, , 1–21 ()

A Parallel Algorithm to Reconstruct Bounding Surfaces in 3D Images

FREDERIC H. MABIN frederic.mabin@forenap.asso.fr
Foundation for Applied Neuroscience Research in Psychiatry, Centre Hospitalier, Service du Dr. J.P. Macher, 27 rue du 4e RSM, F-68250 Rouffach

CATHERINE MONGENET mongenet@icps.u-strasbg.fr
Université Louis Pasteur de Strasbourg, ICPS laboratory, Pôle API, Boulevard Sébastien Brant, F-67400 Illkirch

Editor:

Abstract. The growing size of $3D$ digital images causes sequential algorithms to be less and less usable on whole images and a parallelization of these algorithm is often required. We have developed an algorithm named Sewing Faces which synthesizes both geometrical and topological information on bounding surface of 6-connected $3D$ objects. We call such combined information a skin. In this paper we present a parallelization of Sewing Faces. It is based on a splitting of $3D$ images into several sub-blocks. When all the sub-blocks are processed a gluing step consists of merging all the sub-skins to get the final skin. Moreover we propose a fine-grain approach where each sub-block is processed by several parallel processors.

Keywords: parallel applications, computer graphics, $3D$ digital images, bounding surfaces reconstruction, coarse and fine-grain parallelization.

1. Introduction

Over the past decade, $3D$ digitalization techniques such as Magnetic Resonance Imaging have been extensively developed. They have opened new research topics in $3D$ digital image processing and are of primary importance in many application domains such as medical imaging. The classical notions of $2D$ image processing have been extended to $3D$ (pixels into voxels, 4-connectivity into 6-connectivity, etc) and the $2D$ algorithms have to be adapted to $3D$ problems ([4], [2]). In this process the amount of data is increased by an order of magnitude (from n^2 pixels in a $2D$ image to n^3 voxels in a $3D$ image, where n is the size of the image edges) and in consequence the time complexity of $3D$ algorithms is also increased by an order of magnitude. In order to still get efficient algorithms in terms of running time and to deal with images with ever increasing size, these algorithms have to be parallelized.

Among the many problems in $3D$ image processing, we focus in this paper on the problem of the reconstruction of bounding surface of 6-connected objects in $3D$ digital images.

A $3D$ digital image is characterized by a $3D$ integer matrix called block; each integer $I(v)$ of the block defines a value associated with a volume element or *voxel* v of the image. An image describes a set of objects such as organs in medical images. The *contour* of an object is composed of all the voxels which belong to the object but which have at least one of their adjacent voxels in the background. From this set of voxels we compute the *bounding surface* of the object. It is a set of closed surfaces enclosing the object.

We have developed in [6] a sequential algorithm for bounding surface reconstruction. The objective of this paper is to present a parallel version of this algorithm. This parallelization is based on a decomposition of the $3D$ block into sub-blocks. On each sub-block a fragment of the bounding surface is computed. Once all the fragments have been determined a final step consists in merging them together in order to retrieve the complete bounding surface.

The paper is organized as follows. Section 2 recalls the principles of bounding surface reconstruction. Section 3 presents some basic notions of $3D$ digital images while section 4 briefly recalls our sequential algorithm for bounding surface reconstruction. Section 5 discusses the sub-block decomposition, then sections 6 and 7 respectively present the coarse-grain and fine-grain parallelizations of the algorithm. In section 8 we briefly show how to transform a reconstructed surface into a $2D$ mesh. Finally section 9 presents experimental results. Some of the notions introduced in this paper are illustrated with figures. Since they are not always easy to visualize in $3D$ they will be presented using the $2D$ analogy.

2. Surface reconstruction

The closed surfaces that bound an object can be determined in two different ways :

- using a method by approximation, the surface is reconstructed by interpolating the discretized data. The Marching Cubes [5] developed by Lorensen and Cline is such a method : it builds a triangulation of the surface. Various extensions of the method have been proposed, either by defining a heuristic to solve ambiguous cases [9] or by reducing the number of generated triangles. Faster reconstructions have been developed. Some are based on parallelized versions of the algorithm [7]. Others use the octree abstract data type [10] which reduces the number of scanned voxels.

- using an exact method, the surface is composed of faces shared by a voxel of the object and a voxel of the background. Such a method has been proposed by Artzy et al. [1] and later improved by Gordon and Udupa [3].

The efficiency of the various reconstruction algorithms is strongly related to the type of scan used to determine the surface. Hence the surface reconstruction can be realized either by a complete search among all the voxels of the block or by a *contour following* for which only the voxels of the object contour are scanned. The contour following approach yields more efficient algorithms whose time complexity is proportional to the number of voxels of the contour instead of the number of voxels of the whole block. The Marching Cubes algorithm is based on a whole-block scanning while the methods proposed in [1] and [3] rely on a contour following.

The determination of the bounding surface of an object is useful to visualize the object but also to manipulate it, using techniques such as a distortion of a surface, a transformation of a surface into a surface mesh, a derefinement of a surface by merging adjacent coplanar faces, a reversible polyhedrization of discretized volumes. For visualization, the surface needs only to be defined by geometrical information, i.e. the list of its triangles in case of approximation methods or the list of its faces in case of an exact method. For manipulation however, the surface must be defined not only by geometrical information but also by topological information, i.e. information stating how the faces are connected together. Note that it is of course possible to recover the topological information from the geometrical one. For each face, one have to scan all the other faces defining the surface in order to find its adjacent faces, i.e. the ones which share one edge with it. If the surface contains n faces then this topological reconstruction is $O(n^2)$. To avoid this quadratic operation the topological information must be collected together with the geometrical information.

The algorithm we have developed in [6] reconstructs the bounding surface of any 6-connected object of a $3D$ digital image. It is called *Sewing Faces* and its characteristics are the following :

- it is an *exact method*. It extracts faces belonging to a voxel of the object and a voxel of the background.
- it is based on a *contour following*. Its time complexity is therefore proportional to the number of voxels of the contour.
- it synthesizes both *geometrical and topological information*. In this case the reconstructed surface is named a *skin*. The topological information is synthesized using *sews* stating how two adjacent faces of the bounding surface are connected together.
- its time and space complexity are both *linear* according to the number of faces of the skin, as proved in [6].

3. Notions of $3D$ digital images

A $3D$ *block* (see figure 1) can be seen as a stack of adjacent voxel slices pushed together according to any one of the three axes x, y or z. A voxel is made of six *faces* (whose *types* can be numbered as shown in figure 2), and twelve *edges*. Each face has an *opposite* face in a voxel; for instance face of type 2 is opposite to face of type 5 (cf. figure 2). In the following we call face i a face of type i.

Two faces that share one edge are adjacent. Two voxels that share one face are 6-*adjacent*; if they share only one edge they are 18-*adjacent* (see figure 3). In the following we call object in a block a set of 6-connected voxels.

Definition 1. Let u and v be any two voxels of set θ. If there exists a path $x_0, x_1, \ldots, x_n$ with $u = x_0$ and $v = x_n$ such that $\forall k \in [0, n[$, $x_k, x_{k+1} \in \theta$ and are 6-adjacent, then θ is 6-*connected*.

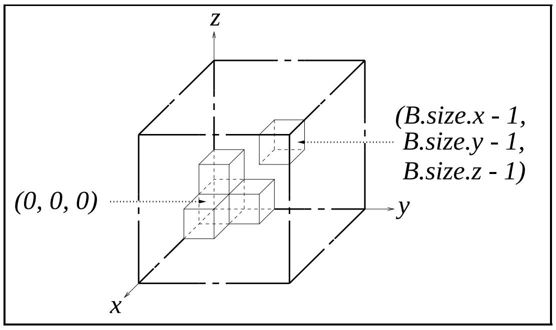

Figure 1. A $3D$ block.

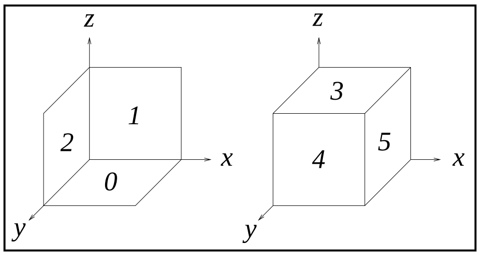

Figure 2. Type of voxel faces.

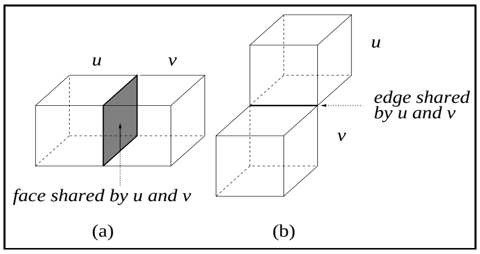

Figure 3. (a) : u and v are 6-adjacent, (b) : u and v are 18-adjacent.

If block B contains more than one object, that is to say if B is made of several 6-connected components θ_i, we call this set of objects a *composed object* and we denote it by Θ : $\Theta = \bigcup_i \theta_i$. The voxels which are not in composed object Θ are in the background.

A boolean function is defined on block B. It is denoted by $\Theta_B(v)$ and states whether or not voxel v belongs to composed object Θ. There are several ways to define function $\Theta_B(v)$, depending on the type of digitalized data. If the block is already thresholded, we may define $\Theta_B(v) = true \Leftrightarrow I(v) = \alpha$ where α is user-defined. If the block is not segmented, we may use a function $\Theta_B(v) = true \Leftrightarrow \alpha_1 < I(v) < \alpha_2$. On the contrary, if we want to define the object as the complement of the background, we may define $\Theta_B(v) = true \Leftrightarrow I(v) \neq \alpha$. Other more sophisticated definitions are possible.

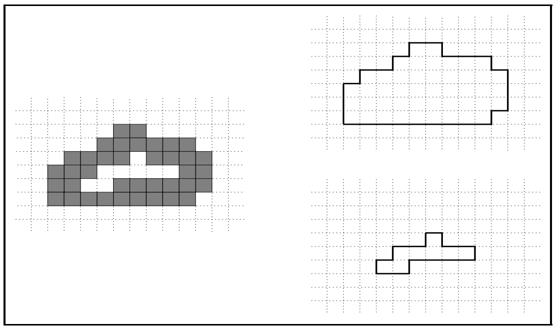

Figure 4. A $2D$ object with one hole and its two borders.

If an object contains n holes, its bounding surface is made of $(1+n)$ *borders* that are not connected together. Each border is a closed surface made of adjacent faces sewed together. Figure 4 illustrates this notion with a 1-hole object using the $2D$ analogy.

Since objects are 6-connected sets of voxels, there exist three different types of *sews* between two adjacent faces of a border. These three types of relations have been presented by Rosenfeld et al. [8] and are named *1-sew*, *2-sew* and *3-sew* (see figure 5). They depend on the adjacency relation between the voxel(s) supporting the two faces.

Definition 2. Let θ_i be a 6-connected object with n holes. Each *border* $\Upsilon_{\theta_i,j}$ $(j = 1, \ldots, 1+n)$ of θ_i is a pair (F, R) where :

- F is the set of faces which separate 6-connected component θ_i from the background by a closed surface;

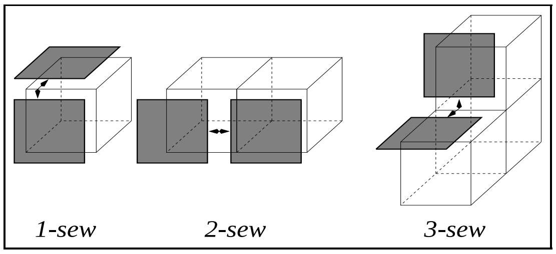

Figure 5. Three types of sews.

- R is the sewing relation. It is a set of 4-tuples (f_1, f_2, e, s) expressing that faces f_1 and f_2 are sewed together through their common edge e using a sew of type s ($s = 1, 2$ or 3).

Using this definition the notions of skin are introduced as follows.

Definition 3. The *skin* of object θ_i characterized by n holes is the union of all its borders and is denoted by S_{θ_i} : $S_{\theta_i} = \bigcup_{j=1}^{1+n} \Upsilon_{\theta_i, j}$.
The skin of composed object $\Theta = \bigcup_{i=1}^{m} \theta_i$ is the union of the skins S_{θ_i} and is denoted by S_Θ : $S_\Theta = \bigcup_{i=1}^{m} S_{\theta_i}$.

Since the skin of a composed object is a union of borders, it is also defined as a pair (F, R) as introduced by definition 2.

4. Sewing Faces : an algorithm for skin reconstruction

For each border Υ to be reconstructed, the starting point of Sewing Faces is a pair (v, i) where v is a voxel of the object contour and i is a face of v belonging to border Υ. Such a pair is called a *starting-voxel* and can be either given by the user or determined by a dichotomous search algorithm that depends on the type of the $3D$ image.

The principles of the sequential version of Sewing Faces are the following. From the starting voxel, the algorithm first computes its faces that belong to the bounding surface and then detects among its adjacent voxels (either 6 or 18-adjacent) the ones that belong to the contour. For each of these voxels it determines the faces that are also included in the bounding surface and realizes their sews with adjacent faces of the skin. The process is then iterated for all the adjacent voxels that also belong to the contour. Step-by-step the bounding surface is reconstructed based on a contour following.

The algorithm uses a *hash table* to memorize the faces that have already been added to the skin, and a *stack* to store the voxels to be examined. It uses two main functions $GetFaces\,(v,\,i)$ and $Treat_{Seq.}\,(v)$ that are now described. They rely on the notion of neighbor of a voxel according to a given face defined by its type (cf. figure 2).

Definition 4. Let v be a voxel whose face of type i is shared with voxel u. Voxel u is called the *neighbor* of v by its face i and is denoted by $n(v,i) = u$.

Function $GetFaces\,(v,\,i)$ determines the faces of v belonging to the skin when its face of type i does. It runs as follows :

1. Add [1] to F face i of v.
2. Add to F any of the four faces of type j of v adjacent to face i such that $n(v,j) \notin \Theta$ or $n(v,j) \notin B$, i.e. such that the neighbor of v by face j is not in the object.
3. If one face has been added to the border during steps (1) or (2) then :
   (A) Add to F face of type k of v which is opposite to i if $n(v,k) \notin \Theta$ or $n(v,k) \notin B$. [2]
   (B) Add to R the 1-sews $(f_1, f_2, e, 1)$ for all faces f_1 and/or f_2 that have been added to F during steps (1), (2) or (3)(A).
   (C) Push v onto the stack of voxels to be treated. Hence voxel v is not entirely treated at this step. In particular its 2 and 3-sews with adjacent voxels have not been detected yet.
   (D) Add to the hash table the faces of v added to the skin.

Function $Treat_{Seq.}\,(v)$ determines all the 6 or 18-adjacent voxels of v such that one or more of their faces belong to the skin, and sews these faces to those of v when they share one edge, i.e. realizes the 2 and 3−sews. The data structure used by $Treat_{Seq.}\,(v)$ is an array $[0 \cdots 5]$ memorized in the hash table. It indicates for each face type, the reference in F of the corresponding face of voxel v if this face has already been added to the skin. The corresponding entry is empty if the face is not in F or if the face has not already been added to the skin. $Treat_{Seq.}\,(v)$ is defined as follows :

$Treat_{Seq.}\,(v)$
    get $faces_v[\,]$ from the hash table
    for each edge e of v shared by faces i and j of v
        such that $faces_v[i] \notin S_\Theta,\, faces_v[j] \in S_\Theta$ **do**
        $u \leftarrow n(v,i)$
        $w \leftarrow n(u,j)$

    if $(\Theta_B\,(w) = false)$
        /* w is not in object Θ_B */
        /* in this case face j of voxel u belongs to the skin */
        $faces_u\,[\,] \leftarrow GetFaces(u, j)$
        $R \leftarrow (faces_v\,[j], faces_u\,[j], e, 2)$
    else
        /* w is in object Θ_B */
        /* in this case the opposite face to face i in voxel w */
        /* (denoted by k) belongs to the skin */
        $k \leftarrow (i + 3)$ **mod** 6
        $faces_w\,[\,] \leftarrow GetFaces(w, k)$
        $R \leftarrow (faces_v\,[j], faces_w\,[k], e, 3)$
    endif
endfor

Using these two functions the sequential version of Sewing Faces, denoted by $SF_{Seq.}\,(\,)$ is defined by:

$SF_{Seq.}\,(\,)$
    for each starting voxel (v, i) **do**
        $GetFaces\,(v,\ i)$
    while *not empty* $(stack)$ **do**
        $Treat_{Seq.}\,(top\,(stack))$

5. Sub-blocks decomposition

Let us suppose that the blocks we deal with contain one byte long integers. The memory size required for blocks of size 128^3, 512^3 or 1024^3 is respectively $2\ MB$, $128MB$ or $1GB$. To allow any computer to run Sewing Faces on such large-size data, the whole block must be decomposed into sub-blocks and the algorithm must be processed on these sub-blocks. The size of the sub-blocks depends on the available amount of memory. In the general case, each voxel intensity being g bytes long, a c bytes memory space can hold sub-blocks up to size (l, l, l) with $l = (g/c)^{1/3}$.

Using a parallel machine, such a decomposition into sub-blocks is also very useful since the bounding surface reconstruction algorithm may be run simultaneously on the different sub-blocks assigned to different processors of the architecture.

5.1. Sub-blocking

Splitting the block into sub-blocks is the *sub-blocking* operation. A sub-block is fully defined by the coordinates of its *origin* in the block and by its size according to axes x, y and z. It has six *faces*, each face is a voxel slice.

On each sub-block, Sewing Faces builds a sub-skin. All the sub-skins must finally be merged to get the whole skin. In order to be able to glue all the sub-skins together, the overlap between two adjacent sub-blocks must be two-slice wide.

Definition 5. Let $b_{i=1,2}$ be two sub-blocks whose origins in block B are $(B.x_1, B.y_1, B.z_1)$ and $(B.x_2, B.y_2, B.z_2)$ and whose sizes are $(b_1.x, b_1.y, b_1.z)$ and $(b_2.x, b_2.y, b_2.z)$. If there exist two axes $\alpha, \beta \in \{x, y, z\}$ such that $B.\alpha_1 = B.\alpha_2$ and $B.\beta_1 = B.\beta_2$ and if the third axis γ is such that $B.\gamma_1 + b_1.\gamma - 2 = B.\gamma_2$ or such that $B.\gamma_2 + b_2.\gamma - 2 = B.\gamma_1$, then sub-blocks b_1 and b_2 are said to be *adjacent*.

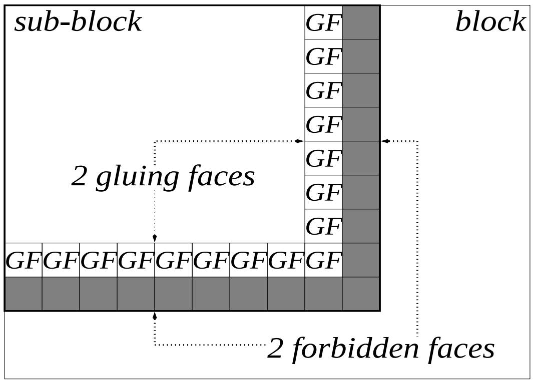

Figure 6. The gluing and forbidden faces of a sub-block.

On any sub-block two particular types of faces are emphasized: the gluing faces and the forbidden faces, as illustrated in figure 6 using the $2D$ analogy. They correspond to the overlap between sub-blocks and are therefore defined only "inside" the whole block and not on its own faces.

Definition 6. Let b be a sub-block and f_b be one of its faces such that f_b is not included into a face of block B. Face f_b is called a *forbidden face*. A slice which is adjacent to a forbidden face is called a *gluing face*.

The execution of Sewing Faces on sub-block b reconstructs the sub-skin associated with b. This sub-skin is similar to a skin except on the "border" of sub-block b where some sews are missing. The missing sews connect a face of a voxel lying on one of its gluing faces with a face of a voxel belonging to a sub-block adjacent to b. Notice that since the sews of type 1 involve only one voxel, they can always be detected even if they are on a gluing face. The missing sews are therefore only of type 2 or 3. They must be detected and memorized to be treated during the final gluing phase. To do so the complete neighboring of the gluing face voxels must be examined. This explains the presence of the forbidden faces adjacent to the gluing faces.

Since the voxels of the forbidden faces are not treated in sub-block b, any forbidden face must be shared by an adjacent sub-block where it is considered as a gluing face. These adjacency relations between sub-blocks guarantee that the gluing process will be possible. They are characterized as follows.

Definition 7. A set $\{b_0, \ldots, b_n\}$ of sub-blocks such that :

$\forall v \in \Theta, \exists\, b_i, i \in [0, n]$ such that $v \in b_i$ but v does not belong to a forbidden face of b_i, and

$\forall v$ such that $v \in b_i$ and $v \in b_j$, then b_i and b_j are adjacent

is a *valid sub-blocking* (with respect to composed object Θ).

Figure 7 shows a valid sub-blocking using the $2D$ analogy. Notice that a valid set of sub-blocks needs not to cover the whole block. If part of the image contains only background voxels, it is unnecessary to process it. In the general case it is easy to automatically split a block into a valid sub-blocking whose sub-blocks can be separately processed.

5.2. *Gluing phase*

Once the sub-skins on the different sub-blocks have been reconstructed, they must be merged together to get the final skin. This process requires the realization of the missing sews. We call the whole process (merging and sewing) the *gluing phase*.

For any sub-block b_i the sub-skin is characterized by pair (F_{b_i}, R_{b_i}) where F_{b_i} is the set of faces belonging to the sub-skin of b_i and R_{b_i} is the corresponding set of fully realized sews. During the sub-skin computation the missing sews called *half-sews* are memorized. They are defined as follows.

Definition 8. A *half-sew* is a quintuple $(e, s, f, coord, t)$ where e is the edge to be sewed, s is the type of sew to realize, f is the face of set F sharing edge e, $coord$ are the coordinates of voxel v that owns f, t is the type of the unknown face that share edge e with face f.

For each sub-block this information is memorized using either a hash table or a list ordered according to field *coord*.

The gluing process is realized in two steps. The first one consists in merging together the different sets F_{b_i} computed on sub-blocks b_i : $F_B = \bigcup_i F_{b_i}$. The second step concerns set R_B defining the sews between all the faces of F_B. This step is realized by building *full-sews* from pairs of half-sews describing the two parts of a sew.

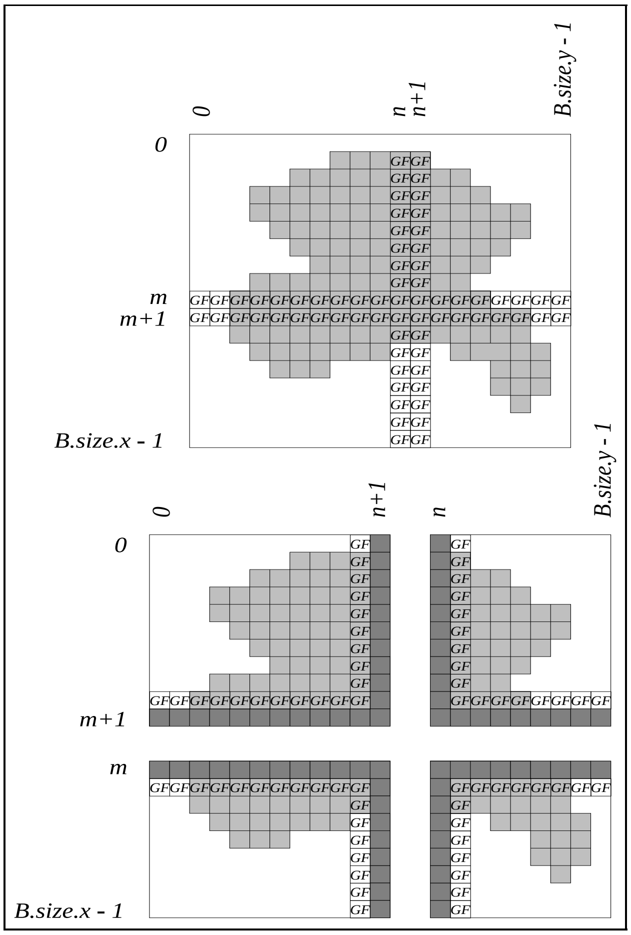

Figure 7. A valid sub-blocking.

Definition 9. A *full-sew* is composed of two half-sews $(e_1, s_1, f_1, coord_1, t_1)$ and $(e_2, s_2, f_2, coord_2, t_2)$ characterized as follows : $coord_1$ and $coord_2$ define the coordinates of the two adjacent voxels involved in the sew, $e_1 = e_2$, $s_1 = s_2$, $t_1 = type(f_2)$ and $t_2 = type(f_1)$.

At the end of the gluing process, the skin (F_B, R_B) describes the geometry and the topology of all the borders that were pointed out by the starting voxels given at the beginning.

5.3. Fragmentation of the objects

The following problem occurs when decomposing a block into sub-blocks. A 6-connected object contained in the block is fragmented on the different sub-blocks and on a given sub-block the fragment of the object *may be non* 6*-connected.* If this problem is not carefully taken into account the skin reconstruction will be incorrect.

Figure 8 illustrates such a situation using the $2D$ analogy. The initial block is splitted into two sub-blocks with an overlap of two slices : the gluing and forbidden faces. The 6-connected object is distributed on the two sub-blocks in such a way that on sub-block 2 there are two non-connected fragments of the object. Therefore if the algorithm on sub-block 2 runs with only one starting voxel, then only one of the two fragments will be reconstructed. Since it is not realistic to expect the user to give as many starting voxels as there are fragments of the objects on the different sub-blocks, this problem must be automatically solved by the method. This is realized on the gluing faces. When a voxel of the contour lies on a gluing face, its adjacent voxels belonging to the forbidden face may also belong to the contour and are therefore considered as new starting voxels for the adjacent sub-block. In figure 8 let the two voxels drawn in dark grey be the starting voxels on each sub-block. During its processing, sub-block 1 detects that the sub-skin it is reconstructing gets out in sub-block 2. A new starting voxel (drawn in black) is then pushed by sub-block 1 onto the stack of sub-block 2 and no part of the skin is lost. Sub-block 1 will also send to sub-block 2 as other starting voxel the voxel mentioned on the figure. Sub-block 2 will discard this voxel since it has already been treated while constructing the small fragment on top of the figure from it initial starting voxel. Vice versa, sub-block 2 will send two new starting voxels to sub-block 1. They will also be discarded by sub-block 1 since they have already been treated. This mecanism guarantees that the whole skin will finally be reconstructed. As a consequence only one starting-voxel (v, i) (as described in section 4) per sub-block is required at the beginning of the algorithm.

6. Coarse-grain parallelization

The coarse-grain parallelization of Sewing Faces is based on the notions of sub-blocking and gluing phase. The idea is to decompose the $3D$ block of data into sub-blocks as defined in section 5. The sub-skins on each sub-block are computed simultaneously on different processors. Once all the sub-skins have been determined the final gluing phase is realized.

The computation of a sub-skin is similar to that used in the sequential version, except for the voxels of the gluing faces that may induce missing sews. For any such voxel the following steps must be realized :

1. Store the corresponding half-sew (as defined in definition 8) in order to later compute the full-sew;
2. Determine which voxel v' of the forbidden face should be treated to realize this sew;

3. Determine which sub-block b_j owns v' on one of its gluing faces;
4. Push voxel v' onto the stack associated with sub-block b_j as a new starting voxel. This allows to solve the fragmentation problem discussed in section 5.3.

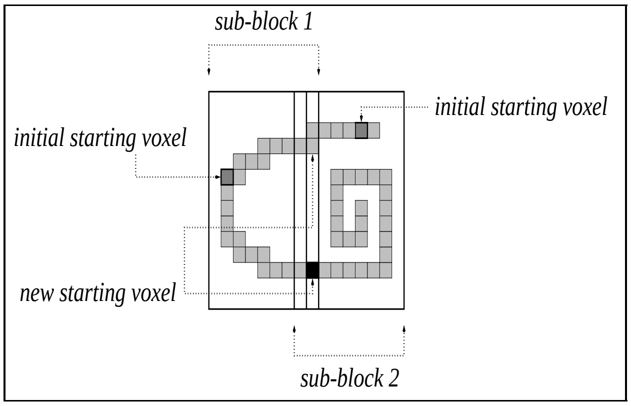

Figure 8. Fragmentation of a 6−connected object into non 6−connected fragments.

These four points are realized by function $HalfSews\,(\,)$. Notice that in order to realize point (3), each sub-block has to know the origin and the size of all the other sub-blocks. Point (4) can be realized either with a message passing mecanism on a distributed memory processor or using shared memory on a share memory machine.

The parallel version of the algorithm, denoted by $SF_{Par.}()$, is obtained by changing function $Treat_{Seq.}(v)$ into $Treat_{Par.}(v)$ as follows.

$Treat_{Par.}\,(v)$
    get $faces_v\,[\,]$ from the hash table
    for each edge e of v shared by faces i and j of v
        such that $faces_v\,[i] \notin S_\Theta, faces_v\,[j] \in S_\Theta$ **do**
        if e defines a missing sew **then**
            $HalfSews\,(\,)$
        else
            $u \leftarrow n(v, i)$
            $w \leftarrow n(u, j)$
            if $(\Theta_B\,(w) = false)$
                /* w is not in object Θ_B */
                /* in this case face j of voxel u belongs to the skin */
                $faces_u\,[\,] \leftarrow GetFaces(u, j)$

```
            R ← (faces_v[j], faces_u[j], e, 2)
        else
            /* w is in object Θ_B */
            /* in this case the opposite face to face */
            /* i in voxel w belongs to the skin */
            k ← (i + 3) mod 6
            faces_w[] ← GetFaces(w, k)
            R ← (faces_v[j], faces_w[k], e, 3)
        endif
    endif
endfor
```

When all the sub-blocks have ended execution, each sub-skin is characterized by a pair (F, R) and the gluing process can begin. As explained in section 5.2 the half-sews associated with each sub-block are memorized into ordered lists. The gluing phase consists therefore in scanning these lists in order to complete the sews. Due to the ordering of the lists according to the coordinates of the supporting voxels, this process is linear relatively to their length.

The number of voxels to be treated in each sub-block obviously depends on the objects that are considered and also on the sub-blocking. Sub-blocking may be data-driven : the whole image is scanned in order to detect the objects before splitting the block. This method is used by [7] to achieve load-balancing in Marching Cubes parallelization. Such a whole block scanning is terribly cost-effective compared to the contour following used by Sewing Faces to get good time performance. Moreover we believe that a good load balancing is strongly related to a given field : for instance brain images obtained from Magnetic Resonance Imaging are all rather similar relatively to the question of load-balancing. Therefore we think that before using Sewing Faces in a given field, a preliminary study must be conducted on a set of standard images in order to detect the appropriate sub-block decomposition that will result in effficent load-balancing. Such a decomposition may be for example to split the block either into cubic sub-blocks or into slices.

7. Fine-grain parallelization

The fine-grain parallelization of Sewing Faces consists in allowing several processors to deal with the same sub-block. In this case input data and data structures must obviously be shared by all the processors running Sewing Faces on the same sub-block. In consequence the main difficulty of such a parallelization is to prevent data from corruption. Let us see in detail where corruption problems may arise :

- sub-block : no possible data-corruption because it is read-only;
- hash table : when array $faces_v[\,]$ is written in the hash table, its previous version currently stored may contain values that are not in $faces_v[\,]$. In this case the

two versions of the array, denoted by $faces_v^{old}[\,]$ and $faces_v^{new}[\,]$ must be merged. Moreover 1-sews between faces i and j such that ($faces_v^{new}[i]$ and $faces_v^{old}[j]$ both belong or not to F) or ($faces_v^{new}[j]$ and $faces_v^{old}[i]$ both belong or not to F) must be added to R. During this merging step, access in the hash table to array $faces_v[\,]$ by other processors must be forbidden using a semaphore-like method;

- stack of voxels : no possible data-corruption since the voxels are just pushed onto or poped from the common stack. The stack implementation must of course guarantee its correctness using semaphore-like operations.
- set F of faces : it is simply implemented using an array. When a new skin face is detected during one $GetFaces(\,)$ execution, the next available face number in F must be read and incremented using non interruptible instructions;
- set R of sews : no data curruption is possible because R is a write-only file;
- set of half-sews : it is implemented as a write-only disk file, therefore no data corruption is possible. Moreover, the gluing process can easily deal with duplicated half-sews by omitting them when the case arises.

The fine-grain parallelization of Sewing Faces solves the load balancing problem in the general case. All the processors executing the algorithm on the same sub-block share one stack of voxels, one hash table, etc. As a result they all finish their execution at the same time, that is when the stack of voxels is empty. When the fine-grain approach is used alone without any coarse-grain parallelization, the load-balancing is always optimal.

The fine-grain parallelization may be combined with a coarse-grain decomposition using a cluster of share-memory processors. The different sub-blocks are assigned to the different machines of the cluster. On each share memory machine the different processors may compute the sub-skin associated with one sub-block through the shared stack of voxels.

8. Embedding

So far we have only realized topological operations (adding faces to set F, adding sews to set R), without considering real coordinates of the vertices. Therefore the extracted surface is a *topological surface*. In order to visualize it, it must be transformed it into a *geometrical surface*, i.e. into a $2D$ mesh. Such a process is called an *embedding*. A face embedded in the $3D$ space becomes a *facet*. To convert all the faces into facets, a starting point is required : the real coordinates of the four vertices of a given face f of each border. From the type of face f and from its sew types, it is easy to deduce the coordinates of the four faces sewed with f. And so on. We thus obtain the real coordinates of all the faces of F_B and we get all the facets. If the three dimensions of the basic parallelepiped representing one voxel are integer values, embedding of the skin does not require any computation with real values. The embedding process is $O(f)$ where f is the number of faces of the skin.

9. Results

We have already proved in [6] that the sequential version of Sewing Faces is linear in time and space according to the number of faces in the skin. Linearity is still achieved by the coarse-grain and fine-grain approaches. In tables 1, 2 and 3 we focus on the execution time of the coarse-grain version versus the sequential version. The input data consist of digital balls of growing size (from 230^3 to 300^3). Each image contains only one object whose skin is made of only one border. The starting voxels are automatically detected by a dichotomous algorithm. Sub-blocking is achieved by splitting each block into eight equal size sub-blocks. Tests summarized in tables 1, 2, and 3 were realized on a Intel Pentium 133 MHz computer running under Linux. Figures include user and system times.

Table 1. Execution time related to the sequential version $SF_{Seq.}()$.

Size of the $3D$ block	Number of faces	Time (s.)
230	247512	4.7
240	269688	5.1
250	292800	5.6
260	316656	6.0
270	341640	6.5
280	367368	7.0
290	394080	7.5
300	421968	8.0

Table 1 recalls some results obtained with the sequential version of Sewing Faces. The number of faces and the elapsed time (in seconds) are indicated. Indicated durations include the block loading, the starting voxel detection and the skin building.

Table 2. Execution time related to the coarse-grain parallel version of $SF_{Par.}()$.

Size of the $3D$ block	Time to compute on a sub-block	Time of the gluing phase	Total (s.)
230	0.7	0.2	0.9
240	0.7	0.3	1.0
250	0.8	0.3	1.1
260	0.8	0.4	1.2
270	0.9	0.4	1.3
280	1.0	0.4	1.4
290	1.0	0.5	1.5
300	1.1	0.5	1.6

Table 2 shows the elapsed time of the coarse-grain version of Sewing Faces on any elementary sub-block. The measured durations include the sub-block loading, the starting voxel detection, the sub-skin building and the half-sews detection. The third column indicates the elapsed time related to the gluing process. Finally the last column shows the theoretical time obtained on a multi-processor architecture where all sub-blocks are simultaneously processed. It is obtained by adding the gluing time and the elementary sub-block time.

Table 3. Speed-up obtained with the coarse-grain parallelization.

Size of the $3D$ block	Time saved (%)	Speed-up
230	80.85	5.2
240	80.39	5.1
250	80.35	5.1
260	80.00	5.0
270	80.00	5.0
280	80.00	5.0
290	80.00	5.0
300	80.00	5.0

Table 3 points out the time saved by the coarse-grain approach. The last column indicates the speed-up factor due to the coarse-grain approach and underlines the fact that using 8 processors we get an speed-up factor of about 5.

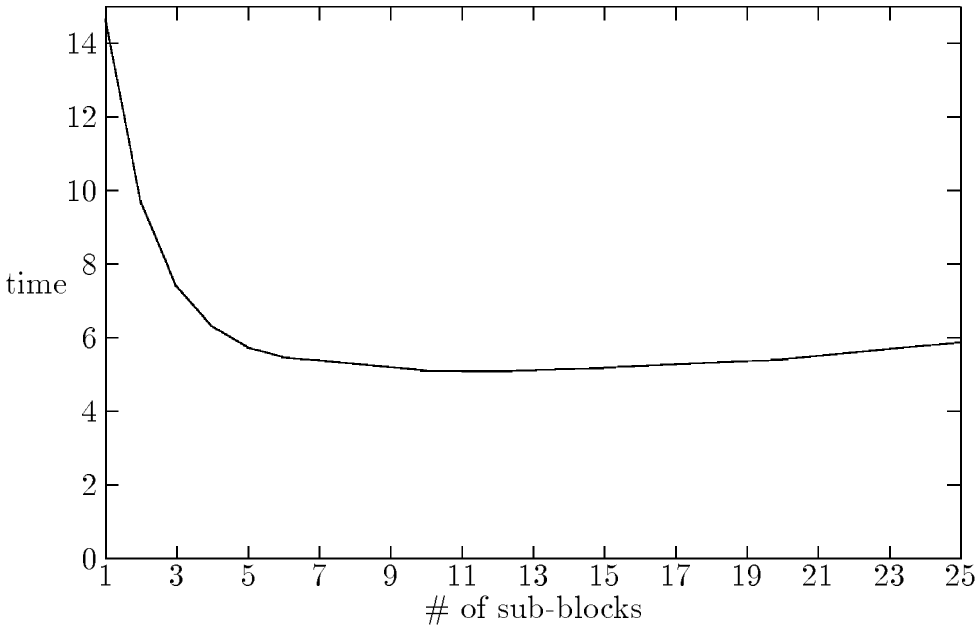

Figure 9. Parallel time computation for the bar.

Let us now study the influence of the sub-blocking on the parallel computation time. The efficiency obviously depends on the number and the structure of the sub-blocks. If the number of sub-blocks is too large, the gluing phase will become more time-consuming due to the increasing number of missing sews. This problem has been studied with a synthetic block of size $225 \times 300 \times 225$ representing a bar of size $210 \times 300 \times 210$. The block is decomposed into vertical sub-blocks of equal width. In order to analyze the influence of the number of sub-blocks on the efficiency of the parallel version, we have executed it on the above-mentioned block, increasing the number of sub-blocks. This experiment has been realized on a SGI R4400. Figure 9 shows the experimental results. It indicates that the computation time decreases with the number of sub-blocks until we reach 12 sub-blocks. With more than 12 sub-blocks the computation time remains almost the same. We may take advantage of the lack of penality in terms of computation time when increasing the number of sub-blocks, to execute Sewing Faces on more than 12 sub-blocks in case of memory limitation. The experiment has also been conducted on brain images (cf. figure 11) using cubic sub-blocks of equal size. The previous results are confirmed : above a certain number of sub-blocks, 27 in this case as indicated in figure 10, the computation time does not decrease anymore due to the increasing number of missing sews to glue. Figure 12 shows the result of the decomposition of a brain image into two sub-blocks.

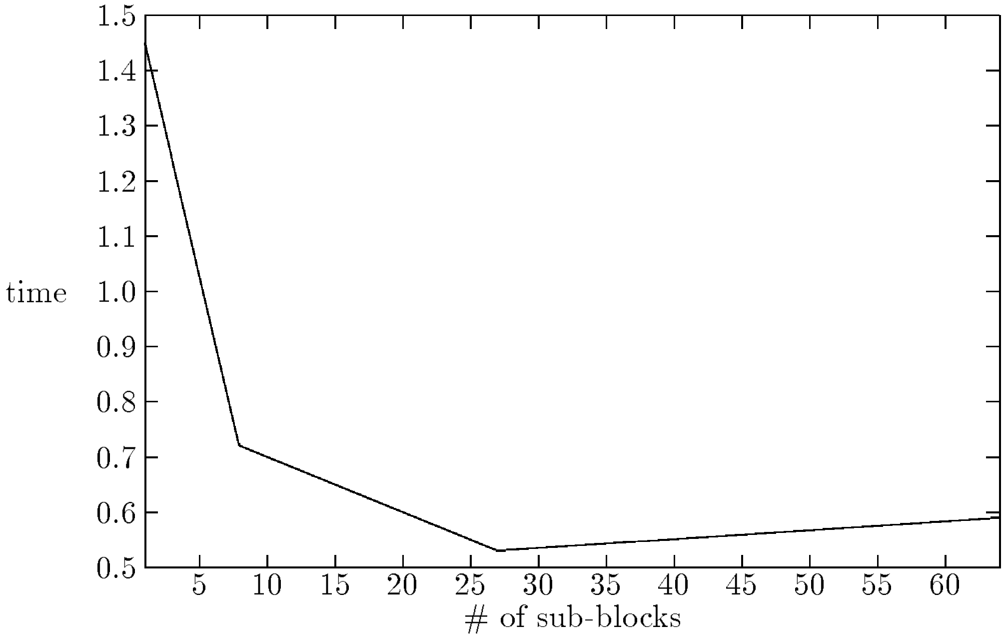

Figure 10. Parallel time computation for the brain.

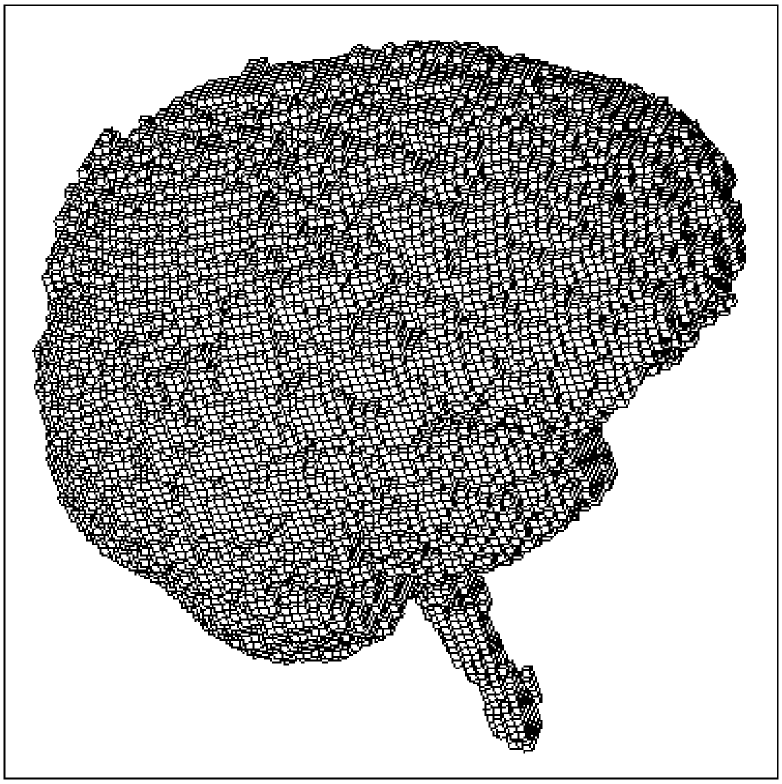

Figure 11. A human brain.

10. Conclusions

We have proposed in this paper a parallel version of a reconstruction surface algorithm. Our goal is not only to increase time performances but also to deal with large $3D$ images that are now currently available in practical fields such as medical imaging. This parallelization is based on a sub-block decomposition. In order to compute the skin using a contour following approach, a two-slice wide overlapping between adjacent sub-blocks is required. Moreover sub-blocks must communicate with their neighbors to guarantee the computation of the whole skin. The initial choice of the data structures used by the sequential version of Sewing Faces (one stack of voxels and one hash table for voxel faces) allows to easily and fully parallelize it, using a coarse-grain and/or a fine-grain approach. Moreover the notions of border, composed object and sub-block overlapping cause the parallel version of Sewing Faces to be very flexible. Notice furthermore that the sub-block based

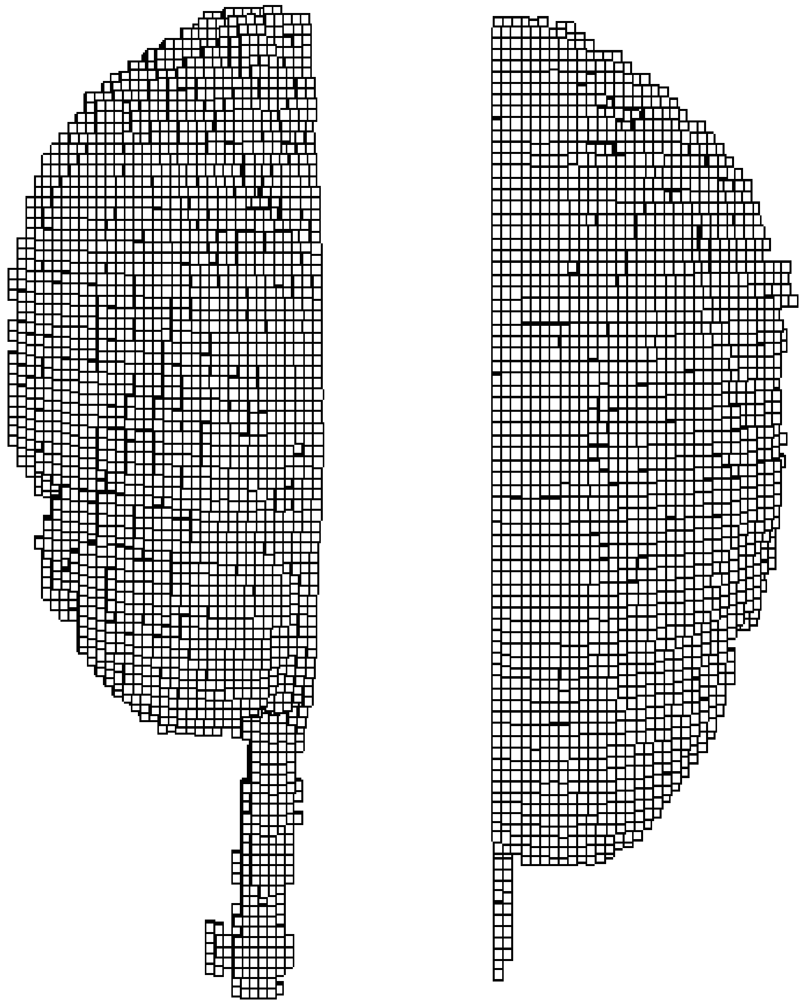

Figure 12. A human brain splitted in two sub-blocks.

algorithm may also be useful on a mono processor machine to deal with blocks of data that are too large to fit in memory. Experimental results have shown the efficiency of the parallel version of the algorithm, both on synthetic data and on brain images.

Acknowledgments

This research is supported by FORENAP, the Foundation for Applied Neuroscience Research in Psychiatry. We wish to thanks Dr J.P. Macher who is the scientific director of FORENAP, Laurent Soufflet and Michel Toussaint of the Computer Science Department of FORENAP for their help.

Notes

1. Operation *add* does not add a face twice in F, i.e. it checks using the hash table whether the given face already belongs to F and actually adds it only if it does not belong to F.
2. Since face i and one of its adjacent faces in v (call it j) belong to the border, face k opposite to i and adjacent to j also belongs to the border if the neighbor of v by face k is not in the object.

References

1. Ehud Artzy, Gideon Frieder, and Gabor T. Herman. The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. *Computer Graphics and Image Processing*, 15:1–24, 1981.
2. Edward R. Dougherty. *Digital image processing methods*, volume 42 of *Optical Engineering series*. Marcel Dekker, Inc., 1994.
3. Dan Gordon and Jayaram K. Udupa. Fast surface tracking in three-dimensional binary images. *Computer Vision, Graphics, And Image Processing*, 45:196–214, 1989.
4. T. Yung Kong and A Rosenfeld. Digital topology : Introduction and survey. *CVGIP*, 48:357–393, 1989.
5. William E. Lorensen and Harvey E. Cline. Marching cubes : a high resolution 3d surface construction algorithm. *ACM Computer Graphics*, 21(4):163–169, July 1987.
6. F. H. Mabin and C. Mongenet. Sewing faces : A topological reconstruction of 6-connected objects bounding surfaces in 3d digital images. *Proceedings of the Fifth International Conference in Central Europe on Computer Graphics and Visualization'97*, vol. 2, February 1997.
7. Serge Miguet and Jean-Marc Nicod. A load-balanced parallel implementation of the marching-cubes algorithm. Research Report 95-24, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, October 1995.
8. A. Rosenfeld, T. Yung Kong, and Angela Y. Wu. Digital surfaces. *CVGIP : Graphical Models and Image Processing*, 53(4):305–312, July 1991.
9. Allen Van Gelder and Jane Wilhelms. Topological considerations in isosurface generation. *ACM Transactions on Graphics*, 13(4):337–375, October 1994.
10. Jane Wilhelms and Allen Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.