How to 3D-print Complex Networks and Graphs

Raymond Aschheim
Polytopics
8 villa Haussmann
Issy, 92130, FRANCE
E-mail: raymond@aschheim.com

Abstract

Artists and scientists have offen the need to get 3D printed models of structures made of nodes and edges, like graphs, networks, skeletons. While 3D printing is now accessible to everyone just by uploading STL files to websites like shapeways or imaterialise, and get an instant validation and quote, the generation of suitable files from graph raw data is still not available through open tools. If the graph has some dozens of edges, it is easy to build the STL with some CSG tool by making the boolean union of cylinders and spheres. But for complex graphs with hundreds of edges, these tools are failing. An elegant solution was presented by George Hart during Bridges 2008 for sculptures with non intersecting struts. We explain here a method suitable when struts are intersecting. We draw in 3D our lines using voxels, then slice it into an image stack, and triangulate it using marching cube algorithm. Optimization steps are needed to minimize the number of faces (under 1M) and the file size.

1. Motivation

3D printing opens the field of artistic sculpture realization by its ability to generate structures with a higher complexity level that what a human hand can do. And the ability to do this in minutes or hours instead of year long 'chef d'oeuvre' (master-piece), and therefore to explore and refine the visual impact of the artwork by manipulating a real object.

Mathematical artists can find inspiration for these complex networks in various domains of maths and physics, like polytopes, graphs, higher dimensional lattices, fractal sets.
A general and fast method to procedurally generate files suitable to 3D printing is missing. We want to print the union of hundreds or thousands of little geometrical shapes, like polyhedron, struts, or spheres. Simply making the boolean union of all this objects with a CSG tool or library will do it. But the free available tools are well working for some douzens of objects but not for thousands.
Two other approaches are possible:
1)In a specific case where we want to print a graph, (our main goal), a very effective algorithm has been given in [1]. A polygon (e.g. a triangle) is choosen at a given distance of each node, along each strut. The convex hull of all this polygons and the node vertice is computed at each node. Then the two polygons ending any strut are replaced by their convex hull. When the struts dont intersect, this algorithm is optimal, as illustrated by many beautiful samples in [1], but if two struts intersect it dont give a 3D printable manifold.
2) The other approach has to handle cases where struts can intersect. It is based on the voxelisation of all elementary shapes of the structure. Then we make the union of all the voxels. Then we have to compute the boundary of our compact voxel cloud, as a list of square oriented faces. They form a watertight manifold. But it is very raugh and need to be smoothed by a Laplacian smooth. And finally the faces should be decimated to less then 1 Million.
We will observe and solve computational contraints for this approach.

General concepts and code of how to address the problem are given in [2], and we too will use Mathematica[3] (abbreviated *M*) to create procedurally our mathematical sculptures, and need a printing solution from the *M* environnement. We limit our available tools to *M*, whose hobbyist licence is available at an affordable price, and to free open source tools.

The **`RegionPlot3D`** function can be used to make the voxelisation of a procedurally defined structure, and to extract its boundary as a watertight manifold, but it is practically limited to resolutions of 200x200x200, so we have to solve the problem otherwise (e.g. for 800^3 resolution).

Since a very efficient marching cube algorithm [4] is available [5] to generate an OBJ file from a stack of images, thanks to the US National Health Institute, for medical applications like body scanner, complex mathematical graphs can now be 3D-printed through watertight STL files. We just have to generate voxels and slice them to images.

2. Our method

Edge and Vertices from Graph. We will study two cases: The geometrical skeleton, or wireframe, and the topological connected graph.
In both cases we have a list of nodes (numbered 1 to n) and a list of edges (expressed as n1→n2) :
Nodes={1,2,3..,n}, Edges={1→2, 1→3, 1→4, 2→3,2→4, 3→4} (* a tetrahedron where n=4 *)

In the first case we already have a list of tridimensional coordinates for each node:
Coordinates={{0,0,0},{1,0,0},{0,1,0},{0,0,1}} (* if the tetrahedron is cut from a cube *)

In the second case we have only the graph topology, expressed by the list of edges. We use the *M* [3] **`GraphPlot3D`** function to create the coordinates list by using a spring-embedded [6] algorithm. Nodes are first placed randomly in a 3D space, then a dynamic simulation of elastic strings at each edge where the strength growths with the string length is applied. This is fast converging to an equilibrium state where all the edges have about the same length. We get back the node positions.

Voxel lines. When we have all edges and node coordinates, we just have to trace lines between start and end nodes. We are not printing on a screen but in a tridimensional voxel array of side-length L choosen between 300 and 1000. At most we have an array of 1000x1000x1000 = 1 Giga-voxels, which may be stored in 1 gigabit. But the big part of it will be filled with empty voxels and bit set to zero. So we use a structure called **`SparseArray`**, whom size is growing like the number of non-empty voxels (the bits set to 1).

Our lines of voxels are generated by discretizing regularly spaced voxels as weighted sum of the beginning and end points. The lines are getting a thickness by translating them to neighbors in spheres around the extreme points.

Voxel slices. `ExportSlices[vx, name, scale]` cut all the needed slices of the cubic sparse array and save them as images in the directory **`baserep`** under the names **`name`**nnn.png, eventually using a **`scale`** factor.

Voxel triangulation, through marching cubes, with Fiji. `Fiji[name, scale*size]` runs Fiji to load the image stack, save a PGM file, and them save a WaveFront OBJ file with the triangulation of the stacked voxels using an efficient marching cube algorithm.

STL optimization, smoothing, face decimation, using Meshlab.

`Meshlab[name]` runs Meshlab [7] to make some finishing on the OBJ file and save it as a STL file. It reverses the face orientations (because Fiji generates a reversed orientation), and applies a Laplacian smooth. The script could also make face decimation when Fiji generates too many faces.

3. Results

Performance and execution time. For the sculpture presented at the *Bridges 2013* art exhibition, we choosed a resolution of 700 pixels and a cylinder radius of 3 voxels.

- To voxelise this graph to 3M voxels in a 700x700x700 sparsearray, our code took 3 minutes.
- Then, the writing of the 700 slice images took 5 minutes.
- The opening of these images and marching cube algorithm execution by Fiji took 3 minutes.
- Finally the smoothing in Meshlab took less than one minute.

The total process time was around 10 minutes (on a end-2010 MacBook Air 2.13Ghz core duo with 4Gb and a SSD, and Mountain Lion)

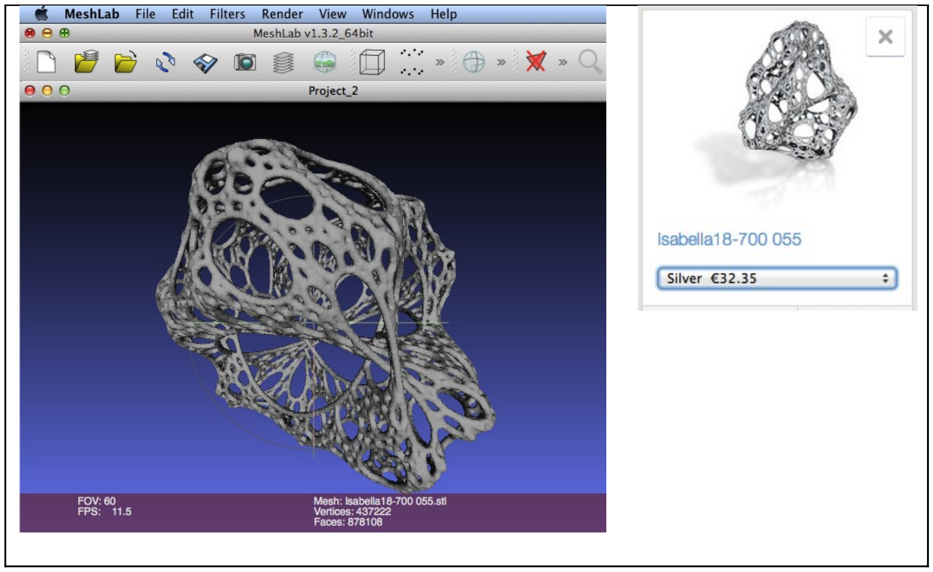

Figure 1: The resulting STL file viewed in Meshlab, and on shapeways.

Finally. In less than 15 minutes it is possible to transform a list of edges to a watertight-manifold STL file (see figure 1) of around 900K faces and 50Mb, upload it to an online 3D Printing website, and get a quote for its realization at a very reasonable price, less than $50 in sterling silver and less than $10 in nylon.

It would be better to improve resolution by a factor 8 (going to 4Gvoxels) or more, but the Fiji software saturates the heap memory. The process is conveniently fast, and still optimizable.

4. Running our own Marching Cube algorithm

This resolution improvement is possible by implementing our own boundary extraction instead of using Fiji. A good simplified marching cube algorithm is given in [2]. We have improved this **`boundary`** function to a **`boundaryvox`** adapted to voxels. The standard **`Export`** can generate STL file from the set of polygons, but we have to slice in files of less than 500000 faces otherwise the writing is too slow. Meshlab will post process it by loading and merging the slices, smooth with Laplacian, Decimate with

Quadric Edge Collapse Decimation, and save a file of around 50MB and 1Mfaces. The decimation introduces manifold defects that can be automatically repaired by using netfabb Basic [9] at the end.

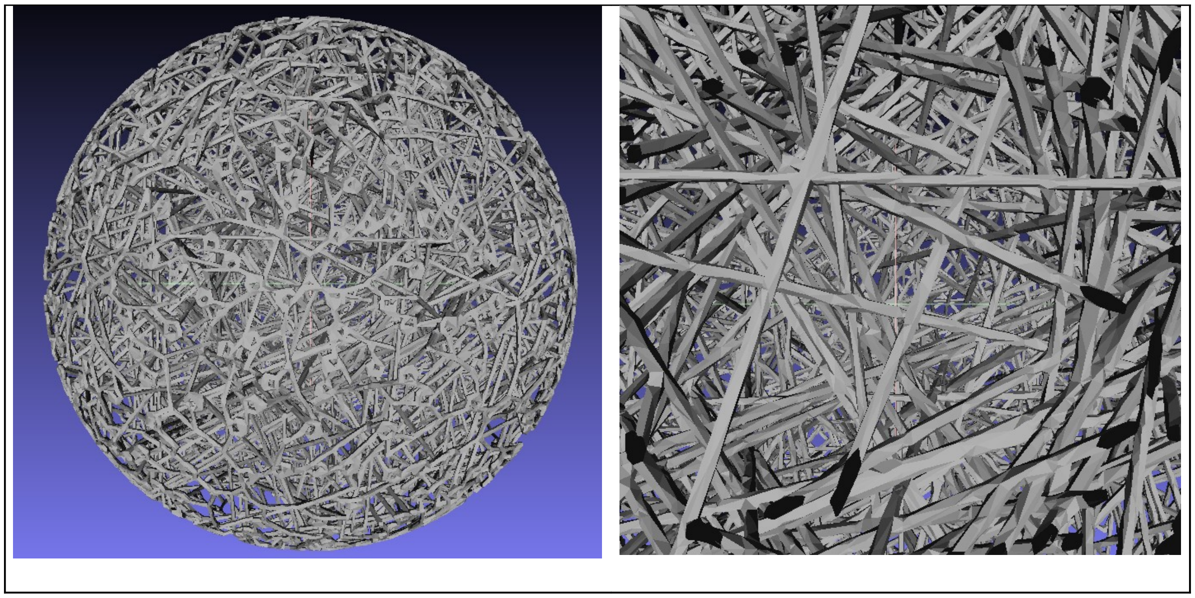

Figure 2: The resulting STL file viewed in Meshlab, and a detailed view of the inside.

5. Conclusion

The Voxel approach presented here is opening new ways for 3D printing of complex networks and graphs with hundreds to thousands of struts, possibly intersecting each other. Graphs with numerous intersecting (see Figure 2) struts are efficiently 3D-printed using this method, but they will have many millions of faces. An alternative way (work in progress) would be to use algorithm from [1] and remove the intersections between struts by replacing lines by zig-zags. The number of faces would be highly reduced by construction and would not need decimation.
My own artistic path inspired by Quantum Gravity needs this tools to go from theory to matter.

References

[1] G. Hart, *An Algorithm for Constructing 3D Struts*, Journal of Computer Science and Technology, 24:1, pp. 56-64. 2009.
[2] G. Hart, *Procedural Generation of Sculptural Forms*, Proceedings of the Bridges 2008 conference.
[3] S. Wolfram. The Mathematica book. Wolfram Media. 1988.
http://reference.wolfram.com/mathematica/ref/RegionPlot3D.html
[4] B. Lorensen, marchingcubes.org, *http://www.marchingcubes.org/index.php/Main_Page.*
[5] NIH.GOV, ImajeJ/Fiji, *http://fiji.sc/Fiji, http://imagej.nih.gov/ij/*
[6] T. Fruchterman and E. Reingold. *Graph drawing by force-directed placement. Software: Practice and Experience,* 21(11):1129–1164. 1991.
[7] Pisa University, meshlab, *http://meshlab.sourceforge.net/*
[8] I. Thiesen, *Networks as Manifolds*,
http://www.wolframscience.com/summerschool/resources/IsabellaThiesen.pdf .2011.
[9] NetFabb Basic Studio, free software, http://www.netfabb.com/basic.php
[10] R. Aschheim Graph3DPrint.nb available on the conference cd and on
http://www.polytopics.com/Bridges2013/Graph3DPrint.nb