Domain mapping with LAR *

Alberto Paoluzzi

November 28, 2015

Abstract

In this module a first implementation (no optimisations) is done of several `LAR` operators, reproducing the behaviour of the plasm `STRUCT` and `MAP` primitives, but with better handling of the topology, including the stitching of decomposed (simplicial domains) about their possible sewing. A definition of specialised classes `Model`, `Mat` and `Verts` is also contained in this module, together with the design and the implementation of the *traversal* algorithms for networks of structures.

Contents

1 Introduction

The `mapper` module, introduced here, aims to provide the tools needed to apply both dimension-independent affine transformations and general simplicial maps to geometric objects and assemblies developed within the LAR scheme.

*This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework [CL13]. November 28, 2015

For this purpose, a simplicial decomposition of the $[0,1]^d$ hypercube ($d \geq 1$) with any possible `shape` is firstly given, followed by its scaled version with any according `size` $\in \mathbb{E}^d$, being its position vector the mapped image of the point $\mathbf{1} \in \mathbb{E}^d$. A general mapping mechanism is specified, to map any domain decomposition (either simplicial or not) with a given set of coordinate functions, providing a piecewise-linear approximation of any curved embedding of a d-dimensional domain in any $\mathbb{E}^n$ space, with $n \geq d$. A suitable function is also given to identify corresponding vertices when mapping a domain decomposition of the fundamental polygon (or polyhedron) of a closed manifold.

The geometric tools given in this chapter employ a normalised homogeneous representation of vertices of the represented shapes, where the added coordinate is the *last* of the ordered list of vertex coordinates. The homogeneous representation of vertices is used *implicitly*, by inserting the extra coordinate only when needed by the operation at hand, mainly for computing the product of the object's vertices times the matrix of an affine tensor.

A set of primitive surface and solid shapes is also provided, 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.

The second part of this module is dedicated to the development of a complete framework for the implementation of hierarchical assemblies of shapes and scene graphs, by using the simplest possible set of computing tools. In this case no hierarchical graphs or multigraph are employed, i.e. no specialised data structures are produced. The ordered list model of hierarchical structures, inherited from PHIGS and PLaSM, is employed in this context. A recursive traversal is used to transform all the component parts of a hierarchical assembly into the reference frame of the first object of the assembly, i.e. in world coordinates.

2 Primitive objects

A large number of primitive surfaces or solids is defined in this section, using the `larMap` mechanism and the coordinate functions of a suitable chart.

2.1 1D primitives

Circle

$\langle$ Circle centered in the origin 2a $\rangle \equiv$

```
def larCircle(radius=1.,angle=2*PI,dim=1):
   def larCircle0(shape=36):
      domain = larIntervals([shape])([angle])
      V,CV = domain
      x = lambda p : radius*COS(p[0])
      y = lambda p : radius*SIN(p[0])
      return larMap([x,y])(domain,dim)
```

```
        return larCircle0
    ◇
```

Macro referenced in 8a.

Helix curve

⟨ Helix curve about the z axis 2b ⟩ ≡

```
    def larHelix(radius=1.,pitch=1.,nturns=2,dim=1):
        def larHelix0(shape=36*nturns):
            angle = nturns*2*PI
            domain = larIntervals([shape])([angle])
            V,CV = domain
            x = lambda p : radius*COS(p[0])
            y = lambda p : radius*SIN(p[0])
            z = lambda p : (pitch/(2*PI)) * p[0]
            return larMap([x,y,z])(domain,dim)
        return larHelix0
    ◇
```

Macro referenced in 8a.

2.2 2D primitives

Some useful 2D primitive objects either in $\mathbb{E}^2$ or embedded in $\mathbb{E}^3$ are defined here, including 2D disks and rings, as well as cylindrical, spherical and toroidal surfaces.

Disk surface

⟨ Disk centered in the origin 3a ⟩ ≡

```
    def larDisk(radius=1.,angle=2*PI):
        def larDisk0(shape=[36,1]):
            domain = larIntervals(shape)([angle,radius])
            V,CV = domain
            x = lambda p : p[1]*COS(p[0])
            y = lambda p : p[1]*SIN(p[0])
            return larMap([x,y])(domain)
        return larDisk0
    ◇
```

Macro referenced in 8a.

Helicoid surface

⟨ Helicoid about the z axis 3b ⟩ ≡

```
def larHelicoid(R=1.,r=0.5,pitch=1.,nturns=2,dim=1):
   def larHelicoid0(shape=[36*nturns,2]):
      angle = nturns*2*PI
      domain = larIntervals(shape,'simplex')([angle,R-r])
      V,CV = domain
      V = larTranslate([0,r,0])(V)
      domain = V,CV
      x = lambda p : p[1]*COS(p[0])
      y = lambda p : p[1]*SIN(p[0])
      z = lambda p : (pitch/(2*PI)) * p[0]
      return larMap([x,y,z])(domain,dim)
   return larHelicoid0
```
◇

Macro referenced in 8a.

Ring surface

⟨Ring centered in the origin 4a⟩ ≡

```
def larRing(r1,r2,angle=2*PI):
   def larRing0(shape=[36,1]):
      V,CV = larIntervals(shape)([angle,r2-r1])
      V = larTranslate([0,r1])(V)
      domain = V,CV
      x = lambda p : p[1] * COS(p[0])
      y = lambda p : p[1] * SIN(p[0])
      return larMap([x,y])(domain)
   return larRing0
```
◇

Macro referenced in 8a.

Cylinder surface

⟨Cylinder surface with z axis 4b⟩ ≡

```
from scipy.linalg import det
"""
def makeOriented(model):
   V,CV = model
   out = []
   for cell in CV:
      mat = scipy.array([V[v]+[1] for v in cell]+[[0,0,0,1]])
      if det(mat) < 0.0:
         out.append(cell)
      else:
         out.append([cell[1]]+[cell[0]]+cell[2:])
   return V,out
```

```
"""
def larCylinder(radius,height,angle=2*PI):
   def larCylinder0(shape=[36,1]):
      domain = larIntervals(shape)([angle,1])
      V,CV = domain
      x = lambda p : radius*COS(p[0])
      y = lambda p : radius*SIN(p[0])
      z = lambda p : height*p[1]
      mapping = [x,y,z]
      model = larMap(mapping)(domain)
      # model = makeOriented(model)
      return model
   return larCylinder0
◇
```

Macro referenced in 8a.

Spherical surface of given radius

⟨ Spherical surface of given radius 5a ⟩ ≡

```
def larSphere(radius=1,angle1=PI,angle2=2*PI):
   def larSphere0(shape=[18,36]):
      V,CV = larIntervals(shape,'simplex')([angle1,angle2])
      V = larTranslate([-angle1/2,-angle2/2])(V)
      domain = V,CV
      x = lambda p : radius*COS(p[0])*COS(p[1])
      y = lambda p : radius*COS(p[0])*SIN(p[1])
      z = lambda p : radius*SIN(p[0])
      return larMap([x,y,z])(domain)
   return larSphere0
◇
```

Macro referenced in 8a.

Toroidal surface

⟨ Toroidal surface of given radiuses 5b ⟩ ≡

```
def larToroidal(r,R,angle1=2*PI,angle2=2*PI):
   def larToroidal0(shape=[24,36]):
      domain = larIntervals(shape,'simplex')([angle1,angle2])
      V,CV = domain
      x = lambda p : (R + r*COS(p[0])) * COS(p[1])
      y = lambda p : (R + r*COS(p[0])) * SIN(p[1])
      z = lambda p : -r * SIN(p[0])
      return larMap([x,y,z])(domain)
   return larToroidal0
◇
```

Macro referenced in 8a.

Crown surface

⟨ Half-toroidal surface of given radiuses 5c ⟩ ≡

```
def larCrown(r,R,angle=2*PI):
   def larCrown0(shape=[24,36]):
      V,CV = larIntervals(shape,'simplex')([PI,angle])
      V = larTranslate([-PI/2,0])(V)
      domain = V,CV
      x = lambda p : (R + r*COS(p[0])) * COS(p[1])
      y = lambda p : (R + r*COS(p[0])) * SIN(p[1])
      z = lambda p : -r * SIN(p[0])
      return larMap([x,y,z])(domain)
   return larCrown0
```

◇

Macro referenced in 8a.

2.3 3D primitives

Solid Box

⟨ Solid box of given extreme vectors 6a ⟩ ≡

```
def larBox(minVect,maxVect):
   size = VECTDIFF([maxVect,minVect])
   print "size =",size
   box = larApply(s(*size))(larCuboids([1]*len(size)))
   print "box =",box
   return larApply(t(*minVect))(box)
```

◇

Macro referenced in 8a.

Solid Ball

⟨ Solid Sphere of given radius 6b ⟩ ≡

```
def larBall(radius=1,angle1=PI,angle2=2*PI):
   def larBall0(shape=[18,36]):
      V,CV = checkModel(larSphere(radius,angle1,angle2)(shape))
      return V,[range(len(V))]
   return larBall0
```

◇

Macro referenced in 8a.

Solid cylinder

⟨ Solid cylinder of given radius and height 6c ⟩ ≡

```
def larRod(radius,height,angle=2*PI):
   def larRod0(shape=[36,1]):
      V,CV = checkModel(larCylinder(radius,height,angle)(shape))
      return V,[range(len(V))]
   return larRod0
◇
```

Macro referenced in 8a.

Hollow cylinder

⟨ Hollow cylinder of given radiuses and height 6d ⟩ ≡

```
def larHollowCyl(r,R,height,angle=2*PI):
   def larHollowCyl0(shape=[36,1,1]):
      V,CV = larIntervals(shape)([angle,R-r,height])
      V = larTranslate([0,r,0])(V)
      domain = V,CV
      x = lambda p : p[1] * COS(p[0])
      y = lambda p : p[1] * SIN(p[0])
      z = lambda p : p[2] * height
      return larMap([x,y,z])(domain)
   return larHollowCyl0
◇
```

Macro referenced in 8a.

Hollow sphere

⟨ Hollow sphere of given radiuses 7a ⟩ ≡

```
def larHollowSphere(r,R,angle1=PI,angle2=2*PI):
   def larHollowSphere0(shape=[36,1,1]):
      V,CV = larIntervals(shape)([angle1,angle2,R-r])
      V = larTranslate([-angle1/2,-angle2/2,r])(V)
      domain = V,CV
      x = lambda p : p[2]*COS(p[0])*COS(p[1])
      y = lambda p : p[2]*COS(p[0])*SIN(p[1])
      z = lambda p : p[2]*SIN(p[0])
      return larMap([x,y,z])(domain)
   return larHollowSphere0
◇
```

Macro referenced in 8a.

Solid torus

⟨ Solid torus of given radiuses 7b ⟩ ≡

```
def larTorus(r,R,angle1=2*PI,angle2=2*PI):
   def larTorus0(shape=[24,36,1]):
      domain = larIntervals(shape)([angle1,angle2,r])
      V,CV = domain
      x = lambda p : (R + p[2]*COS(p[0])) * COS(p[1])
      y = lambda p : (R + p[2]*COS(p[0])) * SIN(p[1])
      z = lambda p : -p[2] * SIN(p[0])
      return larMap([x,y,z])(domain)
   return larTorus0
```

◇

Macro referenced in 8a.

Solid pizza

⟨ Solid pizza of given radiuses 7c ⟩ ≡

```
def larPizza(r,R,angle=2*PI):
   assert angle <= PI
   def larPizza0(shape=[24,36]):
      V,CV = checkModel(larCrown(r,R,angle)(shape))
      V += [[0,0,-r],[0,0,r]]
      return V,[range(len(V))]
   return larPizza0
```

◇

Macro referenced in 8a.

3 Computational framework

3.1 Exporting the library

"larlib/larlib/mapper.py" 8a ≡

```
""" Mapping functions and primitive objects """
from larlib import *
```

⟨ Basic tests of mapper module 9b ⟩
⟨ Circle centered in the origin 2a ⟩
⟨ Helix curve about the z axis 2b ⟩
⟨ Disk centered in the origin 3a ⟩
⟨ Helicoid about the z axis 3b ⟩
⟨ Ring centered in the origin 4a ⟩
⟨ Spherical surface of given radius 5a ⟩
⟨ Cylinder surface with z axis 4b ⟩
⟨ Toroidal surface of given radiuses 5b ⟩

⟨ Half-toroidal surface of given radiuses 5c ⟩
⟨ Solid box of given extreme vectors 6a ⟩
⟨ Solid Sphere of given radius 6b ⟩
⟨ Solid helicoid about the z axis ? ⟩
⟨ Solid cylinder of given radius and height 6c ⟩
⟨ Solid torus of given radiuses 7b ⟩
⟨ Solid pizza of given radiuses 7c ⟩
⟨ Hollow cylinder of given radiuses and height 6d ⟩
⟨ Hollow sphere of given radiuses 7a ⟩
⟨ Symbolic utility to represent points as strings ? ⟩
⟨ Remove the unused vertices from a LAR model pair ? ⟩
◇

3.2 Examples

3D rotation about a general axis The approach used by `lar-cc` to specify a general 3D rotation is shown in the following example, by passing the rotation function `r` the components `a,b,c` of the unit vector `axis` scaled by the rotation `angle`.

"test/py/mapper/test02.py" 8b ≡

```
""" General 3D rotation of a toroidal surface """
from larlib import *

model = checkModel(larToroidal([0.2,1])())
angle = PI/2; axis = UNITVECT([1,1,0])
a,b,c = SCALARVECTPROD([ angle, axis ])
model = larApply(r(a,b,c))(model)
VIEW(STRUCT(MKPOLS(model)))
```
◇

3D elementary rotation of a 2D circle A simpler specification is needed when the 3D rotation is about a coordinate axis. In this case the rotation angle can be directly given as the unique non-zero parameter of the the rotation function `r`. The rotation axis (in this case the x one) is specified by the non-zero (angle) position.

"test/py/mapper/test03.py" 9a ≡

```
""" Elementary 3D rotation of a 2D circle """
from larlib import *

model = checkModel(larCircle(1)())
model = larEmbed(1)(model)
model = larApply(r(PI/2,0,0))(model)
VIEW(STRUCT(MKPOLS(model)))
```
◇

3.3 Tests about domain

Mapping domains The generations of mapping domains of different dimension (1D, 2D, 3D) is shown below.

⟨ Basic tests of mapper module 9b ⟩ ≡

```
""" Basic tests of mapper module """
from larlib import *

if __name__=="__main__":
   V,EV = larDomain([5])
   VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,EV))))
   V,EV = larIntervals([24])([2*PI])
   VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,EV))))

   V,FV = larDomain([5,3])
   VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,FV))))
   V,FV = larIntervals([36,3])([2*PI,1.])
   VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,FV))))

   V,CV = larDomain([5,3,1])
   VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,CV))))
   V,CV = larIntervals([36,2,3])([2*PI,1.,1.])
   VIEW(EXPLODE(1.5,1.5,1.5)(MKPOLS((V,CV))))
```
◇

Macro referenced in 8a.

Testing some primitive object generators The various model generators given in Section 2 are tested here, including LAR 2D circle, disk, and ring, as well as the 3D cylinder, sphere, and toroidal surfaces, and the solid objects ball, rod, crown, pizza, and torus.

"test/py/mapper/test01.py" 10 ≡

```
""" Testing some primitive object generators """
from larlib import *

model = larCircle(1)()
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
model = larHelix(1,0.5,4)()
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
model = larDisk(1)([36,4])
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
model = larHelicoid(1,0.5,0.1,10)()
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
model = larRing(.9, 1.)([36,2])
```

```
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
model = larCylinder(.5,2.)([32,1])
VIEW(STRUCT(MKPOLS(model)))
model = larSphere(1,PI/6,PI/4)([6,12])
VIEW(STRUCT(MKPOLS(model)))
model = larBall(1)()
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
model = larSolidHelicoid(0.2,1,0.5,0.5,10)()
VIEW(STRUCT(MKPOLS(model)))
model = larRod(.25,2.)([32,1])
VIEW(STRUCT(MKPOLS(model)))
model = larToroidal(0.5,2)()
VIEW(STRUCT(MKPOLS(model)))
model = larCrown(0.125,1)([8,48])
VIEW(STRUCT(MKPOLS(model)))
model = larPizza(0.05,1,PI/3)([8,48])
VIEW(STRUCT(MKPOLS(model)))
model = larTorus(0.5,1)()
VIEW(STRUCT(MKPOLS(model)))
model = larBox([-1,-1,-1],[1,1,1])
VIEW(STRUCT(MKPOLS(model)))
model = larHollowCyl(0.8,1,1,angle=PI/4)([12,2,2])
VIEW(STRUCT(MKPOLS(model)))
model = larHollowSphere(0.8,1,PI/6,PI/4)([6,12,2])
VIEW(STRUCT(MKPOLS(model)))
◇
```

3.4 Volumetric utilities

Limits of a LAR Model

⟨Model limits 11a⟩ ≡

```
def larLimits (model):
   if isinstance(model,tuple):
      V,CV = model
      verts = scipy.asarray(V)
   else: verts = model.verts
   return scipy.amin(verts,axis=0).tolist(), scipy.amax(verts,axis=0).tolist()

assert larLimits(larSphere()()) == ([-1.0, -1.0, -1.0], [1.0, 1.0, 1.0])
◇
```

Macro never referenced.

Alignment

⟨Alignment primitive 11b⟩ ≡

```
def larAlign (args):
   def larAlign0 (args,pols):
      pol1, pol2 = pols
      box1, box2 = (larLimits(pol1), larLimits(pol2))
      print "box1, box2 =",(box1, box2)

   return larAlign0
```
◇

Macro never referenced.

References

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