Terrain Reconstruction from Contour Maps

Abigail Martínez Rivas and Luis Gerardo de la Fraga

Computer Science Section
Department of Electrical Engineering. CINVESTAV
Av. Instituto Politécnico Nacional 2508. 07300 México, D.F.
E-mail: fraga@cs.cinvestav.mx

Abstract In this work we address the problem of how to reconstruct the three-dimensional form of a terrain from its contour map. We present a methodology that comprises several algorithms able to solve this problem using both the skeleton and crust geometrical forms. We have divided this problem into two parts: (1) processing the scanned images of the topographic map to obtain samples of the contour lines, and (2) generate the best triangulation from the created sampled points in (1). Furthermore, we present an application example of our algorithms.

Keywords Computational geometry, terrain reconstruction, visualization, skeleton and crust.

1 Introduction

The problem to reconstruct a three-dimensional surface from a set of flat contours is an important problem if different fields. By example, biologists try to understand the form of microscopic objects through serial sections of the object. In clinic medicine, data generated by several techniques such as computerized axial tomography, ultrasound, and nuclear magnetic resonance, supply several cuts through the object under study [1]. In topography, contour line maps are yet the most common form to represent elevation data of Earth surface, and also they are an aid in quantitative studies in the natural sciences: weather at local scale, movement and action of the water, and consequently, numerous biological processes conditioned by them, are closely related to the form and altitude of the terrain surface where they are developed [2].

Despite the development of systems based in modern spatial satellites, a great deal of the world's database of terrain elevation is in the form of contour overlays from traditional mapping agencies (i.e., INEGI [3]). These maps have the advantage that they were developed using human understanding of the observed landforms, but the disadvantage that they can not be easily converted to a useful digital format. Due to the high cost of direct methods for terrain capture, cartographic documents such as contours maps are preferred [7].

By the said it before, the project of reconstruct the surface of a terrain represents an attractive visualization problem.

The problem of terrain reconstruction can be seen as a special case of the general problem of three-dimensional reconstruction from cross bidimensional sections. Reconstruction in this field means an interpolation problem of n contours each one with a specific elevation. The characteristic of these contours is they are not intersected and they are nested.

Here we proposed that the solution to the problem of the 3D terrain reconstruction can be divided in to parts: (i) process the scanned topographic map image to obtain the contour lines, and (ii) generate the best triangulation from the contour map.

Part (i) is a problem of digital image processing. The automatic extraction of both, the contour lines, and the altitude of them, is a complex task with difficult solution [4]. We simplify this problem: we start with a scanned image of a draw of the contour lines. Part (ii) has been resolved by several authors in different ways.

In 1998 Amenta *et al.* [5], in the Texas University, introduced the concepts of *crust* and *β-skeleton* to the reconstruction of curves. These concepts are defined in terms of two basic geometrical constructions: the Voronoi diagram and its dual, the Delaunay triangulation. The idea is to construct a graph over a set of bidimensional points, which capture its form in the following way: if a curve is smooth, F, is sampled with enough density, the graph over a set of samples, S, is a polygonization, or polygonal reconstruction, of the curve without superfluous edges. The polygonal reconstruction of F from S is a graph that connect every pair of samples adjacent along F, but not to others.

In 2002, Dakowicz y Gold [6] of the Politechnique University of Hong Kong, take up again the concept introduced by Amenta and the concept of triangulation methods (TINs), proposed a methodology to terrain reconstruction from contours, in a way that inserting some points in the skeleton in a TIM model is guarantee to eliminate the *flat triangles*, where the three vertices have the same elevation. One additional supposition about the local uniformity of the slopes, provide useful information to assign elevation values to the skeleton's points. In the paper also are compared several interpolation techniques using the extended set of contours. A previous work closely related is proposed by Thibault and Gold [7] in 2000.

Finally, in 2003, Horman *et al.* [8], of the California Institute of Technology, present a new contour interpolation method, For a point between two contours, is calculated the value of its elevation with Hermite interpolation based in the shortest distance to the contours and its heights. The curves of the generated surfaces arc C^1-continuous, except in some terrain portions like mountain chains or valleys, which are reconstructed as closed forms. Their proposed method reconstruct also summits and depressions. According to the authors, their approach allows a efficient numerical implementation.

We going now to develop every part that to solve the problem of 3D terrain reconstruction.

2 Image Processing

We start the image processing part with a digitalized image and we must obtain a set of points that sample each contour. From the input image we can extract all pixels shape the contour. However, it is no necessary to get all the pixels, these can be reduced taking sampled points where the curves are smooth. To carry out this sampling, we can see the problem in an inverse way: clearly it is impossible to build an algorithm that reconstruct any curve from any set of points; it is necessary that the set of points keep some quality conditions. In [5] resolves the sampling problem is the following way: the density of the samples required change in accordance with a measure of local size on the curve, so that the zones with less details can be sampled with a smaller density. A point p must be to a distance less than $r\mathrm{LFS}(p)$, where $r \leq 1$, between two samples over a curve. The LFS is the function of the *Local Feature Size*; the LFS of a point p s the Euclidian distance from p to the nearest point to the medial axis.

If we have a binary input image, then to sample the level curves we need follow the next steps:

1. Thinning the contour lines to a width of one pixel. We have used the thinning algorithm described in [9, Sec. 8.1.5] and programmed in [10]
2. Extract the contour lines
3. Calculate the LFS of the level curves, that is, over the image created before in the step 2, as follows:
   (a) Calculate the distance map using the best approximation with integer operations, the chamfer [11] of 3×3 size
   (b) Extract the highest points from the distance map using the mask showed in Fig. 1. This mask is as well as a detector of maxima (weights sum is zero), it is also a smoothing submask over the nine central pixels. The result to apply steps 3a and 3b is the *medial axis transform* of the contour lines image.
   (c) Calculate the distance map to the output image in the last step 3b. This new image is the LFS of the contour image.
4. Sample the contour lines.

-1	-1	-1	-1	-1
-1	1	1	1	-1
-1	1	8	1	-1
-1	1	1	1	-1
-1	-1	-1	-1	-1

Figure 1: Used mask to peak detection on the distance map

The step 2 of the methodology presented before can be performed using the algorithm 1.

The algorithm 1 is easy to understand: once a contour pixel is found, it is added to the list of pixels that belongs to the contour list. After that, the pixel is checked (using the function check_next_point()), if the pixel is terminal (branch$=0$), or it has a neighbor pixel ((branch$=1$), or it has several branches (branch>0). Finally the current pixel is erased. If the current pixel only has a neighbor, the new pixel simply will be the one returned by the function go_to_next_point(); to implement this function must be considered that the thinning algorithm, in the step 1, give the contours in a 4-connected way, then a 4-neighbor (instead a diagonal neighbor) is preferentially chosen. If the

Algorithm 1 Extraction of the contour lines

Input: Binary image to analyze

Output: A list of the contour lines, $\mathcal{L}_{i,j}$, i lines, j pixels per each one

```
$i \leftarrow 0$ {Counter of the number of contours}
for each pixel $p$ in the input image do
    if $p$ is black then {pixel in a line}
        $j \leftarrow 0$
    else
        continue
    end_contour ← FALSE
    flag ← TRUE
    repeat
        branch ← check_next_point( $p$ )
        if flag = TRUE then
            $\mathcal{L}_{i,j} \leftarrow \mathcal{L}_{i,j} + p$
            $j++$
        erase $p$ {Set pixel to white}
        if branch $= 0$ then
            if empty_stack( ) then
                end_contour ← TRUE
            else
                $p \leftarrow$ pop( )
                flag ← FALSE
        else if branch $= 1$ then
            $p \leftarrow$ go_to_next_point( $p$ )
        else
            push( $p$ )
            $p \leftarrow$ go_to_next_point( $p$ )
    until end_contour = TRUE
    $i++$
```

current pixel has branches, then it is pushed in a *stack* and, how it was added before to a contour list, a flag is set; so the pixel does not will be added again to the contour list in the next iteration, but it will be checked.

The sampling of the contour lines was implemented using the algorithm 2.

Algorithm 2 Sampling the contour lines

Input: The list of pixels that shape the contours, $\mathcal{L}_{i,j}$, i contours, j pixels per contour, and the sampling factor $r \leq 1$.

Output: The list of the set of points r-sampled, $R_{i,k}$, k sampled points per each contour i.

```
for each contour $i$ do
    $k \leftarrow 0$ {Counter of the number of samples}
    $p \leftarrow \mathcal{L}_{i,0}$ {first points in the contour $i$}
    $R_{i,k} \leftarrow R_{i,k} + p$
    for each point $p$ in $F_{i,j}$, $j > 0$ do
        $p_{\text{middle}} \leftarrow F_{i,j}$
        measure $\leftarrow r * \text{LFS}(p_{\text{middle}})$
        $d \leftarrow$ EuclidianDistance( $p$, $p_{\text{middle}}$ )
        if $d \geq$ measure then
            $d \leftarrow 0$
            while $d \geq$ measure and there are points in the contour do
                $p \leftarrow$ pixel $\mathcal{L}_{i,j+1}$
                $d \leftarrow$ EuclidianDistanc( $p_{\text{middle}}$, $p$ )
            $k++$
            $R_{i,k} \leftarrow R_{i,k} + p$
```

The description of algorithm 2 is as follows: the started pixel in a contour belongs to its set of sampled points. For the next points, the *characteristic measure* of the current point ($r * \text{LFS}(p)$), and the Euclidian distance between the previous sample and the current pixel, are calculated. If the distance is greater or equal to measure of the point, then we are found a point, p_{middle}, located roughly in the middle of two samples; we proceed to calculate the Euclidian distance between p_{middle} and the next points. Here, we need to find a point of which distance to the p_{middle} is greater or equal to the *characteristic measure* of p_{middle}. Such point will be the next in the set of sampled points. We iterate until to finish all contours.

3 Interpolating the contour points

Once we have all the sampled points that shape the contours, the easiest way the obtain a 3D reconstruction is calculating the Delaunay triangulation to all of them, and assigning altitude values to each contour. However, this results in a bad quality reconstruction, because of the presence of flat triangles and big triangles among the contours (there are not smooth changes in the surface).

Here we propose a methodology based in [6], and using the geometrical constructions of *the crust* and *the skeleton*:

1. The crust is the sampled points of the contour lines.
2. The skeleton is calculated. All the skeleton's points are stored in a list $\mathcal{L}_e$.
3. The number of skeleton's points is reduced. This is obtained calculating the average distance among the nearest points under a threshold (this could be the smallest characteristic measure).
4. Compute the Delaunay triangulation with the total set of points (crust's and skeleton's points).
5. Assign altitudes to each vertex
6. Visualize the reconstructed terrain

The computation of the skeleton, in the step 2, was obtained according to the algorithm described in [6]. In this algorithm are used the Voronoi diagram, the Delaunay triangulation (both calculated with the *qhull* program [12]), and the standard test *InsideCircle*. The test *InsideCircle* is applied to four different points in the plane, named A, B, C and D; it is determined if point D is outside of the circle drawn through points A, B, and C; it is assumed that the circle is counterclockwise sense.

To obtain the skeleton, it is built using the Voronoi diagram and the Delaunay triangulation, both made with the brush's points. After that, for each Delaunay edge (A, C) and its dual Voronoi edge (B, D), the test *InsideCircle* is computed. If the test is positive, meaning that D lie outside of the circle drawn through (A, B, C), then the Delaunay edge (A, C) is part if the crust. And is the test fail, the Voronoi edge (B, D) is included in the skeleton.

To assign an altitude to each vertex, we used the algorithm 3. This algorithm is too easy: all points of a crust contour have the same altitude, it taken directly from the topographic map. For a skeleton's point, if it lies between two brush's points, then it will be the average of their neighbor points; if the point lies between points of the same skeleton, it will be a altitude proportional to the smallest distance to their neighbor points.

Algorithm 3 Assigning altitudes to each vertex

```
Input: This list of crust's points, $\mathcal{L}_c$; the list of skeleton's points, $\mathcal{L}_e$; the list of triangulation's
  vertices, $\mathcal{L}_t$.
Output: Altitude of each one of the vertices
  for each contour in $\mathcal{L}_c$ do {crust's points}
    Assign by hand the value of the altitude
  for each point $p$ in $\mathcal{L}_e$ do {skeleton's points}
    $\mathcal{L} \leftarrow$ neighborVertices( $p$, $\mathcal{L}_t$ )
    if vertices in $\mathcal{L}$ belongs to different contours in $\mathcal{L}_c$ then
      altitude of $p \leftarrow$ average ( altitudes in $\mathcal{L}$ )
    else if vertices in $\mathcal{L}$ belongs to only one contour in $\mathcal{L}_c$ then
      altitude of $p \leftarrow$ proportional to the smaller distance from $p$ to points in $\mathcal{L}$
  for each point $p$ in $\mathcal{L}_e$ without assigned altitude do
    $\mathcal{L} \leftarrow$ neighborVertices( $p$, $\mathcal{L}_t$ )
    altitude of $p \leftarrow$ proportional to distances from $p$ to $\mathcal{L}$
```

4 Practical example

In the upper part of Fig. 2 we see a binary input image. This image is the thinning contours of a montain taken from the INEGI's topographic map E14B47. In the middle of Fig. 2 we see a image of the contour interpolation, the blue lines are the crust, in red the skeleton, and in gray the triangulation. In the lower part of Fig. 2 a 3D view of the obtained 3D reconstruction is shown. Rendering was obtained with OpenGL.

5 Conclusions

We have proposed a methodology to perform the three-dimensional reconstruction of a terrain, using the geometrical forms of the crust and the skeleton. The problem of 3D reconstruction has been divided in two parts: one is solved using image processing techniques, and the other one by computational geometry. As input to the first part we use a scanned image of the contour lines that define a terrain, and as a output we obtain sampled points per each contour. The sampling density depends of the details in each curve. The second part use the geometrical constructions of the crust and the skeleton. The final visualization require to triangulate all points and assign altitudes to every one.

As future work we expect to add a user interaction with the 3D reconstruction: the user could hide slices, and clicking on the 3D map will supply basic geographic information.

Acknowledgments

This work has been partly supported by grant 45306 from the CONACyT, México.

References

1. S. Shinner, D. Meyers, and K. Sloan. Surface from contours. *ACM Transactions on Graphics*, 3(11):228–258, July 1992.
2. A.M. Felicísimo. *Modelos Digitales del Terreno: Introducción y Aplicaciones en las Ciencias Ambientales*. Pentalfa Ediciones, Oviedo, 1994.
3. Instituto Nacional de Estadística, Geografía e Informática (INEGI). http://www.inegi.gob.mx.
4. F. Dupont, M.P. Deseilligny, and M. Gondran. DMT extraction from topographic maps. In *Proceedings of Fifth International Conference on Document Analysis and Recognition. ICDAR'99*, pages 475–478, 1999.
5. N. Amenta, M. Bern, and D. Eppstein. The crust and the β-skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135, 1998.
6. C.M. Gold and M. Dakowicz. Visualizing terrain models from contours – plausible ridge, valley and slope estimation. In *Proceedings of the International Workshop on Visualization and Animation of Landscape*. Kumming, China, 2002.
7. D. Thibault and C. M. Gold. Terrain reconstruction from contours by skeleton construction. *GeoInformatica*, 4(4):349–373, 2000.

8. K. Hormann, S. Sapinello, and P. Schöder. C^1-continuous terrain reconstruction from sparce contours. In T. Ertl, B. Girod, G. Greiner, H. Niemann, H.-P. Seidel, E. Steinbach, and R. Westermann, editors, *Proceedings of Vision, Modeling, and Visualization 2003*, pages 289–297, München, Germany, November 2003. Infix.
9. Gonzalez R.C and R.E Woods. *Digital Image Processing*. Adisson-Wesley, 1992.
10. J. Cornejo Herrera, A. Lara López, R. Landa Becerra, and L.G. de la Fraga. Una librería para procesamiento de imagen: scimagen. In *VIII Conferencia de Ingeniería Eléctrica*, 4, 5 y 6 de septiembre, 2002. Cinvestav.
11. G. Borgefors. Distance tranformations in digital images. *Comp. V. Graph. and Image Process.*, 34:344–371, 1986.
12. C.B. Barber, D.P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hull. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

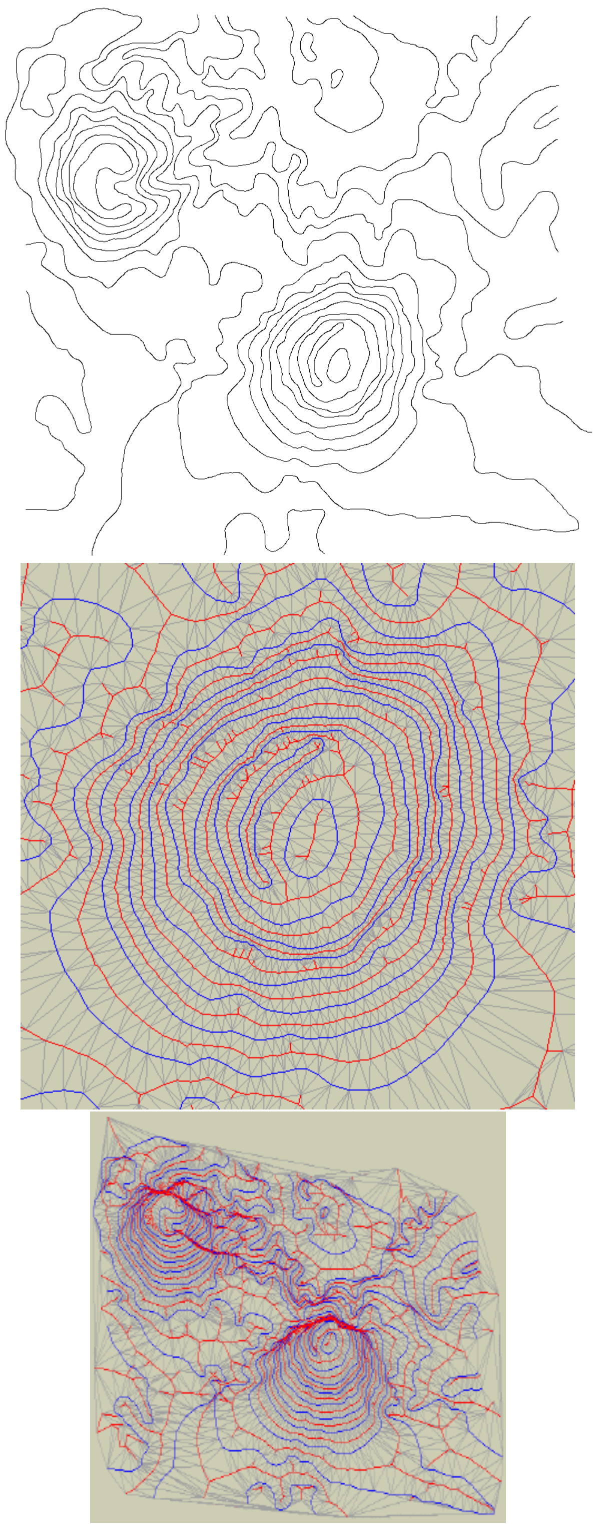

Figure 2: Application example images. See details in the text