Constant Q Transform Implementation for Signal Analyzers (Xcode 3.1.2)


```
/*
 *   RoomAnalyzer.h
 *
 *  Created on 12/05/08.
 *  Copyright 2008 TangerineTech Engineering http://www.tangerinetech.net / Ing. Tommaso Giunti / Dott. Massimo Magrini
 *  Written By Ing. Tommaso Giunti and Dott. Massimo Magrini.
 *
 */


#ifndef __ROOM_ANALYZER__
#define __ROOM_ANALYZER__


      #define  nbr_points    16384
      #define      BINS      24                  //bin per OCTAVE
      #define THRESHOLD      0.0054



#include "globals.h"
#include "FFTReal.h"

class RoomAnalyzer {
       public:
              RoomAnalyzer();
              ~RoomAnalyzer();


         int       enable_buffer;
         int       nextS;
         float     f_step;
         float     FFTmed;

         int               samples_count;
         int               undersampling;

         float     * input_buffer;
         float     * f; //

         float     * hamming;                              ////////////
         float Q;
         int K;
         //float   * specKernel;
         float     specKernel[(nbr_points/2)+1][NUM_FREQUENCIES];
         int conta[NUM_FREQUENCIES];


         FFTReal * fftReal;


         float     * FFT; // MUST CONTAIN MODULE
         float   Roomtransfer [NUM_FREQUENCIES];

         void DoBuffer(float sample);
         void DoConstantQ();
         float GetTransfer(int i);
         int GetGraph(int chan, int w, int h, int * xx, int *yy);

       //private:
};


#endif /* ROOM_ANALYZER */
```

```
/*
 *   RoomAnalyzer.cpp
 *
 *  Created on 12/05/08.
 *  Copyright 2008 TangerineTech Engineering http://www.tangerinetech.net / Ing. Tommaso Giunti / Dott. Massimo Magrini
 *  Written By Ing. Tommaso Giunti and Dott. Massimo Magrini.
 *
 */


#include "RoomAnalyzer.h"
#include "utils.h"
#include "globals.h"
#include "math.h"


/*     int conta[NUM_FREQUENCIES] = {
       2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
       2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
       2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3,
       3,3,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,
       4,6,6,6,6,6,6,6,7,7,7,7,7,8,8,8,8,8,9,9,9,9,10,10,10,10,11,11,
       11,12,12,12,13,13,13,14,14,15,15,15,16,16,17,17,18,18,19,19,
       20,20,21,22,22,23,24,24,25,26,26,27,28,29,30,30,31,32,33,34,
       35,36,37,38,39,41,42,43,44,45,47,48,50,51,52,54,56,57,59,61,
       62,64,66,68,70,72,74,76,79,81,83,86,88,90,93,96,99,102,105,
       108,111,114,118,121,125,128,133,136,140,144,149,153,158,161,167,
       172,176,182,186 };
*/

extern int AnalyzeButton;

RoomAnalyzer::RoomAnalyzer() {

       enable_buffer = 1;
       nextS = 0;
       AnalyzeButton = 0; //INITIALIZE TO ZERO
                          // THEN CTRL BY GUI

       //FFT CLASS BUILDER
       fftReal = new FFTReal(nbr_points);
       input_buffer  = (float *) malloc((size_t) sizeof(float)*nbr_points);

       for (int j=0; j < nbr_points; j++)
          input_buffer[j]= 0.000000119209;

       f  = (float *) malloc((size_t) sizeof(float)*nbr_points);


       //UNDERSAMPLING
       int SR = gSamplingRate;
       switch(SR)
          {
          case (192000):
                    f_step = (float) 48000 / (float)nbr_points;
                    undersampling = 4;
                    SR=48000;
          break;
          case (96000):
                    f_step = (float)48000 / (float)nbr_points;
                    undersampling = 2;
                    SR=48000;
          break;
          case (48000):
                    f_step = (float)48000 / (float)nbr_points;
                    undersampling = 1;
                    SR=48000;
          break;
          case (44100):
                    f_step = (float)44100 / (float)nbr_points;
                    undersampling = 1;
                    SR=44100;
          break;
          }
       samples_count = 0;


       // INITIALIZE CONSTANT Q TRANSFORM
       Q = float(1) / float( (pow(2,((float)1/(float)BINS))-1) );
       int len;
       int s;
       hamming  = (float *) malloc((size_t) sizeof(float)*nbr_points);
       FFT  = (float *) malloc((size_t) sizeof(float)* ((nbr_points/2)+1));


       for (int j=0; j < NUM_FREQUENCIES; j++)
          {
                    len = ceil((float)(Q * SR) / (float)(frequencies[0] * pow(2,((float)j /(float) BINS)))); // siiii!!
                    if (len > nbr_points )
                              {
                              len = nbr_points;
                              }

                    s = ceil( (float)nbr_points/(float)2 - (float)len/(float)2 );

                    for (int n=0; n<nbr_points; n++)
                              {
                              hamming[n] = 0.0;
                              };

                    for (int n=0; n<len; n++)
                              {
                              hamming[s+n] = (0.54 - 0.46 * cos((float)(2*pi*n)/(float)(len-1)));
```

```
                        };

                fftReal->do_fft (f, hamming);

                //fftReal->do_AbsFFT (FFT, f,nbr_points);  //IF YOU USE THIS USE pow(-1,j).

                for (int i = 0; i <=(nbr_points/2); i ++)
                        {
                        FFT[i] = f[i]/len;  // <----------
                        };


                conta[j] = 0;
                for (int fr=0; fr<=(nbr_points/2); fr++)
                        {
                                specKernel[fr][j]=FFT[fr];
                                if (fabs(FFT[fr]) > THRESHOLD)
                                        {conta[j]++;
                                        //printf("f=%d  FFT=%f\n",fr,FFT[fr]);
                                        }
                        };
                Roomtransfer[j] = 0.0;

                //printf("F=%f | j=%d | len=%d | s= %d | conta=%d\n",frequencies[j],j,len,s,conta[j]);
        };



};


RoomAnalyzer::~RoomAnalyzer(){
};


void RoomAnalyzer::DoBuffer(float in_sample) {


   if (samples_count == 0)
    if (enable_buffer == 1){
        if (nextS < nbr_points ) {

                input_buffer[nextS] = in_sample;

                nextS++;
        }
       }

      samples_count = samples_count +1;
      samples_count = samples_count % undersampling;
};



void RoomAnalyzer::DoConstantQ() {
      int fk;
      double sumRe;
      double sumIm;

      for (int i=0; i<NUM_FREQUENCIES; i++)
         {
         sumRe=0;
         sumIm=0;
         fk = ceil(frequencies[i] / f_step);

         sumRe=f[fk] * specKernel[0][i];
         sumIm=f[fk+nbr_points/2] * specKernel[0][i];

         for (int j=1; j< conta[i]; j++)
                {

                //sumRe = sumRe + ( f[fk+j] + f[fk-j] ) * pow(-1,j) * specKernel[j][i];
                //sumIm = sumIm + ( f[fk+nbr_points/2+j] + f[fk+nbr_points/2-j] ) * pow(-1,j) * specKernel[j][i];

                sumRe = sumRe + (( f[fk+j] + f[fk-j] )  * specKernel[j][i]);
                sumIm = sumIm + (( f[fk+nbr_points/2+j] + f[fk+nbr_points/2-j] ) * specKernel[j][i]);
                };

         float dbval = Lin2Db( sqrt(sumRe * sumRe + sumIm * sumIm) ); //MUST BE SCALED ADEQUATELY
         if (dbval < -90)
                dbval = -90;

         Roomtransfer[i] = 0.95 * Roomtransfer[i] + 0.05 * dbval; // LP

         //printf("Roomtransfer= %f\n",Roomtransfer[i]);
         //printf("dbval= %f\n",dbval);

         };
};


float RoomAnalyzer::GetTransfer(int i) {
      if (i < NUM_FREQUENCIES)
                return Roomtransfer[i];
      else
                return 0;
}


int RoomAnalyzer::GetGraph(int chan, int w, int h, int * xx, int *yy) {
      int px;
```

```
	int py;

	enable_buffer=0;


//if(chan == 0) {
	fftReal->do_fft (f, input_buffer);		// CALLS FFT

	nextS = 0;

	DoConstantQ(); //ADAPTS FREQS. ACCORDING TO CONSTANT-Q

//}


	for (int i=0; i< NUM_FREQUENCIES; i++) {
		GraphGetCoord(i,GetTransfer(i),w,h, &px,&py);

	   //if (py < -500)
			//py = -500;
	   //
	   if (py > 209)
	   py = 209;

	   xx[i] = px;
	   yy[i] = py;

	};

	enable_buffer=1;
	return 0;
};
```