Robust unwrapping algorithm for three-dimensional phase volumes of arbitrary shape containing knotted phase singularity loops

Olov Marklund
Luleå University of Technology
Department of Computer Science and Electrical Engineering
SE-97187 Luleå, Sweden

Jonathan M. Huntley
Loughborough University of Technology
Wolfson School of Mechanical and Manufacturing Engineering
Loughborough, Leicestershire LE11 3TU
United Kingdom
E-mail: J.M.Huntley@lboro.ac.uk

Rhodri Cusack
MRC Cognition and Brain Sciences Unit
15 Chauser Road
Cambridge, CB2 7EF, United Kingdom

Abstract. The extension of path-independent 2-D phase unwrapping algorithms, based on placement of branch cut lines between phase singularities of opposite sign, was recently proposed for phase volumes in a paper by Huntley. In 3-D, the singularities were shown to form closed loops, and path independence was achieved by placing branch cut surfaces across the loops. In the current work, we describe in detail an optimized and extended version of Huntley's algorithm. It deals in particular with two aspects that are essential for practical phase volumes: 1. how to close partial loops that pass through arbitrary boundaries separating valid and invalid phase data, and 2. how to select the set of loops having the shortest length. The second algorithm is necessary to deal with ambiguous cases that can arise when the singularities form knots, i.e., two loops pass through a single phase volume element. The performance of the algorithm is demonstrated on 3-D phase maps from two types of medical imaging data: medical resonance imaging (MRI) and x-ray interferometry. © *2007 Society of Photo-Optical Instrumentation Engineers.* [DOI: 10.1117/1.2771652]

Subject terms: phase measurement; medical imaging; data processing.

Paper 060826R received Oct. 23, 2006; revised manuscript received Feb. 9, 2007; accepted for publication Feb. 28, 2007; published online Sep. 7, 2007.

1 Introduction

In many situations where noncontact measurements are recorded and analyzed, the phase, rather than the magnitude, of a recorded signal will represent the important information. Well-known examples are phase stepping techniques employed in 2-D optical metrology applications. However, such measured phase information will by nature be modulo-mapped (wrapped), i.e., we will have $\phi \equiv W[\Phi]$, where the wrapping operator is defined as $W[f] = f(\mathrm{mod}\, 2\pi)$, $\phi \in (-\pi, \pi]$ is the measured wrapped phase, and Φ is the sought continuous phase, and can therefore seldom be analyzed directly as-is. A procedure denoted as phase unwrapping will, in general, have to be applied to ϕ. This procedure aims to reconstruct

$$\Phi = \phi + \nu 2\pi, \tag{1}$$

from the given wrapped data ϕ, i.e., to find the correct integer multiple ν for each data point, resulting in a continuous phase.

Research into the fields of 1-D and 2-D phase unwrapping has been performed for more than two decades, and a large (and growing) number of various approaches to robustly reconstruct Φ from ϕ have been developed and presented.[1] One of the first truly path-independent 2-D phase unwrapping strategies developed is the branch cut method.[2] This method first finds all positions on the phase map ϕ where inconsistencies between neighboring pixels exists, and then puts up barriers (branch cuts) to direct the unwrapping path around these specific positions. The method used to detect the inconsistent positions is based on a theorem that states that (in the continuous case) the line integral of a gradient field over any closed path should yield a zero result, unless the integration path encircles a singular point. The accumulated sum of wrapped discrete differences around a closed path (with defined orientation) connecting four neighboring points on ϕ is examined, and if it is found to be nonzero, one point on the path is tagged as a signed singularity residue. All such possible loops are examined, and a global inconsistency map is produced. Individual residues of different signs are then paired to form branch cut lines. The path chosen during the following unwrapping stage may never directly cross such a cut line. This approach will guarantee an unambiguous unwrapped result, regardless of the specific path. However, no optimal branch cut placement rule is known, and several different strategies have been proposed (see Ref. 3 for a more detailed discussion).

In the growing field of medical engineering applications, measurements will, in many cases, produce 3-D phase volume data. Important examples are results from employing radiology measurement methods such as echo planar magnetic resonance imaging (MRI)[4] and phase contrast x-ray[5] techniques. The need for extensions from 2-D to 3-D phase unwrapping techniques has therefore become an important issue. Recent examples of results obtained in the field of 3-D unwrapping can be found in Refs. 6–14. In Refs. 13 and 14, the unwrapping is carried out in the temporal domain. It should be emphasized that, although the focus in this work lies in the field of medical engineering applica-

0091-3286/2007/$25.00 © 2007 SPIE

tions such as MRI, the presented method can be applied without modifications on any type of 3-D phase data (e.g., time-varying displacement or strain fields measured using whole-field optical techniques such as speckle or Moire interferometry, and reconstruction of wavefronts from digital holography).

In this work we present an implementation of an improved branch surface algorithm based on the results presented in Ref. 6, and analyze its performance. The new algorithm has two significant improvements over the original one: first, it can cope with regions of interest having arbitrary geometries, and second, a robust strategy is developed to cope with the presence of so-called knot points. In Sec. 2 the algorithm is derived in detail, and in Sec. 3 results from applying the method to medical imaging data are presented and discussed.

2 Branch Surface Algorithm

Our initial data are a 3-D discrete wrapped phase volume $\phi_{i,j,k}$, where (i,j,k) denotes the voxel coordinates on a Cartesian grid of size (I,J,K) (however, in many cases ϕ is of an irregular shape and should be considered as embedded in the rectangular grid). As mentioned in the previous section, branch cut algorithms aim to detect inconsistencies in the data and then use this information to find an unwrapping path that does not propagate any errors. The most basic representation of such inconsistency information is denoted residues and is discussed in the next section.

2.1 *Residues and Flag Arrays*

2.1.1 *Residue detection and representation*

In the case of 2-D phase maps,[2] only one residue type can be found (denoted s_z in Ref. 6). A residue s_z is a signed quantity taking values ±1, and is detected through the operation (for the 3-D case)

$$s_z = R[(\phi_{i,j,k} - \phi_{i+1,j,k})/2\pi] + R[(\phi_{i+1,j,k} - \phi_{i+1,j+1,k})/2\pi] + R[(\phi_{i+1,j+1,k} - \phi_{i,j+1,k})/2\pi] + R[(\phi_{i,j+1,k} - \phi_{i,j,k})/2\pi], \tag{2}$$

where the operator $R[.]$ rounds its argument to the nearest integer. The previous finite difference operation can be regarded as a discrete thresholded representation of the line integral theorem discussed in the last section.

In 3-D phase volume data, however, two additional types of residues are encountered. They are denoted s_x and s_y (to be consistent with Ref. 6) and are detected with a similar sum of discrete differences as for s_z, but now over the voxel quadruples

$$\begin{cases} [(i,j,k),(i,j+1,k),(i,j+1,k+1),(i,j,k+1)]; & \text{for } s_x \\ [(i,j,k),(i,j,k+1),(i+1,j,k+1),(i+1,j,k)]; & \text{for } s_y \end{cases}, \tag{3}$$

respectively (see Ref. 6 for a detailed discussion).

The first stage in the algorithm is to apply the prior residue detection operations on the data to identify all existing residues within ϕ. For bookkeeping purposes, all detected residues are entered into a list S, where every row, consisting of five columns, represents a particular residue. The list is of the form shown next.

$$\begin{cases} S(1) \in \{1,2,3\} \\ S(2) = \pm 1 \\ S([3\ 4\ 5]) = (i,j,k) \end{cases}. \tag{4}$$

The first column in a row (representing a certain residue) in S describes the residue type. It is assigned the number one for a type s_x residue, two for a type s_y residue, and three for a type s_z residue. The second column holds the sign of the residue, and columns three to five represent the residue coordinates. At a later stage we discuss how to use the residue information to construct barriers to control which paths can be used during the final unwrapping stage. However, a closely related issue is discussed in the next section.

2.1.2 *Boundary and flag arrays*

The phase function ϕ that we are aiming to unwrap has been defined on a square grid. However, ϕ may not necessarily contain valid phase information throughout the whole grid (one example is the brain scan data seen later in Sec. 3, where the actual valid phase data are embedded in a larger square grid).

It is desirable that the path used during the unwrapping stage is forced to stay within the valid phase volume (and not travel outside into areas consisting solely of noise). The primary reason for this is that in such noise areas the number of detected residues is very large (as is the number of knot points discussed in Sec. 2.2.2), thus making the following residue sorting operation (discussed in Sec. 2.2) unnecessarily time consuming. We therefore need some way to represent the true phase volume boundary.

We assume that a mask array M, i.e., a binary array of the same size as the array ϕ, (I,J,K), and with voxel values set to unity at positions where ϕ consists of valid phase data, and set to zero elsewhere, is available. The procedure to find the true boundary is then straightforward. An array M is often constructed by the application of threshold techniques on magnitude information, corresponding to the phase function under investigation, but techniques where the phase data (or rather the residue density) are used to form a mask have been presented.[14,15]

We introduce four new arrays, B of the same size as M, and F_x, F_y, and F_z, all of size $(I+1,J+1,K+1)$. The B array is used to store information on exactly which voxels make up the true phase boundary and is instrumental during the closing of partial loops (as discussed in Sec. 2.3). In B only voxels residing on the true phase boundary (as determined by M) are set to one and the rest to zero. If the whole phase volume consists of valid phase data, B is constructed as a "frame" of ones enclosing ϕ. The three F arrays all serve as so-called flag arrays, holding information for directing the path used during the unwrapping stage. If the voxel entry with indices (i,j,k) in M is found to be one (and therefore indicating a valid data voxel) and a neighboring voxel entry in M, e.g., $(i+1,j,k)$, is found to be zero (indicating a voxel outside the valid phase volume), then we set the voxel entry with indices (i,j,k) in the corresponding flag array (in this case F_x) to one. For the situation $M(i,j,k)=1$ and $M(i-1,j,k)=0$, we set $F_x(i-1,j,k)=1$. This information indicates that if we, during the unwrapping, are positioned at voxel (i,j,k) and we find that $E_x(i,j,k)=1$, it means that a movement from position (i,j,k) to $(i+1,j,k)$ is forbidden. Also, if at (i,j,k) and $F_x(i-1,j,k)=1$, we are not allowed to move to $(i-1,j,k)$.

Similar rules are of course used for the other two dimensions. If the situation $M(i,j,k)=1$ and $M(i,j+1,k)=0$ is encountered, we set $F_y(i,j,k)=1$, thus stopping the passage from (i,j,k) to $(i,j+1,k)$ during unwrapping, and for $M(i,j,k)=1$ and $M(i,j,k+1)=0$, we consequently set $F_z(i,j,k)=1$.

Also, if $M(i,j,k)=0$ and any of the 26 neighbor voxels $M(i\pm 1,j\pm 1,k\pm 1)$ is found to be one, e.g., $M(i+1,j,k)=1$, then the voxel $(i+1,j,k)$ in B is set to one, indicating that it is a valid boundary voxel and can be used as such by operations discussed in later sections.

In practice, a more compact representation can be used, where all the necessary bookkeeping arrays are combined into one byte array F of size $(I+1,J+1,K+1)$. If we wish to set a flag $F_x(i,j,k)$, we simply set the least significant bit in the byte at position (i,j,k) in F. For setting $F_y(i,j,k)$ and $F_z(i,j,k)$, the second and third least significant bit is used, respectively, and the fourth bit is used for the boundary information held by B. We use for simplicity all four of the previous arrays in the forthcoming discussion.

The flag arrays have now been set up to prevent the path used during the final unwrapping stage to exit the valid phase volume. However, before the actual unwrapping can be performed, we need to also set flags that will make up internal boundaries, so-called branch cut surfaces, that will guide the unwrapping path within the phase volume. The first step is to connect the residues in the list S into larger structures, and this is discussed next.

2.2 *Sorting the Residue List S*

2.2.1 *Initial sorting strategy*

In the 2-D branch cut method, the branch cut lines are formed by selecting pairs of residues s_z of opposite signs and placing a cut line between them. The choice of the particular residues to be paired is somewhat arbitrary, and the decision strategy originally proposed was to do the pairing in such a way that it minimized the total length of all cut lines.[2] However, in Appendix B in Ref. 11, it was shown (the reasoning behind this result is not repeated here) that the residues in a list S detected in a 3-D phase volume must be combined in such a way that they will form closed loops in 3-D space (or partial loops terminating on the phase volume boundary). We have throughout this work adopted the graphical interpretation of residues and loops used in Ref. 6 and explained later.

In Fig. 1 a gray plane with corners made up from four neighboring voxels is shown. These are four voxels used to detect a residue (in this case of type s_x) as discussed in a previous section. The actual residue should be thought of as a vector normal to the plane and passing through the center of the plane. It is also apparent from the figure that such a plane could be seen as one of six faces of one of two cubes (in this case either to the left or right of the plane itself).

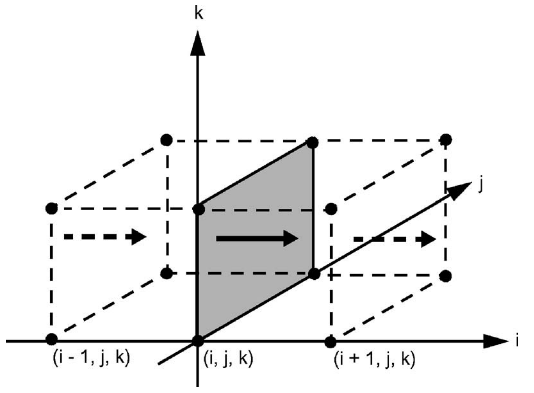

Fig. 1 Graphic depiction of singularity residues (of type s_x).

A very fundamental conservation rule outlined and discussed in Refs. 6 and 11 can be formulated as follows. Let $\sigma=\{S_l^c\}_{l=0}^{N-1}$ be the set of N residues associated with a certain cube c located somewhere in Φ. The sign of each residue is defined with reference to an anticlockwise path as viewed along the outward normal to each face of the cube. With this sign convention, a loop passing through the cube generates a pair of residues of opposite sign at the entrance and exit faces. Then the following result must always be fullfilled,

$$\sum_{l=0}^{N-1} S_l^c = 0, \tag{5}$$

i.e., N must be an even number ($N\in\{0,2,4,6\}$) and σ must consist of an equal number ($N/2$) of negative and positive residues (see Ref. 6 for a detailed discussion). It is worth noting that Eq. (5) is not restricted to $2\times 2\times 2$ cubes, and is in fact applicable to any closed surface lying within the phase volume.

In the following discussion we, for reasons of simplicity, initially make the restriction $N=2$, i.e., only consider one residue of each sign with each cube. This restriction is dropped in a later section where so-called knot points are introduced (in real recorded phase data, the case $N=2$ has proven to be predominant).

We start the sorting procedure with an arbitrary initial residue, e.g., $S_0=(1,1,i,j,k)$ (corresponding to the gray cube face in Fig. 1). The residue notation (the right-hand side) used here is the one outlined in Eq. (4), so here it states that S_0 is a residue of type s_x of positive sign positioned at (i,j,k).

It is clear from Fig. 1 that in this particular example, the initial residue S_0 can be combined with residues entering into the cube to the left of it, e.g., $(1,1,i-1,j,k)$, and with residues leaving the cube to the right of it, e.g., $(1,1,i+1,j,k)$ (illustrated with dashed arrows in Fig. 1).

The complete set of residues that can be connected to a residue of type s_x of positive sign is outlined in Table 1. The same table is also valid for a type s_x residue of negative sign if the entries in the second (sign) column are negated.

These data can be represented in the computer program in the form of look-up tables, in which the source coordinates are given as bias values, i.e., the first row in Table 1,

Table 1 Look-up table for residue type s_x.

Type	Sign	*i* bias	*j* bias	*k* bias
1	1	1	0	0
1	1	−1	0	0
2	−1	0	0	0
2	1	0	1	0
2	1	−1	0	0
2	−1	−1	1	0
3	−1	0	0	0
3	1	0	0	1
3	1	−1	0	0
3	−1	−1	0	1

Table 3 Look-up table for residue type s_z.

Type	Sign	*i* bias	*j* bias	*k* bias
1	−1	0	0	0
1	1	1	0	0
1	1	0	0	−1
1	−1	1	0	−1
2	−1	0	0	0
2	1	0	1	0
2	1	0	0	−1
2	−1	0	1	−1
3	1	0	0	1
3	1	0	0	−1

when compared with S_0 in the example before, would represent the residue $(1,1,i+1,j,k)$. A look-up table like this can thus serve as a convenient way of finding the residues (in the residue list S) that can be combined with a particular residue currently being investigated.

In the same manner, look-up tables for residues of types s_y and s_z can be constructed and they are presented as Tables 2 and 3. The tables are valid for residues of positive sign but can be applied to residues of negative sign if the second (sign) column is negated (precisely as in the s_x case before).

So, if in our small previous example the initial residue S_0 was chosen to be connected to the residue $S_1=(1,1,i+1,j,k)$ [the residue $(1,1,i-1,j,k)$ could have been used instead, the choice between the two is arbitrary], the next step would be to consider S_1 as the residue under examination and find a residue S_2 in the list S that can be connected to S_1. Obviously, as soon as the loop we are assembling consists of at least two residues (S_0 and S_1), there can be at most one valid candidate residue S_2 found in S (if no knot points, discussed later, are present) since one of the two cubes associated with S_1 already includes S_0.

Table 2 Look-up table for residue type s_y.

Type	Sign	*i* bias	*j* bias	*k* bias
1	−1	0	0	0
1	1	1	0	0
1	1	0	−1	0
1	−1	1	−1	0
2	1	0	1	0
2	1	0	−1	0
3	−1	0	0	0
3	1	0	0	1
3	1	0	−1	0
3	−1	0	−1	1

We assume that a valid residue S_2 can be found in S, and it in turn is examined and connected to a suitable residue S_3 from S, and this procedure is repeated until we finally examine a residue S_M that cannot be connected to any of the remaining residues in S.

This can be caused by two different circumstances. First, it may be possible to connect S_M to the initial residue S_0, and this would consequently result in a completed closed loop consisting of $M+1$ residues.

Second, we may not find a matching residue, because S_M is exiting the valid phase volume through the boundary. If this is found to be the case, i.e., no matching residue (including S_0) can be found, we must revisit the initial residue S_0 and also connect it to its other candidate residue [i.e., the one not chosen initially, $(1,1,i-1,j,k)$ in the example before, we denote it S_{-1}].

We then proceed to connect S_{-1} to its matching residue S_{-2} and repeat the connection procedure until again a residue S_{-N} is found that cannot be connected. We now have a partial loop (with both end residues terminating on the volume boundary) of size $M+N+1$. A loop of this kind must be further processed before any branch surface flags can be determined, and this is discussed in Sec. 2.3.

Once a loop, closed or partial, has been completed, the important information about it is stored for subsequent processing. In our implementation, we have chosen the following bookkeeping structure. The loops are stored in a list Λ identical in structure to the list S discussed in Sec. 2.1.1, only now it consists of a number of successive sorted loops. To know where in the list (on which row) a certain loop begins and ends, we use a pointer list P, where every row,

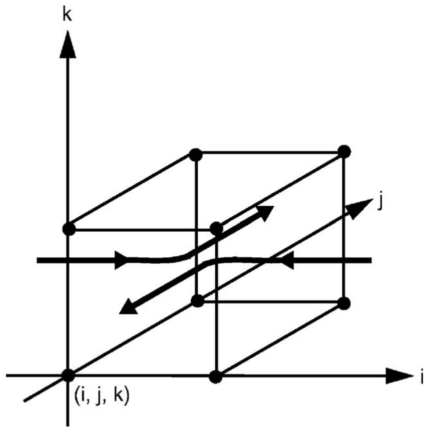

Fig. 2 Graphic depiction of a knot point.

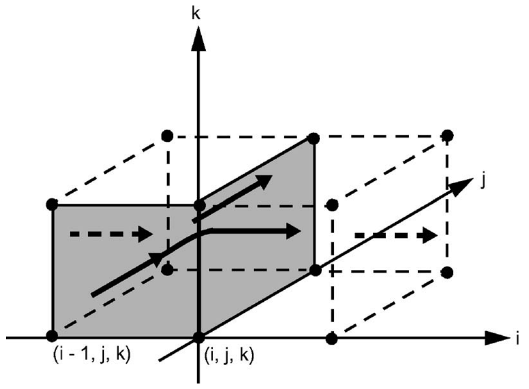

Fig. 3 A sorting scenario where a bifurcation could be introduced.

consisting of three columns, represents a particular loop. Here (for a certain row n in P) $P(1)$ points to the row in Λ where the first residue in the n'th loop is stored. The second column $P(2)$ points to the row in Λ where the last residue in the n'th loop is stored. The third column $P(3)$ holds information about the nature of the specific loop. The value 0 would mean that the loop is a partial loop, and the value 1 would mean that it is closed.

We have outlined basic sorting rules that can be used to find the residue loops in a phase volume. We have, however, left out one crucial problem that is discussed next.

2.2.2 *Knot points*

If every residue in the list S can only be connected to two other residues (like in Fig. 1), the sorting procedure becomes fairly trivial, as demonstrated before. In reality, however, the situation where $N>2$ [in Eq. (5)] will sometimes arise, and this means that more than one loop is passing through the same cube. This gives rise to more complex interconnected structures (cubes where this occurs are referred to as knot points in Ref. 6, and we adopt that notation here). It is worth pointing out that another class of entangled loops exists where two loops may link one another (in the same manner as two links in a chain), but where at no point do the loops share a common $2\times2\times2$ voxel cube. Such situations are easily dealt with by the usual loop contraction rules described elsewhere in the work.

The areas where knot points are most likely to be found are areas with high loop density due to noise. Each individual noise voxel generates its own short closed loop (including typically only four to six residues). However, noise voxels in close vicinity to each other may generate loops that pass through the same cube (as illustrated in Fig. 2). As a consequence, the initial assumption made before, allowing only one residue to enter and one residue to exit a certain cube, must be modified. The new rule states that the number of residues entering a certain cube must be identical to the number of residues exiting the same cube.

For this reason, care must be taken so that no inconsistent loops are formed, i.e., the original look-up tables (e.g., Table 1) will no longer suffice and an extra set of rules must be applied.

Take as an example the situation depicted in Fig. 3. The initial residue $S_0=(2,1,i-1,j,k)$ has been connected to a second residue $S_1=(1,1,i,j,k)$ according to the rules given in Table 2. Of the remaining residues in S, two residues, $(1,1,i-1,j,k)$ and $(1,1,i+1,j,k)$ (illustrated with dashed arrows in the figure), are found to match S_1. However, the cube to the left of S_1 constitutes a knot point, and clearly the residue $(1,1,i-1,j,k)$ cannot be attached to the loop we are assembling without creating a nonphysical branch in the loop structure. Thus the residue (S_0) already connected to the actual residue under investigation (S_1) should always be examined so that the next residue that is connected to the loop belongs to the correct cube. As can be seen from this example, only the rows 1, 3, 4, 7, and 8 in Table 1 are in reality valid choices in this case. It is straightforward to modify the look-up table procedure, so that the cube to which S_0 and S_1 is part of is taken into consideration, thus determining the valid rows in the look-up table.

When knot points are present in the residue list S, the sorting procedure can no longer be made unambiguous. This is illustrated in Fig. 4, where now the cube to the right of the residue $S_1=(1,1,i,j,k)$ constitutes a knot point. In this case, two valid residues that may be connected to S_1 can be found, $(2,1,i,j+1,k)$ and $(1,1,i+1,j,k)$ (indicated with dashed arrows in the figure). We have no way of knowing *a priori* which one of them in reality belongs to the loop we are currently assembling (there must, of course,

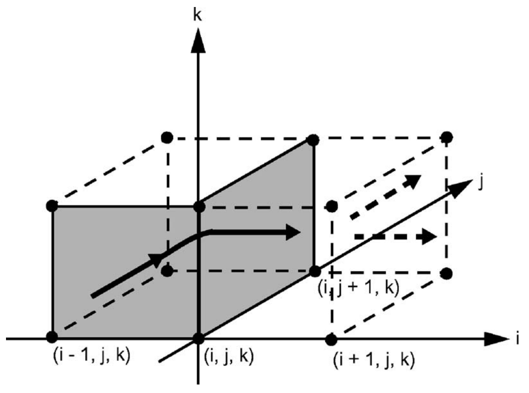

Fig. 4 An ambiguous sorting scenario.

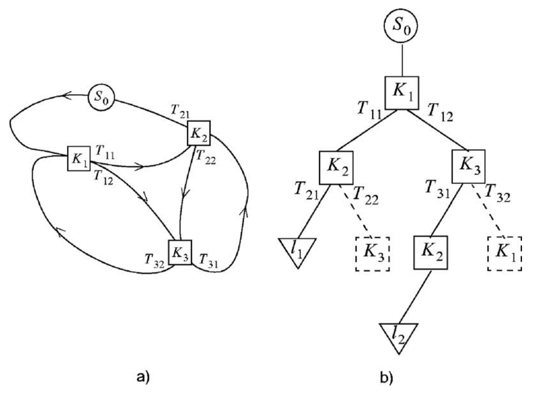

Fig. 5 (a) A nested loop structure. (b) A graphic depiction of a sorting tree used to find the shortest loop.

be another source entering the cube for the conservation reasons outlined before, but since this is not a candidate for joining to S_1, it is not shown in Fig. 4).

This in turn means that instead of sorting the residues in S into a large number of very short loops, we may end up with a situation where a large number of faulty choices are made at knot point positions, resulting in a smaller number of longer and more complex loops. This ambiguity does not necessarily lead to any actual unwrapping errors being introduced during the final stage of the method, but can severely affect the execution time during the computation of the branch surfaces discussed in Sec. 2.4.

Based on the heuristic assumption that most loops found in a noisy phase volume should be short loops associated with individual noise voxels, we have chosen to implement an extensive search routine that, every time a knot point is encountered with multiple choices as a result, builds all possible loop constellations and chooses the shortest one encountered. This is discussed in more detail next.

2.2.3 *Modified sorting routine*

To illustrate the nature of our modified sorting routine, we use a small graphical example outlined in Fig. 5.

In Fig. 5(a) a set of short interconnected loops are depicted, and we examine how the presence of knot points [denoted K_1, K_2, and K_3 in Figs. 5(a) and 5(b)] can influence the residue sorting.

As usual we start out with an initial residue S_0 [top of the tree structure depicted in Fig. 5(b)]. It has proven to be beneficial to the sorting routine if an initial residue is not associated with a knot point (i.e., that it is not one of several residues entering or leaving the same cube). We then repeatedly connect new residues to build a loop. After having added a few residues to the loop, we encounter a knot point [denoted K_1 in Figs. 5(a) and 5(b)] and now have two valid residues [denoted T_{11} and T_{12} in Fig. 5(b)] to choose from.

We could simply choose one of them arbitrarily and continue on with the sorting routine, during which we may encounter new knot points, again making arbitrary choices, and continue on this way until no more residues can be added to the loop. However, as mentioned before, this may result in unnecessarily long and complicated loops that will lead to greatly increased computational complexity in some of the following steps in the method.

Instead, we proceed as follows. We make two copies of the loop segment we have assembled so far (from S_0 up until K_1). We attach T_{11} to one of the copies and T_{12} to the other, and then continue to build these two new loops separately.

If we first look at the loop segment containing T_{11} [represented as the left branch protruding from K_1 in Fig. 5(b)], we find that, after a number of new residues has been added, we encounter yet another knot point [denoted K_2 in Figs. 5(a) and 5(b)]. As a result we again have two valid residues (T_{21} and T_{22}) to consider.

Thus, we must again make two copies of this specific loop segment (from S_0 to K_2 through K_1), attaching T_{21} to one copy and T_{22} to the other.

We then examine the copy including the residue T_{21} and can determine [from studying Fig. 5(a)] that, by adding further residues (from the list S) we will be able to once again connect to our initial residue S_0.

This means that we have constructed a closed loop $l_1 = \{S_0, K_1, K_2, S_0\}$, i.e., a loop originating at S_0 and passing through the knot points K_1 and K_2 before again connecting to S_0. This loop, denoted l_1, is one possible candidate loop including S_0.

We do not discuss the link copy including T_{22} here, but instead concentrate on the loop segment containing the residue T_{12} [represented as the right branch protruding from K_1 in Fig. 5(b)]. We can see from Fig. 5(a) that in this case a new knot point [denoted K_3 in Fig. 5(b)] is encountered, again giving us a choice between two residues [denoted T_{31} and T_{32} in Figs. 5(a) and 5(b)].

If we consider the link copy to which we assign T_{31}, it is clear from Figs. 5(a) and 5(b) that we will again encounter the knot point K_2, and also that if the residue T_{21} is attached to the copy at that point, we will end up at S_0, which means that we have a closed loop $l_2 = \{S_0, K_1, K_3, K_2, S_0\}$.

We have thus ended up with two different completed candidate loops, l_1 and l_2, originating from the initial residue S_0. The examination of all the other possible combinations of link copies and residues (at knot points) will, of course, give rise to further candidate loops, but we do not perform a full investigation in this example. Now we can simply compare the length of all the detected candidate loops and choose the shortest one (say l_1) as the loop associated with S_0. The loop l_1 is then stored in the lists $\{\Lambda, P\}$ discussed in Sec. 2.2.1.

A new initial residue S_0 is then chosen from the list S (from which all residues belonging to l_1 have now been removed) and a new loop is built and stored away. This procedure is repeated until the list S is empty.

In practice, a tree structure algorithm such as the sorting routine discussed before can be made much more effective by the use of different pruning techniques, but such extensions are not discussed in this work.

As mentioned before some of the loops in the completed lists $\{\Lambda, P\}$ may be partial loops with their end residues terminating on the phase volume boundary. All such loops need to be further processed using a technique discussed in the next section.

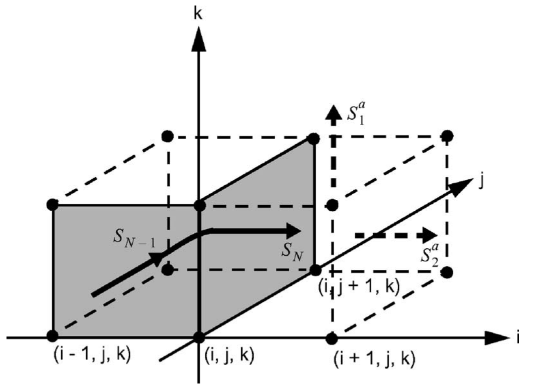

Fig. 6 Graphic depiction of the loop closing strategy.

2.3 *Closing Partial Loops*

When a loop terminates on the boundary of the valid phase volume, we need to close it before the branch cut surface construction algorithm (discussed in the next section) can be applied. The closing procedure consists of adding a segment made up from artificial residues S^a residing on the true phase boundary boundary (as determined by the array B discussed in Sec. 2.1.2) to the partial loop, so that the two endpoints become connected.

Let us assume that $S_N=(1,1,i,j,k)$, shown in Fig. 6, is one of the end residues in our partial loop that needs to be closed. We first determine five possible candidate residues S^a using S_N and $S_{N-1}=(2,1,i-1,j,k)$, Table 1, and the procedure discussed in Sec. 2.1.1 (in this example, the residues S^a would all have to exit the cube to the right of S_N). These new residues (S^a) are not found in S and not all of them are valid candidates for addition to the loop. Only those residues S^a with coordinates (i,j,k), such that the same position in the boundary array B (discussed in Sec. 2.1.2) is set to one, are considered. Also, only residues that can only be connected to S_N (and possibly S_0) and no other residues (S_1 up until S_{N-1}) are considered. The introduction of a residue that does not fulfil this requirement results in an inconsistency at some point in the loop.

In this example, we assume that two artificial residues $S_1^a=(3,1,i,j,k+1)$ and $S_2^a=(1,1,i+1,j,k)$ (indicated with dashed arrows in Fig. 6) have been found to be valid candidates to be connected to the end residues S_N, i.e., they fulfill the rules given before and their coordinates corresponds to the value one in the array B.

The final decision on which of the candidate residues to choose is based on a distance measure. Let us assume that the other end residue (apart from S_N) is given by S_0, and let c_0 denote its coordinate triplet.

As illustrated in Fig. 6, the two candidate residues exit the cube to the right of S_N and enter into two new cubes. The center coordinate triplet for the face of the cube that S_1^a enters is given by $c_1=(i+0.5,j+0.5,k+1)$, and in the case of S_2^a, the center coordinate triplet for the face of the cube it enters is $c_2=(i+1,j+0.5,k+0.5)$.

We then determine the coordinate triplet c_m (in this example, $m\in\{1,2\}$) producing the smallest value $\|c_0-c_m\|$, i.e., the c_m with the shortest distance to c_0, and the corresponding residue S_m^a will be the one among the candidate residues we choose to include in the loop.

The chosen candidate residue S_m^a is now, after being connected to S_N, considered as a new end residue S_{N+1}. We first need to check whether S_{N+1} can be connected to the other end residue S_0, and if this is the case, we have a complete closed loop. We then let this new loop replace the original partial loop in the lists $\{\Lambda,P\}$ (that are updated with information on the new loop length).

If S_{N+1} and S_0 cannot be connected, we proceed to find a new set of candidate residues S^a that can be connected to S_{N+1} (according to the rules discussed before), and again we choose the candidate residue that minimizes the distance to c_0. This procedure is repeated and the new end residues become increasingly close to S_0 until a connection can be made between S^a and S_0 (this will always be the case as long as we have a fully connected boundary). After all partial loops have been closed, we can finally move onto the next step where the loops are used as input information to the branch surface routine, discussed next.

2.4 *Branch Cut Surfaces*

2.4.1 *Surface and loop models*

In the 2-D branch cut method, the purpose of the branch cut lines are to prevent the final unwrapping operation from using any path that would directly cross a cut line. If such a path were to be used, unwrapping errors would propagate throughout the rest of the phase map. The equivalent task that must be performed in the 3-D case is to prevent all paths that would pass through any of the singularity loops.

This can be achieved by considering a closed loop l to be the boundary of a connected surface embedded in 3-D space. If, during the final unwrapping stage, such a surface is encountered, the unwrapping path is forced to go around the surface (and never straight through it).

The optimal shape of such a branch cut surface would probably, by analogy with the 2-D branch cut methods, be the one of all surfaces (having l as its boundary) with the minimum area. However, to determine the parameters for such a surface from a general 3-D space curve is a very demanding operation, so we have settled for a less complex and model-based approach.

Before we discuss the actual surface construction algorithm, we need to define two specific types of loop segments that, when put together in various constellations (in combination with straight segments such as, e.g., the one depicted in Fig. 1, i.e., a string of residues pointing in the same direction) make up different closed loops. We denote these segments L turns and U turns.

An L turn is depicted in Fig. 7, where the two residues outlined with solid lines make up the actual turn (we discuss the two additional residues outlined with dashed lines later). It is made up of two orthogonal connected residues, $(1,1,i,j-1,k)$ and $(2,1,i,j,k)$. As a further illustration, a 2-D version of a section of a loop is depicted in Fig. 8 (it could be considered a 3-D loop segment viewed from above), and an L turn can be seen at the bottom right corner (enclosed in a gray square).

In Fig. 9, a U turn is shown. The turn here consists of the three connected residues outlined with solid lines. It is

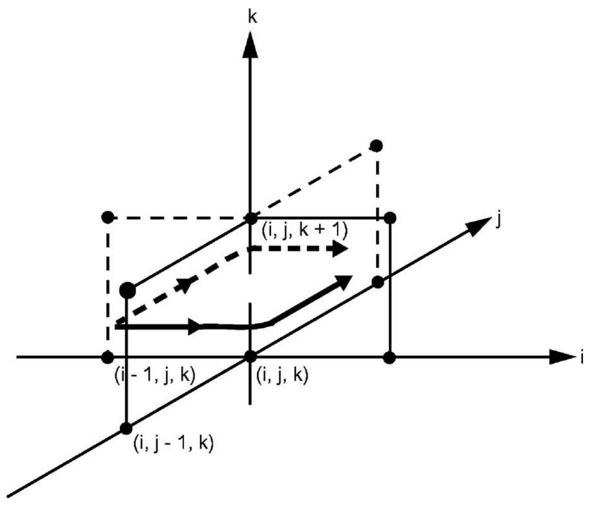

Fig. 7 A graphic model of an *L* turn.

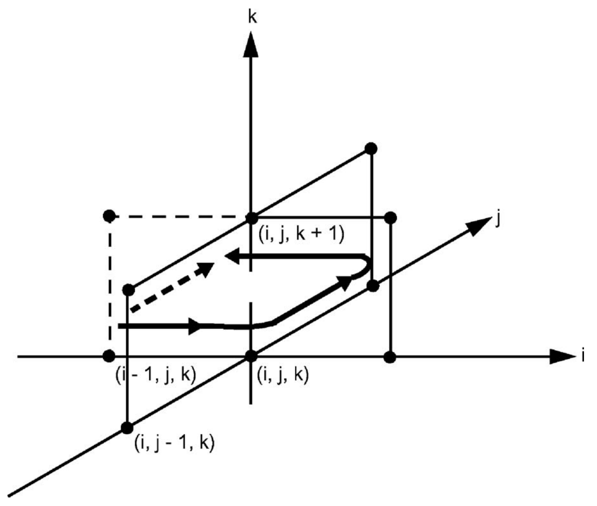

Fig. 9 A graphic model of a *U* turn.

made up of two aligned residues, $(1,1,i,j-1,k)$ and $(1,-1,i,j,k)$, and one orthogonal residue $(2,1,i,j,k)$. A *U* turn can also be seen in the upper left corner (enclosed in a gray square) in Fig. 8.

The branch surface construction method we derive detects *L* and *U* turns in the loop and replaces them with other types of segments. The replacement operations aim to produce a modified loop of shorter length. This consequently shrinks the loop until it finally surrounds just one voxel, and during this shrinking process the flags (like the ones discussed in Sec. 2.1.2) that make up the actual branch surfaces are also set.

2.4.2 *Detection and replacement rules*

U turns. One set of rules that enables us to detect a *U* turn in a closed loop l is given by

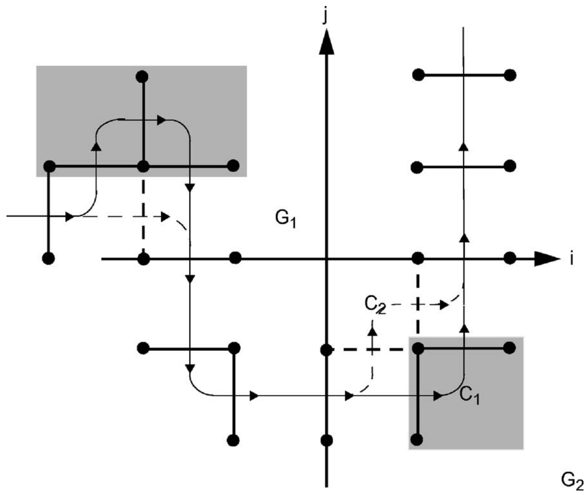

Fig. 8 A 2-D description of the branch surface construction method.

$$\begin{cases} l(n-1,1) = l(n+1,1) \neq l(n,1) \\ l(n-1,2) = -\,l(n+1,2) \\ \|l(n-1,[3\ 4\ 5]) - l(n+1,[3\ 4\ 5])\| = 1 \end{cases} . \tag{6}$$

If all the prior conditions are true for three residues (at positions $n-1$, n, and $n+1$ in the loop), then these three residues make up an *U* turn and should be replaced.

This is done by removing these three residues from the loop and instead connecting the residues at positions $n-2$ and $n+2$ in the loop to one single new artificial residue S_{n-1}^a. This residue should be of the same residue type and have the same sign as the former residue at position n in the original loop, i.e., $S_{n-1}^a(1)=l(n,1)$ and $S_{n-1}^a(2)=l(n,2)$.

The coordinates for the replacement residue S_{n-1}^a must be chosen so that the new residue, together with the three residues in the replaced *U* turn, would make up a minimal closed loop (which would place the four cube faces corresponding to the residues in a cross-configuration, as shown in Figs. 7 and 8). Only one such set of valid coordinates can exist, and in the example in Fig. 7 the new residue, outlined with dashed lines, is thus found to be $S_{n-1}^a=(2,1,i-1,j,k)$.

L turns. To detect an *L* turn in a loop, l is trivial. If the condition $|l(n+1,1)-l(n,1)|>0$ is true, the two residues at positions n and $n+1$ in the loop make up an *L* turn. The replacement strategy is in this case, however, somewhat more complex than for *U* turns. As in the previous case, the residues that make up the *L* turn are removed, but now the residues at positions $n-1$ and $n+2$ in the loop are connected to a segment consisting of two new residues, S_n^a and S_{n+1}^a. The residue types and signs for the two new residues are given by the rules $S_n^a(1)=l(n+1,1)$, $S_n^a(2)=l(n+1,2)$, $S_{n+1}^a(1)=l(n,1)$, and $S_{n+1}^a(2)=l(n,2)$.

The residue coordinates for S_n^a and S_{n+1}^a must (like in the *U*-turn case) be chosen so that they, together with the two original residues making up the *L* turn to be replaced, form a minimal closed loop (which places the four cube faces corresponding to the residues in a cross-configuration, as shown in Figs. 8 and 9). Only one pair of coordinates that fulfills this criteria exists, so there are no ambiguous choices to consider.

However, whereas the replacement of a U turn with one single residue will always reduce the length of the loop, this is not the case when we replace an L turn with a new two-residue segment. Therefore, arbitrarily replacing any detected L turn is not a rational approach, and for this reason one further criterion is introduced, illustrated with an example that follows.

In Fig. 8 the position denoted C_1 represents the center of the cube associated with the original L turn (solid lines), and the point denoted C_2 is the center of the cube associated with a possible replacement segment made up of S_n^a and S_{n+1}^a (dashed lines). Let G represent the position of the geometric center of the loop l, calculated as a "center of mass" position from the coordinates of all current residues in l.

The additional rule states that only if the condition $\|C_2-G\|<\|C_1-G\|$ is true, i.e., C_2 is closer to G than C_1 is, then the L turn under investigation should be replaced.

Thus, if it should turn out that $G=G_1$ (see Fig. 8), the criterion would be fulfilled and the replacement is carried out. If, on the other hand, $G=G_2$ (see Fig. 8) turned out to be the case, C_1 would be closer to G and the loop is left unaltered.

Apart from the previous distance rule, there is one more test that should be performed before a replacement segment is attached to the loop. It is crucial, for the modified loop to remain consistent, that S_n^a (aside from S_{n+1}^a) only be connected to S_{n-1} and no other residue in the loop. Also, S_{n+1}^a should only be able to connect with S_{n+2} (apart from S_n^a). If this condition is not fulfilled, the L turn under investigation should not be replaced, since the addition of the replacement segment (S_n^a,S_{n+1}^a) would create a bifurcation at some position in the modified loop.

Although the actual number of residues making up the modified loop is the same after an eventual replacement of an L turn, it can be argued that the loop has still shrunken toward its geometric center, thus representing a smaller area.

After a replacement operation (of an L or U turn), we must also determine the corresponding flag that should be set. These flags constitute a parameterization of the branch surface we wish to build from the loop, and is discussed next.

2.4.3 *Setting of branch surface flags*

The L turn in the lower right corner of Fig. 8 (enclosed in a gray square) would, as discussed before, be replaced with a new two-residue segment (outlined with dashed lines in Fig. 8) during the loop shrinking procedure. As can be seen, the voxel in the upper left corner of the gray square would be enclosed by the original loop (solid path with arrowheads). A 3-D depiction of this scene is also shown in Fig. 7, and the voxel discussed before is found at position $(i,j,k+1)$. Clearly, during the unwrapping stage, any movement between positions (i,j,k) and $(i,j,k+1)$ would result in a passage directly through the original loop, which must be avoided. This particular movement must therefore be marked as forbidden by setting the appropriate flag in the flag arrays, as discussed in Sec. 2.1.2 [in this example, we would set $F_z(i,j,k)=1$].

As can be seen in Fig. 8, the new loop resulting from the shrinking operation (dashed path with arrowheads) no longer encloses this residue.

To determine which flag to set after a shrinking operation has been performed is straightforward. In the case of both L and U turns, exactly two voxels [the positions (i,j,k) and $(i,j,k+1)$ in the prior example] is common to all (two for L turns and three for U turns) of the cube faces corresponding to the removed residues. There are two such faces for L turns and three for U turns. It is always the passage between these two voxels that should be flagged for forbidden.

Using the rules and methods discussed before, we can now formulate the complete branch surface algorithm.

2.4.4 *Branch surface construction algorithm*

The algorithm will consist of the following steps. It is repeated until no more loops remain in the lists $\{\Lambda,P\}$ (discussed in Sec. 2.2.1).

1. Chose the next (not yet processed) closed loop l from the lists $\{\Lambda,P\}$.
2. Examine all residue positions n and l, according to the rules given in Eq. (3), until a U turn is discovered or the whole loop has been examined.
3. If a U turn was discovered, replace it according to the rules discussed in Sec. 2.4.2 under "U turns," and set the corresponding flag (as discussed in Sec. 2.4.3). If the number of remaining residues in the modified loop l are greater than two, then repeat from step 2 with l. Otherwise, exit the algorithm.
4. Otherwise, examine all the residue positions n in l, according to the rules given in Sec. 2.4.2 under "L turns," until a valid L turn (an L turn that shrinks the loop toward its geometric center) is discovered or the whole loop has been examined.

- If a valid L turn was discovered, replace it according to the rules discussed in Sec. 2.4.2 under "L turns," and set the corresponding flag (as discussed in Sec. 2.4.3). Repeat from step 2 with the modified loop l.
- Otherwise, replace the last visited invalid L turn according to the rules discussed in Sec. 2.4.2 under "L turn," and set the corresponding flag (as discussed in Sec. 2.4.3). Repeat from step 2 with the modified loop l.

When all loops in the list have been transformed into branch surfaces, we can finally proceed with the actual unwrapping.

2.5 *Phase Unwrapping*

The unwrapping stage consists of determining the unknown integer values v in Eq. (1) (as discussed in Sec. 1). In practice, we choose one initial voxel $\phi(i,j,k)$ to represent the corresponding voxel $\Phi(i,j,k)$. All other voxels in ϕ are then to be unwrapped relative to the phase value $\Phi(i,j,k)$. Once a particular voxel [e.g., at position (i,j,k)] has been unwrapped, any of its nearest neighbors [e.g., $(i+1,j,k)$] may be unwrapped through the operation

$$\Phi(i+1,j,k) = \phi(i+1,j,k) - 2\pi W[\Phi(i,j,k) - \phi(i+1,j,k)], \tag{7}$$

as long as the passage between positions (i,j,k) and $(i+1,j,k)$ has not been flagged as forbidden in the flag arrays (as discussed in the former section). If this is the case, the unwrapping procedure must proceed from position (i,j,k) to one of the other neighbors (with no flag set). One way of determining a valid unwrapping path (not including any forbidden passages) is to employ a flood fill approach as follows.

We chose an initial starting voxel, e.g., at position (i,j,k) in ϕ, as $\Phi(i,j,k)$. We then examine its six nearest neighbors at positions $(i+1,j,k)$, $(i,j\pm1,k)$, and $(i,j,k\pm1)$. If while examining a particular neighbor voxel, e.g., at $(i+1,j,k)$, it is found that the passage between positions (i,j,k) and $(i+1,j,k)$ is not flagged as forbidden, the examined voxel is unwrapped relative to $\Phi(i,j,k)$, otherwise it is left unaltered. Once all six neighbor voxels have been examined, those that could be unwrapped relative to $\Phi(i,j,k)$ are used in the further process. If, e.g., $\phi(i+1,j,k)$ could be unwrapped, then all of its nearest neighbors (that have not yet been unwrapped) are examined and, if possible, unwrapped relative to $\Phi(i+1,j,k)$. This procedure ensures that all voxels in ϕ are visited from a valid neighbor voxel (i.e., an allowed passage is made) as long as the phase volume ϕ does not consist of several unconnected regions. If this is the case, then the flood fill algorithm must be applied repeatedly, each time choosing a starting voxel that has not yet been visited.

2.6 *Short Summary of the Branch Surface Method*

In this section, a brief summary of the different algorithms, employed in the branch surface method outlined, is presented.

1. First, the boundary of the valid phase volume (in some instances this comprises the whole square grid) is determined and properly parametrized using the four arrays B, F_x, F_y, and F_z (as discussed in Sec. 2.1.2).
2. Then all residues within the valid phase volume are detected using the rules presented in Eqs. (2) and (3). Information about the residues such as type, sign, and position are then stored in a residue list S (as discussed in Sec. 2.1.1).
3. The residues in the list S are then sorted into closed or partial loops (i.e., loops with end residues terminating on the volume boundary). The result is a set of two new lists $\{\Lambda, P\}$ discussed in Sec. 2.2.1. A sorting routine able to deal with sets of interconnected loops is outlined in Sec. 2.2.3.
4. All partial loops found in the lists $\{\Lambda, P\}$ must be closed before they can be further processed, and a procedure that solves this problem by finding the shortest path, situated on the volume boundary, between the end residues is discussed in Sec. 2.3.
5. The set of closed singularity loops (represented by the lists $\{\Lambda, P\}$) is then further processed to form a set of closed branch cut surfaces (having the loops as their boundaries). The purpose of the surfaces is to guide the final unwrapping process. A method developed to solve this task was outlined in Sec. 2.4.
6. The final unwrapping is performed by employing a flood fill strategy where the voxel queue (determining the order in which the voxels in the phase volume are unwrapped) is built under the restrictions imposed by the flag arrays.

3 Results and Discussion

As was mentioned in the introduction, medical imaging is one important area where 3-D phase data are encountered. In this section, we illustrate our implementation with two medical engineering applications comprising phase volumes acquired with echo planar MRI and x-ray phase contrast interferometry, and discuss the results. We have also included an example where computer-generated data were used to demonstrate the method's ability to cope with noise.

3.1 *Medical Imaging Data*

3.1.1 *X-ray of liver*

In Fig. 10(a), a stack of 2-D slices from a 3-D x-ray phase contrast recording of a human liver is shown. The measurements were performed at University of Tokyo, Japan. The sampled phase volume is of size $450\times100\times100$ voxels. The interferometer used to acquire the data was a Mach-Zehnder design. The phase volume comprises two spatial axes, with the third axis representing the angle of rotation of the sample in one of the two interfering x-ray beams. A total number of 3735 residues were detected in the volume, and the residues were subsequently sorted into 379 singularity loops, of which seven were found to be partial loops (with end residues on the volume boundary) that needed to be closed. Of the total number of loops, 294 consisted of eight residues or less (the larger loops were due to some shear structures in the sampled volume). In the original residue table, 295 knot points were found, and the loops were formed using the sorting routine outlined in Sec. 2.2.3 (i.e., building the set of loops with the overall smallest average number of residues associated with the loops).

The phase volume was successfully unwrapped using the branch surface method, summarized in the previous section, and a stack of slices from the actual unwrapped volume can be seen in Fig. 10(b). In Fig. 10(c), we have highlighted the singularity loops found in the region of the phase volume indicated by the black box in Fig. 10(b). As can be seen, the loops are quite small and no entanglement (knot points) are present in this particular subvolume.

The algorithm described in Secs. 1 and 2 was coded in MATLAB with computationally demanding sections converted to compiled C code. The unwrapping of the previous phase volume took 40 s on a Sun Blade 100 work station with 1 Gb of RAM.

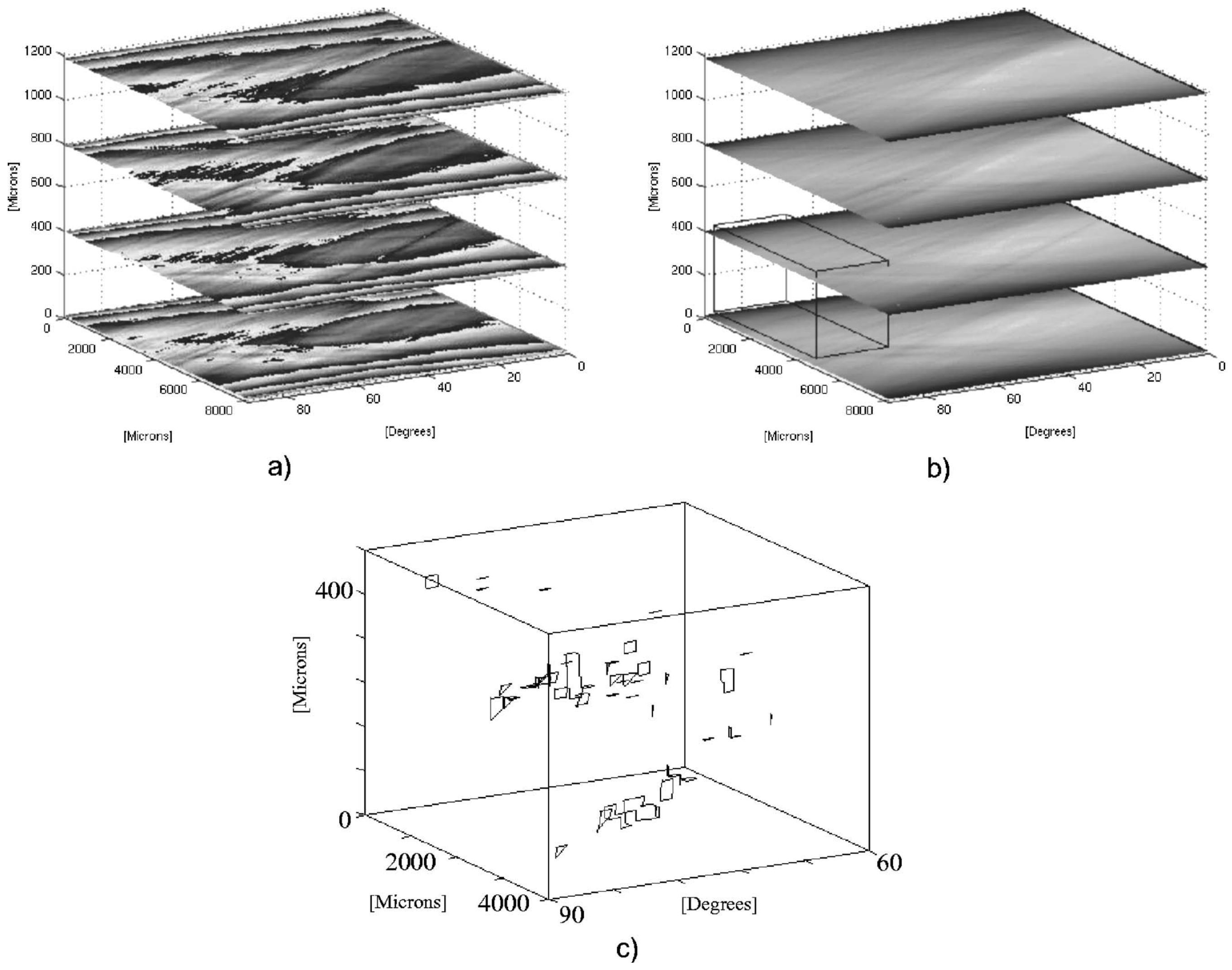

Fig. 10 (a) Slices from a 3-D phase contrast x-ray volume. (b) The corresponding slices from the unwrapped version of the volume in (a). (c) Singularity loops detected in a subvolume of the phase volume in (a).

3.1.2 *Magnetic resonance image of brain*

A stack of 2-D slices from a 3-D MRI scan of a human brain is shown in Fig. 11(a). The phase volume, of size $64\times256\times64$ voxels, was recorded at the Wolfson Brain Imaging Centre by researchers at the MRC Cognition and Brain Sciences Unit, Cambridge, United Kingdom. Unlike the previous example, the three axes of the phase volume in this case represented the three spatial axes of the sample. The phase data encoded magnetic field strength. They were acquired using a 3-D gradient echo sequence with a matrix size of $64\times256\times64$ and a field of view $256\times256\times256\ \text{mm}^3$. Two volumes were acquired, differing only in their echo times (TE), which was either 1 or 10.104 ms. The difference between these maps was chosen to be an integral multiple of the precession time for the signal from fat-based tissues at 3 Tesla. Each acquisition took 2 min 6 s.

In this particular phase volume, the total number of detected residues was 2898, and the sorting and closing operations resulted in 520 closed loops. In this case, 481 partial loops were found, and this is due to the fact that a large percentage of the residues are present near the boundary of the phase volume (which corresponds to the boundary between the brain and the cranial bone structure in the measured head). Also, as can be seen in Fig. 11(a), the measured brain scan data has an irregular boundary, embedded in a rectangular measurement grid, and thus the valid phase data must in this case be separated from the background (as discussed in Sec. 2.1.2). The special segmentation operation used on this kind of brain scan data is described in detail in Ref. 13. However, since it is very hard to get a perfect separation between the two regions, the loops associated with residues near or on the boundary will in many cases be truncated. The corresponding slices from the unwrapped volume can be seen in Fig. 11(b).

For this kind of data, the majority of the loops tend to be very short and the number of knot points very limited (in this specific example, 80 such points were detected).

A large number of examples of this kind of brain scan data has been successfully unwrapped using the branch sur-

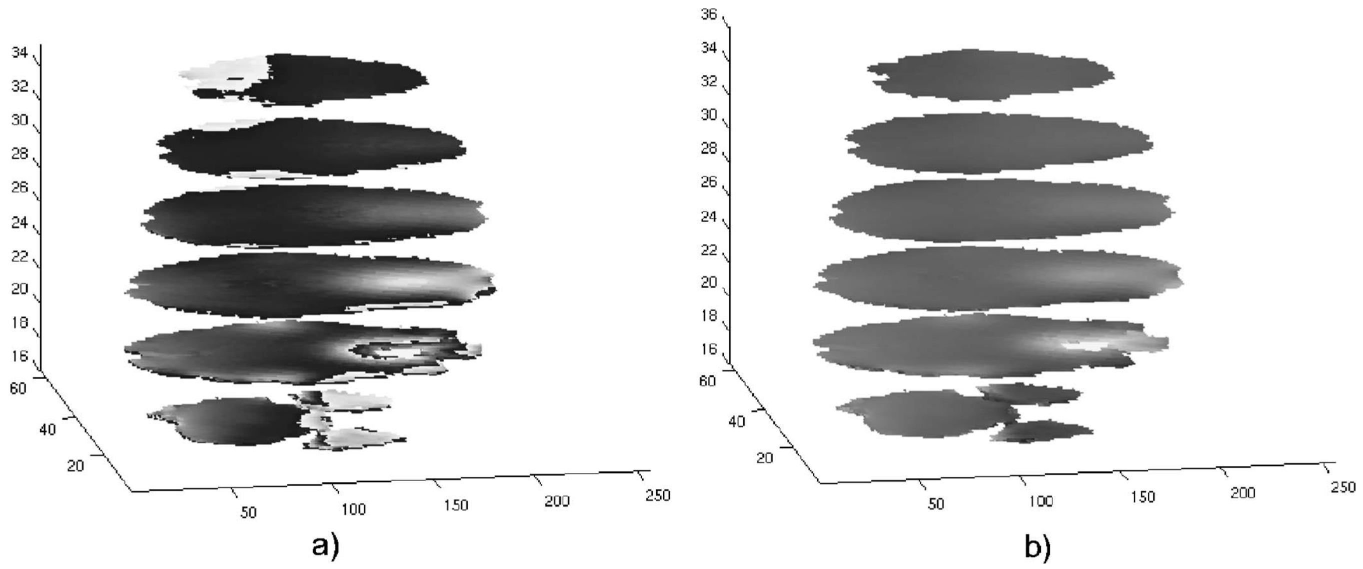

Fig. 11 (a) Slices from a echo planar MRI volume. (b) The corresponding slices from the unwrapped version of the volume in (a).

face method, and the typical computation time (on a Sun Blade 100 work station) is between 5 and 10 s.

3.2 *Computer-Generated Data*

In Fig. 12(a), one slice from a computer-generated phase volume $\tilde{\Phi}$ is shown. First, the noise-free phase was computed as $\Phi(i,j,k)=4\pi[(i/16)^2+(j/16)^2+(k/16)^2]$, with (i,j,k) all in the range $[-16\ldots16]$. The noisy phase volume $\tilde{\Phi}$ was produced by adding zero-mean white Gaussian noise with variance 0.5 to the components $\sin(\Phi)$ and $\cos(\Phi)$, and extracting the noisy wrapped phase.

The number of detected residues in the noisy phase volume was found to be 14,008, a fairly high number considering the total number of voxels in the volume. Also, the number of detected knot points was high: 3615 such points were found. However, by applying the modified sorting strategy outlined in Sec. 2.2.3, the method managed to successfully unwrap the noisy phase volume. A slice from the unwrapped volume, corresponding to the wrapped data in Fig. 12(a), can be seen in Fig. 12(b). As a comparison, and to demonstrate the strength that the reduced ambiguity introduces for the 3-D branch surface method, we include the result from applying the 2-D branch cut method (the method by Goldstein, Zebker, and Werner[16] as implemented in Ref. 1) on the single 2-D slice from Fig. 12(a). As can be clearly seen in Fig. 12(c), where the result is displayed, the 2-D method has been unsuccessful in several large regions of the phase map.

4 Conclusions

We describe and discuss an implementation of a 3-D branch surface method. The method is designed to be able to deal with the situation where constellations of nested singularity loops introduce ambiguity into the loop sorting routine. It is also able to close all partial loops with end residues terminating on the boundary of the valid phase volume, and a model-based branch surface construction algorithm is used to prevent the unwrapping path from passing through any of the phase singularity loops within the volume. The method successfully unwrapped 3-D medical imaging data, and is able to cope with large quantities of noise. The cor-

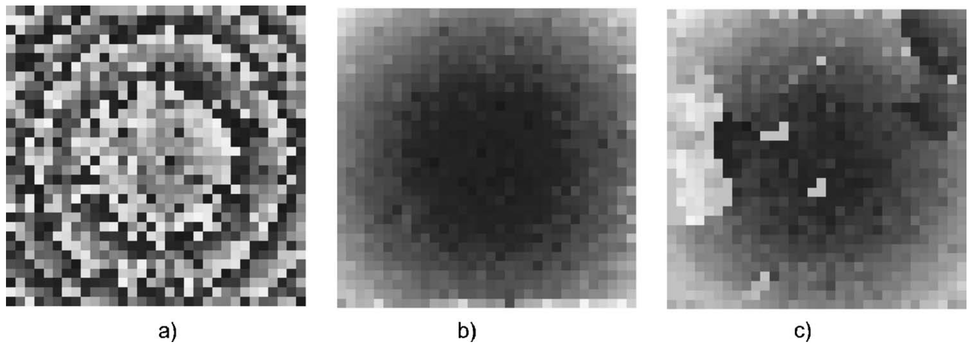

Fig. 12 (a) A slice from a computer-generated noisy phase volume. (b) The corresponding slice from the unwrapped version of the noisy phase volume. (c) The result of applying a 2-D branch surface method on the slice in (a).

rectness of the unwrapped results can be verified by the fact that the underlying field quantities of our measured data are physically constrained to be smoothly varying functions of position. Without a correct branch surface placement, very large and nonphysical gradients will be found in the final result, so success means that we are recovering a distribution that is consistent.

Many useful quantities can be encoded in the phase of acquisitions in medical imaging. For example, in magnetic resonance imaging, phase can be used to measure temperature, velocity, and acceleration, as well as magnetic field strength. New acquisitions are developing continuously, and there is also a strong demand for new robust automatic analysis procedures. This is particularly important in clinical environments where interventions may be made on the basis of the images obtained. Our new 3-D algorithm has strong potential as such a tool in critical situations.

There are many advantages to the branch cut method in 2-D that also hold in 3-D. The algorithm developed here is fast. The time taken for branch cut methods is roughly proportional to the volume to be unwrapped (since the number of residues will, in general, be proportional to the volume size) in contrast to tiling methods[1] for which the time taken grows combinatorialy as the volume size increases. This is especially important in 3-D, as the number of regions can be very large. Branch cut methods also do not show the disadvantage of least-squares procedures, which is that they can show a consistent bias,[1] generally underestimating phase gradients.

While we have applied our 3-D unwrapping algorithm to medical imaging data, the procedure is generic and may well have applications outside this field. In summary, our new robust fast 3-D algorithm has strong potential as a phase analysis tool in many situations.

Acknowledgments

This work was funded by the Medical Research Council, United Kingdom. The authors are grateful to Atsushi Momose, University of Tokyo, and the research staff at the MRC Cognition and Brain Sciences Unit for kind permission to use the medical imaging data presented in this work. J.M.H. is also grateful to the Royal Society and Wolfson Foundation for a Royal Society-Wolfson Research Merit Award.

References

1. D. C. Ghiglia and M. D. Pritt, *Two-Dimensional Phase Unwrapping; Theory, Algorithms and Software*, John Wiley and Sons, New York (1998).
2. J. M. Huntley, "Noise-immune phase unwrapping algorithm," *Appl. Opt.* **28**, 3268–3270 (1989).
3. R. Cusack, J. M. Huntley, and H. T. Goldrein, "Improved noise-immune phase-unwrapping algorithm," *Appl. Opt.* **34**, 781–789 (1995).
4. P. Mansfield, "Multi-planar image formation using NMR spin echoes," *J. Phys. C* 55–58 (1977).
5. K. Hirano and A. Momose, "Investigation of the phase shift in x-ray forward diffraction using a x-ray interferometer," *Phys. Rev. Lett.* **76**, 3735 (1996).
6. J. M. Huntley, "Three-dimensional noise-immune phase unwrapping algorithm," *Appl. Opt.* **40**, 3901–3908 (2001).
7. R. Cusack and N. Papadakis, "New robust 3D phase unwrapping algorithms; application to magnetic field mapping and undistorting echo-planar images," *Neuroimage* **16**, 754–764 (2002).
8. M. Jenkinson, "Fast, automated N-dimensional phase unwrapping algorithm," *Magn. Reson. Med.* **49**, 193–197 (2003).
9. L. An, Q. S. Xiang, and S. Chavez, "A fast implementation of the minimum spanning tree method for phase unwrapping," *IEEE Trans. Med. Imaging* **19**, 805–808 (2000).
10. M. F. Salfity, J. M. Huntley, M. J. Graves, O. Marklund, R. Cusack, and D. Beauregard, "Advanced higher-dimensional phase unwrapping algorithms for phase contrast magnetic resonance velocity imaging," *J. R. Soc., Interface* **3**, 351–469 (2006).
11. M. F. Salfity, P. D. Ruiz, J. M. Huntley, M. J. Graves, R. Cusack, and D. A. Beauregard, "Branch cut surface placement for unwrapping of undersampled three-dimensional phase data: application to magnetic resonance imaging arterial flow mapping," *Appl. Opt.* **45**, 2711–2722 (2006).
12. S. Chavez, Q. S. Xiang, and L. An, "Understanding phase in MRI: A new cutline phase unwrapping method," *IEEE Trans. Med. Imaging* **21**, 966–977 (2002).
13. Q. S. Xiang, "Temporal phase unwrapping for cine velocity imaging," *J. Magn. Reson Imaging* **5**, 529–534 (1995).
14. G. Z. Yang, P. Burger, P. J. Kilner, S. P. Karwatowski, and D. N. Firmin, "Dynamic range extension of cine-velocity measurements using motion-registered spatiotemporal phase unwrapping," *J. Magn. Reson Imaging* **6**, 495–502 (1996).
15. S. M. Smith, "Fast robust automated brain extraction," *Hum. Brain Mapp* **17**, 143–155 (2002).
16. R. M. Goldstein, H. A. Zebker, and C. L. Werner, "Satellite radar interferometry: Two-dimensional phase unwrapping," *Radio Sci.* **23**, 713–720 (1988).

Olov Marklund received his MSc and PhD degrees in electrical engineering from Luleå University of Technology in 1993 and 1998, respectively. He is currently working as a senior lecturer at the Department of Computer Science and Electrical Engineering at Luleå University of Technology. His research interests include optical metrology and multidimensional signal processing.

Jonathan M. Huntley received his BA in physics and PhD degrees from the University of Cambridge in 1983 and 1987, respectively. He was a Royal Society research fellow at the Cavendish Laboratory, Cambridge, from 1989 until 1994, reader at Loughborough University from 1994 to 1999, and was appointed professor of applied mechanics at Loughborough in 1999. His research interests include speckle metrology, phase shifting interferometry, and optical coherence tomography.

Rhodri Cusack obtained a masters in physics from the University of Cambridge in 1993, and then a PhD in psychology from the University of Birmingham in 1997. He is now a senior investigator scientist for the Medical Research Council's Cognition and Brain Sciences Unit in Cambridge, and focuses on methodological development for MRI and cognitive neuroscience.