

# Geometric and Graphics Programming Laboratory: Lecture 8

Alberto Paoluzzi

October 27, 2017

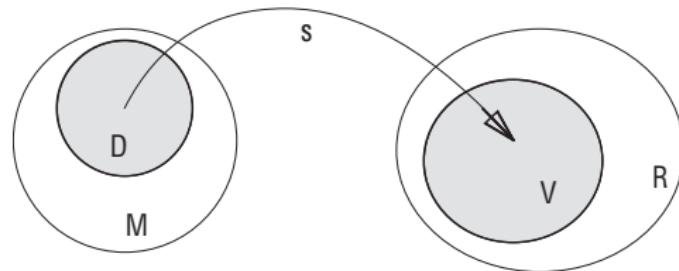
# Outline: LAR1

- 1 Solid Modeling
- 2 Space decompositions
- 3 Cellular complex
- 4 Simplicial mapping

# Solid Modeling

# Representation scheme: definition

mapping  $s : M \rightarrow R$  from a space  $M$  of mathematical models to a space  $R$  of computer representations



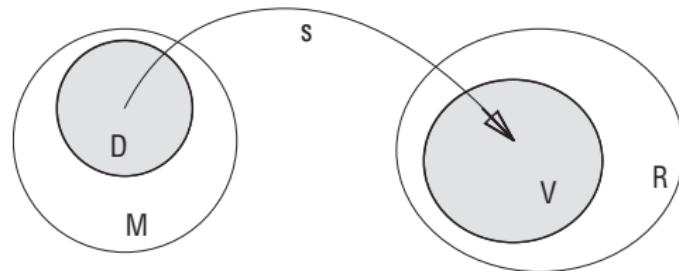
- ① The  $M$  set contains the mathematical models of the class of solid objects the scheme aims to represent

A. Requicha, [Representations for Rigid Solids: Theory, Methods, and Systems, ACM Comput. Surv.](#), 1980.

V. Shapiro, [Solid Modeling](#), In [Handbook of Computer Aided Geometric Design](#), 2001

# Representation scheme: definition

mapping  $s : M \rightarrow R$  from a space  $M$  of mathematical models to a space  $R$  of computer representations



- ① The  $M$  set contains the mathematical models of the class of solid objects the scheme aims to represent
- ② The  $R$  set contains the symbolic representations, i.e. the proper data structures, built according to a suitable grammar

A. Requicha, [Representations for Rigid Solids: Theory, Methods, and Systems, ACM Comput. Surv.](#), 1980.

V. Shapiro, [Solid Modeling](#), In [Handbook of Computer Aided Geometric Design](#), 2001

# Representation schemes

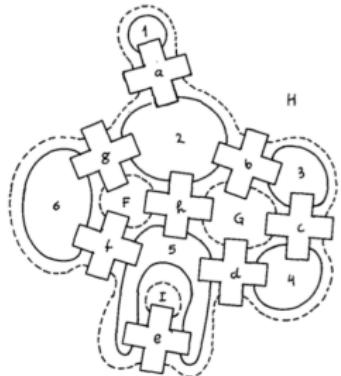
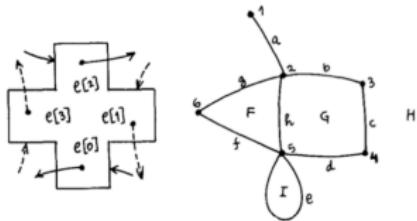
Most of such papers introduce or discuss one or more representation schemes . . .

- 1 Requicha, ACM Comput. Surv., 1980, [?]
- 2 Requicha & Voelcker, PEP TM-25, 1977, [?]
- 3 Rossignac & Requicha, Comput. Aided Des., 1991, [?]
- 4 Bowyer, SVLIS, 1994, [?]
- 5 Baumgart, Stan-CS-320, 1972, [?]
- 6 Braid, Commun. ACM, 1975, [?]
- 7 Dobkin & Laszlo, ACM SCG, 1987, [?]
- 8 Guibas & Stolfi, ACM Trans. Graph., 1985, [?]
- 9 Woo, IEEE Comp. Graph. & Appl., 1985, [?]
- 10 Yamaguchi & Kimura, Comp. Graph. & Appl., 1995, [?]
- 11 Gursoz & Choi & Prinz, Geom.Mod., 1990, [?]
- 12 S.S.Lee & K.Lee, ACM SMA, 2001, [?]
- 13 Rossignac & O'Connor, IFIP WG 5.2, 1988, [?]
- 14 Weiler, IEEE Comp. Graph. & Appl., 1985, [?]
- 15 Silva, Rochester, PEP TM-36, 1981, [?]
- 16 Shapiro, Cornell Ph.D Th., 1991, [?]
- 17 Paoluzzi et al., ACM Trans. Graph., 1993, [?]
- 18 Pratt & Anderson, ICAP, 1994, [?]
- 19 Bowyer, Djinn, 1995, [?]
- 20 Gomes et al., ACM SMA, 1999, [?]
- 21 Raghothama & Shapiro, ACM Trans. Graph., 1998, [?]
- 22 Shapiro & Vossler, ACM SMA, 1995, [?]
- 23 Hoffmann & Kim, Comput. Aided Des., 2001, [?]
- 24 Raghothama & Shapiro, ACM SMA, 1999, [?]
- 25 DiCarlo et al., IEEE TASE, 2008, [?]
- 26 Bajaj et al., CAD&A, 2006, [?]
- 27 Pascucci et al., ACM SMA, 1995, [?]
- 28 Paoluzzi et al., ACM Trans. Graph., 1995, [?]
- 29 Paoluzzi et al., Comput. Aided Des., 1989, [?]
- 30 Ala, IEEE Comput. Graph. Appl., 1992, [?]

and much more . . .

# Representation scheme: Quad-Edge data structure

(Guibas & Stolfi, ACM Transactions on Graphics, 1985)



- (a) Edge record showing Next links.
- (b) A subdivision of the sphere.
- (c) Data structure for the subdivision

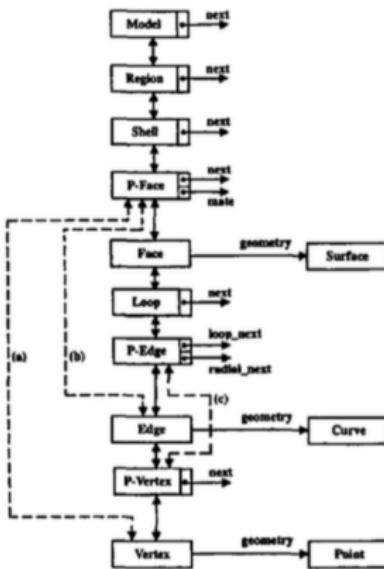
Primitives for the Manipulation  
of General Subdivisions and  
the Computation of Voronoi  
Diagrams

largely used in computational  
geometry algorithms and in  
geometric libraries

# Representation scheme: Partial-Entity data structure

(Sang Hun Lee & Kunwoo Lee, ACM Solid Modeling, 2001)

## Compact Non-Manifold Boundary Representation Based on Partial Topological Entities



```

class Entity {
    int         _id;
    Attribute *attribute;
};

class Model : public Entity {
    Model *_next;           // next model
    Region *_region;        // list of regions
};

class Region : public Entity {
    Region *_next;          // link field of the region list of a model
    Model *_model;          // parent model
    Shell *_shell;           // peripheral shell
};

class Shell : public Entity {
    Shell *_next;            // next void shell
    Region *_region;         // parent region
    Pface *_pface;           // partial face
};

class Pface : public Entity { // partial face (p-face) class
    Pface *_next;            // next p-face
    Shell *_shell;           // parent shell
    Entity *_child;          // child entity: a face, an edge, or a vertex
    Orient *_orient;          // orientation flag w.r.t. the face normal
    Pface *_mate;             // mate p-face
};

class Face : public Entity {
    Pface *_pface;           // one of two incident p-faces
    Loop *_loop;              // peripheral loop
    Surface *_geometry;        // surface
};

class Loop : public Entity { // next hole loop
    Loop *_next;              // next hole loop
    Face *_face;                // parent face
    Pedge *_pedge;              // a p-edge in a loop
};

class Pedge : public Entity { // partial edge (p-edge) class
    Pedge *_loop;              // parent loop
    Entity *_child;              // child entity: an edge or a p-vertex
    Orient *_orient;              // orientation flag w.r.t. the edge direction
    Pvertex *_pvertices;        // start p-vertex
    Pedge *_looped_prev;        // previous p-edge in the loop cycle
    Pedge *_looped_next;        // next p-edge in the loop cycle
    Pedge *_radial_prev;        // previous p-edge in the radial cycle
    Pedge *_radial_next;        // next p-edge in the radial cycle
};

class Pvertex : public Entity { // partial vertex (p-vertex) class
    Pvertex *_next;            // another p-vertex associated with _vertex
    Entity *_parent;            // parent entity: a p-edge or a p-face
    Pvertex *_pvertices[2];      // two end p-vertices
    Curve *_geometry;           // curve
};

class Curve : public Entity { // partial vertex (p-vertex) class
    Entity *_parent;            // parent entity: a p-vertex or a p-face
    Vertex *_vertex;             // mother vertex
};

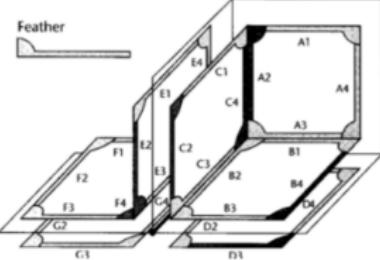
class Edge : public Entity { // partial edge (p-edge) class
    Entity *_parent;            // parent entity: a p-edge or a p-face
    Pvertex *_pvertices[2];      // two end p-vertices
    Curve *_geometry;           // curve
};

class Vertex : public Entity { // partial vertex (p-vertex) class
    Entity *_parent;            // parent entity: a p-vertex or a p-face
    Point *_geometry;            // position
};

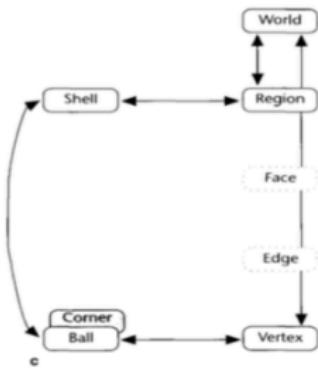
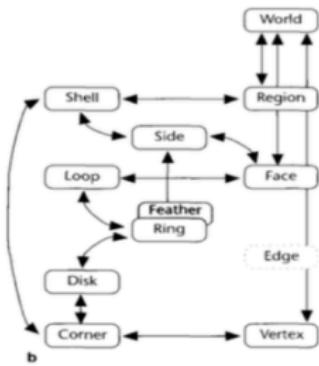
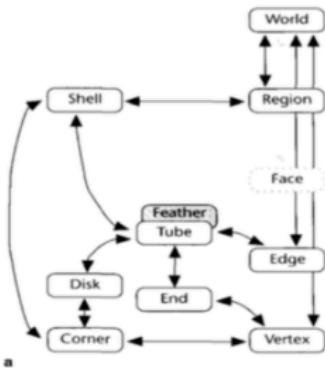
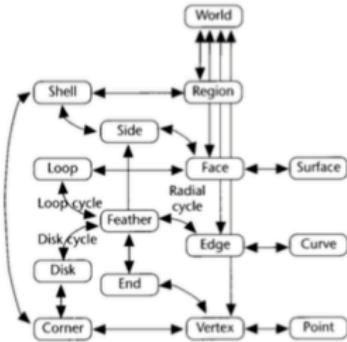
```

# Representation scheme: Coupling Entities data structure

(Yamaguchi & Kimura, IEEE Computer Graphics and Applications, 1995)



Fan	$B_4 \& D_3$	$:D_3 = FM(B_4), B_4 = FM(D_3)$
Blade	$C_2 \& E_2$	$:E_2 = BM(C_2), C_2 = BM(E_2)$
Wedge	$A_2 \& C_4$	$:C_4 = WM(A_2), A_2 = WM(C_4)$
Loop cycle	$A_1 \gg A_2 \gg A_3 \gg A_4$ $A_4 \gg A_3 \gg A_2 \gg A_1$	$:A_2 = CCL(A_1), A_3 = CCL(A_2)$ $:A_1 = CL(A_2), A_2 = CL(A_3)$
Radial cycle	$D_2 \gg C_3 \gg F_4$ $F_4 \gg C_3 \gg D_2$	$:C_3 = CCR(D_2), F_4 = CCR(C_3)$ $:D_2 = CR(C_3), C_3 = CR(F_4)$
Disk cycle	$A_3 \gg C_4 \gg B_2$ $B_2 \gg C_4 \gg A_3$	$:C_4 = CCD(A_3), B_2 = CCD(C_4)$ $:A_3 = CD(C_4), C_4 = CD(B_2)$



# Space decompositions

# Join of pointsets

The **join** of two sets  $P, Q \subset \mathbb{E}^n$  is the convex hull of their points:

$$PQ = \text{join}(P, Q) := \{\gamma p + \lambda q, \ p \in P, \ q \in Q\}$$

$$\gamma, \lambda \in \mathbb{R}, \ \gamma, \lambda \geq 0, \ \gamma + \lambda = 1$$

The join operation is **associative** and **commutative**.

# Join of pointsets: examples

```

pts = [[0,0],[.5,0],[0,.5],[.5,.5],
       [1,.5],[1.5,.5],[1.5,1],[.25,1]]      # coords
P = AA(MK)(pts)                           # 0-polyhedra
S = AA(JOIN)([P[0:4],P[4:7],P[7]])        # array of d-polyhedra
H = JOIN(S)                                # 2-polyhedron

```

```

VIEW(STRUCT(AA(SKELETON(1))(S)))
VIEW(H)

```

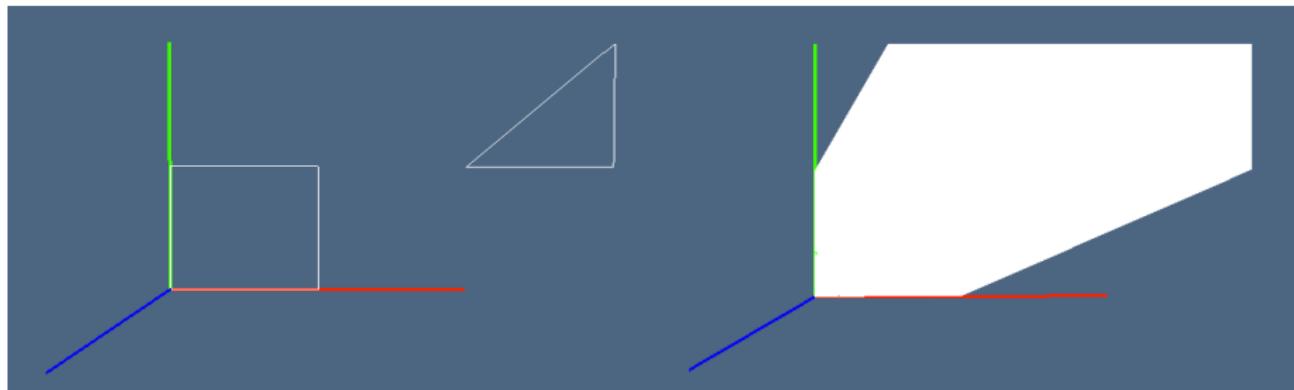


Figure 1: (a) 1-skeleton of pointsets in  $S$ ; (b) convex hull  $H$  of pointset  $P$

# Simplex

A **simplex**  $\sigma \subset \mathbb{E}^n$  of order  $d$ , or  $d$ -simplex, is the **join** of  $d + 1$  **affinely independent points**, called **vertices**.

The  $n + 1$  points  $p_0, \dots, p_n$  are **affinely independent** when the  $n$  vectors  $p_1 - p_0, \dots, p_n - p_0$  are **linearly independent**.

A  **$d$ -simplex** can be seen as a  **$d$ -dimensional triangle**: 0-simplex is a **point**, 1-simplex is a **segment**, 2-simplex is a **triangle**, 3-simplex is a **tetrahedron**, and so on.

# Simplex: examples

```
s0,s1,s2,s3 = [SIMPLEX(d) for d in range(4)]      # array of standard d-simplices
VIEW(s1); VIEW(s2); VIEW(s3);

points = [[1,1,1],[0,1,1],[1,0,0],[1,1,0]]        # coords of 4 points
tetra = JOIN(AA(MK)(points))                         # 3-simplex
VIEW(tetra)
```

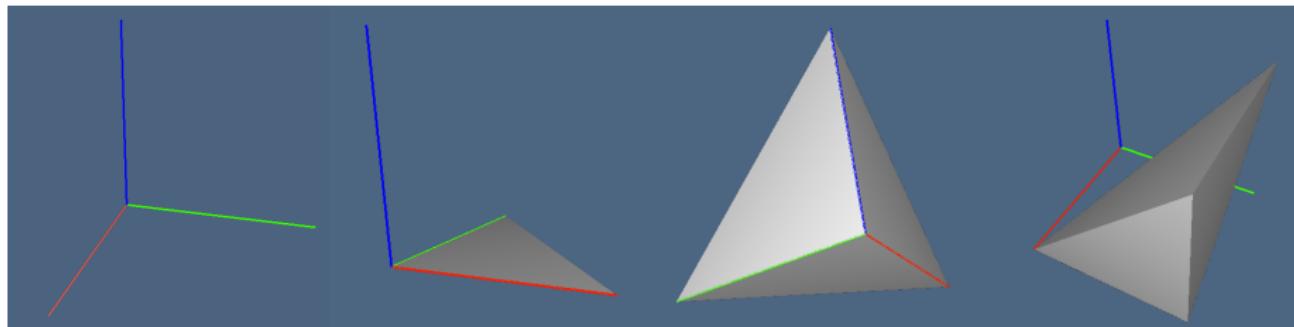


Figure 2: (a,b,c) 1-, 2-, and 3-standard simplex; (d) 3-simplex defined by 4 points

# Simplicial complex

Any subset of  $s + 1$  vertices ( $0 \leq s \leq d$ ) of a  $d$ -simplex  $\sigma$  defines an  $s$ -simplex, which is called  $s$ -face of  $\sigma$ .

# Simplicial complex

Any subset of  $s + 1$  vertices ( $0 \leq s \leq d$ ) of a  $d$ -simplex  $\sigma$  defines an  $s$ -simplex, which is called  **$s$ -face** of  $\sigma$ .

A **simplicial complex** is a set of simplices  $\Sigma$ , verifying the following conditions:

- ① if  $\sigma \in \Sigma$ , then any  $s$ -face of  $\sigma$  belongs to  $\Sigma$ ;

# Simplicial complex

Any subset of  $s + 1$  vertices ( $0 \leq s \leq d$ ) of a  $d$ -simplex  $\sigma$  defines an  $s$ -simplex, which is called  **$s$ -face** of  $\sigma$ .

A **simplicial complex** is a set of simplices  $\Sigma$ , verifying the following conditions:

- ① if  $\sigma \in \Sigma$ , then any  $s$ -face of  $\sigma$  belongs to  $\Sigma$ ;
- ② if  $\sigma, \tau \in \Sigma$ , then  $\sigma \cap \tau \in \Sigma$ .

# Simplicial complex

Any subset of  $s + 1$  vertices ( $0 \leq s \leq d$ ) of a  $d$ -simplex  $\sigma$  defines an  $s$ -simplex, which is called  **$s$ -face** of  $\sigma$ .

A **simplicial complex** is a set of simplices  $\Sigma$ , verifying the following conditions:

- ① if  $\sigma \in \Sigma$ , then any  $s$ -face of  $\sigma$  belongs to  $\Sigma$ ;
- ② if  $\sigma, \tau \in \Sigma$ , then  $\sigma \cap \tau \in \Sigma$ .

# Simplicial complex

Any subset of  $s + 1$  vertices ( $0 \leq s \leq d$ ) of a  $d$ -simplex  $\sigma$  defines an  $s$ -simplex, which is called  **$s$ -face** of  $\sigma$ .

A **simplicial complex** is a set of simplices  $\Sigma$ , verifying the following conditions:

- ① if  $\sigma \in \Sigma$ , then any  $s$ -face of  $\sigma$  belongs to  $\Sigma$ ;
- ② if  $\sigma, \tau \in \Sigma$ , then  $\sigma \cap \tau \in \Sigma$ .

Geometric carrier  $|\Sigma|$  is the pointset union of simplices in  $\Sigma$ .

# Simplicial complex: examples

```

from larlib import *
V,CV = larSimplexGrid1([5,5,5])
FV = larSimplexFacets(CV)
EV = larSimplexFacets(FV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,CV))))
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,FV))))
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,EV))))


BV = [FV[t] for t in boundaryCells(CV,FV)]
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,BV))))      # boundary 2-simplices

```

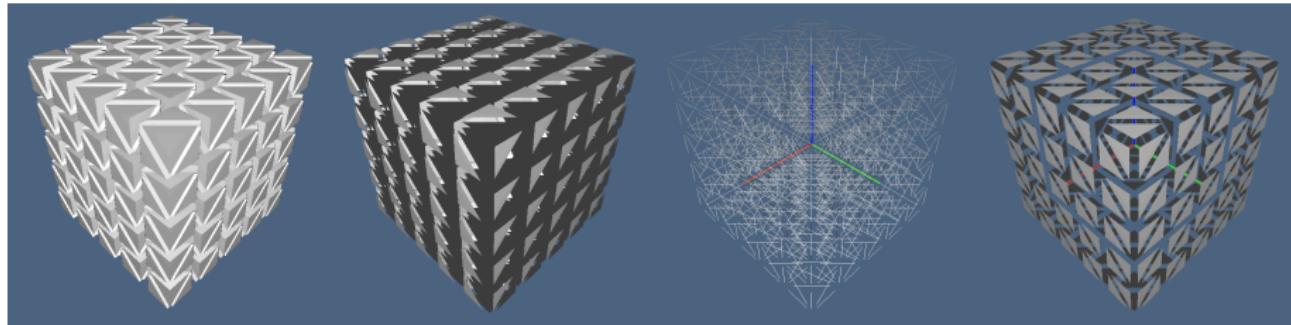


Figure 3: (a) 3-complex; (b) 2-subcomplex; (c) 1-subcomplex; (d) 2-boundary.

# Simplicial complex: examples

(see Disk Point Picking)

```
from larlib import *; from random import random as rand
points = [[2*PI*rand(),rand()] for k in range(1000)]
V = [[SQRT(r)*COS(alpha),SQRT(r)*SIN(alpha)] for alpha,r in points]
cells = [[k+1] for k,v in enumerate(V)]
VIEW(MKPOL([V,cells,None])))

from scipy.spatial import Delaunay
FV = Delaunay(array(V)).vertices
VIEW(EXplode(1.2,1.2,1)(MKPOLS((V,FV))))
VIEW(SKELETON(1)(STRUCT(MKPOLS((V,FV)))))
```

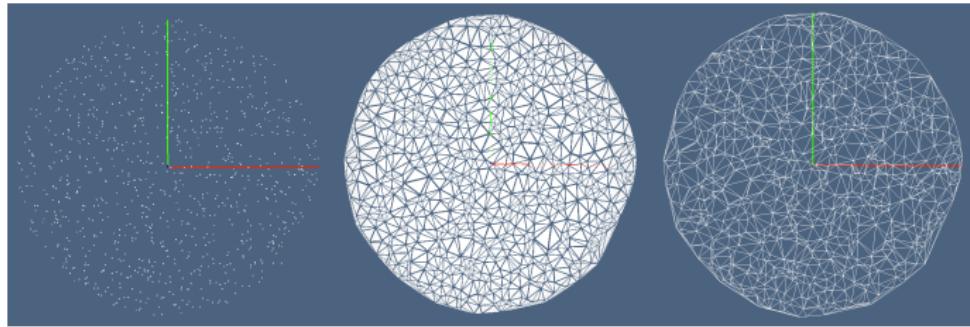


Figure 4: (a) Points; (b) Delaunay triangulation; (c) 1-skeleton.

# Simplicial complex: examples

(see Disk Point Picking)

```
from larlib import *; from random import random as rand
points = [[2*rand()-1,2*rand()-1,2*rand()-1] for k in range(30000)]
V = [p for p in points if VECTNORM(p) <= 1]
VIEW(STRUCT(MKPOLS((V,AA(LIST)(range(len(V)))))))
```

```
from scipy.spatial import Delaunay
CV = Delaunay(array(V)).vertices
def test(tetra): return AND([v[-1] < 0 for v in tetra])
CV = [cell for cell in CV if test([V[v] for v in cell]) ]
VIEW(STRUCT(MKPOLS((V,CV))))
```

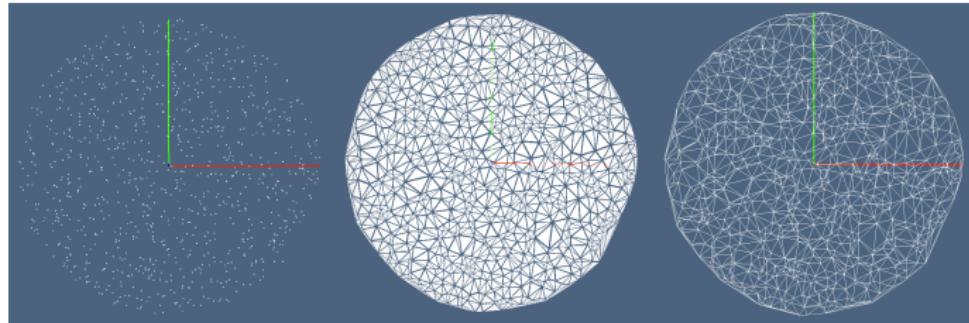


Figure 5: (a) Points; (b) Delaunay triangulation; (c) 1-skeleton

# Cellular complex

# Politopal, simplicial, cuboidal complexes

Cellular complexes characterised by different types of cells:

- ① In polytopal complexes cells are polytopes, i.e. bounded convex sets;

# Politopal, simplicial, cuboidal complexes

Cellular complexes characterised by different types of cells:

- ① In polytopal complexes cells are polytopes, i.e. bounded convex sets;
- ② In simplicial complexes cells are simplices, i.e.  $d$ -polyedra with  $d + 1$  facets ( $(d - 1)$ -faces) and  $d + 1$  vertices.

# Politopal, simplicial, cuboidal complexes

Cellular complexes characterised by different types of cells:

- ① In polytopal complexes cells are polytopes, i.e. bounded convex sets;
- ② In simplicial complexes cells are simplices, i.e.  $d$ -polyedra with  $d + 1$  facets ( $(d - 1)$ -faces) and  $d + 1$  vertices.
- ③ In cuboidal complexes cells are cuboids, (in general, sets homeomorphic to) Cartesian products of intervals, i.e.  $d$ -polyedra with  $2d$  facets and  $2d$  vertices.

# Numbers of vertices and facets

A *d*-simplex, or *d*-dimensional simplex, has  $d + 1$  extremal points called **vertices** and  $d + 1$  facets.

- a **point** (0-simplex) has  $0 + 1 = 1$  vertices and 1 facet ( $\emptyset$ );

# Numbers of vertices and facets

A  $d$ -simplex, or  $d$ -dimensional simplex, has  $d + 1$  extremal points called **vertices** and  $d + 1$  facets.

- a **point** ( $0$ -simplex) has  $0 + 1 = 1$  vertices and  $1$  facet ( $\emptyset$ );
- an **edge** ( $1$ -simplex) has  $1 + 1 = 2$  vertices and  $2$  facets;

# Numbers of vertices and facets

A  $d$ -simplex, or  $d$ -dimensional simplex, has  $d + 1$  extremal points called **vertices** and  $d + 1$  facets.

- a **point** ( $0$ -simplex) has  $0 + 1 = 1$  vertices and  $1$  facet ( $\emptyset$ );
- an **edge** ( $1$ -simplex) has  $1 + 1 = 2$  vertices and  $2$  facets;
- a **triangle** ( $2$ -simplex) has  $2 + 1 = 3$  vertices and  $3$  facets;

# Numbers of vertices and facets

A *d*-simplex, or *d*-dimensional simplex, has  $d + 1$  extremal points called **vertices** and  $d + 1$  facets.

- a **point** (0-simplex) has  $0 + 1 = 1$  vertices and 1 facet ( $\emptyset$ );
- an **edge** (1-simplex) has  $1 + 1 = 2$  vertices and 2 facets;
- a **triangle** (2-simplex) has  $2 + 1 = 3$  vertices and 3 facets;
- a **tetrahedron** (3-simplex) has  $3 + 1 = 4$  vertices and 4 facets, etc.

# Numbers of vertices and facets

A *d*-simplex, or *d*-dimensional simplex, has  $d + 1$  extremal points called **vertices** and  $d + 1$  facets.

- a **point** (0-simplex) has  $0 + 1 = 1$  vertices and 1 facet ( $\emptyset$ );
- an **edge** (1-simplex) has  $1 + 1 = 2$  vertices and 2 facets;
- a **triangle** (2-simplex) has  $2 + 1 = 3$  vertices and 3 facets;
- a **tetrahedron** (3-simplex) has  $3 + 1 = 4$  vertices and 4 facets, etc.

# Numbers of vertices and facets

A  $d$ -simplex, or  $d$ -dimensional simplex, has  $d + 1$  extremal points called vertices and  $d + 1$  facets.

- a point (0-simplex) has  $0 + 1 = 1$  vertices and 1 facet ( $\emptyset$ );
- an edge (1-simplex) has  $1 + 1 = 2$  vertices and 2 facets;
- a triangle (2-simplex) has  $2 + 1 = 3$  vertices and 3 facets;
- a tetrahedron (3-simplex) has  $3 + 1 = 4$  vertices and 4 facets, etc.

A  $d$ -cuboid has conversely  $2^d$  vertices and  $2d$  facets:

- a point (0-cuboid) has  $2^0 = 1$  vertices and 0 facets;

# Numbers of vertices and facets

A  **$d$ -simplex**, or  $d$ -dimensional simplex, has  $d + 1$  **extremal points** called **vertices** and  $d + 1$  facets.

- a **point** ( $0$ -simplex) has  $0 + 1 = 1$  vertices and  $1$  facet ( $\emptyset$ );
- an **edge** ( $1$ -simplex) has  $1 + 1 = 2$  vertices and  $2$  facets;
- a **triangle** ( $2$ -simplex) has  $2 + 1 = 3$  vertices and  $3$  facets;
- a **tetrahedron** ( $3$ -simplex) has  $3 + 1 = 4$  vertices and  $4$  facets, etc.

A  **$d$ -cuboid** has conversely  $2^d$  **vertices** and  $2d$  **facets**:

- a **point** ( $0$ -cuboid) has  $2^0 = 1$  vertices and  $0$  facets;
- an **edge** ( $1$ -cuboid) has  $2^1 = 2$  vertices and  $2$  facets;

# Numbers of vertices and facets

A  **$d$ -simplex**, or  $d$ -dimensional simplex, has  $d + 1$  **extremal points** called **vertices** and  $d + 1$  facets.

- a **point** ( $0$ -simplex) has  $0 + 1 = 1$  vertices and  $1$  facet ( $\emptyset$ );
- an **edge** ( $1$ -simplex) has  $1 + 1 = 2$  vertices and  $2$  facets;
- a **triangle** ( $2$ -simplex) has  $2 + 1 = 3$  vertices and  $3$  facets;
- a **tetrahedron** ( $3$ -simplex) has  $3 + 1 = 4$  vertices and  $4$  facets, etc.

A  **$d$ -cuboid** has conversely  $2^d$  **vertices** and  $2d$  **facets**:

- a **point** ( $0$ -cuboid) has  $2^0 = 1$  vertices and  $0$  facets;
- an **edge** ( $1$ -cuboid) has  $2^1 = 2$  vertices and  $2$  facets;
- a **quadrilateral** ( $2$ -cuboid) has  $2^2 = 4$  vertices and  $4$  facets;

# Numbers of vertices and facets

A  **$d$ -simplex**, or  $d$ -dimensional simplex, has  $d + 1$  **extremal points** called **vertices** and  $d + 1$  facets.

- a **point** ( $0$ -simplex) has  $0 + 1 = 1$  vertices and  $1$  facet ( $\emptyset$ );
- an **edge** ( $1$ -simplex) has  $1 + 1 = 2$  vertices and  $2$  facets;
- a **triangle** ( $2$ -simplex) has  $2 + 1 = 3$  vertices and  $3$  facets;
- a **tetrahedron** ( $3$ -simplex) has  $3 + 1 = 4$  vertices and  $4$  facets, etc.

A  **$d$ -cuboid** has conversely  $2^d$  **vertices** and  $2d$  **facets**:

- a **point** ( $0$ -cuboid) has  $2^0 = 1$  vertices and  $0$  facets;
- an **edge** ( $1$ -cuboid) has  $2^1 = 2$  vertices and  $2$  facets;
- a **quadrilateral** ( $2$ -cuboid) has  $2^2 = 4$  vertices and  $4$  facets;
- a **hexahedron** ( $3$ -cuboids) has  $2^3 = 8$  vertices and  $8$  facets, etc.

# Cellular complex: properties

- Support  $|K|$  of a cellular complex  $K$  is the union of points of its cells

# Cellular complex: properties

- Support  $|K|$  of a cellular complex  $K$  is the union of points of its cells
- A triangulation of a polytope  $P$  is a simplicial complex  $K$  whose support is  $|K| = P$

# Cellular complex: properties

- Support  $|K|$  of a cellular complex  $K$  is the union of points of its cells
- A triangulation of a polytope  $P$  is a simplicial complex  $K$  whose support is  $|K| = P$ 
  - For example, a triangulation of a polygon is a subdivision in triangles

# Cellular complex: properties

- Support  $|K|$  of a cellular complex  $K$  is the union of points of its cells
- A triangulation of a polytope  $P$  is a simplicial complex  $K$  whose support is  $|K| = P$ 
  - For example, a triangulation of a polygon is a subdivision in triangles
- Simplices and cuboids are polytopes.

# Cellular complex: properties

- Support  $|K|$  of a cellular complex  $K$  is the union of points of its cells
- A triangulation of a polytope  $P$  is a simplicial complex  $K$  whose support is  $|K| = P$ 
  - For example, a triangulation of a polygon is a subdivision in triangles
- Simplices and cuboids are polytopes.
- A polytope is always triangulable;

# Cellular complex: properties

- Support  $|K|$  of a cellular complex  $K$  is the union of points of its cells
- A triangulation of a polytope  $P$  is a simplicial complex  $K$  whose support is  $|K| = P$ 
  - For example, a triangulation of a polygon is a subdivision in triangles
- Simplices and cuboids are polytopes.
- A polytope is always triangulable;
  - For example, a quadrilateral by be divided in two triangles, and a cube in either 5 or 6 tetrahedra without adding new vertices

# Simplicial mapping

# Simplicial mapping: definition

## Definition

A **simplicial map** is a map **between simplicial complexes** with the property that the images of the vertices of a simplex always span a simplex.

## Remarks

Simplicial maps are **determined by their effects on vertices**  
for a precise definition of **Simplicial Map** look at [Wolfram MathWorld](#)

# MAP operator in plasm

## Map operator

`MAP(fun)(domain)`

## Semantics

- ① **domain** (HPC value) is decomposed into a **simplicial complex**

# MAP operator in plasm

## Map operator

`MAP(fun)(domain)`

## Semantics

- ① **domain** (HPC value) is decomposed into a **simplicial complex**
- ② **fun** (a simplicial function) is applied to the **domain vertices**

# MAP operator in plasm

## Map operator

`MAP(fun)(domain)`

## Semantics

- ① `domain` (HPC value) is decomposed into a `simplicial complex`
- ② `fun` (a simplicial function) is applied to the `domain` vertices
- ③ the `mapped domain` is returned

# MAP examples: 1-sphere ( $S^1$ ) and 2-disk ( $D^2$ )

```

def sphere1(p): return [COS(p[0]), SIN(p[0])] # point function
def domain(n): return INTERVALS(2*PI)(n)          # generator of domain decomp
VIEW( MAP(sphere1)(domain(32)) )                  # geometric value (HPC type)

def disk2D(p):                                     # point function
    u,v = p
    return [v*COS(u), v*SIN(u)]                   # coordinate functions
domain2D = PROD([INTERVALS(2*PI)(32), INTERVALS(1)(3)]) # 2D domain decompos
VIEW( MAP(disk2D)(domain2D) )
VIEW( SKELETON(1)(MAP(disk2D)(domain2D)) )

```

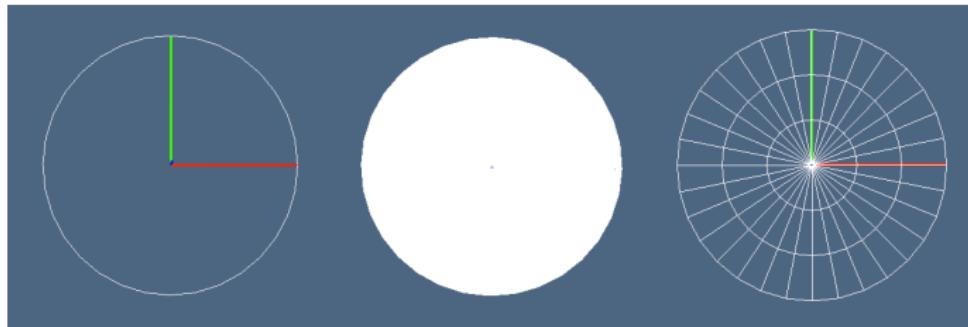


Figure 6: (a) sphere  $S^1$  (b) disk  $D^2$ ; (c) 1-skeleton.