Project Proposal
Augmented Reality Image Processing System

Logan P. Williams & José E. Cruz Serrallés

November 03, 2011

1 Abstract

In our project, we will implement an augmented reality system that can overlay a digital image on video of a real world environment. We begin by reading NTSC video from a video camera and storing it in ZBT SRAM. A picture frame with colored markers on the corners is held in front of the camera. We then perform chroma-based object recognition to locate the co-ordinates of the corners. Using these co-ordinates, we apply appropriate translation, scaling, rotation, and anti-aliasing FIR filters to fit the image to the boundary of the frame. If time allows, we will use a non-linear projective transformation to correct for perspective. We then output VGA video of the original captured image, with the processed image overlayed on top of the frame. The overlayed image (the "augmentation") can be arbitrary. When this image is the frame of video that was previously displayed, we call the system "recursive", as we obtain the same image contained within itself.

2 Top-Level Block Diagram

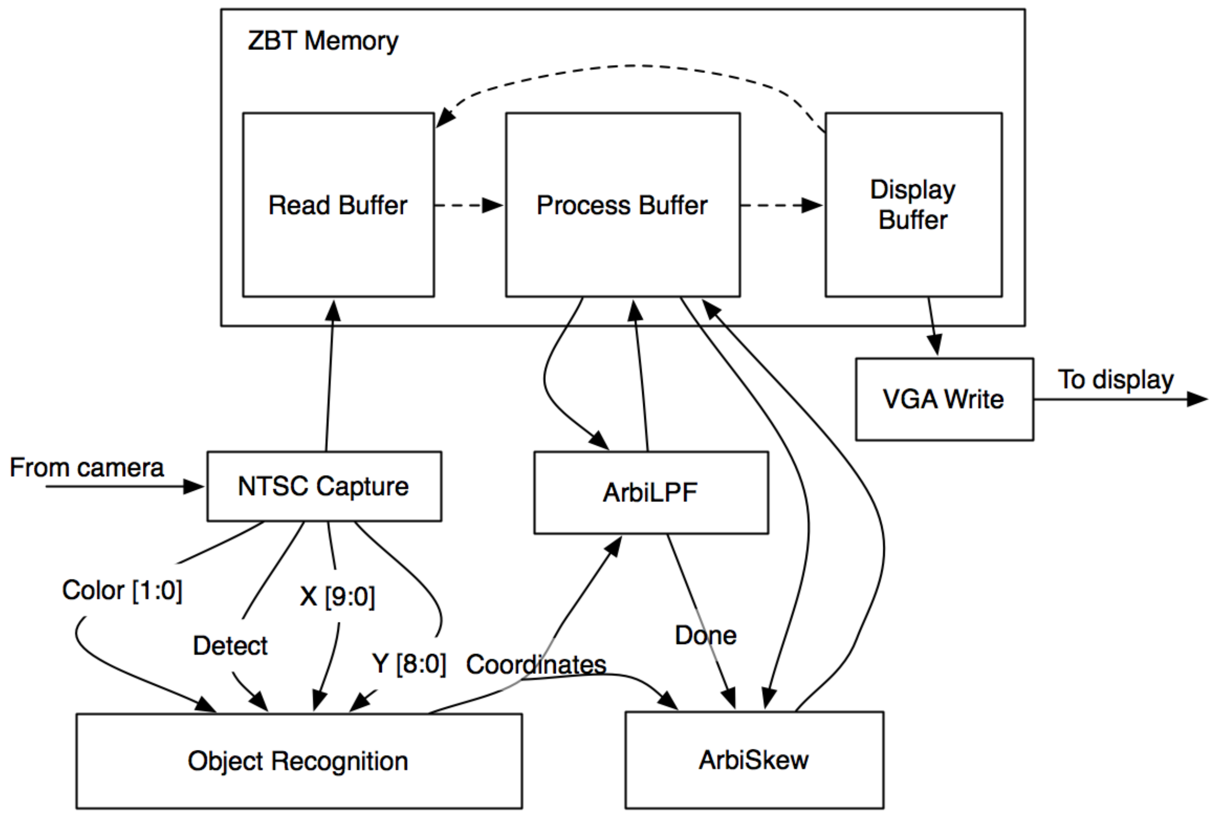

Figure 1: The block diagram of the augmented reality system

3 Submodules & Division of Labor

3.1 NTSC Capture

3.2 ZBT Memory

3.3 Object Recognition

3.4 ArbiLPF (Josè)

The inputs to ArbiLPF are (1) the previously displayed image and (2) the four coordinates of the dotted frame as detected from the NTSC output of the camera in the Object Recognition module. AbiLPF calculates the maximal amount by which the skewing algorithm will shrink the image, which will be referred to as M. ArbiLPF then applies a two-dimensional low-pass filter to the image, with a radial cutoff frequency of $\frac{\pi}{M}$, in order to avoid aliasing in the ArbiSkew module.

Based on the downsampling factor M, the filter will select a set of coefficients from a lookup table and convolve the image values with these coefficients. This table of coefficients will correspond to the coefficients of two-dimensional extrapolations of one-dimensional FIR Parks-McClellan filters with cutoff frequencies of $\frac{\pi}{M}$. Due to the limited number of multipliers on the FPGA, these two-dimensional filters will be constrained to have at most 144 coefficients, which constrains the one-dimensional filters to have at most 12 coefficients. Due to these constraints, the ripple and transition width specifications of the 1D filters will have to be lax. The radial symmetry of these 2D filters will be exploited to reduce the number of required multiplications by a factor of 4, to at most 36 multiplications per color per pixel.

3.5 ArbiSkew (Logan)

The inputs to ArbiSkew are (1) a reference pointer to the location in ZBT memory where the low-passed image begins, and (2) the four coordinates of the corners of the frame provided by the Object Recognition module.

This function maps the original rectangular image to any convex quadrilateral, provided that all sides of the destination quadrilateral are shorter than the original, which is inherent in the overall system. A graphic representation of the transformation is shown in the figure below:

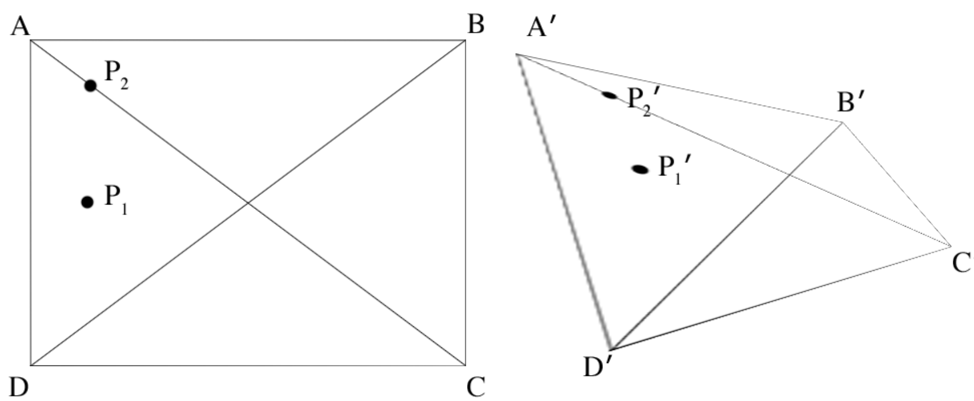

Figure 2: A visual representation of the result of the ArbiSkew module. Input is on the left, a possible output, for four coordinates $A\prime$, $B\prime$, $C\prime$, and $D\prime$ is on the right.

Mathematically, the algorithm works as follows:

1. Calculate the distance of line $\overline{A\prime D\prime}$ and assign it to d_{ad}.
2. Do the same for $\overline{B\prime C\prime}$ and assign it to d_{bc}.
3. Create two "iterator points," point I_A and I_B initially located at $A\prime$ and $B\prime$.
4. Let $o_x = 0$ and $o_y = 0$
5. Calculate the distance between the iterator points, assign it to d_i.
6. Create a third iterator point, I_C at the location I_A.
7. Assign the pixel value of I_C to pixel (o_x, o_y) in the original image.

8. Move I_C along line $\overline{I_A I_B}$ by an amount $= \frac{d_i}{width_{original}}$.
9. Increment o_x.
10. Repeat steps 7–9 until $I_C = I_B$.
11. Move I_A along line $\overline{A\prime D\prime}$ by an amount $= \frac{d_{ad}}{height_{original}}$.
12. Move I_B along line $\overline{B\prime C\prime}$ by an amount $= \frac{d_{bc}}{height_{original}}$.
13. Increment o_y.
14. Repeat steps 5–13 until $I_A = D\prime$ and $I_B = C\prime$.

This is feasible on the FPGA by using lookup tables to calculate sin, cos, and arctan for angle calculations. Besides that, it needs a relatively small number of multiplications, just two per pixel in the original image, and four per line in the original image. There is also a square root that is needed once per line, this can be implemented with either a look up table, or by using an iterative method of calculation.

3.6 VGA Write

4 External Components

5 List of Goals