Realtime plane-wave software beamforming with an iPhone

Cameron Lowell Palmer

NTNU ISB

Ole Marius Hoel Rindal

Sverre Holm

Andreas Austeng

Department of Informatics

University of Oslo

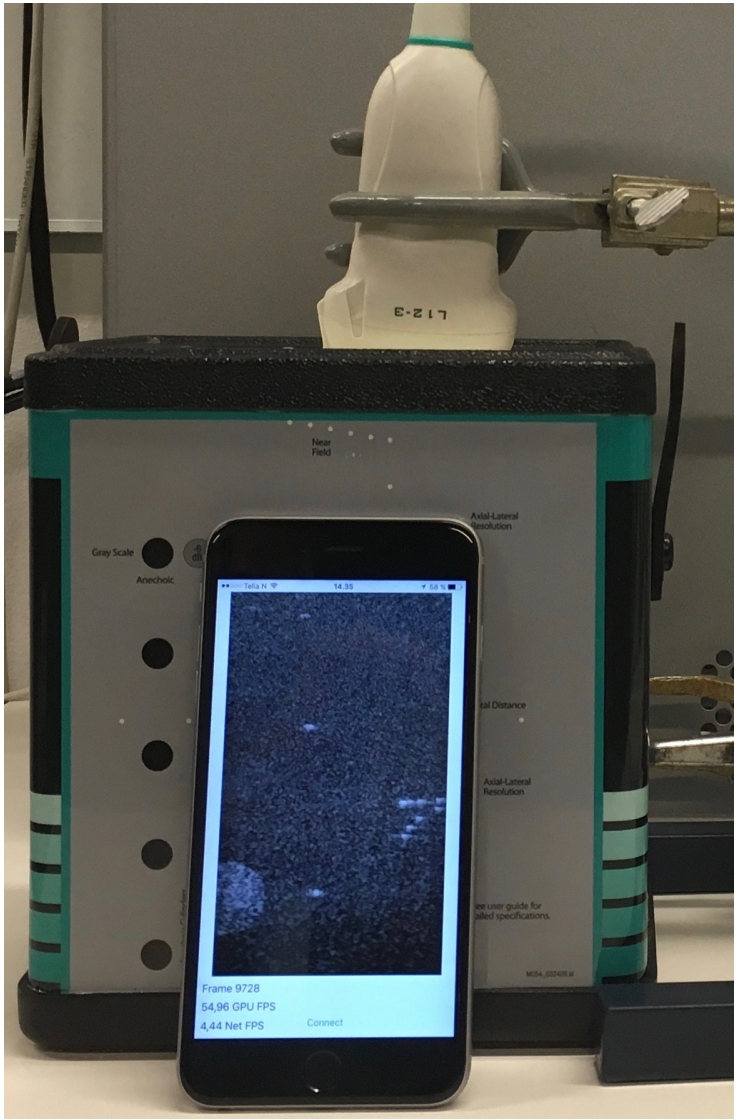
Norwegian University of Science and Technology

Objective

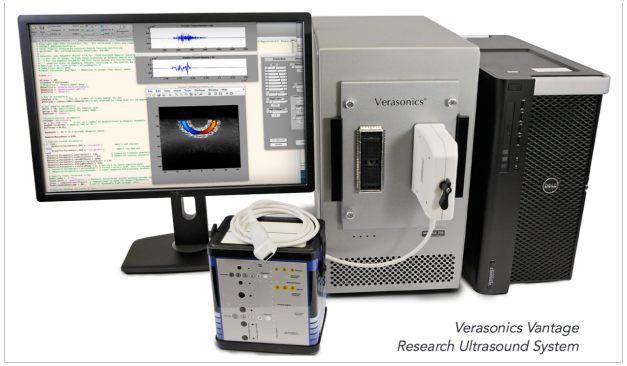
- Investigate the feasibility for an iPhone to beamform plane-wave images with the channel data wirelessly transmitted to the device.

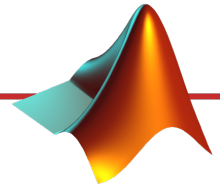
Why is this interesting?

- Proof of concept that the probe in a handheld system only needs hardware to acquire and transmit the channel data, while the beamforming can be done in software on commercially available devices.



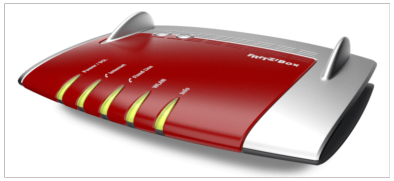
Functional diagram

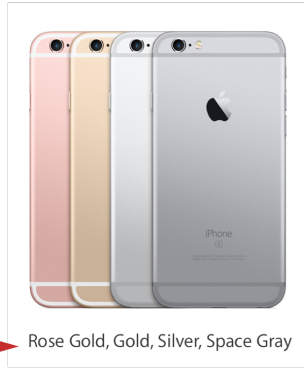




python™

Channel Data





Not to scale

Recording and transmitting the channel data

Recording data on the Verasonics Vantage Scanner

- L12-3v 192-element linear array.
  - Only using 128 elements
  - Transmitting one plane wave 7.8 MHz recording a 45.8 mm deep image.
  - The data is IQ sampled.
  - Total of 128 x 512 IQ samples

Transmitting the data

- The data is JSON-serialized and POST'd with webwrite()
- Python script takes POST and broadcasts to all connected WebSockets

A simple serialization format

```
{
    "identifier"   : x,
    "channel_data" : [[channel_1_samples],
                      [channel_2_samples],
                      [...],
                      [channel_n_samples]]
}
```

Uncompressed: ~400 kB, Compressed: ~40kB

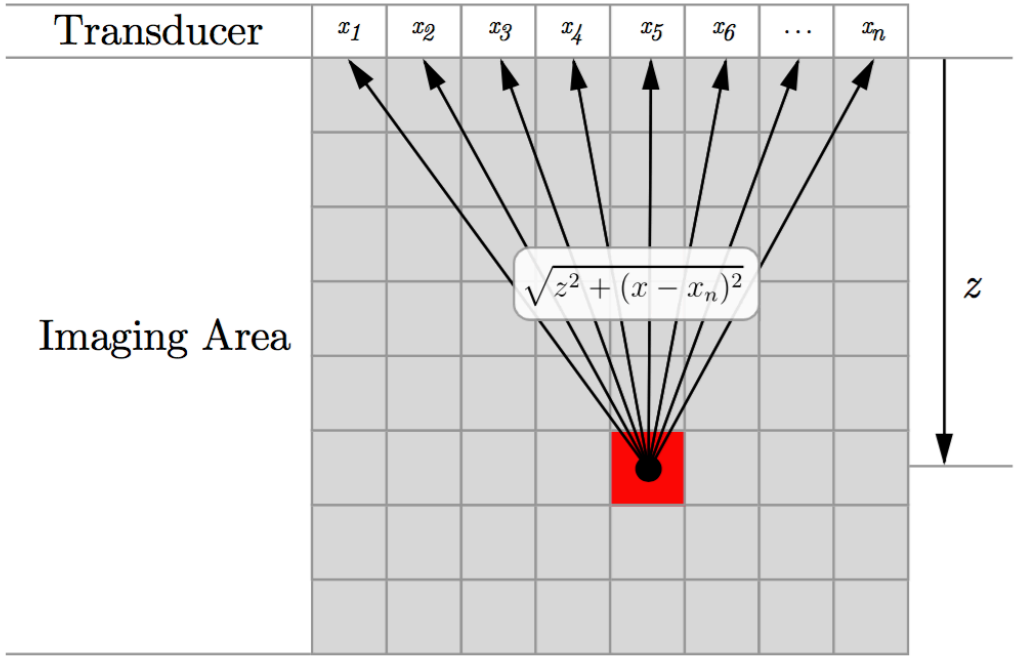
Receiving and beamforming the data on the iPhone - Overview

1. Data arrives on WebSocket
2. Decompression if applicable
3. Deserialization to a first class object
4. Perform delay calcuations (one-time)
5. malloc (one-time) region of memory to hold channel data
6. memcpy channel data to memory
7. Perform channel data computations
8. Convert results into a UIImage
9. Display resulting image

Pixel-based beamforming

1. Calculation of delays

$$\tau(z, x, x_n) = (z + \sqrt{z^2 + (x - x_n)^2})/c.$$



Pixel-based beamforming (PBB)

2. Delay-and-sum calculation

$$I_{DAS}[z,x] = \sum_{m=0}^{M-1} w_m y_m[z,x]$$

3. Take decibel value of envelope

$$I_{\mathrm{dB}} = 20 \cdot \log_{10}(|I_{\mathrm{DAS}}|)$$

How is PBB implemented?

- Swift, Objective-C, and Metal
- Roughly two steps
  1. Pre-calculate and store the delays for every pixel in the defined pixel grid in memory.
  2. Channel data processing
     - One thread per pixel
     - Each thread then delays by linear interpolation and sums the contribution from each channel recorded
     - The amplitude is detected, and the log compression is done

Results

The iPhone can beamform a plane wave image

- GPU ~55 FPS when the channel data is stored on the phone
- Network ~5 FPS when the channel data is wirelessly transmitted

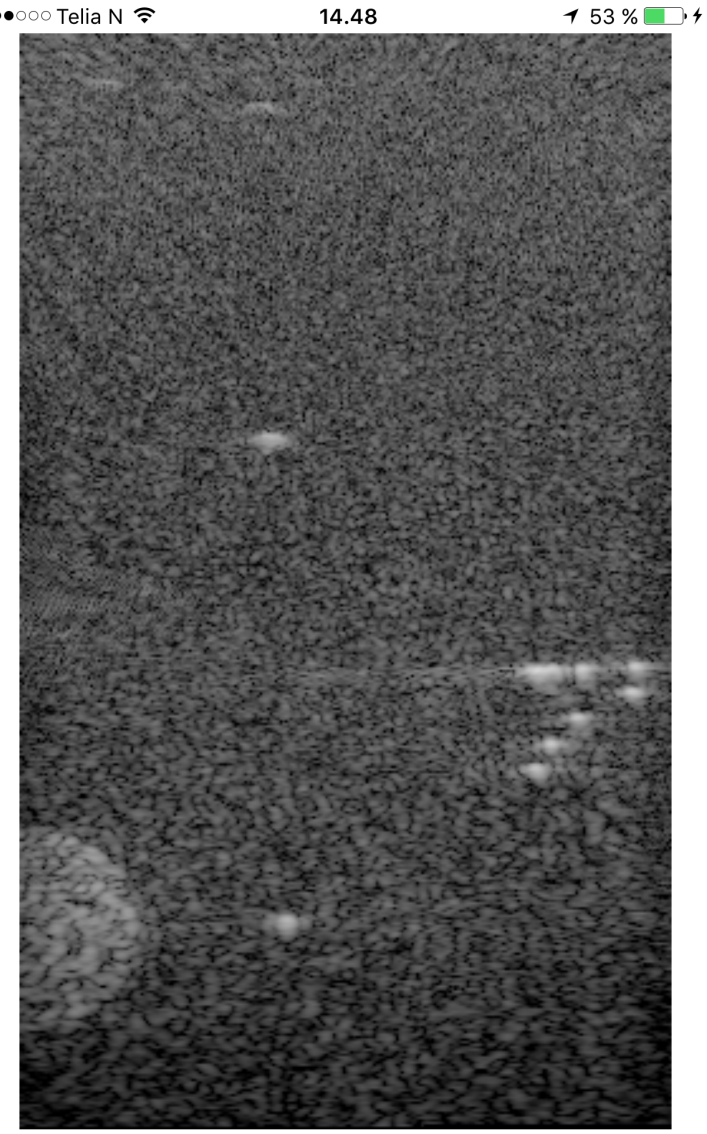
	Net FPS	GPU FPS
129x217px λ	4.5 (σ=0.5)	142.0 (σ=5.9)
258x434px $\lambda/2$	4.5 (σ=0.2)	55.5 (σ=0.7)
375x632px $\lambda/3$	4.9 (σ=3.3)	29.4 (σ=1.1)

Results

Image quality

As expected from single plane wave image

- Visible side-lobes
- Fixed aperture leads to a noisy near field



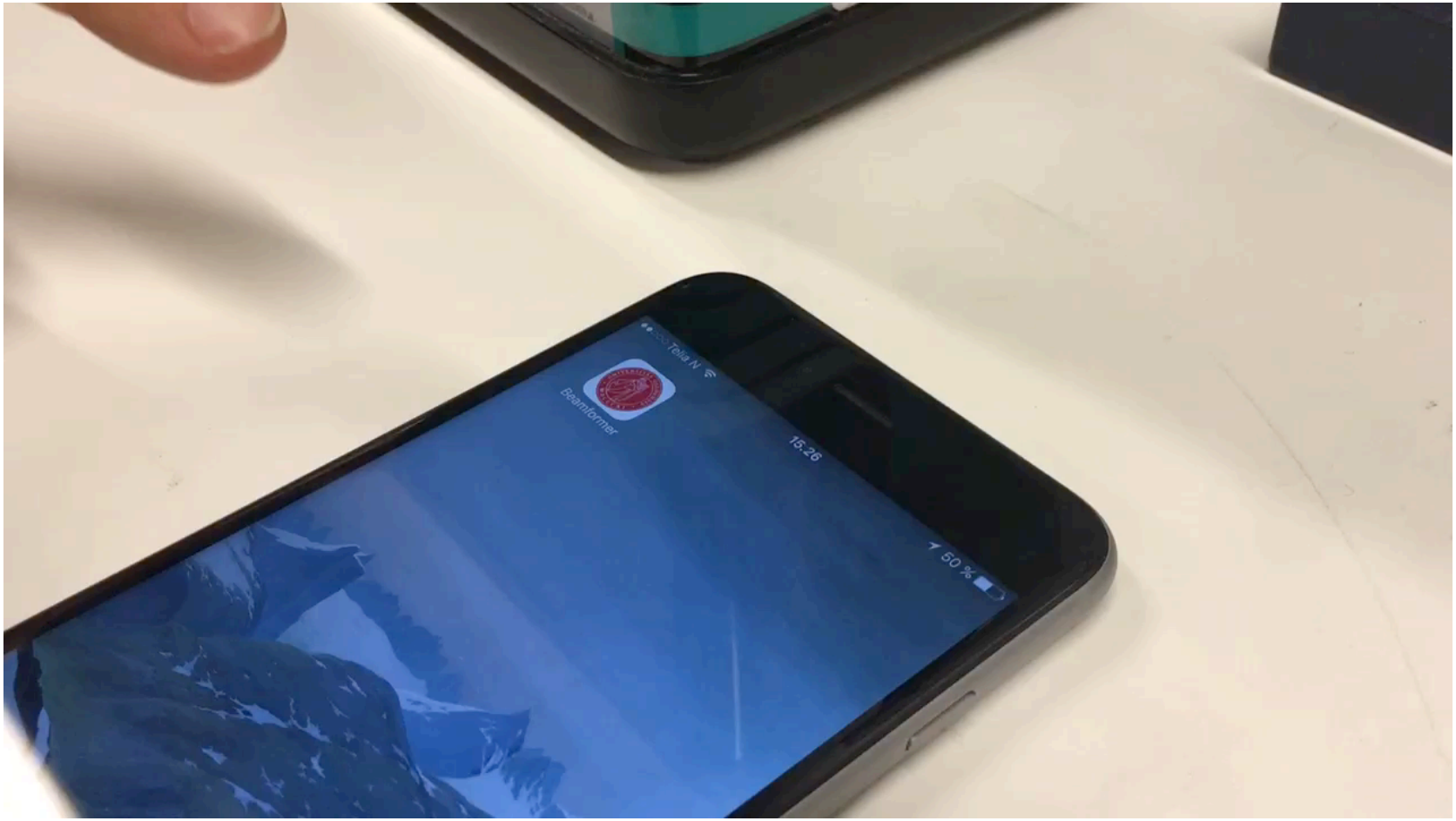
Frame 13398

55,51 GPU FPS

4,43 Net FPS

Connect

Demonstration



Discussion

- The bottleneck is the serialization
  - We are got to switch to a binary protocol
    - Apache Thrift
    - Google Protocol Buffers
- We are currently working with the implementation of compounded plane wave imaging to improve the image quality
  - Adding dynamic aperture
  - Multiple angle compounding

Conclusions

- We have shown that realtime plane-wave ultrasound imaging with an iPhone is feasible when processing wirelessly streamed channel data using a Verasonics Vantage ultrasound scanner and iPhone 6s at ~5 FPS. The GPU processing frame rate is much higher measured at 55 FPS for a 25.5 mm wide and 45.5 mm deep image using $\lambda/2$ pixel spacing and can be viewed as the upper-bound of our current implementation.