RANDOM WALKS FOR INTERACTIVE ALPHA-MATTING

Leo Grady and Thomas Schiwietz and Shmuel Aharon
Imaging and Visualization
Siemens Corporate Research
755 College Road East
Princeton, NJ, USA
Leo.Grady@siemens.com

Rüdiger Westermann
Technische Universität München
Lehrstuhl für Informatik 15
Boltzmannstrasse 3
85748 Garching, Germany

Figure 1. By computing the alpha matte as the probability that a random walker first reaches a foreground pixel, a simple, fast algorithm is presented that is capable of producing quality results, even when applied to low-contrast images or objects with difficult or weak boundaries.

ABSTRACT

Interactive, efficient, methods of foreground extraction and alpha-matting are of increasing practical importance for digital image editing. Although several new approaches to this problem have recently been developed, many challenges remain. We propose a new technique based on random walks that has the following advantages: First, by leveraging a recent technique from manifold learning theory, we effectively use RGB values to set boundaries for the random walker, even in fuzzy or low-contrast images. Second, the algorithm is straightforward to implement, requires specification of only a single free parameter (set the same for all images), and performs the segmentation and alpha-matting in a single step. Third, the user may locally fine tune the results by interactively manipulating the foreground/background maps. Finally, the algorithm has an inherit parallelism that leads to a particularly efficient implementation via the graphics processing unit (GPU). Our method processes a 1024×1024 image at the interactive speed of 0.5 seconds and, most importantly, produces high-quality results. We show that our algorithm can generate good segmentation and matting results at an interactive rate with minimal user interaction.

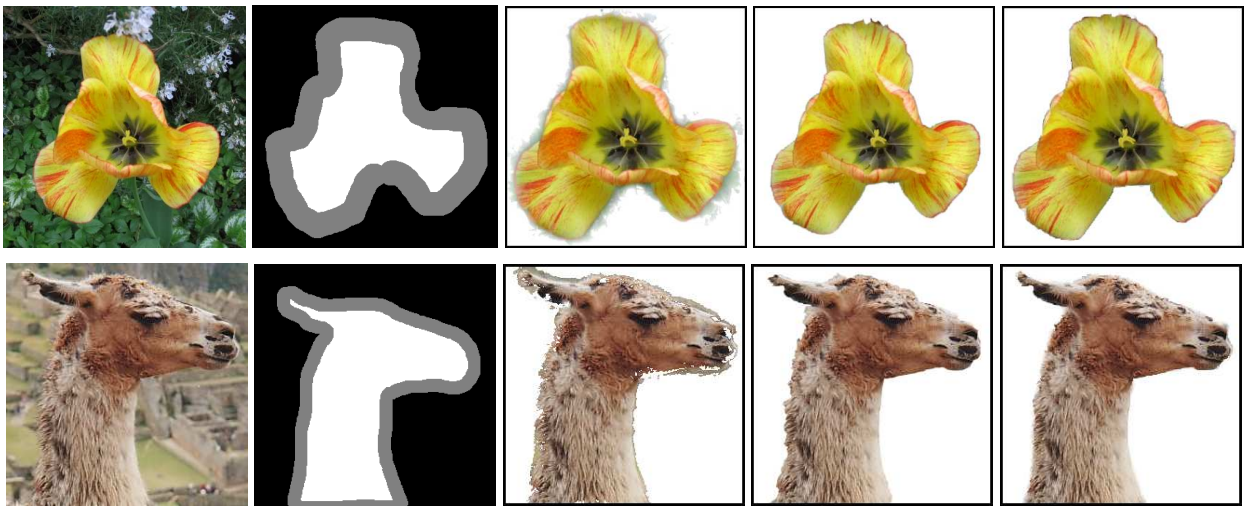
KEY WORDS

Interactive Image Segmentation, Alpha Matting, Random Walks, General Purpose GPU, Image Editing, Object Extraction

1 Introduction

Interactive foreground extraction and alpha-matting of digital images remains a challenging problem. The difficulty of this problem is a result of a simultaneous attempt to minimize user time/interaction, properly handle color, regularize the ill-posedness of the alpha-matting model, provide an efficient algorithm that is feasible to implement and, above all, produce visually pleasing results for arbitrary images.

Recently, the computer vision community has developed several algorithms that produce high-quality results for the "hard" (*i.e.*, binary) image segmentation problem. Although some algorithms, such as the graph cuts approach of Boykov and Jolly [1], produce quality results for hard segmentation tasks on grayscale images, extra care must be taken to extend those techniques to produce an alpha matte on color images [2]. In contrast, the random walker object extraction algorithm presented in [3] directly offers the alpha matting. In this work, we make three extensions to the algorithm presented in [3] to allow for application in the interactive alpha-matting of digital images: 1) A novel utilization of color information, 2) Use of random walker probabilities as the alpha matte, and 3) Details for an implementation on the graphics processor unit (GPU). We show that the random walker algorithm has a straightforward, interactive-speed, implementation, requires specification of only a single free parameter (set consistently for all images), performs the segmentation and alpha-matting in a single step, finds an exact global energy minimum and produces high-quality, visually pleasing, results, even in the presence of low-contrast boundaries and noise. We note also that this technique has straightforward extension



(a) Original (b) Trimap (c) Poisson matting (d) GrabCut (e) Random Walker

Figure 2. A comparison of matting algorithms on two images. a) Original images. b) Trimap, with foreground set to white, background set to black and unknown set to gray. c) Poisson matting is derived with the assumption that the foreground and background intensities are widely separated, which is violated by the llama image but more correct for the flower image. d) GrabCut performs well, but the alpha matting resembles a steep (one-pixel) blur around the boundary. e) Random walker matting. Note that Poisson matting, GrabCut and Random walker were all initialized with a trimap and no additional user-interaction was supplied.

to 3D (*e.g.*, video) or higher dimensional editing problems, although we will limit the remainder of the paper to discussion of the 2D case.

The basic idea of the random walker algorithm is this — Given a user-supplied trimap of the pixels, $P = \{P_f, P_b, P_u\}$, indicating "foreground", "background" and "unknown" regions, we set the α value at each pixel in P_u as the probability that a random walker starting from this location will reach a pixel in P_f before striking a pixel in P_b, when biased to avoid crossing the foreground boundary. Despite the fact that these probabilities may seem prohibitively expensive to compute, it is known [4] that they may be calculated exactly by solving a single system of linear equations. Although a sparse set of linear equations may be efficiently solved with a variety of conventional techniques, several aspects of the problem formulation allow for a particularly efficient solution via a graphics processor unit.

Several properties of the solution to this random walker problem, as demonstrated in [3], make it attractive for use as the alpha-matte. First, it was shown that such an approach will cause the distribution of probabilities (*i.e.*, the matte) to respect low-contrast or even missing boundaries. Second, the computed alpha-matte will vary smoothly, except for clear transitions over an object boundary. Finally, [3] showed that such an algorithm has provable robustness to noise. Furthermore, the algorithm takes just one free parameter (which is fixed for all results in this paper), is straightforward to implement, efficient and produces visually pleasing results.

The alpha matting model may be stated as

$$I = \alpha F + (1-\alpha)B, \tag{1}$$

where α varies between $[0,1]$ and represents the blending coefficient between the foreground object, F, and the background object, B. The α value has been variously interpreted as a *probability* [5] that the pixel belongs to the foreground object, a partial differential equation *solution* [6], or as an *interpolation* [2] between the known foreground and background objects. With the present approach, we take all three viewpoints by formulating α as a probability and showing its mathematical equivalence to a partial differential equation, the solution of which has also been used as an interpolation operator.

The paper is organized as follows: After having stated our approach to alpha-matting, we proceed in Section 2 to review previous approaches to this problem. Section 3 formalizes the algorithm and Section 4 discusses a particularly efficient implementation on commodity graphics hardware. More results follow in Section 5 with the conclusion and future directions.

2 Related Work

A vast literature is devoted to the problems of foreground extraction and alpha-matting. We therefore limit our review to highly successful, ubiquitous and/or mathematically related techniques.

Foreground extraction via **Intelligent Scissors** (a.k.a., magnetic lasso, live wire) [7] is accomplished by user clicks on the foreground boundary. The algorithm finds the object boundary between user clicks by finding the shortest path (via Dijkstra's algorithm) on a weighted graph, with nodes given by pixels and weights set to encourage paths along the foreground boundary. This algorithm performs reasonably well on smooth objects with moderate to high contrast, but requires an excessive number of user clicks for low-contrast images or objects with rough boundary. Furthermore, the intelligent scissors algorithm offers a hard segmentation boundary, requiring an additional algorithm to estimate the alpha-matting.

Given a user specified trimap of foreground, background and unknown pixels, the **Bayesian Matting** algorithm [5], optimizes the α, foreground and background values in (1). First, the color samples for F and B are clustered, and each cluster is fit with an oriented Gaussian distribution. Then, a maximum-likelihood criterion is used to estimate the optimal α values for each pair of foreground and background clusters. This algorithm can generate good matting but only when the foreground and background color distributions are sufficiently well separated and the unknown region is not too large [2]. In addition, the algorithm will perform poorly on a complex scene where two foreground clusters are spatially proximal and also mixed in color space. This algorithm also lacks the ability to refine the solution to improve the results to the user's satisfaction.

Given a trimap, **Poisson matting** [6] makes a smoothness assumption on the foreground and background pixels in order to approximate the alpha value as the solution to the Poisson equation given by

$$\nabla^2 \alpha = \nabla \cdot \left(\frac{\nabla I}{F - B} \right), \tag{2}$$

with Dirichlet boundary conditions on the foreground/background pixels of

$$\alpha_j = \begin{cases} 1 & \text{if } I_j \in P_f, \\ 0 & \text{if } I_j \in P_b, \end{cases} \tag{3}$$

for pixel I_j. For economy of notation, we will employ a single subscript to denote a pixel instead of $I(x,y)$. Although impressive results are shown by Sun *et al.* [6], the images typically have smooth foregrounds and backgrounds (in accordance with the assumptions) and require, at least locally, that the foreground/background intensities are widely separated. In fact, (2) is degenerate if the estimates of F and B are equal at any point in the unknown region, as acknowledged by the authors. Furthermore, if the solution to (2) (*global* Poisson matting) fails to produce a satisfactory matting, a costly user interaction is required to boost the matting in failed regions (*local* Poisson matting), which is reported by the authors to require on the order of ten minutes for images of resolution 600×400. Finally, both the global and local methods of Poisson matting iterate the algorithm, and therefore require both a convergence criterion and are guaranteed only to settle into a local minimum.

The **GrabCut** method introduced by Rother *et al* [2] builds upon the powerful graph cut technique [1] from the computer vision literature to develop a method of interactive foreground extraction and alpha-matting of color images. Given a trimap or a bimap (*i.e.*, only P_b and P_u), the algorithm builds a Gaussian mixture model (GMM) of the RGB image and iteratively alternates between graph cuts and updating the model parameters until a convergence criterion is satisfied. Since graph cuts gives only a binary segmentation, the α values are then estimated by performing a second minimization to find an appropriate rate of interpolation of the α values in a narrow band around the binary segmentation. GrabCuts is efficient, runs at an interactive rate and, for some images, requires no more user interaction than dragging a rectangle over the desired object. However, this algorithm alternates between estimating the segmentation and the GMM parameters, yielding a solution that is not guaranteed to be a global minimum of the target energy functional. Furthermore, the alpha-matting procedure results in what may be described qualitatively as a corona of variable-rate blur around the binary segmentation. Additionally, the approach is quite complex, requiring a convergence criterion for the energy model and the specification of seven free parameters to describe the weighting of energy terms, inter-pixel affinities, the number of Gaussians in the GMM, and four more free parameters to describe the energy model for alpha-estimation.

Since GrabCuts and (global) Poisson matting were convincingly shown to outperform the other methods reviewed, we compare our algorithm to these two techniques in Figure 2 with detailed views shown in Figure 3. Although GrabCuts allows for a less-informative bimap input, all three algorithms were initialized with a trimap in order to provide a fair comparison. Note the trouble with the Poisson matting technique for the low-contrast llama image due to the singularity of (2). Although GrabCuts provides a clean output for both images, the alpha-matting has the appearance of an isotropically blurred binary segmentation found by the iterated graph cuts optimization.

3 Random Walks in Color Space

There are two aspects of the random walks approach described in this work: Using image information to guide the trajectory of the random walker and the analytical computation of the random walker probabilities. We formulate the algorithm on a discrete space (*i.e.*, a graph), where each pixel corresponds to a node and edges are placed between a pixel and its four neighbors in the cardinal directions. Before proceeding, we will fix our notation.

A **graph** [8] consists of a pair $G = (V,E)$ with **vertices**

(a) Poisson (b) GrabCut (c) Random Walker

Figure 3. Detail of the alpha-matting provided by the Poisson, GrabCuts and Random Walker algorithms on the two images of Figure 2.

(nodes) $v \in V$ and **edges** $e \in E \subseteq V \times V$. An edge, e, spanning two vertices, v_i and v_j, is denoted by e_{ij}. A **weighted graph** assigns a value to each edge called a **weight**, which will be nonnegative and real for the present purposes. The weight of an edge, e_{ij}, is denoted by w_{ij}. The **degree** of a vertex is $d_i = \sum w_{ij}$ for all edges e_{ij} incident on v_i. A weight between pixels v_i and v_j induces a probability that a random walker at v_i transitions to v_j as $p_{ij} = \frac{w_{ij}}{d_i}$.

3.1 Manifold learning

In order for the image structure to guide the random walker, a weight must be assigned between neighboring pixels that indicates an affinity for these nodes to be coupled (*i.e.*, to share the same α). A typical (*e.g.*, [2, 1]) function for mapping pixel values to edge weights is

$$w_{ij} = \exp\left(\frac{||z_i - z_j||^2}{\sigma^2}\right), \tag{4}$$

where z_i is a vector representing the RGB color at pixel i and σ is a free parameter. The value of σ is the only free parameter in the algorithm, and its value is set for all results in this paper as $\sigma = \frac{1}{30}$. We assume that the image has been normalized such that $0 \leq ||z_i - z_j||^2 \leq 1$.

Although Rother *et al.* [2] employ a Euclidean norm in (4), such a measure is notoriously unreliable for describing perceptual and object boundaries in RGB color space. Since the purpose of (4) is to distinguish an object boundary as best as possible, we propose to use the recently developed Locality Preserving Projections (LPP) technique of He and Niyogi [9] to define a *conjugate* norm. Recall that a conjugate norm of a vector, z_i, with respect to a matrix, A, denoted $||\cdot||_A$, is given by the inner product

$$\langle z_i, Az_i \rangle = z_i^T A z_i. \tag{5}$$

Examples of conjugate norms are the Euclidean distance (with A equal to the identity matrix) and the Mahalanobis distance (with A equal to the covariance matrix).

He and Niyogi [9] show that LPP compares favorably to a principle components analysis in several respects. Advantages of LPP are linearity, generalization beyond the "training" points and robustness to outliers. The derivation of LPP assumes that the points are connected in a graph. However, in the context of image processing, especially graph-based techniques, a graph is defined naturally based upon the spatial location of the neighboring pixels (*e.g.*, as a 4-connected lattice).

The projections defined by the LPP algorithm are given by the solution to the following generalized eigenvector problem

$$ZLZ^T x = \lambda ZDZ^T x, \tag{6}$$

where Z is the $3 \times N$ matrix with each z_i vector as a column, D is the diagonal matrix defined by $D_{ii} = d_i$ and L is the graph Laplacian matrix given by

$$L_{v_i v_j} = \begin{cases} d_{v_i} & \text{if } i = j, \\ -w_{ij} & \text{if } v_i \text{ and } v_j \text{ are adjacent nodes,} \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $L_{v_i v_j}$ is used to indicate that the matrix L is indexed by pixels v_i and v_j. Denote the solution to the generalized eigenvector problem of (6) by Q, where each eigenvector is a row of Q. We note that computation of (6) is very inexpensive, since the generalized eigenproblem must only be solved for a matrix of size 3×3, regardless of the image size.

Therefore, the solution to (6) is both extremely inexpensive and parameter-free. The conjugate norm we propose in computing the weights of (4) is therefore $||\cdot||_{Q^T Q}$, *i.e.*,

$$w_{ij} = \exp\left(\frac{(z_i - z_j)^T Q^T Q (z_i - z_j)}{\sigma^2}\right). \tag{8}$$

Figure 4 displays the three projected channels (*i.e.*, Qz) that the discrimination is based on, instead of the original RGB channels for the color image.

3.2 Random Walks

It has been known since Kakutani's seminal work [10] that the solution to the random walker problem described above is exactly the solution to the (inhomogeneous) Dirichlet problem from potential theory, given Dirichlet boundary conditions as given in (3). On a graph, this solution may be interpreted as computing the steady-state potentials of an electric circuit where the branch conductances are analogous to the edge weights (*i.e.*, $w_{ij} = 0$ represents infinite resistance) [4]. These probabilities are an exact, steady-state, global minimum to the Dirichlet energy functional $E(\alpha) = \alpha^T L \alpha$, subject to the boundary conditions.

Specifically, L may be decomposed into blocks corresponding to the unknown pixels, P_u, and the known pixels $P_k =$

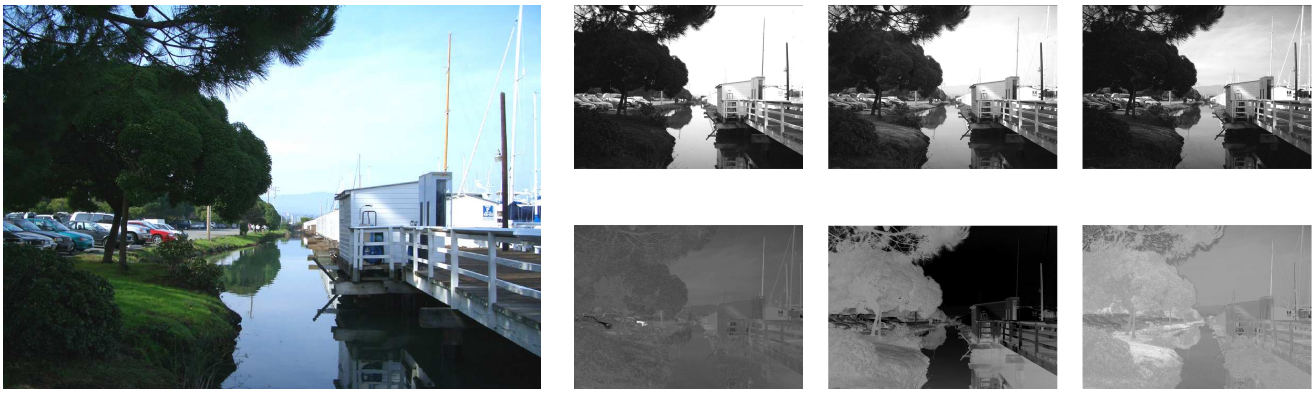


Figure 4. Decomposition of a color image (left) into LPP channels (bottom row) instead of RGB channels (top row). Note how each object takes a roughly uniform color in each of the LPP channels, compared with the RGB images. Consider also that slow gradients in the image (*e.g.*, sky and water) are squashed in LPP-space, but preserved in RGB images. Therefore, by using the conjugate norm defined in (8), we bias our random walkers to avoid crossing boundaries in *LPP space*.

$P_f \cup P_b$ as

$$L = \begin{bmatrix} L_k & R \\ R^T & L_u \end{bmatrix}. \tag{9}$$

Given a vector, m, encoding the boundary conditions of (3), it is shown in [3] that the desired probabilities (*i.e.*, alpha-matte) are the solution to

$$L_u \alpha = -Rm, \tag{10}$$

which is just a sparse, symmetric, positive-definite, system with $|P_u|$ number of equations and the number of nonzero entries less than $5|P_u|$, where $|\cdot|$ denotes cardinality. However, due to the symmetry of L_u and the fact that the diagonal information is redundant (*i.e.*, it is the sum of the off-diagonal entries), storage is only required for $2|P_u|$ values. Since L_u is guaranteed to be nonsingular for a connected graph [8], the solution, α, is guaranteed to exist and be unique. Unlike the solution to (2) from the Poisson matting technique, the solution to (10) is guaranteed to lie in the range $[0,1]$ by the combinatorial analogue of the maximum/minimum principle for continuous harmonic functions. We note that this technique could be easily extended to finding α values for more than two regions (*i.e.*, foreground and background) by solving an additional system of equations (10) for each additional "label" with new boundary conditions (see [3] for details). Finally, we note the similarity of this technique for alpha matting to the successful colorization strategy of Levin *et al.* [11], albeit with a different interpretation of the equations and method for setting edge weights.

4 Implementation

The random walker alpha-matting algorithm may be summarized as follows. Given a user-supplied trimap

1. Solve the 3×3 generalized eigenvector problem in (6) for the eigenvectors, Q.
2. Calculate the edge weights from the LPP-projected RGB values, *i.e.*, (8).
3. Solve the system of equations in (10) for the α values.

If a user is dissatisfied with the solution, the trimap may be updated by user-clicks specifying additional foreground or background points. The figures in this paper show the results of the one-shot solution without further user-interaction.

The first and second steps of the algorithm are trivial to compute. The main computational burden of the random walker algorithm lies in the solution to the system of linear equations. Although many simple, efficient, methods exist to solve a symmetric, positive definite, system of equations, the architecture of the processing unit of commodity graphics hardware (GPU) provides a particularly efficient platform for solution, achieving a running speed of 0.5 seconds on a 1024×1024 image using an ATI X800 XT graphics card. The following section gives additional details of this implementation. However, we stress that a GPU implementation is not required for efficiency. Even a very simple implementation in MATLAB runs the entire algorithm in less than 4 seconds for a 600×400 resolution image.

4.1 GPU implementation

Although it is now well-established that the parallelism of the GPU is powerful for solving systems of linear equations that arise from certain partial differential equations (*e.g.*, [12],[13]), there are several aspects of the present algorithm that make it particularly suited to a GPU implementation: 1) Since most of the image pixels have fixed

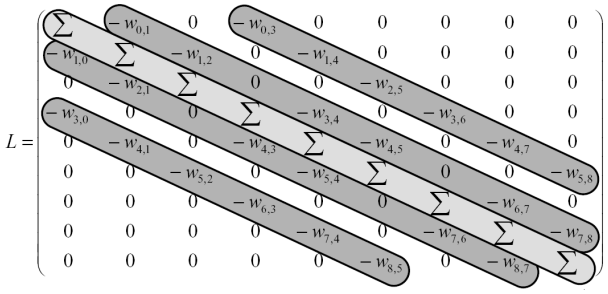


Figure 5. The sparsity structure of the Laplacian matrix for a 2D image has a particularly efficient representation via the GPU, since each of the four secondary diagonals may be stored in the RGBA values of a texture. Since the main diagonal is the negative row sum of the secondary diagonals, no storage is necessary for these values.

α (*i.e.*, $|P_f \cup P_b| >> |P_u|$), the GPU provides a natural method for ignoring these pixels via Z-buffer early termination (effectively performing the block decomposition of (9)), 2) The current solution may be updated at each iteration as a visualization to the user, as seen in the supplemental video, 3) Two RGBA channels may be employed to simultaneously solve for α and $1-\alpha$ (since reversing the boundary conditions modifies only the right hand side of (10)), using the solution from whichever system converges first, and 4) For other applications where the image is modeled as a composition of more than two labels (*i.e.*, foreground/background), these additional labels may also be computed simultaneously by using the additional RGBA channels.

We solve the system of equations (10) on the GPU using the (Jacobi) preconditioned conjugate gradients method. The implementation of a conjugate gradients method requires only two non-trivial operations: a sparse-matrix vector multiply and a vector inner product. The sparse-matrix vector multiply is described below and the vector inner product is described by Bolz *et al.* [12] and Krüger *et al.* [13].

The structure of the Laplacian matrix, described in (7) is well-suited for a texture representation. Figure 5 shows that this matrix has five diagonal bands for a 2D image. The secondary bands contain the edge weights to neighboring pixels (nodes in the graph) and the main diagonal band is the sum of the four secondary bands. Therefore, we need only keep the values on the four secondary diagonals in a four-channel 2D floating texture. Furthermore, using this representation, the Laplacian matrix and the α vector are both kept in 2D textures with size equal to the number of pixels in the image. Therefore, only one rendering pass is needed to calculate a matrix-vector multiply, using a simple shader program:

```
float4 psMultiplyMatrixVector(PosTex5 v) :
COLOR {
    // sample Laplacian matrix
    float4 L = tex2D(sam0, v.TexCoordsC);

    // sample vector
    float4 vecc = tex2D(sam1,
    v.TexCoordsC); // center
    float4 vecu = tex2D(sam1,
    v.TexCoordsU); // upper neighbor
    float4 vecl = tex2D(sam1,
    v.TexCoordsL); // left neighbor
    float4 vecr = tex2D(sam1,
    v.TexCoordsR); // right neighbor
    float4 vecd = tex2D(sam1,
    v.TexCoordsD); // lower neighbor

    // main diagonal is the sum of secondary diagonals: dot(L,1)
    float diag = dot(L,1);

    // multiply matrix by vector
    return diag*vecc - L.x*vecu - L.y*vecl
    - L.z*vecr -
       L.w*vecd;
}
```

Our implementation of this algorithm using DirectX 9.0c on ATI X800 XT takes 0.5s for a 1024×1024 image producing high quality segmentation and matting (see Figure 6).

5 Conclusion

We have presented a new approach to alpha-matting that is easy to implement, achieves a global minimum, has a single free parameter that does not require adjustment and has an efficient solution that provides quality results for low-contrast images or objects with weak or difficult boundaries. Figure 6 demonstrates the results of applying the algorithm to a few images containing objects with moderately difficult (*i.e.*, low-contrast, weak) boundaries.

We note that, since each α is the (weighted) average of its neighboring α values, there will be no local maxima/minima in the alpha distribution. From a practical standpoint, this means that producing a matte for an object of complex topology requires a trimap of complex topology. One approach to overcoming this issue would be to add a regional term in the energy functional to the Dirichlet energy (for which (10) provides a minimum) in a similar manner to Boykov and Jolly [1] or Rother *et al.* [2]. The cost of such an approach would be to require another free parameter (to weight the energy terms) and decrease the present simplicity of the algorithm. In general, we feel that GrabCuts and the present approach produce results of comparable quality. However, the present approach is faster, simpler, and has many fewer free parameters, but at the tradeoff of slightly more user-interaction (*i.e.*, supplying a trimap instead of a bimap).

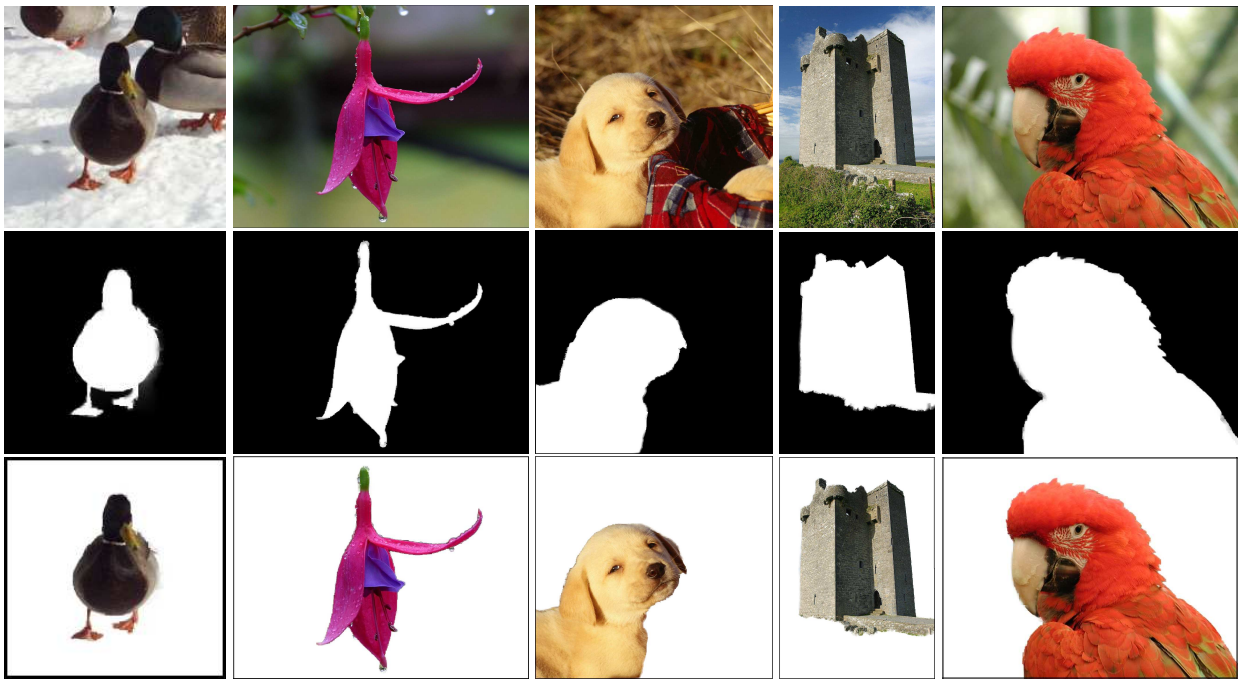


Figure 6. Additional results for a variety of images. Top row: Original images. Middle row: Computed alpha matte. Bottom row: Resulting object cut out. All images were processed with the same value of the free parameter (see text) and required roughly 0.5 seconds to process.

Future work might include additional focus on setting weights, incorporating an intensity prior and application to video editing.

ACKNOWLEDGEMENT

We would like to thank Vladimir Kolmogorov of Microsoft Research for graciously providing images and corresponding GrabCuts results.

References

[1] Y. Boykov and M.-P. Jolly, "Interactive organ segmentation using graph cuts," in *Medical Image Computing and Computer-Assisted Intervention*, Pittsburgh, PA, October 2000, pp. 276–286.

[2] C. Rother, V. Komogorov, and A. Blake, ""GrabCut" — Interactive foreground extraction using iterated graph cuts," in *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2004*, vol. 23, no. 3. ACM, 2004, pp. 309–314.

[3] L. Grady and G. Funka-Lea, "Multi-label image segmentation for medical applications based on graph-theoretic electrical potentials," in *Computer Vision and Mathematical Methods in Medical and Biomedical Image Analysis, ECCV 2004 Workshops CVAMIA and MMBIA*, ser. Lecture Notes in Computer Science, M. Šonka, I. A. Kakadiaris, and J. Kybic, Eds., no. LNCS3117. Prague, Czech Republic: Springer, May 2004, pp. 230–245.

[4] P. Doyle and L. Snell, *Random walks and electric networks*, ser. Carus mathematical monographs. Washington, D.C.: Mathematical Association of America, 1984, no. 22.

[5] Y.-Y. Chuang, B. Curless, D. Salesin, and S. R., "A Bayesian approach to digital matting," in *Proceedings of the CVPR 2001*, vol. 2, 2001, pp. 264–271.

[6] J. Sun, J. Jia, C.-K. Tang, and H.-Y. Shum, "Poisson matting," in *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2004*, vol. 23, no. 3. ACM, 2004, pp. 315–321.

[7] E. Mortensen and W. Barrett, "Interactive segmentation with intelligent scissors," *Graphical Models in Image Processing*, vol. 60, no. 5, pp. 349–384, 1998.

[8] N. Biggs, *Algebraic Graph Theory*, ser. Cambridge Tracts in Mathematics. Cambridge University Press, 1974, no. 67.

[9] X. He and P. Niyogi, "Locality preserving projections," in *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, ser. Advances in Neural Information Processing Systems, Vancouver, Canada, 2003.

[10] S. Kakutani, "Markov processes and the Dirichlet problem," *Proc. Jap. Acad.*, vol. 21, pp. 227–233, 1945.

[11] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," in *Proceedings of ACM SIGGRAPH 2004*, ser. ACM Transaction on Graphics, vol. 23, no. 3, ACM. New York: ACM, August 2004, pp. 689–694.

[12] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid," in *ACM Transactions on Graphics*, ser. SIGGRAPH, vol. 22, no. 3, July 2003, pp. 917–924.

[13] J. Krüger and R. Westermann, "Linear algebra operators for GPU implementation of numerical algorithms," in *ACM Transactions on Graphics*, ser. SIGGRAPH, vol. 22, no. 3, July 2003, pp. 908–916.