Proceedings of TMCE 2014, May 19-23, 2014, Budapest, Hungary, Edited by I. Horváth, Z. Rusák
© Organizing Committee of TMCE 2014, ISBN 978-94-6186-177-1

GENERATION OF SUBDIVISION SURFACE FROM NETWORK OF CURVES

Zhihua Li
Institut Image-Le2i UMR CNRS 6306
Arts et Metiers ParisTech
School of Mechanical Engineering
Shanghai Jiao Tong University
France/China
2012-2353@ensam.eu

Ruding Lou
Institut Image-Le2i UMR CNRS 6306
Arts et Metiers ParisTech
France
Ruding.Lou@ensam.eu

ABSTRACT

Subdivision surfaces are usually used to construct freeform surfaces from network of curves for its ability and flexibility to deal with complex wireframes. In freeform surface designing, the designers usually draw at first some curves for describing the models conceived in their mind which form a curve network representing an object of arbitrary topology. Then 3D surfaces are computed to interpolate these curves in order to create a B-Rep model. If the subdivision surface is used in the workflow, its control polyhedrons generation from curves polygons could be a time-consuming stage. In this article, we develop an approach to generate automatically a control polyhedral mesh from an arbitrary topological curve network. One of common problems in interpolating surface patch using subdivision surfaces is how to determine the connectivity of control points. Arbitrary topological curve network has no restriction in topology structure, so another problem is that it has more ambiguousness in defining surface patches. There are three steps in our approach. Firstly, we compute a 1D mesh (a unique polygonal model) from curves. Secondly, we identify on the polygon different cycles that would be the boundaries of potential surface patches. Finally, in each identified cycle we apply an algorithm of quadrangulation to construct the control mesh of subdivision.

KEYWORDS

Curve network, curve control polygon, subdivision surface control polyhedron, quadrilateral mesh, arbitrary topological

1. INTRODUCTION

Freeform surface, or freeform surfacing, is used in CAD and other computer graphics software to describe the skin of a 3D geometric element. Freeform surfaces do not have rigid dimensions, unlike regular surfaces such as planes, cylinders and conic surfaces. A common task in freeform surface modeling is interpolating a given network of smooth curves by a smooth surface.

Subdivision surfaces [1] are usually used to construct freeform surfaces from network of curves for its ability and flexibility to deal with complex wireframes. A typical solution is based on associating curves network which defines the surface with boundary polygon of subdivision surface. In this article, boundary polygon is the control polygon of input curves. Hence, the input curve network is associated with a polygon network, and curve interpolation conditions are translated into conditions on the boundary polygons. Using these conditions, control polyhedral mesh can be constructed to generate subdivision surface. Then fairing techniques [2, 3] can be used to optimizing control polyhedral mesh satisfying the boundary conditions. Therefore, by using subdivision surface curve interpolating can be carried out for given networks of arbitrary topology (see [4]).

In surface design, designers usually draw at first some curves describing the models according to their imagination which form a curve network of arbitrary topology. Then boundary polygons associating with the drawn curves are used to create control polyhedral meshes. The subdivision surface generated later on from the control mesh would interpolate these initial curves. The process can be seen in the following workflow illustrated in figure 1. However, in many domains of creative design (ex. jewellery design) the stage of control polyhedral mesh generation from curves stays time-consuming.

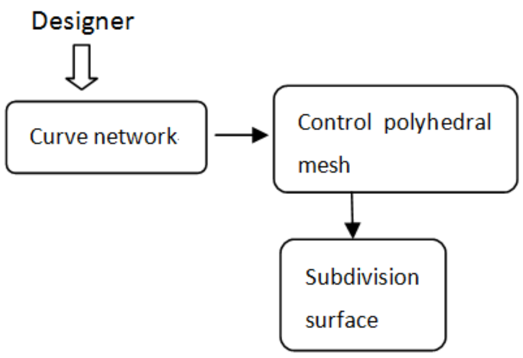


Figure 1 Workflow of freeform surface creation from curve drawing.

In order to improve the workflow and enhance the product design efficiency, the current paper proposes a computational approach to automate the generation of control polyhedral mesh from curves. This approach has been prototyped in java and it consists of four steps: firstly, execute curve treatments to associate the curves as well as their polygons; secondly, identify face cycles from polygon network; thirdly, generate polyhedral mesh in each cycle; finally, apply fairing to optimize the mesh quality. The prototyped operator has been experimented on various examples and the results are very promising. The paper is organized as follows. Section 2 discusses about the existing works relative to our objective. An overview on our approach is presented in the Section 3. Section 4 details how to generate a well-connected polygon network from the input curves. Section 5 illustrated the way to identify all cycles in polygons. Section 6 present how each identified cycle is filled by quadrangle meshes. Finally we show and discuss about some experimentation results of our approach in section 7 and section 8 presents our conclusions.

2. RELATED WORK

Some previous work in 3D modeling used input of 2D sketch, which is also called sketch-based modeling. Modeling by sketching has a fundamental problem of interpreting the 2D sketches to 3D curves as well as 3D surfaces behind.

Koel et al [5] addressed the problem of creating a freeform surface from a network of curves which were generated from 2D sketches. They introduced a method to find a 3D curve network from 2D sketches where the projection of the 3D curves match well the input 2D sketches. This method does not give a unique 3D curve network therefore the one which has minimum curvature among all the solutions is chosen. Then subdivision surface was generated to interpolate these 3D space curves having minimum curvature. However, the presented method of 3D mesh generation may not be feasible when the 3D curves are drawn directly by designers. Since in 3D space, the curves drawn by designers may not intersect correctly.

Gonen and Akleman [6] introduced a method for sketching 3D models in arbitrary topology. Their method converted 2D sketches to 3D meshes that mostly consists of quadrilaterals and then generated subdivision surfaces using Catmull-Clark scheme. But their algorithm can only deal with simple sketches where there is no intersection between any curves.

Our problem is a little different from sketch-based modeling because the input in this paper is 3D curves network instead of 2D sketches.

However, most of previous works in subdivision surface generation using 3D curves have focused either on how to fill a specified surface patch and maintain higher smoothness of refined surface, or on developing new subdivision approaches for lofting.

Doo and Sabin [7] focused on the behavior of recursively divided surfaces near extraordinary points. The behavior of the limits surface defined by recursive divisions can be analyzed in terms of the eigenvalues of a set of matrices. This analysis predicts effects actually observed, and leads to suggestions for the further improvement of subdivision surface smoothness.

Nasri [4] has developed subdivision methods for lofting based on the concept of a polygonal complex. A method that extends the capability of the recursive subdivision technique to generate surfaces that interpolate predefined curves is described. The technique consists of a one-step division of the initial polygon network and a topological modification of the face vertices generated from the edges and vertices of the given control polygon.

Levin [8] presented a subdivision algorithm for the computation and representation of a smooth surface of arbitrary topology that interpolates a given network of smooth curves. While subdivision schemes operate on a given mesh and generate a new mesh, his subdivision schemes generate the new mesh taking into consideration additional conditions - such as boundary conditions, and transfinite interpolation conditions - that are prescribed on the limit surface. But their algorithm is restricted to nets of curves where no more than two curves intersect at one point, which is a considerable restriction for many applications.

While there has been considerable success with these approaches in interpolating a single surface patch from curve network, there isn't much work focusing on generation of control meshes from network of curves.

Schaefer et al [3] described a method to solve the problem of constructing a surface using curve network. They raised an algorithm to determine the topological structure of the control mesh and generate the subdivision surface automatically in specified surface patch. The method automatically quadrangulates cycles formed by the network of polygons and then fair the resulting mesh. However, the quality of control mesh generated using his algorithm was dependent on control points' distribution to some extent. Method for identification the cycles of network was not mentioned.

The existing works doesn't include a framework to interpolate a curve network using subdivision surface, nor generate a polyhedral control mesh from curve network, which is what we want to focus on in this paper. With existing works even few methods is able to treat the cases where the curves don't form neither a connected network nor closed cycles. An example of open curves is shown in figure 14, it is easy to understand that the pretreatment of the curves is quite important for the following meshing stage.

Lou et al [9] proposed a method for merging Finite Element meshes. An algorithm was developed to search the created holes after removing the intersecting faces. Since the polygons network was not closed, their algorithm included the addition of some edges in order to form face loops. However, the problem of identification of the cycles in the article is easier because the face loops in this case have more topological constraint.

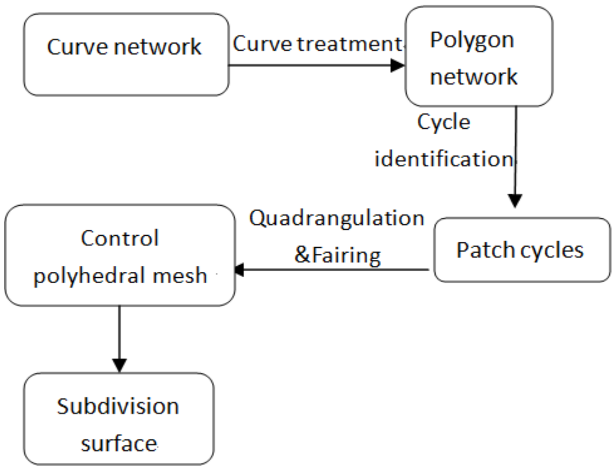


Figure 2 Subdivision surface design process with automatic generation of control polyhedral mesh

3. OVERVIEW OF APPROACH

Generally speaking, a designer draws curves in a CAD system. These curves will be the input of our operator proposed in the current paper. Our operator will then provide to the designer the control quadrilateral mesh, with which the designer could obtain smooth subdivision surfaces interpolating the initial curves. The subdivision surface design process using our approach can be seen in Figure 2. This process is also illustrated on an example for designing a ring (fig.3). As input of our operator the user draws the curves to describe the shape they want to obtain (fig.3.1).

- At first stage our operator does some treatments on the curve network in order to generate a well-connected control polygon network behind the curves (fig.1.2).
- Then a set of elementary cycles are identified in the polygon network using a graph algorithm and they will be the boundaries of surface patches (fig.1.3).
- At end quadrilateral meshes are generated in each cycle (fig.1.4).This will be the control polygon mesh for generating subdivision surface.

Using an existing subdivision scheme the user could obtain the smooth surface (fig.1.5) that interpolates the initial curves he/she has drawn at beginning. In compare with the previous works which focus on a certain part of interpolation of curves, our approach proposes a complete process to answer the needs of generating the initial polyhedral mesh for subdivision. This framework improves the time-costing manual generation of initial mesh, compensates the lack of

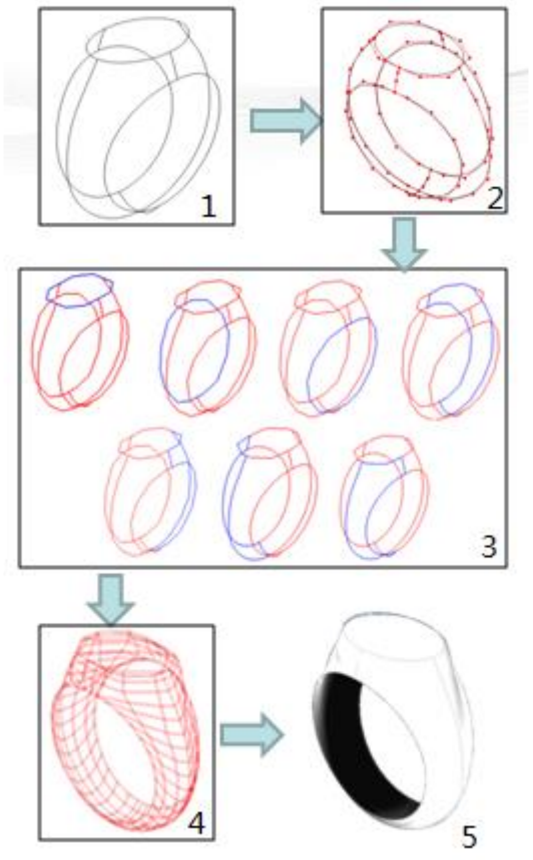


Figure 3 Example, subdivision surface design process

research in generation of initial mesh from arbitrary topology networks.

4. CONSTRUCTION OF POLYGONS

Being given as input a network of 3D curves the first stage is to compute the network of control polygons. This polygon network should be well connected from which the control polyhedral mesh will be generated later on. In addition the smooth surface generated by subdividing the control polyhedral mesh should interpolate the initial curves. Therefore the polygon network should not be the discretization of the curves but an association of all control polygons of curves. The figure 4 is a zoom of the picture in figure.3.2. The curves are drawn in black color and the control polygons (vertices and edges) are drawn in red color.

However, the polygon network obtained directly from the curve network can't be used to generate the polyhedral mesh due to following 4 aspects.

1) Curves may intersect each other while their control polygons may not intersect. For example in figure 5.a the two curves *A* and *B* intersect whereas their control polygons *A'* and *B'* do not interest each other.

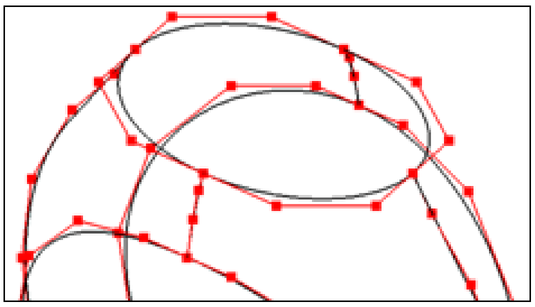


Figure 4 Construction of polygons from curves network of a ring model

2) It can happen also that the intersection points of the curves and polygons are not at the same position. In figure 5.c the two curves *A* and *B* intersect at point *O* but their control polygons *A'* and *B'* intersect at a different point *O'*.

3) Sometimes designers draw several curves which form an entire curve (fig.6.a). In this case the points generated on the polygons may not well distribute for further generated control mesh (fig. 6.b).

4) The density of control points could be very high (fig.7.a) whereas the density of the subdivision control mesh should be relatively low.

The first two points lead to incoherence between the intersections present in the curve network and the ones existing among the control polygons. To solve this problem we split the curves at intersection point in order to impose a common intersection point both on the curves and on their control polygons. For the third aspect the piecewise curves should be merged (fig.6.d) in order to generating control mesh of good quality (fig.6.e). Concerning to the fourth aspect the high density of nodes would also penalize the following mesh generation stage in terms of computation time. Therefore it is necessary to simplify the curve control polygons (fig.7.b).

In sum three kinds of curve treatments are necessary and detailed in the following paragraphs: splitting curves, merging curves and adjusting density of control points.

4.1. Curve splitting

Some curves need to be split at the intersection point in order to get a well-connected control polygon network.

In some cases, while input curves intersect, their corresponding control polygons may not intersect.

For example in figure 5.a, the curves *A* and *B* are intersecting whereas their control polygons *A'* and *B'* do not intersect each other. In other cases, curves intersect each other but their control polygons intersect at a different position. Typically in the example shown in figure 5.c where the two curves *A* and *B* intersect at the point *O* but their control polygons intersect at *O'*. The two intersection points *O* and *O'*, do not have the same position which will cause the fact that future subdivision surface will not correctly interpolate the initial curves.

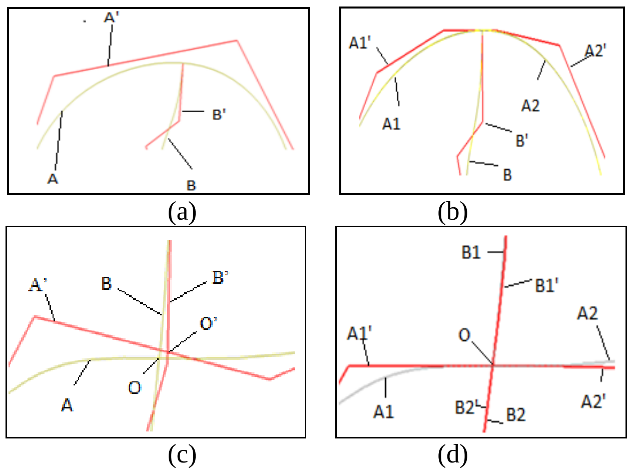


Figure 5 Splitting curves; (a) Curves intersect while their polygons don't intersect; (b) Split the curves in (a); (c) Polygons intersect at a different position with the curves' intersection; (d) Split the curves in (c).

In order to avoid the different intersection points on curves and polygons, these curves should to be split. Therefore the intersection points become endpoints where the curves and their control polygons end at the same point. Splitting curves is in fact inserting knots [10] into the curves to separate the curve into two. Knot insertion is adding a new knot into the existing knot vector without changing the shape of the curve. The inserted knot should lie exactly at the position of the intersection point of curves so the original curves can be considered as split into two.

Coming back to the introduced example (fig. 5.a), the initial curve *A* is split into two curves *A1* and *A2* therefore the three curves (*A1*, *A2* and *B*) meet at the intersection point where their polygons (A1', A2' and B') also intersect (fig.5.b). For the other example (fig.5.c), the two initial curves *A* and *B* are split into four curves *A1*, *A2*, *B1* and *B2* intersecting at point *O* (fig.5.d), so that their control polygons (*A1'*, *A2'*, *B1'* and *B2'*) intersect as well at the point *O*.

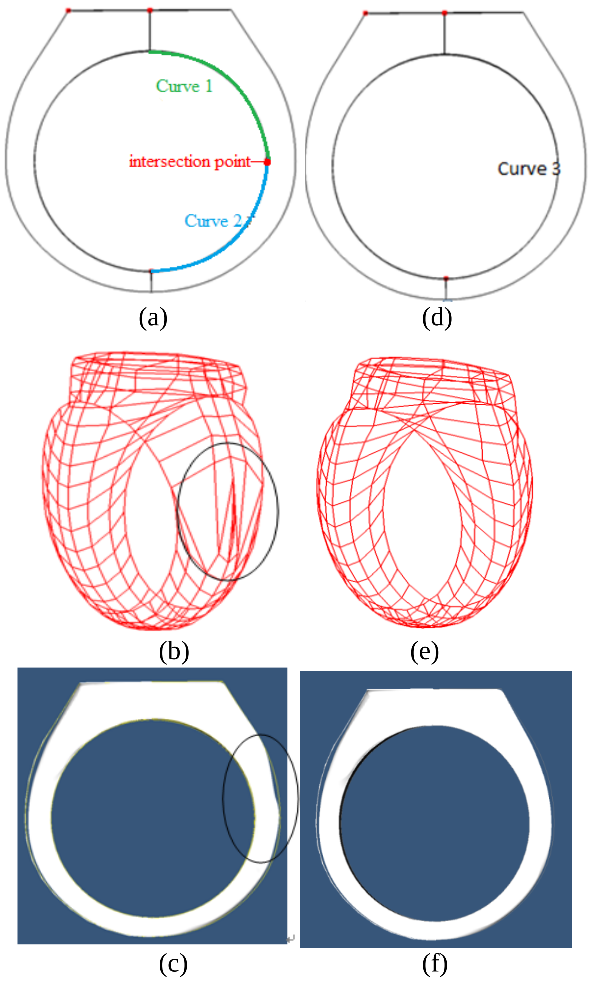


Figure 6 The ring model. (a) Two elementary curves that form an entire curve; (b) bad quality of generated control mesh; (c) bad shape of subdivision surface; (d) A unique curve is created by merging two elementary curves; (e) Mesh generated after merging curves; (f) Subdivision surface generated after merging curves.

4.2. Curve merging

In some cases, several elementary curves are connected successively to form an entire smooth curve. These piecewise curves might be produced due to default of design. These curves introduce extra intersection points, which not only burden the mesh generation process, but also harm the quality of the generated control mesh.

In figure 6.a, there are two curves (Curve 1 and Curve 2) that form actually an entire curve. The generated control polyhedral mesh and smooth subdivision surface created from this curve network can be seen in figures 6.b and 6.c. In the circled area, the quadrilaterals are irregular (fig.6.b) and the subdivision surface is sinking (fig.6.c). This is because these points connecting the two elementary curves will constrain the generation of control mesh and subdivision surface.

To solve this problem, we firstly define the criteria to search all the curves that need to be merged. Two intersecting curves are merged into one curve if the three conditions below are true:

- their extremities are enough close to the intersection point;
- their tangents on these extremities are approximately equal;
- their curvature are almost equal;

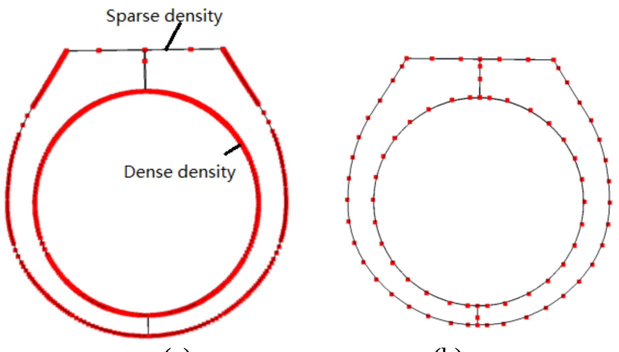
For sure to evaluate these conditions three thresholds should be chosen correctly. Concerning the example in figure 6.a, Curve 1 and Curve 2 are merged into Curve 3 (fig. 6.d).

Merging curves is in fact a process of knot removal [10] which is opposite to the knot insertion used in 4.1. The knots to remove should be the extra intersection points that separate an entire curve into different parts. Our operator is able to detect this kind of curves and to merge them. The result of implementation after merging the curves is more satisfying, with all the quadrilaterals well arranged (fig.6.e), no sinking problem in the generated surface (fig. 6.f).

4.3. Control point density adjustment

Besides knot insertion and knot removal to treat the input curves, the control point density of a curve should be also adjusted while preserving its shape. Here, the density means the ratio between the number of control points of the curve and the curve length.

It can be imagined that with a model where some curves have a high control point density and others have a low one, the inhomogeneity of different curves' control point density causes difficulty in generating polyhedral mesh. In Figure 7(a), the red color implies the control points. It can be seen that curves in the ring model are of different density. For the top part, the curves have very few control points whereas for the curves at lower part their control points much more dense. The very different densities of the control points will cause later on skinning mesh elements in the subdivision control polyhedrons. In addition the control polygon mesh for a subdivision surface is generally not very dense. Therefore what would be respected is more like the model shown in Figure 7(b), where all the control points of all curves have similar and low densities.



(a) (b)

Figure 7 (a) Ring model with inhomogeneous density control points; (b) Ring model after adjusting the density.

In adjusting the control point density, both efficiency and precision of shape approximation should be taken in consideration. It's sure that a model of high control point density can result a better precision of interpolation. However, the searching of face cycles and mesh generation will become much more time-consuming. Therefore, an algorithm is required to adjust density of control points in order to compromise between efficiency and precision.

A preliminary algorithm is used here to adjust the density of control points.

If the length of the shortest input curve is L_0, let the number of control point for this shortest curve be n_0, which is set by user. In case of the ring model (fig. 7.b) in this paper, n_0 is set as 4, which means there are three edges in the polygon created from the shortest curve. The control point numbers on other curves are proportional to their lengths. Hence, for the rest of the curves, their lengths are denoted as L_i. Then the number of control points for the rest curves can be expressed in the following equation:

$$n_i = n_0 \times C \times \frac{L_i}{L_0} \qquad (1)$$

And we take integer n_i as the number of control points for the i^{th} curve. C is the model constant that depends on the different requirement of precisions or efficiency.

5. IDENTIFICATION OF FACE CYCLES

5.1. Pre-computation of cycles

In this section, face cycles or cycle basis [11] are computed in the polygon network.

Most previous work in face cycles computing has focused on two dimensional drawings of wireframe models, sometimes with holes or non-planar faces. For non-manifold surfaces from two-dimensional inputs, Shpitatni and Lipson [12] described a method for identifying the faces of a wireframe object depicted by a single two-dimensional projection. However, the problem they dealt with is significantly different from ours because their input is strictly two-dimensional, rather than three-dimensional, which introduces more ambiguities. The most relevant is probably proposed by Fatemeh and Nina [13]. They developed a simple and robust framework for extracting patches from 3D wireframe models. In this article, an algorithm that is similar with Depth-First-Search [14] is developed.

Starting with an arbitrary edge and an ending point of this edge as starting point, our cycle detection is a process of searching next vertex continually until it finds the starting point. During this process, it's not allowed to use a point more than once. An example of cycle searching is shown in figure 8. Starting with the vertex no. 1 and the edge connecting the two vertices no. 1 and no. 2 (fig.8.a), four possible paths are identified (fig.8.b). Actually only two paths no. 1 and no. 3 allow to reach the starting point (vertex no. 1) without going through more than once the other vertices. Whereas the paths no. 2 and no. 4 are abandoned because they meet the vertex no. 3 twice before reaching the starting point. It can be seen that more than one cycle would be detected starting from a vertex, one of which is the wanted face cycle.

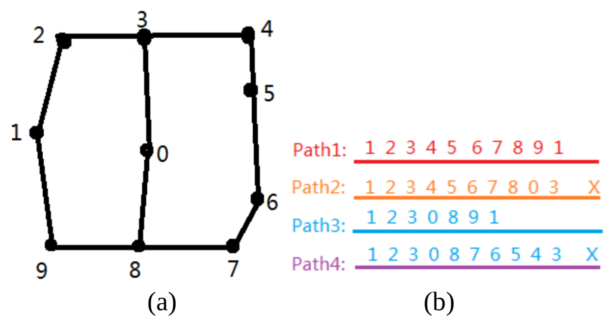

2
3
4
5
1
0
6
9
8
7
Path1: 1 2 3 4 5 6 7 8 9 1
Path2: 1 2 3 4 5 6 7 8 0 3 X
Path3: 1 2 3 0 8 9 1
Path4: 1 2 3 0 8 7 6 5 4 3 X
(a)
(b)

Figure 8 (a) Starting from a vertex and a direction, search all the possible paths in a graph; (b) All the possible paths;

5.2. Selection of face cycles

Beginning with a vertex, we can find a series of cycles. However, not all these cycles are the face cycles. Therefore a weighting function is used here to define these face cycles, or the shortest cycles. Three factors are used in defining face cycles, which are list in the following:

- Number of edges in the cycle (*n*);
- Number of connected components after deleting edges in the cycle (*m*);
- Bounding box volume of cycle;

Using these factors, the weighting function k is defined as:

$$k = n + n(m-1) \qquad (2)$$

The cycle with minimal value of *k* is selected as the shortest cycle. Usually, cycles that separate the curves into different components should be avoided. That's why there is a term of *n(m-1)* in the formula. If there is more than one cycle having the same minimal value for *k*, then the cycles with smallest bounding box volume are preferred. This selected cycle is then added into the set of validated cycles that will be used to generate mesh.

5.3. Exhaustive identification of cycles

It can be noted that if all the face cycles are found and their corresponding faces form closed solid, all the edges in the graph should be used in exactly two cycles. Therefore the two steps (5.1 and 5.2) should be repeated till all the edges appear exactly in two cycles. An algorithm below could summarize the general process of cycle identification. Initially the set *freeEdges* contains all edges from which cycles should be identified. At the end this set should be empty and the set *validatedCyles* should contain all the cycles.

Algorithm. exhaustive identification of cycles

```
While freeEdges is not empty
    Pre-compute cycles in freeEdges using a random edge
    Select one cycle from pre-computed cycles
    Add the cycle into the set of validatedCycles
    Foreach edge in freeEdges
        If edge appear in two validated cycles
            Remove edge from freeEdges
        End If
    End Foreach
End While
```

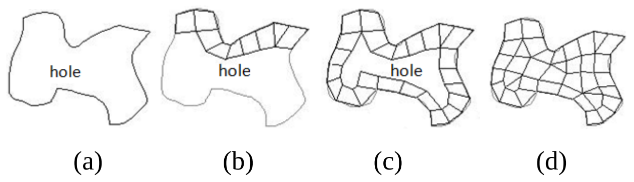


Figure 9 An advancing front method to fill the cycle

6. GENERATION OF CONTROL MESH

6.1. Cycle quadrangulation

The polygon network is constructed from curves (section 4) and different face cycles are identified (section 5). In this section, we will present how to generate polyhedral mesh in each cycle. To complete the prototype of our global approach, we have implemented the advancing front meshing algorithm presented in [3] for quadrangulating each cycle. Generally speaking, a cycle is considered as a hole (fig.9.a) and the algorithm will fill the hole iteratively from the boundary to the center (fig.9). At the first iteration the first mesh elements are created on the boundary (fig.9.b) and the hole becomes smaller. In the next iteration we generate other mesh elements on the new boundary of the smaller hole (fig.9.c). The filling process is repeated until the hole is completely filling (fig.9.d). During this process, the connectivity of mesh is determined without calculating the positions of each vertex.

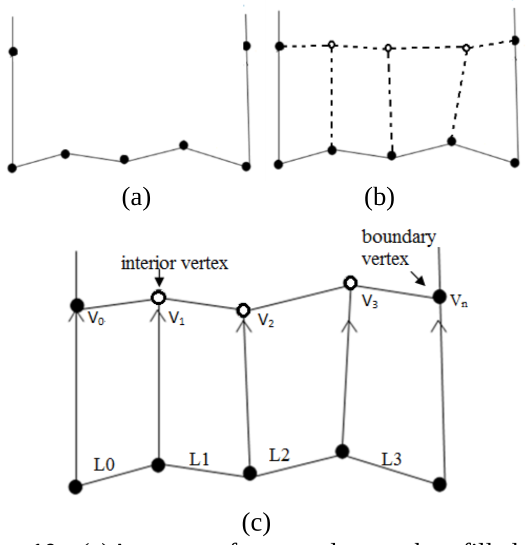


Figure 10 (a)A part of a cycle to be filled; (b) Connectivity constructed in the cycle; (c) Calculation of positions of interior vertices. Solid points are the boundary vertices and hollow points are the interior vertices

For a control mesh, there are boundary vertices and interior vertices. While boundary vertices are the original vertices of the control polygons, interior vertices are newly generated vertices inside each cycle. In figure 10 the solid points are the vertices on the hole boundary whereas the hollow points are the one newly generated in each iteration. The positions of the newly generated interior vertices are not computed for each iteration in the method proposed in [3] during the meshing process. Therefore we propose here a formula (eq. 3) to calculate the preliminary positions of the newly generated vertices based on the boundary vertices.

$$v_i = \left(1 - \frac{\sum_{k=0}^{i-1} L_k}{\sum_{k=0}^{n-1} L_k}\right) v_0 + \frac{\sum_{k=0}^{i-1} L_k}{\sum_{k=0}^{n-1} L_k} v_n \qquad (3)$$

Where the vector v_i starts from a boundary vertex and ends at an interior vertex; v_0, v_n are two vectors, formed by boundary vertices connected to the chosen polyline(using the method in 6.1) and extreme vertices of the chosen polyline; L_i is the length of the ith edge of the boundary polygon (fig.10).

6.2. Mesh faring

In this part we present how to improve the quality of the generated mesh so that its infinitely subdivided surface interpolates the curve network and has a smooth appearance, which is called mesh fairing in this article. It is actually an optimization problem where the objective function is defined by an energy quantity computed from the mesh. The optimal positions of mesh nodes allow us to minimize the energy quantity.

Using the preliminary positions computed by equation 3 the control mesh generated in the model of ring is shown in figure 11.a. However, the quality of generated quadrilaterals is not optimal. For example, in the circled part (fig. 11.b) the mesh elements could be improved as those shown in figure 11.c. Usually, fairing computes the positions of the interior vertices where an energy function [15 and 16] is minimized. Therefore the shape of quadrilaterals in the mesh is more regular, with each angle of the quadrangle more close to 90° (fig. 11.c).

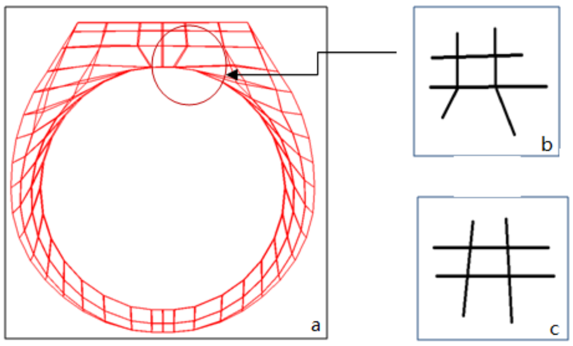


Figure 11 Control polyhedral for the ring model. (a) Preliminary mesh; (b)Quadrilaterals of bad quality; (c) Preferred quadrilaterals.

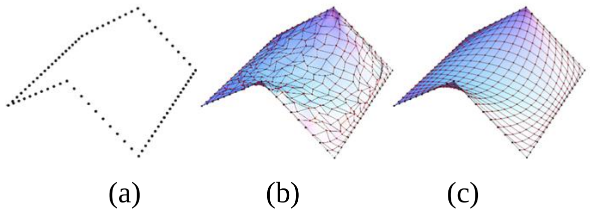


Figure 12 Mesh fairing example. (a) Boundary vertices of a cycle; (b) The preliminary generated mesh; (c) Mesh after fairing

The mesh fairing process actually is considered as an optimization problem. The mesh nodes are repositioned in order to minimize a certain quantity. The border vertices positions should not be changed since they inherit the positions of the initial curve control points. Whereas the interior vertices could be repositioned in order to improve the mesh quality. So we can formulate the optimization problem:

$$\text{Min } E(p) \quad \text{where } p = (x_0, y_0, z_0, \ldots, x_n, y_n, z_n)$$

$E(p)$ is an objective function defining a kind of energy relative to the interior vertices position vector p. To complete the prototype of our global approach we have adopted the energy function we used is presented in the article [3]. Nevertheless this function could be changed according to different needs for different applications. This energy function is not so computationally expensive and requires less effort to implement. By minimizing this energy function, the control mesh is relaxed such as removing any external forces applied on a mass-spring system. Therefore the quads get arranged regular in the mesh. Although it is known to be far from optimal for fine meshes; however, the difference on relatively coarse meshes, such as the typical control meshes of subdivision surfaces, appears to be less significant.

$$E(p)=\sum_{p_i}\Big(\big(\sum_{p_j\in N_i}\alpha^i_j p_j\big)^2+\big(\sum_{p_j\in N_i}\beta^i_j p_j\big)^2+\big(\sum_{p_j\in N_i}\gamma^i_j p_j\big)^2\Big) \quad (4)$$

N_i is the one ring of the vertex p_i. We define the **one ring** of p_i as the set containing p_i itself and its neighbor vertices. The parameters α, β and γ are assigned according to the following algorithm.

The fairing result for a simple example can be seen in figure 12. The polyhedral mesh after fairing is more regular compared to the preliminary mesh.

<u>Algorithm. fairing parameters</u>

```
Denote the valence of $p_i$ as $k_i$.
If $i \neq j$, then we have
    Foreach $k_i$
        If $k_i \neq 4$
            $\begin{cases} \alpha^i_j = 2/k_i \cos(\frac{4\pi j}{k_i}) \\ \beta^i_j = 2/k_i \sin(\frac{4\pi j}{k_i}) \\ \gamma^i_j = \frac{1}{k_i} \end{cases}$    (5)
        Else
            $\begin{cases} \alpha^i_j = 1/4\cos(\pi j) \\ \beta^i_j = 1/4\sin(\pi j) \end{cases}$    (6)
        End If
    End Foreach
Else
    $\alpha^i_i = \beta^i_i = 0, \gamma^i_i = -1$    (7)
End If
```

7. RESULTS AND DISCUSSION

In this chapter, we tested our algorithm in three models and gave the implementation results: the input curve of a ring, a bottle and a spoon (figures 15-17). For example, figure 15.a shows the input curves of a ring model; The control polygon of the ring model after curve treatment is presented in figure 15.b; Figure 15. c. They illustrate the face cycles of the model identified by our algorithm; Figures 15.d and 15.e show respectively the control polyhedral mesh and smooth subdivision surface.

From the result of implementation, it can be seen that the result of the generated subdivision surfaces is very encouraging. The final surface interpolates the input curves and is well smoothed. However, there are still some aspects that need further improvement to get a better interpolation.

Firstly, in the part of curve treatment, the equation (1) used for the control point density adjusting takes only into account the length of each curve as the influence factor, so that the number of control points of each curve is proportional to its length. However, this way of adjusting can cause the incompatibility between curves while constructing the mesh. This may result in quads irregular, harming the mesh quality. A better solution should also consider the curvature of curves. Besides, the model constant C in equation (1) used to make a compromise between precision and efficiency is not easy to fix before trials.

Secondly, in the patch cycle identification, our algorithm fails when dealing with curve network which consists of several connected components. The Depth-First-Search method we implemented can't go further if there isn't any bridge to link these connected components. And it fails when the model includes surface features, like holes. In this case, some curve treatment can be added in future in order to avoid finding these cycles. Or a user interface can be designed to help detect the cycles. For example, in the model of the ring (fig. 13), the cycles that represent the holes of the ring have been deleted manually. If the designers want to present a hole in this model they may have to add some new edges (as shown in the figure 13) in order to drive our approach to the "wall" of the hole.

Thirdly, concerning the mesh fairing stage, the energy function (eq. 4) may not be optimal because it can only rearrange the quads to be more regular, but not change the shape. In the case where we need to change the shape of the mesh, a nonlinear function formulated in terms of curvature approximations is probably to yield somewhat better results. Besides, fairing in mesh is an indirect way to improve the final subdivision surface. Post-treatment, like a function evaluated on the final subdivision surface, is direct and preferred because it will be more reliable though also more computationally expensive.

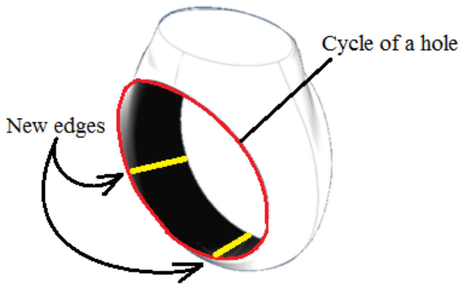


Figure 13: The ring model with surface feature: holes

8. CONCLUSION AND FUTURE WORK

In this article, a new framework is proposed in order to accelerate the freeform shape design process using 3D curves. This framework combines together the curve treatment, the polygon construction, cycle identification, and mesh generation, with the goal to generate a coarse mesh for subdivision surface from curves. An instance of this framework has been prototyped and experimented with various models.

For long term future works, the generation of subdivision surface from network containing open curves could be interesting. Open curve is the curve whose starting position is different with that of the ending position (fig. 14). This is much more difficult because open curves don't form face cycles. The possible solution would be that during curve treatment new curves should be created or the curves should be prolonged so that all the curves form close cycles.

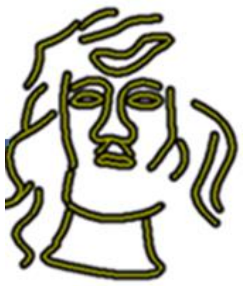


Figure 14 Curves network with open curves

REFERENCES

[1] Catmull, E., Clark, J., (1978), "Recursively generated B-spline surfaces on arbitrary topological meshes"; Computer-aided Design 10, pp. 350–355.

[2] Halstead, M., Kass, M., Derose, T., (1993), "Efficient, fair interpolation using Catmull-Clark surfaces"; In Proceedings of SIGGRAPH 93, Computer Graphics Proceedings, Annual Conference Series, pp. 35–44. 9

[3] SCHAEFER S., WARREN J., ZORIN D., (2004), "Lofting Curves Network using Subdivision Surfaces"; Proceedings of the 2004 Eurographics , pp.103-114.

[4] Nasri, A., (1997), "Curve interpolation in recursively generated b-spline surfaces over arbitrary topology"; Computer Aided Geometric Design, 14: No 1.

[5] Koel, D., Pablo, D-G., M.Gopi., (2005), "Sketching Free-form surfaces using network of curves"; Eurographics Workshop on Sketch-Basisd Interfaces and Modeling (2005)

[6] Gonen,O., Akleman, E., (2012), “Sketch basisd 3D modeling with curvature classification”; Shape Modeling International (SMI) Conference 2012, 36(5), pp.521–525

[7] Doo, D., Sabin, M., (1978), “Behavior of recursive division surfaces near extraordinary points”; Computer-Aided Design, Vol. 10, No. 6, pp. 356-360.

[8] Levin, A., (1999), “Interpolating nets of curves by smooth subdivision surfaces”; Proceedings of the 26th annual conference on Computer graphics and interactive techniques, pp. 57-64.

[9] Ruding, L., Mikchevitch, A., Pernot, J-P. and Veron, P. (2010), “Merging enriched Finite Element triangle meshes for fast prototyping of alternate solutions in the context of industrial maintenance”; Computer-Aided Design, 42(8), pp.670-681.

[10] Les A, P., Wayne, T., (1996), “The NURBS Book”, Chapter Five, Springer 2nd edition

[11] Kavitha, T., Mehlhorn, K., (2009), “Cycle basis graphs: Characterization, Algorithms, Complexity, and Applications”. Computer Science Review, 3(4): 199-243.

[12] Shpitalni, M., Lipson H., (1996), “Identification of faces in a 2D line drawing projection of a wireframe object”; Pattern Analysis and Machine Intelligence, IEEE Transactions on 18, 10 ,1000 –1012. 3, 8

[13] FATEMEH A., NINA A., (2011). “Surface Patches from Unorganized Space Curves”; The Eurographics Association and Blackwell Publishing Ltd. 30,5

[14] Tarjan, R. E., (1976), “Edge-disjoint spanning trees and depth-first search”; Acta Informatica, Vol 6, Issue 2, pp. 171-185

[15] Yongjie, Z., Chandrajit, L., Guoliang, X., (2005), “Surface Smoothing and Quality Improvement of Quadrilateral / Hexahedral Meshes with Geometric Flow”; Proceedings of 14th International Meshing Roundtable, pp. 449-468.

[16] Hoppe, H., DeRose, T., Duchamp, T., MacDonald J., Stuetzle, W., (1993), “Mesh Optimization”; Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pp. 19-26

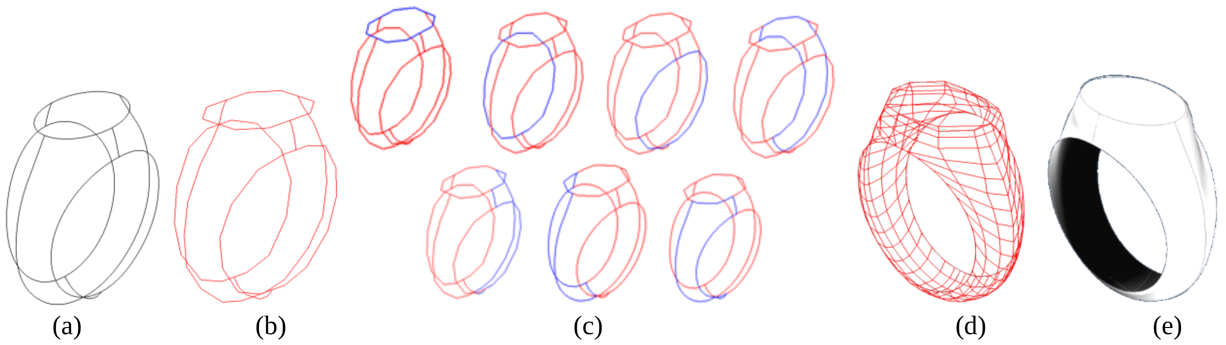


Figure 15 Implementation of the curves network of a ring. (a) Input curves; (b) Polygon network; (c) Face cycles identified in the polygon network; (d) Control polyhedral mesh; (e) Subdivision surface.

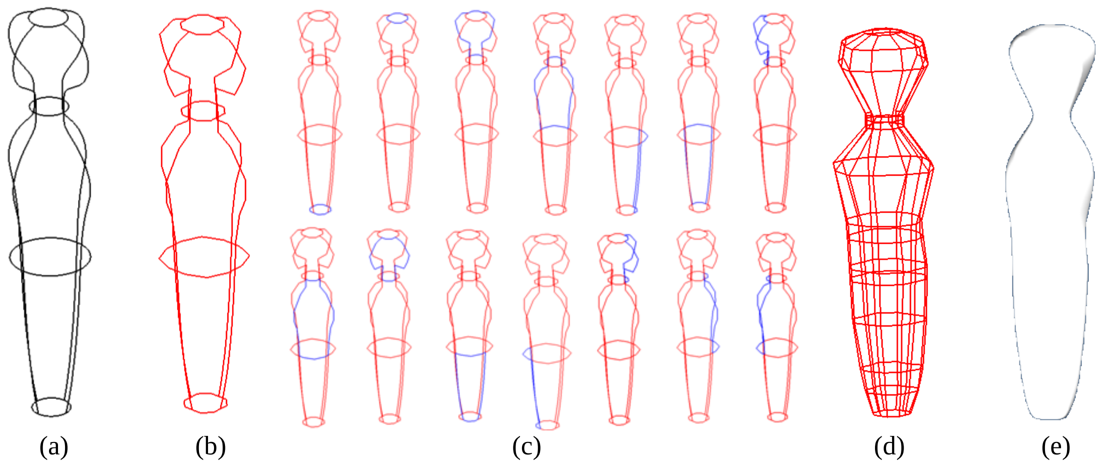


Figure 16 Implementation of the curves network of a bottle. (a) Input curves; (b) Polygon network; (c) Face cycles identified in the polygon network; (d) Control polyhedral mesh; (e) Subdivision surface

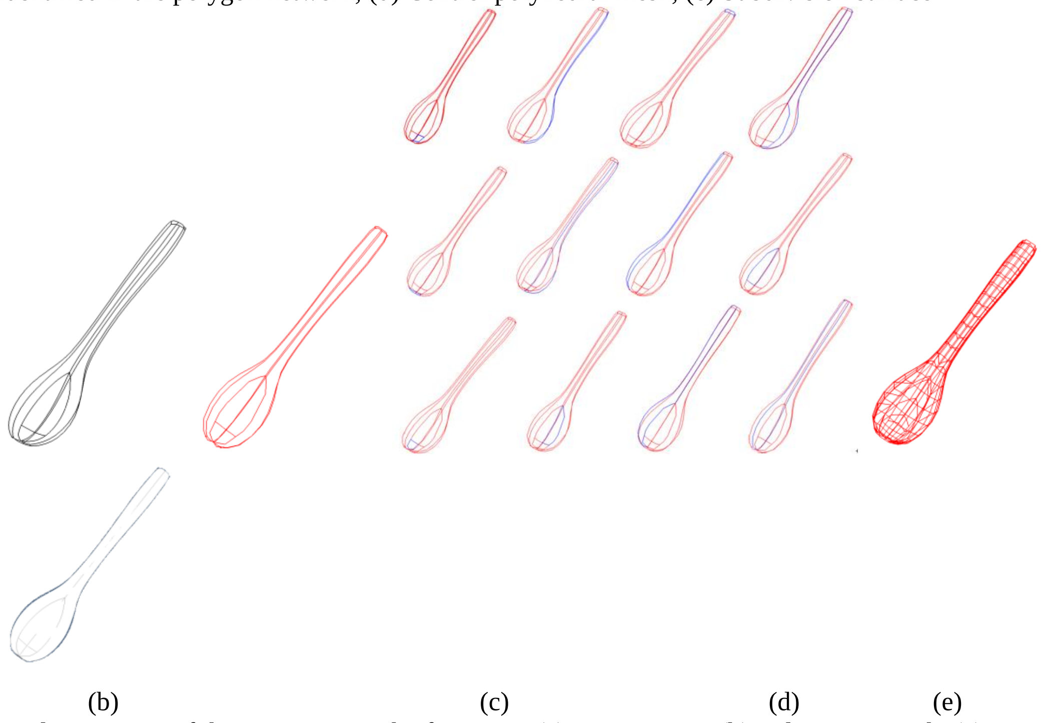


Figure 17 Implementation of the curves network of a spoon. (a) Input curves; (b) Polygon network; (c) Face cycles identified in the polygon network; (d) Control polyhedral mesh; (e) Subdivision surface