```
 1 //
 2 // Created by daran on 1/12/2017 to be used in ECE420
    Sp17 for the first time.
 3 // Modified by dwang49 on 1/1/2018 to adapt to
   Android 7.0 and Shield Tablet updates.
 4 //
 5
 6 #include <jni.h>
 7 #include "ece420_main.h"
 8 #include "ece420_lib.h"
 9 #include "kiss_fft/kiss_fft.h"
10
11 // JNI Function
12 extern "C" {
13 JNIEXPORT void JNICALL
14 Java_com_ece420_lab5_MainActivity_writeNewFreq(JNIEnv
    *env, jclass, jint);
15 }
16
17 // Student Variables
18 #define EPOCH_PEAK_REGION_WIGGLE 30
19 #define VOICED_THRESHOLD 200000000
20 #define FRAME_SIZE 1024
21 #define BUFFER_SIZE (3 * FRAME_SIZE)
22 #define F_S 48000
23 float bufferIn[BUFFER_SIZE] = {};
24 float bufferOut[BUFFER_SIZE] = {};
25 int newEpochIdx = FRAME_SIZE;
26
27 // We have two variables here to ensure that we never
    change the desired frequency while
28 // processing a frame. Thread synchronization, etc.
   Setting to 300 is only an initializer.
29 int FREQ_NEW_ANDROID = 300;
30 int FREQ_NEW = 300;
31
32 bool lab5PitchShift(float *bufferIn_temp) {
33     // Lab 4 code is condensed into this function
34     int periodLen = detectBufferPeriod(bufferIn);
35     float freq = ((float) F_S) / periodLen;
36
```

```
37      // If voiced
38      if (periodLen > 0) {
39
40          LOGD("Frequency detected: %f\r\n", freq);
41
42          // Epoch detection - this code is written for
     you, but the principles will be quizzed
43          std::vector<int> epochLocations;
44          findEpochLocations(epochLocations, bufferIn,
   periodLen);
45
46          // In this section, you will implement the
   algorithm given in:
47          // https://courses.engr.illinois.edu/ece420/
   lab5/lab/#buffer-manipulation-algorithm
48          //
49          // Don't forget about the following functions
   ! API given on the course page.
50          //
51          // getHanningCoef();
52          // findClosestInVector();
53          // overlapAndAdd();
54          // ************************ START YOUR CODE
   HERE  ************************** //
55
56          /* calculate new epoch spacing */
57          int new_epoch_spacing = F_S / FREQ_NEW;
58
59          /* incremement through new epochs based on
   spacing */
60          for (int i = newEpochIdx; i < 2 * FRAME_SIZE
   ; i += new_epoch_spacing) {
61              /* can optimize this by keeping track of
   the last epoch we mapped to later */
62              int curr_epoch_idx = findClosestInVector(
   epochLocations, i, 0, epochLocations.size());
63              int curr_epoch = epochLocations[
   curr_epoch_idx];
64              /* boundary check and find left and right
     indices for calculating p0 */
65              int left_epoch;
```

```
66              int right_epoch;
67              if (curr_epoch_idx == 0)
68                  left_epoch = 0;
69              else
70                  left_epoch = epochLocations[
   curr_epoch_idx - 1];
71              if (curr_epoch_idx == epochLocations.
   size() - 1)
72                  right_epoch = BUFFER_SIZE - 1;
73              else
74                  right_epoch = epochLocations[
   curr_epoch_idx + 1];
75
76              /* calculate p0 */
77              int p0 = (right_epoch - left_epoch) / 2;
78
79              int window_len = 2*p0 + 1;
80              /* apply window to input centered around
    original epoch, and add it to output centered at
   new epoch */
81              for (int j = 0; j < window_len; j++) {
82                  int windowed_idx = j; // index into
   window
83                  int buffer_in_idx = (curr_epoch - p0
   ) + j; // data to use centered around original epoch
84                  int buffer_out_idx = (i - p0) + j;
   // location to add windowed data centered around new
    epoch
85                  /* only sum overlapped data if
   indices are valid */
86                  if ((buffer_out_idx < BUFFER_SIZE
    && buffer_out_idx > 0) && (buffer_in_idx <
   BUFFER_SIZE && buffer_in_idx > 0))
87                      bufferOut[buffer_out_idx] +=
   getHanningCoef(window_len, windowed_idx) * bufferIn[
   buffer_in_idx];
88              }
89
90          }
91
92
```

```
 93
 94
 95
 96
 97         // ************************ END YOUR CODE
    HERE  ***************************** //
 98     }
 99
100     // Final bookkeeping, move your new pointer back
    , because you'll be
101     // shifting everything back now in your circular
     buffer
102     newEpochIdx -= FRAME_SIZE;
103     if (newEpochIdx < FRAME_SIZE) {
104         newEpochIdx = FRAME_SIZE;
105     }
106
107     return (periodLen > 0);
108 }
109
110 void ece420ProcessFrame(sample_buf *dataBuf) {
111     // Keep in mind, we only have 20ms to process
    each buffer!
112     struct timeval start;
113     struct timeval end;
114     gettimeofday(&start, NULL);
115
116     // Get the new desired frequency from android
117     FREQ_NEW = FREQ_NEW_ANDROID;
118
119     // Data is encoded in signed PCM-16, little-
    endian, mono
120     int16_t data[FRAME_SIZE];
121     for (int i = 0; i < FRAME_SIZE; i++) {
122         data[i] = ((uint16_t) dataBuf->buf_[2 * i
    ]) | (((uint16_t) dataBuf->buf_[2 * i + 1]) << 8);
123     }
124
125     // Shift our old data back to make room for the
    new data
126     for (int i = 0; i < 2 * FRAME_SIZE; i++) {
```

```
127         bufferIn[i] = bufferIn[i + FRAME_SIZE - 1];
128     }
129
130     // Finally, put in our new data.
131     for (int i = 0; i < FRAME_SIZE; i++) {
132         bufferIn[i + 2 * FRAME_SIZE - 1] = (float)
    data[i];
133     }
134
135     // The whole kit and kaboodle -- pitch shift
136     bool isVoiced = lab5PitchShift(bufferIn);
137
138     if (isVoiced) {
139         for (int i = 0; i < FRAME_SIZE; i++) {
140             int16_t newVal = (int16_t) bufferOut[i];
141
142             uint8_t lowByte = (uint8_t) (0x00ff &
    newVal);
143             uint8_t highByte = (uint8_t) ((0xff00 &
    newVal) >> 8);
144             dataBuf->buf_[i * 2] = lowByte;
145             dataBuf->buf_[i * 2 + 1] = highByte;
146         }
147     }
148
149     // Very last thing, update your output circular
    buffer!
150     for (int i = 0; i < 2 * FRAME_SIZE; i++) {
151         bufferOut[i] = bufferOut[i + FRAME_SIZE - 1
    ];
152     }
153
154     /* the 'past' buffer was perfectly reconstructed
     and sent, so we can shift it out */
155     for (int i = 0; i < FRAME_SIZE; i++) {
156         bufferOut[i + 2 * FRAME_SIZE - 1] = 0;
157     }
158
159     gettimeofday(&end, NULL);
160     LOGD("Time delay: %ld us",  ((end.tv_sec *
    1000000 + end.tv_usec) - (start.tv_sec * 1000000 +
```

```
160 start.tv_usec)));
161 }
162
163 // Returns lag l that maximizes sum(x[n] x[n-k])
164 int detectBufferPeriod(float *buffer) {
165
166     float totalPower = 0;
167     for (int i = 0; i < BUFFER_SIZE; i++) {
168         totalPower += buffer[i] * buffer[i];
169     }
170
171     if (totalPower < VOICED_THRESHOLD) {
172         return -1;
173     }
174
175     // FFT is done using Kiss FFT engine. Remember
    to free(cfg) on completion
176     kiss_fft_cfg cfg = kiss_fft_alloc(BUFFER_SIZE,
    false, 0, 0);
177
178     kiss_fft_cpx buffer_in[BUFFER_SIZE];
179     kiss_fft_cpx buffer_fft[BUFFER_SIZE];
180
181     for (int i = 0; i < BUFFER_SIZE; i++) {
182         buffer_in[i].r = bufferIn[i];
183         buffer_in[i].i = 0;
184     }
185
186     kiss_fft(cfg, buffer_in, buffer_fft);
187     free(cfg);
188
189
190     // Autocorrelation is given by:
191     // autoc = ifft(fft(x) * conj(fft(x))
192     //
193     // Also, (a + jb) (a - jb) = a^2 + b^2
194     kiss_fft_cfg cfg_ifft = kiss_fft_alloc(
    BUFFER_SIZE, true, 0, 0);
195
196     kiss_fft_cpx multiplied_fft[BUFFER_SIZE];
197     kiss_fft_cpx autoc_kiss[BUFFER_SIZE];
```

```
198
199     for (int i = 0; i < BUFFER_SIZE; i++) {
200         multiplied_fft[i].r = (buffer_fft[i].r *
    buffer_fft[i].r)
201                               + (buffer_fft[i].i *
    buffer_fft[i].i);
202         multiplied_fft[i].i = 0;
203     }
204
205     kiss_fft(cfg_ifft, multiplied_fft, autoc_kiss);
206     free(cfg_ifft);
207
208     // Move to a normal float array rather than a
    struct array of r/i components
209     float autoc[BUFFER_SIZE];
210     for (int i = 0; i < BUFFER_SIZE; i++) {
211         autoc[i] = autoc_kiss[i].r;
212     }
213
214     // We're only interested in pitches below 1000Hz
    .
215     // Why does this line guarantee we only identify
     pitches below 1000Hz?
216     int minIdx = F_S / 1000;
217     int maxIdx = BUFFER_SIZE / 2;
218
219     int periodLen = findMaxArrayIdx(autoc, minIdx,
    maxIdx);
220     float freq = ((float) F_S) / periodLen;
221
222     // TODO: tune
223     if (freq < 50) {
224         periodLen = -1;
225     }
226
227     return periodLen;
228 }
229
230
231 void findEpochLocations(std::vector<int> &
    epochLocations, float *buffer, int periodLen) {
```

```
232     // This algorithm requires that the epoch
    locations be pretty well marked
233
234     int largestPeak = findMaxArrayIdx(bufferIn, 0,
    BUFFER_SIZE);
235     epochLocations.push_back(largestPeak);
236
237     // First go right
238     int epochCandidateIdx = epochLocations[0] +
    periodLen;
239     while (epochCandidateIdx < BUFFER_SIZE) {
240         epochLocations.push_back(epochCandidateIdx);
241         epochCandidateIdx += periodLen;
242     }
243
244     // Then go left
245     epochCandidateIdx = epochLocations[0] -
    periodLen;
246     while (epochCandidateIdx > 0) {
247         epochLocations.push_back(epochCandidateIdx);
248         epochCandidateIdx -= periodLen;
249     }
250
251     // Sort in place so that we can more easily find
     the period,
252     // where period = (epochLocations[t+1] +
    epochLocations[t-1]) / 2
253     std::sort(epochLocations.begin(), epochLocations
    .end());
254
255     // Finally, just to make sure we have our epochs
     in the right
256     // place, ensure that every epoch mark (sans
    first/last) sits on a peak
257     for (int i = 1; i < epochLocations.size() - 1; i
    ++) {
258         int minIdx = epochLocations[i] -
    EPOCH_PEAK_REGION_WIGGLE;
259         int maxIdx = epochLocations[i] +
    EPOCH_PEAK_REGION_WIGGLE;
260
```

```
261         int peakOffset = findMaxArrayIdx(bufferIn,
    minIdx, maxIdx) - minIdx;
262         peakOffset -= EPOCH_PEAK_REGION_WIGGLE;
263 
264         epochLocations[i] += peakOffset;
265     }
266 }
267 
268 void overlapAddArray(float *dest, float *src, int
    startIdx, int len) {
269     int idxLow = startIdx;
270     int idxHigh = startIdx + len;
271 
272     int padLow = 0;
273     int padHigh = 0;
274     if (idxLow < 0) {
275         padLow = -idxLow;
276     }
277     if (idxHigh > BUFFER_SIZE) {
278         padHigh = BUFFER_SIZE - idxHigh;
279     }
280 
281     // Finally, reconstruct the buffer
282     for (int i = padLow; i < len + padHigh; i++) {
283         dest[startIdx + i] += src[i];
284     }
285 }
286 
287 
288 JNIEXPORT void JNICALL
289 Java_com_ece420_lab5_MainActivity_writeNewFreq(
    JNIEnv *env, jclass, jint newFreq) {
290     FREQ_NEW_ANDROID = (int) newFreq;
291     return;
292 }
```