Fast Matting Using Large Kernel Matting Laplacian Matrices

Kaiming He[1] Jian Sun[2] Xiaoou Tang[1,3]
[1]Department of Information Engineering, The Chinese University of Hong Kong
[2]Microsoft Research Asia
[3]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences

Abstract *Image matting is of great importance in both computer vision and graphics applications. Most existing state-of-the-art techniques rely on large sparse matrices such as the matting Laplacian [12]. However, solving these linear systems is often time-consuming, which is unfavored for the user interaction. In this paper, we propose a fast method for high quality matting. We first derive an efficient algorithm to solve a large kernel matting Laplacian. A large kernel propagates information more quickly and may improve the matte quality. To further reduce running time, we also use adaptive kernel sizes by a KD-tree trimap segmentation technique. A variety of experiments show that our algorithm provides high quality results and is 5 to 20 times faster than previous methods.*

1. Introduction

Image matting refers to the problem of softly extracting the foreground object from a single input image. Denoting a pixel's foreground color, background color, and foreground opacity (alpha matte) as $\mathbf{F}$, $\mathbf{B}$, and α respectively, the pixel's color $\mathbf{I}$ is a convex combination of $\mathbf{F}$ and $\mathbf{B}$:

$$\mathbf{I} = \mathbf{F}\alpha + \mathbf{B}(1-\alpha). \quad (1)$$

Image matting has a number of important applications, such as image/video segmentation, layer extraction, new view synthesis, interactive image editing, and film making.

Since the problem is highly ill-posed, a trimap (or strokes) indicating definite foreground, definite background, and unknown regions is usually given by the user. To infer the alpha matte in the unknown regions, Bayesian Matting [4] uses a progressively moving window marching inward from the known regions. In [2], the alpha value is calculated by the geodesic distance from a pixel to the known regions. Affinity-matrix-based methods propagate constraints from the known regions to the unknown regions: Poisson matting [19] solves a homogenous Laplacian matrix; Random Walk matting [7] defines the affinity matrix by a Gaussian function; in the closed-form matting [12], a *matting Laplacian matrix* is proposed under a color line assumption. The matting Laplacian can also be combined

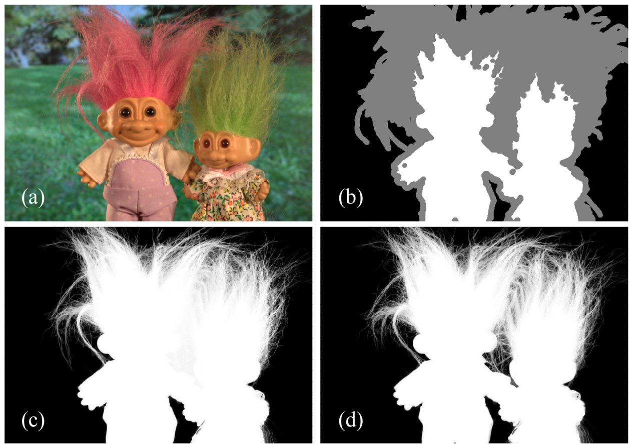

Figure 1. (a) Image (800×563 pixels). (b) Trimap (180k unknown pixels). (c) Result of closed-form matting [12]. The running time is 28 seconds using a coarse-to-fine solver in [12]. (d) Our result. The running time is 2.5 seconds. Notice the gap between the toys.

with different data weights or priors [24, 14, 15]. New appearance models [18] and a learning based technique [25] are proposed to generalize the color line assumption. Currently, matting-Laplacian-based methods produce the best quality results [16, 23].

However, efficiency is also very important for image matting, especially when we are faced with multi-megapixel images produced by today's digital cameras. The progressively moving window in Bayesian Matting [4] is computationally expensive. The geodesic framework [2] presents a linear time algorithm. But the model itself may not be able to handel complex cases like hair. Poisson Matting [19] and Random Walk Matting can be accelerated by a multigrid solver or a GPU implementation, but lack satisfactory quality [16]. The methods using the matting Laplacian [12, 24, 14, 15] or other sophisticated matrices [18, 25] have to solve a large linear system - a very time consuming operation. Soft Scissor [21] locally solves a small linear system while the user is drawing the trimap. But it is still slow for mega-pixel images and not applicable for predrawn trimaps. A more efficient method for high quality matting is still desired.

In this paper, we propose a fast algorithm for high quality image matting using large kernel matting Laplacian matrices. The algorithm is based on an efficient method to solve the linear system with the large kernel matting Laplacian. Using a large kernel can accelerate the constraint propagation, reduce the time of the linear solver for convergence, and improve the matting quality. To further speed-up the algorithm, we use a KD-tree based technique to decompose the trimap so that we can assign an adaptive kernel size to each sub-trimap. Thus, the number of iterations can be fixed beforehand and the running time of the whole algorithm is only linear to the number of the unknown pixels. We demonstrate that our algorithm is about 5 to 20 times faster than previous methods while achieving high quality (Fig. 1). Our algorithm is also useful for other applications using the matting Laplacian, like haze removal [9], spatially variant white balance [10], and intrinsic images [3].

2. Background

The matting Laplacian matrix proposed in [12] is an affinity matrix specially designed for image matting. The key assumption of this method is the *color line model*: the foreground (or background) colors in a local window lie on a single line in the RGB color space. It can be proved that α is a linear transformation of $\mathbf{I}$ in the local window:

$$\alpha_i = \mathbf{a}^{\mathrm{T}}\mathbf{I}_i + b, \forall i \in \omega, \tag{2}$$

Here i is a pixel index, $\mathbf{I}_i$ and $\mathbf{a}$ are 3×1 vectors, and $\mathbf{a}$ and b are assumed to be constant in the local window ω.

Accordingly, a cost function $J(\alpha, \mathbf{a}, b)$ can be defined to encourage the alpha obeying this model:

$$J(\alpha, \mathbf{a}, b) = \sum_{k \in \mathbf{I}} \Big(\sum_{i \in \omega_k} (\alpha_i - \mathbf{a}_k^{\mathrm{T}}\mathbf{I}_i - b_k)^2 + \varepsilon \mathbf{a}_k^{\mathrm{T}}\mathbf{a}_k \Big), \tag{3}$$

where ω_k is the window centered at pixel k, and ε is a regularization parameter. By minimizing the cost function w.r.t. $(\mathbf{a}, b)$, a quadratic function of α can be obtained:

$$J(\alpha) = \alpha^{\mathrm{T}} \mathrm{L} \alpha. \tag{4}$$

Here α is denoted as an N$\times$1 vector, where N is the number of unknowns. And L is called *matting Laplacian*. It is an N$\times$N symmetric matrix whose (i, j) element is:

$$\sum_{k|(i,j) \in \omega_k} \Big(\delta_{ij} - \frac{1}{|\omega_k|}\big(1 + (\mathbf{I}_i - \mu_k)^{\mathrm{T}} (\Sigma_k + \frac{\varepsilon}{|\omega_k|}\mathrm{U})^{-1} (\mathbf{I}_j - \mu_k)\big)\Big), \tag{5}$$

where δ_{ij} is the Kronecker delta, μ_k and Σ_k are the mean and covariance matrix of the colors in window ω_k, $|\omega_k|$ is the number of pixels in ω_k, and U is a 3$\times$3 identity matrix.

Combining this cost function with the user-specified constraints (trimap), the whole cost function is defined as:

$$E(\alpha) = \alpha^{\mathrm{T}} \mathrm{L} \alpha + \lambda (\alpha - \beta)^{\mathrm{T}} \mathrm{D} (\alpha - \beta), \tag{6}$$

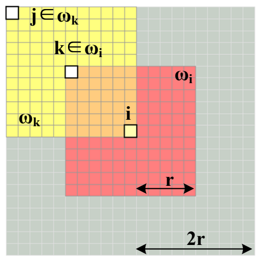

Figure 2. According to (5), the (i, j) element of L is non-zero only if $(i, j) \in \omega_k$. If the radius of the window ω is r, the radius of the kernel centered at i is 2r. The kernel is a rectangle including all the gray pixels in this figure.

where β is the trimap, D is a diagonal matrix whose elements are one for constraint pixels and zero otherwise, and λ is a large number. A data term specified by color sampling confidence [24, 14] or a sparsity prior [15] can also be incorporated. The cost function (6) can be optimized by solving a sparse linear system:

$$(\mathrm{L} + \lambda \mathrm{D})\alpha = \lambda \mathrm{D} \beta. \tag{7}$$

The techniques of haze removal [9], spatially variant white balance [10], and intrinsic images [3] also involve this linear system.

Given the linear system, we need to use an appropriate solver to recover α. Non-iterative methods like LU decomposition are not effective to handle this system in large scale due to the high memory cost. So iterative methods are usually applied, like the Jacobi method or Conjugate Gradient (CG) [17]. The information of the known pixels is propagated into the unknown region by iteratively multiplying the matrix. However, iterative methods are often time consuming. For example, the time complexity of CG is O($\mathrm{N}^{1.5}$) for N unknowns. A medium size image (800×600) may take hundreds or thousands of iterations. Another drawback of the iterative methods is that the time needed is hard to predict because the iteration number depends on the number of unknowns, the image content, and the trimap shape. This is not favored for interactive image matting.

In [12], a coarse-to-fine scheme is proposed to speed-up the linear system solver. But the down-sampling process may degrades the matte quality, especially for the foreground with very thin structures. Moreover, in the fine resolution it also suffers from the drawbacks of the iterative methods. Locally adapted hierarchical basis preconditioning in [20] is not directly applicable because the elements of the matting Laplacian matrix (5) are not first-order smooth.

3. Large Kernel Matting Laplacian

Existing methods [12, 24, 14, 15] usually use a small window because the matrix will be less sparse with a larger window. Conventional wisdoms hold that solving a less sparse system is even slower. But we point out it is not necessarily true. A less sparse system needs fewer iterations to converge; the only bottle neck is the increased computational burden in each iteration. In this section, we propose an algorithm to greatly reduce the time needed for each iteration, so the solver is in fact faster if we use a larger window.

The kernel size of a matting Laplacian is defined as the the number of the non-zero elements in a row of L. Denote the window radius of ω in (5) by r. The kernel size is $(4r+1)^2$ (see Fig. 2). Existing methods mainly use $r=1$ because L will become less sparse when r is larger. Both the memory and the time needed to solve the linear system will increase tremendously.

Take CG for example. The solver iteratively multiplies the conjugate vector by the Laplacian matrix (please see [17] for detailed description of CG). In each iteration, the matrix product Lp dominates the computation cost . Here p is the conjugate vector of the previous iteration. In the view of signal processing, the matrix product Lp is the response of a spatially variant filter L on p, whose i^{th} element is:

$$(\mathrm{L}p)_i = \sum_j \mathrm{L}_{ij} p_j. \tag{8}$$

Computing Lp using (5) and (8) involves spatially variant convolution, whose time and memory complexity is O(Nr^2). This is not affordable when r gets larger.

We notice that in each iteration, a pixel can influence another pixel that is 2r away. So the information is propagated by 2r pixels. The CG solver will converge in fewer iterations if the kernel size is larger. This motivates us to envision that the use of a large kernel may reduce the solver's running time, if we can significantly increase the speed of the convolution in (8).

Toward this goal, we present an O(N) time (independent of r) algorithm to compute the product Lp in each iteration. Instead of computing L's elements and the convolution explicitly, we calculate the product Lp as a whole by the following algorithm:

Algorithm for calculating Lp

Given a conjugate vector p, Lp can be calculated through the following three steps:

$$\mathbf{a}_k^* = \Delta_k^{-1}\Big(\frac{1}{|\omega|}\sum_{i\in\omega_k}\mathbf{I}_i p_i - \mu_k \bar{p}_k\Big), \tag{9}$$

$$b_k^* = \bar{p}_k - \mathbf{a}_k^{*\mathrm{T}}\mu_k, \tag{10}$$

$$(\mathrm{L}p)_i \equiv q_i = |\omega| p_i - \Big(\big(\sum_{k\in\omega_i}\mathbf{a}_k^*\big)^{\mathrm{T}}\mathbf{I}_i + \big(\sum_{k\in\omega_i} b_k^*\big)\Big), \tag{11}$$

where $\mathbf{a}_k^*$ is a 3×1 vector for each pixel k, $\bar{p}_k$ is the mean of p in ω_k, $\Delta_k = \Sigma_k + \frac{\varepsilon}{|\omega_k|}\mathrm{U}$, and we denote $(\mathrm{L}p)_i$ as q_i.

Theorem: The q given by the above algorithm is identical to the Lp calculated by (5) and (8).

Proof: Written in matrix notation, from (9) there is an affine transform:

$$\mathbf{a}^* = \mathrm{A}p, \tag{12}$$

where A is a coefficient matrix only dependent on $\mathbf{I}$. If we put (9) into (10), we can see that b^* is also p's affine transform: $b^* = \mathrm{B}p$. Similarly, q is p's affine transform: $q = \mathrm{Q}p$.

Consequently, in order to show $q = \mathrm{L}p$, we only need to prove $\partial q_i/\partial p_j = \mathrm{L}(i,j)$. Putting (10) into (11) and eliminating b, we obtain:

$$\frac{\partial q_i}{\partial p_j} = |\omega|\delta_{ij} - \sum_{k\in\omega_i}\Big(\frac{\partial \bar{p}_k}{\partial p_j} + \frac{\partial \mathbf{a}_k^{*\mathrm{T}}}{\partial p_j}(\mathbf{I}_i - \mu_k)\Big) \tag{13}$$

Here, we have:

$$\frac{\partial \bar{p}_k}{\partial p_j} = \frac{1}{|\omega|}\sum_{n\in\omega_k}\frac{\partial p_n}{\partial p_j} = \frac{1}{|\omega|}\delta_{j\in\omega_k} = \frac{1}{|\omega|}\delta_{k\in\omega_j} \tag{14}$$

where $\delta_{j\in\omega_k}$ is 1 if $j \in \omega_k$, and is 0 otherwise. According to (9) we also have:

$$\begin{aligned}\frac{\partial \mathbf{a}_k}{\partial p_j} &= \Delta_k^{-1}\Big(\frac{1}{|\omega|}\sum_{i\in\omega_k}\frac{\partial p_i}{\partial p_j}\mathbf{I}_i - \frac{\partial \bar{p}_k}{\partial p_j}\mu_k\Big) \\ &= \Delta_k^{-1}\Big(\frac{1}{|\omega|}\mathbf{I}_j - \frac{1}{|\omega|}\mu_k\Big)\delta_{k\in\omega_j}\end{aligned} \tag{15}$$

Putting (14) and (15) into (13), we obtain:

$$\frac{\partial q_i}{\partial p_j} = |\omega|\delta_{ij} - \frac{1}{|\omega|}\sum_{k\in\omega_i, k\in\omega_j}\big(1 + (\mathbf{I}_j - \mu_k)^{\mathrm{T}}\Delta_k^{-1}(\mathbf{I}_i - \mu_k)\big)$$

This is exactly $\mathrm{L}(i,j)$ in (5). ■

The proposed algorithm also has intuitive interpretations: Equations (9) and (10) indeed are *linear regression* solutions and the regression model is $p_i \approx \mathbf{a}_k^{*\mathrm{T}}\mathbf{I}_i + b_k^*, \forall i \in \omega_k$. Further, (11) can be rewritten as:

$$(\mathrm{L}p)_i = \sum_{k\in\omega_i}\big(p_i - (\mathbf{a}_k^{*\mathrm{T}}\mathbf{I}_i + b_k^*)\big). \tag{16}$$

For any pixel $\mathbf{I}_i$, the term $(p_i - (\mathbf{a}_k^{*\mathrm{T}}\mathbf{I}_i + b_k^*))$ is the error between p_i and its linear prediction. As $\mathbf{I}_i$ is involved in all the regression processes satisfying $k \in \omega_i$, equation (16) is the sum of errors in all windows around i.

All the summations in (9) and (11) can be very efficiently computed using the integral image technique [6]. Using the integral image, the sum of the any window can be obtained in constant time (four operations). Therefore, the time complexity for computing Lp in each iteration is O(N')≈O(N), where N' is the size of the bounding box of the unknown region. Note that the time complexity is independent of the kernel size.

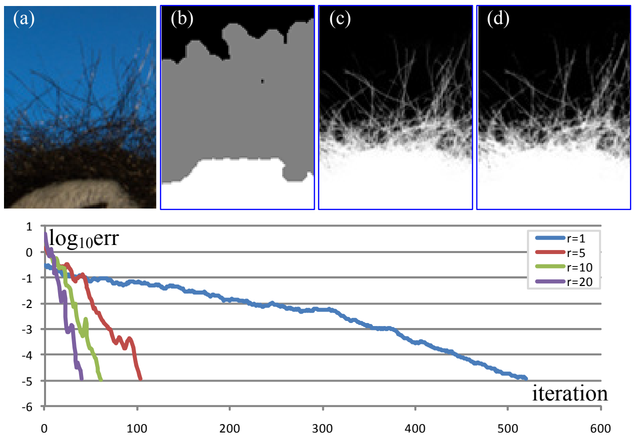

Figure 3. (a) Cropped image (98 × 130). (b) Trimap. (c) r=1, iteration 519. (d) r=20, iteration 44. On the bottom are the curves of error vs. iteration numbers.

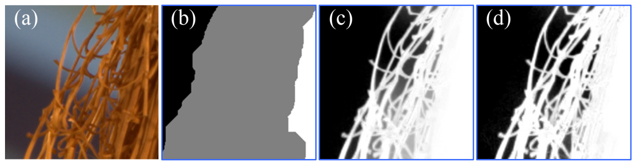

Figure 4. (a) Cropped image (231 × 231). (b) Trimap. (c) r=1. (d) r=20.

Discussion on Kernel Size

Using the above algorithm, the running time of the CG solver can be greatly reduced if we use a large kernel - the solver converges in fewer iterations. In Fig. 3, we experiment several different kernel sizes on the same image. The solver converges much faster in the large kernel case. Our algorithm's running time is 1.07s (r=1, converges in 519 iterations) and 0.090s (r=20, converges in 44 iterations), while the running time of brute force CG using (5) and (8) is 0.95s (r=1) and 22s (r=20). In this example, the resulting mattes are almost visually identical (Fig. 3(c)(d)).

A large kernel may improve the quality because a large window may cover disconnected regions of the foreground/background. This property is particularly favored when the foreground object has holes. In Fig. 4(c), r = 1 is too small to cover the known background and the holes. We can obtain a better matte if we use a larger window size r = 20 (Fig. 4(d)).

Our large kernel method is typically appropriate for high resolution images. In Fig. 5(c), a small window is not sufficient to describe the color lines and results in the loss of the fine structures. However, a large window collects enough color samples and gives a better result (Fig. 5(d)). A large window also covers more known pixels near the boundary of unknowns and provides more stable constraints.

The drawback of a large kernel is that when the foreground/background is complicated, a larger window leads

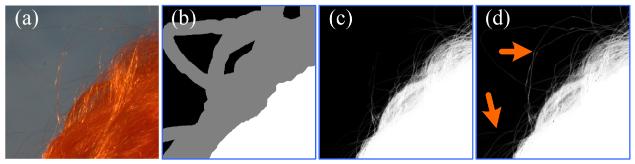

Figure 5. (a) A cropped image (636 × 636) from a 2080 × 2600 image. (b) Trimap. (c) r=1. (d) r=60. We recommend viewing the electronic version of this paper for the fine structures.

to a higher probability of breaking the color line assumption. In the zoomed-in patches of Fig. 6, a large window covers more than one color clusters of the background, so the colors do not lie in the same line. Consequently, the matte is not as accurate as using a small window.

We observe that in practice, the user draws a wide band of unknowns near a fuzzy object or an object with holes, as it is difficult to provide a precise boundary; the band of unknowns near a solid boundary is relatively narrow. We can see this fact in Fig. 1(b) and Fig. 6(b). Besides, a high resolution image may also result in a wider band of unknowns. So we prefer to use a large kernel to efficiently propagate constraints in a wide band. For a narrow band, a small kernel is favored to avoid breaking the color line assumption. To achieve this goal, in the next section, we adaptively set the kernel size based on a trimap segmentation technique.

4. KD-tree Based Trimap Segmentation

In this section, we propose a trimap segmentation method to further improve both the quality and efficiency. This allows us to assign different window sizes to different regions.

Given a trimap, we first calculate the unknown region's barycenter (x_c, y_c), x variance $\sigma_x^2 = \frac{1}{\mathrm{n}}\sum_{\mathrm{U}}(x_c - x)^2$ and y variance σ_y^2, where U is the set of unknown pixels, and n is the size of U. Then we use a line that passes through (x_c, y_c) to divide the map into two regions. This line is vertical to the axis (x or y) with a larger variance. A trimap is recursively divided and a 2D KD-tree is built accordingly.

Next we discuss the conditions when the recursive process stops. We observe that if a cropped image covers enough foreground and background constraints, a reasonably good matte can be obtained by only considering this cropped image independently, as demonstrated in Fig. 3 and Fig. 4. Therefore, we hope the segments are small enough, while as many of them as possible have both foreground (F) and background (B) constraints. So the recursive division stops until one of these conditions is satisfied: (a) the segment only has F and U; (b) the segment only has B and U; (c) if we divide the segment, one of its children only has F and U, and the other only has B and U; (d) The segment's min(width, height) is smaller than a threshold (32 in our experiments). Fig. 7 gives an example of a segmented trimap. Notice that the band width of the unknowns in each segment

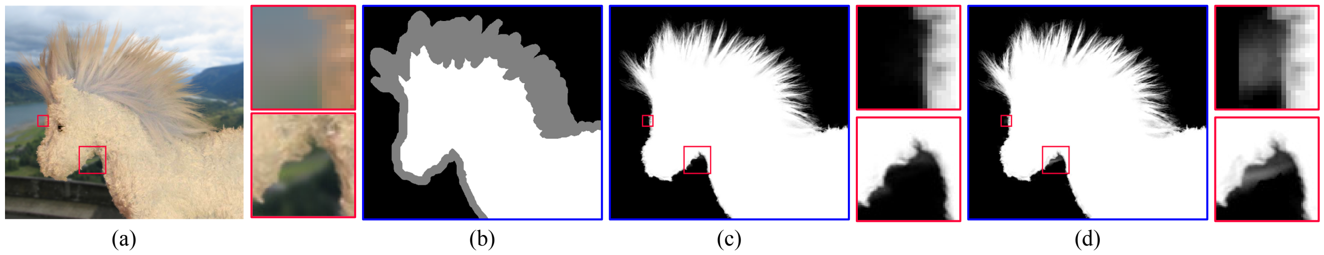

Figure 6. (a) A 640 × 600 image. (b) Trimap. (c) r=1. (d) r=10. We recommend viewing the matte in the electronic version of this paper.

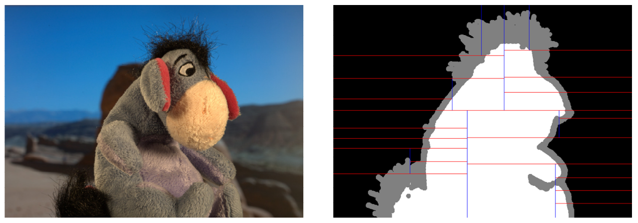

Figure 7. Trimap segmentation.

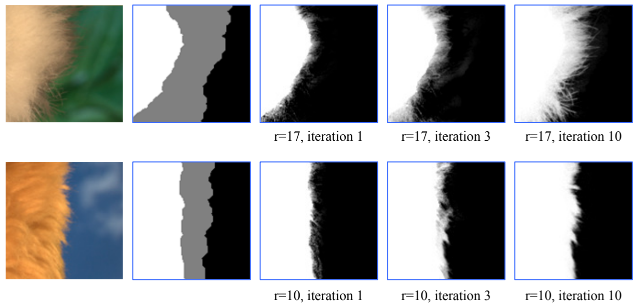

Figure 8. Propagation and window sizes. The band width $w_b \approx 50$ in the first row, and $w_b \approx 30$ in the second row,. The window radius $r = w_b/3$.

is nearly uniform.

Once the trimap is segmented, we can solve the linear system in each segment. The integral image is only calculated in the bounding box of the unknowns in each segment. We first solve the segments that satisfy condition (c) or (d), as they directly contain both F and B constraints. Then, other segments (that only have F and U, or B and U) are solved in inverse Breadth-First Search order, *i.e.*, the deeper leaves in the KD-trees have a higher priority. As they do not contain F or B constraint, we use the solved matte of the neighboring segment as boundary conditions. The inverse Breadth-First Search order insures that at least one of their neighbors has been solved before.

More importantly, we adapt the kernel size to the band width of the unknowns. For each segment, let (w, h) be the width and height of U's bounding box. We approximate the width of the band by w_b=n/max(w,h). We set the window radius $r = w_b/\eta$, where η is a factor. Intuitively, the propagation will influence the other side of the band in η/2 iterations (notice that the kernel radius is 2r). Therefore, the number of iterations needed for convergence is in the order of η, and can be fixed beforehand. In Fig. 8, we show that the propagation speed is adaptive to the band width. The same number of iterations provide high quality mattes for two bands of different width.

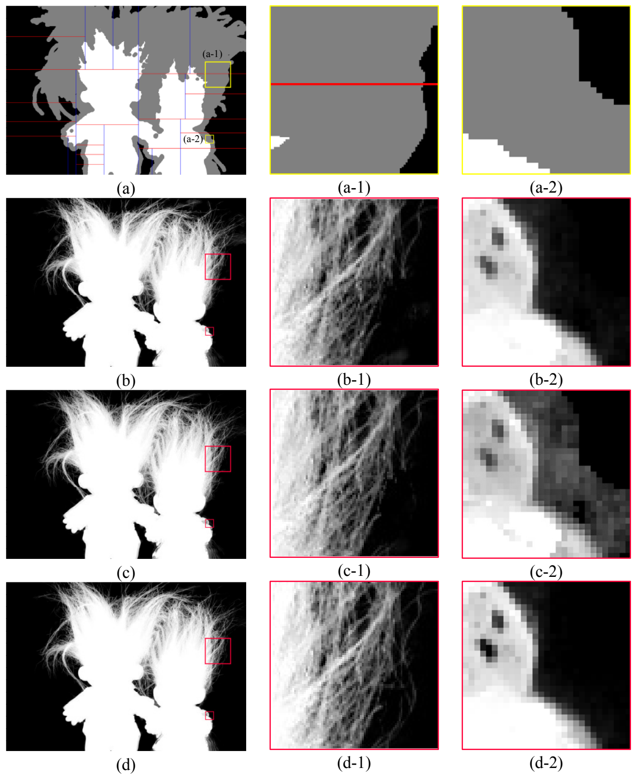

Figure 9. Local-global-local Scheme. (a) Segmented trimap of Fig. 1. (b) Local step 1. (c) Global step. (d) Local step 2. Please refer to the text for explanation. We recommend to see the electronic version of these images.

A Local-Global-Local Scheme

To prevent the result being too locally determined, we propose a local-global-local scheme. Given the trimap segmentation, we first solve the matte for each segment. Here we let η=3. This step propagates information fast. We empirically fix the iteration number to 10 which is sufficient to get a good initial matte for the next step (Fig. 9(b)). Noticeable seams may exist on segment boundaries (Fig. 9(b-1)).

Using the above obtained solution as initial guess, we optimize the whole linear system globally. In this step, we set the window radius r to be min(w, h)/50 where w and h are the image's width and height. The iteration number is fixed on 5. This global step removes the seams (Fig. 9(c-1)). But as the kernel size may be too large for some local regions, there are artifacts due to the failure of the color line assumption (Fig. 9(c-2)).

In the final step, we refine this solution locally using the trimap segmentation again. Here we let η = 15 and run 20 iterations. In experiments we found that more iterations improve the result very little. The window size is much smaller than the first local step because we want to maintain the color line model. This suppresses the artifacts due to the large window size used in previous two steps (Fig. 9(d-2)).

Time Complexity

Denote N' as the total area of all the bounding boxes of the unknowns in all segments. The time complexity of the local steps is O(N'), since the time for calculating Lp in each iteration is O(N') and the iteration number is fixed. For the same reason, the time complexity of the global step is O(M), where M is the size of the bounding box of the whole unknown region. In experiment, we find that the time for the global step is much less than that of the local steps due to fewer iterations. So the total time complexity is O(N'). As N' is slightly larger than the number of unknowns N, the running time is almost linear to N. Besides, once the trimap is segmented, the time can be predicted before running the solver.

5. Results

We compare our algorithm with the closed-form matting in [12] (*i.e.*, r=1). For closed-form matting, we calculate the matting Laplacian using (5) and then solve the linear system using two different methods: brute force CG (simply denoted as CG below), and the coarse-to-fine (C-F) scheme described in [12]. For CG, we fix the convergence tolerance (10^{-5}). For C-F, we run three levels and in each level we solve the system using CG. All algorithms are implemented using C++ on a laptop with an Intel Core Duo 2.0GHz CPU and 4G memory.

Table 1 shows the running time. Our algorithm is about 5 to 20 times faster than CG or C-F for medium and large images. We also study the average running time per unknown pixel ($\tau = t/\mathrm{N}$). Our algorithm gives similar τ (about 0.02) for different images. The small variation of our τ is mainly because the time is linear to N' but not N. On the contrary, τ varies severely for the other two algorithms, because it is influenced by the number of unknowns, image content and the shape of the trimap. Their τ is also much larger for mega-pixel images (Fig. 13).

To further study the effect of image sizes on the running time, we down-sample a 5.4M-pixel image (2080 $\times$ 2600, the first row of Fig. 13) and its trimap to ten different scales and run all algorithms. The curves are shown in Fig. 10.

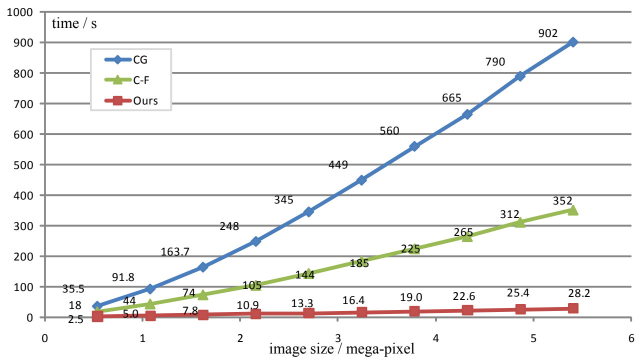

Figure 10. Image size vs. running time.

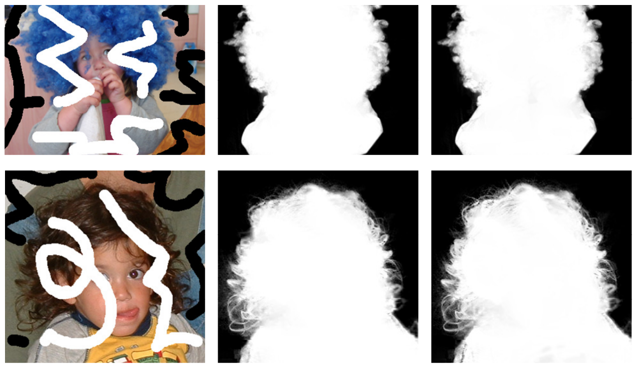

Figure 14. Mattes from strokes. (a) Images and strokes. (b) Results of the closed-form matting. (c) Our results. The images and stroke maps are from [12].

The time of our algorithm increases linearly with the image size, while the time of CG and C-F increases super-linearly.

Next, we compare the matte qualities. For medium images, the results are visually similar for fuzzy objects (Fig. 11). However, our method is better at handling holes, like Fig. 12 and the gap between the toys in Fig. 1. Further, in Fig. 13 we show that our large kernel method greatly improves the quality for high resolution images, as a large kernel collects enough samples to describe the color lines.

In Fig. 14, we show the results of stroke-based constraints. The resulting mattes are visually very similar. Note that the stroke-based framework has been proposed to reduce the user's labor. But strokes lead to large areas of unknowns, which increase the running time and often make the whole interaction process even longer. Our method provides comparable results in very short time, so facilitates the feedback from the user.

Table 2 shows the average rank of mean square error (MSE) on the benchmark data [1]. The MSE rank of our algorithm is comparable with the closed-from matting [12] and Robust Matting [24]. Note that [24, 14, 15] combine the matting Laplacian with data terms, while our method and [12] do not. We believe that our algorithm can also greatly reduce their running time and further advance the state-of-the-art in both efficiency and quality.

image			time			average time per k unknowns (τ)		
	size	unknowns	CG	C-F	ours	CG	C-F	ours
Fig 1	800 × 563	180k	49s (20)	29s (11)	2.5s	0.27	0.16	0.014
Fig 11	720 × 515	48k	7.7s (10)	7.1s (9)	0.8s	0.16	0.15	0.017
	800 × 575	49k	9.4s (10)	6.8s (8)	0.9s	0.19	0.14	0.018
Fig 12	800 × 595	143k	44s (14)	28s (9)	3.1s	0.31	0.20	0.022
	800 × 524	57k	14s (11)	5.9s (5)	1.2s	0.25	0.10	0.023
Fig 13	2080 × 2600	1280k	902s (32)	352s (12)	28s	0.70	0.28	0.022
	3040 × 2350	766k	440s (28)	256s (16)	15.7s	0.57	0.33	0.020
	2890 × 2600	857k	464s (31)	206s (14)	14.9s	0.54	0.24	0.017
Fig 14	348 × 261	60k	12.6s (16)	2.4s (3)	0.8s	0.21	0.04	0.013
	342 × 329	80k	19s (19)	4.7s (5)	1s	0.24	0.06	0.013
Fig 15	800 × 602	67k	12.4s (11)	11.6s (11)	1.1s	0.19	0.17	0.016
Fig 16	278 × 333	86k	36s (51)	5.7s (8)	0.7s	0.42	0.07	0.008

Table 1. Image sizes and running time. The numbers in the brackets are the ratios to our algorithm's running time. Solving the closed-form matting using brute force Conjugate Gradient (CG) is very slow when the number of unknowns gets larger. The coarse-to-fine scheme (C-F) in [12] accelerates the stroke-based examples (Fig. 14 & 16), but still takes hundreds of seconds for high-resolution image (Fig. 13). Our method is about 5 to 20 times faster. Even mega-pixel images with mega unknowns (Fig. 13) only takes 15 to 30 seconds.

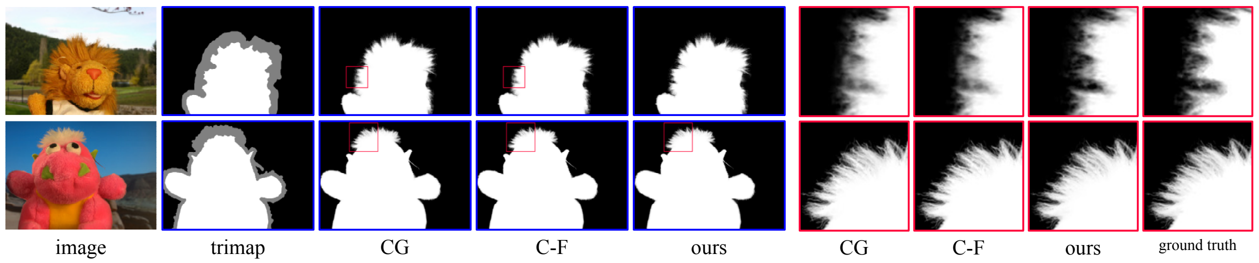

Figure 11. Medium images of fuzzy objects. The images and trimaps are from [1] and [24].

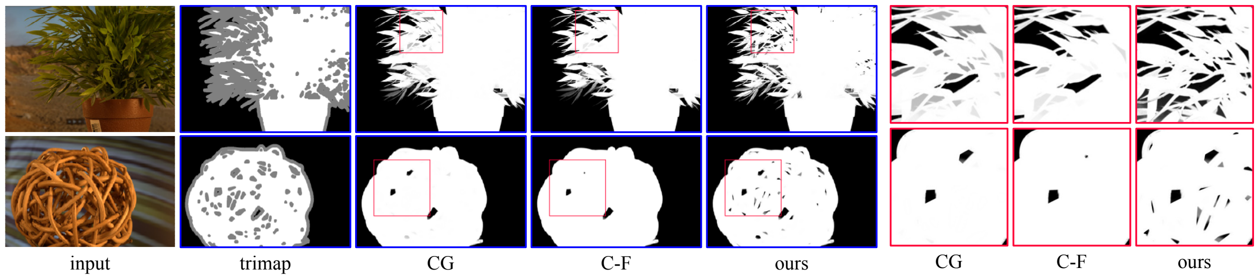

Figure 12. Medium images of foreground objects with holes. The images and trimaps are from [1].

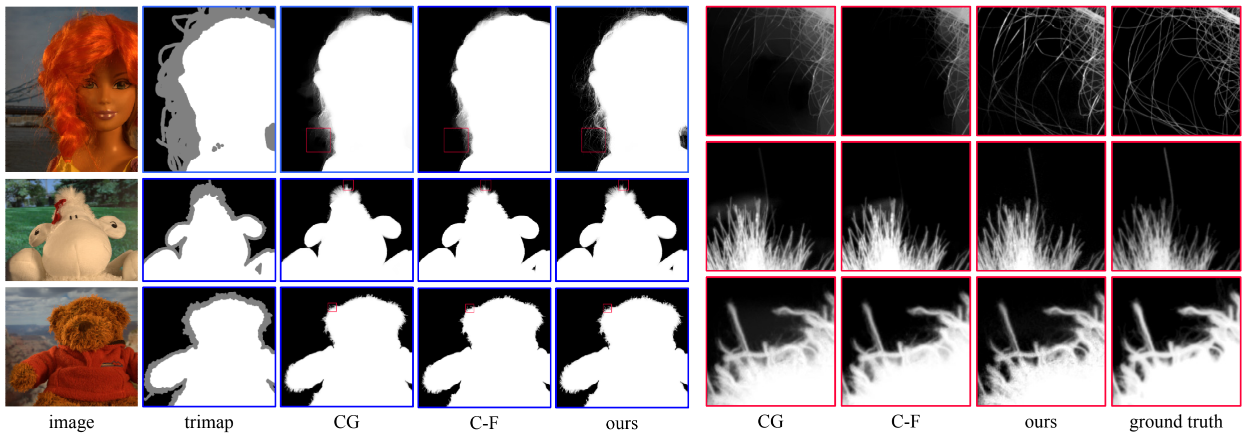

Figure 13. High resolution images. The images and trimaps are from [1]. We recommend to see the electronic version of this paper.

Improved color [14]	2	Random walk [7]	6.8
Our method	3.2	Geodesic [2]	7.5
Closed-form [12]	3.3	Bayesian [4]	8.8
Robust [24]	3.7	Easy matting [8]	9
High-res [15]	4.4	Poisson [19]	10.9
Iterative BP [22]	6.6		

Table 2. Average rank of mean square error (MSE) on a data set from [1]. Please visit [1] for a comprehensive comparison.

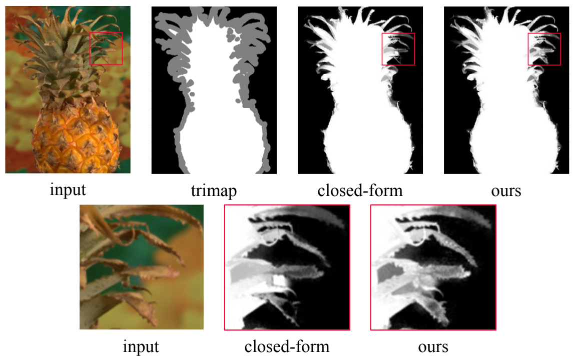

Figure 15. Failure of the color line assumption.

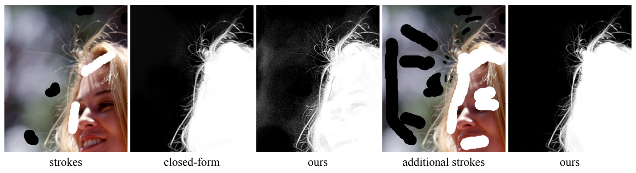

Figure 16. Failure case where the constraints are too sparse.

Like the closed-form matting, our method may fail when the color line assumption is invalid, *e.g.*, when the foreground/background colors are complicated (Fig. 15). Moreover, our method may give poorer results when the band of unknowns is too wide, because the large kernel increases the probability of breaking the color line assumption (Fig. 16). However, as our method runs quickly, it is convenient for the user to improve the result by refining the trimap.

6. Conclusion and Discussion

In this paper, we propose a fast and effective algorithm for image matting. Our novel insight is that solving a large kernel matrix is not necessarily slow. It indeed takes fewer iterations to converge; we only need to focus on reducing the time in each iteration. Besides, a large kernel may also improve quality. This will encourage us to restudy a great category of matrices, like the homogenous Laplacian in Poisson image editing [13], Gaussian affinity Laplacian in colorization [11], or other edge-preserving matrices [5]. Generalizing their definitions to larger kernels will lead to new research topics.

References

[1] http://www.alphamatting.com.

[2] X. Bai and G. Sapiro. A geodesic framework for fast interactive image and video segmentation and matting. *ICCV*, 2007.

[3] A. Bousseau, S. Paris, and F. Durand. User-assisted intrinsic images. *SIGGRAPH Asia*, 2009.

[4] Y. Chuang, B. Curless, D. Salesin, and R. Szeliski. A bayesian approach to digital matting. *CVPR*, 2001.

[5] Z. Farbman, R. Fattal, D. Lischinski, and R. Szeliski. Edge-preserving decompositions for multi-scale tone and detail manipulation. *SIGGRAPH*, 2008.

[6] C. Franklin. Summed-area tables for texture mapping. *SIGGRAPH*, 1984.

[7] L. Grady, T. Schiwietz, and S. Aharon. Random walks for interactive alpha-matting. *VIIP*, 2005.

[8] Y. Guan, W. Cheny, X. Liang, Z. Ding, and Q. Peng. Easy matting: A stroke based approach for continuous image matting. *Eurographics*, 2006.

[9] K. He, J. Sun, and X. Tang. Single image haze removal using dark channel prior. *CVPR*, 2009.

[10] E. Hsu, T. Mertens, S. Paris, S. Avidan, and F. Durand. Light mixture estimation for spatially varying white balance. *SIGGRAPH*, 2008.

[11] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. *SIGGRAPH*, 2004.

[12] A. Levin, D. Lischinski, and Y. Weiss. A closed-form solution to natural image matting. *CVPR*, 2006.

[13] P. Perez, M. Gangnet, and A. Blake. Poisson image editing. *SIGGRAPH*, 2003.

[14] C. Rhemann, C. Rother, and M. Gelautz. Improving color modeling for alpha matting. *BMVC*, 2008.

[15] C. Rhemann, C. Rother, A. Rav-Acha, M. Gelautz, and T. Sharp. High resolution matting via interactive trimap segmentation. *CVPR*, 2008.

[16] C. Rhemann, C. Rother, J. Wang, M. Gelautz, P. Kohli, and P. Rott. A perceptually motivated online benchmark for image matting. *CVPR*, 2009.

[17] Y. Saad. *Iterative methods for sparse linear systems*, page 178. SIAM, 2003.

[18] D. Singaraju, C. Rother, and C. Rhemann. New appearance models for natural image matting. *CVPR*, 2009.

[19] J. Sun, J. Jia, C. Tang, and H. Shum. Poisson matting. *SIGGRAPH*, 2004.

[20] R. Szeliski. Locally adapted hierarchical basis preconditioning. *SIGGRAPH*, 2006.

[21] J. Wang, M. Agrawala, and M. Cohen. Soft scissors: An interactive tool for realtime high quality matting. *SIGGRAPH*, 2007.

[22] J. Wang and M. Cohen. An iterative optimization approach for unified image segmentation and matting. *ICCV*, 2005.

[23] J. Wang and M. Cohen. Image and video matting: A survey. *Foundations and Trends in Computer Graphics and Vision*, 2007.

[24] J. Wang and M. Cohen. Optimized color sampling for robust matting. *CVPR*, 2007.

[25] Y. Zheng and C. Kambhamettu. Learning based digital matting. *ICCV*, 2009.