Supplemental Material for

"Topology-Constrained Layered Tracking with Latent Flow"

Jason Chang and John W. Fisher III

In Section A, we rigorously prove that propagating particle locations with the sampler shown in the paper does not require weight updates. Next, in Section B, we give a more detailed derivation of the Permutation-based Gaussian Inspired Metropolis Hastings shape sampler presented in the paper. In Section C, we validate the filter approximations used in the Gaussian process flow sampling. As stated in the paper, we have additionally developed a heuristic for dynamically learning the smoothness parameters for the Gaussian processes. This heuristic is described in Section D. The edge sharpening process that was shown to slightly improve results in the paper is discussed in Section E. Lastly, we present quantitative flow performance evaluation on the Middlebury dataset in Section F. For additional tracking results and insights, please view the included movie.

A Efficient Particle Filtering without Weight Updates

We denote y as the set of hidden variables, and x as the set of observed variables in a Markov chain. The graphical model representing this relationship is shown in Figure 1. A typical approach for this type of filtering problem is to use a particle filter [3] where at time, t, the distribution, $p(y^t|x^{0:t})$, is represented by a set of weighted samples, $\{y_s^t, w_s^t\}$. Here, w_s^t is the importance weight associated with particle s. The weights are chosen such that expectations over functions of the true distribution can be approximated with weighted sums of functions of the particles. More precisely, if a sample particle, y_s^t is drawn from some density, $q(y^t)$, the weight is chosen to be

$$w_s^t \equiv w(y_s^t) = \frac{p(y_s^t|x_s^t)}{q(y_s^t)}. \tag{1}$$

This set of weights allows one to calculate expectations of some function, $h(\cdot)$, of the desired distribution

$$\frac{1}{N}\sum_{s=1}^{N} w_s^t h(y_s^t) \approx \mathbf{E}_q\left[w(y^t)h(y^t)\right] = \mathbf{E}_{y^t|x^{0:t}}\left[h(y^t)\right]. \tag{2}$$

Particle filtering propagates particles and corresponding weights through time. In typical algorithms, a particle, y_s^t is propagated to the new time point using the distribution over temporal dynamics, $p(y^{t+1}|y^t)$. In this case, the resulting weight updates must reflect the new data likelihood term

$$y_s^{t+1} \sim p(y^{t+1}|y^t), \quad w_s^{t+1} = p(x^{t+1}|y_s^{t+1})\, w_s^t. \tag{3}$$

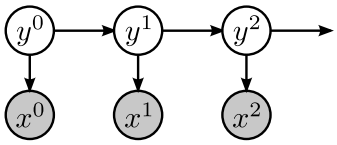


Figure 1: Markov chain that particle filtering can be used on.

As stated in the paper, particle weights decay over time for many likelihood-dominated applications, indicating a poor representation of the desired distribution. Sequential resampling techniques are typically utilized to mitigate this issue. We show here that if particles are propagated with both the observation and prior distributions, weight updates are not needed. This type of sampler can alternatively be thought of as a causal Gibbs sampler.

At time t, assume we have a set of samples y_s^t, drawn from some density $q(y^t)$ and with corresponding importance weights, w_s^t. If at time $t+1$, sample s is drawn from

$$y_s^{t+1} \sim p(y^{t+1}|x^{t+1}, y^t = y_s^t). \tag{4}$$

We show that the previous weights accurately represent the importance weights without updates:

$$\begin{aligned}
\sum_s w_s^t h(y_s^{t+1}) &\approx \mathbf{E}_q\left[w(y^t)h(y^{t+1})\right] && (5)\\
&= \mathbf{E}_q\left[\mathbf{E}_{y^{t+1}|x^{t+1},y^t}\left[w(y^t)h(y^{t+1})|y^t\right]\right] && (6)\\
&= \int_{y^t} q(y^t)\int_{y^{t+1}} p(y^{t+1}|x^{t+1},y^t)w(y^t)h(y^{t+1})dy^{t+1}dy^t && (7)\\
&= \int_{y^t}\int_{y^{t+1}} p(y^t|x^{1:t})p(y^{t+1}|x^{t+1},y^t)h(y^{t+1})dy^{t+1}dy^t && (8)\\
&= \int_{y^t}\int_{y^{t+1}} p(y^{t+1},y^t|x^{1:t+1})h(y^{t+1})dy^{t+1}dy^t && (9)\\
&= \int_{y^{t+1}} p(y^{t+1}|x^{1:t+1})h(y^{t+1})dy^{t+1} && (10)\\
&= \mathbf{E}_{y^{t+1}|x^{1:t+1}}\left[h(y^{t+1})\right]. && (11)
\end{aligned}$$

Therefore, assuming the previous weights were correct importance weights, by sampling particle locations using both prior and data evidence, the weights do not need to be updated and resampling does not need to be performed.

B Permutation-based Gibbs Inspired Metropolis Hastings

The GIMH algorithm [2] attempted to sample from the space of implicit shapes defined over some level set function, $\pi(\phi)$. An assumption in the work was made that the underlying segmentation was only a function of the sign of the level set function. Consequently, it was stated that the target distribution was of the form $\pi(\phi) = f(\text{sgn}(\phi))$. That is, two different level set functions with the same sign at every pixel evaluate to the same likelihood. Unfortunately, this is not a valid distribution over ϕ. If we denote $\ell = \text{sgn}(\phi)$, and observe that ℓ is deterministic conditioned on ϕ, the level set distribution can be expressed as

$$\pi(\phi) = \pi(\phi)\pi(\ell|\phi) = \pi(\ell,\phi) = \pi(\ell)\pi(\phi|\ell). \tag{12}$$

While $\pi(\ell)$ is chosen to be the user-specified distribution, $\pi(\phi|\ell)$ is not defined. This is equivalent to assuming that $\pi(\phi|\ell)$ is uniform for *all* level set functions with $\text{sgn}(\phi) = \ell$. Since there are an infinite number of such level set functions, this distribution is not absolutely integrable and is therefore invalid. We thank Janick Cardinale and Ivo Sbalzarini for noticing errors in the resulting marginal statistics which ultimately lead to the discovery of this oversight.

B.1 Theoretical Sampler

Our Permutation-based GIMH algorithm addresses this issue by explicitly using a random ordering, o, on all pixels. In this section, we discuss the general M-ary sampler. Following this discussion, we explain how it is applied to our layered model. The label assigned to pixel i is denoted ℓ_i, and can take on values in $\{1, \cdots, M\}$. We begin with some definitions.

Definition B.1 (Valid Ordering) *Let o_i be the order index assigned to pixel i. If the vector o is a permutation of the integers 1 to N, then it is a valid ordering. Additionally, if $o_i < o_j$, we say that pixel i is placed before pixel j in the ordering.*

Definition B.2 (Relative Ordering) *Let o define a valid ordering on the pixels, and W be a subset of all the pixels. The ordering of the pixels within W implied by o is defined to be the relative ordering of pixels in W.*

Definition B.3 (Consistent Ordering) *Let o define a valid ordering on the pixels. A consistent ordering is defined to be an ordering that places all pixels with $\ell_i = k$ in a contiguous order for all k. More formally, for consistent orderings, if $a_k = \min_{\{i|\ell_i=k\}} o_i$ and $b_k = \max_{\{i|\ell_i=k\}} o_i$, then $\ell_i = k \quad \forall i \in \{i|o_i \in [a_k, b_k]\}$. If o is a consistent ordering for the labels ℓ, we say that o is consistent with ℓ.*

We note that, in general, a relative ordering need not be consistent. However, if the total ordering on pixels is consistent, the derived relative ordering must also be consistent. We are now ready to define the joint distribution over labels and orderings. We denote this joint target distribution with $\pi(\ell, o) = \pi(\ell)\pi(o|\ell)$. We are free to choose any valid conditional distribution of orderings, and find the that a uniform distribution over all consistent orderings works well. Denoting N_j as the number of pixels with label j, the joint distribution can be expressed as

$$\pi(\ell)\pi(o|\ell) = \frac{\pi(\ell)}{M! \prod_j N_j!}. \tag{13}$$

The first factorial term over M computes the number of permutation of the M possible labels, and the product of factorials computes the number of permutations within a label.

We formulate a Metropolis-Hastings Markov chain Monte Carlo (MH-MCMC) sampler to sample from Equation 13. In MH-MCMC algorithms, one constructs a Markov chain with the stationary distribution being the target distribution. If the chain is simulated for a long period of time, the chain will converge to the stationary distribution, and the values when the process is stopped will correspond to a sample from the target distribution. Conditioned on the previous values, $\ell^{(t)}$ and $o^{(t)}$, we sample values from a user-specified proposal distribution, $q(\hat{\ell}, \hat{o}|\ell^{(t)}, o^{(t)})$. These values are accepted with probability:

$$\Pr\left[\{\ell^{(t+1)}, o^{(t+1)}\} = \{\hat{\ell}, \hat{o}\}\right] = \min\left[1, \frac{\pi(\hat{\ell}, \hat{o})}{\pi(\ell^{(t)}, o^{(t)})} \cdot \frac{q(\ell^{(t)}, o^{(t)}|\hat{\ell}, \hat{o})}{q(\hat{\ell}, \hat{o}|\ell^{(t)}, o^{(t)})}\right] = \min\left[1, H\right]. \tag{14}$$

Otherwise, the old values are kept. The ratio within the minimization is often referred to as the Hastings Ratio, which we denote as H.

We construct a particular proposal distribution here that will correspond to an efficient MH-MCMC sampler. Our proposal distribution is composed of four steps:

1. Select two random labels, k, and l, uniformly.
2. Sample a subset of pixels, $W \sim q(W)$, that only contains pixels with labels k or l.

3. Sample a new set of labels, $\hat{\ell} \sim q(\hat{\ell}|W, \ell^{(t)}, o^{(t)})$ that possibly changes labels within W to k or l and that preserves the consistency of the *relative* ordering.
4. Sample a new consistent ordering uniformly from all orderings that preserves the relative ordering of the pixels in W, and is consistent with the proposed label.

We note that the number of permutations with a subset of indices following a specific order is $\frac{N!}{C!}$, where N is the length of the full permutation, and C is the size of the ordered subset. The probability of generating from this proposal can then be expressed as

$$q(\hat{\ell}, \hat{o}|\ell^{(t)}, o^{(t)}) = \left[\frac{2}{M(M-1)}\right] \cdot \left[q(W)\right] \cdot \left[q(\hat{\ell}|W, \ell^{(t)}, o^{(t)})\right] \cdot \left[\frac{2}{M!} \cdot \frac{\hat{N}_{W_k}!\hat{N}_{W_l}!}{\hat{N}_k!\hat{N}_l! \prod_{j\neq k,l} N_j!}\right], \quad (15)$$

where N_{W_k} is the number of pixels of within W with label $\ell^{(t)} = k$, and $\hat{N}_{W_k}$ is the number of pixels within W with label $\hat{\ell} = k$. The resulting Hastings ratio can be expressed as

$$H = \frac{\pi(\hat{\ell})\frac{1}{M!\prod_j \hat{N}_j!}}{\pi(\ell^{(t)})\frac{1}{M!\prod_j N_j!}} \cdot \frac{\frac{2}{M(M-1)} \cdot q(W) \cdot q(\ell^{(t)}|W, \hat{\ell}, \hat{o}) \cdot \frac{2}{M!} \cdot \frac{N_{W_k}!N_{W_l}!}{N_k!N_l!\prod_{j\neq k,l} N_j!}}{\frac{2}{M(M-1)} \cdot q(W) \cdot q(\hat{\ell}|W, \ell^{(t)}, o^{(t)}) \cdot \frac{2}{M!} \cdot \frac{\hat{N}_{W_k}!\hat{N}_{W_l}!}{\hat{N}_k!\hat{N}_l!\prod_{j\neq k,l} N_j!}} \quad (16)$$

$$= \frac{\pi(\hat{\ell})N_k!N_l!}{\pi(\ell^{(t)})\hat{N}_k!\hat{N}_l!} \cdot \frac{q(\ell^{(t)}|W, \hat{\ell}, \hat{o}) \cdot N_{W_k}!N_{W_l}!\hat{N}_k!\hat{N}_l!}{q(\hat{\ell}|W, \ell^{(t)}, o^{(t)}) \cdot \hat{N}_{W_k}!\hat{N}_{W_l}!N_k!N_l!} \quad (17)$$

$$= \frac{\pi(\hat{\ell})N_{W_k}!N_{W_l}!}{\pi(\ell^{(t)})\hat{N}_{W_k}!\hat{N}_{W_l}!} \cdot \frac{q(\ell^{(t)}|W, \hat{\ell}, \hat{o})}{q(\hat{\ell}|W, \ell^{(t)}, o^{(t)})} \quad (18)$$

Consequently, if we choose the proposal of labels in Step 3 to be

$$q(\hat{\ell}|W, \ell^{(t)}, o^{(t)}) \propto \pi(\hat{\ell})\frac{1}{\hat{N}_{W_k}!\hat{N}_{W_l}!}, \quad (19)$$

the Hastings ratio evaluates to one, and every proposed sample is accepted. Also, similar to GIMH, because proposed labels must preserve the relative ordering of the pixels in W, there exist only $|W| + 1$ possible label moves. Since the number of moves is linear in the size of W, they can be sampled directly by simply enumerate the possible moves.

It is interesting to note that this relationship holds for any data-independent proposal distribution of the subset, W. In particular, in the binary case ($M = 2$), if W is chosen to be a random single pixel, PGIMH simplifies to the typical Gibbs sampler since the denominator of Equation 19 evaluates to $0! \cdot 1! = 1$. Thus, PGIMH is essentially a generalization of Gibbs sampling that allows larger local moves.

B.2 An Efficient Implementation

The previous section described an algorithm for correctly sampling from target distributions. However, the computational complexity of a naïve implementation can be quiet bad. We note that the GIMH algorithm has computational complexity $\mathcal{O}(|W| \log |W|)$ for each iteration. In PGIMH, Step 1 and 2 can be computed in constant time (depending on the choice of $q(W)$). Step 3 requires sorting the pixels in W which is $\mathcal{O}(|W| \log |W|)$ followed by enumerating the possible moves which is $\mathcal{O}(|W|)$. Finally, Step 4 requires one to sample a new consistent order which takes $\mathcal{O}(N)$ time, where $N \gg |W|$. The naïve implementation exhibits $\mathcal{O}(|W| \log |W| + N)$ complexity, which is clearly undesirable. In this section, we describe an exact implementation of the above algorithm that performs a proposal in $\mathcal{O}(|W|)$.

It is well known in the MCMC sampling literature that one can mix any combination of valid proposal distributions while still preserving the stationary distribution of the chain. We consider mixing the above sampler with a Gibbs iteration to sample an ordering conditioned on the labels. While this may seem like it adds complexity to the model, it actually allows us to simplify the implementation. We draw on the following key observation: when a naïve PGIMH iteration is preceded by a Gibbs iteration that samples a consistent ordering, the relative ordering of the pixels within W will be uniformly distributed over all consistent relative orderings. Thus, this iterative procedure can be exactly reproduced with the method described in Algorithm 1. We note that because a random relative ordering is sampled at each iteration of

Algorithm 1 An iteration of sampling $\pi(\ell)$ via PGIMH

1. Select two random labels, k, and l, uniformly.
2. Sample a subset of pixels, $W \sim q(W)$, that contains only contains pixels with labels k or l.
3. Sample a consistent relative ordering of pixels in W uniformly using a Knuth shuffle [4] on each W_k and W_l.
4. Sample a new set of labels, $\hat{\ell} \sim q(\hat{\ell}|W, \ell^{(t)}, o^{(t)})$ that possibly changes labels within W to k or l and that preserves the consistency of the *relative* ordering from the previous step.

PGIMH, the explicit ordering, o, does not actually need to be maintained. Additionally, since the Knuth shuffle can be performed in $\mathcal{O}(|W|)$ time, and the total ordering does not need to be updated, the overall complexity of an iteration is now $\mathcal{O}(|W|)$.

W is often selected by first sampling a random circle center and a random radius around it (from some pre-specified range of valid radii). W is then chosen as the subset of pixels in the circle that have labels k and l. When sampling from a binary distribution, Algorithm 1 can be simplified by omitting the subset selection and setting W to be the entire circle of pixels.

In the layered model presented in the paper, each layer maintains a binary support image. Thus, each iteration starts by first selecting a random layer, followed by an iteration of PGIMH. This procedure can be repeated tens of thousands of times in less than a second on a single core.

B.3 Topology Constraints

As stated in the paper, imposing hard topology constraints on the sampler is a simple extension that follows straightforwardly from the work of [2]. When the possible label changes are enumerated, a simple topology check is performed to find whether the particular change is an allowable topology. If it is not, the move is simply assigned zero probability in $\pi(\hat{\ell})$, and Equation 19 takes care of the rest. For more details on how the topology checks are implement, please refer to [2] and our publicly available source code.

C Approximate Inference

In this section, we show the approximations used to perform the sampling-based inference in the algorithm.

C.1 Approximate Filter-Based Gaussian Process Sampling

We first describe the approximate sampler for the the Gaussian process flow. Assume Σ is the covariance matrix resulting from a stationary covariance kernel, k, so that each component of the covariance can be written as the difference of the locations:

$$\Sigma_{i,j} = k(i-j). \tag{20}$$

Then, the result of multiplying a vector, x, by Σ (i.e. $y = \Sigma\, x$) can be calculated with the convolution of k and x:

$$y_i = [\Sigma\, x]_i = \sum\nolimits_j \Sigma_{i,j} x_j = \sum\nolimits_j k(i-j) x_j, \tag{21}$$

where $[\cdot]_i$ denotes the i^{th} component of the resulting vector.

We now consider the iterative approximate inference technique used to sample $g|f$, where we have omitted the layer subscript m and the time superscript t for notational convenience. As stated in the paper, the distributions of these random variables are

$$p(g) = \mathcal{N}(g\ ;\ 0, \Sigma_g) \tag{22}$$

$$p(f|g) = \mathcal{N}\left(f\ ;\ g, \sigma_f^2 \mathrm{I}\right). \tag{23}$$

By noting that the observation space and inference space are exactly the same, we can manipulate the typical Gaussian process regression [7] to get

$$p(g|f) = \mathcal{N}\left(g\ ;\ \mu_g^*, \Sigma_g^*\right), \tag{24}$$

$$\mu_g^* = \Sigma_g \left[\Sigma_g + \sigma_f^2 \mathrm{I}\right]^{-1} f\,, \qquad \Sigma_g^* = \Sigma_g - \Sigma_g \left[\Sigma_g + \sigma_f^2 \mathrm{I}\right]^{-1} \Sigma_g \tag{25}$$

Using the filtering approximation of Equation 21, the mean vector can be approximated with

$$\mu_g^* \approx h_\mu * f, \tag{26}$$

where h_μ is the covariance kernel corresponding to the covariance matrix $\Sigma_g \left[\Sigma_g + \sigma_f^2 \mathrm{I}\right]^{-1}$. We note that while Equation 21 is exact, this expression is only an approximation because the kernel corresponding to the covariance matrix is not stationary. The approximation is exact in the limiting case of the data, x, having infinite extent. Consequently, the approximation degrades near image boundaries. We find this kernel by using the Fourier domain and three additional facts: (1) convolutions correspond to multiplications in the Fourier domain; (2) the identity matrix corresponds to an all-pass filter; and (3) inverting the matrix before multiplying corresponds to dividing in the Fourier domain. This results in the following filter

$$h_\mu = \mathcal{F}^{-1}\left\{\frac{\mathcal{F}\{k\}}{\mathcal{F}\{k\} + \sigma_f^2}\right\}, \tag{27}$$

where $\mathcal{F}\{\cdot\}$ and $\mathcal{F}^{-1}\{\cdot\}$ denote the Fourier and inverse Fourier transform. We approximate this filter by using discrete Fourier transforms.

While this shows how to find the mean of the posterior Gaussian process, we need to be able to *sample* from the posterior. We first remind the reader of the useful property of Gaussian random variables:

$$g \sim \mathcal{N}\left(\mu_g^*, \Sigma_g^*\right) \equiv \mu_g^* + \mathcal{N}\left(0, \Sigma_g^*\right). \tag{28}$$

That is, the randomness in g can be completely captured by the second term involving the covariance. Additionally, it is well known that if $y \sim \mathcal{N}(0, \mathrm{I})$ (i.e. a vector of standard normals), then multiplying this vector by a matrix A results in correlated Gaussian variables:

$$Ay = \mathcal{N}\left(0, AA^\top\right). \tag{29}$$

While this relationship is most often used to generate multivariate normals by setting A to be the Cholesky decomposition of the covariance matrix, the identity holds for any matrix, A. We first find the corresponding filter that is equivalent to multiplying a vector with Σ_g^* as

$$h_{\Sigma,2} = \mathcal{F}^{-1}\left\{\mathcal{F}\{k\} - \frac{\mathcal{F}\{k\}^2}{\mathcal{F}\{k\} + \sigma_f^2}\right\}. \tag{30}$$

Then using Equation 29, we choose a particular A so that it is symmetric, resulting in:

$$h_{\Sigma} = \mathcal{F}^{-1}\left\{\sqrt{\mathcal{F}\{k\} - \frac{\mathcal{F}\{k\}^2}{\mathcal{F}\{k\} + \sigma_f^2}}\right\}. \tag{31}$$

Using this filter, we can approximately generate a sample from the Gaussian process of Equation 23 with:

$$[h_\mu * f] + \mathcal{N}(0, \mathrm{I}) * h_\Sigma \tag{32}$$

C.2 Approximate Marginalization of Independent Flow

Equations 22-25 in the paper briefly describe the approximation used to marginalize out the independent flow f. We describe this derivation in more detail here.

$$p(\ell_m^t | g_m^t, \ell_{\setminus m}^t, \ell^{t-1}, a_m^{t-1}, x^t, z^t) \tag{33}$$
$$\propto p(x^t, \ell_m^t | g_m^t, \ell_{\setminus m}^t, \ell_m^{t-1}, a_m^{t-1}, z^t) \tag{34}$$
$$= p(x^t | \ell^t, g_m^t, \ell_m^{t-1}, a_m^{t-1}, z^t) p(\ell_m^t | g_m^t, \ell_m^{t-1}, a_m^{t-1}, z^t) \tag{35}$$
$$= \int p(f_m^t | g_m^t) p(\ell_m^t | f\ell_m^{t-1}) p(a_m^t | f a_m^{t-1}) p(x^t | \ell^t, a^t, z^t) df_m^t \tag{36}$$
$$= Q_L(\ell_m^t) \prod_i \int p(f_{m,i}^t | g_{m,i}^t) Q_S(\ell_{m,i}^t | f\ell_{m,i}^{t-1}) p(a_{m,i}^t | f a_{m,i}^{t-1}) p(x_i^t | \ell_i^t, a_i^t, z^t) df_m^t \tag{37}$$
$$= Q_L(\ell_m^t) \prod_i \int \mathcal{N}(f_{m,i}^t; g_{m,i}^t, \sigma_f^2) Q_S(\ell_{m,i}^t | f\ell_{m,i}^{t-1}) p(a_{m,i}^t | f a_{m,i}^{t-1}) p(x_i^t | \ell_i^t, a_i^t, z^t) df_m^t \tag{38}$$

We note that evolving an image (e.g. a) with a flow (e.g. f) can be expressed as

$$f a_i = a_{i+f_i} = a_{i+g_i+f_i-g_i} = g a_{i+f_i-g_i}. \tag{39}$$

Using this relationship, we can express the following

$$p(\ell_m^t | g_m^t, \ell_{\setminus m}^t, \ell^{t-1}, a_m^{t-1}, x^t, z^t) \tag{40}$$
$$\propto Q_L(\ell_m^t) \prod_i \int \mathcal{N}(f_{m,i}^t; g_{m,i}^t, \sigma_f^2) Q_S(\ell_{m,i}^t | g\ell_{m,i+f_i-g_i}^{t-1}) p(a_{m,i}^t | g a_{m,i+f_i-g_i}^{t-1}) p(x_i^t | \ell_i^t, a_i^t, z^t) df_m^t \tag{41}$$
$$= Q_L(\ell_m^t) \prod_i \int \mathcal{N}(f_{m,i}^t - g_{m,i}^t; 0, \sigma_f^2) Q_S(\ell_{m,i}^t | g\ell_{m,i+f_i-g_i}^{t-1}) p(a_{m,i}^t | g a_{m,i+f_i-g_i}^{t-1}) p(x_i^t | \ell_i^t, a_i^t, z^t) df_m^t \tag{42}$$
$$= Q_L(\ell_m^t) \prod_i \int \mathcal{N}(j; 0, \sigma_f^2) Q_S(\ell_{m,i}^t | g\ell_{m,i+j}^{t-1}) p(a_{m,i}^t | g a_{m,i+j}^{t-1}) p(x_i^t | \ell_i^t, a_i^t, z^t) dj \tag{43}$$

We denote $\mathcal{L}^t_{m,i}(j)$ the same as in the paper:

$$\mathcal{L}^t_{m,i}(j) = Q_S(\ell^t_{m,i}|g^{t-1}_{m,i+j})p(a^t_{m,i}|ga^t_{m,i+j})p(x^t_i|\ell^t_i, a^t_i, z^t). \tag{44}$$

Using this notation, and making a discrete approximation to the integral allows us to approximate the likelihood as

$$p(\ell^t_m|g^t_m, \ell^t_{\setminus m}, \ell^{t-1}, a^{t-1}_m, x^t, z^t) \tag{45}$$

$$\propto Q_L(\ell^t_m) \prod_i \int \mathcal{N}(j; 0, \sigma^2_f)\mathcal{L}^t_{m,i}(j)dj \tag{46}$$

$$\approx Q_L(\ell^t_m) \prod_i \sum_j \mathcal{N}(j; 0, \sigma^2_f)\mathcal{L}^t_{m,i}(j) \tag{47}$$

$$= Q_L(\ell^t_m) \prod_i \sum_j h_f(j)\mathcal{L}^t_{m,i}(j), \tag{48}$$

which is exactly Equation 25 in the paper.

D Flow Parameter Learning

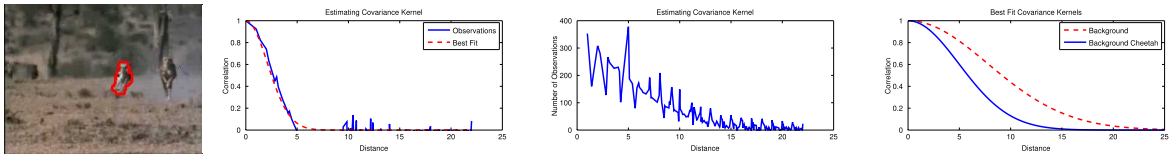
As stated in the paper, the smoothness constraints on the flow field can greatly affect the tracking results. If we are tracking an object moving under rigid transformations, we may desire to constrain the flow to be very smooth, whereas the flow for a deformable object may need to be more loosely constrained. There are three parameters to estimate, the variance of the covariance kernel, ρ^2_g, the absolute variance scaling for the covariance of g, σ^2_g, and the variance of the independent flow, σ^2_f. Although we did not state this in the paper, we can express the covariance of g as $\Sigma_{i,j} = \sigma^2_g \exp\left[-\frac{(i-j)^2}{2\rho^2_g}\right]$.

We use the optical flow measurement of [5] as an initial guess to the flow, and design a heuristic for fitting flow parameters to the measured optical flow. Given the optical flow for each layer, m, and its corresponding support, we begin by estimating the optimal ρ^2_g for each layer. We sample random pairs of locations, and for each pair, we record the distance between the points and their corresponding values. We then find the empirical correlation of these observations as a function of distance, $\rho(d)$. We then minimize the L_2 norm of the difference between the empirical covariances and the parametrized squared exponential kernel, weighted by the number of observations for each distance. We note that empirical correlations may be negative even though the underlying Gaussian process is always attractive. Thus, we ignore negative empirical correlations. We find the best ρ_g from a discrete set of possible values by minimizing

$$\arg\min_{\rho_g} \sum_d \mathbb{1}[\rho(d) > 0] \left(N_d\rho(d) - \exp\left[-\frac{d^2}{2\rho^2_g}\right]\right)^2, \tag{49}$$

where N_d is the number of observations at distance d apart. ρ_g should be tuned to how much a particular object can deform. An example of the best fit covariance kernel is shown in Figure 2b. Once we learn that an object can evolve with a very unsmooth flow field, we should allow it to do so in future frames as well. Thus, we set this parameter at time t to be the minimum value of the previously used smoothness versus the value that optimizes Equation 49 for the current frame. Similarly, the current optimal value is taken to be the minimum of Equation 49 for the x- and y-directional flow.

Because users may want to track only one object of many, a layer may sometimes contain multiple moving objects. For example, consider the frame shown in Figure 2a, where the ground truth annotation only tracks the deer and the cheetah on the right is marked as the background. While the actual



(a) Original (b) FG Fitting (c) FG Observation Count (d) BG Fitting

Figure 2: (a) Annotation to track a deer. Notice how the cheetah on the right is grouped with the background. (b) The optimal fit for the foreground region overlaid with observations. (c) The number of observations of each distance for the foreground region. (d) The optimal covariance kernel for the entire background region and the cheetah contained in the background.

background may have very smooth motions, the untracked cheetah may evolve with a very deformable motion. Additionally, because these two objects are represented with one layer, the necessary flow field for this composited layer must be adequately loose to allow the modeling of both the background and the cheetah. The observations of the entire layer, however, will be dominated by pairs of pixels in the background rather than the cheetah. We therefore take random circles of radius 100 throughout the support of the layer, and find the corresponding optimal ρ_g for each circle. The optimal values for the entire background and for the cheetah are shown in Figure 2d. The resulting optimal ρ_g for the current time frame is then chosen as the minimum of all ρ_g values found in all circles for each layer.

Conditioned on this covariance kernel, we then find the optimal absolute variance, σ_g^2, and the variance of the independent flow, σ_f^2, by finding the values that minimize the L_2 norm between the mean flow field and the optical flow. We note that the actual value found in this step does not seem to affect results very much; however, we do find that we must use the same σ_f^2 for all layers. If different values are used, the layers with a larger value of σ_f^2 tend to have higher likelihood. We believe that this is due to the marginalization of the independent flow, allowing larger values to explain more observation.

E Edge Sharpening

As mentioned in the paper, we sharpen the edges of frames prior to tracking to slightly improve results. This improvement is due to the fact that we do not explicitly model edge effects and motion blur. While this image acquisition phenomenon could potentially be incorporated in the model, we find that a simple procedure for fixing edges works well. We run each frame through three Sobel filter (one for each color channel). Then, we threshold each image using the threshold described in [6]. We define an edge pixel as a pixel that is declared to be an edge in any of the color channels. Finally, we set each edge pixel to the color of the nearest non-edge pixel. An example of this process is shown in Figure 3.

F Middlebury Flow Evaluation

As stated in the paper, while our method does infer a dense flow field, the purpose of the flow is not to be accurate on a subpixel level. Rather, the particular Gaussian process formulation was chosen for purposes of efficient inference in object tracking. Any realization of the flow will not be locally smooth because of the independence assumption in the composite flow field.

Regardless, we show quantitative results on the Middlebury optical flow dataset [1] for our algorithm. We note that in our actual formulation, we assume that objects have been tracked. In the Middlebury dataset, because other algorithms do not have access to this segmentation data, we also do not use it.

Figure 3: Edge sharpening results. Original image (left), edge detection (middle), edge sharpened image (right). Bottom row shows a detail of the image.

Video	Without Initialization	With Initialization
Dimetrodon	0.6864	0.3195
Grove2	1.0042	0.4917
Grove3	1.5564	1.0516
Hydrangea	0.5151	0.4302
RubberWhale	0.4103	0.4053
Urban2	6.1440	0.8631
Urban3	4.5784	0.8631
Venus	1.1397	0.6317

Table 1: Average endpoint error for training set of Middlebury dataset [1].

Rather, we treat the entire image as one layer and hope that the independent flow can capture the necessary discontinuities. Additionally, while our method is designed to track objects moving in natural scenes, the frames from [1] are all synthetically created or pictures in a laboratory setup. As such, their motion vectors are only a few pixels in magnitude as compared to the tens of pixels common in natural scenes. Estimated flow using our algorithm with the optical flow initialization of [5] and initialization with zero flow are evaluated quantitatively in Table 1 and shown in Figure 4. The flow field for each sequence is calculated as the mean flow over 100 samples, and each result is computed using the same set of parameters. We note that from Figure 4, it seems like the flow near the center of objects is estimated quite well while the flow near object boundaries is overly smoothed. This is a result of treating the entire scene as one layer. Additionally, we find that using optical flow as an initialization can greatly help the inference scheme for large regions of similar color.

References

[1] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, and R. Szeliski. A database and evaluation methodology for optical flow. In *ICCV*, 2007.

[2] J. Chang and J. W. Fisher III. Efficient topology-controlled sampling of implicit shapes. In *ICIP*, 2012.

[3] M. Isard and A. Blake. A mixed-state condensation tracker with automatic model-switching. In *ICCV*, 1998.

[4] D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, Boston, 1969.

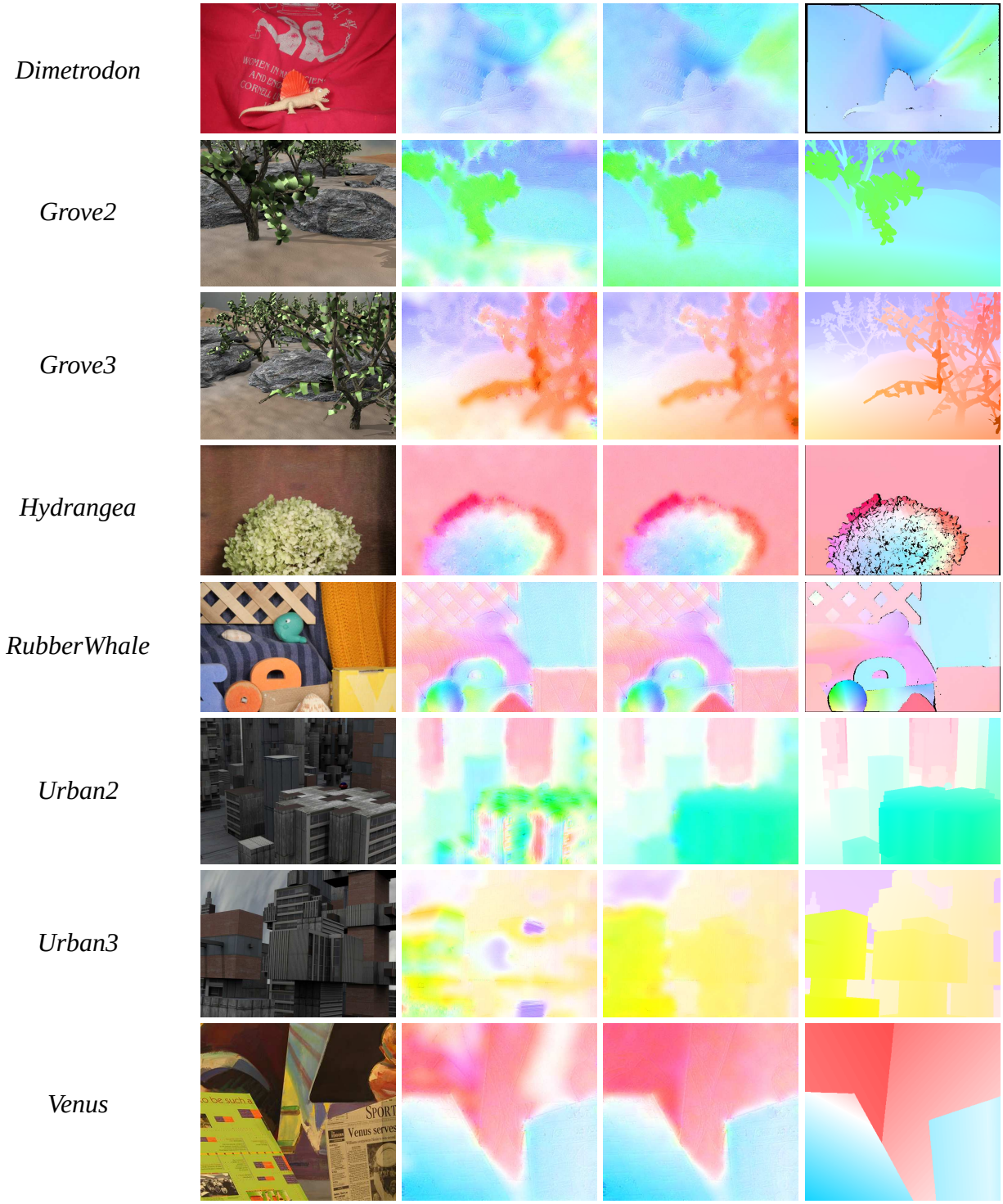


Figure 4: Inferred flow on the Middlebury dataset [1]. The first column shows the initial frame, the second column shows the inferred flow without any initialization, the third column shows the inferred flow with the optical flow of [5] as an initialization, and the fourth column is the ground truth flow. These results are obtained assuming the video is composed of a **single** layer, which would never be used in actual tracking.

[5] C. Liu, W. T. Freeman, E. H. Adelson, and Y. Weiss. Human-assisted motion annotation. In *CVPR*, 2008.

[6] W. K. Pratt. *Digital Image Processing*. John Wiley & Sons, 2007.

[7] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.