# CHAPTER 8

# Optical Flow

## 1. Image Sequences

It is said that a picture is worth a thousand words. Whoever said this was overly optimistic about image compression technology, but assuming it is correct we can only imagine the worth of a sequence of images.

A digital camera can record image sequences from dynamic scenes and forward them to a computer for processing and interpretation. The amount of raw data contained in such a sequence is immense and it can overwhelm many a computer. But there is some information in this data that cannot be obtained from still images namely the history of the scene and most important, the structure of the scene and this makes the image sequences worth their while.

The word *structure* usualy refers to the depth, in the form of a *Z-map*, but it can also mean any other representation of the depth of the scene, like decomposition of the scene in piecewise continuous patches approximated by splines or a set of depth values at distinct points.

The images recorded from a dynamic scene are neither identical nor totally different from each other but have a few certain kinds of differences. The factors that contribute to these differences are

*Brightness changes*
The overall brightness of an object in a scene can change due to change in amount of ambient light (e.g. when lights go on and off or move), motion of shadows, change in reflectance of the surface of the object (e.g. when the wind upsets the fur of a lady, or the surface gets wet), motion of specularities etc. An important contributor in the brightness fluctuation is of course the random noise present in all cameras..

*Drop-ins and drop-outs*
As the camera or the object move, things either fall off the border of the image, or new things come into view after crossing the same borders. Moreover when one object occludes another then part of the occluded object may come into view or a visible object may hide behind another due to the motion of the camera or the objects themselves.

*Projection of 3-D motion*
As the camera moves, the images of the objects move on the focal plane too. Their motion is the projection of the 3-D motion with respect to the camera coordinate system.

*Change of camera parameters*

The camera parameters can change and create effects similar to change in brightness (change of *f*-number) or motion (zoom-in and zoom-out) but can have other effects like change of focus, change of colormap etc.

These are the most common factors of change in the image of a dynamic scene and from all these only the projection of the 3-D motion provides information that can be utilized for the recovery of the structure of the environment. The information provided by the other factors, while relevant to the structure of the scene, are a nuisance rather than help. The shadows, for instance, can help a human viewer interpret a scene, but there is no applicable mathematical model or an algorithm that can help a computer do the same thing. For this reason researchers in the Computer Vision community refer to these factors simply as *noise* although it would be more honest to say *that thing we have no clue how to model*.

## 1.1. Optical Flow vs. Image Flow

There are many applications where the main difference between successive images in a sequence is due to the projection of the 3-D motion which means we know how to model it and as a result we know how to extract the structure in these applications. This projection of the 3-D motion is called *Optical Flow*.

Unfortunately, the light intensity patterns on the image move according to the projection of the 3-D motion, but this does not mean that we can recover optical flow from these images. Consider the image of a spinning white wheel which appears stationary. Its optical flow is non zero, but it appears to be zero.

While optical flow is unique, the flow that appears on the image is not always unique but this is all we have. This flow we call *Image Flow* to distinguish from the optical flow and the best we can hope for is to recover an image flow and hope that it is a good approximation to the optical flow. In practice, it is just one of the difficulties that make the motion problem interesting, and there are many applications that this is not a problem.

## 1.2. Any Hope for Motion

It appears from the above that the motion problem is almost unsolvable, and indeed it is one of the hardest in Computer Vision. But as opposed to many other problems in vision and AI in general, it has a clear statement, can be modeled mathematically to a large extend and the assumptions can be stated clearly. This is not the case with edge detection, object recognition or texture classification where the research community found the hard way that these problems are much harder than previously thought. At least motion, we have a good enough mathematical understanding of the problem to be fully aware of the complexities.

## 2. Differential Formulation

We will attempt to form a set of equations for optical flow by restricting ourselves to an easier version of the problem. Then we are going to ease these restrictions one after the other until we get a practical method to determine flow from a sequence of real world images.

## 2.1. Assumptions

We are going to introduce a set of assumptions that will make a differential formulation possible. Some of these assumptions will be stated quantitatively with a formula, others will be in a qualitative way to indicate that a certain kind of approximation is valid.

### 2.1.1. Infinitesimal Flow

The first assumption is that the interframe motion is small enough to be considered infinitesimal. In practice this usualy means less than a pixel, but it depends on the amount of information carried in the image. If the image contains fine texture then one pixel is indeed the upper limit. If the image contains a smooth intensity pattern that varries very little from one pixel to the next, then we can have larger motion and still call it infinitesimal.

While this is an assumption that we will ease later, it is not unbearably restrictive. With 30 frames per second a camera can take sequences that satisfy this assumption and still have motion of about 30 pixels a second, which is enough to do many things. Not all of course. Consider for example a $512 \times 512$ camera with 90 degrees field of view. It would need 2048 frames to do a whole circle, or about 70 seconds. Certainly Nurejev did not satisfy this assumption (even Pavaroti could beat that while eating pizza).

### 2.1.2. Constant Intensity Assumption

We have to assume that the intensity of the projection of a 3-D point does not change from one frame to the next. If a 3-D point registers an intensity of say 128 on our image, then this is the intensity this point is going to have in all subsequent frames. This assumption can be stated as

$$dI(x, y, t) = 0 \tag{2.1}$$

where $I$ is the image intensity which is a function of the image coordinates and time. This simply means that the intensity remains the same and it will be the starting point for the subsequent derivations.

This assumption holds in environments where we can control the illumination conditions but it is only an approximation when we deal with general environments. We will briefly touch upon methods that attempt to relax this assumption a bit (it does not make sense to relax it completely, because there has to be some relation among the images of a sequence).

### 2.1.3. No Discontinuities

This is perhaps the most painful assumption. The first reason this being so is that there is no easy way to define discontinuities on a discrete quantity like the optical flow we aim to compute. While the optical flow is a continuous quantity, we can only represent it as discrete even if we knew it. So there can be no algorithm for detecting discontinuities, only heuristic approximations. The second reason that this is painful is that, unlike other quantities that are functions of one or more independent variables, like temperature versus time, where the notion of discontinuity is introduced as an approximation to a continuous and smooth quantity, discontinuities do exist in the optical flow. They are the borders of objects occluding other objects in the scene. Which means that the discontinuities are the most information rich regions of an image.

This assumption is also one of the most difficult to deal with. All attempts to relax it result in much slower algorithms. We will only briefly touch the subject.

Finally if we cannot even define the discontinuities, then how can we require their absence? If we buy a sequence full of discontinuities, how do we know if we should ask for our money back? The answer is how we state the assumption. It turns out the most convenient way is to say that we attempt the estimation of the optical flow only on areas where the flow varies smoothly.

### 2.1.4. Other Assumptions

There are several other assumptions that we introduce silently because they are natural to the problem, like the requirement that the image is an adequate discretization at least in the $x$ and $y$ directions so that we do not have aliasing phenomena.

## 2.2. One Dimensional Image

We start by studying the simplest case, that of a 1-D image moving more or less uniformly. We are given the image sequence and we have to find the flow $u$ at every pixel of the one dimensional image. At some point $x$ before the motion the intensity is $I(x,t)$ and at the same point $x$ after the motion the intensity is $I(x,t+\delta t)$ where $\delta t$ is the elapsed time. The original point moved to $x+\delta x$ where $\delta x$ is the displacement and from the constant intensity assumption we know that this point did not change intensity so that $I(x,t) = I(x+\delta x, t+\delta t)$. The only unknown here is $\delta x$. All the other quantities are either images and their derivatives or time, which we assume given.

If we approximate the curves with their tangents (which is OK since we assume infinitesimal motion) we can solve the triangle $[x, I(x,t)]..[x, I(x,t+\delta t)]..[x+\delta x, I(x+\delta x, t+\delta t)]$, which is marked with bold lines in Fig. 2.1) and we can find that $\delta x = \dfrac{I(x,t)-I(x,t+\delta t)}{I_x(x,t)}$. We can manipulate this expression to get something more elegant if we notice that

$$I(x,t) - I(x,t+\delta t) = \delta t I_t(x,t)$$

where $I_t$ is the time derivative of the image intensity. We also substitute the flow $u$

$$u = \frac{\delta x}{\delta t}$$

and we finally get

$$I_x u + I_t = 0 \tag{2.2}$$

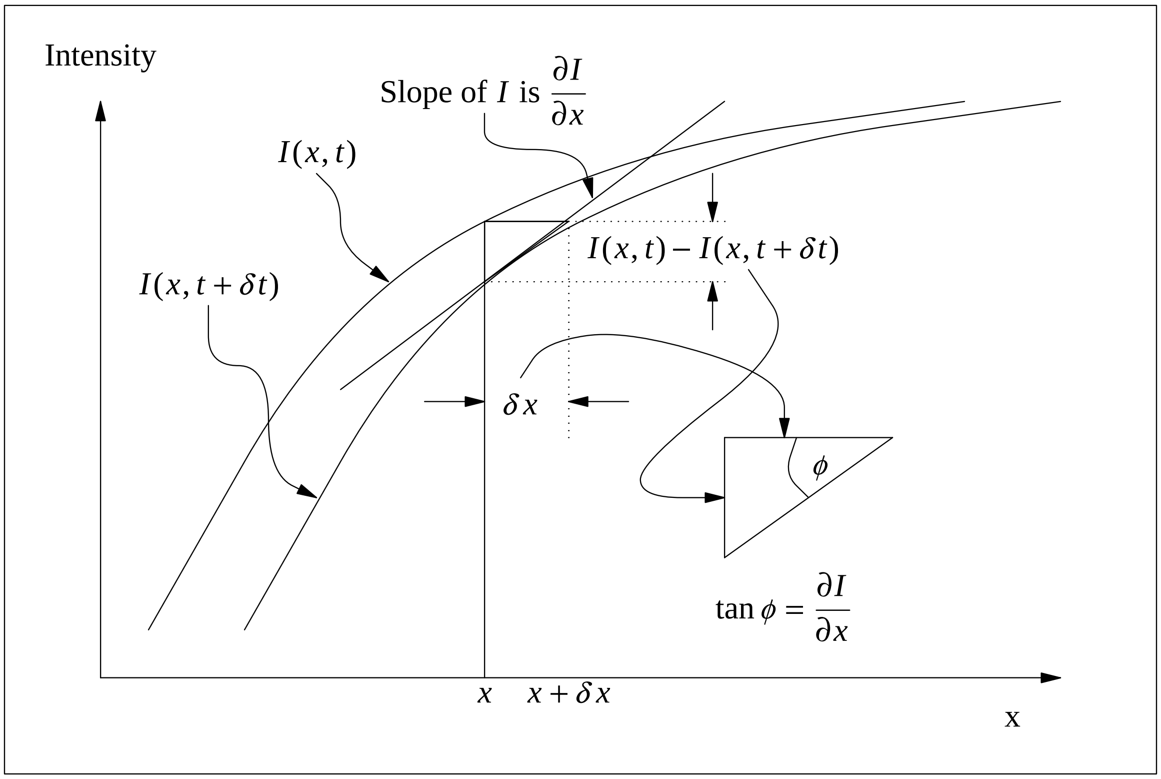

**Figure 2.1**: The one dimensional image $I(x,t)$ moves to the right by $\delta x$. We can find $\delta x$ by solving the triangle in bold lines (shown also in magnification near the lower right of the figure). It can be shown that $\delta x$ satisfies the equation $I_x(x,t)\delta x = I(x,t) - I(x,t+\delta t)$

This is the *Optical Flow Equation* for one dimensional images. This is easy to extend to two dimensions.

## 2.3. Two Dimensional Image

The same exactly procedure could be applied to derive the two dimensional case too. But since repetition of the same procedure would not offer any more insight, we will present a more algebraic derivation.

We will start from the *Constant Intensity Assumption* (Eq. (2.1))

$$dI(x,y,t) = 0$$

which can be written as

$$dI(x,y,t) = \frac{\partial I}{\partial x}\,dx + \frac{\partial I}{\partial y}\,dy + \frac{\partial I}{\partial t}\,dt$$

and by dividing both sides by *dt* we get

$$I_x u + I_y v + I_t = 0 \tag{2.3}$$

where $v = \frac{dx}{dt}$. This is the *Optical Flow Equation* which is often written in vector form

$$\nabla I \cdot \mathbf{u} + I_t = 0$$

where

$$\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$$

and

$$\nabla I = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial x} \end{bmatrix}.$$

### 2.4. Effects of the Assumptions

We derived the optical flow equation (Eq. (2.3)) under several assumptions, which is bound to create some problems. Let' s see what are these problems and what kinds of remedies we can devise for them. We will see two kinds of problems, ones that are related to the fact that the image is two dimensional and we have to study them as such and the ones that apply under similar circumstances to one dimensional images as well and we will study them in one dimension only to keep things simple.

#### 2.4.1. Equation Deficit, Aperture Problem

The most fundamental problem which is directly related to the pointwise application of the intensity constancy assumption is the so called *Aperture Problem*. It is easy to notice that Eq. (2.3) has two unknowns while it is just a single scalar (e.g. non-vector) equation. Thus we know that it does not have a unique solution.

In general if a point on the image has a certain value, then, unless it is a local extremum, there should be other points in its neighborhood that have the same value. This might not be obvious on the discretized image, but it is true for the continuous image that is represented by the discretized. All these points lie on a line which is called an isophote, e.g. a line where the intensity is constant.

When the intensity pattern on the image deforms and moves, the isophotes move along with it of course. But since Eq. (2.3) tracks the intensity it is natural that it cannot distinguish between points on the same isophote over time. So the lack of uniqueness in the solution is indeed the result of the constant intensity assumption and not an artifact of the way we manipulated the equation.

##### 2.4.1.1. Region Matching

The only way to solve the apperture problem is to augment the constant intensity assumption. One such way is to apply *Region Matching* and match small regions rather than single points. In order to do this we have to invoke one of the assumptions that we did not really use so far that of image smoothness.

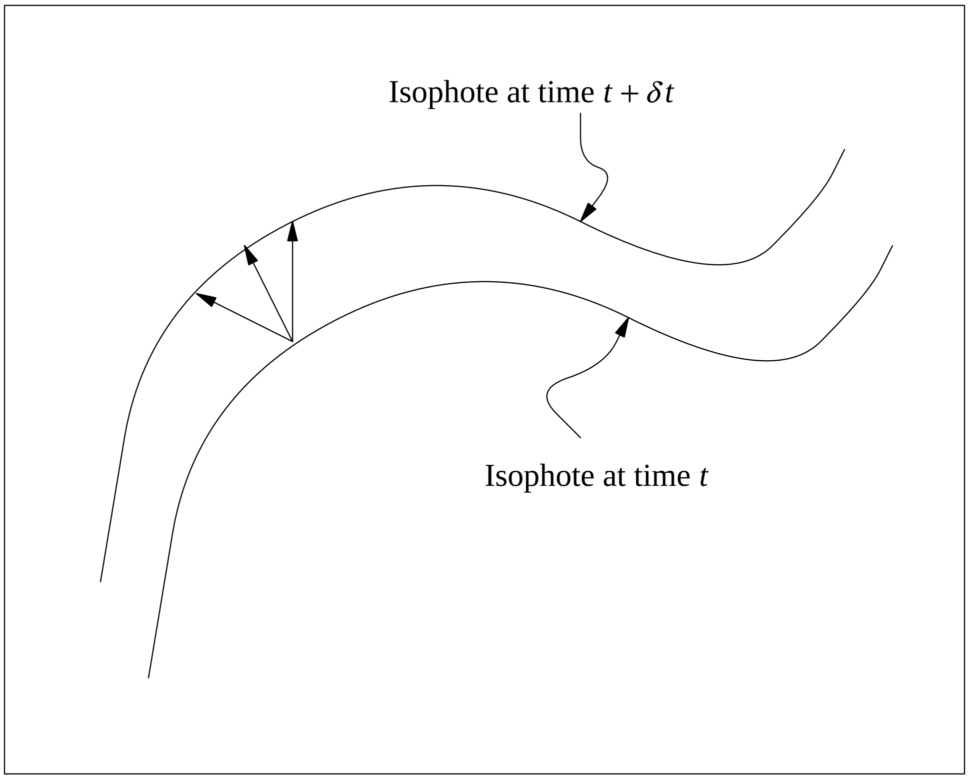

**Figure 2.2**: The future position of a point along an isophote is indeterminate.

The fact that the image is smooth implies that the flow at one pixel is more or less the same with the flow of its neighbors. If we carry this observation one step further and say that the flow (e.g. $u$ and $v$) is the same on a pair of pixels, then for every such pair we have two unknowns (the common $u$ and $v$) and two equations (by applying Eq. (2.3) on each one of them) and we can solve the system uniquely.

We still have some small difficulties to overcome but we are close. The main difficulty is that it is rather unlikely that the two equations will always be independent. What if we always choose the left neighbor and we happen to be on a horizontal isophote? We need then to combine the equations of a larger region, say $3 \times 3$ or $5 \times 5$, to increase our chances to have two independent equations but then we have more equations than unknowns. Luckily there is a very elegant way to solve this problem: it is called *Least Squares*.

### 2.4.1.1.1. Least Squares

The basic idea of the least squares is simple. When we have more equations than unknowns and the coefficients of these equations are noisy and unreliable, we have to combine them in a way that reduces the effects of the noise, prety much the same way we combine multiple uncertain measurements to get the average which is, in general more reliable. But how do we average equations?

The first step in applying least squares is to bring all the equations in the form $f_k(x_1, x_2, \cdots) = 0$, which in our case are already in this form.

$$I_x[i_0 - i, j_0 - j]u[i_0, j_0] + I_y[i_0 - i, j_0 - j]v[i_0, j_0] + I_t[i_0 - i, j_0 - j] = 0$$

where $i_0$ and $j_0$ are the coordinates of the center of the region we consider and $i$ and $j$ range within $-1..1$ for a $3 \times 3$ region or within $-2..2$ for a $5 \times 5$ etc. The unknowns are $u[i_0, j_0]$.

The second step is to square all these equations and sum them up. This sum is always a non negative quantity. If could find a pair of $u$ and $v$ that could satisfy all the equations simultaneously then the sum would be zero, otherwise it is positive. It is easy to see that the closer we are to satisfy all the equations at the same time the closer to zero the sum should be. Our goal then is to find the $u$ and $v$ that make this sum minimum.

The sum $S$ can be written as

$$S[i_0, j_0] = \sum_{i,j \text{ in } R} \left\{ I_x[i_0 - i, j_0 - j]u[i_0, j_0] + I_y[i_0 - i, j_0 - j]v[i_0, j_0] + I_t[i_0 - i, j_0 - j] \right\}^2$$

where $R$ is the region centered at $i_0, j_0$. The above expression is fine but weighs the pixels close to the center of the region the same as the pixels further away. It is obvious that we could do a bit better if we weighed the pixels near the center heavier than those away from it. This can be done by introducing a template $w[i, j]$ that has the appropriate form (usualy bell shaped).

$$S[i_0, j_0] =$$

$$\sum_{i,j} w[i, j] \left\{ I_x[i_0 - i, j_0 - j]u[i_0, j_0] + I_y[i_0 - i, j_0 - j]v[i_0, j_0] + I_t[i_0 - i, j_0 - j] \right\}^2 \quad (2.4)$$

We can minimize $S$ the way we minimize any function, by taking the derivatives with respect to the unknowns $u$ and $v$ and setting them to zero. We start with the derivative with respect to $u$.

$$\frac{\partial}{\partial u[i_0, j_0]} S[i_0, j_0] = \frac{\partial}{\partial u[i_0, j_0]} \sum_{i,j} w[i, j]\{I_x[i_0 - i, j_0 - j]u[i_0, j_0] +$$

$$I_y[i_0 - i, j_0 - j]v[i_0, j_0] + I_t[i_0 - i, j_0 - j]\Big\}^2$$

$$\sum_{i,j} w[i, j] \frac{\partial}{\partial u[i_0, j_0]} \left\{ I_x[i_0 - i, j_0 - j]u[i_0, j_0] + I_y[i_0 - i, j_0 - j]v[i_0, j_0] + I_t[i_0 - i, j_0 - j] \right\}^2 =$$

$$\sum_{i,j} w[i, j]2I_x[i_0 - i, j_0 - j]\left\{ I_x[i_0 - i, j_0 - j]u[i_0, j_0] + I_y[i_0 - i, j_0 - j]v[i_0, j_0] + I_t[i_0 - i, j_0 - j] \right\} =$$

$$2\left\{\left(\sum_{i,j} w[i, j]I_x[i_0 - i, j_0 - j]^2\right)u[i_0, j_0] + \left(\sum_{i,j} w[i, j]I_x[i_0 - i, j_0 - j]I_y[i_0 - i, j_0 - j]\right)v[i_0, j_0] +\right.$$

$$\left.\left(\sum_{i,j} w[i, j]I_x[i_0 - i, j_0 - j]I_t[i_0 - i, j_0 - j]\right)\right\}.$$

It is easy to notice that the summations are actually convolutions

$$\sum_{i,j} w[i, j]I_x[i_0 - i, j_0 - j]^2 = I_x{}^2(*)w$$

and similarly for the rest. We can rename these quantities

$$E_{xx} = I_x{}^2(*)w$$
$$E_{xy} = I_x I_y(*)w$$
$$E_{xt} = I_x I_t(*)w.$$

So finally the equation becomes

$$E_{xx}u + E_{xy}v + E_{xt} = 0.$$

If we apply the same derivative with respect to $v$ we get

$$\begin{bmatrix} E_{xx} & E_{xy} \\ E_{xy} & E_{yy} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} E_{xt} \\ E_{yt} \end{bmatrix}. \tag{2.5}$$

The solution of this system is very simple because it is a $2 \times 2$ system and the coefficients are easy to compute because the involve only a a few simple operations like convolution, multiplication etc. This is the heart of the *Lucas and Kanade* algorithm, which performs extremely well. In fact it outperforms many much more sophisticated and computationally expensive algorithms.

### 2.4.1.1.2. Algorithm

Among the nicest things about this algorithm is how easy it is to implement. Assume that the two successive images are `im1` and `im2` we compute the $E_{xx}$ etc first:

```
gtmpl = mk_gauss_tmpl(sigma);
Ix = D_x(im1);
Iy = D_y(im1);
It = im2 - im1;
Exx = ( Ix * Ix ) (*) gtmpl;
Exy = ( Ix * Iy ) (*) gtmpl;
Eyy = ( Iy * Iy ) (*) gtmpl;
Ext = ( Ix * It ) (*) gtmpl;
Eyt = ( Iy * It ) (*) gtmpl;
```

and then invert the matrix using Kramer's rule (or anything else that is convenient). Instead of having a double `for` loop to invert the matrix at every pixel we apply the corresponding operations to `Exx`, `Exy` etc which are images

```
det = - 1.0 / ( Exx*Eyy - Exy*Exy );
u = (Eyy*Ext - Exy*Eyt)*det;
v = (-Exy*Ext + Exx*Eyt)*det;
```

### 2.4.1.2. Color

Since the apperture problem is caused mainly from the equation deficit, any introduction of new independent equations would solve it. One such source is color. We can get one independent equation per color band and combine them with least squares.

Unfortunately, this is more interesting as an application of the least squares idea in the context of optical flow, rather than practical. There are several reasons for that. One is that the colors do not change independently of each other, so they tend to produce interdependent equations. Second, cameras tend to degrade color terribly because most of them are designed to produce images to be viewed by humans and not analyzed by machines. Humans have much lower color resolution than gray resolution and do not mind the color degradation. But this will make the resulting optical flow equations even more interdependent. Nevertheless, the use of color in controlled situations, where it lighting can be engineered in a way that produces mostly independent equations, is beneficial. And we can handle it the same way we handled region matching.

Proceeding as before, we get exactly the same equations with the exception that $E_{xx}$, $E_{xy}$ etc are now

$$E_{xx} = \sum_{c\ in\{r,g,b\}} {I^c}_x^{\,2}$$

$$E_{xy} = \sum_{c\ in\{r,g,b\}} {I^c}_x {I^c}_y$$

$$E_{yy} = \sum_{c\ in\{r,g,b\}} {I^c}_y^{\,2}$$

$$E_{xt} = \sum_{c\ in\{r,g,b\}} {I^c}_x {I^c}_t$$

$$E_{yt} = \sum_{c\ in\{r,g,b\}} {I^c}_y {I^c}_t.$$

These equations are not more complicated than the previous ones and the least squares technique proves once again its elegance and power. Although it is a simple way to average equations, its power is astonishing.

### 2.4.2. Higher Order Effects

One of the assumptions we did was that the motion is assumed infinitesimal. The effects of this assumption can be seen even when the motion is as small as it can practically be. Consider the following case, where we have an 1-D smooth bell-like curve moving to the right, by a small amount (Fig. 2.3). The curve before and after the motion are shown at the top of the picture. The effects we will see hold in the same manner in 2-D.

A simple application of Eq. (2.3) will give an acceptable result in all cases except the peak in the middle of the curve. At this point the derivative $I_x$ (second curve) is crossing zero which will produce a mathematical error. This is not a case of mathematical indeterminacy, which means that our formulas cannot produce a valid answer, because the time derivative $I_t$ (third curve) is not zero so we do not have a zero by zero division.

The problem of the mathematical error is fairly easy to solve using the region matching technique of the previous section. But the answer we get (bottom curve) is misleading near the peak. While the correct answer should be in this case a constant number, the curve has a wild swing and it even produces motion in the opposite direction. This is a very annoying phenomenon, because it appears on an almost ideal situation of a very smooth curve, with very small motion and no noise.

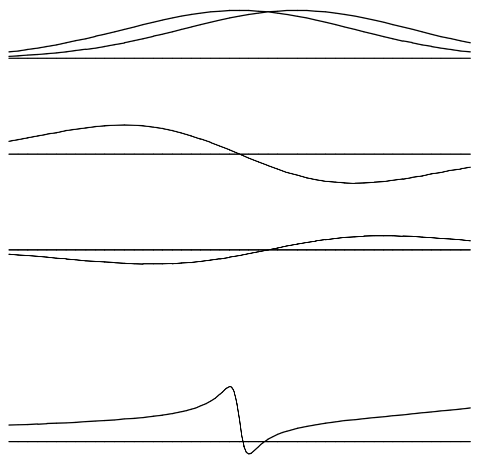

**Figure 2.3**: The top graph shows an 1-D signal before and after the motion. The second graph is the spatial derivative of the signal, the third the time derivative and the forth is the computed flow using 3 pixel wide regions.

The heart of the problem is that at the peak of the curve, the equations should be indeterminate. But due to the asymmetry in the way we compute the spatial derivative $I_x$ by applying the derivative operator on either the image before the motion or after, the zeros of the numerator (the time derivative $I_t$) and the denominator (spatial derivative $I_x$) do not coincide, so the indeterminacy does not materialize. If instead we use as spatial derivative the average of the spatial derivatives before and after the motion then we get much better results (Fig. 2.4). The improvement of the accuracy of the algorithm is astonishing.

### 2.4.3. Temporal Aliasing

So far we have assumed that the motion is small and does not produce more than about one pixel optical flow. Under these circumstances we can assume that the motion is infinitesimal in practice. But we do not always have the luxury of so small motion. So we have to see what is the result of the violation of this assumption and how it can be dealt with.

Consider an image with a fine texture like a plaid cloth, a brick wall, a pebble beach, tree foliage etc. All these textures have the property that the same pattern is repeated although it is not repeated in a strictly periodic sense. Let's say tha for the sake of example that the same pattern is more or less repeated every ten pixels. If the interframe flow is also around the same value then any flow algorithm that matches intensity patterns of pixels or regions, will be confused.

Take for instance the case of a pebble beach where we want to find the flow at a pixel that corresponds to the center of a particular pebble. If the next frame of the image sequence has moved about 10 pixels so that another pebble occupies the same space, then

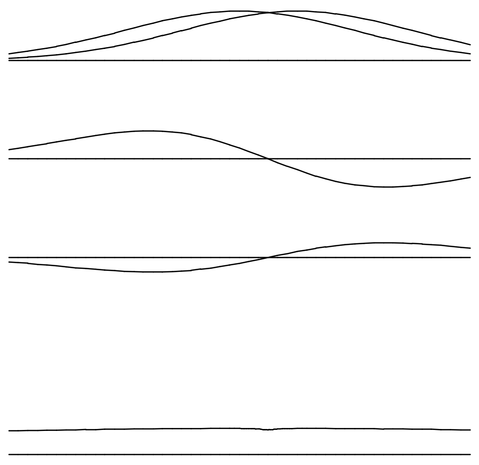

**Figure 2.4**: If we use a more symmetric spatial derivative the resulting flow is much closer to a constant than before.

flow algorithms of the kind that we saw so far will match the old pebble with the new pebble and return as result a flow that is closer to zero rather than ten.

This problem is not particular to two dimensions but works the same way in one dimension. To see it better consider a smooth curve that has two peaks (Fig. 2.5) and moves enough pixels, so that the one hump goes near the place where the other hump was before. It is obvious that a simple algorithm that just looks at intensities, will confuse the two humps and create a false match. This false match will appear in this case as flow in the wrong direction (the part of the flow that is negative). The finer the detail and the larger the motion the easiest it is for this problem to arise.

This is another problem that has a quite good solution. The idea is very simple. Since in practice an optical flow that is less then one pixel is small, then if we have large flow we can shrink the picture to half the original size and apply the algorithm on the smaller scale where the flow is half the original size (Fig. 2.6. Then when we solve the problem, we can use this low resolution solution as a guess to get a head start to the original problem.

When we are at the lower resolution, the problem is easier but not always readily solvable. But since we have discovered a low resolution hammer, everything looks like a nail. We can apply the technique recursively until we go to a very coarse resolution where we apply our favorite algorithm.

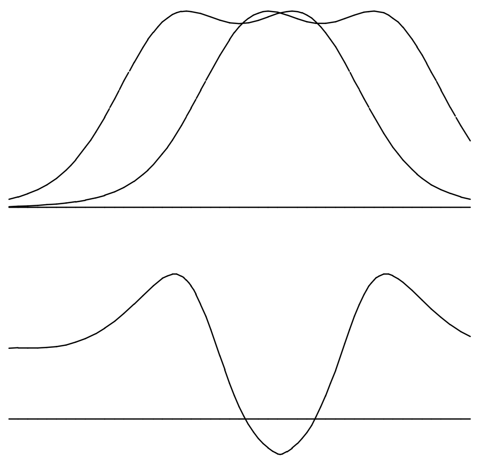

**Figure 2.5**: If the interframe motion is large like the top plot, a single scale algorithm that is based on intensity matching will confuse one hump with the other. This is apparent in the bottom plot which depicts the flow being negative, which means it points in the opposite direction.

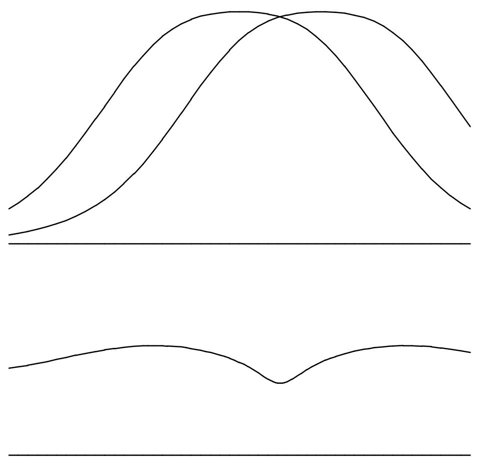

**Figure 2.6**: Before we go to a lower resolution we need to smooth the image to get rid of the fine detail that will only create aliasing problems. As a result, at some lower resolution the two humps will look like one and the original flow algorithm will not be confused. The bottom plot is quite close to the ground truth and is never in the wrong direction.

## 2.5. Other Techniques

### 2.5.1. Affine Motion

### 2.5.2. Smoothness Constraints

The flow in real image sequences is piecewise continuous, which in plain terms means that the image is a patchwork of areas where the flow is continuous which are separated by lines of discontinuity. Assuming that we are working wholly within a patch of the image where the flow is continuous we can recover the flow using the techniques mentioned above. But despite our best effort the flow will look rough and jagged.

There are two mechanisms that conspire to create this problem. The one is the small amount of noise in the images, either random noise, discretization noise, motion blur even violation of the assumptions, that always tends to get amplified. The other is that we do not have any mechanism to tell our programs to prefer the smoother solution among more or less equally probable ones. An obvious, simple and effective way to solve both problems is to use larger regions when we do the matching. This has some unwanted side effects like tends to overblur a bit but overall works fine.

We wouldn't introduce a subsection just to say this, would we? This is a solution that is obvious after reading the previous sections. But it is an excellent opportunity to introduce a set of mathematical techniques while presenting an alternative solution to the same problem.

We will try to modify Eq. (2.5) in such a way that the result of the minimization will almost minimize the sum of the squares of Eq. (2.3) but it will show a preference for smoother solutions. In other words we will trade loyalty towards Eq. (2.3) for some smoothness. After all Eq. (2.5) is approximate itself.

We use the same powerful least squares tool again with some modification. We do not minimize the sum of squares of Eq. (2.3) alone but in combination with some other term that imposes some smoothness on the solution. This solution will minimize the sum of the old terms and this new term. But we have to go one step further before we can use this technique.

In Eq. (2.4) we summed over a single region only because we matched each region independently. And our unknowns where only two for this reason. But now we attempt something much more daring. We try to impose a property that is not strictly local since by looking at an isolated pixel we cannot tell if its flow is smooth. We have to see a few of its neighbors too. So instead of solving $N \times N$ small systems with two unknowns, we solve a single system with $2N \times 2N$ unknowns (where $N \times N$ is the resolution of the image).

So Eq. (2.4) becomes

$$\mathbf{S}_{phys} = \sum_{i_0, j_0} S[i_0, j_0] =$$
$$\sum_{i_0, j_0} \sum_{i, j} w[i, j] \left\{ I_x[i_0 - i, j_0 - j] u[i_0, j_0] + I_y[i_0 - i, j_0 - j] v[i_0, j_0] + I_t[i_0 - i, j_0 - j] \right\}^2 \quad (2.6)$$

which represents the *physical* constraint, e.g. the set of conditions imposed to the solution from the problem. Now, we have to introduce the *a priori* constraints, e.g. constraints that we know in advance, like that this particular part of the image is smooth.

In order to use this a priori constraint with the least squares it has to be expressed by a term that is a real function of the image flow, it is always positive, it is small for smooth flows and large for rough ones. Clearly there are many choices depending on what exactly we mean by “smooth” and how much cpu time we are prepared to spend. The simplest and most intuitive is the sum of squares of the derivatives of the flow which can be expressed as

$$\mathbf{S}_{smooth} = \sum_{i_0, j_0} \left\{ u_x[i_0, j_0]^2 + u_y[i_0, j_0]^2 + v_x[i_0, j_0]^2 + v_y[i_0, j_0]^2 \right\}. \quad (2.7)$$

where the subscript denotes the corresponding derivative. It is obvious that such a smoothness term satisfies the above mentioned criteria.

The beauty of least squares is that various constraints can be combined by simply adding them so we have to minimize

$$\mathbf{S} = \mathbf{S}_{phys} + \lambda \mathbf{S}_{smooth}$$

where $\lambda$ is a constant we choose so that the smoothness term has a small but not negligible effect.

The next step is to minimize **S**. The unknowns here are all the $u[i_0, j_0]$s and $v[i_0, j_0]$s in other words two unknowns per pixel. This may sound like many but the procedure to obtain the equations for the minimazation is exactly the same. We take the derivative of **S** with respect to each one of the unknowns. To keep things simple we do each term separately. The physical term will give exactly the same results as before, so we do not worry about this. The smoothness term is really four distinct terms, but very similar to each other. We will show the derivation of only one of them and the rest will be similar. The next step is to add all five of them (the physical and the four smoothness) and derive an equation that we can solve.

Let's derive the normal equation for one of the four terms

$$\sum_{i_0, j_0} u_x[i_0, j_0]^2$$

which we will differentiate with respect to one of the unknowns which are all the *u*s and *v*s, say $u[k, l]$

$$\frac{\partial}{\partial u[k, l]} \sum_{i_0, j_0} u_x[i_0, j_0]^2$$

which can be transformed to

$$\sum_{i_0, j_0} 2u_x[i_0, j_0] \frac{\partial}{\partial u[k, l]} u_x[i_0, j_0] \tag{2.8}$$

and now we have to take the derivative of $u_x[i_0, j_0]$ with respect to $u[k, l]$. This looks complicated but it is not.

Admitedly, taking the derivative of the derivative of a function with respect to the function sounds confusing. So we go back to the basic definitions. We know that $u$ is a discretized version of the flow and that when we take derivative with respect to $x$ we actualy convolve $u$ with a template $b$. So we use this to get

$$\frac{\partial u_x[i_0, j_0]}{u[k, l]} = \frac{\partial}{u[k, l]} \sum_m b[m] u[i_0, j_0 - m] =$$
$$\sum_m b[m] \frac{\partial}{u[k, l]} u[i_0, j_0 - m].$$

The derivative of $u$ at one point with respect to $u$ at a different point is almost always zero, because each $u$ is an independent variable. The only case of course that this is equal to one is when the points are identical. This can be written as

$$\frac{\partial}{u[k, l]} u[i_0, j_0 - m] = \delta[k - i_0]\delta[j_0 - m - l]$$

So

$$\frac{\partial u_x[i_0, j_0]}{u[k, l]} = \sum_m b[m]\delta[k - i_0]\delta[j_0 - m - l] = \delta[k - i_0]b[j_0 - l]$$

Now that we know this derivative we can substitute it and continue on Eq. (2.8) from where we stopped

$$\frac{\partial}{\partial u[k, l]} \sum_{i_0, j_0} u_x[i_0, j_0]^2 = \sum_{i_0, j_0} 2u_x[i_0, j_0]\delta[k - i_0]b[j_0 - l]$$

and we can simplify the double summation to simple by eliminating $i_0$ by making use of the properties of the $\delta$ function

$$\sum_{j_0} 2u_x[i_0, j_0]b[j_0 - l]$$

which is nothing more than a convolution. There is a catch though. If it was a regular convolution then the index of $b$ would be $b[l - j_0]$ and not $b[j_0 - l]$. This means that the derivative template $b$ is flipped around zero and since it is an odd template (changes sign if flipped) we can just change sign. So finaly

$$\frac{\partial}{\partial u[k, l]} \sum_{i_0, j_0} u_x[i_0, j_0]^2 = -u_{xx}[k, l]$$

This is a beautifull equation that we derived using descrete techniques. We could also use continuous techniques and derive exactly the same thing. These continuous techniques are called *Calculus of Variations*.

If we put the whole thing together we cannot write it as a simple matrix expression as in Eq. (2.5), because the equation about the flow in one pixel depends on the flow of the surrounding pixels. So we write it as

$$-\lambda(u_{xx} + u_{yy}) + E_{xx}u + E_{xy}v + E_{xt} = 0$$
$$-\lambda(v_{xx} + v_{yy}) + E_{xy}u + E_{yy}v + E_{yt} = 0$$

And how do we solve this equation that involves both the $u$s and $v$s and their derivatives? Mathematicians call these things differential equations, a dreadful name indeed, but justifiably so. How do we solve them? There are two cases: The continuous, which we cannot solve, and all we do is guess a solution and try to verify it (mathematicians have conspired to invent all kind of nice theories to hide this ugly detail of their trade) and the descrete which can be reduced to the solution of a huge linear system. This we know how to solve. Use a high quality numerical computation library.