Fast and Differentiable Message Passing on Pairwise Markov Random Fields – Supplementary Material –

Zhiwei Xu[1,2][0000−0001−8283−6095], Thalaiyasingam Ajanthan[1][0000−0002−6431−0775], and Richard Hartley[1][0000−0002−5005−0191]

[1] Australian National University and Australian Centre for Robotic Vision
[2] Data61, CSIRO, Canberra, Australia
{firstname.lastname}@anu.edu.au

A Pseudocode of Backpropagation of ISGMR and TRWP

Due to the limited space of the main paper, we provide the pseudocode of backpropagation of ISGMR and TRWP in this appendix, in Algorithms 3-4 respectively.

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 $\{p^r_{k,i}(\lambda)\}$, $\{q^r_{k,i}\}$, iteration number K. We replace $\nabla m^{r,k+1}$ by $\nabla \hat{m}^r$ and $\nabla m^{r,k}$ by ∇m^r for simplicity.

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

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

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 $\{p^r_{k,i}(\lambda)\}$, $\{q^r_{k,i}\}$, iteration number K.

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

$\nabla \mathbf{m^r} \leftarrow \nabla \mathbf{\Theta_i} \leftarrow d\mathbf{c}$, $d\mathbf{\Theta_{i,j}} \leftarrow 0$ ▷back Eq. (7)
for iteration $k \in \{K, ..., 1\}$ **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**
        $\lambda^* \leftarrow q^r_{k,i} \in \mathcal{L}$ ▷extract index
        $\nabla m^r_i(\lambda^*) \mathrel{-}= \sum_{\lambda\in\mathcal{L}} \nabla m^r_i(\lambda)$ ▷back Eq. (5)
        for label $\lambda \in \mathcal{L}$ **do**
          $\mu^* \leftarrow p^r_{k,i}(\lambda) \in \mathcal{L}$ ▷extract index
          $\nabla\theta_{i-r}(\mu^*) \mathrel{+}= \rho_{i-r,i}\nabla m^r_i(\lambda)$ ▷back Eq. (11)
          $\nabla m^d_{i-r}(\mu^*) \mathrel{+}= \rho_{i-r,i}\nabla m^r_i(\lambda), \forall d \in \mathcal{R}$
          $\nabla m^{r^-}_{i-r}(\mu^*) \mathrel{-}= \nabla m^r_i(\lambda)$
          $\nabla\theta_{i-r,i}(\mu^*, \lambda) \mathrel{+}= \nabla m^r_i(\lambda)$
      $\nabla \mathbf{m}^r \leftarrow 0$ ▷zero-out

B Maintaining Energy Function in Iterations

With the same notations in Eq.(1) and Eq.(9) in the main paper, let a general energy function in a MRF defined as

$$E(\mathbf{x}|\mathbf{\Theta}) = \sum_{i\in\mathcal{V}} \theta_i(x_i) + \sum_{(i,j)\in\mathcal{E}} \theta_{i,j}(x_i, x_j) \ . \tag{14}$$

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

$$m^{r,k+1}_i(\lambda) = \min_{\mu\in\mathcal{L}} \left(\theta_{i-r}(\mu) + \theta_{i-r,i}(\mu,\lambda) + m^{r,k+1}_{i-r}(\mu) + \sum_{d\in\mathcal{R}\setminus\{r,r^-\}} m^{d,k}_{i-r}(\mu)\right) . \tag{15}$$

In Figure 7, however, if we add a term $m_i(\lambda)$ to node i at label λ via $m_{ji}(\lambda)$ from node j to node i at label λ, the same value should be subtracted along all edges connecting this node i, that is $\forall(i,j) \in \mathcal{E}$, in order to maintain the same Eq. (14) in optimization. This supports the exclusion of r^- from $\mathcal{R}$ in Eq. (15). This is important for multiple iterations because the non-zero messages after the 1st iteration, as additional terms, will change the energy function via Eq. (15). Hence, a simple combination of many standard SGMs will change the energy function due to the lack of the subtraction above.

C Indexing First Nodes by Interpolation

Tree graphs contain horizontal, vertical, and diagonal (including symmetric, asymmetric wide, and asymmetric narrow) trees, shown in Figure 8. Generally, the horizontal and

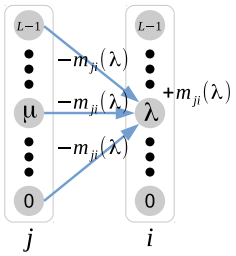
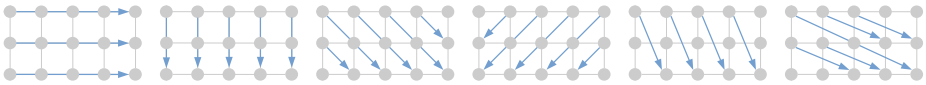


Fig. 7: Energy function maintained in iterative message passing. When adding a term $m_{ji}(\lambda)$ to node i at label λ, the same value should be subtracted on all edges connecting node i at label λ.

vertical trees are for 4-connected graphs, symmetric trees are for 8-connected graphs, and asymmetric trees are for more than 8-connected graphs, resulting in different ways of indexing the first nodes for parallelization. In the following, we denote an image size with 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.



(a) (b) (c) (d) (e) (f)

Fig. 8: 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 $S = (1, 2)$.

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.

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

$$\begin{aligned} N &= W + (H-1) * \text{abs}(S_w) \,, \\ p_w &= [0 : N-1] - (H-1) * \max(S_w, 0) \,, \\ p_h &= \begin{cases} 0 & \text{if } S_h > 0 \,, \\ H-1 & \text{otherwise} \,, \end{cases} \end{aligned} \tag{16}$$

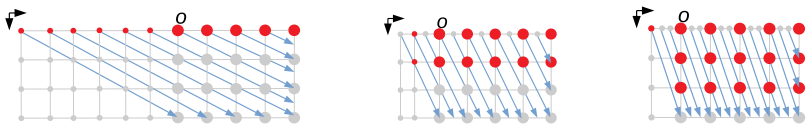
where N is tree number, $\text{abs}(*)$ is absolution, and T_s is shifted indices of trees.

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

$$
\begin{aligned}
c_1 &= \text{mod}(T_s, \text{abs}(S_h)) \ , \\
c_2 &= \frac{\text{float}(T_s)}{\text{float}(\text{abs}(S_h))} \ , \\
p_h &= \begin{cases} \text{mod}(\text{abs}(S_h) - c_1, \text{abs}(S_h)) & \text{if } S_w > 0 \ , \\ c_1 & \text{otherwise} \ , \end{cases} \\
p_h &= H - 1 - p_h \text{ if } S_h < 0 \ , \\
p_w &= \begin{cases} \text{ceil}(c_2) & \text{if } S_w > 0 \ , \\ \text{floor}(c_2) & \text{otherwise} \ , \end{cases}
\end{aligned} \tag{17}
$$

where $\text{mod}(*)$ is modulo, $\text{floor}(*)$ and $\text{ceil}(*)$ are two integer approximations, $\text{float}(*)$ is data conversion for single-precision floating-point values, and the rest share the same notations in Eq. (16).

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 9. This is more practical for realistic stereo image pairs that the width is much larger than the height.



(a) (b) (c)

Fig. 9: 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) round to the nearest integers by Eq. (17).

D Differentiability of ISGMR

Below, we replace $m^{r,k}$ by m^r and $m^{r,k+1}$ by $\hat{m}^r$ for simplicity. This is because from the practical implementation, messages in direction r should be updated instead of allocating new memories in each iteration to avoid GPU memory increase. Thus, we only use two variables m^r and $\hat{m}^r$ for messages before and after an iteration.

D.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

$$\hat{m}_i^r(\lambda) = \min_{\mu \in \mathcal{L}} \big(\theta_{i-r}(\mu) + \theta_{i-r,i}(\mu,\lambda) + \hat{m}_{i-r}^r(\mu) + \sum_{d \in \mathcal{R} \setminus \{r, r^-\}} m_{i-r}^d(\mu)\big), \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L}, \forall r \in \mathcal{R} \; . \tag{18}$$

Applying message reparametrization by

$$\hat{m}_i^r(\lambda) = \hat{m}_i^r(\lambda) - \min_{k \in \mathcal{L}} \hat{m}_i^r(k), \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L}, \forall r \in \mathcal{R} \; . \tag{19}$$

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

$$m_i(\lambda) = \hat{m}_i(\lambda), \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L} \; . \tag{20}$$

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

$$c_i(\lambda) = \theta_i(\lambda) + \sum_{d \in \mathcal{R}} m_i^d(\lambda), \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L} \; . \tag{21}$$

Different from optimization with winner-takes-all for labelling in learning by $\mathbf{x}_i = \text{argmin}_{\lambda \in \mathcal{L}} c_i(\lambda), \forall i \in \mathcal{V}$, 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() is represented by

$$f_i(\lambda) = \text{SoftMin}(c_i(\lambda)), \quad \forall i \in \mathcal{V}, \forall \lambda \in \mathcal{L} \; , \tag{22}$$

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

$$d_i = \sum_{\lambda \in \mathcal{L}} \lambda f_i(\lambda), \forall i \in \mathcal{V} \; . \tag{23}$$

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

D.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, $\{p_{k,i}^r(\lambda)\}$ and $\{q_{k,i}^r\}$ are indices stored in the forward propagation from message minimization and reparameterization respectively, and $\nabla * = dL/d*$.

D.2.1 Gradients of unary potentials

Proposition: Gradients of unary potentials $\{\theta_i(\lambda)\}$ are represented by

$$\begin{aligned}\nabla\theta_i(\lambda) &= \nabla c_i(\lambda) + \sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla\hat{m}^r_{i+r}(v)\big|_{\lambda=p^r_{k,i+r}(v)} \\ &= \nabla c_i(\lambda) + \sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}}\sum_{\mu\in\mathcal{L}} \Big(\nabla\hat{m}^r_{i+2r}(\mu)\big|_{v=p^r_{k,i+2r}(\mu)} \\ &\quad + \sum_{d\in\mathcal{R}\setminus\{r,r^-\}} \nabla m^d_{i+r+d}(\mu)\big|_{v=p^d_{k,i+r+d}(\mu)} \Big)\Big|_{\lambda=p^r_{k,i+r}(v)} .\end{aligned} \tag{24}$$

Derivation:

The backpropagation from Eq. (23)-Eq. (18) is

$$\begin{aligned}\nabla\theta_i(\lambda) &= \frac{dL}{d\theta_i(\lambda)} \\ &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \frac{\partial L}{\partial d_j(v)}\frac{\partial d_j(v)}{\partial f_j(v)}\frac{\partial f_j(v)}{\partial c_j(v)}\frac{\partial c_j(v)}{\partial\theta_i(\lambda)} && \triangleright\text{back Eq. (23)-Eq. (22)} \\ &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_j(v)\Big(\frac{\partial c_j(v)}{\partial\theta_j(v)}\frac{\partial\theta_j(v)}{\partial\theta_i(\lambda)} + \sum_{r\in\mathcal{R}}\frac{\partial c_j(v)}{\partial m^r_j(v)}\frac{\partial m^r_j(v)}{\partial\theta_i(\lambda)}\Big) && \triangleright\text{back Eq. (21)} \\ &= \nabla c_i(\lambda) + \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla m^r_j(v)\frac{\partial m^r_j(v)}{\partial\theta_i(\lambda)} \\ &= \nabla c_i(\lambda) + \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla m^r_j(v)\frac{\partial m^r_j(v)}{\partial\hat{m}^r_j(v)}\frac{\partial\hat{m}^r_j(v)}{\partial\theta_i(\lambda)} && \triangleright\text{back Eq. (20)} \\ &= \nabla c_i(\lambda) + \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla\hat{m}^r_j(v)\frac{\partial\hat{m}^r_j(v)}{\partial\theta_i(\lambda)} .\end{aligned} \tag{25}$$

With backpropagation of Eq. (19) 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

$$\nabla\hat{m}^r_j(v) \leftarrow \begin{cases} \nabla\hat{m}^r_j(v) & \text{if } v \neq v^* , \\ -\sum_{v'\in\mathcal{L}\setminus v^*} \nabla\hat{m}^r_j(v') & \text{otherwise} . \end{cases} \tag{26}$$

Derivation of Eq. (26):

Explicit representation of Eq. (19) 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

$$
\begin{aligned}
\nabla \hat{m}_i^r(\lambda) &= \frac{\partial L}{\partial \hat{m}_i^r(\lambda)} \\
&= \sum_{i' \in \mathcal{V}} \sum_{\lambda' \in \mathcal{L}} \frac{\partial L}{\partial \tilde{m}_{i'}^r(\lambda')} \frac{\partial \tilde{m}_{i'}^r(\lambda')}{\partial \hat{m}_i^r(\lambda)} \\
&= \sum_{i' \in \mathcal{V}} \sum_{\lambda' \in \mathcal{L}} \frac{\partial L}{\partial \tilde{m}_{i'}^r(\lambda')} \Big(\frac{\partial \tilde{m}_{i'}^r(\lambda')}{\partial \hat{m}_{i'}^r(\lambda')} \frac{\partial \hat{m}_{i'}^r(\lambda')}{\partial \hat{m}_i^r(\lambda)} + \frac{\partial \tilde{m}_{i'}^r(\lambda')}{\partial \hat{m}_{i'}^r(\lambda^*)} \frac{\partial \hat{m}_{i'}^r(\lambda^*)}{\partial \hat{m}_i^r(\lambda)} \Big) \\
&= \frac{\partial L}{\partial \tilde{m}_i^r(\lambda)} - \sum_{\lambda' \in \mathcal{L}} \frac{\partial L}{\partial \tilde{m}_i(\lambda')} \Big|_{\lambda = \lambda^*} \\
&= \begin{cases} \nabla \tilde{m}_i^r(\lambda) & \text{if } \lambda \neq \lambda^* \,, \\ -\sum_{\lambda' \in \mathcal{L} \setminus \lambda^*} \nabla \tilde{m}_i^r(\lambda') & \text{otherwise} \,. \end{cases}
\end{aligned} \tag{27}
$$

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

$$
\nabla \hat{m}_i^r(\lambda) = \begin{cases} \nabla \hat{m}_i^r(\lambda) & \text{if } \lambda \neq \lambda^* \,, \\ -\sum_{\lambda' \in \mathcal{L} \setminus \lambda^*} \nabla \hat{m}_i^r(\lambda') & \text{otherwise} \,. \end{cases} \tag{28}
$$

End of the derivation of Eq. (26).

Next, we continue the backpropagation through Eq. (18) for unary potentials as

$$
\begin{aligned}
\nabla \theta_i(\lambda) &= \nabla c_i(\lambda) + \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla \hat{m}_j^r(v) \frac{\partial \hat{m}_j^r(v)}{\partial \theta_i(\lambda)} && \triangleright \text{from Eq. (25)} \\
&= \nabla c_i(\lambda) + \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla \hat{m}_{i+r}^r(v) \frac{\partial \hat{m}_{i+r}^r(v)}{\partial \theta_i(\lambda)} && \triangleright \text{back Eq. (18) without recursion} \\
&= \nabla c_i(\lambda) + \sum_{v \in \mathcal{L}} \sum_{r \in \mathcal{R}} \nabla \hat{m}_{i+r}^r(v) \big|_{\lambda = p_{k,i+r}^r(v)} \,. && \triangleright \text{satisfy argmin() rule in Eq. (18)}
\end{aligned} \tag{29}
$$

Derivation of $\nabla c_i(\lambda)$ by backpropagation from the loss function, disparity regression, and SoftMin(), can be obtained by PyTorch autograd directly. For the readability of derivations by avoiding using $\{m_{i+r}^r(m_i^r(\theta_{i-r}(\lambda))), m_{i+2r}^r(m_{i+r}^r(m_i^r(\theta_{i-r}(\lambda)))), ...\}$, we do not write the recursion of gradients in the derivations. Below, we derive $\nabla \hat{m}_{i+r}^r(v)$ in the backpropagation.

D.2.2 Gradients of Messages

For notation readability, we first derive message gradient $\nabla \hat{m}_i^r(\lambda)$ instead of $\nabla \hat{m}_{i+r}^r(v)$.
Proposition: Gradients of messages $\{\hat{m}_i^r(\lambda)\}$ are represented by

$$
\nabla \hat{m}_i^r(\lambda) = \sum_{v \in \mathcal{L}} \Big(\nabla \hat{m}_{i+r}^r(v) \big|_{\lambda = p_{k,i+r}^r(v)} + \sum_{d \in \mathcal{R} \setminus \{r, r^-\}} \nabla m_{i+d}^d(v) \big|_{\lambda = p_{k,i+d}^d(v)} \Big) \,. \tag{30}
$$

Derivation:

$$
\begin{aligned}
\nabla \hat{m}_i^r(\lambda) &= \frac{dL}{d\hat{m}_i^r(\lambda)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_j(v) \frac{\partial c_j(v)}{\partial \hat{m}_i^r(\lambda)} && \triangleright \text{back Eq. (23)-Eq. (22)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_j(v) \sum_{d\in\mathcal{R}} \frac{\partial c_j(v)}{\partial m_j^d(v)} \frac{\partial m_j^d(v)}{\partial \hat{m}_i^r(\lambda)} && \triangleright \text{back Eq. (21)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_j(v) \sum_{d\in\mathcal{R}} \frac{\partial c_j(v)}{\partial m_j^d(v)} \frac{\partial m_j^d(v)}{\partial \hat{m}_j^d(v)} \frac{\partial \hat{m}_j^d(v)}{\partial \hat{m}_i^r(\lambda)} && \triangleright \text{back Eq. (20)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla \hat{m}_j^d(v) \frac{\partial \hat{m}_j^d(v)}{\partial \hat{m}_i^r(\lambda)} ,
\end{aligned}
\tag{31}
$$

then we update $\nabla \hat{m}_j^d(v)$ by Eq. (26) and continue as follows,

$$
\begin{aligned}
\nabla \hat{m}_i^r(\lambda) &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla \hat{m}_j^d(v) \frac{\partial \hat{m}_j^d(v)}{\partial \hat{m}_i^r(\lambda)} && \triangleright \text{from Eq. (31)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla \hat{m}_j^d(v) \Big(\sum_{\lambda'\in\mathcal{L}} \frac{\partial \hat{m}_j^d(v)}{\partial \hat{m}_{j-d}^d(\lambda')} \frac{\partial \hat{m}_{j-d}^d(\lambda')}{\partial \hat{m}_i^r(\lambda)} \\
&\quad + \sum_{d'\in\mathcal{R}\setminus\{d,d^-\}} \sum_{\lambda'\in\mathcal{L}} \frac{\partial \hat{m}_j^d(v)}{\partial m_{j-d}^{d'}(\lambda')} \frac{\partial m_{j-d}^{d'}(\lambda')}{\partial \hat{m}_i^r(\lambda)} \Big) && \triangleright \text{back Eq. (18)} \\
&= \sum_{v\in\mathcal{L}} \Big(\nabla \hat{m}_{i+r}^r(v)\big|_{\lambda=p_{k,i+r}^r(v)} \\
&\quad + \sum_{d\in\mathcal{R}} \sum_{d'\in\mathcal{R}\setminus\{d,d^-\}} \sum_{\lambda'\in\mathcal{L}} \nabla \hat{m}_j^d(v) \frac{\partial \hat{m}_j^d(v)}{\partial m_{j-d}^{d'}(\lambda')} \frac{\partial m_{j-d}^{d'}(\lambda')}{\partial \hat{m}_i^r(\lambda)} \Big) .
\end{aligned}
\tag{32}
$$

Since $m_{j-d}^{d'}(\lambda')$ is differentiable by $\hat{m}_i^r(\lambda)$ due to Eq. (20) 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

$$
\begin{aligned}
\nabla \hat{m}_i^r(\lambda) = \sum_{v \in \mathcal{L}} \Big(& \nabla \hat{m}_{i+r}^r(v)\big|_{\lambda = p_{k,i+r}^r(v)} \\
& + \sum_{d \in \mathcal{R}} \sum_{d' \in \mathcal{R} \setminus \{d, d^-\}} \sum_{\lambda' \in \mathcal{L}} \nabla \hat{m}_j^d(v) \frac{\partial \hat{m}_j^d(v)}{\partial m_{j-d}^{d'}(\lambda')} \frac{\partial m_{j-d}^{d'}(\lambda')}{\partial \hat{m}_i^r(\lambda)} \Big) && \triangleright \text{from Eq. (32)} \\
= \sum_{v \in \mathcal{L}} \Big(& \nabla \hat{m}_{i+r}^r(v)\big|_{\lambda = p_{k,i+r}^r(v)} \\
& + \sum_{d \in \mathcal{R}} \nabla \hat{m}_{i+d}^d(v) \frac{\partial \hat{m}_{i+d}^d(v)}{\partial m_i^r(\lambda)} \frac{\partial m_i^r(\lambda)}{\partial \hat{m}_i^r(\lambda)}\Big|_{r \notin \{d, d^-\}} \Big) && \triangleright \text{due to Eq. (20)} \\
= \sum_{v \in \mathcal{L}} \Big(& \nabla \hat{m}_{i+r}^r(v)\big|_{\lambda = p_{k,i+r}^r(v)} \\
& + \sum_{d \in \mathcal{R} \setminus \{r, r^-\}} \nabla m_{i+d}^d(v)\big|_{\lambda = p_{k,i+d}^d(v)} \Big) .
\end{aligned}
\tag{33}
$$

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 $m_i^r(\lambda)$ can be derived in the same way.

Now one can derive $\nabla \hat{m}_{i+r}^r(v)$ in the same manner of $\nabla \hat{m}_i^r(\lambda)$ and apply it to Eq. (29) to obtain Eq. (24).

D.2.3 Gradient of Pairwise Potentials

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

$$
\nabla \theta_{i-r,i}(\mu, \lambda) = \nabla \hat{m}_i^r(\lambda)|_{\mu = p_{k,i}^r(\lambda)} , \quad \forall i \in \mathcal{V}, \forall r \in \mathcal{R}, \forall \lambda, \mu \in \mathcal{L} . \tag{34}
$$

Derivation:

$$
\begin{aligned}
\nabla \theta_{i-r,i}(\mu, \lambda) &= \frac{dL}{d\theta_{i-r,i}(\mu, \lambda)} \\
&= \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \nabla c_j(v) \frac{\partial c_j(v)}{\partial \theta_{i-r,i}(\mu, \lambda)} && \triangleright \text{back Eq. (23)-Eq. (22)} \\
&= \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \nabla c_j(v) \sum_{d \in \mathcal{R}} \frac{\partial c_j(v)}{\partial m_j^d(v)} \frac{\partial m_j^d(v)}{\partial \theta_{i-r,i}(\mu, \lambda)} && \triangleright \text{back Eq. (21)} \\
&= \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \sum_{d \in \mathcal{R}} \nabla c_j(v) \frac{\partial c_j(v)}{\partial m_j^d(v)} \frac{\partial m_j^d(v)}{\partial \hat{m}_j^d(v)} \frac{\partial \hat{m}_j^d(v)}{\partial \theta_{i-r,i}(\mu, \lambda)} && \triangleright \text{back Eq. (20)} \\
&= \sum_{j \in \mathcal{V}} \sum_{v \in \mathcal{L}} \sum_{d \in \mathcal{R}} \nabla \hat{m}_j^d(v) \frac{\partial \hat{m}_j^d(v)}{\partial \theta_{i-r,i}(\mu, \lambda)} .
\end{aligned}
\tag{35}
$$

Now we update $\nabla \hat{m}_j^d(v)$ by Eq. (26). Then

$$\begin{aligned}\nabla\theta_{i-r,i}(\mu,\lambda) &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}_j^d(v)\frac{\partial \hat{m}_j^d(v)}{\partial\theta_{i-r,i}(\mu,\lambda)} && \triangleright\text{from Eq. (35)}\\ &= \nabla\hat{m}_i^r(\lambda)|_{\mu=p_{k,i}^r(\lambda)} \ . && \triangleright\text{back Eq. (18) without recursion}\end{aligned} \tag{36}$$

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.

D.2.4 Gradient of Edge Weights

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

$$\nabla\theta_{i-r,i} = \sum_{v\in\mathcal{L}} \nabla\hat{m}_i^r(v)V(p_{k,i}^r(v),v), \quad \forall i\in\mathcal{V}, \forall r\in\mathcal{R}\ . \tag{37}$$

Derivation:

$$\begin{aligned}\nabla\theta_{i-r,i} &= \frac{dL}{d\theta_{i-r,i}}\\ &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_j(v)\frac{\partial c_j(v)}{\partial\theta_{i-r,i}} && \triangleright\text{back Eq. (23)-Eq. (22)}\\ &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_j(v)\sum_{d\in\mathcal{R}}\frac{\partial c_j(v)}{\partial m_j^d(v)}\frac{\partial m_j^d(v)}{\partial\theta_{i-r,i}} && \triangleright\text{back Eq. (21)}\\ &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla c_j(v)\frac{\partial c_j(v)}{\partial m_j^d(v)}\frac{\partial m_j^d(v)}{\partial\hat{m}_j^d(v)}\frac{\partial\hat{m}_j^d(v)}{\partial\theta_{i-r,i}} && \triangleright\text{back Eq. (20)}\\ &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}_j^d(v)\frac{\partial\hat{m}_j^d(v)}{\partial\theta_{i-r,i}}\ .\end{aligned} \tag{38}$$

Again, before updating gradients of edge weights by Eq. (18), $\nabla\hat{m}_j^d(v)$ is updated by Eq. (26). Then

$$\begin{aligned}
\nabla\theta_{i-r,i} &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}_j^d(v)\frac{\partial\hat{m}_j^d(v)}{\partial\theta_{i-r,i}} && \triangleright\text{from Eq. (38)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{d\in\mathcal{R}} \nabla\hat{m}_j^d(v)\frac{\partial\hat{m}_j^d(v)}{\partial\theta_{j-d,j}}V(p_{k,j}^d(v),v)\frac{\partial\theta_{j-d,j}}{\partial\theta_{i-r,i}} && \triangleright\text{back Eq. (18), no recursion} \\
&= \sum_{v\in\mathcal{L}} \nabla\hat{m}_i^r(v)V(p_{k,i}^r(v),v)\ .
\end{aligned} \tag{39}$$

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. (34) above.

D.2.5 Gradients of Pairwise Functions

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

$$\nabla V(\lambda,\mu) = \sum_{j\in\mathcal{V}}\sum_{r\in\mathcal{R}} \theta_{j-r,j}\nabla\hat{m}_j^r(\mu)\big|_{\lambda=p_{k,j}^r(\mu)}\ , \quad \forall\lambda,\mu\in\mathcal{L}\ . \tag{40}$$

Derivation:

$$\begin{aligned}
\nabla V(\lambda,\mu) &= \frac{dL}{dV(\lambda,\mu)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_j(v)\frac{\partial c_j(v)}{\partial V(\lambda,\mu)} && \triangleright\text{back Eq. (23)-Eq. (22)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}} \nabla c_j(v)\sum_{r\in\mathcal{R}}\frac{\partial c_j(v)}{\partial m_j^r(v)}\frac{\partial m_j^r(v)}{\partial V(\lambda,\mu)} && \triangleright\text{back Eq. (21)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla c_j(v)\frac{\partial c_j(v)}{\partial m_j^r(v)}\frac{\partial m_j^r(v)}{\partial\hat{m}_j^r(v)}\frac{\partial\hat{m}_j^r(v)}{\partial V(\lambda,\mu)} && \triangleright\text{back Eq. (20)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla\hat{m}_j^r(v)\frac{\partial\hat{m}_j^r(v)}{\partial V(\lambda,\mu)}\ .
\end{aligned} \tag{41}$$

$\nabla\hat{m}_j^r(v)$ is updated by Eq. (26). Then

$$
\begin{aligned}
\nabla V(\lambda,\mu) &= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla\hat{m}_j^r(v)\frac{\partial\hat{m}_j^r(v)}{\partial V(\lambda,\mu)} && \triangleright\text{from Eq. (41)} \\
&= \sum_{j\in\mathcal{V}}\sum_{v\in\mathcal{L}}\sum_{r\in\mathcal{R}} \nabla\hat{m}_j^r(v)\sum_{\lambda'\in\mathcal{L}}\frac{\partial\hat{m}_j^r(v)}{\partial V(\lambda',v)}\frac{\partial V(\lambda',v)}{\partial V(\lambda,\mu)} && \triangleright\text{from Eq. (18)} \quad (42) \\
&= \sum_{j\in\mathcal{V}}\sum_{r\in\mathcal{R}} \theta_{j-r,j}\nabla\hat{m}_j^r(\mu)\big|_{\lambda=p_{k,j}^r(\mu)} \ .
\end{aligned}
$$

D.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. (29), the gradient of unary potential $\nabla\theta_i(\lambda)$ has $\nabla c_i(\lambda)$ and $\nabla\hat{m}_{i+r}^r(v), \forall r \in \mathcal{R}$ and $\forall v$ satisfying $\lambda = p_{k,i+r}^r(v)$ 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. (33), 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.

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.4 PyTorch GPU version *vs.* our CUDA version

For the compared PyTorch GPU version, we highly paralleled individual trees in each direction while sequential message updates in each tree (equally scanline) are iterative. As Pytorch auto-grad is not customized for our min-sum message passing algorithms, these iterative message updates require to allocate new GPU memory for each updated message, which makes it very inefficient and memory-consuming. Its backpropagation is slower since extra memory is needed to unroll the forward message passing to compute gradients of messages and all intermediate variables that require gradients.

In contrast, our implementation is specific to the min-sum message passing. This min-sum form greatly accelerates our backpropagation by updating gradients only related to the indices which are stored in pre-allocated GPU memory during forward pass (line 10 in Alg. 1). For example, from node i to $i + r$ in Fig. 2(a), forward pass needs messages over 9 edges (grey lines); but only one (1 of 3 blue lines) from $i + r$ to i requires gradient updates in the backpropagation. This makes our CUDA implementation

much faster than the PyTorch GPU version, especially the backpropagation with at least $700\times$ speed-up.

E Computational Complexity of Min-Sum & 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, let $\theta_{i,j}(\lambda, \mu) = \theta_{i,j} V(\lambda, \mu)$.

E.1 Computational Complexity of Min-Sum TRW

Representation of a message update in min-sum TRW is

$$m_i^r(\lambda) = \min_{\mu \in \mathcal{L}} \Big(\rho_{i-r,i} \big(\theta_{i-r}(\mu) + \sum_{d \in \mathcal{R}} m_{i-r}^d(\mu) \big) - m_{i-r}^{r^-}(\mu) + \theta_{i-r,i} V(\mu, \lambda) \Big) \,. \quad (43)$$

In our case where the maximum disparity is less than 256, memory for the backpropagatio of the min-sum TRW above is only for indices $\mu^* = p_{k,i}^r(\lambda) \in \mathcal{L}$ 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 $K \sum_{r \in \mathcal{R}} |\mathcal{E}^r| \, (|\mathcal{L}| + 1)$ bytes for the backpropagation.

E.2 Computational Complexity of Sum-Product TRW

Representation of a message update in sum-product TRW is

$$\begin{aligned} \exp(-m_i^r(\lambda)) &= \sum_{\mu \in \mathcal{L}} \exp \Big(- \rho_{i-r,i} \big(\theta_{i-r}(\mu) + \sum_{d \in \mathcal{R}} m_{i-r}^d(\mu) \big) + m_{i-r}^{r^-}(\mu) \\ &\quad - \theta_{i-r,i} V(\mu, \lambda) \Big) \\ &= \sum_{\mu \in \mathcal{L}} \Big(\exp \big(- \rho_{i-r,i} \theta_{i-r}(\mu) \big) \prod_{d \in \mathcal{R}} \exp \big(- \rho_{i-r,i} m_{i-r}^d(\mu) \big) \\ &\quad \exp(m_{i-r}^{r^-}(\mu)) \exp \big(- \theta_{i-r,i} V(\mu, \lambda) \big) \Big) \,. \end{aligned} \quad (44)$$

Usually, it can be represented as

$$\tilde{m}_i^r(\lambda) = \sum_{\mu \in \mathcal{L}} \Big(\exp^{-\rho_{i-r,i} \underline{\theta_{i-r}(\mu)}_1} \prod_{d \in \mathcal{R}} \big(\underline{\tilde{m}_{i-r}^d(\mu)}_2 \big)^{\rho_{i-r,i}} \frac{1}{\underline{\tilde{m}_{i-r}^{r^-}(\mu)}_3} \exp^{-\underline{\theta_{i-r,i}}_4 \underline{V(\mu,\lambda)}_5} \Big) \,. \quad (45)$$

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. (44), 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. (45), 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,
$K \sum_{r \in \mathcal{R}} |\mathcal{E}^r||\mathcal{L}| \left(8|\mathcal{L}| + 4|\mathcal{R}||\mathcal{L}| + 4\right)$ bytes single-precision floating-point values.
Derivation:

For one message update in Eq. (45), 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 $K \sum_{r \in \mathcal{R}} |\mathcal{E}^r||\mathcal{L}| \left(8|\mathcal{L}| + 4|\mathcal{R}||\mathcal{L}| + 4\right)$ 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, $K \sum_{r \in \mathcal{R}} |\mathcal{E}^r| \left(|\mathcal{L}| + 1\right)$ bytes.

F Additional Evaluations

F.1 More Evaluations with Constant Edge Weights

More results from the main experiments are given in Tables 6-7.

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, and TRWP-4 outperforms MF and SGM.

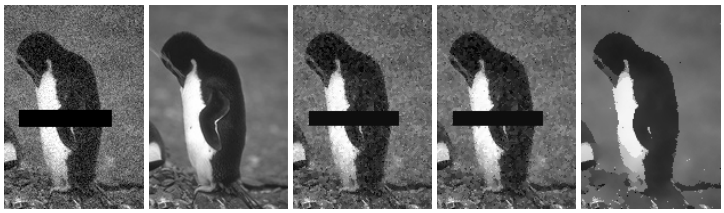
Method	Tsukuba		Teddy		Venus		Cones		Map	
	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter	1 iter	50 iter
MF-4	3121704	1620524	3206347	2583784	108494928	14618819	9686122	6379392	1116641	363221
SGM-4	873777	644840	2825535	2559016	5119933	2637164	3697880	3170715	255054	216713
TRWS-4	352178	314393	1855625	1807423	1325651	1219774	2415087	2329324	150853	143197
ISGMR-4 (ours)	824694	637996	2626648	1898641	4595032	1964032	3296594	2473646	215875	148049
TRWP-4 (ours)	869363	314037	2234163	1806990	32896024	1292619	3284868	2329343	192200	143364
MF-8	2322139	504815	3244710	2545226	68718520	2920117	7762269	3553975	840615	213827
SGM-8	776706	574758	2868131	2728682	4651016	2559933	3631020	3309643	243058	222678
ISGMR-8 (ours)	684185	340347	2532071	1847833	4062167	1285330	3039638	2398060	195718	149857
TRWP-8 (ours)	496727	348447	1981582	1849287	8736569	1347060	2654033	2396257	162432	151970
MF-16	1979155	404404	3315900	2622047	43077872	1981096	6741127	3062965	638753	204737
SGM-16	710727	587376	2907051	2846133	4081905	2720669	3564423	3413752	242932	232875
ISGMR-16 (ours)	591554	377427	2453592	1956343	3222851	1396914	2866149	2595487	190847	165249
TRWP-16 (ours)	402033	396036	1935791	1976839	2636413	1486880	2524566	2660964	162655	164704

F.2 More Visualizations for Image Denoising

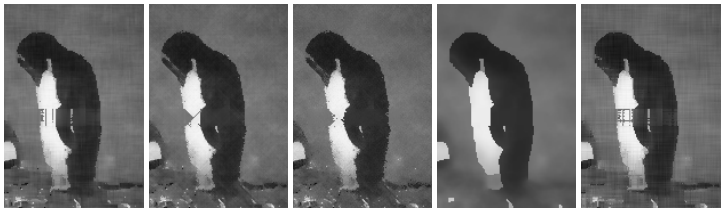
We provide more visualizations of image denoising on "Penguin" and "House" in Figures 10-11 corresponding to Table 2 in the main paper.

Table 7: Energy minimization on 3 image pairs of KITTI2015 and 2 of ETH-3D with constant edge weights. ISGMR is more effective than SGM in optimization in both single and multiple iterations, and TRWP-4 outperforms MF and SGM.

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
MF-4	82523536	44410056	69894016	36163508	72659040	42392548	19945352	9013862	13299859	6681882
SGM-4	24343250	18060026	15926416	12141643	24999424	18595020	5851489	4267990	1797314	1429254
TRWS-4	9109976	8322635	6876291	6491169	10811576	9669367	1628879	1534961	891282	851273
ISGMR-4 (ours)	22259606	12659612	14434318	9984545	23180608	18541970	5282024	2212106	1572377	980151
TRWP-4 (ours)	40473776	8385450	30399548	6528642	36873904	9765540	9899787	1546795	2851700	854552
MF-8	61157072	18416536	53302252	16473121	57201868	21320892	16581587	4510834	10978978	3422296
SGM-8	20324684	16406781	13740635	11671740	20771096	16652122	5396353	4428411	1717285	1464208
ISGMR-8 (ours)	17489158	8753990	11802603	6639570	18411930	10173513	4474404	1571528	1438210	884241
TRWP-8 (ours)	18424062	8860552	13319964	6678844	20581640	10445172	4443931	1587917	1358270	889907
MF-16	46614232	14192750	40838292	12974839	44706364	16708809	13223338	3229021	9189592	2631006
SGM-16	18893122	16791762	13252150	12162330	19284684	16936852	5092094	4611821	1670997	1535778
ISGMR-16 (ours)	15455787	9556611	10731068	6806150	16608803	11037483	3689863	1594877	1324235	937102
TRWP-16 (ours)	11239113	9736704	8187380	6895937	13602307	11309673	2261402	1630973	1000985	950607



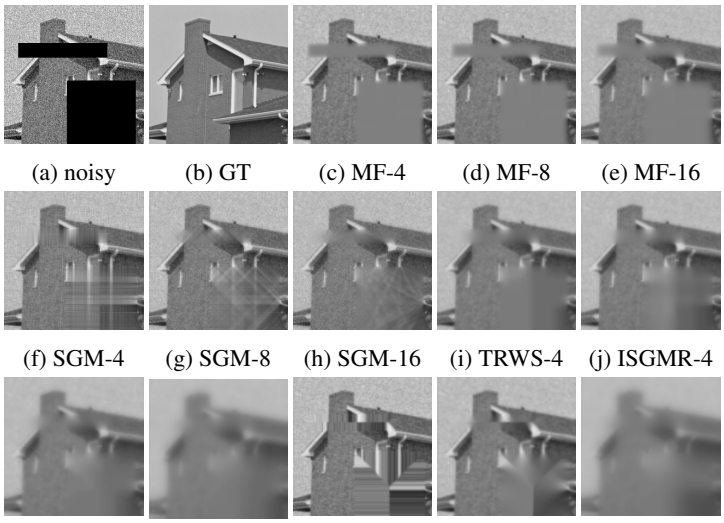
(a) noisy (b) GT (c) MF-4 (d) MF-8 (e) MF-16



(f) SGM-4 (g) SGM-8 (h) SGM-16 (i) TRWS-4 (j) ISGMR-4

(k) ISGMR-8 (l) ISGMR-16 (m) TRWP-4 (n) TRWP-8 (o) TRWP-16

Fig. 10: Visualization of MRF inferences for image denoising on “Penguin”.



(a) noisy (b) GT (c) MF-4 (d) MF-8 (e) MF-16

(f) SGM-4 (g) SGM-8 (h) SGM-16 (i) TRWS-4 (j) ISGMR-4

(k) ISGMR-8 (l) ISGMR-16 (m) TRWP-4 (n) TRWP-8 (o) TRWP-16

Fig. 11: Visualization of MRF inferences for image denoising on “House”.