Fast and Differentiable Message Passing for Stereo Vision

Zhiwei Xu[1,2,3], Thalaiyasingam Ajanthan[1,2], and Richard Hartley[1,2]

[1]Australian National University [2]Australian Centre for Robotic Vision [3]Data61, CSIRO

Abstract

Despite the availability of many Markov Random Field (MRF) optimization algorithms, their widespread usage is currently limited due to imperfect MRF modelling arising from hand-crafted model parameters. In addition to differentiability, the two main aspects that enable learning these model parameters are the forward and backward propagation time of the MRF optimization algorithm and its parallelization capabilities. In this work, we introduce two fast and differentiable message passing algorithms, namely, Iterative Semi-Global Matching Revised (ISGMR) and Parallel Tree-Reweighted Message Passing (TRWP) which are greatly sped up on GPU by exploiting massive parallelism. Specifically, ISGMR is an iterative and revised version of the standard SGM for general second-order MRFs with improved optimization effectiveness, whereas TRWP is a highly parallelizable version of Sequential TRW (TRWS) for faster optimization. Our experiments on standard stereo benchmarks demonstrate that ISGMR achieves much lower energies than SGM and TRWP is two orders of magnitude faster than TRWS without losing effectiveness in optimization. Furthermore, our CUDA implementations are **at least 7 and 650 times faster** *than PyTorch GPU implementations in the forward and backward propagation, respectively, enabling efficient end-to-end learning with message passing.*

1. Introduction

Optimization of Markov Random Field (MRF) is a well-studied problem for decades with a significant impact on many computer vision applications including stereo vision. Despite the availability of many MRF optimization algorithms, their widespread usage is currently limited due to imperfect MRF modelling arising from hand-crafted model parameters. To this end, enabling efficient end-to-end learning through MRF optimization is essential for improving its modelling and in turn the effectiveness for underlying tasks.

Even though end-to-end learning with MRF has been employed successfully in certain cases, the considered algorithms are suboptimal in terms of the optimization capability [25, 33]. Specifically, the choice of an MRF algorithm for learning is mainly driven by the forward and backward propagation time of the algorithm and its parallelization capabilities rather than the quality of the optimization.

In this work, we consider message passing algorithms due to their generality and differentiability and provide efficient CUDA implementations of their forward and backward propagation by exploiting massive parallelism. In particular, we revise the popular Semi-Global Matching (SGM) method [14] and derive an iterative version noting its relation to traditional message passing algorithms [9]. In addition, we introduce a highly parallelizable version of the state-of-the-art Sequential Tree-Reweighted Message Passing (TRWS) algorithm [16], which is more efficient than TRWS and has similar minimum energies. For both of these methods, we derive efficient backpropagation by unrolling their message updates and cost aggregation and discuss massively parallel CUDA implementations which enable their feasibility in end-to-end learning.

Our experiments on standard stereo benchmarks demonstrate that our Iterative and Revised SGM method (ISGMR) obtains much lower energies compared to standard SGM and our Parallel TRW method (TRWP) is two orders of magnitude faster than TRWS while obtaining virtually the same minimum energies. Furthermore, we empirically evaluate various implementations of the forward and backward propagation of these algorithms and demonstrate that our CUDA implementation is the fastest yielding *at least 650 times speedup* in backpropagation compared to PyTorch GPU version. Our code will be released upon publication. The contributions of this paper can be summarised as:

1. We introduce two message passing algorithms, ISGMR and TRWP, where ISGMR has higher optimization effectiveness than SGM in both single and multiple iterations and TRWP is much faster than TRWS.
2. These message passing algorithms are massively parallelized on GPU and can support any pairwise potentials.
3. The differentiability of these algorithms for end-to-end learning is presented with gradient derivations. Our CUDA implementation of the backpropagation is at least 650 times faster than PyTorch GPU version.

2. Related Work

In MRF optimization for stereo, estimating disparities can be regarded as minimizing a particular energy function with given model parameters. Even if the minimum energy is obtained, highly accurate disparities cannot be guaranteed since model parameters of these MRFs are usually handcrafted and imperfect. To tackle this problem, learning based methods were proposed, however, most of these methods greatly rely on finetuning the network architecture or adding learnable parameters to increase its fitting ability with ground truth. This may not be effective and usually requires high GPU memory.

Nevertheless, considering the highly effective MRF optimization algorithms for stereo vision, the field of exploiting their optimization capability with end-to-end learning to alleviate each other's drawbacks is rarely explored. Few works present this capability for certain cases such as CRFasRNN in semantic segmentation [33] and SGMNet in stereo vision [25] with less effective MRF algorithms. To this end, it is important to explore the use of highly effective MRF algorithms for end-to-end learning.

MRF Optimization Algorithms. Determining an effective MRF optimization algorithm for learning needs a thorough study of the possibility of their differentiability and time efficiency. In the two main categories of MRF optimization algorithms, namely, move-making algorithms (known as graph cuts) [1, 2, 5, 6, 11, 29] and message passing algorithms [14, 15, 16, 20, 21, 31], the state-of-the-art methods are α-expansion [5] and Sequential Tree-Reweighted Message Passing (TRWS) [16], respectively. The move-making algorithms, however, cannot easily be used for end-to-end learning as they are not differentiable and usually limited to certain types of energy functions.

In contrast, message passing algorithms has a better adaption to any energy functions and can be differentiable and fast if well designed. Some works in probabilistic graphical models indeed demonstrate the learning ability of the family of TRW algorithms in sum-product and max-product forms [15, 31]. Although Semi-Global Matching (SGM) method [14] is not in the benchmark, it was proved to have a high running efficiency due to the fast one-dimensional dynamic programming that is independent in each scanline and scanning direction.

End-to-End Learning. Sum-product TRW [8, 27, 28] and mean-field [33] were used for end-to-end learning for semantic segmentation, which presented their highly effective learning ability. Meanwhile, for stereo vision, several SGM based methods have been proposed for end-to-end learning in CNNs, such as SGMNet [25] and GANet [32]. Most recently, GANet [32] utilized a modified SGM with a combination of MAP and marginial estimation forms. These works further indicate the high efficiency of selected MRF optimization algorithms for end-to-end learning.

Our work is based on the aforementioned two differentiable message passing algorithms, SGM and TRWS, but we imporve their optimization effectiveness and time efficiency for fast end-to-end learning. In particular, we revise the standard SGM and make it iterative for improvement of its optimization capability, which we denote as ISGMR. Meanwhile, TRWP is a massively parallelizable version of TRWS, which greatly increases the running speed without losing the optimization effectiveness.

Below, we provide forward and backward propagation of ISGMR and TRWP, fast implementation and experiments demonstrating improved optimization effectiveness and time efficiency. Additional details and experiments are given in the Appendix.

3. Message Passing Algorithms

We first briefly review the typical form of an MRF energy function used for stereo and discuss two highly parallelizable message passing approaches. Such a parallelization capability is essential for fast implementation on GPU and enables relatively straightforward integration to existing deep learning models.

3.1. MRF Energy Function

Let X_i be a random variable taking label $x_i \in \mathcal{L}$. A second-order MRF energy function defined over a set of such random variables, parametrized by $\mathbf{\Theta} = \{\theta_i, \theta_{i,j}\}$ can be represented by

$$E(\mathbf{x}|\mathbf{\Theta}) = \sum_{i \in \mathcal{V}} \theta_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{i,j}(x_i, x_j) \,, \qquad (1)$$

where θ_i and $\theta_{i,j}$ denote the unary potentials and pairwise potentials respectively. Here, $\mathcal{V}$ is the set of vertices (*e.g.*, corresponding to pixels or superpixels in the image), and $\mathcal{E}$ is the set of edges in the MRF (*e.g.*, encoding the 4-connected or 8-connected grid over the image pixels).

3.2. Iterative Semi-Global Matching Revised

Now we briefly introduce standard SGM which supports only a single iteration. Then, by noting its connection to message passing, we modify its message update equation of SGM and introduce an iterative version. Figure 1 shows an example of SGM with four directions on a grid MRF.

3.2.1 Revised Semi-Global Matching

We cast the popular SGM algorithm [14] as an optimization method for a particular MRF and discuss its relation to

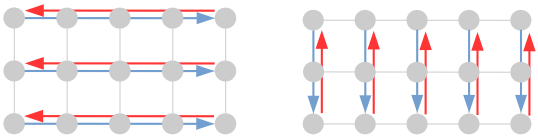


Figure 1: *SGM on a grid MRF with four directions, that is, left-right, right-left, up-down, down-up.*

message passing as noted in [9]. In SGM, pairwise potentials are simplified for all edges $(i,j) \in \mathcal{E}$ as

$$\theta_{i,j}(\lambda,\mu) = \theta_{i,j}(|\lambda-\mu|) = \begin{cases} 0 & \text{if } \lambda = \mu\,, \\ P_1 & \text{if } |\lambda-\mu| = 1\,, \\ P_2 & \text{if } |\lambda-\mu| \geq 2\,, \end{cases} \quad (2)$$

where $0 < P_1 \leq P_2$. The idea of SGM relies on cost aggregation in multiple directions (each direction has multiple one-dimensional scanlines) using dynamic programming. The main observation made by [9] is that, in SGM the unary potentials are over-counted $|\mathcal{R}|-1$ times (where $\mathcal{R}$ denotes the set of directions) compared to the standard message passing and this over-counting corrected SGM is shown to perform slightly better in [10]. Noting this analogy, we use symbol $m^r_{i:\lambda}$ to denote the message at node i, label λ in direction r (*i.e.*, message from the previous node of i in direction r, denoted as $i-r$, to i at label λ). Now, the revised SGM update can be written as

$$m^r_{i:\lambda} = \min_{\mu\in\mathcal{L}} \left(\theta_{i-r:\mu} + m^r_{i-r:\mu} + \theta_{i-r,i}(\mu,\lambda)\right)\,. \quad (3)$$

Due to the permanently increasing values of messages along the scanning direction [14], message $m^r_{i:\lambda}$ can be reparametrized as

$$m^r_{i:\lambda} = m^r_{i:\lambda} - \min_{\mu\in\mathcal{L}} m^r_{i:\mu}\,, \quad (4)$$

which does not alter the minimum energy. Then, the final cost at a particular node i is the sum of messages over all the directions plus the unary term, *i.e.*,

$$c_{i:\lambda} = \theta_{i:\lambda} + \sum_{r\in\mathcal{R}} m^r_{i:\lambda}\,, \quad \forall i \in \mathcal{V}, \forall\,\lambda \in \mathcal{L}\,. \quad (5)$$

The final labelling is then obtained by

$$x^*_i = \operatorname*{argmin}_{\lambda\in\mathcal{L}} c_{i:\lambda}\,, \quad \forall\, i \in \mathcal{V}\,. \quad (6)$$

Note that, the message update in the revised SGM Eq. (3) is performed in parallel for all scanlines for all directions. This massive parallelization makes it suitable for real-time applications [13] and end-to-end learning for stereo [25].

3.2.2 Iteration of the Revised Semi-Global Matching

In spite of the revision for the over-counting problem, the 3-penalty pairwise potential in Eq. (2) is insufficient to obtain dominant penalties under a large range of disparities in different camera settings. To this end, we consider more general pairwise potentials $\theta_{i,j}(\lambda,\mu)$ and introduce an iterative version of the revised SGM. The message update for the iterative version can be written as

$$\hat{m}^r_{i:\lambda} = \min_{\mu\in\mathcal{L}} \Big(\theta_{i-r:\mu} + \hat{m}^r_{i-r,\mu} + \sum_{d\in\mathcal{R}\setminus\{r,r^-\}} m^d_{i-r:\mu} + \theta_{i-r,i}(\mu,\lambda)\Big)\,, \quad (7)$$

where r^- denotes the opposite direction of r and $\hat{m}^r_{i-r:\mu}$ denotes the updated message in an iteration while $m^r_{i-r:\mu}$ is updated in the previous iteration. The exclusion of the messages from direction r^- is important to make sure that the update is analogous to the standard message passing and the same energy function is minimized at each iteration. Thus a simple combination of several standard SGM does not satisfy this rule and performed worse in our experiments than that reported in Table 1 and tables in Appendix E.1. Usually, $\hat{\mathbf{m}}^r$ for all $r \in \mathcal{R}$ are initialized to 0, therefore for a single iteration, this exclusion of r^- from $\mathcal{R}$ is redundant but not for multiple iterations. Even in this case, the messages can be reparametrized by Eq. (4).

In each iteration, after $\hat{m}^r_{i:\lambda}$ is updated for all $r \in \mathcal{R}$, it will be updated for the next iteration by

$$m^r_{i:\lambda} = \hat{m}^r_{i:\lambda}, \quad \forall i \in \mathcal{V}, \forall\lambda \in \mathcal{L}, \forall r \in \mathcal{R}. \quad (8)$$

After multiple iterations, the final cost for node $i \in \mathcal{V}$ is calculated by Eq. (5), and the final labelling is calculated in the same manner as Eq. (6). We denote this iterative and revised SGM as ISGMR, summarized in Algorithm 1.

3.3. Parallel Tree-Reweighted Message Passing

On the other hand, TRWS [16] is a state-of-the-art message passing algorithm that optimizes the Linear Programming (LP) relaxation of a general pairwise MRF energy given in Eq. (1). The main idea of the family of TRW algorithms [30] is to decompose the underlying graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ of the MRF with parameters $\boldsymbol{\Theta}$ into a combination of trees where the sum of parameters of all the trees is equal to that of the MRF, *i.e.*, $\sum_{T\in\mathcal{T}} \boldsymbol{\Theta}_T = \boldsymbol{\Theta}$. Then, at each iteration message passing is performed in each of these trees independently, followed by an averaging operation. Even though any combinations of trees would theoretically result in the same final labelling, the best performance is achieved by choosing a monotonic chain decomposition and a sequential message passing update rule, known as TRWS. We refer the interested reader to [16] for more details.

Since we intend to enable fast message passing by exploiting parallelism, our idea is to choose a tree decomposition that can be massively parallelized, denoted as TRWP. In the literature, edge-based or tree-based parallel TRW algorithms have been considered, namely, TRWE and TRWT

Algorithm 1: *Forward Propagation of ISGMR*

Input: Energy parameters $\mathbf{\Theta} = \{\theta_i, \theta_{i,j}(\cdot,\cdot)\}$, set of nodes $\mathcal{V}$, edges $\mathcal{E}$, directions $\mathcal{R}$, maximum iteration K.

Output: Labelling $\mathbf{x}^*$ for optimization, costs $\{c_{i:\lambda}\}$ for learning, indices $\{p^r_{k,i:\lambda}\}$ and $\{q^r_{k,i}\}$ for backpropagation.

$\hat{\mathbf{m}} \leftarrow 0$ and $\mathbf{m} \leftarrow 0$ ▷initialize all messages
for iteration $k \in \{1, \dots, K\}$ **do**
  forall directions $r \in \mathcal{R}$ **do** ▷**parallel**
    forall scanlines t in direction r **do** ▷**parallel**
      for node i in scanline t **do** ▷**sequential**
        for label $\lambda \in \mathcal{L}$ **do**
          $\Delta_{\lambda,\mu} \leftarrow \theta_{i-r:\mu} + \hat{m}^r_{i-r:\mu} + \sum_{d \in \mathcal{R}\setminus\{r,r^-\}} m^d_{i-r:\mu} + \theta_{i-r,i}(\mu,\lambda)$
          $p^r_{k,i:\lambda} \leftarrow \mu^* \leftarrow \operatorname{argmin}_{\mu\in\mathcal{L}} \Delta_{\lambda,\mu}$ ▷store index
          $\hat{m}^r_{i:\lambda} \leftarrow \Delta_{\lambda,\mu^*}$ ▷message update (7)
        $q^r_{k,i} \leftarrow \lambda^* \leftarrow \operatorname{argmin}_{\lambda\in\mathcal{L}} \hat{m}^r_{i:\lambda}$ ▷store index
        $\hat{m}^r_{i:\lambda} \leftarrow \hat{m}^r_{i:\lambda} - \hat{m}^r_{i:\lambda^*}$ ▷reparametrization (4)
  $\mathbf{m} \leftarrow \hat{\mathbf{m}}$ ▷update messages after iteration
$c_{i:\lambda} \leftarrow \theta_{i:\lambda} + \sum_{r\in\mathcal{R}} m^r_{i:\lambda}\ \forall\, i \in \mathcal{V}, \lambda \in \mathcal{L}$ ▷Eq. (5)
$x^*_i \leftarrow \operatorname{argmin}_{\lambda\in\mathcal{L}} c_{i:\lambda}, \forall i \in \mathcal{V}$ ▷Eq. (6)

in the probability space (specifically sum-product message passing) rather than for minimizing the energy [30]. Optimizing in the probability domain involves exponential calculations which are prone to numerical instability, and the sum-product version requires $\mathcal{O}(|\mathcal{R}||\mathcal{L}|)$ times more memory compared to the min-sum message passing in backpropagation. More details are in Appendix D.

Correspondingly, our TRWP directly minimizes the energy in the min-sum message passing fashion similar to TRWS, and thus, its update can be written as

$$m^r_{i:\lambda} = \min_{\mu\in\mathcal{L}} \big(\rho_{i-r,i}(\theta_{i-r:\mu} + \sum_{d\in\mathcal{R}} m^d_{i-r:\mu}) - m^{r^-}_{i-r:\mu} + \theta_{i-r,i}(\mu,\lambda)\big)\ . \tag{9}$$

Here the coefficient $\rho_{i-r,i} = \gamma_{i-r,i}/\gamma_{i-r}$, where $\gamma_{i-r,i}$ and γ_{i-r} are the number of trees containing the edge $(i-r, i)$ and the node $i-r$ respectively in the considered tree decomposition. For loopy belief propagation, since there is no tree decomposition, $\rho_{i-r,i} = 1$. For a 4-connected graph decomposed into all horizontal and vertical one-dimensional trees, we have $\rho_{i-r,i} = 0.5$ for all edges.

Note that, similar to ISGMR, we use the scanline to denote a tree. The above update can be performed in parallel for all scanlines in a single direction, however, the message updates over a scanline are sequential. The same reparametrization Eq. (4) is applied. The algorithm is summarized in Algorithm 2.

While TRWP cannot guarantee the non-decreasing

Algorithm 2: *Forward Propagation of TRWP*

Input: Energy parameters $\mathbf{\Theta} = \{\theta_i, \theta_{i,j}(\cdot,\cdot)\}$, set of nodes $\mathcal{V}$, edges $\mathcal{E}$, directions $\mathcal{R}$, tree decomposition coefficients $\{\rho_{i,j}\}$, maximum iteration K.

Output: Labelling $\mathbf{x}^*$ for optimization, costs $\{c_{i:\lambda}\}$ for learning, indices $\{p^r_{k,i:\lambda}\}$ and $\{q^r_{k,i}\}$ for backpropagation.

$\mathbf{m} \leftarrow 0$ ▷initialize all messages
for iteration $k \in \{1, \dots, K\}$ **do**
  for direction $r \in \mathcal{R}$ **do** ▷**sequential**
    forall scanlines t in direction r **do** ▷**parallel**
      for node i in scanline t **do** ▷**sequential**
        for label $\lambda \in \mathcal{L}$ **do**
          $\Delta_{\lambda,\mu} \leftarrow \rho_{i-r,i}\big(\theta_{i-r:\mu} + \sum_{d\in\mathcal{R}} m^d_{i-r:\mu}\big) - m^{r^-}_{i-r:\mu} + \theta_{i-r,i}(\mu,\lambda)$
          $p^r_{k,i:\lambda} \leftarrow \mu^* \leftarrow \operatorname{argmin}_{\mu\in\mathcal{L}} \Delta_{\lambda,\mu}$ ▷store index
          $m^r_{i:\lambda} \leftarrow \Delta_{\lambda,\mu^*}$ ▷message update (9)
        $q^r_{k,i} \leftarrow \lambda^* \leftarrow \operatorname{argmin}_{\lambda\in\mathcal{L}} m^r_{i:\lambda}$ ▷store index
        $m^r_{i:\lambda} \leftarrow m^r_{i:\lambda} - m^r_{i:\lambda^*}$ ▷reparametrization (4)
$c_{i:\lambda} \leftarrow \theta_{i:\lambda} + \sum_{r\in\mathcal{R}} m^r_{i:\lambda}, \forall i \in \mathcal{V}, \lambda \in \mathcal{L}$ ▷Eq. (5)
$x^*_i \leftarrow \operatorname{argmin}_{\lambda\in\mathcal{L}} c_{i:\lambda}, \forall i \in \mathcal{V}$ ▷Eq. (6)

monotonicity of the lower bound of energy, it dramatically improves the forward propagation speed and yields virtually similar minimum energies as that of TRWS.

3.4. Relation between ISGMR and TRWP

Both ISGMR and TRWP utilize messages from neighbouring nodes in recursive and iterative message updates via dynamic programming. Comparison of Eq. (7) and Eq. (9) indicates the introduction of the coefficients $\{\rho_{i-r,i}\}$. This is due to the tree decomposition, which is analogous to the difference between loopy belief propagation and TRW algorithms. The most important difference, however, is the way of message updates in iterations. Specifically, within an iteration, ISGMR can be parallelized over directions since the most updated messages $\hat{\mathbf{m}}^r$ are used only for the current scanning direction r and previous messages are used for the other directions (refer Eq. (7)). In contrast, aggregated messages in TRWP are up-to-date in directions. This *direction-by-direction* update largely contributes to the improved effectiveness of TRWP over ISGMR.

3.5. Fast Implementation by Tree Parallelization

Independent trees make the parallelization possible. For ISGMR, a further parallelization over directions could be achieved. We implemented on CPU and GPU, where for the C++ multi-thread versions (CPU), 8 threads on Open Multi-Processing (OpenMP) [7] are utilized while for the CUDA versions (GPU), 512 threads per block are utilized.

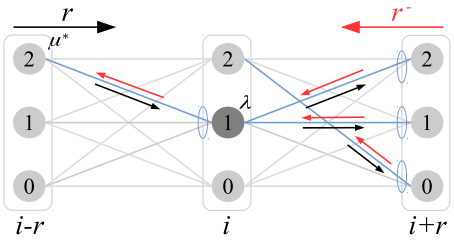


Figure 2: *Forward and backward propagation. Black arrow indicates forward propagation in direction r, red arrow indicates backpropagation in direction r^-, blue ellipse is min operation, blue line indicates an edge having the minimum message, for instance, from $i-r$ at label μ^* to i at label λ.*

Algorithm 3: *Backpropagation of ISGMR*

Input: Partial energy parameters $\{\theta_{i,j}\}$, gradients of final costs $\nabla\mathbf{c} = \{\nabla c_{i:\lambda}\}$, set of nodes $\mathcal{V}$, edges $\mathcal{E}$, directions $\mathcal{R}$, indices $\{p^r_{k,i:\lambda}\}$, $\{q^r_{k,i}\}$, maximum iteration K.

Output: Gradients $\{\nabla\theta_i, \nabla\theta_{i,j}(\cdot,\cdot)\}$.

1 $\nabla\mathbf{m^r} \leftarrow \nabla\mathbf{\Theta_i} \leftarrow \nabla\mathbf{c}$, $\nabla\mathbf{\Theta_{i,j}} \leftarrow 0$ ▷back Eq. (5)
2 $\nabla\hat{\mathbf{m}}^\mathbf{r} \leftarrow \nabla\mathbf{m^r}$ ▷back message updates
3 **for** iteration $k \in \{K, ..., 1\}$ **do**
4 $\nabla\mathbf{m^r} \leftarrow 0$ ▷zero-out
5 **forall** directions $r \in \mathcal{R}$ **do** ▷**parallel**
6 **forall** scanlines t in direction r **do** ▷**parallel**
7 **for** node i in scanline t **do** ▷**sequential**
8 $\lambda^* \leftarrow q^r_{k,i} \in \mathcal{L}$ ▷extract index
9 $\nabla\hat{m}^r_{i:\lambda^*} \mathrel{-}= \sum_{\lambda\in\mathcal{L}} \nabla\hat{m}^r_{i:\lambda}$ ▷back Eq. (4)
10 **for** label $\lambda \in \mathcal{L}$ **do**
11 $\mu^* \leftarrow p^r_{k,i:\lambda} \in \mathcal{L}$ ▷extract index
12 $\nabla\theta_{i-r:\mu^*} \mathrel{+}= \nabla\hat{m}^r_{i:\lambda}$ ▷back Eq. (7)
13 $\nabla\hat{m}^r_{i-r:\mu^*} \mathrel{+}= \nabla\hat{m}^r_{i:\lambda}$
14 $\nabla m^d_{i-r:\mu^*} \mathrel{+}= \nabla\hat{m}^r_{i:\lambda}, \forall d \in \mathcal{R}\setminus\{r, r^-\}$
15 $\nabla\theta_{i-r,i}(\mu^*, \lambda) \mathrel{+}= \nabla\hat{m}^r_{i:\lambda}$
16 $\nabla\hat{\mathbf{m}}^r \leftarrow 0$ ▷zero-out
17 $\nabla\mathbf{m^r} \mathrel{+}= \nabla\hat{\mathbf{m}}^\mathbf{r}$ ▷gather history gradients
18 $\nabla\hat{\mathbf{m}}^\mathbf{r} \leftarrow \nabla\mathbf{m^r}$ ▷back message updates after iteration

Each tree is headed by its first node by interpolation. For a single graph tree, message updates are sequential from the first node to the last. The node indexing details for efficient parallelism are provided in Appendix B.

So far, we have discussed an improved version of SGM and a parallel version of TRWS, and our ultimate goal is to use these message passing algorithms in an end-to-end learning framework. In the next section, we derive efficient backpropagation through each of these algorithms.

4. Differentiability of Message Passing

Effective and differentiable MRF optimization algorithms can greatly improve the performance of end-to-end learning. Typical works such as CRFasRNN for semantic segmentation [33] and SGMNet for stereo vision [25], use mean-field and SGM, respectively which are inferior in the optimization capability compared to ISGMR and TRWP.

Therefore, in order to use ISGMR and TRWP for end-to-end learning, differentiability of these two algorithms is required and essential. Below, we describe the gradient update equations for the learnable MRF model parameters and detailed derivations are given in Appendix C. The backpropagation pseudocodes are provided in Algorithms 3-4.

Since ISGMR and TRWP utilize min-sum message passing, no exponent and logarithm are required. Only indices in message minimization and reparametrization are stored in two unsigned 8-bit integer tensors, denoted as $\{p^r_{k,i:\lambda}\}$ and $\{q^r_{k,i}\}$ with indices of direction r, iteration k, node i, and label λ. This makes the backpropagation time less than 50% of the forward propagation time. Figure 2 illustrates that the gradient updates in backpropagation are performed along the edges that have the minimum messages in the forward direction. Below, we denote the gradient of a variable $*$ with regard to the loss L in learning as $\nabla* = dL/d*$.

For ISGMR at kth iteration, gradients of model parameters in Eq. (7), $\nabla\theta_{i:\lambda}$ and $\nabla\theta_{i-r,i}(\mu,\lambda)$, are

$$\nabla\theta_{i:\lambda} = \nabla c_{i:\lambda} + \sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}}\sum_{\mu\in\mathcal{L}} \Big(\nabla\hat{m}^r_{i+2r:\mu}\big|_{v=p^r_{k,i+2r:\mu}} + \sum_{d\in\mathcal{R}\setminus\{r,r^-\}} \nabla m^d_{i+r+d:\mu}\big|_{v=p^d_{k,i+r+d:\mu}} \Big)\Big|_{\lambda=p^r_{k,i+r:v}},$$

$$\nabla\theta_{i-r,i}(\mu,\lambda) = \nabla\hat{m}^r_{i:\lambda}|_{\mu=p^r_{k,i:\lambda}} \quad . \tag{10}$$

Importantly, within an iteration in ISGMR, gradients of messages $\nabla\mathbf{m^r}$ are updated but do not affect $\nabla\hat{\mathbf{m}}^\mathbf{r}$ until the backpropagation along all directions is executed, see line 18 in Algorithm 3. This is because within an iteration of the forward propagation independently updated messages $\hat{\mathbf{m}}^r$ in r will not affect messages $\mathbf{m}^d, \forall d \in \mathcal{R}\setminus\{r, r^-\}$, until the next iteration, see line 12 in Algorithm 1.

Different from ISGMR, for TRWP, gradients of messages from a direction will affect messages from other directions since, within an iteration in the forward propagation, message updates are *direction-by-direction*. For TRWP at kth iteration, $\nabla\theta_{i:\lambda}$ related to Eq. (9) is

$$\nabla\theta_{i:\lambda} = \nabla c_{i:\lambda} + \sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}}\sum_{\mu\in\mathcal{L}} \Big(-\nabla m^{r^-}_{i:\mu}\big|_{v=p^{r^-}_{k,i:\mu}} + \sum_{d\in\mathcal{R}} \rho_{i+r,i+r+d}\nabla m^d_{i+r+d:\mu}\big|_{v=p^d_{k,i+r+d:\mu}} \Big)\Big|_{\lambda=p^r_{k,i+r:v}}, \tag{11}$$

Algorithm 4: *Backpropagation of TRWP*

Input: Partial energy parameters $\{\theta_{i,j}\}$, gradients of final costs $\nabla\mathbf{c} = \{\nabla c_{i:\lambda}\}$, tree decomposition coefficients $\{\rho_{i,j}\}$, set of nodes $\mathcal{V}$, edges $\mathcal{E}$, directions $\mathcal{R}$, indices $\{p^r_{k,i:\lambda}\}$, $\{q^r_{k,i}\}$, maximum iteration K.

Output: Gradients $\{\nabla\theta_i, \nabla\theta_{i,j}(\cdot,\cdot)\}$.

1 $\nabla\mathbf{m^r} \leftarrow \nabla\mathbf{\Theta_i} \leftarrow d\mathbf{c}$, $d\mathbf{\Theta_{i,j}} \leftarrow 0$ ▷back Eq. (5)
2 **for** iteration $k \in \{K, ..., 1\}$ **do**
3 **for** direction $r \in \mathcal{R}$ **do** ▷**sequential**
4 **forall** scanlines t in direction r **do** ▷**parallel**
5 **for** node i in scanline t **do** ▷**sequential**
6 $\lambda^* \leftarrow q^r_{k,i} \in \mathcal{L}$ ▷extract index
7 $\nabla m^r_{i:\lambda^*} -= \sum_{\lambda\in\mathcal{L}} \nabla m^r_{i:\lambda}$ ▷back Eq. (4)
8 **for** label $\lambda \in \mathcal{L}$ **do**
9 $\mu^* \leftarrow p^r_{k,i:\lambda} \in \mathcal{L}$ ▷extract index
10 $\nabla\theta_{i-r:\mu^*}$ $+= \rho_{i-r,i}\nabla m^r_{i:\lambda}$ ▷back Eq. (9)
11 $\nabla m^d_{i-r:\mu^*}$ $+= \rho_{i-r,i}\nabla m^r_{i:\lambda}, \forall d \in \mathcal{R}$
12 $\nabla m^{\bar{r}}_{i-r:\mu^*}$ $-= \nabla m^r_{i:\lambda}$
13 $\nabla\theta_{i-r,i}(\mu^*, \lambda)$ $+= \nabla m^r_{i:\lambda}$
14 $\nabla\mathbf{m}^r \leftarrow 0$ ▷zero-out

where coefficient $\rho_{i+r,i+r+d}$ is for the edge connecting node $i+r$ and its next one in direction d which is denoted as node $i+r+d$, and the calculation of $\nabla\theta_{i-r,i}(\lambda,\mu)$ is in the same manner as Eq. (10) by replacing $\hat{m}$ with m.

The backpropagation of TRWP can be derived similarly as ISGMR. We must know that gradients of the unary potentials and the pairwise potentials are accumulated along the opposite direction of the forward scanning direction. Therefore an updated message is, in fact, a new variable and its gradient should not be accumulated by its previous value but set to 0. This is extremely important, especially in ISGMR that only one of the $|\mathcal{R}|$ directions has updated messages in the scanning direction while all the others are unchanged since the previous iteration. It requires the message gradients to be accumulated and assigned in every iteration, lines 17-18 in Algorithm 3, and be zero-out, lines 4 and 16 in Algorithm 3 and line 14 in Algorithm 4. Gradient derivations of ISGMR and characteristics are provided in Appendix C.

5. Experiments

The experiments include effectiveness and efficiency studies of the message passing algorithms. We implemented SGM, ISGMR, TRWP on CPU and GPU from scratch. For a fair comparison of TRWS, we adopted the benchmark code from [26] and called the general function for any pairwise potentials, instead of the specialized version by distance transform [4] for (truncated) linear and (truncated) quadratic functions. Our implementations are in C++ single and multiple threads, PyTorch, and CUDA. PyTorch versions are used for time comparison and gradient checking.

CPU experiments are executed on a 3.60 GHz i7-7700 Intel(R) Core(TM) with 15.6 GB memory and 8 threads (4 for each core), and GPU experiments are on a single GeForce GTX 1080Ti with 11 GB memory.

5.1. Datasets

We only consider realistic datasets, Middlebury [22, 23], KITTI2015 [19, 18], and ETH3D two-view [24], instead of synthetic ones such as SceneFlow [17]. Here, partial image pairs from the datasets are used. Specifically, images Tsukuba, Teddy, Venus, Map, and Cones are from Middlebury, 000041_10 and 000119_10 are from KITTI2015, and delivery_area_1l and facade_1 are from ETH3D.

5.2. Quantitative Study of Optimization

The capability to minimize an energy function determines the significance of selected algorithms. Thus its effectiveness study is undoubtedly important for both optimization and learning. The quantitative study includes comparisons among SGM, ISGMR, TRWP, and TRWS.

Additionally, we also demonstrate the optimization capability of a simple combination of 50 standard SGMs for comparison and the description in Section 3.2.2. Unary potentials are reparametrized by aggregating messages from the previous iteration in the same manner by Eq. (5).

Model parameters. Model parameters include unary potentials and pairwise potentials. In practice, the pairwise potentials generally consist of a pairwise function and edge weights and can be written as $\theta_{i,j}(\lambda,\mu) = \theta_{i,j}V(\lambda,\mu)$. In our case, the unary potentials are calculated by [3] for its fairly qualified effectiveness. For the pairwise function $V(\cdot,\cdot)$, one can adopt (truncated) linear, (truncated) quadratic, Cauchy, Huber, *etc*. [12]. For the edge weights $\theta_{i,j}$, some existing methods provide a higher penalty on edge gradients that are under a given threshold. We set it as a constant for a comparison with the standard SGM in Table 1. In the experiments, we adopted edge weights as a constant value provided in [26] as well as the same pairwise functions for Tsukuba, Teddy, and Venus, and [2] for Cones and Map; for the others, the pairwise function is symmetric linear and edges weights are constant 10. More evaluations with constant and threshold-based edge weights are given in Appendix E.

Number of direction(s) matters. In Figure 3, ISGMR-8 and TRWP-4 outperform others in ISGMR-related and TRWP-related methods in most cases. An increase of direction number is not directly proportional to an improvement of effectiveness since messages from different directions may not equally contribute to the message updates.

In our case, 4 directions are sufficient for TRWP, but for ISGMR energies with 8 directions are lower than those with 4 directions. This is because messages from 4 directions

in ISGMR are insufficient to gather local information due to its independent message updates in each direction. In contrast, messages from 4 directions in TRWP are highly updated in each direction and affected by those from other directions. We note that in Eq. (5), messages from all directions are summed equally. This makes the disparities by TRWP over-smooth within the connection area, for instance, in Figure 4(f), the camera is oversmooth. From our experiments, ISGMR with 8 directions and TRWP with 4 directions work the best.

ISGMR vs SGM. In [10], the author demonstrated the ratio of energy decrease of the over-count corrected SGM over the standard SGM. The result showed the improved optimization by subtracting unary potentials $|\mathcal{R}| - 1$ times. For experimental completion, we show both the decreased energies and disparity maps with a single iteration of ISGMR. From Table 1, SGM-related energies are much higher than ISGMR energies because the repeated usage of unary potentials decreases the importance of pairwise potentials. And ISGMR has much lower energies at the 50th iteration than the 1st iteration, which highlights the importance of the iterative optimization. Besides, energies by a combination of 50 SGMs are much higher than ISGMR at 50th iteration.

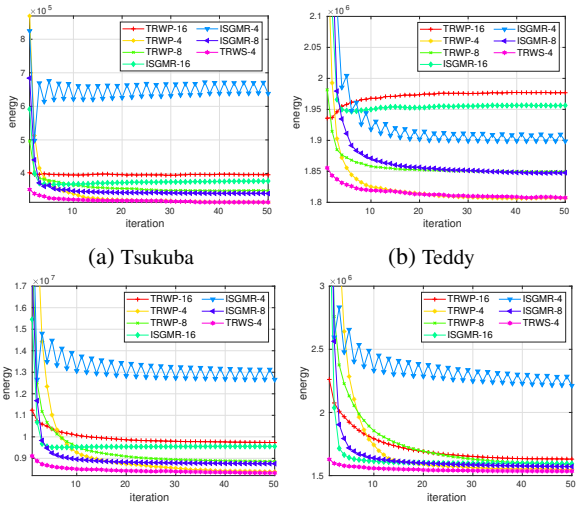
TRWP vs TRWS. TRWP and TRWS share the same manner of updating messages and could have similar minimum energies. Generally, TRWS has the lowest energy; in some iterations, however, TRWP-4 has lower energies, for instance, the energies at the 50th iteration for Tsukuba and Teddy in Table 1. 50 iterations are sufficient to show the high optimization capability of TRWP.

5.3. Speed Improvement

The experiments consist of forward and backward propagation time comparison. The implementation of forward propagation is by PyTorch CPU/GPU tensors, C++ single/multiple threads, and CUDA; the implementation of backpropagation is by PyTorch GPU tensors and CUDA.

Message passing algorithms rely on sequential message updates on a tree(s) or a chain, which cannot be parallelized. Also, to ensure all updated messages are differentiable in PyTorch, the only way is to create a new tensor to store the updated messages and concatenate them with messages from the other directions, for message updates of next nodes in each scanning direction by Eq. (7) and Eq. (9). This increases the memory as well. Our CUDA versions, however, do not have this problem and achieves a much higher speed with less than 50% of the forward propagation time.

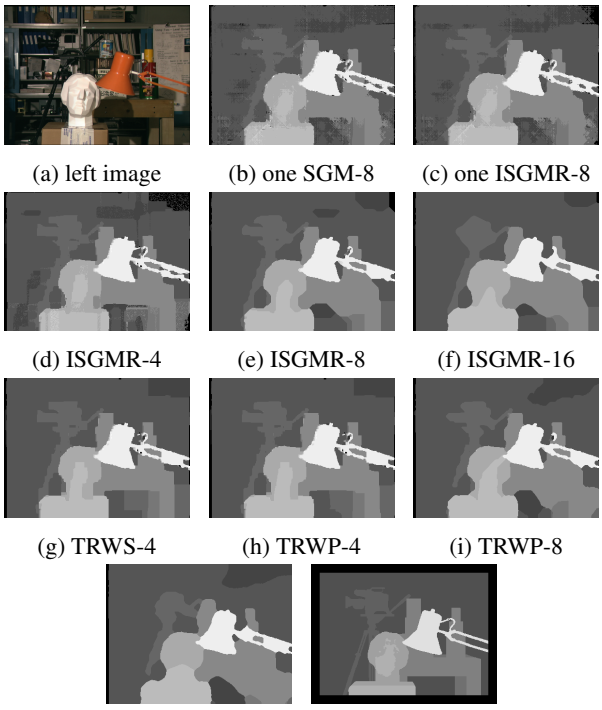
Forward propagation time. In Table 2, the forward propagation by CUDA implementation is the fastest. Our CUDA versions of ISGMR-8 and TRWP-4 are at least 24 and 7 times faster than PyTorch GPU versions at 32 and



(a) Tsukuba (b) Teddy

(c) 000002_11 (KITTI) (d) delivery_area_11 (ETH3D)

Figure 3: *Energy minimization. ISGMR-8 and TRWP-4 have the lowest energies in ISGMR-related and TRWP-related methods respectively. Note that TRWP-4 achieves similar optimization capability to TRWS-4.*



(a) left image (b) one SGM-8 (c) one ISGMR-8

(d) ISGMR-4 (e) ISGMR-8 (f) ISGMR-16

(g) TRWS-4 (h) TRWP-4 (i) TRWP-8

(j) TRWP-16 (k) ground truth

Figure 4: *Disparities of Tsukuba. (b)-(c) are at 1st iteration. (d)-(j) are at 50th iteration. (e) and (h) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*

Method	Tsukuba		Teddy		000002_11 (KITTI)		delivery_area_11 (ETH3D)	
	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter
SGM-4	873777	644840	2825535	**2559016**	24343250	18060026	5851489	**4267990**
SGM-8	776706	**574758**	2868131	2728682	20324684	**16406781**	5396353	4428411
SGM-16	710727	587376	2907051	2846133	18893122	16791762	5092094	4611821
TRWS-4	352178	**314393**	1855625	**1807423**	9109976	**8322635**	1628879	**1534961**
ISGMR-4	824694	637996	2626648	1898641	22259606	12659612	5282024	2212106
ISGMR-8	684185	**340347**	2532071	**1847833**	17489158	**8753990**	4474404	**1571528**
ISGMR-16	591554	377427	2453592	1956343	15455787	9556611	3689863	1594877
TRWP-4	869363	**314037**	2234163	**1806990**	40473776	**8385450**	9899787	**1546795**
TRWP-8	496727	348447	1981582	1849287	18424062	8860552	4443931	1587917
TRWP-16	402033	396036	1935791	1976839	11239113	9736704	2261402	1630973

Table 1: *Energy minimization. ISGMR-8 and TRWP-4 have the lowest energies in ISGMR-related and TRWP-related methods respectively. Note that ISGMR is more effective than SGM in optimization in both single and multiple iterations while TRWP obtains similar energies as TRWS.*

Method	PyTorch CPU		PyTorch GPU		C++ single	
	32	96	32	96	32	96
TRWS-4	-	-	-	-	1.9538	13.2946
ISGMR-4	1.4276	11.7024	0.9589	1.1302	3.2295	25.1851
ISGMR-8	3.1835	24.7831	1.5933	1.9804	8.2503	71.3489
ISGMR-16	7.8937	52.7613	2.3360	4.9575	30.7617	273.6819
TRWP-4	1.4047	11.7428	0.8667	1.0790	1.8383	15.4130
TRWP-8	3.1890	24.2815	1.5719	1.9760	6.3431	57.2451
TRWP-16	7.8572	51.8452	2.8191	5.0752	28.9312	262.2763

Method	C++ multiple		CUDA		*PyTorch GPU/CUDA*	
	32	96	32	96	32	96
TRWS-4	-	-	-	-	-	-
ISGMR-4	0.8786	5.2826	**0.0349**	**0.1488**	**27**	**8**
ISGMR-8	2.1214	15.9042	**0.0661**	**0.2697**	**24**	**7**
ISGMR-16	7.6997	62.7152	**0.1335**	**0.5256**	**17**	**9**
TRWP-4	0.7637	4.4623	**0.0343**	**0.1528**	**25**	**7**
TRWP-8	1.8834	14.2176	**0.0651**	**0.2703**	**24**	**7**
TRWP-16	7.4050	60.4486	**0.1306**	**0.5189**	**22**	**10**

Table 2: *Forward propagation time (seconds) on a 256*512 image with 32 and 96 disparities for one iteration. Time of CUDA is averaged over 1000 times; the rests are over 100 times. CUDA version is 7 – 27 times faster than PyTorch GPU version.*

96 disparities respectively. The increase of time with an increase of disparity in the CUDA versions is due to two loops for neighbour labels, λ and μ, in Eq. (7) and Eq. (9). In PyTorch GPU versions, we utilize tensor-wise tree parallelization to highly speed it up for a fair comparison. Obviously, GPU versions are much faster than CPU versions.

Backpropagation time. In Table 3, the backpropagation time clearly distinguishes the higher efficiency of CUDA versions than PyTorch GPU versions. Statistically, the CUDA versions are at least 650 times faster than PyTorch GPU versions, since only a low memory is used to store indices for the backpropagation. It results in a single loop per tree in the backpropagation. This makes the backpropagation much faster than the forward propagation. In summary, the largely improved speed makes learning using these two message passing algorithms feasible.

6. Limitations

Ideally, ISGMR can achieve around $1/|\mathcal{R}|$ forward propagation time of TRWP by parallelization over directions.

Method	PyTorch GPU		CUDA		*PyTorch GPU/CUDA*	
	32	96	32	96	32	96
ISGMR-4	7.3762	21.4818	**0.0105**	**0.0311**	**702**	**691**
ISGMR-8	18.8833	55.9235	**0.0234**	**0.0672**	**807**	**832**
ISGMR-16	58.2265	173.0217	**0.0618**	**0.1818**	**942**	**952**
TRWP-4	7.3460	21.4514	**0.0089**	**0.0265**	**825**	**810**
TRWP-8	18.8586	55.9392	**0.0208**	**0.0585**	**907**	**956**
TRWP-16	58.2642	172.9487	**0.0558**	**0.1628**	**1044**	**1062**

Table 3: *Backpropagation time (seconds) on a 256*512 image with 32 and 96 disparities for one iteration. Time of PyTorch GPU is averaged over 10 times and CUDA over 1000 times. CUDA version is 691 – 1062 times faster than PyTorch GPU version.*

However, the proposal of multiple streams cannot work well due to GPU resource limitations. Moreover due to the usage of messages from the previous iteration, ISGMR is slower than TRWP. In addition since no post-processing such as disparity consistency and interpolation is used, results in Figure 4 have occlusion and disparity mismatches in such as untextured areas. This, in turn, indicates the importance of learning.

7. Conclusion

In this paper, we have introduced two fast and differentiable message passing algorithms, namely, ISGMR and TRWP. While ISGMR improved the effectiveness of SGM, TRWP sped up TRWS by two orders of magnitude without compromising on the final solution quality. Besides, our CUDA implementations of these algorithms achieved at least 7 times and 650 times speedup compared to PyTorch GPU versions in forward and backward propagation, respectively. These practical improvements enable end-to-end learning with state-of-the-art message passing algorithms, and we intend to explore the benefits in large scale computer vision applications as future work.

8. Acknowledgements

This work is supported by the Australian Research Council Centre of Excellence for Robotic Vision (CE140100016) and Data61, the Commonwealth Scientific and Industrial Research Organisation.

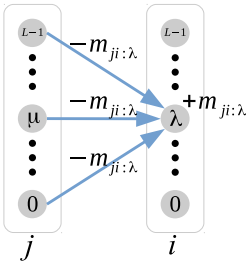


Figure 5: *Energy function maintained in iterative message passing. When adding a term $m_{ji:\lambda}$ to node i at label λ, the same value should be subtracted through all edges connecting node i at label λ.*

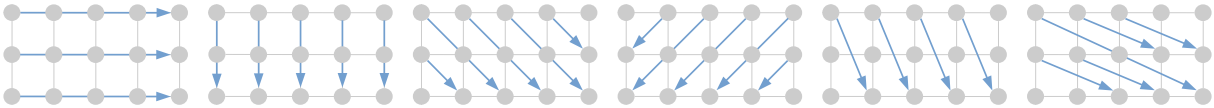


Figure 6: *Multi-direction message passing (forward passing in 6 directions). (a) horizontal trees. (b) vertical trees. (c) symmetric trees from up-left to down-right. (d) symmetric trees from up-right to down-left. (e) asymmetric narrow trees with height and width steps $S = (S_h, S_w) = (2, 1)$. (f) asymmetric wide trees with $S = (1, 2)$.*

Appendices

A. Maintaining Energy Function in Iterations

With the same notations in Eq.(1) and Eq.(7) in the main paper, for a general energy function in MRF below,

$$E(\mathbf{x}|\mathbf{\Theta}) = \sum_{i\in\mathcal{V}} \theta_i(x_i) + \sum_{(i,j)\in\mathcal{E}} \theta_{i,j}(x_i, x_j) \ . \tag{12}$$

In the standard SGM and ISGMR, given a node i and an edge from nodes j to i, the message will be updated as follows,

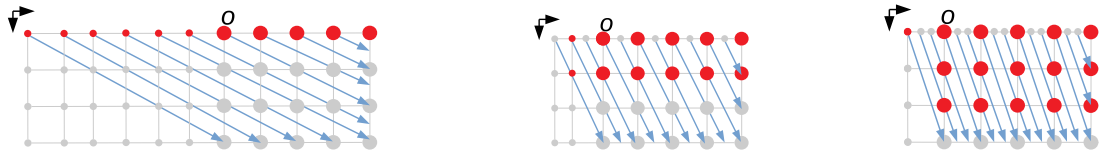
$$\hat{m}^r_{i:\lambda} = \min_{\mu\in\mathcal{L}} \left(\theta_{i-r:\mu} + \hat{m}^r_{i-r:\mu} + \sum_{d\in\mathcal{R}\setminus\{r,r^-\}} m^d_{i-r:\mu} + \theta_{i-r,i}(\mu,\lambda)\right) \ . \tag{13}$$

In Figure 5, if we add an extra term $m_{i:\lambda} := m_{ji:\lambda}$ to node i at label λ, *i.e.*, $\theta_{i:\lambda}$, the same value should be subtracted in order to maintain Eq. (12). This supports the exclusion of r^- from $\mathcal{R}$ in Eq. (13). This is important for multiple iterations because the non-zero messages after the 1st iteration, as extra terms, will change the energy function via Eq. (13). Hence, a simple combination of many standard SGMs will change the energy function due to the lack of the subtraction above.

B. Indexing First Nodes by Interpolation

Tree graphs contain horizontal, vertical, and diagonal (including symmetric, asymmetric wide, and asymmetric narrow) trees, shown in Figure 6. Generally speaking, horizontal and vertical trees are for 4-connected graphs, symmetric trees are for 8-connected graphs, and asymmetric trees are for more than 8-connected graphs. They have different ways of indexing the first nodes for parallelization. In the following, we denote an image size as height H and width W, coordinates of the first node in vertical and horizontal directions as p_h and p_w respectively, and scanning steps in vertical and horizontal directions as S_h and S_w respectively.

Horizontal and vertical graph trees. Coordinate of the first node of a horizontal and vertical tree, $p = (p_h, p_w)$, can be presented by $(p_h, 0)$ and $(0, p_w)$ respectively in the forward pass, and $(p_h, W-1)$ and $(H-1, p_w)$ respectively in the backward pass.



(a) (b) (c)

Figure 7: *Interpolation in asymmetric graph trees in forward passing. (a) asymmetric wide trees with steps $S = (1, 2)$. (b) asymmetric narrow trees with $S = (2, 1)$. (c) asymmetric narrow trees with $S = (3, 1)$. Red circles are first nodes of trees; large circles are within image size; small circles are interpolated; o is axes center. Coordinates of interpolations in (a) are integral; in (b)-(c) they will be round to the nearest integers by Eq. (15).*

Symmetric and asymmetric wide graph trees. Coordinate of the first node $p = (p_h, p_w)$ is calculated by

$$\begin{aligned} N &= W + (H-1) * \text{abs}(S_w) \,, \\ p_w &= [0 : N-1] - (H-1) * \max(S_w, 0) \,, \\ p_h &= \begin{cases} 0 & \text{if } S_h > 0 \,, \\ H-1 & \text{otherwise} \,, \end{cases} \end{aligned} \tag{14}$$

where N is number of graph trees, $\text{abs}(*)$ is absolution, and T_s is shifted indices of trees.

Asymmetric narrow graph trees. Coordinate of the first node p by interpolation is calculated by

$$\begin{aligned} c_1 &= \text{mod}(T_s, \text{abs}(S_h)) \,, \\ c_2 &= \frac{\text{float}(T_s)}{\text{float}(\text{abs}(S_h))} \,, \\ p_h &= \begin{cases} \text{mod}(\text{abs}(S_h) - c_1, \text{abs}(S_h)) & \text{if } S_w > 0 \,, \\ c_1 & \text{otherwise} \,, \end{cases} \\ p_h &= H - 1 - p_h \text{ if } S_h < 0, \\ p_w &= \begin{cases} \text{ceil}(c_2) & \text{if } S_w > 0 \,, \\ \text{floor}(c_2) & \text{otherwise} \,, \end{cases} \end{aligned} \tag{15}$$

where $\text{mod}(*)$ is modulo, $\text{floor}(*)$ and $\text{ceil}(*)$ are two integer approximations, $\text{float}(*)$ is data conversion for single-precision floating-point value, and the rest share the same notations in Eq. (14).

Although ISGMR and TRWP are parallelized over individual trees, message updates on a tree are sequential. The interpolation for asymmetric diagonals avoids as many redundant scanning as possible, shown in Figure 7. This is more practical for realistic stereo image pairs that the width is much larger than the height.

C. Differentiability of ISGMR

C.1. Explicit Representation of Forward Propagation

Since message update in ISGMR relies on recursively updated messages $\hat{\mathbf{m}}^r$ in each scanning direction r and messages $\mathbf{m}^r$ from all the other directions updated in the previous iteration, an explicit ISGMR message update is

$$\hat{m}^r_{i:\lambda} = \min_{\mu \in \mathcal{L}} \left(\theta_{i-r:\mu} + \hat{m}^r_{i-r:\mu} + \sum_{d \in \mathcal{R} \setminus \{r, r^-\}} m^d_{i-r:\mu} + \theta_{i-r,i}(\mu, \lambda) \right), \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L}, \forall r \in \mathcal{R} \,. \tag{16}$$

Applying message reparametrization by

$$\hat{m}^r_{i:\lambda} = \hat{m}^r_{i:\lambda} - \min_{k \in \mathcal{L}} \hat{m}^r_{i:k}, \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L}, \forall r \in \mathcal{R} \,. \tag{17}$$

After updating messages in **all directions within an iteration**, we assign the updated message $\hat{\mathbf{m}}$ to $\mathbf{m}$ by

$$m_{i:\lambda} = \hat{m}_{i:\lambda}, \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L} \,. \tag{18}$$

Eventually **after all iterations**, unary potentials and updated messages from all directions will be aggregated by

$$c_{i:\lambda} = \theta_{i:\lambda} + \sum_{d \in \mathcal{R}} m^d_{i:\lambda}, \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L} \,. \tag{19}$$

Different from optimization with winner-takes-all for labelling by $\mathbf{x}_i = \text{argmin}_{\lambda \in \mathcal{L}} c_{i:\lambda}, \forall i \in \mathcal{V}$, for learning, a regression with disparity confidences calculated by the final costs is used to fit with the real-valued ground truth disparities $\mathbf{g} = \{g_i\}, \forall i \in \mathcal{V}$. Generally, the disparity confidence $f_{i:\lambda}$ with a normalization such as SoftMin() can be represented by

$$f_{i:\lambda} = \text{SoftMin}(c_{i:\lambda}), \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L} \,, \tag{20}$$

and the regression for real-valued disparity $\mathbf{d} = \{d_i\}, \forall i \in \mathcal{V}$ is

$$d_i = \sum_{\lambda \in \mathcal{L}} \lambda f_{i:\lambda}, \forall i \in \mathcal{V} \,. \tag{21}$$

The loss function $L(\mathbf{d}, \mathbf{g})$ in learning can be standard L1 or smooth L1 loss function.

C.2. Derivations of Differentiability

Now we do backpropagation at kth iteration for learnable parameters $\{\theta_i, \theta_{i,j}\}$. With the same notations in Section 4 in the main paper, $\{p^r_{k,i:\lambda}\}$ and $\{q^r_{k,i}\}$ are indices stored in the forward propagation from message minimization and reparameterization respectively, and $\nabla * = dL/d*$.

C.2.1 Gradients of unary potentials

Proposition: Gradients of unary potentials $\{\theta_{i:\lambda}\}$ are represented by

$$\begin{aligned} \nabla\theta_{i:\lambda} &= \nabla c_{i:\lambda} + \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla \hat{m}^r_{i+r:v}\big|_{\lambda = p^r_{k,i+r:v}} \\ &= \nabla c_{i:\lambda} + \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \sum_{\mu \in \mathcal{L}} \Big(\nabla \hat{m}^r_{i+2r:\mu}\big|_{v = p^r_{k,i+2r:\mu}} + \sum_{d \in \mathcal{R} \setminus \{r, r^-\}} \nabla m^d_{i+r+d:\mu}\big|_{v = p^d_{k,i+r+d:\mu}} \Big) \Big|_{\lambda = p^r_{k,i+r:v}} \,. \end{aligned} \tag{22}$$

Derivation:
The backpropagation from Eq. (21)-Eq. (16) is

$$\begin{aligned} \nabla\theta_{i:\lambda} = \frac{dL}{d\theta_{i:\lambda}} &= \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \frac{\partial L}{\partial d_{j:v}} \frac{\partial d_{j:v}}{\partial f_{j:v}} \frac{\partial f_{j:v}}{\partial c_{j:v}} \frac{\partial c_{j:v}}{\partial \theta_{i:\lambda}} && \triangleright \text{back Eq. (21)-Eq. (20)} \\ &= \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \nabla c_{j:v} \Big(\frac{\partial c_{j:v}}{\partial \theta_{j:v}} \frac{\partial \theta_{j:v}}{\partial \theta_{i:\lambda}} + \sum_{r \in \mathcal{R}} \frac{\partial c_{j:v}}{\partial m^r_{j:v}} \frac{\partial m^r_{j:v}}{\partial \theta_{i:\lambda}} \Big) && \triangleright \text{back Eq. (19)} \\ &= \nabla c_{i:\lambda} + \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla m^r_{j:v} \frac{\partial m^r_{j:v}}{\partial \theta_{i:\lambda}} \\ &= \nabla c_{i:\lambda} + \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla m^r_{j:v} \frac{\partial m^r_{j:v}}{\partial \hat{m}^r_{j:v}} \frac{\partial \hat{m}^r_{j:v}}{\partial \theta_{i:\lambda}} && \triangleright \text{back Eq. (18)} \\ &= \nabla c_{i:\lambda} + \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla \hat{m}^r_{j:v} \frac{\partial \hat{m}^r_{j:v}}{\partial \theta_{i:\lambda}} \,. \end{aligned} \tag{23}$$

With backpropagation of Eq. (17) using an implicit message reparametrization with index $v^* = q^r_{k,j}$ at kth iteration, $\nabla \hat{m}^r_{j:v}$ in the second term above is updated by

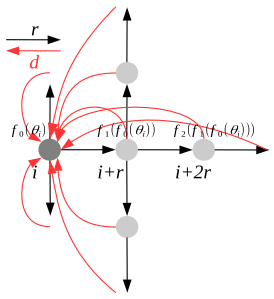


Figure 8: *Gradient accumulation of node i. Unary potential θ_i is passed along the forward directions in black arrows r, thus its gradient has several components from directions in red arrows d, $f_0(\cdot)$, $f_1(\cdot)$, and $f_2(\cdot)$ are message calculations with θ_i as an input. Note that direction d is not the backpropagation direction since the backpropagation is executed sequentially on a tree, for instance, $i+2r \rightarrow i+r \rightarrow i$ rather than $i+2r \rightarrow i$.*

$$\nabla \hat{m}^r_{j:v} \leftarrow \begin{cases} \nabla \hat{m}^r_{j:v} & \text{if } v \neq v^* \,, \\ -\sum_{v' \in \mathcal{L} \setminus v^*} \nabla \hat{m}^r_{j:v'} & \text{otherwise} \,. \end{cases} \tag{24}$$

Derivation of Eq. (24):
Explicit representation of Eq. (17) is $\tilde{m}^r_{i:\lambda} = \hat{m}^r_{i:\lambda} - \hat{m}^r_{i:\lambda^*}$, where $\lambda^* = q^r_{k,i}$, then we have

$$\begin{aligned} \nabla \hat{m}^r_{i:\lambda} = \frac{\partial L}{\partial \hat{m}^r_{i:\lambda}} &= \sum_{i' \in \mathcal{V}} \sum_{\lambda' \in \mathcal{L}} \frac{\partial L}{\partial \tilde{m}^r_{i':\lambda'}} \frac{\partial \tilde{m}^r_{i':\lambda'}}{\partial \hat{m}^r_{i:\lambda}} \\ &= \sum_{i' \in \mathcal{V}} \sum_{\lambda' \in \mathcal{L}} \frac{\partial L}{\partial \tilde{m}^r_{i':\lambda'}} \left(\frac{\partial \tilde{m}^r_{i':\lambda'}}{\partial \hat{m}^r_{i':\lambda'}} \frac{\partial \hat{m}^r_{i':\lambda'}}{\partial \hat{m}^r_{i:\lambda}} + \frac{\partial \tilde{m}^r_{i':\lambda'}}{\partial \hat{m}^r_{i':\lambda^*}} \frac{\partial \hat{m}^r_{i':\lambda^*}}{\partial \hat{m}^r_{i:\lambda}} \right) \\ &= \frac{\partial L}{\partial \tilde{m}^r_{i:\lambda}} - \sum_{\lambda' \in \mathcal{L}} \frac{\partial L}{\partial \tilde{m}_{i:\lambda'}} \Big|_{\lambda = \lambda^*} \\ &= \begin{cases} \nabla \tilde{m}^r_{i:\lambda} & \text{if } \lambda \neq \lambda^* \,, \\ -\sum_{\lambda' \in \mathcal{L} \setminus \lambda^*} \nabla \tilde{m}^r_{i:\lambda'} & \text{otherwise} \,. \end{cases} \end{aligned} \tag{25}$$

Back to the implicit message reparametrization where $\nabla \tilde{m}^r$ is replaced by $\nabla \hat{m}^r$, we have

$$\nabla \hat{m}^r_{i:\lambda} = \begin{cases} \nabla \hat{m}^r_{i:\lambda} & \text{if } \lambda \neq \lambda^* \,, \\ -\sum_{\lambda' \in \mathcal{L} \setminus \lambda^*} \nabla \hat{m}^r_{i:\lambda'} & \text{otherwise} \,. \end{cases} \tag{26}$$

End of the derivation of Eq. (24).
Next, we continue the backpropagation through Eq. (16) for unary potentials as

$$\begin{aligned} \nabla \theta_{i:\lambda} &= \nabla c_{i:\lambda} + \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla \hat{m}^r_{j:v} \frac{\partial \hat{m}^r_{j:v}}{\partial \theta_{i:\lambda}} && \triangleright \text{from Eq. (23)} \\ &= \nabla c_{i:\lambda} + \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla \hat{m}^r_{i+r:v} \frac{\partial \hat{m}^r_{i+r:v}}{\partial \theta_{i:\lambda}} && \triangleright \text{back Eq. (16) without recursion} \\ &= \nabla c_{i:\lambda} + \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla \hat{m}^r_{i+r:v} \big|_{\lambda = p^r_{k,i+r:v}} && \triangleright \text{satisfy argmin() rule in Eq. (16).} \end{aligned} \tag{27}$$

Derivation of $\nabla c_{i:\lambda}$ by backpropagation from the loss function, disparity regression, and SoftMin(), can be obtained by PyTorch autograd directly. In Eq. (27), the recursion is illustrated in Figure 8. For the readability of derivations by avoiding using $\{m^r_{i+r}(m^r_i(\theta_{i-r:\lambda})), m^r_{i+2r}(m^r_{i+r}(m^r_i(\theta_{i-r:\lambda}))), ...\}$, we do not write the recursion of gradients in the derivations. In the following, we derive $\nabla \hat{m}^r_{i+r:v}$ in the backpropagation.

C.2.2 Gradients of Messages

For notation readability, we first derive message gradient $\nabla \hat{m}^r_{i:\lambda}$ instead of $\nabla \hat{m}^r_{i+r:v}$.

Proposition: Gradients of messages $\{\hat{m}^r_{i:\lambda}\}$ are represented by

$$\nabla \hat{m}^r_{i:\lambda} = \sum_{v\in\mathcal{L}} \Big(\nabla \hat{m}^r_{i+r:v}\big|_{\lambda=p^r_{k,i+r:v}} + \sum_{d\in\mathcal{R}\setminus\{r,r^-\}} \nabla m^d_{i+d:v}\big|_{\lambda=p^d_{k,i+d:v}} \Big) . \tag{28}$$

Derivation:

$$\begin{aligned}
\nabla \hat{m}^r_{i:\lambda} = \frac{dL}{d\hat{m}^r_{i:\lambda}} &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v} \frac{\partial c_{j:v}}{\partial \hat{m}^r_{i:\lambda}} && \triangleright \text{back Eq. (21)-Eq. (20)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v} \sum_{d\in\mathcal{R}} \frac{\partial c_{j:v}}{\partial m^d_{j:v}} \frac{\partial m^d_{j:v}}{\partial \hat{m}^r_{i:\lambda}} && \triangleright \text{back Eq. (19)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v} \sum_{d\in\mathcal{R}} \frac{\partial c_{j:v}}{\partial m^d_{j:v}} \frac{\partial m^d_{j:v}}{\partial \hat{m}^d_{j:v}} \frac{\partial \hat{m}^d_{j:v}}{\partial \hat{m}^r_{i:\lambda}} && \triangleright \text{back Eq. (18)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla \hat{m}^d_{j:v} \frac{\partial \hat{m}^d_{j:v}}{\partial \hat{m}^r_{i:\lambda}} ,
\end{aligned} \tag{29}$$

then we update $\nabla \hat{m}^d_{j:v}$ by Eq. (24) and continue as follows,

$$\begin{aligned}
\nabla \hat{m}^r_{i:\lambda} &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla \hat{m}^d_{j:v} \frac{\partial \hat{m}^d_{j:v}}{\partial \hat{m}^r_{i:\lambda}} && \triangleright \text{from Eq. (29)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla \hat{m}^d_{j:v} \Big(\sum_{\lambda'\in\mathcal{L}} \frac{\partial \hat{m}^d_{j:v}}{\partial \hat{m}^d_{j-d:\lambda'}} \frac{\partial \hat{m}^d_{j-d:\lambda'}}{\partial \hat{m}^r_{i:\lambda}} + \sum_{d'\in\mathcal{R}\setminus\{d,d^-\}} \sum_{\lambda'\in\mathcal{L}} \frac{\partial \hat{m}^d_{j:v}}{\partial m^{d'}_{j-d:\lambda'}} \frac{\partial m^{d'}_{j-d:\lambda'}}{\partial \hat{m}^r_{i:\lambda}} \Big) && \triangleright \text{back Eq. (16)} \\
&= \sum_{v\in\mathcal{L}} \Big(\nabla \hat{m}^r_{i+r:v}\big|_{\lambda=p^r_{k,i+r:v}} + \sum_{d\in\mathcal{R}} \sum_{d'\in\mathcal{R}\setminus\{d,d^-\}} \sum_{\lambda'\in\mathcal{L}} \nabla \hat{m}^d_{j:v} \frac{\partial \hat{m}^d_{j:v}}{\partial m^{d'}_{j-d:\lambda'}} \frac{\partial m^{d'}_{j-d:\lambda'}}{\partial \hat{m}^r_{i:\lambda}} \Big) .
\end{aligned} \tag{30}$$

Since $m^{d'}_{j-d:\lambda'}$ is differentiable by $\hat{m}^r_{i:\lambda}$ due to Eq. (18) and, for ISGMR, message gradients in directions except the current direction r come from the next iteration (since in the forward propagation these messages come from the previous iteration), we have

$$\begin{aligned}
\nabla \hat{m}^r_{i:\lambda} &= \sum_{v\in\mathcal{L}} \Big(\nabla \hat{m}^r_{i+r:v}\big|_{\lambda=p^r_{k,i+r:v}} + \sum_{d\in\mathcal{R}} \sum_{d'\in\mathcal{R}\setminus\{d,d^-\}} \sum_{\lambda'\in\mathcal{L}} \nabla \hat{m}^d_{j:v} \frac{\partial \hat{m}^d_{j:v}}{\partial m^{d'}_{j-d:\lambda'}} \frac{\partial m^{d'}_{j-d:\lambda'}}{\partial \hat{m}^r_{i:\lambda}} \Big) && \triangleright \text{from Eq. (30)} \\
&= \sum_{v\in\mathcal{L}} \Big(\nabla \hat{m}^r_{i+r:v}\big|_{\lambda=p^r_{k,i+r:v}} + \sum_{d\in\mathcal{R}} \nabla \hat{m}^d_{i+d:v} \frac{\partial \hat{m}^d_{i+d:v}}{\partial m^r_{i:\lambda}} \frac{\partial m^r_{i:\lambda}}{\partial \hat{m}^r_{i:\lambda}}\Big|_{r\notin\{d,d^-\}} \Big) && \triangleright \text{due to Eq. (18)} \\
&= \sum_{v\in\mathcal{L}} \Big(\nabla \hat{m}^r_{i+r:v}\big|_{\lambda=p^r_{k,i+r:v}} + \sum_{d\in\mathcal{R}\setminus\{r,r^-\}} \nabla m^d_{i+d:v}\big|_{\lambda=p^d_{k,i+d:v}} \Big) .
\end{aligned} \tag{31}$$

Here updating the message gradient at node i depends on its next node $i+r$ along the scanning direction r; this scanning direction is opposite to the forward scanning direction, and thus, it depends on node $i+r$ instead of $i-r$. Gradient of message $m^r_{i:\lambda}$ can be derived in the same way.

Now one can derive $\nabla \hat{m}^r_{i+r:v}$ in the same manner of $\nabla \hat{m}^r_{i:\lambda}$ and apply it to Eq. (27) to obtain Eq. (22).

C.2.3 Gradient of Pairwise Potentials

Proposition: Gradients of pairwise potentials $\{\theta_{i-r,i}(\mu,\lambda)\}$ are represented by

$$\nabla\theta_{i-r,i}(\mu,\lambda) = \nabla\hat{m}^r_{i:\lambda}|_{\mu=p^r_{k,i:\lambda}}, \quad \forall i\in\mathcal{V}, \forall r\in\mathcal{R}, \forall\lambda,\mu\in\mathcal{L}\,. \tag{32}$$

Derivation:

$$\begin{aligned}
\nabla\theta_{i-r,i}(\mu,\lambda) = \frac{dL}{d\theta_{i-r,i}(\mu,\lambda)} &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v}\frac{\partial c_{j:v}}{\partial\theta_{i-r,i}(\mu,\lambda)} && \triangleright\text{back Eq. (21)-Eq. (20)}\\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v}\sum_{d\in\mathcal{R}}\frac{\partial c_{j:v}}{\partial m^d_{j:v}}\frac{\partial m^d_{j:v}}{\partial\theta_{i-r,i}(\mu,\lambda)} && \triangleright\text{back Eq. (19)}\\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla c_{j:v}\frac{\partial c_{j:v}}{\partial m^d_{j:v}}\frac{\partial m^d_{j:v}}{\partial\hat{m}^d_{j:v}}\frac{\partial\hat{m}^d_{j:v}}{\partial\theta_{i-r,i}(\mu,\lambda)} && \triangleright\text{back Eq. (18)}\\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}^d_{j:v}\frac{\partial\hat{m}^d_{j:v}}{\partial\theta_{i-r,i}(\mu,\lambda)}\,.
\end{aligned} \tag{33}$$

Now we update $\nabla\hat{m}^d_{j:v}$ by Eq. (24). Then

$$\begin{aligned}
\nabla\theta_{i-r,i}(\mu,\lambda) &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}^d_{j:v}\frac{\partial\hat{m}^d_{j:v}}{\partial\theta_{i-r,i}(\mu,\lambda)} && \triangleright\text{from Eq. (33)}\\
&= \nabla\hat{m}^r_{i:\lambda}|_{\mu=p^r_{k,i:\lambda}} && \triangleright\text{back Eq. (16) without recursion}\,.
\end{aligned} \tag{34}$$

One can note that the memory requirement of $\{\theta_{i-r,i}(\mu,\lambda)\}$ is $4\sum_{r\in\mathcal{R}}|\mathcal{E}^r||\mathcal{L}||\mathcal{L}|$ bytes using single-precision floating-point values. This will be high when the number of disparities $|\mathcal{L}|$ is large. In practical, since the pairwise potentials can be decomposed by $\theta_{i,j}(\lambda,\mu) = \theta_{i,j}V(\lambda,\mu), \forall(i,j)\in\mathcal{E}, \forall\lambda,\mu\in\mathcal{L}$ with edge weights $\theta_{i,j}$ and a pairwise function $V(\cdot,\cdot)$, it takes up $4(\sum_{r\in\mathcal{R}}|\mathcal{E}^r|+|\mathcal{L}||\mathcal{L}|)$ bytes in total, which is much less than $4\sum_{r\in\mathcal{R}}|\mathcal{E}^r||\mathcal{L}||\mathcal{L}|$ above. Therefore, we additionally provide the gradient derivations of these two terms, edge weights and pairwise functions, for practical implementations of the backpropagation.

C.2.4 Gradient of Edge Weights

Proposition: Gradients of edge weights $\{\theta_{i-r,i}\}$ are represented by

$$\nabla\theta_{i-r,i} = \sum_{v\in\mathcal{L}}\nabla\hat{m}^r_{i:v}V(p^r_{k,i:v},v), \quad \forall i\in\mathcal{V}, \forall r\in\mathcal{R}\,. \tag{35}$$

Derivation:

$$\begin{aligned}
\nabla\theta_{i-r,i} = \frac{dL}{d\theta_{i-r,i}} &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v}\frac{\partial c_{j:v}}{\partial\theta_{i-r,i}} && \triangleright\text{back Eq. (21)-Eq. (20)}\\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v}\sum_{d\in\mathcal{R}}\frac{\partial c_{j:v}}{\partial m^d_{j:v}}\frac{\partial m^d_{j:v}}{\partial\theta_{i-r,i}} && \triangleright\text{back Eq. (19)}\\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla c_{j:v}\frac{\partial c_{j:v}}{\partial m^d_{j:v}}\frac{\partial m^d_{j:v}}{\partial\hat{m}^d_{j:v}}\frac{\partial\hat{m}^d_{j:v}}{\partial\theta_{i-r,i}} && \triangleright\text{back Eq. (18)}\\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}^d_{j:v}\frac{\partial\hat{m}^d_{j:v}}{\partial\theta_{i-r,i}}\,.
\end{aligned} \tag{36}$$

Again, before updating gradients of edge weights by Eq. (16), $\nabla\hat{m}^d_{j:v}$ is updated by Eq. (24). Then

$$
\begin{aligned}
\nabla\theta_{i-r,i} &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}^{d}_{j:v}\frac{\partial\hat{m}^{d}_{j:v}}{\partial\theta_{i-r,i}} && \triangleright\text{from Eq. (36)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}^{d}_{j:v}\frac{\partial\hat{m}^{d}_{j:v}}{\partial\theta_{j-d,j}}V(p^{d}_{k,j:v},v)\frac{\partial\theta_{j-d,j}}{\partial\theta_{i-r,i}} && \triangleright\text{back Eq. (16) without recursion} \\
&= \sum_{v\in\mathcal{L}} \nabla\hat{m}^{r}_{i:v}V(p^{r}_{k,i:v},v)\ .
\end{aligned} \tag{37}
$$

In the case that when edge weights are undirected, *i.e.*, $\theta_{i,j}=\theta_{j,i}$, the derivations above still hold, and if $\theta_{i,j}=\theta_{j,i}$ are stored in the same tensor, $\nabla\theta_{i,j}$ will be accumulated by adding $\nabla\theta_{j,i}$ for storing the gradient of this edge weight. This is also applied to the gradient of pairwise potentials in Eq. (32) above.

C.2.5 Gradients of Pairwise Functions

Proposition: Gradients of a pairwise function $V(\cdot,\cdot)$ are

$$
\nabla V(\lambda,\mu) = \sum_{j\in\mathcal{V}}\sum_{r\in\mathcal{R}} \theta_{j-r,j}\nabla\hat{m}^{r}_{j:\mu}\big|_{\lambda=p^{r}_{k,j:\mu}}, \quad \forall\lambda,\mu\in\mathcal{L}\ . \tag{38}
$$

Derivation:

$$
\begin{aligned}
\nabla V(\lambda,\mu) = \frac{dL}{dV(\lambda,\mu)} &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v}\frac{\partial c_{j:v}}{\partial V(\lambda,\mu)} && \triangleright\text{back Eq. (21)-Eq. (20)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_{j:v}\sum_{r\in\mathcal{R}}\frac{\partial c_{j:v}}{\partial m^{r}_{j:v}}\frac{\partial m^{r}_{j:v}}{\partial V(\lambda,\mu)} && \triangleright\text{back Eq. (19)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla c_{j:v}\frac{\partial c_{j:v}}{\partial m^{r}_{j:v}}\frac{\partial m^{r}_{j:v}}{\partial\hat{m}^{r}_{j:v}}\frac{\partial\hat{m}^{r}_{j:v}}{\partial V(\lambda,\mu)} && \triangleright\text{back Eq. (18)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla\hat{m}^{r}_{j:v}\frac{\partial\hat{m}^{r}_{j:v}}{\partial V(\lambda,\mu)}\ .
\end{aligned} \tag{39}
$$

$\nabla\hat{m}^{r}_{j:v}$ is updated by Eq. (24). Then

$$
\begin{aligned}
\nabla V(\lambda,\mu) &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla\hat{m}^{r}_{j:v}\frac{\partial\hat{m}^{r}_{j:v}}{\partial V(\lambda,\mu)} && \triangleright\text{from Eq. (39)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla\hat{m}^{r}_{j:v}\sum_{\lambda'\in\mathcal{L}}\frac{\partial\hat{m}^{r}_{j:v}}{\partial V(\lambda',v)}\frac{\partial V(\lambda',v)}{\partial V(\lambda,\mu)} && \triangleright\text{from Eq. (16)} \\
&= \sum_{j\in\mathcal{V}}\sum_{r\in\mathcal{R}} \theta_{j-r,j}\nabla\hat{m}^{r}_{j:\mu}\big|_{\lambda=p^{r}_{k,j:\mu}}\ .
\end{aligned} \tag{40}
$$

C.3. Characteristics of Backpropagation

1. Accumulation. Since a message update usually has several components, its gradient is therefore accumulated when backpropagating through every component. For instance, in Eq. (27), the gradient of unary potential $\nabla\theta_{i:\lambda}$ has $\nabla c_{i:\lambda}$ and $\nabla\hat{m}^{r}_{i+r:v},\forall r\in\mathcal{R}$ and $\forall v$ satisfying $\lambda=p^{r}_{k,i+r,v}$ at kth iteration. It is calculated recursively but not at once due to multiple nodes on a tree, multiple directions, and multiple iterations. In Eq. (31), the message gradient of a node relies on the gradient of all nodes after it in the forward propagation since this message will be used to all the message updates after this node. One can also see Figure 8 for a clear understanding.

2. Zero Out Gradients. Message gradients are not accumulated throughout the backpropagation but should be zeroed out in some cases. In more details, in the forward propagation, the repeated usage of $\mathbf{m^r}$ and $\hat{\mathbf{m}}^\mathbf{r}$ is for all iterations but the messages are, in fact, new variables whenever they are updated. Since the gradient of a new message must be initialized to 0, zeroing out the gradients of the new messages is important. Specifically, in ISGMR that within an iteration $\mathbf{m^r} \leftarrow \hat{\mathbf{m}}^\mathbf{r}$ is executed only when message updates in all directions are done. Thus, $\nabla\mathbf{m^r}$ must be zeroed out after $\nabla\hat{\mathbf{m}}^\mathbf{r} \leftarrow \nabla\mathbf{m^r}$. Similarly, after using $\nabla\hat{\mathbf{m}}^\mathbf{r}$ to update the gradients of learnable parameters and messages, $\nabla\hat{\mathbf{m}}^\mathbf{r} \leftarrow 0, \forall r \in \mathcal{R}$.

D. Computational Complexity of Min-Sum and Sum-Product TRW

Given a graph with parameters $\{\theta_i, \theta_{i,j}\}$, maximum iteration K, set of edges $\{\mathcal{E}^r\}$, disparities $\mathcal{L}$, directions $\mathcal{R}$, computational complexities of min-sum and sum-product TRW are shown below. For the efficient implementation, we use $\theta_{i,j}(\lambda,\mu) = \theta_{i,j}V(\lambda,\mu)$ below.

D.1. Computational Complexity of Min-Sum TRW

Representation of a message update in min-sum TRW is

$$m^r_{i:\lambda} = \min_{\mu\in\mathcal{L}} \Big(\rho_{i-r,i}\big(\theta_{i-r:\mu} + \sum_{d\in\mathcal{R}} m^d_{i-r:\mu}\big) - m^{r^-}_{i-r:\mu} + \theta_{i-r,i}V(\mu,\lambda)\Big)\,, \tag{41}$$

In our case where the maximum disparity is less than 256, memory for the backpropagation of the min-sum TRW above is only for indices $\mu^* = p^r_{k,i:\lambda} \in \mathcal{L}$ from message minimization with $K\sum_{r\in\mathcal{R}}|\mathcal{E}^r||\mathcal{L}|$ bytes 8-bit unsigned integer values, as well as for indices from message reparametrization with $K\sum_{r\in\mathcal{R}}|\mathcal{E}^r|$ bytes. In total, the min-sum TRW needs $\underline{K\sum_{r\in\mathcal{R}}|\mathcal{E}^r|\,(|\mathcal{L}|+1)}$ bytes for the backpropagation.

D.2. Computational Complexity of Sum-Product TRW

Representation of a message update in sum-product TRW is

$$\begin{aligned}\exp(-m^r_{i:\lambda}) &= \sum_{\mu\in\mathcal{L}} \exp\Big(-\rho_{i-r,i}\big(\theta_{i-r:\mu} + \sum_{d\in\mathcal{R}} m^d_{i-r:\mu}\big) + m^{r^-}_{i-r:\mu} - \theta_{i-r,i}V(\mu,\lambda)\Big)\\ &= \sum_{\mu\in\mathcal{L}}\Big(\exp\big(-\rho_{i-r,i}\theta_{i-r:\mu}\big)\prod_{d\in\mathcal{R}}\exp\big(-\rho_{i-r,i}m^d_{i-r:\mu}\big)\exp(m^{r^-}_{i-r:\mu})\exp\big(-\theta_{i-r,i}V(\mu,\lambda)\big)\Big)\,,\end{aligned} \tag{42}$$

Usually, it can be represented as

$$\tilde{m}^r_{i:\lambda} = \sum_{\mu\in\mathcal{L}}\Big(\exp^{-\rho_{i-r,i}\underline{\theta_{i-r:\mu}}_1}\prod_{d\in\mathcal{R}}\big(\underline{\tilde{m}^d_{i-r:\mu}}_2\big)^{\rho_{i-r,i}}\frac{1}{\underline{\tilde{m}^{r^-}_{i-r:\mu}}_3}\exp^{-\underline{\theta_{i-r,i}}_4\underline{V(\mu,\lambda)}_5}\Big)\,. \tag{43}$$

Problem 1: Numerical Overflow: For single-precision floating-point data, a valid numerical range of x in $\exp(x)$ is less than around 88.7229; otherwise, it will be infinite. Therefore, for the exponential index in Eq. (42), a numerical overflow will happen quite easily. One solution is to reparametrize these messages to a small range, such as $[0,1]$, in the same manner as SoftMax(), which requires logarithm to find the maximum index, followed by exponential operations.

Problem 2: Low efficiency OR high memory requirement in backpropagation: In the backpropagation, due to the factorization in Eq. (43), it needs to rerun the forward propagation to calculate intermediate values OR store all these values in the forward propagation. However, the former makes the backpropagation at least as slow as the forward propagation while the later requires a large memory, $\underline{K\sum_{r\in\mathcal{R}}|\mathcal{E}^r||\mathcal{L}|\,(8|\mathcal{L}|+4|\mathcal{R}||\mathcal{L}|+4)}$ bytes single-precision floating-point values.

Derivation:

For one message update in Eq. (43), the gradient calculation of terms 1,2-3,4,5 (underlined) requires $4\times\{|\mathcal{L}|,|\mathcal{R}||\mathcal{L}|,1,|\mathcal{L}|\}$ bytes respectively. For K iterations, set of directions $\mathcal{R}$, edges $\{\mathcal{E}^r\}$, $\forall r \in \mathcal{R}$, it requires $K\sum_{r\in\mathcal{R}}|\mathcal{E}^r||\mathcal{L}|\,(8|\mathcal{L}|+4|\mathcal{R}||\mathcal{L}|+4)$ bytes in total. This is in $\mathcal{O}(|\mathcal{R}||\mathcal{L}|)$ order higher than the memory requirement in the min-sum TRW memory requirement, $K\sum_{r\in\mathcal{R}}|\mathcal{E}^r|\,(|\mathcal{L}|+1)$ bytes.

Method	Tsukuba		Teddy		Venus		Cones		Map	
	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter
TRWS-4	430639	**370161**	2141867	**2030380**	1333566	**1261215**	2649137	**2530628**	171787	**159878**
ISGMR-4	1265329	939195	3447655	2133414	6667307	2918766	3585829	2669778	242449	157010
ISGMR-8	1033629	**388437**	3278449	**1944355**	5872271	**1453744**	3308595	**2449702**	218559	**156132**
ISGMR-16	845868	413125	3091476	2027990	4475861	1561886	3033198	2615288	210261	170558
TRWP-4	1357230	**347061**	2832098	**1858952**	53484188	**1404753**	3644137	**2369795**	213834	**148562**
TRWP-8	710008	385325	2278520	1917406	13749612	1497536	2809347	2443408	176296	158625
TRWP-16	538336	438112	2149073	2029241	3695611	1641637	2646387	2662940	174548	169779

Table 4: *Energy minimization on Middlebury with threshold-based edge weights. ISGMR-8 and TRWP-4 have the lowest energies in ISGMR-related and TRWP-related methods respectively.*

Method	000002_11		000041_10		000119_10		delivery_area_1l		facade_1s	
	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter
TRWS-4	9960401	**9024636**	7702391	**7177320**	12249334	**10713682**	1816530	**1710997**	1088406	**1014900**
ISGMR-4	25544606	16008589	17140696	16551954	30963020	33952216	7394748	3273275	2326713	1586659
ISGMR-8	20067104	**9072815**	14771050	**6924595**	24838952	**11012039**	6563987	**1737898**	2113355	**1009487**
ISGMR-16	17485802	9807369	13250866	7066934	21705110	11656305	5460341	1763620	1885825	1071179
TRWP-4	47776632	**8584197**	40542256	**6738209**	49105384	**10212880**	18474580	**1698240**	4530643	**965446**
TRWP-8	21333600	9198270	17169866	6961071	26567380	11518438	7657029	1796645	1945868	1027150
TRWP-16	12305202	10069441	9362012	7198366	16847872	12384605	3209026	1826113	1300611	1101229

Table 5: *Energy minimization on 3 image pairs of KITTI2015 and 2 of ETH-3D with threshold-based edge weights. ISGMR-8 and TRWP-4 have the lowest energies in ISGMR-related and TRWP-related methods respectively.*

E. Additional Evaluations

E.1. Evaluations with Threshold-Based Edge Weights

The importance of edge weights $\theta_{i,j}$ in Eq. (12) lies on the label smoothness with edges preserved. In the standard SGM, however, all edge weights are set to 1. A widely used approach of calculating edge weights is to assign a high penalty to an edge that has a low image gradient in the form of an absolute difference of two pixels' intensities, under a preset threshold [26]. Given an RGB image with intensity $\mathbf{I} = \{I^r, I^g, I^b\}$, a threshold T, a gradient penalty $P > 1$, the edge weight of edge (i, j) includes but not limited to the following.

$$\theta_{i,j} = \begin{cases} P & \text{if } \frac{1}{3}\sum_{d\in\{r,g,b\}} |I_i^d - I_j^d|^2 < T \\ 1 & \text{otherwise} \end{cases} , \forall (i,j) \in \mathcal{E} \, . \tag{44}$$

Results are given in Tables 4-5 and Figures 9-18.

E.2. More Evaluations with Constant Edge Weights

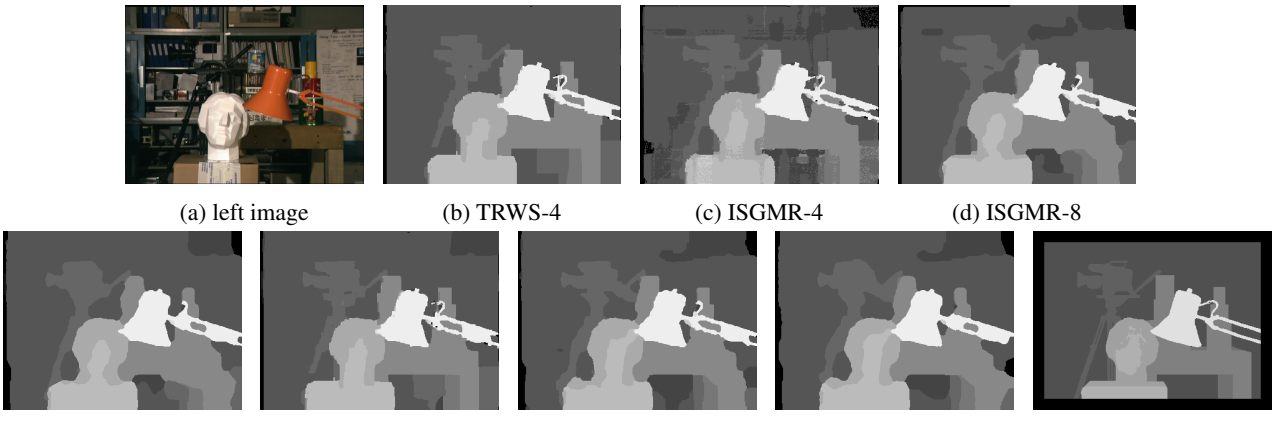
More results from the main experiments are given in Tables 6-7 and Figures 19-21.

Method	Tsukuba		Teddy		Venus		Cones		Map	
	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter
SGM-4	873777	644840	2825535	**2559016**	5119933	2637164	3697880	**3170715**	255054	**216713**
SGM-8	776706	**574758**	2868131	2728682	4651016	**2559933**	3631020	3309643	243058	222678
SGM-16	710727	587376	2907051	2846133	4081905	2720669	3564423	3413752	242932	232875
TRWS-4	352178	**314393**	1855625	**1807423**	1325651	**1219774**	2415087	**2329324**	150853	**143197**
ISGMR-4	824694	637996	2626648	1898641	4595032	1964032	3296594	2473646	215875	**148049**
ISGMR-8	684185	**340347**	2532071	**1847833**	4062167	**1285330**	3039638	**2398060**	195718	149857
ISGMR-16	591554	377427	2453592	1956343	3222851	1396914	2866149	2595487	190847	165249
TRWP-4	869363	**314037**	2234163	**1806990**	32896024	**1292619**	3284868	**2329343**	192200	**143364**
TRWP-8	496727	348447	1981582	1849287	8736569	1347060	2654033	2396257	162432	151970
TRWP-16	402033	396036	1935791	1976839	2636413	1486880	2524566	2660964	162655	164704

Table 6: *Energy minimization on Middlebury with constant edge weights. For Map, ISGMR-4 has the lowest energy among ISGMR-related methods; for others, ISGMR-8 and TRWP-4 have the lowest energies in ISGMR-related and TRWP-related methods respectively. ISGMR is more effective than SGM in optimization in both single and multiple iterations.*

Method	000002_11		000041_10		000119_10		delivery_area_1l		facade_1s	
	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter
SGM-4	24343250	18060026	15926416	12141643	24999424	18595020	5851489	**4267990**	1797314	**1429254**
SGM-8	20324684	**16406781**	13740635	**11671740**	20771096	**16652122**	5396353	4428411	1717285	1464208
SGM-16	18893122	16791762	13252150	12162330	19284684	16936852	5092094	4611821	1670997	1535778
TRWS-4	9109976	**8322635**	6876291	**6491169**	10811576	**9669367**	1628879	**1534961**	891282	**851273**
ISGMR-4	22259606	12659612	14434318	9984545	23180608	18541970	5282024	2212106	1572377	980151
ISGMR-8	17489158	**8753990**	11802603	**6639570**	18411930	**10173513**	4474404	**1571528**	1438210	**884241**
ISGMR-16	15455787	9556611	10731068	6806150	16608803	11037483	3689863	1594877	1324235	937102
TRWP-4	40473776	**8385450**	30399548	**6528642**	36873904	**9765540**	9899787	**1546795**	2851700	**854552**
TRWP-8	18424062	8860552	13319964	6678844	20581640	10445172	4443931	1587917	1358270	889907
TRWP-16	11239113	9736704	8187380	6895937	13602307	11309673	2261402	1630973	1000985	950607

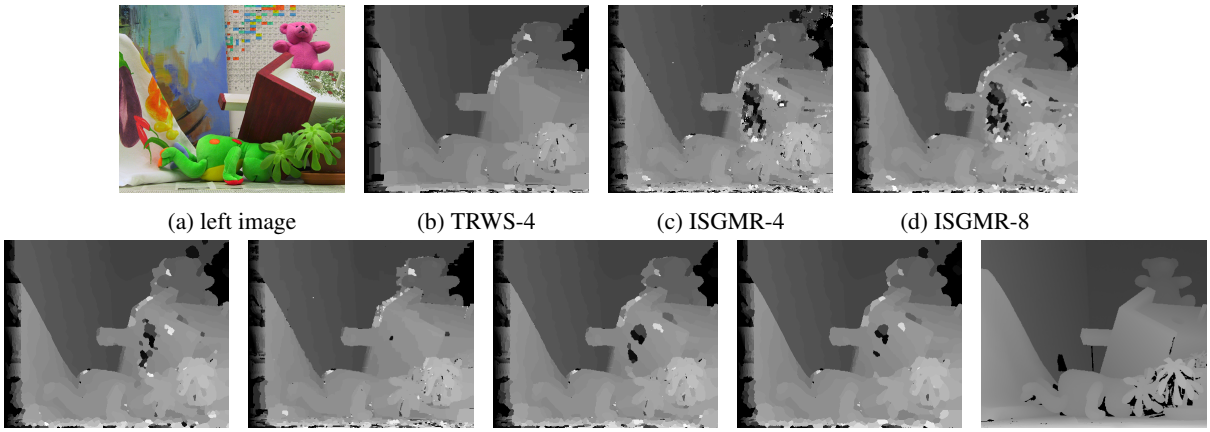
Table 7: *Energy minimization on 3 image pairs of KITTI2015 and 2 of ETH-3D with constant edge weights. ISGMR-8 and TRWP-4 have the lowest energies in ISGMR-related and TRWP-related methods respectively. ISGMR is more effective than SGM in optimization in both single and multiple iterations.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

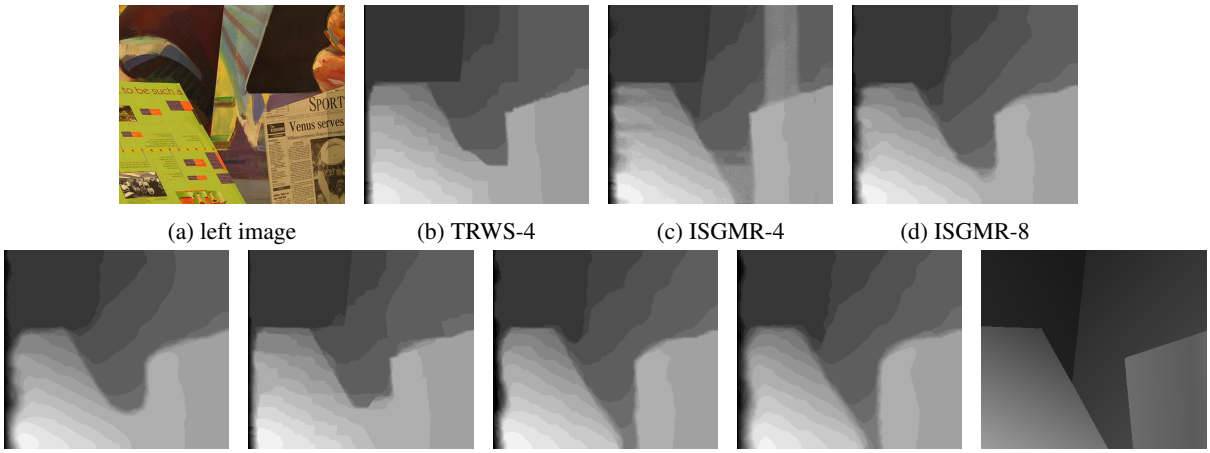
Figure 9: *Disparities of Tsukuba with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

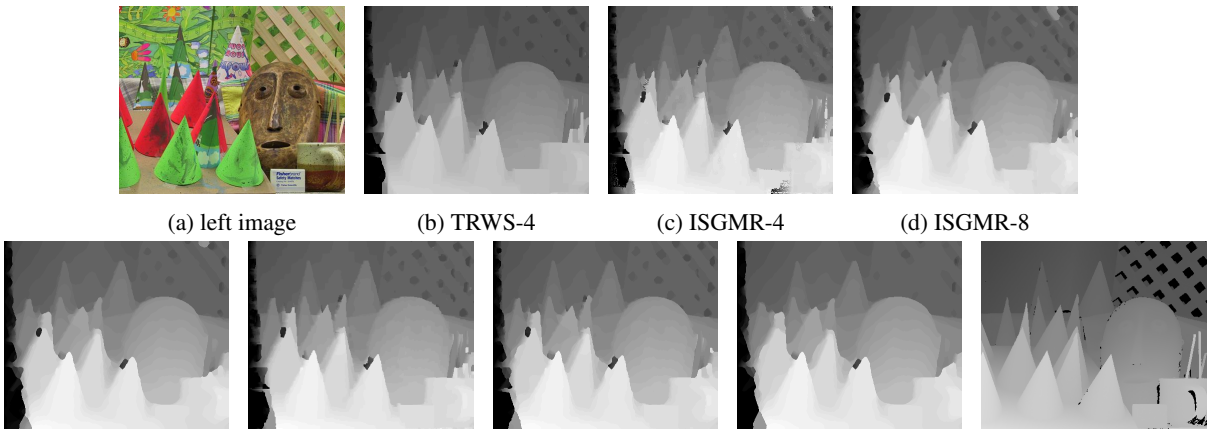
Figure 10: *Disparities of Teddy with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

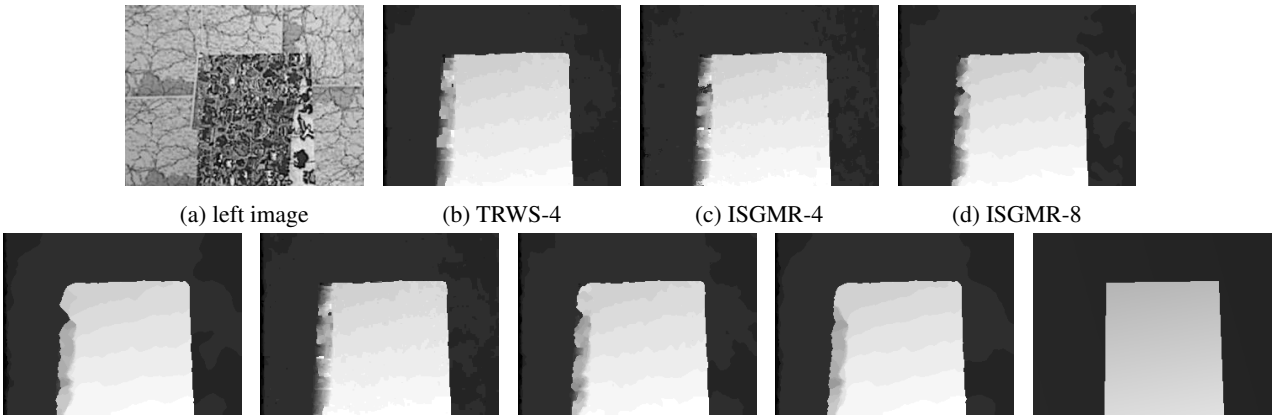
Figure 11: *Disparities of Venus with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

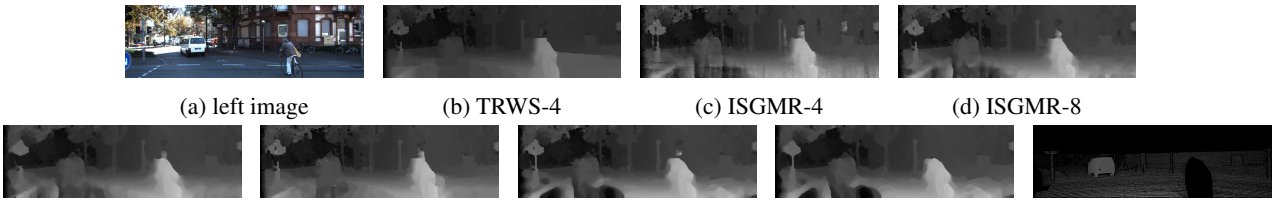
Figure 12: *Disparities of Cones with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

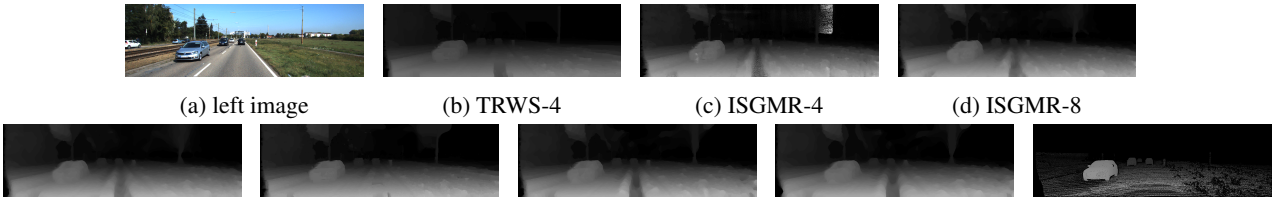
Figure 13: *Disparities of Map with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

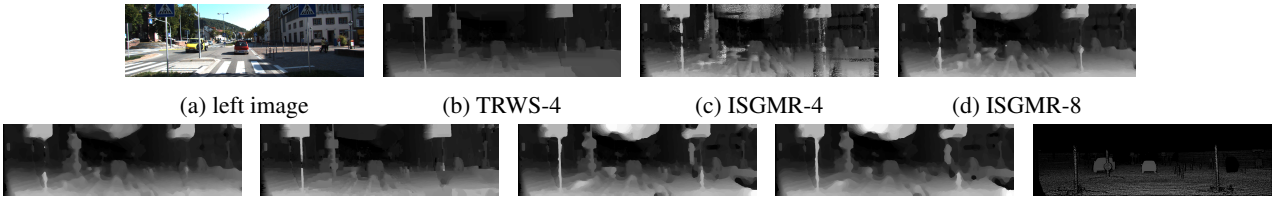
Figure 14: *Disparities of 000002_11 with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

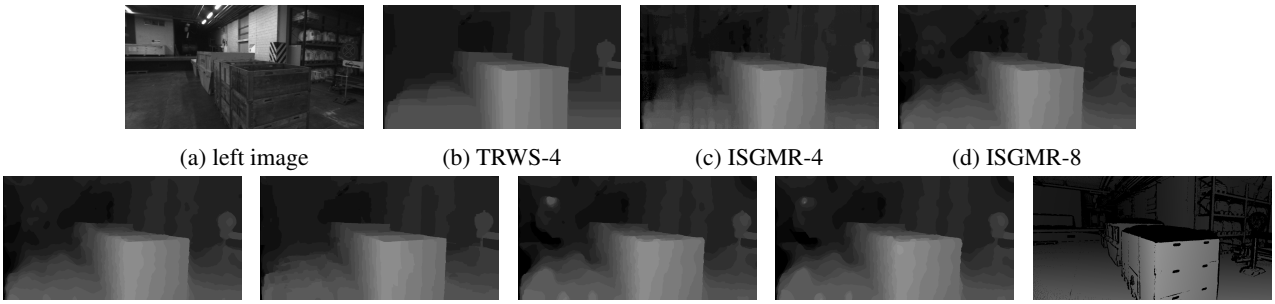
Figure 15: *Disparities of 000041_10 with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

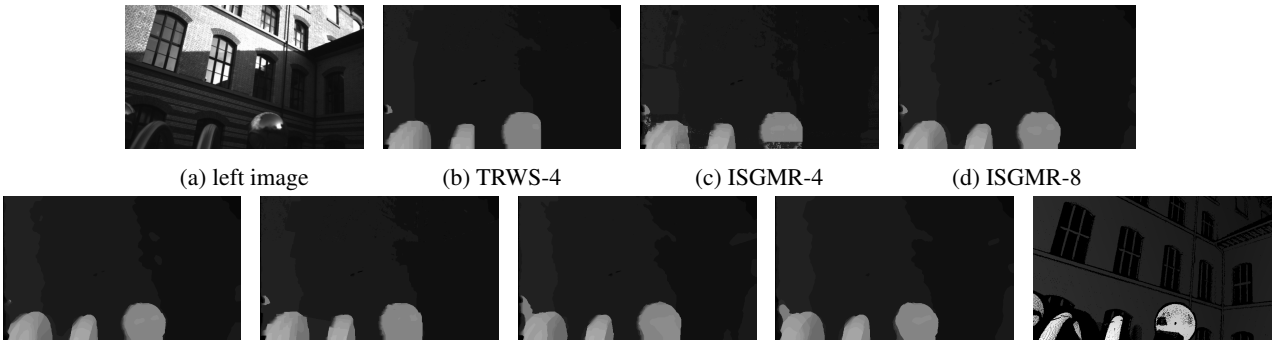
Figure 16: *Disparities of 000119_10 with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

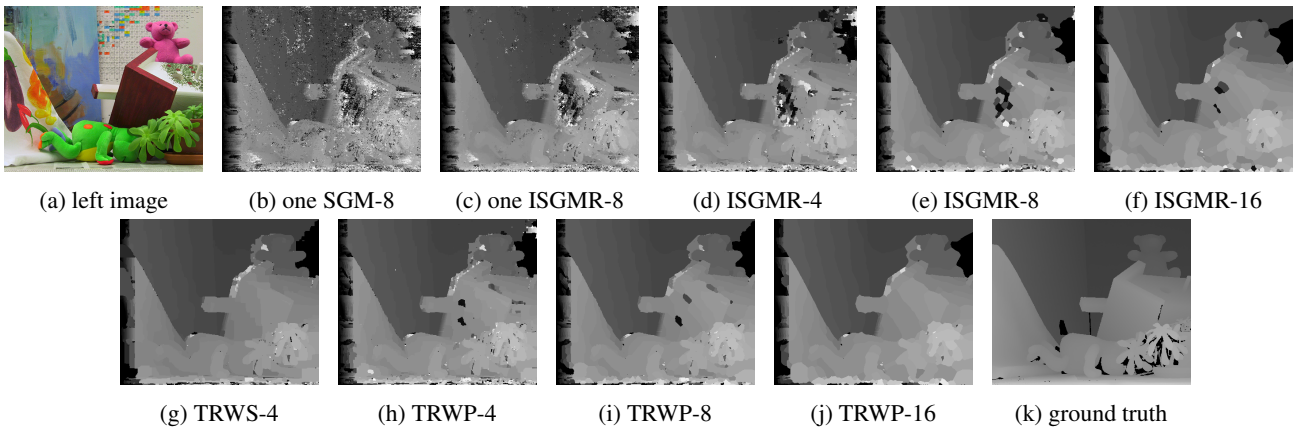
Figure 17: *Disparities of delivery_area_1l with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) TRWS-4 (c) ISGMR-4 (d) ISGMR-8

(e) ISGMR-16 (f) TRWP-4 (g) TRWP-8 (h) TRWP-16 (i) ground truth

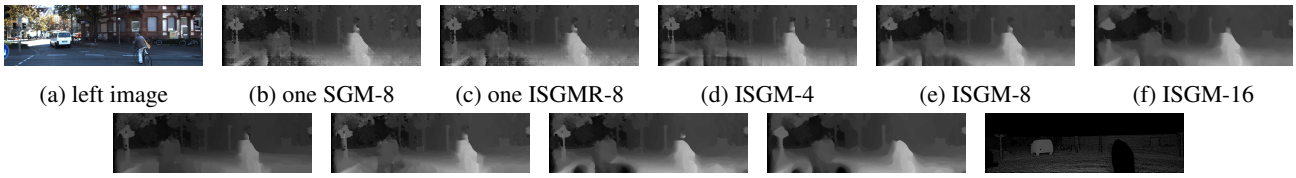
Figure 18: *Disparities of facade_1s with threshold-based edge weights at 50th iteration. (d) and (f) have the lowest energies in ISGMR-related and TRWP-related methods respectively. TRWP-4 and TRWS-4 have similar disparities for the most part.*



(a) left image (b) one SGM-8 (c) one ISGMR-8 (d) ISGMR-4 (e) ISGMR-8 (f) ISGMR-16

(g) TRWS-4 (h) TRWP-4 (i) TRWP-8 (j) TRWP-16 (k) ground truth

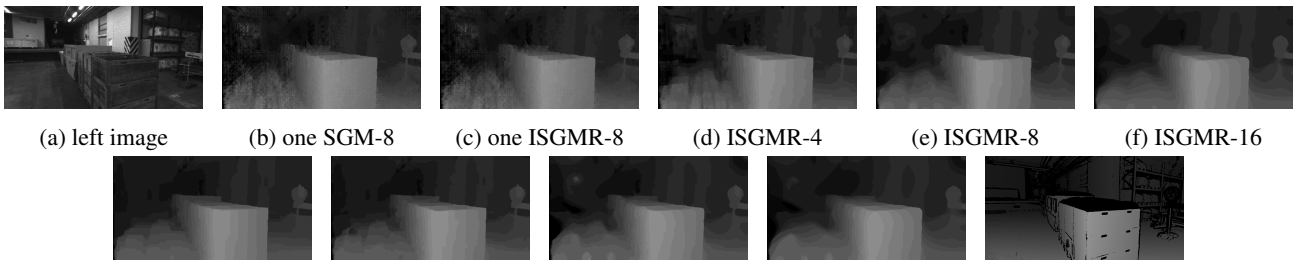
Figure 19: *Disparities of Teddy with constant edge weights over 50 iterations. (b)-(c) are at 1st iteration. (d)-(j) are at 50th iteration. (e) and (h) have the lowest energies in ISGMR-related and TRWP-related methods respectively.*



(a) left image (b) one SGM-8 (c) one ISGMR-8 (d) ISGM-4 (e) ISGM-8 (f) ISGM-16

(g) TRWS-4 (h) TRWP-4 (i) TRWP-8 (j) TRWP-16 (k) ground truth

Figure 20: *Disparities of 000002_11 with constant edge weights over 50 iterations. (b)-(c) are at 1st iteration. (d)-(j) are at 50th iteration. (e) and (h) have the lowest energies in ISGMR-related and TRWP-related methods respectively.*



(a) left image (b) one SGM-8 (c) one ISGMR-8 (d) ISGMR-4 (e) ISGMR-8 (f) ISGMR-16

(g) TRWS-4 (h) TRWP-4 (i) TRWP-8 (j) TRWP-16 (k) ground truth

Figure 21: *Disparities of delivery_area_1l with constant edge weights over 50 iterations. (b)-(c) are at 1st iteration. (d)-(j) are at 50th iteration. (e) and (h) have the lowest energies in ISGMR-related and TRWP-related methods respectively.*

References

[1] T. Ajanthan, R. Hartley, and M. Salzmann. Memory efficient max-flow for multi-label submodular mrfs. *CVPR*, 2016.

[2] T. Ajanthan, R. Hartley, M. Salzmann, and H. Li. Iteratively reweighted graph cut for multi-label mrfs with non-convex priors. *CVPR*, 2015.

[3] S. Birchfield and C. Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *TPAMI*, 1998.

[4] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 1986.

[5] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *TPAMI*, 2001.

[6] P. Carr and R. Hartley. Solving multilabel graph cut problems with multilabel swap. *DICTA*, 2009.

[7] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science and Engineering*, 1998.

[8] J. Domke. Learning graphical model parameters with approximate marginal inference. *TPAMI*, 2013.

[9] A. Drory, C. Haubold, S. Avidan, and F. A. Hamprecht. Semi-global matching: a principled derivation in terms of message passing. *Pattern Recognition*, 2014.

[10] G. Facciolo, C. Franchis, and E. Meinhardt. Mgm: A significantly more global matching for stereo vision. *BMVC*, 2015.

[11] R. Hartley and T. Ajanthan. Generalized range moves. *arXiv:1811.09171*, 2018.

[12] R. Hartley and A. Zisserman. Multiple view geometry in computer vision. *Cambridge university press*, 2003.

[13] D. Hernandez-Juare, A. Chacon, A. Espinosa, D. Vazquez, J. Moure, and A. M. Lopez. Embedded real-time stereo estimation via semi-global matching on the gpu. *International Conference on Computational Sciences*, 2016.

[14] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *TPAMI*, 2008.

[15] M. Jordan. Learning in graphical models. *MIT Press*, 1998.

[16] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *TPAMI*, 2006.

[17] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *CVPR*, 2016.

[18] M. Menze, C. Heipke, and A. Geiger. Joint 3d estimation of vehicles and scene flow. *ISPRS Workshop on Image Sequence Analysis*, 2015.

[19] M. Menze, C. Heipke, and A. Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2018.

[20] K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: an empirical study. *UAI*, 1999.

[21] J. Pearl. Probabilistic reasoning in intelligent systems. *Morgan Kaufmann*, 1988.

[22] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 2002.

[23] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. *CVPR*, 2003.

[24] T. Schops, J. Schonberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. *CVPR*, 2017.

[25] A. Seki and M. Pollefeys. Sgm-nets: Semi-global matching with neural networks. *CVPR*, 2017.

[26] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *TPAMI*, 2008.

[27] B. Taskar, C. Guestrin, and D. Koller. Max-margin markov networks. *MIT Press*, 2003.

[28] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 2005.

[29] O. Veksler. Multi-label moves for mrfs with truncated convex priors. *IJCV*, 2012.

[30] M. Wainwright, T. Jaakkola, and A. Willsky. Map estimation via agreement on (hyper) trees: Message-passing and linear-programming approaches. *Transactions on Information Theory*, 2005.

[31] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 2008.

[32] F. Zhang, V. Prisacariu, R. Yang, and P. H. Torr. Ga-net: Guided aggregation net for end-to-end stereo matching. *CVPR*, 2019.

[33] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. Torr. Conditional random fields as recurrent neural networks. *CVPR*, 2015.