Fast Approximation and Fair Interpolation of Scattered Data with Normalized Powell–Sabin B-splines

Jan Maes, Adhemar Bultheel, Paul Dierckx
Department of Computer Science, K.U.Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium

Abstract
In this paper we describe fast surface reconstruction algorithms based on some useful properties of Powell–Sabin B-splines. The input consists of functional or unorganized scattered data points, sampled from a surface $S$ with disk topology. The surfaces constructed are normalized Powell–Sabin B-spline surfaces. We discuss both approximation and interpolation problems. A local fairing algorithm is applied to the interpolating Powell–Sabin B-spline surfaces to create visually pleasing surfaces.

Categories and Subject Descriptors (according to ACM CCS):
G.1.2 [Approximation]: Spline and piecewise polynomial approximation G.1.1 [Interpolation]: Spline and piecewise polynomial interpolation I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Splines

1. Introduction
Scattered data interpolation refers to the following problem: given a finite set of $N$ points in $\mathbb{R}^3$, find a surface that interpolates the given set of points. There exist many interesting approaches for reconstructing smooth surfaces from discrete scattered data. In [LF99] many scattered data techniques are presented. Often the surface description is in a format not suitable for further processing in CAD systems. There are few exceptions: for instance the procedures described in [FS96] and [GH97].

In this paper we describe a fast method that approximates or interpolates scattered data by a single Powell–Sabin (PS) B-spline patch. Therefore the topology of the given data set is restricted: it must be homeomorphic to a disk. For instance complete spheres or tori cannot be reconstructed without segmenting in different patches. Such an a priori segmentation method is described in [EH96].

Fairness of the reconstructed surface is of central importance during the design process. Our interpolating algorithm produces Powell–Sabin B-spline surfaces that often are locally non-smooth. Therefore these surfaces need an after-treatment. We present a local fairing method for Powell–Sabin splines based on minimizing an energy functional. The resulting surface is still interpolating and smooth.

The paper is organized as follows. In section 2 we recall some general concepts of polynomials on triangles and we give the definition of the space of Powell–Sabin splines. This section also covers the relevant aspects of the construction of a normalized B-spline basis. Then, in section 3 and 4 we describe fast algorithms for approximating and interpolating functional scattered data. The fairness problem is dealt with in section 5. In section 6 we describe how unorganized scattered data can be approximated by parametric splines, by using the algorithms for functional scattered data. Finally, in section 7 we give some remarks on future research.

2. Powell–Sabin splines
2.1. Polynomials on triangles
Consider a non-degenerated triangle $T(V_1, V_2, V_3)$ in a plane, having vertices $V_i$ with Cartesian coordinates $(x_i, y_i)$, $i = 1, 2, 3$. The barycentric coordinates $\tau = (\tau_1, \tau_2, \tau_3)$ of an arbitrary point $(x, y) \in \mathbb{R}^2$ with respect to $T$ are defined as the unique solution of the system

\[
\begin{bmatrix}
  x_1 & x_2 & x_3 \\
  y_1 & y_2 & y_3 \\
  1 & 1 & 1 
\end{bmatrix}
\begin{bmatrix}
  \tau_1 \\
  \tau_2 \\
  \tau_3 
\end{bmatrix} =
\begin{bmatrix}
  x \\
  y \\
  1 
\end{bmatrix}.
\]

Define $\Pi_d$ as the space of bivariate polynomials of total
degree ≤ d, then each polynomial \( P_d(x, y) \in \Pi_d \) on \( T \) has a unique representation

\[
P_d(x, y) := b^d_T(\tau) = \sum_{|\lambda|=d} b_\lambda B^d_\lambda(\tau),
\]

with \( \lambda = (\lambda_1, \lambda_2, \lambda_3) \), \( \lambda_i \geq 0 \) a multi-index of length \( |\lambda| = \lambda_1 + \lambda_2 + \lambda_3 = d \), and

\[
B^d_\lambda(\tau) = \frac{d!}{\lambda_1!\lambda_2!\lambda_3!} \tau_1^{\lambda_1} \tau_2^{\lambda_2} \tau_3^{\lambda_3}
\]

the Bernstein–Bézier polynomials on the triangle [Far86].

![Figure 1: Positions of the Bézier ordinates for d = 2.](image)

The coefficients \( b_\lambda \) are called the Bézier ordinates. The points \( \left( \frac{1}{2}, \frac{1}{2}, \frac{1}{2} \right) \) are the control points for the surface \( z = B^d_T(\tau) \) and the piecewise linear interpolant to these points is the Bézier net or control net. We can display this Bernstein–Bézier representation schematically, as in figure 1. The points \( \frac{1}{2} \), marked with dots on the figure, are called the Bézier domain points.

2.2. The linear space \( S^1_1(\Delta_{PS}) \)

Consider a simply connected subset \( \Omega \subset \mathbb{R}^2 \) with polygonal boundary \( \partial \Omega \). Suppose we have a conforming triangulation \( \Delta \) of \( \Omega \), consisting of triangles \( T_j \), \( j = 1, \ldots, t \), and having vertices \( V_i \) with Cartesian coordinates \( (x_i, y_i) \), \( i = 1, \ldots, n \). The Powell–Sabin refinement \( \Delta_{PS} \) of \( \Delta \) divides each triangle \( T_j \) into six smaller triangles with a common vertex. It can be constructed as follows (see figure 2):

1. Choose an interior point \( Z_j \) for each triangle \( T_j \), so that if two triangles \( T_i \) and \( T_j \) have a common edge, the line joining \( Z_i \) and \( Z_j \) intersects this common edge at a point \( R_{ij} \) between its vertices. We will choose \( Z_j \) as the incenter of triangle \( T_j \).
2. Join the points \( Z_j \) to the vertices of \( T_j \).
3. For each edge of \( T_j \):
   - which belongs to the boundary \( \partial \Omega \), join \( Z_j \) to the middle point of the edge.
   - which is common to a triangle \( T_r \), join \( Z_j \) to \( R_{ij} \).

![Figure 2: A PS-refinement \( \Delta_{PS} \).](image)

Now we consider the space of piecewise quadratic \( C^1 \) continuous polynomials on \( \Omega \), the Powell–Sabin splines. This space is denoted by \( S^2_1(\Delta_{PS}) \). Each of the 6t triangles resulting from the PS-refinement becomes the domain triangle of a quadratic Bernstein–Bézier polynomial, i.e. we choose \( d = 2 \) in equation (1) and (2), as indicated for one subtriangle in figure 2. Powell and Sabin [PS77] showed that the following interpolation problem:

\[
s(V_k) = f_k, \quad \frac{\partial s}{\partial x}(V_k) = f_{sk}, \quad \frac{\partial s}{\partial y}(V_k) = f_{sk}, \quad k = 1, \ldots, n
\]

has a unique solution \( s(x, y) \) in \( S^2_1(\Delta_{PS}) \). Hence, the dimension of the space \( S^2_1(\Delta_{PS}) \) equals \( 3n \).

2.3. A normalized B-spline representation

Dierckx [Die97] presented a normalized B-spline representation for Powell–Sabin splines

\[
s(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{3} c_{ij} B^1_i(x, y), \quad (x, y) \in \Omega
\]

where the B-splines form a convex partition of unity on \( \Omega \), i.e.

\[
B^1_i(x, y) \geq 0 \text{ for all } x, y \in \Omega,
\]

\[
\sum_{i=1}^{n} \sum_{j=1}^{3} B^1_i(x, y) = 1 \text{ for all } x, y \in \Omega.
\]

Furthermore these basis functions have local support: \( B^1_i(x, y) \) vanishes outside the so-called molecule \( M_i \) of vertex \( V_i \), which is the union of all triangles \( T_k \) containing \( V_i \).

The basis functions \( B^1_i(x, y) \) can be obtained as follows: find three linearly independent triplets \( (\alpha_{ij}, \beta_{ij}, \gamma_{ij}) \), \( i = 1, 2, 3 \) for each vertex \( V_i \). \( B^1_i(x, y) \) is the unique solution of the interpolation problem (3) with \( (f_k, f_{sk}, f_{sk}) = (\delta_{ik} \alpha_{ij}, \delta_{ik} \beta_{ij}, \delta_{ik} \gamma_{ij}) \), where \( \delta_{ik} \) is the Kronecker delta.
The triplets \((\alpha_i, \beta_i, \gamma_i), \) \(i = 1, 2, 3\) must be determined in such a way that equations (5) and (6) are satisfied. To find appropriate triplets \((\alpha_i, \beta_i, \gamma_i), \) \(i = 1, 2, 3\) we use the algorithm from [Die97].

1. For each vertex \(V_i\), find its PS-triangle points. These are the immediately surrounding Bézier domain points of the vertex \(V_i\) and vertex \(V_i\) itself. Figure 3 shows the PS-triangle points \(L, L', L''\) and \(V_i\) for the vertex \(V_i\) in the triangle \(T(V_1, V_2, V_3)\).

2. For each vertex \(V_i\), find a triangle \(T_i(Q_1, Q_2, Q_3)\) which contains all the PS-triangle points of \(V_i\) from all the triangles \(T_i\) in the molecule \(M_i\). These triangles \(T_i, i = 1, \ldots, n\) are called PS-triangles and we denote their vertices with \(Q_j(X_j, Y_j)\). Figure 3 also shows such a PS-triangle \(T_1\).

3. Three linearly independent triplets of real numbers \((\alpha_i, \beta_i, \gamma_i), \) \(i = 1, 2, 3\) can be derived from the PS-triangle \(T_i\) of a vertex \(V_i\) as follows:

\[
\alpha_i = (\alpha_1^i, \alpha_2^i, \alpha_3^i) \text{ are the barycentric coordinates of } V_i \text{ with respect to } t_i, \\
\beta_i = (\beta_1^i, \beta_2^i, \beta_3^i) = \left(\frac{Y_3 - Y_0}{f}, \frac{Y_0 - Y_1}{f}, \frac{Y_1 - Y_2}{f}\right), \\
\gamma_i = (\gamma_1^i, \gamma_2^i, \gamma_3^i) = \left(\frac{X_3 - X_0}{f}, \frac{X_0 - X_1}{f}, \frac{X_1 - X_2}{f}\right),
\]

where

\[
f = \begin{bmatrix} X_1 & Y_1 & 1 \\ X_2 & Y_2 & 1 \\ X_3 & Y_3 & 1 \end{bmatrix}.
\]

Figure 3: PS-points and PS-triangle.

This allows to define the useful notion of control triangles. First, we define with the notation introduced above the PS-control points as

\[
C_{ij}(X_{ij}, Y_{ij}, c_{ij}).
\]  

For fixed \(i\), they constitute a triangle \(T_i(C_{i1}, C_{i2}, C_{i3})\) that is tangent to the surface at \((V_i, s(V_i))\). The projection of the control triangles \(T_i\) in the \((x, y)\) plane are the PS-triangles \(t_i\).

Finally, this normalized B-spline basis is a stable basis. In [MVDB04] it is proven that there exist constants \(K_1\) and \(K_2\) depending only on the smallest angle in \(\Delta\) such that for all choices of the coefficient vector \(c\)

\[
K_1||c||_\infty \leq \|\sum_{i=1}^n c_i B_i(x, y)\|_\infty \leq K_2||c||_\infty
\]

with \(||c||_\infty = \max_{i,j} |c_{ij}|\) and \(||f||_\infty = \max_{Q} |f(x, y)|\).

3. Fast functional data approximation

Let the given data be samples from a surface \(S\). In this section we assume that there is a plane in \(\mathbb{R}^3\) with respect to which the data can be described as functional scattered data. So, without loss of generality we assume that our data are of the form \((x_i, y_i, f(x_i, y_i))\). Functional data are substantially easier than arbitrary scattered data because one can easily build a topology for the 2-dimensional points \((x_i, y_i)\) by computing its Delaunay triangulation, for example. In the following, we define \(\pi\) as the mapping \(\pi : (x, y, z) \mapsto (x, y)\).

3.1. Finding the boundary of \(S\)

The first step is to recognize as well as possible the outer boundary of the surface \(S\) the data points are sampled from. To find a boundary it is by the previous assumption sufficient to construct a 2-dimensional boundary for the \((x_i, y_i)\) values of each point. The \(\tilde{c}_i\) can be generated afterwards.

We define the boundary as the broken line interpolant through a subset of the data set, in such a way that its projection on the \((x, y)\) plane contains all the points \((x_i, y_i, 0)\). A possible boundary is the convex hull of the \((x_i, y_i)\) points which is generated by the Delaunay triangulation. However, in general, this boundary is not the best choice as is clear from figure 4 (a). We adopt the following algorithm from [Flo96] for generating a more natural boundary.

Algorithm 3.1

1. Compute the Delaunay triangulation of the \((x_i, y_i)\) points. Its boundary is the convex hull of the data points and acts as an initial guess for the real boundary.
2. Choose a parameter \(\lambda > 0\).
3. Recursively remove any triangle which both (a) contains an edge whose length exceed \(\lambda\), and (b) contains an edge belonging to the boundary of the triangulation.
4. When no more deletions are possible the algorithm stops and the natural boundary has been generated. At the same time, all scattered data points lie on or inside the boundary and they are triangulated.

The user needs to experiment with different values of \(\lambda\) until the boundary is acceptable. An initial value for \(\lambda\) can be obtained automatically by, for instance, choosing \(\lambda\) equal to twice the average edge length. Here the tolerance \(\lambda\) is absolute, but one can easily introduce a relative tolerance by scaling \(\lambda\) by the largest edge length in the triangulation. Figure 4 (b) shows the result of the algorithm.
3.2. Fast PS-spline surface generation

Before describing the actual algorithm we begin with some notations regarding triangular meshes. A triangular mesh \( M \) is defined as a pair \((\mathcal{P}, \mathcal{K})\), where \( \mathcal{P} \) is a set of \( N \) points \( \mathcal{P} = \{ p_i \in \mathbb{R}^3 \mid 1 \leq i \leq N \} \) (either \( p_i = (x_i, y_i, f(x_i, y_i)) \) in the functional setting or \( p_i = (x_i, y_i, z_i) \) in the surface setting), and \( \mathcal{K} \) is a set of triangles \( T_i = \{ p_j, p_k, p_l \} \) with vertices in \( \mathcal{P} \).

Algorithm 3.1 returns a triangular mesh \( M \) that interpolates the given data points at the vertices of the mesh. A Powell–Sabin spline surface can be generated very fast by assigning a PS–control triangle to each triangular face of the mesh \( M \) according to the following algorithm.

**Algorithm 3.2**

1. For each triangle \( T_i \in M \), choose a point \( p_i \) interior to triangle \( T_i \).
2. Apply Algorithm 3.1 to the new points \( p_i \). We now have a new triangular mesh \( M^* \) which can be seen as the dual mesh of \( M \).
3. Define the domain triangulation \( \Delta \) as \( \pi(M^*) \), the projection of \( M^* \) onto the \((x, y)\) plane.
4. Construct the PS-refinement \( \Delta \delta \) of \( \Delta \).
5. For each vertex \( V_i \) of \( \Delta \) we have a corresponding point \( p_i \) in \( M^* \) and a corresponding triangle \( T_i \in M \). Define triangle \( T_i \) as the PS-control triangle of vertex \( V_i \in \Delta \).

The resulting surface is a Powell–Sabin spline surface, but the underlying basis functions \( B_j(x, y) \) do not necessarily satisfy equations (5) and (6), since it is possible that the PS-triangles \( \tau_i \), which are the projection of the PS-control triangles \( T_i \) onto the \((x, y)\) plane, do not contain all PS-triangle points. The algorithm is displayed schematically in figure 5.

3.3. Adapting the PS-triangles

Suppose we find a PS-triangle \( \tau_i \) with vertices \((Q_1, Q_2, Q_3)\) and a PS-triangle point \( L = \tau_1 Q_1 + \tau_2 Q_2 + \tau_3 Q_3 \) that is not contained in the triangle \( \tau_i \). Hence \( \tau_i \) is not a valid PS-triangle, and at least one barycentric coordinate \( \tau_j \) is smaller than 0. Suppose that \( \tau_1 < 0 \) which means that \( L \) lies on the wrong side of the line segment \( Q_2Q_3 \). Then we will move this line segment parallel to its original orientation until \( L \) lies on the new line segment \( \overline{Q_2Q_3} \). Practically, compute \( \overline{Q_2Q_3} = \tau_1 Q_1 + (1 - \tau_1)Q_2 \) and \( \overline{Q_3Q_1} = \tau_1 Q_1 + (1 - \tau_1)Q_3 \), then \( L \) has barycentric coordinates \((0, \tau_2/(1 - \tau_1), \tau_3/(1 - \tau_1))\) with respect to the new PS-triangle \((Q_1, \overline{Q_2Q_3}, \overline{Q_3Q_1})\). The PS-control points \( C_2 \) and \( C_3 \) were originally defined as \((Q_2, c_{22})\) and \((Q_2, c_{23})\) (7). They are redefined as \( C_2 = (Q_2, c_{22}^2) \) and \( C_3 = (Q_3, c_{23}^3) \), with \( c_{22} = \tau_1 c_{11} + (1 - \tau_1) c_{22} \) and \( c_{23} = \tau_1 c_{11} + (1 - \tau_1) c_{23} \). The cases \( \tau_2 < 0 \) and \( \tau_3 < 0 \) are treated analogously.

This procedure can be repeated until all PS-triangles \( \tau_i \) are valid. Hence we have approximated the functional data set with normalized Powell–Sabin B-splines (4). Moreover, the spline surface is tangent to the mesh \( M \) at the domain vertices \( V_i \in \Delta \).

Note that this surface generation method, like polyhedral subdivision methods, knocks off the corners of the mesh \( M \).

3.4. Domain control

In step 1 of Algorithm 3.2 we have to specify an interior point \( p_i \) for each triangle \( T_i \in M \). Common choices are the incenter or the barycenter of triangle \( T_i \). This way, \( \pi(\delta M) \) will differ a lot from \( \pi(\delta M^*) = \delta \Omega \), with \( \delta M \) and \( \delta M^* \) the boundary of resp. \( M \) and \( M^* \).

The subset of the points \( p_i \) that are interior to the boundary triangles of \( M \) will determine \( \delta \Omega \). Therefore we choose the interiors of the boundary triangles of \( M \) as the middle point of a boundary edge of these triangles, see figure 6. Then \( \delta \Omega \) will lie relatively close to \( \pi(\delta M) \), given that \( \pi(\delta M) \) does not make sharp angles.

However, there still might be great differences between \( \delta \Omega \) and \( \pi(\delta M) \) on those places where \( \pi(\delta M) \) makes a sharp angle, see figure 6. Therefore we insert the following algorithm between step 1 and step 2 of Algorithm 3.2.

**Algorithm 3.3**
Figure 6: Domain control by choosing interior points on the boundary edge. If $\pi(\delta M)$ makes a sharp angle, there are still large differences between $\pi(\delta M)$ and $\pi(\delta M^*)$.

1. Choose an angle $\theta$ between 0 and 180 degrees.
2. For each vertex $v \in \delta M$ such that $\pi(\delta M)$ has an angle smaller than $\theta$ in $v$, add vertex $v$ to the set of points $p_i$ that will define $M^*$.
3. The PS-control triangle in $v$ will be defined as the triangle formed by vertex $v$ itself and by its two neighbouring vertices in $\delta M$.

3.5. An example

Figure 7 (a) shows the functional scattered data that we want to approximate with a Powell–Sabin spline surface. It contains 6495 points. The resulting PS-spline surface is shown in figure 7 (b) and its triangulation $\Delta$ contains 12744 vertices and 25238 triangles. Only 4 vertices were created by Algorithm 3.3. Note that as a consequence the Powell–Sabin spline surface approximates the four sharp boundary corners of the data set also very well.

4. Fast functional data interpolation

The previous approximation method does interpolate the mesh $M$, but not at the given scattered data points (with some exceptions). We now describe a method for constructing a PS-spline surface (4) that interpolates the vertices of the mesh $M$, and hence interpolates the functional data set. We make the same assumptions on the data set as in section 3.

Two cases can be distinguished for functional data interpolation according to whether or not the boundary vertices have to be interpolated. We begin with the case where no interpolation of the boundary vertices is required.

4.1. Without boundary interpolation

The following algorithm interpolates all non-boundary data points.

Algorithm 4.1

1. For each triangle $T_i \in M$, add the incenter $v_i$ of triangle $T_i$ to a list.
2. For each boundary vertex $v \in \delta M$, add this vertex $v$ to the list of points $v_i$ created in step 1.
3. Apply Algorithm 3.1 to the created list of points $v_i$. We redefine $M^*$ as the newly created triangular mesh, which again can be seen as the dual mesh of $M$. See figure 8 (a).
4. For each triangle $T_i \in M^*$, choose a point $p_i$ interior to triangle $T_i$. If there exists a non-boundary data point $P_j$ such that $\pi(P_j)$ is contained in $\pi(T_i)$, then choose $p_i$ such that $\pi(p_i) = \pi(P_j)$, and mark this point. See figure 8 (b).
5. Apply the algorithm in section 3.1 to the new points $p_i$.
6. Define the domain triangulation $\Delta$ as $\pi(M^{**})$, the projection of $M^{**}$ onto the $(x, y)$ plane. If a vertex $p_i \in M^{**}$ is marked, then mark also its corresponding vertex $V_i \in \Delta$.
7. Construct the PS-refinement $\Delta_{PS}$ of $\Delta$.
8. For each vertex $V_i$ of $\Delta$ we have a corresponding point $p_i \in M^{**}$ and a corresponding triangle $T_i \in M^*$. Define
triangle \( \pi(T_i) \) as the PS-triangle of vertex \( V_i \in \Delta \). The control points \( (7) \) corresponding to vertex \( V_i \) are located in the vertices of the PS-triangle \( \pi(T_i) \) and their values are computed as linear interpolations of the mesh \( M \). See figure 8 (c).

9. For each marked vertex \( V_i \), there exists a data point \( P_j \) such that \( \pi(P_j) = V_i \). Move the control triangle \( T_i \) of vertex \( V_i \) parallel to its original orientation until the PS-spline surface interpolates data point \( P_j \).

10. Adapt the PS-triangles such that they contain all the PS-triangle points (see section 3.3).

It is important that we choose the incenter in step 1, since this choice guarantees that there is at most one data point \( P_j \) that satisfies the conditions of step 4.

For good domain/boundary approximation the same reasoning as in section 3.4 can be followed.

4.2. With boundary interpolation

First we fix an order on the boundary vertices of \( M \). A loop of boundary edges in \( M \) is created such that all boundary points are connected. We then pass through the loop and give the boundary points a cyclic order.

Only step 4 needs some changing now:

4. For each triangle \( T_i \in \mathcal{M}^* \), choose a point \( p_i \) interior to triangle \( T_i \). If there exists a non-boundary data point \( P_j \) such that \( \pi(p_i) = \pi(P_j) \), then choose \( p_i \) such that \( \pi(p_i) = \pi(P_j) \), and mark this point. If \( T_i \) is a boundary triangle with one boundary edge, then the two vertices of \( T_i \) that define its boundary edge are identical to two boundary data points \( P_j \) and \( P_{j+1} \). Choose \( p_i \) as the data point \( P_j \) with the lowest order. If \( T_i \) is a boundary triangle with two boundary edges (corner triangle), then there are three boundary data points \( P_j, P_{j+1}, P_{j+2} \) that coincide with the three vertices of \( T_i \). Again choose \( p_i \) as the data point \( P_j \) with the lowest order. Also add the vertex of \( T_i \) that coincides with the data point \( P_{j+1} \) with middle order to the list of points \( p_i \).

4.3. An example

We have interpolated the data set of figure 7. Figure 9 shows the resulting PS-spline surface. Its triangulation \( \Delta \) contains 25730 vertices and 51210 triangles.

![Figure 9: Interpolation with PS-splines](image)

The left curvature plots in figure 11 (b) and (c) show that the interpolating surface is not fair at all. Since the original data set has only 6495 data points, we have 19235 vertices for free that can be used for fairing the surface.

5. Local fairing

The previous section creates an interpolating PS-spline surface, but, as can be seen on the figures 9 and 11, the surface is non-smooth. In this section we present a local fairing algorithm for Powell–Sabin spline surfaces. Our approach is motivated by a method of Hadenfeld [Had95] about fairing tensor product B-spline surfaces. The idea of the fairing algorithm is to minimize a given functional by changing in each step only the three control points corresponding to a vertex. The energy functional that we will be using here is the potential energy of a thin elastic plate, given by

\[
E_p = \int_\Omega a(\kappa_1^2 + \kappa_2^2) + 2(1 - b)\kappa_1\kappa_2 dS,
\]

with \( \kappa_1 \) and \( \kappa_2 \) the principal curvatures of the plate and \( a, b \) constants which depend on the material. If we assume that the plate has only small deflections and if we choose \( a = 1 \) and \( b = 0 \), then we get the following approximation for the thin plate energy

\[
E(s) = \int_\Omega (D_{xx}s)^2 + 2(D_{xy}s)^2 + (D_{yy}s)^2 dx dy.  \tag{8}
\]

Here \( s \) is the Powell–Sabin surface as defined in 4. More details about this approximation can be found in [Gre94]. Powell–Sabin splines are piecewise quadratic which means that \( E(s) \) can be computed exactly.

First we introduce the following notations:
1. The PS-spline surface before a new fairing step is denoted by
\[ \tilde{s}(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{3} \tilde{c}_{ij} B'_i(x, y). \]

2. The PS-spline surface after the fairing step with a new control triangle \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\) is denoted by
\[ \tilde{s}_k(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{3} \tilde{c}_{ij} B'_i(x, y) + \sum_{k=1}^{3} \tilde{c}_k B'_k(x, y). \]

5.1. Calculating the new control triangle \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\)

The new values \(\tilde{c}_{kj}\), \(j = 1, 2, 3\) of the control points \(\tilde{C}_{kj}\), \(j = 1, 2, 3\) have to be chosen in such a way that the new surface \(\tilde{s}_k(x, y)\) minimizes the energy function \(E(x)\). Hence, we have to solve the following system of equations
\[ \frac{\partial E(\tilde{s}_k)}{\partial \tilde{c}_{kj}} = 0 \quad j = 1, 2, 3, \]
which can be rewritten in the form
\[ \begin{bmatrix} \tilde{Y}_1^1 & \tilde{Y}_1^2 & \tilde{Y}_1^3 & \tilde{c}_{k1} \\ \tilde{Y}_1^1 & \tilde{Y}_1^2 & \tilde{Y}_1^3 & \tilde{c}_{k2} \\ \tilde{Y}_1^1 & \tilde{Y}_1^2 & \tilde{Y}_1^3 & \tilde{c}_{k3} \end{bmatrix} = \begin{bmatrix} -\sum_{i\neq k} \sum_{j=1}^{3} c_i \tilde{Y}_i^j \\ -\sum_{i\neq k} \sum_{j=1}^{3} c_i \tilde{Y}_i^j \\ -\sum_{i\neq k} \sum_{j=1}^{3} c_i \tilde{Y}_i^j \end{bmatrix}. \]

with the factors \(\tilde{Y}_i^j\) equal to
\[ \int_M \left( D_{xx} B_i' D_{xx} B_i' + 2 D_{xy} B_i' D_{xy} B_i' + D_{yy} B_i' D_{yy} B_i' \right) dxdy. \]

Note that a factor \(\tilde{Y}_i^j\) is non-zero only when the intersection of the molecules \(M_i\) and \(M_k\) is not empty.

5.2. Choosing the best control triangle

In each iteration step only one control triangle is changed. Hence to get the best fairing result in this step, we look for the largest improvement of the energy integral \(E(s)\) (8).

To each control triangle \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\) that has the largest ranking number \(z_k\), we can assign a ranking number
\[ z_k = E(\tilde{s}_k) - E(\tilde{s}) \]
which describes the improvement of the energy integral \(E(s)\) if we change control triangle \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\) to \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\). Each step we will change the control triangle \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\) that has the largest ranking number \(z_k\), hence the energy integral \(E(s)\) is minimized.

A straightforward computation gives that
\[ z_k = \sum_{j=1}^{3} \sum_{i=1}^{3} \tilde{Y}_i^j (\tilde{s}_j - \tilde{c}_{kj})(\tilde{s}_j - \tilde{c}_{kj}). \]

This is a weighted function of the change of the values \(\tilde{s}_j\) of the control points \(\tilde{c}_{kj}\), \(j = 1, 2, 3\).

To change the control triangle with the largest ranking number, we have to compute the ranking numbers for all control triangles. Luckily the complete ranking list has to be computed only once. Suppose that a control triangle \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\) is set to \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\), then this change influences only the control points that are located in the molecule \(M_k\). Hence only a small fraction of the ranking list has to be recalculated for the next step.

It is possible that the surface has some inherent non-smooth properties. In such a case we do not want that these feature points are faired. The corresponding ranking numbers can be set to zero.

5.3. Interpolation constraints

To fair the seahorse of figure 9 only control triangles corresponding to free vertices, i.e. non-interpolating vertices, can be used for fairing. Otherwise we loose the interpolating property of the surface. We will now develop a local fairing step for the interpolating vertices. We want that \(\tilde{s}_k\) minimizes the energy functional \(E(x)\) subject to the constraint
\[ c_{k1} \alpha_{k1} + c_{k2} \alpha_{k2} + c_{k3} \alpha_{k3} = f_k. \]

This constraint assures that the surface \(\tilde{s}_k\) retains the value \(f_k\) at the interpolating vertex \(V_k\).

Introducing the Lagrange multiplier \(\lambda\), the following system arises:
\[ \begin{bmatrix} \tilde{Y}_1^1 & \tilde{Y}_1^2 & \tilde{Y}_1^3 & \alpha_{k1} \\ \tilde{Y}_1^1 & \tilde{Y}_1^2 & \tilde{Y}_1^3 & \alpha_{k2} \\ \tilde{Y}_1^1 & \tilde{Y}_1^2 & \tilde{Y}_1^3 & \alpha_{k3} \end{bmatrix} \begin{bmatrix} \tilde{c}_{k1} \\ \tilde{c}_{k2} \\ \tilde{c}_{k3} \\ \lambda \end{bmatrix} = \begin{bmatrix} -\sum_{i\neq k} \sum_{j=1}^{3} c_i \tilde{Y}_i^j \\ -\sum_{i\neq k} \sum_{j=1}^{3} c_i \tilde{Y}_i^j \\ -\sum_{i\neq k} \sum_{j=1}^{3} c_i \tilde{Y}_i^j \end{bmatrix}. \]

The ranking number \(z_k\) of an interpolating vertex subject to the constraint (11) is the same as for a non-interpolating vertex (10).

5.4. The fairing algorithm

We now have an automatic fairing algorithm for Powell–Sabin splines.

1. Compute the ranking numbers \(z_i\) (9) for \(i = 1 \ldots n\).
2. Find the maximum ranking number \(z_k\).
3. Replace \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\) by the new control triangle \((\tilde{C}_{k1}, \tilde{C}_{k2}, \tilde{C}_{k3})\).
4. If a stop criterion is fulfilled, stop, else recalculate the necessary ranking numbers \(z_i\) and go to step 2.

A first stop criterion is fulfilled if all ranking numbers are less than a prescribed value. Secondly, we restrict the number of iterations.
Note that our fairing algorithm iteratively solves a very large sparse system of equations that would minimize $E(s)$ with respect to all control points $C_{ij}$. However it is not important to obtain convergence. Each iteration step our surface gets smoother, and we can stop when the surface is fair enough.

We fair the surface by minimizing an energy functional. It would be possible also to fair the surface by requesting $C^5$ continuity across the edges of $\Delta_{PS}$ [Win03].

5.5. Examples

We demonstrate the smoothing effect of the algorithm with some examples that are not smooth. Figure 10 (a) shows curvature plots of a noisy Powell–Sabin spline surface with 784 vertices. Figure 10 (b) shows the PS-surface after fairing with 1000 iterations (without interpolation constraints).

![Curvature plots](image1.png)

**Figure 10:** Curvature plots [MVDB03] for demonstrating the fairing algorithm. Left is the Gaussian curvature plot, right is the mean curvature plot. We recall that the less color variation is shown in a curvature plot, the smoother the surface will be.

The seahorse of figure 9 can now be smoothed, see figure 11. We used 100000 iterations. Of course, the control points corresponding to interpolating vertices are constrained (11).

![Fair interpolating PS-surface](image2.png)

**Figure 11:** Fairing of the interpolating PS-surface. The left curvature plots [MVDB03] belong to the interpolating surface before fairing, the right to the interpolating surface after fairing. We recall that the less color variation is shown in a curvature plot, the smoother the surface will be.

6. Parametric surfaces for unorganized scattered data

The techniques discussed before can also be applied to arbitrary scattered data, given that the data is sampled from...
a surface $S$ with disk topology. Therefore we need to find a parameter domain for the scattered data points. The data set can then be approximated or interpolated by a parametric PS-spline surface. Given a triangulation $\Delta$ and accompanying Powell–Sabin refinement $\Delta_{PS}$ of the parameter domain in the $(u,v)$ plane, the space of parametric Powell–Sabin splines is defined by

$$SP^2(\Delta_{PS}) := \{ (s_1, s_2, s_3)^T | s_j(u,v) \in S_j^2(\Delta_{PS}), j = 1, 2, 3 \}.$$

Floater and Reimers [FR01] have developed a method for parameterizing and triangulating single patch unorganized point sets. The points are mapped into a planar parameter domain by solving a sparse linear system. Then a standard triangulation method is applied to the parameter points, and in this way a triangulation of the original data set is obtained. Their method assumes that the boundary points of the unorganized point set are known. Such boundary points can be detected with the method described in [MB04].

An unorganized point set, sampled from a mannequin head, is shown in figure 12 (a). Figure 12 (b) shows the result of the meshless parameterization method of [FR01]. The approximation and interpolation method can now be applied to the splines $s_1(u,v), s_3(u,v)$ and $s_3(u,v)$ separately. Figure 13 shows a parametric Powell–Sabin spline surface that approximates the given scattered data set. We can zoom in on a PS-spline surface as many times as we want. Figure 14 (a) shows the ear of the mannequin head, approximated by a PS-spline surface, and figure 14 (b) shows the Delaunay triangulation of the data points representing the ear. We have used flat shading here to accentuate the difference.

![Figure 13: PS-surface approximation after meshless parameterization](image)

![Figure 14: Zooming in on the ear](image)

7. Future work

We have described an efficient method for constructing fair PS-spline surfaces that approximate or interpolate scattered data, sampled from surfaces with disk topology. The underlying triangulation is not structured in any way. A major research challenge is to create an underlying triangulation with triadic subdivision connectivity. Triangulations with subdivision connectivity are an important class of structured triangulations, and they can be used for multiresolution techniques [EDD*95]. Powell–Sabin spline wavelets have already been developed in [VMB04]. These wavelets are constructed with the lifting scheme [Swe97], where a triadic subdivision scheme [VWBD02] is used as the first lifting step or prediction step. Applications include multiresolution editing, smoothing, data compression, image reconstruction, warping, et cetera.
Acknowledgements

This work is partially supported by the Flemish Fund for Scientific Research (FWO Vlaanderen) project MISS (G.0211.02), and by the Belgian Programme on Interuniversity Attraction Poles, initiated by the Belgian Federal Science Policy Office. The scientific responsibility rests with the authors. We wish to thank Metris for providing the seahorse data set.

References


