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

Alberto Paoluzzi

December 4, 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 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).

[*]This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework [**?**]. December 4, 2014

@D transform svg primitives to basic lar format @""" 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 @

2.2 From `lar` to `struct`

Mapping of a `lar` model to a list of `lar` structures @D transform a lar model to a list of lar structures @""" 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] @

@D transform an absolute lar model to a relative lar structure @""" 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)]) @

2.3 Assembling `structures`

2.4 Exporting to `xGeoJson`

print a lar structure to a geoJson file @D Print a lar structure to a geoJson file @""" print a lar structure to a geoJson file """ def printStruct2GeoJson(path,struct): if struct.$_{name}{}_{()==None:filename=str(id(struct))else:filename=struct.}{}_{name}{}_{()theFile=open(path+filename+".geo","w")print"filename=",filenamed}$

@¡ Print a Model object in a geoJson file @¿ @¡ Print a Mat object in a geoJson file @¿ @¡ Print a Struct object in a geoJson file @¿ @¡ Traverse a structure to print a geoJson file @¿ @

Print a `Model` object in a `xGeoJson` file @D Print a Model object in a geoJson file @""" Print a model object in a geoJson file """ def printModelObject(theFile,tabs, i,name,verts,cells): print ¿¿ theFile, tabs, "i =",i print ¿¿ theFile, tabs, "name =",name print ¿¿ theFile, tabs, "verts =",AA(eval)(AA(vcode)(verts)) print ¿¿ theFile, tabs, "cells =",cells @

Print a `Mat` object in a `xGeoJson` file @D Print a Mat object in a geoJson file @""" Print a mat object in a geoJson file """ def printMatObject(theFile,tabs, theMat): print ¿¿ theFile, tabs, "tVector =", theMat.T[-1].tolist() @

Print a `Struct` object in a `xGeoJson` file @D Print a Struct object in a geoJson file @""" Print a struct object in a geoJson file """ def printStructObject(theFile,tabs, i,name): print ¿¿ theFile, tabs, "i =",i print ¿¿ theFile, tabs, "name =",name @

Traverse a structure to print a geoJson file @D Traverse a structure to print a geoJson file @""" Traverse a structure to print a geoJson file """ def printTraversal(theFile,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_{()printModelObject(t}}$

3 `Iot3D` Exporting

@O lib/py/iot3d.py @"""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 @¿ @¡ transform a lar model to a list of lar structures @¿ @¡ transform an absolute lar model to a relative lar structure @¿ @¡ Print a lar structure to a geoJson file @¿ @

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.

@O test/py/iot3d/test01.py @"""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$_est = larEmbed(1)(polyline2lar(rects2polylines(eastRooms)+ eastTip))ala_sud = larEmbed(1)(polyline2lar(rects2polylines(southRooms)+southTip))ala_ovest = larEmbed(1)(polyline2lar(rects2polylines(westRooms)+westTip))ala_nord = larEmbed(1)(polyline2lar(rec$
$northTip))ascensori = larEmbed(1)(polyline2lar(elevators))spazioComune = larEmbed(1)(polyline2lar(A$

test of input consistency (flat assembly of LAR models) pianoTipo = Struct([ala$_est, ala_sud, ala_ovest, ala_nor$

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

model = struct2lar(Ala$_est)VIEW(EXPLODE(1.2, 1.2, 1.2)(MKPOLS(model)))fork, roominenumerat$
$printroom.body$

hierarchical assembly of simplest LAR models TreAli = Struct([Ala$_est, Ala_sud, Ala_ovest, Ascensori, Spazio$
$Struct(6*[TreAli, t(0,0,30)], "Ali_{BCD}")VIEW(EXPLODE(1.2,1.2,1.2)(MKPOLS(struct2lar(Ali_{BCD})))$

Ala$_A = Struct(4*[Ala_nord, t(0,0,30)], "Ala_A")ALA_A = EXPLODE(1.2,1.2,1.2)(MKPOLS(struct2lar($

Edificio = Struct([Ala$_A, Ali_{BCD}], "Edificio")V, FV = struct2lar(Edificio)EDIFICIO =$
$STRUCT(MKPOLS((V, FV)))VIEW(STRUCT([EDIFICIO, COLOR(BLUE)(SKEL_1(EDIFICIO))]$

VV = AA(LIST)(range(len(V))) SK = SKEL$_1(EDIFICIO)VIEW(larModelNumbering(1,1,1)(V, [VV,$

Va,EVa = struct2lar(TreAli) Vb,EVb = struct2lar(Ala$_A)a = PROD([SKEL_1(STRUCT(MKPOLS(([v[:$
$2]forvinVa], EVa)))), QUOTE(5*[30])])b = PROD([SKEL_1(STRUCT(MKPOLS(([v[:$
$2]forvinVb], EVb)))), QUOTE(3*[30])])glass = MATERIAL([1,0,0,0.3,0,1,0,0.3,0,0,1,0.3,0,0,0,0.3,100$

VIEW(STRUCT([glass(STRUCT([a,b])), EDIFICIO, COLOR(BLUE)(SKEL$_1(EDIFICIO))]))test =$
$printStruct2GeoJson(PATH, Edificio)$@