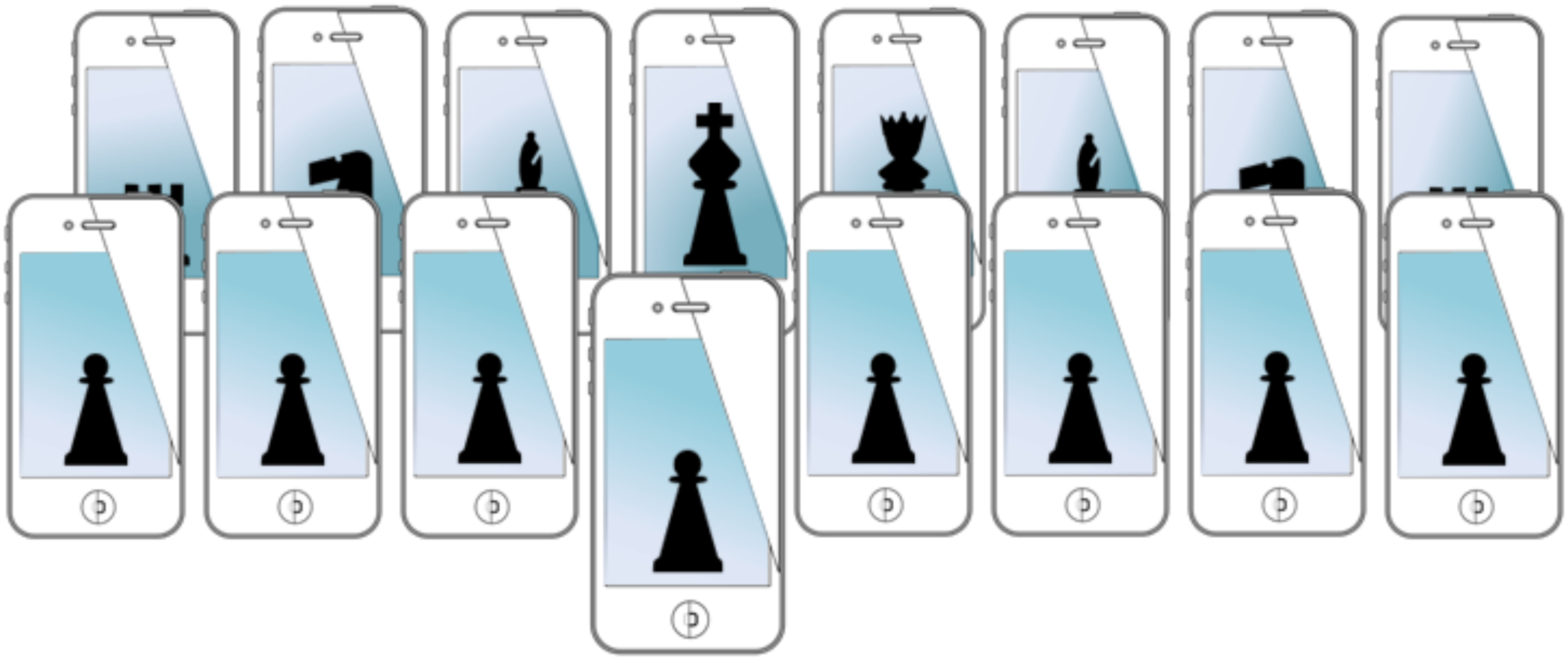
MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

week 3, lecture b: audio graphing, sampled data, & accelerate

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

course logistics

- I am out of town next week
  - there is a flipped module you can do outside of class
  - turn in by Thursday

course logistics

- Look at A2

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

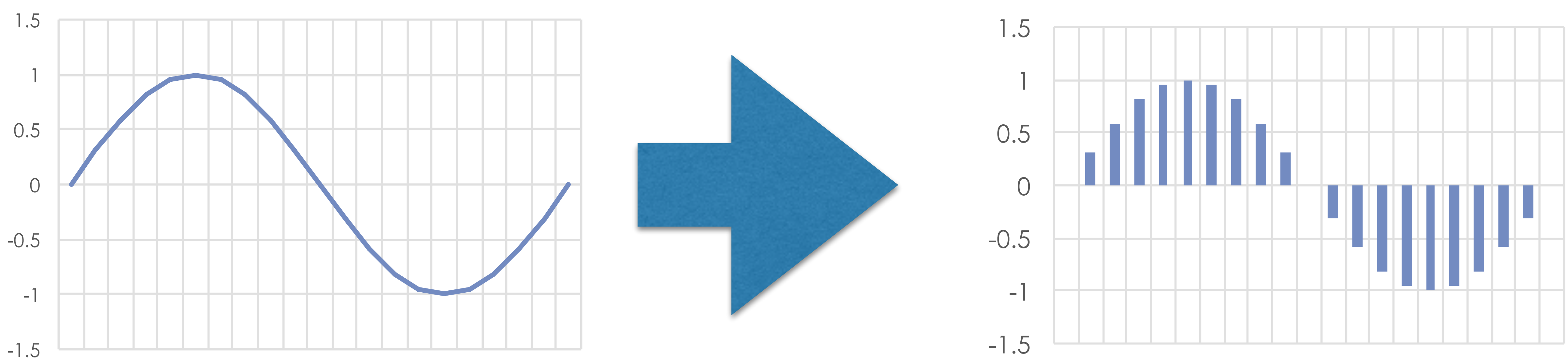
- dealing with sampled data
- the accelerate framework
  - massive digital signal processing library
- graphing audio fast (well, graphing anything)
  - must use lowest level graphing, OpenGL

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

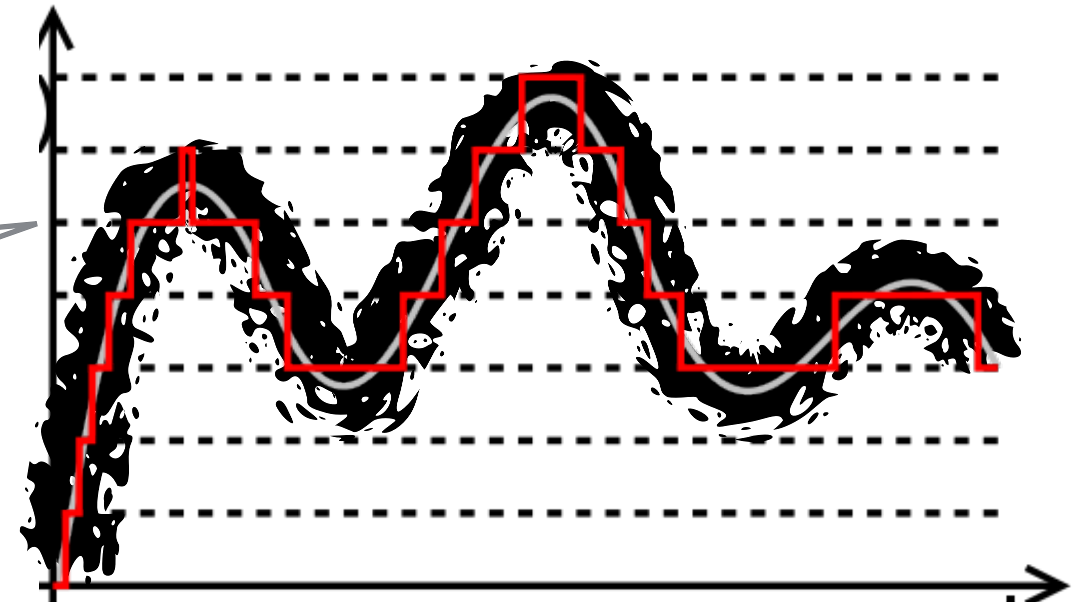
- physical processes are continuous
    - to process with computers, we must digitize it
    - digitization can change how we understand the signal
- digitization occurs in time and amplitude
    - time: sampling
    - amplitude: quantization



sampled data

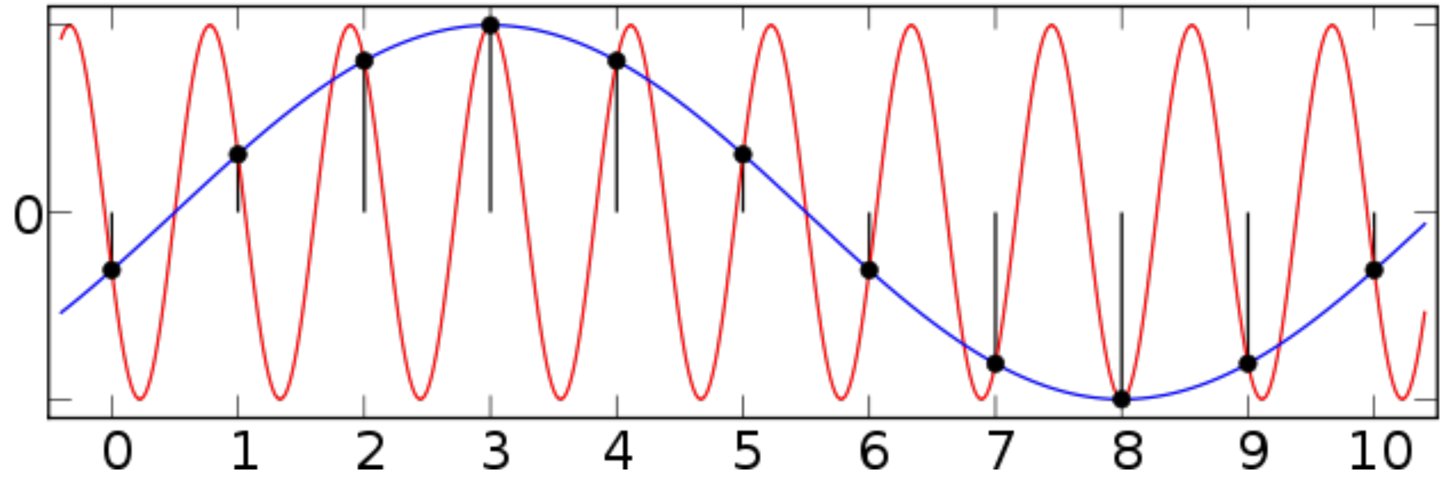
- quantization (amplitude)
  - introduces error in estimating amplitude of a signal
  - error can be reduced by adding more “bits per sample”
- most ADCs are 16 bits, considered “good enough”
- sufficient for most uses
  - not for others!

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?

making a sine wave

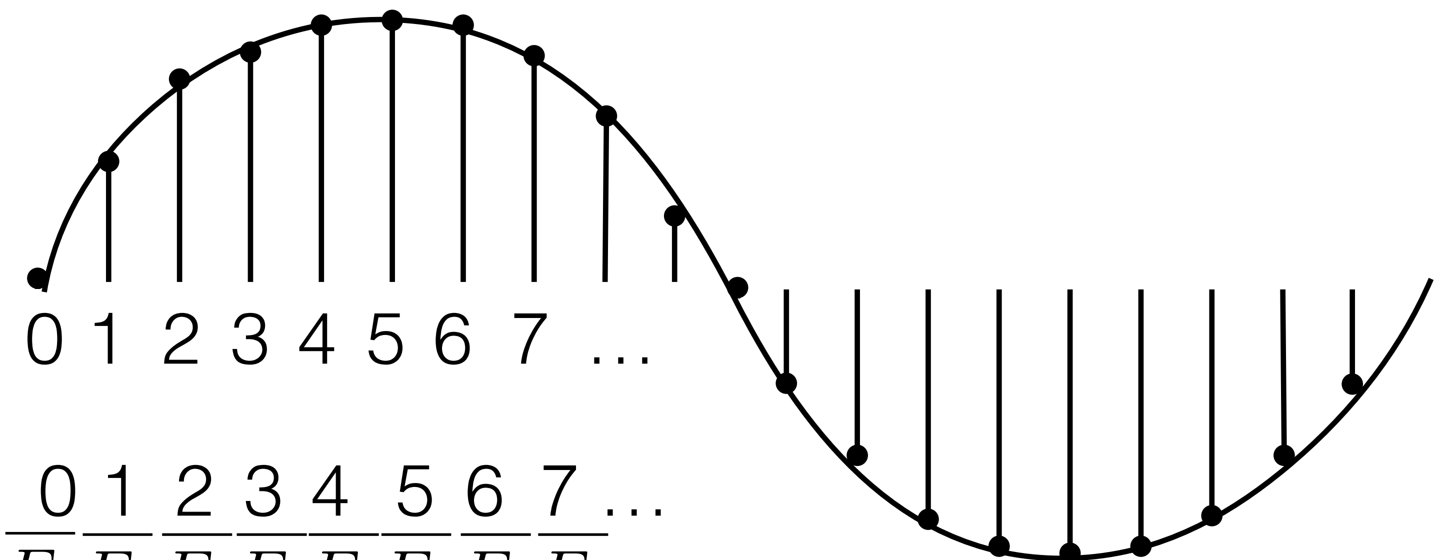
- we want to create a sine wave and play it to the speakers

$$g(t) = \sin(2\pi f t)$$ equation for sine wave

frequency in Hz

time in “seconds”

but we are working digitally, so we have an “index” in an array, not time!



n= 0 1 2 3 4 5 6 7 …

t= $\frac{0}{F_s}\frac{1}{F_s}\frac{2}{F_s}\frac{3}{F_s}\frac{4}{F_s}\frac{5}{F_s}\frac{6}{F_s}\frac{7}{F_s}\ldots$

$$g[n] = \sin\left(2\pi f\left(\frac{n}{F_s}\right)\right)$$

making a sine wave

$$g[n] = \sin\left(2\pi f\left(\frac{n}{F_s}\right)\right)$$

how to program this?

```
for (int n=0; n < numFrames; ++n)
 {
     data[n] = sin(2*M_PI*frequency*n/samplingRate);
 }
```

is this efficient?

```
float phase = 0.0;
double phaseIncrement = 2*M_PI*frequency/samplingRate;
 for (int n=0; n < numFrames; ++n)
 {
     data[n] = sin(phase);
     phase += phaseIncrement;
 }
```

making a sine wave

- bringing it all together $g[n] = \sin\left(2\pi f\left(\frac{n}{F_s}\right)\right)$

```
frequency = 18000.0; //starting frequency
__block float phase = 0.0;
__block float samplingRate = audioManager.samplingRate;

[audioManager setOutputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels)
 {
     double phaseIncrement = 2*M_PI*frequency/samplingRate;
     double sineWaveRepeatMax = 2*M_PI;
     for (int i=0; i < numFrames; ++i)
     {
         data[i] = sin(phase);

         phase += phaseIncrement;

         if (phase >= sineWaveRepeatMax) phase -= sineWaveRepeatMax;
     }

 }];
```

sample from the mic

- demo, play sine wave

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
);
```

examples

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

```
[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);

    }];
```

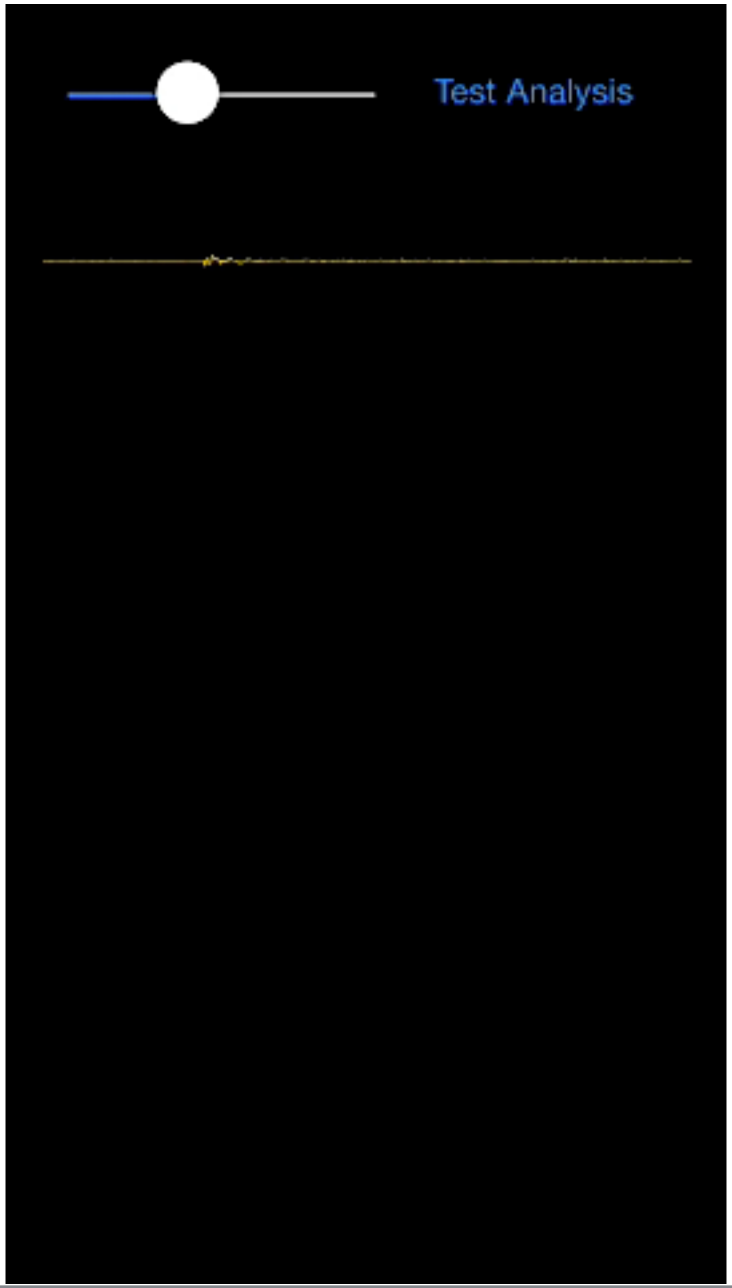
```
[audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32 numChannels)
 {
     vDSP_vsq(data, 1, data, 1, numFrames*numChannels);
     float meanVal = 0.0;
     vDSP_meanv(data, 1, &meanVal, numFrames*numChannels);
 }];
```

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

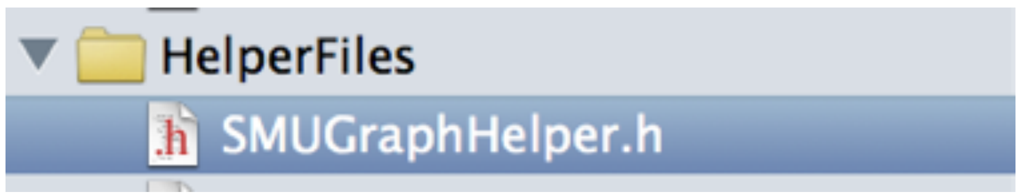
what if we want lots of graphs?

solution

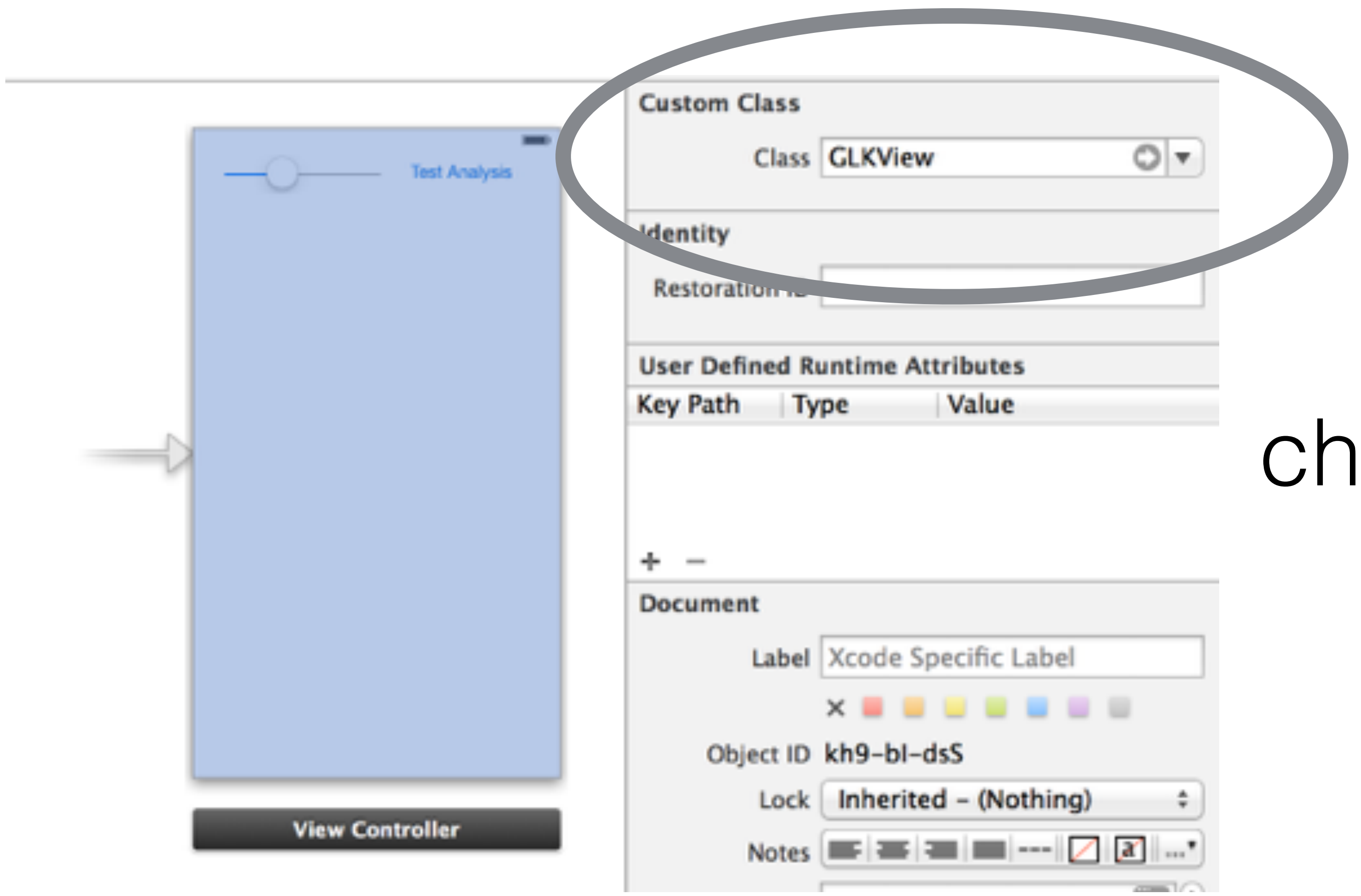
- use the GPU
- set vectors of data on a 2D plane
- let the renderer perform scaling, anti-aliasing, and bit blitting to screen
- …this is not a graphics course

easy solution

- use graph helper, which uses GLKView and GLKViewController



add .h and .m to project



change view to GLK View

the graph helper

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

@property (strong, nonatomic) SMUGraphHelper *graphHelper;
```

inherit from open GL

declare property

```
When setting up:
   // start animating the graph
    _graphHelper = [[SMUGraphHelper alloc]initWithController:self
                                    preferredFramesPerSecond:15
                                                   numGraphs:1
                                                   plotStyle:PlotStyleSeparated
                                           maxPointsPerGraph:BUFFER_SIZE];
```

setup GLKViewController

```
    enum PlotStyle {
        PlotStyleOverlaid,
        PlotStyleSeparated
    };
```

```
In view did load:
[self.graphHelper setScreenBoundsBottomHalf];
[self.graphHelper setScreenBoundsTopHalf];
[self.graphHelper setFullScreenBounds];
[self.graphHelper setBoundsWithTop:(float) bottom:(float) left:(float) right:(float)];
```

bounds for screen different options

setting data

called for each draw to screen

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

prototypes for setting scatter data

```
-(void) setGraphData:(float*)
      withDataLength:(int)
       forGraphIndex:(int)
   withNormalization:(float)
        withZeroValue:(float)
```

```
-(void) setGraphData:(float*)
      withDataLength:(int)
       forGraphIndex:(int)
```

```
//  override the GLKViewController update function, from OpenGLES
- (void)update{
    // just plot the audio stream

    // get audio stream data
    float* arrayData = malloc(sizeof(float)*BUFFER_SIZE);
    [self.buffer fetchFreshData:arrayData withNumSamples:BUFFER_SIZE];

    //send off for graphing
    [self.graphHelper setGraphData:arrayData
                    withDataLength:BUFFER_SIZE
                     forGraphIndex:0];

    [self.graphHelper update]; // update the graph
    free(arrayData);
}
```

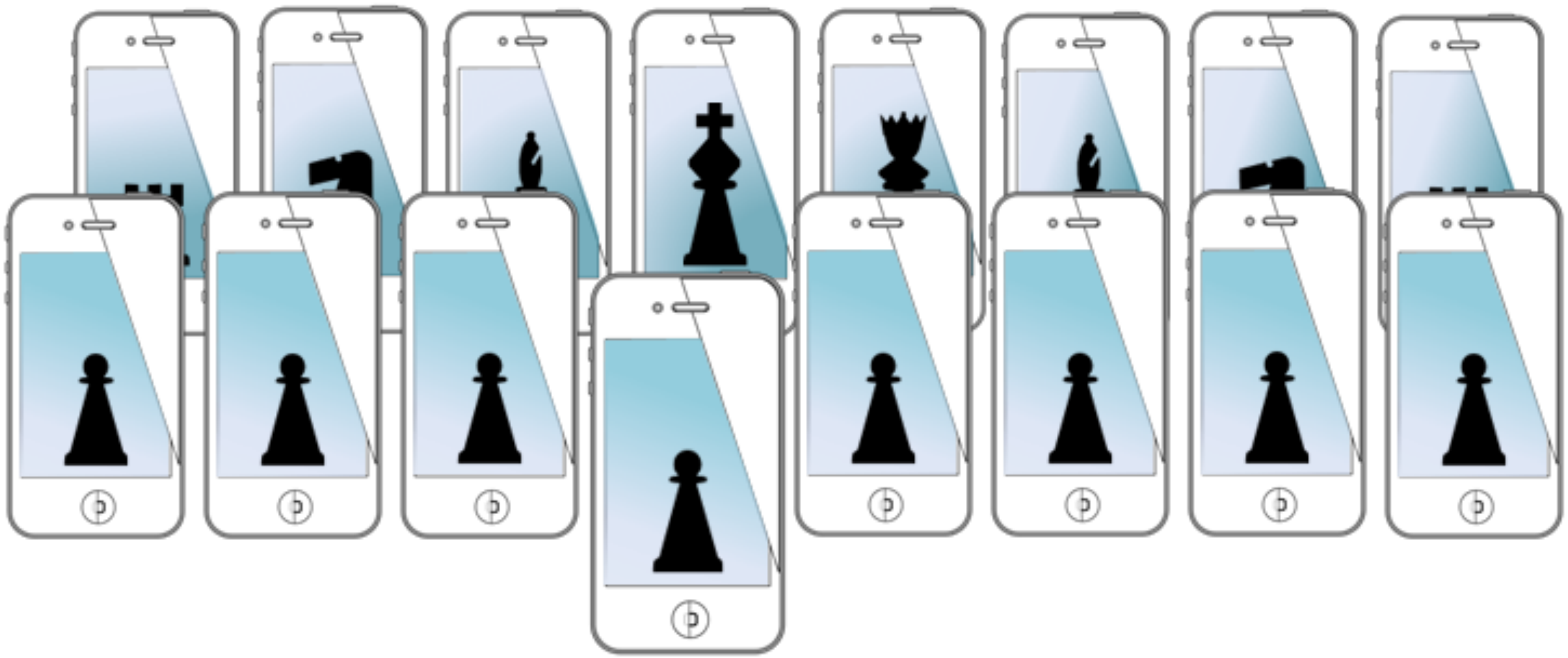
get data (shown here: from buffer)

set data for 0^{th} graph
no normalization

update render state and free memory

audio graphing demo!

MOBILE SENSING LEARNING



CSE5323 & 7323

Mobile Sensing and Learning

week 3, lecture b: audio graphing, sampled data, & accelerate

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