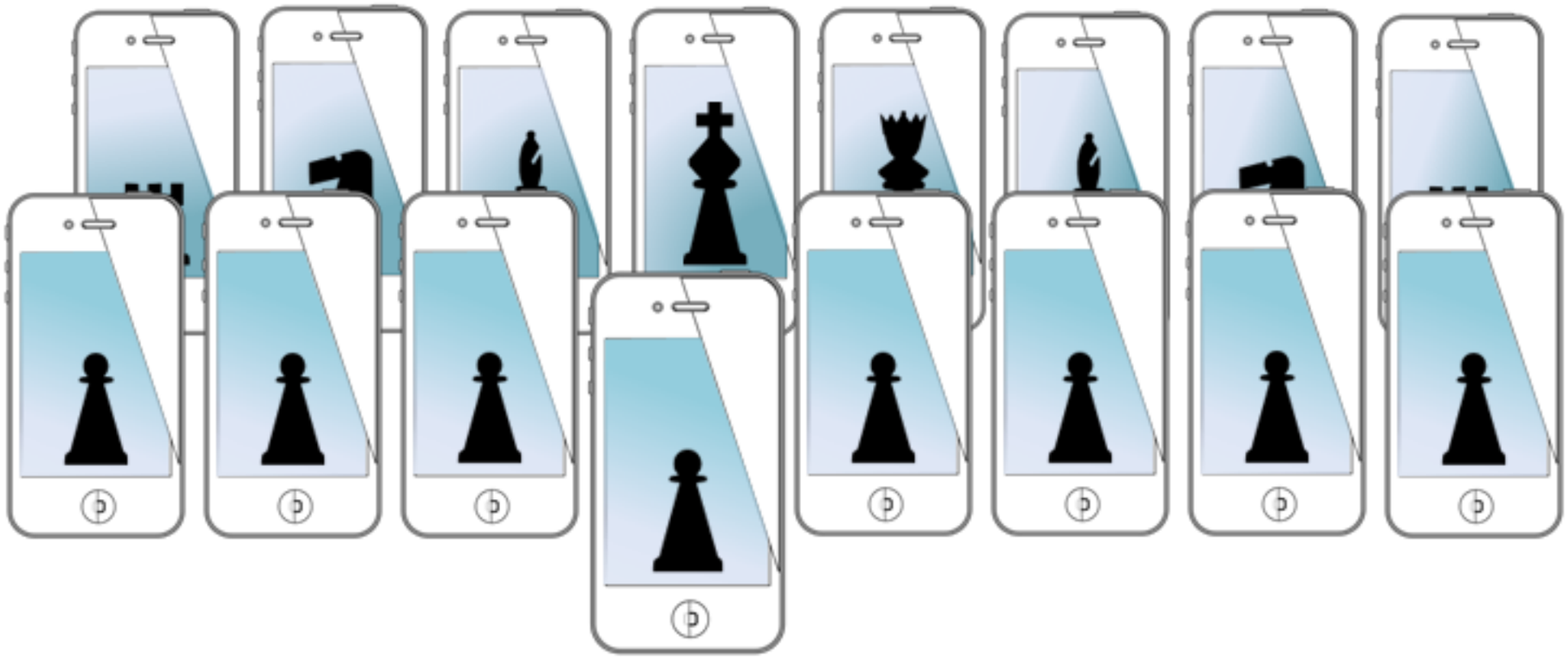
MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture six: audio graphing, sampled data, accelerate, & FFT

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University

course logistics

- A1 grades by Wednesday (I hope)
- A2 is up!

Module A
Create an iOS application using the NovocaineExample template that:

- Reads from the microphone
- Takes an FFT of the incoming audio stream
- Displays the frequency of the two loudest tones within 6Hz accuracy
- Is able to distinguish tones as least 25Hz apart, lasting for 100ms or more

The sound source must be external to the phone (i.e., laptop, instrument, another phone, etc.).

Module B
Create an iOS application using the NovocaineExample template that:

- Reads from the microphone
- Plays a settable (via a slider or setter control) inaudible tone to the speakers (15-20kHz)
- Displays the magnitude of the FFT of the microphone data in decibels
- Is able to distinguish when the user is {not gesturing, gestures toward, or gesturing away} from the microphone using Doppler shifts in the frequency

agenda

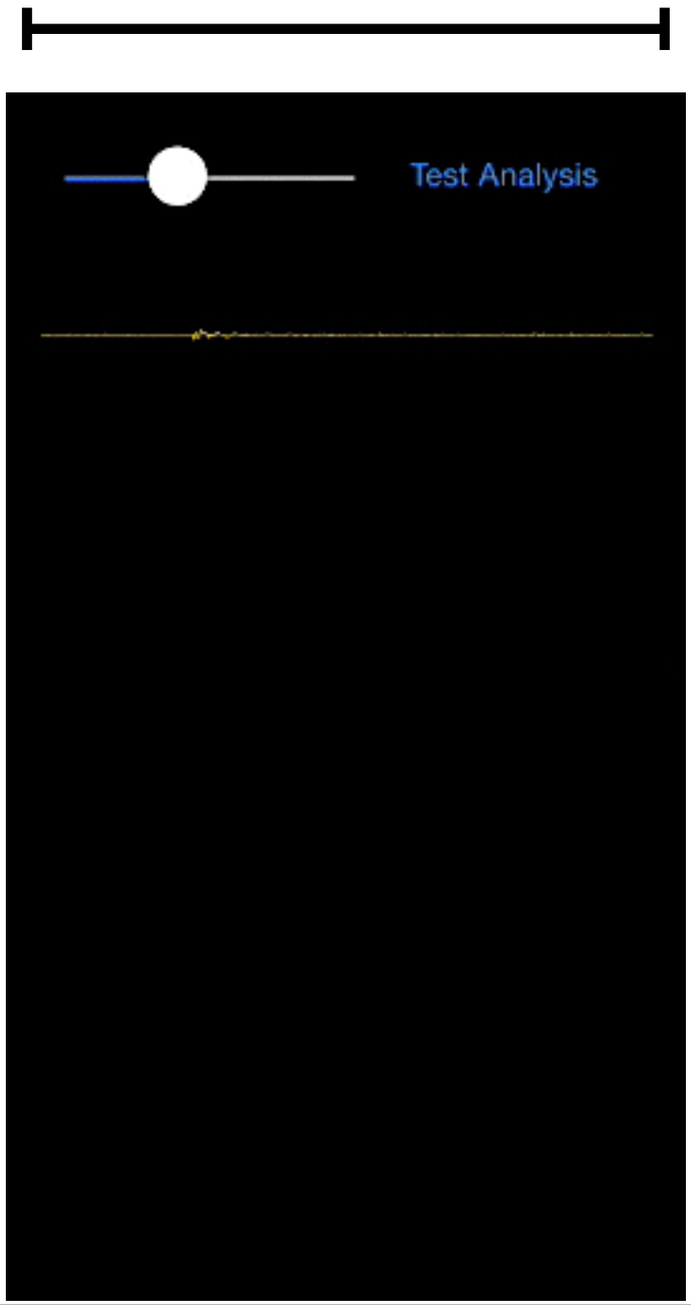
- graphing audio fast (well, graphing anything)
  - must use lowest level graphing, OpenGL
- dealing with sampled data
- the accelerate framework
  - massive digital signal processing library
- the Fourier Transform for spectral analysis

audio graphing

- we want to see the incoming samples
  - good for debugging
  - equalizers
  - oscilloscope type applications

how much data to show?

- sampling at 44.1kHz == 44100 samples per second



0.5 seconds is 22050 samples

display is 640 pixels wide

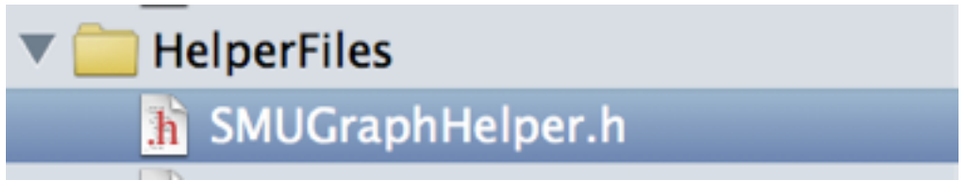
what if we want lots of graphs?

solution

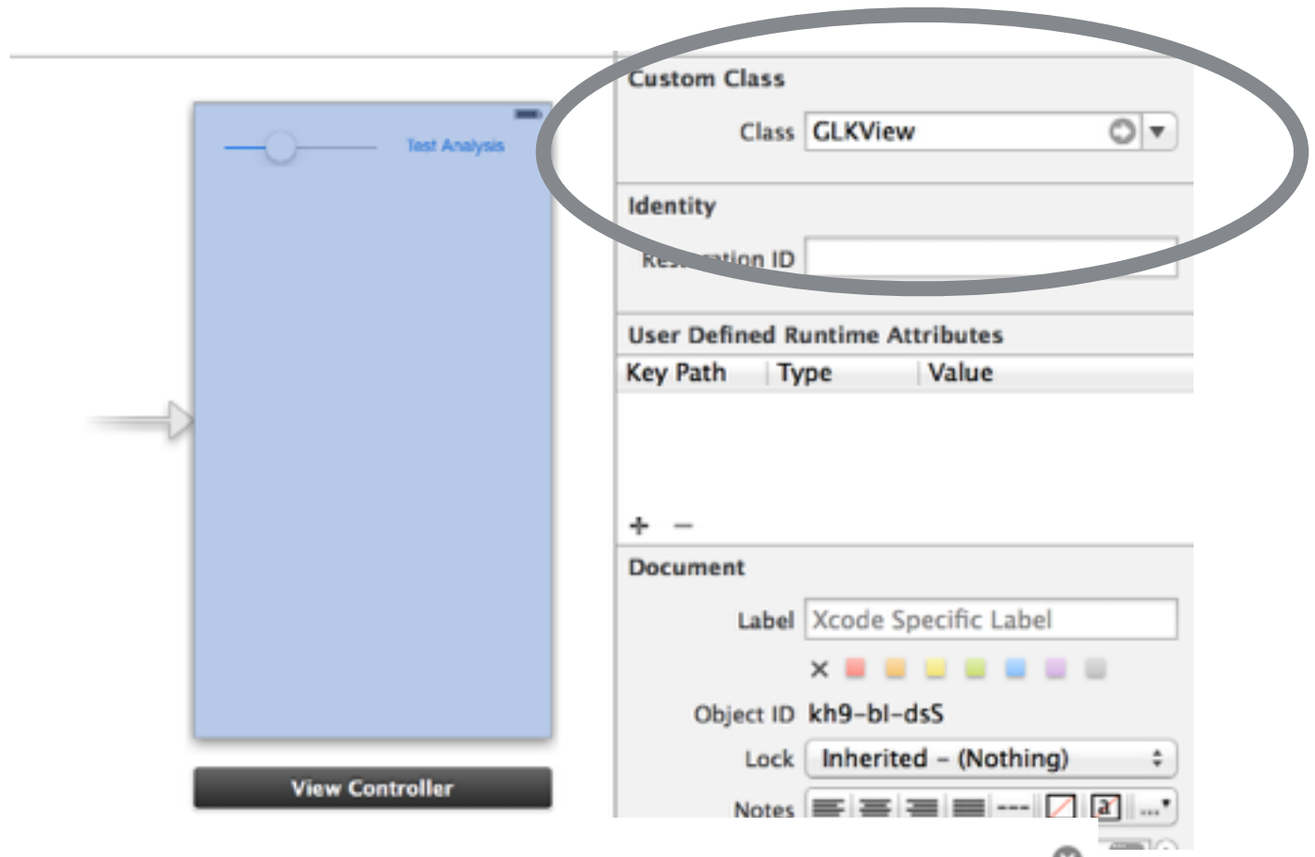
- use the GPU
- set vectors of data on a 2D plane
- let the renderer perform scaling, anti-aliasing, and bit blitting to screen
- …this is not a graphics course

easy solution

- use graph helper, which uses GLKView and GLKViewController



add this to project



add GLKit framework

the graph helper

```
#import <GLKit/GLKit.h>
@interface YourCustomViewController : GLKViewController

GraphHelper     *graphHelper;

In View Did Load
   // start animating the graph
    int framesPerSecond = 15;
    int numDataArraysToGraph = 3;
    graphHelper = new GraphHelper(self,
                                  framesPerSecond,
                                  numDataArraysToGraph,
                                  PlotStyleSeparated);//drawing starts immediately after call

    graphHelper->SetBounds(-0.9,0.9,-0.9,0.9); // bottom, top, left, right,
   //full screen==(-1,1,-1,1)


-(void)dealloc{
    graphHelper->tearDownGL();
    // ARC handles everything else, just clean up what we used c++ for (calloc, malloc, new)
}

                              enum PlotStyle {
                                  PlotStyleOverlaid,
                                  PlotStyleSeparated
                              };
```

declare in implementation

inherit from open GL

setup GLKViewController

bounds for screen

setting data

```
//  override the GLKViewController draw function, from OpenGLES
- (void)glkView:(GLKView *)view drawInRect:(CGRect)rect {
    graphHelper->draw(); // draw the graph
}
```

called for each draw to screen

```
void setGraphData(int graphNum, float *data, int dataLength, float normalization = 1.0,
                  float minValue = 0.0)
```

prototype for setting scatter data

```
//  override the GLKView update function, from OpenGLES
- (void)update{

    graphHelper->setGraphData(0,                        //channel index
                              inputAudioDataBuffer,     //data
                              kBufferLength/2.0,        //data length
                              64.0);                    // max value to normalize (==1 if not set)

    // just plot the audio stream
    graphHelper->setGraphData(1,inputAudioDataBuffer,kBufferLength); // set graph channel

    graphHelper->update(); // update the graph
}
```

set data for 0th graph

set data for 1st graph

update render state

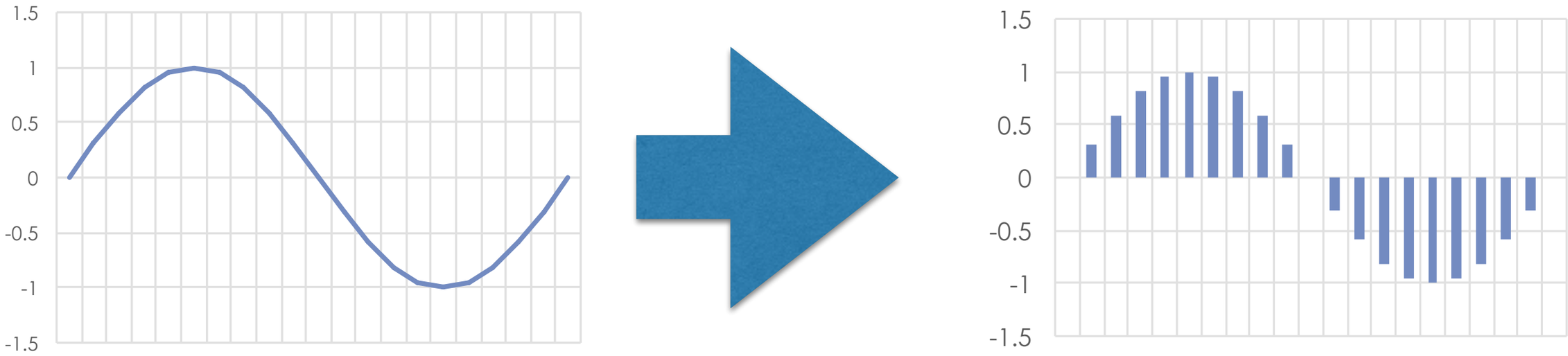
audio graphing demo!

intro to sampled data

- why is understanding sampled data important?
  - because we'll be dealing with it all semester
  - it's important to understand basic mistakes that can be made
- there are entire courses dedicated to sampled time series
  - actually entire courses on analyzing frequency content
- we'll touch on a few guidelines to help you design your projects better

intro to sampled data

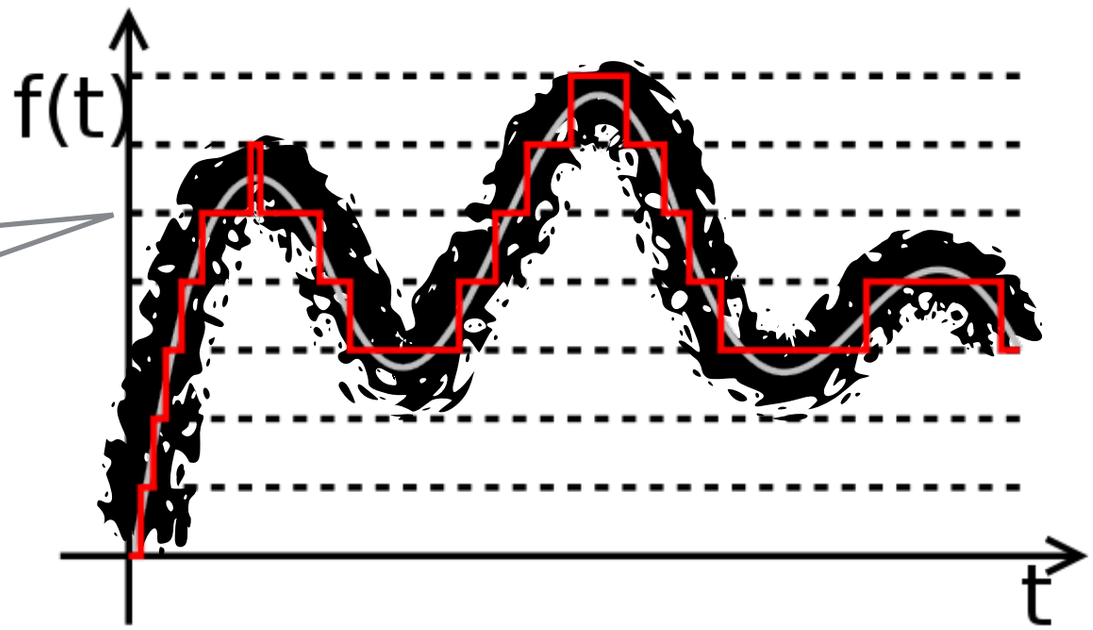
- physical processes are continuous
  - to process with computers, we must digitize it
  - digitization can change how we understand the signal
- digitization occurs in time and amplitude
  - time: sampling
  - amplitude: quantization



sampled data

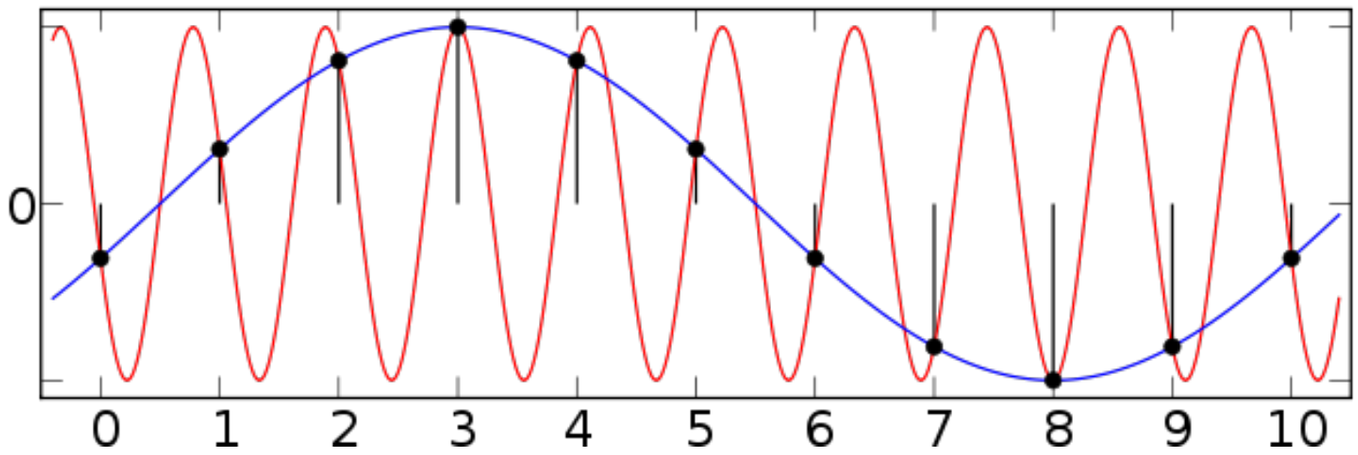
- quantization (amplitude)
    - introduces error in estimating amplitude of a signal
    - error can be reduced by adding more “bits per sample”
- most ADCs are 16 bits, considered “good enough”
- sufficient for most uses
    - not for others!

iPhone uses LPCM 32 bits, Q8.24



sampling errors

- sampling in time
  - introduces errors through ‘aliasing’
  - limits the range of frequencies able to be accurately captured
  - root of most common mistakes with sampled data



so how do I sample?

- heuristics
  - don't try to sample extremely small increments or values!
  - if capturing an "X"Hz signal, need to sample at least 2"X" Hz
  - changing sample rates is complicated, don't just drop every other sample
- for example, speech
  - majority of necessary energy in speech is located < 8000Hz
  - phones (for speech) typically capture at 16KHz or lower
  - good enough for speech, not music!

sanity check

- I need to detect an 80Hz signal
  - what sampling rate should we use?

- I want to detect a feather dropping next to the microphone
  - can the sound be detected?

the accelerate framework

- very powerful digital signal processing (DSP) library
  - look at vDSP Programming Guide on developer.apple.com for the complete API
- provides mathematics for performing fast DSP

input data | stride | scalar | output | array length

```
vDSP_vsmul(data, 1, &mult, data, 1, numFrames*numChannels);
```

```
void vDSP_vsmul (
    const float __vDSP_input1[],
    vDSP_Stride __vDSP_stride1,
    const float *__vDSP_input2,
    float __vDSP_result[],
    vDSP_Stride __vDSP_strideResult,
    vDSP_Length __vDSP_size
);
```

an example

```
[audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels) {
        float volume = userSetVolumeFromSlider;
        vDSP_vsmul(data, 1, &volume, data, 1, numFrames*numChannels);
        ringBuffer->AddNewInterleavedFloatData(data, numFrames, numChannels);
    }];

[audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels) {

        // get the max
        float maxVal = 0.0;
        vDSP_maxv(data, 1, &maxVal, numFrames*numChannels);

        printf("Max Audio Value: %f\n", maxVal);

    }];


[audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels)
 {
     vDSP_vsq(data, 1, data, 1, numFrames*numChannels);
     float meanVal = 0.0;
     vDSP_meanv(data, 1, &meanVal, numFrames*numChannels);
 }];
```

processing audio

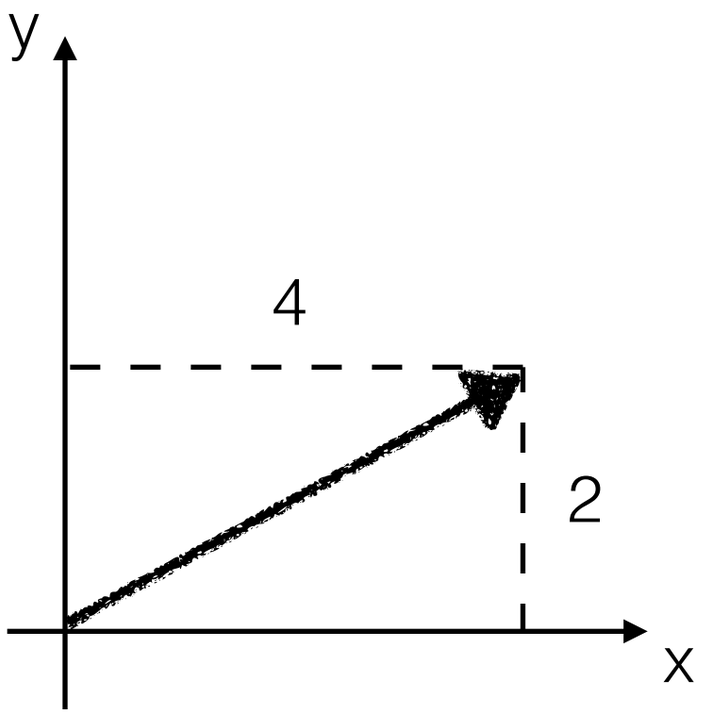
- lots of space to explore
  - great reference: “DSP First” by McClellan, Schafer, Yoder
    - http://www.rose-hulman.edu/DSPFirst/visible3/contents/index.htm
  - filtering
    - manipulate signal: high, low, bandpass
  - analysis
    - analyze characteristics of signal (like speech recognition)
  - synthesis
    - play around with different ideas, and see what sounds good!
    - not just pure synthesis, but also manipulation (like a guitar effect)
- for now, we’re going to stick with analysis
  - specifically, the **Fourier Transform**

the Fourier transform

- extremely useful, not just for signal junkies but also:
  - computer scientists, engineers, physicists, mathematicians, astronomers, oceanographers, health care professionals, etc.
- the Fourier Transform (FT) converts a time series into a frequency spectrum
  - the spectrum is an array of complex numbers which we will represent in polar form (i.e., with magnitude and phase)
  - each complex number represents a sinusoidal wave at a specific frequency
- we will use the FFT in the accelerate framework
  - complexity is $O(N \log_2(N))$ (for radix 2 FFT)

what is the FFT?

think of it as a vector projection (intuitively)



(4,2)

where did these numbers come from?

$\vec{x}$ $(1,0) \bullet (4,2) = 4$

$\vec{y}$ $(0,1) \bullet (4,2) = 2$

point by point multiplication and add it up!

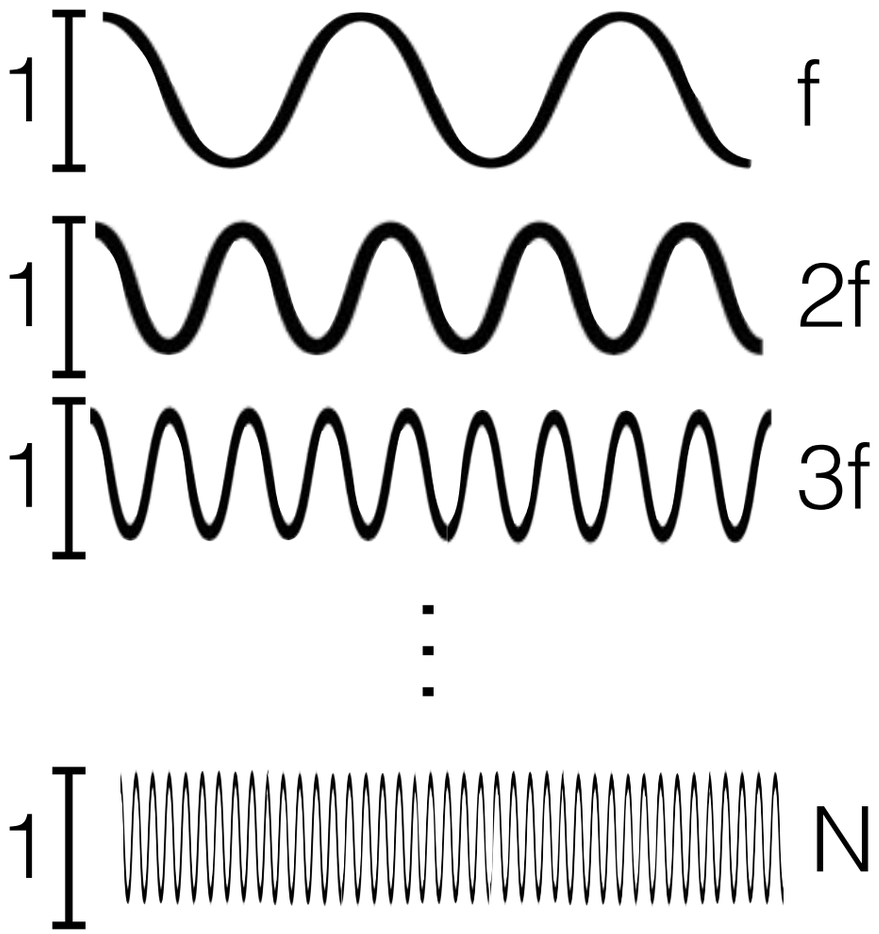
$\vec{x} \bullet \vec{y} = 0$

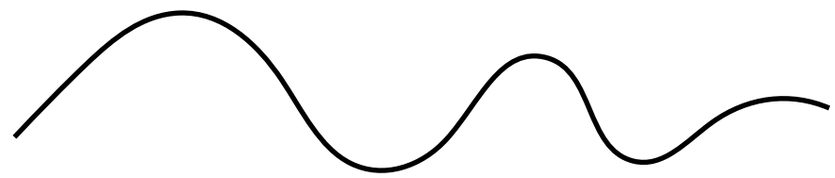
$|\mathbf{x}| = |\mathbf{y}| = 1$

tells us "how much" of the vector $\vec{x}$ is the results of other orthogonal vectors $\vec{y}$

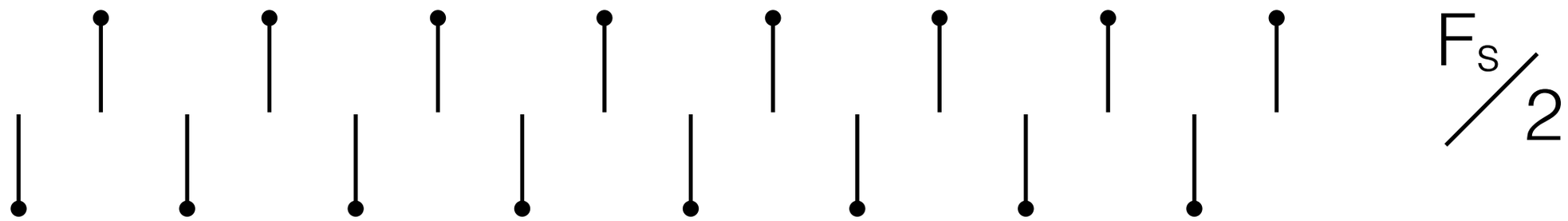
what is the FFT?

what if the orthogonal vectors were functions?





2.5	10°
0.1	45°
0.5	-12°
⋮	⋮
.005	0°

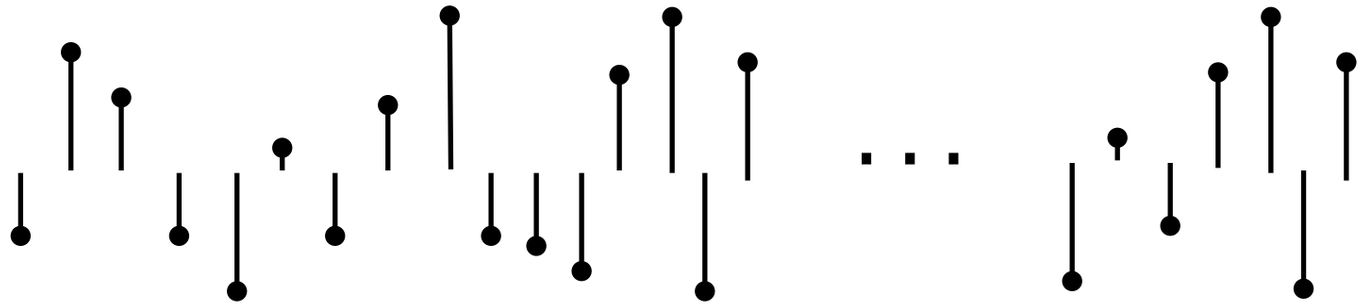


the FFT

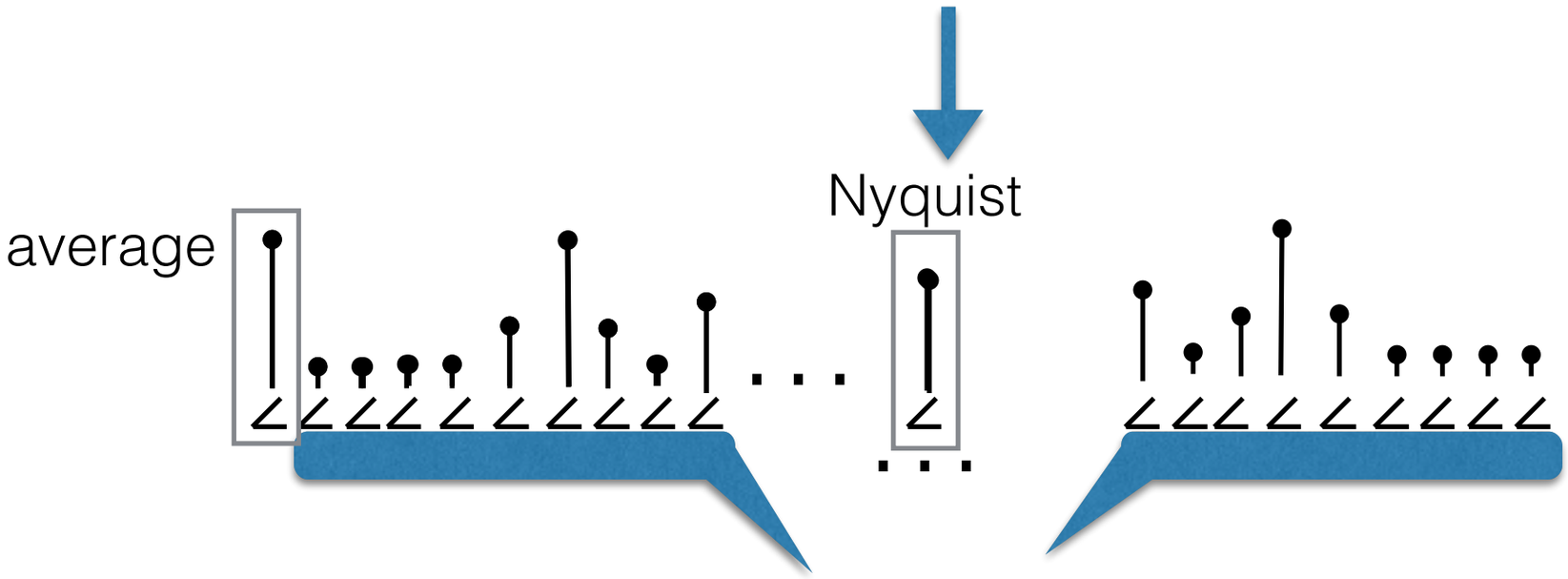
frequency content

0f	0.7	0°						=	0.7
1f	2.5	10°	+	2.5	-10°	-1f		~	2.5 cos(2*pi* (f) t+10°)
2f	0.1	45°	+	0.1	-45°	-2f		~	0.1 cos(2*pi* (2f) t+45°)
3f	0.5	-12°	+	0.5	12°	-3f		~	0.5 cos(2*pi* (3f) t-12°)
⋮	⋮	⋮		⋮	⋮				
N f	.005	0°	+	.005	0°	-N f		~	0.005 cos(2*pi* (Nf) t)

time and frequency



time series, N=512

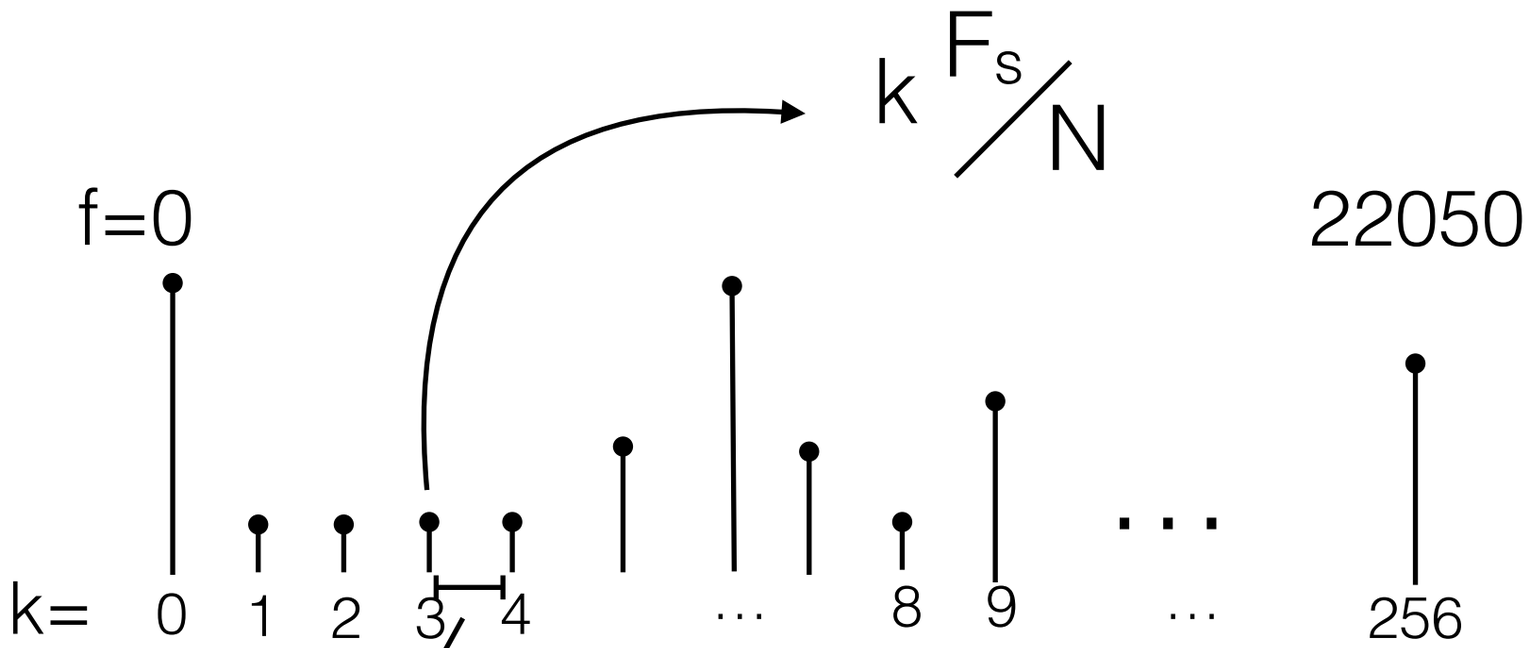


FFT, N = 512

magnitude, even symmetry

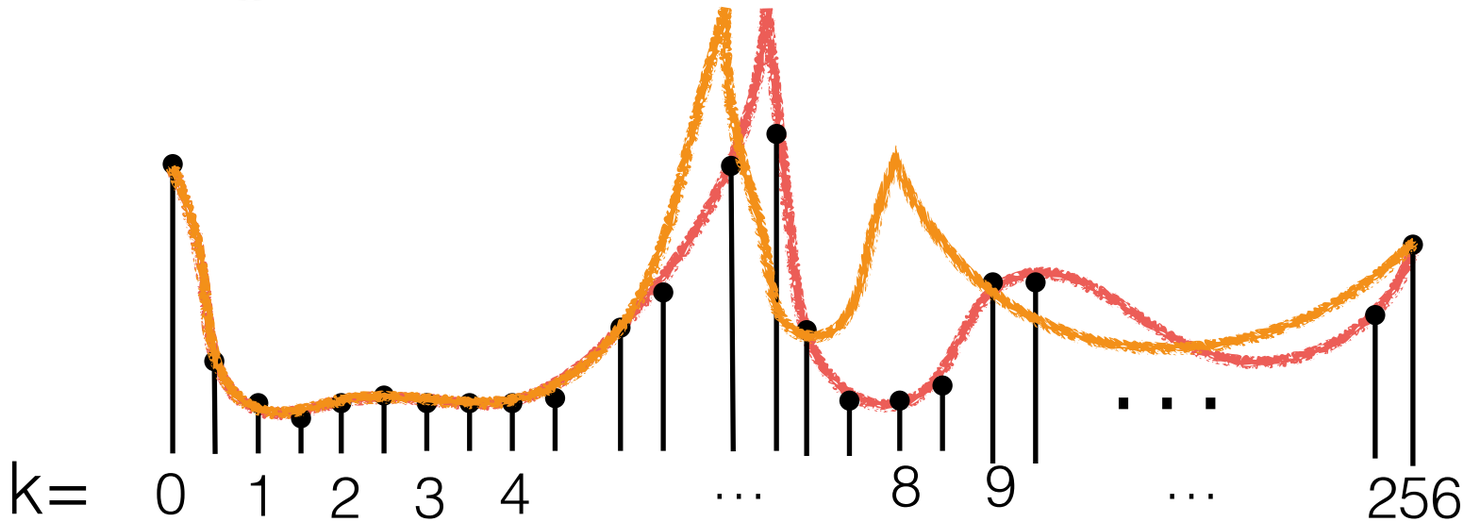
magnitude spectrum

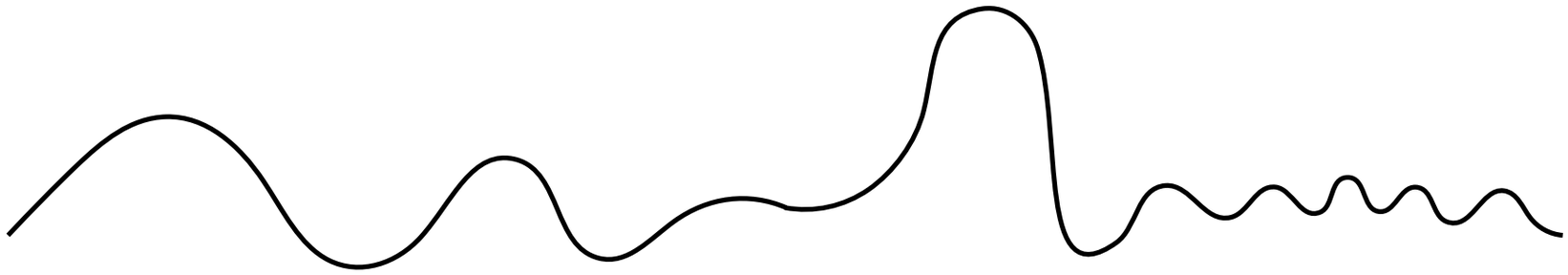
time and frequency



$$df = \frac{F_s/2}{N/2} = F_s/N = 22050/512 = 43.06 \text{ Hz}$$

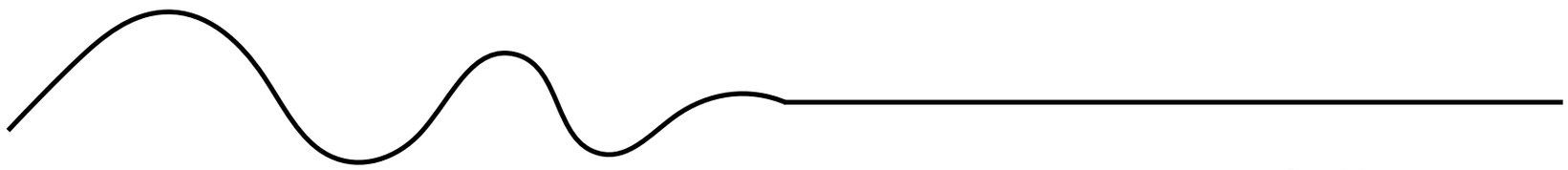
using the FFT





512 points

1024 points

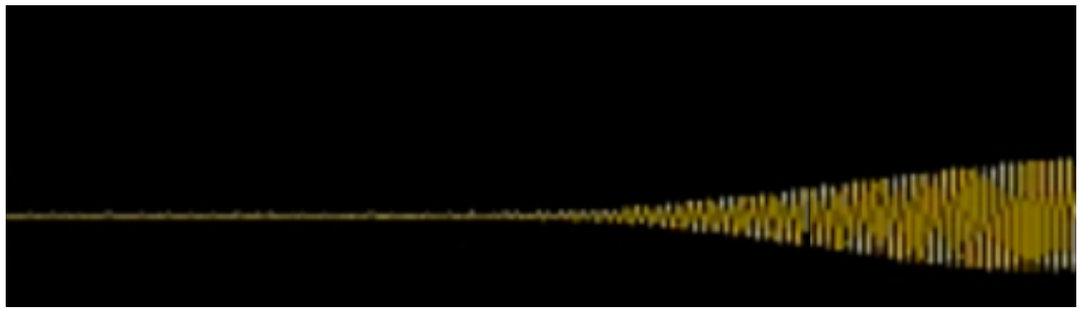


solution!

zero padding

using the FFT

raw audio



magnitude FFT

100

0

magnitude FFT
in dB

$20 \log_{10}(|FFT|)$

40

-30

programming the FFT

```
#import "SMUFFTHelper.h"


float           *fftMagnitudeBuffer;
float           *fftPhaseBuffer;
SMUFFTHelper    *fftHelper;


//setup the fft
fftHelper = new SMUFFTHelper(kBufferLength,kBufferLength,WindowTypeRect);
fftMagnitudeBuffer = (float *)calloc(kBufferLength/2,sizeof(float));
fftPhaseBuffer     = (float *)calloc(kBufferLength/2,sizeof(float));


free(fftMagnitudeBuffer);
free(fftPhaseBuffer);
delete fftHelper;


fftHelper->forward(0,inputAudioDataBuffer, fftMagnitudeBuffer, fftPhaseBuffer);
```

fft size

window size

window type

tear down in dealloc

```
enum WindowType {
    WindowTypeHann,
    WindowTypeHamming,
    WindowTypeRect,
    WindowTypeBlackman,
    };
```

reserved

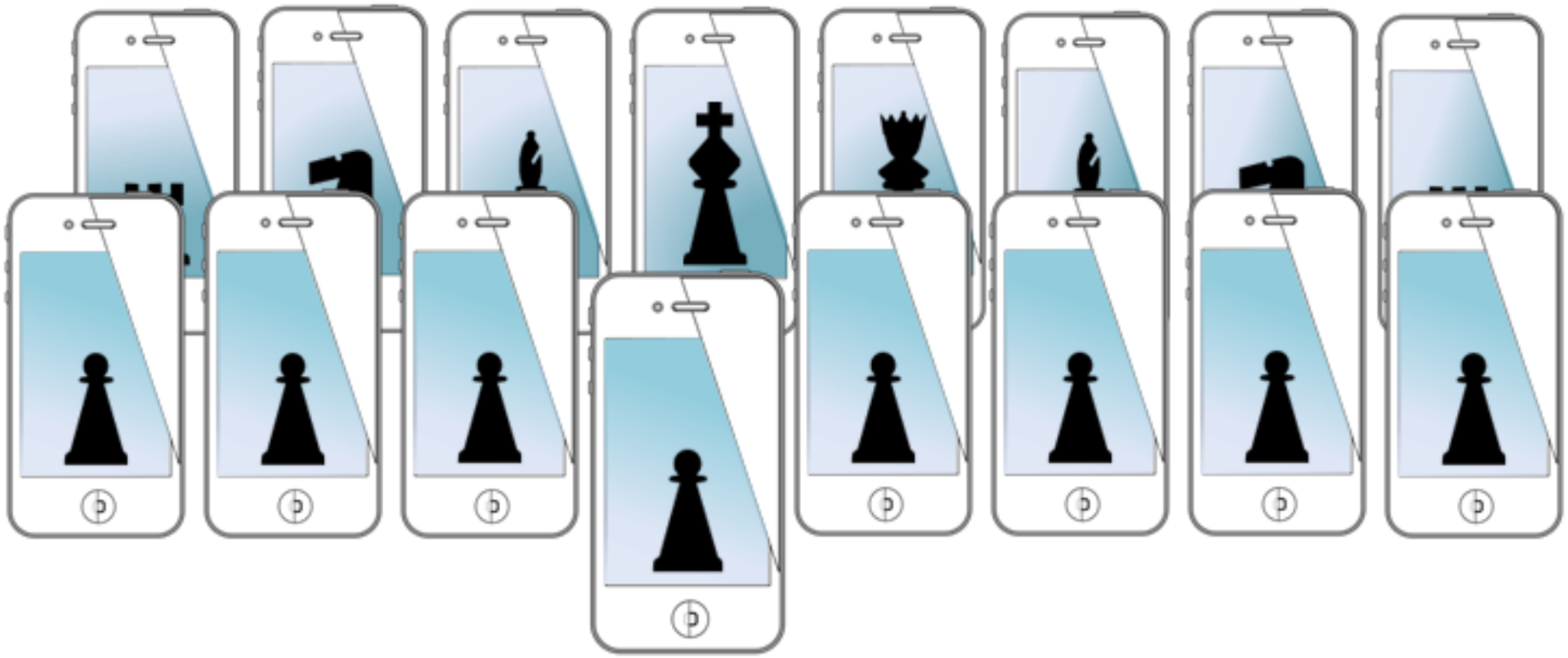
input array

magnitude out

phase out

highly optimized!! even using tricks we have not discussed

MOBILE SENSING LEARNING & CONTROL



CSE5323 & 7323

Mobile Sensing, Learning, and Control

lecture six: audio graphing, sampled data, accelerate, & FFT

Eric C. Larson, Lyle School of Engineering,
Computer Science and Engineering, Southern Methodist University