DXF file to GEM file conversion

Pierre Salou
Pantechnik
13 Rue de la Rsistance, 14400 Bayeux (France)

pierre.salou@pantechnik.com

Simion user meeting, GANIL, 11/06/2015

Introduction

Simion software is widely used to simulate ion optics. Frequently the simulated systems are already drawn on a CAD (Computer-aided design) software. In order to avoid any redrawing of the system, one can use the Simion SL Tools package, which allows a conversion of a STL file to a PA file. This method is however not suitable for axis-symmetry systems; it only provides 3D potential array which can be costly for the memory.

In order to efficiently transfer an axis-symmetry design, we propose to convert a DXF file into a GEM file. Simion is then able to convert this file into 2D PA. The following document presents the "DXF2GEM" lua library and the instructions to use it.

Instructions

The "DXF2GEM" lua library is provided at the end of the document. To run the library, a DXF edition software is needed. We suggest Inkscape[1], as it is a powerful and free software.

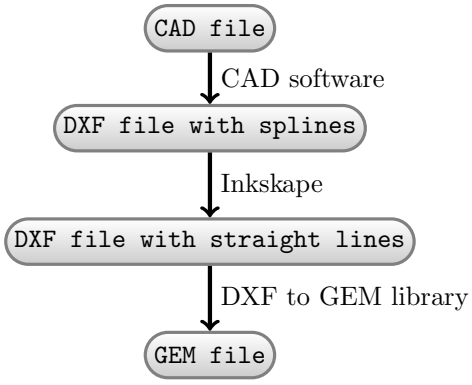

Figure 1: Instructions to use the library.

The steps of the conversion is described in figure 1, we start from the CAD file which has to be transformed to a DXF file. As the DXF format is common, this conversion is possible on many CAD software. During this step make sure that the origin is well defined and no striped area is present (otherwise the stripes would be recognized as drawing lines).

Once the DXF file is generated, the DXF edition software can be used to modify the file. Indeed the "DXF2GEM" lua library is not able to read curved lines (splines). By using a "flatten Bézier curves" function we can convert the curves into series of straight lines (see figure 2). The minimum size of the straight lines has to be defined according to the precision needed. The DXF edition software is also useful to check the origin, to remove any unnecessary parts or even to add new parts by directly drawing it.

Figure 2: Conversion of splines to straight lines.

When the DXF modifications are saved, the conversion is ready to be done. The "DXF2GEM" library has to be edited and placed into the same folder than the DXF file. The table 1 presents the parameters to be modified.

Table 1: Example of table

Parameter	Definition
input_file	DXF file to convert
output_file	output GEM file
mesh_size	mesh size of GEM file
fit_to_drawing	size of the GEM = size of DXF
xmin, xmax, etc.	size of the GEM
merge	merge lines when close enough
merge_tolerance	merging threshold

The "DXF2GEM" library is then ready to run by using the "Run Lua Program" function in Simion. The generated GEM file can be converted into PA by following the normal procedure.

[1] https://inkscape.org/

LUA Code

```
--[[
read DXF
This is a lua script dedicated to the conversion on DXF file to GEM file.
Pierre Salou, Pantechnik 2015
]]--
local input_file = "file.dxf" --file to read
local output_file = "output.gem" -- output file
local mesh_size =0.1 --mesh size in mm

local polylines ={} -- polylines table

local fit_to_drawing =false -- If true : the size of the PA is the size of
   ↪ the drawing,
-- if false : use this values
local xmin = 0
local ymin = 0
local xmax = 505
local ymax = 100

local merge =true -- merge the closest lines together using the merging
   ↪ tolerance
local merge_tolerance = 0.001 -- merging tolerance in mm

------------------------------------------------------------ Read dxf

function readLWPOLYLINE() --read and store all the polylines

        local polylinecmd = {}
        local word
        local tx ={}
        local ty ={}

        repeat
        word = file:read("*line")
        if string.match(word, " 10") then tx[#tx+1] = file:read("*line") end
        if string.match(word, " 20") then ty[#ty+1] = file:read("*line") end

        polylinecmd[#polylinecmd +1] = word
        until string.match(word, " 0")

        local coordinate = {}
        coordinate[1] = tx
        coordinate[2] = ty
        polylines[#polylines+1] = coordinate

end

function readCMD(cmd)

        local word
        repeat
        word = file:read("*line")
        until string.match(word, cmd)

        return file:read("*line")
end

------------------------------------------------------------ gestion poly
```

```
function plotpoly()
        for k,v in pairs(polylines) do
                plotpoly(k)
        end
end

function plotpoly(k)

        print("Poly n=".. k)
        for j,v in pairs(polylines[k][1]) do
                print(j,polylines[k][1][j],polylines[k][2][j])
        end

end

function plotpolydetail()

        for k,v in pairs(polylines) do
                print("Poly n=".. k .. "(".. polylines[k][1][1] .. ",..
                   ↪ polylines[k][2][1] .. ")--(".. polylines[k][1][#
                   ↪ polylines[k][1]] .. ",".. polylines[k][2][#polylines[k
                   ↪ ][2]]..")     nb seg = ".. #polylines[k][1])

        end

end

function cleartable(t)
        for k in pairs (t) do
                t [k] = nil
        end
end

function poly_minx(poly)

        local xm = tonumber(poly[1][1])
        for k,v in pairs(poly[1]) do
                if (xm > tonumber(poly[1][k])) then xm = tonumber(poly[1][k
                   ↪ ]) end
        end
        return xm

end

function compare_min(polya, polyb)
        return poly_minx(polya)<poly_minx(polyb)
end

------------------------------------------------------------------ AB_BC
function comparepolyline_AB_BC(polyAB, polyBC, delta)
        if polyAB==nil or polyBC==nil then return false end
        local x = polyAB[1][#polyAB[1]] - polyBC[1][1]
        local y = polyAB[2][#polyAB[2]] - polyBC[2][1]
        if (delta >= sqrt( x*x + y*y)) then
        end
        return (delta >= sqrt( x*x + y*y))
end

function mergepolyline_AB_BC(polyAB, polyBC)
        for k,v in pairs(polyBC[1]) do
```

```
                polyAB[1][#polyAB[1]+1] = polyBC[1][k]
                polyAB[2][#polyAB[2]+1] = polyBC[2][k]
        end

end

function mergepolylines_AB_BC()
                for k,v in pairs(polylines) do
                        for j,v in pairs(polylines) do
                                if k~=j then
                                        if comparepolyline_AB_BC(polylines[k
                                           ↪ ], polylines[j],
                                           ↪ merge_tolerance) then
                                                print("merge AB-BC poly n="
                                                   ↪ .. k .. "   with poly
                                                   ↪ n=".. j)
                                                mergepolyline_AB_BC(
                                                   ↪ polylines[k],
                                                   ↪ polylines[j])
                                                table.remove(polylines, j)
                                                mergepolylines_AB_BC()
                                        end
                                end
                        end
                end

end

------------------------------------------------------------------ BA_BC
function comparepolyline_BA_BC(polyBA, polyBC, delta)
        if polyBA==nil or polyBC==nil then return false end
        local x = polyBA[1][1] - polyBC[1][1]
        local y = polyBA[2][1] - polyBC[2][1]
        return (delta >= sqrt( x*x + y*y))
end

function mergepolyline_BA_BC(polyBA, polyBC)
        polylineswitch(polyBA)
        mergepolyline_AB_BC(polyBA, polyBC)
end

function mergepolylines_BA_BC()

                for k,v in pairs(polylines) do
                        for j,v in pairs(polylines) do
                                if k~=j then
                                        if comparepolyline_BA_BC(polylines[k
                                           ↪ ], polylines[j],
                                           ↪ merge_tolerance) then
                                                print("merge BA-BC poly n="
                                                   ↪ .. k .. "   with poly
                                                   ↪ n=".. j)
                                                mergepolyline_BA_BC(
                                                   ↪ polylines[k],
                                                   ↪ polylines[j])
                                                table.remove(polylines, j)
                                                mergepolylines_BA_BC()
                                        end
                                end
                        end
                end
```

```
end

------------------------------------------------------------------ AB_CB
function comparepolyline_AB_CB(polyAB, polyCB, delta)
        if polyAB==nil or polyCB==nil then return false end
        local x = polyAB[1][#polyAB[1]] - polyCB[1][#polyCB[1]]
        local y = polyAB[2][#polyAB[2]] - polyCB[2][#polyCB[1]]
        return (delta >= sqrt( x*x + y*y))
end

function mergepolyline_AB_CB(polyAB, polyCB)
        polylineswitch(polyCB)
        mergepolyline_AB_BC(polyAB, polyCB)
end

function mergepolylines_AB_CB()

                for k,v in pairs(polylines) do
                        for j,v in pairs(polylines) do
                                if k~=j then
                                        if comparepolyline_AB_CB(polylines[k
                                           ↪ ], polylines[j],
                                           ↪ merge_tolerance) then
                                             print("merge AB-CB poly n="
                                                ↪ .. k .. "   with poly
                                                ↪ n=".. j)
                                             mergepolyline_AB_CB(
                                                ↪ polylines[k],
                                                ↪ polylines[j])
                                             table.remove(polylines, j)
                                             mergepolylines_AB_CB()
                                        end
                                end
                        end
                end

end

------------------------------------------------------------------ BA_CB
function comparepolyline_BA_CB(polyBA, polyCB, delta)
        if polyBA==nil or polyCB==nil then return false end
        local x = polyBA[1][1] - polyCB[1][#polyCB[1]]
        local y = polyBA[2][1] - polyCB[2][#polyCB[2]]
        return (delta >= sqrt( x*x + y*y))
end

function mergepolyline_BA_CB(polyBA, polyCB)
        polylineswitch(polyBA)
        polylineswitch(polyCB)
        mergepolyline_AB_BC(polyBA, polyCB)
end

function mergepolylines_BA_CB()

                for k,v in pairs(polylines) do
                        for j,v in pairs(polylines) do
                                if k~=j then
                                        if comparepolyline_BA_CB(polylines[k
                                           ↪ ], polylines[j],
                                           ↪ merge_tolerance) then
```

```
                                                print("merge BA_CB poly n="
                                                    ↪ .. k .. "   with poly
                                                    ↪ n=".. j)
                                                mergepolyline_BA_CB(
                                                    ↪ polylines[k],
                                                    ↪ polylines[j])
                                                table.remove(polylines, j)
                                                mergepolylines_BA_CB()
                                            end
                                        end
                                    end
                                end

end

-------------------------------------------------------------------------merge poly
function polylineswitch(polyAB)
        local polyBA = {}
        polyBA[1] ={}
        polyBA[2] ={}
        for k,v in pairs(polyAB[1]) do
                polyBA[1][#polyAB[1] - k +1] = polyAB[1][k]
                polyBA[2][#polyAB[2] - k +1] = polyAB[2][k]
        end

        for k,v in pairs(polyAB[1]) do
                polyAB[1][k] = polyBA[1][k]
                polyAB[2][k] = polyBA[2][k]
        end

end


function mergepolylines()
mergepolylines_AB_BC()
mergepolylines_BA_BC()
mergepolylines_AB_CB()
mergepolylines_BA_CB()
end


------------------------------------------------------------------------ gem
    ↪ file


function make_header(file_output)

        if fit_to_drawing then
        local xm =0
        local ym =0
                for k,v in pairs(polylines) do
                        for j,v in pairs(polylines[k][1]) do
                                if (xm < tonumber(polylines[k][1][j])) then
                                    ↪ xm = tonumber(polylines[k][1][j]) end
                                if (ym < tonumber(polylines[k][2][j])) then
                                    ↪ ym = tonumber(polylines[k][2][j]) end
                        end
                end
                xmax = xm
                ymax = ym
        end
```

```
        file_output:write ([[pa_define(]].. math.floor((xmax-xmin)/mesh_size
           ↪ ) ..[[, ]].. math.floor((ymax-ymin)/mesh_size) ..[[, 1,
           ↪ cylindrical,electrostatic,, ]].. mesh_size ..[[, ]]..
           ↪ mesh_size ..[[, ]].. mesh_size ..[[)
]])
end

function make_electrode(file_output,i,plylines)
        file_output:write ([[
        e(]].. i ..[[) {
                fill {
                        within {
                                polyline(
]])

        for j,v in pairs(plylines[1]) do
                --print(j,polylines[k][1][j],polylines[k][2][j])
        file_output:write ("\t\t\t" .. plylines[1][j] .. "," .. plylines[2][
           ↪ j] ..",\n")

        end

        file_output:write ([[
                                )
                        }
                }
        }
]])
end

function make_gem()

        file_output = assert(io.open(output_file, "w"))
        make_header(file_output)

        for k,v in pairs(polylines) do

                print("electrode n=".. k)

                make_electrode(file_output,k,polylines[k])

        end

        file_output:close()
end

function sort_poly()
         for k,v in pairs(polylines) do
                 for j,v in pairs(polylines) do
                        if compare_min(polylines[k],polylines[j]) then
                                polylines[k],polylines[j] = polylines[j],
                                   ↪ polylines[k]
                        end
                 end

         end


end
```

```
-------------------------------------------------------------------------------
   ↪ main

function readfile()
         print("\n\n")
         print("=============== Loading DXF ==============\n")
         file = assert(io.open(input_file, "r"))

        for line in file:lines() do
        if(line == "LWPOLYLINE") then readLWPOLYLINE() end
        end
         file:close()
        print("nb poly=",#polylines,"\n")

        --plotpolydetail()

        if merge then
        print("ni=".. #polylines)
        mergepolylines()
        print("nf=".. #polylines)
        end

        print("sorting polylines")
        sort_poly()

        print("make gem")
        make_gem()

        print("\n\nAu boulot!!!")
end

readfile()
```