Université de Montpellier

Electronique, énergie électrique, automatique- Informatique Robotique

Rapport de HAE710E

Traitement du signal audio et développement d'une interface de visualisation

Nom de l'étudiant: HaoRan He

Nom de l'encadrant: Strauss Olivier, Croitoru Madalina

Année universitaire: 2024-2025

Introduction

Ces dernières années, le traitement des signaux audio est devenu un domaine de recherche et d'application important, de nombreux domaines utilisant cette technique pour améliorer la qualité du son, extraire des informations précieuses ou traiter le son de manière créative. Les techniques de traitement audio telles que les transformées de Fourier, le filtrage et l'analyse des formes d'onde sont fondamentales pour comprendre et traiter les données sonores.

Aperçu

L'objectif de ce projet est de développer un système de traitement audio intégrant Python et C++ pour l'analyse et le traitement des données audio. Le système utilise des techniques telles que la transformée de Fourier rapide (FFT) pour l'analyse des fréquences, les filtres de Butterworth pour le traitement des signaux, et une interface utilisateur graphique (GUI) pour fournir une expérience d'interaction avec les fichiers audio. L'objectif du projet était de concevoir une solution qui ne se contente pas d'effectuer une analyse audio, mais qui examine et analyse également les données audio en détail.

En combinant Python et C++, ce projet vise à fournir des outils aux utilisateurs du traitement simple des signaux audio. Les outils permettent aux utilisateurs de charger des fichiers audio, de visualiser des formes d'onde, d'analyser le contenu en fréquence, d'appliquer des filtres et d'effectuer des transformations de Fourier.

Ce rapport détaille la mise en œuvre du système, en expliquant ses composants principaux, ses algorithmes et l'intégration de Python et de C++. Les sections suivantes se penchent sur la conception technique et la méthodologie du système, ainsi que sur ses applications potentielles.

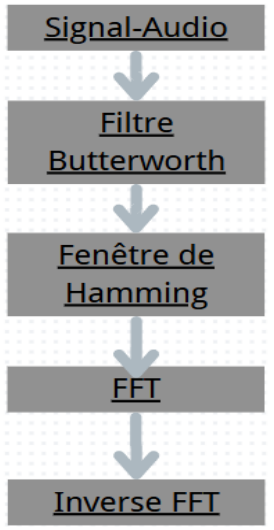
Traitement du Signal Audio

Le traitement des signaux audio est une branche importante dans le domaine du traitement des signaux numériques, qui implique principalement des opérations telles que l'acquisition, l'analyse, l'édition, la synthèse et l'amélioration des signaux sonores.

Diagramme

Dans ce projet, j'ai utilisé la programmation en C++ pour effectuer le traitement des signaux, comprenant notamment la fenêtre de Hamming, le filtre passe-bas Butterworth, la transformation FFT et l'inverse FFT.

Comme indiqué dans le diagramme ci-contre, le programme traite le signal audio en suivant strictement le flux décrit dans le diagramme.



Sélection du Filtre et Application de la Fenêtre

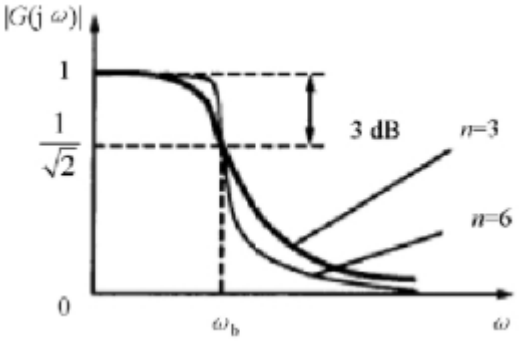
Filtres de Butterworth : La fonction d'un formeur d'onde est d'éliminer les composantes de fréquence non désirées d'un signal et de conserver la partie de fréquence désirée du signal au moyen de caractéristiques de réponse de fréquence spécifiques.

Le choix du filtre Butterworth est motivé par ses caractéristiques suivantes

- Planéité maximale. Les dérivées d'ordre 2n-1 des caractéristiques amplitude-fréquence sont toutes nulles, elles sont donc plates avant la fréquence de coupure, et cette planéité garantit également que la valeur originale du signal n'est pas atténuée par le filtre. L'effet de planéité maximum de la bande passante du filtre passe-bas de Butterworth se traduit par une optimisation plate du gain dans la bande passante.

- La caractéristique amplitude-fréquence est monotone et décroissante, et la caractéristique phase-fréquence est également monotone et décroissante. Le filtre passe-bas de Butterworth est un filtre à tous les pôles, tous les zéros sont à l'infini ; son amplitude diminue de façon monotone lorsque la fréquence augmente, et son amplitude décroît à l'infini.

- Plus l'ordre est grand, plus la réponse amplitude-fréquence est proche du cas idéal.



En résumé, j'ai choisi le filtre Butterworth pour filtrer le signal audio et éliminer le bruit.

Fenêtre de Hamming:

Lorsque l'on utilise la FFT pour mesurer les composantes fréquentielles d'un signal, il est supposé que le signal est périodique et que le domaine temporel et fréquentiel ont une topologie circulaire. Pour un signal non-périodique (non-entier), cette hypothèse peut entraîner des discontinuités aux bords du signal, générant des bruits haute fréquence et provoquant un phénomène appelé fuite de spectre. Pour réduire cet effet, on utilise une technique appelée fenêtrage. Le fenêtrage consiste à multiplier le signal par une fenêtre qui diminue progressivement vers zéro, ce qui permet d'éviter les changements brusques aux bords et d'obtenir un spectre plus lisse.

Il existe maintenant un certain nombre de fonctions de fenêtre parmi lesquelles choisir, et la figure ci-dessous montre les signaux que chacune d'entre elles est adaptée pour résoudre.

Contenu du signal	Fenêtre
Signal sinusoïdal ou combinaison de signaux sinusoïdaux	Hann
Signal sinusoïdal (l'exactitude de l'amplitude est importante)	Profil plat
Signal aléatoire à bande étroite (données de vibration)	Hann
Signal aléatoire à large bande (bruit blanc)	Uniforme
Ondes sinusoïdales rapprochées	Uniforme, Hamming
Signaux d'excitation (marteau d'impact)	Force
Signaux de réponse	Exponentielle
Contenu inconnu	Hann
Signal sinusoïdal ou combinaison de signaux sinusoïdaux	Hann
Signal sinusoïdal (l'exactitude de l'amplitude est importante)	Profil plat
Signal aléatoire à bande étroite (données de vibration)	Hann
Signal aléatoire à large bande (bruit blanc)	Uniforme
Deux tons avec des fréquences proches mais des amplitudes très différentes	Kaiser-Bessel
Deux tons avec des fréquences proches et des amplitudes presque égales	Uniforme
Mesures précises d'amplitude de ton unique	Profil plat

En résumé, j'ai choisi la fenêtre de Hamming pour traiter le signal audio:

- La fenêtre de Hamming réduit les fuites spectrales en lissant les transitions du signal dans le domaine temporel.
- Elle améliore la résolution en fréquence en rendant le volet principal de la réponse spectrale plus étroit et en réduisant rapidement les volets latéraux.
- Elle est couramment utilisée dans l'analyse FFT et la conception de filtres, notamment lorsque les signaux sont limités ou présentent des discontinuités aux limites.

Conception de programme

En ce qui concerne la programmation, j'ai utilisé les bibliothèques suivantes pour garantir le bon fonctionnement du programme.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>
#include <complex>
#include <fftw3.h>
```

- Fournit des fonctions d'entrée et de sortie.
- Fournir une fonction d'opération de fichier
- Fournir des conteneurs de tableaux dynamiques (std::vector), prendre en charge l'expansion automatique de la taille et les opérations courantes sur les tableaux.
- Fournit des fonctions mathématiques telles que cos (cosinus), pow (opération de puissance), sqrt (racine carrée), etc.
- Prise en charge des calculs complexes
- Introduction de fichiers d'en-tête pour la bibliothèque FFTW pour le calcul efficace des transformées de Fourier rapides (FFT) et des transformées inverses (IFFT)

Conception de programme de filtre Butterworth

Un filtre passe-bas de Butterworth d'ordre 3. Ce filtre applique un filtrage au signal d'entrée x en fonction de la fréquence de coupure fc et de la fréquence d'échantillonnage fe,et enregistre le résultat filtré dans y.

```
// Définition de la fonction du filtre Butterworth
int filterButterworthOrder3(double *x, double *y, int N, int fc, int fe) {
    double alpha = M_PI * (double)fc / (double)fe;  // Calcul de la fréquence normalisée
    double A = +1.0 + 2.0 * alpha + 2.0 * alpha * alpha + 1.0 * alpha * alpha * alpha;
    double B = -3.0 - 2.0 * alpha + 2.0 * alpha * alpha + 3.0 * alpha * alpha * alpha;
    double C = +3.0 - 2.0 * alpha - 2.0 * alpha * alpha + 3.0 * alpha * alpha * alpha;
    double D = -1.0 + 2.0 * alpha - 2.0 * alpha * alpha + 1.0 * alpha * alpha * alpha;

    double a0 = 1.0 * alpha * alpha * alpha / A;
    double a1 = 3.0 * alpha * alpha * alpha / A;
    double a2 = 3.0 * alpha * alpha * alpha / A;
    double a3 = 1.0 * alpha * alpha * alpha / A;
    double b1 = -B / A;
    double b2 = -C / A;
    double b3 = -D / A;

    // Application du filtre
    for (int i = 0; i < N; i++) {
        if (i == 0) {
            y[i] = a0 * x[i];
        } else if (i == 1) {
            y[i] = a1 * x[i - 1] + a0 * x[i];
        } else if (i == 2) {
            y[i] = a2 * x[i - 2] + a1 * x[i - 1] + a0 * x[i];
        } else {
            y[i] = a3 * x[i - 3] + a2 * x[i - 2] + a1 * x[i - 1] + a0 * x[i];
        }
    }
```

Conception de programme de la fenêtre de Hamming

Formule de la fenêtre de Hamming:

$$w(i) = 0.54 - 0.46 \cdot cos(2\pi i/(N-1))$$

```cpp
// Application de la fenêtre de Hamming
void applyHammingWindow(std::vector<double>& data) {
    int N = data.size();
    for (int i = 0; i < N; ++i) {
        data[i] *= 0.54 - 0.46 * cos(2 * M_PI * i / (N - 1));
    }
}
```

FFT et Inverse FFT

La transformée de Fourier étant symétrique, je ne conserve dans ce projet que la partie positive du domaine des fréquences (de 0 à la fréquence de Nyquist), la partie de la transformée de Fourier étant symétrique.

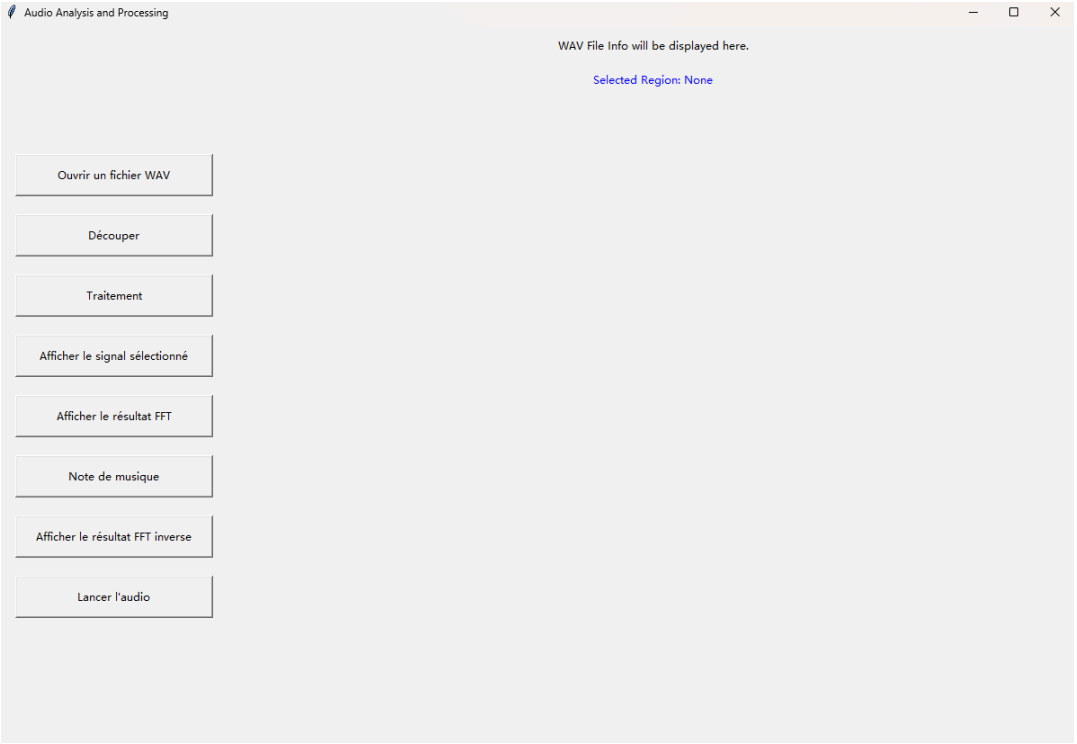
```cpp
// Transformée de Fourier rapide (FFT), conservation uniquement de la partie des fréquences positives
std::vector<std::complex<double>> performFFT(const std::vector<double>& data) {
    int N = data.size();
    std::vector<std::complex<double>> fft_result(N);

    fftw_complex* in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    fftw_complex* out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    fftw_plan plan = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);

    // Remplissage des données d'entrée
    for (int i = 0; i < N; ++i) {
        in[i][0] = data[i];
        in[i][1] = 0.0;
    }

    fftw_execute(plan);  // Exécution de la FFT

    // Conservation uniquement de la partie des fréquences positives
    int half_N = N / 2;
    for (int i = 0; i < half_N; ++i) {
        fft_result[i] = std::complex<double>(out[i][0], out[i][1]);
    }

    fftw_destroy_plan(plan);
    fftw_free(in);
    fftw_free(out);

    return fft_result;
}
```

```cpp
// Transformée de Fourier inverse (Inverse FFT)
std::vector<double> performInverseFFT(const std::vector<std::complex<double>>& fft_data) {
    int N = fft_data.size();
    std::vector<double> result(N);

    fftw_complex* in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    fftw_complex* out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * N);
    fftw_plan plan = fftw_plan_dft_1d(N, in, out, FFTW_BACKWARD, FFTW_ESTIMATE);

    // Remplissage des données d'entrée
    for (int i = 0; i < N; ++i) {
        in[i][0] = fft_data[i].real();
        in[i][1] = fft_data[i].imag();
    }

    fftw_execute(plan);  // Exécution de l'inverse FFT

    // Extraction des résultats
    for (int i = 0; i < N; ++i) {
        result[i] = out[i][0] / N;  // Normalisation
    }

    fftw_destroy_plan(plan);
    fftw_free(in);
    fftw_free(out);

    return result;
}
```

Implémentation d'une interface

Présentation de l'interface

Le programme Python fournit des fonctions interactives via l'interface utilisateur, y compris des opérations visuelles telles que l'ouverture de fichiers audio, le recadrage, l'affichage de formes d'ondes, de spectres FFT et de résultats de traitement, etc. En même temps, il extrait des données audio et des taux d'échantillonnage via la gestion de fichiers pour les utiliser par le programme C++, et lit les fichiers de résultats de traitement pour les afficher. En même temps, il extrait les données audio et la fréquence d'échantillonnage par le biais de la gestion des fichiers pour les utiliser dans le programme C++ et lit les fichiers de résultats de traitement pour les afficher.



L'interface se compose de trois sections :

- la zone des boutons, située sur le côté gauche.
- la zone d'affichage des informations, située en haut.
- la zone d'affichage de la forme d'onde.

Fonctionnalités principales

Le programme est un outil d'analyse et de traitement audio basé sur Tkinter qui intègre le chargement, le traitement, la visualisation et l'interaction avec des programmes externes. Voici un résumé des fonctionnalités et une vue d'ensemble des principaux modules du code.

- Charger un fichier WAV

Charger des données audio et en extraire les informations de base (fréquence d'échantillonnage, nombre total d'échantillons, durée, etc.)

- Sélection de la région audio

Affiche la forme d'onde du domaine temporel pour la région audio sélectionnée et met en évidence la plage sélectionnée.

- Sauvegarde des données audio sélectionnées

Enregistrez le clip audio sélectionné dans un fichier texte (selected_audio_info.txt) au format des données d'amplitude.

- Afficher les signaux sélectionnés

Lire le fichier selected_audio_info.txt enregistré et afficher la forme d'onde dans le domaine temporel du clip sélectionné.

- Visualisation de la transformée de Fourier rapide (FFT)

Chargez le fichier de résultats de la FFT (fft_result.txt) pour afficher le graphique de la fréquence en fonction de l'amplitude.
Permet de cliquer sur une position de fréquence sur le graphique, et une fenêtre contextuelle affiche la valeur de la fréquence à la position cliquée.

- Affiche un tableau de fréquence des notes de musique

Calcule la fréquence de toutes les notes de C0 à C7, affichée dans un tableau déroulant.Le tableau affiche les fréquences dans la plage 16 Hz - 2093 Hz (couvrant la plage des instruments courants).

- FFT inverse (transformée de Fourier inverse)

Lire le fichier de résultats de la transformée de Fourier inverse (inverse fft result.txt) pour afficher la forme d'onde dans le domaine temporel.

- Lancer L'audio

Lecture des données audio traitées.

Stockage des informations audio

J'ai utilisé un programme python pour l'interface et un programme c++ pour le traitement audio principal. J'ai donc fait en sorte qu'il soit possible de relier les deux programmes en sauvegardant les informations audio.

Interface

- selected_audio_info.txt

Ce fichier stocke les données d'amplitude pour des régions sélectionnées extraites de l'audio.

- sample_rate.txt

Enregistrement de la fréquence d'échantillonnage d'un fichier audio

Traitement du signale

- filtered_audio_info.txt

Enregistre les données audio traitées par le filtre de Butterworth et la fenêtre de Hamming.

- fft_result.txt

Préserve le spectre audio après le traitement par transformée de Fourier rapide (FFT)

- inverse_fft_result.txt

Les données audio récupérées à partir des résultats de la transformée de Fourier sont enregistrées.

Détection des note

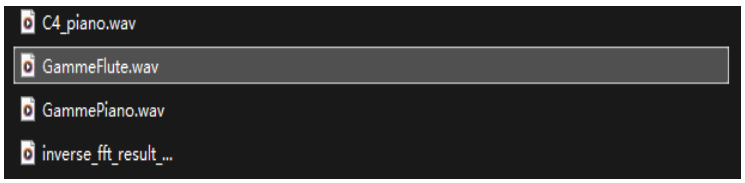
J'ai déjà téléchargé la vidéo sur le traitement audio sur YouTube. Si cela vous intéresse, vous pouvez cliquer sur le lien ci-dessous pour la regarder.

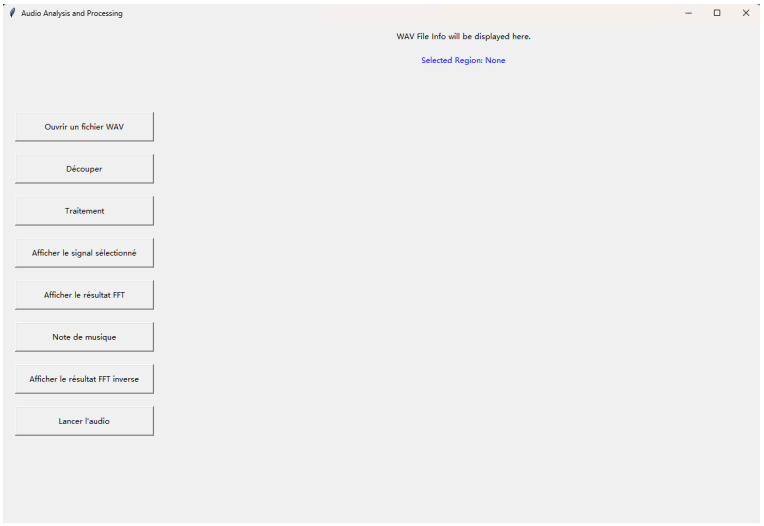
710E traitement du signal

Etape 1

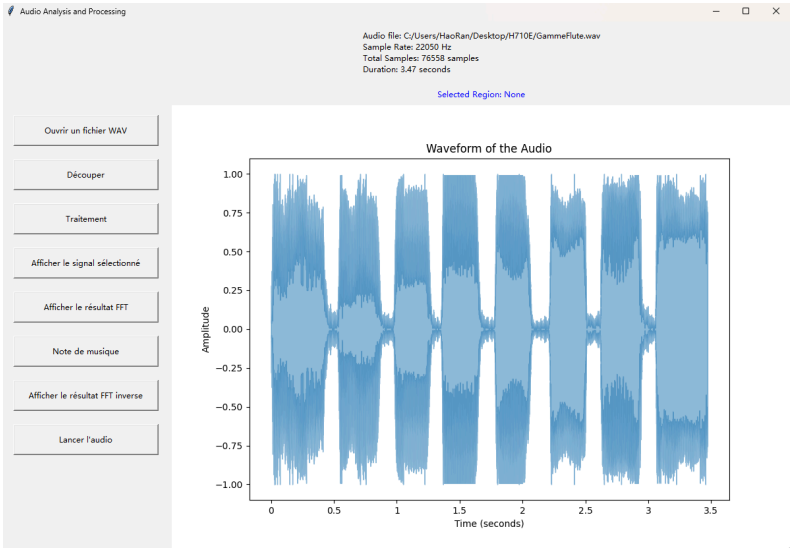
Cliquez sur le bouton « Ouvrir un fichier WAV ».Vous pouvez sélectionner indépendamment les fichiers audio que vous souhaitez analyser.

Ici, je choisis GammeFlute.wav.



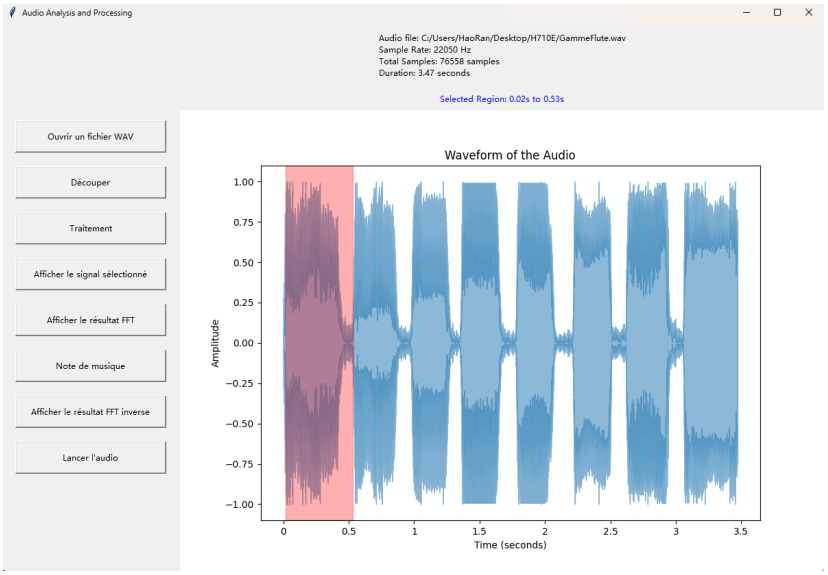


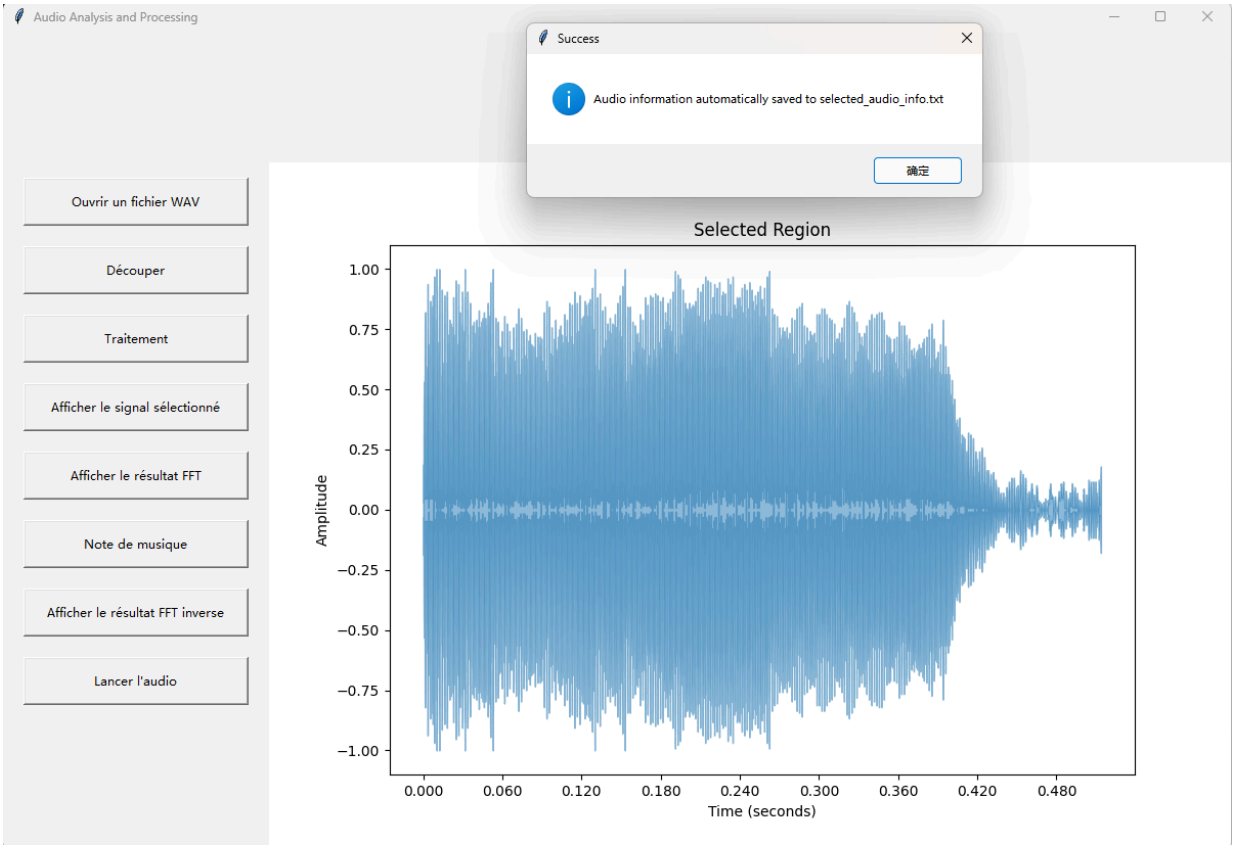
Vous pouvez alors voir des informations sur l'audio en haut, ainsi qu'une image du domaine temporel de l'audio.



Etape 2

Ensuite, cliquez sur le bouton « découper » pour activer le mode découper. Cliquez ensuite sur la zone de l'image que vous souhaitez sélectionner, le point de départ et le point d'arrivée. Confirmez la coupure en cliquant à nouveau sur le découper et enregistrez l'image audio coupée dans le fichier selected_audio_info.txt.



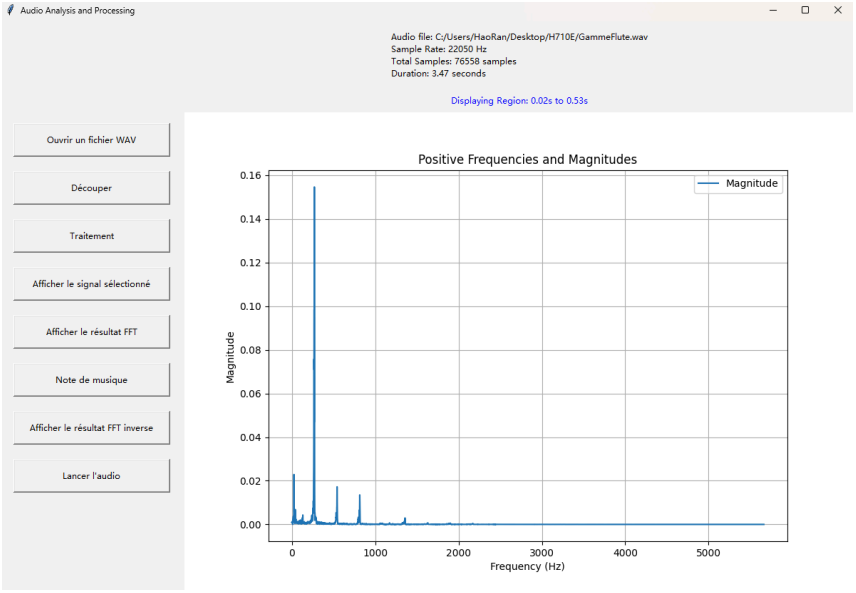


Etape 3

Cliquez sur le bouton « Traitement » pour lancer un programme C++ qui effectue le filtrage, la fenêtre de Hamming, la FFT et la FFT inverse sur le signal sélectionné.

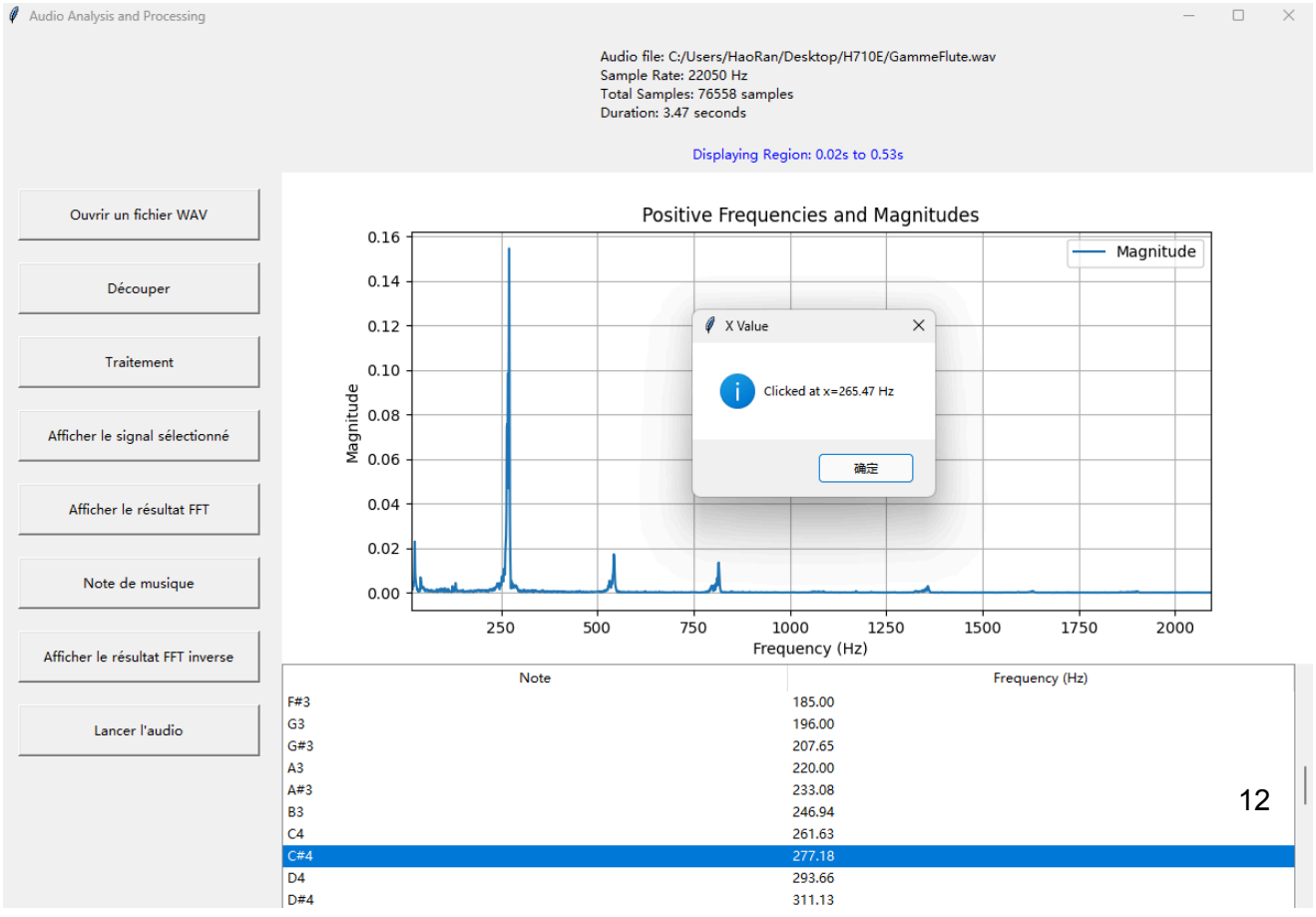
Etape 4

Cliquez sur le bouton « Afficher le résultat FFT » pour afficher l'image FFT du signal.



Etape 5

Cliquez sur le bouton « Note de musique » pour afficher le tableau de fréquence des notes.En cliquant sur l'amplitude qui nous intéresse sur l'image, nous obtenons la fréquence, et avec le tableau des fréquences des notes, nous connaissons la note.



Perspectives

Dans le projet de traitement audio, l'objectif était de traiter de la musique WAV simple et d'être en mesure d'observer les changements dans les notes. Le projet offre une perspective intéressante sur l'analyse des signaux audio, en extrayant les caractéristiques des fichiers audio et en étant capable d'identifier les notes dans la musique. Cela est particulièrement vrai pour le format WAV qui, grâce à sa compression sans perte et à sa haute qualité, conserve une grande quantité d'informations audio qui se prêtent à une analyse détaillée.

Dans le processus de mise en œuvre, l'extraction et l'affichage des notes peuvent être réalisés par des méthodes telles que l'analyse spectrale et la transformée de Fourier. Par exemple, en analysant les composantes de fréquence du signal audio, des caractéristiques telles que la hauteur et la durée des notes peuvent être identifiées et présentées graphiquement. Il s'agit non seulement d'une application fondamentale de la technologie du traitement audio, mais aussi d'un défi dans le domaine du traitement des signaux audio.

En fin de compte, l'objectif du projet n'est pas seulement d'achever l'analyse des signaux audio et l'extraction des notes, mais surtout de pouvoir acquérir une compréhension approfondie des caractéristiques des données audio et de leur potentiel dans les applications pratiques, ce qui, à son tour, jette les bases de tâches de traitement audio plus complexes.

Références

- Comprendre les FFT et le fenêtrage

https://www.ni.com/fr/shop/data-acquisition/measurement-fundamentals/analog-fundamentals/understanding-ffts-and-windowing.html?srsltid=AfmBOopBWhLShDO_YZ-DMjzv7Chnme4HsNM5h0T29aX9Yk_rM9ofcjfG

- Installer la bibliothèque FFTW

https://blog.csdn.net/qq_38967414/article/details/133873064

- Traitement du signal audio

https://www.bilibili.com/video/BV1MW4y1G7FW/?spm_id_from=333.337.search-card.all.click&vd_source=d5c7c76442d3558576900c6c7df964fd

- Interface

https://realpython.com/python-gui-tkinter/