UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

The Definition, Editing, and Contouring of Surfaces for the Analysis of Field Problems

Robert R. Dickinson,
Richard H. Bartels

CS-86-54

November, 1986

UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

The Definition, Editing, and Contouring of Surfaces for the Analysis of Field Problems

Robert R. Dickinson,

University of Waterloo
Department of Systems Design
Waterloo, Ontario
Canada N2L 3G1
(decvax!watmath!watcgl!rdickins)

Richard H. Bartels

University of Waterloo
Computer Graphics Laboratory
Department of Computer Science
Waterloo, Ontario
Canada N2L 3G1
(decvax!watmath!watcgl!rhbartels)

ABSTRACT

This paper reports on an interactive system for manipulating a tensor-product B-spline approximation to field data for applications in which contours are of interest. The features of the system are: an interpolation technique for approximating fields defined from scattered or gridded data by tensor-product B-splines, an interactive display providing control-vertex manipulation of the resulting B-spline approximation, and a contouring algorithm that is designed specifically for B-spline surfaces.

This research was supported under grant number A4076 by the Natural Sciences and Engineering Research Council of Canada, and by the Atmospheric Environment Service of Environment Canada under DSS contract number 02SE.KM147-5-6136.
Keywords: field data, contouring, interactive surface editing, tensor-product B-splines.

1. INTRODUCTION

This paper outlines an approach to the definition and interactive editing of spline approximations to functions of two variables. The techniques described are intended for use in the analysis of field problems in science and engineering. In particular, these techniques have been developed for the input and editing of meteorological data with a view to implementation in operational meteorology environments, but these techniques could be applied to many other field problems.

Input to software for the analysis of field problems is often provided as point information, but the users of such software can more easily relate to analogue information such as contour maps or streamline charts. For this reason we have developed a system to perform (1) a translation between point data and an internal functional representation of a field and (2) a translation between this internal representation and contours to be displayed. In some contexts of field analysis; e.g. in the preparation of meteorological charts for forecasting, editing of the field is legitimate to adjust, for example, for unknown data, spurious values, updated information, or human experience. Accordingly, our system's display provides the user with mechanisms by which the internal representation of the field can be directly modified, thereby modifying the contour display. The internal representation can be sampled for subsequent processing by other systems, if point data is required. The internal representation we have chosen, because of the ease with which it can be interactively manipulated, is that of a tensor-product B-spline surface. Its initial definition is through interpolation, which can be based on either scattered or gridded data.

The paper is structured as follows. Section 2 outlines our approach to the definition of a B-spline surface from a given set of scattered field values. Section 3 outlines our approach to tracking contours. Some aspects relating to the implementation of these concepts in an interactive environment are also discussed. Section 4 gives a brief overview of the experimental software system that has been developed to date. The paper concludes in section 5 with an overview of some aspects requiring further study.

2. DEFINITION OF AN INITIAL SURFACE

Various methods for defining an initial field have been developed based on either gridded or scattered data; the material in [Boehm84] and [Sablonnière85] will provide an introduction and references for this material. Our own decision to use tensor-product B-splines was based upon the ease by which local areas of the surfaces they define can be adjusted through simple manipulations of control vertices [Bartels87]. This means that we chose to deal with a parametric surface representation of the field. In our present system the initial B-spline surface is defined by interpolation, though other forms of approximation could be used. Methods for interpolating data given at the nodes of a rectangular grid using tensor-product B-splines are well known [deBoor78], so only the interpolation of scattered data is discussed here.

In order to investigate the fitting of field data in as flexible a manner as possible, we decided to provide for non-uniform knot spacing and various orders of splines. Consequently, our presentation here deals with general B-splines, and the sort of surface under consideration as an approximation to a bivariate field is

$$f\ (u\ ,v\ ) = \sum_{i\ ,j} V_{i\ ,j}\, B_{i\ ,k}\,(u\ )B_{j\ ,l}\,(v\ )\ \ ,$$

which defines a single-valued function of u and v in terms of a two-dimensional array of control vertices V_{ij} and a tensor-product B-spline basis of order $k \times l$. It is straightforward to provide for multiple field parameters by using multiple sets of control vertices.

To define a surface, given knot sequences

$$u_0, \ldots, u_{m+k} \quad \text{and} \quad v_0, \ldots, v_{n+l}$$

suitable for k-th and l-th order splines respectively, we form one equation in $(m+1)\times(n+1)$ unknown control vertices for each specified data triple

$$u^{(p)}\ ,\ v^{(p)}\ ,\ \theta^{(p)}$$

as follows:

$$\sum_{i=0}^{m} \sum_{j=0}^{n} V_{i,j} \, B_{i,k}(u^{(p)}) B_{j,k}(v^{(p)}) = \theta^{(p)} \quad .$$

The resulting system of linear equations can be solved using orthogonal factorization. Briefly, if $p = 1, \ldots, P$ and if $Q = (m+1)\times(n+1)$, then the collection of equations will form a $P \times Q$ system in the unknown control vertices $V_{i,j}$. In matrix format

$$\boldsymbol{B} \; \boldsymbol{V} = \Theta \quad ,$$

where $\boldsymbol{B}$ contains the $P \times Q$ entries $\sum B_{i,k}(u^{(p)}) B_{j,l}(v^{(p)})$, Θ is the column vector of field values $\theta^{(p)}$ to be interpolated, and $\boldsymbol{V}$ is the column vector of unknown control vertices. In order to attempt interpolation, rather than another form of approximation, this system of equations should not be overdetermined; that is, we should arrange that $P \leq Q$. In this case, the matrix $\boldsymbol{B}$ can be decomposed into the product of an orthogonal matrix $\boldsymbol{Q}$ and a lower triangular matrix $\boldsymbol{L}$ bordered by zero entries,

$$\boldsymbol{B} = [\, \boldsymbol{L} \; , 0 \,] \, \boldsymbol{Q} \quad .$$

Algorithms to achieve this are given in [Lawson74]. The system of equations becomes

$$[\, \boldsymbol{L} \; , 0 \,] \, \boldsymbol{Q} \; \boldsymbol{V} = \Theta \quad .$$

If $\boldsymbol{Q} \; \boldsymbol{V}$ is taken to be a vector $\boldsymbol{W}$, then $\boldsymbol{W}$ can be specified by solving the system

$$\boldsymbol{L} \; \overline{\boldsymbol{W}} = \Theta$$

for $\overline{\boldsymbol{W}}$, the first P components of $\boldsymbol{W}$. By setting the remaining $Q - P$ components of $\boldsymbol{W}$ to zero, and by letting

$$\boldsymbol{V} = \boldsymbol{Q}^{T} \; \boldsymbol{W} \quad ,$$

where $\boldsymbol{Q}^{T}$ stands for the transpose of the matrix $\boldsymbol{Q}$, the result is a set of control vertices that will define an interpolating surface.

It is advisable that, within any u,v –rectangle formed by k successive knots in the u direction and l successive knots in the v direction, there be some $(u^{(p)},v^{(p)})$–sampled field value, $\theta^{(p)}$. This is a condition ensuring that no column of zero entries will

occur in $\boldsymbol{B}$. In the case of gridded data this will result in $\boldsymbol{B}$ having full rank when $P = Q$; see [deBoor78] for a complete discussion. The routines in [Lawson74] are designed to detect and, generally, compensate for deficiency in the numerical rank of $\boldsymbol{B}$. The penalty to be paid for solving systems of deficient rank using these routines is the possible failure to interpolate up to as many data points as the nullity of $\boldsymbol{B}$.

3. COMPUTING AND DISPLAYING CONTOURS

Various algorithms for contouring surfaces described by field values at the nodes of triangular, quadrilateral, and latitude/longitude meshes have been available for some time; examples include meteorological data [Kulikov82] , geological data [Moore77] , arbitrary data [McLain1974] , finite element meshes [Lyness83] , and contouring 2-variable functions [Snyder78, Sutcliffe76] . These are based on low-order approximations to the surface at each facet of the mesh. A limitation of some of these algorithms is that the user control of the trade-off between image quality and computational effort is relatively clumsy. Others have a limited adaptability to changing field gradients; to obtain "high quality" contours a fine mesh is typically required.

More recently, higher-order contouring schemes have been developed in [Preusser84, Preusser84a] , producing good results within each mesh element but displaying sudden changes in contour curvature across element boundaries. An efficient scheme for plotting arbitrary C^2 continuous functions was recently developed [Suffern84] that made good use of the osculating circle geometry of the contour passing through a given point.

None of the foregoing work was directly suitable for our purposes, though we have made use of some concepts presented in the last-cited reference. We wanted to be able to edit local areas of field surfaces, which suggested the use of tensor-product B-spline approximations. To provide generality, we wished to be able to contour B-spline surfaces of any order and corresponding continuity. In the most general context, we would like to be able to handle arbitrary discontinuities associated with corresponding knot multiplicities. Accordingly, we felt that the best results would be

given by designing contouring algorithms specifically geared to surfaces composed of piecewise-bivariate polynomial patches.

Our method of contouring proceeds patch by patch over the surface. The approach we have adopted consists of four stages. The first stage prepares a table of power-series coefficients for use in computing bivariate polynomial values and derivatives within a selected patch. The second stage locates the contours that cross the boundary of the patch. The third stage uses Taylor expansion and Newton correction to compute each point in a stream of points located along a contour and within some user-specified function-value tolerance. The fourth stage uses each such stream of points to determine the display of the corresponding contour. We begin the detailed discussion with the crux of the algorithm; the computation of a stream of points located along a contour.

Taylor expansion

A given patch can be considered as an arbitrary bivariate polynomial $f\ (u\ ,v\ )$ of order $k \times l$ restricted to the rectangular domain defined by adjacent knots in u and v, meaning that u is in some knot interval $[u_i\ ,u_{i+1}]$ and v is in some knot interval $[v_j\ ,v_{j+1}]$. Contour tracking begins at some $u\ ,v$ location on the patch boundary that is known to be "on" a given contour; i.e. the value $f\ (u\ ,v\ )$ is within a specified tolerance of the value C for the given contour. In the Taylor expansion of $f\ (u\ ,v\ )$, terms involving partial derivatives of order k or higher in u and l or higher in v are zero, so that we have the exact equality

$$\begin{aligned} f\ (u+\delta\ ,\ v+\delta v\ ) & \\ &= f\ (u\ ,v\ ) + \delta u \frac{\partial f}{\partial u} + \delta v \frac{\partial f}{\partial v} \\ &+ \cdots \\ &+ \frac{\delta u^{k-1} \delta v^{l-1}}{(k-1)!\ (l-1)!} \frac{\partial^{k+l-2} f}{\partial u^{k-1} \partial v^{l-1}} \end{aligned} \tag{1}$$

For contouring we can simply set

$$f\ (u+\delta u\ \ ,\ v+\delta v\ ) = f\ (u\ ,v\ )$$

to yield:

$$\delta u \frac{\partial f}{\partial u} + \delta v \frac{\partial f}{\partial v} + \frac{\delta u^2}{2} \frac{\partial^2 f}{\partial u^2} + \cdots \tag{2}$$
$$= \phi_{(2)}(\delta u, \delta v) = 0$$

We will track a contour by piecewise linear steps, and the step length will be:

$$\delta u^2 + \delta v^2 - length^2 = \phi_{(3)}(\delta u, \delta v) = 0 \tag{3}$$

Hence, contour tracking amounts to the simultaneous solution of equations (2) and (3) for the unknown values δu and δv that define the next point

$$u + \delta u \ , v + \delta v$$

in the contour stream, in brief:

$$\phi_{(2)}(\delta u, \delta v) = 0$$
$$\phi_{(3)}(\delta u, \delta v) = 0 \ .$$

These two equations also provide a natural basis for adapting computational effort to field gradients to achieve a specified image quality. That is, *length* can be adjusted at each step based upon local measures of contour curvature.

As long as the step *length* is small relative to the local gradients, we can easily obtain an approximate solution to $\delta u, \delta v$, and solve via Newton's method. It should be emphasized here that we are only interested in one particular root, for which we can arrange to have a good initial guess, as will be discussed in the next subsection. Symbolically inverting the Jacobian, we obtain the following iterative scheme for finding the particular solution to these two equations for which we have a good initial guess:

$$\begin{bmatrix} \delta u \\ \delta v \end{bmatrix}^{K+1} = \left([\mathbf{J}]^K \right)^{-1} \begin{bmatrix} \phi_{(2)} \\ \phi_{(3)} \end{bmatrix}^K \tag{4}$$

where

$$[\mathbf{J}]^K = \begin{bmatrix} J_{1,1} & J_{1,2} \\ J_{2,1} & J_{2,2} \end{bmatrix}^K = \begin{bmatrix} \frac{\partial \phi_{(2)}}{\partial(\delta u)} & \frac{\partial \phi_{(2)}}{\partial(\delta v)} \\ \frac{\partial \phi_{(3)}}{\partial(\delta u)} & \frac{\partial \phi_{(3)}}{\partial(\delta v)} \end{bmatrix}^K .$$

The second row of $[\mathbf{J}]^K$ is easily computed as $J_{2,1} = 2\,\delta u$ and $J_{2,2} = 2\,\delta v$. At each step along a contour, f and all its partial derivatives are evaluated once at the current point, u, v, and used to compute the coefficients of the variables δu and δv in (2). At each iteration in (4), $J_{1,1}$, $J_{1,2}$ and $\phi_{(2)}$ can be computed from these coefficients.

Computing an initial guess

The Newton iteration demands an initial guess at the solution,

$$\begin{bmatrix} \delta u \\ \delta v \end{bmatrix}^0 .$$

To compute a second-order guess, we use the osculating circle to the current point, (u, v), on the current contour of f. The radius of the osculating circle at (u, v) is [Suffern84]:

$$R = (f_u^2 + f_v^2)^{\frac{3}{2}} \times \left| [f_{uu} f_v^2 - 2 f_u f_v f_{uv} + f_u^2 f_{vv}] \right| .$$

The centre of the circle is located in the direction of the curvature vector, which is normal to the contour at u, v. The direction of the tangent vector at (u, v) is

$$\psi = artan\left(-\frac{f_u}{f_v}\right) .$$

The rate of change of direction of the tangent vector along the curve at (u, v) is given by:

$$\dot{\psi} = \frac{-(f_{uu} f_v^2 - 2 f_u f_v f_{uv} + f_u^2 f_{vv})}{(f_u^2 + f_v^2)^{\frac{3}{2}}} \quad .$$

From this background, the centre of the osculating circle can be located as:

$$u_c = u + \sigma R \sin(\psi)$$

$$v_c = v + \sigma R \cos(\psi)$$

where $\sigma = \text{sign}(\psi)\text{sign}(\dot{\psi})\text{sign}(f_u) = \pm 1$ and sign(0) is defined as 1. It is noted that as $f_v \rightarrow 0$, $\psi \rightarrow \pm\frac{\pi}{2}$. Under this condition we can compute the centre of the circle from

$$u_c = u + \sigma R \sin(\psi) \quad , \quad v_c = v \quad ,$$

where $\sigma = \text{sign}(\dot{\psi})\text{sign}(f_u)$.

To compute a particular point on the circle, the initial guess vector is now given as:

$$\delta u^0 = \sigma R \, [\sin(\psi) - \sin(\psi + sense \cdot \delta\psi)]$$

$$\delta v^0 = \sigma R \, [\cos(\psi) - \cos(\psi + sense \cdot \delta\psi)] \, .$$

where $sense = \pm 1$ for right handed and left handed contouring respectively. If $\delta\psi$ is sufficiently small, tabulated approximations for the sine and cosine functions can be used.

It is desirable to provide the user with a simple way of controlling the image quality while maintaining the natural adaptive attributes of the contouring algorithm. It appears that a simple bi-linear approximation of the form

$$\delta\psi = \frac{0.7 \, l_{nom}}{R} + 0.17 \quad , \quad \frac{l_{nom}}{R} \leq 1.0$$

$$\delta\psi = 0.87 \quad , \quad \frac{l_{nom}}{R} > 1.0$$

where $\delta\psi$ is in radians, l_{nom} is a user-specified "nominal segment length", and R is the radius of curvature, provides a fast way of maintaining an almost uniform maximum deviation between piecewise linear approximations of contours and actual contour curves.

Convergence and Correction

A suitable convergence criterion for iteration (4) is given by checking only whether $\phi_{(2)} \rightarrow 0$, and ignoring $\phi_{(3)}$. This is because the segment length defined by a given solution is not critical, but the tolerance on equation (2) determines the accuracy with respect to the current contour value.

It should be pointed out that equations (2) and (3) are exact, since a full Taylor expansion is being used. Errors arise, however, from the termination of the Newton iteration after a finite number of steps and from the given tolerance that specifies acceptable deviation from the contour. The effect of these numerical errors is easily checked by evaluating f at the new point $(u+\delta u, v+\delta v)$. A correction can be made when generating each new segment by using the same procedure as described above, except for adding c_0 to the left-hand-side of (2) where $c_0 = C - f(u, v)$. C is the current contour value, and (u, v) are the coordinates of the current starting point. Hence the algorithm can be designed to be continuously self-adjusting.

Display

Various curve-interpolation or curve-approximation techniques could be used to render the stream of points produced along a contour independently of the algorithm that produces the points. The simplest technique is, of course, linear interpolation between adjacent points of the stream, and it is this technique that we have used to produce the figures included in this paper.

Repetitive evaluations within a given patch

In practice, the implementation of the above techniques involves a great many function and derivative evaluations within a given patch, and we would like to be able to perform these as quickly as possible. To achieve this, suggestions given in [deBoor78] have been followed by setting up tables of power series coefficients for a given patch prior to entering the contouring procedure itself. These can then be used in a bivariate version of Horner's rule to compute the function and partial derivatives values. Specifically, an order $k \times l$ bivariate polynomial in u and v,

$$f\ (u\ ,v\ ) = \sum_{i=0}^{k} \sum_{j=0}^{l} c_{i\ ,j}\, u^{i}\, v^{j} \quad ,$$

can be re-arranged as

$$f\ (u\ ,v\ ) = \sum_{i=0}^{k} \left\{ \sum_{j=0}^{l} c_{i\ ,j}\, v^{j} \right\} u^{i} \quad .$$

For a given v, the term in braces can be evaluated for each i using Horner's rule. The resulting $k+1$ numbers serve as coefficients for evaluations of $f\ (u\ ,v\ )$, using Horner's rule again, for a given u. Partial derivatives are related to this formulation; e.g.,

$$\frac{\partial f}{\partial v} = \sum_{i=0}^{k} \left\{ \sum_{j=1}^{l} (l-j)\times c_{i\ ,j}\, v^{j} \right\} u^{i}$$

is the coefficient of $\delta u^{0} \delta v^{1}$ in (2). The l coefficients of the form $(l-j)\times c_{i\ ,j}$, and so on for higher-order partials, are constant for a given patch. For simplicity, the coefficients $c_{i\ ,j}$, $(l-j)\times c_{i\ ,j}$, and so on can be stored in a 4-dimensional table of the form $c_{i\ ,j\ ,du\ ,dv}$.

The overhead involved in allocating space for and calculating the table of coefficients pays off as long as there are a large number of evaluations to perform within a given patch, which is precisely the situation in which we find ourselves when tracking a contour.

Finding the first point of a contour

As was mentioned above, the contouring algorithm begins at some known point on a given contour at the boundary of a patch. This sub-section discusses ways of finding such a point.

Within the bounds of a given patch, each contour must fit into one of the following categories:

1) an open contour: one that intersects the boundary at two distinct points;

or

2) a closed contour: one that does not intersect the boundary at all.

Our present approach assumes that all contours have been put into the first category by suitable refinement of the patches; e.g., by a suitable application of the "Oslo Algorithm" [Cohen80] as a preprocessing step.

We follow the surface profile along a patch boundary, stepping from each contour value C to the next higher or lower value using an incremental, root-finding process. Along each boundary we have a Taylor-series expansion analogous to that for the above contouring discussion. From a given point along either a u or v boundary, the intersection of the contour with value C is a positive distance δu or δv away, as follows, where the parameter t can be either u or v and the order h can be either k or l :

$$f(t+\delta t) = f^{0}(t) + \delta t\ f^{1}(t) + \frac{\delta t^{2}}{2} f^{2}(t) + \ldots + \frac{\delta t^{h-1}}{(h-1)!} f^{h-1}(t) \tag{5}$$

or

$$-C + f^{0}(t) + \delta t\ f^{1}(t) + \frac{\delta t^{2}}{2} f^{2}(t) + \ldots + \frac{\delta t^{h-1}}{(h-1)!} f^{h-1}(t) = 0\ , \tag{6}$$

which is a function, $g(\delta t)$, in the unknown that we want to find.

Using Newton's Method again, we obtain the following iterative scheme for finding δt .

$$[\delta t]^{K} = [\delta t]^{K-1} - \frac{[g(\delta t)]^{K-1}}{[g'(\delta t)]^{K-1}} \tag{7}$$

The initial guess for δt is given by the symbolic solution of a first order approximation of (6):

$$[\delta t]^{0} = \frac{C - f^{0}(t)}{f^{1}(t)}\ . \tag{8}$$

4. SOFTWARE OVERVIEW

This section gives a brief overview of the design of the experimental software system. Feasibility tests with respect to response times, using the above algorithms, indicate that contour drawing is feasible on fast hardware (e.g. VAX-level or better) at nearly refresh rates (30 Hz or better). Contour drawing on a typical microcomputer-based workstation is sufficiently slow, however, to require some special design considerations. Where approximate times are given below, they are stated for a Silicon Graphics IRIS 2400 (MC 68010) graphics workstation with a floating-point accelerator.

The interpolation sub-system

For the present experimental purposes, all data storage required by the interpolation sub-system is dynamically allocated, except for the display list names of the graphical icons representing the scattered data points. In this way, the original scattered data points can be instantaneously displayed at any time, upon the request of the user. Provision is made for the automatic verification of the accuracy with which the original data is interpolated.

The interpolation sub-system is designed to be used once only, before editing commences, and is independent of the contouring sub-system. It takes as input; (1) a B-spline basis given by the parameters k ,l ,m ,n , and knots sequences of length $(m+k+1)$ in the u direction and $(n+l+1)$ in the v direction; and (2) scattered data point locations and a corresponding array of field values. Figures 1 to 3 show various representations of the same surface. In these figures, the B-spline basis was specified as $k=l=3$ (biquadratic) and $m=n=5$, with knots uniformly spaced to provide a simple test case. The number of data points is 34. The interpolation step itself took about 6 seconds for this test case.

The cyan lines indicate the knot layout. The white discs indicate control vertex locations; they are returned by the interpolation sub-system. In Figure 1, the yellow numbers indicate the value of the interpolating surface at the centre of each patch. The red numbers represent the input data. In Figure 2, the surface has been contoured. In Figure 3, the contoured surface is shown in a 3-D perspective view, with spheres indicating the data points.

The contouring sub-system

The following contouring parameters can be made available for interactive editing by the user to control the trade-offs between system response time, the level of contour detail, and image quality.

A contouring switch

This provides the user with a computationally simple substitute for contours so that frame changes near 30 Hz can be made. In the present system this substitute is a map of numeric field values displayed when operating in "map" mode, as shown in Figure 1. As the selected control vertex is raised or lowered, the yellow values give sample surface heights at the cross-hair positions which change in "speedometer" fashion, providing a real-time digital readout. In "perspective view" mode, 3-D cross-hairs located at the centre of each patch "float" on the surface in real time, moving up and down in response to editing a control vertex.

Contour interval

While an interactive interface has not been completed yet, both global and local setting of the contour interval have been provided for. A coarse contour interval can be specified for fast interactive editing, while a finer interval can specify a higher density of contours for a high-quality, static display.

Adjacent-point spacing

This defines the accuracy of the initial guess for the Newton iteration as well as the nominal stepsize to be maintained along the contour. It can be indirectly specified by the user in terms of the "nominal segment length" described earlier.

Contour rendering

At the time of writing, the stream of points on contours were simply connected by straight line segments. Experiments with B-spline approximations will be conducted in the near future, and a switch will be provided.

The main editing functions

Provision for interactive moving, insertion, deletion, and appending of knots (and corresponding rows or columns of control vertices) are foreseen for future versions of the software. At present, only the interactive editing of control-vertex values is provided for. Control vertices are picked by the mouse, and the patches effected by a picked vertex are displayed in green.

A picked vertex is tied to Y-movement of the mouse when a designated mouse button is pressed, and a computationally simple substitute for the effected contours, sample function values changing in real time, is used to monitor the continuous sliding of the vertex height. Upon release of the button, the locally effected patches are re-contoured. This sequence of events is illustrated in Figures 4 to 7 in map mode, and in Figures 8 to 11 in perspective view.

A typical time to contour 16 bi-cubic patches is about 10 seconds. The previous contours are blanked out, and the altered contours are displayed as each patch is completed to provide the user some visual gratification. Contouring time varies with the level of complexity. If there are few contours to be drawn, and the spline is of low degree, then the response is very fast. For a complex patch with many contours, the system is seen to pause while the contours are computed. It should be noted, however, that at the time of writing the design was not fully optimized, and significant speedup should be possible, as will be mentioned below.

5. FURTHER STUDY

Firstly, it is worth noting that major reductions in numerical effort and some of the associated overhead could be achieved by selecting special cases of B-splines. For example, if we chose to limit the basis to uniform bi-cubics, then all function and derivative evaluations can be much more rapidly computed from well-known formulae, at the cost of the editing flexibility available to the user.

Secondly, in the present experimental system, each patch is being contoured independently of any other patch, and each contour/patch-boundary intersection is regarded as separate and unique. While this has been very useful from a software-engineering point of view, it clearly results in duplication of effort:

the contour intersection module operates on every internal patch boundary twice — once for each incident patch — and each contour is traced twice within a patch — once from each boundary intersection. The development of a data structure for this information is planned, and an increase in speed of from 2 to 4 times could be expected.

Thirdly, in the context of editing on high-speed hardware, a contour will change only slightly from one refresh to the next. It is possible to take advantage of this coherence to speed up the process of locating the intersections of contours with patch boundaries.

6. REFERENCES

Bartels87.

R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to the Use of Splines in Computer Graphics,* Morgan Kaufmann Publishers, Los Altos, California (1987).

Boehm84.

W. Boehm, G. Farin, and J. Kahmann, A Survey of Curve and Surface Methods in CAGD, *Computer Aided Geometric Design* **1**(1) pp. 1-60 (1984).

deBoor78.

C. de Boor, *A Practical Guide to Splines,* Springer-Verlag (1978).

Cohen80.

E. Cohen, T. Lyche, and R. Riesenfeld, Discrete B-splines and Subdivision Techniques in Computer-Aided Geometric Design and Computer Graphics, *Computer Graphics and Image Processing* **14**(2) pp. 87-111 (1980).

Kulikov82.

A. I. Kulikov and G. S. Rivin, Mapping Fields of the Meteorological Elements Given at the Nodes of a Latitude Longitude Grid, *Meteorol. and Gidrol. (Meteorology and Hydrology) (U.S.S.R.)* **1** pp. 41-48 (1982).

Lawson74.

C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems,* Prentice Hall (1974).

Lyness83.
J. F. Lyness and L. Asquith, A Simple Contour Plotting Program for Finite Element Output, *Advanced Eng. Software* **5**(1) pp. 23-31 (1983).

McLain1974.
D. H. McLain, Drawing contours from arbitrary data, *The Computer Jnl.* **17** pp. 318-324 (1974).

Moore77.
I. G. Moore, Automatic Contouring of Geological Data, *15th APCOM Symposium*, pp. 209-219 (1977).

Preusser84.
A. Preusser, Computing Contours by Successive Solution of Quintic Polynomial Equations, *ACM Trans. on Math. Software* **10**(4) pp. 463-472 (1984).

Preusser84a.
A. Preusser, Algorithm 626 — TRICP: A contour plot program for triangular meshes, *ACM Trans. on Math. Software* **10**(4) pp. 473-475 (1984).

Sablonnière85.
P. Sablonnière, Bernstein-Bézier Methods for the Construction of Bivariate Spline Approximants, *Computer Aided Geometric Design - Surfaces in CAGD '84*, pp. 29-36 North-Holland, Amsterdam, (1985).

Snyder78.
W. V. Snyder, Contour Plotting [J6], *ACM Trans. on Math. Software* **4**(3) pp. 290-294 (1978).

Suffern84.
K. G. Suffern, Contouring Functions of Two Variables, *The Australian Computer Jnl.* **16**(3) pp. 102-106 (1984).

Sutcliffe76.
D. C. Sutcliffe, An algorithm for drawing the curve f(x,y) = 0, *The Computer Jnl.* **19** pp. 246-249 (1976).

Figure 1:

An example of the interpolation of scattered data, where $k = l = 3$, $n = m = 5$, and the number of scattered points (red numbers) is 34. The yellow numbers indicate the interpolating surface's height at the centre of each patch.

Figure 2:

This figure is the same as Figure 1, except that the interpolating surface has been contoured at an interval of 1.0.

Figure 3:

This figure displays the information in Figure 2 in perspective view. Map view (Figure 2) and perspective view (Figure 3) are toggled by a mouse button. The data points are displayed as spheres.

Figure 4:

An example of a surface to be edited, where $k = l = 4$ and $m = n = 8$; The contour interval is 0.5 and the control vertex values range from -2.0 to 7.0.

Figure 5:

The first step in control vertex editing: The cursor (in red) is picking a vertex near the top of a "hill" on the surface. The patches controlled by the picked vertex are coloured green.

Figure 6:

The second step in control vertex editing: After a vertex has been picked, and while the left mouse button is pressed, a computationally

simple substitute for contours is displayed to monitor changes in height of the picked control vertex, controlled by Y-movement of the mouse.

Figure 7:

Upon releasing the left mouse button, there is a pause while each patch is re-contoured. This figure shows the result of lowering the picked control vertex of Figure 6.

Figure 8:

The perspective view of Figure 4.

Figure 9:

The perspective view of Figure 5.

Figure 10:

The perspective view of Figure 6. In this case, the control vertex was picked while in this mode of view, so 3-D cross hairs float on the surface in real time, in lieu of the numbers in Figure 6. The cross-hairs move up and down smoothly, in response to movement of the picked vertex.

Figure 11:

The perspective view of Figure 7.

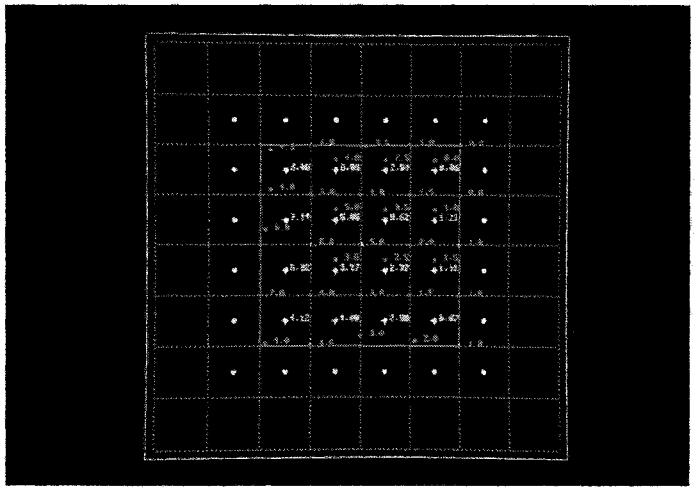

Figure 1:
An example of the interpolation of scattered data, where $k = l = 3$, $n = m = 5$, and the number of scattered points (red numbers) is 34. The yellow numbers indicate the interpolating surface's height at the centre of each patch.

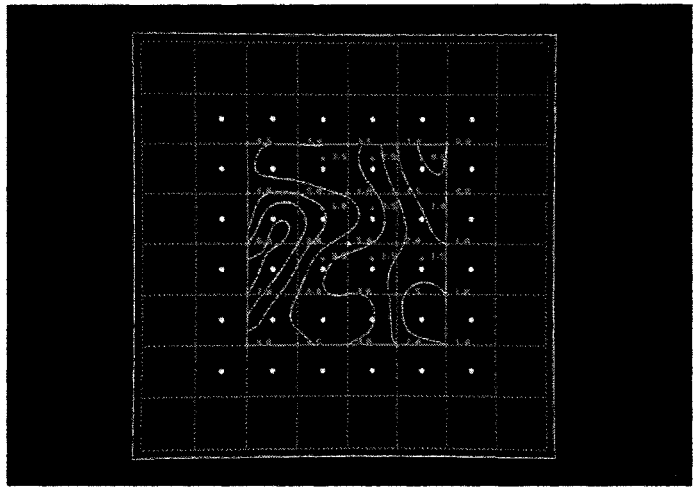

Figure 2:
This figure is the same as Figure 1, except that the interpolating surface has been contoured at an interval of 1.0.

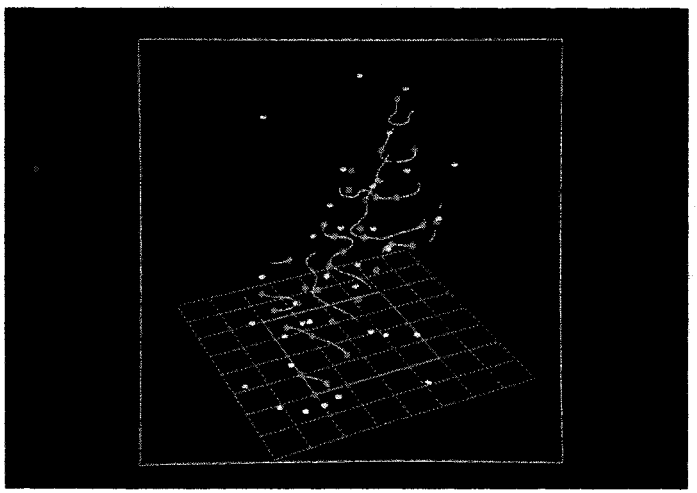

Figure 3:
This figure displays the information in Figure 2 in perspective view. Map view (Figure 2) and perspective view (Figure 3) are toggled by a mouse button. The data points are displayed as spheres.

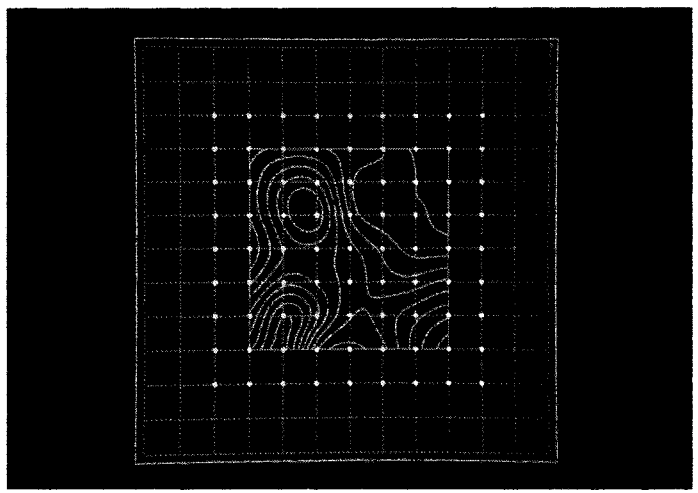

Figure 4:
An example of a surface to be edited, where $k = l = 4$ and $m = n = 8$; The contour interval is 0.5 and the control vertex values range from -2.0 to 7.0.

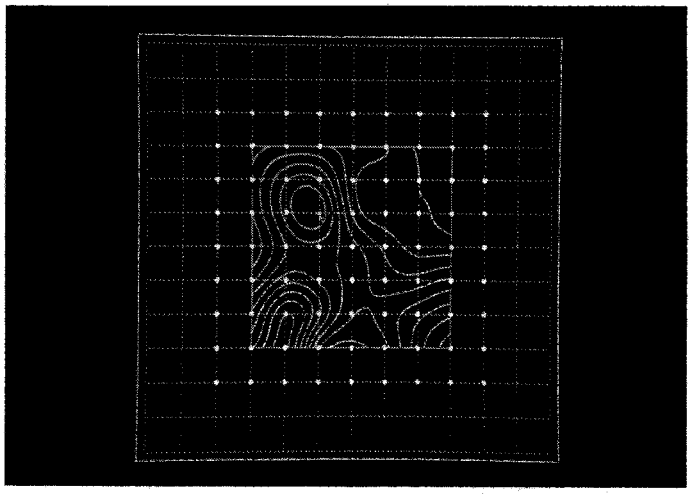

Figure 5:
The first step in control vertex editing: The cursor (in red) is picking a vertex near the top of a "hill" on the surface. The patches controlled by the picked vertex are coloured green.

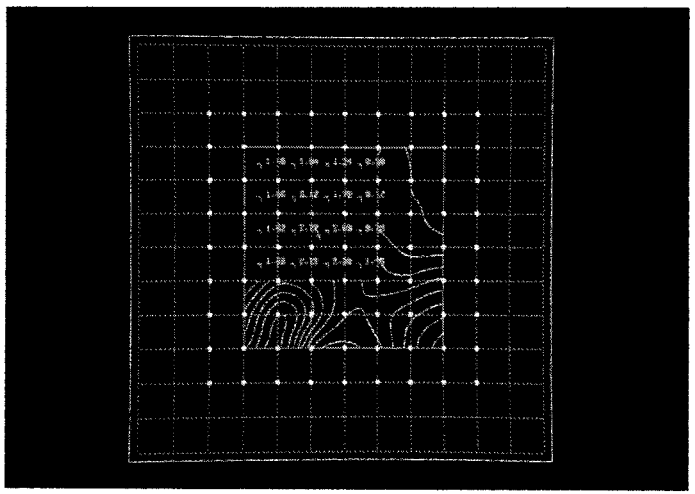

Figure 6:
The second step in control vertex editing: After a vertex has been picked, and while the left mouse button is pressed, a computationally simple substitute for contours is displayed to monitor changes in height of the picked control vertex, controlled by Y-movement of the mouse.

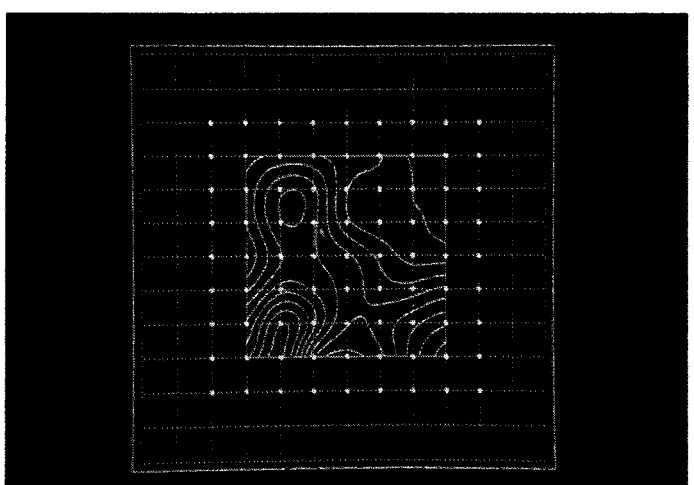

Figure 7:
Upon releasing the left mouse button, there is a pause while each patch is re-contoured. This figure shows the result of lowering the picked control vertex of Figure 6.

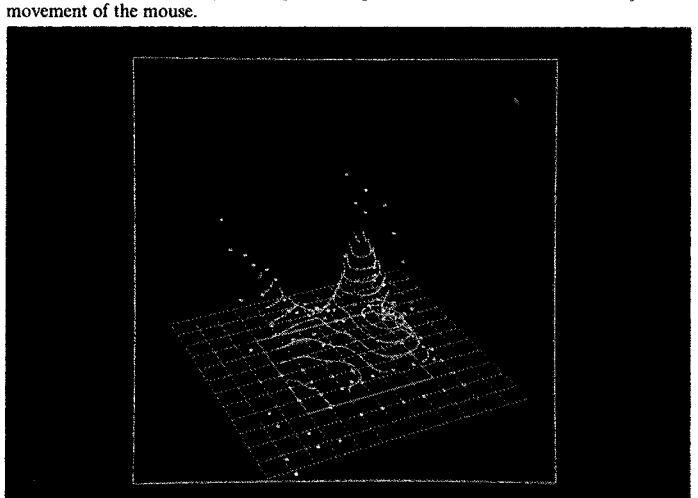

Figure Figure 8:
The perspective view of Figure 4.

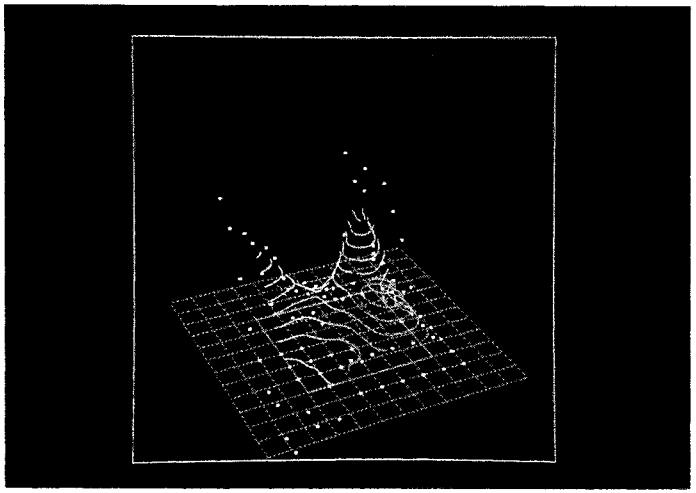

Figure 9:
The perspective view of Figure 5.

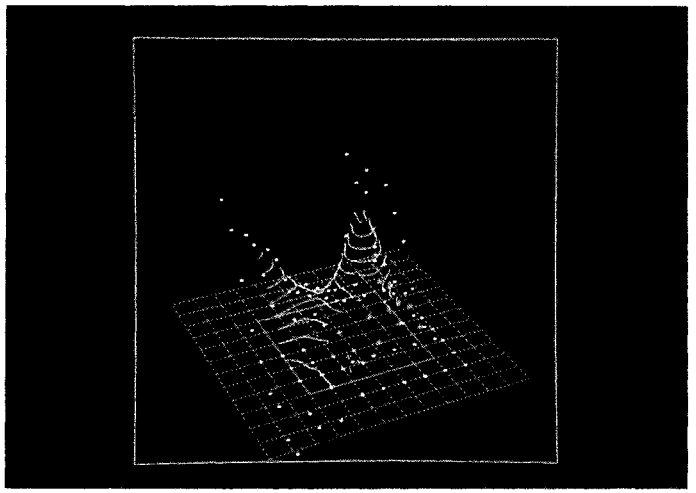

Figure 10:
The perspective view of Figure 6. In this case, the control vertex was picked while in this mode of view, so 3-D cross hairs float on the surface in real time, in lieu of the numbers in Figure 6. The cross-hairs move up and down smoothly, in response to movement of the picked vertex.

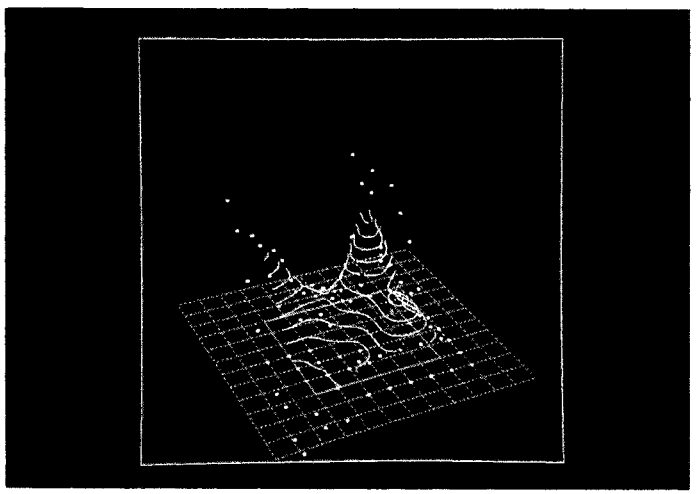

Figure 11:
The perspective view of Figure 7.