The Chinese University of Hong Kong,
Shenzhen

CIE6004 Image Processing and Computer Vision

Homework 2 Report

Grabcut: Interactive Foreground Extraction using Iterated Graph Cuts

---

Name: Xiang Fei
Student ID: 120090414
Email: xiangfei@link.cuhk.edu.cn
Date: 2022.10

Table of Contents

List of Figures

List of Tables

1 Introduction

In homework 2, we are required to write a program to reproduce GrabCut, which is an Interactive Foreground Extraction using Iterated Graph Cuts, based on the given paper. Separating an interesting part of an image or video from the original image or video, the main function is to combine it with another image or video to form a new image or video. The so-called image segmentation refers to dividing the image into several non-overlapping regions according to features such as grayscale, color, texture, and shape, and making these features appear similar in the same region, but obvious in different regions. difference.

We first give an overview of the current main image segmentation methods. First, the threshold-based segmentation method: The basic idea of the threshold method is to calculate one or more grayscale thresholds based on the grayscale features of the image, and compare the grayscale value of each pixel in the image with the threshold, and finally divide the pixels according to The comparison results are sorted into appropriate categories. Therefore, the most critical step of this kind of method is to solve the optimal gray threshold according to a certain criterion function.

Second, edge-based segmentation methods. The so-called edge refers to the set of continuous pixels on the boundary line of two different areas in the image, which is the reflection of the discontinuity of the local features of the image, and reflects the sudden change of image characteristics such as grayscale, color, and texture. Usually, edge-based segmentation method refers to edge detection based on gray value, which is based on the observation that the gray value of the edge will show a step-like or roof-like change. There are obvious differences in the gray value of pixels on both sides of the step edge, while the roof edge is located at the turning point where the gray value rises or falls. Based on this characteristic, the differential operator can be used for edge detection, that is, the extremum of the first derivative and the zero-crossing point of the second derivative are used to determine the edge. In the specific implementation, the image and the template can be used for convolution to complete.

Third, Segmentation method based on graph theory. Such methods relate the problem of image segmentation to the problem of minimum cuts of graphs. First, the image is mapped to a weighted undirected graph. Each node in the graph corresponds to each pixel in the image, and each edge connects a pair of adjacent pixels. Non-negative similarity in terms of degree, color, or texture. A segmentation s of an image is a clipping of the image, and each segmented area corresponds to a sub-image in the image. The optimal principle of segmentation is to keep the similarity of the divided subgraphs to the maximum internally and the similarity between the subgraphs to be the smallest. The essence of the segmentation method based on graph theory is to remove specific edges and divide the graph into several subgraphs to achieve segmentation. The methods based on graph theory that we have learned so far include GraphCut, GrabCut and Random Walk.

In the given paper, GrabCut method is proposed, which is an improved version of Graph Cut, an iterative Graph Cut. The algorithm utilizes the texture (color) information and boundary

(contrast) information in the image, and only a small amount of user interaction can get better segmentation results.

2 Design and Method

2.1 Graph Cut

Graph cuts is a very useful and popular energy optimization algorithm, which is widely used in the field of computer vision for foreground and background segmentation, stereo vision, matting, etc.

Such methods relate the problem of image segmentation to the problem of minimum cuts of graphs. The image to be segmented is first represented by an undirected graph, where V and E are sets of vertices and edges, respectively. The Graph here is slightly different from the normal Graph. Ordinary graphs are composed of vertices and edges. If the edges are directed, such a graph is called a directed graph, otherwise it is an undirected graph, and the edges are weighted, and different edges can have different weights , which respectively represent different physical meanings. The Graph Cuts graph has two more vertices on the basis of the ordinary graph. These two vertices are represented by the symbols "S" and "T", which are collectively referred to as terminal vertices. All other vertices must be connected to these two vertices to form part of the edge set. So there are two kinds of vertices in Graph Cuts, and two kinds of edges. The Graph Cuts graph is like the following.

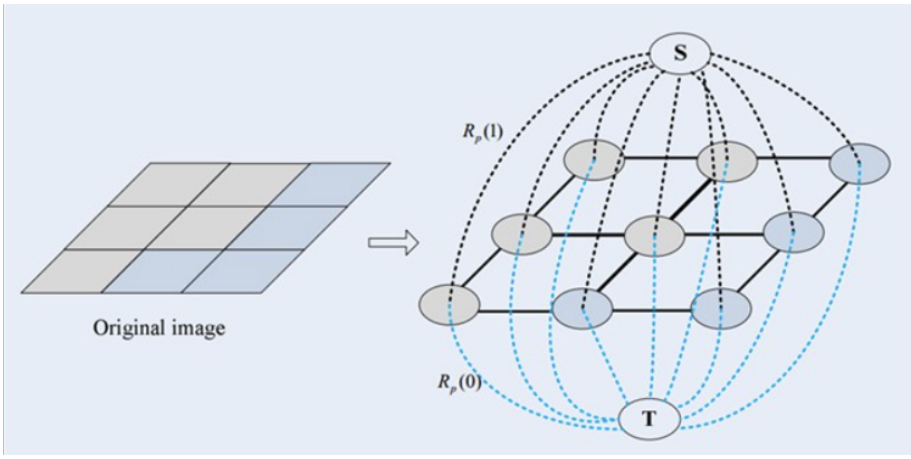


Figure 1: Graph Cut

The first kind of vertices and edges are: The first kind of ordinary vertices corresponds to every pixel in the image. The connection of every two neighbor vertices (corresponding to every two neighbor pixels in the image) is an edge. Such edges are also called n-links.

The second type of vertices and edges are: in addition to image pixels, there are two other terminal vertices, called S (source, meaning source) and T (sink, meaning convergence). There

is a connection between each ordinary vertex and these 2 terminal vertices, forming the second kind of edge. Such edges are also called t-links.

In foreground and background segmentation, s generally represents the foreground target, and t generally represents the background. Each edge in the graph has a non-negative weight we, which can also be understood as cost (cost or cost). A cut is a subset C of the edge set E in the graph, and the cost of this cut (expressed as —C—) is the sum of the weights of all the edges of the edge subset C.

Cuts in Graph Cuts refers to such a set of edges. Obviously, these edge sets include the above two kinds of edges. The disconnection of all edges in this set will lead to the separation of the residual "S" and "T" graphs, so it is called for "cut". If a cut has the smallest sum of the ownership values of its edges, then this is called a minimum cut, which is the result of a graph cut. The Ford-Fuksen theorem states that the maximum flow max flow of the network is equal to the minimum cut min cut. So the max-flow/min-cut algorithm invented by Boykov and Kolmogorov can be used to obtain the minimum cut of the s-t graph. This minimum cut divides the vertices of the graph into two disjoint subsets S and TThese two subsets correspond to the foreground pixel set and the background pixel set of the image, which is equivalent to completing the image segmentation.

Image segmentation can be seen as a pixel labeling problem. The label of the target (s-node) is set to 1, and the label of the background (t-node) is set to 0. This process can minimize the energy function by minimizing the graph cut. get. It is obvious that the cut that occurs at the boundary between the target and the background is what we want (equivalent to cutting the connection between the background and the target in the image, which is equivalent to dividing it). At the same time, the energy should be minimal at this time. Suppose the label label of the whole image (the label of each pixel) is L = l1,l2,,,,lp, where li is 0 (background) or 1 (target). When assuming that the segmentation of the image is L, the energy of the image can be expressed as:

$$E(L) = aR(L) + B(L) \tag{1}$$

Among them, R(L) is the regional term, B(L) is the boundary term, and a is the important factor between the regional term and the boundary term, which determines their influence on the energy. If a is 0, then only the boundary factor is considered, and the regional factor is not considered. E(L) represents the weight, that is, the loss function, also called the energy function. The goal of the graph cut is to optimize the energy function to minimize its value.

2.2 Differences between Graph Cut and Grab Cut

1 : The model of the target and background of Graph Cut is a grayscale histogram, and Grab Cut is replaced by the mixed Gaussian model GMM of RGB three channels.

2 : The energy minimization (segmentation) of Graph Cut is achieved at one time, and

Grab Cut is replaced by an interactive iterative process of continuous segmentation estimation and model parameter learning.

3 : Graph Cut requires the user to specify some seed points for the target and background, but Grab Cut only needs to provide the pixel set of the background area. That is to say, you only need to frame the target, then all the pixels outside the frame are regarded as the background. At this time, you can model the GMM and complete a good segmentation. That is, Grab Cut allows incomplete labelling.

2.3 Color Model

We use the RGB color space and use a full-covariance GMM (Gaussian Mixture Model) with K Gaussian components (generally K=5) to model the target and the background. Then there is an additional vector k = k1, . . ., kn, . . ., kN, where kn is the Gaussian component corresponding to the nth pixel, kn belongs to 1, ... ,K. For each pixel, either from some Gaussian component of the target GMM, or from some Gaussian component of the background GMM. So the Gibbs energy used for the whole image is as follows.

$$\mathbf{E}(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) = U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) + V(\underline{\alpha}, \mathbf{z}), \tag{7}$$

$$U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}) = \sum_{n} D(\alpha_n, k_n, \underline{\theta}, z_n), \tag{8}$$

$$D(\alpha_n, k_n, \underline{\theta}, z_n) = -\log \pi(\alpha_n, k_n) + \frac{1}{2} \log \det \Sigma(\alpha_n, k_n) + \frac{1}{2}[z_n - \mu(\alpha_n, k_n)]^{\top} \Sigma(\alpha_n, k_n)^{-1} [z_n - \mu(\alpha_n, k_n)]. \tag{9}$$

$$\underline{\theta} = \{\pi(\alpha, k), \mu(\alpha, k), \Sigma(\alpha, k),\ \alpha = 0, 1,\ k = 1 \ldots K\}, \tag{10}$$

Among them, U is the area item. As mentioned above, you represent the penalty for a pixel being classified as the target or background, that is, the negative logarithm of the probability that a pixel belongs to the target or background. We know that the mixture Gaussian density model is of the form:

$$D(x) = \sum_{i=1}^{K} \pi_i g_i(x; \mu_i, \Sigma_i) \quad , \sum_{i=1}^{K} \pi_i = 1 \text{且} 0 \le \pi_i \le 1$$

$$g(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left[-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right]$$

So after taking the negative logarithm, it becomes the form of formula (9), in which GMM

has three parameters θ: the weight of each Gaussian component, the mean vector u of each Gaussian component (because there are three RGB channels , so it is a three-element vector) and the covariance matrix (because there are three RGB channels, so it is a 3x3 matrix). Such as formula (10). That is to say, all three parameters of the GMM describing the target and the GMM describing the background need to be learned and determined. Once these three parameters are determined, then after we know the RGB color value of a pixel, we can substitute the GMM of the target and the GMM of the background to get the probability that the pixel belongs to the target and the background respectively, that is, the area of Gibbs energy The energy term can be determined, that is, we can find the weight of the t-link of the graph.

$$V(\underline{\alpha}, \mathbf{z}) = \gamma \sum_{(m,n)\in \mathbf{C}} [\alpha_n \neq \alpha_m] \exp -\beta \|z_m - z_n\|^2 . \qquad (11)$$

The boundary term is similar to the Graph Cut mentioned earlier, which reflects the discontinuity penalty between the neighborhood pixels m and n. If the difference between the two neighborhood pixels is small, then it is very likely to belong to the same target or the same background. If they are very different, it means that the two pixels are likely to be at the edge of the target and the background, and they are more likely to be separated, so when the difference between the two neighboring pixels is greater, the energy is smaller. In RGB space, to measure the similarity of two pixels, we use Euclidean distance (two norm). The parameter β here is determined by the contrast of the image. It is conceivable that if the contrast of the image is low, that is to say, for pixels m and n that are different, their difference is still relatively low, then we need to Multiply by a relatively large β to amplify this difference, and for images with high contrast, then perhaps the difference between pixels m and n that belong to the same target is still relatively high, then we need to multiply by A relatively small β narrows the difference so that the V term works fine with high or low contrast. The constant γ is 50 (a good value obtained by the author after training with 15 images). OK, at this time, the weight of n-link can be determined by formula (11). At this time, the graph we want can be obtained, and we can segment it.

2.4 Iterative Energy Minimization Segmentation Algorithm

The algorithm of Graph Cut is one-time minimization, while Grab Cut is the smallest iteration. Each iteration process makes the parameters of the GMM modeling the target and background better, making the image segmentation better. We illustrate directly through the algorithm.

2.4.1 Initialization

1 : The user obtains an initial trimap T by directly selecting the target, that is, the pixels outside the box are all used as background pixels TB, and the pixels in the TU within the box are all used as "possible target" pixels.

2 : For each pixel n in the TB, the label $\alpha n = 0$ of the initialized pixel n is the background pixel; and for each pixel n in the TU, the label $\alpha n = 1$ of the initialized pixel n, that is, as the "possible target". pixel.

3 : After the above two steps, we can get some pixels belonging to the target, and the rest are pixels belonging to the background. At this time, we can use this pixel to estimate the target and background. GMM too. We can cluster the pixels belonging to the target and the background into K categories through the k-mean algorithm, that is, K Gaussian models in the GMM. At this time, each Gaussian model in the GMM has some pixel sample sets. At this time, its The parameter mean and covariance can be estimated by their RGB values, and the weight of the Gaussian component can be determined by the ratio of the number of pixels belonging to the Gaussian component to the total number of pixels.

2.4.2 Iterative Minimization

1 : Assign the Gaussian component in the GMM to each pixel (for example, if pixel n is the target pixel, then substitute the RGB value of pixel n into each Gaussian component in the target GMM, and the one with the highest probability is the one most likely to generate n, that is, the knth Gaussian component of pixel n):

$$k_n := \arg\min_{k_n} D_n(\alpha_n, k_n, \theta, z_n).$$

2 : For a given image data Z, learn to optimize the parameters of the GMM (because in step (1) we have classified which Gaussian component each pixel belongs to, then each Gaussian model has some pixel sample set, At this time, its parameter mean and covariance can be estimated by the RGB values of these pixel samples, and the weight of the Gaussian component can be determined by the ratio of the number of pixels belonging to the Gaussian component to the total number of pixels.):

$$\underline{\theta} := \arg\min_{\underline{\theta}} U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z})$$

3 : Segmentation estimation (through the Gibbs energy term analyzed in 1, build a graph, and find the weights t-link and n-link, and then perform the segmentation through the max flow/min cut algorithm):

$$\min_{\{\alpha_n:\, n\in T_U\}} \min_{\mathbf{k}} \mathbf{E}(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}).$$

4 : Repeat steps (1) to (3) until convergence. After the segmentation of (3), whether each pixel belongs to the target GMM or the background GMM changes, so the kn of each pixel changes, so the GMM also changes, so each iteration will interactively optimize the GMM model and segmentation result. In addition, because the processes of steps (1) to (3) are all processes of decreasing energy, it is guaranteed that the iterative process will converge.

5 : Use border matting to smooth the segmented boundaries and other post-processing.

2.5 User Editing (Interactive)

- Edit: artificially fix some pixels to be target or background pixels, and then perform step (3) in 2.4 again;
- Reoperation: Repeat the entire iterative algorithm. (Optional, in fact this is the undo function of the program or software cutout)

2.6 My Code Implementation

In this part, I will introduce my code implementation through illustrate the functions I have written in detail.

2.6.1 GMM class

I implemented a GMM class using KMeans in sklearn.clunster, the class has 6 member variables, and the member function fit is called when it is initialized. The get_component_score here will be used in the get_probabilty function to calculate the scores of each category, and the get_probabilty function will be used later to calculate the U term in the Gibbs energy.

The function np.einsum I used in get_component_score is Einstein summation convention. This function can easily and efficiently convert a given mathematical expression into a code implementation

```python
class GMM(object):
    def __init__(self, X, n_components=5):
        self.n_components = n_components
        self.n_samples = np.zeros(self.n_components)
        self.n_features = X.shape[1]
        self.means = np.zeros((self.n_components, self.n_features))
        self.full_cov = np.zeros((self.n_components, self.n_features, self.n_features))
        self.coefficients = np.zeros(self.n_components)
        self.fit(X, KMeans(n_clusters=self.n_components, n_init=1).fit(X).labels_)

    def get_component_score(self, X, ci):
        if self.coefficients[ci] > 0:
            multiple = np.einsum('ij,ij->i', (X-self.means[ci]), np.dot(np.linalg.inv(self.full_cov[ci]), (X-self.means[ci]).T).T)
            score = np.exp(-.5 * multiple) / (((2*np.pi)**(1/2))*np.sqrt(np.linalg.det(self.full_cov[ci])))
        else:
            score = np.zeros(X.shape[0])
        return score

    def get_probability(self, X):
        return np.dot(self.coefficients, [self.get_component_score(X, ci) for ci in range(self.n_components)])

    def fit(self, X, labels):
        variance = 0.05
        categories,count = np.unique(labels, return_counts=True)
        self.n_samples[categories] = count
        for ci in categories:
            self.coefficients[ci] = self.n_samples[ci] / np.sum(self.n_samples)
            self.means[ci] = np.mean(X[ci == labels],axis=0)
            if(self.n_samples[ci] <= 1):
                self.full_cov[ci] = 0
            else:
                self.full_cov[ci] = np.cov(X[ci == labels].T)
            det = np.linalg.det(self.full_cov[ci])
            if det <= 0:
                # Adds the noise to avoid singular covariance matrix.
                self.full_cov[ci] += np.eye(self.n_features)*variance
                det = np.linalg.det(self.full_cov[ci])
```

Figure 2: The code of GMM

2.6.2 construct_grabcut_graph

In this function, I construct the grabcut graph. I first constructs the t-links and n-links edges. For the t-links, I link all vertices to the source vertex and sink vertex using zip function. And for the n-links, I use np.arange to generate the indices of vertices. Then, I use zip function to link vertices to each other. Also, I this function, I calculate the Gibbs energy. The calculation just follows the formula in the paper, there are two terms U and V, which is known as the region term and smoothness term, respectively. When calculate U, for probable indices, I use two GMM to calculate the probability, then I can get the U term. The smoothness term V is basically unchanged from the monochrome case except that the contrast term is computed using Euclidean distance in colour space.

```python
# this function is used to construct the grabcut graph
def construct_grabcut_graph(image,mask,source_vertex,sink_vertex,fgd_gmm,bgd_gmm,gamma,rows,cols,left_V,upleft_V,up_V,upright_V):
    # get the indices of each type of mask value
    bgd_indices = np.where(mask.reshape(-1) == 0) # background points indices
    fgd_indices = np.where(mask.reshape(-1) == 1) # foreground points indices
    probable_indices = np.where(np.logical_or(mask.reshape(-1) == 2,mask.reshape(-1) == 3)) # probable background/foreground point

    edges = []
    gibbs_energy = []
    capa_coef = 10*gamma

    # t-links
    edges += zip([source_vertex] * probable_indices[0].size, probable_indices[0])
    edges += zip([sink_vertex] * probable_indices[0].size, probable_indices[0])
    edges += zip([source_vertex] * bgd_indices[0].size, bgd_indices[0])
    edges += zip([sink_vertex] * bgd_indices[0].size, bgd_indices[0])
    edges += zip([source_vertex]*fgd_indices[0].size, fgd_indices[0])
    edges += zip([sink_vertex] * fgd_indices[0].size, fgd_indices[0])

    # n-links
    image_indices = np.arange(rows*cols,dtype=np.uint32).reshape(rows,cols)
    edges += zip(image_indices[:,1:].reshape(-1), image_indices[:,:-1].reshape(-1))
    edges += zip(image_indices[1:,1:].reshape(-1), image_indices[:-1,:-1].reshape(-1))
    edges += zip(image_indices[1:,:].reshape(-1), image_indices[:-1,:].reshape(-1))
    edges += zip(image_indices[1:,:-1].reshape(-1), image_indices[:-1,1:].reshape(-1))
```

```python
# add U to Gibbs energy
D_function_bgd = -np.log(bgd_gmm.get_probability(image.reshape(-1, 3)[probable_indices]))
D_function_fgd = -np.log(fgd_gmm.get_probability(image.reshape(-1, 3)[probable_indices]))
gibbs_energy += D_function_bgd.tolist()
gibbs_energy += D_function_fgd.tolist()
gibbs_energy += [0]*bgd_indices[0].size
gibbs_energy += [capa_coef]*bgd_indices[0].size
gibbs_energy += [capa_coef]*fgd_indices[0].size
gibbs_energy += [0] * fgd_indices[0].size

# add V to Gibbs energy
gibbs_energy += left_V.reshape(-1).tolist()
gibbs_energy += upleft_V.reshape(-1).tolist()
gibbs_energy += up_V.reshape(-1).tolist()
gibbs_energy += upright_V.reshape(-1).tolist()

graph = ig.Graph(cols*rows + 2)
graph.add_edges(edges)
return graph,source_vertex,sink_vertex,gibbs_energy
```

Figure 3: The code of construct_grabcut_graph

2.6.3 classify

In the classify function, I implement the edge segmentation, I use the st_mincut method for ig.Graph to solve the mincut problem. After that, I can get the indices of background and foreground respectively.

```
def classify(mask,graph,source_vertex,sink_vertex,gibbs_energy,rows,cols):
    mincut = graph.st_mincut(source_vertex,sink_vertex, gibbs_energy)
    probable_indices = np.where(np.logical_or(mask == 2, mask == 3))
    image_indices = np.arange(rows * cols,dtype=np.uint32).reshape(rows, cols)
    mask[probable_indices] = np.where(np.isin(image_indices[probable_indices], mincut.partition[0]),3,2)
    bgd_indices = np.where(np.logical_or(mask == 0,mask == 2))
    fgd_indices = np.where(np.logical_or(mask == 1,mask == 3))
    return mask,bgd_indices,fgd_indices
```

Figure 4: The code of classify

2.6.4 GrabCut

In this function, I read the image, and set the initial mask (it is all zeros now. For mask, 0 means background; 1 means foreground; 2 means probable background; 3 means probable foreground). Then, I get the region user chooses, change the mask value of pixels in the region as 3 (they are probable foreground). And then, get the foreground the user point out, change these pixels' mask value as 1 (user choose these as determined foreground). Also, get the background the user point out, change these pixels' mask value as 1 (user choose these as determined background). Then, I get the distance in the four neighbors (left, upleft, up, upright). After that, I compute the beta value and the smoothness term. In addition, I invoke the above mentioned functions to get the final image.

```
def GrabCut(filename,foreground=[],background=[], pos1x=1,pos1y=1,pos2x=511,pos2y=511):
    image = cv.imread(filename)
    image = np.asarray(image, dtype=np.float64)
    rows,cols, _ = image.shape
    mask = np.zeros(image.shape[:2], dtype=np.uint8)

    mask[pos1y:pos1y+pos2y, pos1x:pos1x+pos2x] = 3

    for y1,x1,y2,x2 in foreground:
        if x1==x2:
            mask[x1,min(y1,y2):max(y1,y2)+1] = 1
        else:
            k = (y1-y2)/(x1-x2)
            x,y = min_x(x1,x2,y1,y2)
            while True:
                mask[x,y] = 1
                x = x+1
                y = int(round(y+k))
                if x>max(x1,x2):
                    break

    for y1,x1,y2,x2 in background:
        if x1==x2:
                mask[x1,min(y1,y2):max(y1,y2)+1] = 0
        else:
            k = (y1-y2)/(x1-x2)
            x,y = min_x(x1,x2,y1,y2)
            while True:
                mask[x,y] = 0
                x = x+1
                y = int(round(y+k))
                if x>max(x1,x2):
                    break
```

```python
def GrabCut(filename,foreground=[],background=[], pos1x=1,pos1y=1,pos2x=511,pos2y=511):
    image = cv.imread(filename)
    image = np.asarray(image, dtype=np.float64)
    rows,cols, _ = image.shape
    mask = np.zeros(image.shape[:2], dtype=np.uint8)

    mask[pos1y:pos1y+pos2y, pos1x:pos1x+pos2x] = 3

    for y1,x1,y2,x2 in foreground:
        if x1==x2:
            mask[x1,min(y1,y2):max(y1,y2)+1] = 1
        else:
            k = (y1-y2)/(x1-x2)
            x,y = min_x(x1,x2,y1,y2)
            while True:
                mask[x,y] = 1
                x = x+1
                y = int(round(y+k))
                if x>max(x1,x2):
                    break

    for y1,x1,y2,x2 in background:
        if x1==x2:
                mask[x1,min(y1,y2):max(y1,y2)+1] = 0
        else:
            k = (y1-y2)/(x1-x2)
            x,y = min_x(x1,x2,y1,y2)
            while True:
                mask[x,y] = 0
                x = x+1
                y = int(round(y+k))
                if x>max(x1,x2):
                    break
```

Figure 5: The code of GrabCut

2.6.5 main.py

I also write a main.py file to implement the user interface. The implementation is based on the wxpython. And the main code can be generated by wxFormBuilder. The code is too large to show here. I can just show a part of the code. You'd better see the complete version in my source code.

```python
#-----------------------------------------------------------------------------------------
class GrabFrame(wx.Frame):
#   this class builds the frame of the application,which
#   consists of a doodle window and a control panel.
    def __init__(self, parent):
        wx.Frame.__init__(self, parent, -1, "Xiang Fei's GrabCut", size=(638,512),
                          style=wx.DEFAULT_FRAME_STYLE | wx.NO_FULL_REPAINT_ON_RESIZE)

        self.doodle = DoodleWindow(self, -1)            #build a doodle window object
        cPanel = ControlPanel(self, -1, self.doodle)    #build a control panel object

        #set the layout of the two objects above
        box = wx.BoxSizer(wx.HORIZONTAL)
        box.Add(cPanel, 0, wx.EXPAND)
        box.Add(self.doodle, 1, wx.EXPAND)

        self.SetSizer(box)
        self.Centre()#to make the UI on the centre of the screen
#-----------------------------------------------------------------------------------------
class GrabApp(wx.App):
#   this class builds the whole application.
    def OnInit(self):
        frame = GrabFrame(None)
        frame.Show(True)
        return True
#-----------------------------------------------------------------------------------------
if __name__ == '__main__':
    app = GrabApp()
    app.MainLoop()
```

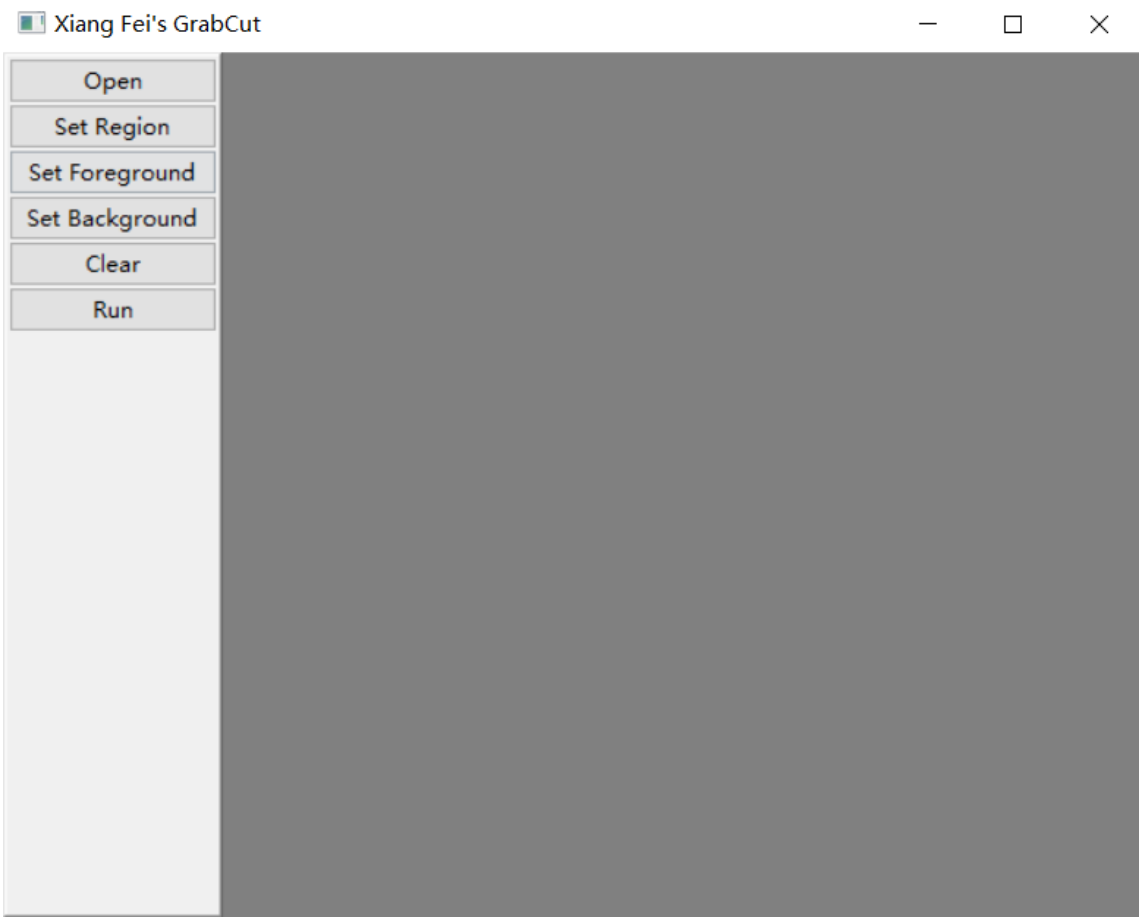
Figure 6: The code of user interface

3 Execution

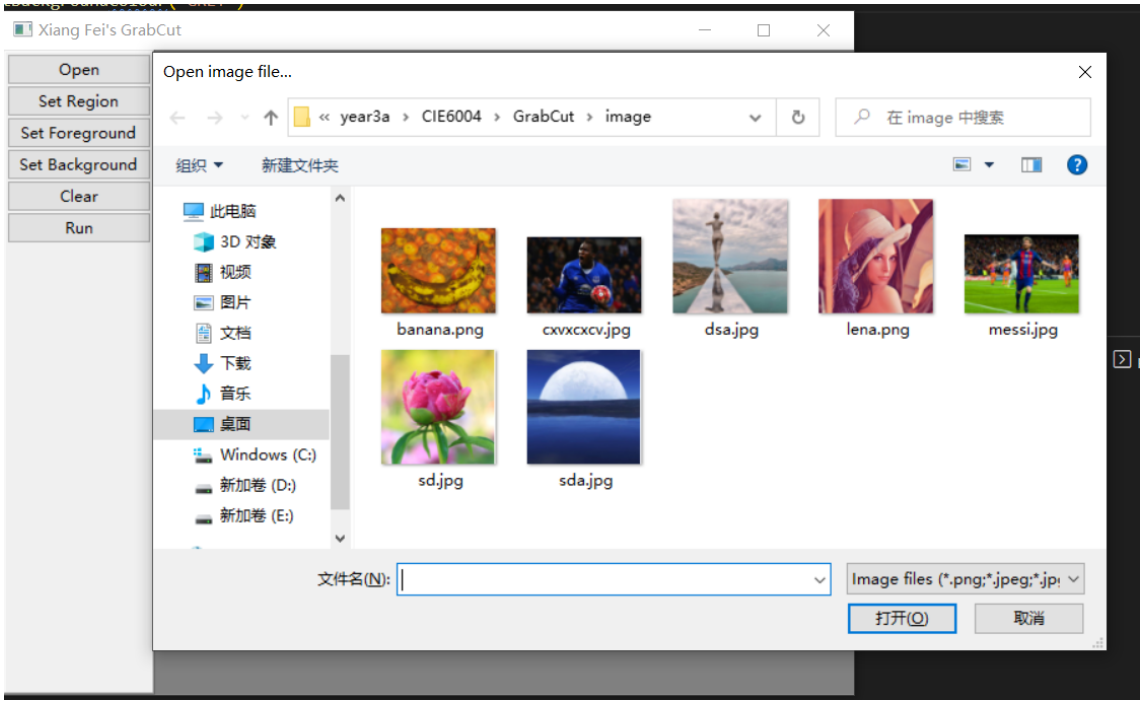
grabcut.py and main.py should be placed in the same folder.

```
1 python main.py
```

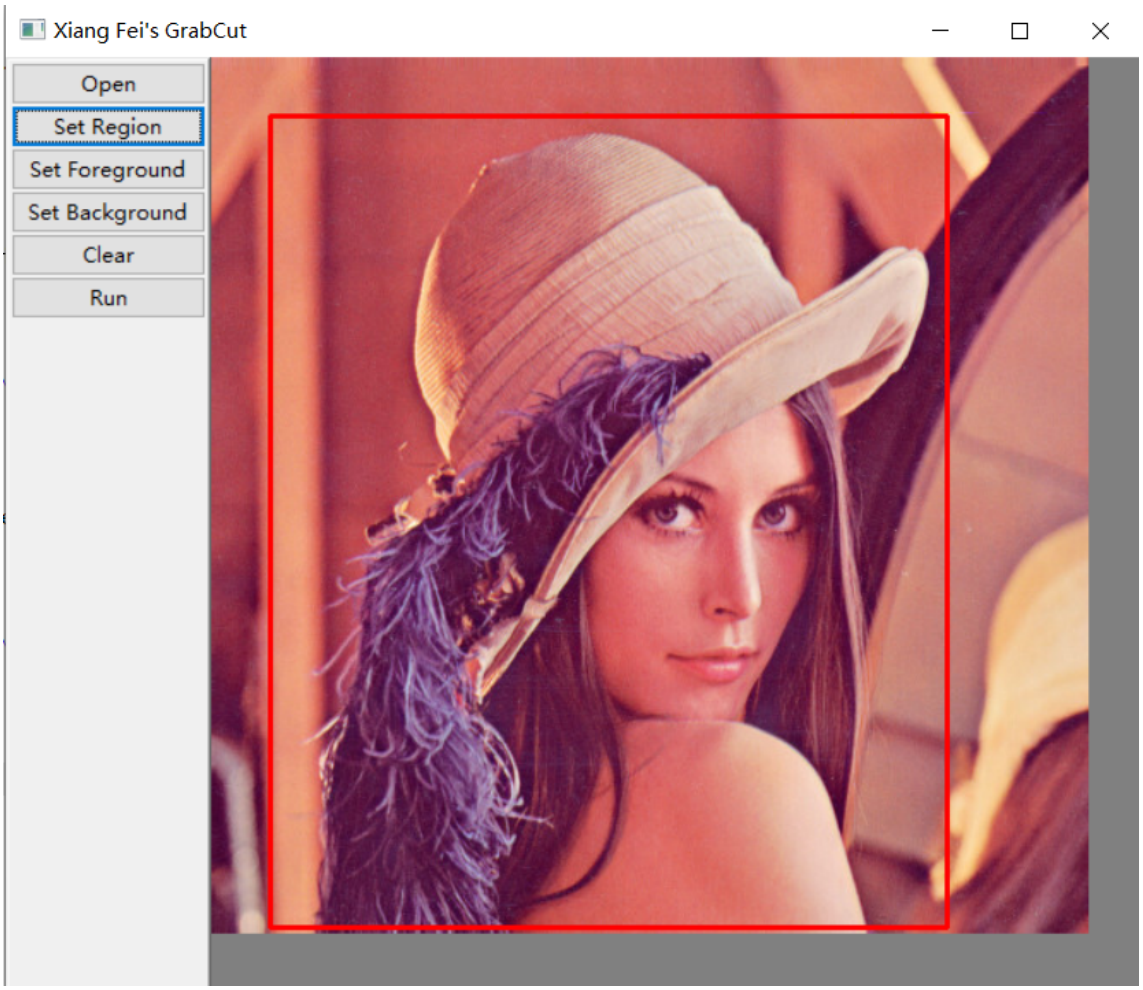
Then, you can get the user interface like the follows.



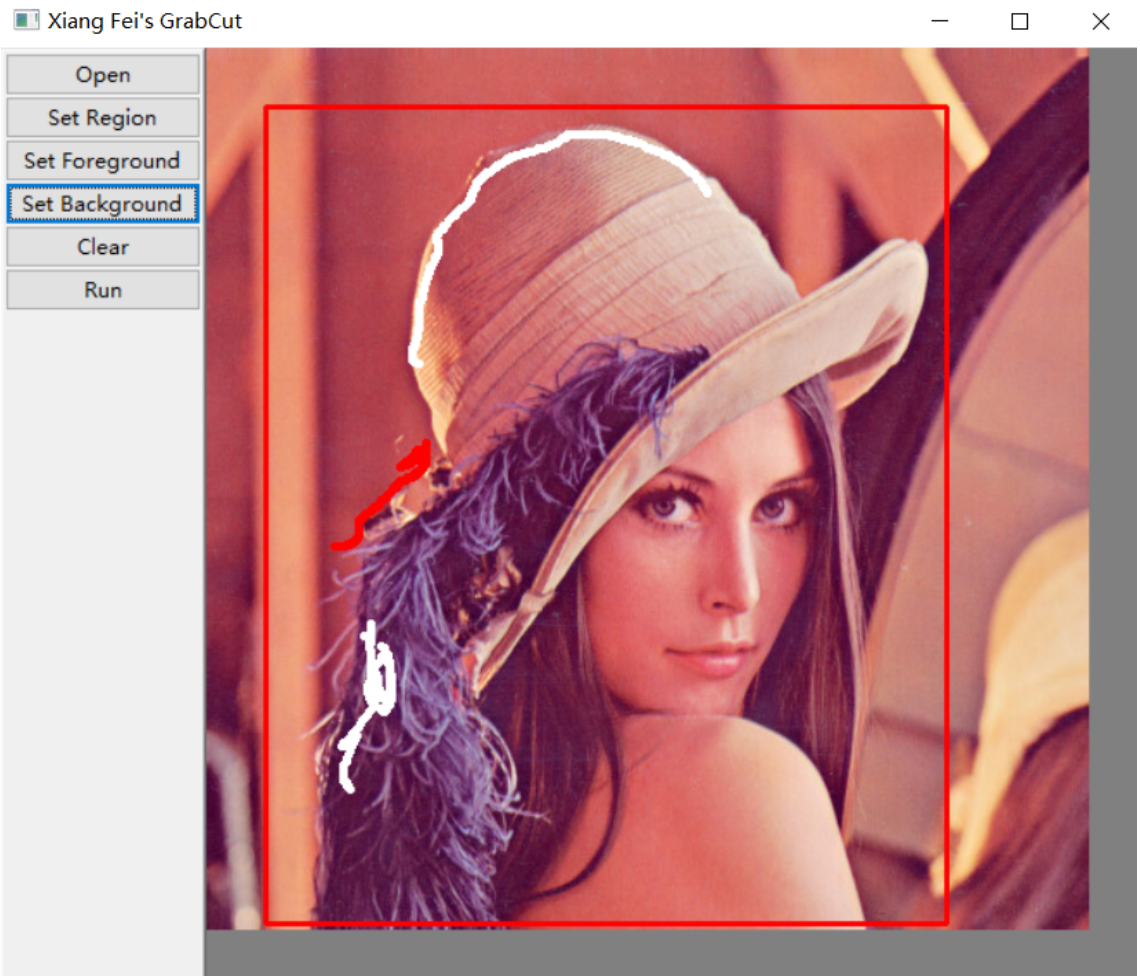
Here, the "Open" button is used to choose an image, like the follows.



Then, you can use "Set Region" to drags a rectangle loosely around the object. Like the follows, the red rectangle is the region I choose.



Also, you can use "Set Foreground" (white pencil) and "Set Background" (red pencil) to choose the foreground and background pixels, respectively. And you can use "Clear" to cancel the foreground and background settings.

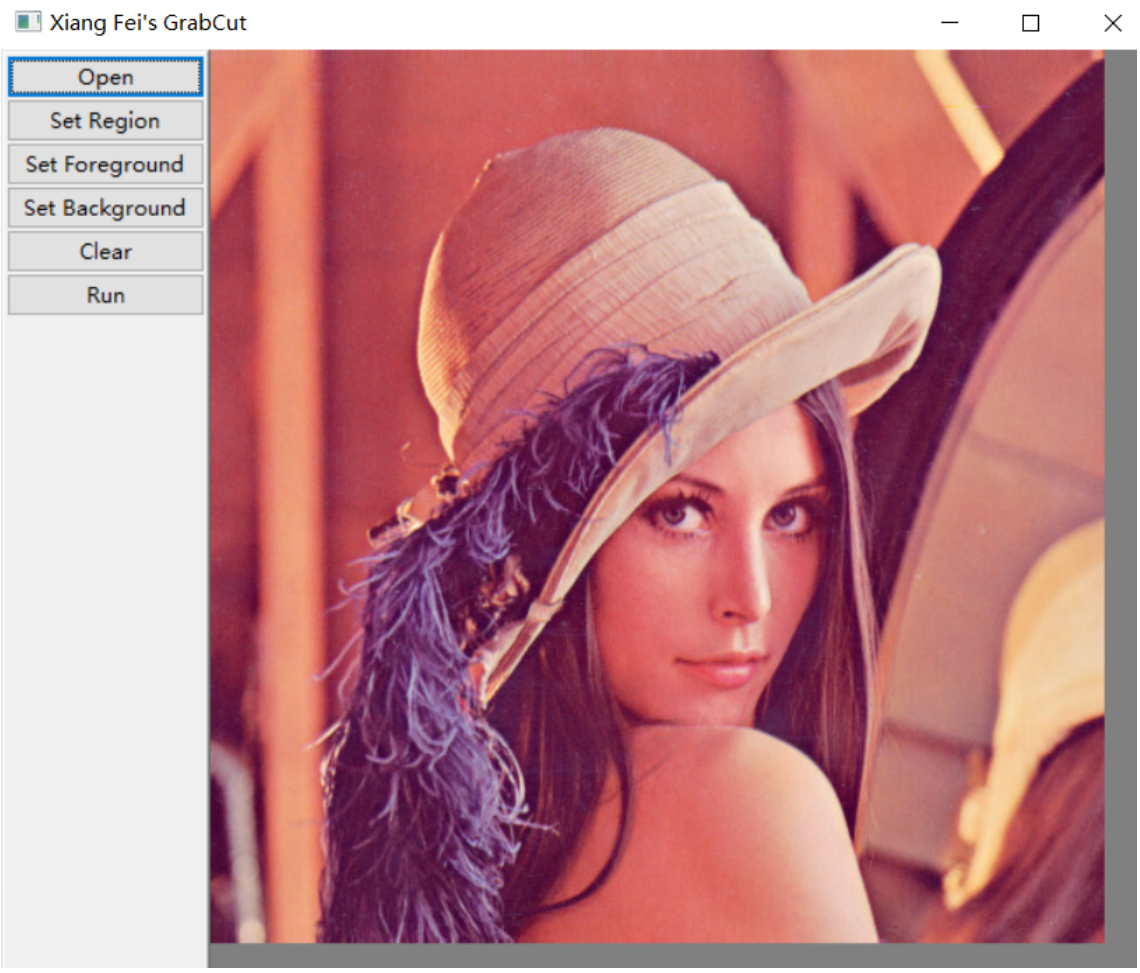


Last, use "Run" to get the final output. My example is like the follows.

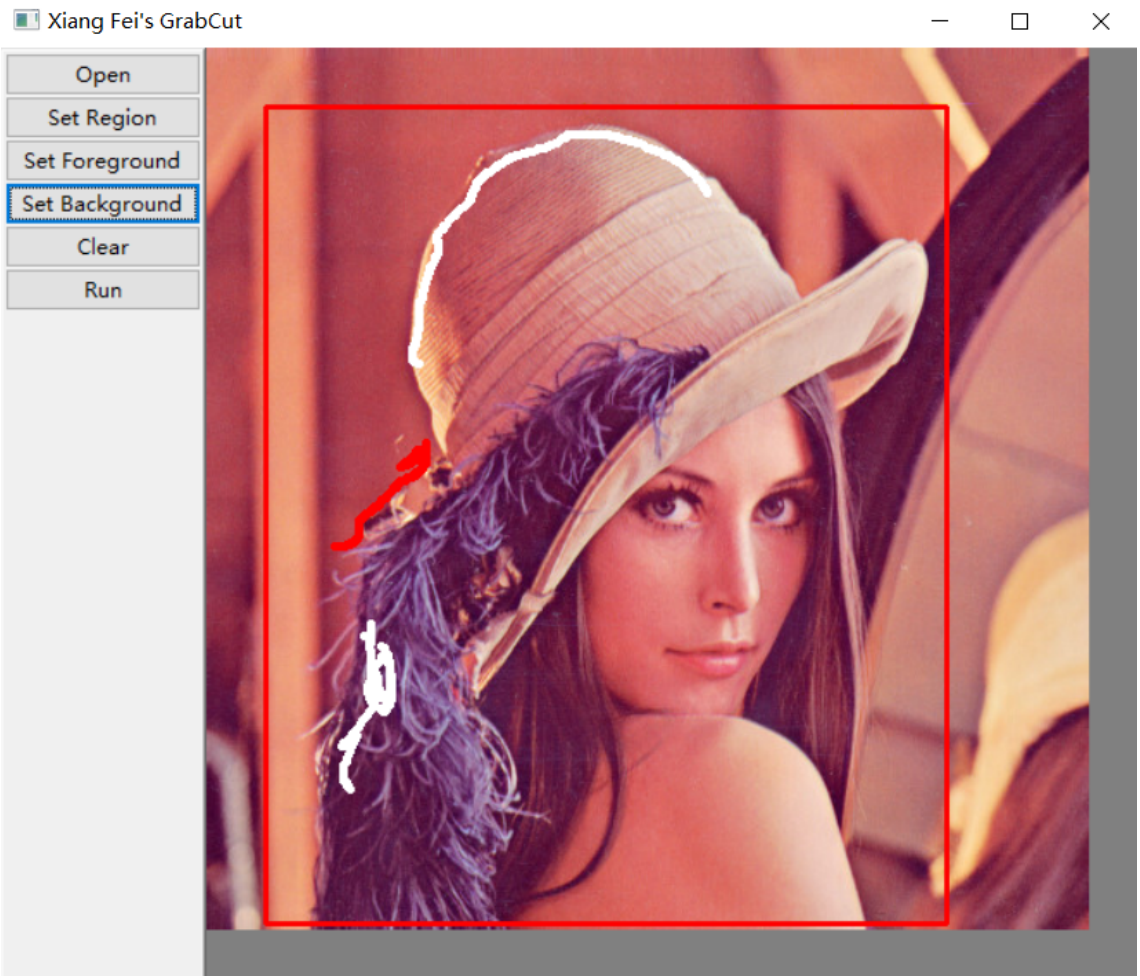
4 Result

4.1 test 1

original image:



rectangle and bgd/fgd settings:



output result:

4.2 test 2

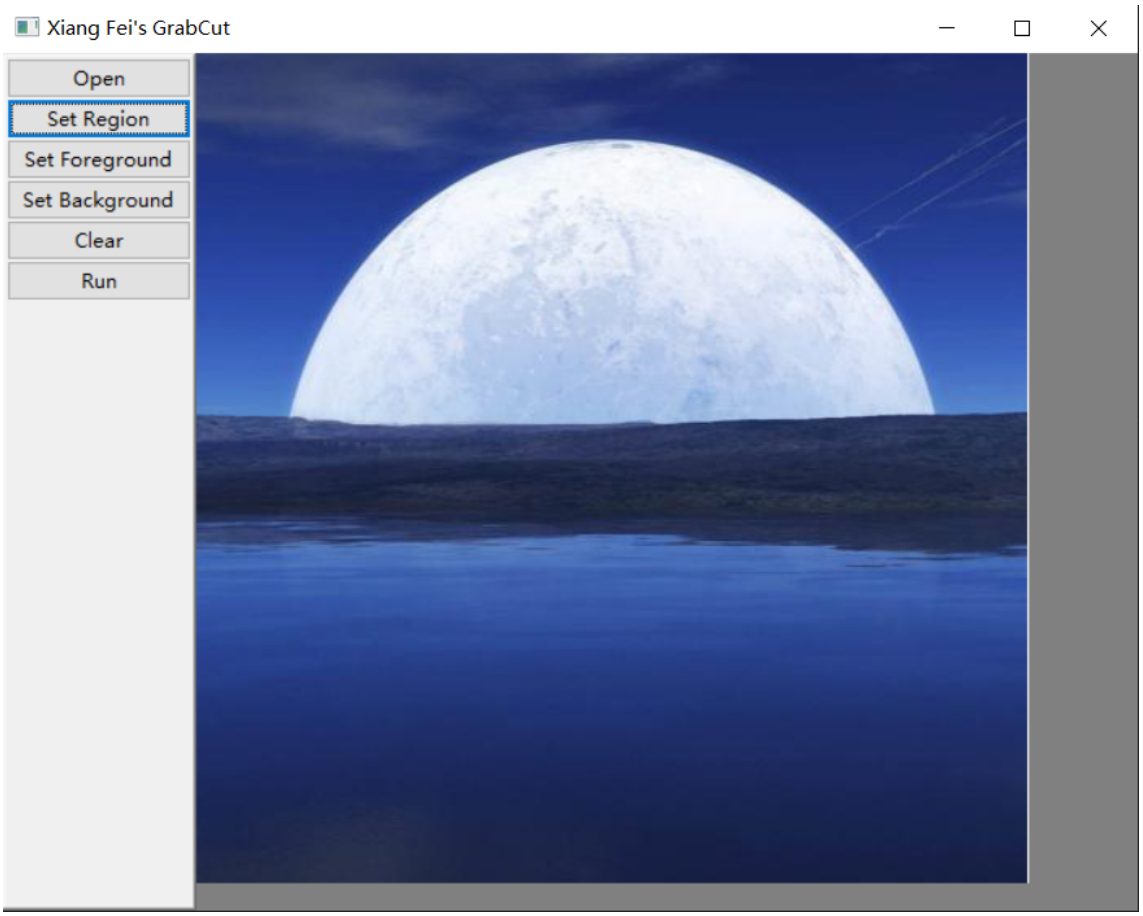
original image:

rectangle and bgd/fgd settings:

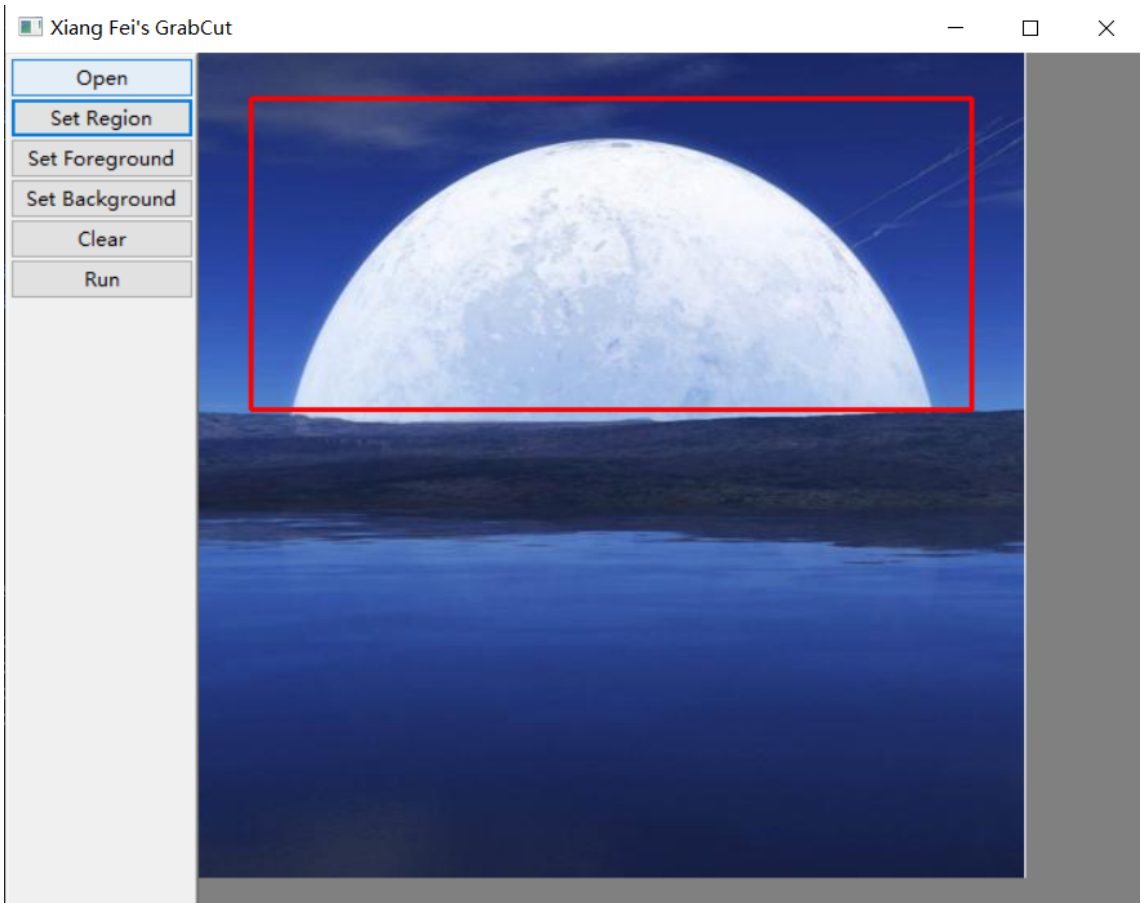
output result:

4.3 test 3

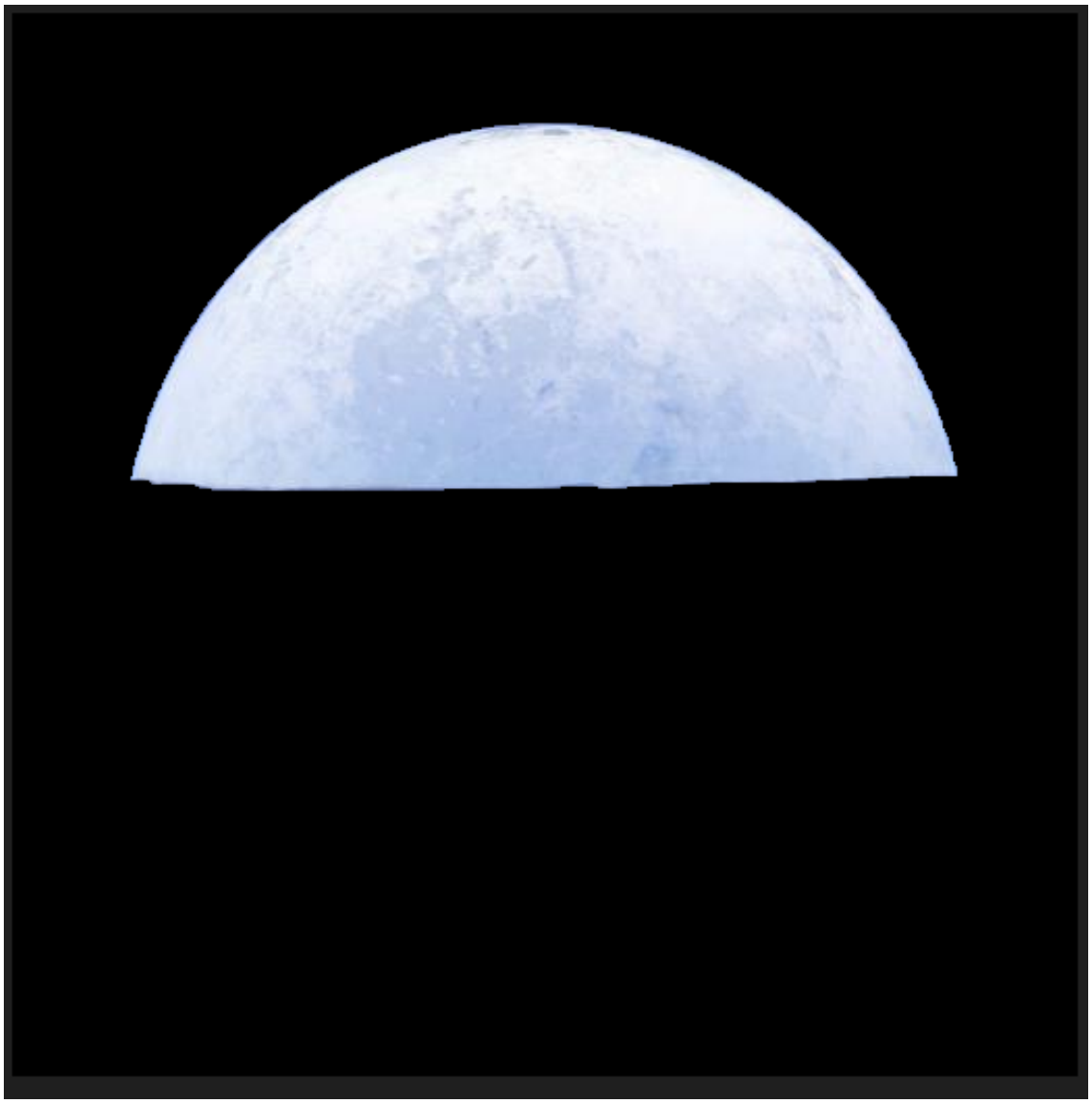
original image:



rectangle and bgd/fgd settings:

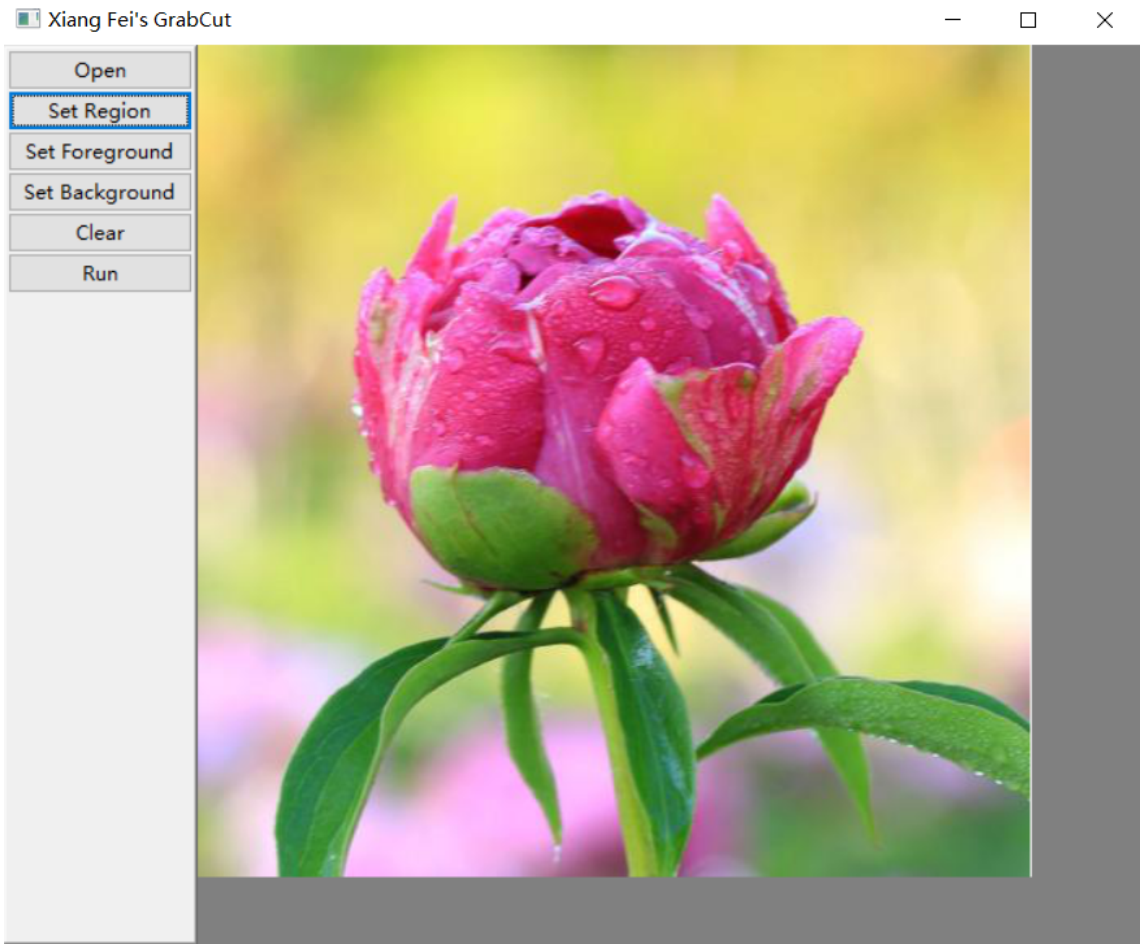


output result:

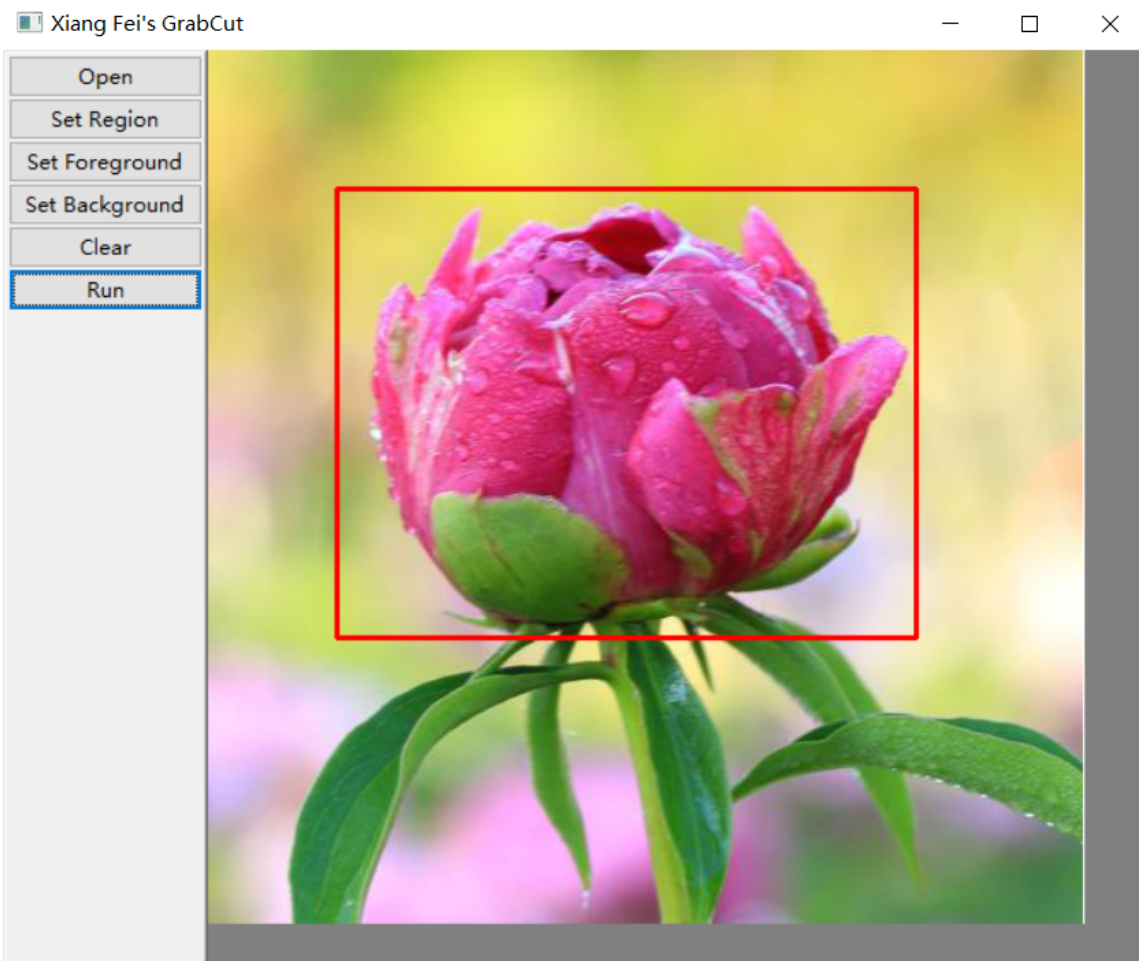


4.4 test 4

original image:



rectangle and bgd/fgd settings:

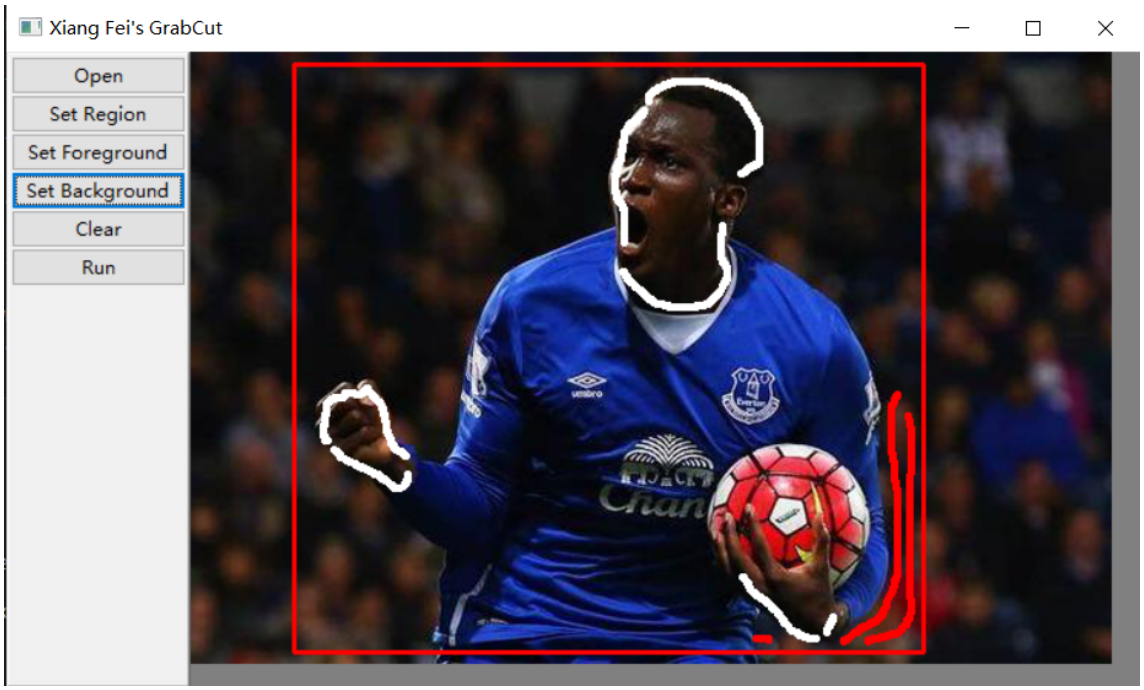


output result:

4.5 test 5

original image:

rectangle and bgd/fgd settings:



output result:

5 My feeling

As an undergraduate, this was my first exposure to the field of foreground/background segmentation, and I thought GrabCut was really a very elegant approach. I feel that I have gained a lot from the whole process this time.

First of all, after my study, I think that the essence of GrabCut here is to Using the texture (color) information and boundary (contrast) information in the image to solve a mincut optimization problem, which I think is very important for my understanding. And I also read the given paper very seriously, I think this is a very good exercise for me, and it has cultivated my ability to study independently through the paper. This time I also implemented GrabCut from a very low level. I think it is very helpful for me to clearly understand the problem and my ability to code.

However, I also encountered many difficulties in this study. First of all, reading a paper is not an easy process. It is difficult for me to understand this problem very well by relying on the full English paper, so I also found a lot of relevant information on the Internet, which is very helpful for me to understand this problem. Secondly, I didn't understand the method of solving gibbs energy in the paper very well at first, but by looking up some github source code, some c++ implementations and some online blogs, I understood how to formulate the gibbs energy using code. I also encountered a lot of difficulties in the implementation of the code, because I am not familiar with the commonly used interfaces in the field of computer

vision such as igraph, which makes me need to check the information frequently. And for the implementation of user interface, it's also a big challenge for me. By looking up various ways to implement user interface on the Internet, I finally learned to use wxFormBuilder to efficiently generate relevant code and embed it into my program.

All in all, I learned a lot through this homework!!!

Bibliography

Rother, Carsten, Vladimir Kolmogorov and Andrew Blake (Aug. 2004). '"GrabCut": Interactive Foreground Extraction Using Iterated Graph Cuts'. In: *ACM Trans. Graph.* 23.3, pp. 309–314. ISSN: 0730-0301. DOI: 10.1145/1015706.1015720. URL: https://doi.org/10.1145/1015706.1015720.