3

Elements of linear algebra

A few important concepts of linear algebra are rapidly recalled in this chapter, mostly without proofs. Our goal is to recall the algebraic framework of computer graphics and geometric modeling, as well as to discuss the algebraic concepts underlying the geometric calculus with `PLaSM`. In particular, we show in this chapter how `PLaSM` represents vectors and vector operations. Also, we introduce its functional representation of linear (invertible) transformations as vector space automorphisms. This approach is quite different from what is usually done in computer graphics, where linear transformations of vector spaces are most often represented by normalized homogeneous matrices. We only assume that the reader is already acquainted with some matrix calculus.

3.1 Vector spaces

A vector space $\mathcal{V}$ over a field $\mathcal{F}$ is a set with two composition rules

$$\begin{array}{ll} + & : \mathcal{V} \times \mathcal{V} \to \mathcal{V} \text{ (addition)} \\ \cdot & : \mathcal{F} \times \mathcal{V} \to \mathcal{V} \text{ (product by a scalar)} \end{array}$$

such that, for each $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathcal{V}$ and for each $\alpha, \beta \in \mathcal{F}$, the rules $+, \cdot$ satisfy the following axioms:

1. $\boldsymbol{v} + \boldsymbol{w} = \boldsymbol{w} + \boldsymbol{v}$; (commutativity of addition)
2. $\boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w}) = (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w}$; (associativity of addition)
3. there is a $\mathbf{0} \in \mathcal{V}$ such that $\boldsymbol{v} + \mathbf{0} = \boldsymbol{v}$; (neutral element of addition)
4. there is a $-\boldsymbol{v} \in \mathcal{V}$ such that $\boldsymbol{v} + (-\boldsymbol{v}) = \mathbf{0}$; (inverse of addition)
5. $\alpha \cdot (\boldsymbol{v} + \boldsymbol{w}) = \alpha \cdot \boldsymbol{v} + \alpha \cdot \boldsymbol{w}$; (distributivity of addition w.r.t. product)
6. $(\alpha + \beta) \cdot \boldsymbol{v} = \alpha \cdot \boldsymbol{v} + \alpha \cdot \boldsymbol{v}$; (distributivity of product w.r.t. addition)
7. $\alpha \cdot (\beta \cdot \boldsymbol{v}) = (\alpha\beta) \cdot \boldsymbol{v}$; (associativity of product)
8. $1 \cdot \boldsymbol{v} = \boldsymbol{v}$. (neutral element of product)

The dot operator to multiply a vector by a scalar will be dropped in the remainder of this book, so that the operation will be denoted by the juxtaposition of the arguments. Hence we write $\alpha(\beta\boldsymbol{v})$ instead of $\alpha \cdot (\beta \cdot \boldsymbol{v})$.

Geometric Programming for Computer-Aided Design Alberto Paoluzzi
© 2003 John Wiley & Sons, Ltd ISBN 0-471-89942-9

Example 3.1.1 (Vector space of real matrices)
Let $\mathcal{M}_n^m(\mathrm{I\!R})$ be the set of $m \times n$ matrices with elements in the field $\mathrm{I\!R}$. An element A in such a set is denoted as

$$A = (\alpha_{ij})$$

Addition and multiplication by a scalar are defined component-wise:

$$A + B = (\alpha_{ij}) + (\beta_{ij}) = (\alpha_{ij} + \beta_{ij})$$

$$\gamma A = \gamma(\alpha_{ij}) = (\gamma\alpha_{ij})$$

To indicate the vector space of matrices with either only one column or only one row, we write, respectively

$$\begin{array}{rclc}
\mathcal{M}_1^m(\mathrm{I\!R}) & = & \mathrm{I\!R}^m & \text{(column vectors)} \\
\mathcal{M}_n^1(\mathrm{I\!R}) & = & \mathrm{I\!R}_n & \text{(row vectors)} \\
\mathcal{M}_n^m(\mathrm{I\!R}) & = & \mathrm{I\!R}_n^m & \text{(matrices)}
\end{array}$$

Subspace Let $(\mathcal{V}, +, \cdot)$ be a vector space on the field $\mathcal{F}$. We say that $\mathcal{U} \subset \mathcal{V}$ is a *subspace* of $\mathcal{V}$ if $(\mathcal{U}, +, \cdot)$ is a vector space with respect to the same operations. In particular, $\mathcal{U} \subset \mathcal{V}$ is a *subspace* of $\mathcal{V}$ if and only if:

1. $\mathcal{U} \neq \emptyset$;
2. for each $\alpha \in \mathcal{F}$ and $\boldsymbol{u}_1, \boldsymbol{u}_2 \in \mathcal{U}$, $\alpha\boldsymbol{u}_1 + \boldsymbol{u}_2 \in \mathcal{U}$

The *codimension* of a subspace $\mathcal{U} \subset \mathcal{V}$ is defined as $\dim \mathcal{V} - \dim \mathcal{U}$. It is useful to note that the intersection of subspaces is a subspace. In particular, if $\mathcal{U}_1, \mathcal{U}_2$ are subspaces of $\mathcal{V}$, then $\mathcal{U}_1 \cap \mathcal{U}_2$ is a subspace of $\mathcal{V}$.

Linear combination Let $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n \in \mathcal{V}$ and $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathcal{F}$, ($\mathcal{V}$ a vector space on the field $\mathcal{F}$). The vector

$$\alpha_1\boldsymbol{v}_1 + \cdots + \alpha_n\boldsymbol{v}_n = \sum_{i=1}^{n} \alpha_i\boldsymbol{v}_i \in \mathcal{V}$$

is called a *linear combination* of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$.

The set of all linear combinations of elements of a set $S \subset \mathcal{V}$ is a subspace of $\mathcal{V}$. Such a subspace is called the *span* of S and is denoted as $\mathrm{lin}\, S$. If a subspace $\mathcal{U}$ of $\mathcal{V}$ can be generated as the span of a set S of vectors in $\mathcal{V}$, then S is called a *generating set* or a *spanning set* for $\mathcal{U}$.

A set of vectors $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\}$ is *linearly independent* if $\sum_{i=1}^n \alpha_i\boldsymbol{v}_i = \mathbf{0}$ implies that $\alpha_i = 0$ for each i. As a consequence, a set of vectors is linearly independent when none of them belongs to the span of the others.

3.1.1 Bases and coordinates

When working with vector spaces, the concept of *basis*, a discrete subset of linearly independent elements, is probably the most useful to deal with. In fact, each element of the space can be represented uniquely as linear combination of basis elements. This leads to a *parametrization* of the space, i.e. to representing each element by using a sequence of scalars, called its *coordinates* with respect to the chosen basis.

Basis A set of vectors $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$ is a *basis* for the vector space $\mathcal{V}$ if the set is linearly independent, and $\mathcal{V} = \mathrm{lin}\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$.

Some important properties of the bases of a vector space are listed below:

1. each spanning set for $\mathcal{V}$ contains a basis;
2. each minimal spanning set is a basis;
3. each linearly independent set of vectors is contained in a basis;
4. each maximal set[1] of linearly independent vectors is a basis;
5. two bases of $\mathcal{V}$ have the same number ($\dim \mathcal{V}$) of elements, that is called the *dimension* of $\mathcal{V}$.

Components If $(\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n)$ is an ordered basis for $\mathcal{V}$, then for each $\boldsymbol{v} \in \mathcal{V}$ there exists a *unique* n-tuple of scalars $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathcal{F}$ such that

$$\boldsymbol{v} = \sum_{i=1}^{n} \alpha_i \boldsymbol{e}_i .$$

The n-tuple of scalars (α_i) is called the *components* of $\boldsymbol{v}$ with respect to the ordered basis $(\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n)$. If such a n-tuple were not unique, then $\boldsymbol{v} = \sum \alpha_i \boldsymbol{e}_i = \sum \beta_i \boldsymbol{e}_i$. But this one would imply $\sum(\alpha_i - \beta_i)\boldsymbol{e}_i = \boldsymbol{0}$, hence $(\alpha_i - \beta_i) = 0$, i.e. $\alpha_i = \beta_i$, for each i.

3.1.2 `PLaSM` *representation of vectors*

An element of a vector space on a field will be represented in `PLaSM` as a *sequence* of field elements. In this book we usually consider sequences of either real numbers or polynomial functions of bounded degree.

It is very useful to start our computer representation of a vector space by defining a predicate, i.e. a `PLaSM` function which returns a truth value, in order to test if the value returned by a language expression is a vector. First, we define a vector as a sequence of real numbers.

```
DEF IsVect = IsSeqOf:IsReal;
```

A more general set of definitions is given in Script 3.1.1, that allows us to consider a sequence of either real numbers or functions as a vector. The `PLaSM` representation of matrices will be shown in Script 3.3.3.

[1] A set S is said to be *maximal* with respect to some property $\mathcal{P}$ when S satisfies $\mathcal{P}$ and S is not a proper subset of any other set which satisfies $\mathcal{P}$.

Script 3.1.1 (`IsVect` predicate definition)

```
DEF IsRealVect = IsSeqOf:IsReal;
DEF IsFunVect = IsSeqOf:IsFun;
DEF IsVect = OR∼[IsRealVect,IsFunVect];
```

Example 3.1.2 (`IsVect` predicate application)
With the last definition for `IsVect` the right-hand values will be returned by the interpreter when evaluating the left-hand expressions. Remember that ≡ means that the *expression* on the left-hand side evaluates to the *value* on the right-hand side.

```
IsVect:<1.0,2.0,3.0> ≡ TRUE
IsVect:<1,2,3,4,5,6> ≡ TRUE
IsVect:<SIN,COS,TAN> ≡ TRUE
IsVect:<+,-,*,ID,TRANS> ≡ TRUE
IsVect:<+,-,*,1,2,3> ≡ FALSE
```

Vector operations Below the `PLaSM` implementation of the most common vector operations in $\mathcal{V}^n$ is given. Notice that the language allows us to work indifferently with real vectors and with function vectors. Also, most of the following operations work in vector spaces of any dimension.

In particular, the `VectSum` and `VectDiff`, respectively for *vector sum* and *vector difference*, transpose the argument pair, and finally apply to each generated pair of numbers either the sum or difference operator, respectively. The operation `ScalarVectProd` multiplies a scalar times a vector. To better understand the meaning of such definitions, the reader should look again at Script 2.1.19.

Script 3.1.2 (Vector sum, vector difference and product times a scalar)

```
DEF VectSum = AA:+ ∼ TRANS;
DEF VectDiff = AA:- ∼ TRANS;
DEF ScalarVectProd = AA:* ∼ IF:< IsNum ∼ S1, DISTL, DISTR >;
```

As always in `PLaSM`, binary operators can be applied to both prefix and infix. Let us notice that `ScalarVectProd` can be indifferently applied to the pair $(\alpha, \boldsymbol{v})$ and to the input pair $(\boldsymbol{v}, \alpha)$. If the first element of the pair argument is a number, then the `DISTL` operator is applied to the input, else the `DISTR` operator is used.

```
VectSum:  <<11,12,13>,<4,5,6>> ≡ <15, 17, 19>
VectDiff: <<11,12,13>,<4,5,6>> ≡ <7, 7, 7>
<11,12,13>  VectSum  <4,5,6> ≡ <15, 17, 19>
<11,12,13>  VectDiff <4,5,6> ≡ <7, 7, 7>
ScalarVectProd:<10, <1,2,3>> ≡ <10, 20, 30>
ScalarVectProd:<<1,2,3>, 10> ≡ <10, 20, 30>
```

Example 3.1.3 (Vector product in 3D)
Let us compute the vector product of $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^3$. The `PLaSM` implementation is

straightforward if you remember that $\boldsymbol{u} \times \boldsymbol{v}$ is defined as the determinant of a matrix:

$$\boldsymbol{w} = \boldsymbol{u} \times \boldsymbol{v} = \det \begin{pmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{pmatrix} = \begin{vmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \boldsymbol{e}_3 \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix}.$$

Hence we have:

$$\boldsymbol{w} = \begin{vmatrix} u_2 & u_3 \\ v_2 & v_3 \end{vmatrix} \boldsymbol{e}_1 - \begin{vmatrix} u_1 & u_3 \\ v_1 & v_3 \end{vmatrix} \boldsymbol{e}_2 + \begin{vmatrix} u_1 & u_2 \\ v_1 & v_2 \end{vmatrix} \boldsymbol{e}_3$$

Script 3.1.3

```
DEF vectProd (u,v::IsVect) = <w1, w2, w3>
   WHERE
      w1 = (u2 * v3) - (u3 * v2),
      w2 = (u3 * v1) - (u1 * v3),
      w3 = (u1 * v2) - (u2 * v1),
      u1 = s1:u, u2 = s2:u, u3 = s3:u,
      v1 = s1:v, v2 = s2:v, v3 = s3:v
   END;

vectProd:<<1,0,0>,<1,1,0>> ≡ <0, 0, 1>
```

3.2 Affine spaces

The idea of affine space corresponds to that of a set of points where the *displacement* from a point $\boldsymbol{x}$ to another point $\boldsymbol{y}$ is obtained by summing a vector $\boldsymbol{v}$ to the $\boldsymbol{x}$ point.

Affine space A set $\mathcal{A}$ of points is called an *affine space* modeled on the vector space $\mathcal{V}$ if there is a function

$$\mathcal{A} \times \mathcal{V} \to \mathcal{A} : (\boldsymbol{x}, \boldsymbol{v}) \mapsto \boldsymbol{x} + \boldsymbol{v}$$

called *affine action*, with the properties:

1. $(\boldsymbol{x} + \boldsymbol{v}) + \boldsymbol{w} = \boldsymbol{x} + (\boldsymbol{v} + \boldsymbol{w})$ for each $\boldsymbol{x} \in \mathcal{A}$ and each $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{V}$;
2. $\boldsymbol{x} + \mathbf{0} = \boldsymbol{x}$ for each $\boldsymbol{x} \in \mathcal{A}$, where $\mathbf{0} \in \mathcal{V}$ is the null vector;
3. for each pair $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{A}$ there is a unique $(\boldsymbol{y} - \boldsymbol{x}) \in \mathcal{V}$ such that

$$\boldsymbol{x} + (\boldsymbol{y} - \boldsymbol{x}) = \boldsymbol{y}.$$

Dimension The affine space $\mathcal{A}$ is said of *dimension* n if modeled on a vector space $\mathcal{V}$ of dimension n.

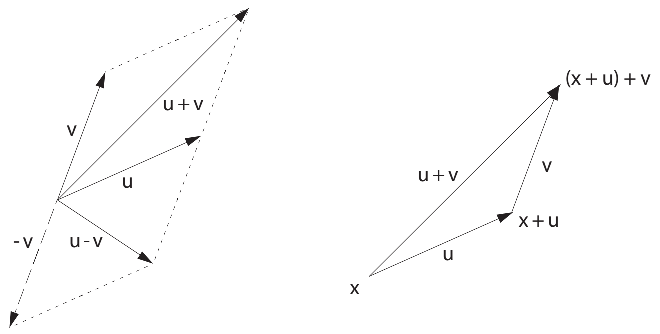

Figure 3.1 (a) Vector sum and difference are given by the parallelogram rule (b) associativity of displacement (point and vector sum) in an affine space

3.2.1 Operations on vectors and points

The *addition* of vectors is a primitive operation in a vector space. The *difference* of vectors is defined through the two primitive operations:

$$\boldsymbol{v}_1 - \boldsymbol{v}_2 = \boldsymbol{v}_1 + (-1)\boldsymbol{v}_2.$$

Addition and difference of vectors are geometrically produced by the parallelogram rule, shown in Figure 3.1a. Figure 3.1b shows the associative property of the affine action on a point space. Notice also that:

1. the addition of points is *not* defined;
2. the difference of two points is a vector;
3. the sum of a point and a vector is a point.

The sum of a set $\{\boldsymbol{v}_i\}$ of vectors $(i = 1, \ldots, n)$ can be geometrically obtained, in an affine space, by setting $\boldsymbol{p}_0 = \boldsymbol{0}$ and $\boldsymbol{p}_i = \boldsymbol{p}_{i-1} + \boldsymbol{v}_i$, so that $\sum_i \boldsymbol{v}_i = \boldsymbol{p}_n - \boldsymbol{p}_0$.

3.2.2 Positive, affine and convex combinations

Three types of combinations of vectors or points can be defined. They lead to the concepts of cones, hyperplanes and convex set, respectively.

Positive combination Let $\boldsymbol{v}_0, \ldots, \boldsymbol{v}_d \in \mathbb{R}^n$ and $\alpha_0, \ldots, \alpha_d \in \mathbb{R}^+ \cup \{0\}$. The vector

$$\alpha_0 \boldsymbol{v}_0 + \cdots + \alpha_d \boldsymbol{v}_d = \sum_{i=0}^{d} \alpha_i \boldsymbol{v}_i$$

is called a *positive combination* of such vectors. The set of all the positive combinations of $\{\boldsymbol{v}_0, \ldots, \boldsymbol{v}_d\}$ is called the *positive hull* of $\{\boldsymbol{v}_0, \ldots, \boldsymbol{v}_d\}$ and denoted pos $\{\boldsymbol{v}_0, \ldots, \boldsymbol{v}_d\}$. This set is also called the *cone* generated by the given elements.

Affine combination Let $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d \in \mathbb{E}^n$ and $\alpha_0, \ldots, \alpha_d \in \mathbb{R}$, such that $\alpha_0 + \cdots + \alpha_d = 1$. The point

$$\sum_{i=0}^{d} \alpha_i \boldsymbol{p}_i := \boldsymbol{p}_0 + \sum_{i=1}^{d} \alpha_i (\boldsymbol{p}_i - \boldsymbol{p}_0) \tag{3.1}$$

is called an *affine combination* of the points $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d$.

The set of all affine combinations of $\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d\}$ is an affine subspace, denoted by $\text{aff}\,\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d\}$. It is easy to verify that:

$$\text{aff}\,\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d\} = \boldsymbol{p}_0 + \text{lin}\,\{\boldsymbol{p}_1 - \boldsymbol{p}_0, \ldots, \boldsymbol{p}_d - \boldsymbol{p}_0\}.$$

The *dimension* of an affine subspace is the dimension of the corresponding linear vector space. Affine subspaces of $\mathbb{E}^d$ with dimensions 0, 1, 2 and $d-1$ are called *points*, *lines*, *planes* and *hyperplanes*, respectively. Affine subspaces are also called *flats*.

Every affine subspace can be described either as the intersection of affine hyperplanes, or as the affine hull of a finite set of points.

Convex combination Let $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d \in \mathbb{E}^n$ and $\alpha_0, \ldots, \alpha_d \in \mathbb{R}^+ \cup \{0\}$, with $\alpha_0 + \cdots + \alpha_d = 1$. The point

$$\alpha_0 \boldsymbol{p}_0 + \cdots + \alpha_d \boldsymbol{p}_d = \sum_{i=0}^{d} \alpha_i \boldsymbol{p}_i$$

is called a *convex combination* of points $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d$.

A convex combinations is both affine and positive. The set of all convex combinations of $\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d\}$ is a convex set, called *convex hull* of $\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d\}$, and is denoted by $\text{conv}\,\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_d\}$.

Example 3.2.1 (Convex hull)
In `PLaSM`, an operator which generates the convex hull of a discrete set of points can be easily defined by using the primitive function `MKPOL`. We remember (see Section 1.5.1) that this one needs as input a triplet `< points, cells, polyhedra >`, with a sequence of points, a sequence of convex cells and a sequence of polyhedral cells, respectively.

In order to generate the set

$$\text{conv}\,\{\boldsymbol{p}_i \in \mathbb{E}^d | i = 1, \ldots, n\}$$

of a discrete set of points, a `ConvexHull` operator can be defined in a very straightforward way, as done in Script 3.2.1. We like to note that the *beneath-beyond* algorithm [Ede87], used by `PLaSM` to compute the convex hull, is actually hidden from the user by the language interface.

According with the *dimension-independent* nature of the language, the `ConvexHull` function may be applied to sequences of points in Euclidean spaces of arbitrary finite dimension. In Figure 3.2 are shown the Monge projections of the convex hull generated by the last row of Script 3.2.1.

Script 3.2.1

```
DEF ConvexHull (points::IsSeq) = MKPOL:< points, <1..LEN:points>, <<1>> >

ConvexHull:<<0,0>,<1,0>,<0,1>,<1,1>,<2,3>,<0.5,4>>
ConvexHull:<<0,0,2>,<1,0,-1>,<0,1,1>,<1,1,0.5>,<2,3,0>,<0.5,4,-1.5>>
```

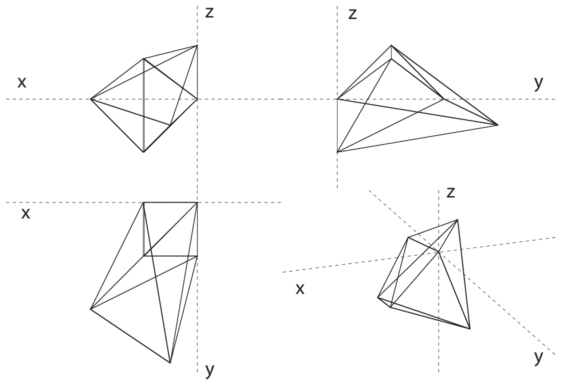

Figure 3.2 Monge projections of the convex hull of the set of points of $\mathbb{E}^3$ given in Script 3.2.1

3.2.3 Linear, affine and convex independence

A set of *vectors* is said to be *linearly independent* if none of them can be obtained as a linear combination of the other ones.

Analogously, a set of *points* is said *affinely/convexely independent* if none of them can be obtained as an affine/convex combination of the other ones, respectively.

Example 3.2.2 (Affine combination)
The set of affine combinations of two affinely independent (i.e. distinct) points $\boldsymbol{p}_0, \boldsymbol{p}_1 \in \mathbb{E}^n$, $\boldsymbol{p}_0 \neq \boldsymbol{p}_1$, is the line through them:

$$\begin{aligned} \alpha_0 \boldsymbol{p}_0 + \alpha_1 \boldsymbol{p}_1 &:= (1-\alpha_1)\boldsymbol{p}_0 + \alpha_1 \boldsymbol{p}_1 \\ &= \boldsymbol{p}_0 + \alpha_1(\boldsymbol{p}_1 - \boldsymbol{p}_0) \end{aligned}$$

Remember that a difference of points is a vector, and that the sum of a point and a vector is a point.

Example 3.2.3 (Convex combination)
The set of convex combinations of two affinely independent (i.e. distinct) points $\boldsymbol{p}_0, \boldsymbol{p}_1 \in \mathbb{E}^n$, $\boldsymbol{p}_0 \neq \boldsymbol{p}_1$, is the line segment joining $\boldsymbol{p}_0$ with $\boldsymbol{p}_1$:

$$\begin{aligned} \boldsymbol{p}(\beta) &= (1-\beta)\boldsymbol{p}_0 + \beta \boldsymbol{p}_1 \\ &= \boldsymbol{p}_0 + \beta(\boldsymbol{p}_1 - \boldsymbol{p}_0), \qquad 0 \leq \beta \leq 1 \end{aligned}$$

Notice that:

$$\boldsymbol{p}(0) = \boldsymbol{p}_0, \quad \text{and} \quad \boldsymbol{p}(1) = \boldsymbol{p}_1.$$

The set of affine combinations of three affinely independent (i.e. not aligned) points is their plane. The set of convex combinations of three affinely independent points is their triangle, i.e. the triangle whose vertices are those points.

Example 3.2.4 (Plane for three points)
A non-trivial geometric construction is done in Script 3.2.2 using quite typical graphics methods. The resulting object is shown in Figure 3.3. In particular, we give the definition of a `plane` function, with with parameters `point0`, `point1` and `point2`. This function returns as output a triplet constituted by the `normal` vector, by `point0` and by a transparent rectangle representing the plane `geometry`. We leave to the reader the task of reading and interpreting the given `PLaSM` code. We notice that (a) the `optimize` function, given in Script 6.5.2, that actually performs some "flattening" of the internal data structure, is needed to correctly export to VRML the generated rectangle, and (b) that the `PLaSM` libraries `'colors'` and `'vectors'` must firstly be loaded in the `PLaSM` environment.[2]

Script 3.2.2 (2-plane in $\mathbb{E}^3$)
```
DEF plane (point0, point1, point2::IsPoint) = < normal, point0, geometry >
WHERE
   normal = (unitVect ∼ vectProd):< v1, v2 >,
   v1 = point1 vectDiff point0,   v2 = point2 vectDiff point0,
   axis = <0,0,1> vectProd normal,
   side1 = vectNorm:v1,   side2 = vectNorm:v2,
   angle = ACOS:(<0,0,1> innerProd unitVect:normal),
   geometry = (Optimize
      ∼ T:<1,2,3>:point0
      ∼ Rotn:< angle, axis >
      ∼ EMBED:1
      ∼ T:<1,2>:< -1*side1,-1*side2 >
      ∼ CUBOID): <2*side1, 2*side2 >
   MATERIAL Transparentmaterial:<GREEN, 0.8>
END;

DEF out = (S3 ∼ plane):<<0,0,0>,<1,0,0>,<1,1,1>> STRUCT MKframe;
```

The functions for vector calculus named `unitVect`, `vectProd`, `vectDiff`, `vectNorm`, `innerProd` and `Rotn` are given in following examples of this chapter, but their meaning can easily be forecast by their name. `MKframe` is the constructor of the model of the 3D standard basis, and is given in Script 6.5.3. The `MATERIAL`, `GREEN` and `Transparentmaterial` symbols are defined in the `'colors'` library.

[2] Currently, this loading is automatically done at the starting of the interpreter.

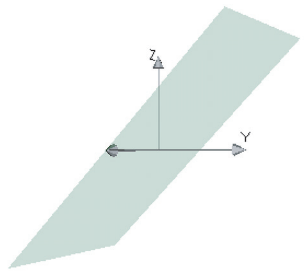

Figure 3.3 Plane passing for $(0,0,0)$, $(1,0,0)$ and $(1,1,1)$

3.2.4 Convex coordinates

Given a point $\boldsymbol{p} \in \operatorname{conv}\{\boldsymbol{p_0}, \ldots, \boldsymbol{p_d}\}$, the scalars $\alpha_0, \ldots, \alpha_d$ such that:

$$\boldsymbol{p} = \alpha_0 \boldsymbol{p_0} + \cdots + \alpha_d \boldsymbol{p_d}.$$

are called *convex coordinates* of $\boldsymbol{p}$. Convex coordinates are *unique* if the points $\boldsymbol{p_0}, \ldots, \boldsymbol{p_d}$ are affinely independent.

Example 3.2.5 (Triangle through 3 points)
Given four points $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}, \boldsymbol{s} \in \mathbb{E}^2$, with $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}$ non-colinear, there are two disjoint possibilities:

$$\boldsymbol{s} \in \operatorname{conv}\{\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}\}, \qquad \boldsymbol{s} \notin \operatorname{conv}\{\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}\}.$$

In the first case there exists a triplet of numbers (α, β, γ), with $\alpha + \beta + \gamma = 1$ and $0 \leq \alpha, \beta, \gamma \leq 1$, such that:

$$\boldsymbol{s} = \alpha \boldsymbol{p} + \beta \boldsymbol{q} + \gamma \boldsymbol{r}. \tag{3.2}$$

Notice that the product of a point by a scalar has not been defined. However, equation (3.2) may be given meaning as follows:

$$\begin{aligned} \boldsymbol{s} &= \alpha \boldsymbol{p} + \beta \boldsymbol{q} + \gamma \boldsymbol{r} \\ &= (1 - \beta - \gamma)\boldsymbol{p} + \beta \boldsymbol{q} + \gamma \boldsymbol{r} \\ &= \boldsymbol{p} + \beta(\boldsymbol{q} - \boldsymbol{p}) + \gamma(\boldsymbol{r} - \boldsymbol{p}) \end{aligned}$$

Notice also that the three vertices $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}$ are associated with convex coordinates $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$, respectively.

Example 3.2.6 (Convex coordinates and volume)
The convex coordinates (α, β, γ) of a point $\boldsymbol{s} \in \operatorname{conv}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}) \subset \mathbb{E}^d$ can be defined as the ratios between the areas of suitable triangles. In particular:

$$\alpha = \frac{\operatorname{area}(\boldsymbol{s}, \boldsymbol{q}, \boldsymbol{r})}{\operatorname{area}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r})}, \quad \beta = \frac{\operatorname{area}(\boldsymbol{p}, \boldsymbol{s}, \boldsymbol{r})}{\operatorname{area}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r})}, \quad \gamma = \frac{\operatorname{area}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{s})}{\operatorname{area}(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r})}.$$

as shown in Figure 3.4.

This property immediately generalizes to any convex set in any d-dimensional space, by using ratios of volumes of d-simplices.

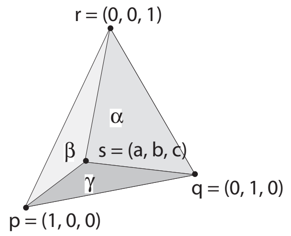

Figure 3.4 Interpretation of convex coordinates as ratios of triangle areas

A few examples of both affine or convex independence follow:

1. two distinct points are independent in any $\mathbb{E}^d$, $d \geq 1$;
2. three non-colinear points are independent in any $\mathbb{E}^d$, $d \geq 2$;
3. four non-coplanar points are independent in any $\mathbb{E}^d$, $d \geq 3$;
4. $d+1$ points not belonging to the same affine subspace of codimension 1 (i.e. dimension $d-1$) are independent in any $\mathbb{E}^D$, $D \geq d$;
5. m points are convexely independent if none of them is in the convex hull of the other $m-1$ points.

Example 3.2.7 (Convex polygon)
The vertices of a convex polygon in $\mathbb{E}^2$ are convexly independent, as none of them can be generated as a convex combination of the others. This is easy to see, since each vertex is external to the convex hull of the other vertices (see Figure 3.5a). All of the vertices of a convex polyhedron in $\mathbb{E}^d$ are convexely independent for the same reason.

Notice that points in conv $\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n\}$, with $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n \in \mathbb{E}^d$, have *non unique* convex coordinates when $d < n$ (see, e.g. Figure 3.5b).

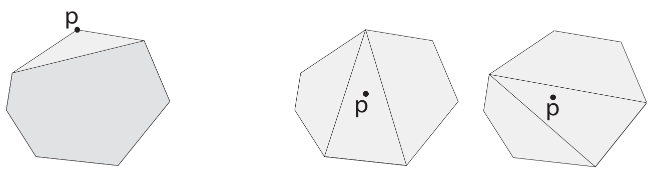

Figure 3.5 (a) Each vertex of a convex polygon is external to the convex hull of the others (b) Convex coordinates of an internal point are not unique when the point is contained in more than one simplex

Parametric form of affine sets The affine subspace generated by $d+1$ affinely independent points $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_d \in \mathbb{E}^n$, also called a *d-flat*, can be written in *parametric form* as an affine function of d independent real parameters:

$$S(\alpha_1, \ldots, \alpha_d) = \{\boldsymbol{p} \in \mathbb{E}^n | \ \boldsymbol{p} = \boldsymbol{p}_0 + \alpha_1(\boldsymbol{p}_1 - \boldsymbol{p}_0) + \cdots + \alpha_d(\boldsymbol{p}_d - \boldsymbol{p}_0)\}$$

Example 3.2.8 (Parametric segment)
Consider the segment in $\mathbb{E}^2$ joining $\boldsymbol{p}_0 = (p_{0x}, p_{0y})$ with $\boldsymbol{p}_1 = (p_{1x}, p_{1y})$:

$$L(\alpha) = \{\boldsymbol{p} \in \mathbb{E}^2 | \boldsymbol{p} = \boldsymbol{p}_0 + \alpha(\boldsymbol{p}_1 - \boldsymbol{p}_0)\}.$$

Example 3.2.9 (Parametric plane)
A plane (or "2-flat") in $\mathbb{E}^n$ generated by points $\boldsymbol{q}, \boldsymbol{r}, \boldsymbol{s}$ is defined as:

$$P(\alpha, \beta) = \{\boldsymbol{p} \in \mathbb{E}^n |\ \boldsymbol{p} = (1-\beta)((1-\alpha)\boldsymbol{q} + \alpha\boldsymbol{r}) + \beta\boldsymbol{s}\} \tag{3.3}$$

A direct implementation of definition (3.3) is given in Script 3.2.3. It is a quite advanced example (for *advanced readers* only) that makes use of bilinear *transfinite interpolation* (see Section 12.5). It is interesting to notice that it generates a 2-flat in $\mathbb{E}^d$, depending on the number d of coordinates of the actual values for `p0`, `p1` and `p2` parameters. Notice also that the vector calculus functions `scalarVectProd` and `vectSum`, defined in this chapter, are now working in a functional vector space. The `IsPoint` predicate is just an alias for `IsVect`. The geometric object generated in $\mathbb{E}^3$ is quite similar to the one generated by Script 3.2.2.

Script 3.2.3 (2-plane in $\mathbb{E}^d$, $2 \leq d$)

```
DEF IsPoint = IsVect;

DEF mapping (p0,p1,p2::IsPoint) = CONS:(
   ((K:1 - S2) scalarVectProd (term0 vectSum term1)) vectSum term2)
WHERE
   term0 = (K:1 - S1) scalarVectProd AA:K:p0,
   term1 = S1 scalarVectProd AA:K:p1,
   term2 = S2 scalarVectProd AA:K:p2
END;

DEF out = MAP:(mapping:<<0,0,0>,<1,0,0>,<1,1,1>>):(CUBOID:<1,1>)
   MATERIAL Transparentmaterial:< GREEN, 0.8>
   STRUCT MKframe
```

3.2.5 Euclidean spaces

Let now introduce in a vector space the new structure induced by the functions of *norm* and *inner product*, also called *scalar product*, which generalize the concepts of length and angle, respectively. A (real) vector space equipped with norm and inner product is said to be a *Euclidean space*.

Norm Let $\mathcal{V}$ be a real vector space. A *norm* on $\mathcal{V}$ is defined as a function $\mathcal{V} \to \mathbb{R} : \boldsymbol{u} \mapsto \|\boldsymbol{u}\|$, which satisfies the following axioms for each $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$ and $\alpha \in \mathbb{R}$:

1. $\|\boldsymbol{u}\| \geq 0$, with $\|\boldsymbol{u}\| = 0$ if and only if $\boldsymbol{u} = \boldsymbol{0}$; (positivity)
2. $\|\alpha\boldsymbol{u}\| = |\alpha|\|\boldsymbol{u}\|$ (scalar multiple)
3. $\|\boldsymbol{u}\| + \|\boldsymbol{v}\| \geq \|\boldsymbol{u} + \boldsymbol{v}\|$ (triangle inequality)

Inner product A *Euclidean inner product* on $\mathcal{V}$ is a bilinear function $\cdot : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ which satisfies the following properties for each $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathcal{V}$ and $\alpha \in \mathbb{R}$:

1. $\boldsymbol{v} \cdot (\boldsymbol{u} + \boldsymbol{w}) = \boldsymbol{v} \cdot \boldsymbol{u} + \boldsymbol{v} \cdot \boldsymbol{w}$ (distributivity w.r.t. vector addition)

2. $(\alpha \boldsymbol{v}) \cdot \boldsymbol{u} = \alpha(\boldsymbol{v} \cdot \boldsymbol{u})$ (associativity w.r.t. scalar product)
3. $\boldsymbol{v} \cdot \boldsymbol{w} = \boldsymbol{w} \cdot \boldsymbol{v}$ (commutativity)
4. $\boldsymbol{u} \cdot \boldsymbol{u} \geq 0$, with $\boldsymbol{u} \cdot \boldsymbol{u} = 0$ iff $\boldsymbol{u} = \boldsymbol{0}$. (positivity)

The inner product of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ is defined so that the inner product and norm functions are linked by the equality:

$$\boldsymbol{u} \cdot \boldsymbol{u} =: \|\boldsymbol{u}\|^2$$

Orthogonal vectors Let $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$, where $\mathcal{V}$ is an inner product space. We say that $\boldsymbol{u}$ is orthogonal to $\boldsymbol{v}$ when

$$\boldsymbol{u} \cdot \boldsymbol{v} = 0$$

Notice that the null vector is orthogonal to all vectors, since $\boldsymbol{0} \cdot \boldsymbol{u} = 0$ for each $\boldsymbol{u} \in \mathcal{V}$. Two subspaces U, V of a vector space $\mathcal{V}$ equipped with an inner product are said to be *orthogonal* if each vector in U is orthogonal to each vector in V. A set of vectors $(\boldsymbol{e}_i)$ is said to be *orthonormal* when

$$\boldsymbol{e}_i \cdot \boldsymbol{e}_j = \begin{array}{ll} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{array}$$

Each set of orthonormal vectors in an inner product space $\mathcal{V}$ is linearly independent. Hence, there is no orthonormal set having more than $n = \dim \mathcal{V}$ elements.

Euclidean point space An affine space modeled on the vector space $\mathbb{R}^n$ and equipped with the standard inner product is called *Euclidean space* $\mathbb{E}^n$. It can be proved (see e.g. [LD93], pages 122–133) that a two dimensional affine space with the additional structure given by an Euclidean inner product on the underlying vector space satisfies the congruence postulates and Hilbert's parallel postulate. Such an affine space is hence a good algebraic model for the Euclidean plane.

Cartesian coordinates A *Cartesian coordinate system* in an Euclidean space $\mathbb{E}^n$ modeled on a vector space $\mathbb{R}^n$ consists of a *orthonormal basis*

$$(\boldsymbol{e}_i) = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$$

in $\mathbb{R}^n$, together with a point $\boldsymbol{o} \in \mathbb{E}^n$ called the *origin.* In such a system the components of a vector $\boldsymbol{u}$ are $u_i = \boldsymbol{u} \cdot \boldsymbol{e}_i$. Analogously, the coordinates of a point $\boldsymbol{x}$ are:

$$x_i = (\boldsymbol{x} - \boldsymbol{o}) \cdot \boldsymbol{e}_i.$$

Linear forms and the dual space

The linear functions from a vector space $\mathcal{V}$ into $\mathbb{R}$ are called *linear forms.* A linear form $\psi : \mathcal{V} \to \mathbb{R}$ satisfies the following properties for each $\boldsymbol{u}, \boldsymbol{w} \in \mathcal{V}$ and $\alpha \in \mathbb{R}$. Notice that the operations on the left-hand side are in $\mathcal{V}$, whereas the operations on the right-hand side are in $\mathbb{R}$:

1. $\psi(\boldsymbol{u}+\boldsymbol{w}) = \psi(\boldsymbol{u}) + \psi(\boldsymbol{w})$, (addition)
2. $\psi(\alpha\boldsymbol{u}) = \alpha\psi(\boldsymbol{u})$ (product by a scalar)

The vector space $\text{lin}\,(\mathcal{V};\mathbb{R})$ of linear forms on $\mathcal{V}$ is called the *dual space* of $\mathcal{V}$ and is denoted as $\mathcal{V}^*$. The elements of this space are also called *covectors*. It is easy to see that $\mathcal{V}^*$ is isomorphic to $\mathcal{V}$. The duality relationship is symmetric, i.e. $(\mathcal{V}^*)^* = \mathcal{V}$.

Representation theorem Let $\psi \in \mathcal{V}^*$, i.e. let $\psi : \mathcal{V} \to \mathbb{R}$ be a linear form. Then there exists a unique vector $\boldsymbol{a} \in \mathcal{V}$ such that

$$\psi(\boldsymbol{v}) = \boldsymbol{a}\cdot\boldsymbol{v} \tag{3.4}$$

for each vector $\boldsymbol{v} \in \mathcal{V}$. In other words, it is possible to represent a linear mapping $\psi \in \mathcal{V}^*$ by means of the scalar product with its image $\boldsymbol{a} \in \mathcal{V}$ in the space isomorphism $\mathcal{V}^* \to \mathcal{V}$.

Example 3.2.10 (Plane in 3D space)
A plane P in $\mathbb{E}^3$, defined by the Cartesian equation

$$ax + by + cz + d = 0,$$

can be seen as the set of points defined by

$$P = \boldsymbol{o} - \nu^{-1}(d),$$

where $\boldsymbol{o}$ is is the origin of the Cartesian reference system, and ν is a covector in $\text{lin}\,(\mathbb{R}^3;\mathbb{R})$. With a more explicit notation we also have:

$$P = \{\boldsymbol{p} \in \mathbb{E}^3 |\ \nu(\boldsymbol{p}-\boldsymbol{o}) = -d\},$$

where $\nu(\boldsymbol{p}-\boldsymbol{o}) = \boldsymbol{n}\cdot(\boldsymbol{p}-\boldsymbol{o})$, by the representation theorem above. The vector $\boldsymbol{n} = (a,b,c)$ is said to be the *normal vector* to the P plane.

Example 3.2.11
To compute the norm $\|\boldsymbol{v}\|$ of a vector $\boldsymbol{v}$ we need to define a "square" function, denoted as `SQR`, which is not predefined in `PLaSM`. A function `SQRT` for computing the square root is instead predefined. A `UnitVect` function is given to normalize a given non-zero vector. The `ScalarVectProd`, given in Script 3.1.2, computes the scalar product of a scalar times a vector. Some examples of computation are also given in the following script. In particular, the `InnerProd` function is used both prefix and infix.

3.3 Linear transformations and tensors

The concept of tensor is very important for the `PLaSM` language described in this book. In such a language a geometric transformation (e.g. a rotation, a translation, a scaling) of an Euclidean space is just represented as a tensor. Tensors are applied to geometric objects and assemblies as functions; last but not least, pipelines of different transformations can be transformed into a single function by functional composition.

Script 3.2.4

```
DEF SQR = ID * ID;
DEF VectNorm = SQRT ∼ + ∼ AA:Sqr;
DEF UnitVect (v::IsVect) = ScalarVectProd:<1 / VectNorm:v, v>;
DEF InnerProd = + ∼ AA:* ∼ TRANS;

UnitVect:<10,20,30> ≡ < 0.26726124191242434, 0.5345224838248487,
   0.8017837257372731 >
(VectNorm ∼ UnitVect):<10,20,30> ≡ 0.9999999999999999
InnerProd:<<11,12,13>,<4,5,6>> ≡ 182
<11,12,13> InnerProd <4,5,6> ≡ 182
```

Linear transformation A *linear transformation* $\boldsymbol{T} : \mathcal{V}_1 \to \mathcal{V}_2$ is a function between vector spaces that preserves the linear combinations:

$$\boldsymbol{T}(\alpha_1 \boldsymbol{v}_1 + \cdots + \alpha_n \boldsymbol{v}_n) = \alpha_1 \boldsymbol{T}\boldsymbol{v}_1 + \cdots + \alpha_n \boldsymbol{T}\boldsymbol{v}_n.$$

Affine transformation A *affine transformation* $\boldsymbol{T} : \mathbb{E}_1 \to \mathbb{E}_2$ is a function between affine spaces that preserves the affine action:

$$\boldsymbol{T}(\boldsymbol{x} + \alpha(\boldsymbol{y} - \boldsymbol{x})) = \boldsymbol{T}\boldsymbol{x} + \alpha(\boldsymbol{T}\boldsymbol{y} - \boldsymbol{T}\boldsymbol{x})$$

An affine transformation extends naturally to the underlying vector space, by defining

$$\boldsymbol{T}\boldsymbol{v} = \boldsymbol{T}\boldsymbol{y} - \boldsymbol{T}\boldsymbol{x}, \quad \text{where} \quad \boldsymbol{v} = \boldsymbol{y} - \boldsymbol{x}.$$

Tensors The term *tensor* is here used as a synonym of *invertible linear transformation* $\boldsymbol{T}$ from a vector space $\mathcal{V}$ to itself. In other words, a tensor is a linear function $\boldsymbol{T}$ which maps the vector $\boldsymbol{u}$ to the vector $\boldsymbol{T}\boldsymbol{u}$.

The set of all tensors on $\mathcal{V}$ is a vector space, denoted as $\operatorname{lin}\mathcal{V}$, if addition of tensors and product of a tensor by a scalar are defined as:

$$\begin{aligned} (\boldsymbol{S} + \boldsymbol{T})\boldsymbol{v} &= \boldsymbol{S}\boldsymbol{v} + \boldsymbol{T}\boldsymbol{v}, \\ (\alpha \boldsymbol{S})\boldsymbol{v} &= \alpha(\boldsymbol{S}\boldsymbol{v}). \end{aligned}$$

The *null tensor* $\mathbf{0}$ and the *identity tensor* $\boldsymbol{I}$ map each vector $\boldsymbol{v} \in \mathcal{V}$ to the null vector and to itself, respectively:

$$\begin{aligned} \mathbf{0}\boldsymbol{v} &= \mathbf{0}, \\ \boldsymbol{I}\boldsymbol{v} &= \boldsymbol{v}. \end{aligned}$$

3.3.1 Tensor operations

Product The *product* of tensors $\boldsymbol{S}, \boldsymbol{T} \in \operatorname{lin}\mathcal{V}$ is defined as *composition* of functions:

$$\boldsymbol{S}\boldsymbol{T} = \boldsymbol{S} \circ \boldsymbol{T}.$$

Hence $(\boldsymbol{S}\boldsymbol{T})\boldsymbol{v} = \boldsymbol{S}(\boldsymbol{T}\boldsymbol{v})$ for each $\boldsymbol{v} \in \mathcal{V}$.

For the products of a tensor $\boldsymbol{S}$ with itself we have

$$\boldsymbol{S}^2 := \boldsymbol{S}\boldsymbol{S}, \qquad \boldsymbol{S}^3 := \boldsymbol{S}^2\boldsymbol{S}, \qquad \text{etc.}$$

Example 3.3.1 (Tensor product)
In `PLaSM`, invertible linear and affine transformations are represented, in a very natural way, as space automorphism, i.e. as invertible transformations of a space into itself. Therefore the *product* of tensors is given as *composition* of functions. In Script 3.3.1 we show that a rotation with axis parallel to the z axis and passing for $(0.5, 0.5, 0)$ can be espressed as the composition of elementary transformations of translation and rotation about the z axis.

Script 3.3.1 (Tensor product)
```
DEF tensor = T:<1,2>:<0.5,0.5> ~ R:<1,2>:(PI/4) ~ T:<1,2>:<-0.5,-0.5>;

DEF rotatedCube = tensor:(CUBOID:<1,1,1>);

VRML:rotatedCube:'out.wrl'
```

Transpose The *transpose* $\boldsymbol{S}^T$ of $\boldsymbol{S} \in \text{lin}\,\mathcal{V}$ is the unique tensor in $\text{lin}\,\mathcal{V}$ such that, for each $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$,

$$\boldsymbol{S}\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{u} \cdot \boldsymbol{S}^T\boldsymbol{v}$$

The transposition of tensors satisfies the properties:

$$\begin{aligned} (\boldsymbol{S}+\boldsymbol{T})^T &= \boldsymbol{S}^T + \boldsymbol{T}^T, \\ (\boldsymbol{S}\boldsymbol{T})^T &= \boldsymbol{T}^T\boldsymbol{S}^T, \\ (\boldsymbol{S}^T)^T &= \boldsymbol{S}. \end{aligned}$$

Tensor decomposition A tensor $\boldsymbol{X}$ is said *symmetric* if $\boldsymbol{X} = \boldsymbol{X}^T$. It is said to be *skew* if $\boldsymbol{X} = -\boldsymbol{X}^T$. Each tensor $\boldsymbol{X}$ can be expressed *uniquely* as the sum of a symmetric and a skew part:

$$\boldsymbol{X} = \boldsymbol{E} + \boldsymbol{W}$$

where

$$\boldsymbol{E} = \frac{1}{2}(\boldsymbol{X} + \boldsymbol{X}^T), \qquad \boldsymbol{W} = \frac{1}{2}(\boldsymbol{X} - \boldsymbol{X}^T).$$

A `PLaSM` implementation of the tensor decomposition in a symmetric and a skew part is given in Script **??**.

Tensor product of vectors The *tensor product* of two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{V}$ is the tensor $\boldsymbol{a} \otimes \boldsymbol{b} \in \operatorname{lin} \mathcal{V}$ that maps a vector $\boldsymbol{v}$ to the vector $(\boldsymbol{b} \cdot \boldsymbol{v})\boldsymbol{a}$. More formally:

$$\otimes : \mathcal{V}^2 \to \operatorname{lin} \mathcal{V} : (\boldsymbol{a}, \boldsymbol{b}) \mapsto \boldsymbol{a} \otimes \boldsymbol{b}$$

such that, for each $\boldsymbol{v} \in \mathcal{V}$

$$(\boldsymbol{a} \otimes \boldsymbol{b})\boldsymbol{v} = (\boldsymbol{b} \cdot \boldsymbol{v})\boldsymbol{a}.$$

Properties of tensor product of vectors A few important properties of tensor product follow:

1. $(\boldsymbol{a} \otimes \boldsymbol{b})^T = (\boldsymbol{b} \otimes \boldsymbol{a})$;
2. $(\boldsymbol{a} \otimes \boldsymbol{b})(\boldsymbol{c} \otimes \boldsymbol{d}) = (\boldsymbol{b} \cdot \boldsymbol{c})\boldsymbol{a} \otimes \boldsymbol{d}$;
3. $(\boldsymbol{e}_i \otimes \boldsymbol{e}_i)(\boldsymbol{e}_j \otimes e_j) = \begin{matrix} \boldsymbol{0}, & i \neq j, \\ \boldsymbol{e}_i \otimes \boldsymbol{e}_i & i = j; \end{matrix}$
4. $\sum_i \boldsymbol{e}_i \otimes \boldsymbol{e}_i = \boldsymbol{I}$.

Directional decomposition of vectors Let $\boldsymbol{e}, \boldsymbol{v} \in \mathcal{V}$, with $\boldsymbol{e}$ a unit vector. The tensor $\boldsymbol{e} \otimes \boldsymbol{e}$ applied to the vector $\boldsymbol{v}$ gives the projection of $\boldsymbol{v}$ onto the $\boldsymbol{e}$ direction:

$$(\boldsymbol{e} \otimes \boldsymbol{e})\boldsymbol{v} = (\boldsymbol{e} \cdot \boldsymbol{v})\boldsymbol{e}$$

Conversely, the tensor $\boldsymbol{I} - \boldsymbol{e} \otimes \boldsymbol{e}$, when applied to $\boldsymbol{v}$, gives

$$(\boldsymbol{I} - \boldsymbol{e} \otimes \boldsymbol{e})\boldsymbol{v} = \boldsymbol{v} - (\boldsymbol{e} \cdot \boldsymbol{v})\boldsymbol{e},$$

which is the projection of $\boldsymbol{v}$ onto the linear subspace orthogonal to $\boldsymbol{e}$.

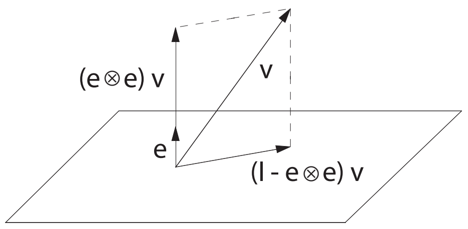

Figure 3.6 Directional and orthogonal projections of a vector

Example 3.3.2 (Directional decomposition)
The directional decomposition of a vector is implemented in Script 3.3.2, where a dimension-independent solution is given, according to the true nature of the `PLaSM` language. The `VectDiff`, `VectSum` and `ScalarVectProd` operations are given in Script 3.1.2. The `InnerProd` and `UnitVect` operations are defined in Script 3.2.4. The last `PLaSM` expression aims to show that the vector addition of the two generated vector projections returns the original vector.

Script 3.3.2

```
DEF DirProject (e::Isvect)(v::Isvect) =
   (UnitVect:e InnerProd v) ScalarVectProd UnitVect:e;
DEF OrthoProject (e::Isvect)(v::Isvect) =
   v VectDiff DirProject:(UnitVect:e):v;

DirProject:<1,1,0,0>:<10,15,20,25>   ≡ <12.5, 12.5, 0.0, 0.0>;
OrthoProject:<1,1,0,0>:<10,15,20,25> ≡ <-2.5, 2.5, 20.0, 25.0>;

(VectSum ~ [DirProject:<1,1,0,0>, OrthoProject:<1,1,0,0>]):<10,15,20,25>
   ≡ <10.0, 15.0, 20.0, 25.0>;
```

3.3.2 Coordinate representation

When an orthonormal basis $(\boldsymbol{e}_i)$ in $\mathcal{V}$ is given, the components S_{ij} of a tensor $\boldsymbol{S} \in \operatorname{lin}\mathcal{V}$ are defined as

$$S_{ij} = \boldsymbol{e}_i \cdot \boldsymbol{S}\boldsymbol{e}_j.$$

By this definition, the vector $\boldsymbol{v} = \boldsymbol{S}\boldsymbol{u}$ can be represented component-wise with respect to the $(\boldsymbol{e}_i)$ basis as (v_i), with

$$v_i = \sum_j S_{ij} u_j.$$

As a matter of fact, by the expression of the component of $\boldsymbol{v}$ with respect to the basis and by the linearity of both the tensor $\boldsymbol{S}$ and the scalar product, we have

$$\begin{aligned} v_i &= \boldsymbol{e}_i \cdot \boldsymbol{v} \;=\; \boldsymbol{e}_i \cdot \boldsymbol{S}\boldsymbol{u} \\ &= \boldsymbol{e}_i \cdot \boldsymbol{S} \sum_j (\boldsymbol{e}_j \cdot \boldsymbol{u}) \boldsymbol{e}_j \;=\; \boldsymbol{e}_i \cdot \boldsymbol{S} \sum_j u_j \boldsymbol{e}_j \\ &= \sum_j u_j \left(\boldsymbol{e}_i \cdot \boldsymbol{S}\boldsymbol{e}_j\right) \;=\; \sum_j S_{ij} u_j. \end{aligned}$$

It is also easy to give a coordinate decomposition of a tensor $\boldsymbol{S}$ as a combination of tensor components S_{ij} with the basis $(\boldsymbol{e}_i \otimes \boldsymbol{e}_j)$ of $\operatorname{lin}\mathcal{V}$ induced by the basis $(\boldsymbol{e}_i)$ of $\mathcal{V}$:

$$\boldsymbol{S} = \sum_{ij} S_{ij} \boldsymbol{e}_i \otimes \boldsymbol{e}_j$$

as well as to give the components of a tensor product of vectors

$$(\boldsymbol{a} \otimes \boldsymbol{b})_{ij} = a_i b_j. \tag{3.5}$$

equation 3.5 will be useful when deriving the tensor product expression of a surface in parametric form (see Section 12.3).

Matrix of a tensor The components of a tensor $\boldsymbol{S} \in \operatorname{lin}\mathcal{V}$ can be assembled in a square matrix, denotated as $[\boldsymbol{S}]$. If $\mathcal{V} = \mathbb{R}^d$, then we have

$$[\boldsymbol{S}] = \begin{bmatrix} S_{11} & S_{12} & \cdots & S_{1d} \\ S_{21} & S_{22} & \cdots & S_{2d} \\ \cdots & \cdots & \cdots & \cdots \\ S_{d1} & S_{d2} & \cdots & S_{dd} \end{bmatrix}$$

A matrix $A = (a_{ij})$ is *transposed* when its rows are interchanged with its columns, and vice versa. The transposed matrix is denoted as $A^T = (a_{ji})$.

A few useful properties of tensor matrices follow:

1. $[\boldsymbol{S}^T] = [\boldsymbol{S}]^T$ (transposition)
2. $[\boldsymbol{ST}] = [\boldsymbol{S}][\boldsymbol{T}]$ (product)
3. $[\boldsymbol{I}] = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$ (identity)

3.3.3 `PLaSM` *matrix representation*

A matrix in $\mathcal{M}_m^n$ is represented in `PLaSM` as a sequence of n sequences of length m. In other words, a matrix is represented by rows. A predicate `IsMat` is therefore given to verify if a data structure is a sequence of sequences of either real numbers or functions. The predicate also tests if all the component elements (rows) have the same length (number of columns). The predicates `IsRealVect` e `IsFunVect`, which test if an object is a sequence of reals or functions, respectively, were defined in Script 3.1.1.

Script 3.3.3 (`IsMat` predicate)

```
DEF IsMat = AND ∼ [OR ∼ [IsSeqOf:IsRealVect,
                         IsSeqOf:IsFunVect],
                   EQ ∼ AA:LEN];
```

Example 3.3.3 (`PLaSM` matrix)
A matrix $A \in \mathbb{R}_3^3$ can be represented by a `PLaSM` symbol (name), as shown in Script 3.3.4.

Script 3.3.4

```
DEF A = <
   < 1.0, 2.0, 3.0 >,
   < 4.0, 5.0, 6.0 >,
   < 7.0, 8.0, 9.0 >>;

IsMat:A ≡ TRUE
```

For example, the identity matrix $[\boldsymbol{I}] \in \mathbb{R}_3^3$ may be defined as:

```
DEF Identity = <<1, 0, 0>, <0, 1, 0>, <0, 0, 1>>
```

A matrix can also be directly used without being named, as we show in several examples in the remainder of this chapter.

Example 3.3.4 (Identity matrix)
A function `IDNT` is defined here. This function, when applied to an integer n, returns the identity matrix $\boldsymbol{I} \in \mathbb{R}_n^n$. The function `IDNT` is defined in pure `FL` style, using

neither iteration nor recursion.

We utilize the following algorithmic scheme. First, the set $N^2 = N \times N$ is built, where $N = \{1, \ldots, n\}$. Then we apply to each pair $(i, j) \in N^2$ the Kröneker's function, defined as

$$\delta : N^2 \to \{0,1\} : (i,j) \mapsto \begin{matrix} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{matrix}$$

which returns either 0 or 1 depending on the identity of input pairs. The `cart` operator, that executes the Cartesian product of finite sets, is given in Script 2.1.17.

Script 3.3.5 (Identity matrix (1))

```
DEF IDNT (n::IsIntPos) = (AA∼AA):Kroneker:(cart:<1..n,1..n>)
WHERE
   Kroneker = IF:<EQ, K:1, K:0>
END

IDNT:4 ≡ <
   <1, 0, 0, 0>,
   <0, 1, 0, 0>,
   <0, 0, 1, 0>,
   <0, 0, 0, 1> >
```

A simpler implementation of the `IDNT` operator is given in Script 3.3.6, where the strong connection between the vertices of a simplex (see Section 4.5) and the identity matrix as a set of unit vectors is exploited. The `RTAIL` function, which returns the elements of a sequence except the last one, is given in Script 4.2.5.

Script 3.3.6 (Identity matrix (2))

```
DEF IDNT (n::IsIntPos) = (RTAIL ∼ S1 ∼ UKPOL ∼ SIMPLEX):n;
```

Predefined matrix operators Some predefined matrix operators are available in `PLaSM`. In particular, the `TRANS` function is a unary operator that returns the transpose when applied to a matrix. Analogously, the `INV` unary operator returns the inverse of its argument when applied to an invertible square matrix. Otherwise it returns an exception.

```
TRANS:<<1,2,3>,<4,5,6>> ≡ <<1, 4>, <2, 5>, <3, 6>>

INV:<<2,0,0>,<0,2,0>,<0,0,2>> ≡
   <<0.5, 0.0, 0.0>, <0.0, 0.5, 0.0>, <0.0, 0.0, 0.5>>
```

Predefined binary operators are available for *addition*, *difference* and *product* of compatible matrices.

Let $A = (a_{ij})$ and $B = (b_{ij})$ in $\mathbb{R}^{\ell}_{m}$, and $C = (c_{ij})$ in $\mathbb{R}^{m}_{n}$. The binary matrix operations have the standard component-wise meaning

$$A + B = (a_{ij} + b_{ij}), \qquad A - B = (a_{ij} - b_{ij}), \qquad BC = (\sum_{k=1}^{m} b_{ik}c_{kj})$$

Such operations are denoted in `PLaSM` with the symbols `+`, `-` and `*`, respectively. As always in the language, a binary operator can be used as either prefix or infix, as shown in the following examples. The function `IDNT`, when applied to an integer n, returns the identity matrix $[\boldsymbol{I}] \in \mathbb{R}^n_n$. This function is not primitive in `PLaSM`. It is defined in Script 3.3.5. Notice that predefined algebraic operators are implicitly lifted (see below) when infix between functions: i.e.:

```
IDNT + IDNT ≡ + ∼ [IDNT, IDNT]
```

Function lifting and raising `RAISE` and `LIFT` are two primitive `FL` combining forms defined by the following equational semantics. The `RAISE` combining form is used to allow overloaded use of operators over both numbers and functions. For example, the `PLaSM` algebraic operators like "`+`" and "`*`", are actually *raised*-up versions of their numeric counterparts.

Script 3.3.7 (`LIFT` and `RAISE` combining forms)

```
LIFT:f:<f₁,... ,fₙ> ≡ f ∼ [f₁,... ,fₙ]

RAISE:f:seq ≡ IF:<IsSeqOf:IsFun, LIFT:f, f>:seq
```

Example 3.3.5 (Matrix expressions)
Some interesting examples of calculus at function level with matrices are given in Script 3.3.8. For example, the $\mathbf{0}$ matrix $n \times n$ can be generated by applying the function `IDNT - IDNT` to an arbitrary positive integer n.

Script 3.3.8 (Matrix calculus examples)

```
(IDNT - IDNT):3 ≡ <<0.0, 0.0, 0.0>,<0.0, 0.0, 0.0>,<0.0, 0.0, 0.0>>
(IDNT + IDNT):3 ≡ <<2.0, 0.0, 0.0>,<0.0, 2.0, 0.0>,<0.0, 0.0, 2.0>>

<<1,2,3>,<4,5,6>,<7,8,9>> * <<1,2,3>,<4,5,6>,<7,8,9>> ≡ <
   <30.0, 36.0, 42.0>,
   <66.0, 81.0, 96.0>,
   <102.0, 126.0, 150.0>>

<<1,2,3>,<4,5,6>> * TRANS:<<1,2,3>,<4,5,6>> ≡ <
   <14.0, 32.0>,
   <32.0, 77.0>>
```

Both matrix operators and data can be freely combined in writing matrix expressions, like in the following example, where `A` is the matrix defined in Example 3.3.3:

```
(TRANS ∼ INV):(IDNT:3 - A) ≡ <
   <-0.5, 0.3125, 0.125>,
   <0.25, -0.65625, 0.4375>,
   <0.0, 0.375, -0.25> >
```

Example 3.3.6 (Scalar multiple of a matrix)
Let us define a function `ScalarMatProd` which computes the product of a scalar $\alpha \in \mathbb{R}$ times a matrix $A = (a_{ij}) \in \mathbb{R}^n_n$, for the arbitrary positive integer n. The problem can be approached by first computing a matrix of pairs (α, a_{ij}) and then by applying to each pair the operation of product between numbers. We use here a pure `FL` style, i.e. without formal parameters (see Section 2.1.1). `ScalarMatProd` can be used as either infix or prefix, but notice that the first argument must be the scalar one.

Script 3.3.9

```
DEF ScalarMatProd = (AA∼AA):* ∼ AA:DISTL ∼ DISTL;

9 ScalarMatProd IDNT:3 ≡
   <<9, 0, 0>,
    <0, 9, 0>,
    <0, 0, 9>>;
```

Trace of tensor The *trace* is the only linear form

$$tr : \text{lin}\ \mathcal{V} \to \mathbb{R}$$

satisfying

$$tr\ (\boldsymbol{u} \otimes \boldsymbol{v}) = \boldsymbol{u} \cdot \boldsymbol{v}$$

for each $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$.

The scalar $tr\,\boldsymbol{S}$ is easy to compute starting from the matrix $[\boldsymbol{S}]$. In fact, for the linearity of tr we have:

$$\begin{aligned} tr\ \boldsymbol{S} &= tr\ \left(\sum_{i,j} S_{ij} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \right) \\ &= \sum_{i,j} S_{ij} tr(\boldsymbol{e}_i \otimes \boldsymbol{e}_j) \\ &= \sum_{i,j} S_{ij} \boldsymbol{e}_i \cdot \boldsymbol{e}_j \ = \ \sum_i S_{ii} \end{aligned}$$

Henceforth, the trace of a square matrix is defined as the sum of diagonal elements. It is possible to verify that:

$$\begin{aligned} tr\ \boldsymbol{S}^T &= tr\ \boldsymbol{S}, \\ tr\ (\boldsymbol{ST}) &= tr\ (\boldsymbol{TS}). \end{aligned}$$

Example 3.3.7 (Trace computation)
A function `Trace` is here defined which computes the trace of a matrix $A \in \mathbb{R}^n_n$, for the arbitrary positive integer n. The algorithm coded by the function can be described as follows:

1. Compute the number n of matrix rows.
2. Generate the integer sequence from 1 to n.
3. Produce the sequence of pairs (i, A_i), where A_i is the i-th row of A.
4. Apply to all pairs a function which select the a_{ii} element.
5. Compute the sum of all selected elements.

Script 3.3.10

```
DEF Trace (matrix::IsMat) = (+ ∼ AA:select ∼ TRANS):< 1..n, matrix >
WHERE
   n = LEN:matrix,
   select = APPLY ∼ [SEL∼S1, S2]
END;

Trace:<<1,2,3>,<4,5,6>,<7,8,9>> ≡ 15
```

Inner product of tensors The space lin $\mathcal{V}$ of tensors on the $\mathcal{V}$ vector space has a natural *inner product*, that is defined as:

$$\boldsymbol{S} \cdot \boldsymbol{T} = tr\ (\boldsymbol{S}^T \boldsymbol{T}),$$

which component-wise becomes:

$$\boldsymbol{S} \cdot \boldsymbol{T} = \sum_{i,j} S_{ij} T_{ij}.$$

in the following example we extend this operations to compatible matrices.

Example 3.3.8 (Inner product of matrices)
The goal is to add all the products of corresponding elements in the argument matrices. This is obtained by the algorithm:

1. Generate a sequence of pairs of corresponding rows.
2. Transpose all such pairs of rows.
3. Catenate the result, so generating a single sequence of pairs of numbers.
4. Apply to all number pairs the product operator.
5. Add all the resulting numbers.

The function `matDotProd` computes the dot product of any two compatible matrices with any number of rows and columns, according to the algorithm described above. Script 3.3.11 also reports a step-wise example of computation, by showing the partial elaborations of an input instance.

Script 3.3.11

```
DEF matDotProd = + ∼ AA:* ∼ CAT ∼ AA:TRANS ∼ TRANS;
DEF A = <<1,2>,<3,4>,<5,6>>;
DEF B = <<10,20>,<30,40>,<50,60>>;

matDotProd:< A, B > ≡ 910
```

3.3.4 Determinant and inverse

Determinant of a matrix A *permutation* is a one-to-one function on a finite set. A permutation $\pi : \{1, 2, \dots, n\} \to \{1, 2, \dots, n\}$ is usually denoted as

$$\pi = \begin{pmatrix} 1 & 2 & \cdots & n \\ \pi(1) & \pi(2) & \cdots & \pi(n) \end{pmatrix}.$$

The *determinant* of a square number matrix A is a number $\det A$ such that

$$\begin{aligned} \det A &= \det \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{vmatrix} \\ &= \sum_{\pi} \operatorname{sign}(\pi) a_{1\pi(1)} a_{2\pi(2)} \cdots a_{n\pi(n)} \end{aligned}$$

where the sum is taken over all the $n!$ permutations π of $\{1, \dots, n\}$, and $\operatorname{sign}(\pi)$ is either $+1$ or -1 according as π is either an even or odd permutation. Notice that each product $a_{1\pi(1)} a_{2\pi(2)} \cdots a_{n\pi(n)}$ contains n matrix elements such that no two of them belong to the same row or to the same column.

Determinant of a tensor The determinant $det : \text{lin}\ \mathcal{V} \to \mathbb{R}$ of a tensor $\boldsymbol{S}$ is defined as the determinant of its matrix $[\boldsymbol{S}]$:

$$det\ \boldsymbol{S} = det\ [\boldsymbol{S}].$$

A tensor $\boldsymbol{S}$ is *invertible* if there exists a tensor $\boldsymbol{S}^{-1}$, called the *inverse* of $\boldsymbol{S}$, such that:

$$\boldsymbol{S}\boldsymbol{S}^{-1} = \boldsymbol{S}^{-1}\boldsymbol{S} = \boldsymbol{I}.$$

$\boldsymbol{S}$ is invertible if and only if $det\ \boldsymbol{S} \neq 0$

Some properties of determinants and inverse tensors follow:

1. $det\ (\boldsymbol{S}\boldsymbol{T}) = (det\ \boldsymbol{S})(det\ \boldsymbol{T})$
2. $det\ \boldsymbol{S}^T = det\ \boldsymbol{S}$
3. $det\ (\boldsymbol{S}^{-1}) = (det\ \boldsymbol{S})^{-1}$
4. $(\boldsymbol{S}\boldsymbol{T})^{-1} = \boldsymbol{T}^{-1}\boldsymbol{S}^{-1}$
5. $(\boldsymbol{S}^{-1})^T = (\boldsymbol{S}^T)^{-1}$

3.3.5 Orthogonal tensors

A tensor $\boldsymbol{Q} \in \text{lin}\,\mathcal{V}$ is said to be *orthogonal* if it preserves the inner products, i.e. if for each $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$:

$$\boldsymbol{Qu} \cdot \boldsymbol{Qv} = \boldsymbol{u} \cdot \boldsymbol{v}.$$

A necessary and sufficient condition for $\boldsymbol{Q}$ be orthogonal is that

$$\boldsymbol{Q}\boldsymbol{Q}^T = \boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}.$$

By properties of tensor matrices, we have in this case:

$$[\boldsymbol{Q}][\boldsymbol{Q}]^T = [\boldsymbol{Q}]^T[\boldsymbol{Q}] = [\boldsymbol{I}]$$

For this reason, a matrix A such that $AA^T = A^TA = I$ is also said *orthogonal.*

Example 3.3.9
We show that the one-parameter family of matrices $\boldsymbol{M} : \mathbb{R} \to R_2^2$ defined as

$$\boldsymbol{M}(u) = \begin{pmatrix} \cos u & -\sin u \\ \sin u & \cos u \end{pmatrix}$$

is orthogonal for each $u \in \mathbb{R}$:

$$\begin{aligned} \boldsymbol{M}(u)\boldsymbol{M}^T(u) &= \begin{pmatrix} \cos u & -\sin u \\ \sin u & \cos u \end{pmatrix}\begin{pmatrix} \cos u & \sin u \\ -\sin u & \cos u \end{pmatrix} \\ &= \begin{pmatrix} \cos^2 u + \sin^2 u & 0 \\ 0 & \cos^2 u + \sin^2 u \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned}$$

We also define the `M` function, and then execute various computations showing the orthogonality of $\boldsymbol{M}(u)$ for different values of u. Remember that `PI` is the `PLaSM` denotation for π.

Script 3.3.12

```
DEF M (u::IsReal) = [[cos, -∼sin],[sin, cos]]:u;
M:(PI/6)
   ≡ <<0.8660254037844387, -0.5>,
      <0.5, 0.8660254037844387>>
M:1
   ≡ <<0.5403023058681398, -0.8414709848078965>,
      <0.8414709848078965, 0.5403023058681398>>
(M * (TRANS∼M)):(PI/6)
   ≡ <<1.0, 0.0>, <0.0, 1.0>>
(M * (TRANS∼M)):1
   ≡ <<1.0, 0.0>, <0.0, 1.0>>
```

3.3.6 `PLaSM` *representation of tensors*

Tensors are represented in `PLaSM` by applying the predefinite function `MAT` to the matrix of the tensor. This is clearly possible when a basis (e_i) has been given for the underlying vector space. The reference frame for a Euclidean space is usually Cartesian. For example, we may define two tensors `X` and `Y` as follows:

```
DEF X = MAT:<<0,0,1>,<1,0,1>,<0,2,1>>;
DEF Y = MAT:<<0,1,1>,<,0,1>,<1,1,1>>;
```

Tensors defined as vector space endomorphisms have a first-class citizenship in `PLaSM`, so that they can be composed by generating new tensors. For instance:

```
DEF Z = Y ∼ X
```

Application of `PLaSM` tensors Tensors can be applied to polyhedral complexes[3] of generic dimension (d, n), where d is the intric dimension of the complex (e.g. $d = 1$ for curves, $d = 2$ for surfaces, $d = 3$ for solids, etc.) and n is the dimension of the embedding space, i.e. the number of point coordinates.

For this purpose, it is necessary to use a normalized homogeneous representation of the affine space $\mathbb{E}^n$, using $n + 1$ coordinates with the first one equal to 1. This allows the user, as we discuss in following chapters, and in particular in Section 6.1.4, to treat any affine mapping, including translations, as a linear mapping. For this purpose, see also Section 4.2.1. Two affine maps can be combined by matrix product of their matrix representations.

Hence we introduce here a useful function `MatHom` (for "Matrix Homogeneize"), which makes a square matrix homogeneous and normalized. This function, when applied to an actual parameter $M \in \mathbb{R}^n_n$, returns the matrix M' obtained by embedding M in the identity $I \in \mathbb{R}^{n+1}_{n+1}$, so that both the first column and the first row contain the unit vector.

Script 3.3.13

```
DEF IsSqrMat = AND ∼ [IsMat, EQ ∼ [LEN, LEN ∼ s1]];
DEF MatHom (m::IsSqrMat) = AL:< firstRow, (AA:AL ∼ DISTL):<0,m> >
WHERE
   firstRow = AL:<1,#:(LEN:m):0>
END;

MatHom:<<1,2,3>,<4,5,6>,<7,8,9>> ≡
   <<1, 0, 0, 0>,
    <0, 1, 2, 3>,
    <0, 4, 5, 6>,
    <0, 7, 8, 9>>
```

Both the `MatHom` definition and the result of its application to a 3×3 matrix are given in Script 3.3.13. Remember that the predefined function `AL` ("Append Left"),

[3] See Sections 4.6 and 14.1.

to be applied to pairs constituted by any element and by any sequence, appends the element as the first one of the output sequence.

Tensors in PLaSM can be applied only to expressions which denote polyhedral complexes. The evaluation of such applications return the polyhedral complex after the transformation of coordinates.

Script 3.3.14 (User-defined PLaSM tensors)

```
DEF Tensor = MAT ∼ MatHom;
DEF X = (Tensor ∼ TRANS):<<0,1,0>, <0,0,2>, <1,1,1>>;
DEF Y = (Tensor ∼ INV ∼ TRANS):<<0,1,0>, <0,0,2>, <1,1,1>>;
DEF Cube = CUBOID:<1,1,1>;

(STRUCT ∼ [ID, X]):Cube;
(STRUCT ∼ [ID, Y]):Cube;
(STRUCT ∼ [Y∼X, X∼Y]):Cube;
```

Application of tensors to points As we have seen, PLaSM tensors can be applied only to polyhedral complexes. But it is also possible to apply a tensor to a point or to a discrete point set in $\mathbb{E}^d$, by transforming the argument into a polyhedral complex of dimension $(0, d)$, and by transforming back to points the complex mapped by the tensor.

Hence we give in Script 3.3.15 both the definition of two operators which execute the object conversion in both-ways, and an example of application of a tensor to either a point or to a set of points.

In particular, the MK (MaKe) function transforms a point in $\mathbb{E}^d$, i.e. a sequence of d coordinates, into a polyhedron of intrinsic dimension 0. The UK (UnmaKe) function performs the inverse transformation. The following expressions show:

1. that UK∼MK equates the identity tensor (plus a casting to real coordinates);
2. how to apply an X tensor to a point (where X is defined in Script 3.3.14);
3. the type of the result of a MK application;
4. how to look for intrinsic and embedding dimensions of a polyhedral complex;
5. how to transform a sequence of d-points into a 0-dimensional polyhedral complex in $\mathbb{E}^d$.

Script 3.3.15 (Point→polyhedron and vice versa)

```
DEF MK = MKPOL∼[LIST,K:<<1>>,K:<<1>>];
DEF UK = S1∼S1∼UKPOL;

(UK∼MK):<0,0,0>      ≡ <0.0, 0.0, 0.0>
(UK∼X∼MK):<0,0,0>    ≡ <0.5, 1.0, 0.0>
MK:<0,1,2>           ≡ A-Polyhedral-Complex{0,3}
[DIM,RN]:(MK:<0,1,2>) ≡ <0, 3>
(STRUCT∼AA:MK):<<0,1,1>,<0.5,0,2>,<1,4,0>> ≡ A-Polyhedral-Complex{0,3}
```