Stereo Depth Continuity

Steven Diamond (stevend2@stanford.edu), Jessica Taylor (jacobt@stanford.edu)

March 17, 2014

---

1 Abstract

We tackle the problem of producing depth maps from stereo video. Some algorithms for doing this using correspondences produce noisy output. Chan et al. (2011) [2] use convex optimization techniques to reduce noise and also smooth out the depth maps over time. We start with the noisy output of OpenCV and apply this technique to smooth it, developing some refinements along the way. Specifically, we ignore pixels with the highest possible depth value (as this indicates that no correspondence was found for the pixel), and we make the smoothing less aggressive if the pixels differ by color. These refinements improve the smoothing to yield more accurate depth estimates. We test the algorithm on multiple videos and find that it generally performs well as long as OpenCV's output was somewhat reasonable.

2 Review of previous work

Given stereo images, it is possible to retrieve depth maps by finding correspondences between the left and right images. This technique is used by OpenCV (http://docs.opencv.org/trunk/doc/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html). We ran OpenCV's StereoBM algorithm with a SADWindowSize parameter of 5, which increases accuracy by matching smaller regions but increases noise. It is straightforward to find depth maps for stereo video by simply finding maps for each frame independently. However, in addition to inheriting all the problems of the original algorithm, the result is usually choppy, lacking smooth changes over time. Chan et al. (2011) [2] apply a smoothing algorithm to stitch together frame-by-frame depth maps, which succeeds in both reducing the noise of each depth map and increasing continuity over time. They solve the following optimization problem:

$$\text{minimize } \mu \sum_{i,j,k} |f_{i,j,k} - g_{i,j,k}| + \sum_{i,j,k} \left\| \begin{array}{c} \beta_x(f_{i+1,j,k} - f_{i,j,k}) \\ \beta_y(f_{i,j+1,k} - f_{i,j,k}) \\ \beta_t(f_{i,j,k+1} - f_{i,j,k}) \end{array} \right\|_2$$

where g is a 3D matrix containing depth estimates from a frame-by-frame algorithm, f is a 3D matrix of smoothed depth estimates, and μ, β_x, β_y, and β_t are non-negative parameters.

3 Improvements on Previous Work

We found that OpenCV produced very noisy output. Additionally, many pixels (in some cases the majority) were labeled as the furthest possible depth value. Usually, these pixels did not actually have such high depths; instead, they were scattered around the image at many different depth values. It is likely that OpenCV found no correspondences for these pixels, and gave them the highest depth value as a default. Therefore, we adjusted the smoothing algorithm to ignore these pixels' depth values. This way, the algorithm is free to vary the smoothed depth estimate corresponding to these pixels in order to produce more continuity.

Additionally, we changed the smoothing to place less emphasis on smoothness for pixels with similar colors. This captures the intuition that depth changes should happen where color changes happen. To do this, we created a weight for each pixel proportional to $\frac{1}{1+\|\Delta\|_2}$, where Δ is a vector combining color differences in the x, y, and time directions.

As explained later, Chan et al. [2] include smoothness constraints between pixels at opposite ends of the image, or between the final and first frame, just as there are smoothness constraints between neighboring pixels or frames. Including these constraints is necessary to make optimization much more efficient. We do not drop these constraints, but we reduce the weight of pixels on edges that correspond to these wrapped constraints (to 1/3 of the original weight) to make these constraints less cumbersome.

4 Summary of Technical Solution

To obtain a smoothed depth map for the video, we solve a convex optimization problem. We regard the video as a time space volume, where each pixel has an x,y, and t coordinate. The x axis corresponds to the pixel column, the y axis to the pixel row, and the t axis to the pixel frame.

Let g be a 3D matrix of pixel depth estimates indexed by x,y, and t. g is obtained by estimating a depth map for each frame of the video independently. Some pixels may not be assigned a depth by the frame-by-frame algorithm. We call such pixels *unlabeled*. Let E be the set of labeled pixels in g.

Let f be a matrix variable representing the smoothed pixel depth estimates. Then our optimization problem is given by

$$\text{minimize } \mu \sum_{(i,j,k)\in E} |f_{i,j,k} - g_{i,j,k}| + \sum_{i,j,k} w_{i,j,k} \left\| \begin{array}{c} \beta_x(f_{i+1,j,k} - f_{i,j,k}) \\ \beta_y(f_{i,j+1,k} - f_{i,j,k}) \\ \beta_t(f_{i,j,k+1} - f_{i,j,k}) \end{array} \right\|_2$$

w, μ, β_x, β_y, and β_t are non-negative parameters.

The optimization problem finds an f that is close to the original depth estimates g but has fewer sharp changes in depth. Specifically, the term $\sum_{(i,j,k)\in E} |f_{i,j,k} - g_{i,j,k}|$ penalizes f's deviation from labeled pixels in g while the term

$$\sum_{i,j,k} w_{i,j,k} \left\| \begin{array}{c} \beta_x(f_{i+1,j,k} - f_{i,j,k}) \\ \beta_y(f_{i,j+1,k} - f_{i,j,k}) \\ \beta_t(f_{i,j,k+1} - f_{i,j,k}) \end{array} \right\|_2$$

penalizes large depth gradients in f. The balance between these terms is controlled by the parameter μ.

The parameter w is a matrix with a non-negative weight for each pixel. The weight $w_{i,j,k}$ determines how heavily the gradient at $f_{i,j,k}$ is penalized. To choose w we followed the principle that neighboring pixels with similar colors or intensities are likely to belong to the same surface and thus be at similar depths. Conversely, neighboring pixels with very different colors or intensities likely belong to different objects and thus could have very different depths.

β_x, β_y, and β_t are global weights applied to the component of the gradient in x,y, and t dimensions, respectively. The β parameters allow changes in depth in certain dimensions to be weighted more (or less) heavily. Empirically, setting β_t less than β_x and β_y improves performance. This makes sense given that moving a single frame typically makes more of a difference to depth than moving a single pixel.

5 Details of Technical Solution

Our optimization problem is extremely high dimensional, since a video contains millions of pixels and we estimate a depth for every pixel. There is no generic solver for convex optimization problems capable of solving our problem in a reasonable amount of time. Hence, we implemented a custom solver for our problem based on the work by Chan et al. (2011) [2].

Like Chan et al., our solver uses an iterative algorithm where each iteration runs in $O(n \log n)$. Here n is the total number of pixels in the video. This runtime is an important distinction of our algorithm, since a generic convex optimization algorithm would run in $O(n^2)$ or $O(n^3)$ [2]. We obtain an $O(n \log n)$ runtime per iteration by exploiting special structure in our problem.

5.1 The ADMM Algorithm

Our algorithm uses the Alternating Direction Method of Multipliers (ADMM). ADMM is an iterative method designed for convex optimization problems too large for standard interior point methods [1].

ADMM solves problems of the form

$$\begin{aligned} &\text{minimize } f(x) + g(z) \\ &\text{subject to } Ax + Bz = c \end{aligned}$$

where f, g are convex functions. ADMM operates on the so-called augmented Lagrangian

$$L_\rho(x, z, y) = f(x) + g(x) + y^T(Ax + Bz - c) + (\rho/2) \left\|Ax + Bz - c\right\|_2^2$$

where y is the dual variable. ADMM consists of the following algorithm:
Data: A,B,c
Result: Primal optimal x, z and dual optimal y
Initialize $x^{(0)}, z^{(0)}, y^{(0)}$;
while *The stopping conditions are not met* **do**
$x^{(k+1)} = \arg\min_x L_\rho(x, z^{(k)}, y^{(k)})$;
$z^{(k+1)} = \arg\min_z L_\rho(x^{(k+1)}, z, y^{(k)})$;
$y^{(k+1)} = y^{(k)} + \rho(Ax^{(k+1)} + Bz^{(k+1)} - c)$;
end

The initialization depends on the problem. ADMM will converge faster if it is initialized near the solution.

The stopping condition is based on the L2 norms of the primal residual

$$r^{(k+1)} = Ax^{(k+1)} + Bz^{(k+1)} - c$$

and dual residual

$$s^{(k+1)} = \rho A^T B(z^{(k+1)} - z^{(k)})$$

When $\|r^{(k+1)}\|_2$ and $\|s^{(k+1)}\|_2$ are both small it indicates that ADMM is near the correct solution.

Boyd et al. [1] recommend using both a relative and absolute metric for measuring convergence. Specifically, they suggest terminating when

$$\|r^{(k+1)}\|_2 \leq \epsilon^{\text{pri}} \text{ and } \|s^{(k+1)}\|_2 \leq \epsilon^{\text{dual}}$$

where

$$\begin{aligned}\epsilon^{\text{pri}} &= \sqrt{p}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\left\{\|Ax^{(k)}\|_2, \|Bz^{(k)}\|_2, \|c\|_2\right\} \\ \epsilon^{\text{dual}} &= \sqrt{n}\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\|A^T y^{(k)}\|_2\end{aligned}$$

p is the number of rows in A and n is the number of columns. ϵ^{abs} and ϵ^{rel} are chosen based on the desired accuracy and the problem scale.

5.2 Applying ADMM

To use ADMM we must reformulate our problem to match the ADMM standard form. Let $f \in \mathcal{R}^{C\times R\times F}$. Denote $N = CRF$. We define new variables $r \in \mathcal{R}^{C\times R\times F}, u^{(x)}, u^{(y)}, u^{(t)} \in \mathcal{R}^N$.

Let $\mathbf{vec}(f)$ denote the vectorized version of f. Specifically, to vectorize f we flatten each frame in row-major order and concatenate the frame vectors. We now define matrices $D_x, D_y, D_t \in \mathcal{R}^{N\times N}$. D_x, D_y, D_t are forward shift operators in the x, y, and t dimensions respectively. $[D_x\mathbf{vec}(f)]_i$ is the difference between the depth at the pixel one space forward in x from pixel i and the depth at pixel i. $[D_y\mathbf{vec}(f)]_i$ and $[D_t\mathbf{vec}(f)]_i$ are defined similarly. Formally, if pixel i occurs at location x, y, t:

$$\begin{aligned}[D_x\mathbf{vec}(f)]_i &= f(x+1 \bmod C, y, t) - f(x, y, z) \\ [D_y\mathbf{vec}(f)]_i &= f(x, y+1 \bmod R, t) - f(x, y, z) \\ [D_t\mathbf{vec}(f)]_i &= f(x, y, t+1 \bmod F) - f(x, y, z)\end{aligned}$$

We use periodic boundary conditions, meaning that for pixels on the boundaries of f we look at the depth difference between that pixel and the next pixel in the flattened version of f. For example, if $x = C - 1$ we look at the difference

$$f(x + 1 \bmod C, y, t) - f(x, y, z) = f(0, y + 1, t) - f(x, y, z)$$

These boundary conditions are not natural, but they are necessary for the matrices D_x, D_y, D_t to be block-circulant. A matrix is defined as block-circulant if every kth column of the matrix is the first column rotated $k - 1$ steps forward. Visually the matrix has the form below

$$\begin{bmatrix} c_0 & c_{n-1} & \dots & c_1 \\ c_1 & c_0 & \dots & c_2 \\ c_2 & c_1 & \dots & c_3 \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1} & c_{n-2} & \dots & c_0 \end{bmatrix}$$

Let c be the first column of a block circulant matrix C. Multiplying a vector x by C is equivalent to a circular convolution of c with x [3]:

$$Cx = c \star x$$

In the Fourier domain, circular convolution is equivalent to elementwise multiplication. Thus applying the discrete fourier transform operator $\mathcal{F}$,

$$\mathcal{F}(c \star x) = \mathcal{F}(c) \cdot \mathcal{F}(x)$$

Remark 5.1. We can thus use the DFT to solve linear equations $Cx = b$ where C is a block-circulant matrix in time $O(n \log n)$. Observe,

$$\begin{aligned} \mathcal{F}(Cx) &= \mathcal{F}(b) \\ \mathcal{F}(c) \cdot \mathcal{F}(x) &= \mathcal{F}(b) \\ x &= \mathcal{F}^{-1}\left(\frac{\mathcal{F}(b)}{\mathcal{F}(c)}\right) \end{aligned}$$

Returning to our formulation of the optimization problem, we make two more definitions to simplify notation

$$u = \begin{bmatrix} u^{(x)} \\ u^{(y)} \\ u^{(t)} \end{bmatrix}, D = \begin{bmatrix} D_x \\ D_y \\ D_t \end{bmatrix}$$

We can now express our optimization problem in ADMM form:

$$\begin{aligned} \text{minimize } & \mu \sum_{(i,j,k) \in E} |r_{i,j,k}| + \sum_i w_i \left\| \begin{bmatrix} u_i^{(x)} \\ u_i^{(y)} \\ u_i^{(t)} \end{bmatrix} \right\|_2 \\ \text{subject to } & r = f - g \\ & u = D\mathbf{vec}(f) \end{aligned}$$

The augmented Lagrangian is

$$L_\rho(f,u,r,y,z) = \mu \sum_{(i,j,k)\in E} |r_{i,j,k}| + \sum_i w_i \left\| \begin{bmatrix} u_i^{(x)} \\ u_i^{(y)} \\ u_i^{(t)} \end{bmatrix} \right\| - z^T \mathbf{vec}(r - f + g) + (\rho/2) \left\| \mathbf{vec}(r - f + g) \right\|_2^2 - y^T(u - D\mathbf{vec}(f)) + (\rho/2) \left\| u - D\mathbf{vec}(f) \right\|_2^2$$

where $y = \begin{bmatrix} y^{(x)} \\ y^{(y)} \\ y^{(t)} \end{bmatrix} \in \mathcal{R}^{3N}, z \in \mathcal{R}^N$ are dual variables.

5.3 Computing ADMM Iterations

We now explain how we compute each iteration of ADMM. The structure of our problem allows us to compute each step of the iteration in at most time $O(n \log n)$ where n is the number of pixels in the video. Without taking advantage of structure, an iteration would take time $O(n^2)$, which is far less feasible given that we have millions of pixels.

For our problem the ADMM algorithm takes the following form:

Data: g

Result: Primal optimal f, u, r and dual optimal y, z

Initialize $f^{(0)} = g, u^{(0)} = D\mathbf{vec}(g), r^{(0)} = z^{(0)} = y^{(0)} = 0$;

while *The stopping conditions are not met* **do**

$u^{(k+1)} = \arg\min_u L_\rho(f^{(k)}, u, r^{(k)}, y^{(k)}, z^{(k)})$;

$r^{(k+1)} = \arg\min_r L_\rho(f^{(k)}, u^{(k+1)}, r, y^{(k)}, z^{(k)})$;

$f^{(k+1)} = \arg\min_f L_\rho(f, u^{(k+1)}, r^{(k+1)}, y^{(k)}, z^{(k)})$;

$y^{(k+1)} = y^{(k)} - \rho(u^{(k+1)} - D\mathbf{vec}(f^{(k+1)}))$;

$z^{(k+1)} = z^{(k)} - \rho\mathbf{vec}(r^{(k+1)} - f^{(k+1)} + g)$;

end

Initializing $f^{(0)} = g, u^{(0)} = D\mathbf{vec}(f^{(0)}), r^{(0)} = f^{(0)} - g = 0$ is useful because starting ADMM near its eventual solution decreases the number of iterations needed. There is no obvious starting estimate for the dual variables, so we initialize them to 0.

We next explain how the u,r, and f updates are computed.

5.3.1 u update

Recall the u update is the solution to the problem

$$u = \arg\min_u \sum_i w_i \left\| \begin{bmatrix} u_i^{(x)} \\ u_i^{(y)} \\ u_i^{(t)} \end{bmatrix} \right\|_2 - y^T(u - D\mathbf{vec}(f)) + (\rho/2) \left\| u - D\mathbf{vec}(f) \right\|_2^2$$

The u update can be expressed in closed form using the shrinkage formula defined by Li [4]. We define

$$v^{(x)} = \beta_x D_x \mathbf{vec}(f) + \frac{1}{\rho} y^{(x)}$$

with $v^{(y)}, v^{(t)}$ defined analogously. We further define $v \in \mathcal{R}^N$ such that

$$v_i = \sqrt{(v_i^{(x)})^2 + (v_i^{(y)})^2 + (v_i^{(t)})^2}$$

for all i.

Then for each pixel i with gradient weight w_i the updated value of $u_i^{(x)}$ is given by

$$u_i^{(x)} = \max\left\{v_i - \frac{w_i}{\rho}, 0\right\} \cdot \frac{v_i^{(x)}}{v_i}$$

The updates for $u_i^{(y)}, u_i^{(t)}$ are analogous.

5.3.2 r update

Recall the r update is the solution to the problem

$$r = \arg\min_r \mu \sum_{(i,j,k)\in E} |r_{i,j,k}| - z^T \mathbf{vec}(r - f + g) + (\rho/2) \left\|\mathbf{vec}(r - f + g)\right\|_2^2$$

For a pixel $(i, j, k) \notin E$, the value $r_{i,j,k}$ does not change. For a pixel $(i, j, k) \in E$, a similar shrinkage formula gives a closed form expression for the r update. We use ℓ to indicate the index of (i, j, k) in the vectorized image.

$$r_{i,j,k} = \max\left\{\left|f_{i,j,k} - g_{i,j,k} + \frac{1}{\rho} z_\ell\right| - \frac{\mu}{\rho}, 0\right\} \cdot \text{sign}\left(f_{i,j,k} - g_{i,j,k} + \frac{1}{\rho} z_\ell\right)$$

5.3.3 f update

Recall the f update is the solution to the problem

$$f = \arg\min_f (\rho/2) \left\|\mathbf{vec}(r - f + g)\right\|_2^2 + (\rho/2) \left\|u - D\mathbf{vec}(f)\right\|_2^2 + y^T D\mathbf{vec}(f) + z^T \mathbf{vec}(f)$$

The optimal f satisfies the normal equations

$$\begin{aligned} \rho(I + D^T D)\mathbf{vec}(f) &= \rho g + (\rho r - z) + D^T(\rho u - y) \\ \rho(I + D_x^T D_x + D_y^T D_y + D_t^T D_t)\mathbf{vec}(f) &= \rho g + (\rho r - z) + D^T(\rho u - y) \end{aligned}$$

Recall that D_x, D_y, D_t are block-circulant matrices. This means D_x^T, D_y^T, D_t^T are also block circulant. The identity matrix I is block circulant as well. Thus by Remark 5.1 we can solve the normal equations in time $O(n \log n)$ using the DFT operator $\mathcal{F}$ and the FFT algorithm. Denote $M = \rho(I + D_x^T D_x + D_y^T D_y + D_t^T D_t)$. Then applying $\mathcal{F}$ to $M\mathbf{vec}(f)$ yields

$$\begin{aligned} \mathcal{F}(M\mathbf{vec}(f)) &= \rho(\mathcal{F}(I) + \mathcal{F}(D_x^T) \cdot \mathcal{F}(D_x) + \mathcal{F}(D_y^T) \cdot \mathcal{F}(D_y) + \mathcal{F}(D_t^T) \cdot \mathcal{F}(D_t)) \cdot \mathcal{F}(\mathbf{vec}(f)) \\ &= \mathcal{F}(M) \cdot \mathcal{F}(\mathbf{vec}(f)) \end{aligned}$$

Here $\mathcal{F}(I)$, $\mathcal{F}(D_x^T)$, $\mathcal{F}(D_x)$, etc., mean the DFT operator applied to the first column of each matrix.

$\mathcal{F}(M)$ need only be computed once, before the ADMM loop begins. We use $\mathcal{F}(M)$ to solve the normal equations

$$\begin{aligned} M\mathbf{vec}(f) &= \rho g + (\rho r - z) + D^T(\rho u - y) \\ \mathcal{F}(M\mathbf{vec}(f)) &= \mathcal{F}(\rho g + (\rho r - z) + D^T(\rho u - y)) \\ \mathcal{F}(M) \cdot \mathcal{F}(\mathbf{vec}(f)) &= \mathcal{F}(\rho g + (\rho r - z) + D^T(\rho u - y)) \\ \mathbf{vec}(f) &= \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\rho g + (\rho r - z) + D^T(\rho u - y))}{\mathcal{F}(M)}\right) \end{aligned}$$

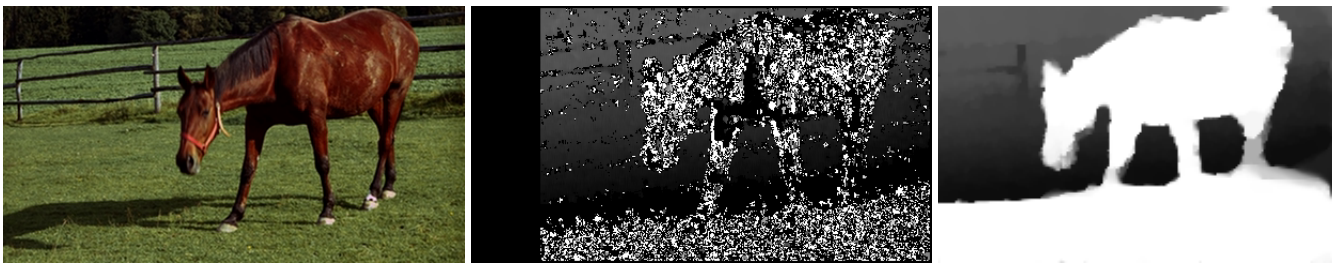
5.3.4 Convergence

We use the stopping criteria described in Section 5.1 with $\epsilon^{\text{abs}} = 1e^{-4}$ and $\epsilon^{\text{rel}} = 1e^{-4}$. This is different from the criteria used by Chan et al. [2]. Our stopping criteria results in more ADMM iterations, but it corresponds much better to the accuracy of the ADMM solution. We were still able to get convergence in relatively few iterations by scaling g so all entries were in $[0, 1]$.

We used a consistent value of $\rho = 1$ rather than updating ρ each iteration as done by Chan et al. [2]. We found that updating ρ prevented ADMM from converging.

6 Experiments

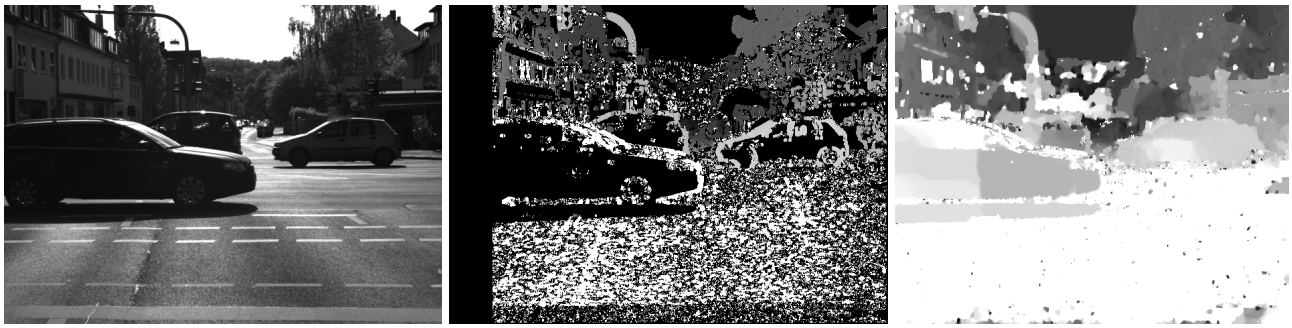
We tested this algorithm on a video also used in [2], specifically the "Horse" video found at `http://sp.cs.tut.fi/mobile3dtv/stereo-video/`. We scaled the video down to be 180 by 120 pixels and used all 140 frames. We optimized over the 3.09 million variables in 980 seconds, or 7 seconds per frame. As there is no ground truth provided for this dataset, qualitative comparison is necessary. We compare depth maps of a single frame of the video (original, OpenCV, and our algorithm):



The smoothed depth map is much less noisy than the original depth map, containing clearly defined regions of similar depth. Black pixels in the original depth map (representing unlabeled pixels with no depth estimate) have been smoothed over with neighboring depth values, improving accuracy.

Although it is difficult to show this in paper form, the depth maps are choppy when they are not smoothed over time (only smoothed over pixel locations). Smoothing over time reduces choppiness without sacrificing accuracy.

We also tried some alternative datasets. Specifically, we used stereo videos found at `http://hci.iwr.uni-heidelberg.de/Benchmarks/document/Challenging_Data_for_Stereo_and_Optical_Flow/`, which do not contain ground truth. The algorithm performed well on some of these videos but not others, mostly in line with how OpenCV's algorithm did on these videos. It performed relatively well on the CrossingCars video:



Our algorithm succeeded in filling in black regions and handling occlusions over time. When an occluded car becomes visible again, it maintains roughly the same depth that it had originally.

Additionally, we tried using the Indoor 1 dataset found at `http://research.microsoft.com/en-us/downloads/29d28301-1079-4435-9810-74709376bce1/` because this dataset contained ground truth. Unfortunately, none of the frame-by-frame algorithms we tested produced reasonable results on this dataset.

We tried algorithms other than OpenCV's StereoBM, but none was very accurate for the datasets we used. These algorithms were OpenCV's StereoSGBM, the MRF minimization-based algorithm found at `http://vision.middlebury.edu/MRF/code/`, and the Microsoft Research StereoMatcher package at `http://research.microsoft.com/en-us/downloads/9bc7fd74-5953-4064-9732-76405573aaef/default.aspx` (with a patch from `http://vision.middlebury.edu/stereo/code/`). None of these algorithms produced reasonable depth maps for our datasets.

For a more quantitative evaluation, we downloaded depth maps from top-ranked algorithms at `http://vision.middlebury.edu/stereo/` and ran our algorithm on them. We counted bad pixels in the same way the website does (proportion of non-occluded pixels that are more than 4 intensity units away from the ground truth value) on the teddy image.

Algorithm	Original Bad Pixels	New Bad Pixels	Percent Change
TSGO	8.08980%	8.09342%	0.04485%
AdaptingBP	7.05619%	7.05075%	-0.07714%
DoubleBP	8.30027%	8.30813%	0.09472%
ADCensus	6.21613%	6.19859%	-0.28215%
CoopRegion	8.30994%	8.31539%	0.06550%

In general, smoothing made very little difference to the accuracy of the depth maps. This is most likely because the output of the top algorithms is already relatively smooth.

7 Conclusion

Our improvements to the smoothing algorithm in [2] make it appropriate for use with OpenCV's depth maps. The algorithm succeeds in smoothing depth maps over space and time as long

as the original depth maps are reasonable. Our extensions to the algorithm handle unlabeled pixels and improve performance around edges. We implement the ADMM algorithm, which greatly improves performance to $O(n \log n)$ per iteration, allowing entire videos to be smoothed. Due to its quality and reasonable performance, our algorithm is appropriate for practical use in smoothing noisy but relatively accurate depth maps. In particular, our algorithm works well when only a small proportion of pixels (such as only pixels near edges) initially have depth estimates and others are unlabeled.

References

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[2] S. Chan, R. Khoshabeh, K. Gibson, P. Gill, and T. Nguyen. An augmented lagrangian method for total variation video restoration. *Image Processing, IEEE Transactions on*, 20(11):3097–3111, Nov 2011.

[3] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[4] C. Li. *An efficient algorithm for total variation regularization with applications to the single pixel camera and compressive sensing*. PhD thesis, Citeseer, 2009.