CHAPTER 34

Optic Flow

Here is a useful trick, widely familiar to flying animals. Compute some measure S of the size of an object in an image. You could use diameter, area, or a similar measure. Move the camera, or eye and compute the time derivative of that measure, S_t. The expression S/S_t is an estimate of time to contact *which is unaffected by camera calibration* (Figure 34.1; **exercises**). If this number is less than some threshold which has to do with your ability to accelerate, etc. you're in trouble. This trick is a simple manifestation of the information that *optic flow* – the movement of pixels in an image caused by camera or scene movement – has to offer about the movement of an agent in the world. There are two big families of method for estimating optic flow: elementary methods, which use various forms of geometric, physical and statistical reasoning to recover flow fields; and regression methods, which follow the recipe of Chapter 21.3.

34.1 OPTIC FLOW AS A CUE TO GEOMETRY AND MOTION

A point in 3D that projects to $\mathbf{x}_1$ in frame 1 moves to $\mathbf{x}_2$ in frame 2. This movement can be visualized by placing the *flow vector* $\mathbf{v}(\mathbf{x}) = \mathbf{x}_2 - \mathbf{x}_1$ at each location in frame 1. Figure 34.2 shows some fields of flow vectors that might be observed by a moving camera. These flow fields convey a great deal of information about how the camera moved and what the shape of the world is.

You can recover 3D shape information from flow, as this simple example shows. View a shape in the standard camera coordinate system. The point at $X_1, X_2, X_3(X_1, X_2))$ projects to $(X_1/X_3, X_2/X_3)$ in the first image. The flow at this point will be $(0, -\frac{t_y}{X_3})$, so if you know the flow, you can recover the depth. There's no real difference between flow and disparity (Chapter 21.3) here.

All the two-camera geometry of Chapter 21.3 applies. Epipoles take an extremely convenient from. Assume that the camera is calibrated, starts in the standard configuration, and translates, so camera 2's focal point is at $\mathbf{p}$ in camera 1's frame. Equivalently, the point at $\mathbf{X}$ in camera 1's frame is at $\mathbf{X} - \mathbf{p}$ in camera 2's frame (check you are sure about the minus sign). Check that the flow will be zero at the point $(p_1/p_3, p_2/p_3)$ in the image plane, whatever objects appear in the 3D world. Flow close to the focus could be contracting or expanding, but this point is often referred to as the *focus of expansion*. It is also the epipole (**exercises**). If the flow points away from this point (expanding), the camera will hit the focus of expansion if it keeps moving as it has been moving. Similarly, if the flow points toward this point (contracting), the camera is escaping from that point in the scene. The three flow fields of Figure 34.2 signal quite different things to a flying agent. Any movement of the camera where the focal point translates will result in a focus of expansion (**exercises**).

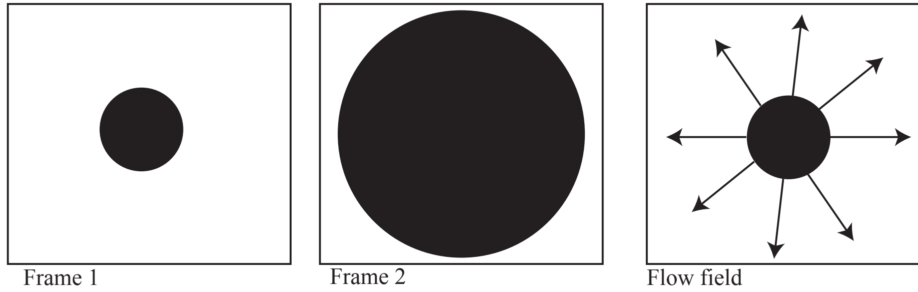

FIGURE 34.1: *Images 1 and 2 here offer an alarming warning of an impending collision, which is captured in the flow field. In my experience, you can startle an audience with carefully constructed slides showing scenes like these two. A darkened room and a bright projector seem to help.*

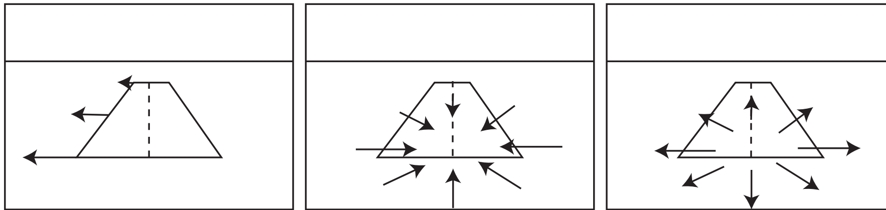

FIGURE 34.2: *Flow offers simple, clear signals about how one is moving with respect to the world. Look out of a bubble canopy on the nose of a small aircraft. If you see the flow on the* **left**, *you are heading from left to right. If you see the flow in the* **center**, *you are leaving the runway behind. If you see the flow on the* **right**, *you are heading towards the near end of the runway. Various reflexes related to flow patterns are found in a number of flying animals. For example, the flow on the right – if large enough compared to the size of the object in the image – will often trigger some collision management reflex.*

34.1.1 Parametric Flow Examples

Looking out a train window: A calibrated camera looks at a ground plane from a fixed height above it (Figure 34.3). The camera translates parallel to the ground plane and parallel to the image plane. The ground plane does not pass through the focal point, so write $bX_2 + cX_3 - 1 = 0$ in the camera coordinate system for the ground plane. The camera translates in the X_1 direction by t_x. The ray from the focal point through $\mathbf{x} = [x_1, x_2]$ in the image plane pierces the ground plane at $X_3 = 1/(bx_2 + c)$. This means the flow is

$$\begin{bmatrix} -t_x(bx_2 + c) \\ 0 \end{bmatrix}.$$

This flow is linear in image position. Notice the flow is zero when $(bx_2 + c) =$

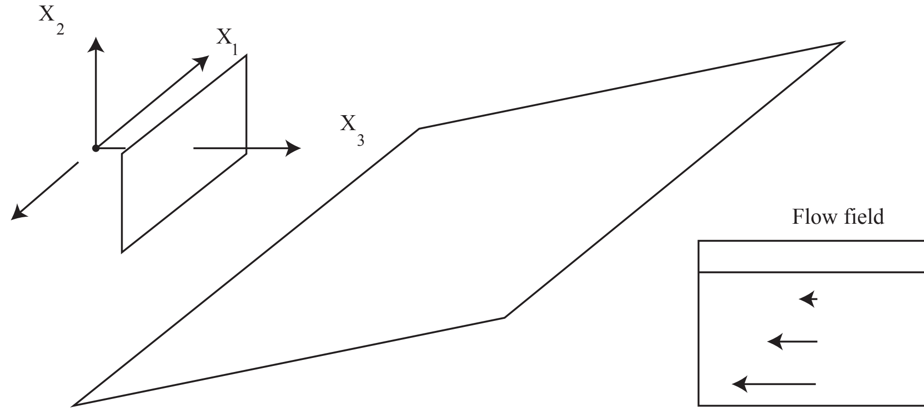

FIGURE 34.3: *A camera looks out of a train window at right angles to the direction of travel (the small arrow) and views a ground plane. As the text shows, the flow field looks like the inset – the flow at the horizon is zero, and flow grows linearly as you move down from the horizon.*

0, which is also the horizon of the plane (check this **exercises**). Here I have grouped the variables quite deliberately to show an ambiguity. You can estimate two parameters, $t_x b$ and $t_x c$. This is an ambiguity in the geometry, but not in the flow. For example, if $c = 0$, this ambiguity says a large movement by a camera high above the ground plane results in the same flow as a small movement by a camera closer to the plane.

Looking down on a bumpy ground: Now the calibrated camera looks directly down at a ground plane from a fixed height above it. The camera translates parallel to the ground plane, and the image plane is parallel to the ground plane. The ground plane will be $Z = c$ for some constant. But the ground isn't really a plane. Houses, trees, people and such stick up out of the ground, and generate variations in the flow field, so write the ground as $Z = c + \delta Z$ where δZ is small compared to c and represents this relief. Choose the ground plane so that the relief is always non-negative. Now set up a coordinate system so that, at time 1, the camera is in the standard coordinate system for a perspective camera. The camera translates parallel to the ground plane, so by $(t_x, t_y, 0)$. The time interval is small. To first order, the flow is

$$\left[\begin{array}{c} \frac{-t_x}{c}(1 - \frac{\delta Z}{c}) \\ \frac{-t_y}{c}(1 - \frac{\delta Z}{c}) \end{array}\right]$$

. I have grouped the variables quite deliberately to show an ambiguity. You can estimate two parameters, $\frac{t_x}{c}$ and $\frac{t_y}{c}$ and a relative height field $\frac{\delta Z}{c}$. This is an ambiguity in the geometry, but not in the flow.

Tilting towards the ground: A calibrated camera looks at a ground plane from a fixed height above it. The camera then tilts slightly about the X-axis. In frame 1, the ground plane is at $Y = c$, so that an image point at (x, y) is the projection of $(\frac{xc}{y}, c, \frac{c}{y})$. It is easier to tilt the ground with a fixed camera, so in

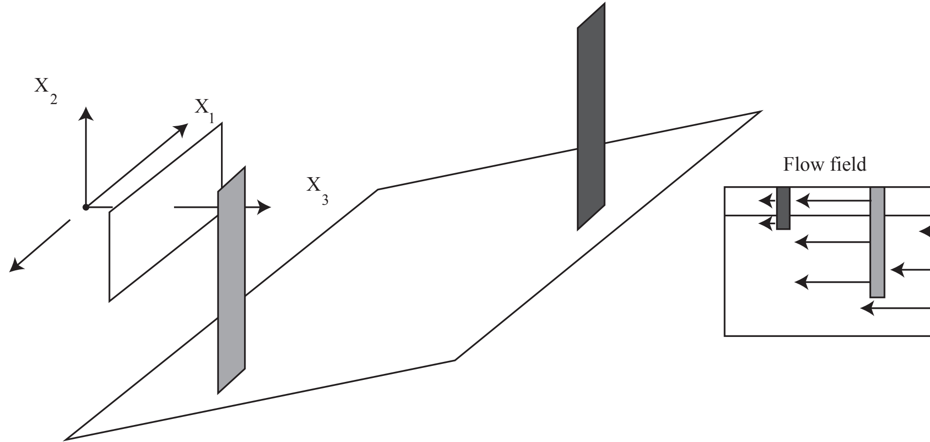

FIGURE 34.4: *A camera looks out of a train window at right angles to the direction of travel (the small arrow) and views a ground plane with two trees on it. The flow field on the ground plane looks like the inset of Figure 34.3, but the flow at the trees is different. The nearby tree has a large flow, and the distant tree has a small flow.*

frame 2 this point is now at $(\frac{xc}{y}, c - \epsilon\frac{c}{y}, \frac{c}{y} + \epsilon c)$ (**exercises**). To first order, the flow is

$$\begin{bmatrix} -\epsilon xy \\ -\epsilon(1-y^2) \end{bmatrix}$$

Multiple objects on a plane: A calibrated camera translates in the x direction at a fixed height above a ground plane. The ground plane is at $Y = c$ in camera coordinates, so the image plane is perpendicular to the ground plane. A collection of N flat objects stick up out of the ground plane at fixed depths (Figure 34.4; the i'th object is at $Z = d_i$). For this setup, the flow in the y direction in the image plane is always zero. Check that, if the pixel at $(x, y)^T$ sees the ground plane, the z coordinate is c/y. The x component of flow at a pixel takes one of $N + 1$ values

$$\begin{cases} -\frac{t_x c}{y} & \text{if the pixel sees the ground plane} \\ -\frac{t_x}{c_1} & \text{if the pixel sees object 1} \\ \dots & \dots \\ -\frac{t_x}{c_N} & \text{if the pixel sees object N} \end{cases}$$

This class of model is often referred to as a *layered motion* model (after []), because there is one flow per layer.

Affine flow models: An *affine flow model* is a flow model where flow at the

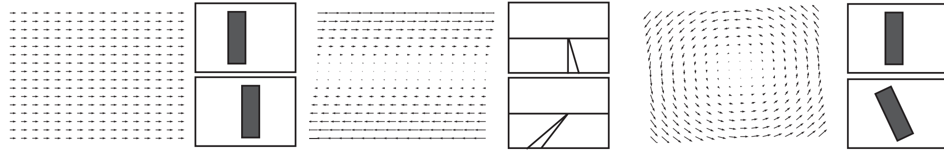

FIGURE 34.5: *A straightforward affine flow model in Section* **??** *encodes many useful cases. On the* **left***, constant flow (the first column of $\mathcal{M}$ gives the flow direction, all others are zeros). In the* **center***, the flow viewed from a train window in Figure 34.3 (here the first three columns of $\mathcal{M}$ are non-zero, others are zeros). On the* **right***, an object rotates (again, the first three columns of $\mathcal{M}$ are non-zero, others are zeros).* **Top** *image frame is frame 1,* **bottom** *is frame 2. Details in* **exercises**.

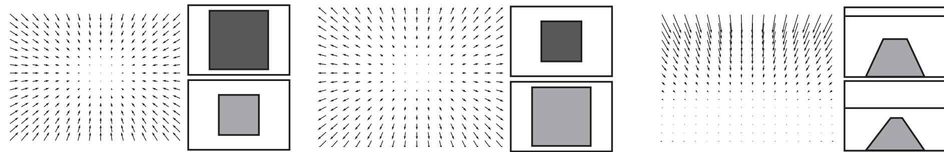

FIGURE 34.6: *A straightforward affine flow model in Section* **??** *encodes many useful cases. On the* **left***, the viewer leaves an object; in the* **center***, a viewer is heading towards an object; and on the* **right***, the plane on which the rectangle sits is tilted backwards (or, equivalently, the viewer tilts backwards).* **Top** *image frame is frame 1,* **bottom** *is frame 2. Details of the relevant $\mathcal{M}$ for each case in* **exercises**.

pixel at $(x, y)^T$ is given by

$$\begin{bmatrix} u(x,y) \\ v(x,y) \end{bmatrix} = \mathcal{M} \begin{bmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{bmatrix}$$

. This covers a number of useful cases (Figures 34.5 and 34.6; **exercises**).

34.2 ESTIMATING DENSE FLOW FIELDS WITH SMOOTHING

Flow estimation must be hard for at least some image points. If all you can see is a field of constant pixel value, you cannot tell whether there is any flow. The scene could move without you knowing it had moved. This has the consequence that the only way you can estimate flow for a blob of pixels of constant value is by some kind of interpolation from the boundary.

In the same way, the flow at a straight edge is locally ambiguous, because you can't tell whether there is any flow *along* the edge wherever the edge is straight. The flow is constrained because you know the component across the edge, but you don't know it uniquely. Figure 34.7 illustrates an exaggerated version of this problem.

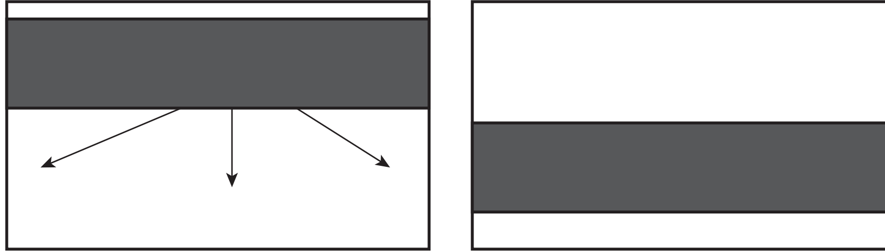

FIGURE 34.7: *Each of the three flow vectors is a plausible explanation of the flow on this moving bar, because you cannot tell how it has translated horizontally. This effect is known as the aperture problem.*

This isn't just a property of edges. If the line segment in the figure was an *isophote* (a curve of equal brightness points in the image) the same problem would occur. You would not be able to tell *locally* if there was any flow *along* the isophote. This problem is known as the *aperture problem*.

You can tell where corners are, so you can tell where they have gone to, which means flow at corners is unambiguous. This is the keystone of flow estimation. Any flow estimation procedure has to "fill in" flow between the scattered locations where it is unambiguous. The "filled in" flow needs to meet constraints at edges.

One procedure for estimating flow applies a version of the master recipe for denoising: find a flow field that is (a) close to consistent with image data and (b) like a real flow field. Write $\mathbf{u}$ for a vector giving the value of the flow at each pixel to construct a cost function

$$\begin{aligned} C(\mathbf{u}) &= \begin{array}{c}[\text{the extent to which } \mathbf{u} \text{ is consistent with image data}] + \\ [\text{the extent to which } \mathbf{u} \text{ is like a real flow field}]\end{array} \\ &= [\textit{data term}] + [\textit{penalty term}]\,. \end{aligned}$$

and choose $\mathbf{u}$ that minimizes this cost function. You can think about each component of flow as an image (one value per pixel). In this view, methods differ by (a) how they compute consistency with the image data and (b) how they measure similarity to a real flow field.

34.2.1 What Makes Flow Estimation Hard

The "filling in" process must involve some kind of smoothing. The train example can be used to show that flow fields tend not to be smooth. Figure 34.4 shows the flow field when there are two trees sticking out of the ground plane. The flow inside the boundary of either tree is very different from that outside the boundary. From Section 34.1 and this example it follows that you should expect discontinuities in flow at discontinuities in depth. These discontinuities in flow should – and do – generate problems estimating the flow.

You can't estimate flow at a pixel that is there in one image but not in the other image. This occurs at *occlusions* – something visible in the first image is

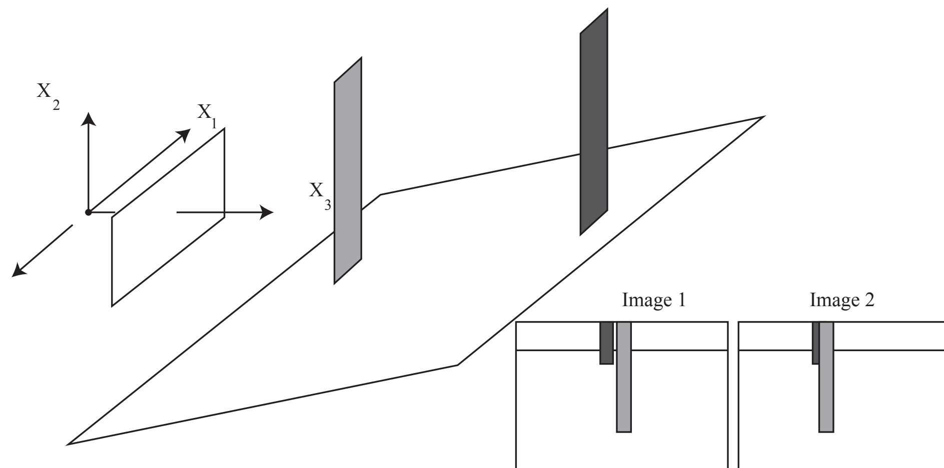

FIGURE 34.8: *A camera looks out of a train window at right angles to the direction of travel (the small arrow) and views a ground plane with two trees on it. Because the nearby tree moves by more than the distant tree (as in Figure 34.4), for this geometry the distant tree will disappear behind the nearby tree. Reversing the direction of camera movement will get an example where the distant tree reappears from behind the nearby tree. In either case, there are pixels in one image that aren't seen in the other, which must create estimation problems.*

hidden behind something else in the second image. Parts of the distant tree disappear behind the nearby tree in Figure **??**, which must create estimation problems. Similarly, parts of the distant tree might reappear from behind the nearby tree. Occlusions tend to appear at *depth discontinuities*, but depth discontinuities have other nasty effects. You are very likely to observe a discontinuity in the flow field at a depth discontinuity, for example.

Fast moving objects present particular difficulties. A fast moving object may cover a pixel only for part of the time that the camera shutter is open, meaning that the value reported by the pixel is some weighted average of the background color and the object color. Some locations may see mostly background, and others will see

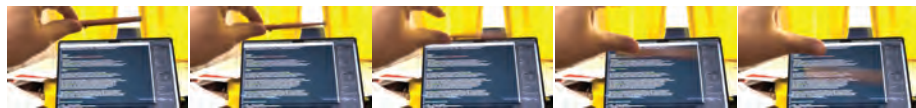

FIGURE 34.9: *Fast moving objects produce heavily blurred images. These are five consecutive frames of a sequence where I dropped a pen on my laptop. Notice in the* **far left** *frame, my hand is in focus as is the pen. Dropping the pen involves a fairly fast movement of my thumb, blurred in the* **left** *frame. As the pen accelerates under gravity, it is increasingly blurred.*

mostly object (Figure 34.9). The result is blur around the boundary of the object, usually referred to as *motion blur*. This will affect the image gradient estimates and the time derivative of intensity, and so tend to make the flow constraint unreliable. Small, fast moving objects are particularly bad, because there may be few pixels on the object and all could be affected by motion blur.

34.2.2 Photometric Consistency

Flow estimation rests on the assumption that the intensity of a point on an object does not change when the object or the camera move. This assumption means you can rule out certain flows, because they imply that the object has changed intensity. It is mostly sound, but there are cases where the assumption doesn't apply. There are a variety of reasons the amount of light reflected from the surface of the object to the camera might change when the object moves by a small amount. If the material of the object has a strong specular or glossy component, the intensity might change because the angles to sources will change. Alternatively, the object might move into a shadow volume. Finally, the illumination intensity varies across space (for example, as a result of interreflections).

This assumption produces a very important constraint on flow. At time t, a point $\mathbf{V}$ on an object projects to the point $\mathbf{x}$ in the image, producing intensity $I(\mathbf{x}, t)$. The camera then moves by a small amount and at time $t + \delta t$ a new image is obtained. The point now projects to $\mathbf{x} + (\delta t)\mathbf{u}$, where $\mathbf{u}$ is the flow. Then $I(\mathbf{x} + (\delta t)\mathbf{u}, t + \delta t)$ is the same as $I(\mathbf{x}, t)$, so

$$I(\mathbf{x} + (\delta t)\mathbf{u}, t + \delta t) - I(\mathbf{x}, t) = 0.$$

I will refer to this property as *photometric consistency*. The form is inconvenient, because the constraint is not linear.

If the flow and δt are small enough, a Taylor series is a good approximation, and you find

$$I(\mathbf{x} + (\delta t)\mathbf{u}, t + \delta t) \approx (I(\mathbf{x}, t) + (\delta t)\left[(\nabla I)^T \mathbf{u} + \frac{\partial I}{\partial t}\right]$$

so that for sufficiently small δt

$$(\nabla I)^T \mathbf{u} + \frac{\partial I}{\partial t} = 0.$$

At any pixel, it is easy to compute estimates ∇I and $\frac{\partial I}{\partial t}$, so this equation could yield one linear constraint on the flow vector $\mathbf{u}$. This isn't enough to recover the flow vector exactly, but can strongly constrain the flow. The linearized constraint has some issues. The movement might be so large that linearizing the expression is reckless, and knowing when this occurs can be difficult. If the image has many sharp edges, the linearized constraint can break down quite quickly. This effect is quite easily seen in an example (Figure 21.3).

34.2.3 Estimating Flow using Smoothing

Now assume the linearized version of the photometric consistency constraint applies. You do not know the image gradient or the time derivative exactly, but you can

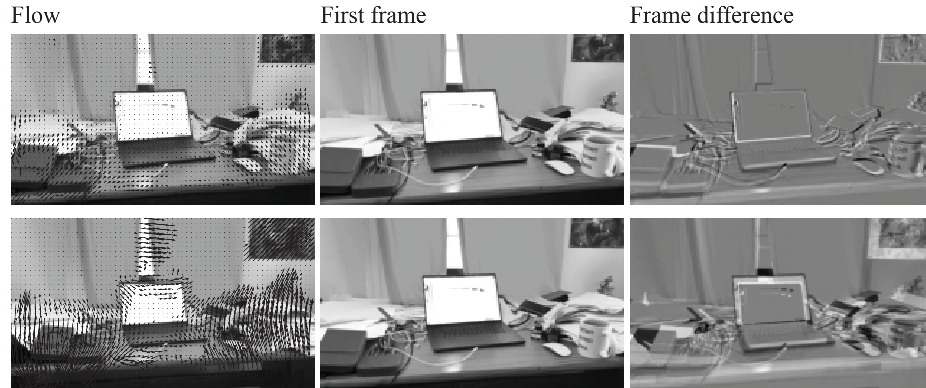

FIGURE 34.10: *Estimates of flow for a pair of images of my laptop, obtained with the method of Section 34.2.4.* **Top** *row shows estimates obtained with a small elapsed time;* **bottom** *shows estimates obtained with a large elapsed time.* **Left** *is the first frame with flow superimposed as small arrows;* **center** *is the second frame; and* **right** *is second frame - first frame, scaled so that zero is gray, negative numbers are dark, and positive numbers are light. This makes it possible for you to check the flow estimates – if, reading left to right, you see a vertical light bar closely followed by a vertical dark bar at the left edge of a bright region, the second frame is moving left.*

estimate them. Use

$$D(u,v) = \sum_{i,j \in \text{pixels}} \left((\nabla I_{ij})^T \mathbf{u}_{ij} + \frac{\partial I_{ij}}{\partial t} \right)^2$$

as a data term. Here the subscripts refer to the value at the locations (so ∇I_{ij} is the gradient of I at i, j and so on). A simple penalty term follows from assuming that the gradient of each flow component is small. Flow components are u, v, so

$$P(u,v) = \sum_{i,j \in \text{pixels}} \left([\nabla u_{ij}]^T [\nabla u_{ij}] + [\nabla v_{ij}]^T [\nabla v_{ij}] \right)$$

should be small. Then $C(u,v) = D(u,v) + \alpha P(u,v)$ where α weights the penalty term against the data term. This cost function is quadratic in u and v, so a solution is obtained with linear algebra. Solving the resulting flow equation is straightforward, but you should expect reasonable results only for very small flows (**exercises**). The method I have described is known as the *Horn-Schunk* method, and was originally described in [].

34.2.4 Fitting for a Flow Estimate

Approximate the intensity in some image window by $\mathcal{I}_1(\mathbf{u}) = \mathbf{u}^T \mathcal{A}_1 \mathbf{u} + \mathbf{b}_1^T \mathbf{u} + c_1$. Now the image translates by $\mathbf{t}$, so that $\mathcal{I}_2(\mathbf{u} - \mathbf{t}) = \mathcal{I}_1(\mathbf{u})$. The function describing

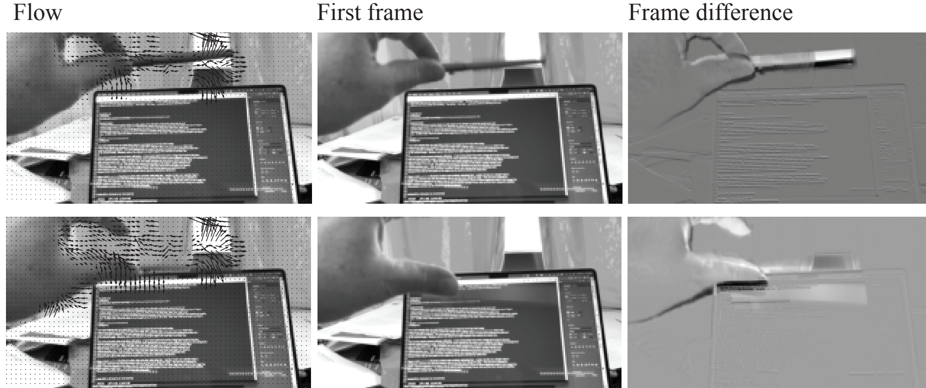

FIGURE 34.11: *Estimates of flow for a pairs of images from the sequence of Figure 34.9, obtained with the method of Section 34.2.4.* **Top** *row shows flow estimate from the first frame to the second frame, where blur is a relatively minor factor;* **bottom** *shows estimates obtained from the third to the fourth – the pen appears to have no effect on the flow field.* **Left** *is the first frame with flow superimposed as small arrows;* **center** *is the second frame; and* **right** *is second frame - first frame, scaled so that zero is gray, negative numbers are dark, and positive numbers are light. This makes it possible for you to check the flow estimates – if, reading left to right, you see a vertical light bar closely followed by a vertical dark bar at the left edge of a bright region, the second frame is moving left.*

the new translated window is

$$\mathbf{u}^T\mathcal{A}_2\mathbf{u} + \mathbf{b}_2^T\mathbf{u} + c_2 = (\mathbf{u}-\mathbf{t})^T\mathcal{A}_1(\mathbf{u}-\mathbf{t}) + \mathbf{b}_1^T(\mathbf{u}-\mathbf{t}) + c_1$$

so that $\mathcal{A}_2 = \mathcal{A}_1$, $\mathbf{b}_2 = (\mathbf{b}_1 - 2\mathcal{A}_1\mathbf{t})$, $c_2 = c_1 - \mathbf{b}_1^T\mathbf{t} + \mathbf{t}^T\mathcal{A}_1\mathbf{t}$, which yields a linear system in $\mathbf{t}$.

In turn, a flow estimation procedure follows. Compute the best quadratic approximation at each location in $\mathcal{I}_1$ and $\mathcal{I}_2$, yielding $\mathcal{A}_1(\mathbf{x})$, and so on. Here $\mathbf{x}$ identifies the center pixel of the window for which you computed the quadratic approximation. Then an estimate of flow at each location is given by solving

$$\frac{1}{2}\left(\mathcal{A}_1(\mathbf{x}) + \mathcal{A}_2(\mathbf{x})\right)\mathbf{t}(\mathbf{x}) = -\frac{1}{2}(\mathbf{b}_2(\mathbf{x}) - \mathbf{b}_1(\mathbf{x}))$$

to obtain $\mathbf{t}(\mathbf{x})$, the flow field. Although $\mathcal{A}_1(\mathbf{x})$ should be the same as $\mathcal{A}_2(\mathbf{x})$, you should use

$$\frac{1}{2}\left(\mathcal{A}_1(\mathbf{x}) + \mathcal{A}_2(\mathbf{x})\right)$$

because it will, very slightly, suppress noise. Estimates obtained like this tend to be noisy. An improvement is available by assuming that the flow field is smooth, so that nearby windows can contribute to the estimate. At $\mathbf{x}$, choose a flow that minimizes a weighted average of errors over a local window given by some set of

offsets δx. This gives the optimization problem

$$\mathbf{t}(\mathbf{x}) = \underset{\mathbf{u}}{\text{argmin}} \sum_{\delta x \in \text{window}} w(\delta_x) \| \mathcal{A}(\mathbf{x}+\delta x)\mathbf{u} + \frac{1}{2}(\mathbf{b}_2(\mathbf{x}+\delta x) - \mathbf{b}_1(\mathbf{x}+\delta x)) \|_2^2 .$$

Typically, one uses gaussian weights. This method is often referred to as the *Farneback method*, and originates in []. Example flow estimates appear in Figure 34.11. As Figure **??** shows, this method is susceptible to motion blur (as are other methods).

34.2.5 Improving Flow Estimates

Each of the flow estimators above can be seen as a solution to a fitting problem, so you should suspect there might be robustness issues. This suspicion is correct, and replacing the quadratic cost in the data term with some more robust estimator will produce improvements. Applying IRLS (Section 21.3) is an instructive, if elaborate, exercise. I do not do this in detail because these flow estimators are now largely obsolete.

Approximating the image can cause serious difficulties and deserves more detailed attention. Both the linearized photometric constraint and the approximation used in Section 34.2.4 approximate $\mathcal{I}(x+u, y+v)$. Neither approximation applies over a sufficiently long spatial scale, and it is quite hard to determine what is too long a scale. There are two strategies for managing these effects, which are quite often interwoven. In *iterative estimation*, you assume that you have a fair estimate of the flow – write u^n, v^n – and want to get a better estimate $u^{n+1} = u^n + \delta u$, $v^{n+1} = v^n + \delta v$. You can assume that the update is small, and the linearized photometric constraint is

$$I(x+u^{n+1}, y+v^{n+1}, t+\delta t) \approx I(x+u^n, y+v^n, t) + \frac{\partial I}{\partial x}\delta u + \frac{\partial I}{\partial y}\delta v + \frac{\partial I}{\partial t}\delta t.$$

You then start with $u^1 = 0$, $u^2 = 0$, and repeatedly re-estimate. For this strategy to succeed, early estimates need to be good enough that small refinements are sufficient. This doesn't always happen.

In *coarse-to-fine estimation*, you notice that in quite smooth images, the gradient is a good guide to the appearance of the image quite a long way away from a pixel. In this case, you can expect the linearized constraint to be usable for quite large flows. But if the image is very smooth, there is no point in solving for flow at every pixel because flow at neighbors will be very like flow at the pixel. In turn, subsample the image, and solve for flow in the subsampled image. This should make you think about a gaussian pyramid.

Here is a starter recipe for a coarse-to-fine procedure. Form a gaussian pyramid for the two images. At the coarsest scale, estimate flow. Upsample this estimate to the next coarsest scale. Notice you have to (a) increase the number of locations to get a flow vector at every pixel, which you can do by interpolation and (b) increase the length of the flow vector, because moving by (say) a pixel at a coarse scale is equivalent to moving rather further at a less coarse scale. Now use the upsampled estimate as an initial estimate of flow, and linearize about that to estimate a correction. Continue to upsample and correct until you arrive at the finest scale.

This is a starter recipe because there may be some advantage to passing the fine scale flow estimate down the pyramid to coarser scales. This option – which turns the flow estimator into a *multigrid method* is out of scope for us. You might start reading at [].

34.3 FLOW ESTIMATION WITH PARAMETRIC MODELS

Section 34.1.1 gave a number of parametric flow models. A natural estimation procedure is to choose a parametric flow model, then solve for the parameters of the flow. At its best, this procedure manages the ambiguity in the optical flow equation and gets very good estimates of parameters – and so, geometry.

The methods of Section 21.3 and Section 21.3 are actually parametric – you can see the u and v at each pixel as two parameters at each pixel – but they have a very large number of parameters and so are not typically thought of as parametric. The large number of parameters means the methods might be able to fit a wide range of flow fields quite accurately, but are more likely to make more errors. They control errors by smoothing the estimated flow field. This section focuses on cases where there are few parameters, typically far fewer than the number of pixels.

34.3.1 Parametric Flow: A General Recipe

An important part of building good parametric flow estimators is coming up with parametric flow models that represent the flows you will encounter well with relatively few parameters. For the moment assume you have a good model.

Write $\mathbf{t}(\theta) = (u(x, y; \theta), v(x, y; \theta))^T$ for a parametric flow field, where θ are the parameters of the field. To obtain a flow field, you must choose some function $D(\delta I)$ for the data term to penalize the photometric inconsistency, another function – which you might even omit – $P(\theta)$ which penalizes the flow for being unrealistic. You then minimize

$$C(\theta) = D(I_2(x + u(x, y; \theta), y + v(x, y; \theta)) - I_1(x, y))) + \alpha P(\theta)$$

as a function of θ. In the simplest case, $C(\delta I) = (\delta I)^2$, you set $\alpha = 0$, and you use the linearized photometric inconsistency. If the parametric flow model is linear in the parameters, you will need to solve a linear system (**exercises**). The simplest case is not necessarily the best. A more effective procedure applies the approximation of Section 34.2.4 to set up an optimization problem. This works best when the flow is linear in the parameters, so $\mathbf{t}(\theta) = \mathcal{M}\theta$. Then solve

$$\theta = \begin{matrix}\text{argmin} \\ \phi\end{matrix} \sum_{\delta x \in \text{window}} w(\delta_x) \| \mathcal{A}(\mathbf{x} + \delta x)\mathcal{M}\phi + \frac{1}{2}(\mathbf{b}_2(\mathbf{x} + \delta x) - \mathbf{b}_1(\mathbf{x} + \delta x)) \|_2^2 .$$

You could do coarse-to-fine estimation with a robust cost function and IRLS, as sketched in Section 21.3. Because – by construction – you have relatively few parameters, you can use quite sophisticated optimization strategies like Newton's method or BFGS (see []). Some cases are explored in the **exercises**.

34.3.2 Estimating a Simple Mixture of Flows

Recall the simple layered motion model of Section 21.3. Assuming you don't know the ground plane constant, the translation, or the depth of any object, you still have

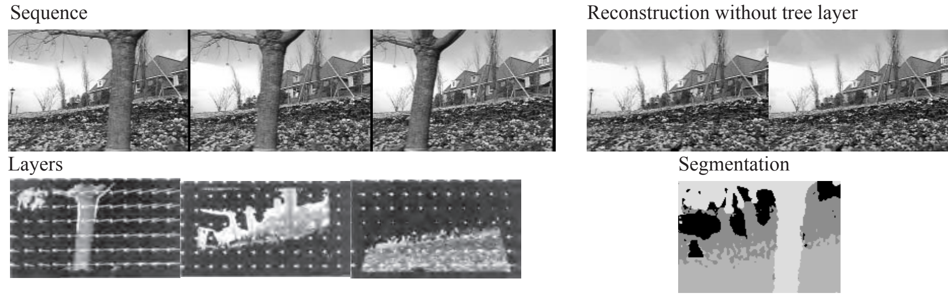

FIGURE 34.12:

only $N+1$ constants to estimate. Estimating these constants is tightly intertwined with segmenting the image into objects and ground plane. If the flow at pixel $(x, y)^T$ is (u, v), then $I_2(x+u, y+v)$ should be very similar to $I_1(x, y)$. The photometric error can be used to estimate this class of flow using an algorithm which is quite strongly analogous to k-means. You iterate: estimate the flow assuming you know which object each pixel belongs to; then estimate which object a pixel belongs to using the flows.

Estimating the flow: Assume that, for every pixel, you know which of the $N+1$ cases applies; that is, whether it is background, or one of the objects. Now take all pixels that lie on object i, and search for the u such that

$$\sum_{(x,y)\in \text{pixels on } i} (I_2(x-u, y) - I_1(x, y))^2$$

is minimized (for example, using methods of Section 21.3). Then $(-u, 0)$ is the flow for that object. For the background, search for the c such that

$$\sum_{(x,y)\in \text{background pixels}} (I_2(x-c/y, y) - I_1(x, y))^2$$

is minimized; the flow at the background is the $(-c/y, 0)$.

Estimating which object a pixel belongs to: Now assume that you know the flow for each object and the background. Then assign each pixel to the segment that has the flow that produces the lowest photometric error. The choices are

$$\begin{cases} (I_2(x-c/y, y) - I_1(x, y))^2 & \text{if the pixel is background} \\ (I_2(x-c_1, y) - I_1(x, y))^2 & \text{if the pixel is on object 1} \\ \dots & \dots \\ (I_2(x-c_N, y) - I_1(x, y))^2 & \text{if the pixel is on object N} \end{cases}.$$

Notice how this approach links flow and segmentation. Pixels are collected together into a segment because they share flows; flows parameters are estimated from all pixels that have the same kind of flow. One significant advantage of this approach is that the estimated flow field can be discontinuous.

34.3.3 Estimating a Mixture of Affine Flows

A natural and widely applied flow model models image flow using multiple affine flows. Each pixel belongs to one of k segments. Each segment is associated with a set of affine flow parameters that explain its movement. This creates a flow model that admits discontinuities (like the simple mixture of flows) but where each particular flow can be quite complex. Just as in the previous section, if you know which flow model a pixel belongs to, you can estimate the parameters for that flow model by minimizing photometric error. Similarly, if you know the parameters of the flow models, you can tell which segment a pixel belongs to by checking which of the flow models minimizes the photometric error. Using the outline sketched in the previous section should seem reasonable to you.

The algorithm has one weakness. Each pixel is assigned to the segment whose flow yields the best photometric error. If two segments have about the same photometric error, this strategy could lose significant amounts of information. There is little justification for one segment's flow estimate getting all the constraint from that pixel, and the other segment's flow estimate getting none.

Improvements can be obtained with *soft assignment*, where each pixel can be assigned to more than one segment using a weight (the original strategy is then called *hard assignment*). To avoid overcounting, the weights must sum to one. The photometric error at a pixel yields a natural set of soft weights. Assume there are N different parametric models (equivalently, segments). Write θ_i for the parameters of the i'th segment, $\mathbf{f}(\mathbf{x}; i, \theta_i)$ for the flow predicted by the i'th model at $\mathbf{x}$ in the first frame and $E(\mathbf{x}; i, \theta_i) = (I_2(\mathbf{x} + \mathbf{f}(\mathbf{x}; i, \theta_i)) - I_1(\mathbf{x}))^2$ for the photometric error resulting from using the i'th segment to predict flow $\mathbf{x}$. Then a natural set of weights is

$$w_i(\mathbf{x}) = \frac{e^{\frac{-E(\mathbf{x}; i, \theta_i)}{2\sigma^2}}}{\sum_u e^{\frac{-E(\mathbf{x}; u, \theta_u)}{2\sigma^2}}}.$$

Here σ is a scale that you choose. Too small a value of σ and the soft assignment is largely a hard assignment because one weight is much larger than all the others. Too large a value of σ and the soft assignment strategy oversmoothes the estimated flow field.

34.4 REGRESSION METHODS FOR OPTIC FLOW

Flow prediction is naturally seen as a regression problem. The simplest approach stacks two images, passes the result into a unet, and trains the unet to produce the right flow field. This approach isn't ideal in practice. A natural improvement notices that estimating flow involves a comparison between the two images. There is relatively little training data with known flow, but methods can be trained very effectively on simulated data, and there are interesting prospects for self-training. This approach can be extended to use a form of matching at multiple scales, and now produces extremely good flow estimates.

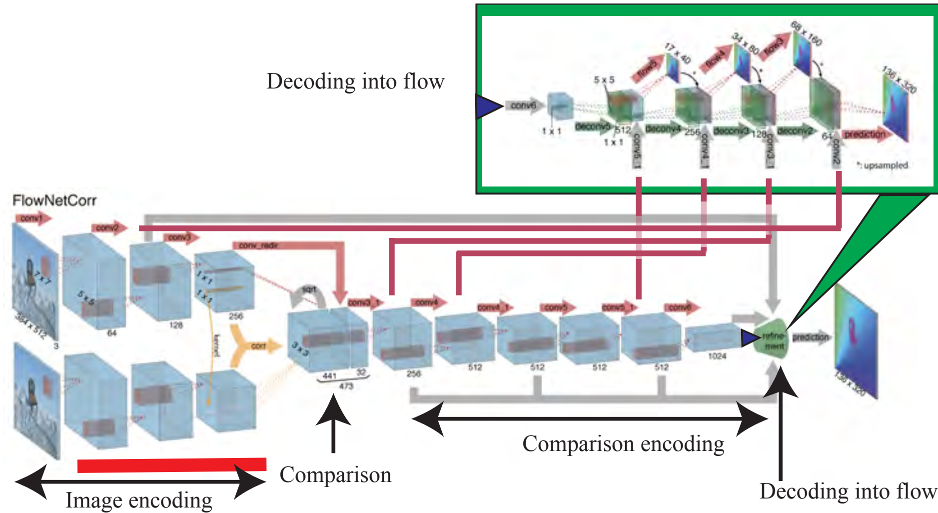

FIGURE 34.13: *The overall architecture of Flownet, a regression based optic flow predictor ([]). The frames pass through an image encoder; the results are compared (details in text), and the comparison further encoded. Finally, the result is decoded into a flow field, using skip connections from the comparison encoder and from the first frame encoder.* Image credit: *Figure 2 of* FlowNet: Learning Optical Flow with Convolutional Networks *by P. Fischer, A. Dosovitskiy, E.Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smag, D. Cremers, and T. Brox*

34.4.1 Simple Flow Regression

At the heart of an optical flow computation is a measure of how much the first image at some location matches the second image at another location. The image representations of Section 21.3 represent what an input image looks like at or around some location. In turn, just stacking two images and passing the result into an embedding does not really capture what an optical flow computation requires. Possibly there won't be a problem if you have enough parameters in the embedding, enough parameters in the decoder and enough training data.

Flownet [] computes an embedding for the two frames, then uses a special *correlation layer* to compare the two image representations. There is a limit to how far the pixel at location in image 1 can travel, so choose some upper bound $d_{\max}$ for the displacement. Choose a half size, k, so that $2k + 1 \times 2k + 1$ windows are compared. Set up the encoding so that location i, j in the encoding of an image corresponds to a feature vector representing the image around the location i, j in the input image. At i, j, there are $2d_{\max}+1 \times 2d_{\max}+1$ comparison values. Each one is obtained by comparing the $2k+1 \times 2k+1$ window centered at $i+u$, $j+v$ to the window centered at i, j using the sum of squared distances. Here u ranges from $-d$ to d, as does v, and you handle the edges of the image by padding. You should think of the resulting object as a block of features, which can be passed through convolutional layers, etc. just like other features.

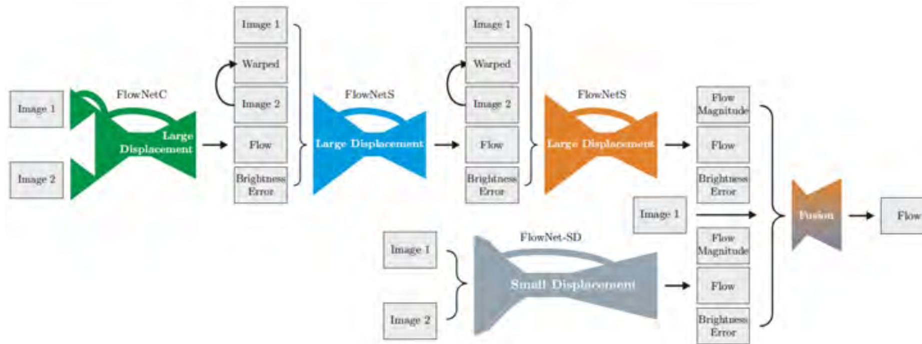

FIGURE 34.14: *Flownet2 obtains significant improvements by composing multiple Flownets. Compute a flow estimate with a flownet; now warp the second image to produce $\mathcal{W}$ (which would be frame 1 if you had the flow right). The magnitude of the residual between frame 1 and the warped version of frame 2 identifies errors in the flow. Provide this error, $\mathcal{I}_1$ and $\mathcal{W}$ to a version of flownet adjusted to accept the extra frames. This produces a revised flow. You can stack these revised versions as required (two seems to be enough). The resulting network is good at estimating large flows. Now build yet another flownet, allowing quite small displacements only. You now have two estimates of flow, so construct a fusion network that accepts the flow estimates and produces a final fused estimate. Show this network $\mathcal{I}_1$, the flow fields, the flow magnitudes, and the brightness errors (Figure* **??**). Image credit: *Figure 2 of* FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks*, by E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy and T. Brox.*

Figure 34.19 shows the overall architecture. Each of the two frames is passed through a copy of an image encoder. The comparison layer – which has no learnable parameters – then computes a comparison of the two frames. This block is further encoded by a series of convolutional layers. The result, together with skip connections from the comparison encoder and from the first frame encoder, is then decoded into a flow field.

34.4.2 Improving Flow Regression

Flownet was the first regression based flow predictor, but it wasn't particularly good. Significant improvements are obtained by using multiple flownets, by analogy with the iterative flow estimation of Section 34.2.5. Start with an instance original flownet, which produces a flow estimate. Use this flow estimate to produce $\mathcal{W}$, a warped version of frame 2. If the flow is correct, the warped estimate will largely look like frame 1, so the magnitude of the residual between frame 1 and the warped version of frame 2 will identify errors in the flow. Compute this brightness error ($\mathcal{E}$), and provide $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{W}$ and $\mathcal{R}$ to a version of flownet adapted to accept the extra frames. This produces a revised flow. You can stack these revised versions as required (two seems to be enough). You should see this stack as a version of iterative reestimation, as in Section 34.2.5.

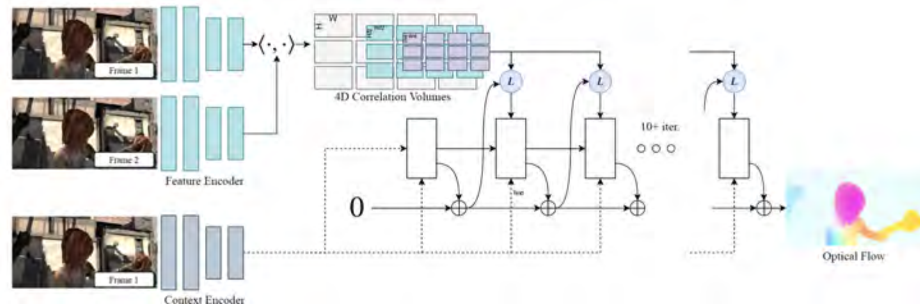

FIGURE 34.15: *RAFT encodes frame 1 and frame 2 to produce a feature representation, then computes an inner product between features for every pair of locations in frame 1 and frame 2. This volume of inner products is then pooled and subsampled to form representations at coarser and coarser scales. Pooling is done so you can tell where a specific pixel in frame 1, represented in the original resolution, went to in frame 2 at various resolutions. The result is a representation of flow at multiple levels of precision. An incremental procedure then starts with zero flow, and repeatedly updates the flow using the previous flow estimate, a context feature, and a readout from the multi-scale representation.* Image credit: *Figure 1 of* RAFT: Recurrent All-Pairs Field Transforms for Optical Flow *by Z. Teed and J. Deng.*

The resulting network should be good at estimating large flows, but may make errors with small flows. For an example, think about images showing a periodic structure. The flow may be too big by a fixed number of periods because the network is capable of comparing quite widely separated locations. This is easily dealt with. Build yet another flownet, allowing quite small displacements only. You now have two estimates of flow, so construct a fusion network that accepts the flow estimates and produces a final fused estimate. Show this network $\mathcal{I}_1$, the flow fields, the flow magnitudes, and the brightness errors (Figure **??**).

34.4.3 RAFT Flow Prediction

Flownet 2 parallels optimization based flow estimation in some ways. Section 34.2.5 might suggest other ways of assembling Flownets to obtain improved flow estimates. A good place to start would be some version of a pyramid or coarse-to-fine estimation. There is now a lively literature on this point; start with []. As of writing, to my knowledge, the best available regression based flow predictor is RAFT (for Recurrent All-pairs Flow Transforms). The paper is [], and a github page is at `https://github.com/princeton-vl/RAFT`.

RAFT uses a network structure that is informed by the key tricks used in optimization-based flow methods (Section 34.2.5): multi-scale representations and iterative flow estimation. Frame 1 and frame 2 are passed through an encoder to produce a feature representation. The correlation layer then computes an inner product between features for every pair of locations (so if the images are $W \times H$, the correlation layer produces a $W \times H \times W \times H$ data block). Think of the

FIGURE 34.16: *Flow fields predicted by RAFT are very close to ground truth when it is available. Strong discontinuities in flow generate no noticeable difficulties. These flow fields are obtained from two frames from the SINTEL dataset (***left***), compared with ground truth (***right***). SINTEL is a dataset of rendered frames from a CGI movie, so the true flow can be known. The direction of a flow vector is shown by hue (what changes from red to yellow to green to purple) and the magnitude by saturation (very small vectors are white, longer are pale colors, longest have intense colors).* Image credit: *Figure 1 of* RAFT: Recurrent All-Pairs Field Transforms for Optical Flow *by Z. Teed and J. Deng.*

resulting block as a set of fine-scale comparisons. The fine-scale comparisons are pooled to form comparisons at coarser and coarser scales, yielding a multi-scale representation. Comparisons are pooled across the last two dimensions only. This means you can determine where a specific pixel in frame 1 went to in frame 2 at various scales in the frame 2 representation. At each scale, the representation supports a flow arrow at every pixel, but different scales represent that flow arrow at different levels of precision. Knowing the location of the source pixel at the finest resolution yields better estimates of flow on small objects because they aren't subsampled away. Having the flow represented at a variety of precisions supports a coarse-to-fine search, giving a better chance of getting large movements right.

The comparisons are decoded into flow vectors by an incremental procedure. This starts with an estimate of zero flow. Each stage estimates an update to the previous stage's flow. The update is a function of the previous stage's flow, a context feature, and a readout from the multi-scale correlation representation. The context feature is taken from an encoding of the first frame. There are 10 or more (depending on the particular version) of these update stages. There is a strong analogy between this procedure and traditional optimization procedures. Like this procedure, traditional optimization minimizes a cost function by taking a sequence of steps – equivalently, updating an estimate of the location of the minimum a number of times. However, there is no explicit cost function and so no reason to believe that the updates are descent directions. Flow estimates are very strong

FIGURE 34.17: *For real images, there is no ground truth to compare to. This figure shows flow fields predicted by RAFT for two datasets. KITTI data has to do with autonomous vehicles. The DAVIS dataset is at* 1088×1920 *resolution. Flow fields should look plausible to you (for example, low flow at distant objects; strong discontinuities in flow around object boundaries).* Image credit: *Figure 1 of* RAFT: Recurrent All-Pairs Field Transforms for Optical Flow *by Z. Teed and J. Deng.*

(Figure **??**), inference is fairly fast (0.1s for a pair of images at 1088×436 resolution on a relatively outdated consumer GPU), and high resolution is possible.

34.4.4 More than Flow

A regression predictor that can predict flow well should be able to make a number of other possibly useful predictions. Natural candidates are motion boundaries (places where there is a very sharp change in flow) and occlusions (pixels that are visible in one frame, but not the other).

34.4.5 Datasets and Resources

It is not easy to find natural image pairs where the flow is known. One important strategy is to build a synthetic image dataset for which the flow is known. Important synthetic flow datasets include

- **Sintel** is a dataset of frames rendered from an open-source animated film. The film is "Sintel" which is an open source animated short film produced by Ton Roosendaal and the Blender Foundation. The dataset consists of long sequences with known flow, and can be found at `http://sintel.is.tue.mpg.de`.

- **Flying chairs** is a dataset consisting of 22872 image pairs with known flow fields. The frames show renderings of 3D chair models moving in front of images downloaded from the internet. The motion of chairs is planar, as is the

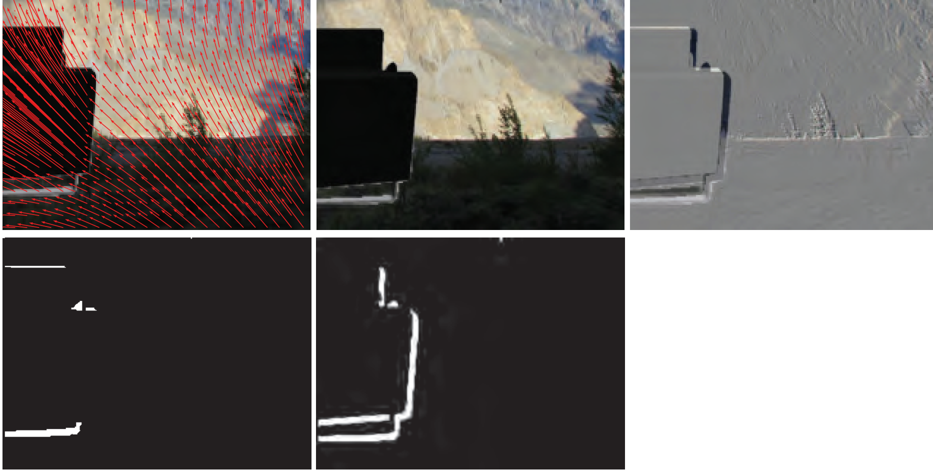

FIGURE 34.18:

motion of backgrounds. **Flying chairs 2** contains more information about the frames, including information about occlusions and motion boundaries. These datasets can be found at `https://lmb.informatik.uni-freiburg.de/resources/datasets/FlyingChairs.en.html`.

- **Flyingthings3D** contains more than 39000 stereo frames rendered from synthetic sequences. There are associated annotations of a variety of kinds, including optical flow maps and full intrinsic and extrinsic camera data. The dataset can be found at `https://lmb.informatik.uni-freiburg.de/resources/datasets/SceneFlowDatasets.en.html#downloads`.

- **ChairsSDHom:** was built to produce flow magnitude histograms similar to those estimated from a real world motion dataset (UCF 101, a dataset of human actions, `ucf101`). Displacements tend to be small, and there are many untextured regions. **ChairsSDHom 2** contains more information about the frames, including information about occlusions and motion boundaries. These datasets can be found at `https://lmb.informatik.uni-freiburg.de/resources/datasets/FlyingChairs.en.html` (this isn't a typo; scroll down).

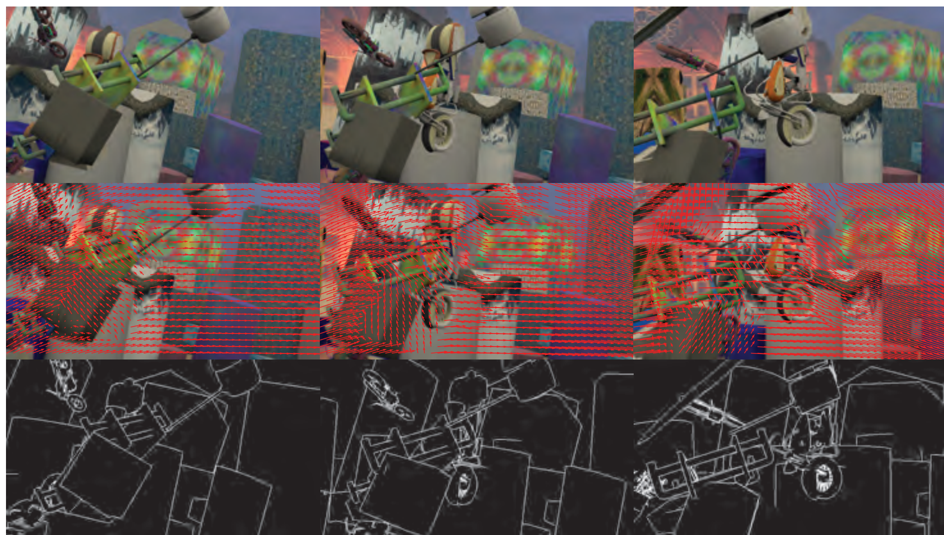

FIGURE 34.19:

34.5 YOU SHOULD

34.5.1 remember these facts:

34.5.2 remember these procedures:

34.5.3 exploit these resources:

34.5.4 be able to:

EXERCISES

QUICK CHECKS

LONGER PROBLEMS

PROGRAMMING EXERCISES