# Augmented Reality on FPGA

## Realtime Object Recognition and Image Processing

Logan Williams  José E. Cruz Serrallés

15 November 2011

# Introduction

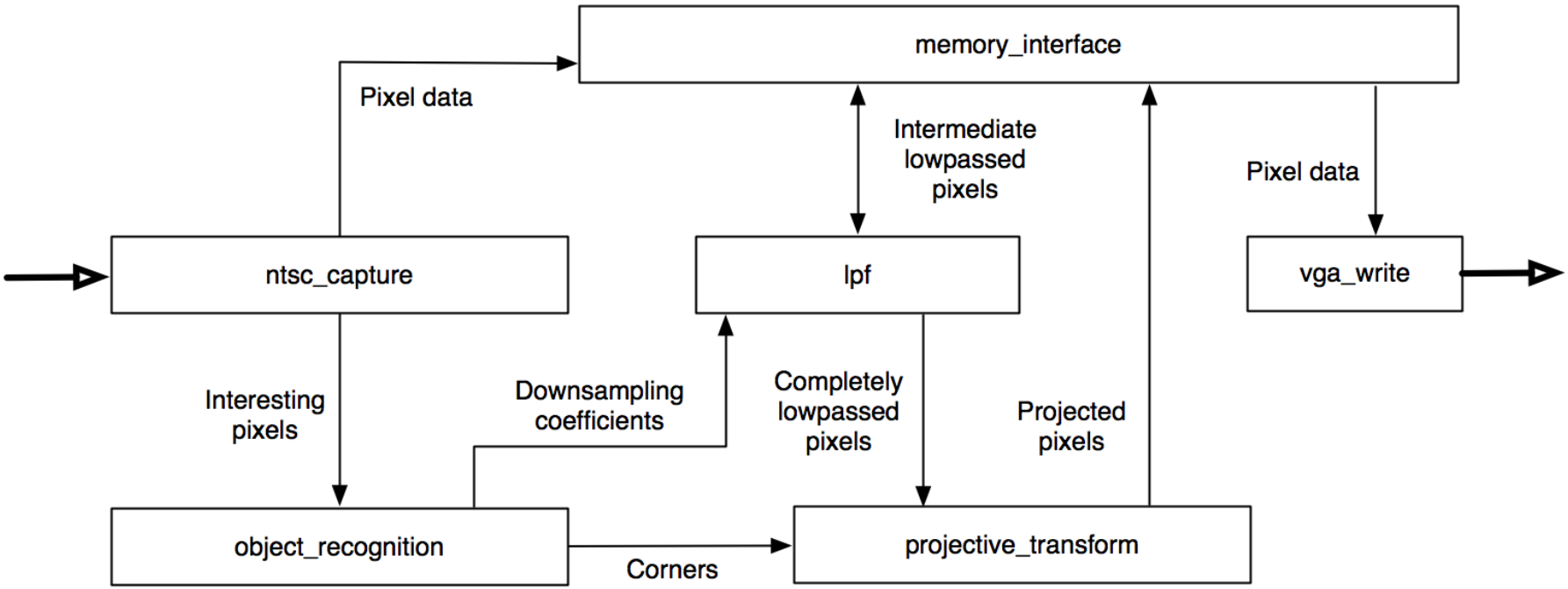
- Overlay a digital image on a physical object in realtime.
- In this case, we want to identify a picture frame in captured video, and output video with another image distorted to fit on top of the picture frame.

# Example Image

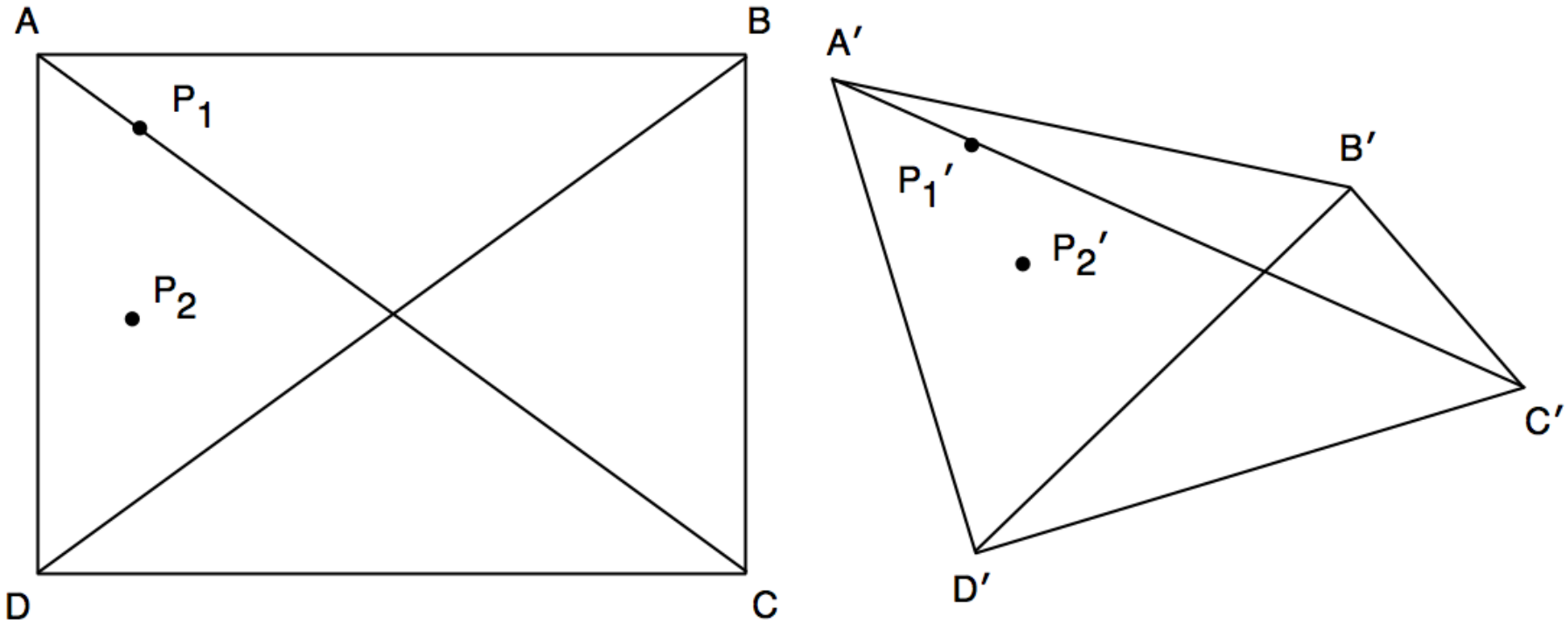
# Example Image

# Example Image

# Top-Level Overview

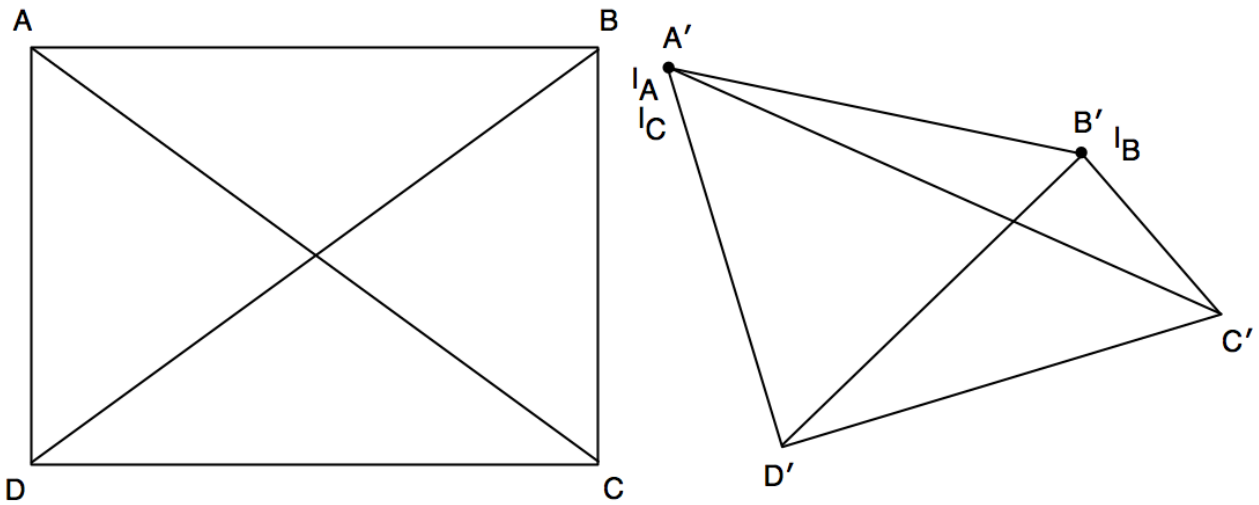


# projective_transform: Purpose



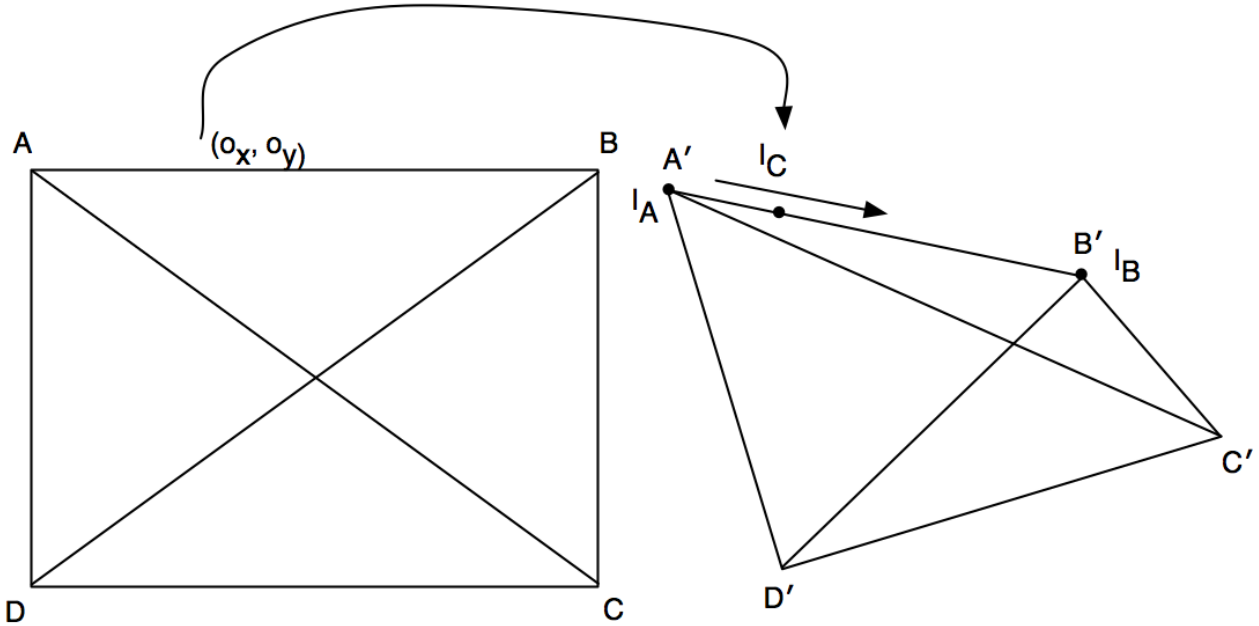

- Skew to any arbitrary convex quadrilateral

# projective_transform: How the algorithm works



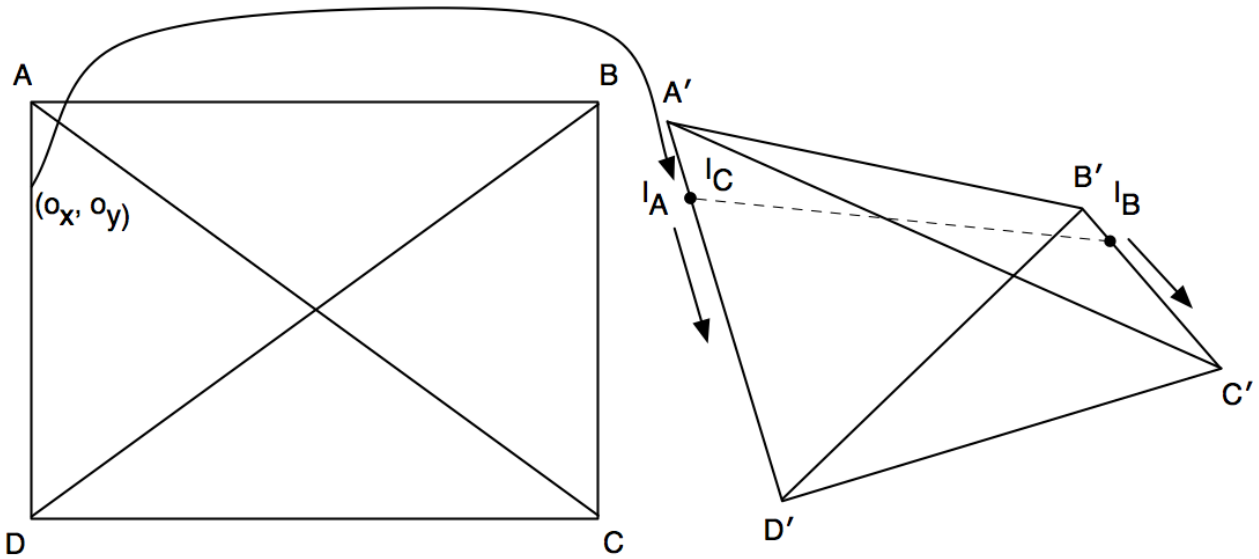

1. Calculate the distance of line $\overline{A\prime D\prime}$ and assign it to $d_{ad}$.
2. Do the same for $\overline{B\prime C\prime}$ and assign it to $d_{bc}$.
3. Create two "iterator points," point $I_A$ and $I_B$ initially located at $A\prime$ and $B\prime$.
4. Let $o_x = 0$ and $o_y = 0$
5. Calculate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

# projective_transform: How the algorithm works




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 $I_C = I_B$.

# `projective_transform`: How the algorithm works



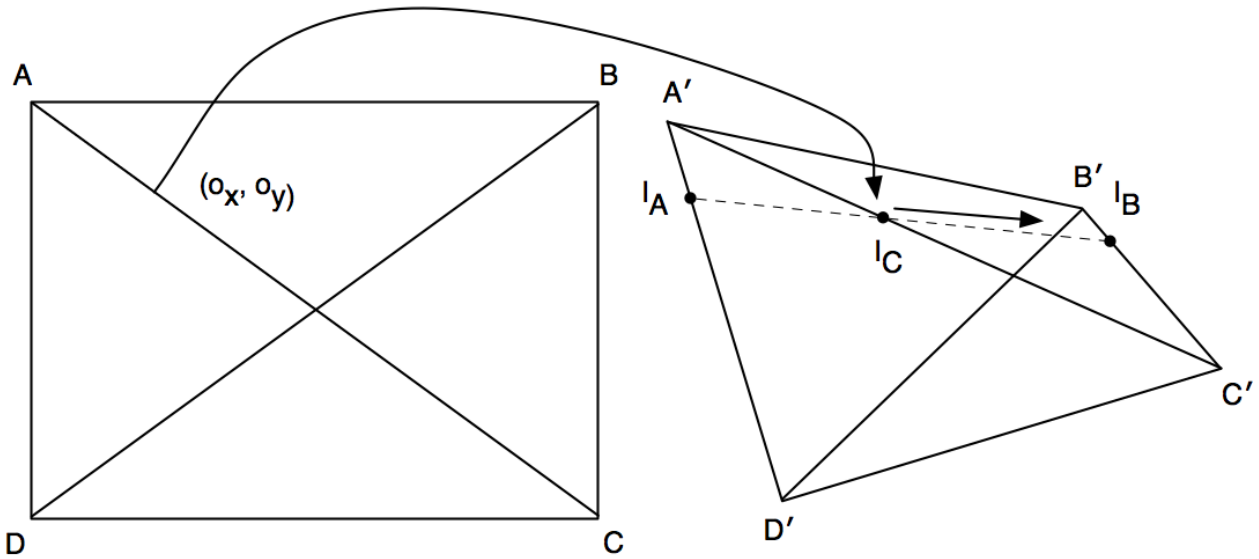

11 Move $I_A$ along line $\overline{A\prime D\prime}$ by an amount $= \frac{d_{ad}}{height_{original}}$.

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 $I_B = C\prime$.

# `projective_transform`: How the algorithm works




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 $= \frac{d_{bc}}{height_{original}}$.

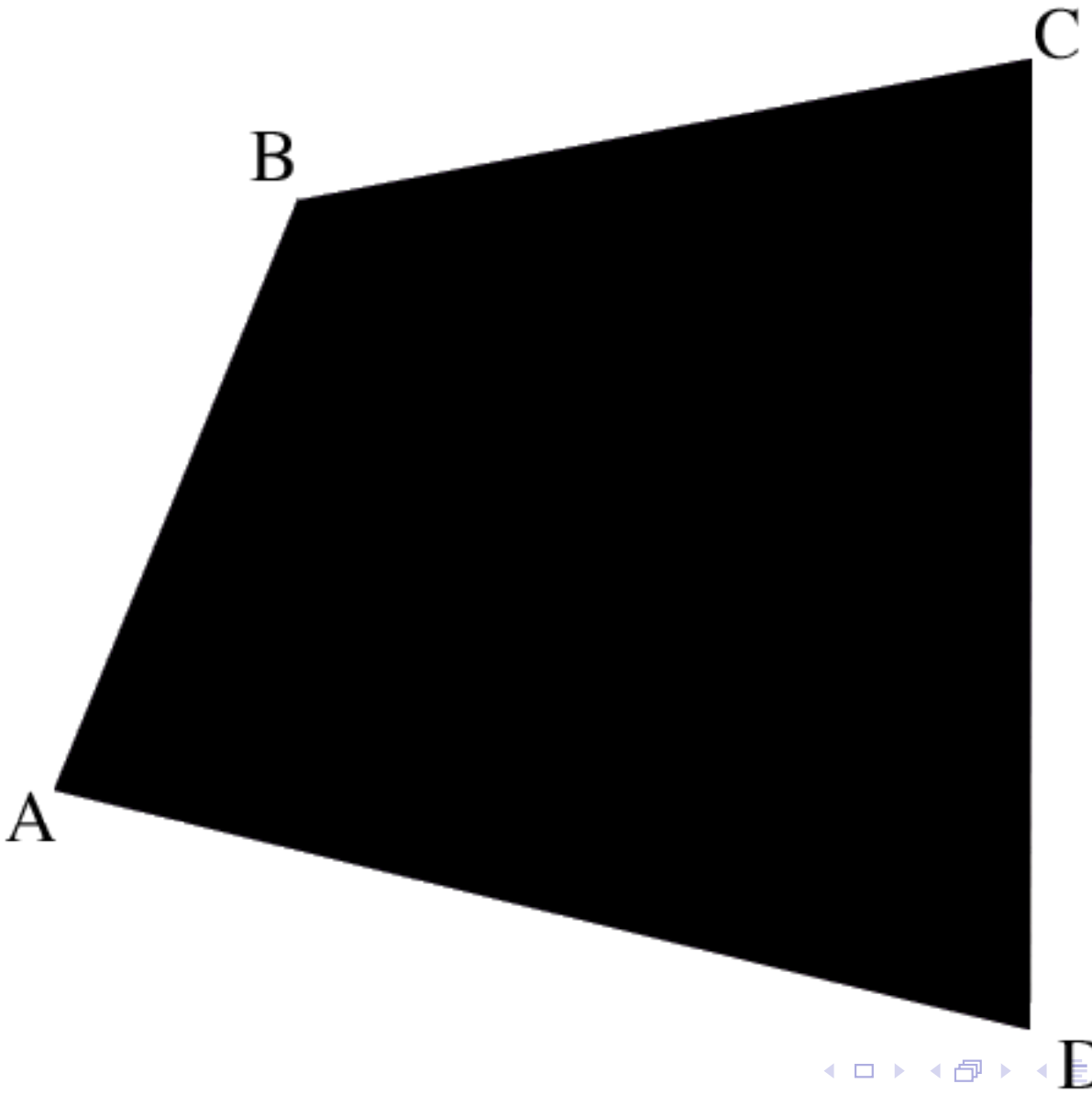
13 Increment $o_y$.

14 Repeat steps 5–13 until $I_A = D\prime$ and $I_B = C\prime$.

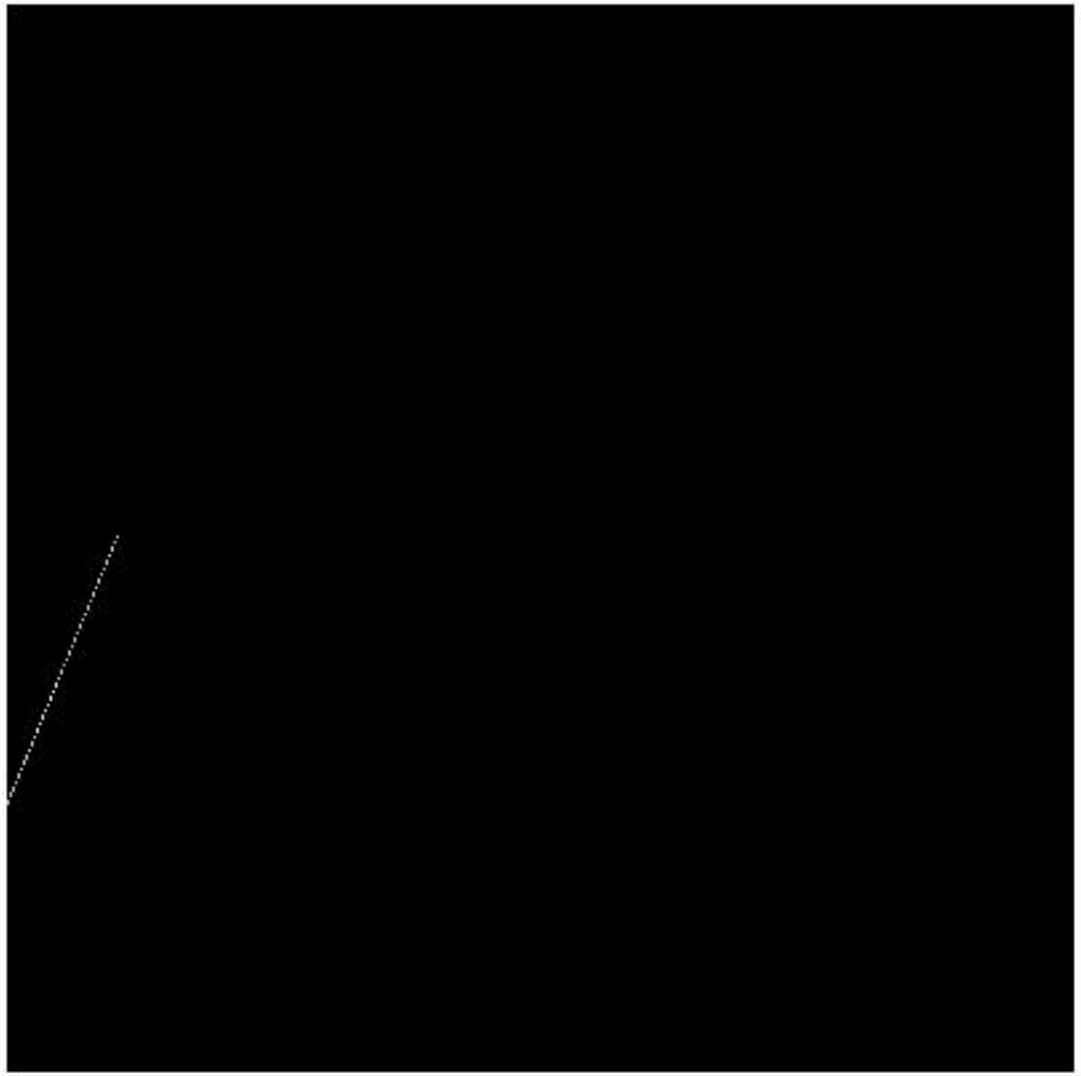
# projective_transform: Example

Figure: The original image

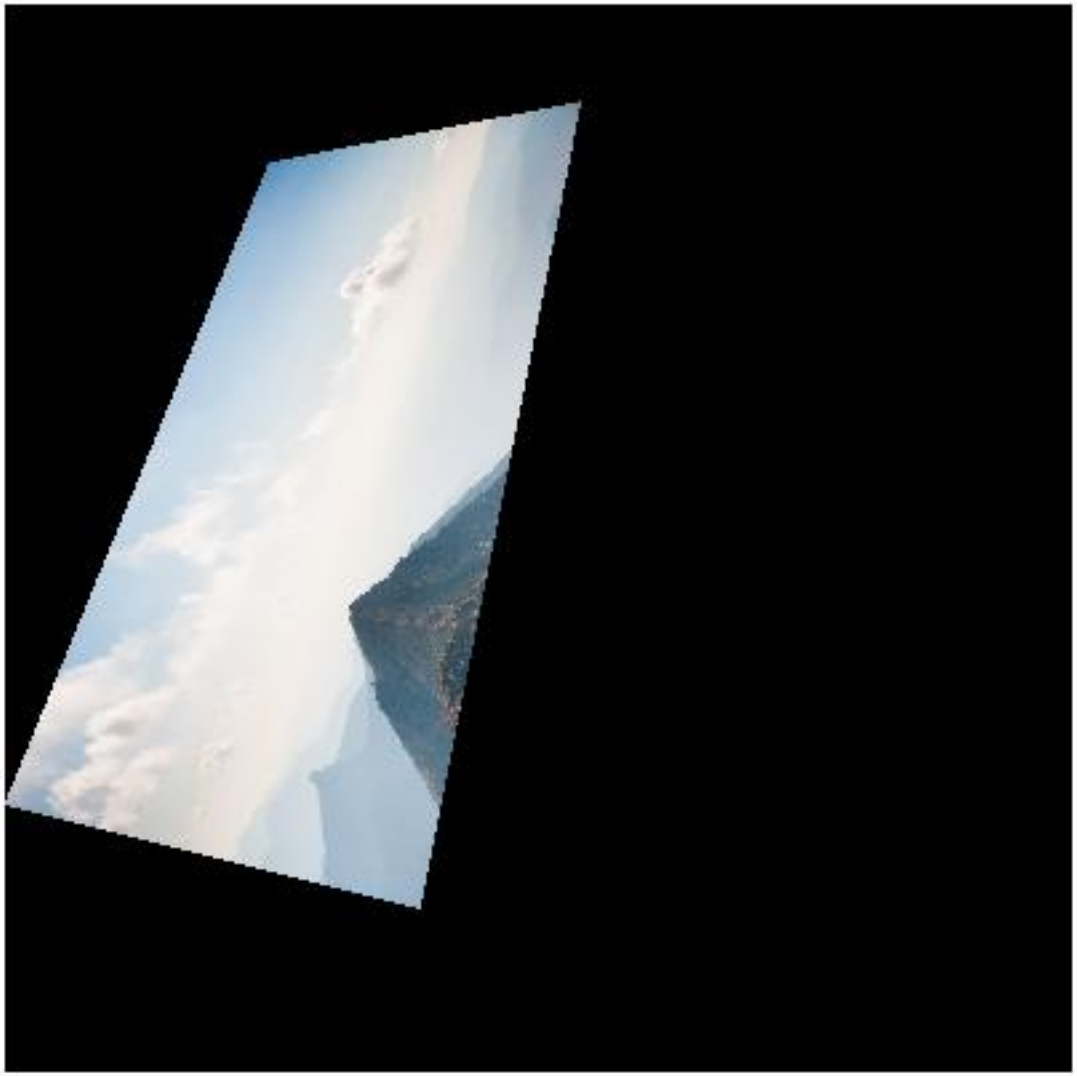
# `projective_transform`: Example



# `projective_transform`: Example



# `projective_transform`: Example

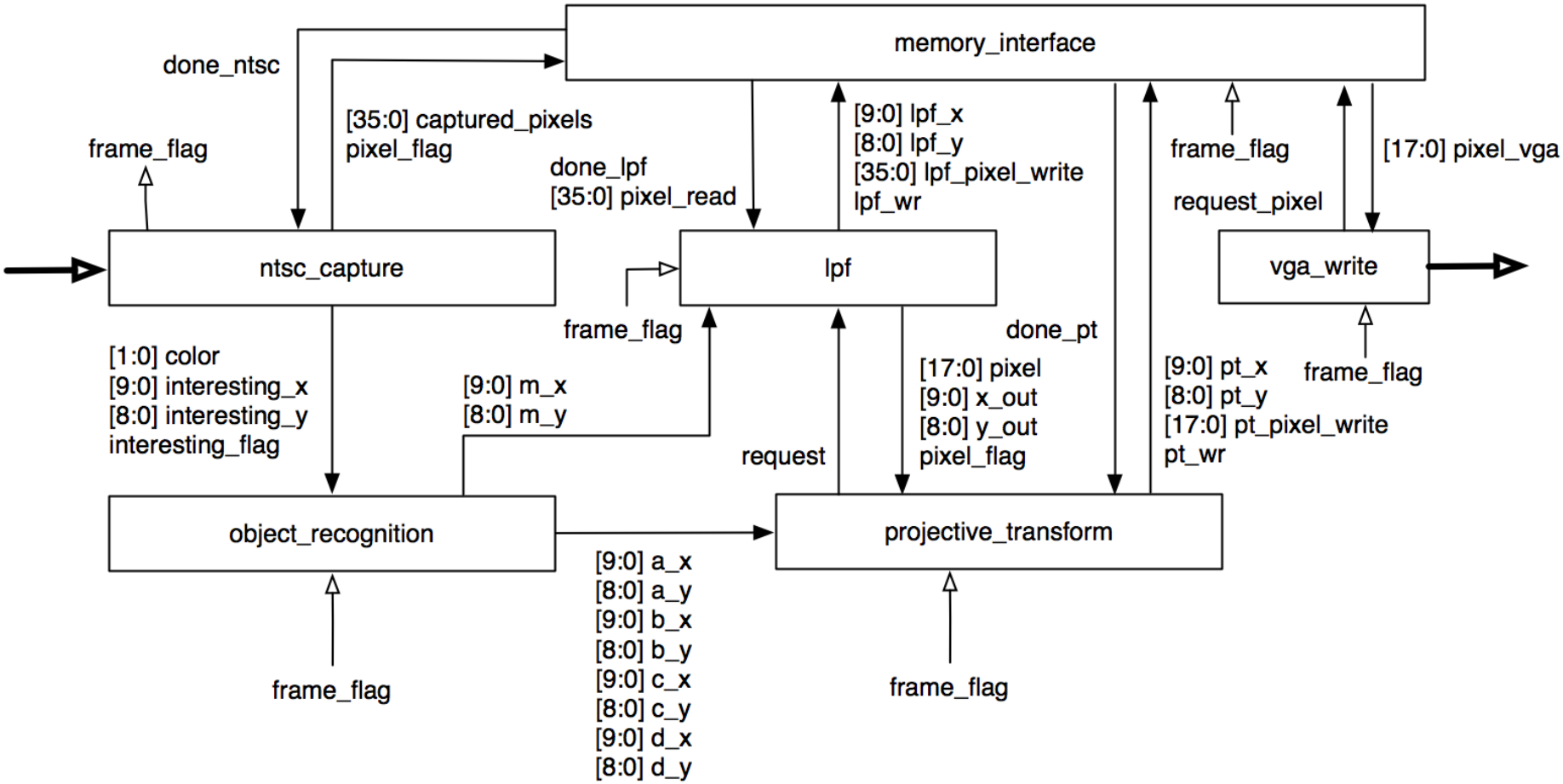


# `projective_transform`: Example

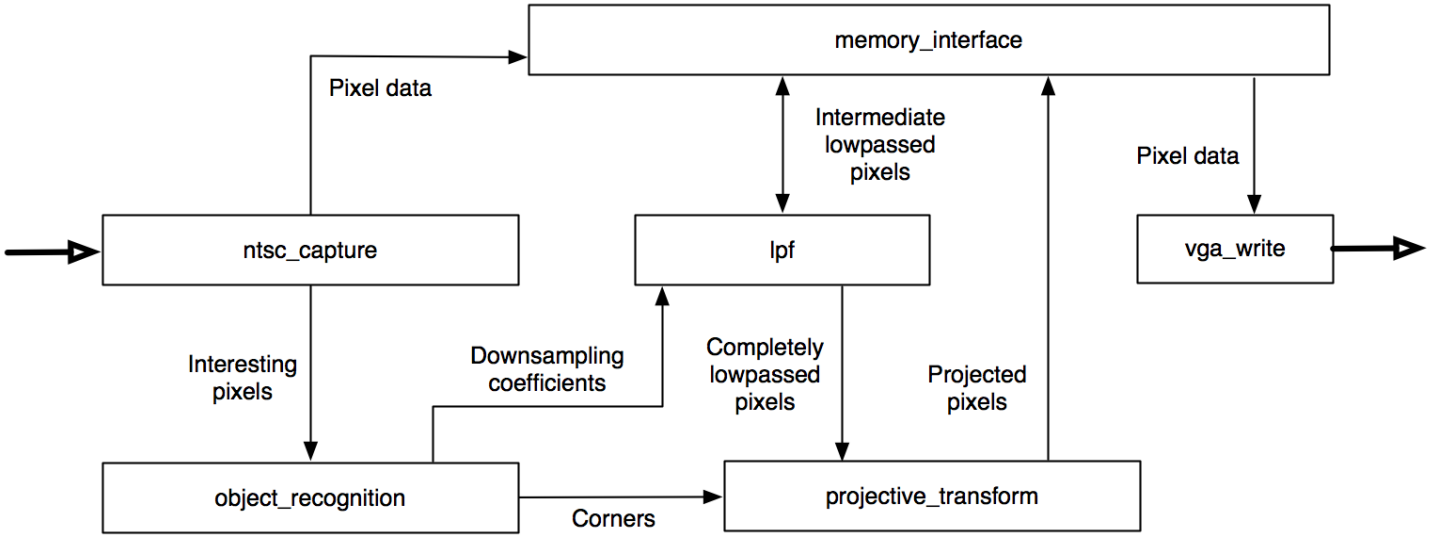
# `projective_transform`: FPGA implementation

- Straightfoward implementation of the above algorithm
- Uses coregen Divider modules for the divisions
- Requires only 2*640*480 + 4*480 multiplications per clock cycle
- Uses an iterative algorithm for finding distances (pipelined at the end of each line of the image)
- Processes pixels “on-the-fly” from `LPF`
- Negligible memory requirements (a handful of registers)

# projective_transform: How it Interfaces



# object_recognition




- Mark corners of frame with four differently colored dots.
- Recognition begins in the `ntsc_capture` module, which detects these colors as it is capturing data and sends the pixel info to the `object_recognition` module.

# object_recognition

- Take linear weighted center of mass for each image
- Sums the (x,y) coordinates for each color as it receives them. (8 running sums, 2 for each color)
- When the frame is done, divide each sum by the number of summed items
- The resulting 4 (x,y) pairs are the corners of the frame
- By looking for pixels in `ntsc_capture` we significantly reduce the amount of time spent in `object_recognition`

# `LPF`: its purpose

- `projective_transform` → aliasing

graphic showing normal signal

# LPF: its purpose

- `projective_transform` → aliasing

graphic aliases

# LPF: its purpose

- `projective_transform` → aliasing
- Aliasing reduces the quality of an image

zoom in on aliased pixels

# `LPF`: its purpose

- `projective_transform` → aliasing
- Aliasing reduces the quality of an image
- Lowpass filtering prevents aliasing

picture depicting lowpass filter in 2D

# `LPF`: its purpose

- `projective_transform` → aliasing
- Aliasing reduces the quality of an image
- Lowpass filtering prevents aliasing
- Information of an image is mostly phase

picture of original picture

# `LPF`: its purpose

- `projective_transform` → aliasing
- Aliasing reduces the quality of an image
- Lowpass filtering prevents aliasing
- Information of an image is mostly phase

picture of other image

# LPF: its purpose

- projective_transform → aliasing
- Aliasing reduces the quality of an image
- Lowpass filtering prevents aliasing
- Information of an image is mostly phase

picture of original phase with other's magnitude

# `LPF`: its purpose

- `projective_transform` → aliasing
- Aliasing reduces the quality of an image
- Lowpass filtering prevents aliasing
- Information of an image is mostly phase
- Symmetric Type I FIR filter → 0 phase distortion

picture of original phase with other's magnitude

# `LPF`: its purpose

- `projective_transform` → aliasing
- Aliasing reduces the quality of an image
- Lowpass filtering prevents aliasing
- Information of an image is mostly phase
- Symmetric Type I FIR filter → 0 phase distortion
- Parks-McClellan: reasonable accuracy, symmetric, easily calculable

frequency response of Parks-McClellan filter

# `LPF`: its purpose

- `projective_transform` → aliasing
- Aliasing reduces the quality of an image
- Lowpass filtering prevents aliasing
- Information of an image is mostly phase
- Symmetric Type I FIR filter → 0 phase distortion
- Parks-McClellan: reasonable accuracy, symmetric, easily calculable
- FIR PM filter reduces mem. acceses to 1.5/pixel

# LPF: the algorithm

1. Given an arbitrary image & skewing coefficients $M_x$ & $M_y$.

graphic showing the interface between object_recognition and LPF
image
magnitude fourier plot of image

# LPF: the algorithm

1. Given an arbitrary image & skewing coefficients $M_x$ & $M_y$.
2. Fetch a filter with cutoff $\frac{\pi}{M_y}$.

magnitude plot of image
magnitude plot of filter with cutoff pi/2

# LPF: the algorithm

1. Given an arbitrary image & skewing coefficients $M_x$ & $M_y$.
2. Fetch a filter with cutoff $\frac{\pi}{M_y}$.
3. Filter each column and store in memory.

magnitude plot of image
magnitude plot of filter with cutoff pi/2
magnitude fourier plot of filtered

# LPF: the algorithm

1. Given an arbitrary image & skewing coefficients $M_x$ & $M_y$.
2. Fetch a filter with cutoff $\frac{\pi}{M_y}$.
3. Filter each column and store in memory.
4. Fetch a filter with cutoff $\frac{\pi}{M_x}$.

magnitude plot of filtered image
magnitude plot of filter with curoff pi/4

# LPF: the algorithm

1. Given an arbitrary image & skewing coefficients $M_x$ & $M_y$.
2. Fetch a filter with cutoff $\frac{\pi}{M_y}$.
3. Filter each column and store in memory.
4. Fetch a filter with cutoff $\frac{\pi}{M_x}$.
5. Filter each row and output to `projective_transform`.

magnitude plot of filtered image
mangitude plot of filter with cutoff pi/5
magnitude plot of output

# LPF: the algorithm

1. Given an arbitrary image & skewing coefficients $M_x$ & $M_y$.
2. Fetch a filter with cutoff $\frac{\pi}{M_y}$.
3. Filter each column and store in memory.
4. Fetch a filter with cutoff $\frac{\pi}{M_x}$.
5. Filter each row and output to `projective_transform`.
6. Repeat this process every refresh cycle.

magnitude plot of original
magnitude plot of filter
magnitude plot of output

# memory_interface

- 1 image is a lot of data:
  $640 \cdot 480 \cdot 24$ bits $\approx 0.88$MiB

# memory_interface

- 1 image is a lot of data:
  $640 \cdot 480 \cdot 24$ bits $\approx 0.88$MiB
- Total BRAM on board: 0.316MiB

# memory_interface

- 1 image is a lot of data:
  $640 \cdot 480 \cdot 24$ bits $\approx$ 0.88MiB
- Total BRAM on board: 0.316MiB
- We need to store 4 images in memory!

# memory_interface

- 1 image is a lot of data:
  $640 \cdot 480 \cdot 24$ bits $\approx$ 0.88MiB
- Total BRAM on board: 0.316MiB
- We need to store 4 images in memory!
- Let's use the ZBT RAM: $2 \cdot$ 2.25MiB

# memory_interface

- 1 image is a lot of data:
  $640 \cdot 480 \cdot 24$ bits $\approx$ 0.88MiB
- Total BRAM on board: 0.316MiB
- We need to store 4 images in memory!
- Let's use the ZBT RAM: $2 \cdot$ 2.25MiB
- Not so fast: 1 clock cycle per memory access

# memory_interface

- 1 image is a lot of data:
  $640 \cdot 480 \cdot 24$ bits $\approx 0.88$MiB
- Total BRAM on board: 0.316MiB
- We need to store 4 images in memory!
- Let's use the ZBT RAM: $2 \cdot 2.25$MiB
- Not so fast: 1 clock cycle per memory access
- 1 pixel per address would require a clock speed $> 100MHz$

# memory_interface

- 1 image is a lot of data:
  $640 \cdot 480 \cdot 24$ bits $\approx 0.88$MiB
- Total BRAM on board: 0.316MiB
- We need to store 4 images in memory!
- Let's use the ZBT RAM: $2 \cdot 2.25$MiB
- Not so fast: 1 clock cycle per memory access
- 1 pixel per address would require a clock speed $> 100MHz$
- Let's store store 18 bits per pixel or 2 per address

# memory_interface: operation

1.

# system io: `ntsc_capture`

# system io: vga_write

# timeline