Technische Universität Chemnitz
Sonderforschungsbereich 393
Numerische Simulation auf massiv parallelen Rechnern

M. Randrianarivony G. Brunnett

Parallel implementation of surface reconstruction from noisy samples

Preprint SFB393/02-16

Abstract

We consider the problem of reconstructing a surface from noisy samples by approximating the point set with non-uniform rational B-spline surfaces. We focus on the fact that the knot sequences should also be part of the unknown variables that include the control points and the weights in order to find their optimal positions. We show how to set up the free knot problem such that constrained nonlinear optimization can be applied efficiently. We describe in detail a parallel implementation of our approach that give almost linear speedup. Finally, we provide numerical results obtained on the Chemnitzer Linux Cluster supercomputer.

Keywords: rational B-splines, surface reconstruction, parallel implementation, noisy samples.

Preprint-Reihe des Chemnitzer SFB 393
ISSN 1619-7178 (Print) **ISSN 1619-7186** (Internet)

SFB393/02-16 **September 2002**

Contents

Author's addresses:

M. Randrianarivony
TU Chemnitz
Fakultät für Informatik
D-09107 Chemnitz
maharavo@informatik.tu-chemnitz.de
`http://www.tu-chemnitz.de/~ranh`

G. Brunnett
TU Chemnitz
Fakultät für Informatik
D-09107 Chemnitz
brunnett@informatik.tu-chemnitz.de
`http://www.informatik.tu-chemnitz.de/~gdv`

1 Introduction

Non-uniform rational B-spline (or NURBS) are frequently used ([11, 4, 10]) for the representation of surfaces in many applications because they allow flexible description of both free form surfaces and usual geometries such as conic sections. Indeed, the set of rational functions is much larger than that of polynomial functions, so in general NURBS provide better approximation than their polynomial B-spline counterparts do. Another reason for the appreciation of NURBS is that this representation is supported by many software packages. For instance, OpenGL and ACIS ([12], [2]) offer commands for creating and manipulating NURBS surfaces.

Our interest in the subject of approximation with NURBS is motivated by the application of reverse engineering. In reverse engineering, one is concerned with the automated generation of a computer aided design model from a set of points digitized from an existing 3D object. Since many real world objects have been constructed using both simple algebraic surfaces as well as free-form surfaces, NURBS surfaces appear to be a universal class for surface fitting in reverse engineering.

It is well known (see [7]) that the choice of the knot vector of a spline (also called parameterization) has a tremendous influence on the result of the fitting procedure. For this reason, several suggestions for a reasonable knot spacing have been made (see works of Foley, Nielson, Lee which are referenced in [7]). However, for all these suggestions of knot spacings, examples can be found where these methods provide unsatisfactory results. Furthermore, these methods can only be applied for interpolatory splines while we are interested in spline approximation. Therefore, for reliable results, one has to treat the knots as unknowns in the approximation process. Since in many applications, we have to cope with large data sets, we consider the parallel implementation of our approximation procedure.

Existing work on this problem has mainly be done on the polynomial case (see [18, 9, 17]). For the case of NURBS, the following approaches have been taken: in [4], the author uses an iterative segment determination in order to establish the positions of the knots. The authors of [11] use an improved version of Polak-Ribiere algorithm in order to minimize some cost functional without trying to reduce the number of parameters.

To our knowledge, no papers on the parallel implementation of this problem have been published. The context of the paper is organized as follows. In the next section we will state the problem and introduce the various notations. We will also discuss how to reduce the problem of surface fitting to be similar to the one in curve fitting by means of lexicographic ordering. In section 2, we show how to solve the free knot problem which is the main goal of this paper. The parallel realization will be described in section 3 where we show how to achieve load balancing. Parallel performance analysis including speedup and efficiency investigation will be depicted in section 4.

2 Problem setting and notations

2.1 NURBS surface

A nonuniform rational B-spline (NURBS) surface with weights $w_{i,j} \in \mathbf{R}_+$ and control points $\mathbf{d}_{i,j} \in \mathbf{R}^3$ $i = 0, \cdots, n_u$ $j = 0, \cdots, n_v$ is given by:

$$\mathbf{X}(u,v) := \frac{\sum_{i=0}^{n_u} \sum_{j=0}^{n_v} w_{i,j} \mathbf{d}_{i,j} N_i^{k_u}(u) N_j^{k_v}(v)}{\sum_{i=0}^{n_u} \sum_{j=0}^{n_v} w_{i,j} N_i^{k_u}(u) N_j^{k_v}(v)}\,, \tag{1}$$

where $N_i^{k_u}$ and $N_j^{k_v}$ are the usual B-spline basis functions ([5]) defined respectively on the knot sequences:

$$\mu_0 = \cdots = \mu_{k_u-1}, \mu_{k_u}, \cdots, \mu_{n_u}, \mu_{n_u+1} = \cdots = \mu_{n_u+k_u}\,, \tag{2}$$

and

$$\nu_0 = \cdots = \nu_{k_v-1}, \nu_{k_v}, \cdots, \nu_{n_v}, \nu_{n_v+1} = \cdots = \nu_{n_v+k_v} \tag{3}$$

respectively.

2.2 Lexicographic ordering

Let us first introduce the following lexicographic ordering of the surface information:

$$\begin{aligned} \tilde{w}_{i(n_v+1)+j} &:= w_{i,j} & (4)\\ \tilde{\mathbf{d}}_{i(n_v+1)+j} &:= \mathbf{d}_{i,j} & (5)\\ \tilde{N}_{i(n_v+1)+j}(u,v) &:= N_i^{k_u}(u) N_j^{k_v}(v)\,. & (6) \end{aligned}$$

Note that the new expressions $\tilde{w}_s$, $\tilde{\mathbf{d}}_s$, $\tilde{N}_s(u,v)$ have only one index $s = 0, \cdots, n$ where $n := n_u(n_v+1) + n_v$, whereas the old ones $w_{i,j}$, $\mathbf{d}_{i,j}$, $N_i^{k_u}(u)N_j^{k_v}(v)$ have two indices $i = 0, \cdots, n_u$ $j = 0, \cdots, n_v$.
With the help of these new notations, the definition (1) becomes as simple as:

$$\mathbf{X}(u,v) = \frac{\sum_{s=0}^{n} \tilde{w}_s \tilde{\mathbf{d}}_s \tilde{N}_s(u,v)}{\sum_{s=0}^{n} \tilde{w}_s \tilde{N}_s(u,v)}\,, \tag{7}$$

We will assume that

$$\mu_0 = \nu_0 = 0\,, \quad \text{and} \quad \mu_{n_u+k_u} = \nu_{n_v+k_v} = 1\,.$$

In the sequel, we will denote:

$$\begin{aligned} \mathbf{W} &:= (\tilde{w}_0, ..., \tilde{w}_n) & (8)\\ \mathbf{D} &:= (\tilde{\mathbf{d}}_0, ..., \tilde{\mathbf{d}}_n) & (9)\\ \mathbf{T} &:= (\mu_{k_u}, ..., \mu_{n_u}, \nu_{k_v}, ..., \nu_{n_v}). & (10) \end{aligned}$$

2.3 Free knot problem

Suppose we are given a sequence of noisy samples $((u_i, v_i), \mathbf{M}_i)$ $i = 0, ..., m$ with $k_u \ll m$ and $k_v \ll m$. We want to find the NURBS surface $\mathbf{X}(u, v) = \mathbf{X}_{\mathbf{W},\mathbf{D},\mathbf{T}}(u, v)$ which fits these data best in a least square sense. Since we want to find the optimal positions of the knots, we put them as variables. That means, we have the following problem:

$$\min_{\mathbf{W},\mathbf{D},\mathbf{T}} \sum_{i=0}^{m} \|\mathbf{X}_{\mathbf{W},\mathbf{D},\mathbf{T}}(u_i, v_i) - \mathbf{M}_i\|^2 . \tag{11}$$

This problem is too difficult to solve because of the nonlinear dependence of $\mathbf{X}$ on the parameters $(\mathbf{W}, \mathbf{D}, \mathbf{T})$. Furthermore, we need to add some constraints that guarantee the positivity of the weights. In the next section, we will show how to simplify this problem.

2.4 The fixed knot problem

If we are given a knot sequence $\mathbf{T}$, then the problem

$$\min_{\mathbf{W},\mathbf{D}} \sum_{i=0}^{m} \|\mathbf{X}_{\mathbf{W},\mathbf{D},\mathbf{T}}(u_i, v_i) - \mathbf{M}_i\|^2$$

will be refered to as fixed knot problem. It has been investigated for example in [4] where it has been shown to be equivalent to the linear system:

$$(A + \lambda B)\mathbf{y} = \lambda \mathbf{r} , \tag{12}$$

where A and B are given in block structure:

$$A := \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad B := \begin{bmatrix} 0 & 0 \\ 0 & \tilde{B} \end{bmatrix}$$

$$A_{rs} := \begin{bmatrix} \sum_{i=0}^{m} \overline{N}^i_{00} \mathbf{Z}_i^{rs} & \cdots & \sum_{i=0}^{m} \overline{N}^i_{0n} \mathbf{Z}_i^{rs} \\ \vdots & & \vdots \\ \sum_{i=0}^{m} \overline{N}^i_{n0} \mathbf{Z}_i^{rs} & \cdots & \sum_{i=0}^{m} \overline{N}^i_{nn} \mathbf{Z}_i^{rs} \end{bmatrix} \tag{13}$$

$$\mathbf{Z}_i^{(1,1)} := \mathbf{I} - c_i \mathbf{M}_i \mathbf{M}_i^T \tag{14}$$

$$\mathbf{Z}_i^{(1,2)} := c_i \mathbf{M}_i \tag{15}$$

$$\mathbf{Z}_i^{(2,1)} := c_i \mathbf{M}_i^T \tag{16}$$

$$\mathbf{Z}_i^{(2,2)} := 1 - c_i \tag{17}$$

$$\tilde{B} := \begin{bmatrix} \sum_{i=0}^{m} \overline{N}^i_{00} & \cdots & \sum_{i=0}^{m} \overline{N}^i_{0n} \\ \vdots & & \vdots \\ \sum_{i=0}^{m} \overline{N}^i_{n0} & \cdots & \sum_{i=0}^{m} \overline{N}^i_{nn} \end{bmatrix} \tag{18}$$

$$
\begin{aligned}
\mathbf{y} &:= [\bar{\mathbf{d}}_0, ..., \bar{\mathbf{d}}_n, \tilde{w}_0, ..., \tilde{w}_n]^T \\
\mathbf{r} &:= [\mathbf{0}, ..., \mathbf{0}, \sum_{i=0}^{m} \tilde{N}_0(u_i, v_i), ..., \sum_{i=0}^{m} \tilde{N}_n(u_i, v_i)]^T \\
\mathbf{I} &:= \text{identity matrix of order } 3 \\
\bar{\mathbf{d}}_i &:= [\tilde{w}_i \tilde{d}_{ix}, \tilde{w}_i \tilde{d}_{iy}, \tilde{w}_i \tilde{d}_{iz}] \\
\overline{N}^i_{pq} &:= \tilde{N}_p(u_i, v_i) \tilde{N}_q(u_i, v_i) \\
c_i &:= 1/(1 + \mathbf{M}_i^2).
\end{aligned}
$$

In all these expressions, λ is a positive constant which should be chosen large enough (see [4]) in order to ensure positivity of the weights.

3 Solving the free knot problem

The choice of the knot vector $\mathbf{T}$ has a large influence on the quality of the results of the surface fitting. It is well known that a bad placement of the knots may lead to overshooting effects that distort the shape of the surface. In this section, we describe our method of surface fitting that involves the determination of optimal knot positions.

3.1 Preparing the problem for nonlinear optimization

In the following, we set up the problem such that nonlinear optimization methods can be applied. According to section 2.4, for a given knot $\mathbf{T}$, we can solve the subproblem (12) in order to determine the corresponding weights $\mathbf{W}$ and the control points $\mathbf{D}$. In other words $\mathbf{W}$ and $\mathbf{D}$ are functions of $\mathbf{T}$ i.e.

$$(\mathbf{W}, \mathbf{D}) = (\mathbf{W}(\mathbf{T}), \mathbf{D}(\mathbf{T})) .$$

Problem (11) is therefore simplified into:

$$\min_{\mathbf{T}} \sum_{i=0}^{m} \|\mathbf{X}_{\mathbf{W}(\mathbf{T}),\mathbf{D}(\mathbf{T}),\mathbf{T}}(u_i, v_i) - \mathbf{M}_i\|^2 . \tag{19}$$

From now on, we will write only $\mathbf{X}_{\mathbf{T}}(u_i, v_i)$ instead of $\mathbf{X}_{\mathbf{W}(\mathbf{T}),\mathbf{D}(\mathbf{T}),\mathbf{T}}(u_i, v_i)$ in order to simplify the notation. Then we have

$$\min_{\mathbf{T}} \sum_{i=0}^{m} \|\mathbf{X}_{\mathbf{T}}(u_i, v_i) - \mathbf{M}_i\|^2 . \tag{20}$$

This problem still allows the occurence of the situations where μ_i and ν_i are not increasing. Therefore, we will modify this problem so that only knots with $\mu_{k_u} \le \mu_{k_u+1} \le \cdots \le \mu_{n_u}$ and $\nu_{k_v} \le \nu_{k_v+1} \le \cdots \le \nu_{n_v}$ will appear. By denoting:

$$\mathbf{X_T}(u,v) = (X_{\mathbf{T},x}(u,v), X_{\mathbf{T},y}(u,v), X_{\mathbf{T},z}(u,v)) \text{ and}$$
$$\mathbf{M}_i = (M_{ix}, M_{iy}, M_{iz}) \,,$$

we have

$$\min_{\mathbf{T}} \sum_{i=0}^{m} (X_{\mathbf{T},x}(u_i,v_i) - M_{ix})^2 + (X_{\mathbf{T},y}(u_i,v_i) - M_{iy})^2 + (X_{\mathbf{T},z}(u_i,v_i) - M_{iz})^2 \,. \tag{21}$$

By defining

$$\begin{cases} S_{3i}(\mathbf{T}) & := & X_{\mathbf{T},x}(u_i,v_i) - M_{ix} \\ S_{3i+1}(\mathbf{T}) & := & X_{\mathbf{T},y}(u_i,v_i) - M_{iy} \\ S_{3i+2}(\mathbf{T}) & := & X_{\mathbf{T},z}(u_i,v_i) - M_{iz} \,, \end{cases}$$

we obtain

$$\min_{\mathbf{T}} \sum_{i=0}^{3m+2} S_i^2(\mathbf{T}). \tag{22}$$

Now we introduce the function

$$R(x) := \begin{cases} 0 & \text{if } \; x > 0 \\ (-x)^3 & \text{if } \; x \le 0 \,, \end{cases}$$

and we define

$$\begin{aligned} R(\mathbf{T}) \; := \; & R(\mu_{k_u+1} - \mu_{k_u}) + R(\mu_{k_u+2} - \mu_{k_u+1}) + \cdots + R(\mu_{n_u} - \mu_{n_u-1}) + \\ & R(\nu_{k_v+1} - \nu_{k_v}) + R(\nu_{k_v+2} - \nu_{k_v+1}) + \cdots + R(\nu_{n_v} - \nu_{n_v-1}) \,. \end{aligned} \tag{23}$$

Instead of (22), we will consider

$$\min_{\mathbf{T}} \sum_{i=0}^{3m+2} [S_i(\mathbf{T}) + \alpha R(\mathbf{T})]^2 \,, \tag{24}$$

where α is a very large positive number. To understand the relationship between (22) and (24), we note the following two properties of equation (24).

- If we have $\mu_{k_u} \le \mu_{k_u+1} \le ... \le \mu_{n_u}$ and $\nu_{k_v} \le \nu_{k_v+1} \le ... \le \nu_{n_v}$, then $\mu_{k_u+r} - \mu_{k_u+r-1} \ge 0$ and $\nu_{k_v+s} - \nu_{k_v+s-1} \ge 0$ for all $r = 1, ..., n_u - k_u$ $s = 1, ..., n_v - k_v$ and therefore $R(\mathbf{T}) = 0$. Thus,
$$\sum_{i=0}^{3m+2} [S_i(\mathbf{T}) + \alpha R(\mathbf{T})]^2 = \sum_{i=0}^{3m+2} S_i^2(\mathbf{T}) \,.$$

- If there is some r with $\mu_{k_u+r} < \mu_{k_u+r-1}$ or some s with $\nu_{k_v+r} < \nu_{k_v+s-1}$, then

  $R(\mu_{k_u+r} - \mu_{k_u+r-1}) > 0$ or $R(\nu_{k_v+s} - \nu_{k_v+s-1}) > 0$ and so $R(\mathbf{T})$ is nonzero. Because of our assumption that α is a very large number, we can expect that $\sum_{i=0}^{3m+2} [S_i(\mathbf{T}) + \alpha R(\mathbf{T})]^2$ is also very large.

Since we are searching for the minimum of $\sum_{i=0}^{3m+2}[S_i(\mathbf{T})+\alpha R(\mathbf{T})]^2$, the preceeding two points show that a $\mathbf{T}$ with $\mu_{k_u+r} < \mu_{k_u+r-1}$ or $\nu_{k_v+r} < \nu_{k_v+s-1}$ can never realize this minimum. That means that the integration of the penalizing term in (24) avoids those $\mathbf{T}$ with $\mu_{k_u+r} < \mu_{k_u+r-1}$ or $\nu_{k_v+r} < \nu_{k_v+s-1}$.
In situation where it is desirable to have $|\mu_i - \mu_{i+1}| > \varepsilon$ and $|\nu_j - \nu_{j+1}| > \varepsilon$ for $i = k_u, ..., n_u - 1$ and $j = k_v, ..., n_v - 1$, we replace (23) by

$$\begin{aligned} R(\mathbf{T}) \ := \ & R(\mu_{k_u+1} - \mu_{k_u} - \varepsilon) + \cdots + R(\mu_{n_u} - \mu_{n_u-1} - \varepsilon) + \\ & R(\nu_{k_v+1} - \nu_{k_v} - \varepsilon) + \cdots + R(\nu_{n_v} - \nu_{n_v-1} - \varepsilon)\,. \end{aligned} \tag{25}$$

3.2 Nonlinear optimization

By introducing

$$r_i(\mathbf{T}) := S_i(\mathbf{T}) + \alpha R(\mathbf{T}) \quad \text{and} \quad K := 3m+2\;, \tag{26}$$

the surface fitting problem has the form of a usual nonlinear least square problem:

$$\min_{\mathbf{T}} \sum_{i=0}^{K} [r_i(\mathbf{T})]^2\,. \tag{27}$$

Such a problem can be solved by nonlinear least square solvers like Levenberg-Marquardt and Gauss-Newton (see [13, 6]). Note that for each evaluation of the function $r_i(\mathbf{T})$, we need to solve the subproblem (12) in order to know the corresponding $\mathbf{W}(\mathbf{T})$, $\mathbf{D}(\mathbf{T})$. We note that the order of the linear system (12) is small. It does not depend on the number of data points. It depends exclusively on the order n of the NURBS surfaces (see relation (7)). Furthermore, taking into account that the support of $N_i^{k_u}$ (resp. $N_j^{k_v}$) is $[\mu_i, \mu_{i+k_u})$ (resp. $[\nu_j, \nu_{j+k_v})$), we conclude that the matrices in (13) are sparse and therefore we need only to compute a few entries. On the other hand, we must note that the computation of one entry of this system involves all data points. The remedy to that problem is to assemble the following matrix and vector

$$F := \begin{bmatrix} I - c_0\mathbf{M}_0\mathbf{M}_0^T \\ I - c_1\mathbf{M}_1\mathbf{M}_1^T \\ \cdots \\ I - c_m\mathbf{M}_m\mathbf{M}_m^T \end{bmatrix}, \qquad \mathbf{c} := \begin{bmatrix} c_0 \\ c_1 \\ \cdots \\ c_m \end{bmatrix} \tag{28}$$

only once and store them in arrays so that they do not need to be recomputed in subsequent computations. Note that F and $\mathbf{c}$ are independent of $\mathbf{T}$. They depend only on the initial data points. The only expressions which need to be updated in each iteration of the nonlinear optimization are the values of $N_l^j(u_i, v_i)$ which are mostly zero except for some few values. They can be computed recursively in a fast way (see [1]).
As an illustration of the formerly described algorithm, we consider here two benchmarks which consist in surface reconstruction from two sets of points which are respectively of number 12221 and 12099 (See Fig 1 and Fig. 3). We plot in Fig. 2 and Fig. 4 the corresponding reconstructed surfaces.

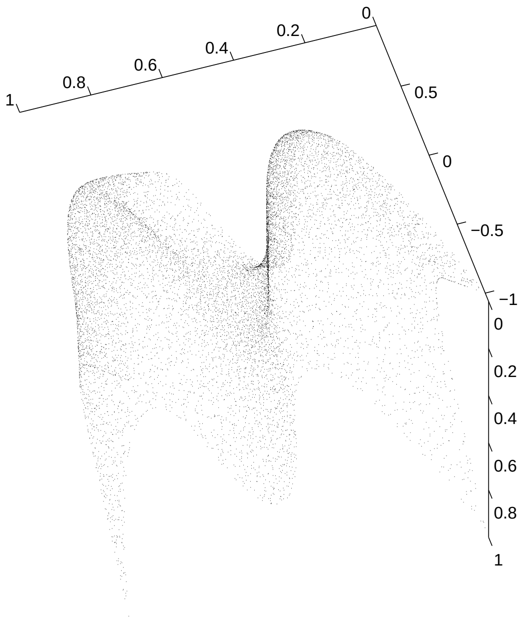

Figure 1: 12221 data points

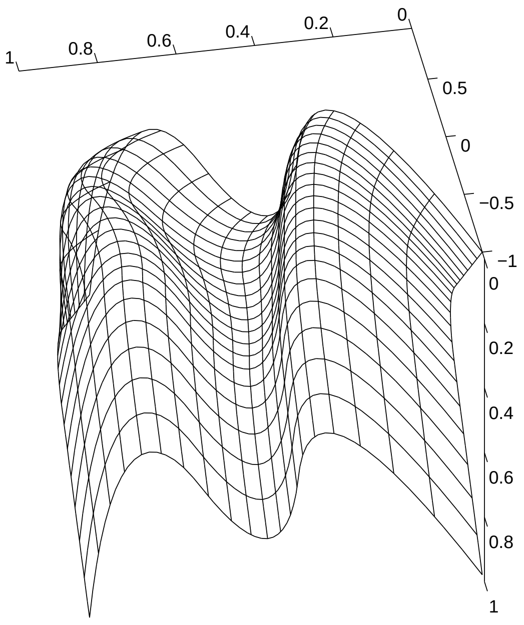

Figure 2: Reconstructed surface

4 Parallel realization

4.1 Parallel assembly of the matrices in the fixed-knot problem

We want to show how to assemble the matrices in (13) in parallel. We use the matrix A_{11} to describe our approach and note that the other matrices can be assembled in a similar way. The order of A_{11} is small but the difficulty in its computation is that it is computationally very expensive to calculate every single entry of it. In fact, for each entry (see (29)), we have to compute a sum $\sum_{i=0}^{m}$ over all data points. That makes it very time consuming if we have a large number $m+1$ of data points.

$$A_{11} := \begin{bmatrix} \sum_{i=0}^{m} \overline{N}_{00}^{i}(\mathbf{I} - c_i\mathbf{M}_i\mathbf{M}_i^T) & \cdots & \sum_{i=0}^{m} \overline{N}_{0n}^{i}(\mathbf{I} - c_i\mathbf{M}_i\mathbf{M}_i^T) \\ \vdots & & \vdots \\ \sum_{i=0}^{m} \overline{N}_{n0}^{i}(\mathbf{I} - c_i\mathbf{M}_i\mathbf{M}_i^T) & \cdots & \sum_{i=0}^{m} \overline{N}_{nn}^{i}(\mathbf{I} - c_i\mathbf{M}_i\mathbf{M}_i^T) \end{bmatrix} \tag{29}$$

We can partition the indices $i = 0, ..., m$ into κ sub-indices S_p:

$$\{0, ..., m\} = \bigcup_{p=0}^{\kappa-1} S_p \,. \tag{30}$$

Therefore, we have the following simple summation:

$$\sum_{i=0}^{m} E_i = \sum_{p=0}^{\kappa-1} \sum_{r \in S_p} E_r \,. \tag{31}$$

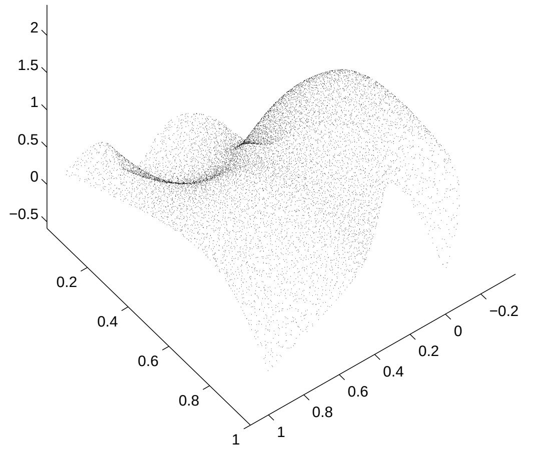

Figure 3: 12099 data points

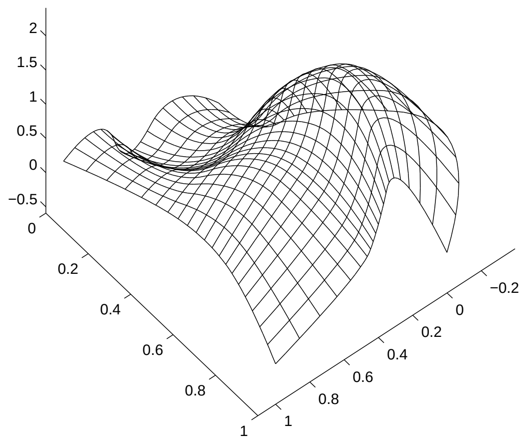

Figure 4: Reconstructed surface

That means that all processors $proc_p$ $p = 0, ..., \kappa - 1$ can compute simultaneously their own entries and afterwards we need only to sum up all local outputs in order to have the global result.

4.2 Short description of Levenberg-Marquardt algorithm

For ease of reference, we want to recall here the Levenberg-Marquardt (see [19]) for the resolution of problem (27). Let us denote by $\mathbf{r}(\mathbf{T})$ the vector function whose components are $r_i(\mathbf{T})$:

$$\mathbf{r}(\mathbf{T}) = \left[r_0(\mathbf{T}), r_1(\mathbf{T}), ..., r_{3m+2}(\mathbf{T})\right]. \tag{32}$$

The components of the unknown $\mathbf{T}$ will be denoted by τ_i i.e. $\mathbf{T} = (\tau_1, ..., \tau_l)$ where $l := (n_u - k_u + 1) + (n_v - k_v + 1)$. $\mathbf{J}(\mathbf{T})$ will stand for the Jacobian matrix whose entries are

$$J_{ij}(\mathbf{T}) = \frac{\partial r_i(\mathbf{T})}{\partial \tau_j}$$

We define also $f(\mathbf{T}) := 2 \sum_{i=0}^{K} [r_i(\mathbf{T})]^2$

```
Levenberg-Marquardt Algorithm:
step 0 : Take $\alpha_0 > 0$, $\gamma > 1$ $0 < \beta < 1$ an initial $\mathbf{T}_0$
step 1 : Set $k = 0$ and compute $f_0 := f(\mathbf{T}_0)$
step 2 : Compute $\mathbf{J}_k := \mathbf{J}(\mathbf{T}_k)$, $\mathbf{r}_k := \mathbf{r}(\mathbf{T}_k)$ and $\mathbf{g}_k := \mathbf{J}_k\mathbf{r}_k$. If
         $\|\mathbf{g}_k\| = 0$ stop, otherwise compute
             $\mathbf{Q}_k := \mathbf{J}_k^T\mathbf{J}_k$    (33)
step 3 : Solve $(\mathbf{Q}_k + \alpha_k\mathbf{I})\mathbf{u}_k = -\mathbf{g}_k$ and compute $f_{k+1} := f(\mathbf{T}_k + \mathbf{u}_k)$
step 4 : Compute $\sigma_k := (f_{k+1} - f_k)/[\mathbf{g}_k^T\mathbf{u}_k + \frac{1}{2}(\mathbf{u}_k^T\mathbf{Q}_k\mathbf{u}_k^T)]$
step 5 : If $\sigma_k < \beta$, set $\alpha_k = \gamma\alpha_k$ and go to step 3; otherwise, set
         $\mathbf{T}_{k+1} = \mathbf{T}_k + \mathbf{u}_k$ and $\alpha_{k+1} = \alpha_k/\gamma$.
step 6 : Set k=k+1 and go to step 2.
```

4.3 Parallel computation of the Jacobian matrix

The Jacobian matrix $\mathbf{J}$ is of size $(3m + 3) \times l$. That means that it has in practice very large number of raws and a few columns. When the number of points become very large, it is very costly to assemble and store this matrix. Our method is to assemble and store it simultaneously (see sketch in Fig. 5). We can still follow the approach in section 4.1 and each processor assembles therefore $3 \times \text{card}(S_p)$ rows.

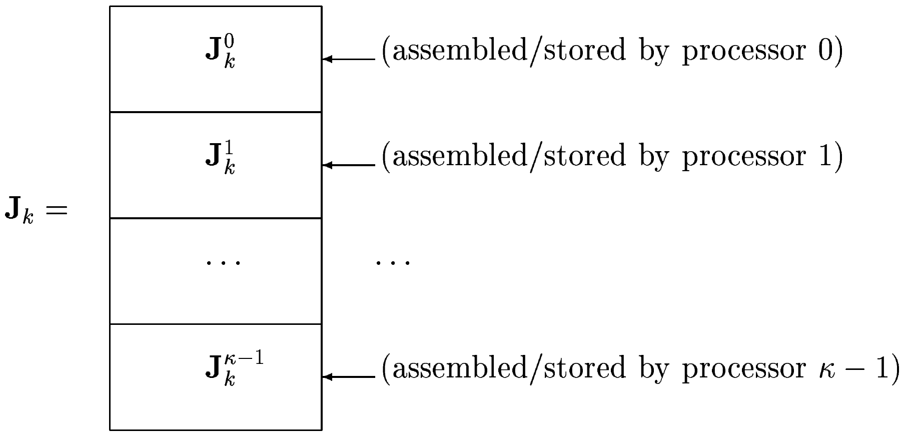
$\mathbf{J}_k =$
$\mathbf{J}_k^0$
(assembled/stored by processor 0)
$\mathbf{J}_k^1$
(assembled/stored by processor 1)
$\dots$
$\dots$
$\mathbf{J}_k^{\kappa-1}$
(assembled/stored by processor $\kappa - 1$)

Figure 5: Distribution of the matrix $\mathbf{J}_k$ to the processors

In practice, it is possible that the derivation of the function $r_i(\mathbf{T})$ is very difficult, so we use finite difference to approximate its value:

$$\frac{\partial r_i(\mathbf{T})}{\partial \tau_s} \approx \frac{r_i(\mathbf{T} + \varepsilon\mathbf{e}_s) - r_i(\mathbf{T})}{\varepsilon} \tag{34}$$

where $\mathbf{e}_s = (0, ..., 0, 1, 0, ..., 0)$ takes only value 1 at the s-th entry. We can distribute the vector $\mathbf{r}(\mathbf{T})$ in (32) among the processors in the following way:

$$\mathbf{r}(\mathbf{T}) = \begin{bmatrix} \mathbf{r}^0 \\ \mathbf{r}^1 \\ \dots \\ \mathbf{r}^{\kappa-1} \end{bmatrix} .$$

Therefore, the matrix $\mathbf{J}_j^p$ can be written block-wise as:

$$\mathbf{J}_j^p = \left[\frac{\partial \mathbf{r}^p}{\partial \tau_1} \mid \frac{\partial \mathbf{r}^p}{\partial \tau_2} \mid \dots \mid \frac{\partial \mathbf{r}^p}{\partial \tau_l} \right] . \tag{35}$$

The computation of the entries of $\mathbf{J}_j^p$ can therefore be done in l steps by noting from (34) that:

$$\frac{\partial \mathbf{r}^p}{\partial \tau_s} \approx \frac{\mathbf{r}^p(\mathbf{T} + \varepsilon \mathbf{e}_s) - \mathbf{r}^p(\mathbf{T})}{\varepsilon} . \tag{36}$$

4.4 Parallel computation of the matrix $\mathbf{Q}_k$ and the vector $\mathbf{g}_k$

Now we consider how the matrix $\mathbf{Q}_k$ in the relation (33) can be assembled in parallel. Note that $\mathbf{Q}_k$ is a very small matrix compared to its sources $\mathbf{J}_k^T$ and $\mathbf{J}_k$(see Fig. 6). We note the following simple relation:

$$\mathbf{Q}_k = (\mathbf{J}_k^0)^T(\mathbf{J}_k^0) + (\mathbf{J}_k^1)^T(\mathbf{J}_k^1) + \cdots + (\mathbf{J}_k^{\kappa-1})^T(\mathbf{J}_k^{\kappa-1}) \tag{37}$$

In order to assemble $\mathbf{Q}_k$, relation (37) simply induces that each processor $proc_p$ can assemble its own $\mathbf{Q}_k^p := (\mathbf{J}_k^p)^T(\mathbf{J}_k^p)$. And $\mathbf{Q}_k$ will be the sum of all local $\mathbf{Q}_k^p$. That can be done for instance with MPI_Reduce() if we use MPI to implement this idea. The same idea can be applied to the computation of $\mathbf{g}_k$ in step 2 by noting:

$$\mathbf{g}_k = \mathbf{J}_k^0\mathbf{r}_k^0 + \mathbf{J}_k^1\mathbf{r}_k^1 + \cdots + \mathbf{J}_k^{\kappa-1}\mathbf{r}_k^{\kappa-1} \tag{38}$$

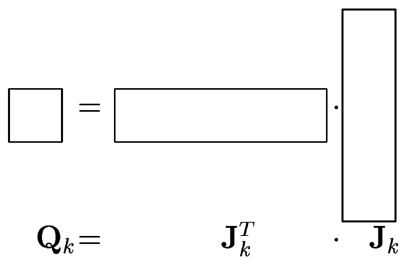

Figure 6: Size of $\mathbf{Q}_k$, $\mathbf{J}_k^T$ and $\mathbf{J}_k$

4.5 Load balancing

In order to achieve a good performance result in parallel computing, it is desirable that all processors execute approximately the same amount of computation because the goal is that if we multiply the number of processors by two, then we expect also that the program executes twice faster.
So as to achieve that load balancing, the sub-indices S_p in (30) should have approximately the same size. In other words, if we denote

$$L(p) := \mathrm{card}(S_p)\,,$$

then $L(p)$ which is an integer number that should be almost the same for all $p = 0, \cdots, \kappa-1$. The following strategy can be used to achieve that goal:

$$m_0 := \left\lfloor \frac{m+1}{\kappa} \right\rfloor, \qquad m_1 := m + 1 - m_0 \cdot \kappa$$

(where $\lfloor x \rfloor$ stands for the largest integer which is smaller or equal to x).

$$L(p) = \begin{cases} m_0 & \text{if} \quad p \geq m_1\,, \\ m_0 + 1 & \text{if} \quad p < m_1\,. \end{cases}$$

As an illustration: if we have $m + 1 = 123,025$ data points and we have $\kappa = 9$ processors, then $L(p) = 13669$ if $p \geq 4$ and $L(p) = 13670$ otherwise.

5 Parallel performance

In this section, we would like to evaluate the performance of our parallel program. First, we investigate the speedup of our algorithm. We recall the definition (see [16]) of speedup for a parallel implementation involving κ processors:

$$S(\kappa) := \frac{\text{runtime of serial program}}{\text{runtime of parallel program with}\,\kappa\,\text{processors}}$$

In Fig. 7, we show graphically the speedup of our parallel algorithm together with the ideal speedup. We can notice that we have a speedup which is practically linear. In other words, if we multiply the number of processors by two, then the time for running the algorithm is also twice faster.
Secondly, we want to investigate the efficiency (see [16, 15]) which is defined by

$$E(\kappa) := \frac{S(\kappa)}{\kappa}$$

If the efficiency $E(\kappa)$ approaches 0 then the parallel program is said to be bad because it reflects a slowdown (see [15]). Contrarily, an efficiency of 1 demonstrate a good performance. In Table 1, we gather the numerical results for the efficiency. As we can see, our

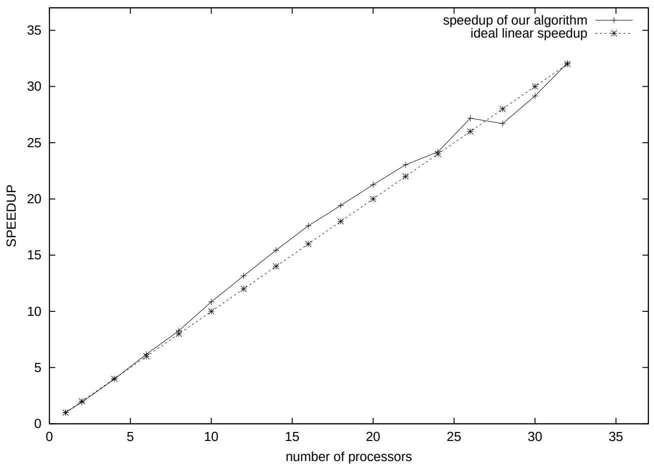

Figure 7: speedup

number of proc	runtime	efficiency
1	308.23 sec	1.000
2	161.35 sec	0.955
4	77.71 sec	0.991
8	37.25 sec	1.034
16	17.50 sec	1.100
32	9.61 sec	1.002

Table 1: efficiency

algorithm can be categorized as having good performance because the efficiency is close to 1.
In Table 2, we gather the run-times of the programs for different number of data on various number of processors. It confirms the linear speedup that we mentioned earlier.
Note that all the numerical results were obtained from the CLiC (Chemnitzer Linux Cluster) supercomputer ([3]) and the parallel program was implemented by using MPI (Message Passing Interface, see [14, 15, 8]).

number of data	64 processors	32 processors	16 processors
100,000	21.73 sec	42.54 sec	87.69 sec
200,000	43.25 sec	85.40 sec	172.03 sec
300,000	67.27 sec	129.83 sec	256.44 sec
400,000	90.08 sec	176.76 sec	342.26 sec
500,000	116.70 sec	225.38 sec	425.50 sec

Table 2: runtime in second

6 Future work

In this document, we have only considered the case of a whole NURBS surface. We plan to investigate in the near future the case of trimmed NURBS surface fitting which is has more interesting practical applications.

References

[1] C. de Boor, A practical guide to splines, Springer (New York, 1978).

[2] J. Corney, 3D modeling with ACIS kernel and toolkit, John Wiley & sons (Chichester, 1997).

[3] Chemnitzer Linux Cluster (CLiC). http://www.tu-chemnitz.de/urz/

[4] B. Elsässer, Approximation mit rationalen B-Spline Kurven und Flächen, Ph.D. thesis, Department of mathematics, Darmstadt, 1998.

[5] G. Farin, Curves and surfaces for computer aided geometric design, Academic Press, 2nd edition (Boston, 1990).

[6] R. Fletcher, Practical methods of optimization, John Wiley & Sons, 2nd edition (Chichester, 1987).

[7] J. Hoschek and D. Lasser, Grundlagen der geometrischen Datenverarbeitung, Teubner (Stuttgart, 1989).

[8] P. K. Jimack and N. Touheed, An introduction to MPI for computational mechanics, in: Parallel and Distributed Processing for Computational Mechanics: Systems and Tools, ed. B.H.V. Topping (Saxe-Coburg, 1999) 24-25.

[9] D. Jupp, Approximation to data by splines with free knots, SIAM J. Numer. Anal. 15, No. 2(1978) 328-343.

[10] K. Konno and H. Chiyokura, An approach of designing and controlling free-form surfaces by using NURBS boundary Gregory patches. Journal of Computer Aided Geometric Design 13, No. 9 (1996) 825-849.

[11] P. Laurent-Gengoux and M. Mekhilef, Optimization of a NURBS representation, Computer Aided Design 25, No. 11 (1993) 699-710.

[12] J. Neider, T. Davis and M. Woo, OpenGL programming guide, Addison-Wesley publishing company (Reading, 1994).

[13] J. Nocedal and S. Wright, Numerical Optimization, Springer Series in Operation Research (New York, 1999).

[14] P. S. Pacheco, A user's guide to MPI, Technical report, Department of mathematics, University of San Fransisco, California, 1998.

[15] P. S. Pacheco, Parallel programming with MPI, Morgan Kaufmann (San Francisco, 1997).

[16] S. Ragsdale, Parallel programming, McGraw-Hill (New York, 1991).

[17] T. Schütze, Diskrete Quadratmittelapproximation durch Splines mit freien Knoten, Ph.D. thesis, Mathematics faculty, Dresden, 1998.

[18] H. Schwetlick and T. Schütze, Least squares approximation by splines with free knots, BIT 35, No. 3 (1995) 361-384.

[19] J. Z. Zhang and L.H. Chen, Nonmonotone Levenberg-Marquardt algorithms and their convergence analysis, Journal of optimization theory and applications 92, No. 2 (1997) 393-418.

01-22 A. Eilmes, Rudolf A. Römer, M. Schreiber. Localization properties of two interacting particles in a quasi-periodic potential with a metal-insulator transition. September 2001.

01-23 M. Randrianarivony. Strengthened Cauchy inequality in anisotropic meshes and application to an a-posteriori error estimator for the Stokes problem. September 2001.

01-24 Th. Apel, H. M. Randrianarivony. Stability of discretizations of the Stokes problem on anisotropic meshes. September 2001.

01-25 Th. Apel, V. Mehrmann, D. Watkins. Structured eigenvalue methods for the computation of corner singularities in 3D anisotropic elastic structures. October 2001.

01-26 P. Cain, F. Milde, R. A. Römer, M. Schreiber. Use of cluster computing for the Anderson model of localization. October 2001. Conf. on Comp. Physics, Aachen (2001).

01-27 P. Cain, F. Milde, R. A. Römer, M. Schreiber. Applications of cluster computing for the Anderson model of localization. October 2001. Transworld Research Network for a review compilation entitled "Recent Research Developments in Physics", (2001).

01-28 X. W. Guan, A. Foerster, U. Grimm, R. A. Römer, M. Schreiber. A supersymmetric Uq[(osp)(2—2)]-extended Hubbard model with boundary fields. October 2001.

01-29 K. Eppler, H. Harbrecht. Numerical studies of shape optimization problems in elasticity using wavelet-based BEM. November 2001.

01-30 A. Meyer. The adaptive finite element method - Can we solve arbitrarily accurate? November 2001.

01-31 H. Harbrecht, S. Pereverzev, R. Schneider. An adaptive regularization by projection for noisy pseudodifferential equations of negative order. November 2001.

01-32 G. N. Gatica, H. Harbrecht, R. Schneider. Least squares methods for the coupling of FEM and BEM. November 2001.

01-33 Th. Apel, A.-M. Sändig, S. I. Solov'ev. Computation of 3D vertex singularities for linear elasticity: Error estimates for a finite element method on graded meshes. December 2001.

02-01 M. Pester. Bibliotheken zur Entwicklung paralleler Algorithmen - Basisroutinen für Kommunikation und Grafik. Januar 2002.

02-02 M. Pester. Visualization Tools for 2D and 3D Finite Element Programs - User's Manual. January 2002.

02-03 H. Harbrecht, M. Konik, R. Schneider. Fully Discrete Wavelet Galerkin Schemes. January 2002.

02-04 G. Kunert. A posteriori error estimation for convection dominated problems on anisotropic meshes. March 2002.

02-05 H. Harbrecht, R. Schneider. Wavelet Galerkin Schemes for 3D-BEM. February 2002.

02-06 W. Dahmen, H. Harbrecht, R. Schneider. Compression Techniques for Boundary Integral Equations - Optimal Complexity Estimates. April 2002.

02-07 S. Grosman. Robust local problem error estimation for a singularly perturbed reaction-diffusion problem on anisotropic finite element meshes. May 2002.

02-08 M. Springmann, M. Kuna. Identifikation schädigungsmechanischer Materialparameter mit Hilfe nichtlinearer Optimierungsverfahren am Beispiel des Rousselier Modells. Mai 2002.