A Comparison of TV-L1 Optical Flow Solvers on GPU

(Implementation Notes)*

Linchao Bao[1,2], Hailin Jin[1], Byungmoon Kim[1], Qingxiong Yang[2]
[1]Adobe Systems, Inc. [2]City University of Hong Kong
linchaobao@gmail.com

1 Classic TV-L1 Optical Flow

The classic TV-L1 optical flow is to minimize one of the following functionals:

$$E[u(x,y),v(x,y)] = \iint \psi((I_x u + I_y v + I_t)^2) + \lambda[\psi(u_x^2) + \psi(u_y^2) + \psi(v_x^2) + \psi(v_y^2)]dxdy \tag{1}$$

$$E[u(x,y),v(x,y)] = \iint \psi((I_x u + I_y v + I_t)^2) + \lambda[\psi(u_x^2 + u_y^2) + \psi(v_x^2 + v_y^2)]dxdy \tag{2}$$

$$E[u(x,y),v(x,y)] = \iint \psi((I_x u + I_y v + I_t)^2) + \lambda\psi(u_x^2 + u_y^2 + v_x^2 + v_y^2)dxdy \tag{3}$$

where u and v are unknown displacements along x and y direction (both are a function of x and y), respectively, and $\psi(s) = \sqrt{s + \epsilon^2}$ ($\epsilon = 0.001$). The regularization term in Eq. (1) is often called *anisotropic total variation* (employed in [11]), and that in Eq. (2) is often called *isotropic total variation* (employed in [12, 1]). Eq. (3) is used in [3, 4].

Some other variants of the TV-L1 formulation include incorporating *gradient constancy model* into data term [3], adding weights into data term and regularization term [13], incorporating occlusion detection [1], etc.

The solvers for TV-L1 optical flow include the graduated non-convexity solver [2, 11], fixed-point iteration solver [3], multigrid solver [5], duality-based solver [12], Fast Explicit Diffusion (FED) solver [9], split-Bregman solver [1], etc. In this study, we are interested in the fixed-point iteration solver, duality-based solver, and the split-Bregman solver because of their simplicity and effectiveness. We will use the following isotropic TV formulation (Eq. (2)) as example to derive the solution (for simplicity, we hereafter do not explicitly write u and v as the function of x and y):

$$E[u,v] = \iint \sqrt{(I_x u + I_y v + I_t)^2 + \epsilon^2} + \lambda\left(\sqrt{u_x^2 + u_y^2} + \sqrt{v_x^2 + v_y^2}\right)dxdy. \tag{4}$$

1.1 Classic Coarse-To-Fine Framework

The algorithm is described in Algorithm 1.

*This document is a supplementary material for the poster "A Comparison of TV-L1 Optical Flow Solvers on GPU" appeared in GPU Technology Conference (GTC) 2014 and for academic purpose only. Please cite the poster if you find this document helpful. Last update: March 24th, 2014.

Algorithm 1 Classic Coarse-To-Fine Optical Flow Estimation Framework

Input: number of image pyramid levels L, number of warpings W
Output: (u, v)
initialize $(u^{(0)}, v^{(0)})$ for level 1
for l from 1 to L **do**
  for w from 1 to W **do**
    (1) warp image using $(u^{(w-1)}, v^{(w-1)})$;
    (2) set (du, dv) to zero;
    (3) solve (du, dv) in Eq. (4) (see the following sections for solvers);
    (4) $(u^{(w)}, v^{(w)}) \leftarrow \big((u^{(w-1)}, v^{(w-1)}) + (du, dv)\big)$;
    (5) median filtering on $(u^{(w)}, v^{(w)})$;
  end for
  upscale flow to get the $(u^{(0)}, v^{(0)})$ for level $l + 1$;
end for

1.2 Fixed-Point Iteration Solver [3]

The Euler-Lagrange equation (two unknowns u and v, two variables x and y) of the functional in Eq. (4) is

$$\begin{cases} \frac{(I_x u+I_y v+I_t)I_x}{\sqrt{(I_x u+I_y v+I_t)^2+\epsilon^2}} - \lambda \mathrm{div}\left(\frac{\nabla u}{\sqrt{u_x^2+u_y^2}}\right) = 0, \\ \frac{(I_x u+I_y v+I_t)I_y}{\sqrt{(I_x u+I_y v+I_t)^2+\epsilon^2}} - \lambda \mathrm{div}\left(\frac{\nabla v}{\sqrt{v_x^2+v_y^2}}\right) = 0. \end{cases} \tag{5}$$

The idea of using *fixed-point iteration* [3] to remove the non-linearity of the above equations is that, for each iteration, the nonlinear terms is computed with u and v value from the last iteration. Denote

$$\begin{aligned} \Psi_d &= \frac{1}{\sqrt{(I_x u+I_y v+I_t)^2+\epsilon^2}}, \\ \Psi_u &= \frac{1}{\sqrt{u_x^2+u_y^2+\epsilon^2}}, \\ \Psi_v &= \frac{1}{\sqrt{v_x^2+v_y^2+\epsilon^2}}. \end{aligned} \tag{6}$$

Then, for the k-th iteration, we solve the following equations w.r.t. u and v,

$$\begin{cases} \Psi_d^{(k-1)} I_x^2 u + \Psi_d^{(k-1)} I_x I_y v + \Psi_d^{(k-1)} I_x I_t - \lambda \mathrm{div}\left(\Psi_u^{(k-1)} \nabla u\right) = 0, \\ \Psi_d^{(k-1)} I_x I_y u + \Psi_d^{(k-1)} I_y^2 v + \Psi_d^{(k-1)} I_y I_t - \lambda \mathrm{div}\left(\Psi_v^{(k-1)} \nabla v\right) = 0. \end{cases}$$

For simplicity, we hereafter ignore the superscript $(k-1)$. After discretization (the *gradient* operator uses forward difference, while the *divergence* operator uses backward difference), the above linear equations, for each pixel (i, j), can be written as

$$\begin{cases} \Psi_d I_x^2 u^{i,j} + \Psi_d I_x I_y v^{i,j} + \Psi_d I_x I_t - \lambda \left(\Psi_u^{i,j}(u^{i+1,j} - u^{i,j}) - \Psi_u^{i-1,j}(u^{i,j} - u^{i-1,j}) + \Psi_u^{i,j}(u^{i,j+1} - u^{i,j}) - \Psi_u^{i,j-1}(u^{i,j} - u^{i,j-1})\right) = 0, \\ \Psi_d I_x I_y u^{i,j} + \Psi_d I_y^2 v^{i,j} + \Psi_d I_y I_t - \lambda \left(\Psi_v^{i,j}(v^{i+1,j} - v^{i,j}) - \Psi_v^{i-1,j}(v^{i,j} - v^{i-1,j}) + \Psi_v^{i,j}(v^{i,j+1} - v^{i,j}) - \Psi_v^{i,j-1}(v^{i,j} - v^{i,j-1})\right) = 0. \end{cases}$$

Rearranging the terms, it becomes

$$\begin{cases} \left(\Psi_d I_x^2 + 2\lambda\Psi_u^{i,j} + \lambda\Psi_u^{i-1,j} + \lambda\Psi_u^{i,j-1}\right) u^{i,j} + \Psi_d I_x I_y v^{i,j} + \Psi_d I_x I_t - \lambda \left(\Psi_u^{i,j} u^{i+1,j} + \Psi_u^{i-1,j} u^{i-1,j} + \Psi_u^{i,j} u^{i,j+1} + \Psi_u^{i,j-1} u^{i,j-1}\right) = 0, \\ \Psi_d I_x I_y u^{i,j} + \left(\Psi_d I_y^2 + 2\lambda\Psi_v^{i,j} + \lambda\Psi_v^{i-1,j} + \lambda\Psi_v^{i,j-1}\right) v^{i,j} + \Psi_d I_y I_t - \lambda \left(\Psi_v^{i,j} v^{i+1,j} + \Psi_v^{i-1,j} v^{i-1,j} + \Psi_v^{i,j} v^{i,j+1} + \Psi_v^{i,j-1} v^{i,j-1}\right) = 0. \end{cases}$$

The idea of Jacobi iterations for solving the above linear system is that, when solving unknowns at (i,j), the unknowns at other pixels are treated as knowns (from the last iteration). Denote

$$\begin{aligned}
B_1 &= \lambda\left(\Psi_u^{i,j}u^{i+1,j} + \Psi_u^{i-1,j}u^{i-1,j} + \Psi_u^{i,j}u^{i,j+1} + \Psi_u^{i,j-1}u^{i,j-1}\right) - \Psi_d I_x I_t,\\
B_2 &= \lambda\left(\Psi_v^{i,j}v^{i+1,j} + \Psi_v^{i-1,j}v^{i-1,j} + \Psi_v^{i,j}v^{i,j+1} + \Psi_v^{i,j-1}v^{i,j-1}\right) - \Psi_d I_y I_t,\\
A_{11} &= \Psi_d I_x^2 + 2\lambda\Psi_u^{i,j} + \lambda\Psi_u^{i-1,j} + \lambda\Psi_u^{i,j-1},\\
A_{22} &= \Psi_d I_y^2 + 2\lambda\Psi_v^{i,j} + \lambda\Psi_v^{i-1,j} + \lambda\Psi_v^{i,j-1},\\
A_{12} &= \Psi_d I_x I_y.
\end{aligned}$$

The above linear system w.r.t. $(u^{i,j}, v^{i,j})$ becomes,

$$\begin{cases} A_{11}u^{i,j} + A_{12}v^{i,j} = B_1,\\ A_{12}u^{i,j} + A_{22}v^{i,j} = B_2. \end{cases}$$

The solution is,

$$\begin{cases} u^{i,j} = \frac{A_{22}B_1 - A_{12}B_2}{A_{11}A_{22} - A_{12}A_{12}},\\ v^{i,j} = \frac{A_{11}B_2 - A_{12}B_1}{A_{11}A_{22} - A_{12}A_{12}}. \end{cases} \tag{7}$$

This is the formula for computing $(u^{i,j}, v^{i,j})$ in each Jacobi iteration.

The fixed-point iteration solver is summarized in Algorithm 2. Note that in the above equations we assume I_x, I_y, I_t are computed after warping, thus the unknowns (u,v) is indeed the unknown flow incremental (du, dv) described in Algorithm 1.

Algorithm 2 Fixed-Point Iteration Solver (FP)

Parameters: number of optimizing iterations K, number of Jacobi iterations M
Output: (u,v) (when plugging the solver into Algorithm 1, it is for solving (du,dv))
initialize (u,v) to zero;
for k from 1 to K **do**
  compute nonlinear terms according to Eq. (6);
  for m from 1 to M **do**
    compute $(u^{i,j}, v^{i,j})$ according to Eq. (7);
  end for
end for

1.3 Split-Bregman Solver [1, 8]

The split-Bregman solver can directly solve L1 regularized problems. We can ignore the small constant ϵ in Eq. (4) and write out the original TV-L1 formulation as follows,

$$E[u,v] = \iint |I_x u + I_y v + I_t| + \lambda\left(\sqrt{u_x^2 + u_y^2} + \sqrt{v_x^2 + v_y^2}\right) dxdy. \tag{8}$$

The idea of split-Bregman algorithm is to first split the L1 data term and TV regularization terms using auxiliary variables (usually denoted as d variables), and then apply Bregman iterations by incorporating residuals during the optimization process (the residuals are usually denoted as b variables). Specifically, the split-Bregman algorithm converts the above equation into the following formulation,

$$\begin{aligned}
\iint |d_1| + \tfrac{\mu}{2}|d_1 - (I_x u + I_y v + I_t) - b_1|^2 &+ \lambda\left(\sqrt{d_{2ux}^2 + d_{2uy}^2} + \tfrac{\mu}{2}|d_{2ux} - u_x - b_{2ux}|^2 + \tfrac{\mu}{2}|d_{2uy} - u_y - b_{2uy}|^2\right)\\
&+ \lambda\left(\sqrt{d_{2vx}^2 + d_{2vy}^2} + \tfrac{\mu}{2}|d_{2vx} - v_x - b_{2vx}|^2 + \tfrac{\mu}{2}|d_{2vy} - v_y - b_{2vy}|^2\right) dxdy,
\end{aligned} \tag{9}$$

Then the problem can be solved by alternatively solving the following three subproblems:

1. solve (u, v);
2. solve d variables;
3. update b variables.

1.3.1 Solving (u, v)

When solving (u, v), the above problem is indeed a quadratic problem. Denote

$$\begin{aligned} C_1 &= d_1 - b_1, \\ C_{2ux} &= d_{2ux} - b_{2ux}, \\ C_{2uy} &= d_{2uy} - b_{2uy}, \\ C_{2vx} &= d_{2vx} - b_{2vx}, \\ C_{2vy} &= d_{2vy} - b_{2vy}, \end{aligned}$$

Eq. (9) becomes

$$\iint (I_x u + I_y v + I_t - C_1)^2 + \lambda \left((u_x - C_{2ux})^2 + (u_y - C_{2uy})^2 + (v_x - C_{2vx})^2 + (v_y - C_{2vy})^2\right) dxdy. \tag{10}$$

The Euler-Lagrange equation for minimizing the above functional is

$$\begin{cases} (I_x u + I_y v + I_t - C_1) I_x - \lambda \left(\mathrm{div}(\nabla u) - \frac{\mathrm{d}C_{2ux}}{\mathrm{d}x} - \frac{\mathrm{d}C_{2uy}}{\mathrm{d}y}\right) = 0, \\ (I_x u + I_y v + I_t - C_1) I_y - \lambda \left(\mathrm{div}(\nabla v) - \frac{\mathrm{d}C_{2vx}}{\mathrm{d}x} - \frac{\mathrm{d}C_{2vy}}{\mathrm{d}y}\right) = 0. \end{cases} \tag{11}$$

After discretization, the equation for each pixel (i, j) becomes (again, the *gradient* operator uses forward difference, while the *divergence* operator uses backward difference)

$$\begin{cases} I_x^2 u^{i,j} + I_x I_y v^{i,j} + I_x I_t - I_x C_1 - \lambda \left((u^{i+1,j} - u^{i,j}) - (u^{i,j} - u^{i-1,j}) + (u^{i,j+1} - u^{i,j}) - (u^{i,j} - u^{i,j-1}) - D_u\right) = 0, \\ I_x I_y u^{i,j} + I_y^2 v^{i,j} + I_y I_t - I_y C_1 - \lambda \left((v^{i+1,j} - v^{i,j}) - (v^{i,j} - v^{i-1,j}) + (v^{i,j+1} - v^{i,j}) - (v^{i,j} - v^{i,j-1}) - D_v\right) = 0, \end{cases}$$

where (note that here the finite difference for C_2 should be the same as the above divergence operator, i.e., backward difference)

$$\begin{aligned} D_u &= (C_{2ux}^{i,j} - C_{2ux}^{i-1,j}) + (C_{2uy}^{i,j} - C_{2uy}^{i,j-1}), \\ D_v &= (C_{2vx}^{i,j} - C_{2vx}^{i-1,j}) + (C_{2vy}^{i,j} - C_{2vy}^{i,j-1}). \end{aligned}$$

Rearranging the terms, it becomes

$$\begin{cases} (I_x^2 + 4\lambda) u^{i,j} + I_x I_y v^{i,j} = \lambda(u^{i+1,j} + u^{i-1,j} + u^{i,j+1} + u^{i,j-1} - D_u) - I_x I_t + I_x C_1, \\ I_x I_y u^{i,j} + (I_y^2 + 4\lambda) v^{i,j} = \lambda(v^{i+1,j} + v^{i-1,j} + v^{i,j+1} + v^{i,j-1} - D_v) - I_y I_t + I_y C_1. \end{cases}$$

Denote

$$\begin{aligned} B_1 &= \lambda \left(u^{i+1,j} + u^{i-1,j} + u^{i,j+1} + u^{i,j-1} - D_u\right) - I_x I_t + I_x C_1, \\ B_2 &= \lambda \left(v^{i+1,j} + v^{i-1,j} + v^{i,j+1} + v^{i,j-1} - D_v\right) - I_y I_t + I_y C_1, \\ A_{11} &= I_x^2 + 4\lambda, \\ A_{22} &= I_y^2 + 4\lambda, \\ A_{12} &= I_x I_y, \end{aligned}$$

The equation can be solved using Jacobi iteration (Eq. (7)).

1.3.2 Solving d variables

Regarding d variables, Eq. (9) can be breaks into three problems w.r.t. d_1, (d_{2ux}, d_{2uy}), (d_{2vx}, d_{2vy}), respectively:

$$\iint |d_1| + \frac{\mu}{2}(d_1 - (I_x u + I_y v + I_t) - b_1)^2 dxdy, \tag{12}$$

$$\iint \sqrt{d_{2ux}^2 + d_{2uy}^2} + \frac{\mu}{2}(d_{2ux} - u_x - b_{2ux})^2 + \frac{\mu}{2}(d_{2uy} - u_y - b_{2uy})^2 dxdy, \tag{13}$$

$$\iint \sqrt{d_{2vx}^2 + d_{2vy}^2} + \frac{\mu}{2}(d_{2vx} - v_x - b_{2vx})^2 + \frac{\mu}{2}(d_{2vy} - v_y - b_{2vy})^2 dxdy. \tag{14}$$

Note that if the objective function is anisotropic TV (Eq. (1)), Eqs. (13) and (14) can be further decoupled into two subproblems, which is much simplier.

As suggested in [8], the minimizer of Eq. (12) can be expressed using the *shrinkage* formula

$$d_1 = \max\left(|s| - \frac{1}{\mu}, 0\right) \cdot \frac{s}{|s|}, \tag{15}$$

where

$$s = I_x u + I_y v + I_t + b_1. \tag{16}$$

The minimizer of Eq. (13) can be expressed using the *generalized shrinkage* formula

$$d_{2ux} = \max\left(s - \frac{1}{\mu}, 0\right) \cdot \frac{u_x + b_{2ux}}{s}, \tag{17}$$

$$d_{2uy} = \max\left(s - \frac{1}{\mu}, 0\right) \cdot \frac{u_y + b_{2uy}}{s}, \tag{18}$$

where

$$s = \sqrt{(u_x + b_{2ux})^2 + (u_y + b_{2uy})^2}. \tag{19}$$

Eq. (14) can be minimized in the same way.

1.3.3 Updating b variables

Updating b variables is relatively simpler, for the k-th iteration,

$$\begin{array}{rcl} b_1^{(k)} & = & b_1^{(k-1)} + (I_x u + I_y v + I_t) - d_1, \\ b_{2ux}^{(k)} & = & b_{2ux}^{(k-1)} + u_x - d_{2ux}, \\ b_{2uy}^{(k)} & = & b_{2uy}^{(k-1)} + u_y - d_{2uy}, \\ b_{2vx}^{(k)} & = & b_{2vx}^{(k-1)} + v_x - d_{2vx}, \\ b_{2vy}^{(k)} & = & b_{2vy}^{(k-1)} + v_y - d_{2vy}. \end{array} \tag{20}$$

1.3.4 Algorithm

The split-Bregman solver is summarized in Algorithm 3 (the number of Bregman iteration is fixed to 1, as suggested in [8]).

1.4 Duality-based Solver [12, 7]

We also derive the solution of the duality-based solver from the original TV-L1 formulation in Eq. (8). The idea of the duality-based solver [12] is to first decouple the L1 data term and the TV regularization term by introducing auxiliary variables (u', v'), which is

$$\iint \frac{1}{\lambda}|I_x u + I_y v + I_t| + \frac{\theta}{2}\left[(u - u')^2 + (v - v')^2\right] + \left(\sqrt{u'^2_x + u'^2_y} + \sqrt{v'^2_x + v'^2_y}\right) dxdy, \tag{21}$$

Algorithm 3 Split-Bregman Solver (SB)

Parameters: number of optimizing iterations K, number of Jacobi iterations M
Output: (u, v) (when plugging the solver into Algorithm 1, it is for solving (du, dv))
initialize (u, v) to zero;
for k from 1 to K **do**
  for m from 1 to M **do**
    compute $(u^{i,j}, v^{i,j})$ according to Eq. (7) (see Sec. 1.3.1);
  end for
  update d variables according to Eqs. (15) - (19);
  update b variables according to Eq. (20);
end for

The decoupled subproblems are

$$\iint \frac{1}{\lambda}|I_x u + I_y v + I_t| + \frac{\theta}{2}\left[(u-u')^2 + (v-v')^2\right] dxdy, \tag{22}$$

$$\iint \frac{\theta}{2}\left[(u-u')^2 + (v-v')^2\right] + \left(\sqrt{u'^2_x + u'^2_y} + \sqrt{v'^2_x + v'^2_y}\right) dxdy, \tag{23}$$

Then the minimization can be performed by alternating the following two steps:

1. for fixed (u', v') value, minimize Eq. (22) to solve (u, v), which is a per-pixel optimization;
2. for fixed (u, v) value, minimize Eq. (23) to solve (u', v'), which is the classic ROF model [10].

1.4.1 Per-pixel optimization (solving u, v)

The original algorithm in [12] uses a thresholding step to solve the per-pixel minimization for Eq. (22), which is

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u' \\ v' \end{bmatrix} + \begin{bmatrix} I_x \\ I_y \end{bmatrix} \cdot \begin{cases} \frac{1}{\lambda\theta}, & \text{if } I_x u' + I_y v' + I_t < -\frac{1}{\lambda\theta}(I_x^2 + I_y^2) \\ -\frac{1}{\lambda\theta}, & \text{if } I_x u' + I_y v' + I_t > \frac{1}{\lambda\theta}(I_x^2 + I_y^2) \\ -\frac{I_x u' + I_y v' + I_t}{I_x^2 + I_y^2}, & \text{otherwise.} \end{cases} \tag{24}$$

Note that this formula only works when the data term has only one channel. For RGB color images, the data term $|I_x u + I_y v + I_t|$ in Eq. (22) is actually three terms. More than one absolute terms will make the problem difficult. Such thresholding step no longer works well.

Instead, we can use split-Bregman to minimize Eq. (22) for multiple data terms. By introducing d and b variables for each data term, the functional of Eq. (22) becomes

$$\iint |d_1| + \frac{\mu}{2}|d_1 - (I_x u + I_y v + I_t) - b_1|^2 + \frac{\lambda\theta}{2}\left[(u-u')^2 + (v-v')^2\right] dxdy. \tag{25}$$

Then the minimizer w.r.t. (u, v) can be solved in closed-form, i.e., Eq. (7), where

$$\begin{aligned} B_1 &= \lambda\theta u' + \mu I_x(d_1 - b_1 - I_t), \\ B_2 &= \lambda\theta v' + \mu I_y(d_1 - b_1 - I_t), \\ A_{11} &= \lambda\theta + \mu I_x^2, \\ A_{22} &= \lambda\theta + \mu I_y^2, \\ A_{12} &= \mu I_x I_y. \end{aligned}$$

Then d_1 and b_1 can be updated using the same formula in Eqs. (15) and (20).

1.4.2 Solving ROF model (w.r.t. u', v')

Minimizing Eq. (23) is a classic problem [10]. The flow solver in [12] employs an efficient duality-based algorithm from Chambolle [7]. To solve the (u', v') in Eq. (23), the algorithm introduces $\mathbf{p}$ dual variables and iteratively compute the following two steps (take computing u as an example)

$$u'^{(m)} = u + \frac{1}{\theta}\text{div}\mathbf{p}^{(m-1)}, \tag{26}$$

$$\mathbf{p}^{(m)} = \begin{bmatrix} p_1^{(m)} \\ p_2^{(m)} \end{bmatrix} = \frac{\begin{bmatrix} p_1^{(m-1)} \\ p_2^{(m-1)} \end{bmatrix} + \tau\theta \begin{bmatrix} u_x'^{(m)} \\ u_y'^{(m)} \end{bmatrix}}{1 + \tau\theta\sqrt{(u'^{(m)})_x^2 + (u'^{(m)})_y^2}} = \frac{\mathbf{p}^{(m-1)} + \tau\theta\nabla u'^{(m)}}{1 + \tau\theta||\nabla u'^{(m)}||_{\ell 2}}, \tag{27}$$

where τ is a step size (should be smaller than 0.25). In the same way, v can be computed. Note that Eq. (26) uses (u, v) value from the per-pixel optimization step, rather than the value from the $(m-1)$-th iteration in this step. Another place requiring attention is that $\mathbf{p}$ should be passed into the next warping, rather than discarded in each warping.

1.4.3 Algorithm

The duality-based algorithm in [12] is described in Algorithm 4. The modified algorithm is described in Algorithm 5.

Algorithm 4 Duality-based Solver [12] (DL)

Parameters: number of optimizing iterations K, number of Chambolle iterations M
Output: (u, v) (NOTE that this solver directly solves (u, v), but not (du, dv)!)
initialize (u', v') to zero;
for k from 1 to K **do**
  update (u, v) according to Eq. (24);
  for m from 1 to M **do**
    update (u', v') according to Eq. (26);
    update dual variable $\mathbf{p}$ according to Eq. (27);
  end for
end for

1.5 Weighted TV

In order to not oversmooth flow across edges, the total variation regularization can be weighted by image edges. Specifically, the weighted TV terms is $\sqrt{(w_x u_x)^2 + (w_y u_y)^2} + \sqrt{(w_x v_x)^2 + (w_y v_y)^2}$, where w_x and w_y is computed by the amplitude of image gradient along x and y direction, respectively, by

$$w_x = \nu + (1-\nu)\exp(-\beta||I_x||_{\ell 2}^2), \tag{28}$$

$$w_y = \nu + (1-\nu)\exp(-\beta||I_y||_{\ell 2}^2). \tag{29}$$

The parameters are usually set to $\beta = 0.001$ and $\nu = 0.01$. The weighted TV can only be applied to FP and SB solver.

2 Comparison

We studied the best parameters for each solver and compared them on Middlebury's training dataset. Note that the FP and SB solvers employ a weighted TV regularizer, while DL solvers do not. See Table 1 for the parameters and Table 2 for the comparison.

Algorithm 5 Split-Bregman-Chambolle Solver (DL2)

Parameters: number of optimizing iterations K, number of Split-Bregman iterations N, number of Chambolle iterations M
Output: (u, v) (NOTE that this solver directly solves (u, v), but not (du, dv)!)
initialize (u', v') to zero;
for k from 1 to K **do**
  for n from 1 to N **do**
    solve w.r.t. (u, v) in Eq. (25);
    update auxiliary variables d and b;
  end for
  for m from 1 to M **do**
    update (u', v') according to Eq. (26);
    update dual variable $\mathbf{p}$ according to Eq. (27);
  end for
end for

Table 1: Suggested parameters

	FP	SB	DL/DL2
inner	10	2	1
outer	20	20	10
parameter	–	$\mu = 2.0$	$\theta = 6.0$

Table 2: Average EPE on Middlebury training dataset (timing is for 640 × 480 image)

solver	EPE	Timing † (sec)
SB	0.27	0.31
FP	0.30	0.54
DL	0.30	0.13
DL2	0.29	0.14

†Reported on an NVIDIA Geforce GTX 780 Ti GPU .

3 Occlusion Handling

According to [1], adding an additional occlusion term e into the cost function in Eq. (8), the cost function becomes

$$\iint |I_x u + I_y v + I_t - e| + \alpha \|e\|_{\ell_0} + \lambda \left(\sqrt{u_x^2 + u_y^2} + \sqrt{v_x^2 + v_y^2} \right) dxdy. \tag{30}$$

Relaxing the ℓ_0 norm to weighted ℓ_1 norm, the cost function becomes

$$\iint |I_x u + I_y v + I_t - e| + \alpha |w_e e| + \lambda \left(\sqrt{u_x^2 + u_y^2} + \sqrt{v_x^2 + v_y^2} \right) dxdy. \tag{31}$$

where w is initialized as 1.0 and updated by $w_e = \frac{1}{|e|+0.001}$ in each iteration (see [6]). After adding d and b variables, the objective function becomes

$$\begin{aligned} \iint \quad & |d_1| + \tfrac{\mu}{2} |d_1 - (I_x u + I_y v + I_t - e) - b_1|^2 + \alpha |w_e e| \\ + \quad & \lambda \left(\sqrt{d_{2ux}^2 + d_{2uy}^2} + \tfrac{\mu}{2} |d_{2ux} - u_x - b_{2ux}|^2 + \tfrac{\mu}{2} |d_{2uy} - u_y - b_{2uy}|^2 \right) \\ + \quad & \lambda \left(\sqrt{d_{2vx}^2 + d_{2vy}^2} + \tfrac{\mu}{2} |d_{2vx} - v_x - b_{2vx}|^2 + \tfrac{\mu}{2} |d_{2vy} - v_y - b_{2vy}|^2 \right) dxdy. \end{aligned} \tag{32}$$

Thus the subproblem w.r.t. e is

$$\iint \frac{\mu}{2}|d_1 - (I_x u + I_y v + I_t - e) - b_1|^2 + \alpha|w_e e| \; dxdy. \tag{33}$$

Then the algorithm described in Sec. 1.3 needs one more step in each iteration: update e variable using the *shrinkage* formula

$$e = \max\left(|s| - \frac{\alpha w_e}{\mu}, 0\right) \cdot \frac{s}{|s|}, \tag{34}$$

where

$$s = I_x u + I_y v + I_t + b_1 - d_1. \tag{35}$$

Note that the value of d and b for updating e should be from the previous iteration.

References

[1] Alper Ayvaci, Michalis Raptis, and Stefano Soatto. Sparse occlusion detection with optical flow. *IJCV*, 2012.

[2] Michael J Black and Paul Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 1996.

[3] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV*, 2004.

[4] Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *TPAMI*, 2011.

[5] Andrés Bruhn, Joachim Weickert, Timo Kohlberger, and Christoph Schnörr. A multigrid platform for real-time motion computation with discontinuity-preserving variational methods. *IJCV*.

[6] Emmanuel J Candes, Michael B Wakin, and Stephen P Boyd. Enhancing sparsity by reweighted 1 minimization. *Journal of Fourier Analysis and Applications*, 14(5-6):877–905, 2008.

[7] Antonin Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision*, 20(1-2):89–97, 2004.

[8] Tom Goldstein and Stanley Osher. The split bregman method for l1-regularized problems. *SIAM Journal on Imaging Sciences*, 2(2):323–343, 2009.

[9] Pascal Gwosdek, Henning Zimmer, Sven Grewenig, Andrés Bruhn, and Joachim Weickert. A highly efficient gpu implementation for variational optic flow based on the euler-lagrange framework. In *Trends and Topics in Computer Vision*. 2012.

[10] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[11] Deqing Sun, Stefan Roth, and Michael J Black. Secrets of optical flow estimation and their principles. In *CVPR*, 2010.

[12] A. Wedel, T. Pock, C. Zach, H. Cremers, and D. Bischof. An improved algorithm for TV-L1 optical flow. In *Proc. Dagstuhl Motion Workshop*, 2008.

[13] Henning Zimmer, Andrés Bruhn, and Joachim Weickert. Optic flow in harmony. *IJCV*, 2011.