Automatic generation of simplified buildings from 2D wire-frame *

Alberto Paoluzzi

March 18, 2015

Abstract

In this module we develop a small library of functions to generate automatically simplified models of 3D buildings starting from 2D wire-frame drawings. The generated geometries will be exported to xGeoJson, a hierarchical data format extending GeoJson, the standard for geographical web maps, with assemblies and local rectangular coordinates.

Contents

1 Introduction

A simplified 2D layout of building floors may be generated by using either a simple web UI providing only two or three interactive graphics primitives (rect and polyline, in particular)

*This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework [CL13]. March 18, 2015 or by exporting the output of a drawing program to a standard 2D graphics data format, in particular to SVG (Simple Vector Graphics, the vector graphics standard for the web). This information will be used to automatically generate a simplified 3D model of the building and to export its geometry and topology to an external data format called xGeoJson, a customised extension of GeoJson an opensource standard for geographical information.

2 Transformation filters

2.1 From svg to lar

SVG primitives to lar Two SVG primitives are currently used to define the wire-frame layout of the building floors using either a drawing application or an interactive user interface within the browser: rect (for the definition of rectangles) and polyline (for the definition of closed polylines).

⟨transform svg primitives to basic lar format 2a⟩ ≡

```
""" transform svg primitives to basic lar format """
def rects2polylines(rooms):
   return [[[x,y],[x+dx,y],[x+dx,y+dy],[x,y+dy]] for x,y,dx,dy in rooms]

def polyline2lar(polylines):
   index,defaultValue = -1,-1
   Vdict,FV = dict(),[]
   for k,polyline in enumerate(polylines):
      cell = []
      for vert in polyline:
         key = vcode(vert)
         if Vdict.get(key,defaultValue) == defaultValue:
            index += 1
            Vdict[key] = index
            cell += [index]
         else:
            cell += [Vdict[key]]
      FV += [cell]
   items = TRANS(Vdict.items())
   V = TRANS(sorted(zip(items[1],AA(eval)(items[0]))))[1]
   #FV = AA(sorted)(FV)
   EV = face2edge(FV)
   return V,FV,EV
```
◇

Macro referenced in 6a.

2.2 From lar to struct

Mapping of a lar model to a list of lar structures

⟨ transform a lar model to a list of lar structures 2b ⟩ ≡

```
""" transform a lar model to a list of lar structures """
def lar2Structs(model):
   V,FV,_ = model
   return [ Struct([[[V[v] for v in cell], [range(len(cell))]]]) for cell in FV]
◇
```

Macro referenced in 6a.

⟨ Struct class pushing local origin at the bottom of the structure 2c ⟩ ≡

```
class Struct2(Struct):
   def flatten(self):
      structs = copy(self.body)
      structList = lar2Structs(CAT(structs.body))
      return [ absModel2relStruct(struct[0].body) for struct in structList ]
◇
```

Macro referenced in 6a.

⟨ transform an absolute lar model to a relative lar structure 3a ⟩ ≡

```
""" transform an absolute lar model to a relative lar structure """
def absModel2relStruct(larModel):
   V,E = larModel
   Vnew = (array(V) - V[0]).tolist()
   return Struct([ t(*V[0]), (Vnew,E) ])
◇
```

Macro referenced in 6a.

2.3 Assembling structures

2.4 Exporting to xGeoJson

Exporting to JSON via YML a lar structure

⟨ Exporting to JSON via YML 3b ⟩ ≡

```
""" exporting to JSON via YML a lar structure """
file_handle = open(path+filename+".yml")
my_dictionary = yaml.safe_load(file_handle)
file_handle.close()
with open(path+filename+".json", 'w') as outfile:
     json.dump(my_dictionary, outfile, sort_keys=True, indent=4, ensure_ascii=False)
◇
```

Macro referenced in 3c.

print a lar structure to a geoJson file

⟨ Print a lar structure to a geoJson file 3c ⟩ ≡

```
""" print a lar structure to a geoJson file """
import yaml
import json

def printStruct2GeoJson(path,struct):
    if struct.__name__() == None:
        filename = str(id(struct))
    else:
        filename = struct.__name__()
    theFile = open(path+filename+".yml", "w")
    print >> theFile, "---"
    print "filename =", path+filename+".yml"
    dim = checkStruct(struct.body)
    CTM, stack = scipy.identity(dim+1), []
    fathers = [filename]
    scene,fathers = printTraversal(theFile, CTM, stack, struct, [], 0, fathers,filename)
    theFile.close()
    ⟨ Exporting to JSON via YML 3b ⟩
    return scene,fathers

⟨ Print a Model object in a geoJson file 4a ⟩
⟨ Print a Mat object in a geoJson file 4b ⟩
⟨ Print a Struct object in a geoJson file 5a ⟩
⟨ Traverse a structure to print a geoJson file 5b ⟩
◇
```

Macro referenced in 6a.

Print a `Model` object in a `xGeoJson` file

⟨ Print a Model object in a geoJson file 4a ⟩ ≡

```
""" Print a model object in a geoJson file """
def printModelObject(theFile,tabs, i,name,category,verts,cells,father):
    tab = "    "
    print >> theFile, "-   ","id:", name
    print >> theFile, tab, "type:", "Feature"
    print >> theFile, tab, "geometry:"
    print >> theFile, tab+tab, "type:", "Polygon"
    print >> theFile, tab+tab, "coordinates:"
    print >> theFile, tab+tab+tab, AA(eval)(AA(vcode)(verts))
    print >> theFile, tab, "properties:"
    print >> theFile, tab+tab, "class:", category
    print >> theFile, tab+tab, "parent:", father
    print >> theFile, tab+tab, "son:", i
```

```
        print >> theFile, tab+tab, "description:", name
        print >> theFile, tab+tab, "tVector:", [0, 0, 0]
        print >> theFile, tab+tab, "rVector:", [0, 0, 0]
    ◇
```

Macro referenced in 3c.

Print a Mat object in a xGeoJson file

⟨ Print a Mat object in a geoJson file 4b ⟩ ≡

```
    """ Print a mat object in a geoJson file """
    def printMatObject(theFile,tabs, theMat):
        tab = "    "
        print >> theFile, tab+tab, "tVector:", theMat.T[-1,:-1].tolist()
    ◇
```

Macro referenced in 3c.

Print a Struct object in a xGeoJson file

⟨ Print a Struct object in a geoJson file 5a ⟩ ≡

```
    """ Print a struct object in a geoJson file """
    def printStructObject(theFile,tabs, i,name,category,box,father):
        tab = "    "
        print >> theFile, "-   ","id:", name
        print >> theFile, tab,"type:", "Feature"
        print >> theFile, tab,"geometry:"
        print >> theFile, tab+tab, "type:", "Polygon"
        print >> theFile, tab+tab, "box:", box
        print >> theFile, tab,"properties:"
        print >> theFile, tab+tab, "class:", category
        print >> theFile, tab+tab, "parent:", father
        print >> theFile, tab+tab, "son:", i
        print >> theFile, tab+tab, "description:", name
        print >> theFile, tab+tab, "tVector:", [0, 0, 0]
        print >> theFile, tab+tab, "rVector:", [0, 0, 0]
    ◇
```

Macro referenced in 3c.

Traverse a structure to print a geoJson file

⟨ Traverse a structure to print a geoJson file 5b ⟩ ≡

```
    """ Traverse a structure to print a geoJson file """
    def printTraversal(theFile,CTM, stack, obj, scene=[], level=0, fathers=[],father=""):
       tabs = (4*level)*" "
       for i in range(len(obj)):
          if isinstance(obj[i],Model):
```

```
            i,verts,cells = obj[i]
            if obj[i].__name__() == None:
               name = father+'-'+ str(id(obj[i]))
            else:
               name = father+'-'+ str(obj[i].__name__())
            printModelObject(theFile,tabs, i,name,obj[i].__category__(),verts,cells,father)
            scene += [larApply(CTM)(obj[i])]
            fathers += [father]
         elif (isinstance(obj[i],tuple) or isinstance(obj[i],list)) and len(obj[i])==2:
            verts,cells = obj[i]
            name = father+'-'+ str(id(obj[i]))
            printModelObject(theFile,tabs, i,name,obj[i].__category__(),verts,cells,father)
            scene += [larApply(CTM)(obj[i])]
            fathers += [father]
         elif isinstance(obj[i],Mat):
            printMatObject(theFile,tabs, obj[i])
            CTM = scipy.dot(CTM, obj[i])
         elif isinstance(obj[i], Struct):
            if obj[i].__name__() == None:
               name = father+'-'+ str(id(obj[i]))
            else:
               name = father+'-'+ str(obj[i].__name__())
            box = obj[i].box
            printStructObject(theFile,tabs, i,name,obj[i].__category__(),box,father)
            stack.append(CTM)
            level += 1
            fathers.append(name)
            printTraversal(theFile,CTM, stack, obj[i], scene, level, fathers,name)
            name = fathers.pop()
            level -= 1
            CTM = stack.pop()
      return scene,fathers
   ◇
```

Macro referenced in 3c.

3 Iot3D Exporting

"lib/py/iot3d.py" 6a ≡

```
   """Module with automatic generation of simplified 3D buildings"""
   import sys; sys.path.insert(0, 'lib/py/')
   from architectural import *
   ⟨Struct class pushing local origin at the bottom of the structure 2c⟩
   ⟨transform svg primitives to basic lar format 2a⟩
   ⟨transform a lar model to a list of lar structures 2b⟩
   ⟨transform an absolute lar model to a relative lar structure 3a⟩
```

⟨Print a lar structure to a geoJson file 3c⟩
◇

4 Examples

4.1 A complete 3D building example

In this section a complete 3D building example is developed, that starts by importing the data associated to the rect and polyline primitives of the 2D layout exported as svg file, and finishes by generating a complete 3D mock-up of a quite complex multi-floor office building.

"test/py/iot3d/test01.py" 6b ≡

```
"""Automatic construction of a simplified 3D building from 2D layout"""
import sys
PATH = "/Users/paoluzzi/Documents/RICERCA/sogei/edifici/"
sys.path.insert(0, PATH)

from buildings import *
from iot3d import *

# LAR models (absolute coordinates)
ala_est = larEmbed(1)(polyline2lar(rects2polylines(eastRooms) + eastTip))
ala_sud = larEmbed(1)(polyline2lar(rects2polylines(southRooms) + southTip))
ala_ovest = larEmbed(1)(polyline2lar(rects2polylines(westRooms) + westTip))
ala_nord = larEmbed(1)(polyline2lar(rects2polylines(northRooms) + northTip))
ascensori = larEmbed(1)(polyline2lar(elevators))
spazioComune = larEmbed(1)(polyline2lar(AA(REVERSE)(newLanding)))

# test of input consistency (flat assembly of LAR models)
pianoTipo = Struct([ala_est,ala_sud,ala_ovest,ala_nord,ascensori,spazioComune],"pianoTipo")
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(struct2lar(pianoTipo))))
VIEW(SKEL_1(EXPLODE(1.2,1.2,1.2)(MKPOLS(struct2lar(pianoTipo)))))

# LAR to structs
Ala_est = Struct(lar2Structs(ala_est),"Ala_est")
Ala_sud = Struct(lar2Structs(ala_sud),"Ala_sud")
Ala_ovest = Struct(lar2Structs(ala_ovest),"Ala_ovest")
Ala_nord = Struct(lar2Structs(ala_nord),"Ala_nord")
Ascensori = Struct(lar2Structs(ascensori),"Ascensori")
SpazioComune = Struct(lar2Structs(spazioComune),"SpazioComune")

model = struct2lar(Ala_est)
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(model)))
for k,room in enumerate(Ala_est.body):
```

```
    print room.body

# hierarchical assembly of simplest LAR models
TreAli = Struct([Ala_est,Ala_sud,Ala_ovest,Ascensori,SpazioComune],"TreAli")
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(struct2lar(TreAli))))
Ali_B_C_D = Struct(6*[TreAli, t(0,0,30)],"Ali_B_C_D")
VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(struct2lar(Ali_B_C_D))))

Ala_A = Struct(4*[Ala_nord,  t(0,0,30)],"Ala_A")
ALA_A = EXPLODE(1.2,1.2,1.2)(MKPOLS(struct2lar(Ala_A)))
VIEW(EXPLODE(1.2,1.2,1.2)([ ALA_A, COLOR(BLUE)(SKEL_1(ALA_A)) ]))

Edificio = Struct([ Ala_A, Ali_B_C_D ],"Edificio")
V,FV = struct2lar(Edificio)
EDIFICIO = STRUCT(MKPOLS((V,FV)))
VIEW(STRUCT([ EDIFICIO, COLOR(BLUE)(SKEL_1(EDIFICIO)) ]))

VV = AA(LIST)(range(len(V)))
SK = SKEL_1(EDIFICIO)
VIEW(larModelNumbering(1,1,1)(V,[VV,[],FV],STRUCT([ COLOR(BLUE)(SK), EDIFICIO ]),20))

Va,EVa = struct2lar(TreAli)
Vb,EVb = struct2lar(Ala_A)
a = PROD([ SKEL_1(STRUCT(MKPOLS( ([ v[:2] for v in Va], EVa) ))), QUOTE(5*[30]) ])
b = PROD([ SKEL_1(STRUCT(MKPOLS( ([ v[:2] for v in Vb], EVb) ))), QUOTE(3*[30]) ])
glass = MATERIAL([1,0,0,0.3,  0,1,0,0.3,  0,0,1,0.3, 0,0,0,0.3, 100])
VIEW(glass(STRUCT([a,b])))

VIEW(STRUCT([ glass(STRUCT([a,b])), EDIFICIO, COLOR(BLUE)(SKEL_1(EDIFICIO)) ]))
scene,fathers = printStruct2GeoJson(PATH,pianoTipo)
scene,fathers = printStruct2GeoJson(PATH,Edificio)
```
◇

References

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