æ

Effective Homogeneous Transformation of Large Raster Images

Vlastimil Havran, Jiří Žára
CTU Prague, Dept. of Computer Science

1 Introduction

Homogeneous transformation is one of the most general linear transformations. For coordinates x, y in input image, the calculation of coordinates x', y' in output image can be expressed by the following formulae:

$$\begin{aligned} x' &= (x * a_{11} + y * a_{12} + a_{13})/(x * a_{31} + y * a_{32} + a_{33}), \\ y' &= (x * a_{21} + y * a_{22} + a_{23})/(x * a_{31} + y * a_{32} + a_{33}), \end{aligned} \qquad H = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

is the homogeneous transformation matrix. All common types of linear transformations like rotation, scaling, shear, etc. can be described by this matrix ([1]).

Sometimes we need to perform transformation of very large raster images obtained by scanner, where the size 10.000×10.000 pixels in 8-bit grayscale is not exceptional. Storing of such images for direct data manipulation in main memory requires 100 Mbytes RAM or more, which causes the problem for IBM PC based machines. The standard page fault technique is completely inconvenient. Although output pixels are processed one by one in a row, input pixels are taken from positions, which are placed in different memory locations. We can find complicated cases (large images, 45 degrees rotation), where for each pair of neighbouring output pixels, two different pages are needed. It causes very low computational performance.

2 Implementation

We have designed virtualization technique specialized for transformations of raster images. The system with two caches implemented in software is used. The **free cache** is the standard cache managed by LRU technique. The second one called **locked cache** stores the part of input image, which has not to be swapped to a disk. The granularity of both caches corresponds with an image row or with a part of a row respectively.

The row can be registered either in the **locked cache** or in the **free cache**. Both caches cooperate and exchange their data without disk using the **cache management subsystem**. The restriction for used memory capacity is defined by the following expression:

$$management\_system\_RAM + free\_cache\_RAM + locked\_cache\_RAM < CONST$$

The image transformation is performed in the following steps. First, we prepare the inversion matrix for the direct calculation from output to input coordinates. Second, boundaries of input image are transformed and clipped by output image rectangle. We obtain a polygonal output area with up to 8 vertices, which has to be filled by transformed input pixels (see fig. 1).

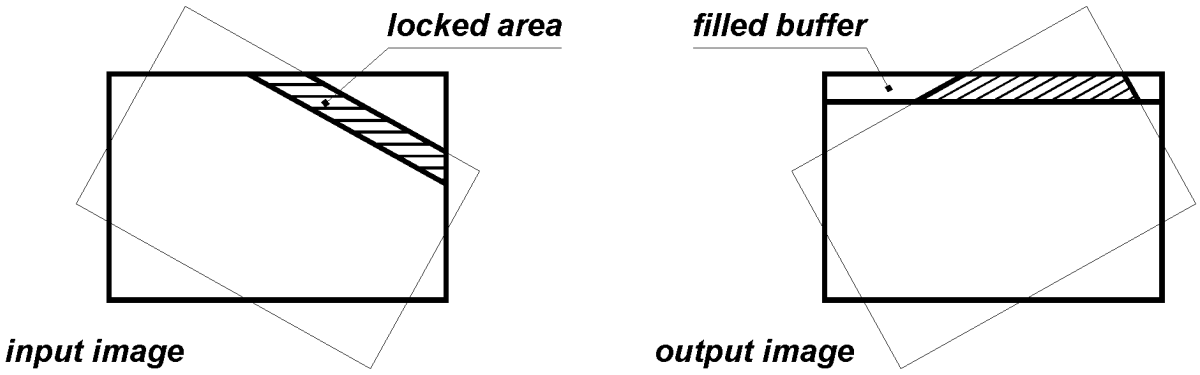

Figure 1: Correspondence between input and output images

Output image is subdivided into horizontal strips. The width of each strip is iteratively computed with the aim to utilize the maximum size of the **locked cache**. The required input image data are loaded to the **locked cache** directly from disk. Since input data are ready, calculation of the intensities in output horizontal strip is performed row by row and written immediately to a disk.

The bilinear interpolation or a simple method of nearest neighbour can be used for evaluation of output pixel intensities. These methods are convenient, if the average scale ratio between input and output image is about 1.0. In other cases we need to calculate the average value from certain quadrilateral area. We have found out during the implementation, that there are some problems with arithmetical precision concerning calculation of pixel values on output image boundaries. They cause the processing of pixels behind bounds of input image. This can be eliminated by careful processing bounds of output image rows.

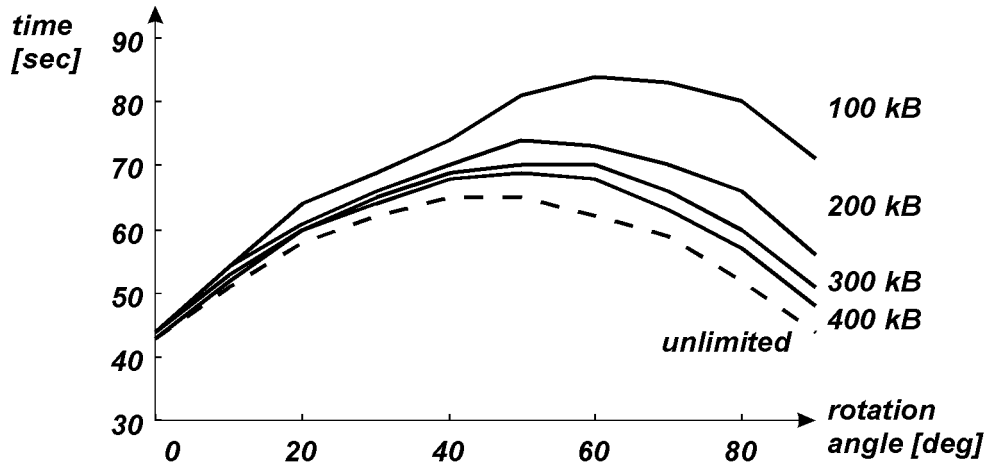

Figure 2: Total processing times for different sizes of the **locked cache**

Figure 3: System times for different sizes of the **locked cache**

3 Results

The implementation has been tested on rotation transformation of an image with 5.000×5.000 pixels (25 MB). The size of the output image was adjusted so that whole transformed input would not be cut off. Figure 2 shows total processing time graphs dependent on different rotation angles and different sizes of the **locked cache**. Computing time was measured on SGI machine Indigo^2 (R4000/100 MHz). Figure 3 shows times devoted to system operations (disk read/write ops.). Sizes of the locked cache were chosen as 1%, 3%, 5%, 7%, 9% and 48% of the input image size. We consider as very promising result, that using the cache more than 10 times smaller than input data, the processing time was increased only by factor 1.2.

4 Conclusion

The implemented method is able to transform images, which are much larger than available operating memory. High efficiency of the method is achieved by virtualization subsystem of two caches, which was designed especially for large uncompressed image files.

We have compared our approach with software systems for image processing. Aldus PhotoStyler 1.1 was chosen for processing of an image with size of 1 Mbyte (MS Windows, 8 MB RAM) and the speed was nearly the same, although our method has extremely low memory requirements (hundreds of kbytes only). Other tests performed using PhotoStyler on bigger images (9 Mbyte) consumed incredibly long times (several hours).

In our implementation, processing time depends linearly on the image size. The efficiency is remarkable especially on images several times bigger than RAM available.

References

[1] Wolberg, G.: *Digital Image Warping.* IEEE Computer Society Press, Los Amitos, CA, 1990. ISBN 0-8186-8944-7.