Reconstruction of Smooth Surfaces with Arbitrary Topology Adaptive Splines

A. J. Stoddart and M. Baker

Centre for Vision, Speech and Signal Processing
University of Surrey, Guildford, Surrey GU2 5XH, UK,
a.stoddart@ee.surrey.ac.uk,
WWW: `http://www.ee.surrey.ac.uk`

Abstract. We present a novel method for fitting a smooth G^1 continuous spline to point sets. It is based on an iterative conjugate gradient optimisation scheme. Unlike traditional tensor product based splines we can fit arbitrary topology surfaces with locally adaptive meshing. For this reason we call the surface "slime".
Other attempts at this problem are based on tensor product splines and are therefore not locally adaptive.

1 Introduction

Range sensing is an area of computer vision that is being successfully applied to a variety of industrial problems. By combining several range images it is possible to build complete detailed surface models of real world objects for applications in VR, graphics and manufacturing [11,10,9].

Existing methods produce large datasets consisting of up to a million polygons. There is considerable interest in the use of more efficient representations such as spline surfaces [5,17,14]. Spline surfaces are much more efficient at representing smooth manufactured surfaces, e.g. car bodywork. For this reason splines are heavily used in CAD applications. Most of the spline surfaces commonly used in computer vision have severe limitations because they cannot have arbitrary topology and cannot be adaptively meshed.

In previous work [21] we presented a deformable surface that could have arbitrary topology, and we later went on to formulate a linear form that allowed fast computation [20]. In this paper we have developed a powerful scheme for surface reconstruction from point sets. The advances presented in this paper include

- A new method for constructing seed meshes.
- Techniques for locally adaptive spline surface representation. (Very few commonly used spline surfaces can be made locally adaptive.)
- Fast techniques for solving this optimisation problem based on an iterated conjugate gradient scheme.

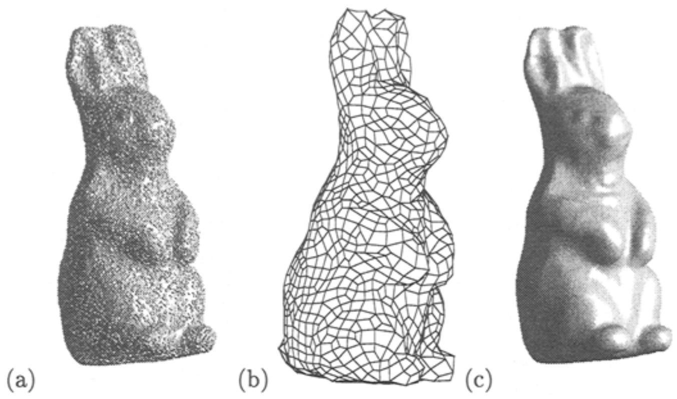

Fig. 1. Results from a surface fitted to a cloud of 15000 points (a) the point set (b) the control mesh (c) the fitted spline surface *flatshaded*

Our earlier work established a powerful representation. This work provides the tools necessary to use it in a variety of applications. Because of its special properties we have dubbed the surface 'slime'.

This work could be applied to a range of problems. We consider the following processing pipeline. Several range images are captured, registered and fused using a volumetric approach [11,10]. In this type of approach a volumetric field function is computed and the marching cubes algorithm can be used to obtain an isosurface. The result is a large piecewise flat mesh made up of triangular faces. Of course we may wish to reuse the original point measurements if accuracy has been lost by the volumetric representation. Volumetric approaches are limited by the processing considerations and memory usage rises rapidly as the voxel edge length is reduced below 1% of a side of the working volume.

This initial dense mesh and point set is taken as the starting point for our work. The aim of subsequent processing is to reduce the size of the representation, smooth the surface and increase the fidelity to the measured point set.

Accuracy is a key factor in commercial uptake of range sensing technology. A meaningful way of quoting accuracy independent of scale is to scale the working volume to lie inside a unit cube and express the rms error as a fraction of the cube size. The accuracy of commercially available range scanners varies between 1% and 0.01%. Mechanical Coordinate Measuring Machines (CMMs) can achieve 0.0001%. Any technique that proceeds via a discrete voxel approach has an error of the order of the voxel size. It is difficult to use small voxel sizes because routines for extracting an implicit surface (e.g. marching cubes) produce more than $O(10^5)$ triangles if the voxel size is reduced below 1%.

A sample set of results is now shown. In figure 1 we show a surface fitted to a cloud of 15000 points, the final spline control mesh (1459 patches) and a rendered view of the spline surface. *The rendered surface is rendered with flatshading, not a smoothed shading algorithm such as Gouraud shading as is common.*

2 Related work

Deformable curves and surfaces have been applied to many problems in computer vision. Medical imaging is an area where deformable surfaces are presently receiving much attention [18] due to the need for processing of volumetric data sets.

There has been much recent interest in shapes with non-trivial topology in computer vision in general and in deformable surfaces in particular. De Carlo and Metaxas have proposed an adaptive shape evolution scheme by blending together parts consisting of simple tensor product patches [5]. Another approach was presented by McInerney and Terzopoulos [17] who use a parallel 2D data structure to achieve a topologically adaptable snake. Related work includes that of Casselles *et al* [3] who proposed geodesic active contours based on a level set approach.

In the graphics community there is considerable interest in building models from range scanner data. Recent advances in fusion algorithms [11,10] allow the creation of detailed million polygon meshes and there is much interest in reducing the size of the representation by using higher order surfaces. Most recent work has concentrated on stitching together tensor product patches. [6,14].

3 The surface representation

The spline based surface that we use is a G^1 continuous arbitrary topology surface called a generalised biquadratic B-Spline (GBBS). It was first developed in the context of computer graphics by Loop and De Rose [15,16]. It is important to note that it is not possible to maintain C^1 continuity (first order parametric derivative) over an arbitrary topology surface. Instead the concept of G^1 continuity (first order geometric) is introduced. In effect it means that the tangent plane at a point on the surface varies continuously as the point moves on the surface.

We first presented an application of this surface to problems in computer vision in [21] and more recently [20] we succeeded in formulating a matrix form and a fast method of computation. The main weakness of the earlier work was the absence of algorithms for creating valid mesh topologies and adapting these topologies.

A full description of the GBBS would take up too much space so we present here only a brief summary of the salient points as they affect the algorithms. The reader is referred to the original papers for further details.

The GBBS is a powerful and elegant generalisation of the Biquadratic B-Spline. It automatically maintains G^1 continuity. The GBBS is defined by a set of M 3D control points $\mathbf{Q} = \{\mathbf{c}_m : m = 1..M\}$ together with connectivity information K. The connectivity defines a mesh topology which is restricted to 4-sided faces, (see for example figure 2). Thus the surface is defined by $S = (\mathbf{Q}, K)$. The connectivity information can be summarised in a function $f(i, j, k, l)$ which is equal to 1 for each set of vertices i, j, k, l connected up to a face ordered

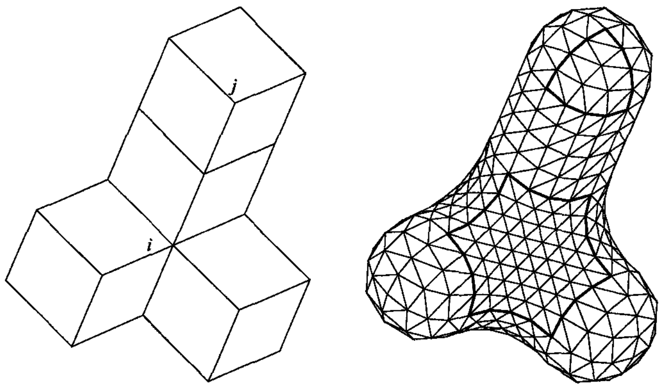

Fig. 2. A simple control mesh and the resulting spline surface (converted to triangles). Control point i generates the 6-sided patch and j generates the 3-sided patch.

anti-clockwise around an outward pointing normal. From this can be derived a function $e(i, j)$ equal 1 if i and j form an edge and 0 otherwise. It is also convenient to define $f(i, j)$ equal 1 if i and j are distinct but are part of the same face.

Each vertex gives rise to an n-sided surface patch where n is the number of 4-sided faces using that vertex. Patch m depends only on the $2n + 1$ element control vector $\mathbf{q}_m^\top = [c_m, c_k : f(m, k) = 1]$ consisting of c_m and the set of all vertices on adjacent faces, i.e. with k in the neighbourhood of m.

Previously we introduced a matrix-based scheme to compute the surface based on notation similar to that of [4]. The principal steps in computing the surface are as follows. The control vector $\mathbf{q}_m$ is converted to a vector of Bezier control points $\mathbf{r}_m$ by a matrix multiplication $\mathbf{r}_m = \mathbf{M}\mathbf{q}_m$ This is combined with a column vector containing all the Bezier polynomials $\mathbf{B}(p)$ to compute the point. Thus we obtain the surface patch S_m as a mapping from points $p = (u, v)$ contained in a regular n-gon domain D_n to a 3D surface

$$S_m = \{\mathbf{r}(p) | p \in D_n, \ \ \mathbf{r}(p) = \mathbf{B}^\top(p)\mathbf{M}\mathbf{q}_m\} \tag{1}$$

The whole surface S is the union of the patches S_m, $S = \bigcup_m S_m$. The control vector for patch m, $\mathbf{q}_m$ can be obtained from the vector of all control points $\mathbf{Q}^\top = [c_1..c_M]$ by a connectivity matrix $\mathbf{q}_m = G_m\mathbf{Q}$.

The simplest example of a Bezier polynomial is a Bezier curve [19,7]. When discussing Bezier curves it is useful to replace the usual single parameter u with two parameters $u_1 (= u)$ and u_2 and a constraint that $u_1 + u_2 = 1$. A depth d Bezier curve $C = \{\mathbf{r}(u_1, u_2) | \, u_1 \epsilon [0, 1], u_1 + u_2 = 1\}$ is defined in terms of $d + 1$ control points $\mathbf{r}_i$ and the Bernstein-Bezier polynomials $B_i^d(u_1, u_2)$ as follows

$$\mathbf{r}(u) = \mathbf{r}(u_1, u_2) = \sum_{i=0}^{d} \mathbf{r}_i B_i^d(u_1, u_2) = \sum_{i=0}^{d} \mathbf{r}_i \frac{d!}{i!(d-i)!} u_1^i u_2^{d-i} \tag{2}$$

The Bezier curve admits an elegant generalisation called a B-form that maps a $[(k+1)$-variate$]$ k-dimensional parameter space onto any number of scalars. Firstly we must define multivariate Bernstein-Bezier polynomials. For these we will need a notation for multi-indices $\mathbf{i} = \{i_1, i_2, ...i_{k+1}\}$. The symbol $\hat{e}_j$ denotes a multi-index whose components are all zero except for the j component which is 1. It is useful to define a modulus of a multi-index as $|\mathbf{i}| = i_1 + i_2 + ... + i_{k+1}$. The k-variate depth d Bernstein-Bezier polynomials are a simple generalisation of equation (2).

$$B^d_{\mathbf{i}}(u_1, u_2, ...u_{k+1}) = \frac{d!}{i_1!i_2!...i_{k+1}!} u_1^{i_1}, u_2^{i_2}, ...u_{k+1}^{i_{k+1}}, \qquad |\mathbf{i}| = d \qquad (3)$$

The Loop and De Rose scheme is based on S-patches, which are n-sided smooth patches which map a point $p = (u, v)$ inside n-sided domain polygon D to a 3D surface. Firstly we form n barycentric variables $l_i, i = 1..n$ defined as follows. Define the n vertices of the regular n-gon as p_i, $i = 1..n$. Define the fractional areas $\alpha_i(p)$ as the area of a triangle enclosed by points p, p_i, and p_{i+1} divided by the total area of D. Now form n new variables $\pi_i(p)$ by

$$\pi_i(p) = \alpha_1(p) \times ...\alpha_{i-2}(p)\alpha_{i+1}(p)...\alpha_n(p) \qquad (4)$$

Then form normalised variables $l_i(p)$

$$l_i(u, v) = l_i(p) = \frac{\pi_i(p)}{\pi_1(p) + ... + \pi_n(p)} \qquad (5)$$

The S-patch is now simply defined in terms of the variables $l_i(p)$ and the Bezier control points $\mathbf{r}_{\mathbf{i}}$. It is a mapping $S = \{\mathbf{r}(u, v) | (u, v) \epsilon D_n\}$ where D_n is a n-sided domain polygon and

$$\mathbf{r}(u, v) = \sum_{|\mathbf{i}|=d} \mathbf{r}_{\mathbf{i}} \, B^d_{\mathbf{i}}(l_1, l_2, ...l_n) \qquad (6)$$

Note that the n-sided patch uses a $k+1$ variate Bezier polynomial where $n = k+1$.

3.1 Computation

For details of computation of the matrix $\mathbf{M}$ the reader is referred to [20]. It contains constants so only needs to be computed once and stored. When repeated computation of a point p on a patch is required $\mathbf{B}^\top(p)\mathbf{M}$ may be pre-computed and stored. Then point evaluation consists of a weighted sum of the control points, and is very fast, $O(2n+1)$. This is typically what we use when rendering the surface. When an arbitrary point is required this can be slower for $n > 6$ because there are $\frac{(n+6)!}{6!(n+1)!}$ Bezier control points and in general we avoid patches with more than 6 sides.

4 Seeding

An important new result presented in this paper is a solution to the seeding problem. A valid slime surface must have a mesh of control points connected in a special way. The mesh must be made up of 4 sided faces and each non-boundary vertex must have 3 or more faces using it.

A precondition to adaptive meshing algorithms is a valid starting point. In our processing pipeline we indicated that our starting point is a triangular mesh. In our first paper [21] we suggested a method that would convert a triangular mesh to a mesh of four-sided faces. The idea was to subdivide each triangle into 3 four-sided faces. This was a valid solution, but not ideal, because it required the number of faces be increased by a factor of three.

A better option would be to group pairs of three-sided faces to form a four sided mesh with half the number of faces. This is a nontrivial problem because if even one triangle is unpaired the solution is of no use. It bears some superficial resemblance to the NP-hard problem of finding a Hamilton cycle on a graph.

The algorithm is now presented. It is based on region growing over the set of triangles t on the surface. Each triangle has edges e and the region boundary, denoted B is not allowed to self intersect. A pair of adjacent triangles is chosen as a seed and the seed boundary is passed to `RegionGrow`(B).

The region boundary is grown by adding a pair of adjacent triangles at a time. This operation can fail when the boundary self intersects and two non-connected regions each with an odd number of triangles are left outside the boundary. The algorithm backtracks from this point to the last boundary that contained no "elbows". By elbow we mean a part of the boundary where two adjacent boundary edges lie on the same triangle. The significance of this is that when a pair of triangles is grown there is no choice as to how the triangles are paired.

This is illustrated in figure 3 when the boundary encloses the shaded region and attempts to grow a pair of white triangles. If triangle 1 is added only triangle 2 can be paired with it. This is not the case for triangle 3 which can be paired with triangle 4 or 5.

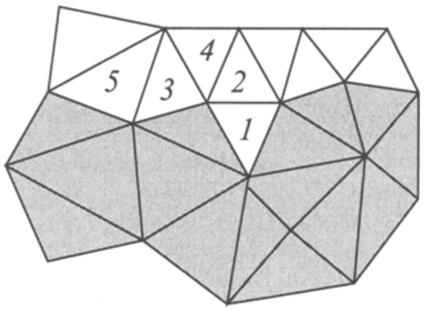

Fig. 3. The boundary contains an 'elbow' at triangle 1

The algorithm below is not guaranteed to succeed but has succeeded for all our data sets. A depth first recursive search is guaranteed to succeed but has worst case exponential complexity. The search presented here stores the last non-elbow boundary so it is a modification of a depth first search which only ever backtracks up one level. In practice, because there are many non-elbow boundaries, the algorithm is linear in the number of faces. Since the number of faces is potentially 10^6 this is welcome.

Pseudo code for the algorithm is presented below.

```
RegionGrow($B$) {
    $B' := B$
    repeat {
        for each $t_1$ on $B$ {
            for each $t_2$ adjacent $t_1$ outside $B$ {
              $B := B'$
              ForcedGrow($B, t_1, t_2$,pass)
              if (pass) exit loop over $t_2$ and $t_1$;
            }
        }
        if (not pass) report algorithm failed and exit.
        $B' := B$
    } until no more triangles to add.
}

ForcedGrow($B, t_1, t_2$,pass) {
    Add $t_1$ and $t_2$ to $B$.
    If $B$ self intersects set pass:=FALSE and return.
    If all triangles used up set pass:=TRUE and return.
    while ($B$ contains elbow) {
        Add next two triangles to $B$ at elbow.
        If $B$ self intersects set pass:=FALSE and return.
        If all triangles used up set pass:=TRUE and return.
    }
    set pass:=TRUE and return.
}
```

Finally we show a sample output from the algorithm in figure 4. F 4(a) shows the input triangulated surface which is paired and the 4-sided mesh is shown in (b). This is a valid GBBS control mesh.

4.1 Limitations

The algorithm presented in this section has not been tested for open or closed surfaces with holes. This is because a region growing algorithm needs to have more that one boundary on such a surface. The algorithm will need to be extended for such surfaces.

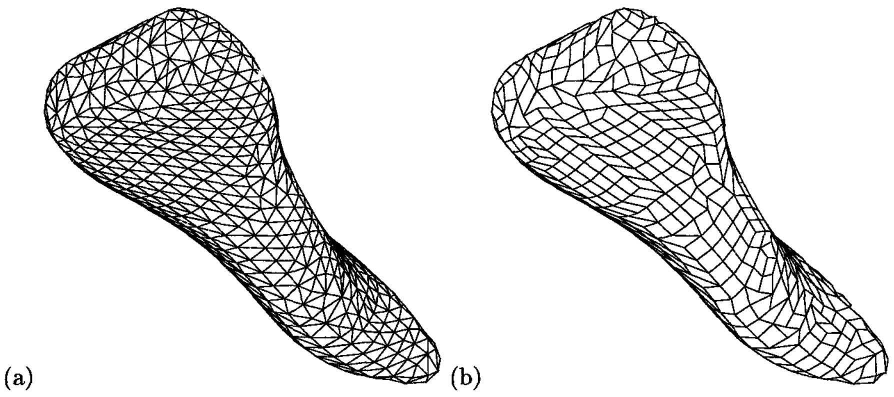

Fig. 4. Results from the seeding algorithm, (a) is the input mesh and (b) is a valid spline control mesh.

5 The energy function

Our approach to reconstructing the surface is similar to that of Hoppe [12]. Hoppe provided a comprehensive scheme that worked for piecewise flat surfaces. We have succeeded in generalising this approach to the case of GBBS surfaces.

The surface S is defined by a set of control points $\mathbf{Q}$ and a mesh topology K, i.e. $S = (K, \mathbf{Q})$. The goal of our spline optimisation is to find a surface which is a good fit to the point set $X = \{x_1..x_P\}$ with a small number of vertices. Thus we wish to minimise the energy function

$$E(K,\mathbf{Q}) = E_{dist}(K,\mathbf{Q}) + E_{rep}(K) + E_{spring}(K,\mathbf{Q}) \tag{7}$$

The first term depends on the quality of the fit to the point set. It is equal to the sum of the squared distances from the points $X = \{x_1..x_P\}$ to the surface.

$$E_{dist}(K,\mathbf{Q}) = \sum_{i=1}^{P} d^2(x_i, S(K,\mathbf{Q})) \tag{8}$$

The representation energy penalises meshes with many vertices. There are M vertices so

$$E_{rep}(K) = k_{rep}M \tag{9}$$

A regularisation term is needed during fitting (because the problem is under-constrained when no data points lie on a patch) and we use a simple spring-like term

$$E_{spring}(K,\mathbf{Q}) = k_{spring} \sum_{e(j,k)=1} |c_i - c_j|^2 \tag{10}$$

This term may be reduced to zero in the final stages of the optimisation, and so it need have no effect on the result. In particular it need not cause any smoothing of the surface.

6 Optimisation of the energy function

6.1 Fixed patch coordinates and fixed topology

We start by considering a simple case for optimisation. We consider only a single patch, S_m and those data points $\{x'_1..x'_R\}$ for which patch m is the closest patch. We assume that the closest point on patch m to point x'_i is $\mathbf{r}(p_i)$ with parametric coordinates p_i. Therefore we wish to optimise the energy function

$$E(\mathbf{q}) = \sum_{i=1}^{R} |x'_i - \mathbf{r}(p_i)|^2 + \sum_{e(i,j)=1} |c_i - c_j|^2 \tag{11}$$

with respect to the position of the patch control points $\mathbf{q}_m$. It is helpful to note that

$$\mathbf{r}(p_i) = \sum_{j:c_j \in \mathbf{q}_m} w_j(p_i)\mathbf{c}_j \tag{12}$$

where the weighting factors $w_j(p_i) = (\mathbf{B}^{\mathrm{T}}(p_i)\mathbf{M})_j$ are fixed numbers adding up to 1. This problem may be formulated as a matrix minimisation problem of the form $|Av - d|^2$ which can be solved rapidly. The column vector v is formed from the control points for the patch. The first R rows of the matrix A contain the weights $w_j(p_i)$ so that multiplication of row i with column vector v results in $\mathbf{r}(p_i)$. Correspondingly the first R rows in column vector d contain the data points x'_i.

The spring terms are attached along the edges of control mesh faces. For each edge there is another row of A and d. The row in d contains zero and the row of A contains a $\sqrt{k_{spring}}$ in column i and $-\sqrt{k_{spring}}$ in column j. It is easy to verify that $E(\mathbf{q})$ from equation (11)

$$E(\mathbf{q}) = |Av - d|^2 \tag{13}$$

It is worth noting that the above formulation is based on the column vectors v and d containing 3D vectors, but in fact it separates into 3 independent matrix equations to be solved for the x, y and z components. We have shown how the energy can be be reduced to a matrix equation for one patch, and the same procedure can easily be applied to generate a matrix equation for the whole mesh.

The matrix for the whole mesh is large but sparse. Such least square problems may be solved efficiently using the conjugate gradient method [8]. If we consider only one patch and fix all vertices except the central vertex then the problem reduces to 3 *quadratic* equations with straightforward analytic solutions.

6.2 Variable patch coordinates

The true cost in equation (11) depends on the distance to the closest point which varies as we vary the control points. We solve this iteratively by finding the closest point, then optimising over the control points and repeating the process until convergence. An attractive feature of the process is that the closest point step and the control point minimisation step both decrease the energy monotonically.

6.3 Variable mesh topology

Finally we wish to optimise the full energy function search over control point positions and mesh topologies. This is potentially a computationally expensive task especially if we aim to find a global optimum. However we can do a quite adequate job by local search techniques which can find a good local minimum.

Firstly we examine how mesh topology is allowed to change. The scheme used by Hoppe for triangles is reviewed in figure 5. It consists of 3 simple operations performed on edge $\{i, j\}$. It can be collapsed to a single vertex, split into two or swapped. It is worth noting that there are some conditions under which the

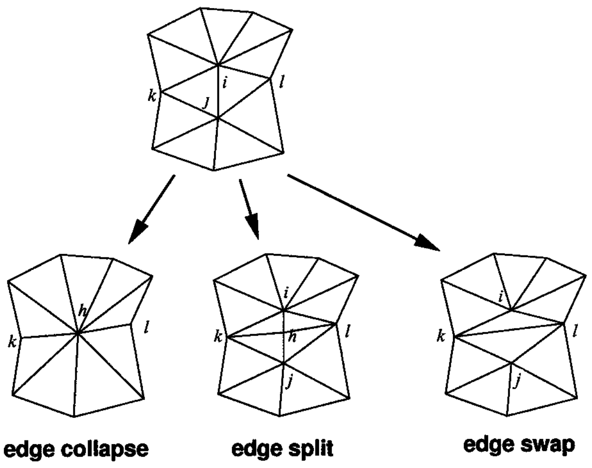

Fig. 5. Topology editing operations for triangular meshes

edge collapse operation is not allowed and these are detailed in [12].

In the case of our mesh edge collapse is not an allowed operation since it can reduce 4-sided faces to 3-sided faces. Instead we use the operation of face collapse and its inverse face creation as shown in figure 6. We have not yet determined what conditions must be satisfied before face collapse is allowed, however we disallow face collapse when it results in a vertex used by 2 or fewer faces.

6.4 Closest Point Computation

The optimisation over control points is relatively quick, and the complexity of the computation is dominated by the nearest neighbour step. This is mirrored

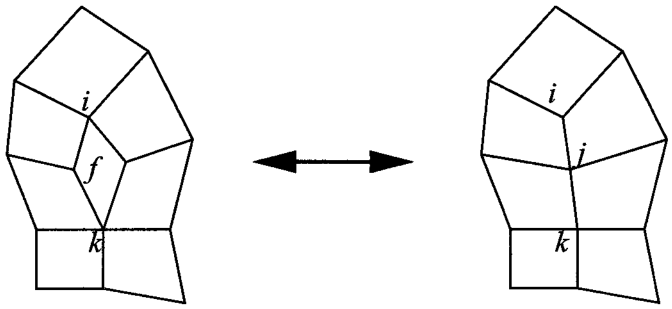

Fig. 6. Topology editing operations for 4-sided face meshes

in other problems such as surface registration by the iterated closest point algorithm [2] and also some formulations of the surface fusion problem.

The general closest point to point set problem can be solved in $O(N \log N)$ by use of appropriate spatial partitioning data structures. By encoding triangles into such a structure one can be guaranteed of finding all triangles within a threshold distance. Following this a routine for closest point to triangle is required, and it is worthwhile carefully optimising this routine.

Finding the closest point to a spline is slightly more computationally intensive. Each patch may be approximated to within a threshold by a piecewise planar triangular mesh according to a tessellation method of [13], see page 262. The nearest point to triangle routine may then be used. By decreasing the triangle size a very good approximation to the closest point may be found. In this way the closest point to spline can be found in less than 10 closest point to triangle operations.

In the first iteration the closest point search is performed over the entire mesh. Subsequent searches can be performed on a purely local basis, while the distance to the surface lies within a threshold.

6.5 Overall strategy

Our starting point is a detailed mesh and point set. A global search assigns each point to a triangle. Initially we proceed with a triangle optimisation scheme until the number of triangles has been reduced. This is mainly because the spline method is slower by about a factor of ten, so it saves time.

Then the seeding algorithm is applied to convert the triangular surface to a spline surface. Firstly all vertices are optimised followed by recomputing the closest point. These steps are iterated until convergence. Then local face collapse operations are performed. A face collapse is performed and the central vertex is optimised over position followed by a closest point computation for a few iterations. If the energy has been lowered the collapse is accepted, if not it is rejected.

The faces are sorted into ascending size and this forms a queue to be processed. Faces that fail to collapse are marked. When no faces can collapse the algorithm terminates.

We have not yet tested the face creation operation so we do not know if it can substantially improve the fit.

7 Results

We now present results for the foot dataset. The original surface is shown in figure 7 (a). A point set X is created by uniformly random sampling the original surface with 4000 points. We decimate to a triangular surface containing 118 faces. This is shown rendered in figure 7 (b) and also in figure 7 (d). The spline fit contains 59 faces (61 patches) and is shown rendered in 7 (c), the control mesh is shown in figure 7 (e). The rms distance from the point set may be computed.

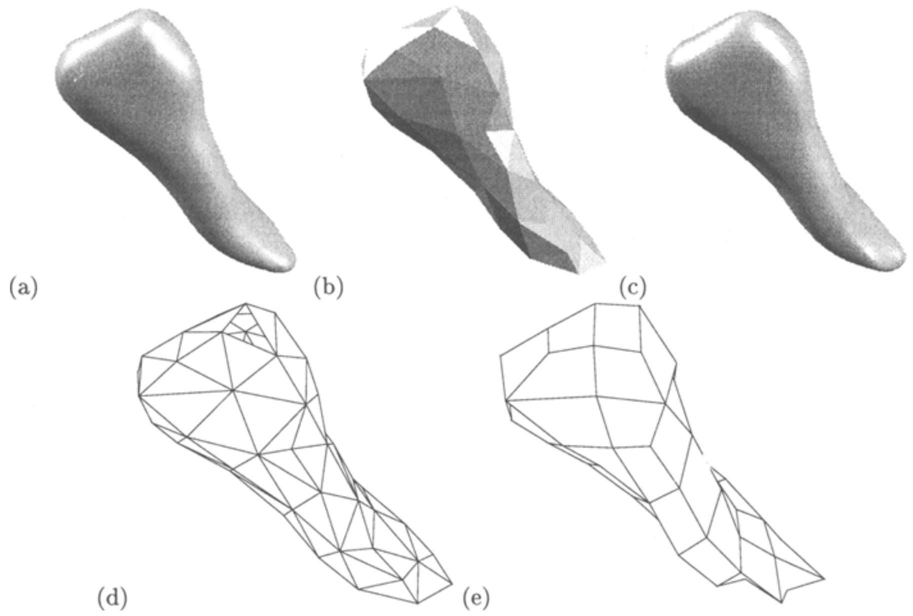

Fig. 7. Surface optimisation applied to the foot (a) original surface (b) best fit with 118 triangles - flat rendered (c) best fit with 59 spline patches - flat rendered (d) best fit with 118 triangles - line drawing (e) best fit with 59 spline patches - control mesh

The foot is firstly scaled to a unit cube. The triangular fit is 0.35% of the cube edge length and the spline fit is 0.18%. This is an improvement of a factor 2. A more dramatic improvement is to be expected in higher order derivatives such as the normal or curvature. This is apparent from the flat rendered versions in figure 7.

8 Conclusions

We have now provided a powerful new representation which can be used in a variety of applications in computer vision. We have previously developed a matrix formalism for easy algebraic manipulation in the same form as [4] and fast techniques for computing points on the spline. The matrices used for convenient computation of the GBBS surface have been made available on the Web [1].

In this paper we have developed a scheme for seeding the surface and adaptively remeshing the control points. An optimisation approach provides the framework for driving the adaptive meshing.

9 Future work

At present we can fit point sets of size 5000 in minutes on a workstation. We intend to optimise the code with the objective of dealing with point sets of size 500 000 in less than 30 minutes cpu time, followed by more detailed characterisation of the gains in accuracy over a number of data sets.

Extensions of the software are necessary to deal with open surfaces and internal crease edges.

10 Acknowledgements

This work was carried out as part of EPSRC Research Grants, GR/K04569 and GR/K91255. Both datasets were taken on a Cyberware scanner, the bunny is available on the Cyberware web site and the foot dataset was supplied by Tim McInerney.

References

1. A. J. Stoddart, Matrix data on the Web, `http://www.ee.surrey.ac.uk/showstaff?A.Stoddart`.
2. P. Besl and N. McKay. A method for registration of 3D shapes. *IEEE Trans. Pattern Analysis and Machine Intell.*, 14(2):239–256, 1992.
3. V. Casselles, R. Kimmel, and G. Sapiro. Geodesic active contours. In *5th Int. Conference on Computer Vision*, pages 694–699, Cambridge, Massachusetts, 1995.
4. R. Curwen and A. Blake. Dynamic contours: Real-time active contours. In *Active Vision*, pages 39–57, MIT Press, Cambridge, Mass, 1992.
5. D. DeCarlo and D. Metaxas. Adaptive shape evolution using blending. In *5th Int. Conference on Computer Vision*, pages 834–839, Cambridge, Massachusetts, 1995.
6. M. Eck and H. Hoppe. Automatic reconstruction of b-spline surfaces of arbitrary topological type. In *SIGGRAPH*, pages 325–334, New Orleans, 1996.
7. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design.* Academic Press, Boston, 1990.
8. G. Golub and C. V. Loan. *Matrix Computations.* John Hopkins University Press, 1989.

9. A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Marching triangles: range image fusion for complex object modelling. In *1996 Int. Conference on Image Processing*, pages II381–384, Lausanne, Switzerland, 1996.
10. A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *Fourth European Conference on Computer Vision*, pages 117–126, Cambridge, U.K., 1996.
11. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH*, pages 71–78, 1992.
12. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH*, pages 19–25, 1993.
13. D. Kirk. *Graphics Gems*. Academic Press, London, U.K., 1992.
14. V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH*, pages 313–324, New Orleans, 1996.
15. C. T. Loop and T. D. DeRose. A multisided generalization of bezier surfaces. *ACM Trans. on Graphics*, 8(3):204–234, 1989.
16. C. T. Loop and T. D. DeRose. Generalized b-spline surfaces of arbitrary topology. *ACM Computer Graphics*, 24(4):347–356, 1990.
17. T. McInerney and D. Topologically adaptable snakes. In *5th Int. Conference on Computer Vision*, pages 840–845, Cambridge, Massachusetts, 1995.
18. T. McInerney and D. Terzopoulos. Deformable models in medical image analysis. *Medical Image Analysis*, 1(2):91–108, 1996.
19. M. E. Mortenson. *Geometric Modeling*. John Wiley and Sons, New York, 1985.
20. A. Saminathan, A. J. Stoddart, A. Hilton, and J. Illingworth. Progress in arbitrary topology deformable surfaces. In *British Machine Vision Conference*, pages 679–688, Colchester, England, 1997.
21. A. J. Stoddart, A. Hilton, and J. Illingworth. Slime: A new deformable surface. In *British Machine Vision Conference*, pages 285–294, York, England, 1994.