User-constrained Natural Image Matting

Chengxian Liu, Qiao Chen
School of Computing Science
Simon Fraser University
{cla284, qiaoc}@sfu.ca

Abstract

In this project, we examine the theoretical basis of Closed Form matting algorithm (Levin et al., 2006).

We provide detailed proof to the derivation of the paper's mathematical model, which is not thoroughly explained. We also implement the core functions of the paper and point out some possibly future work.

1 Introduction

A natural image can be constituted by foreground and background. In mathematical representation, a 2-D image I can be regarded as the composite of a foreground image F and a background image B. In this way, the color of each pixel is equal to the linear combination of corressponding foreground and background:

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i \quad (1)$$

where α represents the opacity of the foreground image. For pixels on the foreground object, the value of α equals to 1. For pixels on the background object, the value of α equals to 0. For the pixel on the boundary of foreground and background, the α can be fractional. Image matting algorithm try to find the optimal α to separate the foreground object from the background.

However, image matting is an under-constrained problem. For each pixel, we need to estimate the foreground, the background and the opacity of the foreground. To solve this severly under-constrained problem, firstly we assume the image conforms to the color line model, which will be discussed in section 2.2.

And user interaction is necessary to perform image matting. Take Rubins vase (Figure 1) as an example. There are two possible interpretations of the image. The white part of the image can be considered both as the foreground (the vase) or the background (the wall behind two opposite profiles). The detial of user interaction will be discussed in section 2.3.

Figure 1: Rubins Vase

1.1 Motivation

Natural image matting algorithm solves the problem of accurate foreground estimation in an image. As an important and useful technique, it is already widely used in many area like image editing and film production. An effective and accurate matting algorithm can save the manpower cost and improve work efficiency to a large extent.

At the same time, with the popularity of smart phones, nowadays people take a lot of photos in their daily life, which enhances the demand for efficiency image editing algorithm. If a matting algorithm can be implemented efficiently enough to run on the mobile device, the matting feature will very likely to be one of the most important functions in camera.

Thus, we try to find a matting algorithm which can accurately separate the foreground object from the backgound. At the same time, it will be better if the algorihms only need a little user interaction and can be run on a mobile device in a reasonable time.

1.2 Related work

The matting algorihms in (Boykov and Jolly, 2001) and (Li et al., 2004) translate matting problem into image segmentation problem. By constructing an undirected graph and calculate its minmum cut, it decides whether a pixel belongs to the background or the foreground. Since a fractional alpha will describe the foreground opacity better, the disadvantage of this kind of algorithm is that the alpha it generates is only either 0 or 1.

Both of the poisson matting method (Sun et al., 2004) and Bayesian matting algorithm (Chuang et al., 2001) make use of trimap to perform matting, which labels each pixel as foreground, background, or unknown. Poisson matting method assumes that the matte gradient can be approximated as $\nabla I(F - B)$. However, this assumption is too strong and leads to some local error.

The Bayesian matting algorithm perform the most successful matting among the methods described above, which uses mixture of oriented Gaussian and then α, a, b are estimated given that distribution. The algorithm performs well when there are little overlap betwen the color distribution on foreground and background, and the unknown area is small. Otherwise it might lead to a erroneous matte as well.

In this project, we apply the closed form matting (Levin et al., 2006). The method firstly derives a cost function with color line model, and eliminate F and B from the cost function, leaving α as the only variable. The algorithm calculates the optimal alpha matte that minimizes the cost function. With small amount of user's scribbles on natural image as input, high quality α can be generated.

2 Derivation

2.1 Grayscale Image

In the paper, Levin assumes that foreground image and background image are steady inside small local windows (3 by 3 in his case). Therefore, he gives the following mathematical assumption:

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i$$
$$\Rightarrow \alpha \approx aI_i + b, \forall i \in w$$
$$a = \frac{1}{F - B}, b = \frac{-B}{F - B}$$

Levin also defines the cost function J given α, a, b. Intuitively, the cost function measures the error of the above approximation of α. The bias term ε is used for numerical stability.

$$J(\alpha, a, b) = \sum_{j \in I} (\sum_{i \in w_j} (\alpha_i - a_j I_i - b_j)^2 + \varepsilon a_j^2)$$

The goal of this paper is to find the optimal (α, a, b) to minimize the cost function J.

Theorem1: Define $J(\alpha)$ as

$$J(\alpha) = \min_{a,b} J(\alpha, a, b)$$

Then

$$J(\alpha) = \alpha^T L \alpha$$

where $L = \bar{G_k}^T \bar{G_k}$, (i, j)-th entry of $\bar{G_k}$ is:

$$\sum_{k|(i,j) \in w_k} (\delta_{ij} - \frac{1}{|w_k|}(1 + \frac{1}{\frac{\varepsilon}{|w_k|} + \sigma_k^2}(I_i - \mu_k)(I_j - \mu_k)))$$

Here δ_{ij} is the Kronecker delta, μ_k is the mean of the intensities in the window w_k nearby k, σ_k^2 is the variance of the intensities in the window w_k nearby k, and $|w_k|$ is the number of pixels in the current window.

Proof of Theorem1:

Use G_k and $\bar{\alpha_k}$ to denote $J(\alpha)$, where

$$G_k^T = \begin{bmatrix} I_1 & I_2 & I_3 & ... & I_{w_k} & \sqrt{\varepsilon} \\ 1 & 1 & 1 & ... & 1 & 0 \end{bmatrix}$$

$$\bar{\alpha_k}^T = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & ... & \alpha_{w_k} & 0 \end{bmatrix}$$

$$\sum_k ||G_k \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha_k}||^2$$

$$= \sum_k || \begin{bmatrix} I_1 a_k + b_k \\ I_2 a_k + b_k \\ \vdots \\ I_{|w_k|} a_k + b_k \\ \sqrt{\varepsilon} a_k \end{bmatrix} - \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_{w_k} \\ 0 \end{bmatrix} ||^2$$

$$= \sum_k (\sum_{i=1}^{|w_k|} (\alpha_i - a_k I_i - b_k)^2 + \varepsilon a_k^2)$$

Meanwhile

$$J(\alpha, a, b) = \sum_{j \in I} (\sum_{i \in w_j} (\alpha_i - a_j I_i - b_j)^2 + \varepsilon a_j^2)$$

Thus

$$J(\alpha, a, b) = \sum_k ||G_k \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha_k}||^2 \quad (2)$$

After rewriting $J(\alpha, a, b)$, for each window, we try to find the (a_k^*, b_k^*) that minimizes $J(\alpha)$, given a specific α. Using the solution to the least quares problem we can get:

$$(a_k^*, b_k^*) = argmin ||G_k \begin{bmatrix} a_k \\ b_k \end{bmatrix} \bar{\alpha_k}|| = (G_k^T G_k)^{-1} G_k^T \bar{\alpha_k} \quad (3)$$

Substituting (3) back to (2), we get:

$$J(\alpha, a, b) = \sum_k ||G_k \begin{bmatrix} a_k \\ b_k \end{bmatrix} - \bar{\alpha_k}||^2$$

$$= \sum_k ||G_k(G_k^T G_k)^{-1} G_k^T \bar{\alpha_k} - \bar{\alpha_k}||^2$$

Let $\bar{G_k} = I - G_k(G_k^T G_k)^{-1} G_k^T$, we get

$$\sum_k ||G_k(G_k^T G_k)^{-1} G_k^T \bar{\alpha_k} - \bar{\alpha_k}||^2$$

$$= \sum_k ||(I - \bar{G_k})\bar{\alpha_k} - \bar{\alpha_k}||^2$$

$$= \sum_k ||\bar{G_k}\bar{\alpha_k}||^2 = \sum_k (\bar{G_k}\bar{\alpha_k})^T \bar{G_k}\bar{\alpha_k}$$

$$\sum_k \bar{\alpha_k}^T \bar{G_k}^T \bar{G_k}\bar{\alpha_k}$$

Thus we have

$$J(\alpha) = \sum_k \bar{\alpha_k}^T \bar{G_k}^T \bar{G_k}\bar{\alpha_k}$$

Then we try to find out the value of $\bar{G_k}$, since $\bar{G_k} = I - G_k(G_k^T G_k)^{-1} G_k^T$, firstly we calculate $G_k^T G_k$:

$$G_k^T G_k = \begin{bmatrix} I_1 & I_2 & I_3 & ... & I_{w_k} & \sqrt{\varepsilon} \\ 1 & 1 & 1 & ... & 1 & 0 \end{bmatrix} \begin{bmatrix} I_1 & 1 \\ I_2 & 1 \\ I_3 & 1 \\ \vdots & \vdots \\ I_{w_k} & 1 \\ \sqrt{\varepsilon} & 0 \end{bmatrix}$$

$$= \begin{bmatrix} \sum_1^{|w_k|} I_i^2 + \varepsilon & \sum I_i \\ \sum I_i & |w_k| \end{bmatrix}$$

Secondly, with the formula to calculate the inverse of a 2 by 2 matrix, we can get $(G_k^T G_k)^{-1}$:

$$(G_k^T G_k)^{-1} = \frac{\begin{bmatrix} |w_k| & \sum -I_i \\ \sum -I_i & \sum_1^{|w_k|} I_i^2 + \varepsilon \end{bmatrix}}{|w_k|(\sum I_i^2 + \varepsilon) - (\sum I_i)^2}$$

Thirdly, we calcualte the $G_k(G_k^T G_k)^{-1}$:

$$G_k(G_k^T G_k)^{-1} = \frac{\begin{bmatrix} I_1 & 1 \\ I_2 & 1 \\ \vdots & \vdots \\ I_{w_k} & 1 \\ \sqrt{\varepsilon} & 0 \end{bmatrix} \begin{bmatrix} |w_k| & \sum -I_i \\ \sum -I_i & \sum I_i^2 + \varepsilon \end{bmatrix}}{|w_k|(\sum I_i^2 + \varepsilon) - (\sum I_i)^2}$$

$$= \frac{\begin{bmatrix} I_1 w_k - \sum I_i & -I_1 \sum I_i + \sum I_i^2 + \varepsilon \\ I_2 w_k - \sum I_i & -I_2 \sum I_i + \sum I_i^2 + \varepsilon \\ \cdots & \cdots \\ I_{|w_k|} w_k - \sum I_i & -I_{|w_k|} \sum I_i + \sum I_i^2 + \varepsilon \\ \sqrt{\varepsilon}|w_k| & -\sqrt{\varepsilon} \sum I_i \end{bmatrix}}{|w_k|(\sum I_i^2 + \varepsilon) - \sum I_i^2}$$

Fourthly, we calcualte $G_k(G_k^T G_k)^{-1} G_k^T$:

$$G_k(G_k^T G_k)^{-1} G_k^T = G_k(G_k^T G_k)^{-1} \begin{bmatrix} I_1 & I_2 & ... & I_{w_k} & \sqrt{\varepsilon} \\ 1 & 1 & ... & 1 & 0 \end{bmatrix}$$

When $i \leq p, j \leq q$, we have

$$[G_k(G_k^T G_k)^{-1} G_k^T]_{i,j}$$

$$\frac{(I_i|w_k| - \sum I_i) * I_j - I_i \sum I_i + \sum I_i^2 + \varepsilon}{|w_k|(\sum_1^{|w_k|} I_i^2 + \varepsilon) - (\sum I_i)^2}$$

$$= \frac{I_i I_j |w_k| - (I_j + I_i) \sum I_i + \sum I_i^2 + \varepsilon}{|w_k|(\sum_1^{|w_k|} I_i^2 + \varepsilon) - (\sum I_i)^2}$$

Since μ_k and σ_k^2 are the mean and the variance of the intensities in the window w_k nearby k, we have:

$$\mu_k = \frac{1}{|w_k|} \sum I_i \Rightarrow \sum I_i = |w_k|\mu_k$$

$$\sigma_k^2 = E(x^2) - E(x)^2 = \frac{1}{|w_k|} \sum I_i^2 - \mu_k^2$$

$$\Rightarrow \sum I_i^2 = (\sigma_k^2 + \mu_k^2)|w_k|$$

Substituting the $\sum I_i^2$ and $\sum I_i$ with μ_k and σ_k^2, we get $[G_k(G_k^T G_k)^{-1} G_k^T]_{i,j}$

$$= \frac{I_i I_j |w_k| - (I_j + I_i)|w_k|\mu_k + (\sigma_k^2 + \mu_k^2)|w_k| + \varepsilon}{(\sigma_k^2 + \mu_k^2)|w_k|^2 + \varepsilon|w_k| - |w_k|^2 \mu_k^2}$$

$$= \frac{I_i I_j |w_k| - (I_j + I_i)|w_k|\mu_k + |w_k|\mu_k^2 + |w_k|\sigma_k^2 + \varepsilon}{\sigma_k^2 |w_k|^2 + \varepsilon|w_k|}$$

$$= \frac{I_i I_j - (I_j + I_i)\mu_k + \mu_k^2}{\sigma_k^2 |w_k| + \varepsilon} + \frac{|w_k|\sigma_k^2 + \varepsilon}{\sigma_k^2 |w_k|^2 + \varepsilon|w_k|}$$

$$= \frac{1}{|w_k|}(1 + \frac{I_i I_j - (I_j + I_i)\mu_k + \mu_k^2}{\sigma_k^2 + \frac{\varepsilon}{|w_k|}})$$

$$= \frac{1}{|w_k|}(1 + \frac{(I_i - \mu_k)(I_j - \mu_k)}{\sigma_k^2 + \frac{\varepsilon}{|w_k|}})$$

$$= \frac{1}{|w_k|}(1 + \frac{1}{\frac{\varepsilon}{|w_k|} + \sigma_k^2}(I_i - \mu_k)(I_j - \mu_k))$$

Since $\bar{G_k} = I - G_k(G_k^T G_k)^{-1} G_k^T$, we have:

$$[\bar{G_k}]_{i,j} = [I - G_k(G_k^T G_k)^{-1} G_k^T]_{i,j}$$

$$= [I]_{i,j} - [G_k(G_k^T G_k)^{-1} G_k^T]_{i,j}$$

$$= \delta_{ij} - \frac{1}{|w_k|}(1 + \frac{1}{\frac{\varepsilon}{|w_k|} + \sigma_k^2}(I_i - \mu_k)(I_j - \mu_k))$$

Therefore, we can denote $J(\alpha) = \alpha^T L \alpha$, where $L = \bar{G_k}^T \bar{G_k}$ and

$$[\bar{G_k}]_{i,j} = \delta_{ij} - \frac{1}{|w_k|}(1 + \frac{1}{\frac{\varepsilon}{|w_k|} + \sigma_k^2}(I_i - \mu_k)(I_j - \mu_k))$$

2.2 Color Image

To prove that $J(\alpha)$ applies to the color images as well, Levin uses color line model (Omer and Werman, 2004) to represent the color image.

Color line model assumes that the foreground color and the background color inside a small window is an interpolation of two color points, i.e.,

$$F_i = \beta_i^F F_1 + (1 - \beta_i^F) F_2 \tag{4}$$

$$B_i = \beta_i^B B_1 + (1 - \beta_i^B) B_2 \tag{5}$$

Where β_i is a constant factor. F_1, B_1 and F_2, B_2 are the foreground color and background color of two different points.

Using color line model, Levin claims that α_i can be represented as:

$$\alpha_i = \sum_c a^c I_i^c + b, \forall i \in w$$

Proof:
Substitude (4) and (5) into $I_i^c = \alpha_i F_i^c + (1 - \alpha_i) B_i^c$:

$$\Rightarrow I_i^c = \alpha_i(\beta_i^F F_1^c + (1 - \beta_i^F) F_2^c)$$

$$+(1 - \alpha_i)(\beta_i^B B_1^c + (1 - \beta_i^B) B_2^c)$$

$$\Rightarrow I_i^c - B_2^c = \alpha_i(\beta_i^F F_1^c + (1 - \beta_i^F) F_2^c)$$

$$+(1 - \alpha_i)(\beta_i^B B_1^c + (1 - \beta_i^B) B_2^c) - B_2^c$$

$$= (F_2^c - B_2^c)\alpha_i + (F_1^c - F_2^c)\alpha_i \beta_i^F + (B_1^c - B_2^c)(1 - \alpha_i)$$

$$= \begin{bmatrix} F_2^c - B_2^c & F_1^c - F_2^c & B_1^c - B_2^c \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_i \beta_i^F \\ (1 - \alpha_i)\beta_i^B \end{bmatrix}$$

Let

$$H = \begin{bmatrix} F_2^R - B_2^R & F_1^R - F_2^R & B_1^R - B_2^R \\ F_2^G - B_2^G & F_1^G - F_2^G & B_1^G - B_2^G \\ F_2^B - B_2^B & F_1^B - F_2^B & B_1^B - B_2^B \end{bmatrix}$$

Then from above we know:

$$H \begin{bmatrix} \alpha_i \\ \alpha_i \beta_i^F \\ (1 - \alpha_i)\beta_i^B \end{bmatrix} = \begin{bmatrix} I_i^R \\ I_i^G \\ I_i^B \end{bmatrix} - \begin{bmatrix} B_2^R \\ B_2^G \\ B_2^B \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \alpha_i \\ \alpha_i \beta_i^F \\ (1 - \alpha_i)\beta_i^B \end{bmatrix} = H^{-1}(\begin{bmatrix} I_i^R \\ I_i^G \\ I_i^B \end{bmatrix} - \begin{bmatrix} B_2^R \\ B_2^G \\ B_2^B \end{bmatrix})$$

Denote $\begin{bmatrix} a^R & a^G & a^B \end{bmatrix}$ as the first row of H^{-1}, then we obtain:

$$\alpha_i = (a^R I_i^R + a^G I_i^G + a^B I_i^B) - (a^R B_2^R + a^G B_2^G + a^B B_2^B)$$

Let $b = a^R B_2^R + a^G B_2^G + a^B B_2^B$, we obtain:

$$\alpha_i = \sum_c a^c I_i^c + b, \forall i \in w$$

Similar to the cost function of grayscale images, Levin defines the cost function as:

$$J(\alpha, a, b) = \sum_{j \in I}(\sum_{i \in w_j}(\alpha_i - \sum_c a_j^c I_i^c - b_j)^2 + \varepsilon \sum_c a_j^{c2})$$

Using the similar derivation process in section 2.1, we can eliminate a and b in the above equation and get $J(\alpha) = \alpha^T L \alpha$ where $L = \bar{G_k}^T \bar{G_k}$ and (i, j)-th element of $\bar{G_k}$ is:

$$\sum_{k|(i,j) \in w_k} (\delta_{ij} - \frac{1}{|w_k|}(1 + (I_i - \mu_k)(\sum_k + \frac{\varepsilon}{|w_k|} I_3)^{-1}(I_j - \mu_k)))$$

where $\sum_k$ is a convariance matrix.

2.3 User Constraint

As is mentioned in the introduction, user constrain is required to obtain a meaningful alpha matte. We allow users to indicate some background pixels($\alpha = 0$) and some foreground pixels($\alpha = 1$). With S as the user constraint, we solve for:

$$\alpha = argmin(\alpha^T L \alpha), s.t. \alpha_i = s_i, \forall i \in S$$

2.4 Alpha Derivation

The target of image matting is to find the pixels foreground opacity α, which can be derived by solution the equation:

$$\alpha = argmin(\alpha^T L \alpha), s.t. \alpha_i = s_i, \forall i \in S$$

The value of $J(\alpha) = \alpha^T L \alpha$ will be minimal when the derivative of $J(\alpha)$ is 0, thus we have:

$$\frac{\partial J(\alpha)}{\partial \alpha} = 0$$

$$\Rightarrow \frac{\partial(\alpha^T L \alpha)}{\partial \alpha} = 2L\alpha$$

$$\Rightarrow L\alpha = 0$$

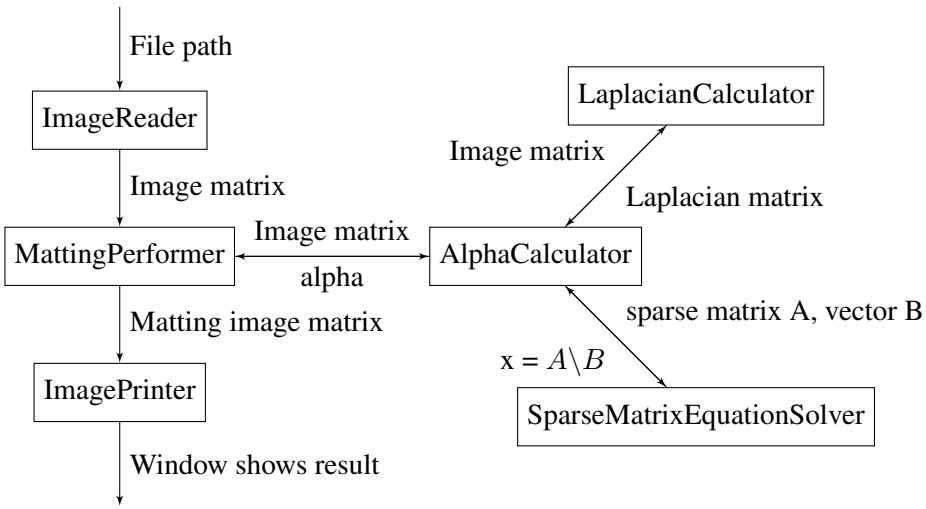


Figure 2: Matting pipeline

Denote $const\_map$ as the matrix whose value is 1 on the scribbled pixel and 0 otherwise. Denote $const\_value$ as the matrix which is derived from $const\_map. * I(:)$. It is obvious that:

$$const\_map * \alpha = const\_val$$

Combine the formulas above together and we get

$$(L + \lambda * const\_map)\alpha = \lambda * const\_val$$

Where λ is a heuristic value to make a balance between the user constraint and the smoothness of the image.

Since both $L + \lambda * const\_map$ and $\lambda * const\_val$ can be calculated from image data, the only unknown variable is α. Thus by solving this equation we can get value of α.

3 Implementation

We implement the matting algorithm in the OOP programming style. The pipeline of the algorithm is shown in Figure 2.

Firstly, given the file paths of original image and scribbled image, ImageReader converts image to matrix and send the matrix to the MattingPerformer.

Secondly, MattingPerformer sends image matrix to AlphaCalculator. With the help of LaplacianCalculator and SparseMatrixEquationSolver, AlphaCalculator gets the alpha and returns it to MattingPerformer. Then MattingPerformer applies alpha to the image, generates matting image and returns it to ImagePrinter.

Finally, ImagePrinter receives the matrix of matting result. It creates a window and draw the image in it to show the matting result.

3.1 ImageReader

Given the file path of an image, ImageReader will read the image and return a matrix which represents the image. The function readImage() uses openCV library to read image data and write into a matrix.

In our implementation, we use ImageReader to read two images: the original image and the image with user input. The pixels of value 1 in the image represent the foreground while the pixels of value 0 represent the background.

3.2 LaplacianCalculator

In the section 2, we proved that $J(\alpha) = \alpha^T L \alpha$, where $L = \bar{G_k}^T \bar{G}$ and (i, j)-th element of $\bar{G}$ is:

$$\sum_{k|(i,j)\in w_k} (\delta_{ij} - \frac{1}{|w_k|}(1 + (I_i - \mu_k)(\sum_k + \frac{\varepsilon}{|w_k|} I_3)^{-1}(I_j - \mu_k)))$$

Given the matrices of image and scribbled image. The task of LaplacianCalculator is to get such a matrix L.

3.3 SparseMatrixEquationSolver

SparseMatrixEquationSolver is specially used to solve the equation $Ax = B$, where A is a sparse matrix of N by N where N is the pixel number of an image.

Assume an image of 400 by 300 pixels, the number of pixel will be 120,000. And the size of matrix A will be 120,000 by 120,000. To store such a large matrix, we use the sparse matrix library, which is provided by Eigen. Then we use umfpack of suitesparse library to solve the equation of $Ax = B$.

3.4 AlphaCalculator

Alpha calculator will firstly use LaplacianCalculator to get Laplacian matrix. Then it constructs the sparse matrix equation

$$(L + \lambda * const\_map)\alpha = \lambda * const\_val$$

as is discussed in section 2.4, and uses SparseMatrixEquationSolver to solve it and gets alpha values.

3.5 MattingPerformer

Matting performer uses AlphaCalculator to calculate the matte alpha. Then apply alpha to the original image and get matting image.

3.6 ImagePrinter

ImagePrinter receieves the result from MattingPerformer and then prints the matting result. The function writeImage uses openCV library to create a window and draw the matting results in it.

4 Experimental results

4.1 Results with different lambda

Since we use the following equation to calculate the alpha, λ actually describes to what extent should output satisfy the user's input.

$$(L + \lambda * const\_map)\alpha = \lambda * const\_val$$

Figure 3 shows the case when the λ given is too small. The matting result α on the scribbled pixel is fractional while user input is integer 0 or 1 to indicate it is a background pixel or foreground pixel. Thus the result generated here does not satisfy the user input strictly.

4.2 Results with differenent epsilon

In the cost function of $J(\alpha)$, we use ε for numerical stability:

$$J(\alpha, a, b) = \sum_{j \in I}(\sum_{i \in w_j}(\alpha_i - a_j I_i - b_j)^2 + \varepsilon a_j^2)$$

Since a larger ε leads to a smaller a_j by the cost function, the larger ε it is, the smoother the matting result will be. Figure 4 shows the case where the ε is too small, in which some details are quite sharpe and inaccurate. Figure 5 shows the case when the ε is too large, which are quite smooth but some details are lost.

4.3 Results with properly chosen epsilon and lambda

In the project, we choose the λ as 100, and ε as 0.00001, which turns out to be a proper choice and generates a satisfying result. The matting result in Figure 6 is much better than the results in Figure 3, 4, 5.

(a) (b)

Figure 3: Matting result for kid.jpg if λ is too small

(a) (b)

Figure 4: Matting result for kid.jpg if ε is too small

(a) (b)

Figure 5: Matting result for kid.jpg if ε is too large

(a) (b)

Figure 6: Matting result for kid.jpg with properly chosen epsilon and lambda

4.4 General results

When we have obtained the foreground opacity α, we apply it to the input image. We use following formula to get foreground image and backround image:

$$I_{foreground} = I * \alpha$$

$$I_{background} = I * (1 - \alpha)$$

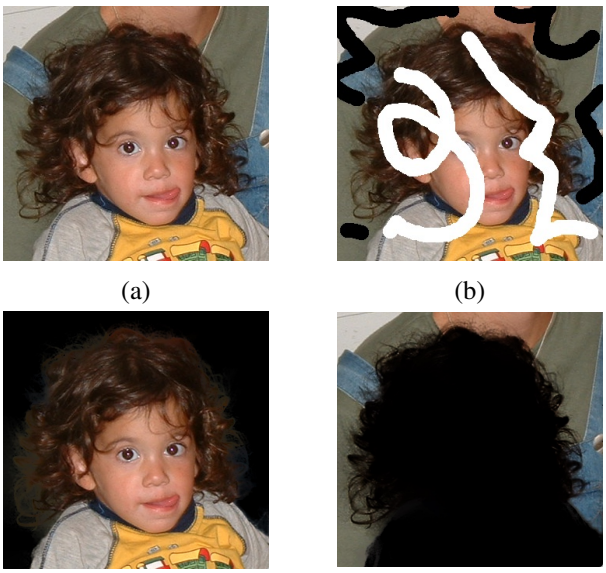
Figure 7, 8, 9 show the matting results for different input image. The first image (a) in each figure is the original input image. The second image (b) is the input image with scribbles provided by user interaction, where 0 means the pixel belongs to the background and 1 means the pixel belongs to the foreground. The third image is the foreground image of matting result $I_{foreground}$ and the fourth image is the background image of matting result $I_{background}$.

From these figures we can find that, with a proper user input, the matting algorithm can successfully separate the foreground object from the background. Some fine or fuzzy features, like the hair of the kid in the Figure 7, the featherlike hairs of dandelion in Figure 8, and the feather of peacock in Figure 9 are separated successfully.

As for the user interaction, the amount of scribbles needed actually depends on the color distribution of the input image.

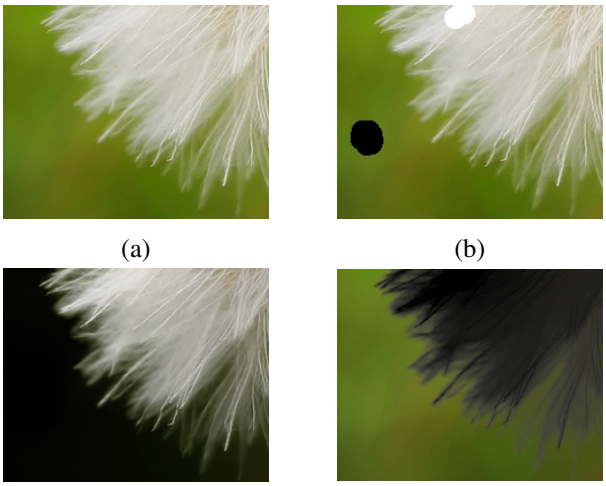
In the Figure 8, since the foreground is dandelion, which has pure white color, and the background is green, the color distribution of foreground and background are simple. For such kind of image, a few scribbles provided by user will be enough to generate a good result.

However, in the Figure7, the case is more complex. The foreground of kid.jpg is the kid in the image, who has a brown hair, white face, yellow and gray clothes. The background is the body of someone else, with green and blue clothes. For such kind of image, scribbles provided by user should cover these different colors as many as possible. Otherwise, the matting result might generate some errors.



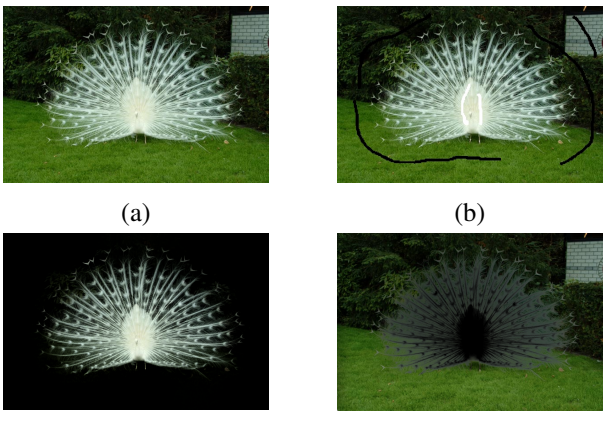
(a) (b) (c) (d)

Figure 7: Matting result for kid.jpg



(a) (b) (c) (d)

Figure 8: Matting result for dandelion.jpg



(a) (b) (c) (d)

Figure 9: Matting result for peacock.jpg

5 Conclusion

5.1 Why the matting algorithm works

Since our assumption is based on the color line model, if we successfully minimize the cost function

$$J(\alpha, a, b) = \sum_{j \in I} (\sum_{i \in w_j} (\alpha_i - a_j I_i - b_j)^2 + \varepsilon a_j^2)$$

then using the alpha generated, we will get the background image and foreground image which fits color line model as much as possible.

Meanwhile, the user constraint specifies the alpha value exactly as 0 or 1. Through the propagation of overlapping windows, most of alpha have a value close to 0 or 1, with a few of them being fractional on the boundary of foreground and background. It is the reason why we can separate the foreground object from the background successfully.

5.2 Accuracy analysis

For the image with simple color distribution of foreground and background, a few scribbles provided by user will be sufficient enough to generate accurate matte alpha. However, if the image has a complex color distribution, user's input will be required to cover most of colors, which means more scribbles are needed on several areas rather than just a few dots.

With a proper choice of λ and ε, given proper user scribbles, the matting algorithm described in the this project can perform image matting accurately.

5.3 Efficiency analysis

To solve the equation

$$(L + \lambda * const\_map)\alpha = \lambda * const\_val$$

where L is the Laplacian matrix with size of N by N, where N is the number of pixels in the input image. Assume we have an image of 800 by 600, the size of L will be 480000 by 480000. The time cost to solve a matrix equation which contains such a huge matrix will be quite expensive.

6 Future work

6.1 Improvement on accuracy

Since the matting problem is an under-constrained problem, we need an accurate and useful user input as constraint. Since the Laplacian matrix can be calculated without any user input, (Levin et al., 2006) suggests that we can make use of the eigenvector of Laplacian matrix to guide user to perform scribbles drawing.

6.2 Improvment on efficiency

As is mentioned in section 5.3, the time cost for current algorithm is quite expensive. Since we want to perform the calculation on mobile device rather than on the computer, efficiency needs improvement.

We can reduce the amount of calculation by downsampling the image. For example, we can downsample the original image from 800 by 600 to 160 by 120. And then perform image matting on the image downsampled. However, if we downsample too much, the detail of image will be lost and the matting result will lose accuracy as well. Thus it is a trade-off and we can work on finding the optimal downsampling rate in the future.

References

Y. Y. Boykov and M. P. Jolly. 2001. Interactive graph cuts for optimal boundary amp; region segmentation of objects in n-d images. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 1, pages 105–112 vol.1.

Yung-Yu Chuang, Brian Curless, David H. Salesin, and Richard Szeliski. 2001. A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, volume 2, pages 264–271. IEEE Computer Society, December.

Anat Levin, Dani Lischinski, and Yair Weiss. 2006. A closed form solution to natural image matting. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 61–68, Washington, DC, USA. IEEE Computer Society.

Yin Li, Jian Sun, Chi-Keung Tang, and Heung-Yeung Shum. 2004. Lazy snapping. *ACM Trans. Graph.*, 23(3):303–308, August.

I. Omer and M. Werman. 2004. Color lines: image specific color representation. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–946–II–953 Vol.2, June.

Jian Sun, Jiaya Jia, Chi-Keung Tang, and Heung-Yeung Shum. 2004. Poisson matting. *ACM Trans. Graph.*, 23(3):315–321, August.