The `smplxn` module *

Alberto Paoluzzi

April 24, 2014

Abstract

This module defines a minimal set of functions to generate a dimension-independent grid of simplices. The name of the library was firstly used by our CAD Lab at University of Rome "La Sapienza" in years 1987/88 when we started working with dimension-independent simplicial complexes [PBCF93]. This one in turn imports some functions from the `scipy` package and the geometric library `pyplasm` [].

Contents

*This document is part of the framework [CL13]. April 24, 2014

1 Introduction

The $Simple^n_X$ library, named `simplexn` within the Python version of the LARCC framework, provides combinatorial algorithms for some basic functions of geometric modelling with simplicial complexes. In particular, provides the efficient creation of simplicial complexes generated by simplicial complexes of lower dimension, the production of simplicial grids of any dimension, and the extraction of facets (i.e. of $(d-1)$-faces) of complexes of d-simplices.

2 Some simplicial algorithms

The main aim of the simplicial functions given in this library is to provide optimal combinatorial algorithms, whose time complexity is linear in the size of the output. Such a goal is achieved by calculating each cell in the output via closed combinatorial formulas, that do not require any searching nor data structure traversal to produce their results.

2.1 Linear extrusion of a complex

Here we discuss an implementation of the linear extrusion of simplicial complexes according to the method discussed in [PBCF93] and [FP91]. In synthesis, for each d-simplex in the input complex, we generate combinatorially a $(d+1)$-simplicial *tube*, i.e. a chain of $d+1$ simplexes of dimension $d+1$. It can be shown that if the input simplices are a simplicial complex, then the output simplices are a complex too.

In other words, if the input is a complex, where all d-cells either intersect along a common face or are pairwise disjoints, then the output is also a simplicial complex of dimension $d+1$. This method is computationally optimal, since it does not require any search or traversal of data structures. The algorithm [FP91] just writes the output making a constant number $O(1)$ of operation for each one of its n output d-cells, so that the time complexity is $\Omega(n)$, where $n = d\,m$, being m the number and d the dimension (and the storage size) of the input cells, represented as lists of indices of vertices.

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 $I = [0,1]$:

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

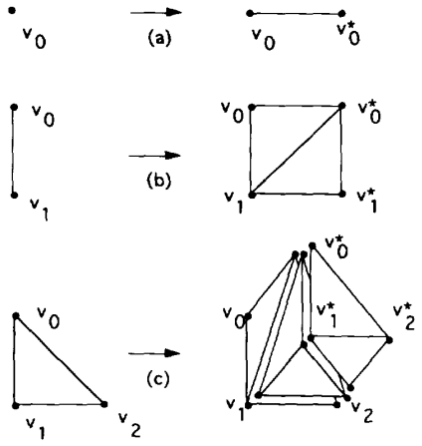

Figure 1: Extrusion of (a) a point; (b) a straight line segment; (c) a triangle.

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.

In our implementation the combinatorial algorithm above is twofold generalised:

1. by applying it to all d-simplices of a LAR model of dimension d;
2. by using instead of the single interval $I = [0, 1]$, the possibly unconnected set of 1D intervals generated by the list of integer numbers stored in the `pattern` variable

Implementation In the macro below, `larExtrude1` is the function to generate the output model vertices in a multiple extrusion of a LAR model.

First we notice that the `model` variable contains a pair (`V`, `FV`), where `V` is the array of input vertices, and `FV` is the array of d-cells (given as lists of vertex indices) providing the input representation of a LAR cellular complex.

The `pattern` variable is a list of integers, whose absolute values provide the sizes of the ordered set of 1D (in local coords) subintervals specified by the `pattern` itself. Such subintervals are assembled in global coordinates, and each one of them is considered either solid or void depending on the sign of the corresponding integer, which may be either positive (solid subinterval) or negative (void subinterval).

Therefore, a value `pattern = [1,1,-1,1]` must be interpreted as the 1D simplicial complex

$$[0, 1] \cup [1, 2] \cup [3, 4]$$

with five vertices `W = [[0.0], [1.0], [2.0], [3.0], [4.0]]` and three 1-cells `[[0,1], [1,2], [3,4]]`.

`V` is the list of input d-vertices (each given as a list of d coordinates); `coords` is a list of absolute translation parameters to be applied to `V` in order to generate the output vertices generated by the combinatorial extrusion algorithm.

The `cellGroups` variable is used to select the groups of $(d+1)$-simplices corresponding to solid intervals in the input `pattern`, and `CAT` provides to flatten their set, by removing a level of square brackets.

⟨ Simplicial model extrusion in accord with a 1D pattern 4a ⟩ ≡

```
def larExtrude1(model,pattern):
    V, FV = model
    d, m = len(FV[0]), len(pattern)
    coords = list(cumsum([0]+(AA(ABS)(pattern))))
    offset, outcells, rangelimit = len(V), [], d*m
    for cell in FV:
        ⟨ Append a chain of extruded cells to outcells 4b ⟩
    outcells = AA(CAT)(TRANS(outcells))
    cellGroups = [group for k,group in enumerate(outcells) if pattern[k]>0 ]
    outVertices = [v+[z] for z in coords for v in V]
    outModel = outVertices, CAT(cellGroups)
    return outModel
```

◇

Macro referenced in 9b.

Extrusion of single cells For each cell in `FV` a chain of vertices is created, then they are separated into groups of $d+1$ consecutive elements, by shifting one position at a time.

⟨ Append a chain of extruded cells to outcells 4b ⟩ ≡

⟨ Create the indices of vertices in the cell "tube" 4c ⟩
⟨ Take groups of d+1 elements, by shifting one position 4d ⟩
◇

Macro referenced in 4a.

Assembling vertex indices in a tube with their shifted images Here the "long" chain of vertices is created.

⟨ Create the indices of vertices in the cell "tube" 4c ⟩ ≡

```
tube = [v + k*offset for k in range(m+1) for v in cell]
```

◇

Macro referenced in 4b.

Selecting and reshaping extruded cells in a tube Here the chain of vertices is spitted into subchains, and such subchains are reshaped into three-dimensional arrays of indices.

⟨ Take groups of d+1 elements, by shifting one position 4d ⟩ ≡

```
cellTube = [tube[k:k+d+1] for k in range(rangelimit)]
outcells += [reshape(cellTube, newshape=(m,d,d+1)).tolist()]
```

◇

Macro referenced in 4b.

Definition 1 (Big-Omega order)**.** *We say that a function $f(n)$ is* Big-Omega *order of a function $f(n)$, and write $f(n) \in \Omega(g(n))$ when a constant c exists, such that:*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0, \qquad \textit{where } 0 < c \leq \infty.$$

Theorem 1 (Optimality)**.** *The combinatorial algorithm for extrusion of simplicial complexes has time complexity $\Omega(n)$.*

Proof. Of course, if we denote as $g(n) = nd$ the time needed to write the input of the extrusion algorithm, proportional to the constant length d of cells, and as $f(m) = m(d+1)$ the time needed to write the output, where $m = n(d+1)$, we have

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{m(d+1)}{nd} = \lim_{n\to\infty} \frac{[n(d+1)](d+1)}{nd} = \frac{(d+1)^2}{d} = c > 0$$

□

2.1.1 Examples of simplicial complex extrusions

Example 1 It is interesting to notice that the 2D model extruded in example 1 below and shown in Figure 2 is locally non-manifold, and that several instance of the pattern in the z direction are obtained by just inserting a void subinterval (negative size) in the `pattern` value.

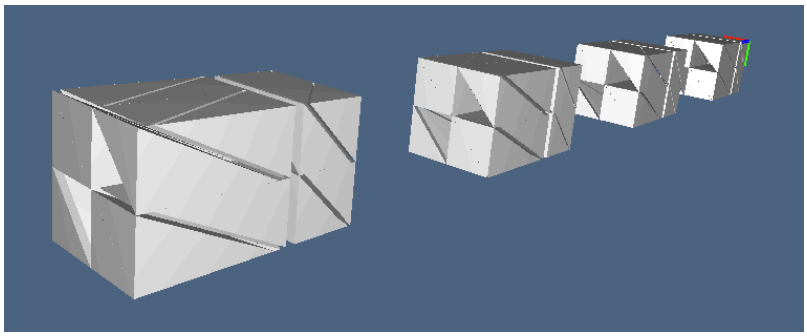

Figure 2: A simplicial complex providing a quite complex 3D assembly of tetrahedra.

Examples 2 and 3 The examples show that the implemented `larExtrude1` algorithm is fully multidimensional. It may be worth noting the initial definition of the empty `model`, as a pair having the empty list as vertex set and the list `[[0]]` as the cell list. Such initial value is used to define a predefinite constant `VOID`.

⟨ Examples of simplicial complex extrusions 5 ⟩ ≡

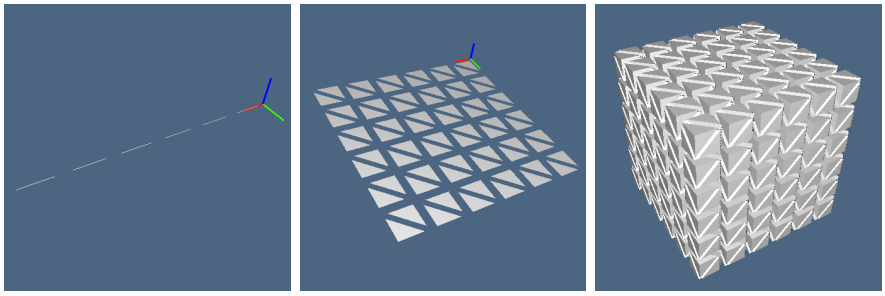

Figure 3: 1-, 2-, and 3-dimensional simplicial complex generated by repeated extrusion with the same pattern.

```
# example 1
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)))

# example 2
model = larExtrude1( VOID, 6*[1] )
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(model)))
model = larExtrude1( model, 6*[1] )
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(model)))
model = larExtrude1( model, 6*[1] )
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(model)))

# example 3
model = larExtrude1( VOID, 10*[1,-1] )
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(model)))
model = larExtrude1( model, 10*[1] )
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(model)))
◇
```

Macro referenced in 9b.

2.2 Generation of multidimensional simplicial grids

The generation of simplicial grids of any dimension and shape using the `larSimplexGrid1` 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, i.e. both the number of its dimensions (given by `len(shape)`) and the `size` of each dimension k (given by the `shape[k]` element). The implementation starts from the LAR model of the VOID simplicial complex (denoted as `VOID`, a predefined constant) and updates the `model` variable extruding it iteratively

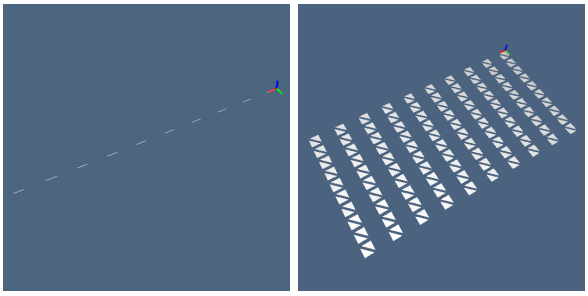

Figure 4: 1- and 2-dimensional simplicial complexes generated by different patterns.

according to the specs given by `shape`. Just notice that the returned grid `model` has vertices with integer coordinates, that can be subsequently scaled and/or translated and/or mapped in any other way, according to the user needs.

⟨ Generation of simplicial grids 7 ⟩ ≡

```
def larSimplexGrid1(shape):
    model = VOID
    for item in shape:
        model = larExtrude1(model,item*[1])
    return model
```

◇

Macro referenced in 9b.

Examples of simplicial grids The two examples of simplicial grids generated by the macro below with `shape` equal to `[3,3]` and `[2,3,4]`, respectively, are displayed in Figure 5.

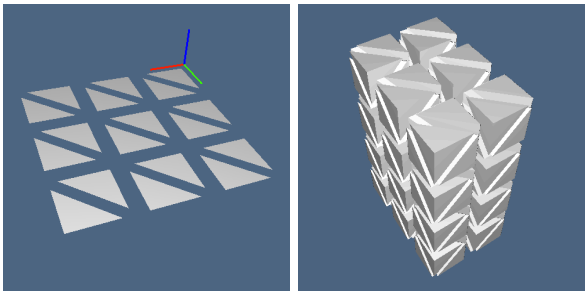

Figure 5: 2- and 3-dimensional simplicial grids.

⟨ Examples of simplicial grids 8a ⟩ ≡

```
grid_2d = larSimplexGrid1([3,3])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(grid_2d)))

grid_3d = larSimplexGrid1([2,3,4])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS(grid_3d)))
```

◇

Macro referenced in 9b.

2.3 Facet extraction from simplices

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.

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 The `larSimplexFacets` function, for estraction of non-oriented $(d-1)$-facets of d-dimensional simplices, returns a list of d-tuples of integers, i.e. the input LAR representation of the topology of a cellular complex. The final *list comprehension* is used to remove the duplicated facets, by taking only the last element of any subsequence with possibly duplicated elements.

⟨ Facets extraction from a set of simplices 8b ⟩ ≡

```
def larSimplexFacets(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]]
```

◇

Macro referenced in 9b.

Examples of facet extraction The simple generation of the LAR model of a simplicial decomposition of a 3D cube as a `larSimplexGrid1` with `shape = [1,1,1]` and of its 2D and 1D skeletons is shown here.

⟨ Examples of facet extraction from 3D simplicial cube 9a ⟩ ≡

```
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)))
```

◇

Macro referenced in 9b.

2.4 Exporting the $Simple_x^n$ library

The current version of the `simplexn` library is exported here. Next versions will take care of the OpenCL acceleration and data partitioning with very-large size simplicial grids and their sets of faces.

"lib/py/simplexn.py" 9b ≡

```
# -*- coding: utf-8 -*-
"""Module for facet extraction, extrusion and simplicial grids"""
from pyplasm import *
from lar2psm import *
from scipy import *

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

⟨ Cumulative sum 12a ⟩
⟨ Simplicial model extrusion in accord with a 1D pattern 4a ⟩
⟨ Generation of simplicial grids 7 ⟩
⟨ Facets extraction from a set of simplices 8b ⟩

```
if __name__ == "__main__":
```

⟨ Examples of simplicial complex extrusions 5 ⟩
⟨ Examples of simplicial grids 8a ⟩
⟨ Examples of facet extraction from 3D simplicial cube 9a ⟩

◇

3 Signed (co)boundary matrices of a simplicial complex

4 Test examples

4.1 Structured grid

4.1.1 2D example

Generate a simplicial decomposition Then we generate and show a 2D decomposition of the unit square $[0,1]^2 \subset \mathbb{E}^2$ into a 3×3 grid of simplices (triangles, in this case), using the `larSimplexGrid1` function, that returns a pair `(V,FV)`, made by the array `V` of vertices, and by the array `FV` of "faces by vertex" indices, that constitute a *reduced* simplicial LAR of the $[0,1]^2$ domain. The computed `FV` array is then dispayed "exploded", being ex, ey, ez the explosion parameters in the x, y, z coordinate directions, respectively. Notice that the `MKPOLS` pyplasm primitive requires a pair `(V,FV)`, that we call a "model", as input — i.e. a pair made by the array `V` of vertices, and by a zero-based array of array of indices of vertices. Elsewhere in this document we identified such a data structure as $\mathrm{CSR}(M_d)$, for some dimension d. Suc notation stands for the Compressed Sparse Row representation of a binary characteristic matrix.

⟨ Generate a simplicial decomposition ot the $[0,1]^2$ domain 10a ⟩ ≡

```
V,FV = larSimplexGrid1([3,3])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,FV))))
```

◇

Macro referenced in 10c.

Extract the $(d-1)$-faces Since the complex is simplicial, we can directly extract its facets (in this case the 1-faces, i.e. its edges) by invoking the `larSimplexFacets` function on the argument `FV`, so returning the array `EV` of "edges by vertex" indices.

⟨ Extract the edges of the 2D decomposition 10b ⟩ ≡

```
EV = larSimplexFacets(FV)
ex,ey,ez = 1.5,1.5,1.5
VIEW(EXPLODE(ex,ey,ez)(MKPOLS((V,EV))))
```

◇

Macro referenced in 10c.

Export the executable file We are finally able to generate and output a complete test file, including the visualization expressions. This file can be executed by the `test` target of the `make` command.

"test/py/simplexn/test01.py" 10c ≡

⟨ Inport the $Simple_X^n$ library 12b ⟩

⟨ Generate a simplicial decomposition ot the $[0,1]^2$ domain 10a ⟩
⟨ Extract the edges of the 2D decomposition 10b ⟩
◇

4.1.2 3D example

In this case we produce a $2 \times 2 \times 2$ grid of tetrahedra. The dimension (3D) of the model to be generated is inferred by the presence of 3 parameters in the parameter list of the `larSimplexGrid1` function.

⟨ Generate a simplicial decomposition ot the $[0,1]^3$ domain 11a ⟩ ≡

```
V,CV = larSimplexGrid1([2,2,2])
VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,CV))))
```

◇

Macro referenced in 11c.

and repeat two times the facet extraction:

⟨ Extract the faces and edges of the 3D decomposition 11b ⟩ ≡

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

◇

Macro referenced in 11c.

and finally export a new test file:

"test/py/simplexn/test02.py" 11c ≡

⟨ Inport the $Simple_X^n$ library 12b ⟩
⟨ Generate a simplicial decomposition ot the $[0,1]^3$ domain 11a ⟩
⟨ Extract the faces and edges of the 3D decomposition 11b ⟩
◇

4.2 Unstructured grid

4.2.1 2D example

4.2.2 3D example

A Utilities

⟨ Cumulative sum 12a ⟩ ≡

```
def cumsum(iterable):
    # cumulative addition: list(cumsum(range(4))) => [0, 1, 3, 6]
    iterable = iter(iterable)
    s = iterable.next()
    yield s
    for c in iterable:
        s = s + c
        yield s
```

◇

Macro referenced in 9b.

⟨ Inport the $Simple^n_X$ library 12b ⟩ ≡

```
""" import modules from larcc/lib """
import sys
from scipy import reshape
sys.path.insert(0, 'lib/py/')
from pyplasm import *
from lar2psm import *
from simplexn import *
```

◇

Macro referenced in 10c, 11c.

References

[CL13] CVD-Lab, *Linear algebraic representation*, Tech. Report 13-00, Roma Tre University, October 2013.

[FP91] Vincenzo Ferrucci and Alberto Paoluzzi, *Extrusion and boundary evaluation for multidimensional polyhedra*, Computer-Aided Design **23** (1991), no. 1, 40–50.

[PBCF93] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci, *Dimension-independent modeling with simplicial complexes*, ACM Trans. Graph. **12** (1993), no. 1, 56–102.