Source code documentation

for

Computer vision based recognition of materials boundaries and fill level of solids and liquids transparent vessels.

Use curvature of the vessel surface to adjust for reflections and functional parts and improve recognition accuracy

Based on the paper

"Tracing the boundaries of materials in transparent vessels using computer vision"

The source code is based on the method described in the paper: "*Tracing the boundaries of materials in transparent vessels using computer vision*" Freely available at Arxiv:

http://arxiv.org/ftp/arxiv/papers/1501/1501.04691.pdf

Given an image of a transparent vessel (**Icolor**, Figure 1) containing some liquid or solid material and the boundary of the vessel in the image (**Iborder**, Figure 1) the program finds the boundary of the material inside the vessel (Figure 1).

Finding the boundary of the transparent vessel (**Iborder**, Figure 1) in the image need to be done beforehand. This could be done using freely available code given in:

http://www.mathworks.com/matlabcentral/fileexchange/46887-find-boundary-of-symmetric-object-in-image
http://www.mathworks.com/matlabcentral/fileexchange/46907-find-object-boundaries-in-image-using-template--variable-image-to-template-size-ratio-

The boundary can also be traced manually, but it must be a binary image of line in thickness of 1 pixel and closed contour.

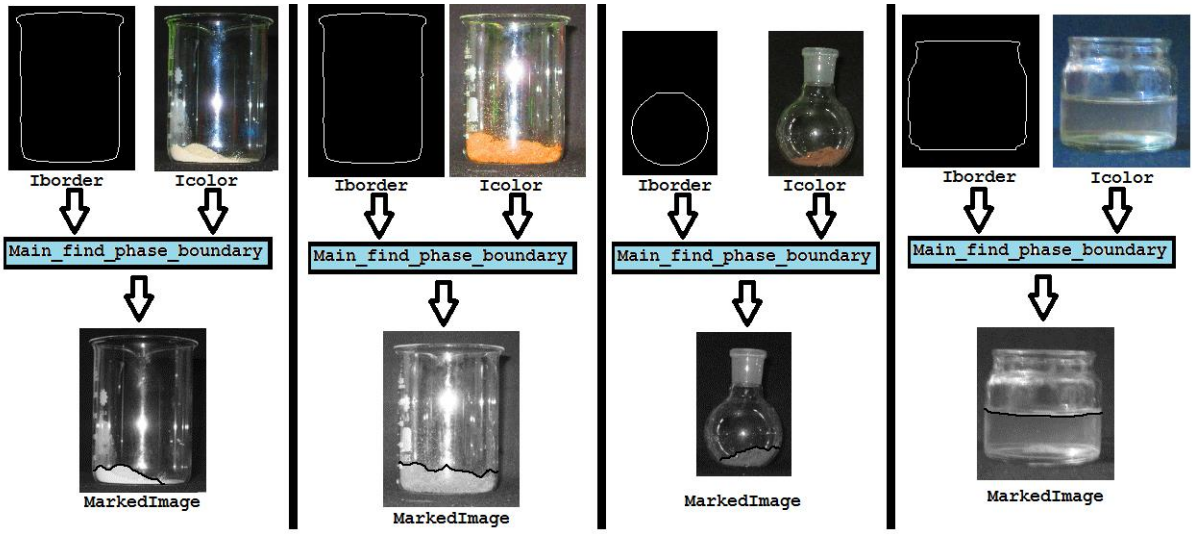

Figure 1. Image analysis method for finding the boundaries of liquid and solid materials in transparent containers.

Document arrangement

Section 1: Give basic instruction how to use and run the program.

Section 2: Give a list of adjustable parameters that can be changed in order to tune the recognition process for specific cases and specific materials.

Section 3: Give a description of the code, including a description of each function and major parameters in the software. In addition the source code include many remarks and descriptions for each parameter and function.

Contents

1. Instructions how to use code

The program is run using the function:

*[**MarkedImage**,**BinaryMaterialBoundaries**]=Main_find_phase_boundary(**Icolor**,**Iborder**);*

Which receive the image of the vessel containing the material (***Icolor***, Figure 1).

And binary image containing the boundary of the vessel in the image marked 1 (***Iborder***, Figure 1).

And return the boundary of the material in the image marked on the image (***MarkedImage***, Figure 1) and as a binary edge image (***BinaryMaterialBoundaries***)

1.1. Input:

***Icolor*:** color image of a transparent vessel containing the material (Figure 1).

Iborder*:** binary edge image in the size of ***Icolor with the boundaries vessel in the image ***Icolor*** marked 1 (all other pixels are 0, Figure 1).The template in ***Iborder*** must be a curve of close contour and thickness of 1 pixel (Figure 1). Finding the vessel boundary (***Iborder***) could be done automatically using the code available at freely available at the addresses below :

http://www.mathworks.com/matlabcentral/fileexchange/46887-find-boundary-of-symmetric-object-in-image
http://www.mathworks.com/matlabcentral/fileexchange/46907-find-object-boundaries-in-image-using-template--variable-image-to-template-size-ratio-

1.2. OUTPUT

***MarkedImage**:* The image of the vessel with the material boundaries marked in black on the image (Figure 1).
BinaryMaterialBoundaries Binary edge image in size the input image with the material boundaries marked on it in 1 and all other pixels are zero.

In addition the above two images are displayed on the screen.

1.3. Running

In Matlab command line Run the line:

[MarkedImage,BinaryMaterialBoundaries]=Main_find_phase_boundary(Icolor,Iborder);

The input and output parameters defined above.
The results will be displayed on the screen and in output parameters.

1.4. General remarks:

a) The above code will run with no additional parameters. However, depending on the type of material and container used there are a considerable number of parameters that can be adjust to improve accuracy and speed (see section 2).

b) The Image of the vessel (***Icolor***) doesn't have be a color image. For using grayscale image instead, simply replace the line.
I=double(rgb2gray(Icl));
With:
I=double(Icl);
Within the function: ***Main_find_phase_boundary***

c) If the input images ***Icolo***r and ***Iborder*** are not of the same size then ***Icolor*** will be resized to fit the size of ***Iborder***.

1.5. Example images

Example images available at the **EXAMPLES** folder could be used by running the script **run_test**.

2. Adjustable parameters:

The parameters described in this section all have default values which mean that their definition is optional. However, Adjusting the parameters according to type of vessel and materials used can considerably improve the accuracy and speed of the program . All parameters are defined in the first part of the function: ***Main_find_phase_boundary.*** Discussion on in parameter is given in sections 2-4 of the paper: "*Tracing the boundaries of materials in transparent vessels using computer vision*" Freely available at Arxiv:

http://arxiv.org/ftp/arxiv/papers/1501/1501.04691.pdf

2.1. Parameters controlling material boundary curve slope and angle

VerticalMoves=3

The Maximal number of sequential vertical steps that could be made along the material boundary curve (Figure 2). In other words the maximal length in pixels of the vertical sections along the boundary curve (Figure 2). See section 4.1-4.1.1 of the paper for more details.

Kvert=1.2

The extra cost of vertical move (move from (x,y) to (x,y+/-1)) used to discourage the vertical propagation (Figure 2). The cost of all vertical moves will be multiplied by ***Kvert.***

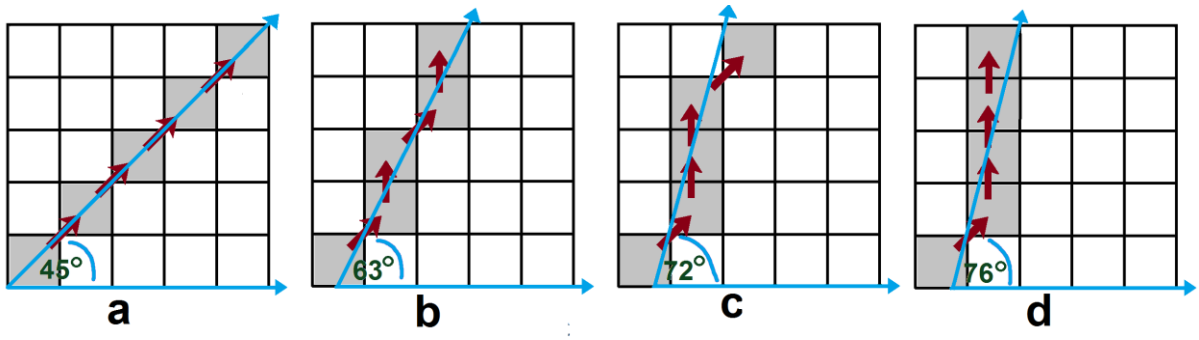

Figure 2. Limiting the path local slope by limiting sequential number of vertical steps to *VerticalMoves*. a) Forbidding vertical moves (*VerticalMoves*=0) limit the curve angle to 45˚. b) Limiting the number of sequential vertical moves to one (*VerticalMoves*=1) limits the curve slope to 63˚. c) Limits of two sequential vertical moves (*VerticalMoves*=2) limit the path angle to 72˚. d) Limit of three sequential vertical moves in arrow limit the path slope angle to 76˚.

MaxAngAB=50

The maximal angle of the line between the start point and end point of path(material boundary). Hence, boundary can exist between two points ***E, S*** on the vessel contour only if the slop of the line that form ***SE*** have an absolute angle smaller than ***MaxABAng*** (figure 3). For liquid materials this should be set around 0 degrees. For solids 50 degrees is a good value, but depend on the type of material used. See section 3.1 of the paper for more details.

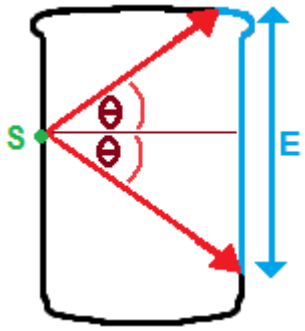

Figure 3: To limit slope of the material boundary the angle of the vector between boundary curve starting point (S) and end point (E) most not exceed some threshold angle *MaxAngAB=θ*. For a given starting point (S) only ending at points on the range E will be scanned.

MaxPathAng=65

Maximal angle in degrees between endpoints (***S,E,*** Figure 4b,c) of the path (boundary curve) and any point in the path (Figure 4b,c). Hence, any point (***C***, Figure 4a,b) along the phase boundary curve and the curve endpoints must have an angle smaller than ***MaxPathAng***. Hence, any pixel (***C***) for which the vector to the path start point (***SC***, Figure 4b) exceed specific angle (***MaxPathAng=φ***) is removed from the group of legal pixels in which the path can pass (Figure 4b,c). The path has two endpoints and as a result the constraint was applied twice, once for each endpoint (Figure 4b-c). For liquids materials, angle of 30-20˚ degrees is good while for solid 65˚ degree is usually good. See section 4.2.1 in the paper.

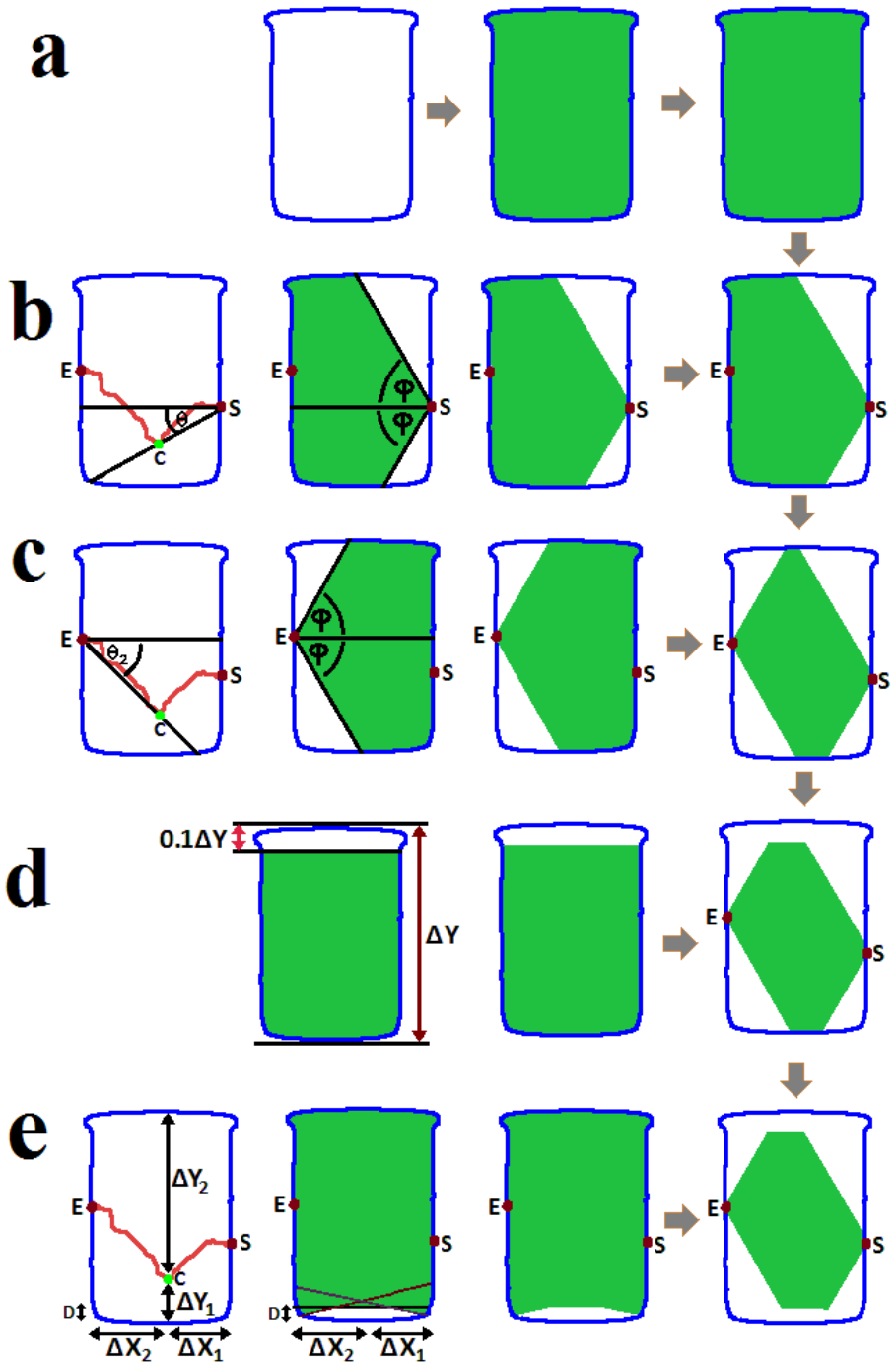

Figure 4. Legal image regions for path (material boundary) between point S and point E. Legal region marked green. Vessel boundary marked blue. a) Material boundary must be within the vessel region of the image. b,c) Slope filter: The angle (θ) of the vector between any point on the path (C) and the path starting/ending points (E, S) must not exceed some threshold $MaxPathAng=\varphi$. d) Top/bottom filter: The path is not allowed to propagate in the top $TopPathLimit$ % of the vessel. e) Flat path filter: The vertical distances ($\Delta Y_1, \Delta Y_2$) between every point on the path (C) and the vessel boundary most exceed some fraction ($MinDy = D\%$) of the minimal horizontal distances ($\Delta X_1, \Delta X_2$) between the point (C) and the path edge points (E,S) or some minimal constant distance D.

Hence: $\Delta Y_{1/2} > min(\Delta X_1 \cdot MinDy, \Delta X_2 \cdot MinDy, MinDyD(D))$.

2.2. Parameters that limit the material boundary curve in specific regions of the vessel.

TopPathLimit=0.0;

Ignore the ***TopPathLimit***% percent of the vessel when scanning for the material boundary in the image (Figure 4d, 5). Hence, any pixel for which the vertical distance from the top pixel of the vessel region in the image is smaller than ***TopPathLimit*** % percent of the vessel height will be ignored (The phase boundary will not be allowed to pass in this pixel).
The top of the vessel contained corks funnel or other covers which tend to give false positives which mean that in many cases should be ignored (Figure 5). To apply this every pixel in the image for which the vertical distance from the vessel top or bottom is smaller than ***TopPathLimit***% percent of the vessel height in pixels (Figure 4d) is removed from the list of legal pixels in which the path can pass. See section 4.2.2 in the paper for more details.

BottumPathLimit=0.0;

Similar to ***TopPathLimit*** only for the distance from the bottom of the image. See section 4.2.2 in the paper for more details.

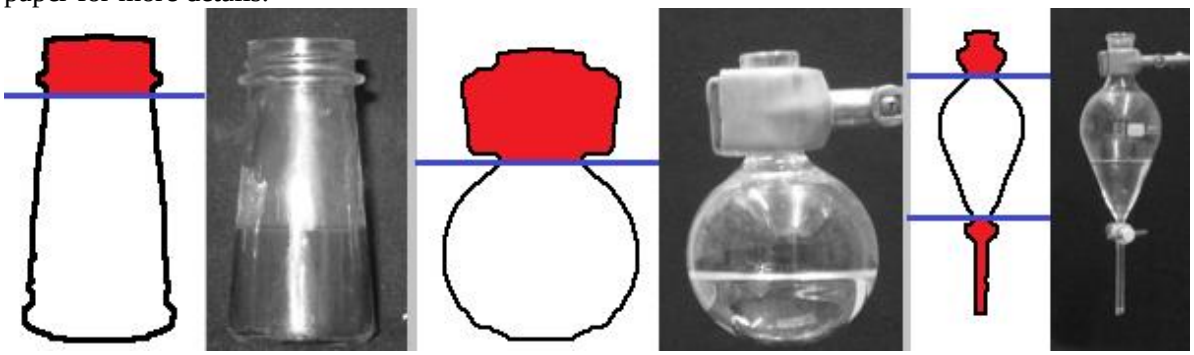

Figure 5. Areas at the top and bottom of the vessel region of the image (red) often correspond to corks, funnels and valves, and are therefore ignored.

MinPathDist

The minimal horizontal distance (in pixels) between the boundary path endpoints (**E,S** Figure 4). Hence, paths for which the horizontal distance between endpoint smaller than ***MinPathDist*** will not be explored (assuming they belong to corks and funnels which usually correspond to narrow region of the vessel). The values of these parameters is are usually defined as some fraction of the vessel maximal width in pixels. See section 3.2 of the paper for more details.

HighPriceZoneWidth=0.1;

The width of the high-price /penalty-zone as a fraction of the vessel minimum dimension. Hence width of the region in the image around the vessel boundary in which the cost of path propagation will be tripled (Figure 6). If the average width of the vessel is **AvWidth** and the average height of the vessel is ***AvHeigh***.
Then the every pixel in distance of ***HighPriceZoneWidth*min(AvHeight,AvWidth)*** from the vessel boundary will be in the penalty zone (Red region, Figure 6) . The path cost in this region will be tripled.
See section 4.3.2 of the paper for more details.

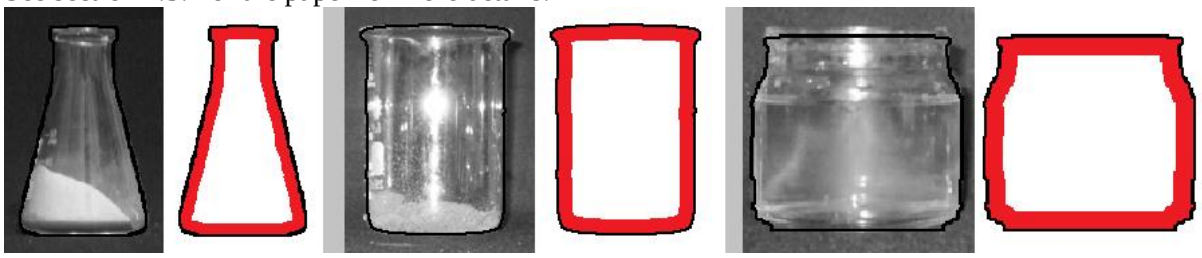

Figure 6. Penalty zone: Image areas near the vessel boundary (marked black) are more likely to cause false recognition, therefore all pixels in some distance of the vessel boundary (red) are considered as penalty zone where the cost of the path is triple.

MinDy=0.25, MinDyD=10;

The vertical distance ($\Delta Y_{1/2}$ Figure 4e) between any point (C) on the path and the vessel boundary, most exceed ***MinDy*** % percent of this point (C) horizontal distance ($\Delta X_{1/2}$ Figure 4e) to the path closest endpoints (S, E Figure 4e) or some minimal distance ***MinDyD***.
Hence: $\Delta Y_{1/2} > \mathbf{min}(\Delta X_1 \cdot MinDy\,, \Delta X_2 \cdot MinDy\,, MinDyD)$. (**min** stands for minimum)

2.3. Deifinig Cost for path

Evaluation of the path (phase boundary curve) cost in each point in the image is done using cost function described in section 5 of the paper.

In the code The cost of each point of the path is calculated using the function: ***PointPrice()***

Found in the file ***GetPathPriceMap.m***

A few different methods for calculating the path cost defined in section 5 of the paper.
Each function correspond to different methods for calculating the path cost is defined as function with name:

PointPriceX()
Where **X** is a number. These functions are defined in the bottom part of ***GetPathPriceMap.m*** file.

You can replace the cost calculating method by replacing the number ***X*** in the line ***PointPriceX()*** within the function ***PointPrice()*** in ***GetPathPriceMap.***m file.

The default function ***PointPrice6()*** is based on the difference in edge density between path and its surroundings and give best results for all cases examined (see section 5.3 in the proper).

3. Main code description functions and parameters description

Below are basic description of basic functions. Note that the source code contains remarks and can be read as stand alone. This section can be used as an addition. The theory beyond the program is described **in the paper:** "*Tracing the boundaries of materials in transparent vessels using computer vision*" Freely available at Arxiv:
http://arxiv.org/ftp/arxiv/papers/1501/1501.04691.pdf

3.1. Global parameters

Below is the list of the most important parameters in the code. Note that most parameters transferred between function using global (extern) and not input output. Additional parameters are discussed in section 2.2.

Icol: Color image of the vessel containing the material.
I: Grayscale image of the vessel containing the material.
Iedge: Canny edge image of the vessel containing the material.
Hight, Width: Height and width of of the image examined (***I***).

Ax,Ay, Bx,By: Coordinates of endpoints of the path that is explored in this step. This coordinates refer to the pixels in the image in which the path start (***Ax,Ay***) and end (***Bx,By***).

LegalPath**:** Binary image in the size of the image examined (***I***) in which all legal pixels in which the path currently explored can pass marked 1(green region in Figure 4) and the rest 0.

LegalPathA**:** Binary image in the size of the image examined (***I***) in which all legal pixels in which the paths starting at point ***A*** can pass marked 1(green region in Figure 4) and the rest 0.

LegalPathAB**:** Binary image in the size of the image examined (***I***) in which all legal pixels in which the paths starting from point ***A*** and ending at point ***B*** can pass marked 1(green region in Figure 4) and the rest 0.

PathMap: Matrix in size of the image examined (***I***) in which every cell contains the cost of the path from start point ***A*** to this cell

BackX, BackY: Matrix in size of the image examined (***I***) in which every cell contains coordinates of the previous pixel in the path from the start point ***A*** to this cell. This matrix allows you to track all paths leading from ***A*** in the image.

Y, X: coordinates of the current pixel examined on the path.
Xb, Yb: coordinates of the previous pixel examined on the path.

HighPriceZone: Binary image with the region corresponding to the penalty zone in the paper marked 1(section 4.3.2) cost of paths in this pixel will be triple. Hence a binary image in size of the image examined (***I***) where the pixels of the penalty zone marked 1 and the rest zero

BstPath: Binary image size of the image examined (***I***) were the best path (hence material boundary) is marked with 1. In the end of the run this will contain the material boundary.

BstPrice: Cost of the best path (BstPath) found so far.

3.2. Description of main functions

Function Main_find_phase_boundary

[MarkedImage,BinaryMaterialBoundaries]=Main_find_phase_boundary(Icolor,Iborder);

***Description*:**
The main function finds return the boundary of material in the image of the vessel containing the material.

Input:

***Icolor*:** color image of a transparent vessel containing the material (Figure 1).

Iborder*:** binary edge image in the size of Icolor with the boundaries vessel in the image Icolor marked 1 (all other pixels are 0, Figure 1). The line in this image must be of close contour and thickness of 1 pixel. Finding the vessel boundary (Iborder***) could be done automatically using the code available at freely available at the addresses below :

http://www.mathworks.com/matlabcentral/fileexchange/46887-find-boundary-of-symmetric-object-in-image
http://www.mathworks.com/matlabcentral/fileexchange/46907-find-object-boundaries-in-image-using-template--variable-image-to-template-size-ratio-

OUTPUT

***MarkedImage**:* The image of the vessel with the material boundaries marked in black on the image (Figure 1).
BinaryMaterialBoundaries Binary edge image in size the input image with the material boundaries marked on it in 1 and all other pixels are zero.

In addition the above two images are displayed on the screen.

Running

In Matlab command line Run the line:

[MarkedImage,BinaryMaterialBoundaries]=Main_find_phase_boundary(Icolor,Iborder);

The input and output parameters defined above.
The results will be displayed on the screen and in output parameters.

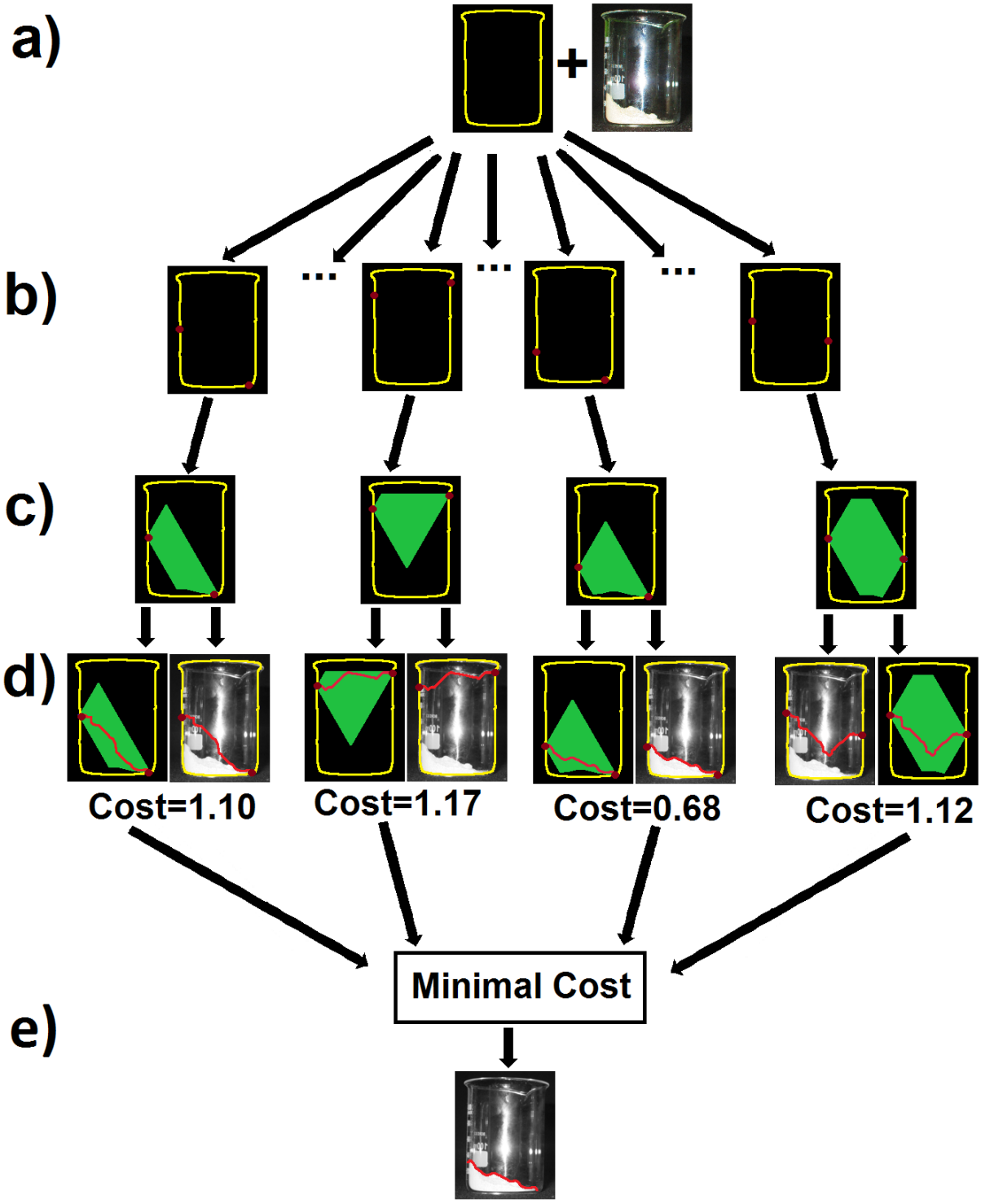

Figure 7. Simplified description of function *Main_find_phase_boundary* A general method for finding material boundary in the image. a) Receive image of the material in transparent vessel and the boundary of the vessel in this image. b) Scan every pair of points on the vessel boundary in the image. c) Use physical constraint to limit the image regions in which the phase boundary curve between this pair of points can pass. d) Find the optimal path (on the image) between this pair of points according to some cost function that is based image property that act as an indicator for the material boundary. e) The path with the lowest cost in the entire image is chosen as the material boundary.

General description of algorithm

The function basic principles are described in section 2 of the paper.

The simplified version of the code is given below and illustrated in Figure 7:

Remarks are marked in *italic green fonts.*

Simplistic description

```
Input1: Image %Image of the vessel containing the material
Input2: VesselContourPoints % Array containing pixels corresponding to vessel boundary in Image
Input3: NumVesselContourPoints % Number of points in VesselContourPoints array
-----------------------------------------------------------------------------------------------------------------------------------------------------------
For f=1:NumVesselContourPoints  %Scan all points on the vessel contour
    Endpoint1=VesselContourPoints(f);  % Define first endpoint of the path
    For f2=f1+1:NumVesselContourPoints  %Scan all points on the vessel contour
        Endpoint2=VesselContourPoints(f2);  %Define second endpoint of the path
        LegalPathRegion=FindLegalPropogationRegion(EndPoint1,EndPoint2);
% Find the legal region in the image in which the path between the two endpoints can pass
        (Path,PathCost)= FindBestPath(EndPoint1,EndPoint2, LegalPathRegion,Image);
%Find the best (Cheapest) path on Image between the two endpoints that pass only through the legal region
        If (PathCost<MinimalPathCost)
            MinimalPathCost=PathCost;
            MaterialBoundary= Path; %Boundary of the material in the image
        end;
%If the cost of this path is lower than the cost of any previous path, use this path as the new material boundary
    end; %End of loop with f1
end; %End of loop with f2
-----------------------------------------------------------------------------------------------------------------------------------------------------------
Output: MaterialBoundary %Boundary of the material in the image
```

Algorithm fast version (one actually used)

The simplistic method (appendix 10.1) demands the calculation of the optimal path for each pair of points on the vessel contour separately and is therefore time consuming. A better method is to calculate all the optimal paths leading from a given starting point A in one round. This is indeed possible using Dijkstra's algorithm. This, however, leads to a problem: The legal region in which the path is allowed to pass is defined by both endpoints of the path (Figures 7). Hence, if only the starting point (A, Figure 15a) is known, the resulting path might pass in image regions that are illegal for the second endpoint (B, Figure 15b). This means that calculating all optimal paths from point A might result in the optimal path from B to A passing illegal image regions with respect to point B (Figure 15). If this is the case, then the optimal path from A to B will have to be recalculated. However, given that the method is only looking for the single lowest cost path in the entire image, this will need to happen only if the path cost is lower than the cost of any previously calculated path, which is a rare event. Hence this will happen in very few cases and will not significantly slow the program.

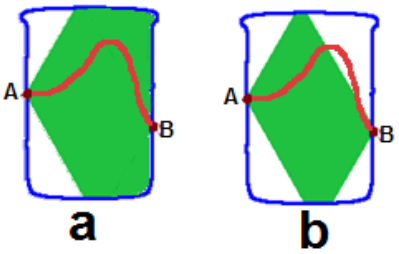

Figure 8: Path pass in legal regions with respect to endpoint *A (a)* might pass in an illegal region with respect to end point *B*(b). Legal region marked green path marked red.

A version of the algorithm based on the above concept is given below in Matlab notation.

```
Input1: Image %Image of the vessel containing the material
Input2: VesselContourPoints % Array containing pixels corresponding to the vessel boundary in Image
Input3: NumVesselContourPoints % Number of points in VesselContourPoints array
-----------------------------------------------------------------------------------------------------------------------------------------------
For f=1:NumVesselContourPoints  %Scan all points on the vessel contour
    Endpoint1=VesselContourPoints(f);  % Define first endpoint of the path
        LegalPathRegion1=FindLegalPropogationRegionPoint1(EndPoint1);
% Find the legal region in the image in which the paths leading from endpoint1 can pass.
        (Path,PathCostMatrix)= FindAllBestPaths(EndPoint1, LegalPathRegion1,Image);
%Find all the best path leading from EndPoint1 that pass only through the legal regions of  EndPoint1.
%Return a matrix (PathCostMatrix) of the cost of the path from EndPoint1 to every point on the image
        For f2=f1+1:NumVesselContourPoints  %Scan all points on the vessel contour
        Endpoint2=VesselContourPoints(f2);  %Define second endpoint of the path
                If (PathCostMatrix(EndPoint2)<MinimalPathCost)
%If the cost of this path from EndPoint1 to EndPoint2 is lower than the cost of any previous path found, then
%consider this path as the new material boundary
                    LegalPathRegion12=FindLegalPropogationRegion12(EndPoint1,EndPoint2);
% Find the legal region in the image in which the path between the two endpoints can pass
                    If (Path inside LegalPathRegion)
%if the path found pass only inside legal region defined for both endpoints write it as a new best path
                        MinimalPathCost=PathCost;
                        MaterialBoundary=PathMat(EndPoint2); %Boundary of the material in the image
                    else
%If the path pass in an illegal region with respect to the endpoint2, recalculate the path between  endpoints
                     (Path,PathCost)= FindBestPath(EndPoint1,EndPoint2, LegalPathRegion,Image);
%Find the best (Cheapest) path on Image between the two endpoints that pass only through the legal region
%for both endpoints
                     If (PathCost<MinimalPathCost)
                          MinimalPathCost=PathCost;
                          MaterialBoundary= Path; %Boundary of the material in the image
                    end;
%If the cost of this path is lower than the cost of any previous path use this path as the new material boundary
                end;
            end; %End of loop with f1
 end; %End of loop with f2
-----------------------------------------------------------------------------------------------------------------------------------------------
Output: MaterialBoundary %Boundary of the material in the image
```

Function FindLegalPathA()

Description

Find the legal region in the image in which a path starting at point ***A*** can pass. It returns the binary image in the size of the examined image (I) were pixel in which the path can pass are marked 1 (figure 4 green region) and illegal regions in which the path don't allow to pass marked 0. All parameters are passed from and to this function are global. Therefore, it has no direct input or output. For more details on the main principles of this function see section 4 of the paper.

Input (global)

Ax,Ay: Coordinates of start point of the path that is explored in this step. This coordinates refer to the pixel in the image in which the path start (***Ax,Ay***).

Output (global)

LegalPathA*:** Binary image in the size of the image examined (I***) in which all legal pixels in which the paths starting at point ***A*** can pass marked 1(green region in Figure 4) and the rest 0.

Function FindLegalPathAB()

Description

Find the legal region in the image in which a path starting at point ***A*** and ending at point ***B*** can pass. It returns the binary image in the size of the examined image (I) were pixel in which the path can pass are marked 1 (figure 4 green region) and illegal regions in which the path don't allow to pass marked 0. All parameters are passed from and to this function are global. Therefore, it has no direct input or output. For more details on the main principles of this function see section 4 of the paper.

Input (global)

Ax,Ay, Bx,By: Coordinates of endpoints of the path that is explored in this step. This coordinates refer to the pixels in the image in which the path start (***Ax,Ay***) and end (***Bx,By***).

Output (global)

LegalPathAB*:** Binary image in the size of the image examined (I***) in which all legal pixels in which the paths starting from point ***A*** and ending at point ***B*** can pass marked 1(green region in Figure 4) and the rest 0.

Function GetPathPriceMap()

[BestPathPrice]=GetPathPriceMap()

Description:

Find all cheapest paths on the image, leading from pixel ***A*** to any pixel in the image.
Or
Find the cheapest legal path on the image between points ***A*** and ***B*** and return this path and its cost.

Input Global:

Icol: Color image of the vessel containing the material.
I: Grayscale image of the vessel containing the material.
Iedge: Canny edge image of the vessel containing the material.
Hight, Width: Height and width of of the image examined (***I***).

Ax,Ay, Bx,By: Coordinates of endpoints of the path that is explored in this step. This coordinates refer to the pixels in the image in which the path start (***Ax, Ay***) and end (***Bx,By***).

LegalPath: Binary image in the size of the image examined (***I***) in which all legal pixels in which the path currently explored can pass marked 1 (green region in Figure 4) and the rest marked 0.

Output

PathMap: Matrix in size of the image examined (***I***) in which every cell contains the cost of the path from start point ***A*** to this cell

BackX, BackY: Matrix in size of the image examined (***I***) in which every cell contains coordinates of the previous pixel in the path from the start point ***A*** to this cell. This matrix allows you to track all paths leading from ***A*** in the image.

Algorithm

The algorithm is performed by step by propagation of all possible paths from starting position ***A*** to final position ***B*** inside legal image using Dijkstra's algorithm.

At each propagation step all propagation moves are evaluated. The Move is accepted only if the cost of the path containing the move to the move final position is lower than the current cost of the current best path leading from point ***A*** to this pixel.

Move is defined as the propagation of a path from one pixel to the adjacent pixel (Figure 9). The three constraints on the move are: a) The move initial and final postions (pixels) must within the legal region (***LegalPath***). b) the cost of the path from point ***A*** to the move ending position must be known. At start the only position for which the cost of the path from starting point ***A*** as been determined is the path starting position ***A*** (which have cost of zero).

If a move is accepted all the moves leading from this move final position are added to the list of moves to explore and the move is removed from the list of moves to be explored. If move is rejected it is removed from the list of moves to be explored

The algorithm is based on 3 steps.

1) Explore all available moves with none vertical propagation (from position **(x,..)** to **(x+1,…))**.

2) Explore all vertical moves (from position **(x,y)** to **(x,y+/-1)**). Repeat this step (2) for ***VerticalMoves*** times (**VerticalMoves**=3 as default, see section 2.2).

Repeat the above two steps until the path reach point ***B*** or until no more moves has been left.

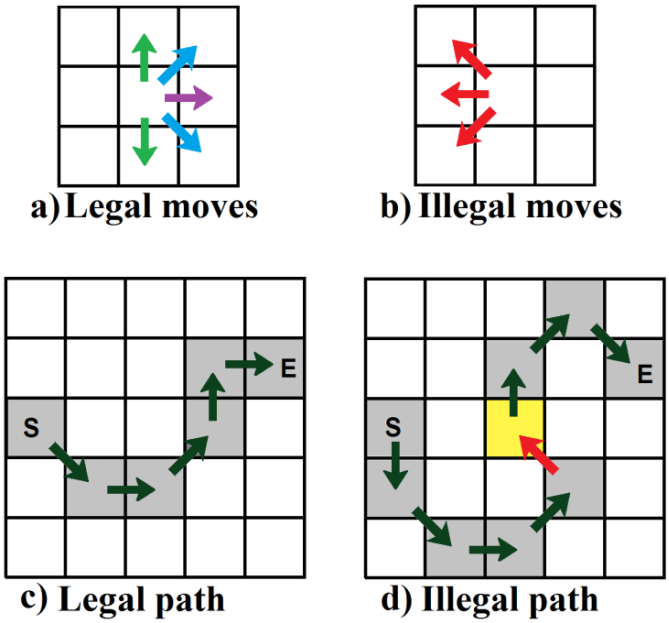

Figure 9. a) Legal directions for moves in the boundary path exploration include all moves to neighboring pixels from left to right and vertical moves. b) Illegal moves in path propagation include all moves from right to left. c) Example of a legal path from S to E (only left to right and vertical moves). d) An illegal path from pixel S to E (due illegal backward move [red arrow]).

Function PointPrice()

[PointPrice]= PointPrice()

Description:

Find the cost of the current move in the path. See section 2.3 of this document and section 5 of the paper for more details.

Input (global)

Y, X: coordinates of the current pixel examined on the path.
Xb, Yb: coordinates of the previous pixel examined on the path.
HighPriceZone: Binary image with The region corresponding to the penalty zone in the paper marked 1(section 4.3.2). Hence a binary image in size of the image examined (***I***) where the pixels of the penalty zone marked 1 and the rest zero.) Cost of paths in these pixels will be triple.

One of the images:

Icol: Color image of the vessel containing the material.
I: Grayscale image of the vessel containing the material.
Iedge: Canny edge image of the vessel containing the material.

Output

PointPrice The cost of the move from (***Xb,Yb***) to (***X,Y***).

Function GetPathFromMap()

[BstPathAB]=GetPathFromMap()

Input (global)

BackX, BackY: Matrix in size of the image examined (***I***) in which every cell contains coordinates of the previous pixel in the path from the start point ***A*** to this cell. This matrix allows you to track all paths leading from ***A*** in the image.

Ax,Ay, Bx,By: Coordinates of endpoints of the path that is explored in this step. This coordinates refer to the pixels in the image in which the path start (***Ax,Ay***) and end (***Bx,By***).

Input

BstPathAB: Binary image size of the image examined (***I***) where the path between points ***A*** and ***B*** is marked with 1.

Function CreateCurvatureMap()

[Mat]=CreateCurvatureMap(I)

Description:

Create a curvature factor map for cylindrical (axisymmetric) vessels based on their outline in the image.

Use the contour line (***I***) of the vessel in the image to find the curvature factor of each point in the vessel surface in the image. Return map (***Mat***) of the curvature factor as matrix of type double and size of I. Could be used for adjusting for reflections from the vessel surface in recognition of materials in transparent vessels

Input:

I: a Boundary curve of axisymmetric vessel in the image. Hence contour line of the vessel in the image as binary edge image.
I most be a close curve in the thickness of one pixel and symmetric with respect to the Y axis

Output:

Mat: output curvature map (Curvature factor F) in each point inside the vessel region of the image. A double matrix of size Y.