PyMatting: A Python Library for Alpha Matting

Thomas Germer[1], **Tobias Uelwer**[1], **Stefan Conrad**[1], **and Stefan Harmeling**[1]

1 Department of Computer Science, Heinrich Heine University Düsseldorf

DOI: 10.21105/joss.02481

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @ziatdinovmax
- @macrocosme

Submitted: 02 July 2020
Published: 13 October 2020

Summary

A fundamental problem of many image processing tasks is the extraction of specific objects from an image in order to place them in a scene of a movie or compose them onto another background. Alpha matting describes the problem of separating the objects in the foreground from the background of an image given only a rough sketch. Besides everyday image editing, alpha matting has been applied to medical image analysis (Fan et al., 2019; Yin, Li, Kanade, & Chen, 2010) and microscopy image restoration (Kanade et al., 2011).

For an image I with foreground pixels F and background pixels B, alpha matting asks to determine opacities α, such that the equality

$$I_i = \alpha_i F_i + (1 - \alpha_i) B_i \tag{1}$$

holds for every pixel i. This problem is ill-posed since, for each pixel, we have three equations (one for each color channel) with seven unknown variables. The implemented methods rely on a trimap, which is a rough classification of the input image into foreground, background and unknown pixels, to further constrain the problem. Subsequently, the foreground F can be extracted from the input image I and the previously computed alpha matte α using a foreground estimation method (Figure 1).

We introduce the PyMatting toolbox for Python which implements various approaches to solve the alpha matting problem. Our library is also able to extract the foreground of an image given the alpha matte. The target audience are researchers of image processing and computer vision. The implementation aims to be computationally efficient and easy to use.

Implemented Methods for Alpha Matting

- Closed-form Matting: Levin, Lischinski, & Weiss (2008) show that assuming local smoothness of pixel colors yields a closed-form solution to the alpha matting problem.
- KNN Matting: Lee & Wu (2011) and Chen, Li, & Tang (2013) use nearest neighbor information to derive closed-form solutions to the alpha matting problem which they note to perform particularly well on sparse trimaps.
- Large Kernel Matting: He, Sun, & Tang (2010) propose an efficient algorithm based on a large kernel matting Laplacian. They show that the computational complexity of their method is independent of the kernel size.
- Random Walk Matting: Grady, Schiwietz, Aharon, & Westermann (2005) use random walks on the pixels to estimate alpha. The calculated alpha of a pixel is the probability that a random walk starting from that pixel will reach a foreground pixel before encountering a background pixel.
- Learning Based Digital Matting: Zheng & Kambhamettu (2009) estimate alpha using local semi-supervised learning. They assume that the alpha value of a pixel can be learned by a linear combination of the neighboring pixels.

Implemented Methods for Foreground Estimation

- Closed-form Foreground Estimation: For given α, the foreground pixels F can be determined by making additional smoothness assumptions on F and B. Our library implements the foreground estimation by Levin et al. (2008).
- Multi-level Foreground Estimation: Furthermore, the PyMatting library implements a novel multi-level approach for foreground estimation (Germer, Uelwer, Conrad, & Harmeling, 2020). For this method, our library also provides GPU implementations using PyCuda and PyOpenCL (Klöckner et al., 2012).

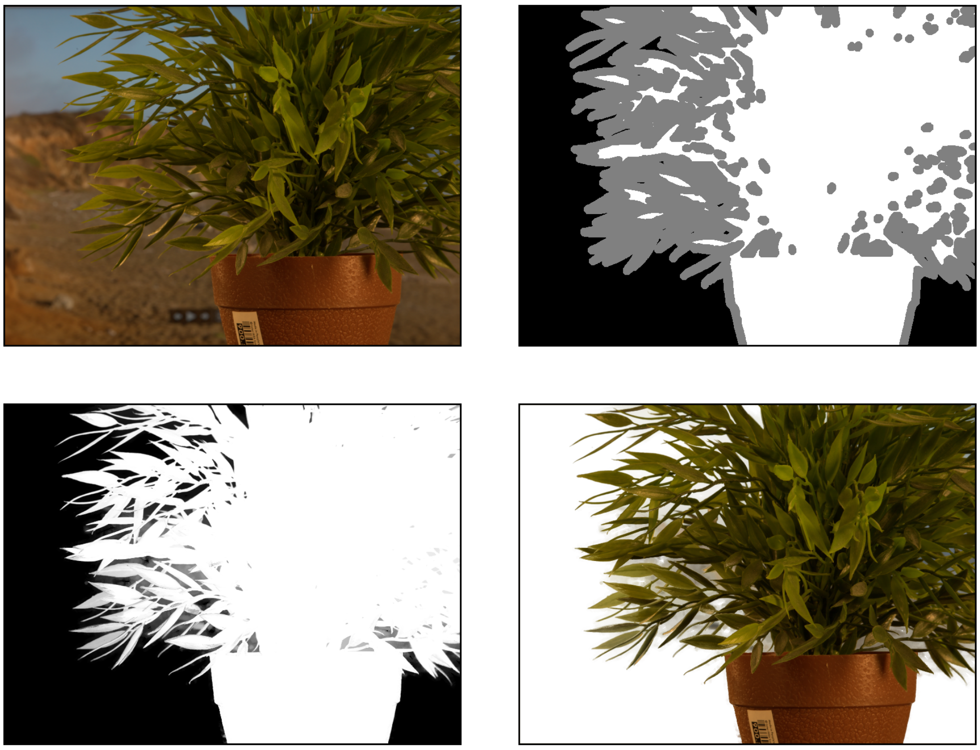

Figure 1: Input image (top left) and input trimap (top right) are used to estimate an alpha matte (bottom left) and a foreground image (bottom right, composed onto a white background) using the Pymatting library. Input image and input trimap are courtesy of Rhemann et al. (2009).

Installation and Code Example

The PyMatting library can be easily installed via `pip3 install pymatting`.

The following code snippet demonstrates the usage of the library:

```
from pymatting import *
image = load_image("plant_image.png", "RGB")
trimap = load_image("plant_trimap.png", "GRAY")
alpha = estimate_alpha_cf(image, trimap)
foreground = estimate_foreground_cf(image, alpha)
cutout = stack_images(foreground, alpha)
save_image("result.png", cutout)
```

The Journal of Open Source Software

The `estimate_alpha_cf` method implements closed-form alpha estimation, whereas the `estimate_foreground_cf` method implements the closed-form foreground estimation (Levin et al., 2008). The `stack_images` method can be used to compose the foreground onto a new background.

More code examples at different levels of abstraction can be found in the documentation of the library.

Performance Comparison

Since all of the considered methods require to solve large sparse systems of linear equations, an efficient solver is crucial for good performance. Therefore, the PyMatting package implements the conjugate gradient method (Hestenes, Stiefel, & others, 1952) together with different preconditioners that improve convergence: Jacobi, V-cycle (Lee & Wu, 2014) and thresholded incomplete Cholesky decomposition (Jones & Plassmann, 1995; Kershaw, 1978).

To evaluate the performance of our implementation, we use the benchmark images from Rhemann et al. (2009). Figure 2 shows the mean squared error between the estimated alpha matte and the ground truth alpha matte, computed over the region of pixels with unknown alpha values. Our results are consistent with the results achieved by the authors' implementations (if available).

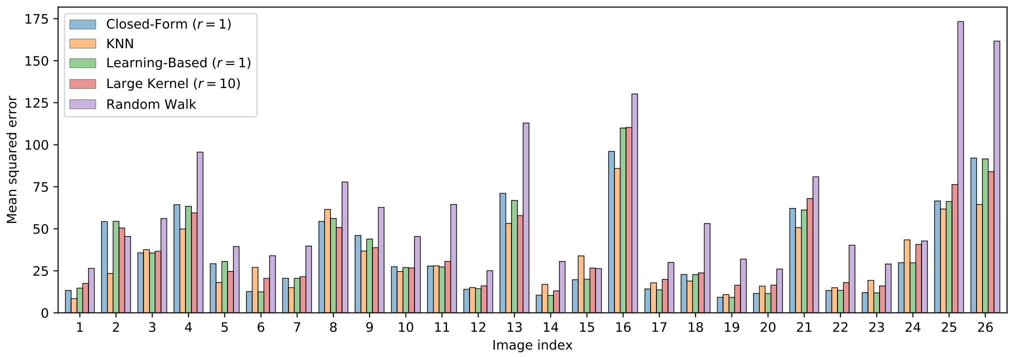

Figure 2: Mean squared error between the estimated alpha matte and the ground truth alpha matte.

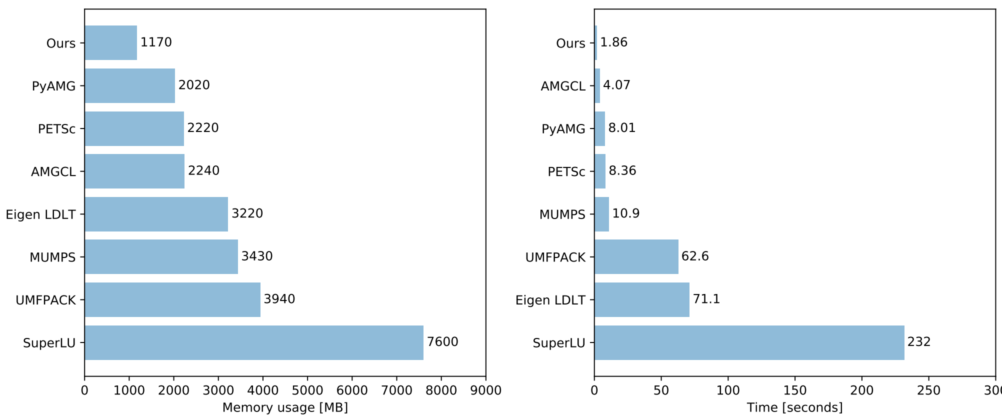

Figure 3: Comparison of peak memory usage in MB (left) and runtime in seconds (right) of our implementation of the preconditioned CG method compared to other solvers for closed-form matting.

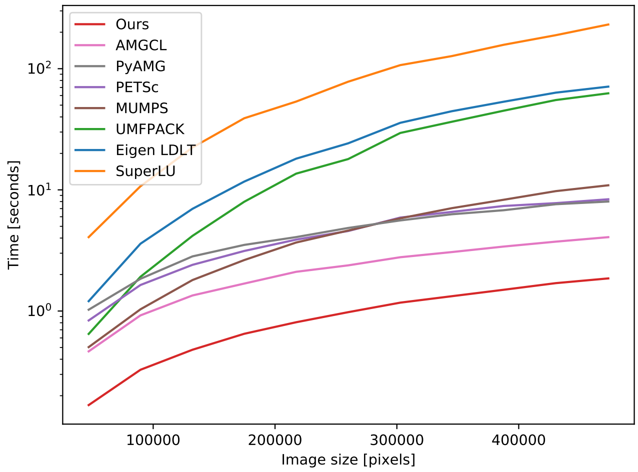

Figure 4: Comparison of runtime for different image sizes.

We compare the computational runtime of our solver with other solvers: PyAMG (Olson & Schroder, 2018), UMFPACK (Davis, 2004), AMGCL (Demidov, 2019), MUMPS (Amestoy, Duff, L'Excellent, & Koster, 2001; Amestoy, Guermouche, L'Excellent, & Pralet, 2006), Eigen (Guennebaud, Jacob, & others, 2010) and SuperLU (Li et al., 1999). Figure 3 and Figure 4 show that our implemented conjugate gradient method in combination with the incomplete Cholesky decomposition preconditioner outperforms the other methods in terms of computational runtime by a large margin. For the iterative solver, we use an absolute tolerance of 10^{-7}, which we scale with the number of known pixels, i.e., pixels that are either marked as foreground or background in the trimap. The benchmarked linear system arises from the matting Laplacian by Levin et al. (2008). Figure 3 shows that our solver also outperforms the other solvers in terms of memory usage. All benchmarks are performed on a high-performance computer with an Intel Xeon Gold 6134 CPU (3.20 GHz) and 196 GB memory running Ubuntu 18.04. For better comparability, only a single thread is used.

Compatibility and Extendability

The PyMatting package has been tested on Windows 10, Ubuntu 16.04 and macOS 10.15.2. New methods can easily be implemented by adding new definitions of graph Laplacian matrices. We plan on continuously extending our library with new methods.

References

Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., & Koster, J. (2001). A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, *23*(1), 15–41. doi:10.1137/S0895479899358194

Amestoy, P. R., Guermouche, A., L'Excellent, J.-Y., & Pralet, S. (2006). Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, *32*(2), 136–156. doi:10.1016/j.parco.2005.07.004

Chen, Q., Li, D., & Tang, C. (2013). KNN matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(9), 2175–2188. doi:10.1109/TPAMI.2013.18

Davis, T. A. (2004). Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, *30*(2), 196–199. doi:10.1145/992200.992206

Demidov, D. (2019). AMGCL: An efficient, flexible, and extensible algebraic multigrid implementation. *Lobachevskii Journal of Mathematics*, *40*(5), 535–546. doi:10.1134/S1995080219050056

Fan, Z., Lu, J., Wei, C., Huang, H., Cai, X., & Chen, X. (2019). A hierarchical image matting model for blood vessel segmentation in fundus images. *IEEE Transactions on Image Processing*, *28*(5), 2367–2377. doi:10.1109/TIP.2018.2885495

Germer, T., Uelwer, T., Conrad, S., & Harmeling, S. (2020). Fast multi-level foreground estimation. Retrieved from http://arxiv.org/abs/2006.14970

Grady, L., Schiwietz, T., Aharon, S., & Westermann, R. (2005). Random walks for interactive alpha-matting. In *Proceedings of VIIP* (Vol. 2005, pp. 423–429).

Guennebaud, G., Jacob, B., & others. (2010). Eigen v3. http://eigen.tuxfamily.org.

He, K., Sun, J., & Tang, X. (2010). Fast matting using large kernel matting laplacian matrices. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 2165–2172). doi:10.1109/CVPR.2010.5539896

Hestenes, M. R., Stiefel, E., & others. (1952). Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, *49*(6), 409–436. doi:10.1201/9781351069397-24

Jones, M. T., & Plassmann, P. E. (1995). An improved incomplete cholesky factorization. *ACM Transactions on Mathematical Software*, *21*(1), 5–17. doi:10.1145/200979.200981

Kanade, T., Yin, Z., Bise, R., Huh, S., Eom, S., Sandbothe, M. F., & Chen, M. (2011). Cell image analysis: Algorithms, system and applications. In *2011 IEEE Workshop on Applications of Computer Vision (WACV)* (pp. 374–381). doi:10.1109/WACV.2011.5711528

Kershaw, D. S. (1978). The incomplete cholesky—conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, *26*(1), 43–65. doi:10.1016/0021-9991(78)90098-0

Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., & Fasih, A. (2012). PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation. *Parallel Computing*, *38*(3), 157–174. doi:10.1016/j.parco.2011.09.001

Lee, P. G., & Wu, Y. (2014). Scalable matting: A sub-linear approach. *arXiv preprint arXiv:1404.3933*.

Lee, P., & Wu, Y. (2011). Nonlocal matting. In *CVPR 2011* (pp. 2193–2200). doi:10.1109/CVPR.2011.5995665

Levin, A., Lischinski, D., & Weiss, Y. (2008). A closed-form solution to natural image matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(2), 228–242. doi:10.1109/TPAMI.2007.1177

Li, X. S., Demmel, J. W., Gilbert, J. R., Grigori, L., Shao, M., & Yamazaki, I. (1999). SuperLU users' guide. *Lawrence Berkeley National Laboratory*. doi:10.2172/751785

Olson, L. N., & Schroder, J. B. (2018). PyAMG: Algebraic multigrid solvers in Python v4.0. Retrieved from https://github.com/pyamg/pyamg

Rhemann, C., Rother, C., Wang, J., Gelautz, M., Kohli, P., & Rott, P. (2009). A perceptually motivated online benchmark for image matting. In *2009 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1826–1833). doi:10.1109/CVPR.2009.5206503

Yin, Z., Li, K., Kanade, T., & Chen, M. (2010). Understanding the optics to aid microscopy image segmentation. In T. Jiang, N. Navab, J. P. W. Pluim, & M. A. Viergever (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2010* (pp. 209–217). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-15705-9_26

Zheng, Y., & Kambhamettu, C. (2009). Learning based digital matting. In *2009 IEEE 12th International Conference on Computer Vision* (pp. 889–896). doi:10.1109/ICCV.2009.5459326