# An Implementation of Bayesian Defogging

Gabriel Schwartz
Drexel University
gbs25@cs.drexel.edu

Ko Nishino
Drexel University
kon@cs.drexel.edu

## Abstract

*In this report, we present an implementation of the Bayesian defogging method described in Nishino* et al. *[6]. In our previous work, we propose a defogging method that factors a foggy image into albedo and depth layers. By recovering these latent layers, we may remove the effects of fog on an image. Here we describe a new implementation that dispenses with a time-consuming graph-cuts optimization in favor of a non-linear optimizer. This solution enables us to apply the same model to larger images in less time.*

## 1. Introduction

First introduced in Kratz and Nishino [5], with further details in Nishino *et al*. [6], we propose to model the foggy image formation process as a combination of statistically independent latent albedo and depth layers. We represent this process using a factorial Markov random field (FMRF), and perform inference on the model to obtain estimates for the albedo and depth layers.

The results from our past work show that the FMRF formulation is a good model for foggy images, achieving high-quality results in challenging scenes. The main drawback of the approach was its long running time.

In this report, we describe an implementation of Bayesian defogging using non-linear optimization in lieu of graph cuts. As shown in Figure 1, this implementation accurately reproduces the original results. The non-linear optimization achieves an accurate result in far less time, enabling application to large images. We have made our implementation available at http://www.cs.drexel.edu/~kon/defog.

## 2. FMRF Inference via Non-Linear Optimization

Our new implementation employs a fast non-linear solver and closed-form gradient computations to find good local minima for the FMRF energy function. We use a non-linear solver that allows us to provide explicit gradient information to aid in fast convergence. Furthermore, we perform the optimization in a continuous space of albedo and depth values as opposed to the discrete range of $[0, 255]$ in the graph-cuts setting. Multiscale optimization is used when necessary to allow initial estimates to be modified by changes with a large spatial extent.

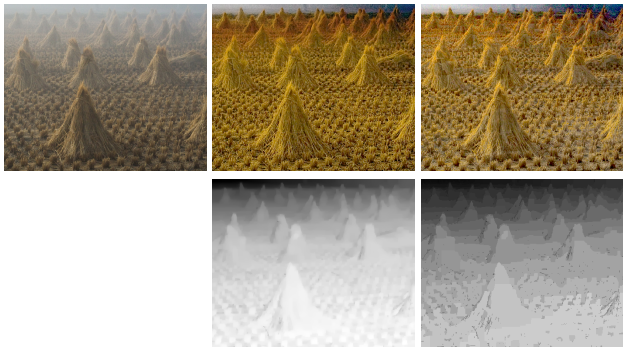

Figure 1. Reproduction of previous results. The columns contain, from left to right: the input image, our factorization result from this method, and the results of the previous method using graph cuts. These results were obtained in 57s for a $465 \times 384$ pixel image, while a graph cuts implementation takes 3700s (over an hour). This comparison shows that we may accurately reproduce the original results much more quickly.

### 2.1. Stability Modifications

In the interests of stability, we slightly modify the equations underlying the original FMRF model. Previously, the likelihood was defined as the following normal distribution:

$$\mathcal{N}\left(\ln\left(1-\mathbf{I}_N\right) \;\middle|\; \ln\left(1-\tilde{\rho}\left(\mathbf{x}\right)\right)+\tilde{d}\left(\mathbf{x}\right), \sigma^2\right),$$

where $\mathbf{I}_N = \frac{\mathbf{I}|\mathbf{L}_\infty|}{\eta \mathbf{L}_\infty}$ with $\eta$ chosen to minimize the number of invalid pixels. Both of these introduce numerical issues in the non-linear optimization process as they are not well-defined near extreme albedo and depth values.

To address stability in the likelihood, we use the original image formation definition of Equation 7 in [6] to obtain the

following :

$$\mathcal{N}\left(\mathbf{I}_N \,\middle|\, \tilde{\rho}\left(\mathbf{x}\right) e^{-\tilde{d}(\mathbf{x})} + \left(1 - e^{-\tilde{d}(\mathbf{x})}\right), \sigma^2\right),$$

with $\mathbf{I}_N = \frac{\mathbf{I}}{\mathbf{L}_\infty} \max\left(\frac{\mathbf{I}}{\mathbf{L}_\infty}\right)$. Even for depth values at $\infty$ or albedo values of 1, the likelihood is still defined.

As a final step, we may remove the parameter $\sigma^2$ from the optimization process. In the non-linear optimization setting, each of the distribution variances for the likelihood and priors acts as a weight on the corresponding term of the energy function. We may obtain an equivalent solution by dividing two of the weights by the third. This eliminates one hyper-parameter without changing global or local minima.

## 2.2. Computing Gradients

The optimal estimates for albedo and depth values minimize the FMRF energy function, which is now:

$$\tilde{\rho}^*, \tilde{d}^* = \arg\min_{\tilde{\rho},\tilde{d}} E$$

$$\begin{aligned} E \;=\; & \left\|\mathbf{C}e^{-\mathbf{D}} + \left(1 - e^{-\mathbf{D}}\right) - \mathbf{I}_N\right\|_2^2 + \\ & \alpha \sum_i \sum_{j \in \Omega_i} \frac{\left|\tilde{\rho}_i - \tilde{\rho}_j\right|^\gamma}{\lambda} + \\ & \xi \sum_i \sum_{j \in \Omega_i} \begin{cases} \left|\tilde{d}_i - \tilde{d}_j\right| & \text{Laplace Prior} \\ \left(\tilde{d}_i - \tilde{d}_j\right)^2 & \text{Gaussian Prior} \end{cases}, \end{aligned} \tag{1}$$

with $\mathbf{C}$ and $\mathbf{D}$ representing the albedo and depth layers with pixels $\tilde{\rho}_i$ and $\tilde{d}_i$ respectively. $\Omega_i$ is the 4-connected neighborhood around pixel $i$. $\alpha$ and $\xi$ are user-specified prior weights.

In order to apply most non-linear optimization algorithms efficiently, we must be able to compute the gradient of the energy function in Equation 1 w.r.t. each of the parameters. In this case, the parameters are the latent variables in the albedo and depth layers. For conciseness, we may define the three terms in the FMRF energy function $E$ as a likelihood term and two prior terms:

$$E = E_L + \alpha E_C + \xi E_D \,. \tag{2}$$

The following equations provide closed form expressions for the gradients of each term of the energy function $E$ w.r.t. parameters $\tilde{\rho}$ and $\tilde{d}$:

$$\begin{aligned} \frac{\partial E_L}{\partial \mathbf{C}} &= 2e^{-\mathbf{D}}\left(\mathbf{C}e^{-\mathbf{D}} + \left(1 - e^{-\mathbf{D}}\right) - \mathbf{I}\right) \\ \frac{\partial E_C}{\tilde{\rho}_i} &= \sum_{j \in \Omega_i} \operatorname{sgn}\left(\tilde{\rho}_i - \tilde{\rho}_j\right) \frac{\gamma \left|\tilde{\rho}_i - \tilde{\rho}_j\right|^{\gamma-1}}{\lambda} \\ \frac{\partial E_L}{\partial \mathbf{D}} &= -2\left(\mathbf{C} - 1\right) e^{-\mathbf{D}}\left(\mathbf{C}e^{-\mathbf{D}} + \left(1 - e^{-\mathbf{D}}\right) - \mathbf{I}\right) \\ \frac{\partial E_D}{\partial \tilde{d}_i} &= \sum_{j \in \Omega_i} \begin{cases} \operatorname{sgn}\left(\tilde{d}_i - \tilde{d}_j\right) & \text{Laplace Prior} \\ 2\left(\tilde{d}_i - \tilde{d}_j\right) & \text{Gaussian Prior} \end{cases}. \end{aligned}$$

The gradients of the likelihood term w.r.t $\mathbf{C}$ and $\mathbf{D}$ can be expressed in an element-wise fashion for the entire image at once; the prior gradient terms are defined on a per-pixel basis.

## 2.3. Multiscale Optimization

Though the depth priors are intended to create large, consistent depth regions (as found in natural images), they are only defined in a 4-connected neighborhood around each pixel. While this poses less of a problem in a graph-cuts setting, non-linear optimization may take many iterations to effect a change with large spatial extent in regions of high texture. We employ a multiscale coarse-to-fine optimization process to solve this problem. Starting at a coarse scale, we may obtain an estimate for depth that satisfies the priors without time-consuming extra optimization iterations. This coarse estimate becomes the initial value for finer scale optimization. We start the coarsest scale with the initial estimate of [6].

## 2.4. Minimization

Though we no longer use graph-cuts, the logic behind the alternating minimization strategy in our original method still applies. We use the initial depth estimate described in [6] and start by estimating albedo. The optimization continues optimizing depth and albedo in alternation.

For the optimization implementation, we rely on the Scipy [4] interface to the L-BFGS-B algorithm [3]. L-BFGS-B is a fast, memory-efficient non-linear optimizer that relies on an approximation of the Hessian matrix obtained from first-order derivatives. If available, our implementation uses Theano [1], a symbolic math expression compiler, to define and evaluate the energy function and its gradients. This allows for transparent use of available GPU hardware.

# 3. Experimental Results

Using the graph-cuts optimization approach, our previous method took a full 22 minutes to perform inference on a $180 \times 120$ pixel image. Our non-linear optimization approach finds a solution for the same image in only 19.4s,

a $68\times$ speedup. On larger images, the speedup is comparable: optimization for a $465 \times 384$ pixel image takes 57s with the non-linear approach compared to 3700s with the original graph-cuts solver.

The original long runtime is not due to the formulation, but rather the choice of optimization algorithm (graph cuts). The computational complexity of graph cuts depends on the chosen max-flow solver [2]. The user may, however, specify any arbitrary energy function; the optimization must thus work with only one parameter at a time. By comparison, BFGS operations can be broken down into simple linear algebra operations and can be accelerated with tuned BLAS libraries and GPU gradient computations.

Figures 2 and 3 compare the results from this implementation with those of our original method. Each column contains the input image, the new optimization results and the original optimization results respectively. Our new recovered albedo and depth images are faithful to the original algorithm. At the same time we may obtain these results significantly faster than with a graph-cuts optimization.

## 4. Software Package

We have made our Python implementation of this method available at `http://www.cs.drexel.edu/~kon/defog`. To run the implementation, you will need Python, Numpy[1], Scipy[2], and optionally, Theano[3]. To factor an image into albedo and depth layers using default parameters, run

```
./defog.py image.png
```

Output is saved to `image_albedo.png` and `image_depth.png`.

If you would like to specify prior weights, airlight color and other options, a list of parameters is available by running

```
./defog.py -h.
```

Parameters used to obtain our results with this code may also be found on the software package website. Please cite this report, as well as [5, 6], in any publications derived from this work.

[1] `http://www.numpy.org`
[2] `http://www.scipy.org`
[3] `http://deeplearning.net/software/theano/`

## References


[1] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU Math Expression Compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010. 2

[2] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 1222–1239, 1999. 3

[3] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A Limited-Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1994. 2

[4] E. Jones, T. Oliphant, and P. P. et al. SciPy: Open source scientific tools for Python, 2001–. 2

[5] L. Kratz and K. Nishino. Factoring Scene Albedo and Depth from a Single Foggy Image. In *ICCV*, pages 1701–1708, 2009. 1, 3

[6] K. Nishino, L. Kratz, and S. Lombardi. Bayesian Defogging. *IJCV*, 98(3):263–278, 2012. 1, 2, 3

Figure 2. Comparison with previous results. The columns contain, from left to right: input image, our new result, and the original graph-cuts solution. These comparisons show that the new approach quickly reproduces the original results.

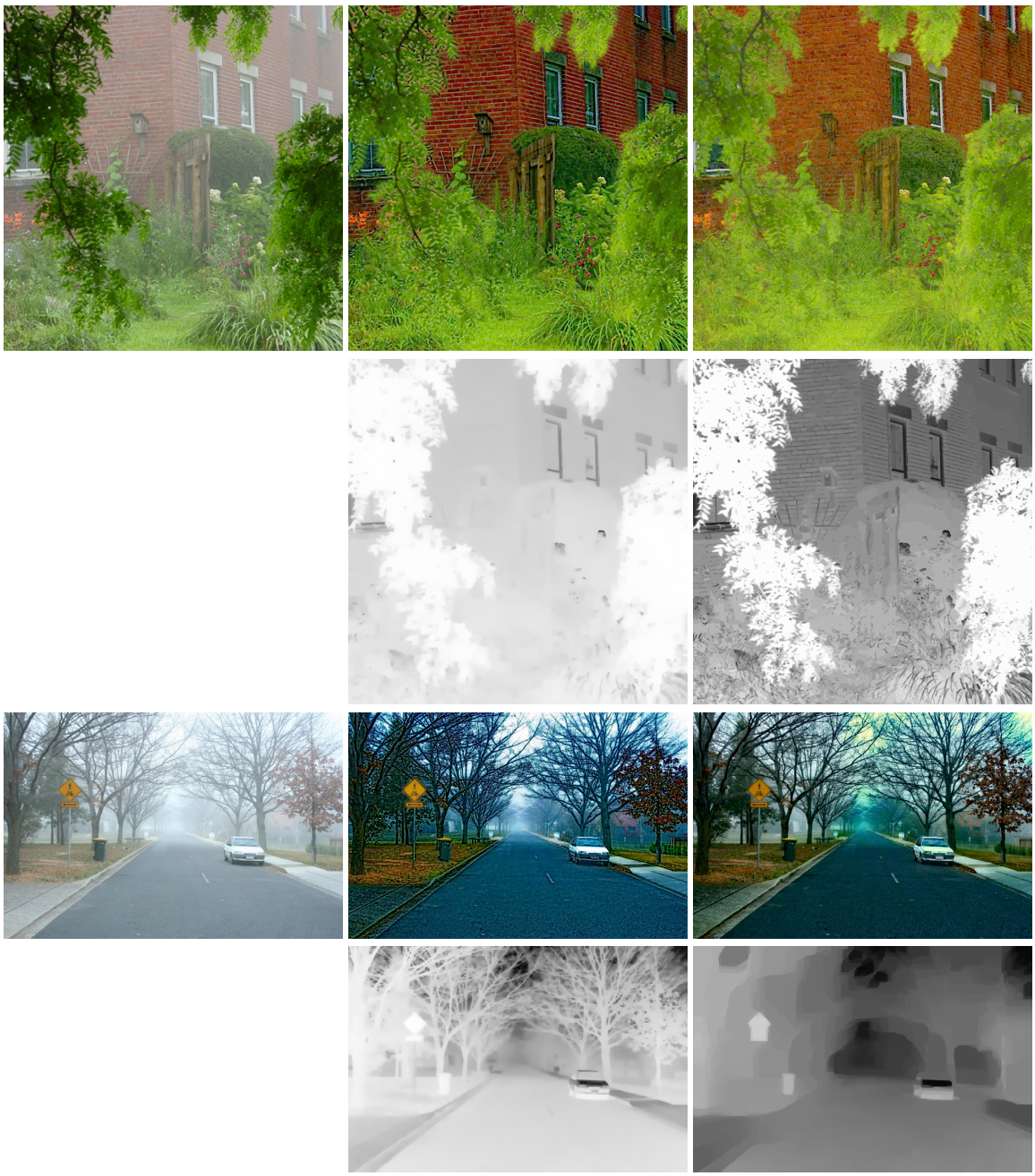

Figure 3. Additional comparisons. As is evident in street scene and the hay cones image above, we may achieve a more complete depth reconstruction as we do not limit depth values to a discrete set.