Accelerating 'Intelligent Scissors' Using Slimmed Graphs

Kevin Chun-Ho Wong Pheng-Ann Heng Tien-Tsin Wong
Dept. of Computer Science & Engineering
The Chinese University of Hong Kong

Abstract

In this paper, we describe an acceleration technique for the semi-automatic image segmentation algorithm, intelligent scissors. Using intelligent scissors, user can accurately and interactively extract the object from the digitized image. However, the original algorithm suffers from slow performance when large images are treated. In practice, pixels within the non-edge regions are seldom involved in the determination of boundaries (segmentation curves). If these pixels are removed before boundary determination, the performance of intelligent scissors can be sped up. We generate a slimmed graph to achieve such goal. Significant improvement in response time is resulted using the slimmed graph.

1 Introduction

Accurate segmentation of an interested object from a digitized image has long been an open problem in the field of computer graphics, computer vision and user interface. Many previous work concentrate on designing fully automatic algorithms. However, we believe fully automation is not possible if no knowledge of the world is modeled in the algorithm. Analyzing pixel values alone is not enough to accurately determine those segmentation curves (boundaries). On the other hand, semi-automatic algorithm allows the user to enter his/her knowledge of the world and leave the algorithm to automatically refine the boundary. Mortensen and Barrett [4] introduced an semi-automatic tool known as intelligent scissors to assist the segmentation of desired object. Unfortunately, the computational cost of the original algorithm is highly dependent on image resolution. Hence large image is slow to process. In this paper, we describe a technique to accelerate the intelligent scissor algorithm by reducing the number of pixels to search during the segmentation.

During the image segmentation, the user is required to specify the starting point (seed point) and ending point (goal point) of the boundary curve. After that, the boundary curve between these two points is obtained by searching the "optimal" path. The searching is done using an optimal path searching algorithm similar to Dijkstra's algorithm [1]. The optimality is defined by an objective function. To find the path, the image is first modeled as a graph. Each pixel is mapped to a node connecting with its eight neighboring pixels (nodes) by edges (Figure 1). A cost value is assigned to each edge. Hence, an $W \times H$ image is represented by a graph of $W \times H$ nodes and $4WH - 3(W+H) + 2$ edges. A dynamic programming technique of time complexity $O(n)$ is used to search the path, where n is the total number of nodes in the graph. Therefore, the performance of intelligent scissors is highly dependent on the size of the graph. By observation, most nodes within the non-edge regions are seldom involved in the path searching. If these nodes are pruned before path searching, the performance can be significantly improved. We call the pruned graph the *slimmed graph*. For a real-time application, the speed is critical. Fast segmentation tool provides users an interactive control and real-time response on the screen.

In this paper, we discuss how to speed up the intelligent scissors by generating the slimmed graph. Section 2 briefly illustrates the original intelligent scissors algorithm. Section 3 discusses how we generate the slimmed graph to reduce the computation. Results of using our fast intelligent scissor is also shown. Conclusions and future directions are drawn in Section 4.

2 Intelligent Scissors

When using intelligent scissors, the user is asked to specify two pixels in the image, namely the seed point and the goal point. Then the algorithm will try to find the *cost-minimized path* from the seed point to the goal point. To find the path, the image is first modeled as a graph. Each pixel is replaced by a node in the graph. Every node connects to its eight neighbors by edges (Figure 1). Each edge is associated with a *cost*. The cost value is determined by a cost function. A path searching algorithm similar to Dijkstra's algorithm [1] is then used to search for the optimal path. Algorithm 1 shows the steps of this 2D dynamic programming searching algorithm. Since the main focus of this paper is not on the searching algorithm, readers are referred to Mortensen and Barrett's paper [4] for detail description of the algorithm. We shall not repeat here. The right diagram in Figure 1 shows an example boundary (optimal path) determined by intelligent scissor.

The cost function is usually defined as a function of local image features including image gradient and Laplacian zero-crossing. Mortensen and Barrett [4]

Notation

s is the seed point.
L is the list of active nodes.
$N(q)$ is the neighborhood of node q.
$e(q)$ indicates whether node q is marked/processed.
$T(q)$ returns the total cost from s to q.
$\text{cost}(p,q)$ returns the local cost from p to q.
$\min(L)$ returns the node with minimum cost within the list L.
$B(q)$ is the back pointer of node q.

Algorithm *Boundary-Searching*

$T(s) = 0$
Add s to L
While ($L \neq$ *empty*)
  $q = \min(L)$
  $e(q) = \text{TRUE}$
  For each $r \in N(q)$ s.t. not $e(r)$ **do**
    If $r \in L$ and $T(q) + \text{cost}(q,r) < T(r)$ **then**
      Remove r from L
    If r not $\in L$ **then**
      $T(r) = T(q) + \text{cost}(q,r)$
      $B(r) = q$
      Add r into L

Algorithm 1: 2D dynamic programming algorithm for boundary searching, proposed by Mortensen and Barrett.

defined the cost function between two neighbor pixels p and q as,

$$\text{cost}(p,q) = \omega_Z f_Z(q) + \omega_D f_D(p,q) + \omega_G f_G(q) \tag{1}$$

where ω_Z, ω_D and ω_G are user-defined weights,
f_Z is the Laplacian zero crossing,
f_D is the gradient direction,
f_G is the gradient magnitude,

Stalling and Hege [5] defined the cost function as,

$$\text{cost}(p,q) = \max(f_G(p), f_G(q)) - \frac{1}{2}(f_G(p) + f_G(q)) \tag{2}$$

It is not surprise that the cost function defined is tightly related to the local image gradient properties because we want to identify the edge region (region contains boundaries). In other words, there is less interest in searching in the non-edge region. Most of the nodes (pixels) in the non-edge region are seldom included in the final path. However, the original intelligent scissors algorithm treats every pixel and every region in the image equally. Hence, these nodes (pixels) still consume

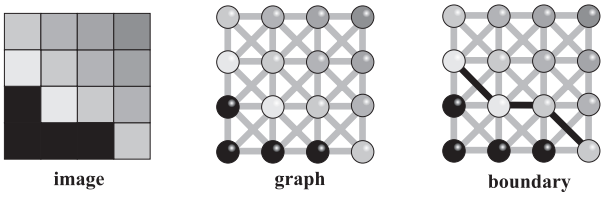


Figure 1: The image is represented as a graph during boundary determination.

computation and memory. If we can reduce this kind of nodes before searching, the algorithm can be sped up and less memory is consumed.

3 A Fast Image Segmentation Tool

The construction of the slimmed graph is a preprocessing step. In the slimmed graph, a node may represent a region (of pixels), instead of a single pixel. Larger regions are formed if the gradient is low. On the other hand, smaller regions are formed if the gradient is high. The problem is how to subdivide the image into regions (blocks of various sizes). This can be done through binary space partitioning (BSP) [3]. Obviously the gradient information is a good criterion to guide the image subdivision. During this subdivision process, split lines are used to segment the image into rectilinear blocks. Smaller blocks are generated in the region with higher gradient while larger blocks are used in the region of lower gradient. Another problem in generating the slimmed graph is how to connect neighboring nodes (which now represent regions instead of pixels) and how to assign cost value to edges.

3.1 Subdivision Using BSP

The first step to generate the slimmed graph is to segment the image into blocks of various sizes according to the total sum of normalized gradient magnitude of pixels in the block. A user-controlled threshold is used to limit the total sum of gradient magnitude in each block. One scheme to subdivide the image is quadtree. However, quadtree is not flexible enough to generate rectangular blocks and it may introduce unnecessary fragmentation. Instead we use BSP-tree which allows the generation of rectangular blocks and the number of children sub-blocks needs not be four.

Figure 2 illustrates the BSP-tree subdivision scheme graphically. The subdivision starts with an image of normalized gradient magnitudes. This image is

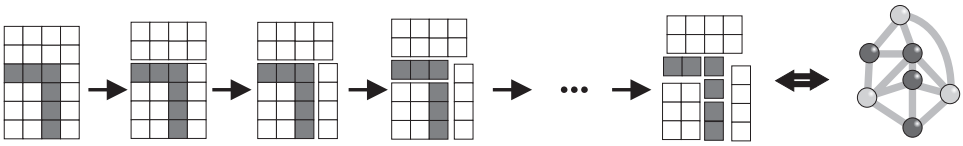


Figure 2: Segmenting the image using BSP-tree.

obtained from the original image by calculating the normalized magnitude $G(i,j)$ of 2D gradient vector at each pixel (i,j),

$$G(i,j) = 1 - \frac{\eta}{\max(\eta)}$$

$$\eta = \sqrt{(\frac{dI}{dx})^2 + (\frac{dI}{dy})^2}$$

where $\frac{dI}{dx}$ and $\frac{dI}{dy}$ are the partial derivatives in x and y directions respectively. Function $\max(\eta)$ returns the maximum gradient magnitude over the whole image. Figure 5(b) shows one such map of the original image in Figure 5(a).

The image is subdivided into two blocks if the total sum of normalized gradient magnitude exceeds a user-controlled threshold. For simplicity, we call the total sum of normalized gradient magnitudes the *gradient sum*. Each block will be recursively subdivided until the gradient sum is below the threshold. In each subdivision, a block is divided into two sub-blocks by either a horizontal or vertical split line. Vertical split line will be used if the width is longer than the height of block. Otherwise, horizontal line will be used. The two sub-blocks need not be equal in size. Once the subdivision is done, the image can be transformed into a slimmed graph by replacing each block with a node and connecting neighboring nodes by edges (Figure 2).

3.1.1 Placement of Split Lines

To decide where to place the split lines, we first calculate the gradient sum of the horizontal and vertical scanlines. A split line is placed at the scanline with the maximum difference of gradient sum across all scanlines in the block. Let the size of block be $N \times M$, where N and M are the width and height respectively. If $N > M$, we calculate gradient sum $S_y(i_0)$ for each vertical scanline i_0 in the block and place vertical split line. On the other hand, if $M \geq N$, we calculate gradient sum $S_x(j_0)$ for each horizontal scanline j_0 and place horizontal split line.

$$S_y(i_0) = \sum_{j=0}^{M-1} G(i_0, j)$$

$$S_x(j_0) = \sum_{i=0}^{N-1} G(i, j_0)$$

Next, we compute the finite difference of S_x and S_y,

$$\begin{aligned} \Delta S_x(j) &= S_x(j+1) - S_x(j) \\ \Delta S_y(i) &= S_y(i+1) - S_y(i) \end{aligned}$$

If vertical split line is needed, the split line is positioned between the vertical scanlines with index i_m and $i_m + 1$ such that $\forall$ vertical scanline k in the block, $|\Delta S_y(i_m)| \geq |\Delta S_y(k)|$. That is the split line is placed at a position with the largest difference of gradient sum. Similarly, in the case of horizontal splitting, the split line is placed in between the horizontal scanlines with index j_m and $j_m + 1$ such that $\forall$ horizontal scanline k in the block, $|\Delta S_x(j_m)| \geq |\Delta S_x(k)|$.

3.1.2 Acceleration with Summed Area Table

Naively computing the gradient sums and the finite differences is expensive. A faster computation of $\Delta S_x(j)$ and $\Delta S_y(i)$ can be achieved by using the summed area table [2]. Let $A(x,y)$ be the precomputed summed area function where

$$A(x,y) = \begin{cases} \sum_{i=0}^{x} \sum_{j=0}^{y} G(i,j) & 0 \leq x < W, 0 \leq y < H \\ 0 & \text{otherwise} \end{cases}$$

where $W \times H$ is the resolution of the whole image.

Suppose the top-left and bottom-right corners of a block are (x_l, y_t) and (x_r, y_b). The gradient sums of the horizontal scanline j and the vertical scanline i in the block are simply,

$$\begin{aligned} S_x(j) &= A(x_r, j) + A(x_l - 1, j - 1) - A(x_l - 1, j) - A(x_r, j - 1) \\ S_y(i) &= A(i, y_t) + A(i - 1, y_b - 1) - A(i - 1, y_t) - A(i, y_b - 1) \end{aligned}$$

respectively. And the finite differences of the horizontal and vertical gradient sums are

$$\begin{aligned} \Delta S_x(j) = {} & A(x_r, j+1) + 2A(x_l - 1, j) + A(x_r, j - 1) \\ & - A(x_l - 1, j+1) - 2A(x_r, j) - A(x_l - 1, j - 1) \\ \Delta S_y(i) = {} & A(i+1, y_t) + 2A(i, y_b - 1) + A(i - 1, y_t) \\ & - A(i+1, y_b - 1) - 2A(i, y_t) - A(i - 1, y_b - 1) \end{aligned}$$

As all values of $A(x,y)$ are precomputed and stored in a 2D array, the finite difference $\Delta S_x(j)$ and $\Delta S_y(i)$ can be computed in constant time.

3.2 Slimmed Graph Generation

Algorithm 2 shows our pseudocode that generates the slimmed graph. In the algorithm, the image is recursively subdivided into blocks until (1) the gradient sum of block is smaller than λ or (2) the area of block is smaller than κ. Both λ and κ are user-controlled parameters. Parameter λ limits the gradient sum of a block while parameter κ controls the minimal size of a block. When the values of both parameters increase, a slimmer graph is obtained. At the same time, the accuracy of the resultant boundaries reduces.

Notation
- G is the graph.
- T is a list of blocks to be subdivided.
- $N_r(n)$ is a list of neighboring nodes of node n.
- $B(n)$ is the corresponding block of node n.
- $n(B)$ is the corresponding node of block B.
- B_0 is the initial block representing the whole image.

Algorithm *Slimmed-Graph-Generation*

```
T = {B_0}
G = {n(B_0)}
While (T ≠ empty)
  Pop b from T
  If size of b ≥ κ and gradient sum of b ≥ λ then
    Split block b into b_1, b_2
    Add b_1, b_2 to T
    Remove n(b) from graph G
    Add n(b_1), n(b_2) to graph G
    Connect n(b_1), n(b_2) with an edge
    For each neighbor node p in N_r(b)
      If B(p) and b_1 are neighbors then
        Connect p, n(b_1) with an edge
      If B(p) and b_2 are neighbors then
        Connect p, n(b_2) with an edge
```

Algorithm 2: Generation of slimmed graph.

Once the image is subdivided into blocks, it can be transformed to a graph. Each block is represented by a node. Neighboring blocks are connected by edges. Since the node no longer represents a single pixel, but a block, it may connect to variable number of neighboring nodes. Any two blocks that touch each other are considered as neighbors. Figure 3 illustrates two cases of neighborhood. Let $(x_{i,l}, y_{i,t})$ and $(x_{i,r}, y_{i,b})$ be the corner positions of a block i. Whether or not block i and block j are neighbors is determined by the following criteria:

$$x_{\min} = \min(x_{i,l}, x_{j,l}) \qquad x_{\max} = \max(x_{i,r}, x_{j,r})$$

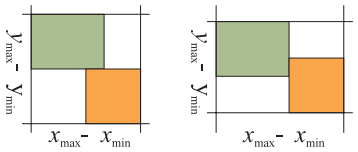


Figure 3: Two cases of neighborhood.

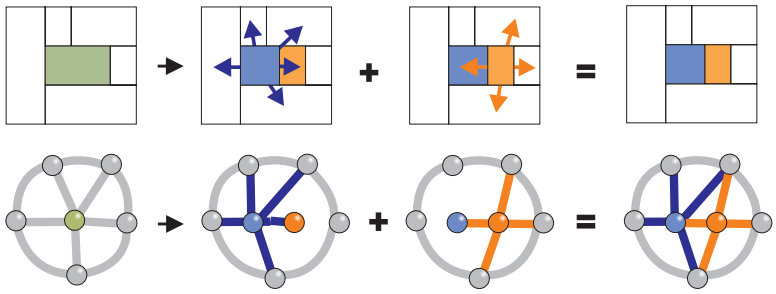


Figure 4: An example of connecting neighboring nodes while subdividing a block.

$$y_{\min} = \min(y_{i,t}, y_{j,t}) \qquad y_{\max} = \max(y_{i,b}, y_{j,b})$$

$$\begin{array}{ll} \text{If} & x_{\max} - x_{\min} \leq (x_{i,r} - x_{i,l}) + (x_{j,r} - x_{j,l}) \\ \text{and} & y_{\max} - y_{\min} \leq (y_{i,b} - y_{i,t}) + (y_{j,b} - y_{j,t}) \\ \text{then} & \text{block } i \text{ and } j \text{ are neighbors.} \end{array}$$

Instead of connecting the nodes after the image is completely subdivided, the graph is constructed gradually while subdividing the image. New edges are added to connect the newly created node to its neighboring blocks. Figure 4 illustrates how a node is split into two connected nodes during the image subdivision. It also demonstrates how neighboring nodes are connected by new edges. The detail steps are described in Algorithm 2.

3.3 Cost Function

Once the slimmed graph is generated, it will be used to determine the boundary curves during the run time. Note that the slimmed graph generation is done only once and in the preprocessing phase. During the run time, the user is asked to select a seed point s, the block enclosing this point $B(s)$ is then located. The corresponding node $n(B(s))$ will be used as the starting node. After the user selects the goal point g, the optimal path between the nodes $n(B(s))$ and $n(B(g))$ would be calculated by 2D dynamic programming algorithm (Algorithm 1) as in the original intelligent scissor algorithm.

		Slimmed Graph Thresholds		The graph in Original I.S.		Slimmed Graph in Fast I.S.		% of size reduced	
Figure	Size(pixels)	κ	λ	# of node	# of edge	# of node	# of edge	node	edge
5	100×100	0.08	5	10000	39402	1043	3619	89.57	90.82
6	320×240	0.08	5	76800	305522	8134	27391	89.41	91.03
7	512×512	0.08	7	262144	1045506	17825	59848	93.20	94.28
8	416×600	0.12	7	249600	995394	18370	60514	92.64	93.92

Table 1: This table shows the reduction in size of the slimmed graph in various types of images.

Since the cost functions (Equations 1 & 2) in the original intelligent scissor algorithm is designed for traversing between two pixels, they are no longer applicable to our case which each node represents a region. To determine the cost between two nodes, we first calculate the center of mass and the average gradient magnitude for each block. The position of the center of mass c_k of a block k is determined by,

$$c_k = \frac{\sum_{i=x_l}^{x_r-1}\sum_{j=y_b}^{y_t-1} G(i,j)\,p(i,j)}{\sum_{i=x_l}^{x_r-1}\sum_{j=y_b}^{y_t-1} G(i,j)} \tag{3}$$

where (x_l, y_t) and (x_r, y_b) are the top-left and right-bottom corners of block k,

$p(i,j) = \begin{pmatrix} i \\ j \end{pmatrix}$ is the 2D position of the pixel (i,j).

We also calculate the average gradient magnitude in the block k using,

$$g_k = \frac{\sum_{i=x_l}^{x_r-1}\sum_{j=y_b}^{y_t-1} G(i,j)}{(x_r - x_l)(y_t - y_b)} \tag{4}$$

Since the slimmed graph is no longer in grid structure, the distance between two nodes should affect the cost. Hence, the cost of the edge between two neighboring blocks is defined as a function of distance between the two centers of mass and their average gradient magnitudes.

$$\text{cost}(n_1, n_2) = \omega_D |c_1 - c_2| + \omega_F (1 - g_1)(1 - g_2) \tag{5}$$

where ω_D and ω_F controls the smoothness and the fitness of boundaries respectively. Note that both weights are positive.

3.4 Results

We have implemented the described algorithm on SGI Indigo2 with CPU MIPS R4400 250MHz. Different types of images including noisy image captured from

low-cost video camera, medical image and other real-world images are tested. Table 1 summarizes the size of slimmed graph generated in each test case. The statistics in the table indicate that our slimmed graph generation algorithm can significantly reduce the size of graphs. The reduction in the searching time is directly proportional to the reduction in the total number of nodes shown in the table.

Figure 5 shows the slimmed graph and subdivided blocks of an image of a Chinese character. Figures 6 to 8 show the boundaries found by our fast intelligent scissors. Figure 6 shows a noisy image captured from a low-cost video camera. Note that our algorithm can effectively extract the boundary of the furry toy. Figure 7 shows the result of segmenting the essential features in the CT image using our algorithm. Figure 8 shows how the four wooden rods can be conveniently extracted from the image even though the background contains subtle details such as leaves and stones.

4 Discussion

The key of the described technique is the graph slimming algorithm. The algorithm offers two parameters, λ (maximum gradient sum) and κ (minimum block size), to control the total number of nodes and edges to be generated. Increasing either one of these two parameters will generate a slimmer graph. Hence, the user can trade off the accuracy of boundaries with the interactiveness of program.

No matter how intelligent the algorithm is, the user may still not satisfy with the generated path. In this case, the desired path may have to be constructed by a series of shorter segments. The user can anchor the goal point of each segment by clicking on it. The goal point of a segment will then be the seed point of the following segment.

Gradient magnitude may not be the best criterion for guiding the BSP-tree subdivision. In the future, we will use more sophisticated criterion to guide the subdivision process. This will improve the accuracy of the boundaries being tracked even though the total number of nodes is restricted to be small.

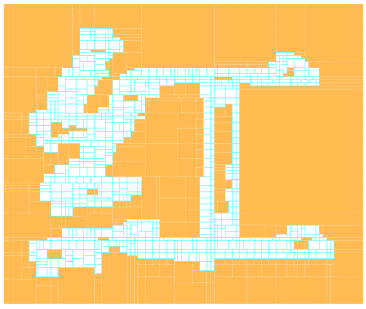
References

[1] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1989.

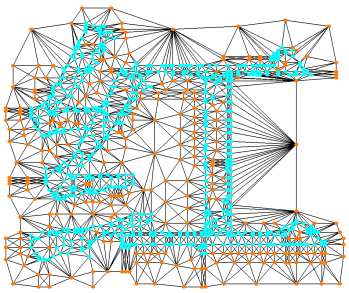
[2] F. C. Crow. Summed-area tables for texture mapping. In *Computer Graphics (Proceedings of SIGGRAPH'84)*, volume 18, pages 207–212, 1984.

(a) A 100 $\times$ 100 image of a Chinese character 'Red'

(b) The image of normalized gradient magnitude



(c) The blocks generated by BSP-tree subdivision



(d) The slimmed graph

Figure 5: The slimmed graph generation process of a Chinese character image.

[3] H. Fuchs. On visible surface generation by a priori tree structures. In *Computer Graphics (Proceedings of SIGGRAPH'80)*, volume 14, pages 124–133, 1980.

[4] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 191–198. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[5] D. Stalling and H.-C. Hege. Intelligent scissors for medical image segmentation. In B. Arnolds, H. Müller, D. Saupe, and T. Tolxdorff, editors, *Proceedings of 4th Freiburger Workshop* Digitale Bildverarbeitung in der Medizin, *Freiburg*, pages 32–36, March 1996.

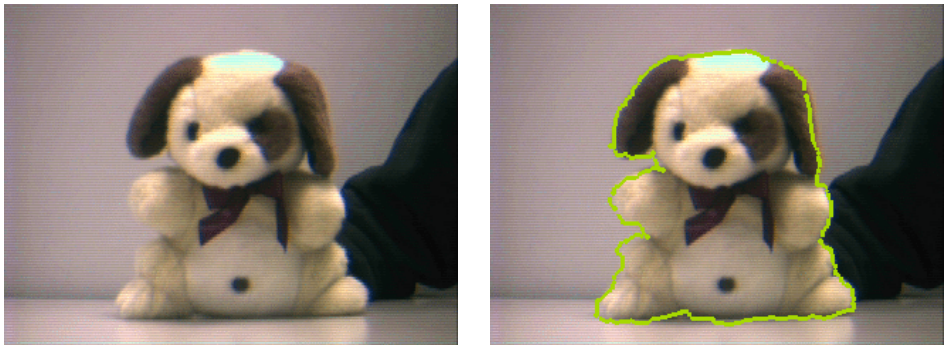


Figure 6: (a) A noisy image captured from the video camera. (b) Result of segmentation.

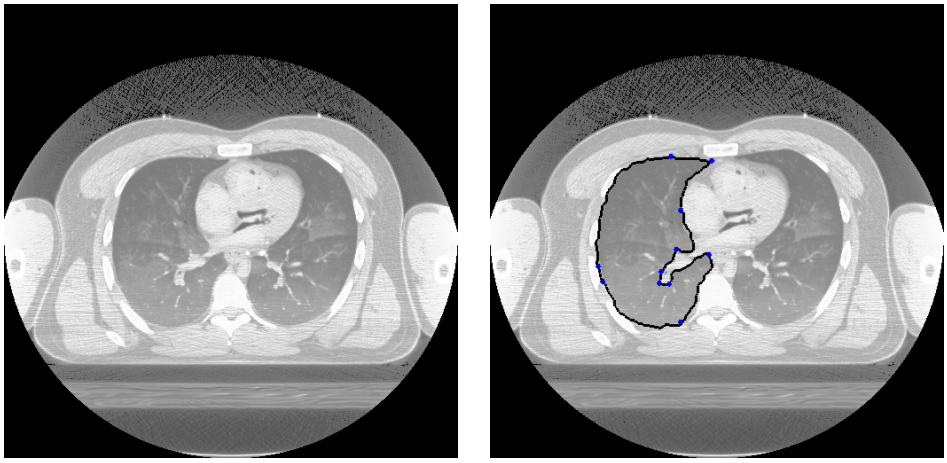


Figure 7: (a) The left figure is a 512×512 CT image which shows the cross-section of a male chest. (b) The left lung has been segmented out by our fast intelligent scissors. 11 seed points are used to outline the boundary.

Figure 8: (a) An image with many subtle details. (b) Four wooden rods are segmented out.