Fast and Local Fairing of B-Spline Curves and Surfaces

P. Salvi[†], H. Suzuki[†], T. Várady[‡]

[†] University of Tokyo, [‡] Geomagic Hungary

Abstract. The paper proposes a fast fairing algorithm for curves and surfaces. It first defines a base algorithm for fairing curves, which is then extended to the surface case, where the isocurves of the surface are faired. The curve fairing process involves the discrete integration of a pseudo-arc-length parameterization of B-spline curves, with a blending and fitting phase concluding the algorithm. In the core of the fairing method, there is a fairness measure introduced in an earlier paper of the authors. This measure is based on the deviation from an ideal or target curvature. A target curvature is a series of smooth curvature values, generated from the original curve or surface. This curve and surface fairing technique is local and semi-automatic, but the user can also designate the region to be faired. The results are illustrated by a few examples on real-life models.

Key words: Curves and Surfaces, Geometric Optimization, Reverse Engineering

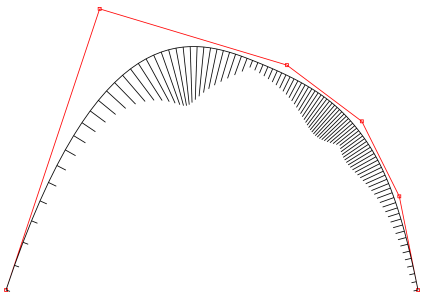
1 Introduction

Fairing curves and surfaces plays an important role in CAGD, especially in the automobile industry. Connected curve nets and surfaces have to be smoothed while preserving their original features and connectivity. Automating this process is a crucial problem of Digital Shape Reconstruction[1] [3].

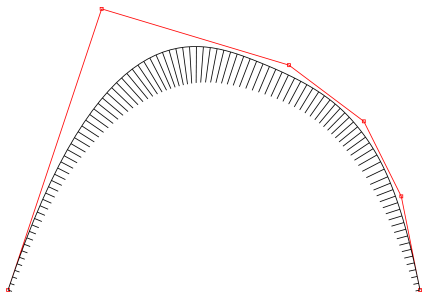
Fairness does not have an exact mathematical definition, and it may have a different meaning depending on the context [9]. Still, there are some common properties of what we would call *fair*. It certainly includes some kind of mathematical continuity, e.g. C^2 or G^2. It also incorporates the requirement that a surface should have even reflections. Another important, although less intuitive, requirement is the smooth transition of curvature [6]. These properties can be tested using an arsenal of interrogation methods, e.g. isophote lines, curvature combs or curvature maps. But even with these, finding and mending small artifacts is a laborious manual process.

The concept of *target curvature* and an iterative algorithm based on it is reviewed in Section 2. A new, fast algorithm is explained in Section 3, accompanied by test results in Section 4.

[1] formerly called Reverse Engineering.



(a) Curve with curvature comb



(b) The same curve with the target curvature comb.

Fig. 1. Target curvature of a cubic curve.

2 Related Work

There is an abundant literature on creating fair curves and surfaces, dealing with both the definition of fairness and smoothing algorithms. Here we will only cover a previous publication of the authors, defining a fairness measure based on a target curvature. The algorithm in Section 3 will use this measure to generate fair curves and surfaces. This section also presents an iterative algorithm based on the same measure. For a more comprehensive review on fairness measures and algorithms, look at [9,10].

2.1 Fairness Measure

Most fairing algorithms use some kind of fairness measure as a numerical representation of smoothness. In other words, we can say that a curve c is fair, if

$$\mathcal{F}(c) < \tau$$

applies, where τ is a user-defined tolerance. Salvi and Várady [10] propose a measure based on a *target curvature* — a sufficiently smoothed series of sampled curvature values. It can be thought of as a smoothed curvature comb (Fig. 1). From the target curvature, a new fairness measure naturally arises for curves:

$$E = \sum_i |\kappa(t_i) - g(t_i)|^2,$$

where g is the smoothed (target) curvature and t_i are sampled parameter points. The definition means that the less a curve's curvature deviates from the target curvature, the more fair it is. This measure is parameterization-independent, since it uses the curvature, that is a geometrical property of the curve. Most smoothness measures in use today are parameterization-dependent, and thus cannot be used safely when the parameterization considerably deviates from the arc-length parameterization [5].

Determining the Target Curvature For creating the target curvature, simple, fast algorithms can be used. One way is averaging consecutive sampled values of the curvature. As global averaging may remove or flatten important parts of the curvature, the user should be allowed to modify the target curvature manually, or restrict its changes.

Another possibility is to fit a curve over sampled values of the curvature (fitting over curvature comb endpoints would be more intuitive, but it may have self-intersections). For smoother results, we may also minimize the curvature of the fitted curve. This leads to the minimization of the functional

$$F(g) = \sum_i |g(t_i) - \kappa(t_i)|^2 + \int \hat{\kappa},$$

where $\hat{\kappa}$ is the curvature of the fitted curve g.

For both of these methods, the number of samples may have a great impact on the resulting target curvature. Since smoothness is much more important than faithfulness to the original curvature, a loose sampling rate is recommended.

Extension to Surfaces Most fairness measures for curves have their surface counterparts. This measure is no exception: it can be logically applied to surfaces using some combination of the principal curvatures, e.g.:

$$\Pi = \sum_i \sum_j (|\kappa_1(u_i, v_j) - g_1(u_i, v_j)|^2 + |\kappa_2(u_i, v_j) - g_2(u_i, v_j)|^2).$$

2.2 An Iterative Algorithm

The algorithm presented by the authors in [10] fixes all but one control point and minimizes the fairness measure defined above by moving the remaining one. It also constrains the deviation from the original curve using its control polygon. One may apply various heuristics to find the overall optimal shape, e.g. moving the control point with the largest error in every turn. Although not mentioned in the original paper, the downhill simplex method [8] can be used for minimization, and checking the deviation of a suitable neighbourhood instead of the whole domain can speed up the calculation of the fairness measure as well. Its extension to the surface case is straightforward; a sample result is presented in Fig. 2.

3 The Proposed Algorithm

The main idea is that if we reparameterize a curve to arc-length parameterization, the curvature will be equal to the norm of the second derivative [5], so we can create a target curvature and integrate it twice to find the faired curve directly. Discrete integration provides us with points we can use for fitting, eventually resulting in a faired curve or surface. The use of fitting harms locality of course, but in exchange we get a great increase in speed. In the following sections we first define the algorithm for curves and then extend it to surfaces.

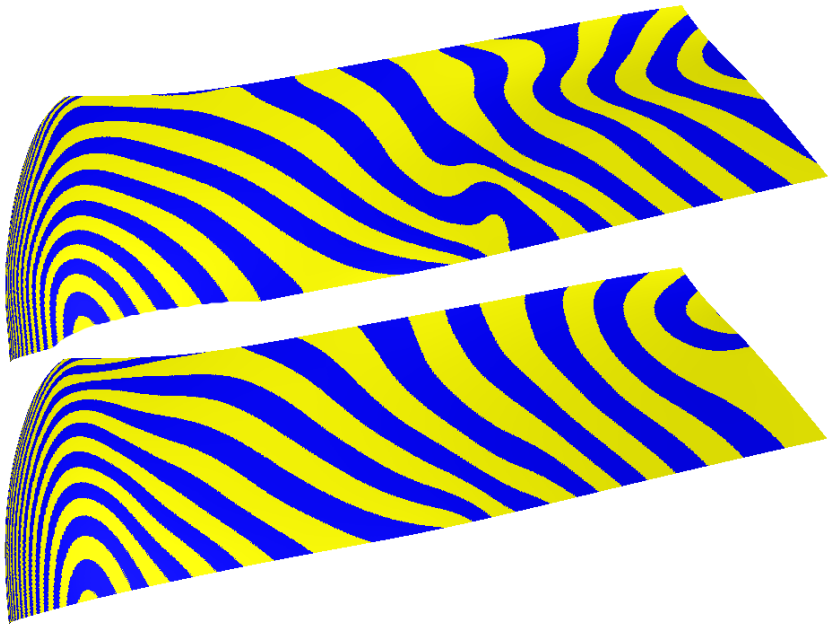


Fig. 2. Fiat body part faired by a previous algorithm of the authors [10].

3.1 Curve Case

The algorithm can be broken down to the following four phases:

1. Creating a pseudo arc-length parameterization for n points.
2. Integrating twice starting at the left end to the right and vice versa, with deviation control.
3. Blending the results.
4. Fitting a spline that approximates the blended points.

Pseudo arc-length parameterization means that the parameter domain is divided into $n-1$ small segments that have approximately equal length. At these points the curvature is the same as the norm of the second parametric derivative of the curve in arc-length parameterization, i.e. when the same distance in parameter space means the same distance in arc length. We will denote the parameter of the k^{th} point with p_k.

The second phase is carried out both from the left and from the right side. Since these are symmetrical, we examine only the former. The numerical integration algorithm needs the direction of the second derivative, as well as the starting point. In arc-length parameterization the second derivative is in the normal direction, which can be easily calculated. In the first integration step we should approximate the first derivative at the starting point of the curve in an arc-length parametric sense, i.e. $\hat{f}'(0) = \frac{f(p_1)-f(p_0)}{h}$, where $h = \frac{\text{length}(p_0,\ p_{n-1})}{n-1}$, f

denotes the actual, $\hat{f}$ the arc-length parameteric representation of the curve, and the function length$(p_i,\ p_j)$ gives the length of the curve in the interval $[p_i,\ p_j]$. In the second integration step, we can use the coordinate of the original curve: $\hat{f}(0) = f(p_0)$.

The Euler method is the simplest numerical integration algorithm, and even this produces good results. The equation for the first step is

$$\hat{f}'(x+h) = \hat{f}'(x) + h\hat{f}''(x),$$

and for the second step:

$$\hat{f}(x+h) = \hat{f}(x) + h\hat{f}'(x).$$

The second-order Runge–Kutta method uses a midpoint for better approximation. Let $n = 4k$, then for the first step the equation is

$$\hat{f}'(x+2h) = \hat{f}'(x) + 2h\hat{f}''(x+h),$$

and for the second step:

$$\hat{f}(x+4h) = \hat{f}(x) + 4h\hat{f}'(x+2h).$$

In both cases, we need to control the deviation from the original curve. This can be achieved by adding an extra term $c(x)$ to the equations, that pulls back the points when they get too close to a user-defined deviation tolerance. Imagine a band around the original curve with the tolerance as its breadth (Fig. 3). If the integrated curve gets close to the border of the band, it will be turned towards the middle.

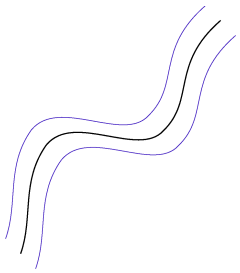


Fig. 3. Maximum deviation band around a curve.

$$c(x) = \min\left[1, \left(\frac{|d|}{m}\right)^2\right] \cdot d,$$

where $d = f(x) - \hat{f}(x)$ and m is the predefined tolerance.

In the third phase we apply a blending function to the two curves (integrated from the left and right side, respectively). We can use a suitable blending function[2], e.g. the Hermite blending function $\lambda(t) = 3t^2 - 2t^3$, or the 5th-degree polynomial $\lambda(t) = 6t^5 - 15t^4 + 10t^3$, so the blended points will be of the form

$$g(x) = (1 - \lambda(t))\hat{f}_{\text{left}}(x) + \lambda(t)\hat{f}_{\text{right}}(x),$$

where $t = \frac{x}{\text{length}(p_0,\ p_{n-1})}$.

Finally, an approximating spline is computed by fitting a least-squares B-spline over the points [7]. Usually the original knot vector can be passed to the fitting algorithm, and (e.g. curvature minimizing) smoothing terms can also be employed in the fitting process.

[2] A suitable blending function $\lambda(t)$ has the following properties: (i) $\lambda(0) = 0$ and $\lambda(1) = 1$ (ii) $\lambda^{(k)}(0) = 0$ and $\lambda^{(k)}(1) = 0$ for some $k = 1 \ldots$.

3.2 Surface Case

We can use the curve fairing method to fair individual isocurves of the surface. Let the B-spline surface be given in the form

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} \mathbf{d}_{ij} N_{i,k}^{U}(u) N_{j,l}^{V}(v),$$
$$(u,v) \in [u_{k-1}, u_{n+1}] \times [v_{l-1}, v_{m+1}],$$

where $U = (u_i)_{i=0}^{n+k}$ and $V = (v_j)_{j=0}^{m+l}$ represent the knot vectors. Then the u-isocurve for a fixed v parameter can be written as

$$C_v(u) = \sum_{i=0}^{n} \left(\sum_{j=0}^{m} N_{j,l}^{V}(v) \mathbf{d}_{ij} \right) N_{i,k}^{U}(u),$$

i.e. a B-spline curve with the same knot vector U, and with control points $\sum_{j=0}^{m} N_{j,l}^{V}(v)\mathbf{d}_{ij}$. We can apply steps 1–3 of the curve fairing method to get the faired points. These points can be collected to form two point clouds (one from the u-parametric and one from the v-parametric curves). Blending them and fitting a surface on the blended point set will result in a fair surface.

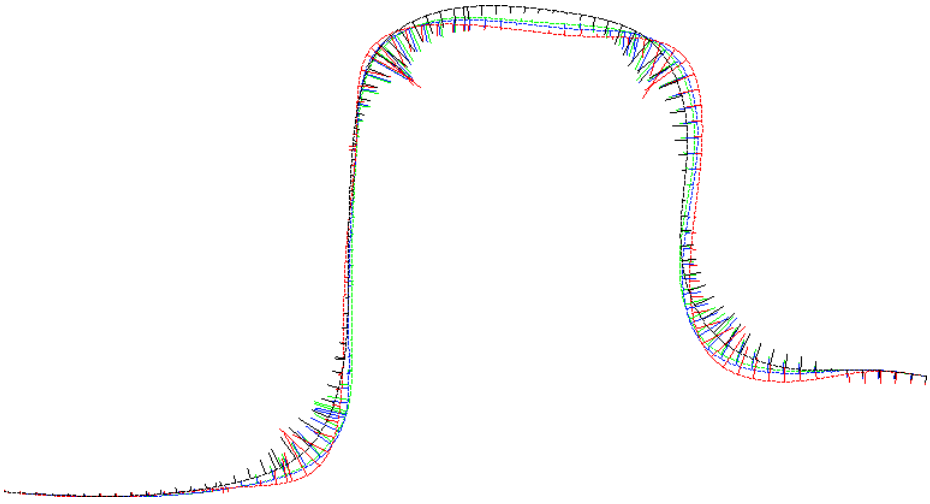
4 Test Results

Figure 4 shows a curve before and after fairing, using three different tolerances. We can see that fairness comes in exchange for a larger deviation. The change in fairness is more visible on the extrusion surfaces — the wiggles in the isophote lines are reduced in the tight and medium tolerance case, and totally smoothed out with loose tolerance.

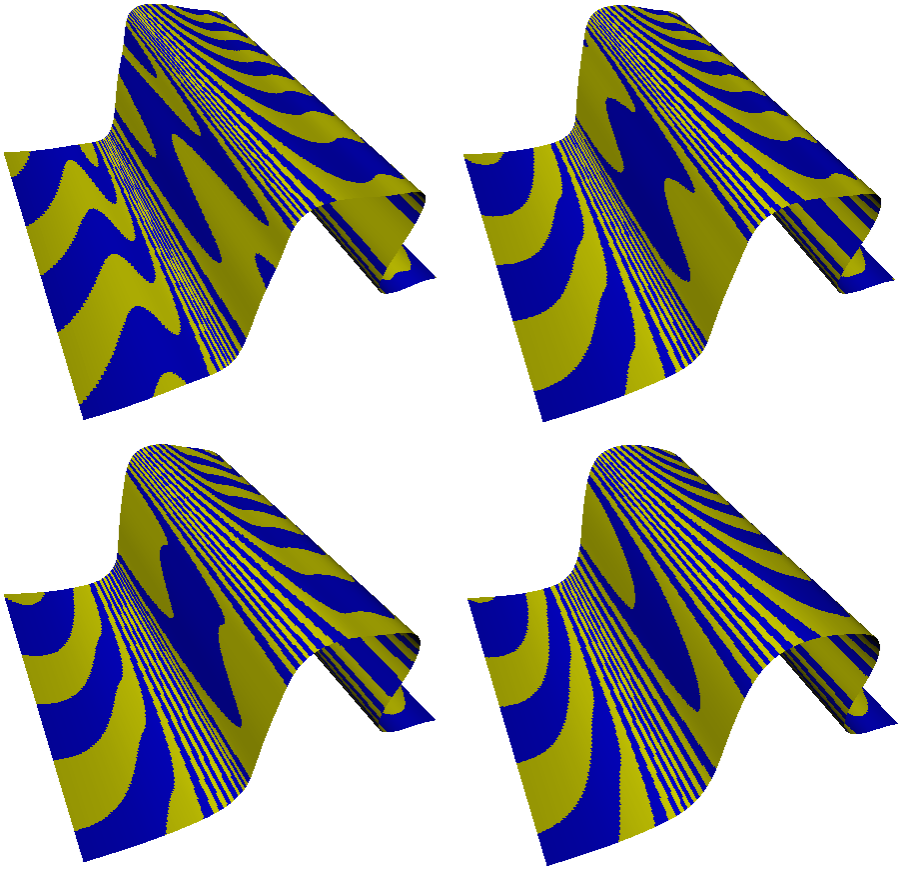
Figure 5 shows local fairing on a Fiat body part. The region on the right was selected for fairing, and the curvature became much more smooth as a result. Other regions were not affected (substantially).

Conclusions

A fast and local algorithm was introduced for both curve and surface fairing. It uses a parameterization-independent fairness measure based on a target curvature. Effectiveness is achieved by integration and fitting instead of iteration. User interaction may be necessary, but minimal. Our future research will be the fairing of multiple curves / surfaces together, while preserving or enhancing the continuity between them.

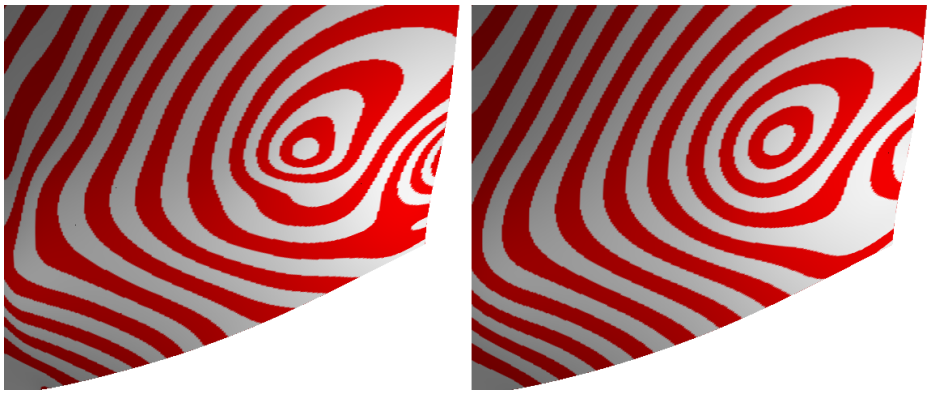


(a) Original: red; tight: blue; medium: green; loose: black.

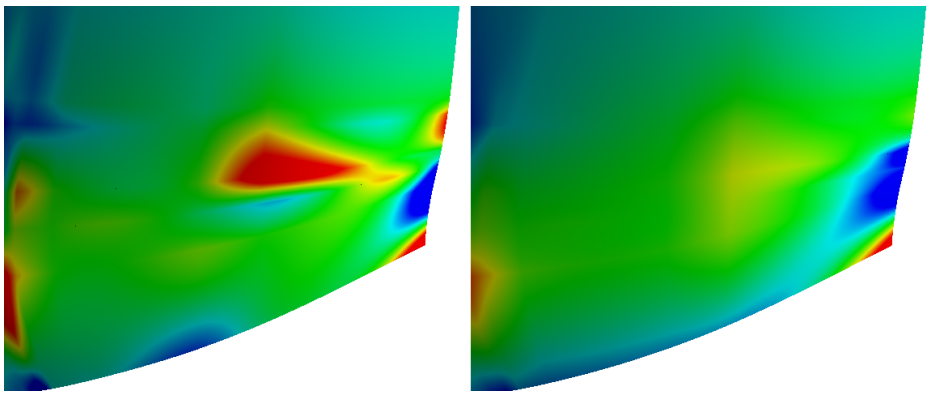


(b) Isophotes of the extrusion surfaces.

Fig. 4. Fairing a curve with tight, medium and loose tolerance.



(a) Isophote map.



(b) Mean map.

Fig. 5. Local fairing of a Fiat body part.

References

1. M. Eck, J. Hadenfeld, *Local Energy Fairing of B-Spline Curves.* Computing Supplement 10, pp. 129-147, 1995.
2. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide.* Academic Press, 5th Edition, 2002.
3. G. Farin, J. Hoschek, M.-S. Kim (Eds), *Handbook of Computer Aided Geometric Design.* North-Holland, 2002.
4. S. Hahmann, S. Konz, *Knot-Removal Surface Fairing Using Search Strategies.* Computer Aided Design 30, pp. 131-138, 1998.
5. E. T. Y. Lee, *Energy, fairness, and a counterexample.* Computer Aided Design 22(1), pp. 37-40, 1990.
6. H. P. Moreton, C. H. Séquin, *Minimum Variation Curves and Surfaces for Computer-Aided Geometric Design.* In: N. S. Sapidis (Ed.), Designing Fair Curves and Surfaces, pp. 123-159, SIAM, 1994.
7. L. Piegl, W. Tiller, *The NURBS Book*, Springer, 2nd Edition, 1997.

8. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C*. Cambridge University Press, 2nd Edition, 1992.
9. J. Roulier, T. Rando, *Measures of Fairness for Curves and Surfaces*. In: N. S. Sapidis (Ed.), Designing Fair Curves and Surfaces, pp. 75-122, SIAM, 1994.
10. P. Salvi, T. Várady, *Local Fairing of Freeform Curves and Surfaces*. In: Proceedings of the Third Hungarian Graphics and Geometry Conference, 2005.