HAL
open science

Simploidal Sets: a data structure for handling simploidal Bézier spaces

Samuel Peltier, Pascal Lienhardt

To cite this version:

Samuel Peltier, Pascal Lienhardt. Simploidal Sets: a data structure for handling simploidal Bézier spaces. Computer Aided Geometric Design, 2018, 62, pp.44 - 62. 10.1016/j.cagd.2018.03.010 . hal-01274880v3

HAL Id: hal-01274880

https://hal.science/hal-01274880v3

Submitted on 31 Aug 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Simploidal Sets: a data structure for handling simploidal Bézier spaces

Samuel Peltier [1], Pascal Lienhardt [1]

[1] CNRS, Univ. Poitiers, XLIM, UMR 7252, F-86000 Poitiers, France

Abstract

Simplicial sets and cubical sets are combinatorial structures which have been studied for a long time in Algebraic Topology. These structures describe sets of regular cells, respectively simplices and cubes, and any kind of assembly of cells can be represented. They are used for many applications in Computational Topology, Geometric Modeling, CAD/CAM, Computer Graphics, etc. For instance, simplicial and cubical sets are "naturally" associated with simplicial and cubical Bézier spaces.

In this paper, a new combinatorial structure, namely *simploidal sets* is defined for representing and handling Bézier simploids. Simploidal sets describe sets of simploids, which are also regular cells corresponding to cartesian product of simplices. Simplices and cubes are then particular simploids.

In order to associate shapes with structures, structural relations between simploidal sets and simploidal Bézier spaces are also stated. In fact, simploidal sets generalize and homogenize simplicial and cubical sets.

Construction operations are also defined, extending all those of simplicial and cubical sets: cone, cartesian product, degeneracy, and identification. In their basic version, the first three operations allow to create any simploid, and the last one, to create any assembly of simploids. It is then possible to simultaneously handle through a single formalism any assembly of simplices, cubes, and other simploids, with a very low additional cost, regarding space (data structure), time (construction or computation operations) or software development.

1 Introduction

The design of complex free form objects is an important issue in Geometric Modeling. We are interested in representation models and basic operations for constructing sketches of complex subdivided objects, from which the final geometric object can be obtained by applying techniques of subdivision, deformation and others (e.g. [10, 28]).

In this context, the complexity lies in the types of cells and the way they are glued together. Many works deal with the definition of geometric models of complex cells, mainly for low dimensions (e.g. [27]). Among models defined for higher dimensions, simploids are complex cells which structures correspond to cartesian products of simplices [4]. Different models have been defined for representing unstructured meshes (for instance taking into account the refinement process [26]). Well known combinatorial structures as simplicial and cubical sets [1, 20, 25, 3] can be used for representing sketches of subdivided objects. More recently, semi-simploidal sets [1] [23] have been defined for representing assemblies of Bézier simploids [4]. As a cube is the cartesian product of edges (i.e. 1-dimensional simplices), a simploid is the cartesian product of (any) simplices. Simplices and cubes are then simply particular simploids, so all these cells can be handled together using semi-simploidal sets. For example, Figure 1 illustrates an assembly of 3 simploids: a cube and a tetraedron are linked through a prism. This can be interesting for several applications, e.g. related to meshes. Structural relations between semi-simploidal sets and Bézier simploids make it possible to efficiently implement simploidal Bézier spaces using semi-simploidal sets.

When defining a geometric model, it is also important to provide basic operations from which higher-level operations can be designed. Cone and identification (i.e. the merging of cells) are the basic operations associated with simplicial structures, as extrusion and identification are the basic operations associated with cubical structures[2]. Since simploids generalize simplices and cubes, the three operations: cone, cartesian product and identification have to be their associated basic operations. Unfortunately, the cone operation is not defined for semi-simploidal sets.

In order to solve this problem, a new combinatorial structure, namely *simploidal sets* is defined (cf. Definition 1 page 10), for representing any assembly of simploids, and any simploidal set can be constructed by applying cone, cartesian product and/or identification. Moreover, another operation, degeneracy, is also defined, due to the definition of simploidal sets.

In order to associate simploidal Bézier spaces with simploidal sets, the structural relations between them has to be stated: we first establish this relation for simplicial sets and simplicial Bézier spaces, and then it is generalized for simploidal sets. As a consequence, simploidal Bézier spaces can be efficiently implemented using simploidal sets (cf. Section 4.3): control points can be stored without any redundancy, avoiding problems related to space complexity and possible inconsistencies.

Basic construction operations and topological properties computations can be efficiently implemented, since all the necessary structural information as adjacency and incidence relations between simploids is explicitly represented.

Regarding implementation issues, since the definition of simploidal sets is very close to that of simplicial and cubical sets, the additional cost is very low, with respect to space (cost of data structure), time (cost of operations) or software development (cf. Section 4).

So, the main contributions of this paper are:

- the definition of simploidal sets;
- Property 1, which establishes the precise structural relations between simplicial sets and the control points of triangular Bézier spaces; and Property 2, which extends Property 1 for simploidal sets;
- a data structure implementing simploidal Bézier spaces;
- definitions and algorithms for all basic construction operations: cone, cartesian product, identification and degeneracy (cf. Section 4.4), generalizing the basic construction operations defined for simplicial and cubical sets.

[1] Note that in [23], the term "*simploidal*" meant "*semi-simploidal*". This terminology is modified here in order to be consistent with those related to simplicial and cubical structures.

[2] i.e. any simplicial object can be constructed using cone and identification, as any cubical object can be constructed using extrusion and identification.

- a comparison with *supercomplexes*, a "similar" combinatorial structure, proposed by `Gugenheim` in 1957 for the study of topological properties [13] (cf. Theorem 4).

Note that cone and identification operations are defined on semi-simplicial sets, and that simplicial sets extend them so that the cartesian product operation can be defined. Conversely, cartesian product and identification are defined on semi-simploidal sets, and simploidal sets extend them so that the cone operation can be defined. It has to be noticed that the way for deducing simploidal sets from semi-simploidal sets is similar to the way for deducing simplicial sets from semi-simplicial sets, and lies in the fact that simploids (or simplices) can be degenerate ones.

As a consequence, it is possible to represent more general simploids with simploidal sets than with semi-simploidal sets, and this versatility can be of interest for geometric modeling purposes.

It is shown that the simploidal framework is general, coherent and homogeneous, since it contains also the simplicial and cubical sets frameworks as particular cases.

At last, these structures are designed for representing *any* assembly of cells (e.g. unstructured meshes), so they can be quite space consuming: this is the reason why they are generally employed for the design of sketches of complex geometric objects. But note that the representation can be optimized by applying usual optimization or compression techniques [18].

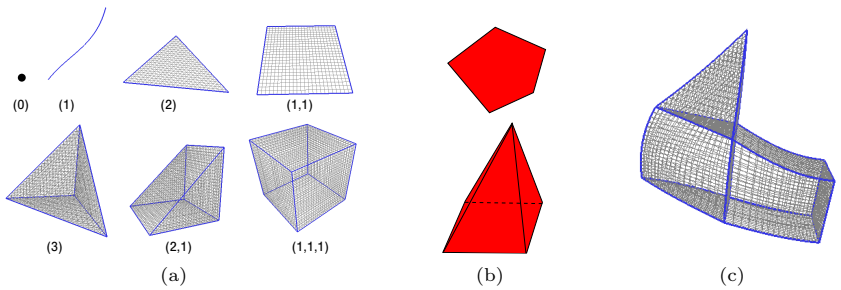


Figure 1: (a) Simploids and their types. (b) Counterexamples of simploids: a pentagon and a pyramid (i.e. a cone on a square face): the main cells are not the cartesian product of simplices. (c) A geometric representation of a simploidal set made of 3 3−dimensional simploids: a tetraedron (type (3)), a cube (type $(1,1,1)$) and a prism (type $(2,1)$ or $(1,2)$).

The outline of the paper is the following. For clarity purpose, all necessary notions are presented in a progressive way: notions related to structures without degeneracy operators, namely semi-simplicial, semi-cubical and semi-simploidal sets, are recalled in Section 2. These three structures handle abstract cells (i.e. simplices, cubes and simploids) equipped with *face operators* associating with an i-dimensional cell the $(i-1)$-cells of its boundary. Simploids generalize simplices and cubes: as a simplex or a cube is characterized by its dimension, a simploid is characterized by its *type* $(a_1, \cdots, a_n)$ (cf. Figure 1(a)): intuitively, a simploid can be seen as the cartesian product of simplices $\sigma_1 \times \cdots \times \sigma_n$, and a_i is the dimension of σ_i. So, n-simplices and n-cubes are particular simploids of types (n) and $(a_1 = 1, \cdots, a_n = 1)$. Note that these structures allow the representation of any assembly of simplices, cubes, or simploids. Regarding basic construction operations, cartesian product and identification can directly be defined for semi-simploidal sets, but the cone operation can not (cf. Figure 1(b)).

Section 3 focusses on simplicial and cubical sets, which extend semi-simplicial and semi-cubical sets by adding *degeneracy operators*. The precise relations with Bézier spaces are established.

Simploidal sets are defined in Section 4, by adding degeneracy operators. The relations with simploidal Bézier spaces are established, and a data structure avoiding any redundancy for Bézier embedding is proposed. All basic simplicial and cubical construction operations are extended within the simploidal framework: cone, cartesian product, identification, and degeneracy, and it is shown that the simploidal cone operation can be defined in a simple way thanks to degeneracy operators. So, simploidal sets generalize and homogenize simplicial and cubical sets.

2 Structures Without Degeneracy

N-dimensional simplicial "objects" or "triangulations" have been studied for a long time in Algebraic Topology. *Combinatorial structures*, as abstract simplicial complexes or simplicial sets, describe the adjacency and incidence relations of abstract simplices, and a *geometric realization* can be associated with each structure. For instance, the geometric realization of an abstract simplicial complex is a simplicial complex, whereas the geometric realization of a simplicial set is a CW-complex [1, 20, 21, 19]. Such structures are used for different purposes, in Computational Topology (e.g. computation of topological properties) [8, 24, 7], Geometric Modeling, CAD/CAM and Computer Graphics (e.g. representation of subdivided geometric objects, meshes) [22, 12], etc. In particular, structural relations between semi-simplical sets and triangular Bézier spaces [11] have been established [16], leading to the definition of efficient data structures for handling triangular Bézier spaces.

N-dimensional cubical "objects" have been studied [25, 3] for similar purposes in Topology, Geometric Modeling, etc. So, semi-simplicial, semi-cubical and semi-simploidal sets [23] describe sets of (products of) simplices, such that any assembly of cells can be represented.

This section is mainly based upon the papers [15, 23], in which algorithms implementing basic operations for handling semi-simplicial sets and semi-simploidal sets are also described.

2.1 Semi-Simplicial Sets

Combinatorial structure. Most well-known triangulated objects are simplicial complexes [1], which are sets of geometric simplices satisfying some properties, a i-dimensional simplex (or i-simplex) being the convex hull of $i+1$ linearly independent points. The corresponding combinatorial structure is also well-known: an abstract simplicial complex is a set of abstract simplices, and an abstract i-simplex is a set of $i+1$ (abstract) vertices.

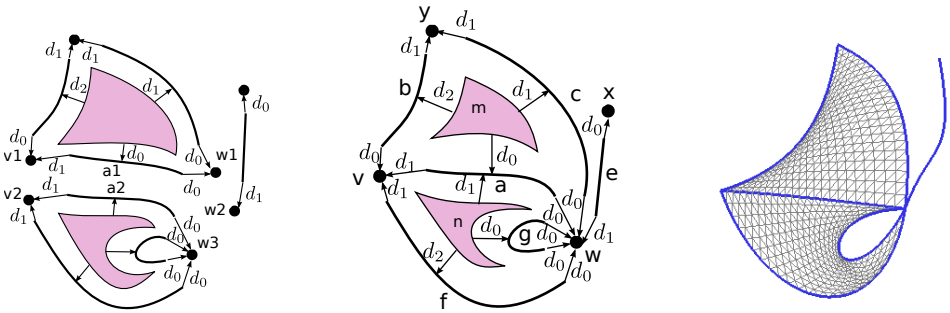
A n-dimensional *semi-simplicial set* (or Δ-complex) [9, 14] $S = (K, (d_j))$ is a family of sets $K = (K^i)_{i \in [0..n]}$, such that each K^i is equipped with face operators: $d_j : K^i \to K^{i-1}$ for $1 \leq i \leq n$ and $0 \leq j \leq i$; the face operators satisfy "commutation

properties": for $\sigma \in K^i$, and $0 \leq l < j \leq i$, $\sigma d_j d_l = \sigma d_l d_{j-1}$, where $\sigma d_j d_l$ denotes $d_l \circ d_j(\sigma)$.

K^i is the set of i-dimensional (abstract) simplices (or i-simplices); the $i+1$ face operators defined for a i-simplex σ associate with σ the (at most) $i+1$ $(i-1)$-simplices of its boundary. Intuitively, if σ is assimilated with a sequence $(v_0, \cdots, v_j, \cdots, v_i)$ of $i+1$ vertices, σd_j corresponds to the sequence $(v_0, \cdots, v_{j-1}, v_{j+1}, \cdots, v_i)$. More generally, the "commutation properties" involve that at most $\binom{i+1}{j+1}$ $j-$simplices belong to the boundary of a $i-$simplex.

Semi-simplicial sets generalize abstract simplicial complexes, since the definition of a simplex boundary through face operators permits to handle i-simplices which are incident to less than $(i+1)$ vertices (cf. Figure 2); moreover, simplices are not necessarily linearly embedded (cf. below).

Let σ be a i-simplex; σd_j is its jth-face. Simplex τ is a *face* of σ if it can be obtained from σ by applying a non-empty sequence of face operators. The *boundary* of σ is the subset of S restricted to the faces of σ. The *star* of σ is the set of simplices of which σ is a face. σ is a *main* simplex if its star is empty. The boundary of a *complete* i-simplex contains exactly $(i+1)$ 0-simplices. For example in Figure 2(b): y and b are faces of m; the boundary of n contains the simplices $\{a, g, f, v, w\}$; the star of w is the set $\{f, g, a, c, e, n, m\}$; m,n and e are the main simplices; all simplices but n and g are complete. This terminology can be extended in a straightforward way for all structures described in this paper.



(a) (b) (c)

Figure 2: (a) A semi-simplicial set. Semi-simplicial set (b) is obtained by identifying vertices $v1$ and $v2$, vertices $w1$, $w2$ and $w3$, and edges $a1$ and $a2$. (c) A geometric representation of semi-simplicial set (b). The commutation properties satisfied by face operators guarantee the consistency of the data structure. For example, for any triangle σ: $\sigma d_2 d_1 = \sigma d_1 d_1$, $\sigma d_2 d_0 = \sigma d_0 d_1$, and $\sigma d_1 d_0 = \sigma d_0 d_0$, so a triangle has at most 3 vertices in its boundary.

Triangular Bézier spaces and relation with semi-simplicial sets. As abstract simplicial complexes can be associated with simplicial complexes, semi-simplicial sets can be associated with triangular Bézier spaces. A *triangular Bézier space* is a set of *Bézier simplices* satisfying some conditions (see [11]).

More precisely, the set Γ_d^i of *i-dimensional multi-indices of degree d* is defined by $\Gamma_d^i = \{\alpha = (\alpha_0, \cdots, \alpha_i) \in \mathbb{N}^{i+1} \mid |\alpha| = \alpha_0 + \cdots + \alpha_i = d\}$. In the following, multi-index $(\alpha_0, \cdots, \alpha_i)$ is denoted $\alpha_0 \cdots \alpha_i$. For instance, $\Gamma_3^1 = \{03, 12, 21, 30\}$, and $\Gamma_3^2 = \{003, 012, 021, 030, 102, 111, 120, 201, 210, 300\}$. The cardinality of Γ_d^i, denoted $\mid \Gamma_d^i \mid$, is equal to $\binom{i+d}{i}$.

Multivariate Bernstein polynomials of degree d are defined by $B_\alpha^d(u) = \binom{d}{\alpha} u_0^{\alpha_0} \cdots u_i^{\alpha_i}$, with $\alpha \in \Gamma_d^i$, and $\binom{d}{\alpha} = \frac{d!}{\alpha_0! \cdots \alpha_i!}$ are *multinomial coefficients*, and $\{u_j\}$ are the barycentric coordinates of u, a point of the standard i-simplex, i.e. $\{u_j\}$ satisfies: $\forall j, 0 \leq j \leq i, 0 \leq u_j \leq 1$ and $\sum_{j=0}^{i} u_j = 1$.

A *Bézier i-simplex of degree d* is defined by $P(u) = \sum_{\alpha \in \Gamma_d^i} P_\alpha B_\alpha^d(u)$, where $\{P_\alpha, \alpha \in \Gamma_d^i\}$ is its set of *control points* (cf. Figure 3), which can be stored in a *simplicial array*, according to the lexicographical order of the multi-indices [6]. The index in a simplicial array of any control point P_α can be retrieved using algorithm 1. The idea is the following: let $\alpha = \alpha_0 \ldots \alpha_i$ be the multi-index of a control point. If $i = 1$ then the point is stored at index α_0. If $i > 1$, then it is necessary to store first α_0 sets of control points corresponding to $(i-1)$-dimensional Bézier simplices of degrees decreasing from d to $d - \alpha_0 + 1$, and then to store the point in the same way as if it would belong to the set of control points of a $(i-1)$-dimensional Bézier simplex of degree $d - \alpha_0$.

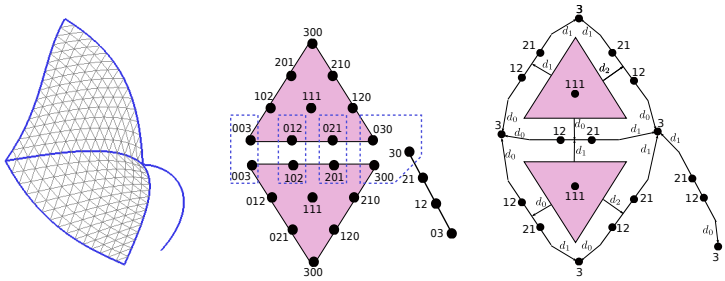
Algorithm 1: multiIndex2Index

Input : $\alpha = \alpha_0 \cdots \alpha_i$: MultiIndex Γ_d^i
Output: *pos* : Integer

if $i == 1$ **then**
  $pos = \alpha_0$;
else
  offset $= \sum_{j=d-\alpha_0+1}^{d} \mid \Gamma_j^{i-1} \mid$;
  pos = offset + `multiIndex2Index(`$\alpha_1 \cdots \alpha_i$`)`;
return pos;

A triangular Bézier space can easily be implemented by a set of simplicial arrays, such that each simplicial array contains the control points of a *main* Bézier simplex. Several construction operations as cone and identification (cf. operations below) can be easily implemented for handling this data structure. But all simplices are not explicitly represented, nor adjacency and incidence relations. So, such a structure is not well suited for implementing several operations, for instance the computation of topological properties as homology. Moreover, the control points which are "shared" by adjacent main simplices are duplicated. For example in figure 3(b), the control points surrounded by dashed lines have to be the same.

More efficient data structures can be defined, based upon the following relation between triangular Bézier spaces and semi-simplicial sets. It is well known that a Bézier $i-$simplex β contains its boundary, i.e. the boundary of β is a set of lower dimensional



(a) (b) (c)

Figure 3: (a) An example of Bézier space of degree 3. (b) Main storage: only the control points of the main simplices are stored. (c) Distributed storage: only the proper control points are stored.

Bézier simplices defined by subsets of control points of β. More precisely, all control points P_α of β such that the jth component of α is 0 define a $(i-1)$ Bézier simplex. For instance in Figure 3(b), the points numbered $003, 102, 201, 300$ of each triangle define a Bézier curve of degree 3. So, the following constraint has to be satisfied in order to represent a triangular Bézier space: let σ be a i-simplex, $1 \leq i$, and $\mu = \sigma d_j$, for $0 \leq j \leq i$; let $P = P_{\alpha_0 \cdots \alpha_j \cdots \alpha_i}$ be a control point of σ such that $\alpha_j = 0$; then point $P_{\alpha_0 \cdots \widehat{\alpha_j} \cdots \alpha_i}$ of μ is equal to P, where $\widehat{\alpha_j}$ means α_j is removed.

Consequently, a data structure can be defined, based upon semi-simplicial sets, such that any simplex is associated with its *proper control points*, i.e. control points P_α such that no component of α is null (cf. Figure 3(c)). The set of proper control points of an $i-$simplex of degree d has the structure of a $i-$simplex of degree $d-(i+1)$, and thus contains $\binom{d-1}{i}$ control points. This set can be stored in a simplicial array as before, by using the lexicographical order of multi-indices.

The set of all control points of a simplex can be retrieved from its proper control points and from the control points of the simplices of its boundary, by taking into account the face operators indices. For instance, let σ be a triangle of Figure 3(c): the proper control points numbered $21, 12$ of the edge σd_0 (resp. σd_1 and σd_2) correspond to the control points numbered $021, 012$ (resp. $201, 102$ and $210, 120$) of σ, as shown in Figure 3(b).

Data structure. The following data structure is a straightforward implementation of semi-simplicial sets associated with triangular Bézier spaces of degree `degree`. The notation `*Type` denotes an access to a `Type` data ; `[x..y] of Type` and `[] of Type` denote an array of `Type`: in the first case, the indices are specified; `List<Type>` denotes a list of `Type` data.

```
type Simplex is record :
   dim :  integer
   faceOps :  [0..dim] of *Simplex
   ctrlPts :  [0..C_dim^(set.degree-1) - 1] of CtrPt
   set :  *SemiSimplicialSet
end record
```

```
type SemiSimplicialSet is record :
   dim :  integer
   degree :  integer
   allSimplices :  [0..dim] of List<*Simplex>
end record
```

In this `SemiSimplicialSet` data structure, it is assumed that all Bézier simplices of a given Bézier space have the same degree, `allSimplices` gives access to the lists of i-simplices, $0 \leq i \leq$ `dim`, where `dim` is the dimension of the set.

Regarding the `Simplex` structure, `dim` is its dimension, `faceOps` gives access to the $(dim-1)$-simplices of its boundary, `set` gives access to its semi-simplicial set and `ctrlPoints` stores its proper control points, ordered by lexicographical order. Algorithm 3 can be used to deduce which proper multi-index α is stored at index pos by computing firstly $\alpha' = index2MultiIndex(i, d-(i+1), pos)$, and then α is obtained by adding 1 to each α'_j, $0 \leq j \leq i$.

Algorithm 3: index2MultiIndex

Input : i, d, pos : integer `/* i is the dimension, d is the degree, and pos the index */`
Output: α: MultiIndex of Γ_d^i

Algorithm 2: index2MIrec

Input : i, d, pos, j : integer
Output: $\alpha = \alpha_0, \cdots, \alpha_i$: MultiIndex of Γ_d^i
if $i == 1$ **then**
  $\alpha_0 = pos$;
  $\alpha_1 = d - pos$;
else
  offset $= \binom{i+d-j-1}{i-1}$;
  if $pos <= offset - 1$ **then**
    $\alpha_0 = j$;
    $\alpha_1 \cdots \alpha_i$=`index2MIrec(`*i-1,d-j,pos,0*`)`;
  else
    α=`index2MIrec(`*i,d,pos-offset,j+1*`)`;
return α;

return `index2MIrec(`*i,d,pos,0*`)`;

Data structure variants. Obviously, this structure can be adapted for handling Bézier simplices with different degrees, using for instance well-known results about degree elevation for controling the continuity [11]. A list of accesses can be added for

any i-simplex, in order to access the $(i+1)$-simplices of its star.

Basic Operations. Any semi-simplicial set S can be constructed by applying two basic operations, namely cone and identification [15].

The *cone* of S consists in adding a 0-simplex v to S, and in creating a new $(i+1)$-simplex μ incident to σ and v for any i-simplex σ of S (cf. Figure 4). The proper control points of μ can be computed for instance by linear interpolation between the proper control points of σ and the control point associated with v. Note that a complete i-simplex (with its boundary) is a cone upon a complete $(i-1)$-simplex (with its boundary).

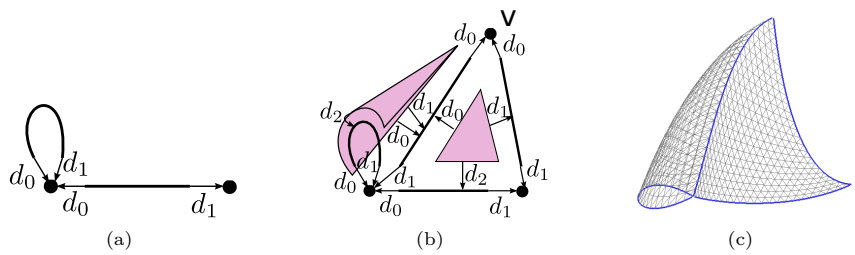


Figure 4: The semi-simplicial set b) can be obtained by a cone operation applied to the semi-simplicial set of a), v is the new vertex. c) is a geometric representation of semi-simplicial set b).

The *identification operation* can be applied to two distinct i-simplices σ and τ, if they share the same boundary. The operation consists in replacing σ and τ by a new i-simplex μ such that its boundary is equal to the boundary of σ (and τ), and its star is the union of the stars of σ and τ. More precisely, if $\nu d_j = \sigma$ or $\nu d_j = \tau$ before operation, then $\nu d_j = \mu$ after operation. For instance, the semi-simplicial set of Figure 2(b) can be obtained from the one of Figure 2(a) by identifying 0-simplices $v1$ and $v2$, 0-simplices $w1$, $w2$ and $w3$, and 1-simplices $a1$ and $a2$. As for the cone operation, the proper control points of new simplices can be computed for instance by linear interpolation of the proper control points of the identified simplices.

Higher-level construction operations have been defined: for instance, any two i-simplices can be identified, by first identifying their boundaries by increasing dimensions, then by identifying the i-simplices themselves. See also the identification of subsets of simplices, split operation, flip operation, etc. [2]. Operations for computing topological properties, as homology, have also been defined [20].

2.2 Semi-Cubical Sets

A *n-dimensional semi-cubical set* $S = (K, (d^i_j))$ is a family of sets $K = (K^p)_{p\in[0..n]}$ together with face operators: $d^i_j : K^p \to K^{p-1}$ satisfying commutation properties: for $1 \le p \le n$, $1 \le i \le p$ and $j \in \{0,1\}$, $d^i_j d^k_l = d^k_l d^{i-1}_j$ if $k < i$.

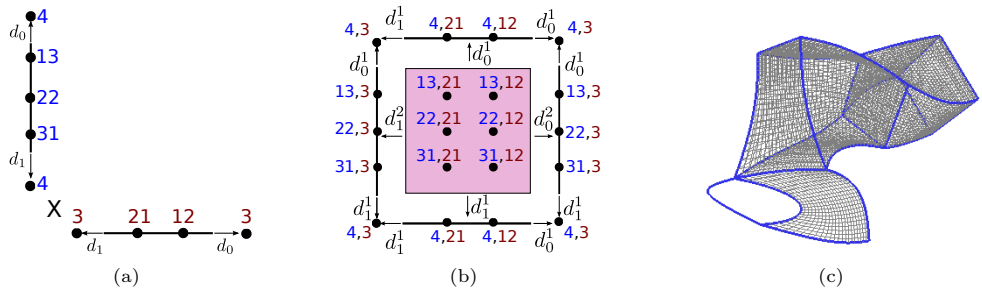


Figure 5: The 2-cube in (b) results from the product of the vertical edge by the horizontal edge in (a), so face operators d^1_j (resp. d^2_j) which acts on the 2−cube "are oriented" vertically (resp. horizontally). A cubical Bézier space corresponding to a cubical set containing two 3-cubes, one 2−cube and one 1−cube as main cells.

An element σ of K^p is a *p-cube*. Intuitively, a p-cube is the cartesian product of p 1-simplices $\sigma_1 \times \ldots \times \sigma_i \times \ldots \times \sigma_p$, and face operator d^i_j associates σ with the $(p-1)$-cube corresponding to $\sigma_1 \times \ldots \times \sigma_i d_j \times \ldots \times \sigma_p$ (cf. Figure 5). Note that the cartesian product of a p-cube by a vertex is a p-cube.

A *Bézier patch* is the product of Bézier curves, a *cubical Bézier space* is a set of Bézier patches satisfying some conditions, and the correspondence described above between semi-simplicial sets and triangular Bézier spaces is retrieved here for semi-cubical sets and cubical Bézier spaces (cf. Figure 5). Note that the degrees of the generating edges can be different, and that the control points of a i-dimensional patch are denoted by i-tuples of 1-dimensional multi-indices. Note that a generating edge of degree d contains $d-1$ proper control points, so a i−cube obtained from i edges of degrees $d_1 \cdots d_i$ contains $\prod_{j=1}^{i}(d_j - 1)$ proper control points.

Data structure. The data structure presented for semi-simplicial sets can directly been adapted for semi-cubical sets :

```
type Cube is record :
   dim :  integer
   degrees :  [1..dim] of integer
   faceOps :  [1..dim] of ([0..1] of *Cube)
   ctrlPts :  [0..∏_{j=1}^{dim}(degrees[j] − 1) − 1] of CtrPt
   set :  *SemiCubicalSet
end record
```

```
type SemiCubicialSet is record :
   dim :  integer
   allCubes :  [0..dim] of List<*Cube>
end record
```

Any semi-cubical set can be constructed by applying two basic operations: extrusion and identification. Extrusion is the particular case of the cartesian product by a 1-simplex (and its boundary); note that any complete p-cube can be constructed by the extrusion of a complete $(p-1)$-cube. Two cubes can be identified if their boundaries are equal, and if their control points have the same structure. Obvioulsy, higher-level operations have been defined: identification and cartesian product of semi-cubical subsets, split, homology computation, etc.

2.3 Semi-Simploidal Sets

Combinatorial Structure. A semi-simploidal set is a set of simploids on which act face operators (cf. Figure 6). In fact, semi-simploidal sets generalize semi-simplicial and semi-cubical sets, according to the intuitive fact that a simploid is the cartesian product of simplices of any dimensions. So, *a simploid σ is characterized by its type* $(a_1, \ldots, a_n)$. Intuitively, a_i is the dimension of its ith generating simplex, and $a_i > 0$ for any $1 \leq i \leq n$ since the cartesian product by a vertex is the identity. The face operator d^i_j associates with σ the same product of simplices, except that the ith component is replaced by its jth face. So, the type of σd^i_j is $(a_1, \cdots, a_i - 1, \cdots, a_n)$ if this j^{th} face is not a vertex, $(a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$ otherwise. For example in Figure 6, let $\sigma = \mu \times \tau$ be the simploid of type $(2, 1)$, μ be the $2-$simplex and τ the $1-$simplex. Simploids σd^1_j (resp. σd^2_j) are faces of type $(1, 1)$ (resp. (2)) as it corresponds to the simploids $\mu d_j \times \tau$ (resp. $\mu \times \tau d_j$).

The formal definition of semi-simploidal sets is not provided here (cf. [23]), it can be retrieved by keeping only (I)-1 and (II) from definition 1 of simploidal sets (cf. Section 4). Note that the definitions of semi-simplicial sets and semi-cubical sets can also be retrieved from definition 1 by keeping only (I)-1 and (II), and by considering i-simplices as simploids of type (i), and i-cubes as simploids of type $(a_1 = 1, \ldots, a_i = 1)$.

Bézier Embedding. A simploidal Bézier space is a set S of Bézier simploids such that the intersection of any two simploids σ and τ of S is either empty or a simploid of S, corresponding to a common face of σ and τ. A Bézier simploid of type $(a_1, \ldots, a_n)$ and degree $(d_1, \ldots, d_n)$ is defined by:
$P(u^1, \ldots, u^n) = \sum_{\alpha^1 \in \Gamma^{a_1}_{d_1}} \cdots \sum_{\alpha^n \in \Gamma^{a_n}_{d_n}} P_{(\alpha^1, \ldots, \alpha^n)} B^{d_1}_{\alpha^1}(u^1) \times \ldots \times B^{d_n}_{\alpha^n}(u^n)$ where any u^i is a point of the standard a_i-simplex and $\{P_{(\alpha^1, \cdots, \alpha^n)}\}$ is its set of control points.

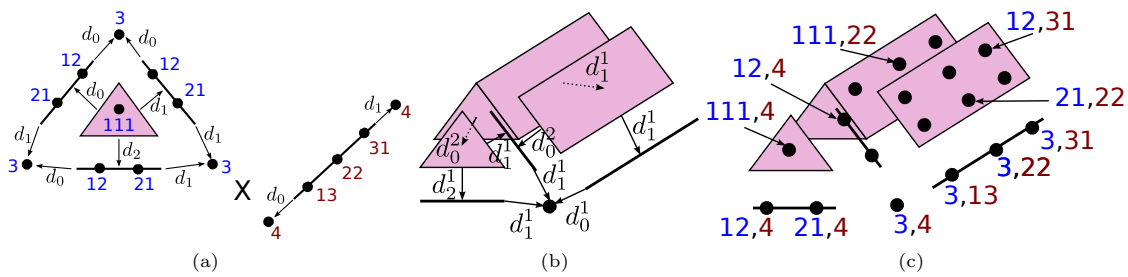


Figure 6: a) and b): correspondence between the product of two simplices (and their boundaries) and a simploid of type $(2, 1)$. For clarity purpose, only a part of its boundary is displayed and some of its control points are displayed in c).

As before, a correspondence exists between the semi-simploidal set structure and the structure of n-tuples of multi-indices: cf. Figure 6. This correspondence makes it possible to associate its proper control points with any simploid, and to retrieve all control points (and their tuples of multi-indices) of a simploid from its proper control points and the proper control points of the simploids of its boundary, using the numbering of the face operators.

Algorithm 4: index2TupleMultiIndex

/* Returns the posth tuple of multiIndex of $\Gamma^{i_1}_{d_1} \times \cdots \times \Gamma^{i_k}_{d_k}$ according to the lexicographical order. */

Input : $type = [(i_1, d_1), \cdots, (i_k, d_k)]$: List$< (Integer, Integer) >$; pos : integer
Output: $\alpha = (\alpha^1, \cdots, \alpha^k)$: $k-$tuple of multi-index

if *(len(type)==1)* **then**
  α = (index2MultiIndex(i_1, d_1,*pos*));
else
  card $= \prod_{j=2}^{k} \binom{i_j + d_j}{i_j}$;
  α = (index2MultiIndex(i_1, d_1,*pos/card*)) + index2TupleMultiIndex($[(i_2, d_2), \cdots, (i_k, d_k)]$, *pos%card*);
return α;

As before, control points can be stored into a *simploidal array*, using the lexicographical order of the tuples of multi-indices, which induces a correspondence between the index of a point in the array and its associated tuple of multi-indices [6] (cf. Algorithm 4). A data structure for implementing semi-simploidal sets embedded as simploidal Bézier spaces can be deduced from the data structure presented in Section 4.3.

Basic Operations. Any semi-simploidal set can be constructed using the cartesian product and the identification operations. The cartesian product definition can be deduced from the fact that the cartesian product of two simploids of types $(a_1, \cdots, a_n)$ and $(b_1, \cdots, b_m)$ is a simploid of type $(a_1, \cdots, a_n, b_1, \cdots, b_m)$. The control points of the resulting simploid can be computed for instance by "adding" the control points (i.e. their coordinates) of the initial simploids (Minkowski sum). For instance on Figure 6(c), the control point $P_{111,22}$ can be defined as the "sum" of the control points P_{111} and P_{22} of Figure 6(a). As before, two simploids can be identified if they share the same boundary, and if their control points have the same structure. As before, higher level operations can be easily defined.

3 Simplicial sets

This section is mainly based upon [20, 16]. Our main contribution here is Property 1, which establishes relations between degeneracy operators and control points.

Combinatorial Structure. The cartesian product operation is not directly defined for semi-simplicial sets, but it is defined in a direct way on simplicial sets, thanks to the notion of *degenerate* simplex: cf. Figure 7.

A *simplicial set* $S = (K, (d_j), (s_j))$ is a family $K = (K^i)_{i \in \mathbb{N}}$ of simplices together with two families of maps, such that $(K, (d_j))$ is a semi-simplicial set, and (s_j) are *degeneracy operators*: $\forall i \geq 0$, $\forall j$, $0 \leq j \leq i$, $s_j : K^i \to K^{i+1}$; the degeneracy operators satisfy the commutation properties:

- $s_j s_l = s_l s_{j+1}$ if $l \leq j$;
- $s_j d_l = d_l s_{j-1}$ if $l < j$, $s_j d_l = d_{l-1} s_j$ if $l > j+1$, and $s_j d_j = s_j d_{j+1} = \mathrm{Id}$.

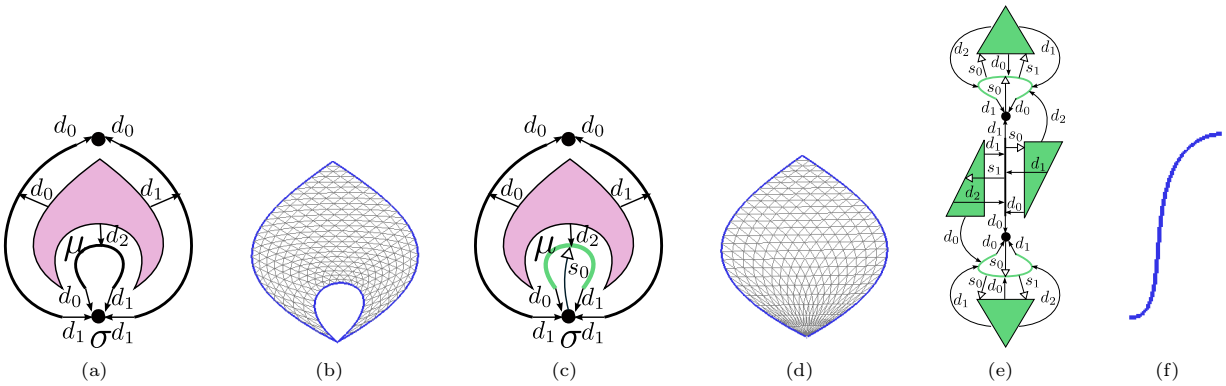


Figure 7: a) a semi-simplicial set and (b) its associated Bézier triangular space. (c) a simplicial set: the green edge μ is degenerate into the bottom vertex σ. Otherwise said, $\mu = \sigma s_0$. (d) the corresponding triangular Bézier space: the degenerate edge is reduced to a point, corresponding to its incident vertex. Note that edge μ is the only degenerate simplex which is a face of a non degenerate one. (e) a simplicial set representing an edge (and its boundary) and all its degenerate simplices up to dimension 2 (in green) and f) its corresponding Bézier triangular space.

An $(i+1)$-simplex μ is *degenerate* if $\mu = \sigma s_j$ for some σ and some j: intuitively, if σ is assimilated with a sequence of $i+1$ vertices $(v_0, \cdots, v_j, \cdots, v_i)$, μ corresponds to $(v_0, \cdots, v_j, v_j, \cdots, v_i)$. Note that the geometric realization of μ is the one of σ.

From a theoretical point of view, all degererate simplices deduced from a simplex by a sequence of degeneracy operators exist in a simplicial set: thus any simplicial set contains an infinite number of degenerate simplices. For instance, the existence of all these degenerate simplices makes it possible to get a very simple definition of the cartesian product (cf. below). But from a practical point of view, only the explicit representation of non degenerate simplices and their faces (degenerate or not) is required (cf. Figures 7(c) and 7(d)); the other degenerate simplices, can be implicitly handled. Indeed, any degenerate simplex can be obtained by applying a sequence of degeneracy operators to a non degenerate simplex. So, the degenerate simplices which appear only in the boundary of degenerate simplices can be implicitly handled, through the corresponding non degenerate simplex and the sequence of indices of the corresponding degeneracy operators. Compared with the implementation of semi-simplicial sets, the additional cost of the implementation of simplicial set is thus low, and related to the object which has to be represented (for instance, if there are no degenerate simplices in the boundary of any non degenerate simplex, the additional cost is quite null). Obviously, this implicit representation of degenerate simplices has to be taken into account when implementing the construction operations.

Bézier embedding. Let σ be any i-simplex of a simplicial set: a Bézier simplex is associated with σ. The relation between the proper control points of σ and those of σd_j, for any j, is the relation described for semi-simplicial sets. If μ is degenerate (i.e. $\mu = \sigma s_j$), Property 1 has to be satisfied, in order to guarantee that μ has same shape as σ (cf. Figure 8).

Property 1. *Let σ be a i-simplex, and let B_σ be its corresponding Bézier simplex defined by the set of control points $\{P_\alpha\}$. Let $\mu = \sigma s_j$, $0 \leq j \leq i$, and let B_μ be its corresponding Bézier simplex defined by the set of control points $\{P_{\alpha'}\}$. If all control points $P_{\alpha'_0 \cdots \alpha'_j \alpha'_{j+1} \cdots \alpha'_{i+1}}$ such that :*

- $\alpha'_k = \alpha_k$, *for* $0 \leq k \leq j-1$,
- $\alpha'_j + \alpha'_{j+1} = \alpha_j$, *for* $0 \leq \alpha'_j \leq \alpha_j$ *and* $0 \leq \alpha'_{j+1} \leq \alpha_j$,
- $\alpha'_k = \alpha_{k-1}$, *for* $j+2 \leq k \leq i+1$

are equal to $P_{\alpha_0 \cdots \alpha_i}$, then $B_\sigma = B_\mu$. $\square$

Proof. The complete proof is not provided, since it is quite straightforward: for a given P_α, it is sufficient to show that:

$$\sum_{\alpha'_j+\alpha'_{j+1}=\alpha_j} \binom{d}{\alpha_0 \cdots \alpha'_j \alpha'_{j+1} \cdots \alpha_i} u_0^{\alpha_0} \cdots u_j^{\alpha'_j} u_{j+1}^{\alpha'_{j+1}} \cdots u_i^{\alpha_i} = \binom{d}{\alpha} u_0^{\alpha_0} \cdots (u_j + u_{j+1})^{\alpha_j} \cdots u_i^{\alpha_i}$$

. This is quite direct, since $(u_j + u_{j+1})^{\alpha_j} = \sum_{k=0}^{\alpha_j} \binom{\alpha_j}{k} u_j^{\alpha_j - k} u_{j+1}^k$ □

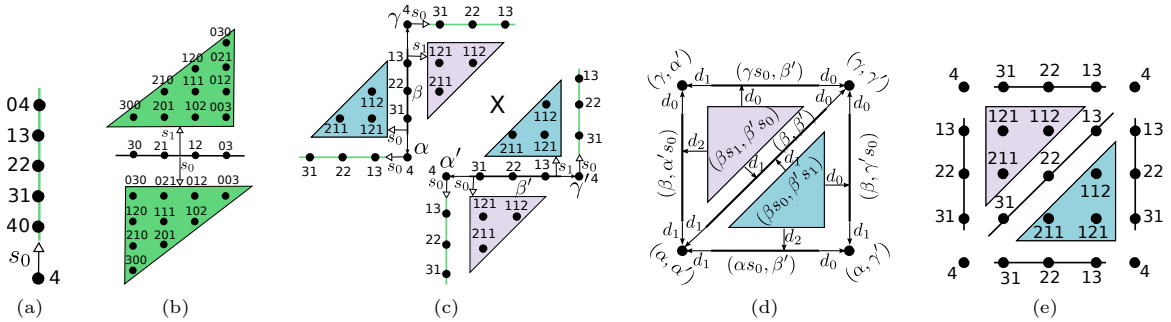


Figure 8: (a) According to Property 1, if all the control points of the degenerate edge are equal to the vertex, then the Bézier simplex corresponding to the edge is equal to the Bézier simplex corresponding to the vertex. (b) illustrates an edge and its two possible degenerate triangles. According to Property 1, if all control points (of a triangle) which are aligned vertically are equal to the control point on the edge in the same alignment, then the Bézier triangles and the Bézier edge are equal. (c) Cartesian product of two edges: the *useful* degenerate simplices are represented up to dimension 2. According to Property 1, all proper control points of the degenerate edge αs_0 (resp. γs_0) are equal to control point P_4 of vertex α (resp. γ); and proper control points P_{211} and P_{121} (resp. P_{112}) of the degenerate edge βs_0 are equal to control point P_{31} (resp. P_{22}) of β. Similarly, proper control points P_{121} and P_{112} (resp. P_{211}) of degenerate simplex βs_1 are equal to control point P_{13} (resp. P_{22}) of β. (d) the simplicial set resulting from the cartesian product of the two edges of (c). Each $2-$simplex in (d) is the simplex resulting from the cartesian product of the two $2-$simplices of the same color in (c). For example, $\beta s_0 \times \beta' s_1$ is the simplex $(\beta s_0, \beta' s_1)$. For clarity purpose, the control points are displayed in (e).

So, a triangular Bézier space can be associated with any simplicial set. From a practical point of view, only the control points of non degenerate simplices are needed. The control points of any simplex (degenerate or not) can be retrieved from its proper control points and from the relations corresponding to face and degeneracy operators: in particular, when the simplex is degenerate, its control points can be deduced from the control points associated with the highest dimensional non degenerate simplex of its boundary, using Property 1. So, the additional cost for embedding simplicial sets, compared with semi-simplicial sets, is null, except the computing time for accessing the control points of a degenerate simplex: in this case, the access time is lower than the dimension of the simplex, with an adequate data structure (cf. Section 4.3). Note that, as for the semi-simplicial case, it is assumed that the degrees of all Bézier simplices are equal, whether they are degenerate or not (and as before, it is possible to avoid this constraint by applying results about the degree elevation: cf. [11]).

Data structure. The following data structure is a a direct extention of the semi-simplicial set data structure presented in Section 2. As mentioned before, only degenerate simplices that are faces of non degenerate simplices are usefull for handling Bézier spaces. So, for any simplex σ, $\sigma.degenOps$ references its "usefull" degenerate simplices in the following way: each element (i, τ) corresponds to $\sigma s_i = \tau$. Note that if a simplex σ is degenerate, then its access $ctrlPts$ is null, since its control points can be deduced from those of the highest non degenerate simplex of its boundary.

```
type Simplex is record :
   dim :  integer
   faceOps :  [0..dim] of *Simplex
   degenOps :  Map<Integer,*Simplex>
   ctrlPts :  *[0..C_dim^(set.degree-1) - 1] of CtrPt
   set :  *SimplicialSet
end record
```

```
type SimplicialSet is record :
   dim :  integer
   degree :  integer
   allSimplices :  [0..dim] of List<*Simplex>
end record
```

Basic Operations are still cone and identification.
The cone definition is a direct extension of the cone of a semi-simplicial set.
The identification of two i-simplices σ and μ having same boundary (cf. Algorithms 5 and 6) implies now the identification two by two of all degenerate simplices deduced from σ and μ by the same sequence of degeneracy operators. More precisely, let K be the set of simplices of S before identification of σ and μ, K can be partitionned into $K = C_0 \cup \cdots \cup C_n \cup K'$ such that for any i, all simplices belonging to C_i have to be identified, and all simplices of K' remains after the operation. So, after the identification of σ and μ, the resulting simplicial set contains one simplex γ_i for each equivalence class C_i, and all simplices of K'.

Notation 1. *Let $S = (K, (d_j), (s_j))$ be a simplicial set, and let $K = C_0 \cup \cdots \cup C_n \cup K'$ be a partition of the simplices as described above. Algorithm 6 uses the following notations:*

- *for any simplex $\tau \in C_i$, for any i, $rep(\tau) = \gamma_i$ is the resulting simplex associated with C_i;*
- *for any C_i, τ_i is any simplex of C_i;*
- *$star(C_i)$ denotes the set of simplices of S having a face in C_i;*
- *$boundary(C_i)$ denotes the set of simplices of S having a degenerate simplex in C_i.*

So, the identification of two simplices σ and μ having same boundary consists in computing the set of equivalence classes $C_0 ... C_n$, and then to merge each equivalent class into a simplex γ_i of same dimension. Then, for each simplex γ_i, to set its face and degeneracy operators according to equivalent classes, and finally, to update the face operators of its star (only for simplices belonging to K'), and to update the degeneracy operators of its boundary (only for simplices belonging to K').

Algorithm 5: buildIdentifications

```
Input  : toIdent : List< {Simplex, Simplex} >
Output: eqClasses : List<Set<Simplex>>
toIdentNextDim : List< {Simplex, Simplex} >= [];
eqClasses = [];
if isEmpty(toIdent) then
    return [];
else
    while not isEmpty(toIdent) do
        {σ, μ} = removeFirstElem(toIdent);
        addToClasses(eqClasses,{σ, μ})/* adds {σ, μ} such that equivalent classes are updated */
    for class in eqClasses do
        for (σ, μ) in class do
            for i ∈ 0..dim(σ) do
                if σs_i ≠ null and μs_i ≠ null and σs_i ≠ μs_i then
                    add({σs_i, μs_i}, toIdentNextDim);
    return eqClasses + buildIdentification(toIdentNextDim) /* + is the concatenation of lists */
```

Algorithm 6: identification

```
Input  : S : SimplicialSet ; σ, μ : Simplex
Output: S
[C_0, ··· , C_n]=buildIdentification({σ,μ});
K' = {ν ∈ S | ν ∉ [C_0, ··· , C_n]};
for i in 0..n do
    γ_i = new Simplex(dim(C_i)) /* dim(C_i) is the dimension of the simplices in C_i */
    add(S,γ_i);
for each γ_i do
    for j in 0..dim(γ_i) do
        γ_i d_j = rep(τ_i d_j);
        γ_i s_j = rep(τ_i s_j);
    for τ in star(C_i) s.t. τ ∈ K' do
        for j in 0..dim(τ) do
            if τd_j ∈ C_i then
                τd_j = γ_i
    for τ in boundary(C_i) s.t. τ ∈ K' do
        for j in 0..dim(τ) do
            if τs_j ∈ C_i then
                τs_j = γ_i
    remove(S, C_i);
```

The cartesian product of two simplicial sets S and S' can be easily defined: a i-simplex μ from $S \times S'$ is associated with any pair (σ, σ') of i-simplices of S and S', such that for any j, $\mu d_j = (\sigma d_j, \sigma' d_j)$ and $\mu s_j = (\sigma s_j, \sigma' s_j)$ (cf. Figure 8(d)). The control points associated with μ can be computed by "adding" the corresponding points of σ and σ' (Minkowski sum). For instance on Figure 8(e), point P_{121} of $(\beta s_1, \beta' s_0)$ corresponds to the sum of P_{13} of β (and thus P_{121} of βs_1) and P_{31} of β' (and thus P_{121} of βs_0). In practice, only the degenerate simplices having a dimension lower than the sum of the dimensions of S and S' are needed for computing the cartesian product of S and S'. At last, note that it is possible to directly deduce the cartesian product of two semi-simplicial sets by implicitly handling the degenerate simplices (and thus the degeneracy operators): cf. [17].

These results can be extended for cubical sets [25, 3], which can be defined in the following way. A *cubical set* $S = (K, (d_j^i), (s^i))$ is a family $K = (K^p)_{p \in \mathbb{N}}$ of cubes together with two families of maps, such that $(K, (d_j^i))$ is a semi-cubical set, and (s^i) are *degeneracy operators*: $\forall p \geq 0$, $\forall i$, $0 \leq i \leq p$, $s^i : K^p \to K^{p+1}$, which satisfy: $s^i s^k = s^k s^{i+1}$ if $k \leq i$, $s^i d_j^k = d_j^k s^{i-1}$ if $k \leq i$, $s^i d_j^k = d_j^{k-1} s^i$ if $k > i+1$ and $s^i d_j^{i+1} = \mathrm{Id}$.

4 Simploidal

As seen in section 2, semi-simploidal sets contain and generalize semi-simplicial and semi-cubical sets.

Similarly to simplicial and cubical sets, simploidal sets are defined by adding *degeneracy operators* to semi-simploidal sets. In particular, it is then possible to define the cone operation (cf. Figure 10) thanks to these operators.

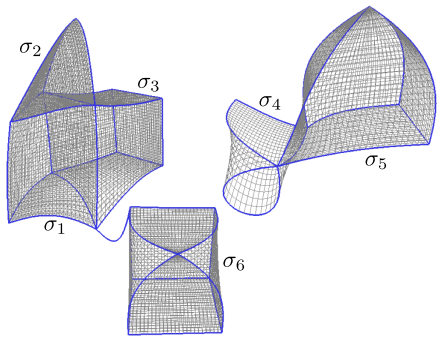


Figure 9: A simploidal Bézier space of dimension 3. Note that σ_1 is a prism (i.e. simploid of type $(2,1)$) connecting the simplex σ_2 and the cube σ_3, σ_4 is a square (i.e. simploid of type $(1,1)$) such that one of its edge is a self loop, σ_6 is a cube (i.e. simploid of type $(1,1,1)$) with a degenerate edge, and σ_5 is a cone over a square face (i.e. a simploid of type $(1,1,1)$ with a degenerate face).

4.1 Combinatorial Structure

Definition 1. *A simploidal set $S = (K, (d^i_j), (s^k_l))$ is a set of* simploids *equipped with a* type *operator $\mathcal{T} : K \mapsto \bigcup_{i=0}^{\infty} \mathbb{N}^{*i}$, face operators d^i_j and degeneracy operators s^k_l.*

Let $\sigma \in K$; $\sigma\mathcal{T}$ is the type *of σ. Let $\sigma\mathcal{T} = (a_1, \cdots, a_n)$: σd^i_j (resp. σs^k_l) is defined if $1 \leq i \leq n, 0 \leq j \leq a_i$ (resp. $0 \leq k \leq n$ and $l = -1$, or $1 \leq k \leq n$ and $0 \leq l \leq a_k$). Operators satisfy:*

(I) **Action on the type**

1) $\sigma d^i_j \mathcal{T} = \begin{cases} (a_1, ..., a_i - 1, ..., a_n) & \text{if } a_i > 1 \\ (a_1, ..., a_{i-1}, a_{i+1}, ..., a_n) & \text{otherwise} \end{cases}$

2) $\sigma s^i_j \mathcal{T} = \begin{cases} (a_1, ..., a_i + 1, ..., a_n) & 1 \leq i \leq n, 0 \leq j \leq a_i \\ (a_1, ..., a_i, 1, a_{i+1}, ..., a_n) & 0 \leq i \leq n, j = -1 \end{cases}$

(II) **Commutation of face operators**

1) $d^i_j d^i_l = d^i_l d^i_{j-1} \quad l < j, a_i > 1$

2) $d^i_j d^k_l = \begin{cases} d^k_l d^i_j & \text{if } a_k > 1 \\ d^k_l d^{i-1}_j & \text{otherwise} \end{cases} \qquad k < i$

(III) **Commutation of degeneracy operators**

1) $s^i_{-1} s^k_{-1} = s^k_{-1} s^{i+1}_{-1} \qquad k \leq i$

2) $\text{if } l \neq -1, s^i_{-1} s^k_l = \begin{cases} s^k_l s^i_{-1} \text{ if } k \leq i \\ s^{k-1}_l s^i_{-1} \text{ if } i < k-1 \end{cases}$

3) $\text{if } j, l \neq -1, s^i_j s^k_l = \begin{cases} s^k_l s^i_j \text{ if } i \neq k \\ s^k_l s^i_{j+1} \text{ if } i = k, l \leq j \end{cases}$

(IV) **Commutation of face/degeneracy operators**

1) $\text{if } l \neq -1, s^i_l d^i_j = \begin{cases} d^i_j s^i_{l-1} & \text{if } j < l \\ d^i_{j-1} s^i_l & \text{if } j > l+1 \\ s^i_l d^i_{j+1} = Id & \text{if } j = l \end{cases}$

2) $\text{if } l = -1, s^{i-1}_l d^i_j = Id$

In the other cases:

3) $\text{if } k \leq i-1, s^k_l d^i_j = \begin{cases} d^i_j s^k_l & \text{if } l \neq -1 \\ d^{i-1}_j s^k_l & \text{otherwise} \end{cases}$

4) $\text{else (i.e. } k \geq i), s^k_l d^i_j = \begin{cases} d^i_j s^k_l & \text{if } a_i > 1 \\ d^i_j s^{k-1}_l & \text{otherwise} \end{cases}$

□

Let σ be a simploid of type $(a_1, ..., a_n)$: n is the *length* of σ, and its *dimension* is $\sum_{i=1}^{n} a_i$, or 0 if $n = 0$. Intuitively, and as explained in section 2.3, a simploid is either a vertex, or the cartesian product of n "generating simplices" of respective dimensions $a_1, ..., a_n$, with $a_i > 0$ for any i. No "0" appears in the type of a simploid, since the cartesian product by a vertex is, from a structural point of view, equal to the identity. This explains the fact that several cases are distinguished in the definition of simploidal sets, even if these cases are intuitively similar. More generally, Definition 1 can be retrieved from the previous remark, the properties of face and degeneracy operators of simplicial sets, and from the fact that the actions of two operators on two different "generating simplices" are independent.

Equations (I) denotes the action of an operator on the simploid type (equations (I-1) for a face operator, equations (I-2) for a degeneracy operator). The action of a face (resp. degeneracy) operator on a "generating simplex" decreases (resp. increases) its dimension; hence, if a zero appears after the application of a face operator (i.e. if $a_i = 1$ and d^i_j is applied), it is removed from the type. Conversely, the application of degeneracy operator s^k_{-1} intuitively consists in adding a degenerate edge as new "generating simplex".

Equations (II) (resp. (III), (IV)) corresponds to the commutation relation of face operators (resp. degeneracy operators, degeneracy and face operators). Several cases correspond to the commutation properties of simplicial sets, i.e. when operators are applied on the same "generating simplex": cf. equation (II-1), the last case of equation (III-3), equation (IV-1). Equation (IV-2) corresponds to the removal, by a face operator, of a new 1-dimensional "generating simplex" issued from a degeneracy operator (i.e. nothing is modified).

The other cases correspond to the independence of the actions of operators applied to distinct "generating simplices":

- equations (II-2) : two cases are distinguished, according to the fact that a 1-dimensional "generating simplex" disappears when a face operator is applied;

- equations (III-1), (III-2) and the first case of equation (III-3) : these cases are distinguished, according to the number of new "generating simplices" created by operator s_{-1}; note that there is no commutation property in equation (III-2) when $k = i + 1$, since $\sigma s_{-1}^{k-1} s_l^k \mathcal{T} = (a_1, \ldots, a_{k-1}, 2, \ldots, a_n)$ (the first degeneracy operator creates a new 1-dimensional "generating simplex", on which another degeneracy operator is applied);
- equations (IV-3) and (IV-4): these cases are distinguished, according to the creation or removal of 1-dimensional "generating simplices".

The following theorem comes directly from the definition of simploidal sets.

Theorem 2. *A unique simploidal set can be associated with:*

- *any simplicial set*
- *any cubical set.*

□

The proof of Theorem 2 is straightforward:

- given a simplicial set $S = (K, (d_j), (s_j))$, a simploidal set $S' = (K', (d'^1_j), (s'^1_j))$ can be defined, such that:
  1. a simploid of type (n) is associated with any n-simplex;
  2. d'^1_j (resp. s'^1_j) corresponds to d_j (resp. s_j) for any j.
- This is similar for cubical sets: a simploid of type $(a_1, \cdots, a_n)$, such that $a_i = 1$ for any i, is associated with any n-cube.

Even if the definition of simploidal sets seems complicated since the face and degeneracy operators have to satisfy many properties, its implementation is direct, contrary to the simpler definition of supercomplexes proposed by Gugenheim in 1957, for topological purposes (cf. section 4.5 for a comparison of simploidal sets and supercomplexes).

Simploidal sets are also efficient, since the properties of the definition are not implemented explicitly. These properties have to be satisfied by the construction operations, but this involves no particular cost. More precisely, and as for simplicial sets, only non degenerate simploids and theirs faces (degenerate or not) have to be explicitly represented. So, the additional cost of simploidal sets is low, even compared with semi-simplicial sets; it is related to the representation of the type of any simploid, and to the "complexity" of the object which has to be represented (cf. section 4.3).

4.2 Bézier embedding

Associating a simploidal set with a simploidal Bézier space can be done by extending the results described for the simplicial case. From a theoretical point of view, assume a Bézier simploid is associated with any simploid; the following property has to be satisfied in order to represent a simploidal Bézier space:

Property 2. *Let $S = (K, (d^i_j), (s^k_l)$ be a simploidal set, let σ be a simploid of type $(a_1, ..., a_n)$ of K, let $P^\sigma = \{P_{(\alpha^1,\ldots,\alpha^n)}, \alpha^i \in \Gamma^{a_i}_{d_i}, 1 \leq i \leq n\}$ be its set of control points, and let B_σ be the Bézier simploid associated with σ.*

(1) *Let $\mu = \sigma d^i_j$, let P^μ be its set of control points, and let B_μ be the Bézier simploid associated with μ.*
If any control point $P^\sigma_{(\alpha^1,\ldots,\alpha^i,\ldots,\alpha^n)}$ is equal to:
- *$P^\mu_{(\alpha^1,\ldots,\alpha^{i-1},\alpha'^i,\alpha^{i+1},\ldots,\alpha^n)}$, where $\alpha'^i = \alpha^i_0 \cdots \alpha^i_{j-1}\alpha^i_{j+1} \cdots \alpha^i_{a_i}$ if $a_i > 1$ and $\alpha^i_j = 0$,*
- *$P^\mu_{(\alpha^1,\ldots,\alpha^{i-1},\alpha^{i+1},\ldots,\alpha^n)}$, if $a_i = 1$,*

then $B_\sigma(u) = B_\mu(u')$ where:
- *$u = (u^1, \cdots, u^n)$,*
- *$u' = (u^1, \cdots, u^{i-1}, u'^i, u^{i+1} \cdots, u^n)$, where $u'^i = (u^i_0, \cdots, u^i_{j-1}, u^i_{j+1}, \cdots, u^i_{a_i})$, if $a_i > 1$ and $u^i_j = 0$,*
- *$u' = (u^1, \cdots, u^{i-1}, u^{i+1} \cdots, u^n)$ if $a_i = 1$.*

(2) *Let $\mu = \sigma s^i_j$, let P^μ be its set of control points, and let B_μ be the Bézier simploid associated with μ. If:*
- *if $j \geq 0$: all control points $P^\mu_{(\alpha^1,\ldots,\alpha'^i,\ldots,\alpha^n)}$ are equal to the control point $P^\sigma_{(\alpha^1,\ldots,\alpha^i,\ldots,\alpha^n)}$, where $\alpha'^i_k = \alpha^i_k$ for $k < j$, $\alpha'^i_j + \alpha'^i_{j+1} = \alpha^i_j$, $\alpha'^i_k = \alpha^i_{k-1}$ for $k > j + 1$, for any $a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n$,*
- *if $j = -1$: all control points $P^\mu_{(\alpha^1,\ldots,\alpha^i,\alpha'^{i+1},\alpha^{i+1},\ldots,\alpha^n)}$ of σs^i_{-1} are equal to the control point $P^\sigma_{(\alpha^1,\ldots,\alpha^n)}$, where $\alpha'^{i+1} \in \Gamma^1_{d'_{i+1}}$,*

then $B_\sigma = B_\mu$

□

As a consequence of Property 2, *it is sufficient in practice to associate its proper control points with any non degenerate simploid.* The set of control points of any simploid σ can be reconstructed according to Property 2 and to the numbering of face operators:

- if σ is a vertex, a point is associated with it;
- else, if σ is not degenerate, its associated control points can be retrieved from its proper control points, and from the proper control points of the simploids of its boundary, using the numbering of face operators.
  If σ is degenerate, it is necessary to access the control points of its associated non degenerate simploid, using the numbering of degeneracy operators and the Property 2, which links the control points of simploids σ and σs^i_j.

.

Remember also that the computation of all Bézier simploids of a simploidal set can be useless, depending on the application: for example, only non degenerate main simploids are required in order to display a simploidal set.

At last, note that control points of non degenerate simploids can be stored into a *simploidal* array according to the lexicographical order of tuples of multi-indices [6]. For instance, let $P = P_{(\alpha^1,\ldots,\alpha^n)}$ be a control point of a Bézier simploid of type $(a_1, \ldots, a_n)$ and degree $(d_1, \ldots, d_n)$. If $n = 1$, then the simploid corresponds to a simplex, and P is stored as described in section 2.1. If $n > 1$, it is necessary to store all control points of a simploid of type $(a_2, \ldots, a_n)$ of degree $(d_2, \ldots, d_n)$ for each multi-index of $\Gamma^{a_1}_{d_1}$ preceding α^1; then point P can be stored as if it would be a control point of a Bézier simploid of type $(a_2, \ldots, a_n)$ and degree $(d_2, \ldots, d_n)$, with a tuple of multi-indices equal to $(\alpha^2, \ldots, \alpha^n)$.

4.3 Data Structure

The following data structure is a possible implementation of simploidal sets associated with simploidal Bézier spaces, where each simploid is associated with its proper control points. In this way, no control point is duplicated. In order to compute the Bézier space corresponding to a simploidal set, the control points of main Bézier simploids can be retrieved thanks to the boundary operators. Note that if a simploid σ is degenerate (i.e. $\sigma = \mu s_j^i$ for some i and j), then σ and μ have the same shape, so there is no need to store its control points of σ (i.e. $\sigma.ctrlPts$ is null).

```
type Simploid is record :
   n :  integer
   type :  [1..n] of integer
   faceOps :  [1..n] of ([] of *Simploid)
   degenOps :  Map<((integer,integer),*Simploid)>
   ctrlPts :  *[0..C_{a_1}^{d_1-1} × ... × C_{a_n}^{d_n-1} − 1] of CtrPt
   set :  *SimploidalSet
end record
```

```
type SimploidalSet is record :
   dim :  integer
   degree :  integer
   allSimploids :  [0..dim] of <*Simploid>
end record
```

Regarding the `SimploidalSet` structure, `dim` is the dimension of the simploidal set, `allSimploids` stores the lists of simploids, ordered by increasing dimensions; `degree` is the degree of each "generating simplex". It is thus assumed here that, whatever its type, the degree of a simploid of length n is ($d_1 =$ `degree`$, \ldots, d_n =$ `degree`); this simplifies several operations. Else it could be necessary to modify some degrees [11], for instance for identifying two simploids having same types but different degrees.

Regarding the `Simploid` structure, let σ be a simploid of type $(a_1, ..., a_n)$ and degree $(d_1, \ldots, d_n)$, `set` gives access to its simploidal set. `n` is the length of its `type`, a_i corresponds to σ`.type[i]` and d_i corresponds to σ`.set.degree`.

Let $d = \sum_{i=1}^{\sigma.n} \sigma.type[i]$ be the dimension of σ. `faceOps` gives access to the boundary simploids of dimension $d-1$.

`ctrlPts` is an access to the array containing the proper control points of σ, ordered by lexicographical order, the access is null for degenerate simploids. The n-tuple of multi-indices of a point can be retrieved from its index in the array.

Remember that only simploids which are in the boundaries of non degenerate simploids are explicitly represented. So, it is assumed that there are few degenerate simploids (compared to non degenerate ones). So, in the presented structure, `degenOps` stores only useful degeneracy operators: if $\sigma s_j^i = \tau$ and τ is a face of a non degenerate simploid, then the element $((i,j),\tau)$ is stored in `degenOps`. Another solution would be to define `degenOps` as a reference to an array of references to simploids. At each index of the array, a reference is either null, either corresponds to a degenerate simploid. The respective interests of these solutions depend on the objects which are represented.

Note that it could be interesting to access all simploids of dimension $d+1$ in the star of σ; these accesses corresponds to the inverse of the face operators, and can be represented by a list of $*Simploid$ associated with each simploid.

If σ is degenerate, it is important to access the corresponding non degenerate simploid τ. In fact, τ is in the boundary of σ, and can be accessed by using the face operators, thanks to the relations between face and degeneracy operators. In order to optimize the access to τ, i.e. to avoid to check all face operators, it can be interesting to associate with σ a list of numberings, corresponding to the face operators accessing the face from which σ is degenerate.

The structure can be optimized for different subclasses of simploidal sets, for instance for semi-simploidal sets, cubical sets (only the dimensions of cubes are required, since the type of a d-dimensional cube is the d-tuple $(1, \ldots, 1)$). Other optimizations can be proposed, for instance for manifolds: all simploids have the same dimension, and face operators are replaced by adjacency operators.

At last, note that the additional cost, compared with a similar structure implementing simplicial sets, is low, since it corresponds to the types which have to be explicitly associated with all simploids: it is thus at most the dimension of the simploidal set times the number of simploids.

Remark: similarly to triangular Bézier spaces (cf. section 2.1), another possible data structure for handling Bézier simploidal spaces is a list of simploidal arrays, where each array contains the control points of a main Bézier simploid. The main drawbacks are related to:

- the redundancy of the representation, since control points of a non main simploid appear in all simploidal arrays associated with the main simploids of its star: this may lead to inconsistency and space complexity problems;
- the efficiency of several construction operations and topological properties computations, since all simploids are not represented nor the adjacency and incidence relations between simploids.

4.4 Basic Operations

From a theoretical point of view, and similarly to the simplicial case, all degenerate simploids associated with any simploid exist in a simploidal set; so, a simploidal set contains an infinite number of simploids. In practice, only the non degenerate simploids and their faces (degenerate or not) are represented. For some operations, additional degenerate simploids can be of interest. For example, a simploid can be simply degenerate using the identification operation with a degenerate simploid.

Cartesian Product (cf. Algorithm 7) As for semi-simploidal sets, the definition of the cartesian product of two simploidal sets S^1 and S^2 is deduced from the following property: the cartesian product of two simploids σ^1 of type $(a_1, ..., a_k)$, and σ^2 of type $(b_1, ..., b_l)$ is a simploid σ of type $(a_1, ..., a_k, b_1, ..., b_l)$.

So, the cartesian product of S^1 and S^2 consists in creating all simploids $\sigma^1 \times \sigma^2$ for each $\sigma^1 \in S^1$ and $\sigma^2 \in S^2$. The creation of each new simploid σ involves the computation of its control points if σ is not degenerate. Then, its face operators are set according to the following property : $\sigma d_j^i = \sigma^1 d_j^i \times \sigma^2$ for $1 \leq i \leq k$, $\sigma d_j^{i+k} = \sigma \times \sigma^2 d_j^i$ for $1 \leq i \leq l$. Then, degeneracy operators are set according to the following property: $\sigma s_j^i = \sigma^1 s_j^i \times \sigma^2$ for $0 \leq i \leq k$, $\sigma s_j^{i+k} = \sigma \times \sigma^2 s_j^i$ for $0 \leq i \leq l$. Note that $\tau^1 = \sigma^1 s_{-1}^k \times \sigma^2$ and $\tau^2 = \sigma^1 \times \sigma^2 s_{-1}^0$ correspond to the same simploid, so τ^1 and τ^2 have to be merged if they both are not null.

This operation can be easily extended in order to define the cartesian product of subsets of given simploidal sets.

Identification As for semi-simploidal sets, two simploids σ and μ can be identified under some structural conditions (equality of type and boundary) and geometrical conditions (equality of degree). The algorithm can directly be deduced from Algorithm 6

Algorithm 7: Cartesian Product

Input : S^1, S^2 : SimploidalSet
Output: S : SimploidalSet

$\sigma, \sigma_1, \sigma_2$: Simploid;

S.dim $= S^1$.dim $+ S^2$.dim;
S.degree $= S^1$.degree;
for $\sigma^1 \in S^1$ **do**
  for $\sigma^2 \in S^2$ **do**
    $\sigma =$ create Simploid $\sigma^1 \times \sigma^2$;
    `add`(S, σ)

`/* Setting face and degeneracy operators */`
for $\sigma = \sigma^1 \times \sigma^2 \in S$ **do**
  $(a_1, \cdots, a_k) =$`type`(σ^1);
  $(b_1, \cdots, b_l) =$`type`(σ^2);
  for i *in* $1..k$ **do**
    for j *in* $0..a_i$ **do**
      $\sigma d^i_j = \sigma^1 d^i_j \times \sigma^2$;
  for i *in* $1..l$ **do**
    for j *in* $0..b_i$ **do**
      $\sigma d^{i+k}_j = \sigma^1 \times \sigma^2 d^i_j$;
  for *each* s^i_j *defined for* σ^1 **do**
    $\sigma s^i_j = \sigma^1 s^i_j \times \sigma^2$;
  for *each* s^i_j *defined for* σ^2 **do**
    $\sigma s^{k+i}_j = \sigma^1 \times \sigma^2 s^i_j$;
  if $\sigma^1 s^k_{-1} \neq$ *null and* $\sigma^2 s^0_{-1} \neq$ *null* **then**
    `merge`$(\sigma^1 s^k_{-1}, \sigma^2 s^0_{-1})$;

defined for simplicial sets. The basic identification operation is also defined for two simploids having same boundary. As for simplicial sets, the identification of σ and μ induces the identifications of their degenerate simploids.

When σ and μ are not degenerate, the computation of the control points of the simploid resulting from the identification of σ and μ can be done as in the semi-simploidal framework. If σ or μ is degenerate, then the resulting simploid is degenerate. This basic operation can be generalized in order to identify simploidal subsets: the identification of two simploids induces then the identification of their boundaries.

Note that two simploids σ and μ may have similar "shapes" but different types; in these cases, σ and μ can not be directly identified:

- When the type of σ is a permutation of the type of μ. For instance, a simploid of type $(1, 2)$ is not a simploid of type $(2, 1)$ (the cartesian product is not commutative). A similar remark can be stated for simplicial sets, since simplicial sets exist, which are not isomorphic, but their geometric realizations are.
- When a simploid has degenerate faces: for instance, in Figure 10(d), the 4 lateral faces of the pyramid are simploids of type $(1, 1)$ but look like triangles (type (2)), because one of their edges is degenerate. But the difference of such simploids with triangles is clear regarding the structure of their control points (cf. Figures 10(e) and 10(f)).

Degeneracy From a theoretical point of view, there is no atomic operation of degeneracy. In fact, degenerating a simploid μ consists in an identification of μ with a degenerate simploid having same boundary. More precisely, degenerating μ into one of its faces σ consists in identifying μ with a degenerate simploid of same type, associated with σ by a degeneracy operator.

In practice, μ can be degenerate into σ if it has the same boundary as a degenerate simplex σs^i_j for some i and j. In case σs^i_j is not explicitly represented, then it is just necessary to set the degeneracy operator σs^i_j to μ. Note that if μ is a main simploid, then μ can simply be removed from the simploidal set.

This operation can be generalized in order to degenerate a simploidal set S into a single vertex: this can be achieved by identifying all vertices of S, and then by iteratively identifying all edges and in degenerating the resulting edge, then in applying the same process to all 2-dimensional simploids (according to their types), and so on. Note that the resulting simploidal set $S^{"}$ contains only degenerate simploids except the vertex. More precisely, $S^{"}$ contains one simploid of each type contained in S. In practice, degenerating S into a single vertex can simply be done by creating one simploid of each type contained in S, and then to set all the face and degeneracy operators according to the type. For example, when a prism is degenerate into a vertex, the resulting simploidal set contains one simploid of each type (0), (1), (2), $(1, 1)$, and $(2, 1)$. Then all face and degeneracy operators are set according to the simploid types.

Cone Intuitively, the cone operation of a simploidal set S with a vertex v consists in extruding S by an edge incident to v; and then in degenerating $S \times v$ into v (cf. Figures 10(a) 10(b) 10(c) 10(d))) as described above.

In practice, it is not necessary compute the extrusion and then to degenerate $S \times v$ into v. As said before, degenerating S into v can simply be done by creating one simploid of each type of S, and then by setting all the face and degeneracy operators according to the simploid types. Then, for each simploid $\sigma \in S$ of type $(a_1, \cdots, a_k)$, a new simploid $\overline{\sigma}$ of type $(a_1, \cdots, a_k, 1)$ is created. Its face and degeneracy operators are set according to Algorithm 8. Finally, as mentionned for the cartesian product, it

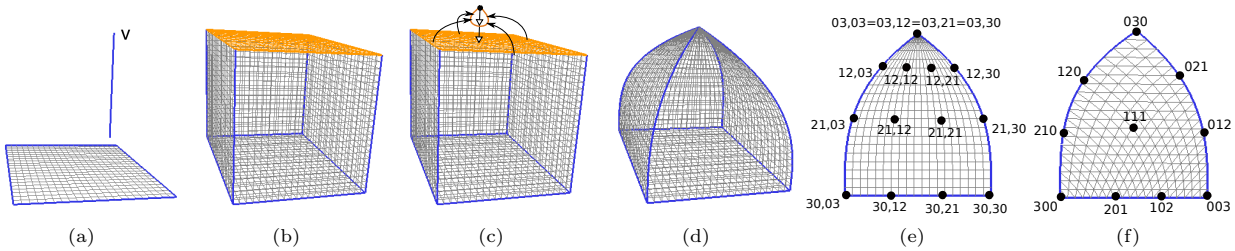


Figure 10: (a) The cone applied to a square (and its boundary) is a square-based pyramid (d). Thanks to degeneracy operators, the cone operation consists in an extrusion $(a)-(b)$ followed by the degeneracy of the copy of the original simploidal set (in orange in (b). In practice, the computation can be improved by creating directly all the degenerate simploids incident to the new vertex and then setting all the boundary and degeneracy operators (c). (e) the structure of the control points of a lateral Bézier square face of (d) is different from the structure of the control points of a Bézier triangle (f).

is possible to define the cone operation for a simploidal subset.

Homology computation: Homology of semi-simploidal sets has been studied in [23]. Its computation is based on the notion of simploid boundary, which can be deduced from the simploidal face operators. This notion can be extended for simploidal sets: only non degenerate simploids are taken into account; in particular if a degenerate simploid τ is a face of a non degenerate simploid σ, then τ does not appear in the boundary of σ.

Obviously, other construction operations can be defined, based on these basic operations, in order to construct simploidal sets.

4.5 Discussion about Gugenheim's definition

Supercomplexes were defined in 1957 (cf. [13] page 37). By mimicking the definition of simploidal sets, (the structure of) supercomplexes could be defined in the following way:

Definition 3. *A supercomplex $S = (K, (d^i_j), (s^i_j))$ is a set of simploids equipped with a type operator $\mathcal{T} : K \mapsto \bigcup_{i=1}^{\infty} \mathbb{N}^i$, face operators d^i_j and degeneracy operators s^k_l. Let $\sigma\mathcal{T} = (a_1, \cdots, a_n)$: σd^i_j (resp. σs^k_l) is defined if $1 \leq i \leq n, a_i > 0, 0 \leq j \leq a_i$ (resp. $1 \leq k \leq n$ and $0 \leq l \leq a_k$). Operators satisfy:*

- $\sigma d^i_j \mathcal{T} = (a_1, ..., a_i - 1, ..., a_n)$,
  $\sigma s^i_j \mathcal{T} = (a_1, ..., a_i + 1, a_{i+1}, ..., a_n)$;
- *face and degeneracy operators having the same exponent satisfy the commutation properties of face and degeneracy operators of simplicial sets;*
- *face and degeneracy operators having different exponents commute.*

□

Algorithm 8: Cone

Input : S : SimploidalSet ; v : Vertex
Output: S' : SimploidalSet

```
S'' = degenerateToVertex(S, v);
S' = S ∪ S'';
K̄ = {} /* for storing new simploids of the cone */
for σ ∈ S do
    (a_1, ··· , a_k)=type(σ);
    σ̄ = create Simploid of type (a_1, ··· , a_k, 1);
    add(K̄,σ̄);
    σ̄d^{k+1}_1 = σ;
    σ̄d^{k+1}_0 = getSimploidOfType(S'', σ)/* S'' contains only one simploid of each type */
add(S',K̄);
for σ̄ ∈ K̄ do
    (a_1, ··· , a_k, 1)=type(σ̄);
    for i in 1..k do
        for j in 0..a_i do
            σ̄d^i_j = σd^i_j;
    for each s^i_j defined for σ do
        σ̄s^i_j = σs^i_j;
```

Obviously, this definition of supercomplexes is simpler than the definition of simploidal sets given in 4.1. This is due to the fact that the type of a simploid may contain components equal to 0. Let σ be a supercomplex of type $(2, 1)$ (i.e a prism), then the

type of σd_0^2 is $(2,0)$ (i.e. a triangle). Conversely, in order to associate a degenerate cube with a square, the type of the square has to be for instance $(1,1,0)$, so the degeneracy operator s_0^2 can be applied. This simple definition of supercomplexes is based on the fact that, implicitly, an equivalence relation exists between types which differ only by components equal to 0: this corresponds to the fact that the cartesian product by a vertex is (structurally speaking) equal to the identity. Note that Gugenheim was not interested in the definition of embedding, construction operations nor implementation, and that this equivalence relation was never explicitly stated. Let Q be the set of supercomplexes quotiented by this equivalence relation.

Theorem 4. *The set of simploidal sets is equivalent to Q.* □

The proof is not provided here because it is rather technical.

The definition of supercomplexes is not well suited for a direct implementation. In particular, the length of the type of a n-dimensional simploid is not bounded, since it can contain any number of zeros. So, contrary to simploidal sets, *it is not possible to bound the complexity* of the space representation nor that of operations as simple as the comparison of two types, which is necessary for instance for the identification operation.

In order to warrant the unicity of the types, no 0 appears in the type of a simploid. So, our definition of simploidal sets contains more constraints over boundary and degeneracy operators, since many cases have to be distinguished in the definition; but, as said above, *this does not involve any additional cost*, since the properties of the operators don't involve any requirement in space nor in computation. The properties described by equations (I) to (IV) are taken into account by the modifications which are performed during the application of an operation, but they don't involve a particular computation. Even if it would be necessary, checking that the properties of a simploidal set are satisfied would not involve additional space or time requirements, compared with supercomplexes.

So, our definition is more efficient regarding implementation issues, and the definition of relations with simploidal Bézier spaces as that of construction operations is quite simple, compared with similar operations defined on simplicial and cubical sets. The counterpart is the fact that the commutation properties of the face and degeneracy operators cannot be expressed as simply as in the supercomplexes definition. Otherwise said, our definition corresponds to the one of `Gugenheim` such that for each equivalence class, we chose the representative simploid with no 0.

5 Conclusion - Future Work

Simploidal sets generalize simplicial sets and cubical sets: they are defined in a similar way, by abstract simploids on which act face and degeneracy operators. All basic operations defined in the simplicial and cubical frameworks are extended for simploidal sets: cone, cartesian product, degeneracy and identification. Relations between simploidal sets and simploidal Bézier spaces generalize similar relations between simplicial (resp. cubical) sets and triangular (resp. cubical) Bézier spaces.

The simploidal framework presented in this paper is thus general and coherent, meaning that it is possible to conceive softwares implementing simploidal sets embedded with Bézier spaces (e.g. for computing topological properties or for geometric modeling), which make it possible to handle also simplicial sets or cubical sets as particular cases. Moreover, it is possible to handle simultaneously different types of cells, for hybrid meshes applications for instance.

The cost of this generalization is not significant, since the definition of simploidal sets is very close to the definitions of simplicial and cubical sets: the main difference, when implementing simploidal sets, is that the type of a simploid has to be explicitly represented.

Note that a difference exists between semi-simplicial sets and semi-simploidal sets. It is possible to directly adapt the definition of the cartesian product of simplicial sets for semi-simplicial sets, by *implicitly* handling degeneracy operators [17]: this comes from the fact that the cartesian product of any two simplices (and their boundaries) can be described as a semi-simplicial set. Unfortunately, it is not possible to directly adapt the definition of the cone of simploidal sets for semi-simploidal sets, because the cone of a simploid is not necessarily a simploid.

The relations between simploidal sets and simploidal Bézier spaces have been stated. The basic data structure presented in this paper is a straightforward implementation of the combinatorial structure, but some optimizations can be considered for subclasses of simploidal sets. For instance, for manifolds, it is sufficient to represent the main cells and their adjacency relations (cf. [5] chapter 10). When dealing with low-dimensional simploidal sets (e.g. 2 or 3), even the basic data structure can obviously be optimized. More generally, optimization techniques related to mesh compression can be applied [18].

Acknowledgments: The authors gratefully acknowledge Laurent Fuchs for helpful discussions and encouragements.

References

[1] M. K. Agoston. *Algebraic Topology, a first course.* Pure and applied mathematics. Marcel Dekker Ed., 1976.

[2] B. Bechmann, D. et Péroche. *Informatique graphique, modélisation géométrique et animation.* Signal et Image, Traité IC2. Lavoisier, 2007.

[3] R. Brown and P. J. Higgins. On the algebra of cubes. *Journal of Pure and Applied Algebra*, 21(3):233 – 260, 1981.

[4] W. Dahmen and C. A. Micchelli. On the linear independence of multivariate b-splines I. Triangulation of simploids. *SIAM J. Numer. Anal.*, 19, 1982.

[5] G. Damiand and P. Lienhardt. *Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing.* HAL CCSD, 2014.

[6] T. DeRose, R. N. Goldman, H. Hagen, and S. Mann. Functional composition algorithms via blossoming. *Transactions On Graphics*, 12(2):113–135, 1993.

[7] J.-G. Dumas, F. Heckenbach, B. D. Saunders, and V. Welker. Computing simplicial homology based on efficient smith normal form algorithms. In *Algebra, Geometry, and Software Systems*, pages 177–206, 2003.

[8] H. Edelsbrunner and H. Harer. *Computational Topology - an Introduction.* American Mathematical Society, 2010.

[9] S. Eilenberg and J. Zilber. Semi-simplicial complexes and singular homology. *Annals of Mathematics*, 51:499–513, 1950.

[10] J. Escobar, J. Cascón, E. Rodríguez, and R. Montenegro. A new approach to solid modeling with trivariate t-splines based on mesh optimization. *Computer Methods in Applied Mechanics and Engineering*, 200(45):3210 – 3222, 2011.

[11] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2002.

[12] P. Frey and P.-L. George. *Mesh Generation: application to finite elements.* Wiley, 2008.

[13] V. K. A. M. Gugenheim. On supercomplexes. *Transactions of the American Mathematical Society*, 85(1):35–51, May 1957.

[14] A. Hatcher. *Algebraic Topology.* Cambridge University Press, 2002.

[15] V. Lang and P. Lienhardt. Geometric modeling with simplicial sets. In T. K. S.Y. Shin, editor, *Computer Graphics and Applications*, Pacific Graphics, pages 475–494, Seoul, Corea, 1995. World Scientific Publishing.

[16] V. Lang and P. Lienhardt. Simplicial sets and triangular patches. In *Computer Graphics International Conference, CGI 1996, Pohang, Korea, June 24-28, 1996*, pages 154–163, 1996.

[17] P. Lienhardt, X. Skapin, and A. Bergey. Cartesian product of simplicial and cellular structures. *Int. J. Comput. Geometry Appl.*, 14(3):115–159, 2004.

[18] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot. 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Computing Surveys (CSUR)*, 47(3):1–41, 2015.

[19] W. S. Massey. *A basic course in algebraic topology.* Springer-Verlag, 1991.

[20] J. P. May. *Simplicial Objects in Algebraic Topology.* Van Nostrand, 1967.

[21] J. R. Munkres. *Elements of algebraic topology.* Perseus Books, 1984.

[22] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Trans. Graph.*, 12(1):56–102, 1993.

[23] S. Peltier, L. Fuchs, and P. Lienhardt. Simploidals sets: Definitions, operations and comparison with simplicial sets. *Discrete App. Math.*, 157:542–557 (extended version of "Homology of Simploidal Sets", DGCI 2006, Szeged, Hungary, 235–246), feb. 2009.

[24] J. Rubio and F. Sergeraert. Constructive homological algebra and applications, August, 28 - September, 02 2006. http://arxiv.org/abs/1208.3816.

[25] J.-P. Serre. Homologie singulière des espaces fibrés. *The Annals of Mathematics*, 54(3):425–505, November 1951.

[26] L. Untereiner, P. Kraemer, D. Cazier, and D. Bechmann. CPH: A compact representation for hierarchical meshes generated by primal refinement. *Comput. Graph. Forum*, 34(8):155–166, 2015.

[27] T. Várady, P. Salvi, and I. Kovács. Enhancement of a multi-sided bézier surface representation. *Computer Aided Geometric Design*, 55:69 – 83, 2017.

[28] S. Zeng and E. Cohen. Hybrid volume completion with higher-order bézier elements. *Computer Aided Geometric Design*, 35-36(Supplement C):180 – 191, 2015. Geometric Modeling and Processing 2015.