Efficient Computation of 3D Skeletons by Extreme Vertex Encoding

Jorge Rodríguez[1], Federico Thomas[2], Dolors Ayala[1], and Lluís Ros[2]

[1] Technical University of Catalonia (UPC), Computer Science Department (LSI),
Diagonal 647, 8 planta, 08028 Barcelona, Spain
{jrodri, dolorsa}@lsi.upc.es
[2] Industrial Robotics Institute (CSIC-UPC),
Llorens Artigas 4-6, 2 planta, 08028 Barcelona, Spain
{fthomas, llros}@iri.upc.es

Abstract. Many skeletonisation algorithms for discrete volumes have been proposed. Despite its simplicity, the one given here still has many theoretically favorable properties. Actually, it provides a connected surface skeleton that allows shapes to be reconstructed with bounded error. It is based on the application of directional erosions, while retaining those voxels that introduce disconnections. This strategy is proved to be specially well-suited for extreme vertex encoded volumes, leading to a fast thinning algorithm.

Keywords. 3D surface skeleton, mathematical morphology, extreme vertex encoding.

1 Introduction

The computation of three-dimensional skeletons is a fundamental tool for an increasing number of applications related to shape matching and tracking, virtual navigation, shape abstraction, animation control, growth modelling, analysis of symmetries, path planning, feature recognition, etc. Actually, the skeleton of a solid has been proposed as an alternative to its boundary or constructive solid model because it provides a complete representation [6].

The word *skeleton* is usually understood in 2D to mean the medial axis of a given shape. The *medial surface* of a 3D solid is defined similarly to its 2D counterpart: it is the set of the centers of all inscribed spheres of maximal radius. The computation of the medial surface for arbitrary solids is a complex problem. Herein we concentrate ourselves in the computation of skeletons of discrete solids, i.e. objects described in terms of sets of voxels. When working in a discrete space, spheres must be necessarily approximated and, as a consequence, the concept of skeleton should be redefined. Then, 3D discrete skeletons are approximations of the medial surface. In any case, the three common required properties for any of these approximations are [15]: (a) it should have no interior (thinness); (b) it must be homotopic to the object it corresponds to, i.e. it must preserve 26-neighbor connectedness for the foreground and 6-neighbor connectedness for

the background (connectedness); and (c) it must retain sufficient information to reconstruct the original object (reconstructability). Other interesting criteria for some applications have been recently introduced [17], but it is important to realize that in the discrete space all these requirements become mutually incompatible [7], and, as a consequence, practical skeletonisation methods are invariably a compromise between them.

Although other alternatives are possible [17], two main methods to obtain skeletons in discrete spaces have been proposed: methods based on thinning and methods based on distance transforms. Thinning methods produce skeletons by iteratively deleting voxels from the boundary of the solid. The deletion of a voxel can be in a sequential algorithm or parallel one. Each iteration of sequential algorithms consists, in general, of three steps: (1) identify all border voxels, and label them with the iteration number; (2) inspect all voxels labelled with the current iteration number, and mark those that cannot be removed in order to preserve the shape of the original solid; and (3) remove all unmarked border voxels. An example of algorithm using this technique can be found in [12]. The methods based on distance transforms first convert the volume, which consists of object (foreground) and non-object (background) voxels, into an object where every object voxel has the value corresponding to the minimum distance to the background. Different types of metrics for discrete solids are used, aiming at approximating the euclidean distance so that rotation-invariance is attained to some extend. Then, the ridges of the induced scalar field constitute the skeleton. In general, these algorithms are not iterative, so that the skeleton is produced in a fixed number of passes through the solid. An example of algorithm using this technique can be found in [11].

The procedure proposed here can be outlined as follows. Those voxels whose deletion by a directional erosion might destroy the connectedness are retained and classified as *gaps*, then the region is eroded and the corresponding *residuals* computed. Gaps and residuals are retained in the solid, and this process is repeated until no progress is made. This approach satisfies the requirements given above in the following order of priority: connectedness, thinness and reconstructability. The preservation of the connectivity is the essential condition for the skeleton in order to extract the shape of the original solid. In our case, shapes can be nearly reconstructed with an error bounded to one voxel. This strategy was first introduced in [7] for 2D objects. A similar approach has recently been presented in [12], but using the classical template-based thinning approach. In our case, we avoid this explicit use of templates by reducing all the steps of the presented algorithm to boolean operations between solids.

This paper is structured as follows. In order to make it as self-contained as possible, the required morphological operations, as well as the concepts of residuals and gaps associated with directional erosions, are introduced in Section 2. The skeletonisation algorithm is presented in Section 3. The adopted spatial encoding and how boolean operations can be performed between encoded objects is described in Section 4. The results, showing how this encoding improves the performance of the algorithm are detailed in Section 5.

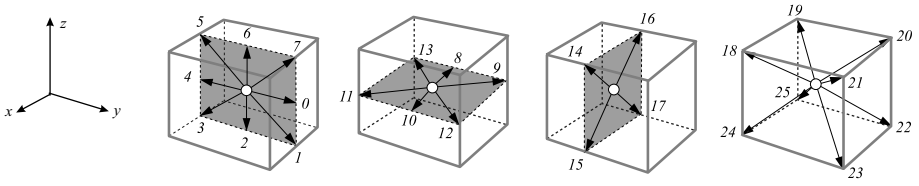

Fig. 1. Numbering assignment to the 26 neighbors of a voxel.

2 Background

Let Z^3 be the discrete space. Let $X \subset Z^3$ be a 3D solid. Let $\bar{X} = Z^3 \backslash X$ denote the background of X. The connectivity used herein is (26,6)-connectivity, which means 26-connectivity for the solid and 6-connectivity for the background. Each of the 26 neighbors of a voxel in the solid defines a vector which will be numbered as shown in figure 1.

The *erosion* of X using the structuring element B is defined as $X \ominus B = \{y | \forall b \in B, y+b \in X\}$, its *dilation* using the same structuring element as $X \oplus B = \{y | y = x + b, x \in X, b \in B\}$, and its *opening* as $XoB = ((X \ominus B) \oplus B)$.

The *residual*, $X \perp B$, is the set made of those voxels in X which do not belong to its opening using the structuring element B, that is, $X \perp B = X \backslash (XoB)$.

If X_b denotes the translation of X in the direction associated with $b \in B$, then it can be shown that $X \ominus B = \bigcap_{b \in B} X_{-b}$. In other words, this erosion can be accomplished by taking the intersection of all the translates of X, where the shifts in the translates are the negated members of B, seen as vectors.

An especially interesting case for B is that in which B consists of two voxels, where one is centered in the origin. Then, the erosion of X using B can be computed simply by $X \ominus B = X \cap X_{-b}$, and its opening by $XoB = (X \cap X_{-b}) \cup (X \cap X_{-b})_b$. Since $(B1 \ominus B2) \ominus B3 = B1 \ominus (B2 \oplus B3)$, then, if $B = B1 \oplus B2 \oplus \cdots \oplus Bk$, one concludes that $X \ominus B = (\ldots[(X \ominus B1) \ominus B2] \ominus \cdots \ominus Bk)$. Thus, if a structuring element can be broken down to a chain of dilations of smaller substructuring elements, the desired operation may be performed as a sequence of suboperations.

As a first approximation, a skeleton can be defined as the set of all the residuals of the successive erosions of X, using the following simple algorithm:

algorithm T1;
input: X;
output: S;
$S \leftarrow \emptyset$;
while $X \neq \emptyset$
    $E \leftarrow X \ominus B$;
    $S \leftarrow S \cup (X \backslash (E \oplus B))$;
    $X \leftarrow E$;
endwhile;
end.

Now, let us assume that B is a centered 3x3x3 cubic structuring element which can be broken down into a chain of 6 dilations of two-voxel elements in the directions 0, 2, 4, 6, 8, and 10. Then, the above algorithm can be rewritten as follows:

algorithm T2;
input: X;
output: S;
$S \leftarrow \emptyset$;
while $X \neq \emptyset$
    $E \leftarrow X \cap X_0 \cap X_2 \cap X_4 \cap X_6 \cap X_8 \cap X_{10}$;
    $S \leftarrow S \cup (X \backslash (E \cup E_4 \cup E_2 \cup E_8 \cup E_0 \cup E_6 \cup E_{10}))$;
    $X \leftarrow E$;
endwhile;
end.

The main advantage of this algorithm over T1 is that it only involves directional erosions and dilations along the coordinate axes. Although the skeleton that it obtains allows to entirely reconstruct the initial shape by simply dilating each voxel of the result (according to the iteration in which it was obtained), it is neither thin nor connectivity preserving. The first drawback can be easily overcome as follows:

algorithm T3;
input: X;
output: S;
$S \leftarrow \emptyset$;
$A \leftarrow \emptyset$;
while $X \neq \emptyset$
    /* directional erosion along y+ */
    $E \leftarrow X \cap X_0$;
    $A \leftarrow A \cup (X \backslash (E \cup E_4))$;
    $X \leftarrow E$;
    /* repeat for directions z+, x+, y-, z-, and x- */
    $\vdots$
    $S \leftarrow S + A$;
endwhile;
end.

Now, since residuals are independently thin (they are obtained from single directional erosions), the obtained skeleton is thin. As a counterpart, the original shape can only be nearly reconstructed but, as it has already been pointed out, thinness and reconstructability are mutually incompatible goals. Then, shapes can be nearly reconstructed with an error, in our case, bounded to one voxel.

In order to overcome the remaining drawback (connectivity) we first introduce the concept of directional gaps. Those voxels required to ensure connectivity in the final skeleton and not included in the medial surface computed by algorithm T3, will be part of a set of disjoint regions that we call *gaps*. Contrarily to what one might expect, when considering only directional erosions, gaps can

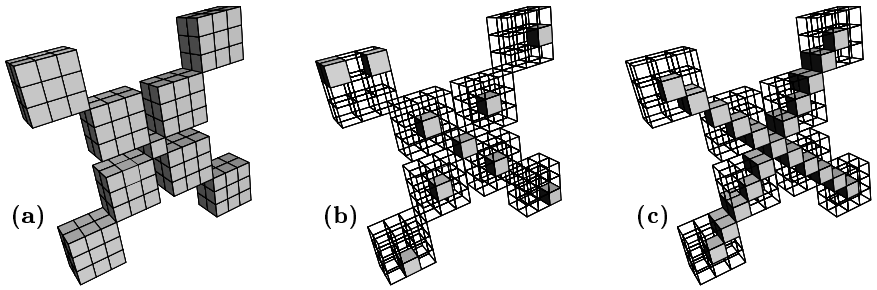

Fig. 3. Result of the application of T3 (b), and T4 (c) on the solid in (a). The input solid is shown in wireframe.

be easily computed. For example, the directional gap of a binary region X in direction $y+$ can be obtained by computing:

$$\begin{aligned}(X\backslash X_0) \cap ((X_7\backslash X_6) \cup (X_1\backslash X_2) \cup (X_{20}\backslash X_{16}) \cup (X_{22}\backslash X_{17}) \\ \cap (X_{21}\backslash X_{14}) \cup (X_{23}\backslash X_{15}) \cup (X_{12}\backslash X_{10}) \cup (X_9\backslash X_8))\,.\end{aligned} \tag{1}$$

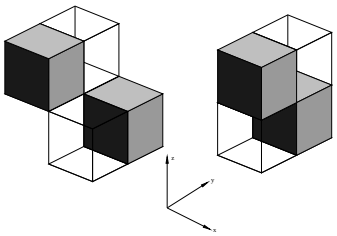

Fig. 2. Gaps.

This boolean formula detects the voxel configurations shown in figure 2 and all its $\pi/2$ rotated versions around the y axis. While grey cubes correspond to points in the foreground, transparent ones correspond to those in the background.

Gaps along the other coordinate axes, either in positive or negative directions, can analogously be obtained. The above expression is obtained as a generalization of the two-dimensional case. It is worth noting that the concept of gaps, first introduced in [7], is closely related to the set of β templates presented in [12].

3 The thinning algorithm

The motivation behind our thinning algorithm is seen as follows. First those voxels whose deletion by a directional erosion might destroy the connectedness are retained and classified as gaps, then the region is effectively eroded and the corresponding residual computed. Gaps and residuals are removed from the solid in order to concentrate the thinning effort on the thick region. Iterations continue until the solid becomes empty. The following algorithm in pseudo-code implements this procedure.

algorithm T4
input: X;

output: S; /* skeleton of X*/
$S \leftarrow \emptyset$;
$L \leftarrow \emptyset$;
do
  /* erosion along y+ */
  $I \leftarrow \emptyset$; /* increment of skeleton */
  $G \leftarrow (X \backslash X_0) \cap [(X_7 \backslash X_6) \cup (X_1 \backslash X_2) \cup \cdots \cup (X_9 \backslash X_8)]$; /* gap */
  $E \leftarrow X \cap X_0$; /* eroded solid */
  $R \leftarrow X \backslash (E \cup E_4)$; /* residual */
  $I \leftarrow I \cup R \cup G$;
  $X \leftarrow E \cup I$;
  /* repeat for directions z+, x+, y-, z-, and x- */
  ...
  $X \leftarrow X \backslash L$;
  $S \leftarrow S \cup L$;
  $L \leftarrow I \backslash L$;
until $(X == \emptyset)$;
end.

Note that the number of iterations is exactly half the 1-1-1 maximum thickness of the volume. This is perhaps the simplest thinning algorithm directly expressed in terms of boolean operations that provides a connected single-pixel-in-width well-centered homotopic skeleton.

Figure 3 shows a solid and the obtained skeletons using algorithms T3 and T4. The thinning action of T3 causes clear topological changes. The introduction of gaps by T4 fixes the problem. Moreover, it avoids the explicit consideration of a huge number of 3D templates, characterizing deletable voxels.

Finally, it is worth to highlight three properties of the proposed algorithm. First, each pixel of the skeleton can be labelled with its 1-1-1 distance to its nearest volume border by storing the iteration in which it was obtained. Second, the iterations continue until a volume becomes empty, which is faster than detecting idempotence. Third, the thinning effort is concentrated on the thick volume; in other words once a thin volume is obtained, it is removed from the image and hence it is no longer considered. This is of interest when processing spatially-encoded objects, as described in the next section.

4 Extreme Vertex Encoding

Any subset of $\mathbf{Z}^3$ is geometrically analogue to an orthogonal pseudo-polyhedron (OPP), that is, a polyhedron with all its faces oriented according to the three orthogonal coordinate axes and with a non-manifold boundary [9]. Figure 4(a) shows an OPP with seven non-manifold vertices and three non-manifold edges.

Let P be an OPP and Π_c a plane whose normal is parallel, without loss of generality, to the X axis, intersecting it at $x = c$, where c ranges from $-\infty$ to ∞. Then, this plane sweeps the whole space as c varies within its range, intersecting P at some intervals. Let us assume that this intersection changes at $c = c_1, \ldots, c_n$. More formally, $P \cap \Pi_{c_i - \delta} \neq P \cap \Pi_{c_i + \delta}$, $i = 1, \ldots, n$, where δ is

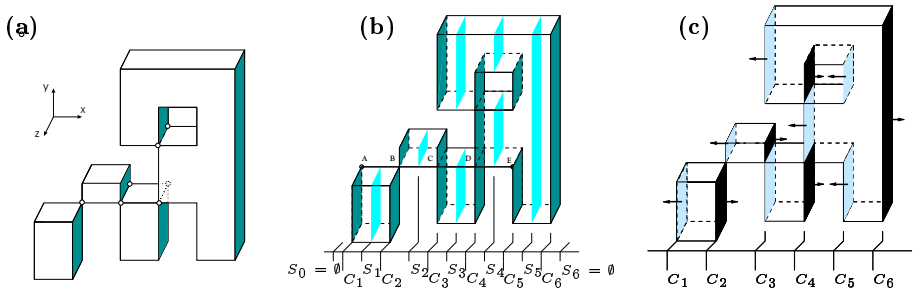

Fig. 4. **(a)** An OPP with three non-manifold edges and seven non-manifold vertices. **(b)** A brink from vertex A to vertex E where cuts and sections perpendicular to the X axis are shown in dark and light grey, respectively. **(c)** Forward and backward differences in black and light grey, respectively.

an arbitrarily small quantity. Then, $C_i(P) = P \cap \Pi_{c_i}$ is called a *cut* of P and $S_i(P) = P \cap \Pi_{c_s}$, $c_i < c_s < c_{i+1}$, is called a *section* of P. Figure 4(b) shows an OPP with its cuts and sections perpendicular to the X axis.

Since we work with bounded regions, $S_0(P) = P \cap \Pi_{-\infty} = \emptyset$ and $S_n(P) = P \cap \Pi_{\infty} = \emptyset$, n being the total number of cuts along a given coordinate axis.

The concept of cuts and sections can be extended to pseudopolygons. Hence, we can compute the cuts and sections of $C_i(P)$ by intersecting it with a sweeping line. Each resulting 1D cut is, in general, a set of disjoint segments. Each of these segments is called a *brink*. Actually, a brink is a maximal uninterrupted segment built out of a sequence of collinear edges of P. The ending vertices of a brink are called *extreme vertices*. Figure 4(b) shows an OPP with a brink going from vertex A to vertex E and traversing the non-extreme vertices B, C and D.

The organization of extreme vertices of an OPP in terms of all its brinks parallel to a given coordinate axis, then in terms of 1D cuts and, finally, in terms of 2D cuts is called an *extreme vertex encoding* [2]. This codification can obviously be done in six different possible ways depending on the chosen sequence of coordinate axis: XYZ, XZY, YXZ, YZX, ZXY, or ZYX. For example, in an XYZ ordering, the 2D cuts are perpendicular to the X axis and ordered from low to high x values and, for each of them, the 1D cuts are parallel to the Y axis and ordered from low to high y values.

Sections can be computed from cuts and vice versa by noting that:

$$S_i(P) = S_{i-1}(P) \otimes C_i(P), \text{ and } C_i(P) = S_{i-1}(P) \otimes S_i(P),$$

for $i = 1 \ldots n$, where $\otimes$ denotes the regularized XOR operation. Applying the definition of $\otimes$ in the above expression for C_i, we get

$$C_i(P) = S_{i-1}(P) \otimes S_i(P) = (S_{i-1}(P) \backslash S_i(P)) \cup (S_i(P) \backslash S_{i-1}(P)),$$

for $i = 1 \ldots n$. As a consequence, we can decompose any cut into two terms named *forward difference* and *backward difference* defined as $F_i(P) = S_{i-1}(P) \backslash S_i(P)$ and $B_i(P) = S_i(P) \backslash S_{i-1}(P)$, respectively. Figure 4(c) shows an OPP with its

sections perpendicular to the X axis, together with the corresponding forward and backward differences. It can be checked that $F_i(P)$ and $B_i(P)$ coincide with sets of faces of P with normals pointing forward and backward, respectively. This guarantees that the correct orientation of all faces of P can be obtained from its extreme vertex encoding. This together with the fact that sections can be computed from cuts allows to prove the completeness of this codification as a boundary model [3], [4]. Moreover, its efficiency in terms of memory requirements, compared to that of the semiboundary representation [16], which is usually considered as the most efficient block-form representation, has been shown to be clearly favorable [1].

Arbitrary boolean operations between OPPs can be carried out by applying recursively the same operation over the corresponding OPP sections which can be reduced to XOR operations between the extreme vertex encodings of both operands [2].

The computation of gaps in algorithm T4 consists of successive boolean operations between the input object, X, and a displaced version of itself, X_d, $d = 1, \ldots, 25$. The direct implementation of these operations requires computing the extreme vertex encoding of all possibly displaced operands. For example computing expression (1) would require computing 17 extreme vertex encodings. Fortunately, this can be avoided so that displacements of an object are taken into account, introducing the proper indices, directly when operating with them. In practice this means that a directional erosion or dilation takes the same time as an AND or OR operation.

5 Results and conclusions

The above algorithms have been implemented in C on a Sun Ultra2-2200 workstation with one processor running at 360 Mhz and 4 Mbytes of cache memory. It has been criticized that in many papers on skeletonisation of solid objects the only given examples are tiny test images, which makes it difficult to understand what would be the results for reasonable sized real objects [5]. In our case, T4 has been tested on 3-D images obtained using a CAT device. Figure 5 shows the result for a human vertebra. Before skeletonisation, it contains 22263 voxels. The memory required to store it using its extreme vertex encoding is 26 Kbytes, instead of 59 Kbytes required by the semiboundary codification. The total number of extreme vertices for this model is 6028, that is, 3014 brinks. The skeleton is computed in 1004 seconds if the codifications of the shifted objects are explicitly computed before operating with them. This time drops to 399 seconds when these computations are avoided. Table I shows the evolution of the processing time for each iteration using both approaches in separated columns. Note the reduction in the time required for each iteration thanks to the adopted incremental strategy that concentrates the computational effort in the thick regions. The last column of Table I gives an idea of the complexity of these thick regions in terms of their total number of brinks before the corresponding iteration. The resulting skeleton contains 6059 voxels, encoded using 3387 brinks.

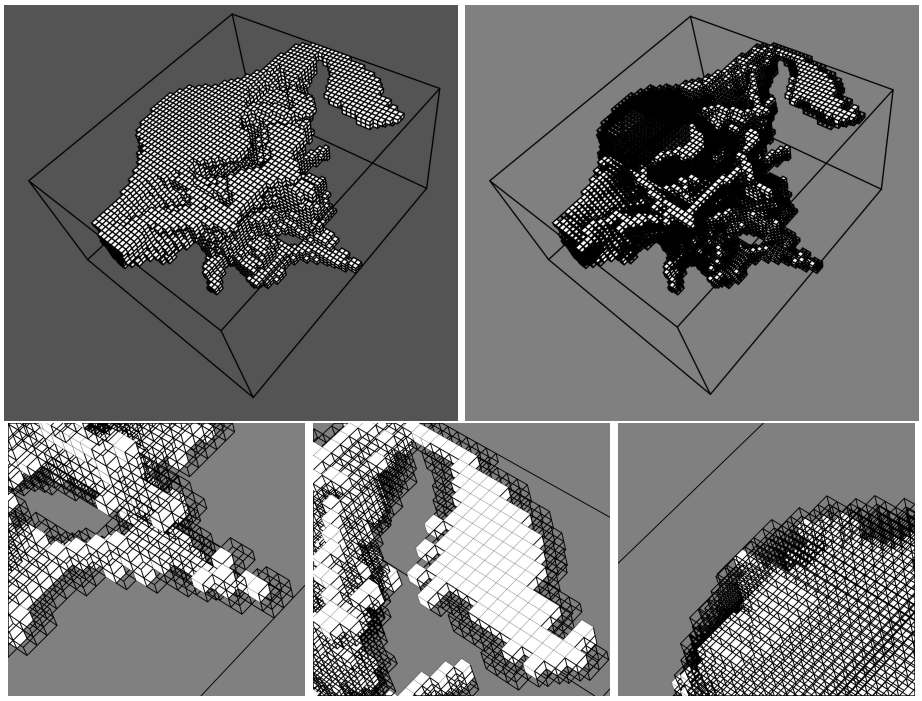

Fig. 5. Top-left: A voxel model of a human vertebra. Top-right: its skeleton, overlaid on the original voxel model (in wireframe). As a rule of thumb, we can think of darker areas as corresponding to places where the erosion is deeper. Bottom: zoom-in on three selected areas.

We have presented an skeletonisation algorithm fully described using only differences, unions and intersections of possibly shifted solids. It has been shown how the efficiency of the algorithm can be greatly improved and its memory requirements dramatically reduced by using extreme vertex encoded solids. The enormous algorithmic difficulties caused when working at a voxel level due to the identification of all removable voxels using large look-up tables—as standard thinning algorithms usually do—are thus avoided. Thanks to this fact, it can be applied—contrarily to all other alternative algorithms—to other solid representations, as long as they allow the easy computation of regularized boolean operations.

Iteration	t1	t2	Brinks
1	274	108	6028
2	329	132	5814
3	253	98	4174
4	101	41	1798
5	34	13	628
6	12	5	216
7	4	2	78
Total	1007	399	

Table I

References

1. J. Rodríguez, D. Ayala and A. Aguilera, "A Complete Solid Model for Surface Rendering," in *Geometric Modeling for Scientific Visualization*, Springer-Verlag, to appear, 2003.
2. A. Aguilera and D. Ayala, "Orthogonal Polyhedra as Geometric Bounds in Constructive Solid Geometry", *ACM SM'97*, pp. 56-67, 1997.
3. A. Aguilera and D. Ayala, "Domain extension for the extreme vertices model (EVM) and set-membership classification," *CSG'98. Ammerdown (UK)*, Information Geometers Ltd., pp. 33-47, 1998.
4. A. Aguilera and D. Ayala, "Converting orthogonal polyhedra from extreme vertices model to B-Rep and to alternative sum of volumes," *Computing Suppl. Springer-Verlag*, No. 14, pp. 1-28, 2001.
5. G.B. Borgefors, I. Nystrom and G. Sanniti Di Baja, "Computing skeletons in three dimensions", *Pattern Recognition*, Vol. 32, pp. 1225-1236, 1999.
6. J.W. Brandt, "Describing a solid with the three-dimensional skeleton," *SPIE Curves and Surfaces in Computer Vision and Graphics III*, No. 1830, pp. 258-269, 1992.
7. R. Cardoner and F. Thomas, "Residuals + directional gaps = skeletons," *Pattern Recognition Letters*, Vol. 18, pp. 343-353, 1997.
8. P.P. Jonker, "Morphological operations on 3D and 4D images: from shape primitive detection to skeletonization," *Proc. 9th Conf. on Discrete Geometry for Computer Imaginery*, (DGCI 2000), LNCS 1953, pp. 371-391, Springer Verlag, 2000.
9. L. Latecki, "3-D well-composed pictures," *Graphical Models and Image Processing*, Vol. 59, No. 3, pp. 164-172, 1997.
10. Ch. Lohou and G. Bertrand, "A new 3D 6-subiteration thinning algorithm based on p-simple points," *Proc. 9th Conf. on Discrete Geometry for Computer Imaginery*, (DGCI 2000), LNCS 1953, pp. 102-113, Springer Verlag, 2000.
11. G. Malandain and S. Fernández-Vidal, "Euclidean skeletons," *Image and Vision Computing*, Vol. 16, pp. 317-327, 1998.
12. A. Manzanera, T.M. Bernard, F. Prêteux, and B. Longuet, "Medial faces from a concise 3D thinning algorithm," *Proc. 7th IEEE Int. Conf. on Computer Vision*, pp. 337-343, 1999.
13. A. Manzanera, T.M. Bernard, F. Prêteux, and B. Longuet, "Ultra-fast skeleton based on an isotropic fully parallel algorithm," *Proc. 8th Conf. on Discrete Geometry for Computer Imaginery* (DGCI 1999), Lecture Notes in Computer Science 1568, pp. 313-324, Springer Verlag, 1999.
14. E. Remy and E. Thiel, "Medial axis for chamfer distances: computing look-up tables and neighbourhoods in 2D and 3D," *Pattern Recognition Letters*, Vol. 23, No. 6, pp. 649-661, 2002.
15. A. Sudhalkar, L. Gürxöz, and F. Prinz, "Box-skeletons of discrete solids," *Computer-Aided Design*, Vol. 28, No. 6-7, pp. 507-517, 1996.
16. J. Udupa and O. Odhner, "Fast visualization, manipulation and analysis of binary volumetric objects," *IEEE Computer Graphics and Applications*, Vol. 11, No. 6, pp 53-62, 1991.
17. Y. Zhou and A.W. Toga, "Efficient skeletonization of volumetric objects," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 5, No. 3, pp. 196-209, 1999.