Int J Adv Manuf Technol (2011) 54:691–700
DOI 10.1007/s00170-010-2947-1

ORIGINAL ARTICLE

Fast and accurate NURBS fitting for reverse engineering

Djordje Brujic · Iain Ainsworth · Mihailo Ristic

Received: 16 February 2010 / Accepted: 14 September 2010 / Published online: 30 September 2010

Abstract Fast and accurate fitting of non-uniform rational B-spline (NURBS) curves and surfaces through large sets of measured data is an important problem in applications such as reverse engineering and geometric modelling. This paper presents a method for realising significant improvements in the computational efficiency of this task. The basic idea is that the sparsity structures of the relevant matrices that are specific to the problem of NURBS fitting can be precisely defined and that full exploitation of these structures leads to significant savings in both computational and storage requirements. These savings allow for a large number of control points to be used in order to define the surface and consequently to improve the accuracy of shape representation. The achieved computational complexity is linear in both the number of measured points and the number of control points while the storage requirements of the algorithm are linear with the number of control points only. The complexity analysis, as well as the analysis of actual running times is presented. The results demonstrate that, using this approach, highly complex shapes may be modelled accurately with a single NURBS surface.

Keywords Reverse engineering · Geometric modelling · NURBS

1 Introduction

Fitting of non-uniform rational B-spline (NURBS) curves and surfaces is most often undertaken in order to reconstruct the shape of a digitised object from measured data and to represent the geometry in a form suitable for computer-aided design (CAD) [1, 2], or when an existing CAD model needs to be modified using measured data so that it reflects the actual shape of the manufactured object [3–5]. NURBS fitting is also needed in situations when an existing CAD model needs to be represented by a smaller number of entities. This may be required by the down-stream applications such as mesh generation for computer-aided engineering (CAE) analysis [6], which often fail in the presence of large number of entities and modelling imperfections such as surface gaps and overlaps.

For shape reconstruction from measured data it is often suggested that the algorithms are to be executed in two phases: (1) segmentation of point clouds and (2) sub-cloud fitting [7]. Unfortunately, such approach is fraught with difficulties and algorithms often fail [8]. Segmentation typically requires analysis of the underlying shape, such as evaluation of curvatures and edge extraction, which is difficult to perform without prior surface generation and is particularly hard in the presence of measurement noise and non-uniform data distribution. Combining surface patches into a contiguous model is also non-trivial in view of the need to match the parametric directions, parameterisation and control point locations of the adjacent surface patches. The issues outlined above usually demand a large amount of user interaction and a wide range of modelling functions.

As a general rule, it is desired that the overall surface is composed of as few entities as possible, but this in turn demands the ability to efficiently deal with a large number of data points that define the object shape and a large number of control points that allow accurate modelling of that shape. Both the computational time and the memory requirements have been found to be the factors limiting the size of the problem that can be adequately handled in practice [9].

D. Brujic (✉) · I. Ainsworth · M. Ristic
Imperial College of Science, Technology and Medicine, London, UK
e-mail: d.brujic@imperial.ac.uk

The work presented in this paper focuses on the computational efficiency in executing the task of fitting a single NURBS surface using all of the available data points, where knot insertion may be arbitrarily applied to introduce as many control points as may be needed to represent the surface features with the required accuracy. It is illustrated by the example in Fig. 1, which shows the level of detail and quality of reconstruction which can be obtained using our algorithm implementation and modest computing hardware. The measured data set is a laser scan of the TI85 calculator. In this example, a set of 120,000 measured points was used. The fitted surface comprises 44,225 control points, obtained through repeated uniform knot insertion and fitting until the desired level of detail was obtained.

The demonstrated surface fitting would normally lead to instability of the system as a direct result of the insufficient measurement density in relation to knot segments, which in our experience can often happen under repeated knot insertion, even with dense data sets such as this [10]. Here, such problems were readily overcome by adopting our solution based on regularisation, which is briefly explained in Section 2, while for the full treatment of the problem we refer the readers to our previous paper [11]. The improvements in computational efficiency presented in Section 3 remain fully applicable to this regularisation method.

2 Overview of least squares NURBS surface fitting

A NURBS surface of degree *p* in the *u* direction and degree *q* in the *v* direction is a bivariate vector-valued piecewise rational function of the form:

$$S\,(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}\,(u,v)\,P_{i,j} \tag{1}$$

where control points $P_{i,j}=(x_{ij}, y_{ij}, z_{ij})$ form a bi-directional control net with $N_u=n+1$ and $N_v=m+1$control points in each direction ($N_u\,N_v=N$ control points in total) and $R_{ij}(u,v)$ are piecewise rational basis functions [12], of degree *p* and *q*, defined by the knot vectors.

2.1 Least squares fitting procedure

Applying the Eq. 1 to each of *M* measured points, $Q_k=(x_k, y_k, z_k)$, a system of equations is created:

$$Q_k = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}\,(u_k,v_k)\,P_{i,j}, \quad k=0,\ldots,\ M-1$$

This can be expressed in matrix form as:

$$\mathbf{RP} = \mathbf{Q} \tag{2}$$

where ***R*** is the observation matrix, ***P*** is the vector of unknown control points and ***Q*** is the vector of measured points. The fitting task can be summarised as follows: given a vector $\boldsymbol{Q} \in \boldsymbol{R}^M$ and a matrix $\boldsymbol{R} \in \boldsymbol{R}^{MxN}$, vector $\boldsymbol{P} \in \boldsymbol{R}^N$ has to be found such that ***RP*** is the “best” approximation to ***Q***. There are many possible ways of defining the “best” solution. A choice, often motivated by statistical reasons is the least squares method, as it effectively reduces the influence of random errors in measurements. It computes ***P*** such that it minimises the functional *f*:

$$f = \sum_{k=0}^{M-1} (Q_k - S(u_k,v_k))^2 \tag{3}$$

or in a matrix form:

$$f = \|\mathbf{RP}-\mathbf{Q}\|_2^2$$

It follows from elementary calculus that the minimum of *f*, and at the same time the least squares solution for the system of Eq. 2 may be found by solving the set of normal equations [13, 14]:

$$\mathbf{R}^T\mathbf{RP} = \mathbf{R}^T\mathbf{Q} \tag{4}$$

In order to overcome ill-conditioning and even rank deficiency, it is necessary to regularise the linear system. The principle of regularisation [13] is to expand the

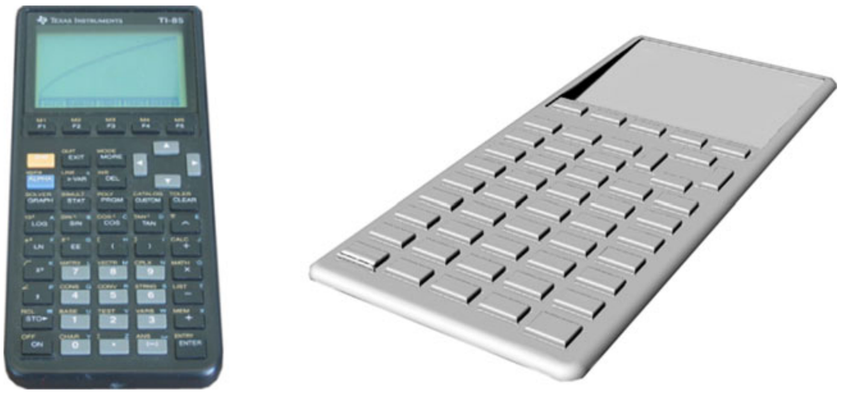
Fig. 1 Calculator and the model after repeated knot refinement and fitting (44,225 control points)

functional to be minimised as follows:

$$f = \|\mathbf{RP} - \mathbf{Q}\|_2^2 + \lambda^2 \|\mathbf{CP} - \mathbf{D}\|_2^2 \tag{5}$$

where $\|\mathbf{CP} - \mathbf{D}\|$defines some other fitting criterion as a constraint and the constant λ>0 provides a trade-off between the two criteria.

Importantly, when a quadratic minimisation principle is combined with a quadratic constraint and both are positive, only one of the two need be non-degenerate in order to make the overall problem well posed. It is also worth noting that the two parts of the minimisation will have comparable weights [15] by choosing:

$$\lambda^2 = \frac{\mathrm{Tr}(\mathrm{R}^\mathrm{T}\mathrm{R})}{\mathrm{Tr}(\mathrm{C}^\mathrm{T}\mathrm{C})}$$

Thus in the knowledge that matrix $\boldsymbol{R^TR}$ is possibly degenerate, a new criterion and a corresponding weight λ may be chosen to set up a well-posed problem. A number of such regularisation schemes have been proposed.

2.2 Regularisation solution

Our main idea is based on the fact that control points do approximate the surface and it seems natural to keep them as close to the surface as possible by introducing an additional criterion. We therefore suggest minimising the sum of the squared distances between the control points and their corresponding points on the fitted surface. We expect this criterion to smoothen the surface and to involve an equivalent of energy minimisation.

Mathematically, we expand the functional of Eq. 3 to include an additional *"α-criterion"*, as follows:

$$\mathrm{f} = \sum_{\mathrm{k}=0}^{\mathrm{M}-1} |\mathrm{Q_k} - \mathrm{S}(\mathrm{u_k}, \mathrm{v_k})|^2 + \alpha \sum_{\mathrm{i}=0}^{\mathrm{n}} \sum_{\mathrm{j}=0}^{\mathrm{m}} \left|\mathrm{P_{i,j}} - \mathrm{S}(\mathrm{u_{i,j}}, \mathrm{v_{i,j}})\right|^2$$

where $P_{i,j}$ are the control points and $S(u_{i,j},v_{i,j})$are their corresponding points on the surface, while coefficient $\alpha \geq 0$ provides the required trade-off flexibility

Naturally, the question arises as to how to define the corresponding surface points. We adopted a solution using Greville abscissae [16] because they provide the most regular matrix, as they are obtained using knot averaging. For example, in the case of a cubic B-spline, no two adjacent knot segments will be without Greville points.

After implementing this idea, we conducted extensive experiments with a variety of shapes and concluded that inclusion of the *α-criterion* is indeed highly beneficial, especially in situations involving large deformations of the original model. The experiments have also shown that the *α-criterion* can produce the effect of flattening the fitted surface, especially in the regions where the data points are sparse and the control points are relatively far from the base surface. This effect is in fact often desirable. The regions with no, or with few, data points are generally re-shaped to accommodate the deformation of the regions with dense data, such that a gentle transition between the two regions is obtained. Furthermore, the flattening effect may be controlled and significantly reduced by knot insertion, making the control polygon approximate the surface more closely.

However, the inclusion of the *α-criterion* does not guarantee that the overall problem is well posed. Furthermore, it was also recognised that there are many situations when it is desired that the shape of the unmeasured, or sparsely measured, regions is preserved after updating. For this reason an additional constraint, we call it *"β-criterion"*, was introduced, which attempts to limit the displacement of the control points relative to their original positions. The overall minimisation problem still remains linear and the cost function to be minimised becomes:

$$\mathrm{f} = \sum_{\mathrm{k}=0}^{\mathrm{M}-1} |\mathrm{Q_k} - \mathrm{S}(\mathrm{u_k}, \mathrm{v_k})|^2 + \alpha \sum_{\mathrm{i}=0}^{\mathrm{n}} \sum_{\mathrm{j}=0}^{\mathrm{m}} \left|\mathrm{P_{i,j}} - \mathrm{S}(\mathrm{u_{i,j}}, \mathrm{v_{i,j}})\right|^2 + \beta \sum_{\mathrm{i}=0}^{\mathrm{n}} \sum_{\mathrm{j}=0}^{\mathrm{m}} \left|\mathrm{P_{i,j}} - \mathrm{P^0_{i,j}}\right|^2 \tag{6}$$

where $P^0_{i,j}$ is the original position of the control point $P_{i,j}$, and $\beta \geq 0$ is a weighting factor. The solution of 6 is, [17]:

$$\left[\mathbf{R}^\mathrm{T}\mathbf{R} + \alpha\,(\mathbf{B} - \mathbf{I})^\mathrm{T}\,(\mathbf{B} - \mathbf{I}) + \beta\mathbf{I}^\mathrm{T}\mathbf{I}\right]\mathbf{P} = \mathbf{R}^\mathrm{T}\mathbf{Q} + \beta\mathbf{P}^0 \tag{7}$$

where:

$$\mathbf{B} = \begin{bmatrix} \mathrm{R_{0,0}}(\mathrm{u_{0,0}}, \mathrm{v_{0,0}}) & \ldots & \ldots & \mathrm{R_{n,m}}(\mathrm{u_{0,0}}, \mathrm{v_{0,0}}) \\ \mathrm{R_{0,0}}(\mathrm{u_{0,1}}, \mathrm{v_{0,1}}) & & & \mathrm{R_{n,m}}(\mathrm{u_{0,1}}, \mathrm{v_{0,1}}) \\ \vdots & & & \vdots \\ \mathrm{R_{0,0}}(\mathrm{u_{n,m}}, \mathrm{v_{n,m}}) & \ldots & \ldots & \mathrm{R_{n,m}}(\mathrm{u_{n,m}}, \mathrm{v_{n,m}}) \end{bmatrix}$$

and $\mathbf{P}^0 = \left[P^0_{0,0} P^0_{0,0} \quad \ldots \quad \ldots \quad P^0_{n,m}\right]^T$

This represents a set of generalised normal equations.

By examining Eq. 7, it can be seen that all elements of $\boldsymbol{R^TR}$ are positive and that $\beta\boldsymbol{I}$ is a positive diagonal matrix for $\beta > 0$. This clearly means that any non-negative value of β will guarantee that the system is stable. In practice, the choice of α and β is based on the relative weighting that the user assigns to the corresponding terms relative to the weighting of the measured data [11]. We have found that optimal settings for α and β are α=0.1 and β=0.001, respectively.

The system of Eq. 7 is linear with respect to the control points, $P_{i,j}$, but it would become non-linear if the unknowns also included the knot vectors, parameters (u_k, v_k) or the weights, $w_{i,j}$. In the work presented in this paper, the data set is unordered and in principle unknowns do include all of the above parameters, so an appropriate strategy is required to keep the system linear. As proposed by Ma and Kruth [18], and further elaborated by Piegl and Tiller [19], the knot vectors and weights may be taken directly from a base surface which may be considered as a first approximation of the final surface. The source presents a number of methods to generate the base surface. The parameters (u_k, v_k) are then computed by projecting the measured points onto the base surface [20]. If we take the fitted surface as a new base surface and repeat both the parameterisation and the fitting steps, then the results can be significantly improved, albeit at the cost of extra computations.

Consequently, there are three main parts to the surface fitting algorithm. The first is to initialise the fitting surface and the related parameters, the second to fit the surface to the data points, and the third to insert additional knots if necessary [19]. The second and the third steps may be repeated until the required accuracy is achieved.

The setting up of normal equations constitutes the preparation phase in the fitting process. Clearly, this is a conceptually simple task. However, in terms of computational resources it is an extremely demanding one and the proposed solution will be presented in Section 3.1.

The normal equations may be solved by a number of methods, which can be broadly divided into direct ones, which require a fixed number of computations, and iterative ones. A suitable direct method is Cholesky factorisation, but in common with all direct methods it suffers from “fill in”, i.e., it has non-zero values in positions which are zero in $\boldsymbol{A}=\boldsymbol{R}^T\boldsymbol{R}$. Here we note that the QR factorisation is a more stable way to determine the least squares solution of 2, but requires even more computation [21].

This can be overcome by employing an iterative method, for example, the method of successive displacements—the Gauss–Seidel method—which computes a sequence of approximations $\boldsymbol{P}^{(1)}$,..., $\boldsymbol{P}^{(n)}$, until a sufficiently accurate solution is obtained. Thus given $\boldsymbol{P}^{(k)}$ we compute:

$$\mathrm{P}_i^{(k+1)}=\frac{1}{\mathrm{A}_{ii}}\left(\mathrm{Q}_i-\sum_{j=0}^{i-1}\mathrm{A}_{i,j}\mathrm{P}_j^{(k+1)}-\sum_{j=i+1}^{N-1}\mathrm{A}_{i,j}\mathrm{P}_j^{(k)}\right),\quad i=0,\ldots,N-1 \tag{8}$$

The Gauss–Seidel method was adopted as the primary method in our implementation for the solution phase, while the improved Cholesky decomposition using band method [22] was also implemented as a direct method alternative.

3 Efficiency improvements in least squares NURBS fitting

Firstly, we turn our attention to a matrix multiplication algorithm and then study the detailed structure of the matrices $\boldsymbol{R}$ and $\boldsymbol{R}^T\boldsymbol{R}$ and suggest how to use them to improve computational efficiency.

3.1 The preparation phase

A significant proportion of the time needed for processing the matrices can be attributed to matrix multiplication of the type $\boldsymbol{R}^T\boldsymbol{R}$. Using the “standard multiplication” each element $A_{i,j}$ is calculated as:

$$\mathrm{A}_{i,j}=\sum_{k=0}^{M-1}\mathrm{R}_{k,i}\mathrm{R}_{k,j}$$

Clearly, the computational complexity is O (N^2M). This algorithm is not suitable for large problems since each column needs to be accessed many times and consequently the total storage requirement is of complexity O ($M\times N+N^2$), i.e. it is proportional to the sum of spaces for the storage of $\boldsymbol{R}$ and spaces for the storage of $\boldsymbol{A}$. Another way to compute the matrix $\boldsymbol{A}$ would be to consider a single row $\boldsymbol{R}_k$ of a matrix $\boldsymbol{R}$ at a time, multiply each element $R_{k,i}$, by each of the elements $R_{k,j}$ *of the same row* and add the product to $A_{i,j}$ [13]. The validity of this construction may be verified by considering an element A_{ij} and the elements of $\boldsymbol{R}^T$ and $\boldsymbol{R}$ that contribute to its value. When using this method, rather than computing elements of the matrix $\boldsymbol{A}$ sequentially, we compute contribution of each measured point in turn to all the elements of the matrix $\boldsymbol{A}$. It should be noted that each row of $\boldsymbol{R}$ corresponds to a single measured point. By sequencing the operations in this manner the algorithm uses only a single pass through the data and storage is needed only for the matrix $\boldsymbol{A}$. Note that this makes memory requirements independent of the number of measured points.

Much of the usefulness of NURBS stems from their local support property, which means that an individual control point can influence the shape of the curve/surface only in a strictly limited area in its vicinity. It can be proven that the position of a point on the surface is determined by only (p+1) (q+1) control points, as all other $R_{i,j}$ values are zero [21]. Actually, assuming that control points are stored by constant v, every row of $\boldsymbol{R}$ contains precisely (q+1) groups of (p+1) consecutive non-zero elements. The “pitch” between those groups is equal to the number of control points in the u direction, N_u.

Formally, given a data point with parameterisation u and v, within the knot segments $u_l<u<u_{l+1}$ and $v_k<v<v_{k+1}$ respectively, where $p<=l<=N_u$-1 and $q<=k<=N_v-1$, the only possible non-zero elements within the row of the matrix $\boldsymbol{R}$ that correspond to this measured point are given by the index i where:

$$i=l-\mathrm{p}+(\mathrm{k}-\mathrm{q})\times\mathrm{N_u}+\mathrm{a}+\mathrm{b}\times\mathrm{N_u}$$
$$\text{where}:\ \mathrm{a}=0,\ldots,\mathrm{p}\text{ and }\mathrm{b}=0,\ldots,\mathrm{q} \tag{9}$$

The regularity of occurrence of the non-zero elements in $\boldsymbol{R}$ introduces a regularity of occurrence in the pattern of the non-zero elements in $\boldsymbol{A}$ [7]. The necessary and sufficient condition for an element $A_{i,j}$ to be non-zero is that at least one row of a matrix $\boldsymbol{R}$ has non-zero elements at both columns i and j. For the purpose of this work it is more convenient to consider this situation in terms of a distance δ between the ith and the jth elements of $\boldsymbol{R}$ in a row m, hence

$j=i+\delta$. Also, we note that this element, $A_{i,i+\delta}$, is situated exactly δ places away from the main diagonal.

As the indices of the non-zero elements of any row of a matrix ***R*** are given by formula 9, we can represent the index of a non-zero element, $n=i+\delta$, by:

$$i+\delta=l-\mathrm{p}+(\mathrm{k}-\mathrm{q})\times\mathrm{N_u}+\mathrm{c}+\mathrm{d}\times\mathrm{N_u}\ \text{where}:\ \mathrm{c}=0,\ldots,\mathrm{p}\ \text{and}\ \mathrm{d}=0,\ldots,\mathrm{q} \tag{10}$$

Subtracting (9) from (10) we get

$$\delta=l-\mathrm{p}+(\mathrm{k}-\mathrm{q})\times\mathrm{N_u}+\mathrm{c}+\mathrm{d}\times\mathrm{N_u}$$
$$-(l-\mathrm{p}+(\mathrm{k}-\mathrm{q})\times\mathrm{N_u}+\mathrm{a}+\mathrm{b}\times\mathrm{N_u})$$

giving:
$$\delta=c-a+(d-b)\times N_u$$

Substituting all possible values for c and a in turn it can be seen that (c-a) can only take the values within the limits [–p, p]. Likewise, substituting all possible values for d and b in turn we can see that (d-b) can only take the values within the limits [–q, q]. Hence the value of δ is given by:

$$\delta=\mathrm{e}+\mathrm{f}\times\mathrm{N_u}\ \text{where}:\ \mathrm{e}=-\mathrm{p},\ldots,\mathrm{p}\ \text{and}\ \mathrm{f}=-\mathrm{q},\ldots,\mathrm{q} \tag{11}$$

Consequently, the only possible non-zero elements in a matrix A are found at the positions $A_{i,i+\delta}$ (situated exactly δ places away from the main diagonal) and are confined in stripes. From Eq. 11, we can work out that there will be exactly $2q+1$ stripes of width at $2p+1$ each and separated by a distance equal to the number of control points in the u direction. Figure 2 presents the detailed sparsity structure of the matrix ***A***. The length of each stripe is less than or equal to N and the total number of non-zero elements in the whole matrix ***A*** is less than $(2p+1)$ $(2q+1)N$. Therefore by accounting for the sparsity structure of the matrix ***A***, the algorithm's storage complexity has dropped significantly, from $O(M\times N+N^2)$ to $O(N)$.

To summarise, for each measured point Q_k, the closest point on the base surface is computed and its parametric values are used in the evaluation of the rational basis functions ***R***. The contributions of this point are then added to all possibly non-zero elements of the matrix ***A***. This is implemented through the use of a new matrix ***S***, big enough to keep the non-zero values of ***A***. Each non-zero stripe of the matrix ***A*** then becomes a column of ***S***. As a result, instead of storing the non-zero element $A_{i,j}$ into ***A*** we store it as $S_{g,i,h}$ into a matrix ***S***. The indices i,j referring to an element of the matrix ***A*** are transformed into the indices g,i,h referring to an element in the new matrix ***S***, as follows:

$$\mathrm{g}=(\mathrm{j}+\mathrm{q}{*}\mathrm{N_u}+\mathrm{p}-\mathrm{i})/\mathrm{N_u}$$

$$\mathrm{i}=\mathrm{i}$$

$$\mathrm{h}=\mathrm{j}+\mathrm{q}{*}\mathrm{N_u}+\mathrm{p}-\mathrm{i}-\mathrm{g}{*}\mathrm{N_u}$$

The results of these improvements are presented in the Table 1. It is evident that the speed improvement is of $(N/(p+1)\,(q+1))^2$ times. Roughly, the computations that previously could have taken 100 h to complete would be executed in less than a second when the sparsity of the matrix ***R*** is exploited in this way.

The following C++ code, omitting declaration and initialisation, provides an example of how computation and storage of non-zero elements of ***A*** may be implemented.

```
for (m=0; m<M; m++) {                          // for each measured point
      CalcUV(Q[m], &u, &v);                    // compute u and v of its closest point
      CalcLK(u, v, &l, &k);                    // compute indices of its closest point
      CalcBasisFn(Entity, u, v, R[]);          // compute non zero basis functions
      for (b=0; b<q; b++) {                    // update the affected Aij only
            for (a=0; a<p; a++) {
                  i = l-p + (k-q) * Nu +a +b*Nu;
                  for (b1=0; b1<q; b1++) {
                        for (a1=0; a1<p; a1++) {
                              j = l-p + (k-q) * Nu +al +bl*Nu;
                              A[i][j] += R[i] * R[j];
}     }     }     }     }
```

The results in Table 2 are included in order to provide an indication of the realised performance, obtained using a Intel Xeon 3.2 GHz processor and 2 GB of RAM computer. From the table it is apparent that the storage requirements in the proposed method are independent of the number of measured points and are proportional only to the number of control points, meaning that very large systems are solvable using modest computing resources. Also, the time com-

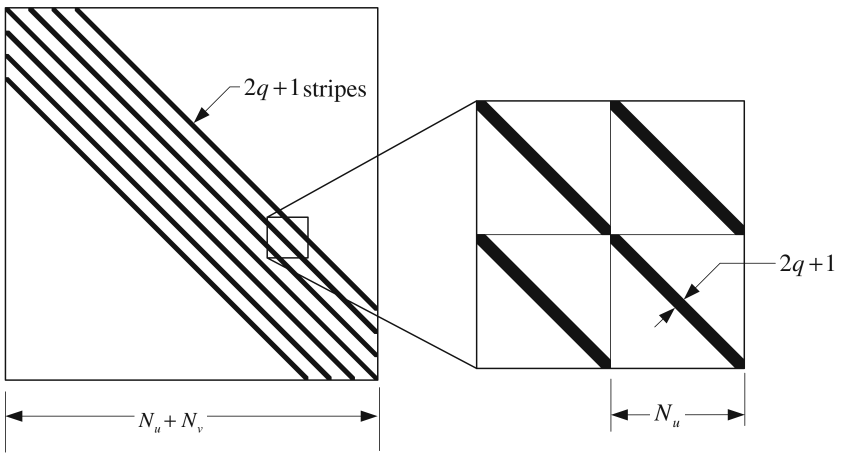

Fig. 2 Sparsity structure of the matrix $A=R^TR$

plexity of the proposed method is linear with the number of measured points. It is worth mentioning that $\boldsymbol{A}$ is symmetric, therefore it will suffice to calculate one half of the matrix $\boldsymbol{A}$.

3.2 The solution phase—Gauss–Seidel method

The observed sparsity of $\boldsymbol{A}$ is now ready to be further exploited in the solution phase in order to reduce its computational complexity. Clearly, the knowledge of matrix sparsity would lead to speed enhancements in both the direct and the iterative solution methods, but the iterative ones, such as the Gauss–Seidel method [23], are better suited for solving very large, sparse systems [13, 24]. The enhancement of the computational efficiency is achieved by multiplying only the non-zero elements of the corresponding rows and columns, instead of multiplying them in their entirety.

The optimised C++ code that illustrates the computation of Eq. 10 is presented below:

```
for (i=0; i<N; i++) {                                   // for each row
        for (n=-q; n<q+1; n++) {                        // for each non-zero element
                for (m=0; m<2*p+1; m++) {
                        j = n * Nu - p + i + m;         // compute intermediate index
                        if (i != j) {                   // if not on the diagonal
                                g = (j + q * Nu + p – i) / Nu; // compute indices in S
                                h = j + q * Nu + p - i - g * Nu;
                                sum -= S[g][i][h] * P[j];
        }       }       }
        P[i] = (Q[i] + sum) / S[q][i][p];
}
```

Consequently, the computational complexity for one iteration of the algorithm has dropped in the solution phase from a quadratic dependence on the number of control points, $O(N^2)$, to a linear dependence, $O(N)$.

Table 3 provides an illustration of the computational performance achieved by the code above for the solution

Table 1 Computational and storage complexity in the preparation phase

	Storage complexity	Computational complexity
Standard technique	$O(MN+N^2)$	$O(MN^2)$
Sparsity of A exploited	$O((p+1)(q+1)+N(2p+1)(2q+1))$	$O(M((p+1)(q+1))^2)$

Table 2 Preparation phase: performance with regard to computational and storage complexity

Number of data points	Number of control points			
	10^2	10^3	10^4	10^5
10^4	46 ms	63 ms	93 ms	–
	42.6 kb	425 kb	4,253 kb	
10^5	422 ms	502 ms	875 ms	1,159 ms
	42.6 kb	425 kb	4,253 kb	4,2534 kb
10^6	4,172 ms	4,813 ms	8,797 ms	1,1468 ms
	42.6 kb	425 kb	4,253 kb	4,2534 kb

Table 3 Solution phase: Gauss–Seidel method performance with regard to computational complexity

Number of data points	Number of control points			
	10^2	10^3	10^4	10^5
10^4	15 ms	140 ms	1,125 ms	–
10^5	31 ms	156 ms	2,007 ms	13,672 ms
10^6	32 ms	171 ms	1,797 ms	21,285 ms

Table 4 Solution phase: Cholesky band method performance with regard to computational complexity

Number of data points	Number of control points			
	10^2	10^3	10^4	10^5
10^4	11 ms	93 ms	7,344 ms	–
10^5	10 ms	94 ms	7,235 ms	1,967,339 ms
10^6	12 ms	94 ms	7,328 ms	1,926,738 ms

phase. The termination criterion was that, for each control point, the relative distance it had moved in the last step was smaller than a pre-set tolerance—in our case, 0.0001.The time is given in milliseconds and the observed number of iterations required for convergence varied between 20 and 50. As for the preparation phase, the storage requirements in the proposed method are independent of the number of measured points and are proportional only to the number of control points while the computational complexity of the proposed method is linear with the number of control points.

The Gauss–Seidel method was subsequently compared with the Cholesky method as a direct method alternative. The computational complexity of the standard Cholesky method is $O(N^3)$, [13]. However, we have implemented the band method as the most suited variation of Cholesky's techniques, in view of the sparsity of the matrices involved. In a banded matrix all of the non-zero elements belong to a band centred at the main diagonal. The definition is as follows: $\boldsymbol{A} \in \boldsymbol{R}^{NxN}$ is a banded matrix with half bandwidth p if $A_{ij}=0$ for $|i\text{-}j|>p$. In our implementation of the Cholesky band algorithm, the upper triangle of the banded matrix A to be factorised is supplied in banded storage mode [22]. In this storage scheme, the band of the matrix is stored into a rectangular matrix and, if the half bandwidth is p, the new array has $p+1$ rows and as many columns as the original matrix.

A detailed treatment of this method is provided in George and Liu [22], including the derivation of expressions for computational complexity. In the case of NURBS fitting and the consequent band width (Fig. 2), the computational complexity can be shown to be $O(N^2)$. The corresponding results of our tests are presented in Table 4.

It is difficult to provide a direct comparison of the computational speed between the two solution methods, as one uses direct computation and the other an iterative procedure that stops at a pre-set value. Also, the computational time of an iteration procedure varies significantly with the quality of the initial guess. Nevertheless, the data in Tables 3 and 4 provides an indicative comparison based on experiments with the two methods, involving variation in the number of control points on a large scale, over several orders of magnitude.

The results obtained with the proposed method are exactly the same as the results obtained by the standard solutions of the generalised normal equations and for this reason the accuracy is considered identical to the one of the standard methods.

The comparison results are also shown in Fig. 3. When the number of control points, N, is smaller than 1,000 Cholesky method appears to be superior, while the iterative procedure is seen to be faster for $N>1{,}000$.

As an illustration, for 100,000 control points, the processing time is over half an hour if Cholesky method is used, while it takes only 13 s to process the data using the iterative method. With increased number of control points ($N>100{,}000$), Cholesky becomes too slow and it takes up prohibitive amount of memory, which renders it practically unusable for such applications in surface fitting. The precise limit of course varies, depending on the computer specification.

4 Examples

Two examples where chosen to illustrate the key aspects of the proposed method—its iterative nature and its usefulness in practice. The first examples-modelling of the mobile telephone keypad, illustrates the iterative nature of fitting using base surface, while the second-modelling of the injection moulding tool, illustrates usability of the presented work in engineering design, analysis, and manufacture.

Figure 4 shows a data set comprising 40,400 points, corresponding to the front cover and keypad of a mobile telephone handset. The aim was to create a model that accurately captures all the shape details represented by the data. The points were unordered and the density was non-uniform, providing significantly more data in the curved areas than in the flat areas, as the figure demonstrates. The base surface is shown in Fig. 5a, and it was constructed interactively as a Coons patch, with 189 control points. The NURBS surface was then fitted through the points using the knot vectors from the base surface and parameters for each point taken from the projection on the base surface. The

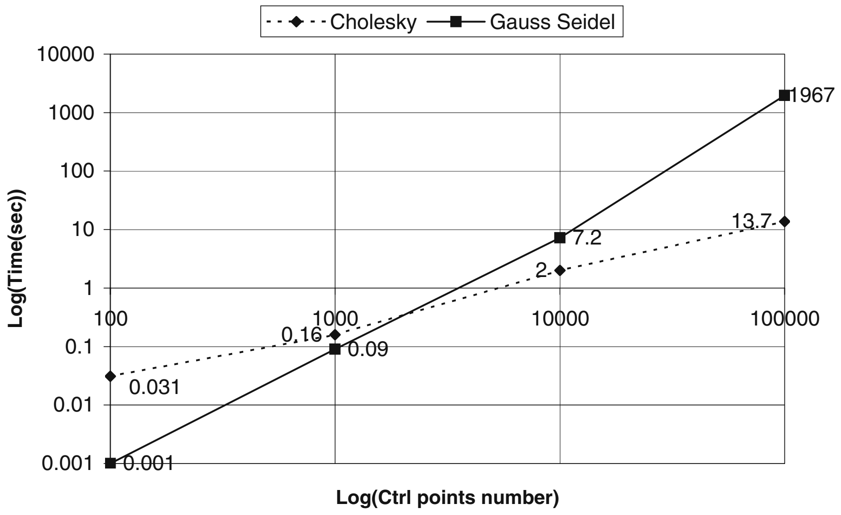

Fig. 3 Time for Cholesky (*dashed line*) and optimised iterative method (*solid line*) against a number of control points

result is shown in Fig. 5b. After a uniform knots insertion the process was repeated in order to obtain more accurate reconstruction, as shown in Fig. 5c. The final model, shown in Fig. 5d was constructed after repeated knots insertion and fitting, eventually using 163,840 control points that were needed to model the fine details. The entire procedure included 730 iterations and it took less than 1 h to complete using an Intel Xeon 3.2 GHz processor and 2 GB of RAM.

Figure 6 shows an injection moulding tool for a car dashboard. This is a representative example of a high cost engineering component that is relatively large (about 2,000 mm length) yet contains numerous fine features that need to be represented with high accuracy (better than 0.1 mm). An important part of the mould development process is to assess its structural performance using finite element analysis, because the mould is in production subjected to high pressures that it must withstand with acceptably small deformation. The original design of the mould was produced using a standard CAD package and consisted of 156,443 entities. It was provided as a neutral format IGES file for processing by downstream systems. The model inevitably contains numerous imperfections such as small gaps and overlaps between surface entities. The native CAD system accommodates such imperfections based on the tolerance definitions and extensive topological information produced during model creation, however such information is largely lost when converting to a neutral data exchange format. Consequently, stand alone mesh generation packages often fail when dealing with such models. Interactive model repair in preparation for meshing is a hugely laborious process, often requiring many hours or 11 weeks of work.

The solution was sought in fitting a single surface to a cloud of points generated from the model. Thus the model was adaptively sampled to produce 800,000 points with non-uniform distribution and local point density adapted to the local feature size. The final result, a single surface with 120,000 control points was fitted using all data points, is presented in Fig. 6. The required mesh was subsequently readily generated from this model and the concept of using a single surface representation was proved successful.

Figure 7 shows the airfoil-shaped blade of a propeller. Measured data set containing 54 000 points was modelled with 1,020 control points.

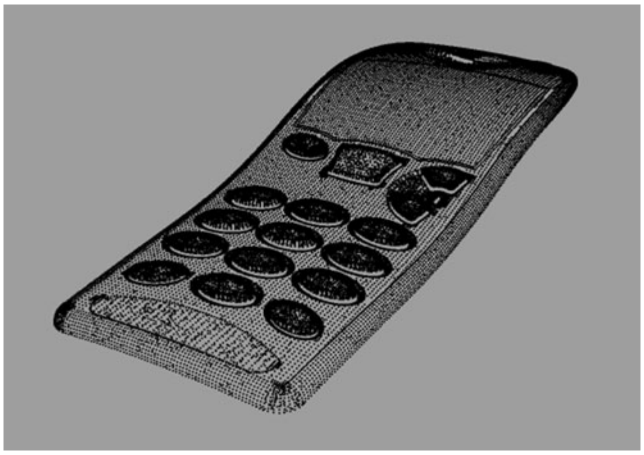

Fig. 4 Point data set for telephone handset, showing regions of very dense data required to capture fine detail of the keypad

5 Conclusions

This paper has presented details of the implementation of a method for generation and solution of generalised normal equations in NURBS fitting, which realises significant improvements in the memory requirements and the computing speed. The newly observed sparsity structure of the relevant matrices is specific to the problem of NURBS

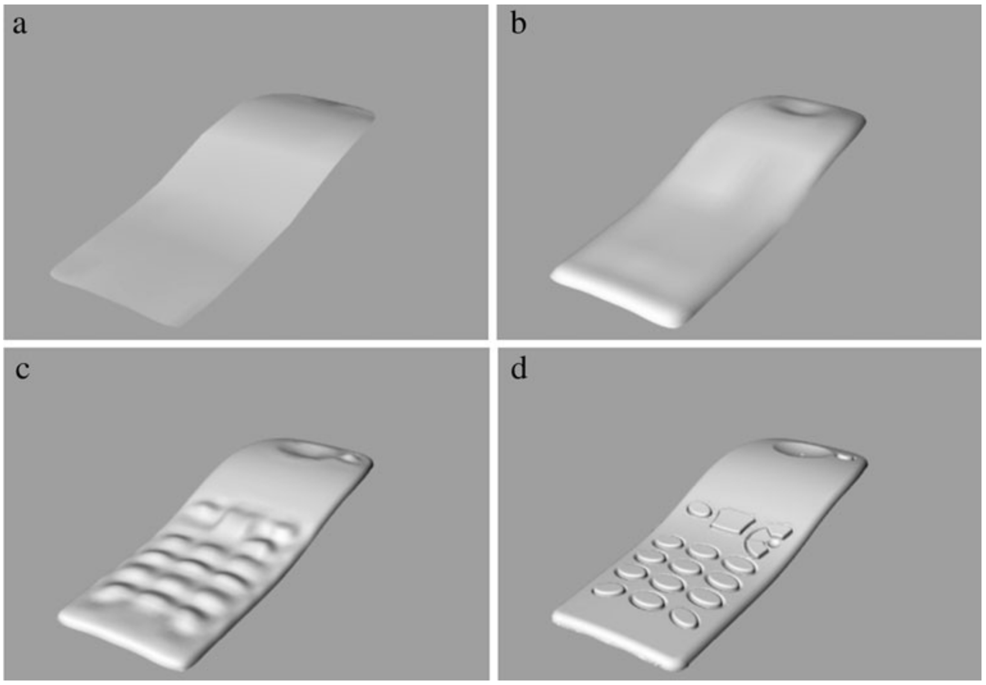

Fig. 5 Phases of surface fitting, **a** initial base surface fitted with 9×21 control points, **b** initial surface fitted to the data, **c** intermediate surface fitted with 17×41 control points, **d** final surface fitted with 256×640 control points

fitting and lends itself well to the row oriented method of matrix multiplication. The critical aspects of the fitting process were identified to be the storage of matrices $\boldsymbol{R}$ and $\boldsymbol{R}^T\boldsymbol{R}$ and the computation of the matrix product $\boldsymbol{R}^T\boldsymbol{R}$. The storage requirements for the presented method are independent of the number of measured points and linear with the number of control points. The computing speed is linear with the number of measured points and independent of the number of control points.

The normal equations were solved using two methods: A Cholesky band method and optimised iterative procedure. Both methods were implemented in order to make viable comparison in terms of their computational speed and storage requirements. The optimised iterative procedure proved distinctly superior for the large number of control points (N>1,000). The Cholesky method however proves more efficient when the number of control points is smaller (N<1,000). The results of this research provide a novel and efficient methodology for solving an important class of fitting problems.

Consequently, the method is well suited for applications involving modelling of complex shapes from a cloud of digitised points [25], where both the number of data points and the number of control points may quickly lead to a problem of insoluble size.

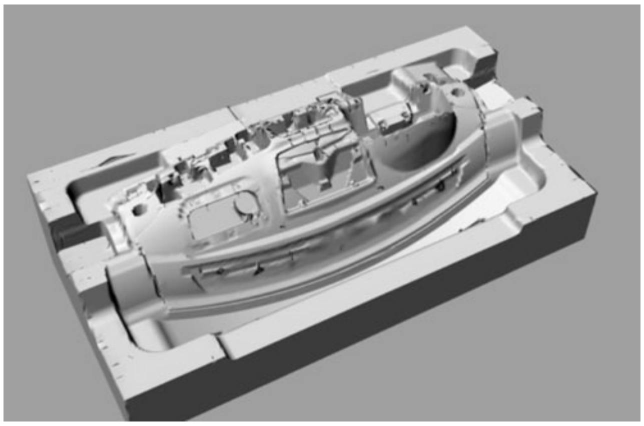

Fig. 6 Injection mould for a car dashboard

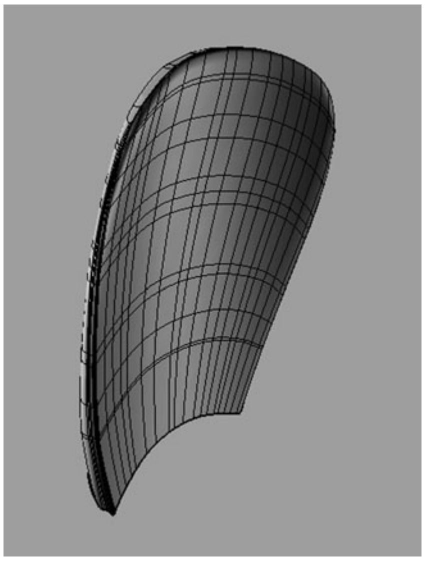

Fig. 7 Blade of a propeller

Acknowledgements This work has been partly supported by the Engineering and Physical Science Research Council grants GR/N11223 and GR/M/87351.

References

1. Dan J, Lancheng W (2006) An algorithm of NURBS surface fitting for reverse engineering. Int J Adv Manuf Technol 31:92–97
2. Ma W, Kruth JP (1998) NURBS curve and surface fitting for reverse engineering. Int J Adv Manuf Technol 14:918–927
3. Gao J, Chen X, Yilmaz O, Gindy N (2008) An integrated adaptive repair solution for complex aerospace components through geometry reconstruction. Int J Adv Manuf Technol 36:1170–1179
4. Varady T, Martin RR, Cox J (1997) Reverse engineering of geometric models, an introduction. Comput Aided Des 29(4):255–268
5. Weiss V, Andor L, Renner G, Varady T (2002) Advanced surface fitting techniques. Comput Aided Geom Des 19:19–42
6. Armstrong CG, Monaghan DJ, Price MA, Ou H, Lamont J (2002) Integrating CAE Concepts with CAD Geometry. In: Topping B, Bittnar Z (eds) Engineering computational technology. Civil-Comp Press, Edinburgh, pp 75–104
7. Dietz U (1996) B-Spline approximation with energy constraints. In: Hoschek J, Kaklis P (eds) Advanced course on fairshape. Teubner, Stuttgart, pp 229–240
8. Che X, Liang X, Li Q (2005) G1 continuity conditions of adjacent NURBS surfaces. Comput Aided Geome Des 22:285–298
9. Koch KR (2009) Identity of simultaneous estimates of control points and of their estimates by the lofting method for NURBS surface fitting. Int J Adv Manuf Technol 44:1175–1180
10. Pereyra V, Scherer G (2002) Least squares scattered data fitting by truncated SVDs. Appl Numer Math 40:73–86
11. Brujic D, Ristic M, Ainsworth I (2002) Measurement-based Modification of NURBS surfaces. Comput Aided Des 34 (3):173–183
12. Piegl L, Tiller W (1997) The NURBS book, 2nd edn. Springer, Berlin
13. Bjorck A (1996) Numerical methods for least squares problems. Society for Industrial and Applied Mathematics, Philadelphia
14. Golub GH, Van Loan CF (1996) Matrix Computations, 3rd edn. Johns Hopkins, Baltimore, MD
15. Press WH, Teukolsky SA, Vetterling SW, Flannery BP (1993) Numerical recipes in C: the art of scientific computing, 2nd edn. Cambridge University Press, Cambridge, UK
16. Farin G (2002) Curves and Surfaces for CAGD A Practical Guide, 5th edn. Morgan-Kaufmann, San Francisco, CA
17. Brujic D (2002) A Framework for CAD based Shape Metrology. Dissertation, Imperial College
18. Ma W, Kruth JP (1995) Parameterization of randomly measured points for least squares fitting of B-spline curves and surfaces. Comput Aided Des 27:663–675
19. Piegl LA, Tiller W (2001) Parameterization for surface fitting in reverse engineering. Comput Aided Des 33:593–603
20. Ma YL, Hewitt WT (2003) Point inversion and projection for NURBS curve and surface: control polygon approach. Comput Aided Geom Des 20:79–99
21. Dierckx P (1993) Curve and surface fitting with splines. Clarendon, Oxford
22. George A, Liu J (1981) Computer solution of large sparse positive definite systems. Prentice-Hall, Englewood Cliffs, NJ
23. Varga RS (1962) Matrix iterative Analysis. Prentice-Hall, Englewood Cliffs, NJ
24. Young DM (1971) Iterative solution of large linear systems. Academic, New York
25. Yin Z (2004) Reverse engineering of a NURBS surface from digitized points subject to boundary conditions. Comput Graph 28 (2):207–212