Using Geometric Algebra to Interactively Model the Geometry of Euclidean and non-Euclidean Spaces

Hugh Vincent

A thesis submitted to Middlesex University in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

February 2007

Abstract

This research interprets and develops the 'conformal model of space' in a way appropriate for a graphics developer interested in the design of interactive software for exploring 2-dimensional non-Euclidean spaces.

The conformal model of space extends the standard projective model - instead of adding just one extra dimension to standard Euclidean space, a second one is added that results in a Minkowski space similar to that of relativistic spacetime. Also, standard matrix algebra is replaced by geometric (i.e. Clifford) algebra.

The key advantage of the conformal model is that both Euclidean and non-Euclidean spaces are accommodated within it. Transformations in conformal space are generated by bivectors which are special elements of the geometric algebra. These induce geometric transformations in the embedded non-Euclidean spaces. However, the relationship between the bivector generated transformations of the Minkowski modelling space and the geometric transformations they induce is extremely obscure.

This thesis provides new analytical tools for determining the nature of this relationship. Their derivation was motivated by the need to successfully solve key implementation problems relating to navigation and in-scene mouse interaction.

The analytic approaches developed not only successfully solved these problems but pointed the way to implementing other unplanned features. These include facilities for dynamically altering on-screen geometry as well as using multiple viewports to allow the user to interact with the same objects embedded in different geometries. These new analytical approaches could be powerful tools for solving future and as yet unforeseen implementation problems.

Dedicated to my wife Maggie for her unwavering encouragement and support.

Thanks

I am indebted to Dr Tony Crilly for his steadying hand and encouragement, especially at times when direction seemed lost, and to Dr Matt Jones for keeping a sharp mathematical eye over the proceedings. Thanks also to Professor Huw Jones who helped get the show on the road and to the many others at Middlesex University who helped make the experience so worthwhile.

Contents

Introduction

The classical homogeneous model of space

Existing computer packages for exploring aspects of geometry tend to use a standard matrix-based approach to computer graphics. Generally, they are therefore limited to standard linear projective space. This research is motivated by growing research into the 'conformal model of space' which enables non-Euclidean geometries to be represented on screen. The model uses geometric algebra rather than matrix algebra.

The standard matrix-based approach effectively adds an extra dimension to a base space. The use of 4D matrices to implement 3D graphics is a well established technique. Theoretically, the way that the 3D space is modelled in 4D means that 3D non-linear maps such as translations and perspective scalings are 'linearised' in the 4D model. Equally, any linear map in the 4D 'homogeneous' model has its possibly non-linear 'real world' equivalent map in the 3D base space. The relationship between 'canonical' linear transformations in the 4D 'homogeneous' model and actual transformations in the 3D base space are generally well understood.

The conformal model of space

In contrast, the conformal model of space adds two extra dimensions to the base space to turn it into a Minkowski space similar to that of relativistic spacetime. The geometric algebra associated with this Minkowski space provides the transformations, loosely corresponding to the way that matrix algebra provides the transformations for the standard model. The key advantage of the conformal model is that both Euclidean and non-Euclidean spaces are accommodated within it, again rather in the way that the standard 4D homogeneous model accommodates the Euclidean geometry of standard 3D space. How this is achieved is explored in Chapter 1.

A disadvantage of the conformal model is that the relationship between transformations of the higher dimensional Minkowski modelling space and their equivalent geometric transformations induced in the various embedded Euclidean and non-Euclidean geometries is extremely obscure. The second half of this thesis develops new tools for analysing this relationship and uses them to solve various implementation problems.

Geometric algebra

Geometric algebra seems to be the new name for Clifford algebra. In the 1870's the English mathematician William Kingdon Clifford (1845-1879) translated Hermann Grassmann's work on linear algebra for English speaking audiences. As well, he developed his own abstract 'Clifford algebra' and applied it to the study of geometry. Currently, this algebra has important applications in many aspects of theoretical physics including cosmology and quantum theory.

Geometric or Clifford algebra has the ability to unify apparently disparate topics in a geometric-oriented way. That is its strength in physics, and is a feature when used in a computational context. For example, classical complex numbers, quarternions and the 'motor algebra' used in robotics are all Clifford algebras. Clifford algebras are also beginning to play a role in new approaches to computer visualisation, signal processing and quantum computing.

A Clifford algebra is built up from a vector space by introducing an extra 'geometric' or Clifford product into the vector space. This gives rise to new 'abstract' elements which extend the original vector space, similar to the way that the imaginary number i extends the real numbers. Generally, the extra added elements are more numerous and are 'graded' into grade 2 elements (bivectors), grade 3 elements (trivectors) and so on. Standard vectors are grade 1 and scalars are grade zero. Differently graded elements can be additively combined to form so called multivectors, rather in the way that real and imaginary numbers can be added to form complex numbers.

From the geometric product it is possible to derive two further products, the inner or 'dot' and outer or 'wedge' product. For vectors, the former is loosely analogous to the familiar dot product and yields a real number result. It is used to determine the 'signature' of a vector - the sign of its square, formed from the dot product. Vectors may have negative, positive or zero signature - interpreting the dot product geometrically can therefore be problematic.

The other derived product, the wedge product, when applied to a series of vectors (grade 1 elements), effectively represents the linear subspace spanned by them. Such a product is called a 'blade', as opposed to the wedge product formed from more generic multivectors.

The base-vectors of the original vector space may be of any signature and the number of positive, negative and zero signature base-vectors uniquely determine the associated Clifford algebra. This gives rise to a succinct notation to describe a Clifford algebra: Cl(p,q,r) means the Clifford algebra built on a vector space of dimension p + q + r with p base-vectors having positive signature, q having negative signature and r with zero signature. In this notation, trailing zero values may be omitted, for example Cl(p,q) = Cl(p,q,0) and Cl(p) = Cl(p,0,0).

The quarternions are the even sub-algebra of the Clifford algebra Cl(3). (The even sub-algebra is the subset of elements of even grade.) The 'motor algebra' of robotics is the 'degenerate' even sub-algebra of Cl(3,0,1), degenerate because one of the base-vectors has zero signature. The complex numbers are the even sub-algebra of Cl(2).

Appendix D contains a further informal review of other key geometric algebra concepts used in this research.

The role of the null cone in the conformal model

The Conformal Model uses Cl(n+1,1) to model standard n-dimensional space, which is denoted by R^n. The extra two dimensions added to the original standard n-dimensional vector space have opposite signature. The resulting space, with its n+1 positive-signature base-vectors and a single negative-signature base-vector is known as a Minkowski space. Following the same indexing convention as that used for Clifford algebras, it is denoted $R^{n+1,1}$. The Clifford algebra Cl(n+1,1) of the conformal model extends the Minkowski space $R^{n+1,1}$.

A Minkowski space has a 'cone' of zero-signature vectors. This 'null cone' plays a key role in the conformal model - points in any of the embedded geometries are represented by vectors lying on this null cone.

Transformations in the conformal model

As indicated, canonical transformations in the Minkowski modelling space are achieved using the extended geometric algebra. In fact, the geometric product is used in a 2-sided 'sandwich' transformation using either a positive signature grade-1 vector s as follows

$$X \to sXs^{-1} \tag{0.1}$$

or the exponential of a grade-2 bivector B

$$X \to e^{B/2} X e^{-B/2}. \tag{0.2}$$

The first is loosely analogous to a reflection, the second to a rotation, though any analogy is fraught with difficulties, as the thesis will demonstrate.

Canonical transformations like these, implemented in the higher dimensional Minkowski space of the conformal model, induce transformations in the

embedded non-Euclidean geometries. As indicated, finding tools to analyse the relation between these turned out to be the main thrust of this research.

The research falls naturally into two halves. The first 4 chapters interpret the model, explore interactive design ideas, develop software tools for validation and identify key implementation challenges. These turned out to be the implementation of non-Euclidean navigation controls, centralised mouse-dragging and viewport transformations.

To implement navigation controls, a new computational approach was developed in Chapter 5 that utilised retained mode graphics and extended the notion of quarternions. This was used to implement navigation controls in 2D geometries, using Cl(2+1,1), as well as in 3D geometries, using Cl(3+1,1).

The remaining chapters consider the problems of centralised mouse dragging and viewport transformations. The new analytic tools that are developed are based on the notion of pencils (families) of circles. Remarkably, the resulting analysis has yet more historical resonance, linking the work of Poncelet (1788-1867) to the null or light cone of relativistic spacetime.

The nature of the induced non-Euclidean geometry

Finally, it should be pointed out that the use of the conformal model forces a certain interpretation of non-Euclidean geometry where non-Euclidean points are still points, but non-Euclidean lines may be lines or arcs of circles. The non-Euclidean line (i.e. line or arc) of shortest distance been two points is known as a d-line or geodesic.

The conformal model embeds a non-Euclidean geometry in two senses. For example, in the case of spherical geometry, the 4D Minkowski space contains a 3D sphere. However, the model also contains the means of stereographically projecting the surface of the sphere onto a flat screen. This projection is conformal - it preserves angles and therefore circles map to circles, and arcs to

arcs. In other words non-Euclidean lines and angles are preserved. Thus what appears on the screen is a stereographic projection of a sphere's surface - the screen shows a flat 'model' of spherical space. In the same way, for hyperbolic space, what is seen on the screen is a flat 'model' - a stereographic projection of a 'Minkowski' sphere of negative unit radius. This is none other than the familiar Poincaré disc in which non-Euclidean geodesic lines always meet the circular horizon orthogonally.

In these conformal models (i.e. stereographic projections) of non-Euclidean spaces, the fundamental transformation is inversion/reflection in a non-Euclidean line (i.e. line or arc). Rotations, translations and dilations can be constructed by combining an even number of inversions. The conformal model of space seems to lead naturally to viewing non-Euclidean geometries as being essentially inversive geometries. This is very different from the way that notions of non-Euclidean geometry arose historically.

In the higher dimensional Minkowski space, in so called 'conformal space', the vector generated 'reflection' expressed by the map 0.1 above induces an inversion in any embedded non-Euclidean geometry. The bivector generated transformation 0.2 above may induce a translation, rotation or dilation. It may even generate more exotic transformations. The thesis concentrates on bivector generated transformation.

Summary

This thesis provides new analytical tools for determining the nature of non-Euclidean geometric transformations induced by bivector generated transformations in conformal space. Their derivation was motivated by the need to successfully solve key implementation problems. It is felt that the analytic approach developed will be a powerful tool for solving future and as yet unforeseen implementation problems.

1 Picturing the Conformal Model of Space

1.1 Introduction

The conformal model of space requires at least four dimensions. To picture it in our 3D space means adopting dimension reducing conventions. A possible approach is described here based loosely on a selective synthesis and recasting of ideas expressed in various sources but particularly in [1], [2], [3] and [4].

The conformal model of space extends the standard projective model - instead of adding just one extra dimension to standard n-dimensional Euclidean space, a second one is added that results in a Minkowski space similar to that of relativistic spacetime. As if that were not radical enough, standard matrix algebra is replaced by geometric (i.e. Clifford) algebra.

The journal *IEEE Computer Graphics and Applications* has recently carried articles on geometric algebra by Stephen Mann and Leo Dorst where the model is referred to as the 'double homogeneous model' [5]. Daniel Fontijne and Leo Dorst have also reported comparative ray-tracing bench-mark tests based on the standard projective and conformal models [6]. Elsewhere, Eckhard Hitzer has described the development of an interactive 3D sketching package based on the model [7].

In the classical model of projective space, 'climbing in and out' of the extra fourth dimension is through the maps

$$(x, y, z) \rightarrow (x, y, z, 1)$$
$$(x', y', z', w') \rightarrow (x'/w', y'/w', z'/w')$$

Unfortunately the various ways to explain or visualise these established transformations do not extend easily to the conformal model, though perspective projection does play a role.

The way that the extra two dimensions are added to project points into the higher dimensional Minkowski space can seem bizarre, especially when considered purely algebraically. Fortunately, there are a number of ways to view this 'dimension climbing', and the model as a whole, geometrically.

One way into the conformal model is to utilise the fact that it echoes familiar ideas associated with classical conic sections. In essence, the model provides representations of Euclidean, spherical and hyperbolic space as sections of the null cone of a Minkowski space, see figure 1.1 (A brief introduction to Minkowski space will be given later.)

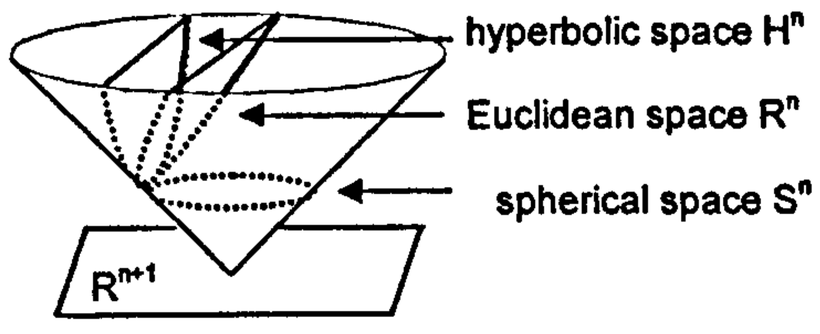


Figure 1.1 Euclidean and non-Euclidean spaces depicted as sections of the null cone.

However, the picture can be misleading since the three dimensions available to us for visualisation are usually insufficient, and such pictures attempt to visualise Minkowski space in a 2D drawing-space that is essentially Euclidean. Despite this, visualisation can provide a feel for how the model works - it is a question of interpretation.

The idea (and the original significance of the name 'conformal model') seems to have emerged from spacetime physics that predates the re-emergence of Clifford algebra. The latter gives it its power, yet can get in the way of understanding the geometry. This was one motivation for presenting the ideas visually. The other was to reconcile and make sense of current geometric algebra based literature.

To illustrate the computational side of the model, a short example shows how the same 5-step core algorithm can be used to generate grids of circles in the various models of non-Euclidean space. To implement it, all that is needed is software to handle vector dot products and to draw circles.

However, the subsequent example needs a very basic 'Clifford processor' or geometric algebra 'engine'. The one developed will generate the Cayley table (i.e. multiplication table) for the geometric algebra of any signature and handle all the key associated products. It also includes simple routines for drawing arcs and circles. It is described in chapter 3 and is used in both examples.

1.2 Visualising spherical space

One way to describe the conformal model makes use of stereographic projections between flat and spherical space. This dictates the starting point – the standard model of n-dimensional spherical space, the unit sphere

$$S^n = \left\{ \mathbf{x} \in R^{n+1} \mid \mathbf{x}^2 = 1 \right\}$$

The 'dot' or inner product $\mathbf{x} \cdot \mathbf{x} = \mathbf{x}^2$ is constructed in the ambient space R^{n+1}.

Our drawing-space is sufficient to faithfully represent the sphere S^2, the circle S^1 and the two-point 'circle' S^0 embedded respectively in their ambient spaces R^3, R^2 and R^1, see figure 1.2.

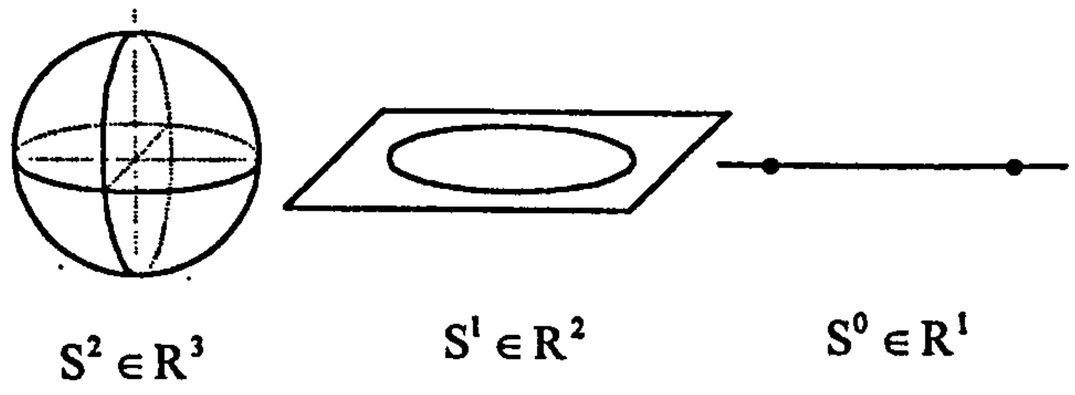


Figure 1.2 Spheres of 2, 1 and zero dimension depicted in their respective ambient spaces.

Because of symmetry, the same drawings can represent S^n in R^{n+1} for any n. In particular figure 1.3 shows three depictions of S^2 in its ambient space R^3

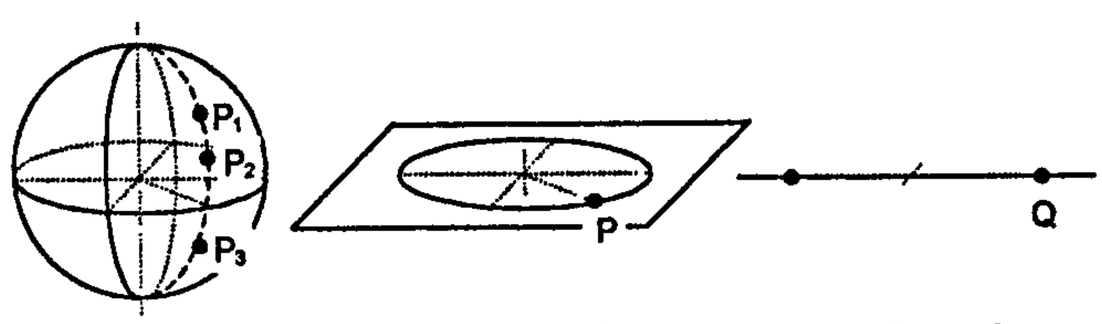


Figure 1.3 A 2 dimensional sphere and its ambient space depicted in 3D, 2D and 1D drawing space.

Only the first picture is a one-to-one representation. In relation to the second, and using the Earth as an analogy, the single point P on the circle represents all the points on the earth-sphere S^2 with the same longitude (for example P_1, P_2 and P_3). In other words, the position of P on the drawn plane conveys information about longitude but says nothing about latitude. In a similar way the single point Q in the third one-dimensional depiction of S^2 represents all the points on the right half of the earth-sphere. The other point represents the other hemisphere.

The second and third drawings in figure 1.3 'claw back' drawing dimensions but at the price of lost information. However, this lost information can be conceptualised and re-introduced locally in interesting dynamic ways.

For example, figure 1.4 shows a 'control lever' attached locally to point P for setting the missing latitude angle read off against the attached 'protractor'. Dragging point P around the circle sets the longitude and also drags the lever and protractor with it so that they remain attached to P. Rotating the end of the lever sets the latitude.

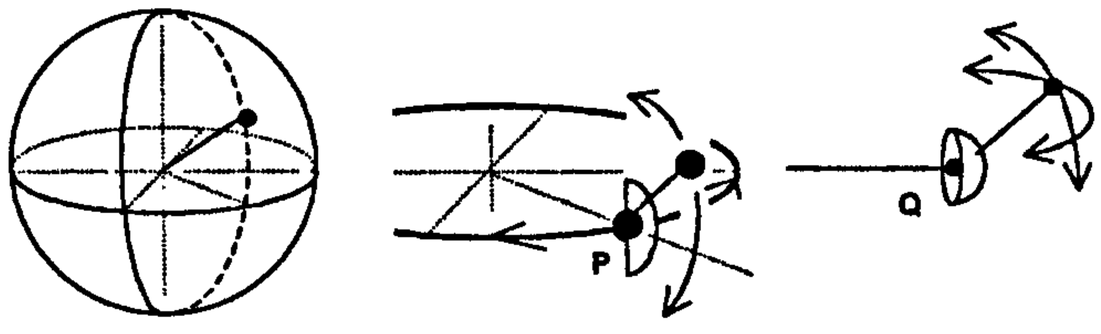


Figure 1.4 A 2 -sphere depicted in 3, 2 and 1D drawing space.
To compensate for lost information, the point P in the 2D drawing space has a 'latitude lever' attached. The point Q in the 1D drawing space has a 'joystick' attached to set both latitude and longitude.

Each point P on the circle has its own potential latitude-lever for selecting the value of missing information caused by the loss of a global dimension - the lever locally represents the missing dimension. Put another way, although the picture depicts P as being a zero-dimensional point, it is actually 1 dimensional. In the same way, point Q is actually 2 dimensional and its equivalent embedded 'control lever' would be a 'joy stick' capable of setting both latitude and longitude for the selected hemisphere, see figure 1.4.

Dimension saving techniques like these allow S^n in R^{n+1} to be depicted as a circle on a plane, allowing one extra dimension for depicting the added Minkowski dimension. However, each point on the drawn 2D circle representation of S^n represents an n-1 dimensional hemisphere, or semicircle when n=2. The situation is similar with points not on the circle. For example, a single point P, drawn inside the 2D circle representing S^n, which is at a distance $r<1$ from the centre of the circle, could represent actual points P_1, P_2, P_3 in R^{n+1} which are at the same distance r from the origin. In the case n = 2, these points would lie on a semicircle with the same longitude angle as P. Similar comments would apply to a point Q drawn outside the 2D circle representing S^n, see figure 1.5.

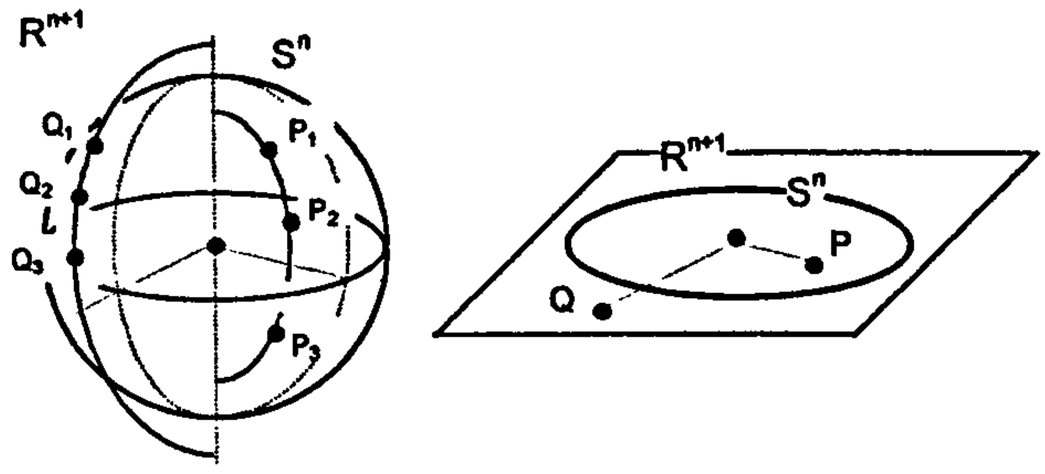


Figure 1.5 Points inside and outside an n-sphere depicted in 3D and 2D drawing space.

The process of turning the 3D picture into 2D can be viewed as follows: if the 3D drawing space is thought of as consisting of lines through the centre, then the 2D representation is obtained by rotating each line in a vertical plane so that it becomes horizontal. This 'fanning out' of the rays so they all become flat effectively removes the vertical dimension, but does so in a way that preserves the distance of a point from the centre. The process is of course very different from standard parallel projection.

1.3 Visualising Minkowski space

Minkowski space is closely associated with Einstein's Special Theory of Relativity which combines 3D Euclidean space with time into a four dimensional 'spacetime' of 'events'. Two observers moving at different speeds relative to each other do not necessarily produce identical measurements for the same event, though the measurements are related through the Lorentz transformation. However, when measuring the speed of a photon, different observers will always produce the same value for the speed of light c – the speed of light is the same for all observers.

If a photon of light is observed to be emitted at time 0 from the origin then after time t its observed position (x,y,z) is related to t by

$$x^2 + y^2 + z^2 = c^2 t^2 .$$

Traditionally, units of space and time are chosen so that c has unit value, in which case the relationship becomes

$$x^2 + y^2 + z^2 - t^2 = 0.$$

There are a number of ways of viewing this equation. In one sense, it represents the expanding wave front of photons emitted from the origin at the same time. On the other hand, if the spacetime of the observer is given co-ordinates (x, y, z, it), where $i = \sqrt{-1}$, then this equation states that the modulus of the 4D vector representing the path of the photon is null – the time co-ordinate is made imaginary for this interpretation to work.

The null vectors representing radiating photons emitted from the origin at the same time form a cone in the observer's spacetime known as the null cone or light cone. By restricting motion to the plane R^2, this can be visually depicted, see figure 1.6.

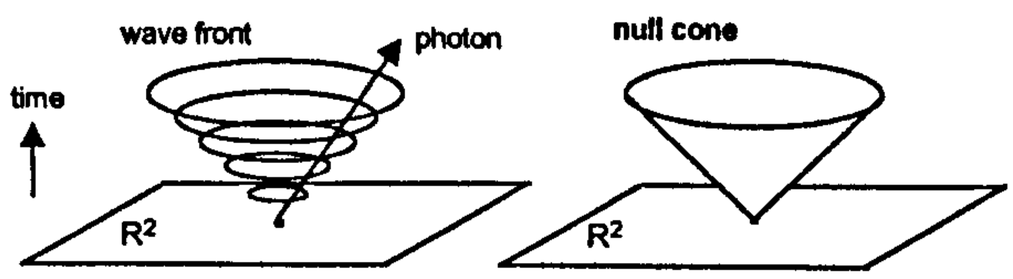


Figure 1.6 Null cone as the locus of a wave front of light.

Minkowski space bypasses the need for an imaginary time co-ordinate by the simple expedient of decreeing that the extra time dimension has negative signature, i.e. that spacetime has an orthogonal bases $\{e_1, e_2, e_3, e_4\}$ where $e_1^2 = e_2^2 = e_3^2 = +1$ and $e_4^2 = -1$. These values are used in the calculation of dot products. To reflect the signatures of the basis set, this interpretation of spacetime is denoted by $R^{3,1}$. Other approaches to spacetime use $R^{1,3}$.

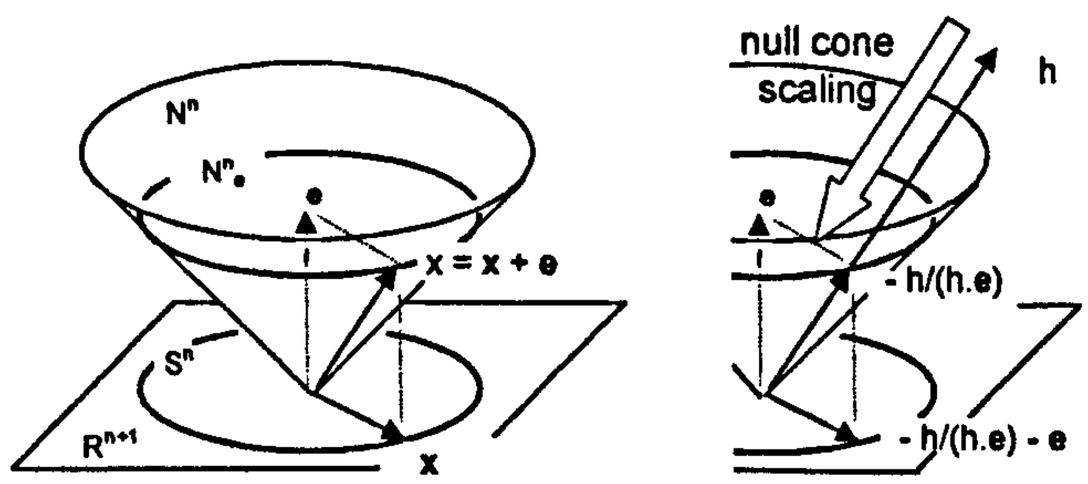
1.4 Lifting S^n onto the null cone of the Minkowski space $R^{n+1,1}$

The Minkowski space $R^{n+1,1}$ can be constructed by augmenting an orthonormal basis set of R^{n+1} with an extra unit vector **e** of negative signature ($\mathbf{e}^2 = -1$) orthogonal to each of the basis vectors R^{n+1}. By linearity, this extra 'Minkowski' base vector will be normal to any vector **x** constructed in R^{n+1}, i.e. $\mathbf{x} \cdot \mathbf{e} = 0$.

The null cone N^n of $R^{n+1,1}$ is the set of null vectors

$$N^n = \left\{ x \in R^{n+1,1} \mid x^2 = 0 \right\},$$

see figure 1.7a.



(a) (b)

Figure 1.7 (a) A vector **x** on an n-sphere S^n (depicted as a circle on R^{n+1} drawn as a plane). The vector $x = \mathbf{x} + \mathbf{e}$ is its lifting onto the set N^n_e (depicted as a circle on the null cone).

(b) A vector h on the null cone. The scaled vector -h/(h.e) lies on the 'circle' N^n_e. Subtracting **e** drops the vector onto the 'circle' S^n.

Any point **x** in S^n can then be mapped or 'lifted' onto the null cone N^n by the mapping

$$j : \mathbf{x} \rightarrow \mathbf{x} + \mathbf{e} = x.$$

The mapping is shown in figure 1.7a. That the result x lies on the null cone is easily verified since

$$\mathbf{x}^2 = (\mathbf{x}+\mathbf{e})^2 = \mathbf{x}^2 + 2\mathbf{x}\cdot\mathbf{e} + \mathbf{e}^2 = 1 + 0 - 1 = 0.$$

If x is a point on the null cone obtained in this way then

$$\mathbf{x}\cdot\mathbf{e} = (\mathbf{x}+\mathbf{e})\cdot\mathbf{e} = \mathbf{x}\cdot\mathbf{e} + \mathbf{e}^2 = 0 - 1 = -1.$$

Thus j maps $S^n \in R^{n+1}$ into the set

$$N_e^n = \left\{ x \in R^{n+1,1} \mid x^2 = 0, \quad x\cdot\mathbf{e} = -1 \right\}.$$

Here bold typeface is used for vectors in S^n and for the special Minkowski base vector **e**. Normal typeface is used for all other 'Minkowski vectors', i.e. those with a non-zero **e** component.

The set N_e^n provides a representation of the spherical space S^n in the conformal model. In a certain sense the representation is projective since any vector h on the null cone which does not lie in N_e^n can be scaled so that it does. The scaling transformation is

$$h \rightarrow -h/(h\cdot\mathbf{e}).$$

The result lies in N_e^n since

$$(-h/(h\cdot\mathbf{e}))\cdot\mathbf{e} = -(h\cdot\mathbf{e})/(h\cdot\mathbf{e}) = -1.$$

Figure 1.7b shows this scaling diagrammatically.

This 'null cone scaling' is highly significant. Figure 1.1 at the beginning of the chapter hints at the fact that other spaces embedded in $R^{n+1,1}$ could also be mapped to the null cone. Thus, by using suitable null cone scaling, a single projective vector on the null cone could represent a different vector in each of these spaces. Or, a vector in one of these spaces could be mapped to the null cone, then scaled so that it represented another vector in another space, thus providing mappings between spaces. In particular, stereographic projections between spaces of various types, or their models, can be represented as simple scalings of the null cone in conformal space. This theme will be returned to in section 1.7.

The dot product in conformal space can be used to create expressions for the metrics in the base space. Identical conformal points have zero dot product, reflecting the fact that the distance between them in the base space is zero. In the present case, the dot product of two different conformal null vectors representing points in S^2 reduces to

$$\mathbf{x}\cdot\mathbf{y} = (\mathbf{x}+\mathbf{e})\cdot(\mathbf{y}+\mathbf{e}) = \mathbf{x}\cdot\mathbf{y} - 1 = \cos(\theta) - 1,$$

where θ is the spherical distance between **x** and **y**. Thus the dot product formed on vectors in conformal space encodes the spherical distance between the points in the base space.

1.5 The conformal representation of circles in S^2

Vectors in conformal space that do not lie on the null cone can also be significant. For example, if **c** is a point on S^2, then the conformal vector

$$\mathbf{s} = \mathbf{c} + k\mathbf{e} \qquad 0<k<1$$

represents a circle in S^2 centred at the point **c**, see figure 1.8.

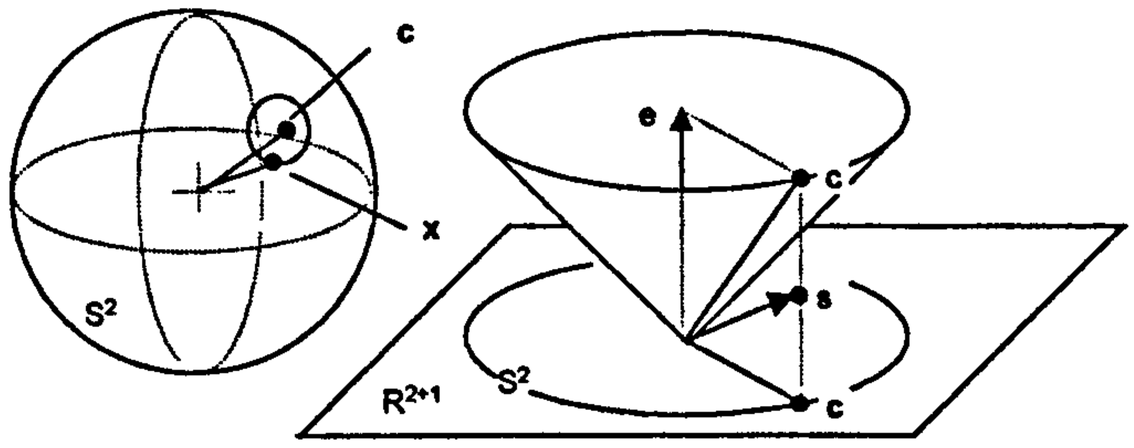


Figure 1.8 A spherical circle with centre **c** on the sphere S^2 and its representation as a vector s in conformal space.

Points of the circle are represented in conformal space by null vectors x which are also 'perpendicular' to the given vector s, i.e. that satisfy

$$\mathbf{x} \cdot \mathbf{s} = 0.$$

If $\mathbf{x} \in S^2$ is the vector corresponding to $x \in R^{3,1}$, then the condition translates as

$$\mathbf{x} \cdot \mathbf{s} = 0$$

$$(\mathbf{x} + \mathbf{e}) \cdot (\mathbf{c} + k\mathbf{e}) = 0$$

$$\mathbf{x} \cdot \mathbf{c} + \mathbf{e} \cdot \mathbf{c} + k\mathbf{x} \cdot \mathbf{e} + k\mathbf{e} \cdot \mathbf{e} = 0$$

$$\mathbf{x} \cdot \mathbf{c} + 0 + 0 - k = 0$$

$$\mathbf{x} \cdot \mathbf{c} = k\,.$$

Thus the vectors **x** in S^2 form a circle with centre **c** where k is the component of **x** in the direction **c**. If k is close to 0, this component is small implying that the **x** vectors are nearly perpendicular to **c**, so the circle is nearly a great circle. In the extreme, when k = 0, the circle is a great circle. On the other hand, if k is approximately 1, the **x** vectors are close to **c** and so define a small circle that becomes a single point as k approaches 1.

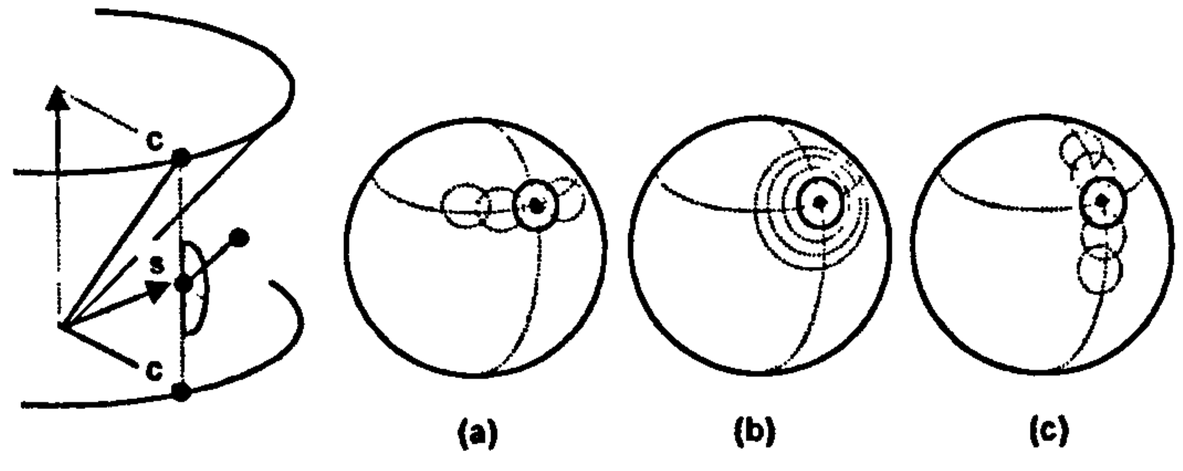


Figure 1.9 Controlling (a) the longitude, (b) the size and (c) the latitude of a spherical circle.

The effect on the circle of altering the conformal vector s in the Minkowski space is depicted in figure 1.9:

(a) Rotating the conformal point c about the vertical axis, or equivalently its projection **c** onto the R^2 plane, changes the longitude of the centre of the circle.

(b) Moving the conformal point s vertically changes the value of k and therefore the size of the circle.

(c) Rotating the 'latitude-lever' attached to s changes the latitude of the circle.

The single conformal vector s encodes information about the spherical circle - the scalar k is the Minkowski component and is directly related to the radius of the spherical circle. The remaining part **c** exists in spherical space and is the centre of the circle.

In fact, vectors in the Minkowski space $R^{3,1}$ perpendicular to s (i.e. that satisfy the condition $x \cdot s = 0$) constitute a 3D hyperplane perpendicular to s. Thus the actual circle can also be represented in conformal space by the intersection of this hyperplane with the null cone. If the circle is a great circle, its representative vector s is perpendicular to **e**, so the alternative representation, the hyperplane perpendicular to s, contains **e**. However, this is best approached through the algebra of blades where, in $R^{3,1}$, the hyperplane is seen as being the dual of the vector s.

All this is illustrative of a common feature of the model – single entities in conformal space, or its associated geometric algebra, can often encode complex geometric entities in the base space, often in such a way that algebraic operations are geometrically meaningful. Also, there are sometimes significant dual representations with interesting interactions, for example a circle can be represented conformally by either a vector or its dual, a trivector blade, see section 1.13. This thesis uses mainly the former representation.

1.6 Stereographic projection of spherical space S^n to Euclidean space R^n

Angle preserving stereographic projections play a key role in the conformal model.

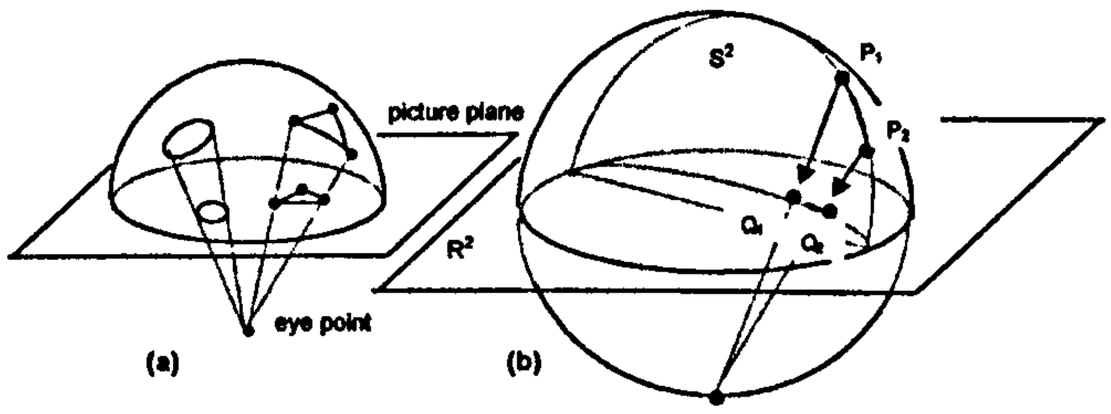


Figure 1.10 Stereographic projection of the northern hemisphere onto the equatorial plane using the south pole as the point of projection.

Figure 1.10a shows a spherical circle and triangle being projected via the south pole to the equatorial plane. Since both shapes are in the northern hemisphere, their projections lie inside the equatorial circle. Points in the southern hemisphere would project outside the circle and, as a point approached the south pole, its projection would lie progressively further from the centre. In this sense, the south pole could be thought of as representing infinity.

Figure 1.10b shows the shortest path between two spherical points P_1 and P_2 which is a segment of the great circle through them. The latter meets the equator at diametrically opposed antipodal points, so the shortest path between the projection Q_1 and Q_2 on the plane lies on a circle that also meets the

equator at antipodal points. The assumption here is that geodesics map to geodesics.

The interior of the equatorial circle can therefore provide a flat, potentially screen-based model of the north hemisphere of spherical space that is sometimes referred to as the 'hemisphere' model. The way that triangles and circles appear on this disc or 2D 'ball' is what the eye would see if it were looking at the shapes from the south pole, assuming they were painted on a transparent globe. Straight lines in this disc model would correspond to the circles that passed through the south pole. Unless they were also great circles, their straight-line projections onto the disc would not represent shortest path routes. On this flat model of spherical space, the shortest distance between two points would only be a straight line if the points lay on the same diameter.

1.7 The projection of R^n onto the null cone of the Minkowski space $R^{n+1,1}$

The reverse of the stereographic projection provides a way of embedding R^n onto the null cone of $R^{n+1,1}$ by first projecting it onto S^n. The process can be pictured provided that a dimension is freed up to allow for the representation of the Minkowski dimension.

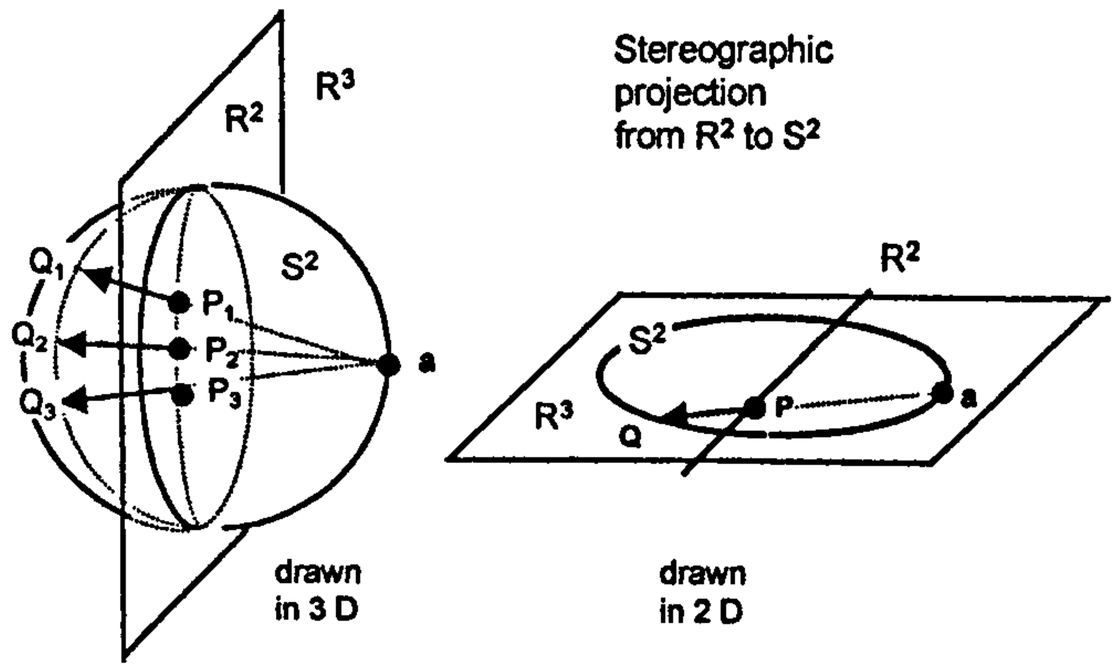


Figure 1.11 Stereographic projection of R^2 to S^2 depicted in 3D and 2D drawing space.

For example, figure 1.11 shows the stereographic projection from R^2 to S^2 depicted in both 3D and 2D. As before, the projected point Q in the second picture represents all points with the same longitude angle (e.g. Q_1, Q_2, Q_3). Equally, the point P represents the points in R^2 that map into the points represented by Q. As the latter lie on a great circle, the former lie on a circle that meets the unit circle in R^2 at antipodal points. The convention used here to regain a drawing dimension is therefore different from that used in figure 1.5. However, this does not undermine the intentions of the pictures.

As stated, the reverse stereographic projection of R^n to S^n can be used to project R^n onto the null cone. Generalising the drawing conventions above to higher dimensions, the process is depicted in figure 1.12.

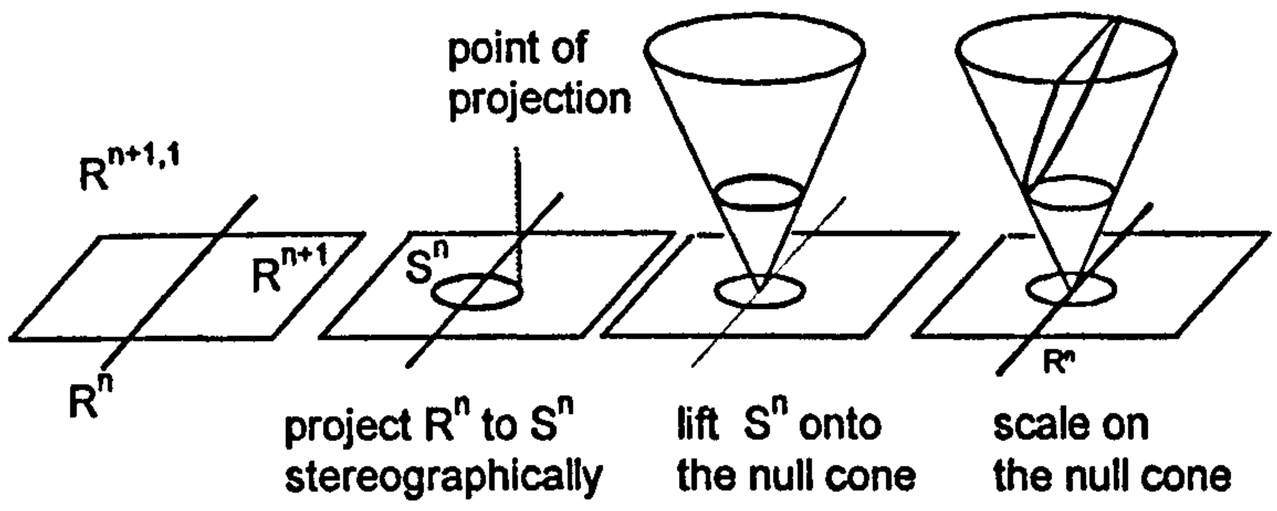


Figure 1.12 Projecting Euclidean space R^n onto the null cone of the Minkowski space $R^{n+1,1}$.

The last step of null cone scaling is necessary to ensure the representation 'has an appropriate metric' in the sense of wanting dot products formed in conformal space to represent the standard Euclidean metric in the base space in a simple way.

The mapping can be represented algebraically through dot products.

Suppose $\mathbf{x} \in R^n$ is mapped to $\mathbf{v} \in S^n$ via the polar projection point **a**, see figure 1.13.

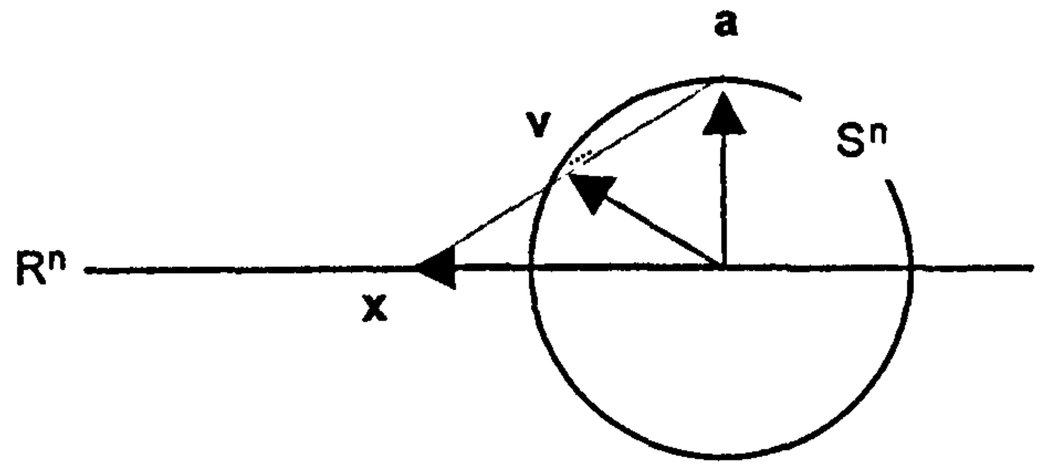


Figure 1.13 Stereographic projection of a spherical point **v** in S^n to a Euclidean point **x** in R^n using the spherical point **a** as the point of projection.

Then for some scalar $k \neq 0$,

$$\mathbf{v} = \mathbf{a} + k(\mathbf{x} - \mathbf{a}).$$

Squaring and utilising the fact that $\mathbf{v}^2 = \mathbf{a}^2 = 1$ and $\mathbf{x} \cdot \mathbf{a} = 0$

$$\mathbf{v}^2 = \mathbf{a}^2 + 2k\mathbf{a} \cdot (\mathbf{x} - \mathbf{a}) + k^2(\mathbf{x}^2 - 2\mathbf{x} \cdot \mathbf{a} + \mathbf{a}^2)$$

$$1 = 1 - 2k + k^2(\mathbf{x}^2 + 1)$$

$$k = 2/(\mathbf{x}^2 + 1).$$

Hence

$$\mathbf{v} = \{(\mathbf{x}^2 - 1)\mathbf{a} + 2\mathbf{x}\}/(\mathbf{x}^2 + 1).$$

This is lifted onto the null cone by adding **e** to get

$$\{(\mathbf{x}^2 - 1)\mathbf{a} + 2\mathbf{x}\}/(\mathbf{x}^2 + 1) + \mathbf{e}.$$

Scaling by $(\mathbf{x}^2 + 1)$ finally produces the mapping

$$f : \mathbf{x} \rightarrow \mathbf{x} = (\mathbf{x}^2 - 1)\mathbf{a} + 2\mathbf{x} + (\mathbf{x}^2 + 1)\mathbf{e}.$$

The dot product of two conformal vectors scaled in this way is then given by

$$\begin{aligned} & \mathbf{x}_1 \cdot \mathbf{x}_2 \\ = \; & \mathbf{f}(\mathbf{x}_1) \cdot \mathbf{f}(\mathbf{x}_2) \\ = \; & \{(\mathbf{x}_1^2 - 1)\mathbf{a} + 2\mathbf{x}_1 + (\mathbf{x}_1^2 + 1)\mathbf{e}\} \cdot \\ & \{(\mathbf{x}_2^2 - 1)\mathbf{a} + 2\mathbf{x}_2 + (\mathbf{x}_2^2 + 1)\mathbf{e}\} \\ = \; & -2(\mathbf{x}_1 - \mathbf{x}_2)^2 \qquad \text{since } \mathbf{a}^2 = 1,\ \mathbf{e}^2 = -1 \text{ and } \mathbf{x}_i \cdot \mathbf{a} = \mathbf{x}_i \cdot \mathbf{e} = 0. \end{aligned}$$

Thus the chosen null cone scaling not only simplifies the final mapping by removing denominators, it ensures that the resulting conformal dot product produces a result closely linked to the Euclidean metric in the base space.

It is easily verified that all conformal vectors x produced in this way satisfy the relation

$$\mathbf{x} \cdot (\mathbf{e} + \mathbf{a}) = -2.$$

Thus any vector h on the null cone can be scaled or normalised to the Euclidean representation by the mapping

$$\mathbf{h} \rightarrow -2\mathbf{h}/(\mathbf{h} \cdot \mathbf{n}), \qquad \text{where} \quad \mathbf{n} = \mathbf{e} + \mathbf{a}.$$

The conformal vector **n** lies on the null cone and represents the Euclidean point at infinity. This can be shown informally by considering the map

$$\mathbf{f} : \mathbf{x} \rightarrow (\mathbf{x}^2 - 1)\mathbf{a} + 2\mathbf{x} + (\mathbf{x}^2 + 1)\mathbf{e}.$$

As $\mathbf{x}^2$ becomes very large, the coefficients of **a** and **e** dominate and approach each other, leaving a result that approximates to a large scalar multiple of **a** + **e**. The result follows since **a** + **e** = **n** and the representation is homogeneous. On the other hand, f(**0**) = **e** – **a**, so the origin is represented conformally by the null vector **e** – **a**, see figure 1.14.

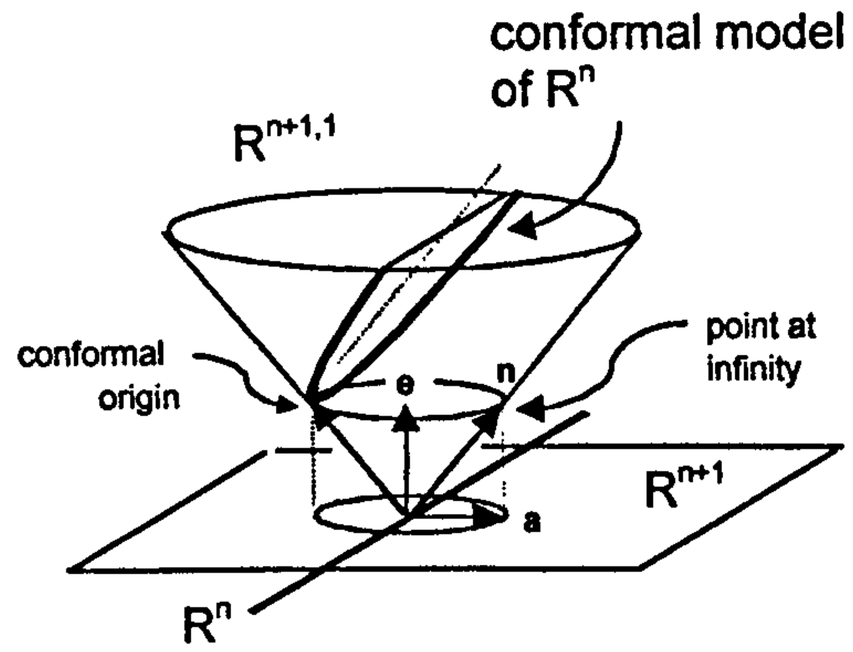


Figure 1.14 Conformal model of R^n showing vectors representing the origin and the point at infinity.

The vectors $\mathbf{e} + \mathbf{a}$ and $\mathbf{e} - \mathbf{a}$ can be derived from $\mathbf{a}$ and $-\mathbf{a}$ by lifting the latter from S^n onto the null cone. In S^n, $\mathbf{a}$ is the point of projection between R^n and S^n so is naturally associated with the point at infinity. Similarly, $-\mathbf{a}$ is the point of S^n that is associated with the origin of R^n, see figure 1.14.

If the conformal models of R^n and S^n are viewed as intersections of the null cone with the hyperplanes $x \cdot \mathbf{e} = -1$ and $x \cdot \mathbf{n} = -2$, then the null scone scaling corresponding to the stereographic projection of S^2 to R^2 has the appearance of being a rotation of the intersecting hyperplane, see figure 1.15.

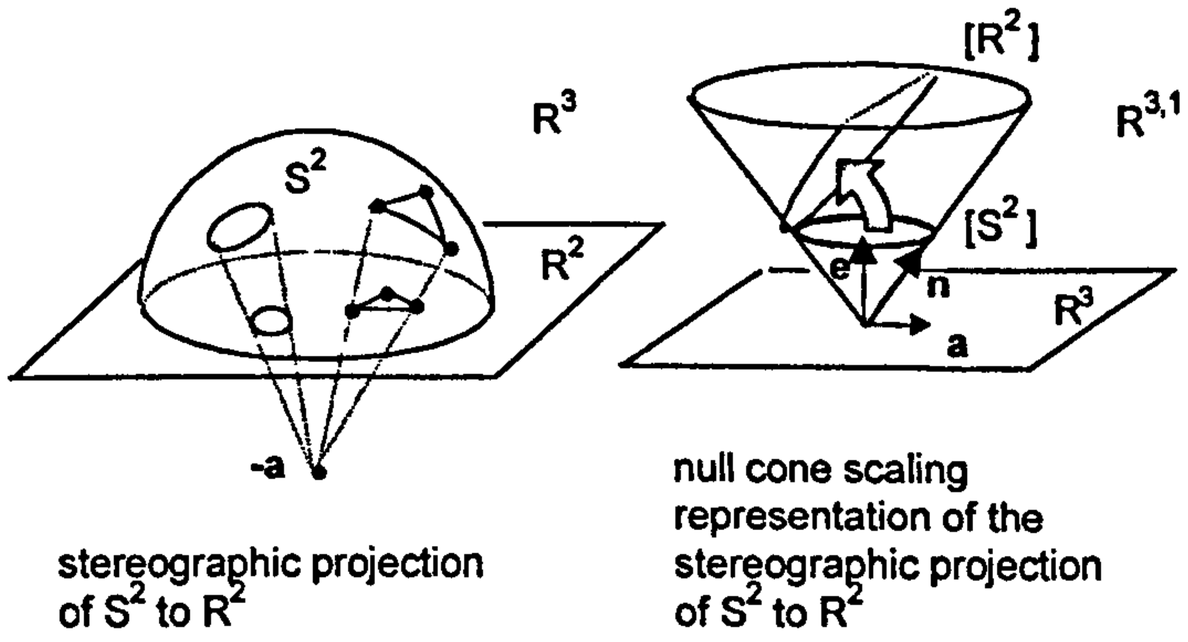


Figure 1.15 Stereographic projection between S^2 and R^2 depicted as a scaling on the null cone.

Figure 1.15 also shows that the orientations of the hyperplanes relative to the vector defining them appear to be inconsistent. In the picture, the hyperplane defined by $\mathbf{x} \cdot \mathbf{e} = -1$ is perpendicular to **e**, whereas the hyperplane $\mathbf{x} \cdot \mathbf{n} = -2$ is parallel to **n**. This is the price paid for attempting to draw Minkowski space in a Euclidean drawing space.

1.8 The conformal representation of circles in R^2

The representations of the two spaces S^n and R^n is substantially determined by the vectors **e** and $\mathbf{n} = \mathbf{e} + \mathbf{a}$ respectively. These vectors also play a determining role in the representation of circles in the two respective spaces. As discussed, circles in S^2 are represented by vectors of the form

$$\mathbf{s} = \mathbf{c} + k\mathbf{e}.$$

It is convenient to write this in the form

$$\begin{aligned} \mathbf{s} &= \mathbf{c} - \mathbf{e} + k\mathbf{e} \\ \mathbf{s} &= \mathbf{c} - (1 - k)\mathbf{e} \\ \mathbf{s} &= \mathbf{c} - r^2\mathbf{e} \qquad \text{where} \quad k = 1 - r^2. \end{aligned}$$

If s is perpendicular to **e**, i.e. the hyperplane perpendicular to s contains **e**, then the circle is a geodesic great circle.

Similarly, circles in R^n are represented by vectors of the form

$$\mathbf{s} = \mathbf{c} - r^2\mathbf{n}.$$

The verification is similar to that for the spherical space, though the algebra is more complicated. Assuming the conformal point x lies on the hyperplane perpendicular to the vector s that conformally defines the circle, then

$$\mathbf{x} \cdot \mathbf{s} = 0$$

$$\Rightarrow \mathbf{x}\cdot(\mathbf{c}-r^2\mathbf{n})=0$$

$$\Rightarrow ((\mathbf{x}^2-1)\mathbf{a}+2\mathbf{x}+(\mathbf{x}^2+1)\mathbf{e})\cdot$$
$$((\mathbf{c}^2-1)\mathbf{a}+2\mathbf{c}+(\mathbf{c}^2+1)\mathbf{e}-r^2(\mathbf{a}+\mathbf{e}))=0$$

$$\Rightarrow ((\mathbf{x}^2-1)\mathbf{a}+2\mathbf{x}+(\mathbf{x}^2+1)\mathbf{e})\cdot$$
$$((\mathbf{c}^2-1-r^2)\mathbf{a}+2\mathbf{c}+(\mathbf{c}^2+1-r^2)\mathbf{e})=0.$$

The multiplication table for the vectors in this product is

	x	**c**	**a**	**e**
x	$\mathbf{x}^2$	**x.c**	0	0
c	**x.c**	$\mathbf{c}^2$	0	0
a	0	0	1	0
e	0	0	0	-1

Applying the table gives

$$4\mathbf{x}\cdot\mathbf{c}+(\mathbf{x}^2-1)(\mathbf{c}^2-1-r^2)-(\mathbf{x}^2+1)(\mathbf{c}^2+1-r^2)=0\,,$$

which reduces to $(\mathbf{x}-\mathbf{c})^2=r^2$, defining a circle with centre **c** and radius **r**.

If s is perpendicular to **n**, the hyperplane perpendicular to s contains **n**, so the 'circle' represented by s contains the point at infinity and is therefore a geodesic straight line. If not, the (Euclidean) radius can be extracted from s with the formula

$$r^2=s^2/(\mathbf{s}\cdot\mathbf{n})^2\,.$$

This is readily verified using the multiplication table above.

1.9 Computational example

The following example is an illustration of how the conformal model can work in practice. It entails drawing spherical circles of equal specified spherical radius about a series of points on the screen. Assuming that the screen is depicting the hemisphere model of spherical space, the expectation is that the circles should appear larger and more off-centre toward the equatorial circle.

We start with a row of equally spaced screen points and stereographically project them onto spherical space S^2, see figure 1.16a. Around each point in S^2 we construct a small spherical circle of spherical radius r (figure 16b). These circles are then stereographically projected back onto the screen (figure 16c).

The aim is to draw the row of final circles on the screen, noting that they will no longer be of the same size nor centred round the original points. We therefore need to calculate the actual centre and radius of each final circle given the original start point **p** and the spherical radius **r**. This is shown in figure 1.16d where the stereographic projection of **p** is denoted by **q**.

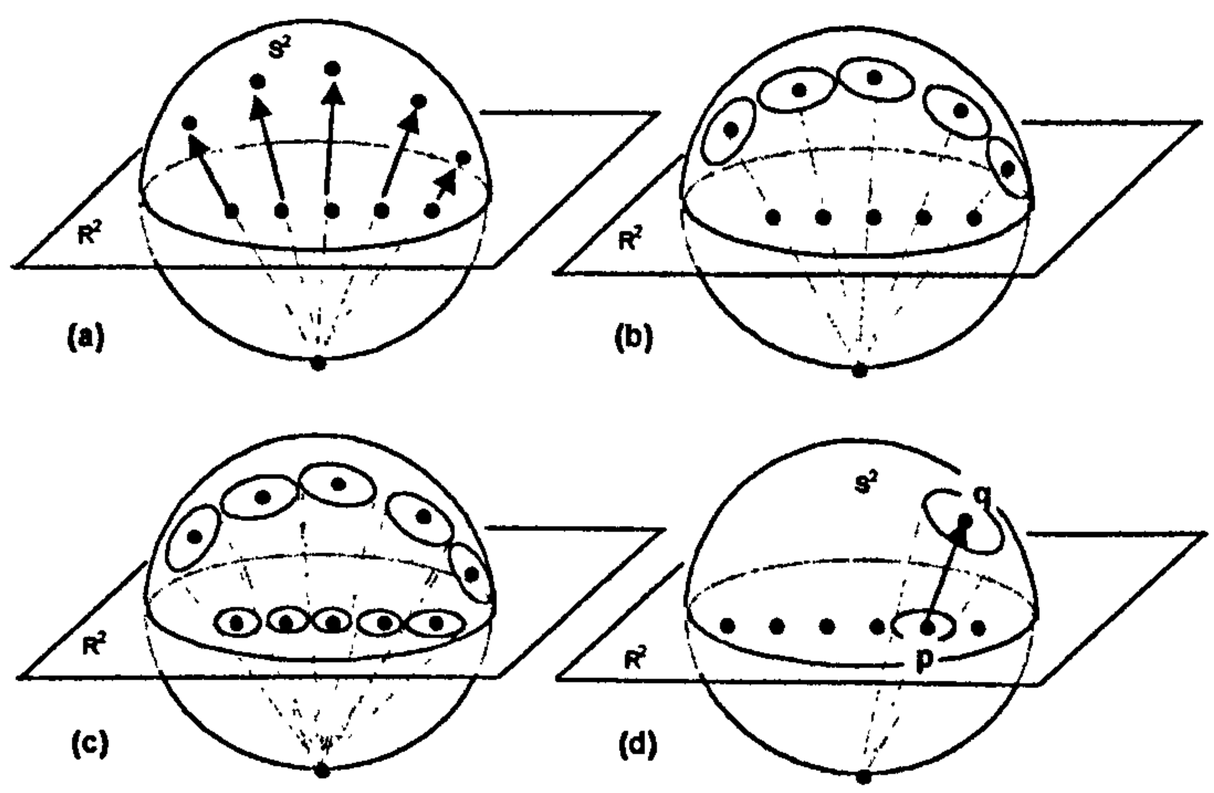


Figure 1.16 Constructing a row of spherical circles in the hemisphere model of spherical space by

(a) projecting points in R^2 onto S^2,

(b) constructing equal radius circles around the projected points ,

(c) projecting the circles back onto R^2.

The calculation proceeds as follows:

Project the Euclidean point **p** onto the null cone to get

$$\mathbf{p} = (\mathbf{p}^2 - 1)\mathbf{a} + 2\mathbf{p} + (\mathbf{p}^2 + 1)\mathbf{e}\,, \qquad \textit{step 1.}$$

Because **q** in S^2 is obtained from **p** in R^2 by stereographic projection, the conformal vector representing q is obtained by the null cone scaling of p

$$\mathbf{q} = -\mathbf{p}/(\mathbf{p} \cdot \mathbf{e})\,, \qquad \textit{step 2.}$$

Since q is the *spherical* centre of a *spherical* circle of *spherical* radius r it is represented by the conformal vector

$$\mathbf{s} = \mathbf{q} - r^2\mathbf{e}\,, \qquad \textit{step 3.}$$

This conformal vector also represents the projected Euclidean circle. Its Euclidean radius ρ can be extracted using

$$\rho^2 = \mathbf{s}^2/(\mathbf{s} \cdot \mathbf{n})^2\,, \qquad \textit{step 4.}$$

The conformal representation of its Euclidean centre is then given by

$$\mathbf{c} = \mathbf{s} + \rho^2\mathbf{n}\,, \qquad \textit{step 5.}$$

This follows from the fact that the spherical circle and its inverse stereographic projection have the same conformal representation in conformal space, hence s is also given by

$$\mathbf{s} = \mathbf{c} - \rho^2\mathbf{n}\,.$$

The actual Euclidean centre **c** can then be extracted from **c** by null cone scaling by a factor of -2**n** then ignoring **a** and **e** components. (Geometric algebra provides a more elegant approach using 'projection' operators.)

Figure 1.17a shows the result of repeatedly applying this calculation to a regular grid rather than to a row of points. The point of projection is taken to be the 3D point (0,0,-1). The Euclidean horizontal and vertical distance between the grid points is 0.2 and the spherical radius of all circles is 0.1. Circles nearer the north pole appear smaller as they are further from the eye point. The Euclidean centres of the circles are not shown but their position can be easily judged. The points that are shown represent the spherical centres of the circles. The discrepancy between them is more pronounced for circles nearer the equator that are more steeply inclined.

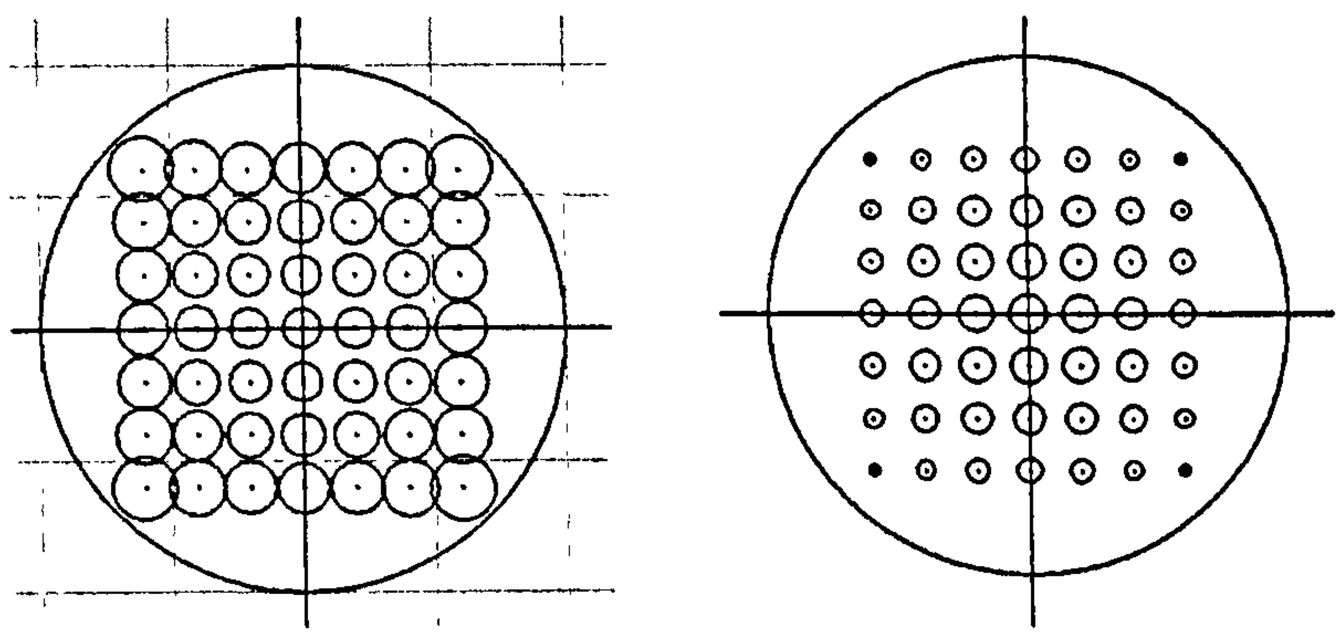


Figure 1.17 Spherical and hyperbolic circles of equal radius constructed around points that form a square grid when viewed as Euclidean points.

The projection of the spherical circles onto the plane faithfully reproduces the intersection properties among them. As evident from the planar projection, the circles do not overlap equally on the sphere even though they are of the same spherical radius. This is simply because the spherical centres, which are the projections of the original planar points onto the sphere, are not equally spaced on the sphere.

This computational example is chosen to highlight certain points: geometric entities such as points and circles have generic representations in Minkowski space which, in a sense, are free of any metric properties, other than the intrinsic dot product. By performing certain operations that entail either **e** or **n**, these entities can be given metric properties related to S^n or R^n respectively. In the case of points, the operation is the appropriate null cone scaling, using either **e** or **n**, followed by the operation that maps the scaled conformal point back into S^n or R^n. In the case of a conformal vector s representing a circle, it can be constructed from the centre and radius in either Euclidean or spherical space.

As another possible interpretation an internet-based metaphor springs to mind. A Minkowski 'server' delivers geometric information that the local clients interpret according to their own local geometry by providing the missing 'key' of either **e** or **n**. Also, a geometry-specific client can post data on the server that can be interpreted by other clients with possibly different geometries. Posting data to the Minkowski server effectively removes (or generalises) the local geometry.

Another way to look at this is related to how projective planes can be constructed from projective space modelled as 3D vectors radiating from a point. The choice of a plane determines a planar realisation, see figure 1.18.

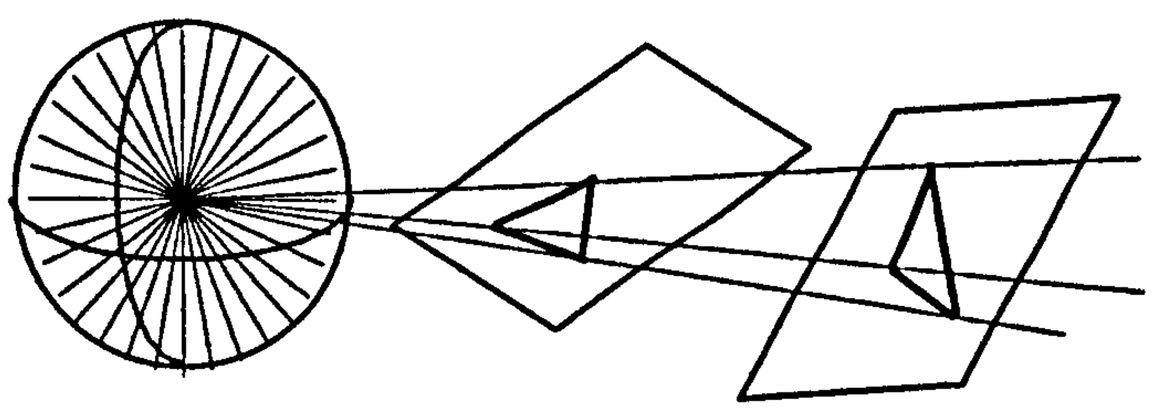


Figure 1.18 Projective planes as intersections of projective space.

In a similar way, the choice and application of either **e** or **n** to determine a section of the null cone provides a relative geometry – a view of how generic

geometric entities in Minkowski space would appear for 'spherical' and 'Euclidean' observers respectively.

The fact that local geometry is determined by a single vector in the Minkowski space has powerful computational consequences. With this in mind, we will identify a single vector that can determine a hyperbolic space and then apply simple substitutions to the above computational example to generate a flat image of a grid of hyperbolic circles – this will be the Poincaré disc model. We will then apply the same ideas to quickly generate a grid of hyperbolic circles in the half-space model.

1.10 Embedding hyperbolic space onto the null cone of Minkowski space

The standard model of hyperbolic space is the surface H^n of a unit 'sphere' of negative radius in Minkowski space. This surface is asymptotic to the null cone and appears as two hyperboloids, see figure 1.19a.

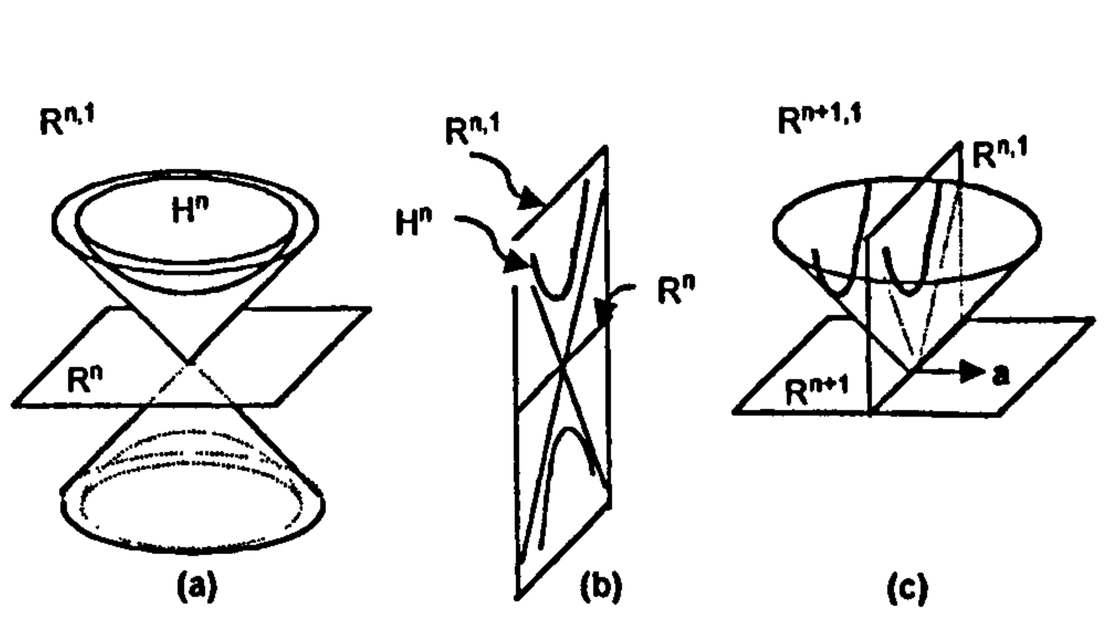


Figure 1.19 Projecting hyperbolic space H^n onto the null cone of the Minkowski space $R^{n,n+1}$.

Figure 1.19b shows H^n in a drawing space of one less dimension. The dimension reducing convention being used here differs from the previous two and is based on axial symmetry. As before, the extra available drawing dimension is used in figure 1.19c to depict the null cone of $R^{n+1,1}$.

As figure 1.19 suggests, H^n can be embedded on the null cone using a unit Euclidean vector **a** perpendicular to $R^{n,1}$ via the map

$$\mathbf{x} \to \mathbf{X} = \mathbf{x} - \mathbf{a}.$$

The vector x is null since

$$(\mathbf{x}-\mathbf{a})^2 = \mathbf{x}^2 - 2\,\mathbf{a}\cdot\mathbf{x} + \mathbf{a}^2$$
$$= -1 - 0 + 1 = 0.$$

Most results obtained for S^n carry over to H^n, but in this case the key vector defining the geometry is **a** rather than **e**. For example, if the conformal model of H^n is viewed as a 'planar' section of the null cone of $R^{n+1,1}$, the intersecting hyperplane has equation $\mathbf{x}\cdot\mathbf{a} = -1$ and the model is described by

$$N_{\mathbf{a}}^n = \left\{ \mathbf{x} \in R^{n+1,1} \mid \mathbf{x}^2 = 0, \quad \mathbf{x}\cdot\mathbf{a} = -1 \right\}.$$

This is valid since $\mathbf{x}\cdot\mathbf{a} = (\mathbf{x}-\mathbf{a})\cdot\mathbf{a} = \mathbf{x}\cdot\mathbf{a} - \mathbf{a}^2 = 0 - 1 = -1$. Normalisation of a conformal vector h is achieved through the null cone scaling $-\mathbf{h}/(\mathbf{h}\cdot\mathbf{a})$ rather than by $-\mathbf{h}/(\mathbf{h}\cdot\mathbf{e})$. Similarly, a hyperbolic circle with hyperbolic centre **c** and hyperbolic radius r is defined by the conformal vector $\mathbf{s} = \mathbf{c} - r^2\mathbf{a}$. In general, results for the spherical case carry over to the hyperbolic case by simply interchanging **e** with **a**.

1.11 Stereographic projection of H^2 to R^2 : The Poincaré disc

The stereographic projection of H^2 to R^2 is the Minkowski equivalent of the stereographic projection of S^2 to R^2. The projection is via the Minkowski vector **-e** rather than the Euclidean vector **-a**, see figure 1.20.

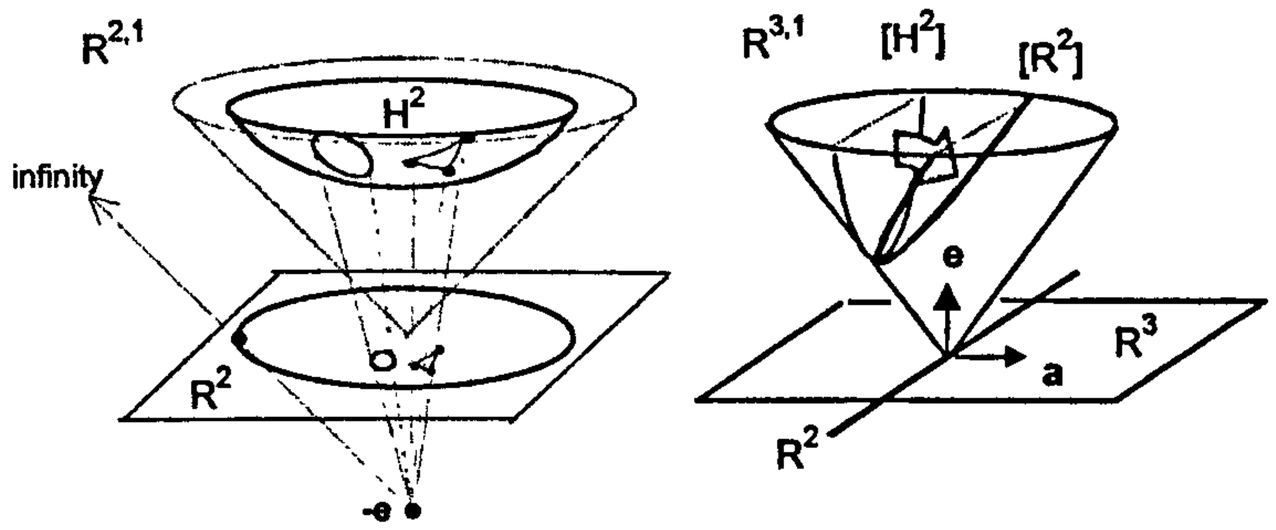


Figure 1.20 The stereographic projection of H^2 to depicted as a scaling on the null cone.

Projecting H^n to R^n through **-e** provides a 'flat' model of hyperbolic space known as the 'conformal ball model', a generalisation of the Poincaré disc.

If the original computational example is repeated with **a** replacing **e** in steps 2 and 3, a picture of the equal-radius hyperbolic circles appear on the Poincaré disc, see figure 1.17b. The fact that the circles are of equal hyperbolic radius gives some idea of how the metric changes on the Poincaré disc – as the bounding 'horizon' circle is approached, the circles shrink indefinitely as they 'recede into the distance'. On H^2 the circles are moving up the hyperboloid asymptotically toward the null cone.

1.12 The half-space model of H^2

The half-space model can be obtained by stereographic projections from H^2 to R^2 to S^2 then back to R^2 using different points of projection, see figure 1.21.

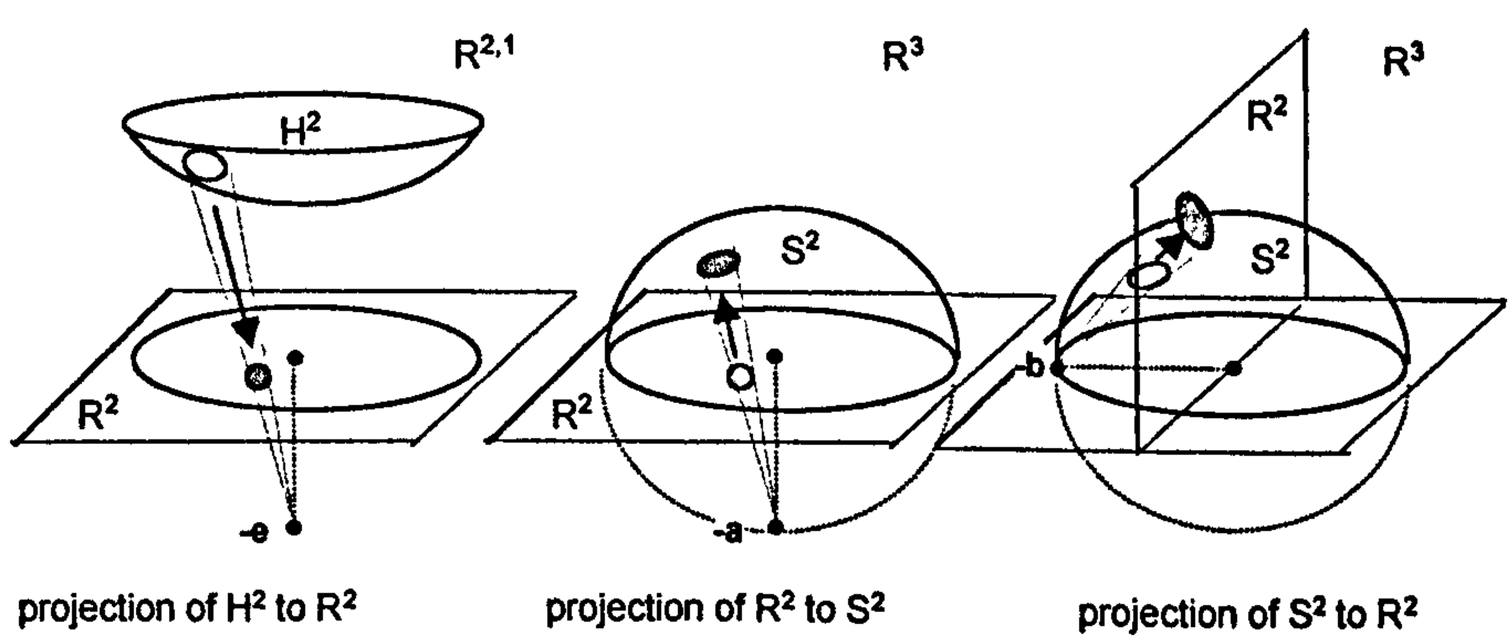


Figure 1.21 The three projections used in the construction of the half-space model of hyperbolic space.

Each successive projection is represented respectively by the null-cone scaling depicted in figure 1.22.

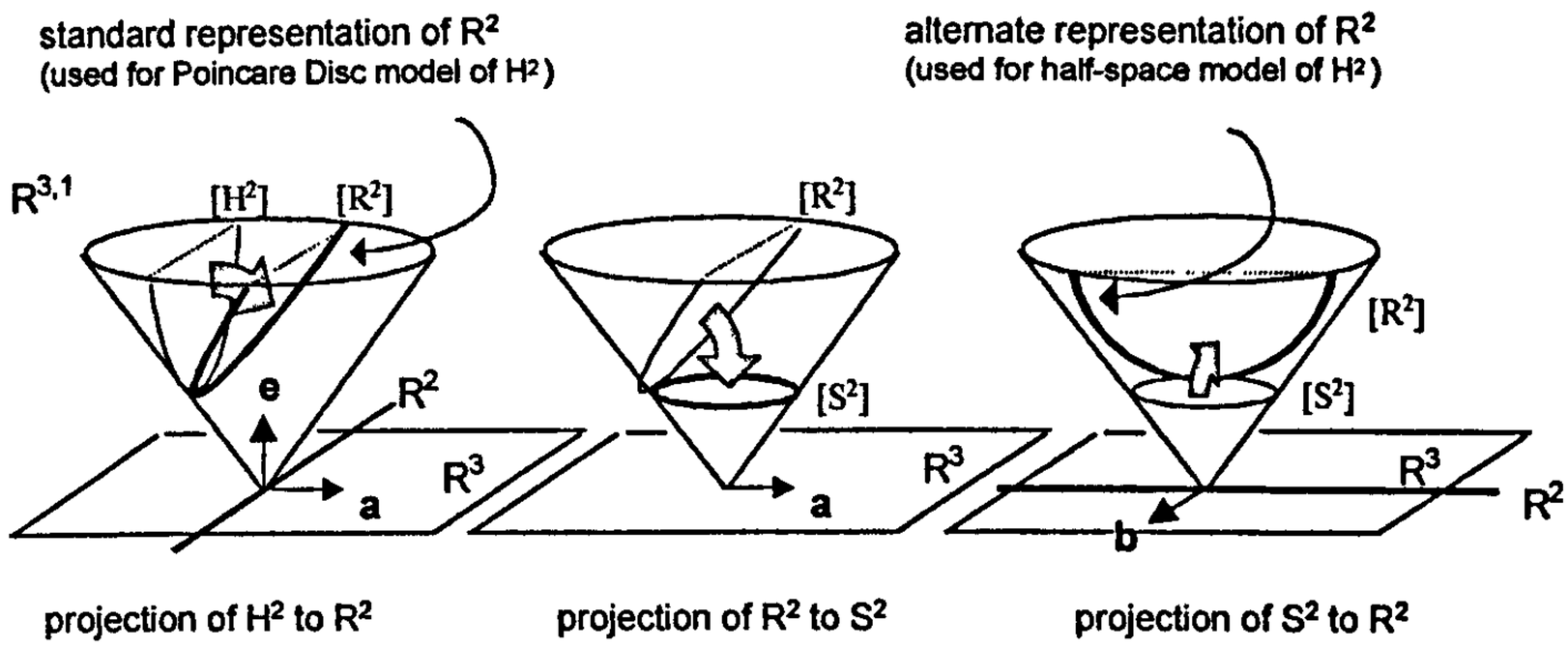


Figure 1.22 The three projections used in the construction of the half-space model of hyperbolic space depicted as null cone scalings.

The null cone section representing the R^2 plane of the half-space model is derived from a stereographic projection of S^2 using the vector **b** = (0,1,0,0). The standard null cone representation of R^2 is derived in a similar way using the vector **a**. The vectors **a** and **b** relate to the way that R^2 is assumed to be embedded in R^3. Each defines a different null cone scaling to a different representation of R^2.

If the original computational example is repeated with **b** now replacing **e** in steps 2 and 3, a picture of the equal-radius hyperbolic circles appears on the half-space model, see figure 1.23. In the figure the range of the original point grid has been changed so that they are within the unit circle and have positive second co-ordinate.

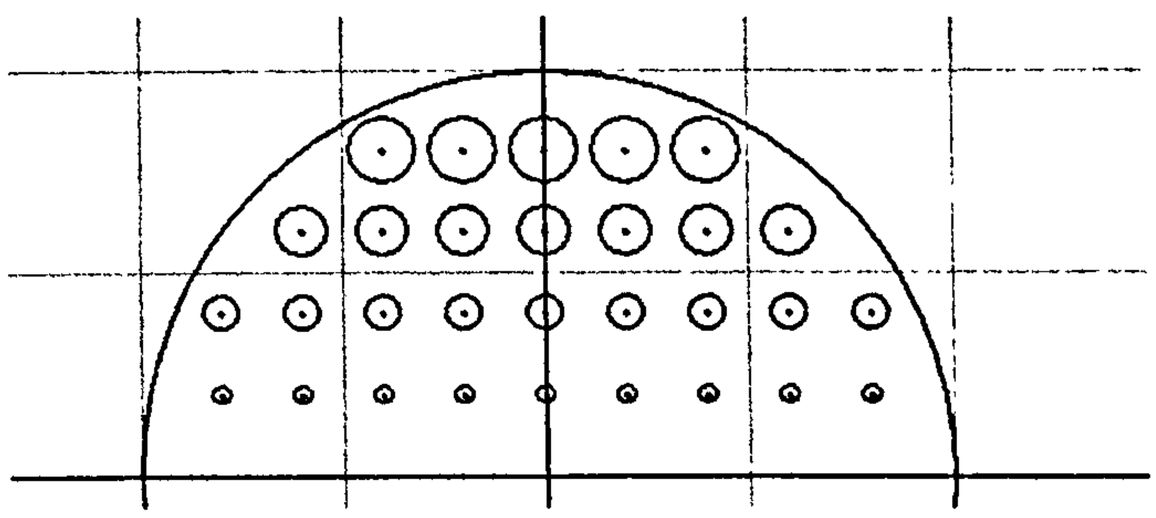


Figure 1.23 The half-space model of hyperbolic space showing circles of equal hyperbolic radius constructed around points that form a regular grid when viewed as Euclidean points.

In the half-space model, the metric changes so that the circles of equal hyperbolic radius appear smaller as they move down toward the horizontal 'horizon'.

1.13 Blades and the wedge or external product.

The idea that a formula derived for Euclidean geometry, and which therefore entails **n**, can be adapted for other geometries by simply swapping **n** seems both powerful and pervasive. This final section shows the idea being used to draw triangles in four different geometries. The geodesic sides of the triangles are arcs of circles represented by blades rather than vectors.

A blade is the wedge product of one or more vectors and it represents the linear subspaces spanned by them. The grade of the blade corresponds to the number of vectors in the wedge product so that, for example, if v_1, v_2 and v_3 are vectors, $v_1 \wedge v_2$ is a grade-2 bivector and $v_1 \wedge v_2 \wedge v_3$ is a grade-3 trivector, and so on. Generally, every grade-2 blade is a bivector, but not every bivector can be factored into the wedge product of 2 vectors. The latter is particularly true in higher dimension spaces. The same applies to grade-3 blades and so on.

In our conformal model a circle through **p**, **q** and **r** is represented by the intersection of the blade $p \wedge q \wedge r$ with the null cone. In particular, if **x** and **y** are

Euclidean points in R^2, then the blade $\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{n}$ contains $\mathbf{n}$, the point at infinity, and therefore the 'circle' is a geodesic straight line. In the same way, if $\mathbf{x}$ and $\mathbf{y}$ are points in S^2, then the blade $\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{e}$ contains $\mathbf{e}$ and so represents a geodesic great circle. This vector swapping extends to models of hyperbolic space; the blade $\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{a}$ represents a geodesic in the Poincaré disc model, and $\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{b}$ represents one in the half-space model.

It is thus possible to easily draw triangles in any of the spaces by joining the points in pairs with geodesic arcs. However, formulae are needed for extracting the Euclidean centre c and radius r from the blade representation L of an arc. Geometric algebra provides two generic formulae, [4, p363-364],

$$r^2 = -L^2/(L \wedge \mathbf{n})^2 \qquad \text{and} \qquad c = \mathbf{n}L\mathbf{n}.$$

The implied product here is the 'geometric' product. Since Euclidean information is being extracted from the blade (for drawing purposes), the formulae entail $\mathbf{n}$ as expected.

Figure 1.24 shows a triangle drawn in four different spaces. The calculation of the Euclidean triangle entails creating blades $\mathbf{x} \wedge \mathbf{y} \wedge \mathbf{n}$, $\mathbf{y} \wedge \mathbf{z} \wedge \mathbf{n}$ and $\mathbf{z} \wedge \mathbf{x} \wedge \mathbf{n}$ and extracting Euclidean drawing data from each. For the remaining images, the process is then repeated with $\mathbf{n}$ replaced respectively by $\mathbf{e}$, $\mathbf{a}$ and $\mathbf{b}$.

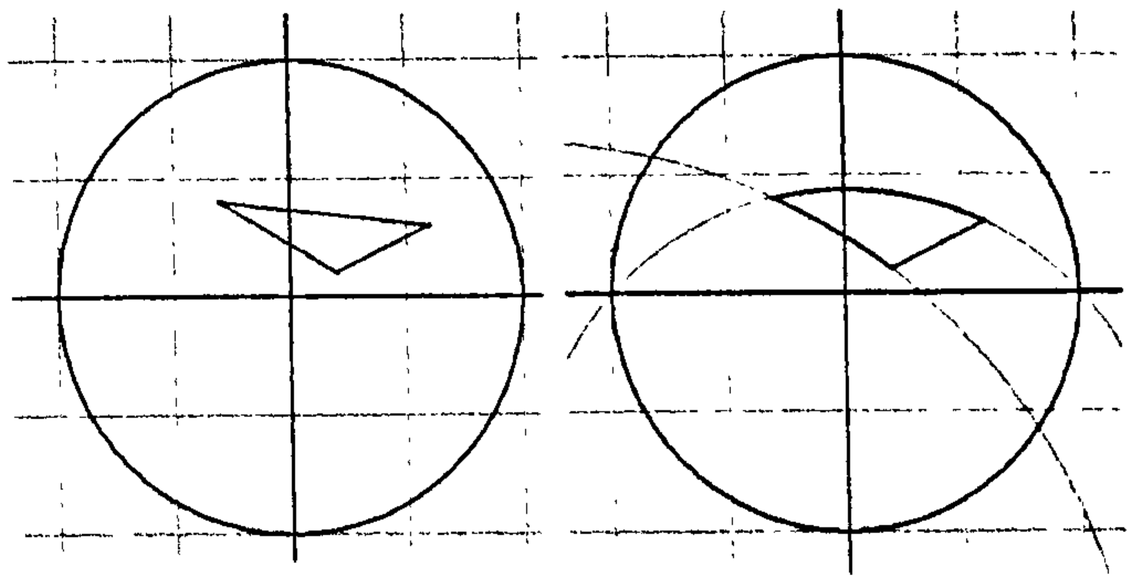


Figure 1.24 Euclidean and non-Euclidean triangles.

Though incomplete in its explanation, this example does again show how the underlying approach of swapping geometry-defining vectors still applies.

1.14 Conclusion

This chapter has shown that the conformal model can be adequately explained and visualised without recourse to geometric algebra - the dot product being used was applied to vectors only, there being no other 'external' elements. This was possible because points and circles were represented solely by vectors.

However, the last section introduced the alternate trivector blade representation of circles, since it provides the most efficient way to derive a circle through three specified points. The blade exists in the 'external' geometric algebra,

though the 3D hyper-plane it represents does not. In the blade interpretation, the circle is the collection of points where the 3D hyper-plane meets the null cone. The dimension reducing techniques developed here are simply not suited for visualising this intersection.

This chapter also successfully reconciled two profoundly different viewpoints about the relationship between the geometries embedded in the conformal model. That of Hongbo Li et al [1], [2] and [3] is primarily focused on stereographic projection, whereas that of Doran and Lasenby [4] is more focused toward null cone scaling. This reconciliation was tested with the half-space model of hyperbolic space. The description given by Hongbo Li et al, which is entirely in terms of stereographic projection, was successfully reinterpreted and implemented as a series of null cone scalings.

By deriving a 5-step implementation algorithm, the chapter successfully started the process of turning 'theory into practice'. The algorithm constructed a non-Euclidean circle of a given non-Euclidean radius around a specified screen point. The challenge in constructing the algorithm lay in the fact that its start and end point, of necessity, needed to be Euclidean. The original point was specified in screen/viewport co-ordinates and, in order to draw the final circle, its centre and radius needed to be specified in the same way. The software developed for producing the screen shots of this chapter is discussed in chapters 3 and 4.

This chapter lays the foundation for much of the later work in the way that it provides a visualisation of the conformal representation of circles, especially in relation to the null cone. This concentration on the conformal vector representation of circles takes the work in new directions as described in chapters 6 to 9.

The chapter also represents a start to the consideration of implementation issues. It highlights the fact that for most non-Euclidean interactive graphic processes, the starting and ending geometry will always be Euclidean. For

example, a new mouse co-ordinate may give rise to a repositioned circle which, in order to be drawn, needs to have its screen centre and radius re-calculated.

Algorithms entail first projecting out of the starting Euclidean geometry of the screen onto the null cone of conformal space. Once there, various operations may take place. The final step is invariably projection from the null cone back to the screen, but before doing so, null cone scaling may be necessary to ensure that Euclidean geometry is the end product.

2 Visions of Interaction

2.1 Introduction

This chapter describes interactive ideas that emerged when examining how concepts associated with transformations could be communicated. The examples given show the variety and nature of the interaction and do not represent a coherent curricular programme in any way.
Emergent design principles and influences are discussed at the end of the chapter.

A central aim was to build a Java 'package' (i.e. a class library) which would hide complexity allowing experimental scenes to be built quickly and easily using basic Java. The examples in this chapter are implemented as applets on a web page with two including the source code used to generate them.

The accompanying CD contains the example web pages discussed in this chapter, together with other example web pages. It also contains the Java source code for the package, which is called *transformax*.

2.2 Controlling turtles and sprites

An appropriate place to start the description of the environment seems to be with a classical dart-shaped 'turtle'. However, in this case it is driven by a small console or 'controller' rather than by a sequence of commands which form part of a wider programming language, see figure 2.1.

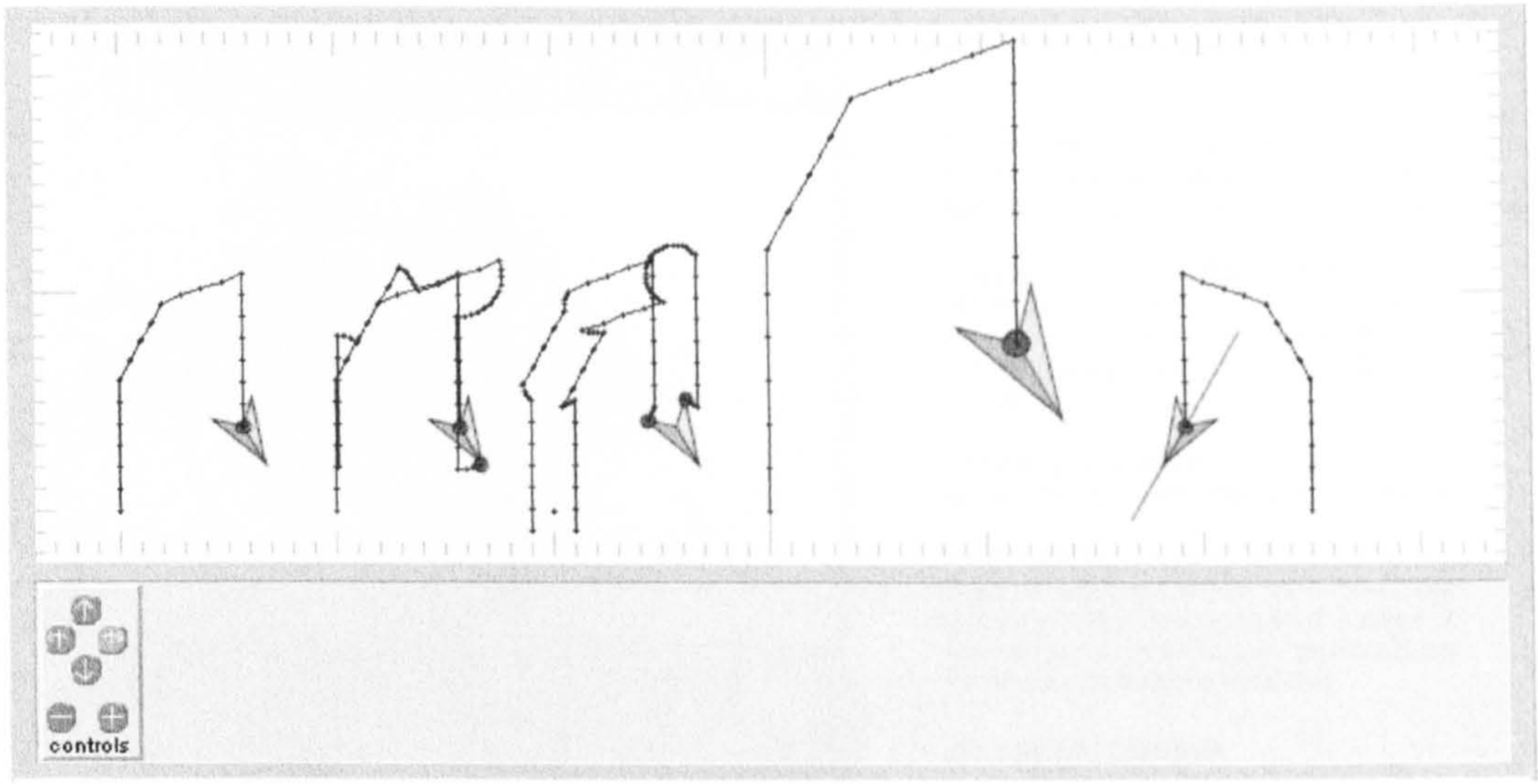
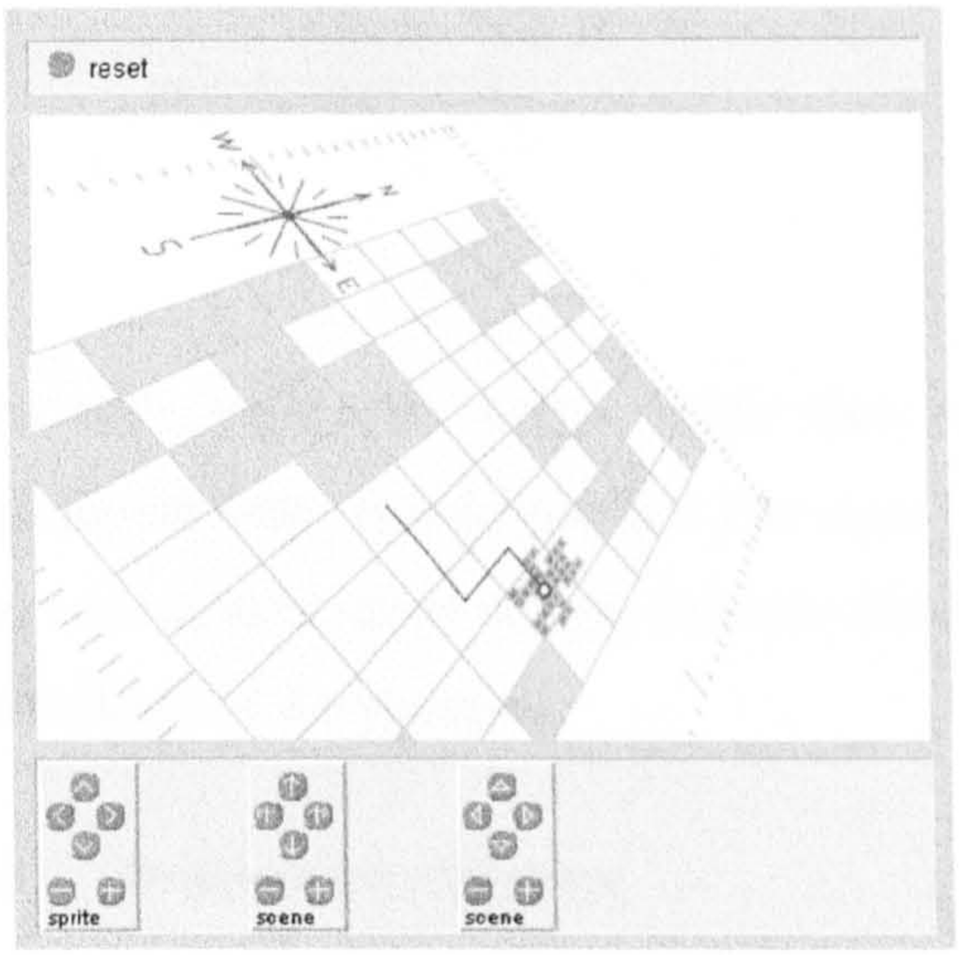


Figure 2.1 Applet showing paths traced out by different sets of locus points attached to five turtles driven by one turtle controller.

In the figure, the controller is driving all five turtles each with different sets of 'locus points' attached. Though each locus point is associated with a particular turtle, it can be dragged relative to that turtle to reposition it. The fourth turtle has been enlarged and the fifth has been flipped about its axis by clicking on the 'reflector' positioned on its axis. It therefore turns left when the right-turn button is pressed, and vice versa, thus tracing out a reflected path.

The 'turtle' controller, which rotates an object left or right, can be applied to any scene object. The same applies to a 'sprite' controller that moves its target objects left or right rather than rotate them. A third 'perspective' controller 'leans' an object into the third dimension. Figure 2.2 shows a web page activity based on these.



Introducing the sprite

The first controller moves the sprite north, south and east and west. Use it to move the sprite around avoiding the maze obstacles.

Use the second controller to rotate the scene so that north no longer points directly up the screen. Try driving the sprite when the scene is completely upside down. Avoid the obstacles!

Use the third controller to introduce perspective and drive the sprite some more.

Notice that the compass always points north. Drag the compass around the scene. Note that even if its shape appears to change, it always points north. So you can always get your bearings from the compass.

view Java source code

Figure 2.2 Web page: navigating a sprite through a perspective maze.

The animation is generated by the following Java source code:

```
import transformax.*;

public class Applet03 extends GApplet
{
   public void start()
   {
      scene.add(new Maze());
      scene.add(new Compass(-18,8));

      Sprite sammy = new Sprite();
      sammy.add(new TrackerPoint(5));
      scene.add(sammy);

      Controller sc = new SpriteController(3,3,"sprite");
      sc.target(sammy);
      panel.add(sc,0,0);

      Controller tc = new TurtleController("scene");
      tc.target(scene);
      panel.add(tc,100,0);

      Controller pc = new PerspectiveController("scene");
      pc.target(scene);
      panel.add(pc,200,0);
```

```
        scene.render();
    }
}
```

The code builds the scene graph then renders it, so object creation and attachment order is unimportant. The class library that defines the behaviour of the various new scene objects is made accessible through the import statement at the head of the program.

2.3 In-scene transformers

The three controllers above transform objects (or the whole scene) but they are not part of the scene. In contrast, there are transformer objects that can be placed in the scene so that, when clicked, they transform a target object. These can also be attached directly to the object being transformed so that they move with the object.

For example, the 'translator' is a simple transformer in the shape of an arrow. The arrow has drag points at either end - the point at its base drags the arrow as a whole while the other is used for reshaping the arrow to change the parameters of the translation.

A translator can also be created without drag points so that it is fixed in size and orientation. A group of such translators can be made to target a single object to precisely define its allowed transformations, see figure 2.3.

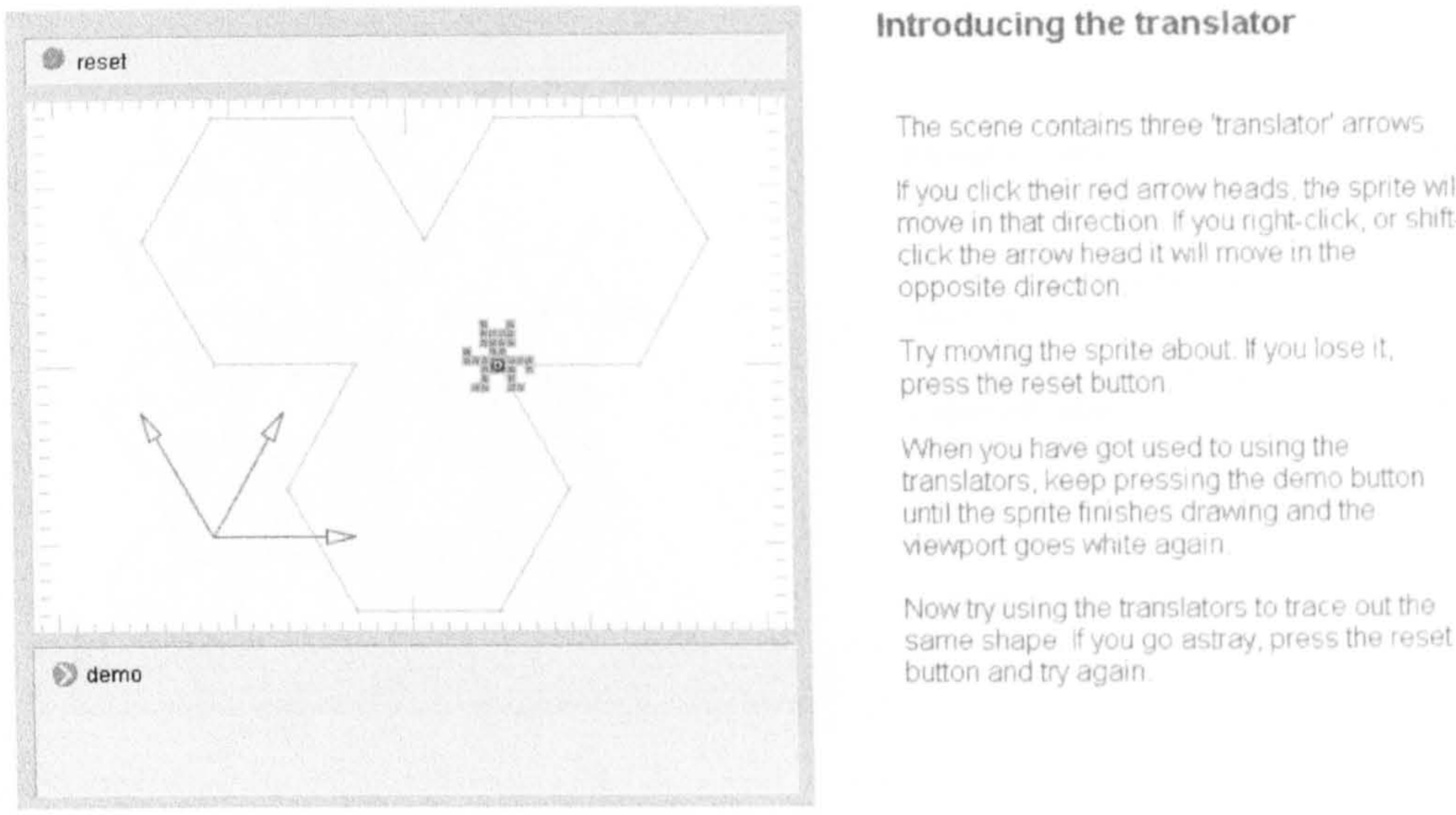


Figure 2.3 Web page: path tracing exercise using a multi-translator.

The idea underlying the translator was readily extendible, as depicted in the hierarchy of newly developed transformer types depicted in figure 2.4.

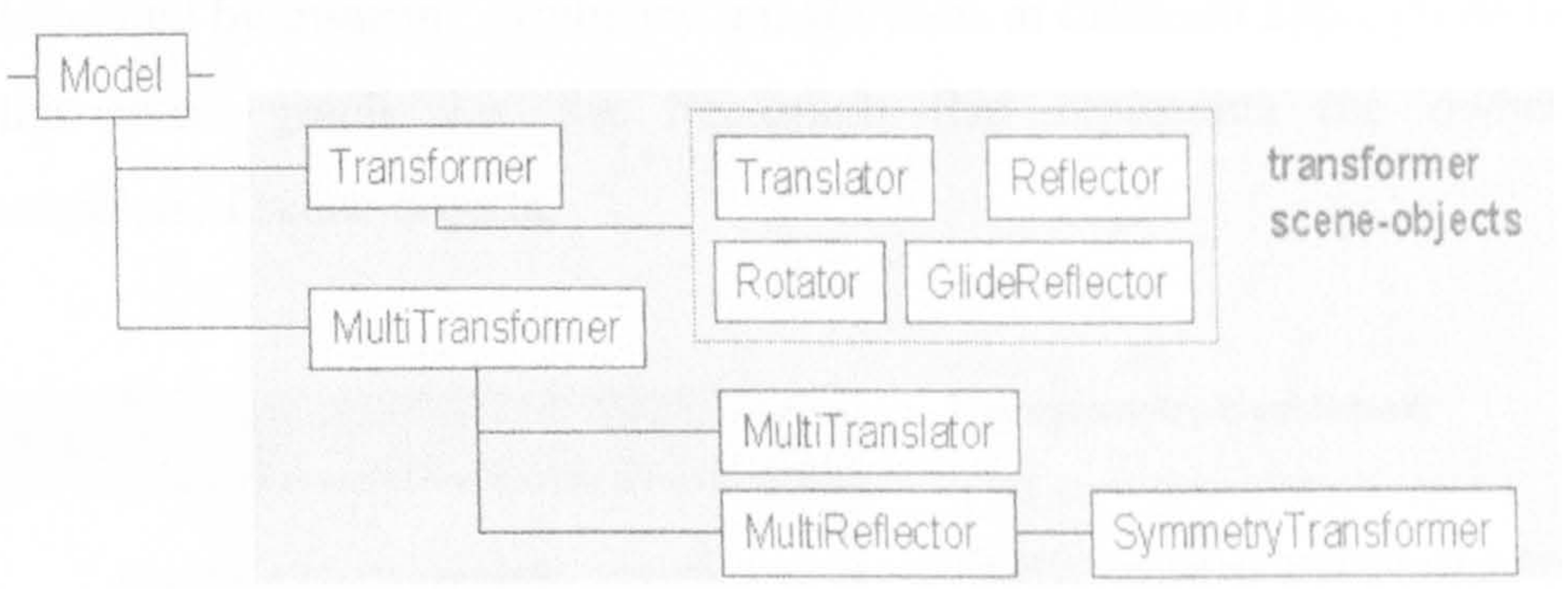


Figure 2.4 Transformer and MultiTransformer class hierarchies.

One of these types, the multi-translator, can be used to create tessellations. For example, figure 2.5 shows a tessellation built from a hexagon to which a 6-fold multi-translator has been attached. The size of its constituent translators are chosen to create the star-shaped gaps in the pattern, and a 'trace' option is set to ensure old images remain.

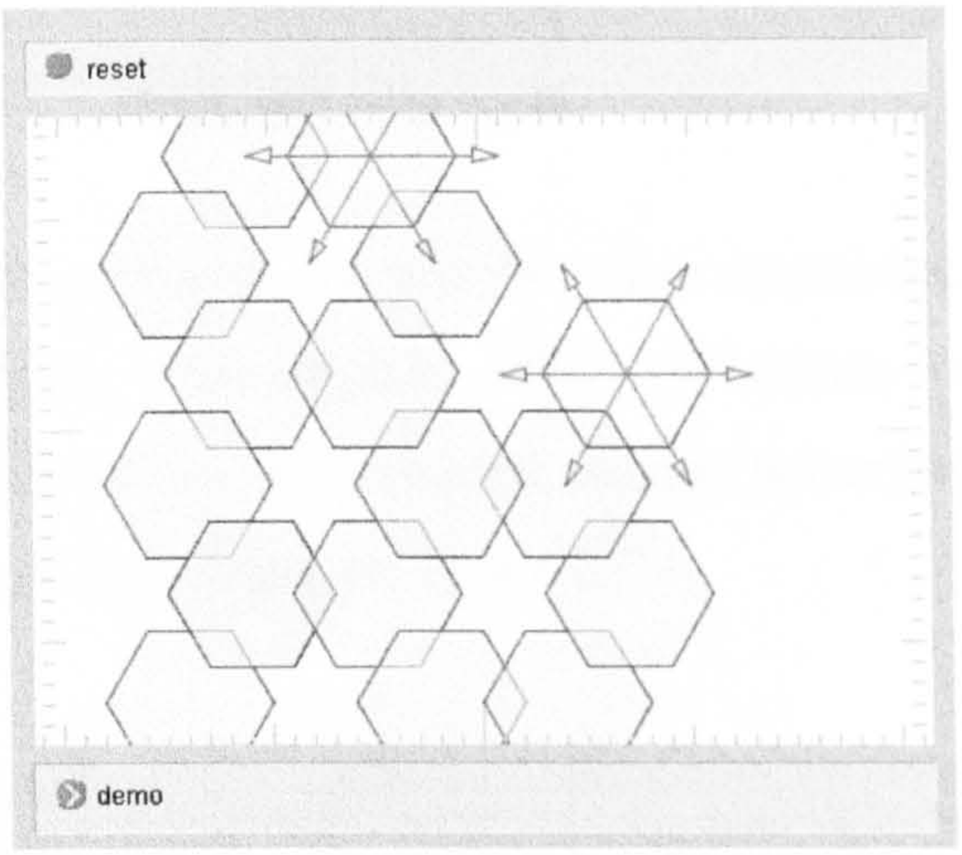


Figure 2.5 Web page: tessellation building using the multi-translator.

A second transformer type, the symmetry transformer, combines reflectors and a central rotator into a single entity using an icon sometimes adopted in crystallography. The symmetry of the icon reflects the underlying symmetry of the transformer. To illustrate, the more challenging task shown in figure 2.6 is implemented by inserting symmetry transformers at different appropriate levels in the scene graph, i.e. the tree-graph that represents the ownership organisation of scene objects.

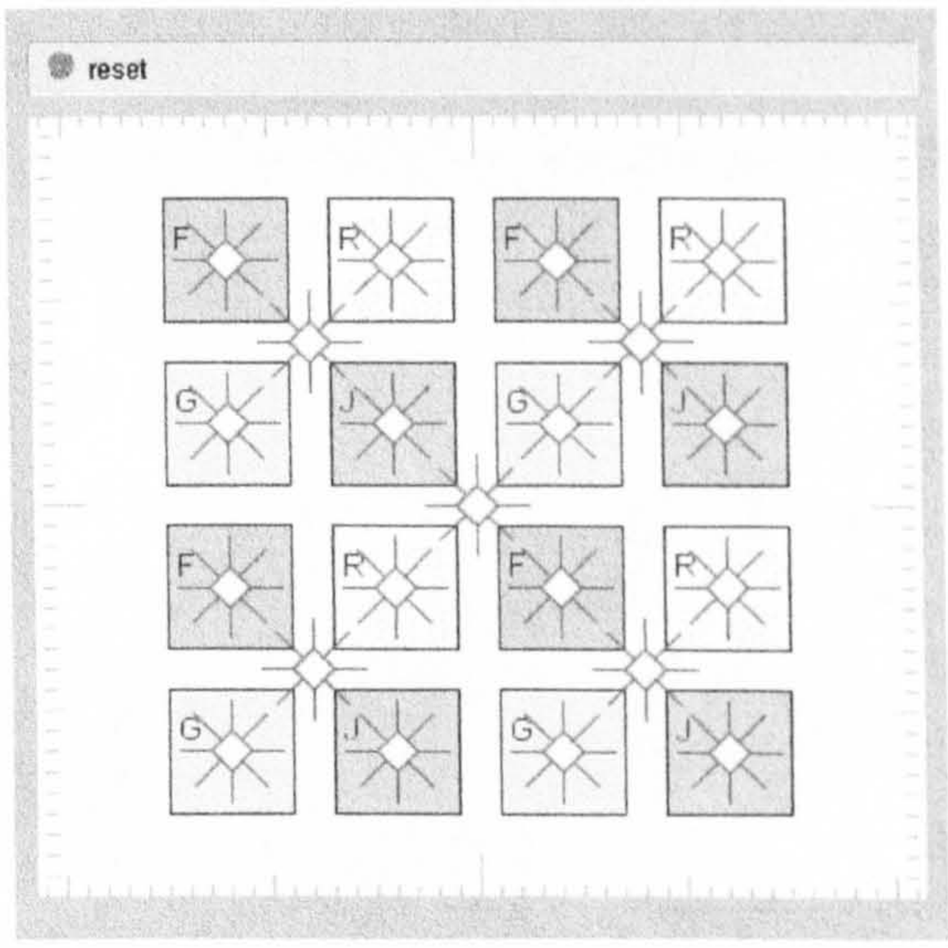


Figure 2.6 Web page: puzzle based on the symmetry transformer.

2.4 Click and drag tools

Scene objects can also be made responsive to a range of click and drag tools which, when applied, effect all objects lower in the scene hierarchy. The introduction of interactive robotic behaviour out of context can be the basis of novel activities, see figure 2.7.

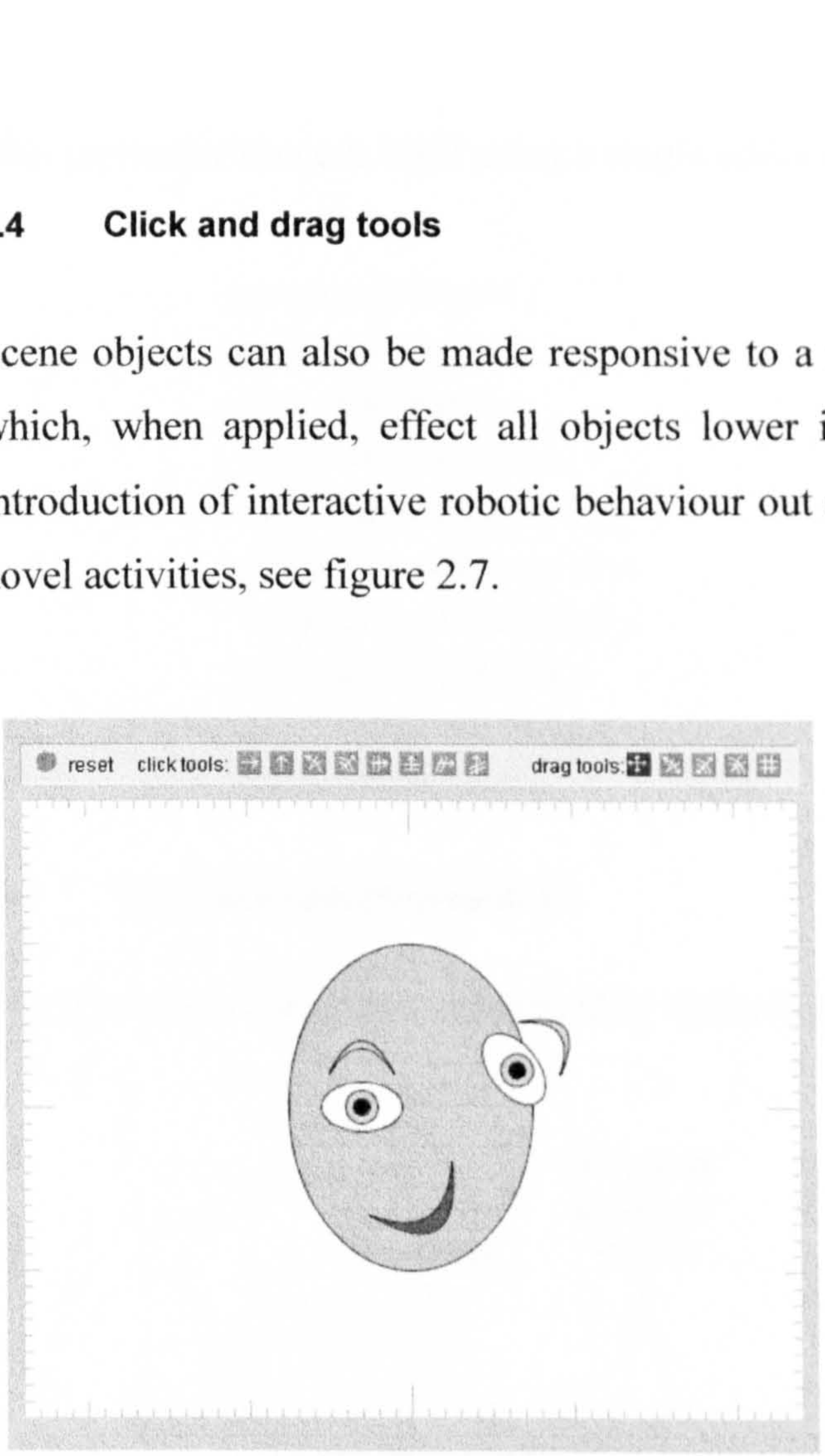


Figure 2.7 Web page: using a robot-like structure to Introduce the transformer toolbar click tools.

In this case the underlying scene graph structure is depicted in figure 2.8.

.

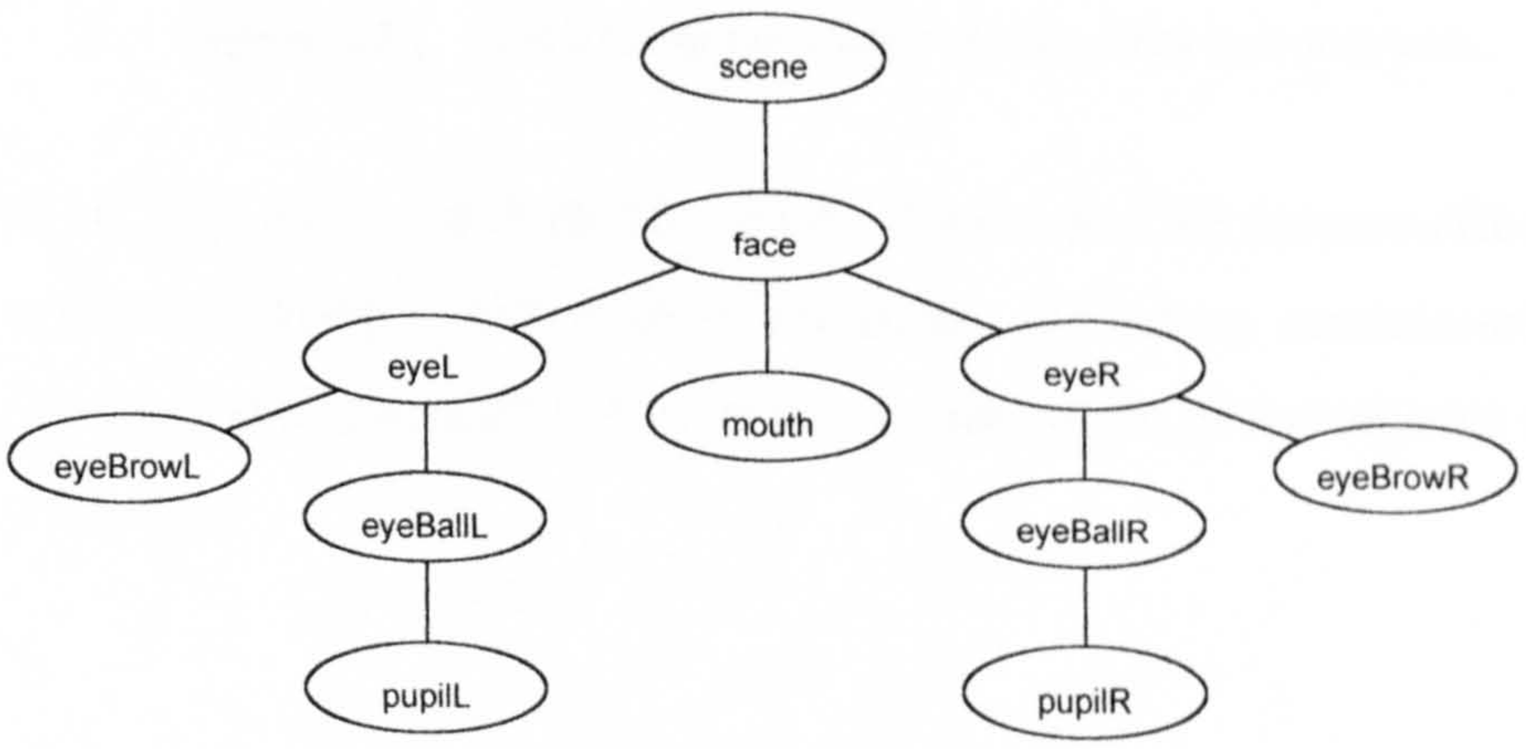


Figure 2.8 Face scene graph.

This particular scene is built using a single add() function:

```
scene.add(face);
face.add(eyeL);
face.add(eyeR);
eyeL.add(eyeBallL);
eyeR.add(eyeBallR);
eyeBallL.add(pupilL);
eyeBallR.add(pupilR);
eyeL.add(eyeBrowL);
eyeR.add(eyeBrowR);
face.add(mouth);
```

2.5 The transformation toolbar

The click and drag tools are labelled in figure 2.9

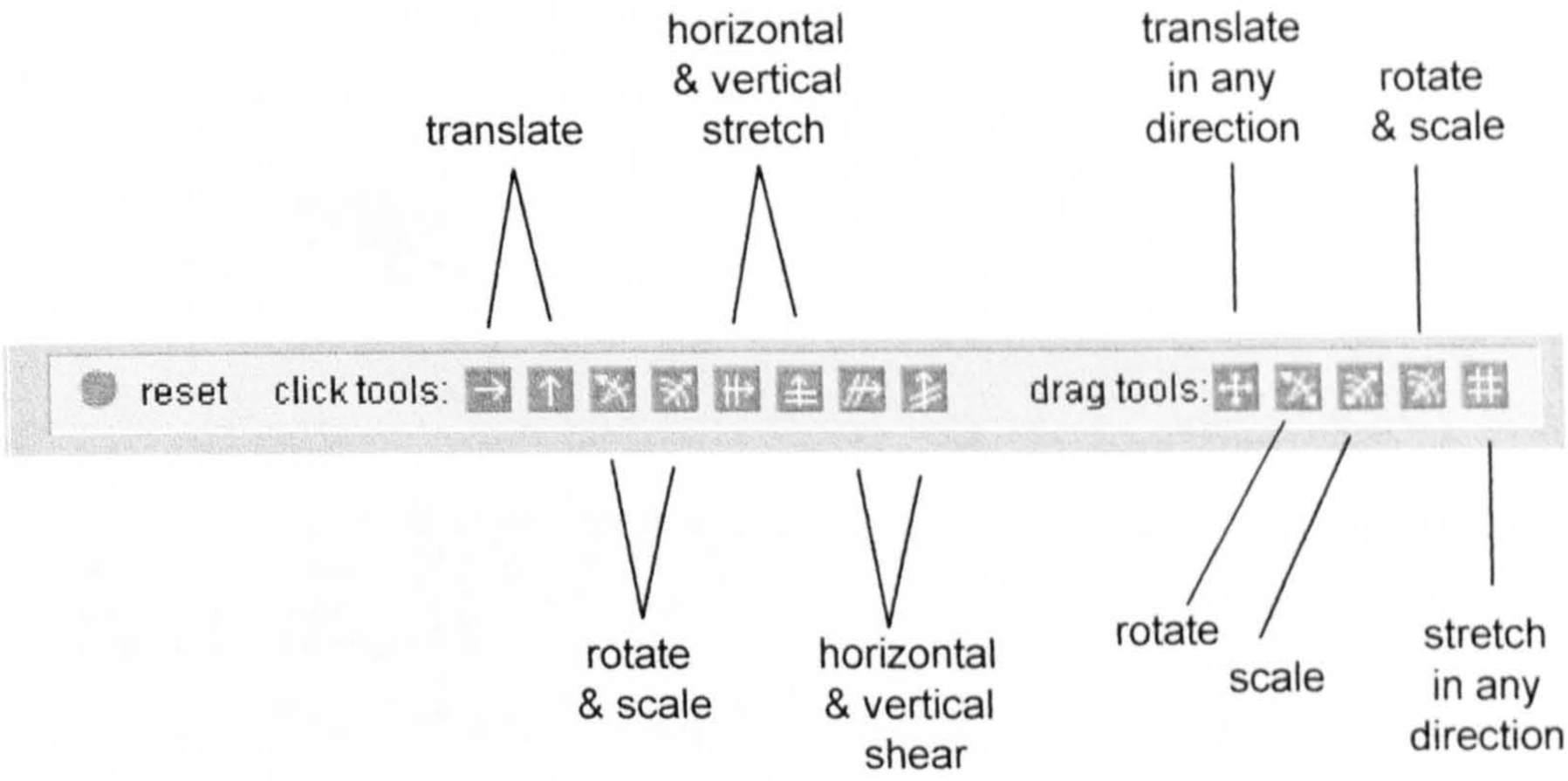


Figure 2.9 The transformer toolbar showing click and drag tools.

Each click tool is activated by the left-button and the inverse effect by the right button. Holding a button down duplicates the action continuously. Rotations, stretches and shears are done relative to the origin of the object's modelling co-ordinates.

2.6 Grids and co-ordinates

If a rectangular or polar grid is attached to an object it partly obscures it. Consequently, visible objects have a selection of background grids and axes that can be selectively shown. As these are part of the object, they transform when the object is transformed and therefore show local modelling co-ordinates which, in turn, are the basis upon which any further transformation is applied. The net effect is that when an object's grid is shown, transformations are perceived as being totally localised, to such a degree that 3D perception may intervene. For example, the sprite and turtle in figure 2.10 have undergone different transformations so that their grids show conflicting perspective.

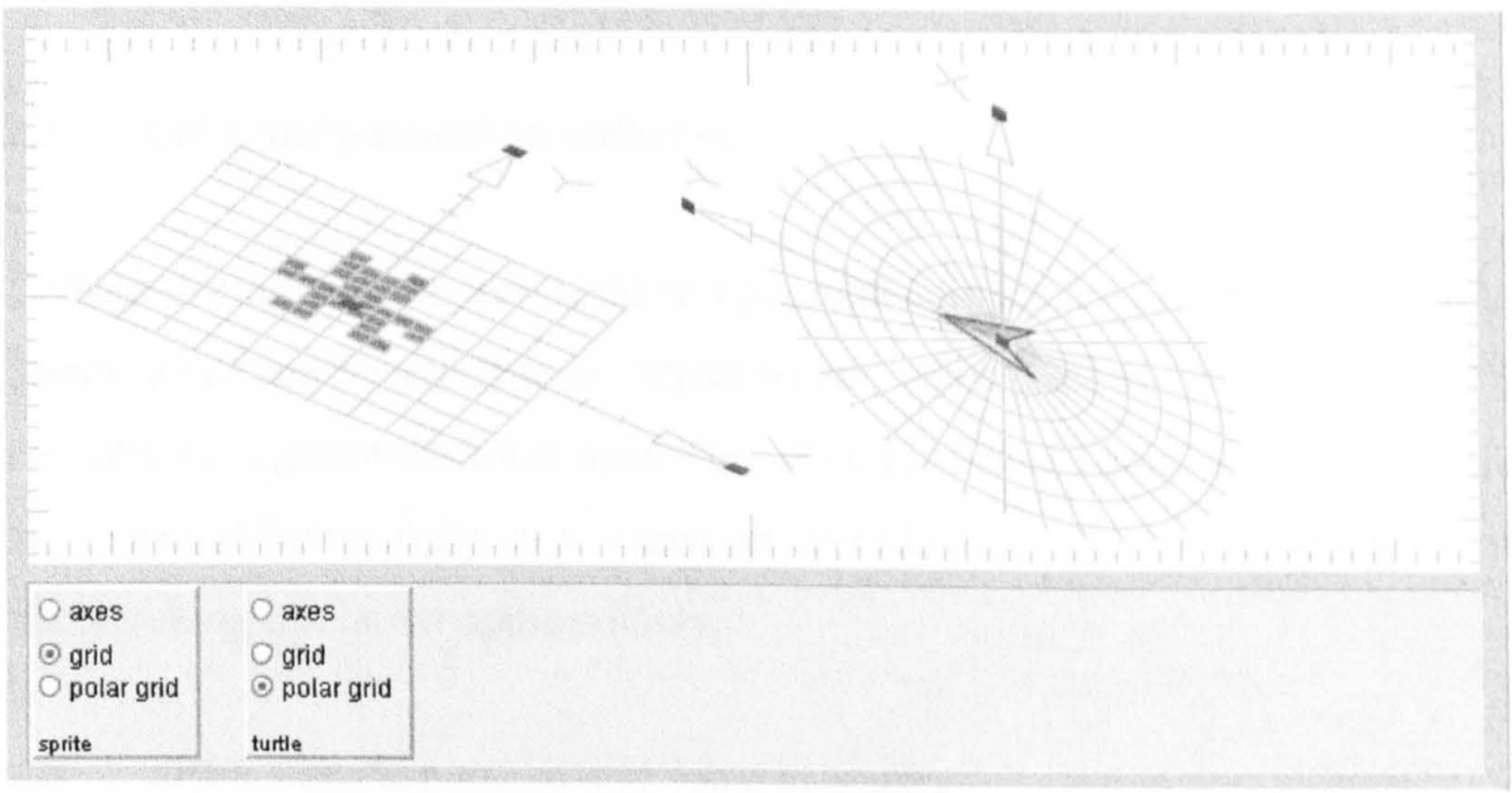


Figure 2.10 Sprite and turtle with rectangular and polar grids attached.

This example was genereted by the source code:

```
import transformax.*;

public class Applet11 extends GApplet
{
   public void start()
   {
       Sprite s1 = new Sprite();
```

```
        scene.add(s1);
        s1.add(new AffineTransformer());

        Turtle s2 = new Turtle();
        scene.add(s2);
        s2.add(new AffineTransformer());

        GridSelector g1 = new GridSelector("sprite");
        panel.add(g1,0,0);
            g1.target(s1);

            GridSelector g2 = new GridSelector("turtle");
            panel.add(g2,100,0);
            g2.target(s2);

           scene.render();
      }
   }
```

2.7 Local and parental co-ordinates

To help distinguish between local and parental co-ordinate systems, a clickable 'math' point can be added to an object which, when clicked, shows the point's co-ordinates against the local modelling grid. If clicked again, it does the same in the parent frame, right up to viewport co-ordinates. However, the degree of backtracking can be set appropriately.

2.8 Affine transformations and local geometry

The draggable 'affine transformer' consists of two base vectors with draggable end points, see figure 2.10. When an end point is dragged, an incremental affine transformation is applied to the object to which the transformer is attached. This transform is therefore also applied to the base vectors which consequently update themselves so that the end point that is being dragged follows the mouse pointer. Unlike the earlier transformers, the affine transformer causes a continuous change, giving that the impression that the object (or scene) is being smoothly reshaped. If a grid is on display, the impression is that its changing geometry effects everything that is dependent

upon it - the local frame describes local geometry. This theme is echoed at a lower level by allowing mathematical 'instruments', a protractor, ruler and (magnetic) compass to be dragged through the 2D space to gauge local geometry. The use of grids (or instruments) and gradual comparative change, implemented through mouse dragging, could provide a means of refocusing attention from the object to the space in which it exists.

2.9 Multiple viewports and avatars

Because the architecture is scene-graph based it supports multiple viewports – the same scene can be looked at, and interacted with, from different viewpoints. Thus it is possible to place an avatar in the scene and to view what the avatar sees in a separate viewport. Interactivity applies to both viewports making it possible to drag the avatar while at the same time seeing what the avatar sees.

2.10 Conclusion

The experimental system developed here uses a classical scene-graph architecture, but in a 2D context. The architecture supports a 'matrices only' approach where all calculations are implemented by composing matrices, some of which may have been inverted. The significance of this lies in the converse, i.e. that once the composition and inversion rules for transformations are known, a scene graph architecture can be readily implemented.

Fortunately, the composition and inversion rules for geometric algebra based bivector generated transformations are fairly straightforward which makes the building of an equivalent scene-graph based system feasible. However, this thesis concentrates instead on exploring the nature of transformations generated by bivectors. This is because transformations that seemed straightforward to implement with matrices in the Euclidean context proved to be far from straightforward when using geometric algebra in a non-Euclidean context.

The question arose as to which types of transformations should form the core of an equivalent geometric algebra based system. Translations seemed an obvious first choice - they would form the basis of any navigation controls and would also play a role in mouse dragging. Chapter 5 considers bivector generated navigation controls.

Because the in-scene transformers and multi-transformers described in this chapter are major new innovations, it was felt that the transformations at their core, rotation about a fixed point and reflection in a line, should also be central to any new geometric algebra based system. Reflections are relatively easy to implement so are not discussed in this thesis (see introduction). However, this is not true of rotations. Chapters 6 to 8 explore the nature of bivector generated rotations and dilations since, as these chapters show, these transformations are naturally linked.

The chapter showed how the in-scene transformers could be used to support a range of geometric activities, including path tracing, tessellation building and puzzles based on rotations and reflections. These innovations resulted from using the scene graph architecture. Other ideas resulting from the choice of architecture included facilities for attaching locus points and co-ordinate grids to objects, and directional compass, ruler and protractor to the scene. In addition, multiple viewports (with one possibly showing an avatar view) were also made feasible. This was in addition to the potential for the architecture to support the construction of robot-like structures. Provided that any new geometric algebra based system was scene graph based, most of these ideas would carry over quite naturally.

Finally, the chapter briefly alluded to facilities for an in-scene 'affine transformer' that continuously transforms the shape of an object (section 2.8). If the transformer is attached to the scene, it continuously transforms the underlying geometry, giving the appearance of a viewport transformation. The

implied relationship between geometry and viewport transformation is explored in a non-Euclidean context in chapter 9.

3 A Simple Clifford Processor

3.1 Introduction

This chapter gives a brief account of Java software that was designed to generate the main products of a Clifford algebra of any signature. The design assumes the base space has a set of base vectors orthogonal in the sense that they anti-commute. These are denoted by e1, e2, e3, etc. Bases of the higher grade spaces are generated from these and denoted by e12, e13, e23, etc. The pseudoscalar is denoted by I. The adopted view of geometric algebra broadly follows that of [8] and [9].

Various Clifford algebra software systems have been developed. A key issue in their design has been the construction of a scripting language in which the user 'types in the algebra' (the user interface) and the kernel or 'engine' 'does the processing'. Final designs reflect the focus (interface versus kernel) as well as the programming environment, particularly whether mathematically or symbolically oriented, and the degree of support for the construction of a scripting language. For these, and other historical reasons, implementations vary considerably. Object orientation seems to be a minor issue relevant to those implementations based on or aimed at C++.

One of the earliest systems seems to be 'Clical' (Clifford Calculator), a DOS based system with command-line input [10]. Josep M Parr and Llorenç Roselló described early developments based on Mathematica [11]. More recently 'Clifford' and 'Glyph' used the symbolic processing language Mable V [12,13]. The system 'Gable' uses Matlab [14, 15] and a derivative 'Caigen' generates C++ source code [16]. Arvind Raja describes object oriented approaches developed directly in C++ [17].

Most of the more powerful systems are dimension and/or signature specific to various degrees. Partly because of that, this work is a deliberate 'fresh start' aimed at handling any dimension and any signature.

The resulting source code is included in a single class called GA, listed in full in appendix B. Extracts discussed in this chapter are listed as appropriate.

3.2 Computational background

As indicated, this section cites results drawn from various sources but particularly [8, chapters 1 and 2] and [9, chapters 1, 2 and 4]. There is no attempt to provide algebraic justifications since these invariably depend on the adopted axiomatic approach which is not a key issue.

For any two *vectors* v_1 and v_2, the geometric, inner (dot) and outer (wedge) products are related by

$$v_1 v_2 = v_1 . v_2 + v_1 \wedge v_2 \qquad (3.1)$$

Using the fact that the dot product is commutative and the wedge product anti-commutative, this is equivalent to describing the dot and wedge products as

$$v_1 . v_2 = \tfrac{1}{2}(v_1 v_2 + v_2 v_1) \qquad (3.2)$$

$$v_1 \wedge v_2 = \tfrac{1}{2}(v_1 v_2 - v_2 v_1) \qquad (3.3)$$

Thus, for vectors, it is possible to describe these 'derived' products in terms of the geometric product. This notion can be extended - the derived products for two homogeneous (single grade) multivectors A_r and B_s of grade r and s are given by

$$A_r . B_s = \langle A_r B_s \rangle_{|r-s|} \qquad (3.4)$$

$$A_r \wedge B_s = \langle A_r B_s \rangle_{r+s} \qquad (3.5)$$

Where $\langle M \rangle_k$ is the k-grade part of M.

In fact, the decomposition (3.1) generalises to

$$A_r B_s = \langle A_r B_s \rangle_{|r-s|} + \langle A_r B_s \rangle_{|r-s|+2} + \ldots \quad \ldots + \langle A_r B_s \rangle_{r+s}$$

The lowest and highest grade parts are the dot and wedge products respectively, though these and other grade parts may evaluate to zero.

By linearity, the result (3.4) and (3.5) generalises to two not necessarily homogenous multivectors M and N as

$$M \cdot N = \sum_{i,j} \langle M \rangle_i \cdot \langle N \rangle_j$$

$$M \wedge N = \sum_{i,j} \langle M \rangle_i \wedge \langle N \rangle_j$$

This is used to construct the corresponding dot- and wedge-product functions dp() and wp().

3.3 Generating the Cayley table

The software generates a Cayley table and uses it for internal look-up. The table can also be printed for validation. Two examples follow:

Example 1 : Generating the quarternion algebra

The quarternions are the even algebra of Cl(3) with suitable name changes. The following code generates the Cayley table for Cl(3).

```
GA ga = new GA(3);
ga.printTable();
```

The output is shown in table 3.1.

1	←------ vectors ---->			←--- bi-vectors --→			
	e1	**e2**	**e3**	**e12**	**e13**	**e23**	**I**
e1	1	e12	e13	e2	e3	I	e23
e2	-e12	1	e23	-e1	-I	e3	-e13
e3	-e13	-e23	1	I	-e1	-e2	e12
e12	-e2	e1	I	-1	-e23	e13	-e3
e13	-e3	-I	e1	e23	-1	-e12	e2
e23	I	-e3	e2	-e13	e12	-1	-e1
I	e23	-e13	e12	-e3	e2	-e1	-1

Table 3.1 The Cayley Table for Cl(3).

The even sub-algebra is generated by adding an extra '+' parameter to the GA constructor:

```
GA ga = new GA(3,'+');
ga.printTable();
```

This produces table 3.2.

1	**e12**	**e13**	**e23**
e12	-1	-e23	e13
e13	e23	-1	-e12
e23	-e13	e12	-1

Table 3.2 The Cayley Table for the even sub-algebra of Cl(3).

To match the classical form of the quarternions, names of the grade-2 base vectors are changed respectively to p, -q and r. These bivectors are the fourth, fifth and sixth elements of the base element set

$$1, e_1, e_2, e_3, e_{12}, e_{13}, e_{23}, e_{123}$$

The table is generated and the names changed using

```
GA ga = new GA(3,'+');
ga.changeName(4,"p",true);
ga.changeName(5,"q",false);
ga.changeName(6,"r",true);
ga.printTable();
```

The last boolean parameter in the changeName() function accounts for the sign change - traditionally q corresponds to $e_{31} = -e_{13}$.

This produces the classical table for quarternions shown in table 3.3.

1	p	q	r
p	-1	-r	q
q	r	-1	-p
r	-q	p	-1

Table 3.3 The Cayley Table for the quarternions.

Example 2 : Generating the complex numbers

Complex numbers are isomorphic to the even algebra of Cl(2). Their Cayley table is generated by the code

```
GA ga = new GA(2,'+');
ga.changeName(3,"i",true);
ga.printTable();
```

which produces table 3.4.

1	i
i	-1

Table 3.4 The Cayley Table for the complex numbers.

3.4 Internal representation of base vectors and multivectors

Base vectors are represented internally by byte arrays with values of 0 or 1, with the first position being reserved for sign. (Java does not support bit-arrays.) Multivectors are represented by double-precision floating point arrays. Figure 3.1 shows these arrays for Cl(3).

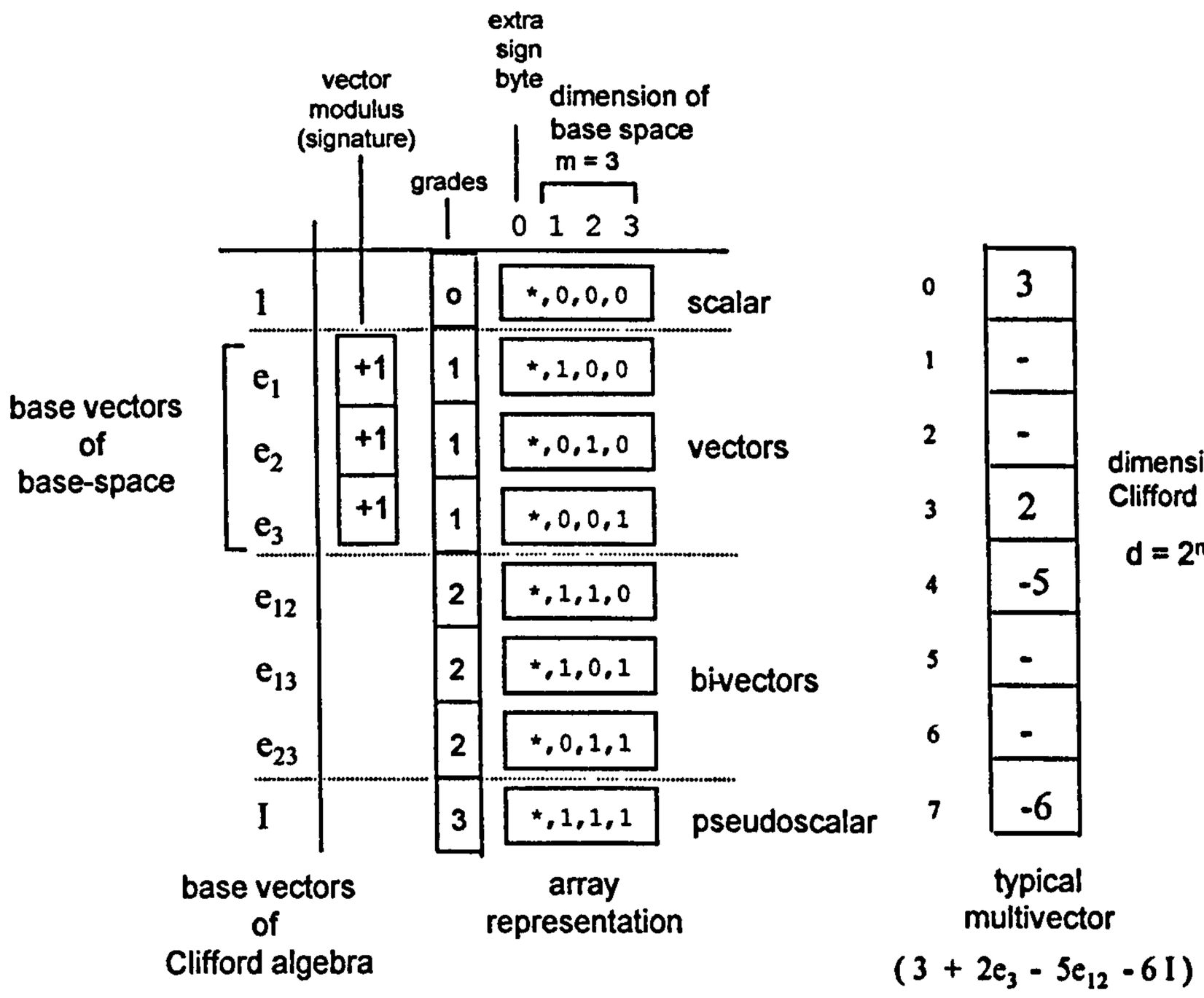


Figure 3.1 Internal representation of base elements of the 8-dimensional Clifford algebra Cl(3).

3.5 Filling the base vector array

The software uses a non-recursive algorithm to fill the basis[] array. The example of figure 3.2 shows the filling sequence for the grade-3 base elements of Cl(6):

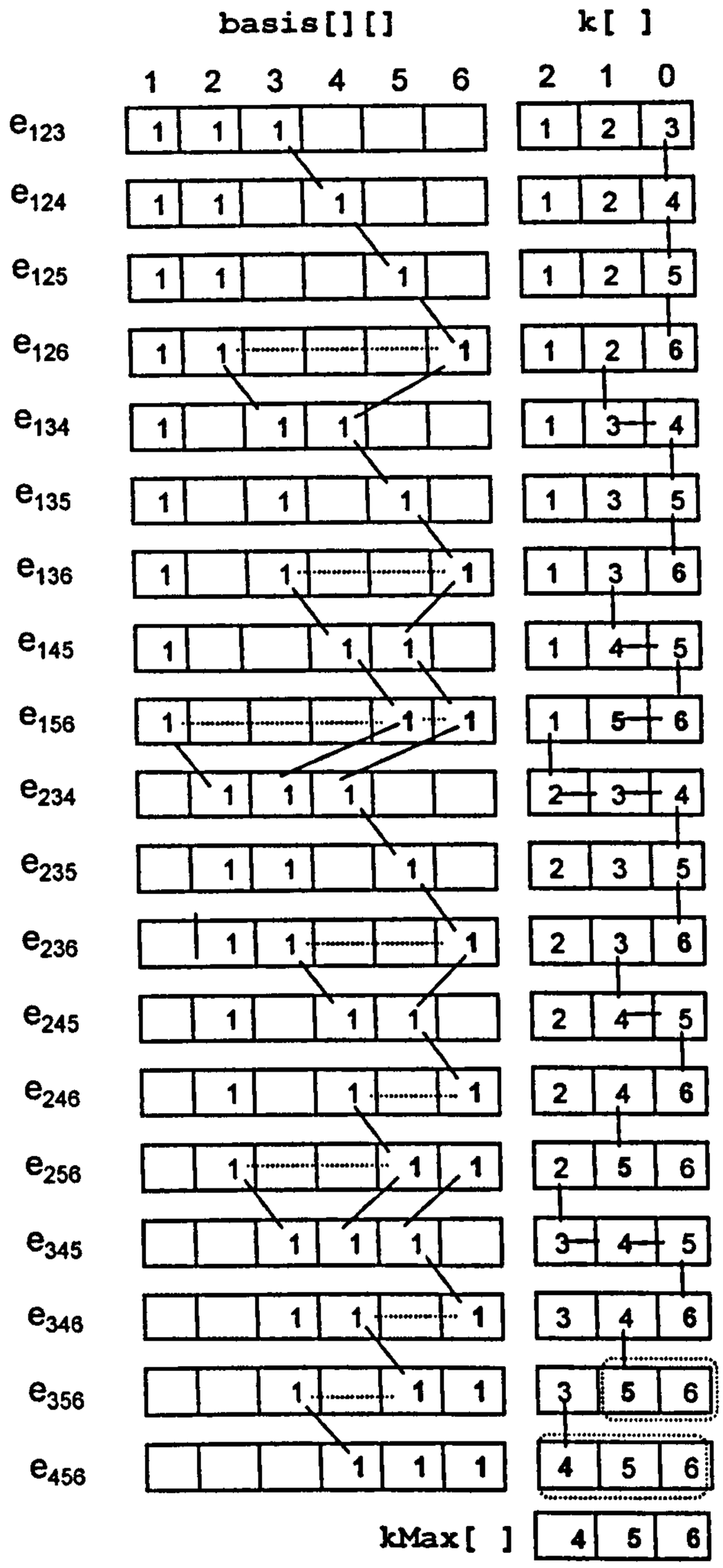


Figure 3.2 Algorithm for filling the array k[ ] which, when filled, contains the index set of all base trivectors of Cl(6).

The array k[] stores the grade-3 index set which initially hold the values 1,2,3. The final values are 4,5,6. These represent the maximum values for each array position and are stored as a separate array kMax[]. The algorithm increments the right-most index until it reaches its maximum value whereupon the one on its left is incremented. If that in turn reaches its maximum value, the one on its

left is incremented, and so on. The indices to the right of the last index to be incremented are then set to have sequential values.

Assuming space parameters are defined as

```
int m = 6;                              // base-space dimension
int d = (int) Math.pow(2,m);     //  dimension of Clifford algebra
byte[][] basis  = new byte[d][m+1];
byte[]  grades  = new byte[d];
```

the code for this algorithm is

```
static void createBasisVectors()
{
   for (int i=0; i<d; i++) basis[i][0] = +1;  // set sign

   int count = 0;
   byte grade = 1;

   while (grade<=m)
{
   byte n = grade;

   int[] k    = new int[n];
   int[] kMax = new int[n];

   for (int i=0; i<n;  i++) k[i] = i+1;
   for (int i=0; i<n;  i++) kMax[i] = m - n + i +1 ;

   count++;
   for (int i=0; i<n; i++) basis[count][k[i]] = 1;
   grades[count] = n;

while (k[0]< kMax[0])
{
   k[n-1]++;

   int i = n-1;
   while (i>=0)
   {
     if (k[i]>kMax[i])
     {
```

```
k[i-1]++;

int j = i;
while( j<n)
{
   k[j] = k[j-1]+1;
   j++;
 }
}
```

```
            i--;
          }

          count++;
          for (int j=0; j<n; j++) basis[count][k[j]] = 1;
          grades[count] = n;
        }

        grade++;
      }
    }
```

3.6 Calculation of Cayley table products: multiplying base vectors

The algorithm for evaluating products in the Cayley table utilises the fact that indexes in all base vectors are in numerical order.

To illustrate, the calculation below evaluates the product of e_{1245} and e_{135} in Cl(4,1) where, because of the nature of the signature, $e_1^2 = e_2^2 = e_3^2 = e_4^2 = 1$ and $e_5^2 = -1$;

$$
\begin{aligned}
& e_{1245}e_{135} \\
= \; & (e_1e_2e_4e_5)\,(e_1e_3e_5) \\
= \; & -(e_2e_1e_4e_5)\,(e_1e_3e_5) \\
= \; & +(e_2e_4e_1e_5)\,(e_1e_3e_5) \\
= \; & -(e_2e_4e_5e_1)\,(e_1e_3e_5) \\
= \; & -(e_2e_4e_5)\,(+1)\,(e_3e_5) \\
= \; & -(e_2e_4e_5)\,(e_3e_5) \\
= \; & +(e_2e_4e_3e_5)\,(e_5) \\
= \; & -(e_2e_3e_4e_5)\,(e_5) \\
= \; & -(e_2e_3e_4)\,(-1) \\
= \; & +(e_2e_3e_4) \\
= \; & +e_{234}
\end{aligned}
$$

3 sign changing swaps

2 sign changing swaps

Starting with subscript 1, the e_1 vector in the left bracket also appears in the right bracket, so is swapped past the remaining vectors in its bracket which induces three sign changes.

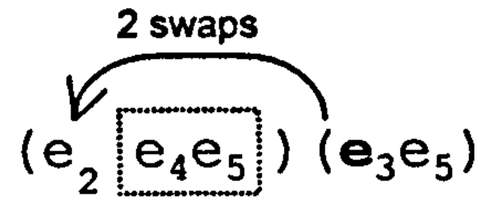
3 swaps

$\mathbf{e_1}\,e_2\;e_4e_5\;)\,(e_1e_3e_5)$

It is then adjacent to the e_1 vector in the right bracket and so annihilates, leaving +1.

Considering subscript 2 vectors, the e_2 vector in the left bracket is not repeated in the right hand bracket and is therefore left unaltered.

The situation with the subscript 3 vectors is different - an e_3 vector appears only in the right bracket. To bring it into the correct overall position it is swapped past the higher subscript vectors in the left bracket:

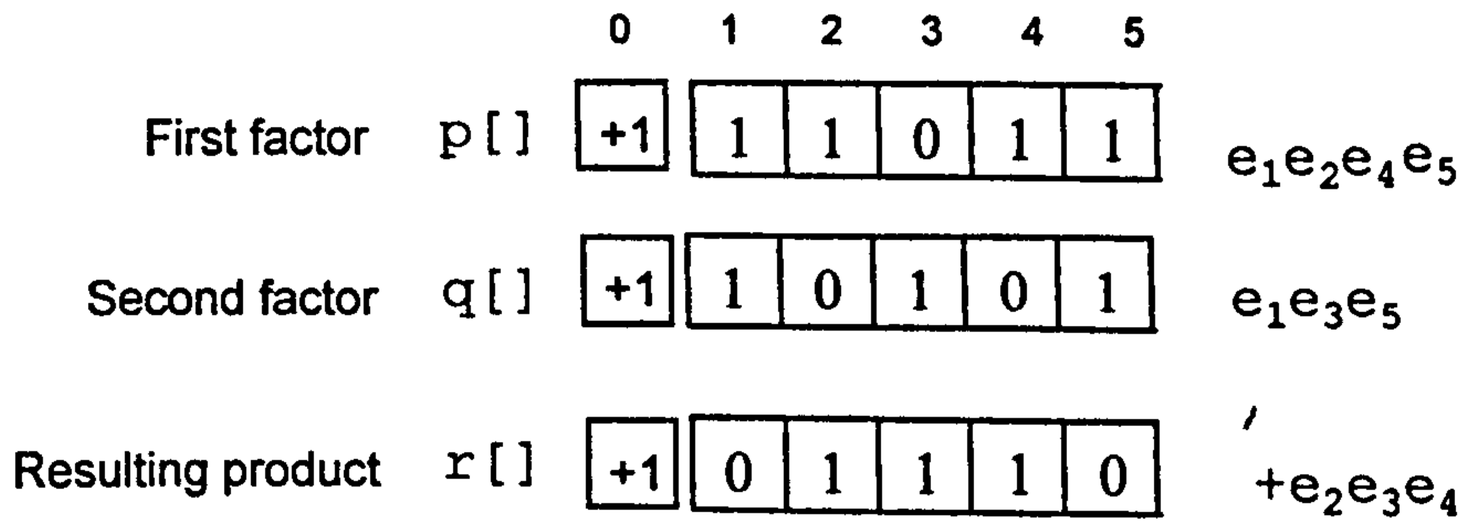


In this case, the even number of swaps do not induce a sign change.

When a vector appears in the left bracket only, it remains where it is. When it appears in both brackets, or just the right bracket, swaps have to be counted. In both cases, the number of swaps (i.e. sign changes) correspond to the number of remaining vectors in the left bracket.

Put another way, if a vector does not appear in the second bracket then no swapping is required. If the same vector appears in the first bracket then it remains where it is. On the other hand, if a vector does appear in the second bracket then swaps need to be counted. If the same vector also appears in the first bracket then they ultimately annihilate and induce an overall sign change according to the value of the square of the vector. This occurs only if the vector appears in both brackets so this can be handled separately.

The algorithm uses arrays to represent the bracketed factors and the resulting product:



These arrays are part of the table multiply() function where table elements are passed by index value:

```
byte[] multiply(int i, int j) // table multiplication
{
```

```
byte[] p =   basis[i];
byte[] q =   basis[j];
byte[] r =   new byte[m+1];
byte sign = +1;

for (int k=1; k<=m; k++)
{
   if (p[k]==1 && q[k]==1) sign = (byte) (sign*vectorModulus[k]);
   if    (q[k]==0) r[k] = p[k];
   else
   {
      r[k] = (byte) (1 -  p[k]);

      if (sign != 0)
         for (int j=k+1; j<=m; j++)              // count swaps
            if (p[j]==1) sign = (byte) (-sign);
   }
}

if (p[0]==-1 && q[0]==+1)  sign = (byte) -sign;
if (p[0]==+1 && q[0]==-1)  sign = (byte) -sign;
   r[0] = sign;
   return r;
}
```

3.7 Calculating the geometric product

The geometric product function gp() uses the table multiply() function to multiply basis vectors and a findBasisVectorIndex() function to calculate the index of the result so that products can be accumulated:

```
double[] gp(double[] m1, double[] m2)  // geometric product
{
   double[] r = new double[d];
   byte[] p;

   int d1 = Math.min(d, m1.length);
   int d2 = Math.min(d, m2.length);

   for (int i=0; i<d1; i++)
```

```
    for (int j=0; j<d2; j++)
       if (m1[i]!=0 && m2[j]!=0)
       {
          p = multiply(i,j);
          int k = findBasisVectorIndex(p);
          if ( p[0]==+1)  r[k] = r[k] + m1[i]*m2[j];
          if ( p[0]==-1)  r[k] = r[k] - m1[i]*m2[j];
       }
return r;
}
```

3.8 Calculating the dot and wedge products

The functions for calculating the wedge and dot products, wp() and dp(), use the gp() function. For homogeneous arguments, the functions are titled wpH() and dpH() and are:

```
double[] wpH(double[] m1, double[] m2) // wedge product
{
   if ( !isHomogeneous(m1)) return null;
   if ( !isHomogeneous(m2)) return null;

   return gradePart(gp(m1,m2), grade(m1) + grade(m2));
}

double[] dpH(double[] m1, double[] m2) // dot product
{
   if ( !isHomogeneous(m1)) return null;
   if ( !isHomogeneous(m2)) return null;

   return gradePart(gp(m1,m2),
                    Math.abs(grade(m1)-grade(m2)));
}
```

These return 'null' if either of the arguments is not homogeneous. The operation of these functions can be seen in the following example code which constructs all three products for two grade-2 multivectors mv1 and mv2:

```
GA ga = new GA(4);
                    // e1  e2  e3 e4  e12 e13 e14 e23 e24 e34
double[] mv1 = {0,   0,  0,  0, 0,   0,  0,  1,  0,  5,  4 };
```

```
double[] mv2 = {0,   0,  0,  0, 0,   2,  3,  0,  0,  0,  0 };
ga.show("mv1 = ", mv1, false);
ga.show("mv2 = ", mv2, false);
ga.show("mv1^mv2 = ", ga.wpH(mv1,mv2), false);
ga.show("mv1.mv2 = ", ga.dpH(mv1,mv2), false);
ga.show("mv1mv2 = ", ga.gp(mv1,mv2), false);
```

This produces the output:

```
mv1 =  + 1.0 e14 + 5.0 e24 + 4.0 e34
mv2 =  + 2.0 e12 + 3.0 e13
mv1^mv2 =  - 7.0 I
mv1.mv2 = 0
mv1mv2 =  - 22.0 e14 + 2.0 e24 + 3.0 e34 - 7.0 I
```

The dot and wedge products are manifested in the geometric product as the 0-grade and 4-grade parts, namely 0 and -7.0 I. If the data were changed to

```
                    // e1  e2  e3 e4  e12 e13 e14 e23 e24 e34
double[] mv1 = {0,   0,  0,  0, 0,   0,  0,  1,  3,  5,  0 };
double[] mv2 = {0,   0,  0,  0, 0,   2,  0,  0,  0,  0,  1 };
```

the results become:

```
mv1 =  + 1.0 e14 + 3.0 e23 + 5.0 e24
mv2 =  + 2.0 e12 + 1.0 e34
mv1^mv2 = 0
mv1.mv2 = 0
mv1mv2 =  - 7.0 e13 - 10.0 e14 - 5.0 e23 + 5.0 e24
```

In this case, the dot and cross products are both zero, reflected in the fact that the geometric product has no grade-0 or grade-4 term. (Geometrically, the linear spaces represented by mv1 and mv2 intersect in more than just the origin yet are 'perpendicular').

The generalised non-homogeneous versions of these functions use the homogeneous versions, for example

```
double[] wp(double[] m1, double[] m2) // wedge or outer product
{
    double[] r = new double[d];

    for (int i=0;i<=m;i++)
       for (int j=0;j<=m;j++)
            r = add(r, wpH(gradePart(m1,i),gradePart(m2,j)));

    return r;
}
```

This uses the add() function to accumulate the results obtained from forming the products on the homogeneous grade parts.

3.9 Constructing blades

To construct blades, i.e. the wedge product of many vectors, the software makes use of the associativity of the wedge product:

$$v_1 \wedge v_2 \wedge v_3 = ((v_1 \wedge v_2) \wedge v_3)$$

$$v_1 \wedge v_2 \wedge v_3 \wedge v_4 = (((v_1 \wedge v_2) \wedge v_3) \wedge v_4).$$

The corresponding function is called createBlade():

```
double[] createBlade(double[][] v)
{
    for (int i=0; i<v.length; i++)
        if (!isVector(v[i])) return null;

    double[] r = v[0];

    for (int i=1; i<v.length; i++)
       r = wp(v[i],r);

    return r;
}
```

Vectors are passed to the function as arrays of multivectors. The result 'r' is seeded with the initial vector so that progressive wedge product can be calculated.

To partially validate this function, an alternative approach uses the formulation cited in [9, page 91],

$$a\wedge(a_1\wedge a_2\wedge a_3\wedge\cdots\wedge a_r)=\frac{1}{2}\left(a(a_1\wedge a_2\wedge a_3\wedge\cdots\wedge a_r)+(-1)^r(a_1\wedge a_2\wedge a_3\wedge\cdots\wedge a_r)a\right).$$

The corresponding function is:

```
double[] createBlade2(double[][] v)
{
    for (int i=0; i<v.length; i++)
            if (!isVector(v[i])) return null;

    double[] Ar = v[0];
    int sign = -1;
    for (int i=1; i<v.length; i++)
    {
        if (sign==+1) Ar = mul(add(gp(v[i],Ar), gp(Ar,v[i])),1/2.0);
        if (sign==-1) Ar = mul(sub(gp(v[i],Ar), gp(Ar,v[i])),1/2.0);
        sign = -sign;
    }

    return Ar;
}
```

Addition, subtraction and scalar multiplication (by ½) are performed with the functions mul(), add() and sub(). The sign generated by the $(-1)^r$ factor alternates as r increases.

The following partial validation routine randomly creates sets of vectors and passes them to both versions of the blade creation function for subsequent comparison:

```
GA ga = new GA(3,2,1);  // generates Cl(3,2,1)
```

```
for (int i=0; i<100;i++)
{
      int k = (int) ((ga.m)*Math.random())+1;  // number of vectors
      double[][] v = new double[k][ga.m+1];

      for (int j=0; j<k; j++) v[j] = ga.randomVector();

      double[] blade1 = ga.createBlade(v);
      double[] blade2 = ga.createBlade2(v);

      if (!ga.isEqual(blade1,blade2)) System.out.println("error!");;
}
```

This 100 times loop test was run successfully with Cl(i,j,k) for $0 < i,j,k < 5$.

3.10 The dot product and square of vectors

If u and v are vectors then u.v is a scalar - being of equal grade, the dot product of u and v is the zero grade part of the geometric product. Equally,
$v^2 = vv = v.v$ is also a scalar.

However, if the dp() function is used to calculate the dot product of two vectors the result is a multivector representing a scalar, i.e. an array with elements all zero except for the first. Rather than have a special vector-specific dp() function that return a single scalar, the software utilises the fact that if a multivector is a scalar, its value is stored in the first position with array index zero. For example, if u and v are vectors, dp(u,v) returns an array of doubles representing a multivector, whereas dp(u,v)[0] is a pure double.

3.11 Converting a scalar to a multivector

The Cl(3,1) implementation includes the multivector

$$i = (1,\ \ 0,0,0,0,\ \ 0,0,0,0,0,0,\ \ 0,0,0,0,\ \ 0).$$

An arbitrary scalar k can be converted to a multivector by multiplying it by i.

3.12 Conclusion

The software of this chapter was written as a validation exercise in two senses. First, it was used to validate personal conceptual understanding of the mathematical language and ideas in which key formulae were expressed. In this capacity it succeeded in providing considerable insights into the nature of geometric algebra.

Secondly, it is the basis for the graphics software of the next chapter. This was used throughout the remainder of the thesis to generate graphical screen output in order to validate theoretical ideas.

4 Implementing the Conformal Model Cl(3,1)

4.1 Introduction

This chapter describes an extension to the GA class of the previous chapter. It specifically limits the Clifford algebra to Cl(3,1) but adds functionality for drawing circles, geodesic arcs and lines using the various 2D models of non-Euclidean space described in chapter 1. The geometry is determined by any one of the following 'type' vectors all of which, except the last, are described in Chapter 1.

```
double[] n  = {0,   0, 0, 1, 1};  // R2,  standard Euclidean

double[] e  = {0,   0, 0, 0, 1};  // S2, hemi-sphere model
double[] a  = {0,   0, 0, 1, 0};  // H2, Poincare Disc model
double[] b  = {0,   0, 1, 0, 0};  // H2, half-space model
double[] c  = {0,   1, 0, 0, 0};  // H2, half-space model
```

The new CM class also adds functions for creating vector and blade-based conformal representations of circles, and for extracting information from these representations. The resulting source code is listed in full in appendix C. Extracts discussed in this chapter are listed as appropriate.

4.2 Functions for mapping points between R^2 and the null cone

The mapping from R^2 to the null cone is

$$f: x \rightarrow x = (x^2 - 1)a + 2x + (x^2 + 1)e$$

This is implemented by:

```
double[] F(double[] p)
 {
```

```
        double[] t1 = mul(a, dp(p,p)[0]-1);
        double[] t2 = mul(p, 2);
        double[] t3 = mul(e, dp(p,p)[0]+1);

        return add(add(t1,t2),t3);
    }
```

In the case of Cl(3,1), the reverse map can be achieved by scaling the conformal point X to the representation of R^2, by multiplying by –2/(X.**n**), then ignoring the 3rd and 4th **a** and **e** components and halving the result. This can be achieved with

```
double[] f(double[] X)
{
      double[] d = new double[16];
      for (int k=0; k<16; k++) d[k] = scale(X, n)[k];
      d[3] = 0; d[4] = 0;
      return d;
}
```

This uses the following function for null cone scaling

```
double[] scale(double[] P, double[] type)
{
   return div(P, -dp(P,type)[0]);
}
```

However, an alternative more generic approach is to use the projection formulae [4, page 380]

$$x = -\frac{X \wedge N}{X \cdot \mathrm{n}} N,$$

where N = ea = e^a is the bi-vector blade representing the Minkowski plane appended to R^2. The formula effectively projects out of this plane.

This more generic operation is the one that is adopted and implemented as

```
double[] f(double[] X)
```

```
{
 double    k = dp(X,n)[0];
 double[]  x = mul(X, -1.0/k);

 double[] N = gp(e,a);
 return gp(wp(x,N),N);
}
```

4.3 Constructing and measuring circles

Although a circle is represented conformally by a single vector, the centre and radius of a particular realisation are dependent on the underlying geometry in which it occurs. Thus the functions that measure a circle's properties take a second geometry-defining 'type' vector as an extra parameter. These are

```
double getRadius(double[] S, double[] type)
{
 return Math.sqrt(dp(S,S)[0]/(dp(S,type)[0]*dp(S,type)[0]));
}
```

```
double[] getCentre(double[] S, double[] type)
{
    double R = getRadius(S,type);
    return add(S, mul(type, R*R));
}
```

These two functions reflect the formulae cited in Chapter 1,

$$\mathbf{r}^2 = \mathbf{s}^2/(\mathbf{s}.\mathbf{n})^2$$

$$\mathbf{c} = \mathbf{s} + \rho^2\mathbf{n}.$$

The latter function uses the symbol R in place of ρ. It returns a null vector which is the conformal representation of the centre with respect to the geometry in question – the geometry specified by the 'type' vector.

Conversely, when constructing the 'abstract' conformal vector representation of a circle from its properties defined in a specific geometry, its 'type' vector is needed to ensure the correct construction. The function is

```
double[] makeCircle(double[] C, double R, double[] type)
{
    return sub(C, mul(type, R*R));
}
```

This assumes that the conformal centre C and radius R are correctly specified with respect to the implied geometry. This may require that the centre undergoes appropriate null-cone scaling using the 'type' vector before being passed to this function. The function is based on the inverse formula

$$c = C - nR^2.$$

4.4 Computational example 1: Drawing grids of circles

The following code produces the screen shots shown in figures 1.17, page 29 and 1.23, page 35:

```
double[]  type = ga.e;

double[] origin = {0, 0,0,0,0};
ga.drawCircle(g, origin, 1);

double R = 0.1, r;    // R = spherical radius of spherical circles
double[] P,Q,S,C,c;

for (int i= -3; i<=3; i++)
for (int j=-3; j<=3; j++)
{
   double[] p = {0,  0.2*i,0.2*j,0,0 };

   P = ga.F(p);                      // 1: project on to null cone
   Q = ga.scale(P,type);             // 2: null cone scaling
   S = ga.makeCircle(Q,R,type);      // 3: construct conformal circle
   r = ga.getRadius(S, ga.n);        // 4: extract Euclidean (E.) radius
   C = ga.getCentre(S, ga.n);        // 5: get conformal rep of E. centre
```

```
    c = ga.f(C);                        //    get actual Euclidean centre
    ga.drawPoint(g,p);
    ga.drawCircle(g,c,r);
}
```

The numbered comments correspond to the 'steps' in the computational example of chapter 1. However, upper case letters are being used here to represent conformal or non-Euclidean elements. Lower case letters are used for Euclidean elements needed by the drawing functions.

By changing the value of the 'type' vector, different models of spherical and hyperbolic space can be easily realised. The code uses point and circle drawing functions which are passed to the current graphics context as a first parameter.

4.5 A function to draw circles in non-Euclidean space

The routine above is encapsulated into the following circle-drawing function

```
void drawCircle(Graphics g, double[] centre,
                double radius, double[] type)
{
   double[] c = F(centre);
   double[] circleBlade = makeCircle(scale(c,type),radius,type);
   double[] C = getCentre(circleBlade, n);
   double   R = getRadius(circleBlade, n);
   drawCircle(g, f(C), R);
   drawPoint(g,centre);
}
```

The last vector parameter specifies the underlying geometry. The function draws a circle in the specified geometry around a point specified in viewport co-ordinates.

4.6 Computational example 2: Drawing concentric circles

To demonstrate the drawCircle() function, the following code draws concentric circles of radius 0.1, 0.2, 0.3... around the point with viewport co-ordinates (0.4,0.5)

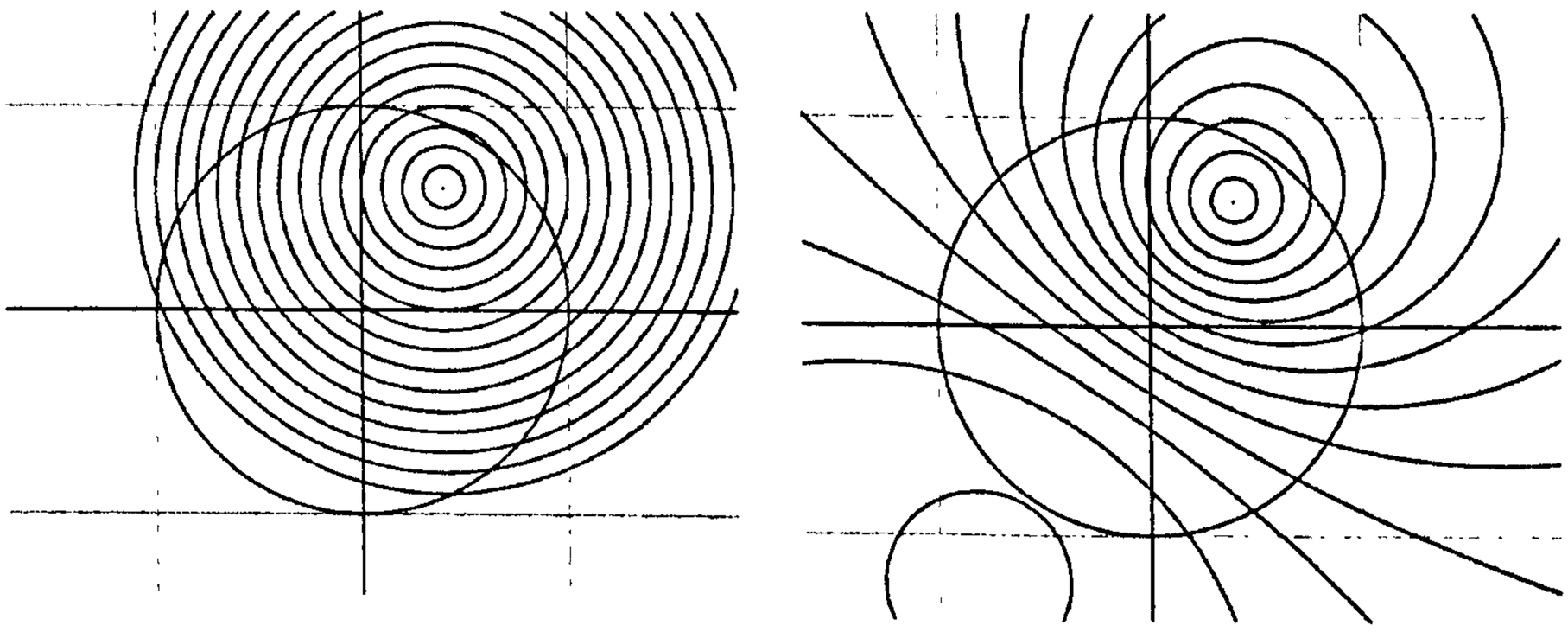
```
double x = 0.4;
double y = 0.6;
double R = 0.1;

double[] centre = {0,    x, y, 0, 0};

ga.drawPoint(g, centre);

for (int i=0; i<15; i++)
{
   ga.drawCircle(g, centre, R, type);
   R = R + 0.1;
}
```

This produces the output shown in figures 4.1 and 4.2 dependent on the value of the type vector.



Euclidean space

spherical space
(hemi-sphere model)

Figure 4.1 Concentric circles in Euclidean and spherical space (hemisphere model).

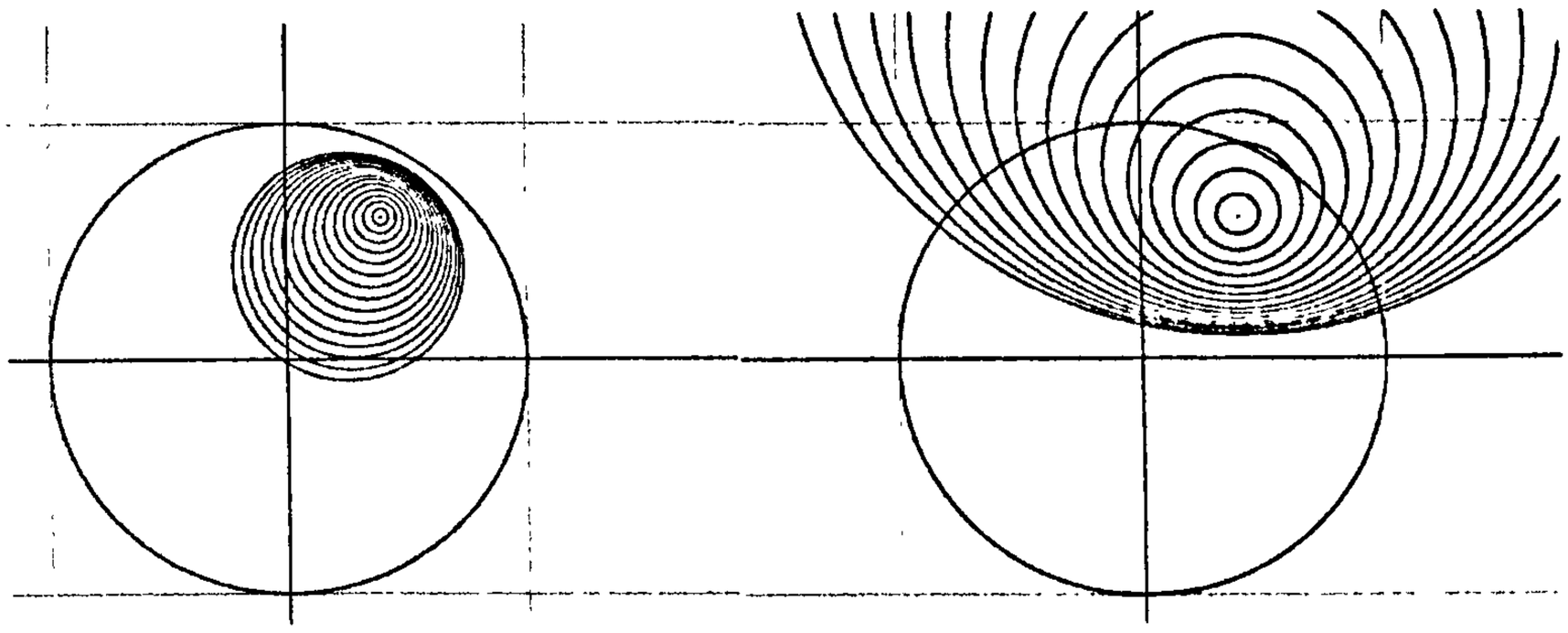


Figure 4.2 Concentric circles in hyperbolic space.
(Poincaré disc and half space models.)

4.7 Problems of scale

There appear to be problems of scale in relation to the drawCircle() function – in certain cases the radii of the circles are proportionally too large or too small. In the Euclidean case, normalising the type vector **n** appears to remedy the problem (a second normalised vector **m** is provided for this purpose).

4.8 A function to draw geodesic arcs in non-Euclidean space

The drawCircle() function is viewport specific so can be used to write a mouseDrag() function that continuously redraws a non-Euclidean circle around the current mouse-pointer position, making it appear as though the circle was being dragged through non-Euclidean space. The circle's screen size and central-offset would continuously change depending on its position.

However, this approach is not ideal. To accommodate a scene graph architecture it would be preferable to use a transformation-based approach, i.e. to determine what transformation represents a mouse-movement and apply it to the circle. This will become one of the key themes of chapters 7 and 8.

Equally, it is possible to draw geodesics without considering (translational) transformations. Because geodesics are represented conformally by arcs of

circles which pass through the 'point at infinity' it is possible to draw them without specifying a metric or considering 'shortest path' routes. In each case, the conformal 'point at infinity' is the 'type' vector defining the geometry.

A circle through three (conformal) points is represented by the blade or tri-vector formed from their wedge product. In the case of a geodesic, one of these is the conformal 'point at infinity', the geometry-defining 'type' vector.

To obtain the viewport co-ordinates of the centre and radius of a geodesic arc joining points with viewport co-ordinates $\mathbf{p}_1$ and $\mathbf{p}_2$ would entail mapping $\mathbf{p}_1$ and $\mathbf{p}_2$ onto the null cone; constructing the appropriate blade using the geometry-defining point-at-infinity as the third point; extracting the (conformal representation of) the centre and mapping it back to Euclidean space of the viewport. This is coded in the drawArc() function:

```
.
double[][] D = {F(p1), F(p2), type};
double[] B = createBlade(D);

double[] centre = f(getCentreFromBlade(B,n));
double   radius = getRadiusFromBlade(B,n);
.
```

The function also extracts the Euclidean (viewport) radius of the circle represented by the blade. The centre and radius are then used to draw the arc.

4.9 Computational example 3: drawing quadrilaterals

The following example uses the drawArc() function to draw the geodesic sides of a quadrilateral

```
double[][] p = {{ 0,   0.3,   0.1 }, { 0,   0.5,   0.3 },
                { 0,   0.3,   0.7 }, { 0,   0.1,   0.5 }};

for (int i=0; i<4; i++)
{
 ga.drawPoint(g,p[i]);
```

```
        ga.drawArc(g,p[i], p[(i+1)%4], type, true);
    }
```

The final parameter 'true' passed to the drawArc() function causes the arc to be drawn superimposed on a complete geodesic circle, see figures 4.3 and 4.4

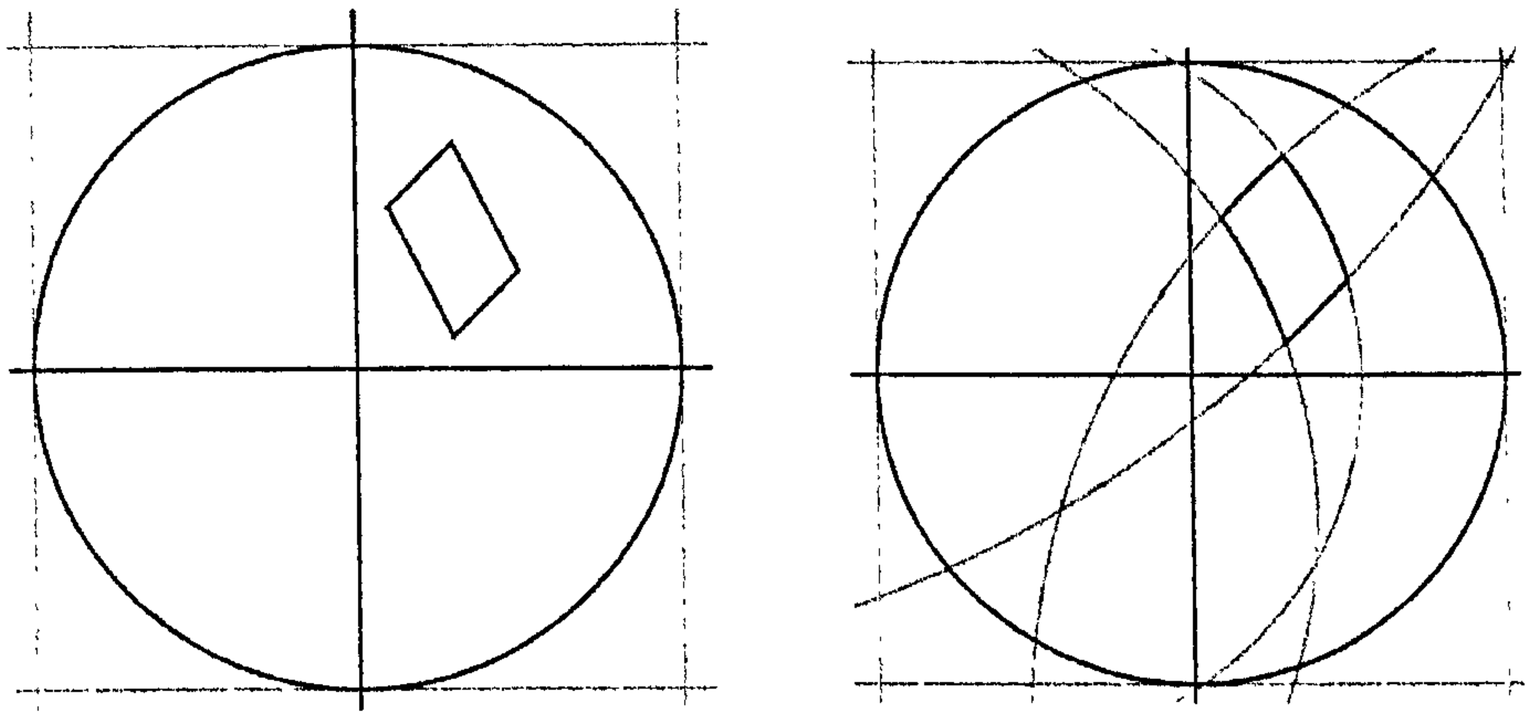


Figure 4.3 A quadrilateral in Euclidean and spherical space (hemisphere model).

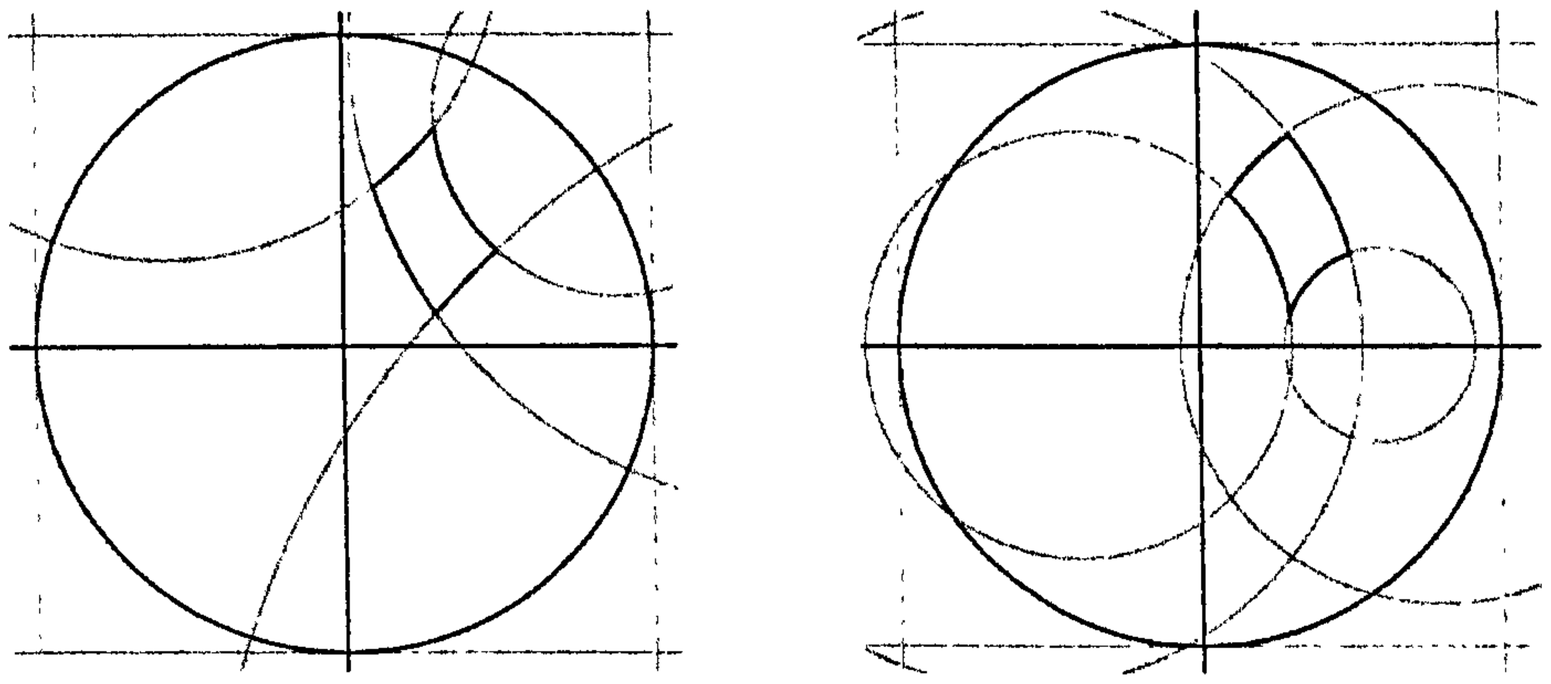


Figure 4.4 Quadrilaterals in hyperbolic space. (Poincaré disc and half space models.)

4.10 Implementing mouse dragging

Because point information for both the drawArc() and drawCircle() functions is specified in terms of viewport co-ordinates, these functions could be used to

implement simple mouse dragging. If the centre of a circle were dragged elsewhere, it would simply be a question of redrawing it. Thus it would be possible to drag a circle through space (by its centre) and watch its radius change accordingly, becoming smaller if it approached an 'horizon'. The same applies if one end of a geodesic arc was dragged. The 'line' joining the points would change in subtle ways.

Though expedient, this approach is severely limited. To implement generic mouse dragging requires calculating the transformation represented by a mouse drag so that, if necessary, it can be applied to all associated objects. This requires the implementation of translations defined by two points.

4.11 Implementing translations in the conformal model

Classical angle-preserving conformal transformations are often represented in conformal space by 'sandwich' transformations of the form

$$X \to e^{B/2} X e^{-B/2}, \tag{4.1}$$

where B is a bi-vector referred to in the literature as the 'generator of the transformation'. This is the approach adopted in [4] where the exponential (of a bivector) is assumed here to be the result of summing a classical Taylor series which is assumed to converge.

It is stated in [4, page 373] that if x and y are conformal representations of points in a base space with geometry specified by the type vector t, then the bi-vector

$$B = (x \wedge y \wedge t)t$$

generates a transformation that takes x to y and leaves t, the 'point at infinity', invariant. The transformation is therefore a translation. This stated result begs many questions, some of which are addressed in chapter 7. At this point, the

immediate concern is to test the result in the context of different geometries since, in [4], it is elaborated upon only in the context of the hyperbolic Poincaré disc model where, it is claimed, $B^2 > 0$. In fact, there is a strong hint in this and other literature, that the nature of the 'translation' is somehow determined by the sign of B^2. The indication being that it is the *translation*, rather than the *geometry of the base space*, that carries the epitaph 'hyperbolic', 'elliptic' or 'Euclidean'. At this stage it is difficult to source the roots of this differing perspective. This issue is resolved in chapters 8 and 9.

What seems to be tacitly assumed in [4] is that the generating bi-vector squares to a scalar whose sign determines the nature of the exponential Taylor expansion in a fairly straightforward way so that if $B = k\hat{B}$, where $\hat{B}^2 = 1$ then

$$\begin{aligned} \exp(B) = \exp(k\hat{B}) &= \cosh(k) + \hat{B}\sinh(k), && \text{if } B^2 > 0, \\ &= \cos(k) + \hat{B}\sin(k), && \text{if } B^2 < 0, \\ &= 1 + k\hat{B}, && \text{if } B^2 = 0. \end{aligned}$$

The following is an implementation of the exponential function as expressed above preceded by necessary precursor functions.

```
double cosh(double a) { return (Math.exp(a) + Math.exp(-a))/2; }
double sinh(double a) { return (Math.exp(a) - Math.exp(-a))/2; }
double cos(double a)  { return Math.cos(a); }
double sin(double a)  { return Math.sin(a); }

double[] exp(double[] b)  // use only for bi-vectors
{
double b_squared = gp(b,b)[0];

if (b_squared>0)  // hyperbolic
{
 double k  = Math.sqrt(b_squared);
 double[]  bHat  = div(b,k);

 return add(mul(i,cosh(k)), mul(bHat, sinh(k)));
}
```

```
else if (b_squared<0)  // elliptic/spherical
{
 double k  = Math.sqrt(-b_squared);
 double[]  bHat  = div(b,k);

 return add(mul(i, cos(k)), mul(bHat, sin(k)));
}
```

```
  else return add(i,b);   // Euclidean
}
```

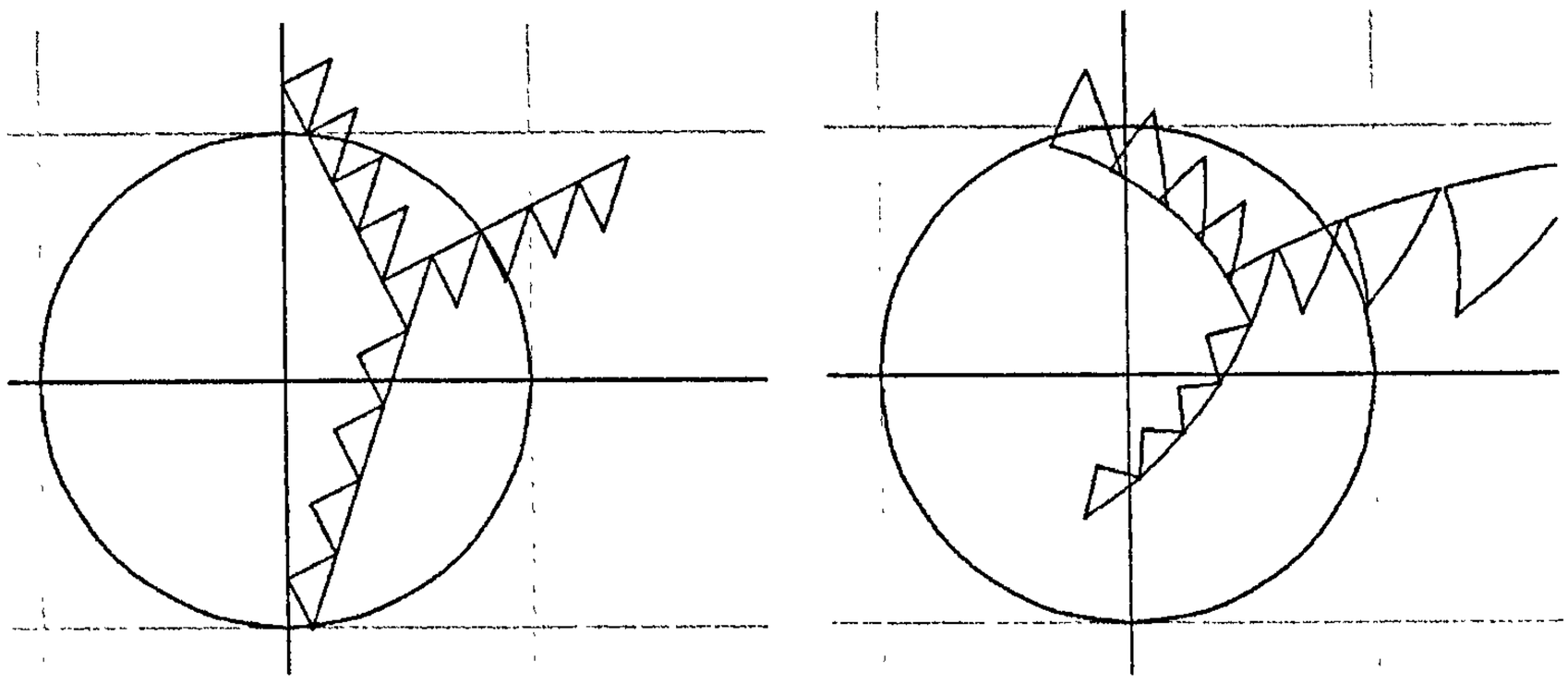
The vector i is a multivector representing the scalar value 1.

Using this exponential function, the following function implements the 'sandwich' transformation (4.1) with the scaling modification indicated:

```
double[] est(double[] X, double[] B)
{
    return gp(exp(div(B, -4)), X, exp(div(B, 4)));
}
```

4.12 Computational example 4: Translating a triangle

Using a scaling factor of 4, rather than 2, the preceding function seems to correctly translate points for Euclidean and spherical space. The test consisted of repeatedly translating a triangle in a direction corresponding to one of its sides, then repeating this for each side, see figure 4.5.



Euclidean space spherical space

Figure 4.5 Translating a triangle parallel to its three edges in Euclidean and spherical space (hemisphere model).

In both cases the 'line' formed by the translated touching sides does seem to form a geodesic 'curve'. In the Euclidean and spherical cases these geodesic curves are respectively straight lines and circles that meet the unit circle at antipodal points.

Inconsistencies in scaling also appear in the hyperbolic cases. Figure 4.6 depicts the half-space model.

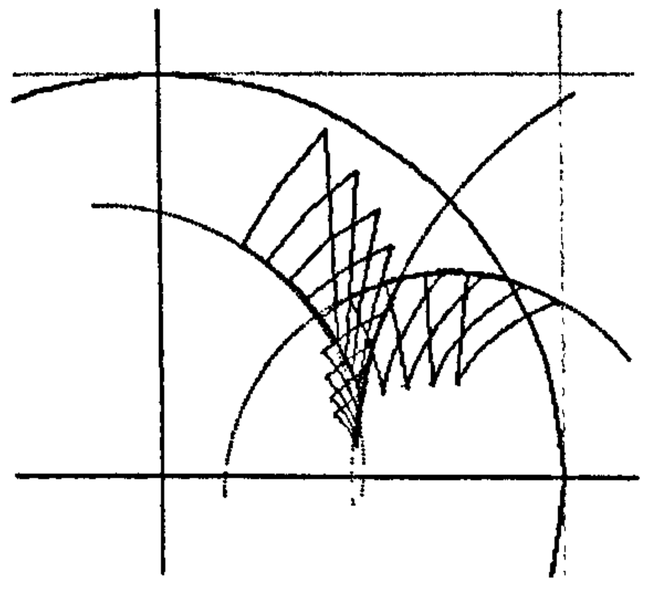


Figure 4.6 Translating a triangle parallel to its three edges in hyperbolic space (Poincaré disc model).

Though the scaling is problematic, here again the curves formed by the translated sides form geodesics, in this case circles perpendicular to the x-axis. These screen shots of figures 4.5 and 4.6 were produced by the code

```
type = ga.e;                    // defines geometry

double[][] p  = { {0,+0.6,+0.5},    // p[0]
                  {0,+0.5,+0.2},    // p[1]
                  {0,+0.4,+0.4}};   // p[2]

double[][] q = new double[3][3];

for (int i=0; i<3; i++)
{
 // construct generator for transform

 double[][] data = {ga.F(p[i]), ga.F(p[(i+1)%3]),type };
 double[] B = ga.gp(ga.createBlade(data),type);

 for (int m=0; m<3; m++)            // reset q-points
        for (int n=0; n<3; n++)
          q[m][n] = p[m][n];

 for (int j=0; j<5;j++)             // repeatedly transform q-points
 {
        for (int k=0; k<3; k++)
        {
          ga.drawPoint(g, q[k]);
          ga.drawArc(g, q[k], q[(k+1)%3], type);
        }

        for (int k=0; k<3; k++)       // transform each q-point
          q[k] = ga.f(ga.est(ga.scale(ga.F(q[k]),type), B));
 }
}
```

4.13 Computational example 5: Translating circles

An alternative approach to translations utilising the fact that the (conformal) translation can be applied to the blade representation of a circle using exactly the same 'sandwich' transformation formulas 4.1

$$C \rightarrow e^{B/2}Ce^{-B/2} = C'. \qquad (4.2)$$

This transforms the conformal blade representation of the circle from C to C′

In the base space, this should cause the (non-Euclidean) centre to move as though its centre had undergone the same point translation and, for non-Euclidean space, the radius of the circle would also possibly change.

The following code repeatedly translates a circle by mapping its conformal representation. The generator for the transformation is that needed to map the point x to y:

```
double[] type = ga.e;

double[] centre  = {0,+0.4,+0.5};
double radius = 0.4;

// construct blade representation of circle

double[] circleBlade =
        ga.makeCircle(ga.scale(ga.F(centre),type), radius, type);

ga.drawCircle(g, circleBlade, type);

// construct generator of translation from x to y

double[] x = {0,  +0.4, +0.5};
double[] y = {0,  +0.3, +0.4};
double[][] data = {ga.F(x), ga.F(y), type };
double[] B = ga.gp(ga.createBlade(data),type);

for (int i=0; i<10; i++)
{
   circleBlade = ga.est(circleBlade, B);  // transform circle blade
   ga.drawCircle(g,circleBlade, type);    // draw circle
}
```

In the Euclidean and spherical cases this produces the output of figure 4.7

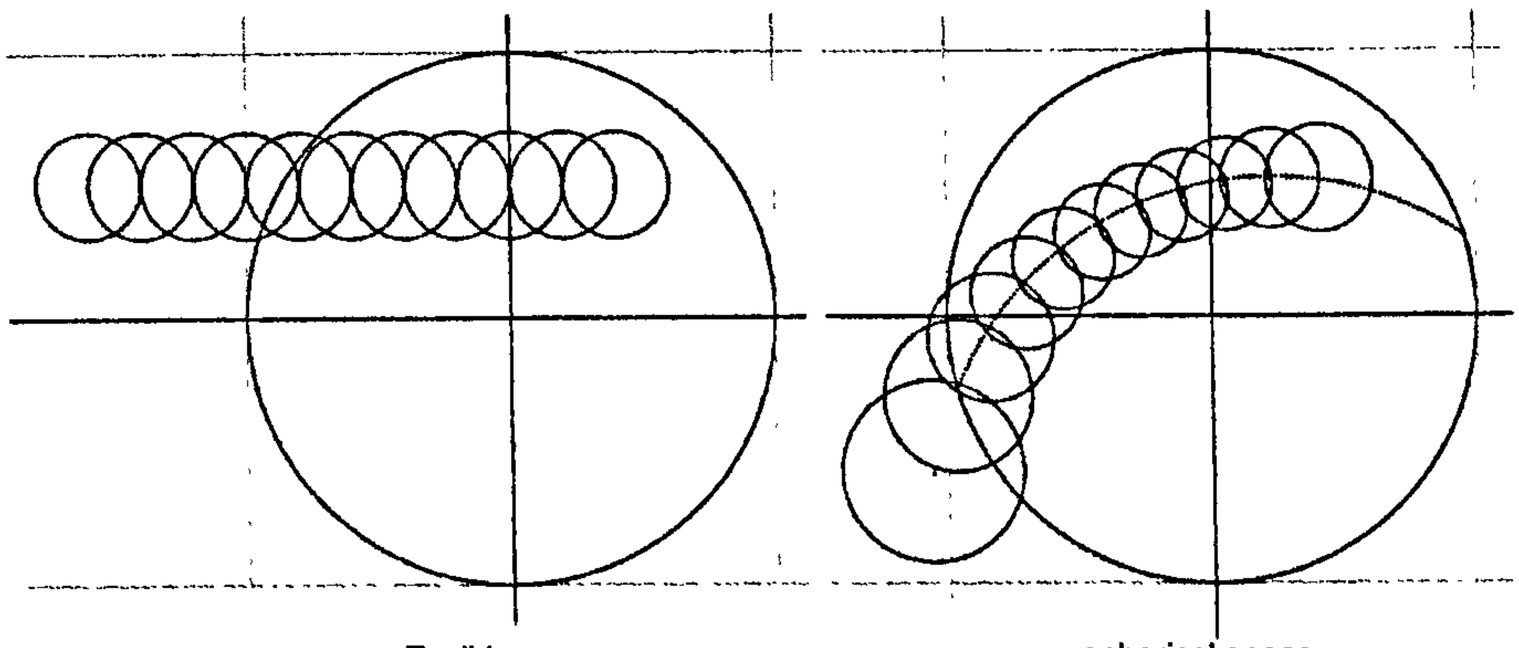


Figure 4.7 Translating a circle in Euclidean and spherical space (hemisphere model).

In both cases the circle's centre tracks a geodesic path. This is also true in the hyperbolic cases though, once again, problems of scale appear.

4.14 Conclusion

Apart from problems of scale, the implementation of Cl(3,1) in this chapter seems to support the theory in relation to the conformal representation of transformations when applied to points and circles. Though the literature suggest ways to represent other conformal transformations (rotations, dilations, inversions, etc) these were not attempted as it was felt that the theoretical basis of the model needed to be explored and understood at much greater depth before proceeding with software design and experimentation.

For the same reason, it was felt that attempts to implement mouse interaction should be postponed, though the ideas tested here could be used to implement rudimentary mouse dragging.

5 Implementing non-Euclidean Navigation Controls

5.1 Introduction

This chapter presents a solution to the problem of creating navigation controls for moving through non-Euclidean spaces. It uses the bivector-generated map

$$X \to e^{B/2} X e^{-B/2} \tag{5.1}$$

in conjunction with retained-mode graphics which was used to build scene-graphs structures in Chapter 2. The approach arose out of extending the idea of quarternions.

The chapter concludes with comments about the nature of the exponential map used in 5.1. To put it in its broader context, it is necessary to briefly mention Lie transformation groups.

5.2 Quarternion based rotation

Quarternions are elements of the even sub-algebra of Cl(3,0). If B is a bivector (pure) quarternion, then the map

$$X \to e^{B/2} X e^{-B/2}$$

induces a rotation in Cl(3,0). The plane of the rotation is the plane of the bivector and the magnitude of the bivector is a measure of the degree of the rotation. Each bivector can be scaled to make the rotation angle small. In particular, the bivectors e_{12}, e_{23}, e_{13} generate rotations represented by these planes. However, because the quarternions relate to 3D space, these 'planar' rotations correspond to axial rotations about the principal axes. By restricting

vectors of R^3 to unit vectors on the sphere S^2, unit-norm quarternions can also be thought of as acting on S^2.

Viewing a small area of S^2 directly down the e_3 axis so that it appears locally 'flat', small rotations induced by e_{12}, e_{23} and e_{13} appear to respectively rotate the sphere's surface anti-clockwise, or translate it horizontally or vertically. Similarly, small rotations induced by $-e_{12}$, $-e_{23}$, $-e_{13}$ rotate or translate the sphere in opposite directions. Thus, locally, the bivectors e_{12}, e_{23}, and e_{13} can be associated with 2D control buttons of the style adopted in chapter 2, see figure 5.1.

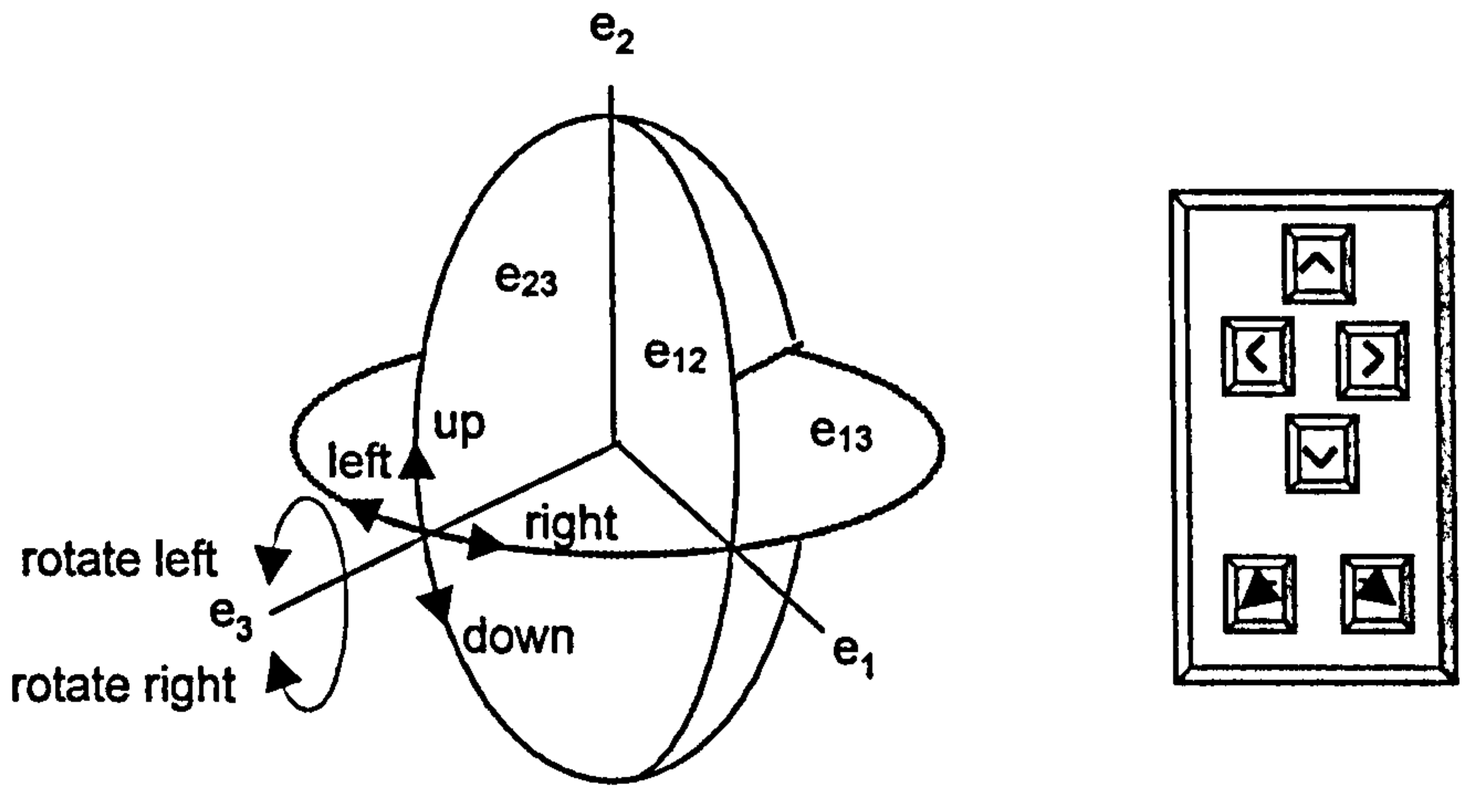


Figure 5.1 **Left**: Rotations of a sphere about the three principal axes viewed as (left/right) horizontal and (up/down) vertical translations together with a (left/right) rotation.

Right: Control button interface to implement the rotations.

These six control buttons could be used to 'nudge' the sphere S^2 into any position, or equivalently, to move a 'turtle' on a static sphere into any position and orientation on the sphere. In fact, classically, the turtle could be driven over the surface of the sphere using just the up/down (forward/backward) and rotate right/left buttons or, equivalently, the two bivectors e_{12} and e_{23}. These essentially 2D controls could provide movement in 3D in much the way a tracker-ball could, except that in this case the tracker ball could only be nudged vertically and horizontally. It would also be possible to rotate this 'virtual' tracker ball about a vertical axis through its centre.

5.3 Implementing quarternion rotations in the conformal model Cl(3,1)

The 6-dimensional bivector subspace of the conformal model Cl(3,1) is spanned by the base bivectors e_{12}, e_{13}, e_{14}, e_{23}, e_{24} and e_{34}. Through the 2-sided map 5.1 their exponentials act on elements of Cl(3,1) in general and on 4D null vectors in particular. In so doing they induce transformations in the embedded space S^2, H^2 and R^2.

Because the computational approach being used here is specific to a pre-chosen orthonormal basis, this embedding takes a rather simple form - it is merely a matter of ignoring, or not working with certain components. (Extracting an embedded space or structure in a co-ordinate free way often entails some sort of algebraic 'split', for example, the classical 'projective' and 'conformal' splits alluded to in the literature. These will not be needed here.)

Because of the embedding of S^2 in the conformal model Cl(3,1), quarternion rotations in S^2 can be simulated in the model by using only three of the 6D bivectors, namely e_{12}, e_{13} and e_{23} and applying the exponential transformations generated by them to the those 4D conformal points representing points or circles lying on the surface S^2 in R^3, or to those 6D blades of Cl(3,1) that also conformally represent circles on S^2.

To illustrate, figure 5.2 shows the result of successive quarternion rotations of a point on S^2 that would be induced by successive presses of the control buttons associated with e_{23}, e_{13}, $-e_{23}$, $-e_{13}$, e_{12}, namely Up, Right, Down, Left, and Rotate (right).

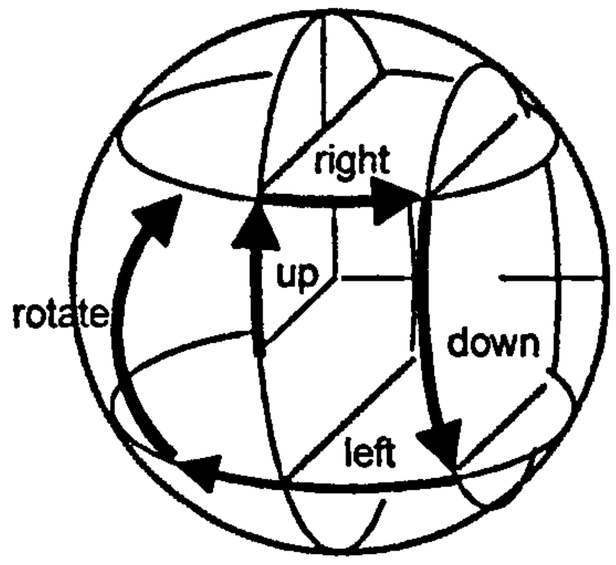


Figure 5.2 Path traced out by a point on a sphere undergoing a sequence of quarternion rotations about fixed axes.

Figure 5.3 is a screen dump of the same quarternion rotation sequences implemented in the conformal model Cl(3,1) and applied to a small circle viewed in the 2D hemisphere model - i.e. the projection onto the equatorial plane via the south pole.

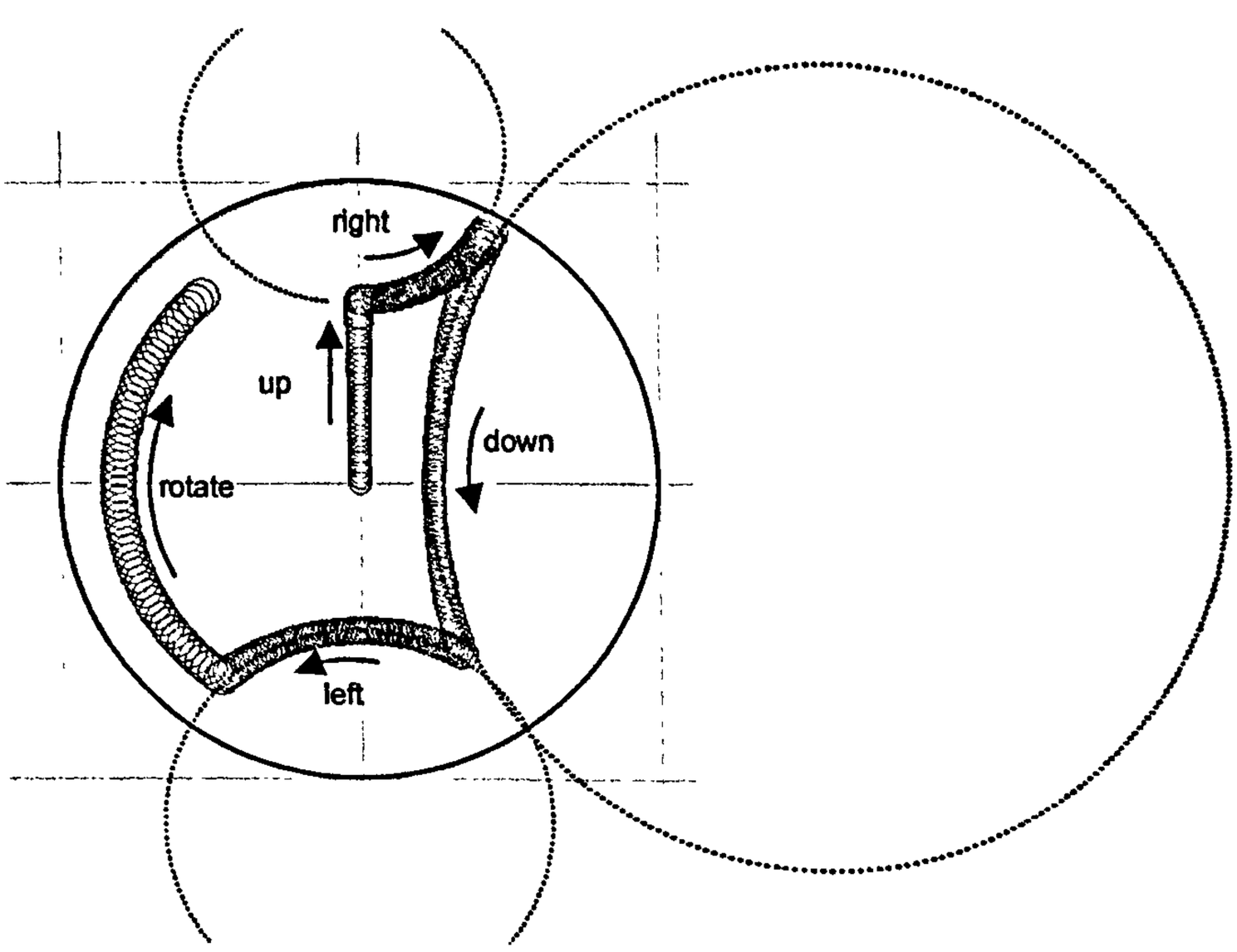


Figure 5.3 Stereographic projection of a path traced out by a circle on a sphere undergoing a sequence of quarternion rotations using immediate mode graphics.

As can be seen from figure 5.3, when horizontal or vertical control buttons of the same type are successively applied in the hemisphere model, the object (the

small circle) does not necessarily move along a geodesic path. In fact, the paths are arcs of projection of the circular paths in S^2, see figure 5.2. As these paths are not necessarily great-circles, their projections need not be geodesics. However, if an object is positioned at the origin, then the motion-control buttons do generate geodesic movement - the initial 'up' movements from the origin were along a geodesic path.

5.4 Using retained mode graphics to implement geodesic navigation

It is only when the object moves away from the origin that non-geodesic movement occurs. This provides a clue to solving the problem of implementing geodesic navigation, namely

to constantly redefine the origin to keep track with the object.

Coincidentally, this is precisely the strategy used in chapter 2 to implement the scene-graph architecture. There each object is effectively defined relative to the origin and its subsequent motion history is stored. Any new transformation is then applied relative to a new origin defined by its motion history.

In terms of classic matrix operations the composite transformation is

$$\mathbf{x} \rightarrow \text{prevTransforms.(currentTransform).}\mathbf{x}$$

i.e.

$$\mathbf{x} \rightarrow \text{(prevTransforms.currentTransform).}\mathbf{x}$$

Looked at in temporal terms, the vector **x** is first subject to the current transform, as though it were still at the origin, and the result is then subject to a transformation to compensate for the fact that it is not at the origin but actually has a previous history of movement.

Stated another way, the motion-history matrix 'prevTransforms' is updated by post-concatenation on the right, instead of the usual pre-concatenation on the left, i.e.

prevTransforms → prevTransforms.currentTransform

New transformations thus no longer transform objects - they update motion histories which are used to transform the object from its original position at render time. This rendering strategy is sometimes called 'retained mode' as opposed to 'immediate mode'.

In terms of the sandwich map of the conformal model, this means changing the transformation composition strategy from immediate mode

$$\mathbf{x} \rightarrow S_2 \, (S_1 \, \mathbf{x} \, S_1^{-1}) \, S_2^{-1}$$

to retained mode

$$\begin{aligned} \mathbf{x} \rightarrow & \; S_1 \, (S_2 \, \mathbf{x} \, S_2^{-1}) \, S_1^{-1} \\ = & \; (S_1 S_2) \, \mathbf{x} \, (S_1 S_2)^{-1} \\ = & \; S \, \mathbf{x} \, S^{-1}, \end{aligned}$$

where the evolving motion history multivector s is updated by post-concatenation on the right.

In terms of bivector-generated transformations this could be interpreted as

$$\begin{aligned} \mathbf{x} \rightarrow & \; \exp(B_1/2) \, \exp(B_2/2) \, \mathbf{x} \, \exp(-B_2/2) \, \exp(-B_1/2) \\ = & \; tL_1 \, \exp(B_2/2) \, \mathbf{x} \, \exp(-B_2/2) \, tR_1 \end{aligned}$$

The 'left' and 'right' history multivectors tL_i and tR_i are updated on the right and left by $\exp(B_{i+1}/2)$ and $\exp(-B_{i+1}/2)$ respectively. The updated history multivectors tL_{i+1} and tR_{i+1} are used to transform **x** from its original home position. This strategy obviates the need for inverting multivectors. (An alternative approach that uses the fact that the history multivectors belong to

the $Spin_+$ group, where the inverse corresponds to the reverse, is discussed in section 5.10.)

Figure 5.4 shows a small circle being moved through the hemisphere model of S^2 using this new retained-mode strategy. The navigation actions are similar to those of the previous example.

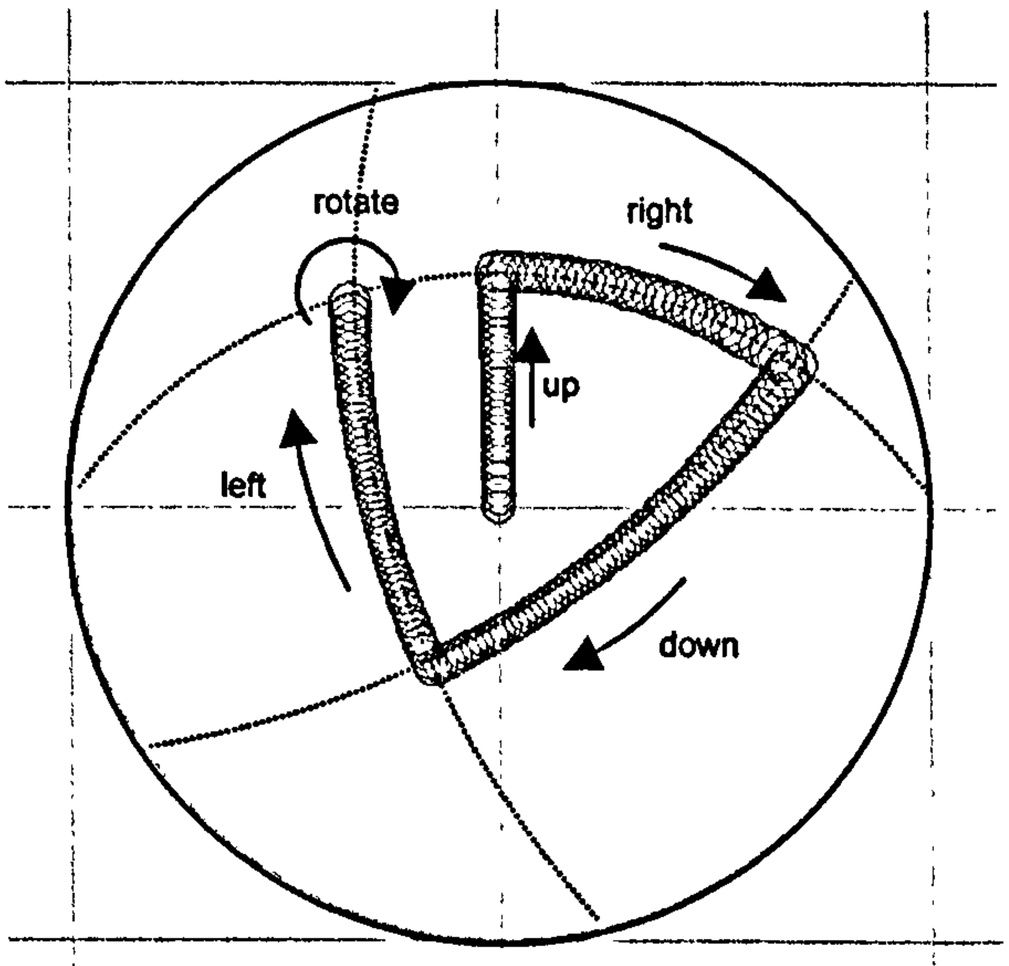


Figure 5.4 Stereographic projection of a path traced out by a circle on a sphere undergoing a sequence of quartemion rotations using retained mode graphics.

As can be seen, the translations are now geodesic, and the rotation is relative to the circle's centre.

The screen-dump for figures 5.3 was implemented using the transform() function

```
void transform()    // immediate mode
{
   tL = ga.exp(ga.div(B, +2));
   tR = ga.exp(ga.div(B, -2));
   S  = ga.gp(tL, S, tR);
}
```

This function used the bivector B to transform a multivector S representing the circle.

To implement retained mode graphics, and hence generate the geodesic movement shown in figure 5.4, this function was changed to

```
void transform()    // retained mode
{
    tL = ga.gp(tL, ga.exp(ga.div(B, +2)));
    tR = ga.gp(ga.exp(ga.div(B, -2)), tR);
    S  = ga.gp(tL, T, tR);
}
```

In this case, the bivector B was used to update the left and right motion-history multivectors tL and tR. These, in turn, were used to transform the retained multivector T representing the circle at its start position centred at the origin.

5.5 Extension to H^2

Geodesic navigation through the hemisphere model of S^2 was implemented using quarternion rotations in the conformal model. This meant selecting from the 6 bivectors of Cl(3,1) the 'quarternion' bivectors e_{12}, e_{13} and e_{23}. As stated, these are essentially the 3 bivectors of the Clifford Algebra of R^3, the space in which S^2 is embedded.

H^2 is embedded in $R^{2,1}$ and, because of the latter is embedded in $R^{3,1}$, the appropriate bivectors for geodesic navigation in the Poincaré disc model are e_{12} and the Minkowski bivectors e_{14}, e_{24}. Using retained-mode rendering, these generate the geodesic motion in the Poincaré disc model depicted in figure 5.5.

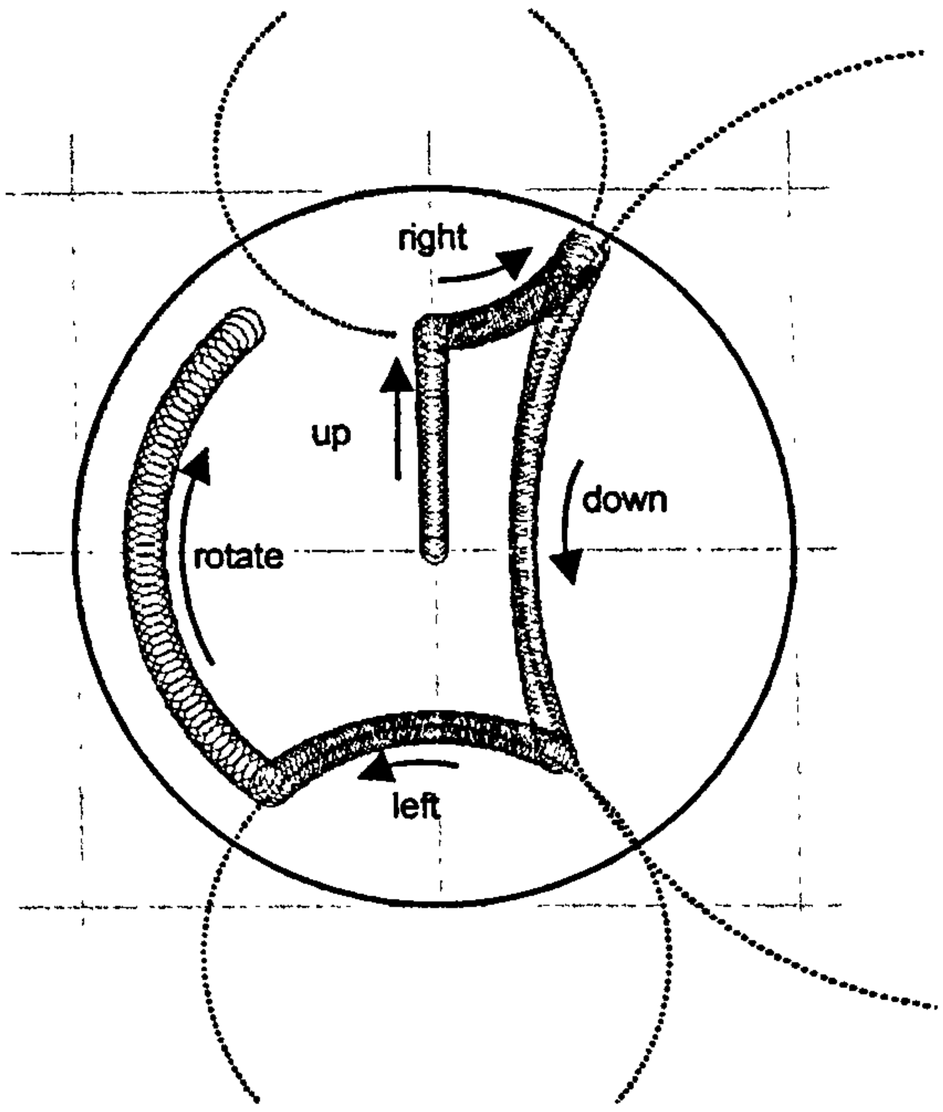


Figure 5.5 Geodesic path traced out by a hyperbolic circle undergoing rotations generated by the bivectors e_{12} , e_{14} and e_{24} using retained mode graphics.

5.6 Unification

Because the above technique can be extended to R^2, a single algorithm can be constructed to handle navigation in S^2, H^2 and R^2. In each case the bivector that generates locally planar rotations (as viewed down the e_3 axis) is e_{12}. However, locally planar horizontal translations are generated by e_{13}, e_{14} and $e_1\mathbf{n}$ in each case respectively. Vertical translations are generated by e_{23}, e_{24} and $e_2\mathbf{n}$. Table 5.1 summarises this:

geometry	defining vector **t** :		S^2 e_4	H^2 e_3	R^2 $\mathbf{n} = e_3 + e_4$
rotations	central	Ro	e_{12}	e_{12}	e_{12}
translations	horizontal	Tx	e_{13}	e_{14}	$e_1\mathbf{n}$
	vertical	Ty	e_{23}	e_{24}	$e_2\mathbf{n}$

Table 5.1 Navigation bivectors for subspaces associated with Cl(3,1)

In each geometry, the horizontal and vertical translation bivectors Tx and Ty can be calculated from the pre-set geometry-defining-vector **t** using the formula

$$Tx = e_1 e_{34} \mathbf{t}$$

and $$Ty = e_2 e_{34} \mathbf{t}.$$

Thus it is possible, for example, to write a single routine that will allow a polar grid, which is easily generated at the origin, to be translated and rotated about its centre at will, in any of the three geometries, see figure 5.6.

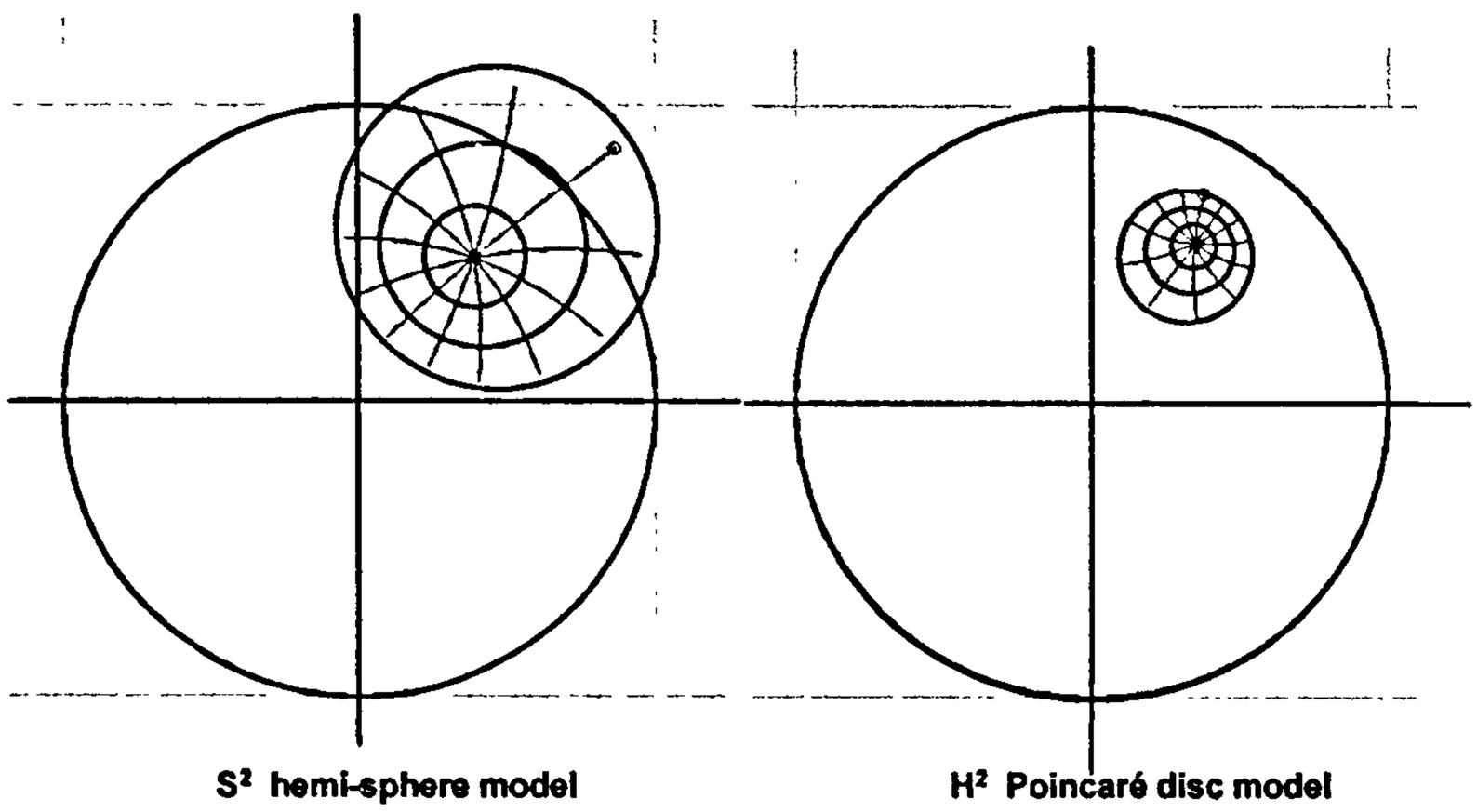


Figure 5.6 A polar grid in spherical and hyperbolic geometry.

5.7 Extension to higher dimensions

This geodesic navigation scheme extends quite easily to higher dimensions. For example, table 5.2 shows the extensions to the spaces S^3, H^3 and R^3 embedded in Cl(4,1).

		S^3	H^3	R^3
geometry	defining vector **t** :	e_5	e_4	$\mathbf{n} = e_4 + e_5$
rotations	roll	e_{12}	e_{12}	e_{12}
	pitch	e_{13}	e_{13}	e_{13}
	yaw	e_{23}	e_{23}	e_{23}
translations	Tx	e_{14}	e_{15}	$e_1\mathbf{n}$
	Ty	e_{24}	e_{25}	$e_2\mathbf{n}$
	Tz	e_{34}	e_{35}	$e_3\mathbf{n}$

Table 5.2 Navigation bivectors for subspaces associated with Cl(4,1)

If the six bivectors that produce navigation in S^3, namely e_{12}, e_{13}, e_{23}, e_{14}, e_{24} and e_{34} are treated as though they were elements of Cl(4), rather than Cl(4,1), then they effectively rotate the 4-dimensional sphere S^3. In that context there is nothing to distinguish them, but by stereographically projecting S^3 (via the vector e_5) into the 3-dimensional hemisphere model, the six bivectors split into two equal groups - those that produce rotations and those that produce translations.

In this navigation scheme, the translation bivectors for the respective models of S^3, H^3 and R^3 are again remarkably similar and given in terms of the geometry-defining vector **t** by

$$Tx = e_1\, e_{45}\, \mathbf{t}$$

$$Ty = e_2\, e_{45}\, \mathbf{t}$$

$$Ty = e_3\, e_{45}\, \mathbf{t}$$

Thus it was once again possible to write successful 'universal' code that handled navigation in all three geometries. Figures 5.7 and 5.8 show forward (geodesic) flight in 3D spherical and hyperbolic space of a '3D turtle' that is initially translated vertically, then pitched and rolled before flight begins (these

initial movements are not shown). To facilitate faster rendering, the turtle's edges are drawn using screen-lines that are straight, not geodesic arcs. In a certain sense, the turtle is drawn using its locally flat geometry - the 'curved' geometry only manifests itself when it flies.

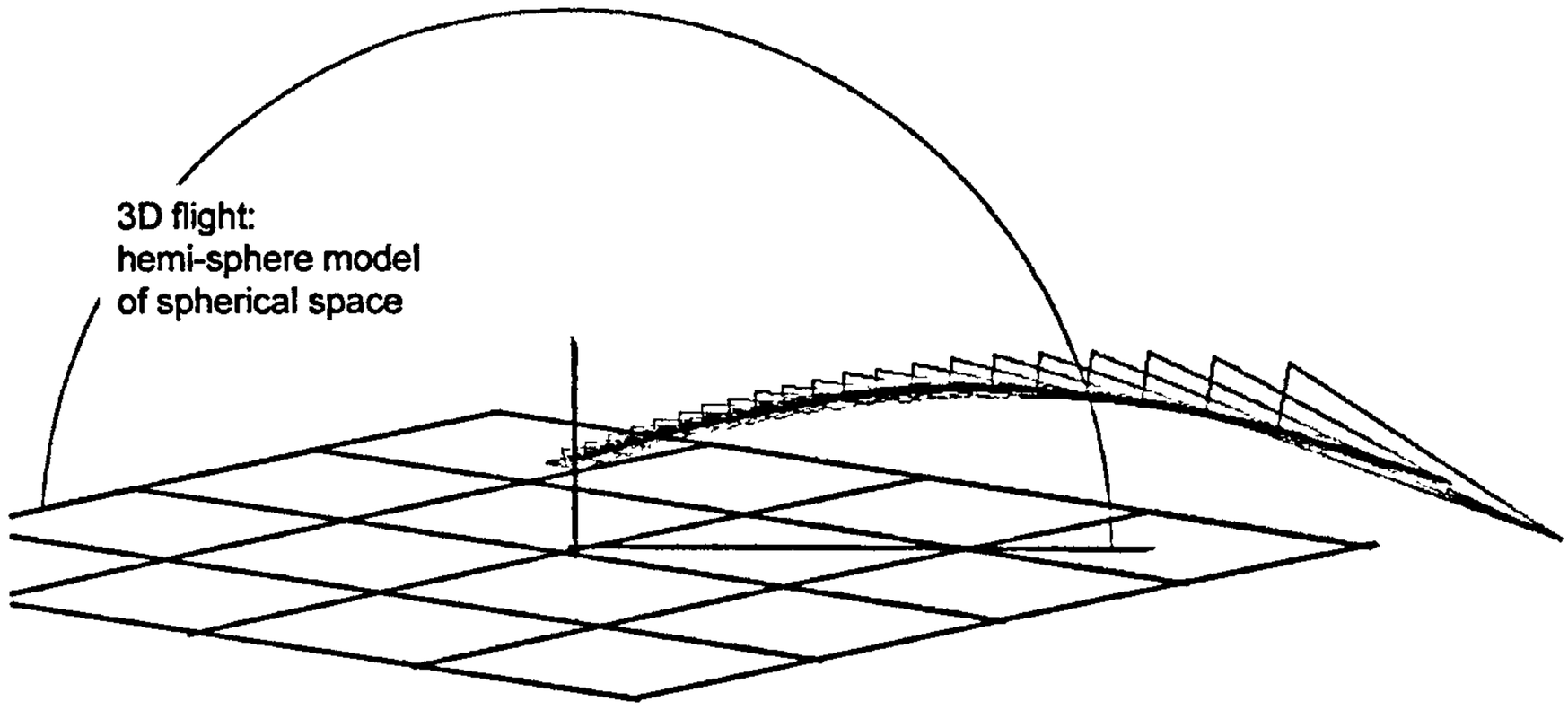


Figure 5.7 Geodesic flight path of a 3D turtle in spherical space. (The curve is due to the curvature of the space.)

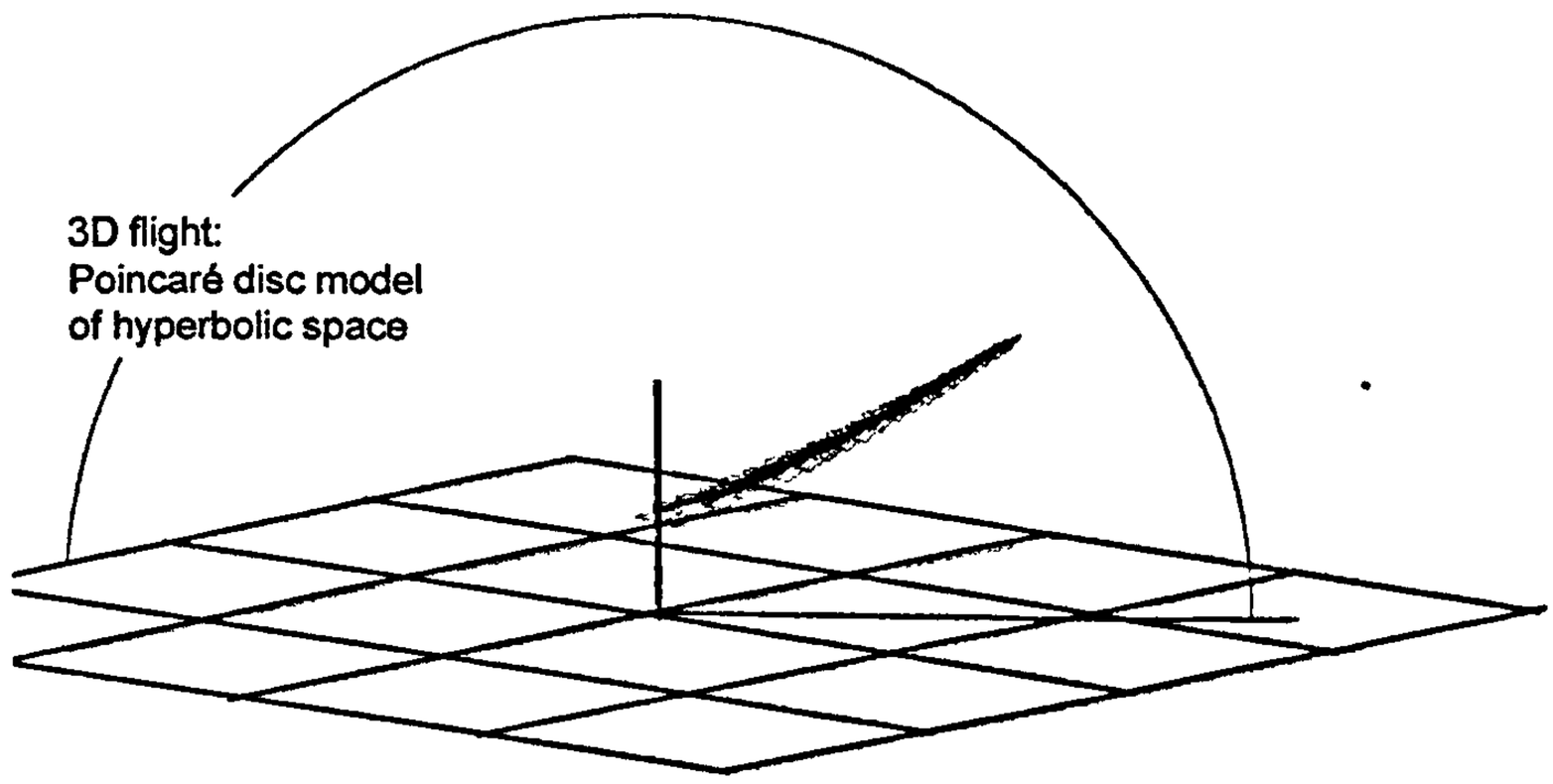


Figure 5.8 Geodesic flight path of a 3D turtle in hyperbolic space. (The curve is due to the curvature of the space.)

5.8 The Lie group $\mathbf{Spin_+(p,q)}$

The mixed signature non-degenerate Clifford algebra Cl(p,q) contains a transformation group $\mathbf{Spin_+(p,q)}$ which, in many senses, seems to generalise the idea of quarternions. An element s of $\mathbf{Spin_+(p,q)}$ acts on a vector $X \in$ Cl(p,q) through the two-sided transformation

$$X \rightarrow sXs^{-1} \tag{5.2}$$

Elements s and -s give rise to the same transformation.

There are a number of ways of formally defining $\mathbf{Spin_+(p,q)}$. One approach is to consider the group generated by the products of invertible vectors, the so-called Lipschitz or Clifford group, then to apply the further restriction that the elements should have even grade and 'norm' $ss^\dagger = 1$, in which case $s^{-1} = s^\dagger$ so that the two-sided map 5.2 can also be defined as

$$X \rightarrow sXs^\dagger \tag{5.3}$$

where $s^\dagger$ is the reverse of the multivector s. (A multivector is reversed by changing the sign of its elements of grade 2,3,6,7,10,11, and so on.)

A more geometrically intuitive approach, used by Hestenes, Li and Rockwood in [18, p 16] is to again start with elements of the Clifford group, the so-called 'versors', but restrict the products to an even numbers of vectors only. In this treatment the elements of $\mathbf{Spin_+(p,q)}$ are sometimes called 'rotors' reflecting the fact that two sided transformations by single vectors represent reflections, and that generalised rotations are the products of two (or an even number) of such reflections.

The transformation group $\mathbf{Spin_+(p,q)}$ is a Lie group because it is possible to continuously vary transformations to gradually change their effect. However, the behaviour of such transformations, particularly those that cause small

'nudging' changes and that are therefore 'close to the identity', is usually described through the Lie algebra associated with the Lie group.

5.9 The Lie algebra of the spin group $\mathbf{Spin_+(p,q)}$

In relation to the Lie group $\mathbf{Spin_+(p,q)}$, Lounesto [19, p221] makes two key general comments which relate the exponentials of bivectors of Cl(p,q) to the Lie group

1. The Lie algebra of $\mathbf{Spin_+(p,q)}$ is the space of bivectors $\Lambda^2 R^{p,q}$;
2. The exponentials of bivectors generate the group $\mathbf{Spin_+(p,q)}$.

This means that if B is a bivector, i.e. an element of the Lie algebra associated with a Lie group, then the map

$$k \to e^{kB/2} X e^{-kB/2}$$

represents a 1-parameter 'continuous' group of transformations in the following sense: A given bivector B determines a specific transformation and scalar multiples of it magnify or diminish its effect without changing its essential nature. If the scalar multiple becomes zero, the transformation 'reduces to nothing', i.e. it becomes the identity transformation.

More formally, the zero bivector generates the identity multivector, i.e.

$$e^0 = 1.$$

Also, within the group of transformations generated by B, the inverse transformation is achieved by changing the sign of the parameter, i.e.

$$e^{-kB} = \left(e^{kB}\right)^{-1}.$$

The similarity between these transformation group identities and classical exponential formulae for real numbers can be misleading, since other classical results need not apply. For example, the composite map generated by bivectors B_1 and B_2 is not always that generated by $B_1 + B_2$. However, if B_1 and B_2 commute then it is true that

$$e^{B_1} e^{B_2} = e^{B_1+B_2}.$$

If they do not commute, the equivalent general result is known as the Campbell-Baker-Hausdorff formula [20, p58] which can be expressed as

$$e^{X}\, e^{Y} = e^{f(X,Y)}$$

where f can be expressed in terms of Lie brackets [X,Y] = XY - YX as

$$f(X,Y) = X + Y + 1/2[X, Y] + 1/12([X, [X, Y]] + [Y, [Y, X]]) + ...$$

It is not generally true that all elements of a Lie group can be generated by the exponentials of its Lie algebra. However, as comment 2 above indicates, this is a special property of Cl(p,q). Thus for any element $s \in \mathbf{Spin}_+(p,q)$ there exists bivectors $B_1, B_1 \ldots B_k$, such that

$$s = \exp(B_1)\exp(B_2)\ldots \exp(B_k)$$

Riesz [21, page 172], cited by Lounesto [19, page 223], further shows that the bivector B generating a given transformation is generally not unique in the sense that it is often possible to find another bivector F such that $e^{B} = -e^{F}$ with the consequence that both B and F induce the same sandwich transformation, i.e. $e^{B/2} X e^{-B/2} = e^{F/2} X e^{-F/2}$. However, Riesz points to exceptions in the following cases

$R^{1,1}$ for all B,

$R^{2,1}$ and $R^{1,2}$ for all $B \neq 0$ such that $B^2 >= 0$

and $R^{3,1}$ and $R^{1,3}$ for all $B \neq 0$ such that $B^2 = 0$.

This means, for example, that if $s \in \mathbf{Spin}_+(3,1)$ and there is a null bivector B such that exp(B) = s, then it is not possible to find another bivector that induces the same mapping as s in the manner described above. In other words, if s is the exponential of a null bivector B then B is unique.

5.10 The meaning of the exponential

In Lie theory, the exponential is a map for moving from the Lie algebra to the Lie group. Its definition is dependent on context and, in some cases, can be highly abstract. At the other extreme, there are often contexts when a definition based on a Taylor expansion will suffice.

The cited remarks of the previous section assume a fairly abstract definition. Fortunately, for the purposes of this research, the definition given in chapter 4 based on a Taylor expansion will suffice:

$$\begin{aligned} \exp(B) = \exp(k\hat{B}) &= \cosh(k) + \hat{B}\sinh(k) && \text{if } B^2 > 0, \\ &= \cos(k) + \hat{B}\sin(k), && \text{if } B^2 < 0, \\ &= 1 + k\hat{B} && \text{if } B^2 = 0. \end{aligned}$$

This expansion assumes that a bivector squares to a real value. In spaces of 3 dimensions this is always the case. In higher dimensions, such as in our conformal model Cl(3,1), the square of a bivector is real if and only if the bivector is 'simple', i.e. a pure blade that can be expressed as the outer product of vectors. This condition is fortunately satisfied for the transformation bivectors used in the remaining chapters, making the above Taylor expansions computationally valid.

The fact that the exponential maps bivectors of Cl(p,q) to the spin group $\mathbf{Spin}_+(p,q)$ has computational consequences. The bivector generated mapping

$$X \to e^{B/2} X e^{-B/2}$$

can be written as

$$X \to sXs^{-1}, \qquad \text{where } s = e^{B/2}.$$

However, because $s \in \mathbf{Spin}_+(p,q)$, the mapping can also be written as

$$X \to sXs^{\dagger}$$

where $s^{\dagger}$ is the reverse of s. This means that instead of having to compute the exponential twice in order to obtain $e^{B/2}$ and $e^{-B/2}$, it is only necessary to compute the first exponential and then reverse the result to effectively obtain the second. This means swapping the signs of the elements of grade 2 and 3.

5.11 Conclusion

This chapter used the familiar notion of quarternion rotations to try to 'make sense' of bivector induced transformations in Cl(3,1). In this context, there is a remarkable similarity between S^2 and H^2 which allowed simple extension from one to the other. It was as though H^2 had an algebra associated with it analogous to that of the pure quarternions. The successful extension to R^2 followed.

In fact, unit-norm quarternions correspond to the group $\mathbf{Spin}_+(3,0) = \mathbf{Spin}_+(3)$ whose Lie algebra is the 3D space of bivectors of Cl(3,0), i.e. the space of pure quarternions. The analogous pure 'hyperbolic' quarternions are the Lie Algebra of $\mathbf{Spin}_+(2,1)$, i.e. the 3D space of bivectors of Cl(2,1).

Thinking in terms of 4-dimensional 'quarternion rotations' then led naturally to 6 navigation bivectors for S^3. These were drawn from the 6-dimensional Lie algebra of $\mathbf{Spin}_+(4)$. The stereographic projection of S^3 to R^3 meant that the induced transformations in the hemisphere model split into 3 rotations and 3 translations. It is not clear, at this stage, whether this split is related to the group isomorphism

$$\mathbf{Spin}(4) \sim \mathbf{Spin}(3) \times \mathbf{Spin}(3)$$

mentioned in [19, p 88 and 21, p 89] which suggests that one way to rotate 4D space could be to use two 3D tracker balls.

In fact, there is an extensive theoretical literature on relevant group-related structures that can encompass the theory of minimal ideals, group representations, spinors and so on. This is not surprising since many of these ideas have relevance in theoretical physics, but only rarely do they impinge on the conformal model where the focus of interest is non-Euclidean geometry.

Rather than attempt to go down these theoretical paths, this thesis will take an entirely different direction in its search to understand bivector-generated transformations. The remaining chapters focus on the nature of families of circles (i.e. pencils) generated by repeatedly applying these transformations to a single circle. As will be shown, the results obtained have immediate computational applicability.

6 Bivector based transformations - a picture emerges

6.1 Introduction

This chapter describes an analysis of bivector generated transformations arising out of a need to implement centralised mouse dragging operations.

The analysis not only points towards an implementation, but also suggests a novel approach to defining non-Euclidean viewport transformations using just three control points. These, in turn, raise fundamental questions about the nature of geometry and its interactive representation on the computer screen. The chapter concludes by identifying some of these issues.

6.2 Mouse-induced rotations and dilations

A key aim of this research is to develop centralised mouse dragging operations analogous to the classical Euclidean rotation and dilation about a point, see diagrams in figure 6.1 where, in both cases, the point P is dragged to Q.

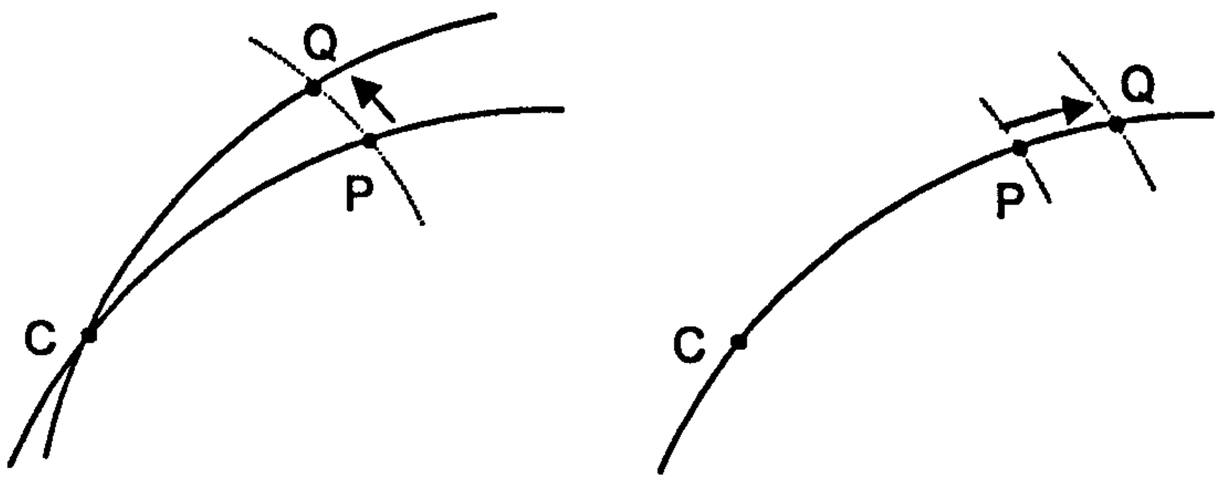


Figure 6.1 Mouse dragging operations about a fixed point .

In the figure the radial geodesics through the centre C are represented by arcs of circles. Both mappings transform P to Q while leaving the centre C invariant. In the case of the rotation, the geodesic circle defined by the trivector blade p^c^t should map to the circle defined by q^c^t, where t is the

'point at infinity'. (Here, the lower case is used to denote the conformal representation of points in upper case.)

In the case of the dilation, the circle through P with centre C should map to the 'concentric' circle with the same centre but passing through Q. In both cases the transformations induced by the mouse are circle-to-circle mappings. The assumption is that such mappings can be effected by standard bivector-based 'sandwich' maps of the form

$$s \rightarrow e^{B/2} s e^{-B/2},$$

where s is the original circle in each case, and B is an appropriate 'simple' bivector, i.e. a pure blade.

6.3 Composing rotations and dilations

If the bivector blades generating these two mappings commute, then the composite mapping can be efficiently generated using their bivector sum. This follows from the Cambell-Baker-Hausdorff result mentioned in the previous chapter. The converse assumption then is that any centralised mouse-drag, where P is dragged to *any* position Q, while the centre C remains fixed, can be decomposed into a rotation and a dilation, and that these commute. At this stage these are speculative comments based purely on intuition. However, such an elegant solution, if it exists, is worth pursuing.

The key to implementing this intended approach is to find the bivector blade to generate a transformation that maps a given circle onto another. There is a brief hint in [23, page 10] that the solution may be to form the wedge product of the two circles, though no explanation is given. Even though this approach was found to work in special circumstances, the lack of any theoretical justification meant looking elsewhere for a suitable perspective.

6.4 Pencils of circles generated by two circles

A clue to understanding bivector generated transformations seems to lie in the description of pencils of circles given in [2, pages 50 to 52]. This account discusses pencils of higher dimensional 'r-spheres' in the context of the conformal model Cl(n+1,1) of R^n.

The comments below reinterpret these observations in the context of the Cl(3,1) model of R^2 where the dual of a bivector happens to be also a bivector. This simplifies a duality which is mentioned in [2] but which is somewhat obscure in the context of higher dimensional spaces.

The pencils of circles described in [2] are generated from the conformal vector representations s_1 and s_2 of two circles by taking linear combinations of the form $s = as_1 + bs_2$, with the proviso that s has a non-negative signature. The nature of the pencil generated depends on the whether the circles intersect, touch or do not intersect. The pencils generated are called respectively intersecting, tangent and Poncelet pencils, see figure 6.2.

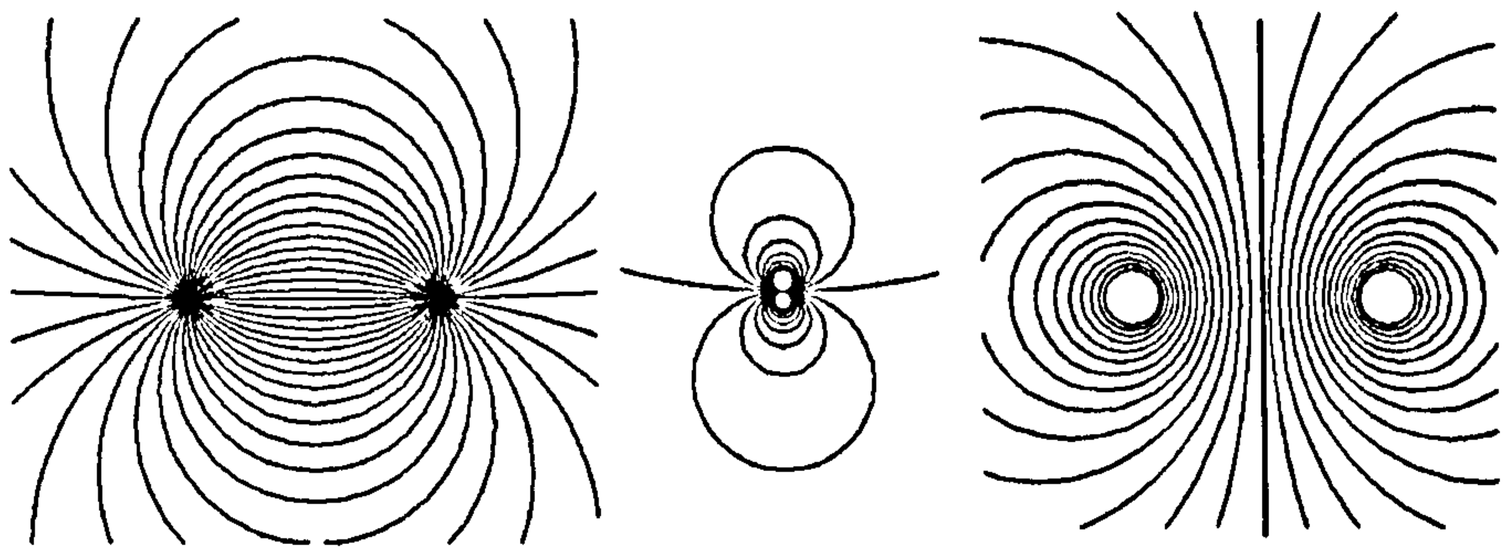


Figure 6.2 The three types of pencil.

Two circles are needed to generate a pencil, and any two circles of the pencil can be used as substitute generators for the pencil.

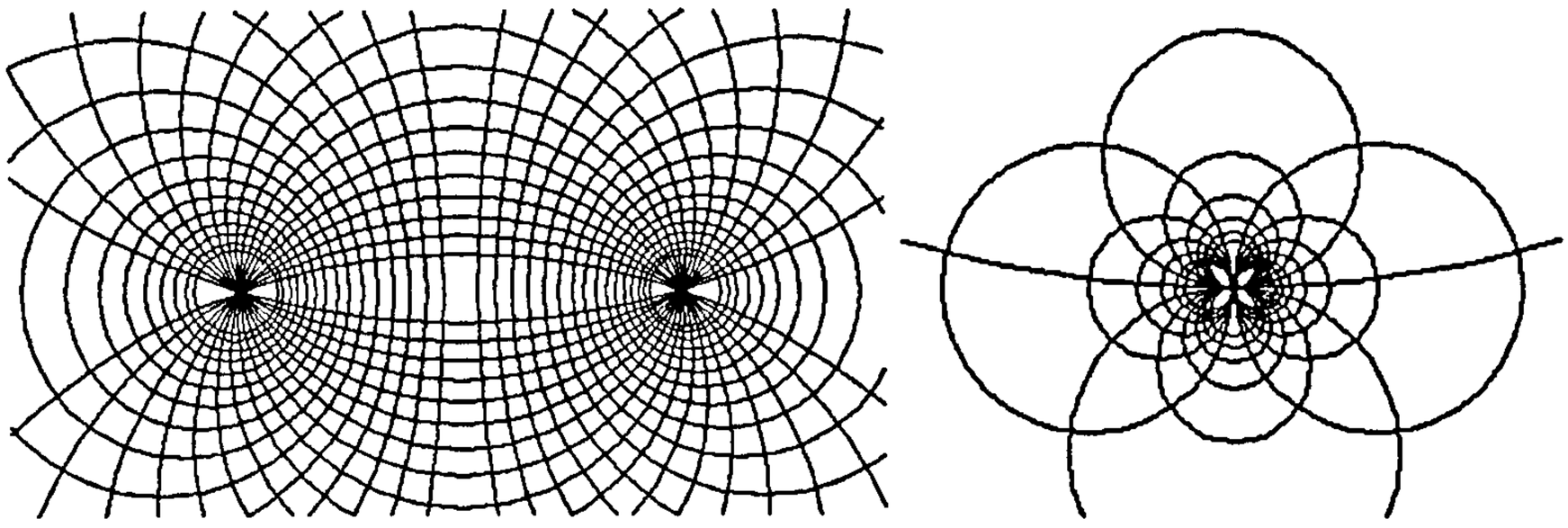
6.5 Pencil representation in conformal space

In conformal space, the set of vectors $s = as_1 + bs_2$ parametrically define a plane which can be equally represented by the bivector blade $B = s_1 \wedge s_2$.

The intersection relationship between the circles represented by s_1 and s_2 is exactly mirrored by the signature of B which, in turn, determines whether the plane it defines intersects, touches or does not intersect the null cone in conformal space.

If $B^2<0$, the plane B intersects the null cone and does so exactly twice, defining two null vectors that fix two points on the screen viewport, i.e. in R^2. These are the common points of concurrency of members of the intersecting pencil. If $B^2=0$, the plane touches the null cone along a single null vector. The corresponding point on the screen viewport is the common point of tangency of the tangent pencil. If $B^2>0$ the blade B does not meet the null cone. In this case the blade defines a Poncelet pencil.

It turns out that in R^2 each type of circle-pencil has a 'dual' circle-pencil: a set of circles that are orthogonal to all the circles in the original pencil. The dual of an intersecting pencil is a Poncelet pencil, and vice versa. The two points of concurrency of the original intersecting pencil are the centres of the circles in the dual Poncelet pencil. The dual of a tangent pencil is an orthogonal tangent pencil with the same point of tangency, see figure 6.3.



intersecting pencil and dual Poncelet pencil

tangent pencil and dual tangent pencil

Figure 6.3 Pencil and dual pencil superimposed.

Remarkably, if the bivector B represents a pencil, then its dual $B^{\sim} = BI^{-1}$, represents its dual pencil.

6.6 Generating pencils using the bivector as a transformation generator

The following paragraphs extend the description in [2] by relating the results presented there to bivector generated transformations.

The conformal vectors representing the circles s_1 and s_2 lie on the plane represented by $B = s_1 \wedge s_2$. Any member s of the pencil generated by s_1 and s_2 lies on this plane, and the mapping

$$s \rightarrow e^{kB/2} s e^{-kB/2}$$

can be viewed as being a rotation in conformal space of the vector s in the plane $s_1 \wedge s_2$. Repeating the transformation on s with different values of k generates the pencil of circles. Equally, successive applications of the mapping applied to s, with a given value of k, generate a *subset* of the pencil.

Thus it is possible to view the bivector B, when applied successively to any circle s on B, as being a generator of a pencil of circles. The dual bivector $B^{\sim}$ generates the orthogonal pencil. In other words, a bivector B generates an orthogonal co-ordinate system - a co-ordinate 'patch' that spans the whole of

R^2. However, to do so, an initial circle s needs to be specified. This 'seed' circle can be any member of the pencil. A second orthogonal circle will be needed to seed the dual pencil.

In fact a dual intersecting/Poncelet co-ordinate patch can be generated using just three control points, two to specify the points of concurrency of the intersecting pencil, and a third to select out a seed circle from the pencil. This same point can be used to select a seed circle from the orthogonal pencil, see figure 6.4.

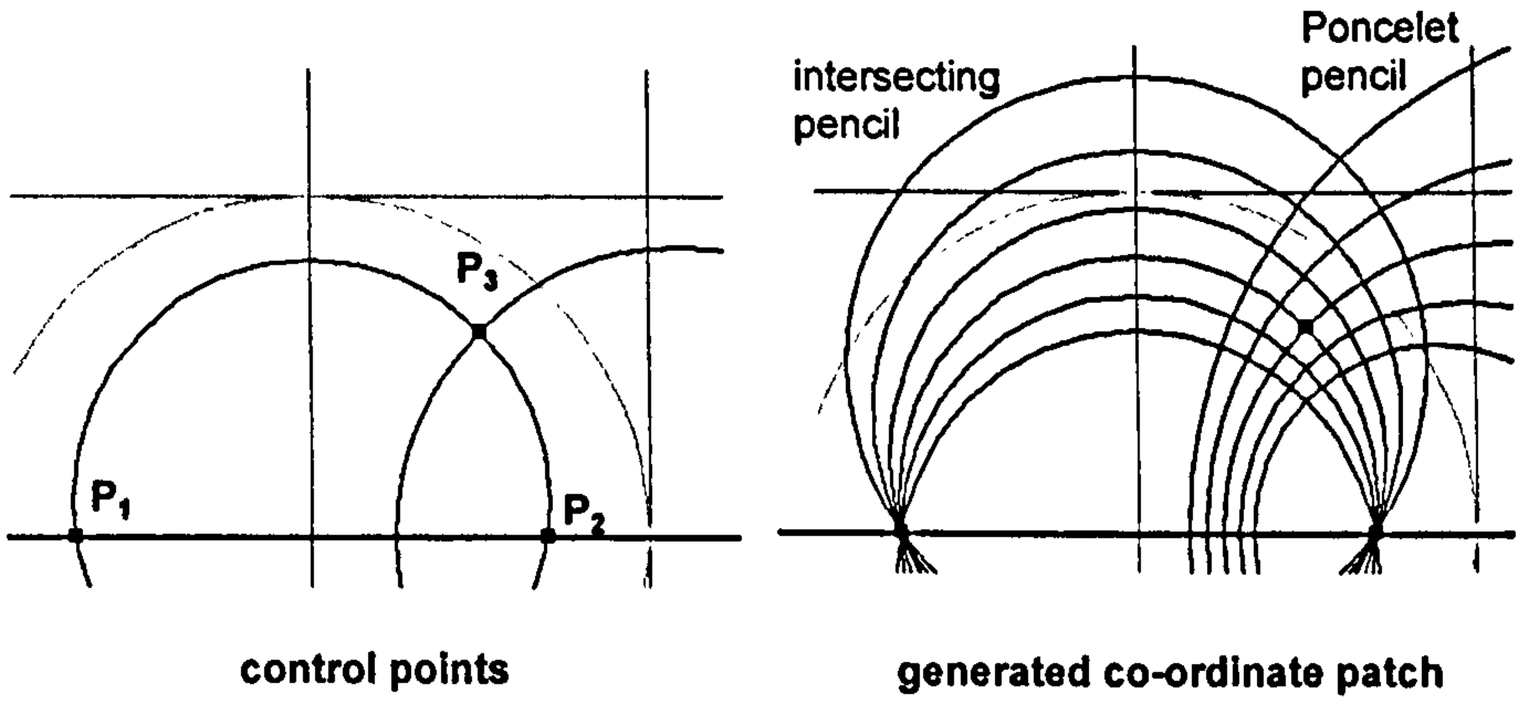


Figure 6.4 Co-ordinate patch around a control point P_3 generated by control points P_1 and P_2 which define an intersecting pencil.

With reference to figure 6.4, two defining members of the intersecting pencil are the circle through P_1, P_2 and P_3, and the 'generalised' circle (i.e. straight line) through P_1, P_2 and the Euclidean 'point at infinity' **n**. These are given by

$$s_1 = (p_1 \wedge p_2 \wedge p_3)^{\sim} \quad \text{and} \quad s_2 = (p_1 \wedge p_2 \wedge \mathbf{n})^{\sim}.$$

The generating bivector is therefore $B = s_1 \wedge s_2$. Either s_1 or s_2 could act as seed circle.

The generating bivector for the dual Poncelet pencil is $B^{\sim} = BI^{-1}$. A seed circle for this pencil can be obtained by using the dual bivector $B^{\sim}$ to map the point P_3 repeatedly to obtain the points P_4 and P_5. The seed circle is then $s_3 = (p_3 \wedge p_4 \wedge p_5)^{\sim}$, see figure 6.5.

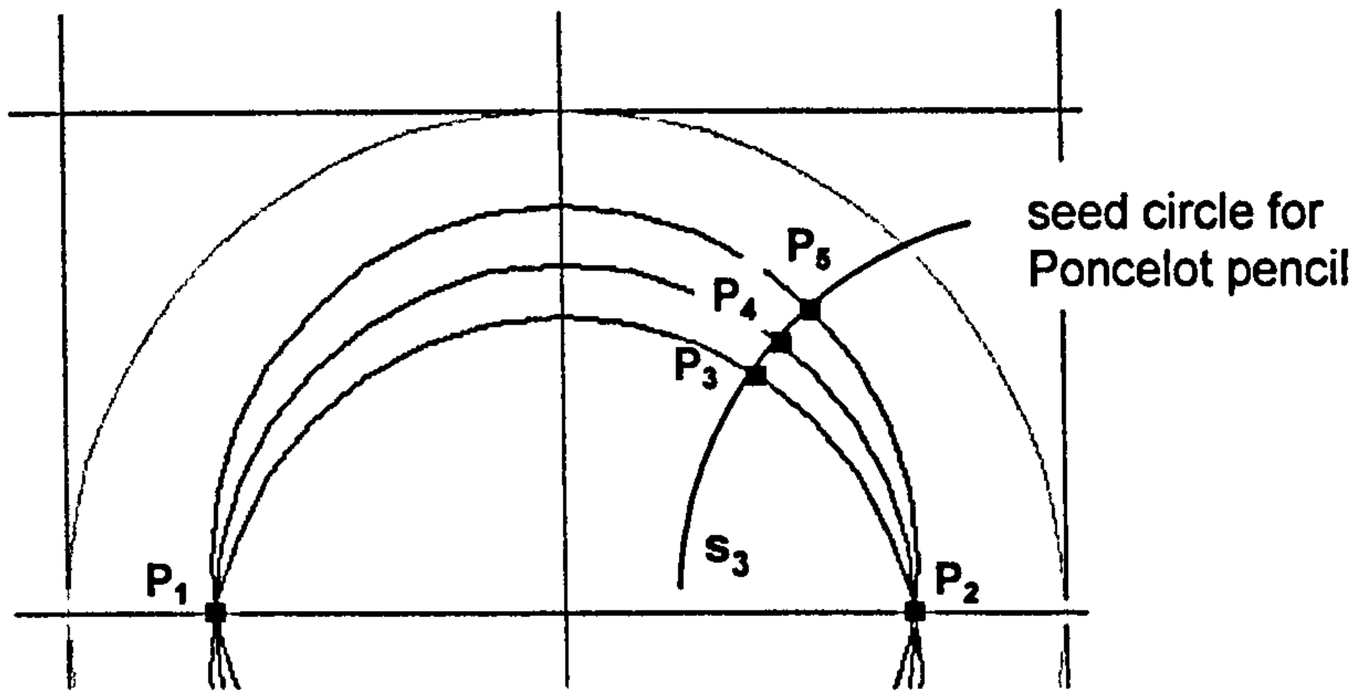


Figure 6.5 Seed circle through the control point P_3 used to generate the Poncelet pencil dual to the intersecting pencil defined by control points P_1 and P_2.

6.7 Bivectors and classical transformations

The signature of a bivector B determines the nature of the pencil it generates. Looked at as a transformation, if $B^2<0$ the rotation is classical or 'Euclidean' in the sense that the exponential function generates trigonometric functions. If $B^2>0$ the rotation is 'hyperbolic' in the sense that hyperbolic functions are generated. Euclidean rotations are 'circular' whereas hyperbolic rotations are asymptotic, see figure 6.6.

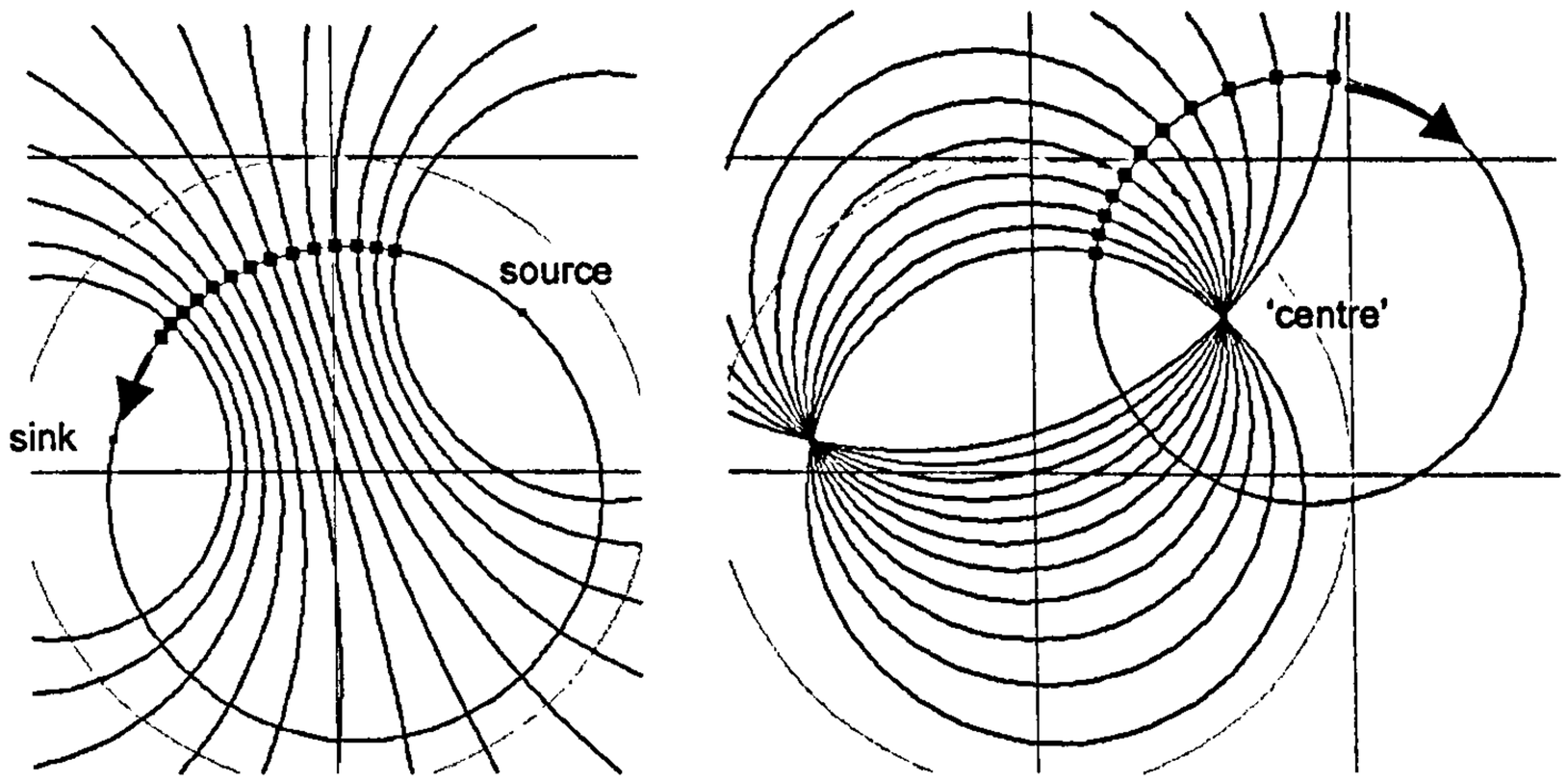


Figure 6.6 Hyperbolic and Euclidean rotations.

In the figure, the two pencils shown are dual, and the transformations are in fact generated by a bivector B and its dual bivector B˜. Thus, for any point on the plane, a bivector and its dual define two localised mutually orthogonal transformation directions that are directly related to the underlying co-ordinate patch realised as two dual pencils.

This has interactive implications. It means that once the three control points have been positioned (in order to define a co-ordinate system), the cursor keys can be naturally linked with directions associated with the co-ordinates.

Because the underlying algebra is taking place in conformal space, with its two extra dimensions, it is theoretically possible to allow the control points to become 'points at infinity', or to tend toward each other; this is discussed in [2] in relation to pencils.

However, for the end-user, unaware of this underlying algebra, the notions of 'very far away' or 'very close together' can be dealt with at a pragmatic level by including simple zoom controls acting on the viewport as a whole. To move points far apart, the user would simply zoom-out for a distant view and vice versa.

The rather abstract notions of Euclidean and hyperbolic 'rotations' alluded to above can sometimes be realised in very straight forward ways - it is a matter of choosing the three control points appropriately, and sometimes also an imposed geometry. For example, figure 6.7 shows how polar co-ordinates can be generated by moving the control point P_1 'far away' to the left.

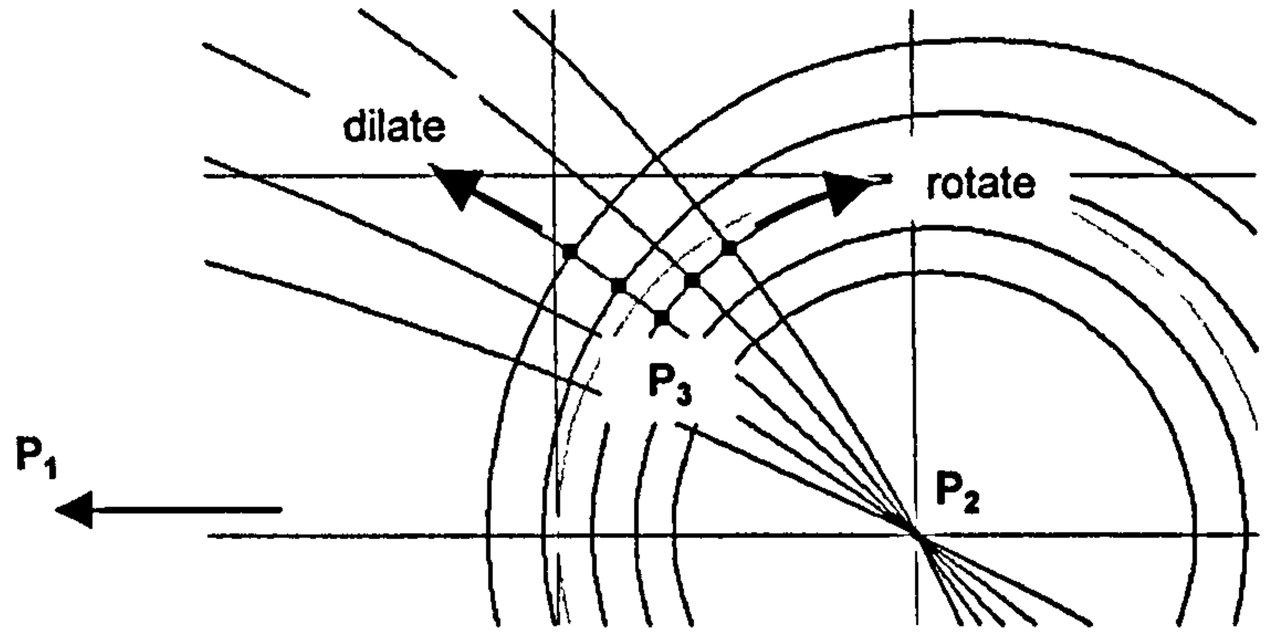


Figure 6.7 Generating polar co-ordinates.

The associated bivector generated transformations are (nearly) Euclidean rotations and dilations. The former is Euclidean (in the rotational sense) and the second hyperbolic, reflecting the signature of the generating bivector and its dual. Thus successive contractions cause the circles to asymptotically approach the centre of the Poncelet pencil. Though essentially polar, the co-ordinates are not linear. One pair of opposing cursor keys could induce rotations in either direction, the other pair dilations and contractions.

Cartesian co-ordinates can be generated by moving both control points p_1 and p_2 'far away' - for example, one to the left and one to the right, see figure 6.8.

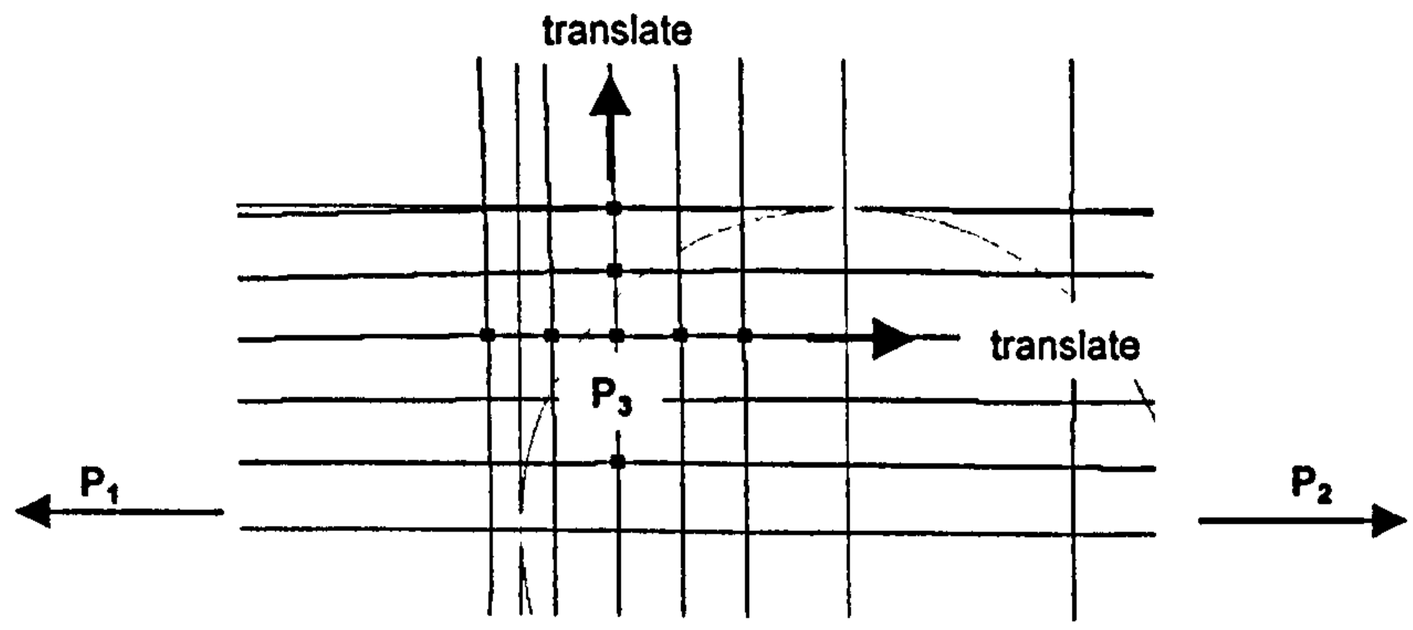


Figure 6.8 Generating cartesian co-ordinates.

In the extreme case, when P_1 and P_2 both become infinite, the associated bivector, and its dual, become degenerate with $B^2 = (B^{\sim})^2 = 0$. In this case the associated transformations are not rotations (Euclidean or hyperbolic) but translations, and the co-ordinate grid comprises two orthogonal dual tangent pencils. In this case the cursor keys would adopt their normal role.

6.8 Imposing a geometry

Because our models of S^2 and H^2 are embedded in R^2, bivector generated co-ordinate systems and their associated transformations can be given interpretations relative to these geometries. For example, figure 6.9 shows the co-ordinate system and transformations induced when the control points p_1 and p_2 are chosen at (0,1) and (0,-1) respectively.

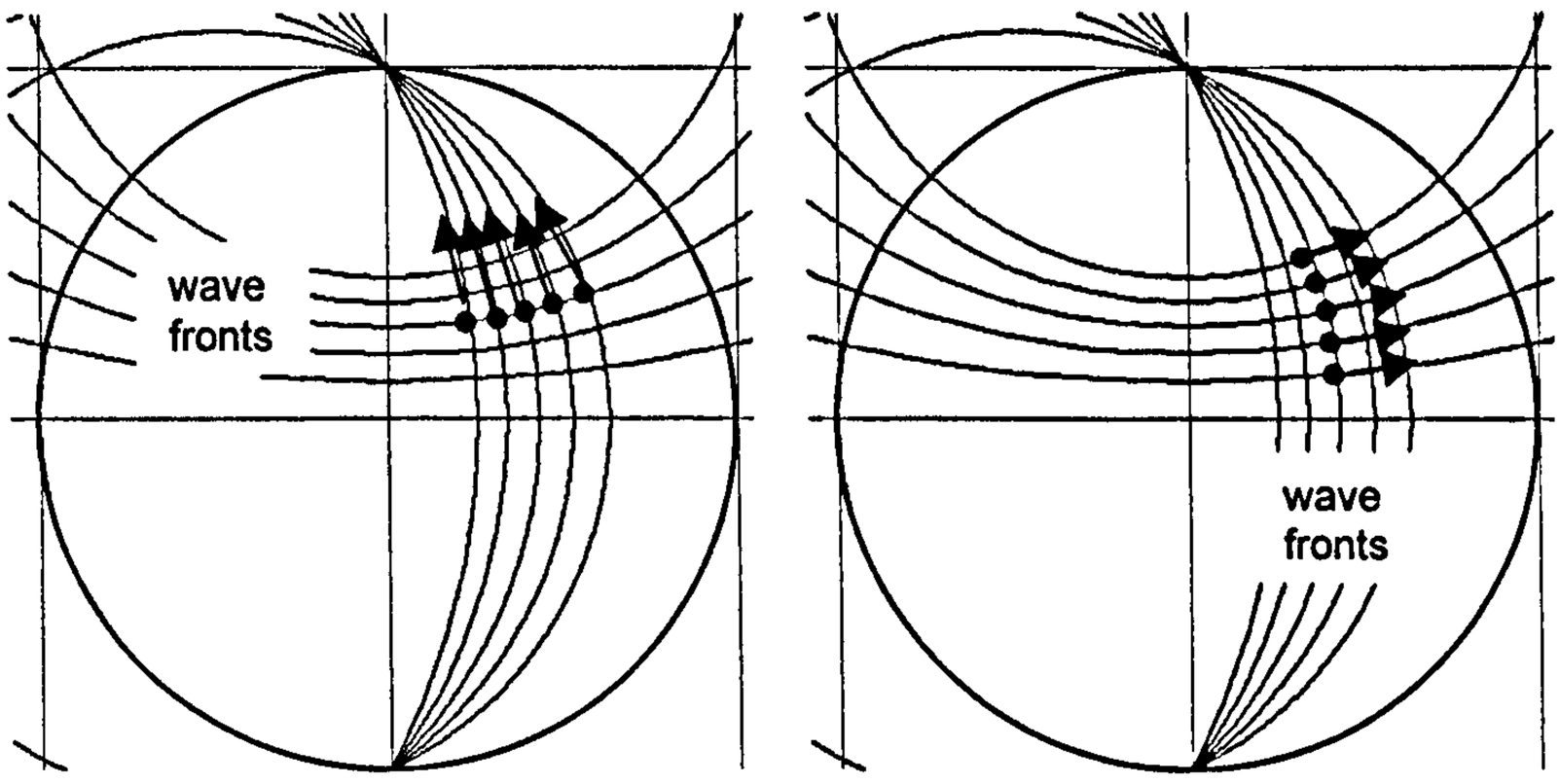


Figure 6.9 Wave fronts of hyperbolic and Euclidean rotations.

If the figures represent the hemisphere model of S^2, then the first transformation depicted is a tidal transformation with the north-pole as the sink. Individual points follow geodesic longitudinal paths. The second transformation depicts a rotation - geodesic longitudes form the 'wave fronts'. Applied to the viewport as a whole, this latter transformation would 'rotate the globe' revealing its dark 'hidden' side. The former would have space appearing to emerge from the source at the south pole and move northward, disappearing

down the sink at the north pole. In fact the transformations are hyperbolically asymptotic, so no space is created or lost.

If, on the other hand, the figures represent the Poincaré disc model of H^2, then the first transformation has geodesic wave fronts, but individual points on the wave front do not follow geodesic paths. It is also a 'tidal transformation', but here the sink and source are both on the infinite circular horizon. If applied to the viewport as a whole, this same tidal transformation would take on a different geometric meaning. The dual transformation, depicted in the second figure, has individual points of the wave front following geodesic paths. However, these are not non-Euclidean 'translations' because the rotation is actually Euclidean - the bivector generator has negative signature so that each point will eventually rotate back to its start position, instead of 'disappearing into the infinite circular horizon'. These transformations would have little geometric meaning in the context of the Poincaré disc model.

6.9 The decomposition of a central transformation - a first attempt

An aim of this chapter is to implement centralised mouse dragging. We require that the transformation induced by a centralised mouse drag from point P to point Q keeps a central point C fixed and maps the geodesic circle s_1 through P and C to the geodesic circle s_2 through Q and C.

The blade $B = s_1 \wedge s_2$ generates an intersecting pencil since s_1 and s_2 meet in two points, namely C and the 'point at infinity' **t**. Hence B generates a rotation about C. To create the specific rotation to move P *exactly* to Q, as shown in figure 6.10, the bivectors B and $B^{\sim}$ need to be suitably scaled.

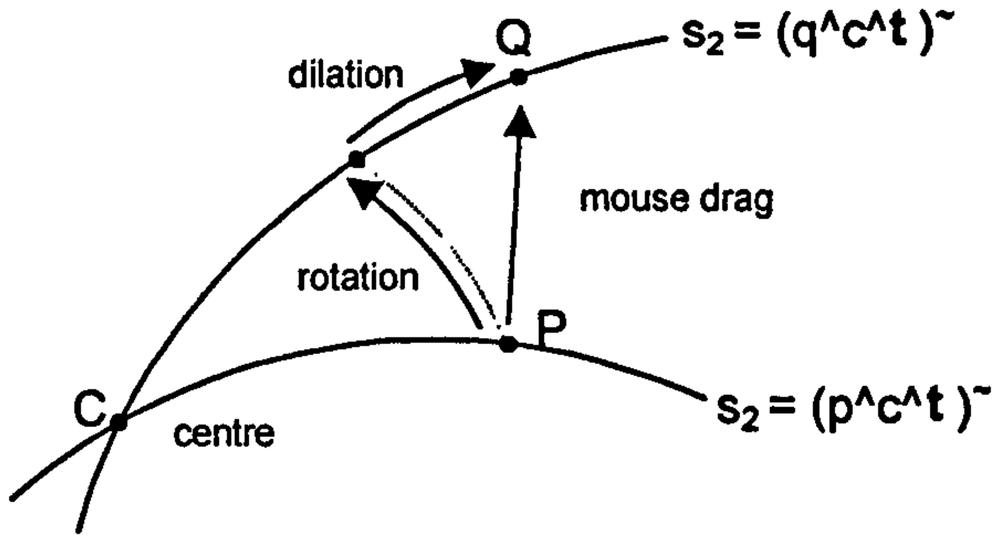


Figure 6.10 The decomposition of a centralised mouse-drag into a rotation and dilation.

The angle between the circles represented by s_1 and s_2 is given by

$$\theta = \cos^{-1}(\hat{s}_1 \cdot \hat{s}_2),$$

where $\hat{s}_i$ denotes normalisation to unit modulus. The appropriate rotation bivector is then

$$\theta\, \hat{B}/2 .$$

These conclusions draw on well known facts about normal rotations in a plane defined by a bivector.

By twice applying this rotation to P and Q, sets of three points on each of the two Poncelet circles r_1 and r_2 through P and Q can be obtained, see figure 6.11.

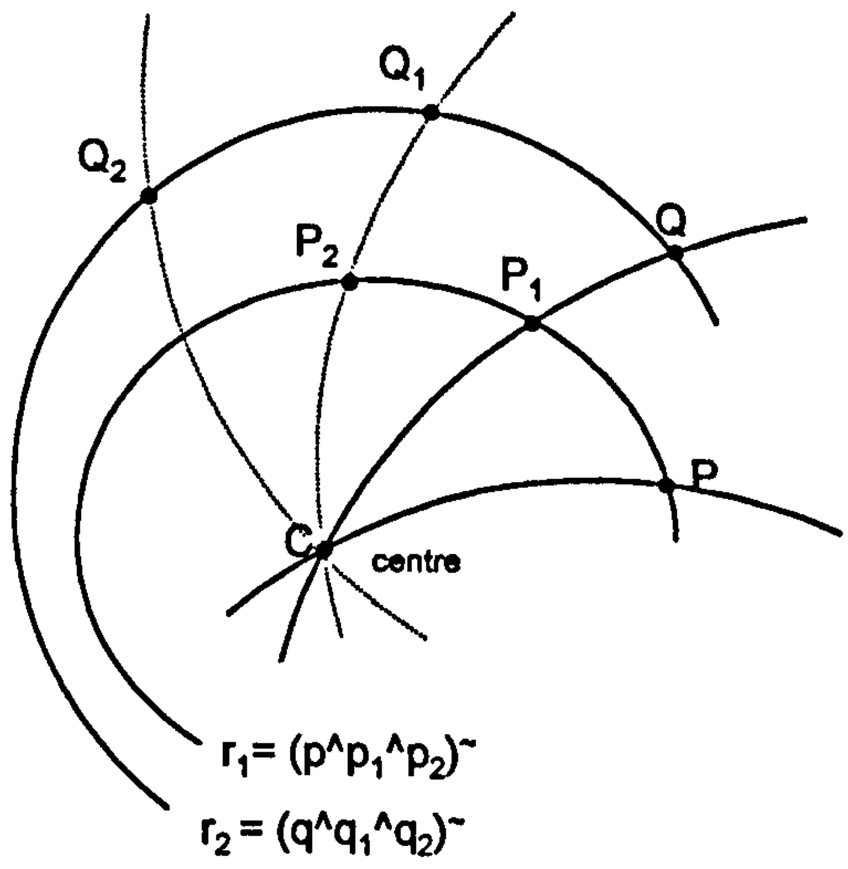
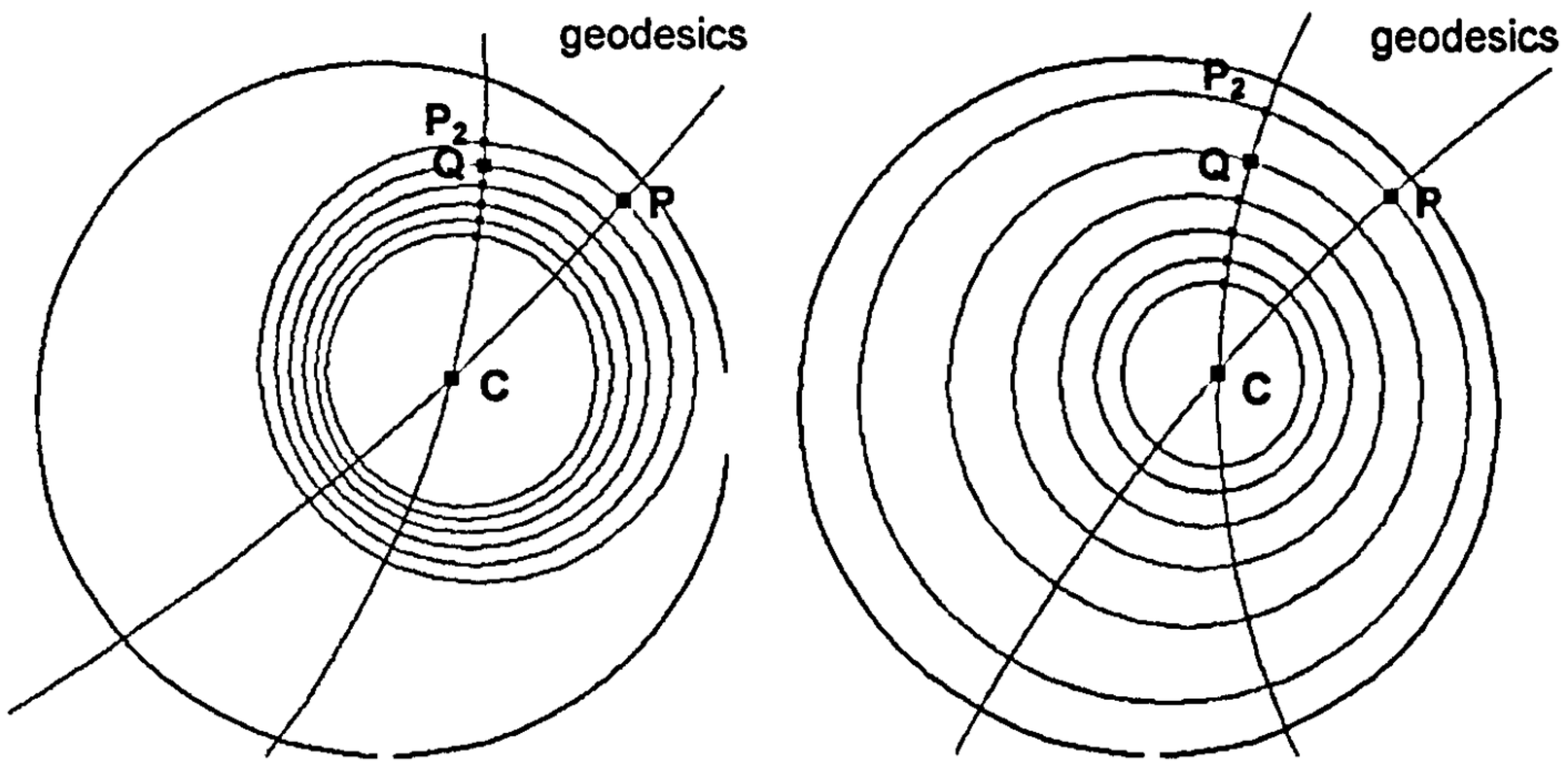


Figure 6.11 Constructing circles through P and Q concentric about a common centre C.

It is then a matter of applying similar scaling to the bivector $B' = r_1 \wedge r_2$. In fact, the dilation bivector B' is a scalar multiple of $B^{\sim}$ and so commutes with B. Hence applying the Cambell-Baker-Hausdorff formula, the bivector that generates the centralised drag from P to Q is B + B'.

However, the exact scaling factor needed to obtain B' proved difficult to determine, so an 'ad hoc' scale factor was used to see whether the expected results were geometrically generic, i.e. that the same algorithm generated rotations and dilations in all relevant geometries. The issue of the exact value of the scaling factor is addressed in the chapter 8.

The suggested approach was tried in four different models of non-Euclidean space. Figure 6.12 shows a point P being mapped to Q via a rotation which takes it to P_2, followed by a contraction taking it to Q. The contraction was then repeated several times to show the corresponding Poncelet circles. The geodesic circles through P and C, and through Q and C were also shown. The figures relate to the hemisphere model of S^2 and the Poincaré disc model of H^2.

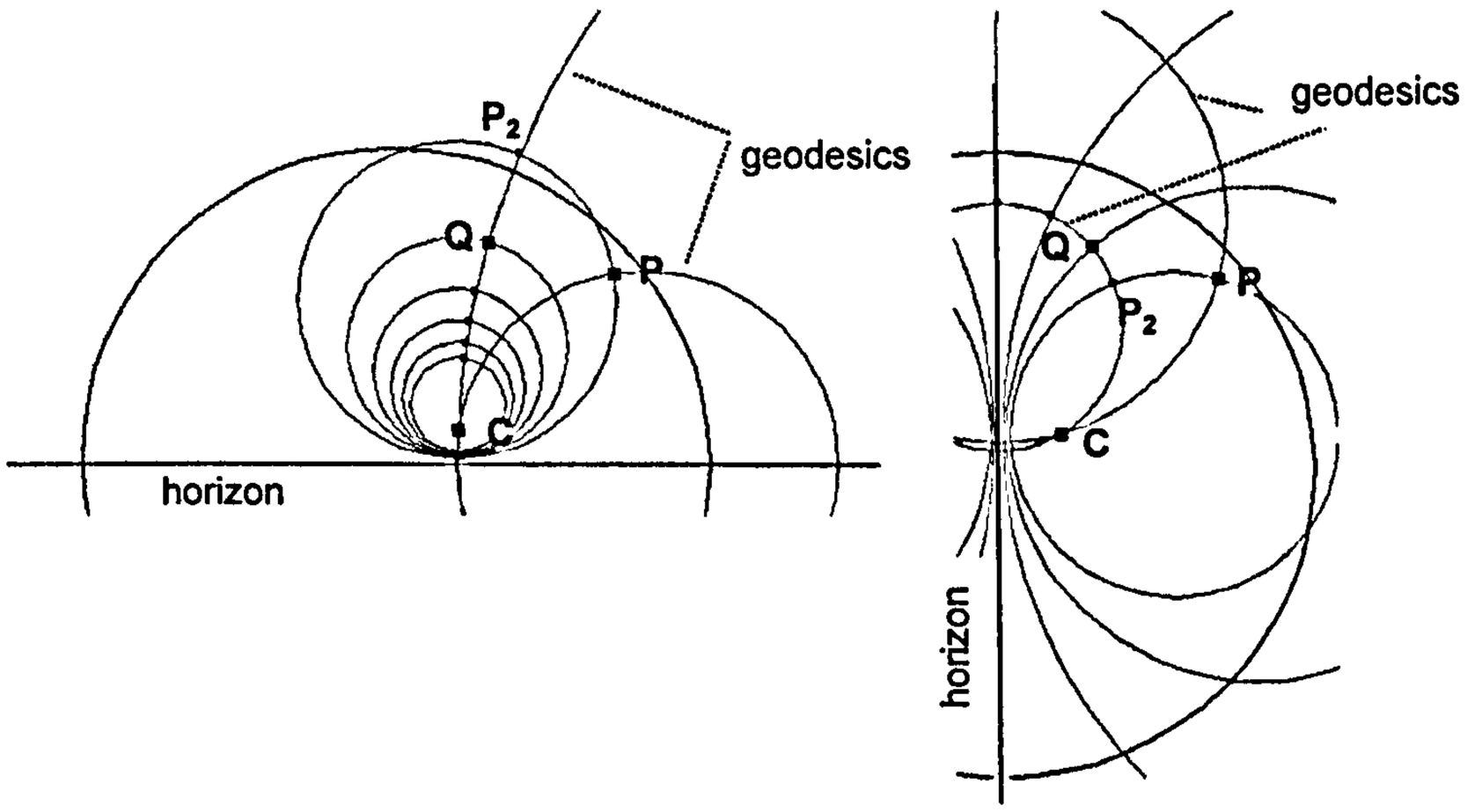


Hemisphere model of S^2
geometry defining vector: e_4

Poincaré disc model of H^2
geometry defining vector: e_3

Figure 6.12 A centralised mouse-drag from P to Q decomposed into a rotation from P to P_2 and a contraction from P_2 to Q.

Figure 6.13 shows a similar decomposition in the horizontal and vertical half-space models of H^2.



Half-space model of H^2
geometry defining vector: e_2

Half-space model of H^2
geometry defining vector: e_1

Figure 6.13 A centralised mouse-drag from P to Q decomposed into a rotation from P to P_2 and a contraction (respectively dilation) from P_2 to Q.

In all cases, the composite bivector B + B' mapped P to Q directly.

6.10 Conclusion

This chapter exploited an apparent connection between the theory of pencils and bivector induced rotations. This connection suggested a way of defining viewport transformations through just three control points which could give a 'feel' for the nature of conformal space. It also led to an approach for implementing non-Euclidean versions of some of the centralised mouse drag operations of Chapter 2.

The emerging perspective seems to be increasingly bivector-centric. This is perhaps not surprising in view of the correspondence between the bivector

algebra of Cl(4,1) and the Lie algebra of $\mathbf{Spin}_+(4,1)$. Also, unusually, the Lie algebra generates the full Lie group $\mathbf{Spin}_+(4,1)$.

Using the Conformal Model Cl(4,1) naturally gives bivectors a central role in the computational algebra. Bivectors give rise to co-ordinate patches which have associated with them fundamental conformal transformations that map circles to circles. The various models of non-Euclidean space then super-impose another level of geometry.

This approach seems to effectively place conformal geometry at the foundation of the various planar models of non-Euclidean geometry. This is certainly algebraically true, but it seems that a suitable interactive environment could potentially do the same thing didactically.

It is perhaps worth observing that in their recent text book, Brannan, Epslen and Gray [24] introduce inversive geometry before discussing hyperbolic and spherical geometry (the latter two presentations use the complex number plane and the Riemann surface). Inversive geometry was seen to underpin hyperbolic and spherical geometry.

7 Pencils of circles - exploiting the idea

7.1 Introduction

This chapter uses pencils of circles to derive needed computational results.

The chapter also explores how conclusions obtained in a specific geometry can be generalised - the key seems to lie in interpreting the nature of the geometry-defining-vector appropriately. In the case of spherical geometry in particular, this can require consideration of the null cone of the encompassing 4D Minkowski space.

7.2 An alternative approach to determine a circle of a Poncelet pencil

To implement centralised mouse dragging, the previous chapter suggested a method to create two generating circles of the dilation-defining Poncelet pencil dual to a rotation-defining intersecting pencil. It entailed successively rotating two specified points twice to create two sets of three points from which two Poncelet circles could be generated (see fig 6.11 of the previous chapter).

The analysis presented here arose out of an attempt to find a more computationally efficient approach to creating Poncelet circles.

Geometrically, any two points P_1 and P_2 define an intersecting and a Poncelet pencil. A third point P can be used to select a specific circle from each pencil, see figure 7.1.

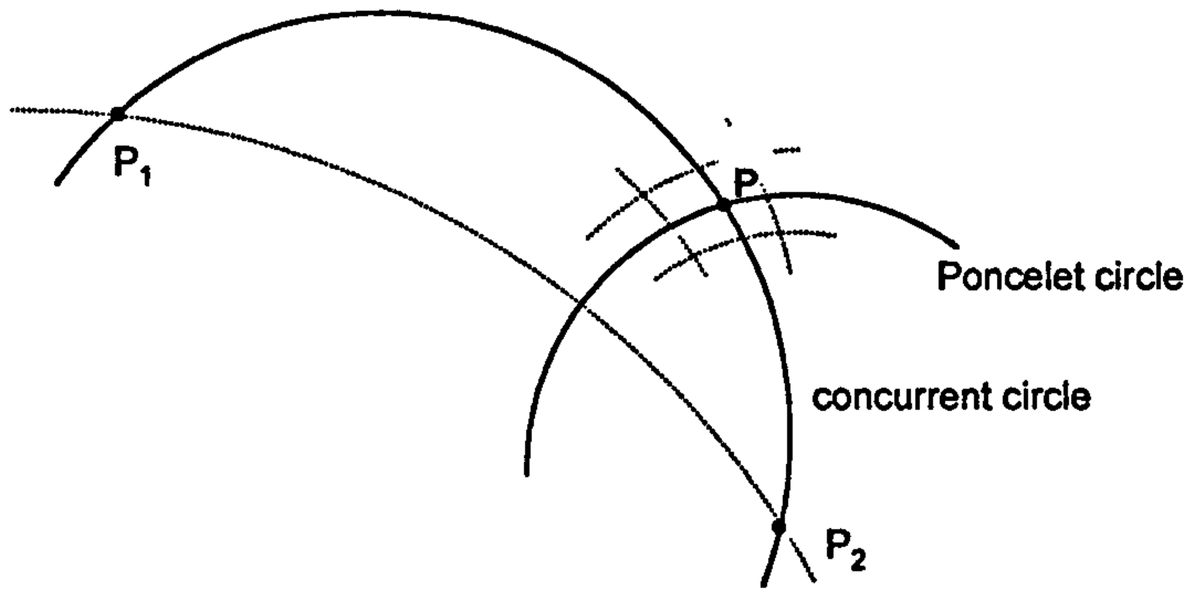


Figure 7.1 Intersecting and Poncelet circles passing through P and defined by points P_1 and P_2

Algebraically, the member of the intersecting pencil through P is given simply by $s = (p_1 \wedge p_2 \wedge p)\tilde{}$. As suggested above, the corresponding Poncelet circle through P could be found by constructing a second member of the intersecting pencil by creating the line joining P_1 to P_2, i.e. circle, $s' = (p_1 \wedge p_2 \wedge n)\tilde{}$, then forming the bivector $s \wedge s'$. This bivector could then be used to map P progressively to P_1 then P_2. The points P, P_1 and P_2 could then be used to create the required Poncelet circle.

An alternative way to find the Poncelet circle through P is to recognise that points are extreme cases of circles and that the (conformal) points p_1 and p_2 are in fact members of the Poncelet pencil which can therefore be described parametrically as

$$s = a\,p_1 + b\,p_2 . \qquad (7.1)$$

The required member of the pencil passes through p, so

$$(a\,p_1 + b\,p_2)\cdot p = 0 .$$

Hence

$$a(p_1 \cdot p) = -b(p_2 \cdot p) = k, \text{ say.}$$

Solving for a and b

$$a = +k/(p_1 \cdot p),$$
$$b = -k/(p_2 \cdot p).$$

Substituting in 7.1

$$s = k\frac{p_1}{p_1 \cdot p} - k\frac{p_2}{p_2 \cdot p},$$

i.e.

$$s = K(p_1(p_2 \cdot p) - p_2(p_1 \cdot p)), \text{ where } K = k(p_1 \cdot p)(p_2 \cdot p)).$$

Ignoring the multiplicative constant K, s is also represented homogeneously by

$$s = p_1(p_2 \cdot p) - p_2(p_1 \cdot p). \qquad (7.2)$$

7.3 Euclidean circle through P with centre C

As formula 7.2 relates vectors in conformal space, it is feasible to let $p_1 = n$, the point at infinity. If p_2 is renamed c, then 7.2 becomes

$$s = n(c \cdot p) - c\,(n \cdot p). \qquad (7.3)$$

In R^2, s is the circle through P with centre C, see figure 7.2.

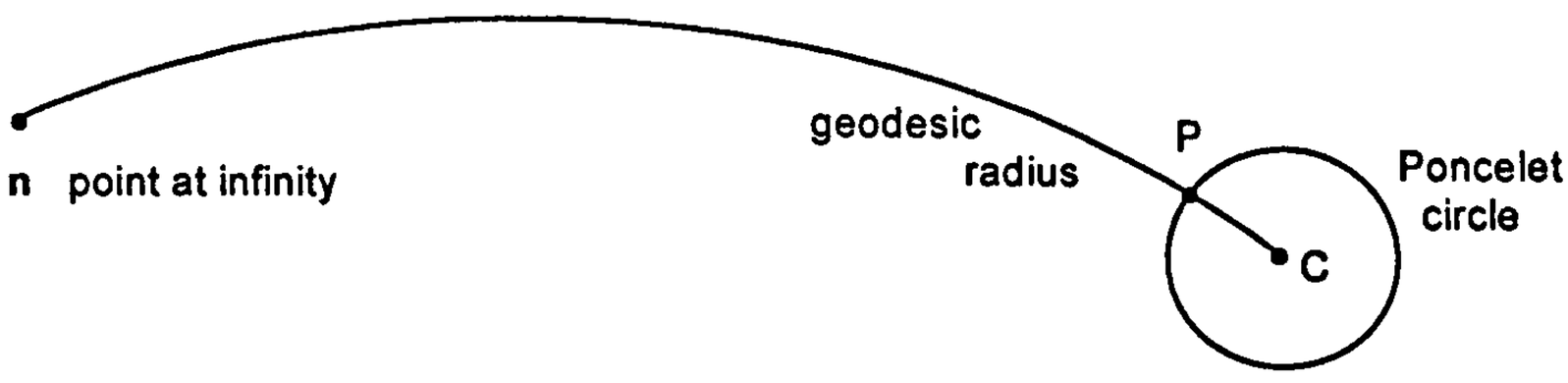


Figure 7.2 Euclidean circle through P with centre C.
(The circle is the Poncelet circle through P
defined by C and the point at infinity n.)

The generalised circle (p^c^n)~ is the geodesic diameter through P and C and is a member of the intersecting pencil through n and c.

7.4 Hyperbolic circle through P with centre C not on the horizon

In conformal space it turns out that it is also valid to modify 7.3 by letting $n = t$ where, for now, t is the geometry-defining-vector of the Poincaré disc model of hyperbolic space. Formula 7.3 then becomes

$$s = t(c \cdot p) - c\,(t \cdot p) \tag{7.4}$$

In this case, s defines the *hyperbolic* circle through P with *hyperbolic* centre C, see figure 7.3.

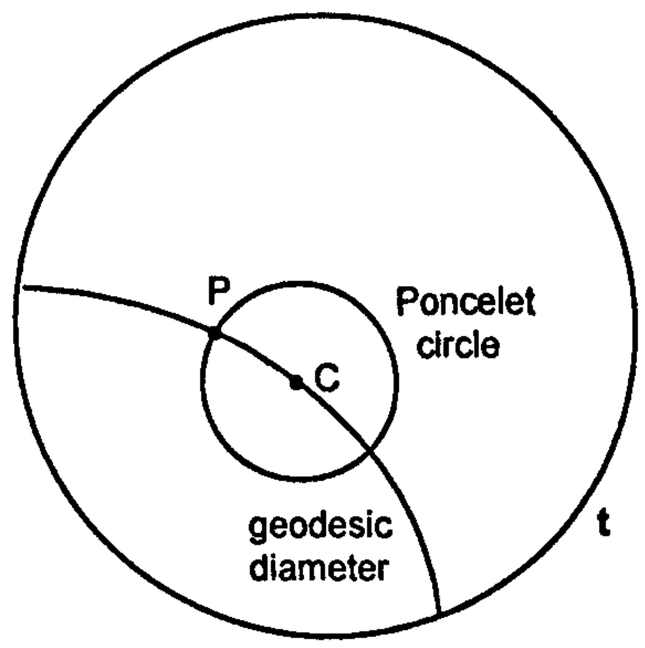


Figure 7.3 Hyperbolic circle through P with hyperbolic centre C.
(The circle is the Poncelet circle through P defined
by C and the geometry-defining-vector t.)

With reference to the original algebra presented, this generalisation works because the geometry-defining vector t is in fact the horizon, and the conformal vectors t and c still generate a Poncelet pencil. The corresponding member of the intersecting pencil through P is still (p^c^t)˜. This is the hyperbolic diameter through P.

7.5 Hyperbolic circle through P with centre C on the horizon

It turns out that the formula 7.4 still applies if the centre C is on the horizon t. However, in this case the pencil defined by t and c is a *tangent* pencil. Also, in this case, the circle (p^c^t)˜ is a member of the dual tangent pencil through C, see figure 7.4.

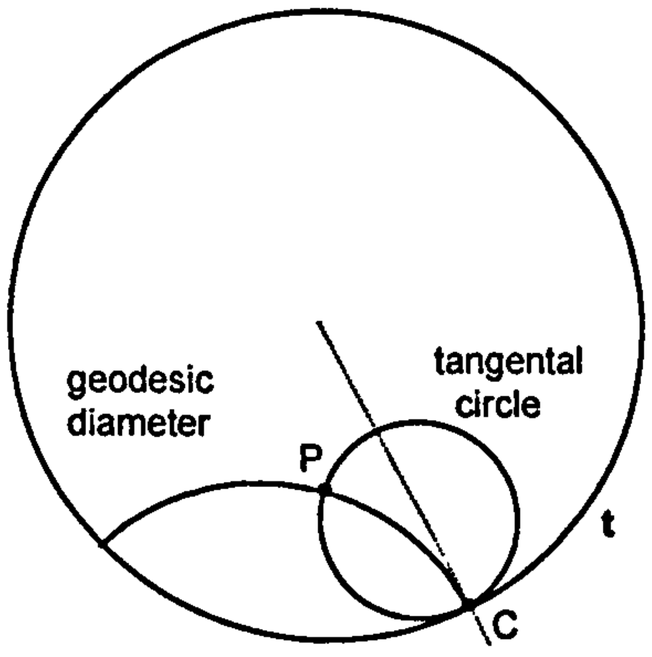


Figure 7.4 Hyperbolic circle through P with centre C on the horizon. (The circle is the tangent circle through P dual to the tangent pencil defined by C and the horizon t.)

In conformal space, the bivector B = t^c is null and therefore represents a plane touching the null cone along the vector c. The dual bivector B˜ also touches the null cone along c but represents the dual tangent pencil at C. The diameter through P passes through C and is a member of this dual tangent pencil.

7.6 Spherical circle through P with centre C

Remarkably, formula 7.4 also works in the spherical case provided t is now the geometry-defining-vector for the hemisphere model. In this case, the vector t, having negative signature, has no direct interpretation in the model. Nevertheless the conformal vectors t and c parametrically generate a Poncelet pencil in the normal way. Figure 7.5 shows the member of the pencil that passes through P.

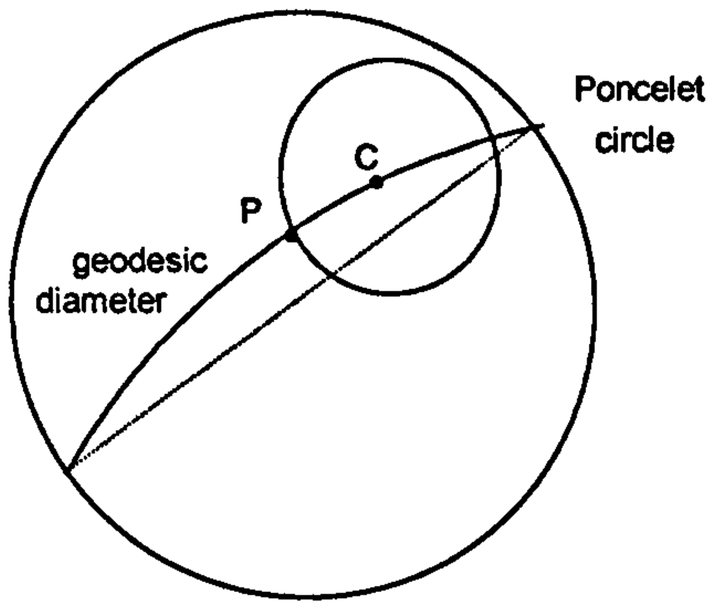


Figure 7.5 Spherical circle through P with spherical centre C.
(The circle is the Poncelet circle through P defined by C and the geometry-defining-vector t = e_4.)

In fact, the bivector representing the Poncelet pencil, t^c, has positive signature and so cuts the null cone twice, once along the null vector c. The other line of intersection with the null cone, a null vector, represents the other centre C' of the dual Poncelet pencil, see figure 7.6.

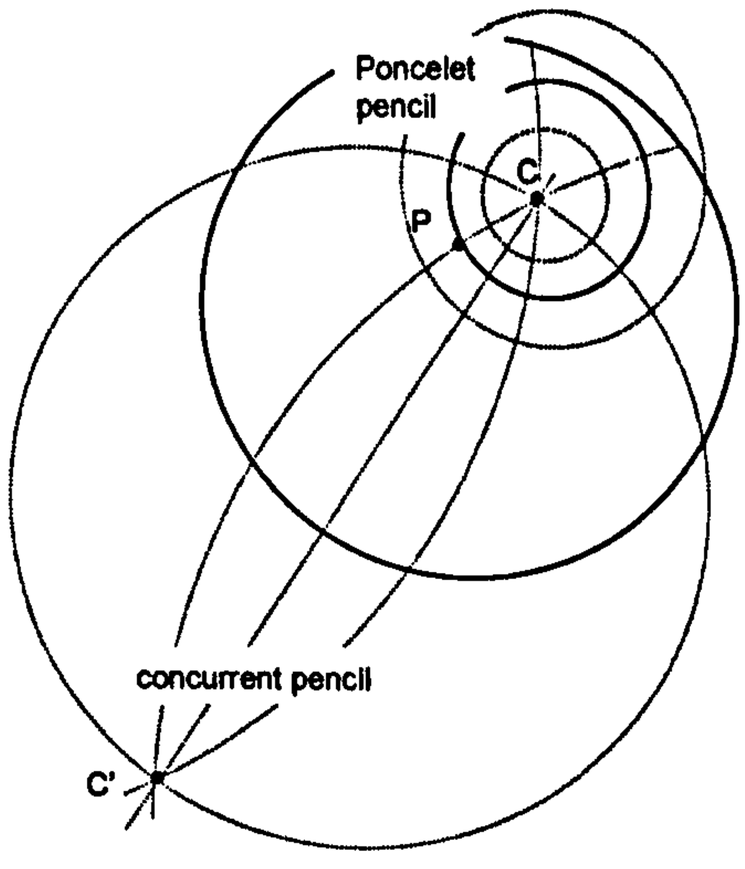


Figure 7.6 Intersecting pencil of diameters of a spherical circle with centre C. The concentric circles form the dual Poncelet pencil.

This is pictured in conformal space in figure 7.7 where c is on the *upper* null cone, i.e. on the upper hemisphere of S^2, because its stereographic projection C lies *inside* the unit circle, the projection of the equator. In contrast c' lies on the

lower hemisphere of S^2 so that C' lies outside the unit circle. Note that the diagram 'lacks a dimension' as explained in chapter 1.

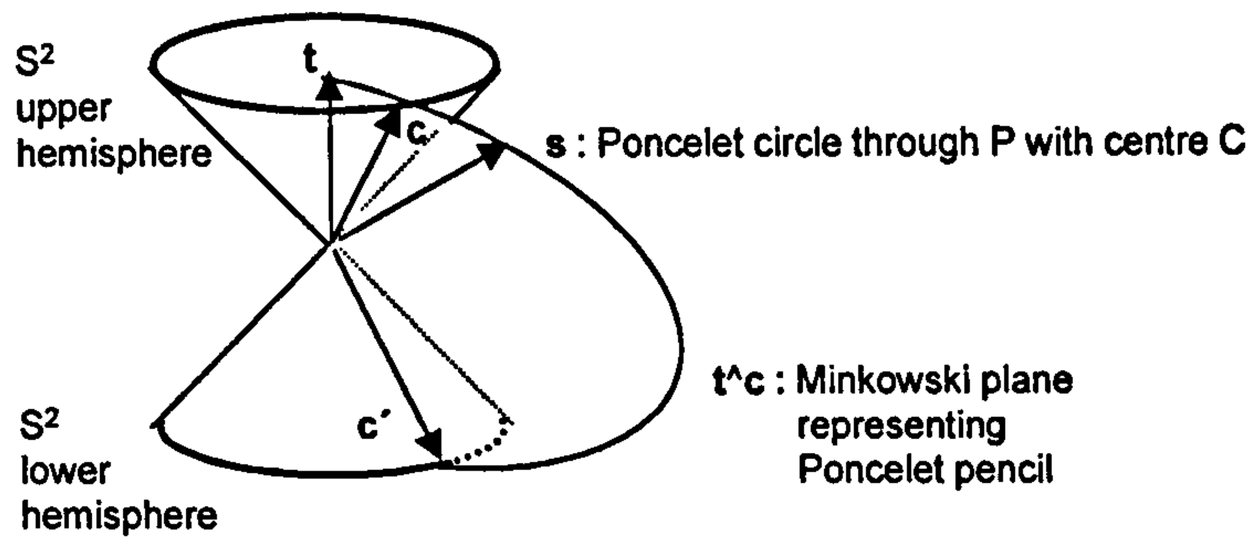


Figure 7.7 Conformal representation of Poncelet pencil of spherical circles concentric to the circle s through P with centre C.

7.7 Deriving the translation formula for Euclidean space

Chapter 4 states that if t is the geometry-defining vector, then the bivector (p^q^t)t generates a translation from P to Q. This section provides a justification for Euclidean space.

In chapter 6, cartesian co-ordinates were formed from a dual Poncelet-intersecting pencil by letting the two centres of the dual pencil both tend to the point at infinity n. This is equivalent to forming a dual tangent pencil at n. Thus, a Euclidean translation from point P to point Q can be generated by the bivector representing the tangent pencil centred at n with circles through P and n, and through Q and n, as members, see figure 7.8.

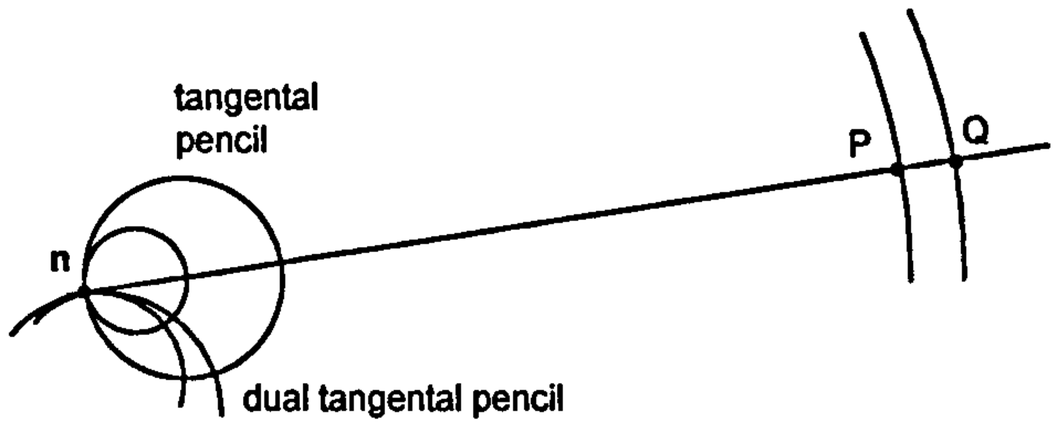


Figure 7.8 Generating a Euclidean translation from P to Q by constructing dual tangent pencils at the point-at-infinity n.

As extreme cases, the dual tangent pencil contains the point n and the geodesic through P and Q, namely (p^q^n)˜. Thus the pencil is represented by the bivector

$$((p \wedge q \wedge n)^{\sim}) \wedge n.$$

The dual tangent pencil is represented by

$$B = (((p \wedge q \wedge n)^{\sim}) \wedge n)^{\sim}. \qquad (7.5)$$

Algebraically, this is equivalent to the bivector cited in [4] and introduced in chapter 4, section 4.11, namely

$$B = (p \wedge q \wedge n)n. \qquad (7.6)$$

This equivalence can be shown using the 'deMorgan rules' for a multivector M and a vector v [18, p8]

$$M^{\sim} \cdot v = (M \wedge v)^{\sim} \quad \text{and} \quad M^{\sim} \wedge v = (M \cdot v)^{\sim}.$$

Letting $M = x \wedge y \wedge t$ it follows that

$$\begin{aligned} (x \wedge y \wedge t)\, t &= Mt \\ &= M \cdot t + M \wedge t \\ &= M \cdot t + x \wedge y \wedge t \wedge t \\ &= M \cdot t \end{aligned}$$

whereas

$$((x \wedge y \wedge t)\tilde{} \wedge t)\tilde{} = (M\tilde{} \wedge t)\tilde{} = M.t$$

7.8 Translations in hyperbolic geometry

The experimental work in Chapter 4, together with comments in [4, page 373] suggest that 7.5 generalises to

$$B = (p\hat{}q\hat{}t)t\,, \tag{7.7}$$

where t is the geometry-defining-vector for any geometry supported by the model. If this is written in the form of 7.4, namely

$$B = (((p\hat{}q\hat{}t)\tilde{})\hat{}t)\tilde{}, \tag{7.8}$$

then the formulae can once again be interpreted in terms of dual pencils for hyperbolic geometries, see figure 7.9 which shows circular and flat horizons.

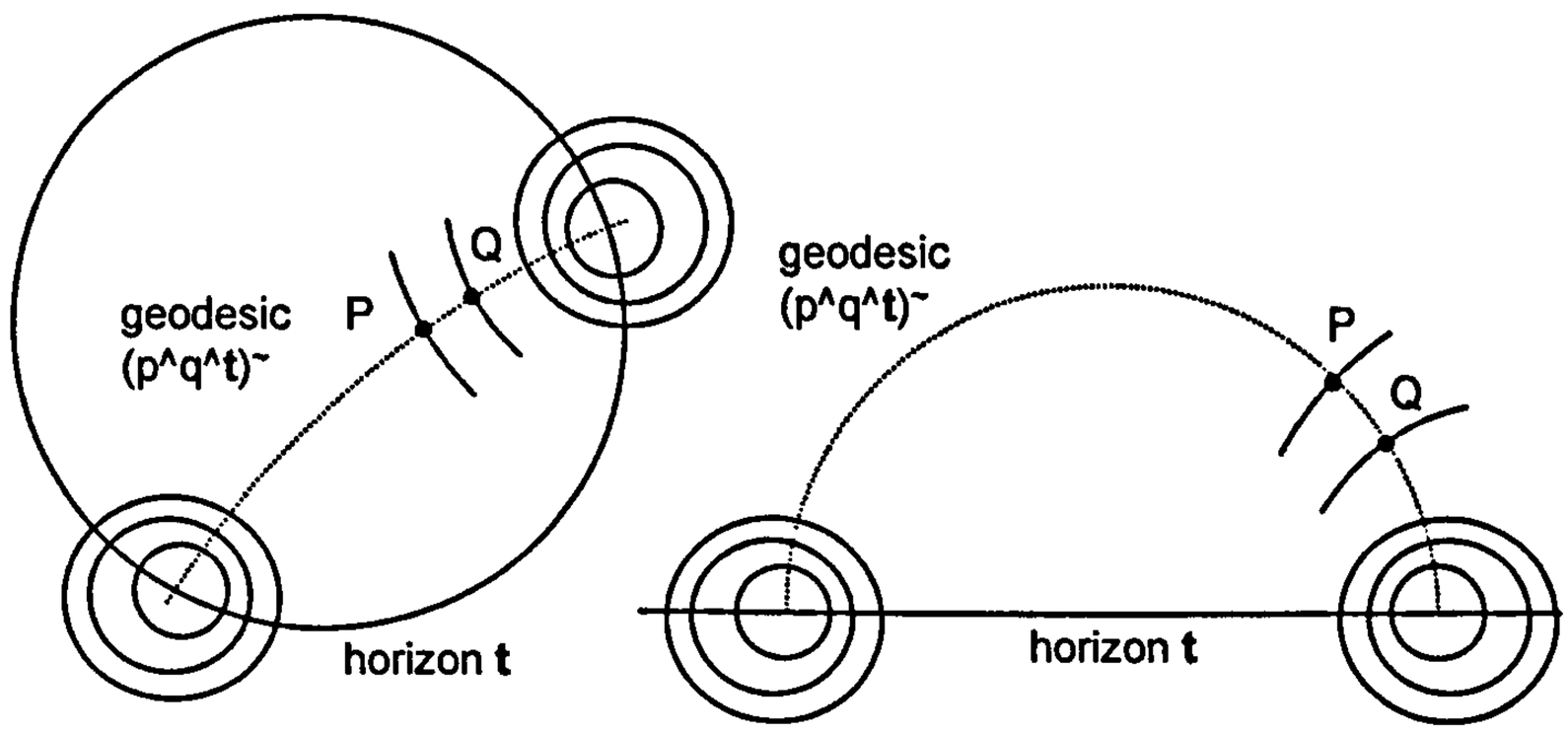


Figure 7.9 Poncelet pencils for generating a hyperbolic translation from P to Q. (The pencils are dual to the intersecting pencils with the horizon t and the geodesic through P and Q as members.)

In this case ((p^q^t)~)^t represents an intersecting pencil. The two points of concurrency are the points where the geodesic through P and Q meets the

horizon t. The dual of this bivector represents the dual Poncelet pencil and generates a translation from P to Q.

7.9 Conclusion

This chapter has shown that analysis in terms of pencils is a powerful tool for visualisation and interpretation, especially when taking account of how the bivector plane representing a pencil meets the null cone.

Three important generic computational formulae were discussed

geodesic through P and Q	p^q^t
circle through P centre C	$t(c\cdot p) - c\,(t\cdot p)$
translation bivector, P to Q	(p^q^t)t

The first was assumed and the second was derived and then generalised.

The third was cited in chapter 4 but the original citation lacked derivation. Though the formula was 'seen to work' in the relevant geometries, this chapter has used a pencil based analysis to provide an independent theoretical justification. However, in relation to the approach taken here, the formula has a more intuitive feel when written in the equivalent form (((p^q^t)~)^t)~.

8 Taylor approximations

8.1 Introduction

Implementing a pure dilation or rotation entails creating a bivector-generated transformation which maps one circle s_1 to another circle s_2. For a rotation (about a finite centre) the circles are geodesics through the centre of rotation and are therefore members of an intersecting pencil through the centre and the point at infinity. The bivector $s_1 \wedge s_2$ representing the pencil has negative signature. For a dilation the circles are concentric about the centre of dilation, and they form a Poncelet pencil with the common centre and the point at infinity being the two 'Poncelet' points of the pencil. The bivector $s_1 \wedge s_2$ has positive signature. See figure 8.1.

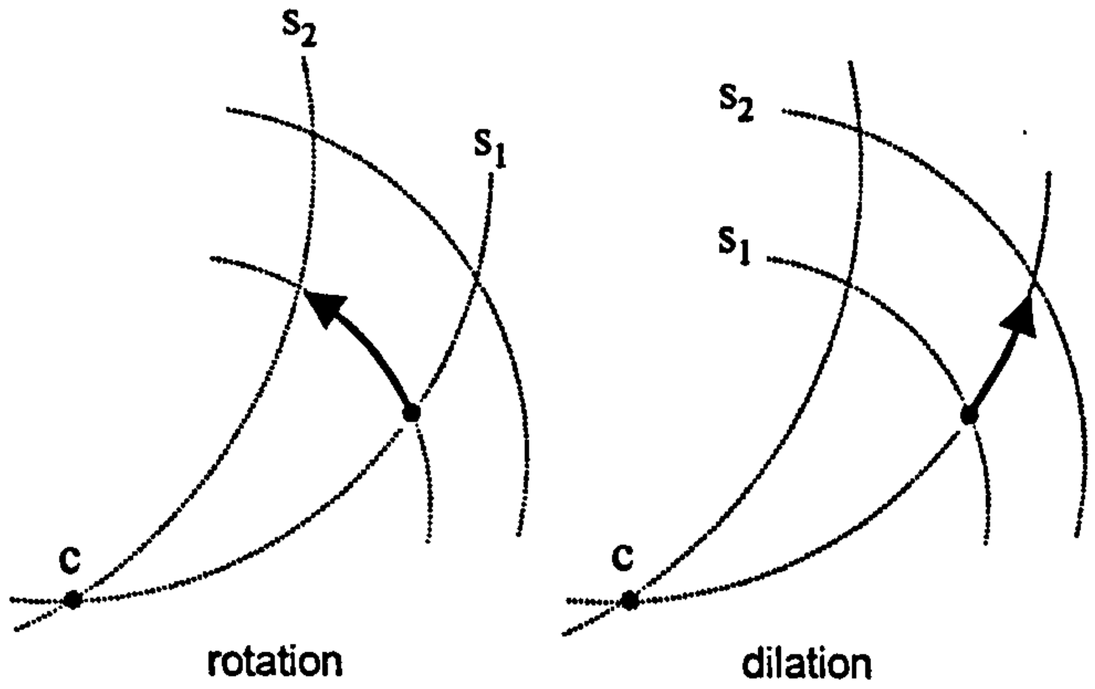


Figure 8.1 Rotation and dilation about a centre C viewed as circle to circle mappings.

In both cases the scaled bivector $B = -(s_1 \wedge s_2)/(s_1 . s_2)$ seems to generate the appropriate mapping. However, on testing, inconsistencies were found, suggesting this rather elegant formula is an approximation of some sort.

The independent analysis presented here elucidates the basis of this approximation and then uses the ideas generated to discuss rotations and dilations about centres at infinity.

8.2 Preliminary results and assumptions

It is assumed that $(e^{kB})^{-1} = e^{-kB}$, that is to say that an inverse mapping corresponding to a particular parameter value k is generated by changing the sign of the parameter.

The algebra in the next sections also needs the following result:

if $B = s_1 \wedge s_2$ then s_1 and s_2 anti-commute with B.

This follows from the fact that s_1^2 is always a scalar, since

$$s_1 B = s_1 (s_1 \wedge s_2) = \tfrac{1}{2} s_1 (s_1 s_2 - s_2 s_1) = \tfrac{1}{2} (s_1 s_1 s_2 - s_1 s_2 s_1) = \tfrac{1}{2} (s_1^2 s_2 - s_1 s_2 s_1)$$
$$B s_1 = (s_1 \wedge s_2) s_1 = \tfrac{1}{2} (s_1 s_2 - s_2 s_1) s_1 = \tfrac{1}{2} (s_1 s_2 s_1 - s_2 s_1 s_1) = \tfrac{1}{2} (s_1 s_2 s_1 - s_1^2 s_2).$$

8.3 Analysis of the hyperbolic case, $B^2 > 0$

If $B = k\hat{B}$ generates the transformation that maps circle s_1 to s_2 then

$$s_2 = e^{k\hat{B}} s_1 e^{-k\hat{B}}$$

and

$$s_1 = e^{-k\hat{B}} s_2 e^{k\hat{B}}.$$

If $B^2 > 0$ then

$$e^{k\hat{B}} = \cosh(k) + \hat{B}\sinh(k)$$
$$e^{-k\hat{B}} = \cosh(k) - \hat{B}\sinh(k)$$

so the first mapping can be written as

$$s_2 = \left(\cosh\frac{k}{2} + \hat{B}\sinh\frac{k}{2}\right) s_1 \left(\cosh\frac{k}{2} - \hat{B}\sinh\frac{k}{2}\right)$$

$$= \left(\cosh^2\frac{k}{2}\right)s_1 + \sinh\frac{k}{2}\cosh\frac{k}{2}\left(\hat{\mathrm{B}}\,s_1 - s_1\,\hat{\mathrm{B}}\right) - \left(\sinh^2\frac{k}{2}\right)\hat{\mathrm{B}}\,s_1\,\hat{\mathrm{B}}$$

$$= \left(\cosh^2\frac{k}{2} - \sinh^2\frac{k}{2}\right)s_1 + 2\sinh\frac{k}{2}\cosh\frac{k}{2}\left(\hat{\mathrm{B}}\,s_1\right)$$

hence $$s_2 = \cosh k\; s_1 + \sinh k\;\hat{\mathrm{B}}\,s_1 \qquad (8.1a)$$

and inversely $$s_1 = \cosh k\; s_2 - \sinh k\;\hat{\mathrm{B}}\,s_2\,. \qquad (8.1b)$$

Forming the geometric product gives

$$s_2 s_1 = \cosh^2 k\,(s_1 s_2) - \sinh^2 k\;\,(\hat{\mathrm{B}}\,s_1\,\hat{\mathrm{B}}\,s_2) + \cosh k \sinh k\,(\hat{\mathrm{B}}\,s_1 s_2 - s_1\,\hat{\mathrm{B}}\,s_2)$$
$$= (\cosh^2 k + \sinh^2 k)\,s_1 s_2 + (2\cosh k \sinh k)\,\hat{\mathrm{B}}\,s_1 s_2,$$

therefore $$s_2 s_1 = \cosh 2k\; s_1 s_2 + \sinh 2k\;\hat{\mathrm{B}}\,s_1 s_2 \qquad (8.2a)$$

and inversely $$s_1 s_2 = \cosh 2k\; s_2 s_1 - \sinh 2k\;\hat{\mathrm{B}}\,s_2 s_1\,. \qquad (8.2b)$$

Adding and using the fact that

$$s_1 \cdot s_2 = s_2 \cdot s_1 = (s_1 s_2 + s_2 s_1)/2$$

and $$s_1 \wedge s_2 = -s_2 \wedge s_1 = (s_1 s_2 - s_2 s_1)/2$$

gives $$2s_2 \cdot s_1 = 2\cosh 2k\;\, s_1 \cdot s_2 - 2\sinh 2k\;\,\hat{\mathrm{B}}(s_1 \wedge s_2)\,.$$

Substituting $$\hat{\mathrm{B}} = (s_2 \wedge s_1)/|s_2 \wedge s_1| \;\text{ where }\; (s_2 \wedge s_1)^2 = |s_2 \wedge s_1|^2$$

yields $$2s_2 \cdot s_1 = 2\cosh 2k\;\, s_1 \cdot s_2 - 2\sinh 2k\;\,|s_1 \wedge s_2|\,.$$

Therefore $$(1 - \cosh 2k)\;\, s_1 \cdot s_2 = \sinh 2k\;\,|s_1 \wedge s_2|\,,$$

which reduces to

$$\frac{|s_1 \wedge s_2|}{s_1 \cdot s_2} = -\tanh k\,.$$

So that

$$\mathrm{B} = k\hat{\mathrm{B}} = \left(\tanh^{-1}\frac{-|s_1 \wedge s_2|}{s_1 \cdot s_2}\right)\frac{s_1 \wedge s_2}{|s_1 \wedge s_2|}. \qquad (8.3)$$

If |x|<1, the Taylor expansion for $\tanh^{-1}$x is

$$\tanh^{-1} x = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \cdots.$$

Applying this to 7.2 above, and ignoring all terms except the first, gives the approximation

$$\mathrm{B} \approx -\frac{s_1 \wedge s_2}{s_1 \cdot s_2} \quad \text{when} \quad |s_1 \wedge s_2| / s_1 \cdot s_2 \text{ is small.} \qquad (8.4)$$

8.4 The spherical case, $B^2 < 0$

Applying the formula

$$e^{k\hat{B}} = \cos(k) + \hat{\mathrm{B}}\sin(k)$$
$$e^{-k\hat{B}} = \cos(k) - \hat{\mathrm{B}}\sin(k)$$

to the transformation formulae

$$s_2 = e^{k\hat{B}} s_1 e^{-k\hat{\mathrm{B}}}$$

$$s_1 = e^{-k\hat{B}} s_2 e^{k\hat{\mathrm{B}}}$$

yields

$$s_2 = \cos k\, s_1 + \sin k\, \mathrm{B} s_1 \qquad (8.5a)$$

$$s_1 = \cos k\, s_2 - \sin k\, \hat{\mathrm{B}} s_2. \qquad (8.5b)$$

Forming the Clifford products gives

$$s_2 s_1 = \cos 2k\ s_1 s_2 + \sin 2k\ \hat{\mathrm{B}}\, s_1 s_2 \qquad (8.6a)$$

$$s_1 s_2 = \cos 2k\ s_2 s_1 - \sin 2k\ \hat{\mathrm{B}}\, s_2 s_1, \qquad (8.6b)$$

leading to

$$\mathrm{B} = k\hat{\mathrm{B}} = \left(\tan^{-1} \frac{-|s_1 \wedge s_2|}{s_1 \cdot s_2} \right) \frac{s_1 \wedge s_2}{|s_1 \wedge s_2|}. \qquad (8.7)$$

If $|x|<1$, the Taylor expansion for $\tan^{-1}x$ is

$$\tan^{-1} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

once again yielding the approximation

$$\mathrm{B} \approx -\frac{s_1 \wedge s_2}{s_1 \cdot s_2} \quad \text{when} \quad |s_1 \wedge s_2| / s_1 \cdot s_2 \text{ is small.} \qquad (8.8)$$

8.5 The 'Euclidean' case, $B^2 = 0$

The Euclidean case ($B^2 = 0$) arises when the centre of rotation or dilation is the common point of contact of a tangent pencil, see figure 8.2 which shows mutually orthogonal pencils with common point of contact c. Each is represented by a null bivector.

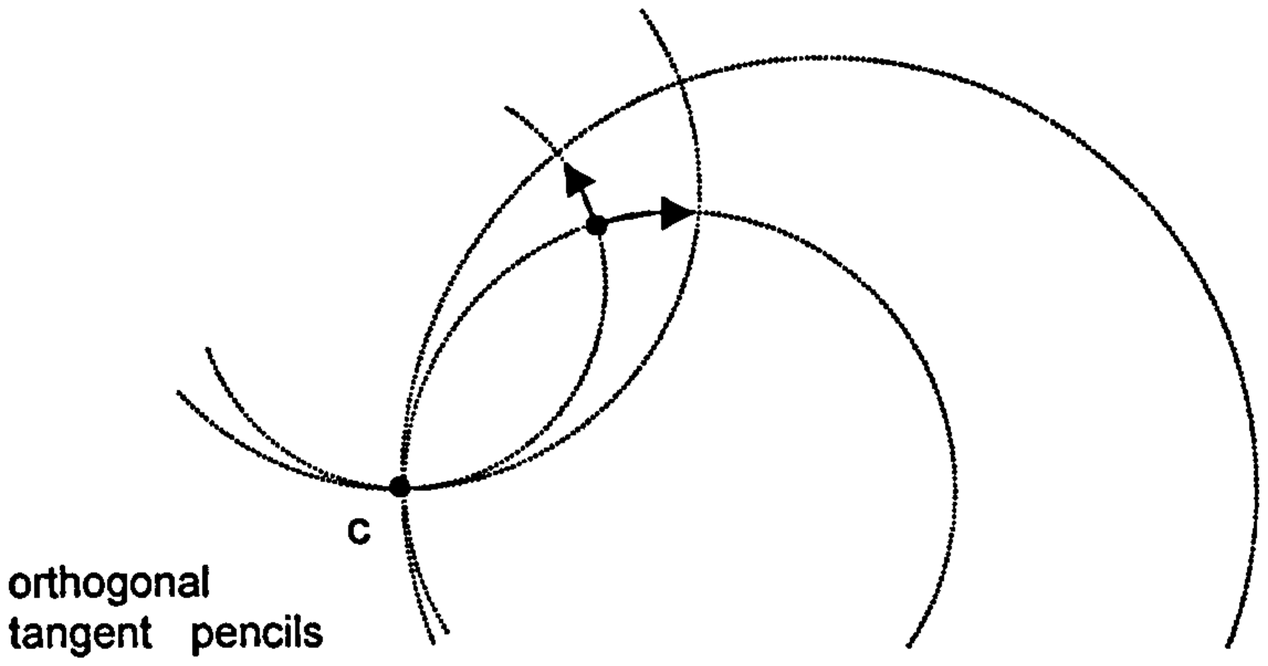


Figure 8.2 Mutually orthogonal mappings induced by dual tangent pencils.

In fact, in this case, the terms 'dilation' and 'rotation' have no clear meaning unless one of the pencils represents geodesics in the imposed geometry.

Suppose a mapping maps a circle s_1 in one pencil to another circle s_2 in the same pencil then, as usual, the pencil is represented by $s_1 \wedge s_2$ where $s_1 \wedge s_2$ is now null. The *exact* mapping from s_1 to s_2 is assumed, as before, to be generated by a bivector of the form $B = k\hat{B}$, where $\hat{B} = s_1 \wedge s_2 / | s_1 \wedge s_2 |$. The problem, as before, is to find k and hence B.

In this case the initial Taylor expansions yield

$$e^{k\hat{B}} = 1 + k\hat{B}$$
$$e^{-k\hat{B}} = 1 - k\hat{B}.$$

Substituting into the transformation equations

$$s_2 = e^{k\hat{B}} s_1 e^{-k\hat{B}}$$
$$s_1 = e^{-k\hat{B}} s_2 e^{k\hat{B}}$$

gives

$$s_2 = s_1 + 2k\hat{B}s_1 \qquad (8.9a)$$
$$s_1 = s_2 - 2k\hat{B}s_2. \qquad (8.9b)$$

Clifford multiplication then gives

$$s_2 s_1 = s_1 s_2 + 4k\,\hat{\mathrm{B}}\, s_1 s_2 \tag{8.10a}$$

$$s_1 s_2 = s_2 s_1 - 4k\,\hat{\mathrm{B}}\, s_2 s_1 . \tag{8.10b}$$

Subtraction leads to

$$s_2 \wedge s_1 = s_1 \wedge s_2 + 4k\ \hat{\mathrm{B}}\, s_1 \cdot s_2$$

$$s_2 \wedge s_1 = s_1 \wedge s_2 + 4k \frac{(s_1 \wedge s_2)}{|s_1 \wedge s_2|} s_1 \cdot s_2$$

$$s_1 \wedge s_2 = -2k \frac{(s_1 \wedge s_2)}{|s_1 \wedge s_2|} s_1 \cdot s_2$$

$$k = -\frac{|s_1 \wedge s_2|}{2(s_1 \cdot s_2)}$$

$$\mathrm{B} = k\hat{\mathrm{B}} = -\frac{|s_1 \wedge s_2|}{2(s_1 \cdot s_2)} \frac{s_1 \wedge s_2}{|s_1 \wedge s_2|} .$$

Therefore

$$\mathrm{B} = -\frac{s_1 \wedge s_2}{2(s_1 \cdot s_2)} . \tag{8.11}$$

In this case, the formula is exact and not an approximation.

8.6 Centre at infinity

If the centre of a drag transformation is either the point at infinity in Euclidean space, or a point on the horizon of a hyperbolic space, then the bivectors

generating the rotation and dilation are both null and the simplified formula 8.11 applies.

8.7 Summary

The sometimes loosely stated claim that the bivector

$$B = -\frac{s_1 \wedge s_2}{s_1 \cdot s_2} \tag{8.12}$$

generates a transformation that maps circle s_1 to s_2 needs to be elaborated.

It is true (up to a factor of 2) only if the circles touch and thereby define a tangent pencil. In this case $B^2 = 0$.

If $B^2 > 0$ the circles do not intersect and so define a Poncelet pencil and the formula 8.12 is a Taylor approximation of the formula

$$B = k\hat{B} = \left(\tanh^{-1}\frac{-|s_1 \wedge s_2|}{s_1 \cdot s_2}\right)\frac{s_1 \wedge s_2}{|s_1 \wedge s_2|}.$$

If $B^2 < 0$ the circles intersect and define an intersecting pencil. Formula 8.12 is then a Taylor approximation to

$$B = k\hat{B} = \left(\tan^{-1}\frac{-|s_1 \wedge s_2|}{s_1 \cdot s_2}\right)\frac{s_1 \wedge s_2}{|s_1 \wedge s_2|}.$$

By writing k as

$$\tan k = \frac{-|\hat{s}_1 \wedge \hat{s}_2|}{\hat{s}_1 \cdot \hat{s}_2}$$

it follows that k is the angle between the conformal vectors s_1 and s_2 and therefore also between the intersecting circles. In this case, k is also given by

$$\cos k = \hat{s}_1 \cdot \hat{s}_2$$

$$\sin k = -|\, \hat{s}_1 \wedge \hat{s}_2 \,|.$$

This gives rise to the traditional rotational form of the exponential map

$$s' = \cos k + \hat{\mathrm{B}} \sin k \quad \text{where} \quad \cos k = \hat{s}_1 \cdot \hat{s}_2 .$$

8.8 Conclusion

As in the previous chapter, this chapter has provided an independent analysis of bivector generated transformations that map one circle onto another. Once again it was necessary to find theoretical underpinnings for what turned out to be somewhat unreliable results cited in the literature.

The analysis has implication when attempting to factor mouse-dragging operations into pure rotations and dilations. If the Taylor approximations are used it is likely that the object being dragged will not follow the mouse pointer exactly. However, a final correction could be made at the end of the cumulative drag operation by applying the correct non-approximated formula across the cumulative drag.

However, if the centre of dragging is a point on the horizon of a hyperbolic space, or the point at infinity of Euclidean space, then any mouse drag automatically factors into transformations represented by tangent pencils so formula 8.12 is exact (up to a factor of 2).

9 Transforming the Geometry

9.1 Introduction

This chapter initially presents two interface designs for transforming the geometry. They are developed for hyperbolic space then extended for spherical space. The pencil-based analysis used led to a number of theoretical results relating to diameters of circles and rotations with respect to various geometries. These ideas suggested further interface designs which are briefly outlined.

Ideas in this chapter have been incorporated into a small working prototype described in appendix A.

9.2 The problem

The geometry-defining vector t of a space defines the geodesics - a circle s is a geodesic if

$$s \cdot t = 0. \qquad (9.1)$$

For hyperbolic space, t has positive signature and the circle it represents is the horizon. Equation 9.1 encodes the fact that geodesics meet the horizon orthogonally.

For Euclidean space, t is null and represents a single 'point at infinity'. Equation 9.1 encodes the fact that (straight line) geodesics pass through the point at infinity. (Other interpretations view t as representing a 'circle at infinity' in which case geodesics meet it orthogonally.)

For spherical space, the geometry-defining vector t has negative signature so there is no visible horizon. In this case, equation 9.1 evidently encodes the fact that geodesics meet a certain 'equatorial-circle' at antipodal points. This has

been tacitly assumed thus far and seen to work in relevant examples. This chapter will examine this notion in more detail.

These differing interpretations of the geometry-defining vector t, and therefore of equation 9.1, have consequences when considering how to change from one geometry to another. If both geometries are hyperbolic, it is simply a matter of changing from one circular (or straight line) horizon to another. Methods developed in the previous three chapters are appropriate for these circle-to-circle mappings.

If the geometries are spherical the situation is somewhat more complicated, as will be shown.

9.3 A first interface for changing hyperbolic geometry

The circular horizon of the standard Poincaré disc model of hyperbolic space has the unit circle e_3 as the horizon. Dilating this circle expands the horizon and reduces the effective curvature of the geometry.

The Poncelet pencil of dilated horizons concentric to e_3 has the origin and the point at infinity as Poncelet points. However, to paramatise the pencil, any two representative circles (including points) can be chosen. Using e_3 and n (the point or circle at infinity) as representatives the pencil is given by

$$\lambda e_3 + \mu n.$$

Values of λ and μ equal to 1 and 0 give the unit circle e_3, values of 0 and 1 give the circle at infinity n.

A simple slider can change values of λ of μ and so dilate the horizon from the initial unit circle up to the circle at infinity, see figure 9.1.

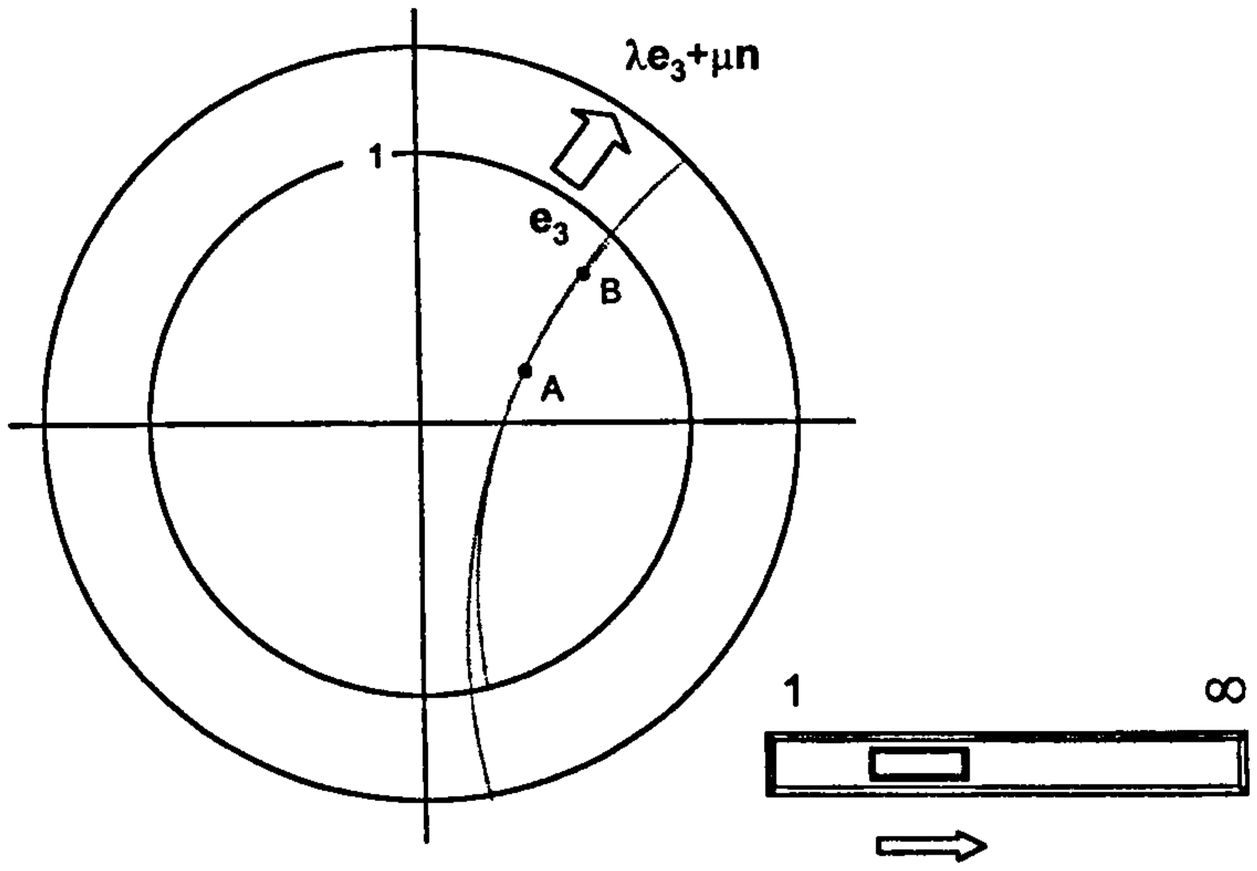


Figure 9.1 Using a slider to dilate the horizon of the Poincaré disc model of hyperbolic space.

The geometry-defining vector

$$t = \lambda e_3 + \mu n$$

determines the geodesic between two points A and B - it is the circle (a^b^t)~.

As the horizon dilates the geodesic flattens, see figure 9.1.

In this interface design, conformal geometric objects (i.e. points and circles) remain fixed on the screen as the underlying geometry changes. However, geometry related objects, e.g. geodesics as well as centres and diameters of circles would change as the geometry changed - their values would depend on the current value of the geometry-defining vector.

As the horizon dilates the proportion of the total hyperbolic space seen in the viewport decreases. However, the central area remains in view - it is the fixed centre of focus.

9.4 A second interface for changing hyperbolic geometry

The centre of focus of the next interface is the southern part of an expanding horizon. The variable horizon or geometry-defining vector is defined by

$$t = \lambda e_3 + \mu e_2.$$

This defines an intersecting pencil with the unit circle e_3 and the x-axis e_2 as members, see figure 9.2.

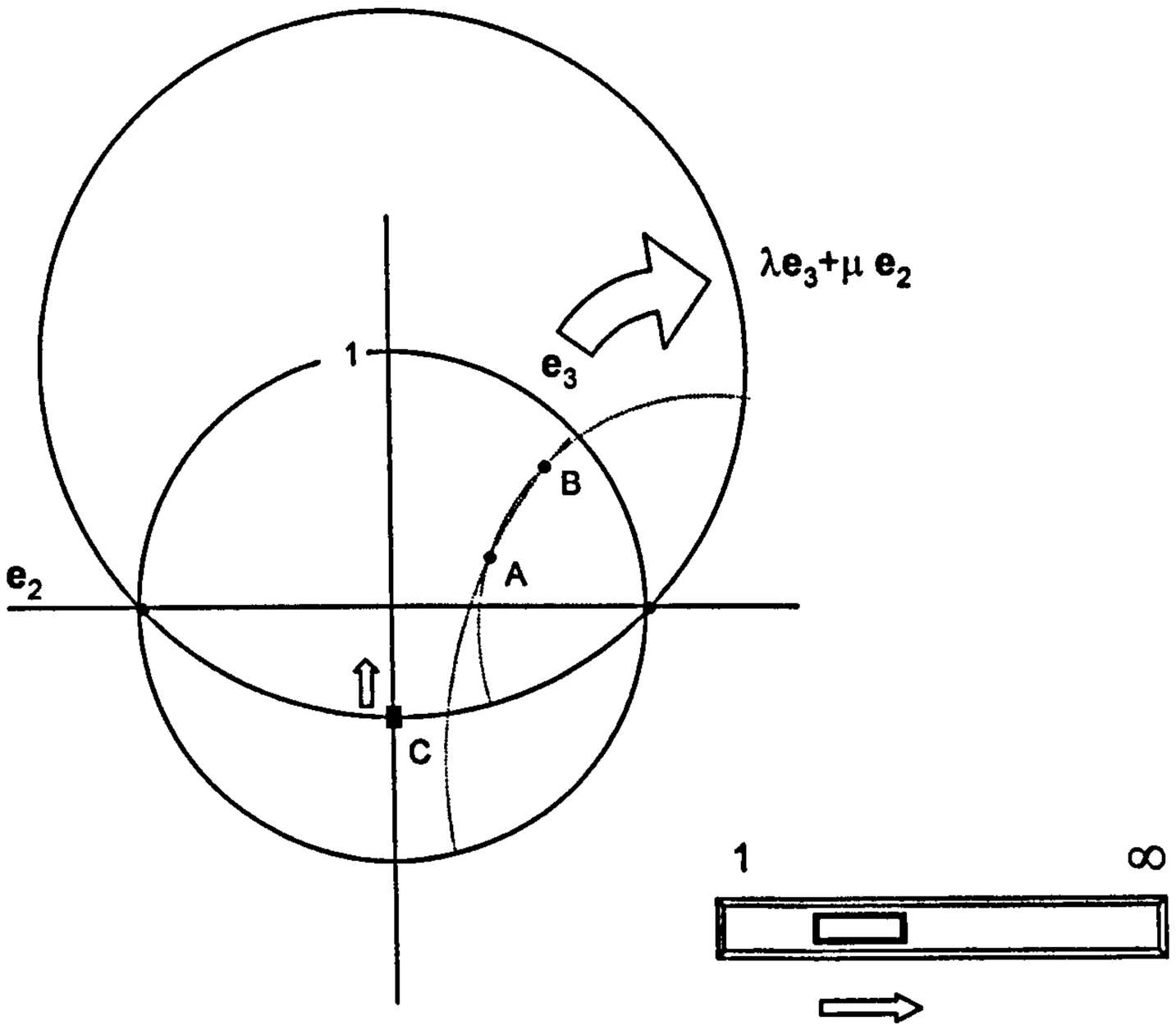


Figure 9.2 Using a slider to dilate and flatten the horizon of the Poincaré disc model of hyperbolic space so that it becomes the half space model.

Moving an equivalent slider would change the horizon from the unit circle of the Poincaré disc model to the horizontal horizon of the classical half-space model. The geometry therefore does not 'flatten out' but remains strongly hyperbolic throughout

In this case, an alternative to the slider would be to use an in-scene vertically-moveable control point C which would control where the current horizon cut the y-axis, see figure 9.2.

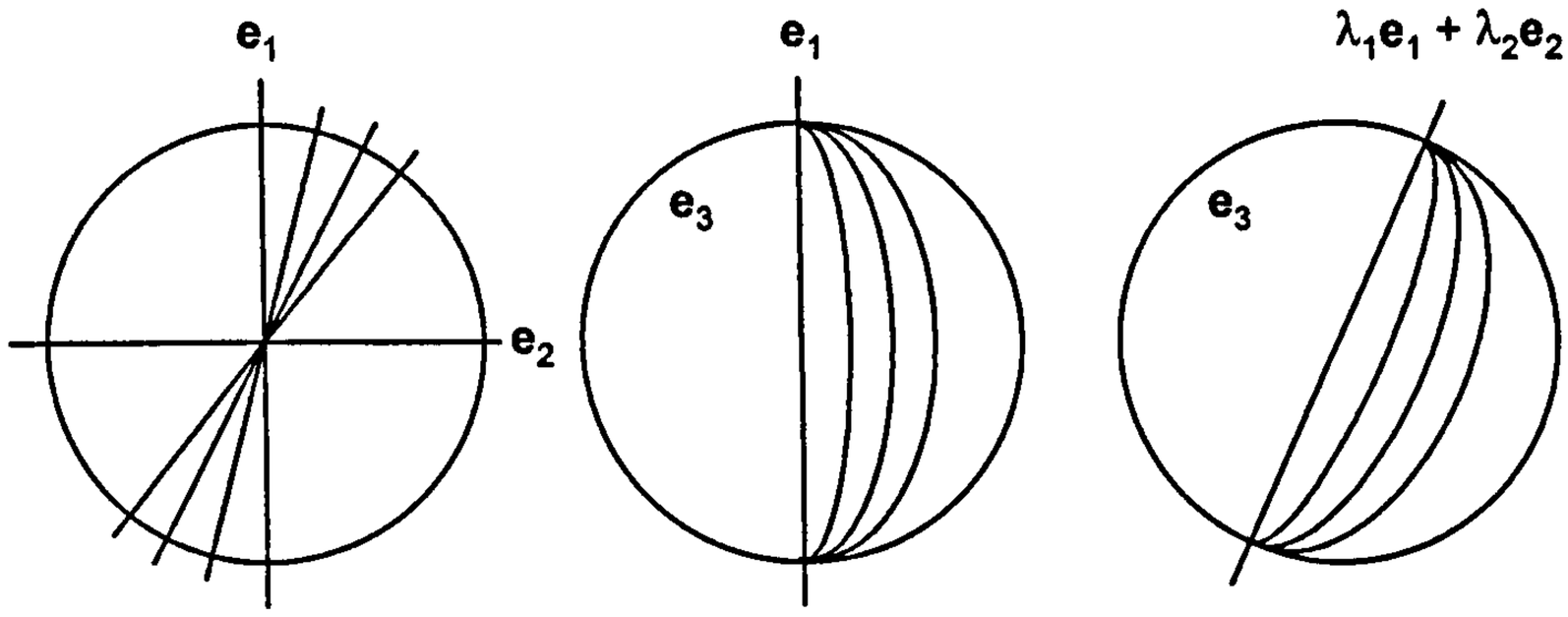
9.5 Constructing interfaces for changing spherical geometry

The two interfaces above could also be used to modify spherical space - it is a question of re-interpreting the meaning of the transformed unit circle so that it represents an equatorial circle r of a spherical space rather than the horizon of a hyperbolic space.

However, to be able to draw geodesics we need to be able to construct the geometry-defining vector t from r. To do so, we observe that the set of geodesics s that satisfy equation 9.1 defines a trivector orthogonal to t. Thus if we can represent the set of geodesics by a trivector, then its (orthogonal) dual will be the required vector t.

To see how this can work, consider the standard spherical model where the equatorial circle is the unit circle e_3. Any line through the origin is a geodesic and is a member of the intersecting pencil with the origin and the point at infinity as points of concurrency. This pencil can be represented by the two axes e_1 and e_2, so the pencil is represented by $\lambda_1 e_1 + \lambda_2 e_2$. This linear subspace is the bivector $e_1 \wedge e_2 = e_{12}$. (In fact, only sub-space vectors of *positive* signature represent circles - this necessary condition will be omitted in what follows.)

On the other hand the pencil $\lambda_1 e_1 + \lambda_3 e_3$, represented by the bivector e_{13}, is the set of geodesics that meet the unit equatorial circle at antipodal points where the vertical line e_1 meets the circle e_3, see figure 9.3.



The pencil $\lambda_1 e_1 + \lambda_2 e_2$

The pencil $\lambda_1 e_1 + \lambda_3 e_3$

The pencil $\sigma_1(\lambda_1 e_1 + \lambda_2 e_2) + \sigma_2\, e_3$

Figure 9.3 Pencils generated by the vertical axis e_1, the horizontal axis e_2 and the unit circle e_3.

The complete 3-parameter family of geodesics is taken by constructing the pencil formed from an arbitrary line through the origin, $\lambda_1 e_1 + \lambda_2 e_2$, and the unit circle e_3 (see figure 9.3)

$$\sigma_1(\lambda_1 e_1 + \lambda_2 e_2) + \sigma_2\, e_3$$
$$= \mu_1\, e_1 + \mu_2\, e_2 + \mu_3\, e_3.$$

This linear subspace is the trivector $e_1 \wedge e_2 \wedge e_3 = e_{123}$. Its dual $-e_4$ is (a scalar multiple of) the expected geometry-defining vector.

Generalising, possible computational steps for defining the geometry-defining vector t from the equatorial circle r are:

construct diameters m_1 and m_2,
form the trivector $m_1 \wedge m_2 \wedge r$ - this represents all the geodesics,
the geometry-defining vector t is then given by the dual $(m_1 \wedge m_2 \wedge r)\tilde{}$.

9.6 Modifying the first interface for spherical geometry

The first interface dilated the unit circle e_3 using the pencil $\lambda e_3 + \mu n$. Using the fact that $n = e_3 + e_4$, this circle could be represented homogeneously by

$$r = e_3 + \rho n = (1 + \rho)\, e_3 + \rho e_4.$$

For the spherical interface, this circle now represents an equatorial circle rather than an (hyperbolic) horizon. The horizontal and vertical diameters of this circle are still e_1 and e_2, so the trivector of geodesics is

$$\begin{aligned} & e_1 \wedge e_2 \wedge ((1 + \rho)\, e_3 + \rho e_4) \\ = \; & (1 + \rho)\, e_{123} + \rho_{124}. \end{aligned}$$

The geometry-defining vector t is the dual of this trivector

$$\begin{aligned} t = \; & ((1 + \rho)\, e_{123} + \rho_{124}) I_4^{-1} \\ = \; & ((1 + \rho)\, e_{123} + \rho_{124})(-e_{1234}) \\ = \; & (1 + \rho)\, e_4 + \rho e_3 + \rho \\ = \; & e_4 + \rho n. \end{aligned}$$

In this case, the changes in the original equatorial circle $r = e_3$ and the geometry-defining vector $t = e_4$ can be depicted in relation to the null cone, see figure 9.4.

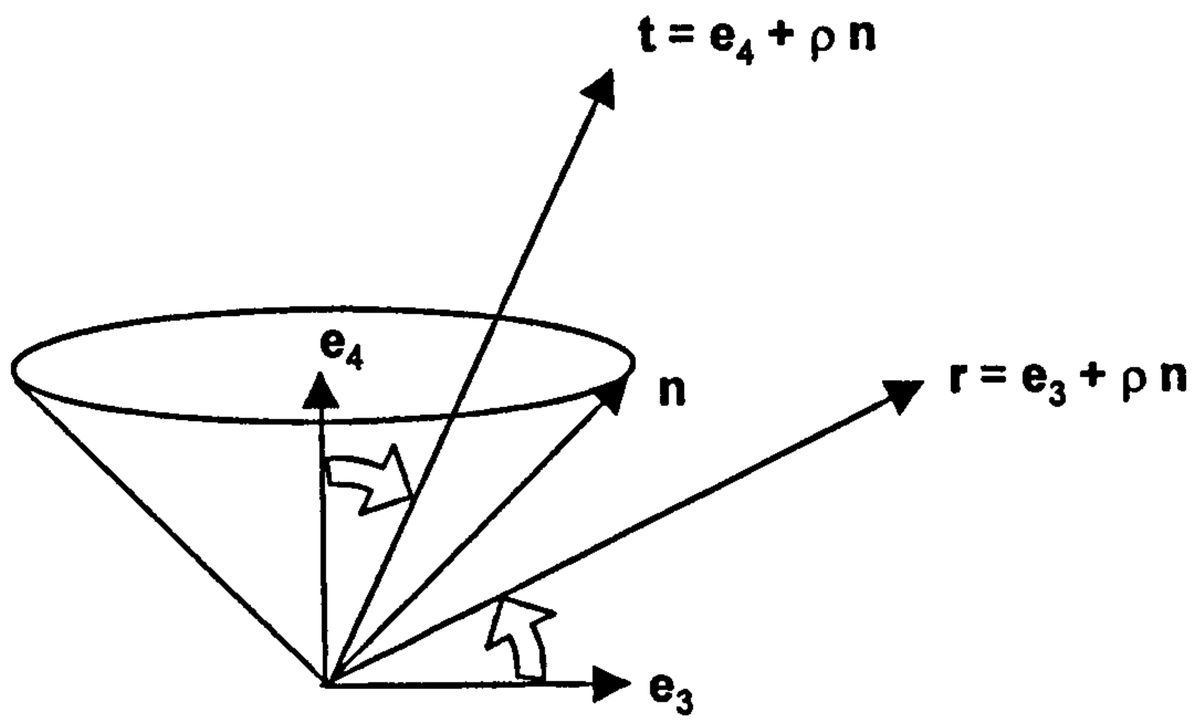


Figure 9.4 Orientations of the conformal representation of an equatorial circle of spherical space defined by r, and the corresponding geometry-defining vector t .

The dilation of the equatorial circle initially represented by e_3 is reflected in a rotation of e_3 toward e_4. This agrees with the analysis in chapter 1. The initial geometry-defining vector e_4 rotates by the same amount toward e_3. Both vectors remain in the embedded Minkowski plane defined by the bivector $e_{34} = e_3 \wedge e_4$.

9.7 Modifying the second interface for spherical geometry

The second interface altered the original unit circle e_3 by effectively adding an e_2 component to obtain the circle $r = e_3 + \rho e_2$.

If this is re-interpreted as an equatorial circle of spherical space, the horizontal diameter through it is of the form $e_2 + \lambda n$ (lines parallel to e_2 form a Poncelet pencil with the point at infinity n as one of the Poncelet centres). For this line to be a diameter of the circle $e_3 + \rho e_2$ it must meet it orthogonally. Hence

$$(e_3 + \rho e_2) \cdot (e_2 + \lambda n) = 0$$

$$(e_3 + \rho e_2) \cdot (e_2 + \lambda e_3 + \lambda e_4) = 0$$

$$\lambda + \rho = 0$$

$$\rho = -\lambda.$$

The horizontal diameter is therefore $e_2 - \rho n = e_2 - \rho e_3 - \rho e_4$,
the vertical diameter is still e_1.

The trivector of geodesics is therefore

$$\begin{aligned}
&(e_2 - \rho e_3 - \rho e_4) \wedge e_1 \wedge (e_3 + \rho e_2) \\
&= (e_{21} - \rho e_{31} - \rho e_{41}) \wedge (e_3 + \rho e_2) \\
&= e_{213} - \rho^2 e_{312} - \rho e_{413} - \rho^2 e_{412} \\
&= -(1 + \rho^2) e_{123} - \rho e_{413} - \rho^2 e_{412}.
\end{aligned}$$

The geometry-defining vector is given by the dual

$$\begin{aligned}
t &= (-(1 + \rho^2) e_{123} - \rho e_{413} - \rho^2 e_{412})(-e_{1234}) \\
&= (1 + \rho^2) e_4 + \rho e_2 - \rho^2 e_3.
\end{aligned}$$

If $\rho = 0$, $r = e_3$ while $t = e_4$ as expected.

Replacing ρ by $1/\lambda$, r and t are homogeneously given by

$$r = \lambda e_3 + e_2,$$

and

$$t = (\lambda^2 + 1) e_4 + \lambda e_2 - e_3.$$

Thus when $\lambda = 0$, $r = e_2$ and $t = e_4 - e_3$.

The behaviour of the r and t vectors in relation to the null cone is depicted in figure 9.5 where, as r rotates from e_3 to e_2, t rotates from e_4 to $e_4 - e_3$.

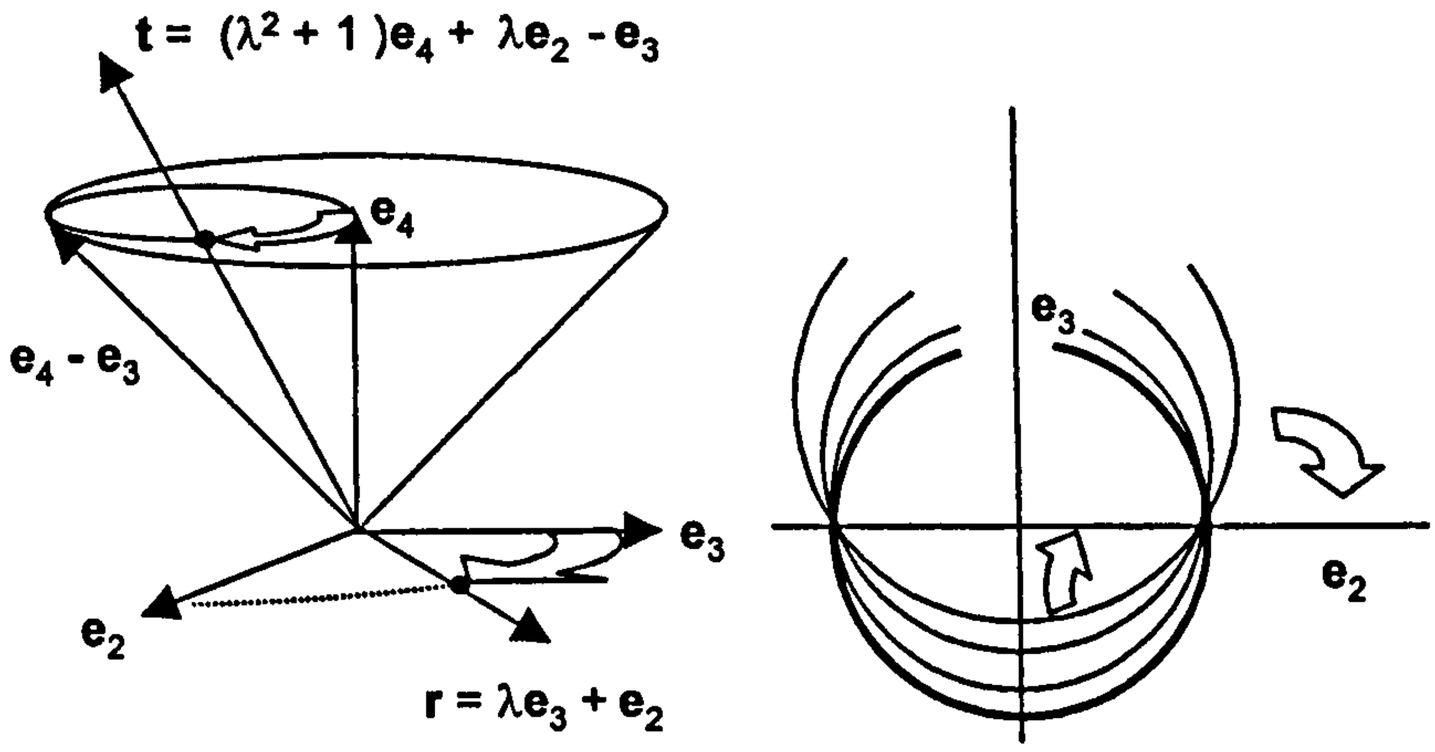


Figure 9.5 **Left:** Rotations in conformal space of the representation of an equatorial circle of spherical space defined by r, and the corresponding geometry-defining vector t .

Right: Dilation and flattening of the actual equatorial circle r due to the rotation of its representation in conformal space.

When r rotates from e_3 to e_2 in the opposite direction of rotation, t rotates from e_4 to $e_4 - e_3$, also in the opposite direction, see figure 9.6. This counter-rotation of r generates the other half of the pencil.

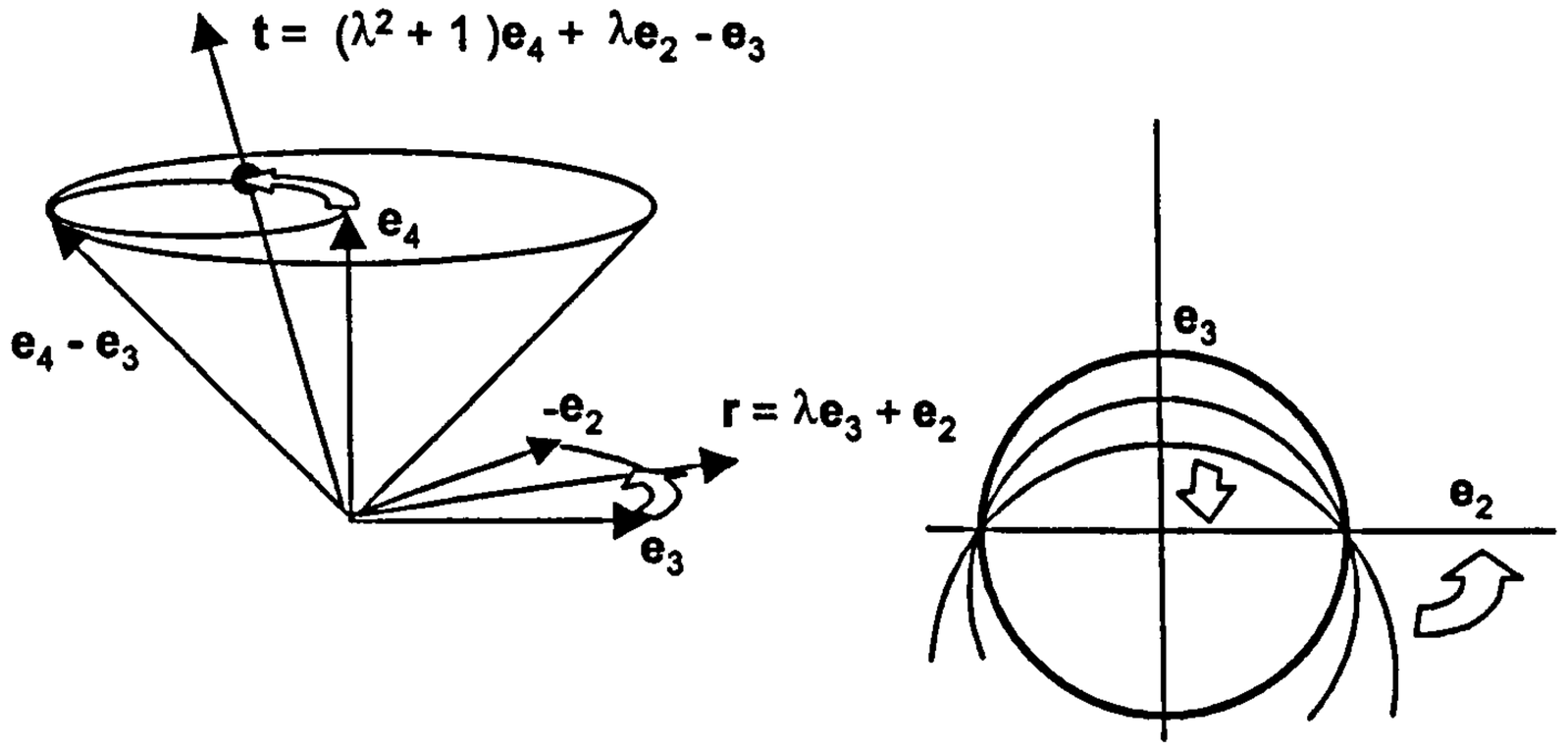


Figure 9.6 **Left:** Opposite rotations in conformal space of the representation of an equatorial circle of spherical space defined by r and the corresponding geometry-defining vector t .

Right: Dilation and flattening of the actual equatorial circle r due to the rotation of its representation in conformal space.

Figures 9.5 and 9.6 depict 4D conformal space with the e_1 dimension omitted. The t vector describes a cone tangent to the null cone.

9.8 Combining interfaces

The first interfaces for spherical and hyperbolic space could be combined, see figure 9.7.

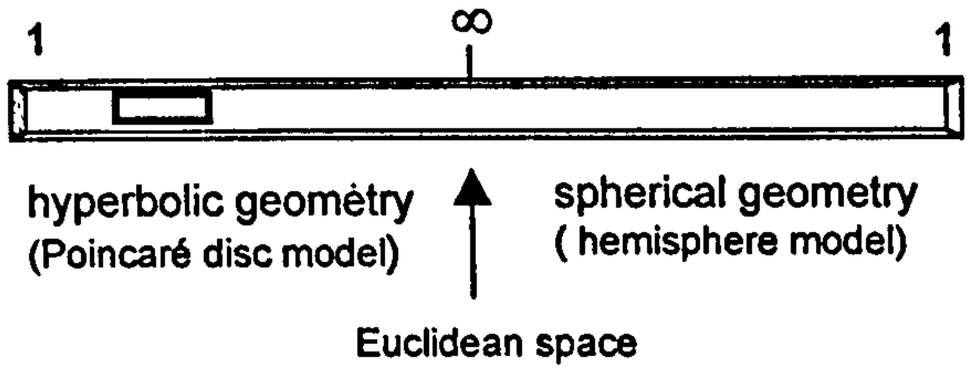


Figure 9.7 Slider to change from hyperbolic to spherical geometry.

The slider could start with a left-value of 1, corresponding to the Poincaré disc model with the horizon the unit circle. Moving it to the right would expand the horizon until it became infinite, which would correspond to Euclidean space. Continuing to slide it to the right would cause the circle at infinity to shrink, but this time it would represent the equatorial circle of spherical space.

In effect, the slider would rotate the conformal geometry-defining vector t from e_3 to e_4 using the bivector e_{34} as a generator. The rotation is hyperbolic, so the transition across the null cone would have to be handled carefully.

In a similar way, figure 9.8 shows a slider for transforming from the horizontal to vertical half-space models of hyperbolic space via the Poincaré disc model.

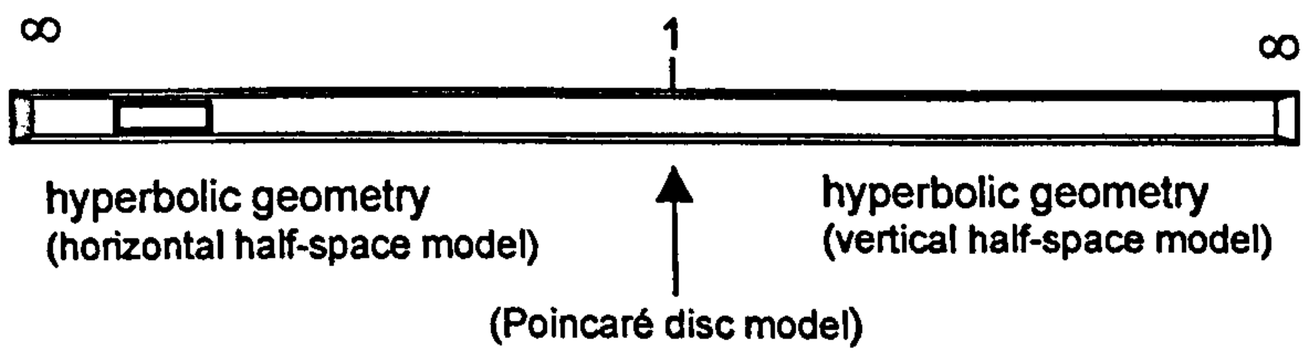


Figure 9.8 Slider to change from the horizontal to vertical half-space models of hyperbolic geometry.

In this case the geometry-defining vector would rotate from e_2 to e_3 along the plane e_{23} and then from e_3 to e_1 along the plane e_{13}.

9.9 Diameters in spherical and hyperbolic geometries

The pencil-based approach throws light on the nature of diameters in hyperbolic and spherical geometries. It also provides an answer to whether, if a circle becomes the horizon of a hyperbolic geometry or the equatorial circle of a spherical geometry, its diameters are geodesics of the 'local' geometry defined by the circle.

It is assumed that a diameter of a circle is a geodesic that meets the circle orthogonally. (In the spherical case this is easily verified by considering the circle as a stereographic projection of a circle in S^2.)

Figure 9.9 shows the hyperbolic diameters of a circle s defined by the horizon circle e_3. The diameters are members of the intersecting pencil dual to the Poncelet pencil defined by s and e_3. The Poncelet and intersecting pencils are orthogonal.

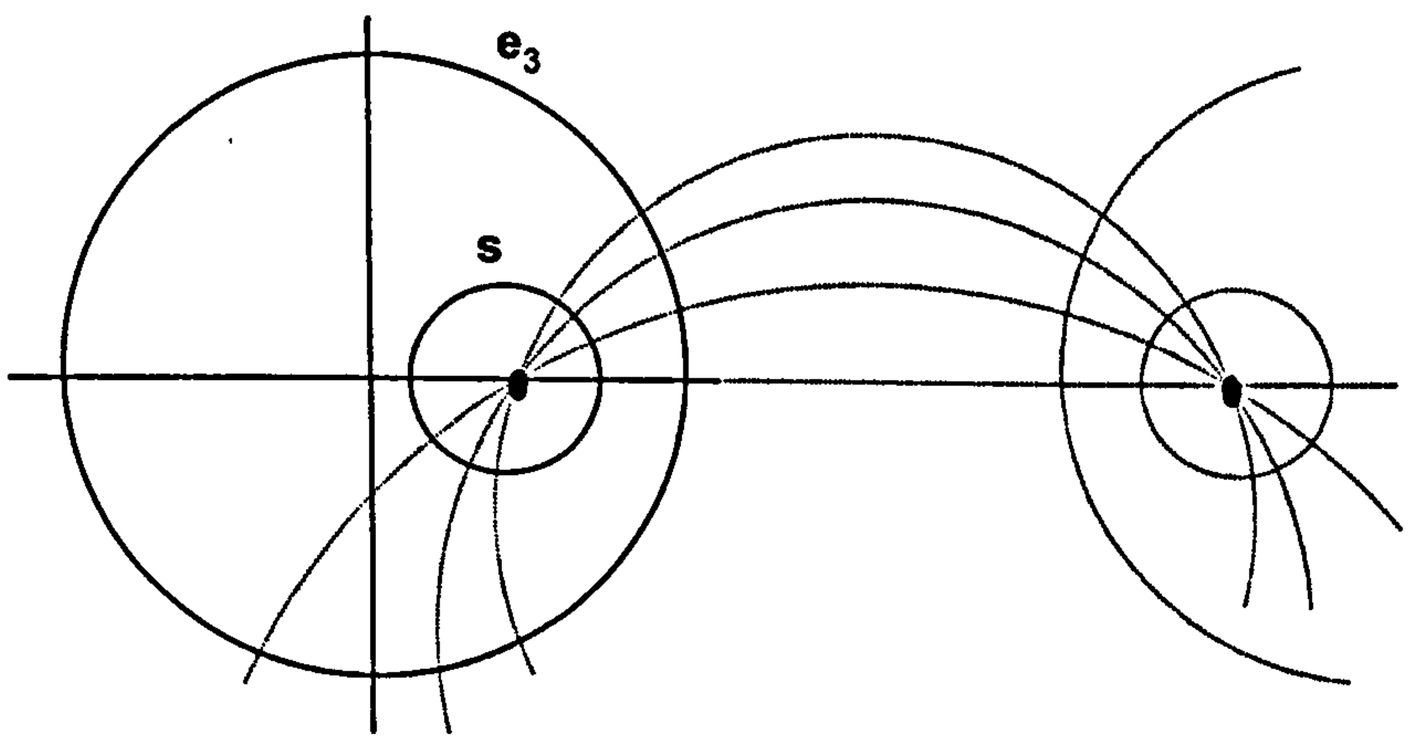


Figure 9.9 Intersecting pencil of hyperbolic diameters of a circle s.

Figure 9.10 shows the spherical diameters of the same circle s with e_3 taken as the equatorial circle.

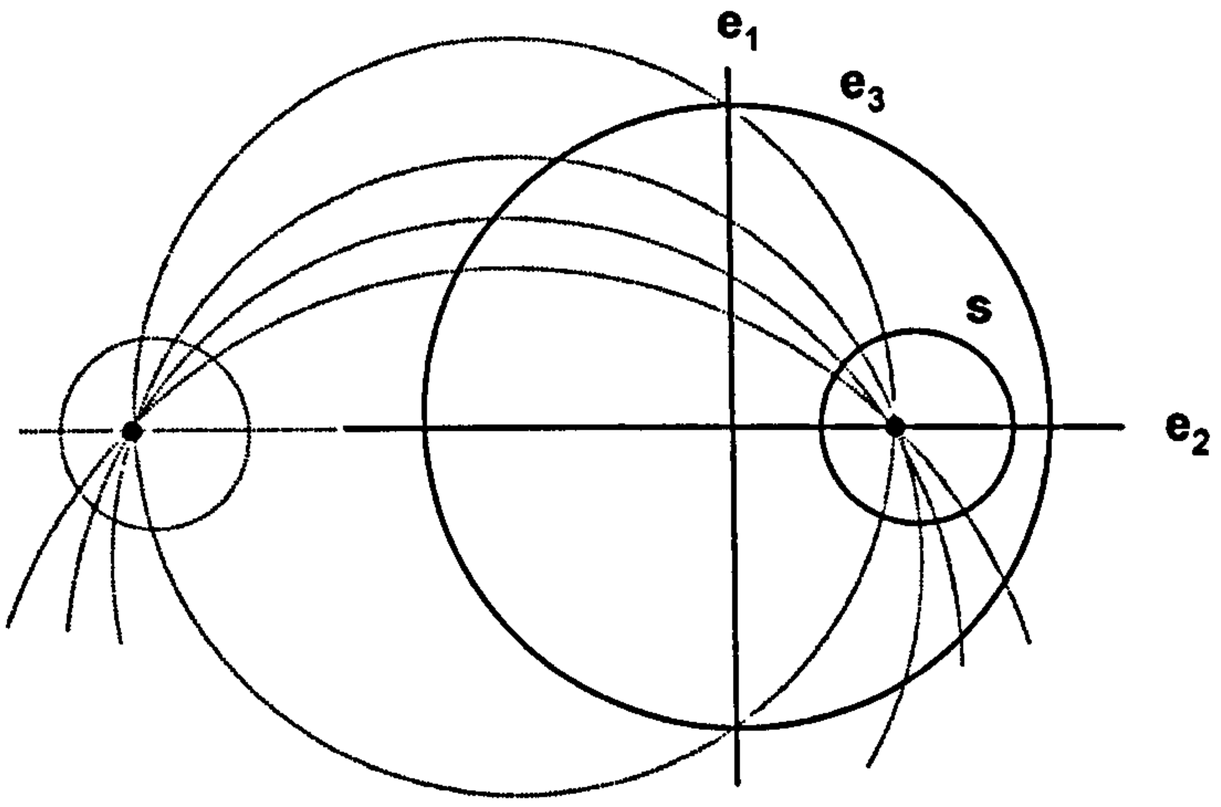


Figure 9.10 Intersecting pencil of spherical diameters of a circle s.

The diameters form a intersecting pencil defined by the horizontal diameter, i.e. the axis e_2, and the 'vertical' diameter. This latter diameter meets e_3 at antipodal points where e_3 meets the vertical axis e_1. The circle s is orthogonal to this pencil and is therefore a member of the dual Poncelet pencil.

The fact that any member of the intersecting pencil through the centre of s is a spherical diameter can be shown with reference to figure 9.11.

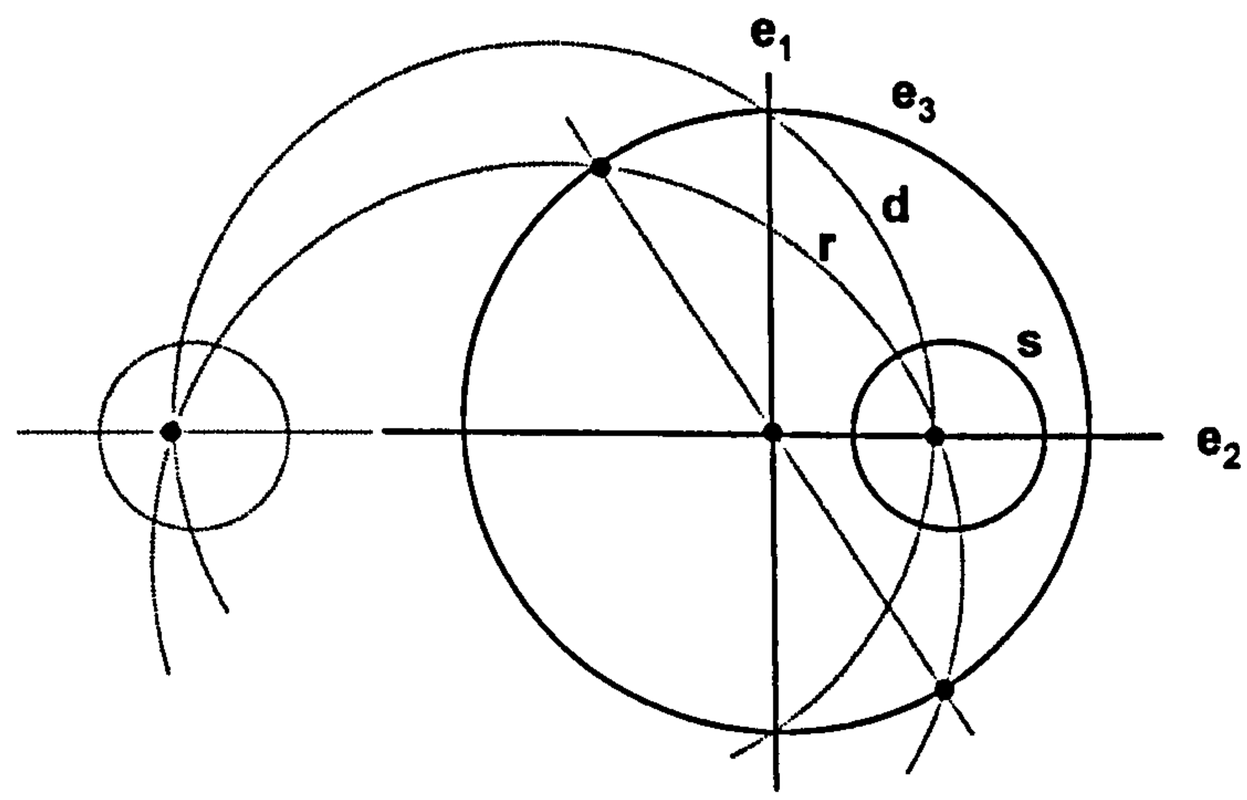


Figure 9.11 Spherical diameters r, d and e_2 of a circle s showing their antipodal points of intersection with the equatorial circle e_3.

Without loss of generality, assuming the centre of s is on the horizontal axis e_2, the 'vertical' diameter d is a member of the pencil $e_1 + \lambda e_3$. The horizontal diameter of s is the horizontal axis e_2. Any member of the intersecting pencil through the centre therefore has the form

$$r = d + \mu e_2,$$

i.e. $$r = e_1 + \lambda e_3 + \mu e_2.$$

Hence

$$r \cdot e_4 = 0.$$

This is equivalent to saying that r meets the equatorial circle e_3 at antipodal points. The equivalence comes from the fact that any straight line diameter through the centre of the equatorial circle is of the form $\lambda e_1 + \mu e_2$. By definition, such a diameter meets the circle e_3 at antipodal points. Any other circle that passes through the same two antipodal points has the form

$$c = \rho(\lambda e_1 + \mu e_2) + \sigma e_3.$$

It follows that

$$c \cdot e_4 = 0.$$

9.10 Families of diameters

For a hyperbolic space defined by a geometry-defining vector t, the family of diameters through a circle s is represented by the bivector

$$B = (t \wedge s)\tilde{}. \qquad (9.2)$$

This is an encoding of the fact that the diameters form a intersecting pencil dual to the Poncelet pencil formed from the horizon circle t and the circle s.

Formula 9.2 is evidently generic to other geometries as will now be shown.

If t = n, so that it defines Euclidean geometry, the same argument as above applies. In this case t^s = n^s defines a Poncelet pencil of *concentric* circles, and its dual is the family of *straight line* diameters.

Finally, suppose t is the geometry-defining vector of a spherical space, i.e. $t^2<0$. A circle d is a diameter of a circle s if it is a geodesic and it meets s orthogonally, i.e. if d·t = 0 and d·s = 0. We need to show that d is then contained in the linear subspace defined by the bivector (t^s)~. To do this we show that d is *not* in the complementary linear space defined by t^s. Suppose it was, then d can be written in the form

$$d = \lambda t + \mu s.$$

Hence $d\cdot s = \lambda t\cdot s + \mu s^2 = 0$

and $d\cdot t = \lambda t^2 + \mu s\cdot t = 0.$

Thus $\mu^2 s^2 - \lambda^2 t^2 = 0.$

This is not possible since $s^2 > 0$ and $t^2 < 0$. Hence d is in the subspace defined by (t^s)~.

9.11 Horizontal and vertical diameters - pencil based analysis

The following pencil-based analysis derives the vertical diameter of a circle s for a hyperbolic geometry with horizon circle t.

It is assumed that s is contained in t, so that they generate a Poncelet pencil. The axis a of this pencil is perpendicular to both s and t and, being a straight line, it follows that a·s = a·t = a·n = 0. Hence a is perpendicular to the linear subspace spanned by s, t and n, so a = (s^t^n)~

The vertical diameter d meets this axis at the same angle that the y-axis e_1 meets it, see figure 9.12.

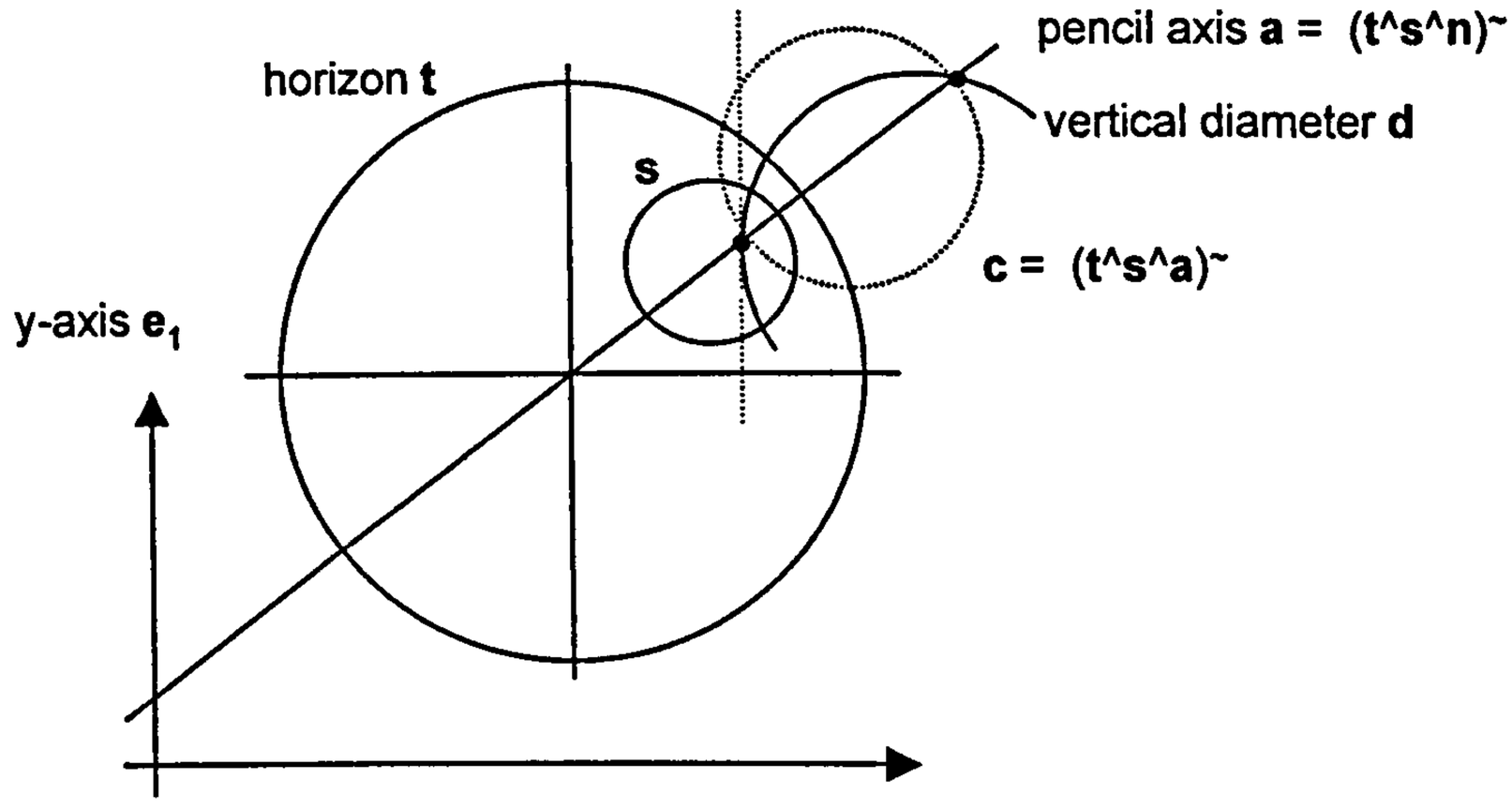


Figure 9.12 Vertical diameter d of a hyperbolic circle s.

Hence

$$\hat{d}\cdot\hat{a} = e_1\cdot\hat{a}\,. \tag{9.3}$$

On the other hand, d is a member of the intersecting pencil dual to the Poncelet pencil generated by t and s. One of the generators of this pencil is the axis a. Another generator is the circle c which has axis a as diameter, see figure 9.12. This circle is perpendicular to s, t and a, so it is given by

$$c = (t\wedge s\wedge a)^{\sim}\,.$$

Thus the required diameter is of the form $d = \lambda a + c$. Substituting in 9.3 above gives

$$\left(\sigma\left(\lambda\,\hat{a}+\hat{c}\,\right)\right)\cdot\hat{a} = e_1\cdot\hat{a}\,, \tag{9.4}$$

were ρ is a factor to normalise $\lambda a + c$ so that

$$\sigma^2\left(\lambda\,\hat{a}+\hat{c}\,\right)^2 = 1$$

i.e. $$\sigma^2\left(\lambda^2+1\right) = 1 \quad \text{since } a\cdot c = 0. \tag{9.5}$$

Expanding 9.4 and using the fact that a·c = 0 gives

$$\sigma\lambda = e_1 \cdot \hat{a}$$

so that

$$\sigma = \frac{(e_1 \cdot \hat{a})}{\lambda}.$$

Substituting in 9.5 yields

$$\lambda = \pm\sqrt{\frac{(e_1 \cdot \hat{a})^2}{(e_1 \cdot \hat{a})^2 - 1}}\ , \qquad (9.6)$$

where a = (s^t^n)˜ and c = (t^s^a)˜. (If it assumed that d ≠c, then λ ≠ 0.)

This approach was tested computationally in different geometries. The method was generically successful but limited because of the ambiguity of sign arising from 9.6. It was found that alternating between the plus and minus sign generated *two* circular diameters both of which met the axis a at the same angle that the axis met the y-axis, see figure 9.13.

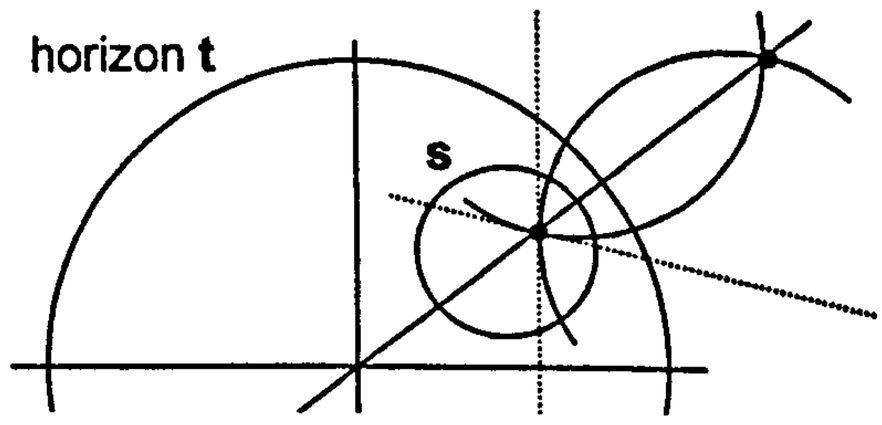


Figure 9.13 A hyperbolic circle s showing two diameters equally inclined to the line joining the centre of the circle to the centre of the horizon circle t.

9.12 Rotation about a point in any geometry

In the extreme case, the circle s in 9.2 can be replaced by a point p. Formula 9.2 then becomes

$$B = (t \wedge p)^{\sim} . \qquad (9.7)$$

This represents the pencil of geodesic lines through p and hence B generates a rotation about the point p in all geometries.

This agrees with the conclusion in chapter 5 that in the standard models of H^2, S^2 and R^2, a rotation about the origin is generated by the bivector e_{12}. To show this, suppose p is the origin $e_4 - e_3$, then for the Poincaré disc model, $t = e_3$ so that

$$B = (t \wedge p)^{\sim} = ((e_3) \wedge (e_4 - e_3))^{\sim} = e_{34}^{\sim} = -e_{12}.$$

For the hemisphere model of S^2, $t = e_4$ so that

$$B = (t \wedge p)^{\sim} = ((e_4) \wedge (e_4 - e_3))^{\sim} = e_{43}^{\sim} = e_{12}.$$

For R^2, $t = e_3 + e_4$ so that

$$B = (t \wedge p)^{\sim} = ((e_3 + e_4) \wedge (e_4 - e_3))^{\sim} = 2e_{34}^{\sim} = -2e_{12}.$$

Apart from a scalar multiple, which determines the sense and degree of rotation, the result is the same in each case.

This confirms that e_{12} generates rotations about the origin in all three basic geometries and is also an indication of why Euclidean geometry should perhaps be better described using n/2 rather then n.

Formula 9.7 could also provide an alternative navigation strategy to that suggested in chapter 5. Rather than retain the current cumulative transformation for each shape, it would be possible to retain its current position p, since it is relatively easy to rotate the shape about p using 9.7. However this approach would also require a way of defining, retaining and using a shape's current orientation.

On the other hand, if the transformation generated by B is applied to all scene elements, the scene will effectively rotate about p while p itself remains fixed. Thus 9.7 can also be viewed as generating a global rotational scene-transformation.

9.13 Multiple viewports with parallel geometries

Ideas emerging from this chapter suggest further possible interface designs.
One interesting possibility is for a scene, stored in conformal space, to be viewed through more than one viewport, each with a different geometry.
Transformation actions taken in one viewport ultimately effect the scene in conformal space and these changes are then projected into the remaining viewports. Alternatively, transformations could be applied to multiple viewports simultaneously.

As an example of the latter, a simple scene could be shown in two viewports, one with the spherical geometry of the hemisphere model and the other with the hyperbolic geometry of the Poincaré disc model. A cursor-key based interface might use the normal cursor keys to effect normal viewport translations, the shifted cursor keys to effect scene translation in the actual geometry using the translation bivectors derived in chapter 5, namely

$$T_x = e_1 e_{34} t$$
$$T_y = e_2 e_{34} t.$$

A third use of the cursor keys, with the control key held down say, could be to use the horizontal keys to rotate the scene (i.e. space) about an axis, and the vertical keys to change the orientation of that axis, all relative to the appropriate geometry. Figure 9.14 depicts the spherical case showing the changing points of concurrency for the intersecting pencil associated with the rotation.

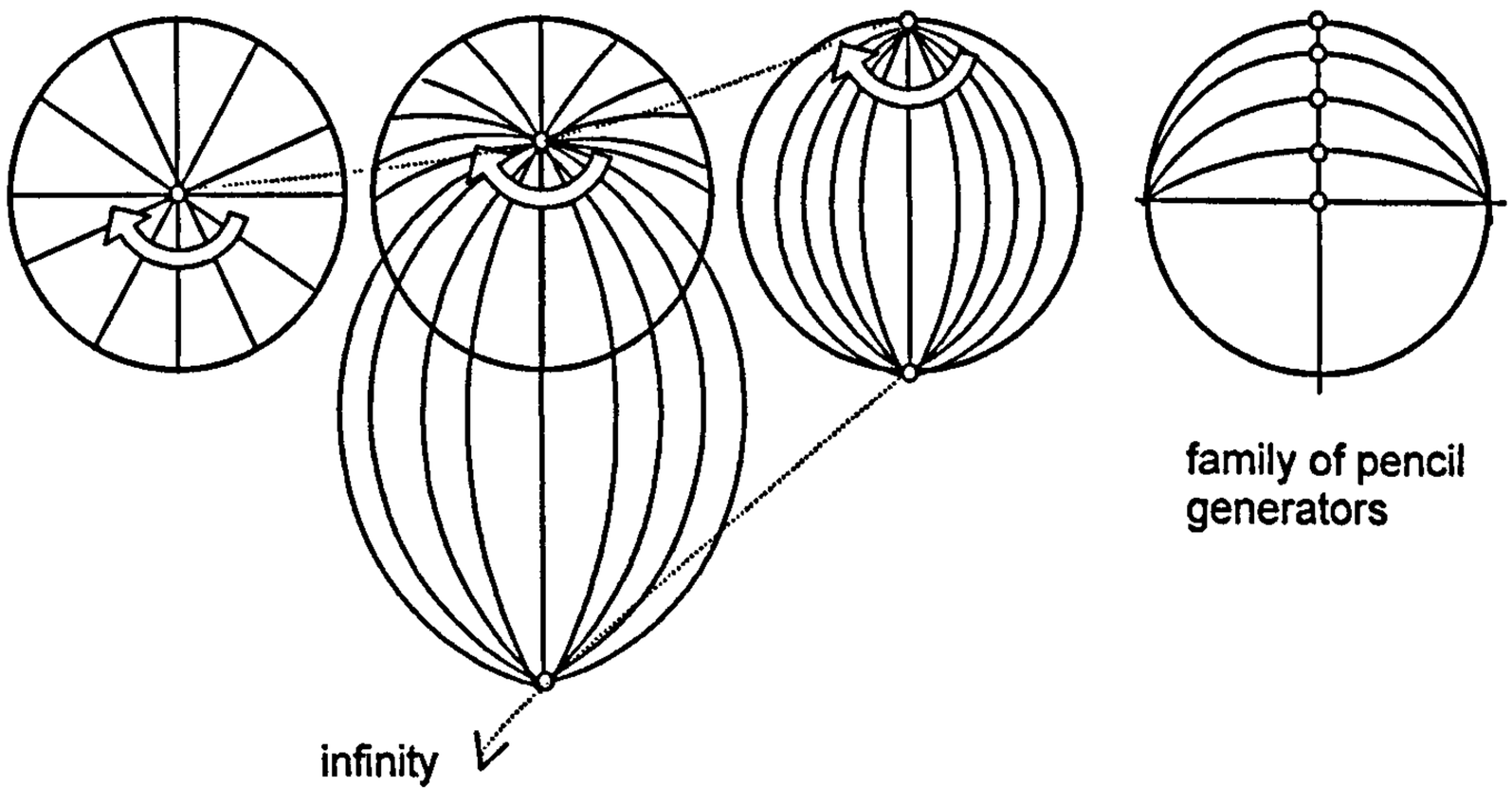


Figure 9.14 Sequence showing a point of rotation of spherical space migrating upward.

Representatives of this pencil are the vertical axis e_1 and a geodesic circle through the current point of rotation. The latter meets the unit circle at antipodal points, i.e. where it meets the horizontal axis e_2. These second generating circles, in turn, form the pencil $\lambda e_2 + \mu e_3$ (see figure 9.14). Thus the rotational bivector B is given by

$$B = (\lambda e_2 + \mu e_3)\wedge e_1.$$

Using the control-up and control-down cursor keys to simultaneously vary λ and μ between 1 and 0, and 0 and 1 respectively changes B so that the centre of rotation moves from the centre to the north pole, as shown in figure 9.14. The

control-left and control-right cursor keys can then be used to effect a rotation about the current centre of rotation using the current value of the bivector B.

The hyperbolic case is shown in figure 9.15.

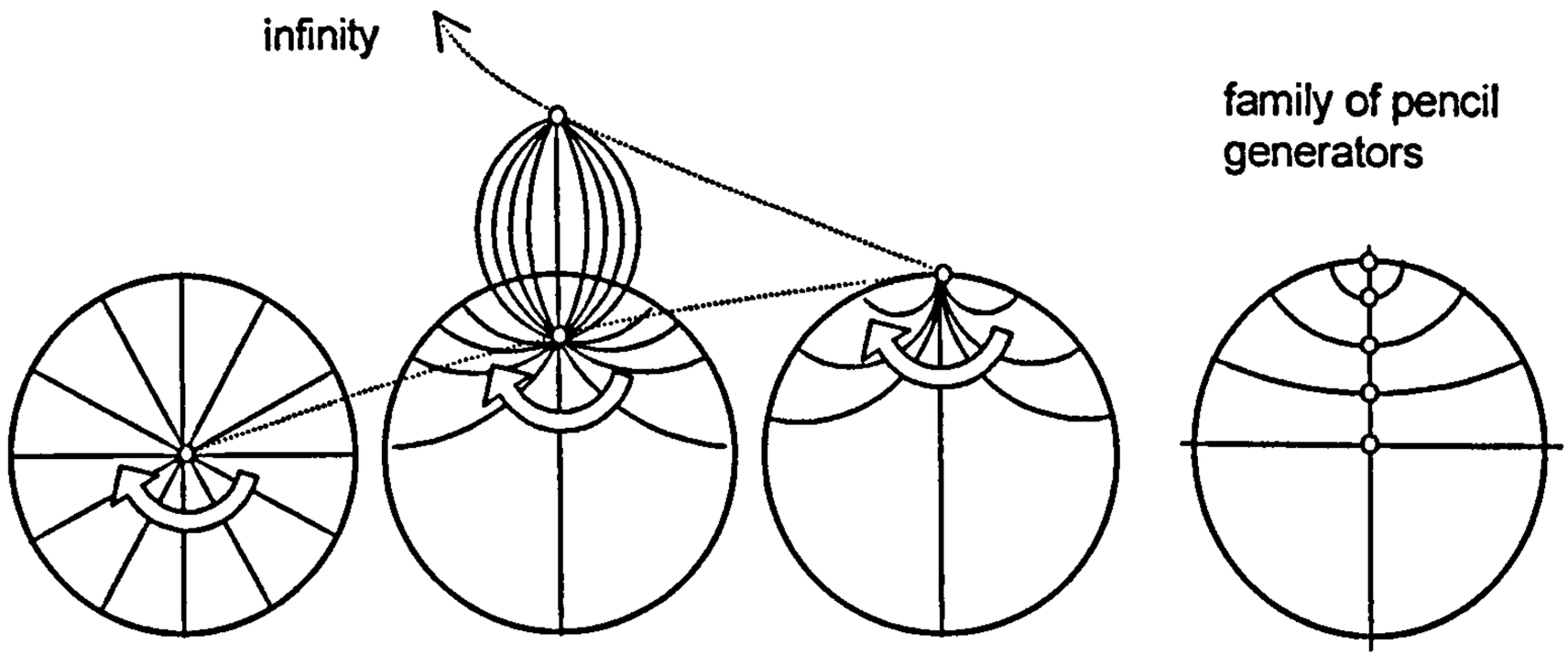


Figure 9.15 Sequence showing a point of rotation of hyperbolic space migrating upward..

As before, generators for the intersecting pencil are the vertical axis e_1 and the geodesic through the current centre of rotation. However, in this case these second generators form a Poncelet pencil with the north pole e_4+e_2 and the horizontal axis e_2 as generators (see figure 9.14). An arbitrary member of the pencil is given by $\lambda(e_4 +e_2) + \mu e_2$. The variable rotational bivector in this case is therefore

$$B = (\lambda(e_4 +e_2) + \mu e_2)\wedge e_1.$$

9.14 Conclusion

This chapter considered the relationship between the equatorial circle r of a spherical space and its geometry-defining vector t. The relationship seems complex and was only pursued in relation to two specific geometry-changing interfaces. To develop a more general relationship would probably require construction of the meet of the trivector of all geodesics ($t\tilde{}$) and the trivector of all straight lines generated by $\lambda e_1 + \mu e_2 + \nu n$. This would contain the straight-line geodesics which could be used to obtain the equatorial circle.

More significantly, the chapter further supports the idea that the emerging pencil-based approach can provide the graphics developer with an intuitive way of deriving and analysing powerful generic formulae such as 9.2. Such analyses then led to further suggestions for easy-to-implement prototype interfaces, an example of which is described in appendix A.

Conclusion

The main thrust of the research was to interpret and develop the conformal model in a way appropriate for a graphics developer interested in the design of interactive software for exploring non-Euclidean space.

There seem to be a number of fundamental conceptual barriers to getting to grips with the model. One relates to the fact that the non-Euclidean geometries of interest are embedded in a Minkowski space of more than 3 dimensions which is difficult to visualise. A second is that points and circles in the embedded non-Euclidean geometries are represented homogeneously by vectors in the Minkowski space. A third relates to the fact that transformations in the embedded geometries are generated by bivector generated transformations in the higher dimensional Minkowski space.

The techniques of visualisation and pencil-based analysis developed here seem to hold considerable promise for both aiding conceptual understanding and for future implementation research and development.

For obvious reasons, there seems reluctance in the literature to visually represent spaces of more than 3 dimensions, though pared down diagrams do sometimes hint at possibilities. Because of the inherent dangers, the visualisations developed in Chapter 1 took great care to identify the nature of the dimension reducing technique used so that the 4D Minkowski space could be represented as a 3D picture. The intuitive techniques made use of common metaphors - latitude lines, fans, joy sticks, and so on. The resulting visualisations were successful in showing how, by altering the orientation of a conformal vector in relation to the null cone, the circle it represented changed accordingly. The visualisation of this notion was key to the later work of chapters 6 to 9 where rotating conformal vectors generated pencils (families) of circles.

Of equal importance, the visualisation successfully reconciled two very different approaches to the conformal model: The classical approach based on the idea of stereographic projection, and one where the various geometries are seen as sections of the null cone. The orientation of a conic section is defined by a single geometry-defining-vector 'normal' to its plane. However, as pointed out in that chapter, in the visualisation of Minkowski space, 'normal' does not necessarily mean perpendicular.

In relation to the dimension reducing techniques introduced, it is perhaps worth pointing out that although they were largely intuitive they do seem to be particular cases of mathematical 'fibrations'. In other words, there is a formal theory relating to fibre bundles to underpin the intuitive, should that ever be required.

Central to the work was the need to make sense of the two-sided mapping in conformal space generated by a bivector B

$$X \rightarrow e^{B/2} X e^{-B/2}$$

The approach in chapter 5 was somewhat serendipitous - it utilised the fact that Cl(3,1) embeds the 2D sphere, allowing for comparisons with classical quarternions. Bivectors of Cl(3,1) that do not entail the fourth negative-signature component act on the 2D sphere like bivectors of Cl(3), i.e. like pure quarternions. This led to the symmetrical discovery that bivectors of Cl(3,1) that do not entail the third positive-signature component act on the embedded Minkowski sphere of negative unit radius is much the way that quarternions do. Using these ideas in conjunction with retained-mode graphics led to the derivation of navigation controls for 2D non-Euclidean geometries. These ideas were extended to 3D geometries.

The major challenge of this research was dealing with transformations generated by more generic bivectors. The question became more urgent when trying to solve the problem of how to implement on-screen mouse-induced rotations and dilations about a fixed point. The problem proved surprisingly intractable - apparent 'quick fixes' gleaned from the literature proved unreliable, with unaccountable errors creeping in. To track down the source of the errors it was necessary in Chapter 8 to carry out a new and thorough mathematical analysis where it was found that errors were due to the fact that some results casually referred to in the literature were in fact Taylor approximations. This illustrates a problem frequently faced by this research - often results are quoted without explanation or background as though being 'well known'. Perhaps they are, but probably only in the context in which the authors normally work which is usually advanced theoretical/mathematical physics.

The major breakthroughs in this research began when the role of the conformal bivector that generated a transformation was given a dual interpretation. In conformal space it defines a plane of rotation and 'on the ground', in the geometry of interest, it defines a pencil of circles. Though this dual interpretation is not new, it does not seem to have been exploited and developed in the way that is here. This is probably because the exploration was driven by a need to initially solve the in-scene mouse interaction problems. The new approach proved to be surprisingly powerful and was used to solve other problems, particularly that of how to implement viewport transformations and how to dynamically alter geometry.

Perhaps another reason why these ideas have not been developed previously is that, rather ironically, much of the research in the conformal model ignores the fact that it can model non-Euclidean geometry but focuses only on normal 3D space. The underlying agenda seems to be the promotion of the conformal model as an alternative to the standard homogeneous model. The case for this is weakened by the fact that the conformal model seems very good at dealing with circles and spheres, and their intersections, but not with other geometric

elements. When dealing with such intersections, circles are better represented by blades, rather than by vectors. Doing so introduces a conformity in approach since spheres are also represented by blades.

Thus, most implementation research on the conformal model tends to ignore the vector representation of circles. It is precisely here where this work breaks new ground.

References

1 Hongbo Li, David Hestenes and Alyn Rockwood. A Universal Model for Conformal Geometries of Euclidean, Spherical and Double-Hyperbolic Spaces, In: *Geometric Computing with Clifford Algebras, G. Sommer (Ed.) Springer, 2001.*

2 Hongbo Li, David Hestenes and Alyn Rockwood. Generalised Homogeneous Coordinates for Computational Geometry, In: *Geometric Computing with Clifford Algebras, G. Sommer (Ed.) Springer, 2001.*

3 Hongbo Li, David Hestenes and Alyn Rockwood. Spherical Conformal Geometry with Geometric Algebra. In: *Geometric Computing with Clifford Algebras, G. Sommer (Ed.) Springer, 2001.*

4 Chris Doran and Anthony Lasenby. *Geometric Algebra for Physicists* (chapter 10), Cambridge University Press, 2003.

5 Leo Dorst and Stephen Mann. Geometric Algebra: A Computational Framework for Geometrical Applications Parts 1 and 2. *IEEE Computer Graphics and Applications, May/June and July/August 2002.*

6 Daniel Fontijne and Leo Dorst. Modelling 3D Euclidean Geometry. *IEEE Computer Graphics and Applications*, March/April 2003.

7 Ekhard M.S. Hitzer. KAMIWAAI – Interactive 3D Sketching with Java based on Cl(4,1) Conformal Model of Euclidean Space. *Advances in Applied Clifford Algebras, Vol 13(1) June 2003.*
http://redquimica.pquim.unam.mx/clifford_algebras/

8 David Hestenes, *New Foundations of Classical Mechanics 2nd Edition. (Chapter 1 & 2), Kluwer Academic Press, 1999.*

9 Chris Doran and Anthony Lasenby. *Geometric Algebra for Physicists (chapter 1,2 & 4), Cambridge University Press, 2003.*

10 CLICAL: Complex Number, Vector Space and Clifford Algebra Calculator. Pertti Lounesto, Riste Mikkola, Vesa Vierros. Institute of Mathematics, Helsinki University of Technology, 1988.

11 General Clifford algebra and related differential geometry calculations with MATHEMATICA, Josep M Parra and Llorenç Roselló In: *Clifford Algebra with Numeric and Symbolic Computation. Eds: Rafal Ablamowicz, Pertti Lounesto and Josep M Parra, Birkhäuser, 1996.*

12 Clifford: A Maple V Package for Clifford Algebra Computations. Rafal Ablamowicz and Bertfried Fauser. Department of Mathematics, Tennessee Technological University, 2001.

13 Glyph: A Maple V Package for Clifford Algebra Computations. Rafal Ablamowicz and Bertfried Fauser. Department of Mathematics, Tennessee Technological University, 2001.

14 Gable: A Matlab Tutorial for Geometric Algebra Leo Dorst, Stephen Mann and Tim Bouma, University of Amsterdam, 2002.

15 Leo Dorst, Stephen Mann and Tim Bouma, The making of Gable: A geometric Algebra learning Environment in Matlab in: *Geometric Algebra with Applications in Science and Engineering, Eds: Eduardo Bayo Corrachano and Garret Sobczyk, Birkhäuser, 2001.*

16 Caigen: A Geometric Algebra Implementation Generator. Daniël Fontijne, Leo Dorst, University of Amsterdam, 2002.

17 Arvind Raja, Object Oriented Implementations of Clifford Algebras in C++: A prototype. In: *Clifford Algebra with Numeric and Symbolic Computation. Eds: Rafal Ablamowicz, Pertti Lounesto and Josep M Parra, Birkhäuser, 1996.*

18 David Hestenes , Hongbo Li and Alyn Rockwood. New Algebraic Tools for Classical Geometry, *In: Geometric Computing with Clifford Algebras, G. Sommer (Ed.) Springer, 2001.*

19 Pertti Lounesto *Clifford Algebra and Spinors 2nd Edition, Cambridge University Press, 2001.*

20 J.M.Selig, *Geometric Methods in Robotics*. Springer-Verlag, 1996

21 Marcel Riesz, *Clifford Numbers and Spinors Edited by E Folke Bolinder and Pertti Lounesto. Kluwer Academic Publishers, 1993.*

22 W.E. Bayliss and S Hadi, Rotations in *n* Dimensions as Spherical Vectors. In: *Applications of Geometric Algebra in Computer Science and Engineering, Birkhäuser, 2002. Eds: Leo Dorst, Chris Doran and Joan Lazenby.*

23 Chris Doran, *Circle and sphere blending with conformal geometric algebra.* Astrophysics Group, Cavendish Laboratory, Cambridge, 2003.

24 David Brannan, Matthew Esplen and Jeremy Gray, *Geometry. Cambridge University Press, 1999, reprinted 2004.*

Appendix A A short test program

This program implements a double viewport system. The cursor keys control the view in the mouse-selected viewport as follows:

Unmodified, the cursor keys effectively translate the viewport.

Modified with the SHIFT key, they induce non-Euclidean translations using the generic formula discussed in chapter 5, namely

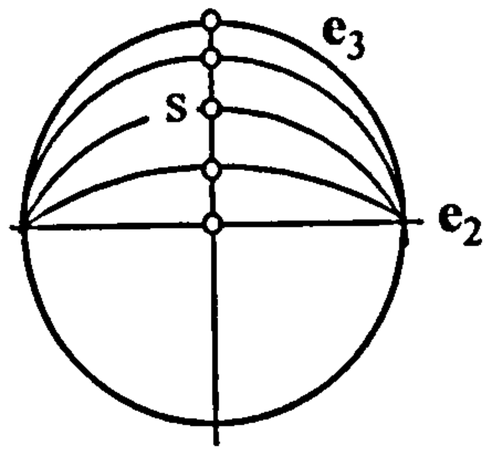
$$Tx = e_1\, e_{34}\, \mathbf{t}$$
$$Ty = e_2\, e_{34}\, \mathbf{t}$$

where t is the geometry-defining vector.

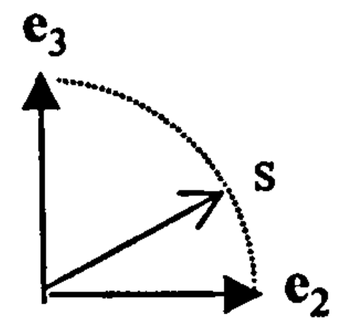
Modified with the CONTROL key, they induce rotations about a moveable point. The *left & right* cursor keys rotate the scene, the *up* and *down* keys move the centre of rotation up and down relative to the viewport.

The technique used is a slight extension to the approach discussed at the end of chapter 9 where the rotation bivector was the wedge product of the vertical axis e_1 and the current member s of the pencil $\lambda e_2 + \mu e_3$ or $\lambda(e_4 + e_2) + \mu e_2$, depending on whether the geometry was spherical or hyperbolic.

Rather than select the member of the pencil parametrically, it is possible to view each selection as a rotation of the e_2 vector in conformal space. In spherical space the rotation is from e_2 to e_3 and is Euclidean, see figure A.1.



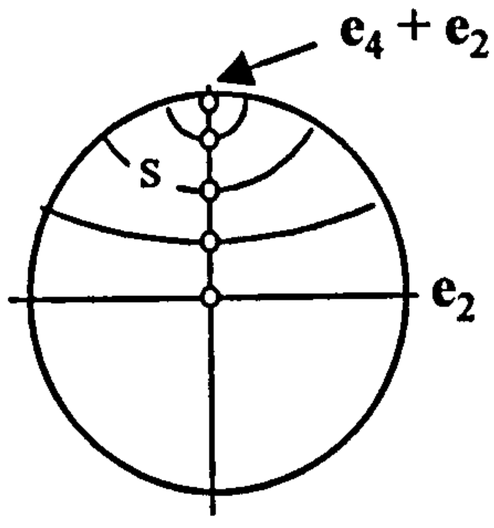
$s = \lambda e_2 + \mu e_3$



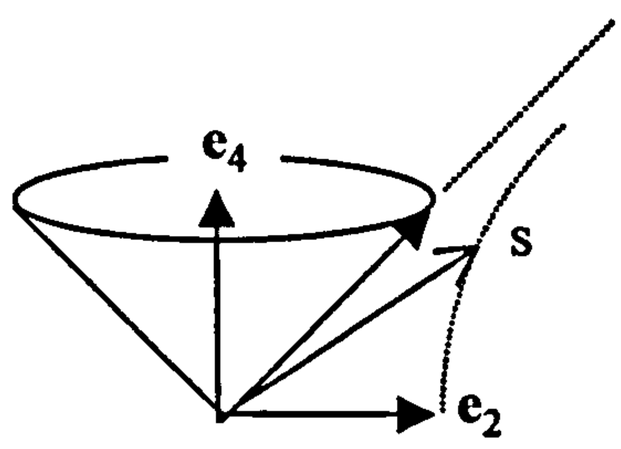
Euclidean rotation
in the e_{23} plane

Figure A.1 Rotating s from e_2 to e_3 on the e_{23} plane.
(The rotation is Euclidean.)

In the hyperbolic case, the rotation is hyperbolic from e_2 to e_4, though e_4 is never reached - the rotation is asymptotic to the null cone value $e_2 + e_4$, see figure A.2.



$s = \lambda(e_4 + e_2) + \mu e_2$



hyperbolic rotation
in the e_{24} plane

Figure A.2 Attempting to rotate s from e_2 to e_4 on the e_{24} Minkowski plane.
(The rotation is hyperbolic so s only rotates as far as $e_2 + e_4$.)

In the spherical case where the geometry-defining vector $t = e_4$, the rotation bivector for generating s is e_{23}. In the spherical case, when $t = e_3$, the bivector is e_{24}. In the spirit of generality, but only in these two cases, the bivector can be expressed as te_{234}.

The program displays three shape types:

circles, specified by their conformal blade representation,

points, specified by their position specified conformally and drawn as a 4-pixel wide square,

geodesics arcs, specified by two points conformally specified, and drawn as a *full* circles or as straight line joining the points as appropriate.

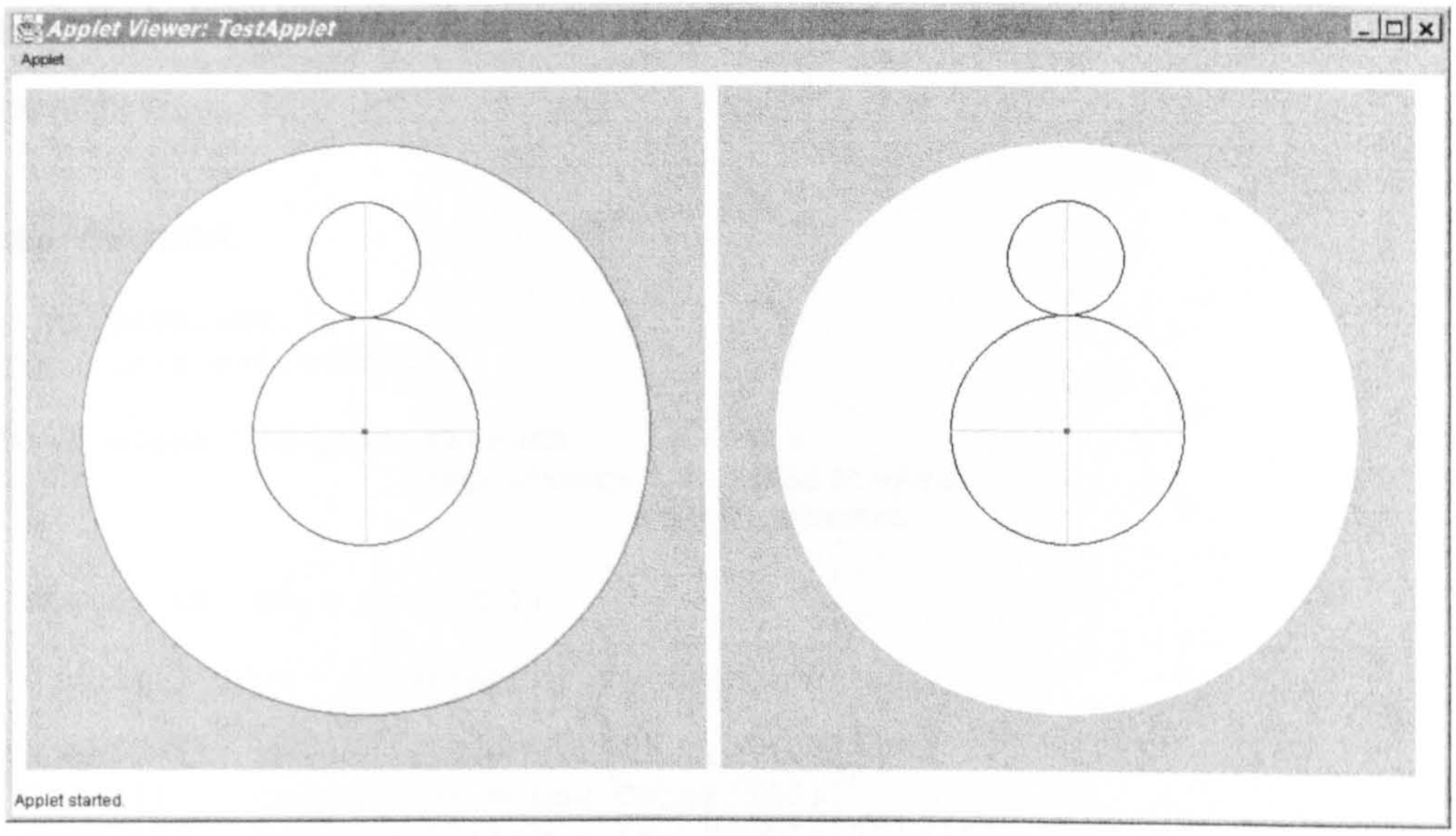
The first two shapes are fully specified by a single multivector, the third requires a second 'overflow' multivector. Conformal shape data is stored in the arrays `conformalArray` and `overflowArray`.

The drawing of each shape requires up to four values to specify its bounding box. This data are stored in the array `shapeArray` and held as decimal valued modeling co-ordinates, appropriately scaled for viewport coordinates on the fly using the current viewport parameters.

When a scene transformation takes place, all stored conformal multivector data are transformed identically then the data in the shape array are modified accordingly.

Rather than passing transformation bivectors as parameters, their exponentials are passed instead. These are referred to as 'versors' and often denoted by a V.

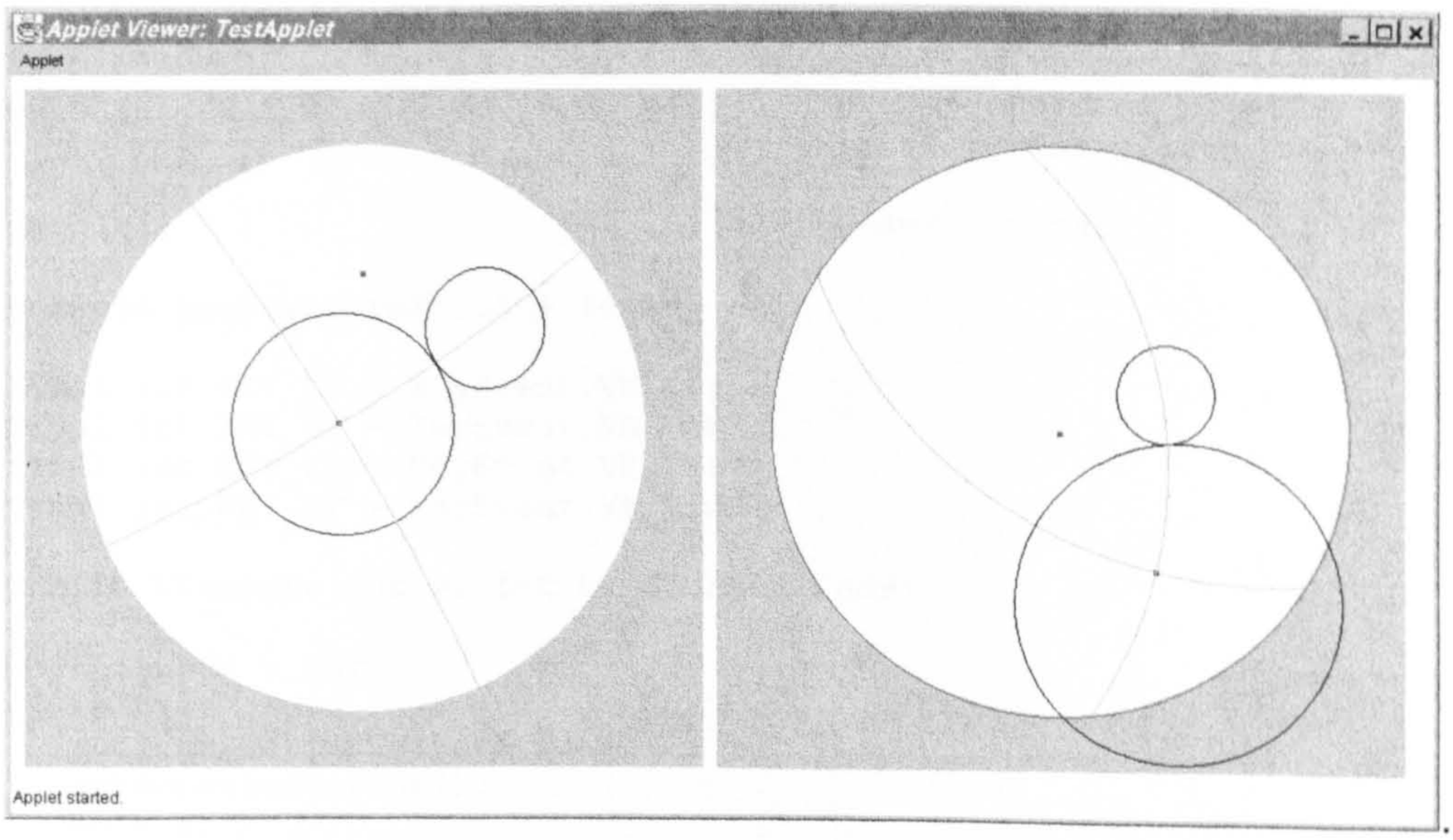
The program generates the original scenes depicted in figure A.3.



hyperbolic geometry spherical geometry

Figure A.3 Initial positions of objects.

After various transformations the scenes may appear as in figure A.4.



hyperbolic geometry spherical geometry

Figure A.4 Subsequent positions of objects.

The code is listed below. It makes use of the CM class listed in appendix C.

class Viewport

```
import java.awt.*;
import java.awt.event.*;

public class Viewport extends     Canvas
                      implements  FocusListener,
                                  KeyListener
{
   static CM  ga = new CM();

   double[] gdv;  // geometry defining vector

   double[][] shapeArray = new double[10][5];
   Color[]    colorArray = new Color[10];
   double[][] conformalArray = new double[10][16];
   double[][] overflowArray  = new double[10][16];
   int[]  conformalArrayType = new int[10];

   int n = 0;                          // current size of arrays

   int ww,hh;                          // viewport metrics
   int cx, cy, scale = 200;

   double k = Math.PI/20;

   double[] Vrt, Vlt, Vup, Vdn;     // translation versors
   double[] s  = ga.e2;             // rotation bivector/versor generator
   double[] Vr1;                       // rotation versor generated by s
   double[] cp = ga.sub(ga.e4,ga.e3); // shows centre of rotation
                                       // initialy the origin: e4 - e3

   double[] V ;                        // generic versor

   boolean hasKeyboardFocus = false;

   final int KEY_UP = KeyEvent.VK_UP;
   final int KEY_DN = KeyEvent.VK_DOWN;
   final int KEY_LT = KeyEvent.VK_LEFT;
   final int KEY_RT = KeyEvent.VK_RIGHT;

   public Viewport(int w, int h, double[] gdv)
   {
      this.gdv = gdv;
      init();
      setTranslationVersors();
      setRotationVersor();
      hh = h;
      ww = w;
      cx = ww/2;
      cy = hh/2;
      this.setSize(ww,hh);
      this.setBackground(Color.lightGray);
      this.addFocusListener(this);
      this.addKeyListener(this);
   }
```

```
void init()
{
   double[] p11 = ga.F(new double[] { 0, +.4, .0});
   double[] p21 = ga.F(new double[] { 0, -.4, .0});
   double[] p31 = ga.F(new double[] { 0, 0, .4});

   double[] B1 = ga.createBlade(new double[][] { p11, p21, p31 });
   insertCircleIntoArray(B1, Color.black);

   double[] p12 = ga.F(new double[] { 0, 0, .4});
   double[] p22 = ga.F(new double[] { 0, 0, .8});
   double[] p32 = ga.F(new double[] { 0, 0.2, .6});

   double[] B2 = ga.createBlade(new double[][] { p12, p22, p32 });
   insertCircleIntoArray(B2, Color.black);


   double[] q = ga.F(new double[] { 0, 0, 0});
   insertPointIntoArray(q, Color.blue);


   double[] p13 = ga.F(new double[] { 0, -0.4, 0});
   double[] p23 = ga.F(new double[] { 0, +0.4, 0});
   insertGeodesicArcIntoArray(p13, p23, Color.lightGray);

   double[] p14 = ga.F(new double[] { 0, 0, +0.8 });
   double[] p24 = ga.F(new double[] { 0, 0, -0.4 });
   insertGeodesicArcIntoArray(p14, p24, Color.lightGray);

}

void setTranslationVersors()
{
   Vrt = ga.exp(ga.mul(ga.gp(ga.e1, ga.e34, gdv),+k));
   Vlt = ga.rev(Vrt);
   Vup = ga.exp(ga.mul(ga.gp(ga.e2, ga.e34, gdv),+k));
   Vdn = ga.rev(Vup);
}

void setRotationVersor()
{
   Vr1 = ga.exp(ga.mul(ga.dp(gdv, ga.e234),k));
}

void insertCircleIntoArray(double[] blade, Color c)
{
   conformalArray[n] = blade;
   conformalArrayType[n] = 2;

   double[] centre = ga.f(ga.getCentreFromBlade(blade,ga.n));
   double   radius = ga.getRadiusFromBlade(blade,ga.n);

   shapeArray[n][0] = 1;
   shapeArray[n][1] = centre[1];
   shapeArray[n][2] = centre[2];
   shapeArray[n][3] = radius;
   shapeArray[n][4] = radius;
   colorArray[n] = c;
   n++;
}
```

```
void insertPointIntoArray(double[] vector, Color c)
{
   conformalArray[n] = vector;
   conformalArrayType[n] = 1;

   double[] position = ga.f(vector);

   shapeArray[n][0] = 0;
   shapeArray[n][1] = position[1];
   shapeArray[n][2] = position[2];
   colorArray[n] = c;
   n++;
}

void insertGeodesicArcIntoArray(double[] v1, double[] v2, Color c)
{
   conformalArray[n] = v1;
   overflowArray[n]  = v2;
   conformalArrayType[n] = 3;


   double[] B = ga.createBlade(new double[][] {v1,v2,gdv});
   double[] centre = ga.f(ga.getCentreFromBlade(B, ga.n));
   double   radius = ga.getRadiusFromBlade(B, ga.n);

   double[] p1 = ga.f(v1);
   double[] p2 = ga.f(v2);

   double X1 = p1[1];
   double Y1 = p1[2];
   double X2 = p2[1];
   double Y2 = p2[2];

   if (radius==-1)  // circle is a straight line
   {
      shapeArray[n][0] = 2;
      shapeArray[n][1] = X1;
      shapeArray[n][2] = Y1;
      shapeArray[n][3] = X2 - X1;
      shapeArray[n][4] = Y2 - Y1;
   }
   else
   {
      shapeArray[n][0] = 3;
      shapeArray[n][1] = centre[1];
      shapeArray[n][2] = centre[2];
      shapeArray[n][3] = radius;
      shapeArray[n][4] = radius;
   }
      colorArray[n] = c;
      n++;
   }
```

```
void transformConformalArrays(double[] V)
{
   double[] V_rev = ga.rev(V);
   double[] v;

   for (int i=0; i<n; i++)
   {
        v =  conformalArray[i];
       conformalArray[i] = ga.gp(V,v,V_rev);

       if (conformalArrayType[i]==3)
      {
         v = overflowArray[i];
         overflowArray[i] = ga.gp(V,v,V_rev);
      }

   }
}


void updateShapeArray()
{
   int m = n;
   n = 0;
   for (int i=0; i<m; i++)
   {
      double[] v = conformalArray[i];
       int type = conformalArrayType[i];
      Color c = colorArray[i];

       switch (type)
       {
          case 1 : insertPointIntoArray(v,c); break;
          case 2 : insertCircleIntoArray(v,c); break;
          case 3 : double[] v2 = overflowArray[i];
                insertGeodesicArcIntoArray(v,v2,c);
                break;
       }
   }
}


public void paint(Graphics g) { update(g);}
public void update(Graphics g)
{
      g.setColor(Color.lightGray);
      g.fillRect(0,0,ww,hh);
      g.setColor(Color.white);
      g.fillOval(cx-scale,cy-scale, 2*scale,2*scale);

for (int i=0; i<n; i++)
{
  g.setColor(colorArray[i]);

  int shapeType = (int) shapeArray[i][0];

  int X = (int) (cx + scale*shapeArray[i][1]);
  int Y = (int) (cy - scale*shapeArray[i][2]);
  int R = (int) (scale*shapeArray[i][3]);
  int S = (int) (scale*shapeArray[i][4]);
```

```
switch (shapeType)
{
   case 0 : g.fillRect(X-2,Y-2,4,4);
            break;
   case 1 : g.drawOval(X-R,Y-R,2*R,2*R);
            break;
   case 2 : g.drawLine(X,Y,X+R,Y-S);
            break;
   case 3 : g.drawOval(X-R,Y-R,2*R,2*R);
            break;
     }
   }


   int X =  cx + (int) (scale*ga.f(cp)[1]);
   int Y =  cy - (int) (scale*ga.f(cp)[2]);
   g.setColor(Color.red);
   g.fillRect(X-2,Y-2,4,4);

   if (hasKeyboardFocus) g.setColor(Color.red);
                    else g.setColor(Color.lightGray);

   g.drawOval(cx-scale-1, cy-scale-1,
              2*scale+2,2*scale+2);

}

// implement interface functionality
public void focusGained(FocusEvent e)
{
   hasKeyboardFocus = true;
   repaint();
}
public void focusLost(FocusEvent e)
{
   hasKeyboardFocus = false;
   repaint();
}

public void keyTyped(KeyEvent e)    {}
public void keyReleased(KeyEvent e){}
public void keyPressed(KeyEvent e)
{
   int k = e.getKeyCode();
   int m = e.getModifiers();
   if ((m & KeyEvent.SHIFT_MASK)!=0 )
   {
   switch (k)
   {
      case KEY_UP : V = Vup;  break;
      case KEY_DN : V = Vdn;  break;
      case KEY_LT : V = Vlt;  break;
      case KEY_RT : V = Vrt;  break;
          default : return;
   }
   transformConformalArrays(V);
   updateShapeArray();
   repaint();
}
```

```
else if ((m & KeyEvent.CTRL_MASK)!=0 )
{
   switch (k)
   {
      case KEY_UP : s  = ga.gp(Vr1, s ,ga.rev(Vr1));
                    cp = ga.gp(Vr1,cp,ga.rev(Vr1));
                     break;

      case KEY_DN : s  = ga.gp(ga.rev(Vr1), s ,Vr1);
                    cp = ga.gp(ga.rev(Vr1),cp,Vr1);
                    break;

         case KEY_LT : V = ga.exp(ga.mul(ga.wp(s, ga.e1),+k/10));
                    transformConformalArrays(V);
                    updateShapeArray();
                    break;

         case KEY_RT : V = ga.exp(ga.mul(ga.wp(s, ga.e1),-k/10));
                    transformConformalArrays(V);
                    updateShapeArray();
                    break;
             default : return;
   }
   repaint();
}
else
{
   switch (k)
   {
      case KEY_UP : cy = cy - 10; break;
      case KEY_DN : cy = cy + 10; break;
         case KEY_LT : cx = cx - 10; break;
         case KEY_RT : cx = cx + 10; break;
             default : return;
      }
      repaint();
   }
}
```

class TestApplet

```
import java.awt.*;
import java.applet.*;


public class TestApplet extends Applet
{
   CM ga = Viewport.ga;

public void init()
{
   Viewport v1 = new Viewport(480,480, ga.e3); // hyperbolic
   Viewport v2 = new Viewport(480,480, ga.e4); // spherical

   this.addViewport(v1, 10,  10);
   this.addViewport(v2, 500, 10);

   setLayout(null);
}
```

```
    void addViewport(Viewport v, int x, int y)
    {
        this.add(v);
        v.setLocation(x,y);
        v.setVisible(true);
        this.setVisible(true);
    }
}
```

Appendix B Source code listing of the GA class

This class provides functionality for a geometric algebra of any dimension and any signature, see chapter 3.

```
public class GA
{
int m;                                  // base-space dimension
int d;                                  // dimension of GA
byte[] signature = new byte[3];         // signature;
byte[][]     basis;
byte[]       grades;
String[]     names;
byte[]       vectorModulus;
byte[][][]   table;
boolean[]    signModifiedByNameChange;
boolean      evenSubAlgebra;
double[]     I;

double[]     i    = {1};

public GA() {}
public GA(int p) { this(p,0,0,'-');}
public GA(int p, char sign) { this(p,0,0, sign);}
public GA(int p, int q) { this(p,q,0,'-');}
public GA(int p, int q, char sign) { this(p,q,0, sign);}
public GA(int p, int q, int r) { this(p,q,r,'-');}
public GA(int p, int q, int r, char sign)
{
   if (sign== '+') evenSubAlgebra = true;

   signature[0] = (byte) p;
   signature[1] = (byte) q;
   signature[2] = (byte) r;
   m = p + q + r ;
   d = (int) Math.pow(2,m);
   basis  = new byte[d][m+1];
   grades = new byte[d];
   names  = new String[d];
   vectorModulus =  new byte[m+1];
   table = new byte[d][d][m+1];
   signModifiedByNameChange = new boolean[d];
   I = new double[d];
   I[d-1] = 1;

   fillVectorModulusArray();
   createBasisVectors();
   createNames();
   constructTable();
}

void fillVectorModulusArray()
{
   for (int i=1; i<=m;i++)
   {
      if (i <= signature[0]) vectorModulus[i] = + 1;
```

```
      else if (i <= signature[0]+signature[1]) vectorModulus[i] = - 1;
      else if (i <=m) vectorModulus[i] = 0;
   }
}

void createBasisVectors()
{
   for (int i=0; i<d; i++) basis[i][0] = +1;

   int count = 0;
   byte grade = 1;

while (grade<=m)
{
   byte n = grade;

   int[] k    = new int[n];
   int[] kMax = new int[n];

   for (int i=0; i<n; i++)   k[i] = n-i;
   for (int i=0; i<n; i++)   kMax[i] = m-i;

   count++;
   for (int i=0; i<n; i++) basis[count][k[i]] = 1;
   grades[count] = n;


   while (k[n-1]< kMax[n-1])
   {
      k[0]++;
int i = 0;
while (i<n)
{
  if (k[i]>kMax[i])
  {
     k[i+1]++;

   int j = i;
   while( j>=0)
   {
      k[j] = k[j+1]+1;
      j--;
   }
  }
  i++;
}

count++;
      for (int ii=0; ii<n; ii++) basis[count][k[ii]] = 1;
      grades[count] = n;
   }

   grade++;
}
}
```

```
void createNames()
{
   for (int i=0; i<d; i++)
      names[i] = convertToInternalName(basis[i]);

}

void changeName(int pos, String newName, boolean changeSign)
{
   names[pos] = " " + newName;
   signModifiedByNameChange[pos] = changeSign;
   if (changeSign) basis[pos][0] = -1;
   constructTable();
}

void constructTable()
{
   for (int i=0; i<d; i++)
      for (int j=0; j<d; j++)
         table[i][j] = multiply(i,j);
}


int findBasisVectorIndex(byte[] r)
{
   boolean matchFound = false;

   int j = 0;
   while (!matchFound)
   {
      matchFound = true;
      int i = 1;

      while(i<=m)
      {
           if (r[i] != basis[j][i]) matchFound = false;
         i++;
      }

      j++;
   }

   return j - 1;
}

String convertToTableName(byte[] r)
{
   if ( r[0]==0) return " 0";

   String s = null;
   int j = findBasisVectorIndex(r);
   s = names[j].substring(1);
   if (r[0]==+1 )  s = " " + s;
   if (r[0]==-1 )  s = "-" + s;
   return s;
}
```

```
String convertToInternalName(byte[] r)
{
   String s = "e";

   boolean allZeros = true;
   boolean allOnes  = true;

   for (int i=1; i< r.length; i++)
   {
      if (r[i]==0) allOnes  = false;
      if (r[i]==1) allZeros = false;
   }

   if ( r[0]==0)     return " 0";
   if (allOnes )    return " I";
   if (allZeros)    return " 1";

   for (int i=1; i<=m; i++) if (r[i]==1) s += i;

   if (r[0]==+1)  s = " " + s;
   if (r[0]==-1)  s = "-" + s;

   return s;
}

// table multiplication..

public byte[] multiply(int a, int b) // table multiplication
{
   byte[] p =    basis[a];
   byte[] q =    basis[b];
   byte[] r =    new byte[m+1];
   byte sign = +1;

   for (int k=1; k<=m; k++)
   {
      if (p[k]==1 && q[k]==1) sign = (byte) (sign*vectorModulus[k]);

      if    (q[k]==0) r[k] = p[k];
      else
      {
         r[k] = (byte) (1 -  p[k]);

         if (sign != 0)
            for (int j=k+1; j<=m; j++)
               if (p[j]==1) sign = (byte) (-sign);
      }
   }

   if (p[0]==-1 && q[0]==+1)  sign = (byte) -sign;
   if (p[0]==+1 && q[0]==-1)  sign = (byte) -sign;

   r[0] = sign;
   return r;
}
```

```
// linear-algebra related functions

double[] complete(double[] m1)  // completes a truncated multi-vector
{
   double[] r = new double[d];
   for (int i=0;i<m1.length;i++) r[i] = m1[i];
   for (int i=m1.length;i<d;i++) r[i] = 0;
   return r;
}

public double[] add(double[] m1, double[] m2)        // multivector add
{
   double[] r = new double[d];
   double[] mm1 = complete(m1);
   double[] mm2 = complete(m2);

   for (int i=0; i<d; i++) r[i] = mm1[i] + mm2[i];
   return r;

}

public double[] add(double[] m1, double[] m2, double[] m3)
{
   double[] r = new double[d];
   double[] mm1 = complete(m1);
   double[] mm2 = complete(m2);
   double[] mm3 = complete(m3);

   for (int i=0; i<d; i++) r[i] = mm1[i] + mm2[i] + mm3[i];
   return r;
}

public double[] sub(double[] m1, double[] m2)    // multivector
{                                                // subtract
   double[] r = new double[d];
   double[] mm1 = complete(m1);
   double[] mm2 = complete(m2);

   for (int i=0; i<d; i++) r[i] = mm1[i] - mm2[i];
   return r;

}

public double mod(double[] m1)
{
   return Math.sqrt(dp(m1,m1)[0]);
}

public double[] mul(double[] m1, double k)           // scalar multiply
{
   double[] r = new double[d];
   double[] mm1 = complete(m1);

   for (int i=0; i<d; i++) r[i] = mm1[i]*k;
   return r;

}
```

```
public double[] div(double[] m1, double k)        // scalar divide
{
   double[] r = new double[d];
   double[] mm1 = complete(m1);

   for (int i=0; i<d; i++) r[i] = mm1[i]/k;
   return r;
}


// geometric algebra related products

public double[] gp(double[] m1, double[] m2)       // geometric product
{
   double[] r = new double[d];
   byte[] p;

   int d1 = Math.min(d, m1.length);
   int d2 = Math.min(d, m2.length);

   for (int i=0; i<d1; i++)
      for (int j=0; j<d2; j++)
         if (m1[i]!=0 && m2[j]!=0)
         {
            p = multiply(i,j);
            int k = findBasisVectorIndex(p);
            if ( p[0]==+1)  r[k] = r[k] + m1[i]*m2[j];
            if ( p[0]==-1)  r[k] = r[k] - m1[i]*m2[j];
         }
   return r;
}

public double[] gp(double[] m1, double[] m2, double[] m3) // geometric
{                                                         // product
    return gp(gp(m1,m2),m3);
}

public double[] rev(double[] m)  //reverse
{
   int sign;
   double[] mm = complete(m);
   for (int i=0;i<d;i++)
   {
      if ((grades[i]/2)%2==0) sign = +1;
                         else sign = -1;
      mm[i] = sign*mm[i];
   }
   return mm;
}

public double[] copy(double[] m)
{
    double[] r = new double[m.length];
   for (int i=0; i<m.length; i++) r[i] = m[i];
   return r;
}
```

```
public double[] gradePart(double[] m1, int g)
{

   double[] r = new double[d];
   if (g<0 || g>m ) return r;
   int dd = Math.min(d, m1.length);
   for (int i=0; i<dd; i++)
      if (grades[i]==g)
         r[i] = m1[i];
   return r;
}

public int grade(double[] mv)
{
  if (!isHomogeneous(mv)) return -1;

  for (int i=0; i<=m; i++)
     if (!isZero(gradePart(mv,i)))
      return i;

  return 0;
}

public boolean isHomogeneous(double[] mm)
{
   double[] m = complete(mm);
   boolean nonZeroElementFound = false;
   int grade = 0;

   for (int i=0; i<d; i++)
   {
      if (m[i]!=0 && !nonZeroElementFound)
      {
         nonZeroElementFound = true;
         grade = grades[i];
      }

      if (m[i]!=0 && grades[i]!=grade) return false;
   }

   return true;
}

public double[] wpH(double[] m1, double[] m2) // wedge product
{                                             // (homogeneous case)
   if ( !isHomogeneous(m1)) return null;
   if ( !isHomogeneous(m2)) return null;

   return gradePart(gp(m1,m2), grade(m1) + grade(m2));
}

public double[] dpH(double[] m1, double[] m2) // dot product
{                                             // (homogeneous case)
   if ( !isHomogeneous(m1)) return null;
   if ( !isHomogeneous(m2)) return null;

   return gradePart(gp(m1,m2), Math.abs(grade(m1) - grade(m2)));
}
```

```
public double[] wp(double[] m1, double[] m2) // wedge product
{
   double[] r = new double[d];

   for (int i=0;i<=m;i++)
      for (int j=0;j<=m;j++)
            r = add(r, wpH(gradePart(m1,i),gradePart(m2,j)));

   return r;
}
```

```
public double[] dp(double[] m1, double[] m2) // dot product
{
   double[] r = new double[d];

   for (int i=0;i<=m;i++)
      for (int j=0;j<=m;j++)
            r = add(r, dpH(gradePart(m1,i),gradePart(m2,j)));

   return r;
}

// blade creation..

public boolean isVector(double[] mv)
{
   if (!isHomogeneous(mv)) return false;
   if (grade(mv)!=1) return false;
   return true;
};
```

```
public double[] forceVector(double[] v)
{
   v[0] = 0;
   for (int i=5; i<16; i++) v[i] = 0;
   return v;
}

public double[] createBlade(double[][] v)
{

   double[] r = v[0];

   for (int i=1; i<v.length; i++)
      r = wp(v[i],r);

   return r;
}
```

```
public double[] createBlade2(double[][] v)
{
   for (int i=0; i<v.length; i++)
      if (!isVector(v[i])) return null;

   double[] Ar = v[0];
   int sign = -1;
   for (int i=1; i<v.length; i++)
   {
      if (sign==+1) Ar = mul(add(gp(v[i],Ar), gp(Ar,v[i])),1/2.0);
      if (sign==-1) Ar = mul(sub(gp(v[i],Ar), gp(Ar,v[i])),1/2.0);
```

```
      sign = -sign;
   }

   return Ar;
}

public double[] exp(double[] b)  // use only for simple bi-vectors
{
   double b_squared = gp(b,b)[0];

   if (b_squared>0)  // hyperbolic case
   {
      double k = Math.sqrt(b_squared);
      double[]  bHat  = div(b,k);

      double A = Math.exp(k);
      double B = 1/A;
      double coshk = (A + B)/2;
      double sinhk = (A - B)/2;

      return add(mul(i,coshk), mul(bHat, sinhk));
   }

   else if (b_squared<0)  // elliptic/spherical case
   {
      double k  = Math.sqrt(-b_squared);
      double[]  bHat  = div(b,k);

      return add(mul(i, Math.cos(k)), mul(bHat, Math.sin(k)));
   }

   else return add(i,b);  // Euclidean case
   }

   boolean isZero(double[] m)
   {
      boolean b = true;
      for (int i=0; i<m.length; i++)
         if (m[i]!=0) b = false;
      return b;
   }

// display functions

   public void printTable()
   {
      String s;
      byte[] r;
      int k = 1;
      int m = 1;

for (int i=0; i<d; i = i + m)
{
   if (!evenSubAlgebra || ( evenSubAlgebra && grades[i]%2==0))
   {
      for (int j=0; j<d; j = j + m)
      {
         if (!evenSubAlgebra ||
              ( evenSubAlgebra && grades[j]%2==0))
         {
            r = multiply(i,j);
```

```
                s = convertToTableName(r);
                k = findBasisVectorIndex(r);
                if (signModifiedByNameChange[k] && r[0]==-1)
                        s = " " + s.substring(1);
                if (signModifiedByNameChange[k] && r[0]==+1)
                        s = "-" + s.substring(1);

                System.out.print(s + "\t\t");
            }
        }

        System.out.print("\n\r");
    }
}
```

```
public void show(double[] m, boolean showArray)    // show multivector
{
    int j = Math.min(d,m.length);
    boolean allTermsZero = true;

    for (int i=0; i<j; i++)
    {
        if (m[i]!=0)
        {
            allTermsZero = false;
            if (m[i]>0) System.out.print(" + ");
                  else  System.out.print(" - ");
            System.out.print(Math.abs(m[i]));
            if (i!=0) System.out.print(names[i]);
        }

    if (allTermsZero) System.out.print("0");

    if (showArray)
    {
        System.out.print(" : { ");
        for (int i=0; i<j; i++)   System.out.print(m[i] + ", ");
        System.out.print("}");
    }

    System.out.println();
}

public void show(String label, double[] m )    // show multivector
{
  System.out.print(label);
  show(m, true);
}

public void show(String label, double[] m, boolean b )    // show
{                                                         // multivector
  System.out.print(label);
  show(m, b);
}

public void show(double[] m){ show(m, true); }  // show multivector
```

```
public void show(int n) { System.out.println(n);}

public void show(double d) { System.out.println(d);}

public void show(String label, int n)
{
    System.out.print(label);
    show(n);
}

public void show(String label, double d)
{
     System.out.print(label); show(d);
}
```

Appendix C Source code listing of the CM class

This class extends the GA class of Appendix A. It provides functionality for the Clifford algebra Cl(3,1) of the Conformal Model and also provides simple drawing functionality, see chapter 4.

```
import java.awt.*;
import java.awt.geom.*;

public class CM extends GA
{
   double[] origin = {0,    0, 0, 0, 0};

   double[]    a    = {0,    0, 0, 1, 0};
   double[]    e    = {0,    0, 0, 0, 1};
   double[]    n    = {0,    0, 0, 1, 1};
   double[]    n2   = {0,    0, 0, 0.5, 0.5};
   double[]   nBar  = {0,    0, 0, 1,-1};
   double[]    b    = {0,    0, 1, 0, 0};
   double[]    c    = {0,    1, 0, 0, 0};

   double[]    e1   = {0,    1, 0, 0, 0};
   double[]    e2   = {0,    0, 1, 0, 0};
   double[]    e3   = {0,    0, 0, 1, 0};
   double[]    e4   = {0,    0, 0, 0, 1};

   double[]    e12  = {0,    0, 0, 0, 0,  1, 0, 0, 0, 0, 0};
   double[]    e13  = {0,    0, 0, 0, 0,  0, 1, 0, 0, 0, 0};
   double[]    e14  = {0,    0, 0, 0, 0,  0, 0, 1, 0, 0, 0};
   double[]    e23  = {0,    0, 0, 0, 0,  0, 0, 0, 1, 0, 0};
   double[]    e24  = {0,    0, 0, 0, 0,  0, 0, 0, 0, 1, 0};
   double[]    e34  = {0,    0, 0, 0, 0,  0, 0, 0, 0, 0, 1};

   double[]   e234  = {0,    0, 0, 0, 0,  0, 0, 0, 0, 0, 0,  0, 0, 0, 1};

   double[]  I  = {0,    0, 0, 0, 0,  0, 0, 0, 0, 0, 0,  0, 0, 0, 0,  1};

   int scale = 200;
   int W = 800;     // viewport dimensions
   int H = 600;
   int Ox = W/2;    // screen co-ordinates of centre of viewport
   int Oy = H/2;

    public CM() { super(3,1);}
    public CM(int Ox, int Oy, int scale)
    {
      super(3,1);
      this.Ox = Ox; this.Oy = Oy;
      this.scale = scale;
    }
```

```
// funtion for mapping screen point into conformal space

 public double[] F(double[] p)
 {
     double[] t1 = mul(a, dp(p,p)[0]-1);
     double[] t2 = mul(p, 2);
     double[] t3 = mul(e, dp(p,p)[0]+1);

     return add(add(t1,t2),t3);

   //       return  sub(add(mul(n,dp(p,p)[0]), mul(p,2)),nBar);
   //    (alternate design)
}
```

```
// funtion for  mapping  conformal point to the screen
// (point undergoes null cone scaling before being mapped)

 public double[] f(double[] X)
 {
    double[] d = new double[16];
    for (int k=0; k<16; k++) d[k] = scale(X, n)[k];
    d[3] = 0;
    d[4] = 0;

    return d;
 }
```

```
// function for null cone scaling

 public double[] scale(double[] P, double[] type)
 {
    return div(P, -dp(P,type)[0]);
 }
```

```
// function for calculating the dual

 public double[] dual(double[] mv)   { return gp(I,mv); }
```

```
// construction non-Euclidean circle vector representation
// from conformal centre C & radius R

public double[] makeCircle(double[] C, double R, double[] type)
{
   return sub(C, mul(type, R*R));
}
```

```
//  functions for extracting  non-Euclidean circle properties
//  from blade representation L

public double[] getCentreFromBlade(double[] L, double[] type)
{
   double[] C = gp(gp(L,type),L);
   return C;
}
```

```
public double getRadiusFromBlade(double[] L, double[] type)
{
    double[] q = wp(L,type);
    if (dp(q,q)[0]==0) return -1;

   double r = Math.sqrt(-dp(L,L)[0]/dp(q,q)[0]);
   return r;
}

//  functions for extracting  non-Euclidean circle properties
//  from vector representation S

public double[] getCentreFromVector(double[] S, double[] type)
{
   double R_squared  = getRadiusSquaredFromVector(S,type);
   return add(S, mul(type, R_squared));
}

public double getRadiusSquaredFromVector(double[] S, double[] type)
{
   return dp(S,S)[0]/(dp(S,type)[0]*dp(S,type)[0]);
}



public double getRadiusFromVector(double[] S, double[] type)
{
   return Math.sqrt(dp(S,S)[0]/(dp(S,type)[0]*dp(S,type)[0]));
}


// drawing functions
// (viewport parameters cx, cy & scale passed as parameters)
// (does not use java.geom package, uses only java.awt)

public void drawCircle(Graphics g, int cx, int cy, int scale,
                       double[] blade, Color c)
{
   double[] centre = f(getCentreFromBlade(blade,n));
   double   radius =   getRadiusFromBlade(blade,n);

   double x = centre[1];
   double y = centre[2];

   int R = (int) (radius*scale);
   int X = (int) (cx + scale*x)  - R;
   int Y = (int) (cy - scale*y)  - R;

   g.setColor(c);
   g.drawOval(X,Y,2*R, 2*R);
}


public void drawPoint(Graphics g,  int cx, int cy, int scale,
                      double[] p, Color c)
{
   double [] P = f(p);

   double x = P[1];
   double y = P[2];
```

```
   int X = (int) (cx + scale*x);
   int Y = (int) (cy - scale*y);

   g.setColor(c);
   g.fillRect(X-2,Y-2,4,4);
}


public void drawArc(Graphics g, int cx, int cy, int scale,
                    double[] p1, double[] p2, double[] type, Color c)
{
   g.setColor(c);

   double[] B = createBlade(new double[][]{p1,p2,type});

   double[] centre = f(getCentreFromBlade(B, n));
   double   radius = getRadiusFromBlade(B, n);

   int X1 = (int) (cx + scale*f(p1)[1]);
   int Y1 = (int) (cy - scale*f(p1)[2]);
   int X2 = (int) (cx + scale*f(p2)[1]);
   int Y2 = (int) (cy - scale*f(p2)[2]);


   if ( radius==-1)
   {
      g.drawLine(X1, Y1, X2, Y2);
     return;
   }


   int CX = (int) (cx + scale*centre[1]);
   int CY = (int) (cy - scale*centre[2]);

   int theta1 = (int) (Math.atan2(-(Y1 - CY), X1 - CX)*180/Math.PI);
   int theta2 = (int) (Math.atan2(-(Y2 - CY), X2 - CX)*180/Math.PI);

   int deltaTheta = theta2-theta1;
   if (deltaTheta>+180 ) deltaTheta = 360 - deltaTheta;
   else if (deltaTheta<-180 ) deltaTheta = 360 + deltaTheta;

   int R = (int) (scale*radius);
   g.drawArc(CX-R,CY-R,2*R,2*R, theta1, deltaTheta);
}


// earlier prototype and experimental drawing functions
// (viewport parameters Ox, Oy & scale specified globally)
// (uses java.geom package)


public void drawCircle(Graphics g, double[] blade, double[] type)
{
    double[] vector = scale(gp(blade, I),type);
    double[] centre = f(getCentreFromVector(vector, type));
    drawPoint(g,centre);
    drawCircle(g,blade,Color.red);
}
```

```
public void drawCircle(Graphics g, double[] blade,
                       double[] type, Color c)
{
    double[] vector = scale(gp(blade, I),type);
    double[] centre = f(getCentreFromVector(vector, type));
    drawPoint(g,centre);
    drawCircle(g,blade,c);
}


 public void drawCircle(Graphics g, double[] blade, Color c)
 {
    g.setColor(c);
    double[] centre = f(getCentreFromBlade(blade, n));
    double   radius = getRadiusFromBlade(blade,n );//

    drawCircle(g,centre,radius,c);
 }


// general low level utility drawing functions

 public void drawCircle(Graphics g, double[] centre,
                        double radius, Color c)
 {
    Graphics2D g2D = (Graphics2D) g;
    double x = centre[1];
    double y = centre[2];
    double w = 2*radius;
    double h = 2*radius;
    drawCircle(g,x,y,w,h,c);
 }


public void drawCircle(Graphics g, double x, double y,
                       double w, double h, Color c)
{
  Graphics2D g2D = (Graphics2D) g;
  g2D.setColor(c);

  g2D.draw(new Ellipse2D.Double(Ox + scale*x - w*scale/2.0,
                                Oy - scale*y - h*scale/2.0,
                                w*scale, h*scale));
}

public void fillRectangle(Graphics g, double x, double y,
                          double w, double h, Color c)
{
  Graphics2D g2D = (Graphics2D) g;
  g2D.setColor(c);

  g2D.fill(new Rectangle2D.Double(Ox + scale*x - w*scale/2.0,
                                  Oy - scale*y - h*scale/2.0,
                                  w*scale, h*scale));
}
```

```
public void drawLine(Graphics g, double x1, double y1,
                     double x2, double y2)
{
   Graphics2D g2D = (Graphics2D) g;
   g.setColor(Color.red);
   g2D.draw(new Line2D.Double(Ox + scale*x1, Oy - scale*y1,
                              Ox + scale*x2, Oy - scale*y2));
}

public void drawGrid(Graphics g)
{
   g.setColor(Color.lightGray);
   for (int i= -4; i<=4; i++) drawLine(g, i,-4,i,4);
   for (int j= -4; j<=4; j++) drawLine(g, -4,j,4,j);
}
```

```
public void drawGrid(Graphics g,  boolean withCircle)
{
   drawGrid(g);
   if (withCircle) drawCircle(g,0,0,2,2, Color.lightGray );
}


 public void drawPoint(Graphics g, double[] p)
 {
    Graphics2D g2D = (Graphics2D) g;
    double x = p[1];
    double y = p[2];
    drawCircle(g2D,x,y, 0.05, 0.05, Color.blue);
 }

 public void drawSquarePoint(Graphics g, double[] p, Color c, int k)
 {
    Graphics2D g2D = (Graphics2D) g;
    double x = p[1];
    double y = p[2];
    fillRectangle(g2D,x,y, k*0.02, k*0.02,c);
 }

}
```

Appendix D Geometric algebra review

This selective review of certain aspects of the geometric algebra used in this thesis follows on from the introduction. It is not meant to serve as a comprehensive introduction.

Base vectors

Table D.1 shows the base vectors for geometric algebras built on base spaces of dimension n = 2, 3 and 4. (The thesis assumes that base spaces admit orthonormal basis denoted by e_1, e_2, e_3 and so on.)

n	scalar	vectors	bivectors	trivectors	
2	1	e_1, e_2	$e_{12} = I$		
3	1	e_1, e_2, e_3	e_{12}, e_{13}, e_{23}	$e_{123} = I$	
4	1	e_1, e_2, e_3, e_4	$e_{12}, e_{13}, e_{14}, e_{23}, e_{24}, e_{34}$	$e_{123}, e_{124}, e_{134}, e_{234}$	$e_{1234} = I$
	grade 0	grade 1	grade 2	grade 3	grade 4

Table D.1 Base vectors of geometric algebras built on base spaces of 2, 3 and 4 dimensions.

Table D.1 also shows the names and grades of the base vectors. The highest grade element in each case is known as the pseudoscalar and is often denoted by I.

The dimensions of the resulting geometric algebras are shown in table D.2. They result from summing the elements in each grade, equivalent to summing the rows of Pascal's triangle.

dim of base space	number of elements of each grade	dimension of geometric algebra
1	1 + 2 + 1	$= 4 = 2^2$
2	1 + 3 + 3 + 1	$= 8 = 2^3$
3	1 + 4 + 6 + 4 + 1	$= 16 = 2^4$

Table D.2 Number of elements of each grade and dimension of each geometric algebra.

Cayley table

As an example, table D.3 shows the Cayley table for the base vectors of the geometric algebra Cl(3). It gives the geometric product of any two base vectors.

	1	e1	e2	e3	e12	e13	e23	I
1	1	e1	e2	e3	e12	e13	e23	I
e1	e1	1	e12	e13	e2	e3	I	e23
e2	e2	-e12	1	e23	-e1	-I	e3	-e13
e3	e3	-e13	-e23	1	I	-e1	-e2	e12
e12	e12	-e2	e1	I	-1	-e23	e13	-e3
e13	e13	-e3	-I	e1	e23	-1	-e12	e2
e23	e23	I	-e3	e2	-e13	e12	-1	-e1
I	I	e23	-e13	e12	-e3	e2	-e1	-1

Table D.3 Cayley table for geometric products of Cl(3).

Because all multivectors in Cl(3) can be expressed as a linear combination of its eight base vectors, the Cayley table can be used to compute the geometric product of any number of multivectors.

The 'Clifford processor' discussed in Chapter 3 builds the Cayley table for a geometric algebra of any dimension and signature. This table is then used for calculating geometric and other derived products.

Other products of vectors

For two grade-1 vectors v_1 and v_2 the dot and wedge products are defined in terms of the geometric product as

$$v_1 . v_2 = \tfrac{1}{2}(v_1 v_2 + v_2 v_1) \qquad \text{(D.1a)}$$

and

$$v_1 \wedge v_2 = \tfrac{1}{2}(v_1 v_2 - v_2 v_1) . \qquad \text{(D.1b)}$$

The dot product is commutative and always yields a scalar. The wedge product is anti-commutative and yields a grade-2 bivector blade. (A blade is a multivector formed from the wedge product of one or more vectors, see below.)

Formulae D.1a and D.1b lead to

$$v_1 v_2 = v_1 . v_2 + v_1 \wedge v_2 . \qquad \text{(D.2)}$$

Thus, in general, the geometric product of two vectors *decomposes into a scalar part and a bivector part, either of which may have zero component.*

If $v_1 = v_2 = v$, say, then formula D.2 becomes

$$vv = v.v + v \wedge v,$$

so that

$$vv = v.v .$$

Thus the square of a vector can be defined in terms of the geometric or dot product as $v^2 = vv = v.v$.

If e_i is a non-degenerate base vector then $e_i^2 = e_i e_i = e_i . e_i = \pm 1$, depending on the signature of the base vector. If it is degenerate $e_i^2 = 0$. If e_i and e_j are two base vectors with $i \neq j$, then $e_i e_i = e_i \wedge e_i = e_{ij}$ and $e_i . e_j = 0$. These follow from the assumption that the $\{e_i\}$ basis is orthonormal. These multiplicative relationships among the vectors of the base space can be used for constructing

the Cayley table of products for the base vectors of the whole geometric algebra, see Chapter 3.

Inverse of a vector

If a vector is not degenerate so that $v^2 \neq 0$, then its geometric inverse is given by $v^{-1} = v/v^2$ since $vv^{-1} = vv/v^2 = 1$.

Extending the products

The dot and wedge products are usually first extended to homogeneous multivectors. (A homogeneous multivector has all elements of the same grade r, say, and is often denoted by A_r, B_r and so on.) The extension to general multivectors M and N then follows by applying the earlier extension to the various grade parts of M and N, often denoted by $\langle M \rangle_i$ and $\langle N \rangle_j$. This is discussed in more detail in chapter 3.

Sometimes other extensions are derived to cover special cases, for example formula D.2 can be extended to

$$Mv = M \cdot v + M \wedge v$$

where M is a multivector and v is a vector.

Blades

A blade is the wedge product of one or more vectors. This assumes that the definition of the wedge product has been extended for this statement to make sense. The extension is surprisingly complex and is discussed in chapter 3.

Blades represent the linear subspaces spanned by its vectors. If any two are linearly dependent, the space spanned is null so the wedge product has zero component.

Blades normally represent planes and hyperplanes. However, in the conformal model they represent n-spheres in general. In the conformal model Cl(3,1) they represent circles. In fact, the circle through the points P_1, P_2 and P_3 can be represented by the blade $p_1 \wedge p_2 \wedge p_3$ where p_1, p_2 and p_3 are the representations of points P_1, P_2 and P_3 in conformal space. However, there is an alternative representation for a circle based on a single vector in conformal space, see below.

Duals

The concept of a dual is perhaps best understood in the context of blades. The dual of a blade A_r of grade r, denoted by $A_r\tilde{}$, is the blade representing the linear subspace complementary to that represented by A_r. If the dimension of the geometric algebra is m, then the grade of the dual is m - r. For example, in Cl(3) the dual of the grade-2 blade representing a plane is a grade-1 blade representing a line. In the 4D conformal model Cl(3,1), the dual of a grade-3 trivector blade is a grade-1 vector. The former represent a 3D hyper plane normally and a circle conformally. The latter represents a 1D line normally, or a circle conformally. Thus a circle through points P_1, P_2 and P_3 can be represented conformally by the *vector* $(p_1 \wedge p_2 \wedge p_3)\tilde{}$. However, in this thesis the conformal vector representation of circles is approached through another route, see Chapter 1.

The dual of a multivector is surprisingly easy to calculate, it is the geometric product MI^{-1}, where I is the pseudoscalar. Calculating the inverse of I is relatively simple since $I^2 = \pm 1$, depending on the dimension and signature of the geometric algebra, so that $I^{-1} = \pm I$.

The exponential of a bivector

This thesis uses the following definition of the exponential of a bivector B based on the Taylor expansion

$$\begin{aligned} \exp(B) = \exp(k\hat{B}) &= \cosh(k) + \hat{B}\sinh(k), && \text{if } B^2 > 0, \\ &= \cos(k) + \hat{B}\sin(k), && \text{if } B^2 < 0, \\ &= 1 + k\hat{B}, && \text{if } B^2 = 0, \end{aligned}$$

where the scalar k is defined so that $B = k\hat{B}$ and $\hat{B}^2 = 1$.

This assumes that the bivector B squares to a scalar. Note that in all cases the result produces the combination of a scalar and bivector, which is the same as that produced by the product of two vectors (see above). This has implications when considering the transformation groups discussed briefly in chapter 5, sections 5.8 to 5.10.

THESIS CONTAINS CD/DVD