Used \_kiss\_fft\_guts.h, kiss\_fft.h, and kiss\_fft.c from: <https://github.com/mborgerding/kissfft>  
(Compile without openMP: g++ main.cpp -o main -I kissfft kiss\_fft.c)

My file, sfftScript.cpp:

// Script to do what the Matlab does, hopefully

#include <stdio.h>

#include <cstdlib>

#include <cmath>

#include <vector>

#include <iostream>

#include <cstring>

// Using a cute lil FFT library I found. Makes computation easier

#include "kiss\_fft.h"

#define PI 3.14159265358979323846

typedef struct Complex

{

double real;

double imag;

} complex\_t;

double hanning(int len, int n)

{

// Basically, just make the multiplier...

return 0.5 \* (1.0 - cos(2.0 \* PI \* ((double)n / (double)len)));

}

double absComplex(complex\_t value)

{

return sqrt((value.real \* value.real) + (value.imag \* value.imag));

}

int main()

{

int N = 0; // This should store the length of above.

// In MATLAB, N = 1104573. Hopefully we're close-ish?

// Start by reading the file gliss.ascii

// - One channel is the left and one is the right

// - Write code that can either read the left or right channel

FILE \*fp = fopen("gliss.ascii", "r");

if (fp == NULL)

{

std::cout << "oof" << std::endl;

}

char \*line = NULL;

bool stillReading = true;

std::vector<double> left;

std::vector<double> right;

std::cout << "Reading file..." << std::endl;

line = (char \*)malloc(256);

while (stillReading)

{

memset(line, 0, 256);

if (fgets(line, 256, fp) != NULL)

{

// Read the line, split it into two doubles, and store them in the vectors

double l, r;

sscanf(line, " %lf %lf", &l, &r);

left.push\_back(l);

right.push\_back(r);

N++;

}

else

{

stillReading = false;

}

}

free(line);

// close the file

fclose(fp);

std::cout << "Done reading " << N << " lines." << std::endl;

#pragma omp barrier

int nFFT = 1024;

int hop = floor(nFFT / 4);

std::cout << "Hop: " << hop << std::endl;

int nFrames = floor(N / hop) - 1;

std::cout << nFrames << " frames." << std::endl;

// Create an F matrix of size nFFT x nFrames, all filled with zeroes

// These are all complex doubles...

complex\_t \*F = (complex\_t \*)calloc(nFFT \* nFrames \* sizeof(complex\_t), sizeof(complex\_t));

// Create a w array of size nFFT, populated by 'Hanning' values

double \*w = (double \*)malloc(nFFT \* sizeof(double));

double wchecksum = 0;

#pragma omp for

for (int i = 0; i < nFFT; i++)

{

w[i] = hanning(nFFT, i);

wchecksum += w[i];

}

#pragma omp barrier

// Compute a w checksum (for correctness checking!!!)

std::cout << "W checksum: " << wchecksum << "." << std::endl;

// Also create G and make it way too big because

double \*G = (double \*)calloc((nFFT / 2) \* nFrames \* sizeof(double), sizeof(double));

// ACTUAL COMPUTATION

// Set up kiss FFT

kiss\_fft\_cfg cfg = kiss\_fft\_alloc(nFFT, 0, 0, 0);

// This is where we'll load things in

kiss\_fft\_cpx \*cx\_in = new kiss\_fft\_cpx[nFFT];

// \* This is the F matrix from MATLAB

kiss\_fft\_cpx \*cx\_out = new kiss\_fft\_cpx[nFFT];

int iStart;

for (int n = 0; n < nFrames; n++)

{

iStart = (n - 1) \* hop;

if (iStart + nFFT > N)

break;

// Load in the complex data

#pragma omp for

for (int k = 0; k < nFFT; k++)

{

if (iStart + 1 + k >= left.size())

break;

cx\_in[k].r = w[k] \* left.at(iStart + 1 + k);

cx\_in[k].i = 0; // We have no complex input data

}

#pragma omp barrier

// F(:,n) = fft(w .\* y(iStart+1 : iStart+nFFT));

kiss\_fft(cfg, cx\_in, cx\_out); // Do the FFT

// Copy this data over, in case we need it later???

#pragma omp for

for (int i = 0; i < nFFT; i++)

{

F[n \* nFFT + i].real = cx\_out[i].r;

F[n \* nFFT + i].imag = cx\_out[i].i;

}

#pragma omp barrier

// G(:,n) = 20\*log10(abs(F(1:nFFT/2, n)));

#pragma omp for

for (int i = 0; i < nFFT / 2; i++)

{

G[n \* (nFFT / 2) + i] = 20 \* log10(absComplex(F[n \* nFFT + i]));

}

}

// Compute a Gchecksum

double Gchecksum = 0.0;

for (int i = 0; i < nFFT / 2; i++)

{

#pragma omp for

for (int j = 0; j < nFrames; j++)

{

Gchecksum += G[i \* (nFFT / 2) + j];

}

}

#pragma omp barrier

std::cout << "G checksum: " << Gchecksum << std::endl;

// Free the memory, because this is the land of the free

free(F);

free(w);

free(G);

return 0;

}