# Lazy Snapping and Poisson Image Editing

*Abstract: in this report, I implement lazy snapping and Poisson Image blending for image editing. First of all, I introduce that how lazy snapping works and the results from graph cut, the second part is about Poisson image editing.*

## 1. Lazy snapping and graph cut

### 1.1algorithm

in order to understand what lazy snapping and graph cut are, we need to convert an image to a format of graph.

First of all, we need to evaluate the cost of each pixel, which is the basically the probability of assigning each pixel to source or terminal. Since we are given the two different strokes corresponding to foreground and background, so we can use the pixel value (R, G, B) in different strokes to fit in two gaussian models to represent the distributions of background and foreground points. During this process, we can use K-means to select how many clusters we want to fit in the each gaussian model.

After finding two gaussian models of foreground and background, that is to say, for every single pixel in the image, we can calculate the probability in these two gaussian distribution to give a coarse prediction of which distribution this pixel belongs to. Meanwhile, we can get two probabilities, and one is corresponding to the probability of background and the other one is corresponding to foreground. We repeat this procedure for every pixel in the image, then we can have a coarse prediction label and a probability table.

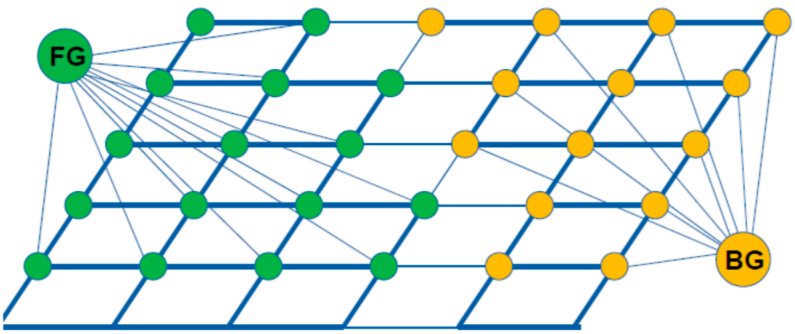
Now, we need to calculate the color similarity to evaluate the similarity of two neighboring pixels using gaussian models showed below. This similarity matrix evaluates how similar two neighboring pixels are.

$$e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Now, we introduce the smooth cost. This term is dependent on how similar of the two neighboring pixels are. If two neighboring pixels are assigned both to foreground or background, then the smooth cost will be assigned with 0, otherwise, the value is 1. The reason that we do in this way is because we would like to penalize when two neighboring pixels are assigned to different classification.

So far, we have a coarse prediction model, a probability table, the similarity matrix and smooth cost. So now we can fit in all these four variables into the min-cut function. Then the result is going to be a label, so we just need to multiply our original image with this label to get the segmentation result from mean cut.

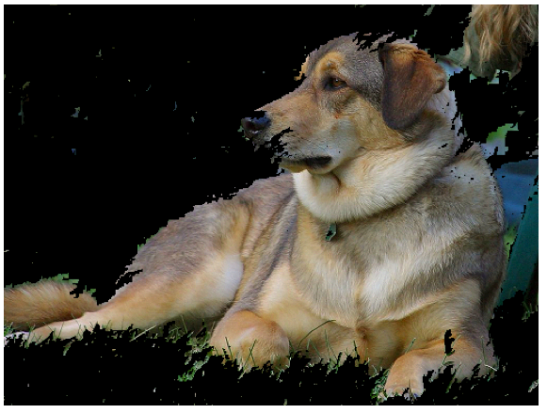
### 1.2more illustration of lazy snapping

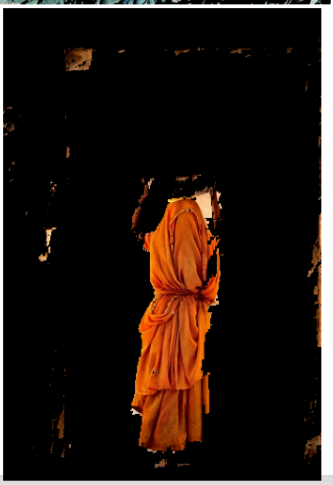
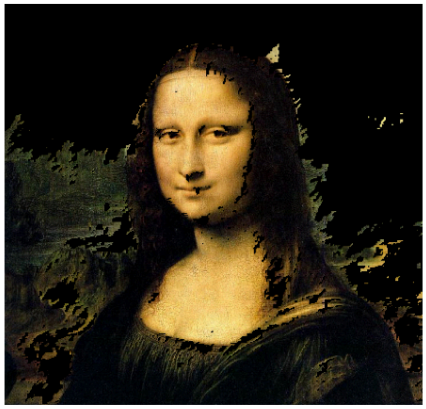
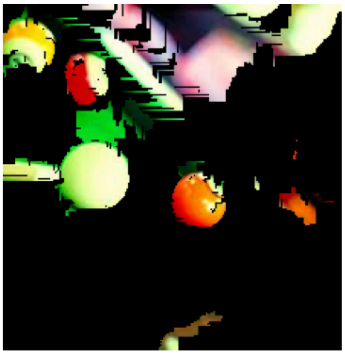



So, basically what we need to do is to transfer an image into the format of the above graph. The "FG" and "BG" is corresponding to the 2 gaussian model we find. And the edges between two nodes are defined by the similarity matrix. And smooth cost is defined by a 2*2 matrix representing the how we emphasis on the smooth. The blue and yellow points corresponding to the coarse prediction label we get, edges connected to background or foreground are corresponding to the probability table.

## 1.3Results from lazy snapping

After setting everything done, we have results in the following way:

## 1.4 Discussion about lazy snapping

1.4.1 what makes the lazy snapping difficult?

The difficult part is that we need to understand the meaning behind this min-cut algorithm. For instance, we need to understand how to convert an image to a graphical model and fit the probability model to an image.

1.4.2 what kind of data works best/worst

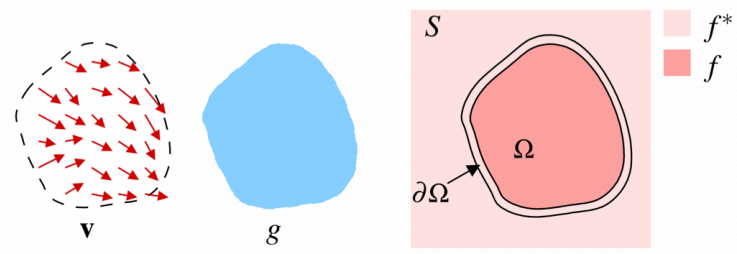
Since lazy snapping is really a data dependent algorithm. That is to say, for those images which have a clear border between foreground and background, the result is pretty good. However, when it does not have a clear border, the result is pretty noisy. Like table balls and Mona Lisa, these two images do not have a clear border, then the result is not so good.

1.4.3 how the implemented algorithm can be improved?

Since each image's properties are different, We can try different hyperparameters to find the best hyperparameters to fit for different images. For example, then first time I used [0 1;1 0] for the "dog" image, but I got a pretty noisy result. And I found that the border is even worse. So I increase the label cost to penalize the neighboring pixels, then I get a pretty good result.

# 2. Poisson Image Blending

## 2.1algorithm




$S$: target image;
$f$: to be solved in in domain Ω
$f$*: known region;
$g$: source image;
$v$: vector field for guidance

The algorithm about Poisson image blending is pretty simple and straightforward. Basically, the math behind Poisson image blending is capturing the image second derivative from the source image and keep the reconstruction part's second derivative as same to the source image as possible. The reason we do that is because image gradient can describe the change, like color change or intensity change. With this idea, we can make the pasted part's border pretty smooth.

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}$$

So basically, we are minimizing the loss function in the above formula.

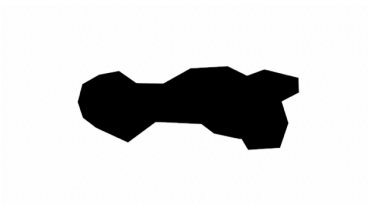
And this second derivative can be done by convolving the image with Laplacian kernel.

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

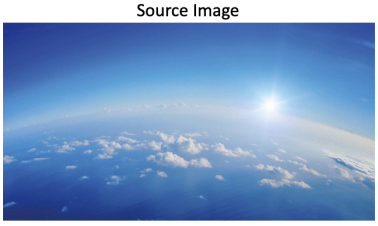
Then building a sparse matrix to contain all coefficients and solve it by linear least squares, then we can recover the area.

## 2.2Results

Source Image



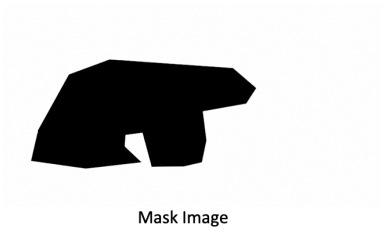
Mask Image



Target Image

Result Image

Source Image



Mask Image

Target Image

Result Image

## 2.3discussion about Poisson image Blending

as we can see, the results are pretty good, if we do not zoom in image, we cannot even see that this is artificial. The reason could be in this way: the target image is pretty clear, that is to say, the gradient is not so complex, so the reconstruction part is pretty smooth.