Parallel & Distributed Computing: Lecture 9

Alberto Paoluzzi

October 24, 2017

1. Student Programming Projects 2017-18

2. Work Breakdown and Assignments

Student Programming Projects 2017-18

Novel Project: Larlib.jl

Geometric Computing with Chain Complexes

Design and Features of a Julia Package

Geometric computing with chain complexes allows for the computation of the whole chain of linear spaces and (co)boundary operators generated by a space decomposition into a cell complex.

The space decomposition is stored and handled with LAR (Linear Algebraic Representation), i.e. with sparse integer arrays, and allows for using cells of a very general type, even non convex and with internal holes.

In this paper we discuss the features and the merits of this approach, and describe the goals and the implementation of a so ware package aiming at providing for simple and efficient computational support of geometric computing with any kind of meshes, using linear algebra tools with sparse arrays.

A library is being written in Julia, the novel efficient and parallel language for scientific computing. This software, that is being ported on hybrid architectures (CPU+GPU) of last generation, is under development.

Work Breakdown and Assignments

Module List

Larlib.py – subpackage porting to Julia

- boundary
- integr
- lar2psm
- larcc
- largrid
- larstruct
- mapper
- morph
- simplexn
- splines
- triangulation

boundary Module

Computation of (co)boundary operators

In the current LarLib implementation, we have to distinguish between between dimension-independent, dimension-dependent, oriented and non-oriented operators.

We make a distinction between the (matrices of) boundary operators for the linear spaces C_k of chains over the field $\mathbb{Z}_2 = \{0, 1\}$ and over the field $\mathbb{Z}$ of integer numbers. We call either non-oriented or oriented the corresponding boundary operators, respectively, since the matrix elements take values within the sets $\{-1, 0, +1\}$ or $\{0, 1\}$, correspondingly. Of course, the associated matrices of coboundary operators are their transpose matrices.

For several computations, the knowledge of the matrices of non-oriented boundary operators is sufficient.

In order to compute the non-oriented boundary operator ∂_d, it it sufficient to have knowledge of the M_d and M_{d-1} characteristic matrices of d-cells and their $(d-1)$-facets, at least in the case of cellular complexes with convex cells.

`integr` Module

Finite integration of polynomial on simplicial 2-chains and 3-chains

In order to plan or control the static/dynamic behaviour of models in CAD applications, it is often necessary to evaluate integral properties ot solid models (i.e. volume, centroid, moments of inertia, etc.). This module deals with the exact evaluation of inertial properties of homogeneous polyhedral objects.

A finite integration method from (Cattani and Paoluzzi 1990) is developed here the computation of various order monomial integrals over polyhedral solids and surfaces in 3D space. The integration method can be used for the exact evaluation of domain integrals of trivariate polynomial forms.

`lar2psm` Module

Back- and forth-conversion from/to HPC (pyplasm) and LAR (sparse arrays)

This module contains all the functions needed to interface the LAR data structure and/or the geometric objects defined by it with the Plasm environment. In particular, it will include the interfaces towards the visualization primitives provided by the language.

The standard definition of vectors and matrices in plasm is the list of vector coordinates and the list of matrix rows, respectively.

The function `MKPOLS` returns a list of HPC objects, i.e.~the geometric type of the PLaSM language. This list is generated to be displayed, possibly exploded, by the `pyplasm` viewer.

Each cell `f` in the model (i.e.~each vertex list in the `FV` array of the previous example) is mapped into a polyhedral cell by the `pyplasm` operator `MKPOL`. The vertex indices are mapped from base 0 (the Python and C standard) to base 1 (the Plasm, Matlab, and FORTRAN standard).

The reverse mapping PLaSM $\rightarrow$ Larlib.jl is also implemented.

`larcc` Module

Basic data structures and conversions between sparse arrays

A few basic representation of topology are used in LARCC. They include some common sparse matrix representations: CSR (Compressed Sparse Row), CSC (Compressed Sparse Column), COO (Coordinate Representation), and BRC (Binary Row Compressed).

We denote as BRC (Binary Row Compressed) the standard input representation of our LARCC framework. A BRC representation is an array of arrays of integers, with no requirement of equal length for the component arrays. The BRC format is used to represent a (normally sparse) binary matrix. Each component array corresponds to a matrix row, and contains the indices of columns that store a 1 value. No storage is used for 0 values.

the LAR topology is based on CSR of sparse binary (and integer) matrices. We discuss the stack of matrix representations and operations implemented by this module. The current python prototype makes reference to the *scipy** implementation of sparse matrices.

Later implementations in different languages will necessarily make reference to different matrix packages.

`largrid` Module

Multidimensional cuboidal grids

Here we develop an efficient implementation of multidimensional grid generation of cuboidal and simplicial cell complexes, and a fast implementation of the more general Cartesian product of cellular complexes.

Both kind of operators, depending on the dimension of their input, may generate either full-dimensional (i.e.~solid) output complexes or cellular complexes of dimension d embedded in Euclidean space of dimension n, with $d \leq n$.

In particular, we show that both n-dimensional grids of (hyper)-cuboidal cells and their d-dimensional skeletons ($0 \leq d \leq n$), embedded in $\mathbb{E}^n$, may be properly and efficiently generated by assembling the cells produced by a number n of either 0- or 1-dimensional cell complexes, that in such lowest dimensions coincide with simplicial complexes.

`larstruct` Module

Assemblies of spaces as hierarchical assemblies

Hierarchical models of complex assemblies are generated by an aggregation of subassemblies, each one defined in a local coordinate system, and relocated by affine transformations of coordinates.

This operation may be repeated hierarchically, with some subassemblies defined by aggregation of simpler parts, and so on, until one obtains a set of elementary components, which cannot be further decomposed.

Two main advantages can be found in a hierarchical modeling approach. Each elementary part and each assembly, at every hierarchical level, are defined independently from each other, using a local coordinate frame, suitably chosen to make its definition easier.

Furthermore, only one copy of each component is stored in the memory, and may be instanced in different locations and orientations how many times it is needed.

Elementary matrices for affine transformation of vectors in any dimensional vector space are defined here. They include translation, scaling, rotation and shearing.

mapper Module

Generation of 1D, 2D, 3D manifolds from parametric equations

The mapper module aims at providing the tools needed to apply both dimension-independent affine transformations and general simplicial maps to geometric objects and assemblies developed within the LAR scheme.

A set of primitive surface and solid shapes is also implemented, via the mapping mechanism of a simplicial decomposition of a d-dimensional chart. A simplified version of the PLaSM specification of dimension-independent elementary affine transformation is given as well.

A large number of primitive manifolds, either curves, surfaces or solid,s is defined in this package, using parametric coordinates, the larMap mechanism and the coordinate functions of a suitable chart. They include 2D disks and rings, as well as cylindrical, spherical and toroidal surfaces

`morph` Module

Main operators of *Mathematical Morphology* over images

In this module we have implemented the four operators of mathematical morphology, i.e.~the dilation, erosion, opening and closing operators, by the way of matrix operations representing the linear operators—boundary and coboundary—over LAR.

According to the multidimensional character of LAR, our implementation is dimension-independent.

In few words, it works as follows: (a) the input is (the coordinate representation of) a d-chain γ; (b) compute its boundary $\partial_d(\gamma)$; (c) extract the maximal $(d-2)$-chain $\epsilon \subset \partial_d(\gamma)$; (d) consider the $(d-1)$-chain returned from its coboundary $\delta_{d-2}(\epsilon)$; (e) compute the d-chain $\eta := \delta_{d-1}(\delta_{d-2}(\epsilon)) \subset C_d$ without performing the mod 2 final transformation on the resulting coordinate vector, that would provide a zero result, according to the standard algebraic constraint $\delta \circ \delta = 0$.

It is easy to show that $\eta \equiv (\oplus\gamma) - (\ominus\gamma)$ provides the morphological gradient operator. The four standard morphological operators are therefore consequently computable.

`simplexn` Module

Combinatotial generation of d-dimensional simplicial complexes and grids

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 (Paoluzzi et al. 1993). This one in turn imports some functions from the `scipy` package and the geometric library `pyplasm`.

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.

In particular, this module 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.

`splines` Module

Tensor product methods for polynomial and rational (curve, surface and solid) splines

In this module we implement above LAR most of the parametric methods for polynomial and rational curves, surfaces and splines discussed in the book (Paoluzzi 2003), and implemented in the PLaSM language and in the python package pyplasm.

The tensor product form of surfaces will be primarily used, in this module, to support the LAR implementation of polynomial (or rational) surfaces. For this purpose, we start by defining some basic operators on function tensors.

In particular, a toolbox of basic tensor operations is given.

The `ConstFunTensor` operator produces a tensor of constant functions starting from a tensor of numbers; the recursive `FlatTensor` may be used to flatten a tensor with any number of indices by producing a corresponding one index tensor; the `InnerProd` and `TensorProd` are used to compute the inner product and the tensor product of conforming tensors of functions, respectively.

`triangulation` Module

Constrained Delaunay Triangulation (CDT) of 2-complexes

In this module we perform the whole set of manipulations necessary to triangulate the boundary polygon of a 2-chain using a CDT triangulation algorithm implemented in C, largely diffuse in games applications.

Notice that the input data may be of very general kind, corresponding to polygons that are non-connected, non-manifold w/o nested polygons of general kind. Conversely, the used library allows only for quite special polygons, with a simple exterior boundary and multiple (non-touching) internal holes, and without repeated vertices.

The module is based Python Triangle is a python wrapper around *Jonathan Richard Shewchuk**'s two-dimensional quality mesh generator and delaunay triangulator library. The Triangle package was already ported to Julia by Francesco Furiani.

People assignment

1. Beco
2. Bernieri
3. Camerini
4. Ciccone
5. Cossu
6. Fatelli
7. Gabellieri
8. Graziano
9. Loprevite
10. Melillo
11. Prevete
12. Rampogna
13. Salierno
14. Santorsola
15. Sellarione
16. Tofoni
17. Trani
18. Valletti

References

Cattani, C., and A. Paoluzzi. 1990. "Boundary Integration over Linear Polyhedra." Computer-Aided Design 22 (2): 130–35.

Paoluzzi, A. 2003. Geometric Programming for Computer Aided Design. Chichester, UK: John Wiley & Sons.

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.