

Least Squares Approximation

Yong Ming Li

Intergraph

March, 1997

Contents

1	Interpolation by B-spline Curve	1
1.1	Interpolation without end conditions	1
1.1.1	Choice of \bar{t}_j	1
1.1.2	Choice of knot sequence	2
1.1.3	Solving the linear system	2
1.2	Interpolation with natural end conditions	3
2	Approximation by B-spline Curve	4
2.1	Mathematics	4
2.2	Curve approximation to ordered points	5
2.3	Curve approximation to random points	6
2.4	Penalized B-spline curve approximation	7
2.4.1	Computing \mathbf{E} (BScomptEij)	8
2.4.2	Computing $\tilde{\mathbf{R}}$	9
3	Approximation by B-spline Surface	10
3.1	Mathematics	10
3.2	Surface approximation based on grid points	10
3.3	Surface approximation based on random points	13
3.4	Penalized B-spline surface approximation	15
3.4.1	Computing \mathbf{D} , \mathbf{G} and \mathbf{F}	16
3.5	Applications	19

Chapter 1

Interpolation by B-spline Curve

1.1 Interpolation without end conditions

Given $M + 1$ points \mathbf{q}_j ($j = 0, 1, \dots, M$) with associated parameter values \bar{t}_j , we want to construct a B-spline curve of order k (or degree n) with $M + 1$ control points \mathbf{p}_i such that it interpolates the given points, i.e.,

$$\sum_{i=0}^M N_{i,k}(\bar{t}_j) \mathbf{p}_i = \mathbf{q}_j \quad (j = 0, 1, \dots, M). \quad (1.1)$$

As it is seen, the creation of interpolating B-spline curve is equivalent to solving a linear system for $M + 1$ unknowns \mathbf{p}_i . Before we can actually solve this system, however, we need to answer the following two questions:

- Given a set of points \mathbf{q}_j , how do we determine the associated parameters \bar{t}_j ?
- How do we choose the knot sequence for B-spline curve such that we can evaluate $N_{i,k}(\bar{t}_j)$?

These questions will be answered in the subsequent sections.

1.1.1 Choice of \bar{t}_j

Three methods are often used to determine the \bar{t}_j with respect to the given \mathbf{q}_j . Namely,

Method 1 : equally spaced parametrization, i.e.,

$$\bar{t}_j = \frac{j}{M} \quad (j = 0, 1, \dots, M).$$

This method is simple however may produce unwanted shapes (e.g., loops) when the data is unevenly spaced.

Method 2 : chordal parametrization. Let $d = \sum_{j=1}^M \|\mathbf{q}_j - \mathbf{q}_{j-1}\|$ and $\bar{t}_0 = 0$. Then, we have

$$\bar{t}_j = \bar{t}_{j-1} + \frac{\|\mathbf{q}_j - \mathbf{q}_{j-1}\|}{d}, \quad (j = 1, 2, \dots, M)$$

This is one of the most widely used methods since it gives an approximation to the arc length parametrization.

Method 3 : centripetal parametrization. Let $\bar{t}_0 = 0$ and $d = \sum_{j=1}^M \sqrt{\|\mathbf{q}_j - \mathbf{q}_{j-1}\|}$. Then,

$$\bar{t}_j = \bar{t}_{j-1} + \frac{\sqrt{\|\mathbf{q}_j - \mathbf{q}_{j-1}\|}}{d}, \quad (j = 1, 2, \dots, M)$$

This is a relatively new method [Lee'1989] that gives better results than the chordal parametrization when the data takes very sharp turns.

There are other parametrization methods as well [Hoschek and Lasser'1993]. With all possible parametrization methods, we prefer to using the centripetal parametrization.

1.1.2 Choice of knot sequence

We now consider the choice of knot sequence for the interpolating B-spline curve of order k (or degree n). Since the interpolating B-spline curve has $M + 1$ control points, there will be $M + 1 + k$ knots in the sequence. A simple choice of all t is again the equally spaced parametrization:

$$t_0 = t_1 = \dots = t_n = 0, \quad t_{M+1} = t_{M+2} = \dots = t_{M+k} = 1$$

$$t_{n+i} = \frac{i}{M+1-n} \quad (i = 1, 2, \dots, M-n)$$

Such simple parametrization does not take the distribution of the $\{\bar{t}_j\}$ into consideration and can easily lead to a singular system of equations. A better parametrization is recommended in [Piegl and Tiller'1997]. It derives the interior knots by averaging all \bar{t}_j as follows:

$$t_{n+i} = \frac{1}{n} \sum_{j=i}^{i+n-1} \bar{t}_j, \quad i = 1, 2, \dots, M-n. \quad (1.2)$$

The knot sequence determined above reflects the distribution of \bar{t}_j and guarantees there is at least one \bar{t}_j at each knot interval. Accordingly, the system is positive and well-conditioned.

1.1.3 Solving the linear system

Having obtained \bar{t}_j and the knot sequence, we are ready to solve the linear system to determine the control points. We write (1.1) explicitly as follows:

$$\begin{pmatrix} N_{0,k}(\bar{t}_0) & \cdots & N_{M,k}(\bar{t}_0) \\ \vdots & \vdots & \vdots \\ N_{0,k}(\bar{t}_M) & \cdots & N_{M,k}(\bar{t}_M) \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \vdots \\ \mathbf{p}_M \end{pmatrix} = \begin{pmatrix} \mathbf{q}_0 \\ \vdots \\ \mathbf{q}_M \end{pmatrix}.$$

From the local property of B-spline base functions, it is known that the above $(M + 1) \times (M + 1)$ matrix is banded with each row having at most k non-vanishing elements. With this knowledge, we can store non-vanishing elements in a one-dimensional array with proper indices to reduce the use of memory. Furthermore, by avoiding zero elements we can implement a fast algorithm to solve the system.

1.2 Interpolation with natural end conditions

It is often desired to create an interpolating B-spline curve with zero curvature at both ends (i.e., stress-free at the ends). Such ending constraints are often called the natural end conditions. A simple way to impose the natural end condition is to enforce the second derivatives at both ends are zero:

$$\mathbf{r}''(0) = \sum N''_{i,k}(0)\mathbf{p}_i = 0$$

$$\mathbf{r}''(1) = \sum N''_{i,k}(1)\mathbf{p}_i = 0$$

noting that the knot sequence is normalized to $[0, 1]$. In addition to interpolating $M + 1$ data points \mathbf{q}_j , we now have two more conditions. Therefore, we will need a B-spline curve with $M + 3$ control points to meet all $M + 3$ conditions ($M + 1$ data points plus $\mathbf{r}''(0) = 0$ and $\mathbf{r}''(1) = 0$). Accordingly, the $M + 3$ linear equations are given by

$$\begin{pmatrix} N''_{0,k}(0) & \cdots & N''_{M,k}(0) \\ N''_{0,k}(1) & \cdots & N''_{M,k}(1) \\ N_{0,k}(\bar{t}_0) & \cdots & N_{M,k}(\bar{t}_0) \\ \vdots & \vdots & \vdots \\ N_{0,k}(\bar{t}_M) & \cdots & N_{M,k}(\bar{t}_M) \end{pmatrix} \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_{M+2} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \mathbf{q}_0 \\ \vdots \\ \mathbf{q}_M \end{pmatrix}.$$

Again, the above matrix is banded and better solved as a special case to improve the performance. Since we have two more control points, we need to modify (1.2) to pick up two additional knots as follows

$$t_{n+i+1} = \frac{1}{n} \sum_{j=i}^{i+n-1} \bar{t}_j, \quad i = 0, 1, \dots, M - n + 1.$$

Chapter 2

Approximation by B-spline Curve

2.1 Mathematics

Given a set of points \mathbf{q}_l ($l = 0, 1, \dots, L + 1$) with associated parameter values \bar{t}_l , we want to construct a B-spline curve of degree n (or order k) with $M + 2$ control points \mathbf{p}_i , i.e.,

$$\mathbf{r}(t) = \sum_{i=0}^{M+1} N_{i,k}(t) \mathbf{p}_i$$

such that

- it interpolates the start and end points (i.e., \mathbf{q}_0 and \mathbf{q}_{L+1}),
- it approximates the remaining L points \mathbf{q}_l in a least squares sense, i.e.,

$$\phi(\mathbf{P}) = \sum_{l=1}^L (\mathbf{r}(\bar{t}_l) - \mathbf{q}_l)^2 \rightarrow \min,$$

where \mathbf{P} denotes a collection of $M + 2$ control points \mathbf{p}_i .

If the multiplicity of knots at both end points is equal to k , then the first and last control points coincide with the start and end points of the curve. In this case, we have $\mathbf{p}_0 = \mathbf{q}_0$ and $\mathbf{p}_{M+1} = \mathbf{q}_{L+1}$. Let

$$\mathbf{Q}_l = \mathbf{q}_l - N_{0,k}(\bar{t}_l) \mathbf{q}_0 - N_{M+1,k}(\bar{t}_l) \mathbf{q}_{L+1}.$$

We can then write $\phi(\mathbf{P})$ in a simpler form as

$$\phi(\mathbf{P}) = \sum_{l=1}^L \left(\sum_{i=1}^M N_{i,k}(\bar{t}_l) \mathbf{p}_i - \mathbf{Q}_l \right)^2. \quad (2.1)$$

Our purpose is to find all M control points such that (2.1) is minimized. Differentiating ϕ with respect to \mathbf{p}_j gives

$$\frac{\partial \phi(\mathbf{P})}{\partial \mathbf{p}_j} = 2 \sum_{l=1}^L \left(\sum_{i=1}^M N_{i,k}(\bar{t}_l) \mathbf{p}_i - \mathbf{Q}_l \right) N_{j,k}(\bar{t}_l) \quad (j = 1, 2, \dots, M).$$

Accordingly, the relation $\phi(\mathbf{P}) \rightarrow \min$ is equivalent to solving the following system of M linear equations:

$$\sum_{l=1}^L \left(\sum_{i=1}^M N_{i,k}(\bar{t}_l) \mathbf{p}_i - \mathbf{Q}_l \right) N_{1,k}(\bar{t}_l) = 0$$

$$\begin{aligned} \sum_{l=1}^L \left(\sum_{i=1}^M N_{i,k}(\bar{t}_l) \mathbf{p}_i - \mathbf{Q}_l \right) N_{2,k}(\bar{t}_l) &= 0 \\ &\vdots \\ \sum_{l=1}^L \left(\sum_{i=1}^M N_{i,k}(\bar{t}_l) \mathbf{p}_i - \mathbf{Q}_l \right) N_{M,k}(\bar{t}_l) &= 0 \end{aligned}$$

The above system of linear equations may also be written as

$$\sum_{i=1}^M \mathbf{p}_i \left(\sum_{l=1}^L N_{i,k}(\bar{t}_l) N_{j,k}(\bar{t}_l) \right) = \sum_{l=1}^L N_{j,k}(\bar{t}_l) \mathbf{Q}_l \quad (j = 1, 2, \dots, M)$$

Let \mathbf{N} be the following $L \times M$ matrix

$$\mathbf{N} = \begin{pmatrix} N_{1,k}(\bar{t}_1) & \cdots & N_{M,k}(\bar{t}_1) \\ \vdots & \vdots & \vdots \\ N_{1,k}(\bar{t}_L) & \cdots & N_{M,k}(\bar{t}_L) \end{pmatrix}.$$

Then, the system of linear equations is given by

$$(\mathbf{N}^T \mathbf{N}) \mathbf{P} = \mathbf{R} \quad (2.2)$$

where, $\mathbf{P} = (\mathbf{p}_1 \ \mathbf{p}_2 \ \cdots \ \mathbf{p}_M)^T$ and $\mathbf{R} = N^T \mathbf{Q}$, i.e.,

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}_1 \\ \vdots \\ \mathbf{R}_M \end{pmatrix} = \begin{pmatrix} N_{1,k}(\bar{t}_1) \mathbf{Q}_1 + \cdots + N_{1,k}(\bar{t}_L) \mathbf{Q}_L \\ \vdots \\ N_{M,k}(\bar{t}_1) \mathbf{Q}_1 + \cdots + N_{M,k}(\bar{t}_L) \mathbf{Q}_L \end{pmatrix} \quad (2.3)$$

As it was discussed in the previous chapter, solution of the above system requires the knowledge of \bar{t} the knot sequence. We discuss them in subsequent sections depending on the nature of data points.

2.2 Curve approximation to ordered pints

If the given points \mathbf{q}_l are ordered, we can use the centripetal parametrization to determine \bar{t} . The choice of knot sequence for the least squares approximation was suggested by de Boor [de Boor'1978]. Let

$$d = \frac{L+1}{M-n+1}.$$

Then, the internal knots are defined by

$$\begin{aligned} i &= \text{int}(jd), & \alpha &= jd - i \quad (j = 1, 2, \dots, M-n) \\ t_{n+j} &= (1-\alpha)\bar{t}_{i-1} + \alpha\bar{t}_i. \end{aligned}$$

If $d > 2.0$, then this parametrisation guarantees that every knot span contains at least one \bar{t}_i and, under this condition, the matrix $(N^T N)$ in (3) is positive definite and well-conditioned. Therefore, the system of linear equations can be solved by Gaussian elimination without pivoting.

The pseudo code is given below:

Process: BSparambsp(ndeg,npoles,npts,bar_t,knots)

Computing the knot sequence of a B-spline curve with respect to the parametrisation for data points.

Input:

ndeg: degree of B-spline curve

npoles: number of control poles of B-spline curve.

npts: number of knots \bar{t}_i .

bar_t(npts): array of data points.

Output:

knots(npoles+n+1): knot sequence of B-spline curve.

Author: Y.M. LI, May 1997.

```

do i=0, ndeg
  knots(i) = bar_t(0)
enddo

d = npts / (npoles - ndeg)
do i=1, npoles-ndeg
  j = integer (i * d)
  alpha = i * d - j
  knots(ndeg+i) = bat_t(j-1) + alpha * (bar_t(j) - bar_t(j-1))
enddo

do i=npoles, npoles+ndeg+1
  knots(i) = bar_t(npts - 1)
enddo

```

Return

2.3 Curve approximation to random points

Since \mathbf{q}_l are randomly sampled, the methods of computing associated parameters \bar{t}_l and knot sequence discussed in the previous sections cannot be applied here. If we already have a B-spline curve and want to refine it to get better approximation to the given points, then the associated parameters can be computed by calling minimum distance routine. If we do not have any B-spline curve to start with, we may compute the least squares line to approximate these discrete points. By representing the line in B-spline form and adding enough knots into the B-spline curve, we can use it as a seed curve to refine the curve such that it approximate the curve within the specified tolerance.

It should be pointed out that solutions of 2.1 or 2.2 minimizes the distance between $\mathbf{r}(\bar{t}_l)$ and \mathbf{q}_l , which does not necessarily give the shortest distance to the resulting curve. This is because all the associated parameters \bar{t}_l were obtained before refining operation is down. Accordingly, \mathbf{q}_l is not orthogonal to the resulting curve $\mathbf{r}(\bar{t}_l)$. To obtain an optimized result, we may repeat the refining

process for a few times to ensure \mathbf{q}_l is almost orthogonal to the final curve $\mathbf{r}(\bar{t}_l)$. Such process is known as the parametrization correction.

2.4 Penalized B-spline curve approximation

This section is based on Lujun Wang's work.

As it was discussed, we have different ways to determine \bar{t}_l that associated with the given points \mathbf{q}_l and the B-spline knot sequence. Careful choice of parametrization will improve the smoothness of resulting B-spline curve. However, it is not uncommon to find the resulting B-spline curve oscillating when the given data are badly distributed. Furthermore, we may encounter a situation where we do not have any point (more precisely, any \bar{t}_l) in certain knot interval. Accordingly, the linear system becomes singular and has to be solved by SVD method.

It is known that a spline curve is a mathematic representation of an elastic beam. When a curve oscillates, it indicates the elastic beam has high bending energy \mathcal{E} that is proportional to the integral of square of its curvature κ [Hoschek and Lasser'1993]. For simplicity we approximate the bending energy of curve by the following formula

$$\mathcal{E}(\mathbf{P}) = \int_a^b \left(\mathbf{r}''(t) \right)^2 dt = \int_a^b \left(\sum_{i=0}^{M+1} N''_{i,k}(t) \mathbf{p}_i \right)^2 dt.$$

By introducing $\lambda \geq 0$, we can incorporate the bending energy term into (2.1):

$$\sum_{l=1}^L \left(\sum_{i=1}^M N_{i,k}(\bar{t}_l) \mathbf{p}_i - \mathbf{Q}_l \right)^2 + \lambda \mathcal{E}(\mathbf{P}). \quad (2.4)$$

Minimization of the first term is achieved by solving (2.2). The second term is minimized if we can solve the following system of $M + 2$ linear equations for all the control points \mathbf{P} :

$$\frac{\partial E}{\partial \mathbf{p}_j} = 2 \int_a^b \left(\sum_{i=0}^{M+1} N''_{i,k}(t) \mathbf{p}_i \right) N''_{j,k}(t) dt = 0 \quad (j = 0, \dots, M + 1)$$

It is noted that \mathbf{p}_0 and \mathbf{p}_{M+1} are constrained to interpolate the start and end point (\mathbf{q}_0 and \mathbf{q}_{L+1}). For the i th linear equation, the first term

$$\mathbf{p}_0 \int_a^b N''_{i,k}(t) N''_{0,k}(t) dt$$

and the last term

$$\mathbf{p}_{M+1} \int_a^b N''_{i,k}(t) N''_{M+1,k}(t) dt$$

are known and will be combined later with the constant vector of that equation. Therefore, we need to solve only M linear equations for M control points. Accordingly, minimization of (2.4) boils down to solve (referring to (2.2)):

$$(\mathbf{N}^T \mathbf{N} + \lambda \mathbf{E}) \mathbf{P} = \tilde{\mathbf{R}}, \quad (2.5)$$

where \mathbf{E} is $M \times M$ matrix associated with minimization of bending energy. Each element of \mathbf{E} is given by

$$E_{ij} = \int_a^b N''_{i,k}(t)N''_{j,k}(t)dt.$$

$\tilde{\mathbf{R}}$ is a column vector revised from (2.3)

$$\tilde{\mathbf{R}} = \begin{pmatrix} \mathbf{R}_1 - \lambda \left(\mathbf{p}_0 \int_a^b N''_{1,k}(t)N''_{0,k}(t)dt + \mathbf{p}_{M+1} \int_a^b N''_{1,k}(t)N''_{M+1,k}(t)dt \right) \\ \vdots \\ \mathbf{R}_M - \lambda \left(\mathbf{p}_0 \int_a^b N''_{M,k}(t)N''_{0,k}(t)dt + \mathbf{p}_{M+1} \int_a^b N''_{M,k}(t)N''_{M+1,k}(t)dt \right) \end{pmatrix} \quad (2.6)$$

2.4.1 Computing \mathbf{E} (BScomptEij)

In this section we discuss how to compute $E_{ij} = \int_a^b N''_{i,k}(t)N''_{j,k}(t)dt$. Since B-spline base functions are polynomials, the integrand $N''_{i,k}(t)N''_{j,k}(t)$ is also a polynomial. For this reason, we can use Gaussian Quadrature method to compute E_{ij} precisely. There are a few things we can speed up the computation:

- It is readily seen that \mathbf{E} is a symmetric matrix since $E_{ij} = E_{ji}$. Therefore, we can save computations by half.
- B-spline curve is a piecewise polynomial. In theory, we need to loop through every distinct knot interval and apply Gaussian Quadrature in this interval to obtain local E_{ij} . The final result will be the summation of all local E_{ij} . In implementation, however, we loop through only a few relevant intervals because of the local property of B-spline functions. For any $t \in [t_l, t_{l+1})$, there are at most k non-zero B-spline functions

$$N_{l-k+1,k}(t), N_{l-k,k}(t), \dots, N_{l,k}(t).$$

Therefore, we need to take this local property of B-spline functions into consideration when we loop through the interior knots to compute piecewise E_{ij} . In essence, we set l to be the largest number of $(k, i+1, j+1)$ and start to compute the local E_{ij} at $[t_{l-1}, t_l]$. We terminate the loop when l is larger than $i+k$ or $j+k$.

As it will be seen in next chapter, we need a generic routine to compute $\int_a^b N_{i,k}^{(r)}(t)N_{j,k}^{(r)}(t)dt$. For this reason, we define BScomptEij as follows

```
void BScomptEij(
    int    ndeg,    // Input: degree of curve
    int    npoles,  // Input: number of control points
    double *knots,  // Input: B-spline knot sequence
    int    nder,    // Input: order of derivative desired
    int    ii,      // Input: index for row
    int    jj,      // Input: index for column

    double *dValue, // Output: integral
    BSrc   *rc)     // Output: error code
```

A direct method to compute E_{ij} is proposed in [Wang'2011]. When $k = 4$, she comes up a very efficient way to compute E_{ij} . In essence, she wrote $N''_{i,k}(t)$ as a combination of three lower degree polynomial

$$\begin{aligned} N''_{i,k}(t) &= \frac{(k-1)(k-2)}{(t_{i+k-1}-t_i)(t_{i+k-2}-t_i)} N_{i,k-2}(t) \\ &\quad - \left(\frac{(k-1)(k-2)}{(t_{i+k-1}-t_i)(t_{i+k-1}-t_{i+1})} + \frac{(k-1)(k-2)}{(t_{i+k}-t_{i+1})(t_{i+k-1}-t_{i+1})} \right) N_{i+1,k-2}(t) \\ &\quad + \frac{(k-1)(k-2)}{(t_{i+k}-t_{i+1})(t_{i+k}-t_{i+2})} N_{i+2,k-2}(t). \end{aligned}$$

She then compute the integral of lower degree polynomials directly.

2.4.2 Computing $\tilde{\mathbf{R}}$

Referring to (2.6), the i th entry of $\tilde{\mathbf{R}}$ is computed as follows

$$\tilde{\mathbf{R}}_i = \mathbf{R}_i - \lambda \left(\mathbf{p}_0 \int_a^b N''_{i,k}(t) N''_{0,k}(t) dt + \mathbf{p}_{M+1} \int_a^b N''_{i,k}(t) N''_{M+1,k}(t) dt \right).$$

Based on the local property of B-spline functions, it is noted that both $N''_{0,k}(t)$ and $N''_{M+1,k}(t)$ vanishes when $k \leq i \leq M - k + 1$. Therefore, we recompute only the first and last $k - 1$ terms as follows:

- $\tilde{\mathbf{R}}_i = \mathbf{R}_i - \lambda \mathbf{p}_0 \int_a^b N''_{i,k}(t) N''_{0,k}(t) dt, \quad i = 1, \dots, k - 1.$ (note $N''_{M+1,k}(t)$ vanishes when $M > 2k - 3$)
- $\tilde{\mathbf{R}}_i = \mathbf{R}_i - \lambda \mathbf{p}_{M+1} \int_a^b N''_{i,k}(t) N''_{M+1,k}(t) dt, \quad i = M - k + 2, \dots, M.$ (note $N''_{0,k}(t)$ vanishes when $M > 2k - 3$)

If $M \leq 2k - 3$, the above computation is still correct as \mathbf{R}_i will be subtracted by both terms in the loop.

Chapter 3

Approximation by B-spline Surface

3.1 Mathematics

Given a set of points \mathbf{q}_l and associated knots (u_l, v_l) with $l = 0, 1, \dots, L$, we want to construct a non-rational B-spline surface

$$\mathbf{S}(u, v) = \sum_{i=0}^N \sum_{j=0}^M N_{i,k_u}(u) N_{j,k_v}(v) \mathbf{P}_{i,j}$$

that fits \mathbf{q}_l using a least squares approximation method, i.e.,

$$\phi(\mathbf{P}) = \sum_{l=0}^L \left(\sum_{i=0}^N \sum_{j=0}^M N_{i,k_u}(u_l) N_{j,k_v}(v_l) \mathbf{P}_{i,j} - \mathbf{q}_l \right)^2 \rightarrow \min, \quad (3.1)$$

where \mathbf{P} is a collection of $(N+1) \times (M+1)$ control points $\mathbf{P}_{i,j}$. Mathematically, this is equivalent to solving the following $(N+1) \times (M+1)$ linear equations:

$$\sum_{l=0}^L \left(\sum_{i=0}^N \sum_{j=0}^M N_{i,k_u}(u_l) N_{j,k_v}(v_l) \mathbf{P}_{i,j} - \mathbf{q}_l \right) N_{r,k_u}(u_l) N_{s,k_v}(v_l) = 0 \quad (r = 0, \dots, N; s = 0, \dots, M).$$

Alternatively, we can write the above system of linear equations as follows:

$$\sum_{l=0}^L \sum_{i=0}^N \sum_{j=0}^M N_{i,k_u}(u_l) N_{j,k_v}(v_l) N_{r,k_u}(u_l) N_{s,k_v}(v_l) \mathbf{P}_{i,j} = \sum_{l=0}^L N_{r,k_u}(u_l) N_{s,k_v}(v_l) \mathbf{q}_l \quad (3.2)$$

for $r = 0, \dots, N; s = 0, \dots, M$.

As it is seen, the above system involves four matrices manipulation and multiplication. It is neither trivial nor efficient to setup the final matrix based on the above representation. In the subsequent sections we shall derive two simpler methods to solve the approximation issue, depending on how the points \mathbf{q}_l are sampled.

3.2 Surface approximation based on grid points

Given a set of grid points $\mathbf{Q}_{p,q}$ and associated knots (\bar{u}_p, \bar{v}_q) with $p = 0, 1, \dots, r$ and $q = 0, 1, \dots, s$, we want to construct a non-rational B-spline surface

$$\mathbf{S}(u, v) = \sum_{i=0}^N \sum_{j=0}^M N_{i,k_u}(u) N_{j,k_v}(v) \mathbf{P}_{i,j}$$

that fits $\mathbf{Q}_{p,q}$ using a least square surface approximation method. A classical solution to this problem is to employ the so called discrete least square method of Gauss. In this section, however, we present a much simpler least square surface approximation method that relies upon only the least square curve approximation algorithm and thus can make the best use of curve routines.

We start our discussion by considering an interpolation problem. A B-spline surface that interpolates the given set of points $\mathbf{Q}_{p,q}$ at (\bar{u}_p, \bar{v}_q) may be represented as

$$\mathbf{S}(\bar{u}, \bar{v}) = \sum_{i=0}^s \sum_{j=0}^r N_{i,k_u}(u) N_{j,k_v}(v) \mathbf{P}_{i,j}^-,$$

which satisfies the property $\mathbf{Q}_{p,q} = \bar{\mathbf{S}}(\bar{u}_p, \bar{v}_q)$. Alternatively, we may write the above B-spline surface as

$$\bar{\mathbf{S}}(u, v) = \sum_{j=0}^s N_{j,k_v}(v) \bar{\mathbf{P}}_j(u) \quad (3.3)$$

where

$$\bar{\mathbf{P}}_j(u) = \sum_{i=0}^r N_{i,k_u}(u) \bar{\mathbf{P}}_{i,j}$$

are B-spline curves that interpolate $\mathbf{Q}_{p,j}$ at $u = \bar{u}_p$. Instead of interpolating $\mathbf{Q}_{p,j}$ by $\bar{\mathbf{P}}_j(u)$, we may use a least square curve approximation method to construct

$$\mathbf{P}_j(u) = \sum_{i=0}^N N_{i,k_u}(u) \mathbf{P}_{i,j}^{\check{}}$$

that fit $\mathbf{Q}_{p,j}$ ($p = 0, 1, \dots, r$). Replacing $\bar{\mathbf{P}}_j(u)$ in 3.3 by $\mathbf{P}_j(u)$ gives

$$\bar{\mathbf{S}}(u, v) \approx \sum_{j=0}^s N_{j,k_v}(v) \mathbf{P}_j(u) = \sum_{i=0}^N N_{i,k_u}(u) \left[\sum_{j=0}^s N_{j,k_v}(v) \mathbf{P}_{i,j}^{\check{}} \right].$$

Let $\mathbf{P}_i^{\check{}}(v) = \sum_{j=0}^s N_{j,k_v}(v) \mathbf{P}_{i,j}^{\check{}}$. Then,

$$\bar{\mathbf{S}}(u, v) \approx \sum_{i=0}^N N_{i,k_u}(u) \mathbf{P}_i^{\check{}}(v). \quad (3.4)$$

Analogously, we may use a least square curve approximation method to construct

$$\mathbf{P}_i(v) = \sum_{j=0}^M N_{j,k_v}(v) \mathbf{P}_{i,j}$$

which fit $\mathbf{P}_{i,j}^{\check{}}$ ($j = 0, 1, \dots, s$). Replacing $\mathbf{P}_i^{\check{}}(v)$ in 3.4 by $\mathbf{P}_i(v)$ gives

$$\bar{\mathbf{S}}(u, v) \approx \sum_{i=0}^N N_{i,k_u}(u) \sum_{j=0}^M N_{j,k_v}(v) \mathbf{P}_{i,j} = \sum_{i=0}^N \sum_{j=0}^M N_{i,k_u}(u) N_{j,k_v}(v) \mathbf{P}_{i,j} = \mathbf{S}(u, v).$$

As it is seen, we fit across the data points first in u direction, then v direction. One may, of course, fit across the data points first in v , then u direction. In general, the resulting surfaces are not the

same. To our knowledge, there is no criterion to decide in advance which approach will yield a better solution.

Finally, we should point out that the construction of all $\mathbf{P}_j(u)$ uses the same u knot sequence. Therefore, we need to compute only once the coefficient matrices \mathbf{N} and $\mathbf{N}^T\mathbf{N}$ (see equation 2.2) for constructing $\mathbf{P}_j(u)$. Furthermore, the LU decomposition of $\mathbf{N}^T\mathbf{N}$ is only done once for the u direction fit. A similar rule applies to the v direction fit as well.

The pseudo code is listed as follows:

Process: BSlstfitptsf

Given a set of points $\mathbf{Q}_{p,q}$ and associated knots (\bar{u}_p, \bar{v}_q) , it computes a non-rational B-spline surface using a least square surface approximation method.

Input:

nptu: number of data points in u
 nptv: number of data points in v
 Qpts(3,nptu,nptv): data points, i.e., $\mathbf{Q}_{p,q}$
 bar_u(nptu): associated knots \bar{u}_p
 bar_v(nptv): associated knots \bar{v}_q
 nu: degree of B-spline surface in u
 nv: degree of B-spline surface in v
 npoleu: number of control poles of B-spline surface in u
 npolev: number of control poles of B-spline surface in v

Output:

poles(npoleu,npolev,ndim): control poles of B-spline surface
 uknots(npoleu+nu+1): the u knot sequence of B-spline surface
 vknots(npolev+nv+1): the v knot sequence of B-spline surface
 rc: return code

Author: Y.M. LI, May 1997.

Allocate spaces:

```

npts_max = max(nptu,nptv)
npole_max = max(npoleu,npolev)
ndeg_max = max(ndegu,ndegv)
ipbeg(npts_max)
ipend(npts_max)
iabeg(npole_max)
iaend(npole_max)
Nmat(npts_max,0:ndeg_max)
Amat(npole_max,npole_max)
Rpts(npole_max,ndim)
poles_temp(ndim,nptv,npoleu)

```

Computing knots sequences in both u and v :

```

BSparambsp(ndegu,npoleu,nptu,bar_u,uknots)
BSparambsp(ndegv,npolev,nptv,bar_v,vknots)

```

Fitting data points in u direction:

```

BScompmat(nptu,bar_u,ndegu,npoleu,uknots,Nmat,ipbeg,ipend,rc)
BSprodnmat(nptu,npoleu,ipbeg,ipend,Nmat,iabeg,iaend,Amat)
do j=0, nptv
  BScomprvec(nptu,ndim,Qpts(*,*,j),bar_u,ipbeg,ipend,
             Nmat,ndeg,npoles,uknots,Rvec)
  BSbndcholmt(npoleu-2,ndim,iabeg,iaend,Amat,Rvec,sol,rc)

  do nd=1, ndim
    poles_temp(nd,j,0) = Qpts(nd,0,j)
  enddo
  do i=0, npoleu-2
    poles_temp(nd,j,i+1) = sol(i,nd)
  enddo
  do nd=1, ndim
    poles_temp(nd,j,npoleu-1) = Qpts(nd,nptu-1,j)
  enddo
enddo

```

Fitting data points in v direction:

```

BScompmat(nptv,bar_v,ndegv,npolev,vknots,Nmat,ipbeg,ipend,rc)
BSprodnmat(nptv,npolev,ipbeg,ipend,Nmat,iabeg,iaend,Amat)
do i=0, npoleu
  BScomprvec(nptv,ndim,poles_temp(*,*,i),bar_v,
             ipbeg,ipend,Nmat,ndeg,npoles,vknots,Rvec)
  BSbndcholmt(npolev-2,ndim,iabeg,iaend,Amat,Rvec,sol,rc)

  do nd=1, ndim
    poles(nd,i,0) = poles_temp(nd,0,i)
  enddo
  do j=1, npolev-2
    poles(nd,i,j) = sol(j,nd)
  enddo
  do nd=1, ndim
    poles(nd,i,npolev-1) = poles_temp(nd,npolev-1,i)
  enddo
enddo

```

Return

3.3 Surface approximation based on random points

Since \mathbf{q}_l are randomly sampled, the method discussed in the previous section cannot be applied here. Instead, we have to solve the so-called least squares approximation to discrete points. To

avoid manipulating and multiplying four matrices in (3.2), we need to derive a simpler representation. For easy understanding, we start by writing a bi-quadratic B-spline surface explicitly as follows:

$$\begin{aligned} \mathbf{S}(u, v) &= [\mathbf{p}_{00}N_0(u) + \mathbf{p}_{10}N_1(u) + \mathbf{p}_{20}N_2(u)]N_0(v) \\ &+ [\mathbf{p}_{01}N_0(u) + \mathbf{p}_{11}N_1(u) + \mathbf{p}_{21}N_2(u)]N_1(v) \\ &+ [\mathbf{p}_{02}N_0(u) + \mathbf{p}_{12}N_1(u) + \mathbf{p}_{22}N_2(u)]N_2(v). \end{aligned}$$

If we store our control points in one-dimensional array as $\mathbf{p}_{00}, \mathbf{p}_{10}, \mathbf{p}_{20}, \mathbf{p}_{01}, \mathbf{p}_{11}, \dots, \mathbf{p}_{22}$ with index ranges from $k = 0$ to $k = 8$, we can then write the surface as

$$\mathbf{S}(u, v) = \sum_{k=0}^8 \mathbf{p}_k B_k(u, v),$$

where $B_{j \times 3+i} = N_i(u)N_j(v)$. It is noted that a B-spline surface is represented similarly as a B-spline curve. We now expand the bi-quadratic B-spline surface to a generic B-spline surface that has $(N + 1) \times (M + 1)$ control points. Denoting $K = (N + 1) \times (M + 1) - 1$ we have

$$\mathbf{S}(u, v) = \sum_{k=0}^K \mathbf{p}_k B_k(u, v),$$

where $B_{j \times (N+1)+i} = N_i(u)N_j(v)$. Accordingly, the least squares problem shown in (3.1) becomes

$$\phi(\mathbf{P}) = \sum_{l=0}^L \left(\sum_{k=0}^K \mathbf{p}_k B_k(u_l, v_l) - \mathbf{q}_l \right)^2 \rightarrow \min. \quad (3.5)$$

The above equation is similar to equation(2.1). Therefore, the control points of B-spline surface can be computed similarly as we did for a curve.

Let \mathbf{B} be the following $(L + 1) \times (K + 1)$ matrix

$$\mathbf{B} = \begin{pmatrix} B_0(u_0, v_0) & B_1(u_0, v_0) & \cdots & B_K(u_0, v_0) \\ B_0(u_1, v_1) & B_1(u_1, v_1) & \cdots & B_K(u_1, v_1) \\ \vdots & \vdots & \vdots & \vdots \\ B_0(u_L, v_L) & B_1(u_L, v_L) & \cdots & B_K(u_L, v_L) \end{pmatrix}.$$

Then, the $(K + 1) = (N + 1) \times (M + 1)$ control vertices \mathbf{p}_k can be determined by solving the following $(K + 1) \times (K + 1)$ linear system

$$(\mathbf{B}^T \mathbf{B}) \mathbf{P} = \mathbf{Q}. \quad (3.6)$$

We now discuss how to determine (u_l, v_l) associated with sample points \mathbf{q}_l . In some applications, we already have a B-spline surface (the seed surface) that approximate the given $(L + 1)$ sampling points but want to refine the surface to get a better approximation. In this case, the (u_l, v_l) can be obtained by calling a minimum distance routine to get the minimum distance point on the surface and associated parameters. For other applications, we may not have any seed surface. In this case, we can start with the least squares plane and represent it in a B-spline form. We then elevate the degrees of the surface in u and v to obtain some flexibility. In order to avoid oscillations commonly

seen in high degree B-spline surface, it is recommended to limit the surface to cubic. By solving (3.6), we should get a better approximation to the given points. If the accuracy does not meet the specified tolerance, we will add more knots into the u and v knots sequence to obtain more flexibility to manipulate the shape of the surface. By repeating the process, we may eventually reach the accuracy requirement. It should be pointed, however, that we do not want to repeat the process too many times for the following reasons:

- It takes significant time to compute minimum distance parameters, \mathbf{B} matrix, and large scale linear system when the surface has considerable number of control points. Assume that we are given 5,000 sampling points \mathbf{q}_l to refine a surface that has 100 control vertices (a very moderate surface). We first need to compute 5,000 (u_l, v_l) parameters based on a call to minimum distance routine. We then compute $\mathbf{B}_{5,000 \times 100}$ matrix. Next, we need to solve a linear system whose matrix is 100×100 .
- To ensure the system is positive definite, we need to have at least one sampling point at each knot interval. With the increase of control vertices, we may encounter a singular case. Although singular system can be solved using SVD method, it is not desirable in practice because of performance and oscillation considerations. Some researchers suggest to introduce energy minimization constraints into the system to ensure it is positive definite. However, it degrades the approximation accuracy in general.

It should be pointed out that solutions of 3.5 or 3.6 minimizes the distance between $\mathbf{S}(u_l, v_l)$ and \mathbf{q}_l , which does not necessarily give the shortest distance to the resulting surface. This is because all the associated parameters (u_l, v_l) were obtained before refining operation is down. Accordingly, \mathbf{q}_l is not orthogonal to the resulting surface $\mathbf{S}(u_l, v_l)$. To obtain an optimized result, we may repeat the refining process for a few times to ensure \mathbf{q}_l is almost orthogonal to the final surface $\mathbf{S}(u_l, v_l)$. Such process is known as the parametrization correction.

3.4 Penalized B-spline surface approximation

This section is again derived from Lujun Wang's work [Wang'2011].

The bending energy of a surface may be defined as:

$$\mathcal{E}(\mathbf{P}) = \int_a^b \int_c^d \left(\|\mathbf{S}_{uu}''(u, v)\|^2 + 2\|\mathbf{S}_{uv}''(u, v)\|^2 + \|\mathbf{S}_{vv}''(u, v)\|^2 \right) dudv,$$

where

$$\mathbf{S}_{uu}''(u, v) = \sum_{k=0}^K \mathbf{p}_k \frac{\partial^2 B_k(u, v)}{\partial u^2}, \quad \mathbf{S}_{uv}''(u, v) = \sum_{k=0}^K \mathbf{p}_k \frac{\partial^2 B_k(u, v)}{\partial uv}, \quad \mathbf{S}_{vv}''(u, v) = \sum_{k=0}^K \mathbf{p}_k \frac{\partial^2 B_k(u, v)}{\partial v^2}.$$

Similar to penalized B-spline curve approximation, we may introduce the above energy term to (3.5)

$$\sum_{l=0}^L \left(\sum_{k=0}^K \mathbf{p}_k B_k(u_l, v_l) - \mathbf{q}_l \right)^2 + \lambda \mathcal{E}(\mathbf{P}) \quad (3.7)$$

Minimization of the first term is achieved by solving (3.6). The second term is minimized if we can solve the following system of $(K + 1) \times (K + 1)$ linear equations

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial \mathbf{p}_j} &= \int_a^b \int_c^d \left(\sum_{k=0}^K \mathbf{p}_k \frac{\partial^2 B_k(u, v)}{\partial u^2} \right) \frac{\partial^2 B_j(u, v)}{\partial u^2} dudv \\ &+ 2 \int_a^b \int_c^d \left(\sum_{k=0}^K \mathbf{p}_k \frac{\partial^2 B_k(u, v)}{\partial uv} \right) \frac{\partial^2 B_j(u, v)}{\partial uv} dudv \\ &+ \int_a^b \int_c^d \left(\sum_{k=0}^K \mathbf{p}_k \frac{\partial^2 B_k(u, v)}{\partial v^2} \right) \frac{\partial^2 B_j(u, v)}{\partial v^2} dudv = 0, \quad (j = 0, 1, \dots, K) \end{aligned}$$

noting that we discarded the common factor 2. We also note that $(K + 1)$ is the total number of control points of a B-spline surface. Let D_{jk} , G_{jk} , and F_{jk} denote

$$\begin{aligned} D_{jk} &= \int_a^b \int_c^d \frac{\partial^2 B_k(u, v)}{\partial u^2} \frac{\partial^2 B_j(u, v)}{\partial u^2} dudv \\ G_{jk} &= \int_a^b \int_c^d \frac{\partial^2 B_k(u, v)}{\partial uv} \frac{\partial^2 B_j(u, v)}{\partial uv} dudv \\ F_{jk} &= \int_a^b \int_c^d \mathbf{p}_k \frac{\partial^2 B_k(u, v)}{\partial v^2} \frac{\partial^2 B_j(u, v)}{\partial v^2} dudv. \end{aligned}$$

By blending minimization of bending energy into (3.6) we have

$$(\mathbf{B}^T \mathbf{B} + \lambda \mathbf{E}) \mathbf{P} = \mathbf{Q},$$

where $\mathbf{E} = (\mathbf{D} + 2\mathbf{G} + \mathbf{F})$ with \mathbf{D} , \mathbf{G} and \mathbf{F} being $(K + 1) \times (K + 1)$ matrices whose elements are D_{jk} , G_{jk} , and F_{jk} respectively.

3.4.1 Computing \mathbf{D} , \mathbf{G} and \mathbf{F}

From the previous section it is known that $B_{j(N+1)+i}(u, v) = N_{i, k_u}(u) N_{j, k_v}(v)$. Differentiating it twice with respect to u gives

$$\frac{\partial^2}{\partial u^2} B_{j(N+1)+i}(u, v) = N''_{i, k_u}(u) N_{j, k_v}(v)$$

By rearranging indices ($j := j(N + 1) + i$ and $k := s(M + 1) + r$) we can compute elements in \mathbf{D} , \mathbf{G} , and \mathbf{F} as follows

$$\begin{aligned} D_{j(N+1)+i, s(M+1)+r} &= \int_a^b N''_{i, k_u}(u) N''_{r, k_u}(u) du \int_c^d N_{j, k_v}(v) N_{s, k_v}(v) dv \\ G_{j(N+1)+i, s(M+1)+r} &= \int_a^b N'_{i, k_u}(u) N'_{r, k_u}(u) du \int_c^d N'_{j, k_v}(v) N'_{s, k_v}(v) dv \\ F_{j(N+1)+i, s(M+1)+r} &= \int_a^b N_{i, k_u}(u) N_{r, k_u}(u) du \int_c^d N''_{j, k_v}(v) N''_{s, k_v}(v) dv \end{aligned}$$

In the previous section we discussed how to compute $\int_a^b N_{i, k}^{(r)}(t) N_{j, k}^{(r)}(t) dt$ and implemented `BScomptEij` to compute such integral numerically. We can call this function to compute \mathbf{D} , \mathbf{G} , and \mathbf{F} as follows

```

index = 0;
for (j=0; j<npolesV; j++)
{
  for (i=0; i<npolesU; i++)
  {
    for (s=0; s<npolesV; s++)
    {
      for (r=0; r<npolesU; r++)
      {
        BScomptEij(ndegU, npolesU, igr_sf->u_knots, 2, i, r, &temp1, &bsrc);
        BScomptEij(ndegV, npolesV, igr_sf->v_knots, 0, j, s, &temp2, &bsrc);
        Dij = temp1 * temp2;

        BScomptEij(ndegU, npolesU, igr_sf->u_knots, 1, i, r, &temp1, &bsrc);
        BScomptEij(ndegV, npolesV, igr_sf->v_knots, 1, j, s, &temp2, &bsrc);
        Gij = temp1 * temp2;

        BScomptEij(ndegU, npolesU, igr_sf->u_knots, 0, i, r, &temp1, &bsrc);
        BScomptEij(ndegV, npolesV, igr_sf->v_knots, 2, j, s, &temp2, &bsrc);
        Fij = temp1 * temp2;

        Emat[index] = Dij + 2.0 * Gij + Fij;
        index++;
      }
    }
  }
}

```

It is noted that `npolesU` and `npolesV` are numbers of control points in u and v direction respectively. The above code segment is very simple and hence easy to maintain. However, it is terribly inefficient (an $\mathcal{O}(n^4)$ method). It would take roughly 3.8 seconds (Core i7 processor) to compute the energy matrix for a B-spline surface that has $30 \times 35 = 1050$ control poles. This is because there are $1050 \times 1050 = 1,102,500$ elements in \mathbf{E} .

For a B-spline curve/surface of order k , there are at most k non-vanishing B-spline base functions at any given parameter. Taking this local property of B-spline base functions into consideration, we can improve the performance by rewriting the above code segment as follows:

```

for (i=0; i<npolesU; i++)
{
  for (r=max(0, i-ndegU); r<min(npolesU, i+ndegU+1); r++)
  {
    // Compute integrals in u
    BScomptEij(ndegU, npolesU, igr_sf->u_knots, 0, i, r, &Nir0, &bsrc);
    BScomptEij(ndegU, npolesU, igr_sf->u_knots, 1, i, r, &Nir1, &bsrc);
    BScomptEij(ndegU, npolesU, igr_sf->u_knots, 2, i, r, &Nir2, &bsrc);

    // Compute integrals in v
    for (j=0; j<npolesV; j++)

```

```

{
  m = (j * npolesU + i) * nsizeE + r;
  for (s=max(0, j-ndegV); s<min(npolesV, j+ndegV+1); s++)
  {
    BScomptEij(ndegV, npolesV, igr_sf->v_knots, 0, j, s, &Njs0, &bsrc);
    BScomptEij(ndegV, npolesV, igr_sf->v_knots, 1, j, s, &Njs1, &bsrc);
    BScomptEij(ndegV, npolesV, igr_sf->v_knots, 2, j, s, &Njs2, &bsrc);
    Dij = Nir2 * Njs0;
    Gij = Nir1 * Njs1;
    Fij = Nir0 * Njs2;
    Emat[m + s * npolesU] = Dij + 2.0 * Gij + Fij;
  }
}
}
}

```

It is noted that ndegU and ndegV are degrees of B-spline surfaces in u and v . The above method will pefrom

$$4 \times (\text{ndegU} + 1) \times (\text{ndegV} + 1) \times \text{npolesU} \times \text{npolesV}$$

loops. If ndegU and ndegV are relatively small in comparison with npolesU and npolesV , it is then an $\mathcal{O}(n^2)$ algorithm and takes 0.35 second to compute \mathbf{E} for the same surface.

Lujun Wang made further optimization by utilizing the symmetric property

$$N_{sj} = N_{js} = \int_c^d N_{j,k_v}^{(m)}(v) N_{s,k_v}^{(m)}(v) dv \quad (m = 0, 1, 2; j, s = 0, 1, \dots, \text{npolesV})$$

and saving the results in three $\text{npolesV} \times \text{npolesV}$ temporary matrices as follows:

```

// Compute Djs, Gjs, and Fjs (i.e., integrals in v):
for (j=0; j<npolesV; j++)
{
  n = j * npolesV;
  endV = min(j+ndegV+1, npolesV);
  for (s=max(0, j-ndegV); s<endV; s++)
  {
    m = s * npolesV;
    if (s < j)
    {
      // Symmetric property
      tempGmat[s + n] = tempGmat[m + j];
      tempDmat[s + n] = tempDmat[m + j];
      tempFmat[s + n] = tempFmat[m + j];
    }
    else
    {
      BScomptEij(ndegV, npolesV, igr_sf->v_knots, 0, j, s, &Njs0, &bsrc);
      BScomptEij(ndegV, npolesV, igr_sf->v_knots, 1, j, s, &Njs1, &bsrc);
      BScomptEij(ndegV, npolesV, igr_sf->v_knots, 2, j, s, &Njs2, &bsrc);
      tempGmat[s + n] = Njs1;
    }
  }
}

```

```

        tempDmat[s + n] = Njs0;
        tempFmat[s + n] = Njs2;
    }
}
}

```

It requires $(\text{ndegV} + 1) \times \text{npolesV}$ loops to compute all $\int_c^d N_{j,k_v}^{(m)}(v)N_{s,k_v}^{(m)}(v) dv$. She then compute

$$N_{ir} = N_{ri} = \int_c^d N_{i,k_u}^{(m)}(u)N_{r,k_u}^{(m)}(u) dv \quad (m = 0, 1, 2)$$

in the same fashion and combine them to complete \mathbf{E} matrix. By utilizing the symmetric property of integration and the local property of B-spline base functions, it is not difficult to prove that her approach reduced to a linear (or $\mathcal{O}(n)$) algorithm when ndegU and ndegV are relatively small. Therefore, the final version is extremely fast. It takes less than 0.004 second to compute \mathbf{E} that has 1, 102, 500 elements.

3.5 Applications

In this section we discuss how the least squares approximation method can be used to compute a single B-spline surface that approximates a plate consisting of multiple B-spline surface.

Referring to Figure 3.1, this plate has six trimmed B-spline surfaces. For certain applications users may wish to approximate these surfaces by a single B-spline surface to simplify downstream operations. Since these surfaces are usually trimmed, we may not be able to merge them. In this case, we can use the least squares approximation method to derive a single B-spline surface that approximates the given surfaces with respect to the specified tolerance.

The procedure is outlined below:

1. Sample enough points from each trimmed surfaces.
2. Compute the least squares plane and represent it in B-spline form.
3. Elevate the degree of the least squares plane surface to obtain flexibility. We recommend to elevate the plane to a bi-cubic surface.
4. Refine the surface by solving equation 3.6. If approximation accuracy is within the specified tolerance, terminate. Otherwise, go to next step.
5. Add more knots into the u and v knots sequences of the resulting surface and go to step 4.

It is noted that the plate has a horse-shoe like shape; while the resulting surface has usually rectangular shape. This means that we will have large amount of points missing at the crescent part. As a result, we may either see oscillations in the final surface or encounter a singular system. With the use of energy minimization in our least squares approximation, we can mitigate oscillations and avoid calling singular value decomposition method. The final result is shown in Figure 3.2.

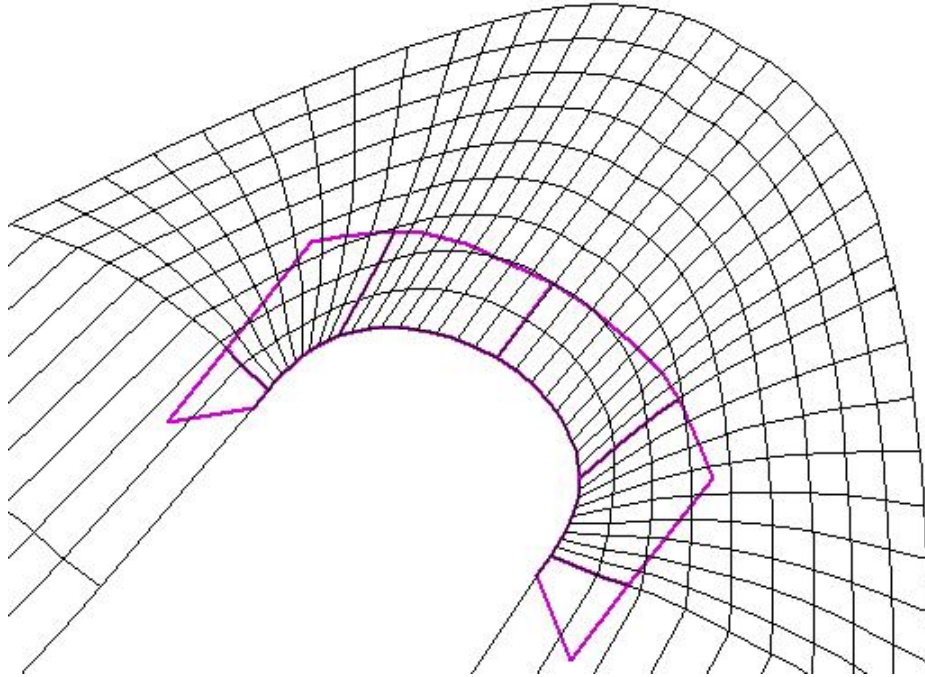


Figure 3.1: A plate consists of 6 B-spline surfaces

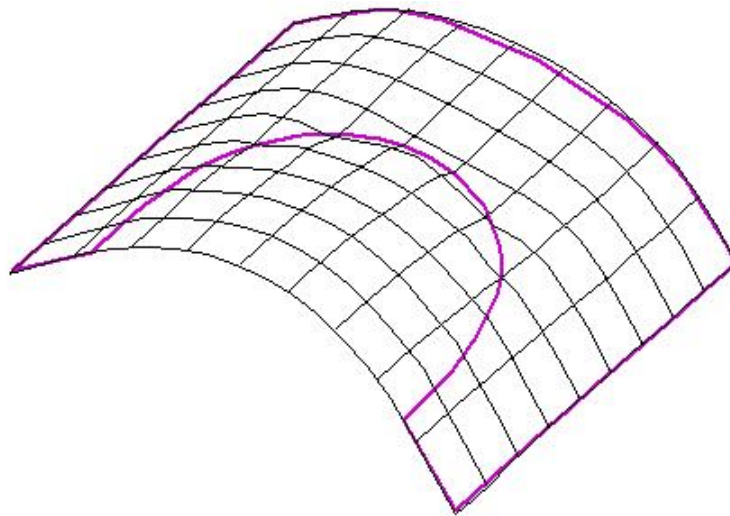


Figure 3.2: The "merged" B-spline surface

Bibliography

- [1] De Boor., C., *A Practical Guide to Splines* (New York, Springer-Verlag, 1978).
- [2] Hoschek, J. and Lasser, D., *Computer Aided Geometric Design* (A K Peter, Ltd., 1993).
- [3] Piegl, L. and Tiller, W.. *The NURBS Book, 2nd Ed.* (Springer, 1997).
- [4] Bolton, K.M. (1975) Biarc curves, *Computer Aided Design* Vol. 7, 89-92.
- [5] Meek, D.S and Walton, D.J. (1993) Approximating quadratic NURBS curves by arc splines, *Computer Aided Design* Vol. 25, 371-376.
- [6] Lujun Wang, *Penalized least squares approximation*, Intergraph design document, 2011.