Fair Curves and Surfaces

Ph.D. Dissertation

written by Péter Salvi

Supervisor: Dr. Tamás Várady

14th January 2012

Eötvös Loránd University, Budapest
PhD School of Computer Science
Numeric and Symbolic Calculus Program

Head of PhD School: Dr. András Benczúr
Program Director: Dr. Antal Járai

Contents

Chapter 1

Introduction

You shall be yet far fairer than you are.

— Shakespeare: Antony and Cleopatra, Act I, Scene 2

Creating aesthetically pleasing, *fair* curves and surfaces is a principal issue of Computer Aided Geometric Design (CAGD). It is especially so for automobiles and other consumer goods, such as household appliances, where sales largely depend on the appearance of the products. Using today's technology, professional designers still have to spend many days manually adjusting control points in order to create Class A surfaces with smooth connections. This research is aimed at developing new, automatic or semi-automatic methods for creating fair geometries.

Fairness, however, is an elusive concept. There is no exact mathematical definition, and different applications may have different requirements. Still, researchers agree that evenly distributed curvature is favorable [6, 29]. It is not sufficient, though, just to smooth the surface: we have to preserve the highly curved features of the original model. Locality and deviation control are thus important factors in fairing algorithms.

Most real-life models, where appearance counts, contain smooth edges, usually defined by fillet surfaces and corner patches. These need to be continuously joined to the primary surfaces, so besides individual fairing, care should be taken to enhance the connections between surfaces as well. In this context, "smooth connections" usually mean G^2 continuity. G-continuity is

less strict than C-continuity: it is sufficient that there exists a parameterization of the surfaces such that C-continuity applies [6]. Informally speaking, this means that for G^1 connections the two surfaces share a common tangent plane at every point of the boundary curve, while for G^2 connections they also have matching surface curvatures. These are often referred to as *tangent continuity* and *curvature continuity*, respectively.

Fair curves and surfaces are crucial in many practical applications. One of these is Digital Shape Reconstruction (DSR), which deals with the creation of geometric models based on measured data. Its workflow comprises triangulation, segmentation, classification, surface fitting and surface improvement [40]. It is a very complex procedure, where even slight measurement errors can cause significantly decreased quality. This is why fairing methods are indispensable in DSR. So-called *variational* methods can be integrated into the surface approximation process, playing an important part in the surface fitting phase. In contrast, *post-processing* fairing algorithms are applied at the final improvement stage.

Another fundamental application of fairing is in surface design. There are various ways to build a model; one approach is to first create a curve network representing the edges and feature lines of the actual object. The curves may come from several sources, like traditional blueprints, 2D sketches, or directly by some GUI interface. These curves need to be smooth, in order to be able to fit high-quality surfaces onto them at a later stage.

In curvenet-based design it is also crucial to adopt the most suitable types of surfaces. Transfinite interpolation surfaces are particularly suitable for this purpose, and their application can be considered as a kind of fairing — after all, one could define transfinite interpolation surfaces as "surfaces with fair interior that maintain boundary constraints". In the light of this, though not fairing per se, I have dedicated a chapter to the creation of these surfaces.

1.1 Interrogation Tools

There are various visualization tools that help a designer to find low-quality regions in a curve or surface. These display the geometry in such a way, that

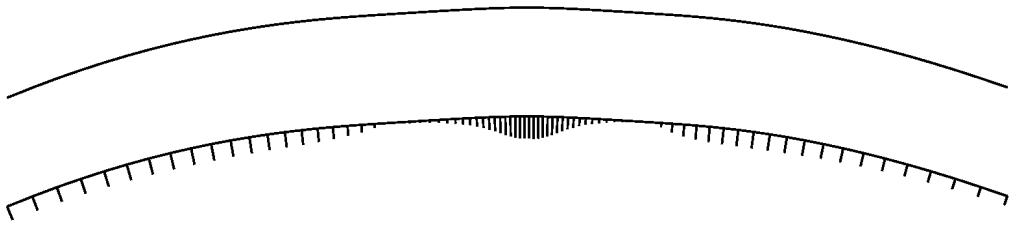


Figure 1.1: A curve and its curvature comb

flaws of a given property become more discernible, or even obvious. One particularly useful tool for examining curves is the *curvature comb*, where we place normal-directed "teeth", proportional to the curvature at their footpoints. The curve in Fig. 1.1, for example, looks quite smooth at first sight, but if we look at the comb, it reveals that the curvature is uneven and it should be faired.

Perhaps the most well-known method for evaluating a surface, called *contouring*, is to intersect it with parallel planes and inspect its cross section curves (Fig. 1.2). These curves, in turn, can be examined using curvature combs as explained above. Another variant, called *slicing map*, uses a texture of two colors alternating at the cross section curves. The top example in Fig. 1.3 shows two surfaces: the coloring emphasizes the small bumps on the surfaces, which appear as wiggles, and the sudden breaks in the lines also show the error in G^1 continuity. The bottom example in Fig. 1.3 shows a fair version of these surfaces with G^1 continuity.

Environment maps can also help in finding small artifacts on the surface (Fig. 1.4). However, most of the time we need to study the curvature as well. There are several *curvature maps* for this purpose, displaying the principal curvatures, their mean (*mean curvature map*) or their product (*Gauss curvature map*). These encode the curvature information using a color range — here we use the convention that blue means negative, green means (approximately) zero, and red means positive. Other colors of this spectrum are interpolated, see for example Fig. 1.5.

Last but not least, *reflection lines* or *isophotes* can also be applied to detect regions of low surface quality and analyze the connection between

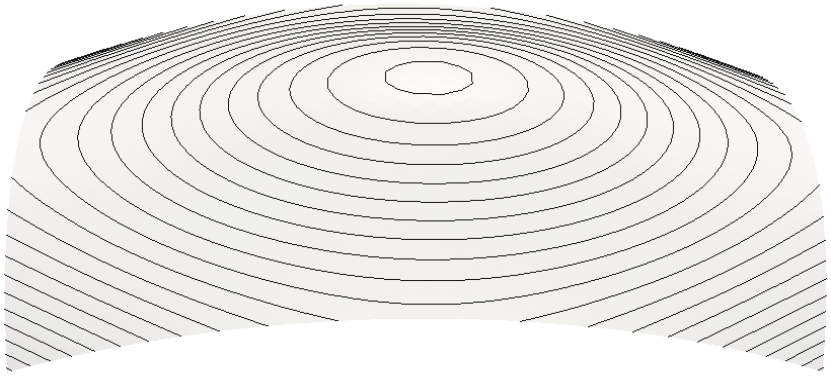


Figure 1.2: Cross section curves of a surface.

patches. Isophotes can be thought of as the reflections of parallel neon lights; it is a texture alternating two colors, similarly to the slicing map, but the color change is determined by the angle between the surface normal and a vector originating from a reference point (see Fig. 1.6).

As the slicing map can expose problems of tangent plane continuity, reflection lines can expose changes in surface curvature. Fig. 1.7 shows five surfaces, four fillets connecting to a corner patch. The isophotes do not match at the left boundary, because the connection is only C^0. The other boundaries have matching lines, but the stripes have a sudden turn, showing that there is only G^1 continuity. Smooth lines require G^2 continuity, like in the interior of the surfaces. Furthermore, even with G^2 continuity, uneven curvature distribution can be spotted by wiggles, such as on the left surface. Isophotes are invaluable for assessing surface quality and will be used throughout this document.

Isophotes can also be useful for evaluating curves, as they can be applied to extrusion surfaces. Fig. 1.8 shows the extrusion of the curve in Fig. 1.1.

1.2 Overview

Each of the following chapters covers a separate topic, and thus begins with an introduction to the problem statement and previous research results before

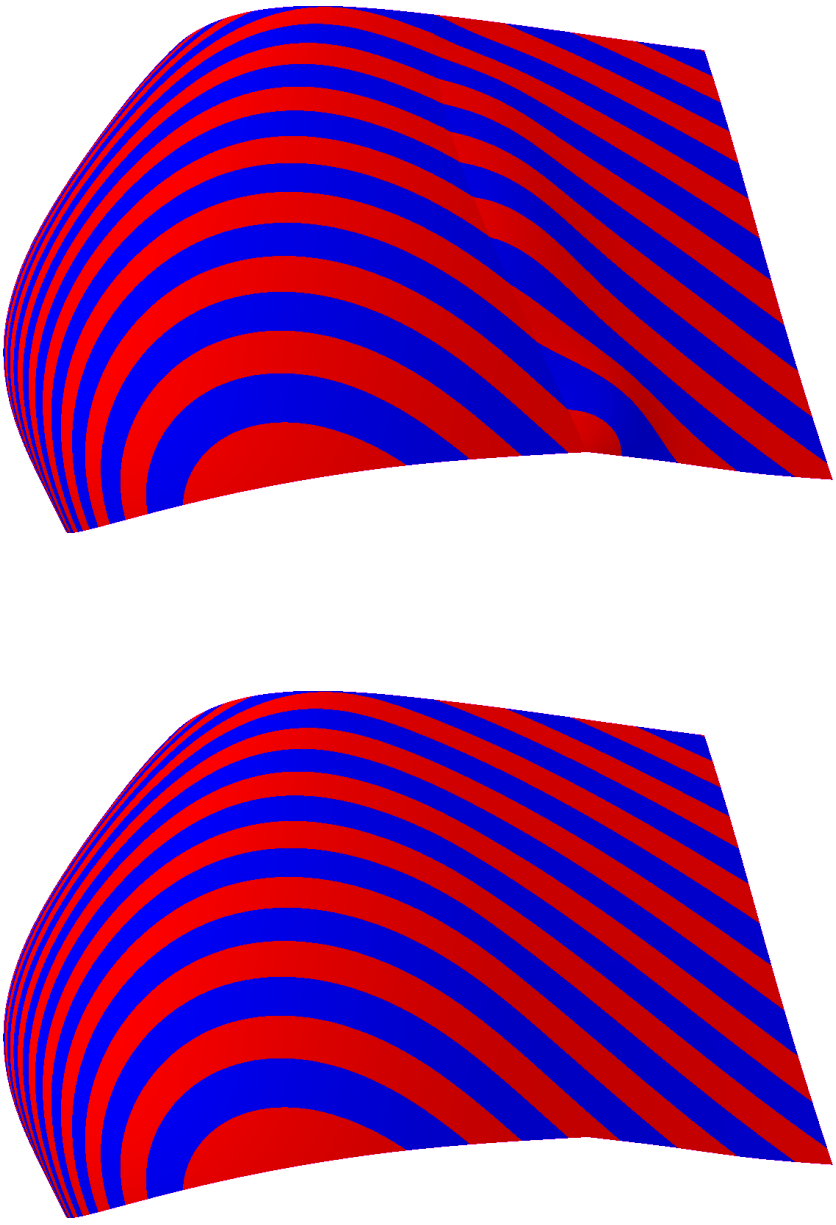


Figure 1.3: Slicing map of two surfaces with C^0 (top) and G^1 (bottom) continuity.

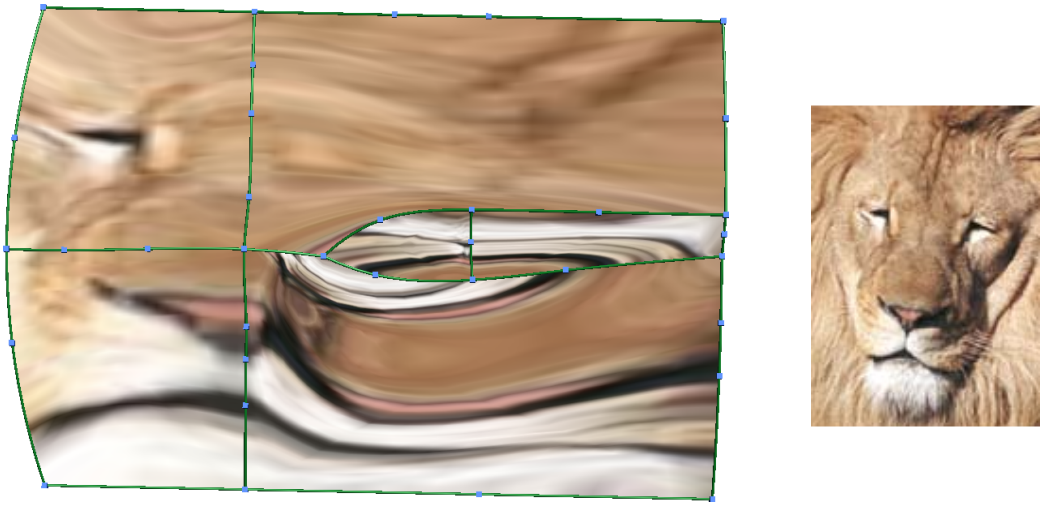


Figure 1.4: Surfaces with environment mapping (left) and the texture used (right).

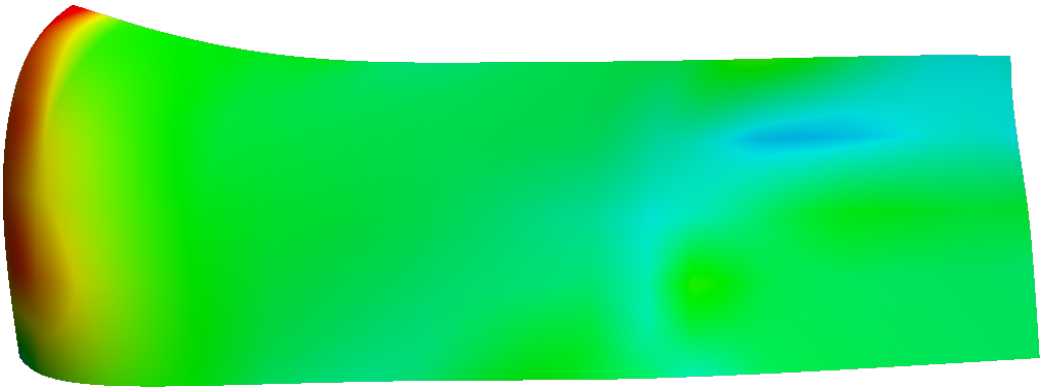


Figure 1.5: Mean curvature map of a surface.

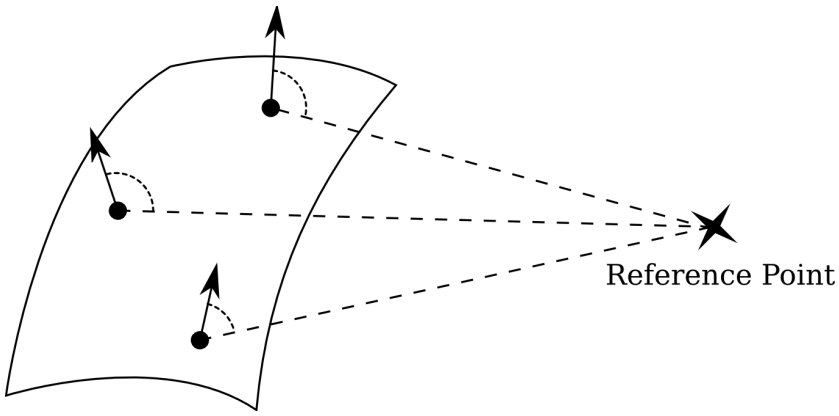


Figure 1.6: Computing isophotes.

Figure 1.7: Five surfaces with isophote visualization.

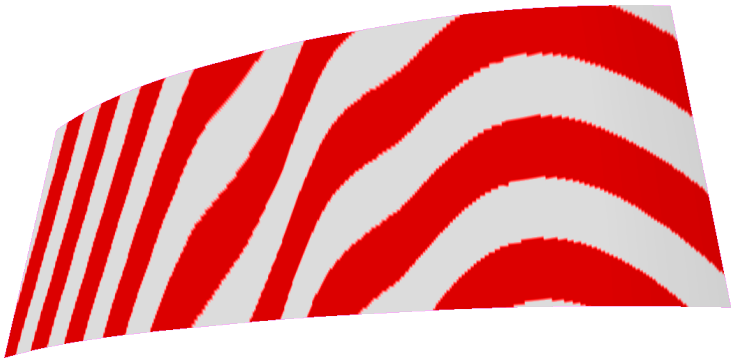


Figure 1.8: Extrusion surface with isophotes.

introducing new developments. In Chapters 2 and 3 new fairing algorithms are proposed that operate on conventional parametric curves and surfaces. Since today's CAGD softwares predominantly employ B-splines, all the equations are written using this formulation, but the methods could be applied to other parametric constructions. In Chapter 2 a new procedure is defined to evaluate the quality of curves and surfaces, and new fairing routines are introduced. In Chapter 3 the problem of fairing multiple-patch models is examined and a novel fairing approach is described that handles continuity constraints. In Chapter 4 we study curve network-based design using transfinite interpolation surfaces. Two new patch representations are proposed: the first can be regarded as the direct generalization of the Coons patch, and the second is a natural application of curved side interpolants. Further enhancements are also introduced, such as non-regular domains and new parameterizations. A summary of the results concludes the dissertation.

1.3 Notations

The following notations are going to be used:

$\underline{d}$, $\underline{e}$, $\underline{n}$, $\underline{v}$, $\underline{w}$	vectors
c, α, β, γ, ξ	constants
κ	curvature (with indices: principal curvatures)

ϕ, χ, ψ	angles
P, Q	points
A, B	matrices (elements are in lowercase, e.g. a_{ij})
E	a fairness measure or energy
$C(t)$, $S(u,v)$	a parametric curve or surface, respectively
$R(s,d)$, $C(s,d)$	ribbons (side interpolants)
$Q(s,d)$	corner correction patch
T, U, V	knot vectors (knots are in lowercase, e.g. t_i)
i, j, k, l	indices
m, n, r	limits
p, q	degrees
u, v, s, d, t	parameters
$N_i^{T,p,(k)}(u)$	k^{th} derivative of the i^{th} p-degree B-spline basis of T at u

Coordinates are written as superscripts, while elements are subscripts, e.g. n^x and P^y, but v_i and a_{ij}.

The above notations are used as a rule of thumb only. When deviations occur, the alternative meaning will be evident from the context.

Chapter 2

Fairing based on Target Curvature

There is no such thing as the "best" fairness measure. Different applications require different approaches, and this is the reason why an abundance of measures coexists in the literature. One well-known method has its roots in 18th century shipbuilding technology, where in order to draw a smooth curve, metal weights were placed at the interpolation points and a flexible spline was spanned between them. The resulting curve minimizes the strain energy, yielding the measure

$$E_{\text{strain}}(C) = \int (\kappa(t))^2 \mathrm{d}t.$$

This favors low mean curvature, while giving a penalty to extreme values by squaring [29].

Computing the curvature can be difficult, so it is often replaced by a simpler, parameter-dependent formula:

$$\hat{E}_{\text{strain}}(C) = \int (C''(t))^2 \mathrm{d}t.$$

The drawback of this simplification is that the second derivative and the curvature may be far apart, when the curve's parameterization is substantially different from the arc-length parameterization, and thus any fairing

algorithm based on this quantity may have unexpected results. Still, this is one of the most popular measures today.

One classical definition of fairness by Farin and Sapidis [6] is as follows:

> A curve is fair if its curvature plot is continuous and consists of only a few monotone pieces.

In other words, sudden changes in the curvature and inflections are considered unfavorable. The energy associated with Minimum Variation Curves, proposed by Moreton and Séquin [23, 24], respects this assumption:

$$E_{\mathrm{MVC}}(C) = \int (\kappa'(t))^2 \mathrm{d}t.$$

However, this measure is much more difficult to compute.

All of these measures have their counterparts for surfaces. For strain energy there is the *thin plate energy*:

$$E_{\mathrm{th.pl.}}(S) = \iint_S a\left[(\kappa_1(u,v))^2 + (\kappa_2(u,v))^2\right] + 2(1-b)\kappa_1(u,v)\kappa_2(u,v)\mathrm{d}u\mathrm{d}v,$$

where usualy $a = 1$ and $b = 0$ or 1 [11]. Its simplified version is

$$\hat{E}_{\mathrm{th.pl.}}(S) = \iint_S S_{uu}^2 + 2S_{uv}^2 + S_{vv}^2 \mathrm{d}u\mathrm{d}v.$$

This latter energy is parameterization-dependent, which renders it unstable, but still it is the most often used fairness measure for surfaces.

There is also the energy for Minimum Variation Surfaces [24], which uses the directional derivatives of the principal curvatures:

$$E_{\mathrm{MVS}}(S) = \iint_S \left(\frac{\partial \kappa_1(u,v)}{\partial \underline{e}_1(u,v)}\right)^2 + \left(\frac{\partial \kappa_2(u,v)}{\partial \underline{e}_2(u,v)}\right)^2 \mathrm{d}u\mathrm{d}v,$$

where $\underline{e}_1(u,v)$ and $\underline{e}_2(u,v)$ are the principal directions at the (u,v) parameter. While theoretically it is a very good measure of fairness, it cannot be used in many applications due to its computational complexity.

Another way to look at these energies is to find a perfect curve or surface

(with zero energy): for the strain measure, this would be a line, since it penalizes curvature, but the MVC energy uses the variation of curvature, so the zero-energy curve would be a circular arc. Similarly, the thin plate energy gives zero for a plane and the MVS measure for a sphere. Roulier and Rando [29] give a very comprehensive, detailed review on these and other fairness measures and their effects.

As it was outlined in Chapter 1, fairing algorithms can be divided into two types: variational and post-processing methods. We will only deal with the latter approach here, as it is more useful in the DSR context. Fairing algorithms are often derived from fairness measures, for example by iteratively moving the control points of a curve or surface in order to minimize the chosen energy. The curve fairing method of Eck and Hadenfeld [5] fixes all control points but one, and locally minimizes the $\hat{E}_{\text{strain}}$ measure (or its variant using the third derivative), while keeping the distance from the original curve under a given tolerance using the convex hull property. This technique has a natural extension to surface fairing as well.

One of the simplest, but widely used curve fairing algorithm is *knot removal and reinsertion* (KRR), originally conceived by Kjellander and later optimized by Farin et al. [7]. This method uses the fact that (i) removing a knot from a B-spline curve's knot vector makes it smoother (but of course, the details around the knot are lost), and that (ii) a new knot can be added without changing the shape of the curve. The idea is to remove a knot and then reinsert it, resulting in a smoother curve with the same degree of freedom. The removal step can be solved by moving only one control point. Alternatively, the deviation from the original control polygon can be minimized, if we allow the movement of multiple points. The iteration of this process, combined with some heuristics to choose the next knot based on a fairing measure [13, 8], is a very efficient algorithm both in computational time and in quality. Unfortunately, its extension to surfaces [12] is much less usable, since it can only guarantee fairing in one parametric direction, which is not satisfactory in real-life applications.

In Section 2.1, I will propose a new method for fairness evaluation, that tries to overcome the difficulties of the above mentioned measures — it is

based on the distribution of curvature, but it is still cheap to compute [31, 34]. In Sections 2.2 and 2.3, I will present curve and surface fairing algorithms that are based on this measure [34, 30]. Some concluding thoughts will end the chapter.

2.1 Fairness Evaluation

We have seen that curvature plays a central role in the fairness of a curve — it is even used as a visualization tool. A good curvature comb indicates a fair curve, and this gives the idea to fair the curvature comb first, yielding an "ideal" or *target curvature*, and then compute the deviation from the original curvature values. More precisely, we take the curvatures, which is practically the same as the evolute, since $\kappa = 1/\rho$, and smooth it. Then the fairness measure is

$$E_{\text{curve}} = \int (\kappa(t) - g(t))^2 \mathrm{d}t,$$

where $g(t)$ is the smooth target curvature. To make the calculation easier, the curvature is only computed at sampled points, so the discretized measure becomes

$$\tilde{E}_{\text{curve}} = \sum_i (\kappa(t_i) - g(t_i))^2.$$

The only question that remains is how to smooth the curvature values. Various simple and fast methods can be effectively used, for example averaging consecutive sampled curvature values. Global averaging can remove parts of the curvature that represent features, so the user should be allowed to restrict the smoothed area or to edit the target curvature manually. Another possibility is to fit a curve over the plot of sampled curvatures — minimizing the curvature of the fitted curve, in turn, gives even nicer results.

Fig. 2.1 shows a curve with its control points, curvature comb and target curvature. We can see that the target curvature is much smoother than the original comb; the fairness energy is the summed deviation of these two values. One particularly interesting feature of this fairness evaluation is that the target curvature depends on the original curve — there is no fixed zero-

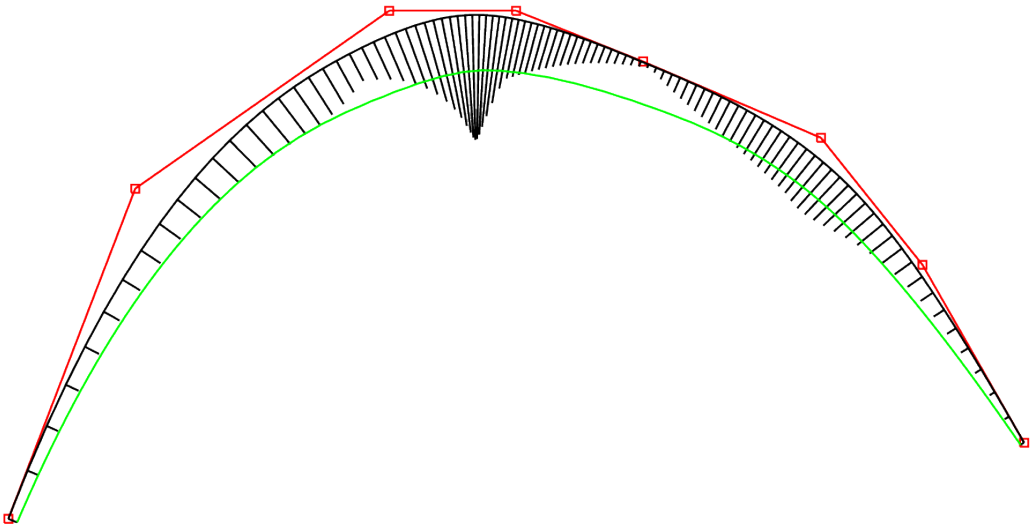


Figure 2.1: Curve with curvature comb (black) and target curvature (green).

energy curve, but for each curve we construct an "idealized" shape similar to the original.

The same concept also works for surfaces, just substituting the curvature with a combination of principal curvatures. For example,

$$\tilde{E}_{\text{surface}} = \sum_i \sum_j \left((\kappa_1(u_i, v_j) - g_1(u_i, v_j))^2 + (\kappa_2(u_i, v_j) - g_2(u_i, v_j))^2 \right),$$

is a meaningful fairness measure, where $g_1(u, v)$ and $g_2(u, v)$ are target principal curvatures, depicted in Fig. 2.2.

2.2 Curve Fairing Methods

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 [21], so we can integrate it twice (using appropriate boundary conditions) to reconstruct the curve. Discrete integration provides us with points we can use for fitting, eventually resulting in a close approximation of the original

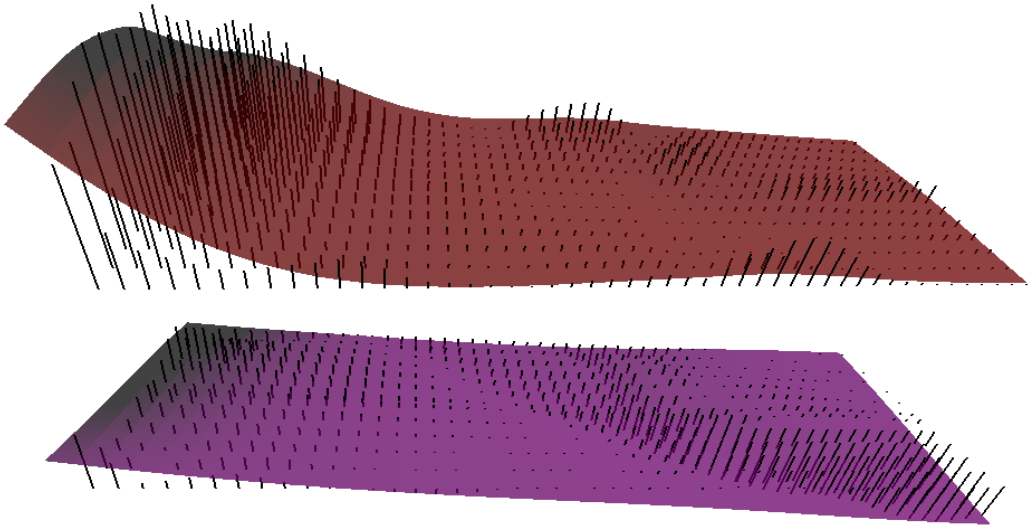


Figure 2.2: Target principal curvatures of a surface: g_1 (top) and g_2 (bottom). Black teeth represent sampled curvature values.

curve. Consequently, using the target curvature instead of the real curvature, a faired curve can be constructed (we assume that at both ends G^1 continuity is required, so there are fixed starting values for position and tangent). Unfortunately, parameterization is arbitrary and can be substantially different from arc-length parameterization.

However, suitable sampling can provide something similar, which we will call *pseudo arc-length parameterization*. In this way, the target curvature can be integrated twice to find discrete points of the faired curve directly. Finally, these points can be used to fit a curve, which is expected to be better than the original. The use of fitting harms locality, of course, but in return we get a fast fairing method based on the variation of curvature.

For a more robust operation, numerical integration should be done starting from both ends of the curve, and the two curves obtained will be blended together to yield a single curve. Another important feature of every fairing algorithm is deviation control, so this should be incorporated into the integration step as well.

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

1. Define a pseudo arc-length parameterization for n points.

2. Create a target curvature.

3. Integrate twice starting at the "left" end to the "right" and vice versa, with deviation control.

4. Blend the results.

5. Fit 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. The target curvature at these parameters can be regarded as the uniformly sampled norms of second derivatives for an arc-length parameterized curve with the same shape. Let t_k denote the parameter of the k^{th} parameter point.

The third step 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 a starting point, as well as the direction at every point of the second derivative. In arc-length parameterization the second derivative is in the normal direction, which can easily be calculated. In the first integration step we approximate the first derivative at the starting point of the curve in an arc-length parametric sense, i.e., $\hat{C}'(0) = \frac{C(t_1)-C(t_0)}{h}$, where $h = \frac{l}{n-1}$, C denotes the actual, $\hat{C}$ the arc-length parameteric representation of the curve, and l is the total length of the curve. In the second integration step, we can use the coordinates of the original curve: $\hat{C}(0) = C(t_0)$.

The Euler method [28] is the simplest numerical integration algorithm, and even this produces good results. The equation for the first step is

$$\hat{C}'(s+h) = \hat{C}'(s) + h\hat{C}''(s),$$

and for the second step:

$$\hat{C}(s+h) = \hat{C}(s) + h\hat{C}'(s).$$

The second-order Runge–Kutta method [28] uses a midpoint for better

approximation. Let $n = 4k$, then for the first step the equation is

$$\hat{C}'(s+2h) = \hat{C}'(s) + 2h\hat{C}''(s+h),$$

and for the second step:

$$\hat{C}(s+4h) = \hat{C}(s) + 4h\hat{C}'(s+2h).$$

Another alternative is Numerov's method [25], which gives an even better approximation of the second-order integral:

$$\hat{C}(s+h) = 2\hat{C}(s) - \hat{C}(s-h) + \frac{h^2}{12}\left(\hat{C}''(s+h) + 10 \cdot \hat{C}''(s) + \hat{C}''(s-h)\right).$$

In this case the $C(t_1)$ position is also needed, which should cause no problem, assuming an appropriately small h and a fixed tangent at the starting point.

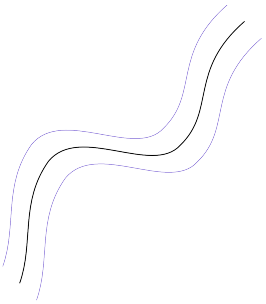


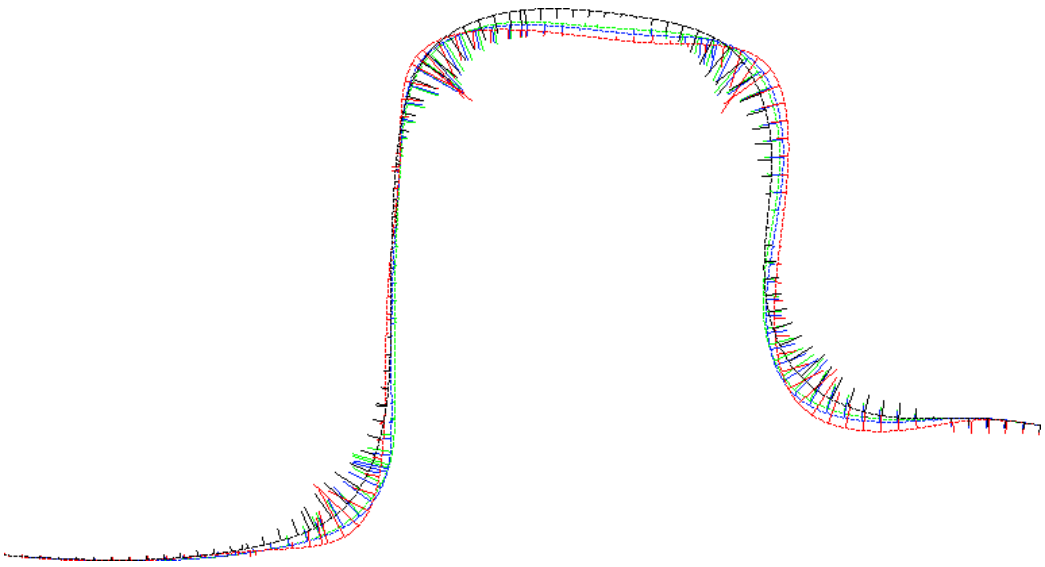
Figure 2.3: Maximum deviation band around a curve.

Regardless of the integration method, we need to control the deviation from the original curve. This can be achieved by adding an extra term $c(s)$ to the equations, that pulls back the points when they get too far from the original curve, i.e., they got too close to a user-defined deviation tolerance. Imagine a band around the original curve with the tolerance as its width (Fig. 2.3). If the integrated curve gets close to the border of the band, it will be turned towards the middle.

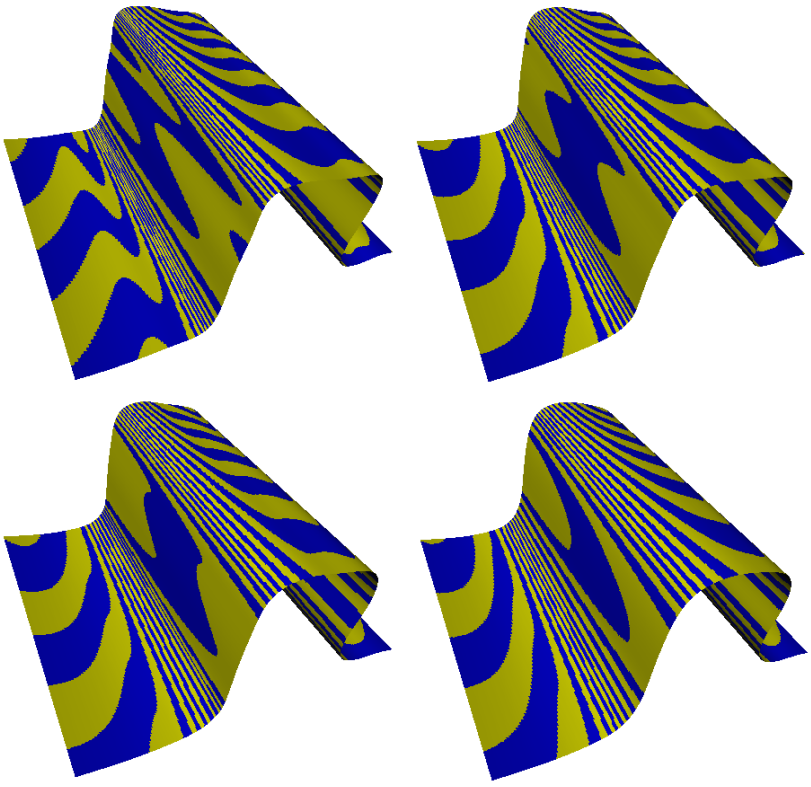
$$c(s) = \min\left[1, \left(\frac{|d(s)|}{\alpha}\right)^2\right] \cdot d(s),$$

where $d(s) = C(s) - \hat{C}(s)$ and α is the predefined tolerance.

In the fourth step we apply a blending function to the two curves (integrated from the left and right side, respectively). This is essential, as the integration step does not guarantee that the point series arrives at the origi-



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



(b) Isophotes of the extrusion surfaces.

Figure 2.4: Fairing a curve with tight, medium and loose tolerance.

nal curve's endpoint, which would be a natural requirement. Blending retains the proper positions (and tangents), using a suitable blending function[1], e.g. the Hermite blending function $\lambda(u) = 3u^2 - 2u^3$, or the 5th-degree polynomial $\lambda(u) = 6u^5 - 15u^4 + 10u^3$. Here $u = s/l$, i.e., the arc-length parameterization normalized to $[0, 1]$. Consequently, the blended points will be of the form

$$P(u) = (1 - \lambda(s/l))\hat{C}_{\text{left}}(s) + \lambda(s/l)\hat{C}_{\text{right}}(l - s).$$

Finally, an approximating spline is computed by fitting a least-squares B-spline over the points. Usually the original knot vector can be passed to the fitting algorithm.

Figure 2.4 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.

2.3 Surface Fairing Methods

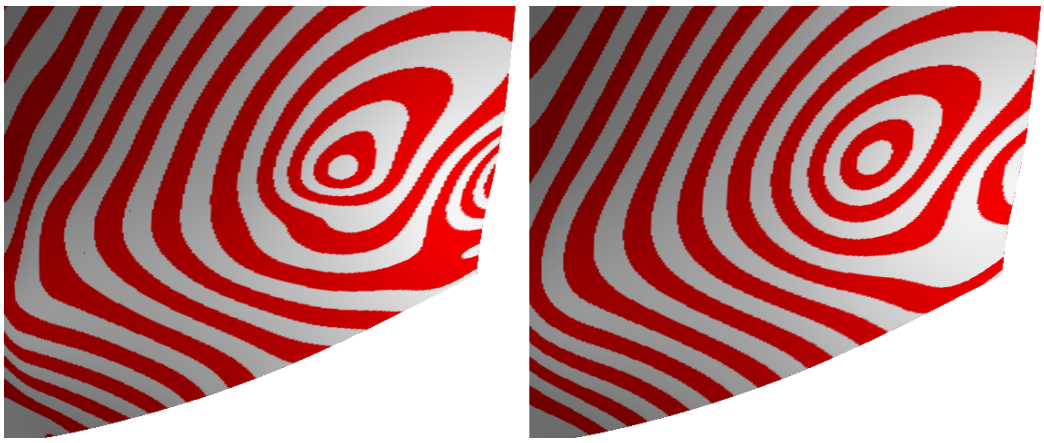
A direct extension of the curve fairing method is, regrettably, infeasible, since the isometric parameterization for a surface cannot be constructed (generally), not even in an approximate sense. Nevertheless, several options exist for generalizing the target curvature-based approach.

2.3.1 Extension by Isocurves

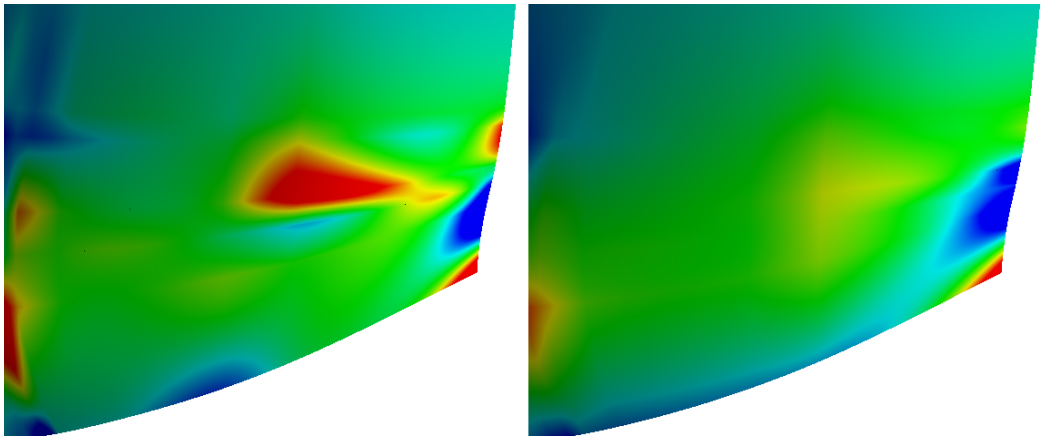
The curve fairing method can be applied to individual isocurves of the surface. Let the B-spline surface be given in the form

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

[1] The blending function $\lambda(u)$ should have the following properties: (i) $\lambda(0) = 0$ and $\lambda(1) = 1$ (ii) $\lambda^{(k)}(0) = 0$ and $\lambda^{(k)}(1) = 0$ for some $k = 1 \ldots$.



(a) Isophote map.



(b) Mean map.

Figure 2.5: Local fairing of a Fiat car panel.

Then the u-isocurve for a fixed v_0 parameter can be written as

$$C_{v_0}(u) = \sum_{i=0}^{n} \left(\sum_{j=0}^{m} N_{j,l}^{V}(v_0) P_{ij} \right) N_{i,k}^{U}(u),$$

i.e., a B-spline curve with the same knot vector U, and with control points $\sum_{j=0}^{m} N_{j,l}^{V}(v_0) P_{ij}$. We can apply steps 1–4 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), both of them retaining the original constraints by the four boundaries. Blending them and fitting a surface on the blended point set will result in a fair surface.

Figure 2.5 shows a local fairing on a Fiat car body part. The region on the right side of the surface was selected for fairing, and as the isophote lines show, the distribution of the curvature has become much more smooth due to fairing. Other regions were not affected (substantially).

2.3.2 Extension by PDE

Even though not as versatile as curvatures, parametric second derivatives still reflect surface quality. Using the same pattern as before, *target second derivatives* g_{uu} and g_{vv} can be computed on a uniform grid, which leads to the finite difference solution of the Poisson-equation

$$\frac{\partial^2 \hat{S}(u,v)}{\partial u^2} + \frac{\partial^2 \hat{S}(u,v)}{\partial v^2} = \Delta^{\text{target}}(u,v),$$

where $\Delta^{\text{target}} = g_{uu} + g_{vv}$ is the target Laplacian. In most applications, the edge curves of the surface should remain constant, which gives us the Dirichlet boundary conditions $\hat{S}(u,v) = S(u,v)$, when either u or v is extremal.

The equation can be approximated by Gauss–Seidel iteration [28]:

$$\begin{aligned} \hat{S}_{n+1}(u_j, v_k) \;=\; & \frac{1}{4}\Big(\hat{S}_{n+1}(u_j, v_{k-1}) + \hat{S}_{n+1}(u_{j-1}, v_k) \\ & \quad + \hat{S}_n(u_j, v_{k+1}) + \hat{S}_n(u_{j+1}, v_k)\Big) \\ & - h^2 \Delta^{\text{target}}(u_j, v_k), \end{aligned}$$

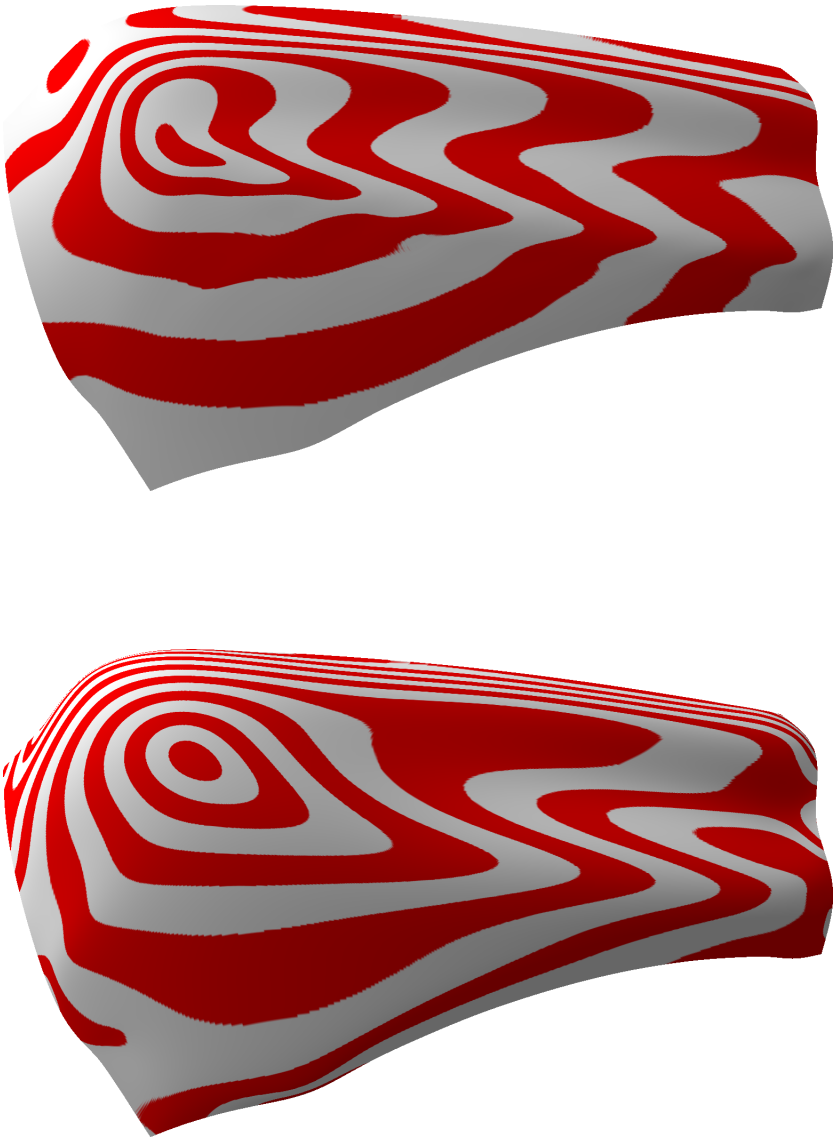


Figure 2.6: Surface fairing by PDE-based extension of the target curvature algorithm.

where h is the parameter difference between grid points. Sampled points of the original surface can be used as starting values for the iteration. The equation above can be supplemented by deviation control, as suggested earlier. Introducing the notation $P^n_{jk} = \hat{S}_n(u_j, v_k)$, the original points can be written as P^0_{jk}. Then the required post-processing step is

$$P^{n+1}_{jk} \leftarrow P^0_{jk} + \underline{v}\left(1 - \left(\frac{\|\underline{v}\|}{\alpha}\right)^2\right),$$

where $\underline{v} = P^{n+1}_{jk} - P^0_{jk}$ and α is the maximum deviation. Fig. 2.6 shows an example.

2.3.3 Fairing by Curvature Approximation

While the algorithm in the previous section is fast and simple, it uses second derivatives as an approximation of curvature. This is a popular practice, but we can do better. Given a reference surface R and a twice differentiable scalar function $\tilde{h}$ defined on it, Greiner [10] shows that the Hessian matrix of $h = \tilde{h} \circ R$ is

$$\mathrm{Hess}_R(h) = \left(\sum_l g^{kl}(\partial_j\partial_l h - \sum_i \partial_i h \Gamma^i_{jl})\right)_{kj},$$

where ∂_i is the partial derivative by the i^{th} argument, (g^{ij}) is the inverse of the first fundamental form of R, and $\Gamma^i_{jl} = \sum_m g^{im}\langle\partial_j\partial_l R, \partial_m R\rangle$ (every index can take the values 1 and 2). Note that both Γ and g can be computed in advance. The paper also shows that if we use the coordinate functions R_c ($c \in [1\ldots3]$) as h, we have

$$\mathrm{Hess}_R(R_c) = \frac{1}{EG - F^2}\begin{bmatrix} G & -F \\ -F & E \end{bmatrix}\begin{bmatrix} L & M \\ M & N \end{bmatrix} n_c,$$

where $\underline{n}$ is the surface normal. This has several nice properties, for example it is easy to see that

$$\begin{aligned}\sum_c \operatorname{trace}\left(\operatorname{Hess}_R(R_c)\right)^2 &= (\kappa_1^R + \kappa_2^R)^2,\\ \sum_c \det\left(\operatorname{Hess}_R(R_c)\right) &= \kappa_1^R \cdot \kappa_2^R,\end{aligned}$$

where κ_1^R and κ_2^R are the principal curvatures of R. The claim is that using S_c instead of R_c will result in good approximations of the curvatures of S:

$$\begin{aligned}\sum_c \operatorname{trace}\left(\operatorname{Hess}_R(S_c)\right)^2 &\approx (\kappa_1^S + \kappa_2^S)^2,\\ \sum_c \det\left(\operatorname{Hess}_R(S_c)\right) &\approx \kappa_1^S \cdot \kappa_2^S.\end{aligned}$$

The reader can consult the original paper [10] for more details.

In our case, we already have a very good reference surface — the original surface itself. Following the mechanism of the curve fairing algorithm, the goal is to set up a target curvature and find a surface with similar curvature. The general scheme would then be as follows:

1. Sample the original surface at intervals.
2. Compute Γ and g in these positions.
3. Set up a target curvature.
4. Minimize the deviation from the target curvature.

We can write up an equation system that depends linearly on the control points of S, so it will be easy to minimize. One option is to create a target mean curvature by smoothing the traces of Hessian matrices in the sampled points. Another alternative is to average every element of the Hessian matrices over the sampled points. Since the computation of the Hessian matrix by the control points is linear, these lead to overdefined linear equation systems, that can be solved in a least-squares sense. An example is shown in Fig. 2.7. The advantage of this method is that it is independent of the parametric di-

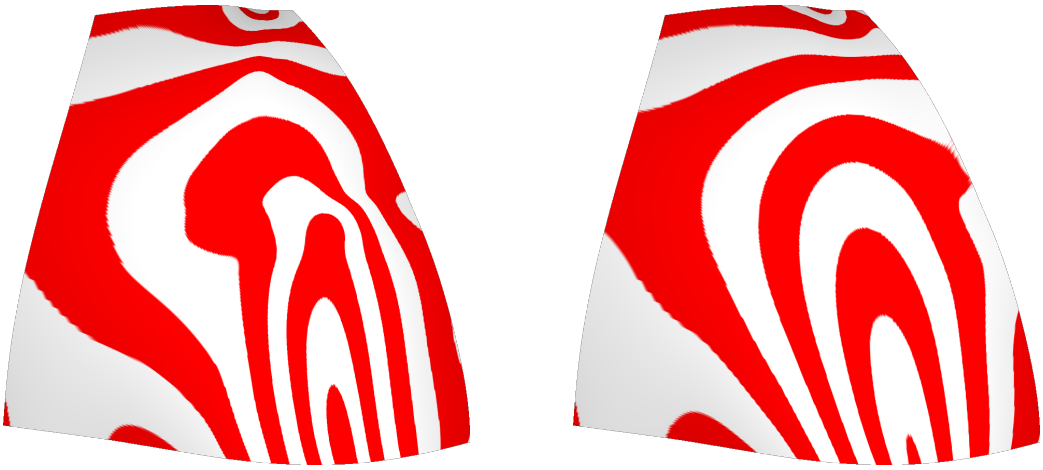


Figure 2.7: Isophotes of a car body panel before (left) and after (right) fairing by the curvature approximation method

rections, yet works with the surface directly. However, we have to minimize a fairly large equation system, which makes its computational cost quite high.

Conclusions

Fast and local algorithms were introduced for both curve and surface fairing, involving a parameterization-independent fairness measure based on a target curvature. Efficiency was achieved by integration and fitting instead of the conventional iterative approaches. The fairing method was also extended to surfaces in three different ways: (i) fitting a surface on the accumulated points of individual isocurve fairing, (ii) solving a simplified problem as a PDE, and (iii) using curvature approximation. Each of the algorithms has its strengths and weaknesses, but the examples show that all of them considerably enhance surface quality.

Chapter 3

Hierarchical Fairing with Constraints

The previous chapter presented fairing methods that work well when applied to primary surfaces independently of their environment; however, when a complete object composed of many connected surfaces needs to be faired, not only the smoothing of the individual surfaces is needed, but the hierarchy of surfaces must be taken into account as well. The fairing of dependent connection surfaces, such as fillets or corner patches, thus poses a new problem.

Various methods have been published that deal with this issue using the variational paradigm (see for example Lai et al. [20] or Hsu et al. [16]), but a solution to constrained fairing of multiple surfaces in the post-processing context, which seems to be more suitable for Digital Shape Reconstruction, is not known to the author. This chapter aims to fill this need.

According to functional decomposition described in [40], a complex CAD model can be broken down into a set of surfaces with continuity constraints. Typically there is a hierarchy comprising (i) primary surfaces (ii) connecting surfaces, such as fillets, and (iii) corner patches. In the DSR context, surface reconstruction is also performed accordingly, providing continuity constraints from the previous phases. Primary surfaces are independent; fillets smoothly connect to two primary surfaces, and corner patches can be joined to several

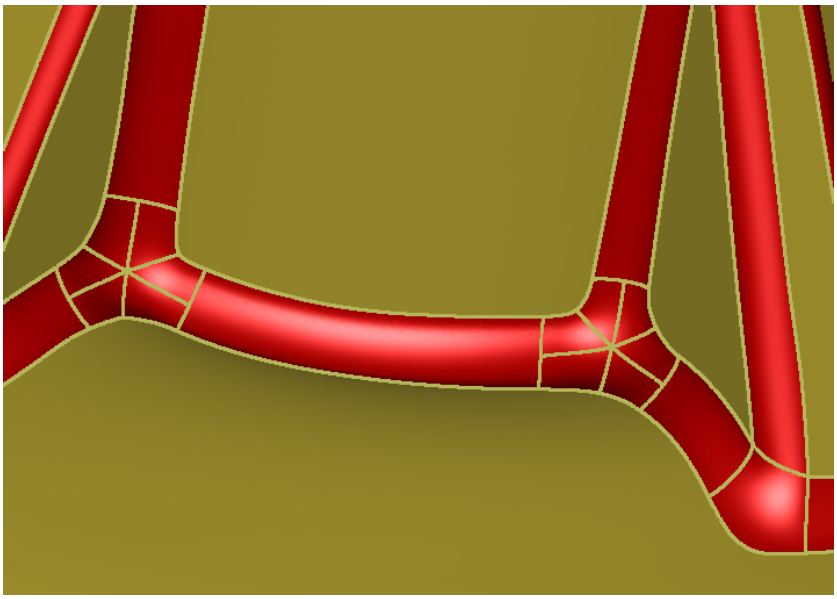


Figure 3.1: Primaries, connections, corner patches.

surfaces up in the hierarchy — including not only fillets, but primaries as well, sharing common boundaries with corner patches at T-node junctions or when setbacks are applied [41], see Fig. 3.1.

We would like to perform fairing in a hierarchical order, first fairing the primary surfaces, then fillets, and finally corner patches. In order to achieve this, we need algorithms that create fair surfaces while retaining smooth connections to the adjacent surface elements. For fillets, smooth connection to two primary surfaces must be satisfied; for corner patches, smooth connection to n surrounding surface elements is needed. Our goal is to deal primarily with the latter problem of constrained fairing, which includes a solution for fillets, as well [32, 33].

A general observation is that while primary surfaces are relatively large and are supposed to preserve the original design intent, fillets and corner patches, being much smaller (and the related measured data points less accurate), are more lenient about deviation in favor of ensuring continuity and fairness.

While there are several papers in the literature that deal with continuity constraints or surface fairing, their connection has not been investigated

thoroughly. Generally, when an individual surface is faired, the original accuracy along the surface boundaries is lost; when continuity constraints are satisfied, often undesirable curvature artifacts can be observed in the interior of the surface. Our goal is to accomplish both of these seemingly conflicting tasks: maintain continuity and provide fair surfaces simultaneously in the post-processing context, which makes it possible to perfect dependent surface geometries of a complex CAD model. The connections between the adjacent surface elements have a large influence on the overall quality of the model, and at least numerical (tolerance-driven) G^2 continuity is recommended.

3.1 Overview

We can break this problem into two separate subproblems. The first one is to ensure continuous connections, i.e., to tweak a surface S to match fixed *master surfaces* M_i by numerical G^1 and G^2 constraints. The second one is to perform the actual fairing, while retaining the continuity constraints to the master surfaces.

Concerning curvature continuity constraints, Pegna and Wolter [26] proved the *Linkage Curve Theorem,* which gives a necessary and sufficient condition for G^2 continuity between two surfaces that share common tangent planes. The theorem states the following (as rephrased in Hermann et al. [14], who also extended the theorem to G^n continuity):

> Two surfaces tangent along a C^1-smooth linkage curve[1] are curvature continuous if and only if at every point of the linkage curve, their normal curvature agrees for an arbitrary direction other than the tangent to the linkage curve.

This is the basis of the G^2 algorithm in Section 3.2.2. As for the fairing of the surface interior, slightly modified variations of the algorithms presented in the previous chapter will be applied (see Section 3.3).

A combination of the above two types of algorithms yields a solution to our main problem — a workflow that reaches the above goal step-by-step.

[1] i.e., the common boundary curve

Assume that the surface we want to modify is connected to n neighboring surfaces up in the hierarchy (*master surfaces*) with C^0 continuity. The process (described in detail in Section 3.4) uses the two algorithms in an alternating manner: the first sets geometric continuity to the master surfaces (enhancing C^0 to G^1 continuity in the first phase and G^1 to G^2 continuity in the second), and the other performs fairing retaining the continuity already achieved. The alternation with fairing is essential, because the continuity constraints may crease the surface near the sides, thus harming surface quality.

For the extension to n-sided corner patches (see Section 3.5), bi-parametric surface algorithms cannot be used. Instead, a mesh-based fairing algorithm is applied, described by Kobbelt [18], where the n-sided surface region is approximated by a mesh and discrete fairing is applied. For a completely different approach on n-sided patches using only one continuous surface, the reader is referred to Chapter 4.

3.2 Continuity Constraints

A B-spline surface $S(u, v)$ is defined by means of its control points P_{ij} ($i \in [0 \dots n]$, $j \in [0 \dots m]$) and its knot vectors U and V:

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} N_i^{U,p}(u) N_j^{V,q}(v),$$

where p and q are the degrees of the surface in the u and v directions, respectively.

The four boundaries of the surface are defined by the outermost control points — P_{0j}, P_{nj}, P_{i0} and P_{im}, respectively ($i \in [0 \dots n]$, $j \in [0 \dots m]$). In this context we call these together as the *positional frame*.

Let us assume hereinafter that the outermost control points are fixed. The tangent planes at the points of the boundaries are indirectly determined by the first cross-boundary derivatives, i.e., by the inner control points P_{1j}, $P_{(n-1)j}$, P_{i1} and $P_{i(m-1)}$, respectively ($i \in [1 \dots n-1]$, $j \in [1 \dots m-1]$). We

call these control points together the *tangential frame*.

Finally, assume that control points of the positional and tangential frames are fixed. Then the surface curvatures at the points of the boundaries are indirectly determined by the second cross-boundary derivatives, i.e., by the inner control points P_{2j}, $P_{(n-2)j}$, P_{i2} and $P_{i(m-2)}$, respectively ($i \in [2 \dots n-2]$, $j \in [2 \dots m-2]$). We call these control points together the *curvature frame*. These frames are depicted in Figure 3.2.

It is also well-known, that when we enforce G^1 continuity independently for the individual boundaries, a so-called twist compatibility condition must be satisfied for the mixed partial derivatives, which are indirectly determined by the twist control points P_{11}, $P_{1(m-1)}$, $P_{(n-1)1}$ and $P_{(n-1)(m-1)}$. After we enforce G^2 continuity for the individual boundaries, a similar compatibility condition must be satisfied to tweak the inner twist control points P_{22}, $P_{2(m-2)}$, $P_{(n-2)2}$ and $P_{(n-2)(m-2)}$.

The following methods set continuity between two B-spline surfaces M (master) and S (slave) by modifying the appropriate frame of S. The general idea is to draw up the equations for a number of sampled parameter points along the border and minimize the least-squares error of the equation system. For the sake of simplicity, the algorithms will be explained on a per side basis. The twists, however, need special attention, and they will be dealt with at the end of this section.

3.2.1 G^1 Continuity

Let M and S be joined (without loss of generality) along the $u = u_{\min}$ parameter line with C^0 continuity. The goal is to modify the corresponding side of the tangential frame, i.e., the second control row of S, such that the two surfaces will have numerical G^1 continuity.

Taking some sampled parameters v_k ($k \in [1 \dots K]$) along the common boundary, let the normal vectors at these (u, v_k) points of the master surface be denoted by $\underline{n}_k$. If displacement vectors $\underline{w}_j$ are added to the control points

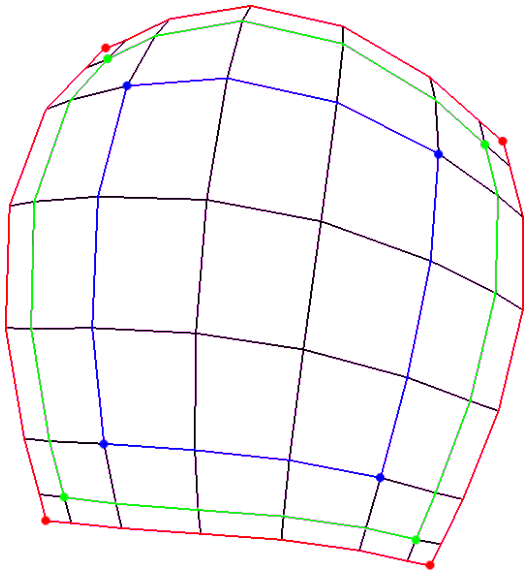


Figure 3.2: Frames of a biparametric surface: positional (red), tangential (green) and curvature (blue)

P_{1j} $(j \in [1 \ldots m-1])$, they will modify the original tangents $\underline{e}_k$ as follows:

$$\hat{\underline{e}}_k = \underline{e}_k + N_1^{U,p,(1)}(u) \sum_{j=1}^{m-1} N_j^{V,q}(v_k)\underline{w}_j \tag{3.1}$$

$$= \underline{e}_k + \sum_{j=1}^{m-1} c_{k,j}\underline{w}_j. \tag{3.2}$$

Since it is better to avoid irrelevant control point movements, deviation should be minimized only in the surface normal direction. This leads to

$$\hat{\underline{e}}_k = \underline{e}_k - \underline{n}_k(\underline{e}_k\underline{n}_k). \tag{3.3}$$

Subtracting $\underline{e}_k$ from (3.2) and (3.3) gives the linear equation system $A\underline{x} = \underline{b}$, where

$$A = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1(m-1)} \\ c_{21} & c_{22} & \cdots & c_{2(m-1)} \\ \vdots & \vdots & \ddots & \vdots \\ c_{K1} & c_{K2} & \cdots & c_{K(m-1)} \end{bmatrix},$$

$$\underline{x} = \begin{bmatrix} \underline{w}_1 \\ \underline{w}_2 \\ \vdots \\ \underline{w}_{m-1} \end{bmatrix}, \; \underline{b} = \begin{bmatrix} -\underline{n}_1(\underline{e}_1\underline{n}_1) \\ -\underline{n}_2(\underline{e}_2\underline{n}_2) \\ \vdots \\ -\underline{n}_K(\underline{e}_K\underline{n}_K) \end{bmatrix}.$$

There are K equations and $m-1$ unknowns. Solving the least-square equation $(A\underline{x} - \underline{b})^2 = 0$ leads to a linear system of $(A^T A)\underline{x} = A^T\underline{b}$.

3.2.2 G^2 Continuity

Consider the same assumptions as in the G^1 case, but now also with numerical G^1 connection. Take K parameter points from the v domain: v_k, $1 \leq k \leq K$. For every v_k, calculate the normal curvature κ_k^M of M at $(u_{\min}, v_k)$ in the u direction. If S is modified in a way that its normal curvature in the u direction is the same as κ_k^M, the Linkage Curve Theorem ensures G^2 continuity at $(u_{\min}, v_k)$.

The normal curvature of a surface at (u, v) in some $\underline{d}$ direction can be calculated as:

$$\kappa(\lambda) = \frac{L + 2M\lambda + N\lambda^2}{E + 2F\lambda + G\lambda^2},$$

where E, F, G and L, M, N are the coefficients of the first and second fundamental forms, respectively; $\lambda = \frac{d_v}{d_u}$ and $\underline{d} = d_u\underline{S}_u + d_v\underline{S}_v$. In the special cases where $\underline{d}$ is the u or v parametric direction, this is simplified into L/E and N/G, respectively [6].

3.2.2.1 Minimization

The curvature of the surface curve $C_k(u) = S(u, v_k)$ can be used instead of directly optimizing for surface curvature. This results in much simpler equations, as it will be explained below. Meusnier's theorem shows that at a given point, the curvature κ_C of a surface curve C and the normal curvature κ of a surface S in the tangent direction of C have the following relationship [39]:

$$\kappa = \kappa_C \cos\theta = \frac{\|\underline{C}' \times \underline{C}''\|}{\|\underline{C}'\|^3} \cos\theta = \frac{\langle \underline{C}'', \underline{n} \rangle}{\|\underline{C}'\|^2},$$

where $\underline{n} = \frac{\underline{S}_u \times \underline{S}_v}{\|\underline{S}_u \times \underline{S}_v\|}$ is the surface normal and θ is the angle between $\underline{n}$ and the curve normal $(\underline{C}' \times \underline{C}'') \times \underline{C}'$.

Consequently, the equations to be solved have the form

$$\frac{\langle C_k''(u_{\min}), \underline{n}_k \rangle}{\|C_k'(u_{\min})\|^2} = \kappa_k^M. \tag{3.4}$$

Since

$$\begin{aligned} C_k(u) &= S(u, v_k) = \sum_{i=0}^{n} N_i^{U,p}(u) \left(\sum_{j=0}^{m} N_j^{V,q}(v_k) P_{ij} \right) \\ &= \sum_{i=0}^{n} N_i^{U,p}(u) \hat{P}_i, \end{aligned}$$

the first and second derivatives at the end are

$$\begin{aligned} C_k'(u_{\min}) &= \frac{p}{u_{p+1} - u_1}(\hat{P}_1 - \hat{P}_0), \\ C_k''(u_{\min}) &= \frac{p-1}{u_{p+1} - u_2} \left(\frac{p}{u_{p+2} - u_2}(\hat{P}_2 - \hat{P}_1) - \frac{p}{u_{p+1} - u_1}(\hat{P}_1 - \hat{P}_0) \right), \end{aligned}$$

see [27] (assuming that $u_0 = u_1 = \cdots = u_p = u_{\min}$).

If we want to modify the P_{2j} control point by a vector $\underline{d}_j$, we can reformulate Eq. 3.4 as

$$\Delta\kappa_k \|C_k'(u_{\min})\|^2 \frac{(u_{p+1} - u_2)(u_{p+2} - u_2)}{p(p-1)} = \sum_{j=2}^{m-2} N_j^{V,q}(v_k) \left\langle \underline{d}_j, \underline{n}_k \right\rangle, \tag{3.5}$$

where $\Delta\kappa_k = \kappa_k^M - \kappa_k^S$. Note that the second half of C_k'' is eliminated in the scalar product by the (perpendicular) surface normal.

The following two sections propose different solutions for this equation system. In the first one, in order to avoid irrevelant control point movements, the P_{2j} points are only allowed to move in an *outward* direction ($\underline{w}_j$ for P_{2j}). In the second one, the requirement is that the sum of the squared deviations should be minimal.

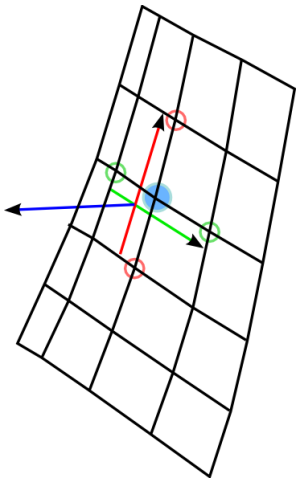


Figure 3.3: Outward direction

3.2.2.2 Solution by Fixed Directions

An easy and intuitive choice for outward directions is to get the cross product of the difference of the neighboring control points (Fig. 3.3), i.e., $\underline{w}_j = (P_{3j} - P_{1j}) \times (P_{2(j+1)} - P_{2(j-1)})$. Defining the deviation vectors $\underline{d}_j$ as $\xi_j \underline{w}_j$ and introducing the constants α_{kj} and β_k, Eq. 3.5 can be rewritten as $\beta_k = \sum_{j=2}^{m-2} \alpha_{kj} \xi_j$.

This leads to the overdefined equation system $A\underline{x} = \underline{b}$:

$$A = \begin{bmatrix} \alpha_{12} & \alpha_{13} & \cdots & \alpha_{1(m-2)} \\ \alpha_{22} & \alpha_{23} & \cdots & \alpha_{2(m-2)} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{K2} & \alpha_{K3} & \ldots & \alpha_{K(m-2)} \end{bmatrix},$$

$$\underline{x} = \begin{bmatrix} \xi_2 \\ \xi_3 \\ \vdots \\ \xi_{m-2} \end{bmatrix}, \quad \underline{b} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_K \end{bmatrix}.$$

Solving the $(A^T A)\underline{x} = A^T \underline{b}$ equation results in a least-squares approximation, as earlier.

3.2.2.3 Solution by Minimal Deviation

The equations can also be solved while minimizing the squared deviation of the $\hat{P}_2$ control point of C_k from its original position, by requiring that it should change only in the $\underline{n}_k$ direction. This means that Eq. 3.5 becomes

$$\beta_k \underline{n}_k = \sum_{j=2}^{m-2} N_j^{V,q}(v_k)\underline{d}_j = \sum_{j=2}^{m-2} \gamma_{kj}\underline{d}_j.$$

The equation system is now $A\underline{x} = \underline{b}$, where

$$A = \begin{bmatrix} \gamma_{12} & \gamma_{13} & \cdots & \gamma_{1(m-2)} \\ \gamma_{22} & \gamma_{23} & \cdots & \gamma_{2(m-2)} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{K2} & \gamma_{K3} & \cdots & \gamma_{K(m-2)} \end{bmatrix},$$

$$\underline{x} = \begin{bmatrix} \underline{d}_2 \\ \underline{d}_3 \\ \vdots \\ \underline{d}_{m-2} \end{bmatrix}, \; \underline{b} = \begin{bmatrix} \beta_1\underline{n}_1 \\ \beta_2\underline{n}_2 \\ \vdots \\ \beta_K\underline{n}_K \end{bmatrix}.$$

As before, $(A^TA)\underline{x} = A^T\underline{b}$ gives a least-squares approximation.

3.2.3 Twists

Every twist (or inner twist) control point of a frame has two values coming from the independent side constraints. In order to get a valid B-spline, a single twist control point is needed. There is a range of methods how to determine a common value, though experience shows, that the two candidate control points coming from the adjacent boundaries generally lie very close to each other, so a simple averaging works well. Having tweaked the twist control points, we can repeat the continuity setting algorithms, now constraining only the inner $j \in [2 \ldots m-2]$ control points for G^1 and the $j \in [3 \ldots m-3]$ control points for G^2 continuity, respectively, which will yield better frames that comprise the new twist values.

Another option is to solve the constraint equations for all sides as one large system, in which the incompatible twist conditions appear with equal weights.

3.3 Fairing Algorithms

There is one extra requirement for the surface fairing algorithms used here: they should be able to preserve the first k frames ($k = 1 \ldots 3$). For a control point-based method like the KRR algorithm, this is trivially achieved by limiting the set of modifiable control points. The curvature approximation based method presented in Section 2.3.3 also uses a linear equation system on the control points, so it suffices to pin down some points (and thus reduce the number of variables).

The isocurve and PDE extensions (Sections 2.3.1 and 2.3.2), on the other hand, use surface fitting as a final step, so the main modifications must be embedded into the surface fitting algorithm. There are several ways to fit a surface with partially fixed geometry, see e.g. [44]. Additionally, some adjustments in the mesh generation process may also be necessary for better results. In the isocurve-based extension, constraining k end control points of the isocurves helps in creating a consistent mesh. For the same reason, the first few triangle frames should also be fixed in the PDE extension.

With these modified algorithms, and the continuity setting methods proposed in the previous section, we are now ready to sketch out the constrained fairing process.

3.4 Fairing Four-sided Patches

The algorithm for fairing and setting continuity constraints proceeds in the following way:

1. Insert some knots into the surface, if it has too few control points.

2. Fair the surface, while retaining C^0 continuity for each boundary, i.e., only control points within the positional frame are used for fairing.

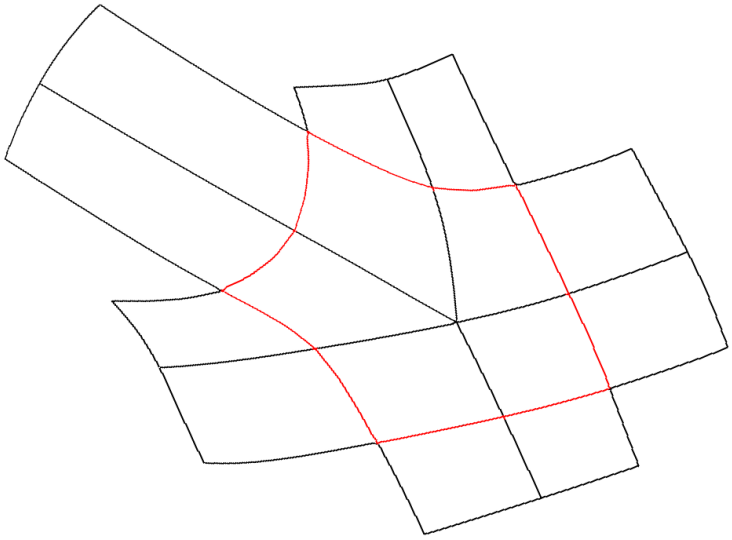


Figure 3.4: Five-sided corner patch consisting of five quadrilateral surfaces

3. Fix the positional frame and enforce G^1 continuity for each boundary.

4. Fair the surface, while retaining G^1 continuity, i.e., only control points within the tangential frame are used for fairing.

5. Fix the tangential frame and enforce G^2 continuity for each boundary.

6. Fair the surface, while retaining G^2 continuity, i.e., only control points within the curvature frame are used for fairing.

To sum it up, the sequence is always alternating, i.e.: (i) constrain, (ii) apply fairing, until a faired G^2 surface is achieved.

3.5 Extension to n-sided Patches

There are corner patches with three or more than four (usually five or six) connecting surfaces. The simplest representation of these is based on the so-called central split, where n quadrilateral surfaces are stitched together, see Fig. 3.4 for a five-sided example.

The main difficulty here is that corner patches consist of more than one biparametric surface, so continuity has to be ensured both along the external

boundaries and the internal subdividing curves between the quadrilaterals. Moreover, for multi-surface n-sided regions the previously mentioned fairing methods (and incidentally any other fairing method known to the author) cannot be used.

3.5.1 Extension of the Algorithm

In order to cope with these difficulties, the procedure presented in Section 3.4 is modified as follows:

1. Global fairing of the quadrilaterals (retaining C^0 or G^1 continuity on the perimeter).
2. Enforce G^1 continuity on the perimeter and between the quadrilaterals.
3. Local fairing of each quadrilateral, retaining G^1 continuity.
4. Enforce G^2 continuity on the perimeter and between the quadrilaterals.
5. Local fairing of each quadrilateral, retaining G^2 continuity.

The first step aims at creating a good base for further operations. It can be omitted if the original surface has adequate quality. Unlike in the four-sided case, the first step here needs special attention, since multiple surfaces need to be processed simultaneously. A mesh-based fairing seems to be a natural approach.

3.5.2 Mesh-based Fairing

In order to avoid the parameterization problems, the corner patch should first be discretized, resulting in a triangle mesh as shown in Fig. 3.5. The actual fairing is done by a method suggested by Kobbelt [18], inheriting G^1 continuity at the boundaries. Finally, the surfaces can be fitted back using the points of the faired mesh. Although this algorithm, due to precision loss, may be inferior to the others, it creates a good shape generally and is easy to use regardless of the number of quadrilaterals involved. Fig. 3.6 shows how the reflection lines change on the mesh after fairing.

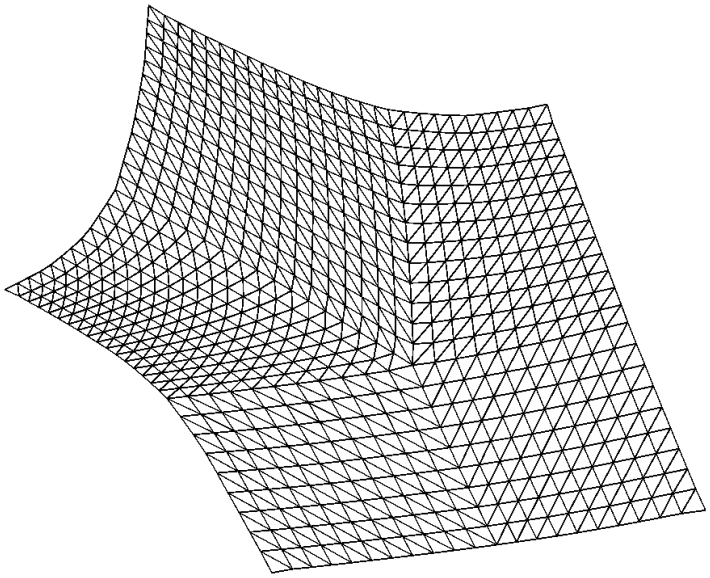


Figure 3.5: Triangle mesh of a five-sided corner patch

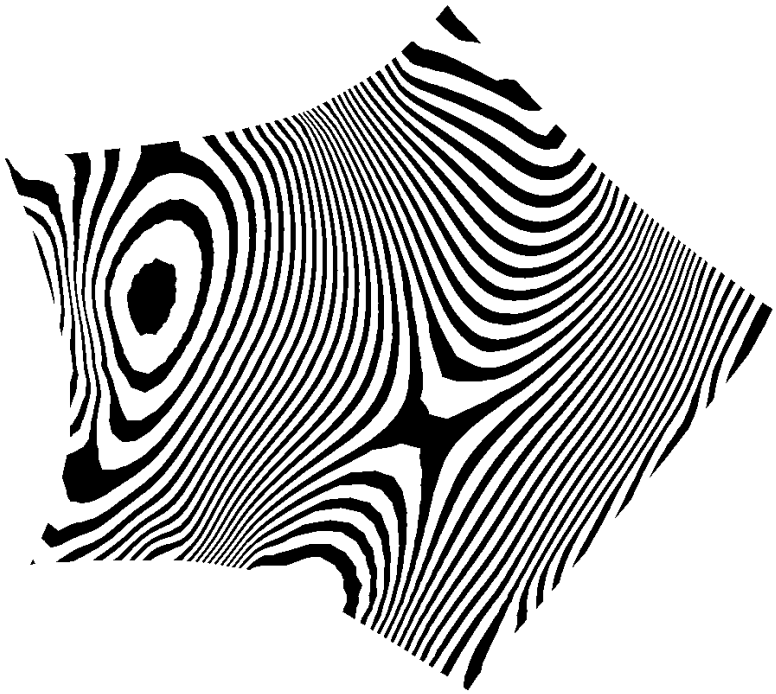
A smooth mesh can also be generated from sampled points of a transfinite interpolation surface (see the next chapter), when all boundary constraints are well-defined.

3.6 Examples

Figures 3.7 and 3.8 both show the effect of constrained fairing for four-sided corner patches. In these cases, the central surface is relatively simple; the original (a) has reasonable isophotes, but the isophote stripes break when they reach the boundaries (only G^1 continuity). After constrained fairing (b) the images show that numerical G^2 continuity has been achieved and fairing also nicely affected the interior without changing the master surfaces.

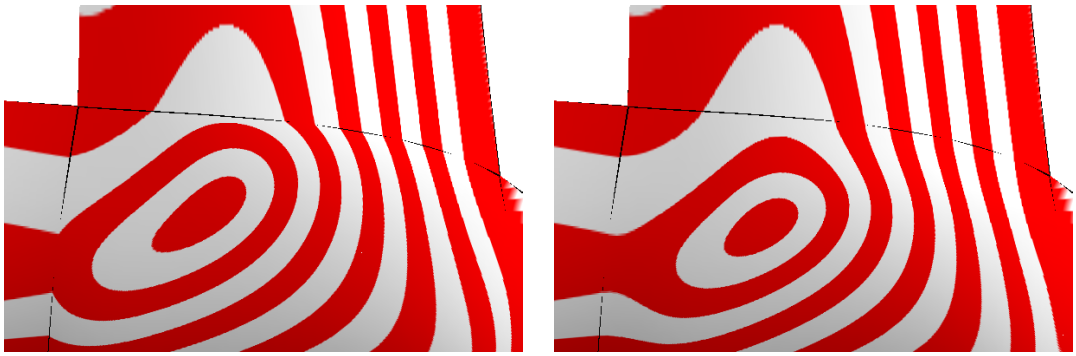
Figure 3.9 also shows a similar example, but here the corner patch joins the fillets with only C^0 continuity (the isophote lines do not match at the boundaries). In the first step, continuity is enhanced to G^1 (image on the right) which was further enhanced to G^2 in the second step (bottom image). The smoothness of the surface is due to the fairing at the end of each step, as described above.

(a) Before fairing



(b) After fairing

Figure 3.6: Reflection lines on the corner mesh



(a) Before fairing (b) After fairing

Figure 3.7: Fairing an X-node (junction of four fillets)

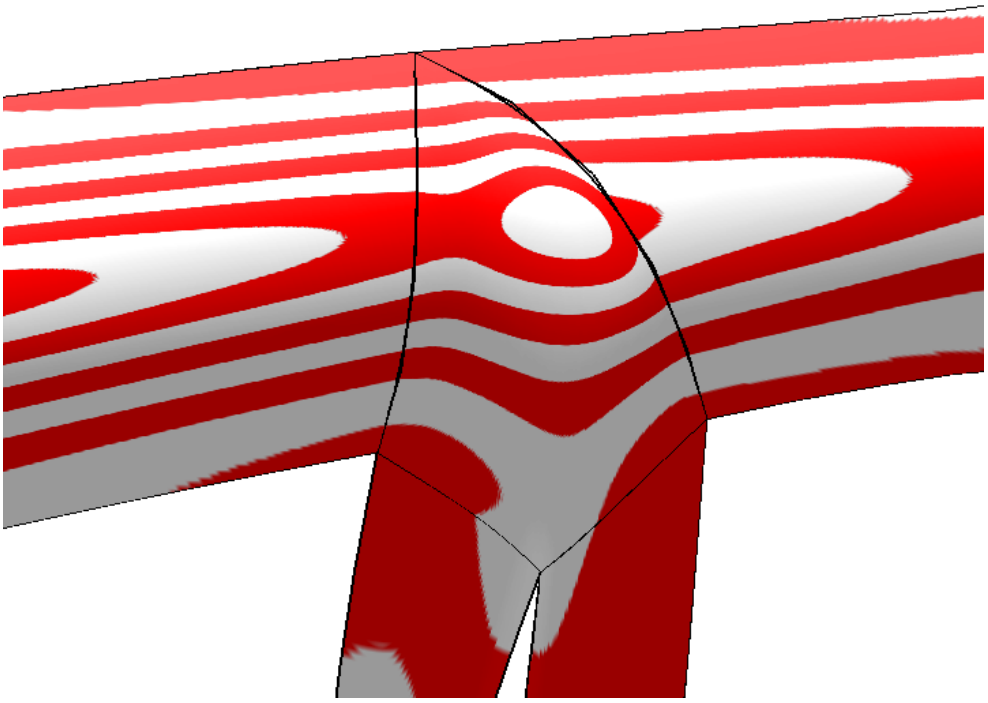
We have a different case in Figure 3.10. Here the original corner patch already had numerical G^2 that needed only very minor changes. On the other hand, fairing increased the overall surface quality very much, leaving the connectivity intact.

Finally Figure 3.11 shows the stages of hierarchical fairing: first the connecting surfaces were faired (independently), followed by the corner patch, where continuity constraints were also taken into account.

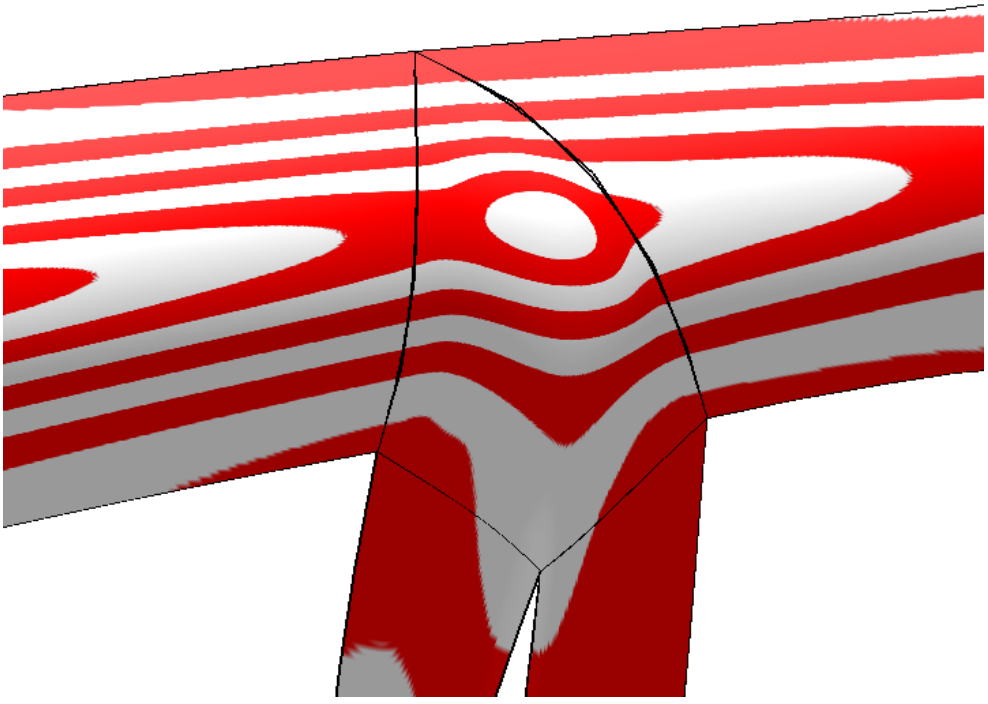
Conclusions

A new approach was presented for fairing surface elements of complex CAD models in order to support digital shape reconstruction based on measured data. The process combines fairing and continuity setting algorithms to perfect functionally decomposed surfaces following a hierarchical order.

Nonetheless, there is still room for future research and enhancements. For example, in the n-sided corner patch case, the discrete fairing step may be replaced by some continuous fairing technique. Another idea is to fair functionally decomposed surfaces locally, i.e., using selected portions of adjacent primaries, connections and corner patches and then apply hierarchical fairing which will affect only the selected areas.

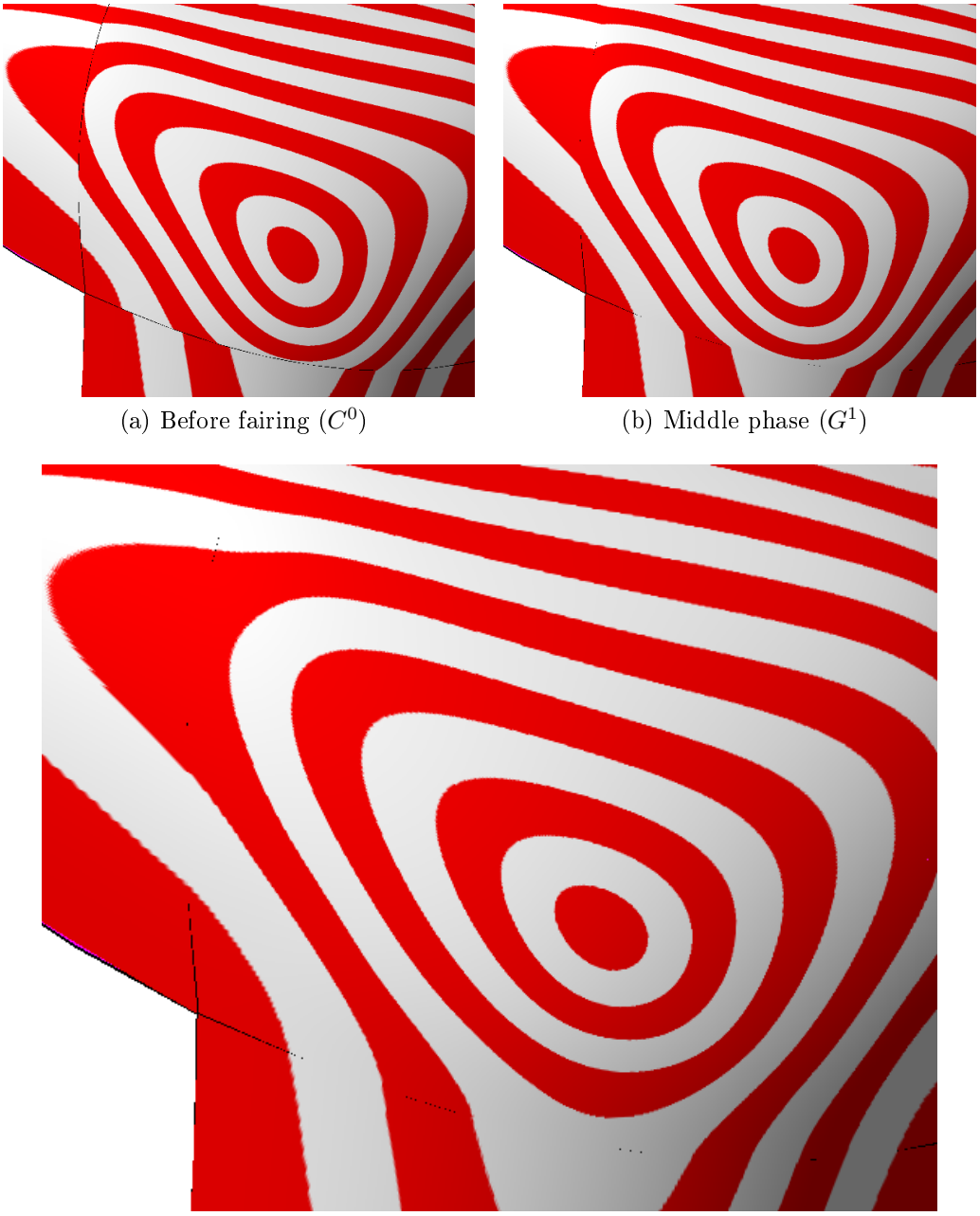


(a) Before fairing



(b) After fairing

Figure 3.8: Fairing another X-node

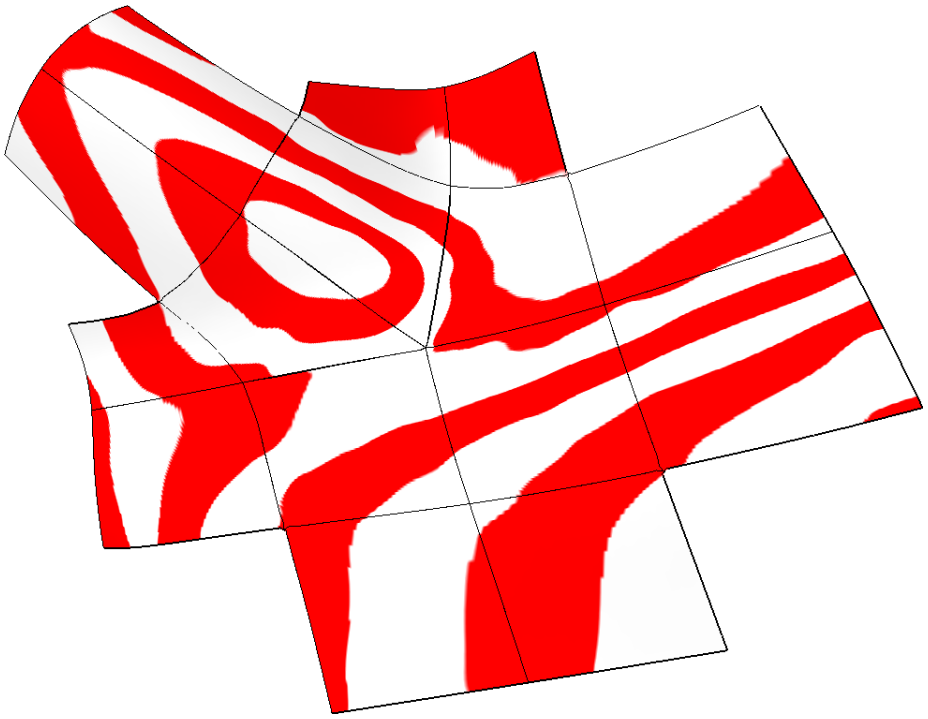


(a) Before fairing (C^0)

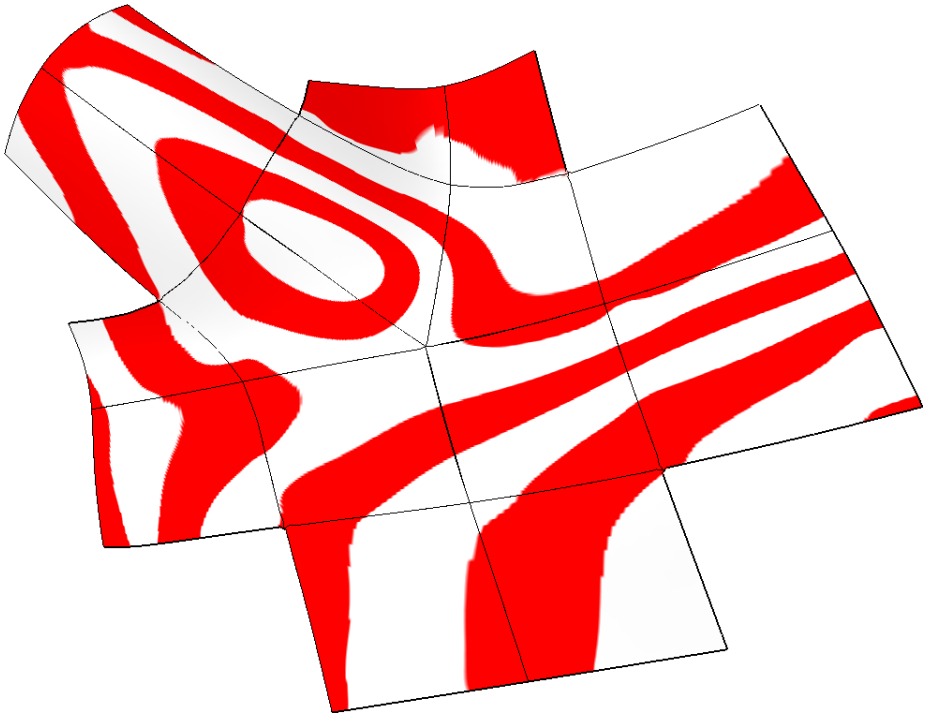
(b) Middle phase (G^1)

(c) Final result (G^2, faired)

Figure 3.9: Fairing a third X-node



(a) Before fairing



(b) After fairing

Figure 3.10: Fairing a five-sided corner patch.

(a) Before fairing

(b) Fairing the connecting surfaces

(c) Fairing the corner patch

Figure 3.11: Two levels of fairing

Chapter 4

Transfinite n-sided Surfaces

One of the major problems in Computer-Aided Geometric Design (CAGD) is to create mathematical representations for complex free-form objects composed of several smoothly connected surfaces. An intuitive solution is the curvenet-based design paradigm, where the designer defines the feature curves of the object, then these are semi-automatically interpolated by surfaces. The feature curves may come from 2D manual sketches (see e.g. [19]) or directly drawn in 3D (as in [1]).

It is difficult to apply traditional parametric surfacing techniques in this situation, due to the lack of interior data points. General topology surfaces using recursive subdivision methods [2] are also inconvenient, as interpolating positional and tangential constraints at the boundaries is hardly possible. Coons patches [4, 6], on the other hand, provide a viable solution for four-sided curve networks, since they are defined by boundary curves and their cross-derivatives.

Unfortunately, while the majority of surface patches in a model is four-sided, almost all industrial objects contain general n-sided patches, as well. Most frequently three-, five- or six-sided patches are needed; however, one- or two-sided patches may also occur in practical design. Conventional methods for treating these surfaces include *trimming*, where a larger quadrilateral patch is created and then it is intersected with given boundary curves, and *splitting* [15], where multiple four-sided surfaces are stitched together to rep-

resent one n-sided patch, as explained in Section 3.5.

It is easy to see that splitting is sub-optimal: any modification of the boundary conditions forces the (non-trivial) recomputation of the subdivision curves, the continuity constraints and fairness in the interior (for example via the algorithm introduced in Section 3.5), so design changes can hardly be expected to be executed in real-time. The problem with trimming is much more subtle: finding an adequate quadrilateral requires a good parameterization, and depending on the nature of the boundary curves, the resulting surface may have sudden curvature changes. These can only be prevented by variational fairing methods, which are computationally demanding. Also, both of the above approaches lack flexibility.

The real solution to this problem is the multi-sided generalization of the Coons patch, called *transfinite surface interpolation*. This contains several separate subproblems, such as (i) constructing interpolants, (ii) finding suitable domain polygons, (iii) defining parameterizations that map the domains of the interpolants onto the n-sided polygonal domain, and (iv) designing blending functions to combine the interpolant surfaces. Transfinite surface interpolation is a well-researched area of CAGD, originating from the late 1960s, and thus each of these tasks have numerous variations. For a complete overview of the subject, the reader is referred to [37, 22], and also the recent review in [42]. We will examine some of the major variants in Section 4.1. Note that one- and two-sided surfaces need special attention, since most of these methods need at least three sides. One possible construction for these has been described recently in [43], along with techniques for more precise interior control.

In the subsequent sections new techniques are presented to construct non-regular domains (Sec. 4.2) and special line-sweep parameterizations (Sec. 4.3). Then in Section 4.4 a new transfinite surface representation is introduced that can be regarded as the natural multi-sided generalization of the Coons patch. In Section 4.5 we propose the use of curved side interpolants, having definite benefits over the usual linear ribbons, and in Section 4.6 another new interpolation scheme is defined that is based on these curved interpolants. A few test results conclude the chapter in Section 4.7.

4.1 Previous Work

In this section we will review the construction of Coons patches and will rewrite it in a form that is easier to generalize, using linear ribbons. We will also mention two multi-sided surface formulations that were influential to the work presented in this chapter.

4.1.1 Reformulating the Coons Patch

Let us assume that we have a four-sided surface patch S parameterized in the (u, v) plane ($u, v \in [0, 1]$). Given positional side constraints $S(u, 0)$, $S(u, 1)$, $S(0, v)$, $S(1, v)$ and tangential (cross-derivative) constraints $S_v(u, 0)$, $S_v(u, 1)$, $S_u(0, v)$, $S_u(1, v)$, the original Coons patch can be written as

$$\begin{aligned}
U &= \begin{bmatrix} \alpha_0(u) & \beta_0(u) & \alpha_1(u) & \beta_1(u) \end{bmatrix}, \\
V &= \begin{bmatrix} \alpha_0(v) & \beta_0(v) & \alpha_1(v) & \beta_1(v) \end{bmatrix}, \\
S^u &= \begin{bmatrix} S(u,0) & S_v(u,0) & S(u,1) & S_v(u,1) \end{bmatrix}, \\
S^v &= \begin{bmatrix} S(0,v) & S_u(0,v) & S(1,v) & S_u(1,v) \end{bmatrix}, \\
S^{uv} &= \begin{bmatrix} S(0,0) & S_u(0,0) & S(1,0) & S_u(1,0) \\ S_v(0,0) & S_{uv}(0,0) & S_v(1,0) & S_{uv}(1,0) \\ S(0,1) & S_u(0,1) & S(1,1) & S_u(1,1) \\ S_v(0,1) & S_{uv}(0,1) & S_v(1,1) & S_{uv}(1,1) \end{bmatrix}, \\
S(u,v) &= V(S^u)^T + S^v U^T - V S^{uv} U^T.
\end{aligned}$$

This is the well-known Boolean sum formulation [6], where α_0, α_1, β_0, β_1 are the Hermite blending functions

$$\begin{aligned}
\alpha_0(t) &= 2t^3 - 3t^2 + 1, & \alpha_1(t) &= -2t^3 + 3t^2, \\
\beta_0(t) &= t^3 - 2t^2 + t, & \beta_1(t) &= t^3 - t^2.
\end{aligned}$$

In order to reformulate the above definition on a per-side basis, we will use cyclic indices (with 1 coming after 4), and introduce so-called *side parameters,*

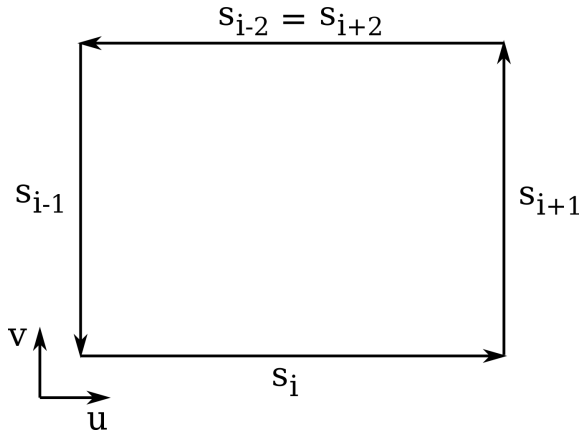


Figure 4.1: Domain of a Coons patch using side-based parameterization

$s_i = s_i(u, v)$, associated with the i-th side of the domain, taking the values u, v, $1-u$ and $1-v$, as appropriate (see Fig. 4.1). Then, denoting the positional and tangential constraints by $P_i(s_i)$ and $T_i(s_i)$, respectively, as well as using $W_{i,i-1}$ for the twist vector, the Coons patch can be rewritten as

$$\begin{aligned} S(u,v) &= \sum_{i=1}^{4} \begin{bmatrix} \alpha_0(s_{i+1}) & \beta_0(s_{i+1}) \end{bmatrix} \begin{bmatrix} P_i(s_i) \\ T_i(s_i) \end{bmatrix} \\ &- \sum_{i=1}^{4} \begin{bmatrix} \alpha_0(s_{i+1}) & \beta_0(s_{i+1}) \end{bmatrix} \begin{bmatrix} P_i(0) & T_{i-1}(1) \\ T_i(0) & W_{i,i-1} \end{bmatrix} \begin{bmatrix} \alpha_0(s_i) \\ \beta_0(s_i) \end{bmatrix}. \end{aligned}$$

We assume that twist vector compatibility is ensured, i.e.,

$$W_{i,i-1} := \frac{\partial}{\partial s_i} T_i(0) = -\frac{\partial}{\partial s_{i-1}} T_{i-1}(1).$$

For contradicting boundary constraints, this can be resolved by Gregory's rational twists (see [9, 6]).

This formulation can be further simplified, if we replace the tangential blending with linear interpolants or *ribbons*:

$$R_i(s_i, d_i) = P_i(s_i) + d_i T_i(s_i).$$

Here we introduced another local parameter, the so-called *distance parameter*

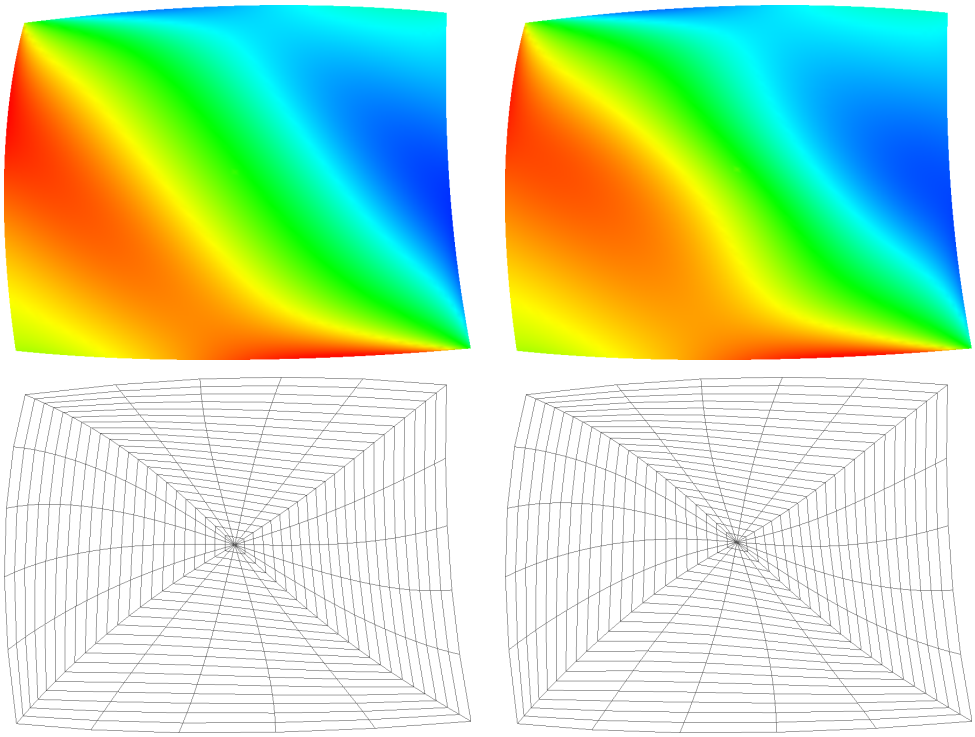


Figure 4.2: Coons patch using the original (left) and the ribbon-based formulation (right), showing fairly similar mean curvature maps and constant parameter lines.

$d_i = d_i(u, v)$, that measures some kind of distance from the i-th side. In this four-sided case $d_i(u, v) = s_{i+1}(u, v)$ is a simple choice. The resulting patch becomes

$$S(u, v) = \sum_{i=1}^{4} R_i(s_i, d_i)\alpha_0(d_i) - \sum_{i=1}^{4} Q_{i,i-1}(s_i, s_{i-1})\alpha_0(d_i)\alpha_0(s_i), \qquad (4.1)$$

where $Q_{i,i-1}$ is a corner correction patch given by

$$Q_{i,i-1}(s_i, s_{i-1}) = P_i(0) + (1 - s_{i-1})T_i(0) + s_i T_{i-1}(1) + s_i(1 - s_{i-1})W_{i,i-1}.$$

This will not be identical to the Coons patch, but it is a very similar construction (see Fig. 4.2) that trivially satisfies all the boundary constraints. This formulation will be the basis of the generalization presented in Section 4.4.

4.1.2 Traditional Multi-sided Surfaces

We can interpret the same problem for an arbitrary number of boundary curves. Given n three-dimensional curves $P_i(s_i)$, $1 \leq i \leq n$, and cross-derivatives $T_i(s_i)$ along these curves, we want to construct an n-sided patch that interpolates these boundary curves and derivatives. For this, we will need a domain polygon in the parameter plane. The generation of domain polygons is explained in detail in Section 4.2. We will also need a parameterization scheme that transforms the (u, v) coordinates of the parameter plane into local parameters of the individual boundary curves. These will be explored in Section 4.3.

The rest of this section concentrates on how these curves and tangents can be combined to create an interpolating patch, and what kind of blending functions can be used. Note that we continue to use cyclic indices in this chapter (e.g. $i-1$ for $i=1$ is n, and conversely, $i+1$ for $i=n$ is 1).

Charrot and Gregory [3] proposed an n-sided patch based on *corner interpolants*, i.e., surfaces that interpolate two consecutive boundary curves, defined as

$$\begin{aligned} R^{\text{corner}}_{i,i-1}(s_i, s_{i-1}) &= P_i(s_i) + (1-s_{i-1})T_i(s_i) + P_{i-1}(s_{i-1}) + s_i T_{i-1}(s_{i-1}) \\ &- P_i(0) - (1-s_{i-1})T_i(0) - s_i T_{i-1}(1) - (1-s_{i-1})s_i W_{i,i-1}. \end{aligned}$$

It is easy to see that these surfaces interpolate the i-th and $(i-1)$-th sides and their cross-derivatives. A similar technique will be introduced in Section 4.5 for interpolating three sides. The Gregory patch is constructed in the following way:

$$S^{\text{corner}}(u,v) = \sum_{i=1}^{n} R^{\text{corner}}_{i,i-1}(s_i(u,v), s_{i-1}(u,v)) \cdot B_{i,i-1}(d_1(u,v), \ldots, d_n(u,v)).$$

Here the blending function is

$$B_{i,i-1}(d_1, \ldots, d_n) = \frac{\prod_{k \notin \{i,i-1\}} d_k^2}{\sum_l \prod_{k \notin \{l,l-1\}} d_k^2} = \frac{D^2_{i,i-1}}{\sum_l D^2_{l,l-1}},$$

with the notation $D^e_{i_1 \dots i_k} = \prod_{j \notin \{i_1 \dots i_k\}} d^e_j$. This blending function will be used in Sec. 4.4.2 for the generalized Coons patch. Note that the d_i parameters in an n-sided polygon are generally not equal to s_{i+1} (as we have seen for rectangles in the previous section), but can be independent "distance" parameters. For details, see Sections 4.3 and 4.4.3.

Kato [17] also used the linear side interpolants R_i introduced above to define a patch as

$$S^{\text{side}}(u,v) = \sum_{i=1}^{n} R_i(s_i(u,v), d_i(u,v)) \cdot B_i^{\text{side}}(d_1(u,v), \dots, d_n(u,v)),$$

where the blending function is

$$B_i^{\text{side}}(d_1, \dots, d_n) = \frac{(1-d_i^2)/d_i^2}{\sum_j (1-d_j^2)/d_j^2}.$$

An alternative blending function for this patch, proposed by several authors (for example [42]), is

$$B_i^{\text{side}}(d_1, \dots, d_n) = \frac{1/d_i^2}{\sum_j 1/d_j^2} = \frac{D_i^2}{\sum_j D_j^2}.$$

Both variants have the drawback of being singular in the corners.

In the following two sections we will present important enhancements to these patches in terms of domain construction and parameterization.

4.2 Domain Constructions

One particularly neglected problem in transfinite surface interpolation is the construction of a good polygonal domain. It is a widely accepted principle in CAGD, that the parameterization of the domain and the mapped 3D objects should have a "similar" shape. In other words, the domain should be mapped into 3D with minimal distortion. For example, non-uniform B-splines are defined in this fashion, thus avoiding undesirable overshoots.

Most papers in the literature only make use of regular polygons, but

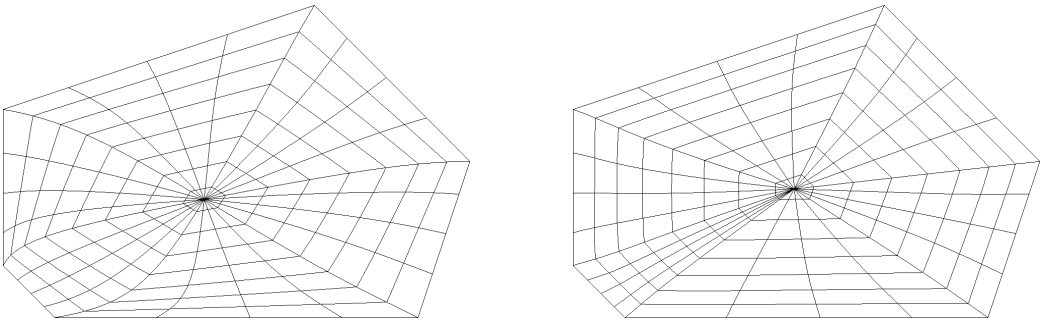


Figure 4.3: Parameterization with regular (left) and non-regular (right) domains.

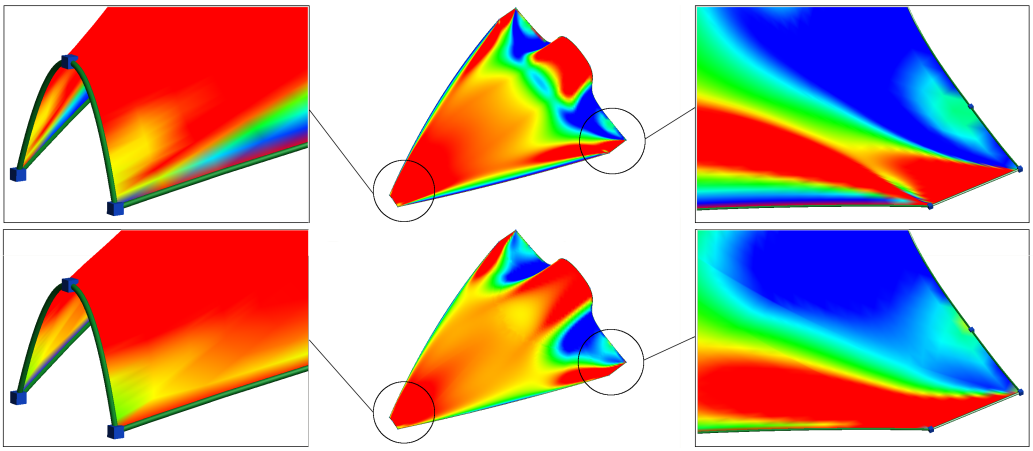


Figure 4.4: Mean maps of a surface with regular (top) and non-regular (bottom) domain polygon.

our experience shows that the use of non-regular domains instead of regular ones yields significant surface quality improvements for uneven boundary conditions. Compare the two parameterizations of a planar patch in Figure 4.3, shown by *spider-net curves* (constant parameter lines parallel to domain sides). Observe the unnatural bulge at the short side in the regular case. Also compare the mean maps of another surface, using regular and non-regular domains, in Fig. 4.4.

Let Ω denote a convex domain in the (u, v) plane, Γ its boundary, and $p_i = (u_i, v_i)$, $i = 1, \ldots, n$ the vertices to be determined. Index i runs in a counterclockwise order (Fig. 4.5). Denote the arc lengths of the given

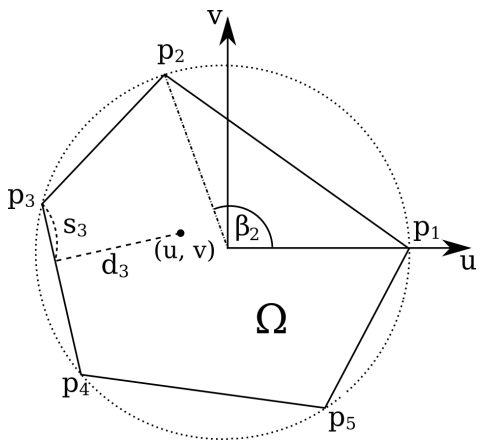


Figure 4.5: Circular polygonal domain I.

three-dimensional boundary curves by L_i, and the angles between the end tangents of the $(i-1)$-th and i-th boundaries by ϕ_i. Denoting the sides and the angles of the domain by l_i and α_i, respectively, we seek to minimize the squared deviation of the chord lengths and the angles, i.e., $\sum_i (l_i - c_{\text{length}} L_i)^2 + \sum_i (\alpha_i - c_{\text{angle}} \phi_i)^2$, where c_{length} and c_{angle} are properly chosen constants. This is a non-linear problem, but simple heuristic methods work well in practice. Three such algorithms are proposed below.

4.2.1 Arcs by Arc Lengths

The simplest method is to place the domain vertices on the perimeter of a unit circle proportionally to the arc lengths (inscribed polygon). Place the first vertex on the u axis, and the subsequent ones $(i = 2, \ldots, n-1)$ at angle

$$\beta_i = 2\pi \cdot \frac{\sum_{k=1}^{i-1} L_k}{\sum_{k=1}^{n} L_k}.$$

This is depicted in Figure 4.5.

4.2.2 Sides by Arc Lengths

For a better solution, instead of central angles, we propose to set the sides of the polygon proportional to the arc lengths of the boundaries. As Fig. 4.6

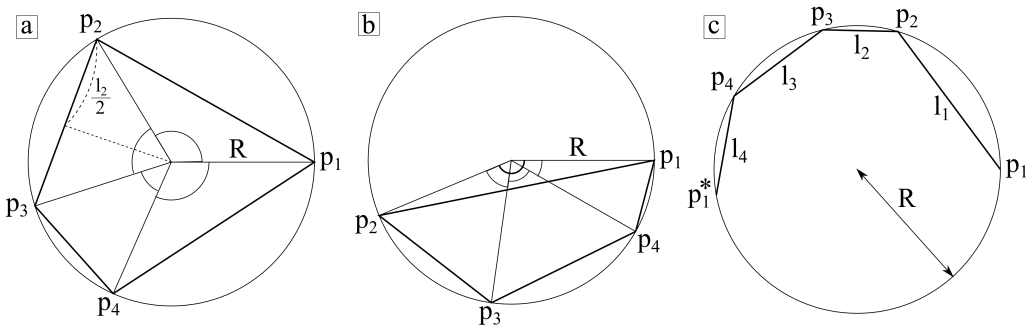


Figure 4.6: Circular polygonal domain II.

shows, two cases need to be distinguished: in case A, the center point of the circle is contained in the convex hull (Fig. 4.6a), while in case B, it lies outside (Fig. 4.6b).

Now, take a sufficiently large circle with radius R, and place the chord lengths L_i onto the circle one by one, placing an endpoint of the longest side (here denoted by L_1) onto the u axis (Fig. 4.6c). Then start decreasing the radius, and let the chord endpoints slide towards the other end, i.e., in the figure p_1^* moves towards p_1. In case A, this will be successful when the central half-angles of the chords satisfy $\sum \arccos \frac{L_i}{2R} = \pi$, assuming $L_1 < 2R$; otherwise, at one instant of the circle shrinking process, $2R$ becomes equal to L_1 without closing the loop. Then radius R needs to start growing again (case B), until we find an appropriate configuration, where $\sum_2^n \arccos \frac{L_i}{2R}$ is equal to $\arccos \frac{L_1}{2R}$. Note that, based on the given construction, for the existence of such a domain polygon it is sufficient to have $L_1 < \sum_2^n L_i$.

4.2.3 Sides and Angles

A third algorithm has also been elaborated that takes the local 3D angles into consideration. First, we normalize the angles to satisfy the necessary criterion for the n-sided convex polygon; i.e., let $c_{\text{angle}} = (n-2)\pi / \sum_i \phi_i$; then $\alpha_i = c_{\text{angle}} \phi_i$. Now, plake the chord lengths in sequence, retaining the angles, which will likely yield an open polyline, having a difference vector e between the first and last points (see Fig. 4.7). In order to amend this, we fix the very first point, and modify the subsequent ones sequentially, first by $\frac{1}{n}e$,

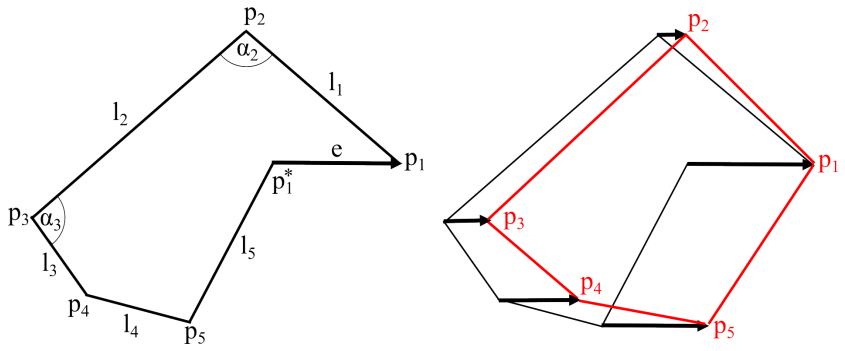


Figure 4.7: Length/angle-based convex polygonal domain.

then by $\frac{i}{n}e$ $(i = 2, \ldots, n)$. In the closed polygon obtained, both the chord lengths and the angles are somewhat distorted, but the results still proved to be satisfactory.

4.3 Parameterization Using Line Sweeps

There are various parameterization methods in the literature to map a (u, v) point from a convex polygon onto a rectangular domain. Recall that linear ribbons have two parameters, the side parameter s and the distance parameter d, where s gives the position on the defining boundary curve and d is the multiplier of the cross-tangent vector going in the direction of the surface interior. The d parameters are also used to determine the blending weights, as seen in Section 4.1.2.

As a side note, it is evident that these two functions of the d parameter are independent. This allows us to use a parameter pair (s, d) as ribbon parameters, and some other distance t for blending, essentially using a parameter triple (s, d, t) for every point. This has been found to be useful in practice, with perpendicular distance (see below) as the choice for t. Nevertheless, here we will continue to use only (s, d) parameters for simplicity's sake.

Side parameters are often derived from distance parameters by simple methods like

$$s_i = \frac{d_{i-1}}{d_{i-1} + d_{i+1}}.$$

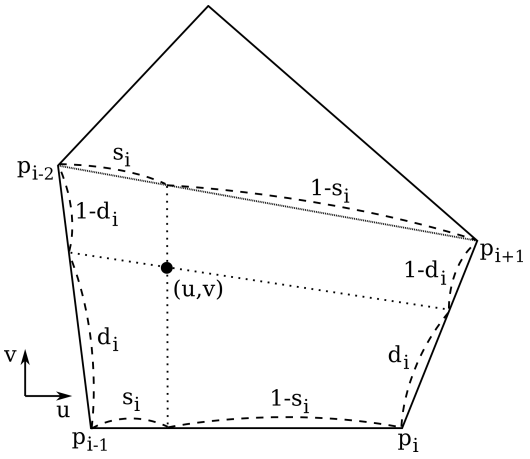


Figure 4.8: Proportions in bilinear parameterization.

We will see the reverse process in Section 4.4.3.1. One trivial example for the d parameter is the *perpendicular distance*. Take a (u, v) point from the polygon, and drop a perpendicular line to each side. The (normalized) lengths of the resulting line segments can be used as a distance parameter, see Fig. 4.10a. A variation of this method is the *barycentric parameterization*, where the d parameter is computed as the relative area of the triangle defined by the (u, v) point and the endpoints of the i-th side.

One particular family of parameterization methods is called *line sweeps*, where the s parameter lines constitute a continuum of straight lines, sweeping from the $(i-1)$-th side to the $(i+1)$-th. The simplest member of this family is *bilinear* parameterization. Let us look at the parameterization for the i-th side of the polygon, defined by p_{i-1} and p_i. A (u, v) point can be written as the linear interpolant of two points on the $(i-1)$-th and $(i+1)$-th sides, that are themselves linear interpolants of p_{i-2}—p_{i-1}, and p_i—p_{i+1}, respectively, see Figures 4.8 and 4.10b. Thus

$$(u, v) = (1 - s_i)\left((1 - d_i)p_{i-1} + d_i p_{i-2}\right) + s_i\left((1 - d_i)p_i + d_i p_{i+1}\right).$$

To obtain (s_i, d_i) for a given (u, v), a quadratic equation needs to be solved.

Another classic parameterization scheme is the *radial distance*, proposed by Charrot and Gregory [3], where the $(i-1)$-th and $(i+1)$-th sides of the

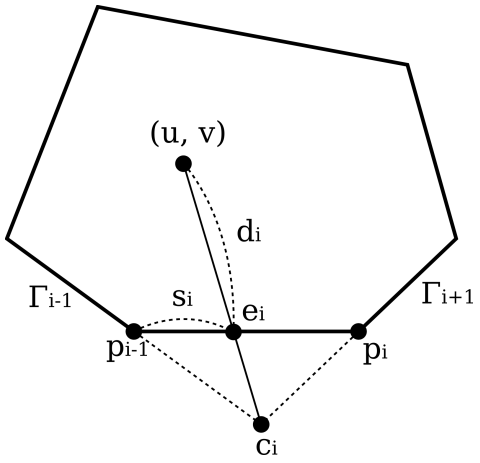


Figure 4.9: Constructing the radial parameterization.

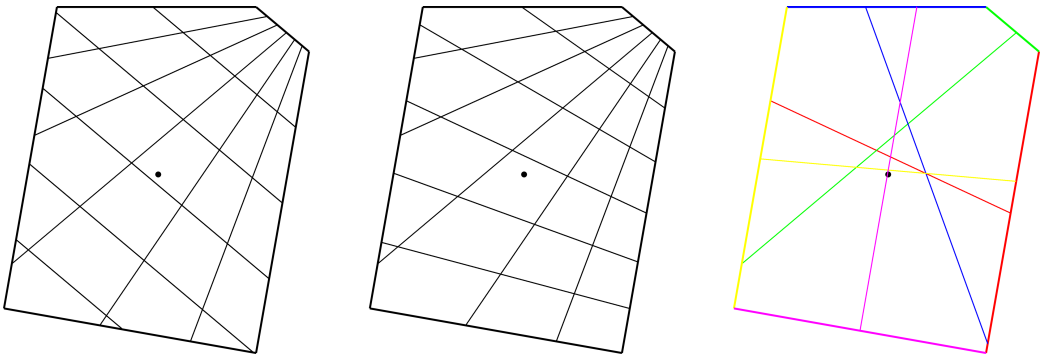
polygon are elongated (see Fig. 4.9), yielding a point c_i. The intersection of the i-th side with a radial line going through c_i and (u, v) produces the (s, d) parameters. Fig. 4.10c shows an example.

There is one natural requirement for all parameterizations. We wish to map the middle parameter lines of the individual ribbons "somewhere" in the middle of the n-sided domain. Our analysis has shown that for the above methods this property is often not satisfied, which may lead to undesirable 3D shape artifacts. On the right-hand side of Fig. 4.10 the map of the central constant parameter lines is shown for a five-sided patch, using three different parameterization schemes. Observe the skewed location, running far from the approximate center point of the domain, especially in the radial parameterization case. As a consequence, the middle of the ribbons would contribute to the n-sided patch in a fairly asymmetric manner.

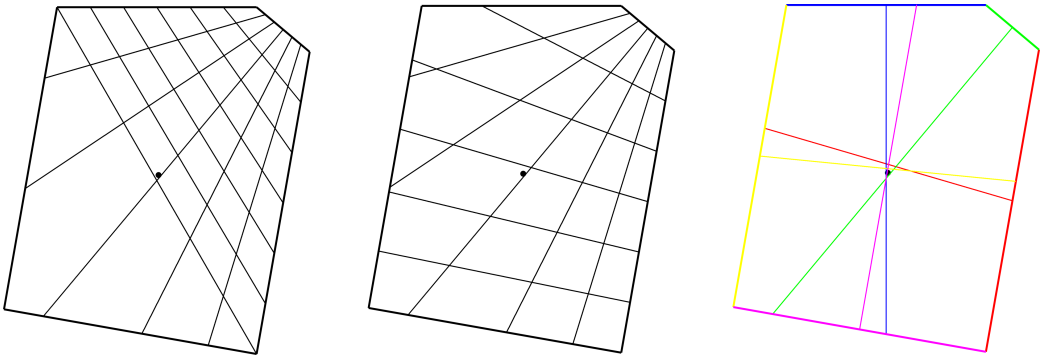
This motivated us to develop a new line sweep method, which guarantees a balanced ribbon mapping, as follows.

4.3.1 Central Line Sweep

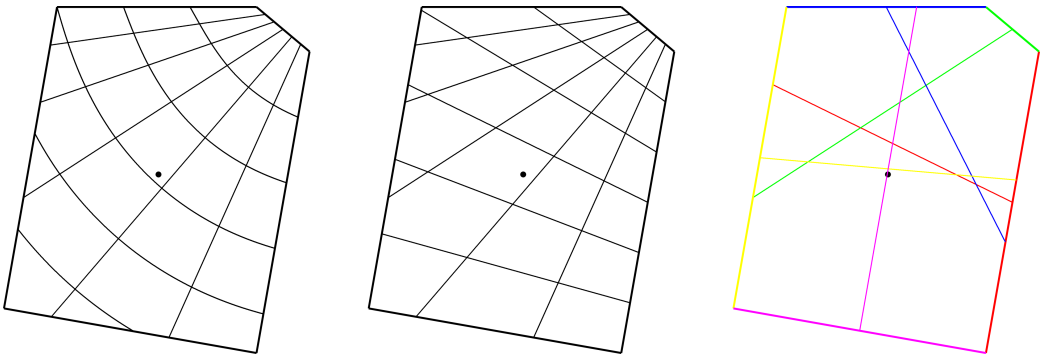
The goal of this method is to force the central s constant parameter line to go through the center of the polygonal domain (see Fig. 4.11). The exact computations are detailed below.



(a) Perpendicular parameterization



(b) Bilinear parameterization



(c) Radial parameterization

Figure 4.10: Constant parameter lines of different schemes: s and d lines based on the top right side (left image), s lines based on the right and top right sides (middle image) and $s = \frac{1}{2}$ lines for all sides (right image).

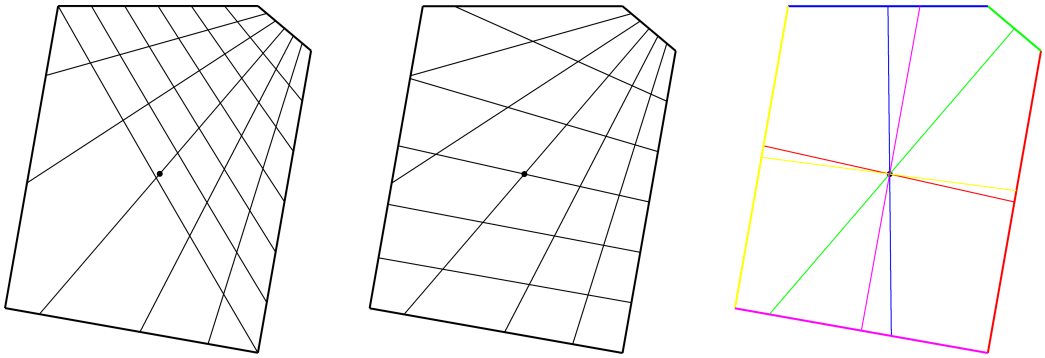


Figure 4.11: Constant parameter lines for the central line sweep parameterization.

The center of the polygon $C = (c^u, c^v)$ can be computed as the weighted sum of the polygon vertices. The weights can be uniform, but it is better to use the summed length of the adjacent sides as weights, in order not to be biased by a sequence of short polygon sides, i.e.,

$$C = \frac{1}{\sum_i \mu_{i,i-1}} \sum_i (\mu_{i+1,i} + \mu_{i,i-1}) p_i,$$

where $\mu_{i,i-1} = \|p_i - p_{i-1}\|$.

Let us look at the mapping for side i, defined by vertices p_{i-1} and p_i. Without loss of generality, place p_i on the u-axis, with p_{i-1} being at the origin (see Fig. 4.12). Assuming that the reverse mapping is done by a function $r_i(s_i, d_i)$, we know that $r_i(\frac{1}{2}, d_i^c) = C$ for some unknown d_i^c distance parameter. Now a linear-by-quadratic map is introduced:

$$r_i(s_i, d_i) = p_i s_i + \left[w_l (1 - s_i)^2 + w_c 2(1 - s_i) s_i + w_r s_i^2 \right] d_i,$$

where the vectors w_l, w_r and w_c define the direction of the quadratic sweep. While the first two are defined by the direction vectors of the adjacent sides, $w_c = (w_c^u, w_c^v)$ remains unknown. To simplify the calculation, let us require that

$$w_c^v = \frac{w_l^v + w_r^v}{2}.$$

Now if we examine the center point, we find that

$$c^v = \frac{1}{4}\left[w_l^v + 2w_c^v + w_r^v\right] d_i^c,$$

thus $d_i^c = \frac{2c^v}{w_l^v + w_r^v}$. From the other coordinate equation

$$c^u = \frac{p_i^u}{2} + \frac{1}{4}\left[w_l^u + 2w_c^u + w_r^u\right] d_i^c,$$

so we can express the missing u component of w_c. All that is left is to solve the quadratic equation

$$\begin{aligned} d_i(u,v) &= \frac{u - p_i^u s_i}{w_l^u(1-s_i)^2 + 2w_c^u(1-s_i)s_i + w_r^u s_i^2} \\ &= \frac{v}{w_l^v(1-s_i) + w_r^v s_i}, \end{aligned}$$

where s_i stands for $s_i(u,v)$. After some straightforward algebra, we arrive at the familiar expression $\eta_0 s_i^2 + \eta_1 s_i + \eta_2 = 0$, where

$$\begin{aligned} \eta_0 &= p_i^u(w_r^v - w_l^v) + v(w_r^u - 2w_c^u + w_l^u), \\ \eta_1 &= p_i^u w_l^v + u(w_l^v - w_r^v) + 2v(w_c^u - w_l^u), \\ \eta_2 &= v w_l^u - u w_l^v. \end{aligned}$$

4.3.2 Variants of the Distance Parameter

If we also want to constrain the $d = \frac{1}{2}$ constant parameter line to go through the center point, we can do this by post-processing:

$$\hat{d}_i(u,v) = \left[(1-s_i)^2 + \frac{1-d_i^c}{d_i^c} 2(1-s_i)s_i + s_i^2\right] d_i.$$

The effect of this change is shown in the left-hand image of Fig. 4.13.

Another natural choice for d is the distance from the sweep line's foot-point, as in radial parameterization:

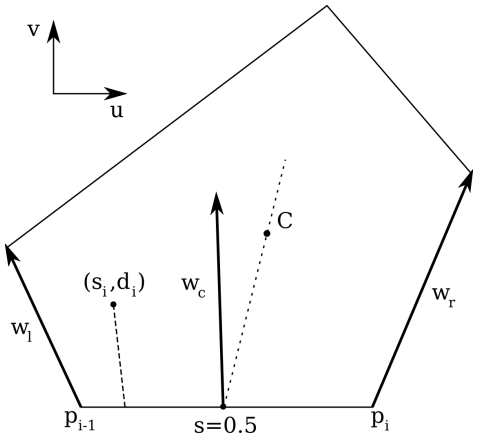


Figure 4.12: Computing the central line sweep parameterization.

$$d_i^{\text{foot}}(u,v) = \|(u,v) - (s_i p_i^u, 0)\|,$$

depicted in the middle image. Arguably this is more desirable than the linear function used in the previous section. Using the same technique, we can force its central distance parameter line to go through the center of the domain:

$$\begin{aligned}\tilde{d}_i(u,v) \;=\; & \left[(1-s_i)^2\frac{1}{\|w_l\|} + s_i^2\frac{1}{\|w_r\|} + \right. \\ & \left.\left(\frac{2}{\|C-(p_i^u/2,0)\|} - \frac{1}{\|w_l\|} - \frac{1}{\|w_r\|}\right)(1-s_i)s_i\right] d_i^{\text{foot}}.\end{aligned}$$

The right-hand image of Fig. 4.13 shows an example.

4.4 Multi-sided Coons Patch

The next few sections deal with the generalization of Coons patches to n sides. The proposed interpolating surface uses the same principles as the original Coons patch, i.e., combines linear side interpolants and corner correction patches, and thus can be regarded as a natural generalization, even though the surface presented here will not revert to a Coons patch for quadrilateral configurations.

In Section 4.4.1 a new patch construction is introduced, and Appendix A

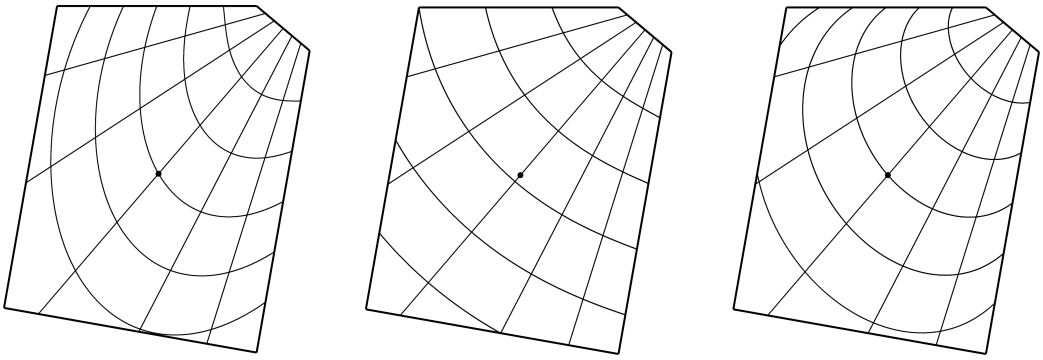


Figure 4.13: Different distance parameterizations: centralized (left), footpoint (center) and centralized footpoint (right).

provides a proof of its basic interpolation properties. In Sections 4.4.2 and 4.4.3 we show suitable functions for blending and parameterization, respectively.

4.4.1 Coons Patch Generalization

With appropriate parameterization functions $s_i(u,v)$ and $d_i(u,v)$, Equation (4.1) can be generalized as

$$S(u,v) = \sum_{i=1}^{n} R_i(s_i,d_i)\cdot B_i(d_1,\ldots,d_n) - \sum_{i=1}^{n} Q_{i,i-1}(s_i,s_{i-1})\cdot B_{i,i-1}(d_1,\ldots,d_n),$$

where $B_i = B_{i,i-1} + B_{i+1,i}$ is a *side blend*, and $B_{i,i-1}$ is a *corner blend* function. $B_{i,i-1}$ is required to have the following properties:

$$\begin{aligned} B_{i,i-1}(d_1,\ldots,d_j=0,\ldots d_n) &= 0,\ j \notin \{i-1,i\} && (4.2)\\ B_{i,i-1}(d_1,\ldots,d_i=0,\ldots,d_n) &+ \\ B_{i+1,i}(d_1,\ldots,d_i=0,\ldots d_n) &= 1, && (4.3)\\ \frac{\partial}{\partial d_j} B_{i,i-1}(d_1,\ldots,d_j=0,\ldots,d_n) &= 0,\ \forall j. && (4.4) \end{aligned}$$

Section 4.4.2 will show a possible construction. Equation 4.2 means that the blend function vanishes on all sides not connected to the corner. Boundary interpolation is satisfied due to Eq. 4.3, and finally Eq. 4.4 is needed for

proving the tangential properties of the patch (see Appendix A). From these, it follows that

$$\begin{aligned}
B_i(d_1,\ldots,d_j=0,\ldots,d_n) &= 0,\ j\notin\{i-1,i,i+1\}\\
B_i(d_1,\ldots,d_i=0,\ldots,d_n) &= 1,\\
B_{i-1}(d_1,\ldots d_i=0,\ldots d_n) &= B_{i,i-1}(d_1,\ldots,d_n),\\
B_{i+1}(d_1,\ldots d_i=0,\ldots d_n) &= B_{i+1,i}(d_1,\ldots,d_n),\\
\frac{\partial}{\partial d_j}B_i(d_1,\ldots,d_j=0,\ldots,d_n) &= 0,\ \forall j.
\end{aligned}$$

There are several requirements for the (s_i, d_i) parameterization, as well. First, for s_i we need

$$s_i \in [0,1], \tag{4.5}$$

and for a point on the i-th side, the parameterization has to satisfy that

$$d_i = 0, \tag{4.6}$$

$$s_{i-1} = 1, \qquad s_{i+1} = 0, \tag{4.7}$$

$$d_{i-1} = s_i, \qquad d_{i+1} = 1 - s_i, \tag{4.8}$$

$$\frac{\partial d_{i-1}}{\partial u} = \frac{\partial s_i}{\partial u}, \qquad \frac{\partial d_{i-1}}{\partial v} = \frac{\partial s_i}{\partial v}, \tag{4.9}$$

$$\frac{\partial d_{i+1}}{\partial u} = -\frac{\partial s_i}{\partial u}, \qquad \frac{\partial d_{i+1}}{\partial v} = -\frac{\partial s_i}{\partial v}. \tag{4.10}$$

We can see that there are more restrictions, than for the patches reviewed in Section 4.1 (only (4.5)–(4.7) are required for those), but in Section 4.4.3 various techniques will be presented for creating eligible parameterizations.

4.4.2 Blending functions

Recall the blending function $B_{i,i-1}$ described in [3]:

$$B_{i,i-1}(d_1,\ldots,d_n) = \frac{\prod_{k\notin\{i,i-1\}} d_k^2}{\sum_l \prod_{k\notin\{l,l-1\}} d_k^2} = \frac{D_{i,i-1}^2}{\sum_l D_{l,l-1}^2}.$$

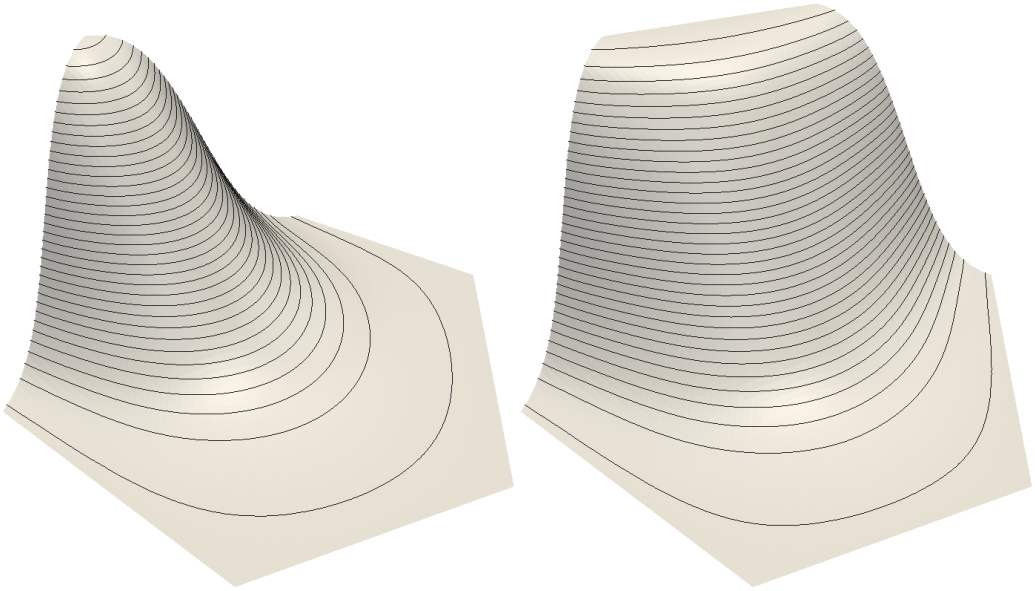


Figure 4.14: Blending functions $B_{i,i-1}$ and B_i over a six-sided domain.

Assumptions (4.2) and (4.3) are trivially true. In order to prove (4.4), examine the following difference:

$$\lim_{\Delta \to 0} \frac{B_{i,i-1}(d_1, \ldots, d_j = \Delta, \ldots, d_n) - B_{i,i-1}(d_1, \ldots, d_j = 0, \ldots, d_n)}{\Delta}.$$

First assume that $j \notin \{i, i-1\}$. Consequently, the right hand side of the numerator is 0 and the left hand side contains Δ^2 in the product, so it can be reformulated as

$$\lim_{\Delta \to 0} \frac{\Delta \prod_{k \notin \{i,i-1,j\}} d_k^2}{\Delta^2 \sum_{l \notin \{j+1,j\}} \prod_{k \notin \{l,l-1,j\}} d_k^2 + \sum_{l \in \{j+1,j\}} D_{l,l-1}^2} = 0,$$

since the right hand side of the denominator does not contain Δ. Now assume that $j \in \{i, i-1\}$. Introducing the notations $\alpha = D_{i,i-1}^2$, $\beta = \sum_{l \notin \{j,j+1\}} D_{j,l,l-1}^2$ and $\gamma = D_{j,j-1}^2 + D_{j+1,j}^2$, the expression can be rewritten as

$$\lim_{\Delta \to 0} \frac{1}{\Delta} \left(\frac{\alpha}{\Delta^2 \beta + \gamma} - \frac{\alpha}{\gamma} \right) = \lim_{\Delta \to 0} \frac{1}{\Delta} \left(\frac{\alpha\gamma - (\Delta^2 \beta + \gamma)\alpha}{\Delta^2 \beta\gamma + \gamma^2} \right)$$

$$= \lim_{\Delta \to 0} -\frac{\Delta\alpha\beta}{\Delta^2\beta\gamma + \gamma^2} = 0.$$

Thus this blend function adheres to all requirements. Fig. 4.14 shows a graphical representation of $B_{i,i-1}$ and B_i. The overall blending effect is demonstrated on Figures 4.15 and 4.16, using color coding. The blending functions used here are computed by the d coordinates of the interconnected parameterization (based on the central sweep method), see the next section. Black lines show the points where the contribution of a side (corner) reaches approximately 90%; white lines show the points where the blending terms corresponding to two sides (corners) are equal.

4.4.3 Parameterizations

While the parameterization methods presented in Section 4.3 work for conventional transfinite surface interpolation, they do not satisfy all the requirements (4.5)–(4.10). In order to be able to satisfy these, we need more conformity between the s and d parameters of adjacent sides. The following sections explore three alternatives for such constructions.

4.4.3.1 Interconnected Parameterizations

Take functions $s_i(u, v)$ that give 0 for every point on the $(i-1)$-th side and 1 for those on the $(i+1)$-th side; for all other points inside the convex domain, they return a value in $[0, 1]$. For example, the s coordinates of radial line sweeps are such functions. These naturally satisfy (4.5) and (4.7). Define a blending function $\alpha(t) \in [0, 1] \to [0, 1]$ with $\alpha(0) = 1$ and $\alpha(1) = \alpha'(0) = \alpha'(1) = 0$. Examples are the Hermite function $\alpha_0(t)$ from Section 4.1.1, or a variation of the rational blend function presented in the previous section: $\alpha(t) = \frac{(1-t)^2}{t^2+(1-t)^2}$. Now we can define d_i by means of s_{i-1} and s_{i+1} as follows:

$$d_i(u, v) = (1 - s_{i-1}(u, v)) \cdot \alpha(s_i) + s_{i+1}(u, v) \cdot \alpha(1 - s_i).$$

If we are on the i-th side, $s_{i-1} = 1$ and $s_{i+1} = 0$, so $d_i = 0$, satisfying (4.6).

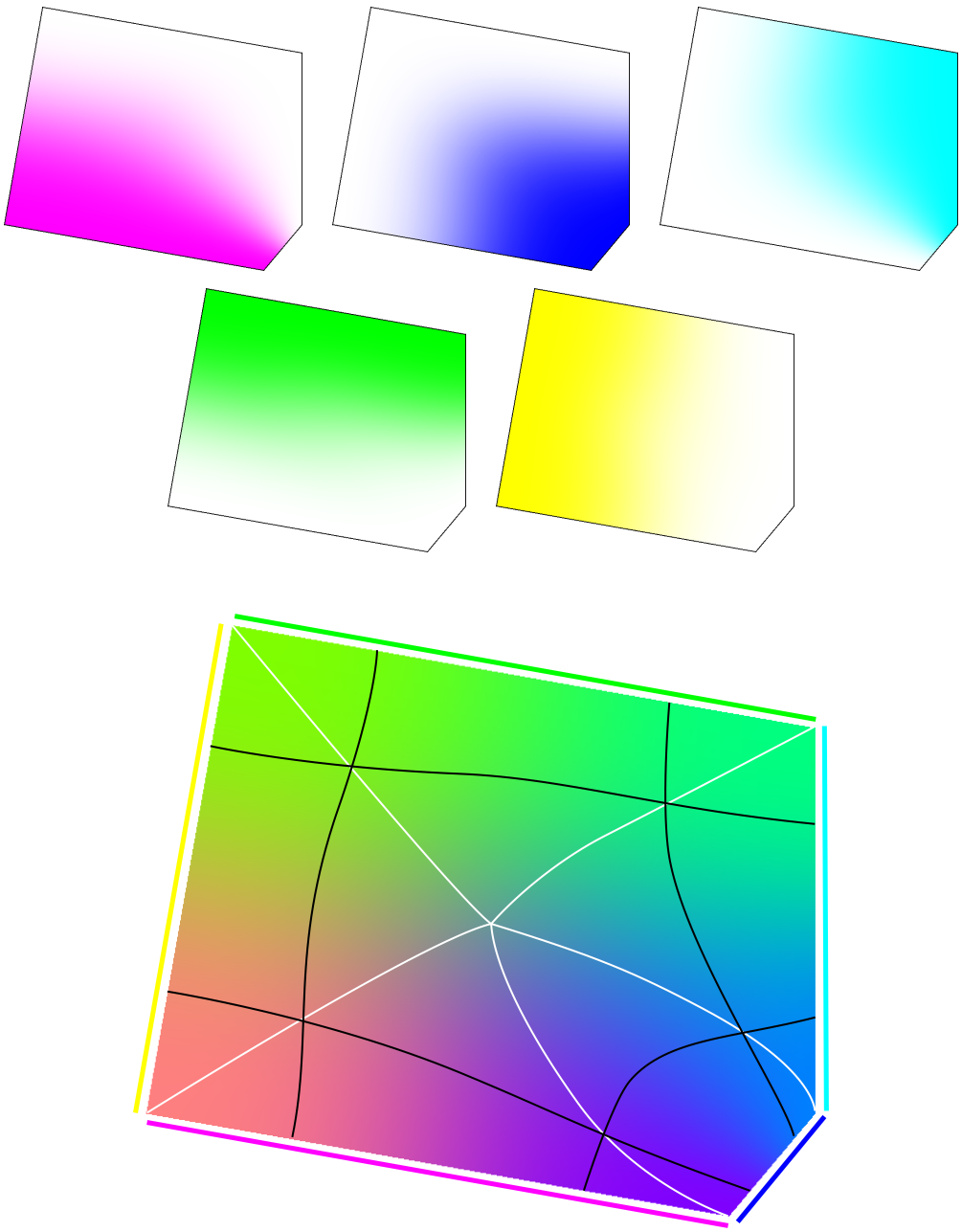


Figure 4.15: Distribution of the blending functions B_i using different colors for each side

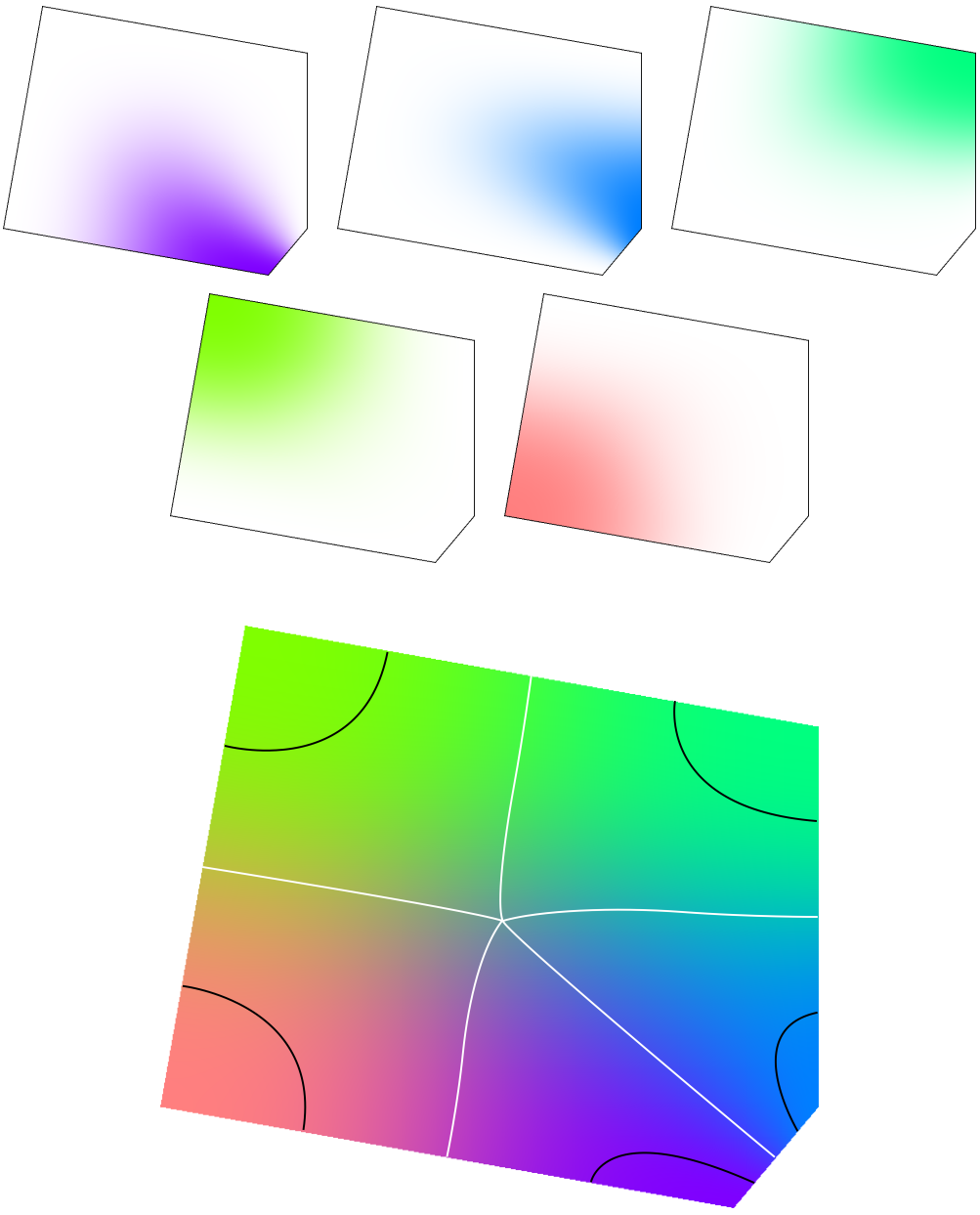


Figure 4.16: Distribution of the blending functions $B_{i,i-1}$ using different colors for each corner.

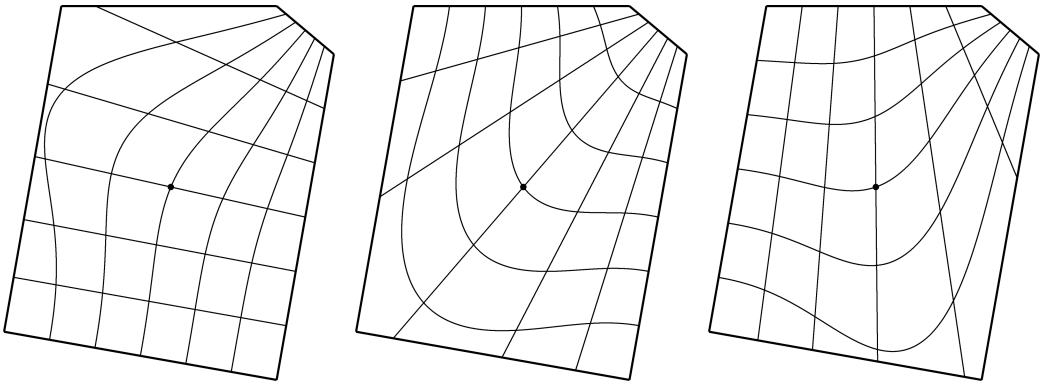


Figure 4.17: Constant parameter lines of the interconnected parameterization.

Still on the i-th side, d_{i-1} and its derivative are the same as s_i:

$$\begin{aligned} d_{i-1} &= (1 - s_{i-2}) \cdot \alpha(s_{i-1}) + s_i \cdot \alpha(1 - s_{i-1}) = s_i, \\ \frac{\partial d_{i-1}}{\partial u} &= \frac{\partial}{\partial u}(1 - s_{i-2}) \cdot \alpha(s_{i-1}) + \frac{\partial}{\partial u} s_i \cdot \alpha(1 - s_{i-1}) \\ &= \frac{\partial s_i}{\partial u}, \end{aligned}$$

because the derivatives of the blend function vanish. The same reasoning works for the derivative by v. Similarly

$$\begin{aligned} d_{i+1} &= (1 - s_i) \cdot \alpha(s_{i+1}) + s_{i+2} \cdot \alpha(1 - s_{i+1}) = 1 - s_i, \\ \frac{\partial d_{i+1}}{\partial u} &= \frac{\partial}{\partial u}(1 - s_i) \cdot \alpha(s_{i+1}) + \frac{\partial}{\partial u} s_{i+2} \cdot \alpha(1 - s_{i+1}) \\ &= \frac{\partial}{\partial u}(1 - s_i) = -\frac{\partial s_i}{\partial u}, \end{aligned}$$

so the requirements (4.8), (4.9) and (4.10) are all satisfied.

Fig. 4.17 shows constant s and d lines for this parameterization (using the central line sweep parameterization as a basis). The first image is based on the right side of the polygon; the second image is based on the small side at the top-right; and the third image is based on the top side. Note that all lines of the second image start the same way (in a differential sense) as their counterparts in the first and third images.

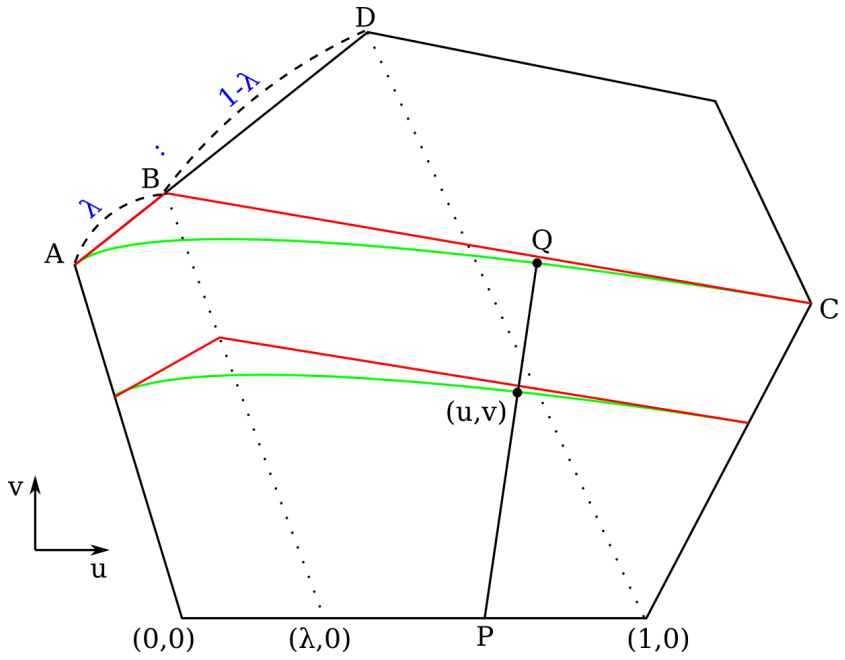


Figure 4.18: Parabolic parameterization — left parabola case.

4.4.3.2 Parabolic Parameterization

While the interconnected parameterization scheme satisfies all requirements, it is easy to see that its constant parameter lines always have inflections. In this section we will show a more natural parameterization. However, this comes with a price: its computation is much more complex, involving the solution of a third or fourth degree equation. The following method uses quadratic curves, but it can be extended to curves of higher degree.

This method also requires a base parameterization; we will use bilinear parameterization here, but the same computation can be done for radial or other parameterizations as well (note, though, that the equation's degree may be higher for other parameterizations, such as for central line sweep). For ease of computation, we will assume that the base side lies on the u axis, with its endpoints at the origin and $(1, 0)$. Points A and C are the distant endpoints of the previous and next domain polygon sides, respectively, see Fig. 4.18.

The idea is the following: we define two sets of curves, one for the left

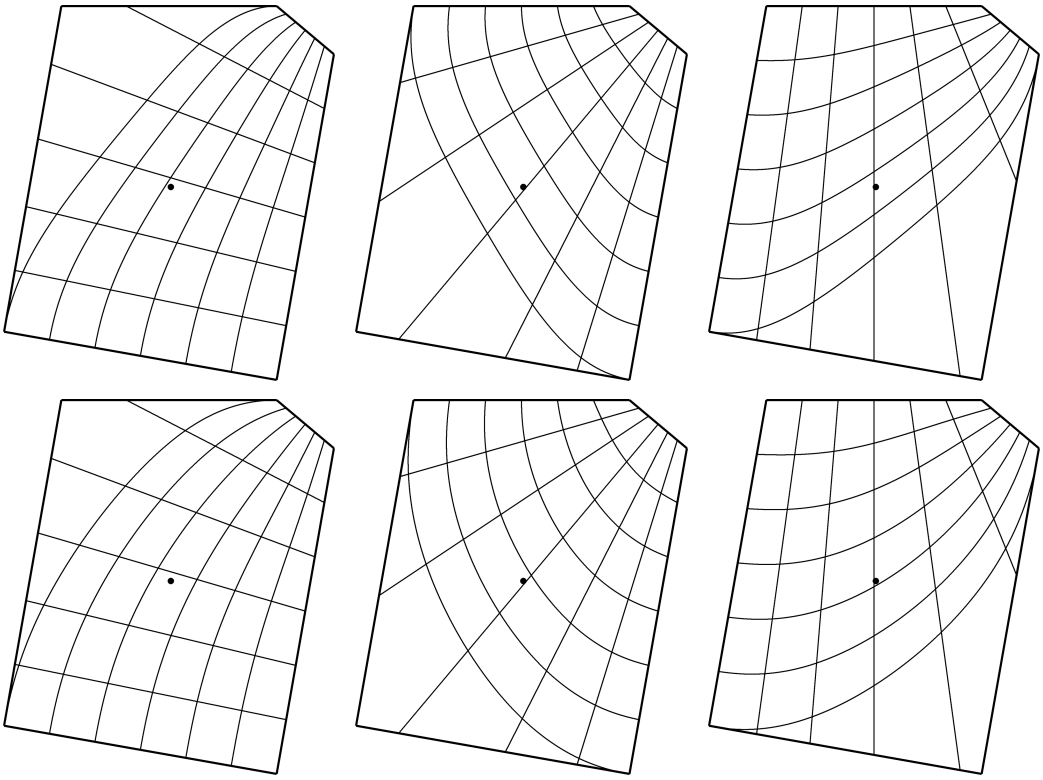


Figure 4.19: Constant parameter lines of the parabolic parameterization with $\lambda = \frac{1}{2}$ (top) and $\lambda = \frac{3}{4}$ (bottom).

side, and one for the right. The curves have the same positions and tangents near the left (right) side, as the bilinear sweeping lines for that side. Then we blend the two curve sets together. Here we will only deal with the left curve set, but the computations are analogous for the right side.

We place point B as in the figure, at λ proportion of the AD side, where λ is the so-called *fullness parameter*. The ABC triangle defines a quadratic Bézier curve, drawn in green. Note that the AB segment is the sweeping line of the bilinear parameterization based on the left side of the polygon, i.e., the $A - (0,0) - (1,0) - D$ quadrilateral. Other sweeping lines define different Bézier curves (with the middle point of the triangle being on the constant bilinear parameter defined by B). One of these curves goes through the (u,v) point, at parameter $\hat{s}$. Let Q be the point of the curve defined by the ABC triangle at $\hat{s}$. Connecting Q and (u,v) crosses the baseline at P.

Trivially

$$v = \frac{u - p^u}{q^u - p^u} \cdot q^v,$$

which can also be written as

$$(q^u - p^u)v + q^v(p^u - u) = 0.$$

From the definition of Q, we have

$$Q = (1 - \hat{s})^2 A + 2(1 - \hat{s})\hat{s}B + \hat{s}^2 C,$$

and similarly, $P = (p^u, 0)$, where

$$p^u = (1 - \hat{s})^2 \cdot 0 + 2(1 - \hat{s})\hat{s} \cdot \lambda + \hat{s}^2 \cdot 1 = (1 - 2\lambda)\hat{s}^2 + 2\lambda\hat{s}.$$

This leads to a fourth-degree equation in $\hat{s}$:

$$\eta_0 \hat{s}^4 + \eta_1 \hat{s}^3 + \eta_2 \hat{s}^2 + \eta_3 \hat{s} + \eta_4 = 0,$$

where

$$\begin{aligned}
\eta_0 &= (1 - 2\lambda)(a^v - 2b^v + c^v), \\
\eta_1 &= 2\lambda(a^v - 2b^v + c^v) + (1 - 2\lambda)2(b^v - a^v), \\
\eta_2 &= (a^u - 2b^u + c^u)v - (a^v - 2b^v + c^v)u + \\
&\quad 4\lambda(b^v - a^v) + (1 - 2\lambda)(a^v - v), \\
\eta_3 &= (2(b^u - a^u) - 2\lambda)v - 2(b^v - a^v)u + 2\lambda a^v, \\
\eta_4 &= a^u v - a^v u.
\end{aligned}$$

Now we define d^l as

$$d^l = \frac{v}{q^v} = v / \left((1 - \hat{s})^2 a^v + 2\hat{s}(1 - \hat{s})b^v + \hat{s}^2 c^v\right).$$

Note that d^l behaves the same way near the left side as the sweeplines based on it. The same can be done for the right side, resulting in d^r. Finally, we

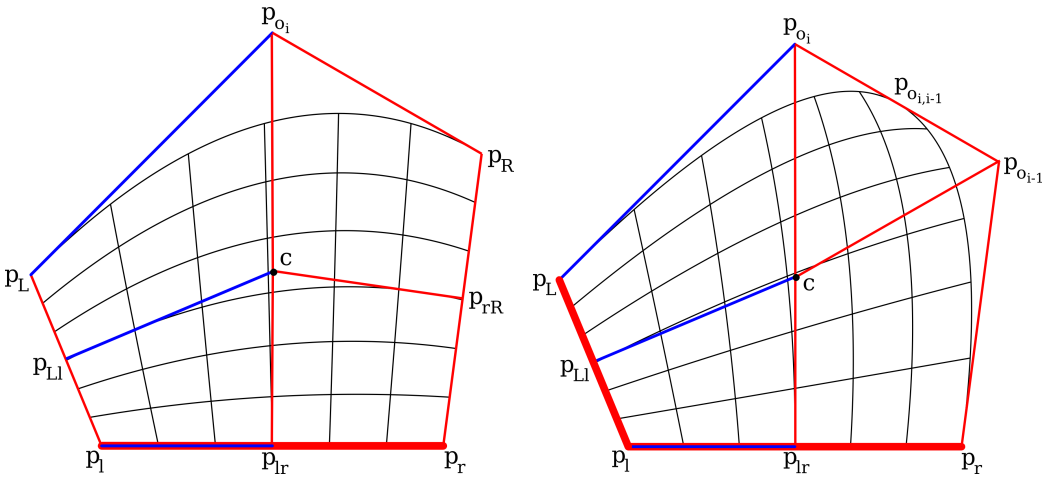


Figure 4.20: Control net and constant parameter lines of the biquadratic parameterization (left: side-based, right: corner-based).

compute s from the bilinear parameterization, and then use it to blend the two d values:

$$d = \alpha_0(s)d^l + \alpha_1(s)d^r.$$

An example is shown in Fig. 4.19. Experiments show that the choice of $\lambda = \frac{3}{4}$ seems to generate nice constant parameter lines. It is worth noting, though, that setting λ to $\frac{1}{2}$ eliminates η_0 and thus we will only have a third-degree equation. This parameterization satisfies all requirements, see the proof in Appendix B.

4.4.3.3 Biquadratic Parameterization

The patch defined in Section 4.4 is using side-based linear interpolants, but also corner-based correction patches. In the following, two parameterizations will be used, one for the ribbons (side-based parameterization), and one for the correction patches (corner-based parameterization). The idea is that a linear side interpolant should have the same parameterization near the side adjacent to its base as the correction patch of the corresponding corner, enabling the correction patch to cancel out the "garbage" generated by the ribbons. This will be achieved by biquadratic maps similar to the overlap patch parameterization [38].

We place a three-by-three control net on the domain, and the resulting planar surface defines a parameterization (see Fig. 4.20). We need the inverse of this biquadratic map, which can be computed by numerical methods, such as the Newton–Raphson algorithm. Note that the control points on the left-hand side are the same in both the side-based and the corner-based variants, so the two parameterizations behave identically near the left boundary. In the following we will examine the construction of these biquadratic control nets.

Imagine a domain with its i-th side on the u axis. For the side-based case, the control points are defined as

$$\begin{bmatrix} p_l & p_{lr} & p_r \\ p_{Ll} & c & p_{rR} \\ p_L & p_{o_i} & p_R \end{bmatrix},$$

where p_l and p_r are the left and right endpoints of the i-th (base) side, p_L (p_R) is the left (right) endpoint of the domain edge on the left (right) of the i-th side, c is the center of the domain and p_{o_i} is domain vertex opposite to the i-th side (for even-sided polygons p_{o_i} is the midpoint of the opposite side). The notation p_{ij} means the midpoint of the segment defined by p_i and p_j.

When the basis is the corner at p_l, the control points are

$$\begin{bmatrix} p_l & p_{lr} & p_r \\ p_{Ll} & c & p_{o_{i-1}} \\ p_L & p_{o_i} & p_{o_{i,i-1}} \end{bmatrix},$$

where $p_{o_{i,i-1}}$ is the vertex opposite to the vertex defined as the intersection of the i-th and $(i-1)$-th sides (for odd-sided polygons $p_{o_{i,i-1}}$ is the midpoint of the opposite side).

Note that the first two columns are the same as in the side-based case. Also the first two rows have their counterparts in the $(i-1)$-th side-based biquadratic as well. These relationships tell us that the parameterization based on the $i, i-1$ corner behaves the same way for a point on the $(i-1)$-th

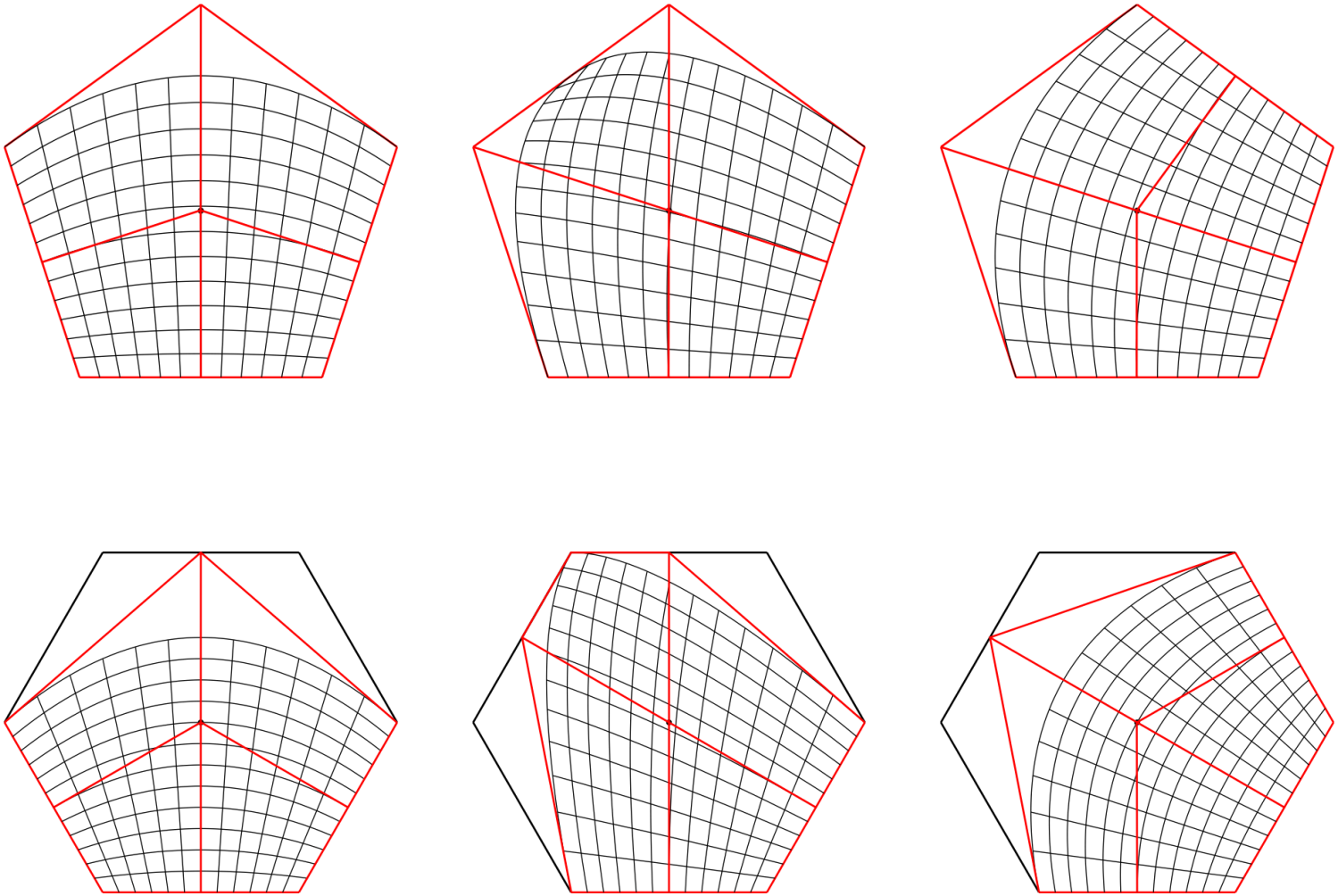


Figure 4.21: Control nets using only points of the domain polygon, based on the bottom side (left image), the right side (right image) and both (middle image).

Figure 4.22: Control nets using Bézier midpoints, based on the bottom side (left image), the right side (right image) and both (middle image).

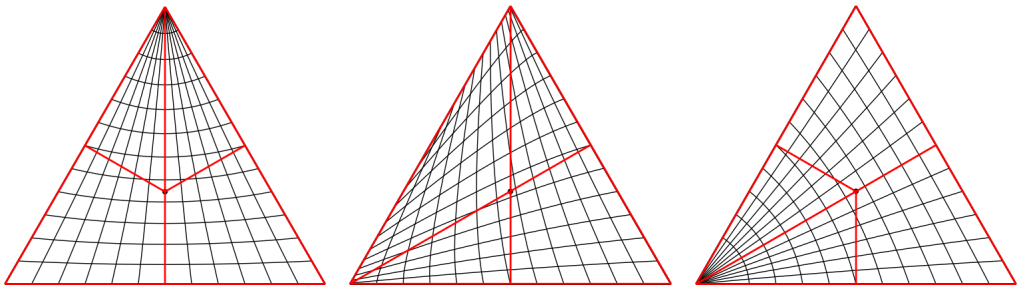


Figure 4.23: Control nets for triangular domains, based on the bottom side (left image), the right side (right image) and both (middle image).

side as the parameterization based on the i-th side; and for a point on the i-th side as the parameterization based on the $(i-1)$-th side. This is important, because it means that all superfluous data that comes into the equations from the ribbon interpolation can be eliminated by corner-parameterized correction patches.

Figure 4.21 shows some examples for even- and odd-sided regular polygons. Scrutinous readers may have noticed that there is a perceivable effect of shearing away from the base side(s). This effect can be lessened by using midpoints of Bézier curves instead of all the "opposite" vertices and midpoints, as in Fig. 4.22. Comparison of the two sets show that significant changes appear only far from the base side(s), where the blending function almost vanishes, thus the two schemes are virtually identical. There is one special case: three-sided segments have a very intuitive, singular parameterization (depicted in Figure 4.23).

So in our transfinite surface scheme, side-based parametrizations are used for ribbons (R_i) and blends ($B_{i,i-1}$), but corner-based parametrizations are used for the correction patches ($Q_{i,i-1}$). We will use the convention of referring to corner-based parameters by (s_i^*, d_i^*), while retaining (s_i, d_i) for side-based ones.

The parametrization requirements (4.5)–(4.7) are trivially satisfied for side-based schemes. Linearity on the sides also guarantees (4.8), even between a side-based and a corner-based patch. The remaining constraints do not hold for either type, but are valid in-between: the d_{i-1} side-parameters

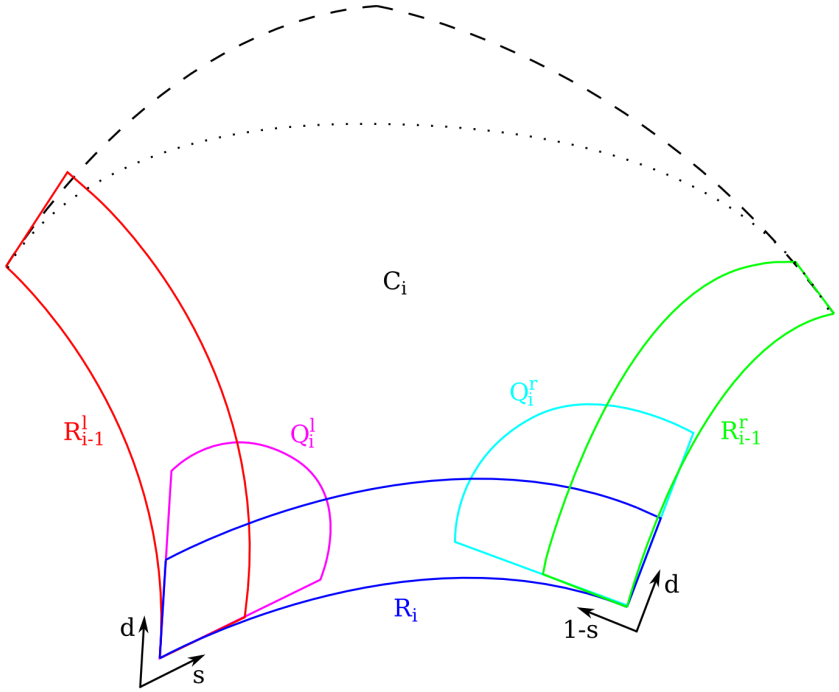


Figure 4.24: Constituents of a curved ribbon.

are equal to the s_i^* corner-parameters in derivative sense (a corner-parameter with index i is associated with the biquadratic patch based on the i-th and $(i-1)$-th edges, using the i-th edge as constant $d_i^* = 0$). Similarly all requirements are satisfied in this way.

Unfortunately this means that we have to alter the original definition of the surface, substituting $Q_i(s_i^*, d_i^*) = Q_{i,i-1}(s_i^*, 1 - d_i^*)$ for $Q_{i,i-1}(s_i, s_{i-1})$:

$$S(u,v) = \sum_i R_i(s_i, d_i) \cdot B_i(d_1, \ldots, d_n) - \sum_i Q_i(s_i^*, d_i^*) \cdot B_{i,i-1}(d_1, \ldots, d_n).$$

The patch still interpolates the boundary data correctly, as it is proven in Appendix A.3.

4.5 Curved Side Interpolants

The patches in the previous sections use linear ribbons. These have the advantage of being very simple and can be computed efficiently. On the

other hand, highly curved surfaces deviate far from their linear interpolants, thereby decreasing stability. Unintuitive bulges, coming from contradicting cross-derivatives, pose a common problem. Recall, however, that conventional side interpolant-based transfinite interpolation surfaces do not depend on the linearity of the ribbons. We can thus define curved ribbons that follow the shape of the surface more closely.

Let $C_i(s_i, d_i)$ denote the curved ribbon for the i-th side. In order to simplify the notation, we will drop the indices of s and d when talking about only one curved ribbon, as it does not cause any ambiguity. The definition of C_i is as follows (see Fig. 4.24):

$$\begin{aligned} C_i(s,d) &= R_i^l(s,d)H(s) + R_i(s,d)H(d) + R_i^r(s,d)H(1-s) \\ &- Q_i^l(s,d)H(s)H(d) - Q_i^r(s,d)H(1-s)H(d), \end{aligned}$$

where

$$\begin{aligned} R_i^l(s,d) &= R_{i-1}(1-d,s) = P_{i-1}(1-d) + sT_{i-1}(1-d) \\ R_i^r(s,d) &= R_{i+1}(d,1-s) = P_{i+1}(d) + (1-s)T_{i+1}(d) \\ Q_i^l(s,d) &= Q_{i,i-1}(s,1-d) = P_i(0) + sT_{i-1}(1) + dT_i(0) + sdW_{i,i-1} \\ Q_i^r(s,d) &= Q_{i+1,i}(d,s) \\ &= P_{i+1}(0) + dT_i(1) + (1-s)T_{i+1}(0) + d(1-s)W_{i+1,i} \end{aligned}$$

and $H(t)$ is a blend function, for example the Hermite blend function $\alpha_0(t)$.

It is interesting to note that the curved corner interpolants of the Gregory patch [3] can be defined in terms of ribbons and correction patches, as well:

$$R_{i,i-1}^{\text{Gregory}}(s_i, s_{i-1}) = R_{i-1}(s_{i-1}, s_i) + R_i(s_i, 1-s_{i-1}) - Q_{i,i-1}(s_i, s_{i-1}).$$

The curved ribbon defined above interpolates three consecutive boundary curves using blends, which makes it essentially a three-sided Coons patch. These ribbons can be used as side interpolants for conventional transfinite interpolation patches, but there is also a more natural blending scheme, as proposed in the next section.

4.6 Composite Ribbon Patch

The generalization of Coons patches, as introduced in Section 4.4, does not work with curved side interpolants. Here we propose a new representation that utilizes the properties of curved ribbons in order to eliminate the need for correction patches. This new n-sided transfinite surface patch is defined as

$$S(u,v) = \frac{1}{2}\sum_{i=1}^{n} C_i(s_i(u,v), d_i(u,v)) B_i(d_1(u,v), \ldots, d_n(u,v)),$$

where the parameterization satisfies the requirements listed in (4.5)–(4.10).

According to the characteristics of the B_i blend function, for any point on the i-th boundary all addends of the sum vanish except for C_{i-1}, C_i and C_{i+1}. Since each of these ribbons also interpolates the adjacent curves, the three ribbon points are the same. Their cumulative blend is

$$\begin{aligned} B_{i-1} + B_i + B_{i+1} &= B_{i,i-1} + B_i + B_{i+1,i} \\ &= (B_{i,i-1} + B_{i+1,i}) + B_i \\ &= 1 + 1 = 2, \end{aligned}$$

hence the division by two in the surface equation. So this patch also interpolates the boundary data — the full proof can be found in Appendix C.

The proof also shows that the parameterization constraints can be loosened. If we leave out the last two requirements (4.9 and 4.10), the patch remains valid, i.e., it will have the same tangent plane for every boundary point as the respective Coons ribbon, but it won't have the exact same tangent vector. This enables the use of many other parameterizations, such as the bilinear or the central line sweep parameterizations.

In other words, we have created a new transfinite surface representation that (i) has the same computational complexity as other conventional methods, (ii) uses curved side interpolants, and (iii) employs non-singular blend functions.

4.7 Test Results

In this research on transfinite surface interpolation we have presented a wide collection of algorithms with a large number of variables. The surfaces depend on the domain polygon, the parameterization scheme, the choice of transfinite interpolation patch, and several algorithm-dependent parameters, such as the fullness parameter in Section 4.4.3.2. Comparing all possible combinations would be extremely difficult, and one cannot help the feeling that any comparison is contrasting apples and oranges. The utmost attention was paid to make sensible examples, comparing the following surface patch configurations:

- Linear Ribbon patch (LR): The side interpolant-based surface reviewed in Section 4.1.2, using the alternative blending functions $B_i^{\text{side}} = \frac{D_i^2}{\sum_j D_j^2}$ based on perpendicular distances, as described at the beginning of Section 4.3. The domain is constructed using the "arcs by arc lengths" algorithm (Section 4.2.1), and the ribbons are parameterized by central line sweep (Section 4.3.1).

- Generalized Coons patch (GC): The surface representation introduced in Section 4.4, using the same domain as the LR patch, and parameterized by the central line sweep-based interconnected parameterization method (Section 4.4.3.1).

- Composite Ribbon patch (CR): The curved ribbon-based surface proposed in Section 4.6, using the same domain and parameterization as the GC patch.

Also note that the compatibility of ribbons needed for the GC patch and the CR patch was established by appropriate scaling and the use of Gregory's variable twists.

The first example in Fig. 4.25 shows the mean map of a model containing three-, four- and five-sided surfaces. The first image was created by a LR patch. Note the blue areas near the sides — these are artifacts of the blending function, which abruptly drops as we move away from the boundaries. The CR patch below is free of these unwanted curvature changes.

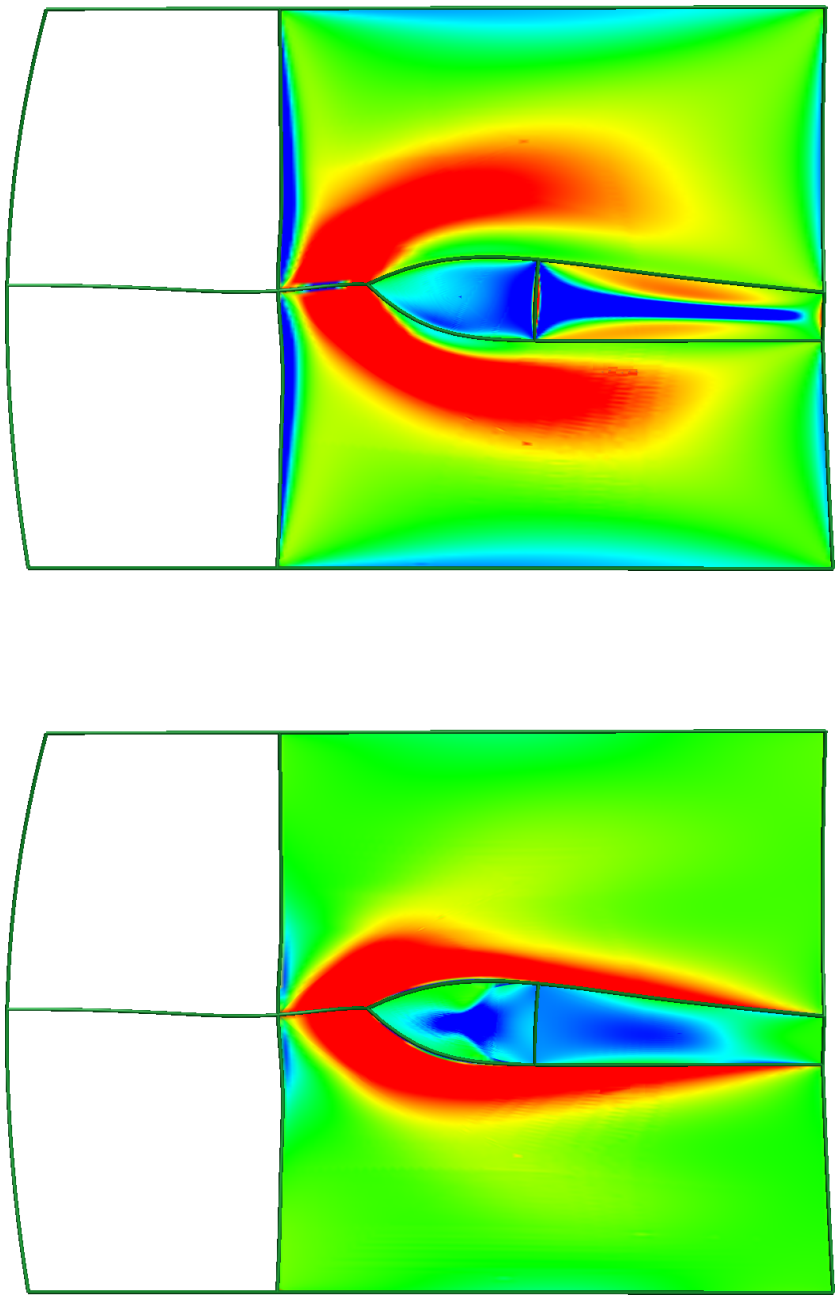


Figure 4.25: Mean map comparison of a model with four surfaces (top: LR, bottom: CR).

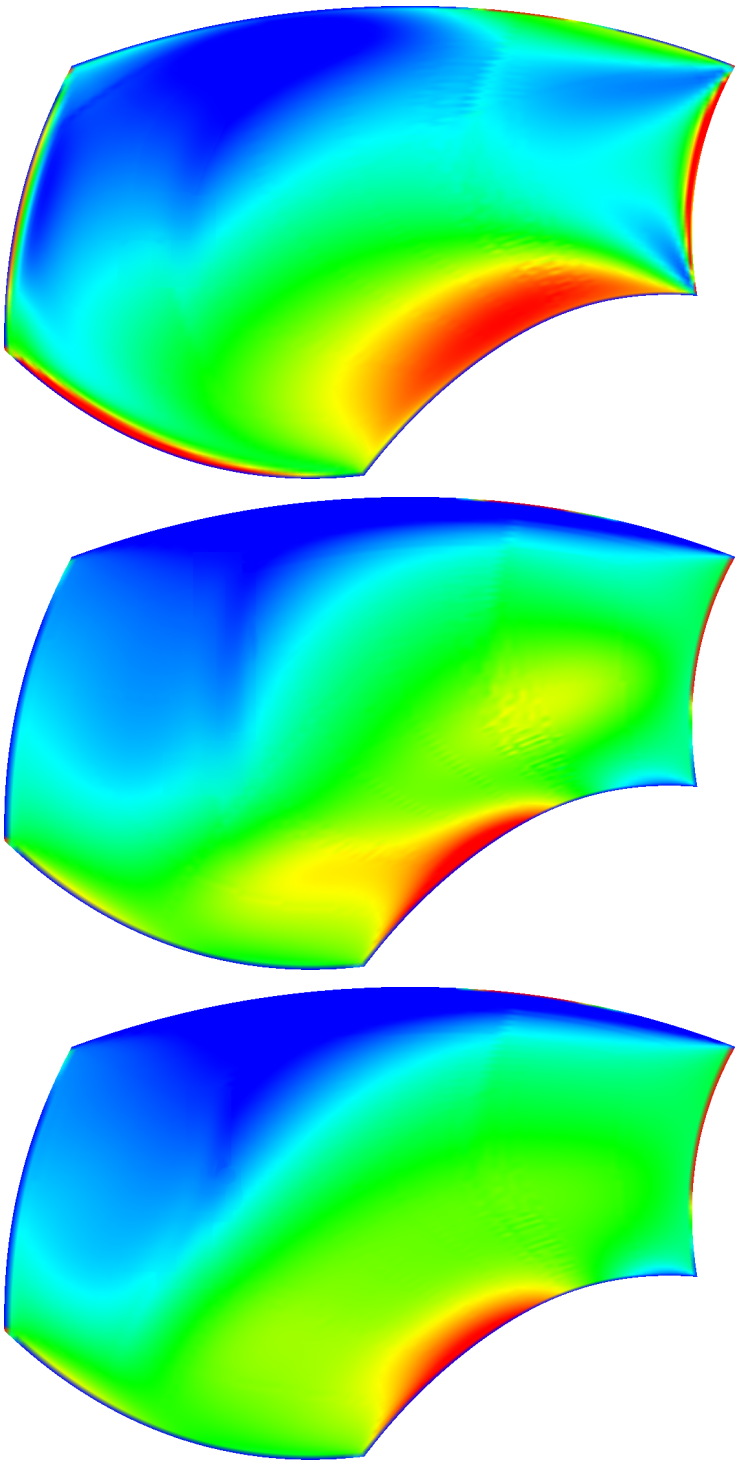


Figure 4.26: Mean map comparison of a surface using different patch types (top: LR, middle: GC, bottom: CR).

In Fig. 4.26 the three schemes defined above are contrasted using a single surface. The low-curvature areas at the right-hand corners are smoothed out in both of the new patches. The GC and CR patches are very similar, but the latter has even more smooth curvature transitions.

In order to understand the essential difference between the GC and CR patches, we should examine their ribbons. Figure 4.27 depicts them over the constant parameter lines of the patches. In the linear ribbon case, the interpolants start to deviate from the surface at a very "early phase", thus points in the interior are actually computed as the affine combination of relatively distant positions. On the other hand, the curved interpolants on the right-hand side image are going close to the predictable surface position, and in this way, points in the interior are combinations of ribbon points that are relatively close, keeping curvature variation at a low level. The slicing maps show that both surfaces have sufficiently good quality.

Finally, Fig. 4.28 shows isophote lines on two CR patches connected by a smooth edge. The ends of the lines match along the boundary, confirming G^1 continuity, and in most points they are also unbroken, exhibiting approximately G^2 behavior.

Conclusions

In this chapter traditional approaches to transfinite surface interpolation have been enhanced and generalized in several ways, as follows. New heuristic algorithms were proposed for generating convex domain polygons. These are constructed by approximately reproducing the arc lengths and the corner angles of the three-dimensional loop of boundary curves. It was shown through examples that these polygons lead to better parameterizations and thus improve surface quality. A new parameterization, called central line sweep, was also introduced. This scheme adapts the line sweep parameterization methods to work better over non-regular domains. This is achieved by constraining the middle isoline of the ribbons to go through the center of the polygon. A new n-sided generalization of the Coons patch was developed as well, based on the same idea of combining linear ribbons and corner correc-

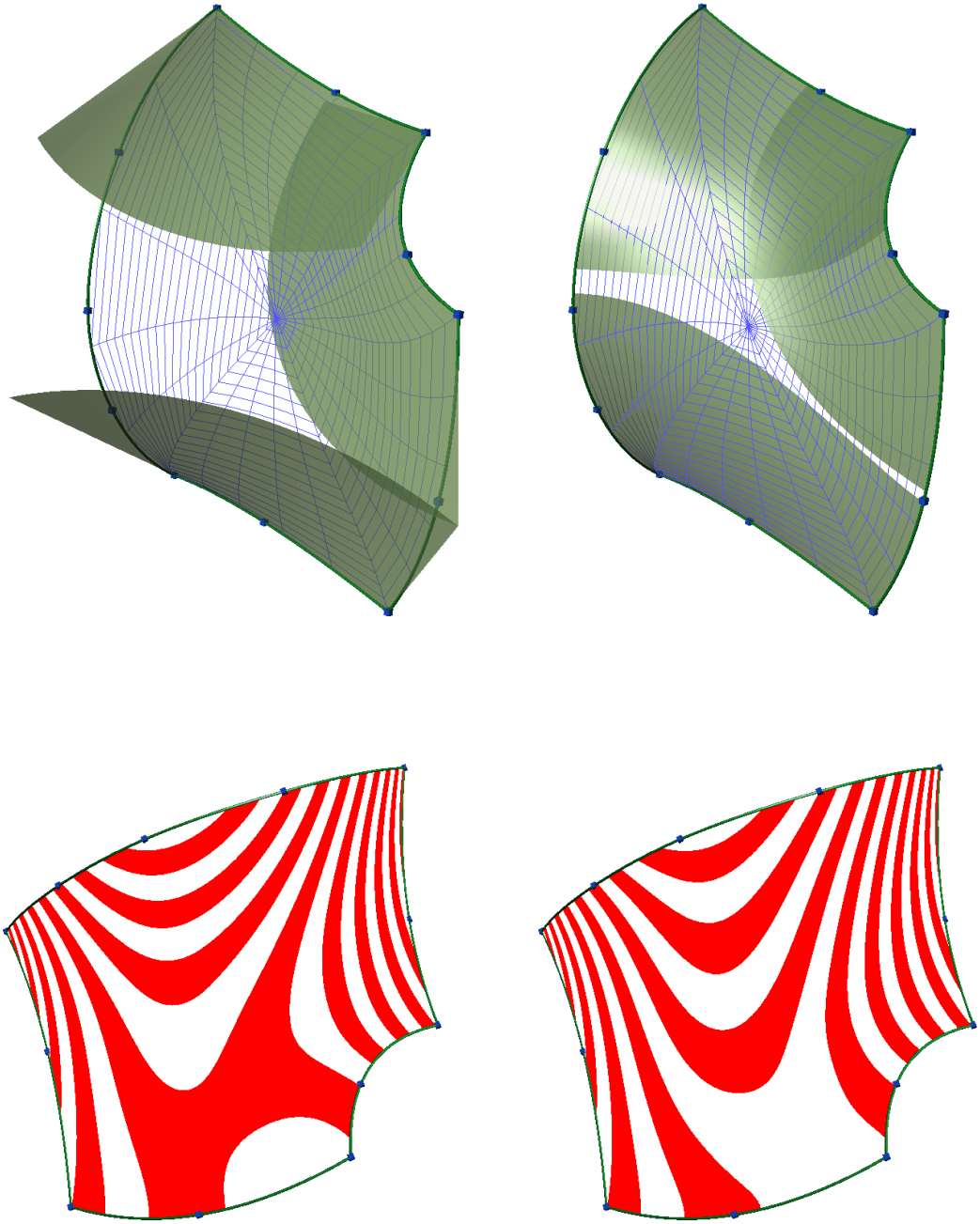


Figure 4.27: A five-sided boundary configuration using a GC patch (left) and a CR patch (right), visualized by spider lines and three ribbons (top) and slicing map (bottom).

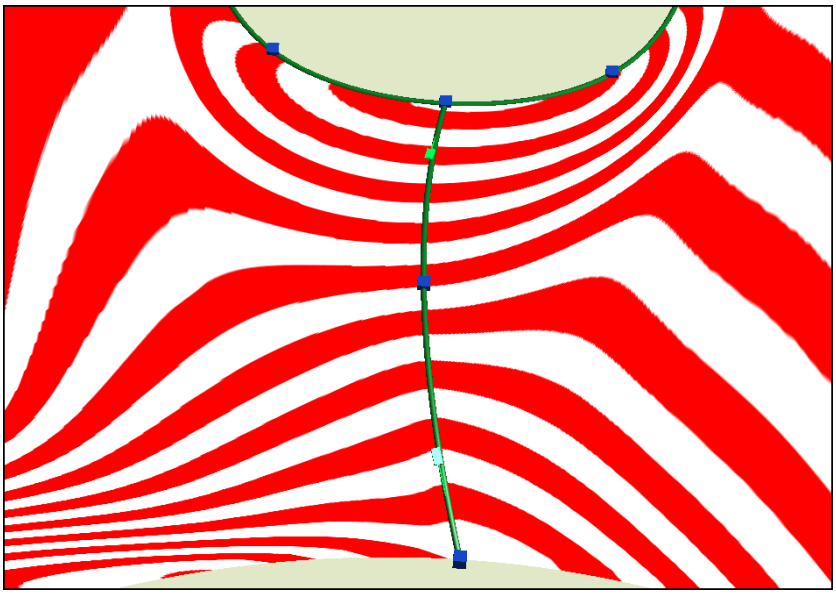


Figure 4.28: A G^1 model with two CR patches, using isophote line visualization.

tion patches. Three novel parameterization methods were also elaborated, that meet the requirements of this new patch. Furthermore, curved ribbons were defined lying on three boundary curves, ensuring better surface interior for side interpolant-based patches. Finally, another transfinite surface representation was created, based on curved ribbons. Comparisons of the two new surface patches with traditional methods showed significant improvement in surface quality.

Summary

We have examined fair curves and surfaces in two different contexts. In the Digital Shape Reconstruction area, post-processing methods were elaborated in order to enhance the quality of surfaces while deviation from the original ones was kept below a prescribed tolerance (Chapters 2 and 3). In the Computer-Aided Geometric Design area new transfinite interpolation techniques were developed in order to create multi-sided surface patches that satisfy a set of positional and tangential boundary constraints (Chapter 4).

Noisy point clouds and the resulting loss of surface fitting quality are facts of life in the CAD model reconstruction process that have to be dealt with. Fairing constitutes the final phase in the pipeline, having a direct effect on the output. Identifying the low quality areas is the first step — to this end, a new smoothness evaluation method was introduced. Efficient curve and surface fairing algorithms were proposed that minimize this measure. The smoothing effect and deviation were analyzed using visual interrogation tools, such as mean curvature maps and isophote lines.

A complex model with smooth edges cannot be faired by its components, since there are G^1 or G^2 continuity constraints between individual patches. A novel fairing mechanism was designed that goes through the patches in a hierarchical order, first smoothing primary surfaces, then edge blends, and finally coner patches, always retaining and enhancing continuity with the surfaces higher in the hierarchy. Special master–slave algorithms were developed for numerical continuity enhancement, which are glued together by a framework capable of handling not only four-sided surfaces, but also split patches with an arbitrary number of sides.

In the design context, we usually have a clear view of what kind of fea-

ture curves we need. These may come from various sources, for example 2D sketches. On the other hand, we rarely have any real information on the interior of the surfaces, which makes conventional surfacing techniques inconvenient. Transfinite interpolation seems to be the ideal solution to this problem, spreading a surface within a loop of boundary curves.

The Coons patch is a well-known example for four-sided configurations, widely used due to its natural curvature distribution. Former n-sided variants of the Coons patch, however, do not exhibit the same nice properties, partially owing to the parameterization, limited by regular domain polygons, and to the linear side interpolants. Three new algorithms were suggested for creating non-regular domain polygons, along with a new line sweep parameterization that resolves the issues of traditional methods defined for regular domains. In order to lessen the artifacts of affine combination near the center of the surface, curved side interpolants were proposed.

Two novel surface patch representations were also introduced: the first one can be regarded as the direct generalization of the Coons patch, using linear ribbons, and the second is a natural application of curved side interpolants. In addition, several new parameterization methods were developed that satisfy the more strict requirements of these surfaces.

We believe that these patches offer numerous advantages over control point-based approaches, and thus could become a viable alternative to other techniques such as T-splines [36, 35], especially in a sketch-based environment, where predefined curves and tangent planes have to be interpolated.

There are various areas for future research. These include the automatic creation of boundary ribbons based on a given 3D network, the use of non-convex polygonal domains, additional shape control of the patch interior, and surface approximation using transfinite surface patches.

Acknowledgements

I would like to express my gratitude to my supervisor, Prof. Tamás Várady, who introduced me to the world of CAGD, and without whose ideas and encouragement this work would never have been finished. I am also indebted to Prof. Hiromasa Suzuki for his invaluable support during my time at the University of Tokyo. Special thanks are due to Prof. László Szirmay-Kalos for supporting my research work at the Budapest University of Technology and Economics.

Parts of this research have been conducted within Geomagic, Inc. and ShapEx Ltd., Budapest. I highly appreciate the inspiring research environment of these teams, in particular the help of Zsolt Terék, Pál Benkő, György Karikó and Mike Facello. I am also grateful for our fruitful research cooperation with Prof. Alyn Rockwood (King Abdullah University of Science and Technology).

Appendix A

Generalized Coons Patch — Proof

We will prove that the boundary constraints for an arbitrary point on the i-th side are satisfied in both positional and tangential sense.

A.1 Positional Constraints

Take a (u, v) point on the i-th side, where $d_i = 0$. Here all blend functions vanish, except for $B_{i,i-1}$ and $B_{i+1,i}$, and thus B_{i-1}, B_i and B_{i+1} are also non-zero. Consequently,

$$\begin{aligned} S(u,v) &= (R_{i-1}(s_{i-1}, d_{i-1}) - Q_{i,i-1}(s_i, s_{i-1})) \cdot B_{i,i-1}(d_1, \ldots, d_n) + \\ &\quad R_i(s_i, d_i) + \\ &\quad (R_{i+1}(s_{i+1}, d_{i+1}) - Q_{i+1,i}(s_{i+1}, s_i)) \cdot B_{i+1,i}(d_1, \ldots, d_n). \end{aligned}$$

Since $d_i = 0$, i.e., $s_{i-1} = 1$ and $d_{i-1} = s_i$, it follows that

$$\begin{aligned} R_{i-1}(s_{i-1}, d_{i-1}) &= P_{i-1}(s_{i-1}) + d_{i-1}T_{i-1}(s_{i-1}) \\ &= P_{i-1}(1) + s_i T_{i-1}(1) \end{aligned}$$

and also

$$\begin{aligned} Q_{i,i-1}(s_i, s_{i-1}) &= P_i(0) + s_i T_{i-1}(1) + \\ &\quad (1 - s_{i-1})T_i(0) + s_i(1 - s_{i-1})W_{i,i-1} \end{aligned}$$

$$= P_i(0) + s_i T_{i-1}(1) = P_{i-1}(1) + s_i T_{i-1}(1),$$

so these two cancel out each other. Similarly, on the $(i+1)$-th side

$$\begin{aligned} R_{i+1}(s_{i+1}, d_{i+1}) &= P_{i+1}(s_{i+1}) + d_{i+1} T_{i+1}(s_{i+1}) \\ &= P_{i+1}(0) + (1 - s_i) T_{i+1}(0) \end{aligned}$$

and

$$\begin{aligned} Q_{i+1,i}(s_{i+1}, s_i) &= P_{i+1}(0) + s_{i+1} T_i(1) + \\ &\quad (1 - s_i) T_{i+1}(0) + s_{i+1}(1 - s_i) W_{i+1,i} \\ &= P_{i+1}(0) + (1 - s_i) T_{i+1}(0). \end{aligned}$$

Accordingly,

$$S(u, v) = R_i(s_i, d_i) = P_i(s_i) + d_i T_i(s_i) = P_i(s_i).$$

The patch is thus positionally correct.

A.2 Tangential Constraints

For ease of notation, let us drop the arguments of the interpolants and the blend functions, i.e., $R_k = R_k(s_k, d_k)$, $Q_{k,k-1} = Q_{k,k-1}(s_k, s_{k-1})$, $B_k = B_k(d_1, \ldots, d_n)$ and $B_{k,k-1} = B_{k,k-1}(d_1, \ldots, d_n)$. Then the derivative of S can be written as follows:

$$\begin{aligned} \frac{\partial}{\partial u} S(u, v) &= \sum_{j=1}^{n} \left[\left(\frac{\partial}{\partial s_j} R_j \frac{\partial s_j}{\partial u} + \frac{\partial}{\partial d_j} R_j \frac{\partial d_j}{\partial u} \right) \cdot B_j + \right. \\ &\qquad \left. R_j \left(\frac{\partial}{\partial d_1} B_j \frac{\partial d_1}{\partial u} + \cdots + \frac{\partial}{\partial d_n} B_j \frac{\partial d_n}{\partial u} \right) \right] \\ &- \sum_{j=1}^{n} \left[\left(\frac{\partial}{\partial s_j} Q_{j,j-1} \frac{\partial s_j}{\partial u} + \frac{\partial}{\partial s_{j-1}} Q_{j,j-1} \frac{\partial s_{j-1}}{\partial u} \right) \cdot B_{j,j-1} + \right. \\ &\qquad \left. Q_{j,j-1} \left(\frac{\partial}{\partial d_1} B_{j,j-1} \frac{\partial d_1}{\partial u} + \cdots + \frac{\partial}{\partial d_n} B_{j,j-1} \frac{\partial d_n}{\partial u} \right) \right]. \end{aligned}$$

The derivative by v is similar. Fortunately, most blend functions (and all their derivatives) vanish, when $d_i = 0$. The remaining equation is simplified to terms of the $(i-1)$-th, i-th and $(i+1)$-th side:

$$\begin{aligned}
\frac{\partial}{\partial u}S(u,v) &= \left(\frac{\partial}{\partial s_{i-1}}R_{i-1} - \frac{\partial}{\partial s_{i-1}}Q_{i,i-1}\right)\cdot\frac{\partial s_{i-1}}{\partial u}\cdot B_{i,i-1} \\
&+ \left(\frac{\partial}{\partial d_{i-1}}R_{i-1} - \frac{\partial}{\partial s_i}Q_{i,i-1}\right)\cdot\frac{\partial d_{i-1}}{\partial u}\cdot B_{i,i-1} \\
&+ \frac{\partial}{\partial s_i}R_i\cdot\frac{\partial s_i}{\partial u} + \frac{\partial}{\partial d_i}R_i\cdot\frac{\partial d_i}{\partial u} \\
&+ \left(\frac{\partial}{\partial s_{i+1}}R_{i+1} - \frac{\partial}{\partial s_{i+1}}Q_{i+1,i}\right)\cdot\frac{\partial s_{i+1}}{\partial u}\cdot B_{i+1,i} \\
&+ \left(\frac{\partial}{\partial d_{i+1}}R_{i+1} + \frac{\partial}{\partial s_i}Q_{i+1,i}\right)\cdot\frac{\partial d_{i+1}}{\partial u}\cdot B_{i+1,i}.
\end{aligned}$$

Investigate this expression part by part:

$$\begin{aligned}
\frac{\partial}{\partial s_{i-1}}R_{i-1} &= \frac{\partial}{\partial s_{i-1}}P_{i-1}(s_{i-1}) + d_{i-1}\frac{\partial}{\partial s_{i-1}}T_{i-1}(s_{i-1}) \\
&= \frac{\partial}{\partial s_{i-1}}P_{i-1}(1) + s_i\frac{\partial}{\partial s_{i-1}}T_{i-1}(1) \\
&= -T_i(0) - s_iW_{i,i-1}
\end{aligned}$$

and

$$\frac{\partial}{\partial s_{i-1}}Q_{i,i-1} = -T_i(0) - s_iW_{i,i-1},$$

the difference is 0. Similarly,

$$\begin{aligned}
\frac{\partial}{\partial s_{i+1}}R_{i+1} &= \frac{\partial}{\partial s_{i+1}}P_{i+1}(s_{i+1}) + d_{i+1}\frac{\partial}{\partial s_{i+1}}T_{i+1}(s_{i+1}) \\
&= \frac{\partial}{\partial s_{i+1}}P_{i+1}(0) + (1-s_i)\frac{\partial}{\partial s_{i+1}}T_{i+1}(0) \\
&= T_i(1) + (1-s_i)W_{i+1,i}
\end{aligned}$$

and

$$\frac{\partial}{\partial s_{i+1}}Q_{i+1,i} = T_i(1) + (1-s_i)W_{i+1,i},$$

the difference is 0 once again.

We have

$$\frac{\partial}{\partial d_{i-1}} R_{i-1} = T_{i-1}(s_{i-1}) = T_{i-1}(1)$$

and

$$\frac{\partial}{\partial s_i} Q_{i,i-1} = T_{i-1}(1) + (1 - s_{i-1}) W_{i,i-1} = T_{i-1}(1),$$

giving a result of 0, and similarly

$$\frac{\partial}{\partial d_{i+1}} R_{i+1} = T_{i+1}(s_{i+1}) = T_{i+1}(0)$$

and

$$\frac{\partial}{\partial s_i} Q_{i+1,i} = -T_{i+1}(0) - s_{i+1} W_{i+1,i} = -T_{i+1}(0),$$

that sum to 0 as well.

This leaves only

$$\begin{aligned}
\frac{\partial}{\partial u} S(u,v) &= \frac{\partial}{\partial s_i} R_i \cdot \frac{\partial s_i}{\partial u} + \frac{\partial}{\partial d_i} R_i \cdot \frac{\partial d_i}{\partial u} \\
&= \left(\frac{\partial}{\partial s_i} P_i(s_i) + \frac{\partial}{\partial s_i} T_i(s_i) d_i \right) \cdot \frac{\partial s_i}{\partial u} + T_i(s_i) \cdot \frac{\partial d_i}{\partial u} \\
&= \frac{\partial}{\partial s_i} P_i(s_i) \cdot \frac{\partial s_i}{\partial u} + T_i(s_i) \cdot \frac{\partial d_i}{\partial u}.
\end{aligned}$$

The same reasoning results in

$$\frac{\partial}{\partial v} S(u,v) = \frac{\partial}{\partial s_i} P_i(s_i) \cdot \frac{\partial s_i}{\partial v} + T_i(s_i) \cdot \frac{\partial d_i}{\partial v}.$$

A.3 Biquadratic Parameterization Case

As the formula of the patch is different, when we apply biquadratic parameterization, the proof needs to be modified accordingly. It is enough, however, to check only the parts concerning Q_i, since that is the only term that has changed.

Positional Constraints

This is trivial, since for a point lying on the i-th side, $s_i^* = s_i$ and $d_i^* = 0$. So:

$$\begin{aligned} Q_i(s_i^*, d_i^*) &= P_i(0) + s_i^* T_{i-1}(1) + d_i^* T_i(0) + s_i^* d_i^* W_{i,i-1} \\ &= P_i(0) + s_i T_{i-1}(1) = P_{i-1}(1) + s_i T_{i-1}(1), \end{aligned}$$

and also, using $s_{i+1}^* = 0$ and $d_{i+1}^* = 1 - s_i$:

$$\begin{aligned} Q_{i+1}(s_{i+1}^*, d_{i+1}^*) &= P_{i+1}(0) + s_{i+1}^* T_i(1) + \\ &\quad d_{i+1}^* T_{i+1}(0) + s_{i+1}^* d_{i+1}^* W_{i+1,i} \\ &= P_{i+1}(0) + (1 - s_i) T_{i+1}(0). \end{aligned}$$

Tangential Constraints

Let us review the whole surface, dropping the interpolant and blend arguments once again (with $Q_k = Q_k(s_k^*, d_k^*)$ instead of $Q_{k,k-1}$):

$$\begin{aligned} \frac{\partial}{\partial u} S(u,v) &= \left(\frac{\partial}{\partial s_{i-1}} R_{i-1} + \frac{\partial}{\partial d_i^*} Q_i \right) \cdot \frac{\partial s_{i-1}}{\partial u} \cdot B_{i,i-1} \\ &+ \left(\frac{\partial}{\partial d_{i-1}} R_{i-1} - \frac{\partial}{\partial s_i^*} Q_i \right) \cdot \frac{\partial d_{i-1}}{\partial u} \cdot B_{i,i-1} \\ &+ \frac{\partial}{\partial s_i} R_i \cdot \frac{\partial s_i}{\partial u} + \frac{\partial}{\partial d_i} R_i \cdot \frac{\partial d_i}{\partial u} \\ &+ \left(\frac{\partial}{\partial s_{i+1}} R_{i+1} - \frac{\partial}{\partial s_{i+1}^*} Q_{i+1} \right) \cdot \frac{\partial s_{i+1}}{\partial u} \cdot B_{i+1,i} \\ &+ \left(\frac{\partial}{\partial d_{i+1}} R_{i+1} - \frac{\partial}{\partial d_{i+1}^*} Q_{i+1} \right) \cdot \frac{\partial d_{i+1}}{\partial u} \cdot B_{i+1,i}. \end{aligned}$$

This holds, because

$$\frac{\partial d_i^*}{\partial u} = -\frac{\partial s_{i-1}}{\partial u}, \qquad \frac{\partial s_i^*}{\partial u} = \frac{\partial d_{i-1}}{\partial u},$$
$$\frac{\partial s_{i+1}^*}{\partial u} = \frac{\partial s_{i+1}}{\partial u}, \qquad \frac{\partial d_{i+1}^*}{\partial u} = \frac{\partial d_{i+1}}{\partial u}.$$

Note that the signs are flipped where we derive by d^*, which is good, since the role of the second argument in Q has also changed.

Let's examine the terms containing Q:

$$\begin{aligned}
\frac{\partial}{\partial d_i^*} Q_i &= T_i(0) + s_i^* W_{i,i-1} = T_i(0) + s_i W_{i,i-1}, \\
\frac{\partial}{\partial s_{i+1}^*} Q_{i+1} &= T_i(1) + d_{i+1}^* W_{i+1,i} = T_i(1) + (1 - s_i) W_{i+1,i}, \\
\frac{\partial}{\partial s_i^*} Q_i &= T_{i-1}(1) + d_i^* W_{i,i-1} = T_{i-1}(1), \\
\frac{\partial}{\partial d_{i+1}^*} Q_{i+1} &= T_{i+1}(0) - s_{i+1}^* W_{i+1,i} = T_{i+1}(0).
\end{aligned}$$

These all cancel out the corresponding R terms, which, repeating the same argument with v partial derivatives, ends our proof.

Appendix B

Parabolic Parameterization — Proof

We are going to prove that the parabolic parameterization proposed in Section 4.4.3.2 satisfies all the properties (4.5)–(4.10) listed in Section 4.4.1.

Requirements (4.5) and (4.7) follow from the bilinear parameterization. If (u, v) is on the base side, we have $v = 0$ and consequently $d^l = d^r = 0$, so $d_i = 0$ as well (4.6). If (u, v) is on the left side, we have $\hat{s} = 0$, so $q^v = a^v$ and thus $d^l = v/a^v$. Also, $s_i = 0$, hence $d_i = d^l$, which means that d_i is linear on the left side. The same is true for the right side, proving (4.8).

For the last two constraints, (4.9) and (4.10), we will only show $\frac{\partial d_{i+1}}{\partial u} = -\frac{\partial s_i}{\partial u}$, the other proofs are similar. Once again, if (u, v) is on the left side, we have $s_i = 0$, so

$$\frac{\partial}{\partial u} d_i = \frac{\partial}{\partial u} d^l = \frac{\partial}{\partial u} \frac{v}{q^v(\hat{s})},$$

while

$$\frac{\partial}{\partial u} s_{i-1} = \frac{\partial}{\partial u} \frac{v}{a^v + (d^v - a^v)s_i},$$

because

$$(u, v) = s_{i-1}(s_i, 0) + (1 - s_{i-1})(A + (D - A)s_i),$$

and thus

$$v = (1 - s_{i-1})(a^v + (d^v - a^v)s_i)$$

and

$$s_{i-1} = 1 - \frac{v}{a^v + (d^v - a^v)s_i}.$$

Since $\hat{s} = 0 = s_i$, we have $q^v(\hat{s}) = a^v = a^v + (d^v - a^v)s_i$. We only need to prove that their derivatives are also the same.

$$\begin{aligned} \frac{\partial}{\partial u}(a^v + (d^v - a^v)s_i) &= (d^v - a^v)\frac{\partial s_i}{\partial u}, \\ \frac{\partial}{\partial u}q^v(\hat{s}) &= 2(b^v - a^v)\frac{\partial \hat{s}}{\partial u}. \end{aligned}$$

We know that $s_i = (1 - 2\lambda)\hat{s}^2 + 2\lambda\hat{s}$ near the left side, and it follows that

$$\frac{\partial s_i}{\partial u} = (2(1 - 2\lambda)\hat{s} + 2\lambda)\,\frac{\partial \hat{s}}{\partial u} = 2\lambda\frac{\partial \hat{s}}{\partial u}.$$

Since

$$b^v - a^v = \lambda(d^v - a^v),$$

we arrive at

$$\frac{\partial}{\partial u}q^v = 2\lambda(d^v - a^v)\frac{\partial \hat{s}}{\partial u} = \frac{\partial}{\partial u}(a^v + (d^v - a^v)s_i),$$

concluding our proof.

Appendix C

Composite Ribbon Patch — Proof

In the following sections, we will examine the i-th curved ribbon, and prove that it behaves the same way (in positional and tangential sense) as R_i for points of the i-th boundary curve, which proves that the patch interpolates the prescribed boundary data.

C.1 Positional Constraints

The i-th curved ribbon has an $(s, d) = (s_i, d_i)$ local parameterization, where d is zero for points on the i-th side (4.6), giving $H(d) = 1$. Looking at the ribbons at such an $(s_0, 0)$ point, we can see that the left correction patch cancels out the contribution of the left ribbon:

$$\begin{aligned} R_i^l(s_0, 0) &= P_{i-1}(1) + s_0 T_{i-1}(1) \\ &= P_i(0) + s_0 T_{i-1}(1) = Q_i^l(s_0, 0), \end{aligned}$$

and the right correction patch cancels out the contribution of the right ribbon:

$$R_i^r(s_0, 0) = P_{i+1}(0) + (1 - s_0) T_{i+1}(0) = Q_i^r(s_0, 0).$$

This leaves only the central ribbon:

$$C_i(s_0, 0) = R_i(s_0, 0) = P_i(s_0).$$

For a point on the $(i+1)$-th side, the local parameterization has $s = 1$ (4.7). At a $(1, d_0)$ point the right correction patch cancels out the contribution of the central ribbon:

$$\begin{aligned} R_i(1, d_0) &= P_i(1) + d_0 T_i(1) \\ &= P_{i+1}(0) + d_0 T_i(1) = Q_i^r(1, d_0), \end{aligned}$$

and since $H(1) = 0$, the left ribbon and correction patch, blended by $H(s)$, also vanish, leaving only

$$C_i(1, d_0) = R_i^r(1, d_0) = P_{i+1}(d_0).$$

Finally, for a point on the $(i-1)$-th side, the local parameterization has $s = 0$ (4.7). At a $(0, d_0)$ point the left correction patch cancels out the contribution of the central ribbon:

$$R_i(0, d_0) = P_i(0) + d_0 T_i(0) = Q_i^l(0, d_0),$$

and since $H(1) = 0$, the right ribbon and correction patch, blended by $H(1-s)$, also vanish, leaving only

$$C_i(0, d_0) = R_i^l(0, d_0) = P_{i-1}(1 - d_0).$$

Accumulating the previous results and using the fact that all the blend functions vanish except for B_{i-1}, B_i and B_{i+1}, the surface equation for a point on the i-th side becomes the following:

$$\begin{aligned} S(u_0, v_0) &= \frac{1}{2} \left(P_i(d_{i-1}) B_{i-1}(d_1, \ldots, d_n) + \right. \\ &\quad P_i(s_i) B_i(d_1, \ldots, d_n) + \\ &\quad \left. P_i(1 - d_{i+1}) B_{i+1}(d_1, \ldots, d_n) \right). \end{aligned}$$

Using the parameterization requirement $d_{i-1} = s_i = 1 - d_{i+1}$ (4.8) this simplifies to

$$S(u_0, v_0) = P_i(s_i) \frac{1}{2} (B_{i-1} + B_i + B_{i+1}) = P_i(s_i).$$

C.2 Tangential Constraints

We follow the same process to prove that correct derivatives are produced. Note that

$$\frac{\partial}{\partial s}H(0) = \frac{\partial}{\partial s}H(1) = \frac{\partial}{\partial d}H(0) = \frac{\partial}{\partial d}H(1) = 0.$$

First, for a point on the i-th side, $(s_0, 0)$, the left correction patch cancels out the contribution of the left ribbon for both s- and d-derivatives:

$$\begin{aligned}
\frac{\partial}{\partial s}R_i^l(s_0,0) &= T_{i-1}(1) = \frac{\partial}{\partial s}Q_i^l(s_0,0),\\
\frac{\partial}{\partial d}R_i^l(s_0,0) &= -\frac{\partial}{\partial d}P_{i-1}(1) - s_0\frac{\partial}{\partial d}T_{i-1}(1)\\
&= T_i(0) + s_0 W_{i,i-1} = \frac{\partial}{\partial d}Q_i^l(s_0,0),
\end{aligned}$$

and the right correction patch cancels out the contribution of the right ribbon:

$$\begin{aligned}
\frac{\partial}{\partial s}R_i^r(s_0,0) &= -T_{i+1}(0) = \frac{\partial}{\partial s}Q_i^r(s_0,0),\\
\frac{\partial}{\partial d}R_i^r(s_0,0) &= \frac{\partial}{\partial d}P_{i+1}(0) + (1-s_0)\frac{\partial}{\partial d}T_{i+1}(0)\\
&= T_i(1) + (1-s_0)W_{i+1,i} = \frac{\partial}{\partial d}Q_i^r(s_0,0).
\end{aligned}$$

This leaves only the central ribbon:

$$\begin{aligned}
\frac{\partial}{\partial s}C_i(s_0,0) &= \frac{\partial}{\partial s}R_i(s_0,0) = \frac{\partial}{\partial s}P_i(s_0),\\
\frac{\partial}{\partial d}C_i(s_0,0) &= \frac{\partial}{\partial d}R_i(s_0,0) = T_i(s_0).
\end{aligned}$$

For a point on the $(i+1)$-th side, the local parameterization has $s = 1$ (4.7). At a $(1, d_0)$ point the right correction patch cancels out the contribution of the central ribbon:

$$\begin{aligned}
\frac{\partial}{\partial s}R_i(1,d_0) &= \frac{\partial}{\partial s}P_i(1) + d_0\frac{\partial}{\partial s}T_i(1)\\
&= -T_{i+1}(0) - d_0 W_{i+1,i} = \frac{\partial}{\partial s}Q_i^r(1,d_0),
\end{aligned}$$

$$\frac{\partial}{\partial d}R_i(1,d_0) = T_i(1) = \frac{\partial}{\partial d}Q_i^r(1,d_0),$$

and since $H(1) = 0$, the left ribbon and correction patch, blended by $H(s)$, also vanish, leaving only

$$\begin{aligned}\frac{\partial}{\partial s}C_i(1,d_0) &= \frac{\partial}{\partial s}R_i^r(1,d_0) = -T_{i+1}(d_0),\\ \frac{\partial}{\partial d}C_i(1,d_0) &= \frac{\partial}{\partial d}R_i^r(1,d_0) = \frac{\partial}{\partial d}P_{i+1}(d_0).\end{aligned}$$

Finally, for a point on the $(i-1)$-th side, the local parameterization has $s = 0$ (4.7). At a $(0,d_0)$ point the left correction patch cancels out the contribution of the central ribbon:

$$\begin{aligned}\frac{\partial}{\partial s}R_i(0,d_0) &= \frac{\partial}{\partial s}P_i(0) + d_0\frac{\partial}{\partial s}T_i(0)\\ &= T_{i-1}(1) + d_0 W_{i,i-1} = \frac{\partial}{\partial s}Q_i^l(0,d_0),\\ \frac{\partial}{\partial d}R_i(0,d_0) &= T_i(0) = \frac{\partial}{\partial d}Q_i^l(0,d_0),\end{aligned}$$

and the right ribbon and correction patch also vanish, leaving only

$$\begin{aligned}\frac{\partial}{\partial s}C_i(0,d_0) &= \frac{\partial}{\partial s}R_i^l(0,d_0) = T_{i-1}(1-d_0),\\ \frac{\partial}{\partial d}C_i(0,d_0) &= \frac{\partial}{\partial d}R_i^l(0,d_0) = -\frac{\partial}{\partial d}P_{i-1}(1-d_0).\end{aligned}$$

Using the fact that all the blend functions vanish except for B_{i-1}, B_i and B_{i+1} (and even these vanish in the derivatives), the surface equation for a point on the i-th side becomes the following:

$$\begin{aligned}&\frac{\partial}{\partial u}S(u_0,v_0) =\\ &\quad\frac{1}{2}\left[\left(\frac{\partial}{\partial s_{i-1}}C_{i-1}(s_{i-1},d_{i-1})\frac{\partial s_{i-1}}{\partial u} + \frac{\partial}{\partial d_{i-1}}C_{i-1}(s_{i-1},d_{i-1})\frac{\partial d_{i-1}}{\partial u}\right)B_{i-1} +\right.\\ &\qquad\left(\frac{\partial}{\partial s_i}C_i(s_i,d_i)\frac{\partial s_i}{\partial u} + \frac{\partial}{\partial d_i}C_i(s_i,d_i)\frac{\partial d_i}{\partial u}\right)B_i +\\ &\qquad\left.\left(\frac{\partial}{\partial s_{i+1}}C_{i+1}(s_{i+1},d_{i+1})\frac{\partial s_{i+1}}{\partial u} + \frac{\partial}{\partial d_{i+1}}C_{i+1}(s_{i+1},d_{i+1})\frac{\partial d_{i+1}}{\partial u}\right)B_{i+1}\right]\end{aligned}$$

Substituting the above computed values for the Coons ribbons (adjusting the indices accordingly), we get

$$\begin{aligned}
&\frac{\partial}{\partial u}S(u_0,v_0) = \\
&\quad\frac{1}{2}\left[\left(\frac{\partial}{\partial d_{i-1}}P_i(d_{i-1})\frac{\partial d_{i-1}}{\partial u}+T_i(d_{i-1})\frac{\partial s_{i-1}}{\partial u}\right)B_{i-1}+\right.\\
&\qquad\left(\frac{\partial}{\partial s_i}P_i(s_i)\frac{\partial s_i}{\partial u}+T_i(s_i)\frac{\partial d_i}{\partial u}\right)B_i+\\
&\qquad\left.\left(-\frac{\partial}{\partial d_{i+1}}P_i(1-d_{i+1})\frac{\partial d_{i+1}}{\partial u}+T_i(1-d_{i+1})\frac{\partial s_{i+1}}{\partial u}\right)B_{i+1}\right].
\end{aligned}$$

Using the parameterization requirement $d_{i-1}=s_i=1-d_{i+1}$ (4.8), the above equation becomes

$$\begin{aligned}
&\frac{\partial}{\partial u}S(u_0,v_0) = \\
&\quad\frac{1}{2}\left[\left(\frac{\partial}{\partial s_i}P_i(s_i)\frac{\partial d_{i-1}}{\partial u}+T_i(s_i)\frac{\partial s_{i-1}}{\partial u}\right)B_{i-1}+\right.\\
&\qquad\left(\frac{\partial}{\partial s_i}P_i(s_i)\frac{\partial s_i}{\partial u}+T_i(s_i)\frac{\partial d_i}{\partial u}\right)B_i+\\
&\qquad\left.\left(-\frac{\partial}{\partial s_i}P_i(s_i)\frac{\partial d_{i+1}}{\partial u}+T_i(s_i)\frac{\partial s_{i+1}}{\partial u}\right)B_{i+1}\right],
\end{aligned}$$

that is, the derivative vector will be a combination of the s_i- and d_i-derivatives of R_i ($\frac{\partial}{\partial s_i}P_i(s_i)$ and $T_i(s_i)$, respectively). This means that the derivative of S is in the tangent plane of R_i. If we also require $\frac{\partial d_{i-1}}{\partial u}=\frac{\partial s_i}{\partial u}$ and $\frac{\partial d_{i+1}}{\partial u}=-\frac{\partial s_i}{\partial u}$ (4.9 and 4.10), this simplifies to the u-derivative of R_i:

$$\begin{aligned}
\frac{\partial}{\partial u}S(u_0,v_0) &= \left(\frac{\partial}{\partial s_i}P_i(s_i)\frac{\partial s_i}{\partial u}+T_i(s_i)\frac{\partial d_i}{\partial u}\right)\frac{B_{i-1}+B_i+B_{i+1}}{2}\\
&= \frac{\partial}{\partial s_i}P_i(s_i)\frac{\partial s_i}{\partial u}+T_i(s_i)\frac{\partial d_i}{\partial u}.
\end{aligned}$$

The same reasoning works for the v-derivative.

Bibliography

[1] S.-H. Bae, R. Balakrishnan, K. Singh, Everybodylovessketch: 3d sketching for a broader audience, in: Proceedings of the 22nd annual ACM symposium on User interface software and technology, UIST '09, ACM, New York, NY, USA, 2009, pp. 59–68.

[2] E. Catmull, J. Clark, Recursively generated B-spline surfaces on arbitrary topological meshes, Computer-Aided Design 10 (6) (1978) 350–355.

[3] P. Charrot, J. A. Gregory, A pentagonal surface patch for computer aided geometric design, Computer Aided Geometric Design 1 (1) (1984) 87–94.

[4] S. A. Coons, Surfaces for computer-aided design of space forms, Tech. rep., Massachusetts Institute of Technology, Cambridge, MA, USA (1967).

[5] M. Eck, J. Hadenfeld, Local energy fairing of b-spline curves, in: H. Hagen, G. E. Farin, H. Noltemeier, R. F. Albrecht (eds.), Geometric Modelling, vol. 10 of Computing Supplement, Springer, 1995, pp. 129–147.

[6] G. Farin, Curves and surfaces for CAGD: a practical guide, 5th ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[7] G. Farin, G. Rein, N. Sapidis, A. Worsey, Fairing cubic b-spline curves, Computer Aided Geometric Design 4 (1987) 91–103.

[8] F. Glover, Tabu search, ORSA Journal on Computing 1 (3) (1989) 190–206.

[9] J. A. Gregory, Smooth interpolation without twist constraints, in: R. E. Barnhill, R. F. Riesenfeld (eds.), Computer Aided Geometric Design; Proceedings of a Conference Held at the University of Utah, Salt Lake City, Utah, March 18-21, 1974, Academic Press, Inc., Orlando, FL, USA, 1974, pp. 71–88.

[10] G. Greiner, Curvature approximation with application to surface modeling, in: J. Hoschek, P. Kaklis (eds.), Advanced Course on FAIRSHAPE, Teubner Stuttgart, 1996, pp. 241–252.

[11] J. Hadenfeld, Local energy fairing of b-spline surfaces, in: M. Daehlen, T. Lyche, L. L. Schumaker (eds.), Mathematical Methods for Curves and Surfaces, Vanderbilt University Press, 1995, pp. 203–212.

[12] S. Hahmann, Shape improvement of surfaces, Geometric Modelling 13 (1998) 135–152.

[13] S. Hahmann, S. Konz, Knot-removal surface fairing using search strategies, Computer-Aided Design 30 (2) (1998) 131–138.

[14] T. Hermann, G. Lukács, F.-E. Wolter, Geometrical criteria on the higher order smoothness of composite surfaces, Computer Aided Geometric Design 16 (9) (1999) 907–911.

[15] T. Hermann, G. Renner, Subdivision of n-sided regions into four-sided patches, in: Proceedings of the 3rd IMA Conference on the Mathematics of Surfaces, Clarendon Press, New York, NY, USA, 1989, pp. 347–357.

[16] K. L. Hsu, D. M. Tsay, Corner blending of free-form n-sided holes, IEEE Computer Graphics and Applications 18 (1998) 72–78.

[17] K. Kato, n-sided surface generation from arbitrary boundary edges, in: P.-J. Laurent, P. Sablonnière, L. L. Schumaker (eds.), Curve and Surface Design: Saint-Malo 1999, Innovations in Applied Mathematics, Vanderbilt University Press, Nashville, TN, 2000, pp. 173–181.

[18] L. P. Kobbelt, Discrete fairing and variational subdivision for freeform surface design, The Visual Computer 16 (2000) 142–158, 10.1007/s003710050204.

[19] I. Kozen, The realistic modeling technique for computer aided industrial art and design system which enable creative crafting works, or "monozukuri" (in japanese), Jidousha Gijutsu 64 (6) (2010) 18–24.

[20] J.-Y. Lai, W.-D. Ueng, G2 continuity for multiple surfaces fitting, The International Journal of Advanced Manufacturing Technology 17 (2001) 575–585, 10.1007/s001700170141.

[21] E. T. Y. Lee, Energy, fairness, and a counterexample, Computer-Aided Design 22 (1) (1990) 37–40.

[22] P. Malraison, A bibliography for n-sided surfaces, in: Mathematics of Surfaces (VIII), Information Geometers, Winchester, UK, 1998, pp. 419–430.

[23] H. Moreton, C. Séquin, Surface design with minimum energy networks, in: Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications, SMA '91, ACM, New York, NY, USA, 1991, pp. 291–301.

[24] H. P. Moreton, C. H. Sequin, Minimum variation curves and surfaces for computer-aided geometric design, in: N. S. Sapidis (ed.), Designing Fair Curves and Surfaces, SIAM, 1994, pp. 123–159.

[25] B. Numerov, Note on the numerical integration of d2x/dt2 = f(x, t), Astronomische Nachrichten 230 (19) (1927) 359–364.

[26] J. Pegna, F.-E. Wolter, Geometrical criteria to guarantee curvature continuity of blend surfaces, Journal of Mechanical Design 114 (1) (1992) 201–210.

[27] L. Piegl, W. Tiller, The NURBS book (2nd ed.), Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[28] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical recipes in C (2nd ed.): the art of scientific computing, Cambridge University Press, New York, NY, USA, 1992.

[29] J. Roulier, T. Rando, Measures of fairness for curves and surfaces, in: N. S. Sapidis (ed.), Designing Fair Curves and Surfaces, SIAM, 1994, pp. 75–122.

[30] P. Salvi, H. Suzuki, T. Várady, Fast and local fairing of b-spline curves and surfaces, in: Proceedings of the 5th international conference on Advances in geometric modeling and processing, GMP'08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 155–163.

[31] P. Salvi, T. Várady, Local fairing of freeform curves and surfaces, in: Proceedings of the Third Hungarian Conference on Computer Graphics and Geometry, 2005, pp. 113–118.

[32] P. Salvi, T. Várady, Constrained fairing of freeform surfaces, in: Proceedings of the Fifth Hungarian Conference on Computer Graphics and Geometry, 2010, pp. 65–72.

[33] P. Salvi, T. Várady, Hierarchical surface fairing with constraints, in: Proceedings of the 14th ACM Symposium on Solid and Physical Modeling, SPM '10, ACM, New York, NY, USA, 2010, pp. 195–200.

[34] P. Salvi, T. Várady, H. Suzuki, Local fairing of curves and surfaces, in: Proceedings of JSPE Semestrial Meeting, vol. 2007S, 2007, pp. 11–12.

[35] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, T. Lyche, T-spline simplification and local refinement, in: ACM SIGGRAPH 2004 Papers, SIGGRAPH '04, ACM, New York, NY, USA, 2004, pp. 276–283.

[36] T. W. Sederberg, J. Zheng, A. Bakenov, A. Nasri, T-splines and T-NURCCs, in: ACM SIGGRAPH 2003 Papers, SIGGRAPH '03, ACM, New York, NY, USA, 2003, pp. 477–484.

[37] T. Várady, Survey and new results in n-sided patch generation, in: Proceedings on Mathematics of surfaces II, Clarendon Press, New York, NY, USA, 1988, pp. 203–235.

[38] T. Várady, Overlap patches: a new scheme for interpolating curve networks with n-sided regions, Computer Aided Geometric Design 8 (1) (1991) 7–27.

[39] T. Várady, T. Hermann, Best fit surface curvature at vertices of topologically irregular curve networks, in: Proceedings of the 6th IMA Conference on the Mathematics of Surfaces, Clarendon Press, New York, NY, USA, 1996, pp. 411–427.

[40] T. Várady, R. Martin, Reverse engineering, in: Handbook of Computer Aided Geometric Design, chap. 26, Elsevier, 2002, pp. 193–229.

[41] T. Várady, A. Rockwood, Geometric construction for setback vertex blending, Computer-Aided Design 29 (6) (1997) 413–425.

[42] T. Várady, A. Rockwood, P. Salvi, Transfinite surface interpolation over irregular n-sided domains, Computer Aided Design 43 (2011) 1330–1340.

[43] T. Várady, P. Salvi, A. Rockwood, Transfinite surface interpolation with interior control, Graphical Models (under revision).

[44] V. Weiss, L. Andor, G. Renner, T. Várady, Advanced surface fitting techniques, Computer Aided Geometric Design 19 (1) (2002) 19–42.