V. 8, n.º 3 (setembre 1984) pp. 133-144

SMOOTH CONTOUR LINE CONSTRUCTION WITH SPLINE INTERPOLATION

PERE BRUNET I LL. PÉREZ VIDAL

UNIVERSITAT POLITÉCNICA DE CATALUNYA

Contour maps are frequently used to represent three-dimensional surfaces from geographical applications or experimental results. In this paper, two new algorithms for the generation and display of such contours are presented. The first of them uses local spline interpolation to obtain contour maps from data points in a rectangular mesh, whereas the other interpolates a set of irregular points through recursive subdivision of triangles. In both algorithms, precision of the contours can be adjusted by the user by means of a parameter . Results are presented and discussed.

Keywords: SURFACE REPRESENTATION, CONTOUR MAPS, SPLINE INTERPOLATION

1. INTRODUCTION.

Contour maps are commonly used to represent a three dimensional surface in two dimensions. They are used mainly in geographical maps, but are also useful to show isobars in weather maps, to display experimental results or to present finite element resulting stresses and deformations.

When data are supplied in a rectangular mesh, some algorithms follow the contours through the mesh rectangles, while others treat each rectangle at a time, /1/.

The former are better in the sense that they minimize pen movements (assuming that contours are drawn with a plotter, and that contour labeling is easier. Some algorithms /2/, use linear interpolation in every rectangle of the mesh.

The resulting are only approximate, but they are very useful when the output is done through serial character-oriented devices. More sophisticated algorithms interpolate every rectangle by means of quadratic or bicubic functions; in a second step, they compute the intersection between the functions and horizontal planes. A third class of algorithms compute the intersection of the contours with the grid lines; after this, the set of computed points from the same contour are joined together. A good survey of these algorithms can be found in /1/.

Many methods have been devised for the interpolation and obtention of contour lines /3/ for data consisting of a set of points (x_i, y_i, z_i) scattered over the x-y plane. Among them we have algorithms based on a single global interpolant: Shepard energy ; they are usually expensive in terms of computing time, and they may present smoothness problems. A second class of algorithms translate the data onto a rectangular grid, and then apply the known algorithms for this kind of grids. Another important group of methods automatically compute a triangulation of the data points in the x-y plane, and then fill every triangle with a suitable mathematical function, ensuring continuity between adjacent triangles. The surface equation within a triangle may be linear (this is the simplest choice, and gives poligonal approximate contours), a set of quadratic pieces (Powell and Sabin), cubic pieces or even a quintic (Akima). However, the most precise methods follow each contour, using values interpolated close to the track of the contour to

- Pere Brunet i Ll. Pérez Vidal - Departament de Metodes Informatics de l'Escola Tecnica Superior d'Enginyers Industrials de Barcelona - Universitat Politecnica de Catalunya - Av. Diagonal, 647 - Barcelona 08028

determine its position.

Two new contouring algorithms are introduced in this paper. In the next two sections, an algorithm for the construction of contour lines from rectangular grids of data is presented. It is very clear and easy to implement. It uses local bicubic spline interpolation, thus reducing the computing time required. The user can adjust a precision parameter that controls the smoothness of the contours. Intersection of contour lines is avoided by automatical refinement of the computations at complex zones.

Section 4 of the paper presents an algorithm for the generation of contour lines in the case of scattered data. It is based on an initial triangulation of the domain. In a second step, the partial derivatives at the data points are estimated. Then, every triangle is recursively divided into four, conputing the ordinates and partial derivative estimates at the middle of every side of the triangle.

Both algorithms reduces the problem of computing contour lines, to the determination of contours through very small rectangles or triangles, which is a well-konwn problem. In both cases, smoothness is obtained by a reduction of the size of these elementary polygons, that can be controlled by the user.

2. SURFACE INTERPOLATION BY MEANS OF LOCAL SPLINES:

Let us suppose that a set of "n" samples of a function $y = f(x)$ is given

$$(x_1,x_1),\ (x_2,y_2),\ldots\ldots,\ (x_n,y_n) \qquad (1)$$

and we want to interpolate them.

To simplify the spline equations, we will assume that the x_k are equally spaced, $x_{k+1}-x_k=1$, $\forall k$.

One of the best known interpolation algorithms is cubic spline interpolation. In this case, from the set of ordinates $y_1 \cdots y_n$, a curve $y = s(x)$ is obtained such that,

- $s(x)$ is a cubic in very interval $[x_k, x_{k+1}]$
- $s(x)$ interpolates the set of data points, $S(x_i) = y_i \ \forall\ i$
- A certain continuity is required between every two cubics, in order to obtain a smooth interpolant.

Let us suppose that we are able to estimate the slopes $\dot{y}_1 \ldots\ldots \dot{y}_n$ of the interpolated curve at the given points; then, we can define the cubic spline in the interval $x_k,\ x_{k+1}$ as the Hermite interpolant,

$$s(x) = \left[t^3 t^2 t 1\right] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} y_k \\ y_{k+1} \\ \dot{y}_k \\ \dot{y}_{k+1} \end{Bmatrix}, \quad t = x - x_k \qquad (2)$$

It is easy to see that $s(x_k) = y_k$, $s(x_{k+1}) = y_{k+1}$, $\dot{s}(x_k) = \dot{y}_k$, $\dot{s}(x_{k+1}) = \dot{y}_{k+1}$.

It is a cubic spline with slope continuity in the overall intervall $[x_1, x_n]$. If a well-known tridiagonal linear system is solved to compute the vector of slopes $\dot{y}_k$, /4/ global splines with continuity c^2 are obtained-provided that two extra end conditions are supplied-. On the other hand, local splines are obtained /5/ if every slope estimate $\dot{y}_k$ depends only on adjacent values y_i, with $i \sim k$.

Although these splines are only c^1 and they are not so smooth as global splines, they are much easier to compute and behave locally: a change of value of one data point produces a local change in the shape of the interpolant $s(x)$. In what follows, local spline interpolation will be used, because of the simplicity of the obtantion algorithm. In addition, the smoothness of the resulting contour is very acceptable as will be seen in the figures.

At the first step in the derivation of local splines, the slopes are computed,

$$\begin{aligned} \dot{y}_k &= (y_{k+1} - y_{k-1}) / 2, \quad k=2\ldots n-1 \\ \dot{y}_1 &= (-3y_1+4y_2-y_3) / 2 \\ \dot{y}_n &= (y_{n-2}-4y_{n-1}+3y_n) / 2 \end{aligned} \qquad (3)$$

($\dot{y}_1$ is the slope of the quadratic interpo-

lant of the set y_1, y_2, y_3, and the same with y_n).

Due to the linearity of equations (2) (3), an alternative definition of local splines is possible. Let us define the basis elements $L_1(x) \ldots L_n(x)$ as the local spline functions such that L_k interpolates the values $0 \ldots 0,1,0,\ldots 0$,

$$L_k(x_k) = 1$$
$$L_k(x_i) =), \quad i \neq k \qquad (4)$$

for every function L_k, $k=1 \ldots n$

Then, it is clear that they are a basis of the local cubic splines interpolating n equally spaced points. Given the ordinates $y_1 \ldots . y_n$, the local spline S(x) is,

$$S(x) = \sum_{i=1}^{n} y_i \cdot L_i(x) \qquad (5)$$

Equations (3) and (4) imply that the basis functions $L_i(x)$ have small support. In other words, $L_i(x)=0$ if:

$$x \notin \left[x_{1-2},\ x_{1+2}\right]$$

Then, equation (5) can be rewritten as

$$s(x) = \sum_{i=k-1}^{k+2} y_i \cdot L_i(x) \qquad (6)$$

if $x_k < x \leq x_{k+1}$, $k>1$, $k \leq n-1$

And now, local behaviour of these splines is obvious.

This algorithm can be extended to the obtention of a surface from a given mesh of points, z_{ij}, $i=1 \ldots n$.

The well-known tensor product algorithm,/6/ allows computation of interpolated values s(u,v) from the two sets of one-dimensional cubic spline basis elements, $L_i(u) \ldots L_n(u)$ and $L_1(v) \ldots L_m(v)$,

$$s(u,v) = \sum_{i=1}^{n} \sum_{j=1}^{m} z_{ij},\ L_i(u) \cdot L_j(v) \qquad (7)$$

As we did before, equation (7) can also be reduced if locality of the splines is taken into account. Then, assuming that the point (u,v) we want to interpolate has the properties

$$u \in [x_1,\ x_{k+1}] \quad , \text{ and } \quad v \in [y_1,\ y_{1+1}] \ ,$$

$$s(u,v) = \sum_{i=k-1}^{k+2} \sum_{j=1-1}^{1+2} z_{ij} \cdot L_i(u) \cdot L_j(v) \qquad (8)$$

(it is assumed that $k>1$, $1>1$, $k<n$, $1<m$).

As a consequence, every interpolated value depends only on sixteen initial values z_{ij}. Using the definition of the basis elements L_i and L_j, a straightforward mathematical computation leads to the following representation of the spline,

$$S(u,v) = \left[t^3 t^2 t 1\right] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \left[P\right] \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{Bmatrix} w^3 \\ w^2 \\ w \\ 1 \end{Bmatrix} \qquad (9)$$

where,

$$\left[P\right] = \begin{bmatrix} z_{ij} & z_{i,j+1} & \Delta_j(i,j) & \Delta_j(i,j+1) \\ z_{i+i,j} & z_{i+1,j+1} & \Delta_j(i+1,j) & \Delta_j(i+1,j+1) \\ \Delta_i(i,j) & \Delta_i(i,j+1) & \Delta_j[\Delta_i(i,j)] & \Delta_j[\Delta_i(i,j+1)] \\ \Delta_i(i+1,j) & \Delta_i(i+1,j+1) & \Delta_j[\Delta_i(i+1,j)] & \Delta_j[\Delta_i(i+1,j+1)] \end{bmatrix}$$

$x_i \leq u \leq x_{i+1} \quad t=u-x_i \ , \quad (x_{i+1}-x_i=1. \quad \forall i)$

$y_i \leq v \leq y_{i+1} \quad w=v-y_i \ , \quad (y_{i+1}-y_i=1, \quad \forall i)$

$\Delta_i(k,\ell) = 1/2[Z_{k+1,\ell} - Z_{k-1,\ell}]$,

$\Delta_j(k,\ell) = 1/2[Z_{k,\ell+1} - Z_{k,\ell-1}]$

Equation (9) is the well-known formulation of the bicubic patch /7/. The interpolated value depends on the position (t,w), and the matrix P. In this matrix, the four top-left terms are the data values of the points surrounding (u,v). Top-right terms are estimates of the partial derivatives D_{yz} at the same four points. They can be computed by means of equation (3) which, can be used for every value of "i" and "j", including the boundaries of the domain. The four bottom-left terms are the estimates of the derivatives D_{xz}, which can also be computed using equation (3). And the bottom-right terms can be interpreted as estimates of the cross-derivatives. They measure the twist of the surface at the four corners of the square of the mesh that contains point (u,v). To estimate them, we must use the whole set of 16 data values surrounding (u,v). However, in

most applications, twist derivatives are assumed to be null. In this case, matrix P has four null terms $P_{33}=P_{34}+P_{43}+P_{44}=0$, and only first partial derivatives must be estimated. The bicubic patch thus obtained is called an F-patch, /7/.

3. OBTENTION OF SMOOTH CONTOUR MAPS FROM EQUALLY DISTRIBUTED DATA POINTS.

In this section an easy-to-implement algorithm to obtain contour maps from a set of points distributed in a rectangular mesh is presented. Surface interpolation by means of local splines is performed to smooth initial data. A local algorithm assures enough smoothness of the contours with small computing time. Once the interpolation has been performed, a thinner mesh is computed and the pieces of contour within every rectangle of this second mesh are obtained, using linear interpolation. Direct computation of contours of bicubic patches, which is expensive and involves iterative solution of cubic polynomials, is avoided.

In more detail, the algorithm proceeds through the following steps,

- For every rectangle (i,j) in the mesh, i=1..n-1, j-1..m-1,

  1 - Estimate the partial derivatives $D_x z$, $D_y z$, at the four corners of the rectangle, using equation (3).

  2 - Form the matrix P of equation (9)

  3 - Using (9), compute the interpolated value at points within the rectangle. These points (u_k, v_e) are equally spaced and form a finer mesh. In particular,

$$u_k=x_i+(k-1)*(x_{i+1}-x_i)/k_p, \quad k=1..k_p+1$$

$$v_l=y_j+(l-1)*(Y_{j+1}-y_j)/k_p, \quad l=1..k_p+1 \qquad (10)$$

  4 - For every contour,

    4.1- Find the rectangles of the finer mesh containing part of the contour. These rectangles must have some of the vertex ordinates higher than the contour, while the others must be lower.

    4.2- For every rectangle found in step 4.1, draw a straight segment aproximating the contour within it.

The algorithm used in this last step 4.2 is similar to that of /2/. The four edges of the rectangle are studied; is an intersection between the contour and the edge is detected (the heigh of the contour is between those of the ends of the edge), the coordinates of the intersection point are computed and stored in a stack. At the end, there are only two possible states,

- The stack contains two points. In this case, a straight segment is drawn, connecting the points.

- The stack contains four points. In this case, the contour intersects the four edges of the rectangle, and the connection between the points cannot be determined from the information from the rectangle vertices. As we know the mathematical equation of the bicubic patch, it is possible to divide recursively the rectangle into four quadrants, until indetermination is avoided, and the stack contains zero or two points for every elementary rectangle. An alternative solution is to simply connect the two pairs of opposite intersection points; then, a cross-hatch will appear in the contour map, indicating to the user that precission must be increased.

The input arguments to the contouring procedure are "n", "m" the initial matrix of values z_{ij}, i=1...n, j=1...m, the desired heights of the contours, and the precision parameter k_p. The user must also supply a working two-dimensional array of dimension $(k_p+1) \times (k_p+1)$, that will contain the set of interpolated values in very rectangle of the cell (step 3 of the algorithm). If k_p is small, the amount of interpolated points is also small, and the contours will be aproximated by large straight segments.

If k_p increases, contours tend to be smoother. A good choice for parameter k_p is usually between 5 and 15 (observe that the working array has always a reasonable size). Values of k_p beyond 15 or 20 produce excesively short

segments and increase drawing time.

Figures 1 to 6 show the results of the above algorithm when applied to the function

$$z(x,y)=15.(x^2-y^2).\cos(x^2+y^2+4x-5y)$$

Imput data was a 6 x 6 matrix z_{ij} containing the z(x,j) values at $x_i=-1+.4*(i-1)$, i=1...6 and $y_j=-1+.4*(j-1)$, j=1...6

Contours were required at
z_k= -5, -4, -3, -2, -1, 1,2,3,4,5.

To show in more detail the effect of a change in the precision parameter k_p, the recursive division of the rectangles of the finer mesh was eliminated.

Figure 1 shows the results of the contouring algorithm with k_p=1 (no interpolation is applied to data values). A great number of indeterminations is observed and the discontinuities in the contours make interpretation of the drawing difficult.

Figures 2 and 3 show the results when k_p is increased to 2 and 3, respectively. Many of the indeterminations have disappeared, and the contours become smoother. These effects are quite clear in figures 4 and 5, where k_p=5 and 10. These last figures would be quite acceptable if recursion in problematic rectangles had been used.

However, in our case, we must increase k_p up to 18 in order to avoid all indeterminations.

Comparing figures 4, 5 and 6 we see however that a good rate of smoothness is reached soon.

In all cases, the computing time taken by the algorithm is negligible in front of the representation time in a screen or plotter.

The presented algorithm contains a loop of (n-1) x (m-1) bicubic patches; on everyone of them, contours are generated one after the other. The algorithm follows every contour all along the patch. Because of this, there are very few movements of the plotter pen in the "not drawing" mode.

4. CONTOURS MAPS FROM SCATTERED DATA POINTS

We now face the problem of a set of data points that are not regularly distributed over the x-y domain we wish to consider.

The heights of data points can be either calculated from a given function

z = f(x,y)

or entered as a third datum value for each point.

The problem has been reviewed by Farin /8/: He has proposed a corrected triangular bivariate Bernstein-Bezier patch; each triangle side is devided in three segments, and a third-degree polynomial is made to interpolate the three points.

The approach we have considered to solve the problem involves the use of recursive refinement of the initial mesh dividing the sides of the triangles by two.

The procedure starts by the obtention of a triangularization mesh having as vertexs the given data points. After this first step we have the convex polygon enclosing all points.

The second step is the initial estimation of partial derivatives at each vertex of the original mesh.

The third step is a recurring subdivision of the original mesh in order to archieve a smooth transition of the values of partial derivatives from one node to the next.

The fourth and final step is the actual drawing of contour lines, which are in fact straight segments: Each and every triangle resulting from the last subdivision is inspected and the corresponding "heights" are interpolated in its three sides.

4.0. OPTIMAL TRIANGULARIZATION FOR A GIVEN SET OF POINTS.

We have used extensively Lawson's chapter /9/ to generate a triangulation for a given set of points.

The criterion to define a "better" triangle is the max-min angle:

The procedure will seek to maximize the smallest angle in a triangle which means that small angles in a triangle are "bad". The local optimization procedure proceeds as follows: A single internal edge of the global triangulation is chosen; by internal we mean that it is a common edge of two triangles belonging to the triangulation.

Then a swap of edges is considered: the minimum angle in the two triangles is calculated in the actual configuration and in the alternative configuration we would have if the internal edge being considered disappeared and the other diagonal of the quadrilateral thus formed was taken as the internal edge.

The edge giving the smallest internal angle is discarted.

Any internal edge of the triangulation is locally optimal if application of the local optimization procedure would not swap it.

When all internal edges in a triangulation are locally optimal, the triangulation is globally optimal.

The triangulation algorithm proceeds by first generating a near optimal triangulation with a constructive approach: From the minimum x-minimum y point we choose the two nearest ones. Then we choose the nearest "free" point and join it to the existing set of edges. The preceding phase is repeated until there are no more free points. And finally the global optimization procedure is run over the whole set of edges.

4.1. INITIAL ESTIMATION OF PARTIAL DERIVATIVES.

The problem has been reviewed by Nielson and Franke /10/ who have made reference to several works: Akima /11/ uses weighted normals of neighbouring triangles, Klusewicz /12/ the plain average of neighbouring triangle slopes, and Little /13/ again weighted normals of neighbouring polygons with weight factors involving edge lenghts.

We have found, and experimented, the truth of the statement by Nielson and Franke /10/: "The choice of technique used in this phase is rather crucial since it can have a significant effect on the overall quality of the method".

We started by averaging the normals (not the slopes) of the triangles with a common node and then assigned that node a weighted proportion of those normals. We tried several weight criteria, and all results were bad, because what has to be computed is the slope, that is, the partial derivatives of "z" with respect to "x" and "y".

The next trial was the assignment of a partial derivative which was that of each of the adjacent triangles weighted with a factor that was the angle formed by the edges that cross in the node being studied.

This criterion has shown its inadequacy; the reason is that when a triangle is relatively big its vertexs are far apart whereas their angles might be significantly great, thus giving great weight to a slope which might be very different from the local slope of the surface at the vertex being studied.

We have finally come to another criterion: The estimated slopes are calculated from those of the triangles concurring at the vertex being studied.

$$S(x,y) = \Sigma\ W(t\ i) * s(t\ i) / \Sigma\ W(t\ i) \quad (11)$$

Where S(x,y) is the estimated slope at vertex x,y; Σ is the sumatory for all triangles concurring at the (x,y) vertex;

W(t i) is the weight factor of triangle i for vertex x, y; and

S(t i) the slope of triangle i.

We do not actually compute the slope but the two partial derivatives, and so, S(x,y) S(t i) should be taken first as dz/dx and then as dz/dy.

$$dz/dx\ (y,y) = \Sigma(W(t\ i)*dz/dx\ (t\ i)) / \Sigma W\ (t\ i) \quad (12)$$

$$dz/dy\ (x,y) = \Sigma(W(t\ i)*dz/dy\ (t\ i)) / \Sigma W\ (t\ i) \quad (13)$$

The weight factor W (t i) is

$$W(t\ i) = \frac{1}{\Sigma\ (d^2_1 \quad d^2_2)^{\ alfa}} \qquad (14)$$

where d1 and d2 are the distances from the vertex being studied to the other two vertex of the triangle which slope is being weighted.

It is interesting to note that this criterion includes, as a particular case, when alfa=0, Klucewicz's criterion /12/.

We carried out several trials to determine the evolution of estimated slopes following several values of alfa. We use to that end, an hiperboloid of equation

```
z=.2* (x4.5)** 2-.3* (y-.5) ** 2 + .25   (15)
```

of which we know, analitically, the partial derivatives

```
dz/dx = .- * x - .2                      (16)
dz/dy = .3- .6 * y                       (17)
```

We first calculated the average of differences to the square between the estimated and the analytical partial derivatives, shown in Table 1 for a set of 41 regularly distributed data points (Fig 7). The influence of alfa is not significant. The reason is that the improvement of our criterion does not show when points are evently distributed.

We then tried with another set of 50 irregularly distributed data points (Fig. 8), and the results, in Table 2, show an optimum value of alfa=8 which we have retained for the rest of this work.

We have thus tackled the problem of estimating the partial derivatives at vertex points of a data set. But it has to be pointed out that it is not completely solved: the solution is quite satisfactory inside the convex polygon but shows some alterations in boundary vertexes due to lack of information on surrounding slopes for those points.

We propose to further study the possibility of assessing a better method to estimate slopes of points with little information on surrounding slopes.

TABLE 1

α	$\sum D_x^2$	$\sum D_y^2$
.01	.0221	.027
1	.021	.026
10	.0192	.0233
100	.019	.0230

TABLE 2

α	$\sum D_y^2$	$\sum D_y^2$
.01	.045	.045
1	.040	.041
4	.03	.034
6	.028	.0324
8	.027	.0320
10	.026	.032
100	.029	.036

4.2. RECURSIVE ALGORITHM FOR CONTOUR PLOT DRAWING.

In this section we present an algorithm to plot the contour lines of the surface interpolating a set of scattered data points.

The first procedure in the algorithm is the triangularization in the x-y domain for the given data points.

The second procedure is the initial estimation of the partial derivatives fully described in its details and implications in the preciding paragraph 4.1.

The third procedure is recursive and plays the main role in the algorithm. It takes triangles one at a time. First of all, it checks the depth of recursivity: If it falls under the given threshold then the contour lines can be plotted; if it does not, a "new" node is placed at the middle of each of the sides of the triangle in the x-y plane, and four new smaller triangles are obtained from each old bigger one. The height and slopes at each "new" node are obtained in different computations: The slopes are plainly the average from the two surrounding ones, that is, we perform a linear interpolation, but the height is a cubic spline interpolation from the data of, again, the surrounding points.

This method is clearly different from Farin's /8/: He divides each projection of the sides of the triangle by three and then computes a third degree polynomial that contains the four points; this polynomial is the interpolant in 3D space for that side.

The algorithm is, formally, as follows:

1- Read data points and other data (precision, alfa for weight factors, etc.)

2- Obtain an optimal triangularization on the x-y plane for the given set of data points.

3- Estimate partial derivatives dz/dx and dz/dy at each node of the triangularization.

4- For each triangle i=1.. nt

5- Computations for a triangle

5.1. Compute the difference between the partial derivatives at each one of the three vertexs and those of the triangle, and print them.

5.2. If the depht of recursivity has been reached, then

5.2.1. Draw the contours for this triangle

else

5.2.2. For each side of this triangle j=1,2,3

5.2.2.1. Divide the side by two and interpolate: components of a versor in the direction of the side:

```
u=x(2)-x(1)
v=y(2)-y(1)
new-x=x(1)+u/2
new-y=y(1)+v/2
d=sqrt (u*u + v*v)
ul=u/d
vl=v/d
```

basis rotation for the derivatives:

```
dl=dz/dx(1)*ul+dz/dv(1)*vl
d2=dz/dx(2)*ul+dz/dy(2)*vl
d3=dz/dx(1)*vl+dz/dy(1)*ul
d4=dz/dx(2)*vl+dz/dy(2)*ul
```

computation of partial derivatives at the midpoint:

```
nn=(dl+d2)/2
nm=(d3+d4)/2
new- dzdx=nn*ul-nm*vl
new- dzdy=nn*vl+nm*ul
```

5.2.3. Join the three mid-sides to form four small triangles.

5.2.4. For each of the four triangles k=1,2,3,4

5.2.4.1. Call (recursively) routine 5. for the triangle.

It has to be pointed out that the depth of recursivity for each and every triangle in the domain must be the same to ensure C1 continuity.

The final form of the algorithm has been described, but we have carried out tests with several other combinations of possible computations for height and slopes: average of normals, cubic spline interpolation of normals.

To demonstrate the effect of an increase in the threshold value for differences between slopes, and thus of the depth of recursivity, we have run and plotted two examples of the use of the algorithm for a set of points which yield a triangularization shown in figure 8 and that take heights that belong to the hyperboloid (15) for values of x and y between 0 and 1.

Data points in the x-y plane, as per figure 8, were generated randomly, but imposing the presence of four points at the corners of the domain and four at the mid-sides. Then the heights were calculated using equations (15), and an output data file was left on mass storage to be taken as input by the contour drawing program.

The plots are figure 9 for a depth of recursivity of 0 and 10 for a depth of 2. The smoothing of contours is clear as depth increases.

We have also studied a surface proposed by Nielson and Franke /10/,

```
z=4*(exp(-((x-2)/. 4)**2-(y-.2)**2)+exp(-((x-1.)/.35)**
**2-((y-.3)/.7)**2)-1/(1.+((x-.4)/.6)) **2+((y-9)/.4)**2))+.3
```

(18)

A considerably smoother contour is observed in figure 12 (depth=2) than in figure 11 (depth=0). Nevertheless, even with the smooth contours of figures 10 and 12 some perturbations can be ob-

erved at or near the confines of the plots: these problems arise from inaccurate estimations of partial derivatives at the nodes on the convex polygon frontier of the triangularization. And the inaccurancy comes, in turn, from a shortage of information:

In an "internal" node there is information on values of the slope of several surrounding nodes in every direction, whereas in "frontier" nodes we have a sector of at least 180 degrees (maybe more) about which slope we have no information. This effect is worsened by the bad shape of many triangles along the frontier, that are very elongated and thus with nodes far apart.

To separate the consequences of an inaccurate initial estimation of slopes at mesh vertexs, we have run the algorithm on the surface for equation (18); but instead of estimating the slopes by the procedure presented in paragraph 4.1., we have calculated them with a finite diference scheme, obtaining a far better set of values for slopes. The results of the contour map algorithm as applied with a good slope estimation are shown in figure 13; this shows clearly the remarkably good performance of contour line generation.

It has to be observed that the critical step in the set of procedures is the initial slope estimation at the given mesh nodes: if the estimation is accurate the latter part of the algorithm yields a good-looking contour map.

5. CONCLUSIONS.

Two algorithms have been presented to draw contour lines for regularly and irregularly distributed data points.

Both are simple to implement and work well.

The two algorithms compute spline interpolations and yield, either a triangular or rectangular mesh of sufficiently small polygons.

The contour lines being drawn are, for both, actual straight segments.

Tu further refine the smoothness of the resulting plots, the user must only increase threshold values of permissible deviations.

Several lines of further development are presented and work will proceed to enhance the quality of results.

6. BIBLIOGRAPHY.

/1/ SUTCLIFFE, D.C.: "Contouring over rectangular and skewed rectangular grids- An introduction". Math. Methods in Computer Graphics and Design, Ed. by K.W. Brodlie Academic Press 1980.

/2/ WARD, S.A.:"Real time plotting of approximate contour maps" Comm of the ACM vol. 21 n9, 1978.

/3/ SABIN, M.A.: "Contouring. A review of methods for scattered data". In Math. Methods in Computer Graphics and Design. Academic Press 1980.

/4/ BRUNET, P.: "Interpolació per funcions spline. Introducció automàtica de les condicions d'extrem". QUESTIIO, V.6 n.4, 1982.

/5/ DE BOOR, C.: "A practical guide to splines" Springer Verlag, 1978.

/6/ BRUNET, P., AYALA, D. NAVAZO, I.: "An interactive algorithm for the generation of B-spline surfaces", Procs of ICS-83, Ed. Berichte, n13, 1983.

/7/ ROGERS, ADAMS.: "Mathematical Elements for Computer Graphics". McGraw-Hill, 1976.

/8/ FARIN, G.: "Smooth interpolation to scattered 3D data", in Surfaces in Computer Aided Geometric Design;North Holland 1983.

/9/ LAWWSON, C.L.: "Software for C1 surface interpolation", in Mathematical Software III,Academix Press 1977.

/10/ NIELSON, M., FRANKE, R.: "Surface construction based upon triangulation", in Surfaces in Comp. Aid. Geom. Des. North Holland 1983.

/11/ AKIMA, H.: "A method of bivariate interpolation" ACM Trans. Math. Software, vol.4, 1978, pp. 148-159.

/12/ KLUCEWICZ, I.M.: "A piecewise C1 interpolant to arbitrarily spaced data". M.S. Thesis. Univ. of Utah 1977.

/13/ LITTLE, F.: CAGD report. Univ. of Utah 1984.

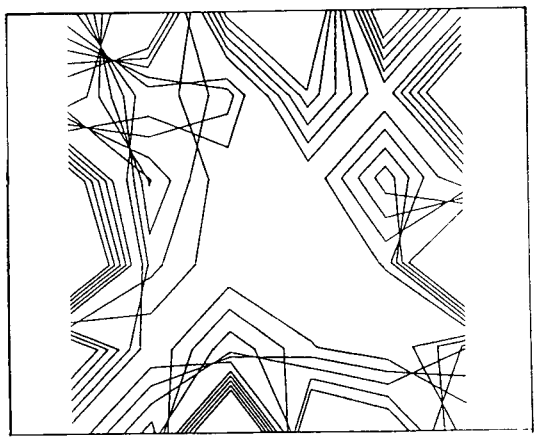

Figure 1:

Ten contour lines of a mathematical surface. Initial contours without spline interpolation.

Figure 2:

Contour lines of a mathematical surface. Interpolation by means of local splines, kp=2

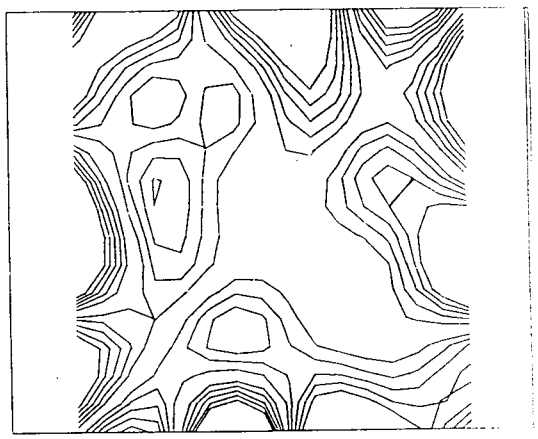

Figure 3:

Contour lines of the surface in figures 1 and 2. Interpolation with kp=3

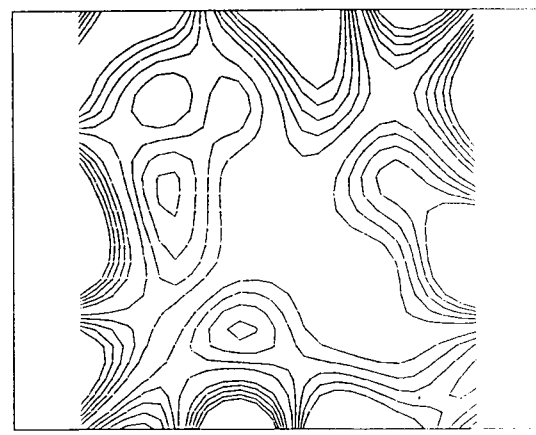

Figure 4:

Contour lines of the same surface as the preceding figures with kp=5

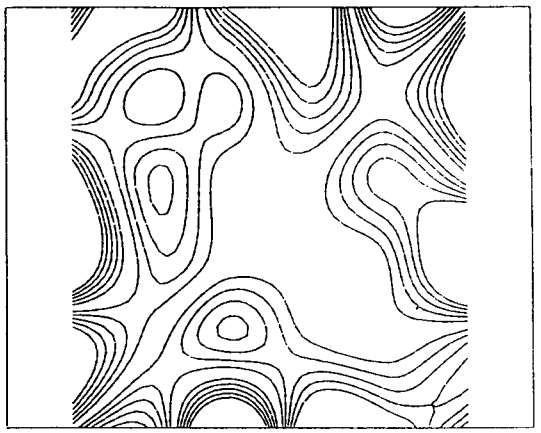

Figure 5:

Contour lines of the surface in the previous figures, with kp=10. Note that there is one indetermination still left

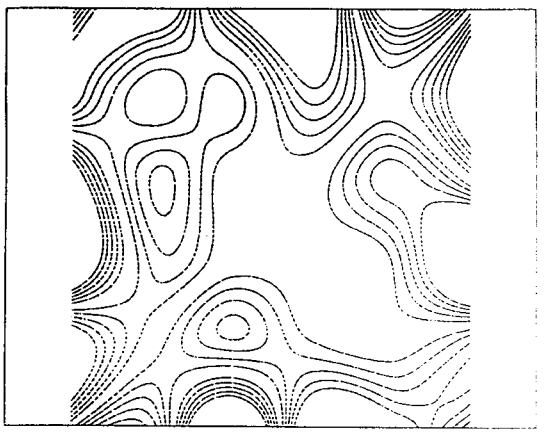

Figure 6:

Contour lines of the surface in the previous figures with kp=18. The single contour crossing in fig. 5 has been eliminated.

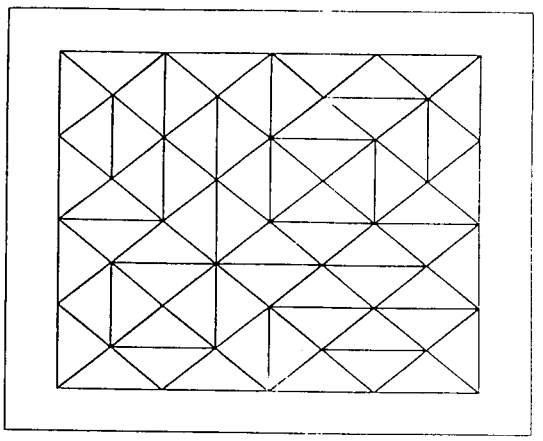

Figure 7:

A set of 41 regularly distributed data points.

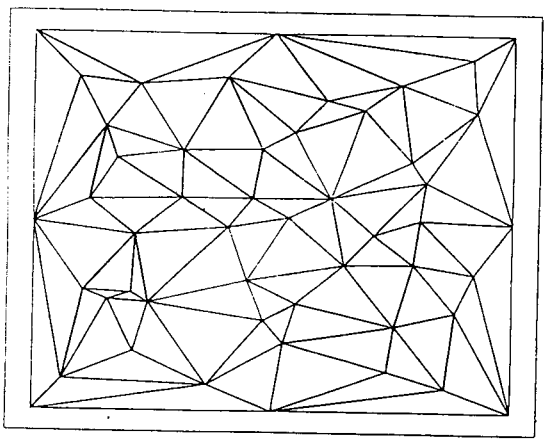

Figure 8.

A set of 50 random data points.

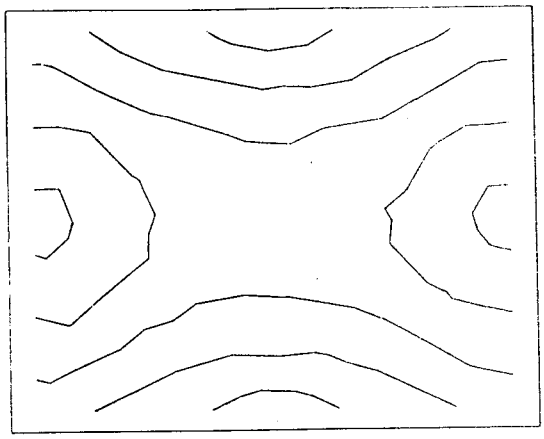

Figure 9:

Contour lines of an hiperboloid, without smoothing. Data points from figure 8

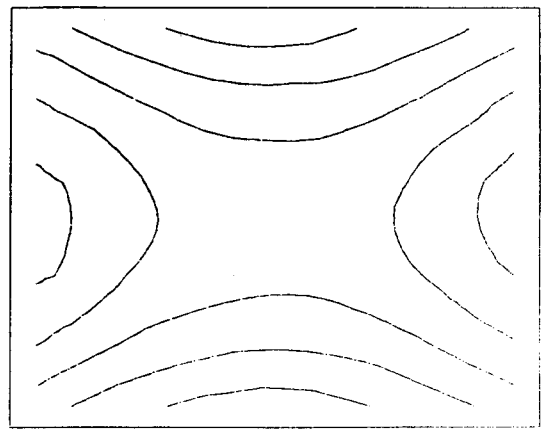

Figure 10:

The same contours as in figure 9, with a depth of recursivity of two. Note the imprevement in smoothness.

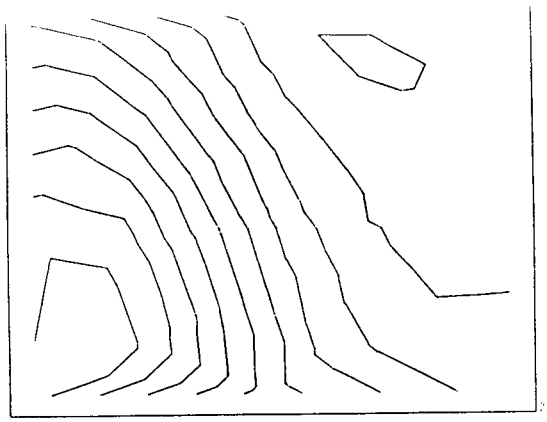

Figure 11:

Contour lines from a complex surface, without smoothing. 50 random data points.

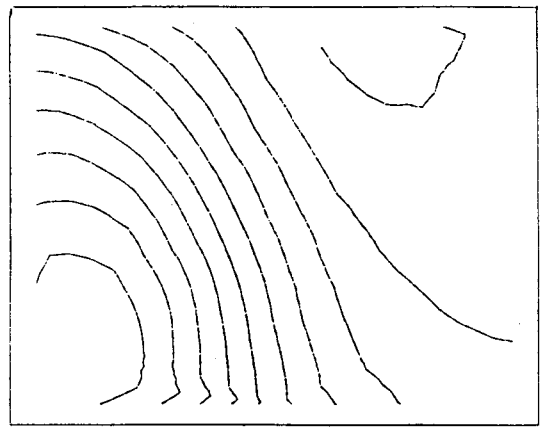

Figure 12:

The same contours as in figure 11, with a depth of recursivity of two. Some perturbations can be seen near the edge of the plot.

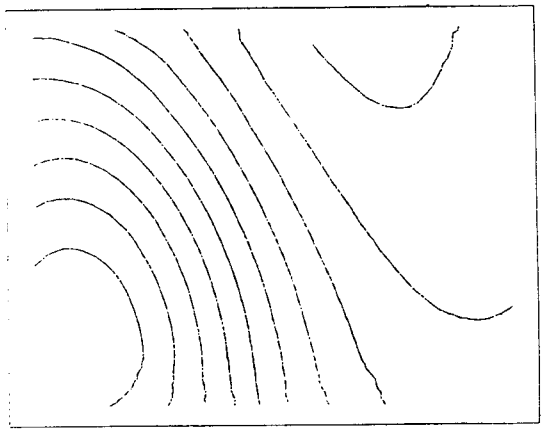

Figure 13: The same contours as in figure 12, with a good slope estimation.