CTIM Technical Report

ISSN 2254-2353

Computing Inverse Optical Flow

Javier Sánchez Pérez, Agustín J. Salgado de la Nuez and Nelson Monzón López

No. 3

Las Palmas de Gran Canaria
06 November 2013

Computing Inverse Optical Flow*

Javier Sánchez, Agustín Salgado and Nelson Monzón
Centro de Tecnologías de la Imagen
Departamento de Informática y Sistemas
Universidad de Las Palmas de Gran Canaria
35017 Las Palmas de Gran Canaria, Spain

Abstract

We propose four algorithms for computing the inverse optical flow between two images. We assume that the forward optical flow has already been obtained and we need to estimate the flow in the backward direction. The forward and backward flows can be related through a warping formula, which allows us to propose very efficient algorithms. These are presented in increasing order of complexity. The proposed methods provide high accuracy with low memory requirements and low running times. In general, the processing reduces to one or two image passes. Typically, when objects move in a sequence, some regions may appear or disappear. Finding the inverse flows in these situations is difficult and, in some cases, it is not possible to obtain a correct solution. Our algorithms deal with occlusions very easy and reliably. On the other hand, disocclusions have to be overcome as a post-processing step. We propose three approaches for filling disocclusions. In the experimental results, we use standard synthetic sequences to study the performance of the proposed methods, and show that they yield very accurate solutions. We also analyze the performance of the filling strategies.
Keywords: Inverse Optical Flow, Backward Flow, Inverse Mapping, Back Registration, Occlusions, Disocclusions, Optical Flow, Image Registration

1 Introduction

Computing the optical flow is a basic problem in computer vision and image processing. It has many applications in different areas, such as motion analysis, medical imaging, robot guiding, image editing, stereoscopic vision, etc. In this article we address the problem of estimating the inverse optical flow, which is important in problems where it is necessary to find the correspondences back in time.

The optical flow is a vector field that puts in correspondence the pixels from one image with the pixels of another image. There is a wide range of methods that allow estimating these vector fields [7], [22], [15]. Typically, these methods compute the directed registration map from one image to another, which is normally non congruent with its registration in the opposite direction. There exist many symmetric registration methods that look for congruent solutions in both directions. These symmetric methods have arisen in the fields of optical flow and in non-rigid registration problems in general, like, for instance, in 2D/3D medical image registration. The estimation of the inverse optical flow, or the inverse registration map, is necessary in this kind of methods and has shown to provide better results than the one-directional solutions.

Another interesting application is in the field of medical imaging for computing anatomical atlases. Atlases are very useful for studying and comparing the anatomy of different

*This article is under consideration at Pattern Recognition Letters

patients. In order to compute these atlases, a reference model is registered with respect to several patient models. Then the inverse registrations from the patient models to the atlas are needed for averaging the values and create a more realistic model.

More recently, the inverse flow has also been used in methods where it is necessary to track the flow in previous time instants. If we have a large sequence of images, then it is interesting to obtain continuous and congruent flows through the frames. The backward flow allows to search for the correspondences in the previous frames and create spatio-temporal constraints that yield continuous flow fields in time. We will review all these topics in Section 2.

If we know the optical flow, then it is possible to estimate the backward correspondences with some limitations. In fact, computing the inverse optical flow can be carried out easily in the case of bijective transformations, which is typical in some non-rigid registration problems. Nevertheless, we have to deal with the discrete nature of images that makes it difficult to obtain accurate inverse maps. In the case of natural sequences, the problem is harder due to the presence of occlusions and disocclusions. Occluded regions occur when several correspondences arrive to the same position in the target image and disocclusions occur when no correspondences can be established in one location. The transformation in these regions is not bijective and it is not possible to estimate the inverse map directly.

In this work we propose four algorithms for estimating the backward flow directly from the forward optical flow and the image intensities. We assume that the forward flow has been computed and we need to estimate the backward flow with high precision. The proposed algorithms are simple, fast and accurate. The taxonomy of these algorithms can be separated in two main classes: (i) flow-based methods that only depend on the forward optical flow; and (ii) image-based methods that also rely on the information of the images. We will show that the former are very efficient in running time and memory requirements, whereas the latter are more accurate and stable.

An important issue is the interpolation strategy: vectors are typically given in float precision, so we need to interpolate among several values to get the result in a discrete position. Most state-of-the-art methods use nearest neighbor or linear interpolation. In this work we implement and compare these two strategies. We will see that nearest neighbor is a good solution for translational movements and linear interpolation better adapts to shrinking or diverging motions. One inconvenient of linear interpolation is that it strongly affects the flow discontinuities. Higher order interpolation schemes can be used, but they increase the complexity and running times of the methods.

The proposed methods can automatically cope with occlusions. This problem can be solved with a relative simple approach, since all we have to do is to select the best candidate among several values. Depending on the complexity of motions, we may find two possible situations: occlusions are due to objects that move faster in the scene, which is typical in fronto-parallel stereoscopic sequences; or, in the most general case, occlusions can be produced by any static or moving object. The former situation is interesting in that it only depends on the forward flow, which leads to the aforementioned flow-based methods. Solving this type of occlusions is as simple as selecting the maximum motion [17]. On the other hand, in a general setting, the flow itself is not enough and we also need to use the images (image-based methods [18]), so that the color information allows us to dicriminate between the candidates.

Disocclusions are more difficult to deal with, since no vector points at these regions. In this case, there is a lack of information and the best we can do is to guess their values from the surrounding estimates. These regions can be filled in a post-processing step. We analyze three filling strategies based on the minimum and average values around the area, and an oriented filling strategy.

In the experimental results, we study the precision of our methods using synthetic sequences from the Middlebury benchmark database [6]. This provides complex sequences

with ground truths. We will see that the solutions are very accurate. We also analyze the effect of disocclusions, where the errors are usually higher.

2 Related Works

The use of the backward flow is necessary in several problems. This is the case, for instance, in symmetric optical flow methods [2] or diffeomorphic image registration, e.g. [9], [12] or [3]. These methods introduce the inverse mapping to improve the coherence between the forward and backward flows, reducing the number of false matchings.

Many of these works appear in medical image registration problems, where the insterest is to find a bijective transformation between the images or volumes. In particular, most of these look for a diffeomorphism, which is a continuous, one-to-one and differentiable transformation. These contraints make the estimation of the inverse registration easier. In the case of optical flow, the problem is more difficult because the flow is not normally continuous, presenting strong discontinuities and occlusions.

For instance, the diffeomorphic method presented in [9] explicitly computes the inverse mapping using an iterative algorithm: for each position, it iteratively searches for the inverse correspondence until the estimated error is below a threshold. In each iteration, the algorithm uses trilinear interpolation to compute the flow. This iterative process considerably increases the run-time of the method and its performance is poor at flow dicontinuities. Moreover, it cannot process occlusions correctly. In the symmetric method presented in [8], the inverse optical flow is computed using a Newton scheme. This solution is similar to the previous iterative process, so it presents the same drawbacks, providing poor results at discontinuities and occlussions.

The work presented in [4] proposes a more complex method for inverting a 3D deformation field, based on the ideas of tetrahedral and affine transformation inversion. This method has been implemented in the Statistical Parametric Mapping (SPM) software toolkit [10] and has also been used in [20]. The authors from the last reference claim that the method was very accurate for their test sequences, with average errors smaller than 0,05, but this information is not analyzed in their work.

In [21] the authors propose a stochastic inverse consistency approach that allows discarding regions where no inverses are reliably found. This method is used for stereoscopic sequences and image stitching, where occlusions may be present. They estimate the flow in both directions and then compute two inverses. For each position, if the inverse coincides with the other estimate, then they use this value. Otherwise, the inverse is calculated by searching in the neighborhood and averaging between several candidates. The main drawback of this approach is that it needs a previous backward registration to estimate the inverse flow.

Some other symmetric methods manage to avoid the estimation of the inverse map, like [12]. In this case, the backward flow is modelled in terms of the descent directions of the forward mapping, so afterwards the energy model only depends on the forward flow. Another interesting symmetric model is presented in [1], where the optical flow is computed as a function in the middle position between two frames. In this case, the optical flow represent the same displacement in both directions and it has to be transformed to obtain a vector field in the coordinate system of the first image. In fact, this transformation is very similar to inverting the flow, but it has the advantage that it only needs to be computed once at the end.

Another field of application is in the creation of anatomical atlases. In [11], the authors propose a method for automatically building average anatomical models of the brain. They register the images of several patients with respect to a reference image and then estimate the average model using "forward resampling": the average intensities are distributed using trilinear interpolation. They do not estimate the inverse flow explicitly,

but it basically follows the same steps. Finally, the holes are filled with mathematical morphology operators. Another method for building anatomical atlases is explained in [5]. For computing the inverse flows, they propose to take the nearest neighbour and then filling the holes convolving with a Gaussian.

More recently, the backward flow has been used in spatio-temporal optical flow methods, e.g. [16] or [19]. The objective is to preserve the temporal coherence of the optical flows with the previous estimated flows. The inverse mapping is used to find the correspondences back in time and impose some kind of temporal continuity. Another example of the use of the backward flow is given in [13] and [14]. In this case, the authors propose a method that relies on the reverse optical flow to automatically follow the road in autonomous cars. It tracks features from the current position to a past position, so the robot may identify similar shapes at different scales.

In Sect. 3 we explain the basic concepts of the problem and analyze the problems of occlusions and disocclusions. The four algorithms are presented in Sect. 4. In the experimental results, Sect. 6, we evaluate the algorithms using several sequences from the Middlebury benchmark datasets. We study several aspects, related with the accuracy of the inverse estimations, the filling strategies, the stability of the solutions and the behavior with respect to different types of occlusions. Finally the conclusions in Sect. 7.

3 Notation and General Framework

Let $\mathbf{I}_1(\mathbf{x})$ and $\mathbf{I}_2(\mathbf{x})$ be two color images in a sequence, with $\mathbf{x} = (x, y)$, and $\mathbf{h}(\mathbf{x}) = (u(\mathbf{x}), v(\mathbf{x}))$ the vector field that establishes the correspondences between the pixels of the images. $\mathbf{h}(\mathbf{x})$ is said to be *dense* if it is a mapping of every pixel in the first image to the pixels in the second image. We define the backward flow, $\mathbf{h}^*(\mathbf{x}) = (u^*(\mathbf{x}), v^*(\mathbf{x}))$, as the inverse of $\mathbf{h}(\mathbf{x})$. $\mathbf{h}^*(\mathbf{x})$ puts in correspondence the pixels in the second image with the pixels in the first image. The forward and backward flows can be related as:

$$\mathbf{h}(\mathbf{x}) = -\mathbf{h}^*(\mathbf{x} + \mathbf{h}(\mathbf{x})) \;\text{ or }\; \mathbf{h}^*(\mathbf{x}) = -\mathbf{h}(\mathbf{x} + \mathbf{h}^*(\mathbf{x})). \tag{1}$$

This relation can be intuitively derived from the graphic depicted in Fig. 1. The corresponding positions are given by the warping function, $\mathbf{x} \rightarrow \mathbf{x} + \mathbf{h}(\mathbf{x})$.

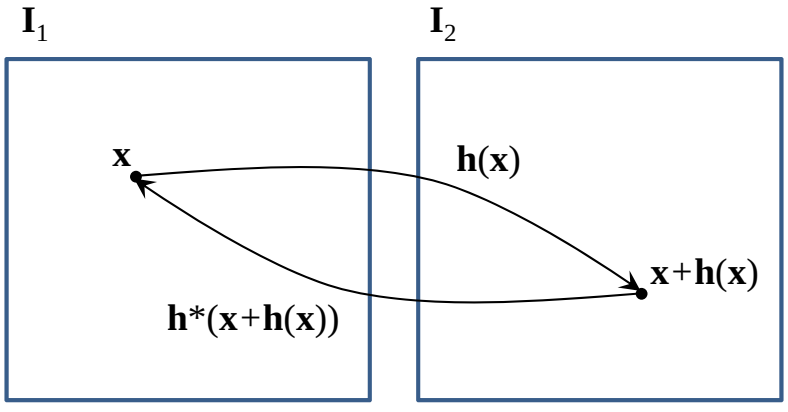

Figure 1: Relation between the forward, $\mathbf{h}(\mathbf{x})$, and backward, $\mathbf{h}^*(\mathbf{x})$, optical flows.

Typically, $\mathbf{h}(\mathbf{x})$ is not bijective and it is not possible to estimate $\mathbf{h}^*(\mathbf{x})$ in the whole domain. This is true, for instance, in occluded and disoccluded regions: occlusions occur when several correspondences arrive to the same position in the second image; disocclusions occur when an object appears and there is no correspondence in the first image.

Since occlusions and disocclusions are common in most natural sequences, $\mathbf{h}(\mathbf{x})$ is in general not invertible. Thus, we have to deal with these two problems in order to find

dense inverse functions. These are easy to detect but their solutions have to be overcome from different perspectives. Looking at Fig. 2, we observe that occlusions appear in the direction of the moving objects and disocclusions appear in the opposite side.

We also find two possible situations: occlusions may appear when moving objects occlude other background objects, like depicted in Fig. 2; on the other hand, occlusions may also occur when objects displace behind other objects, like in Fig. 3. The former situation is typical in fronto-parallel stereoscopic sequences, so we will refer this situation as the *stereoscopic occlusion*. It can be efficiently solved, like proposed in [17]. The latter usually appears in general optical flow problems [18].

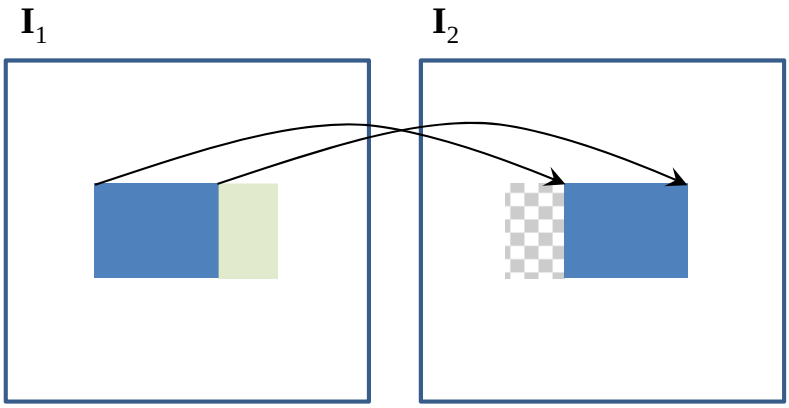

Figure 2: *Stereoscopic Occlusion*: when the blue square moves horizontally, it creates a disocclusion and occlusion before and after the square, respectively.

We call this last situation the *street-lamp occlusion*, as it typically appears in traffic sequences where a car moves behind a street lamp. Looking at Fig. 3, we observe that occlusions are not only present in front of the square but also inside it, since these pixels fall exactly behind the bar in the second image. The effect of this occlusion is like projecting the bar into the square at a distant equivalent to its motion.

The problem of occlusions can be solved by means of a selection process. Several vectors point at the same position, and we need to select the appropriate one. The selection problem may only depend on the vector field itself, as it happends in the stereoscopic occlusion. In this case, we can select the value corresponding to the biggest motion. On the other hand, the more general case of the street-lamp occlusion requires not only the values of the optical flow but also the image intensities: we have to select the motion associated with the pixel that is finally visible in the second image.

In the disocclusion problem, there is no information available for a given position. This can be addressed as a filling procedure, selecting the information on the neighborhood of the pixel. Notice the symmetric behavior between occlusions and disocclusions from Figs. 2 and 3.

Fig. 5 shows two examples of inverse optical flows. The motion fields are shown using the color scheme depicted in Fig. 4. In this case, we have used two sequences from the Middlebury benchmark database. The forward optical flows are known (second column) and the inverse flows are calculated using one of our algorithms (third column). We also show the disocclusion maps in the fourth column.

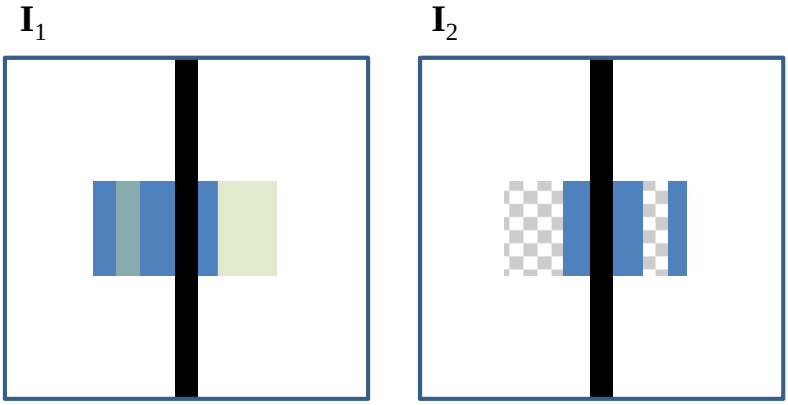

Figure 3: *Street-lamp Occlusion*: the blue square moves behind the static central bar. Occlusions appear in front of the square and also inside the square, induced by the static bar.

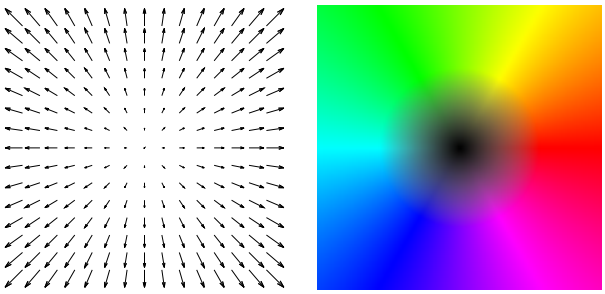

Figure 4: Color scheme used to represented the optical flows. The colors represent the orientation and magnitude of the flow field.

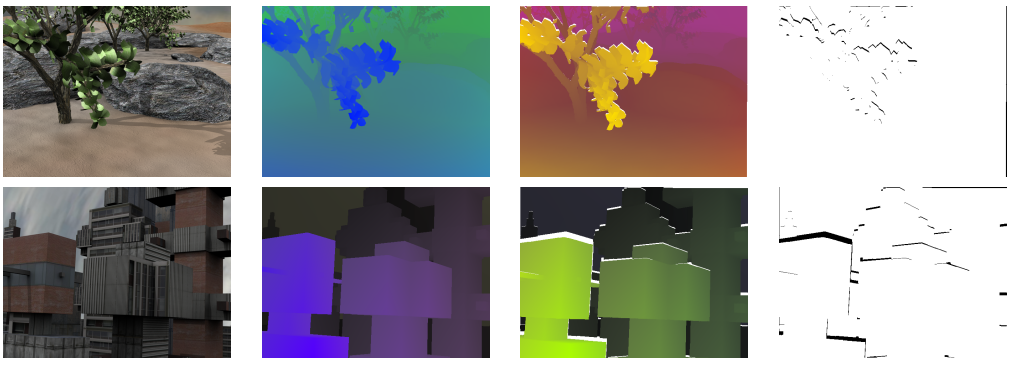

Figure 5: Two examples using sequences from the Middlebury benchmark database: the first column shows the Grove2 and Urban3 sequences, respectively; the second depicts the forward optical flows; the third, the inverse optical flows; and the last one, the corresponding disocclusions.

4 Algorithms for Computing the Inverse Optical Flow

In this section we explain the algorithms. These algorithms have been grouped in two categories: *nearest neighbor algorithms* that use a single value from the forward flow; and *interpolation algorithms* that compute an average between the values that get close to a position. These are explained in subsections 4.1 and 4.2, respectively. Nearest neighbor algorithms only need one image pass, while the others need two passes.

These categories are further divided in flow- and image-based strategies. The flow-based strategy uses the information of the forward optical flow, whereas the image-based strategy also uses the information of the image intensities. The former can be applied in the stereoscopic occlusion case and the latter in the more general street-lamp occlusion.

4.1 Nearest Neighbor Algorithms

Our first algorithm is based on the optical flow information. It is highly efficient and provides good results for the stereoscopic occlusion case. Algorithm 1 depicts the steps to compute the backward flow. This algorithm is very fast since only one pass through the image is necessary. It does not use any temporal buffer because the operations are carried out in the output backward flow.

Algorithm 1: Inverse Flow

Input: $\mathbf{h}$
Output: $\mathbf{h}^*$
Initialize $\mathbf{h}^*$ to 0
foreach *position* $\mathbf{x}$ **do**
  $\mathbf{x}_w \leftarrow \mathbf{x} + \mathbf{h}(\mathbf{x})$
  Find the four neighbors of $\mathbf{x}_w : \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$
  Compute the bilinear interpolation weights: w_1, w_2, w_3, w_4 (see Fig. 6)
  $d \leftarrow \|\mathbf{h}(\mathbf{x})\|^2$
  $d_1 \leftarrow \|\mathbf{h}^*(\mathbf{x}_1)\|^2$
  $d_2 \leftarrow \|\mathbf{h}^*(\mathbf{x}_2)\|^2$
  $d_3 \leftarrow \|\mathbf{h}^*(\mathbf{x}_3)\|^2$
  $d_4 \leftarrow \|\mathbf{h}^*(\mathbf{x}_4)\|^2$
  if $w_1 \geq 0,25$ ***and*** $d \geq d_1$ **then**
    $\mathbf{h}^*(\mathbf{x}_1) \leftarrow -\mathbf{h}(\mathbf{x})$
  end
  if $w_2 \geq 0,25$ ***and*** $d \geq d_2$ **then**
    $\mathbf{h}^*(\mathbf{x}_2) \leftarrow -\mathbf{h}(\mathbf{x})$
  end
  if $w_3 \geq 0,25$ ***and*** $d \geq d_3$ **then**
    $\mathbf{h}^*(\mathbf{x}_3) \leftarrow -\mathbf{h}(\mathbf{x})$
  end
  if $w_4 \geq 0,25$ ***and*** $d \geq d_4$ **then**
    $\mathbf{h}^*(\mathbf{x}_4) \leftarrow -\mathbf{h}(\mathbf{x})$
  end
end
`FillDisocclusions`

$\|\cdot\|$ stands for the Euclidean norm of a vector. At the beginning, we initialize the

inverse flow to zero. In each pixel, $\mathbf{x} = (x, y)$, we compute the corresponding position in the other image, $\mathbf{x} + \mathbf{h}(\mathbf{x})$. Normally, this position will lie in the middle of four pixels.

We obtain the neighbors around this position, given by $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$, and the interpolation weights, w_1, w_2, w_3, w_4. These are shown in Fig. 6. The weights represent the proportional area of the pixel left to each neighbor. Then, we estimate the magnitude of the forward flow and compare it with the magnitude of the already stored backward flows.

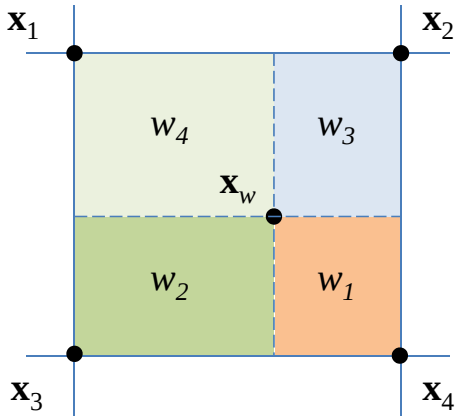

Figure 6: Weights.

If the value of the forward magnitude is bigger than the previous stored value, and the corresponding weight is bigger than a given threshold, then we keep the negative value of the flow at that position. This threshold has been set to 0,25 because it represents the situation when the correspondence falls exactly in the middle of the pixel. Another alternative is to use distances instead of areas. Nevertheless, the behavior is very similar, but we will prefer areas for the interpolation algorithms.

In this way, occlusions are automatically handled by the algorithm: if there are collisions in one position, we retain the flow with higher magnitude. In the last step, we fill disocclusions, which correspond to the positions that have not been visited in the backward flow.

If we want to deal with the street-lamp occlusion case, then we need more information than the optical flow. Now the objects with the smallest motions can occlude other fast moving objects. Thus, the magnitude of the flow cannot be a discriminant.

We propose to use the intensities of the images as discriminant. We select the motion corresponding to the pixel with the most similar intensities in both images. Algorithm 2 shows the steps of the new method. This algorithm is also very efficient and applies to any configuration. Additionally, we need a buffer to store the pixel similarities, so that, in case of an occlusion, we retain the value corresponding to the most similar value.

The input data includes the forward flow and both images. At the beginning, we initialize the $buffer$ to a big number. Then, the algorithm goes over each position of the vector field, carrying out the same steps as in the previous algorithm. The main difference is that it estimates the similarity between the images at the given positions. It assigns the negative value of the original flow to each of the neighbors, if it has the best similarity and lies close to the destiny pixel.

Notice that similarities are stored in a $buffer$, so that when there are several pixels that arrive to the same position, i.e., in the case of occlusions, the algorithm retains the value corresponding to the most similar pixels in both images. In this case, the use of the $buffer$ allows us to automatically deal with occlusions. This algorithm is also very fast, since only one pass through the image is necessary.

For the similarity measure, we have used the RGB color information. This may pose some drawbacks if there are brightness changes, which is typical in real sequences. Other

Algorithm 2: Inverse Image-based Maximum Flow

Input: $\mathbf{I}_1, \mathbf{I}_2, \mathbf{h}$
Output: $\mathbf{h}^*$

Initialize each position of a $buffer$ to a big number
foreach *position* $\mathbf{x}$ **do**
  $\mathbf{x}_w \leftarrow \mathbf{x} + \mathbf{h}(\mathbf{x})$
  Find the four neighbors of $\mathbf{x}_w : \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$
  Compute the bilinear interpolation weights: w_1, w_2, w_3, w_4 (see Fig. 6)
  $d_1 \leftarrow \|\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}_1)\|^2$
  $d_2 \leftarrow \|\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}_2)\|^2$
  $d_3 \leftarrow \|\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}_3)\|^2$
  $d_4 \leftarrow \|\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}_4)\|^2$
  if $w_1 \geq 0,25$ ***and*** $buffer(\mathbf{x}_1) \geq d_1$ **then**
    $\mathbf{h}^*(\mathbf{x}_1) \leftarrow -\mathbf{h}(\mathbf{x})$
    $buffer(\mathbf{x}_1) \leftarrow d_1$
  end
  if $w_2 \geq 0,25$ ***and*** $buffer(\mathbf{x}_2) \geq d_2$ **then**
    $\mathbf{h}^*(\mathbf{x}_2) \leftarrow -\mathbf{h}(\mathbf{x})$
    $buffer(\mathbf{x}_2) \leftarrow d_2$
  end
  if $w_3 \geq 0,25$ ***and*** $buffer(\mathbf{x}_3) \geq d_3$ **then**
    $\mathbf{h}^*(\mathbf{x}_3) \leftarrow -\mathbf{h}(\mathbf{x})$
    $buffer(\mathbf{x}_3) \leftarrow d_3$
  end
  if $w_4 \geq 0,25$ ***and*** $buffer(\mathbf{x}_4) \geq d_4$ **then**
    $\mathbf{h}^*(\mathbf{x}_4) \leftarrow -\mathbf{h}(\mathbf{x})$
    $buffer(\mathbf{x}_4) \leftarrow d_4$
  end
end
`FillDisocclusions`

robust similarity measures can be used, based on the gradient or the curvature of the images. The noise of the images may also affect the similarity measure. A simple approach to mitigate this problem is to convolve the images with a Gaussian kernel.

Fig. 7 shows an example of the street-lamp occlusion case. We have modified the Urban2 sequence from the Middlebury benchmark database by introducing two static bars. We show the results for Algorithms 1 and 2. We observe that the first algorithm can not correctly deal with the static bars. On the other hand, the second algorithm can cope with this situation, but it does not detect the correct motion in a few pixels of the front building. This region coincides with an occlusion and brightness changes.

4.2 Interpolation Algorithms

Given the discrete nature of images, normally the correspondences fall among several pixels in the other image. In fact, many correspondences may get around one pixel. It could be more convenient to use all this information and compute an average between these values. In the previous algorithms, we have selected one value; therefore we are introducing a shift on the positions of the inverse flow.

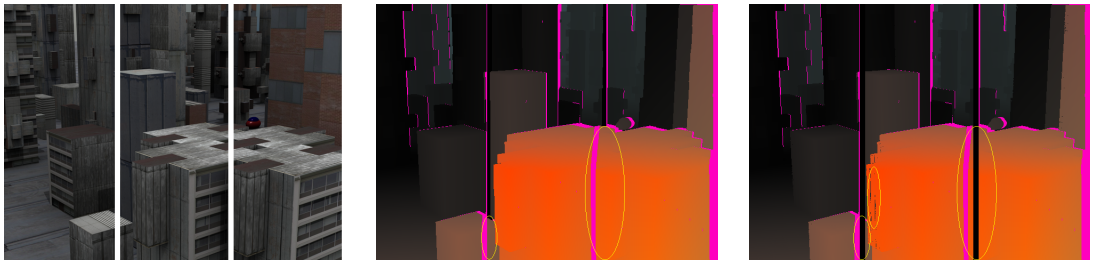

Figure 7: Dealing with street-lamp occlusions: left, the Urban2 sequence; middle, result of Algorithm 1; and right, result of Algorithm 2. The first algorithm does not detect the motion of the bars and the second algorithm introduces some errors due to brightness changes (see yellow ellipses).

In this section we propose two algorithms addressing this problem. We note that, although it is interesting to compute an average between the correspondences that get around a pixel, at flow discontinuities or occlusions we have to average only with the values of their corresponding regions. This is necessary to avoid degrading these areas.

In order to compute the inverse flow, we calculate the weighted average of the flows, using the following equation:

$$\mathbf{h}^*(\mathbf{x}) = -\frac{\sum\limits_{\mathbf{x}_i \in \mathcal{N}_i} w_i \mathbf{h}(\mathbf{x}_i)}{\sum\limits_{\mathbf{x}_i \in \mathcal{N}_i} w_i} \tag{2}$$

where $\mathcal{N}_i \equiv \{\mathbf{x}_i : \|\mathbf{x}_i + \mathbf{h}(\mathbf{x}_i) - \mathbf{x}\| < 1\}$, i.e., $\mathcal{N}_i$ is the set of correspondences that fall around $\mathbf{x}$.

The first algoritm in this section relies on a procedure that is in charge of computing the weighted average (2). This formula is computed iteratively, as we find new values around a position.

Procedure SelectMotion($d, \mathbf{h}, wght, d^*, \mathbf{h}^*, wght^*$)

```
if wght ≥ 0,25 then
    if abs(d − d*) ≤ MOTION_TH then
        h* ← h* + h ∗ wght
        wght* ← wght* + wght
    else if d ≥ d* then
        d* ← d
        h* ← h ∗ wght
        wght* ← wght
    end
end
```

This procedure receives the forward optical flow, $\mathbf{h}$, its magnitude, d, the weight as depicted in Fig. 6, $wght$, and the accumulative values stored at this position d^*, $\mathbf{h}^*$ and $wght^*$. If the current flow magnitude, d, is within a threshold (`MOTION_TH`) of the stored flow magnitude, then we accumulate the values for $\mathbf{h}^*$ and $wght^*$, in order to progressively estimate (2). Otherwise, if the current magnitude is bigger than the stored one, then we initialize these variables to the current values. In this procedure, we use a constant, `MOTION_TH`, that is used to detect similar motions. By default, its value is equal to $0,25$. The second condition allows us to preserve the flow discontinuities and select the correct value at occlusions.

Algorithm 3 is an improvement with respect to Algorithm 1. The steps of both algorithms are essentially the same, with the following differences: the new method uses the buffers **avg_h** and $wght^*$ to accumulate the value of the additions in (2); an additional buffer, d^*, to store the calculated flow magnitudes; and four calls to the previous procedure, to select the correct motion in each position. Finally, the algorithm calculates (2) using the accumulation buffers. This is carried out through another image pass.

Algorithm 3: Inverse Average Flow

Input: $\mathbf{h}$
Output: $\mathbf{h}^*$

Initialize buffers $d^*, \mathbf{avg\_h}, wght^*$ to 0

foreach *position* $\mathbf{x}$ **do**
  $\mathbf{x}_w \leftarrow \mathbf{x} + \mathbf{h}(\mathbf{x})$
  Find the four neighbors of $\mathbf{x}_w : \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$
  Compute the bilinear interpolation weights: w_1, w_2, w_3, w_4 (see Fig. 6)
  $d \leftarrow \|\mathbf{h}(\mathbf{x})\|^2$
  `SelectMotion` $(d, \mathbf{h}(\mathbf{x}), w_1, d^*(\mathbf{x}_1), \mathbf{avg\_h}(\mathbf{x}_1), wght^*(\mathbf{x}_1))$
  `SelectMotion` $(d, \mathbf{h}(\mathbf{x}), w_2, d^*(\mathbf{x}_2), \mathbf{avg\_h}(\mathbf{x}_2), wght^*(\mathbf{x}_2))$
  `SelectMotion` $(d, \mathbf{h}(\mathbf{x}), w_3, d^*(\mathbf{x}_3), \mathbf{avg\_h}(\mathbf{x}_3), wght^*(\mathbf{x}_3))$
  `SelectMotion` $(d, \mathbf{h}(\mathbf{x}), w_4, d^*(\mathbf{x}_4), \mathbf{avg\_h}(\mathbf{x}_4), wght^*(\mathbf{x}_4))$
end
foreach *position* $\mathbf{x}$ **do**
  if *pixel* $\mathbf{x}$ *is not a disocclusion* **then**
    $\mathbf{h}^*(\mathbf{x}) \leftarrow -\frac{\mathbf{avg\_h}(\mathbf{x})}{wght^*(\mathbf{x})}$
  end
end
`FillDisocclusions`

This algorithm is also very efficient. It uses two image passes to compute the inverse flow: one for creating the intermediate buffers and another for calculating (2). The number of operations in the second pass reduces to two divisions and two assignments per pixel. The main advantage is that the accuracy of the calculated inverse flows is higher.

The fourth algorithm relies on another procedure that plays the same role as the previous one. If the magnitudes of the current and stored flows are similar, then the procedure accumulates the information. Otherwise, the procedure keeps the flow associated with the most similar pixel intensity. Thus, the main difference is that the discriminant for selecting a value depends on the RGB similarity.

Algorithm 4 computes flow averages and uses the image intensities to select the correct flow. This new method uses the buffers, **avg_h** and $wght^*$ to accumulate the value of the summations in (2); an additional buffer, dI, to store the calculated RGB differences of both images; and four calls to the previous procedure, to select the correct motion in each position. A final image pass is necessary to calculate (2).

The performance of this algorithm is similar to the previous algorithm. The benefit is that it properly works for stereoscopic and street-lamp occlusions. Nevertheless, it carries out some more operations for calculating the RGB differences between the images. Note that these interpolation algorithms can also cope with occlusions and respect the flow discontinuities.

The algorithms explained in this section can be easily extended to higher order dimensions. This is useful, for instance, in three dimensional medical images. In this case, the

Procedure SelectImageMotion($dI, \mathbf{h}, wght, dI^*, \mathbf{h}^*, wght^*$)

if $wght \geq 0,25$ **then**
  $d \leftarrow \|\mathbf{h}\|^2$
  if `abs`$(d - d^*) \leq$ `MOTION_TH` **then**
    $\mathbf{h}^* \leftarrow \mathbf{h}^* + \mathbf{h} * wght$
    $wght^* \leftarrow wght^* + wght$
  else if $dI^* \geq dI$ **then**
    $d^* \leftarrow d$
    $dI^* \leftarrow dI$
    $\mathbf{h}^* \leftarrow \mathbf{h} * wght$
    $wght^* \leftarrow wght$
  end
end

problem of occlusions and disocclusions is not very important and the accuracy of these methods can be even higher.

Algorithm 4: Inverse Image-based Average Flow

Input: $\mathbf{I}_1, \mathbf{I}_2, \mathbf{h}$
Output: $\mathbf{h}^*$

Initialize buffers $\mathbf{avg\_h}, wght^*$ to 0

Initialize buffer dI to a big number

foreach *position* $\mathbf{x}$ **do**

$\mathbf{x}_w \leftarrow \mathbf{x} + \mathbf{h}(\mathbf{x})$

Find the four neighbors of $\mathbf{x}_w : \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$

Compute the bilinear interpolation weights: w_1, w_2, w_3, w_4 (see Fig. 6)

$d_1 \leftarrow \|\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}_1)\|^2$
$d_2 \leftarrow \|\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}_2)\|^2$
$d_3 \leftarrow \|\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}_3)\|^2$
$d_4 \leftarrow \|\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}_4)\|^2$

`SelectImageMotion` $(d_1, \mathbf{h}(\mathbf{x}), w_1, dI(\mathbf{x}_1), \mathbf{avg\_h}(\mathbf{x}_1), wght^*(\mathbf{x}_1))$
`SelectImageMotion` $(d_2, \mathbf{h}(\mathbf{x}), w_2, dI(\mathbf{x}_2), \mathbf{avg\_h}(\mathbf{x}_2), wght^*(\mathbf{x}_2))$
`SelectImageMotion` $(d_3, \mathbf{h}(\mathbf{x}), w_3, dI(\mathbf{x}_3), \mathbf{avg\_h}(\mathbf{x}_3), wght^*(\mathbf{x}_3))$
`SelectImageMotion` $(d_4, \mathbf{h}(\mathbf{x}), w_4, dI(\mathbf{x}_4), \mathbf{avg\_h}(\mathbf{x}_4), wght^*(\mathbf{x}_4))$

end

foreach *position* $\mathbf{x}$ **do**

if *pixel* $\mathbf{x}$ *is not a disocclusion* **then**

$\mathbf{h}^*(\mathbf{x}) \leftarrow -\frac{\mathbf{avg\_h}(\mathbf{x})}{wght^*(\mathbf{x})}$

end

end

`FillDisocclusions`

5 Filling Disocclusions

The last step of the algorithms is to fill disocclusions. These are regions with no correspondences in the source image, so we get empty spaces in the backward flow. It is not possible to find an exact solution if only two images are taken into account. Probably, the use of more frames may allow us to uniquely identify the values inside these regions.

The best we can do is to guess the information from the neighbor values. We propose three filling strategies: on the one hand, we use a minimum fill strategy that looks for the minimum value around the disocclusion; then, we propose an averaging solution that computes the average from the values around the region; finally, we propose an oriented filling scheme that searches the value in the opposite direction to the flow.

5.1 Minimum fill strategy

The min-fill strategy is justified in the case of stereoscopic occlusions. In this situation, disocclusions are associated with the lowest moving objects, so it seems reasonable to select the minimum motion. In [17], we analyzed two min-fill strategies.

The first strategy uses a connected component labeling process to group the regions. It assigns a label to each region, finds a unique minimum around each area, and then assigns the value to each position inside. This strategy is simple and fast. It works correctly if the size of the regions is small. However, when regions are large, it may assign values which are very far from the position.

The second strategy finds the minimum value that is near the current position. We

go through the image and try to fill disocclusions using a fix-sized window. The size of the window may not be large enough to attain values outside the region. In this case, the process is run again to fill the remaining disocclusions. The default window radius is 5 in the experimental results. In the experimental results, we use this strategy, since it usually provides better results, as shown in [17].

5.2 Average fill strategy

An alternative to the min-fill strategy is to use an average filling. Instead of selecting a unique value, we compute an average in a fix-sized window. The process is very similar to the previous approach: we accumulate the values and assign the average to its corresponding position if the number of values is greater than a threshold. This threshold is equal to the windows radius (5 in the experiments). This process is carried out iteratively until every disocclusion is succesfully filled.

This approach is a compromise between both types of occlusions: it is not going to work as well as the min-fill strategy for the stereoscopic case, but it should outperform this strategy for the street-lamp occlusion.

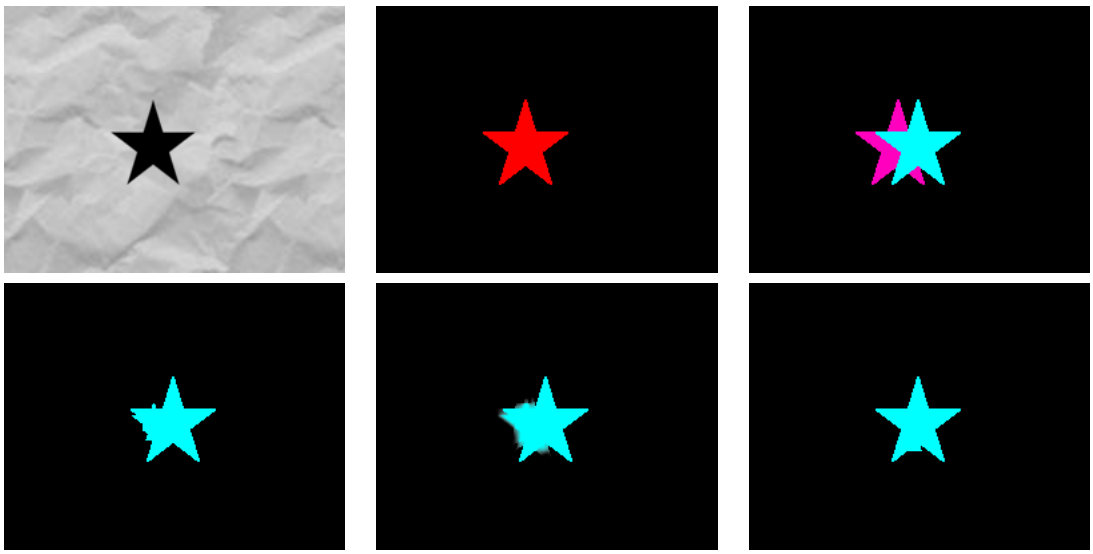

Figure 8: Comparison of filling strategies. In the first row: first column, the left image; second, the ground truth optical flow; and third, the inverse flow with disocclusions in pink. In the second row: first column, the solution using the min-fill strategy; second, using the average-fill strategy; and third, using the oriented-fill strategy.

5.3 Oriented fill strategy

We propose another strategy that takes into account the information of the flow orientation. If one object moves in one direction, it seems reasonable that the motion at disocclusions corresponds to the motion that is in the opposite direction.

In this sense, what we do is the following: we first compute the search vector for each position as $-\frac{\mathbf{h}(\mathbf{x})}{\|\mathbf{h}(\mathbf{x})\|}$; then, it iterates until it finds a value outside the disoccluded region in this direction.

Note that we use the forward flow to estimate the vector orientation. This probably works very well for the stereoscopic occlusion, since normally the motion corresponds to the backward object. In the street-lamp case, this strategy works properly if the disocclusion falls inside the correct object.

An improvement to this strategy is to use, not only one vector in the opposite direction, but also its orthogonal vectors, so that the filling process searches for information in more directions. This can be efficiently addressed through an oriented diffusion process.

In Fig. 8 we compare these filling strategies. We use a simple sequence of a star moving fifteen pixels horizontally. This sequence is interesting because disoccluded regions are large and complex (see the image on the first row and last column). The performance of the average-fill strategy (second row in the middle) is the poorest. The min-fill strategy provides very good results, but when the search window is not large enough, then it takes values from the motion of the star. Finally, the oriented-fill strategy provides the best results in this case.

6 Experimental results

In order to evaluate the methods, we use an error measure based on the following *reprojection error*: we compute the backward flow twice, $(\mathbf{h}^*(\mathbf{x}))^*$; then, we compare the result with the original ground truth using the average End-point Error (EPE) and Angular Error (AAE), as explained in [6]. Fig. 9 shows the three functions – $\mathbf{h}(\mathbf{x})$, $\mathbf{h}^*(\mathbf{x})$ and $(\mathbf{h}^*(\mathbf{x}))^*$– for the Urban2 sequence. The last two functions have been computed using Algorithm 1 and the min-fill strategy.

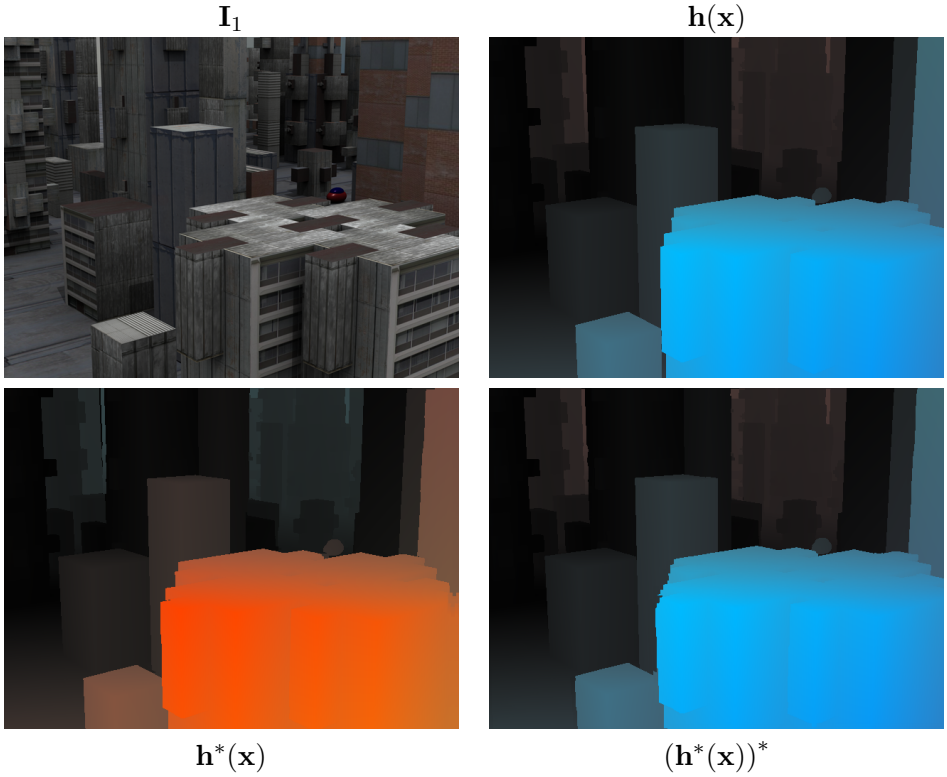

Figure 9: Reprojection error: first image in the top row, the Urban2 sequence from the Middlebury database; second image, the ground truth optical flow; first image in the bottom row, the inverse flow, $\mathbf{h}^*(\mathbf{x})$; and, second in the bottom, the inverse of the inverse flow, $(\mathbf{h}^*(\mathbf{x}))^*$.

Notice that $(\mathbf{h}^*(\mathbf{x}))^*$ is slightly different than $\mathbf{h}(\mathbf{x})$, especially at occlusions. In the experimental results we will be using the EPE and AAE between these two functions for comparing the accuracy of the algorithms. We will use the test sequences from the Middlebury benchmark database [6], which provides the correct ground truths. Unfortunately, there are some sequences in the database that mix the information of occlusions with the

optical flows, so we will only use these when possible.

Another alternative is to use a symmetric error metric based on (1) as $\|\mathbf{h}(\mathbf{x}) + \mathbf{h}^*(\mathbf{x} + \mathbf{h}(\mathbf{x}))\|$. This has been previously used in the symmetrical optical flow method explained in [2]. However, we need to use an interpolation method to estimate $\mathbf{h}^*(\mathbf{x} + \mathbf{h}(\mathbf{x}))$, which will introduce a bias in the computed errors, especially at discontinuities. The advantage of the previous metric is that it only depends on the inverse calculations.

6.1 Analysis of inverse flow accuracy

In this section we analyze the methods without taking into account disocclusions. Figures 10 and 11 show the results for Algorithm 1.

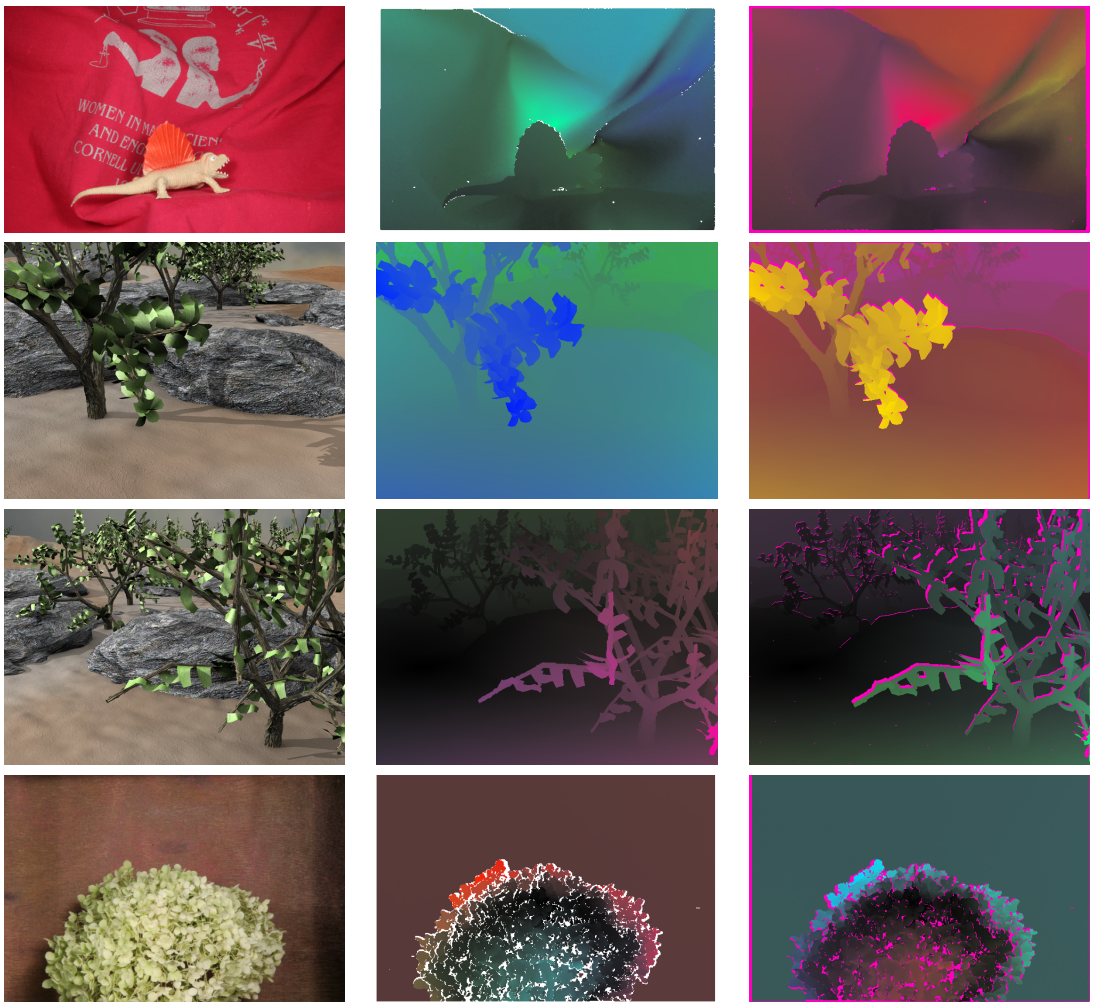

Figure 10: Middlebury test sequences. First column, the source image; second, the ground truth optical flow; third, the inverse optical flow using Algorithm 1, with disocclusions in pink.

Disocclusions are marked in pink. We can see that the precision of the inverse optical flow seems high and occlusions are correctly handled by the algorithm. Disocclusions are large in some sequences like, for instance, in the building or tree sequences.

The EPE and AAE results are shown in Tables 1 and 2, respectively. The best result per row is in bold letters and the second in italics. The first algorithm provides very good results but these are the poorest in the table. This method is interesting because of its simplicity: it only depends on the optical flow, does not use any additional buffer and requires less calculations. The third algorithm has the same features as the first one, but

it computes averages with several values per position. Its accuracy is slightly better for most sequences and much better in sequences where the flow is continuous or divergent, like in Dimetrodon and Yosemite.

The best EPE results are obtained for Algorithms 2 and 4. These two rely on the image intensities in order to select the appropriate inverse value. This means that the pixel information is more discriminant and reliable than the flow data.

Table 1: Reprojection error (EPE) for the Middlebury sequences.

Sequence	**Algorithm 1**	**Algorithm 2**	**Algorithm 3**	**Algorithm 4**
Dimetrodon	0,014	*0,006*	0,007	**0,006**
Grove2	0,020	**0,008**	0,018	*0,009*
Grove3	0,094	**0,044**	0,089	*0,045*
Hydrangea	0,019	**0,009**	0,017	*0,010*
RubberWhale	0,010	**0,003**	0,006	*0,004*
Urban2	0,027	*0,011*	0,025	**0,011**
Urban3	0,030	*0,010*	0,027	**0,010**
Venus	0,015	**0,006**	0,015	*0,006*
Yosemite	0,009	0,004	*0,004*	**0,004**

It is interesting to note that, while the second algorithm outstands in most of the EPE results, the fourth wins in the AAE comparisons. This is because Algorithm 4 is based on the average value, which may benefit a better estimation of the flow direction. This effect is more noticeable in the Dimetrodon and Yosemite sequences as well.

Table 2: Reprojection error (AAE) for the Middlebury sequences.

Sequence	**Algorithm 1**	**Algorithm 2**	**Algorithm 3**	**Algorithm 4**
Dimetrodon	$0{,}359^o$	$0{,}203^o$	$\mathit{0{,}154^o}$	$\mathbf{0{,}138^o}$
Grove2	$0{,}438^o$	$\mathbf{0{,}229^o}$	$0{,}466^o$	$\mathit{0{,}242^o}$
Grove3	$1{,}205^o$	$\mathit{0{,}736^o}$	$1{,}233^o$	$\mathbf{0{,}671^o}$
Hydrangea	$0{,}463^o$	$\mathit{0{,}259^o}$	$0{,}356^o$	$\mathbf{0{,}250^o}$
RubberWhale	$0{,}441^o$	$\mathit{0{,}195^o}$	$0{,}273^o$	$\mathbf{0{,}169^o}$
Urban2	$0{,}318^o$	$\mathit{0{,}163^o}$	$0{,}307^o$	$\mathbf{0{,}150^o}$
Urban3	$0{,}320^o$	$\mathbf{0{,}171^o}$	$0{,}366^o$	$\mathit{0{,}178^o}$
Venus	$0{,}257^o$	$\mathbf{0{,}087^o}$	$0{,}257^o$	$\mathit{0{,}093^o}$
Yosemite	$0{,}261^o$	$0{,}132^o$	$\mathit{0{,}122^o}$	$\mathbf{0{,}111^o}$

We may conclude that image-based algorithms (Algorithms 2 and 4) outperform algorithms based on the flow (Algorithms 1 and 2). On the other hand, these algorithms are more complex and require more memory resources than their corresponding flow versions.

The interpolation Algorithms 3 and 4 behave similar to their corresponding nearest neighbor algorithms, but they outperform the latters in the presence of divergent or continuous flow fields, like in Dimetrodon or Yosemite.

Fig. 12 compares the solution of a flow-based algorithm (Algorithm 1) with respect to an image-based algorithm (Algorithm 2). In the latter case, we note that some values may fail due to the sampling of the image, as can be seen in the last image. It appears some flow errors in the building because the intensities of the building get confused with the intensities in the background. This may worsen in natural sequences, where the presence of noise and brightness changes is more important. However, this problem only appears in the area of occlusions.

6.2 Street-lamp occlusions

Next, we study the performance of the algorithms with respect to the street-lamp occlusion. We modify two Middlebury sequences, Grove2 and Urban2, adding five static bars in front of the scene. This easily simulates the effect of the objects moving behind a fence.

In Fig. 13 we show the results for Grove2. The motion of the objects in this sequence is not big, so the effect of the bars is limited. Nevertheless, we appreciate the width of bars being smaller in the first result (Algorithm 1) than in the second solution (Algorithm 2). This means that the first algorithm has partially selected the motion of the background instead of the motion corresponding to the bars.

Table 3 presents the EPE and AAE errors for each algorithm. We observe that image-based algorithms clearly outperform flow-based ones. In fact, flow-based methods are unwilling to deal with this kind of occlusions.

Table 3: Reprojection errors for Grove2 with bars.

Error	**Algorithm 1**	**Algorithm 2**	**Algorithm 3**	**Algorithm 4**
EPE	0,035	**0,008**	0,033	*0,009*
AAE	0,947	**0,201**	1,048	*0,216*

Fig. 14 shows the results for the Urban2 sequence with bars. The motion in this sequence is bigger; therefore the effect on the bars is more important. The first algorithm is unable to detect the bars in the front building and they completely disappear in the backward flow. The second algorithm correctly handles the motion of the bars and the rest of objects.

Looking at Table 4, we observe that the EPE and AAE of image-based algorithms are much smaller than those of flow-based methods. Even, the fourth algoritm provides an important improvement in AAE with respect to the second algorithm.

Table 4: Reprojection errors for Urban2 with bars.

Urban2 (5 bars)	**Algorithm 1**	**Algorithm 2**	**Algorithm 3**	**Algorithm 4**
EPE	0,065	*0,011*	0,063	**0,011**
AAE	0,824	*0,163*	0,877	**0,151**

These experiments show that image-based algorithms correctly deal with any kind of occlusions. This is at the expense of using intensity information and an additional buffer for storing the comparisons between the image intensities.

6.3 Analysis of filling strategies

In this section, we study the behavior of the filling strategies explained in Section 5. The experiments in this section were performed using Algorithm 2.

Tables 5 and 6 show the EPE and AAE for the Middlebury sequences, respectively. We have only used the sequences where the flow is also defined at occlusions. If we compare these results with the results presented in Tables 1 and 2, we observe that the errors increased considerably.

Looking at these tables, we conclude that the best performing strategies are the oriented-fill and the min-fill approaches. The oriented-fill approach seems to have a better behavior. The average-fill strategy behaves better for the sequences with the static bars (two last sequences).

This means that this strategy is better for the street-lamp occlusions. This kind of occlusions is very difficult to deal with, because they produce a disocclusion at a place that

Table 5: Reprojection errors (EPE) for the filling strategies.

Sequence	Minimum fill	Average fill	Oriented fill
Grove2	**0,023**	0,039	0,040
Grove3	**0,166**	0,264	0,206
Urban2	0,083	0,130	**0,041**
Urban3	0,149	0,166	**0,055**
Venus	0,021	0,038	**0,017**
Yosemite	0,008	0,006	**0,005**
Grove2 with bars	0,092	**0,065**	0,094
Urban2 with bars	0,160	**0,132**	0,160

Table 6: Reprojection errors (AAE) for the filling strategies.

Sequence	Minimum fill	Average fill	Oriented fill
Grove2	$\mathbf{0{,}595}^o$	$1{,}010^o$	$1{,}014^o$
Grove3	$\mathbf{1{,}972}^o$	$3{,}417^o$	$2{,}761^o$
Urban2	$0{,}472^o$	$1{,}293^o$	$\mathbf{0{,}371}^o$
Urban3	$1{,}725^o$	$2{,}262^o$	$\mathbf{0{,}676}^o$
Venus	$0{,}325^o$	$0{,}791^o$	$\mathbf{0{,}284}^o$
Yosemite	$0{,}171^o$	$0{,}184^o$	$\mathbf{0{,}133}^o$
Grove2 with bars	$3{,}062^o$	$\mathbf{1{,}346}^o$	$3{,}112^o$
Urban2 with bars	$2{,}851^o$	$\mathbf{1{,}129}^o$	$2{,}878^o$

may be far from the object that originated it (see Fig. 14 for example). The minimum and the oriented strategies may not assure a correct filling.

In any case, dealing with the filling of the street-lamp case is not easy. An alternative would be to use a maximum-fill strategy, but there is no way to differentiate between the stereoscopic and the street-lamp occlusions. A good approach would be to use the oriented-fill for the stereoscopic and the average-fill for the street-lamp case, but we have not found an easy mechanism to differentiate between these two situations.

6.4 Recursive application

In this section, we study the effect of applying the algorithms multiple times. This will give us an idea of how the solutions degrade when we apply the methods recursively. For this experiment, we use the Yosemite and the Urban3 sequences.

In Tables 7 and 8 we show the EPE and AAE results for the Yosemite sequence, respectively. We compare the solutions applying the inverse optical flow 2 times ($\mathbf{h}^{2*}(\mathbf{x})$), 10 times ($\mathbf{h}^{10*}(\mathbf{x})$), 20 times ($\mathbf{h}^{20*}(\mathbf{x})$) and 100 times ($\mathbf{h}^{100*}(\mathbf{x})$). In this case, we use the average-fill strategy.

Table 7: Reprojection errors (EPE) for the Yosemite sequence with recursive application.

Error	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4
$\mathbf{h}^{2*}(\mathbf{x})$	0,011	0,006	0,006	**0,006**
$\mathbf{h}^{10*}(\mathbf{x})$	0,046	**0,011**	0,018	0,011
$\mathbf{h}^{20*}(\mathbf{x})$	0,094	**0,012**	0,037	0,014
$\mathbf{h}^{100*}(\mathbf{x})$	0,614	**0,013**	0,395	0,026

The results for Algorithms 2 and 4 are surprisingly very stable. In fact, the former

hardly degrades the results and, even computing the inverse flow 100 times, we obtain a very similar result. On the contrary, Algorithms 1 and 3 strongly degrade the solutions, the first one being the worst.

Table 8: Reprojection errors (AAE) for the Yosemite sequence with recursive application.

Error	**Algorithm 1**	**Algorithm 2**	**Algorithm 3**	**Algorithm 4**
$\mathbf{h}^{2*}(\mathbf{x})$	$0{,}321^o$	$0{,}182^o$	$0{,}184^o$	$\mathbf{0{,}155}^o$
$\mathbf{h}^{10*}(\mathbf{x})$	$1{,}141^o$	$0{,}287^o$	$0{,}458^o$	$\mathbf{0{,}265}^o$
$\mathbf{h}^{20*}(\mathbf{x})$	$2{,}163^o$	$\mathbf{0{,}316}^o$	$0{,}794^o$	$0{,}325^o$
$\mathbf{h}^{100*}(\mathbf{x})$	$12{,}101^o$	$\mathbf{0{,}335}^o$	$7{,}350^o$	$0{,}620^o$

Fig. 15 shows the results $\mathbf{h}^{10*}(\mathbf{x})$, $\mathbf{h}^{20*}(\mathbf{x})$ and $\mathbf{h}^{100*}(\mathbf{x})$ for Yosemite. In the first row, we show the results for Algorithm 1 and, in the second, the results for Algorithm 2. The last row shows that the latter image-based algorithm is very stable, while the first shows the unstability of flow-based algorithms.

Similar results can be seen in Tables 9 and 10 for the Urban3 sequence. In this case we have used the oriented-fill strategy.

Table 9: Reprojection errors (EPE) for the Urban3 sequence with recursive application.

Error	**Algorithm 1**	**Algorithm 2**	**Algorithm 3**	**Algorithm 4**
$\mathbf{h}^{2*}(\mathbf{x})$	0,058	0,053	0,055	**0,052**
$\mathbf{h}^{10*}(\mathbf{x})$	0,225	0,082	0,211	**0,082**
$\mathbf{h}^{20*}(\mathbf{x})$	0,438	**0,093**	0,409	0,093
$\mathbf{h}^{100*}(\mathbf{x})$	1,957	**0,100**	1,907	0,104

Table 10: Reprojection errors (AAE) for the Urban3 sequence with recursive application.

Urban3 (AAE)	**Algorithm 1**	**Algorithm 2**	**Algorithm 3**	**Algorithm 4**
$\mathbf{h}^{2*}(\mathbf{x})$	$\mathbf{0{,}575}^o$	$0{,}856^o$	$0{,}676^o$	$0{,}895^o$
$\mathbf{h}^{10*}(\mathbf{x})$	$1{,}973^o$	$\mathbf{1{,}351}^o$	$2{,}320^o$	$1{,}354^o$
$\mathbf{h}^{20*}(\mathbf{x})$	$3{,}750^o$	$1{,}551^o$	$4{,}288^o$	$\mathbf{1{,}526}^o$
$\mathbf{h}^{100*}(\mathbf{x})$	$12{,}026^o$	$1{,}693^o$	$12{,}446^o$	$\mathbf{1{,}600}^o$

We can see in Fig. 16 that the results of Algorithm 2 (second row) degrades less severely than the results of Algorithm 1 (first row).

These results seem reasonable, since flow-based algorithms use the flow information as discriminant. This information degrades with every inverse calculation, so the errors accumulate in every iteration. On the other hand, image-based algorithms rely on the image information, which is not modified.

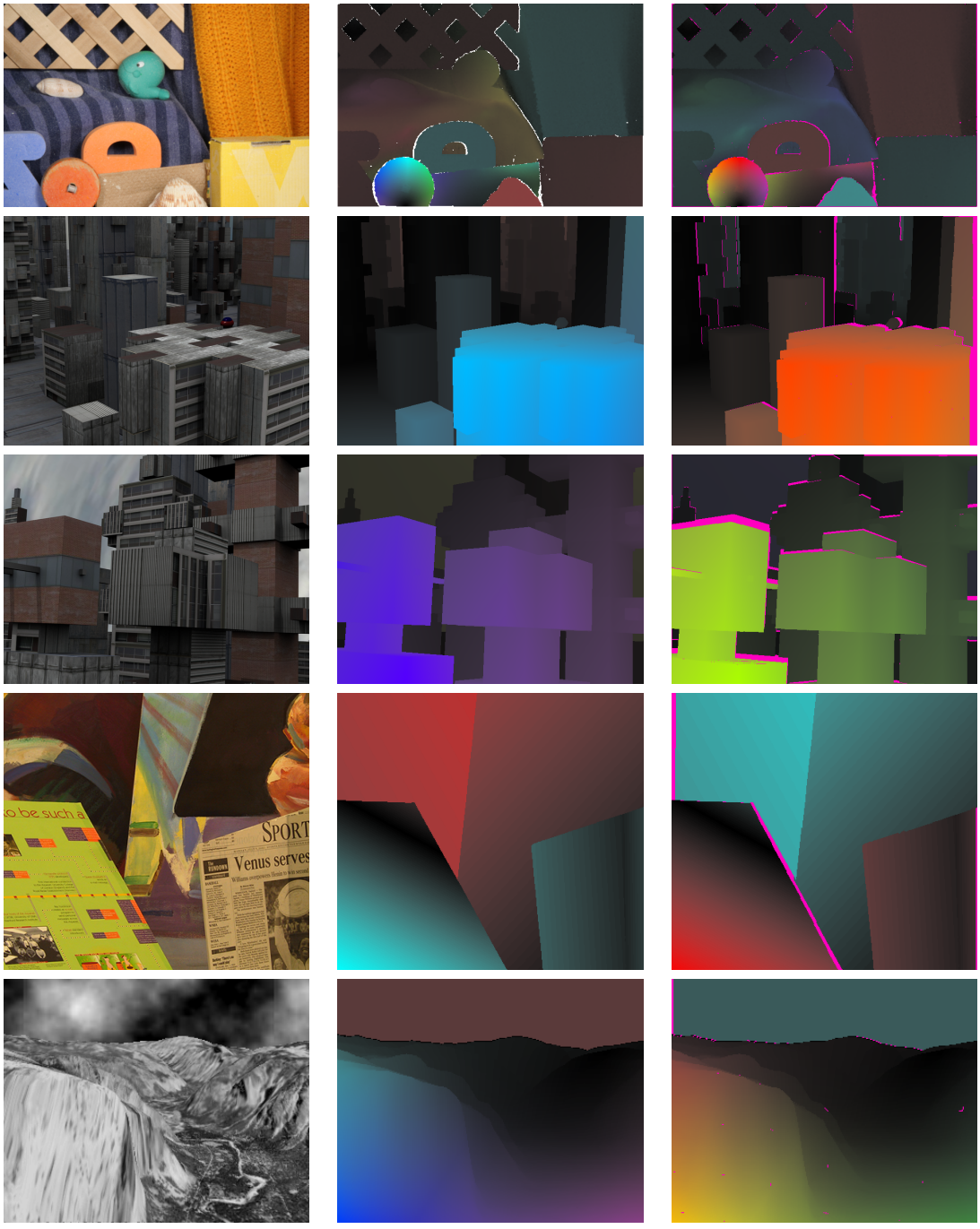

Figure 11: Middlebury test sequences. First column, the source image; second, the ground truth optical flow; third, the inverse optical flow using Algorithm 1, with disocclusions in pink.

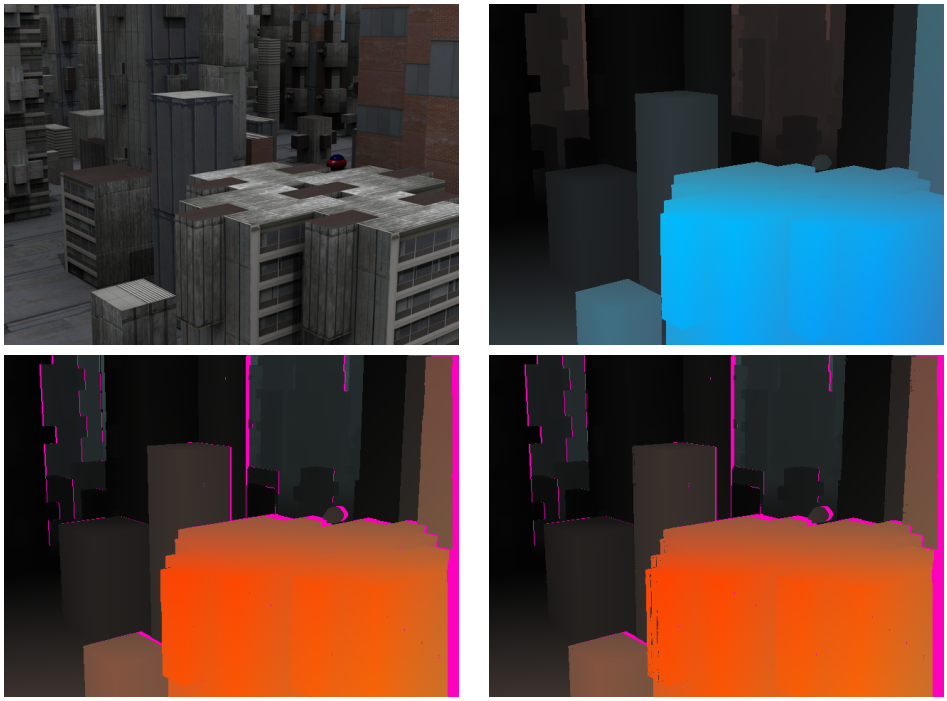

Figure 12: Flow-based versus image-based algorithms. Comparison between Algorithms 1 and 2.

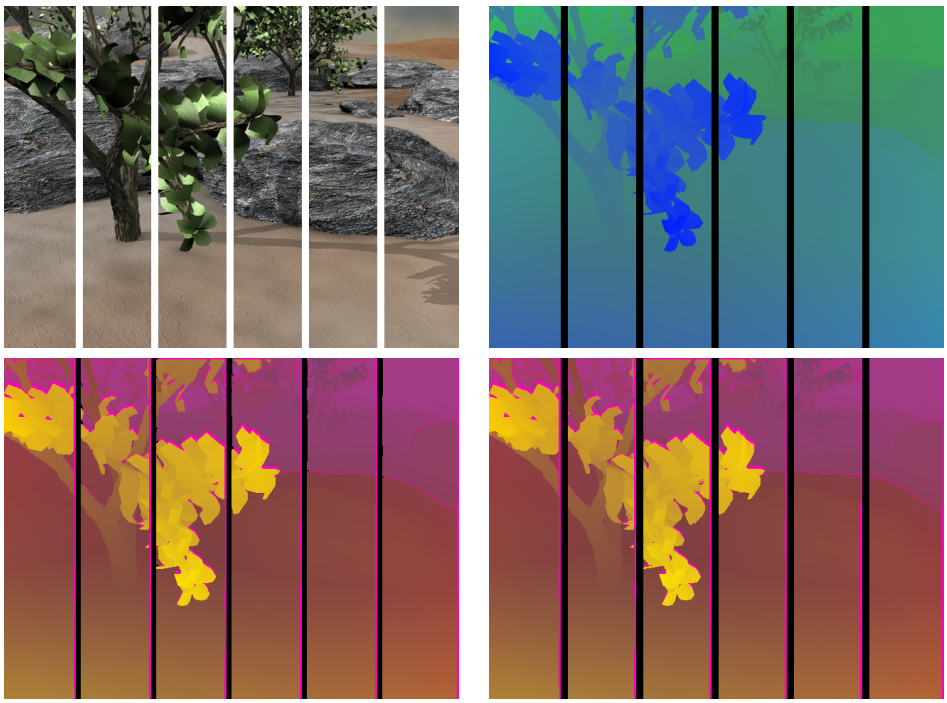

Figure 13: Street-lamp occlusion for Grove2. First row, the source image and the ground truth. Second row, the backward flows for Algorithms 1 and 2.

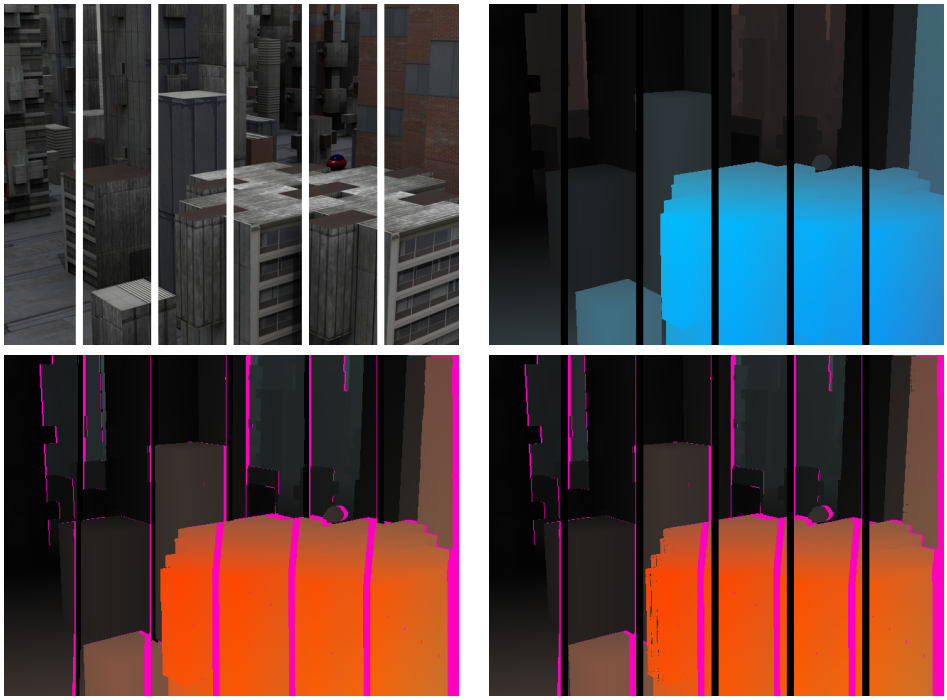

Figure 14: Street-lamp occlusion for Urban2. First row, the source image and the ground truth. Second row, the backward flows for Algorithms 1 and 2.

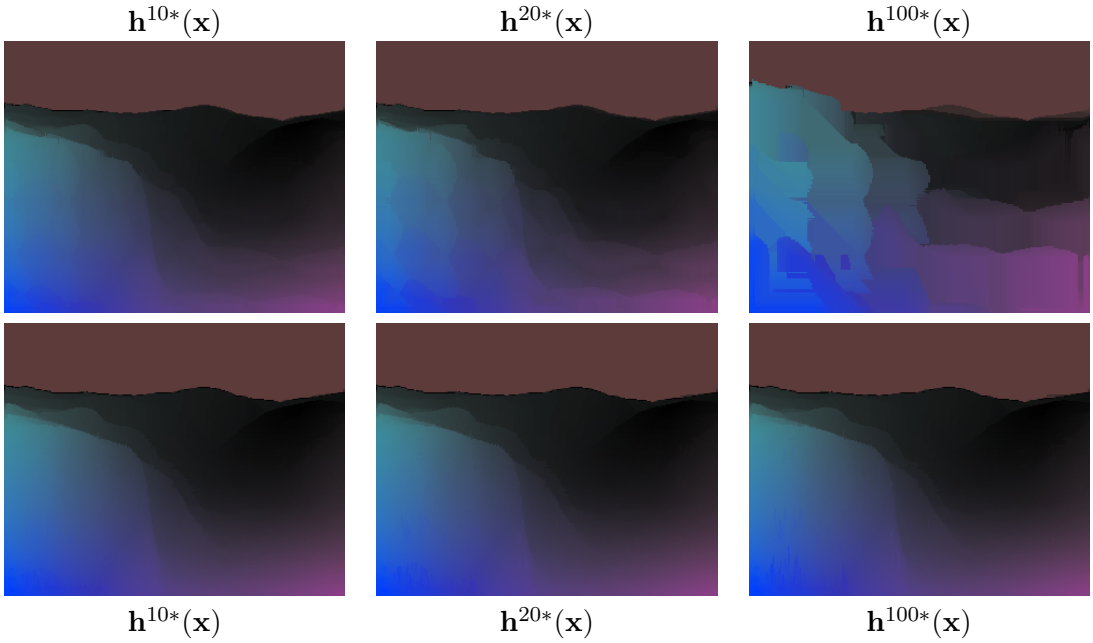

Figure 15: Yosemite recursive. First row, results for Algorithm 1. Second row, results for Algorithm 2.

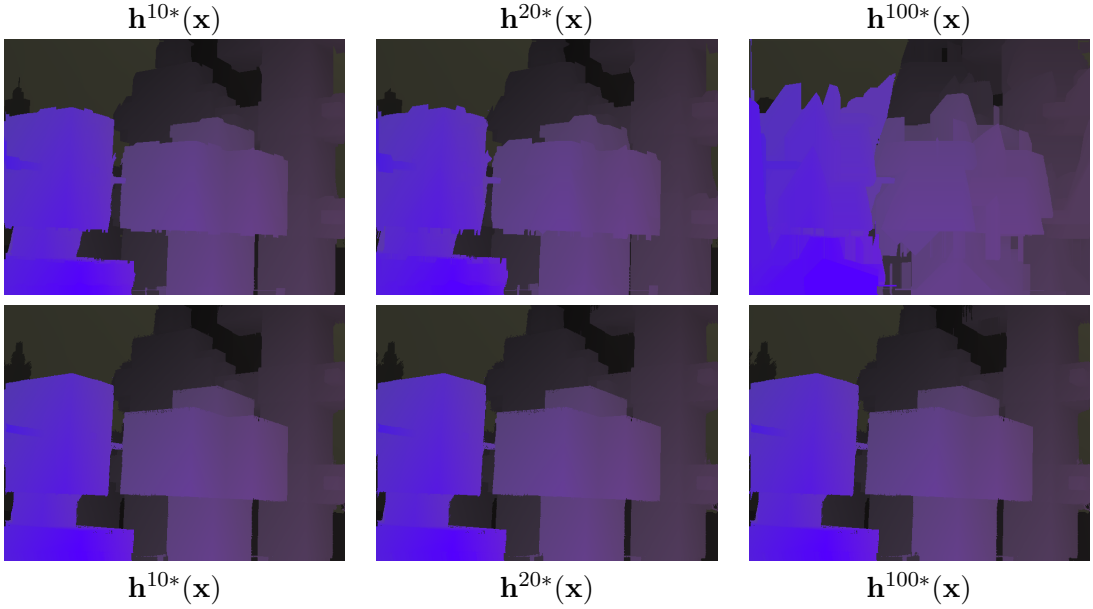

Figure 16: Urban3 recursive. First row, results for Algorithm 1. Second row, results for Algorithm 2.

7 Conclusions

In this work, we have proposed four new efficient algorithms for estimating the inverse optical flow. We have classified these in flow-based methods, that only use the forward optical flow as input, and image-based methods, that also use the information of the images. The former are limited for sequences with stereoscopic occlusions, whereas the latter are more general and can also deal with the street-lamp occlusion case.

We have seen the importance of occlusions and disocclusions in the estimation of the backward flow. All the algorithms can easily handle occlusions, but disocclusions have to be processed separately. In order to fill dissoclusions, we have proposed three filling strategies, based on the minimum and average flows, and an oriented filling process.

Flow-based algorithms are easy to implement and very efficient in running time and memory requirements. However, image-based algorithms yield the most accurate results, at the expense of using more buffers and calculations. Typically, we need a buffer for dealing with occlusions. On the other hand, nearest neighbor algorithms need one image pass, while interpolation algorithms need two image passes. The latter also need more buffers to store the accumulative values.

We have shown that the accuracy of the methods is very high in general. If we do not take into account disocclusions, then the errors are smaller than 1% of a pixel in Euclidean distance and less than 1^o in angular error. Errors increase considerably when we deal with disocclusions. The oriented fill strategy seems to provide the best results, followed by the min-fill strategy. Nevertheless, in the presence of street-lamp occlusions, the average fill strategy outperforms the other approaches.

We have also shown that image-based algorithms are also more stable. When we calculate the inverse flow many times, the results of flow-based methods rapidly degrade. This is due to the fact that the flow itself is modified and the errors are accumulated. This kind of methods might only be used when a very few iterations are needed. These are very interesting if speed or memory requirements are the first considerations. On the other hand, image-based algorithms are very stable, yielding very good results even with many inverse calculations.

Although the proposed algorithms are very efficient, they can still be improved. The major effort should be oriented to improve the filling strategies. This problem can be tackled using more temporal information, so that the unknown information can be discovered from a different time instant. Another improvement is to integrate the filling process with the main part of the algorithm, as it happens with occlusions.

Acknowledgment

This work has been partly founded by the Spanish Ministry of Science and Innovation through the research project TIN2011-25488.

References

[1] Luis Álvarez, Carlos Alberto Castaño, Karl Krissian, Luis Mazorra, Agustín J. Salgado, and Javier Sánchez. Symmetric Optical Flow. In Roberto Moreno Díaz, Franz Pichler, and Alexis Quesada Arencibia, editors, *Computer Aided Systems Theory - EUROCAST 2007*, volume 4739 of *Lecture Notes in Computer Science*, pages 676–683. Springer Verlag, Heidelberg, February 2007.

[2] Luis Álvarez, Rachid Deriche, Théodore Papadopoulo, and Javier Sánchez. Symmetrical dense optical flow estimation with occlusions detection. *International Journal of Computer Vision*, 75(3):371–385, 2007.

[3] J. Ashburner. A fast diffeomorphic image registration algorithm. *NeuroImage*, 38(1):95–113, 2007.

[4] J. Ashburner, J. Andersson, and K.J. Friston. Image registration using a symmetric prior - in three-dimensions. *Human Brain Mapping*, 9(4):212–225, 2000.

[5] Brian Avants and James C. Gee. Geodesic estimation for large deformation anatomical shape averaging and interpolation. *NeuroImage*, 23, Supplement 1(0):139 – 150, 2004.

[6] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. In *International Conference on Computer Vision*, pages 1–8, 2007.

[7] John L. Barron, David J. Fleet, and Steven S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, 1994.

[8] Pascal Cachier and David Rey. Symmetrization of the non-rigid registration problem using inversion-invariant energies: Application to multiple sclerosis. In ScottL. Delp, AnthonyM. DiGoia, and Branislav Jaramaz, editors, *Medical Image Computing and Computer-Assisted Intervention MICCAI 2000*, volume 1935 of *Lecture Notes in Computer Science*, pages 472–481. Springer Berlin Heidelberg, 2000.

[9] G.E. Christensen and H.J. Johnson. Consistent image registration. *IEEE Transactions on Medical Imaging*, 20(7):568 –582, july 2001.

[10] K.J. Friston, J. Ashburner, S.J. Kiebel, T.E. Nichols, and W.D. Penny, editors. *Statistical Parametric Mapping: The Analysis of Functional Brain Images*. Academic Press, 2007.

[11] Alexandre Guimond, Jean Meunier, and Jean-Philippe Thirion. Average brain models: A convergence study. *Computer Vision and Image Understanding*, 77(2):192 – 210, 2000.

[12] Alex D. Leow, Sung-Cheng Huang, Alex Geng, James T. Becker, Simon W. Davis, Arthur W. Toga, and Paul M. Thompson. Inverse consistent mapping in 3d deformable image registration: Its construction and statistical properties. In Gary E. Christensen and Milan Sonka, editors, *Information Processing in Medical Imaging, 19th International Conference, IPMI 2005, Glenwood Springs, CO, USA, July 10-15, 2005, Proceedings*, volume 3565 of *Lecture Notes in Computer Science*, pages 493–503. Springer, 2005.

[13] David Lieb, Andrew Lookingbill, and Sebastian Thrun. Adaptive road following using self-supervised learning and reverse optical flow. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.

[14] A. Lookingbill, J. Rogers, D. Lieb, J. Curry, and S. Thrun. Reverse optical flow for self-supervised adaptive autonomous robot navigation. *International Journal of Computer Vision*, 74:287–302, 2007.

[15] J.B.A. Maintz and M.A. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1):1–36, 1998.

[16] Agustín Salgado and Javier Sánchez. A temporal regularizer for large optical flow estimation. In *IEEE International Conference on Image Processing ICIP*, pages 1233–1236, 2006.

[17] Javier Sánchez, Agustín Salgado, and Nelson Monzón. Direct estimation of the backward flow. In *International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 268–274. Institute for Systems and Technologies of Information, Control and Communication, 2013.

[18] Javier Sánchez, Agustín Salgado, and Nelson Monzón. An efficient algorithm for estimating the inverse optical flow. In *6th Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA)*, pages 390–397. Springer-Verlag, 2013.

[19] Javier Sánchez, Agustín Salgado, and Nelson Monzón. Optical flow estimation with consistent spatio-temporal coherence models. In *International Conference on Computer Vision Theory and Applications (VISAPP)*, pages 366–370. Institute for Systems and Technologies of Information, Control and Communication, 2013.

[20] Deshan Yang, Hua Li, Daniel A Low, Joseph O Deasy, and Issam El Naqa. A fast inverse consistent deformable image registration method based on symmetric optical flow computation. *Physics in Medicine and Biology*, 53(21):6143, 2008.

[21] Sai-Kit Yeung, Chi-Keung Tang, Pengcheng Shi, J. P W Pluim, M.A. Viergever, A.C.S. Chung, and H.C. Shen. Enforcing stochastic inverse consistency in non-rigid image registration and matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[22] B. Zitová and J. Flusser. Image registration methods: A survey. *Image and Vision Computing*, 21(11):977–1000, 2003.

Centro de Tecnologías de la Imagen
Universidad de Las Palmas de Gran Canaria
http://www.ctim.es