Discrete 3D Tools Applied to 2D Grey-Level Images

Gabriella Sanniti di Baja[1], Ingela Nyström[2], and Gunilla Borgefors[3]

[1] Institute of Cybernetics "E.Caianiello", CNR, Pozzuoli, Italy
gsdb@imagm.cib.na.cnr.it
[2] Centre for Image Analysis, UU, Uppsala, Sweden
ingela@cb.uu.se
[3] Centre for Image Analysis, SLU, Uppsala, Sweden
gunilla@cb.uu.se

Abstract. 2D grey-level images are interpreted as 3D binary images, where the grey-level plays the role of the third coordinate. In this way, algorithms devised for 3D binary images can be used to analyse 2D grey-level images. Here, we present three such algorithms. The first algorithm smoothes a 2D grey-level image by flattening its geometrical and grey-level peaks while simultaneously filling in geometrical and grey-level valleys, regarded as non significant in the problem domain. The second algorithm computes an approximation of the convex hull of a 2D grey-level object, by building a covering polyhedron closely fitting the corresponding object in a 3D binary image. The result obtained is convex both from the geometrical and grey-level points of view. The third algorithm skeletonizes a 2D grey-level object by skeletonizing the top surface of the object in the corresponding 3D binary image.

1 Introduction

Algorithms for 3D voxel images are generally regarded as computationally more expensive than algorithms for 2D pixel images. This is certainly at least partially true both due to the large number of elements generally constituting a voxel image, and due to that a number of geometrical and topological problems arise when working with 3D objects that are not present in 2D space. However, there are cases when 3D algorithms can be conveniently used, even though the images to be analysed belong to the 2D space. This is the case, in our opinion, when working with 2D grey-level images. A 2D grey-level image can be interpreted as a terrain elevation map, where the grey-level of a pixel is interpreted as its height with respect to the grey-level of the background. By using this interpretation, we can pass from the input 2D grey-level image to a 3D binary image. In this way, algorithms developed for treating binary images can be used. These are generally computationally less expensive than algorithms taking into account also grey-level information. Moreover, some of the processing to be performed will be concerned mainly with the voxels belonging to the top surface of the so obtained solid object, which limits the number of elements to be processed notwithstanding the fact that generally a rather large number of voxels constitutes the 3D binary image corresponding to the 2D grey-level image.

In this paper, we convert a 2D grey-level image to a 3D binary image and present some 3D binary image analysis tools intended for analysing the 2D grey-level image. We assume that the grey-level image is already segmented, so that the regions corresponding to the grey-level objects can be distinguished from a uniform background.

We modify the algorithm suggested in [1] to smooth 3D objects in binary images, to actually smooth a 2D grey-level object. Geometrical and grey-level peaks of the object, regarded as non significant in the problem domain, are flattened and non significant geometrical and grey-level valleys are simultaneously filled in. The smoothing algorithm is based on the standard concept of erosion/dilation [2], but is here accomplished by using the distance transform of the object with respect to the background and the distance transform of the background with respect to the object. This smoothing algorithm is a useful tool per se and, in particular, it can be used before applying any of the remaining tools we introduce in this paper, namely convex hull approximation and skeletonization, to improve their performance.

We adapt the algorithm introduced in [3] to compute an approximation of the convex hull of 3D objects in binary images, to the case of 2D grey-level objects. A first attempt in this direction has recently been done [4]. The approximation of the convex hull is a covering polyhedron, closely fitting the object, obtained by iteratively filling local concavities in the 3D binary image. The obtained result is convex both from the geometrical point of view and as concerns grey-levels. The grey-level convex deficiency can be used to eliminate uneven illumination in grey-level images.

Using the algorithm suggested in [5] to compute the curve skeleton of surface-like objects in binary images and its modification introduced in [6], we present an algorithm to skeletonize 2D grey-level objects by actually skeletonizing the top surface of the 3D objects in the corresponding binary images. Voxels in the top surface of a 3D object are examined in the order of their distances from the initial border of the surface, which limits the computation time. Moreover, skeleton branches will be placed along significant (geometrical and grey-level) ridges of the 2D grey-level object, since the skeletonization of the 3D surfaces takes into account geometric information.

Our algorithms are based on the use of local operations and, except for skeletonization which requires a number of scans dependent on the size of the top surface of the object in the converted 3D image, all computations are done in a fixed and small number of scans. Therefore, they have reasonable computational complexity and short processing times on standard computers. We have tested our algorithms on a variety of images and the obtained results have been satisfactory. A comparison with other algorithms for 2D grey-level smoothing and skeletonization remains to be done. The method for a grey-level convex hull is unique.

2 Conversion Between 2D Grey-Level and 3D Binary Images

We start with an $M \times N$ 2D grey-level image G. For simplicity, we assume that only one object is present in G and that the background is uniform. The pixels in G belong to $[0,K]$, typically, $K = 255$. Object pixels range from 1 to K, while background pixels are 0. To build an $M \times N \times (K + 1)$ binary image I, for each pixel at position (x,y) in G with grey-level g, all voxels at positions (x,y,z) in I with $1 \leq z \leq g$ are set to 1 (object), while all voxels at positions (x,y,z) with $g + 1 \leq z \leq K$ and all voxels with z=0 are set to 0 (background). Note that since grey-level is not equivalent to height, the correspondence between grey-level g and z value is not given. It depends on the specific application. Linear scaling may be necessary. Each pixel (voxel) v in G (I) has two (three) types of neighbours called edge- or vertex- (or face-) neighbours, depending on whether they share an edge or a vertex (or a face) with v.

The obtained 3D image undergoes image analysis. Whenever the process is completed on the 3D binary image, the so resulting image can be converted to a 2D grey-level image by projection. This process is straightforward.

3 Grey-Level Image Smoothing

Smoothing can be done by erosion/dilation operations [2], or by distance transform based methods [1]. We favour the latter approach in particular as it allows us to smooth an object by simultaneously removing its protrusions, cavities and concavities of negligible size. To achieve this, the distance transform of the whole 3D image I is computed and, to save computation time, the distance transform of the object and the distance transform of the background are computed simultaneously.

Let w_f, w_e, and w_v denote the local distances from any voxel to its face-, edge-, and vertex-neighbours, respectively. The distance transform is computed by propagating local distance information during a forward and a backward scan of the image. Each inspected voxel v is set to the minimum of the value of the voxel itself and the values of its already visited neighbours increased by the local distances (w_f, w_e, or w_v) to the voxel v. Let *min* denote such a minimum. To distinguish between object and background, we ascribe positive sign to distance values computed for object voxels, and negative sign to distance values computed for background voxels. Initially, background voxels are set to -infinity and object voxels to +infinity. During the forward scan, every voxel v in the object (background) is set to:

w_f ($-w_f$), if v has a negative (positive) face-neighbour;

w_e ($-w_e$) if v has a negative (positive) edge-neighbour, but all its face-neighbours are positive (negative);

w_v ($-w_v$) if v has a negative (positive) vertex-neighbour, but all its face- and edge-neighbours are positive (negative);

min (-*min*), otherwise.

During the backward scan, only voxels with values different from $\pm w_f$, $\pm w_e$, and $\pm w_v$ are examined and are assigned value $\pm$*min*. Obviously, absolute values are taken into account to correctly compute the *min* for the background voxels.

The obtained distance transform of the image is "shrunk" by setting all voxels that are less than or equal to a chosen threshold h_o for the object, and h_b for the background to zero. In this way, a hollow space is created around the border of the object. To smooth the object, a new border should be placed midway in the hollow space. A computationally convenient way to reach this goal is to compute the distance transform of the hollow space with respect to its complement. Again, positive and negative signs are used so that we can distinguish distance information, depending on whether it comes from object voxels or background voxels. Object voxels are initially set to +1, background voxels to -1 and voxels in the hollow space are set to +infinity. When a voxel in the hollow space has the same distance from both positive and negative voxels, a decision should be taken on the sign to be assigned to the voxel itself. Choosing a negative sign favours smoothing of protrusions, while choosing a positive sign favours removal of concavities. In our case, we favour protrusion flattening and hence choose the negative sign. Once the distance transform of the hollow space is computed, the image is binarized by setting all voxels with negative value to 0 (i.e., to background), and all voxels with positive value to 1 (i.e., to object).

The two thresholds h_o and h_b are set depending on the size of the largest protrusions or concavities to be smoothed. Equal values for the two thresholds are selected to avoid that the border of the object is altered wherever smoothing is not necessary and to keep the size of the object approximately constant. Any distance transform, i.e., any choice of the local distances w_f, w_e, and w_v is possible. Good results are obtained with w_f=3, w=4, and w_v=5, [7].

A synthetic example showing the effect of smoothing is given in Fig. 1, where white is used for the background, h_o=h_b=3 and w_f=3, w_e=4, and w_v=5. A 2D noisy grey-level image, a), consisting of four square frames, each with different width and with pixels all having uniform grey-level except for a few sparse pixels with either higher or lower grey-level, is converted to the 3D binary image, b), where different grey tones are used only to help readers to associate the regions of the 3D image with the corresponding regions in the 2D image. Thin spikes (and thin elongated concavities) are the 3D visible effect of noise. In the 3D image resulting after smoothing, c), spikes and concavities have been flattened and filled in, respectively. Moreover, 90-degree corners, created in the 3D image due to the sharp change of grey-level in neighbouring frames of the 2D image, have been smoothed. The 3D image is projected to provide the resulting 2D grey-level smoothed image, d).

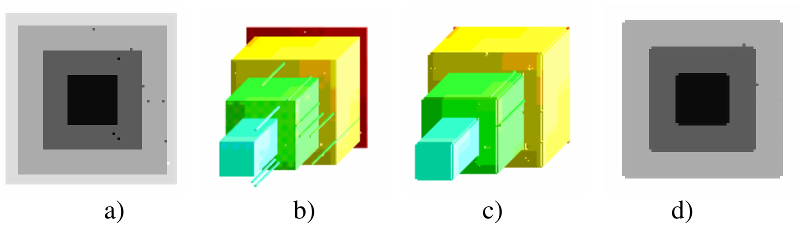

Fig. 1. A noisy 2D grey-level image, a), the converted 3D binary image, b), the smoothed 3D binary image, c), and the smoothed 2D grey-level image, d), obtained by projection

Smoothing can be used in general to improve the image quality. In particular, its use before convex hull computation and skeletonization allows us to obtain smoother covering polyhedra and skeletons with simpler structure. In the following, we use as running example a grey-level digit 6. The effect of smoothing can be seen in Fig. 2.

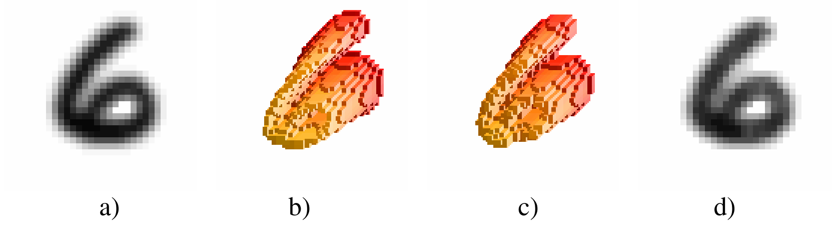

Fig. 2. The original digit 6, a) and b), and the smoothed digit 6, c) and d)

4 Grey-Level Convex Hull Computation

The convex hull of an object is the smallest convex set enclosing the object [8]. For a 2D grey-level image, one could compute the convex hull by taking into account either only geometric information, or also adding grey-level information. What is the best choice depends on the application. Here, we use both geometric and grey-level information and suggest a purely discrete and local approach to compute a sufficiently good approximation of the convex hull of a 2D grey-level object in the corresponding 3D binary image [4]. Our algorithm builds a covering polyhedron, closely fitting the object, by iteratively filling local concavities, [3]. Local concavities are defined as border voxels (i.e., background voxels with at least one face-neighbouring object voxel) with a certain number of neighbouring object voxels. The larger the neighbourhood used to identify local concavities is, the more half-spaces can be used for delimiting the covering polyhedron, and the better the approximation of the convex hull will be. Naturally, the computational cost increases with the size of the adopted neighbourhood. If the 3×3×3 neighbourhood, is used, the computation cost of the algorithm is the smallest possible, but the polyhedron would have at most 26 faces. This is a too rough approximation of the convex hull. On the other hand, resorting to larger neighbourhoods and larger operators, would make the algorithm significantly more heavy.

We have found a good compromise between computational cost and quality of the obtained results. Our method still uses only 3×3×3 operators, but curvature information is derived from the 5×5×5 neighbourhood. This goal is achieved by splitting each iteration in two subiterations. During the first subiteration we count, for each border voxel v, the number of its object face- and edge-neighbours, individually in each of the x-, y-, and z-planes, and store the three resulting sums, S_x, S_y and S_z, as a vector label for v. Then, during the second subiteration, any voxel v with at least one S_k>4 (k = x, y, z), or such that one S_k=4 and having, in the same plane k, at least one neighbour with S_k>2 is defined as being a local concavity and changed from background to object. In this way, a local concavity is filled. By iteratively filling local concavities, the concavity regions are filled globally: The filling process terminates when all border voxels belong to planes oriented along permitted orientations. The resulting covering polyhedron is convex and can have up to 90 faces.

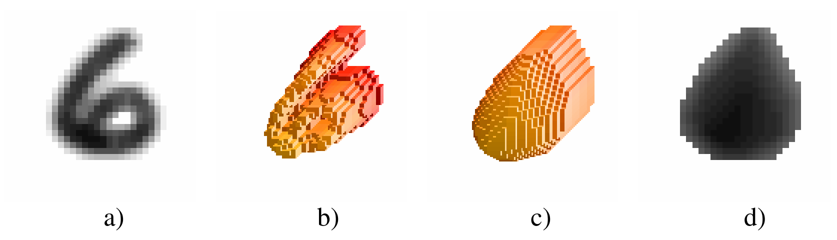

Fig. 3. The 3D covering polyhedron, c), for the smoothed digit 6, a) and b), and the 2D grey-level (approximation of the) convex hull, d), obtained by projection.

The difference between our covering polyhedron and the convex hull is reasonably small, and the covering polyhedron is good enough for most practical purposes. When the binary 3D image is converted to 2D, the resulting convex hull is a grey-level set where both geometric (planar) concavities and grey-level concavities (areas with lower grey-level than their surroundings) of the object are filled.

The performance of the algorithm on the running example can be seen in Fig. 3.

5 Grey-Level Skeletonization

The skeleton of a grey-level object is a thin subset of the object, centered within the regions of the object with locally higher intensity, and having the same topology as the object. Literature on skeletonization is rich for binary images (see, e.g., [9]), but not equally rich for grey-level images (see [10] for an early example). To compute the grey-level skeleton of a 2D object, we use the algorithm introduced in [5] to compute the skeleton of a 3D surface-like object in a binary image. The algorithm is based on the intuitive classification of all voxels belonging to a surface-like object into curve, junction, inner, and edge voxels (see [5] for more details). The advantage of this algorithm is that it uses geometrical information to automatically ascribe to the curve skeleton the voxels that will be its end-points. In fact, it is well known that a critical step in skeletonization algorithms is the correct identification of end points.

We compute the curve-skeleton of the top surface of the 3D object corresponding to the 2D grey-level object [6]. For each (x,y), the voxel with maximal z value belongs to the top surface. However, there is no guarantee that this set is a connected and tunnel-free surface. Thus, we define the top surface of the object as the set consisting of all object voxels in positions (x,y,z) with $z \neq 0$ and having a face-neighbour in the background. The so defined top surface is generally 1-voxel thick, but it may be 2-voxel thick in presence of surface parts forming 90-degree corners. This is typically the case when in the 2D grey-level image a region with constant grey-level g is adjacent to a region whose grey-level is equal to or smaller than g-2 (see, e.g., Fig. 1). These 90-degree corners would be classified as consisting of junction voxels by the classification method used within the skeletonization algorithm, while they actually correspond to an artefact created during the conversion from 2D to 3D. Since the skeletonization algorithm does not remove voxels classified as belonging to junctions, we should remove from the top surface these spurious junctions consisting of corner voxels to obtain a significant skeleton. Indeed, most of these spurious junctions are smoothed by the procedure illustrated in Section 3. Corner voxels still remaining in the top surface are assigned to the background, which is equivalent to decreasing the grey-level g by 1 in the corresponding positions.

The curve skeleton is computed by means of an iterative process. At each iteration, the voxels that are currently classified as curve or edge voxels are identified as border voxels. Among them, only the edge voxels are removed, provided that their removal does not alter topology. In this way, curve voxels (classified as such in the initial top surface or at any successive stage of the process) are automatically preserved so that unwanted shortening of skeleton branches is avoided.

When junction voxels are identified in the initial classification of the top surface, skeletonization is done in two steps to guarantee that the most significant shape information remains in the resulting skeleton. Voxels classified as junction voxels in the input top surface may be transformed into curve voxels, inner voxels or edge voxels during the removal process. In the first case (curve voxels), they are directly ascribed to the curve skeleton; in the second case (inner voxels), they cannot be considered for removal; finally, in the third case (edge voxels), their removal is possible, but it is done only during the second step of the process. This step starts when all voxels currently classified as edge voxels are those that were classified as junction voxels in the input surface. To this purpose, during the first step, both the current classification and the initial classification are taken into account, iteration after iteration. Voxels, identified as edge voxels according to the current classification, can be removed only if they were not classified as junction voxels in the initial classification. During the second step, only the current classification is used. The resulting set is a nearly-thin skeleton, i.e., an at most 2-voxel thick set.

Final thinning of the curve skeleton can be performed by using standard topology preserving removal operations, e.g., based on the identification of simple voxels [11]. To prevent excessive branch shortening, we split the final thinning into six directional processes (top, back, right, down, front, left), each of which reduces the thickness of the skeleton in the corresponding scanning direction.

Before converting the 3D image to 2D, the 3D curve skeleton is pruned. The classification into edge, curve, inner, and junction voxels done on the top surface is used also to perform pruning without significant loss of shape information. In fact, voxels classified as curve or junction voxels in the top surface are considered as those retaining the most significant shape information and, hence, are marked as significant voxels. Only branches including a small number of significant voxels compared to the total number of voxels in the branch are pruned. Pruning is active only on peripheral branches (i.e., branches delimited by an end point and a branch point). When the structure of the curve skeleton is complex, pruning only peripheral branches may not produce satisfactory results. In fact, internal branches, i.e., branches delimited by two branch points in the initial curve skeleton, may include a large number of non-significant voxels with respect to the total number of voxels. These branches may become peripheral only after removal of adjacent peripheral branches. In such a case, pruning is iterated. Of course, information about the already removed branches is used, to avoid an excessive shortening due to pruning iteration.

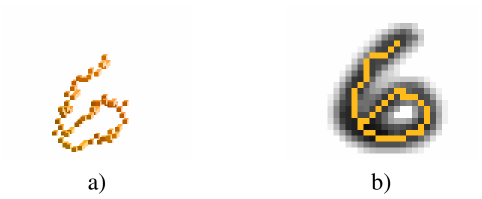

Fig. 4. The 3D curve skeleton of the smoothed digit 6, a), and the 2D grey-level skeleton, superimposed on the original object, b), obtained by projection

Before applying context dependent pruning, a brute force removal of branches including only a few voxels is performed. These short branches are interpreted as noisy branches even if they include almost only significant voxels. Their presence in the skeleton would postpone to the second iteration of pruning, or even completely prevent, removal of other longer but still non-significant branches.

At this point, the 3D binary image can be converted to the 2D grey-level skeleton image. In general, final thinning and postprocessing (spurious loop filling and, possibly, pruning) have to be applied on the so obtained 2D skeleton to reduce its business and to achieve a topologically correct result. In fact, 3D curves, separate but close to each other, may originate touching 2D curves, which may produce thickening and spurious loops in the resulting 2D skeleton. The performance of the skeletonization algorithm on the running example is shown in Fig. 4.

6 Conclusion

In this paper, we have shown the results obtained by applying three algorithms, initially intended for 3D binary images, to 2D grey-level images. The 2D grey-level image is initially converted to a 3D binary image, by using the grey-level as the third coordinate. The 3D binary algorithms have been applied to this image and the results have been projected back to 2D to obtain the grey-level resulting images.

The results obtained are promising and we plan to continue along this direction.

References

1. G. Sanniti di Baja, S. Svensson, “Editing 3D binary images using distance transforms”, *Proc. 15th ICPR*, Barcelona, Spain, pp. 1034-1037, 2000.
2. J. Serra, “Image Analysis and Mathematical Morphology” Vol. I, Academic Press, London, 1982
3. G. Borgefors, I. Nyström, G. Sanniti di Baja, “Computing covering polyhedra of non-convex objects”, *Proc. of BMVC94*, York, 275-284, 1994.
4. I. Nyström, G. Borgefors, G. Sanniti di Baja, “2D Grey-level Convex Hull Computation: A Discrete 3D Approach”, in *Image Analysis,* J. Bigun and T.Gustavsson Eds., LNCS 2749, Springer Verlag, Berlin, pp. 763-770, 2003.
5. S. Svensson , I. Nyström, G. Sanniti di Baja, “Curve skeletonization of surface-like objects in 3D images guided by voxel classification”, *Pattern Recognition Letters*, 23/12 pp. 1419-1426, 2002.
6. G. Sanniti di Baja, I. Nyström “2D Grey-level Skeleton Computation: A Discrete 3D Approach”, *Proc. 17 ICPR*, Cambridge, UK, pp. 455-458, 2004.
7. G. Borgefors, “On digital distance transforms in three dimensions”, *Computer Vision Image Understanding* 64/3, pp. 368-376, 1996.
8. F.P. Preparata, M.I. Shamos, “Computational Geometry. An Introduction”, Springer Verlag, New York, 1985.
9. L. Lam, S-W. Lee, C.Y. Suen, “Thinning methodologies. A comprehensive survey”, *IEEE Trans. on PAMI*, 14/9 pp. 869-885, 1992.
10. B.J.H. Verwer, L.J. Van Vliet, P.W. Verbeek, “Binary and grey-value skeletons: Metrics and algorithms”, *IJPRAI,* 7 pp. 1287-1308, 1993.
11. P.K. Saha, B.B. Chaudhuri, “Detection of 3-D simple points for topology preserving transformations with application to thinning”, *IEEE Trans. on PAMI*, 16/10 pp. 1028–1032, 1994.