Computational Graphics: Lecture 4

Alberto Paoluzzi

October 16, 2017

Outline: larlib

Literate programming

ACM Turing Award winner 1974

DONALD ("DON") ERVIN KNUTH

ACM Turing Award winner 1974

For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.

ACM Turing Award winner 1974

DONALD ("DON") ERVIN KNUTH

ACM Turing Award winner 1974

For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.

http://www-cs-faculty.stanford.edu/~uno/

ACM Turing Award winner 1974

DONALD ("DON") ERVIN KNUTH

ACM Turing Award winner 1974

For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.

http://www-cs-faculty.stanford.edu/~uno/

Quotes from Donald E. Knuth . . . ☺

ACM Turing Award winner 1974

DONALD ("DON") ERVIN KNUTH

ACM Turing Award winner 1974

For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.

http://www-cs-faculty.stanford.edu/~uno/

Quotes from Donald E. Knuth . . . ☺

- *Beware of bugs in the above code; I have only proved it correct, not tried it*

ACM Turing Award winner 1974

DONALD ("DON") ERVIN KNUTH

ACM Turing Award winner 1974

For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.

http://www-cs-faculty.stanford.edu/~uno/

Quotes from Donald E. Knuth . . . ☺

- *Beware of bugs in the above code; I have only proved it correct, not tried it*

ACM Turing Award winner 1974

DONALD ("DON") ERVIN KNUTH

ACM Turing Award winner 1974

For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to the "art of computer programming" through his well-known books in a continuous series by this title.

http://www-cs-faculty.stanford.edu/~uno/

Quotes from Donald E. Knuth . . . ☺

- *Beware of bugs in the above code; I have only proved it correct, not tried it*
- *If you optimize everything, you will always be unhappy.*

Literate programming

Donald Knuth. "Literate Programming (1984)" in Literate Programming. CSLI, 1992, pg. 99

I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: “Literate Programming.”

Literate programming

Donald Knuth. "Literate Programming (1984)" in Literate Programming. CSLI, 1992, pg. 99

I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: "Literate Programming."

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

http://www.literateprogramming.com

Introduction to literate LarLib

Literate programming IDE for LAR-CC

Introduction to literate LarLib

Introduction to literate `LarLib`

Literate programming IDE for LAR-CC

Template for your project

A generally useful tool:

Pandoc a universal document converter

If you need to convert files from one markup format into another, pandoc is your swiss-army knife.

Start using "LarLib" with simplicial complexes

The `simplexn` module

- This module defines a minimal set of functions to generate a dimension-independent grid of simplices.

The `simplexn` module

- 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].

Introduction

The *Simple*$_X^n$ library, named `simplexn` within the Python version of the LARCC framework, provides combinatorial algorithms for some basic functions of geometric modelling with simplicial complexes.

Introduction

The $\textit{Simple}_X^n$ 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.

Aim of module

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.

Aim of module

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.

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.

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-simplices, returns a list of d-tuples of integers, i.e.∼the LAR representation of the topology of a cellular complex.

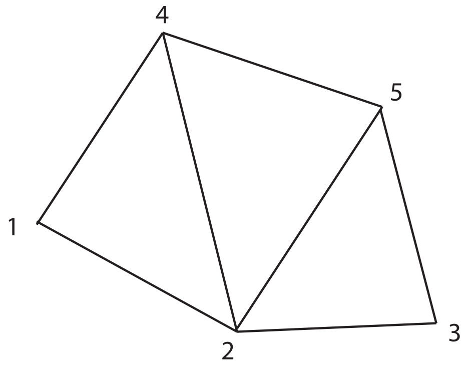
Implementation

The larSimplexFacets function, for estraction of non-oriented $(d-1)$-facets of d-simplices, returns a list of d-tuples of integers, i.e.~the LAR representation of the topology of a cellular complex.

The final steps are used to remove the duplicates (facets), by transforming the sorted facets into a set of tuples, so removing the duplicated elements.

```
def larSimplexFacets(simplices):
    out = []
    d = len(simplices[0])
    for simplex in simplices:
        out += AA(sorted)(
            [simplex[0:k]+simplex[k+1:d] for k in range(d)])
    out = set(AA(tuple)(out))
    return  sorted(out)
```

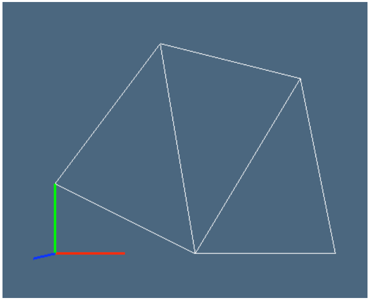
Example 1/2



```
In [13]: simplices = [[1,2,4],[2,4,5],[2,3,5]]

In [14]: larSimplexFacets(simplices)
Out[14]: [(1, 2), (1, 4), (2, 3), (2, 4), (2, 5), (3, 5), (4, 5)]
```

Example 2/2

```
V = [[0,0], [0,1],[2,0],[4,0],[1.5,3],[3.5,2.5]]
FV = simplices
model = V,FV

VIEW(STRUCT(MKPOLS(model)))
VIEW(SKEL_1(STRUCT(MKPOLS(model))))

EV = larSimplexFacets(FV)
VIEW(STRUCT(MKPOLS((V,EV))))
```



Linear extrusion of a complex

- It follows an implementation of the linear extrusion of simplicial complexes according to the method discussed in (A. Paoluzzi et al. 1993) and (V. Ferrucci and Paoluzzi 1991).

Linear extrusion of a complex

- It follows an implementation of the linear extrusion of simplicial complexes according to the method discussed in (A. Paoluzzi et al. 1993) and (V. Ferrucci and Paoluzzi 1991).
- 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 $d+1$.

Linear extrusion of a complex

- It follows an implementation of the linear extrusion of simplicial complexes according to the method discussed in (A. Paoluzzi et al. 1993) and (V. Ferrucci and Paoluzzi 1991).
- 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.

Linear extrusion of a complex

- It follows an implementation of the linear extrusion of simplicial complexes according to the method discussed in (A. Paoluzzi et al. 1993) and (V. Ferrucci and Paoluzzi 1991).
- 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 $d+1$.

Linear extrusion of a complex

- It follows an implementation of the linear extrusion of simplicial complexes according to the method discussed in (A. Paoluzzi et al. 1993) and (V. Ferrucci and Paoluzzi 1991).
- 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.

Algorithm 1/2

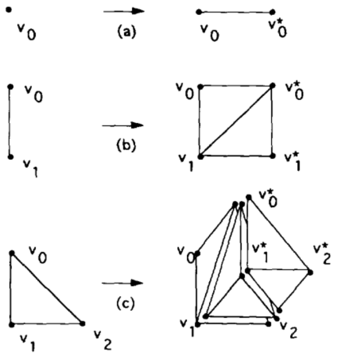


Figure 1: Extrusion of (a) a point; (b) a straight line segment; (c) a triangle.

Algorithm 2/2

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.

Implementation 1/3

`larExtrude1` is the function to generate the output model vertices in a multiple extrusion of a LAR model.

Implementation 1/3

`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.

Implementation 1/3

`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).

Implementation 1/3

`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]]`.

Implementation 2/3

`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.

Implementation 2/3

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.

Implementation 3/3

```python
def larExtrude1(model,pattern):
    """ Simplicial model extrusion in accord with a 1D 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:
        tube = [v + k*offset for k in range(m+1) for v in cell]
        cellTube = [tube[k:k+d+1] for k in range(rangelimit)]
        outcells += [reshape(cellTube, newshape=(m,d,d+1)).tolist()]
    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
```

Examples

```
model = [[0,0],[1,0],[0,1]], [[0,1,2]]
pattern = [1]

larExtrude1(model,pattern)

VIEW(STRUCT(MKPOLS((larExtrude1(model,pattern)))))
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS((larExtrude1(model,pattern)))))

pattern = [1]*10
VIEW(STRUCT(MKPOLS((larExtrude1(model,pattern)))))
```

Examples

The examples show that the implemented `larExtrude1` algorithm is fully multidimensional.

Note the initial definition of the empty `model`, as a pair with the empty list as vertex set, and the list `[[0]]` as cell list.

Such initial value is used to define a predefinite constant `VOID`.

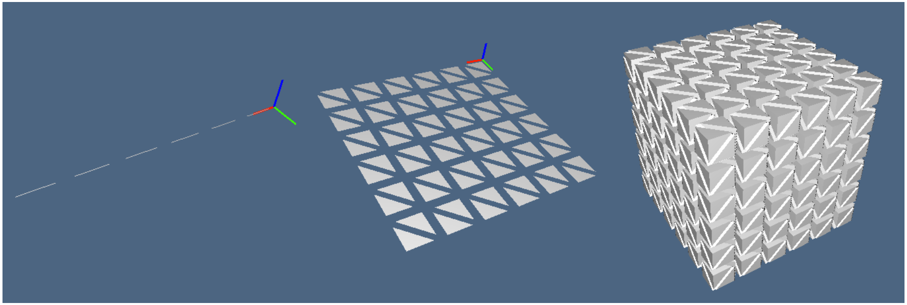


Figure 2: 1-, 2-, and 3-dimensional simplicial complex generated by repeated extrusion with the same pattern.

Examples

```
# 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, 10*[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)))
model = larExtrude1( model, 10*[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)))
```

References

References

Ferrucci, Vincenzo, and Alberto Paoluzzi. 1991. "Extrusion and Boundary Evaluation for Multidimensional Polyhedra." Computer-Aided Design 23 (1): 40–50.

Paoluzzi, A., F. Bernardini, C. Cattani, and V. Ferrucci. 1993. "Dimension-Independent Modeling with Simplicial Complexes." ACM Trans. Graph. 12 (1). New York, NY, USA: Acm: 56–102. doi:10.1145/169728.169719.