A Novel Approach to Real-time Bilinear Interpolation

K.T. Gribbon and D.G. Bailey
Institute of Information Sciences and Technology,
Massey University, Private Bag 11-222
Palmerston North, New Zealand
k.gribbon@massey.ac.nz, d.g.bailey@massey.ac.nz

Abstract

Bilinear interpolation is often used to improve image quality after performing spatial transformation operations such as digital zooming or rotation. In the traditional case where the input coordinates appear in a raster-based fashion, the required pixel values can be obtained from the previous calculation, the frame buffer and a single line cache. This paper presents a novel approach to performing real-time bilinear interpolation that is useful in applications such as lens distortion correction where the input coordinates follow a curved path that spans multiple rows. To help retrieve the required pixels in a single clock cycle under imposed data bandwidth constraints a unique caching system has been devised. In the event that constraints make it impossible to obtain the four required pixel values, the approach performs a modified three-point interpolation. An example field programmable gate array implementation of the bilinear interpolation method used in conjunction with a lens distortion correction algorithm has been successfully completed.

1. Introduction

A common class of image processing operations used in image restoration are spatial transformations, which redefine the "arrangement" of pixels on the image plane [1]. Example transformations are rotation, zooming and operations to correct for geometrical image defects such as perspective and radial lens distortion.

A spatial transformation can be thought of as a mapping function according to the equation:

$$\mathbf{p}' = f(\mathbf{p}) \tag{1}$$

where

$$\mathbf{p}' = \begin{bmatrix} x' \\ y' \end{bmatrix} \text{ and } \mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$$

The distorted and undistorted coordinates are represented by (x, y) and (x', y') respectively. Equation (1) is a forward mapping effectively giving the coordinates in the undistorted image as a function of those of the distorted image. This form is

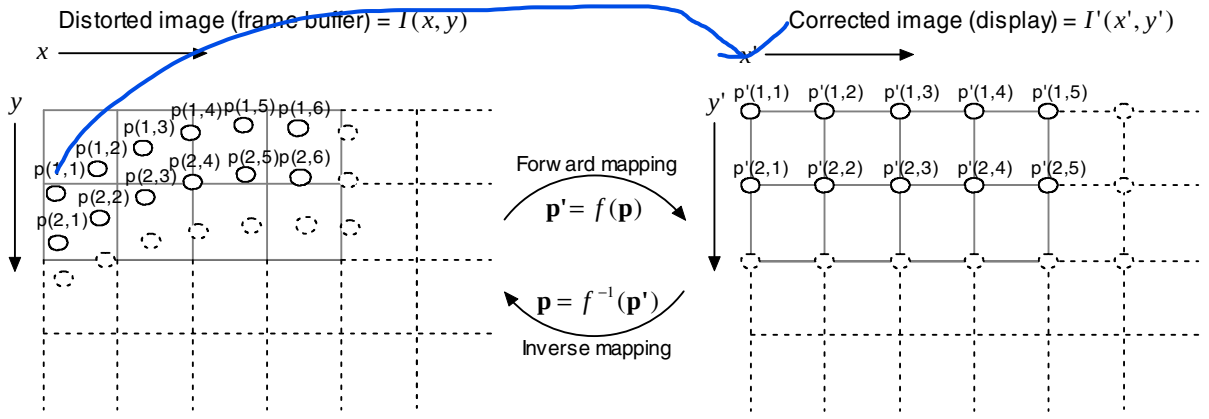

Figure 1. Illustration of forward and inverse mapping between distorted and corrected image

Proceedings of the Second IEEE International Workshop on Electronic Design, Test and Applications (DELTA 2004)
0-7695-2081-2/04 $20.00 © 2004 **IEEE**

unsuitable for correction since the discrete pixel locations in the distorted image do not necessarily correspond directly to the pixels in the corrected output image. This can result in problems such as patches of undefined pixels or overlapping values on a single output pixel [2].

Instead, the inverse mapping function, f^{-1} is required so that the coordinates in the undistorted image can be used to determine which pixel in the distorted image to display. Essentially, every pixel in the corrected image needs to obtain its value from the corresponding point in the distorted image as illustrated in figure 1. Therefore, the equation needs to be of the form:

$$\mathbf{p} = f^{-1}(\mathbf{p}') \tag{2}$$

As the coordinates calculated by the inverse mapping function are rarely integer values, their location lies "between" the pixels in the original image. Simplistic methods that round or truncate the fractional component of the calculated coordinate can introduce substantial error in the pixel location, one effect of which distorts lines by producing jagged-edge artefacts.

One method that can be used to deal with this problem is bilinear interpolation, which is often used in many image processing applications because it provides a compromise between computational efficiency and image quality [1]. The algorithm obtains the pixel value by taking a weighted sum of the pixel values of the four nearest neighbors surrounding the calculated location as shown below in figure 2 and equation (3):

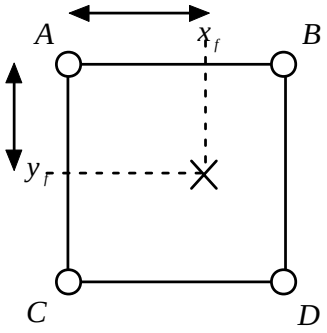

Figure 2. Bilinear interpolation neighborhood

$$I'\left(\begin{bmatrix} x' \\ y' \end{bmatrix}\right) = I\left(f^{-1}\left(\begin{bmatrix} x' \\ y' \end{bmatrix}\right)\right) = I\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) \tag{3}$$

$$= (1-x_f)(1-y_f)A + x_f(1-y_f)B + y_f(1-x_f)C + x_f y_f D$$

where

$$A = I\left(\begin{bmatrix} x_i \\ y_i \end{bmatrix}\right) \qquad B = I\left(\begin{bmatrix} x_i+1 \\ y_i \end{bmatrix}\right)$$

$$C = I\left(\begin{bmatrix} x_i \\ y_i+1 \end{bmatrix}\right) \qquad D = I\left(\begin{bmatrix} x_i+1 \\ y_i+1 \end{bmatrix}\right)$$

and $x = x_i + x_f$ and $y = y_i + y_f$ with x_i and y_i being the integer components of x and y and x_f and y_f being the fractional components ($0 \le x_f, y_f < 1$).

This paper presents the design and field programmable gate array (FPGA) implementation of a new approach to real-time bilinear interpolation where input coordinates can appear on smooth curves and lines with non-uniform step size. Section two describes the constraints that such an implementation imposes. The detailed design of the interpolation algorithm and caching system is discussed in section three and implementation details such as pipelining are described in section four. Finally, implementation results are presented in section five.

2. Constraints

Bilinear interpolation is a trivial task to perform "offline" or in software but a real-time implementation often imposes several constraints that can complicate the design.

2.1. Real-time constraints

A real-time implementation constrains the design into performing all of the required calculations for each pixel at the pixel clock rate. The need to produce one pixel every 40 ns (VGA output) precludes the use of a DSP or other serial processor as the implementation platform. An FPGA is a better choice but producing one pixel every 40 ns is still difficult to achieve because for each pixel, equations (2) and (3) must be evaluated. This can introduce significant propagation delay, which may easily exceed a single pixel clock cycle.

A pipelined approach is thus needed that accepts an input coordinate with fixed precision and outputs an interpolated pixel value every clock cycle with several clock cycles of latency between the input and output. This allows several pipeline stages each for the evaluation of the inverse mapping (equation (2)) and for the interpolation.

2.2. Memory bandwidth constraints

In general, a spatial transformation requires that the image be buffered because the order that the pixels are required for display does not correspond directly to the order in which they are input. The size of the frame buffer depends on the transform itself. In the worst case (rotation by 90º, for example) the whole image must be buffered. The rest of this paper therefore assumes that the input to the spatial transformation is from a frame buffer.

Bilinear interpolation requires simultaneous access to four pixels from the input image (see figure (2) and equation (3)). However, only a single access can be made to the frame buffer per clock cycle. Possible alternatives to deal with this problem are the use multiport RAM, multiple RAM banks in parallel or using a faster RAM clock to read multiple locations in a single pixel clock cycle.

These alternatives all have significant disadvantages. Multiport RAM is specialized and expensive. The use of multiple banks is clumsy because the added redundancy is expensive in both cost and space. Finally, using a faster RAM clock requires expensive high speed memory and introduces synchronization issues. The approach in this paper avoids all these complications by caching data read from the frame buffer that is likely to be used in subsequent calculations.

2.3. Problem constraints

A simple example of the use of caching is in the real-time implementation of window filters. Input data from the previous few rows is cached using a shift register (or circular memory buffer) for when the window is scanned along subsequent lines. Bilinear interpolation (equation (3)) is effectively a 2×2 filter with weights that vary according to the location within the image.

However, for spatial transformations the caching will be more complicated because the input data is not generally read in a raster fashion and is also dependent on the required transformation. For example, in simple zooming operations the calculated coordinates will be presented in a raster-based fashion, which makes it easy to obtain the required pixel values for interpolation. These values can be obtained from the previous calculation, the frame buffer and a single row cache.

Pure rotation operations introduce additional complications because the coordinates now appear on straight lines at the angle of rotation, one pixel apart. Perspective correction is even more complicated because although the coordinates appear on straight lines, the angle of these lines depends on the scan position within the image.

The approach detailed in this paper focuses on the more general case where the step size is not uniform and coordinates appear on smooth "curves", as illustrated in figure 1. Curved coordinate paths are typical of applications such as lens distortion correction and this makes traditional row caching ineffective because of the curved nature of the mapping.

It must be noted that if the coordinate step size is greater than one pixel, any caching arrangement will be limited in its effectiveness because pixels fetched from the frame buffer are less likely to be reused. This paper will focus on the case where the step size between adjacent pixels and rows is less than one input pixel.

3. Design

The design of the bilinear interpolation algorithm focused on how to devise a caching arrangement that provided all four input samples for performing the interpolation with only a single access to frame buffer memory. To determine what pixel values are needed, an analysis of the various scenarios relating to the current and previous coordinates was undertaken.

3.1. Scenarios

Assuming that as the output image is scanned, the distorted image is traversed in a predominantly left to right and top to bottom direction, there are six possible relationships between the current and previous coordinates as shown in figure 3.

Scenario one is the simplest as all four pixel values from the previous interpolation calculation are available. In scenarios two, three and four only two pixels are available and in scenario five and six only one pixel is in common with the previous calculation. One of these unknown values can be read from the frame buffer but the others must be obtained from the cache.

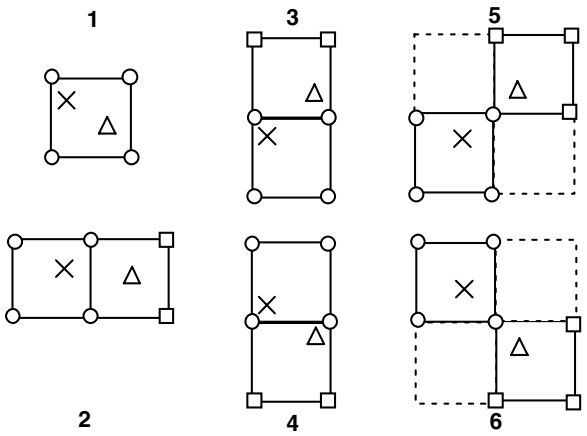

Figure 3. Possible scenarios derived from the relationship between the current and previous coordinate where
× = Previous coordinate
Δ = Current coordinate
○ = Pixel value from previous calculation
□ = Pixel values that need to be obtained

3.2. Caching

The cache is constructed with a length equal to the number of horizontal pixels in the input image. The integer part of the x-coordinate is used as the address with the y-offset and pixel value stored at the corresponding cache location. The cache is implemented using dual-port RAM so that pixel values may be read and updated in a single clock cycle.

The pixel value to be written into the cache on each clock cycle must be chosen carefully in order to maximise the likelihood of it being used for interpolation on the next line. Since output to the display is performed in a raster-based fashion, when scanning smoothly through the image the warped scanlines will shift in the same direction by less than one pixel. That is, coordinates will predominantly appear going from left to right and downwards even if a curved path is taken for the line.

Based on this assumption and on figure 3 it is useful to write the bottom-right pixel value used in the current interpolation to the cache as it is more likely to be used in subsequent calculations. Therefore, caching a single scanline will ensure the retrieval of one of the missing pixel values. As the next row will be less than one pixel apart, only a single bit of the y-coordinate is needed for the y-offset.

Figure 4(a) shows one possible scenario and how to retrieve the four surrounding pixels in a single clock cycle. The pixel values in the left column of the neighbourhood are obtained from the previous interpolation. After reading the cache contents of the address corresponding to the current location, the y-offset indicates that it corresponds to the bottom-right pixel of the neighbourhood. Thus we need to read the top-right pixel from the frame buffer. The bottom-right pixel is rewritten to the cache. This is just one example as the cache may contain either the top-right or bottom-right pixel and this will select which pixel location is read from the frame buffer.

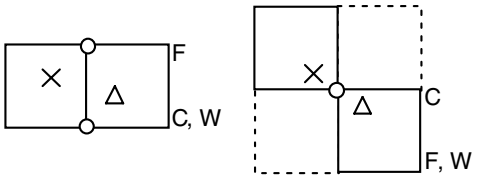

Figure 4. Example scenarios where
F = Pixel value read from frame buffer memory
C = Pixel value read from cache
W = Pixel value written to cache

In some scenarios it may not possible to retrieve all four neighbouring pixel values and therefore a modified three-point interpolation must be used. This is only a possibility in scenarios five and six (figure 3), where the top-left or bottom-left pixel value may not be available.

An example of this scenario is shown in figure 4(b). Here, the top-right and bottom-right pixel values are read from the cache and the frame buffer respectively, but we only have one pixel value in common with the previous interpolation and therefore the bottom-left pixel value cannot be retrieved.

It can be shown that there will always be at least three-points available and therefore the unknown corner value can be estimated from the three known points using a planar patch as shown in figure 5 and equation (4).

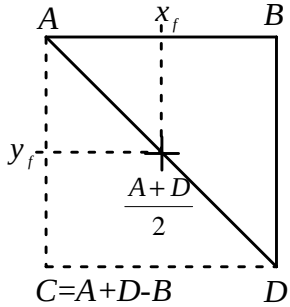

Figure 5. Illustration of the derivation for an unknown corner pixel value

$$C = B + 2\left(\frac{A+D}{2} - B\right) = A + D - B \quad (4)$$

Substituting equation (4) into equation (3) gives:

$$I\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = (1 - x_f)A + (x_f - y_f)B + y_f D \quad (5)$$

A similar equation can be derived for scenario five when the top-left pixel value cannot be retrieved.

4. Implementation

An example FPGA implementation of the bilinear interpolation method detailed above has been completed. It was designed to work in conjunction with an implementation of a lens distortion correction algorithm that was developed at Massey University [3].

4.1. Platform

To satisfy real-time constraints (section 2.1) the algorithm must be implemented in hardware. A fixed hardware approach using an application specific integrated circuit (ASIC) is not appropriate for prototyping the design making field programmable gate arrays (FPGA) a better choice.

The hardware used to support this implementation is the RC100 prototyping and development board from Celoxica, which incorporates a Xilinx Spartan-II FPGA, video decoder, offchip RAM and video DAC.

4.2. Design Entry

FPGA design entry can be achieved using low-level development methods such as schematic capture or hardware description languages. Another alternative is to use system-level design languages such as Celoxica's Handel-C that take a more high-level approach [4].

Handel-C is a C-based language that allows developers to express their design in a more algorithmic manner and with little knowledge of the underlying hardware. Traditionally, algorithms are prototyped in higher-level software programming languages like C before being ported into VHDL or Verilog for implementation. The porting process can increase the risk of errors but Handel-C avoids this problem by directly compiling from the high-level algorithmic design to hardware [5].

The bilinear interpolation algorithm was programmed entirely in Handel-C.

4.3. Fixed point

A fixed point representation was used. Therefore all variables were represented as signed or unsigned integers and were commented to indicate the position of the binary point. When arithmetic operations were performed operands were explicitly shifted to ensure alignment.

4.4. Pipeline

Handel-C is implicitly a register transfer level language because each assignment is registered after being evaluated. However, the need to explicitly break the algorithm down into pipeline stages required moving the design from a higher-level algorithm to a relatively low-level data flow approach. A diagram of the four-stage pipeline is shown in figure 6.

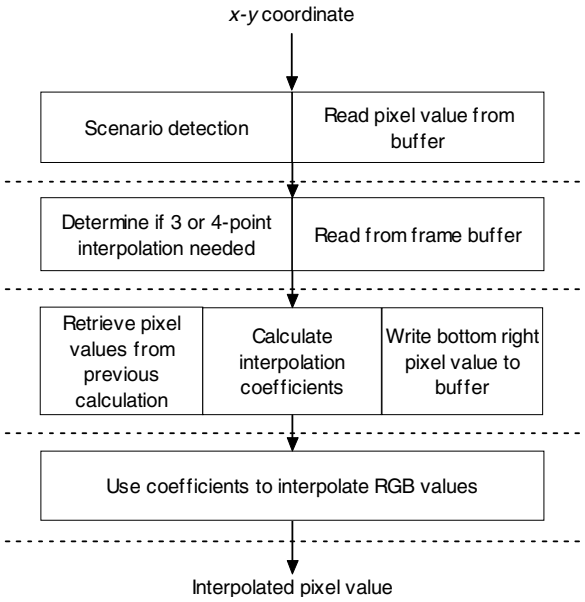

Figure 6. Pipeline for interpolation algorithm

The first pipeline stage of the algorithm uses the input x-y coordinates to determine which of the scenarios in figure 3 applies. In parallel it also reads from the cache a value from the right pixel column of the four-point neighborhood. This will be either the top-right or bottom-right pixel.

In stage two, the y-offset read from the cache is used to determine whether the top or bottom pixel was read. The other pixel is then read from the frame buffer. Using the scenario information determined in stage one, the algorithm selects whether to perform a

three or four-point interpolation and what coefficients are needed.

Stage three performs three tasks in parallel. Using the scenario information from stage one again, pixel value(s) in common with the previous calculation are retrieved and the interpolation coefficients are calculated. The bottom-right pixel value is also written to the buffer in this stage.

In the final stage the calculated coefficient values are used to determine the interpolated pixel value. The red, green and blue components of the pixel are calculated concurrently.

4.5. Pipelining issues

Some of the stages in the pipeline are constrained because they rely on results from a previous stage. For example, it is only after determining the location of a pixel value read from the cache that the location to read from in the frame buffer can be determined.

When starting a frame the cache will not have the values needed for the first line. Therefore it is necessary to prime the cache by sending the first row of coordinates twice and discarding the erroneous interpolated pixels. This can be accomplished during the vertical blanking period.

5. Results

The bilinear interpolation algorithm has been mapped to hardware. The utilization of the device is shown in Table 1.

Table 1. Resource utilization of target device (XC2S200)

CLBS (1172 total)	BlockRAM (14 total)
329 (28%)	3 (22%)

The interpolation algorithm uses 28% of the device with most of this being used to implement the multipliers and adders in the interpolation stage of the pipeline (figure 6). This leaves significant resources to perform the transformation or other tasks.

6. Summary

It is often useful to perform bilinear interpolation to improve image quality after carrying out spatial transformation operations. However, for certain applications such as lens distortion correction, bilinear interpolation is complicated by the fact that input coordinates do not appear on straight lines. This is in addition to the usual constraints imposed by a real-time implementation with limited resources.

In order to help retrieve needed pixel values to perform interpolation on a curved coordinate path, this paper has discussed a unique caching system devised from analyzing the relationship between the current and previous input coordinates. The other pixel values are obtained from frame buffer memory and the previous interpolation calculation.

An example FPGA implementation of the interpolation algorithm used in conjunction with lens distortion correction [3] has been successfully completed on a Spartan-II FPGA demonstrating that real-time bilinear interpolation is achievable at VGA rates (sixty frames per second) even with modest resources.

Several improvements can still be made to the algorithm and implementation. For example, the use of the three-point instead of the normal four-point interpolation introduces additional error into the calculated pixel value. Therefore, it would be advantageous to remove the three-point interpolation cases. It may be possible to do this with the use of a second line cache but this would come at additional cost to block RAM resources.

7. References

[1] Gonzalez, R.C., Woods, R.E., *Digital Image Processing*, 2nd Edition, Prentice-Hall, New Jersey, 2002.

[2] Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, 1990.

[3] Gribbon, K.T., Johnston, C.T., Bailey, D.G. "A Real-time FPGA Implementation of a Barrel Distortion Correction Algorithm with Bilinear Interpolation". *Proc. Image and Vision Computing NZ*, 2003, pp. 402-407.

[4] Alston, I., Madahar, B., "From C to netlists: hardware engineering for software engineers?", *IEE Electronics & Communication Engineering Journal*, August 2002, pp 165-173.

[5] Celoxica Ltd., *HANDEL-C Language Overview,* http://www.celoxica.com, 2002.