mathsf{e}_3)^*\big) = \mathbb{R}^3.$$

It is now also clear that the relation between the OPNS and IPNS is the duality. For example, we have seen before that

$$\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3) = \mathbb{R}^3 \ominus \mathbb{NO}(\mathsf{e}_1) = \mathbb{NI}(\mathsf{e}_1).$$

Since $\mathsf{e}_1 = (\mathsf{e}_2 \wedge \mathsf{e}_3)^*$ we have

$$\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3) = \mathbb{NI}\big((\mathsf{e}_2 \wedge \mathsf{e}_3)^*\big).$$

In general we have for some blade $A_{\langle k \rangle} \in \mathcal{G}(\mathbb{R}^n)$

$$\mathbb{NO}(A_{\langle k \rangle}) = \mathbb{NI}(A^*_{\langle k \rangle}). \tag{22}$$

12.9. Geometric Interpretation of the IPNS

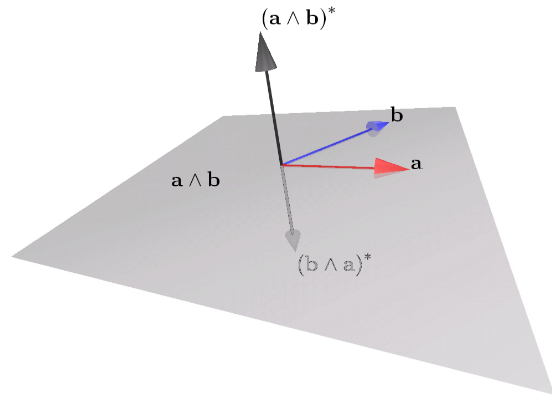


Figure 10: *Dual of plane represented by bivector* $\mathbf{a} \wedge \mathbf{b}$*.*

We have already seen that the IPNS of some vector $\mathbf{n} \in \mathbb{R}^3$ is a plane through the origin, whereby $\mathbf{n}$ is the plane's normal. With respect to the dual operation, it was shown in the previous section that the normal of a plane spanned by $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$, is given by $(\mathbf{a} \wedge \mathbf{b})^*$. Suppose that $\mathbf{n} = (\mathbf{a} \wedge \mathbf{b})^*$. The side of the plane $\mathbf{a} \wedge \mathbf{b}$ from which the normal $\mathbf{n}$ sticks out from is usually regarded as the "front"-side of the plane. Thus, a bivector represents a *sided* plane. For example, the normal $\mathbf{m}$ of $\mathbf{b} \wedge \mathbf{a}$ is given by

$$\mathbf{m} = (\mathbf{b} \wedge \mathbf{a})^* = -(\mathbf{a} \wedge \mathbf{b})^* = -\mathbf{n}.$$

Hence, the plane represented by $\mathbf{b} \wedge \mathbf{a}$ consists of the same subspace in $\mathbb{R}^3$ as the plane represented by $\mathbf{a} \wedge \mathbf{b}$, but their front-sides point in opposite directions. This situation is shown in figure 10. This also shows the relation between the vector cross product and the outer product:

$$\mathbf{a} \times \mathbf{b} = (\mathbf{a} \wedge \mathbf{b})^*.$$

Aside. Note that the idea of a plane normal vector does only work in $\mathbb{R}^3$. In any dimension higher than three the set of vectors perpendicular to one

© The Eurographics Association 2004.

vector spans a higher dimensional space than a plane. Nevertheless, a bivector always describes a plane, independent of the dimension it is embedded in.

Now that we are happy that a vector in $\mathbb{R}^3$ represents a plane with respect to its IPNS, we can ask what the IPNS of blades of higher grade is. Consider the non-zero bivector $\mathbf{a} \wedge \mathbf{b} \in \mathcal{G}(\mathbb{R}^3)$. In order to give its IPNS we have to find which vectors $\mathbf{x} \in \mathbb{R}^3$ satisfy $\mathbf{x} \cdot (\mathbf{a} \wedge \mathbf{b}) = 0$. With the help of equation (10) we find

$$\mathbf{x} \cdot (\mathbf{a} \wedge \mathbf{b}) = (\mathbf{x} \cdot \mathbf{a})\, \mathbf{b} - (\mathbf{x} \cdot \mathbf{b})\, \mathbf{a}.$$

Since we assumed that $\mathbf{a} \wedge \mathbf{b} \neq 0$, $\mathbf{a}$ and $\mathbf{b}$ have to be linearly independent. Therefore, the above expression can only become zero if and only if

$$\mathbf{x} \cdot \mathbf{a} = 0 \quad \text{and} \quad \mathbf{x} \cdot \mathbf{b} = 0.$$

Geometrically this means that $\mathbf{x}$ has to lie on the plane represented by $\mathbf{a}$ *and* on the plane represented by $\mathbf{b}$, in their IPNS. Hence, $\mathbf{x}$ lies on the intersection of the two planes represented by $\mathbf{a}$ and $\mathbf{b}$. This shows that the outer product of two vectors represents the intersection of their separately represented geometric entities. In terms of sets this reads

$$\mathbb{NI}(\mathbf{a} \wedge \mathbf{b}) = \mathbb{NI}(\mathbf{a}) \cap \mathbb{NI}(\mathbf{b}). \tag{23}$$

Such an intersection line also has an orientation, which in this case is given by $(\mathbf{b} \wedge \mathbf{a})^*$.

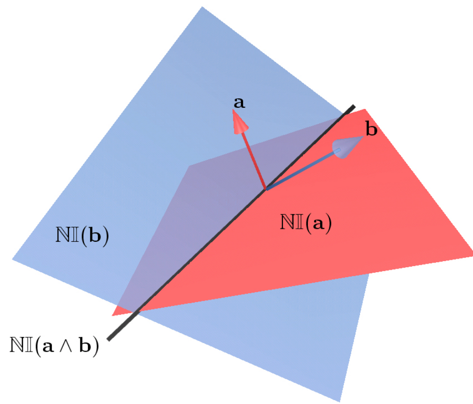


Figure 11: *Intersection of two planes in terms of IPNS.*

Aside. Note that in $\mathbb{R}^3$ we cannot represent two parallel but not identical planes through the IPNS of two vectors, since all such planes go through the origin.

The last type of blade we can discuss in $\mathbb{R}^3$ with respect to its IPNS is a 3-blade, or *trivector*. As we have seen already a trivector $A_{\langle 3 \rangle} \in \mathcal{G}(\mathbb{R}^3)$ is a pseudoscalar and thus

$$A_{\langle 3 \rangle} = \|A_{\langle 3 \rangle}\|\, I,$$

where I is the unit-pseudoscalar of $\mathcal{G}(\mathbb{R}^3)$. Let $A_{\langle 3 \rangle}$ be given by

$$A_{\langle 3 \rangle} := \mathbf{a} \wedge \mathbf{b} \wedge \mathbf{c}.$$

If $A_{\langle 3 \rangle} \neq 0$ then $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are linearly independent. In order to find the IPNS of $A_{\langle 3 \rangle}$, we need to find which vectors $\mathbf{x}$ satisfy $\mathbf{x} \cdot A_{\langle 3 \rangle} = 0$. Using again equation (10) it follows

$$\begin{aligned} \mathbf{x} \cdot A_{\langle 3 \rangle} = \; & (\mathbf{x} \cdot \mathbf{a})\,(\mathbf{b} \wedge \mathbf{c}) \\ - \; & (\mathbf{x} \cdot \mathbf{b})\,(\mathbf{a} \wedge \mathbf{c}) \\ + \; & (\mathbf{x} \cdot \mathbf{c})\,(\mathbf{a} \wedge \mathbf{b}). \end{aligned}$$

The bivectors $(\mathbf{b} \wedge \mathbf{c})$, $(\mathbf{a} \wedge \mathbf{c})$ and $(\mathbf{a} \wedge \mathbf{b})$ are linearly independent and thus $\mathbf{x} \cdot A_{\langle 3 \rangle} = 0$ if and only if

$$\mathbf{x} \cdot \mathbf{a} = 0 \quad \text{and} \quad \mathbf{x} \cdot \mathbf{b} = 0 \quad \text{and} \quad \mathbf{x} \cdot \mathbf{c} = 0.$$

Geometrically this means that $\mathbf{x} \cdot A_{\langle 3 \rangle} = 0$ if and only if $\mathbf{x}$ lies on the intersection of the three planes represented by $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$. Since all planes represented through the IPNS of vectors pass through the origin, the only point all three planes can meet in is the origin. Hence, the only solution for $\mathbf{x}$ to $\mathbf{x} \cdot A_{\langle 3 \rangle} = 0$ is the trivial solution $\mathbf{x} = \mathbf{0} \in \mathbb{R}^3$. Figure 12 illustrates this.

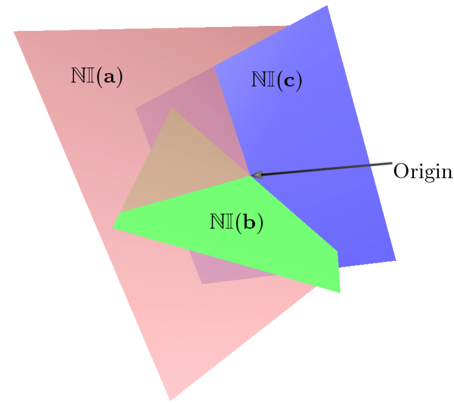


Figure 12: *Intersection of three planes in terms of IPNS.*

12.10. The Meet Operation

We have seen that we can intersect subspaces quite easily, if they are represented through the IPNS of blades. For example, two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ represent two planes in their IPNS. The intersection of these two planes is simply represented by $\mathbf{a} \wedge \mathbf{b}$. (Recall figure 11) The question we would like to answer in this section is: *is there an operation that evaluates the intersection of subspaces represented through the OPNS of blades?*

The short answer is: *yes*. The longer answer will follow now. First we need to remember how the OPNS and IPNS are connected. Given a bivector $\mathbf{a} \wedge \mathbf{b} \in \mathcal{G}(\mathbb{R}^3)$ representing a plane in its OPNS, we can find the respective representation

© The Eurographics Association 2004.

of the plane in term of the IPNS by taking the dual of the bivector. Suppose that $\mathbf{c} \in \mathbb{R}^3$ is given by $\mathbf{c} = (\mathbf{a} \wedge \mathbf{b})^*$, then

$$\mathbb{NI}(\mathbf{c}) = \mathbb{NI}\big((\mathbf{a} \wedge \mathbf{b})^*\big) = \mathbb{NO}\big(\mathbf{a} \wedge \mathbf{b}\big).$$

Using a so far unproven property of the inner product (equation (31)), we can also write

$$\begin{aligned} \mathbf{c} \cdot I &= (\mathbf{a} \wedge \mathbf{b})^* \cdot I \\ &= \big((\mathbf{a} \wedge \mathbf{b}) \cdot I^{-1}\big) \cdot I \\ &= (\mathbf{a} \wedge \mathbf{b})\ (I^{-1} \cdot I) \\ &= \mathbf{a} \wedge \mathbf{b}, \end{aligned}$$

where I is again the unit pseudoscalar of $\mathcal{G}(\mathbb{R}^3)$. That means, in order to transform an IPNS representation into an OPNS representation, we have to multiply with the unit pseudoscalar, a kind of "inverse" dual. In terms of sets,

$$\mathbb{NO}\big(\mathbf{a} \wedge \mathbf{b}\big) = \mathbb{NO}\big(\mathbf{c} \cdot I\big) = \mathbb{NI}\big(\mathbf{c}\big).$$

Now we can see how to express the intersection of two subspaces in terms of the OPNS of two blades. Suppose $\mathbf{a}_1 \wedge \mathbf{a}_2, \mathbf{b}_1 \wedge \mathbf{b}_2 \in \mathcal{G}(\mathbb{R}^3)$ represent two planes in terms of their OPNS. Let their respective normals be denoted by $\mathbf{n}_a = (\mathbf{a}_1 \wedge \mathbf{a}_2)^*$ and $\mathbf{n}_b = (\mathbf{b}_1 \wedge \mathbf{b}_2)^*$. Then in terms of the IPNS the intersection of the two planes is given by $\mathbf{n}_a \wedge \mathbf{n}_b$. As we have seen above, the corresponding expression of the intersection line in terms of the OPNS is simply $(\mathbf{n}_a \wedge \mathbf{n}_b) \cdot I$. Substituting now for $\mathbf{n}_a$ and $\mathbf{n}_b$ gives,

$$\big[(\mathbf{a}_1 \wedge \mathbf{a}_2)^*\ \wedge\ (\mathbf{b}_1 \wedge \mathbf{b}_2)^*\big] \cdot I.$$

This is actually not quite the general intersection operation we were looking for, but it is already pretty good and is thus given its own name: the *regressive* product. Here is the proper definition.

Let $A, B \in \mathcal{G}(\mathbb{R}^n)$ be two arbitrary multivectors and let I denote the unit pseudoscalar of $\mathcal{G}(\mathbb{R}^n)$. The *regressive* product is denoted by $\triangledown$ and is defined as

$$A \triangledown B := \big[A^* \wedge B^*\big] \cdot I. \qquad (24)$$

For the above example this means that given the bivectors $\mathbf{a}_1 \wedge \mathbf{a}_2$ and $\mathbf{b}_1 \wedge \mathbf{b}_2$, representing two planes in their OPNS, the intersection of these planes in the OPNS is given by

$$(\mathbf{a}_1 \wedge \mathbf{a}_2)\ \triangledown\ (\mathbf{b}_1 \wedge \mathbf{b}_2).$$

Unfortunately, their is a problem. Let $\{\mathsf{e}_1, \mathsf{e}_2, \mathsf{e}_3\}$ again denote an orthonormal basis of $\mathbb{R}^3$. Now suppose we wanted to find the intersection of a line represented by e_2 and a plane represented by $\mathsf{e}_2 \wedge \mathsf{e}_3$, through their OPNS. We see immediately that since e_2 is also contained in the bivector $\mathsf{e}_2 \wedge \mathsf{e}_3$, the line is completely contained within the plane and thus their intersection should be the line e_2 itself. However, the regressive product gives

$$\begin{aligned} \mathsf{e}_2 \triangledown (\mathsf{e}_2 \wedge \mathsf{e}_3) &= \big[\mathsf{e}_2{}^* \wedge (\mathsf{e}_2 \wedge \mathsf{e}_3)^*\big] \cdot I \\ &= \big[(\mathsf{e}_1 \wedge \mathsf{e}_3) \wedge \mathsf{e}_1\big] \cdot I \\ &= \big[-(\mathsf{e}_1 \wedge \mathsf{e}_1) \wedge \mathsf{e}_3\big] \cdot I \\ &= 0, \end{aligned}$$

where I is the pseudoscalar of $\mathcal{G}(\mathbb{R}^3)$. The problem is that the line $\mathbb{NO}(\mathsf{e}_2)$ and the plane $\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3)$ live in a 2d-subspace of $\mathbb{R}^3$ spanned by e_2 and e_3. The dimension e_1 is of no importance for the evaluation of their intersection. Suppose now that we work in the subalgebra $\mathcal{G}(\mathbb{R}^2) \subset \mathcal{G}(\mathbb{R}^3)$, where $\{\mathsf{e}_2, \mathsf{e}_3\}$ give an orthonormal basis of $\mathbb{R}^2$. Then the respective unit pseudoscalar is $I = \mathsf{e}_2 \wedge \mathsf{e}_3$ and $I^{-1} = \mathsf{e}_3 \wedge \mathsf{e}_2$, and we obtain

$$\mathsf{e}_2{}^* = -\mathsf{e}_3 \quad \text{and} \quad (\mathsf{e}_2 \wedge \mathsf{e}_3)^* = 1.$$

Hence, the regressive product now gives

$$\begin{aligned} \mathsf{e}_2 \triangledown (\mathsf{e}_2 \wedge \mathsf{e}_3) &= \big[\mathsf{e}_2{}^* \wedge (\mathsf{e}_2 \wedge \mathsf{e}_3)^*\big] \cdot I \\ &= \big[-\mathsf{e}_3 \wedge 1\big] \cdot I \\ &= -\mathsf{e}_3 \cdot I \\ &= \mathsf{e}_2, \end{aligned}$$

which is what we want. This shows that the regressive product works, if we evaluate it in the correct subalgebra. This notion is captured in the general intersection operation: the *meet*.

The meet is basically the regressive product where the pseudoscalar is chosen appropriately. "Appropriately" means that instead of the pseudoscalar of the whole space, the pseudoscalar of the space spanned by the two blades of which the meet is to be evaluated, is used. This introduces the concept of the *join*.

Given two blades $A_{\langle k \rangle}, B_{\langle l \rangle} \in \mathcal{G}(\mathbb{R}^n)$, then their join is a *unit* blade $J \in \mathcal{G}(\mathbb{R}^n)$ such that

$$\mathbb{NO}(J) = \mathbb{NO}(A_{\langle k \rangle})\ \oplus\ \mathbb{NO}(B_{\langle l \rangle}).$$

The join is sometimes also written as an operator, denoted by $\dot{\wedge}$. For example, the join of e_2 and $\mathsf{e}_2 \wedge \mathsf{e}_3$ is simply

$$\mathsf{e}_2\ \dot{\wedge}\ (\mathsf{e}_2 \wedge \mathsf{e}_3) = \mathsf{e}_2 \wedge \mathsf{e}_3,$$

since $\|\mathsf{e}_2 \wedge \mathsf{e}_3\| = 1$ and

$$\mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3) = \mathbb{NO}(\mathsf{e}_2)\ \oplus\ \mathbb{NO}(\mathsf{e}_2 \wedge \mathsf{e}_3).$$

> **Aside.** Note that this definition of the join does not fix the sign of J. This is just as for the unit pseudoscalar I, where we only demanded that its magnitude is unity, but we did not say anything about its sign. We will not discuss this problem further apart from noting that it becomes irrelevant when working in projective spaces.

We can now define the meet in terms of the join. Let $A_{\langle k \rangle}, B_{\langle l \rangle} \in \mathcal{G}(\mathbb{R}^n)$ and let $J = A_{\langle k \rangle} \dot{\wedge} B_{\langle l \rangle}$ be their join. Then the *meet* of $A_{\langle k \rangle}$ and $B_{\langle l \rangle}$ is denoted by $\vee$ and defined as

$$A_{\langle k \rangle} \vee B_{\langle l \rangle} := \Big[\big(A_{\langle k \rangle} \cdot J^{-1}\big)\ \wedge\ \big(B_{\langle l \rangle} \cdot J^{-1}\big)\Big] \cdot J. \qquad (25)$$

In terms of sets this is

$$\mathbb{NO}\big(A_{\langle k \rangle} \vee B_{\langle l \rangle}\big) = \mathbb{NO}\big(A_{\langle k \rangle}\big)\ \cap\ \mathbb{NO}\big(B_{\langle l \rangle}\big).$$

Note that the meet is only defined for blades and it becomes the regressive product, if the join is the pseudoscalar.

© The Eurographics Association 2004.

Equation (25) can also be simplified to read

$$A_{\langle k\rangle} \vee B_{\langle l\rangle} = \left(A_{\langle k\rangle} \cdot J^{-1}\right) \cdot B_{\langle l\rangle}. \qquad (26)$$

12.11. The Geometric Product

We have already seen a lot of features of Geometric Algebra. However, so far, we managed to avoid the actual algebra product, the *geometric product*. The formula most often shown right in the beginning of a Geometric Algebra introduction is

$$\mathbf{a}\,\mathbf{b} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathbf{b}, \qquad (27)$$

where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ are two vectors, and juxtaposition of two vectors, as in $\mathbf{a}\,\mathbf{b}$, denotes the geometric product. **It is important** to note that this equation is **only** valid for vectors, **not** for blades or multivectors in general. It might at first seem strange to add a scalar $(\mathbf{a} \cdot \mathbf{b})$ and a bivector $(\mathbf{a} \wedge \mathbf{b})$, but they are just different elements of the Geometric Algebra. This is just like for complex numbers, where a real and an imaginary part are added.

A somewhat more general form of equation (27) is

$$\mathbf{a}\,B_{\langle l\rangle} = \mathbf{a} \cdot B_{\langle l\rangle} + \mathbf{a} \wedge B_{\langle l\rangle}, \qquad (28)$$

with $B_{\langle l\rangle} \in \mathcal{G}(\mathbb{R}^n)$ and $l > 0$. For $l = 0$, ie $B_{\langle l\rangle}$ a scalar, we have

$$\mathbf{a}\,B_{\langle 0\rangle} = \mathbf{a} \wedge B_{\langle 0\rangle}.$$

In general we always have for a scalar $\alpha \in \mathbb{R}$ and a multivector $A \in \mathcal{G}(\mathbb{R}^n)$ that their inner product is **identically** zero,

$$\alpha \cdot A \equiv 0.$$

This turns out to be a necessary definition to keep the system of operations in Geometric Algebra self-consistent.

The geometric product is associative and distributive but in general not commutative. That is, for multivectors $A, B, C \in \mathcal{G}(\mathbb{R}^n)$

$$\begin{aligned} (A\,B)\,C &= A\,(B\,C), \\ A\,(B+C) &= (A\,B) + (A\,C), \\ (B+C)\,A &= (B\,A) + (C\,A), \\ A\,B &\neq B\,A, \quad \text{in general.} \end{aligned} \qquad (29)$$

Two further useful properties of the geometric product are the following. Given two blades $A_{\langle k\rangle}, B_{\langle l\rangle} \in \mathcal{G}(\mathbb{R}^n)$, then

$$\mathbb{NO}(A_{\langle k\rangle}) \cap \mathbb{NO}(B_{\langle l\rangle}) = \emptyset \iff A_{\langle k\rangle}\,B_{\langle l\rangle} = A_{\langle k\rangle} \wedge B_{\langle l\rangle}, \qquad (30)$$

and

$$\left.\begin{array}{rl} & \mathbb{NO}(A_{\langle k\rangle}) \subseteq \mathbb{NO}(B_{\langle l\rangle}) \\ \text{or} & \mathbb{NO}(B_{\langle l\rangle}) \subseteq \mathbb{NO}(A_{\langle k\rangle}) \end{array}\right\} \iff A_{\langle k\rangle}\,B_{\langle l\rangle} = A_{\langle k\rangle} \cdot B_{\langle l\rangle}. \qquad (31)$$

Equation (31) for example implies that for some vector $\mathbf{a} \in \mathbb{R}^n$,

$$\mathbf{a}^* \cdot I = (\mathbf{a} \cdot I^{-1}) \cdot I = (\mathbf{a}\,I^{-1})\,I = \mathbf{a}\,(I^{-1}\,I) = \mathbf{a},$$

where I is the pseudoscalar of $\mathcal{G}(\mathbb{R}^n)$.

12.12. Reflection

So far we have seen how to construct linear subspaces using the outer product and to subtract linear subspaces from one another using the inner product. We also now know how to intersect linear subspaces using the meet and how to form their union with the join. We now would like to operate on subspaces while keeping their dimensionality unchanged. For example, rotating a line results in another line, not in a point or a plane. An operation on a blade that does not change its grade, is called *grade preserving*.

Without much further ado, we will look at such a grade preserving operation. Let $\mathbf{a}, \mathbf{n} \in \mathbb{R}^n$ denote two vectors, whereby $\|\mathbf{n}\| = 1$. Also write $\mathbf{a} = \mathbf{a}^{\|} + \mathbf{a}^{\perp}$, where $\mathbf{a}^{\|}$ is the component of $\mathbf{a}$ parallel and $\mathbf{a}^{\perp}$ the component perpendicular to $\mathbf{n}$. Note that the following calculation is valid for all dimensions $n \geq 2$ of the vector space.

$$\begin{aligned} \mathbf{n}\,\mathbf{a}\,\mathbf{n} &= (\mathbf{n}\,\mathbf{a})\,\mathbf{n} \\ &= (\mathbf{n} \cdot \mathbf{a} + \mathbf{n} \wedge \mathbf{a})\,\mathbf{n} \\ &= (\mathbf{n} \cdot \mathbf{a})\,\mathbf{n} + (\mathbf{n} \wedge \mathbf{a}) \cdot \mathbf{n} + \underbrace{(\mathbf{n} \wedge \mathbf{a}) \wedge \mathbf{n}}_{=0}. \end{aligned}$$

So far we only applied the associativity of the geometric product and equation (27). Using equation (10) we see that

$$(\mathbf{n} \wedge \mathbf{a}) \cdot \mathbf{n} = (\mathbf{a} \cdot \mathbf{n})\,\mathbf{n} - (\mathbf{n} \cdot \mathbf{n})\,\mathbf{a}.$$

Hence,

$$\begin{aligned} \mathbf{n}\,\mathbf{a}\,\mathbf{n} &= (\mathbf{n} \cdot \mathbf{a})\,\mathbf{n} + (\mathbf{a} \cdot \mathbf{n})\,\mathbf{n} - \underbrace{(\mathbf{n} \cdot \mathbf{n})}_{=1}\,\mathbf{a} \\ &= 2\,(\mathbf{n} \cdot \mathbf{a})\,\mathbf{n} - \mathbf{a}. \end{aligned}$$

Clearly we have $\mathbf{n} \cdot \mathbf{a} = \mathbf{n} \cdot \mathbf{a}^{\|}$, and since $\mathbf{a}^{\|}$ is the component of $\mathbf{a}$ parallel to $\mathbf{n}$, we can also write $\mathbf{a}^{\|} = \|\mathbf{a}^{\|}\|\,\mathbf{n}$. Thus,

$$\begin{aligned} \mathbf{n}\,\mathbf{a}\,\mathbf{n} &= 2\,(\mathbf{n} \cdot \mathbf{a}^{\|})\,\mathbf{n} - \mathbf{a} \\ &= 2\,\|\mathbf{a}^{\|}\|\,\mathbf{n} - \mathbf{a} \\ &= 2\,\mathbf{a}^{\|} - \mathbf{a}^{\|} - \mathbf{a}^{\perp} \\ &= \mathbf{a}^{\|} - \mathbf{a}^{\perp}. \end{aligned}$$

That is, the component of $\mathbf{a}$ perpendicular to $\mathbf{n}$ has been negated, while the parallel component $\mathbf{a}^{\|}$ remained unchanged. Geometrically this is a *reflection* of the vector $\mathbf{a}$ on the line through the origin with direction $\mathbf{n}$. This is illustrated in figure 13.

The really nice thing about this reflection operation is that it can be applied to any blade. For example, given a plane as bivector $A_{\langle 2\rangle} \in \mathcal{G}(\mathbb{R}^3)$, it can be reflected in a normalized vector $\mathbf{n} \in \mathbb{R}^3$ simply by evaluating $\mathbf{n}\,A_{\langle 2\rangle}\,\mathbf{n}$. This is shown in figure 14.

Let $A_{\langle 2\rangle} = \mathbf{a}_1 \wedge \mathbf{a}_2$ with $\mathbf{a}_1, \mathbf{a}_2 \in \mathbb{R}^3$, then it may in fact be shown that

$$\mathbf{n}\,A_{\langle 2\rangle}\,\mathbf{n} = \left(\mathbf{n}\,\mathbf{a}_1\,\mathbf{n}\right) \wedge \left(\mathbf{n}\,\mathbf{a}_2\,\mathbf{n}\right).$$

That is, the reflection of the outer product of two vectors, is

© The Eurographics Association 2004.

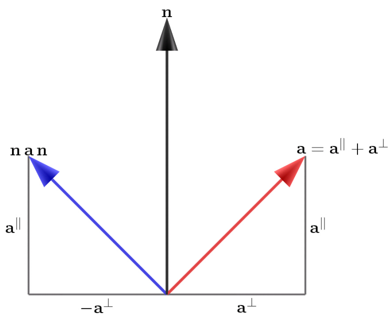


Figure 13: *Reflection of vector* **a** *on vector* **n***.*

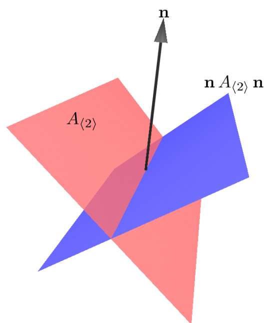


Figure 14: *Reflection of bivector* $A_{\langle 2 \rangle}$ *on vector* **n***.*

the outer product of the separately reflected vectors. By the way, this property is also called *outer-morphism*, not to be confused with auto-morphism.

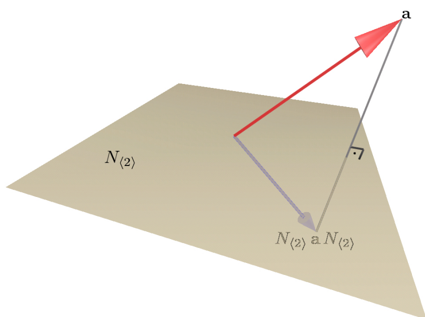


Figure 15: *Reflection of vector* **a** *on bivector* $N_{\langle 2 \rangle}$*.*

A blade may also be reflected on another blade. Figure 15 shows the reflection of a vector $\mathbf{a} \in \mathbb{R}^3$ on a bivector $N_{\langle 2 \rangle} \in \mathcal{G}(\mathbb{R}^3)$ by evaluating $N_{\langle 2 \rangle}\mathbf{a}N_{\langle 2 \rangle}$. This operation again results in

$$N_{\langle 2 \rangle}\mathbf{a}N_{\langle 2 \rangle} = \mathbf{a}^{\|} - \mathbf{a}^{\perp},$$

where $\mathbf{a}^{\|}$ and $\mathbf{a}^{\perp}$ are this time the parallel and perpendicular components of **a** with respect to $N_{\langle 2 \rangle}$.

The reflection operation is in fact the only operation we will ever be using in Geometric Algebra. Any other operation needed will be obtained by combining a number of different reflections. In Euclidean space this confines us in fact to reflections and rotations about axes that pass through the origin, as will be shown in the next section. To extend the set of available operations Euclidean space will have to be embedded in other spaces, which will be discussed later on.

12.13. Rotation

Reflections with respect to a normalized vector **n** are always reflections on a line with direction **n**, passing through the origin. It may be shown that two consecutive reflections on different, normalized vectors **n** and **m** are equivalent to a rotation of twice the angle between **n** and **m**.

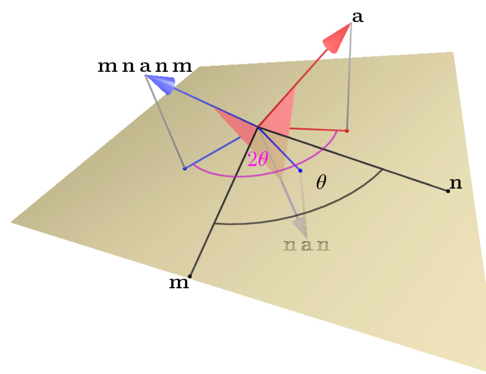


Figure 16: *Rotation of vector* **a** *by consecutive reflections of* **a** *on* **n** *and* **m***.*

Figure 16 shows such a setup in 3d-Euclidean space. The normalized vectors $\mathbf{n}, \mathbf{m} \in \mathbb{R}^3$ enclose an angle $\angle(\mathbf{n}, \mathbf{m}) = \theta$ and define a rotation plane through their outer product $\mathbf{n} \wedge \mathbf{m}$. Reflecting a vector $\mathbf{a} \in \mathbb{R}^3$ first on **n** and then on **m**, rotates the component of **a** that lies in the rotation plane by 2θ. The component of **a** perpendicular to the rotation plane remains unchanged.

The rotation of vector **a** in the plane $\mathbf{n} \wedge \mathbf{m}$ by an angle 2θ may then be written as

$$\mathbf{b} = \mathbf{m}\,\mathbf{n}\,\mathbf{a}\,\mathbf{n}\,\mathbf{m}. \tag{32}$$

From the definition of the geometric product we find that

$$\mathbf{m}\,\mathbf{n} = \mathbf{m} \cdot \mathbf{n} + \mathbf{m} \wedge \mathbf{n}$$

and also

$$\mathbf{n}\,\mathbf{m} = \mathbf{n} \cdot \mathbf{m} + \mathbf{n} \wedge \mathbf{m} = \mathbf{n} \cdot \mathbf{m} + (\mathbf{m} \wedge \mathbf{n})\tilde{}.$$

© The Eurographics Association 2004.

Since the reverse of a scalar is still the same scalar it follows

$$\mathbf{m}\,\mathbf{n} = (\mathbf{n}\,\mathbf{m})\tilde{}.$$

Equation (32) may therefore also be written more succinctly as

$$\mathbf{b} = R\,\mathbf{a}\,\widetilde{R}, \quad \text{with } R := \mathbf{m}\,\mathbf{n}. \tag{33}$$

Since applying R as above has the effect of a rotation, R is called a *rotor*. Note that a rotor has to satisfy the equation

$$R\widetilde{R} = 1,$$

because it would otherwise also scale the entity it is applied to. We can actually recognize this as something familiar, by expanding R as

$$\begin{aligned} R &= \mathbf{m}\,\mathbf{n} \\ &= \mathbf{m}\cdot\mathbf{n} + \mathbf{m}\wedge\mathbf{n} \\ &= \cos\theta + \sin\theta\, U_{\langle 2\rangle}, \end{aligned} \tag{34}$$

where $\theta = \angle(\mathbf{m},\mathbf{n})$ and $U_{\langle 2\rangle}$ is the normalized version of $\mathbf{m}\wedge\mathbf{n}$, ie

$$U_{\langle 2\rangle} := \frac{\mathbf{m}\wedge\mathbf{n}}{\|\mathbf{m}\wedge\mathbf{n}\|}.$$

From equation (15) we know that

$$U_{\langle 2\rangle}\cdot U_{\langle 2\rangle} = (-1)^{2(2-1)/2}\,\|U_{\langle 2\rangle}\|^2 = -1.$$

Since $U_{\langle 2\rangle}$ squares to -1, the expression for R in equation (34) is similar to that of a complex number z in the polar representation

$$z = r\,(\cos\theta + \mathrm{i}\,\sin\theta),$$

where $\mathrm{i} = \sqrt{-1}$ represents the imaginary unit and $r \in \mathbb{R}$ is the radius. For complex numbers it is well known that the above expression can also be written as

$$z = r\,\exp(\mathrm{i}\,\theta).$$

The definition of the exponential function can be extended to Geometric Algebra, and it can be shown that the Taylor series of $\exp(\theta\, U_{\langle 2\rangle})$ does indeed converge to

$$\exp\big(\theta\, U_{\langle 2\rangle}\big) = \cos\theta + \sin\theta\, U_{\langle 2\rangle} = R. \tag{35}$$

It turns out that $R = \exp(\theta U_{\langle 2\rangle})$ actually represents a clockwise rotation by an angle 2θ in the plane $U_{\langle 2\rangle}$. The term "clockwise" only makes really sense in 3d-space. Here it means clockwise relative to the rotation axis given by $U^*_{\langle 2\rangle}$. If we want to represent a mathematically positive, ie *anti*-clockwise, rotation about an angle θ, within the plane $U_{\langle 2\rangle}$, we need to write the corresponding rotor as

$$R = \exp\big(-\tfrac{\theta}{2}\, U_{\langle 2\rangle}\big). \tag{36}$$

Just as for reflections, a rotor represents a rotation in any dimension. A rotor can also rotate any blade. That is, with the same rotor we can rotate vectors, bivectors, etc. It turns out that for a rotor we also have an outer-morphism. This means that given a blade $A_{\langle k\rangle} = \bigwedge_{i=1}^{k} \mathbf{a}_i$, with $\{\mathbf{a}_i\} \subset \mathbb{R}^n$, and a rotor R, we can expand the expression $RA_{\langle k\rangle}\widetilde{R}$ as

$$RA_{\langle k\rangle}\widetilde{R} = \big(R\,\mathbf{a}_1\widetilde{R}\big)\wedge\big(R\,\mathbf{a}_2\widetilde{R}\big)\wedge\ldots\wedge\big(R\,\mathbf{a}_k\widetilde{R}\big). \tag{37}$$

Hence, the rotation of the outer product of a number of vectors is the same as the outer product of a number of rotated vectors.

13. Geometry

In the previous sections we first talked about Geometric Algebra and how elements of that algebra are taken to represent geometric entities. We also saw how we can operate on such entities in order to reflect or rotate them. In this section we would like to return to the geometric interpretation of the algebra.

Although we will talk in the following about spaces which embed Euclidean space in some way, the basic meaning of blades as linear subspaces and the reflection operator remain the same within these spaces. However, their effect on the embedded Euclidean space, or rather their interpretation in terms of the embedded Euclidean space may change quite substantially.

We will denote the homogeneous embedding of Euclidean space $\mathbb{E}^n$ by $\mathbb{PE}^n$. $\mathbb{PE}^n$ is also called a projective space. The properties of $\mathbb{PE}^n$ basically derive from the way Euclidean space is embedded in it. The projective space $\mathbb{PE}^n$ will be represented by $\mathbb{R}^{n+1}\setminus\mathbf{0}$, ie a $(n+1)$-dimensional vector space without the origin. The canonical (orthonormal) basis of $\mathbb{R}^{n+1}$ will be denoted by $\{\mathsf{e}_1,\ldots,\mathsf{e}_n,\mathsf{e}_{n+1}\}$. The basis vector e_{n+1} is also called the *homogeneous* component or dimension.

13.1. The Setup

The transformation operator from Euclidean to the corresponding projective space will be denoted by $\mathcal{P}$ and its inverse by $\mathcal{P}^{-1}$. The operator $\mathcal{P}$ is defined as

$$\mathcal{P}\,:\,\mathbf{x}\in\mathbb{E}^n \;\mapsto\; \mathbf{x}+\mathsf{e}_{n+1}\in\mathbb{PE}^n. \tag{38}$$

That is, Euclidean space is embedded as a particular hyperplane $\mathcal{P}(\mathbb{E}^n)$ in projective space. A vector in $\mathbb{PE}^n$ will also be called a *homogeneous* vector. Note that the origin of Euclidean space becomes e_{n+1} in projective space. This means that the origin of Euclidean space, as represented in projective space is not a special point any more. For example, while the scalar product of a vector with the origin in Euclidean space is always identically zero, this is not necessarily the case in projective space.

Figure 17 illustrates the embedding of Euclidean vectors in projective space for the case of $\mathbb{E}^2$. A vector $\mathbf{a}\in\mathbb{E}^2$ from Euclidean space is embedded in projective space $\mathbb{PE}^2$ by adding the homogeneous dimension e_3. The homogeneous representation of $\mathbf{a}$ in $\mathbb{PE}^2$ is then denoted by $\mathbf{A} = \mathcal{P}(\mathbf{a})$.

© The Eurographics Association 2004.

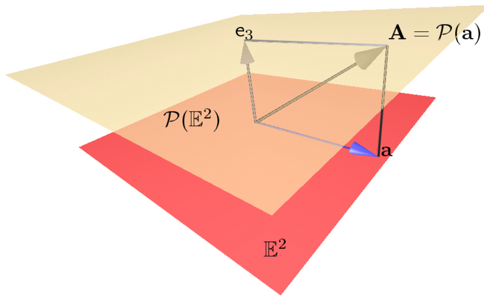


Figure 17: *Embedding of Euclidean vector* $\mathbf{a} \in \mathbb{E}^2$ *in projective space* $\mathbb{PE}^2$ *as* $\mathbf{A} = \mathcal{P}(\mathbf{a})$.

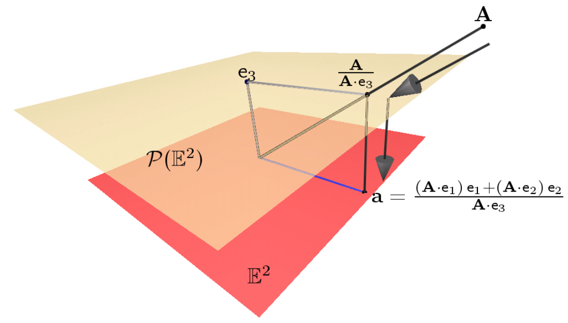


Figure 18: *Projections of a homogeneous vector* $\mathbf{A} \in \mathbb{PE}^2$ *into the corresponding Euclidean space* $\mathbb{E}^2$ *as* $\mathbf{a} = \mathcal{P}^{-1}(\mathbf{A})$.

Although Euclidean vectors are mapped to a hyperplane in projective space, a general homogeneous vector may lie anywhere in $\mathbb{PE}^n \equiv \mathbb{R}^{n+1} \setminus \mathbf{0}$. Therefore, the question is how homogeneous vectors that do not lie on $\mathcal{P}(\mathbb{E}^n)$ are projected back to $\mathbb{E}^n$. This projection is in fact the key to the power of the homogeneous representation.

The transformation from $\mathbb{PE}^n$ to $\mathbb{E}^n$ is denoted by $\mathcal{P}^{-1}$ and is defined as

$$\mathcal{P}^{-1} : \mathbf{A} \in \mathbb{PE}^n \mapsto \frac{1}{\mathbf{A} \cdot \mathsf{e}_{n+1}} \sum_{i=1}^{n} \left(\mathbf{A} \cdot \mathsf{e}_i\right) \mathsf{e}_i \in \mathbb{E}^n. \quad (39)$$

Clearly, this transformation is only valid for homogeneous vectors that have a non-zero homogeneous component. Those homogeneous vectors that do have a zero homogeneous component would map to infinity and are accordingly called points at infinity or *direction* vectors.

Using the transformation $\mathcal{P}^{-1}$ the whole of $\mathbb{PE}^n$ apart from the plane $\mathsf{e}_{n+1} = 0$ is mapped to $\mathbb{E}^n$. What does this mean for a particular homogeneous vector? Well, the homogeneous vector is first scaled such that its homogeneous component is unity, and then its first n components are taken as the n components of the corresponding Euclidean vector. This is illustrated in figure 18.

The effect of $\mathcal{P}^{-1}$ is that the overall scale of a homogeneous vector in projective space is of no importance. For example, given a vector $\mathbf{a} \in \mathbb{E}^n$ and a scale $\alpha \in \mathbb{R} \setminus 0$, then

$$\mathcal{P}^{-1}\Big(\alpha \mathcal{P}(\mathbf{a})\Big) = \mathbf{a}.$$

Hence, the name "projective space": homogeneous vectors are projected onto the hyperplane $\mathcal{P}(\mathbb{E}^n)$ before they are "orthographically" projected into $\mathbb{E}^n$. The hyperplane $\mathcal{P}(\mathbb{E}^n)$ is also called the *affine* plane. Note that affine transformations are in fact just those that when applied to a point on $\mathcal{P}(\mathbb{E}^n)$ leave the point on that plane. Projective transformations on the other may move points through the whole space $\mathbb{PE}^n$.

13.2. Geometric Algebra on $\mathbb{PE}^n$

Recall that elements of Geometric Algebra are given geometric meaning by looking at their OPNS or IPNS, the outer or inner product null space. When we write down a blade, its OPNS always represents a linear subspace. For example, a bivector in $\mathbb{PE}^2$ is a two dimensional subspace, since we represent $\mathbb{PE}^2$ by $\mathbb{R}^{2+1}$. However, we are not really interested in what this bivector represents in $\mathbb{PE}^2$. We would like to know what it represents in the corresponding $\mathbb{E}^2$. How do we do that? Well, we need to be more precise about which null space we are actually interested in.

Given a bivector $A_{\langle 2 \rangle} \in \mathcal{G}(\mathbb{PE}^2)$, we are only interested in those vectors in $\mathbb{PE}^2$ that lie in one of its null spaces, which we can also map back to Euclidean space. The other way around: we ask which vectors in $\mathbb{E}^2$ when transformed to $\mathbb{PE}^2$ lie in the null space of $A_{\langle 2 \rangle}$. We therefore introduce the concept of the *Euclidean* outer and inner product null space, denoted by $\mathbb{NO}_E$ and $\mathbb{NI}_E$, respectively. For $\mathcal{G}(\mathbb{PE}^n)$ they are defined as follows.

$$\begin{aligned} &\mathbb{NO}_E(A_{\langle k \rangle} \in \mathcal{G}(\mathbb{PE}^n)) \\ &\quad := \left\{ \mathbf{a} \in \mathbb{E}^n \,:\, \mathcal{P}(\mathbf{a}) \wedge A_{\langle k \rangle} = 0 \in \mathcal{G}(\mathbb{PE}^n) \right\}, \\ \text{and} \quad &\mathbb{NI}_E(A_{\langle k \rangle} \in \mathcal{G}(\mathbb{PE}^n)) \\ &\quad := \left\{ \mathbf{a} \in \mathbb{E}^n \,:\, \mathcal{P}(\mathbf{a}) \cdot A_{\langle k \rangle} = 0 \in \mathcal{G}(\mathbb{PE}^n) \right\}. \end{aligned} \quad (40)$$

13.3. The Euclidean OPNS

So how can we evaluate the Euclidean IPNS or OPNS of a blade in projective space? Consider, for example, a vector $\mathbf{a} \in \mathbb{E}^n$ with homogeneous representation $\mathbf{A} = \mathcal{P}(\mathbf{a}) \in \mathbb{PE}^n$. The OPNS of $\mathbf{A}$ is simply given by

$$\mathbb{NO}(\mathbf{A}) = \left\{ \alpha\mathbf{A} \in \mathbb{PE}^n \,:\, \alpha \in \mathbb{R} \setminus 0 \right\},$$

© The Eurographics Association 2004.

a projective line in $\mathbb{PE}^n$. The factor α must not be zero since the origin of $\mathbb{R}^{n+1}$ is not an element of $\mathbb{PE}^n$. Since all elements of $\mathbb{NO}(\mathbf{A})$ can be mapped to $\mathbb{E}^n$ by $\mathcal{P}^{-1}$, we find that

$$\begin{aligned}
\mathbb{NO}_E(\mathbf{A}) &= \mathcal{P}^{-1}\Big(\mathbb{NO}(\mathbf{A})\Big) \\
&= \Big\{ \tfrac{1}{(\alpha A)\cdot \mathsf{e}_{n+1}} \textstyle\sum_{i=1}^{n} \Big((\alpha A)\cdot \mathsf{e}_i\Big)\,\mathsf{e}_i \;:\; \alpha \in \mathbb{R}\setminus 0 \Big\} \\
&= \Big\{ \mathcal{P}^{-1}(\mathbf{A}) \;:\; \alpha \in \mathbb{R}\setminus 0 \Big\} \\
&= \mathbf{a}.
\end{aligned}$$

This shows that even though the OPNS of $\mathbf{A}$ is a (projective) line in $\mathbb{PE}^n$, the Euclidean OPNS of $\mathbf{A}$ is only the vector $\mathbf{a} \in \mathbb{E}^n$. This is great, since it enables us to represent a zero-dimensional object, ie a point, in $\mathbb{E}^n$ by a line in $\mathbb{PE}^n$.

An example of this has already been shown for the case of $\mathbb{E}^2$ in figure 18. All points in $\mathbb{PE}^2$ along the line from, but excluding, the origin of $\mathbb{PE}^2$ to the homogeneous vector $\mathbf{A}$, represent the same point $\mathbf{a}$ in $\mathbb{E}^2$.

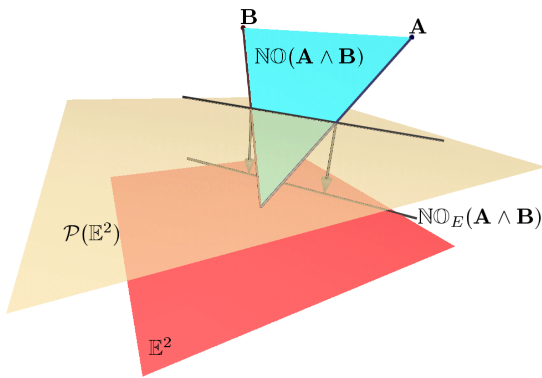


Figure 19: *Representation of line in* $\mathbb{E}^2$ *through bivector in* $\mathcal{G}(\mathbb{PE}^2)$.

Figure 19 illustrates the OPNS and Euclidean OPNS of a bivector in $\mathbb{PE}^2$. The OPNS of the outer product of two homogeneous vectors $\mathbf{A}, \mathbf{B} \in \mathbb{PE}^2$ is a plane in $\mathbb{PE}^2$. The orthographic projection of the intersection of $\mathbb{NO}(\mathbf{A}\wedge\mathbf{B})$ with the plane $\mathcal{P}(\mathbb{E}^2)$, then gives the Euclidean OPNS of $\mathbf{A}\wedge\mathbf{B}$: a line in $\mathbb{E}^2$. Note that this line does not pass through the origin. This shows one of the advantages of working in $\mathcal{G}(\mathbb{PE}^2)$ instead of $\mathcal{G}(\mathbb{E}^2)$. In $\mathcal{G}(\mathbb{E}^2)$ we could only represent lines through the origin, whereas in $\mathcal{G}(\mathbb{PE}^2)$ we can represent arbitrary lines in the corresponding $\mathbb{E}^2$.

Without going into any more detail, it may be shown that the Euclidean OPNS of the outer product of three homogeneous vectors in $\mathcal{G}(\mathbb{PE}^3)$ represents a plane in $\mathbb{E}^3$. That is, given vectors $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{E}^3$ and $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{PE}^3$ with

$$\mathbf{A} = \mathcal{P}(\mathbf{a}) \quad \text{and} \quad \mathbf{B} = \mathcal{P}(\mathbf{b}) \quad \text{and} \quad \mathbf{C} = \mathcal{P}(\mathbf{c}),$$

it may be shown that $\mathbb{NO}_E(\mathbf{A}\wedge\mathbf{B}\wedge\mathbf{C})$ is a plane in $\mathbb{E}^3$ which passes through the points $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$. To summarize, we have

$\mathbb{NO}_E(\mathbf{A})$	**Point a**
$\mathbb{NO}_E(\mathbf{A}\wedge\mathbf{B})$	**Line** through **a** and **b**
$\mathbb{NO}_E(\mathbf{A}\wedge\mathbf{B}\wedge\mathbf{C})$	**Plane** through **a**, **b** and **c**

13.4. The Euclidean IPNS

We can also consider the Euclidean IPNS of blades of $\mathcal{G}(\mathbb{E}^3)$. We will do this in some detail for a homogeneous vector. Let $\mathbf{A} \in \mathbb{PE}^3$ be given by

$$\mathbf{A} = \hat{\mathbf{a}} - \alpha\,\mathsf{e}_o,$$

where $\hat{\mathbf{a}} \in \mathbb{E}^3$ and $\|\hat{\mathbf{a}}\| = 1$. Furthermore, $\alpha \in \mathbb{R}$ and e_o denotes the homogeneous dimension e_{3+1}, in order to emphasize its meaning as the vector in $\mathbb{PE}^3$ representing the origin of $\mathbb{E}^n$. Let us now try to evaluate the Euclidean IPNS of $\mathbf{A}$. That is, we are looking for all those vectors $\mathbf{x} \in \mathbb{E}^3$ that satisfy $\mathbf{A}\cdot\mathcal{P}(\mathbf{x}) = 0$.

$$\begin{aligned}
\mathbf{A}\cdot\mathcal{P}(\mathbf{x}) = 0 \quad &\Longleftrightarrow \quad (\hat{\mathbf{a}} - \alpha\,\mathsf{e}_o)\cdot(\mathbf{x} + \mathsf{e}_o) = 0 \\
&\Longleftrightarrow \quad \hat{\mathbf{a}}\cdot\mathbf{x} - \alpha = 0 \\
&\Longleftrightarrow \quad \hat{\mathbf{a}}\cdot\mathbf{x}^{\parallel} - \alpha = 0 \\
&\Longleftrightarrow \quad \mathbf{x}^{\parallel} = \alpha\,\hat{\mathbf{a}},
\end{aligned}$$

where $\mathbf{x}^{\parallel}$ is the component of $\mathbf{x}$ parallel to $\hat{\mathbf{a}}$. If we write the component of $\mathbf{x}$ perpendicular to $\hat{\mathbf{a}}$ as $\mathbf{x}^{\perp}$, then it follows that any vector $\mathbf{x} \in \mathbb{E}^3$ of the form

$$\mathbf{x} = \alpha\,\hat{\mathbf{a}} + \mathbf{x}^{\perp},$$

lies in the Euclidean IPNS of $\mathbf{A}$. Hence, $\mathbf{A}$ represents a plane with normal $\hat{\mathbf{a}}$ and distance α from the origin in $\mathbb{E}^3$. As for Euclidean space it may also be shown that for homogeneous vectors $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{PE}^3$, we have

Plane:	$\mathbb{NI}_E(\mathbf{A})$
Line:	$\mathbb{NI}_E(\mathbf{A}\wedge\mathbf{B}) = \mathbb{NI}_E(A) \cap \mathbb{NI}_E(B)$
Point:	$\mathbb{NI}_E(\mathbf{A}\wedge\mathbf{B}\wedge\mathbf{C}) = \mathbb{NI}_E(A) \cap \mathbb{NI}_E(B) \cap \mathbb{NI}_E(C)$

In conformal space the Euclidean OPNS and IPNS of blades are non-linear objects like circles and spheres, since the embedding of Euclidean space in conformal space is a non-linear one. Please see [PH03] for more details on this embedding.

13.5. The Pinhole Camera Model

The Geometric Algebra of projective space is very useful to represent projections in the pinhole camera model. Figure 20 show such a setup. Homogeneous vectors $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4 \in \mathbb{PE}^3$ form a basis of $\mathbb{PE}^3$. The homogeneous vector $\mathbf{A}_4$ represents the optical center of the pinhole camera, while $P = \mathbf{A}_1\wedge\mathbf{A}_2\wedge\mathbf{A}_3$ represents the image plane. In order to project a homogeneous vector $\mathbf{X}$ onto the image plane, we simply have to intersect the image plane P with the line L connecting $\mathbf{X}$ with the optical center $\mathbf{A}_4$, ie $L = \mathbf{A}_4\wedge\mathbf{X}$. We can do this with the meet operation,

$$\mathbf{Y} = \mathbf{L}\vee\mathbf{P} = (\mathbf{A}_4\wedge\mathbf{X}) \;\vee\; (\mathbf{A}_1\wedge\mathbf{A}_2\wedge\mathbf{A}_3).$$

© The Eurographics Association 2004.

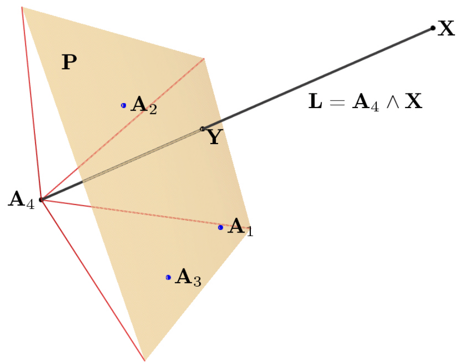


Figure 20: *Model of a pinhole camera in* $\mathbb{PE}^3$.

Since the join of L and P is the whole space $\mathbb{PE}^3$, we can also use the regressive product instead of the meet, which simplifies the evaluation of the meet.

By using such simple geometric constructions, which can be readily translated into Geometric Algebra equations, also the relations between two, three or more cameras can be analyzed. This then leads, for example, to the fundamental matrix and the trifocal tensor as was shown in [LL98, PL98, PL01, Per00].

13.6. Reflections in Projective Space

By going from Euclidean to projective space, an additional dimension, the homogeneous dimension, is introduced. We may therefore wonder what effect this has when using the reflection operator as introduced earlier. First of all consider a vector $\mathbf{a} \in \mathbb{E}^2$ and its homogeneous representation

$$\mathbf{A} = \mathcal{P}(\mathbf{a}) = \mathbf{a} + e_o \in \mathbb{PE}^2,$$

where e_o denotes again the homogeneous dimension $e_3 \in \mathbb{PE}^2$. A reflection about e_o gives

$$\begin{aligned} e_o \mathbf{A} e_o &= e_o \mathbf{a} e_o + e_o e_o e_o \\ &= -\mathbf{a} e_o e_o + e_o \\ &= -\mathbf{a} + e_o, \end{aligned}$$

where we used the fact that e_o is perpendicular to all vectors in $\mathbb{E}^2$. Therefore,

$$e_o \mathbf{a} = e_o \wedge \mathbf{a} = -\mathbf{a} \wedge e_o = -\mathbf{a} e_o.$$

We thus have

$$\mathcal{P}^{-1}\Big(e_o \mathcal{P}(\mathbf{a})\, e_o\Big) = -\mathbf{a},$$

which shows that a reflection of $\mathbf{A}$ about e_o represents a reflection about the origin of $\mathbf{a}$.

Next consider a vector $\mathbf{n} \in \mathbb{E}^2$, with $\|\mathbf{n}\| = 1$. Although this is mathematically not quite rigorous, we can regard the vector $\mathbf{n}$ also as a direction vector of $\mathbb{PE}^2$, since it has no e_o component. If we take $\mathbf{A}$ as given above, we can ask what a reflection of a homogeneous vector $\mathbf{A}$ on a direction vector $\mathbf{n}$ in $\mathbb{PE}^2$ means.

$$\begin{aligned} \mathbf{n}\mathbf{A}\mathbf{n} &= \mathbf{n}(\mathbf{a} + e_o)\mathbf{n} \\ &= \mathbf{n}\mathbf{a}\mathbf{n} + \mathbf{n} e_0 \mathbf{n} \\ &= \mathbf{n}\mathbf{a}\mathbf{n} - e_o \mathbf{n}^2 \\ &= \mathbf{n}\mathbf{a}\mathbf{n} - e_o. \end{aligned}$$

For convenience, let us at this point introduce an operator $\mathcal{A}$ that projects homogeneous vectors in $\mathbb{PE}^n$ onto the affine plane $\mathcal{P}(\mathbb{E}^n) \subset \mathbb{PE}^n$. The operator is therefore defined as

$$\mathcal{A} : \mathbf{A} \in \mathbb{PE}^n \mapsto \frac{\mathbf{A}}{\mathbf{A} \cdot e_o} \in \mathbb{PE}^n, \tag{41}$$

where e_o is again the homogeneous dimension. We may also say that $\mathcal{A}$ transforms homogeneous vectors to affine vectors. This operator is also useful, since homogeneous vectors on $\mathcal{P}(\mathbb{E}^n)$ can be immediately identified with their corresponding Euclidean vectors in $\mathbb{E}^n$. For our reflection example from above we find,

$$\begin{aligned} \mathcal{A}(\mathbf{n}\mathbf{A}\mathbf{n}) &= -\mathbf{n}\mathbf{a}\mathbf{n} + e_o \\ &= -(\mathbf{a}^{\|} + \mathbf{a}^{\perp}) + e_o \\ &= \mathbf{a}^{\perp} - \mathbf{a}^{\|} + e_o, \end{aligned}$$

where $\mathbf{a}^{\|}$ and $\mathbf{a}^{\perp}$ are the orthogonal and parallel components of $\mathbf{a}$ with respect to $\mathbf{n}$, respectively. This shows that the component of the homogeneous vector $\mathbf{A}$ that is *parallel* to the reflection direction $\mathbf{n}$, is reflected and not the part perpendicular to it. Figure 21 shows this setup.

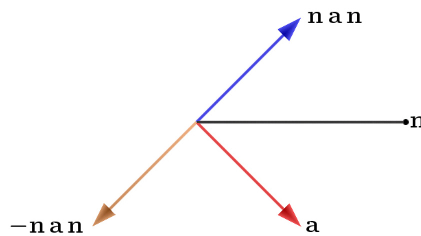


Figure 21: *Effect in* $\mathbb{E}^2$ *of reflection of homogeneous vector on direction vector in* $\mathbb{PE}^2$.

This is not really what we wanted to achieve. However, we can remedy the situation by reflecting $\mathbf{n}\mathbf{A}\mathbf{n}$ again through the origin. That is, in order to reflect a homogeneous vector on a line with direction $\mathbf{n}$, we have to use as operator $(\mathbf{n} e_0)$ instead of $\mathbf{n}$.

$$\begin{aligned} (\mathbf{n} e_0)\mathbf{A}(e_0 \mathbf{n}) &= \mathbf{n}(-\mathbf{a} + e_0)\mathbf{n} \\ &= -\mathbf{n}\mathbf{a}\mathbf{n} + \mathbf{n} e_o \mathbf{n} \\ &= -\mathbf{n}\mathbf{a}\mathbf{n} - e_o, \end{aligned}$$

and thus

$$\mathcal{A}\Big((\mathbf{n} e_0)\mathbf{A}(e_0 \mathbf{n})\Big) = \mathbf{n}\mathbf{a}\mathbf{n} + e_o.$$

© The Eurographics Association 2004.

13.7. Rotations in projective space

In the last section we saw how a reflection in $\mathbb{E}^2$ has to be expressed in projective space $\mathbb{PE}^2$ when applied to homogeneous vectors. Since a rotation expressed by a rotor is nothing else than two consecutive reflections, a rotor may also take on a different form in projective space.

Suppose we want to rotate the vector $\mathbf{a} \in \mathbb{E}^2$ by reflecting it first on $\mathbf{n} \in \mathbb{E}^2$ and then on $\mathbf{m} \in \mathbb{E}^2$. However, we want to do this in projective space where $\mathbf{A} = \mathcal{P}(\mathbf{a}) \in \mathbb{PE}^2$. Since a reflection on $\mathbf{n}$ has to be expressed as $(\mathbf{n}\,\mathsf{e}_o)$ and a reflection on $\mathbf{m}$ as $(\mathbf{m}\,\mathsf{e}_o)$, the rotation of $\mathbf{A}$ has to look like this

$$(\mathbf{m}\,\mathsf{e}_o)\,(\mathbf{n}\,\mathsf{e}_o)\,\mathbf{A}\,(\mathsf{e}_o\,\mathbf{n})\,(\mathsf{e}_o\,\mathbf{m}) = R\,\mathbf{A}\,\widetilde{R}, \quad R := (\mathbf{m}\,\mathsf{e}_o)\,(\mathbf{n}\,\mathsf{e}_o).$$

Such a double reflection is illustrated in figure 22. Here vector $\mathbf{a} \in \mathbb{E}^2$ is represented in $\mathbb{PE}^2$ by $\mathbf{A}$. A first reflection of $\mathbf{A}$ on $\mathbf{n}\,\mathsf{e}_o$ gives $\mathbf{B}$. A further reflection of $\mathbf{B}$ on $\mathbf{m}\,\mathsf{e}_o$ gives $\mathbf{C}$.

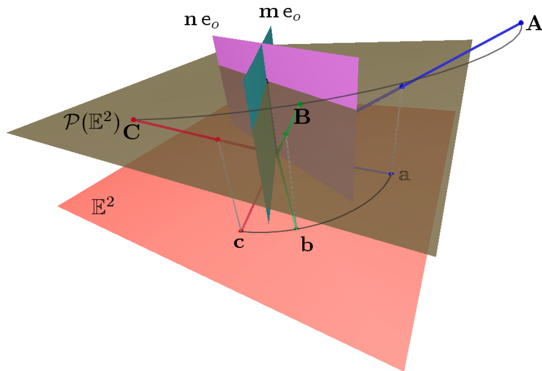


Figure 22: *Double reflection of homogenous vector* $\mathbf{A}$ *on reflection planes* $\mathbf{n}\,\mathsf{e}_o$ *and* $\mathbf{m}\,\mathsf{e}_o$ *in* $\mathbb{PE}^2$.

However, the expression for R can be simplified.

$$\begin{aligned} R &= (\mathbf{m}\,\mathsf{e}_o)\,(\mathbf{n}\,\mathsf{e}_o) \\ &= -\mathbf{m}\,\mathbf{n}\,\mathsf{e}_o\,\mathsf{e}_o \\ &= -\mathbf{m}\,\mathbf{n}. \end{aligned}$$

That is, compared to the expression of the rotor in $\mathbb{E}^2$, a minus sign is introduced. This, however, cancels out when the rotor is applied.

$$R\,\mathbf{A}\,\widetilde{R} = (-\mathbf{m}\,\mathbf{n})\;\mathbf{A}\;(-\mathbf{n}\,\mathbf{m}) = (\mathbf{m}\,\mathbf{n})\;\mathbf{A}\;(\mathbf{n}\,\mathbf{m}).$$

We may also argue that since an overall scalar factor is of no importance for homogeneous vectors with respect to their projection into Euclidean space, the minus sign of the rotor in projective space may be neglected. Hence, **we can use the same representation of a rotor in Euclidean and projective space**.

© The Eurographics Association 2004.

PART FOUR

Animation and Motion

Dietmar Hildenbrand

In this section, we focus on an application particularly suitable to Conformal Geometric Algebra, animations and motions.
It is based on the previous sections and on chapter 3 of the tutorial script [PH03]. Some other introductions to the Geometric Algebra will be found in[MDB01],[LHR01],[BC01] and [DM02].

We use the **CLUCalc** software to **calculate with Geometric Algebra** and to **visualize the results** of these calculations. CLUCalc is available for download at [Per02]. With help of the CLUCalc Software you are able to edit and run Scripts called **CLUScripts**. A screenshot of CLUCalc can be seen in figure 3 (p. 18).

14. CLUCalc example RotationAxis

According to table 6 lines are basic entities in Conformal Geometric Algebra. They are used to represent the axis of rotation for transformations and motions.
In the following CLUScript RotationAxis.clu a line representing a rotation axis is shown in green based on the red points a, b.

```
DefVarsN3();
:IPNS;

:Red;
:a = VecN3(0,-2,0);
:b = VecN3(0,2,0);

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

DefVarsN3(); in this CLUScript 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 6. The leading colon means that this geometric object is not only computed, but also visualized.

With help of **axis = *(a ∧ b ∧ einf);** a bivector representing a line *axis* is computed.
According to table 6 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 calculated, visualized and printed.

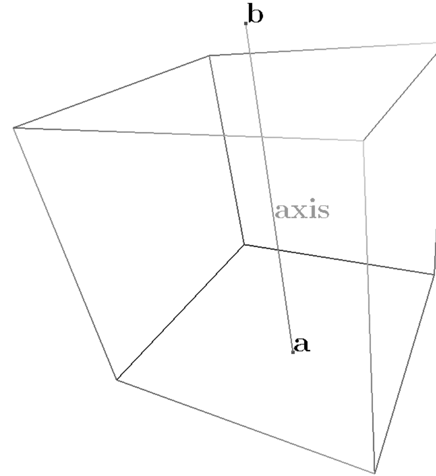


Figure 23: *RotationAxis.clu*

Table 6: *list of the conformal geometric entities*

entity	representation	dual representation
Sphere	$s = p - \frac{1}{2}r^2 e_\infty$	$s^* = x_1 \wedge x_2 \wedge x_3 \wedge x_4$
Point	$p = \mathbf{x} + \frac{1}{2}\mathbf{x}^2 e_\infty + e_0$	$x^* = s_1 \wedge s_2 \wedge s_3 \wedge s_4$
Plane	$\pi = \mathbf{n} + d e_\infty$	$\pi^* = x_1 \wedge x_2 \wedge x_3 \wedge e_\infty$
Line	$l = \pi_1 \wedge \pi_1$	$l^* = x_1 \wedge x_2 \wedge e_\infty$
Circle	$z = s_1 \wedge s_2$	$z^* = x_1 \wedge x_2 \wedge x_3$
Point Pair	$P_{pi} = s_1 \wedge s_2 \wedge s_3$	$P_{pi}^* = x_1 \wedge x_2$

Figures generated by CLUScripts are labeled by the name of the script.
All the CLUScripts of this section can be downloaded at

```
http://www.dgm.informatik.tu-
darmstadt.de/staff/dietmar/
```

For details regarding CLUScript please refer to the CLUCalc online help [Per02].

Table 6 lists the conformal geometric entities. **x** and **n** are marked bold since they represent 3D entities

- **x** is a 3D point and $\mathbf{x}^2$ its scalar product

© The Eurographics Association 2004.

- **n** is a normalized 3D normal vector

15. Transformations

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

$$\mathbf{a}_{transformed} = V\,\mathbf{a}\,\widetilde{V}.$$

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

15.1. Rotors

For rotations, the operator

$$R = e^{-\frac{\phi}{2}l} \tag{42}$$

describes a so-called **rotor**.
l is the rotation axis represented by a normalized bivector and ϕ is the rotation angle around this axis.
R can also be written as

$$R = cos(\frac{\phi}{2}) - l sin(\frac{\phi}{2}) \tag{43}$$

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

$$\mathbf{a}_{rotated} = R\,\mathbf{a}\,\widetilde{R}.$$

Note : $\widetilde{R}$ is the reverse of R.

In the following CLUScript example `Rotor.clu` we will rotate the sphere *Earth* around the sphere *Sun* located at the origin.

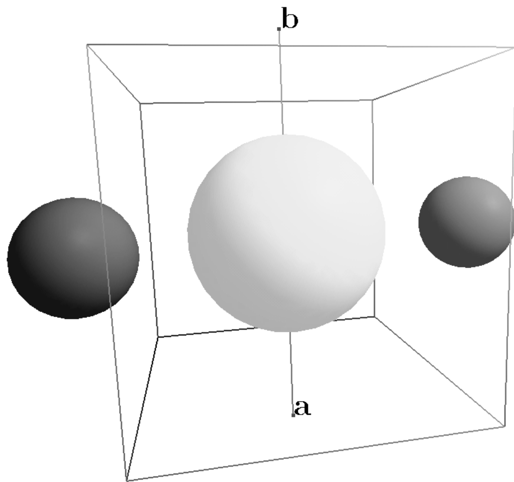


Figure 24: *Rotor.clu*

```
_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;
```

The bivector representing the rotation axis is computed as before and normalized with help of the *abs*-function.

```
: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 6). 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 help of the product $R Earth \widetilde{R}$.
The rotation operator depends on the fixed *axis* and the continuously changing *angle*.

15.2. Translators

In Conformal Geometric Algebra, translations can be expressed in a multiplicative way with help of **translators** T defined by

$$T = e^{\frac{e_\infty \mathbf{t}}{2}} \tag{44}$$

where **t** is an inhomogenous vector

$$\mathbf{t} = t_1 e_1 + t_2 e_2 + t_3 e_3$$

Another form of a translator T is

$$T = 1 + \frac{e_\infty \mathbf{t}}{2}$$

© The Eurographics Association 2004.

Proof :
With help of the Taylor series

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

Since $(e_\infty)^2 = 0$

$$T = 1 + \frac{e_\infty \mathbf{t}}{2}$$

.

15.3. Rigid Body Motion

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

$$M = RT \quad (45)$$

with
R being a rotor (see section 15.1).
T being a Translator(see section 15.2).
A rigid body motion of an object **a** is described by

$$\mathbf{a}_{rigid_body_motion} = M\mathbf{a}\widetilde{M}.$$

In the CLUScript example `RigidBody.clu` we will perform a rigid body motion of the sphere *Earth*. It is based on the rotation example of the previous section.

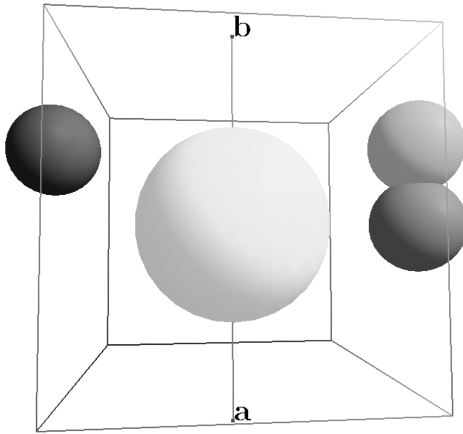


Figure 25: *RigidBody.clu*

```
_DoAnimate = 1; 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;

:Yellow;
:Sun = e0 -0.5*einf;
:Red;
:Earth=VecN3(2,0,0)-0.125*einf;

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

TVEC = angle/4*e2;
?T= exp(e*TVEC/2);

:Green;
:T* Earth * ~T;
?Motor=R*T;

:Blue;
:Motor* Earth * ~Motor;
```

The sun is drawn as a yellow sphere.
The red sphere is the basis for the rigid body motion of the earth.
The green sphere shows only the translation part of the motion. In this example it is a translation in the direction of e_2 dependent on the angle of the rotation.
The blue sphere is rotated and translated with help of the geometric product $Motor\ Earth\ \widetilde{Motor}$.

16. Interpolation of motions

An alternative description of a rigid body motion is a screw motion that is very suitable for the interpolation of motions.

16.1. Screw Motion

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 **a** is described by

$$\mathbf{a}_{screw_motion} = M\mathbf{a}\widetilde{M}.$$

with the motor

$$M = e^{-\frac{\theta}{2}(\mathbf{l} + e_\infty \mathbf{m})} \quad (46)$$

whereas

- **l** is a bivector representing an axis through the origin,
- θ is the angle of rotation,
- **m** is a 3D vector.

© The Eurographics Association 2004.

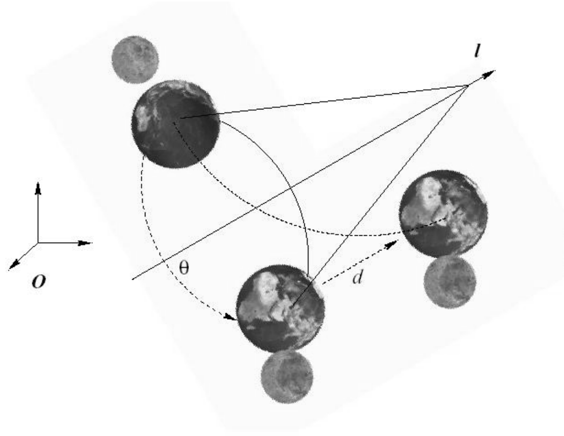


Figure 26: *Screw motion along l*

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

$$M = e^{-\frac{\theta}{2}(l+e_\infty \mathbf{d})} \tag{47}$$

whereas (see figure 26)

- $l = \mathbf{l} + \mathbf{m}^\perp$ 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 [BR].

16.2. Interpolation of twists

The exponent of a motor representing a screw motion is called a **twist**. Assume two transformations described by the two twists T_1 and T_2.

$$T_1 = -\frac{\theta_1}{2}(l_1 + e_\infty \mathbf{d_1})$$

$$T_2 = -\frac{\theta_2}{2}(l_2 + e_\infty \mathbf{d_2})$$

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

$$T(t) = (1-t) * T_1 + t * T_2 \tag{48}$$

with the resulting Motor

$$M(t) = e^{T(t)} \tag{49}$$

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

$$M(0) = e^{T_1}, M(1) = e^{T_2},$$

In the CLUScript example `LinCombRBM.clu` an interpolation of two transformations is performed based on the linear interpolation of twists.

```
...

PHI1 = 1;
d1=-0.7;
M1 = d1*d1_vec;
?Twist1 = -PHI1 / 2 *(L1+e*M1);

Motor1 = exp(Twist1);

:Green;
:Motor1* Earth1 * ~Motor1;

PHI2 =Pi/2;
d2=-0.1;
M2 = d2*d2_vec;
?Twist2 = -PHI2/2*(L2+einf*M2);

Motor2 = exp(Twist2);

:Magenta;
:Motor2* Earth1 * ~Motor2;

:Blue;
?LinComb_LOG = (1-t)*Twist1+t*Twist2;

?LinComb = exp(LinComb_LOG);
:LinComb* Earth1 * ~LinComb;
```

The twists *Twist*1 and *Twist*2 and the motors *Motor*1 and *Motor*2 are computed according to equation 47. *LinComb_LOG* is related to the linear interpolation of the two twists dependent on the continuously changing parameter t. It has to be exponentiated in order to describe a motion.

17. Kinematic chains

Objects like robots or virtual humans can be modelled as a set of rigid links connected together at various joints. These objects are described as kinematic chains.
In the following CLUScript example `KinematicChain.clu` a robot with 5 degrees of freedom (DOF) is visualized. With help of sliders you are able to change the 5 angles $\theta_1, \theta_2, \theta_3, \theta_4, \theta_5$.
In the include file `robot_globals.clu` all the robot parameters are described.

```
len = (0.8, 0.7, 0.6, 0.0, 0.3);
rad = (0.2, 0.15,0.08,0.07,0.04);
angle=( 0, -1, -0.4, 0.5, 0 );
rot_direction = (9,0,0,0,0);
```

Each cylindrical link is described with help of its length and radius. Each joint is described by a direction of rotation and the default joint angle.

The function computeRotation in the include file `Environment.clu` is responsible for computing the local rotations of each joint dependent on the direction of rotation and the angle.

© The Eurographics Association 2004.

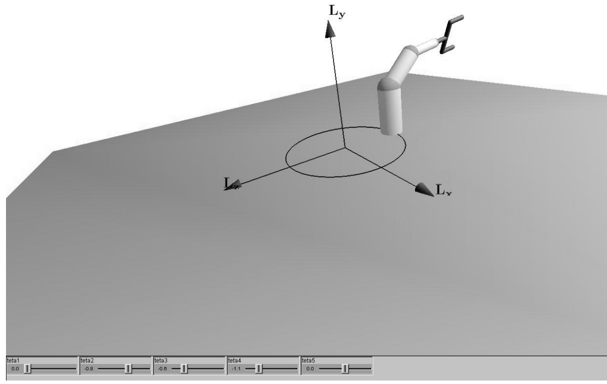


Figure 27: *KinematicChain.clu*

```
computeRotation = {
  // evaluate the rotor dependend on
  // _P[1] : the direction of rotation
  // _P[2] : the angle
  if (_P[1] ==0) {
    Rot = RotorN3(1,0,0, _P[2]);
  }
  else {
    ...
  }
}
```

The rotations are computed with help of the CLUCalc function RotorN3. The first 3 parameters mean a 3D rotation axis and the last one describes the angle.

The function computeTransformation in the include file `Cylinder.clu` is responsible for computing the chain of transformations for the different joints.

```
computeTransformation = {
  _idx = _P[1];
  computeRotation(rot_direction[_idx],
                  angle[_idx]);

  ...

M=M*TranslatorN3(0,0,len[_idx-1])*Rot;

  ...
}
```

M is the global transformation accumulated with all the local transformations. Each local transformation consists of the translation dependent on the length of the relevant link and the rotation of the relevant joint computed by the function computeRotation().

18. Inverse Kinematics

In the previous section we determined the position and orientation of a robot gripper in terms of the joint angles. This section is concerned with the inverse problem of finding the joint angles in terms of a position and orientation.

In Conformal Geometric Algebra, this so-called **inverse kinematics** can be done in a geometrically very intuitive way because of its easy handling of intersections of spheres, circles, planes etc.

In the following CLUScript example `InverseKinematics.clu` we show the inverse kinematics of the 5DOF robot of the previous section. With

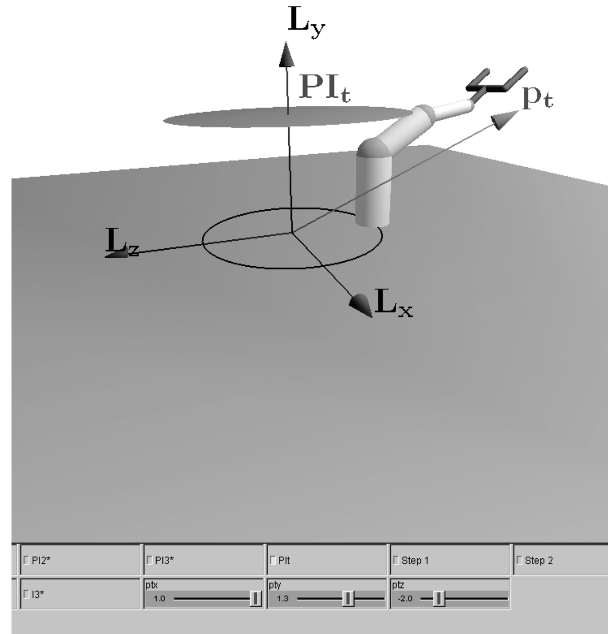


Figure 28: *InverseKinematics.clu*

help of sliders you are able to change the target position p_t of the gripper while the orientation of the gripper is determined by the plane PI_t

This approach is based on the paper [BCZE04]. With help of some check boxes you are able to visualize the different steps of the algorithm.

18.1. Step 1

In the first step point p_2 is calculated.

p_2 is the joint location of the last link of the robot.

```
St = pt - 0.5*len[5]*len[5]*einf;
```

This means that it has to lie on the sphere S_t with the center point p_t and with the length of this link as radius. Please refer to table 6 for details.

```
zt = St^PIt;
```

Since the gripper also has to lie in the orientation plane PI_t, we have to intersect it with S_t. The result is the circle z_t.

© The Eurographics Association 2004.

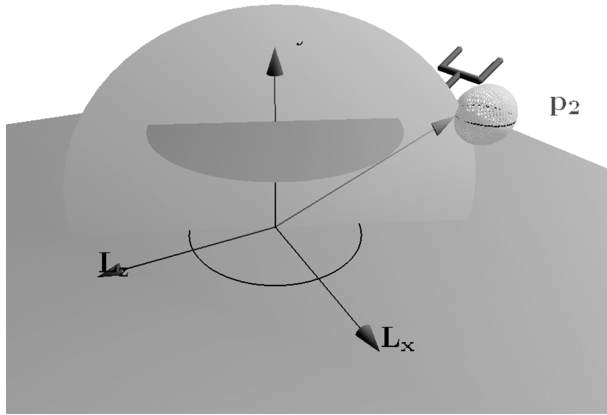


Figure 29: *InverseKinematics.clu, Step 1*

```
jz_dual = zt^einf;
ly_dual=e0 ^ e2 ^ einf;
ly =*ly_dual;

l_proj = proj(pt,ly_dual);

ld_dual = l_proj ^ pt ^ einf;
ld=*ld_dual;

PIj_dual = jz_dual ^(ld_dual*(einf*e0));
PIj = *PIj_dual;
```

Since p_2 also has to lie in a plane spanned by the y coordinate and p_t, this plane PI_j is computed.

```
Pp2= *(PIj ^ zt);
// choose one of the two points
p2 = DissectFirst(Pp2);
```

Its intersection with the circle z_t results in a point pair. Only one makes sense from the mechanics point of view. We choose it as our point p_2. Please find some details on dissecting a point pair in section 9.2.

18.2. Step 2

In the second step point p_0 is calculated.

```
h=len[1];
d0 = sqrt(r*r+h*h);
S0 = e0 - 0.5*d0*d0*einf;
PI0 = e2 + h*einf;
z0=S0^PI0;
```

p_0 is the location of the first joint. It rotates on a circle in height h with radius r.

```
PI1 = *(ly_dual^p2);
Pp0 = *(z0^PI1);
// choose one of the two points
p0 = DissectSecond(Pp0);
```

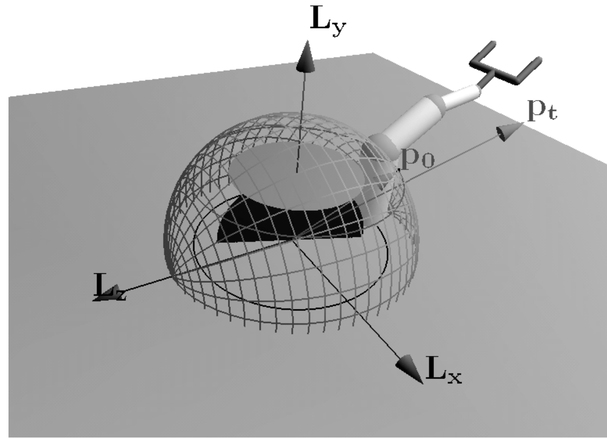


Figure 30: *InverseKinematics.clu, Step 2*

The intersection with plane PI_1 delivers a point pair P_{p0}. Again, we choose the second one.

18.3. Step 3

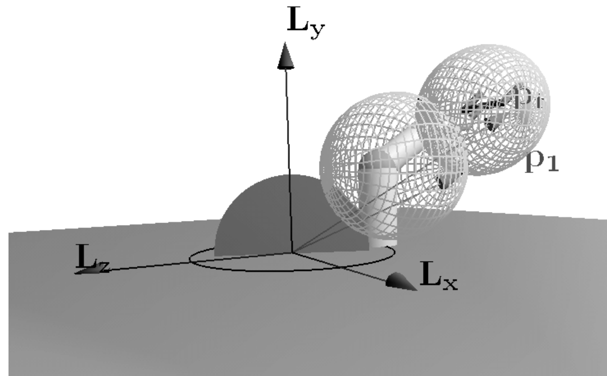


Figure 31: *InverseKinematics.clu, Step 3*

In the third step point p_1 is calculated.

```
S1 = p0 - 0.5*len[2]*len[2]*e;
S2 = p2 - 0.5*len[3]*len[3]*e;
Pp1 = *(S1^S2^PI1);
// choose one of the two points
p1 = DissectSecond(Pp1);
```

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 PI_1.

© The Eurographics Association 2004.

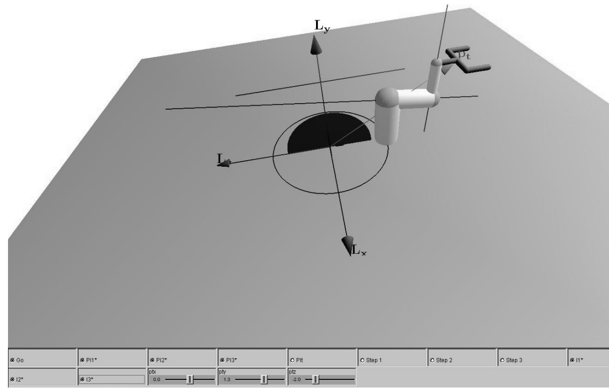


Figure 32: *InverseKinematics.clu, Step 4*

18.4. Step 4

In the fourth step all the planes and lines, that are needed for the computation of the angles of the joints are calculated.

```
PI3_DUAL = p1 ^ p2 ^ pt ^einf;
PI2_DUAL = e3 * I;
PI1_DUAL = *PI1;
l1_dual = p0^p1^einf;
l2_dual = p1^p2^einf;
l3_dual = p2 ^ pt ^ einf;
```

18.5. Computation of the joint angles

Now, we are able to compute all the joint angles

```
angle[1]=
Pi/2-computeAngle(PI1_DUAL, PI2_DUAL);
angle[2]=
-computeAngle(l1_dual, ly_dual);
angle[3]=
computeAngle(l1_dual,l2_dual);
angle[5]=
Pi-computeAngle(PI1_DUAL,PI3_DUAL);
angle[4] = -
computeAngle(l2_dual, l3_dual);
```

with help of the function

```
computeAngle = {acos ( (_P[1] . _P[2])
              /(abs(_P[1])*abs(_P[2])) )}
```

Please find some details on the computing of angles in section 9.3.

19. Dynamics

Since a time dependent versor $D = D(t)$ describes the rigid body motion, its time derivative $\dot{D}(t)$ is needed for the generalized **screw velocity** that we are able to split into the well-known **angular velocity** and **translational velocity**.

For the purpose of dynamics also the second derivative is needed. The dynamical equation for combined rotational and translational motion takes the compact form

$$\dot{P} = F \tag{50}$$

with P as **comomentum** and F as **coforce**.

For details please refer to [DDL02]. In chapter 19, Hestenes and Fasse describe the basics of rigid body mechanics in Conformal Geometric Algebra.

© The Eurographics Association 2004.

PART FIVE

Implementation and Performance of Geometric Algebra

Daniel Fontijne & Leo Dorst

We describe and compare several approaches to implementing a numerical Geometric Algebra package. We also compare the performance and elegance of implementing geometry using Geometric vs using linear algebra in a ray tracing application.

20. Introduction

In this short paper we give an overview of existing approaches to developing numerical, low dimensional Geometric Algebra implementations. Low dimensional because these approaches fail to be practical beyond 8D or 9D Geometric Algebras. However, typical computer graphics geometry problems require 3, 4 or 5 dimensional GAs, so this limitation of scope should not be a problem.

We focus mainly on the run time performance of the implementation because efficiency is still very important for computer graphics and related areas. A typical remark after a Geometric Algebra from a game developer or computer graphics programmer is 'I learned some useful techniques, but why present this if it is slower than linear algebra'.

The packages we consider are GABLE [MDB01], GAP [ZD03], CLU [Per02], NKlein [Fle], Gaigen [FBD01], Clifford [Sut03] and Gaigen 2. The performance difference between these packages turns out to be in the order of several magnitudes.

We also compare the use of Geometric Algebra to the use of traditional linear algebra in the context of a ray tracing application. We implemented a simple ray tracer five times, where the only difference between the implementations was the way we did the geometry. This allows us to compare the efficiency of the various ways of doing geometry and the elegance of the equations used implement the geometry.

21. Issues in Efficiently Implementing a Numerical Geometric Algebra Package

There are four issues inherent to Geometric Algebra that complicate its efficient implementation. Three of these issues are a consequence of the fact that GA is a very rich mathematical language. This richness is to the advantage of GA user, but it complicates the job of the implementation developer. The issues are

scalars	point pairs	free vectors
5D pseudoscalar	circles	free bivectors
the point at infinity	spheres	free trivectors
points	flat points	tangent vectors
	lines	tangent bivectors
	planes	tangent trivectors

Table 7: *Blades in the conformal model. See [Dor03] for a formal classification of these blades.*

1. The large number of primitive objects: In theory, we need only one primitive in Geometric Algebra: the multivector. However, this is similar to stating the only car you'll ever need is a tank. While the multivector can contain and represent any other primitive, it is usually much slower in use than specialized primitives such as a '3D bivector' or a '5D versor'. Introducing specialized primitives can be useful for several reasons: it is more efficient to store and compute with such a specialization: they require less coordinates to be represented and hence less operations when computed upon. Also, the GA user thinks in terms of specific (groups of) primitives instead of the general multivector.
   The first classsification of specialized primitives we make is based on grade. In 3D, we distinguish scalars (grade 0), vectors (grade 1), bivectors (grade 2) and trivectors (grade 3). The are also odd (grade 1 and 3) and even versors (grade 0 and 2).
   The second class of primitives arises once we introduce basis vectors with some special meaning or inner product. For instance, in the conformal model, the special definition of points introduces a whole array of primitives (see table 7). Compare this to traditional use of linear algebra for doing geometry, where the computational primitives are limited to scalars, matrices, vectors and possibly quaternions.
2. The large number of basic operations. Table 8 lists some basic operations that every GA implementation should provide.
3. The arbitrary definition of the inner product: In Geometric Algebra, it is common to use vectors that have a 0 (i.e., *null* vectors) or negative inner product with themselves. Even *reciprocal* basisvectors are used. E.g., the conformal model uses $e_0 \cdot e_0 = 0, e_\infty \cdot e_\infty = 0, e_0 \cdot e_\infty = -1$. Arbitrary definition of the inner product complicates efficient implementation because the inner products can not easily be 'hard coded'.
4. The large number of coordinates required to store a high dimensional Geometric Algebra primitive with respect to a basis. A n-dimensional Geometric Algebra requires 2^n coordinates. Of course, this is only a problem if the GA implementation developer chooses to represent the

© The Eurographics Association 2004.

geometric product	outer product
Hestenes inner pr.	modfied Hestenes inner pr.
left contraction	right contraction
scalar product	commutator product
meet	join
projection	rejection
exponent	logarithm
add	subtract
negation	reversion
inversion	dualization
grade involution	clifford conjugate

Table 8: *Some common Geometric Algebra operations.*

```
// a = dual circle, b = dual sphere,
// c = intersection of circle & sphere
void c3gai_opt_04_op_02(const float *a,
    const float *b, float *c) {
    c[0]=a[2]*b[0]-a[1]*b[1]+a[0]*b[2];
    c[1]=a[4]*b[0]-a[3]*b[1]+a[0]*b[3];
    c[2]=a[5]*b[0]-a[3]*b[2]+a[1]*b[3];
    c[3]=a[5]*b[1]-a[4]*b[2]+a[2]*b[3];
    c[4]=a[7]*b[0]-a[6]*b[1]+a[0]*b[4];
    c[5]=a[8]*b[0]-a[6]*b[2]+a[1]*b[4];
    c[6]=a[8]*b[1]-a[7]*b[2]+a[2]*b[4];
    c[7]=a[9]*b[0]-a[6]*b[3]+a[3]*b[4];
    c[8]=a[9]*b[1]-a[7]*b[3]+a[4]*b[4];
    c[9]=a[9]*b[2]-a[8]*b[3]+a[5]*b[4];
}
```

Figure 33: *A function taken from the bowels of source code generated by Gaigen. It computes the intersection of a circle and a sphere in the conformal model.*

multivectors on such a basis instead of using some other method.

The combination of the number of primitives with the number of operations make it a hard task to achieve optimal efficiency. Table 7 lists 16 primitives for the conformal model alone (omitting the versors, that can represent all conformal (angle preserving) transformations!). A binary operation on these primitives allows for 16x16=256 combinations. Given about the same number of basic binary operations leads a number of functions in the order of 10^3 to 10^5, when versors specialization are also included. These functions can grow quite large (not to mention boring) because of the number of coordinates required to represent a primitive grows as the dimensionality of the algebra grows. E.g., figure 33 shows a function generated by Gaigen that computes the intersection of a circle and a sphere in the conformal model.

Efficient implementation by hand, as is often done for low dimensional linear algebras, is close to impossible.

We must note that the 'optimal performance' we are referring to depends heavily on how much flexibility the GA user is willing to sacrifice. For instance, if a users *wants* to use only general multivectors, many types of optimizations are barred by that choice. But by programming explicitly in terms of highly specialized primitives (e.g. lines, spheres), the user sacrifices some of the flexibility of Geometric Algebra: many GA algorithms will work for many types of primitives and this stimulates to think and program in terms of families of primitives instead of specific instances. However, it is possible to define functions over multivectors and then specialize them for specific primitives.

In general, the more information the programmer using GA is willing to specify at compile time, the more optimizations can theoretically be applied by the Geometric Algebra implementation. An analogy to this is type systems in programming languages: a dynamic (run-time) type systems is versatile but slow, a static (compile-time) type system is more restrictive but also more efficient.

22. Approaches to Implementing a Numerical Geometric Algebra Package

22.1. Matrix based

Most of the basic operations of Geometric Algebra are linear, and linear operations can be implemented using matrix-vector or matrix-matrix multiplication. Suppose we want to implement a 3D GA. The general multivector is then represented using 8 coordinates. Then, to compute the geometric product $\mathbf{A}\,\mathbf{B}$, we fill a 8×8 matrix $[\underline{\mathbf{A}}^G]$ in the right way (see figure 34 for an illustration, and [MDB01] for the details). We can then compute the geometric product by multiplying this matrix with a vector $[\underline{\mathbf{B}}]$ appropriately filled with the coordinates of $\mathbf{B}$. We can even multiply two of these matrices, and the resulting matrix can be immediately used to compute another geometric product: e.g., $([\underline{\mathbf{A}}^G][\underline{\mathbf{B}}^G][\underline{\mathbf{C}}])$.

To compute the product derived from the geometric product, specific entries in these matrices are set to zero according to the rules that define the products. This allows us to compute outer products, inner products and scalar products.

Addition and subtraction and most unary operations are trivial to implement. Inversion can be performed using matrix inversion.

This is, in short, how GABLE works. The upside of this approach is that this a simple and effective way to implement GA. The downside is that it is slow, wastes memory and does not scale well with the dimension of the algebra. All of this is due to the large matrices involved in computing the products. [MR02] implemented an algorithm for computing singularities of 3D vector fields with Geometric Algebra in GABLE. That same algorithm ported to Gaigen (below) ran approximately 6000× faster.

© The Eurographics Association 2004.

$$[\underline{\mathbf{A}}^G] = \begin{bmatrix} +A_s & +A_1 & +A_2 & +A_3 & -A_{12} & -A_{13} & -A_{23} & -A_{123} \\ +A_1 & +A_s & +A_{12} & +A_{13} & -A_2 & -A_3 & -A_{123} & -A_{23} \\ +A_2 & -A_{12} & +A_s & +A_{23} & +A_1 & +A_{123} & -A_3 & +A_{13} \\ +A_3 & -A_{13} & -A_{23} & +A_s & -A_{123} & +A_1 & +A_2 & -A_{12} \\ +A_{12} & -A_2 & +A_1 & +A_{123} & +A_s & +A_{23} & -A_{13} & +A_3 \\ +A_{13} & -A_3 & -A_{123} & +A_1 & -A_{23} & +A_s & +A_{12} & -A_2 \\ +A_{23} & +A_{123} & -A_3 & +A_2 & +A_{13} & -A_{12} & +A_s & +A_1 \\ +A_{123} & +A_{23} & -A_{13} & +A_{12} & +A_3 & -A_2 & +A_1 & +A_s \end{bmatrix}$$

Figure 34: *Geometric product matrix for 3D Euclidean multivector A. Notation: A_{12} is coordinate* **A** *that refers to* $\mathbf{e}_1 \wedge \mathbf{e}_2$.

22.2. Run-time Configurable

A number of implementations (GAP[ZD03], CLU[Per02], NKlein[Fle], and a hardware implementation [PGS03]) use more or less the following approach: First, the programmer who wants to use GA defines a vector space through some kind of datastructure. This definition includes the dimensionality of the space, the signature of the basis vectors and possibly some extra information like the names or canonical order of these basis vectors.

Given the space, basis blades can be defined, which in turn can summed (that is, stored in lists or arrays) to create general multvectors or any other primitive GA object. Commonly, each coordinate is accompanied by a bitmap that indicates what basis blade it refers to.

To compute the products, one iterates over the basis blades of both operands and computes the required product for the every pair of basis blade. This computation can be done using a precomputed multiplication table or entirely at run time. The definition of the vector space is required for such computations, since that defines the outcome of the inner products. The results of the loop are summed, to acquire the final result of the product.

Many other basic operations are also implemented for each (pair of) basis blade, and to compute such operations for a multivector or other primitive, one simply iterates over each blade of the (pair of) multivectors. For lack of a better name we call this class of approaches 'run-time configurable'.

This approach is faster than the matrix approach. In most of these implementations it is easy to represent sparse multivectors efficiently. This saves both memory and computation time. It is also a very intuitive and versatile implementation method. However, due the explicit iteration over the individual basis blades, this type of implementation is far from optimal. The loops cause many misprediction branches. Benchmarks presented in section 24 will show that this approach is in the order of 100× slower than traditional hand coded linear algebra.

22.3. Generative Programming

The third class of approaches we discuss here is generative programming [CE00]. Generative programming means that you write a program that outputs programs according to some specification in a domain specific language. This can be achieved in several ways. The most explicit way is to write a program that outputs source files that are then compiled by an ordinary compiler. This is what Gaigen and Gaigen 2 do. Another approach is the use of template metaprogramming techniques. Here, the template capabilities of C++ are used to let the C++ compiler generate the required code at compile-time. This is what Clifford does.

22.3.1. Gaigen

Gaigen stands for Geometric Algebra Implementation Generator [FBD01]. Gaigen takes a definition of a Geometric Algebra and turns this into C++ code that implements this definition. This removes the need for explicit loops at runtime that were required with the run-time configurable approaches.

The definition of the algebra includes the dimensionality of the space, and the names, order and signature of the basis vectors, and what products and other operations are required.

Gaigen generates from this definition a C++ class that implements the general multivector. However, this general multivector class exploits sparseness of blades and versors by storing only grade parts that are not zero. To make this possible, each multivector variable contains a bitfield that tracks what grade parts are (non-)zero.

Particular about Gaigen is its use of a profile guided optimization (PGO) step that allows for optimization of the generated code: once a full application that used has been Gaigen has been developed, the program is run with PGO enabled and Gaigen then counts what combinations of operations and primitives are used. Using this information it regenerates the code that implements the most-used combinations of operations and primitives more efficiently. A typical piece of code generated by Gaigen is shown in figure 33.

Gaigen is pretty efficient (about 25× faster than CLU),

© The Eurographics Association 2004.

but about 3× slower than optimal code. This non-optimality of Gaigen is mainly caused by the run-time tracking of which grade parts are non-zero: this requires conditional jumps that are not required in truly optimal code.

22.3.2. Gaigen 2

Gaigen 2 is currently under development. It targets to be an optimally efficient Geometric Algebra implementation for multiple programming languages. The user of Gaigen 2 will not only provide the definition of the vector space, but also definition of specialized primitives and their properties (like bladedness). Using this information Gaigen 2 generates an implementation for the requested output language (C++, Java, Matlab).

Like its predecessor, Gaigen 2 also uses a PGO step to find out what functions are called on what types of primitives by the program. Given the profile information, Gaigen 2 regenerates the source code, which will this time result in near optimal run time performance.

Additionally, the user can write functions in Gaigen 2's own programming language. For instance, the user can define some function over general multivectors. When this function is called with specialized primitives as arguments, Gaigen 2 will generate an optimized version of the function specifically for those arguments.

Gaigen 2's own programming language is C-like. However, it supports Geometric Algebra directly. For Geometric Algebra types, the precedence of the operators (see table 9) is handled correctly. Optimizations that Gaigen 2 will be able to perform on functions written in its own language include:

- algebraic manipulations to simplify expressions,
- generating very efficient code for exponentiation and taken a logarithm of a versor (this is useful for interpolation of transfomations in computer graphics),
- detecting linear operations and implementing those as matrix-vector multiplications when that is more efficient.
- detecting piece of code that operate only in a subspace of the algebra and taking advantage of that by switching to appropriate (lower dimensional) basis.

22.3.3. Clifford

Clifford [Sut03] is a C++ implementation of Geometric Algebra that uses template meta-programming and expression templates (see e.g., [VJ02]). This means that the C++ compiler gets to do the code generation work by expanding a set of templates and template classes that define the algebra, its primitives and its operations. Meta-programming and expressiong templates have also been used to implement linear algebra [CE00] and tensor algebra [Lan02] efficiently.

This time, a *traits class* is used to define the vector space (dimensionality of the space, signature of the basis vectors).

$\wedge$	outer product
&	meet
|	join
$\cdot$	inner product
	geometric product
+	add
-	subtract

Table 9: *Precedence order of the operators in Geometric Algebra. The outer product $\wedge$ is the most important operator since it is used to construct the blades. Note that this precedence order is entirely different from standard precedence order used for integers and floats, and that the 'space' operator is used for the Geometric Algebra.*

This definition is used as a template argument to the templates that implements the multivector and the operations. At compile-time, the compiler expands the templates and this results in just the right code for the job. This is combined with expression templates to further increase efficiency.

This way, Clifford generates optimal code for most situations (equal or exceeding traditional linear algebra efficiency).

A disadvantage of this approach is that it works only for programming languages that allow for 'Turing Complete' template meta programming. Compilation of code can be quite slow (due to the compile time template expansion) and a standards compliant compiler is required (luckily, the quality of C++ compilers w.r.t. to the C++ ISO standard has increased over the last few years).

Unfortunately, the status of the Clifford source code is uncertain to us. The source code has disappeared from its respective website [Sut03] and a request for set of benchmarks of Clifford v.s. Gaigen in a ray tracer application (section 24) went unanswered.

22.4. Direct Integration into Programming Language

Optimal efficiency combined with maximum user satisfaction can only be achieved by directly integrating Geometric Algebra into programming languages. Compiler for these languages will have total control over the optimization process. Precedence of the Geometric Algebra operators can be handled correctly. No profile guided optimization or long template compile times will be required anymore.

We realize that new languages don't get into mainstream use overnight, but still, we entertain the idea that some day there might be a language designed (amongst others) for doing numerical and geometric computations that supports lin-

© The Eurographics Association 2004.

ear algebra, Geometric Algebra and tensor algebra directly, or at least seamlessly.

22.5. Summary Pro / Cons of the approaches

- Matrix based
  - pro: simple.
  - con: slow.
- Run time configurable
  - pro: versatile, can be implemented in most languages.
  - con: not very efficient.
- Code generation (like Gaigen)
  - pro: very efficient, can be implemented in many languages. With built-in language can get precedence of operators right.
  - con: requires PGO, requires separate program (the code generator). Building a code generator takes a lot of time.
- Code generation (meta programming)
  - pro: very efficient, requires no extra tools.
  - con: limited to few languages (and sturdy compilers), long compile times. Hard to implement and debug.
- Direct integration into programming language
  - pro: optimally efficient, gets precedence of operators right.
  - con: compiler extensions required.

23. Geometric Algebra Hardware

Little research has been done on implementing Geometric Algebra directly into hardware. The only research known to us is [PGS03]. A field programmable gate array (FPGA) was used as co-processor that can compute geometric products. Central to the design was a basis blade pipeline that could compute geometric products for individual basis blades. The results of these were summed afterwards. Multiple basis blade pipelines could be used in parallel, but due to hardware limitations, only one pipeline could be implemented on the FPGA. Benchmarks indicated that the single pipeline version is about as fast as Gaigen, given that the FPGA would operate at the same clockspeed as the CPU that Gaigen ran on.

Another approach to hardware implementation could be SIMD instructions fit for doing Geometric Algebra. However, the low level code for computing the products from Geometric Algebra resemble traditional cross products (or computing (sub-)determinants, if you will). CPU SIMD instruction sets (such as Intels SSE) are designed mostly for matrix-vector multiplications. They lack efficient 'swizzle-negate' hardware that enables computing these products efficiently.

Current graphics processors (GPUs) do have zero-cost swizzle-negate hardware and thus are a great target for doing fast Geometric Algebra computations. Geometric Algebra could easily be integrated into high level shading languages. It could also be used in compilers that intend to use GPUs as general purpose processors (e.g. BrookGPU, see[BFH*02])

The conformal model seems to be especially fit for hardware implementation. It is by nature more parallel than the traditional way of doing geometric computations. For instance, the code in figure 33 could theoretically execute in three cycles (multiply, add, add). However, this would require very wide SIMD hardware and fast memory access to get the 10 coordinates of the circle and the 5 coordinates of the sphere into the processor in time.

24. Performance and Elegance of Five Models of Geometry in a Ray Tracing Application

Computations of 3D Euclidean geometry can be performed using various computational models of different effectiveness. We decided to compare five alternatives, from plain old 3D linear algebra to the 5D conformal model using Geometric Algebra. We wanted to compare them not just theoretically, but by showing how you would implement a simple recursive ray tracer in each of them. The project was meant as a tangible case study of the profitability of choosing an appropriate model, investigating the trade-offs between elegance and performance for this particular application.

This section is a summary of the full paper [FD03] and more information and source code can be downloaded at [FD02].

The models we compare are: 3D linear algebra (*3D LA*); 3D Geometric Algebra (*3D GA*, which naturally absorbs the quaternions into 3D real geometry); 4D linear algebra (*4D LA*, i.e. the familiar homogeneous coordinates extended with Plücker coordinates); the 4D homogeneous model (*4D GA*, a Geometric Algebra which naturally absorbs Plücker coordinates of lines and planes into homogeneous computations); and the 5D conformal model (*5D GA)*. We picked both 3D LA and 4D LA because we wanted a basic and an advanced mainstream model as baseline. We selected 3D GA and 4D GA because they are the (improved) GA variants of the 3D LA and 4D LA models. The 5D GA model is used to demonstrate what kind of improvements are possible with more sophisticated models.

Our reasons for choosing a ray tracer as benchmark are the following. 1) Everybody familiar with computer graphics knows how a basic ray tracer works, and possibly has implemented one. 2) Implementing the core of a ray tracer can be done with relatively little code, which was important to us, since we were going to write many different implementations of the same algorithm. 3) A ray tracer contains a diverse selection of geometric computations, like rotation,

© The Eurographics Association 2004.

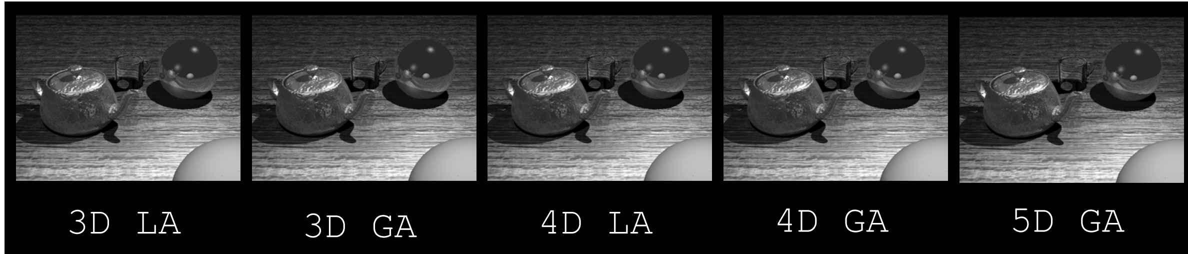


Figure 35: *The same result can be achi eved in many ways. These images are identical, but each one was rendered using a different model of 3D Euclidean geometry. The scene consist of 5 objects modeled with about 7800 triangles: a textured/bumpmapped teapot, a transparent drinking glass, a reflective sphere , a red diffuse sphere, and a textured/bumpmapped piece of wood.*

model	implementation	rendering time	executable size	run time memory usage
3D LA	standard	1.00×(0.99s)	52KB	6.2MB
4D LA	standard	1.22×	56KB	6.4MB
3D GA	Gaigen	1.86×	64KB	6.7MB
4D GA	Gaigen	2.62×	72KB	7.7MB
5D GA	Gaigen	4.58×	100KB	9.9MB
3D GA	CLU	72.0×	164KB	12.6M
4D GA	CLU	97.1×	176KB	14.7MB
5D GA	CLU	178.0×	188KB	19.0MB

Table 10: *Performance benchmarks run on a Pentium III 700 MHz notebook, with 256 MB memory, running Windows 2000. Programs were compiled using Visual C++ 6.0. All support libraries, such as fltk, libpng and libz were linked dynamically to get the executable size as small as possible. Run time memory usage was measured using the task manager.*

	3D LA	3D GA	4D LA	4D GA	5D GA
Line representation	$\mathbf{q}$: vector from origin to a point on the line $\mathbf{u}$: direction of the line	$\mathbf{q}$: vector from origin to a point on the line $\mathbf{u}$: direction of the line	$\mathbf{q}_1 = (\vec{q}_1 : 1)$, $\mathbf{q}_2 = (\vec{q}_2 : 1)$: two points $\mathbf{l} = (\vec{q}_1 - \vec{q}_2 : \vec{q}_1 \times \vec{q}_2)$	$\mathbf{q}_1, \mathbf{q}_2$: two points $\mathbf{l} = \mathbf{q}_1 \wedge \mathbf{q}_2$	$\mathbf{q}_1, \mathbf{q}_2$: two points $\mathbf{l} = \mathbf{q}_1 \wedge \mathbf{q}_2 \wedge \mathbf{e}_\infty$
Line-plane intersection	$\mathbf{q}_l$: point on line $\mathbf{u}$: direction of line $\mathbf{n}$: normal vector of plane δ: distance of plane to origin $\mathbf{q}_i = \mathbf{q}_l - \frac{((\mathbf{q}_l \cdot \mathbf{n}) - \delta)\,\mathbf{u}}{\mathbf{u} \cdot \mathbf{n}}$	$\mathbf{q}_l$: point on line $\mathbf{u}$: line direction $\mathbf{p}$: bivector of the plane δ: distance of plane to origin $\mathbf{q}_i = \mathbf{q}_l - \frac{((\mathbf{q}_l \wedge \mathbf{p})^* - \delta)\,\mathbf{u}}{(\mathbf{u} \wedge \mathbf{p})^*}$	$\mathbf{l} = (\vec{u} : \vec{v})$: line $\mathbf{p} = [\vec{n} : \delta]$: plane $\mathbf{q} = \left(\frac{\vec{v} \times \vec{n} + \delta\vec{u}}{\vec{u} \cdot \vec{n}} : 1\right)$	$\mathbf{l}, \mathbf{p}$: line, plane $\mathbf{q} = \mathbf{p}^* \cdot \mathbf{l}$	$\mathbf{l}, \mathbf{p}$: line, plane $\mathbf{q} \wedge \mathbf{e}_\infty = \mathbf{p}^* \cdot \mathbf{l}$ plus one flat point decomposition, if required

Figure 36: *Summary of the representation of a line and the intersection of a line and a plane in 5 models of Euclidean Geometry. See the full matrix at [FD02].*

© The Eurographics Association 2004.

translation, reflection, refraction, (signed) distance computation, and line-plane and line-sphere intersection computations. This allows us to show by example how to perform these computations in different models. But we emphasize that our main goal was to compare frameworks for representation and computation of geometry in some practical situation, not to build a ray tracer per se. The resulting ray tracer is not a marvel of contemporary computer graphics; yet it is sufficiently sophisticated to render images such as figure 35.

Table 10 gives benchmarks for the 5 ray tracer implementations (+3 extra where CLU was used instead of Gaigen). Our conclusions as far as performance goes were the following:

- More refined models of geometry are less efficient. The conformal model is about $2.5\times$ slower than 3D GA. The penalty for the 4D models was less severe.
- Gaigen is not as efficient as hand coded linear algebra. Since 3D GA and 3D LA use (on a low level) the same operations, the $1.86\times$ slowdown of 3D GA is entirely due to Gaigen.
- If we assume that we could improve Gaigen to match the performance of 3D LA for the 3D GA case, then we can extrapolate that, without further optimizations, the conformal model should be about $2.5\times$ slower than 3D LA.
- CLU is a far from optimally efficient GA implementation (both in processing time and memory).

On the elegance side of the equation we see great improvements by using more refined models. Figure 36 shows an two examples. The top example shows how a line is represented in each of the 5 models. In 3D LA and 3D GA the line is represented in terms of two separate vectors. The 4D models improve on this by having a separate primitive for representing lines, although 4D LA lacks the ability to express this in a coordinate free way. 5D GA generalizes the line representation in a way that also allows for circles.

The bottom example in figure 36 shows the improvements for the computing the intersection point of a line and a plane. The computations in the 3D models are basically the same, and quite involved. Both 4D models compute the same thing, but again the speech impediment of 4D LA/Plücker coordinates leads to an involved equation. 5D GA generalizes the equation of 4D GA so it works on line/circles and planes/spheres.

25. Conclusion

GA implementation is still in a state of flux. There is no single do-it-all implementation that is easy to use, optimally efficient and available in several programming languages, all at the same time.

However, we have shown that GA is usable right now for developing computer graphics applications by implementing a ray tracer using GA. Performance may still lack somewhat, but will no doubt improve in coming years.

Investment in development of advanced implementations such as programming languages directly supporting GA or special GA hardware may depend on the invention of some (profitable) 'killer applications' that are only possible using GA.

© The Eurographics Association 2004.

References

[aF02] AMOWICZ R. A., FAUSER B.: The CLIFFORD home page. Last visited 15. Sept. 2003. 18

[amo96] AMOWICZ R. A.: Clifford algebra computations with maple. In *Clifford (Geometric) Algebras* (1996), Baylis W. E., (Ed.), Birkhäuser, Boston, pp. 463–501. 18

[BC01] BAYRO-CORROCHANO E.: *Geometric Computing for Perception Action Systems*. Springer Verlag, NY, 2001. 33

[BCZE04] BAYRO-CORROCHANO E., ZAMORA-ESQUIVEL J.: Inverse kinematics, fixation and grasping using conformal geometric algebra. In *IROS 2004, September 2004, Sendai, Japan* (2004). to appear. 37

[BFH*02] BUCK I., FOLEY T., HORN D., SUGERMAN J., HANRAHAN P., HOUSTON M., FATAHALIAN K.: BrookGPU. *http://graphics.stanford.edu/projects/brookgpu* (2002). 44

[BR] B. ROSENHAHN TITLE = POSE ESTIMATION REVISITED S. . I. Y. . . P. . H.:. PhD thesis. 36

[Bro02] BROWNE J.: The grassmannalgebra book home page. HTML document, 2002. Last visited 15. Sept. 2003. 18

[CE00] CZARNECKI K., EISENECKER U.: Generative programming: Methods, tools, and applications. *Addison-Wesley* (2000). 42, 43

[Cli82] CLIFFORD W. K.: On the classification of geometric algebras. In *Mathematical Papers* (1882), Tucker R., (Ed.), Macmillian, London, pp. 397–401. 6

[DDL02] DORST L., DORAN C., LASENBY J. (Eds.):. *Applications of Geometric Algebra in Computer Science and Engineering* (2002), Birkhäuser. 18, 39

[Dif02] DIFFER A.: The Clados home page. HTML document, 2002. Last visited 15. Sept. 2003. 18

[DM02] DORST L., MANN S.: Geometric algebra: a computational framework for geometrical applications (part i: algebra). *Computer Graphics and Application 22*, 3 (May/June 2002), 24–31. 8, 33

[Dor01] DORST L.: Honing geometric algebra for its use in the computer sciences. In *Geometric Computing with Clifford Algebra* (2001), Sommer G., (Ed.), Springer-Verlag. 6, 18

[Dor03] DORST L.: Classification and parametrization of blades in the conformal model of euclidean geometry. *In preparation, available from http://www.science.uva.nl/ga* (2003). 15, 40

[FBD01] FONTIJNE D., BOUMA T., DORST L.: Gaigen: A geometric algebra implementation generator. *Available at http://www.science.uva.nl/ga/gaigen* (2001). 18, 40, 42

[FD02] FONTIJNE D., DORST L.: 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). 6, 44, 45

[FD03] FONTIJNE D., DORST L.: Modeling 3D euclidean geometry. *IEEE Computer Graphics and Applications 23*, 2 (MarchApril 2003), 68–78. 44

[Fle] FLECKENSTEIN P.: C++ template classes for geometric algebras. *Available at http://www.nklein.com/products/geoma*. 40, 42

[GLD93] GULL S. F., LASENBY A. N., DORAN C. J. L.: Imaginary numbers are not real – the geometric algebra of space time. *Found. Phys. 23*, 9 (1993), 1175. 18

[GM91] GILBERT J. E., MURRAY M. A. M.: *Clifford algebras and Dirac operators in harmonic analysis*. Cambridge University Press, 1991. 18

[Hes86] HESTENES D.: *New Foundations for Classical Mechanics*. Dordrecht, 1986. 18

[HS84] HESTENES D., SOBCZYK G.: *Clifford Algebra to Geometric Calculus: A Unified Language for Mathematics and Physics*. Dordrecht, 1984. 6, 18

[HZ91] HESTENES D., ZIEGLER R.: Projective Geometry with Clifford Algebra. *Acta Applicandae Mathematicae 23* (1991), 25–63. 6, 18

[Lan02] LANDRY W.: Implementing a high performance tensor library. *http://www.oonumerics.org/FTensor* (2002). 43

[Leo02] LEOPARDI P.: The GluCat home page. HTML document, 2002. Last visited 15. Sept. 2003. 18

[LFLD98] LASENBY J., FITZGERALD W. J., LASENBY A., DORAN C.: New geometric methods for computer vision: An application to structure and motion estimation. *International Journal of Computer Vision 3*, 26 (1998), 191–213. 6, 18

[LHR01] LI H., HESTENES D., ROCKWOOD A.: Gen-

© The Eurographics Association 2004.

eralized homogeneous coordinates for computational geometry. *Geometric Computing with Clifford Algebra* (2001), 27–59. 6, 33

[LL98] LASENBY J., LASENBY A. N.: Estimating Tensors for Matching over Multiple Views. *Phil. Trans. R. Soc. Lond. A 356*, 1740 (1998), 1267–1282. 18, 31

[Lou87] LOUNESTO P.: The CLICAL home page. HTML document, 1987. Last visited 15. Sept. 2003. 18

[Lou97] LOUNESTO P.: *Clifford Algebras and Spinors*. Cambridge University Press, 1997. 18

[Mac99] MACDONALD A.: Elementary Construction of the Geometric Algebra. In *Proceedings 5th International Conference on Clifford Algebras and their Applications in Mathematical Physics* (1999). To be published. 18

[MD02] MANN S., DORST L.: Geometric algebra: a computational framework for geometrical applications (part ii: applications). *Computer Graphics and Application 22*, 4 (July/August 2002), 58–67. 8

[MDB01] MANN S., DORST L., BOUMA T.: The making of GABLE, a geometric algebra learning environment in matlab. *Geometric Algebra with Applications in Science and Engineering* (2001), 491–511. 18, 33, 40, 41

[MR02] MANN S., ROCKWOOD A.: Computing singularities of 3d vector fields with geometric algebra. *Proceedings of the conference on Visualization '02* (2002), 283–290. 41

[Per00] PERWASS C.: *Applications of Geometric Algebra in Computer Vision*. PhD thesis, Cambridge Universtiy, 2000. 18, 31

[Per02] PERWASS C.: The CLU home page. HTML document, 2002. Last visited 28. May 2004. 18, 33, 40, 42

[PGS03] PERWASS C., GEBKEN C., SOMMER G.: Implementation of a clifford algebra co-processor design on a field programmable gate array. *CLIFFORD ALGEBRAS: Application to Mathematics, Physics, and Engineering* (2003), 561–575. 42, 44

[PH03] PERWASS C., HILDENBRAND D.: *Aspects of Geometric Algebra in Euclidean, Projective and Conformal Space*. Technical Report Number 0310, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, September 2003. 18, 30, 33

[PL98] PERWASS C., LASENBY J.: A Geometric Analysis of the Trifocal Tensor. In *Image and Vision Computing New Zealand, IVCNZ'98, Proceedings* (1998), R. Klette G. Gimel'farb R. K., (Ed.), The University of Auckland, pp. 157–162. 31

[PL01] PERWASS C., LASENBY J.: A Unified Description of Multiple View Geometry. In *Geometric Computing with Clifford Algebra* (2001), Sommer G., (Ed.), Springer-Verlag. 6, 18, 31

[Por95] PORTEOUS I. R.: *Clifford Algebras and the Classical Groups*. Cambridge University Press, 1995. 18

[Rie93] RIESZ M.: *Clifford Numbers and Spinors*. Kluwer Academic Publishers, 1993. 18

[Sel96] SELIG J. M.: *Geometrical Methods in Robotics*. Springer–Verlag, 1996. 18

[Som01] SOMMER G. (Ed.): *Geometric Computing with Clifford Algebra*. Springer Verlag, 2001. 18

[Sut03] SUTER J.: Clifford. *Used to be available at http://www.jaapsuter.com* (2003). 40, 43

[VJ02] VANDEVOORDE D., JOSUTTIS N. M.: C++ templates: The complete guide. *Addison-Wesley* (2002). 43

[ZD03] ZAHARIA M., DORST L.: Interface specification and implementation internals of a program module for geometric algebra. *Accepted for Journal of Logic and Algebraic Programming* (2003). todo todo todo see website. 40, 42

© The Eurographics Association 2004.

26. Biography

26.1. L. Dorst

Leo Dorst
Intelligent Autonomous Systems
Informatics Institute
Faculty of Sciences University of Amsterdam
Amsterdam, The Netherlands
email: leo@science.uva.nl
Phone : 31-20-525 7511
Fax : 31-20-525 7490

Leo Dorst is an assistant professor at the Informatics Institute of the University of Amsterdam. His research interests include geometric algebra and its applications to computer science. He has an MSc and PhD in the applied physics of computer vision.

Previously he presented lectures at various specialist conferences on geometric algebra and co-presented geometric algebra courses at SIGGRAPH 2000 and 2001. He recently co-published three tutorials on geometric algebra in IEEE Computer Graphics and Applications.

26.2. D. Fontijne

Daniel Fontijne
Intelligent Autonomous Systems Informatics
Institute Faculty of Sciences
University of Amsterdam
Amsterdam, The Netherlands
email: fontijne@science.uva.nl
Phone : 31-20-525 7511
Fax : 31-20-525 7490

Daniel Fontijne is a scientific programmer and PhD student at the University of Amsterdam. His main goal is to integrate geometric algebra into various programming environments and languages in ways that are both efficient and usable. He has earned an MSc with distinction in Artificial Intelligence. Previously, he presented a course on geometric algebra at the Game Developers Conference 2003 and published a tutorial on geometric algebra in IEEE Computer Graphics and Applications.

26.3. D. Hildenbrand

Dietmar Hildenbrand
University of Technology Darmstadt, Germany
Fraunhoferstr. 5
64283 Darmstadt
email:dietmar.hildenbrand@gris.informatik.tu-darmstadt.de
Phone: +49 6151 155 667
Fax: +49 6151 155 669
URL:http://www.dgm.informatik.tu-darmstadt.de/staff/dietmar/

Dietmar Hildenbrand is a researcher and PhD student with the Interactive Graphics Systems group in Darmstadt, Germany. He holds a Masters degree in Computer Science. His main research interest lies in describing animations with the help of Geometric Algebra. He prepared a tutorial on Geometric Algebra together with C. Perwass for the DAGM 2003 conference.

26.4. C. Perwass

Dr. Christian Perwass,
university of Kiel, Germany
Institut fuer Informatik,
Olshausenstr. 40,
24098 Kiel, Germany
email: christian@perwass.de
Tel.: +49 431 880-7548,
Fax: +49 431 880-7550,
URL : www.perwass.de

Christian Perwass is a post-doctoral researcher and assistant teacher at the Cognitive Systems Group of the University of Kiel, Germany. His main research interest lies in the application of Geometric Algebra to Computer Vision and related fields. He holds a MSci degree in Physics and a PhD in applications of Geometric Algebra in Computer Vision.

He regularly gives a course on Geometric Algebra at the University of Kiel, and has presented a tutorial on Geometric Algebra together with D. Hildenbrand at the DAGM 2003 conference. He recently published an award-winning paper on the application of Geometric Algebra in artificial neural networks and is the author of a visualization and teaching software program for Geometric Algebra.

© The Eurographics Association 2004.