COMPUTER VISION

Linda G. Shapiro
Department of Computer Science and Engineering
Department of Electrical Engineering
University of Washington
Seattle, Washington
shapiro@cs.washington.edu

George C. Stockman
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan
stockman@cse.msu.edu

Prentice Hall

PRENTICE HALL, Upper Saddle River, New Jersey 07458

9 Motion from 2D Image Sequences

A changing scene may be observed via a sequence of images. One might learn a golf swing by observing the motions of an expert with a video camera, or better understand root growth by observing the root using many images taken hours apart. Action phenomena observed over time can be due to the motion of objects or the observer, or both. Changes in an image sequence provide features for detecting objects that are moving or for computing their trajectories. In case the viewer is moving through a relatively static world, image changes allow for computing the motion of the viewer in the world.

Similarly changing pixels in an image provide an important feature for object detection and recognition. Motion can reveal the shape of an object as well as other characteristics, such as speed or function. Analysis of object motion over time may be the ultimate goal; for instance, in controlling traffic flow or in analyzing the gait of a person with a new prosthesis. Today, a huge amount of videos are made to record events and structure in the world. It is necessary to have methods of segmenting these image sequences into meaningful events or scenes for easy access, analysis, or editing.

This chapter concentrates on detection of motion from 2D images and video sequences and the image analysis used to extract features. Methods for solution of the application problems just mentioned are discussed. Analysis of 3D structure and motion derived from 2D images is discussed in Chapter 13.

9.1 MOTION PHENOMENA AND APPLICATIONS

It is useful to consider the various cases of motion observable in an image sequence and the several important related applications. The problems to be solved range from mere detection of a moving object to analyzing the related motion and shape of multiple moving objects.

We identify the following four general cases of motion. We use the term *camera* to be interchangable with the term *observer.*

- Still camera, single moving object, constant background,
- Still camera, several moving objects, constant background,
- Moving camera, relatively constant scene,
- Moving camera, several moving objects.

The simplest case occurs when a still sensor stares at a relatively constant background. Objects moving across that background will result in changes to the image pixels associated with the object. Detection of these pixels can reveal the shape of the object as well as its speed and path. Such sensors are commonly used for safety and security. It is common for homes to use such sensors to automatically switch on a light upon detection of significant motion, which might be due to the owner coming home or to an unwelcome intruder. These simple motion sensors can also be used in manufacturing to detect the presence of a part fed into a workspace or in traffic control systems that detect moving vehicles.

A staring camera can also provide data for analysis of the movements of one or several objects. Moving objects must be tracked over time to produce a trajectory or path, which in turn can reveal the behavior of the object. For example, a camera might be used to analyze the behavior of people who enter the lobby of some business or who use some workspace. Several cameras can be used to produce different views of the same object, thus enabling the computation of paths in 3D. This is often done in the analysis of the motion of athletes or patients in rehabilitation. A system currently under development tracks the players and ball in a tennis match and provides an analysis of the elements of the game.

A moving camera creates image change due to its own motion, even if the 3D environment is unchanging. This motion has several uses. First, it may create more observations of the environment than available from a single viewpoint—this is the case when a panning camera is used to provide a wider (panoramic) view of the scene. Second, it can provide for computation of relative depth of objects since the images of close objects change faster than the images of remote objects. Third, it can provide for perception and/or measurement of the 3D shape of nearby objects—the multiple viewpoints allow for a triangulating computation similar to binocular stereo. In processing or analyzing video or film content, it is often important to detect points in time when the camera is panned or zoomed: In this case, we may not be interested in the contents of the scene but rather in the manner in which the scene was viewed.

The most difficult motion problems involve moving sensors and scenes containing so many moving objects that it is difficult to identify any constant background. Such a case arises when a robot vehicle navigates through heavy traffic. Another interesting case might be using communicating cameras to make correspondences in their observations in order to track several moving objects in the workspace.

The next sections examine various image analysis methods which analyze a sequence of two or more images in order to detect changes due to motion, or to analyze the objects themselves or their motion.

Exercise 9.1

Locate a motion detector in your neighborhood that is used to switch on a light. These devices are commonly used near a garage or house entry. Verify that quickly entering the area switches on a light. (a) Experiment by walking very slowly. Can you fool the motion detector into missing you? (b) What does this experiment tell you about how the motion detector works? (c) How does this relate to the Tyrannosaurus rex in the movie *Jurassic Park?*

9.2 IMAGE SUBTRACTION

Image subtraction was introduced in Chapter 1 as a means of detecting an object moving across a constant background. Suppose a video camera provides thirty frames per second of a conveyor belt that creates a uniform dark background. As brighter objects move across the camera view, the forward and rear edges of the object advance only a few pixels per frame. By subtracting the image I_t from the previous image I_{t-1}, these edges should be evident as the only pixels significantly different from zero.

Figure 9.1 shows the results of differencing over an interval of a few seconds for the purpose of monitoring a workspace (surveillance). A background image, which is nonuniform in this case, is derived from many video frames. A person who enters the workspace changes a region of the image, which can be detected by image subtraction as shown in Figure 9.1. The bounding box delimits a rectangular region where the change is detected. Further analysis of this bounding box might reveal object shape and even object type. The center image of Figure 9.1 actually shows three separate regions of change corresponding to (1) the person; (2) the door that the person opened; and (3) a computer monitor. A surveillance system may be provided the knowledge of the location of such objects and might even be primed to observe them or to ignore them. For example, the door might be carefully monitored while the CRTs are ignored. Related applications include monitoring and inventory of parking lots and monitoring the flow of vehicles on streets or people in rooms.

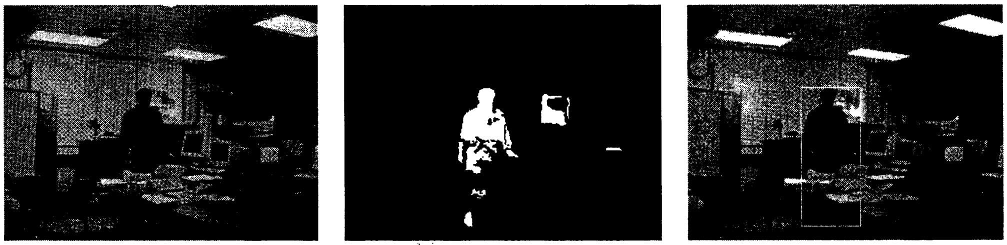


Figure 9.1 (left) A person appears in a formerly unoccupied workspace; (center) image substraction reveals changed regions where the person occludes the background and at the door and a CRT; and (right) the change due to the person is deemed significant while the other two are expected and hence ignored. (Images courtesy of S.-W. Chen.)

```
Detect changes between two images

Input $I_t[r, c]$ and $I_{t-\Delta}[r, c]$: two monochrome input images taken $\Delta$ seconds apart.
Input $\tau$ is an intensity threshold.
$I_{out}[r, c]$ is the binary output image; B is a set of bounding boxes.

1. For all pixels [r,c] in the input images,
   set $I_{out}[r, c] = 1$ if $(|I_t[r, c] - I_{t-\Delta}[r, c]| > \tau)$
   set $I_{out}[r, c] = 0$ otherwise.
2. Perform connected components extraction on $I_{out}$.
3. Remove small regions assuming they are noise.
4. Perform a closing of $I_{out}$ using a small disk to fuse neighboring regions.
5. Compute the bounding boxes of all remaining regions of changed pixels.
6. Return $I_{out}[r, c]$ and the bounding boxes B of regions of changed pixels.
```

Algorithm 9.1 Detection of change via image subtraction.

A sketch of the steps in change detection via image subtraction is given as Algorithm 9.1. Steps of this algorithm are operations given in Chapter 3.

Exercise 9.2

This exercise requires a workstation equipped with an attached camera and software to access the frames from the camera. Write a program that monitors the top of your desk (next to the workstation). The program should successively acquire frames, compute a histogram for each, and sound an alarm whenever there is a significant change in the histogram. Test your program on various still scenes and by moving various objects on and off the desk.

9.3 COMPUTING MOTION VECTORS

Motion of 3D scene points results in motion of the image points to which they project. Figure 9.2 shows three typical cases. Zooming out can be performed by reducing the focal length of a still camera or by backing away from the scene while keeping the focal length fixed. The optical axis points toward a scene point whose image does not move: this is the *focus of contraction. Zooming in* is performed by increasing the focal length of a still camera or by moving toward a particular scene point whose image does not change (*focus of expansion*). Panning a camera or turning our heads causes the images of the 3D scene points to translate, as shown at the right in Figure 9.2.

70 Definition. A 2D array of 2D vectors representing the motion of 3D scene points (as shown in Figure 9.2) is called the **motion field.** The motion vectors in the image represent the displacements of the images of moving 3D points. Each motion

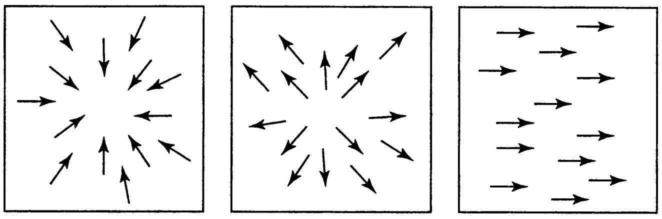


Figure 9.2 Effects of zooming and panning on imaged features. The effect of zoom in is similar to that observed when we move forward in a scene, and the effect of panning is similar to that observed when we turn.

vector might be formed with its tail at an imaged 3D point at time t and its head at the image of that same 3D point imaged at time $t + \Delta$. Alternatively, each motion vector might correspond to an instantaneous velocity estimate at time t.

71 Definition. The **focus of expansion (FOE)** is that image point from which all motion field vectors diverge. The FOE is typically the image of a 3D scene point toward which the sensor is moving. The **focus of contraction (FOC)** is that image point toward which all motion vectors converge, and is typically the image of a 3D scene point from which the sensor is receding.

Computation of the motion field can support both the recognition of objects and an analysis of their motion. One of two—not too constraining—assumptions is usually made in order to compute motion vectors. First, we can assume that the intensity of a 3D scene point P and that of its neighbors remain nearly constant during the time interval (t_1, t_2) over which the motion estimate for P is made. Alternatively, we can assume that the intensity differences observed along the images of the edges of objects are nearly constant during the time interval (t_1, t_2).

72 Definition. **Image flow** is the motion field computed under the assumption that image intensity near corresponding points is relatively constant.

Two methods for computing image flow are given next. Before developing these we describe a video game application that uses motion fields.

9.3.1 The Decathlete Game

Researchers at Mitsubishi Electric in Sagamihara, Japan, and at (Mitsubishi Electric Research Laboratory) MERL in Cambridge, Massachusetts have reported results of using motion analysis to control the Sega Saturn *Decathlete* game. They replaced a keypad interface with control through image flow computation from a low-resolution camera. The actual motion of the player of the game was used to control the motion of the player's avatar in the simulation. In the example shown here, the avatar is running the hurdles against another simulated runner. In Figure 9.3 the man at the left is the player and is making running motions with his arms and hands. The faster the motions, the faster his avatar runs. The avatar

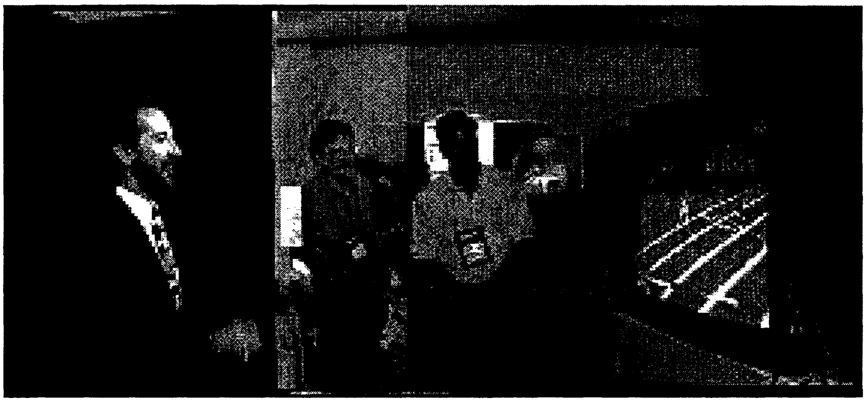


Figure 9.3 The man at the left is making running motions with his arms and hands to control the game of running the hurdles. The game display is shown at the right. In the lower right corner, a video camera (Mitsubishi Electric's CMOS image sensor with on-chip image processing) observes the motion of the player, which is used to control the running speed and jumping of the hurdling avatar. (Reprinted from *IEEE Computer Graphics,* vol. 18, no. 3 (May–June 1998) by permission of IEEE.)

must also jump the hurdles at the proper instants of time: The player jumps by raising both fists upward. The *Decathlete* display is shown on the monitor at the right in Figure 9.3. In the lower right corner of that figure, one can see the video camera that observes the motion of the player. The two persons in the middle of the picture are watching the fun.

Figure 9.4 illustrates the motion analysis used to control the hurdling game. Figure 9.4(a) gives a snapshot of the motion analysis, while Figure 9.4(b) provides an explanatory map of the content of (a). Note that the top left of (a) shows a video frame of the *running* player seen by the camera, while the middle left of (a) shows motion vectors extracted from multiple frames. The bottom left of (a) shows a time history of the average horizontal motion over the video frame; the middle of (a) shows the average vertical motion.

The camera must be set up to view the player's two hands as the player "runs" and "jumps." A running motion causes alternations in the horizontal average motion. The frequency of those alternations indicates how fast the player is running in place, which controls the runner's speed. A jumping motion causes the average vertical velocity to exceed a threshold, which sends a jump command to the game. The spatial resolution is very low but the temporal resolution is high.

The type of coarse motion analysis used in the decathalon game might provide a general gesture interface to computers. For example, future computers might provide for input using American Sign Language (ASL) or some smaller gesture language.

9.3.2 Using Point Correspondences

A sparse motion field can be computed by identifying pairs of points that correspond in two images taken at times t_1 and $t_1 + \Delta$. The points we use must be distinguished in some way so that they can be identified and located in both images. Detecting corner points or

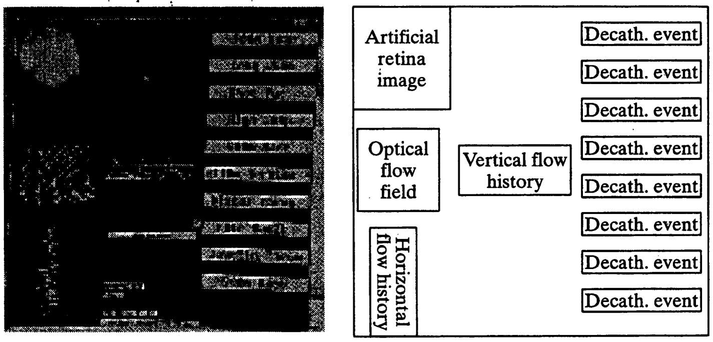


Figure 9.4 Illustration of the motion analysis used to control the hurdling game. The top left shows a video frame of the running player, while the middle left shows motion vectors extracted from multiple frames. The jumping of the hurdles is indicated by the vertical motion plot shown in the middle. (Reprinted from *IEEE Computer Graphics,* vol. 18, no. 3 (May–June 1998) with permission of IEEE.)

high interest points should work for both color and monochrome images. Alternatively, centroids of persistent moving regions from segmented color images might be used. Corner points can be detected by using masks such as the Kirsch edge operator or the ripple masks of the Frie-Chen operator set (Chapter 5). Alternatively, an *interest operator* can be used. The operator computes intensity variance in the vertical, horizontal, and diagonal directions through the neighborhood centered at a pixel **P**. Only if the minimum of these four variances exceeds a threshold is **P** passed as an interesting point. These operations are sketched in Algorithm 9.2. An alternative operator based on texture is developed in Exercise 9.3.

Exercise 9.3: Texture-based interest operator.

Experiment with the following interest operator, which is based on the texture of an entire $n \times n$ neighborhood. First, compute the gradient magnitude for the entire input image using a 3×3 or 2×2 mask. Second, threshold the magnitude image to produce a binary image. A pixel $[r, c]$ in the original image is interesting only if there is significant variation in each of the four main directions in the $n \times n$ neighborhood of $B[r, c]$ in the binary image. Variation in direction $[\Delta r, \Delta c] = [0, 1], [1, 0], [1, 1], [1, -1]$ is just the sum of $B[r, c] \otimes B[r + \Delta r, c + \Delta c]$ for all pixels in the $n \times n$ neighborhood centered at $B[r, c]$. $\otimes$ is the **exclusive or** operator that returns 1 if and only if the two inputs are different. An *interest image* is formed by assigning IN[r, c] the minimum of the four variations in $B[r, c]$ computed as above. Try your operator on a few monochrome images, including a checkerboard.

Once a set of interesting points $\{P_j\}$ is identified in the image I_1 taken at time t, corresponding points must be identified in the image I_2 taken at time $t + \Delta$. Rather than

```
Find interesting points of a given input image.

procedure detect_corner_points(I, V);
{
        "I[r, c] is an input image of MaxRow rows and MaxCol columns"
        "V is an output set of interesting points from I."
        "$\tau$ is a threshold on the interest operator output"
        "$w$ is the halfwidth of the neighborhood for the interest operator"
        for r := 0 to MaxRow - 1
           for c := 0 to MaxCol - 1
              {
              if I[r,c] is a border pixel then break;
              else if (interest_operator (I, r, c, w ) $\geq \tau_1$) then add
              [(r,c),(r,c)] to set V;
        "The second (r, c) is a place holder in case vector tip found later."
              }
}
real procedure interest_operator (I, r, c, w)
{
        "$w$ is the halfwidth of operator window"
        "See alternate texture-based interest operator in the exercises."
        v1 := variance of intensity of horizontal pixels $I_1[r, c - w] \ldots I_1[r, c + w]$;
        v2 := variance of intensity of vertical pixels $I_1[r - w, c] \ldots I_1[r + w, c]$;
        v3 := variance of intensity of diagonal pixels
                                    $I_1[r - w, c - w] \ldots I_1[r + w, c + w]$;
        v4 := variance of intensity of diagonal pixels
                                    $I_1[r - w, c + w] \ldots I_1[r + w, c - w]$;
        return minimum $\{v1, v2, v3, v4\}$;
}
```

Algorithm 9.2 Algorithm for detecting interesting image points.

extract points from I_2 in the same manner and then attempt to make correspondences, we can directly search I_2 to determine the new location of each point from I_1. This can be done by using the cross-correlation method described in Chapter 5. Given an interesting point P_j from I_1, we take its neighborhood in I_1 and find the best correlating neighborhood in I_2 under the assumption that the amount of movement is limited. Figure 9.5 sketches how to search frame I_2 for a good match to the neighborhood of point P_j in frame I_1. The center $P_k = [P_{kr}, P_{kc}]$ of the best correlating neighborhood in $\mathbf{I}_2$ is taken to be the corresponding point and will become the tip of a motion vector with $P_j = [P_{jr}, P_{jc}]$ being the tail. The search for P_k is limited to a rectangular $C \times R$ region in image rows $P_{jr} - R \ldots P_{jr} + R$ and image columns $P_{jc} - C \ldots P_{jc} + C$. A small search region speeds up the search for a match and also reduces ambiguity, but is only useful when there is a justifiable assumption on the limit to the velocity of objects. The resulting algorithm is given as Algorithm 9.3.

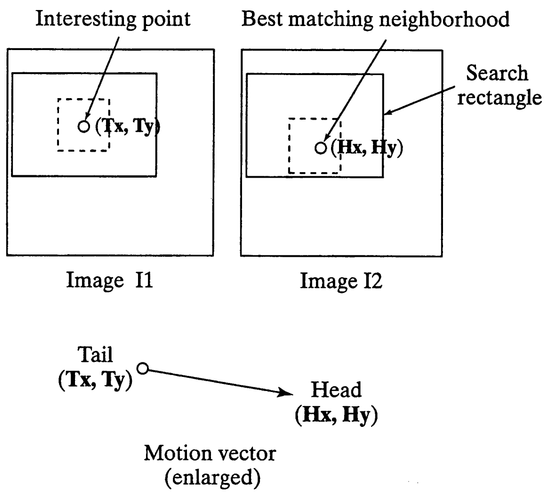


Figure 9.5 For each interesting point (**Tx, Ty**) in image $\mathbf{I}_1$ a rectangular region of image $\mathbf{I}_2$ is searched for the best match to a small neighborhood of (**Tx, Ty**). If the best match is good, then it becomes the head (**Hx, Hy**) of a motion vector.

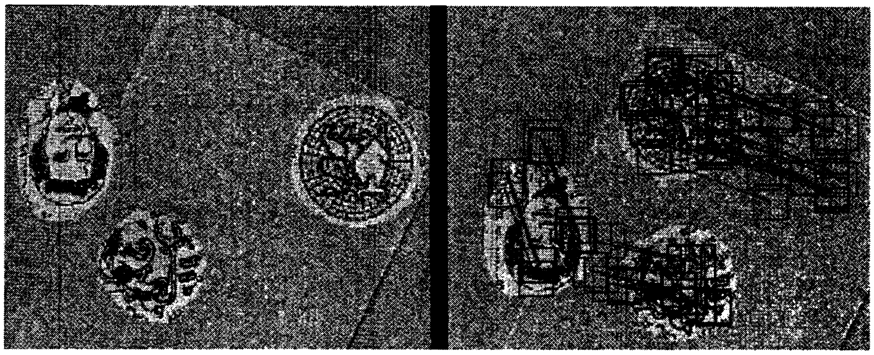


Figure 9.6 Results of applying Algorithm 9.3. At the left is the image at time t_1. At the right is the image at time t_2 with the motion analysis overlaid. Red squares indicate the location of the original neigborhoods detected by the interest operator in the image at the left. Blue squares indicate the best matches to these neighborhoods in the image at the right. There are three coherent sets of motion vectors (green lines) corresponding to the three moving objects. The leftmost object moved down and slightly right. The lowest object moves right and slightly down; the rightmost object moves left and slightly up. (Analysis courtesy of Adam T. Clark.) See colorplate.

Results of applying this algorithm are shown in Figure 9.6. The test imagery was created by moving three highly textured cutouts across a lightly textured background.

Algorithm 9.3 can be controlled to iterate through pairs of frames so that feature points can be continuously tracked over many frames. The corner points identified in frame $t + \Delta$ can replace those formerly identified in frame t and the new, possibly changed, neighborhoods used for cross-correlation. In this manner, significant points can be tracked in a dynamic scene provided that their neighborhoods change in a gradual manner. In general, we must also provide for the disappearance of corner points due to occlusion and the appearance of new unoccluded corner points. These ideas are discussed in Section 9.4.

```
From two input images, derive motion vectors for interesting points.

I1[r, c] and I2[r, c] are input images of MaxRow rows and MaxCol columns.
V is the output set of motion vectors $\{[(T_x, T_y), (H_x, H_y)]_i\}$
where $(T_x, T_y)$ is the tail of a motion vector and $(H_x, H_y)$ is its head.

procedure extract_motion_field(I1, I2, V)
{
"Detect matching corner points and returning motion vectors V"
"$\tau_2$ is a threshold on neighborhood cross-correlation"
    detect_corner_points(I1, V);
    for all vectors $[(T_x, T_y), (U_x, U_y)]$ in V
      match := best_match( I1, I2, $T_x, T_y, H_x, H_y$);
      if ($match < \tau_2$) then delete $[(T_x, T_y), (U_x, U_y)]$ from V;
      else replace $[(T_x, T_y), (U_x, U_y)]$ with $[(T_x, T_y), (H_x, H_y)]$ in V;
}
real procedure best_match( I1, I2, $T_x, T_y, H_x, H_y$);
"$(H_x, H_y)$ is returned as the center of the neighborhood in I2 that matches best"
"to the neighborhood centered at $(T_x, T_y)$ in I1."
"sh and sw define search rectangle: h and w define neighborhood size."
{
    "first indicate that a good match has not been found"
    $H_x$ := -1; $H_y$ := -1;   best := 0.0;
    for r := $T_y - sh$ to $T_y + sh$
      for c := $T_x - sw$ to $T_x + sw$
        {
          "cross correlate N in I1 with N in I2 as in Chapter 5"
          match := cross_correlate(I1, I2, $T_x, T_y, r, c, h, w$);
          if ( $match > best$) then
            {
              $H_y$ := r;   $H_x$ := c;   best := match;
            }
        }
}
```

Algorithm 9.3 Compute sparse set of motion vectors from a pair of input images.

Exercise 9.4

Consider the image of a standard checkerboard: (a) Design a corner detector that will respond only at the corners of four of the squares and not within squares or along the edges between two squares; (b) take several images by slowly moving a checkerboard in front of a stationary camera; (c) test your corner detector on these images and report on the number of correct and incorrect detections in each of the images; and (d) implement and test Algorithm 9.3 on several pairs of the images with a small amount of motion between them.

9.3.3 MPEG Compression of Video

MPEG video compression uses complex operations to compress a video stream up to 200:1. We note the similarity between MPEG motion compensation and Algorithm 9.3. The subgoal of MPEG is not to compute a motion field but to compress the size of an image sequence by predicting the content of some frames from other frames. It is not important that the motion vectors be correct representations of moving objects, but only that they provide for good approximations of one image neighborhood from another. An MPEG encoder replaces an entire 16 × 16 image block in one frame with a motion vector defining how to locate the best matching 16 × 16 block of intensities in some previous frame. Figure 9.7 illustrates the use of motion estimation in the MPEG compression scheme. Details of the scheme are given in the figure caption. Distinguished image points are not used; instead a uniform grid of blocks is used and a match of each block is sought by searching a previous image of the video sequence. The figure only shows how a few of the many blocks are computed. Ideally, each block B_k of 16 × 16 pixels can be replaced by a single vector $[V_x, V_y]_k$, which the encoder found to locate the best block of matching intensities in a previous frame. If there is a difference in the two intensity arrays, then this difference can be represented by a small number of bits and transmitted also.

Although the MPEG motion vectors are designed for the purpose of compression and not motion analysis, researchers have begun to experiment with using them for the purpose of providing a motion field. The advantage is that MPEG encoders can now compute these

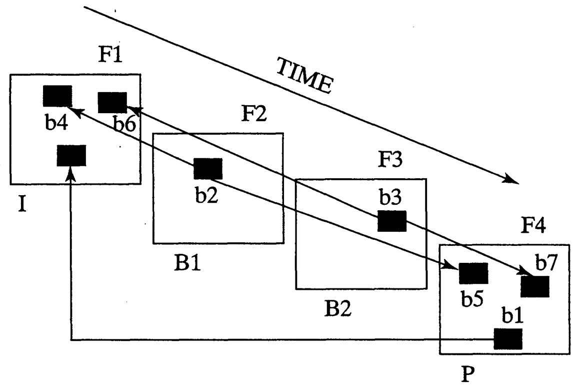


Figure 9.7 A coarse view of the MPEG use of motion vectors to compress the video sequence of four frames F1, F2, F3, F4. F1 is coded as an *independent* (I) frame using the JPEG scheme for single still images. F4 is a P frame predicted from F1 using motion vectors together with block differences: 16 × 16 pixel blocks (b1) are located in frame F1 using a motion vector and a block of differences to be added. Between frames B1 and B2 are determined entirely by interpolation using motion vectors: 16 × 16 blocks (b2) are reconstructed as an average of blocks (b4) in frame F1 and (b5) in frame F4. Between frames F2 and F3 can only be decoded after predicted frame F4 has been decoded even though these images were originally created before F4. Between frames yield the most compression since each 16 × 16 pixel block is represented by only two motion vectors. I frames yield the least compression.

vectors in real time and they are already present in the video stream. Options in future codecs may indeed provide useful motion fields for motion analysis applications.

Exercise 9.5

Assume a video sequence of frames that each have 320 × 240 8-bit monochrome pixels. (a) What is the representation output by an MPEG-type encoder for a between frame? (b) How many bytes are needed for the representation? (c) What is the compression ratio for the between frame relative to an original image?

9.3.4 Computing Image Flow*

Methods have been developed for estimating image flow at all points of an image and not just at interesting points. We study a classical method that combines spatial and temporal gradients computed from at least two consecutive frames of imagery. Figure 9.8 shows an ideal example of what a camera might observe when an object moves in its field of view. The image (a) from time t_1 shows a triangular object in the lower left corner, while (b) shows that the triangle has moved upward by time t_2. This simple example serves to illustrate some of the assumptions that we will make to develop a mathematical model of image flow.

- We assume that the object reflectivity and the illumination of the object do not change during the interval $[t_1, t_2]$.
- We assume that the distances of the object from the camera or light sources do not vary significantly over this interval.
- We shall also assume that each small intensity neighborhood $N_{x,y}$ at time t_1 is observed in some shifted position $N_{x+\Delta x, y+\Delta y}$ at time t_2.

These assumptions do not hold tight in real imagery, but in some cases they can lead to useful computation of image flow vectors. Having motivated the theory with a simple discrete case, we now proceed with the development of the image flow equation for the case of a continuous intensity function $f(x, y)$ of continuous spatial parameters.

```
3333333333    3333333333
3333333333    3333333333
3333333333    3373333333
3373333333    3397533333
3397533333    3399753333
3399753333    3399975333
3399975333    3333333333
3333333333    3333333333
  (a) t1        (b) t2
```

Figure 9.8 An example of image flow. A brighter triangle moves one pixel upward from time t_1 to time t_2. Background intensity is 3 while object intensity is 9.

Exercise 9.6

Refer to Figure 9.8. The intensity function is $f(x, y, t)$. Consider the topmost pixel at time t_1 with intensity 7 within the context 9 7 5 (its spatial coordinates are $x = y = 4$). Estimate the spatial derivatives of the image function $\partial f/\partial x$ and $\partial f/\partial y$ at $x = y = 4$; $t = t_1$ using a 3×3 neighborhood. Estimate the temporal derivative $\partial f/\partial t$ at $x = y = 4$; $t = t_1$: what method is appropriate?

9.3.5 The Image Flow Equation*

Using the previous assumptions, we now derive what is called the *image flow equation* and show how it can be used to compute image flow vectors. Using the continuous model of the intensity function $f(x, y, t)$, we apply its Taylor series representation in a small neighborhood of an arbitrary point (x, y, t).

$$f(x+\Delta x, y+\Delta y, t+\Delta t) = f(x, y, t) + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y + \frac{\partial f}{\partial t}\Delta t + h.o.t. \tag{9.1}$$

Note that Equation 9.1 is a multivariable version of the very intuitive approximation for the one variable case: $f(x+\Delta x) \approx f(x) + f'(x)\Delta x$. For small neighborhoods of (x, y, t) we ignore the higher order terms $h.o.t.$ of Equation 9.1 and work only with the linear terms. Our next crucial step is illustrated in Figure 9.9. The image flow vector $\mathbf{V} = [\Delta x, \Delta y]$ for which we want to solve carries the intensity neighborhood N_1 of (x, y) at time t_1 to an identical intensity neighborhood N_2 of $(x+\Delta x, y+\Delta y)$ at time t_2. This assumption means that

$$f(x+\Delta x, y+\Delta y, t+\Delta t) = f(x, y, t) \tag{9.2}$$

We obtain the image flow equation by combining Equations 9.1 and 9.2 and ignoring the higher order terms.

$$-\frac{\partial f}{\partial t}\Delta t = \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] \circ [\Delta x, \Delta y] = \nabla f \circ [\Delta x, \Delta y] \tag{9.3}$$

The image flow equation does not give a unique solution for the flow vector $\mathbf{V}$, but imposes a linear constraint on it. In fact, there may be many neighborhoods N_2 that have the same intensities as N_1. Figure 9.10 shows how multiple possibilities exist for the flow vector when restricted to a small neighborhood, or *aperture,* about point (x, y). Observing only the small aperture about point $\boldsymbol{P}$, it is possible that $\boldsymbol{P}$ moves to $\boldsymbol{R}$ or $\boldsymbol{Q}$ or some other

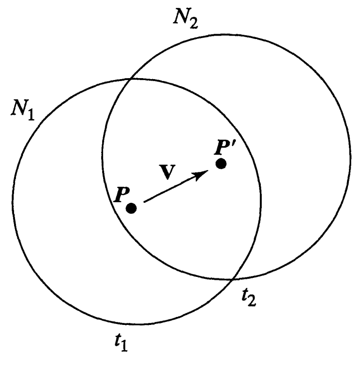


Figure 9.9 Due to motion in direction **v**, the intensities of neighborhood N_1 at time t_1 are the same as the intensities of neighborhood N_2 at time t_2.

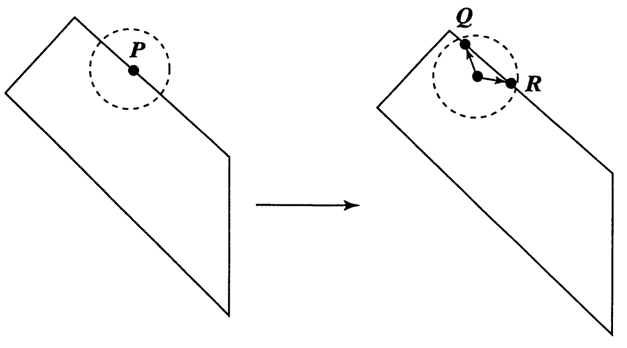


Figure 9.10 An intensity edge moves toward the right from time t_1 to time t_2. However, due to the limited size of the neighborhood, or *aperture*, used for matching, the location of the displaced point ***P*** could be ***R*** or ***Q*** or some other point along the edge segment determined by them.

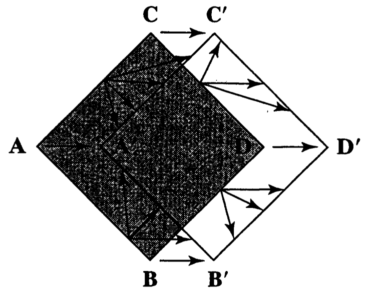


Figure 9.11 A square object is moving toward the right. Motion vectors with their tails on the edge at time t_1 are constrained by a linear relationship that puts their heads on an edge at time t_2. Common constraints at the corners A, B, C, D force the *move right* interpretation, which can then be propagated to all edge points by enforcing consistency along the entire boundary.

point along the segment ***QR***. Figure 9.11 shows four different edge cases from a square object. In general, we may not have pronounced object edges; however, Figure 9.9 still applies to curved isobrightness contours. The edges shown in the picture would be the tangents to the contour that approximate the contour locally.

We can interpret Figure 9.10 as follows. A change is observed at point ***P*** and can be measured by $-\frac{\partial f}{\partial t}\Delta t$. This change equals the dot product of the spatial gradient ∇f and the flow vector $\mathbf{V}$. $|\mathbf{V}|$ can be as small as the perpendicular distance to the new edge location, or it can be much larger in case the flow vector is in a direction much different from the spatial gradient. The latter case would result when a rope being pulled very fast vertically vibrates slightly horizontally resulting in small changes of image edge position over time.

9.3.6 Solving for Image Flow by Propagating Constraints*

The image flow equation provides a constraint that can be applied at every pixel position. By assuming coherence, neighboring pixels are constrained to have similar flow vectors. Figure 9.12 shows how neighboring constraints can be used to reduce the ambiguity in the direction of motion. Figure 9.12(*b*) shows a blowup of the neighborhood of corner A of the moving square shown in (*a*). The image flow equation constrains the direction θ_x of motion at point X to be in the interval between $5\pi/4$ and $\pi/4$. The direction θ_y of motion at point Y is constrained to the interval between $-\pi/4$ and $3\pi/4$. If points X and Y are assumed to be on the same rigid object, then the flow vectors at X and Y are now constrained to the intersection of these constraints, which is the interval $-\pi/4$ and $\pi/4$.

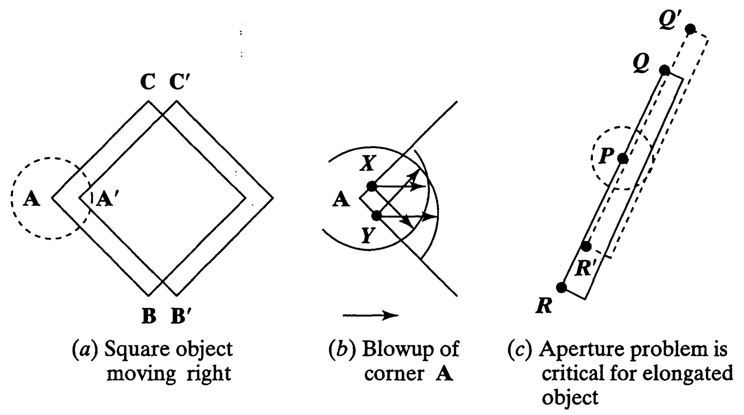

C C′
A
A′
B B′
(a) Square object moving right
X
A
Y
(b) Blowup of corner A
Q′
Q
P
R′
R
(c) Aperture problem is critical for elongated object

Figure 9.12 (*a*) A square object moves right; (*b*) a blowup of corner *A* shows how the constraints from two neighboring image flow equations can reduce the ambiguity of direction to $\pi/2$, and (*c*) an extreme aperture problem: an elongated object moves in the direction of its length; from the aperture around point *P* and its neighbors, the ambiguity in the direction of motion is π.

Figures 9.11 and 9.12 emphasize two points. First, only at the interesting corner points can image flow be safely computed using small apertures. Second, constraints on the flow vectors at the corners can be propagated down the edges; however, as Figure 9.12(c) shows, it might take many iterations to reach an interpretation for edge points, such as P, that are distant from any corner. Some experiments in flow computations have been performed using random pixel images. Our development shows that such images are probably easier than highly structured images because neighborhoods are more likely to be unique. Relaxation methods in 2D are studied in Chapter 11. Use of differential equations to solve for image flow can be found in the paper by Horn and Schunck (1981) cited in the references in Section 9.6.

9.4 COMPUTING THE PATHS OF MOVING POINTS

The previous sections discussed methods for identifying interesting points of a frame at time t_1 and for locating that same object point in a following frame at time t_2 close in time to t_1. If the intensity neighborhood of each point is uniquely textured, then we should be able to track the point over time using normalized cross-correlation. Also, domain knowledge might make it easy to track an object in an image sequence, as in the case of tracking an orange tennis ball in a tennis match, or a pink face looking at a workstation.

We now consider a more general situation where moving points are not tagged with unique texture or color information. In this case, the characteristics of the motion itself must be used to collect points into trajectories. Figure 9.13 shows the smooth trajectories of three objects over six instances of time. We mention three concrete problems before considering the abstract general case. For one situation, consider a box of many tennis balls dumped out on the ground: the problem is to compute the path of each ball from a video sequence.

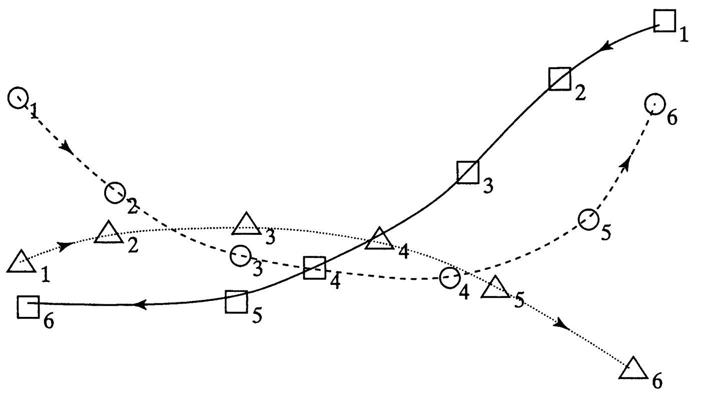


Figure 9.13 Trajectories of three objects, ○, △, □ are shown: the location of each object is shown for six instants of time. ○ and △ are generally moving from left to right while □ is moving right to left

In a second situation, we want to study fluid flow through a chamber by mixing fluorescent particles with the fluid and observing their motion over time. We assume that individual particles appear to be the same in an image. In a third case, we want to compute the paths of people walking in a mall. Clothing may make the images of some people unique, but surely there would be a high probability that some sets of individuals would have similar appearances in the images.

We can exploit the following general assumptions that hold for physical objects in 3D space.

1. The location of a physical object changes smoothly over time.
2. The velocity of a physical object changes smoothly over time: This includes both its speed and direction.
3. An object can be at only one location in space at a given time.
4. Two objects cannot occupy the same location at the same time.

The first three assumptions hold for the 2D projections of 3D space; smooth 3D motion creates smooth 2D trajectories. The fourth assumption may be violated under projection, since one object might occlude another, and this will create problems for a single observer. Experiments with humans have shown that humans can recognize objects and analyze their motion when presented with many frames of the moving object points. In a well-known experiment by Johansson (1976), lights were affixed to various points of a human body. Observers of only the lights on a still person could not recognize that it was a person; however, when the person moved, the observers could easily recognize that they were seeing a person.

We now present an algorithm, developed by Sethi and Jain (1987), that uses these four assumptions to compute a smoothest set of paths through points observed in a sequence of frames. First, we give a mathematical definition for the smoothness of a single path. Second, we define the smoothest set of m paths as that set of paths with the optimal sum of the m

smoothness values. Finally, we define the *Greedy Exchange Algorithm,* which iteratively extends the m paths from time t_1 through time t_n by making an optimal set of assignments at each time instant.

73 Definition. If an object i is observed at time instants $t = 1, 2, \ldots, n$, then the sequence of image points $T_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,t}, \ldots, p_{i,n})$ is called the **trajectory of i.**

Between any two points of a trajectory, we can define their difference vector

$$V_{i,t} = p_{i,t+1} - p_{i,t} \tag{9.4}$$

We can define a smoothness value at a trajectory point $p_{i,t}$ in terms of the difference of vectors reaching and leaving that point. Smoothness of direction is measured by their dot product, while smoothness of speed is measured by comparing the geometric mean of their magnitudes to their average magnitude.

$$S_{i,t} = w\left(\frac{V_{i,t-1} \circ V_{i,t}}{|V_{i,t-1}|\,|V_{i,t}|}\right) + (1-w)\left(\frac{2\sqrt{|V_{i,t-1}|\,|V_{i,t}|}}{|V_{i,t-1}| + |V_{i,t}|}\right) \tag{9.5}$$

The weight w of the two factors is such that $0 \le w \le 1$, which yields $0 \le S_{i,t} \le 1$ (See the exercises in this section.) Note that for a straight trajectory with equally spaced points all the difference vectors are the same and Equation 9.5 yields 1.0, which is the optimal point smoothness value. Change of direction or speed decreases the value of $S_{i,t}$. We assume that m unique points are extracted from each of the n frames, although we can relax this later. Points of the first frame are labeled $i = 1, 2, \ldots, m$. The problem is to construct the m trajectories T_i with the maximum total smoothness value. Total smoothness is defined in Equation 9.6 as a sum of smoothness of all the interior points of all the m paths.

$$\textit{total smoothness } T_s = \sum_{i=1}^{m}\sum_{t=2}^{n-1} S_{i,t} \tag{9.6}$$

Exercise 9.7

Assume that $w = 0.5$ so that direction and speed are weighted equally. (a) Show that point smoothness at each vertex of a regular hexagon with unit sides is 0.75. (b) What is the point smoothness at the vertices of a square?

Exercise 9.8

(a) Refer to the Cauchy-Schwartz inequality (Chapter 5) and show that the factor $\frac{V_{i,t-1} \circ V_{i,t}}{|V_{i,t-1}|\,|V_{i,t}|}$ of $S_{i,t}$ is between 0 and 1. (b) Show that for two positive numbers x and y, their geometric mean $\sqrt{xy}$ never exceeds their arithmetic mean $(x + y)/2$. Use this to show that the factor $\frac{2\sqrt{|V_{i,t-1}|\,|V_{i,t}|}}{|V_{i,t-1}| + |V_{i,t}|}$ is between 0 and 1. (c) Now show that $S_{i,t}$ in Equation 9.5 is between 0 and 1 provided that w is between 0 and 1.

Exercise 9.9

What is the total smoothness of these two paths: a 4-point trajectory along four sides of an octagon with sides s and a 4-point trajectory along a square with sides s?

Algorithm 9.4 develops a set of m trajectories over n frames. It is not guaranteed to produce the minimum possible T_s, but experiments have shown it to work well. First, we intuitively examine some of its operations with reference to the simple example in Figure 9.14. Table 9.1 provides a record of the smoothness of some paths considered. We can assign point labels arbitrarily in the first frame; for example, object $\square_1 \equiv 1 = T[1, 1]$ and object $\bigcirc_1 \equiv 2 = T[2, 1]$. The trajectories can then be extended to the nearest point in subsequent frames: $T[1, 2] = \bigcirc_2$, the closest point, and $T[2, 2] = \square_2$ by process of elimination. We have made a mistake by switching the actual trajectories. We compute the

```
Input sets of 2D points over time and compute smooth paths.
P[i, t] holds $i = 1, 2, \ldots, m$ 2D points from frames $t = 1, 2, \ldots, n$;
T[i, t] is the output set of trajectories of m rows and n columns.
T[i, t] = k means that object i is observed as the $k$-th point of frame $t$.

1. initialize: create $m$ complete paths by linking nearest neighbors
    (a) First frame: for all $i$, set object labels T[i, 1] = $i$;
    (b) Other frames: for $t = 2, 3, \ldots, n$, assign $\mathbf{T}[i, t] = k$ where point $\mathbf{P}[k,t]$
        is the nearest point to point $\mathbf{P}[\mathbf{T}[i,t-1],t-1]$ which is not already assigned.
2. Exchange Loop: for $t := 2$ to $n - 1$
    (a) for all pairs $(j,k)$ with $j \neq k$, compute the increase of smoothness due
        to exchanging the assignments $\mathbf{T}[j, t]$ and $\mathbf{T}[k, t]$;
    (b) make the exchange that produces the maximum increase in smoothness;
        make no exchange if none would increase total smoothness;
    (c) set the exchange flag on only if an exchange was made;
3. Test Termination: If an exchange was made in the above loop, reset the ex-
   change flag to off and repeat the exchange loop.
```

Algorithm 9.4 Greedy-Exchange Algorithm.

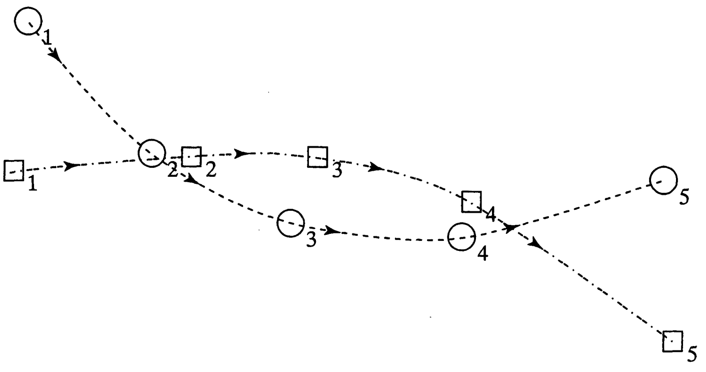


Figure 9.14 Trajectories of two objects, $\bigcirc$ and $\square$ are shown along with the image flow vector at each of the first five points. A tracker would consider $\square_2$ as a likely successor to $\bigcirc_1$ and $\bigcirc_5$ to be a possible ending of the sequence $\square_1, \square_2, \square_3, \square_4$.

TABLE 9.1 SMOOTHNESS FOR PATHS OF FIGURE 9.14

t = 1	t = 2	t = 3	t = 4	t = 5	smoothness
$\bigcirc_1$(112 262)	$\square_2$(206 185)	$\bigcirc_3$(250 137)			0.97
$\square_1$(106 175)	$\bigcirc_2$(180 188)	$\square_3$(280 185)			0.98
$\bigcirc_1$(112 262)	$\bigcirc_2$(180 188)	$\bigcirc_3$(250 137)			0.99
$\square_1$(106 175)	$\square_2$(206 185)	$\square_3$(280 185)			0.99
$\bigcirc_1$(112 262)	$\bigcirc_2$(180 188)	$\bigcirc_3$(250 137)	$\bigcirc_4$(360 137)		1.89
$\square_1$(106 175)	$\square_2$(206 185)	$\square_3$(280 185)	$\square_4$(365 156)		1.96
$\bigcirc_1$(112 262)	$\bigcirc_2$(180 188)	$\bigcirc_3$(250 137)	$\bigcirc_4$(360 137)	$\square_5$(482 80)	2.84
$\square_1$(106 175)	$\square_2$(206 185)	$\square_3$(280 185)	$\square_4$(365 156)	$\bigcirc_5$(478 170)	2.91
$\bigcirc_1$(112 262)	$\bigcirc_2$(180 188)	$\bigcirc_3$(250 137)	$\bigcirc_4$(360 137)	$\bigcirc_5$(478 170)	2.89
$\square_1$(106 175)	$\square_2$(206 185)	$\square_3$(280 185)	$\square_4$(365 156)	$\square_5$(482 80)	2.94

total smoothness for these two paths after looking ahead to the nearest neighbor assignments at time $t = 3$. From the first two rows of Table 9.1 we see that the total smoothness for these two paths is $0.97 + 0.98 = 1.95$. If the assignments $T[1,2] = \bigcirc_2$ and $T[2,2] = \square_2$ are exchanged, a better smoothness value of $0.99 + 0.99 = 1.98$ is achieved. After this exchange, the two trajectories up to time $t = 2$ are $(\square_1, \square_2)$ and $(\bigcirc_1, \bigcirc_2)$. Nearest point initial assignments will give the best smoothness values at times $t = 3, 4$ and no exchanges are needed. However, at time $t = 5$ the nearest point assignments yield trajectories $(\square_1, \square_2, \square_3, \square_4, \bigcirc_5)$ and $(\bigcirc_1, \bigcirc_2, \bigcirc_3, \bigcirc_4, \square_5)$. Computing total smoothness with these last two assignments exchanged improves total smoothness from $2.84 + 2.91 = 5.75$ to $2.89 + 2.94 = 5.83$ over the interior 3 trajectory points so the final result will be correct according to the labels in Figure 9.14.

Algorithm 9.4 initializes m complete paths of n points each before it applies any smoothness criteria. A variable number of exchange loops attempt to improve smoothness by exchanging points between two paths. If an improvement is made (and it is always

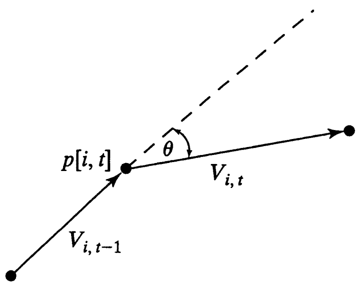


Figure 9.15 Vectors entering and leaving trajectory point $p[i, t]$.

the biggest improvement) by an exchange at any time t, then the entire exchange loop is repeated. In general, $\binom{m}{2}$ possible exchanges must be considered at each frame t. The algorithm requires at least $(n-2)\binom{m}{2}$ operations; more effort is required for any additional exchange loops. Total smoothness cannot increase beyond the value $1.0m(t-2)$ so the number of times that it can increase is limited and the algorithm must terminate.

Given that the assignments at frame $t = 1$ are arbitrary, there are m^{n-1} possible paths to consider overall—an exhorbitant number to evaluate. The Greedy-Exchange Algorithm does not consider exchanges of more than one pair of assignments at a time and hence might not obtain a global minimum. Algorithm 9.4 may be modified so that it is initialized using only one or three frames and continues as each new frame is sensed and processed for feature points. If all points of all frames are available, the algorithm can be improved by using both forward and backward processing in the exchange loop. Also, the algorithm has been extended to handle cases where points may appear or disappear between frames, primarily due to occlusions of one moving object by another. *Ghost points* can be used to represent points in frames containing fewer than m points.

Exercise 9.10

The following sets of three points were extracted from six frames of video and correspond to the data shown in Figure 9.13. Identify the smoothest set of three trajectories according to the Greedy-Exchange Algorithm.

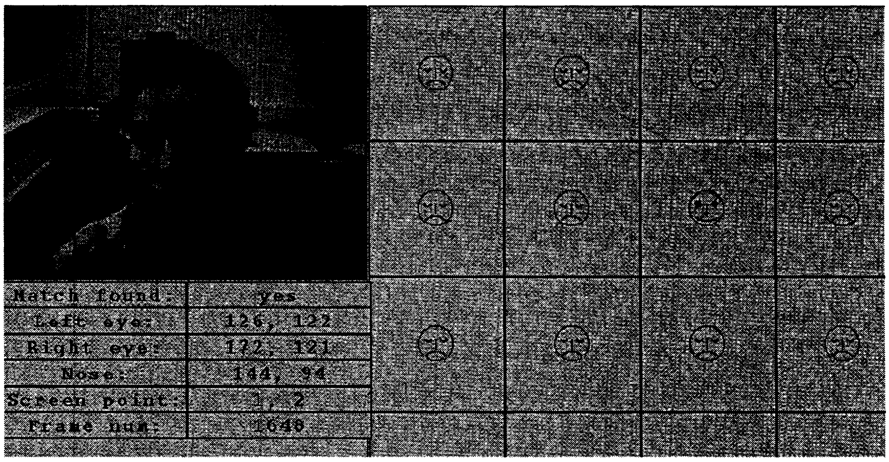
t = 1	t = 2	t = 3	t = 4	t = 5	t = 6
(483 270)	(155 152)	(237 137)	(292 128)	(383 117)	(475 220)
(107 225)	(420 237)	(242 156)	(358 125)	(437 156)	(108 108)
(110 133)	(160 175)	(370 180)	(310 145)	(234 112)	(462 75)

Exercise 9.11

Would you expect the Greedy-Exchange Algorithm to succeed in constructing trajectories from points in image sequences in the following cases? Explain why or why not. (a) The video is of a rotating carrousel with wooden horses that move up and down. (b) The video is of a street taken from the sidewalk: two cars pass just in front of the camera going 35 MPH in opposite directions. (c) High speed film is taken of the collison of two billiard balls. The moving white ball hits a still red ball: After the collision, the white ball is still and the red ball has taken all of its momentum.

9.4.1 Integrated Problem-Specific Tracking

Use of Algorithm 9.4 has demonstrated the power of using only the general constraints of smoothness. In specific applications much more information might be available to increase both the robustness and speed of tracking. If features are available for each of the m points, then feature matching can be included in the smoothness computations. Going even further, fitting of the current partial trajectory up to time t can be used to predict the location of the next trajectory point in frame $t+1$, which can greatly reduce the effort of cross-correlation used to find that point. Algorithms incorporating these methods can be found in the recent research literature. Maes and others (1996) have tracked human motion by computing the trajectories of the hands, feet, and head, which are identified as high-curvature protrusions from the silhouette of the moving human form. Bakic and Stockman (1999) track the human face, eyes, and nose using a workstation camera in order to move the mouse cursor. Figure 9.16 shows a sample screen exhibiting the features detected in the current frame and



Match found:	yes
Left eye:	126, 122
Right eye:	172, 121
Nose:	144, 94
Screen point:	1, 2
Frame num:	1648

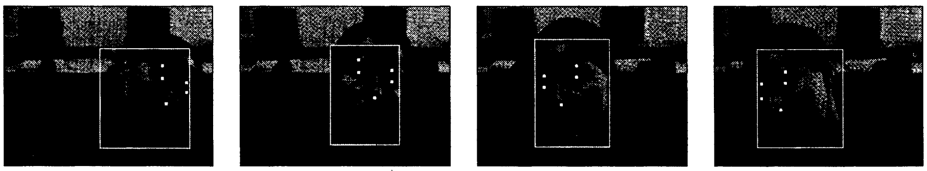


Figure 9.16 Tracking of the eyes and nose of a workstation user enables movement of the cursor without using the mouse. (top) Face pose determines selection in menu; (bottom) A sequence of face images showing tracking of the eyes and nose. (Images courtesy of Vera Bakic.)

the resulting position of the cursor in an array of 8×8 menu choices. The smiling face in the second row, third column indicates the user's selection. Processing can be done at 15 or more frames per second because of the integrated use of domain knowledge. The knowledge of face color is used to find the face in the image and knowledge of the structure of a face is used to locate the eyes and nose. Moreover, trajectories of the eyes and nose are used to predict where those features should be found in the next incoming frame: when they are found in the predicted neighborhood, global face detection is not done.

The making of the movie *Titanic* is one of many examples where integrated computer graphics and computer vision techniques were used to combine real imagery and synthetic imagery. Imagery was shot of a model ship and was later augmented by placing moving models on the deck of the ship. An actress was used to capture the motion of a live human wearing a flowing dress typical of the early twentieth century. Many small lights were attached to a real dress so that feature points could easily be detected in a motion sequence. The trajectories of these moving points were then used to orchestrate the motion of model people and model clothes, which were added to various positions in the images taken of the model ship. Many minutes of computer and operator time are used per frame of such a blockbuster movie, so not all steps need to be fully automatic.

9.5 DETECTING SIGNIFICANT CHANGES IN VIDEO

Video sequences may record minutes or hours of surveillance, different takes of a TV news crew, or a finished documentary or movie. It is becoming increasingly important to segment and store subsequences in digital libraries for random access. Some important concepts and methods related to parsing and analyzing video sequences are discussed in this section. First, we define some points of change found in videos and some other image sequences.

- A **scene change** is a change of environment; for example, from a restaurant scene to a street scene. Gross changes in the background are expected. Often, scene changes are made over ten to fifty frames using one of the camera effects below.
- A **shot change** is a significant change of camera view of the same scene. Often, this is accomplished by switching cameras. For example, in the restaurant scene camera one views actor A speaking and then frames are taken from camera two to view the response of the actor B across the table.
- A **camera pan** is used to sweep a horizontal view of the scene. If the camera pans from right to left, objects appear to enter at the left and move across the images to the right, finally exiting right. Motion vectors computed from consecutive frames of such a panning sequence of a static scene will be in the same direction from left to right.
- **Camera zoom** changes the focal length over time to expand the image of some part of the scene (zoom in) or to reduce the image of a scene part and include more adjacent background (zoom out).
- **Camera effects: fade, dissolve, and wipe** are used for transitions from one source of imagery to a different source of imagery. A *fade out* is a continuous transition from

one video source to black or to white frames, whereas a *fade in* is a transition from black or from white frames to some video source. A transition from video source A to video source B can be achieved by fading out A and then fading in B. A *dissolve* changes the pixels of A into pixels of B over several frames. One type of dissolve weights the pixels of A by $(1 - t/T)$ and the pixels of B by t/T over the frames $t = 0, \dots, T$. A *wipe* makes the transition from source A to B by changing the size of the regions where A and B appear in the frame. One can imagine a windshield wiper crossing our window with source A displayed on one side of the wiper and source B on the other. Wipes can be done using a vertical, horizontal, or diagonal boundary between the two regions. Or, source B can appear within a small circular region that grows to cover the entire frame.

Exercise 9.12

Construct a pseudo-code algorithm that blends video source A into video source B using a wipe. Source A is the image sequence $A_t[r, c]$ and source B is the image sequence $B_t[r, c]$. (a) Suppose the wipe is accomplished from time t_1 to time t_2 by using a diagonal line of slope 1 originating through pixel $[0, 0]$ (top left) at time t_1 and ending through pixel $[M-1, N-1]$ at time t_2. (b) Suppose the wipe is accomplished by a growing circular region at the frame center. At time t_1 the radius of the circle is 0 and at time t_2 the radius is large enough such that the circle circumscribes the entire frame.

9.5.1 Segmenting Video Sequences

The goal of the analysis is to parse a long sequence of video frames into subsequences representing single shots or scenes. As an example, consider a 30-minute video of a TV news program. There will be several 10 to 15 second segments where the camera shot is of a newscaster reporting from a desk: the background is a constant office background, but there may be zooming of the camera. After such a segment, it is common to transition to another source documenting another event, perhaps frames of a flood, an interesting play in sports, a meeting, or a government official jogging. Often there are several different shots of the event being reported with transitions between them. The transitions can be used to segment the video and can be detected by large changes in the features of the images over time.

One obvious method of computing the difference between two frames I_t and $I_{t+\Delta}$ of a sequence is to compute the average difference between corresponding pixels as in Equation 9.7. Depending on the camera effect being detected, the time interval Δ may be one or more frames.

$$d_{pixel}(I_t, I_{t+\Delta}) = \frac{\sum_{r=0}^{MaxRow-1} \sum_{c=0}^{MaxCol-1} | I_t[r, c] - I_{t+\Delta}[r, c] |}{MaxRow \times MaxCol} \tag{9.7}$$

Equation 9.7 is likely to mislead us by yielding a large difference when there is even a small amount of camera pan or some object motion in an otherwise stable shot. A more robust variation, proposed by Kasturi and Jain (1991), is to break the image into larger blocks and test to see if a majority of the blocks are essentially the same in both images. A likelihood ratio test defined in Equation 9.8 was proposed to evaluate whether or not there was significant change in the intensities of corresponding blocks. Let block B_1 in image I_1 have intensity mean and variance u_1 and v_1 and block B_2 in image I_2 have intensity mean and variance u_2 and v_2. The block difference is defined in terms of the likelihood ratio in Equation 9.8. If a sufficient number of the blocks have zero difference then the decision is that the two images are from essentially the same shot. Clearly, Equation 9.8 will be more stable than Equation 9.7 when the images are highly textured and are not stabilized from frame to frame to remove the effects of small motions of the camera.

$$\begin{aligned} r &= \frac{\left[\frac{v_1+v_2}{2} + \left(\frac{u_1-u_2}{2}\right)^2\right]^2}{v_1 v_2} \qquad (9.8) \\ d_{block}(B_1, B_2) &= 1 \quad \text{if } r > \tau_r \\ &= 0 \quad \text{if } r \leq \tau_r \\ d(I_1, I_2) &= \sum_{B_{1i} \in I_1; B_{2i} \in I_2} d_{block}(B_{1i}, B_{2i}) \end{aligned}$$

The difference between two images can be represented in terms of the difference between their histograms as was done in $d_{hist}(I, Q)$ of Chapter 8. In our current discussion, I is image I_1 and Q is image I_2. A 64-level histogram can provide enough levels. For color video frames, a value in the interval $[0, 63]$ can be obtained by concatenating the higher order two bits of the red, green, and blue color values. Histogram comparison can be faster than the previous methods and is potentially a better representative of the general features of the scene. Since it totally avoids any spatial coherence checking, it can be completely fooled when two images have similar histograms but totally different spatial distributions, and in fact are from two different shots.

Figure 9.17 shows four frames from the same documentary video. The top two frames occur prior to a scene break and the bottom two frames occur after the scene break. Figure 9.18 shows the histograms computed from the first three frames shown in Figure 9.17. The top two histograms are similar: this means that the two frames from which they were derived are likely to be from the same shot. The bottom histogram differs significantly from the first two, and thus the third frame from Figure 9.17 is likely to be from a different shot.

9.5.2 Ignoring Certain Camera Effects

We do not want to segment the video sequence when adjacent frames differ significantly due only to certain camera effects such as pan or zoom. Transitions detected as in Section 9.5.1 can be subject to some simple motion analysis to determine if they are effects to be ignored. Panning can be detected by computing motion vectors and determining if the motion vectors closely cluster around some modal direction and magnitude. We can do this by simple analysis of the set **V** output from Algorithm 9.3. Zooming can be detected by examining the motion vectors at the periphery of the motion field. Zooming in or out is indicated by the

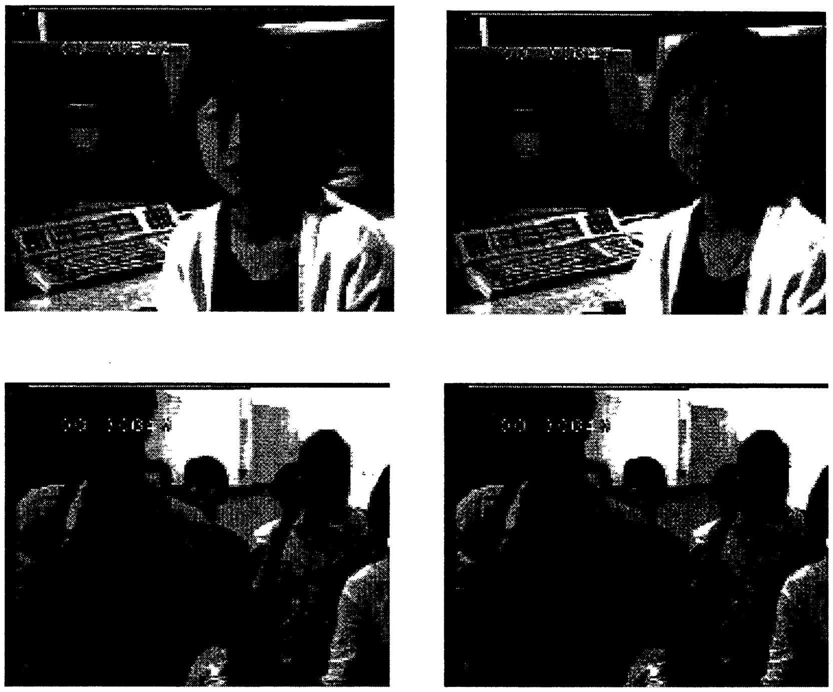


Figure 9.17 Four frames from a documentary video. The top two and the bottom two are separated across a camera break. (Reprinted from Zhang and others (1993) with permission from Springer-Verlag.)

motion vectors at the periphery approximately summing to zero. Using only the periphery of the motion field accounts for cases where the FOE or FOC is not near the center of the field. Suppose that motion vectors are computed using MPEG-type block matching techniques so that there are motion vectors for blocks along the top and bottom of the motion field determined by I_1 and I_2. The difference between the vertical components of motion vectors in corresponding positions across the image should be greater than that of either the top or bottom motion vector as shown in Figure 9.19. The relationship is similar for the horizontal components of motion vectors horizontally related across the image. Both zoom in and out can be detected reasonably well using these heuristics. The quality of the motion field derived from block matching will deteriorate, however, due to change of scale as the zooming speeds up.

Exercise 9.13

Obtain two consecutive frames of a video of the same scene. (a) Compute the average pixel difference as defined in Equation 9.7. (b) Partition the image into $2 \times 2 = 4$ blocks and compute the sum of their block differences as defined in Equation 9.8.

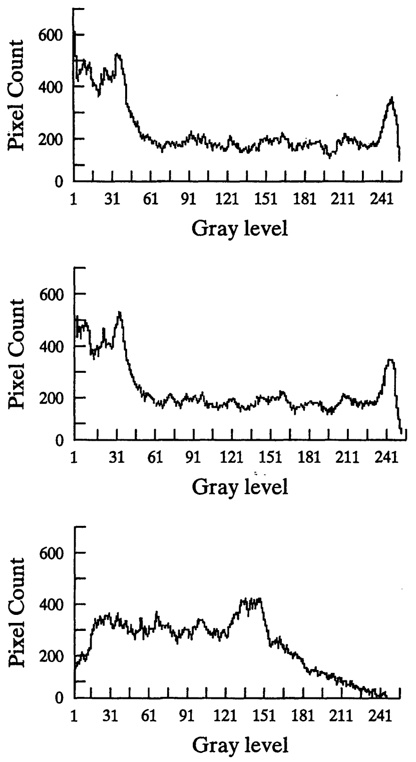


Figure 9.18 Histograms from the first three frames shown in Figure 9.17. The top two are similar as are the frames from which they were derived. The bottom histogram is significantly different from the top two, indicating that the frame from which is was derived is different from the first two. (Reprinted from Zhang and others (1993) with permission from Springer-Verlag.)

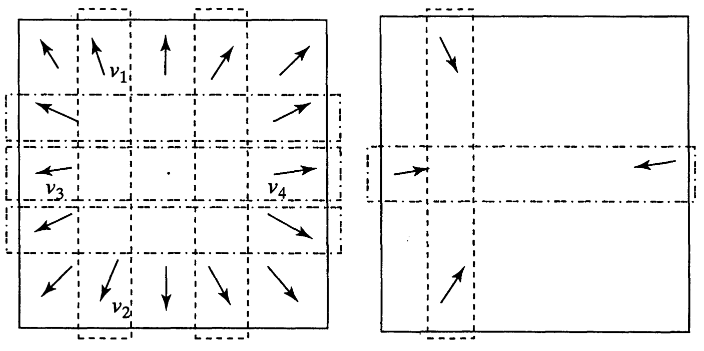


Figure 9.19 Heuristics for detection of camera zoom by comparing motion vectors across the periphery of the motion field. The difference of vertical components across the field exceeds the vertical component of either vector: $|v_{1r} - v_{2r}| > max\{|v_{1r}|, |v_{2r}|\}$. Similarly, $|v_{3c} - v_{4c}| > max\{|v_{3c}|, |v_{4c}|\}$ for horizontally opposed motion vectors. These relations hold for both zoom in (left) and zoom out (right).

9.5.3 Storing Video Subsequences

Once a long video sequence is parsed into meaningful subsequences, these subsequences can be stored in a video database for query and retrieval. They can be accessed using some of the same methods discussed in Chapter 8. Certain *key frames* can be identified and used for access in the same manner as in Chapter 8. In the future, we will probably be able to go much farther; for example, automatic motion analysis might be performed to assign symbolic action labels such as *running, fighting,* or *debating*. Face recognition might be performed in order to label famous people, or general object recognition might provide labels such as *horse* or *house*. Although the use of many frames implies more computational effort with video compared to still images, the information provided by motion analysis should increase both the cability to segment objects from background and the capability to classify them.

Exercise 9.14

Consider the application, discussed earlier in this chapter, of computing an analysis of a tennis match from a video of the match. (a) What are the actions or events that an analysis program should report? (b) What quantitative data should be reported?

9.6 REFERENCES

The discussion of tracking the players and ball in a tennis match is based upon the recent work of Pingali, Jean, and Carlbom (1998) at Bell Laboratories. The paper by Freeman and others (1998) describes the results of several experiments that have used computer vision techniques to create a gesture interface for interacting with several existing applications. Included is a description of a fast motion estimation algorithm. The details of the fast motion estimation algorithm, and much detail about the game interface are given in Kage and others (1999). Our treatment of video parsing and indexing followed the work of Zhang and others (1993) and Smolier and Zhang (1996). The treatment of computing smooth trajectories from many frames of featureless points was based on the work of Sethi and Jain (1987), which was done in an era of computer vision where it was common to study what could be computed from a small set of general assumptions. More recent work, such as that reported by Maes and others (1996), Darrell and others (1998) and Bakic and Stockman (1999) integrates problem-specific knowledge in order to speed up the process and make it more robust. The work by Ayers and Shah (1998) shows how to interpret motion and change in terms of the semantics relevant to a surveillance application.

1. Ayers, D., and M. Shah. 1998. Recognizing human actions in a static room. *Proc. 4th IEEE Workshop on Applications of Computer Vision,* Princeton, NJ (19–21 Oct. 1998), 42–47.
2. Bakic, V., and G. Stockman. 1999. Menu selection by facial aspect. *Proc. Vision Interface '99,* Quebec, Canada (18–21 May 1999),18–21.

3. Darrell, T. 1998. A radial cumulative similarity transform for robust image correspondence. *Proc. IEEE CVPR,* Santa Barbara, CA (June 1998), 656–662.
4. Darrell, T., G. Gordon, M. Harville, and J. Woodfill. 1998. Integrated person tracking using stereo, color, and pattern detection. *Proc. IEEE CVPR,* Santa Barbara, CA (June 1998), 601–608.
5. Freeman, W., D. Anderson, P. Beardsley, C. Dodge, M. Roth, C. Weissman, W. Yerazunis, H. Kage, K. Kyuma, Y. Miyake, and K. Tanaka. 1998. Computer vision for interactive computer graphics. *IEEE Comput. Graphics and Applications,* v. 18(3) (May–June 1998), 42–53.
6. Horn, B., and B. Schunck. 1981. Determining optical flow. *Artificial Intelligence,* v. 17:185–203.
7. Johansson, G. 1964. Perception of motion and changing form. *Scandanavian J. Psychology,* v. 5:181–208.
8. Kage, H., W. T. Freeman, Y. Miyake, E. Funatsu, K. Tanaka, and K. Kyuma. 1999. Artificial retina chips as on-chip image processors and gesture-oriented interfaces. *Optical Engineering,* 38(12):1979–1988.
9. Kasturi, R., and R. Jain. 1991. Dynamic vision. In *Computer Vision Principles,* R. Kasturi and R. Jain, eds. IEEE Computer Society Press, Washington, D.C., 469–480.
10. Maes, P., T. Darrell, B. Blumberg, and A. Pentland, 1996, *The ALIVE System: Wireless, Full-Body, Interaction with Autonomous Agents.* ACM Multimedia Systems: Special Issue on Multimedia and Multisensory Virtual Worlds, Sprint.
11. Pingali, G., Y. Jean, and I. Carlbom. 1998. Real time tracking for enhanced tennis broadcasts. *Proc. IEEE CVPR,* Santa Barbara, CA (June 1998), 260–265.
12. Salari, V., and I. Sethi. 1990. Correspondence of feature points in presence of occlusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence,* v. 12(1):87–91.
13. Sethi, I., and R. Jain. 1987. Finding trajectories of feature points in a monocular image sequence. *IEEE Trans. on Pattern Analysis and Machine Intelligence,* v. 9(1):56–73.
14. Smolier, S., and H-J Zhang. 1996. Video indexing and retrieval. In *Multimedia Systems and Techniques,* B. Furht, ed. Kluwer Academic Publishers, Boston, 293–322.
15. Zhang, H-J., A. Kankanhalli, and S. Smoliar. 1993. Automatic partitioning of full-motion video. *Multimedia Systems,* v. 1(1):10–28.