A Fast Algorithm for Liver Surgery Planning

Fajie Li[1], Xinbo Fu[2], Gisela Klette[3], and Reinhard Klette[4]

[1] College of Computer Science and Technology
Huaqiao University, Xiamen, Fujian, China
[2] Xiamen ZhiYe Software Engineering Company Limited, Xiamen, Fujian, China
[3] School of Computing & Mathematical Sciences, Auckland
University of Technology Private Bag 92006, Auckland 1142, New Zealand
[4] Computer Science Department, The University of Auckland
Private Bag 92019, Auckland 1142, New Zealand
li.fajie@hqu.edu.cn

Abstract. Assume that a simplified liver model consists of some vein cells and liver cells. Such a liver model contains two kinds of components, the vein component and the liver components, each of them consists of cells which are 26-connected. The vein component has a tree-shape topology. Suppose that the vein component has already been cut into two parts, and one of them is diseased. Liver surgery planning systems need to design an algorithm to decompose the liver components into two kinds of subsets, one (usually just one component) that has been affected by the diseased vein component while the other one is still healthy. So far, existing algorithms depend heavily on surgeons' personal expertise to detect the diseased liver component which needs to be removed. We propose an efficient algorithm for computing the diseased liver component which is based on the diseased vein component, and not on surgeons' personal manipulations.

1 Introduction and Related Work

In 2000 it was estimated that liver cancer remains the fifth most common malignancy in men and the eighth in women worldwide, and the number of new cases is 564,000 per year [1]. Liver resection is an often cure for primary liver cancer. The literature reports many liver resection surgical techniques. For example, see [7, 10].

Existing liver surgery planning usually requires surgeons' personal expertise to interact during surgery. For example, the planning stage proposed in [10] needs branch labelling which is the most time-consuming step in the planning procedure and usually involves some trial and error on the user's part. *Mint Liver*, a novel 3D image analysis software for liver resection, has to be used by experienced hepatic surgeons for designing the new transection plan. The preoperative planning in [13] calculates the vascular perfusion area using an algorithm based on direction and diameter of the portal vein branch. Reference [8] proposes a probabilistic atlas for liver surgery planning, and [3] discusses a deformable cutting plane for virtual resection where 3D interaction techniques are used to specify and to modify the clip geometry by medical doctors. The system of [5] relies on the surgeon's capacity to perform a mental alignment between the resection map and the operating field. The squared Euclidean distance transform was applied in [12] for approximately computing the liver part which should be removed.

In this paper, we apply basic ideas of digital geometry [4] to propose an algorithm for computing the diseased part of a liver. Our algorithm is both time-efficient[5] and "accurate". The problem to be solved is as follows: Let S_l be the set of cells in the given 3D input image classified to be liver cells. Set S_h contains all cells classified to be healthy vein cells. Set S_d contains all the detected diseased vein cells. We have to calculate that part of the liver which is affected by diseased vein cells.

The *accurate* solution for this problem is defined by the maximum subset $A \subseteq S_l$ such that A "is affected" (still to be defined) by the set S_d of diseased cells. This is an optimization problem. It is solved in this paper by computing exactly three sets S_{a_b}, S_{a_h}, and S_{a_d} such that $S_l = S_{a_b} \cup S_{a_h} \cup S_{a_d}$ where S_{a_b} is "affected" by both S_h and S_d, and S_{a_i} is "affected" by S_i only, for $i = d, h$.

Existing algorithms compute only approximately the liver part which should be removed; so far there is no exact specification of the part which should be removed. S_{a_b} can be understood as being a set of boundary cells "between" healthy liver cells and the diseased vein cells. $S_{a_b} \cup S_{a_d}$ is finally the set of all liver cells which should be removed.

The paper is structured as follows: In Section 2 we define some notions and notations which are used in our algorithms. In Section 3 we describe and explain the algorithms whose time complexities are analysed in Section 4. We show some experimental results in Section 5 and conclude the paper in Section 6.

2 Basics

Image data are given in a regular 3D grid of grid constant $\theta_0 > 0$. We only consider finite sets in this paper. We identify a (grid) *cell* with its centroid, which is a grid point. In this section we discuss the 2D case (i.e. one slice of the 3D data) only; generalization to 3D is straightforward. We also consider a multi-grid approach by varying the given grid constant. Let $\theta > 0$ be an arbitrary grid constant. Let $d_e(p, q)$ be the Euclidean distance between two points p and q in the plane (e.g. centroids of cells). Given two sets A and B in the plane, define $d_{min}(A, B) = \min\{d_e(p, q) : p \in A \wedge q \in B\}$. In particular, $d_{min}(A, B)$ is denoted by $d_{min}(p, B)$ if A contains only a single point p. Define $d_{max}(A, B) = \max\{d_e(p, q) : p \in A \wedge q \in B\}$.

We consider healthy vein cells (type-h), diseased vein cells (type-d), and liver cells (type-l), each *cell* being one voxel. Considering the 2D case only in this section, a *cell* is a pixel. Let S_i be the set of type-i cells, for $i = d, h, l$. Let S be the union of those three sets. See Fig. 1 on the left. Cells can be uniquely either of *type-d*, *type-h*, *type-l*, or of no assigned type at all (i.e. background cells).

Thus, the three types define sets S_d, S_h, and S_l of cells in the plane which are pairwise disjoint. We say that set S_l is *only affected* by S_h if for each pixel $p_l \in S_l$, $d_{min}(p_l, S_h) < d_{min}(p_l, S_d)$; analogously, we can also have that set S_l is only affected by S_d. We say that S_l is *affected* by both S_h and S_d if for each pixel $p_l \in S_l$, $d_{min}(p_l, S_h) = d_{min}(p_l, S_d)$.

[5] For example, [10] reports about an average labelling time of about 17 minutes, depending on the data set.

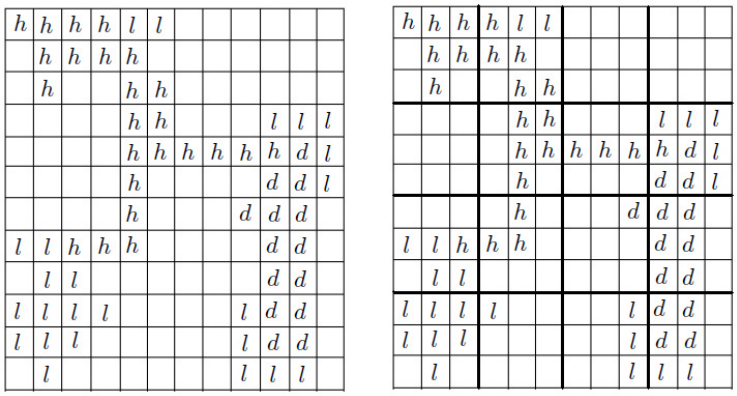

Fig. 1. *Left*: A set S of labelled cells for original grid constant θ_0, with h for type-h, d for type-d, and l for type-l. *Right*: Cells for grid constant $\theta = \theta_0 \times 3$.

Let Ω be the rectangle of minimum size which contains all the cells of size $\theta_0 \times \theta_0$ of the given set S. For a positive integer m, let $\theta = \theta_0 \cdot m$. We analyse Ω by using grid constant θ. See Fig. 1, right, for an example. Set Ω is subdivided into larger $\theta \times \theta$ cells (*supercells*) which contain several $\theta_0 \times \theta_0$ cells. Case $m = 1$ is possible and defines the original constant θ_0.

At constant θ, set S can be described by an (undirected weighted) θ-*graph* $G = [V, E]$ based on 8-adjacency. Each vertex in V corresponds to one $\theta \times \theta$ cell which contains at least one of the labelled $\theta_0 \times \theta_0$ cells of set S. Two vertices v_1 and v_2 in V define an edge $e = \{v_1, v_2\}$ iff the corresponding cells C_1 and C_2 are 8-adjacent. Such a graph structure is used in Procedures 1 and 2, and in the main algorithm in Subsection 3.2. See Fig. 2 for an example.

We use standard adjacency definitions of digital geometry to specify four different types of adjacency sets. Consider grid constant θ. For a cell C, L_∞-distances $i \geq 0$ define layers $N(C, 1, i)$ of $\theta \times \theta$ cells around this cell. In general, we have $8 \times i$ cells in set $N(C, 1, i)$, as already discussed in [11]. The four *corner cells* in $N(C, 1, i)$ have a Euclidean distance $\sqrt{2} \times \theta \times i$ to cell C. We call $N(C, 1, i)$ the *first* adjacency set of supercell C with radius i (i.e. cells *near* to C but not including C and layers of radius $j < i$, thus not a neighbourhood in the sense of topology which would include C).

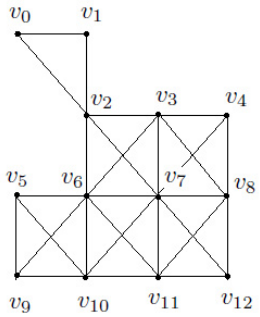

Fig. 2. Illustration of the θ-graph corresponding to Fig. 1, right. Weights are either θ or $\theta\sqrt{2}$.

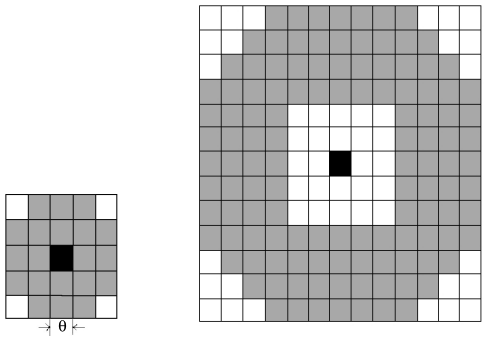

Fig. 3. *Left*: adjacency set of a type-$sldoh$ supercell C, containing twenty grey cells. *Right*: adjacency set $N(C, 3, 5, 2)$ of a type-sl supercell C, shown by grey cells.

We call $N(C, 2, i) = \cup_{j=1}^{i} N(C, 1, j)$ the *second* adjacency set of supercell C with radius i (also not including supercell C). Furthermore, we also use adjacency sets defined by the Euclidean metric L_2; let $N(C, 3, r)$ be the set of all cells C' with $d_{max}(C', C) \leq r$. We call $N(C, 3, r)$ the *third* adjacency set of supercell C with radius r (not including supercell C). So far, this is all very basic digital geometry and just listed here for specifying the used notation.

For our particular application context, we define that an *type-*$sldoh$ cell is one supercell (i.e. with edge length $\theta_0 \cdot m$) that contains at least one type-l cell (with edge length θ_0) but also at least one type-h or type-d cell. A *type-*sl cell is one supercell that contains type-l cells only. For example, the $\theta \times \theta$ cell in Fig. 1 (right), corresponding to vertex v_1 in Fig. 2, is of type-$sldoh$ because it contains five type-h cells and two type-l cells; the cell in Fig. 1 (right), corresponding to v_9, is of type-sl because it contains seven cells which are all of type-l. The following adjacency set definitions for type-$sldoh$ or type-sl cells are motivated by the particular application, and they have been heuristically derived from the given (extensive) image data. Those adjacency sets can be modified without affecting the basic ideas of the algorithms. Assume grid constant θ. The *adjacency set* of a type-$sldoh$ supercell C is defined by $N(C, 3, 2 \times \sqrt{2} \times \theta)$. See Fig. 3, left. We make use of this in Lines 1–8 of Procedure 1 as shown in Fig. 6. For defining the adjacency set of a type-sl supercell C, let m and n be two non-negative integers with $n < m$. Set

$$N(C, 4, m, n) = N(C, 3, m(\sqrt{2} \times \theta)) \setminus N(C, 2, n)$$

is the *fourth* adjacency set of supercell C. Figure 3, right, illustrates a set $N(C, 4, 5, 2)$. We make use of this in Lines 1–9 of Procedure 2.

Definitions given in this section can be generalized for the 3D case, and we do not specify them here because those generalizations are straightforward.

3 Algorithms

Assume that we have a number of slices of 2D images from a CT-scan. Each 2D image contains $\theta_0 \times \theta_0$ cells of type-d, type-h, type-l, or "other" cells (that is, background

cells). Each type-h cell represents a healthy vein cell. Each type-d cell represents a diseased vein cell. Each type-l cell represents a liver cell. Our goal is to classify type-l cells $p_l \in S_l$ depending on the value $d_{min}(p_l, S_h)$ and $d_{min}(p_l, S_d)$.

In this section, we describe a naive brute-force algorithm (Algorithm 1), its improved version (Algorithm 2), and then a more efficient main algorithm (Algorithm 3). These three algorithms are used to classify type-l cells based on type-d and type-h cells. As usual, S_i is the set of type-i cells, for $i = d, h, l$.

3.1 A Brute-Force Algorithm and Its Improved Version

The idea of Algorithm 1 is simple. We scan through the set S_l of all type-l cells. For each cell $p_l \in S_l$, we decide to which subset (S_{a_b}, S_{a_d} or S_{a_h}) cell p_l belongs to by simply testing the values of $d_{min}(p_l, S_i)$, for $i = d, h$: If $d_{min}(p_l, S_h) < d_{min}(p_l, S_d)$, then let p_l be in S_{a_h}; else, if $d_{min}(p_l, S_h) > d_{min}(p_l, S_d)$, then let p_l be in S_{a_d}; otherwise let p_l be in S_{a_b}. The pseudocode is given in Fig. 4.

Algorithm 1 (A brute-force algorithm for separating type-l cells)
Input: Three sets S_d, S_h, and S_l such that S_i contains type-i cells, where $i = d, h, l$.
Output: Three sets S_{a_b}, S_{a_d} and S_{a_h} such that $S_l = S_{a_b} \cup S_{a_d} \cup S_{a_h}$, where S_{a_b} is affected by both S_d and S_h, and S_{a_i} is affected by S_i only, for $i = d, h$.
Pseudocode: See Fig. 4.

```
 1: Let S_{a_b} = S_{a_d} = S_{a_h} = ∅.
 2: for each p_l ∈ S_l do
 3:    Go through S_h for computing d_min(p_l, S_h).
 4:    Go through S_d for computing d_min(p_l, S_d).
 5:    if d_min(p_l, S_d) = d_min(p_l, S_h) then
 6:       S_{a_b} = S_{a_b} ∪ {p_l}
 7:    else
 8:       if d_min(p_l, S_d) < d_min(p_l, S_h) then
 9:          S_{a_d} = S_{a_d} ∪ {p_l}
10:       else
11:          S_{a_h} = S_{a_h} ∪ {p_l}
12:       end if
13:    end if
14: end for
15: Return S_{a_b}, S_{a_d}, and S_{a_h}.
```

Fig. 4. A brute-force algorithm for separating type-l cells (pseudocode of Algorithm 1).

The idea of Algorithm 2 is also simple. We may not have to go through each grid point in S_d. If there exists a cell p_d such that $d_e(p_l, p_d) < d_{min}(p_l, S_h)$ then let p_l be in S_{a_d}, and break both this for-loop and the outer for-loop, and test the next cell after p_l in S_l. The pseudocode of Algorithm 2 is modified from the code of Algorithm 1 by inserting a few lines after Line 3 in Fig. 4. It is described in Fig. 5.

Algorithm 2 (An improved version of Algorithm 1)
Input and output are the same as for Algorithm 1 but for set cardinalities assume that $|S_h| \leq |S_d|$.
Pseudocode: See Fig. 5.

1: Lines 1–3 from Fig. 4.
2: **for** each $p_d \in S_d$ **do**
3: **if** $d_e(p_l, p_d) < d_{min}(p_l, S_h)$ **then**
4: $S_{a_d} = S_{a_d} \cup \{p_l\}$
5: Break both this for-loop and the outer for-loop.
6: **end if**
7: **end for**
8: Exactly copy Lines 4–15 from Fig. 4, but remove Line 7.

Fig. 5. An improved version of Algorithm 1 for separating type-l cells: Simply insert Lines 3–6 in the code of Algorithm 2 after Line 3 in the code of Algorithm 1. Note that the 'outer for-loop' refers to the outer loop as specified in Algorithm 1.

3.2 Algorithm in 2D

Algorithm 3, our main algorithm, is based on Algorithm 2 and Procedures 1 and 2. Procedure 1 is used to compute relevant adjacent cells within the adjacency set of a type-$sldoh$ supercell. The procedure is shown in Fig. 6. The word *relevant* means here that each returned supercell is both in the adjacency set of the given type-$sldoh$ supercell as well as of the corresponding supercell of the θ-graph, for grid constant $\theta \geq \theta_0$.

Procedure 1 (Compute relevant supercells in adjacency set of a type-$sldoh$ supercell)
Input: A θ-graph $G = [V, E]$, and a type-$sldoh$ supercell represented by vertex $v \in V$.
Output: Return a subset N_v of V such that $d_{min}(v', v) \leq \theta$, for each supercell $v' \in N_v$.
Pseudocode: See Fig. 6.

1: Let N_i be the sets of supercells of the first two adjacency sets of supercell v, where $i = 1, 2$.
2: Let $N_2' = \emptyset$.
3: **for** each supercell $u \in N_2$ **do**
4: **if** $d_{min}(u, v) \leq \theta$ **then**
5: $N_2' = N_2' \cup \{u\}$
6: **end if**
7: **end for**
8: Let $N = N_1 \cup N_2'$.
9: Let $N_v = \emptyset$.
10: **for** each supercell $u \in N$ **do**
11: **if** $u \in V$ **then**
12: $N_v = N_v \cup \{u\}$
13: **end if**
14: **end for**
15: Return N_v.

Fig. 6. Computation of relevant cells adjacent to a type-$sldoh$ supercell v (Procedure 1).

The following Procedure 2 is used to compute relevant supercells in the adjacency set of a type-sl supercell. The word *relevant* means here that the returned supercells are both in the adjacency set of the type-sl supercell and the supercells of the θ-graph. The pseudocode is shown in Fig. 7.

Procedure 2 (Compute relevant supercells in the adjacency set of a type-sl supercell)
Input: A θ-graph $G = [V, E]$, a type-sl supercell $v \in G$. Assume that there exist type-d or type-h cells.

1: $i = 1$
2: **while** there is not any type-d or type-h cell contained in a supercell in $N(v, 1, i)$ (i.e., the first adjacency set with distance i of the supercell v) **do**
3: $i = i + 1$
4: **end while**
5: Take any corner supercell u in $N(v, 1, i)$.
6: Let $R_v = d_{max}(u, v)$.
7: Compute $N(v, 3, R_v)$ (i.e., the third adjacency set of the supercell v with radius R_v).
8: Compute $N(v, 2, i)$ (i.e., the second adjacency set of the supercell v with radius i).
9: Let $N_4 = N(v, 3, R_v) \backslash N(v, 2, i)$.
10: Let $N_v = \emptyset$.
11: **for** each supercell $u \in N_4$ **do**
12: **if** $u \in V$ **then**
13: $N_v = N_v \cup \{u\}$
14: **end if**
15: **end for**
16: Return N_v.

Fig. 7. Computation of relevant supercells adjacent to a type-sl supercell v (Procedure 2).

Output: Return a subset N_v of V such that, for each supercell $v' \in N_v$, we have that $d_e(v', v) \leq R_v$, where R_v is the radius of $N(v, 3, R_v)$.
Pseudocode: See Fig. 7.

The imaged part of the liver is defined by all type$-l$ cells. Its veins consists of type-i cells, where $i = d, h$.

The main idea of the following main algorithm (Algorithm 3) is to decompose the liver into some supercells so as to reuse the improved version of the above brute-force algorithm (i.e., Algorithm 2) "locally" by removing unnecessary type-i cells (where $i = d, h$) which are "too far" from the current supercell (thus, also "too far" from any type-l cells contained in the current supercell).

Algorithm 3 (Main Algorithm)
Input: A set S containing type-i cells, where $i = d, h, l$, and a parameter $m > 0$ (for example, $m = 20$).
Output: Three sets S_{a_b}, S_{a_d}, and S_{a_h} such that $S_l = S_{a_b} \cup S_{a_d} \cup S_{a_h}$, where S_{a_b} is affected by both S_d and S_h, and S_{3_i} is affected by S_i only, for $i = d, h$.
Pseudocode: See Fig. 8.

Regarding a proof of the correctness of Algorithm 3, at first, the set of liver cells (i.e. of their centroids) can be assumed to be digitally convex (i.e. the Gauss digitization of a convex polyhedron). Thus we can define *affected* by using d_{min} as in Section 2, based on the Euclidean distance d_e.

For each supercell C, if C is of type-$sldoh$ supercell then, for any two original (i.e., before digitization in Line 3) cells p_1 and p_2 contained in C, for their distance we have that $d_e(p_1, p_2) < \sqrt{2} \cdot \theta_0$. Thus, we can only consider type-i cells inside of $N(C, 3, \sqrt{2} \cdot \theta)$ for separating type-l cells in C, for $i = d, h$. In short, the candidate sets are reduced from S_d and S_h to $S_d \cap N(C, 3, \sqrt{2} \cdot \theta)$ and $S_h \cap N(C, 3, \sqrt{2} \cdot \theta)$.

For each supercell C, if C is of type-sl then for any two original cells p_1 and p_2 contained in C and the corner supercell C' (See Fig. 9 for an illustration of C and

1: Let $\theta = m \times \theta_0$.
2: Let Ω be the smallest isothetic circumscribing rectangle that contains all cells of set S.
3: Digitize Ω using grid constant θ and label a supercell C to be "active" if C contains type-i cells, where $i = d, h, l$.
4: Construct the h-graph $G = [V, E]$ as follows: Let V be all "active" supercells. For any two supercells C_1 and C_2 in V, define an edge $e = \{C_1, C_2\}$ if C_1 and C_2 are 8-adjacent.
5: Let $S_{a_b} = S_{a_d} = S_{a_h} = \emptyset$.
6: **for** each supercell $v \in V$ **do**
7: **if** v is a type-$sldoh$ supercell **then**
8: Let G and v be the input for Procedure 1 for computing relevant adjacent supercells N_v for the type-$sldoh$ supercell v.
9: Let $S_d = S_h = \emptyset$.
10: Let S_l be the set of all cells in v.
11: **for** each supercell $u \in N_v \cup \{v\}$ **do**
12: **for** each cell $p \in u$ **do**
13: **if** p is type-d cell **then**
14: $S_d = S_d \cup \{p\}$
15: **else**
16: **if** p is type-h cell **then**
17: $S_h = S_h \cup \{p\}$
18: **end if**
19: **end if**
20: **end for**
21: **end for**
22: Let S_d, S_h and S_l be the input for Algorithm 2 for computing three sets $S_{a_b}(v)$, $S_{a_d}(v)$ and $S_{a_l}(v)$ such that $S_l = S_{a_b}(v) \cup S_{a_d}(v) \cup S_{a_l}(v)$, where $S_{a_b}(v)$ is affected by both S_d and S_h, and $S_{3_i}(v)$ is affected by S_i only, for $i = d, h$.
23: $S_{3_i} = S_{3_i} \cup S_{3_i}(v)$, where $i = d, h, l$.
24: **else**
25: **if** v is a type-sl supercell **then**
26: Let G and v be the input for Procedure 2 for computing relevant adjacent supercells N_v for the type-sl supercell v.
27: Exactly copy Lines 9–23 into here.
28: **end if**
29: **end if**
30: **end for**
31: Return the three sets S_{a_b}, S_{a_d}, and S_{a_l}.

Fig. 8. Pseudocode of the main algorithm: m is a parameter and can be adjusted depending on the size and distribution of the input data set S.

C'.), we have that $d_{max}(p_1, p_2) \leq R_C$. Note that the radius $R_C = d_{max}(C', C)$ is defined in Line 6 in Procedure 2. Thus, we can only consider type-i cells inside of N_C for separating the type-l cells in C, where $i = d, h$ (N_C is the subset returned by Procedure 2).

In short, the candidate sets are reduced from S_1 and S_2 to $S_1 \cap N(C, 3, \sqrt{2} \times \theta)$ and $S_2 \cap N(C, 3, \sqrt{2} \times \theta)$, respectively. See Fig. 9 for an illustration of R_C and N_C.

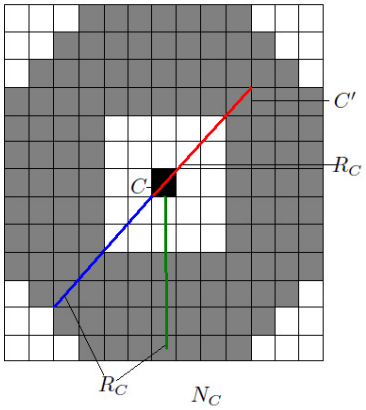

Fig. 9. Illustration for the correctness proof of Algorithm 3.

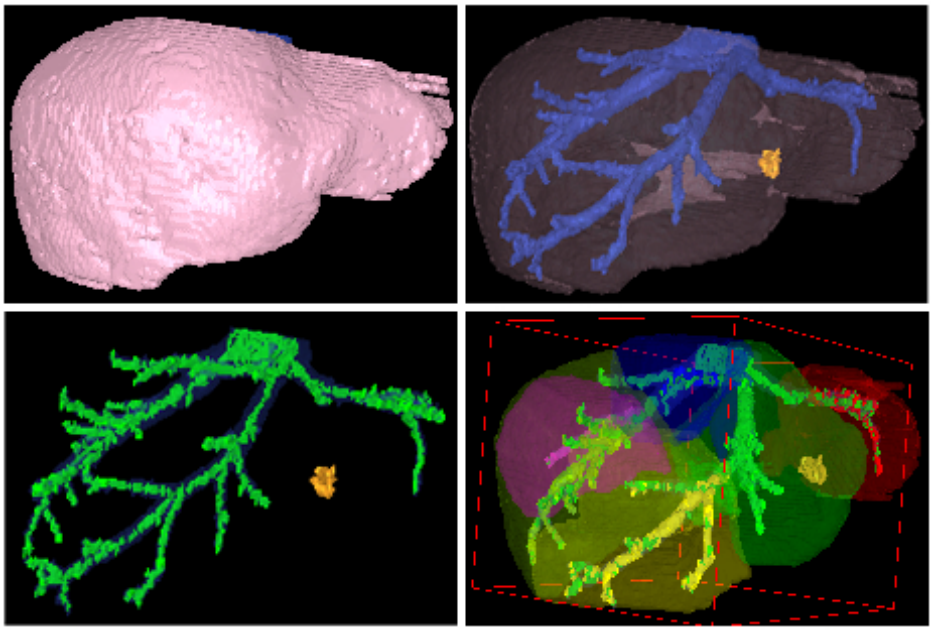

Fig. 10. *Left, top*: The liver model. *Right, top*: The vein component which has a tree shape topology, and the tumour. *Left, bottom*: The cells in the vein component and the tumour. *Right, bottom*: A liver usually consists of eight parts shown in eight colours. Only the red part on the right is the diseased part, as detected by our algorithm, and it should be removed. It seems there are some green cells between red cells. This may disappear if we change the angle of view.

3.3 Algorithm in 3D

The limited space does not allow a full description. However, the algorithms are very straightforward extensions of the 2D case: copy from Subsection 3.2 and replace $\sqrt{2}$ by $\sqrt{3}$. Figure 10 shows some experimental results of our main algorithm in 3D.

4 Time Complexity

Regarding the time complexity of Algorithms 1 and 2, and of Procedures 1 and 2, we have the following:

Lemma 1. *Algorithm 1 takes* $|S_3| \cdot (|S_1| + |S_2|)$ *operations.*

Lemma 2. *Algorithm 2 takes at most* $|S_3| \cdot (|S_1| + |S_2|)$ *operations.*

Lemma 3. *Procedure 1 takes* $\mathcal{O}(|N|)$ *operations, where* N *is defined as in Line 8 of Procedure 1.*

It is $N = N(v, 3, \sqrt{2} \times \theta)$ in the 2D case, and $N = N(v, 3, \sqrt{3} \times \theta)$ in the 3D case. For any integer parameter $m \geq 1$ it is $|N| = 20$ in the 2D case, and $|N| = 80$ in the 3D case.

Lemma 4. *Procedure 2 takes* $\mathcal{O}(|N(v, 3, R_v)|)$ *operations, where* R_v *is defined in Line 6 of Procedure 2.*

Regarding the time complexity of the main algorithm (Algorithm 3), the main computations occur in Lines 8, 22, and 26.

By Lemma 3, the computation in Line 8 takes $\mathcal{O}(n \times |N|)$ operations, where n is the number of supercells.

By Lemma 4, the computation in Line 26 takes $\mathcal{O}(n \times n_{max})$ operations, n_{max} is the maximal value of all $|N(v, 3, M_{R_v})|$'s, and R_v is defined in Line 6 in Procedure 2.

By Lemma 2, the computation in Line 22 takes $\mathcal{O}(c_i^2 \times (n \times |N| + n \times n_{max})$ operations, where c_i^2 is the maximum number of cells in a supercell, $i = 2$ for the 2D case, and $i = 3$ for the 3D case. By the definition of supercells, we have that $c_i^2 = m^i$, for $i = 2, 3$. Thus, the computation in Line 22 takes $\mathcal{O}(m^i \times n \times (|N| + n_{max}))$ operations, where i is 2 or 3 for the 2D or 3D case, respectively. Recall that $|N| = 20$ in the 2D case and $|N| = 80$ in the 3D case. Thus, we have the following

Theorem 1. *The runtime of Algorithm 3 is in* $\mathcal{O}(m^i \times n \times n_{max})$*, where* n_{max} *is the maximum value of all* $|N(v, 3, M_{R_v})|$*'s,* R_v *is defined in Line 6 in Procedure 2, and* $i = 2$ *for the 2D case, and* $i = 3$ *for the 3D case.*

Exact Euclidean Distance Transform takes $\mathcal{O}(m^i \times n^i)$ operations [2, 6]. Thus, the main algorithm (Algorithm 3) may be faster or slower than the exact Euclidean Distance Transform depending on the value of n_{max} which depends on the distribution of input type-i cells, for $i = d, h, l$.

5 Experimental Results

Our experiments used a liver model of 10^7 cells within a cuboid which is 324 pixels long, 243 pixel wide, and 129 pixels high. This kind of constant is typical for a current CT scan of a liver. Each voxel is not perfectly cubic, having side length 0.683 in two directions and 1.0 in the third. We used a PC with 2.50 GHz CPU and 3.0 Gb RAM.

m	Time	m	Time	m	Time	m	Time	m	Time
0	53	0	167	0	69	0	53	0	49
10	24	10	36	10	12	10	36	10	33
20	24	20	23	20	14	20	37	20	32
40	38	40	41	40	23	40	57	40	50

Table 1. The results of five experiments are shown from left to right, organized in columns. Parameter m is applied in Line 1 of Algorithm 3, and the time is in seconds for a 3D voxel data set of dimensions $324 \times 243 \times 129$.

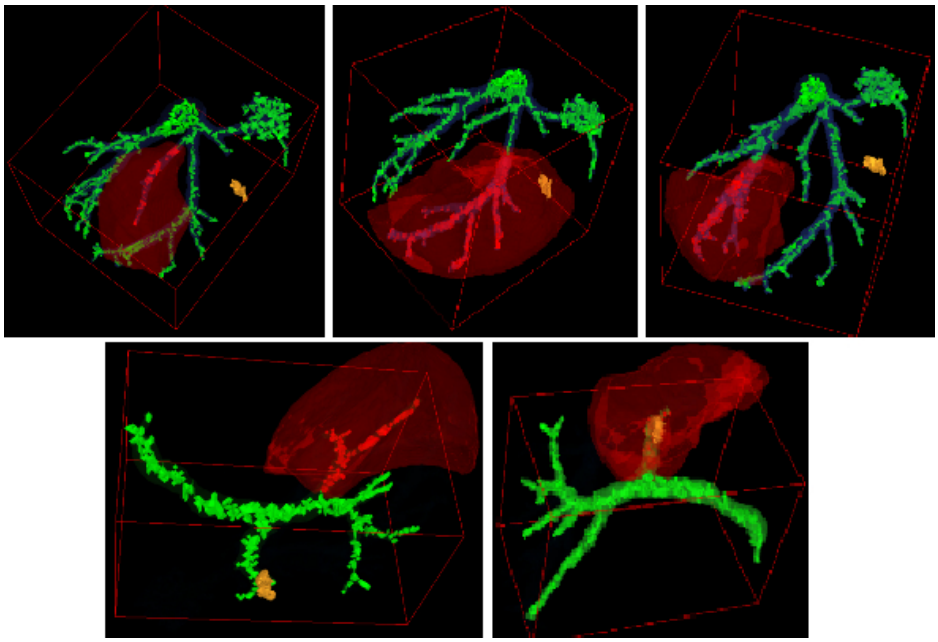

Fig. 11. The diseased liver volumes (i.e., sets V in Table 2) in the i-th experiment, from left to right and top to bottom. Note that the tumour was not included in set V in Experiments 1, 3 and 4. This is because the diseased vein S_2 in Table 2 is only simulated in the experiments.

i	V	S_2	S_1	i	V_1	S_2	S_1
1	87.491	105	2120	4	251.6717	182	1487
2	350.4517	525	1700	5	232.631	153	1516
3	242.179	331	1894				

Table 2. By i we denote the index of an experiment. V is the volume of the diseased liver, S_2 are the cells inside the volume of the diseased vein, and S_1 are the cells inside of the volume of the healthy vein.

Table 1 shows the relationship between parameter m as applied in Line 1 of Algorithm 3 and the running time. Times for $m = 0$ are the running times of the improved

brute-force algorithm (Algorithm 2). The experiments indicate that Algorithm 3 is better than Algorithm 2 for $m = 10$ and $= 20$. The algorithm appears to be inefficient if m is either to small or to large.

Table 2 shows diseased volumes in five experiments. See Fig. 11 for sets V as mentioned in the table.

6 Conclusions

We presented a simple and time-efficient algorithm for separating liver cells using basic ideas of digital geometry. In contrast to existing liver-surgery planning algorithms, our algorithm is not only independent of a surgeons' personal interactive manipulations, but also outputs the exact solution. The paper introduced an important existing problem to the digital geometry community.

Acknowledgements: The authors thank all three anonymous reviewers for very valuable comments which have been taken into account for the final paper.

References

1. Bosch, F. X., Ribes, J., Diaz, M., Cléries, R.: Primary liver cancer: worldwide incidence and trends. Gastroenterology, 127:S5–S16 (2004)
2. Cao, T.-T., Tang, K., Mohamed, A., Tan, T.-S.: Parallel banding algorithm to compute exact distance transform with the GPU. In: Symp. Interactive 3D Graphics, pp. 83–90 (2010)
3. Konrad-Verse, O., Preim, B., Littmann, A.: Virtual resection with a deformable cutting plane. In: Simulation und Visualisierung, pp. 203–214 (2004)
4. Klette, R., Rosenfeld, A.: Digital Geometry. Morgan Kaufmann, San Francisco (2004)
5. Lamata, P., Lamata, F., Sojar, V., Makowski, P., Massoptier, L., Casciaro, S., Ali, W., Stüdeli, T., Declerck, J., Jackov Elle, O., Edwin, B.: Use of the resection map system as guidance during hepatectomy. Surg. Endosc., 24:2327–2337 (2010)
6. Maurer, C. R., Qi, R., Raghavan, V.: A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. IEEE Trans. Pattern Analysis Machine Intelligence, 25:265–270 (2003)
7. Meinzer, H.-P., Thorn, M., Cordenas, C. E.: Computerized planning of liver surgery – an overview. Computers & Graphics, 26:569–576 (2002)
8. Park, H., Bland, P. H., Meyer, C. R.: Construction of an abdominal probabilistic atlas and its application in segmentation. IEEE Trans. Medical Imaging, 22:483–492 (2003)
9. Pianka, F., Baumhauer, M., Stein, D., Radeleff, B., Schmied, B. M., Meinzer, H.-P., Müller, S. A.: Liver tissue sparing resection using a novel planning tool. Langenbecks Arch Surg., 396:201–208 (2011)
10. Reitinger, B., Bornik, A., Beichel, R., Schmalstieg, D.: Liver surgery planning using virtual reality. IEEE Computer Graphics Applications, 26:36–47 (2006)
11. Rosenfeld, A., Pfaltz, J. L.: Distance functions on digital pictures. Pattern Recognition, 1:33–61 (1968)
12. Shevchenko, N., Seidl, B., Schwaiger, J., Markert, M., Lueth, T. C.: MiMed Liver: A planning system for liver surgery. In: Int. Conf. IEEE EMBS, pp. 1882–1885 (2010)
13. Yamanaka, J., Saito, S., Fujimoto, J.: Impact of preoperative planning using virtual segmental volumetry on liver resection for hepatocellular carcinoma. World J. Surg., 31:1249–1255 (2007)