Near-Constant Density Wireframe Meshes for 3D Printing

Ole Untzelmann
Visual Computing Institute
RWTH Aachen University
untzelmann@cs.rwth-aachen.de

Leif Kobbelt
Visual Computing Institute
RWTH Aachen University
kobbelt@cs.rwth-aachen.de

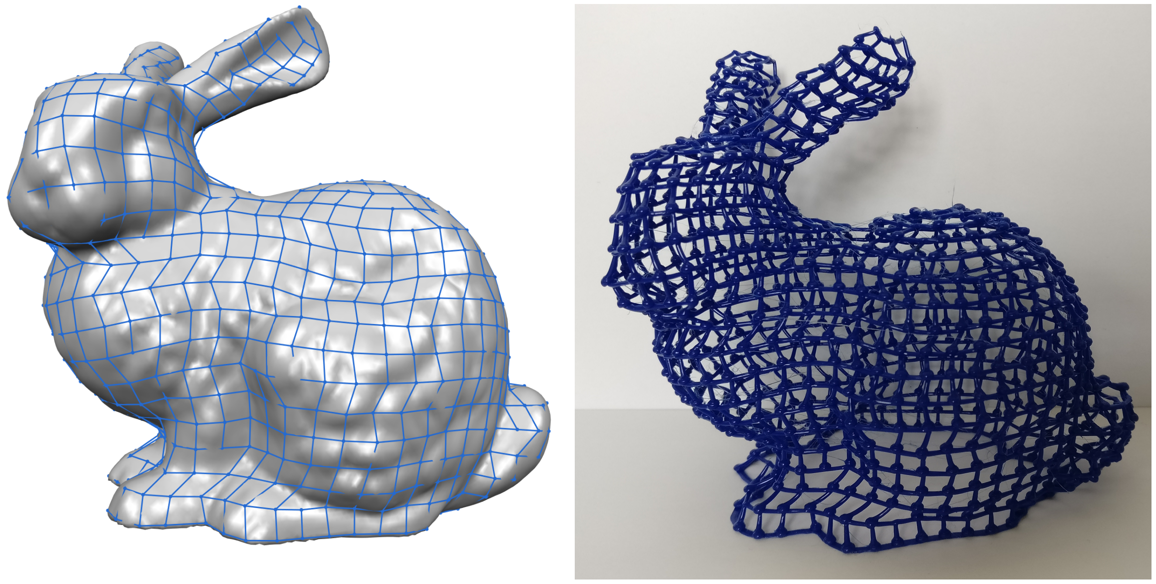

Figure 1: Wire mesh generated using the presented algorithm.

ABSTRACT

In fused deposition modeling (FDM) an object is usually constructed layer-by-layer. Using FDM 3D printers it is however also possible to extrude filament directly in 3D space. Using this technique, a wireframe version of an object can be created by directly printing the wireframe edges into 3D space. This way the print time can be reduced and significant material saving can be achieved.

This paper presents a technique for wireframe mesh generation with application in 3D printing. The proposed technique transforms triangle meshes into polygonal meshes, from which the edges can be printed to create the wiremesh. Furthermore, the method is able to generate near-constant density of lines, even in regions parallel to the build platform.

CCS CONCEPTS

•Computing methodologies →Mesh models; Mesh geometry models;

KEYWORDS

additive manufacturing, digital fabrication, rapid prototyping, FDM

ACM Reference format:
Ole Untzelmann and Leif Kobbelt. 2018. Near-Constant Density Wireframe Meshes for 3D Printing. In *Proceedings of Symposium on Computational Fabrication, Cambridge, MA, June 17 - 19, 2018,* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

There are several techniques for additive manufacturing, such as Selective Laser Sintering (SLS), Stereolithography (SLA), or Fused Deposition Modeling (FDM). In recent years a lot of affordable 3D printers, which use Fused Deposition Modeling (FDM), have become available on the market. Using FDM machines, it is possible to move the print head in 3D space while extruding material, which is not possible for most other techniques. As the printed objects are printed solid in most cases, slicing software typically makes no use of this possibility and prints models layer-by-layer. But it has been shown, that it is possible to print wireframe meshes, by directly moving the print head in 3D space [Mueller et al. 2014]. In this paper, the process of printing edges directly in 3D space is called spatial 3D printing.

The focus in this work is on the creation of wiremesh layouts, which are suitable for spatial 3D printing. Suitable means that the generated wire meshes can be printed without collisions, while retaining the most important geometric features of the input mesh. The contribution of this paper is a novel method for the generation of wiremesh layouts, which is able to generate contours with near-constant offset distance in regions parallel to the build platform. Based on iso-contours of a scalar field defined on the input mesh, our method tranforms the input triangle mesh into a polygonal T-mesh, from which the edges can be printed to create the wiremesh.

2 RELATED WORK

WirePrint

The first published approach of printing wireframe meshes directly in 3D space is WirePrint by Müller et al. [Mueller et al. 2014]. The input mesh is sliced by a set of planes parallel to the build plate. These planes are adaptively spaced to enhance model quality in regions with steep overhangs and to handle topology changes between layers. (i.e. when a single contours splits into multiple contours). The generated slices are connected using a zigzag pattern. For their test prints, a cooling system based on air jet was attached to the printer, which allows for very fast cooling of the material. This cooling is important for fast wiremesh prints, as the material solidifies faster.

On-the-fly print

With On-the-fly print, Peng et al. [Peng et al. 2016] presented a tool, which can create a wiremesh preview print during the design process. A 5DOF printer is used, which can change the printing direction during the print. The wire mesh layout is generated using a UV parametrization of the mesh surface. Iso-lines are extracted from the parametrization to generate the edges along one direction. Then points are sampled along the contour, which are connected to the points from the contour below, to generate the edges along the other direction. Their printer also has a custom cooling solution, using a pair of atomizing sprays.

Printing arbitrary meshes with a 5DOF Wireframe Printer

Another approach to wire mesh 3D printing was published by Wu et al. [Wu et al. 2016]. Instead of generating the wiremesh, polygonal meshes are used as input and all edges of the mesh are printed. A technique was developed, which is able to calculate an ordering of the edges, such that all edges can be printed without introducing collisions with the print head. Similar to Peng et al. [Peng et al. 2016] a 5DOF printer was used, which is able to adjust the printing direction. For calculating a valid ordering, a collision graph is constructed, which implies a partial ordering of the edges. The complete ordering is then created by successively adding edges to the result with respect to the collision graph, until all edges are added.

3 WIRE MESH GENERATION

Conceptually, our wiremeshes consists of (near-) horizontal contours that are connected by (near-) vertical pillars. When constructing a good wiremesh for spatial 3D printing, we have to observe a number of constraints and desiderata in order to maximize output quality and to minimize material consumption. The constraints mostly emerge from collision avoidance between the printing head and previously printed parts of the model while the desiderata cover aspects like shape approximation as well as uniform distribution and equal length of mesh edges.

We define the contours of our wiremesh as the iso-contours of a piecewise linear scalar function f given by its function values $f(v)$ at the vertices of the input mesh. In Section 3.1 we will explain how this function can be computed. Depending on the resolution of the input mesh, the resulting iso-contours can be of rather high resolution. In Section 3.2 we hence present a simple method to downsample the contours such that a prescribed minimum distance between neighboring vertices on the same contour is kept.

Finally, in order to construct the pillars between adjacent contours, we find a globally optimal set of connections between contours that are placed at geometrically relevant locations and avoid high valence nodes (see Section 3.3). The resulting wiremeshes are of high quality with the exception of regions around saddle points of the function f where larger gaps between adjacent contours can appear. Hence we describe a simple post-process in Section 3.4 that identifies saddle points and incorporates them into the wiremesh for gap closing.

3.1 Contour Field Generation

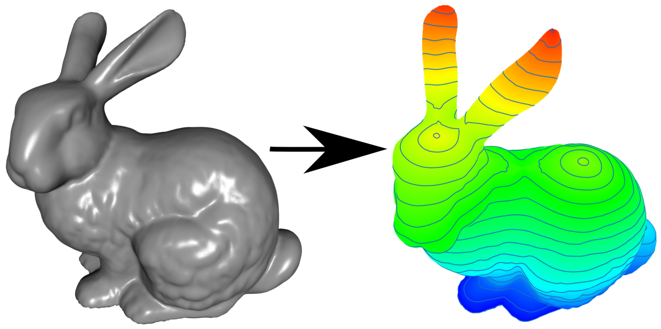

Figure 2: A contour field is generated on the surface of the mesh, which is used to slice the mesh.

In conventional FDM printing, the 3D model is sliced into planar horizontal slices. Müller et al. [Mueller et al. 2014] have adapted this slicing strategy to wiremesh contour generation by defining the slices as iso-contours of a piecewise linear scalar field f, with the z-coordinates of the input mesh vertices assigned as function values $f(v) = p(v)_z$, where $p(v)$ is the 3D position of the vertex v. For wiremesh printing, however, this simple contouring strategy is not well suited since the visible geodesic distance between adjacent contours depends on (one over the cosine of) the angle between surface normal and printing (gravity) direction leading to infeasible results in regions where the surface is nearly horizontal. Addressing this issue by adapting the vertical distance between contours Müller et al. [Mueller et al. 2014] solve the problem only for simple classes of shapes such as surfaces of revolution.

To create a function, which is better suited, a vector field $g(f) \in \mathbb{R}^3$ is assigned to the faces of the input mesh. Let $n(f)$ be the normal of face f. For initialization, all faces which are below a certain height h_z are assigned $g(f) = n(f) \times e_z \times n(f)$, where e_z is the unit z-vector. The vector field is then propagated to all other faces in the mesh, by rotating the vectors around the common edge. After each face has a vector assigned, the vector field is smoothed using a fixed number of averaging steps. In each iteration the vector are normalized. Furthermore, the vectors are rotated, such that the angle $\alpha = \angle(g(f), n(f) \times e_z \times n(f)) < \alpha_{max}$. This makes sure that later the resulting pillar edges are printable, as the maximal slope of generated slices is limited.

After the vector field has been calculated, it is integrated to create the scalar function $f(v) \in \mathbb{R}$, which is assigned to each vertex of the input mesh. For each triangle, the gradient of f can be expressed by a linear operator as

$$\nabla f = M \begin{pmatrix} f(v_i) \\ f(v_j) \\ f(v_k) \end{pmatrix},$$

where v_i, v_j and v_k are the vertices of the face. And M is defined as

$$M = (n \times e_{jk}, n \times e_{ki}, n \times e_{ij})/(2 * A),$$

where e_{ij} is the edge vector between the vertices v_i and v_j, and A is the area of the face.

The integral function f is calculated as the solution of the least squares problem

$$\begin{array}{ll} \text{minimize} & \sum_{s_i \in F}(\|k(s_i) * g(s_i) - \nabla f\|^2 + k(s_i)^2) \\ \text{subject to} & f(v_i) = p(v_i)_z \quad \forall v_i \in V \text{ with } p(v)_z < h_z \\ & k(s_i) \geq 1 \quad \forall s_i \in F \\ & f(v_i) \geq 0 \quad \forall v_i \in V \end{array}$$

Here we introduce an additional scaling factor $k(s_i)$ for each face s_i. We bound $k(s_i)$ from below, to guarantee an upper bound for the distance between contours.

3.2 Contour Sampling

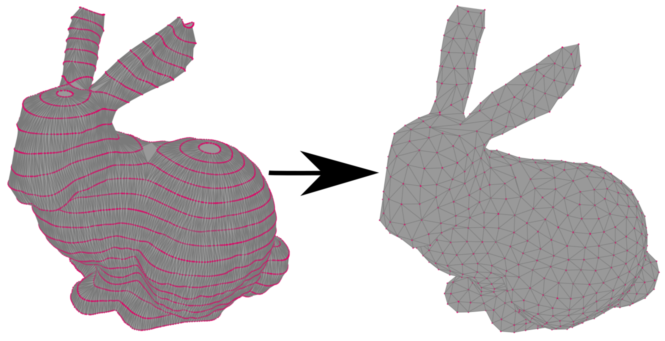

Figure 3: The vertices on each contour are sampled to obtain a set of collision-free printable vertices.

To generate the contours, the mesh is sliced uniformly along the iso-lines of the function $f(v)$. The distance of iso lines is controlled by a parameter h_{slice}. The edges on the slices are added to the mesh, by splitting the faces accordingly.

All vertices which do not belong to a countour are then removed from the mesh by using edge collapses. It has to be taken care, that there are no edges connecting two vertices on the same contour. After having removed the non-contour vertices, the edges between adjacent slices can be flipped to reduce the total edge length.

The contours may contain vertices which are too close to each other, to be printed without collisions. Therefore, a subset of vertices have to be selected, such that all vertices have at least the distance d_{min} to each other. On the other hand. if too many vertices are removed, important features of the mesh may be removed.

To select the vertices, the slices are processed individually from bottom to top. On each slice the binary labeling problem

$$\begin{array}{lll} \text{maximize} & \sum_{x_i \in V_s} w_i x_i & \\ \text{subject to} & (\sum_{x_j \in N_s(x_i)} x_j) \leq 1, & \forall x_i \in V_s \\ & x_i \in \{0,1\}, & \forall x_i \in V_s \end{array}$$

is solved. V_s is the set of all vertices on the slice s, and $N_s(v) = \{x \in V_s | \|x - v\|_2 < d_{min}\}$ is the set of all vertices on the same slice which are closer than d_{min}. w_i is the weight assigned to each vertex. The selected vertices should be located at regions of high curvature, such that important features of the mesh are retained. Also the vertices of adjacent slices should be aligned in their position. To incorporate both properties for selection, the weight w_i is chosen as $w_i = \frac{1.0+c_i}{l_i}$. c_i models the curvature of vertex i and is set to the angle between the the two adjacent contour edges. l_i is the length of the shortest edge to the slice below, which promotes coherent positions between vertices on adjacent slices. The vertices on the first slice have no edges to the slice below, in this case we set $l_i = 1$.

All vertices x_i which get a value of 1 are kept, the other vertices are removed by using edge collapses. After each iteration, edge flips are applied to reduce the total length of all pillar edges. The mesh now contains only the vertices which are used in the final wiremesh.

3.3 Pillar Selection

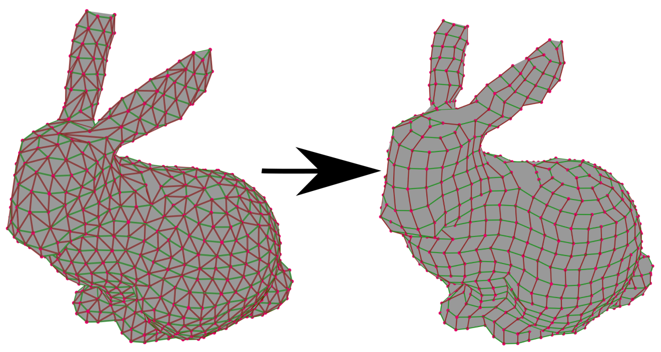

Figure 4: Pillar edges are selected, such that they can be printed without collision.

If multiple edges are connected to the same vertex, it may happen, that the print head destroys the edge which was printed first, because it collides with this edge, when printing the second one. Therefore, it is desirable that each vertex has at most one edge to the slice below. On the other hand each vertex needs to be supported by an edge from the slice below.

To avoid the risk of collision, our solution allows at most one edge from each vertex to the slice above and at most one edge to the slice below. Furthermore, the selected pillar edges should have minimal length. To select the edges, the binary labeling problem

$$\begin{aligned} \text{maximize} \quad & \sum_{e_i \in E} \frac{1}{l(e_i)} e_i + \sum_{v_j \in V} (1.0 + c_j) v_j \\ \text{subject to} \quad & \Big(\sum_{e_i \in E_{up}(v_j)} e_i \Big) \leq 1, && \forall v_j \in V \\ & \Big(\sum_{e_i \in E_{down}(v_j)} e_i \Big) = v_j, && \forall v_j \in V \\ & v_i \in \{0, 1\}, && \forall v_i \in V \\ & e_i \in \{0, 1\}, && \forall e_i \in E \end{aligned}$$

is solved. $l(e_i)$ is the length of edge e_i. c_j is the curvature of vertex v_j, as defined for the vertex sampling. $E_{down}(v)$ is the set of all adjacent edges of vertex v to the slice below and $E_{up}(v)$ is the set of all adjacent edges of vertex v to the slice above. All edges with $e_i = 1$ are kept in the final mesh, all other edges are removed. All vertices with $v_j = 0$ have no edge to support them. These vertices are moved to the center between the adjacent vertices with $v_j = 1$.

The objective function assures that vertices with large curvature and edges with short length are preferred. The constraints only allow that at most one edge to the slice above and one edge to the slice below are selected. Furthermore, the selected vertices must have an edge to the slice below.

3.4 Saddle Point Handling

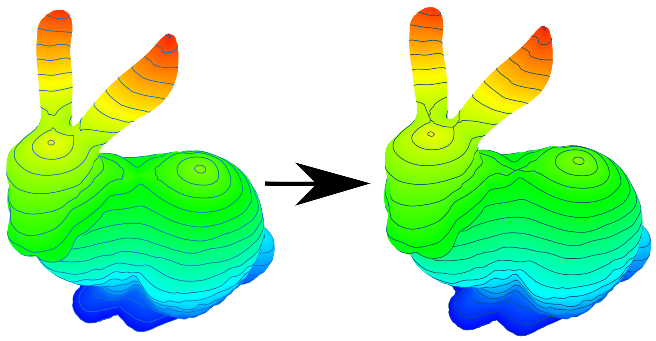

Figure 5: The contour field is optimized, such that iso-lines are close to the saddle points.

If a wire mesh is calculated using $f(v)$, the generated slices may have relatively large distances to each other in regions where a saddle point is located. To avoid this, the function values at saddle points should be very close to a value which is used for iso-line extraction. The saddle points are detected by counting the number of sign changes of $f(v_i) - f(v_j)$, where v_j are the vertices in the one-ring of the vertex v_i. If there are more than two sign changes, it is a saddle point. We introduce the function $q(v_i)$. For the saddle points this function contains the desired function values. For saddle points, $q(v_i)$ is set to be equal to the next larger value, which is used for iso-line extraction. The maximal distance between iso-lines is not violated if the function values are increased, as the generated iso-lines have a smaller distance to each other. The function $f(v)$ can then be calculated by modifying the least squares problem from Section 3.1

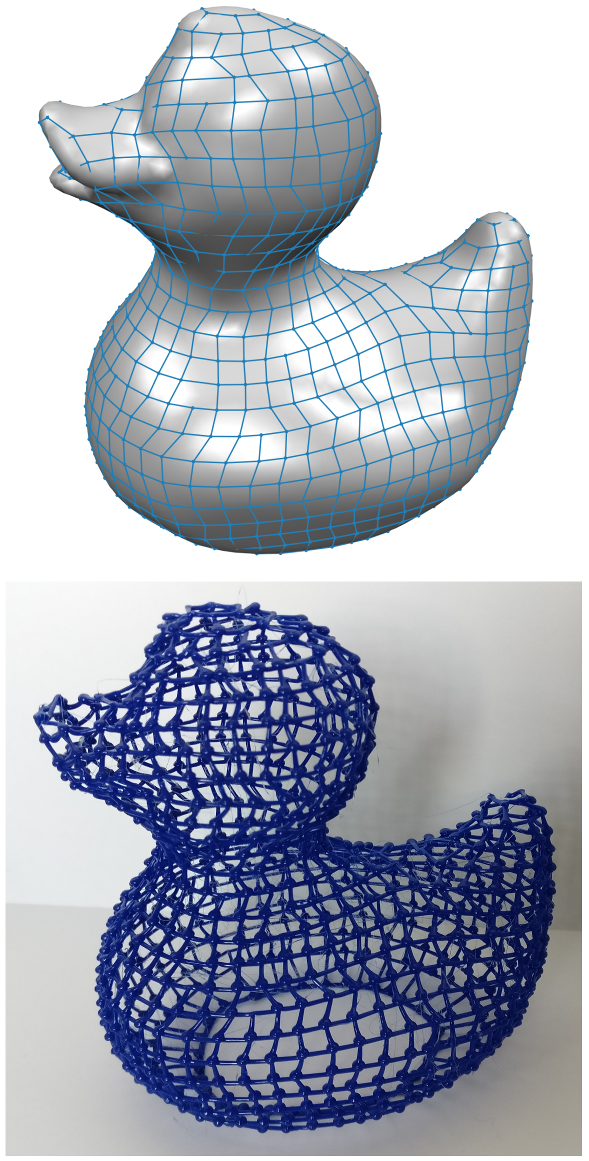

Figure 6: Preview and print of the rubber duck model.

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_{s_i \in F} (\|k(s_i) * g(s_i) - \nabla f\|^2 + k(s_i))^2 \\ \text{subject to} \quad & f(v_i) = q(v_i) - \epsilon \quad \forall v_i \in V \text{ with } v_i \text{ is saddle point} \\ & f(v_i) = p(v_i)_z \quad \forall v_i \in V \text{ with } p(v)_z < h_z \\ & k(s_i) \geq 1 \quad \forall s_i \in F \\ & f(v_i) \geq 0 \quad \forall v_i \in V \end{aligned}$$

The problem is the same as used for the contour field generation in Section 3.1, but with one additional constraint, which assures that the generated contours are close to the saddle points. But the saddle points to not lie on the contour, since this would cause numerical instabilities during the slicing process.

Overall, we first solve the problem without this additional constraint. Then the saddle point vertices can be determined and the problem can be solved with this additional constraint.

4 RESULTS

We demonstrate our technique by printing wire meshes of the Stanford Bunny model and the Rubber Duck model. Both models were printed at a scale of 10cm height. Resulting prints can be seen in Figure 1 and Figure 6.

The wire meshes were printed on a FDM delta printer, with nozzle diameter of 1mm. Our printer does not have a special cooling system, such as the printer of Müller et al. [Mueller et al. 2014]. Therefore, we were not able to reproduce their printing speeds.

After each pillar edge a delay of 5 seconds is added, to let the material solidify. Horizontal edges are printed without any delay. The Stanford Bunny was printed in 2h 33m, whereas the Rubber Duck model was printed in 3h 18m. With Cura as slicer for solid prints, using a layer height of 0.2mm, the same models could be printed in 3h 24m for the Stanford Bunny and 4h 01m for the Rubber Duck. Even though the delay after each printed edge is large for our printer, the printing time could be reduced by about 30%. The bigger advantage of wiremesh printing is material saving. Compared to the solid prints, both models consumed less than 10% of material.

5 CONCLUSION

A novel way of generating wiremeshes for spatial 3D printing was proposed. A function f was introduced, which can generate isolines with nearly constant distance to each other, even in regions parallel to the build platform. Furthermore a mesh transformation was proposed which takes a piecewise linear function on the vertices as input and generates an printable polygonal T-mesh from it.

In future research the use of a 5DOF 3D printer could be investigated. This would give more freedom in the orientation of the guiding vector field.

REFERENCES

Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, François Guimbretière, and Patrick Baudisch. 2014. WirePrint: 3D Printed Previews for Fast Prototyping. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14).* ACM, New York, NY, USA, 273–280. DOI: http://dx.doi.org/10.1145/2642918.2647359

Huaishu Peng, Rundong Wu, Steve Marschner, and François Guimbretière. 2016. On-The-Fly Print: Incremental Printing While Modelling. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16).* ACM, New York, NY, USA, 887–896. DOI: http://dx.doi.org/10.1145/2858036.2858106

Rundong Wu, Huaishu Peng, François Guimbretière, and Steve Marschner. 2016. Printing Arbitrary Meshes with a 5DOF Wireframe Printer. *ACM Trans. Graph.* 35, 4, Article 101 (July 2016), 9 pages. DOI: http://dx.doi.org/10.1145/2897824.2925966