Intelligent Scissors

Problem Definition

Selection tools (similar to the one in MS-Paint ex: fig1) can be used to select objects in an image to resize/delete/copy/move the objects. There are many types of selection tools such as rectangles or free-form selection tool, sometimes free-form selection tools are called *Lasso's*. You can imagine a lasso as a rope surrounding your selection. Unfortunately, selection using ordinary lasso's can be tedious and boring. In Photoshop, there is a more advanced version of ordinary lasso's called *Magnetic Lasso Tool.* Magnetic Lasso Tool is a lasso that automatically snaps to the objects' boundaries (ex: fig2). You can watch a demo of it here

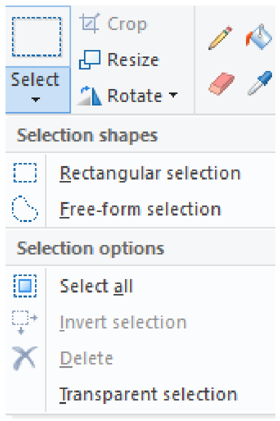


Fig1: selection in MS-Paint

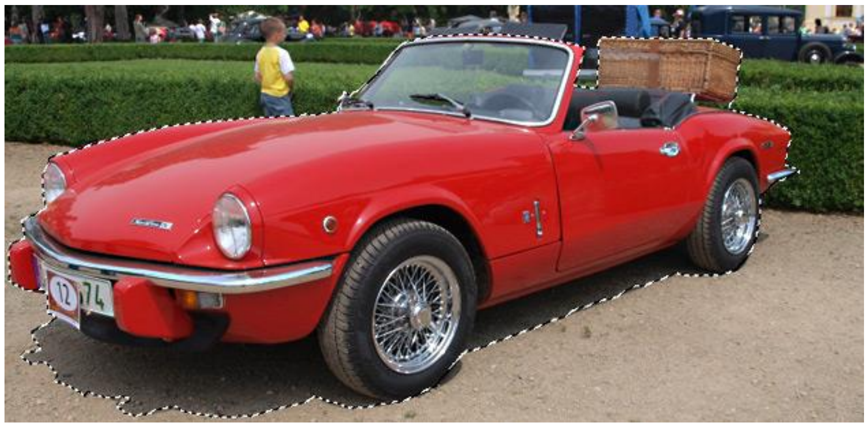


Fig2: car selected using magnetic lasso tool

The technical term for the Magnetic Lasso Tool is *Livewire* or *Intelligent Scissors*. In this project we want to implement a simple magnetic lasso to learn more about image processing, graphs, and greedy algorithms.

Terminologies

1) **Livewire:**
   - A livewire is defined by two points and a wire (path) between them:
     i. Anchor point: a fixed point on the image the user selects at the beginning.
     ii. Free point: a moving point following the mouse cursor.
2) **Image:**
   - 2D images are usually represented as a 2D array of pixels.
   - Each pixel may contain either one (for gray images) or three (for colored images) values (fig3.a). These values are often called *Image intensity (I)*.
   - Image intensity at pixel(i,j) (I[i,j]) is the color of the image at that pixel.
   - Ex: in the rubik's cube image shown (Fig3.a), it is a 512x512 RGB image, and each pixel contains three values to represent the color RGB (Red, Green, Blue).
3) **Converting colored image to grayscale:**
   - Colored images can often be converted to grayscale by taking the average value of the image's three channel's (Red, Green, Blue) (fig3.b).

4) Edge detection:

- There many simple image filtering techniques that can detect the object boundaries (edges) and tell us the position and strength of an edge at a certain pixel (fig3.c).
- An image-edge can be simply defined as a sudden change in image intensity at a certain position.
- Since these edges represent the object's boundary, we can use these edges to snap or pull the lasso towards them.

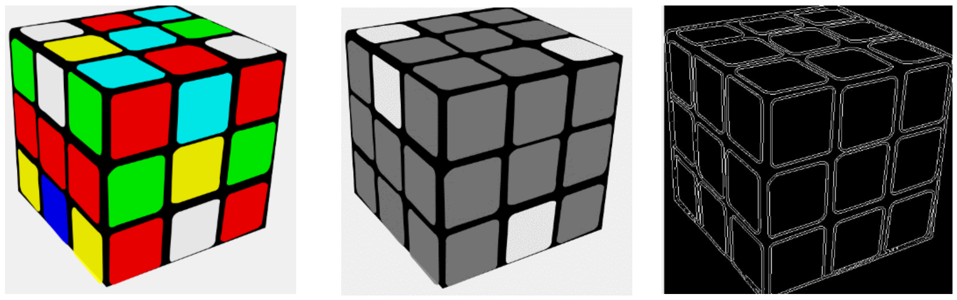


Fig3: a) colored image. b) gray-scale image. c) edge-image.

To calculate the energy of an image, you need to convert the input image to grayscale by looping through the image and for each pixel the gray value is the average of the red, green, and blue values of this pixel. After that, you need to calculate the derivative of the image in the x-axis which results in Gradient-X (Gx) and in the y-axis which results in Gradient-Y (Gy). Then, the energy function is defined as follows:

$$E(img) = \sqrt{Gx^2 + Gy^2}$$

Gx and Gy can be calculated be applying a filter on the input image using convolution. The filter of x-direction is defined as:

$$f_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

The filter in y-direction:

$$f_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

To apply a filter on the input image, we need to pad the image with zeros from top, left, right, and bottom.

$$\text{image} = \begin{bmatrix} 10 & 20 & 51 & 61 \\ 41 & 94 & 12 & 55 \\ 52 & 22 & 31 & 81 \\ 12 & 43 & 68 & 30 \end{bmatrix} \quad \rightarrow \quad \text{Padded\_image} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 20 & 51 & 61 & 0 \\ 0 & 41 & 94 & 12 & 55 & 0 \\ 0 & 52 & 22 & 31 & 81 & 0 \\ 0 & 12 & 43 & 68 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

For each actual pixel (not a padded pixel) in the padded_image you need to apply convolution operation between the filter and a window of size 3x3 around the current pixel. Ex:

Window of size 3x3 (W)

$\downarrow$

$$f_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \text{Padded\_image} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{10} & \mathbf{20} & 51 & 61 & 0 \\ \mathbf{0} & \mathbf{41} & \mathbf{94} & 12 & 55 & 0 \\ 0 & 52 & 22 & 31 & 81 & 0 \\ 0 & 12 & 43 & 68 & 30 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

New _value = W * f_x

$= 0 * -1 + 0 * 0 + 0 * 1 + 20 * 2 + 94 * 1 + 41 * 0 + 0 * -1 + 0 * -2 + 10 * 0 = 134$

$$\text{Gx} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 134 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Repeat the same operation for each pixel to get the full output image

$$\text{Gx} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 134 & 53 & 43 & -114 & 0 \\ 0 & 230 & -38 & 22 & -106 & 0 \\ 0 & 181 & -15 & 66 & -142 & 0 \\ 0 & 108 & 91 & 33 & -167 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Do the same to get Gy by applying convolution on the image with f_y. Then, create an empty image for Magnitude (M), and for each pixel in M will be calculated as:

$$M[i,j] = \sqrt{Gx[i,j]^2 + Gy[i,j]^2}$$

Edge angle is calculated as following:

$$\alpha[i,j] = \arctan^2(Gy, Gx)$$

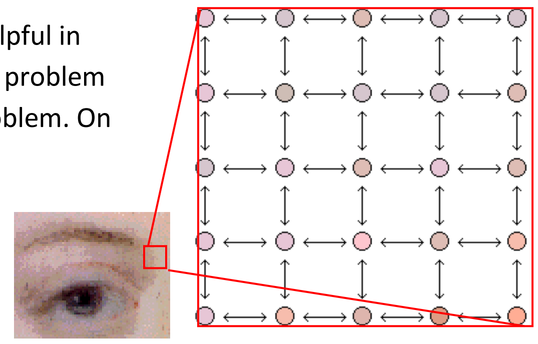
The Energy function is calculated as following:

$$E[i,j].x = M[i,j] * \cos\left(\alpha[i,j] + \text{PI} / 2.0\right)$$

$$E[i,j].y = M[i,j] * \sin\left(\alpha[i,j] + \text{PI} / 2.0\right)$$

5) Representing images by graphs:

- Images can be represented as graphs, which can be helpful in many image analysis problems, because it reduces the problem from an image domain problem to a graph domain problem. On



such graphs you can apply typical graphs algorithms (Dijkstra, BFS, DFS, …etc) to solve the problem at hand.

6) **Graph construction:**
   - To construct an undirected weighted-graph we need to define:
     i. Vertices (nodes).
     ii. Connectivity (edges).
   - This image-graph can be structured as follows:
     i. Vertices: Each image pixel is considered as a vertex in the graph. So if we have an NxM image then we have an NxM vertices in our graph.
     ii. Connectivity: there are many ways to connect the vertices grid, the simplest way is to establish a 4-connectivity (fig5). So we need to connect each pixel with the pixel on the above, below, left, and right.

Fig4: image pixels represented as graph vertices (nodes), and the pixels' neighborhood is represented as graph edges.

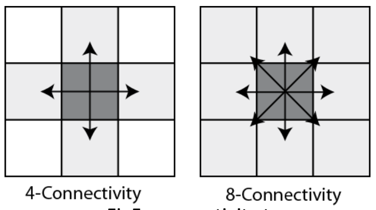


Fig5: connectivity types

7) **Mapping a "livewire in an image" problem to a "shortest path in a graph" problem:**
   - Assuming that we have an undirected weighted-graph for the image, with **small weights** on the objects' boundaries (**image-edges**) and **large weights** at the **homogenous parts** of the image.
   - If we need to generate a livewire between two pixels P1(i,j) and P2(x,y), it is the same as getting the shortest path between the two corresponding vertices V1(i,j) and V2(x,y), because the low edge-weights are at the image-edges on which we want our livewire to snap on.
   - Now remains one issue towards constructing our graph, which is determining the edges' weights between pixels.

8) **Edge Weights Generation:**
   - Assuming that you have a value G that measures the image-edge strength and direction between two pixels P1 and P2.
   - Then we can set the edge-weight between P1 and P2 as $W_{p1,p2} = 1/G$. so regions with Low G have high weight, and regions with high G have low weight. In 4-connectivety, if P2 is a horizontal neighbor of P1 then G = Ex. On the other hand, if P2 is a vertical neighbor of P1 then G = Ey.

Project Requirements

Provided Implementation

1. Template for opening and displaying the images.
2. Function to calculate the edge-strength G between two pixels.

Required Implementation

1. Construct an undirected weighted-graph for a given image.
2. Calculate the shortest path EFFICIENTLY from an anchor pixel (vertex) to all pixels (vertices in the graph).
3. Backtrack the shortest path from a free point (mouse position) to the anchor point.
4. Draw the path on the image.
5. Generate a sequence of connected paths using multiple anchor points.
6. When the user finish selection close the lasso by generating a path between the last and first anchors.

Input

1. Image (2D array of pixels).
2. Anchor point.
3. Free point.

Output

1. Path between the anchor and the free point.
2. Final lasso closed path and draw it as red lines on the input image.

BONUSES

1. As you can see in Photoshop example you can 1) Click to place anchor. 2) Move the mouse to generate the livewire. 3) When the wire's length exceeds a certain length, an automatic anchor point is placed to make the wire more stable.

   Bonus: implement a similar algorithm that automatically places new anchor points.
2. Add the ability to increase the frequency of anchor points in some critical regions (or any other Photoshop-like features as shown here)