Published in Image Processing On Line on 2014–10–15.
Submitted on 2013–06–11, accepted on 2014–07–09.
ISSN 2105–1232 © 2014 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
http://dx.doi.org/10.5201/ipol.2014.97

Kolmogorov and Zabih's Graph Cuts Stereo Matching Algorithm

Vladimir Kolmogorov[1], Pascal Monasse[2], Pauline Tan[3]

[1] IST, Austria (vnk@ist.ac.at)
[2] Université Paris-Est, France (monasse@imagine.enpc.fr)
[3] CMLA, ENS Cachan, France (pauline.tan@ens-cachan.fr)

Abstract

Binocular stereovision estimates the three-dimensional shape of a scene from two photographs taken from different points of view. In rectified epipolar geometry, this is equivalent to a matching problem. This article describes a method proposed by Kolmogorov and Zabih in 2001, which puts forward an energy-based formulation. The aim is to minimize a four-term-energy. This energy is not convex and cannot be minimized except among a class of perturbations called expansion moves, in which case an exact minimization can be done with graph cuts techniques. One noteworthy feature of this method is that it handles occlusion: The algorithm detects points that cannot be matched with any point in the other image. In this method displacements are pixel accurate (no subpixel refinement).

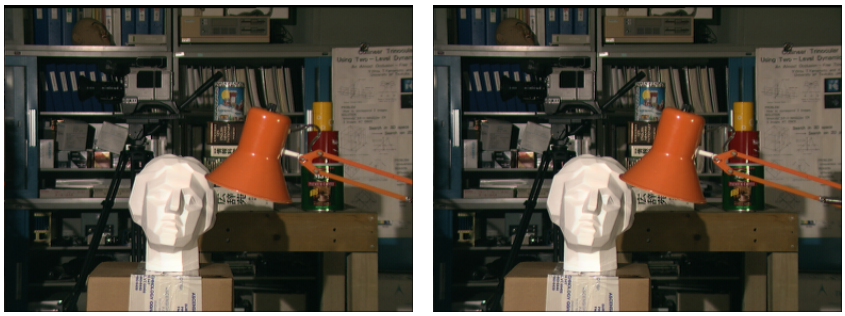
Source Code

The software rewritten from Kolmogorov's code is available at the IPOL web page of this article[1]. A set of stereo pairs is available and Kolmogorov and Zabih's algorithm can be tried on line. In the demo, the algorithm is run on six overlapping slices of the images, for efficiency purpose. Essentially, two parameters are needed: K associated to occlusion cost and λ to data fidelity. By default they are tuned automatically but they can be adapted to get better results.

Supplementary Material

In the demo, an optional rectification step can be launched before running the algorithm. The source code for this preprocessing step (not reviewed) can be found at the IPOL web page of this article[2].

Keywords: stereo matching; global method; graph cuts



(a) Left (reference) image (b) Right image

Figure 1: *Tsukuba* stereo pair, from Middlebury benchmark [8].

1 Introduction: Stereovision Concepts

Binocular stereovision estimates the 3D model of a scene given two images taken from different points of view. Such images are called a *stereo pair* (Figure 1). The simplest configuration that allows the estimation of a 3D map is the *rectified epipolar geometry*. In this case every 3D-point projected on one image is projected on the same horizontal line in the other image. These lines are *epipolar lines*. Basically, it corresponds to the configuration where the x-axes of the cameras are parallel to the line joining their centers and the principal (z-) axes are parallel (e.g. human eyes). When the images are not in this geometry, a rectification step can be used to warp the images. A method achieving the rectification, published in IPOL [7], is used as an optional preprocessing step of the demo associated to this article. In what follows, any stereo pair is supposed to be in rectified epipolar geometry.

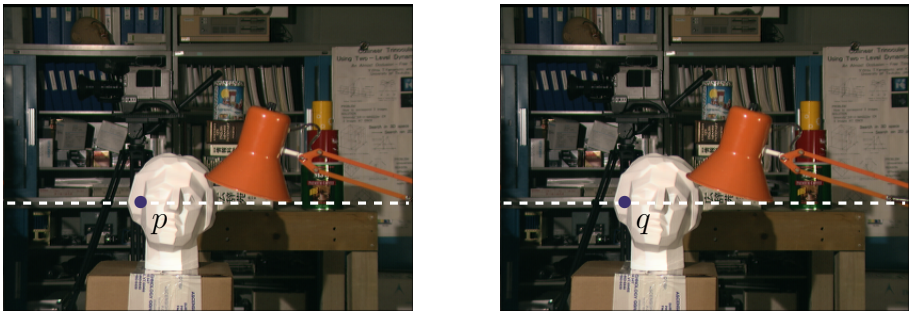
1.1 Stereovision in Epipolar Geometry

In epipolar geometry the motion of each point from one image to the other (called *disparity*) is horizontal (Figure 2). Moreover, Thales's theorem proves that each pixel disparity is inversely proportional to its distance from the observer. Thus, estimating the disparity map is sufficient to know the relative depth of a scene. Hence, binocular stereo algorithms usually only consist of matching every pixel of one image (the reference image) to a pixel in the other image.

1.2 Occlusion Problem

Unfortunately, estimating the depth of a scene from a stereo pair is an ill-posed problem. Indeed, some pixels are only visible in one image. They are called *occluded*. These pixels are difficult to deal with.

[1] http://dx.doi.org/10.5201/ipol.2014.97
[2] http://dx.doi.org/10.5201/ipol.2014.97



(a) Left image (b) Right image

Figure 2: Point p corresponds to q. The white dashed lines are the corresponding epipolar lines.

2 Problem Representation

2.1 Definitions

Here are some useful definitions and notations. Let $\mathcal{L}$ (resp. $\mathcal{R}$) be the set of pixels of the left (reference) image I_1 (resp. right image I_2). Then p (a pair of coordinates) denotes a pixel location from the left image and q a pixel from the right image. We also assume that the disparity lies in the 1D–range $\mathrm{I}_{\mathrm{disp}} = [x_{\min}, x_{\max}]$ since the images are supposed to be in rectified epipolar geometry.

Let $\mathcal{A} \subset \mathcal{L} \times \mathcal{R}$ be the set of pairs of pixels (denoted (p, q)) which may potentially correspond, i.e., $(p, q) \in \mathcal{A}$ iff $q - p \in \mathrm{I}_{\mathrm{disp}} \times \{0\}$. Elements of $\mathcal{A}$ are called *assignments*. Then we define for every assignment $a = (p, q)$ its disparity $d(a) = d(p, q) := q - p$. Let us say that two assignments $a_1 = (p_1, q_1)$ and $a_2 = (p_2, q_2)$ are *neighbors*, written $a_1 \sim a_2$, if p_1 and p_2 are adjacent and $d(a_1) = d(a_2)$.

A *configuration* is any map $f : \mathcal{A} \to \{0, 1\}$. If $a = (p, q)$ is an assignment, then $f(a) = 1$ means that p and q correspond under the configuration f. Such assignments are called *active assignments*. If $f(a) = 0$, then a is said to be *inactive*. A configuration is *unique* if for all pixels p (resp. q), there is at most one active assignment involving p (resp. q): for instance, considering p, if $f(p, q_1) = f(p, q_2) = 1$, then $q_1 = q_2$.

If an assignment $a = (p, q)$ is active under the unique configuration f, then the disparity of pixel p is $d_f(p) := d(a) = q - p$ and the disparity of pixel q is given by $d_f(q) := -d(a)$. Note that if every assignment $a = (p, q)$ involving p is inactive then p is not matched with any pixel. In that case, p is called occluded under the configuration f.

The method of Kolmogorov and Zabih [4] is based on assignments, not multi-labeled pixels. The advantage is that it handles uniqueness properly in both images. It also provides a natural way to handle occlusion, whereas adding a special label for occlusion would not use both images symmetrically.

2.2 Energy Function

The energy of a configuration f is defined as:

$$\mathrm{E}(f) = \mathrm{E}_{\mathrm{data}}(f) + \mathrm{E}_{\mathrm{occlusion}}(f) + \mathrm{E}_{\mathrm{smoothness}}(f) + \mathrm{E}_{\mathrm{uniqueness}}(f). \tag{1}$$

This energy has four terms. Each term promotes a desired property of the configuration we are looking for. The data term measures how well matched pairs fit, the occlusion term minimizes the

number of occluded pixels, the smoothness term penalizes the nonregularity of the configuration, and the last term enforces the uniqueness.

Data term The aim is to drive the algorithm to make the best matches. The better the matches (in terms of color similarity), the smaller the data term. The data term is given by

$$\mathrm{E}_{\mathrm{data}}(f) := \sum_{a, f(a)=1} D(a) = \sum_{a} D(a) \cdot 1\big(f(a) = 1\big), \tag{2}$$

where the function $1(\cdot)$ equals 1 when its argument is true, and 0 otherwise. The function D is a distance function which measures the dissimilarity between pixels p and q if $a = (p, q)$. Hence if pixels p and q are matched, the assignment $a = (p, q)$ is active and contributes to the data term. The demo chooses the classical SD (Squared Difference), which is adapted to Gaussian white noise, though the code offers as another option the AD (Absolute Difference). The difference is trimmed by a function T and can be written:

$$D_d(p, q) := T(|I_1(p) - I_2(q)|)^d, \tag{3}$$

for gray-scale images and

$$D_d(p, q) := \frac{T(|I_1^R(p) - I_2^R(q)|)^d + T(|I_1^G(p) - I_2^G(q)|)^d + T(|I_1^B(p) - I_2^B(q)|)^d}{3}, \tag{4}$$

for color images $I_i = (I_i^R, I_i^G, I_i^B)$, with $d = 1$ for AD and $d = 2$ for SD. The trim function T uses a threshold of `CUTOFF=30` and is defined as

$$T(x) := \min(\texttt{CUTOFF}, x). \tag{5}$$

A variant due to Birchfield and Tomasi is used to limit some effects due to sampling (see Section 4.3).

Occlusion term This term consists in maximizing the number of matches. In other words, one wants to minimize the number of occluded pixels. Since the number of occluded pixels is an affine function of the number of inactive assignments, any inactive assignment is penalized by penalty K:

$$\mathrm{E}_{\mathrm{occlusion}}(f) := \sum_{a, f(a)=0} K = \sum_{a} K \cdot 1\big(f(a) = 0\big) = K \times \#\mathcal{A} - \sum_{a} K \cdot 1\big(f(a) = 1\big). \tag{6}$$

This is equivalent to counting the number of inactive assignments. Then, the fewer the occluded pixels, the smaller the occlusion term.

Smoothness term The smoothness term favors piecewise constant maps. The idea is that adjacent pixels should have similar disparities, especially if their colors are close. In other words, if two assignments involving adjacent pixels p_1 and p_2 have the same disparity, then both should be either active or inactive, otherwise there is a penalty. The smoothness term concerns neighbor assignments $a_1 \sim a_2$ and is defined by

$$\mathrm{E}_{\mathrm{smoothness}}(f) := \sum_{a_1 \sim a_2} V_{a_1, a_2} \cdot 1\big(f(a_1) \neq f(a_2)\big), \tag{7}$$

where V_{a_1, a_2} is defined by

$$V_{a_1, a_2} := \begin{cases} \lambda_1 = 3\,\lambda & \text{if } \max\big(|I_1(p_1) - I_1(p_2)|, |I_2(q_1) - I_2(q_2)|\big) < 8 \\ \lambda_2 = \lambda & \text{otherwise.} \end{cases} \tag{8}$$

In (8), absolute values are replaced with ℓ^∞ norm for color images. If p_1 and p_2 have the same disparity the smoothness term for (a_1, a_2) is zero. Otherwise the penalty for not having the same disparity is small (λ) if there is a significant contrast between the adjacent pixels, otherwise it is bigger ($3\,\lambda$). Thus the term penalizes more the disparity jumps where there are no jumps in the intensity. Notice that if $d_f(p_1) \neq d_f(p_2)$ for the unique configuration f, the pair of assignments (a_1, a_2) contributes:

$$V_{a_1,(p_2,p_2+d_f(p_1))} + V_{(p_1,p_1+d_f(p_2)),a_2} \tag{9}$$

to the smoothness term, provided $(p_2, p_2 + d_f(p_1)) \in \mathcal{A}$ and $(p_1, p_1 + d_f(p_2)) \in \mathcal{A}$.

By default, the parameters λ_1 and λ_2 are set relatively to a common parameter λ as described above, but they can also be set by the user as an argument of the program. See Section 4.2 for more explanations about the choice of parameters K and λ.

Uniqueness term This term enforces the uniqueness of the configuration. It should be infinity if the configuration is nonunique, null otherwise. Then, one can write it as

$$\mathrm{E}_{\text{uniqueness}}(f) := \sum_{\substack{a_1=(p,q_1)\\a_2=(p,q_2)\\q_1\neq q_2}} \infty \cdot 1\big(f(a_1) = f(a_2) = 1\big) + \sum_{\substack{a_1=(p_1,q)\\a_2=(p_2,q)\\p_1\neq p_2}} \infty \cdot 1\big(f(a_1) = f(a_2) = 1\big). \tag{10}$$

Note that the energy we constructed is defined on binary label configurations (any assignment can only get value 0 or 1 under the configuration f).

3 Algorithm

3.1 Representating Energy Functions with Graphs

In this section we show that, in some particular cases, energies defined on binary label configurations can be minimized by graph cuts.

3.1.1 Graph and Cuts

A *directed graph* (or simply *graph*) is composed of a set $\mathcal{V}$ of vertices (or nodes) and a set $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ of directed edges, each with a nonnegative weight. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with two distinguished vertices s and t called the source and the sink. An *s-t cut* (or *cut*) is a partition $(\mathcal{V}^s, \mathcal{V}^t)$ of the vertices, such that $s \in \mathcal{V}^s$ and $t \in \mathcal{V}^t$. The cost of the cut is the sum of the weights of the edges from a vertex in $\mathcal{V}^s$ to a vertex in $\mathcal{V}^t$. A *minimum cut* of the graph is a cut with minimal cost.

If x and y are two vertices of the set $\mathcal{V}$, then (x, y) denotes the edge directed from x to y. Its weight (or capacity) in the graph $\mathcal{G}$ is denoted by $c_{\mathcal{G}}(x, y)$. Hence, the cost of a cut $(\mathcal{V}^s, \mathcal{V}^t)$ of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is given by the following formula:

$$c_{\mathcal{G}}(\mathcal{V}^s, \mathcal{V}^t) = \sum_{\substack{(u,v)\in\mathcal{E}\\u\in\mathcal{V}^s, v\in\mathcal{V}^t}} c_{\mathcal{G}}(u, v). \tag{11}$$

Figure 3 illustrates a graph with capacities and a flow. A flow is a map Φ from $\mathcal{E}$ to $\mathbb{R}^+$ such that:

$$\forall\, e = (x, y) \in \mathcal{E}, \quad 0 \leq \Phi(e) \leq c_{\mathcal{G}}(x, y), \tag{12}$$

$$\forall\, x \in \mathcal{V} \setminus \{s, t\}, \quad \sum_{e=(y,x)\in\mathcal{E}} \Phi(e) = \sum_{e=(x,y)\in\mathcal{E}} \Phi(e). \tag{13}$$

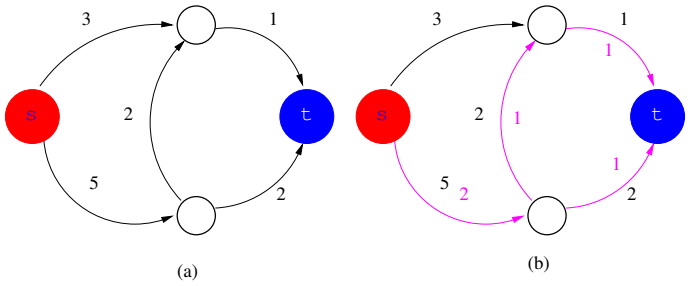


Figure 3: Vocabulary of network flow theory. (a) A graph with capacities. (b) In magenta a flow of value 2: the value on any edge must not exceed its capacity, at any node (except s and t) the incoming flow should be equal to the outgoing flow.

The value of a flow is the common value

$$\sum_{e=(s,x)\in\mathcal{E}} \Phi(e) = \sum_{e=(x,t)\in\mathcal{E}} \Phi(e). \tag{14}$$

A *maximum flow* of the graph is a flow with maximal value.
Min-cut and max-flow problems are dual linear problems:

Theorem 1 (Max-Flow/Min-Cut) *The cost of a minimum cut of a graph is the value of a maximum flow.*

Thanks to Ford-Fulkerson style algorithms, based on augmenting paths, a maximum flow is computable. In Kolmogorov and Zabih's software the max-flow is computed by an algorithm developed by Boykov and Kolmogorov [2]. Our implementation uses the same algorithm, but its inner working is out of the scope of this article. Experimentally, this algorithm is faster than most other known methods on typical vision graphs. As a result it is included in several higher level discrete optimization toolboxes, such as FastPD [6], which reputedly achieves the best performance.

3.1.2 Graph-Representable Functions

A function E of n binary variables is said to be *graph-representable* if there exists a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with source s and sink t and vertices $\mathcal{V}_0 = \{v_1, \ldots, v_n\} \subset \mathcal{V}$, and a constant C, such that for any $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ the value $\mathrm{E}(x_1, \ldots, x_n)$ is equal to C plus the minimum cost among all cuts $(\mathcal{V}^s, \mathcal{V}^t)$ of $\mathcal{G}$ in which $v_i \in \mathcal{V}^s$ if $x_i = 0$ and $v_i \in \mathcal{V}^t$ if $x_i = 1$ for any $i \in \{1, \ldots, n\}$. Note that there is no unicity of the representation.

The class $\mathcal{F}^2$ is defined to be the set of functions that can be written as a sum of functions of up to two binary variables:

$$\mathrm{E}(x_1, \ldots, x_n) = \sum_i \mathrm{E}^i(x_i) + \sum_{i<j} \mathrm{E}^{i,j}(x_i, x_j). \tag{15}$$

Theorem 2 ($\mathcal{F}^2$ theorem, [3, 5]) *Let* E *be a function from the class* $\mathcal{F}^2$. *Then,* E *is graph-representable if each term involving two binary variables is* submodular, *i.e., it satisfies the following inequality:*

$$\mathrm{E}^{i,j}(0,0) + \mathrm{E}^{i,j}(1,1) \leq \mathrm{E}^{i,j}(0,1) + \mathrm{E}^{i,j}(1,0). \tag{16}$$

Intuitively, submodularity favors pairs of variables with identical values. Let us check this theorem by constructing a graph representing a function from the class $\mathcal{F}^2$ that satisfies the submodularity

condition. We only consider unary functions E^i and pairwise functions $E^{i,j}$, since we show in Appendix A that the sum of graph-representable functions is still graph-representable. The reader should go to Appendix B to check in detail that each graph does represent the right function. Note that the submodularity condition is not only sufficient, as expressed here, but actually necessary [5]. We do not need this result in the current article, so we do not reproduce its proof. The spirit of the proof appears in Appendix C.

Remark: The definition of graph-representability allows the representing graph to have more vertices than the number of variables occurring in the graph-representable function. However, in practice, we only construct graphs with the minimal number of vertices.

- Unary function: A function E^i of one binary variable is always graph-representable. We assume that $E^i(0) = E_0$ and $E^i(1) = E_1$. If E_0 and E_1 are nonnegative, then, for instance, the graph of Figure 4 represents E^i.

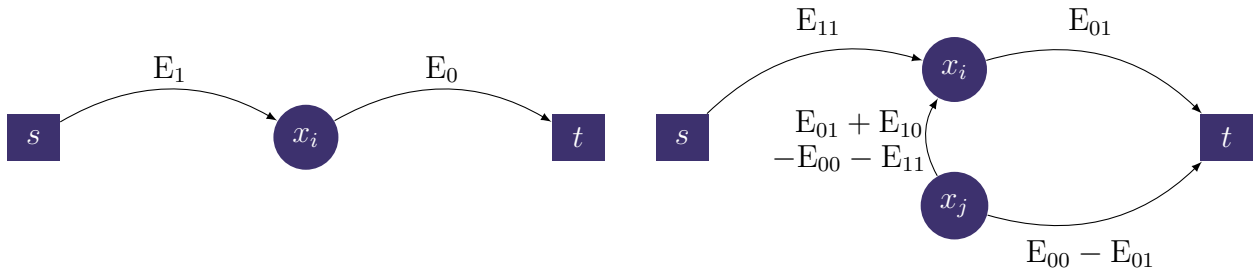
  In the general case (e.g., if E_0 or E_1 is negative), one may subtract the quantity $\min(E_0, E_1)$ to the capacities of the graph constructed above in order to get nonnegative weights. Then, only one edge is to be constructed to represent a given unary function. For instance, if $\min(E_0, E_1) = E_1$, then an edge from the source s to the node x_i with capacity $E_0 - E_1 > 0$ is constructed. For the sake of simplicity, in what follows, any unary function is represented by two edges that can *formally* be negative. This is used for the occlusion term by counting only energy terms $-K$ for active assignments, instead of K for inactive assignments, according to the last equality of (6).

- Pairwise function: A function $E^{i,j}$ of two binary variables is graph-representable if it is *submodular*. Indeed it may be written as

$$\begin{aligned} E^{i,j}(x_i, x_j) = &\begin{cases} E_{01} & \text{if } x_i = 0 \\ E_{11} & \text{if } x_i = 1 \end{cases} \\ &+ \begin{cases} E_{00} - E_{01} & \text{if } x_j = 0 \\ 0 & \text{if } x_j = 1 \end{cases} \\ &+ \begin{cases} E_{01} + E_{10} - E_{00} - E_{11} & \text{if } (x_i, x_j) = (1, 0) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{17}$$

  where $E^{i,j}(0,0) = E_{00}$, $E^{i,j}(0,1) = E_{01}$, $E^{i,j}(1,0) = E_{10}$, and $E^{i,j}(1,1) = E_{11}$. The first term only depends on variable x_i, and the second one only on x_j. So they can be represented as shown above. For the last term, one creates an edge (x_j, x_i) with the weight $E^{i,j}(0,1) + E^{i,j}(1,0) - E^{i,j}(0,0) - E^{i,j}(1,1)$ (which is nonnegative if the function is submodular). The complete construction of the graph of $E^{i,j}$ is illustrated in Figure 4.

>Code: `energy/energy.h` provides the interface to minimize $E \in \mathcal{F}^2$. The method of class `Energy` `add_variable(E0,E1)` adds a variable given the energies E^i of its two possible values. The method `add_term2(x,y, E00, E01, E10, E11)` adds a term $E^{i,j}$. These methods are used to construct the associated `Graph` object, defined in `maxflow/graph.h`, while `maxflow/maxflow.cpp` implements the max-flow computation. Unary functions are represented in `add_tweights` that allows negative weights, but that constructs in practice only one edge, from the source or to the sink, depending on the sign of the quantity $E_0 - E_1$. This edge is then weighted by $|E_0 - E_1|$ and a constant $\min(E_0, E_1)$ is added to the total flow value in order to represent exactly the function. Pairwise functions that satisfy the submodularity condition are represented in `add_edge`, that only allows nonnegative weights. Their definition is in `maxflow/graph.cpp`. Files in directory `maxflow` are not reviewed with this article.



(a) Representation of a unary function (go to Figure 17 in Appendix B to see the cuts).

(b) Representation of a pairwise function (go to Figure 18 in Appendix B to see the cuts).

Figure 4: Graph construction for unary and pairwise functions.

3.2 The Expansion Move Algorithm

3.2.1 Expansion Move

The energy (1) is unfortunately not graph-representable. However, the minimization of this energy can be approximated by an iterated constrained minimization, given by so-called expansion moves. The latter is the minimization of a graph-representable energy, so it is exact.

Fix f a unique configuration and $\alpha \in \mathrm{I}_{\mathrm{disp}} \times \{0\}$ a disparity value. A configuration f' is said to be an *α-expansion move* of f if:

$$f(a) = 1 \text{ and } d(a) = \alpha \quad \Rightarrow \quad f'(a) = 1, \tag{18}$$
$$f(a) = 0 \text{ and } d(a) \neq \alpha \quad \Rightarrow \quad f'(a) = 0. \tag{19}$$

Pixels with disparity α (in configuration f) keep the disparity α in f'; other pixels (occluded or not) adopt the disparity α in f', keep their state (occluded or keep their disparity), or become occluded. In terms of assignments,

- any active assignment with disparity α remains active;
- any inactive assignment with disparity different from α remains inactive;
- any other assignment can change state (active/inactive).

Thus, after an α-expansion move the number of assignments with disparity α cannot decrease. Note that for any given (unique) configuration f and any disparity value α, f is an α-expansion move of itself.

Against all odds, the energy $\mathrm{E}(f')$ is not graph-representable. It is indeed clear that the uniqueness terms do not comply with the *same state preference* sustained by submodularity. One solution to this issue consists in applying a change of variable by introducing a function g_α associated to an α-expansion move configuration f'. This new function acts as a toggle on the activity of each variable assignment a. In other words, $g_\alpha(a) = 1$ indicates that the activity of a is swapped during the current α-expansion move. Mathematically, this interpretation reads:

$$g_\alpha(a) = 1 \text{ iff } f(a) \neq f'(a). \tag{20}$$

We denote by $\mathcal{A}^\circ$ the set of active assignments with disparity different from α, $\mathcal{A}^\alpha$ the set of assignments (either active or inactive) with disparity α. Because of the α-expansion move strategy itself (18) and (19), only certain assignments ($\{a \in \mathcal{A}^\alpha \mid f(a) = 0\} \cup \mathcal{A}^\circ$) can see their status

swapped. The rest of the assignments are not actual variables during the α-expansion move. To leverage this consideration, we actually do not include those assignments in the definition of g_α. As a result, the vector g_α is shorter than the original vector f'. Nonetheless, considering the interpretation of $g_\alpha(a)$ (20), one can easily recover the corresponding values of $f'(a)$ for all assignments (variable or non-variable alike) as follows:

$$f'(a) = \begin{cases} g_\alpha(a) & \text{if } a \in \mathcal{A}^\alpha \text{ and } f(a) = 0 \\ 1 - g_\alpha(a) & \text{if } a \in \mathcal{A}^\circ \\ f(a) & \text{otherwise.} \end{cases} \tag{21}$$

The key idea is that, given the previous change of variable, the energy assumes a new expression that shall be proven graph representable

$$\mathrm{E}_{f,\alpha}(g_\alpha) := \mathrm{E}(f'). \tag{22}$$

3.2.2 Energy Minimization

Let us show that the function $\mathrm{E}_{f,\alpha}$ is graph-representable. To do so, we need to express $\mathrm{E}_{f,\alpha}$ explicitly as a function of g_α and then verify that the pairwise terms are submodular. The function $\mathrm{E}_{f,\alpha}$ can be written as the sum of four terms:

$$\mathrm{E}_{f,\alpha}(g_\alpha) = \mathrm{E}_{f,\alpha,\mathrm{data}}(g_\alpha) + \mathrm{E}_{f,\alpha,\mathrm{occlusion}}(g_\alpha) + \mathrm{E}_{f,\alpha,\mathrm{smoothness}}(g_\alpha) + \mathrm{E}_{f,\alpha,\mathrm{uniqueness}}(g_\alpha). \tag{23}$$

- Data term: If $\mathrm{E}_{f,\alpha,\mathrm{data}}(g_\alpha) = \mathrm{E}_{\mathrm{data}}(f')$, then by distinguishing assignments in $\mathcal{A}^\alpha$, in $\mathcal{A}^\circ$ and the other ones:

$$\mathrm{E}_{f,\alpha,\mathrm{data}}(g_\alpha) = \sum_{\substack{a \in \mathcal{A}^\alpha \\ f(a)=1}} D(a) + \sum_{\substack{a \in \mathcal{A}^\alpha \\ f(a)=0}} D(a) \cdot 1\big(g_\alpha(a) = 1\big) + \sum_{a \in \mathcal{A}^\circ} D(a) \cdot 1\big(g_\alpha(a) = 0\big). \tag{24}$$

  This term is a sum of unary functions, so it is graph-representable.

- Occlusion term: Similarly, $\mathrm{E}_{f,\alpha,\mathrm{occlusion}}(g_\alpha) = \mathrm{E}_{\mathrm{occlusion}}(f')$, i.e.,

$$\mathrm{E}_{f,\alpha,\mathrm{occlusion}}(g_\alpha) = \sum_{\substack{a \in \mathcal{A}^\alpha \\ f(a)=0}} K \cdot 1\big(g_\alpha(a) = 0\big) + \sum_{a \in \mathcal{A}^\circ} K \cdot 1\big(g_\alpha(a) = 1\big) + \sum_{a \notin \mathcal{A}^\alpha \cup \mathcal{A}^\circ} K, \tag{25}$$

  so this term is also graph-representable.

- Smoothness term: By definition, $\mathrm{E}_{f,\alpha,\mathrm{smoothness}}(g_\alpha) = \mathrm{E}_{\mathrm{smoothness}}(f')$. Notice that $a_1 \sim a_2$ implies by definition that they have the same disparity, so that a_1 and a_2 are either both in $\mathcal{A}^\alpha$ or none is. So, by distinguishing the case where $a_1 \sim a_2$ are of disparity α and the case where they are not, we have

$$\begin{aligned} \mathrm{E}_{f,\alpha,\mathrm{smoothness}}(g_\alpha) = &\sum_{\substack{a_1 \sim a_2 \\ a_1, a_2 \in \mathcal{A}^\alpha \\ f(a_1)=0 \\ f(a_2)=0}} V_{a_1,a_2} \cdot 1\big(g_\alpha(a_1) \neq g_\alpha(a_2)\big) + \sum_{\substack{a_1 \sim a_2 \\ a_1, a_2 \in \mathcal{A}^\alpha \\ f(a_1)=0 \\ f(a_2)=1}} V_{a_1,a_2} \cdot 1\big(g_\alpha(a_1) = 0\big) \\ &+ \sum_{\substack{a_1 \sim a_2 \\ a_1, a_2 \in \mathcal{A}^\circ}} V_{a_1,a_2} \cdot 1\big(g_\alpha(a_1) \neq g_\alpha(a_2)\big) + \sum_{\substack{a_1 \sim a_2 \\ a_1 \in \mathcal{A}^\circ \\ a_2 \notin \mathcal{A}^\circ}} V_{a_1,a_2} \cdot 1\big(g_\alpha(a_1) = 0\big). \end{aligned} \tag{26}$$

The second and the last sums are graph-representable as sums of unary functions. Let us check that the remaining terms are submodular.

For all pairs of neighbor assignments (a_1, a_2) in $\mathcal{A}^\alpha$ or in $\mathcal{A}^\circ$, we set $\mathrm{E}_{a_1,a_2}(x_1, x_2) = V_{a_1,a_2} \cdot 1\big(x_1 \neq x_2\big)$. Then, $\mathrm{E}_{a_1,a_2}(0,0) = \mathrm{E}_{a_1,a_2}(1,1) = 0$ and $\mathrm{E}_{a_1,a_2}(0,1) = \mathrm{E}_{a_1,a_2}(1,0) = V_{a_1,a_2} \geq 0$, which proves that E_{a_1,a_2} is submodular. Hence, the first and the third sums are both graph-representable thanks to the $\mathcal{F}^2$ theorem.

- Uniqueness term: There is only one assignment involving a given pixel p in $\mathcal{A}^\alpha$ and at most one in $\mathcal{A}^\circ$. Assuming that uniqueness was actually enforced in f, then at most one of them can be active in f'. The same argument applies to a pixel q in the other image. Thus, the uniqueness term, defined by $\mathrm{E}_{f,\alpha,\mathrm{uniqueness}}(g_\alpha) = \mathrm{E}_{\mathrm{uniqueness}}(f')$, can be written:

$$\begin{aligned}\mathrm{E}_{f,\alpha,\mathrm{uniqueness}}(g_\alpha) = &\sum_{\substack{a_1=(p,p+\alpha)\in\mathcal{A}^\alpha \\ a_2=(p,p+d_f(p))\in\mathcal{A}^\circ \\ f(a_1)=0}} \infty \cdot 1\big(g_\alpha(a_1)=1 \text{ and } g_\alpha(a_2)=0\big) \\ + &\sum_{\substack{a_1=(q-\alpha,q)\in\mathcal{A}^\alpha \\ a_2=(q+d_f(q),q)\in\mathcal{A}^\circ \\ f(a_1)=0}} \infty \cdot 1\big(g_\alpha(a_1)=1 \text{ and } g_\alpha(a_2)=0\big).\end{aligned} \tag{27}$$

Indeed, in each term a_1 is inactive, while a_2 is active in f. Remembering that g_α takes value 1 for a *change* of activity, the terms ensure that a_1 cannot become active while a_2 remains active. If we write E for the function $\mathrm{E}(x_1, x_2) = \infty \cdot 1(x_1 = 1 \text{ and } x_2 = 0)$, then $\mathrm{E}(0,0) + \mathrm{E}(1,1) = 0 \leq \mathrm{E}(0,1) + \mathrm{E}(1,0) = \infty$, that is, E is submodular. Thus, the uniqueness term is graph-representable. Recall that submodularity of this term was our main motivation to introduce g_α in the first place. As an additional justification, one can easily check that the uniqueness term of the energy with respect to f' is not graph-representable, because none of the terms is submodular.

All the previous developments show that $\mathrm{E}_{f,\alpha}$ is graph-representable. As a result, there exists a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertices $\{v_a\}_{a\in\{a\in\mathcal{A}^\alpha | f(a)=0\}\cup\mathcal{A}^\circ} \subset \mathcal{V}$ (on top of s and t) such that

$$\mathrm{E}_{f,\alpha}(g_\alpha) = \min_{\substack{(\mathcal{V}^s,\mathcal{V}^t) \text{ cut} \\ \text{satisfying } (*)}} c_{\mathcal{G}}(\mathcal{V}^s, \mathcal{V}^t) + C \qquad \text{with} \qquad (*) \begin{cases} v_a \in \mathcal{V}^s & \text{if } g_\alpha(a) = 0 \\ v_a \in \mathcal{V}^t & \text{if } g_\alpha(a) = 1 \end{cases}. \tag{28}$$

Thus, finding the α-expansion move with the lowest energy is equivalent to minimizing the function $\mathrm{E}_{f,\alpha}$ among all g_α:

$$\min_{g_\alpha} \mathrm{E}_{f,\alpha}(g_\alpha) = \min_{g_\alpha} \min_{\substack{(\mathcal{V}^s,\mathcal{V}^t) \text{ cut} \\ \text{satisfying } (*)}} c_{\mathcal{G}}(\mathcal{V}^s, \mathcal{V}^t) + C = \min_{(\mathcal{V}^s,\mathcal{V}^t) \text{ cut}} c_{\mathcal{G}}(\mathcal{V}^s, \mathcal{V}^t) + C, \tag{29}$$

which is a minimum cut problem, the last equality being due to the fact that any cut is uniquely associated with a g_α through condition $(*)$. This can be computed exactly by maximizing the flow in the graph $\mathcal{G}$. The optimal vector g^*_α is then obtained thanks to the optimal cut $(\mathcal{V}^{*,s}, \mathcal{V}^{*,t})$, by

$$\forall\, a \in \mathcal{A}^\alpha \cup \mathcal{A}^\circ \setminus \{a \in \mathcal{A}^\alpha \mid f(a) = 1\}, \qquad g^*_\alpha(a) = \begin{cases} 0 & \text{if } v_a \in \mathcal{V}^{*,s} \\ 1 & \text{if } v_a \in \mathcal{V}^{*,t} \end{cases}, \tag{30}$$

and we can recover the optimal α-expansion move f^* thanks to (21).

Remark: For an initial configuration f and any $\alpha \in \mathrm{I}_{\mathrm{disp}} \times \{0\}$, a graph of at most $(2 \times \#\mathcal{L})$ vertices is constructed to compute the α-expansion move of f that decreases the most the energy.

Algorithm 1, implemented in method `Match::run`, file `kz2.cpp`, proceeds by looking iteratively for optimal α-expansion move configurations, for varying values of α. Such an operation is called an *iteration*. The values α are selected in a random order. More precisely the array of α values is either shuffled once for all and kept fixed at each iteration (this is the case in the demo and the default settings in the code) or reshuffled every time. The code performs 4 iterations at most (parameter of the program), but it may stop earlier.

Algorithm 1: An iteration of the expansion move algorithm

Input: a unique configuration f, interval of disparities $\mathrm{I}_{\mathrm{disp}}$, achieved α-expansions array `done`
Output: updated unique configuration f with smaller or equal energy

```
1 foreach α (in randomly ordered I_disp) do
2     if not done[α] then
3         Find the α-expansion move f* of f that decreases the most the energy:
                        f* ← arg min_{f' α−expansion move of f} E(f')
          if E(f*) < E(f) then
4             f ← f*
5             done[:]← false
6         done[α]← true
7         if done[:]=true then
8             return f
```

Thus, the expansion move algorithm does not proceed to an exact minimization of the energy function, but considers a class of perturbations. However, each step of energy decrease is optimal. The array `done` records for each α expansion whether it decreased the energy (before the first iteration, it must be set to `false`). This ensures that we do not reattempt an α-expansion for the same disparity α if nothing has changed since the last one. The test in line 7 of Algorithm 1 is verified by maintaining a counter `nDone` of the number of false entries in the array `done`, which has to be updated in lines 5 and 6.

All elements of the array `done` are set to `false` before the first iteration and the initial configuration f has only inactive assignments (all pixels occluded).

4 Implementation Details

In this section, we give some details on the implementation of the Kolmogorov and Zabih's method. One can also read the comments in the source code provided on the IPOL webpage.

4.1 Expansion Move Algorithm

We recall that the expansion move algorithm for a given disparity value α aims at finding the α-expansion move of a configuration f with lowest energy. This is equivalent to minimizing the graph-representable energy $\mathrm{E}_{f,\alpha}$ among all reduced configurations g_α. In other words, one wants to find the minimal cut of a graph that represents the energy $\mathrm{E}_{f,\alpha}$. Once this minimal cut is computed,

it leads to an optimal configuration f^*. The energy $\mathrm{E}(f^*)$ of the latter is then compared with the energy of the initial configuration f. The graph construction will be detailed in Section 4.1.1, the minimization in 4.1.2 and steps to recover the optimal configuration in 4.1.3.

The initial configuration f is encoded by the pixel disparities in images I_1 and I_2, which are stored in two arrays `d_left` and `d_right`. For nonoccluded pixels p in $\mathcal{L}$ and q in $\mathcal{R}$,

$$d_f(p) = \texttt{d\_left(p)} \qquad \text{and} \qquad d_f(q) = \texttt{d\_right(q)}. \tag{31}$$

A special value `OCCLUDED` is used in `d_left` and `d_right` for occluded pixels.

>Code: The graph is built and the optimal graph cut interpreted as a new configuration in method `Match::ExpansionMove` of file `kz2.cpp`.

4.1.1 Graph Construction

Let f be a unique configuration and α a disparity value. We construct one node for each variable of g_α and appropriate edges to represent the energy $\mathrm{E}_{\alpha,f}$.

Adding nodes in the graph For the sake of notation brevity, we introduce $\mathcal{A}(f) := f^{-1}(\{1\}) = \{a \in \mathcal{A} \mid f(a) = 1\}$, representing the set of all active assignments under configuration f. We recall that $\mathcal{A}^\circ$ is by definition a subset of $\mathcal{A}(f)$ whose complement $\mathcal{A}(f) \setminus \mathcal{A}^\circ = \mathcal{A}(f) \cap \mathcal{A}^\alpha$ gathers all active assignments with disparity α. With this notation, the energy $\mathrm{E}_{\alpha,f}$ depends on three types of assignments, $a \in \mathcal{A}^\alpha \cap \mathcal{A}(f)$, $a \in \mathcal{A}^\alpha \setminus \mathcal{A}(f)$ and eventually $a \in \mathcal{A}^\circ$. Nonetheless, only the last two categories correspond to actual variables, and the graph under construction should exclusively allocate one node per true variable assignment. The remaining type of assignments $a \in \mathcal{A}^\alpha \cap \mathcal{A}(f)$ shall however be involved in the definition of the edge capacities as constant counterparts.

If a is an inactive assignment from $\mathcal{A}^\alpha$, then $a = (p, p+\alpha)$; if a is an assignment from $\mathcal{A}^\circ$, then $a = (p, p + d_f(p))$. Thus, for each pixel p in $\mathcal{L}$, we distinguish two cases:

- If p is nonoccluded (under the initial configuration f), then there exists a unique pixel q such that the assignment $a = (p, q)$ is active. If $d_f(p) = \alpha$, no node is constructed for the pixel p. If $d_f(p) \neq \alpha$, then $a \in \mathcal{A}^\circ$ and one has to construct a node for a; moreover, if $p+\alpha$ lies within the right image domain, then another node is constructed for the inactive assignment $a' = (p, p+\alpha)$.
- If p is occluded, there is no active assignment involving p. Hence, if $p + \alpha$ exists in the right image, a node for the inactive assignment $a' = (p, p + \alpha)$ is constructed ; else, there is no node associated to the pixel p.

Therefore, for each pixel p in the left image, one constructs 0, 1 or 2 nodes associated to assignments in $\mathcal{A}^\alpha \setminus \mathcal{A}(f)$ and $\mathcal{A}^\circ$.

Since there may be up to two nodes created per pixel, we introduce two arrays, `varsA` and `vars0`, indexed by pixel p, to track the nodes associated to each pixel. For each pixel p, `varsA(p)` (resp. `vars0(p)`) can take three different values: `VAR_ALPHA`, `VAR_ABSENT`, and `a` (resp. `o`); the latter is an identifier of a node concerning p, created on demand and corresponding to a binary variable of g_α. In the first case, no node is created for pixel p because the associated assignment $a = (p, p+\alpha)$ remains active among all configurations; in the second case, no node in $\mathcal{A}^\circ$ or $\mathcal{A}^\alpha \setminus \mathcal{A}(f)$ is built for pixel p because the assignment p is occluded or $p + \alpha$ does not exist in the right image; the last case leads to the construction of a node in $\mathcal{A}^\alpha \setminus \mathcal{A}(f)$ (resp. $\mathcal{A}^\circ$). More precisely,

- `varsA(p)=a` means that the node `a` associated to the assignment $(p, p + \alpha)$ has to be added;
- `vars0(p)=o` means that the node `o` associated to the assignment $(p, p+d_f(p))$ has to be added.

>Code: A function named `IS_VAR` was specifically designed to check if a node `a` or `o` points to a node of the graph (as opposed to the `VAR_ALPHA` and `VAR_ABSENT` cases).

According to our analysis, Table 1 summarizes the filling rules of the arrays `varsA` and `vars0`, depending on the state of pixel p.

<table>
<tr><td></td><td colspan="2">p is nonoccluded</td><td rowspan="2">p is occluded</td></tr>
<tr><td></td><td>$d_f(p) = \alpha$</td><td>$d_f(p) \neq \alpha$</td></tr>
<tr><td rowspan="2">$p + \alpha$ is in the right image</td><td>varsA(p)=VAR_ALPHA</td><td colspan="2">varsA(p)=a</td></tr>
<tr><td>vars0(p)=VAR_ALPHA</td><td>vars0(p)=o</td><td>vars0(p)=VAR_ABSENT</td></tr>
<tr><td rowspan="2">$p + \alpha$ is not in the right image</td><td rowspan="2">impossible</td><td colspan="2">varsA(p)=VAR_ABSENT</td></tr>
<tr><td>vars0(p)=o</td><td>vars0(p)=VAR_ABSENT</td></tr>
</table>

Table 1: Filling variables `vars0(p)` and `varsA(p)` for a pixel p in the left image, with `o` and `a` denoting node identifiers for new variables created on demand in $\mathcal{A}^\circ$ and $\mathcal{A}^\alpha$ respectively.

>Code: The nodes are constructed in method `Match::build_nodes`. They are created by the function `add_variable` when `varsA(p)` (resp. `vars0(p)`) identifies a variable, that is, its value is neither `VAR_ALPHA` nor `VAR_ABSENT`. Notice that there are $2n$ nodes at most in the graph, excluding s and t, where n stands for the number of pixels in each image.

Adding edges in the graph To construct a graph representing a function in class $\mathcal{F}^2$, we construct unary functions and pairwise functions. In the software, such terms are constructed in several functions:

1. `add_term1(node,E0,E1)`: adds a unary function depending on variable `node`, such that the node contributes `E0` to the cost of the cut if g_α(`node`) $= 0$, and `E1` otherwise;

2. `add_term2(node1,node2,E00,E01,E10,E11)`: adds a pairwise term depending on variables `node1` and `node2`, such that if g_α(`node1`) $= 0$ and g_α(`node2`) $= 0$, the cost `E00` is added to the cut cost; if g_α(`node1`) $= 0$ and g_α(`node2`) $= 1$, the added cost is `E01`; if g_α(`node1`) $= 1$ and g_α(`node2`) $= 0$, the added cost is `E10`; if g_α(`node1`) $= 1$ and g_α(`node2`) $= 1$, the added cost is `E11`.

3. `forbid01(node1,node2)` is the formal equivalent of `add_term2(node1,node2,0,`∞`,0,0)` but takes care of not overflowing. It prevents the configuration g_α(`node1`) $= 0$ and g_α(`node2`) $= 1$.

Even though the minimizer g_α is independent of the constant C in the graph representation of the energy $\mathrm{E}_{f,\alpha}$, it is still useful to recover the exact energy from the cost of the minimum cut. Hence, constant values also have to be taken into account; they are handled by the function:

4. `add_constant(value)`: adds the constant `value` to the cost of all graph cuts.

This function is cumulative, that is, the constants of each term are cumulated in a member `Econst` of class `Energy`. In practice, that only means adding `Econst` to the cost of the minimal cut, which is useful to test whether $\mathrm{E}(f^*)$ is lower than $\mathrm{E}(f)$ without computing directly $\mathrm{E}(f^*)$.

>Code: These functions are implemented in `energy/energy.h`. The computation of the energy of the current configuration is done in method `Match::ComputeEnergy`, file `kz2.cpp`, but it is only called in debug mode for sanity check; otherwise the energy is always recovered from the graph cut.

Let us detail the construction of the edges for each term of the energy $\mathrm{E}_{f,\alpha}$. For the sake of simplicity, we deal with the data term and the occlusion term simultaneously. That is, instead of

considering the penalty $D(a)$ for the data term and the penalty K for the occlusion term, we assign to the data penalty the value $D'(a) := D(a) - K$ and 0 to the occlusion penalty. It may happen that $D'(a) < 0$, but we have shown in Section 3.1.2 that it is not a problem for unary terms.

$$\mathrm{E}_{f,\alpha,\mathrm{data+occ}}(g_\alpha) := \sum_{\substack{a\in\mathcal{A}^\alpha \\ f(a)=1}} D'(a) + \sum_{\substack{a\in\mathcal{A}^\alpha \\ f(a)=0}} D'(a)\cdot 1\big(g_\alpha(a)=1\big) + \sum_{a\in\mathcal{A}^\circ} D'(a)\cdot 1\big(g_\alpha(a)=0\big) \tag{32}$$

$$= \mathrm{E}_{f,\alpha,\mathrm{data}}(g_\alpha) + \mathrm{E}_{f,\alpha,\mathrm{occlusion}}(g_\alpha) - (\#\mathcal{A}\times K). \tag{33}$$

This is equivalent to subtract the constant $\#\mathcal{A}\times K$, which depends on neither f nor α, hence we may ignore it in the computation of the energy. In what follows, we may refer to the data+occlusion term as the data term.
>Code: The data penalty $D(a)$ is computed in `data.cpp`. The data+occlusion penalty $D'(a)$ for the assignment $a = (p, q)$ is given by `Match::data_occlusion_penalty(p,q)`.

Since $\mathrm{E}_{f,\alpha,\mathrm{data}}$ and $\mathrm{E}_{f,\alpha,\mathrm{occlusion}}$ are graph-representable, this term is also graph-representable. It depends of three kinds of assignments:

- active assignments in $\mathcal{A}^\alpha$ (set to `VAR_ALPHA` in `varsA` and `vars0`): such assignments contribute the constant $D'(a)$ to the energy;
- inactive assignments in $\mathcal{A}^\alpha$ (encoded in `varsA` by variables `a`): such assignments contribute the cost $D'(a)$ to the energy if $g_\alpha(\mathtt{a}) = 1$ and 0 otherwise;
- assignments in $\mathcal{A}^\circ$ (encoded in `vars0` by variables `o`): such assignments contribute $D'(a)$ to the energy if $g_\alpha(\mathtt{o}) = 0$ and 0 otherwise.

Table 2 gives the construction of the data+occlusion term for each case.

assignment	node behavior	encoding	term construction
$a \in \mathcal{A}^\alpha \cap \mathcal{A}(f)$ $a = (p, p+\alpha)$	a remains active	`varsA(p)=VAR_ALPHA` `vars0(p)=VAR_ALPHA`	`add_constant(`$D'(a)$`)`
$a \in \mathcal{A}^\alpha \setminus \mathcal{A}(f)$ $a = (p, p+\alpha)$	a is variable	`varsA(p)=a`	`add_term1(a,0,`$D'(a)$`)`
$a \in \mathcal{A}^\circ$ $a = (p, p+d_f(p))$	a is variable	`vars0(p)=o`	`add_term1(o,`$D'(a)$`,0)`

Table 2: Construction of the data+occlusion term.

>Code: The edges associated to the data term are added in `Match::build_nodes` as we go along constructing the nodes themselves. This is made simple because each node is associated to a unique data term.

To handle the smoothness term, given by (26), one must first interpret the notion of assignment neighborhood in terms of pixels, since the assignments are indexed by their first pixel. We recall that two assignments $a_1 = (p_1, q_1)$ and $a_2 = (p_2, q_2)$ are said to be neighbors, written $a_1 \sim a_2$, if p_1 and p_2 are adjacent and $d(a_1) = d(a_2)$. Let p_1 and p_2 be two adjacent pixels. We present all the assignment pairs involving these pixels in (26). To simplify the following discussion we introduce with a slight abuse of notation the generic assignment $a_i(d) := (p_i, p_i + d)$ where $i \in \{1, 2\}$. The assignment pairs under consideration are listed hereafter:

1. First row of (26): At least one of $a_1(\alpha)$ or $a_2(\alpha)$ is inactive. Then we have two subcases:

- $a_1(\alpha)$ and $a_2(\alpha)$ are both inactive: a pairwise term is built, since a cost is added to the energy iff $g_\alpha(a_1(\alpha)) \neq g_\alpha(a_2(\alpha))$;
- one, say $a_1(\alpha)$, is inactive while the other one $a_2(\alpha)$ is active (`varsA(p2)=VAR_ALPHA`): this brings up the construction of a unary term, which depends on the inactive assignment $a_1(\alpha)$.

2. Second row of (26): Both $a_1(\alpha)$ and $a_2(\alpha)$ are inactive but at least one of p_1 or p_2 is non occluded. There are two subcases:

   - Both are non occluded and $d_f(p_1) = d_f(p_2) := d$, then $a_1(d) \sim a_2(d)$ and a pairwise term is added to the energy;
   - Only one is non occluded or $d_f(p_1) \neq d_f(p_2)$, then there may be two pairs of assignments involving p_1 and p_2, namely:
     - $a_1(d_f(p_1))$ associated with $a_2(d_f(p_1))$. This pair yields a unary term (last sum in (26)) under the condition that $a_1(d_f(p_1)) \in \mathcal{A}^\circ$ and $a_2(d_f(p_1))$ exists (it is then inactive).
     - $a_2(d_f(p_2))$ associated with $a_1(d_f(p_2))$. Similarly this pair is involved in (26) when $a_2(d_f(p_2)) \in \mathcal{A}^\circ$ and $a_1(d_f(p_2))$ exists (it is then inactive).

Table 3 summarizes the construction of the smoothness term with the notations `a1`, `a2` for assignments in $\mathcal{A}^\alpha$, and `o1`, `o2` for assignments in $\mathcal{A}^\circ$.

assignments	node behavior	encoding	term construction
$a_1 \in \mathcal{A}^\alpha \setminus \mathcal{A}(f)$ $a_1 = (p_1, p_1 + \alpha)$ $a_2 \in \mathcal{A}^\alpha \setminus \mathcal{A}(f)$ $a_2 = (p_2, p_2 + \alpha)$	a_1 is variable a_2 is variable	`varsA(p1)=a1` `varsA(p2)=a2`	`add_term2(a1,a2,` `0,`V_{a_1,a_2}`,`V_{a_1,a_2}`,0)`
$a_1 \in \mathcal{A}^\alpha \setminus \mathcal{A}(f)$ $a_1 = (p_1, p_1 + \alpha)$ $a_2 \in \mathcal{A}^\alpha \cap \mathcal{A}(f)$ $a_2 = (p_2, p_2 + \alpha)$	a_1 is variable a_2 remains active	`varsA(p1)=a1` `varsA(p2)=VAR_ALPHA`	`add_term1(a1,`V_{a_1,a_2}`,0)`
$a_1 \in \mathcal{A}^\circ$ $a_1 = (p_1, p_1 + d)$ $a_2 \in \mathcal{A}^\circ$ $a_2 = (p_2, p_2 + d)$	a_1 is variable a_2 is variable	`vars0(p1)=o1` `vars0(p2)=o2`	`add_term2(o1,o2,` `0,`V_{a_1,a_2}`,`V_{a_1,a_2}`,0)`

$a_1 \in \mathcal{A}^\circ$
$a_1 = (p_1, p_1 + d)$
$a_2 \in \mathcal{A} \setminus \mathcal{A}^\circ$
$a_2 = (p_2, p_2 + d)$
	a_1 is variable
	a_2 remains inactive
		`vars0(p1)=o1`
		`d_left(p1)`$\neq$`d_left(p2)`
		`p2+d_left(p1)`$\in \mathcal{R}$
			`add_term1(o1,`V_{a_1,a_2}`,0)`

Table 3: Construction of the smoothness term for two adjacent pixels `p1` and `p2`.

>Code: The smoothness penalty for assignments $a_1 = (p_1, p_1 + d)$ and $a_2 = (p_2, p_2 + d)$ is given by `Match::smoothness_penalty`. The corresponding edges are built in `Match::build_smoothness`. To enumerate the neighbors of a pixel, we use the array of two unit translations `NEIGHBORS`. This is actually *half* of the neighborhood system so as to avoid counting twice the energy of equivalent assignement pairs (i.e., (a_1, a_2) and (a_2, a_1)).

Let us eventually build the uniqueness term. In (27), the first term enforces the uniqueness condition in the left image, the second one enforces the uniqueness in the right image, that is, the

first (resp. second) term prevents a pixel $p \in \mathcal{L}$ (resp. $q \in \mathcal{R}$) to be matched with more than one pixel in the right image (resp. left image). For any pixel p in the left image (resp. q in the right image) with disparity different from α (resp. $-\alpha$), it forbids the assignments $a_1 = (p, p + d_f(p))$ and $a_2 = (p, p+\alpha)$ (resp. $a_1 = (q + d_f(q), q)$ and $a_2 = (q - \alpha, q)$), when they exist, to be both active. See Table 4 for the construction of the uniqueness term.

assignments	node behavior	encoding	term construction
$a_1 \in \mathcal{A}^\circ$ $a_1 = (p, p + d_f(p))$ $a_2 \in \mathcal{A}^\alpha \setminus \mathcal{A}(f)$ $a_2 = (p, p + \alpha)$	a_1 is variable a_2 is variable	`vars0(p)=o` `varsA(p)=a`	`forbid01(o,a)`

$a_1 \in \mathcal{A}^\circ$
$a_1 = (q - d(a_1), q)$
$a_2 \in \mathcal{A}^\alpha \setminus \mathcal{A}(f)$
$a_2 = (q - \alpha, q)$
	a_1 is variable
	a_2 is variable
		`vars0(q+d)=o`
		with `d=d_right(q)`
		`varsA(q-`α`)=a`
			`forbid01(o,a)`

Table 4: Construction of the uniqueness term.

>Code: The edges enforcing uniqueness are built in `Match::build_uniqueness_LR` (first row of Table 4) and `Match::build_uniqueness_RL` (second row). To implement the infinite term, it is possible to use the function `add_term2` with a large E_{01}. Alas, looking at (17) or at Figure 4(b), the risk of overflow is plain to see. It is especially serious in our implementation since capacities are encoded with short integers. For that reason, we haved designed a dedicated function called `Energy::forbid01`. This function does not follow the pattern of Figure 4(b), but instead it merely puts the maximum representable short integer for the weight of the edge (see Figure 19 in Appendix B). Nonetheless, this alternative still represents the pairwise energy under consideration with the advantage of completely eliminating the risk of overflow in terminal edges (that is to say, edges involving s or t). As for non-terminal edges, overflow is not an issue because those occurring in the uniqueness term do not appear in other energy terms, so that their weight is set up once for all. This is not the case for weights of terminal edges, which are incremented several times during the graph construction. Notice there are at most $12n$ non-terminal edges in the graph, with n the number of pixels in each image. Indeed, for each pixel we can get:

- 2 pixels in half neighborhood, leading each to 2 smoothness terms. Total: 4.
- 1 left-right and 1 right-left uniqueness terms. Total: 2.

For a technical reason related to the implementation of the max-flow algorithm, every edge is duplicated into two copies reversed of one another. Therefore, we get $2 \times (4 + 2) = 12$ edges, excluding terminal edges. To avoid costly memory reallocations of the arrays of vertices and edges in the code, the maximum memory is allocated in method `Match::ExpansionMove`, that is, $2 \times \#\mathcal{L}$ vertices and $12 \times \#\mathcal{L}$ edges.

4.1.2 Minimization of the Graph Cut

Once the graph is constructed, we launch the max-flow algorithm to get the cut of minimal cost of the graph by maximizing the flow. By definition, if the energy $E_{f,\alpha}$ is represented by the graph $\mathcal{G}$ constructed in the previous section, then, for any cut $(\mathcal{V}^s, \mathcal{V}^t)$ of $\mathcal{G}$,

$$E_{f,\alpha}(g_\alpha) = c_{\mathcal{G}}(\mathcal{V}^s, \mathcal{V}^t) \qquad \text{with } g_\alpha(a) = \begin{cases} 0 & \text{if } v_a \in \mathcal{V}^s \\ 1 & \text{if } v_a \in \mathcal{V}^t \end{cases} \qquad \text{for } a \in \mathcal{A}^\alpha \cup \mathcal{A}^\circ, \tag{34}$$

where v_a denotes the node associated with the assignment a. Such a notation is valid since there are equal numbers of nodes in the graph and of variables in $\mathrm{E}_{f,\alpha}$. Going back to the associated configuration f' (thanks to (21)), it may be reformulated in terms of active/nonactive assignments:

$$\mathrm{E}(f') = c_{\mathcal{G}}(\mathcal{V}^s, \mathcal{V}^t) \qquad \text{with } f'(a) = \begin{cases} 0 & \text{if } g_\alpha(a) = 0 \text{ and } a \in \mathcal{A}^\alpha \setminus \mathcal{A}(f) \\ 1 & \text{if } g_\alpha(a) = 1 \text{ and } a \in \mathcal{A}^\alpha \setminus \mathcal{A}(f) \\ 1 & \text{if } g_\alpha(a) = 0 \text{ and } a \in \mathcal{A}^\circ \\ 0 & \text{if } g_\alpha(a) = 1 \text{ and } a \in \mathcal{A}^\circ \end{cases}. \tag{35}$$

Since the graph exactly represents the energy, the minimal cost of a cut is the value of the lowest energy of the α-expansion moves of f. Hence, if the value is strictly lower than the energy `oldE` of the initial configuration f, the expansion move has decreased the energy and one keeps the resulting configuration f^*.

4.1.3 Recovering the Optimal Configuration and Updating the Disparity

After finding the optimal cut of the graph, nodes in `varsA` and `vars0` are set to values `VAR_ALPHA`, `VAR_ABSENT`, 0 and 1, indicating the activity of the node, see (35).

Updating the configuration By construction, nonvariable nodes keep their initial state, that is, $f^*(a) = f(a)$ for the associated assignment. Hence, for each pixel p in the left image, if `varsA(p)=vars0(p)=VAR_ALPHA`, then the assignment $a = (p, p + \alpha)$ remains active. Assignments not in $\mathcal{A}^\alpha \cup \mathcal{A}^\circ$ remain inactive. Let us now consider the variable assignments.

In the code, the function `get_var` returns the value $g_\alpha(\mathtt{a})$ of a binary variable. If the nodes `varsA(p)` and/or `vars0(p)` associated to the pixel p are variable, then, according to (35)

- if `get_var(vars0(p))=1`, then the assignment $a = (p, p + d_f(p))$ becomes inactive;
- if `get_var(varsA(p))=1`, then the assignment $a = (p, p + \alpha)$ becomes active;
- the other assignments keep their original state.

Updating the disparity map This allows the software to update the disparity map `d_left`, as below:

- if `get_var(vars0(p))=1`, then `d_left(p)=OCCLUDED` (the pixel p becomes occluded because it lost its former disparity, unless the following rule gives it a new disparity α);
- if `get_var(varsA(p))=1`, then `d_left(p)=`α (the pixel p has adopted the disparity α).

The remaining pixels keep their former disparity (or remain occluded).

>Code: Update of the disparity map `d_left` by examination of the value of variables is done in `Match::update_disparity`. The redundant inverse disparity map `d_right`, from right to left image, is also updated, which is used in the next α-expansion move to speed up the right to left uniqueness constraint construction (last row of Table 4).

4.2 Parameters

Let us recall that the data penalty for an assignment a is $D'(a)$, where $D'(a) = D(a) - K$ if a is active, 0 otherwise. Thus, only active assignments contribute to the energy and this contribution can be either negative or nonnegative.

If K is too large (larger than every possible value $D(a)$), the data penalty for any active assignment is negative. Then each active assignment makes the energy decrease, so it favors configurations with maximal active assignments. On the contrary, if K is too small (smaller than every possible value $D(a)$) the data penalty for active assignments is nonnegative. So occlusions are favored.

Kolmogorov and Zabih proposed a heuristic to choose automatically a suitable value of K avoiding the above mentioned situations. K is chosen such that for each pixel, only 25% of the possible assignments give negative (so advantageous) penalties on average.

Algorithm 2 gives the details of the automatic computation of K. In line 2, we just consider pixels p such that $p + \mathrm{I}_{\mathrm{disp}} \subset \mathcal{R}$, which amounts to ignore some pixels close to one image border.

Algorithm 2: Automatic choice of the parameters

1 Set k to one fourth of the number of its possible assignments if larger than 3, and 3 otherwise;
2 **foreach** *pixel* p `// but not pixels` p `close to border (see text)`
3 **do**
4 Compute $C(p)$ as the k-th smallest value of all the $D(a)$ values for assignments a involving the pixel p;
5 Set K to the average of all $C(p)$ values.

>Code: K is computed in `statistics.cpp`.

Since λ and K should be balanced, λ is chosen to be proportional to K. The user can also tune the parameters in order to enhance the results. Note that the automatic estimation takes as input the disparity range given by the user. See Section 6.2 concerning the influence of these settings.

By default $\lambda = K/5$. The `Graph` class is a template and can accept weights of different types, but the `Energy` class uses integer weights (specifically type `short int`). Just passing the nearest integer to $K/5$ may be too imprecise. Therefore the parameters are approximated as fractions: the smallest denominator between 1 and 16 minimizing

$$N = \arg \min_{i=1,\cdots,16} \left| \frac{[i \cdot K]}{i \cdot K} - 1 \right| + \left| \frac{[i \cdot \lambda_1]}{i \cdot \lambda_1} - 1 \right| + \left| \frac{[i \cdot \lambda_2]}{i \cdot \lambda_2} - 1 \right| \tag{36}$$

with $[x]$ the nearest integer to x. The minimized term is the sum of relative errors in approximating K, λ_1 and λ_2 as fractions with same denominator i. If any of the three parameters is 0, its contribution is ignored in the sum. Finally, all parameters K, λ_1 and λ_2 are recomputed as fractions with this denominator N. Since only the relative scales of the parameters are important, we can just use the *numerators* and replace the data+occlusion term by:

$$D'(a) = N \times D(a) - K', \tag{37}$$

with K'/N the fraction approximating K.

The constraint $N \leq 16$ prevents a possible overflow. Concerning the data+occlusion term, $ND(a)$ being positive, it is enough to check the term does not exceed the maximum short integer, normally $2^{15} - 1$. Indeed, the worst case is when $D(a)$ is based on the L^2 norm. In that case, since according to (5)

$$D(a) \leq \texttt{CUTOFF}^2 = 30^2 < (2^5)^2 = 2^{10}, \tag{38}$$

$N \leq 2^4$ ensures that $ND(a) < 2^{14}$. The positivity of K' prevents $D'(a)$ from overflowing by excess. Besides the data+occlusion terms, there might be up to $4 \times E_{01} \leq 4 \times N\lambda_1$ (one per neighbor) in edge weight to the terminal node t contributed by the smoothness term, see Figure 4(b). This must not exceed 2^{14}, which amounts to $\lambda_1 < 2^8$. Since $\lambda_1 = 3\lambda$, this makes an upper bound of about 85 for λ. The online demo restricts λ to at most 50.

>Code: The parameters λ_1, λ_2 and the constant 8 occurring in (8) can be independently chosen when running the program with the arguments `--lambda1`, `--lambda2` and `--threshold` (or equivalently `-t`). However the demo does not offer such flexibility: it sets λ_1 and λ_2 with respect to λ and sets the edge threshold to 8.

4.3 Birchfield and Tomasi's Dissimilarity Measure

Let us recall the formulation of this measure. The BT dissimilarity measure [1] of a pair of pixels (p, q) is the distance from $I_1(p)$ to interval $\big[I_2^{\min}(q), I_2^{\max}(q)\big]$:

$$D_{\mathrm{BT}}(p,q) = \max\{0, I_1(p) - I_2^{\max}(q), I_2^{\min}(q) - I_1(p)\}, \tag{39}$$

where $I_2^{\max}(q)$ and $I_2^{\min}(q)$ are resp. the larger and the smaller values on the one-pixel-large-neighborhood (in both directions x and y) centered on q of $\tilde{I}_2$, with $\tilde{I}_2$ the bilinear interpolation of intensity I_2:

$$I_2^{\max}(q) := \max_r \left\{ \tilde{I}_2\left(q + \frac{1}{2}r\right) := \frac{1}{2}\big(I_2(q) + I_2(q+r)\big) \right\} \tag{40}$$

$$I_2^{\min}(q) := \min_r \left\{ \tilde{I}_2\left(q + \frac{1}{2}r\right) := \frac{1}{2}\big(I_2(q) + I_2(q+r)\big) \right\}, \tag{41}$$

with $r \in \big\{(0,0),(1,0),(-1,0),(0,1),(0,-1)\big\}$. The last two vectors have been added by Kolmogorov and Zabih and did not occur in the original BT measure: this addition could compensate a slight error in the epipolar rectification. That is, if $I_1(p) \in \big[I_2^{\min}(q), I_2^{\max}(q)\big]$, then there exists a q' in the subpixel neighborhood of q ($[q_x - 1/2, q_x + 1/2] \times [q_y - 1/2, q_y + 1/2]$) such that $I_1(p) = \tilde{I}_2(q')$. Hence, the apparent dissimilarity between p and q can be a sampling effect and should not be taken into account.

A trimmed and symmetric version of D_{BT} is used in the code, as the smaller of the distance of $I_1(p)$ to interval $[I_2^{\min}(q), I_2^{\max}(q)]$ and of the distance of $I_2(q)$ to interval $[I_1^{\min}(p), I_1^{\max}(p)]$:

$$\begin{aligned} D(p,q) = T(\min(\ & \max\{0, I_1(p) - I_2^{\max}(q), I_2^{\min}(q) - I_1(p)\}, \\ & \max\{0, I_2(q) - I_1^{\max}(p), I_1^{\min}(p) - I_2(q)\})), \end{aligned} \tag{42}$$

using the trim function T of (5).

>Code: This dissimilarity is computed in `data.cpp` (in functions `SubPixel`, `SubPixelColor`, `Match::data_penalty_gray` and `Match::data_penalty_color`).

5 About the Online Demo

In order to reduce the computational time, we decided to slice the images in six strips, running the program in parallel. Experimental results have shown only few differences between running the program on the entire canvas or running it on the cropped images and then merging them (see Figure 6). The strips overlap by 12px. The last strip must not be too narrow, so its height is remainder + quotient (Figure 5). We compute the parameters K and λ on the whole image and then

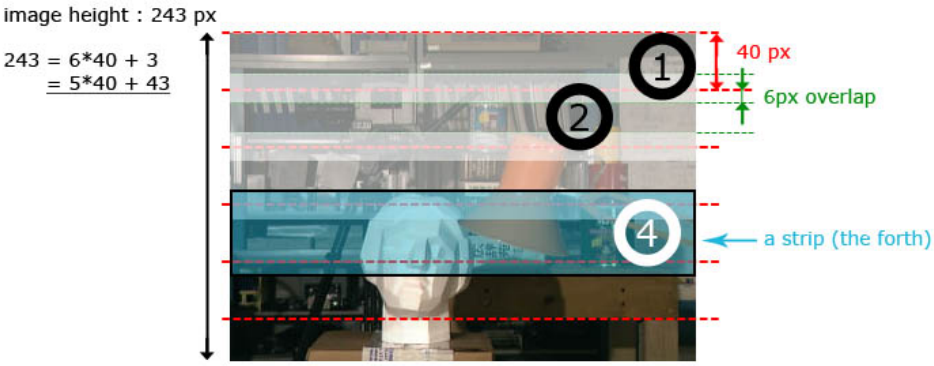
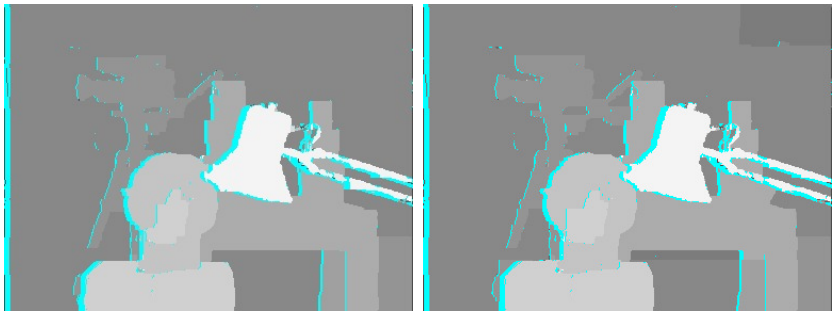


Figure 5: The six strips of the sliced image.

use them on each strip, instead of letting the program choose different values of parameters for each strip.

For comparison, Figure 6 shows the results of computing the disparity map on the entire image vs. computing it on sliced images. The two are quite similar. However, one can observe some impact



(a) Without cutting in six strips

(b) With cutting in six strips

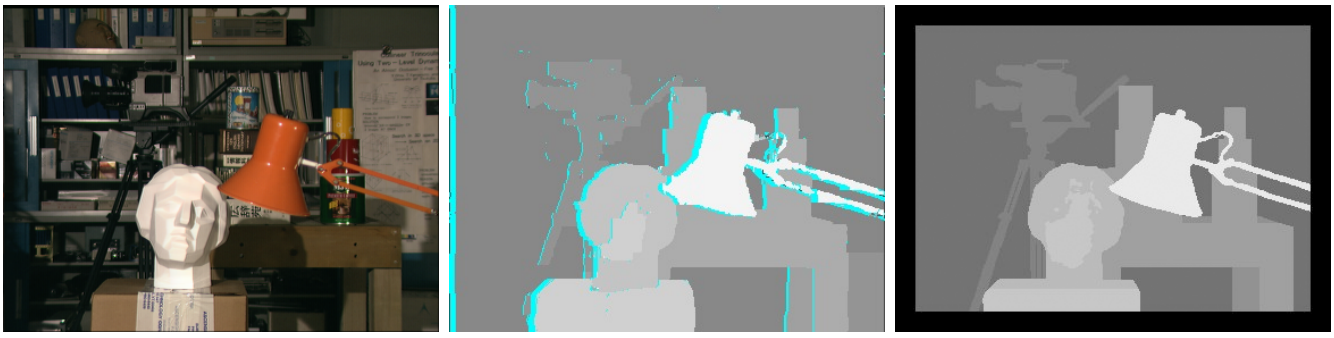
Figure 6: Experimental results on the entire image and the sliced images.

on the regularity at the junctions of two stripes, e.g., on the top of the right table leg. In such cases, it may not be worth assigning the correct disparity to the background and paying the smoothness term. It is then more advantageous to pay the data term.

6 Examples

6.1 Experimental Results

Tsukuba (Figure 7) The algorithm works fairly well on this image: we see almost no difference with the ground truth. Moreover occluded pixels (in cyan) are correctly found. Note though that some pixels (very few) are labeled occluded by mistake. This image typically shows the superiority of complex global algorithms over simple local methods such as block matching, especially in occlusion detection. Because the matches do not involve patches, there is no fattening effect, as it is often observed in (even advanced) local methods.



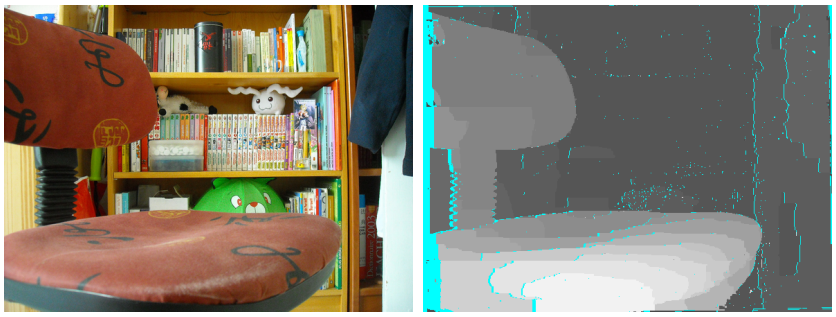
(a) Tsukuba (left image)

(b) Result of algorithm run without cutting the image into several strips

(c) Groundtruth (black pixels mean "no data")

Figure 7: Results of the algorithm on the *Tsukuba* pair.

Chair (Figure 8) The result of the algorithm is also good. The pair is actually an easy one, since it avoids most of the difficulties encountered in stereovision (lots of textures, a few reflections; compare with the next example). However, one can see on this example that the quantification of the disparity (the algorithm can only produce pixel disparity map) is a limitation of the method. Indeed the disparities on the chair (e.g.) are expected to be smoother than the estimated ones. Notice the thin bands of occluded pixels at the left of disparity level sets on the seat. In a way this is



(a) Chair (left image)

(b) Result of algorithm

Figure 8: Results of the algorithm on the *Chair* pair.

disturbing because these erroneous occlusions display a similarity with the real ones. To understand this somehow troubling phenomenon, consider for example a mainly uniform patch in the image whose real disparity map is slanted, as in Figure 9. If $a = b + 1$, the combined effect of quantization and the uniqueness constraint compell an occlusion to occur at their interface, otherwise the two pixels would arrive on the same pixel in the right image. Assuming the part with label b is in front, the last a in each line has to be occluded.

Bottle (Figure 10) The result is not very satisfactory. The shapes of the bottle and the speaker are correct, but many matches are not. The reason is mainly that the scene is very difficult for a stereo algorithm: first, the photos were taken from two distant points of view. Hence many pixels are

a	a	b	b	b	b	b	b	b	b
a	a	b	b	b	b	b	b	b	b
a	a	a	a	a	b	b	b	b	b
a	a	a	a	a	a	b	b	b	b
a	a	a	a	a	a	b	b	b	b
a	a	a	a	a	a	a	a	b	b

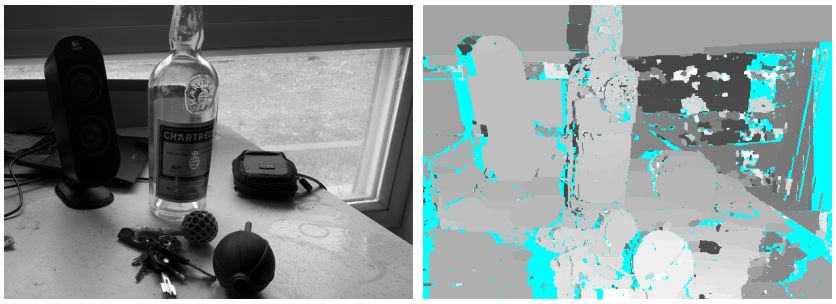
(a) True (but quantized) disparity map

a	o	b	b	b	b	b	b	b	b
a	o	b	b	b	b	b	b	b	b
a	a	a	a	o	b	b	b	b	b
a	a	a	a	a	o	b	b	b	b
a	a	a	a	a	o	b	b	b	b
a	a	a	a	a	a	a	o	b	b

(b) Computed disparity map

Figure 9: Combined effect of quantification and uniqueness constraint on computed disparity of a slanted roughly uniform surface. We assume $a = b + 1$ and o is the occlusion label.

occluded. Then the window at the back is semi transparent and the table provides large reflections (water on the table, bottle). Indeed, since a reflection reveals the image of points that are somewhere else (e.g., behind the camera), the actual depth of the pixel is not the one of the reflecting surface. That is why one can observe some kind of holes (e.g., besides the ball on the foreground). Moreover there is grass behind the window. Because it changes a lot with different points of view, vegetation also generally poses a problem for stereo algorithms. Yet parts of the image such as the middle left (around the loudspeaker) seem easy to match (small disparity range, no reflection) but the result is not really good anyway. This is due to the fact that the matches are global, so when a large number of matches are wrong, they can perturb the others (even the easiest ones).



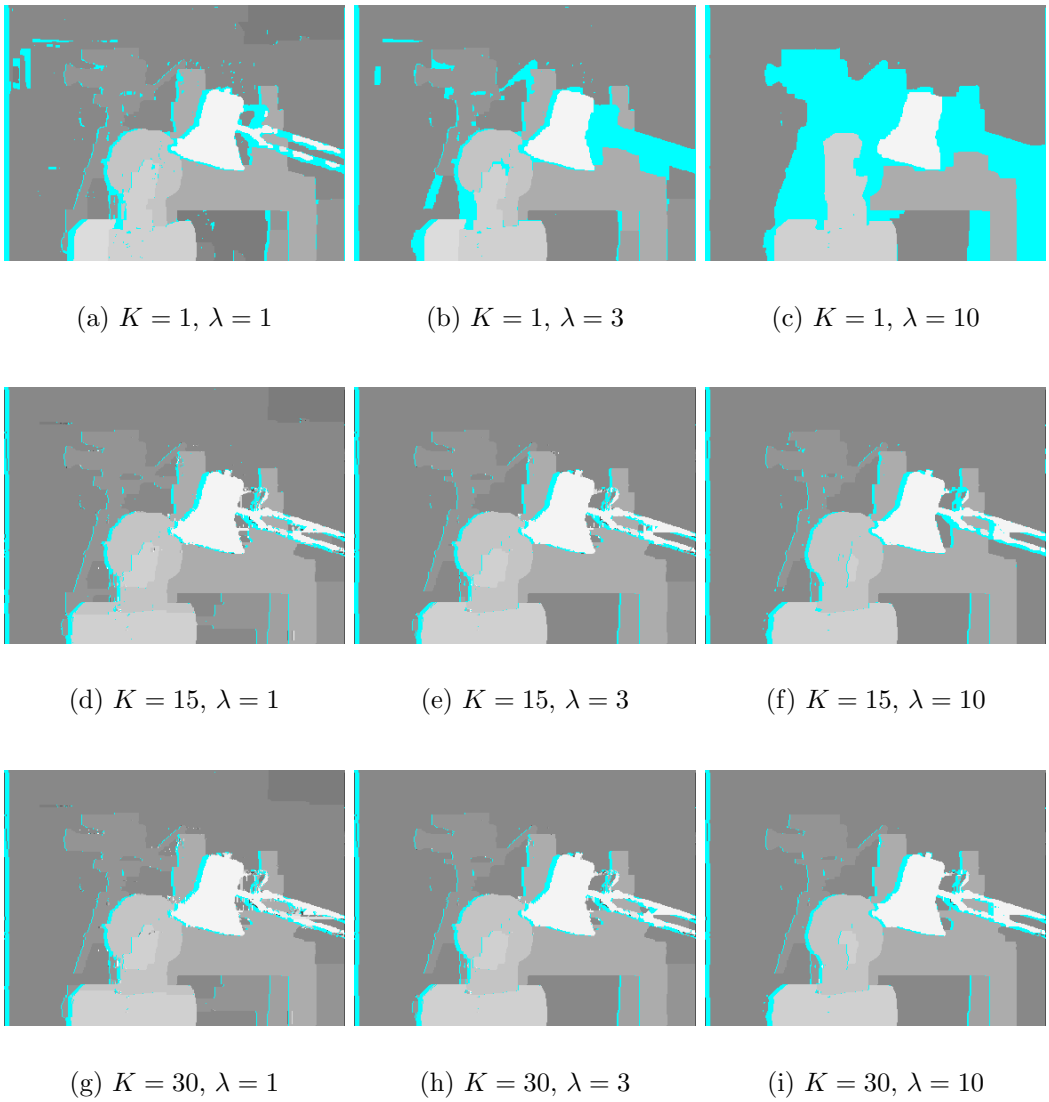
(a) Bottle (b) Result with algorithm

Figure 10: Results of the algorithm on the *Bottle* pair.

6.2 Influence of Parameters

We have run the algorithm with several different values of parameters λ and K (Figure 11). We are thus able to see if Kolmogorov's heuristic is relevant. Let us recall that λ corresponds to the smoothness penalty (the bigger λ, the smoother the disparity map). K is the occlusion penalty (the bigger K, the bigger the number of matches and the less the number of occluded pixels).

We can first observe that there is a range where the results are quite stable ($(K, \lambda) \in [15, 30] \times [3, 10]$). Thus, there is no need to accurately select the parameters to get correct results. However, with $K = 15$ for instance, we are able to see the influence of the smoothness parameter: the bigger the parameter, the smoother the disparity map. This is remarkable at the top right corner of the



(a) $K = 1$, $\lambda = 1$ (b) $K = 1$, $\lambda = 3$ (c) $K = 1$, $\lambda = 10$

(d) $K = 15$, $\lambda = 1$ (e) $K = 15$, $\lambda = 3$ (f) $K = 15$, $\lambda = 10$

(g) $K = 30$, $\lambda = 1$ (h) $K = 30$, $\lambda = 3$ (i) $K = 30$, $\lambda = 10$

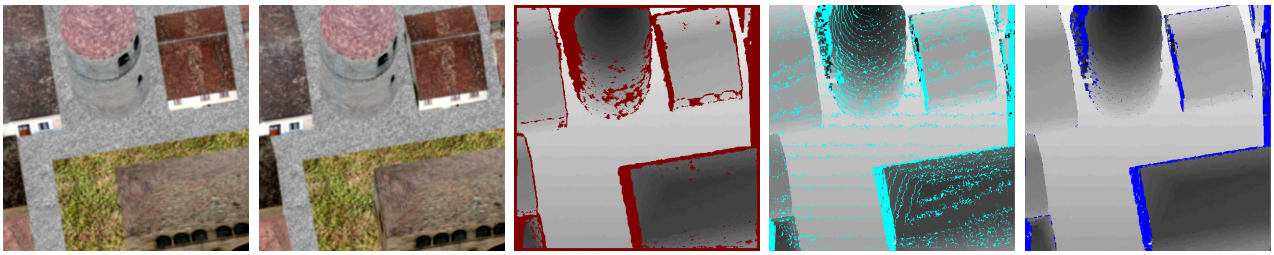
Figure 11: For Tsukuba image, automatic paramaters are: $K = 15$ and $\lambda = 3$.

image (d). The corner is actually a uniform (textureless) region, which means that several disparities are acceptable. Then when the smoothness is favored, it is advantageous to give to this area the same disparity as the whole background of the scene. One can also see that at the bottom right corner in image (e), the disparity is less smooth than in image (f).

Then, when K is too small (e.g., $K = 1$), only reliable matches are selected, since it is almost costless not to do uncertain matches. When λ increases, there is less constant-disparity-component, to minimize the topological frontiers between these components (every disparity jump is expensive). When K is too big, we observe more incorrect matches (e.g., in the lamp area). This is due to the fact that by setting a large occlusion parameter, we encourage the program to maximize the matches, even if they are not good. Hence, the disparity map often seems to be noisy.

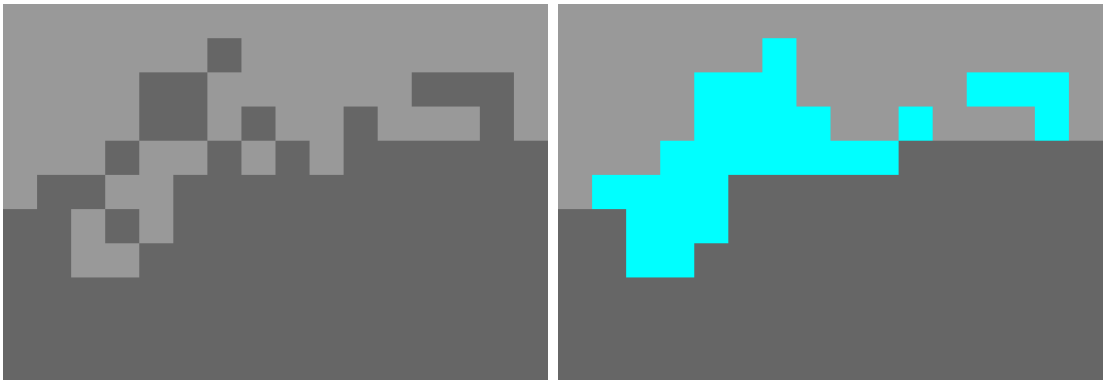
7 Future work

The algorithm described in this article can be modified so that one can specify one's own disparity range for each pixel of the images. There are many possible applications of this new algorithm.



(a) Image pair (b) Local stereo matching (c) After the filling process (d) Union of both

Figure 12: The initial disparity map was obtained by a block-matching method where the red pixels have been rejected by several filters (left-right/right-left consistency check, self similarity and flat patch detection). After the filling process by graph cuts, "occluded" pixels were marked in cyan. After replacing the "occluded" pixels that had an initial disparity, we get a denser map, with final rejected pixels in blue.

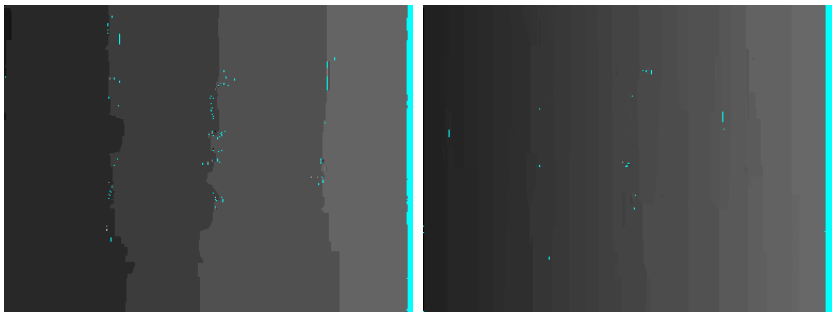


(a) Local stereo matching detail. (b) After graph cuts "filling".

Figure 13: A large occlusion area after graph cut filling of the disparity map issued from block matching. (a) Detail of Figure 12(b). (b) Detail of Figure 12(c). (Contrast enhanced for better visualization)

Filling process (Figure 12) Many local stereo matching methods add a rejection step which aims at removing false matches. The resulting disparity map needs to be filled. A global method can be used for this task. This may be done by setting to each non-rejected pixel p the singleton-interval $\{d^\circ(p)\}$ where d° denotes the initial disparity map. Every rejected pixel is then set the initial disparity range $\mathrm{I}_{\mathrm{disp}}$. In this case every non-rejected pixel may keep its initial disparity or become occluded. In the experiment of Figure 12 we let pixels with initial disparities become occluded, therefore we do not really "fill" the map. However, we could also prevent this so that the resulting map is at least as dense as the initial map. A simpler solution is to reset to $d^\circ(p)$ the pixels not rejected by the local method, as in Figure 12(c).

It is surprising that so many pixels are occluded at boundaries of disparity level sets in Figure 12(c). This is due to the classical adhesion effect (aka fattening effect) of block matching, combined with the trend toward regularity of the graph cuts. This can be observed in the detail displayed in Figure 13. The irregular pattern issued from block matching has a lower energy when occluded by graph cuts.



(a) Scale 0 (b) Scale 4

Figure 14: Synthetic disparity ramp applied to left Tsukuba image: $d(p) = a\,p_x + b$, with $a = 0.01$ and $b = 1.5$. Left: pixel accurate recovered disparity map. Right: $1/16^{\text{th}}$-pixel accurate recovered disparity map through multiscale refinement. For such a disparity map subpixel precision is needed to avoid quantification effects.

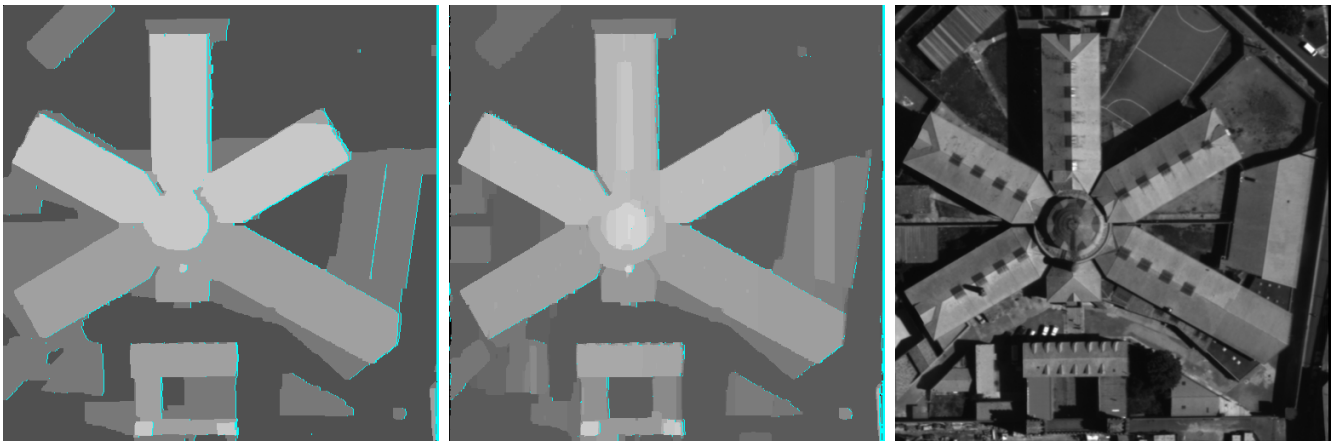
Scale	0	1	2	3	4
Tsukuba - Multiscale	30 s	98 s	310 s	1252 s	5384 s
Tsukuba - Direct	30 s	126 s	557 s	2405 s	10629 s
Venus - Multiscale	61 s	154 s	712 s	2541 s	
Venus - Direct	61 s	276 s	1386 s	6806 s	
Prison - Multiscale	22 s	68 s	420 s	1759 s	6537 s
Prison - Direct	22 s	116 s	557 s	2969 s	15183 s

Table 5: Computation times for subpixel accurate estimation. The multiscale approach indicates cumulated times.

Ground Control Points (GCP) In a similar way one can take advantage of reliable information on some matches. The disparity range of such pixels is set to the singleton-interval $\{d^\circ(p)\}$ where d° denotes the reliable disparity and every other pixel is set to the whole disparity range I_{disp}.

Subpixel refinement thanks to a multiscale approach (Figures 14, 15) The original algorithm described here only produces pixel accurate disparity map. For instance to get a subpixel refinement with a precision of 0.5 pixel one needs upsampling the images by a factor 2. Since the disparity range doubles, such an operation will double the time of each iteration and the number of nodes in the graphs. To avoid a complexity explosion one may proceed by adopting a multiscale approach. That is, using the disparity map estimation at a previous scale to refine the result at the current scale. At each step the images are upsampled by a factor 2. The disparity range is then assigned to the previous disparity estimation for every non occluded pixel (relaxed of half a pixel at each boundary, then multiplied by 2) and every occluded pixel is reassigned the whole initial disparity range I_{disp} multiplied by the current scale factor. Hence, at scale n, for each pixel in the original image, there is a range of 2^{n-1} interpolated pixels, so 2^{n-1} disparity values. We obtained the disparity of the pixel in the original image by computing the median value of the 2^{n-1} disparity values (and the pixel is declared occluded if at least half of the interpolated pixels are occluded).

Table 5 shows that the multiscale approach is at least twice as fast as the direct approach. So far experimental results have shown few differences in terms of quality between the multiscale approach and the direct approach (Figure 16).



(a) Scale 0 (b) Scale 4 (c) Left image

Figure 15: *Prison* pair. The initial disparity range is $[0, 6]$. Here the disparity range is so narrow that a pixel accurate result is not satisfactory. The refinement allows to get better results both on the ground (see lower left image area) and on the building roof.

A Additivity of the Graph-Representability

We show in this section that a sum of two graph-representable functions is graph-representable. At the same time, it gives a constructive procedure to represent the sum of functions of binary variables by a graph. Let E be a graph-representable function of n binary variables and E$'$ be a graph-representable function of m variables. We assume that E and E$'$ have $0 \leq k \leq m, n$ variables in common, such that we can write

$$\mathrm{E}(x_1, \ldots, x_k, x_{k+1}, \ldots, x_n) \qquad \text{and} \qquad \mathrm{E}'(x_1, \ldots, x_k, x_{n+1}, \ldots, x_{n+m-k}), \tag{43}$$

where $\{x_1, \ldots, x_k\}$ denote the k variables in common. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (resp. $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$) be a graph that represents the function E (resp. E$'$) and let v_i the vertex associated with the variable x_i for each $i = 1, \ldots, n + m - k$. The source and sink are chosen to be the same in both graphs but, apart from $\{v_1, \ldots, v_k\}$, vertices in graphs $\mathcal{G}$ and $\mathcal{G}'$ are distinct. Now consider the graph $\mathcal{G}'' = (\mathcal{V}'', \mathcal{E}'')$ defined as follows:

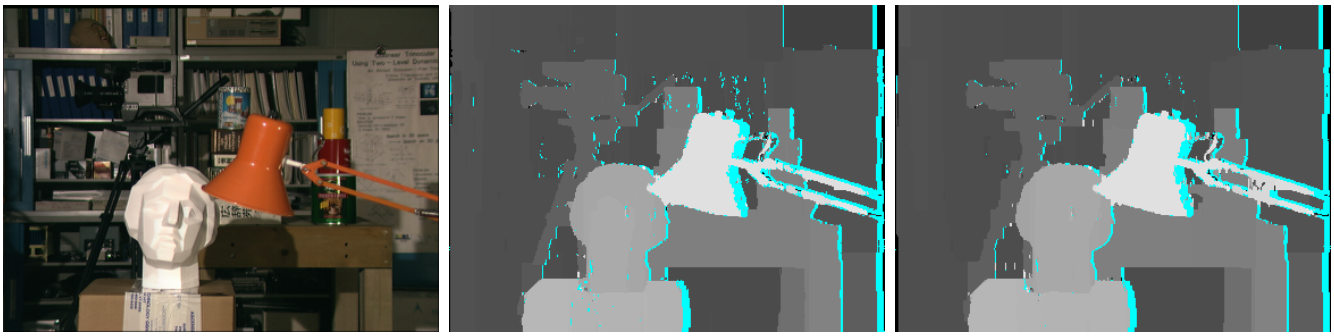
$$\mathcal{V}'' = \mathcal{V} \cup \mathcal{V}' \qquad \text{and} \qquad \mathcal{E}'' = \mathcal{E} \cup \mathcal{E}', \tag{44}$$

and such that, if $e \in \mathcal{E} \cap \mathcal{E}'$, then its capacity is $c_{\mathcal{G}}(e) + c_{\mathcal{G}'}(e)$, and otherwise, it keeps its initial capacity. Let us prove that this graph represents the function $\mathrm{E}'' = \mathrm{E} + \mathrm{E}'$ of $(n + m - k)$ binary variables. Fix $x'' = (x_1, \ldots, x_{n+m-k})$ a configuration and let $(\mathcal{V}''^s, \mathcal{V}''^t)$ be a cut of the graph $\mathcal{G}''$, such that $v_i \in \mathcal{V}''^s$ if $x_i = 0$ and $v_i \in \mathcal{V}''^t$ if $x_i = 1$. Such a cut will be referred to as an x''-cut. Similarly, we pose $x = (x_1, \cdots, x_n)$ and $x' = (x_1, \cdots, x_k, x_{n+1}, \cdots, x_{n+m-k})$ and define x-cuts of $\mathcal{G}$ and x'-cuts of $\mathcal{G}'$. The cost of $(\mathcal{V}''^s, \mathcal{V}''^t)$ is given by

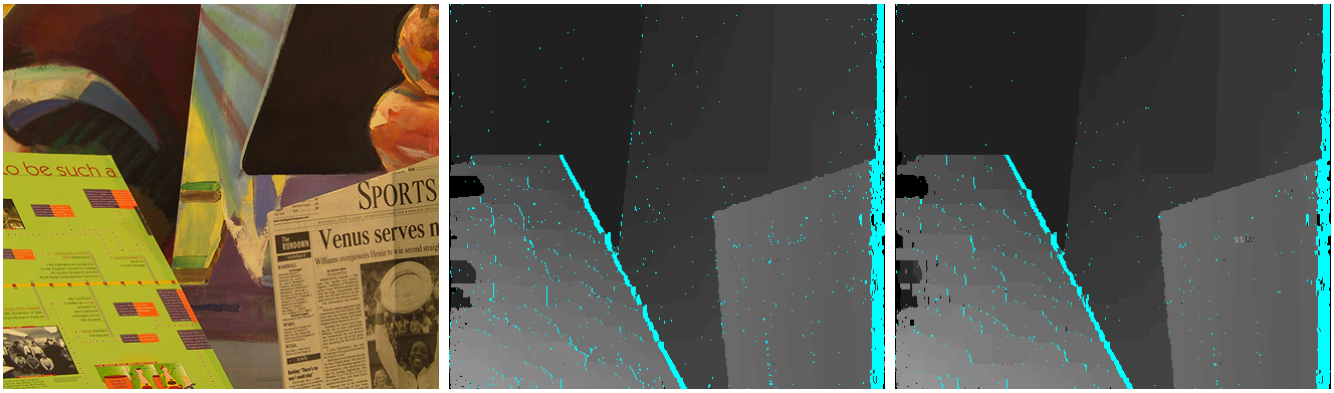
$$c_{\mathcal{G}''}(\mathcal{V}''^s, \mathcal{V}''^t) = \sum_{\substack{(u,v) \in \mathcal{E}'' \\ u \in \mathcal{V}''^s, v \in \mathcal{V}''^t}} c_{\mathcal{G}''}(u, v). \tag{45}$$

Since we have the disjoint union $\mathcal{E}'' = (\mathcal{E} \setminus \mathcal{E}') \sqcup (\mathcal{E}' \setminus \mathcal{E}) \sqcup (\mathcal{E} \cap \mathcal{E}')$, we decompose

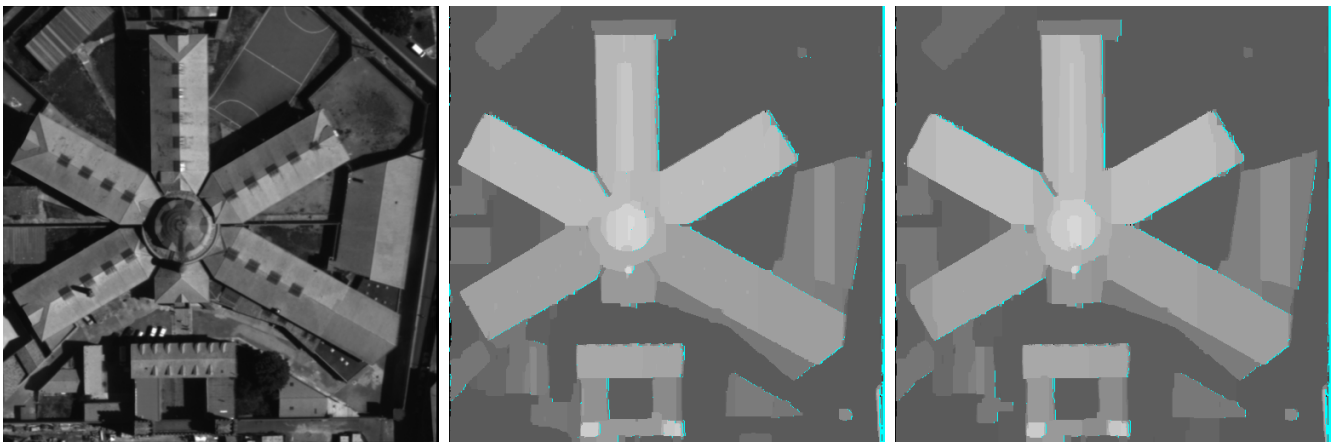
$$c_{\mathcal{G}''}(\mathcal{V}''^s, \mathcal{V}''^t) = \sum_{\substack{(u,v) \in \mathcal{E} \setminus \mathcal{E}' \\ u \in \mathcal{V}''^s, v \in \mathcal{V}''^t}} c_{\mathcal{G}''}(u, v) + \sum_{\substack{(u,v) \in \mathcal{E}' \setminus \mathcal{E} \\ u \in \mathcal{V}''^s, v \in \mathcal{V}''^t}} c_{\mathcal{G}''}(u, v) + \sum_{\substack{(u,v) \in \mathcal{E} \cap \mathcal{E}' \\ u \in \mathcal{V}''^s, v \in \mathcal{V}''^t}} c_{\mathcal{G}''}(u, v). \tag{46}$$



(a) Tsukuba (b) Scale 4 (multiscale) (c) Scale 4 (direct)



(d) Venus (e) Scale 3 (multiscale) (f) Scale 3 (direct)



(g) Prison (h) Scale 4 (multiscale) (i) Scale 4 (direct)

Figure 16: Multiscale approach vs. direct approach in subpixel accurate estimation.

By definition of the weights in $\mathcal{G}''$,

$$c_{\mathcal{G}''}(\mathcal{V}''^s, \mathcal{V}''^t) = \sum_{\substack{(u,v)\in\mathcal{E}\setminus\mathcal{E}' \\ u\in\mathcal{V}''^s, v\in\mathcal{V}''^t}} c_{\mathcal{G}}(u,v) + \sum_{\substack{(u,v)\in\mathcal{E}'\setminus\mathcal{E} \\ u\in\mathcal{V}''^s, v\in\mathcal{V}''^t}} c_{\mathcal{G}'}(u,v) + \sum_{\substack{(u,v)\in\mathcal{E}\cap\mathcal{E}' \\ u\in\mathcal{V}''^s, v\in\mathcal{V}''^t}} \big[c_{\mathcal{G}}(u,v) + c_{\mathcal{G}'}(u,v)\big]. \tag{47}$$

If we set $\mathcal{V}^s = \mathcal{V}''^s \cap \mathcal{V}$ and $\mathcal{V}^t = \mathcal{V}''^t \cap \mathcal{V}$, since $(u,v) \in \mathcal{E}$ only if $u, v \in \mathcal{V}$, then

$$\sum_{\substack{(u,v)\in\mathcal{E}\setminus\mathcal{E}' \\ u\in\mathcal{V}''^s, v\in\mathcal{V}''^t}} c_{\mathcal{G}}(u,v) + \sum_{\substack{(u,v)\in\mathcal{E}\cap\mathcal{E}' \\ u\in\mathcal{V}''^s, v\in\mathcal{V}''^t}} c_{\mathcal{G}}(u,v) = c_{\mathcal{G}}(\mathcal{V}^s, \mathcal{V}^t), \tag{48}$$

where $(\mathcal{V}^s, \mathcal{V}^t)$ is an x-cut of the graph $\mathcal{G}$. Remember that only nodes $\{v_1, \ldots, v_k\}$ are both in $\mathcal{V}$ and in $\mathcal{V}'$, so edges in $\mathcal{E} \cap \mathcal{E}'$ only connect such nodes. Thus, the quantity

$$\sum_{\substack{(u,v)\in\mathcal{E}\cap\mathcal{E}' \\ u\in\mathcal{V}^s, v\in\mathcal{V}^t}} c_{\mathcal{G}}(u,v) = \sum_{\substack{(v_i,v_j)\in\mathcal{E}\cap\mathcal{E}' \\ x_i=0, x_j=1}} c_{\mathcal{G}}(v_i,v_j) \tag{49}$$

does not depend on the selected x-cut $(\mathcal{V}^s, \mathcal{V}^t)$. Hence, its cost may be written

$$c_{\mathcal{G}}(\mathcal{V}^s, \mathcal{V}^t) = \sum_{\substack{(u,v)\in\mathcal{E}\setminus\mathcal{E}' \\ u\in\mathcal{V}^s, v\in\mathcal{V}^t}} c_{\mathcal{G}}(u,v) + \sum_{\substack{(v_i,v_j)\in\mathcal{E}\cap\mathcal{E}' \\ x_i=0, x_j=1}} c_{\mathcal{G}}(v_i,v_j). \tag{50}$$

Similarly, we prove that

$$c_{\mathcal{G}'}(\mathcal{V}'^s, \mathcal{V}'^t) = \sum_{\substack{(u,v)\in\mathcal{E}\setminus\mathcal{E}' \\ u\in\mathcal{V}'^s, v\in\mathcal{V}'^t}} c_{\mathcal{G}'}(u,v) + \sum_{\substack{(v_i,v_j)\in\mathcal{E}'\cap\mathcal{E} \\ x_i=0, x_j=1}} c_{\mathcal{G}'}(v_i,v_j), \tag{51}$$

with $\mathcal{V}'^s = \mathcal{V}''^s \cap \mathcal{V}'$ and $\mathcal{V}'^t = \mathcal{V}''^t \cap \mathcal{V}'$, which defines an x'-cut of $\mathcal{G}'$. Since any pair of an x-cut $(\mathcal{V}^s, \mathcal{V}^t)$ and an x'-cut $(\mathcal{V}'^s, \mathcal{V}'^t)$ yields an x''-cut $(\mathcal{V}''^s, \mathcal{V}''^t)$ of the graph $\mathcal{G}''$ by

$$\mathcal{V}''^s = \mathcal{V}^s \cup \mathcal{V}'^s \qquad \text{and} \qquad \mathcal{V}''^t = \mathcal{V}^t \cup \mathcal{V}'^t \tag{52}$$

we eventually obtain

$$\begin{aligned} \min_{(\mathcal{V}''^s, \mathcal{V}''^t)\ x''\text{-cut}} c_{\mathcal{G}''}(\mathcal{V}''^s, \mathcal{V}''^t) = \min_{(\mathcal{V}''^s, \mathcal{V}''^t)\ x''\text{-cut}} & \sum_{\substack{(u,v)\in\mathcal{E}\setminus\mathcal{E}' \\ u\in\mathcal{V}^s, v\in\mathcal{V}^t}} c_{\mathcal{G}}(u,v) + \sum_{\substack{(v_i,v_j)\in\mathcal{E}\cap\mathcal{E}' \\ x_i=0, x_j=1}} c_{\mathcal{G}}(v_i,v_j) \\ & + \sum_{\substack{(u,v)\in\mathcal{E}'\setminus\mathcal{E} \\ u\in\mathcal{V}'^s, v\in\mathcal{V}'^t}} c_{\mathcal{G}'}(u,v) + \sum_{\substack{(v_i,v_j)\in\mathcal{E}\cap\mathcal{E}' \\ x_i=0, x_j=1}} c_{\mathcal{G}'}(v_i,v_j), \end{aligned} \tag{53}$$

where the first and the third sums involve two disjoint sets of edges. We can thus minimize separately

both terms, which yields

$$\begin{aligned}
\min_{(\mathcal{V}''^s,\mathcal{V}''^t)\ x''\text{-cut}} c_{\mathcal{G}''}(\mathcal{V}''^s,\mathcal{V}''^t) &= \min_{(\mathcal{V}''^s,\mathcal{V}''^t)\ x''\text{-cut}} \sum_{\substack{(u,v)\in\mathcal{E}\setminus\mathcal{E}'\\ u\in\mathcal{V}^s,v\in\mathcal{V}^t}} c_{\mathcal{G}}(u,v) + \sum_{\substack{(v_i,v_j)\in\mathcal{E}\cap\mathcal{E}'\\ x_i=0,x_j=1}} c_{\mathcal{G}}(v_i,v_j) \\
&+ \min_{(\mathcal{V}''^s,\mathcal{V}''^t)\ x''\text{-cut}} \sum_{\substack{(u,v)\in\mathcal{E}'\setminus\mathcal{E}\\ u\in\mathcal{V}'^s,v\in\mathcal{V}'^t}} c_{\mathcal{G}'}(u,v) + \sum_{\substack{(v_i,v_j)\in\mathcal{E}\cap\mathcal{E}'\\ x_i=0,x_j=1}} c_{\mathcal{G}'}(v_i,v_j) \\
&= \min_{(\mathcal{V}^s,\mathcal{V}^t)\ x\text{-cut}} \sum_{\substack{(u,v)\in\mathcal{E}\setminus\mathcal{E}'\\ u\in\mathcal{V}^s,v\in\mathcal{V}^t}} c_{\mathcal{G}}(u,v) + \sum_{\substack{(v_i,v_j)\in\mathcal{E}\cap\mathcal{E}'\\ x_i=0,x_j=1}} c_{\mathcal{G}}(v_i,v_j) \\
&+ \min_{(\mathcal{V}'^s,\mathcal{V}'^t)\ x'\text{-cut}} \sum_{\substack{(u,v)\in\mathcal{E}'\setminus\mathcal{E}\\ u\in\mathcal{V}'^s,v\in\mathcal{V}'^t}} c_{\mathcal{G}'}(u,v) + \sum_{\substack{(v_i,v_j)\in\mathcal{E}\cap\mathcal{E}'\\ x_i=0,x_j=1}} c_{\mathcal{G}'}(v_i,v_j) \\
\min_{(\mathcal{V}''^s,\mathcal{V}''^t)\ x''\text{-cut}} c_{\mathcal{G}''}(\mathcal{V}''^s,\mathcal{V}''^t) &= \min_{(\mathcal{V}^s,\mathcal{V}^t)\ x\text{-cut}} c_{\mathcal{G}}(\mathcal{V}^s,\mathcal{V}^t) + \min_{(\mathcal{V}'^s,\mathcal{V}'^t)\ x'\text{-cut}} c_{\mathcal{G}'}(\mathcal{V}'^s,\mathcal{V}''^t).
\end{aligned} \tag{54}$$

Eventually, since E and E′ are represented by $\mathcal{G}$ and $\mathcal{G}'$ respectively,

$$\begin{aligned}
\min_{(\mathcal{V}''^s,\mathcal{V}''^t)\ x''\text{-cut}} c_{\mathcal{G}''}(\mathcal{V}''^s,\mathcal{V}''^t) &= \mathrm{E}(x_1,\dots,x_n) + C + \mathrm{E}'(x_1,\dots,x_k,x_{n+1},\dots,x_{n+m-k}) + C' \\
&= \mathrm{E}''(x_1,\dots,x_{n+m-k}) + C'',
\end{aligned} \tag{55}$$

that is, $\mathrm{E}'' = \mathrm{E} + \mathrm{E}'$ is graph-representable.

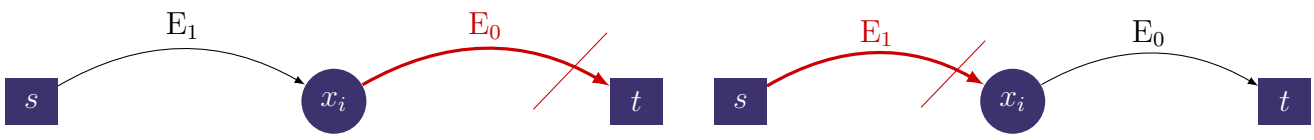
B Cuts in Graphs

Let us show in this section all the cuts in details for each graph, so that one can check that the graph does represent the function.

Unary function We recall that such a function can be written as:

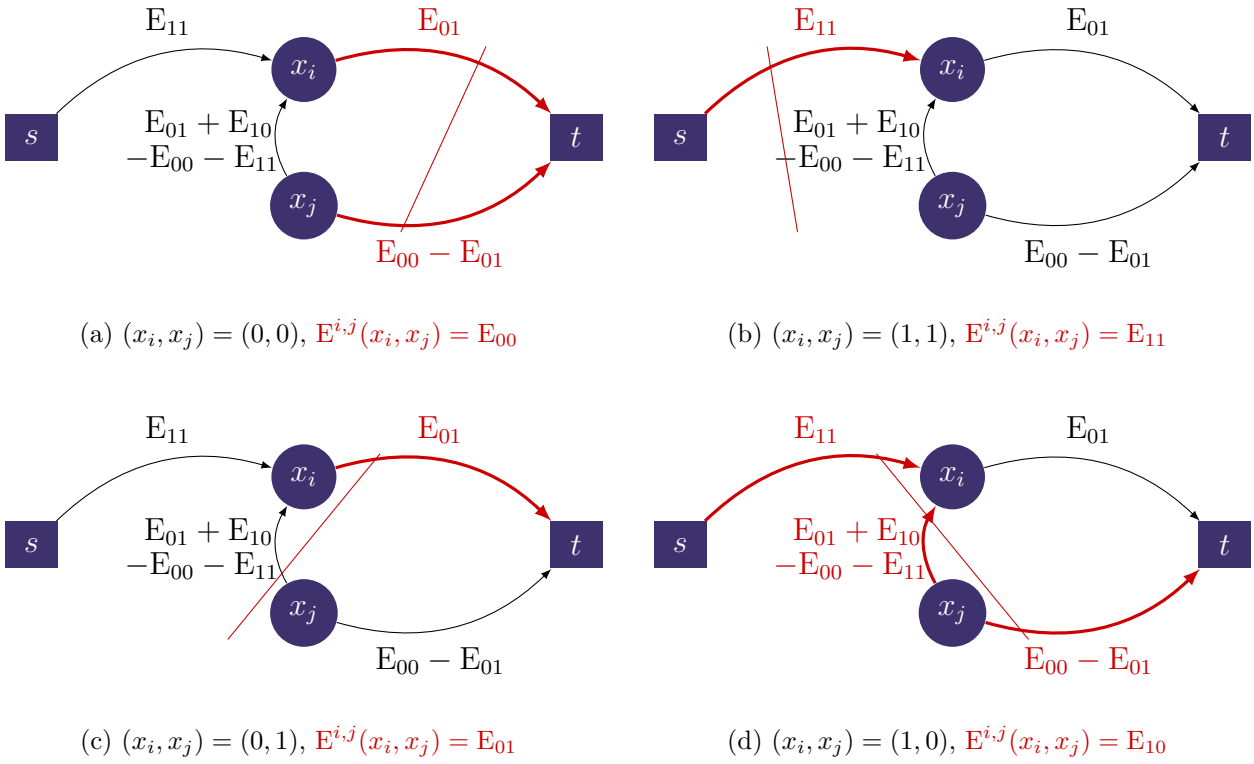
$$\mathrm{E}^i(x_i) = \begin{cases} \mathrm{E}_0 & \text{if } x_i = 0 \\ \mathrm{E}_1 & \text{if } x_i = 1. \end{cases} \tag{56}$$

Then Figure 17 shows every possible cut on the graph we have proposed.



(a) $x_i = 0$, $\mathrm{E}^i(x_i) = \mathrm{E}_0$ (b) $x_i = 1$, $\mathrm{E}^i(x_i) = \mathrm{E}_1$

Figure 17: Cuts for the unary function as constructed in Section 3.1.2.



(a) $(x_i, x_j) = (0, 0)$, $E^{i,j}(x_i, x_j) = E_{00}$

(b) $(x_i, x_j) = (1, 1)$, $E^{i,j}(x_i, x_j) = E_{11}$

(c) $(x_i, x_j) = (0, 1)$, $E^{i,j}(x_i, x_j) = E_{01}$

(d) $(x_i, x_j) = (1, 0)$, $E^{i,j}(x_i, x_j) = E_{10}$

Figure 18: Cuts for the pairwise function as constructed in Section 3.1.2.

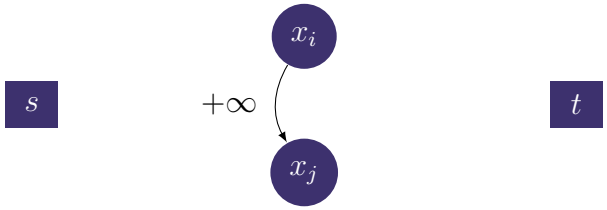


Figure 19: Special representation for function `Energy::forbid01`, preventing the configuratin ($x_i = 0, x_j = 1$). In the code, the infinite weight is replaced by the maximal representable value.

Pairwise function Such a function can be written as:

$$E^{i,j}(x_i, x_j) = \begin{cases} E_{00} & \text{if } x_i = 0 \text{ and } x_j = 0 \\ E_{01} & \text{if } x_i = 0 \text{ and } x_j = 1 \\ E_{10} & \text{if } x_i = 1 \text{ and } x_j = 0 \\ E_{11} & \text{if } x_i = 1 \text{ and } x_j = 1. \end{cases} \tag{57}$$

Then Figure 18 shows every possible cut on the graph we have proposed. To avoid the risk of overflow in this representation, a simpler graph representation equivalent to $E_{00} = E_{10} = E_{11} = 0, E_{01} = +\infty$ is illustrated in Figure 19 and used in function `Energy::forbid01` in the code.

C Submodular Condition in the $\mathcal{F}^2$ Theorem

We have shown in Section 3.1 that the submodularity condition is sufficient to make the function E graph-representable. As proved in [3, 5], it is also a necessary condition. We will not reproduce the

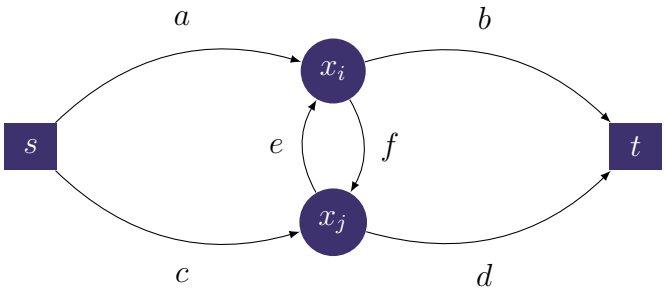


Figure 20: A graph representing a pairwise function.

proof here. Nevertheless, let us show that a pairwise function cannot be represented by a graph with two nodes, as in Figure 20 if it is not submodular. The situation with more than two variables is more complex and is left out of the following proof.

Let $\mathcal{G}$ be a graph representing E with nonnegative weights: $a, b, c, d, e, f \geq 0$, see Figure 20, and with a constant C. Then, the following linear system must be satisfied:

$$a + c + C = \mathrm{E}^{i,j}(1,1) \tag{58}$$
$$b + d + C = \mathrm{E}^{i,j}(0,0) \tag{59}$$
$$a + d + e + C = \mathrm{E}^{i,j}(1,0) \tag{60}$$
$$b + c + f + C = \mathrm{E}^{i,j}(0,1). \tag{61}$$

Adding equalities (60) and (61) yields

$$\begin{aligned} \mathrm{E}^{i,j}(0,1) + \mathrm{E}^{i,j}(1,0) &= a + b + c + d + e + f + 2C \\ &= (a + c + C) + (b + d + C) + e + f \\ &= \mathrm{E}^{i,j}(0,0) + \mathrm{E}^{i,j}(1,1) + e + f \\ &\geq \mathrm{E}^{i,j}(0,0) + \mathrm{E}^{i,j}(1,1), \end{aligned} \tag{62}$$

since $e + f$ is nonnegative. ■

Acknowledgments

This work was supported by Agence Nationale de la Recherche STEREO project (2013-1015). We are grateful for the feedback of one reviewer about this paper, and especially for the very careful and precise proofreading and suggestions of the IPOL section editor, Loïc Simon.

Image Credits

All images by the authors (CC-BY-SA) except:
  Middlebury[3].   CNES.

[3] http://vision.middlebury.edu/stereo/

References

[1] S. Birchfield and C. Tomasi, *A pixel dissimilarity measure that is insensitive to image sampling*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 20 (1998), pp. 401–406. http://dx.doi.org/10.1109/34.677269.

[2] Y. Boykov and V. Kolmogorov, *An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 26 (2004), pp. 1124–1137. http://dx.doi.org/10.1109/TPAMI.2004.60.

[3] V. Kolmogorov, *Graph Based Algorithms for Scene Reconstruction from Two or More Views.*, PhD thesis, Cornell University, 2003.

[4] V. Kolmogorov and R. Zabih, *Computing visual correspondence with occlusions using graph cuts*, in Proceedings of the 8th International Conference on Computer Vision (ICCV), vol. 2, 2001, pp. 508–515. http://dx.doi.org/10.1109/ICCV.2001.937668.

[5] ——, *What energy functions can be minimized via graph cuts?*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 26 (2004), pp. 147–159. http://dx.doi.org/10.1109/TPAMI.2004.1262177.

[6] N. Komodakis, G. Tziritas, and N. Paragios, *Performance vs computational efficiency for optimizing single and dynamic MRFs: Setting the state of the art with primal-dual strategies*, Computer Vision and Image Understanding, 112 (2008), pp. 14–29. http://dx.doi.org/10.1016/j.cviu.2008.06.007.

[7] P. Monasse, *Quasi-euclidean epipolar rectification*, Image Processing On Line, 1 (2011). http://dx.doi.org/10.5201/ipol.2011.m_qer.

[8] D. Scharstein and R. Szeliski, *A taxonomy and evaluation of dense two-frame stereo correspondence algorithms*, International Journal of Computer Vision, 47 (2002), pp. 7–42. http://dx.doi.org/10.1023/A:1014573219977.