Implementation of Affine Warp Using TI DSP

Rama Pailoor and John Bartusek

ABSTRACT

Healthcare and medical research has seen increased usage of medical images in recent years. This usage often involves visualization of three-dimensional (3D) data sets as well as accurately relating information in different images for diagnosis, treatment and basic science. Image registration and volume rendering operations often involve some combination of rotation, scaling, shifting and shearing of images. The general transformation that carries out this operation is commonly referred to as Affine Warp or Affine Transform. Digital signal processors (DSP) offer a compelling power efficient and cost effective compute platform to carry out Affine Warp. This application report presents an algorithm for Affine Warp and provides details about its implementation on TI's TMS320C64x+™ DSP.

Contents

List of Figures

List of Tables

1 Introduction

Affine warp is a general scheme for carrying out combinations of basic image manipulation operations such as rotation, shifting, scaling, and shearing. It is a spatial transformation that maps locations in one image to another. In practice, this is carried out in the reverse order. i.e., manipulated or target image pixels are constructed by mapping their locations in the original or source image. Generally, this inverse mapping does not result in exact pixel locations in the source image. Therefore, target pixel values have to be computed by extracting and interpolating neighbor pixels in the source image. The first part of the report details the algorithm for computing the addresses of the source pixel location and their interpolation to produce target pixel values. This is followed by details on implementation of this algorithm on TI's C64x+™ DSP devices and corresponding benchmarks.

2 Background and Applications

Affine warp is used in a number of applications such as computer vision, image registration, and 3D volume visualization [9]. Affine warp transformations represent mappings from a two-dimensional (2D) object to a 2D image or to get an approximate 2D image of a planar object in 3D space. Therefore, Affine warp is a key operation in 3D volume rendering based on algorithms such Shear-Warp [7] and Shear Image Order [4]. In both of these rendering algorithms the re-sampling that needs to occur on individual image slices is carried out using an Affine warp transformation. Affine transform is also a key component in certain image registration algorithms [10].

3 Basic Operations

A general expression that relates source image locations to locations in the image transformed by Affine warp is as follows:

$$p_T = A \bullet p_S$$

where $p_S = \begin{bmatrix} x_S \\ y_S \\ 1 \end{bmatrix}$ is the source location, $p_T = \begin{bmatrix} x_T \\ y_T \\ 1 \end{bmatrix}$ is the corresponding location in the transformed image and the elements of the matrix $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix}$ are transformation parameters.

Transformation matrix for translation is:

$$A_T = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix}$$

Where d_x and d_y are the amounts of shift in x and y direction, respectively.

Transformation that scales the image is:

$$A_S = \begin{bmatrix} \beta_x & 0 & 0 \\ 0 & \beta_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where β_x and β_y are the scale factors along x and y direction, respectively.

Similarly, the relationship for rotation is:

$$A_R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where θ is the angle of rotation.

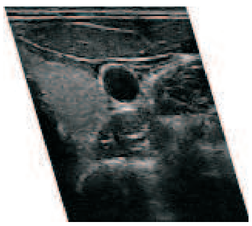
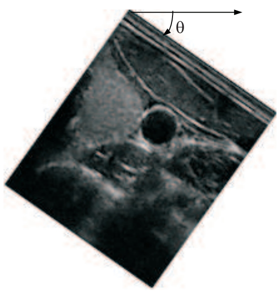
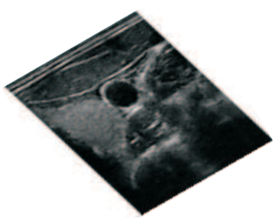
Shearing is also a linear transform described by the following relationship:

$$A_Q = \begin{bmatrix} 1 & s_x & 0 \\ s_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where s_x and S_y are parameters for shearing along x and y axes.

Table 1 provides examples of transformed image frames with primitive operations described earlier.

Table 1. Examples of Affine Warp Rransformation

	Original image frame
	After shearing along x-axis
	After rotating by θ
	An example of original image that has gone through a combination of primitive transformations: shifting, rotation, scaling and shearing.

4 Algorithm Details

Generally, the construction of transformed image is carried out by traversing its pixel locations sequentially row by row and computing the pixel values by getting the source location and interpolating the nearest neighbors in the source location.

Source locations are computed through the inverse transform.

$$p_S = A^{-1} \bullet p_T$$

$$A^{-1} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

In Figure 1, the blue grid is the image block to be generated by transforming a source image. The red grid is the source image block that completely encompasses the transformed image block.

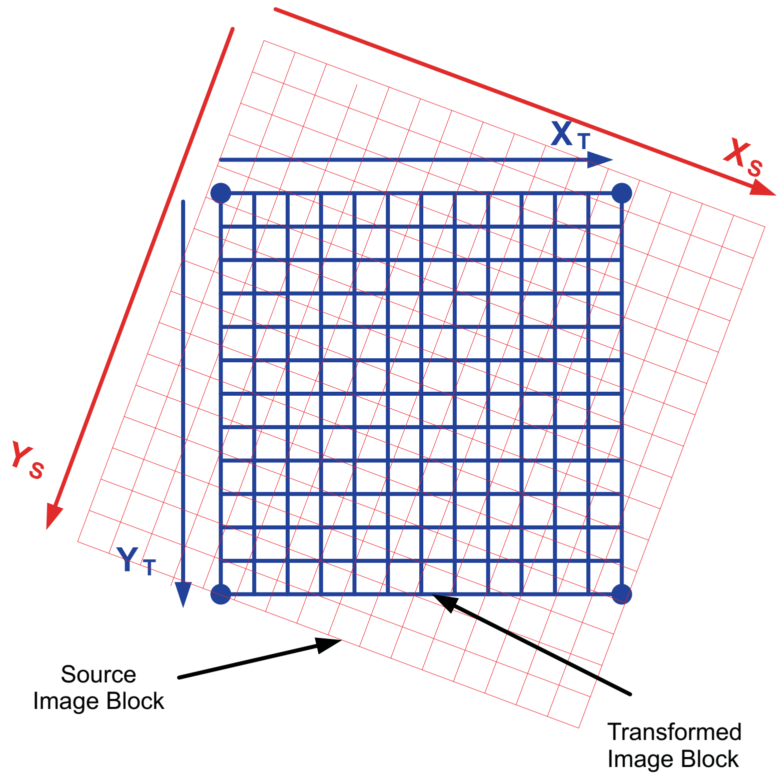


Figure 1. Transformed Image Block and Encompassing Source Image Block

Let the source location for [$x_T = 0$, $y_T = 0$] be [*xshift*, *yshift*]. Then,

$$\begin{bmatrix} xshift \\ yshift \\ 1 \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Therefore, b_{13} = *xshift* and b_{23} = *yshift*.

Let [*xtep_c, ystep_c*] be the change in source image block location when the transformed image block location is changed by one column.

Then,

$$\begin{bmatrix} xstep\_c \\ ystep\_c \\ 1 \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Therefore, b_{11} = *xstep_c* and b_{21} = *ystep_c*.

Similarly, let [*xstep_r* and *ystep_r*] be the change in source image block location when the transformed image block location is changed by one row.

Then,

$$\begin{bmatrix} xstep\_r \\ ystep\_r \\ 1 \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

This means, $b_{12} = xstep\_r$ and $b_{22} = ystep\_r$.

So, the source image block locations can be computed using the following relationship:

$$\begin{bmatrix} x_S \\ y_S \\ 1 \end{bmatrix} = \begin{bmatrix} xstep\_c & xstep\_r & xshift \\ ystep\_c & ystep\_r & yshift \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_T \\ y_T \\ 1 \end{bmatrix}$$

The above relationship is illustrated in Figure 2.

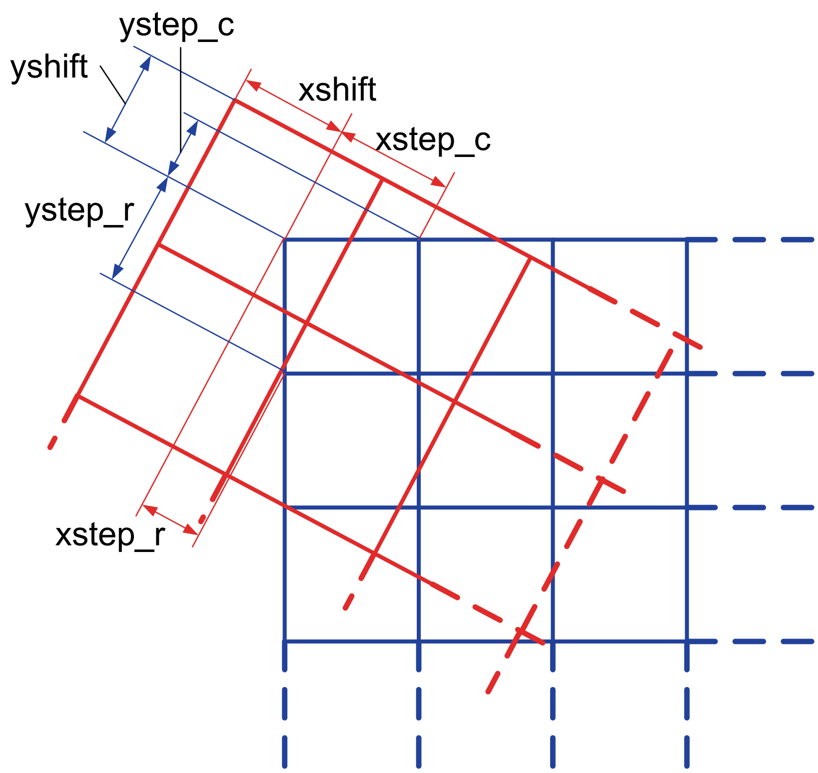


Figure 2. Affine Warp Parameters

```
Compute: xshift & yshift; // changes for every block
Compute:
xstep_c & ystep_c; // change in src x,y as we step across target cols
xstep_r & ystep_r; // change in src x,y as we step down target rows

xstart_r = xshift; ystart_r = yshift;

for (row = 0; row < TARGET_ROWS; row++)
{
      x = xstart_r; y = ystart_r;
      for (col = 0; col < TARGET_COLS; col++)
      {
            compute input addresses and fetch input data;
            compute coeffs;
            target_pixel[row, col] = interpolate;
            x += xstep_c; y += ystep_c;
      }
      xstart_r += xstep_r; ystart_r += ystep_r;
}
```

Figure 3. Affine Warp Algorithm for Computing an Output Image Block

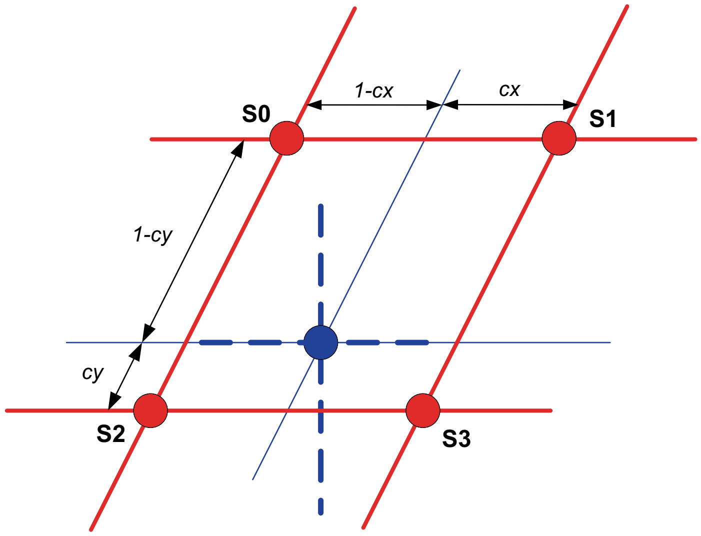


Figure 4. Computing Target Pixel Value Through Bilinear Interpolation of Nearest Neighbor Source Pixels

```
xi = floor(x); yi = floor(y);

// Fractional portions of source location are the interpolating
// coefficients
cx = x - xi; cy = y - yi;

// Get nearest four neighbors around source location.
S0 = input_pixel[xi,   yi  ];
S1 = input_pixel[xi+1, yi  ];
S2 = input_pixel[xi,   yi+1];
S3 = input_pixel[xi+1, yi+1];

// Bi-linear interpolation.
S01 = (1-cx)*S0 + cx*S1;
S23 = (1-cx)*S2 + cx*S3;
```

Figure 5. Affine Warp Interpolation Kernel Operations

5 Implementation

An optimized version of the Affine warp operation for the TI C64x+ DSP is implemented in [11]. These DSPs use a hierarchical memory architecture (L1, L2, DDR) and the Affine warp algorithm has been optimized for use on the C64x+ DSP through use of intrinsics [5], [8].

For a typical 3D rendering use case, images to be processed by Affine warp could be slices from a volume of size 256 by 256 by 256 voxels that reside in external DDR memory. To take advantage of the hierarchical memory architecture, limit the processing to a portion of the image slice that fits in the faster L2 memory. The idea is to divide the Affine warped output slice into multiple but smaller subblocks and transfer only the particular source image subregion that is required to process a particular output subblock.

Target images of size 64 x 64 were chosen as the smaller subblocks because source images that map to it typically fit in L2 memory. TI high-performance DSPs (C645x and C647x) have 512K bytes or more of L2 memory, so this leaves substantial internal memory for other uses by the higher level 3D rendering algorithm, which calls the Affine warp API. In a typical application, the higher level 3D rendering algorithm would be coordinating the DMA transfers for source slice inputs while in parallel calling the Affine warp

API to produce a 64 x 64 output block and then performing the steps related to the chosen rendering algorithm. Internal input ping-pong buffers would be allocated to prefetch via DMA one input source slice region from external DDR to internal L2 memory while a different source slice is being processed by the Affine warp API. In this manner, the Affine warped output slices can be constructed efficiently and various 3D rendering algorithms can be realized from the outputs.

The Affine warp algorithm assumes the source image block encompassing the target block is available in L2 memory. For this to happen, a higher level calling operation needs to compute indices for the minimum and maximum range of the x and y coordinates of the source image. For an example on performing this computation, see [1].

As for quantization of configuration parameters (the elements of the Affine warp matrix), 16 bits were used for the fractional part, which makes a good balance between precision and efficiency that is brought forth by using parallel operations inherent in the intrinsics.

The Affine warp implementation is highly scalable in the sense that it processes small output regions of the output slice and these operations can be parallelized via implementation on a multicore DSP.

6 Benchmarks

The util_affineWarpq API was tested on the TMS320C6455 EVM with a 64 by 64 output slice, and the total processing operations took 22,766 cycles, or 5.5 cycles per pixel.

7 Conclusion

A basic Affine warp utility function has been presented, which has applications in 3D rendering and image registration problems. Some aspects of the implementation have been discussed.

8 References

1. O. D. Evans and Y. Kim, *Efficient Implementation of Image Warping on a Multimedia Processor*, Real-Time Imaging 4, 417-428 (1998)
2. Edward Angel, Interactive Computer Graphics, Addison Wesley 2009.Yin Wu, Vishal Bhatia, Hugh Lauer, Larry Seiler, *Shear-Image Order Ray Casting Volume Rendering*, in Proc. 2003 Symp. Interact. 3D Graph., 2003, pp. 152–162.
3. Edward Angel, Interactive Computer Graphics, Addison Wesley 2009.
4. Yin Wu, Vishal Bhatia, Hugh Lauer, Larry Seiler, *Shear-Image Order Ray Casting Volume Rendering*, in Proc. 2003 Symp. Interact. 3D Graph., 2003, pp. 152–162.
5. *TMS320C6000 Programmer's Guide* (SPRU198)
6. *Hand-Tuning Loops and Control Code on the TMS320C6000* (SPRA666)
7. P. Lacroute and M. Levoy, *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*, in Proc. 21st Annu. Conf. Comput. Graph. Interact. Tech., 1994, pp. 451–458.
8. *TMS320C6000 Optimizing Compiler User's Guide* (SPRU187)
9. Y. Lamdan, J. Schwartz and H. Wolfson, *Affine Invariant Model-Based Object Recognition*, IEEE Trans. on Robotics and Automation, vol. 6, no. 5, pp. 578-589, October 1990.
10. ZOU Xiao-chun, ZHAO Xin-bo, Feng Yan, *An Efficient Medical Image Registration Algorithm Based on Gradient Descent*, 2007 IEEE/ICME International Conference on Complex Medical Engineering, pp 636-639.
11. *Using TI's Embedded Processor Software Toolkit for Medical Imaging (MED-STK)* (SPRABB8).