Fairing Wireframes in Industrial Surface Design

Yu-Kun Lai*, Yong-Jin Liu†, Yu Zang‡, Shi-Min Hu§
Tsinghua National Laboratory for Information Science and Technology,
Department of Computer Science and Technology,
Tsinghua University, P. R. China

ABSTRACT

Wireframe is a modeling tool widely used in industrial geometric design. The term *wireframe* refers to two sets of curves, with the property that each curve from one set intersects with each curve from the other set. Akin to the *u*-, *v*-isocurves in a tensor-product surface, the two sets of curves in a wireframe span an underlying surface. In many industrial design activities, wireframes are usually set up and adjusted by the designers before the whole surfaces are reconstructed. For adjustment, the fairness of wireframe has a direct influence on the quality of the underlying surface. Wireframe fairing is significantly different from fairing individual curves in that intersections should be preserved and kept in the same order. In this paper, we first present a technique for wireframe fairing by fixing the parameters during fairing. The limitation of fixed parameters is further released by an iterative gradient descent optimization method with step-size control. Experimental results show that our solution is efficient, and produces reasonably fairing results of the wireframes.

Keywords: wireframe fairing, industrial design

Index Terms: I.3.5 [Computational Geometry and Object Modeling]: Geometric algorithms, languages, and systems; I.3.5 [Computational Geometry and Object Modeling]: Splines.

1 INTRODUCTION

Wireframe is a useful tool in many industrial design activities. In these activities, two sets of curves $C = \{c_1, c_2, \ldots, c_u\}$ and $D = \{d_1, d_2, \ldots, d_v\}$ are first laid out by the designer. Each c_i and d_j has an intersection point P_{ij}. Akin to the *u*-, *v*-isocurves in a tensor-product surface, the two sets of curves C, D form a *wireframe* that spans an underlying surface. The surface can be generated by using *net surface* or other modeling primitives provided by many popular commercial CAD softwares; e.g., using Gregory patches [4] or using the curvature-continuous interpolation method [15]. The user can further modify the wireframe to improve the design quality. As a useful tool, the fairing of the wireframe is often needed.

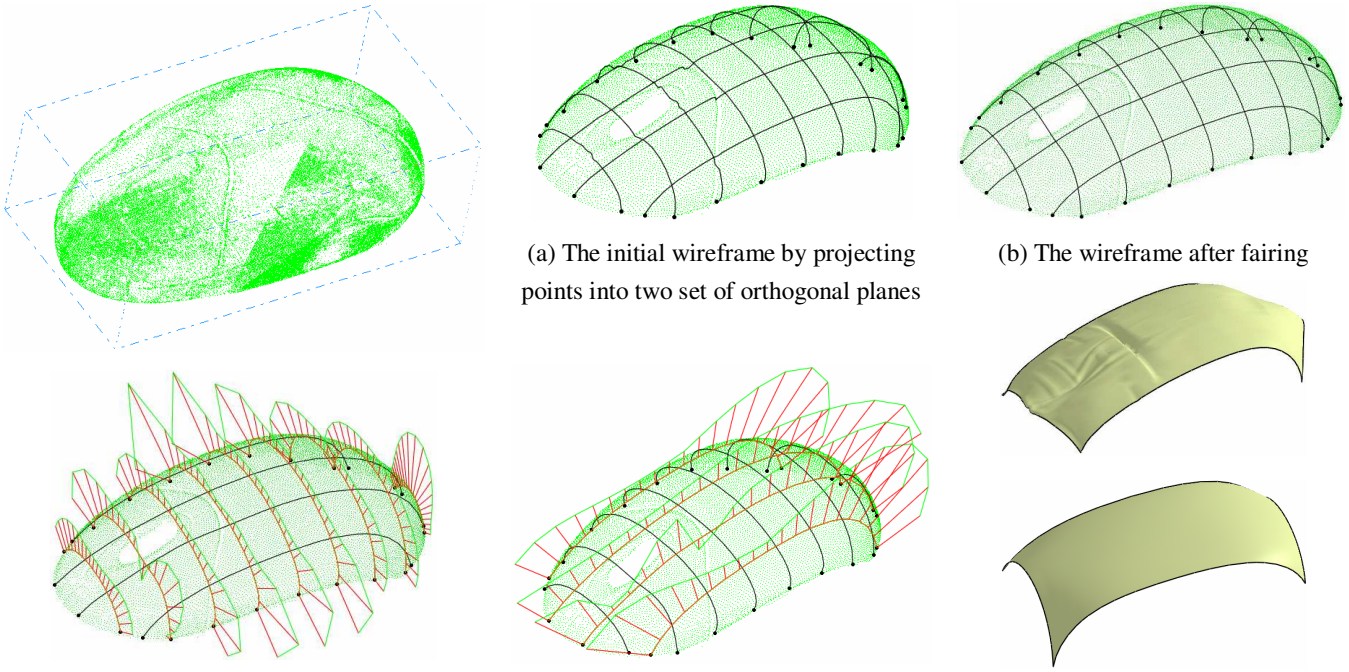
Wireframes can be generated from a variety of sources. Often designers build two sets of intersected curves as an intuitive way to design and edit a free-form surface. A more typical, but simpler example is to model a surface by spine and sectional curves. The spine is intersected with all the other curves, and this property should be maintained after fairing. Another source for wireframes comes from reverse engineering. For a physical surface without CAD model, a laser scanner is used to capture the point cloud data of the given surface. Refer to Fig. 1. To reconstruct the underlying surface, the scanned data is first intersected with two sets of near orthogonal planes. For each plane, a B-spline curve is created by projecting the points nearby onto the plane and approximating them with a planar curve. The two sets of intersection planes specified by users thus result in a wireframe. The wireframe may not be smooth enough due to the existence of scanning noise. So fairing the wireframe is necessary. To make the fairing effects shown more clearly, we have used surface interrogation algorithms introduced in [5] for visualization throughout the paper.

Fairing of curves and surfaces is a well studied problem in computer aided design [6, 3]. However, wireframe fairing is notably different since the intersections among curves must be maintained and kept in the same order. Compared to its wide uses in industrial design, relatively little work has been reported for wireframe fairing. Recently, Wallner et al. [14] propose an approach to compute fair webs, which amounts to smooth a set of connected curves to minimize the sum of integrals of the squared norm of first or second derivative, i.e. minimize $\sum_i \int_s |c_i'|^2 ds$ or $\sum_i \int_s |c_i''|^2 ds$, within the given underlying surface, or in $\mathbb{R}^3$.

Some interesting theoretical results of optimal curve set are presented in [14], as an extension of the work on energy-minimizing spline curves on surfaces [8]. These results are mainly developed for graphics applications such as remeshing, parameterization and surface restoration. However, they are not suitable for the design in industry, due to the following reasons: (1) In [14], each intersection point between two curves is considered as a knot, and each curve segment between knots is considered as a parameter curve. Thus in the design process, the wireframe may be split into a huge number of curve fragments, at the positions of both intersection points and knots of B-spline curves, where two piecewise polynomial curves join together. This is very impractical, since after fairing the surface cannot be edited through interactively modifying the wireframe. (2) Only optimal states of fairing quantities are studied in [14], while it is not clear how to add user control to the quantity of how close the modified wireframe to the original one. (3) The focus of [14] is to study fair webs constrained to given surfaces (or $\mathbb{R}^3$), while our purpose is to use wireframes to guide the design of surfaces.

Our presented work is also related to the traditional problem of curve fairing, but in a much more difficult setting. Automatic fairing of B-spline curves is first proposed by Sapidis and Farin [13]. Curvature variation is examined and the position of the point related to the biggest jump of curvature variation is updated to fair the given curve. A later work by Bonneau and Hagen [2] considers the problem of fairing rational splines. The work by Poliakoff [11] proposes an automatic curve fairing algorithm that generalizes Kjellander's method [10] to non-uniformly parameterized curves. The work by Hahmann generalizes both methods of [13] and [10] by applying iterative local fairing masks to B-spline surfaces [7]. Such methods mainly adjust "bad" points one by one, and thus it usually takes tens or hundreds of iterations to achieve desired result. Zhang et al. [16] extends the method [11] and proposes a method for fairing cubic splines that updates more than one points at each iteration, greatly reducing the required number of iterations. Our work fairs the given wireframe as a whole, minimizing an energy functional related to integral of curvatures. Moreover, we are dealing with wireframes, instead of a single spline curve.

*e-mail:laiyk03@mails.thu.edu.cn
†e-mail:liuyongjin@tsinghua.edu.cn
‡e-mail:zangyu@eyou.com
§e-mail:shimin@tsinghua.edu.cn



(a) The initial wireframe by projecting points into two set of orthogonal planes

(b) The wireframe after fairing

(c) The curvature plot of the curve set C after fairing

(d) The curvature plot of the curve set D after fairing

(e) The net surfaces spanned by the wireframe before (upper) and after fairing (lower)

Figure 1: Reconstruct the upper surface of mouse based on scanned point cloud.

In this work, an approach to fair a given wireframe is proposed, which has the following features: (1) It is guaranteed that each pair of curves from two sets of curves spanning the wireframe has an intersection, and the intersection points are ordered as input. This is necessary to ensure that the faired wireframe can be used to produce surfaces reasonably close to the input one, without degeneracy; (2) The topology of wireframe is not changed after fairing; only control points are changed, allowing easy further editing; (3) A fairness parameter is provided to allow users to balance between fairness and closeness to the original wireframes.

For a clear presentation, in Sec. 2, the fairing algorithm with fixed parameters is proposed. The limitation of fixed parameters is further released by an iterative optimization method, described in Sec. 3. Implementation issues and experimental results are given in Sec. 4 and finally our concluding remarks are presented in Sec. 5.

2 Fairing Algorithm with Fixed Parameters

Industrial applications usually require that wireframes after fairing should not deviate much from the original wireframes, or even controlled by a user-specified error bound. Thus it is practical to assume that the parameters at intersection points will have little changes. A stronger, but often useful assumption is simply fixing the parameters during fairing process, while only optimizing the position of each control point of spline curves. This section presents a fairing algorithm under this assumption. In the next section, this limitation is released by an iterative optimization algorithm which simultaneously optimizes the parameters at the intersection points and the positions of control points of spline curves.

For a pair of B-spline curves $c_i \in C$ and $d_j \in D$, assume their intersection point is P_{ij}, its parameters in c_i and d_j are $t_{ij}^{(c)}$ and $t_{ij}^{(d)}$, respectively. Assume after optimization, curves become $\bar{c}_i (i = 1,2,\ldots,u)$ and $\bar{d}_j (j = 1,2,\ldots,v)$. The following constraints guarantee that intersections between curve sets C, D exist, and in the same order as input:

$$\bar{c}_i(t_{ij}^{(c)}) - \bar{d}_j(t_{ij}^{(d)}) = 0, \tag{1}$$

for $i = 1,2,\ldots,u$ and $j = 1,2,\ldots,v$.

For each curve c_i (d_j is handled similarly), assume it is a B-spline curve of degree $k(c_i)$ with $(n(c_i)+1)$ control points $P_0^{(c_i)}, P_1^{(c_i)}, \ldots, P_{n(c_i)}^{(c_i)}$, either uniform or non-uniform. The curve is formulated as

$$c_i(t) = \sum_{l=0}^{n(c_i)} P_l^{(c_i)} N_{l,k(c_i)}(t), \tag{2}$$

where $N_{l,k(c_i)}$ is the l-th B-spline basis function of degree $k(c_i)$. Fairing the curve c_i amounts to minimizing the following energy term $E(c_i)$:

$$E(c_i) = \int_{t_{k(c_i)-1}}^{t_{n(c_i)+1}} \left|\left|c_i''(t)\right|\right|^2 dt + \alpha \sum_{l=0}^{n(c_i)} \left|\left|\bar{P}_l^{(c_i)} - P_l^{(c_i)}\right|\right|^2, \tag{3}$$

where $\bar{P}'s$ are those control points after fairing. α is a constant which balances the relative importance of fairness (with smaller α) and closeness to the original curves (with larger α).

Assume that

$$B_{rs}^{(c_i)} = \int_{t_{k(c_i)-1}}^{t_{n(c_i)+1}} N''_{r,k(c_i)} \cdot N''_{s,k(c_i)}, \tag{4}$$

then

$$\frac{1}{2}\frac{\partial E(c_i)}{\bar{P}_r^{(c_i)}} = \sum_{s=0}^{n(c_i)} B_{rs}^{(c_i)} \bar{P}_s + \alpha\left(\bar{P}_r^{(c_i)} - P_r^{(c_i)}\right), \tag{5}$$

$r = 0, 1, \ldots, n(c_i)$. Moreover, the endpoints of each curve should not be moved, i.e.,

$$\bar{P}_0^{(c_i)} - P_0^{(c_i)} = 0, \;\; \bar{P}_{n(c_i)}^{(c_i)} - P_{n(c_i)}^{(c_i)} = 0. \tag{6}$$

The wireframe fairing is solved by minimizing

$$E = \sum_{i=1}^{u} E(c_i) + \sum_{j=1}^{v} E(d_j), \tag{7}$$

with $2u+2v+uv$ constraints described by equations 1 and 6. Since x, y, z components of each control point is independent, they can be solved independently, using Lagrange multiplexer approach, i.e., to find a set of $\bar{P}'s$ with $2u+2v+uv$ Lagrange variables $\gamma_1 \ldots \gamma_{2u}$, $\lambda_1 \ldots \lambda_{2v}$, and $\delta_{1,1} \ldots \delta_{u,v}$, minimizing

$$\begin{aligned} \bar{E} = \;& \textstyle\sum_{i=1}^{u} E(c_i) + \sum_{j=1}^{v} E(d_j) \\ + \;& \textstyle\sum_{i=1}^{u} \gamma_i \left(\bar{P}_0^{(c_i)} - P_0^{(c_i)} \right) \\ + \;& \textstyle\sum_{i=1}^{u} \gamma_{u+i} \left(\bar{P}_{n(c_i)}^{(c_i)} - P_{n(c_i)}^{(c_i)} \right) \\ + \;& \textstyle\sum_{j=1}^{v} \lambda_j \left(\bar{P}_0^{(d_j)} - P_0^{(d_j)} \right) \\ + \;& \textstyle\sum_{j=1}^{v} \lambda_{v+j} \left(\bar{P}_{n(d_j)}^{(d_j)} - P_{n(d_j)}^{(d_j)} \right) \\ + \;& \textstyle\sum_{i=1}^{u} \sum_{j=1}^{v} \delta_{i,j} \left(\bar{c}_i(t_{ij}^{(c)}) - \bar{d}_j(t_{ij}^{(d)}) \right) \end{aligned} \tag{8}$$

All the unknowns form a vector

$$U := \left(\ldots \bar{P}_l^{(c_i)} \ldots \bar{P}_l^{(d_j)} \ldots \gamma_i \ldots \lambda_j \ldots \delta_{i,j} \ldots \right). \tag{9}$$

Minimizing $\bar{E}$ amounts to solving $\partial \bar{E} / \partial U = 0$. Due to the local support nature of B-spline curves, a sparse linear system is derived which can be solved efficiently using, e.g., conjugate gradient on normal equations.

Handling of NURBS wireframes. NURBS curves can be usually converted to non-rational representation by homogeneous coordinates. However, for the constraint equations 1 and 6, the homogeneous coordinates do not hold. One possibility is to sacrifice the freedom of optimizing weights $w's$ and optimize the positions $P's$ only. This is practical since NURBS curves are usually used in CAD systems for accurately representing simple curves like conic sections; modifying weights are usually not necessary.

Choice of Parameter α. Parameter α is used to balance fairness and closeness between original and faired wireframes. Larger α prohibits curves from changing too much while smaller α tends to produce fairer output curves. α can be specified for each c_i and d_j curves, or a consistent value for all the curves if no particular preference to each individual curve is necessary. Experimental results on performance of different α are presented in Sec. 4.

Preserving tangents at endpoints. For some applications, not only the positions of endpoints of each curve should be preserved after fairing, some particular tangents of curves at endpoints should also be preserved. This can also be solved by a sparse linear system with the following difference: since x, y, z components are now coupled, they cannot be treated independently, as before. On the contrary, each variable discussed in previous formulae now becomes three variables with suffix x, y and z, respectively.

Without loss of generality, consider that the tangent of a particular curve at P_0 should be kept. By introducing a new variable d, the following constraint should be satisfied:

$$\bar{P}_1 = d(P_1 - P_0) + P_0. \tag{10}$$

To use Lagrange multiplexer approach to solve this problem, three Lagrange variables τ_x, τ_y, τ_z are introduced, and the following term is added to $\bar{E}$:

$$\begin{aligned} \Delta\bar{E} = \;& \tau_x \left(\bar{P}_{1x} - d(P_{1x} - P_{0x}) + P_{0x} \right) \\ + \;& \tau_y \left(\bar{P}_{1y} - d(P_{1y} - P_{0y}) + P_{0y} \right) \\ + \;& \tau_z \left(\bar{P}_{1z} - d(P_{1z} - P_{0z}) + P_{0z} \right). \end{aligned} \tag{11}$$

We then have

$$\frac{\partial \Delta\bar{E}}{\partial \left(\bar{P}_{1x}, \bar{P}_{1y}, \bar{P}_{1z} \right)} = \left(\tau_x, \tau_y, \tau_z \right), \tag{12}$$

$$\frac{\partial \Delta\bar{E}}{\partial \left(\tau_x, \tau_y, \tau_z \right)} = \left(\bar{P}_1 - d(P_1 - P_0) - P_0 \right)_{x,y,z}, \tag{13}$$

$$\frac{\partial \Delta\bar{E}}{\partial d} = -\tau_x(P_{1x} - P_{0x}) - \tau_y(P_{1y} - P_{0y}) - \tau_z(P_{1z} - P_{0z}). \tag{14}$$

Thus the problem can still be solved by the following sparse linear system:

$$\frac{\partial \left(\bar{E} + \Delta\bar{E} \right)}{\partial \bar{U}} = 0, \tag{15}$$

where $\bar{U}$ is the unknown vector combining U and newly introduced variables $\tau_x's, \tau_y's, \tau_z's$ and $d's$.

3 Fairing Algorithm with Optimized Parameters

The algorithm introduced in the previous section is efficient to compute, however, at the cost of producing suboptimal result that may not be satisfied for the best possible performance. This limitation is released by using the following iterative optimization algorithm, i.e., the parameters $t_{ij}^{(c)}$ and $t_{ij}^{(d)}$ of intersection points of curves c_i and d_j are now allowed to be changed, while the ordering of parameters on a particular curve is still preserved.

Assume $V = \left(\ldots t_{ij}^{(c)} \ldots t_{ij}^{(d)} \ldots \right)$ represents the parameter vector to be optimized. Energy functional with Lagrange variables $\bar{E}$ is a functional of both U and V. For a fixed vector of V, the previous section already gives the solution to find the optimal $\bar{E}(U)$. V is initially set to the parameters in the given input wireframe. Starting from this initial V_0, an iterative process is incorporated, producing a serial of $V's$, until convergence. At a particular step, we have V_i and want to compute the improved V_{i+1} together with unknown variables U_{i+1}. Note that given V_i, U_i can be computed with the method in Sec. 2 by solving $\frac{\partial \bar{E}}{\partial U}(U_i, V_i) = 0$ with known V_i. To compute the gradient of E w.r.t. V, $\partial U / \partial V$ is first computed at a particular value of (U_i, V_i). Without loss of generality, consider the parameter $\bar{t}_{kl}^{(c)} = t_{kl}^{(c)} + \Delta t_{kl}^{(c)}$, we have

$$\bar{c}_k(\bar{t}_{kl}^{(c)}) = \bar{c}_k(t_{kl}^{(c)}) + \bar{c}_k'(t_{kl}^{(c)}) \Delta t_{kl}^{(c)} + o(\Delta t_{kl}^{(c)}). \tag{16}$$

Denote $\partial \bar{E} / \partial U = \mathbf{A}(V) \cdot U + B(V)$, where $\mathbf{A}(V)$ and $B(V)$ are a matrix and a vector solely depend on V. $\Delta V_i = (0, \ldots 0, \Delta t_{kl}^{(c)}, 0, \ldots, 0)$, and U_i will be changed to $U_i + \Delta U_i$ so that $\frac{\partial \bar{E}}{\partial U}(U_i + \Delta U_i, V_i + \Delta V_i) = 0$, i.e.,

$$\mathbf{A}(V_i + \Delta V_i) \cdot (U_i + \Delta U_i) + B(V) = 0. \tag{17}$$

Putting equation 16 into equation 17 leads to

$$\left(\mathbf{A}(V_i) + \Delta \mathbf{A}_{kl}^{(c)}(V_i) \Delta t_{kl}^{(c)} \right) (U_i + \Delta U_i) + B(V) = o(\Delta t_{kl}^{(c)}), \tag{18}$$

where $\Delta \mathbf{A}_{kl}^{(c)}$ is also a sparse matrix related to V_i. Since $\mathbf{A}(V_i) \cdot U_i + B(V_i) = 0$, we have

$$\frac{\partial U}{\partial t_{kl}^{(c)}} = -\mathbf{A}^{-1} \Delta \mathbf{A}_{kl}^{(c)} \cdot U_i. \tag{19}$$

Note that $\mathbf{A}$ only depends on V_i. Putting together, we get

$$\frac{\partial U}{\partial V} = -\mathbf{A}(V_i)^{-1} \left[\ldots \Delta \mathbf{A}_{kl}^{(c)} \cdot U_i \ldots \right] \tag{20}$$

Thus, the computation of $\partial U/\partial V$ can be performed efficiently. Particularly, if $\bar{P}$ represents the vector of all the control points (which is part of U), we have $\partial\bar{P}/\partial V$. To find the optimal V that minimizes E, a projected gradient descent algorithm is used. Note that from Eqn. 7, $E = \bar{P}^T\mathbf{T}\bar{P}$, where $\mathbf{T}$ is a symmetrical matrix. At particular position (U_i, V_i), the gradient direction for E w.r.t. V can be computed as:

$$\frac{\partial E}{\partial V}(V_i) = 2\bar{P}_i^T\mathbf{T}\frac{\partial \bar{P}_i}{\partial V}(U_i, V_i). \tag{21}$$

The parameter V_i can be updated along this direction, i.e.,

$$V_{i+1} = V_i - t\frac{\partial E}{\partial V}(V_i), \tag{22}$$

where t is a step-size control to ensure the correct ordering of parameters of intersection points along each curve and the "significant" descent of the energy functional. For the latter requirement, Armijo rule [9] can be applied. Note that $U_i - t\partial U/\partial V(U_i, V_i)$ together with V_{i+1} usually do not lie in the solution space of equation 8; solving U_{i+1} from V_{i+1} with this equation amounts to projecting the positions back to the solution space.

Starting from the initial parameters as V_0, an iterative process is used, until convergence. Note that gradient descent method always converges to a local minimum; however, it is rather reasonable, since the input parameters are usually good approximations to the final parameters.

Controllable global error bound. α in equation 3 is used to balance the fairness and closeness between original and faired wireframes. However, the particular value of α is meaningless to the users in industry. A more intuitive way to them is to fair the wireframe with a user-specified global error bound ε. So it is desired that the program can convert the value ε to the corresponding value α automatically. This can be achieved by the Newton-Raphson method [12]. Starting with an initial α, at each step, the deviation error between original and current wireframes is computed whose value is used by Newton-Raphson method to compute $\Delta\alpha$. The iteration process is stopped if a prescribed iteration number is reached or the deviation error is sufficiently closed to the given error bound. Experimental results on the different error bound control is presented in the next section.

4 Experimental Results

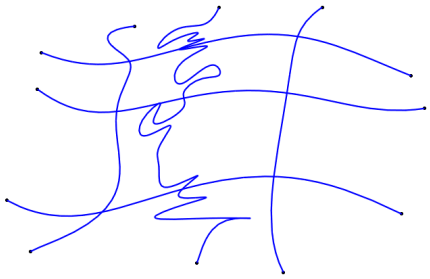


Figure 2: An example of input wireframe.

Choice of Parameter α. Fig. 2 is an example of input wireframe. Two sets of curves C, D both contain three curves. Each pair of curves from both sets has a single intersection. Fig. 3 shows the results obtained with fixed parameters, no tangent restriction and with different $\alpha's$. Table 1 shows the energy E before and after fairing, and the maximal movement of control points in L_2 norm where the largest distance between the two endpoints of curves is normalized to be 1. The computation of fixed-parameter problem can be performed efficiently by conjugate gradient on normal equations

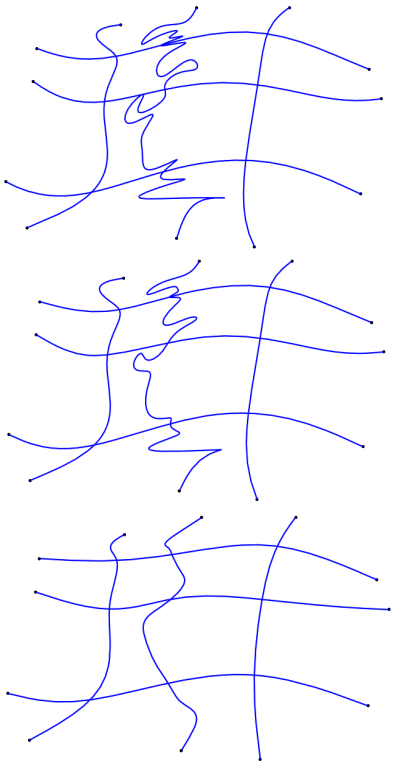


Figure 3: The faired wireframes with different α: 10^{-1}, 10^{-3}, 10^{-5} (from top to bottom)

α	E_{old}	E_{new}	normalized max. change
10^{-1}	12.70	11.67	0.00295
10^{-3}	12.70	2.47	0.0314
10^{-5}	12.70	0.10	0.101

Table 1: Errors and maximal changes of the wireframes with different α parameters.

[12]. Note that intersection points are kept after fairing. Smaller α leads to fairer output wireframe (with smaller E_{new} but at the cost of larger maximal change of control points). It can be easily found that the wireframes are significantly smoother after the fairing operation, especially with smaller parameter α.

Tangent preservation. Using the sparse linear system Eqn. 15, tangents at some or all of the end points can be preserved after fairing. This is extremely useful when the wireframe is used in conjunction with nearby surfaces. Fig. 4 gives such an example. Note that the curves used are space curves embedded in $\mathbb{R}^3$ and each pair of curves from different sets always has a single intersection in this example. The input wireframe is faired without and with tangent preservation, respectively. It can be noticed that using tangent preservation guarantees that faired wireframe has the same tangent directions as input at end points, at the cost of slightly higher E_{new} (for the same α). For this example, E_{new} are 0.3634 and 0.5406 for the results without and with tangent preservation.

Controllable global error bound. Fig. 5 shows a wireframe with $C = \{c_1, c_2, c_3, c_4\}$ and $D = \{d_1, d_2, d_3\}$. The length of each curve is $l(C) = \{224.4mm, 178.6mm, 182.1mm, 169.7mm\}$ and $l(D) = \{241.7mm, 245.4mm, 218.1mm\}$. The wireframes after fairing with different error bounds $26mm, 17.5mm, 7.15mm, 1.45mm$ are illustrated in Fig.5(c1-c4). It clearly shows that the larger the error bound specified by users, the more straightness the resulted curves

in wireframes. The property of fixed endpoints of each curve is also demonstrated in Fig. 5(b).

Industrial applications. Wireframe is a useful tool in industrial design. One application is reverse engineering as shown in Fig. 1 in which the user spans a wireframe by interactive design based on the scanned points. The bounding box of the mouse model in Fig. 1 is $104.8mm \times 45.3mm \times 31.5mm$. The error bound between the original (see Fig. 1b) and the faired (see Fig. 1c) wireframes is $0.991mm$.

Another application is to clean the surface fragments by replacing them with a single whole patch. Refer to Fig. 6. The surfaces used in industry usually go through many design phases. In each phase, different functionality is considered and surface is modified; e.g., to enhance the developability in some central region, designers trim a hole in-between and re-fill a small developable patch with smooth connectivity to the boundary surface. So after some design activities, the surface may consist of many patch fragments though it is overall still smooth (ref. Fig. 6a). Finally, at the manufacturing step, it is desired by users to use a single surface as a whole rather than lots of fragments. In this case, the users span a wireframe by interactively setting up cutting planes to obtain intersection curves. After fairing, a net surface is generated from wireframe to serve as the single surface merging all the fragments (ref. Fig. 6b). The approximation error can be controlled by the number of curves in wireframe and further in the fairing process. In the example in Fig. 6, the whole surface is about $3557mm \times 1046mm$, and the error bound is $57.1mm$. The fairness of the wireframe obviously has effect on the resulting net surface. In the example shown in Fig. 6, highlight line method [1] is used to inspect the surface quality.

We implemented our algorithm as a plugin module to a commercial CAD system with C++. The experiments were carried out on a Intel Core2Duo 2GHz Laptop with 2GB memory. Examples shown in the paper took within 3 seconds to compute; the times are dominated by solving a few sparse linear systems. Thus our method is sufficiently fast for interactive wireframe editing.

5 Conclusions and Future Work

In this paper, a simple yet effective wireframe fairing method is proposed. Optimal algorithms with both fixed parameter and optimized parameters are discussed in details. Experimental results show that this method is feasible to produce valid and reasonably faired wireframes suitable for industrial surface design in an interactive way.

Our proposed method has been implemented as a plugin module for a commercial CAD software with extensive testing. To further fulfill the diverse requirements of real applications, we plan to explore the use of criteria other than energy functional minimization, e.g. improving the monotonicity or convexity of curvature plots in the future. Besides improving the fairness of individual curves while preserving necessary restrictions, it may be desirable to also consider the problem of improving the layout of curves so that resulting curves will have more uniform spacing.

Acknowledgment

The work was supported by the National Natural Science Foundation of China (Project No. 90718035, 60603085, 60736019), the National High Technology Research and Development Program of China (Project No. 2007AA01Z336) and the Project funded by Tsinghua Basic Research Foundation.

References

[1] K. P. Beier and Y. Chen. Highlight-line algorithm for realtime surface-quality assessment. *Computer-Aided Design*, 26(4):268–277, 1994.

[2] G.-P. Bonneau and H. Hagen. Variational design of rational bézier curves and surfaces. In *Curves and Surfaces in Geometric Design*, pages 51–58, 1994.

[3] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Academic Press, 5th edition, 2002.

[4] J. A. Gregory. N-sided surface patches. In *The Mathematics of Surfaces*, pages 217–232, 1986.

[5] H. Hagen, S. Hahmann, T. Schreiber, Y. Nakajima, B. Wordenweber, and P. Hollemann-Grundstedt. Surface interrogation algorithms. *Computer Graphics and Applications*, 12(5):53–60, 1992.

[6] H. Hagen and G. Schulze. Automatic smoothing with geometric surface patches. *Computer Aided Geometric Design*, 4(3):231–235, 1987.

[7] S. Hahmann. Shape improvement of surfaces. *Geometric Modelling, Computing [Suppl.]*, 13:135–152, 1998.

[8] M. Hofer and H. Pottmann. Energy-minimizing splines in manifolds. In *Proceedings of ACM SIGGRAPH*, pages 284–293, 2004.

[9] T. Kelley. *Iterative Methods for Optimization*. SIAM, 5th edition, 1999.

[10] J. Kjellander. Smoothing of cubic parametric splines. *Computer-Aided Design*, 15(3):175–179, 1983.

[11] J. Poliakoff. An improved algorithm for automatic fairing of non-uniform parametric cubic splines. *Computer-Aided Design*, 28(1):59–66, 1996.

[12] W. Press, S. Teukolsky, W. Vettering, and B. Flannery. *Numerical Recipes in C++*. Combrdige University Press, 2nd edition, 2002.

[13] N. Sapidis and G. Farin. Automatic fairing algorithm for b-spline curves. *Computer-Aided Design*, 22(2):121–129, 1990.

[14] J. Wallner, H. Pottmann, and M. Hofer. Fair webs. *The Visual Computer*, 23(1):83–94, 2007.

[15] X. Ye. Curvature continuous interpolation of curve meshes. *Computer Aided Geometric Design*, 14:169–190, 1997.

[16] C. Zhang, P. Zhang, and F. Cheng. Fairing spline curves and surfaces by minimizing energy. *Computer-Aided Design*, 33(13):913–923, 2001.

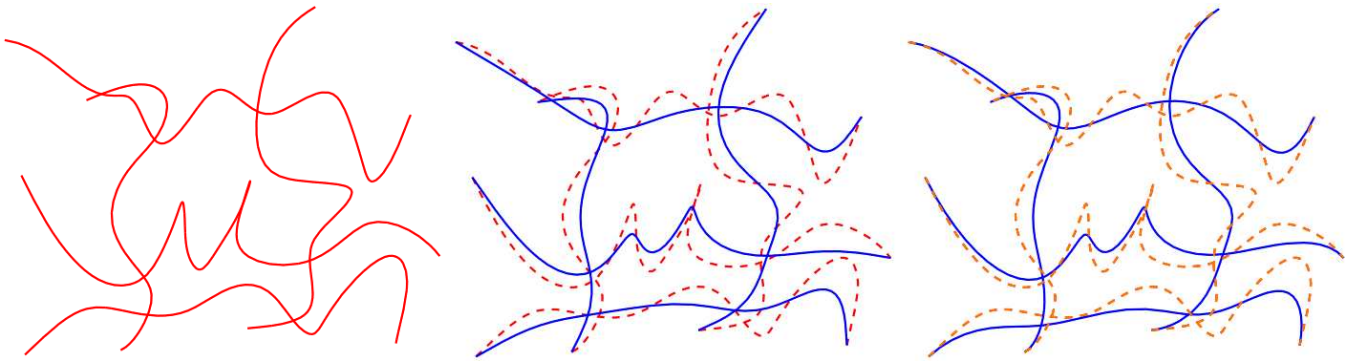
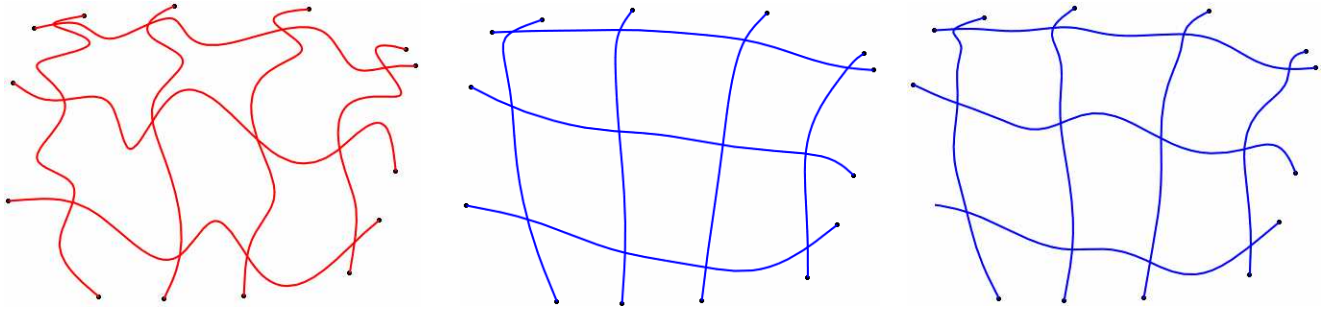


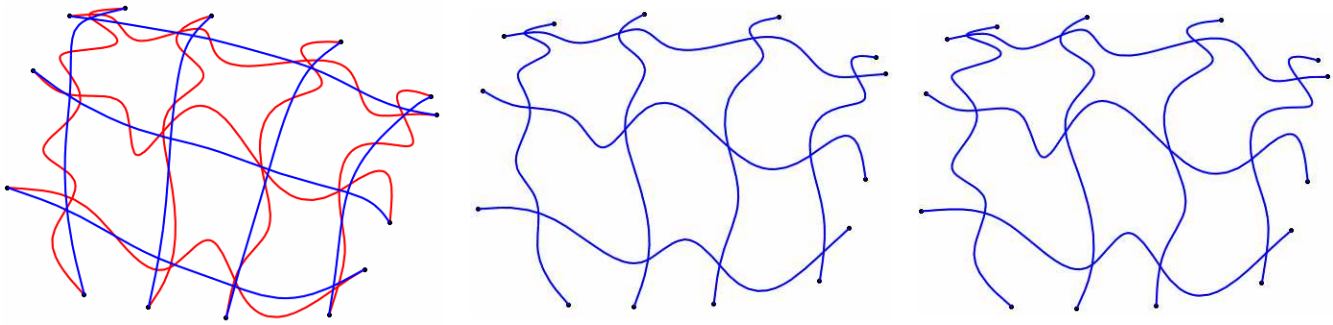
Figure 4: Fairing a given wireframe (left) without (middle) and with (right) tangent preservation.



(a) The original wireframe

(c1) Error bound 26mm

(c2) Error bound 17.5mm



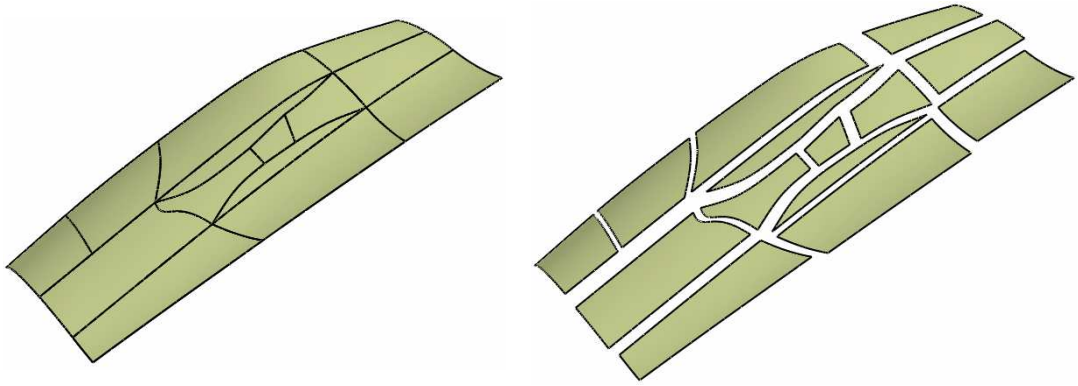
(b) The alignment of the original and faired wireframes in (c1)

(c3) Error bound 7.15mm

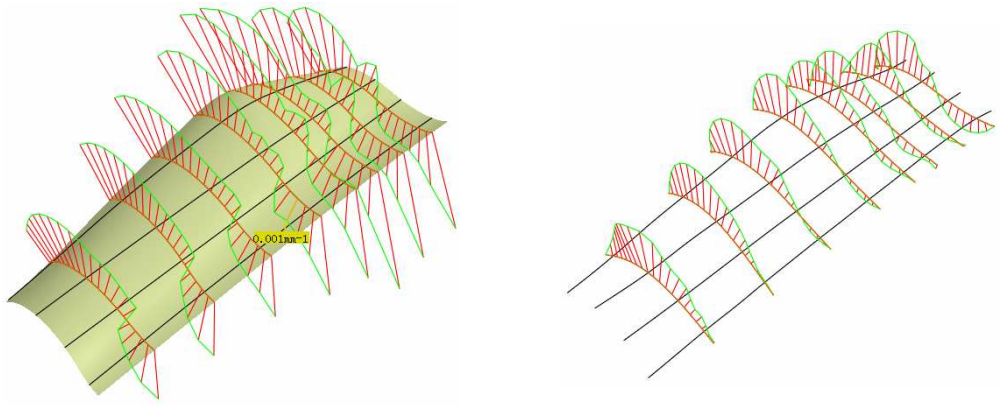
(c4) error bound 1.45mm

(c) Fairing wireframes

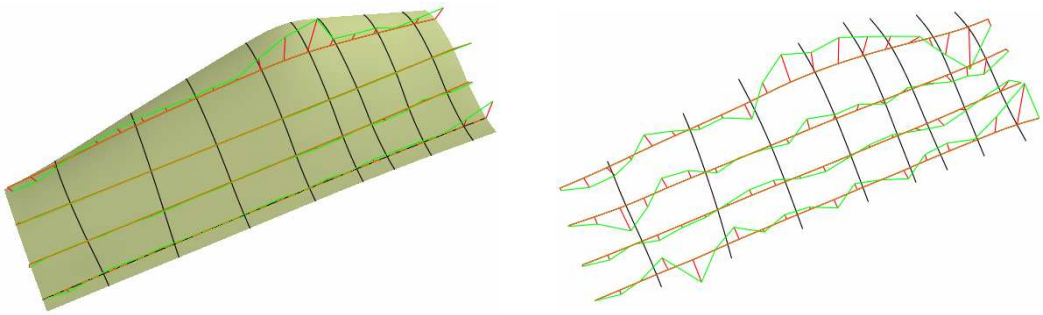
Figure 5: Fairing wireframes with different user-specified error bounds.



(a) The initial design of a smooth wing surface with many patch fragments

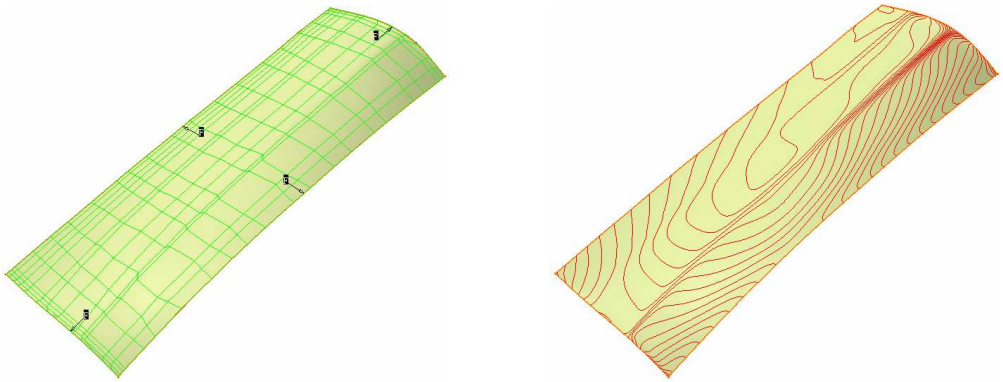


The *C* set of curves in the original wireframe; The *C* set of curves in the wireframe after fairing



The *D* set of curves in the original wireframe; The *D* set of curves in the wireframe after fairing

(b) Wireframes before and after fairing; Curvature plot for each curve in wireframe is also drawn



The control points of the net surface Surface quality inspection by hightlight-line method

(c) The net surface spanned by the wireframe after fairing

Figure 6: Surface fragments cleaning with wireframes.