Computational Graphics: Lecture 7

Alberto Paoluzzi

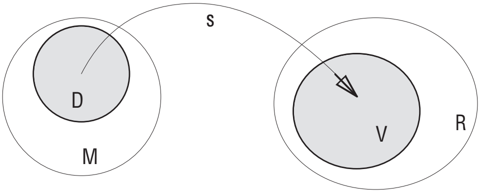
Thu, Mar 19, 2015

Outline: LAR1

Solid Modeling

Representation scheme: definition

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



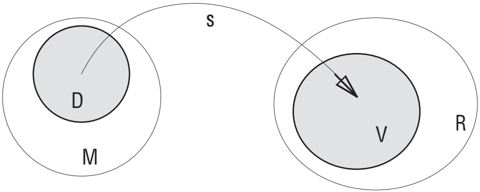
1. The M set contains the mathematical models of the class of solid objects the scheme aims to represent

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 $s : M \to R$ 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 . . .

Space decompositions

Join of pointsets

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

$$PQ = join(P, Q) := \{\gamma p + \lambda q,\ \ p \in P,\ \ q \in Q\}$$
$$\gamma, \lambda \in \mathbb{R},\ \ \gamma, \lambda \geq 0,\ \ \gamma + \lambda = 1$$

.

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)
```

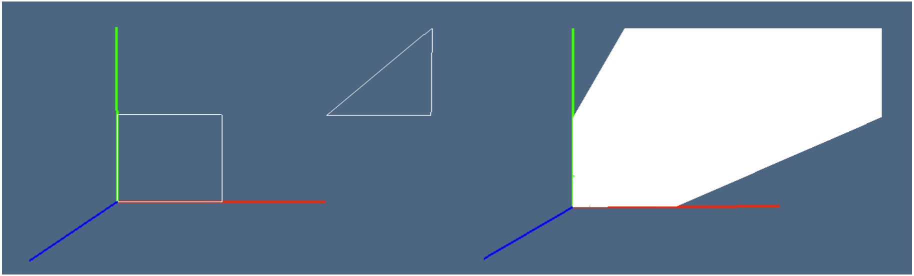


Figure: (a) 1-skeleton of pointsets in S; (b) convex hull H of pointset P

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 $p_0, \ldots, p_n$ are affinely independent when the n vectors $p_1 - p_0, \ldots, p_n - p_0$ 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)
```

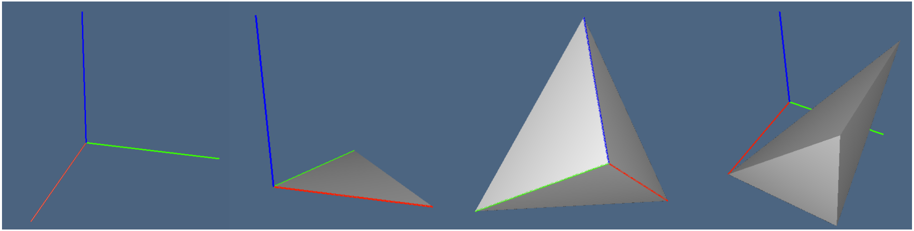


Figure: (a,b,c) 1-, 2-, and 3-standard simplex; (d) 3-simplex defined by 4 points

Simplicial complex

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

Simplicial complex

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

A simplicial complex is a set of simplices Σ, verifying the following conditions:

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

Simplicial complex

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

A simplicial complex is a set of simplices Σ, verifying the following conditions:

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

Simplicial complex

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

A simplicial complex is a set of simplices Σ, verifying the following conditions:

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

Simplicial complex

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

A simplicial complex is a set of simplices Σ, verifying the following conditions:

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

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

Simplicial complex: examples

```
from larcc import *                               # import LAR library
V,CV = larSimplexGrid([5,5,5])                    # structured simplicial grid
FV = larSimplexFacets(CV)                         # 2-simplicial grid
EV = larSimplexFacets(FV)                         # 1-simplicial grid
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)]        # boundary 2-simplices
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,BV))))
```

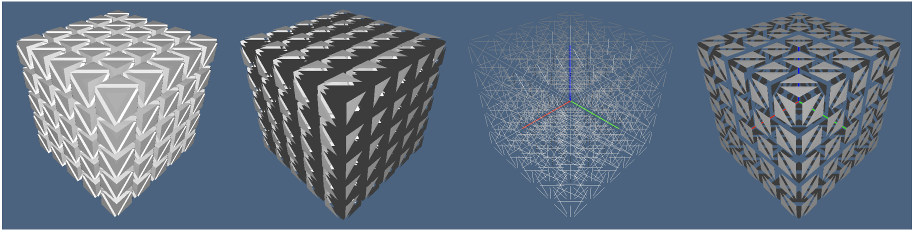


Figure: (a) 3-complex; (b) 2-subcomplex; (c) 1-subcomplex; (d) 2-boundary.

Simplicial complex: examples

(see Disk Point Picking)

```
from larcc 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)))))
```

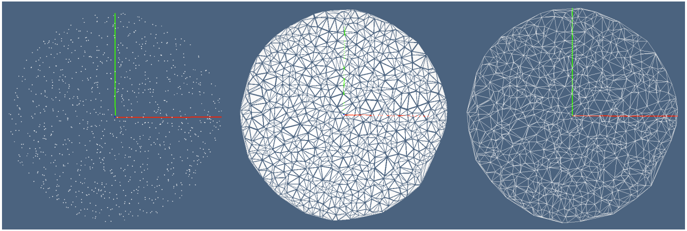


Figure: (a) Points; (b) Delaunay triangulation; (c) 1-skeleton.

Simplicial complex: examples

(see Disk Point Picking)

```
from larcc 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))))
```

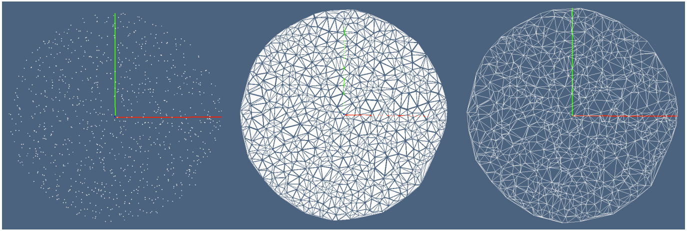


Figure: (a) Points; (b) Delaunay triangulation; (c) 1-skeleton

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)) )
```

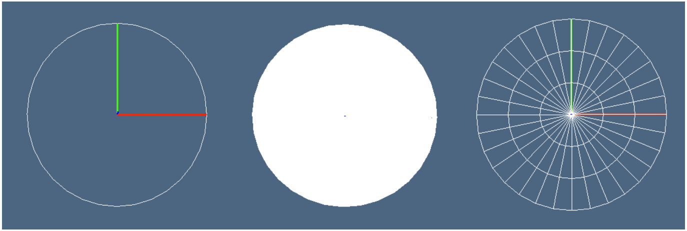


Figure: (a) sphere S^1 (b) disk D^2; (c) 1-skeleton.

LAR-CC library

download from github

```
$ git clone git@github.com:cvdlab/lar-cc.git
```

download from github

```
$ git clone git@github.com:cvdlab/lar-cc.git
```

In your python files:

```
import sys
""" import modules from lar-cc/lib """
sys.path.insert(0, 'lar-cc/lib/py/')
from simplexn import *
from larcc import *
from lar2psm import *
from largrid import *
```

LAR representation

Input of a simplicial complex (brc2csr)

From BRC (Binary Row Compressed) to CSR (Compressed Sparse Row)

- LAR model: (V,FV,EV)

```
V = [[0, 0], [1, 0], [2, 0], [0, 1], [1, 1], [2, 1]]
FV = [[0, 1, 3], [1, 2, 4], [1, 3, 4], [2, 4, 5]]
EV = [[0,1],[0,3],[1,2],[1,3],[1,4],[2,4],[2,5],[3,4],[4,5]]

VIEW(STRUCT(MKPOLS((V,FV)))); VIEW(EXPLODE(1.2,1.2,1)(MKPOLS((V,FV))))
VIEW(STRUCT(MKPOLS((V,EV)))); VIEW(EXPLODE(1.2,1.2,1)(MKPOLS((V,EV))))
```

```
csrFV = csrCreate(FV)
csrEV = csrCreate(EV)

print "\ncsrCreate(FV) =\n", csrFV
print "\n>>> csr2DenseMatrix"
print "\nFV =\n", csr2DenseMatrix(csrFV)
print "\nEV =\n", csr2DenseMatrix(csrEV)
```

Input of a simplicial complex (brc2csr)

From BRC (Binary Row Compressed) to CSR (Compressed Sparse Row)

- LAR model: (V,FV,EV)

```
V = [[0, 0], [1, 0], [2, 0], [0, 1], [1, 1], [2, 1]]
FV = [[0, 1, 3], [1, 2, 4], [1, 3, 4], [2, 4, 5]]
EV = [[0,1],[0,3],[1,2],[1,3],[1,4],[2,4],[2,5],[3,4],[4,5]]

VIEW(STRUCT(MKPOLS((V,FV)))); VIEW(EXPLODE(1.2,1.2,1)(MKPOLS((V,FV))))
VIEW(STRUCT(MKPOLS((V,EV)))); VIEW(EXPLODE(1.2,1.2,1)(MKPOLS((V,EV))))
```

- Lar representation: (CSR matrix)

```
csrFV = csrCreate(FV)
csrEV = csrCreate(EV)

print "\ncsrCreate(FV) =\n", csrFV
print "\n>>> csr2DenseMatrix"
print "\nFV =\n", csr2DenseMatrix(csrFV)
print "\nEV =\n", csr2DenseMatrix(csrEV)
```

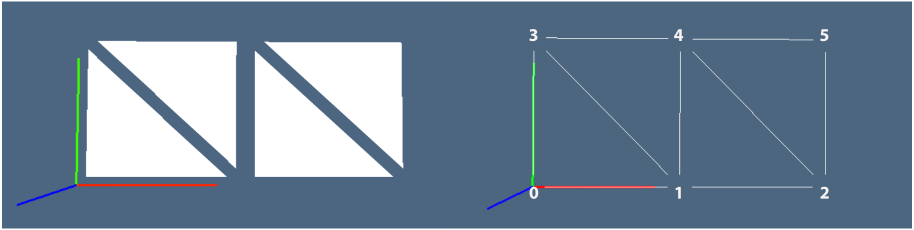
Input of a simplicial complex (brc2csr)

```
csrCreate(FV) =
  (0, 0)        1
  (0, 1)        1
  (0, 3)        1
  (1, 1)        1
  (1, 2)        1
  (1, 4)        1
  (2, 1)        1
  (2, 3)        1
  (2, 4)        1
  (3, 2)        1
  (3, 4)        1
  (3, 5)        1
```

```
>>> csr2DenseMatrix

FV =
[[1 1 0 1 0 0]
 [0 1 1 0 1 0]
 [0 1 0 1 1 0]
 [0 0 1 0 1 1]]

EV =
[[1 1 0 0 0 0]
 [1 0 0 1 0 0]
 [0 1 1 0 0 0]
 [0 1 0 1 0 0]
 [0 1 0 0 1 0]
 [0 0 1 0 1 0]
 [0 0 1 0 0 1]
 [0 0 0 1 1 0]
 [0 0 0 0 1 1]]
```



Facet extraction

Facet extraction from simplices

combinatorial approach

- A k-face of a d-simplex is defined as the convex hull of any subset of k vertices.

Facet extraction from simplices

combinatorial approach

- A k-face of a d-simplex is defined as the convex hull of any subset of k vertices.
- A $(d-1)$-face of a d-simplex

$$\sigma^d = \langle v_0, v_1, \dots, v_d \rangle$$

is also called a facet.

Facet extraction from simplices

combinatorial approach

- A k-face of a d-simplex is defined as the convex hull of any subset of k vertices.
- A $(d-1)$-face of a d-simplex

  $$\sigma^d = \langle v_0, v_1, \ldots, v_d \rangle$$

  is also called a facet.
- Each of the $d+1$ facets of σ^d, obtained by removing a vertex from σ^d, is a $(d-1)$-simplex.

Facet extraction from simplices

combinatorial approach

- A k-face of a d-simplex is defined as the convex hull of any subset of k vertices.
- A $(d-1)$-face of a d-simplex

$$\sigma^d = \langle v_0, v_1, \ldots, v_d \rangle$$

  is also called a facet.
- Each of the $d+1$ facets of σ^d, obtained by removing a vertex from σ^d, is a $(d-1)$-simplex.
- A simplex may be oriented in two different ways according to the permutation class of its vertices.

Facet extraction from simplices

combinatorial approach

- A k-face of a d-simplex is defined as the convex hull of any subset of k vertices.
- A $(d-1)$-face of a d-simplex

$$\sigma^d = \langle v_0, v_1, \ldots, v_d \rangle$$

  is also called a facet.
- Each of the $d+1$ facets of σ^d, obtained by removing a vertex from σ^d, is a $(d-1)$-simplex.
- A simplex may be oriented in two different ways according to the permutation class of its vertices.
- The simplex orientation is so changed by either multiplying the simplex by -1, or by executing an odd number of exchanges of its vertices.

Facet extraction from simplices

combinatorial approach

The chain of oriented boundary facets of σ^d, usually denoted as $\partial\sigma^d$, is generated combinatorially as follows:

$$\partial\,\sigma^d = \sum_{k=0}^{d} (-1)^d \langle v_0, \ldots, v_{k-1}, v_{k+1}, \ldots, v_d \rangle$$

Implementation

```python
def larSimplexFacets(simplices):
    ''' To return the facets of a list of d-simplices '''
    out = []
    d = len(simplices[0])
    for simplex in simplices:
        out += [simplex[0:k]+simplex[k+1:d]
                for k in range(d)]
    out = sorted(out)
    return [facet for k,facet in enumerate(out[:-1])
                if out[k] != out[k+1]] + [out[-1]]
```

Test of implementation

```
>>>larSimplexFacets([[0]])
[[]]
>>>larSimplexFacets([[0,1]])
[[0],[1]]
>>>larSimplexFacets([[0,1,2,]])
[[0,1],[0,2],[1,2]]
>>>larSimplexFacets([[0,1,2,3]])
[[0,1,2],[0,1,3],[0,2,3],[1,2,3]]
>>>larSimplexFacets([[0,1,2,3,4]])
[[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4],[1,2,3,4]]
```

Test of implementation

```
>>>larSimplexFacets([[0]])
[[]]
>>>larSimplexFacets([[0,1]])
[[0],[1]]
>>>larSimplexFacets([[0,1,2,]])
[[0,1],[0,2],[1,2]]
>>>larSimplexFacets([[0,1,2,3]])
[[0,1,2],[0,1,3],[0,2,3],[1,2,3]]
>>>larSimplexFacets([[0,1,2,3,4]])
[[0,1,2,3],[0,1,2,4],[0,1,3,4],[0,2,3,4],[1,2,3,4]]
```

are such facets oriented?

Examples of facet extraction from 3D simplicial cube

```
V,CV = larSimplexGrid([1,1,1])

VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,CV))))
SK2 = (V,larSimplexFacets(CV))
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(SK2)))
SK1 = (V,larSimplexFacets(SK2[1]))
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(SK1)))
```

look also at

```
V,CV = larSimplexGrid([5,5,2])
```

Assignment

Change the larSimplexFacets so that the extracted facets are coherently oriented

Boundary computation

From cells and facets to boundary operator

```
def boundary(cells,facets):
    csrCV = csrCreate(cells)
    csrFV = csrCreate(facets)
    csrFC = matrixProduct(csrFV, csrTranspose(csrCV))
    facetLengths = [csrCell.getnnz() for csrCell in csrCV]
    return csrBoundaryFilter(csrFC,facetLengths)

def coboundary(cells,facets):
    Boundary = boundary(cells,facets)
    return csrTranspose(Boundary)
```

Oriented boundary example

```
V,CV = larSimplexGrid([4,4,4])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,CV))))

FV = larSimplexFacets(CV)
EV = larSimplexFacets(FV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,FV))))

csrSignedBoundaryMat = signedBoundary (V,CV,FV)
boundaryCells_2 = signedBoundaryCells(V,CV,FV)
def swap(l): return [l[1],l[0],l[2]]
boundaryFV = [FV[-k] if k<0 else swap(FV[k]) for k in boundaryCells_2]
boundary = (V,boundaryFV)
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(boundary)))
```

Oriented boundary example

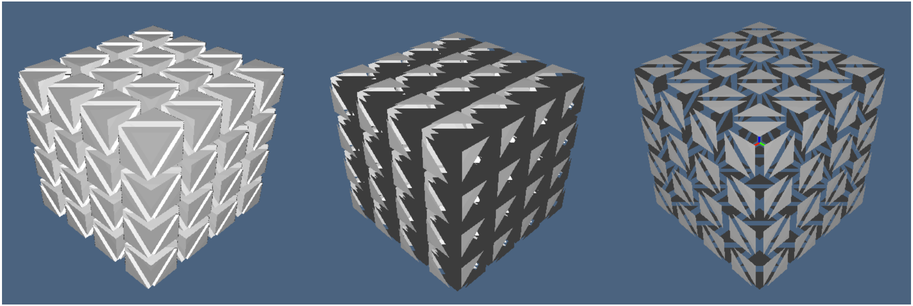


Figure: Simplicial complexes: (a) 3-complex S_3; (b) 2-complex $S_2 = K_2(S_3)$; (c) 2-complex $T_2 = \partial S_3 \subset S_2$)

Extrusion

Simplicial extrusion

Computation

Figure 1: Extrusion of (a) a point; (b) a straight line segment; (c) a triangle.

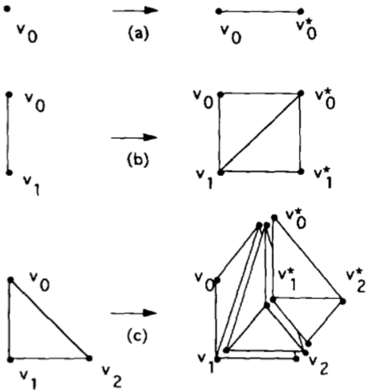


Figure: example caption

Simplicial extrusion

Computation

Let us concentrate on the generation of the simplex chain γ^{d+1} of dimension $d+1$ produced by combinatorial extrusion of a single simplex

$$\sigma^d = \langle v_0, v_1, \ldots, v_d, \rangle.$$

Then we have, with $|\gamma^{d+1}| = \sigma^d \times I$, and $I = [0,1]$:

$$\gamma^{d+1} = \sum_{k=0}^{d} (-1)^{kd} \langle v_k, \ldots v_d, v_0^*, \ldots v_k^* \rangle$$

with $v_k \in \sigma^d \times \{0\}$ and $v_k^* \in \sigma^d \times \{1\}$, and where the term $(-1)^{kd}$ is used to generate a chain of coherently-oriented extruded simplices.

Example of simplicial complex extrusion

```
V = [[0,0],[1,0],[2,0],[0,1],[1,1],[2,1],[0,2],[1,2],[2,2]]
FV = [[0,1,3],[1,2,4],[2,4,5],[3,4,6],[4,6,7],[5,7,8]]
model = larExtrude((V,FV),4*[1,2,-3])
VIEW(EXPLODE(1,1,1.2)(MKPOLS(model)))
```

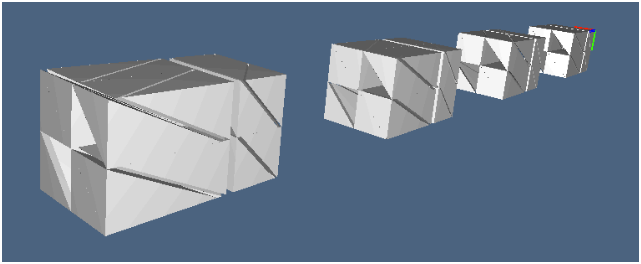


Figure: A simplicial complex providing a quite complex 3D assembly of tetrahedra.

Multidimensional simplicial grids

The generation of simplicial grids of any dimension and shape is amazingly simple

The input parameter `shape` is either a tuple or a list of integers used to specify the shape of the created array

```
VOID = V0,CV0 = [[]],[[0]]      # the empty simplicial model

def larSimplexGrid(shape):
    model = VOID
    for item in shape:
        model = larExtrude(model,item*[1])
    return model
```

The returned `model` has integer vertices, to be scaled and/or translated and/or mapped

Cartesian product of complexes

Cartesian product of two LAR models

```
def larModelProduct(twoModels):
    (V, cells1), (W, cells2) = twoModels
    @< Cartesian product of vertices @>
    @< Topological product of cells    @>
    model = [list(v) for v in vertices.keys()], cells
    return model
```

Cartesian product of two LAR models

```
def larModelProduct(twoModels):
    (V, cells1), (W, cells2) = twoModels
    @< Cartesian product of vertices @>
    @< Topological product of cells    @>
    model = [list(v) for v in vertices.keys()], cells
    return model
```

Cartesian product of vertices

```
vertices = collections.OrderedDict(); k = 0
for v in V:
    for w in W:
        id = tuple(v+w)
        if not vertices.has_key(id):
            vertices[id] = k
            k += 1  @}
```

Cartesian product of two LAR models

```
def larModelProduct(twoModels):
    (V, cells1), (W, cells2) = twoModels
    @< Cartesian product of vertices @>
    @< Topological product of cells    @>
    model = [list(v) for v in vertices.keys()], cells
    return model
```

Cartesian product of vertices

```
vertices = collections.OrderedDict(); k = 0
for v in V:
    for w in W:
        id = tuple(v+w)
        if not vertices.has_key(id):
            vertices[id] = k
            k += 1  @}
```

Topological product of cells

```
cells = [ [vertices[tuple(V[v] + W[w])] for v in c1 for w in c2]
         for c1 in cells1 for c2 in cells2] @}
```

Cuboidal grids

```
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(larCuboids([3,2,1],True))))
```

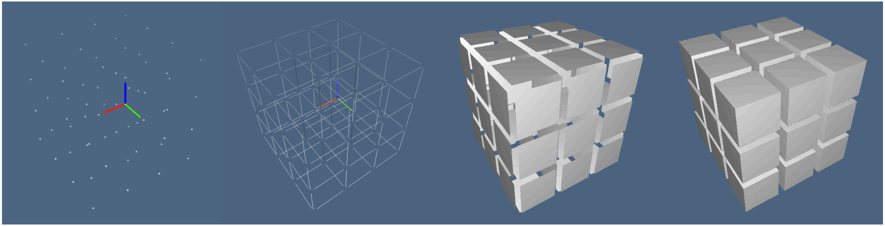


Figure: Exploded views of 0-, 1-, 2-, and 3-dimensional skeletons.

Cuboidal grids

```
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(larCuboids([3,2,1],True))))
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(larCuboids([3,2,1],False))))
```

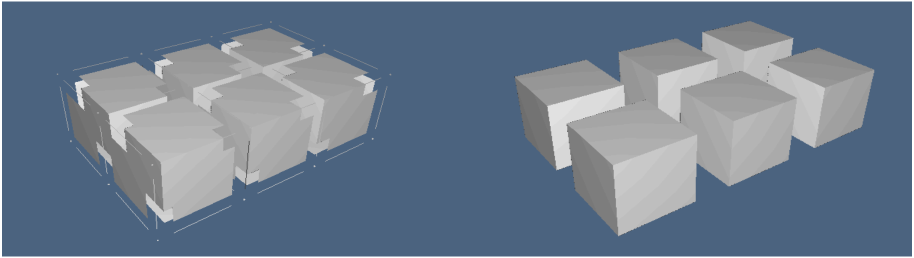


Figure: Exploded views of 0-, 1-, 2-, and 3-dimensional skeletons.

Skeletons

Cuboidal skeletons

A list of BRC characteristic matrices of cellular k-complexes ($0 \leq k \leq d$) with dimension d, where $d =$ `len(shape)`, is returned by the function `gridSkeletons` in the macro below, where the input is given by the shape of the grid, i.e. by the list of cell items in each coordinate direction.

```
def gridSkeletons(shape):
    gridMap = larGridSkeleton(shape)
    skeletonIds = range(len(shape)+1)
    skeletons = [ gridMap(id) for id in skeletonIds ]
    return skeletons
```

Cuboidal skeletons

Just notice that the number of returned *d*-cells is equal to `PROD(shape)`

```
print "\ngridSkeletons([3]) =\n", gridSkeletons([3])
print "\ngridSkeletons([3,2]) =\n", gridSkeletons([3,2])
print "\ngridSkeletons([3,2,1]) =\n", gridSkeletons([3,2,1])
```

Generation of grid boundary complex

```
def gridBoundaryMatrices(shape):
    skeletons = gridSkeletons(shape)
    boundaryMatrices = [boundary(skeletons[k+1],faces)
                        for k,faces in enumerate(skeletons[:-
    return boundaryMatrices

for k in range(1):
    print "\ngridBoundaryMatrices([3]) =\n", \
          csr2DenseMatrix(gridBoundaryMatrices([3])[k])
for k in range(2):
    print "\ngridBoundaryMatrices([3,2]) =\n", \
          csr2DenseMatrix(gridBoundaryMatrices([3,2])[k])
for k in range(3):
    print "\ngridBoundaryMatrices([3,2,1]) =\n", \
          csr2DenseMatrix(gridBoundaryMatrices([3,2,1])[k])
```

References

References

A. DiCarlo, V. Shapiro, and A. Paoluzzi, Linear Algebraic Representation for Topological Structures, Computer-Aided Design, Volume 46, Issue 1 , January 2014, Pages 269-274 (doi:10.1016/j.cad.2013.08.044)