Digital Image Operations

Matthew C. Forman

Contents

Abstract

In applications that deal with digitally represented visual images, various forms of processing are generally required before the results are ready to be displayed. Although many of the methods used are complex, all have their roots in a small number of core concepts and techniques. This chapter looks at these common core spatial domain operations, firstly reviewing those that rely on applying transformations of brightness and color in place within digital images. It then moves on to consider geometric manipulation of image data and resampling issues.

M.C. Forman (✉)
Create-3D, Sheffield, UK
e-mail: matt@create-3d.co.uk

J. Chen et al. (eds.), *Handbook of Visual Display Technology*,
DOI 10.1007/978-3-319-14346-0_26

List of Abbreviations

API	Application programming interface
CPU	Central processing unit
GPU	Graphics processing unit
HSV	Hue, saturation, value (color space)
RGB	Red, green, blue (color space/color storage method)
$Y'C_bC_r$	Luma, blue-difference chroma, red-difference chroma (color space/ color storage method)

Introduction

The rapid growth of digital storage of visual images has been driven by several factors. Digital representations inherently have far better robustness and noise immunity than direct analog recordings. This is extremely advantageous where both long-term storage and communication are concerned. One of the most significant advantages of digital representation, however, is the ease with which useful and complex processing operations can be implemented.

The precise details of a particular implementation of a digital image storage scheme depend on the application: the manner of source content creation, storage or transmission system requirements, and the final destination of the image. At a low level, however, an image is stored in either a *raster* (also commonly known as *bitmap* or *pixmap*) or *vector* representation. A vector representation consists of instructions and parameters for drawing the final image, element by element, from geometric primitives such as lines, curves, polygons, and text. A raster format represents a lower level of abstraction of image data. It contains a sampled representation of any captured or synthesized image and thus offers a more general means of storage. Since display systems themselves are addressed in this manner, the final destination for all image representations is effectively raster; an image in a vector format is *rasterized* for display by executing the appropriate drawing instructions and sampling the result. This article therefore concentrates on processing that can be achieved when an image is stored in a raster format.

Raster Image Processing Format

In the most general sense, a raster image is comprised of a rectangular array of *pixels* ("picture elements") (Watt and Policarpo 1998; Gonzalez and Woods 2008). Each pixel is a sample of the information in a finite area of a spatially continuous image source, centered on a particular geometric location in the plane. The sample value may simply be the scalar irradiance arriving at an image sensor pixel, or equivalently the emittance of a display pixel; an array of these represents a grayscale image. Alternatively a pixel may carry color information, typically by encoding irradiance/ emittance proportions of red, green, and blue light; the array of such pixels can

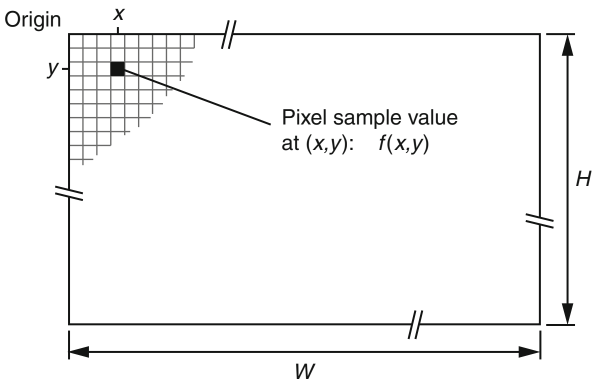

Fig. 1 General raster (bitmap) image layout

represent a full-color image. Figure 1 illustrates the layout of a general raster image (note that the origin is also commonly at bottom left).

Color Image Processing

Although color images are usually *RGB* encoded in capture and display devices, such a color representation is not necessarily best suited for supplying full-color image data to image processing operations. Hence, for processing, images are often transformed into alternative color spaces that are more compatible with the operation (s) to be carried out, or simply for ease of implementation. For example, it is often desired to separate the luma (brightness) from the chroma (color) information and process them separately – the $Y^{\text{â€²}}C_bC_r$ (luma, chroma blue, and chroma red) space is commonly used in these cases. In other operations the HSV space may be appropriate.

Many image processing operations may be carried out directly in the pixel spatial domain – some of which may require resampling – though others are more easily applied in a spatial frequency domain such as Fourier space.

This chapter continues by looking at spatial domain pixel operations, a group of common global processing operations that rely on applying transformations of brightness and color within digital images.

Global Pixel Operations

A fundamental class of image processing techniques, *global pixel operations*, apply a single operation identically to each pixel. This section introduces a number of intensity-only and color-specific pixel operations. A second class of local pixel operations, where a number of values in a local neighborhood of the operation pixel are considered, is often used in filtering and enhancement applications.

Intensity Transformations

An intensity transformation uses a linear transfer function to remap input pixel intensity values (Watt and Policarpo 1998). If *f*(*x*, *y*) represents an intensity raster image and *Ti*) is an intensity transfer function, then the processed image is:

$$g(x,y) = T(f(x,y)).$$

A general remapping facility such as this allows a number of practical enhancement operations to be carried out. It is convenient to visualize the transfer function as a line plot relating output to corresponding input values, and software with intensity transformation features often allows transformations to be defined graphically in this way, generally with reference to the intensity histogram of the image. Some common intensity transformations are illustrated by Fig. 2, which also shows their effects on sample images.

Contrast stretching (Fig. 2a) expands the intensity range of an image in parts of intensity space (typically the center) while compressing or clamping the intensity dynamic range in other parts. The goal is to improve the utilization of dynamic range for the most important parts of the image. The results can be seen in the example shown as improved contrast. Here, limiting low- and high-intensity ranges have been clamped to black and white.

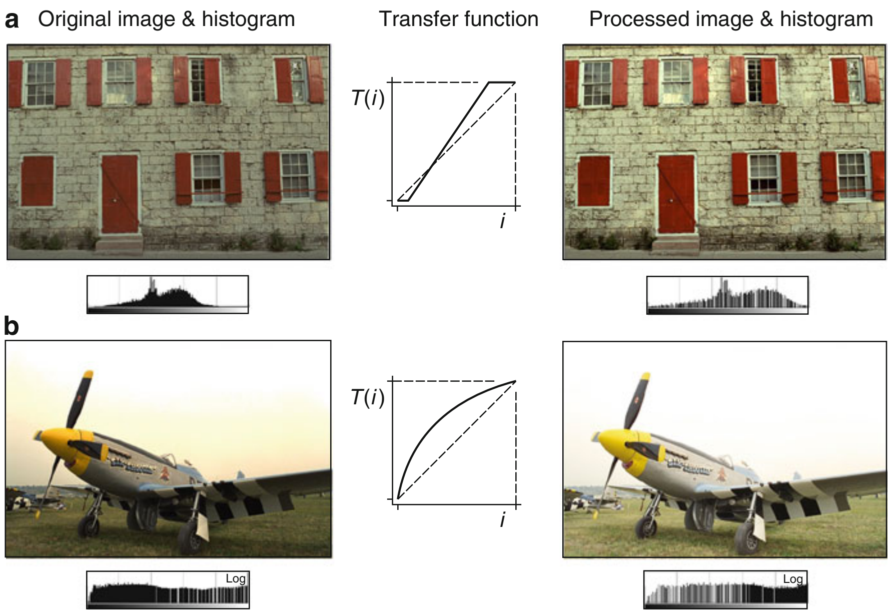

Fig. 2 Intensity transformation operations on sample images. (**a**) Contrast stretching. (**b**) Nonlinear brightness adjustment

Normalization is a related process that determines stretch limits automatically from the image brightness histogram, so that the image's existing intensity range is mapped exactly on to the maximum range available. This is particularly useful to compensate for photographic underexposure.

An offset can be added to the transfer function to increase or decrease overall image brightness; however, this generally results in saturation at the white or black level. As an alternative, *nonlinear brightness adjustment* (Fig. 2b) applies a smooth curve to increase or decrease overall image brightness in such a way that saturation at maximum or minimum intensity cannot occur. A power function, such as that used for display gamma correction (see chapter "▶ Luminance, Contrast Ratio, and Gray Scale"), is often used.

As in the examples shown, any intensity transformation can be applied to a color image by first transforming the image data into a color space which represents intensity information separately from color information, applying the transformation to the intensity component and then transforming back to the original color space. Using the $Y^{\text{â€²}}C_bC_r$ space, for example, the luma (YÊ^1) component would be subject to transformation, while the chroma components (C_b and C_r) would be passed unchanged.

Color Saturation Adjustment and Matrix Methods

It is often desired to make adjustments to color saturation in an image in RGB space. One way to accomplish this is to convert the image data into HSV space and make the appropriate modification to the S (saturation) component before transforming the data back to RGB for display. However, this requires several separate operations and hence may not result in a particularly efficient implementation. It also carries the risk of introducing precision, rounding, and overflow issues.

A convenient way to implement global pixel processing operations directly in RGB space uses a general matrix-based framework (Haeberli 1993). The matrix multiplication of the input color pixel value (in the form of a column vector) with an operation matrix yields the output pixel value. We define the operation as a 4 × 4 matrix to result in a general linear transformation, and several operations can then be concatenated into one just by multiplying operation matrices. The input pixel vector,

$$\mathbf{F}(x, y) = [f_R f_G f_B \, 1]^T$$

with f_R, f_G, and f_B being the source red, green, and blue component values, and the output pixel vector,

$$\mathbf{G}(x, y) = [g_R \, g_G \, g_B \, g_w]^T$$

with g_R, g_G, and g_B being the destination red, green, and blue component values, and g_w is not generally computed. If T is a 4 × 4 matrix defining the desired pixel operation, then the overall operation is represented as

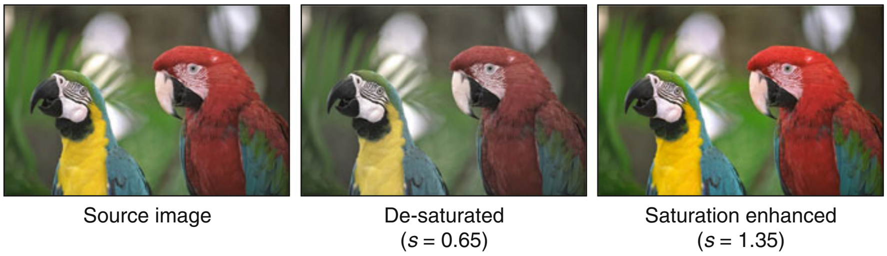

Fig. 3 Image color saturation modifications using matrix methods in RGB space

$$\mathbf{G}(x,y) = \mathbf{T}.\mathbf{F}(x,y).$$

The operation matrix for saturation adjustment is

$$\mathbf{T}_{\text{sat}}(s) = \begin{bmatrix} *20c\alpha + s & \beta & \gamma & 0 \\ \alpha & \beta + s & \gamma & 0 \\ \alpha & \beta & \gamma + s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{with } \alpha = 0.3086\,(1-s)$$

$$\beta = 0.6094\,(1-s)$$

$$\gamma = 0.0820\,(1-s)$$

Here, Î±, Î^2, and Î^3 are factors derived according to the contributions of red, green, and blue components, and s is the saturation adjustment value. If $s = 0$, all color is removed leaving only brightness information. If $s = 1$, there is no change, and values $0 < s < 1$ result in various levels of desaturation. For values $s > 1$, saturation is enhanced.

Figure 3 demonstrates the effects of applying saturation enhancement (example value $s = 1.35$) and desaturation (example value $s = 0.65$) to a sample image, using this process.

Intensity transformations for contrast and brightness changes, as well as many color-specific transformations – for example, hue rotation – can be specified concisely and applied using the matrix framework. Combined operations can also be computed efficiently, concatenating individual operations by multiplying together the appropriate matrices before applying the result.

Matrix pixel transformations can be implemented efficiently using precomputed lookup tables. The fast matrix arithmetic facilities of GPUs and some CPUs are also useful for this.

Application Example: White Point Correction

When source material is shot, the color and luminance responses of the scanner or camera used are ideally calibrated and known. Color management techniques (see chapter "▶ Fundamentals of Image Color Management") can then be used to ensure that the image that is ultimately displayed is perceived to be as close as possible to the original scene, with neutral shades being reproduced as accurately as possible at the display. The response of a camera is not always known a priori, however, or material may have been shot with an incorrect white point setting in force. Correction can be achieved by remapping the white point using a simple global color pixel operation, with the capture device having been used to record a physical reference white point in the scene. This operation is also useful for deliberate manipulation of the white point of an image as for special effect purposes. There are a number of options when defining this operation, particularly with regard to color space (Viggiano 2004). Here we assume operation in the RGB space of the camera itself.

If the measured color pixel value of the white point reference,

$$\mathbf{W} = [w_R\, w_G\, w_B],$$

then assuming the full-scale color component value is 255 (i.e., 8 bits per component color), the white point correction operation can be defined in the matrix framework above as

$$\mathbf{T}_{\mathrm{wpt}} = \begin{bmatrix} *20c255/w_R & 0 & 0 & 0 \\ 0 & 255/w_G & 0 & 0 \\ 0 & 0 & 255/w_B & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that if any color component is zero, the result of the corresponding computation above would be undefined. Although this is very unlikely in practice, it must be considered in an implementation. Also, because of the likelihood of saturation of at least one color component of a white reference to the full-scale representable value, it is often more reliable to measure at least one known *gray* physical reference instead. The white point reference, *W*, can then be computed from these values.

Geometric Image Operations and Resampling

A further important set of processing techniques commonly applied to image data is the class of *geometric image operations*. Image resizing (scaling), rotation, and morphing are some very common practical applications. In the case of vector images, any required geometric transformations are generally applied directly in the vector representation, before rasterization for display takes place. Indeed this point encapsulates the chief advantage of using a vector rather than a raster representation whenever possible: although images destined for digital display inevitably must be sampled into raster form eventually, transformations applied before sampling effectively take place in a continuous domain and are therefore *scale independent*.

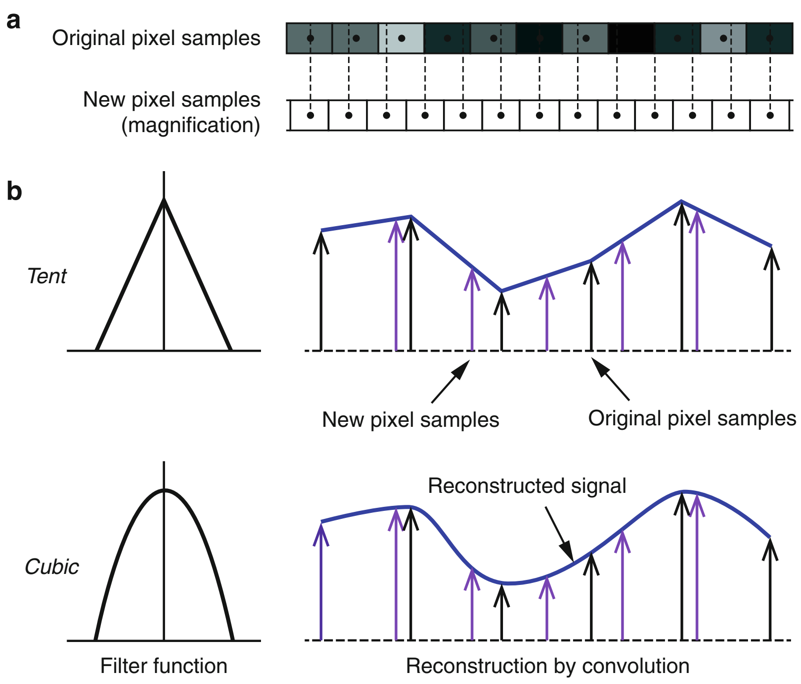

Fig. 4 Image scaling and resampling. (**a**) New pixel centers are not aligned with original ones. (**b**) Filtered resampling using tent and cubic filter functions

Geometric transformations on images in raster form must take account of the fact that these images have already been sampled and discretized. While the pixel processing methods outlined in the previous section deal with modifications only to color or intensity sample *values* in raster digital images, geometric operations involve changes to the *positions* of image samples.

Raster Image Resampling and Scaling

Scaling is a very common requirement in image processing, for example, when zooming at the display to inspect detail closely, or preparing an image optimally for a display device with a certain resolution. Consider for example that an image must be scaled up so that every six pixel rows and columns are instead represented by seven in the destination image. Figure 4a illustrates a section of a single row of pixels of the source image and also the corresponding section of pixels in the destination, scaled image. The problem now is of determining appropriate values for the new pixels. Many of the destination pixel centers are not aligned with source pixel centers, so the source pixel values must be mapped onto the destination pixel grid and resampled. A naïve approach simply selects the closest source pixel value to the center of each destination pixel; however, this results in significant image distortion with portions of the image information being deleted completely or replicated. An approach to scaling without introducing such distortion involves approximating the original continuous intensity/color surface and sampling it at locations corresponding to the new pixel centers (Schumacher 1995).

This approximation is made by convolving the source image pixels with a filter impulse response function and setting destination pixel values at locations in the convolved signal that correspond to their centers. The theoretically ideal reconstruction filter is defined by the sinc function; it corresponds to an ideal low-pass cutoff characteristic in the spatial frequency domain. However, the sinc function contains negative values and has infinite support, and thus is not practical to use directly.

Increasing the scale of an image as above (*magnification*) requires resampling to a higher pixel rate. This corresponds to interpolation in signal processing terms. Reducing image scale, sometimes known as *minification*, requires resampling to a lower pixel rate – decimation in signal processing terms. Note that in this case, care must be taken that the new, lower sampling rate is still adequate for the spatial frequency content of the image so as not to introduce aliasing distortion. This is generally achieved by scaling the filter function before convolution.

The reconstruction process is illustrated in Fig. 4b using two typical filter functions: the "*tent*" (corresponding to linear interpolation between samples) and *cubic* filters (describing the shapes of their impulse responses). The cubic function approximates the ideal sinc function with better precision resulting in higher fidelity results, but is slower to compute than the "tent." Others such as the *Lanczos* filter, offer still better reconstruction accuracy (Turkowski 1995). In implementation, the filter is generally applied one dimensionally through all rows and all columns separately.

Image Rotation

A second extremely useful geometric operation is the rotation of an image. For rotation through multiples of 90°, resampling is not required unless pixels are nonsquare; a simple transfer of pixel values directly from one location to another is sufficient. However, rotation of an image through an arbitrary angle is often needed, and this does require resampling (see Fig. 5). A general, direct implementation would use filter functions for reconstructing the continuous intensity surface, and then resample this according to new, rotated pixel centers. Such an approach is, however, computationally inefficient and cumbersome to implement.

A more practical algorithm is due to Paeth (1995). Here, the rotation operation to be applied anticlockwise through angle $\hat{I}$ is decomposed into three simple *shear* operations. If source and destination pixel coordinates are represented as column vectors $\mathbf{S} = \left[s_x\, s_y\right]^T$ and $\mathbf{D} = \left[d_x\, d_y\right]^T$, respectively, a general transformation from source to destination coordinates according to matrix $\mathbf{M}$ is

$$\mathbf{D} = \mathbf{M}.\mathbf{S}.$$

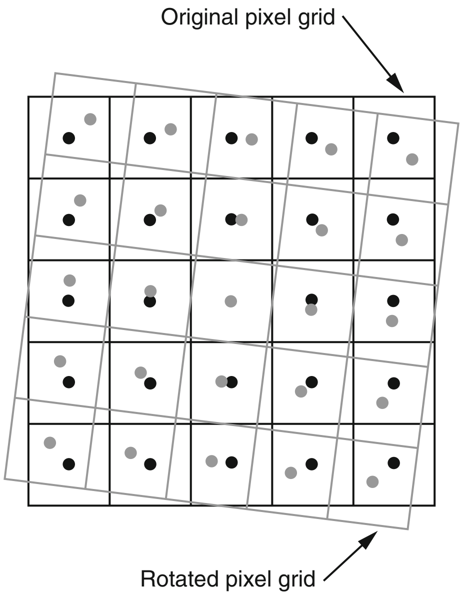

Fig. 5 Arbitrary rotation of a raster image requires resampling due to the complex overlay of source and destination pixels

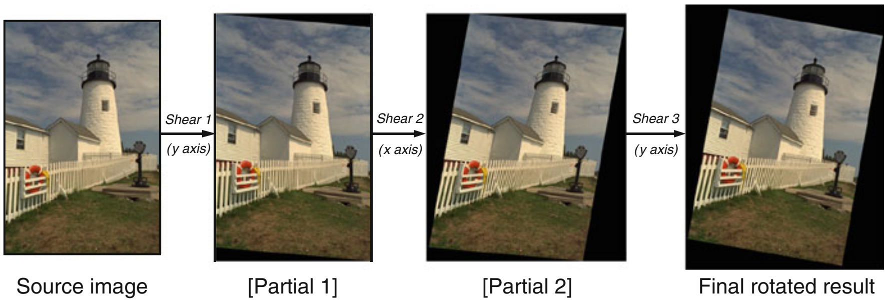

Fig. 6 Rotation of a raster image using the three-shear method

The rotation matrix can be considered as the product of three shear matrices:

$$\mathbf{M_{rot}} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{bmatrix}.$$

Implementation of shear operations is straightforward, requiring only a shift of pixel data along one axis (effectively, resampling of translated pixels using a simple tent function) proportional to the distance along the second axis. Figure 6 illustrates the process for a clockwise rotation through 10°. Note that as a consequence of rotation, the rectangular pixel array area required to hold the image is increased, though cropping is often used to retain a rectangular subregion that does not include the rotated image boundary. For improved accuracy, rotations by angles of 90° or more are implemented by transfer of pixels for right angle portions followed by the three-shear algorithm for the remaining portion.

Other Geometric Image Operations

In addition to scaling and rotation, a number of other geometric operations requiring resampling are possible on raster image data. Considering straightforward affine transformations, the simple *shear* has already been outlined in its application to rotation using the Paeth method. *Translation* is also sometimes useful – this is effectively a phase shift of pixel data by an amount which is not necessarily an integer number of pixels.

More general remapping and warping techniques are often required in certain higher level applications (Watt and Policarpo 1998). An image may be mapped to a regular or irregular mesh, and mesh nodes manipulated to apply modifications to the image structure in a local sense. A common application of such a technique is perspective transformation, often used (with knowledge of camera parameters) to correct for perspective distortion in an image captured from a camera. Warping methods are also used in *morphing*: creating a smooth transition from one image to another, driven by relationships between the nodes of the mesh in both images, defined by the user. A closely related application to morphing is the synthesis of new viewpoints of a scene, given at least two known viewpoint images and a set of correspondences between mesh nodes in the source images. This is useful in three-dimensional imaging and modeling.

Implementation of Fast Geometric Transformations

Modern commodity GPUs have evolved chiefly to accelerate three-dimensional object transformations and rendering for computer entertainment and visualization applications. The parallel processing facilities that make this possible, however, also greatly simplify implementation of very fast geometric operations on raster images – both affine transformations and more complex warps (Qureshi 2001). This can be achieved through common APIs such as *OpenGL* (Silicon Graphics 1992) and *Microsoft DirectX* (Akenine-Möller et al. 2008).

A general technique for implementing 2D affine transformations of raster images is as follows (see Fig. 7):

1. Define a virtual camera, usually with an orthogonal projection and a viewport mapping to a destination pixel buffer.
2. Create a geometric entity in object space. For affine transformations, a simple quadrilateral surface is suitable.
3. Using the texture handling facilities of the API, map the source image to the geometric entity just created.

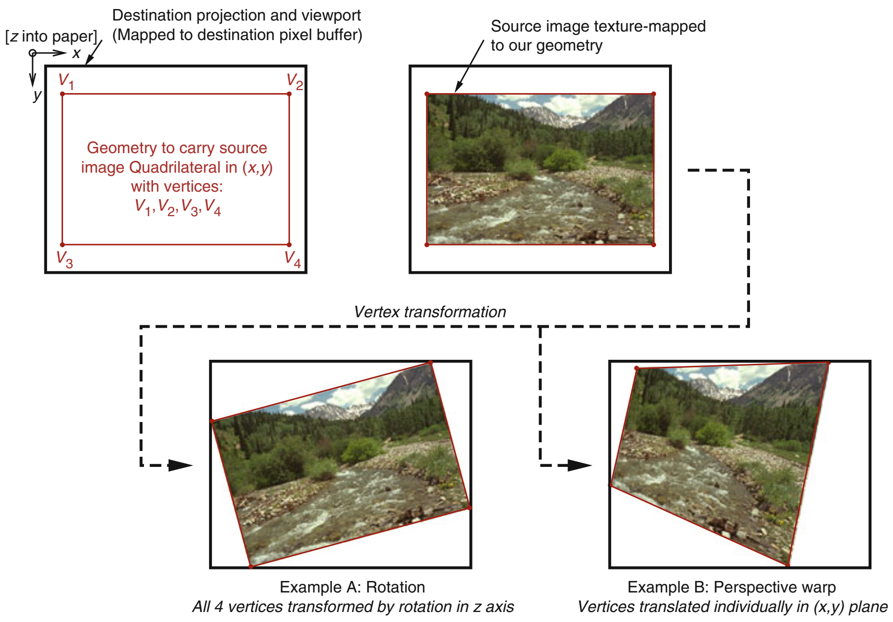
Destination projection and viewport
(Mapped to destination pixel buffer)
[z into paper]
x
y
V_1
V_2
V_3
V_4
Geometry to carry source image Quadrilateral in (x,y) with vertices: V_1, V_2, V_3, V_4
Source image texture-mapped to our geometry
Vertex transformation
Example A: Rotation
All 4 vertices transformed by rotation in z axis
Example B: Perspective warp
Vertices translated individually in (x,y) plane

Fig. 7 Fast geometric transforms

4. Transform the vertices of the geometric surface, either directly or using the vertex affine transformation facilities of the API. Since the geometric surface is “carrying” the source image, the final rendered result will be a destination image transformed accordingly. The GPU’s texture lookup filtering functionality ensures that appropriate resampling takes place automatically.

General mesh warps can also be achieved directly, simply by using more complex geometry to define a suitable mesh containing internal vertices and then transforming those vertices as necessary to apply the desired warp.

Summary

All practical image and video processing applications are built on a core set of low-level operations on digital representations of images. Some of these are applied in place at the pixel level, but others involving geometric transformations result in changes to the inherent structure of the image representation, and therefore must take into account sampling issues. It is relatively straightforward to use modern graphics hardware and APIs to implement extremely fast fundamental image processing operations.

Further Reading

Akenine-Möller T, Haines E, Hoffman N (2008) Real-time rendering, 3rd edn. A. K Peters, Natick

Gonzalez RC, Woods RE (2008) Digital image processing, 3rd edn. Pearson Prentice Hall, New York

Haeberli P (1993) Matrix operations for image processing. http://www.graficaobscura.com/matrix/index.html. Accessed Nov 1993

Paeth AW (1995) A fast algorithm for general raster rotation. In: Kirk D (ed) Graphics gems. Academic, Boston

Qureshi S (2001) Image rotation using OpenGL texture maps. C/C++ User J 19:10–17

Schumacher D (1995) General filtered image rescaling. In: Kirk D (ed) Graphics gems III. Academic, Boston

Silicon Graphics, Inc (1992) The OpenGL graphics system: a specification. Version 1.1

Turkowski K (1995) Filters for common resampling tasks. In: Glassner AS (ed) Graphics gems. Academic, Boston

Viggiano JAS (2004) Comparison of the accuracy of different white balancing options as quantified by their color constancy. In: Proceedings of the SPIE, vol 5301. Bellingham

Watt A, Policarpo F (1998) The computer image. Addison-Wesley, Reading