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

Alberto Paoluzzi

December 3, 2014

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) or by exporting the output of a drawing program to a standard 2D graphics data format, in

*This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework [CL13]. December 3, 2014

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)
   return V,FV
```

◇

Macro referenced in 4.

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 4.

⟨ transform an absolute lar model to a relative lar structure 2c ⟩ ≡

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

Macro referenced in 4.

2.3 Assembling structures

2.4 Exporting to xGeoJson

print a lar structure to a geoJson file

⟨ print a lar structure to a geoJson file 3a ⟩ ≡

```
""" print a lar structure to a geoJson file """
def printStruct2GeoJson(struct):
   dim = checkStruct(struct.body)
   print "\n dim =",dim
   CTM, stack = scipy.identity(dim+1), []
   print "\n CTM, stack =",CTM, stack
   scene = printTraversal(CTM, stack, struct, [], 0)
   return scene
```
◇

Macro referenced in 4.

Traverse a structure to print a geoJson file

⟨ Traverse a structure to print a geoJson file 3b ⟩ ≡

```
""" Traverse a structure to print a geoJson file """
def printTraversal(CTM, stack, obj, scene=[], level=0):
   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 = id(obj[i])
```

```
            else:
               name = obj[i].__name__()
            print tabs, "i =",i
            print tabs, "name =",name
            print tabs, "verts =",AA(eval)(AA(vcode)(verts))
            print tabs, "cells =",cells
            scene += [larApply(CTM)(obj[i])]
         elif (isinstance(obj[i],tuple) or isinstance(obj[i],list)) and len(obj[i])==2:
            verts,cells = obj[i]
            name = id(obj[i])
            print tabs, "i =",i
            print tabs, "name =",name
            print tabs, "verts =",AA(eval)(AA(vcode)(verts))
            print tabs, "cells =",cells
            scene += [larApply(CTM)(obj[i])]
         elif isinstance(obj[i],Mat):
            print tabs, "tVector =", obj[i].T[-1].tolist()
            CTM = scipy.dot(CTM, obj[i])
         elif isinstance(obj[i], Struct):
            if obj[i].__name__() == None:
               name = id(obj[i])
            else:
               name = obj[i].__name__()
            print tabs, "i =",i
            print tabs, "name =",name
            stack.append(CTM)
            level += 1
            printTraversal(CTM, stack, obj[i], scene, level)
            level -= 1
            CTM = stack.pop()
      return scene
```

◇

Macro referenced in 4.

3 Iot3D Exporting

"lib/py/iot3d.py" 4 ≡

```
"""Module with automatic generation of simplified 3D buildings"""
import sys; sys.path.insert(0, 'lib/py/')
from architectural import *
```

⟨ 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 2c ⟩

⟨ print a lar structure to a geoJson file 3a ⟩

⟨ Traverse a structure to print a geoJson file 3b ⟩

◇

4 Examples

References

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