Computational Graphics: Lecture 12

Alberto Paoluzzi

Mon, Mar 30, 2015

Outline: LAR2

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:

```python
import sys
""" import modules from lar-cc/lib """
sys.path.insert(0, '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)
```

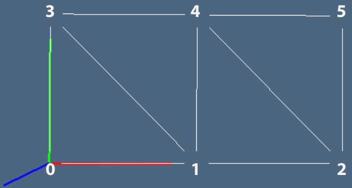
Input of a simplicial complex (brc2csr)

```
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

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

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 = larSimplexGrid1([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 = larSimplexGrid1([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 = larSimplexGrid1([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

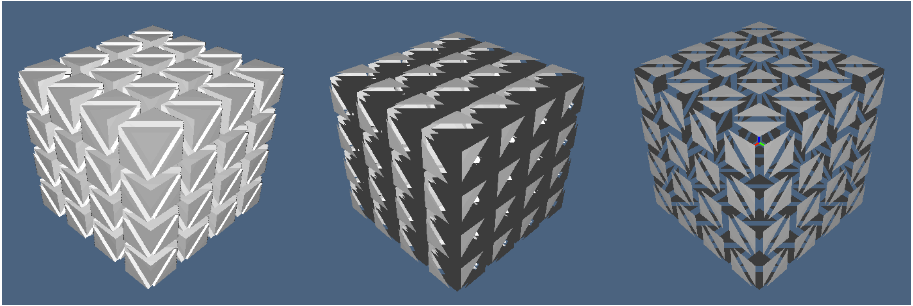


Figure: Simplicial complexes: (a) 3-complex S_3; (b) 2-complex $S_2 = K_2(S_3)$; (c) 2-complex $T_2 = \partial S_3 \subset S_2$)

Extrusion

Simplicial extrusion

Computation

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

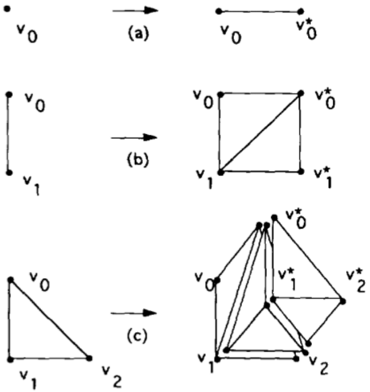


Figure: example caption

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 = larExtrude1((V,FV),4*[1,2,-3])
VIEW(EXPLODE(1,1,1.2)(MKPOLS(model)))
```

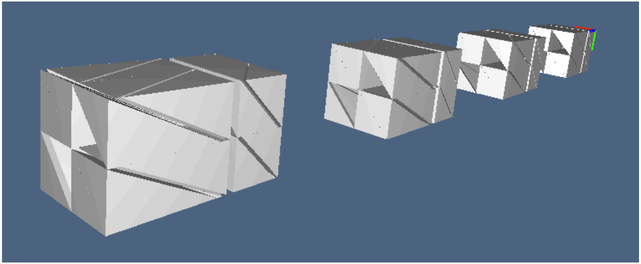


Figure: A simplicial complex providing a quite complex 3D assembly of tetrahedra.

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 larSimplexGrid1(shape):
    model = VOID
    for item in shape:
        model = larExtrude1(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]))))
```

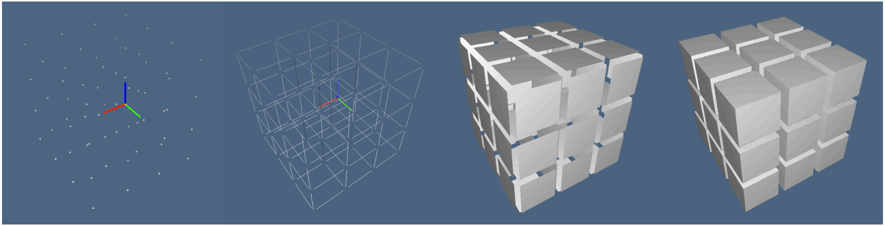


Figure: Exploded views of 0-, 1-, 2-, and 3-dimensional skeletons.

Cuboidal grids

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

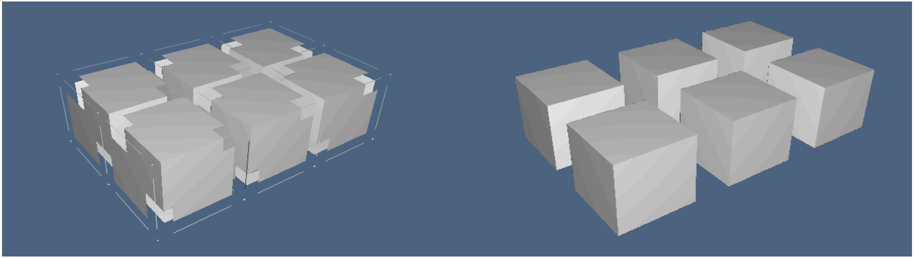


Figure: Exploded views of 0-, 1-, 2-, and 3-dimensional skeletons.

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)