4

Elements of polyhedral geometry

This chapter reviews the algebraic and geometric concepts which underlie the geometric kernel of the `PLaSM` language. The aim is both to collect concepts and tools from different fields, and to provide the reader with key ideas useful in understanding the behavior of basic operators of the language. For this purpose, the chapter contains some materials about convexity theory, double representation of polyhedral sets, polarity, the boundary structure of a polytope, simplicial and polyhedral complexes, Nef polyhedra and linear programming. At the same time, polyhedral geometry offers exciting opportunities to explore a programming language properly designed to support geometric calculus. The straighforward generation of d-dimensional permutahedra and the construction of Platonic solids inscribed in a unit sphere provide convincing examples.

4.1 Basic concepts

Let us first recall some basic topological concepts like *interior*, *boundary* and *closure*, and the basic elements, like *halfplanes*, *halfspaces* and *flats*, of the affine structure of Euclidean spaces.

Topology A *topology* on a set W is a family $\mathcal{T}$ of subsets, called *open sets*, of W such that:

1. any union of elements of $\mathcal{T}$ belongs to $\mathcal{T}$;
2. the intersection of any two elements of $\mathcal{T}$ belongs to $\mathcal{T}$;
3. $\emptyset$ and W belong to $\mathcal{T}$.

The pair $(W, \mathcal{T})$ is called a *topological space*. When no ambiguity arises about the chosen topology, W itself is called a topological space. A *metric* topology on W is the set of open balls centered at each W element, whose points have distance from the center less than the radius. The *natural* topology, i.e. the metric topology induced by the Euclidean distance, is assumed when considering $\mathbb{E}^n$ as a topological space.

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

Boundary, interior, closure Let W be a topological space and $\boldsymbol{x} \in W$. A subset $S \subset W$ is a *neighborhood* of $\boldsymbol{x}$ if there exists an open set V in the topology of W such that $\boldsymbol{x} \in V \subset S$.

A point $\boldsymbol{x} \in W$ is a boundary point of S if neither S nor $W \backslash S$ is a neighborhood of $\boldsymbol{x}$. The *boundary* ∂S is the set of boundary points of S. A point $\boldsymbol{x} \in W$ is an interior point of S if S is a neighborhood of $\boldsymbol{x}$. The *interior* $\operatorname{int} S$ is the set of interior points of S. The *closure* of S is the set $\operatorname{clos} S := S \cup \partial S$. S is said to be *closed* (*open*) if and only if $S = \operatorname{clos} S$ (respectively, $S = \operatorname{int} S$).

A set $A \subset \mathbb{E}^n$ is *relatively closed* (relatively open) if it is closed (open) with respect to the *subspace topology* induced on $\operatorname{aff} A$ by the natural topology of $\mathbb{E}^n$. Analogously, the *relative interior* $\operatorname{relint} A$ is the set of interior points of A with respect to the subspace topology of $\operatorname{aff} A$.

The set A is *bounded* when there exists $\kappa \in \mathbb{R}$ such that $\|\boldsymbol{x} - \boldsymbol{y}\| < \kappa$, for each $\boldsymbol{x}, \boldsymbol{y} \in A$. Also, the set A is said to be *compact* if it is bounded and closed.

Segments A *closed segment* $[\boldsymbol{x}, \boldsymbol{y}]$, with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{E}^n$, is defined as:

$$[\boldsymbol{x}, \boldsymbol{y}] := \{(1 - \lambda)\boldsymbol{x} + \lambda \boldsymbol{y} | 0 \leq \lambda \leq 1\}.$$

An *open* segment, and a right (left) *semi-closed* segment, denoted as $(\boldsymbol{x}, \boldsymbol{y})$, $[\boldsymbol{x}, \boldsymbol{y})$ and $(\boldsymbol{x}, \boldsymbol{y}]$ respectively, correspond to $0 < \lambda < 1$, to $0 \leq \lambda < 1$, and to $0 < \lambda \leq 1$. Notice that open, closed and semi-closed segments are well defined even with coinciding extremes. E.g.: $(\boldsymbol{x}, \boldsymbol{x}) = [\boldsymbol{x}, \boldsymbol{x}] = \{\boldsymbol{x}\}$.

Set operations Minkowski addition of sets $A, B \subset \mathbb{E}^n$ and Minkowski product of a set times a scalar $\lambda \in \mathbb{R}$ are defined respectively as:

$$\begin{aligned} A + B &:= \{a + b | a \in A, b \in B\}, \\ \lambda A &:= \{\lambda a | a \in A\}. \end{aligned}$$

An algorithm for Minkowski addition of a polyhedral complex with a special class of convex sets and its `PLaSM` implementation are provided in Section 14.6. This operation is used for translational *motion planning* of a robot moving amidst obstacles [LPW79]. See Section 15.6.4 for a description.

Hyperplanes and half-spaces A *hyperplane* of $\mathbb{E}^n$ may be written as

$$H_{\boldsymbol{u},\alpha} := \{\boldsymbol{x} \in \mathbb{E}^n | \boldsymbol{u}^T \boldsymbol{x} = \alpha\},$$

where $\boldsymbol{u} \in \mathbb{E}^n \backslash \{\boldsymbol{o}\}$ is the *normal vector* of $H_{\boldsymbol{u},\alpha}$ and $\alpha \in \mathbb{R}$.

A hyperplane may have several equivalent representations. In particular:

$$H_{\boldsymbol{u},\alpha} = H_{\boldsymbol{v},\beta}$$

if and only if $(\boldsymbol{v}, \beta) = (\lambda \boldsymbol{u}, \lambda \alpha)$, with $\lambda \in \mathbb{R}$. It follows that $H_{\boldsymbol{u},\alpha} = H_{\boldsymbol{u}/|\boldsymbol{u}|,\alpha/|\boldsymbol{u}|}$, where $\alpha/|\boldsymbol{u}|$ is the *signed distance* of $H_{\boldsymbol{u},\alpha}$ from the origin.

A hyperplane $H_{\boldsymbol{u},\alpha}$ is a translate of a parallel linear subspace $H_{\boldsymbol{u},0}$, i.e. is an affine subspace:

$$H_{\boldsymbol{u},\alpha} = H_{\boldsymbol{u},0} + \frac{\alpha}{|\boldsymbol{u}|}\frac{\boldsymbol{u}}{|\boldsymbol{u}|} = H_{\boldsymbol{u},0} + \frac{\alpha}{\boldsymbol{u} \cdot \boldsymbol{u}}\boldsymbol{u}.$$

The *dimension* of $H_{\boldsymbol{u},\alpha}$ is the cardinality of a minimal subset of $\mathbb{E}^n$ which spans $H_{\boldsymbol{u},0}$. A hyperplane $H_{\boldsymbol{u},\alpha}$ subdivides the space E^n into two *closed halfspaces*

$$\begin{aligned} H^-_{\boldsymbol{u},\alpha} &:= \{\boldsymbol{x} \in \mathbb{E}^n | \boldsymbol{u}^T\boldsymbol{x} \leq \alpha\}, \\ H^+_{\boldsymbol{u},\alpha} &:= \{\boldsymbol{x} \in \mathbb{E}^n | \boldsymbol{u}^T\boldsymbol{x} \geq \alpha\}, \end{aligned}$$

which will be occasionally named *below* and *above* halfspace, respectively.

Flats and half-flats An affine subspace is also called a *flat*. A flat may be always represented as an intersection of hyperplanes. The intersection of an halfspace with a flat not completely contained in it is called *half-flat*. A *line* is a flat of dimension 1. A *ray* is a half-flat of dimension 1.

4.2 Convex sets

Convex sets are the main ingredient of our language for geometric design, since each geometric value in `PLaSM` is a covering of a compact point set with convex cells of appropriate dimension. It is hence very useful to reserve special attention to the theory of convex sets.

4.2.1 Positive, affine and convex hulls

Here we recall some concepts quickly introduced in Chapter 3, in order to make the present chapter more self-contained. In the remainder we do not distinguish between a Euclidean space as a set of points and the underlying vector space. Anyway, the terms vector and point are appropriately used when useful.

Positive hulls and cones A *positive* (or *conical*) *combination* of a set of vectors $\{\boldsymbol{x}_i | i \in I\}$ is a linear combination with scalars $\alpha_i \geq 0, i \in I$.

A *cone* is a set $C \subseteq \mathbb{E}^n$ which is closed with respect to the positive combination of every subset of elements.

Positive (or *conical*) *hull* of a set $S \subset \mathbb{E}^n$ is the set $\operatorname{cone} S$ of all positive combinations of S elements. The set $\operatorname{cone} S$ is the smallest cone which contains S. Since every positive hull contains the zero vector, it is customary to define $\operatorname{cone} \emptyset = \{\mathbf{0}\}$.

Example 4.2.1 (Finite cone generated by a convex set)
In Script 4.2.1 the `FiniteCone` function generates the set $\operatorname{conv} \mathtt{A} \cup \{\mathbf{0}\}$, where `A` is any geometric value generated by `PLaSM`. The result of application to a translated cube skeleton is shown in Figure 4.1. Clearly, the set $(\mathtt{Cone:A}) \cup \lambda\,(\mathtt{Cone:A})$, $\lambda > 1$, is a cone . We call the set `Cone:A` a *finite cone*. Notice that $\mathtt{FiniteCone:A} \equiv \lambda\,\mathtt{A}$, $\lambda \in [0, 1]$.

The `FiniteCone` generator function, depending on a formal parameter `pol` of polyhedral complex type, is produced by the `JOIN` of its argument $\mathtt{pol} \in \mathcal{P}^{d,n}$ with the convex set $\{\mathbf{0}\}$, where $\mathbf{0} \in \mathbb{R}^n$. Let us remember that the built-in geometric operator `RN` returns the embedding dimension of the complex to which it is applied. Notice that both the `MATERIAL` and the `STRUCT` primitives are used infix within their arguments. To export the `out` object will require a preliminary loading of the `'colors'` library.

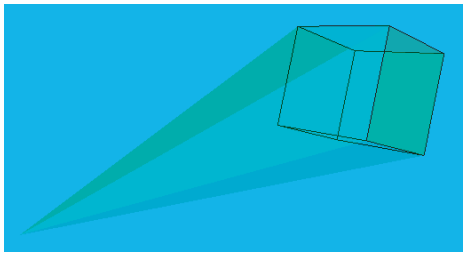

Figure 4.1 Finite cone

Script 4.2.1 (Cone generator)

```
DEF FiniteCone (pol::IsPol) = pol JOIN (MK ∼ #:(RN:pol)):0;
DEF complex = (T:<1,2,3>:<1,2,3> ∼ @1 ∼ CUBOID):<1,1,1>;

DEF out = FiniteCone:complex
   MATERIAL Transparentmaterial:<GREEN, 0.8>
   STRUCT complex
```

Affine hulls and affine spaces An *affine combination* of a set of points $\{\boldsymbol{x}_i | i \in I\}$ is a linear combination with scalars α_i such that $\sum_{i\in I} \alpha_i = 1$.

The *affine hull* of a set $S \subset \mathbb{E}^n$ is the set $\operatorname{aff} S$ of all affine combinations of elements of S. Every affine hull, called also affine space, is the translate of a linear space:

$$\operatorname{aff}\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\} = \boldsymbol{x}_0 + \operatorname{lin}\{\boldsymbol{x}_1 - \boldsymbol{x}_0, \ldots, \boldsymbol{x}_d - \boldsymbol{x}_0\}$$

Let $S \subset \mathbb{E}^n$, and $x \in S$. Then $\dim \operatorname{aff} S = \dim \operatorname{lin}(S - \boldsymbol{x})$.

A non-empty subset $U \subset \mathbb{E}^n$ is an *affine space* if and only if there exist suitable $\boldsymbol{A} \in \mathbb{R}_n^m$ and $\boldsymbol{b} \in \mathbb{R}^m$ such that

$$U = \{\boldsymbol{x} \in \mathbb{E}^n | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}.$$

Convex hulls and convex sets A *convex combination* of a set of points $\{\boldsymbol{x}_i | i \in I\}$ is a linear combination which is both positive and affine, i.e. with scalars $\alpha_i \geq 0$ such that $\sum_{i\in I} \alpha_i = 1$.

The *convex hull* of a set S is the set $\operatorname{conv} S$ of all convex combinations of S elements. The set $\operatorname{conv} S$ is the smallest convex set which contains S. A set $K \subset \mathbb{E}^n$ is *convex* if it contains the segment connecting any pair of points in it, i.e. if for $\boldsymbol{x}, \boldsymbol{y} \in K$ and $0 \leq \lambda \leq 1$

$$(1 - \lambda)\boldsymbol{x} + \lambda\boldsymbol{y} \in K.$$

In other words, a set is convex when it is closed with respect to convex combination of elements. Also, a set K is *convex* if $K = \operatorname{conv} K$.

A convex non-empty set $A \subset \mathbb{E}^n$ is said to be a *convex cone* if $\boldsymbol{x} \in A$ implies $\lambda\boldsymbol{x} \in A$, with $\lambda \geq 0$. Notice that, if A is a convex cone, then $\mathbf{0} \in A$.

The *dimension* of a convex set K is that of the affine hull: $\dim K = \dim \operatorname{aff} K$.

Hyperplanes, halfspaces, affine subspaces, convex cones and polyhedra are convex sets, and so is the space $\mathbb{E}^n$. If $A, B \subset \mathbb{E}^n$ are convex, then $A + B$ and λA, $\lambda \in \mathbb{R}$ are convex. Furthermore, the intersection and projection of convex sets are convex.

Example 4.2.2 (Convex hull)
In Script 4.2.2 a simple implementation of the `ConvexHull` operator is given, where the function `MK`, introduced in Script 3.3.15, is defined as $\texttt{MK} : \mathbb{E}^n \to \mathcal{P}^{0,n}$. A different implementation for the `ConvexHull` operator was provided in Script 3.2.1.

Script 4.2.2 (`ConvexHull` operator)

```
DEF ConvexHull (points::IsSeqof:IsPoint) = (JOIN ~ AA:MK): points;
```

Homogenization A map

$$\text{homog} : \mathbb{E}^n \to \mathbb{E}^{n+1} : \boldsymbol{x} \mapsto \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}$$

is said to *homogenize* the points of a set $A \subset \mathbb{E}^n$. A set of points $A = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\}$ is affinely independent if and only if the vector images of points in $\text{homog}(A)$ are linearly independent.

Barycentric coordinates Let $A = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be affinely independent, and $\boldsymbol{y} \in \text{aff}\, A$, i.e.:

$$\boldsymbol{y} = \sum_{i=0}^{n} \lambda_i \boldsymbol{x}_i, \quad \text{with} \quad \sum_{i=0}^{n} \lambda_i = 1.$$

The (unique) coefficients $\lambda_0, \ldots, \lambda_d$ are called *barycentric coordinates* of $\boldsymbol{y}$ in A.

Carathéodory's theorem This important theorem states that if $A \subseteq \mathbb{E}^n$ and $\boldsymbol{x} \in \text{conv}\, A$, then $\boldsymbol{x}$ can be expressed as the convex combination of at most $n+1$ affinely independent points of A.

Simplex and its interior The set $\text{conv}\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\}$ of $d+1$ affinely independent points of $\mathbb{E}^n$, $d \leq n$, is called *d-simplex*. Such points are called *vertices*.

A consequence of Carathéodory's theorem is that a set $\text{conv}\, A$ is the union of all simplices with vertices in A.

Let $A := \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be affinely independent, i.e. $\text{conv}\, A$ be a simplex, and $\boldsymbol{x} \in \text{aff}\, A$. Then $\boldsymbol{x} \in \text{relint conv}\, A$ if and only if all the barycentric coordinates of $\boldsymbol{x}$ in A are positive.

Example 4.2.3 (Barycentric coordinates)
To compute the barycentric coordinates $\boldsymbol{\lambda} = (\lambda_0, \lambda_1, \ldots, \lambda_d)$ of a point $\boldsymbol{x} \in \mathbb{E}^d$, either internal or external to a simplex $\text{conv}\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\}$, it is sufficient to solve the simultaneous set of $d+1$ linear equations in $d+1$ unknowns, whose first equation codifies the constraint that $\boldsymbol{\lambda}$ must be a partition of the unity:

$$\begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \boldsymbol{x}_0 & \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_d \end{pmatrix} \boldsymbol{\lambda},$$

so that

$$\boldsymbol{\lambda} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \boldsymbol{x}_0 & \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_d \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}.$$

The above method is equivalent to computing a coordinate transformation for $\mathrm{homog}\,(\boldsymbol{x}) \in \mathbb{E}^{d+1}$ with respect to the new basis $\mathrm{homog}\,\{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots \boldsymbol{x}_d\}$.

Implementation A dimension-independent implementation of this approach is given by the `Convexcoords` function in Script 4.2.3, together with some examples of its use. Notice that the given implementation requires a Cartesian (say, without homogeneous coordinate) representation of the input point `x`. Notice also that the expression `AA:[ID]:(1 AL x)` generates a column matrix representation in $\mathbb{R}_1^d$ of the homogenized `x` point. For example:

```
AA:[ID]:(1 AL <0.5, 0, 0>) ≡ < < 1 > , < 0.5 > , < 0 > , < 0 > >
```

The overloaded `*` operator denotes here infix matrix multiplication. The `IsSimplex` predicate is given in Script 4.4.1.

Script 4.2.3 (Barycentric coordinates)

```
DEF Convexcoords (p::IsSimplex)(x::IsPoint) = CAT:(
   (INV ∼ TRANS ∼ AA:AL ∼ DISTL ∼ [K:1, S1] ∼ UKPOL):p
   * AA:[ID]:(1 AL x)
);

Convexcoords:(SIMPLEX:3):< 1/3, 1/3, 1/3 > ≡
   < 0.3333333333333333 , 0.3333333333333333 , 0.3333333333333333 , 0,0 >
```

The above example shows that the point $\boldsymbol{x} = (1/3, 1/3, 1/3)$ belongs to one of external faces of the unit tetrahedron. The reader might find useful to look once more at Example 3.2.6.

Example 4.2.4 (Hyperplane defined by n points in $\mathbb{E}^n$)
It is often necessary to get the Cartesian equation $ax_1 + bx_2 + cx_3 + d = 0$ of the plane passing through three affinely independent points $\boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{E}^3$. This hyperplane is defined as the set

$$H \subset \mathbb{E}^3 = \mathrm{aff}\,\{\boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{y}_2\} = \boldsymbol{y}_0 + \mathrm{lin}\,\{\boldsymbol{y}_1 - \boldsymbol{y}_0, \boldsymbol{y}_2 - \boldsymbol{y}_0\},$$

so that $\dim H = 2$.

This set may be generated, by using homogeneous coordinates, as the projection of the linear subspace of $\mathbb{E}^4$ spanned by linearly independent vectors $(1, \boldsymbol{y}_i) \in \mathbb{E}^4$ $(0 \leq i \leq 2)$, i.e. as the set

$$H = \{\boldsymbol{x} | (1, \boldsymbol{x}) \in H'\},$$

with

$$H' = \mathrm{lin}\,\{(1, \boldsymbol{y}_0), (1, \boldsymbol{y}_1), (1, \boldsymbol{y}_2)\}$$

With matrix language, and by using the standard notation for determinants, i.e. $\det A = |A|$, the previous statement reduces to:

$$\det A = \begin{vmatrix} 1 & x_1 & x_2 & x_3 \\ 1 & y_{01} & y_{02} & y_{03} \\ 1 & y_{11} & y_{12} & y_{13} \\ 1 & y_{21} & y_{22} & y_{23} \end{vmatrix} = 0,$$

since the first row is a linear combination of the others. Notice that $\operatorname{rank} A = 3$, by definition, since we supposed $\boldsymbol{y}_0, \boldsymbol{y}_1, \boldsymbol{y}_2$ affinely independent. By computing

$$\det A = \sum_{j=1}^{4} (-1)^{1+j} |A_{1j}| \; a_{1j} = ax_1 + bx_2 + cx_3 + d = 0$$

where A_{1j} is the submatrix obtained by cancelling the first row and the j-th column from A, we get

$$a = |A_{11}|, \qquad b = -|A_{12}|, \qquad c = |A_{13}|, \qquad d = -|A_{14}|.$$

This approach is readily extended to the hyperplane $\boldsymbol{h}^T \boldsymbol{x} + h_0 = 0$ defined by n affinely independent points in $\mathbb{E}^n$, so that $h_j = (-1)^{j+1} |A_{1j}|$.

For example, the line in $\mathbb{E}^2$ passing for $(1, 0)$ and $(0, 1)$ is

$$\begin{vmatrix} 0 & 1 \\ 1 & 1 \end{vmatrix} x_1 - \begin{vmatrix} 1 & 1 \\ 0 & 1 \end{vmatrix} x_2 + \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} = -x_1 - x_2 + 1 = 0.$$

4.2.2 Support, separation, extreme points

The main source for this section, as well as for several concepts in this chapter, is Schneider's book [Sch93] on convex bodies and Brunn-Minkowski theory.

Support hyperplanes and halfspaces Let $A \subset \mathbb{E}^n$, $H \subset \mathbb{E}^n$ be a set and a hyperplane, respectively, with H^+ and H^- the two closed halfspaces bounded by H. Then H *supports* A *at* x if $x \in A \cap H$ and either $A \subset H^+$ or $A \subset H^-$.

The hyperplane H is called a *support hyperplane* of A if H supports some boundary point x of A. If $A \subset H^+$ ($A \subset H^-$), then H^+ (H^-) is called a *support halfspace* of A.

The existence of support hyperplanes through *every boundary point* of a set $A \subset \mathbb{E}^n$ characterizes it as a *convex* set. In fact, it is possible to prove [Sch93] that if this existence is guaranteed, then the set is convex. Each non-empty closed convex set in $\mathbb{E}^n$ is the intersection of its supporting halfspaces.

Separation The sets A and B are *separated* by an hyperplane H if $A \subset H^+$ and $B \subset H^-$. non-empty convex sets A and B can be properly separated if and only if $\operatorname{relint} A \cap \operatorname{relint} B = \emptyset$

Extremal points An *extremal point* of a convex set A is a point $z \in A$ that cannot be written in the form $z = (1-\lambda)x + \lambda y$, for $x, y \in A$ and $\lambda \in (0,1)$. If $\{z\} \subset A$ is a face (see Section 4.4.2), then z is an extremal point.

It is very important to write a closed convex set as the convex hull of a much smaller finite set. *Minkowski's theorem* states that each convex body is the convex hull of its extremal points.

4.2.3 Duality

Polar set of a convex Let $S \subset \mathbb{E}^n$ be a convex body with $\boldsymbol{o} \in \operatorname{int} S$. The *polar set* of S is properly defined as the subset of dual space $(\mathbb{E}^n)^*$:

$$S^* := \{\boldsymbol{y} \in (\mathbb{E}^n)^* | \boldsymbol{y}(\boldsymbol{x}) \leq 1, \text{for all } \boldsymbol{x} \in S\}.$$

By the canonical isomorphism between $(\mathbb{E}^n)^*$ and $\mathbb{E}^n$ induced by the Euclidean scalar product, and the representation theorem (3.4), the polar set can be directly given in $\mathbb{E}^n$, as

$$S^* := \{\boldsymbol{y} \in \mathbb{E}^n | \boldsymbol{x} \cdot \boldsymbol{y} \leq 1, \text{for all } \boldsymbol{x} \in S\}.$$

Polarity is a duality It is possible to show that polarity is a true duality [Sch93], since: the polar S^* of a convex S is convex, satisfies the requirement that $\boldsymbol{o} \in \operatorname{int} S^*$, and $S^{**} = S$.

The polarity relation is called *duality* in many books, so that we use occasionally the same word, mainly in Chapter 13. Notice that the polar set depends on the position of $\boldsymbol{o}$. This could be avoided by defining "cone polarity" between linearized sets [Zie95].

Example 4.2.5 (Polar sets of d-cubes)
A d-cube C_d is defined as a Cartesian product of d closed intervals $[-1,1] \subset \mathbb{E}$:

$$C_d := [-1,+1]^d = \{\boldsymbol{x} \in \mathbb{E}^d | -1 \leq x_i \leq 1\},$$

The polar set of the standard square $C_2 \subset \mathbb{E}^2$ is the *rhombus* C_2^*. The polar set of the standard cube $C_3 \subset \mathbb{E}^3$ is the *octahedron* C_3^*. Both such cubes and their dual are shown in Figure 4.2. Notice that faces of a convex body correspond to vertices of its dual.

Implementation In Script 4.2.4 some `PLaSM` functions are given, to generate both d-cubes and their polar sets.

In particular, `DeHomog` is the inverse of the homog operator of Section 4.2. In this case it is not sufficient to drop the first coordinate of $\boldsymbol{x} = (x_0, x_1, \ldots, x_d)$, because it may be $x_0 \neq 1$. In fact `DeHomog` is applied to covectors $\boldsymbol{x} \in (\mathbb{E}^d)^*$ returned by `PLaSM` and these are stored in a normalized form such that $|\boldsymbol{x}| = 1$.

The `UKPOLF` (UnmaKe POLyhedron by Faces) function, which is a predefined `PLaSM` operator, returns the internal representation *by faces* of a polyhedral complex as a triplet `<covectors, cells, pols>`, very similar to the triplet `<vertices, cells, pols>` returned by the `UKPOL` operator. For example, we have:

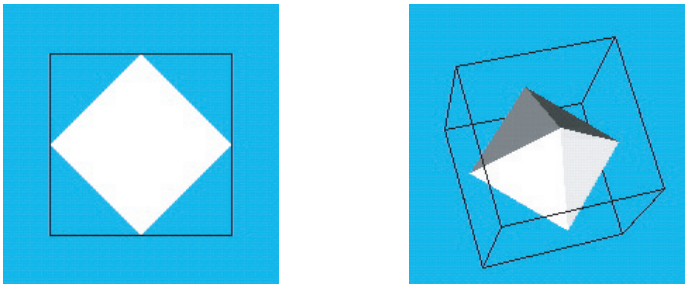

Figure 4.2 (a) C_2 cube and its polar set C_2^*, known as *rhombus*; (b) Octahedron C_3^*, polar set of the C_3 cube

```
UKPOLF:(CUBOID:<1,1>) ≡ <
   < < 1.0, 0.0, 0.0 >,
     < -0.7071067811865475, 0.0, 0.7071067811865475 >,
     < 0.0, 1.0, 0.0 >,
     < 0.0, -0.7071067811865475, 0.7071067811865475 > >,
   < < 1,2,3,4 > >, < < 1 > > >
```

Thus, the `Polar` function just constructs a polyhedron by using as vertices the (projected) face covectors of its dual.

Finally, notice that `Segment` produces a direct construction of the interval $[-1, 1]$ as a polyhedron in $\mathcal{P}^{1,1}$, whereas

$$\texttt{Cube} : \mathrm{Z}^+ \to \mathcal{P}^{d,d} : d \mapsto C_d,$$

with $\mathrm{Z}^+ = \{1, 2, \dots\}$, is the constructor function of d-cubes, via the Cartesian product of d `Segment` instances. The last two expressions of Script 4.2.4 produce the geometric objects displayed in Figure 4.2.

Script 4.2.4 (Polar sets of d-cubes)

```
DEF DeHomog = AA:/ ~ DISTR ~ [TAIL, FIRST];
DEF Polar = MKPOL ~ [AA: DeHomog ~ S1,S2,S3] ~ UKPOLF;

DEF Segment = MKPOL:<<<-1>,<1>>,<<1,2>>,<<1>>>;
DEF Cube (d::IsIntPos) = (* ~ #:d): Segment;
DEF Rhombus = (Polar ~ Cube):2;
DEF Octahedron = (Polar ~ Cube):3;

STRUCT:< (@1 ~ Cube):2, Rhombus >;
STRUCT:< (@1 ~ Cube):3, Octahedron >;
```

No test for $x_0 = 0$ was actually performed before division in function `DeHomog`, because we used the `Polar` function in the previous assumption that $\boldsymbol{o} \in \operatorname{int} C$, where C is the input convex set.

A new implementation of `DeHomog` function is given in Script 4.2.5, where a test for the null value of the homogeneous coordinate of covectors is performed. Such a coordinate is simply dropped out when equal to zero, otherwise it is used to divide

the other ones. The `CLOSETO` function is used to test if the absolute value of difference of the input to some real `number` is smaller than some given `precision`.

Script 4.2.5 (Polar sets)

```
DEF CLOSETO (precision::IsRealPos)(number::IsReal) =
   IF:< LT: precision ~ ABS ~ - ~ [ID, K:number], K:True, K:False >;
DEF RTAIL = REVERSE ~ TAIL ~ REVERSE;
DEF DeHomog =
   IF:< CLOSETO:1E-12:0 ~ LAST, RTAIL , AA:/ ~ DISTR ~ [RTAIL, LAST] >;
DEF Polar = MKPOL ~ [AA: DeHomog ~ S1,S2,S3] ~ UKPOLF;
```

It is interesting to see in Script 4.2.6 that the *Platonic solids*, discussed in Section 4.9.1, are pairwise polar, with the `tetrahedron` being the polar of itself. Notice that in order to apply the `polar` function, the `dodecahedron` implementation given in Script 4.9.4 as a multicell polyhedral complex, must be preliminary transformed into a polyhedron constituted by a single cell. The single cell generated by the expression `JOIN:< dodecahedron >` could be used for this purpose. A *much* better programming strategy, which avoids the quite complex geometric construction of Script 4.9.4, is given below; the `icosahedron` implementation is given in Script 4.9.5.

Script 4.2.6 (Polar of Platonic solids)

```
(Polar):tetrahedron ≡ tetrahedron

(Polar ~ CUBOID):<1,1,1> ≡ octahedron
(Polar ~ Polar ~ CUBOID):<1,1,1> ≡ hexahedron

DEF dodecahedron1 = Polar:icosahedron;

(Polar):dodecahedron1 ≡ icosahedron
(Polar ~ Polar):dodecahedron1 ≡ dodecahedron
```

4.2.4 Boundary structure

Let us suppose in this section that $C \subset \mathbb{E}^n$ is a non-empty closed convex set.

Faces A *face* of C can be defined as a convex subset $F \subseteq C$ such that $\boldsymbol{x}, \boldsymbol{y} \in C$ and $(\boldsymbol{x}+\boldsymbol{y})/2 \in F$ implies $\boldsymbol{x}, \boldsymbol{y} \in F$. Notice that both C and $\emptyset$ are faces of C. A non-empty face F is said to be *proper* when F is a proper subset of C. The set of faces of C will be denoted as $\mathcal{F}(C)$. A face of dimension d will be called a d-face, and $\mathcal{F}_d(C) \subset \mathcal{F}(C)$ will denote the set of d-faces.

The relative interiors of any two faces in $\mathcal{F}(C)$ are disjointed. Thus, the family

$$\{\text{relint}\, F | F \in \mathcal{F}(C)\} \cup \mathcal{F}_0(C)$$

of relative interiors of faces of C, united with the set of 0-dimensional faces $\mathcal{F}_0(C)$, i.e. of faces $\{\boldsymbol{x}\}$, where $\boldsymbol{x}$ is an extreme point, gives a partition of C into disjointed

subsets.

Skeletons The last property leads to a classification of a point $\boldsymbol{x} \in C$ depending on the dimension of the uniquely determined face F_x whose relative interior $\boldsymbol{x}$ belongs to. The set $K_r(C)$ of all points $\boldsymbol{x}$ belonging to faces of dimension $d \leq r$ will be called *r-skeleton* of C:

$$K_r(C) := \{\boldsymbol{x} \in F_x | F_x \in \mathcal{F}_d(C), d \leq r\}.$$

Clearly for $0 \leq r < \dim C$, $K_r(C) \subseteq \partial C$, where the equality holds if and only if $r = \dim C - 1$.

4.3 Polyhedral sets

Polyhedra and cones are collectively called *polyhedral sets*. They are often referred to as $\mathcal{H}$-polyhedra, which means that they can be presented as the intersection of closed halfspaces.

Definitions A set $P \in \mathbb{E}^n$ is a *polyhedron* if and only if it is the intersection of a finite number of closed halfspaces. So, we define:

$$\mathcal{H}(P) := P(\boldsymbol{A}, \boldsymbol{b}) := \{\boldsymbol{x} \in \mathbb{E}^n | \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\},$$

for some $\boldsymbol{A} \in \mathbb{R}_n^m$ and some $\boldsymbol{b} \in \mathbb{R}^m$.

A set $C \in \mathbb{E}^n$ is said a *polyhedral cone* if and only if

$$\mathcal{H}(C) := C(\boldsymbol{A}, \boldsymbol{0}) := \{\boldsymbol{x} \in \mathbb{E}^n | \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{0}\},$$

for some $\boldsymbol{A} \in \mathbb{R}_n^m$. Such a kind of set is both a cone and a polyhedron.

4.3.1 Extremal points

A point of a convex set P is called an *extremal point* if it cannot be written as $(1-\alpha)\boldsymbol{x} + \alpha\boldsymbol{y}$, with $\boldsymbol{x}, \boldsymbol{y} \in P$ and $0 < \alpha < 1$. The set of extremal points of P is denoted as $\operatorname{ext} P$.

The inequality $\boldsymbol{a}\boldsymbol{x} \leq \beta$ is said to be *valid* with respect to the polyhedron P if and only if $P \subseteq H^+_{\boldsymbol{a},\beta}$.

The hyperplane $H_{\boldsymbol{a},\beta}$ is said to be a *support hyperplane* for the P polyhedron if either $\boldsymbol{a}\boldsymbol{x} \leq \beta$ or $\boldsymbol{a}\boldsymbol{x} \geq \beta$ is valid and $H_{\boldsymbol{a},\beta} \cap P \neq \emptyset$.

Let $P \subset \mathbb{E}^n$ be a polyhedron and H a support hyperplane of P. The set $F = P \cap H$ is called a *face* of P. The F face is a *vertex* if $\dim F = 0$; it is an *edge* if $\dim F = 1$, and it is a *facet* if $\dim F = n - 1$.

Algebraic characterization of vertices Let $P = P(\boldsymbol{A}, \boldsymbol{b})$ be a polyhedron of $\mathbb{E}^n$, with $\boldsymbol{A} \in \mathbb{R}_n^m$ and $\boldsymbol{b} \in \mathbb{R}^m$. A point $\boldsymbol{x} \in P$ is an extreme of P if and only if it satisfies as equations n rows of the system $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$. It follows that a polyhedron $P(\boldsymbol{A}, \boldsymbol{b}) \subset \mathbb{E}^n$ contains extreme points if and only if $\operatorname{rank} \boldsymbol{A} = n$.

The extreme points of $P(\boldsymbol{A}, \boldsymbol{b}) \in \mathbb{E}^n$, if they exist, are a discrete set denoted $\operatorname{ext} P(\boldsymbol{A})$, and their number at most equates the number $\binom{m}{n}$ of $\boldsymbol{A}$ minors with rank n. Such points are called polyhedron *vertices*.

4.3.2 Double description

The following *main theorems*[Zie95] give three pairs of alternative characterization of *polyhedra*, *polytopes*, and *cones*, respectively, that appear all to be very useful, maybe in different contexts. Notice that the same notation is used both for matrices, considered as sets of column vectors, and for sets of points in Euclidean space. For proofs, the reader is referred to [Zie95].

Theorem 4.3.1 (Main theorem for polyhedra) *A set $P \subseteq \mathbb{E}^d$ is the sum of a convex hull of a finite set of points plus a conical combination of vectors, say, a $\mathcal{V}$-polyhedron*

$$P = \operatorname{conv} V + \operatorname{cone} E, \qquad \text{for some } V \in \mathbb{E}_n^d, E \in \mathbb{R}_{n'}^d,$$

if and only if it is an intersection of halfspaces, say a $\mathcal{H}$-polyhedron

$$P = P(\boldsymbol{A}, \boldsymbol{b}), \qquad \text{for some } \boldsymbol{A} \in \mathbb{E}_d^m, \boldsymbol{b} \in \mathbb{E}^m,$$

with $V = \operatorname{ext} P(\boldsymbol{A}, \boldsymbol{b})$ and $\operatorname{cone} E = P(\boldsymbol{A}, \boldsymbol{0})$.

In other terms, a set $P \subseteq \mathbb{E}^d$ can be finitely generated either as a convex-conical combination of vectors ($\mathcal{V}$-polyhedron) or as the intersection of halfspaces ($\mathcal{H}$-polyhedron). If $P(\boldsymbol{A}, \boldsymbol{0}) = \{\boldsymbol{0}\}$, the $\mathcal{V}$-polyhedron is called *polytope* or *$\mathcal{V}$-polytope*, so that the previous theorem becomes the following one.

Theorem 4.3.2 (Main theorem for polytopes) *A set $P \subseteq \mathbb{E}^d$ is the convex hull of a finite set of points, say a $\mathcal{V}$-polytope*

$$P = \operatorname{conv} V, \qquad \text{for some } V \in \mathbb{E}_n^d$$

if and only if it is a bounded *intersection of halfspaces, say a $\mathcal{H}$-polytope*

$$P = P(\boldsymbol{A}, \boldsymbol{b}), \qquad \text{for some } \boldsymbol{A} \in \mathbb{E}_d^m, \boldsymbol{b} \in \mathbb{E}^m.$$

An alternate definition of a polytope P is as a *bounded polyhedron*, i.e. such that there exists some $\kappa \in \mathbb{R}$ such that $|\boldsymbol{x} - \boldsymbol{y}| < \kappa$ for each $\boldsymbol{x}, \boldsymbol{y} \in P$. Finally, if $\operatorname{ext} P = \{\boldsymbol{0}\}$, then the $\mathcal{V}$-polyhedron is called a *polyhedral cone*. In this case we have:

Theorem 4.3.3 (Main theorem for cones) *A set $C \subseteq \mathbb{E}^d$ is the conical combination of a finite set of vectors*

$$C = \operatorname{cone} E, \qquad \text{for some } E \in \mathbb{E}_n^d$$

if and only if it is a finite intersection of closed halfspaces containing the origin

$$C = P(\boldsymbol{A}, \boldsymbol{0}), \qquad \text{for some } \boldsymbol{A} \in \mathbb{E}_d^m.$$

Note The PLaSM kernel maintains both a $\mathcal{H}$- and a $\mathcal{V}$-representation of convex cells of a polyhedral complex. They are used as primary representations in the implementation of different operations. For example, the $\mathcal{H}$–representation is mainly useful with the Cartesian product and the Boolean operations, whereas the $\mathcal{V}$–representation is needed to generate the graphical output and to compute the inertial properties of a polyhedral complex.

4.4 Polytopes

We already know that a polyhedron is the solution set of a system of linear inequalities $\boldsymbol{Ax} \leq \boldsymbol{b}$, and that a *polytope* is a bounded polyhedron. In particular we know that a polyhedron is a polytope when it does not contain a cone, i.e. when the associated homogeneous system $\boldsymbol{Ax} \leq \boldsymbol{0}$ has no solutions different from the zero vector.

Properties Some useful properties of polytopes are listed below.

1. Each polytope is the intersection of finitely many closed halfspaces.
2. Each bounded intersection of finitely many closed halfspaces is a polytope.
3. The join, intersection, sum, product, projection of polytopes is a polytope.
4. The projection of a simplex is a polytope.
5. The polar body of a polytope is a polytope.
6. Each face of a polytope is a polytope.
7. Each proper face of a polytope P is contained in some facet of P.

Simplicial and simple polytopes A polytope $P \subset \mathbb{E}^d$ is said to be *simplicial* when all its facets are simplices. It is said to be *simple* when each vertex is generated as intersection of the minimal number d of facets. Examples of simplicial polytopes are the octahedron and the icosahedron, where each face is a triangle. Examples of simple polytopes are the cube and the dodecahedron, where each vertex is the intersection of three facets. Simplicial and simple polytopes are linked by polarity: the polar of a simple polytope is a simplicial polytope, and vice versa. Simplices are the only polytopes which are both simple and simplicial.

Example 4.4.1 (`IsPolytope` and `IsSimplex` predicates)
Two PLaSM predicates `IsPolytope` and `IsSimplex` are given in Script 4.4.1, in order to test if a geometric object is respectively a polytope or a simplex (of whatever dimensions) or not. Both predicates contain a logical `AND` of two component predicates. The predefined `IsPol` function is used to check if the function input is a polyhedral complex; the second predicate of `IsPolytope` just checks if the input contains a unique convex cell. The `IsSimplex` function aims to verify if the input is a polytope and if the number of vertices is $n + 1$, where n is the dimension of the embedding space $\mathbb{E}^n$ of the input object.

4.4.1 Examples of polytopes

Some interesting and useful classes of polytopes are studied and implemented in this section. In particular, we give generative functions for *regular polygons* in 2D, the

Script 4.4.1 (`IsPolytope` and `IsSimplex`)

```
DEF IsPolytope = AND ~ [IsPol, EQ ~ [LEN ~ S2 ~ UKPOL, K:1]];
DEF IsSimplex = AND ~ [ IsPolytope, EQ ~ [LEN ~ S1 ~ UKPOL, RN + K:1] ];

IsSimplex:(CUBOID:<1>) ≡ True
IsSimplex:(SIMPLEX:3) ≡ True
IsSimplex:(CUBOID:<1,1>) ≡ False
IsPolytope:(CUBOID:<1,1>) ≡ True
```

standard *d-simplex*, the d-dimensional *crosspolytope*, the *pyramid* and the *prism* over a d-polytope, and the d-dimensional *permutahedron*, where d is an arbitrary positive integer.

Regular polygons

2-polytopes with n vertices are called *n-gons* or polygons. The word *polygon* is normally reserved to denote a larger class of 2D geometric objects, that are not necessarily convex. No ambiguity arises with *regular polygons*, which are n-gons with internal angles of equal size $\frac{2\pi}{n}$.

Regular polygons are easily generated by the `ngon` function of Script 4.4.2 as polyhedral approximation of the unit circle. The `Circumference` function given in Script 1.6.1 is used at this purpose. The last expression generates the compound geometric value with interposed translations shown in Figure 4.3.

Script 4.4.2 (Regular polygons)

```
DEF ngon (n::AND ~ [IsIntPos, GE:3]) = Circumference:1:n;

(STRUCT ~ CAT): (AA:ngon:(3..8) DISTR T:1:2.5)
```

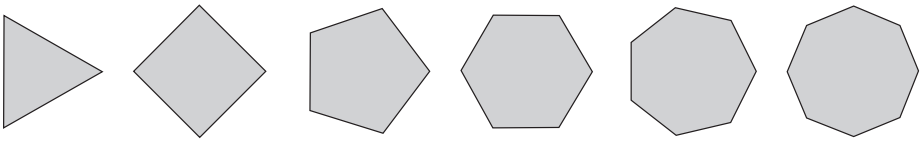

Figure 4.3 Regular polygons with a number of sides from 3 (triangle) to 8 (octagon)

Standard d-simplex

In some books, including [Zie95], the standard d-simplex Δ_d is defined in $\mathbb{E}^{d+1}$ as the intersection of the hyperplane $\mathbb{1}\boldsymbol{x} = 1$ with the standard cone $\boldsymbol{x} \geq 0$, i.e.:

$$\Delta_d := \{\boldsymbol{x} \in \mathbb{E}^{d+1} | \mathbb{1}\boldsymbol{x} = 1, \boldsymbol{x} \geq 0\} = \text{conv}\,\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_{d+1}\}$$

Conversely, in PLaSM, the unit d-simplex is defined in $\mathbb{E}^d$ as

$$\texttt{SIMPLEX:d} \equiv \text{conv}\,\{\boldsymbol{o}, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_d\}.$$

Generating Δ_d is very simple. A direct generation method as convex hull of the extreme points of vectors $\{\boldsymbol{e}_i\} \subset \mathbb{E}^{d+1}$ is given in Script 4.4.3 by the Delta function. The IDNT function, used to produce the sequence of $\{\boldsymbol{e}_i\}$ vectors, is given in Script 3.3.5.

The MyFrame function simply produces a polyhedral complex in $\mathcal{P}^{1,d}$, whose cells are bijectively associated to vectors $\boldsymbol{e}_i$, with $1 \leq i \leq d$. Each unit vector "image" is scaled to 1.5 size.

A picture of the generated Δ_2, compared to a picture of SIMPLEX:2, is shown in Figure 4.4.1. The second one is clearly reducible to the first by an affine transformation. The associated geometric objects are produced by the last two expressions of Script 4.4.3.

Script 4.4.3 (Standard d-simplex)

```
DEF Delta (d::IsInt) = MKPOL:< IDNT:(d+1), <1..(d+1)>, <<1>> >;

DEF MyFrame (d::IsInt) = (S:(1..d):(#:d:1.5) ~ MKPOL):
   < #:d:0 AL IDNT:d, 1 DISTL (2..(d+1)), <1..d> >;

STRUCT:<Delta:2, MyFrame:3>;
STRUCT:<SIMPLEX:2, MyFrame:2>;
```

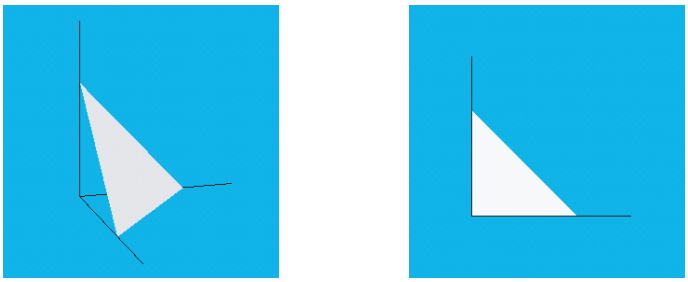

Figure 4.4 (a) Standard d-simplex Δ_2; (b) Geometric value generated by SIMPLEX:2

d-dimensional crosspolytope

A *crosspolytope* of dimension d, denoted C_d^{Δ}, is the convex hull of the set $\{\boldsymbol{e}_i\} \cup \{-\boldsymbol{e}_i\}$, $1 \leq i \leq d$. Such polytopes are very easy to build with PLaSM, as the polyhedral complex whose vertices are generated by the expression

```
(CAT ~ [ID, (AA ~ AA):-] ~ IDNT):d
```

and having just one convex cell, i.e. the convex hull of the above $2d$ vertices. A generating function

$$\texttt{CrossPolytope}:Z^+ \to \mathcal{P}^{d,d} : d \mapsto C_d^\Delta,$$

with $Z^+ = \{1, 2, 3, \ldots\}$, is given in Script 4.4.4. The geometric constructions in Figure 4.5, with *rhombus* $\subset \mathbb{E}^2$, *octahedron* $\subset \mathbb{E}^3$ and their coordinate axes, are respectively generated by the last two PLaSM expressions.

Script 4.4.4 (d-Crosspolytopes)

```
DEF CrossPolytope (d::IsIntPos) = MKPOL:<
   (CAT ~ [ID, (AA ~ AA):-] ~ IDNT):d, <1..(2*d)>, <<1>> >;

(STRUCT ~ [CrossPolytope, Frame]):2;
(STRUCT ~ [CrossPolytope, Frame]):3;
```

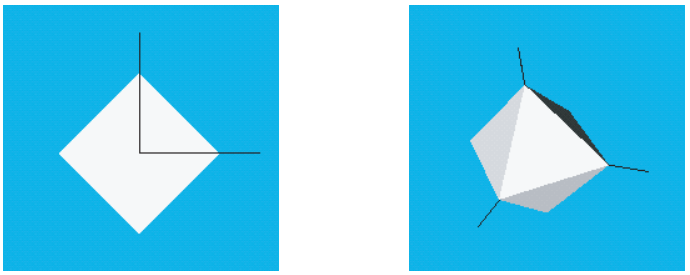

Figure 4.5 CrossPolytopes C_2^Δ and C_3^Δ

Prism over a polytope

A *prism* over a polytope P is defined as a Cartesian product of the polytope times a unit interval $[0, 1] \subset \mathbb{R}$.

$$\texttt{Prism} : \mathcal{P}^{d,n} \to \mathcal{P}^{d+1,n+1} : P \mapsto P \times [0, 1]$$

The PLaSM implementation is extremely simple, because we may use the operation of *Cartesian product* (see Section 14.4) as a language primitive. The `Prism` definition is here usefully abstracted with respect to the `basis` object and to the `height` of the generated object. This operation is called *linear extrusion* in solid modeling.

Script 4.4.5 (Prism over a polytope)

```
DEF Prism (height::IsRealPos)(basis::IsPol) = basis * QUOTE:< height >;
```

Clearly, the `basis` object can be either a polytope or a polyhedral complex.

Permutahedron

The permutahedron $\Pi_{d-1} \subset \mathbb{E}^d$ is defined as the convex hull of the permutations of the first d integers, i.e.:

$$\Pi_{d-1} := \{\boldsymbol{x} \in \text{conv}\{\ (\pi_i(1), \pi_i(2), \ldots, \pi_i(d)) \in \mathbb{E}^d\}\ \}, \qquad i \in \{1, \ldots, d!\}.$$

We remember that each permutation π_i of a finite set is a bijective mapping on it, and in particular:

$$\pi_i : \{1, 2, \ldots, d\} \to \{1, 2, \ldots, d\}, \qquad \text{bijective.}$$

Clearly, each point $(\pi_i(1), \pi_i(2), \ldots, \pi_i(d))$ belongs to the sphere $S_{d-1} \subset \mathbb{E}^d$ of radius $(\pi_i(1)^2 + \pi_i(2)^2 + \cdots + \pi_i(d)^2)^{\frac{1}{2}}$. It is possible to show that $\dim(\text{aff}\ \Pi_{d-1}) = d-1$.

Permutations The `permutations` function given in Script 4.4.6, is a general utility operator that allows generation of the (images of) the whole group of permutations of `<1,2, ... ,d>`.

The used algorithm starts with the pair `<<>,<1,2, ... ,d>>` and produces d pairs `<<1>,<2, ... ,d>>, <<2>,<1,3, ... ,d>>, ... ,<<d>,<1,2, ... ,d−1>>`; then from each `<<`i`>,<1, ... ,`$i-1, i+1$`, ... ,d>>` produces $d-1$ pairs `<<`i`,1>,<2, ... ,`$i-1, i+1$`, ... ,d>>`, `<<`i`,2>,<1,3, ... ,`$i-1, i+1$`, ... ,d>>`, ... , and so on, until the second sequence of each pair becomes empty and is removed.

Script 4.4.6 (Permutations)

```
DEF remove (n::IsIntPos; seq::IsSeq) =
   (CONS ∼ AA:SEL ∼ CAT):< 1 .. n - 1, n + 1 .. LEN:seq >:seq;

DEF permutations (seq::IsSeq) =
   COMP:(AA:CAT AL #:n:(CAT ∼ AA:permute)):<< <>, seq>>
WHERE
   n = LEN:seq,
   extract = AA: remove ∼ DISTR ∼ [INTSTO ∼ LEN, ID],
   permute = TRANS ∼ [ AA:AR ∼ DISTL, extract ∼ S2 ]
END;

permutations:<1,2,3> ≡ <<1,2,3>,<1,3,2>,<2,1,3>,<2,3,1>,<3,1,2>,<3,2,1>>

permutations:<1,2,3,4> ≡
   <<1,2,3,4>,<1,2,4,3>,<1,3,2,4>,<1,3,4,2>,<1,4,2,3>,<1,4,3,2>,<2,1,3,4>
   ,<2,1,4,3>,<2,3,1,4>,<2,3,4,1>,<2,4,1,3>,<2,4,3,1>,<3,1,2,4>,<3,1,4,2>
   ,<3,2,1,4>,<3,2,4,1>,<3,4,1,2>,<3,4,2,1>,<4,1,2,3>,<4,1,3,2>,<4,2,1,3>
   ,<4,2,3,1>,<4,3,1,2>,<4,3,2,1>>
```

Permutahedron implementation As always in this book, a complete implementation of the studied object follows. In particular, the Π_{d-1} permutahedron is implemented in a dimension-independent fashion in Script 4.4.7. Pictures of the objects generated by the last expressions are shown in Figure 4.6.

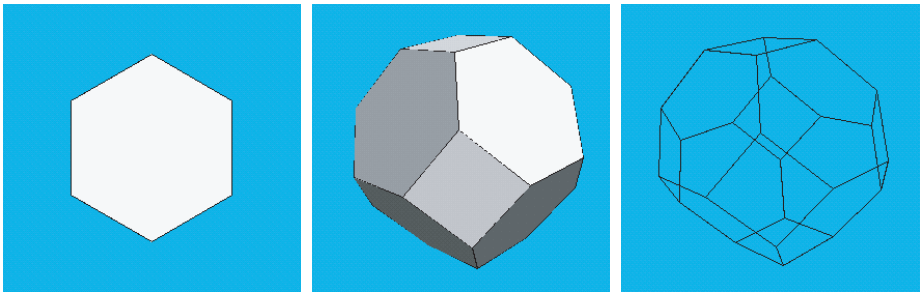

Figure 4.6 (a) Permutahedron Π_2 (b) Permutahedron Π_3 (c) 1-skeleton of Π_3

A d-dimensional `object` is first generated by `MKPOL` as the polytope with `vertices` given by `permutations:<1, ... ,d+1>`. The function `translation` is then applied to it, so moving its `Meanpoint` to the origin. Then a suitable sequence of `rotations` is applied, until the set aff `translation:object` is rotated to the coordinate subspace $\{\boldsymbol{x}|x_{d+1} = 0\}$. The last (zero) coordinate is finally eliminated by application of the function `project:1`.

Script 4.4.7 (Permutahedron)

```
DEF permutahedron (d::IsIntPos) =
   ( project:1 ∼ rotations ∼ translation ):object
WHERE
   object = (MKPOL ∼ [ID, [INTSTO ∼ LEN], K:<<1>>]): vertices,
   vertices = permutations:(1 .. (d+1)),
   center = Meanpoint: vertices,
   translation = T:(1..(d+1)):(AA:-: center),
   rotations = COMP:(((CONS ∼ AA:R):(1..d DISTR (d+1))):(PI/4))
END;

VRML: ((STRUCT ∼ [ID, @1]):(permutahedron:2)):'out1.wrl';
VRML: ((STRUCT ∼ [ID, @1]):(permutahedron:3)):'out2.wrl';
VRML: (@1:(permutahedron:3)):'out3.wrl';
```

It may be interesting to notice that polytopes Π_2 and Π_3 are subject to `rotations` tensors

```
(COMP ∼ [R:<1,3>, R:<2,3>]):π/4 ,
(COMP ∼ [R:<1,4>, R:<2,4>, R:<3,4>]):π/4
```

respectively, before being projected, as they are generated in higher dimension. Analogously, for the Π_d polytope we have:

```
(COMP ∼ [R:<1,d+1>, R:<2,d+1>, ... , R:<d,d+1>]):π/4
```

The diligent reader might also like to know that the functions `Meanpoint` and `project` are defined in Scripts 4.4.8 and 4.4.9, respectively.

4.4.2 Faces of polytopes

Let $P = P(\boldsymbol{A}, \boldsymbol{b}) \subset \mathbb{E}^n$ be a polytope.

A set $F \subset P$ is called a *face* of P if there exists a valid inequality $\boldsymbol{a}^T\boldsymbol{x} \leq \beta$ such that $F = \{\boldsymbol{x} \in P | \boldsymbol{a}^T\boldsymbol{x} = \beta\}$. F is said to be the face induced by $\boldsymbol{a}^T\boldsymbol{x} \leq \beta$.

Face lattice Both the sets $F = P$ and $F = \emptyset$ are faces of P, because the above definition is satisfied by the equalities $\mathbf{0}\boldsymbol{x} = 0$ and $\mathbf{0}\boldsymbol{x} = -1$, respectively.

Also, every intersection of faces is a face. In fact, let $F_i = \{\boldsymbol{x} \in P | \boldsymbol{a}_i^T\boldsymbol{x} = \beta_i\}$ and $F_j = \{\boldsymbol{x} \in P | \boldsymbol{a}_j^T\boldsymbol{x} = \beta_j\}$. Then we have that the set $F_i \cap F_j$ is induced by any linear combination of the equations of F_i and F_j, for example:

$$F_i \cap F_j = \{\boldsymbol{x} \in P | (\boldsymbol{a}_j + \boldsymbol{a}_j)^T\boldsymbol{x} = (\beta_i + \beta_j)\},$$

and hence is a face.

It is possible to see [Zie95] that the set $\mathcal{F}(P)$ of faces of a polytope P is a *lattice*, i.e. a partially ordered algebraic structure with unit P and zero $\emptyset$, and with two operations *join* ($\vee$) and *meet* ($\wedge$) that satisfy the standard axioms of Boolean algebras. The two operations respectively return the unique minimal common ancestor and the unique maximal common descendant of any pair of faces, being the lattice partially ordered with respect to a containment relation.

Such algebraic structure is particularly easy to see in $\mathcal{F}(\sigma_d)$, where σ_d is a simplex, and the face lattice is isomorphic to the family $2^{V(\sigma_d)}$ of subsets of σ_d vertices.

Dimension The *dimension* of a face F is the dimension of $\operatorname{aff} F$.

Pyramid over a polytope

A *pyramid* is defined as the convex combination of a d-polytope $P \subset \mathbb{E}^n$ and a point $\boldsymbol{y} \notin \operatorname{aff} P$:

$$\texttt{Pyramid} : \mathcal{P}^{d,n} \times \mathbb{E}^n \to \mathcal{P}^{d+1,n} : (P, \boldsymbol{y}) \mapsto \operatorname{conv}(P \cup \{\boldsymbol{y}\}).$$

Polytope P and point $\boldsymbol{y}$ are called the *basis* and *apex* of the pyramid, respectively.

Faces of pyramid The set $\mathcal{F}(Q)$ of faces of $Q = \texttt{pyramid}(P, \boldsymbol{y})$ is easy to compute:

$$\mathcal{F}(Q) = \mathcal{F}(P) \cup \{F\boldsymbol{y} | F \in \mathcal{F}(P)\} \cup \{P\boldsymbol{y}\}.$$

where $A\boldsymbol{x}$ stands for the *join* of the set A and the point $\boldsymbol{x}$.

Implementation The implementation of Script 4.4.8 assumes P to be full-dimensional, and embeds it so that $\operatorname{aff} P := \{\boldsymbol{x} \in \mathbb{E}^{n+1} | x_{n+1} = 0\}$, and sets $\boldsymbol{y} := (\texttt{Meanpoint V}(P), h)$, $h \neq 0$, where $\texttt{Meanpoint } S = \frac{1}{|S|}\sum S$, with $S \subset \mathbb{E}^n$ a discrete set. Notice that $\texttt{Meanpoint } S \in \operatorname{conv} S$, because it is a convex combination of points of S. The function

$$\texttt{MK} : \mathbb{E}^n \to \mathcal{P}^{0,n} : \boldsymbol{x} \mapsto \operatorname{conv}\{\boldsymbol{x}\}$$

may be found in Script 3.3.15.

The geometric object generated by last expression of the script below is shown in Figure 4.7. The function `SPLIT`, to extract the sequence of convex cells from a polyhedral complex, is given in Script 10.8.4. The definition of the `house2` object, entered as a `triangleStrip`, can be found in Script 7.2.19.

Script 4.4.8 ($(d+1)$-pyramid)

```
DEF mean = + / LEN;
DEF Meanpoint = AA:mean ~ TRANS;
DEF Pyramid (h::IsReal) = JOIN
   ~ [EMBED:1, MK ~ AR ~ [Meanpoint ~ S1 ~ UKPOL, K:h]];

(STRUCT ~ AA:(Pyramid:1) ~ SPLIT): house2;
```

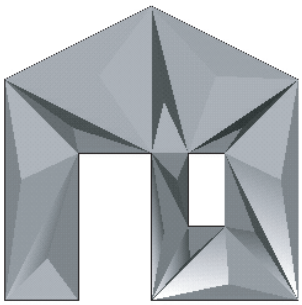

Figure 4.7 Pyramids over the convex cells of a polyhedral complex generated as a triangle strip

4.4.3 Projection

Given two $\mathcal{H}$-polyhedra P and Q, such that

$$P(\boldsymbol{A}, \boldsymbol{b}) \subset \mathbb{E}^n, \quad \text{and} \quad Q(\boldsymbol{A}', \boldsymbol{b}') \subset \mathbb{E}^q,$$

with $q \leq n$, we say that Q is obtained by *projection* of P if and only if:

$$\boldsymbol{y} \in Q \quad \Longleftrightarrow \quad \exists \boldsymbol{z} \in \mathbb{E}^{n-q} \quad \text{such that} \quad \boldsymbol{x} = (\boldsymbol{y}, \boldsymbol{z}) \in P$$

Algebraically speaking, we say that the system of inequalities $\boldsymbol{A}'\boldsymbol{y} \leq \boldsymbol{b}'$ is derived from $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ by *eliminating* the components of $\boldsymbol{z}$.

It turns out more useful to define a projection operator along a single coordinate direction. Let us start with $P(\boldsymbol{A}, \boldsymbol{b}) \subset \mathbb{E}^n$ and suppose we want to project on the coordinate hyperplane $\{\boldsymbol{x} \in \mathbb{E}^n | x_k = 0\}$. So we define, following Ziegler [Zie95], the *projection* of P in the direction of $\boldsymbol{e}_k$ as:

$$\begin{aligned} \text{proj}_k P &:= \{\boldsymbol{x} - x_k \boldsymbol{e}_k | \boldsymbol{x} \in P\} \\ &= \{\boldsymbol{x} \in \mathbb{E}^n | x_k = 0, \exists y \in \mathbb{R} : \boldsymbol{x} + y\boldsymbol{e}_k \in P\}. \end{aligned}$$

A very similar set is the *elimination* of x_k in P, defined as follows. Both $\text{proj}_k P$ and $\text{elim}_k P$ are shown in Figure 4.8.

$$\begin{aligned} \text{elim}_k P &:= \{\boldsymbol{x} - t\boldsymbol{e}_k | \boldsymbol{x} \in P\} \\ &= \{\boldsymbol{x} \in \mathbb{E}^n | \exists y \in \mathbb{R} : \boldsymbol{x} + y\boldsymbol{e}_k \in P\}. \end{aligned}$$

Clearly, given $P \subseteq \mathbb{E}^n$, there is an isomorphism

$$\text{elim}_k P \cong \text{proj}_k P \times \mathbb{E},$$

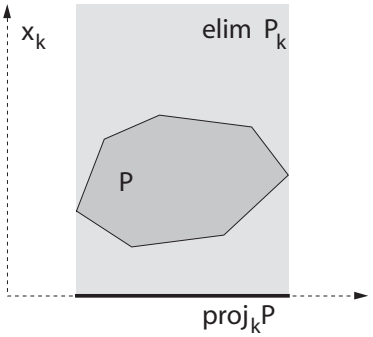

Figure 4.8 Polyhedron P and associated set $\text{proj}_k\, P$ and $\text{elim}_k\, P$

which reduces to equality when $k = n$ and the last coordinate of points in $\text{proj}_n\, P$ is dropped out:

$$\text{elim}_n\, P = \{\boldsymbol{x} \in \mathbb{E}^{n-1} | (\boldsymbol{x}, 0) \in \text{proj}_n\, P\}) \times \mathbb{E}$$

Example 4.4.2 (Projection operator)
A possible `PLaSM` implementation of a projection operator

$$\texttt{Project} : Z^+ \times \mathbb{E}^n \to \mathbb{E}^{n-m} : (m, P) \mapsto (\text{proj}_{n-m} \circ \cdots \circ \text{proj}_{n-1} \circ \text{proj}_n\,)P,$$

where Z^+ is the set of positive integers, is given in Script 4.4.9. The last expression shows that `Project:2`, when applied to a 5-dimensional object, returns a 3-dimensional object. The implementation clearly exploits the property that the projection of a collection $\{P(V_i)\}$ of $\mathcal{V}$-polytopes is the collection of convex hulls of the projected vertices:

$$\text{proj}_k\, \{P(V_i)\} = \{\text{proj}_k\, P(V_i)\} = \{\text{conv}\, (\text{proj}_k\, V_i)\}$$

Script 4.4.9 (Projection operator)

```
DEF Project (m::IsIntPos)(pol::IsPol) =
   (MKPOL ~ [AA:CutCoords ~ S1, S2, S3] ~ UKPOL):pol
WHERE
   CutCoords = Reverse ~ (COMP ~ #:m):TAIL ~ Reverse
END;

Project:2:(CUBOID:<1,1,1,1,1>) ≡ CUBOID:<1,1,1>
```

Fourier-Motzkin elimination The so-called *Fourier-Motzkin projection method*, also known as Fourier-Motzkin elimination, finds one solution, if it exists, of a system of simultaneous linear inequalities. In particular, this algorithm solves a system $\boldsymbol{A}^{(1)}\boldsymbol{x}^{(1)} \leq \boldsymbol{b}^{(1)}$ with n variables, by projecting it onto a system $\boldsymbol{A}^{(2)}\boldsymbol{x}^{(2)} \leq \boldsymbol{b}^{(2)}$, with $n-1$ variables, and so on, until a final projected system $\boldsymbol{A}^{(n)}\boldsymbol{x}^{(n)} \leq \boldsymbol{b}^{(n)}$ with only one

variable is obtained. A single inequality in only one variable is easy to study. If it cannot be solved, then the original system is *incompatible*, i.e. cannot be solved. Otherwise, a solution of $\boldsymbol{A}^{(n)}\boldsymbol{x}^{(n)} \leq \boldsymbol{b}^{(n)}$ is used in the backward phase of the algorithm to reconstruct a solution $\boldsymbol{x}$ to the original system. A detailed discussion of this algorithm is out the scope of the present book. The interested reader is referred to [DC73].

4.5 Simplicial complexes

We already met a definition of *simplex* as the convex hull of affinely independent points. A slightly different approach and notation are used in this section to discuss simplices and well-formed assemblies of simplices, called *simplicial complexes*, by using language and terminology from algebraic topology [Poi53]. At the end of this section we give a function to generate the so-called (after Maldelbrot [Man88]) *fractal simplex* of dimension d and depth n, with d, n arbitrary positive integers

4.5.1 Definitions

A notational warning is first needed. We use here simple subscripts both to denote dimension and indices. In case of possible confusion, parenthesized subscripts are used as indices.

Join operation The *join* of two sets $P, Q \subset \mathbb{E}^n$ is the set

$$PQ = \{\alpha\boldsymbol{x} + \beta\boldsymbol{y} |\ \boldsymbol{x} \in P, \boldsymbol{y} \in Q\},$$

where $\alpha, \beta \in \mathbb{R}$, $\alpha, \beta \geq 0$ and $\alpha + \beta = 1$. The join operation is associative and commutative.

Example 4.5.1 (`JOIN` primitive)
`PLaSM` offers a primitive operator from sequences of polyhedra in $\mathbb{E}^n$ to polyhedra in $\mathbb{E}^n$:

$$\texttt{JOIN} : (\mathcal{P}^{d_i,n})^* \to \mathcal{P}^{m,n} : \{P_1, \ldots, P_s\} \mapsto \operatorname{conv} P_1 \cup \ldots \cup P_s, \qquad i \in \{1, \ldots, s\},$$

where $m = \dim \operatorname{conv} P_1 \cup \ldots \cup P_s$. `JOIN` can be also applied to sequences of points in the same space, but they must be preliminarily transformed into 0-dimensional polyhedra.

Simplex A *d-simplex* $\sigma_d \subset \mathbb{E}^n$ $(0 \leq d \leq n)$ may be defined as the repeated join of $d + 1$ affinely independent points, called *vertices*. A d-simplex can be seen as a d-dimensional triangle: a 0-simplex is a *point*, a 1-simplex is a *segment*, a 2-simplex is a *triangle*, a 3-simplex is a *tetrahedron*, and so on.

The set $\{\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_d\}$ of vertices of σ_d is called the *0-skeleton* of σ_d. The s-simplex generated from *any* subset of $s+1$ vertices $(0 \leq s \leq n)$ of σ_d is called an *s-face* of σ_d.

Let us notice, from the definition, that a simplex may be considered both as a purely combinatorial object and as a geometric object, i.e. as the compact point set defined by the convex hull of a discrete set of points.

Complex A set Σ of simplices is called a *triangulation*. A *simplicial complex*, often simply denoted as *complex*, is a triangulation Σ that verifies the following conditions:

1. if $\sigma \in \Sigma$, then any face of σ belongs to Σ;
2. if $\sigma, \tau \in \Sigma$, then either $\sigma \cap \tau = \emptyset$, or $\sigma \cap \tau$ is a face of both σ and τ.

A simplicial complex can be considered a "well-formed" triangulation. Such kind of triangulations are widely used in engineering analysis, e.g., in topography or in finite element methods.

The *order* of a complex is the maximum order of its simplices. A complex Σ_d of order d is also called a *d-complex*. A d-complex is said to be *regular* or *pure* if each simplex is a face of a d-simplex. A regular d-complex is homogeneously d-dimensional.

The *combinatorial boundary* $\Sigma_{d-1} = \partial\sigma_d$ of a simplex σ_d is a simplicial complex consisting of all proper s-faces ($s < d$) of σ_d.

Two simplices σ and τ in a complex Σ are called *s-adjacent* if they have a common s-face. Hereafter, when we refer to adjacencies into a d-complex, we intend to refer to the maximum order adjacencies, i.e. to $(d-1)$-adjacencies. $\mathcal{K}_s$ ($s \leq d$) denotes the set of s-faces of Σ_d, and $|K_s|$ denotes the number of s-simplices.

With some abuse of language, we call (combinatorial) *s-skeletons* the sets $\mathcal{K}_s$ ($s \leq d$). *Geometric carrier* $|\Sigma|$, also called the *support space*, is the point set union of simplices in Σ.

Orientation The ordering of the 0-skeleton of a simplex implies an *orientation* of it. The simplex can be oriented according to the even or odd permutation class of its 0-skeleton. The two opposite orientation of a simplex will be denoted as $+\sigma$ and $-\sigma$. Two simplices are *coherently oriented* when their common faces have opposite orientation. A complex is *orientable* when all its simplices can be coherently oriented. It is assumed that:

1. the two orientations of a simplex represent its relative interior and exterior;
2. the two orientations of an orientable simplicial complex analogously represent the relative interior and exterior of the complex, respectively;
3. the boundary of a complex maintains the same orientation of the complex.

The volume associated with an orientation of a simplex (or complex) is positive, while the one associated with the opposite orientation has the same absolute value and opposite sign. It is assumed that the bounded object has positive volume. It is also assumed that either a minus sign or a multiplying factor -1 denote a complementation, i.e. an opposite orientation of the simplex, which can be explicitly obtained by swapping two vertices in its ordered 0-skeleton. For example:

$$\begin{aligned} +\sigma_3 &= \langle \boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3 \rangle \\ -\sigma_3 &= \langle \boldsymbol{v}_1, \boldsymbol{v}_0, \boldsymbol{v}_2, \boldsymbol{v}_3 \rangle \end{aligned}$$

Face extraction The oriented facets $\sigma_{d-1,(i)}$ ($0 \leq i \leq d$) of the oriented d-simplex $\sigma_d = +\langle \boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_d \rangle$ are obtained by removing the i-th vertex $\boldsymbol{v}_i$ from the 0-skeleton of σ_d:

$$\sigma_{d-1,(i)} = (-1)^i (\sigma_d - \langle \boldsymbol{v}_i \rangle), \qquad 0 \leq i \leq d. \tag{4.1}$$

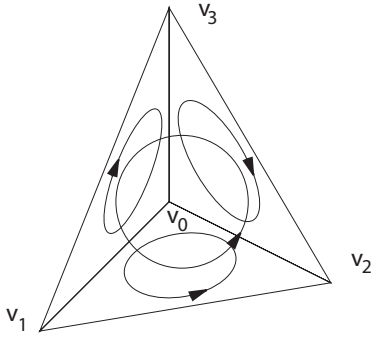

Figure 4.9 Coherent orientation of the faces of a 3-simplex

The 0-skeleton of $\sigma_{d-1,(i)}$ is therefore obtained by removing the i-th vertex from the 0-skeleton of σ_d and either by swapping a pair of vertices or, better, by inverting the simplex sign, when i is odd.

Finally, we will use the notation $\mathcal{A}(\sigma_{(i)})$ to denote the unique d-simplex which is $(d-1)$-adjacent to σ along the face $\sigma_{(i)}$, if it exists. When $\sigma_{(i)}$ is a boundary simplex, $\mathcal{A}(\sigma_{(i)})$ is not defined, which we denote by $\mathcal{A}(\sigma_{(i)}) = \perp$.

Oriented faces of a simplex According to equation (4.1), the set of 2-faces (see Figure 4.9) of the 3-simplex $\sigma_3 = +\langle v_0, v_1, v_2, v_3\rangle$ is: $\mathcal{K}_2(\sigma_3) = \{\sigma_{2,(0)}, \sigma_{2,(1)}, \sigma_{2,(2)}, \sigma_{2,(3)}\}$, where

$$\begin{aligned}
\sigma_{2,(0)} &= +\langle \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\rangle,\\
\sigma_{2,(1)} &= -\langle \boldsymbol{v}_0, \boldsymbol{v}_2, \boldsymbol{v}_3\rangle,\\
\sigma_{2,(2)} &= +\langle \boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_3\rangle,\\
\sigma_{2,(3)} &= -\langle \boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2\rangle.
\end{aligned}$$

Notice that all the triangle faces of the tetrahedron σ_3 are coherently oriented, and that, by using again the equation (4.1), the edges of triangles are generated coherently oriented.

For instance, taking $\sigma_{2,(0)} = +\langle \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\rangle$ and $\sigma_{2,(1)} = -\langle \boldsymbol{v}_0, \boldsymbol{v}_2, \boldsymbol{v}_3\rangle$, we see that their common faces $\sigma_{1,(0),(0)} = +\langle \boldsymbol{v}_2, \boldsymbol{v}_3\rangle$ and $\sigma_{1,(1),(0)} = -\langle \boldsymbol{v}_2, \boldsymbol{v}_3\rangle$, built according again to (4.1), have opposite orientations.

Simplicial prism The prism over a simplex $\sigma_d = \langle \boldsymbol{v}_0, \ldots, \boldsymbol{v}_d\rangle$, defined as the set $P_{d+1} := \sigma_d \times [a, b]$, with $[a, b] \subset \mathbb{E}$, will be called *simplicial* $(d+1)$*-prism*. An oriented complex which triangulates P_{d+1} can be defined combinatorially, by using a closed form formula for its $\mathcal{K}_{d+1}$ skeleton:

$$\mathcal{K}_{d+1} = \{\sigma_{d+1,(i)} = (-1)^{id}\langle \boldsymbol{v}_i^a, \boldsymbol{v}_{i+1}^a, \ldots, \boldsymbol{v}_d^a, \boldsymbol{v}_0^b, \boldsymbol{v}_1^b, \ldots, \boldsymbol{v}_i^b\rangle | 0 \le i \le d\}$$

where $\boldsymbol{v}_i^a = (\boldsymbol{v}_i, a)$ and $\boldsymbol{v}_i^b = (\boldsymbol{v}_i, b)$.

Implementation To generate a triangulation of the simplicial d-prism is quite easy, by considering that the simplices in $\mathcal{K}_{d+1} = \{\sigma_{d+1,(i)} | 0 \le i \le d\}$ of the previous

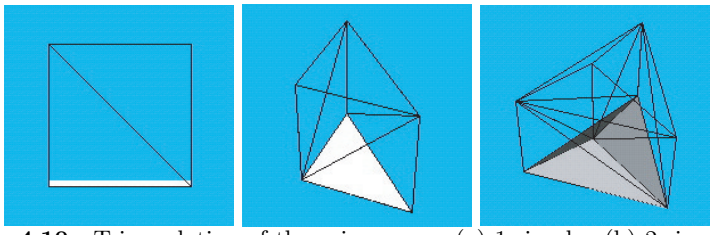

Figure 4.10 Triangulation of the prisms over: (a) 1-simplex (b) 2-simplex (c) 3-simplex, with the basis simplices highlighted.

formula can simply be obtained by extracting the adjacent $(d+2)$-tuples from the ordered set

$$(\boldsymbol{v}_0^a, \boldsymbol{v}_1^a, \ldots, \boldsymbol{v}_d^a, \boldsymbol{v}_0^b, \boldsymbol{v}_1^b, \ldots, \boldsymbol{v}_d^b)$$

with $2(d+1)$ elements.

The geometric object in Figure 4.10b is generated by the last expression of Script 4.5.1. The object shown in Figure 4.10c is first rotated in $\mathbb{E}^4$, then projected in $\mathbb{E}^3$. The `IsSimplex` predicate is given in Script 4.4.1.

Script 4.5.1 (Triangulation of simplicial d-prism)

```
DEF simplexPile (cell::IsSimplex) = (MKPOL ~ [ID, cells, pols]):verts
WHERE
   cells = TRANS ~ AA:FROMTO ~ ((CONS ~ AA:CONS ~ TRANS):
      < AA:K:(1..(d+1)), (AA:(- ~ [K:LEN, K]) ~ reverse):(0..d) >),
   pols = AA:LIST ~ INTSTO ~ K:d,
   verts = (CAT ~ AA:(S1 ~ UKPOL) ~ [ID, T:n:1] ~ EMBED:1): cell,
   d = DIM: cell + 1,
   n = RN: cell + 1
END;

(STRUCT ~ [@1 ~ simplexPile, ID] ~ SIMPLEX):2
```

Closed formulas to triangulate the $(d+1)$-prism over a d-complex in a time linear with the size of the output, while computing also the d-adjacencies between the resulting $(d+1)$-simplices, can be found in [FP91].

4.5.2 Linear d-Polyhedra

In some algebraic topology books a definition of d-polyhedron is given, that is quite different from the one discussed in Section 4.3.

In this case a *d-polyhedron* is defined as a compact set $P_d \subset \mathbb{E}^n$ for which at least a pair (Σ_d, h) exists, where Σ_d $(d \leq n)$ is a simplicial d-complex, and $h : P_d \to |\Sigma_d|$ is a homeomorphic map.[1]

Following the above definition, polyhedra are not necessarily linear; for example, a sphere or a torus or a free-form surface are polyhedra.

---

[1] A homeomorphic map is an invertible continuous topological transformation.

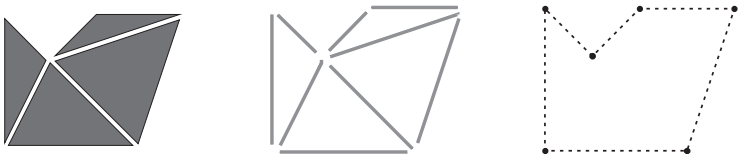

Figure 4.11 Skeletons $K_s(P_2)$, $(2 \geq s \geq 0)$

A polyhedron is *regular* if any associated complex is regular; it is *linear* if the map h is the identity map. Hence, a *linear polyhedron coincides with the geometric carrier of a simplicial complex* (see Figure 4.11). Since we deal only with linear polyhedra, we may use equally the terms polyhedron and complex.

Boundary If any $(d-1)$-simplex in Σ is a face of exactly two d-simplices, then Σ is said to be *closed*; otherwise it is said to be *open*. As an example, let us consider any 1-complex and any 2-complex which triangulate the circumference S_1 and the circle S_2, respectively: the first complex is closed; the second is open.

The *boundary* ∂P_d of a polyhedron P_d is the geometric carrier of the $(d-1)$-complex whose $(d-1)$-simplices are faces of exactly one d-simplex in a complex which triangulates P_d. An important theorem states that a closed complex has no boundary:

$$\partial\partial P_d = \emptyset.$$

A maximal $(d-1)$-connected component of a closed d-complex is called a *shell* of the complex.

4.5.3 Fractal d-simplex

We discuss in this section the generation of the *fractal d-simplex*, sometimes also called the *recursive d-simplex* or the *Sierpinski simplex*, that is generated by subtracting from the standard simplex of the same dimension a central portion defined by the convex hull of middle points of 1-faces.

This subtraction generates $d+1$ new d-simplices, each one adjacent to one of the vertices of the original d-simplex. The 1-faces of the new simplices have half-length with respect to the original ones. This process may be repeated on each of the $d+1$ d-simplices previously generated, so producing $(d+1)^2$ d-simplices. The subdivision can be repeated several times, producing $(d+1)^n$ d-simplices after n subdivision steps. They are spatially organized as a sort of d-dimensional fractal structure, as shown by Figures 4.12 and 4.13.

Implementation We implement here a `PLaSM` function `fractalSimplex`, which is used to produce the fractal d-simplex of any depth n.

Let us first define a `component` function which transforms the sequence of vertices of a simplex into a new sequence where the i-th vertex, called the `pivot`, is fixed, whereas the other are moved to the `Meanpoint` between the `pivot` position and their old position. The script also shows two examples of use of the `component` function on the vertices of the standard $\mathbb{E}^2$ `triangle`.

A local function named `expand` is used to transform the sequence of the $d+1$ vertices of a d-simplex into the sequence of sequences of vertices of the $d+1$ d-simplices

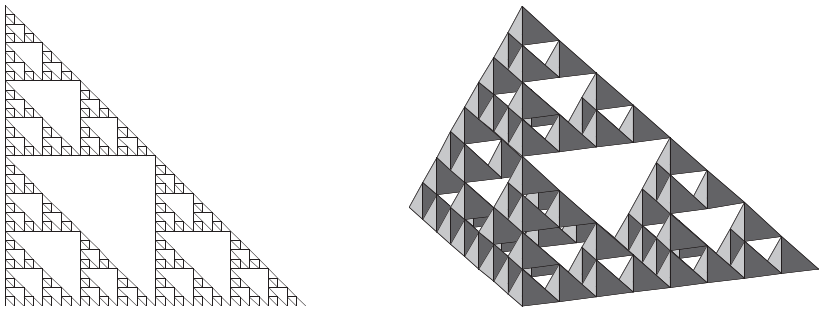

Figure 4.12 (a) Fractal triangle of depth 5 generated by `fractalSimplex:2:5` (b) Fractal tetrahedron of depth 3 generated by `fractalSimplex:3:3`

Script 4.5.2

```
DEF component (i::IsIntPos; seq::IsSeq) = CAT:< firstPart, <pivot>, lastPart >
WHERE
   firstSeq = AS:SEL:(1..(i - 1)):seq,
   pivot = SEL:i:seq,
   lastSeq = AS:SEL:((i + 1)..LEN:seq):seq,
   firstPart = (AA:Meanpoint ~ DISTR):< firstSeq, pivot >,
   lastPart  = (AA:Meanpoint ~ DISTR):< lastSeq,  pivot >
END;
DEF triangle = <<0.0, 0.0>, <0.0, 1.0>, <1.0, 0.0>>

component:<1, triangle> = <<0.0, 0.0>, <0.0, 0.5>, <0.5, 0.0>>
component:<3, triangle> = <<0.5, 0.0>, <0.5, 0.5>, <1.0, 0.0>>
```

generated by the `component` function, by using as pivot element each vertex of the input simplex.

We can finally define the operator `fractalSimplex`, shown in Script 4.5.3. This one will accept as input parameters the dimension d and the depth n of the fractal simplex to be generated. Notice that such a function is written in *pure* FL style, i.e. with neither recursion nor iteration. Conversely, a sort of *stream processing* is used here, where to all simplices generated at each step is applied a sort of *black box*, i.e. the `expand` function, that from each input simplex generates $(d+1)$ smaller d-simplices suitably positioned on its interior.

Script 4.5.3

```
DEF fractalSimplex (d::IsIntPos)(n::IsIntPos) =
   (mkpols ~ splitting ~ [S1] ~ UKPOL ~ SIMPLEX):d
WHERE
   mkpols = STRUCT ~ AA:MKPOL ~ AA:AL ~ DISTR ~ [ID,K:<<1..d+1>,<<1>>>],
   splitting = (COMP ~ #:n):(CAT ~ AA: expand),
   expand = AA:component ~ DISTR ~ [INTSTO ~ LEN, ID]
END;
```

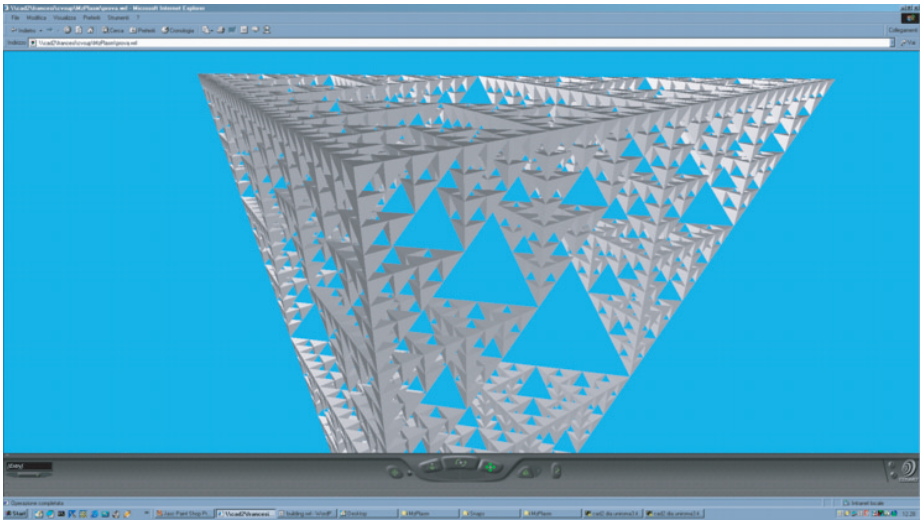

Figure 4.13 Polyhedral complex generated by `fractalSimplex:3:5`

4.6 Polyhedral complexes

Polyhedral complexes are very important in the context of geometric programming, since every object of geometric type in `PLaSM` is a polyhedral complex, possibly composed by just one convex cell, and (only implicitly) by all its faces.

Definition A *polyhedral complex* $\mathcal{C}$ is a finite collection of polyhedra of $\mathbb{E}^n$ such that:

1. if $P \in \mathcal{C}$, then $\mathcal{F}(P) \subseteq \mathcal{C}$;
2. if $P, Q \in \mathcal{C}$, then $P \cap Q \in \mathcal{F}(P)$ and $P \cap Q \in \mathcal{F}(Q)$.

A polyhedral complex $\mathcal{C}$ such that every element $P \in \mathcal{C}$ is a polytope is also called a *polytopal complex*. Actually, the geometric data structure used by `PLaSM`, and named HPC (Hierarchical Polyhedral Complex), is a finite collection of polytopal complexes, each of which satisfies the properties 1. and 2. given above.

The *dimension* $\dim \mathcal{C}$ of a polytopal complex is the highest dimension of its polytope elements. Analogously, the dimension of a collection $\mathcal{P} = \{\mathcal{C}_i\}$, made by polytopal complexes, is their highest dimension.

Polytopal subdivision and triangulation A *polytopal subdivision* of a set $P \subset \mathbb{E}^n$ is a polytopal complex $\mathcal{C} = \{C_i\}$ such that:

1. $\cup_i |C_i| = P$;
2. all the cells $C_i \in \mathcal{C}$ are polytopes.

A polytopal subdivision $\mathcal{T}$ of a set $P \subset \mathbb{E}^n$ is called a (well-formed) *triangulation* when all the cells in $\mathcal{T}$ are simplices or, equivalently, $\mathcal{T}$ is a simplicial complex.

Example 4.6.1 (Polytopal subdivision)
The primitive constructor `MKPOL` of geometric objects in `PLaSM` requires, in order

to work correctly, that the convex cells define a polytopal subdivision of the linear polyhedron we want to represent.

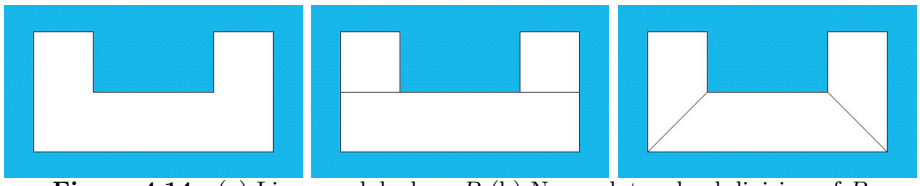

Figure 4.14 (a) Linear polyhedron P (b) Non-polytopal subdivision of P (c) Polytopal subdivision of P

We leave as an exercise for the reader the task of coping with three different definitions of the `complex` object, until the three images of Figure 4.14 from the expression:

```
(STRUCT ~ [ID, @1]): complex
```

are achieved.

Notational warning Through the whole book, we use the notation $\mathcal{P}^{d,n}$ to denote:

> the *space* $\mathcal{P}^{d,n}$ of geometric objects with *intrinsic* dimension d and *embedding* dimension n.

Now we are finally able to give a precise meaning to such expression:

1. "the *space* $\mathcal{P}^{d,n}$ of geometric objects" is the set of finite collections of polytopal subdivisions of linear polyhedra;
2. "with *intrinsic* dimension d" means that, if $P \in \mathcal{P}^{d,n}$, then $P = \{P_i\}$ is a collection of polytopal subdivisions P_i such that
$$d = \max\{\dim(\operatorname{aff} P_i) | P_i \in P\},$$
where $\operatorname{aff} P_i = \operatorname{aff} |P_i|$;
3. "and *embedding* dimension n" means that, if $P \in \mathcal{P}^{d,n}$, then $|P| \subset \mathbb{E}^n$.

For sake of simplicity, we make a further distinction between a (combinatorial) complex $P \in \mathcal{P}^{d,n}$ as a family (of faces) of polytopes, and its (geometric) support space $|P| \subset \mathbb{E}^n$, only in case of need. In both meanings we will normally use the notation P. The distinction should usually be clear from the context.

Furthermore, in the current implementation of the `PLaSM` language, the constraint that each P_i must constitute a polytopal subdivision is actually not enforced.

4.6.1 Schlegel diagrams

Definition The *Schlegel diagram* of a polytope $P \subset \mathbb{E}^d$, with $\dim P = d$, is defined as the polytopal complex $\mathcal{S}(P, F, \boldsymbol{y})$, of dimension $d-1$, which is obtained by projecting P onto a facet $F \in \mathcal{F}(P)$ from an external point $\boldsymbol{y}$.

In some sense, the Schlegel diagram $\mathcal{S}(P)$ encodes the combinatorial structure of a d-polytope P into a $(d-1)$-complex $\mathcal{S}(P)$.

Construction Let us consider a facet $F \in \mathcal{F}(P)$, and let H_F^+ be its support hyperspace, such that $P \subset H_F^+$. Let us also consider a point $\boldsymbol{y} \in H_F^-$, with its orthogonal projection within F, i.e. such that $\pi(\boldsymbol{y}) \in F$.

Let $G\boldsymbol{y}$, i.e. the *join* of face $G \in \mathcal{F}(P)$ and $\boldsymbol{y}$, be the pyramid with basis G and apex $\boldsymbol{y}$. Then the *Schlegel diagram* of P from $\boldsymbol{y}$ on $F \in \mathcal{F}(P)$ is precisely defined as the polytopal complex:

$$\mathcal{S}(P, F, \boldsymbol{y}) = \{G\boldsymbol{y} \cap H_F | G \in \mathcal{F}(P) \backslash \{F\}\}.$$

Implementation The easiest implementation is obtained by projecting the P polytope from the point $(0, \ldots, 0, r) \in \mathbb{E}^d$ into the coordinate hyperplane $\{\boldsymbol{x} | x_d = 0\}$. This choice will require, in most of cases, the preliminary application of some affine transformation to P to adjust its position and orientation in order to have some F facet close and parallel to the hyperplane $\{\boldsymbol{x} | x_d = 0\}$.

Such projection $\pi_d : \mathbb{E}^d \to \mathbb{E}^d$ may be represented in homogeneous coordinates, using standard graphics techniques (see Foley *et al.* [FvDFH90], p. 256), as a matrix $\boldsymbol{M}'_{per}$ which differs from the identity $(d+1) \times (d+1)$ just for the elements of the last two rows, and such that:

$$\begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ 0 & & & 0 & 0 \\ & & & 1/r & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{d-1} \\ x_d \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_{d-1} \\ 0 \\ x_d/r \end{pmatrix} \simeq \begin{pmatrix} \frac{x_1}{(x_d/r)} \\ \vdots \\ \frac{x_{d-1}}{(x_d/r)} \\ 0 \\ 1 \end{pmatrix}.$$

Such perspective transformation can be directly applied in `PLaSM` to a polyhedral domain by using the primitive operator `MAP`, as shown in Script 4.6.1, where two different operators `schlegel2D` and `schlegel3D` are given, to generate the diagrams of 3D and 4D polytopes, respectively. Notice that the center of projection is located at $(0, 0, 0.2)$ and at $(0, 0, 0, 0.2)$, respectively.

Script 4.6.1 (Schlegel diagrams)

```
DEF schlegel2D (d::isreal) = MAP:[s1/(s3/K:d), s2/(s3/K:d), K:0];
DEF schlegel3D (d::isreal) =
   project:1 ~ MAP:[s1/(s4/K:d), s2/(s4/K:d), s3/(s4/K:d), K:0];
```

The three Schlegel diagrams in Figure 4.15, with the 1-skeletons of the 4-simplex Δ_4, of the 4-dimensional cube and of the 4-polytope $\Delta_2 \times \Delta_2$, are generated by the three expressions of Script 4.6.2, respectively.

Script 4.6.2 (Schlegel diagrams examples (1))

```
schlegel3D:0.2: ((@1 ~ T:<1,2,3,4>:<-1,-1,-1,1> ~ CUBOID):<2,2,2,2>);
schlegel3D:0.2: ((@1 ~ T:<1,2,3,4>:<-1/3,-1/3,-1,1>):(SIMPLEX:4));
schlegel3D:0.2: ((@1 ~ T:<1,2,3,4>:<-1/3,-1/3,-1,1>):(SIMPLEX:2 * SIMPLEX:2))
```

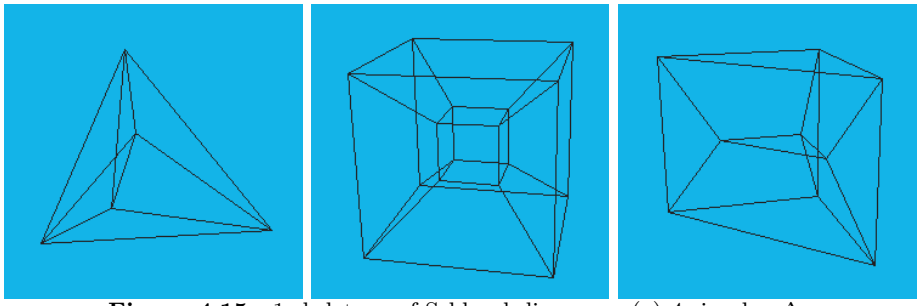

Figure 4.15 1-skeletons of Schlegel diagrams: (a) 4-simplex Δ_4 (b) 4-dimensional cube (c) 4-polytope $\Delta_2 \times \Delta_2$

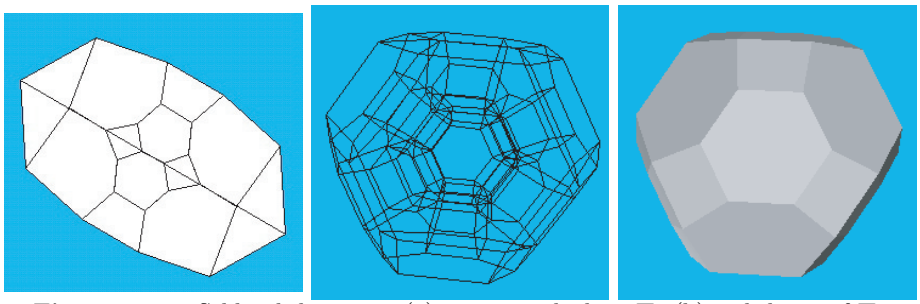

Figure 4.16 Schlegel diagrams: (a) 3-permutahedron Π_3 (b) 1-skeleton of Π_4 (c) 4-permutahedron Π_4

Analogously, the Schlegel diagrams of the 3- and 4-permutahedron given in Figure 4.16 are produced by the three expressions of Script 4.6.3.

Script 4.6.3 (Schlegel diagrams examples (2))

```
(STRUCT ~ [ID,@1] ~ schlegel2D):0.2:((@2 ~ T:3:2.5 ~ permutahedron):3);
schlegel3D:0.2: ((@1 ~ T:4:5):(permutahedron:4));
schlegel3D:0.2: ((@2 ~ T:4:5):(permutahedron:4));
```

4.7 Nef polyhedra

Nef polyhedra are introduced and shortly discussed in this section. This very general definition of polyhedra allows representation of a very large class of piecewise-linear point sets. Nef's concept of polyhedron appears to meet the needs of solid modeling surprisingly well: Nef polyhedra can have internal boundaries, they can be neither closed nor open, as well as non-regular and non-manifold (see Section 13.1.1). Moreover, a complete and mathematically sound theory is contained in the original Nef's work [Nef78]. Such point sets provide a a powerful modeling space for the purpose of (piecewise-linear) solid modeling and related applications (see Chapter 13). Our main references for the present section are Bieri [Bie94b, Bie94a] and Ferrucci [Fer95a, Fer95b].

A Nef polyhedron can be defined in several equivalent ways. In particular:

Definition A set $P \subseteq \mathbb{E}^n$ is a *Nef polyhedron* if and only if one of the following properties is satisfied:

1. P is obtainable by a finite number of intersections and complements of open affine halfspaces;
2. P is generated by any finite number of union, intersection and difference operations between closed affine subspaces;
3. there exist finitely many relatively closed open sets $\{A_i\}$ and $\{B_j\}$ such that $P = \cup_i A_i$ and complement $P = \cup_j B_j$;
4. there exists a family $\{H_k\} \subset (\mathbb{E}^n)^*$ of hyperplanes such that P is the union of the arrangement of $\mathbb{E}^n$ generated by $\{H_k\}$.

4.7.1 Locally adjoined pyramids

A central concept in Nef polyhedra is that of a pyramid around a point $\boldsymbol{x} \in \mathbb{E}^n$, which is called *locally adjoined pyramid* to polyhedron P in $\boldsymbol{x}$, in short l.a. pyramid, and is denoted as $P^{\boldsymbol{x}}$. It is defined as a cone with apex in $\boldsymbol{x}$ and directions defined by vectors internal to some open neighborhood of $\boldsymbol{x}$ in P:

$$P^{\boldsymbol{x}} := \{\boldsymbol{x} + \lambda(\boldsymbol{y} - \boldsymbol{x}) | \boldsymbol{y} \in N_\epsilon(\boldsymbol{x}, P), \lambda > 0\},$$

where $N_\epsilon(\boldsymbol{x}, P) := \{\boldsymbol{y} \in P | d(\boldsymbol{x}, \boldsymbol{y}) < \epsilon\}$ is an ϵ-neighborhood of $\boldsymbol{x}$ in P.

Properties of pyramids Locally adjoined pyramids enjoy several interesting properties, that translate all topological tests on points and polyhedra, often called *point set membership* tests, into a suitable set-theoretical characterization of the pyramid adjoined to the considered point. In particular:

1. If a point $\boldsymbol{x}$ is in P, then it is also in $P^{\boldsymbol{x}}$:
$$\boldsymbol{x} \in P \quad \Longrightarrow \quad \boldsymbol{x} \in P^{\boldsymbol{x}}$$
2. The pyramid adjoined to an interior point is the whole space, in the hypothesis that $\dim P = n$:
$$\boldsymbol{x} \in \operatorname{int} P \quad \Longleftrightarrow \quad P^{\boldsymbol{x}} = \mathbb{E}^n$$
3. Otherwise, if $\dim P < n$, then:
$$\boldsymbol{x} \in \operatorname{relint} P \quad \Longleftrightarrow \quad P^{\boldsymbol{x}} = \operatorname{aff} P$$
4. The converse property states that the pyramid adjoined to exterior points is empty:
$$\boldsymbol{x} \in \operatorname{ext} P \quad \Longleftrightarrow \quad P^{\boldsymbol{x}} = \emptyset$$
5. Also, the adjoined pyramid is neither empty nor the whole space when the point is on the boundary:
$$\boldsymbol{x} \in \partial P \quad \Longleftrightarrow \quad (P^{\boldsymbol{x}} \neq \emptyset) \wedge (P^{\boldsymbol{x}} \neq \mathbb{E}^n)$$

6. Finally, an adjoined pyramid is not empty if and only if the considered point is either on the boundary or on the interior of P:

$$\boldsymbol{x} \in \operatorname{clos} P \quad \Longleftrightarrow \quad P^{\boldsymbol{x}} \neq \emptyset$$

Example 4.7.1 (Nef polyhedron)
An example of Nef polyhedron is given by the geometric value generated by evaluating the `nef_pol` symbol in Script 4.7.1. The generated set is shown in Figure 4.17. The example is aimed to show that a Nef polyhedron may be non-regular, i.e. may be dimensionsionally unhomogeneous. In particular, it is easy to see that the following hold:

$\dim$ (`basis JOIN apex`) = 3,
$\dim$ (`apex JOIN top`) = 1,
$\dim$ (`flag`) = 2,

respectively.

Script 4.7.1 (Example of Nef polyhedron)

```
DEF basis = (T:<1,2>:<-1,-1> ~ CUBOID):<2,2,1>;
DEF apex  = MKPOL:<<<0,0,2>>,<<1>>,<<1>>>;
DEF top   = MKPOL:<<<0,0,3>>,<<1>>,<<1>>>;
DEF flag  = (T:3:2.5 ~ R:<2,3>:(PI/2) ~ EMBED:1 ~ MKPOL):<
   <<0,0>,<0.5,0>,<0.5,0.5>,<0,0.5>,<0.25,0.25>>,
   <<1,2,5>,<3,4,5>,<1,4,5>>, <1..3> >;

DEF nef_pol = STRUCT:< basis JOIN apex, apex JOIN top, flag >;
```

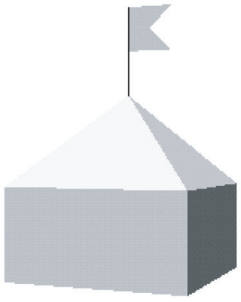

Figure 4.17 Example of Nef polyhedron

4.7.2 Faces of Nef polyhedra

The most interesting aspect of Nef polyhedra is probably that they support a definition of face which is both mathematically sound and intuitively appealing. In particular,

faces are defined by Nef as equivalence classes of points with the same adjoined pyramid.

Definition Let $P \subseteq \mathbb{E}^n$ be a Nef polyhedron. A *face* of P is an equivalence class of the relation $\sim$, where for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{E}^n$:

$$\boldsymbol{x} \sim \boldsymbol{y} \quad \Longleftrightarrow \quad P^{\boldsymbol{x}} = P^{\boldsymbol{y}}.$$

Closure of operations The class of Nef polyhedra is closed under the Boolean operations of *union*, *intersection*, *difference* and *complementation*. It is also closed under the topological operations of *interior*, *boundary* and *closure*. So, Nef polyhedra define the best class of mathematical models (see Chapter 13) for piecewise-linear solid modeling, and more in general for geometric objects generated by selecting some not necessarily regular or connected subset of cells from the arrangement, i.e. the space partition, generated by an arbitrary set of affine hyperplanes. Their great usefulness for applications in graphics and modeling has been recognized quite recently, so that general purpose C++ libraries are yet under development.

Caveat The reader should notice that, whereas it is possible to generate with `PLaSM` the quite large class of non-regular and unconnected but *closed* Nef polyhedra, yet this class is currently not closed under Boolean set operations. The current `PLaSM` implementation of such operations (see Section 14.2) in fact requires the operand objects being regular and full dimensional. Anyway, both the algorithm and the data structures used to this purpose may be (hopefully) extended to process the whole class of Nef polyhedra in arbitrary dimensions, using two basic concepts of solid modeling, i.e. simplicial complexes and ternary space partitioning trees (extension of BSP-trees — see Section 13.3.2) like those used by Vaněček [Van91], enriched with a cell selector function, as suggested by Ferrucci [Fer95a, Fer95b].

4.8 Linear programming

Linear programming methods are extensively used in `PLaSM`, mainly in the implementation of dimension-independent Boolean operations, and in the internal conversions between facet-based and vertex-based representations of polyhedral complexes. So we give here a brief introduction to such methods. The interested reader is referred to [Chv83] and [Mur83]. A more abstract approach, devoted to proving the polynomial time solvability of geometric problems with combinatorial optimization is [GLS88].

Mathematical programming A mathematical programming problem is formulated as the search for

$$\min\{f(\boldsymbol{x}) | \boldsymbol{x} \in X\}$$

where X is the set of *feasible solutions*, and $f : X \to \mathbb{R}$ is called *objective function*. When $X = \emptyset$, the problem is said to be impossible or *unfeasible*; when the objective function is inferiorly unlimited, i.e. when $\min\{f(\boldsymbol{x}) | \boldsymbol{x} \in X\} = -\infty$, the problem is

said *unbounded*. An *optimal solution* is a point $\boldsymbol{x}^* \in X$ such that $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$ for every $\boldsymbol{x} \in X$.

LP problems A mathematical programming problem is called a *linear programming* (LP) problem when it is of the form

$$\min\{\boldsymbol{c}^T\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq 0\}.$$

The above form of the LP problem, where the feasible set is the intersection of the *convex polyhedron* $\{\boldsymbol{x}|\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}\}$ with the cone $\{\boldsymbol{x} \geq 0\}$, is called the *general form*. The LP problem can also be given in *standard form*

$$\min\{\boldsymbol{c}^T\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0\}.$$

where the feasible set is the intersection of the *affine manifold* $\{\boldsymbol{x}|\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ with the cone $\{\boldsymbol{x} \geq 0\}$, and in particular, is an intersection of half-flats.

Due to the economical interpretations of the LP problems, $\boldsymbol{c}$ and $\boldsymbol{b}$ are called *cost* and *resource* vector, respectively, $\boldsymbol{A}$ is the matrix of (technological) *constraints*, and $\boldsymbol{x}$ is the vector of *decision variables*.

Equivalent formulations Several conversions between equivalent problems can be given. In particular:

1. from max to min:

$$\max \boldsymbol{d}^T\boldsymbol{x} \quad \Longrightarrow \quad -\min \boldsymbol{c}^T\boldsymbol{x}$$

   with $\boldsymbol{c} = -\boldsymbol{d}$;
2. from *below* ($\leq$) constraints to equality constraints, by introducing *slack* variables s_i:

$$\boldsymbol{a}_i\boldsymbol{x} \leq \boldsymbol{b}_i \quad \Longrightarrow \quad \left\{ \begin{array}{rcl} \boldsymbol{a}_i\boldsymbol{x} + s_i & = & \boldsymbol{b}_i \\ s_i & \geq & 0 \end{array} \right.$$

3. from *above* ($\geq$) constraints to equality constraints, by introducing *surplus* variables s_i:

$$\boldsymbol{a}_i\boldsymbol{x} \geq \boldsymbol{b}_i \quad \Longrightarrow \quad \left\{ \begin{array}{rcl} \boldsymbol{a}_i\boldsymbol{x} - s_i & = & \boldsymbol{b}_i \\ s_i & \geq & 0 \end{array} \right.$$

4. from equality constraints to *above* ($\geq$) constraints:

$$\boldsymbol{a}_i\boldsymbol{x} = \boldsymbol{b}_i \quad \Longrightarrow \quad \left\{ \begin{array}{rcl} \boldsymbol{a}_i\boldsymbol{x} & \geq & \boldsymbol{b}_i \\ -\boldsymbol{a}_i\boldsymbol{x} & \geq & -\boldsymbol{b}_i \end{array} \right.$$

5. from sign unconstrained variables to sign constrained variables:

$$x_i \lesseqgtr 0 \quad \Longrightarrow \quad \left\{ \begin{array}{rcl} x_i & = & x_i^+ - x_i^- \\ x_i^+ & \geq & 0 \\ x_i^- & \geq & 0 \end{array} \right.$$

4.8.1 Geometry of linear programming

As we know, a (convex) *polyhedron* is defined as the intersection of finitely many affine subspaces and hyperplanes. A *polytope* is a bounded polyhedron. A *vertex* of a polyhedron P is a point which cannot be obtained by convex combination of other points. The fundamental *theorem of Minkowski/Weil* states that each point of a polytope can be obtained by convex combination of its vertices. As an easy corollary, it is possible to show that if the feasible set P is non-empty and inferiorly bounded, then there exists at least one optimal vertex, where the objective function gets its minimum value.

Algebraic characterization of vertices Let us suppose $\boldsymbol{A} \in \mathbb{R}_n^m$, i.e. that there are m constraints and n variables, with $m < n$. So, it is possible to give an arbitrary value, zero in particular, to $n-m$ unknowns and to solve uniquely for the others, while at the same time guaranteeing that the solution, called the *basic feasible* solution, is a vertex of the feasible polyhedron P.

To characterize algebraically the vertices of $P = \{\boldsymbol{x} \geq 0 | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ is quite easy. Let us consider a collection $\boldsymbol{B}$ of m linearly independent columns of $\boldsymbol{A} = (\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n)$, and suppose, for the sake of simplicity, that

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{B} & \boldsymbol{N} \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} \boldsymbol{a}_1, \ldots, \boldsymbol{a}_m \end{pmatrix}, \quad \boldsymbol{N} = \begin{pmatrix} \boldsymbol{a}_{m+1}, \ldots, \boldsymbol{a}_n \end{pmatrix}.$$

The variables $\boldsymbol{x}_B$ associated to $\boldsymbol{B}$ are called *basic* variables; the others $\boldsymbol{x}_N$ are said *non-basic*, and let

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_B \\ \boldsymbol{x}_N \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}^{-1}(\boldsymbol{b} - \boldsymbol{N}\boldsymbol{x}_N) \\ \boldsymbol{x}_N \end{pmatrix}$$

be a solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. The solution $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_B & \boldsymbol{0} \end{pmatrix}^T$ is said to be the *basic solution* associated to the $\boldsymbol{B}$ basis. A basic solution is *feasible* if

$$\boldsymbol{x}_B = \boldsymbol{B}^{-1}\boldsymbol{b} \geq \boldsymbol{0}.$$

By extension, the $\boldsymbol{B}$ basis associated with a basic feasible solution is also called a *feasible* basis.

A fundamental theorem states that $\boldsymbol{x}$ is a vertex of the convex polyhedron $P := \{\boldsymbol{x} \geq 0 | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ if and only if $\boldsymbol{x}$ is a *basic feasible solution* of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

Also, each problem $\min\{\boldsymbol{c}^T\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0\}$ with a non-empty and bounded, say inferiorly limited, feasible set has at least one optimal solution that coincides with a basic feasible solution.

4.8.2 Simplex method

The well-known *simplex method* to solve LP problems is due to Dantzig [Dan51, Dan63]. The simplex method can be geometrically summarized as follows: in a first phase, a vertex $\boldsymbol{v}$ of the feasible set is chosen; then, in a second phase, the algorithm moves stepwise to an adjacent vertex $\boldsymbol{w}$ where $f(\boldsymbol{w})$ is less or equal to the previous value $f(\boldsymbol{v})$, and repeats the move until an optimal vertex is reached. The existence of a decreasing path from each vertex $\boldsymbol{v}$ to the optimal face $F_0 = \{\boldsymbol{x}^* | f(\boldsymbol{x}^*) = z_0\}$, with $z_0 = \min\{\boldsymbol{c}^T\boldsymbol{x} | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq 0\}$, is always guaranteed.

Tableau-based simplex algorithm The *simplex algorithm* is the most well known and practically efficient resolution procedure for linear programs in standard form.

There are several variations of the simplex algorithm [Mur83], to solve efficiently in practical cases very large-scale LP problems [Nem96], say, problems with hundreds of thousands, and even millions, variables and equations.

Some of such methods use the so-called *simplex tableau*, where the data of the problem are organized into a two-dimensional array, and where the main operation is the *pivot* operation, corresponding to Gauss' elimination of one variable. At each suitably performed pivot operation, one of the variables leaves the basis, and another one enters the basis, so moving the current basic feasible solution from the current P vertex to one of its neighborhoods, until an optimal basic solution is found.

Pivoting operation Let us suppose a matrix $\boldsymbol{A} = \left(\; a_{i,j} \; \right) \in \mathbb{R}_n^m$ representing a linear system of m equation in n variables is given, and let $a_{h,k} \neq 0$. A pivoting operation on the matrix $\boldsymbol{A}$, with *pivot element* of indices (h, k) is aimed at eliminating the k-th variable from all equations, but not from the h-th one. The effect on the resulting matrix is the appearance of a unit k-th column, with the 1 element in position (h, k), and with 0 elsewhere.

For this purpose, rows (i.e. equations) are summated to linear combinations of other rows, so that the solutions of the system do not change. In particular, the h-th row is divided for the pivot element $a_{h,k}$, supposed non-zero, while the other rows, for example the i-th, are subtracted by the *new* h-th row times the element on the i-th position of the pivot column. More formally, let denote as $\boldsymbol{A}' = \left(\; a'_{i,j} \; \right) = \left(\; \boldsymbol{a}'_i \; \right)$ the resulting matrix after pivoting on the (h, k) element. Then we have:

$$\boldsymbol{a}'_h := \frac{1}{a_{h,k}} \boldsymbol{a}_h, \tag{4.2}$$

$$\boldsymbol{a}'_i := \boldsymbol{a}_i - a_{i,k} \boldsymbol{a}'_h, \qquad i \neq h \tag{4.3}$$

Example 4.8.1 (Pivoting operation)
Looking carefully at equations (4.2–4.3), it is easy to see that the result $\boldsymbol{A}'$ of the pivot operation about the element (h, k) on the matrix $\boldsymbol{A} \in \mathbb{R}_n^m$ can be obtained as

$$\boldsymbol{A}' = \boldsymbol{P}(h, k) \, \boldsymbol{A}$$

where the matrix $\boldsymbol{P}(h, k) \in \mathbb{R}_m^m$ differs from the identity $m \times m$ only on the h-th column:

$$\boldsymbol{P}(h,k) = \begin{pmatrix} 1 & & & -\frac{a_{1,k}}{a_{h,k}} & & \\ & 1 & & -\frac{a_{2,k}}{a_{h,k}} & & \\ & & \ddots & \vdots & & \\ & & & \frac{1}{a_{h,k}} & & \\ & & & \vdots & \ddots & \\ & & & -\frac{a_{m,k}}{a_{h,k}} & & 1 \end{pmatrix}$$

The pivoting matrix can be generated in PLaSM by using the Pivot function given in Script 4.8.1. The body of the function just substitutes the updated_column to the h-th column of the identity $m \times m$. The function IDNT used to generate the identity matrix is given in Script 3.3.5. The scalarVectProd operator, to execute a product of a scalar times a vector, is given in Script 2.1.20. Notice that the update function updates the n-th element of the seq sequence with the new value x, of any type. To understand the Pivot code, the reader should remember that a matrix in PLaSM is represented as a sequence of rows.

Script 4.8.1 (Pivot matrix)

```
DEF update (n::IsIntPos)(seq::IsSeq)(x::TT) = CAT:<
   (CONS ~ AA:SEL):(1..n - 1):seq,
   < x >,
   (CONS ~ AA:SEL):(n + 1..LEN:seq):seq
>;

DEF Pivot (h,k::IsIntPos) (mat:: IsMat) =
   (TRANS ~ update:h:(IDNT:m)): updated_column
WHERE
   pivot_column  = SEL:k: (TRANS:mat),
   pivot_element = SEL:h: pivot_column,
   m = LEN:mat,
   updated_column = (-1/pivot_element) scalarVectProd update:h:pivot_column:-1
END;
```

Notice that when computing $\boldsymbol{A}' = \boldsymbol{P}(h,k)\,\boldsymbol{A}$, the matrix $\boldsymbol{A}$ is not necessarily squared, whereas the pivot matrix $\boldsymbol{P}(h,k)$ is squared. Two examples of the operator

$$\texttt{Pivot} : (Z^+ \times Z^+) \to (\mathbb{R}_n^m \to \mathbb{R}_m^m)$$

are given in Scripts 4.8.2 and 4.8.3.

Script 4.8.2 (Pivot matrix)

```
Pivot:<2,5>:
< < 11 , -12 ,  13 ,  14 ,  15 , 16 > ,
  < 21 ,  22 , -23 ,  24 ,  25 , 26 > ,
  < 31 ,  32 ,  33 , -34 ,  35 , 36 > ,
  < 41 ,  42 ,  43 ,  44 , -45 , 46 > >
≡
< < 1 , -3/5 , 0 , 0 > ,
  < 0 , 1/25 , 0 , 0 > ,
  < 0 , -7/5 , 1 , 0 > ,
  < 0 ,  9/5 , 0 , 1 > >
```

The effect of a double pivoting on an input matrix is shown in Script 4.8.3. Notice that an infix expression like (Pivot:<2,5> * ID) returns a function, which can be compound with other similar functions.

Script 4.8.3 (Pivoting example)

```
((Pivot:<3,1> * ID) ~ (Pivot:<2,5> * ID)):
< < 11 , -12 ,  13 ,  14 ,  15 , 16 > ,
  < 21 ,  22 , -23 ,  24 ,  25 , 26 > ,
  < 31 ,  32 ,  33 , -34 ,  35 , 36 > ,
  < 41 ,  42 ,  43 ,  44 , -45 , 46 > >
≡
< < 0 ,  -24 ,      92 ,    -68 , 0 ,     0 > ,
  < 0 ,  1/4 , -703/20 , 729/20 , 1 ,   5/4 > ,
  < 1 ,  3/4 ,   163/4 , -169/4 , 0 ,  -1/4 > ,
  < 0 , 45/2 , -6419/2 , 6833/2 , 0 , 225/2 > >
```

4.8.3 Some polyhedral algorithms

Two algorithmic problems are of major interest when using decompositions with d-dimensional polytopes, namely the *vertex enumeration* and the *facet enumeration* problem. We just address here the main coordinates of such problems. For a deep review and a wide survey of the field the interested reader is referred to Avis, Bremner and Seidel's article entitled "How good are convex hull algorithms?" [ABS97].

Beneath/Beyond method

The convex hull of a finite set of points in $\mathbb{E}^n$, for arbitrary positive integer n, can be computed by using Seidel's "Beneath Beyond" method [Sei81], described by Edelsbrunner in [Ede87]. This method is a well-known, simple and efficient algorithm implemented by several geometric codes. It is also used by the `PLaSM` geometric kernel in the implementation of the the basic primitive `MKPOL`.

In an initialization step of the algorithm the input points are sorted lexicographically. Then the convex hull is incrementally built, starting from the empty set and adding a sorted point at a time. In doing so, a distinction is made between the cases where the affine hull of the previous points either contains or does not contain the added point.

The second case is much easier: the updated convex hull is a pyramid having the previous convex hull as the basis and the new point as the apex. All the faces of a pyramid are easily obtained by the join of the apex to all faces of the basis. The highest dimensional face is obtained by the join of the apex to the basis.

The second case, requiring a non-pyramidal update, is quite more complex. A detailed description can be found in Edelsbrunner [Ede87] and in Preparata and Shamos [PS85].

Vertex enumeration problem

Given a polytope P, represented as an intersection of halfspaces $\mathcal{H}(P) = P(\boldsymbol{A}, \boldsymbol{b}) = \{\boldsymbol{x} \in \mathbb{E}^n | \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, the generation of its representation as the convex hull of its vertices $\mathcal{V}(P)$, is called the *vertex enumeration problem*. Several solution algorithms exist for this problem, that can still be considered a research problem in computational geometry.

The solution algorithms can be classified as *pivotal* methods and *progressive* methods. The first class makes use of the pivot operation as the basic operation to move from one vertex to one of the vertices adjacent to it. Unfortunately, it is not possible to consider each vertex only one time. This is possible only if the graph associated with the polyhedron has a Hamiltonian cycle, and this is in general not true. Such algorithms may either pass more than one time on each vertex or visit points which are external to the polyhedron.

We discuss here both a basic trivial method and the solution currently implemented in `PLaSM`.

Trivial approach The easiest way of solving the vertex enumeration problem consists in:

1. considering each n-tuple of boundary hyperplanes $H_{\boldsymbol{a}_i,b_i} = \{\boldsymbol{x} \in \mathbb{E}^n | \boldsymbol{a}_i^T \boldsymbol{x} = b_i\}$;
2. solving their squared system for the common point, say $\boldsymbol{y}$;
3. checking for *feasibility*, i.e. if $\boldsymbol{y} \in P(\boldsymbol{A}, \boldsymbol{b})$. In other words, it is necessary to verify if $\boldsymbol{Ay} \leq \boldsymbol{b}$. Clearly, this is true if and only if $\boldsymbol{y}$ is a vertex of P.

Let us suppose $\boldsymbol{A} \in \mathbb{R}_n^m$, i.e. that there are m inequalities in $\mathcal{H}(P)$, with $m \geq n$. This method requires the solution of a number $\binom{m}{n}$ of $n \times n$ systems of linear equations. A further computational cost is given by the feasibility check $\boldsymbol{Ay} \leq \boldsymbol{b}$ for each solution $\boldsymbol{y}$. Such method may be acceptable only for very small values of both m and n.

Example 4.8.2 (Vertex enumeration)
It is easy to see that the trivial generation of vertices of 3D tetrahedron would require $\binom{4}{3} = 4$ resolutions of 3×3 systems of linear equations, i.e. one for each vertex, because each triplet of face hyperplanes generates a vertex.

The vertex enumeration of the 3D cube would require solving $\binom{6}{3} = 20$ linear systems, and only 8 solutions would actually pass the feasibility test.

Conversely, the same operation for the 3D icosahedron (see Section 23) would require the resolution of $\binom{20}{3} = 1140$ linear systems of dimension 3×3. In this case only 12 solutions would be accepted as icosahedron vertices.

Pivoting algorithms One of best known pivoting methods, named after its authors M. Manas and J. Nedoma [MN68], starts from the simplex tableau in canonical form, which gives the first vertex of the polyhedron, and explores the set of adjacent bases associated to basic feasible solutions, thus constructing a covering tree of the graph described below, until the set of all vertices has been built.

Let $P = \{\boldsymbol{x} \geq \boldsymbol{0} | \boldsymbol{Ax} = \boldsymbol{b}\}$ be in standard form. A graph $G = (N, A)$ with nodes corresponding to m-tuples of indices associated to feasible bases, and arcs between nodes which differ in just one component, can be built incrementally moving from each vertex to its adjacent vertices. Visiting a node corresponds to computing the Cartesian coordinates of the polyhedron vertex associated with a basic feasible solution of the LP problem. The algorithm is initialized by putting the simplex tableau in canonical form, thus achieving an initial node of the graph and a first polyhedron vertex.

David Avis's *lrslib* is a self-contained ANSI C implementation of the reverse search algorithm [AF92] for vertex enumeration/convex hull problems. A more efficient

method of finding all the vertices of a polytope without using any slack/surplus variables is given in Arsham [Ars97].

Progressive algorithms More recent and efficient algorithms make direct use of the polyhedron description as a set of inequalities, thus avoiding increasing the number of problem variables, and compute the set of vertices by repeatedly adding one inequality. Such algorithms often start from the simplex generated from the first n inequalities, then update the current set of vertices by suitable comparisons to the boundary hyperplane associated to the added inequality. New vertices must obviously be added when such a new hyperplane cuts the current polyhedral set. A combined solution of both enumeration problems (say, of vertices and facets enumeration) is given by Bremner, Fukuda and Marzetta in [BFM98].

4.9 Examples

From the very first development of Greek geometry, some interesting solids intrigued the mathematicians because of the extreme perfection of their definition. In particular, five polyhedra, bounded by regular polygons which are all equal to each other, and are joined by equal internal angles, are cited in Plato's work *Timaeus*, so that they are collectively known as *Platonic solids*. The construction of Platonic solids is the last topic in Euclid's *Elements* of geometry book. We present in this section a straightforward `PLaSM` modeling of the five Platonic solids.

4.9.1 Platonic solids

We discuss here the construction of *tetrahedron*, *hexahedron*, *octahedron*, *dodecahedron* and *icosahedron*, all inscribed in a sphere of unit radius, following the lines developed in the second chapter of the beautiful and inspiring book *Polyhedra* by P. Cromwell [Cro97]. Such solids are illustrated in Figure 4.18.[2]

Tetrahedron

The tetrahedron as a polytope, bounded by four triangle faces and having six edges and four vertices, is well known to our reader. Tetrahedra are commonly produced in `PLaSM` by the `SIMPLEX` primitive applied to the integer 3, and are generated as the convex hull of the set $\{\boldsymbol{o}, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$. Such unit 3-simplex lacks the symmetries that the Platonic tetrahedron enjoys, and cannot be inscribed in the unit sphere.

Hence, the generation method of tetrahedron given in Script 4.9.1 closely resembles its construction in the Euclid's *Elements*, as reported by Cromwell [Cro97]. In particular, the solid is produced by the `JOIN` of the equilateral triangle of the basis, inscribed in the unit circle centered at the origin, with the apex vertex located perpendicularly along the z axis, at a distance $\frac{4}{3}$ from the origin. The `ngon` function, used to generate regular polygons with any number of sides, is given in Script 4.4.2.

---

[2] The quality of images is quite poor, because they were produced by using a standard web browser, and not by using a raytracer.

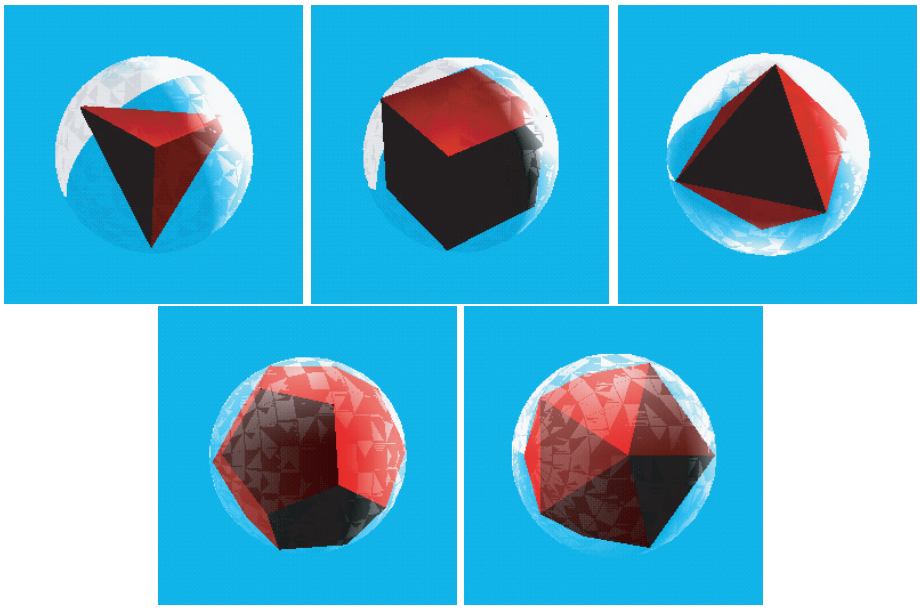

Figure 4.18 *Tetrahedron*, *hexahedron*, *octahedron*, *dodecahedron* and *icosahedron* inscribed in a unit sphere

In order to inscribe the generated polyhedron in the unit sphere we need to move it in the reverse z direction, with translation vector $(0, 0, -\frac{1}{3})$. The same effect is obtained by translating the basis, and joining it to the set $\mathrm{conv}\{\boldsymbol{e}_3\} \in \mathcal{P}^{0,3}$. The function `MK` : $\mathbb{E}^d \to \mathcal{P}^{0,d}$, with d arbitrary positive integer, was given in Script 3.3.15.

Script 4.9.1 (Tetrahedron)

```
DEF tetrahedron = (T:3:(-1/3) ~ EMBED:1 ~ ngon):3 JOIN MK:<0,0,1> ;
```

Hexahedron

The `cube` — bounded by six planes and hence called also a *hexahedron* — is generated, in Script 4.9.2, with center in the origin and inscribed in the unit sphere. Such an inscribed cube has edges of length $a = 2/\sqrt{3}$, as the reader may check by considering the great diagonal of the cube as the hypotenuse of a right-angled triangle whose cathetuses have lengths a and $a\sqrt{2}$, respectively.

Script 4.9.2 (Hexahedron)

```
DEF hexahedron = (T:<1,2,3>:< a/-2,a/-2,a/-2 > ~ CUBOID):< a,a,a >
WHERE a = 2 / sqrt:3 END;
```

Octahedron

The Platonic solid named *octahedron*, which is bounded by 8 triangular faces organized as a double squared pyramid, is simply the three-dimensional *CrossPolytope* defined in Section 4.4.1 as the set $\text{conv}\{\boldsymbol{e}_i, -\boldsymbol{e}_i\}$, $1 \leq i \leq 3$. The implementation given in Script 4.9.3 is consequently simple.

Script 4.9.3 (Octahedron)

```
DEF octahedron = CrossPolytope:3;
```

Dodecahedron

The Platonic solid bounded by 12 pentagons is called *dodecahedron*. As suggested by [Cro97], this one can be constructed by glueing a properly defined "roof" to each face of a central cube, as shown by Figure 4.19a. A first roof is given by the convex set

$$R := \text{conv}\,(B \cup T)$$

i.e. as the convex hull of the embedded 2D *basis* interval $B := [0,1]^2 \times \{0\}$, join the embedded 1D *top* segment $T := \text{conv}\{\boldsymbol{p}_1, \boldsymbol{p}_2\}$, with $\boldsymbol{p}_1 = (1-g, 1, g)$ and $\boldsymbol{p}_2 = (1+g, 1, g)$, respectively, where g is the golden ratio. The other roofs are defined by properly translating and rotating R.

Golden ratio The *golden ratio* is defined as the ratio between the length g of the interval $[0, g] \subset [0, 1]$, and the unit length of the $[0, 1]$ segment. The g value must equate by definition the ratio between the length $1 - g$ of the $[g, 1]$ segment and g itself. In other words, g must satisfy the constraint:

$$g : 1 = (1 - g) : g.$$

It is very easy to see, by getting the positive root of equation $g^2 + g - 1 = 0$, that $g = \frac{1}{2}(\sqrt{5} - 1)$.

Implementation The `dodecahedron` inscribed in the unit sphere is implemented in Script 4.9.4 by using the method described above. For sake of simplicity the primary construction is done using a central cube with edge length equal to 2. The assembled polytope generated by `STRUCT` is then properly scaled by a factor $1/\sqrt{3}$, so reducing the half-diagonal of the interior cube to the unit size of the inscribing sphere. The resulting solid is shown in Figure 4.19b.

The geometric value obtained when evaluating the last expression of Script 4.9.4 is displayed in Figure 4.19a. The `SPLIT` and `explode` operators, to extract the convex 3-cells of the `dodecahedron` complex and to produce a (controlled) "explosion" of cells about the origin, are given in Scripts 10.8.4 and 10.8.6, respectively.

To fully understand the `STRUCT` semantics and the implicit composition of sequences of affine transformations, some preliminary reading of Chapters 6 and 8 may be very useful.

Script 4.9.4 (Dodecahedron)

```
DEF dodecahedron = (S:<1,2,3>:< a,a,a > ~ STRUCT):<
   T:<1,2,3>:< -1,-1,-1 >:(CUBOID:<2,2,2>),
   roofpair, R:<1,3>:(PI/2), R:<1,2>:(PI/2), roofpair ,
   R:<1,2>:(PI/2), R:<2,3>:(PI/2), roofpair >
WHERE
   g = (SQRT:5 - 1)/2,
   top = MKPOL:< <<1-g,1,0-g>,<1+g,1,0-g>>, <<1,2>>, <<1>> >,
   basis = (EMBED:1 ~ CUBOID):<2,2>,
   roof = (T:<1,2,3>:< -1,-1,-1 > ~ JOIN):< basis, top >,
   roofpair = STRUCT:< roof, R:<2,3>:PI, roof >,
   a = 1 / sqrt:3
END;

(STRUCT ~ explode:<1.1,1.1,1.1> ~ SPLIT): dodecahedron
```

In order to obtain a more compact VRML output for the `dodecahedron` value, it might be preferable to insert a pair `MKPOL ~ UKPOL` on the top of the function body. As the reader already knows, this operator insertion flattens the hierarchy of the internal geometry representation, thus reducing the file size of deep hierarchical assemblies, and may eliminate some awkward rendering effects introduced by scaling with negative coefficients, which are formally not allowed by the VRML specification [ISO97].

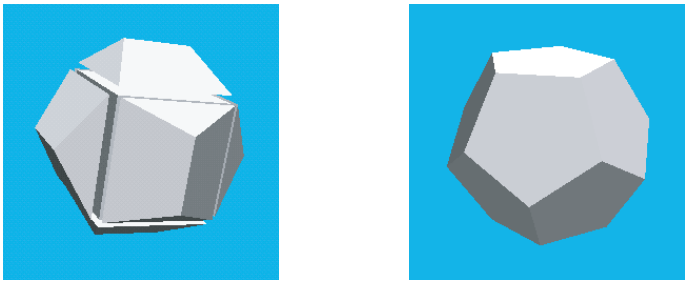

Figure 4.19 (a) Exploded dodecahedron (b) Dodecahedron

Icosahedron

The *icosahedron* is the Platonic solid bounded by 20 equilateral triangles. Its boundary can be seen as composed by threes layers, as shown by Figure 4.20c. The intermediate layer is produced by joining two parallel pentagons mutually rotated by π radians. Since each of their $5 + 5$ sides is joined to one vertex in the opposite pentagon, this layer contains 10 triangles. Also, 5 triangles are contained in the top layer and 5 in the bottom layer.

Implementation According to [Cro97], the solid is generated in Script 4.9.5 as the convex hull of three orthogonal rectangles whose sides are in the golden ratio each other. Such a convex hull is finally scaled to be inscribed in the unit circle. The

scaling coefficient b simply transforms the half-diagonal of such "golden rectangles" to the unit radius of the inscribing sphere.

Script 4.9.5 (Icosahedron)

```
DEF icosahedron = (S:<1,2,3>:<b,b,b> ~ JOIN):< rectx, recty, rectz >
WHERE
   rectx = (EMBED:1 ~ T:<1,2>:<-:g,-1> ~ CUBOID): < 2*g, 2>,
   recty = (R:<1,3>:(PI/2) ~ R:<1,2>:(PI/2)): rectx,
   rectz = (R:<2,3>:(PI/2) ~ R:<1,2>:(PI/2)): rectx,
   g = (SQRT:5 - 1)/2,
   b = 2 / sqrt:(10 - 2*sqrt:5)
END;
```

Example 4.9.1 (Icosahedron constructions)
The icosahedron images shown in Figures 4.9.5a and 4.9.5b are generated by evaluating the `out1` and `out2` symbols given in Script 4.9.6. In particular, the definition of the `planes` assembly is obtained from the `icosahedron` definition by just substituting the operator `STRUCT` to `JOIN` in the function body. Notice that the `COLORS.psm` package must be loaded into memory before evaluating the `out1` symbol.

Figure 4.9.5b of the layered icosahedron structure is generated as value of the `out2` object. In this case (a) the solid is rotated by $\pi/2$ about the $(-1, \frac{1-\sqrt{5}}{2}, 0)$ axis in order to move the diagonal of the rectangle located in the $z = 0$ plane to coincide with the z-axis; (b) the 2-skeleton is extracted; (c) the resulting complex is split into a sequence of convex cells; (d) this sequence is "exploded" along the z direction, and finally (e) a single polyhedral assembly is generated.

Script 4.9.6 (Icosahedron)

```
DEF out1 = STRUCT:<
   icosahedron MATERIAL
      Transparentmaterial:< RGBCOLOR:<0.3, 0.7, 0.9>, 0.6>,
   planes COLOR red >;

DEF out2 = (STRUCT ~ explode:<1,1,1.3> ~ SPLIT ~ @2
   ~ Rot_n:< PI/2, <-1, (1-SQRT:5)/2, 0> >): icosahedron;
```

Notice that the function `Rot_n` for 3D rotation about an axis for the origin can be loaded from `VECTORS.psm` package, whereas the `explode` and `SPLIT` functions came from Scripts 10.8.6 and 10.8.4, respectively.

Example 4.9.2 (Platonic solids in a unit sphere)
The five pictures given in Figure 4.18 are generated from models defined in Script 4.9.7, where each of `out`i objects correspond in order to Platonic solids inscribed in a transparent unit sphere centered in the origin of the $\mathbb{E}^3$ space. The `Sphere` generating function is given in Section 2.2.7. Notice that in order to generate colored or shaded or transparent objects, the `PLaSM` package named `COLORS.psm` must be loaded in memory

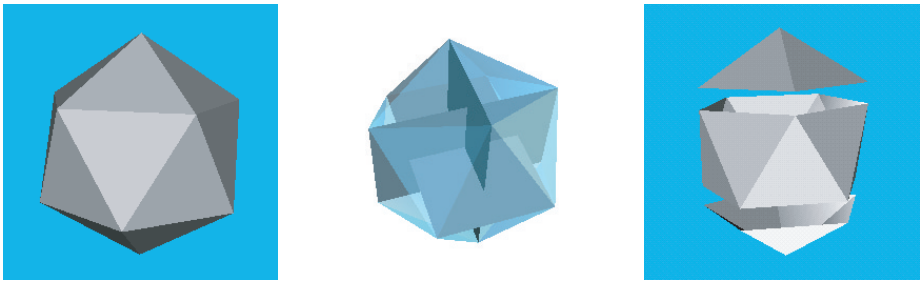

Figure 4.20 (a) Icosahedron (b) Generation of icosahedron as `JOIN` of three orthogonal rectangles with sides in the golden ratio (c) Layered structure of icosahedron

in advance. A quite detailed discussion of color and material properties, both in `PLaSM` and in VRML, may be found in the second part of Chapter 10.

Script 4.9.7 (Platonic solids in a transparent sphere)

```
DEF red = RGBCOLOR:< 1, 0, 0 >;
DEF white = RGBCOLOR:< 1, 1, 1 >;
DEF mymaterial = white Transparentmaterial 0.9;
DEF UnitSphere = Sphere:1:<18,24> CREASE (PI/2) MATERIAL mymaterial;

DEF out1 = STRUCT:< UnitSphere, tetrahedron COLOR red >;
DEF out2 = STRUCT:< UnitSphere, hexahedron COLOR red >;
DEF out3 = STRUCT:< UnitSphere, octahedron COLOR red >;
DEF out4 = STRUCT:< UnitSphere, dodecahedron COLOR red >;
DEF out5 = STRUCT:< UnitSphere, icosahedron COLOR red >;
```

4.10 References

The reader interested to some in-depth study of polyhedral geometry topics we introduced in this chapter, is referred to the following list of papers or books:

[Bal61, Bie95, Bie98, Bie94a, Bie94b, BN88, Bro83, Bro88, Bur74, CHJ90, Che65, Chv83, Cro97, Dan63, DP77a, DP77b, Dye83, Ede87, Fer95a, Fer95a, Gib77, GT87, Grü67, GS85, GW89, Hop83, BF89, Lef49, Lue84, LW69, MN68, MR77, MR80, Mun84, Mur83, Nak91, Nef78, PS85, Req77, RS72, RS89, Sch93, Sch86, Sob89, Sug93, Von81, Wen71, Yao90, Zie95]