Spectrogram Reading

Floriane Payen
ENSEEIHT, Toulouse - floriane.payen@etu.inp-toulouse.fr

November 20, 2025

Abstract

This paper presents the research conducted on how to enable real-time reading of a visual representation of a sound printed on paper, using an Android phone.

The project is structured around three main steps:
- Generating a 2D representation of sound in the form of an image, which will be printed and read by the phone.
- Developing an algorithm to extract and playback the sound from a photograph of the printed 2D representation.
- Designing an Android application integrating this functionality.

The results are promising: in theory, the method allows for the reconstruction of an almost perfect sound. However, within the application, various external factors (poor lighting, low camera quality, etc.) can introduce errors affecting audio playback. Despite these disturbances, the sound remains intelligible. The application is therefore ready for use.

Keywords: Spectrogram, Phase Retrieval, Tracking, Android Application

Introduction

Sound is a complex signal composed of both an amplitude and a phase. A common way to represent sound as an image is by computing its spectrogram (Figure 1). The spectrogram can be seen as an image where:
- Time flows from left to right.
- Frequencies are arranged from bottom to top: low frequencies (bass sounds) at the bottom and high frequencies (treble sounds) at the top.
- The grayscale level represents sound intensity: from low (black) to high (white).

In theory, it is possible to reconstruct the original sound from a spectrogram by applying the inverse transformation to the one used to generate it. However, this reconstruction is imperfect because the phase of the signal is lost during the conversion from sound to spectrogram. This raises an initial question: How can the lost phase be recovered?

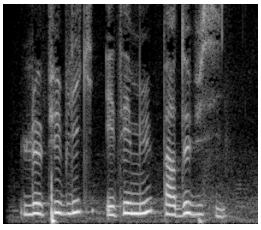


Figure 1: Spectrogram of a human voice

In practice, a photograph of a printed spectrogram (Figure 2) contains not only the spectrogram itself but also its surroundings. Moreover, it may no longer be a perfect rectangle, as it could be distorted by perspective. How can a spectrogram be extracted from an image?

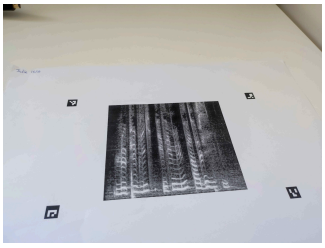


Figure 2: Photography of a spectrogram

Furthermore, one could consider adding a cursor that moves from left to right along the spectrogram to allow sound to be played over time.

From these considerations, the main problem can be defined by : How can a spectrogram be read in real-time using an Android phone?

Spectrogram reading in real time using an Android phone

Spectrogram generation

The first step is to create a spectrogram from an audio file (e.g., .wav or .mp3). To achieve this, a Python script, that applies the following transformations to an audio file, was written :

- A Short-Time Fourier Transform (STFT) with a window size of 1024 and an overlap of 256, creating a matrix S.
- A conversion to the decibel scale: $20 * \log_{10}|S|$.
- A selection of values above −5 dB.
- A scaling of the dB values across all grayscale levels: black corresponds to min S_{dB}, and white corresponds to max S_{dB}.

These transformations ensure a spectrogram with good contrast (Figure 1). They were implemented using two functions from the Librosa library in Python.

Spectrogram reading

Amplitude

To retrieve the amplitude of a sound from its spectrogram, we need to apply the inverse of all the transformations used during spectrogram generation. For practicality in future Android integration and to ensure fast processing, we implemented this part in C++. To perform the inverse STFT, we used the FFTW library [1].

Phase Retrieval

When reconstructing a sound, the amplitude extracted from the spectrogram is not sufficient; the phase is also necessary.

Several methods exist for phase retrieval. In our case, we do not have access to the entire spectrogram, and we need a fast method to approach real-time processing as closely as possible. Based on previous research [2], we identified the phase reconstruction method known as SPSI (Single Pass Spectrogram Inversion) [3] as the most suitable. To use this method, we implemented it in C++.

Spectrogram Detection

To apply the entire process of converting a spectrogram into sound, we first need to extract the spectrogram from the photo taken by the phone. This requires:

- Correcting the perspective to bring the spectrogram back into a proper plane.
- Extracting the portion of the image corresponding to the spectrogram.

To simplify these steps, we placed ArUco markers (Figure 3) along the spectrogram. These markers store for each one a unique identifier. OpenCV [4] provides methods to detect and identify them.

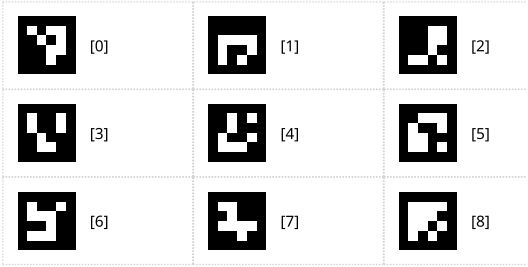


Figure 3: ArUco markers

By detecting the ArUco markers, we know they all belong to the same plane. This allows us to correct the perspective of the entire image, ensuring the spectrogram is properly aligned. We can then extract the portion inside the markers and use contour detection to isolate the spectrogram.

However, detecting ArUco markers in every frame is computationally expensive. Profiling the system, we observed that the ArUco detection was the main bottleneck, significantly slowing down the application.

To improve performance, we implemented tracking to estimate the movement of the image between frames. Tracking is less computationally expensive than repeatedly detecting markers. We first perform an initial ArUco detection, then track the markers in following frames. If tracking fails and there are too few markers to correct the perspective, we perform another detection.

This method greatly improves stability, robustness, and reduces computation time.

Android Application

The final goal of the project was to enable real-time spectrogram reading on a smartphone. To achieve this, we developed an android application using Android Studio. This application integrates the C++ functions described above.

We designed two interfaces:

- The first one (Figure 4) is intended for the end user. It includes a help button and displays only the camera view, with the playback cursor projected onto the spectrogram when possible.
- The second one (Figure 5) was used for debugging purposes. It allowed us to identify the best methods

and parameters, as well as visualize the results of intermediate steps (such as perspective correction, spectrogram extraction, etc.).

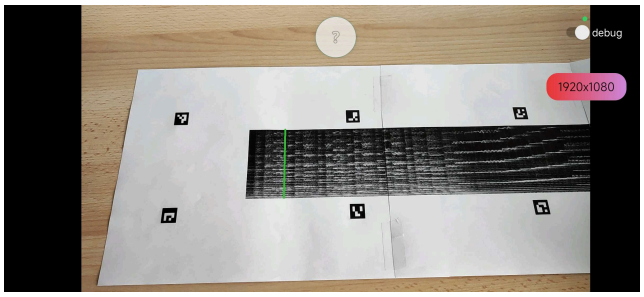
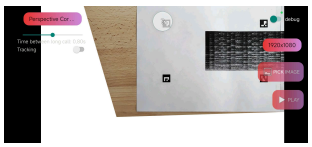
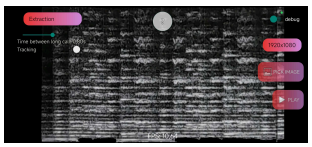


Figure 4: Mobile Application : Interface for classic user.



(a) Perspective Correction.



(b) Spectrogram Extraction.

Figure 5: Mobile Application : Debug Mode.

Conclusion

It is difficult to show the results of our application in a paper, as they are audio. While the quality of the reconstructed sound is not perfect, the results are significantly better than those of the existing Phonopaper application [5], which did not reconstruct the phase.

Moreover, the audio output remains convincing even when the phone is tilted, meaning it does not need to be perfectly aligned with the spectrogram.

To conclude, the application allows to read spectrograms: if the phone moves quickly, the sound plays faster; if it moves slowly, the sound plays slower. Additionally, moving the phone from right to left enables backward playback.

Ackowledgements

We thank the previous research group [2] for their valuable work on spectrogram generation and phase reconstruction. Special thanks to Gilles Azzaro for inspiring this project and meeting us, and to Jean-Denis Durou and Jean Mélou for their guidance.

Bibliography

[1] M. Frigo and S. Johnson, “The Design and Implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.

[2] L. Chatellier, L. Guihur, and G. Le Bihan, “Audioseek Journal de bord,” 2024.

[3] G. T. Beauregard, M. Harish, and L. Wyse, “Single Pass Spectrogram Inversion,” in *2015 IEEE International Conference on Digital Signal Processing (DSP)*, 2015, pp. 427–431.

[4] OpenCV, “OpenCV library.” [Online]. Available: https://opencv.org/

[5] “WarmPlace.ru. PhonoPaper.” Accessed: Mar. 08, 2025. [Online]. Available: https://warmplace.ru/soft/phonopaper

total words : 1098 (= $1000^{+}_{-}10\%$)