FUNDAMENTALS OF COMPUTER VISION[1]

Mubarak Shah
Computer Science Department
University of Central Florida
Orlando, FL 32816

December 7, 1997

Contents

Chapter 1

Imaging Geometry

1.1 Introduction

In[1] the first chapter of this book we will learn about imaging geometry. We will discuss various transformations which will be used in many problems in Computer Vision. These transformations are also used in Computer Graphics. We will learn about coordinate systems and how to relate one coordinate system with another coordinate system. We will discuss translation, rotation, scaling, perspective transformation, camera modeling, and camera calibration. These transformations will help us to know the location and orientation of the object after it is translated by some amount, rotated around a given axis, and scaled by some factor. We will derive a matrix for each of these transformations.

The world we live in is three dimensional, that is any point in space can be specified by three coordinates (X, Y, Z). The image is a 2-D plane, so we need two coordinates (x, y) to represent a point in the image. One dimension is lost in the projection process. One of the important goals in Computer Vision is to recover this lost dimension. The methods for recovering 3-D information from 2-D images are called *shape from 'X'*, where 'X' can be stereo, motion, shading, texture, etc. We will discuss these methods in great detail in this book. However, in this chapter we will focus on the derivation of the camera matrix and camera calibration.

1.2 Translation and Scaling

Consider a point on an object with coordinates $(X1, Y1, Z1)$. Assume that the object is translated by dx, dy, and dz respectively in the X, Y, and Z directions. The new coordinates of the point are given by:

$$X2 = X1 + dx \tag{1.1}$$
$$Y2 = Y1 + dy \tag{1.2}$$
$$Z2 = Z1 + dz \tag{1.3}$$

These equations can be written in matrix form as follows:

$$\begin{bmatrix} X2 \\ Y2 \\ Z2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X1 \\ Y1 \\ Z1 \\ 1 \end{bmatrix} \tag{1.4}$$

$$\begin{bmatrix} X2 \\ Y2 \\ Z2 \\ 1 \end{bmatrix} = T \begin{bmatrix} X1 \\ Y1 \\ Z1 \\ 1 \end{bmatrix} \tag{1.5}$$

where $T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$ is called translation matrix. The inverse translation matrix is given by $T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$. You can verify that $TT^{-1} = T^{-1}T = I$, where I is the identity matrix.

Similarly if the object is scaled by Sx, Sy, and Sz respectively in the X, Y, and Z directions, the new coordinates of the point are given by:

$$X2 = X1 \times Sx \tag{1.6}$$

$$Y2 = Y1 \times Sy \tag{1.7}$$

$$Z2 = Z1 \times Sz \tag{1.8}$$

These equations can be written in matrix form as follows:

$$\begin{bmatrix} X2 \\ Y2 \\ Z2 \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X1 \\ Y1 \\ Z1 \\ 1 \end{bmatrix}, \tag{1.9}$$

$$\begin{bmatrix} X2 \\ Y2 \\ Z2 \\ 1 \end{bmatrix} = S \begin{bmatrix} X1 \\ Y1 \\ Z1 \\ 1 \end{bmatrix}, \tag{1.10}$$

$$\tag{1.11}$$

where $S = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ is called the scaling matrix. The inverse of scaling matrix is given by $S^{-1} = \begin{bmatrix} 1/Sx & 0 & 0 & 0 \\ 0 & 1/Sy & 0 & 0 \\ 0 & 0 & 1/Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$.

1.3 Rotation

Consider a vector whose endpoint is at $(X1, Y1, Z1)$, as shown in Figure 1.1.a. Let R denote the length of vector, and ϕ denote the angle the vector makes with $X-$axis. Assume that the vector is rotated by the angle θ around the $Z-$axis in the counterclockwise direction. The new coordinates $(X2, Y2, Z2)$ of the point after rotation can be computed from the old coordinates and the angle of rotation. Consider a triangle in Figure 1.1.a whose side makes an angle ϕ with the $X-$axis. Using the standard trigonometric relations we can write:

$$X1 = R\cos\phi \tag{1.12}$$
$$Y1 = R\sin\phi. \tag{1.13}$$

Similarly, for the triangle in Figure 1.1.a whose side makes an angle $\theta+\phi$ with the $X-$axis we can write:

$$X2 = R\cos(\theta+\phi) = R\cos\theta\cos\phi - R\sin\theta\sin\phi \tag{1.14}$$
$$Y2 = R\sin(\theta+\phi) = R\sin\theta\cos\phi + R\cos\theta\sin\phi. \tag{1.15}$$

Substituting first two equations into the last two equations, we get:

$$X2 = X1\cos\theta - Y1\sin\theta \tag{1.16}$$
$$Y2 = X1\sin\theta + Y1\cos\theta. \tag{1.17}$$

These equations can also be written in matrix form as follows:

$$\begin{bmatrix} X2 \\ Y2 \\ Z2 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X1 \\ Y1 \\ Z1 \\ 1 \end{bmatrix} \tag{1.18}$$

$$\begin{bmatrix} X2 \\ Y2 \\ Z2 \\ 1 \end{bmatrix} = R^Z_\theta \begin{bmatrix} X1 \\ Y1 \\ Z1 \\ 1 \end{bmatrix} \tag{1.19}$$

where $R^Z_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ is the rotation matrix. We will use a superscript to denote the axis of rotation, a subscript to denote the angle, and we will assume rotation in the counterclockwise direction. The inverse rotation matrix is given by $(R^Z_\theta)^{-1} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$. The inverse rotation by θ is equivalent to the rotation by $-\theta$ around the same axis. Therefore, $(R^Z_\theta)^{-1} = R^Z_{-\theta}$. Moreover, the rotation matrix is an orthonormal matrix, that is, $(R^Z_\theta)^T R^Z_\theta = R^Z_\theta (R^Z_\theta)^T = I$. Therefore, the inverse of a rotation matrix is just its transpose.

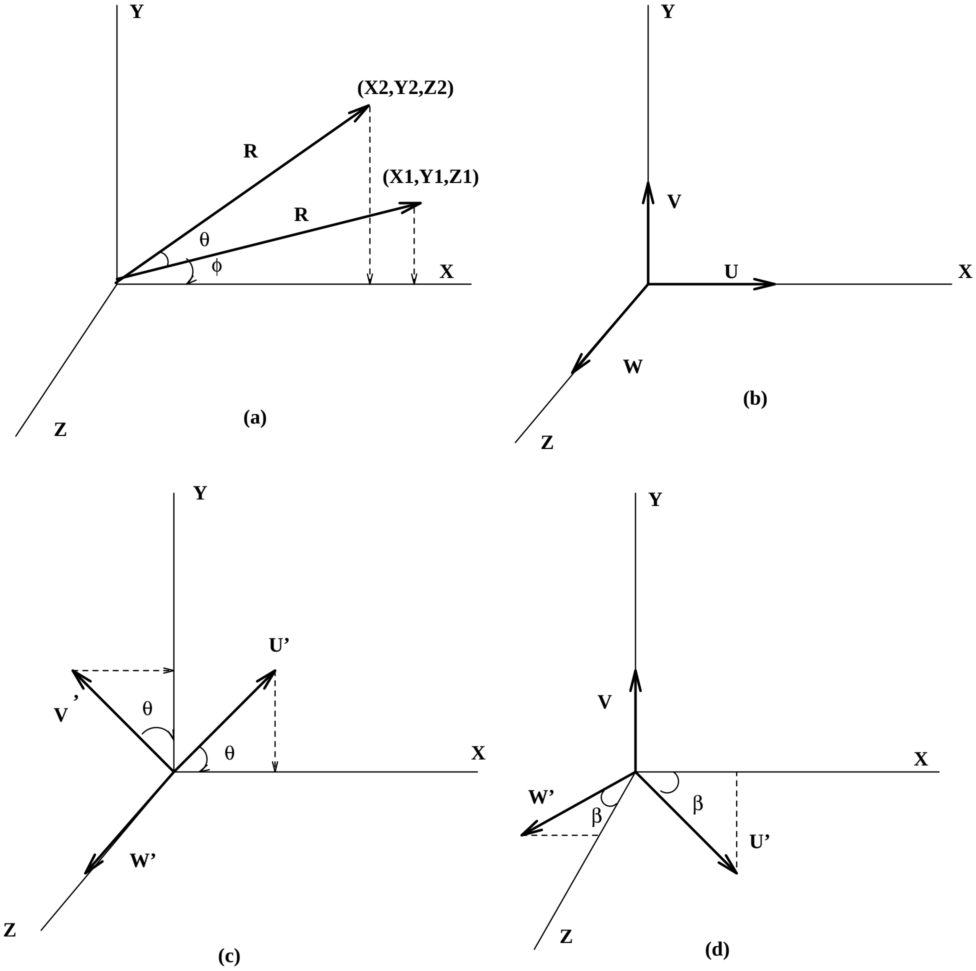


Figure 1.1: Rotation. (a) Rotation around Z axis. (b) Two coordinate systems, UVW superposed on XYZ. (c) UVW rotated by θ around Z axis in the counterclockwise direction. (d)UVW rotated by β around Y axis in the clockwise direction.

An alternate way to derive the rotation matrix is to consider two coordinate systems (X, Y, Z) and (U, V, W) as shown in Figure 1.1.b. Assume that U, V, W are the unit vectors. Let us rotate the UVW coordinate system around the Z-axis by an angle θ as shown in Figure 1.1.c. Note that the $W-$axis remains fixed with the $Z-$axis. However, the $U-$ and V-axes are changed. The new coordinates U', V', and W' can be written in terms of the old coordinate system. The U'-vector has two components, one along $X-$axis, and one along Y-axis. Note that it does not have a Z component. The new U' vector is given by $(\cos\theta, \sin\theta, 0)$. Similarly, the V'-vector is given by $(-\sin\theta, \cos\theta, 0)$. Since the W vector did not change, it remains as $(0, 0, 1)$. The rotation matrix can now be formed by putting the coordinates of the U', V', and W' vectors respectively in the first, second and third columns, and $(0, 0, 0, 1)$ in the fourth column. The resultant matrix is exactly the same as the matrix R^Z_θ.

This technique can be applied to derive the rotation matrix around any axis. For instance, the rotation matrix around the $Y-$axis in the clockwise direction is given by (see Figure 1.1.d):

$$R^Y_{-\beta} = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

1.4 Perspective Transform

A simplified model of image formation is shown in Figure 1.2.a. In this so called pinhole camera model, the lens is assumed to be a single point. The world coordinates (X, Y, Z) of a point are transformed to the image coordinates (x, y) under perspective projection. Assume that the origin of the coordinate system is at the lens center (shown by O in Figure 1.2.a), and f is the focal length of the camera, the distance from the lens to the image plane. Then using the two equivalent triangles we can write:

$$\frac{-y}{Y} = \frac{f}{Z}. \tag{1.20}$$

Since the image is always formed upside down, it is customary to use a negative sign in front of the y in the above equation. From the above equation, the y image coordinates are given by:

$$y = -\frac{fY}{Z}. \tag{1.21}$$

Similarly, the $x-$coordinates of the image are given by:

$$x = -\frac{fX}{Z}. \tag{1.22}$$

The above equations represent the perspective transform with the origin at the lens. If the origin is moved to the image, as shown in Figure 1.2.b, the perspective projection is defined by the following equations:

$$x = \frac{fX}{f - Z} \tag{1.23}$$

$$y = \frac{fY}{f - Z}. \tag{1.24}$$

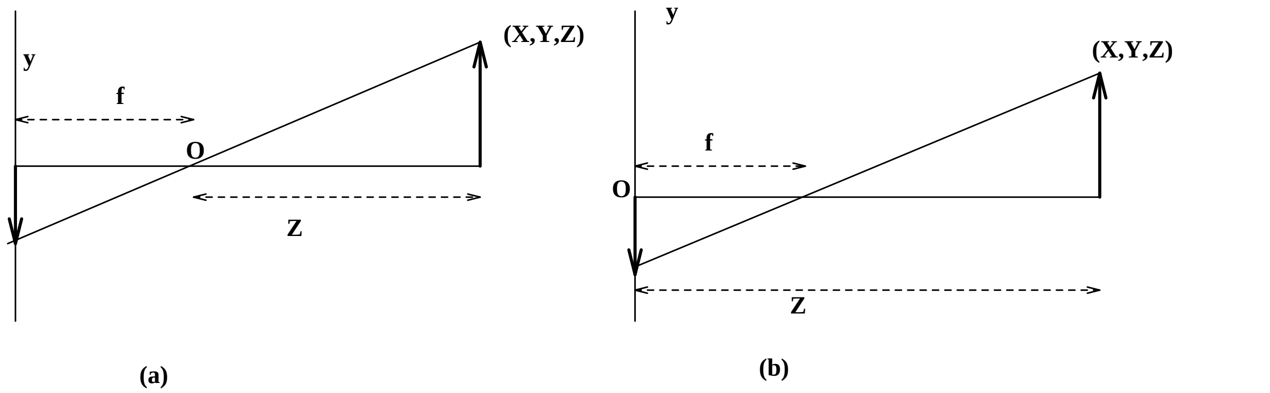


Figure 1.2: Pinhole camera model. (a) Origin at the lens. (b) Origin at the image plane.

It is not possible to derive a perspective matrix similar to the translation, scaling, and rotation matrices due to the nature of equations 1.23 and 1.24. We need to introduce another transformation called *homogeneous transformation*. The homogeneous transformation converts the *Cartesian world coordinates* (X, Y, Z) into the *homogeneous world coordinates* (kX, kY, kZ, k). In this transformation each X, Y, Z coordinate is multiplied by a constant k, and k is appended as the fourth component of the vector. Similarly, the inverse homogeneous transform converts the homogeneous image coordinates $(C_{h_1}, C_{h_2}, C_{h_3}, C_{h_4})$ into the Cartesian image coordinates $(\frac{C_{h_1}}{C_{h_4}}, \frac{C_{h_2}}{C_{h_4}}, \frac{C_{h_3}}{C_{h_4}})$, by dividing each of the three components with the fourth component.

The perspective matrix can now be defined as:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{f} & 1 \end{bmatrix}.$$

The perspective transform, which relates the homogeneous world coordinates with the homogeneous image coordinates is defined as:

$$\begin{bmatrix} kX \\ kY \\ kZ \\ -\frac{kZ}{f} + k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{f} & 1 \end{bmatrix} \begin{bmatrix} kX \\ kY \\ kZ \\ k \end{bmatrix}$$

We can easily derive the Cartesian image coordinates from the above equation as:

$$x = \frac{fX}{f - Z}, \tag{1.25}$$

$$y = \frac{fY}{f - Z}, \tag{1.26}$$

which are exactly the same equations as equations 1.23 and 1.24. The inverse perspective matrix is given by: $P^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{f} & 1 \end{bmatrix}$.

1.5 Camera Model

The perspective transform relates the world coordinates with the image coordinates when the camera is located at the origin of the world coordinates. However, in real life we need to translate and rotate the camera in order to bring the object of interest in the field of view. Therefore, the complete camera model involves several other transformations besides the perspective transform.

In one simple model, assume that the camera is first at the origin of the world coordinates, is then translated by (X_0, Y_0, Z_0) (matrix G), then rotated around the Z-axis by θ in the counterclockwise direction (matrix $R^Z_{-\theta}$), rotated again around the X-axis by ϕ in the counterclockwise direction(matrix $R^X_{-\phi}$), and then translated by (r_1, r_2, r_3) (matrix C). In this case, the world homogeneous coordinates (W_h) are related to the camera image coordinates (C_h) as follows:

$$C_h = PCR^X_{-\phi}R^Z_{-\theta}GW_h \tag{1.27}$$

where $P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{f} & 1 \end{bmatrix}$, $R^Z_{-\theta} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$,

$$R^X_{-\phi} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & \sin\phi & 0 \\ 0 & -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, G = \begin{bmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & -Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, C = \begin{bmatrix} 1 & 0 & 0 & -r_1 \\ 0 & 1 & 0 & -r_2 \\ 0 & 0 & 1 & -r_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that the inverse transforms for the translation and rotations are used here. This is because it is the coordinate system attached to the camera that is being translated and rotated, rather than the object. Now, the Cartesian image coordinates can be computed from equations:

$$x = f\frac{(X - X_0)\cos\theta + (Y - Y_0)\sin\theta - r_1}{-(X - X_0)\sin\theta\sin\phi + (Y - Y_0)\cos\theta\sin\phi - (Z - Z_0)\cos\phi + r_3 + f}, \tag{1.28}$$

$$y = f\frac{-(X - X_0)\sin\theta\cos\phi + (Y - Y_0)\cos\theta\cos\phi + (Z - Z_0)\sin\phi - r_2}{-(X - X_0)\sin\theta\sin\phi + (Y - Y_0)\cos\theta\sin\phi - (Z - Z_0)\cos\phi + r_3 + f}. \tag{1.29}$$

1.6 Camera Calibration

In the previous section we discussed the camera model, which relates the world coordinates with the image coordinates. It was assumed that the camera focal length, rotation angles, and translation displacements are known. An alternative method for computing the camera model is to use the camera as a measuring device to determine the unknowns in the camera model. This process is called camera calibration. In this method, a few points in 3-D with known coordinates and their corresponding image coordinates are used to calibrate the camera.

The projection of 3-D homogeneous coordinates onto the image plane (from equation 1.27) can be summarized as:

$$C_h = AW_h, \tag{1.30}$$

$$\begin{bmatrix} C_{h_1} \\ C_{h_2} \\ C_{h_3} \\ C_{h_4} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{1.31}$$

where C_h and W_h respectively denote the camera and world homogeneous coordinates, and $A = PCR^X_{-\phi}R^Z_{-\theta}G$, denote the camera matrix. The aim in the camera calibration is to determine the matrix A using the known 3-D points and their corresponding image coordinates. The matrix A is 4×4, which has 16 unknowns. However, C_{h_3} is not meaningful in the image coordinates. Because the image is only two dimensional, the image coordinates can be determined by C_{h_1}, C_{h_2}, C_{h_4}. Therefore, we do not need to determine the third row of the matrix, which is used for C_{h_3}. Now, we are only left with 12 unknowns in the matrix A. We will find out later in this section that we can arbitrary fix one of the 12 unknowns and determine the remaining 11 unknowns.

As we know, the Cartesian image coordinates (x, y) are given by

$$x = \frac{C_{h_1}}{C_{h_4}}, \tag{1.32}$$

$$y = \frac{C_{h_2}}{C_{h_4}}. \tag{1.33}$$

From the equations 1.31-1.33 we get:

$$C_{h_1} = a_{11}X + a_{12}Y + a_{13}Z + a_{14} = C_{h_4}x \tag{1.34}$$

$$C_{h_2} = a_{21}X + a_{22}Y + a_{23}Z + a_{24} = C_{h_4}y \tag{1.35}$$

$$C_{h_4} = a_{41}X + a_{42}Y + a_{43}Z + a_{44} \tag{1.36}$$

substituting C_{h_1}, C_{h_2}, and C_{h_4} in equations 1.32-1.33 and rearranging we get:

$$a_{11}X + a_{12}Y + a_{13}Z + a_{14} - a_{41}Xx - xa_{42}Y - xa_{43}Z - xa_{44} = 0 \tag{1.37}$$

$$a_{21}X + a_{22}Y + a_{23}Z + a_{24} - a_{41}Xy - ya_{42}Y - ya_{43}Z - ya_{44} = 0. \tag{1.38}$$

In the above equations there are 12 unknowns $(a_{11}, \ldots\ldots, a_{44})$, and five knowns (X, Y, Z, x, y). One point in 3-D with coordinates (X, Y, Z), and corresponding image coordinates (x, y) gives two such equations with 12 unknowns. n such points will give $2n$ equations, which can be solved for 12 unknowns.

$$a_{11}X_1 + a_{12}Y_1 + a_{13}Z_1 + a_{14} - x_1a_{41}X_1 - x_1a_{42}Y_1 - x_1a_{43}Z_1 - x_1a_{44} = 0 \tag{1.39}$$

$$a_{11}X_2 + a_{12}Y_2 + a_{13}Z_2 + a_{14} - x_2a_{41}X_2 - x_2a_{42}Y_2 - x_2a_{43}Z_2 - x_2a_{44} = 0 \tag{1.40}$$

$$\vdots$$

$$a_{11}X_n + a_{12}Y_n + a_{13}Z_n + a_{14} - x_na_{41}X_n - x_na_{42}Y_n - x_na_{43}Z_n - x_na_{44} = 0 \tag{1.41}$$

$$a_{21}X_1 + a_{22}Y_1 + a_{23}Z_1 + a_{24} - y_1a_{41}X_1 - y_1a_{42}Y_1 - y_1a_{43}Z_1 - y_1a_{44} = 0 \tag{1.42}$$

$$a_{21}X_2 + a_{22}Y_2 + a_{23}Z_2 + a_{24} - y_2a_{41}X_2 - y_2a_{42}Y_2 - y_2a_{43}Z_2 - y_2a_{44} = 0 \tag{1.43}$$

$$\vdots$$

$$a_{21}X_n + a_{22}Y_n + a_{23}Z_n + a_{24} - y_na_{41}X_n - y_na_{42}Y_n - y_na_{43}Z_n - y_na_{44} = 0 \tag{1.44}$$

In these equations the subscripts on (X, Y, Z) and (x, y) denote the point number. The above system can be written in matrix form as follows:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -x_2X_2 & -x_2Y_2 & -x_2Z_2 & -x_2 \\ \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -x_nX_n & -x_nY_n & -x_nZ_n & -x_n \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -y_2X_2 & -y_2Y_2 & -y_2Z_2 & -y_2 \\ \vdots \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -y_nX_n & -y_nY_n & -y_nZ_n & -y_n \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{41} \\ a_{42} \\ a_{43} \\ a_{44} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{1.45}$$

or

$$CP = 0, \tag{1.46}$$

where C is a $2n \times 12$ matrix, P a is 12×1 vector, and 0 is also a 12×1 vector. This is a homogeneous system which has multiple solutions. Therefore, we can arbitrarily pick one of the unknowns, and determine the remaining unknowns. Let $a_{44} = 1$, then the above system can be rewritten as:

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -X_1x_1 & -x_1Y_1 & -x_1Z_1 \\ X_2 & Y_2 & Z_2 & 1 & 0 & 0 & 0 & 0 & -X_2x_2 & -x_2Y_2 & -x_2Z_2 \\ \vdots \\ X_n & Y_n & Z_n & 1 & 0 & 0 & 0 & 0 & -X_nx_n & -x_nY_n & -x_nZ_n \\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -X_1y_1 & -y_1Y_1 & -y_1Z_1 \\ 0 & 0 & 0 & 0 & X_2 & Y_2 & Z_2 & 1 & -X_2y_2 & -y_2Y_2 & -y_2Z_2 \\ \vdots \\ 0 & 0 & 0 & 0 & X_n & Y_n & Z_n & 1 & -X_ny_n & -y_nY_n & -y_nZ_n \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{41} \\ a_{42} \\ a_{43} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{1.47}$$

$$DQ = R \tag{1.48}$$

In the above system, the matrix D and vector R are known. We can determine the 11 unknowns in vector Q by using the pseudo inverse, if at least 5.5 or more 3-D points and their corresponding image coordinates are known. Multiplying the above equation by D^T on both sides and rearranging we get:

$$D^TDQ = D^TR \tag{1.49}$$

$$Q = (D^TD)^{-1}D^TR. \tag{1.50}$$

Once Q is determined, the matrix A is defined, which completes the camera calibration.

1.7 Recovering Camera Parameters

An interesting inverse problem deals with the recovery of camera parameters from the camera matrix. For example, given a photograph taken by an unknown camera from an unknown location which, has been cropped and/or enlarged, how can we determine the camera's position and orientation, and determine the extent to which the picture was cropped or enlarged? Applications of this occur in several areas. For instance, in autonomous navigation, a cruise missile could obtain the camera transformation matrix from a terrain model stored on-board and then compute the camera parameters that define the vehicle's location and heading. Also, a stationary camera viewing a robot arm workspace could determine the position of the arm. The markings of several points on part of the manipulator would allow their easy extraction from an image and provide ground truth for the camera calibration. The camera parameters can then be derived from the camera transformation matrix, and the location and orientation of the manipulator can be obtained relative to the stationary camera.

1.7.1 Camera Location

Consider a 3-D point **X1** with coordinates $(X1, Y1, Z1, 1)$ (see Figure 1.4). If camera matrix A is known we can determine the image coordinates of a point by using equation 1.30, that is $\mathbf{U1} = A\mathbf{X1}$, where $\mathbf{U1} = (C_{h_1}, C_{h_2}, C_{h_3}, C_{h_4})$. If we multiply **U1** with A^{-1}, we should get the original **X1** back. Let us set $C_{h_3} = 0$, then $\mathbf{U1}' = (C_{h_1}, C_{h_2}, 0, C_{h_4})$. Now, backproject $\mathbf{U1}'$ by $A^{-1}\mathbf{U1}' = \mathbf{X11}$. We will get a new 3-D point **X11**, which should lie on the same line connecting **X1** and the center of projection L (see Figure 1.4). Similarly, we can take another 3-D point $\mathbf{X2} = (X2, Y2, Z2, 1)$, and generate **X21**, which will lie on the line passing through **X2** and the center of projection. Now, we can easily determine L by the intersection of these two lines.

1.7.2 Camera Orientation

The orientation of a camera is defined as the orientation of the image plane. It is clear from Figure 1.5 that as the object moves closer to the lens its image moves farther away from the image center along the $Y-$axis. When the object is at the lens, the image will be formed at infinity. The only way the image of a finite world point can be formed at infinity is if the fourth component of the homogeneous image coordinates is zero. From equation 1.30 we have

$$\begin{bmatrix} C_{h_1} \\ C_{h_2} \\ C_{h_3} \\ 0 \end{bmatrix} = A \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{1.51}$$

which gives the following constraint on the camera orientation:

$$a_{41}X + a_{42}Y + a_{43}Z + a_{44} = 0 \tag{1.52}$$

This is the equation of a plane passing through lens L parallel to the image plane. From this equation it is clear that (a_{41}, a_{42}, a_{43}) is the normal to the image plane, and parallel to the camera.

The orientation in terms of clockwise rotation, θ, around the $X-$axis, followed by the clockwise rotation, ϕ, around the $Y-$axis(as shown in figure 1.3), is given by:

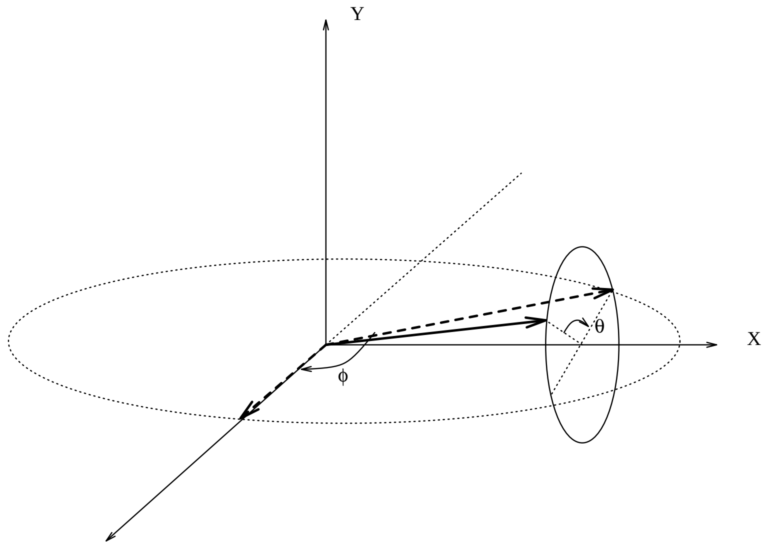


Figure 1.3: Camera Orientation.

$$\theta = \arctan \frac{a_{42}}{-a_{43}}, \tag{1.53}$$

$$\phi = \arcsin \frac{-a_{41}}{\sqrt{{a_{41}}^2 + {a_{42}}^2 + {a_{43}}^2}}. \tag{1.54}$$

It is also possible to compute the focal length, scaling and other camera parameters from the camera matrix. However, the derivations are fairly involved, therefore we do not cover them in this introductory book. The interested reader should look at [23] in the bibliography for the details.

1.8 Rodrigue's Formula

Assume that we want to rotate vector v around vector n by angle θ. We can decompose vector v into two parts, one collinear to n, and other perpendicular to n as follows, (see Figure 1.6):

$$v = (v.n)n + (v - (v.n)n). \tag{1.55}$$

The first component is invariant under rotation, whereas the second component, undergoes a planar rotation through angle θ which may be written:

$$\cos\theta(v - (v.n)n) + \sin\theta(n \times (v - (v.n)n)). \tag{1.56}$$

This gives:

$$v' = \cos\theta\ v + \sin\theta\ n \times v + (1 - \cos\theta)(v.n)n, \tag{1.57}$$

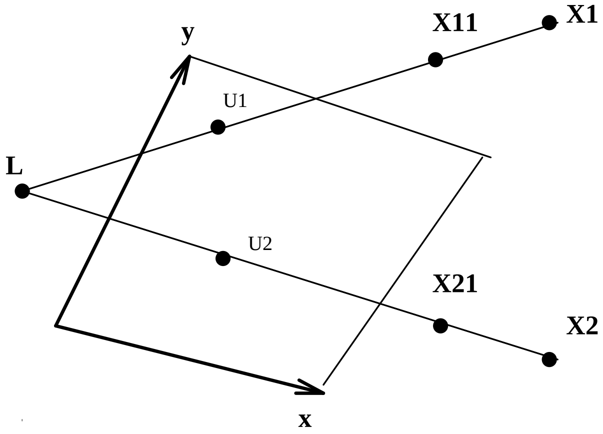


Figure 1.4: Determining the location of camera. The image of 3-D points $X1$ and $X11$ is formed at $U1$, and image of points $X2$ and $X21$ is formed at $U2$. The intersection of a line passing through $X1$ and $X11$ with the line passing through $X2$ and $X21$ is L, which is the location of camera.

or

$$v' = v + \sin\theta \; n \times v + (1 - \cos\theta) n \times (n \times v), \tag{1.58}$$

where $n \times (n \times v) = (v.n)n - v$. Now, from the above equation the rotation matrix R^n_θ can be written as:

$$R^n_\theta = I + \sin\theta \; X(n) + (1 - \cos\theta) X(n)^2, \tag{1.59}$$

where $X(n)$ is operator which is the vector product by n, that is the antisymmetric matrix formed from the components of n as follows:

$$X(n) = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}. \tag{1.60}$$

Therefore to represent the rotation, we need vector n, and the angle θ. We can further simplify this, and represent the rotation by only a vector, r, such that:

$$r = ||r|| \frac{r}{||r||} = \theta \; n. \tag{1.61}$$

The magnitude of r is the amount of rotation, and the direction of of r is the axis of rotation. Finally, the rotation matrix can be written as:

$$R^r = R^r_{||r||} = I + \sin\theta \frac{X(r)}{||r||} + (1 - \cos\theta) \frac{X(r)^2}{||r||^2}, \tag{1.62}$$

where $X(r)$ is given by:

$$X(r) = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}. \tag{1.63}$$

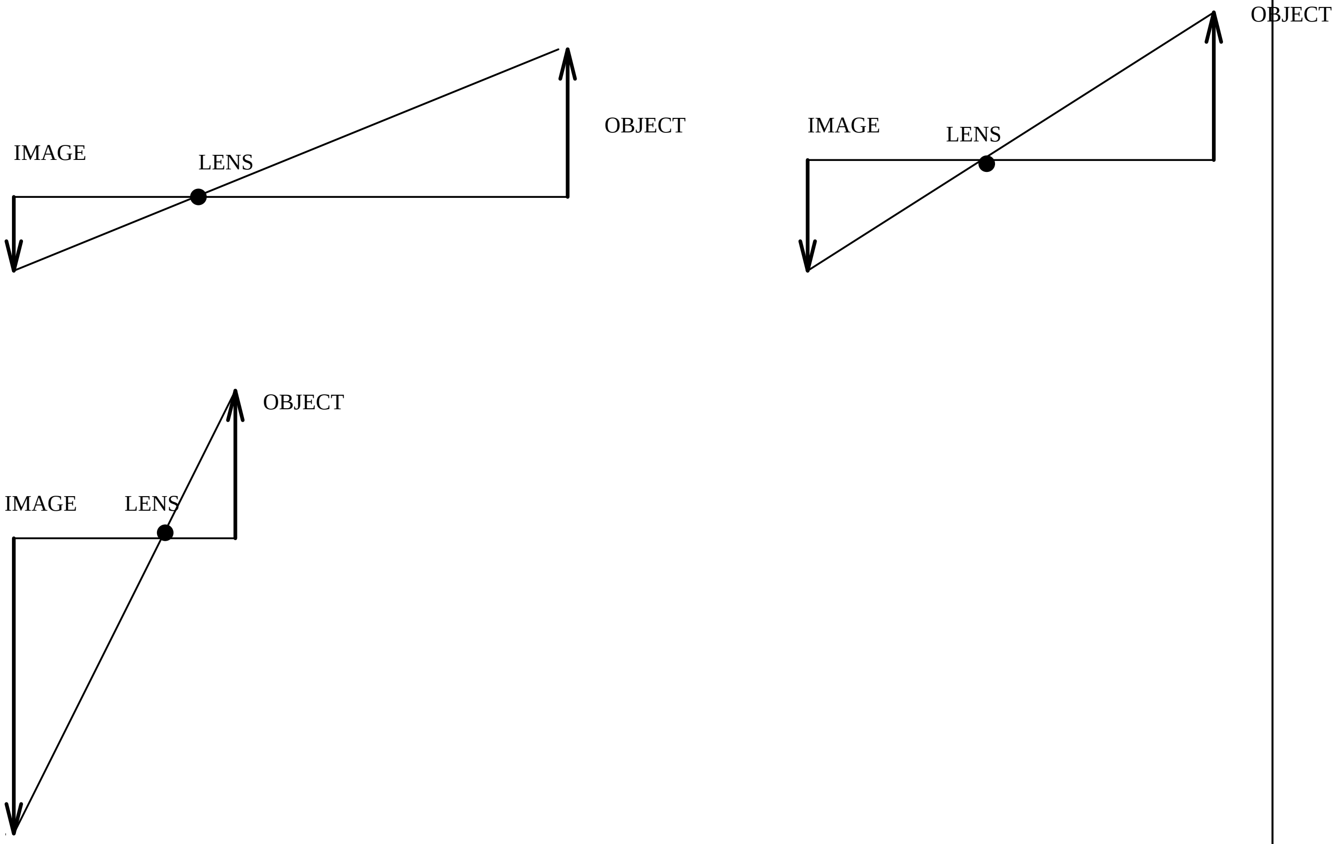


Figure 1.5: Determining the orientation of camera. As the object moves closer to the lens its image moves farther away from the image center along the $Y-$axis. When the object is at the lens, the image will be formed at infinity.

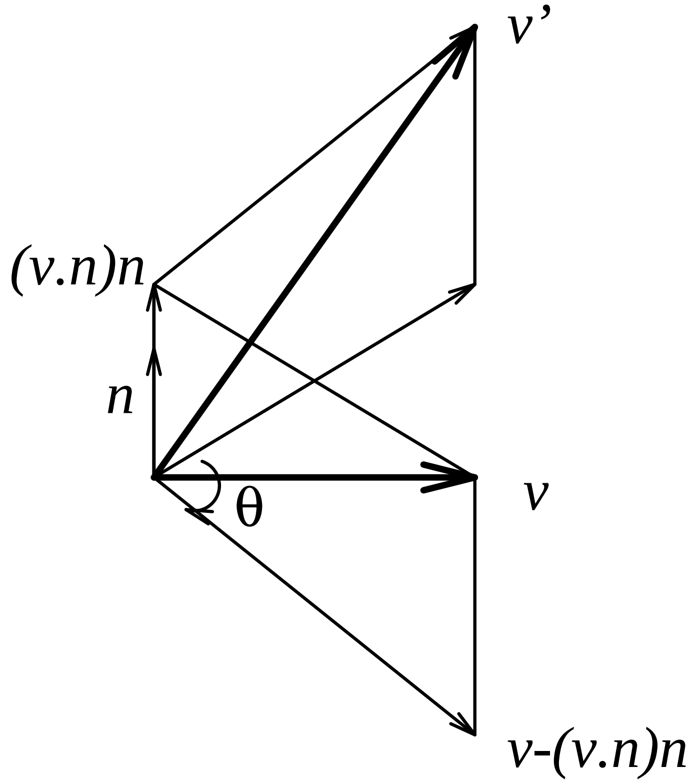


Figure 1.6: Rotation of vector r around vector n by angle θ.

1.9 Quaternions

A quaternion is a pair $q = (s, w)$, where s is a scalar and is commonly called real part of q, and w is a 3-D vector, and is called imaginary part of q. Two quaternions can be added like any 4-D vectors. However, multiplication of two quaternions is given by:

$$(s, w) * (s', w') = (ss' - w.w', w \times w' + sw' + s'w), \tag{1.64}$$

where "$\times$" and "." denote the usual vector and dot products. The conjugate and the norm of a quaternion are defined as:

$$\begin{aligned} \bar{q} &= (s, -w), & (1.65) \\ |q|^2 &= \bar{q} * q = ||w||^2 + s^2. & (1.66) \end{aligned}$$

For any rotation R of a 3-D vector, p, there exist two opposite quaternions q and $-q$ which satisfy $Rp = q * p * \bar{q}$ (p is converted to a quaternion by assuming zero real part, and similarly the real part of the resultant quaternion on the right is set to zero to match with the rotated vector on the left side). The quaternion q is computed from the unit direction vector n of the axis of rotation and its angle θ by the formula:

$$q = (\cos(\frac{\theta}{2}), \sin(\frac{\theta}{2})n). \tag{1.67}$$

Similarly, to each pair of unit quaternions $(q, -q)$ there is a associated unique rotation matrix

R,

$$R = \begin{bmatrix} a^2+b^2-c^2-d^2 & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & a^2+c^2-b^2-d^2 & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & a^2+d^2-b^2-c^2 \end{bmatrix} \tag{1.68}$$

where $q = (a, b, c, d)$.

1.10 Pose Estimation

In pose estimation, the problem is to determine the orientation and position of an object (camera) which would result in the projection of a given set of three-dimensional points into a given set of image points. One of the important applications of pose estimation is in the model based object recognition. In 2-D to 3-D model based object recognition a 3-D model of the object, and its 2-D projection from a particular vantage point of the camera is given, the aim is to determine three rotations and three translations relative to some base coordinate system.

Let (X', Y', Z') be the 3-D model coordinates of a point, and (x', y') be the projected image coordinates. Assume that the object (camera) is rotated first by (ϕ_X, ϕ_Y, ϕ_Z), that maps (X', Y', Z') to (X, Y, Z). Next, the object is translated by (T_X, T_Y, T_Z), and finally the image is produced using perspective transformation using lens centered coordinate system, and taking positive value of f. The image coordinates after perspective projection are given by:

$$(x', y') = (\frac{f(X+T_X)}{Z+T_Z}, \frac{f(Y+T_Y)}{Z+T_Z}). \tag{1.69}$$

In order to simplify these equations, instead of using translation (T_X, T_Y), we will use (D_X, D_Y), which are the projected image displacement in x and y direction, and D_Z is 3-D translation in Z direction. Now, the image coordinates are given by:

$$(x', y') = (\frac{fX}{Z+D_Z} + D_X, \frac{fY}{Z+D_Z} + D_Y) = (fXc + D_X, fYc + D_Y), \tag{1.70}$$

where $c = \frac{1}{Z+D_Z}$.

The pose estimation can be formulated as an optimization problem which minimizes the error between the model and image coordinates [12]. The error between the components of model and the image can be expressed in terms of small changes in the six unknowns $(D_X, D_Y, D_Z, \phi_X, \phi_Y, \phi_Z)$, and their derivatives with respect to the image coordinates. Since errors are known, the small changes in the parameters can iteratively be computed from these equations, and the next estimate can be obtained by adding the changes to the previous estimate. The equation for error, $E_{x'}$, in x' image coordinate as the sum of the products of its partial derivatives times the error correction values (using the first order Taylor series approximation) can be written as:

$$E_{x'} = \frac{\partial x'}{\partial D_X}\Delta D_X + \frac{\partial x'}{\partial D_Y}\Delta D_Y + \frac{\partial x'}{\partial D_Z}\Delta D_Z + \frac{\partial x'}{\partial \phi_X}\Delta \phi_X + \frac{\partial x'}{\partial \phi_Y}\Delta \phi_Y + \frac{\partial x'}{\partial \phi_Z}\Delta \phi_Z \tag{1.71}$$

The partial derivatives in the above equation can be easily computed from equation 1.70. For example, $\frac{\partial x'}{\partial D_X} = 1$, and $\frac{\partial x'}{\partial \phi_Y} = f \frac{\partial X}{\partial \phi_Y} \frac{1}{Z+D_Z} - \frac{fX}{(Z+D_Z)^2} \frac{\partial Z}{\partial \phi_Y}$. We can compute $\frac{\partial X}{\partial \phi_Y}$ and $\frac{\partial Z}{\partial \phi_Y}$ from the rotation matrix, R^Y_ϕ. As we know if (X', Y', Z') is rotated around Y-axis by angle ϕ_Y, the new coordinates (X, Y, Z) are given by:

$$\begin{aligned} X &= X' \cos \phi_Y + Z \sin \phi_Y, & (1.72)\\ Y &= Y', & (1.73)\\ Z &= -X' \sin \phi_Y + Z' \cos \phi_Y. & (1.74) \end{aligned}$$

Then,

$$\begin{aligned} \frac{\partial X}{\partial \phi_Y} &= -X' \sin \phi_Y + Z' \sin \phi_Y = Z, & (1.75)\\ \frac{\partial Y}{\partial \phi_Y} &= 0, & (1.76)\\ \frac{\partial Z}{\partial \phi_Y} &= -X' \cos \phi_Y - Z' \sin \phi_Y = -X. & (1.77) \end{aligned}$$

We can compute the other partial derivatives similarly. We can also write a second equation for y' image coordinates similar to equation 1.71, and compute the partial derivatives. The partial derivatives of x' and y' with respect to each camera parameter are given in the Figure 1.7.

For m image points we can write $2 \times m$ equations with six unknowns. This linear systems of equations can be written as follows:

$$A\Delta = E, \tag{1.78}$$

where A is the derivative matrix , Δ is the vector of six unknowns parameters, (D_X, D_Y, D_Z, ϕ_X, ϕ_Y, ϕ_Z), for determining orientation and location of camera, E is the vector of error terms. Now, the problem is to compute Δ, which can be done using the least squares fit as follows:

$$\Delta = (A^T A)^{-1} A^T E. \tag{1.79}$$

Figure 1.9 shows a demonstration of this method for pose estimation.

We can also use lines instead of points in the pose estimation. One possible approach is to measure errors as the perpendicular distance of each endpoint of the model line from the corresponding line in the image, and then take the derivatives in terms of this distance rather than in terms of x' and y'. The equation of line can be written as:

$$\frac{-m}{\sqrt{m^2+1}} x + \frac{1}{\sqrt{m^2+1}} y = d, \tag{1.80}$$

where m is the slope, and d is the perpendicular distance of the line from the origin. The equation of a line through point (x', y') can be obtained from equation 1.80 by substituting (x', y') instead of (x, y) and d' instead of d. Then the perpendicular distance of point (x', y') from the line is simply $d - d'$. It is easy to calculate the derivatives of d' for use in the iterative scheme, because the derivatives of d' are just linear combination of x' and y', which we already know.

	x'	y'
D_X	1	0
D_Y	0	1
D_Z	$-fc^2X$	$-fc^2Y$
ϕ_X	$-fc^2XY$	$-fc(Z+cY^2)$
ϕ_Y	$fc(Z+cX^2)$	fc^2XY
ϕ_Z	$-fcY$	fcX

Figure 1.7: The partial derivatives of image coordinates with respect to each of the camera parameters.

1. Start with an initial estimate of six parameters, $(D_X^0 D_Y^0, D_Z^0, \phi_X^0, \phi_Y^0, \phi_Z^0)$. If you do not know, assume all parameters to be zero.

2. Apply transformation to the model, and project the model on the image plane by computing (x', y').

3. Compute the errors $E_{x'}$, and $E_{y'}$. If the errors are acceptable, quit.

4. Find change in six parameters,$(\Delta D_X, \Delta D_Y, \Delta D_Z, \Delta\phi_X, \Delta\phi_Y, \Delta\phi_Z^0)$ of transformation, by using least squares fit. Goto step (2).

Figure 1.8: Pose estimation algorithm.

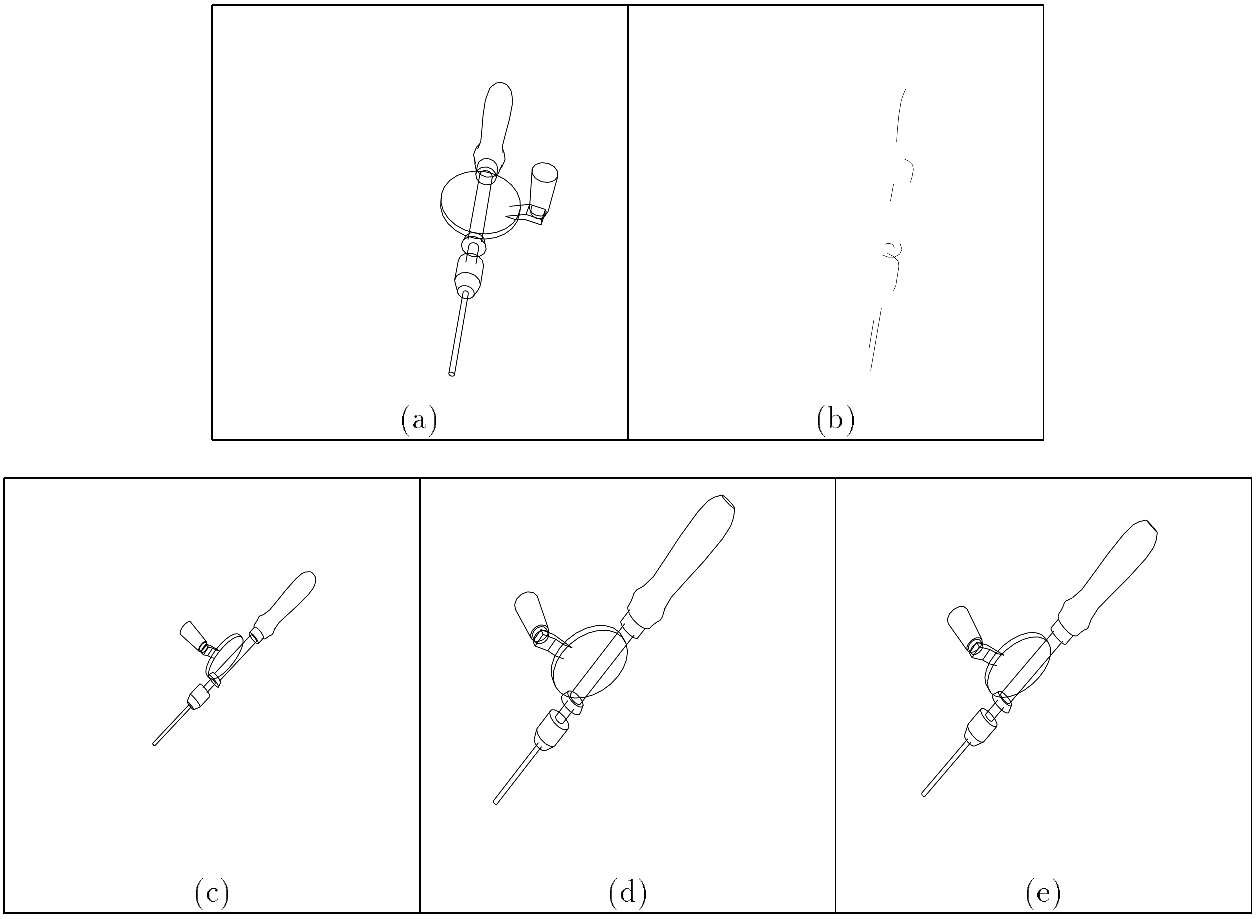


Figure 1.9: (a) The $3-D$ model of a drill. (b) The projection of the edges, when the drill is in unknown pose. (c) The position of drill after one iteration. (d) The position of drill after two iterations. (e) The position of drill after three iterations, which is the correct pose of the drill.

1.11 Exercises

1. What is the rotation matrix for an object rotation of 30^0 around the **Z** axis, followed by 60^0 around the **X** axis, and followed by a rotation of 90^0 around the **Y** axis. All rotations are counter clockwise.

2. What is the rotation matrix for an object rotation of Φ around the **X** axis, followed by Ψ around the **X** axis, and followed by a rotation of Θ around the **Z** axis. All rotations are counterclockwise.

3. Consider a vector $(7, 3, 2)$ which is rotated around the **Z** axis by 90^0, and then rotated around the **Y** axis by 90^0 and finally translated by $(4, -3, 7)$. Find the new coordinates of the vector. All rotations are clockwise.

4. Show that following identities are true.
$R^Y_{90}R^X_{90} = R^Z_{270}R^Y_{90}$, $R^Y_{90}R^X_{180} = R^Z_{180}R^Y_{90}$
$R^y_{180}R^X_{90} = R^Z_{180}R^X_{270}$, $R^Y_{180}R^X_{270} = R^Z_{180}R^X_{90}$.

5. Derive equations 1.28 and 1.29.

6. Consider the following camera model. Assume that first the camera gimbal center is at the origin of the world coordinates. Next, the camera is translated by amount $(0, 2, 2)$, and it is rotated by angle $90°$ around Z axis , followed by rotation of $135°$ around X axis. Both rotations are in the clockwise directions. After that the camera (image plane) is translated by amount $(.02, .01, .03)$ with respect to gimbal center. Finally, image is formed using perspective transform. Assume that focal length of the camera is .030. Find the image coordinates of the point which has world coordinates $(1, 1, 0.2)$.

7. Show how any image point (x_0, y_0) is mapped to the world coordinates using the inverse perspective matrix, verify that the equations 1.25 and 1.26 are satisfied.

8. Derive equations 1.53 and 1.54.

9. Verify equations 1.56, 1.60 and 1.62.

10. Given 3×3 rotation matrix, R, compute r in the Rodrigues' formula.

11. Verify equation 1.68.

12. Given 3×3 rotation matrix, R, compute quaternion q.

13. Suppose we want to minimize the squared sum of distances between a set of points p_i and their rotated versions $p'_i = Rp_i$. Show that minimizing criterion
$$C = \sum_i ||p'_i - Rp_i||^2, \tag{1.81}$$
can be written as
$$C = \sum_i [A_i q]^t [A_i q] = q^t [\sum_i A_i^t A_i] q, \tag{1.82}$$
where A_i is a 4×4 matrix depending on p_i and p'_i. The solution is given by the unit eigen vector associated with the smallest eigen value of the matrix $B = \sum_i A_i^t A_i$.

14. Assume p, q and r be quaternions. Show that:

$$\begin{aligned}
\overline{(p*q)} &= -(\bar{q}*\bar{p}), & (1.83)\\
(p*q).(p*q) &= (p.p).(q.q), & (1.84)\\
(p*q).(p*r) &= (p.p).(q.r), & (1.85)
\end{aligned}$$

where . is the dot product.

15. A rotation matrix R for rotation around an arbitrary vector by some angle can be expressed as the product of three rotations (Euler's angles): rotation around X axis by angle γ, followed by the rotation around Y axis by angle β and followed by the rotation around Z axis by angle α. All the rotations are counter clockwise. Show that:

$$R = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}.$$

16. For small rotations we can assume $\sin\theta = \theta$, and $\cos\theta = 1$. Simplify the above matrix assuming small rotations. Is the rotation matrix now linear in α, β and γ?

17. Given 3×3 rotation matrix, R, compute Euler's angles α, β and γ.

18. Derive the expressions for the partial derivatives given in the Table shown in Figure 1.7.

19. By using equation 1.80 compute the perpendicular distance of a point (x', y') from a line.

Chapter 2

Edge Detection

2.1 Introduction

Edge detection which has attracted the attention of many researchers[1] is one of the most important areas in lower level computer vision. However, it seems that the problem of detecting edges in real scenes is still an unsolved problem in a general sense. Edge detection is important because the success of higher level processing relies heavily on good edges. Gray level images contain an enormous amount of data, much of which is irrelevant, e.g., the background of the scene. So at the initial stage, the effort is made to reduce some of the data; the objects are separated from the background and entities such as edges which are physically significant are identified.

In the stereo techniques, for instance, images taken from two different locations are utilized to get the depth information. All stereo techniques deal with the "correspondence", where the tokens from the right and left image are matched in order to get the disparity of two images. The edge information plays an important role in the selection of tokens. In motion, the detection of moving objects is done by identifying the time varying edges and corners. The approaches to structure from motion, through which the three dimensional structure of objects is computed, also require correspondence of tokens. Object recognition methods based on only the two dimensional shape use the edge information. Most algorithms for recognizing the partially occluded objects assume the presence of good edges.

2.2 Types of edges

Since the gray level function changes significantly at the edges of objects, the ideal edge can be modeled by the step function. In real images, however, the gray level at the edges of an object does not change abruptly, but instead, changes gradually. These edges can be modeled by the ramp function. Sometimes two steps and/or ramps appear close together; these are respectively called spike and roof edges. The possible types of edges are shown in Figure 2.1.

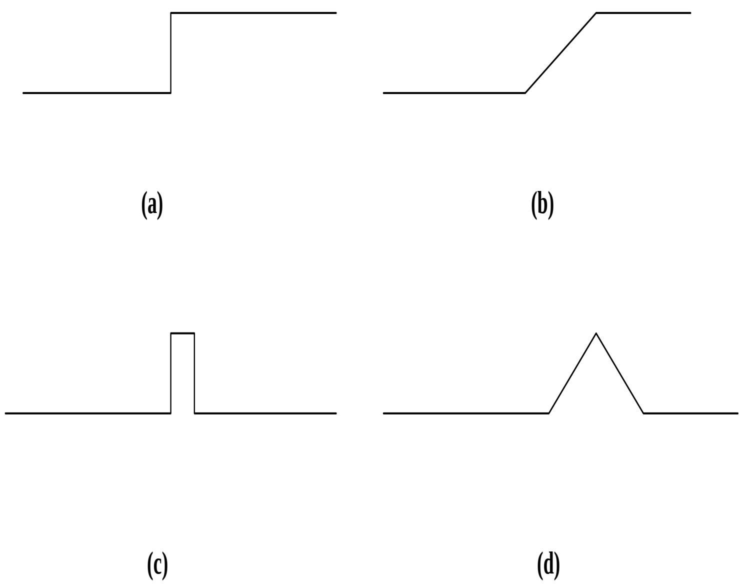


Figure 2.1: Types of Edges. (a) Step, (b) Ramp, (c) Spike, and (d) Roof.

2.3 Three stages in edge detection

Most edge detection schemes consist of three stages: *filtering, differentiation*, and *detection*. In the filtering stage, the image is passed through a filter in order to remove the noise. Noise can be due to the undesirable effects introduced in the sampling, quantization, blurring and defocusing of the camera, and irregularities of the surface structure of the objects. The differentiation stage highlights the locations in the image where intensity changes are significant. Finally, in the detection stage, those points where the intensity changes are significant are localized.

In the earlier approaches the filtering step was skipped, the differentiation step was performed by using the finite-difference approach, and the detection was done by locating the peaks in the gradient of the intensity function using a threshold. In these approaches, filtering was not crucial since only synthetic images and the industrial scenes with a controlled environment were considered. Later, Sobel introduced the averaging step before the differentiation step. Since averaging is a kind of filtering, better results were obtained by the Sobel operator. Determination of the threshold depends on the domain of the application and varies from image to image. Automatic selection of thresholding is still a hard problem.

Haralick [4] performs the differentiation step by first fitting a piecewise continuous polynomial to the gray level neighborhood of a pixel, and then finding the partial derivatives of the polynomial. In this approach, the filtering step is not explicit, but the fitting essentially smooths the gray level image and thus performs a filtering step. Intuitively, the error of fit controls the degree of smoothing. Canny [3, 2] uses Gaussian in the filtering stage and computes the first directional derivative along the gradient direction during the differentiation stage. In the detection stage Canny, detects peaks in the derivative output to locate the edge points. Marr and Hildreth also use Gaussian as a filter [13]. They perform differentiation step by the Laplacian, which is the sum of the second partial derivatives along the x- and y-axis.

2.4 Filtering Stage

The most simple filter is the mean filter. In the mean filter, the gray level at each picture element (pixel) is replaced by the mean of gray levels in a small neighborhood around the pixel. This way the random noise is averaged out. Let f denote the image which we want to filter, h denote the filtered image, and consider a 3×3 neighborhood around a pixel. Then

$$\begin{aligned} h(x,y) &= f(x-1,y-1)\frac{1}{9} + f(x-1,y)\frac{1}{9} + f(x-1,y+1)\frac{1}{9} + f(x,y-1)\frac{1}{9} \\ &+ f(x,y)\frac{1}{9} + f(x,y+1)\frac{1}{9} + f(x+1,y-1)\frac{1}{9} + f(x+1,y)\frac{1}{9} + f(x+1,y+1)\frac{1}{9} \quad (2.1) \\ h(x,y) &= f(x-1,y-1)g(-1,-1) + f(x-1,y)g(-1,0) + f(x-1,y+1)g(-1,1) \\ &+ f(x,y-1)g(0,-1) + f(x,y)g(0,0) + f(x,y+1)g(0,1) \\ &+ f(x+1,y-1)g(1,-1) + f(x+1,y)g(1,0) + f(x+1,y+1)g(1,1) \quad (2.2) \\ h(x,y) &= \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} f(x+i,y+j)g(i,j) \quad (2.3) \\ h(x,y) &= f(x,y) * g(x,y). \quad (2.4) \end{aligned}$$

Here g denote the filter, and $*$ denotes the convolution operation. The filter in this case is 3×3, and each pixel in the filter is set to $\frac{1}{9}$. The filter g is convolved with the image f and the result is a filtered image h. The convolution operation is widely used in Computer Vision and Image Processing. The convolution operation can be visualized as passing around a filter mask (mask g) in the image from left to right, top to bottom as shown in Figure 2.2. The mask is centered at each pixel in image f, and the sum of the corresponding pixel-by-pixel product is computed.

The mean filter performs equal averaging; each pixel value around a small neighborhood is assigned equal weight. Another useful filter is called Gaussian. In the Gaussian filter each pixel's weight is inversely proportional to the distance from the central pixel. The Gaussian is defined as:

$$g(x,y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}}, \quad (2.5)$$

where σ is the standard deviation of Gaussian, and controls the mask size. The mask of $g(x,y)$ can be generated for a given value of σ by taking various values of x, and y, and substituting in the above formula. For example, with $x = -1$, $y = -1$, and $\sigma = 1$, $G(-1,-1) = e^{\frac{-((-1)^2+(-1)^2)}{2(1)^2}} = 0.367$, similarly $G(0,1) = 0.60$. It is common to scale the coefficients in the Gaussian filter to the nearest integers. Because the image gray levels are integers, it is simpler to perform integer multiplication as compared to real number multiplications. The mask of the Gaussian filter for $\sigma = 2$, multiplied by 255, and truncated to the nearest integer is shown in Figure 2.3. Note that this is a 13×13 mask. For values

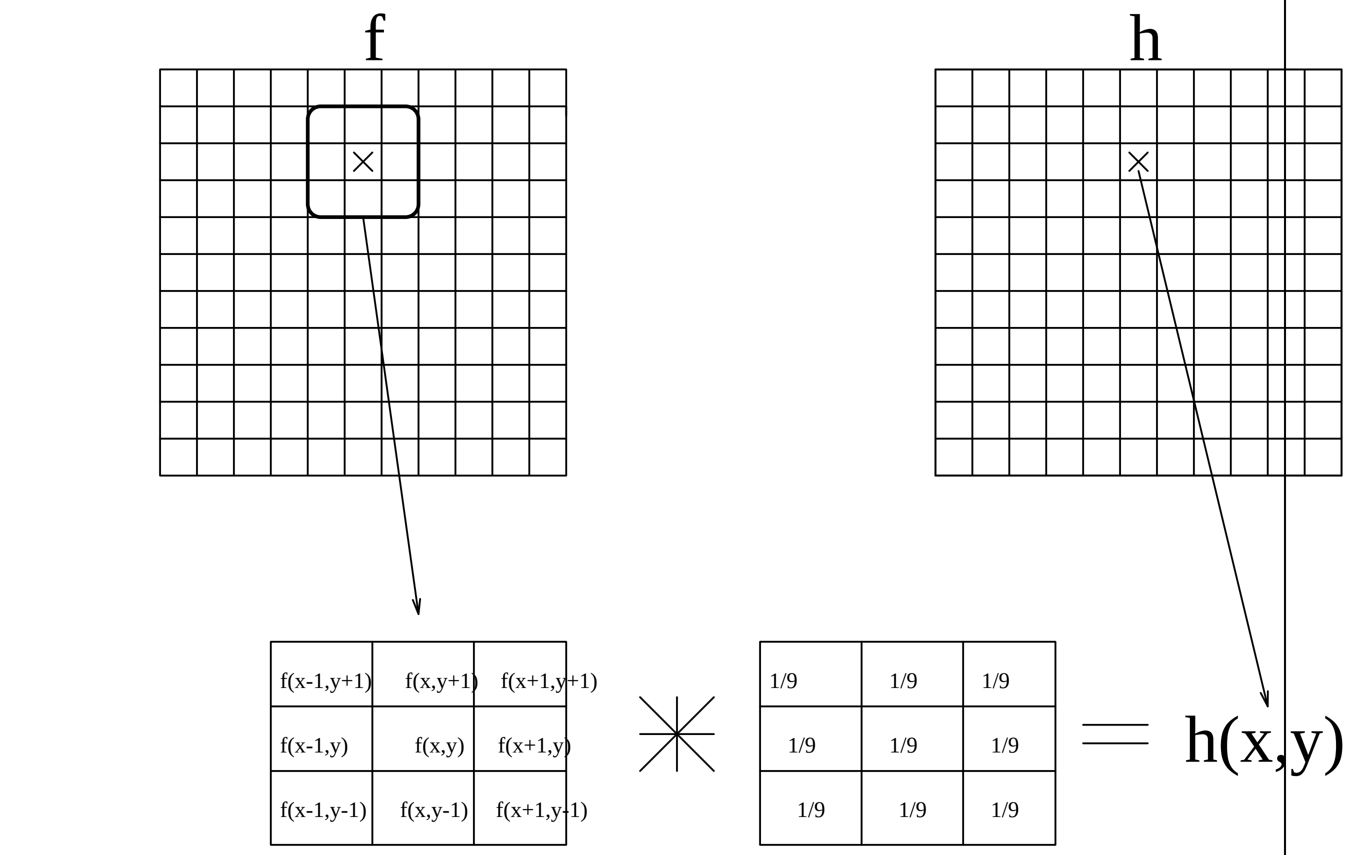


Figure 2.2: Convolution operation.

$x, y > |6|$, the $G(x, y)$ is very small. Therefore, those values have been ignored. However, for the higher values of σ, the mask size has to be bigger. Since the Gaussian function becomes less than 1% of its maximum value for $x, y > 3\sigma$, it is appropriate to use a mask size of at least $6\sigma + 1$.

2.5 Differentiation Stage

The derivative of a continuous function f is defined as:

$$f' = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x}. \tag{2.6}$$

Since images are discrete, the minimum value of Δx can be 1. Now the above formula reduces to:

$$f' = \frac{df}{dx} = f(x) - f(x - 1). \tag{2.7}$$

This is called a discrete approximation of a derivative. Consider a row profile from an image as shown in Figure 2.4.a. Its derivative by using the above formula is given in Figure 2.4.b.

0	0	0	0	1	2	2	2	1	0	0	0	0
0	0	1	3	6	9	11	9	6	3	1	0	0
0	1	4	11	20	30	34	30	20	11	4	1	0
0	3	11	26	50	73	82	73	50	26	11	3	0
1	6	20	50	93	136	154	136	93	50	20	6	1
2	9	30	73	136	198	225	198	136	73	30	9	2
2	11	34	82	154	225	255	225	154	82	34	11	2
2	9	30	73	136	198	225	198	136	73	30	9	2
1	6	20	50	93	136	154	136	93	50	20	6	1
0	3	11	26	50	73	82	73	50	26	11	3	0
0	1	4	11	20	30	34	30	20	11	4	1	0
0	0	1	3	6	9	11	9	6	3	1	0	0
0	0	0	0	1	2	2	2	1	0	0	0	0

Figure 2.3: Gaussian mask with $\sigma = 2$.

Note that the derivative can be computed by applying a mask to f, similar to filtering an image. In this case the mask simply consists of two elements as shown in Figure 2.4.d. f is a step edge, and its derivative gives a high value at the location of an edge. There are several other masks widely used for computing the first derivative as shown in Figure 2.4.e–f.

Images are two dimensional. A gradient vector of $f(x, y)$ is (f_x, f_y), where f_x is the derivative in the x direction, and f_y is the derivative in the y direction. The gradient has magnitude M, and direction θ as defined below:

$$\theta = \arctan \frac{f_y}{f_x}, \tag{2.8}$$

$$M = \sqrt{{f_x}^2 + {f_y}^2}. \tag{2.9}$$

The directional derivative in the direction θ is defined:

$$f'_\theta = \frac{\partial f}{\partial x}\, cos\theta + \frac{\partial f}{\partial y} \sin\theta.$$

There are several masks for computing gradient (f_x, f_y), some are shown in Figure 2.6.

2.6 Detection Stage

2.6.1 Normalized Gradient Magnitude

Let $M(x, y) = \sqrt{f_x^2(x, y) + f_y^2(x, y)}$ be a gradient magnitude at pixel (x, y). Define $N(x, y)$ as the normalized gradient magnitude scaled between 0 to 100 as follows:

$$N(x, y) = \frac{M(x, y)}{max_{i=1,\dots,n.j=1,\dots,n} M(i, j)} \times 100. \tag{2.10}$$

(a)	f(x)	10	10	10	10	10	20	20	20	20	20
(b)	f'(x)	0	0	0	0	0	10	0	0	0	0
(c)	f''(x)	0	0	0	0	0	10	-10	0	0	0

-1	1

(d)

-1	0	1

(e)

-1	0	0	1

(f)

Figure 2.4: Differntiation. (a) A one dimensional function $f(x)$. (b) Derivative $f'(x)$ using the discrete approximation. (c) Second derivative $f''(x)$. (d) Derivative mask. (e)–(f) Two additional derivative masks.

Then edges can be detected by applying a threshold T on the normalized gradient magnitude N.

$$E(x,y) = \begin{cases} 1 & \text{if } N(x,y) > T \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

where $E(x,y)$ is an edge map.

2.6.2 Non-maxima Suppression

In the above detection method, we only used the gradient magnitude; we did not use the gradient direction. The gradient direction is always perpendicular to the edge, and the gray levels change the most in that direction. If a certain pixel's gray level is not changing significantly in the direction of gradient, then that pixel probably is not an edge point. We need to suppress that (non maxima) point. Mathematically the non-maxima suppression is defined as:

$$M(x,y) = \begin{cases} M(x,y) & \text{if } M(x,y) > M(x',y') \text{ and} \\ & \text{if } M(x,y) > M(x'',y'') \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

where $M(x',y')$ and $M(x'',y'')$ are the gradient magnitudes on both sides of edge at (x,y) in the direction of the gradient as shown in Figure 2.5.a. This step will suppress the points which are not potential edge points. Now the non-maxima suppressed gradient can be normalized as done in the previous section, and the threshold can be applied to produce a better edge map. It is necessary to quantize the gradient direction into a fixed number of directions. During the non-maxima suppression the gradient magnitude at the appropriate pixels can then be compared. In one possible scheme the gradient direction is quantized into eight directions consisting of $0, 45, 90, 135, \ldots, 315$. Figure 2.5.b shows the corresponding pixels to be compared during the non-maxima suppression step.

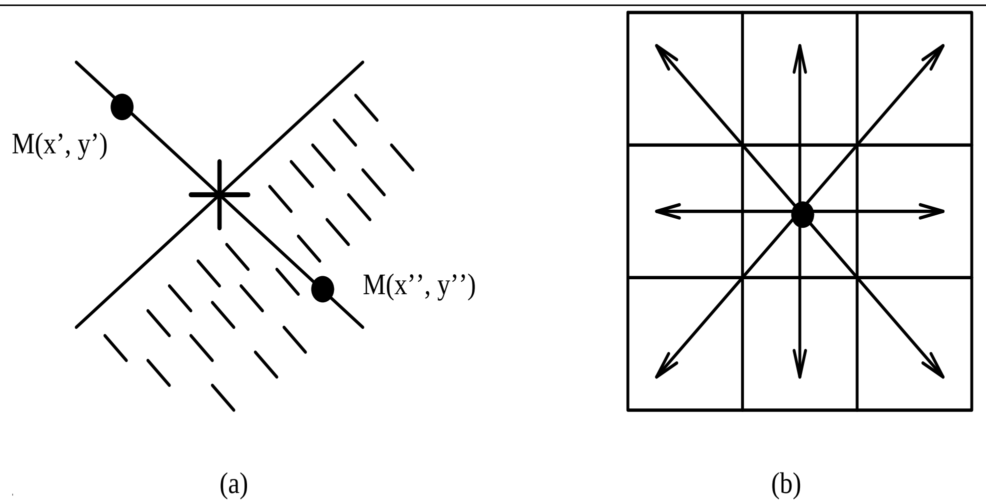


Figure 2.5: (a) Non-maxima suppression. (b)Eight possible gradient directions.

2.7 Classes of edge detection schemes

The edge detection schemes can be classified in three categories. The first category deals with the early 3×3 and 5×5 gradient operators, which includes Prewit, Robert, Sobel, and Laplacian operators. In the second category, the operators which are based on a kind of surface fitting are included. The operators of Hueckel [10], Hartly [5], and Haralick's [4] facet model approach to edge detection belong to this category. Finally, the third category of edge detection schemes consists of the techniques which employ the derivatives of Gaussian. Recently, the operators in this class have become quite popular because these operators perform reasonably well for real scenes. Gaussian does a kind of weighted averaging of the gray level image in order to remove the noise. These operators use fairly larger masks and therefore are computationally expensive. However, some efforts have been made to reduce computational complexity by utilizing some properties of operators, and VLSI based implementation have also been proposed [20].

2.8 Gradient Operators

A number of 3×3 and 5×5 edge operators are known. The first of such operators was proposed by Robert. In addition to Robert's operator, Prewit and Sobel operators have been used extensively in the past. These operators essentially compute the gradient at a pixel location along the principle directions. The most common directions are X and Y, but diagonal and anti-diagonal directions have also been used. The computation of the gradient is performed by the finite difference method for differentials, as discussed in the section on differentiation.

In Figure 2.6 we show the Robert, Prewit and Sobel operators in their most simple form. The Robert and Prewit operators assign equal weights to the pixels, but in the Sobel operator, the closer pixels are assigned more weight.

The computation of the magnitude of the gradient involves square root operation, which is expensive. Therefore, various simplified approximations of the gradient magnitude have been

0	1		1	0
1	0		0	-1

(a)

-1	0	1		1	1	1
-1	0	1		0	0	0
-1	0	1		-1	-1	-1

(b)

-1	0	1		1	2	1
-2	0	2		0	0	0
-1	0	1		-1	-2	-1

(c)

0	-1	0
-1	4	-1
0	-1	0

(d)

Figure 2.6: Classical Gradient Operators. (a) Robert's operator, (b) Prewit's operator, (c) Sobel's operator, (d) Laplacian operator.

used such as: (a) $max(|\Delta x|, |\Delta y|)$, (b) $|\Delta x| + |\Delta y|$. The first one avoids the computational cost of the square and square root operations. In addition, it yields values in the same range as the original gray scale, which is convenient for display purposes. While the second approximation has a larger range of possible values, and introduces a bias for and against the diagonal edges.

The Laplacian operator has also been used as an edge operator. The Laplacian is given by:

$$\Delta^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

This is an orientation independent derivative operator. Its discrete approximation is given as follows:

$$\Delta^2 f(x,y) = -[f(x+1,y) + f(x-1,y) + f(x,\ y+1) + f(x,\ y-1)] + 4f(x,y).$$

The convolution mask for this operator is shown in Figure 2.6.d. In a noisy picture, the noise will produce higher Laplacian values than the edges.

There is no explicit filtering stage in these earlier operators. They perform well in synthetic scenes and in the real scenes where the illumination is controlled. The resulting edges are thick, therefore, extra thining algorithms have to be applied. However, since they are computationally cheap they are still popular in industry. Commonly used mask sizes for these operators are 3×3 and 5×5. The bigger masks for some of the operators can be computed, but, the resulting edges are very thick. Haralick [4] reports that "It is obvious from the experiments that good gradient operators must have larger neighborhood sizes than 3×3. Unfortunately, the larger neighborhood sizes also yield thicker edges."

2.9 Facet Model

There are a number of edge detectors that are based on some kind of image surface modeling. These methods usually involve an initial parameterization of the image in terms of basis functions followed by the estimation of the amplitude and position of the best fitting steps edge from the parameters. One of the earliest examples of this method was the Prewit operator [19], which used a quadratic set of basis functions. The Prewit operator is commonly known as the 3×3 operator which was discussed in the previous section.

Haralick [4] fits a bi-cubic polynomial to the neighborhood of a pixel. He computes the second and the third directional derivatives in the direction of the gradient of the intensity function in terms of the coefficients of the polynomial. The coefficients for a given pixel location can be found by using the least-square fit to the gray level neighborhood of the pixel. A given pixel is declared as an edge point if (i) the second derivative is equal to zero, and (ii) the third derivative is negative.

Haralick considers the following polynomial in x and y which are the row and column coordinates.

$$f(x,y) = k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2 + k_7x^3 + k_8x^2y + k_9xy^2 + k_{10}y^3. \quad (2.13)$$

The gradient angle, θ, with positive y-axis at $(x,y) = (0,0)$ is given by

$$\sin\theta = \frac{k_2}{\sqrt{{k_2}^2 + {k_3}^2}}, \quad (2.14)$$

$$\cos\theta = \frac{k_3}{\sqrt{{k_2}^2 + {k_3}^2}}. \quad (2.15)$$

For a given direction vector $(\sin\theta, \cos\theta)$ the first and second directional derivatives are given by the following equations.

$$f'_\theta(x,y) = \frac{\partial f}{\partial x}\sin\theta + \frac{\partial f}{\partial y}\cos\theta, \quad (2.16)$$

$$f''_\theta(x,y) = \frac{\partial^2 f}{\partial x^2}\sin^2\theta + \frac{\partial^2 f}{\partial y^2}\cos^2\theta + 2\frac{\partial^2 f}{\partial x \partial y}\cos\theta\sin\theta. \quad (2.17)$$

Using the cubic polynomial approximation of f given by equation 2.13 the expressions for $\tan\theta$, $\sin\theta$, and $\cos\theta$, at $x = 0, y = 0$, where θ is the gradient angle, can be computed. Now applying the substitution of variables $x = \rho\sin\theta$, $y = \rho\cos\theta$ into equation 2.13, $f(x,y)$ can be changed into $f_\theta(\rho)$ as shown below:

$$f_\theta(\rho) = C_0 + C_1\rho + C_2\rho^2 + C_3\rho^3, \quad (2.18)$$

where

$$\begin{aligned} C_0 &= k_1, \\ C_1 &= k_2\sin\theta + k_3\cos\theta, \\ C_2 &= k_4\sin^2\theta + k_5\sin\theta\cos\theta + k_6\cos^2\theta, \\ C_3 &= k_7\sin^3\theta + k_8\sin^2\theta\cos\theta + k_9\sin\theta\cos^2\theta + k_{10}\cos^3\theta. \end{aligned}$$

Differentiation the $f_\theta(\rho)$ with respect to ρ we get:

$$f_\theta{}'(\rho) = C_1 + 2C_2\rho + 3C_3\rho^2, \quad (2.19)$$

$$f_\theta{}''(\rho) = 2C_2 + 6C_3\rho, \quad (2.20)$$

$$f_\theta{}'''(\rho) = 6C_3. \quad (2.21)$$

From the second condition $f_\theta'''(\rho) < 0$, we get $6C_3 < 0$, or $C_3 < 0$, and from the first condition $f_\theta''(\rho) = 2C_2 + 6C_3\rho = 0$, we get $|\frac{C_2}{3C_3}| < \rho_0$.

The main steps in Haralick's edge detector are given in the Figure 2.9. The first step involves the computation of $k_1, k_2, k_3, \ldots, k_{10}$ using the least squares fit. Once the ks are known the detection of edges is very straight forward.

It is interesting to note that the computation of ks using least squares can be viewed as the application of masks to the image. We will show that by considering a simple case of a first order polynomial. However, the process can easily be extended to cubic or any higher order polynomials.

Consider a first order polynomial:

$$f(x,y) = k_1 + k_2x + k_3y. \tag{2.22}$$

We want to compute k_1, k_2, k_3 using least squares fit. We will consider a 3×3 window around each pixel in the image. Let us number pixels in the window as shown in Figure 2.7, and assume the origin to be at the center. Therefore, the local coordinates of (x, y) of each pixel are shown in Figure 2.7. If we substitute the (x, y) coordinates of a pixel and its corresponding gray level in the above equation, we get one equation. For nine such pixels in a small neighborhood we will get nine equations as follows:

$$\begin{aligned} f1 &= k_1 + k_2x_1 + k_3y_1, && (2.23)\\ f2 &= k_1 + k_2x_2 + k_3y_2, && (2.24)\\ &\vdots \\ f9 &= k_1 + k_2x_9 + k_3y_9. && (2.25)\\ & && (2.26) \end{aligned}$$

This system of equations can be written as:

$$\begin{bmatrix} f1 \\ f2 \\ \vdots \\ f9 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & & \\ 1 & x_9 & y_9 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \\ k_3 \end{bmatrix}, \tag{2.27}$$

$$f = Ak. \tag{2.28}$$

Now, vector k can be computed using the pseudo inverse method as follows:

$$(A^TA)^{-1}A^Tf = k, \tag{2.29}$$

$$Bf = k, \tag{2.30}$$

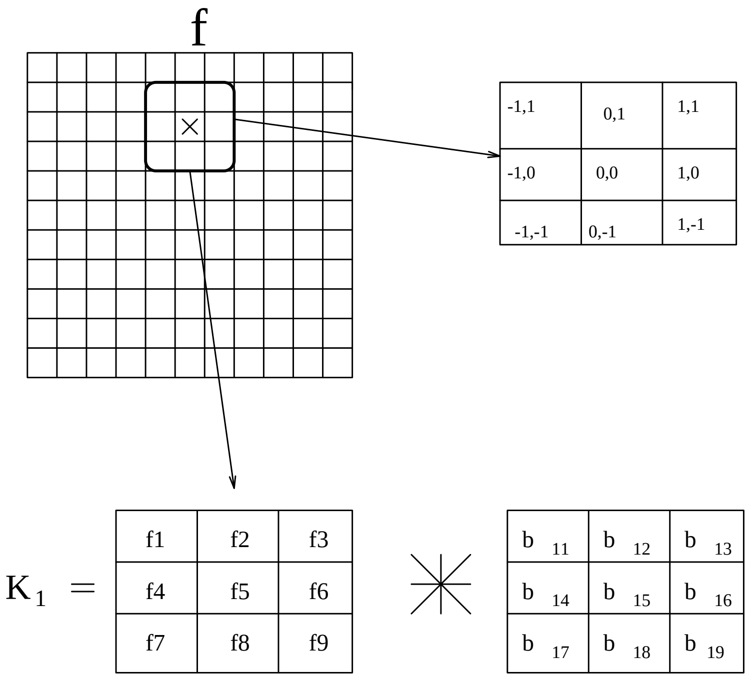


Figure 2.7: Facet convolution. k_1 for a pixel at 'X' is computed by applying a mask b_1 (determined from the first row of matrix B) to a 3×3 neighborhood of image.

where $B = (A^T A)^{-1} A^T$ is a 3×9 matrix. The above system can be written as:

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} & b_{17} & b_{18} & b_{19} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} & b_{27} & b_{28} & b_{29} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & b_{36} & b_{37} & b_{38} & b_{39} \end{bmatrix} \begin{bmatrix} f1 \\ f2 \\ f3 \\ f4 \\ f5 \\ f6 \\ f7 \\ f8 \\ f9 \end{bmatrix} = \begin{bmatrix} k1 \\ k2 \\ k3 \end{bmatrix}. \qquad (2.31)$$

Now, k_1, k_2, and k_3 can be computed from the above equation by using convolution. For example, k_1 is given by:

$$k_1 = b_{11}f1 + b_{12}f2 + b_{13}f3 + b_{14}f4 + b_{15}f5 + b_{16}f6 + b_{17}f7 + b_{18}f8 + b_{19}f9, \qquad (2.32)$$

which is convolution:

$$k_1 = f * b_1,$$

where, b_1 is the first row of matrix B. This shown in Figure 2.7.

The convolution masks for ks are given in Figure 2.8. The motivation behind the use of a facet model was to achieve an analytical expression which describe the gray level function around a pixel in the continuous domain. The definitions of partial derivatives in the continuous domain can be applied in order to detect discontinuities in the gray level images.

$\frac{1}{175}$

-13	2	7	2	-13
2	17	22	17	2
7	22	27	22	7
2	17	22	17	2
-13	2	7	2	-13

k_1

$\frac{1}{420}$

31	-5	-17	-5	31
-44	-62	-68	-62	-44
0	0	0	0	0
44	62	68	62	44
-31	5	17	5	-31

k_2

$\frac{1}{420}$

31	-44	0	44	-31
-5	-62	0	62	5
-17	-68	0	68	17
-5	-62	0	62	5
31	-44	0	44	-31

k_3

$\frac{1}{70}$

2	2	2	2	2
-1	-1	-1	-1	-1
-2	-2	-2	-2	-2
-1	-1	-1	-1	-1
2	2	2	2	2

k_4

$\frac{1}{100}$

4	2	0	-2	-4
2	1	0	-1	-2
0	0	0	0	0
-2	-1	0	1	2
-4	-2	0	2	4

k_5

$\frac{1}{70}$

2	-1	-2	-1	2
2	-1	-2	-1	2
2	-1	-2	-1	2
2	-1	-2	-1	2
2	-1	-2	-1	2

k_6

$\frac{1}{60}$

-1	-1	-1	-1	-1
2	2	2	2	2
0	0	0	0	0
-2	-2	-2	-2	-2
1	1	1	1	1

k_7

$\frac{1}{140}$

-4	-2	0	2	4
2	1	0	-1	-2
4	2	0	-2	-4
2	1	0	-1	-2
-4	-2	0	2	4

k_8

$\frac{1}{140}$

-4	2	4	2	-4
-2	1	2	1	-2
0	0	0	0	0
2	-1	-2	-1	2
4	-2	-4	-2	4

k_9

$\frac{1}{60}$

-1	2	0	-2	1
-1	2	0	-2	1
-1	2	0	-2	1
-1	2	0	-2	1
-1	2	0	-2	1

k_{10}

Figure 2.8: Haralick's masks for computing ks.

1. Find $k_1, k_2, k_3, \ldots, k_{10}$ using least square fit, or masks given in Figure 2.8.
2. Compute θ, $\sin\theta$, $\cos\theta$.
3. Compute C_2, C_3.
4. If $C_3 < 0$ and $|\frac{C_2}{3C_3}| < \rho_0$ then that point is an edge point.

Figure 2.9: The steps in Haralick's Edge Detector.

(a) (b)

Figure 2.10: (a). The bottle image. (b). The edge map from the Haralick edge detector with $\rho = 0.4$, and $C_3 = -1.7$.

1. Generate a mask for LG for a given σ using equation 2.34.
2. Apply mask to the image.
3. Detect Zerocrossings.
   (a) Scan along each row, record an edge point at the location of zerocrossing.
   (b) repeat step (a) along each column.

Figure 2.11: Steps in LG operator.

The results obtained from this edge dtector are shown in Figure 2.10.

2.10 Laplacian of Gaussian Operator

The Laplacian of Gaussian (LG) edge operator, proposed by Marr and Hildreth [13, 8, 14], has been widely used. This operator has been used for image segmentation as well as for image matching for passive stereo. Briefly, this operator has the following properties: It uses the Gaussian filter for noise elimination. A simple detection scheme is used. Since locating the extrema of functions which are flat is not a simple task, the edge points are located by detecting zerocrossings in the Laplacian of the intensity function, rather than locating extrema in the first partial derivative. Further, this operator does not require an explicit thresholding, and there is a possibility of extracting edges from different spatial frequency ranges of the image. Finally, the response of this operator resembles the response of the center-surround cells found in biological vision systems.

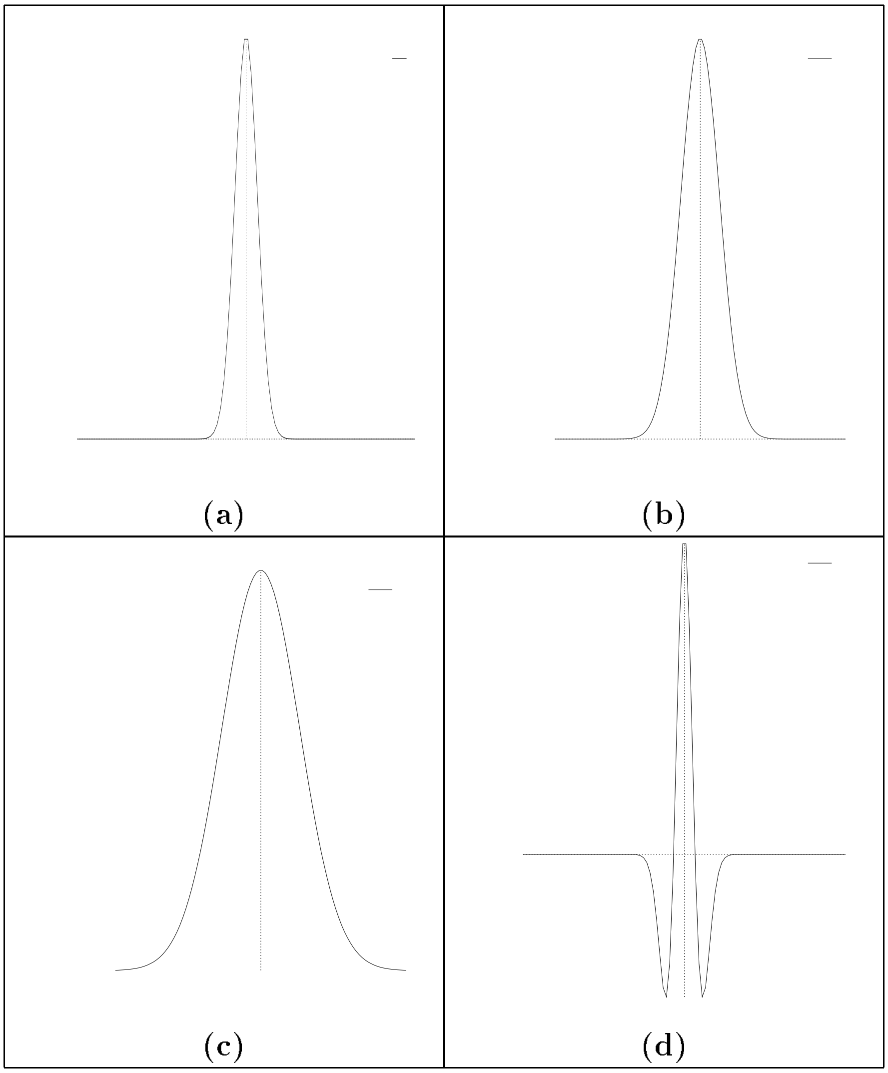


Figure 2.12: (a) 1-D Gaussian filter with $\sigma = 1$. (b) 1-D Gaussian filter with $\sigma = 2$. (c) 1-D Gaussian filter with $\sigma = 4$. (d) Second Derivative of Gaussian, g_{xx}, with $\sigma = 1$.

The analytical expressions of the operator in one and two dimensions are given in the following.

$$g_{xx}(x) = (1 - \frac{x^2}{\sigma^2})e^{\frac{-x^2}{2\sigma^2}}, \quad (2.33)$$

$$\Delta^2 g(x,y) = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} = k(2 - \frac{x^2+y^2}{\sigma^2})e^{\frac{-(x^2+y^2)}{2\sigma^2}}, \quad (2.34)$$

where k is a multiplicative constant which does not depend on (x, y). We show the 1-D profile of Gaussian for $\sigma = 1, 2, 3$ in Figure 2.12.a-c, and the second derivative of Gaussian in Figure 2.12.d. We show the 2-D profile of Gaussian and LG for $\sigma = 1, 2$ respectively in in Figures 2.13.a–b, and 2.12c–d.

In this edge detection scheme the image is convolved with LG operator first. Then zero-crossings in the convolved image, which correspond to the edges in the image, are detected.

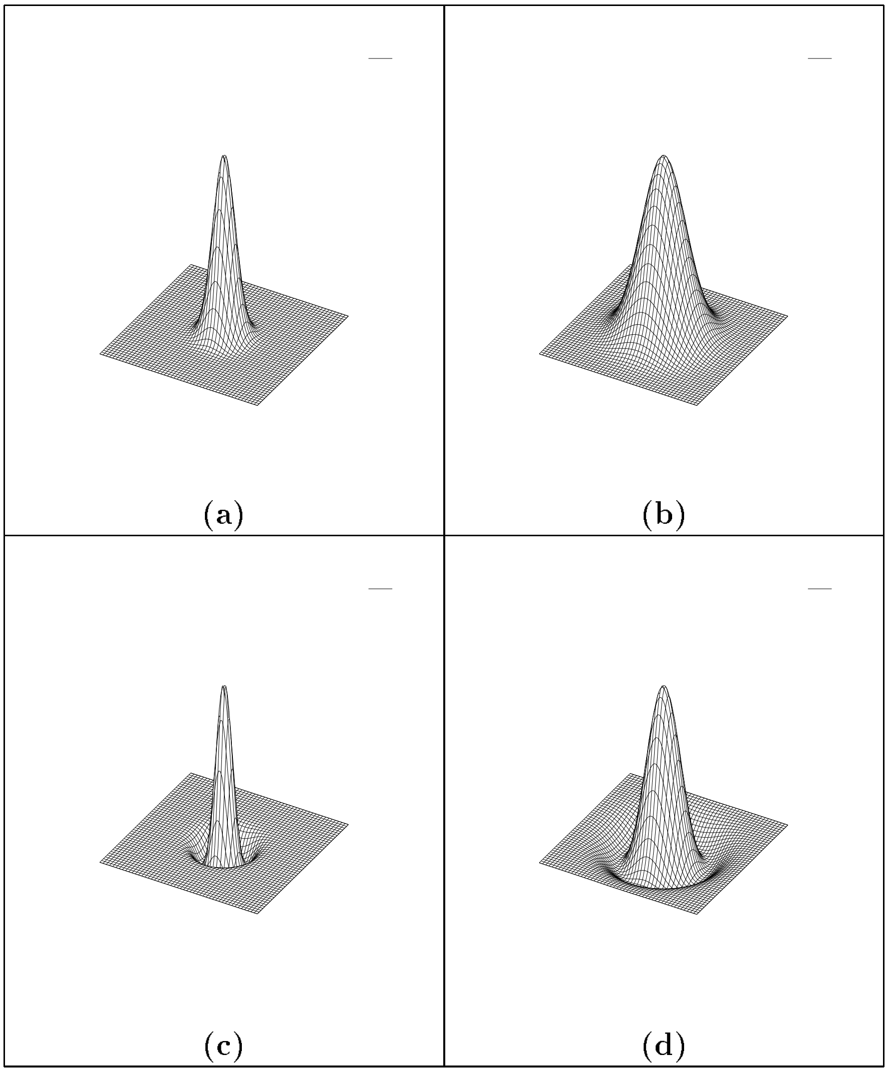


Figure 2.13: (a) 2-D Gaussian filter with $\sigma = 1$. (b) 2-D Gaussian filter with $\sigma = 2$. (c)LG with $\sigma = 1$. (d) LG with $\sigma = 2$.

(a) (b)

Figure 2.14: (a). The bottle image. (b). The edge map from the Laplacian of Gaussian edge detector.

There are four possible cases of zerorcrossings: $\{+,-\}$, $\{+,0,-\}$, $\{-,+\}$, and $\{-,0,+\}$. In order to avoid weak zerocrossings due to noise, a threshold is applied to the slope of zerocrossings. The slope of zerocrossing $\{a,-b\}$ is $|a+b|$. The algorithm for the LG operator is summarized in Figure 2.11. The results obtained by this operator are shown in Figure 2.14.

2.11 Properties of Gaussian

The Gaussian has nice properties such as scaling, separability, and symmetry which can be exploited in order to obtain an efficient implementation. In this section, we will discuss these properties.

2.11.1 Scaling

The Gaussian of standard deviation σ, when convolved with itself, yields a larger Gaussian of standard deviation $\sqrt{2}\sigma$. That is, if an image has been filtered with a Gaussian at a certain spread σ and if the same image must be filtered with a larger Gaussian with spread $\sqrt{2}\sigma$, then, instead of filtering the image with the larger Gaussian, the previous result can just be filtered with the same filter of spread σ to obtain the image filtered with $\sqrt{2}\sigma$. Thus, the total number of operations for filtering the image by Gaussian of σ and $\sqrt{2}\,\sigma$ will be equal to $2n\sigma$. However, without scaling, the number of operations will approximately be equal to $2.4\ n\ \sigma$. This produces a significant reduction in computations like the scale-space where operators of multiple sizes are applied to the same image.

The scaling property also holds in two dimensions. However, the second derivative of Gaussian in one dimension and the Laplacian of Gaussian in two dimensions do not possess this scaling property. It is possible to obtain the effect of applying bigger operators by repeatedly applying the smaller operators to the image. First apply the second derivative of the Gaussian operator of size σ to the image and then apply the Gaussian of size σ to the output. The result will be equivalent to the output obtained by applying the second derivative of the Gaussian operator of size $\sqrt{2}\sigma$. We state and prove the following propositions related to the scaling property of Gaussian.

Proposition 1 *Consider a Gaussian $g^{\sigma}(x)$ of standard deviation σ. The convolution of this Gaussian with itself yields another Gaussian, $g^{\sqrt{2}\sigma}$, with standard deviation $\sqrt{2}\sigma$.*

Proof:

$$
\begin{aligned}
g^{\sigma}(x) \; * \; g^{\sigma}(x) \; &= \; \int_{-\infty}^{\infty} e^{\frac{-\eta^2}{2\sigma^2}} \;\; e^{\frac{-(x-\eta)^2}{2\sigma^2}} \;\; d\eta && (2.35)\\
&= \int_{-\infty}^{\infty} e^{\frac{-(2\eta^2+x^2 \; -2\eta x)}{2\sigma^2}} d\eta && (2.36)\\
&= \int_{-\infty}^{\infty} e^{-\frac{(\sqrt{2}-\frac{x}{\sqrt{2}})^2}{2\sigma^2}} e^{\frac{-x^2}{4\sigma^2}} \; d\eta && (2.37)\\
&= \sqrt{\pi}\sigma e^{\frac{-x^2}{4\sigma^2}} && (2.38)\\
&= \sqrt{\pi}\sigma g^{\sqrt{2}\sigma}(x) \qquad \square && (2.39)
\end{aligned}
$$

Proposition 2 *Consider a second derivative of Gaussian $g^{a}_{xx}(x)$ of standard deviation a and a Gaussian $g^{b}(x)$ of standard deviation b. The convolution of these two functions results in a second derivative of Gaussian $\Delta^2 g^{\sqrt{a^2+b^2}}(x)$ of standard deviation $\sqrt{a^2+b^2}$.*

Proof:
By definition we have:

$$
\Delta^2 g^{a}(x) * g^{b}(x) = (1 - \frac{x^2}{a^2}) g^{a}(x) * g^{b}(x). \tag{2.40}
$$

Taking the Fourier transform of the right hand side we get:

$$
\begin{aligned}
(1 - \frac{x^2}{a^2}) g^{a}(x) * g^{b}(x) \quad &\longleftrightarrow \quad \sqrt{2\pi}(i\omega)^2 \exp^{(-\frac{\omega^2 a^2}{2})} . \sqrt{2\pi} \exp^{(-\frac{\omega^2 b^2}{2})}, && (2.41)\\
&\longleftrightarrow \quad \sqrt{2\pi}(\sqrt{2\pi}(i\omega)^2 \exp^{(-\frac{\omega^2 (\sqrt{a^2+b^2})^2}{2})}). && (2.42)
\end{aligned}
$$

Now, taking the inverse Fourier transform of the right hand side we get:

$$
\begin{aligned}
(1 - \frac{x^2}{a^2}) g^{a}(x) * g^{b}(x) \quad &= \quad \sqrt{2\pi}(1 - \frac{x^2}{(\sqrt{a^2+b^2})^2}) g^{\sqrt{a^2+b^2}}(x), && (2.43)\\
g^{a}_{xx}(x) * g^{b}(x) \quad &= \quad \sqrt{2\pi} g^{\sqrt{a^2+b^2}}_{xx}(x). \square && (2.44)
\end{aligned}
$$

When $a = b = \sigma$, the result will be the second derivative of Gaussian operator of spread $\sqrt{2}\sigma$.

Iterations	$a=1$, $b=1$	$a=1$,$b=2$	$a=1$, $b=3$	$a=2$, $b=3$
1	1.4	2.2	3.1	3.6
2	1.7	3.0	4.3	4.7
3	2.0	3.6	5.3	5.6
4	2.2	4.1	6.0	6.3
5	2.4	4.6	6.8	7.0
6	2.6	5.0	7.4	7.6
7	2.8	5.4	8.0	8.2
8	3.0	5.7	8.5	8.7
9	3.1	6.0	9.0	9.2
10	3.3	6.4	9.5	9.7
11	3.5	6.7	10.0	10.2

Figure 2.15: Successive Operator Sizes.

Iterations	$a=1$,$b=2.2$	$a=1$,$b=3.5$
1	2	5
2	3	6
3	4	7
4		8
5		9
6		10

Figure 2.16: Successive Operator Sizes.

In a particular implementation, values of a and b can be chosen so as to suit the required range of values of the operator. The value of b is important, since the successive operator sizes depend on it. In order to give the reader an idea about how the size of the operator increases depending on the initial values of a and b, we have tabulated a few cases in Figure 2.15.

In Figure 2.16, we show one of the possible schemes for achieving the operator sizes of the range $2, 3, 4, 5, ..., 10$. In this scheme, we have used two sets of initial values of a and b. The first four sizes of the operator are achieved by using the values from the second column, while the remaining sizes are obtained from the third column.

2.11.2 Separability

A two-dimensional Gaussian filter can be separated into two one-dimensional Gaussians, one along the x direction and the other along the y direction. Therefore, the Gaussian filter can be applied to an image by first convolving with a one-dimensional Gaussian along each row and then convolving the result again with a one-dimensional Gaussian along each column (as shown in Figure 2.18.b). Each one-dimensional convolution with an operator of m pixels requires m multiplications per pixel. Hence, two one-dimensional convolutions require $2m$ multiplications, which is a significant improvement over the m^2 multiplications needed for a two-dimensional convolution. Unfortunately, the LG operator is not separable into two single dimensional operators, because the Laplacian is not separable, even though the Gaussian is. We give an algorithm to decompose the two-dimensional LG convolution into four one-dimensional convolutions. This scheme requires $4m$ multiplications. For a large m, $4m$ multiplications are less than m^2 multiplications. Therefore, the number of multiplications is significantly reduced for larger images. The separability of LG is proved in the following proposition.

Proposition 3 *The Laplacian of Gaussian can be written as follows:*

$$\Delta^2 g(x,y) \;=\; \frac{\partial}{\partial y^2}\, g(y) \;*\; g(x) \;+\; g(y) \;*\; \frac{\partial}{\partial x^2}\, g(x). \tag{2.45}$$

Proof:

The above can be shown easily by using Fourier Transform theory. Let

$$g(x) \quad \longleftrightarrow \quad G(\omega),$$

denote a Fourier Transform pair. Since the two terms in equation 2.45 are symmetric in x and y, let us compute the first term only. The second term can be found from the first by replacing x with y and y with x. Assume that $G(\omega_1)$ and $G(\omega_2)$ are the Fourier transforms of Gaussians $g(x)$ and $g(y)$ respectively. By using the convolution property of the Fourier transform we can write:

$$g(x) \;*\; \frac{\partial}{\partial y^2}\, g(y) \;\longleftrightarrow\; G(\omega_1)\;.\;(-i\omega_2)^2\; G(\omega_2), \tag{2.46}$$

$$\longleftrightarrow\; G(\omega_1,\omega_2)\;.\;(-i\omega_2)^2, \tag{2.47}$$

$$\longleftrightarrow\; (-i\omega_2)^2\;.\;G(\omega_1,\omega_2), \tag{2.48}$$

where, $G(\omega_1,\omega_2)$ is the Fourier Transform of Gaussian $g(x,y)$. Now, by taking the inverse Fourier Transform of the right hand side we get:

$$\frac{\partial}{\partial y^2} g(x,y) \quad \longleftrightarrow \quad (-i\omega_2)^2\;.\;G(\omega_1,\omega_2).$$

By comparing pairs 2.48 and 2.11.2 we have,

$$g(x) \;*\; \frac{\partial}{\partial y^2} g(y) = \frac{\partial}{\partial y^2} g(x,y).$$

Similarly, by replacing x by y, and y by x in the above equation we have:

$$g(y) \;*\; \frac{\partial}{\partial x^2} g(x) = \frac{\partial}{\partial x^2} g(x,y).$$

Finally summing the above two equation we get:

$$g(x) \;*\; \frac{\partial}{\partial y^2} g(y) + g(y) \;*\; \frac{\partial}{\partial x^2} g(x) = \frac{\partial}{\partial y^2} g(x,y) + \frac{\partial}{\partial x^2} g(x,y) = \Delta^2 g(x,y).$$

Which is exactly equation 2.45. □

We can summarize the algorithm for the decomposition of the operator in Figure 2.17. Refer to Figure 2.18.a for a block diagram.

1. Convolve the image with a second derivative of Gaussian mask ($g_{yy}(y)$) along each column.
2. Convolve the resultant image from step (1) by a Gaussian mask ($g(x)$) along each row. Call the resultant image I^x.
3. Convolve the original image with a Gaussian mask ($g(y)$) along each column.
4. Convolve the resultant image from step (3) by a second derivative of Gaussian mask ($g_{xx}(x)$) along each row. Call the resultant image I^y.
5. Add I^x and I^y.

Figure 2.17: Decomposition of LG operator into four one dimensional convolutions.

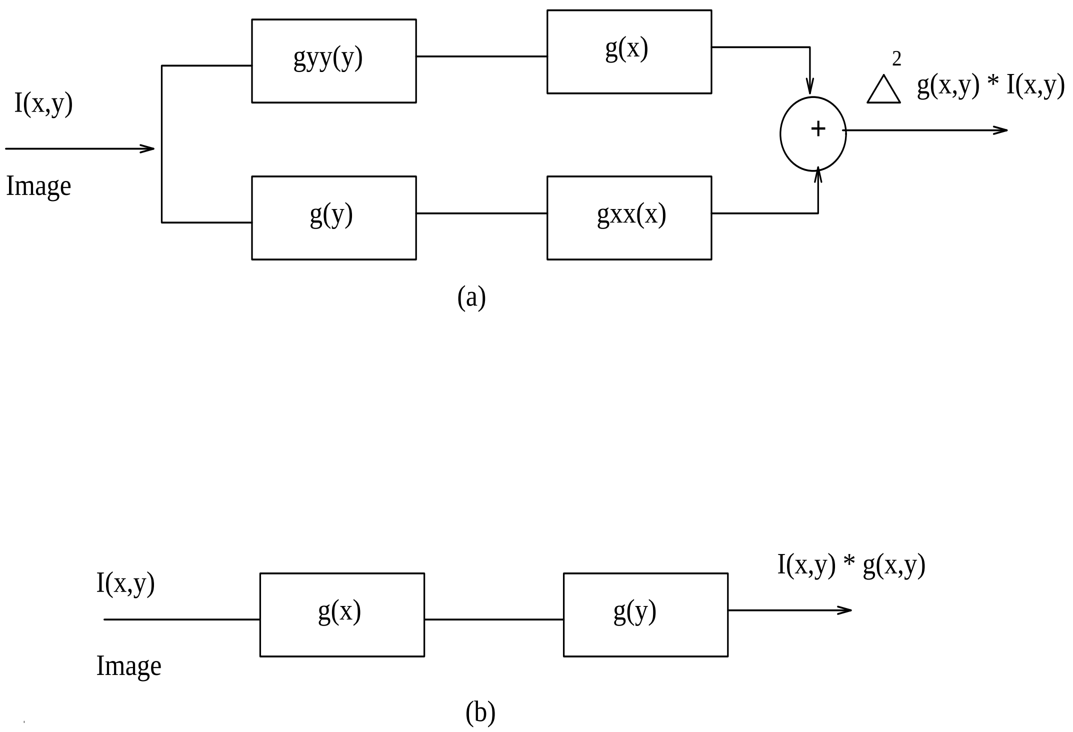


Figure 2.18: Separability. (a) 2-D convolution with LG can be decomposed into four 1-D convolutions. (b) 2-D convolution with Gaussian can be decomposed into two 1-D convolutions.

2.11.3 Symmetry

The Gaussian is symmetric around the origin, i.e., $g(x) = g(-x)$ for any x. This property can be used to reduce the number of multiplications as follows: Assume that the operator of size 5 is to be applied to the input sequence $X_1,\ X_2,\ X_3, X_4, X_5,\ X_6,\ ...., X_n$ in order to get the output sequence Y_4, Y_5, Y_6, ... For instance, the equation for the computation of Y_5 is

$$Y_5 \ = \ w_0X_1 \ + \ w_1X_2 \ + \ w_2X_3 \ + \ w_3X_4 \ + \ w_4X_5.$$

Due to the symmetry, $w_0 = w_4$ and $w_1 = w_3$. The convolution equation can be simplified as:

$$Y_5 \ = \ w_0(X_1 \ + \ X_5) + w_1(X_2 \ + \ X_4) + w_2X_3.$$

Utilizing symmetry, we reduce the number of multiplications from $5n$ to $3n$ to compute all the Y_i elements, where n is the total number of elements. In general, when using the property of symmetry, we only have to perform $(\#\ of\ weights/2\ +\ 1)\ *\ n$ multiplications which is a significant reduction from $(\#\ of\ weights\ )\ *\ n$.

2.12 Canny's Edge Detector

Canny [3, 2] has proposed an edge operator which is the sum of four complex exponentials and can be approximated by the first derivative of a Gaussian. He assumed that edge detection is performed by convolving the noisy image with a function $f(x)$ and by marking edges at the maxima in the output of this convolution. Canny specifies three performance criteria on the output of this operator.

1. Good detection. There should be a low probability of failing to mark edge points, and low probability of falsely marking non-edge points. Since both these probabilities are functions of the output signal to noise ratio, this criterion corresponds to maximizing signal to noise ratio.

2. Good localization. The points marked as edges by the operator should be as close as possible to the center of the true edge.

3. Only one response to a single edge. When two nearby operators respond to the same edge, one of them must be considered a false edge.

For the localization criteria Canny uses the inverse of the distance between the true edge and the edge marked by the operator. For the distance measure he uses the standard deviation in the position of the maximum of the operator output. He found out that (1) and (2) are conflicting and that there is a trade-off or uncertainty principle between them. Broad operators have a good signal-to-noise ratio but poor localization and vice-versa. Canny uses the product of two criteria to combine them in a meaningful way. He then finds $f(x)$, which maximizes this product.

Canny arrives at the solution for $f(x)$, which is the difference-of-boxes operator which was proposed by Rosenfeld and Thurston [21]. Next he uses the third criterion to eliminate the multiple responses. He adds the requirement that the function f will not have too many responses to a single step edge in the vicinity of the step. He limits the number of peaks in the response so that there will be a low probability of declaring more than one edge. He

1. Compute the gradient of image $f(x,y)$ by convolving it with the first derivative of Gaussian in x and y directions.

$$f_x(x,y) = f(x,y) * (\frac{-x}{\sigma^2})\, e^{\frac{-(x^2+y^2)}{2\sigma^2}}, \quad (2.49)$$

$$f_y(x,y) = f(x,y) * (\frac{-y}{\sigma^2})\, e^{\frac{-(x^2+y^2)}{2\sigma^2}}. \quad (2.50)$$

2. Perform non-maxima suppression on the gradient magnitude.

3. Apply hysteresis thresholding to the non-maxima suppressed magnitude. Scan the image from left-right, top-bottom. If the gradient magnitude at the pixel is above the high threshold, declare that as an edge point. Then recursively look at its neighbors (4 connected, or 8 connected). If the gradient magnitude is above the low threshold, declare that as an edge point.

Figure 2.19: Steps in Canny's edge detector.

tries to make the distance between peaks in the noise response approximate to the width of the response of the operator to a single step. This width is about the same as the operator width.

Besides the derivation of a "closed form" for an optimal edge detector, Canny has also used the Stochastic Optimization method to derive the optimal edge operator. The optimization algorithm is essentially a hill-climbing search over the space of possible filters. It proceeds by continuously iterating through the following steps:

1. Create a noisy edge by adding Gaussian random numbers to the sampled values of a step edge.

2. Convolve the filter with this edge, and evaluate the response.

3. Perturb the filter coefficients by a small amount.

4. Convolve this new filter with the edge, and evaluate the new response.

5. Change the filter based on the effects of the perturbation in (3).

The implementation of Canny's operator involves the following steps. After the image has been convolved with a symmetric Gaussian, the edge direction is estimated from the gradient of the smoothed image intensity surface. The gradient magnitude is then non-maxima suppressed in that direction. The algorithm sets thresholds based on local estimates of image noise. It makes use of two thresholds to deal with the problem of streaking.

The implementation of Canny's operator is shown in Figure 2.19. The results obtained with this operator are shown in Figure 2.20.

(a) (b)

Figure 2.20: (a). The bottle image. (b). The edge map from the Canny edge detector.

2.13 Scale Space

So far we have assumed the value of σ in the Gaussian based edge detectors is known. The important problem is how to determine the appropriate value of σ in a given image. The answer to this question is not straightforward. This has been a active area of research, and the problems remain to be unsolved in the general sense.

When Marr and Hildreth first proposed the LG operator, they suggested four different values of σ, which are $\sigma = 1, 2, 4, 8$. Their edge detector was motivated by the neurophysiological findings about the human vision. It has been reported that in human vision, four sizes of cells similar to LG operator exist. Their sizes can be approximated by the above four σ values. Each LG operator can be considered a band pass filter tuned to some frequency. The idea is to use four filters to cover the whole spectrum of frequencies.

As soon as the multiple values of σ are used in edge detection, the important question arises: how do you combine four edge maps? Marr and Hildreth put forward the *spatial coincidence assumption*, which states "The zerocrossings that coincide over several scales are physically significant." However, this criterion, which was never justified, runs into problem very easily.

In 1983, Witkin [25] wrote a very influential paper, in which he introduced scale space. Since then, scale space has been a hot topic in lower level computer vision. Witkin argued that since there is no rationale to select any finite number of operator sizes, the appropriate way is to employ a whole spectrum of operator sizes. When zerocrossings in the 1-D function detected at the continuum of scales are plotted in $x - \sigma$ space, they form contours. This $x - \sigma$ space was termed scale space. The interesting thing about scale space is that instead of treating a zerocrossing at one scale as a single point, now we can treat each edge point as a contour in the scale space. These contours in the scale space have some nice properties. For instance, the contours are always open at the bottom (low σ), and closed at the top (high

σ), like arches. Therefore, in going from the low scale (σ) to the high scale, the zerocrossing may disappear, but they are never created.

2.14 Exercises

1. Consider the image f given in the following. Assume that it is to be convolved with the mask m, and the result is to be put into image g.

f1	f2	f3	f4	f5	f6	f7
f8	f9	f10	f11	f12	f13	f14
f15	f16	f17	f18	f19	f20	f21
f22	f23	f24	f25	f26	f27	f28
f29	f30	f31	f32	f33	f34	f35
f36	f37	f38	f39	f40	f41	f42
f43	f44	f45	f46	f47	f48	f49

m1	m2	m3
m4	m5	m6
m7	m8	m9

g1	g2	g3	g4	g5	g6	g7
g8	g9	g10	g11	g12	g13	g14
g15	g16	g17	g18	g19	g20	g21
g22	g23	g24	g25	g26	g27	g28
g29	g30	g31	g32	g33	g34	g35
g36	g37	g38	g39	g40	g41	g42
g43	g44	g45	g46	g47	g48	g49

   Compute the following:

   $g9$, $g10$, $g32$, and $g42$.

2. Write a C code for convolving f with m. Hint: You can do it by four *do* loops.

3. Derive an expression for the Laplacian of Gaussian $\Delta^2 g$ given in equation 2.34.

4. Generate the mask for $255 \times \Delta^2 g$, when $\sigma = 1$. Truncate all mask values to the nearest integers.

5. Generate the mask for $255 \times \Delta^2 g$, when $\sigma = 1.5$. Truncate all mask values to the nearest integers.

6. Show that Laplacian $\Delta^2 f = f_{xx} + f_{yy}$ is also given by $\Delta^2 f = f''_\theta + f''_n$, where f''_θ is defined as second derivative of f in the direction θ, and f''_n is the second derivative in the direction perpendicular to θ.

7. Consider a bi-cubic polynomial defined in equation 2.13.

   (a) Derive equations 2.14 and 2.15.

   (b) Apply the substitution of variables $x = \rho \sin\theta$, $y = \rho\cos\theta$ in equation 2.13 to change $f(x, y)$ into $f_\theta(\rho)$, and show that $f_\theta(\rho)$ is given by equation 2.18.

8. Consider a second order polynomial:

$$f(x, y) = k_1 + k_2 x + k_3 y + k_4 x^2 + k_5 y^2 + k_6 xy.$$

Compute masks for $k_1, k_2, k_3, k_4, k_5, k_6$ using least squares fit. You can use a 3×3 window around each pixel in the image.

9. Show that gradient magnitude is rotation invariant. Let that gradient at point (x, y) is given by (f_x, f_y). Assume that the object is rotated around Z axis, the point (x, y) moves to point (x', y').

$$\sqrt{{f_x}^2 + {f_y}^2} = \sqrt{{f_{x'}}^2 + {f_{y'}}^2}$$

10. What is the computational complexity of Canny's edge detector and Laplacian of Gaussian edge detector. First identify main steps in each edge detector, then compute the complexity for each individual step.

11. Complete the following table related to three edge detectors we have discussed.

ITEM	**Laplacian of Gaussian**	**Canny's Operator**	**Hralick's Operator**
Filtering			
Differentiation			
Detection			
Complexity			
Comments			

Chapter 3

Region Segmentation

3.1 Introduction

In this chapter we will learn about the segmentation of images into regions [1]. This is a complementary approach to the edge detection methods we learned in the last chapter. In edge detection we segment an image by identifying the boundaries of objects. The boundaries are the locations where the intensity is changing. In the region based approach, we will identify regions occupied by the objects. Here, we will group pixels which are similar in some region property. The first part of this chapter deals with seed segmentation. In seed segmentation a crude segmentation of image is obtained based on the gray level distribution and connectivity of pixels. However, seed segmentation might result in too many small regions due to noise and other factors. In the second part of this chapter, we will discuss some methods for region growing. These methods will be used to merge small adjacent regions into larger ones that represent the objects more closely.

3.2 Definition of Segmentation

A segmentation of an image $f(x, y)$ is a partition of $f(x, y)$ (as shown in Figure 3.1) into sub images $R_1, R_2, \ldots, R_n$ such that the following constraints are satisfied:

1. $\bigcup_{i=1}^{n} R_i = f(x, y)$.

2. $R_i \cap R_j = \phi$, $i \neq j$.

3. Each subimage satisfies a predicate or set of predicates. Some examples are:

   - All pixels in any subimage R_i must have the same grey level.
   - All pixels in any subimage R_i must not differ by more than ΔX grey levels.
   - All pixels in any subimage R_i must not differ by more than ΔX from the mean grey level of the region.
   - The standard deviation of gray levels in any subimage R_i must be small.

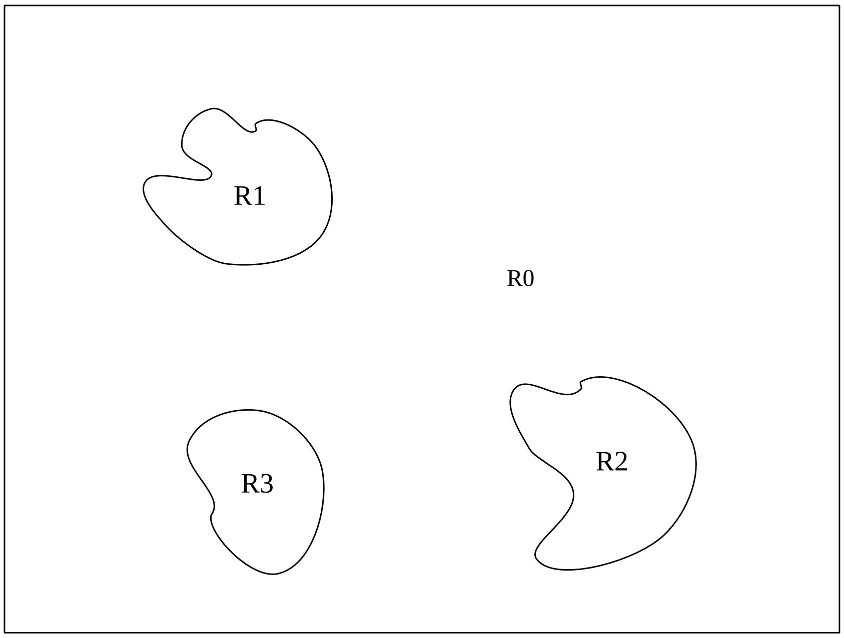


Figure 3.1: Image segmentation.

3.3 Simple Segmentation

In the simple cases where $f(x, y)$ contains one object, the grey level image can be converted into a corresponding binary image $B(x, y)$. In this binary image the object pixels will be 1's and the background pixels will be 0's. To determine a binary image we need to use some threshold T, as follows:

$$B(x, y) = \begin{cases} 1 & \text{if } f(x, y) < T \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Some variations of the above case include the use of two thresholds, T_1 and T_2; or even a range of thresholds, $Z = \{T_1, T_2, ..., T_k\}$.

$$B(x, y) = \begin{cases} 1 & \text{if } T_1 < f(x, y) < T_2 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

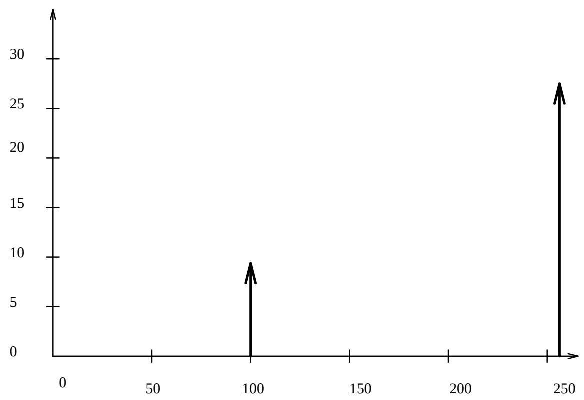
$$B(x, y) = \begin{cases} 1 & \text{if } f(x, y) \in Z \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

3.3.1 Thresholds and Histograms

The distribution of gray levels can be used to determine the threshold used in binary images. A histogram graphs the number of pixels in an image with a particular gray level as a

255	255	255	255	255	255
255	255	255	255	255	255
255	255	100	100	100	255
255	255	100	100	100	255
255	255	100	100	100	255
255	255	255	255	255	255

(a)

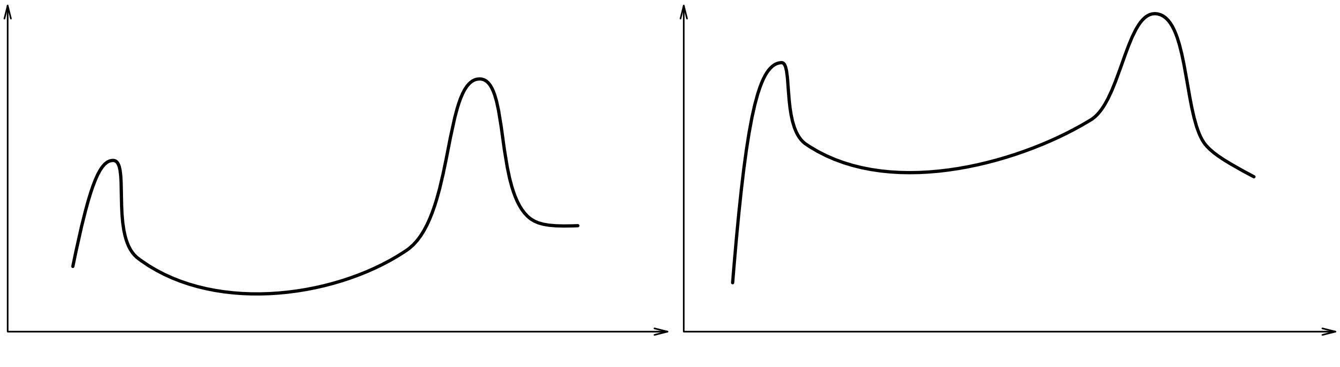


(b)

Figure 3.2: (a) Simple synthetic image with one object. (b) Histogram of image shown in (a).

function of the image of gray levels. For a synthetic image this histogram will contain distinct spikes. In Figure 3.2.a a simple 6×6 image with one object is shown. The pixel gray levels corresponding to the object are 100, and the pixels corresponding to the background are 255. The histogram of this image is shown in Figure 3.2.b. Because this histogram has two spikes, it is called bimodal. The histogram of a real image may not contain clear spikes, instead it may consist of peaks and valleys as shown in Figure 3.3. Real images rarely contain step edges. The gray levels at the edges changes gradually from background to foreground. This results in peaks and valleys in the histogram.

Objects with approximately the same range of gray levels form a class. The histogram of an image will have a peak for each class of objects and one large peak corresponding to the background. To distinguish between k distinct classes of objects, we must choose k thresholds, each lying between two peaks. Figure 3.4.a shows an image consisting of three

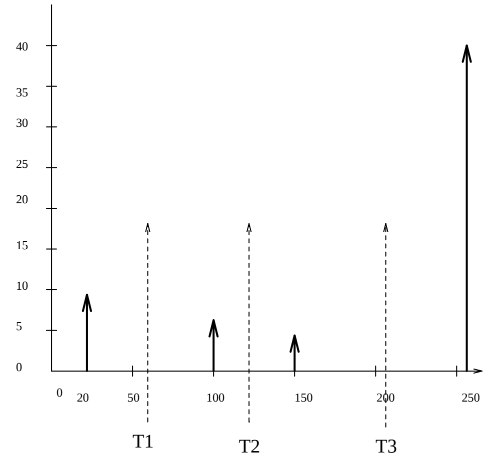


(a) (b)

Figure 3.3: (a)-(b) Examples of bimodal histograms.

255	255	255	255	255	255	255	20
255	255	255	100	100	255	20	20
255	255	255	100	100	255	20	20
255	255	255	100	100	255	20	20
255	255	255	255	255	255	20	20
255	255	255	255	255	255	255	255
150	150	255	255	255	255	255	255
150	150	255	255	255	255	255	255

(a)



(b)

Figure 3.4: (a) Image consisting of three objects. (b) Histogram of image shown in (a).

objects, and the corresponding histogram is shown in Figure 3.4.b. There are four spikes, three corresponding to objects, and one corresponding to the background. Three binary images can be generated using thresholds as follows:

$$B_1(x,y) = \begin{cases} 1 & \text{if } 0 < f(x,y) < T_1 \\ 0 & \text{otherwise} \end{cases}$$

$$B_2(x,y) = \begin{cases} 1 & \text{if } T_1 < f(x,y) < T_2 \\ 0 & \text{otherwise} \end{cases}$$

$$B_3(x,y) = \begin{cases} 1 & \text{if } T_2 < f(x,y) < T_3 \\ 0 & \text{otherwise} \end{cases}$$

One important thing to note about the histogram is that it can only distinguish between classes of objects. It does not contain any spatial information. Therefore a half black and half white image, chess board image with alternate black and white blocks of equal number, and image of random black and white dots of equal distribution have the same histogram.

3.3.2 Peakiness Test

Histograms of a real image may contain small peaks due to noise. Therefore, not all the peaks can be used in segmentation. Before histogram peaks can be used in segmentation, we need to determine genuine peaks which correspond to the object regions. We will use the peakiness test for that purpose. A peak is good peak if it is sharp and deep. A peak is sharp if the area under it is small. We will use the ratio of area of a rectangle enclosing the

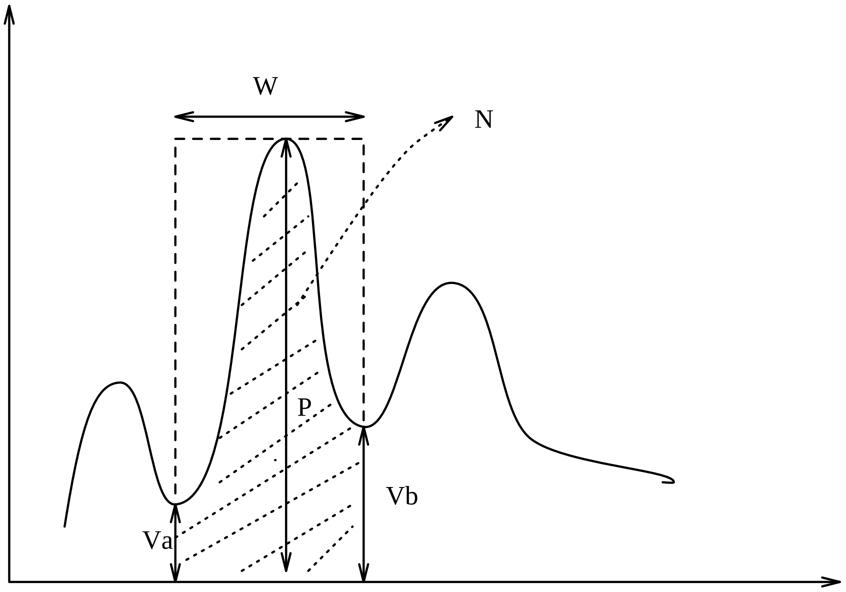


Figure 3.5: Peakiness test. V_a, V_b are the heights of valleys, W is width of peak, P is the height of peak, N is the number of pixels between V_a and V_b.

peak to the number of pixels N under a peak as the sharpness of peak (as shown in Figure 3.5). The depth of peak is the relative height of peak. In this test we will use the following information about a peak:

1. W = width of the peak in grey level range from valley to valley.
2. P = height of the peak.
3. V_a, V_b = the two valley points to each side of peak.
4. N = the number of pixels in the image covered by the peak.

The sharpness of a peak is defined as the ratio $\frac{N}{(W \times P)}$. If this ratio is 1, then we have the worst possible case; a rectangle. The smaller this ratio, the sharper the peak. The ratio of the height of the valleys to the height of the peak is $\frac{(V_a+V_b)}{2P}$. The actual peakiness test will be the product of these ratios:

$$Peakiness = \left(1 - \frac{(V_a + V_b)}{2P}\right) \times \left(1 - \frac{N}{(W \times P)}\right).$$

If the peakiness is greater than some threshold, then that peak will be used for segmentation.

3.4 Connected Component Algorithms

So far we have only used the gray level information for segmentation. We have not used any spatial information of regions. As noted earlier, histograms do not give any spatial information. In order to find a connected group of pixels in an image we need to apply

	4	
4	*	4
	4	

(a)

8	8	8
8	*	8
8	8	8

(b)

6	6	
6	*	6
	6	6

(c)

Figure 3.6: Pixel connectedness. (a) 4-connected. (b) 8-connected. (c) 6-connected.

1. Scan the binary image left to right, top to bottom.
2. If there is an unlabeled pixel with a value of '1' assign a new label to it.
3. Recursively check the neighbors of the pixel in step 2 and assign the same label if they are unlabeled with a value of '1'.
4. Stop when all the pixels of value '1' have been labeled.

Figure 3.7: Recursive Connected Component Algorithm.

a connected component algorithm. There are three possible degrees of connectedness, as shown in Figure 3.6. In four connectedness, a pixel is considered to be connected to four of its immediate neighbors (left, right, up, and down), because these pixels are at a distance one. In 8-connectedness the diagonal elements are also considered. Therefore, a pixel is considered to be connected to all its eight neighbors as shown in Figure 3.6.b. In the discrete domain it is not appropriate to use the Euclidean distance. The Euclidean distance between the center pixel and diagonal pixel is $\sqrt{2}$. In images, only integer pixels are valid. Therefore, other distancess, like the chessboard distance, are used. The chessboard distance is defined as:

$$D_{ch}((x_1, y_1)(x_2, y_2)) = max(|x_1 - x_2|, |y_1 - y_2|). \tag{3.4}$$

According to this distance all eight neighbors of the central pixel are at distance 1. Therefore they are considered connected. We can also use 6-connected as shown in Figure 3.6.c. In 6-connected any two diagonal elements are considered connected.

We will describe in the next two subsections two algorithms: recursive and sequential for finding connected components in an image.

3.4.1 Recursive Algorithm

The recursive algorithm given in Figure 3.7 works well and is easy to implement. The problem is that it is recursive. This algorithm, when run on a small computer with a limited stack, may easily run into stack overflow, which would have to be taken care of by the programmer. For a 512×512 image with a quarter million pixels, the recursive algorithm may have several thousand recursive calls.

1. Scan the binary image left to right, top to bottom.
2. If an unlabeled pixel has a value of '1', assign a new label to it according to the following rules:

$$\begin{array}{cc} & 0 \\ 0 & 1 \end{array} \rightarrow \begin{array}{cc} & 0 \\ 0 & L \end{array} \qquad\qquad \begin{array}{cc} & 0 \\ L & 1 \end{array} \rightarrow \begin{array}{cc} & 0 \\ L & L \end{array}$$

$$\begin{array}{cc} & L \\ 0 & 1 \end{array} \rightarrow \begin{array}{cc} & L \\ 0 & L \end{array} \qquad\qquad \begin{array}{cc} & L \\ M & 1 \end{array} \rightarrow \begin{array}{cc} & L \\ M & L \end{array} \text{ (Set } L = M\text{).}$$

3. Determine equivalence classes of labels.
4. In the second pass, assign the same label to all elements in an equivalence class.

Figure 3.8: Sequential Connected Component Algorithm.

1. Compute the histogram of a given image.
2. Smooth the histogram by averaging to remove small peaks.
3. Identify candidate peaks and valleys in the histogram.
4. Detect good peaks by applying peakiness test.
5. Segment the image using thresholds at the valleys.
6. Apply connected component algorithm.

Figure 3.9: Steps in segmentation using histogram.

3.4.2 Sequential Algorithm

The sequential algorithm is a two pass algorithm which labels the regions according to specific patterns (see Figure 3.8). The first pass scans the binary image and assigns any unlabeled pixel a new label. In the assignment of these labels the labels of neighboring pixels are considered. During the second pass the labels of pixels are changed to the labels of their equivalence class. Assume that at the end of the first pass we ended up with labels 'a' through 'l', and the following labels were determined as to be equal: (a, b), (l, k), (c, f), (a, g), (b, e), (j, l), (h, f), and (i, k). Then we will have the following equivalence classes: $(a, b, e, g), (c, f, h), (i, j)$. During the second pass the labels b, e, and g will be changed to a. Similarly the labels f and h will be changed to c, and label j will be changed to i.

3.5 Seed Segmentation

Now we can summarize various steps in the seed segmentation algorithm as shown in Figure 3.9. In the first step the histogram of an image is computed. The histogram may contain some small peaks due to noise and uneven illumination. During the second step the histogram is smoothed to remove these small peaks. In principle, any of the filters discussed in the last chapter can be used here. In most cases, simple averaging over 3 elements of the histogram works reasonably well. Sometimes, more noisy histograms can be smoothed by smoothing several times. In the third step, candidate peaks and valleys are identified simply by detecting local maxima and minima in the histogram. Next the good peaks are detected by using the peakiness test. In the fifth step the image is segmented using thresholds at the valleys between peaks. The last step is used to determine a set of connected components in the image. The results for region segmentation are shown in Figure 3.10.

3.6 Region Growing

Segmentation using gray level distribution and connectivity of pixels can be considered seed segmentation. This segmentation needs to be refined using information about the shape of the regions and the semantics of regions. This segmentation might also yield too many small regions, which need to be merged with neighboring regions. We will discuss several algorithms for region growing.

3.6.1 Split and Merge Algorithm

This is the most simple algorithm for region growing. In this algorithm the regions are sequentially split and merged using some predicate until nothing can be merged or split (as shown in Figure 3.11). The algorithm is outlined in Figure 3.13. This algorithm can be demonstrated by a simple image shown in Figure 3.12. In this example, we will use a very simple predicate: *the gray levels of pixels in a region are the same*. To start with, the whole image is considered as one region. Since all the pixels in the image do not have the same gray levels, the image is split into 4 quadrants, as shown in Figure 3.12.a. Next, an attempt is made to merge any two adjacent regions. But none of the quadrants satisfy the predicate, therefore there is no merge. Next, all three quadrants except the upper left quadrant are splitted into four smaller regions each (see Figure 3.12.b). At this stage three sub-quadrants can be merged into one large region. Two sub-quadrants at the lower part of the image are further split as shown in Figure 3.12.c. Finally, during the next merge operation the remaining sub-quadrants are merged with the large region. At this stage all regions satisfy the predicate; and no further merging or splitting is done.

3.6.2 Phagocyte Algorithm

This algorithm is also called boundary melting. The idea is to remove (melt) the weak boundaries between two adjacent regions, hence merging the two regions. The phagocyte heuristic is used for the boundary between the two regions. In this algorithm, the crack edges between two regions as shown in Figure 3.15.a are used. The edges can easily be represented using the super grid $((2n+1)(2n+1))$ as shown in Figure 3.15.b. The strength of the edge between point A in region R_1 and point B in region R_2 is given by the absolute difference of gray levels:

$$S(A,B) = |f(x_1,y_1) - f(x_2,y_2)|$$

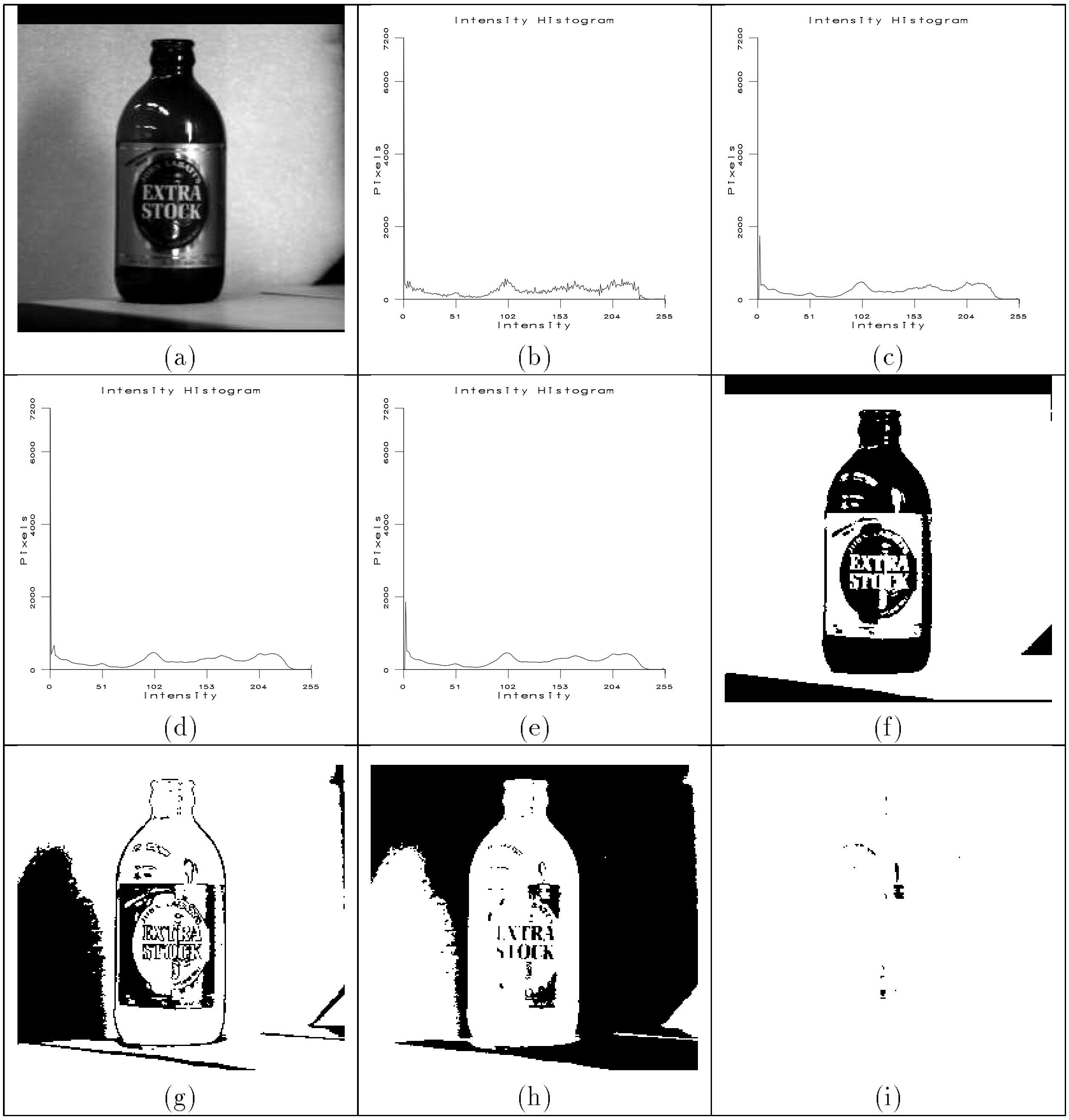


Figure 3.10: Histogram smoothing process using peakiness threshold = 0.1, ave size = 5. (a) Original image. (b) Histogram for image in (a), peaks = 93, peaks after peakiness test = 18. (c) Histogram after one smoothing, peaks = 54, peaks after peakiness test = 7. (d) Histogram after two smoothings, peaks = 21, peaks after peakiness test = 7. (e) Histogram after three smoothings, peaks = 11, peaks after peakiness test = 4. (f) Regions from peak 1 (0..40). (g) Regions from peak 2 (40..116). (h) Regions from peak 3 (116..243). (i) Regions from peak 4 (243..255).

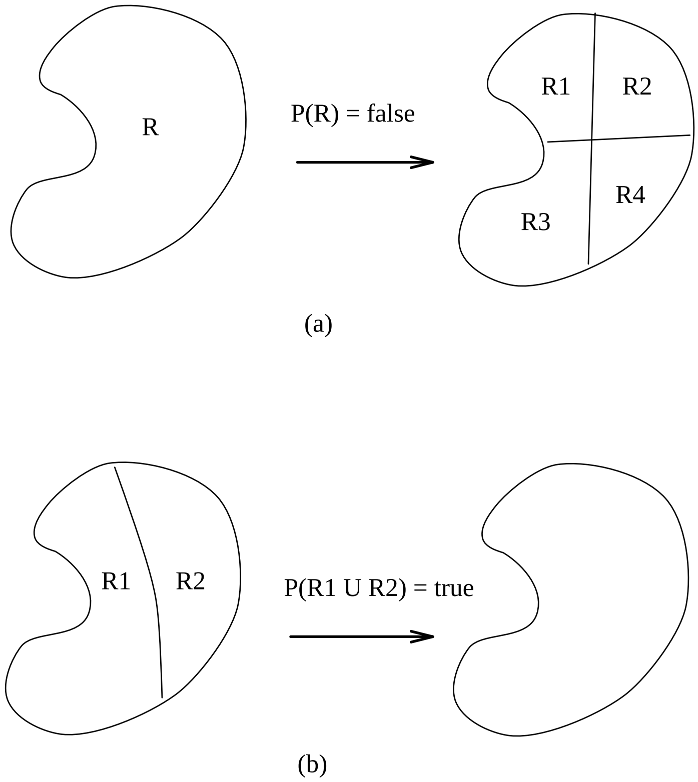


Figure 3.11: Split and merge. (a) If $P(R) = false$, a region R is split into four subregions R_1, R_2, R_3, and R_4. (b) If $P(R_1 \cup R_2) = true$, two regions, R_1 and R_2, are merged.

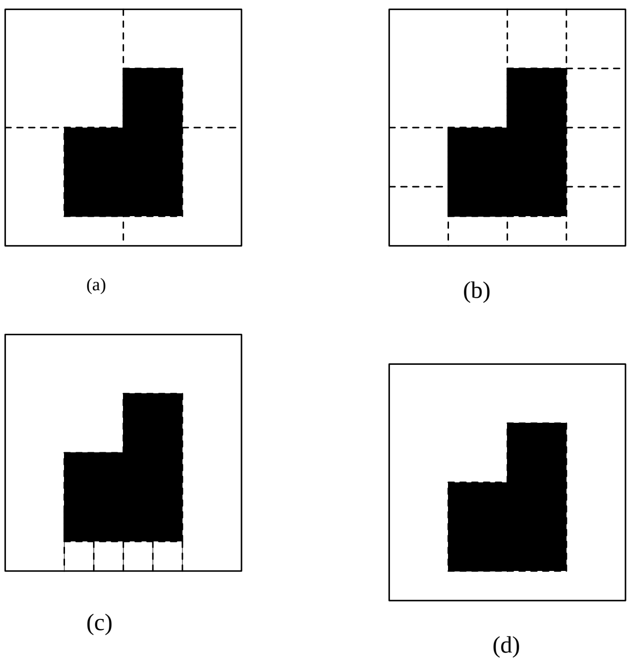


Figure 3.12: Demonstration of split and merge algorithm.

1. Split region R into four adjacent regions (quadrants) if $Predicate(R) = false$.
2. Merge any two adjacent regions R_1 and R_2 if $Predicate(R_1 \cup R_2) = true$.
3. Stop when no further merging and splitting are possible.

Figure 3.13: Split and Merge Algorithm.

1. Merge two regions if
$$\frac{W(\text{Boundary})}{min(P_1, P_2)} > T_2 \quad 0 \leq T_2 \leq 1$$
where P_1 is the perimeter of region R_1, and P_2 is the perimeter of region R_2.
2. Merge regions if
$$\frac{W(\text{Boundary})}{\text{Total number of points on the border}} > T_3 \quad 0 < T_3 < 1$$

Figure 3.14: Phagocyte Algorithm.

The edge is declared weak if the $S(A, B)$ is below some threshold.

$$W(A, B) = \begin{cases} 1 & \text{if } S(A, B) < T_1 \\ 0 & \text{otherwise} \end{cases}$$

The weakness of the whole boundary is the sum of weaknesses of all boundary points.

$$W(Boundary) = \sum_{\forall A,B} W(A, B)$$

The algorithm is outlined in Figure 3.14. Two heuristics are used here: *weakness* and *phagocyte*. The weakness heuristic is straightforward. If the ratio of weak boundary points to the total number of boundary points between any two adjacent regions is above some threshold, these two regions are merged. However, this heuristic tends to overmerge regions. Therefore, we need a phagocyte heuristic, which takes into account the shape of the resultant region. The results for region segmentation are shown in Figure 3.16.

3.6.3 Likelihood Ratio Test

In this method we use gray level distribution of regions to decide if they should be merged. This method is probabilistic in nature. We consider two hypothesis H_1 and H_2.
H_1: There are two regions.
H_2: There is one region.

We will assume that in each region, the gray level distribution can be approximated by a Gaussian distribution. That is, if a pixel is selected at random, the probability that it has gray level X is given by

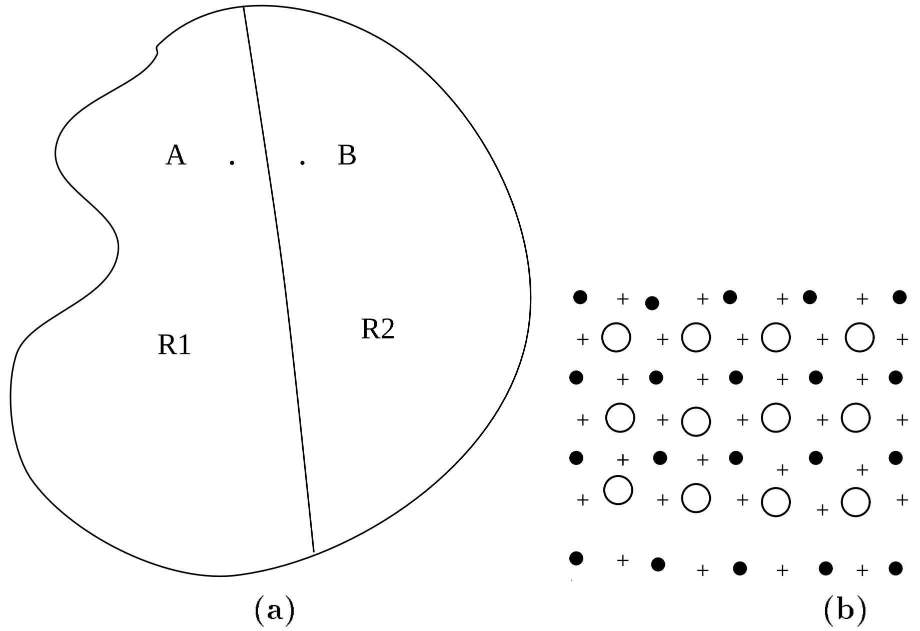


Figure 3.15: Region merging using Phagocyte algorithm. (a) Two regions R_1 and R_2. (b) Super grid $((2n+1) \times (2n+1))$.

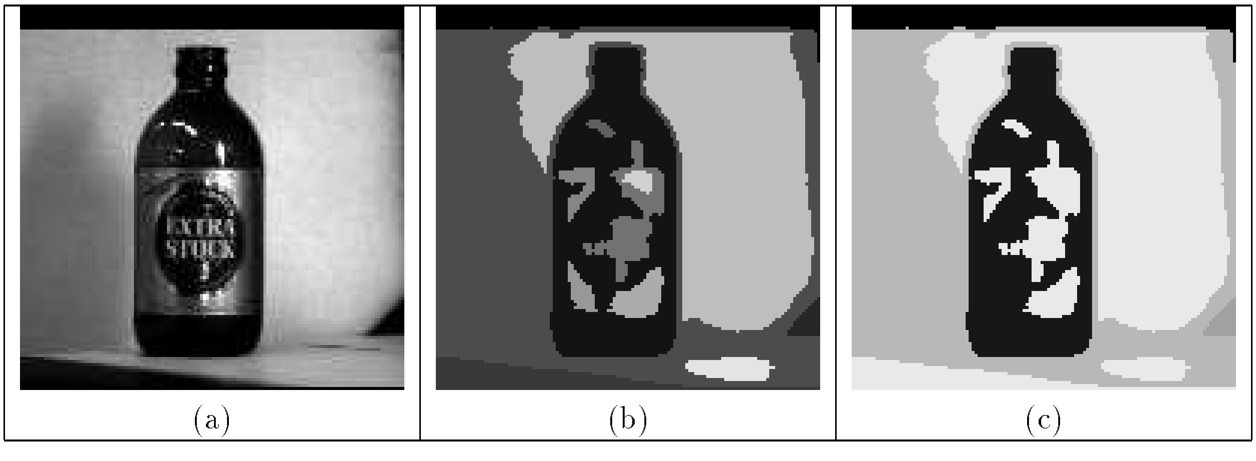


Figure 3.16: (a). The bottle image. (b). Seed Segmentation. (c). After region merging.

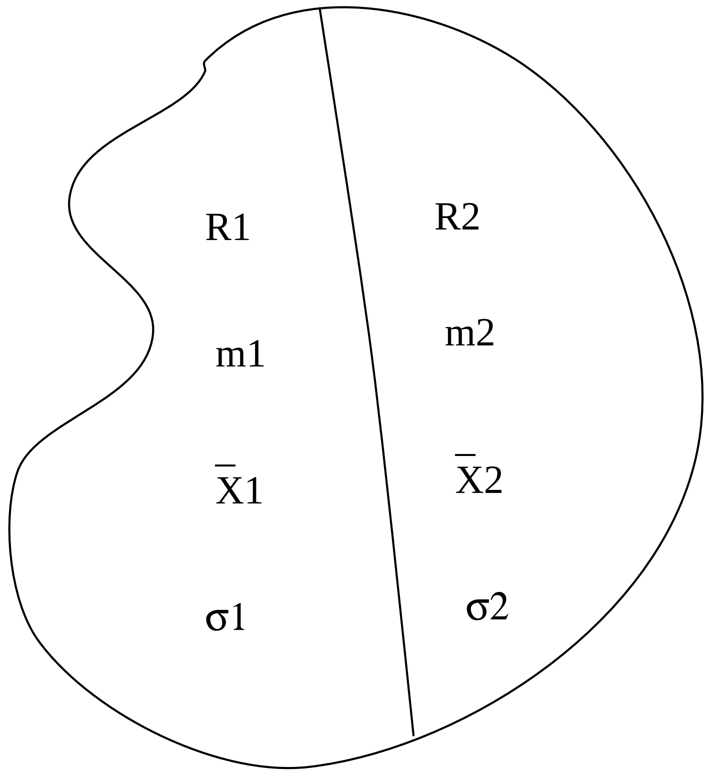


Figure 3.17: Region merging using Likelihood ratio algorithm.

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\bar{x})^2}{2\sigma^2}},$$

where $\bar{x}$ is the mean gray level and σ is the standard deviation.

Assume that region R_1 contains m_1 pixels, and $\bar{x}_1$ and σ_1 are respectively the mean and standard deviation of region R_1 as shown in Figure 3.17. Also assume that the gray levels are independent of each other. Then the probability of region R_1 having gray levels $(x_1, x_2, \ldots, x_{m_1})$ is given by:

$$P(x_1, x_2, \ldots, x_{m_1}) = (\frac{1}{\sqrt{2\pi}\sigma_1})^{m_1} e^{-\frac{m_1}{2}}. \tag{3.5}$$

Similarly, we can compute the probabilities for region R_2, and region $R = R_1 \cup R_2$ as follows:

$$P(x_{m_1+1}, x_{m_1+2}, \ldots, x_{m_1+m_2}) = (\frac{1}{\sqrt{2\pi}\sigma_2})^{m_2} e^{-\frac{m_2}{2}} \tag{3.6}$$

$$P(x_1, x_2, \ldots, x_{m_1}, \ldots, x_{m_1+1}, x_{m_1+2}, \ldots, x_{m_1+m_2}) = (\frac{1}{\sqrt{2\pi}\sigma_0})^{m_1+m_2} e^{-\frac{m_1+m_2}{2}} \tag{3.7}$$

Now, the probabilities for H_1 and H_2 are given as:

$$P(H_1) = P(x_1, x_2, \ldots, x_{m_1}) \times P(x_{m_1+1}, x_{m_1+2}, \ldots, x_{m_1+m_2}) = (\frac{1}{\sqrt{2\pi}\sigma_1})^{m_1} e^{-\frac{m_1}{2}} \times (\frac{1}{\sqrt{2\pi}\sigma_2})^{m_2} e^{-\frac{m_2}{2}} \quad (3.8)$$

$$P(H_2) = P(x_1, x_2, \ldots, x_{m_1}, \ldots, x_{m_1+1}, x_{m_1+2}, \ldots, x_{m_1+m_2}) = (\frac{1}{\sqrt{2\pi}\sigma_0})^{m_1+m_2} e^{-\frac{m_1+m_2}{2}} \quad (3.9)$$

The Likelihood ratio is:

$$LH = \frac{P(H_1)}{P(H_2)} = \frac{(\sigma_0)^{m_1+m_2}}{(\sigma_1)^{m_1}(\sigma_2)^{m_2}}$$

The regions are merged if $LH < T$.
are

3.7 Region Adjacency Graph

Region adjacency graphs (RAGs) are very useful data structures for region merging algorithms. In this graph, regions are represented as nodes and adjacent regions are connected by an arc between the nodes. Consider a simple image consisting of five regions $1, 2, 3, 4$, and 5 as shown in Figure 3.18.a. The pixels corresponding to a region are denoted by its region numbers. Assume that some predicate is used to merge regions. First, regions 4 and 3 are merged. The image with updated regions is shown in Figure 3.18.b. Next region 3 is merged with region 2 as shown in Figure 3.18.c. Finally, region 2 is merged with region 1. In this process, the region labels are changed multiple times. For instance he label of pixels belonging to region 4 is first changed to 3 then to 2, and finally to 1. In this example the image is very small, and each region consists of few pixels. But, it is not uncommon to have thousands of pixels in each region. In this case, the changing of labels require more overhead. Therefore, instead of changing region labels of each pixel during the merging, we will apply the merge operation on a RAG. The RAGs corresponding to Figure 3.18.a-d are shown in Figure 3.18.e-h. When region 4 is merged with the region 3, the node 4 in the RAG is removed, and the arcs are adjusted to reflect the new image as shown in Figure 3.18.f. It is also recorded that the pixel labels 3 and 4 are equivalent. The merge operations for the image shown in Figure 3.18.a are shown in 3.18.h with the equivalent labels and the initial image shown in Figure 3.18.a can be used to get the final segmented image similar to one shown in Figure 3.18.d.

3.8 Issues in Region Growing

1. The number of thresholds used in the algorithm. Some algorithms use one or two thresholds, but others use several thresholds.

2. The order of merging is very important. The final segmentation will depend heavily on the order in which regions are merged.

3. Seed segmentation is also important. If the segmentation to start with is reasonable, the region growing will improve it further by using the shape and semantic information

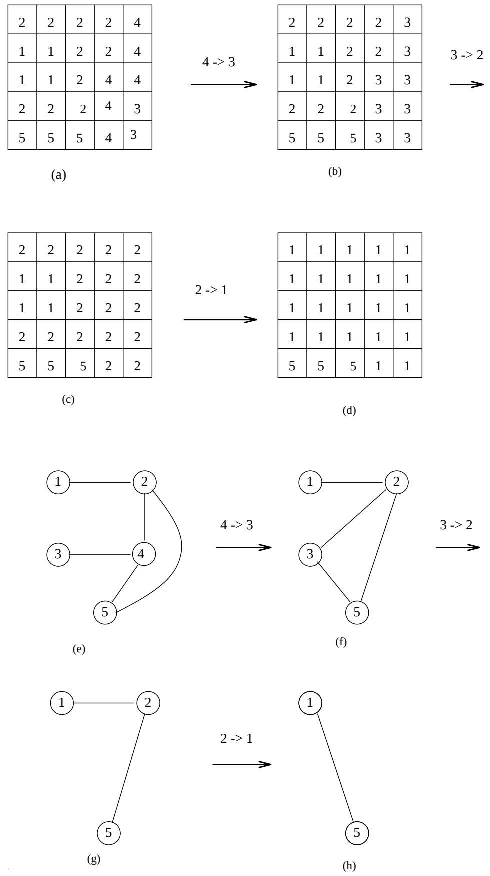


Figure 3.18: Region adjacency graphs (RAGs).

of regions. However, if the initial segmentation is very poor, the region growing will not improve it.

As we have said, edge detection and region segmentation are complementary processes. In principle, we can determine regions from the edges of the object, and edges from regions. However, there are several differences between these two types of segmentation.

1. Region segmentation results in closed boundaries, while the boundaries obtained by edge detection are not necessarily closed.
2. Edge detection is mostly local. Region segmentation is more global.
3. Region segmentation can be improved by using multi-spectral images (e.g. color images), however there is not much an advantage in using multi-spectral images in edge detection.
4. The position of a boundary (edges) is localized in edge detection, but not necessarily in region segmentation.

3.9 Geometrical Properties of Rgions

Once the scene is segmented into regions, we can determine the geometrical propertiese of regions which can be used in the object recognition. In this section, we will describe few important properties. We will assume that the region is represented by a $m \times n$ binary image, $B(x,y)$.

Area: Area or size of the regions is a total number of pixels occupied by the region. It is given by:

$$A = \sum_{x=0}^{m} \sum_{y=0}^{n} B(x,y) \tag{3.10}$$

Centroid: The centriod is like the center of an arbritrary shaped region, and it can be used to represent the location of a region. Centroid, $(\bar{x}, \bar{y})$ is given by:

$$\bar{x} = \frac{\sum_{x=0}^{m} \sum_{y=0}^{n} xB(x,y)}{A}, \tag{3.11}$$

$$\bar{y} = \frac{\sum_{x=0}^{m} \sum_{y=0}^{n} yB(x,y)}{A}. \tag{3.12}$$

Moments: The first moments, M_x^1, M_y^1, and the second moments, M_x^2, M_y^2, are respectively given as:

$$M_x^1 = \sum_{x=0}^{m} \sum_{y=0}^{n} xB(x,y) \tag{3.13}$$

$$M_y^1 = \sum_{x=0}^{m} \sum_{y=0}^{n} yB(x,y) \tag{3.14}$$

$$M_x^2 = \sum_{x=0}^{m}\sum_{y=0}^{n} x^2 B(x,y) \tag{3.15}$$

$$M_y^2 = \sum_{x=0}^{m}\sum_{y=0}^{n} y^2 B(x,y) \tag{3.16}$$

Perimeter: The perimeter of a region is sum of its border pixels. A pixel which has at least one pixel in its neighborhood from the background is called a border pixel.

Compactness Compactness, C, is defined as follows:

$$C = 4\pi \frac{A}{P^2}, \tag{3.17}$$

where A is the area of a region, and P is the perimeter of the region. The most compact region is a circle, with compactness equal to 1 (The area of a circle of radius R is πR^2 and the perimeter is $2\pi R$). The compactness of a square shape is $\frac{\pi}{4}$.

Orientation: The orientation, θ, of a region can be computed by determining the axis of second moment of inertia (see Figure 3.19). It can be determined by minimizing the following expression:

$$E = \int\int (x \sin\theta - y\cos\theta + \rho)^2 B(x,y)dxdy \tag{3.18}$$

It can be shown (see Exercises) that minimzing above expression results into:

$$\sin 2\theta = \pm \frac{b}{\sqrt{b^2 + (a-c)^2}}, \tag{3.19}$$

$$\cos 2\theta = \pm \frac{a-c}{\sqrt{b^2 + (a-c)^2}}, \tag{3.20}$$

where,

$$a = \int\int x'^2 B(x,y)dx'dy', \tag{3.21}$$

$$b = 2\int\int x'y' B(x,y)dx'dy', \tag{3.22}$$

$$c = \int\int y'^2 B(x,y)dx'dy', \tag{3.23}$$

$x' = x - \bar{x}$, and $y' = y - \bar{y}$. For the discrete domain, a, b, and c are given as follows:

$$a = \sum\sum x^2 B(x,y) - A\bar{x}^2, \tag{3.24}$$

$$b = 2\sum\sum xyB(x,y) - A\bar{x}\bar{y}, \tag{3.25}$$

$$c = \sum\sum y^2 B(x,y) - A\bar{y}^2. \tag{3.26}$$

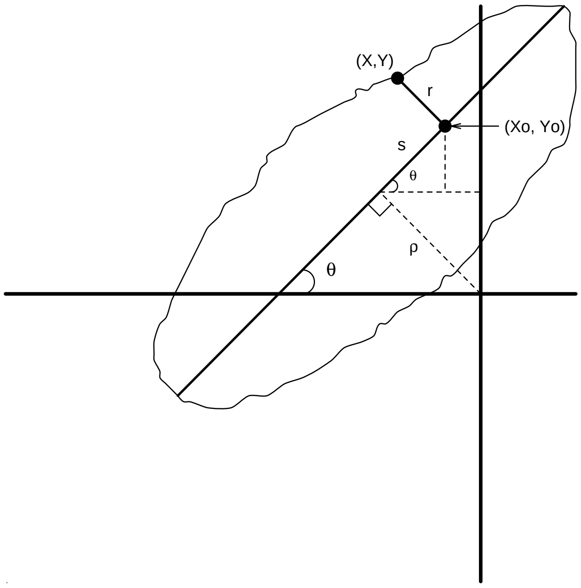


Figure 3.19: Determining orientation of an arbitrary region.

3.10 Exercises

1. Outline the *sequential* algorithm using *eight connectedness.*

2. Devise algorithm for determining equivalence classes.

3. Derive equations: 3.5, 3.6, and 3.7.

4. Determine the computational complexity of sequential and recursive connected component algorithms.

5. Show that the equation of a line shown in Figure 3.19 is given by

$$x \sin\theta - y \cos\theta + \rho = 0. \tag{3.27}$$

Verify that the closest point on the line to the origin is at $(-\rho \sin\theta, +\rho\cos\theta)$, and the parametric equation for a point (x_0, y_0) on the line is given by:

$$x_0 = -\rho \sin\theta + s\cos\theta, \tag{3.28}$$

$$y_0 = +\rho\cos\theta + s\sin\theta, \tag{3.29}$$

where s is as shown in Figure 3.19.

6. The distance, r, between a point (x, y) and the line is given by

$$r^2 = (x - x_0)^2 + (y - y_0)^2. \tag{3.30}$$

Substitute x_0 and y_0 from equations 3.28 and 3.29 in the above equation, and differntiate the resultant equation with respect to s, and set the equation to zero. Show that now s is given by:

$$s = x\cos\theta + y\sin\theta. \tag{3.31}$$

7. Substitute 3.31 in equations 3.28 and 3.29, and show that

$$r^2 = (x \sin\theta - y \cos\theta + \rho)^2.$$

8. Show that differntiating equation 3.18 with respect to ρ and equating the result to zero results:

$$A(\bar{x} \sin\theta - \bar{y} \cos\theta + \rho) = 0, \tag{3.32}$$

where $(\bar{x}, \bar{y})$ is the centriod, A is area.

9. Using $x' = x - \bar{x}$, and $y' = y - \bar{y}$, show that E in equation 3.18 is now given by:

$$E = a \sin^2\theta - b \sin\theta \cos\theta + c \cos^2\theta, \tag{3.33}$$

where a, b, and c are as defined in equations 3.21–3.23. And show that E can also be writtens as

$$E = \frac{1}{2}(a + c) - \frac{1}{2}(a - c) \cos 2\theta - \frac{1}{2} b \sin 2\theta. \tag{3.34}$$

10. By differntiating equation 3.34 with respect to θ and equating the result to zero, show that

$$\tan 2\theta = \frac{b}{a - c}, \tag{3.35}$$

hence, verify equations 3.19 and 3.20.

Chapter 4

2-D Shape

4.1 Introduction

In this chapter we will discuss several techniques for representing the shape of boundaries and regions in a segmented image [1]. These representations are the abstraction of edges and regions in a symbolic form, which will be useful in object recognition. Here, we will mainly be dealing with grouping operations. For instance, the grouping of edge points into straight lines, circles, ellipses, etc. For representing regions we will discuss the medial axis transform, pyramids, and quad trees. For representing boundaries we will discuss the Hough transform, generalized Hough transform, chain code, and shape number. Good shape representation should be compact, complete, unambiguous, and stable.

4.2 Hough Transform

The Hough transform can be used to represent plane curves, e.g., lines, circles, and parabolas, defined by analytical expressions. There are connections between the Hough transform and the least squares fit. In the least squares method, an over-constraint system of equations is used to compute the solution. Therefore, the number of equations is more than the number of unknowns. In the Hough transform, an under-constraint system is used to compute the solution. Therefore, the number of equations is less than the number of unknowns. There are multiple solutions in this case. In fact, in many cases, in Hough transform a single constraint is used to compute all possible solutions. That way each constraint votes for a set of possible solutions. The solution which has the majority of votes is selected. There is a third possibility, in which the number of constraints is equal to the number of unknowns. This method is called the RANSAC (Random Sampling and Consensus) method. The main steps in RANSAC are the following:

1. Randomly select the minimum number of constraints to estimate the solution.

2. Find the error between the estimated solution and all data points. If the error is less than the tolerance, then quit, else go to (1).

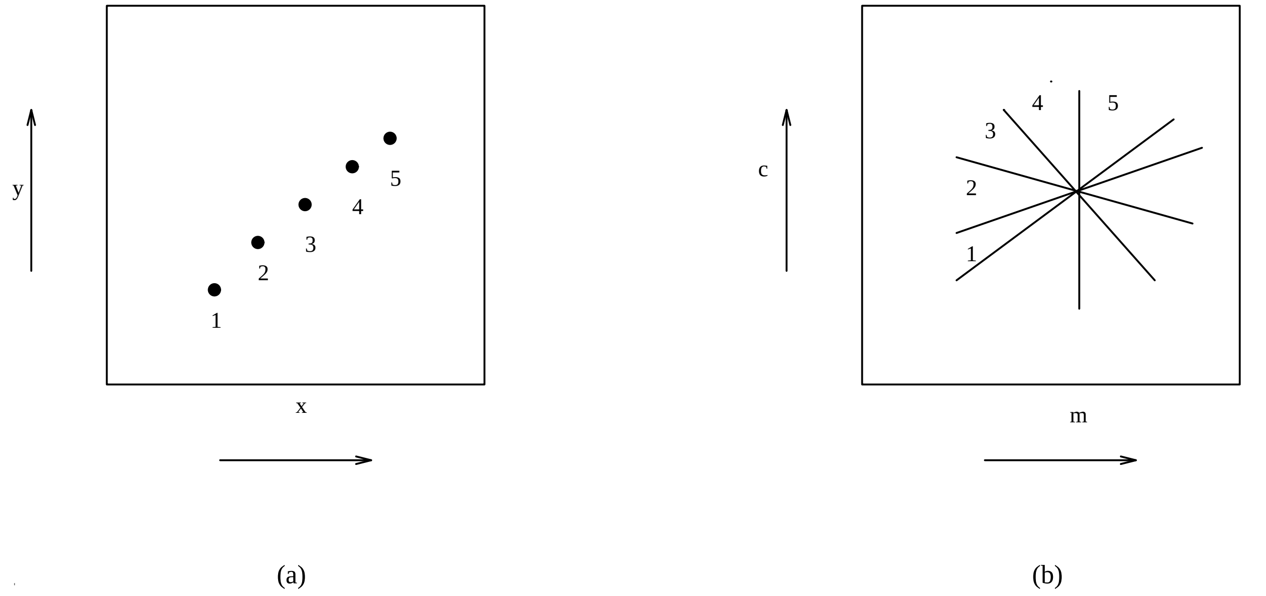


Figure 4.1: Fitting line using the Hough transform. (a) Points in $x-y$ space. (b) Lines in $c-m$ space corresponding to points in $x-y$ space. The lines intersect at a single point.

4.2.1 Straight Line

In this section we will use the Hough transform to fit the equation of a straight line to the edge points. The equation of a line is given by:

$$y = mx + c, \tag{4.1}$$

where m and c are the slope and $y-$intercept of the line. The above equation can be rewritten as:

$$c = (-x)m + y.$$

This is the equation of a line in $c-m$ space, with slope $-x$ and intercept y. Therefore, a point (x, y) in the $x-y$ space is mapped to a straight line in the $c-m$ space (see Figure 4.1). Assume that we have several edge points (x_1, y_1), (x_2, y_2), (x_3, y_3), …, (x_n, y_n) in $x-y$ space to which we want to fit a line. Each point in the (x, y) space maps to a line in the $c-m$ space. These lines intersect at a single point in the $c-m$ space. That point $(\hat{c}, \hat{m})$ is the estimated slope and intercept of a line in $x-y$ space. The algorithm for fitting straight lines to the edge points using the Hough transform is given in Figure 4.2.

The parameterization of a line given in equation 4.1 has one problem. The line parallel to $y-$axis has infinite slope m, which can not be represented by a computer. Another parameterization of a line shown in Figure 4.3 is as follows:

$$p = x\cos\theta + y\sin\theta, \tag{4.2}$$

where θ is the angle between x-axis and a line p, drawn perpendicular from the origin to the line being detected. In this case, θ and p have finite values. Another advantage of using this parameterization is that θ can be computed from the gradient angle during the edge detection. Therefore, instead of looping through all possible values of θ, we can just use the θ from the gradient angle. This way, computational complexity is reduced. The algorithm for fitting line using the polar form of a straight line is given in Figure 4.4.

1. Quantize the parameter space $P[c_{min}, \ldots\ldots, c_{max}, m_{min}, \ldots\ldots, m_{max}]$.
2. For each edge point (x, y) do
   for $(m = m_{min}, m \leq m_{max}, m++)$ do
   $c = (-x)m + y$,
   $P[c, m] = P[c, m] + 1$.
3. Find the local maxima in the parameter space.

Figure 4.2: Hough transform algorithm for fitting straight lines.

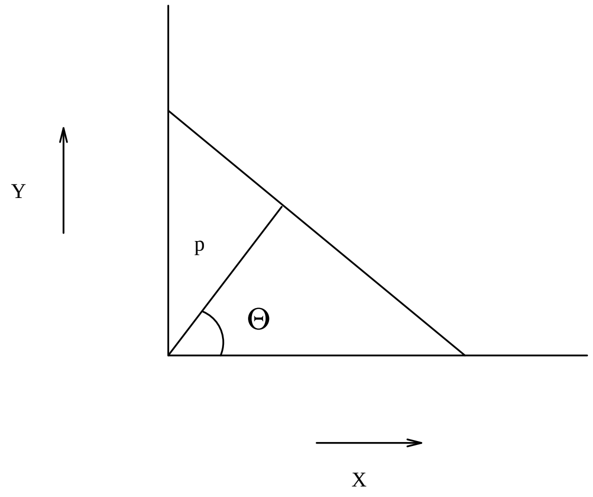


Figure 4.3: Polar form of equation of straight line.

1. Quantize the parameter space $P[\theta_{min}, \ldots\ldots, \theta_{max}, p_{min}, \ldots\ldots, p_{max}]$.
2. For each edge point (x, y) do
   $p = x \cos\theta + y \sin\theta$,
   $P[\theta, p] = P[\theta, p] + 1$.
3. Find the local maxima in the parameter space.

Figure 4.4: Hough transform algorithm using polar form of equation of straight line.

1. Quantize the parameter space
   $P[x_{0min}, \ldots, x_{0max}, y_{0min}, \ldots, y_{0max}, r_{min}, \ldots, r_{max}]$.
2. For each edge point (x, y) do
   for $(x_0 = x_{0min}, x_0 \leq x_{0max}, x_0 + +)$
   for $(y_0 = y_{0min}, y_0 \leq y_{0max}, y_0 + +)$
   do $(x - x_0)^2 + (y - y_0)^2 = r^2$.
   $P[x_0, y_0, r] = P[x_0, y_0, r] + 1$.
3. Find the local maxima in the parameter space.

Figure 4.5: Hough transform algorithm for fitting circle.

1. Quantize the parameter space
   $P[x_{0min}, \ldots, x_{0max}, y_{0min}, \ldots, y_{0max}, r_{min}, \ldots, r_{max}]$.
2. For each edge point (x, y) do
   For $(r = r_{min}, r \leq r_{max}, r + +)$
   $x_0 = x - r\cos\theta$
   $y_0 = y - r\sin\theta$
   $P[x_0, y_0, r] = P[x_0, y_0, r] + 1$.
3. Find the local maxima in the parameter space.

Figure 4.6: Hough transform algorithm for fitting circle using polar form of equation of a circle.

4.2.2 Circle

The equation of a circle centered at (x_0, y_0) with radius r is given by:

$$(x - x_0)^2 + (y - y_0)^2 - r^2 = 0. \tag{4.3}$$

In this case we have three unknowns: x_0, y_0, r. Therefore the parameter space is three dimensional. The algorithm for fitting circle using Hough transform is given in Figure 4.5. The gradient angle information can be used to reduce the computational complexity in this case also. Another parameterization of a circle is given by:

$$x_0 = x - r\cos\theta \tag{4.4}$$
$$y_0 = y - r\sin\theta \tag{4.5}$$

where θ is the gradient angle. The simplified algorithm for fitting a circle is given in Figure 4.6.

4.3 Generalized Hough Transform

So far, the Hough transform has been used to detect shape which can be expressed analytically. However, many shapes in the real world cannot be expressed analytically. In those cases, the generalized Hough transform can be employed to detect any arbitrary shape. The information contained in the equation of a shape can be captured in a table called $R-$Table. The first step in the generalized Hough transform is to develop this $R-$Table. First the centroid of the object is determined as follows:

$$x_c = \frac{\sum_{x=0}^{x=n} \sum_{y=0}^{y=n} f(x,y)x}{\sum\sum f(x,y)} \tag{4.6}$$

$$y_c = \frac{\sum_{x=0}^{x=n} \sum_{y=0}^{y=n} f(x,y)y}{\sum\sum f(x,y)}, \tag{4.7}$$

where $f(x,y)$ is a binary image, defined as follows:

$$f(x,y) = \begin{cases} 1 & \text{if } (x,y) \text{is object pixel} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

Next, for each edge point (x,y), the vector $r = (x', y')$ is determined (as shown in Figure 4.7.a) such that:

$$x_c = x + x' \tag{4.9}$$

$$y_c = y + y' \tag{4.10}$$

The information about the r vectors is collected in the $R-$Table (as shown in Figure 4.7.b) which is indexed by the gradient angle. There is a row corresponding to each possible value of the gradient angle in the $R-$Table. In the second column of this table the r vectors with the same ϕ angle are stored.

During the recognition phase, the aim is to detect the presence of a given shape defined by the $R-$Table, and identify locations where it is present. For each edge point, the gradient angle is determined, and is used to index the $R-$Table. The accumulator array is incremented by all r vectors present in the entry as follows:

$$x_c = x + x' \tag{4.11}$$

$$y_c = y + y' \tag{4.12}$$

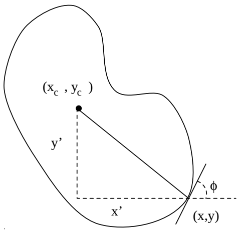
The location of the shape is determined by identifying the local maxima in the accumulator array $P[x_c, y_c]$. The complete algorithm is given in Figure 4.8.

This algorithm will not be able to detect the rotated and scaled version of a given object. However, it is very important for a vision system to be able to identify the scaled, rotated, and translated version of the object. Therefore, we will discuss how the algorithm can be modified to achieve this.

We know that if (x', y') is rotated around $Z-$axis by an angle θ, the new coordinates (x'', y'') are given by:

$$x'' = x' \cos\theta + y' \sin\theta \tag{4.15}$$

$$y'' = -x' \sin\theta + y' \cos\theta \tag{4.16}$$



ϕ_1	$r^1_1, r^1_2, r^1_3, \ \ldots\ , r^1_{m_1}$
ϕ_2	$r^2_1, r^2_2, r^2_3, \ \ldots\ , r^2_{m_2}$
ϕ_3	$r^3_1, r^3_2, r^3_3, \ \ldots\ , r^3_{m_3}$
$\bullet$	$\bullet$
$\bullet$ $\bullet$	$\bullet$ $\bullet$
ϕ_n	$r^n_1, r^n_2, r^n_3, \ \ldots\ , r^n_{m_n}$

(a) (b)

Figure 4.7: Generalized Hough transform. (a) An arbitrary shape with centroid at (x_c, y_c). The tangent angle at point (x, y) is ϕ, and x' and y' are respectively the distance in x and y directions from (x, y) point to the centroid of the object. (b) $R-$Table. The first column contains the gradient angles, and the second column contains the (x', y') vectors.

1. Quantize the parameter space $P[x_{cmin}, \ldots\ldots, x_{cmax}, y_{cmin}, \ldots\ldots, y_{cmax}]$.
2. For each edge point (x, y) do
   compute $\phi(x, y)$
   for each table entry for ϕ do

$$x_c = x + x' \tag{4.13}$$
$$y_c = y + y' \tag{4.14}$$

   $P[x_c, y_c] = P[x_c, y_c] + 1.$
3. Find the local maxima in the parameter space.

Figure 4.8: Generalized Hough transform algorithm.

Similarly, the scaled and rotated version of (x', y') is given by:

$$x'' = s_x(x' \cos\theta + y' \sin\theta) \tag{4.17}$$
$$y'' = s_y(-x' \sin\theta + y' \cos\theta) \tag{4.18}$$

Substituting the above equations in equations 4.9-4.10 we get:

$$x_c = x + s_x(x' \cos\theta + y' \sin\theta) \tag{4.19}$$
$$y_c = y + s_y(-x' \sin\theta + y' \cos\theta) \tag{4.20}$$

Now, if the equations 4.13-4.14 are replaced by the above equations in the generalized Hough transform algorithm given in Figure 4.8, the rotated and scaled version of shape can be detected.

4.4 Shape Number

Chain code is a very simple technique for representing shape of contour. In chain code, each directed line segmented is assigned a code. Two possible codes are shown in Figure 4.9.a-b. In Figure 4.9.a the directions are quantized into four possible directions: 0, 90, 180, and 270 degrees. The corresponding codes for these directions are: 0, 1, 2 and 3. The chain code of a contour is the concatenation of codes of individual line segments. The chain code of a contour shown in Figure 4.9.c is given in Figure 4.9.d. Note that this chain code is dependent upon the start point, and on the orientation of the contour. It is desirable to have code which is translation, rotation, and start point independent. If the contour is rotated by $m \times 90$, where m is an integer, then a constant m is effectively added to the code of each line segment. Since the derivative of a constant is zero, the chain code can be made rotation invariant by simply using the first difference of the chain code instead. The code can be made invariant to the start point by normalizing the first difference such that if the code is interpreted as an integer, it is the minimum possible integer. The normalized first difference of a chain code is called shape number. The number of digits in a shape number is called the *order* of shape number. Some examples of shape number of order 4, 6, and 8 are shown in Figure 4.10.

Finding the shape number of an arbitrary object oriented in an arbitrary direction involves several steps. In order to get an accurate shape number with fewer digits, the contour needs to be re-sampled with a coarser grid which is aligned with the main axes of the object. This will minimize the quantization error. One possible way to achieve this is to impose a grid whose major and minor axes are parallel to the major and minor axis of the rectangle enclosing the object, and the eccentricity of the grid and enclosing rectangle are approximately the same. The complete algorithm is given in Figure 4.11.

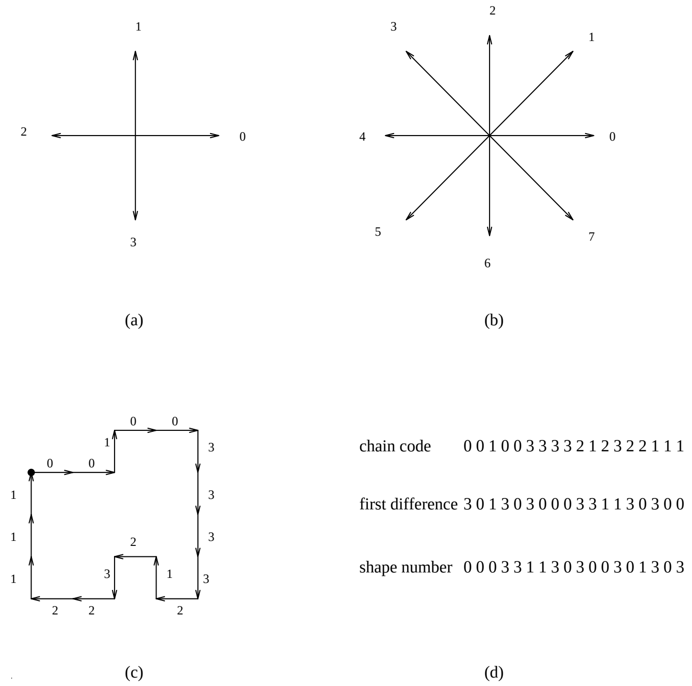


Figure 4.9: Shape number. (a)-(b) Two possible chain codes. (c) A contour with a chain code. (d) Shape number of contour shown in (c).

4.5 Pyramids

Pyramid is very useful data structure for representing image regions. Pyramid is built by using multiple copies of the image. Each level in the pyramid is $\frac{1}{4}$ of the size of the previous level. The lowest level in the pyramid is of the highest resolution, and the highest level is of the lowest resolution.

See Figure 4.12 for an example.

4.5.1 Gaussian Pyramid

In Gaussian pyramid the lowest level, g_0, is the original image I. The level, g_l, is computed by weighted averaging of values in level g_{l-1} in a 5×5 window as follows:

$$g_l(i,j) = \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) g_{l-1}(2i+m, 2j+n). \tag{4.21}$$

This is shown graphically in Figure 4.13 for one dimensional case. Note that the density of nodes in level l in one dimension (two dimensions) is half (one fourth) the nodes at level $l-1$. Therefore, this process is called $REDUCE$ operation:

$$g_l = REDUCE[g_{l-1}]. \tag{4.22}$$

The mask $w(m,n)$ approximates Gaussian, and is separable, that is:

$$w(m,n) = \hat{w}(m)\hat{w}(n). \tag{4.23}$$

The one dimensional mask, $\hat{w}(m)$, is generated using three constraints.

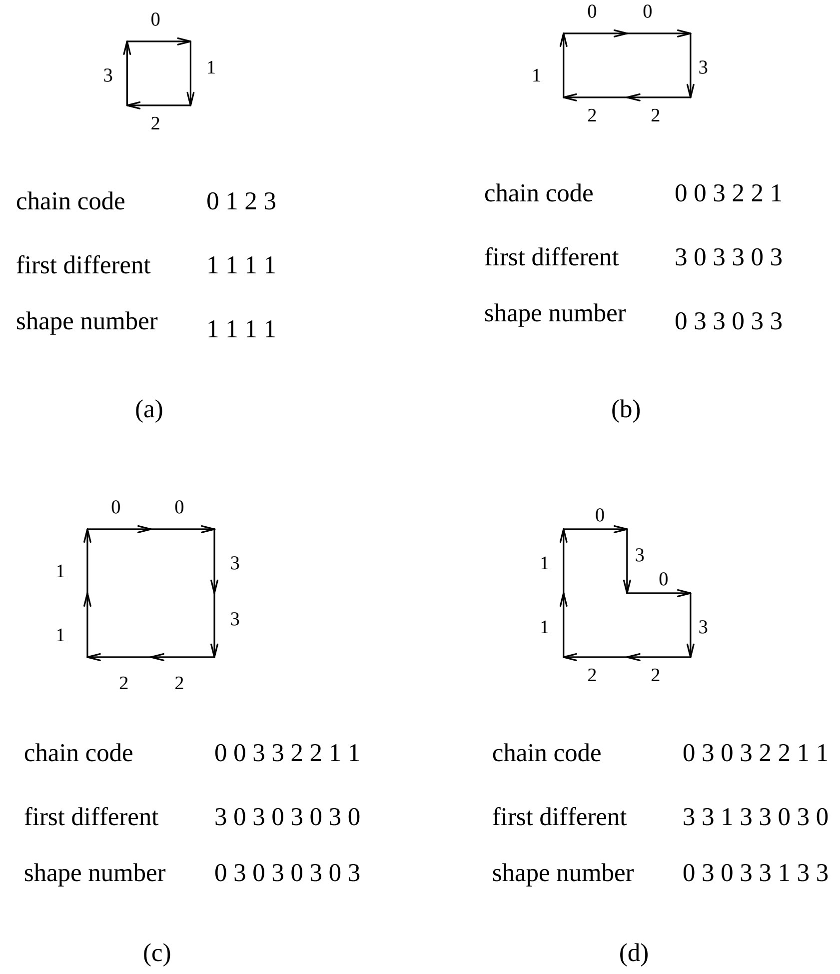


Figure 4.10: Some examples of shape number. (a) Order 4 shape number. (b) Order 6 shape number. (c)-(d) Order 8 shape number.

1. Find eccentricity:
   (a) Find two points at the maximum distance from each other. The line joining those points is the major axis.
   (b) The line perpendicular to the major axis is the minor axis.
   (c) Eccentricity is given by the ratio of the major and minor axis of the enclosing rectangle of the object.
2. Impose a grid structure such that the eccentricity of the grid is equal to the eccentricity of enclosing rectangle.
3. Find the shape number.

Figure 4.11: Algorithm for computing shape number of an arbitrary object.

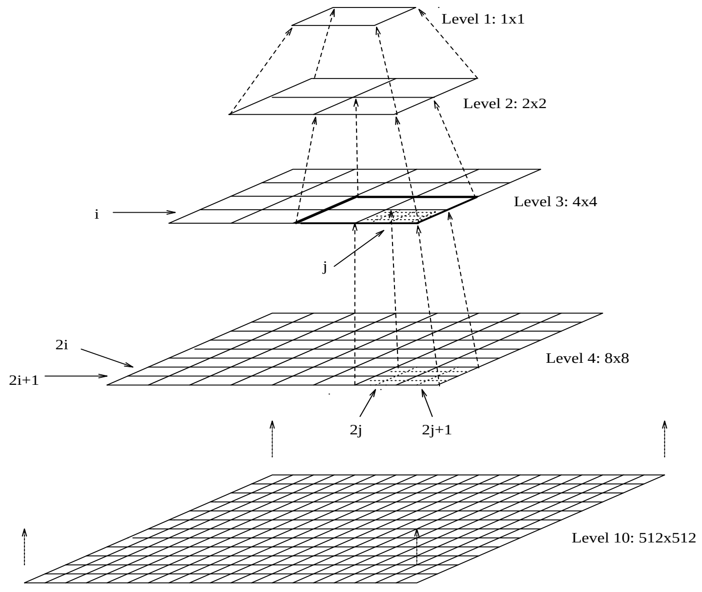


Figure 4.12: Pyramid.

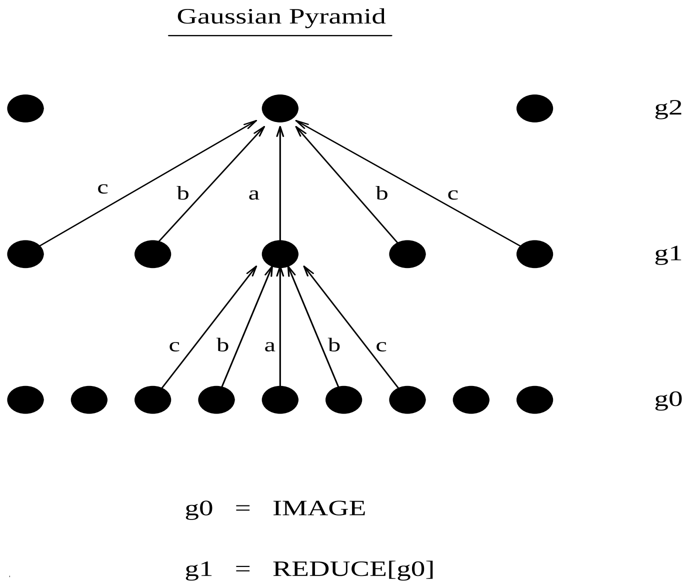


Figure 4.13: A one-dimensional graphic representation of the process which generates a Gaussian pyramid.

1. The mask should be symmetric, that is

$$\hat{w}(i) = \hat{w}(-i) \quad \text{for } i = 0, 1, 2 \tag{4.24}$$

2. The sum of mask should be 1. Let $\hat{w}(0) = a$, $\hat{w}(1) = \hat{w}(-1) = b$, and $\hat{w}(2) = \hat{w}(-2) = c$. Then

$$a + 2b + 2c = 1. \tag{4.25}$$

3. All nodes at a given level must contribute the same total weight to nodes at the next higher level,

$$a + 2c = 2b. \tag{4.26}$$

Using the above three constraints, we have:

$$\hat{w}(0) = a, \tag{4.27}$$

$$\hat{w}(-1) = \hat{w}(1) = \frac{1}{4}, \tag{4.28}$$

$$\hat{w}(-2) = \hat{w}(2) = \frac{1}{4} - \frac{a}{2}. \tag{4.29}$$

The $EXPAND$ operation essentially expands an image of size $M+1$ into an image of size $2M+1$:

$$g_{l,n} = EXPAND[g_{l,n-1}], \tag{4.30}$$

which can be computed using the formula similar to 4.21 as follows:

$$g_{l,n}(i,j) = \sum_{p=-2}^{p=2} \sum_{q=-2}^{q=2} w(p,q).g_{l,n-1}(\frac{i-p}{2}, \frac{j-q}{2}), \tag{4.31}$$

where $g_{l,n}$ is the image at level l obtained by applying $EXPAND$ to g_l n times. Here only terms for which $\frac{i-m}{2}$, and $\frac{i-n}{2}$ are integers are included. One time expansion of the image at level l can be obtained as:

$$g_{l,1}(i,j) = \sum_{p=-2}^{p=2} \sum_{q=-2}^{q=2} w(p,q).g_l(\frac{i-p}{2}, \frac{j-q}{2}), \tag{4.32}$$

Gaussian pyramid for Hamburg taxi image is shown in Figure 4.14.

The Laplacian pyramid, L, is generated by computing the difference between images at successive levels of pyramid:

$$L_l = g_l - EXPAND[g_{l+1}]. \tag{4.33}$$

Each node in image L_l represents the result of applying a difference of two Gaussian functions to the original image. The difference of Gaussian is equivalent to the Laplacian of Gaussian operator commonly used in edge detection.

Hierarchical Discrete Correlation, HDC, is similar to $REDUCE$ operation, except that the density of nodes remains fixed at each level. This is shown graphically in Figure 4.15.

Figure 4.14: Gaussian pyramid for taxi image with $a = 0.4$.

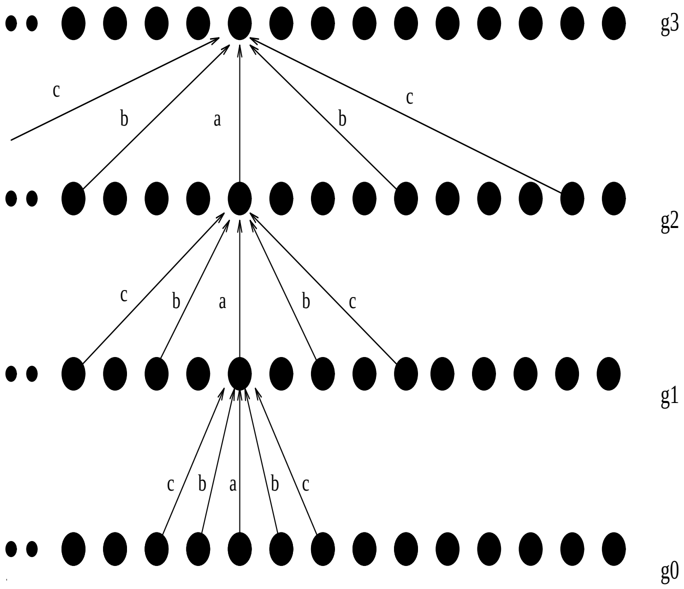


Figure 4.15: A one-dimensional graphic representation of the HDC process.

1. $l = \log \frac{n}{m}$.
2. RefWindow= $m \times m$ window from reference pyramid centered at the feature from level (l).
3. InputWindow = $2m \times 2m$ window from input at the level l (whole image at level l).
4. Find the best match of RefWindow in the InputWindow, let it be at (X_{best}, Y_{best}).
5. if $l = 0$ then stop, else $l = l - 1$.
6. InputWindow=$2m \times 2m$, window at level l centered at location corresponding (X_{best}, Y_{best}).
7. RefWindow= $m \times m$ window centered at the feature from level (l).
8. goto step 4.

Figure 4.16: Algorithm for correlation using pyramids.

4.5.2 Correlation Using Pyramids

Pyramids can be used for several purposes, for example, they can be used for efficiently computing the correlation. Correlation is used in template matching, where a given pattern of a gray level window is passed through the image to find the best match. This can be achieved without using pyramids. Assume that we have to find a match of a 16×16 window in a 256×256 image. For this we need $(256 - 15) \times (256 - 15) = 58,081$ tests. However, it can be shown that using pyramid the tests can be reduced to only $4 \times (16 + 1)^2 = 1,156$, which is a big improvement.

The algorithm for correlation using pyramids is given in Figure 4.16. In this algorithm, first $m \times m$ window from the highest level (l) of the pyramid is used to find the best match in the $2m \times 2m$ window in the input pyramid at the same level. Assume that the best match is at (X_{best}, Y_{best}). Now, $m \times m$ window from the next level $(l - 1)$ of the reference pyramid is used to find the best match in the $2m \times 2m$ window in the input pyramid centerd around (X_{best}, Y_{best}) at the same level. This process continues until the best match is found at the lowest level of pyramid.

4.6 Quad Trees

Quad tree is a useful data strcuture to represent a region. At every node of a quad tree there can be maximum of four descendants. There are three types of descendants: BLACK, WHITE and GRAY. The BLACK node correspond to a region which is uniformly black, and the WHITE note correspond to a region which is uniformly white. While GRAY node corresponds to a region which is a mixture of black and white pixels.

It is easy to generate a quad tree from a pyramid of an image. The algorithm for generating quad tree is given in the Figure 4.17. Three levels of pyramid of an image, and its corresponding quad tree is shown in Figure 4.18.

1. If type of Pyramid is BLACK or WHITE then return.
   else
   (a) Recursively find the quad tree of south-east quadrant.
   (b) Recursively find the quad tree of south-west quadrant.
   (c) Recursively find the quad tree of north-east quadrant.
   (d) Recursively find the quad tree of north-west quadrant.
2. Retrun.

Figure 4.17: Algorithm for computing quad tree using pyramid.

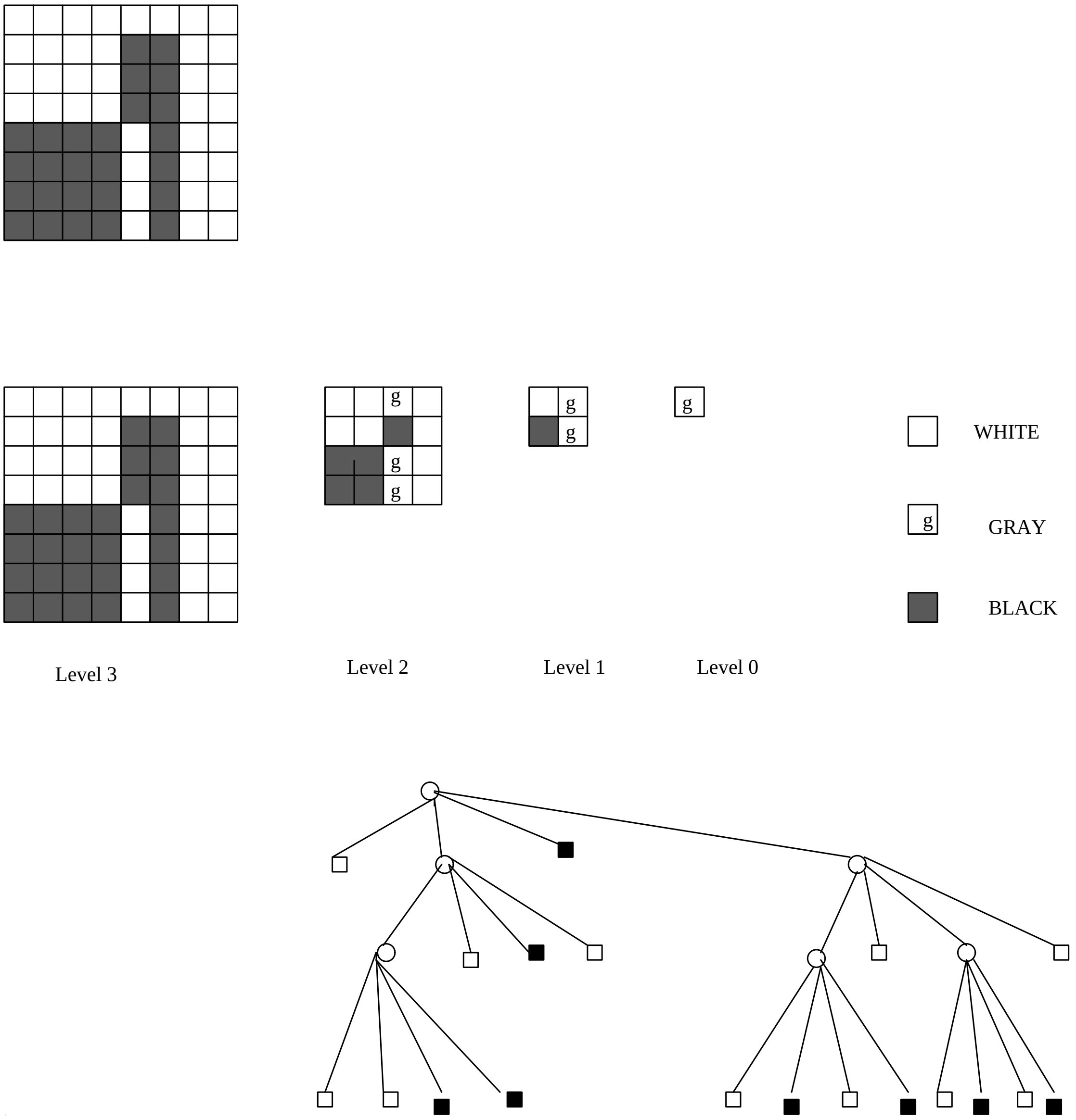


Figure 4.18: Quad tree generation.

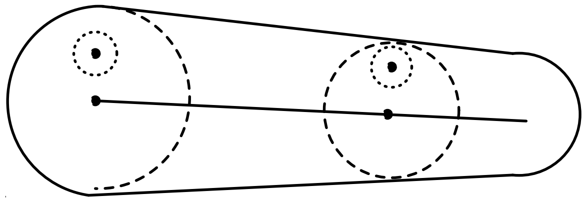


Figure 4.19: Medial axis transform. The points on the medial axis are the centers of the maximal circular neighborhoods totally contained in the shape. Note that the centers of the smaller circles (shown dotted) do not constitute the medial axis fo this shape.

1. Iteratively compute f^k as follows:

$$f^k(x,y) = f^0(x,y) + \min(f^{k-1}(p,q)) \tag{4.34}$$

   $\forall (p,q)$ such that $distance((x,y),(p,q)) \leq 1$.

2. Medial axis is given by all points (x,y) such that:

$$f^k(x,y) \geq f^k(p,q), \tag{4.35}$$

   $\forall (p,q)$ such that $distance((x,y),(p,q)) \leq 1$.

Figure 4.20: Iterative algorithm for computing medial axis transform.

4.7 Medial Axis Transform

The medial axis transform is used to represent the shape of regions. This is basically a skeleton of the region. One way to define the medial axis is as follows: the points on the medial axis are the centers of the maximal circular neighborhoods totally contained in the shape (see Figure 4.19). An iterative algorithm for computing the medial axis transform is given in Figure 4.20. In this algorithm, it is assumed that the binary image $f^0(x,y)$ containing the shape of the region is given. The f^0 is used to iterartively compute $f^1, f^2, f^3, \ldots, f^k$. The first step of the algorithm terminates when f^{k-1} and f^k are the same. In the second step the points which are local maxima in f^k are identified as the medial axis. The medial axis transform of region shown in Figure 4.22.a, is shown in Figure 4.22.c. The various steps involved in computing its medial axis are shown in Figure 4.22.b-c. The points consisting of the medial axis are shown in bold face in Figure 4.22.c.

An iterative algorithm for computing the shape from the medial axis is given in Figure 4.21. If the inverse medial axis transform is applied to the medial axis shown in Figure 4.22.c, the original shape can be recovered. The various steps in this process are shown in Figure 4.23.

1. Iteratively compute g^k as follows:

$$g^k(x,y) = \begin{cases} \max[0, (\max g^{k-1}(p,q)) - 1] & \text{if } g^{k-1} = 0 \\ g^{k-1} & \text{otherwise} \end{cases} \tag{4.36}$$

$\forall (p,q)$ such that $distance((x,y),(p,q)) \leq 1$.

Figure 4.21: Iterative algorithm for inverse medial axis transform.

(a)

1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1

$\longrightarrow$

(b)

1	1	1	1	1	1
1	2	2	2	2	1
1	2	2	2	2	1
1	2	2	2	2	1
1	1	1	1	1	1

$\longrightarrow$

(c)

1	1	1	1	1	**1**
1	**2**	2	2	**2**	1
1	2	**3**	**3**	2	1
1	**2**	2	2	**2**	1
1	1	1	1	1	**1**

Figure 4.22: Medial axis transform. (a) Rectangular shape ($f^0(x,y)$). The background pixels, which are not shown, are all '0". (b) intermediate step ($f^1(x,y)$). (c) $f^2(x,y)$. The points in Medial axis shown in boldface.

(a)

1					**1**
	2			**2**	
		3	**3**		
	2			**2**	
1					**1**

$\longrightarrow$

(b)

1	1			1	**1**
1	**2**	2	2	**2**	1
	2	**3**	**3**	2	
1	**2**	2	2	**2**	1
1	1			1	**1**

$\longrightarrow$

(c)

1	1	1	1	1	**1**
1	**2**	2	2	**2**	1
1	2	**3**	**3**	2	1
1	**2**	2	2	**2**	1
1	1	1	1	1	**1**

Figure 4.23: Inverse medial axis transform. (a) Medial axis ($g^0(x,y)$). (b) Intermediate step ($g^1(x,y)$). (c) Original shape ($g^2(x,y)$).

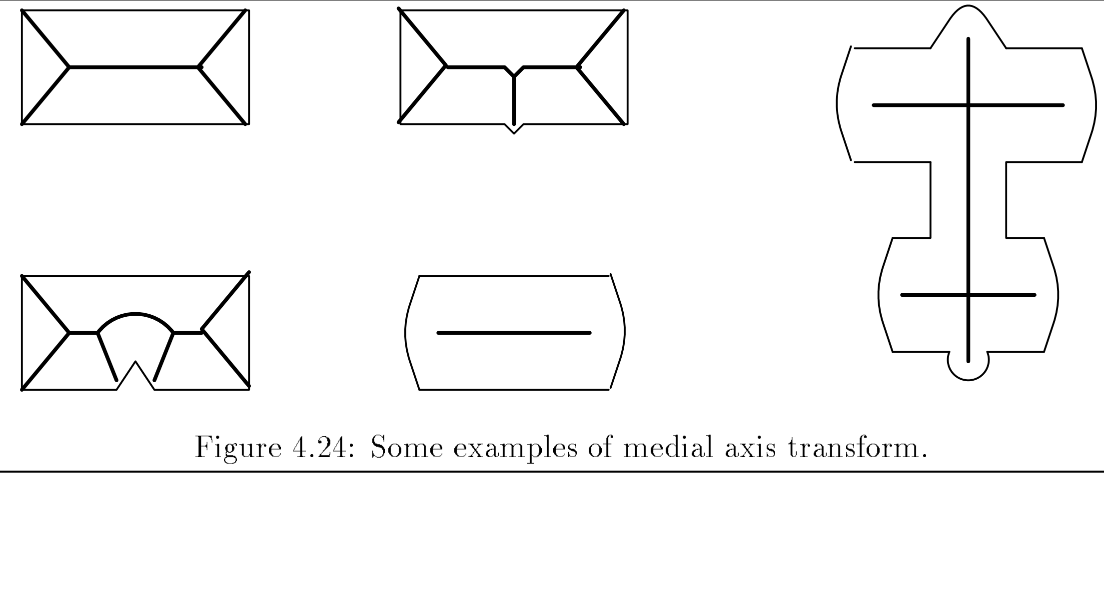


Figure 4.24: Some examples of medial axis transform.

4.8 Moravec's Interest Operator

Moravec's interest operator is a very useful feature point detector, and is widely used in motion and stereo. This operator identifies points (in fact small windows) in the image which are *interesting* in the gray levels. First, for each 4×4 overlapping window, four directional variances in gray levels (horizontal, vertical, diagonal, and anti-diagonal) denoted by V_h, V_v, V_d and V_a are determined as follow (see Figure 4.25.a-d):

$$V_h(x,y) = \sum_{j=0}^{3}\sum_{i=0}^{2}(P(x+i,y+j) - P(x+i+1,y+j))^2 \tag{4.37}$$

$$V_v(x,y) = \sum_{j=0}^{2}\sum_{i=0}^{3}(P(x+i,y+j) - P(x+i,y+j+1))^2 \tag{4.38}$$

$$V_d(x,y) = \sum_{j=0}^{2}\sum_{i=0}^{2}(P(x+i,y+j) - P(x+i+1,y+j+1))^2 \tag{4.39}$$

$$V_a(x,y) = \sum_{j=0}^{2}\sum_{i=1}^{3}(P(x+i,y+j) - P(x+i-1,y+j+1))^2. \tag{4.40}$$

Next, array $V(x,y)$ is computed as follows:

$$V(x,y) = \min(V_h(x,y), V_v(x,y), V_d(x,y), V_a(x,y)). \tag{4.41}$$

A 4×4 window centered at pixel (x,y) is declared *interesting* if it is a local maxima in a 12×12 neighborhood (see Figure 4.25.e). The array $I(x,y)$ stores the interesting points in the image:

$$I(x,y) = \begin{cases} 1 & \text{if } V(x,y) \geq V(p,q), \quad \forall (p,q) \in N(x,y) \\ 0 & \text{otherwise.} \end{cases} \tag{4.42}$$

The resulst for Moravec's interest operator are shown in Figure 4.26.

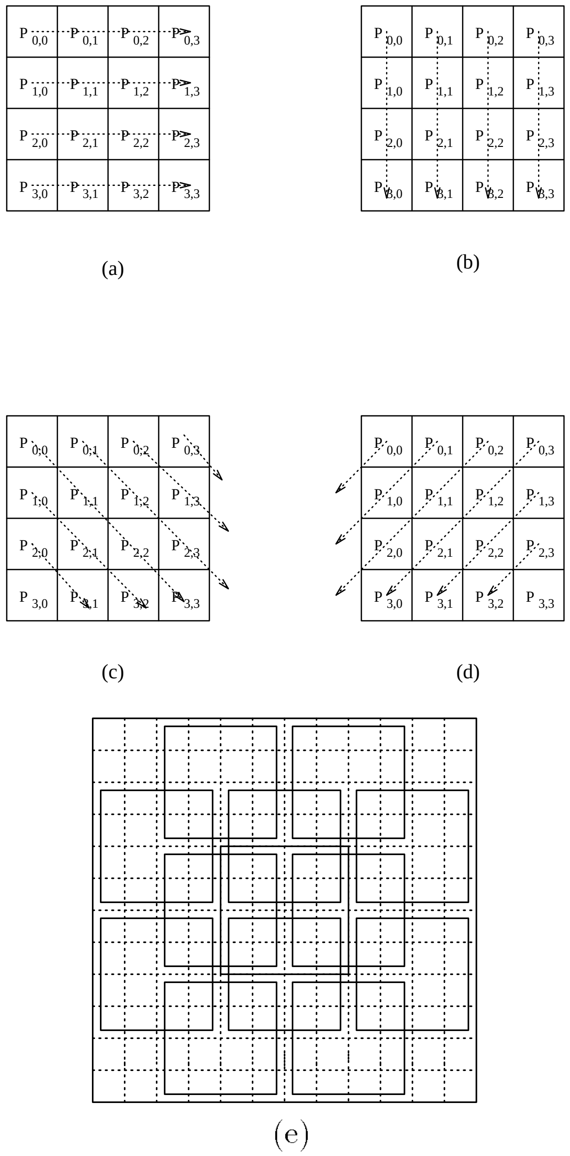
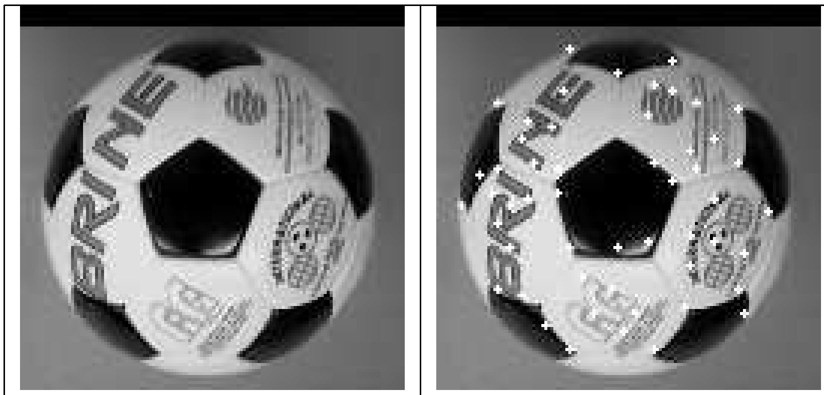


Figure 4.25: Moravec's interest operator. (a) Gray level variance in horizontal direction. (b) Gray level variance in vertical direction. (c) Gray level variance in diagonal direction. (d) Gray level variance in anti-diagonal direction. (e) 25 overlapping windows in a 12×12 neighborhood.



(a) (b)

Figure 4.26: Results for Moravec interest operator. (a) Gray level image. (b) Moravec interest points (total 52 points) superimposed on (a).

4.9 Exercises

1. Devise a scheme for detecting *parabolas* $y - y_0 = a(x - x_0)^2$, centered at (x_0, y_0) using *Hough Transform*. Use the gradient information to reduce the computations. What is the computational complexity of your algorithm?

2. Devise a scheme for detecting *ellipses* that are known to be oriented so that a principal axis is parallel to the x axis using *Hough Transform*. Use the gradient information to reduce the computations. What is the computational complexity of your algorithm?

3. If the *generalized Hough transform* is used for detecting *circle*, how the *R-Table* will look like?

4. How would you distinguish an *arc* from a *complete* circle using the *Generalized Hough* transform?

5. Consider an arbitrary shape object in an image. The gradient angle at each edge pixel is given in the following array. The centroid of the object is shown by $*$.

90	45			90	90	90
180		90	135			0
	225					0
	180		*			0
	180					0
135						0
180				225	270	270
180	270	270	270			

   (a) Prepare the *R-Table* for this object to be used in the Generalized Hough Transform.

(b) Assume now that the some image has been given to you, whose gradient magnitude is given in the above table. Run through the various steps in the *Generalized Hough Transform*; identify the location of the object (X_c, Y_c) using the *R-Table* you generated in the part (a).

6. This problem will give you some experience with correlation matching using pyramids. Below are given two pyramids: reference and input. For simplicity, these pyramids have only two levels. Each level of the pyramid is built by taking alternate rows and columns from the previous level. A 2×2 pattern consisting of a's is to be identified in the input image. Assume that the center of an even-sized window is at the upper left corner pixel. Also, assume that each 2×2 block at a given level of pyramid is mapped to 1×1 block at the next level, in order to map the location of a window at one level to the corresponding location in the next level.

			a	a			
			a	a			

		a	

Reference Pyramid

					a	a	
					a	a	

			a

Input Pyramid

(a) How many comparisons are needed to locate the pattern in the input image without using pyramid?

(b) Run through various steps in the correlation matching using pyramids. For each level identify the window being searched, and the location of the correct match.

(c) How many comparisons are needed to locate the pattern in the input image using pyramid?

(d) If the pattern in the reference image is displaced to some other arbitrary location, will you still be able to locate the pattern in the input image using steps in (b).

(e) If the pattern in the input image is displaced to some other arbitrary location, will you still be able to locate the pattern in the input image using steps in (b).

(f) Can you generalize the result in (a) considering $m \times m$ pattern, and $n \times n$ image, where both m, n are even numbers.

(g) Can you generalize the result in (c) considering $m \times m$ pattern, and $n \times n$ image, where both m, n are even numbers.

7. Outline recursive algorithm for computing an *area* of a region from the quad tree.

8. Compute the *Medial Axis* of the following shape using 8-neighborhood. Please show all the necessary steps.

0	0	0	0	0	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

9. Using the medial axis computed above recover the original shape back by applying the inverse medial axis transform.

10. Assume that an object has the following chain-code (8-connected), compute its shape number. `076666553321212`

Chapter 5

Motion

5.1 Introduction

So far we have been dealing with a single image of a static scene taken by a fixed camera[1]. In this chapter, we will be dealing with a sequence of images taken at different time intervals. The motion of objects in 3-D induces the 2-D motion in the image plane. That motion is called optical flow. In this chapter, we will discuss several methods for computing optical flow. The optical flow can be used to compute 3-D motion, i.e. translation and rotation, and 3-D shape. We will also describe one general method for computing 3-D information from optical flow.

5.2 Optical Flow

5.2.1 Horn and Schunck Method

Let 3-D function $f(x, y, t)$, where x, y are the spatial coordinates, and t is time, denote the image sequence. Then, $f(x_1, y_1, t_1)$ is the gray level at location x_1, y_1 at time t_1. Assume that with a small change dx, dy, and dt in x, y, and t there is no change in the gray levels, that is:

$$f(x, y, t) = f(x + dx, y + dy, t + dt). \tag{5.1}$$

By finding the Taylor series expansion around x, y, t of the right hand side we get:

$$f(x, y, t) = f(x, y, t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt. \tag{5.2}$$

The above equation can be simplified as:

$$f_x dx + f_y dy + f_t dt = 0, \tag{5.3}$$

where $f_x = \frac{\partial f}{\partial x}$, $f_y = \frac{\partial f}{\partial y}$, and $f_t = \frac{\partial f}{\partial t}$. These derivatives can be computed by applying masks shown in Figure 5.1 to the image sequence $f(x, y, t)$.

$$\begin{array}{|c|c|}\hline -1 & 1 \\ \hline -1 & 1 \\ \hline \end{array} \quad \begin{array}{|c|c|}\hline -1 & 1 \\ \hline -1 & 1 \\ \hline \end{array} \qquad \begin{array}{|c|c|}\hline -1 & -1 \\ \hline 1 & 1 \\ \hline \end{array} \quad \begin{array}{|c|c|}\hline -1 & -1 \\ \hline 1 & 1 \\ \hline \end{array} \qquad \begin{array}{|c|c|}\hline -1 & -1 \\ \hline -1 & -1 \\ \hline \end{array} \quad \begin{array}{|c|c|}\hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array}$$

(a) (b) (c)

Figure 5.1: Masks for computing spatial and temporal derivatives. Note that the center of mask is at the lowere right pixel. (a) Masks for f_x, (b) Masks for f_y, and (c) Masks for f_t.

Dividing each term in the above equation by dt we get:

$$f_x u + f_y v + f_t = 0, \tag{5.4}$$

where $u = \frac{dx}{dt}$, and $v = \frac{dy}{dt}$ is the optical flow. There are two unknowns, u and v, in the above equation, we cannot solve this equation easily. The above equation can be rewritten as:

$$v = -\frac{f_x}{f_y}u - \frac{f_t}{f_y}. \tag{5.5}$$

This is the equation of a straight line in $u - v$ space. There are several possible solutions of this equation, the solutions can lie anywhere on the line shown in Figure 5.2. Let $(\hat{u}, \hat{v})$ be the correct solution. This vector can be divided into two components, one along the straight line denoted by p, and another perpendicular to the line denoted by d. It is easy to show by using trigonometric identities that $d = \frac{f_t}{\sqrt{{f_x}^2+{f_y}^2}}$. Therefore, knowing the derivatives f_x, f_y and f_t we can only compute the normal component, d, of optical flow. However, the parallel component, p, can not be computed directly from the derivatives.

One of the first approaches for computing optical flow was proposed by Horn and Schunck [9]. Horn and Schunck proposed to estimate (u, v) such that following error function, E, is minimized:

$$E(x, y) = (f_x u + f_y v + f_t)^2 + \lambda({u_x}^2 + {u_y}^2 + {v_x}^2 + {v_y}^2), \tag{5.6}$$

where the first term is the optical flow constraint from equation 5.4, and the second term corresponds to the smoothness of optical flow. For the correct (true) optical flow the first term should be close to zero, or the square of the first term should be small. Since the motion of most real world objects is smooth, the second term enforces the smoothness constraint. Differentiating E with respect to u and v, and equating it to zero we get:

$$\frac{\partial E}{\partial u} = (f_x u + f_y v + f_t)f_x + \lambda(u_{xx} + u_{yy}) = 0, \tag{5.7}$$

$$\frac{\partial E}{\partial v} = (f_x u + f_y v + f_t)f_y + \lambda(v_{xx} + v_{yy}) = 0. \tag{5.8}$$

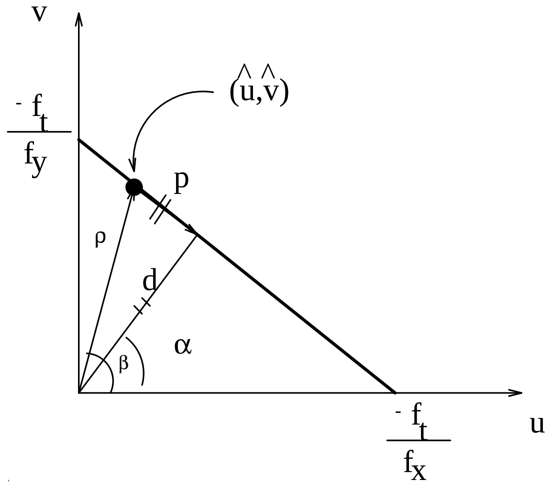


Figure 5.2: Optical flow constraint line in $u - v$ space. d is the length of perpendicular from the origin to the line, α is the angle the perpendicular makes with the x-axis. $(\hat{u}, \hat{v})$ is one possible solution. This optical flow vector, $(\hat{u}, \hat{v})$, can be divided into two components: p, which is along the constraint line, and d which is perpendicular to the constraint line.

Substituting $\Delta^2 u = u_{xx} + u_{yy}$, and $\Delta^2 v = v_{xx} + v_{yy}$ we get:

$$(f_x u + f_y v + f_t) f_x + \lambda(\Delta^2 u) = 0, \tag{5.9}$$
$$(f_x u + f_y v + f_t) f_y + \lambda(\Delta^2 v) = 0. \tag{5.10}$$

Let $\Delta^2 u = u - u_{av}$, where u_{av} is the average of the u component of optical flow taken over the four nearest neighbors of a pixel. Similarly let $\Delta^2 v = v - v_{av}$. Then,

$$(f_x u + f_y v + f_t) f_x + \lambda(u - u_{av}) = 0, \tag{5.11}$$
$$(f_x u + f_y v + f_t) f_y + \lambda(v - v_{av}) = 0. \tag{5.12}$$

These two equations can now be solved for (u, v) as:

$$u = u_{av} - f_x \frac{P}{D}, \tag{5.13}$$
$$v = v_{av} - f_y \frac{P}{D}, \tag{5.14}$$

where $P = f_x u_{av} + f_y v_{av} + f_t$, and $D = \lambda + {f_x}^2 + {f_y}^2$. The iterative algorithm for computing optical flow using the above equations is given in Figure 5.3. The results for Horn and Schunck's algorithm are shown in Figure 5.4.

5.2.2 Schunck Method

Recently, Schunck [22] proposed a simple method for computing optical flow. In this method multiple constraints are used to compute the flow. Since the gray level at a single pixel gives only one constraint, the optical flow can lie anywhere on the straight line determined by the spatial and temporal derivatives. If the second constraint from the neighboring pixel is used

1. $k = 0$.
2. Initialize u^k and v^k to zero.
3. Until some error measure is satisfied, do:

$$u^k = u_{av}^{k-1} - f_x \frac{P}{D}, \tag{5.15}$$

$$v^k = v_{av}^{k-1} - f_y \frac{P}{D}. \tag{5.16}$$

Figure 5.3: Horn and Schunck algorithm for computing optical flow.

then the correct optical flow can be determined by computing the intersection of two lines represented by the constraints. In general, it is desirable to employ multiple constraints, not just two constraints. Schunck uses eight constraints obtained from points around a 3×3 neighborhood of a pixel. That results in eight intersections of lines. If the measurements are noise free, and all pixels in a 3×3 neighborhood belong to the same moving object, then, in principle, all eight lines will intersect at a single point, which is the correct optical flow. But, due to noise, or if some points lie on the boundary of the moving object, all eight lines may not intersect at the same point. The intersections may be spread out. However, the intersections may cluster around some true solution. Therefore, Schunck uses the tightest cluster of the intersection consisting of at least half the intersections to determine the optical flow.

It is easier to consider the polar form of the optical flow equation (equation 5.5) as

$$d = \rho \cos(\alpha - \beta), \tag{5.17}$$

where $d = \frac{f_t}{\sqrt{f_x{}^2 + f_y{}^2}}$ is the length of a perpendicular from the origin to the constraint line, and $\alpha = \arctan \frac{f_y}{f_x}$ is the angle the perpendicular makes with the x-axis (see Figure 5.2), ρ is the speed, and β is the direction of the optical flow vector. Now for points 1 and 2 will give the following two optical flow equations:

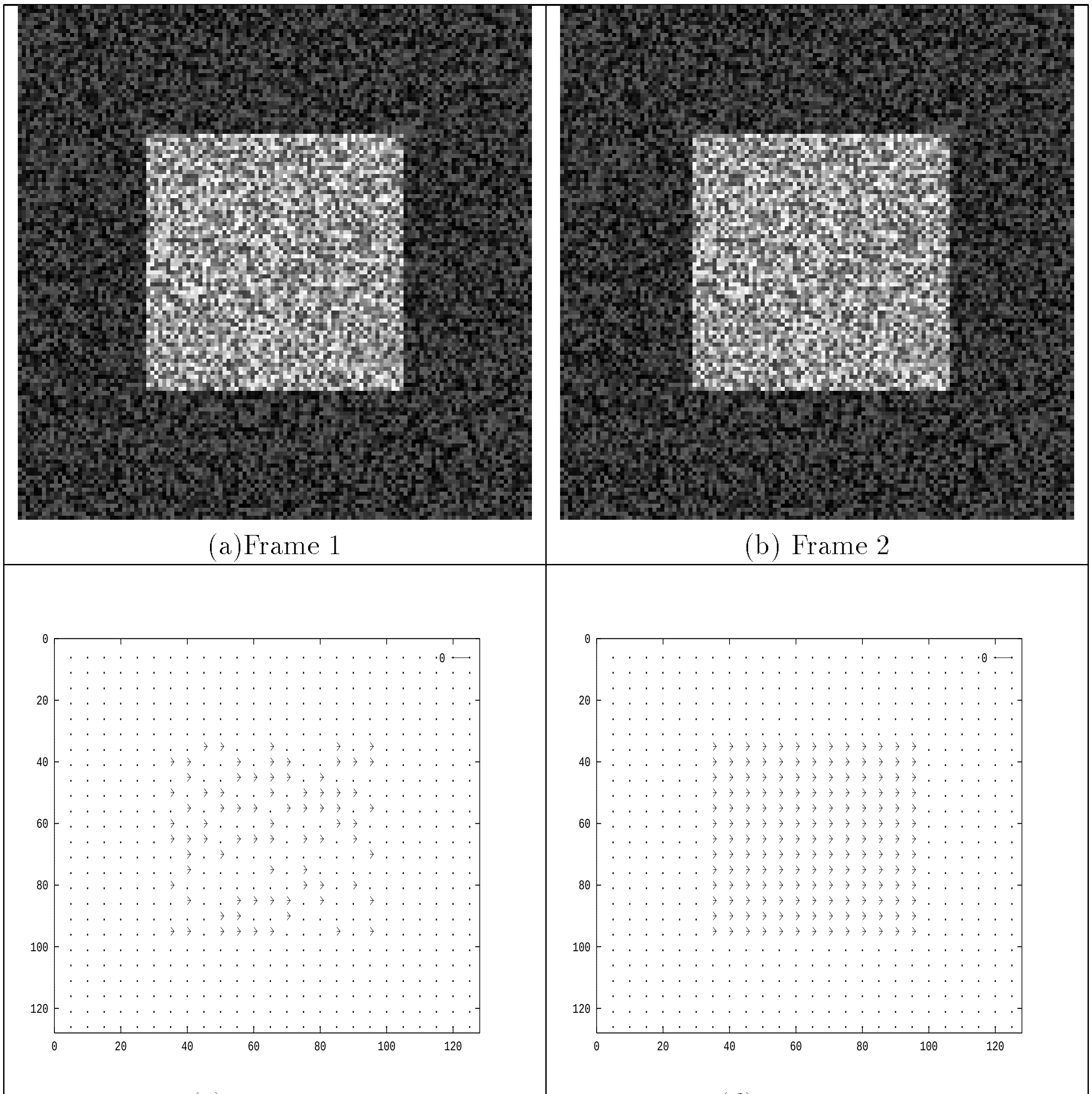
$$d_1 = \rho \cos(\alpha_1 - \beta), \tag{5.18}$$

$$d_2 = \rho \cos(\alpha_2 - \beta). \tag{5.19}$$

These two lines intersect at a point, which is at a distance b_2 (line segment AB) from the the intersection of perpendicular, d_1, drawn from the origin to the constraint line (as shown in Figure 5.5). It is easy to show that b_2 given by

$$b_2 = \frac{d_2 - d_1 \cos(\alpha_2 - \alpha_1)}{\sin(\alpha_2 - \alpha_1)}. \tag{5.20}$$

Similarly, the intersection with lines corresponding to points $3, 4, \ldots, 9$ can be computed. These intersections are likely to cluster around some point. If the center position of the



(a)Frame 1 (b) Frame 2

(c) 1 iteration (d) 10 iterations

Figure 5.4: Results for Horn and Schunck algorithm for displacement of 1 pixel and $\lambda = 4$.

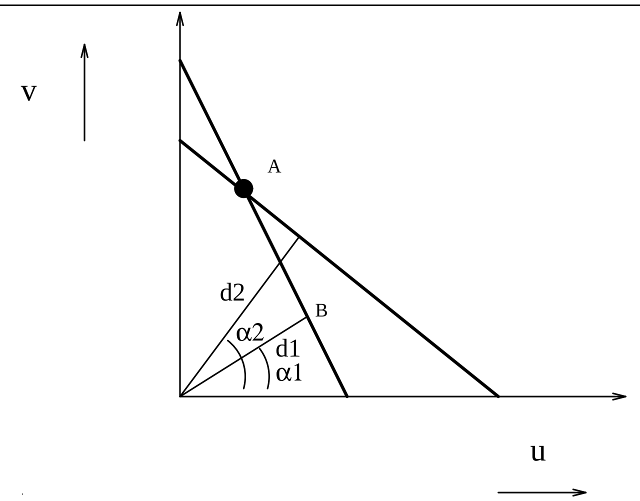


Figure 5.5: Intersection of two optical flow constraint lines.

cluster is $\hat{b}$, then the optical flow is given by:

$$\hat{\rho} = \sqrt{{d_1}^2 + \hat{b}^2}, \tag{5.21}$$

$$\hat{\beta} = \alpha_1 + \arctan(\frac{\hat{b}}{d_1}). \tag{5.22}$$

5.3 Token Based Optical Flow

Given two frames taken at different time instants and m points in each frame, the correspondence problem deals with with a mapping of a point in one frame to another point in the second frame such that no two points map onto the same point. The correspondence is required in many computer vision algorithms, and there exist a number of approaches to this problem. A slightly simplified form of this problem is encountered in *stereo*, where the tokens from the left image are to be matched to those in the right image to compute the disparity. The disparity is proportional to the depth of the objects the tokens belong to in the three dimensional world. The problem in stereo is simplified because the possible matches can only occur along an *epipolar* line. In the case when the objects are moving and frames are taken at different time intervals, the possible matches can occur anywhere in the next frame, unlike stereo where the frames are separated in space not in time.

The correspondence problem is combinatorially explosive. For instance, with 2 frames and m points in each frame, the number of possible mappings is $m!$. To give an idea, the number of possible matches for $m = 5$ is given as $(5!) = 120$. There are two interesting points to be noted here. First, trying all possible mappings will be almost impossible even for a moderate number of frames and points. Second, even if we know all possible trajectories, how do we determine which set is the correct one?

In order to cope with the complexity of this problem, researchers have used a number of constraints. The constraints include *maximum velocity, small velocity change* or *smoothness of motion, common motion, consistent match, rigidity*, etc (see Figure 5.6). The maximum velocity constraint implies that if the bound on the velocity is known a priori, given a position of a point in one frame one can limit the search for a possible match in the next

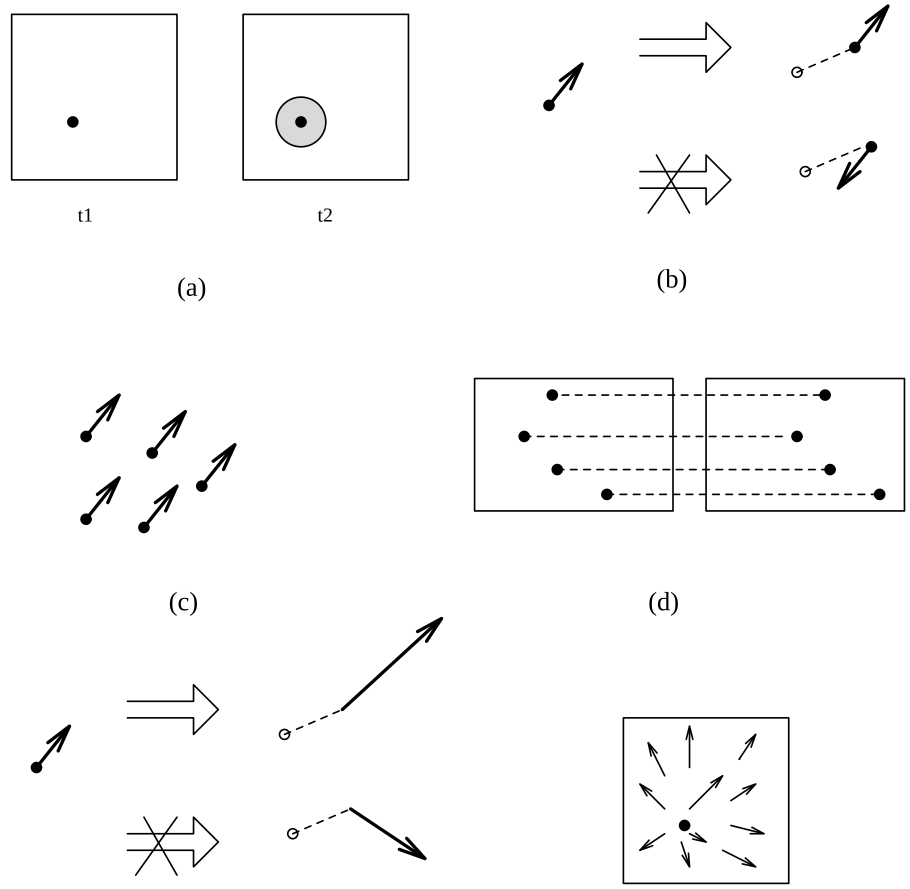


Figure 5.6: Constraints used in motion correspondence. (a) Maximum velocity, (b) Small velocity changes, (c) common motion, (d) consistent match, (e) model, (f) proximal uniformity constraint.

frame to a small neighborhood of the position in the present frame. The small velocity change heuristic assumes that the direction and speed of the motion cannot change by a large amount, therefore one can eliminate some false matches. This constraint essentially leads to *smooth* motion. Common motion constrains the motion of the points in a small neighborhood to be similar, and the consistent match forces only one match for one point.

The rigidity and smoothness of motion assumptions have received the most attention. Rigidity implies that the objects in the three-dimensional world are rigid, therefore the Euclidean distance between any two point on the rigid object will remain unchanged in the next frame. In fact, Ullman *et al* attempted to show that smoothness follows from rigidity, that is, if the objects are rigid their underlying motion will be smooth, but not necessarily vice versa.

An important issue in the correspondence problem is to convert the above *qualitative* heuristics into *quantitative* expressions, which become the cost functions. Then the aim is to search for mapping which minimizes one of these functions. Enumeration of all possible sets, and picking the one with the least cost is not possible. Therefore, one needs to use a good approximation algorithm to obtain a sub-optimal solution that is very close to the optimum solution.

5.3.1 Barnard and Thompson Method

Barnard and Thompson [1] proposed an iterative algorithm for computing motion correspondence or optical flow. They use confidence measure P_{ij} to denote the probability of token i in frame one matches with the token j in frame two. The initial probabilities P^0_{ij} are computed using gray level differences in small window around the location of corresponding tokens.

$$P^0_{ij} = \frac{1}{c + w_{ij}} \tag{5.23}$$

where $w_{ij} = \sum_{dy=-w}^{dy=w} \sum_{dx=-w}^{dx=w} (f_1(x_i + dx, y_i + dy) - f_2(x_j + dx, y_j + dy))^2$. The subsequent iterations are defined as:

$$\begin{aligned} P^n_{ij} &= \frac{\tilde{P}^n_{ij}}{\sum_j \tilde{P}^n_{ij}}, && (5.24)\\ \tilde{P}^n_{ij} &= P^{n-1}_{ij}(A + Bq^{n-1}_{ij}), && (5.25)\\ q^{n-1}_{ij} &= \sum_k \sum_l P^{n-1}_{kl}, && (5.26) \end{aligned}$$

where k is a neighbor of i, l is a neighbor of j, such that $||(x_i, y_i) - (x_k, y_k)|| \leq D_{max}$, and $||V_{ij} - V_{kl}|| \leq V_{\max}$ (V_{ij} is the optical flow of i-the token when it mataches with j-th token in next frame). The demonstration of Barnard and Thompson's algorithm is shown in Figure 5.7. There are three points ($i = 1, 2$ and 3) in frame one, and three points ($j = 1, 2$, and 3) as shown in Figure 5.7.a. The possible data structure for this example is shown in Figure 5.7.b. There is one list corresponding to each point. The first element in the list is the coordinates of a point in the first frames, and the subsequent elements in the list are the possible optical flows with corresponding probabilities. For instance, the first element in the first list is $(4, 10)$ which are the coordinates of the first point in the first frame, the second element in the list, $((5, 0), .7)$, represents the match of the first element in the first frame with the first element in the second frame. The optical flow in this case is $(5, 0)$ with the initial probability .7. In Figure 5.7.c the matrices q_{ij}^0, $\tilde{P}_{ij}^1$, and P_{ij}^1 are shown.

5.4 Motion Correspondence Using Multiple Frames

We are given a sequence of n frames denoted by $f^1, f^2, \ldots, f^n$. We assume that the important tokens in each frame have already been identified using a corner detector or an interest operator. Therefore, each frame f^i is a set of points. Corresponding to the ith point in the jth frame, we have its 2-D coordinates denoted by a vector X_i^j. Our aim is to come up with a one to one onto correspondence Φ^k between points of kth frame and $(k+1)$th frame.

It is not unrealistic to assume that in space objects move small distances in a small time interval, and their corresponding motion is smooth or uniform. If the time interval between frames is small, then the 2-D projection of 3-D motion will also be small and smooth. Therefore, the location of a point in the next frame will be in the *proximity* of the location in the previous frame. Smoothness of the motion implies minimum change in *velocity* of the point, that is the object can not change its direction and speed instantaneously. Hence the objects will follow a *proximal uniform path.* We can establish correspondence by minimizing the following *proximal uniformity function* δ, which will prefer the proximal uniform path.

$$\delta(X_p^{k-1}, X_q^k, X_r^{k+1}) = \frac{||\, \overline{X_p^{k-1}X_q^k} - \overline{X_q^k X_r^{k+1}}\, ||}{\sum_{x=1}^m \sum_{z=1}^m ||\, \overline{X_x^{k-1}X_{\Phi^{k-1}(x)}^k} - \overline{X_{\Phi^{k-1}(x)}^k X_z^{k+1}}\, ||} + \frac{||\, \overline{X_q^k X_r^{k+1}}\, ||}{\sum_{x=1}^m \sum_{z=1}^m ||\, \overline{X_{\Phi^{k-1}(x)}^k X_z^{k+1}}\, ||}$$

where $1 \leq p, q, r \leq m$; $2 \leq k \leq m-1$; $q = \Phi^{k-1}(p)$;

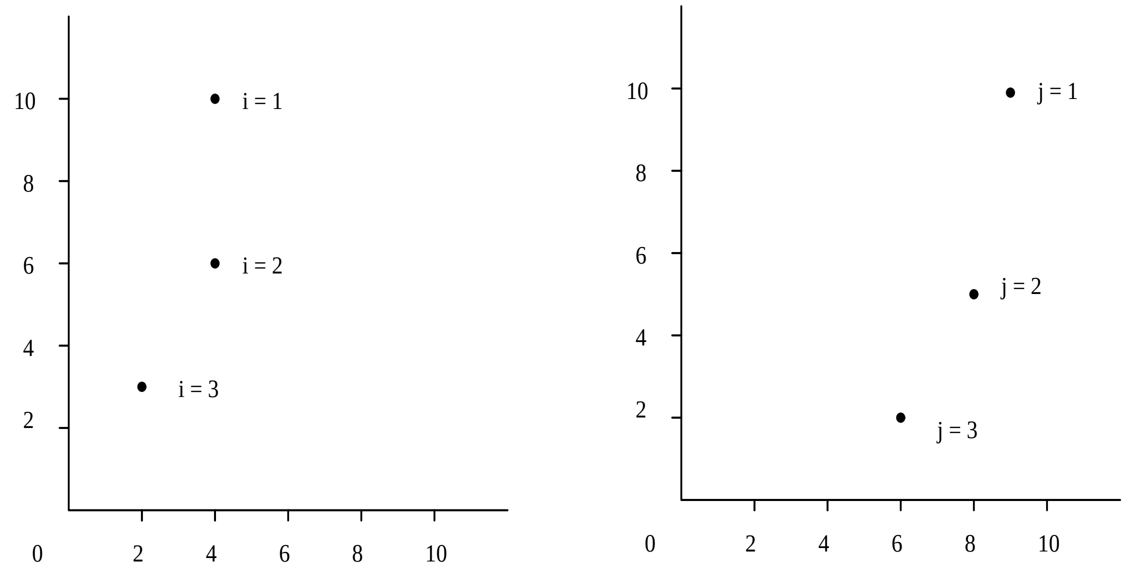
$\overline{X_q^k X_r^{k+1}}$ is the vector from point q in frame k to the point r in frame $k+1$; and $||X||$ denotes the magnitude of the vector X.

The proximal uniformity function obeys the following criteria.

- Speed doesn't change much between two successive frames.
- Direction doesn't change much between two successive frames.
- Displacement of a point between two successive frames tends to be small.

In proximal uniformity function the first term represents a *relative* change in velocity, while the second term denotes a the relative displacement. The second term forces the proximal matches, while the first term leads to smooth and uniform trajectories. Notice that the numerator in both terms represents an *absolute* quantity, for example the numerator of the first term represents a absolute change in velocity of a point q in frame k. While, the denominators denote the sum of absolute quantities for all possible matches. Hence, the ratio gives the relative measure of quantities. Since, a change in velocity is a vector quantity, the magnitude in the first term incorporates both the change in *speed* and *direction.*

In this formulation it is assumed that Φ^1, an initial correspondence is to be known. Φ^k is determined such that $\sum \delta(X_p^{k-1}, X_q^k, X_r^{k+1})$ is minimized. For a smoothness based method to be meaningful, the initial correspondence should be known. Given the correct initial correspondence, the algorithm will correctly grow the trajectories. Applying the smoothness constraint alone without knowing any detail about the initial movement of the points may in many cases lead to false trajectory sets.



(a)

((4,10) ((5,0),.7)((4,-5),.25)((2,-8),.05))
((4,6) ((5,4),.5)((4,-1),.3)((2,-4),.2))
((2,3) ((7,7),.3)((6,2),.35)((4,-1),.2))

(b)

$$q^0_{ij} = \begin{bmatrix} .3 & .2 & 0 \\ .35 & .9 & .25 \\ 0 & .5 & .3 \end{bmatrix}, \quad \tilde{P}^1_{ij} = \begin{bmatrix} .84 & .225 & .015 \\ .675 & .9 & .21 \\ .09 & .63 & .24 \end{bmatrix}, \quad P^1_{ij} = \begin{bmatrix} .76 & .2 & .013 \\ .378 & .5 & .11 \\ .09 & .65 & .25 \end{bmatrix}.$$

(c)

Figure 5.7: Demonstration of Baranard-Thompson algorithm with $A = .3,\ B = 3,\ V_{max} = 2,\ D_{max} = 4$.

Algorithm A

1. For $k = 2$ to $n - 1$ do
   (a) Construct M an $(m*m)$ matrix, with the points from kth frame along the rows and points from $(k+1)$th frame along the columns. Let $M[i,j] = \delta(X_p^{k-1} X_i^k X_j^{k+1})$, when $\Phi^{k-1}(p) = i$.
   (b) for $a = 1$ to m do
      i. Identify the minimum element $[i, l_i]$ in each row i of M.
      ii. Compute *priority matrix* B, such that $B[i, l_i] = \sum_{j=1, j \neq l_i}^{m} M[i,j] + \sum_{k=1, k \neq i}^{m} M[k, l_i]$ for each i.
      iii. Select $[i, l_i]$ pair with highest *priority* value $B[i, l_i]$, and make $\Phi^k(i) = l_i$.
      iv. Mask row i and column l_i from M.

Figure 5.8: Motion correspondence using multiple frames.

A non-iterative greedy algorithm **A** is given in Figure 5.8. This algorithm assigns correspondence of points in one frame with the points in next frame, keeping the overall proximal smoothness function as close to the minimum as possible in addition to being fair to each individual assignment.

In this algorithm when the minimum along the rows is considered it could happen that more than one minimum lies along the same column j. That is, more than one point in frame k competes for point j in the $(k+1)$th frame. To get a one to one onto mapping, we should choose only one of these. However, this scheme should not just choose the minimum possible combination quantitatively, but it should prefer a combination where each individual correspondence is fairly good. The correspondence from frame k to $k+1$ involves m points. The minimum correspondence could be very favorable to some $(m-1)$ points and not favorable for the mth point. The algorithm should prefer a correspondence which is equally favorable to all points; at the same time we should not end up with a very high proximity path uniformity function. The algorithm was designed to take care of these conditions.

The rationale behind the priority measure is that if $[i,j]$ is not assigned, any other element along the ith row or jth column can get assigned. Assuming they are all equally probable, their average value is a good measure for selecting the order of assignments. The algorithm should choose the one with the highest priority measure and assign that first. Priority measure is a rough estimate of the alternate assignment to $[i,j]$.

This algorithm has the nice property that it will pick the *least cost assignment* if there are just two points in the frame. Consider the matrix M, with $M[1,1] = 0.6$, $M[1,2] = 0.3$, $M[2,1] = 0.7$, and $M[2,2] = 0.2$. Minimum along row 1 is element $[1,2]$ with value 0.3, while the minimum along row 2 is element $[2,2]$ with value 0.2. Therefore, $B[1,2] = (0.6 + 0.2) = 0.8$, and $B[2,2] = (0.7 + 0.3) = 1.0$. Now, $B[2,2] > B[1,2]$, hence we choose correspondence $(2,2)$ first. Then, mask row 2 and column 1 with a high value. Next we pick the only assignment possible $[1,1]$. For this assignment $\delta = M[1,1] + M[2,2] = 0.6 + 0.2 = 0.8$, which is the least possible for this configuration.

The results obtained with this algorithm are shown in Figure 5.9.

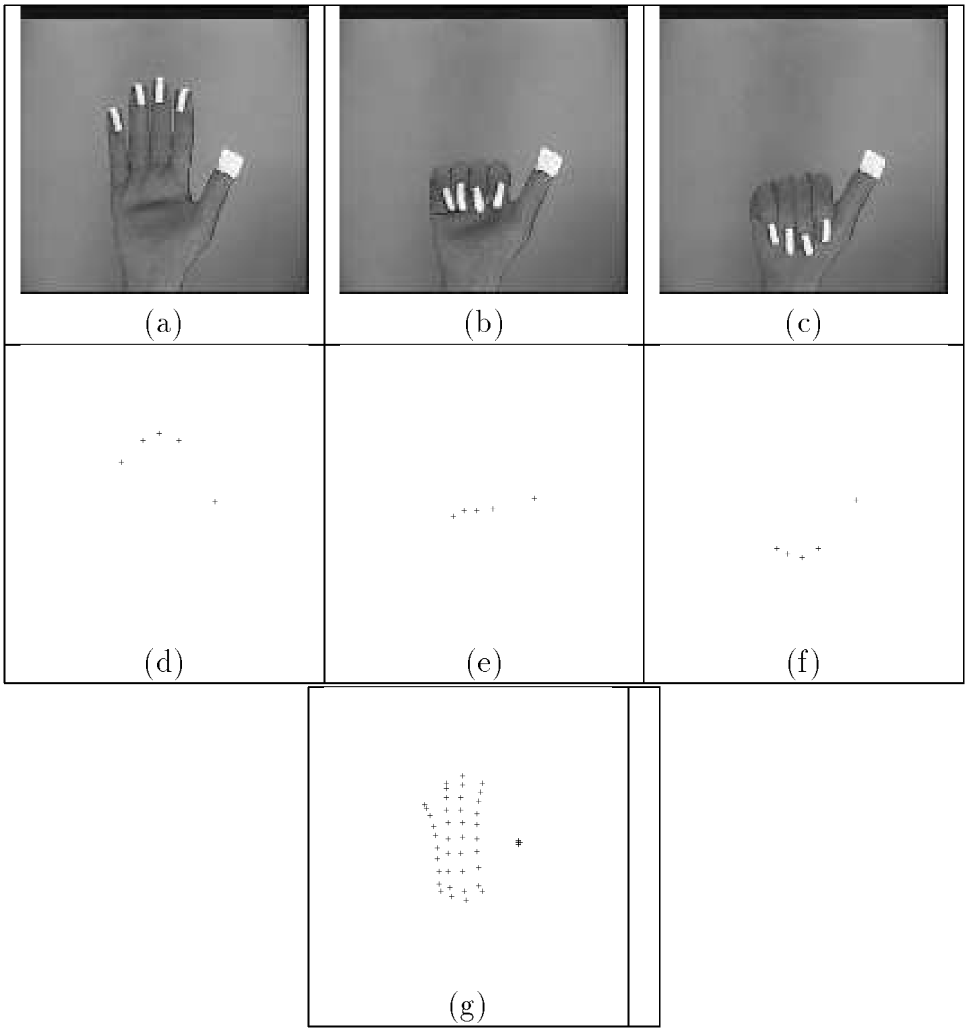


Figure 5.9: Trajectory generation. (a)-(c) Image sequence – selected images shown. (d)-(f) Respective fingertip points for images. (g) Fingertip trajectories.

5.5 Structure From Motion

The structure from motion (SFM) method in computer vision infers the physical properties of the objects present in the scene, such as their three-dimensional structure and motion, given a series of two-dimensional projections. There has been significant interest over the last decade in the Computer Vision community in the structure from motion theory. There are two classes of methods for SFM: the displacement methods and the instantenous methods. In the displacement methods, three dimensional coordinates of points on the moving objects and their three dimensional motion is recovered from a sequence of frames. This problem is formulated in terms of systems of non-linear (linear) equations given 2-D positions of moving points among frames. Interesting theoretical work related to the number of points required for a solution, the uniqueness of such a solution, and the effect of noise on the solution has been studied. In the instantaneous methods the optical flow field is used to recover the 3-D motion and depth values. Previous approaches in this class have dealt with the simplified problems involving some assumptions related to the motion and objects, e.g., the assumption of translation motion only, rotation motion only, known depth of objects, and planar surfaces. Recently, Heeger and Jepson [7] have proposed a general method for computing 3-D motion (rotation and translation) and depth from optical flow. Their method first computes translation, followed by rotation, and then depth. Heeger and Jepson method can be described as follows.

Assume that the motion of a point (X, Y, Z) lying on an object can be described by $\vec{V} = -\vec{T} - \vec{\omega} \times \vec{r}$, where $\vec{V}$ is the 3-D velocity vector, $\vec{T} = (T_x, T_y, T_z)$ is the translation vector, $\vec{\omega} = (\omega_x, \omega_y, \omega_z)$ is the rotation vector, $\vec{r}$ is vector from the origin to the point (X, Y, Z), and $\times$ represents the vector cross product. The optical flow (u, v) under perspective transform (assuming focal length =-1) is given by:

$$u = (-\frac{T_x}{Z} - \omega_y + \omega_z y) - x(-\frac{T_z}{Z} - \omega_x y + \omega_y x), \tag{5.27}$$

$$v = (-\frac{T_y}{Z} - \omega_z x + \omega_x) - y(-\frac{T_z}{Z} - \omega_x y + \omega_y x). \tag{5.28}$$

This equation can be written as

$$\vec{\theta}(x, y) = p(x, y)\mathbf{A}(x, y)\vec{T} + \mathbf{B}(x, y)\vec{\omega}, \tag{5.29}$$

where $p(x, y) = 1/Z(x, y)$, is inverse depth, $\mathbf{A}(x, y) = \begin{bmatrix} -1 & 0 & x \\ 0 & -1 & y \end{bmatrix}$, and $\mathbf{B}(x, y) = \begin{bmatrix} xy & -(1+x^2) & y \\ 1+y^2 & -xy & -x \end{bmatrix}$.

The matrices $\mathbf{A}$ and $\mathbf{B}$ depend only on the image position, not on the unknowns. For each point in the image, a separate equation can be written in this form, and can be combined together into one matrix equation as:

$$\vec{\theta} = \mathbf{A}(\mathbf{T})\vec{p} + \mathbf{B}\vec{\omega}, \tag{5.30}$$

$$\vec{\theta} = \mathbf{C}(\mathbf{T})\vec{q}, \tag{5.31}$$

where $\vec{\theta}$ contains the image velocities for all the image points, and $\vec{p}$ contains the depths.

$$\mathbf{A}(T) = \begin{bmatrix} \mathbf{A}(x_1, y_1)(\mathbf{T}) & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \mathbf{A}(x_n, y_n)(\mathbf{T}) \end{bmatrix}$$

is obtained by collecting together into a single matrix $\mathbf{A}(x, y)T$ for each point, and

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}(x_1, y_1) \\ \vdots \\ \mathbf{B}(x_n, y_n) \end{bmatrix}$$

is obtained by collecting the $\mathbf{B}(x, y)$ matrices, $\vec{q}$ is obtained by collecting together into one vector the unknown depths and rotational velocities, and

$$\mathbf{C}(T) = \begin{bmatrix} | & | \\ \mathbf{A}(\mathbf{T}) & \mathbf{B} \\ | & | \end{bmatrix}$$

is obtained by placing the columns of $\mathbf{B}$ along the columns of $\mathbf{A}$.

The least-squares estimate for translation can be used which minimizes the following expression over all candidate translations, rotations, and depth values:

$$E(\vec{T}) = ||\vec{\theta} - \mathbf{C}(\vec{T})\vec{q}||^2.$$

Minimizing this expression over all $\vec{T}$ and $\vec{q}$ is equivalent to the following function over $\vec{T}$:

$$R(\vec{T}) = ||\vec{\theta}\mathbf{C}^{\perp}(\vec{T})||^2, \tag{5.32}$$

where $\mathbf{C}^{\perp}(\vec{T})$ is the orthonormal basis for the orthogonal complement of $\mathbf{C}(\vec{T})$.

Once $\vec{T}$ has been computed, the rotational velocity can be determined. Consider a a unit vector $\vec{d}(x, y, \vec{T})$ such that: $\vec{d}^t(x, y, \vec{T})\mathbf{A}(x, y)\vec{T} = 0$. Multiplying by $\vec{d}^t(x, y, \vec{T})$ on both sides of equation 5.29 eliminates the dependence on $p(x, y)$;

$$\vec{d}^t(x, y, \vec{T})\vec{\theta}(x, y) = \vec{d}^t(x, y, \vec{T})\mathbf{B}(x, y)\vec{\omega}.$$

A large number of flow vectors can be used in a least-squares estimate for ω.

$$\vec{d}^t(x_1, y_1, \vec{T})\vec{\theta}(x_1, y_1) = \vec{d}^t(x_1, y_1, \vec{T})\mathbf{B}(x_1, y_1)\vec{\omega} \tag{5.33}$$

$$\vdots = \vdots \tag{5.34}$$

$$\vec{d}^t(x_n, y_n, \vec{T})\vec{\theta}(x_n, y_n) = \vec{d}^t(x_n, y_n, \vec{T})\mathbf{B}(x_n, y_n)\vec{\omega}. \tag{5.35}$$

or

$$s(x_1, y_1) = \mathbf{D}(x_1, y_1)\vec{\omega} \tag{5.36}$$

$$\vdots = \vdots \tag{5.37}$$

$$s(x_n, y_n) = \mathbf{D}(x_n, y_n)\vec{\omega} \tag{5.38}$$

where $s(x, y)$ is 1×1, $D(x, y)$ is 1×3. The above equations can be written as:

$$s = \mathbf{D}\vec{\omega} \tag{5.39}$$

where s is $n \times 1$ vector and $\mathbf{D}$ is $n \times 3$ matrix. We can compute $\vec{\omega}$ by puseudo inverse method.

Finally, once the translational and rotational velocities are both known, the equations 5.27 and 5.28 can be used to solve for an unknown depth at each image point using least squares method.

5.6 Exercises

1. Derive equations 5.7 and 5.8.
2. Derive equation 5.17 from equation 5.5.
3. Derive equation 5.20.
4. Derive equations 5.27–5.32.

Chapter 6

Stereo and Shape From Shading

6.1 Introduction

The world is three-dimensional, but its images are two-dimensional, so one dimension is lost during the projection process[1]. One important task in computer vision is to recover the third dimension from single or multiple images. There are several cues for recovering 3-D information from 2-D images which humans use frequently, and which are being studied in machine vision research. These cues include stereo, shading, texture, and motion.

The recovered 3-D shape can be expressed in several ways:

- Depth Z, the distance of a point on the object,
- Surface normal (n_x, n_y, n_z), the orientation of a vector perpendicular to the tangent plane on the object surface,
- Surface gradient $(p, q) = (\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y})$, the rate of change of depth in the x and y directions,
- Surface slant σ, and tilt τ, such that $(n_x, n_y, n_z) = (\rho \sin\sigma \cos\tau, \rho \sin\sigma \sin\tau, \rho \cos\sigma)$.

In this chapter, we will discuss stereo, photometeric stereo, and shape from shading. In stereo, two images (left and right) are used to recover 3-D shape. The image of an object in 3-D is shifted with respect to its image in the right image. This shift, which is inversely proportional to the 3-D, is used to recover the 3-D shape. In shape from shading, gray level variations in a single image are used to recover 3-D shape. In photometeric stereo, multiple images taken by different light sources are used to generate over constrained system for computing 3-D shape.

6.2 Stereo

6.2.1 Stereo Geometry

A simple stereo geometry is shown in Figure 6.1. C_1 is the center of lens for the left camera, C_2 is the center of lense for the right camera, f is the focal length, Z is the distance in the Z direction (depth), x_1 is the image coordinate in the left image, x_2 is the image coordinate in

[1]

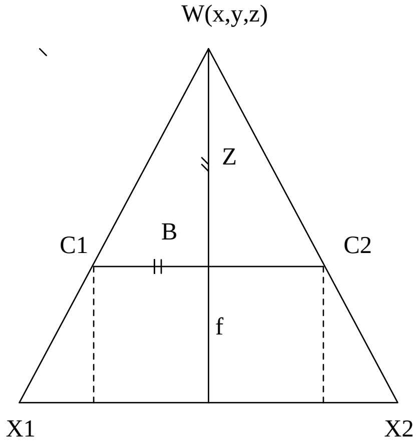


Figure 6.1: Stereo geometry.

the right camera. It is clear from figure that the two triangles $\triangle W x_1 x_2$ and $\triangle W C_1 C_2$ are equivalent. Therefore, we have

$$\frac{Z+f}{Z} = \frac{x_1 + x_2 + B}{B}, \tag{6.1}$$

or,

$$Z = \frac{fB}{x_1 + x_2}, \tag{6.2}$$

where $x_1 + x_2$ is the disparity. For a given stereo system, the focal length f of the camera, and baseline B are fixed and known. If the disparity $x_1 + x_2$ can be computed from left and right images, the depth Z can be determined using the above equation. Note, that the disparity is inversely proportional to the depth. The objects close to the camera produce large disparity, while the objects farther away form the camera produce small disparity.

6.2.2 Steps in Token Based Stereo

There are three main steps in stereo:

1. Token detection
2. Correspondence
3. Surface interpolation.

In the first step, tokens are detected in gray level image pairs. The tokens can be edges, corners, interest points, etc. The depth is computed only at the tokens. The correspondence step is the main step in stereo, where the tokens from the left image are matched with the tokens in the right image. Due to the *epipolar* constraint, matching in stereo is limited to only one dimension. therefore, there can be a large number of possible mappings between left and right image tokens, the constraints are used to limit the search space. In the third step, the depth values at the tokens are used to determine depth at the remaining pixels through interpolation.

B	B	W	B	W

(a)

W	B	B	B	W

(b)

	1	2	3	4	5
d =2	1	1	1		
d =1	1	1			
d =0		1		1	1
d =-1				1	
d =-2			1	1	

(c)

Figure 6.2: Demonstration of Marr-Poggio algorithm. (a) Left image, (b) Right image, and (c) Initial array $C^0(x,d)$.

6.2.3 Marr-Poggio Algorithm

The Marr-Poggio[15] algorithm is one of the first stereo algorithms, and it is easy to explain. This algorithm was motivated by the human vision system. The domain of this algorithm is the random dot stereogram, where each pixel in the image is randomly black or white. The algorithm uses the following constraints.

Compatibility White dots should match with the white dots, and black dots should match with the black dots.

Continuity The depth should be continuous, and neighboring pixels should have similar depth.

Uniqueness One dot from the left image should match to only one dot from the right image.

The initial correspondence, $C^0(x,d)$, is obtained using the compatibility constraint. The continuity and uniqueness constraints are used in the later iterations. The compatibility constraint can be enforced by applying the *exclusive nor* operation between the tokens in the left and right images. One dimensional stereo pair is shown in Figure 6.2.a–b. For simplicity, we assume that the image size is only 5. The white dots are shown by 'W's', and the black dots are shown by 'B's'. In Figure 6.2.c we show the array $C^0(x,d)$, computed by applying the compatibility constraint to the images shown in Figure 6.2.a–b.

From array C^0, the arrays C^1, C^2, …, C^n are iteratively computed using the following scheme, which enforces the continuity and uniqueness constraints.

$$C^n(x,d) = \left\{ \begin{array}{ll} 1 & \text{if} \left\{ \sum_{a=-w, a\neq 0}^{w} C^{n-1}(x+a,d) - \epsilon \sum_{i=-D_{\max}, i\neq 0}^{D_{max}} C^{n-1}(x \pm i, d+i) + C^0(x,d) \right\} > T \\ 0 & \text{otherwise} \end{array} \right\} \quad (6.3)$$

1. Compute the initial correspondence using the *compatibility* constraint as follows:

$$C^0(x,y,d) = L(x,y) \oslash R(x+d,y),$$

   where $\oslash$ is the exclusive nor operation.

2. Iteratively compute C^n as follows:

   Let

$$SUM = \left\{ \sum\sum_{|x-p|+|y-q|\leq w} C^{n-1}(x+p,y+q,d) - \epsilon \sum_{i=-D_{\max}, i\neq 0}^{D_{max}} C^{n-1}(x\pm i,y,d+i) + C^0(x,y,d) \right\}$$

$$C^n(x,y,d) = \begin{cases} 1 & \text{if } SUM > T \\ 0 & \text{otherwise} \end{cases}$$

Figure 6.3: Marr-Poggio stereo algorithm.

		*		
	*	*	*	
*	*	+	*	*
	*	*	*	
		*		

Figure 6.4: Excitatory neighborhood used in Marr-Poggio algorithm.

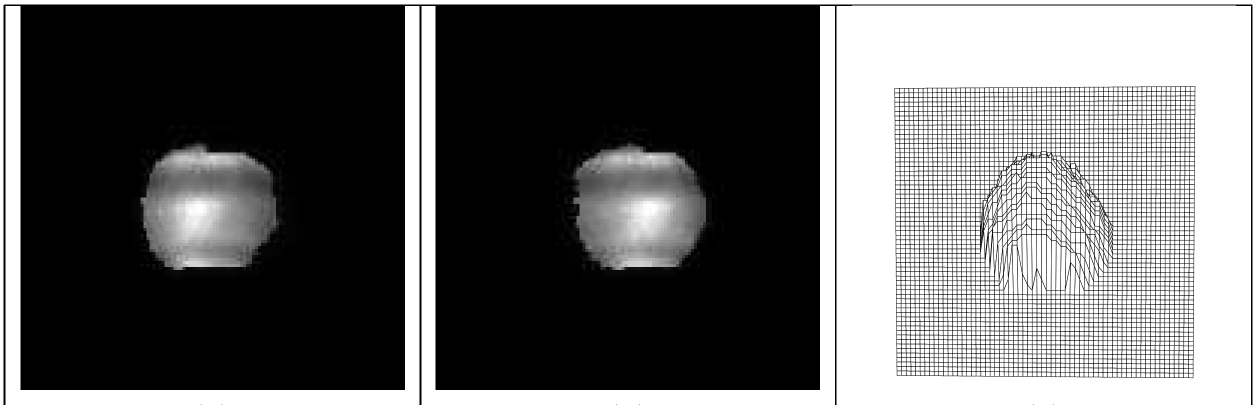
where, D_{max} is the maximum possible disparity, ϵ is constant, T is the threshold, and w is constant. In the above scheme, the first term on the right side is called the *excitatory* term, which computes the support a given match gets from its neighbors with the same disparity. The second term with the minus sign is the called *inhibitory* term, which computes the penalty when the continuity constraint is violated. The third term is the initial array C^0, which is added to speed up the convergence.

The Marr-Poggio algorithm for 2-D is similar to 1-D, and is given in Figure 6.3. The 2-D excitatory neighborhood used in the implementation of the Marr-Poggio algorithm is shown in Figure 6.4. The support from the pixels shown by '*' in a circle of diameter of 5 pixels around the central pixel shown by '+' is used in the excitatory term in equation 2. The typical values for ϵ, T, w respectively are 2, 4, 2.

6.2.4 Correlation Based Stereo Methods

In the token based stereo methods, the depth is computed only at token points. In order to obtain the dense depth map it is necessary to apply the interpolation step. In the correlation based stereo method, depth is computed at each pixel. A gray level patch around a pixel in the left image is correlated with the corresponding pixel in the right image. The disparity for the best match is determined. In this method the disparity map, $D(x,y)$, is computed as follows: Let

$$CR(x,y,d) = \sum_{w_x=-s}^{s} \sum_{w_y=-s}^{s} L(x+w_x, y+w_y) \times R(x+w_x+d, y+w_y)$$



(a) (b) (c)

Figure 6.5: Results for the Tomato images. (a) Left stereo gray level image. (b) Right stereo gray level image. (c) A 3D plot of the estimated depth map from SSD stereo matching algorithm. The image size is 128 by 128, the window used for SSD is 5×5, and the disparity range is 12.

$$D(x,y) = d \;\; s.t. \;\; \text{CR(x, y, d)is maximum for } - D_{max} \leq d \leq D_{max}, \tag{6.4}$$

where $L(x,y)$ and $R(x,y)$ are respectively the left and right images. In order to compute a disparity at point (x,y) a $(2s+1) \times (2s+1)$ window of gray levels in the left image is matched with the gray levels in the right image. The resultant disparity from all possible disparities, $d = \ldots, -2, -1, 0, 1, 2, \ldots$, is the one which produces the maximum correlation value in the above equation. In the above method the correlation value will depend on the local changes in $R(x,y)$. It is better to use normalized correlation as shown below: Let

$$NCR(x,y,d) = \frac{\sum_{w_x=-s}^{s} \sum_{w_y=-s}^{s} L(x+w_x, y+w_y) \times R(x+w_x+d, y+w_y)}{\sqrt{\sum_{w_x=-s}^{s} \sum_{w_y=-s}^{s} \left(R(x+w_x+d, y+w_y)\right)^2}}$$

$$D(x,y) = d \;\; s.t. \;\; NCR(x,y,d) \;\; is \; maximum \; for \; - D_{max} \leq d \leq D_{max}. \tag{6.5}$$

If the right image window is a scaled version of the left image window i.e. $R = cL$, for constant c, then normalized correlation will have a maximum value of $\sqrt{\sum\sum L^2}$.

Another possibility is to use the sum of squared differences (SSD) as shown below:

$$SSD(x,y,d) = \sum_{w_x=-s}^{s} \sum_{w_y=-s}^{s} \left(L(x+w_x, y+w_y) - R(x+w_x+d, y+w_y)\right)^2$$

$$D(x,y) = d \;\; s.t. \;\; SSD(x,y,d) \;\; is \; minimum \; for \; - D_{max} \leq d \leq D_{max}. \tag{6.6}$$

The normalized correlation and the SSD give similar results. The results for SSD stereo algorithm are showing in figures 6.5, 6.6 and 6.7.

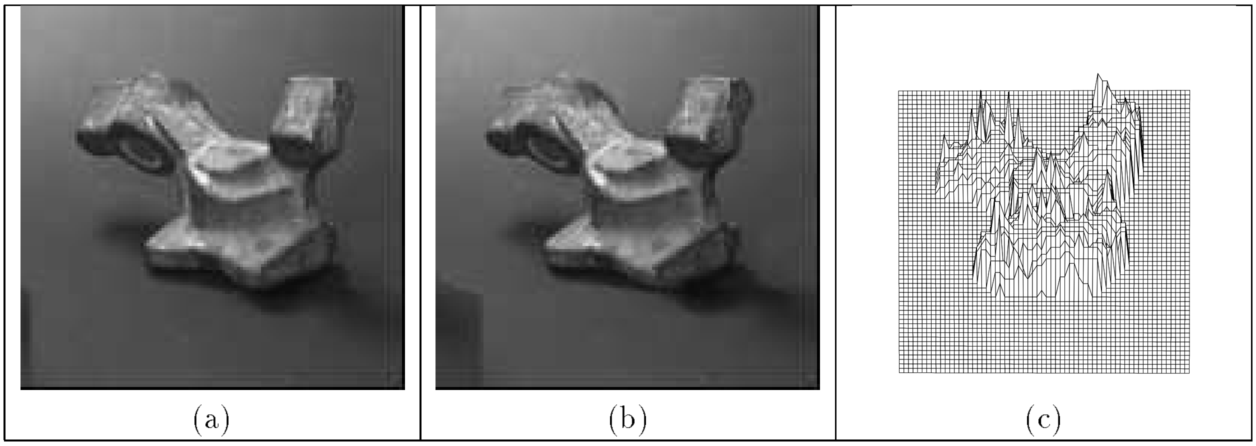


Figure 6.6: Results for the Renault images. (a) Left stereo gray level image. (b) Right stereo gray level image. (c) A 3D plot of the estimated depth map from SSD stereo matching algorithm. The image size is 128 by 128, the window used for SSD is 5×5, and the disparity range is 9.

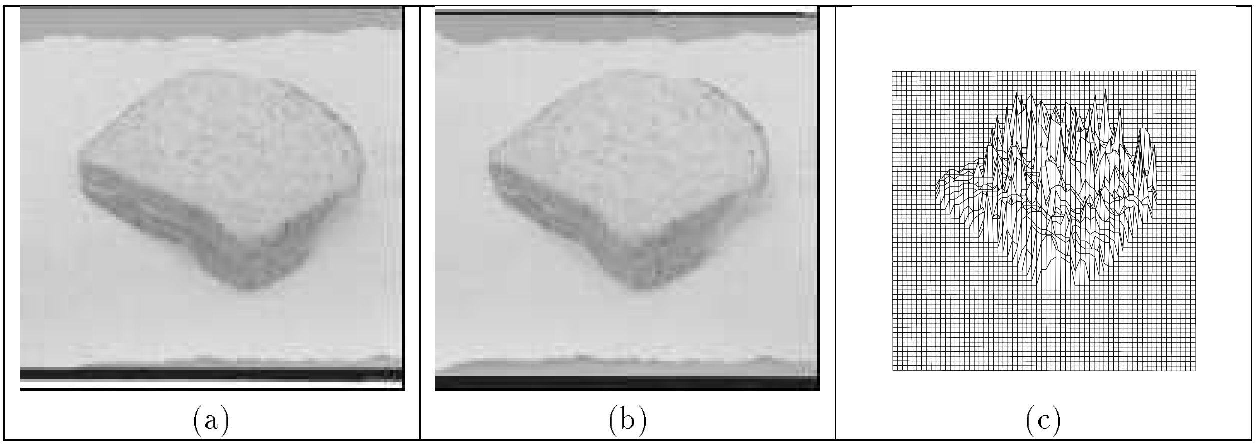


Figure 6.7: Results for Sandwich Renault images. (a) Left stereo gray level image. (b) Right stereo gray level image. (c) A 3D plot of the estimated depth map from SSD stereo matching algorithm. The image size is 128 by 128,the window used for SSD is 5×5, and the disparity range is 13.

6.2.5 Barnard's Stereo Algorithm

In Barnard's stereo approach the problem is to find an assignment of disparities, $D(x, y)$, such that two criteria, *similar intensity* and *smoothness*, are satisfied:

$$E(x,y) = \sum_{w_x=-1}^{1} \sum_{w_y=-1}^{1} \|L(x+w_x, y+w_y) - R(x+w_x+D(x,y), y+w_y)\| + \lambda \| \bigtriangledown D(x,y)\| \quad (6.7)$$

where L and R are the left and right images, $D(x, y)$ is the disparity map, the $\bigtriangledown$ operator computes the sum of the absolute differences between disparity $D(x, y)$ and its eight neighbors, and λ is a constant. For a 128×128 image, and a disparity range of 10 pixels, there are 10^{16384} possible disparity assignments, which results in combinatorial explosion. Barnard uses a simulated annealing to solve this problem. The algorithm is as follows:

1. Select a random state S.
2. Select high temperature T.
3. While T > 0.
   (a) Select S'
   $$\Delta E \leftarrow E(S') - E(S).$$
   (b) if $\Delta E \leq 0$ then $S \leftarrow S'$
   (c) else $P \leftarrow \exp^{\frac{-\Delta E}{T}}$, $X \leftarrow rand(0, 1)$,
   if $X < P$ then $S \leftarrow S'$
   (d) if no decrease in E for several iterations then lower T.

The results are shown in figure 6.8.

6.3 Shape From Shading

In shape from shading, given a single gray level image, the aim is to recover the light source direction and surface shape at each pixel. Since images of most surfaces in the real world can be approximated by Lambertian reflectance, the majority of shape from shading methods use the Lambertian reflectance model. The important parameters in Lambertian reflectance are albedo and illuminant directions. Commonly, the albedo is assumed to be constant.

The reflectance function for Lambertian surfaces is modeled as follows:

$$\begin{aligned} I(x,y) &= R(p,q), && (6.8)\\ &= \frac{1+pp_s+qq_s}{\sqrt{1+p^2+q^2}\sqrt{1+p_s^2+q_s^2}}, && (6.9)\\ &= \frac{\cos\sigma + p\cos\tau\sin\sigma + q\sin\tau\sin\sigma}{\sqrt{1+p^2+q^2}}, && (6.10) \end{aligned}$$

where $I(x, y)$ is the gray level at pixel (x, y), $p = \frac{\partial Z}{\partial x}$, $q = \frac{\partial Z}{\partial y}$, $p_s = \frac{\cos\tau\sin\sigma}{\cos\sigma}$, $q_s = \frac{\sin\tau\sin\sigma}{\cos\sigma}$, τ is the tilt of the illuminant and σ is the slant of the illuminant. In this model, the surface orientation is represented by, surface gradient, (p, q), and illuminant is represented by slant and tilt (τ, σ) (equation 6.10), or gradient (p_s, q_s) (Equation 6.9) as shown in Figure 6.9.

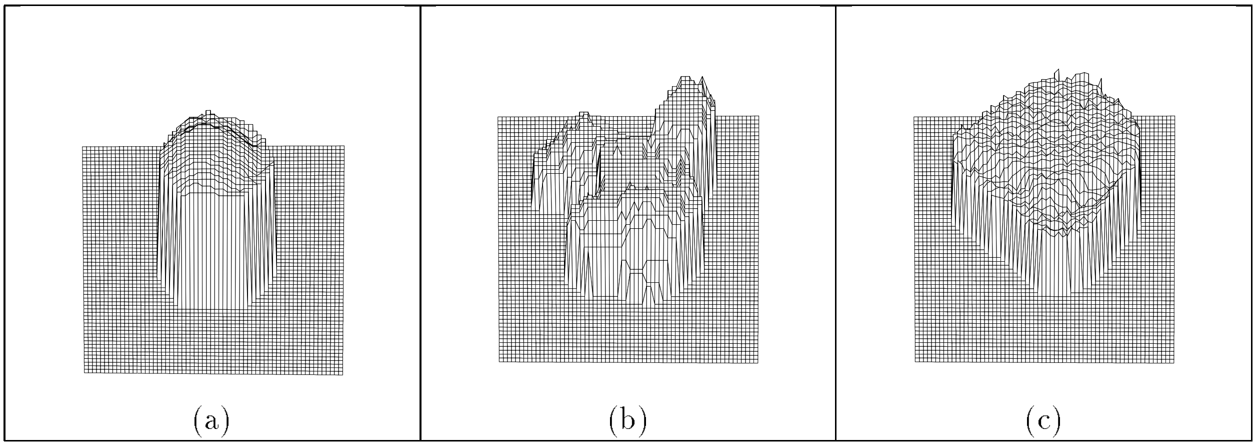


Figure 6.8: Results for Barnard's stereo algorithm. The inital temperature is 100, and it is decreased every time by 10%. The value of λ is 5. (a) A 3D plot of the estimated depth map for Tomato image pairs. (b) A 3D plot of the estimated depth map for Reanult image pairs. (c) A 3D plot of the estimated depth map for Sandwich image pairs.

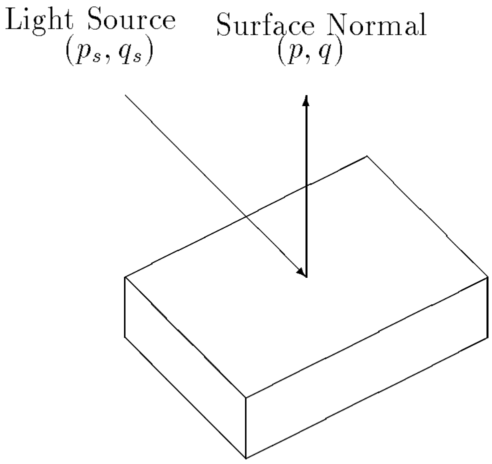


Figure 6.9: Reflectance from a surface patch.

6.3.1 Source From Shading

Most of the shape from shading algorithms require known light source directions. Since the light source is usually assumed to be at infinity, the light source orientation is constant for all of the surface points in the image, one image can provide enough information to estimate the source. There are two ways to describe a light source direction: One uses a 3-D vector, the other uses the two degrees of freedom in 3-D space – slant and tilt. If the image plane is parallel to the X-Y plane, slant is the angle the illuminant vector makes with the Z-axis, and tilt is the angle the image plane component of the illuminant vector makes with the X-axis.

Several techniques to estimate source orientation have been developed. The first one, by Pentland [17], estimates the light source direction from the distribution of image derivatives. By assuming an umbilical surface and isotropic surface normal, a maximum-likehood analysis was performed to estimate the slant and tilt angles of the light source.

The brightness equation is given by

$$I(x,y) = B\vec{N}\cdot\vec{S}$$

or

$$I(x,y) = B(n_x s_x + n_y s_y + n_z s_z)$$

Here, B is a constant including the albedo term. For sphere, $z(x,y) = \sqrt{R^2 - x^2 - y^2}$, $z_x = \frac{x}{z}$, $z_y = \frac{y}{z}$, hence $(n_x, n_y, n_z) = \frac{1}{R}(x,y,z)$. The above equation becomes:

$$I(x,y) = B(\frac{x}{R}s_x + \frac{y}{R}s_y + \frac{z}{R}s_z)$$

Taking the directional derivative of the above equation in the direction θ we get:

$$dI_\theta = B(\frac{1}{R}s_x\cos\theta + \frac{1}{R}s_y\sin\theta)$$

Now taking the mean on both sides:

$$d\bar{I}_\theta = B(\bar{k}s_x\cos\theta + \bar{k}s_y\sin\theta)$$

or

$$d\bar{I}_\theta = (\tilde{s_x}\cos\theta + \tilde{s_y}\sin\theta)$$

where $\tilde{s_x} = B\bar{k}s_x$ and $\tilde{s_y} = B\bar{k}s_y$. Repeating the above process in m different directions $(\cos\theta_i, \sin\theta_i)(i = 1, ..., m)$, the regression model can be described as:

$$\begin{pmatrix} d\bar{I}_{\theta_1} \\ d\bar{I}_{\theta_2} \\ \vdots \\ d\bar{I}_{\theta_m} \end{pmatrix} = \begin{pmatrix} \cos\theta_1 & \sin\theta_1 \\ \cos\theta_2 & \sin\theta_2 \\ \vdots & \vdots \\ \cos\theta_m & \sin\theta_m \end{pmatrix} \begin{pmatrix} \tilde{s_x} \\ \tilde{s_y} \end{pmatrix},$$

where $d\bar{I}_i$ is the average of the intensity change along the image direction $(\cos\theta_i, \sin\theta_i)$,

A typical choice for $(\cos\theta_i, \sin\theta_i)$ is along the eight directions in the image grid: two in the horizontal direction, two in the vertical direction, and four along the diagonals.

Solving the above system by least squares, we get:

$$\begin{pmatrix} \tilde{s_x} \\ \tilde{s_y} \end{pmatrix} = (\beta^T \beta)^{-1} \beta^T \begin{pmatrix} d\bar{I}_{\theta_1} \\ d\bar{I}_{\theta_2} \\ \vdots \\ d\bar{I}_{\theta_m} \end{pmatrix},$$

where β is the matrix of directions $(\cos\theta_i, \sin\theta_i)$.

The tilt, τ_S, of the light source direction is given by:

$$\tau_S = \arctan(\frac{\tilde{s_y}}{\tilde{s_x}}),$$

and the slant, σ_S, of the light source direction is:

$$\sigma_S = \arccos\sqrt{1 - {s_x}^2 - {s_y}^2}.$$

Taking the expected value of the square of intensity derivative $E[dI^2]$, and cancelling out the common terms between $E[dI]^2$ and $E[dI^2]$ by subtracting one from the other, we have the relation

$$E[dI^2] - E[dI]^2 = B^2\bar{k}^2.$$

Since $\tilde{s_x} = B\bar{k}s_x$, $\tilde{s_y} = B\bar{k}s_y$, if we introduce $k = B\bar{k} = \sqrt{E[dI^2] - E[dI]^2}$, the equation for the slant of the light source can be simplified into:

$$\sigma_S = \arccos\sqrt{1 - \frac{\tilde{s_x}^2 + \tilde{s_y}^2}{k^2}}.$$

Pentland also extended this approach to the Fourier domain [18].

6.3.2 Horn and Ikeuchi Method

Horn and Ikeuchi [11] minimize the following error function to estimate the shape (p, q):

$$E = (I(x,y) - R(p,q))^2 + \lambda(p_x^2 + p_y^2 + q_x^2 + q_y^2). \tag{6.11}$$

The first term is the reflectance constraint obtained from equation 6.8, and the second term is the smoothness of the surface constraint. Differentiating the above equation with respect to p and q, equating the resulting equations to zero, and solving them for p and q we get:

$$p(x,y) = p_{av}(x,y) + T(x,y,p,q)\frac{\partial R}{\partial p}, \tag{6.12}$$

$$q(x,y) = q_{av}(x,y) + T(x,y,p,q)\frac{\partial R}{\partial q}, \tag{6.13}$$

where $T(x,y,p,q) = \frac{I(x,y)-R(p,q)}{\lambda}$. The iterative algorithm for shape from shading using the above equations is given in Figure 6.10.

1. $k = 0$, pick $p^0(x, y)$ and $q^0(x, y)$ near boundary.

2. $k = k + 1$;

$$p^k(x, y) = p_{av}^{k-1}(x, y) + T(x, y, p, q)\frac{\partial R}{\partial p} \quad (6.14)$$

$$q(x, y) = q_{av}^{k-1}(x, y) + T(x, y, p, q)\frac{\partial R}{\partial q} \quad (6.15)$$

3. Stop if the error is small.

Figure 6.10: Horn-Ikuchi algorithm for shape form shading.

6.3.3 Pentland Method

Pentland [16] proposed a local algorithm based on the linearity of the reflectance map in the surface gradient (p, q), which greatly simplifies the shape from shading problem. By taking the Taylor series expansion of the reflectance function R, given in equation 6.8, about $p = p_0$, $q = q_0$, up through the first order terms, we have

$$I(x, y) = R(p_0, q_0) + (p - p_0)\frac{\partial R}{\partial p}(p_0, q_0) + (q - q_0)\frac{\partial R}{\partial q}(p_0, q_0). \quad (6.16)$$

For Lambertian reflectance (equation 6.9), the above equation at $p_0 = q_0 = 0$, reduces to

$$I(x, y) = \cos\sigma + p\cos\tau\sin\sigma + q\sin\tau\sin\sigma. \quad (6.17)$$

Next, Pentland takes the Fourier transform of both sides of this equation. Since the first term on the right is a DC term, it can be dropped. Using the identities:

$$\frac{\partial}{\partial x}Z(x, y) \longleftrightarrow F_Z(\omega_1, \omega_2)(-i\omega_1) \quad (6.18)$$

$$\frac{\partial}{\partial y}Z(x, y) \longleftrightarrow F_Z(\omega_1, \omega_2)(-i\omega_2), \quad (6.19)$$

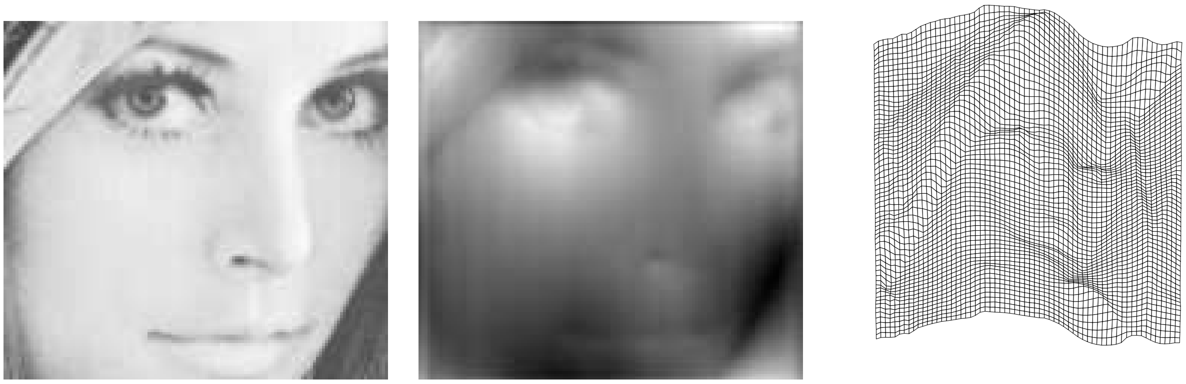
where F_Z is the Fourier transform of $Z(x, y)$, we get,

$$F_I = F_Z(\omega_1, \omega_2)(-i\omega_1)\cos\tau\sin\sigma + F_Z(\omega_1, \omega_2)(-i\omega_2)\sin\tau\sin\sigma,$$

where F_E is the Fourier transform of the image $E(x, y)$. The depth map $Z(x, y)$ can be computed by rearranging the terms in the above equation, and then taking the inverse Fourier transform.

$$F_Z(\omega_1, \omega_2) = \frac{F_I}{(-i\omega_1)\cos\tau\sin\sigma + (-i\omega_2)\sin\tau\sin\sigma}.$$

Typical results for Pentland's methods are shown in Figure 6.11. Other shape from shading algorithms use more complex models such as interreflections, changing albedos and specular reflectance [6]. These problems are more difficult and require more computational time to solve.



Original Lenna image | Pentland's Depth Map | 3-D Plot of Depth Map

Figure 6.11: Results for Pentland's Shape from Shading method.

6.3.4 Tsai and Shah Method

Tsai and Shah [24] instead of linearizing the reflectance in p and q, they use the discrete approximations for p and q in terms of Z, and then linearize the reflectance in $Z(x,y)$. They use the following discrete approximations for p and q

$$p = \frac{\partial Z}{\partial x} = Z(x,y) - Z(x-1,y), \tag{6.20}$$

$$q = \frac{\partial Z}{\partial y} = Z(x,y) - Z(x,y-1). \tag{6.21}$$

The reflectance equation (equation 6.9) can now be rewritten as:

$$f(Z(x,y)) = I(x,y) - R(Z(x,y) - Z(x-1,y), Z(x,y) - Z(x,y-1)) = 0 \tag{6.22}$$

By taking the Taylor series expansion of this function f about $Z(x,y) = Z^{n-1}(x,y)$,where $Z^{n-1}(x,y)$ is the depth at the $n-1$ iteration, up through the first order terms, we have

$$\begin{aligned} 0 &= f(Z(x,y)) \\ &\approx f(Z^{n-1}(x,y)) + (Z(x,y) - Z^{n-1}(x,y))\frac{df}{dZ(x,y)}(Z^{n-1}(x,y)). \end{aligned} \tag{6.23}$$

Then for $Z(x,y) = Z^n(x,y)$, the depth map at the n-th iteration, can be solved directly as follow:

$$Z^n(x,y) = Z^{n-1}(x,y) + \frac{-f(Z^{n-1}(x,y))}{\frac{df}{dZ}(Z^{n-1}(x,y))} \tag{6.24}$$

where

$$\frac{df(Z^{n-1}(x,y))}{dZ} = -1 * \left(\frac{(p_s + q_s)}{\sqrt{p^2+q^2+1}\sqrt{p_s^2+q_s^2+1}} - \frac{(p+q)(pp_s + qq_s + 1)}{\sqrt{(p^2+q^2+1)^3}\sqrt{p_s^2+q_s^2+1}}\right). \tag{6.25}$$

Now, assuming the initial estimate of $Z^0(x,y) = 0$ for all pixels, the depth map can be iteratively refined using Equation 6.24. In most cases, two or three iterations are enough. The results for this algorithm are shown in Figure 6.12.

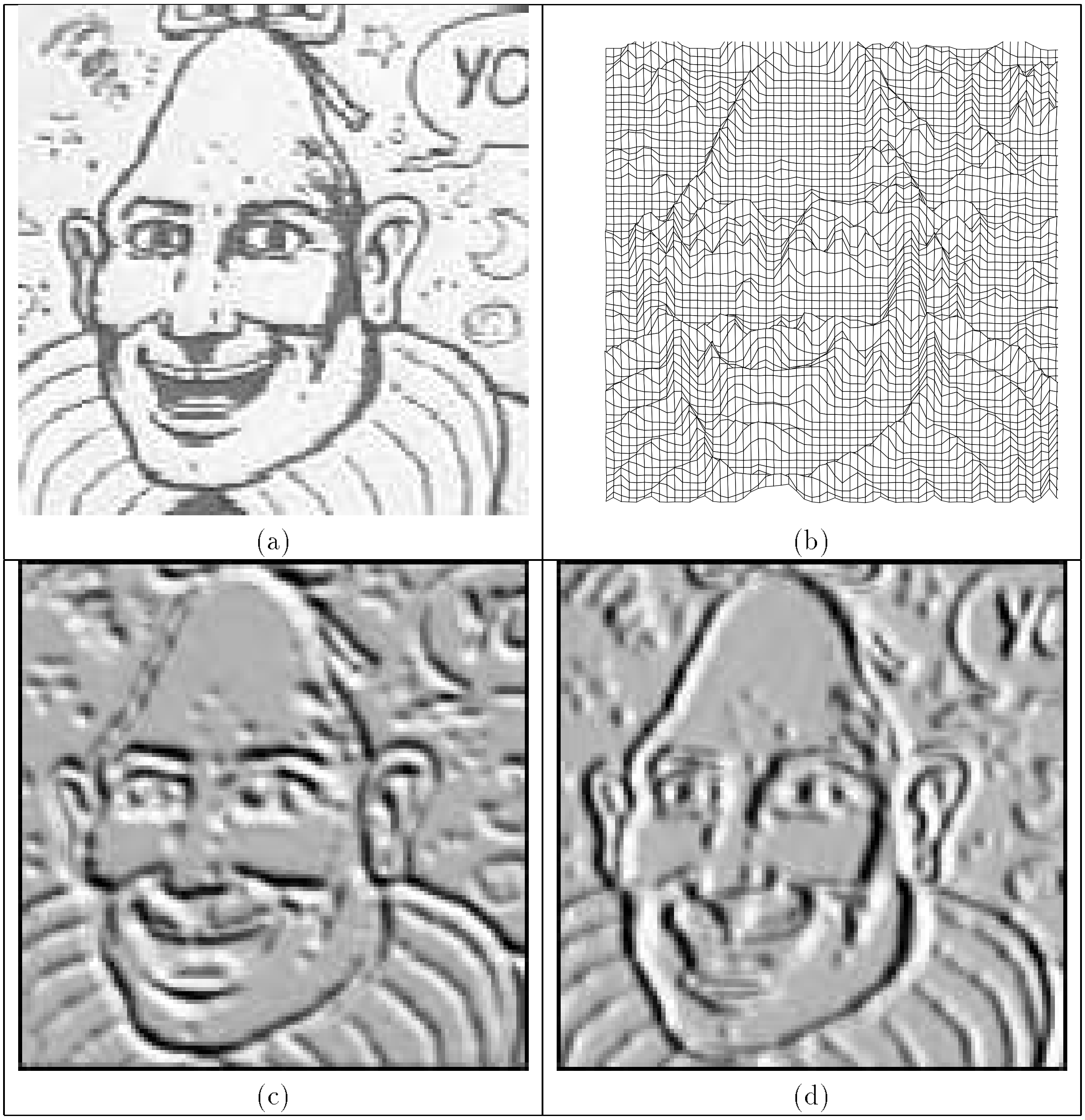


Figure 6.12: The results for Yowman Image. (a) The input image. The light source parameters estimated by Lee & Rosenfeld's method are : $slant = -45.75^o$, $tilt = 62.14^o$. (b) A 3-D plot of the depth map computed by Tsai-Shah algorithm. (c) A reconstructed gray level image using depth map in (b) and constant $albedo = 255$ with the estimated light source direction ($slant = -45.75^o$, $tilt = 62.14^o$). (d) A reconstructed gray level image using depth map in (b) and constant $albedo = 255$ with the light source direction ($slant = 45^o$, $tilt = 0^o$).

6.4 Photometric Stereo

In photometric stereo multiple light sources are used to recover shape. Consider the reflectance equation again, the intensity is given by

$$I(x, y) = B\vec{N} \cdot \vec{S}, \tag{6.26}$$

where B is the albedo, $\vec{S}$ is the light source vector, and $\vec{N}$ is the surface normal. Assume we take three images I_1, I_2 and I_3 with three light sources S^1, S^2 and S^3, then:

$$\begin{aligned} I_1(x, y) &= B\vec{S^1}.\vec{N}, &(6.27)\\ I_2(x, y) &= B\vec{S^2}.\vec{N}, &(6.28)\\ I_3(x, y) &= B\vec{S^3}.\vec{N}. &(6.29) \end{aligned}$$

These equations can be written in the matrix form as:

$$B \begin{bmatrix} s_x^1 & s_y^1 & s_z^1 \\ s_x^2 & s_y^2 & s_z^2 \\ s_x^3 & s_y^3 & s_z^3 \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}. \tag{6.30}$$

Or,

$$\begin{aligned} BA\vec{N} &= I, &(6.31)\\ B\vec{N} &= A^{-1}I, &(6.32)\\ B||\vec{N}|| &= ||A^{-1}I||. &(6.33) \end{aligned}$$

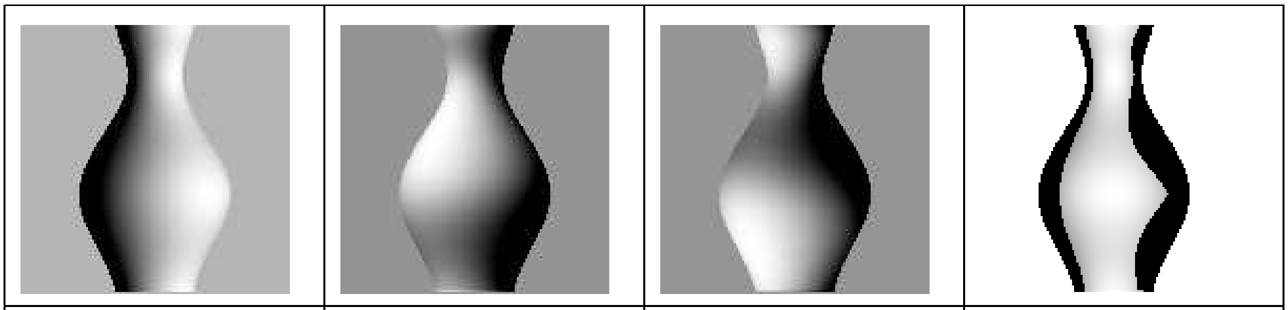
Since surface normal is unit vector, $||n|| = 1$. Now, from the above equation we get:

$$B = ||A^{-1}I||. \tag{6.34}$$

Once we know the albedo, B, we can compute $\vec{N}$ as follows:

$$\vec{N} = \frac{A^{-1}I}{B}. \tag{6.35}$$

The results obtained by photometric stereo are shown in Figure 6.13.



(a)	(b)	(c)	(d)

Figure 6.13: (a)-(c) Input images with light sources $(1, 0, 1)$, $(-1, 1, 1)$, $(-1, -1, 1)$ (d) Intensity image reconstructted using the surface normals computed by photometric stereo.

6.5 Exercises

1. Verify equation 6.17.

2. Verify equation 6.25.

Chapter 7

Range Images

7.1 Introduction

Range images represent the distance of the object from the sensor. These images provide the direct 3-D information (shape) in contrast to the intensity images which record the amount of light reflected from the objects. In intensity images the intensity value at a pixel depends on the light source location, surface reflectance, and surface shape. In order to derive 3-D shape from intensity images we need to apply shape from X methods as discussed in the previous chapter. Range image provide depth information directly. However, range sensors are slow and expensive. They provide arrays of numbers which still needs to be processed and analyzed.

7.2 Range Image Formation

A block-diagram of a typical range sensor is shown in Figure 7.1.

The light beam is transmitted from the transmitter which strikes the object as shown, and reflected back to the receiver through scanning mirror. A second light beam is also sent to the receiver directly. The receiver determines the phase difference between the reflected and the reference beams to compute the distance of object from the sensor. Next, the scanning mirror is appropriately moved in the horizontal and vertical directions, and the process is repeated for the other location on the object.

As we know from Physics the speed, c, frequency, f, and the wave length, λ, are related as follows: $c = \lambda f$. Therefore, with speed of light $c = 3 \times 10^6$, for a maximum range of 30 meters, we need to have light beam with a frequence, f, of $f = \frac{3\times 10^8}{30} = 10 \times 10^6$. Then the distance, D, of the object point from the sensor can be determined by the phase difference, θ, between the reference and reflected beams by the following formula: $2D = \frac{\theta}{360}\lambda$, or $D = \frac{\theta}{360} 2\lambda$.

One commercially available range sensor, called ODETICS 3D Mapper, has the following specifications. The sensor uses a solid state 820 nm (nano meters) GaAlAs (Gallium Alluminum Arsenide) laser diode. The scan system uses a rotatory polygon mirror for the horizontal scan and a planar nodding mirror for the vertical scan. It provides 128×128 pixels image with a frame rate of 835 msec/frame. The range resolution is 1.44 inches, and minimum range of 1.5 feet. As it is obvious that 820 nm wave length sensor will have very

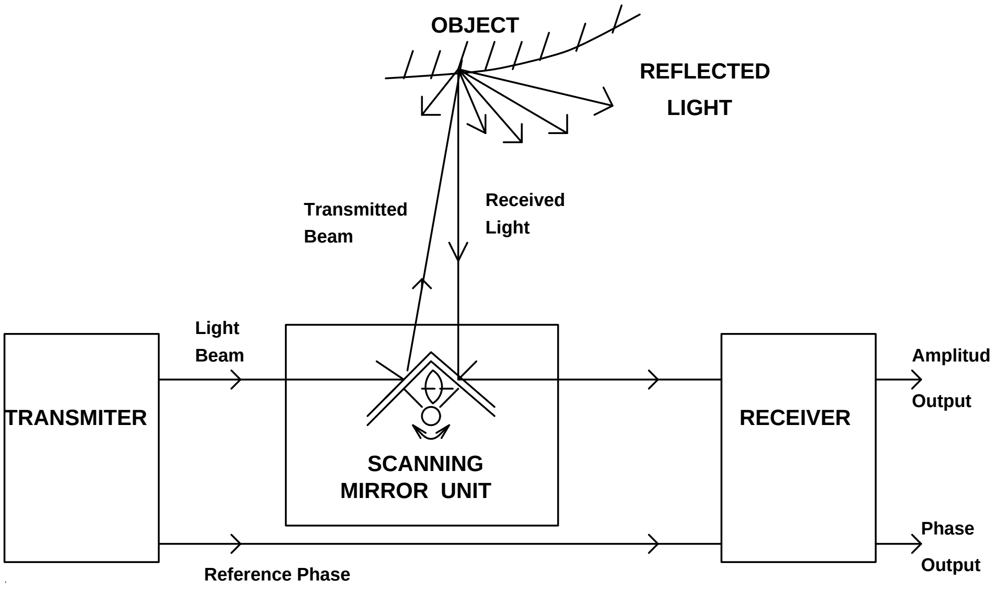


Figure 7.1: A Block-diagram of a range sensor.

limited range, therefore, this laser beam is normally modulated with some high frequency signal (e.g. 10 MHz) to achieve higher range, e.g. 30 m range. Since the sensor computes the range based on the phase difference which can vary between 0 to 360 degrees, the phase difference of more than 360 degrees is not possible to determine accurately due to the nature of the continuous waveforms. For example, the phase difference of 365 degrees will essentially be treated as a phase difference of 5 degrees. Therefore, there is an ambiguity interval, beyond which we can not determine the range accurately. For the ODETICS sensor the ambiguity interval is 30.74 feet.

7.3 Surface Characteristics

Range image is an array of numbers which need to processed and analyzed. Since range represents the distance of object from the object, the object surface characteristics can directly be derived from the range image.

Important surface characteristics can be derived by computing the surface curvature. The Gaussian curvature, k, and mean curvature, H are two very popular surface properties which have been used in Computer Vision. These curvatures are given by:

$$\begin{aligned} k &= \frac{f_{uu}f_{vv} - f_{uu}^2}{(1+f_u^2+f_v^2)^2}, \\ H &= \frac{1}{2}\frac{(1+f_v^2)f_{uu} + (1+f_u^2)f_{vv} - 2f_uf_v2f_uf_vf_{vv}}{(1+f_u^2+f_v^2)^{\frac{3}{2}}}, \end{aligned}$$

where $f(u,v)$ is the range image, f_x represents the derivative of f with respect to x.

The surface types using the signs of mean and Gaussian curvature can be determined in the the following way:

- $K < 0$ *and* $H < 0$: peak

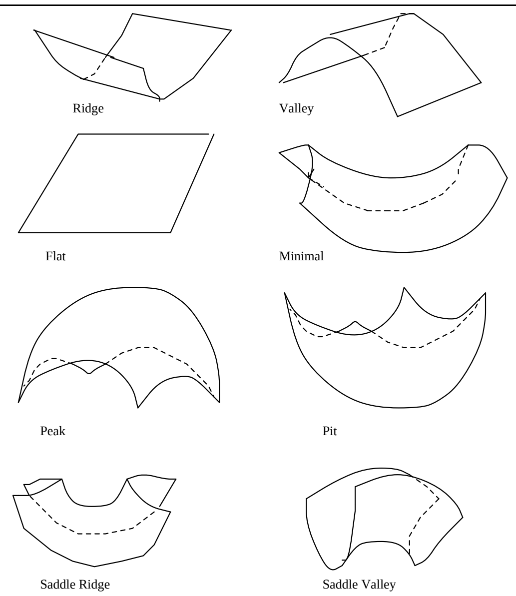


Figure 7.2: Surface types.

- $K < 0$ *and* $H > 0$: pit
- $K = 0$ *and* $H < 0$: ridge
- $K = 0$ *and* $H = 0$: flat
- $K = 0$ *and* $H > 0$: valley
- $K > 0$ *and* $H < 0$: saddle ridge
- $K > 0$ *and* $H = 0$: minimal
- $K > 0$ *and* $H > 0$: saddle valley

Various types are shown in Figure 7.2.

7.4 Edges in Range Images

There are several kinds of edges in the range image.

- **Jump Edges** represent discontinuity in depth. If the depth difference between a pixel and any immediate neighboring pixel is above some threshold then it is a jump edge.
- **Roof Edges or Creases** represent discontinuity in orientation. If the angle between surface normal at a pixel and the surface normal at the immediate neighboring pixel is above some threshold then it is a roof edge. There are two kinds of roof edges: **Convex ("+")**, and **Concave ("-")**. The convex edges are formed when the angle between the surface normals is above 180 degrees, and the concave edges are formed when the angel between surface normals is below 180 degrees.

- **Occluding Edges** ($\longrightarrow$). These are convex edges for which there is only one visible surface (depth discontinuity).
- **Limb Edges** ($\longrightarrow$). When the surface normal of edge is perpendicular to the viewing direction, that edge is called a limb edge.

Bibliography

[1] Barnard, S.T, and Thompson, W.B. Disparity analysis of images. Technical Report 79-1, Computer Science Department, University of Minnesota at Mineapolis, January, 1979.

[2] J. F. Canny. A variational approach to edge detection. In *Proceeding of The National Conference on Artificial Intelligence*, pages 54–58, August 22-26, 1983.

[3] Canny, J. F. Finding edges and lines in images. Master's thesis, MIT, 1983.

[4] R.M. Haralick. Digital step edges from zero-crossings of second directional derivative. *PAMI-6*, 6(1):58–68, January 1984.

[5] R. Hartley. A gaussian-weighted multiresolution edge detector. *Computer Vision, Graphics and Image Processing*, 30:70–83, 1985.

[6] G. Healey and T.O. Binford. Local shape from specularity. *CVGIP*, 42:62–86, 1988.

[7] D. Heeger and A. Jepson. Subspace methods for recovering rigid motion i: Algorithm and implementation. *International Journal of Computer Vision*, 7:95–117, 1992.

[8] Hildreth, E.C. Detection of intensity changes by computer and biological vision systems. *Computer vision, Graphics and Image Processing*, pages 1–27, 1983.

[9] Horn, B. and Schunk, B. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[10] M.H. Hueckel. An operator which locates edges in digitized pictures. *Journal of ACM*, 18:113–15, 1971.

[11] Ikeuchi, K. and Horn, B. Numerical shape from shading and occluding boundaries. *Artificial Intelligence*, 1981.

[12] Lowe, D.G. *Perceptual Organization and Visual Recognition.* Boston: Kluwer Academic Publishers, 1985.

[13] D. Marr and E. Hildreth. Theory of edge detection. In *Proc. R. Soc. Lond.*, volume B-207, pages 187–217, 1980.

[14] Marr, D. *Vision.* Freeman, San Francisco, CA, 1982.

[15] Marr, David and Poggio, T. A computational theory of human vision. In *Royal Soc. London*, volume B, 1979.

[16] A. Pentland. Shape information from shading: a theory about human perception. In *Second International Conference on Computer Vision*, pages 404–413, 1988.

[17] A. P. Pentland. Finding the illuminant direction. *J. Optical Society of America*, pages 448–455, 1982.

[18] A. P. Pentland. Local shading analysis. *IEEE Transactions on PAMI*, 6:170–187, 1984.

[19] Prewitt, J.M. Object enhancement and extraction. In *Picture Processing and Psychopictorics*. Academic press Inc, 1970.

[20] Ranganathan, N. and Shah, M. A vlsi architecture for computing scale space. *Computer Vision, Graphics, and Image Processing*, 43:178–204, 1988.

[21] A. Rosenfeld and M. Thurston. Edge and curve detection for visual scene. *IEEE Trans. on Computers*, C-20(5):562–569, 1971.

[22] Schunck, B. Image flow segmentation and estimation by constraint line clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(10):1010–1027, 1989.

[23] T. Strat. Recovering the camera parameters from a transformation matrix. In *Readings in Computer Vision*. Morgan Kaufmann, 1987.

[24] Tsai, P-S, and Shah, M. . A fast linear shape form shading. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 734–736, June, 1992.

[25] Witkin, A. Scale-space filtering. In *Proceedings Eighth International Joint Conference on Artificial Intelligence*, pages 1019–1021, Karlsruhe, W.Germany: IEEE Computer Society, 1983.

[26] R. J. Woodham. Multiple light source optical flow. *IEEE ICCV*, pages 42–46, 1990.