Pyc2Sound: a Python tool to convert images into sound

Vincent Bragard, Thomas Pellegrini, Julien Pinquier

Pyc2Sound: a Python tool to convert images into sound

Vincent Bragard
Universite de Toulouse, IRIT
118 Route de Narbonne
31062 Toulouse, France
bragard.upssitech@gmail.com

Thomas Pellegrini
Universite de Toulouse, IRIT
118 Route de Narbonne
31062 Toulouse, France
Thomas.Pellegrini@irit.fr

Julien Pinquier
Universite de Toulouse, IRIT
118 Route de Narbonne
31062 Toulouse, France
julien.pinquier@irit.fr

ABSTRACT

This article reports ongoing work on a user interface dedicated to generate sound from pictures and hand drawings. If we imagine what sound would correspond to a given image, on what parameters of the image do we focus and what would the result sound like? In this paper, we try to answer this question by giving a model transforming images into sound based on chosen parameters extracted from the image. For this, an input image is first binarized, then its skeleton is extracted and 'tracks' are identified and used to generate chirps in an additive synthesis approach.

Keywords

Additive synthesis, skeletonization

Figure 1: Input image

1. INTRODUCTION

A tool capable of transforming an image into sound could have several possible applications. For example, it could serve as an artistic tool, which artists use to *draw* sound; it could also serve as a mean to teach children to correctly form letters and numbers; as a last example, it could allow blind people to hear a sound metaphor of a painting. Here, we propose a first prototype of such a tool. In the domain of sound synthesis, several pieces of software proposing a way to transform images into sound already exist, each with their own approach on the subject. For example, Audiosculpt, created by the IRCAM, filters white noise with an input image [1]. Another software, Photosounder, considers the input image as a magnitude spectrogram and applies an inverse FFT on it to synthesize a sound [2, 6]. The limits of these approaches are that we can only treat the image as a whole instead of being able to treat different objects in the image differently. Another problem is to determine the time domain of the resulting signal, different methods exist, but choosing which one to use is difficult and depends on the type of application we wish to make [5]. A solution some softwares like VOSIS have adopted is to treat a video instead of a still image [4]. The tool presented in this article takes a different approach for which, instead of taking the image directly to synthesize a sound, a mathematical model is used to extract a skeleton of shapes contained in the image with a Voronoï diagram [3]. A graphical user interface will allow the user to generate the sound of an image. The user can also draw on a canvas and the drawing will be the basis for synthesis. In this paper, the three main steps (image skeletonization, track identification and synthesis) will be presented first, then the graphical user interface of the tool will be shown. In order to describe all the processing steps, we will consider the simple image depicted in figure 1.

2. IMAGE SKELETONIZATION

In an image, what we, as humans, perceive first, are the general shapes of it. Moreover, we tend to *read* them from left to right, and we also tend to correlate height in an image to the pitch's height in a sound. Since we want to imitate this kind of perception, images are taken as time-frequency representations, where time is represented on the x axis and frequency on the y axis. In order to generate a sound from these shapes (in our case a handwritten '3'), the shapes' skeleton contained in the image has to be extracted. To do so, the size of the image is doubled so that even if the

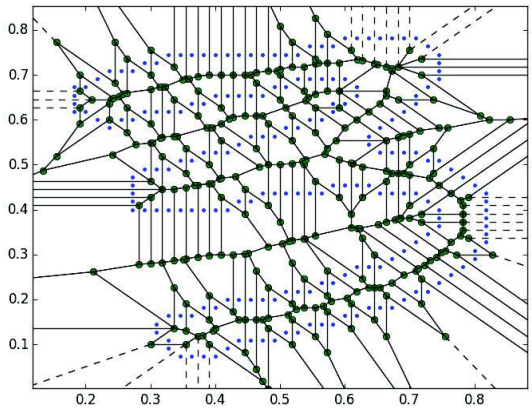

Figure 2: Voronoï diagram

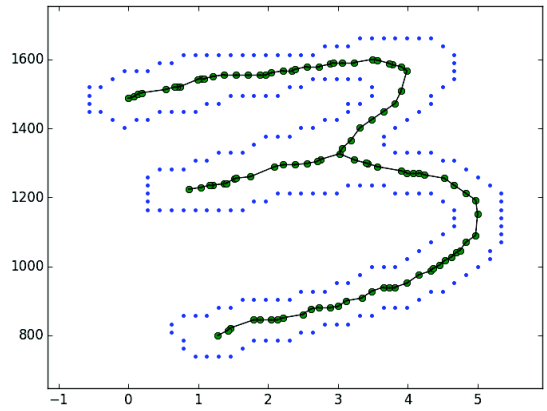

Figure 3: Final skeleton

shapes in it were one-pixel wide lines, they become two-pixel wide and have separated edges on each side. Then, the new image is binarized with a threshold given by the user, in order to precisely identify the shapes. Once it is done, the shapes' edges, i.e. shapes' pixels directly neighboring the background, are isolated and fed to a Voronoï algorithm. It gives the result shown in figure 2.

The Voronoï diagram contains too much information since it contains shapes' skeleton, but also edges in the space between shapes. It needs to be cleaned up. Several steps are needed in order to do so. Firstly, every edges that have one or both of their extremities outside a shape are removed. Secondly, all the vertical edges are 'rewired', as they can not be synthesized due to their infinitesimal time range. To rewire them, the weights of both the extremities of a vertical edge are calculated, the weight of a vertex being the sum of the length of all the edges connected to it. The extremity having the smallest weight is removed and every edge that was connected to it is rerouted to the other extremity of the vertical edge. At this point, there are still few unwanted edges at some places. In order to remove them, successive prunings are done on the skeleton, with a length threshold and a number of successive prunings given by the user. Finally, all the vertices which are not connected to any edges are removed and a clean skeleton is obtained as shown in figure 3. Note that in figure 3, the values of the axes have changed: the skeleton's vertices are now spread from 0 to 5 seconds in the time axis and from 800 to 1600 Hz in the frequency axis. These values are given by the user and were arbitrarily chosen for this example.

3. SEGMENTATION IN TRACKS

Now that the skeleton is properly scaled in the frequency and time domains, each edge can be considered as a linear chirp that will be synthesized.

But first, the different 'tracks' of the skeleton must be separated: since each edge is synthesized separately, we have to be careful of phase alignments at each vertex. A track is defined as a series of connected edges which only go forward in time and with only one simultaneous edge at a time. Figure 4 shows how our example would be split.

To achieve this, all the skeleton's edges are sorted in the time domain, so that the edges starting the soonest are appended first in the resulting list. Then, we apply the following track segmentation algorithm:

```
L <- Sorted list of all the edges
initialize ListTracks as an empty list
while L isn't empty:
  initialize CurTrack as an empty list
  take the first edge of L and put it in
      CurTrack
  while L contains an edge whose starting
      point is the ending point of the last
       edge in CurTrack:
    take that edge from L and put it at the
        end of CurTrack
  save CurTrack in ListTracks
```

After that, *ListTracks* contains the lists of each isolated track.

4. SYNTHESIS

Each track is now correctly isolated, and they are synthesized separately so that the final sound can be generated by additive synthesis. We could have chosen a sample based synthesis approach, where a sample sound is taken then modified, but we have chosen the additive synthesis approach so that we have more control over the synthesized sound. By construction, the edge list of each track is still sorted in the time domain, and the coordinates of its extremities give the starting/ending times and frequencies of the chirp to be synthesized. For each edge, a linear sine chirp is synthesized, then its ending phase is conveyed to the next edge in order for them to be aligned. Once the track is synthesized, a fade-in and a fade-out are applied to it, so that it does not start/end abruptly. The length of the fades are given by the user through the interface. The user can also decide to add harmonics to the synthesized sound and their strength.

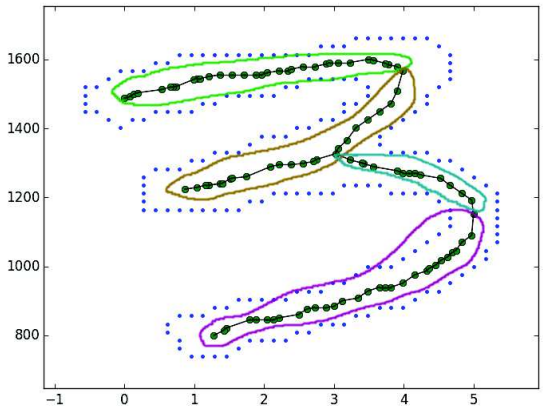

Figure 4: Track identification

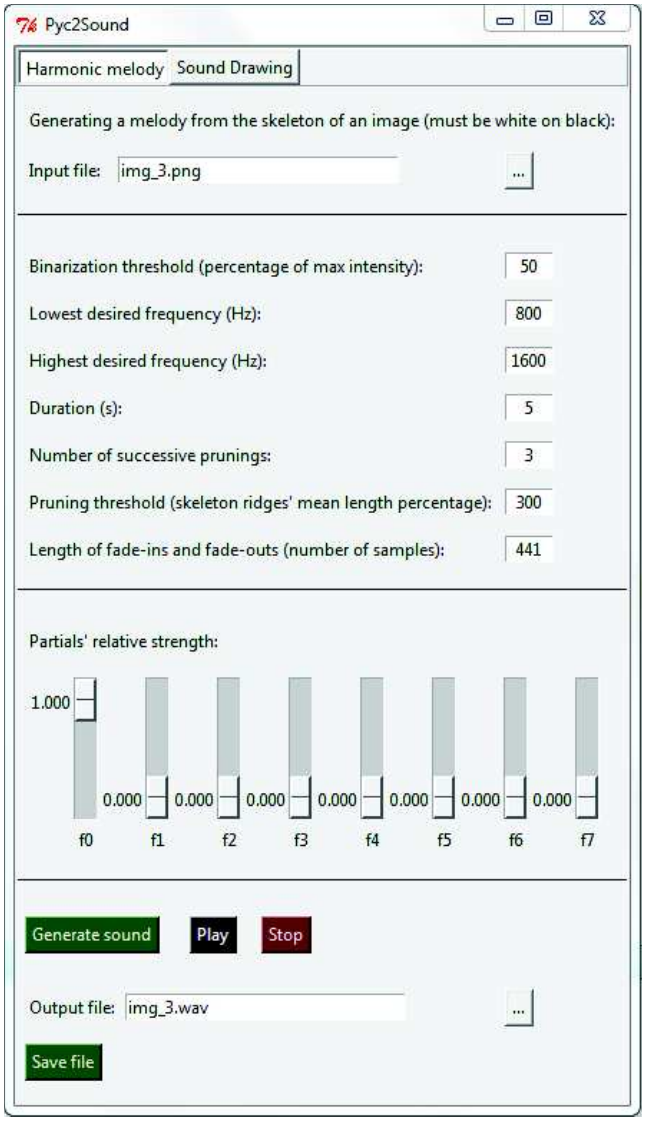

Figure 5: GUI: *Harmonic Melody*

5. USER INTERFACE

The user interface is a simple graphical interface. It contains two utilization modes: *Harmonic melody*, shown in figure 5, and *Sound drawing*, shown in figure 6. *Harmonic melody* allows the user to load an image, and asks for different parameters to generate the corresponding sound, like the binarization threshold, the desired frequency range, the pruning parameters and the length of fades. The user can also add harmonics to the generated sound thanks to sliders. Each time the user generates a sound, the graph of the clean skeleton is shown. It helps the user to tune parameters to his needs. If he is satisfied with the audio result, it can be saved as a file. An illustration of our example is available at `https://goo.gl/QyMuO2`. The second mode, *Sound drawing* is almost identical. However, instead of a text field to generate a sound from an image on the disk, a canvas is displayed, on which the user can draw with his mouse. This canvas supports basic features like undoing/redoing actions or drawing straight lines.

6. CONCLUSIONS

In this paper, a methodology has been proposed, allowing to extract relevant information of objects' shape in the image in order to synthesize a sound representative of these information. The sound itself is generated by additive synthesis, and is composed of different linear sine chirps.

We plan to extend Pyc2Sound's synthesis capabilities. To do this, other pertinent parameters in images will be identified and extracted in order to enrich the complexity of the generated sounds. For example, the signal amplitude will be influenced by the thickness of the shape, and color will influence the harmonics' power. We could also make the sound richer by using the image's spectra as a basis for synthesis.

In order to reach the possible applications we proposed in the introduction, we plan to port the code on more mobile devices, like touchscreen tablets, and improve the graphical user interface, so that it becomes simple and more appealing to use.

Figure 6: GUI: *Sound Drawing*

References

[1] Audiosculpt: a visual and "sculptural" approach to sound manipulation. `http://forumnet.ircam.fr/product/audiosculpt/`.

[2] Photosounder. `http://photosounder.com/`.

[3] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3), 1991.

[4] R. McGee. Vosis: a multi-touch image sonification interface. In *Proc. NIME*, Daejeon, 2013.

[5] W. Seung-Yeo and J. Berger. Application of image sonification methods to music. In *Proc. ICMC*, Barcelona, 2005.

[6] N. Sturmel and L. Daudet. Signal reconstruction from stft magnitude: a state of the art. In *Proc. DAFx*, Paris, 2011.