

Augmented Reality on FPGA

Realtime Object Recognition and Image Processing

Logan Williams José E. Cruz Serrallés

15 November 2011

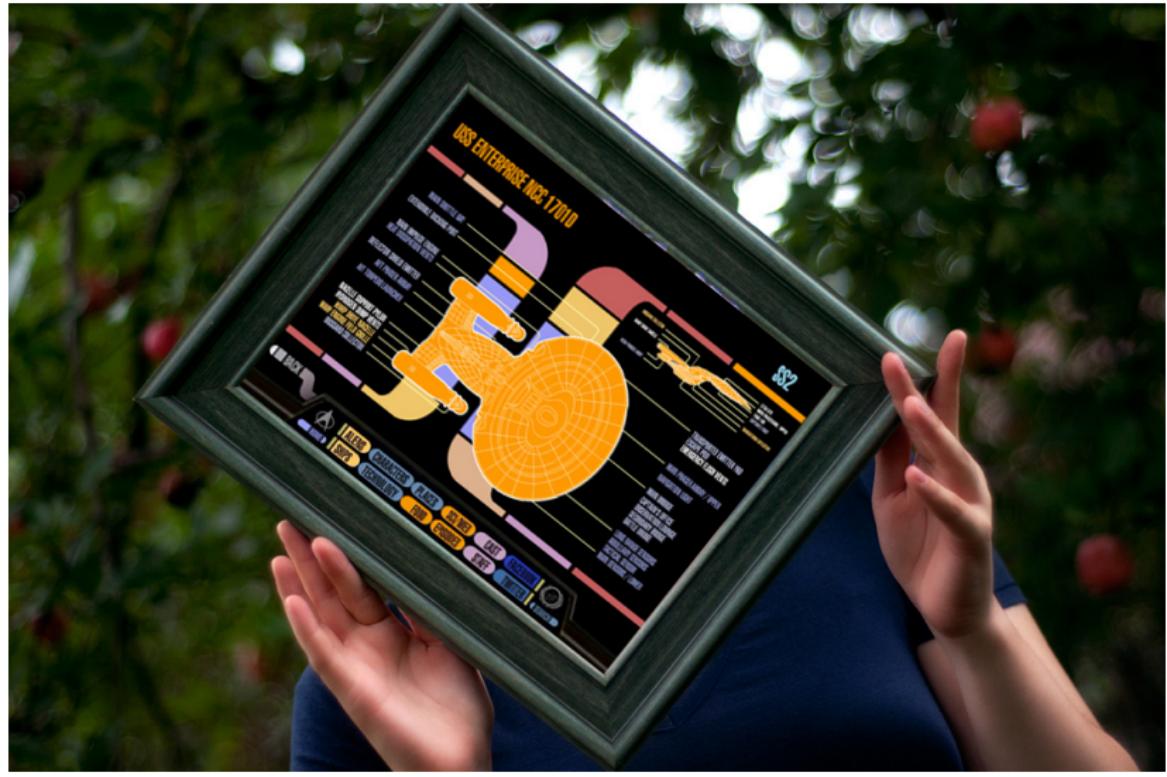
Introduction

- 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.

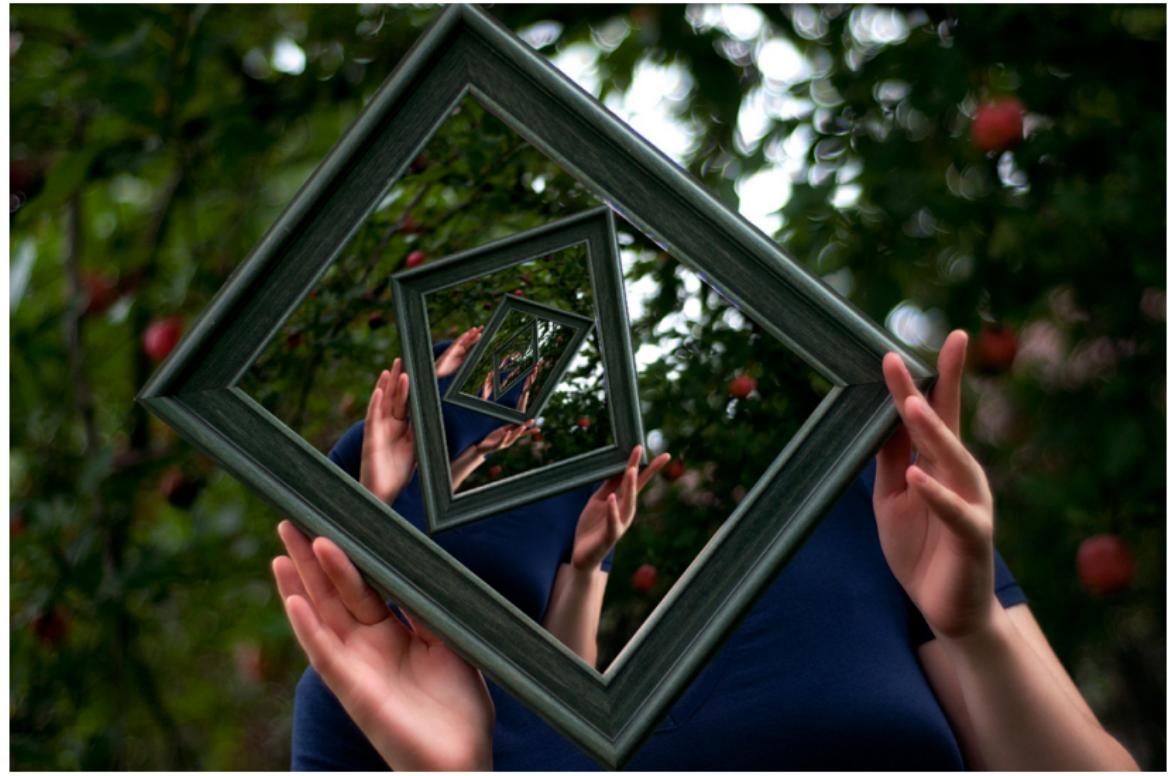
Example Image



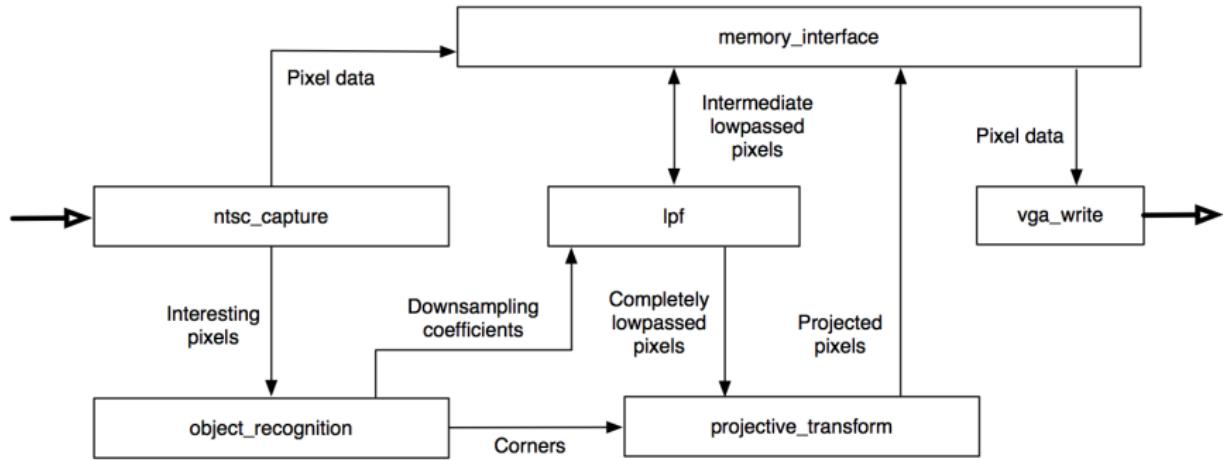
Example Image



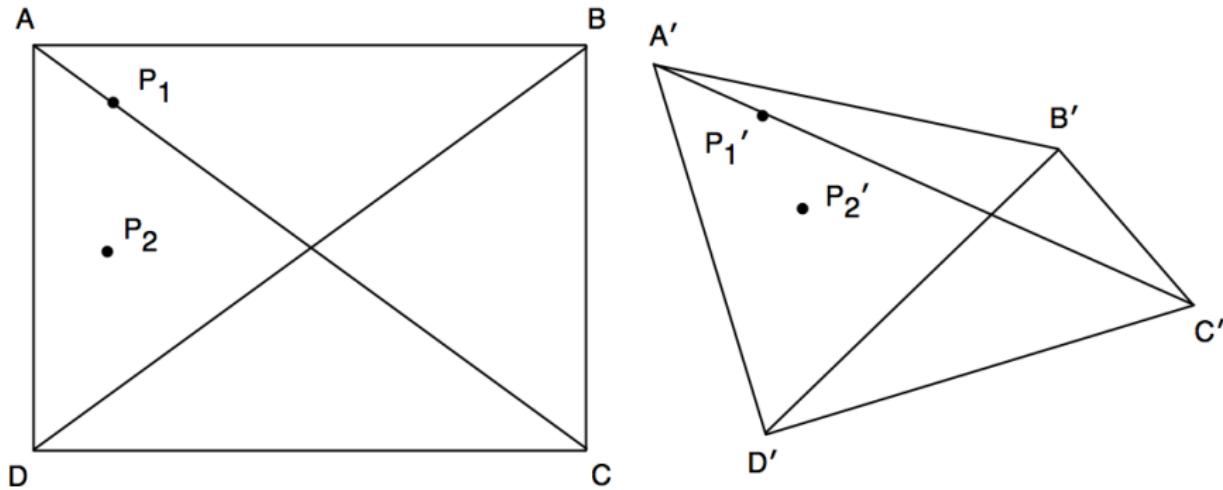
Example Image



Top-Level Overview

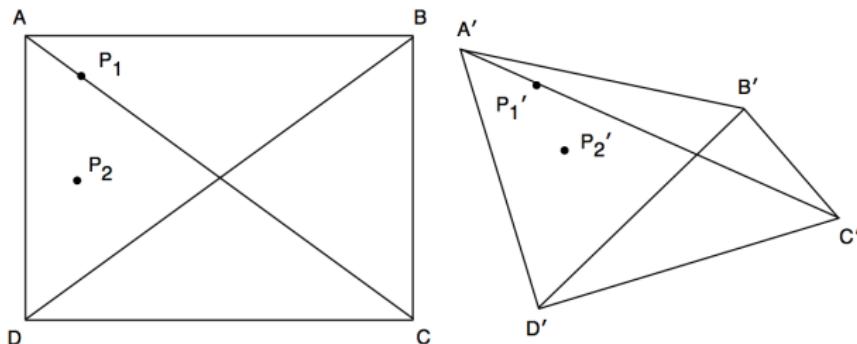


projective_transform: Purpose



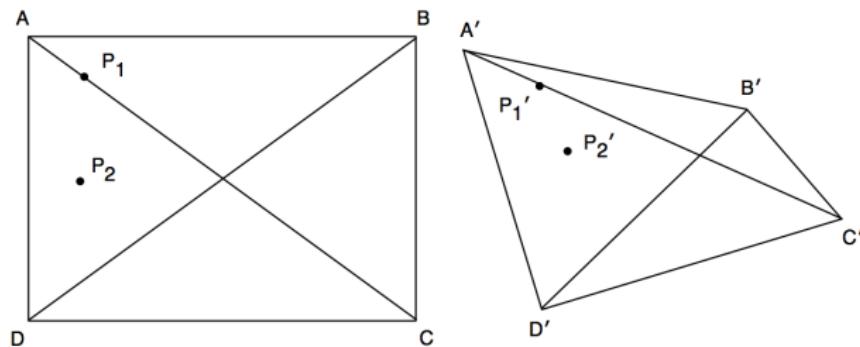
- Skew to any arbitrary convex quadrilateral

projective_transform: How the algorithm works



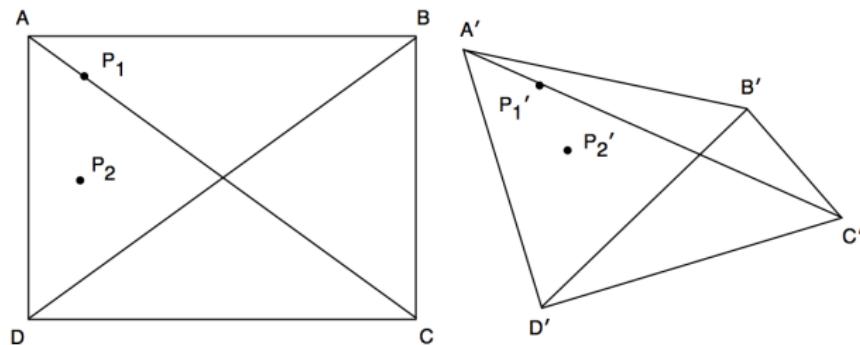
- 1 Calculate the distance of line $\overline{A'D'}$ and assign it to d_{ad} .
- 2 Do the same for $\overline{B'C'}$ and assign it to d_{bc} .
- 3 Create two “iterator points,” point I_A and I_B initially located at A' and B' .
- 4 Let $o_x = 0$ and $o_y = 0$
- 5 Calculate the distance between the iterator 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 $= \frac{d_i}{\text{width}_{\text{original}}}$.
- 9 Increment o_x .
- 10 Repeat steps 7–9 until $I_C = I_B$.

projective_transform: How the algorithm works



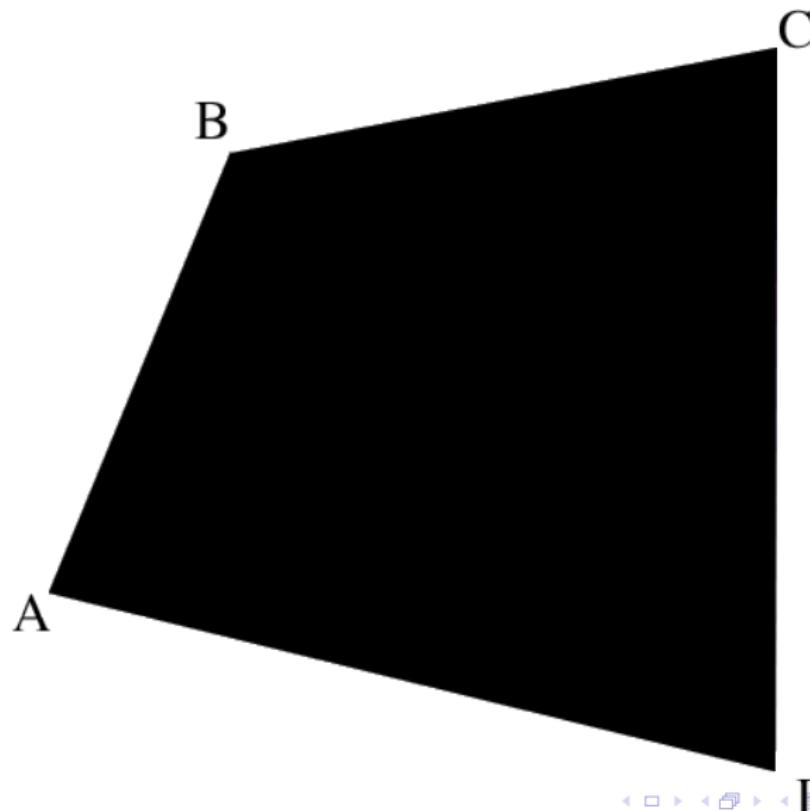
- 11 Move I_A along line $\overline{A'D'}$ by an amount $= \frac{d_{ad}}{\text{height}_{\text{original}}}$.
- 12 Move I_B along line $\overline{B'C'}$ by an amount $= \frac{d_{bc}}{\text{height}_{\text{original}}}$.
- 13 Increment o_y .
- 14 Repeat steps 5–13 until $I_A = D'$ and $I_B = C'$.

projective_transform: Example

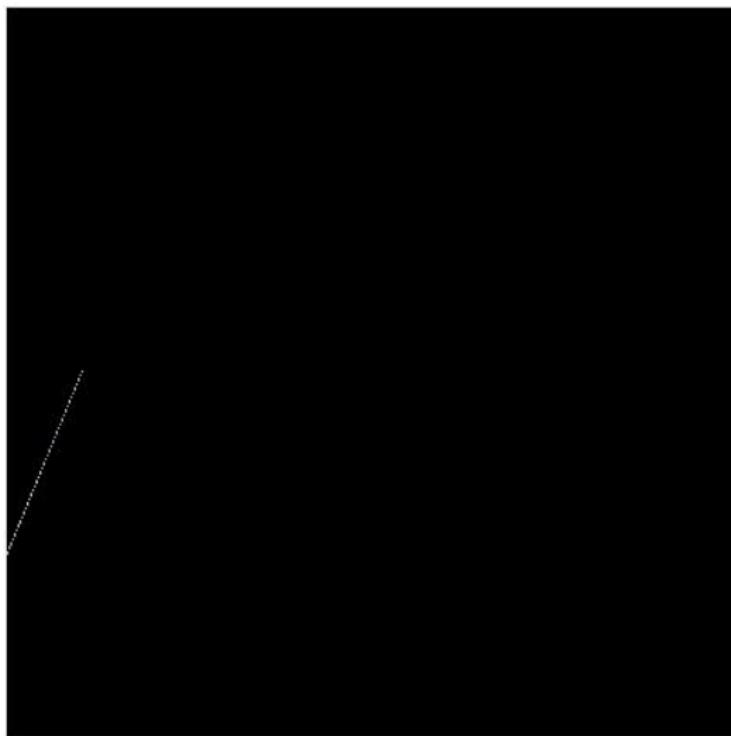


Figure: The original image

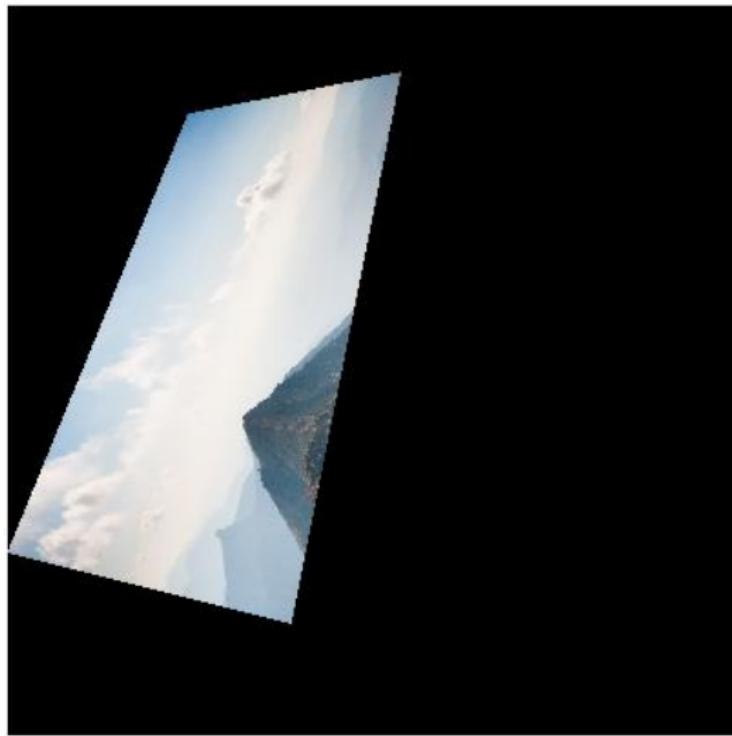
projective_transform: Example



projective_transform: Example



projective_transform: Example



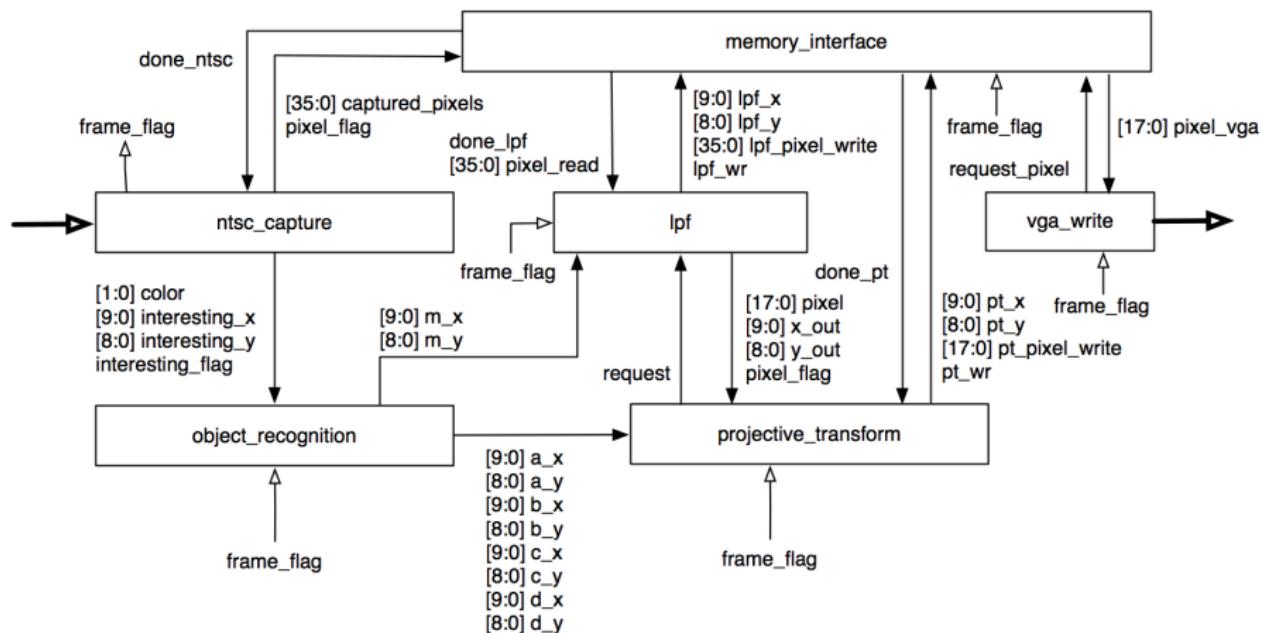
projective_transform: Example



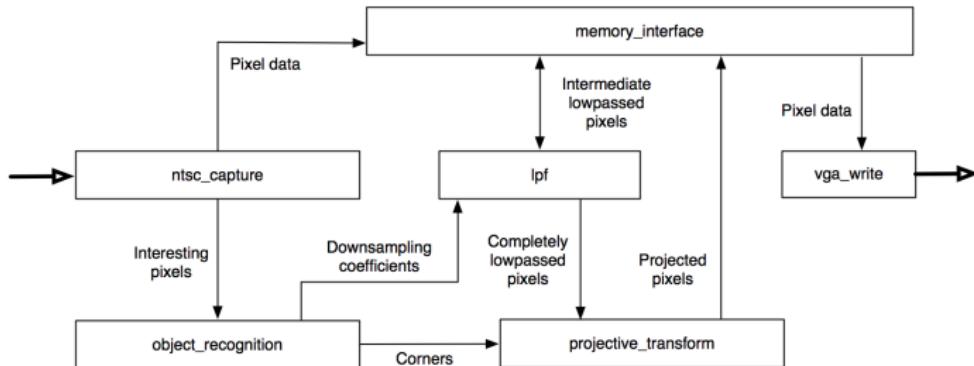
projective_transform: FPGA implementation

- Straightforward 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



Original

LPF: its purpose

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

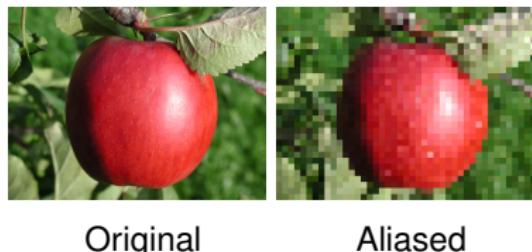


Original

Aliased

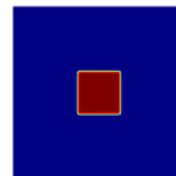
LPF: its purpose

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



Original

Aliased



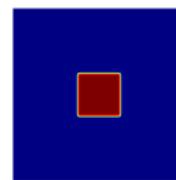
Mag. of Filter

LPF: its purpose

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



Original



Mag. of Filter



Filtered

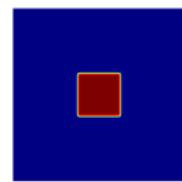
LPF: its purpose

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



Original

Aliased



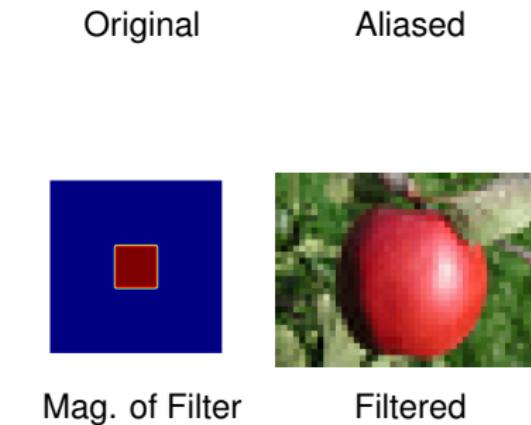
Mag. of Filter



Filtered

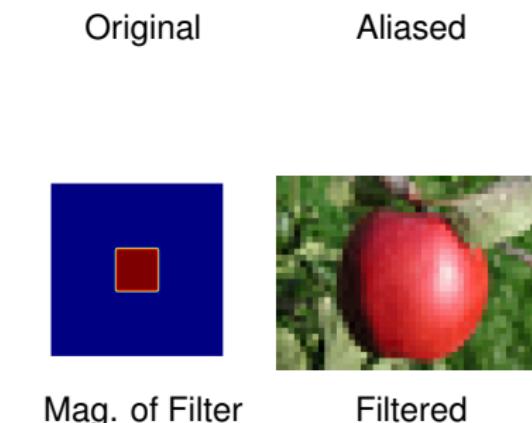
LPF: its purpose

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



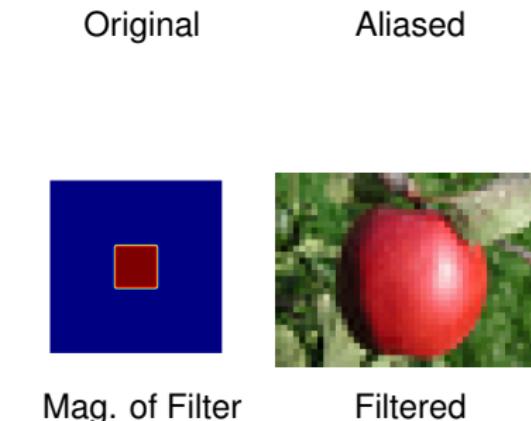
LPF: its purpose

- `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



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



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



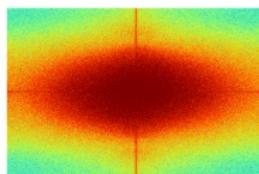
Original Aliased



Mag. of Filter Filtered

LPF: the algorithm

- Given an arbitrary image & skewing coefficients M_x & M_y .



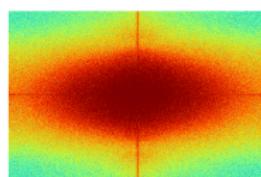
F.T. Original



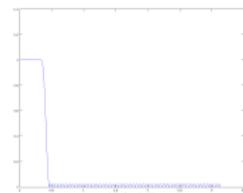
Original Image

LPF: the algorithm

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



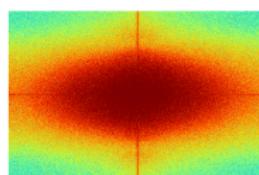
F.T. Original



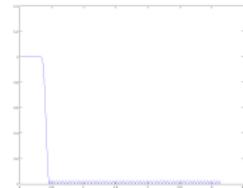
1D FIR, $\omega_c = \frac{\pi}{8}$

LPF: the algorithm

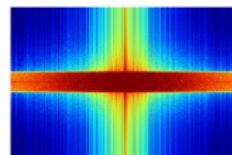
- ① Given an arbitrary image & skewing coefficients M_x & M_y .
- ② Fetch a filter with cutoff $\frac{\pi}{M_y}$.
- ③ Filter each column and store in memory.



F.T. Original



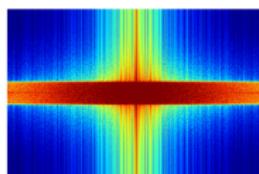
1D FIR, $\omega_c = \frac{\pi}{8}$



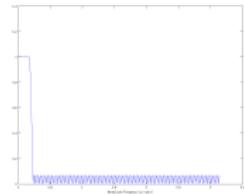
F.T. Filtered

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 $\frac{\pi}{M_x}$.



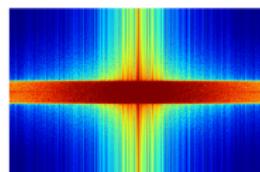
F.T. Filtered



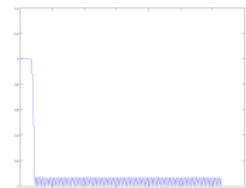
1D FIR, $\omega_c = \frac{\pi}{16}$

LPF: the algorithm

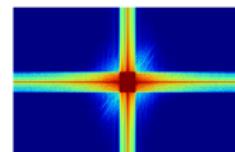
- ① Given an arbitrary image & skewing coefficients M_x & M_y .
- ② Fetch a filter with cutoff $\frac{\pi}{M_y}$.
- ③ Filter each column and store in memory.
- ④ Fetch a filter with cutoff $\frac{\pi}{M_x}$.
- ⑤ Filter each row and output to projective_transform.



F.T. Filtered



1D FIR, $\omega_c = \frac{\pi}{16}$



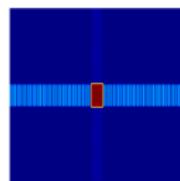
F.T. Output

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.



Original



F.T. of Process



Output

memory_interface

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

memory_interface

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

memory_interface

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

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

memory_interface

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

memory_interface

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

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

memory_interface: operation

- During a cycle

memory_interface: operation

- During a cycle
 - Store pixels from ntsc_capture in *capturing*

memory_interface: operation

- During a cycle
 - Store pixels from `ntsc_capture` in *capturing*
 - Output pixels from *display* to `vga_display`

memory_interface: operation

- During a cycle
 - Store pixels from `ntsc_capture` in *capturing*
 - Output pixels from *display* to `vga_display`
 - Read and write pixels to *processing* from `LPF`

memory_interface: operation

- During a cycle
 - Store pixels from `ntsc_capture` in *capturing*
 - Output pixels from *display* to `vga_display`
 - Read and write pixels to *processing* from `LPF`
 - Store pixels from `projective_transform` in *next_display*

memory_interface: operation

- During a cycle
 - Store pixels from `ntsc_capture` in *capturing*
 - Output pixels from `display` to `vga_display`
 - Read and write pixels to *processing* from `LPF`
 - Store pixels from `projective_transform` in *next_display*
- Every cycle refresh

memory_interface: operation

- During a cycle
 - Store pixels from `ntsc_capture` in *capturing*
 - Output pixels from *display* to `vga_display`
 - Read and write pixels to *processing* from `LPF`
 - Store pixels from `projective_transform` in *next_display*
- Every cycle refresh
 - Load image data to *processing* (could be previous *displaying*)

memory_interface: operation

- During a cycle
 - Store pixels from `ntsc_capture` in *capturing*
 - Output pixels from *display* to `vga_display`
 - Read and write pixels to *processing* from `LPF`
 - Store pixels from `projective_transform` in *next_display*
- Every cycle refresh
 - Load image data to *processing* (could be previous *displaying*)
 - Previous *next_display* location → Next *display* location

memory_interface: operation

- During a cycle
 - Store pixels from `ntsc_capture` in *capturing*
 - Output pixels from *display* to `vga_display`
 - Read and write pixels to *processing* from `LPF`
 - Store pixels from `projective_transform` in *next_display*
- Every cycle refresh
 - Load image data to *processing* (could be previous *displaying*)
 - Previous *next_display* location → Next *display* location
 - Previous *ntsc_capture* location → Next *next_display* location

memory_interface: operation

- During a cycle
 - Store pixels from `ntsc_capture` in *capturing*
 - Output pixels from *display* to `vga_display`
 - Read and write pixels to *processing* from `LPF`
 - Store pixels from `projective_transform` in *next_display*
- Every cycle refresh
 - Load image data to *processing* (could be previous *displaying*)
 - Previous *next_display* location → Next *display* location
 - Previous `ntsc_capture` location → Next *next_display* location

system io: ntsc_capture

system io: vga_write

timeline