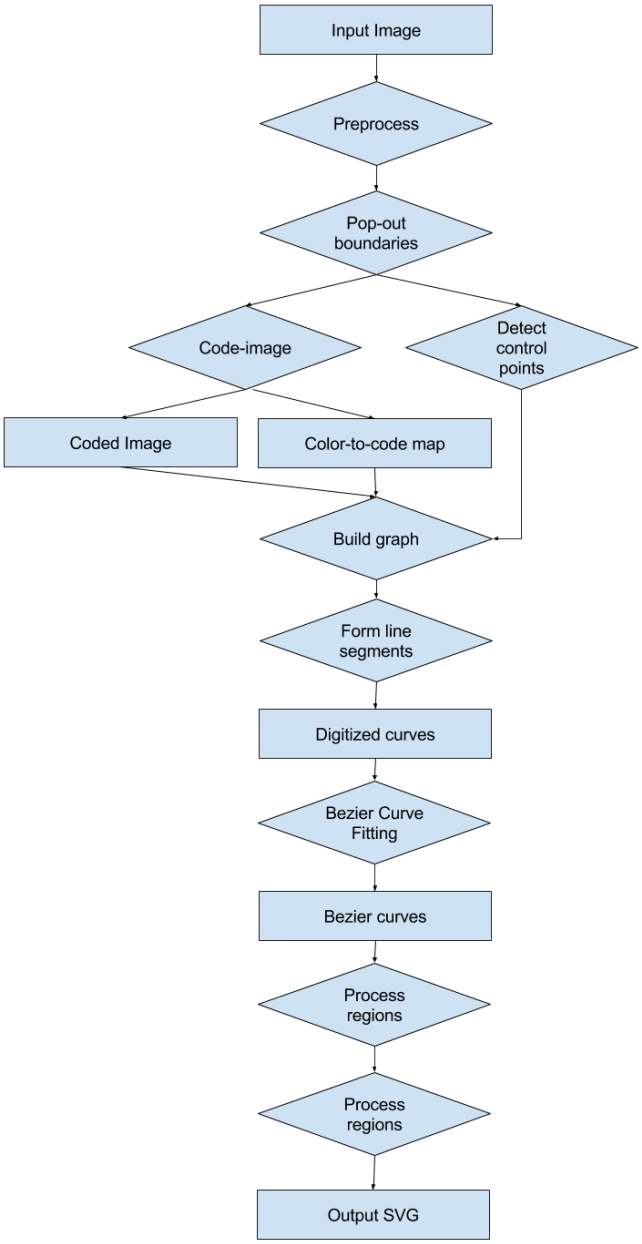
Mindthegap

This documentation contains a brief on some of the routines in mindthegap. Please feel free to raise an issue or contact the authors in case of any doubt.

Flowchart



Preprocess

This stage operation involves the following:

I. Finding a boundary pixel
   - Takes the four corner pixels of the input image.
   - Computes $F_i = R_i + 255G_i + 255^2B_i$ for all four pixels.
   - Computes the median F_{med} of the four values F_i.
   - Retrieves R_{med}, G_{med} and B_{med} from F_{med}.
   - Forms a border pixel whose value is (R_{med}, G_{med}, B_{med})

II. Pads the input image with border pixels (padding size is 1).

III. Computes a boundary pixel (B) which is any one pixel not present in the padded image.
   - This pixel will be used later to represent boundary points.

Pop-out boundaries

This stage operation involves the following:

I. Forms an empty image (G) of width and height, twice the width (w) and height (h) of the padded input image (H).

II. Assigns values to the pixels of G in following manner (i varies [1,h-1], j varies [1,w-1])
   - ($focus_i$, $focus_j$) = (2i+1, 2j+1)
   - G($focus_i$, $focus_j$) = H(i,j)
   - For (p,q) in {(1,0), (0,1), (1,1)}
     - G($focus_i$-p, $focus_j$-q) = H(i-p, j-q) if H(i,j) equals H(i-p, j-q)
     - G($focus_i$-p, $focus_j$-q) = Boundary pixel (B) otherwise.

III. Finally, we get the popped out boundaries image, which we denote by G.

Code-image

This stage operation involves the following:

I. Connected component labelling of the popped out boundaries image (G).

II. Outputs the a 2D matrix with values based on the label of the connected component to which the corresponding pixel belongs. Also, outputs the connected component code (label) to color (pixel) mapping.

Build graph

This stage operation involves the following:

I. Initialize graph with boundary pixels as vertices.
II. Detect control points.
    - These are those boundary pixels which have greater than or equal to 3 adjacent regions. In other words, those boundary pixels with greater than or equal to 3 unique colored adjacent pixels are defined as control points.
    - Note that a single pixel has 8 adjacent pixels.
III. Detect dangerous points.
    - These are those non boundary pixels which have adjacent boundary pixels in L,R,U,D positions.
    - These will be used to remove connections termed as dangerous.
IV. Form adjacency list where two vertices (boundary pixels) form an edge if they are adjacent.
V. Remove dangerous connections.
    - The dangerous points (pixels) have boundary pixels at L,R,U,D positions relative to it. Note that L is connected to U and D and vice versa. Similarly, R is connected to U and D and vice versa.
    - If the a pixel is a dangerous point then the connections between the surrounding boundary pixels (L,R,U,D) are termed as dangerous if one of the following conditions are met:
        1. L -- U and R -- D are dangerous connections if the pixel values at LU and RD positions are same as the pixel value of the dangerous point (pixel).
        2. L -- D and R -- U are dangerous connections if the pixel values at LD and RU positions are same as the pixel value of the dangerous point (pixel).
    - Those connections between boundary pixels which satisfy one of the above condition and hence are termed as dangerous connections are removed from the adjacency list.

Form line segments

This stage operation involves the following:

I. A list of line segments (vector of points) connecting one control point to another or surrounding an island (independent region) is formed by running a graph search algorithm (with some modifications) over the graph formed in the previous stage.
II. Each line segment is assigned with a set of ids of the regions of which it forms a part of the boundary.
III. This stage outputs the digitized curves.

Bezier curve fitting

This stage operation involves the following:

I. The digitized curves formed in the previous stage are passed onto the bezier curve fitting routine based on **[1]**.
II. The output of the routine is a list of bezier curves (vector of points).

Process regions

This stage operation involves the following:

I. The bezier curves surrounding each region are collected and arranged in order to form a closed path.
II. These regions along with their surrounding bezier curves are then passed on to the svg writer which then outputs the final svg.

References

1. Philip J. Schneider. "**An Algorithm for Automatically Fitting Digitized Curves**". In Graphics Gems, Academic Press, 1990, pp. 612–626.

Extra notes

Dangerous Pixels and Connections

Dangerous points/pixels

- Those non boundary pixels which have boundary pixels at UP, DOWN, LEFT, RIGHT.

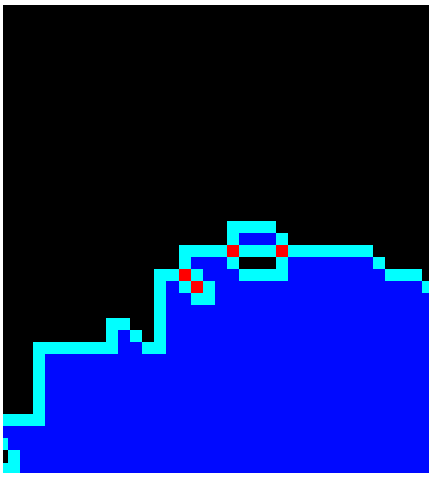
Dangerous connections:

- Let x be a dangerous point.
- If, in popped out boundary bitmap, x->UPLEFT and x->DOWNRIGHT have same rgb value as x then connection between [UP and LEFT] & [DOWN and RIGHT] are dangerous.
- Else if, in popped out boundary bitmap, x->UPRIGHT and x->DOWNLEFT have same rgb value as x then connection between UP and RIGHT & DOWN and LEFT are dangerous.

Following are the pictorial representation of above two cases (B represents boundary, ? represents don't cares):

x	B	?	?	B	x
B	x	B	B	x	B
?	B	x	x	B	?

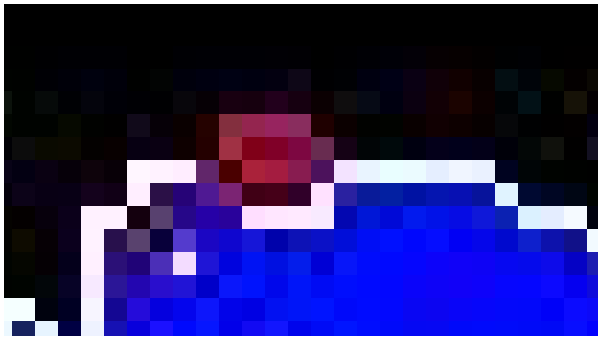
An example:



The pixels marked with red color are dangerous ones. Why these are called dangerous? Consider a representation of the scenario in above images: X = Black, C=Dark Blue, B=Sky Blue (boundary pixel),

X	B_1	C
B_2	X	B_3
C	B_4	X

On running the search algorithm over the vertices of the graph which are the boundary pixels with connections to their adjacent boundary pixels, when the step reaches B_1, the expected next step should be B_3 (because that's how a trail will be formed) but the algorithm faces two options for the next step (B_3 and B_2) and can any of those steps. This may result in incorrect output which is illustrated below.



Here, white colored path is not detected as a boundary of the region while red colored paths are detected. Clearly, they do not form the boundary of the region.