Online transmission of panoramic audio

SOPA project

April 18, 2013

Contents

1 Introduction 2

2 Purpose of the project 2

3 Principles 4
  3.1 Definition of Temporal HRTF 4
  3.2 Estimation of Temporal HRTF 7
    3.2.1 HRTF database 8
  3.3 Direction of sound-image sources 10
    3.3.1 Method using an equilateral triangle 10
    3.3.2 Method using cardioid microphones 11
  3.4 Miniature Head Simulator 14
  3.5 Panning on demand 16

4 Structure of a SOPA file 17

5 Java source code 19
  5.1 WAV player 20
  5.2 Opening a SOPA file 22
  5.3 Preparation of HRTF database 26
  5.4 Decoding SOPA files and reproducing binaural signals 27

6 Binaural rendering of virtual sound sources 35

7 Concluding remarks 38
1 Introduction

In this document, we describe in detail the Streaming Of Panoramic Audio (SOPA) project together with its objectives. We present the principles of the technology behind SOPA and instructions on how to use SOPA data.

The SOPA project is an open source project which introduces a new audio data format. Each SOPA file acts as a media container for spatial audio information necessary for reconstructing the auditory environment where the audio data were recorded. The reproduction of audio signals stored in SOPA files requires special programs for decoding the data. The source code for such programs is freely available at the SOPA archive page accessible from the project web site (http://staff.aist.go.jp/ashihara-k/pan_top.html).

The panoramic audio demonstration page on the project website contains sample SOPA files. By using stereo headphones, users can listen to binaural signals recorded in a real-world environment in which the aspect of the user’s position can be changed to simulate head movements in the real-world environment during the reproduction. We named this functionality ‘panning on demand.’ This kind of manipulation had been possible only in the case of binaural rendering of virtual sound sources. In conventional methods, once the binaural signals are recorded in a real-world environment, the direction of sounds cannot be controlled during reproduction.

It is widely known that binaural recording of real-world environments requires a dummy head or a HATS (Head And Torso Simulator). However, some of the examples provided on the project website were recorded with a Miniature Head Simulator instead, which is a special device designed as part of this project. An image showing the setup used for recording these samples can be seen below (Fig. 1). Although the Miniature Head Simulator is not necessary for recording SOPA data, it is nevertheless a unique and useful tool, and therefore it will be described below in Section 3.4.

To go directly to the details about how to implement SOPA decoding functionality in programs, skip to Sections 4 and 5. Other sections also provide information necessary for recording SOPA files.

2 Purpose of the project

The purpose of the SOPA project is to devise a method for transmitting the spatial audio information of a real-world environment over the Internet in the form of relatively small files with a low bit rate. By using SOPA file, authors of audio and audiovisual content can create and control the directional aspects of the acoustic panorama, which will allow a client (listener)
of such audio and audiovisual content to experience the sensation of being surrounded by sound sources and being present at the actual site where the audio signals were recorded.

In real-world environments, there is no limit to the number of sound sources, and therefore it seems to be difficult to encode the spatial audio information into a small amount of data. But experience shows that, even with only two ears for capturing acoustic signals, people can reconstruct the spatial distribution of sound sources.

In the following sections, we will show that the spatial audio information transmitted as a file whose size is comparable with that of a conventional stereo WAV file is practically sufficient for conveying the spatial distribution of sound sources.

As mentioned above, the size of a SOPA file is comparable to that of a stereo WAV file. However, a SOPA file contains not only PCM data stream but also spatial (directional) information about the sounds. SOPA files can therefore be used as media containers for acoustic signals with directional information, which can be streamed online. The techniques proposed in this project can be used in the presentation of music and other artistic performances, entertainments and recreational events, mobile communication,
remote sensing and monitoring, sound source detection, and various other fields.

This project includes the SOPA file (data) format, the software necessary for decoding such data and the software for binaural rendering of virtual sound sources. Details on recording SOPA data are not included as recording requires not only software but also hardware devices (such as a microphone system), which cannot be distributed freely.

However, this document outlines methods for recording SOPA data (Sections 3.3.1 and 3.3.2), and we hope that this information will aid the user in understanding the data structure.

3 Principles

3.1 Definition of Temporal HRTF

As the basic principles of binaural recording, dummy head recording, Head-Related Transfer Function (HRTF) and Head-Related Impulse Response (HRIR) can be found in a number of books and online resources, we can provide a short description of Temporal HRTF.

We define Temporal HRTF as the time-variant spectral difference between the reference signal and the signals in the ear canals of the listener. The reference signal is the acoustic signal recorded with a single omnidirectional microphone placed at a certain position (reference point) in the real-world environment. The signals in the ear canals are the acoustic signals captured...
inside the listener’s ears when the point in the center of the line connecting the ears is at the reference point. This is illustrated in Fig. 2, where A and B in the figure represent the reference signal and the signals in the ear canals, respectively. As its name indicates, the Temporal HRTF is a time-dependent function.

According to the definition presented above, it can be easily deduced that binaural signals can be obtained by spectral superimposition of the Temporal HRTF to the reference signal, as illustrated in Fig. 3.

In a real-world environment, however, it is impossible to capture the reference signal and the signals in the ear canals simultaneously as the former represents the sound captured at the reference point in the absence of a user, while the latter is the sound captured in the ear canals of the user. Since it is not measurable, Temporal HRTF can be estimated only indirectly.

The real-world environment can theoretically contain an infinite number of sound sources. The sound perceived by a listener is a mixture of different sounds arriving from various sources. Even a single frequency component of the sound perceived by the listener might be a mixture of several independent components from different sources. It is almost impossible to know directions
of all the sound sources, and it is often impossible to estimate even the exact number of the sound sources in our environment.

Here, we assume that even if the same frequency components originate from different directions, they can be substituted with a single sound-image source as long as its direction is carefully determined so that it can satisfy the actual inter-aural phase difference and inter-aural level difference between the signals captured by the left and right ears of the listener. If these differences as produced by a single sound-image source are perceptually equivalent to those produced by the actual sources, the listener will be unable to distinguish between the sound-image source and the real sources.

This means that if the sound-image source of each frequency component can be aligned in an appropriate direction, we can estimate the Temporal HRTF, which can be used to provide the listener with the sensation of being at the position from where the sounds were recorded.

An outline of the decoding process of SOPA data is shown in Fig. 4, where the program performs the following operations.

- FFT of the reference signal
- Estimation of the Temporal HRTF by using the directional data and the HRTF database encoded in the file
- Spectral superimposition of the Temporal HRTF onto the reference signal
Fig. 5: Angles corresponding to the direction of the sound image
A top view of the horizontal plane around the reference point. 360° of the horizontal plane are divided into 72 directions numbered from 1 to 72.

- Inverse FFT of the signal

Since the FFT and inverse FFT procedures are trivial, we describe the process of estimating the Temporal HRTF in the next section.

3.2 Estimation of Temporal HRTF

Although Temporal HRTF can be estimated in a number of different ways, we describe one method which we use in the present version of our programs. Temporal HRTF is estimated by using an HRTF database consisting of 72 subsets. Dividing the 360° of the horizontal plane into 5° yields 72 directions, as shown in Fig. 5, and each subset of database contains the HRTF data for the corresponding direction. HRTF data consist of the level (amplitude) values and phase values represented as a function of frequency. For example, the subset of θ (in units of degrees) contains the HRTF data of θ consisting of the level and phase values for the frequencies from $f_0$ to $f_{n-1}$ (in units of Hz).

Once a single direction is assigned to the sound-image source of each frequency, the level and phase values of the corresponding frequency are read from the corresponding subset of the database. If, for instance, the direction
of the sound-image source of $f_x$ [Hz] is $\theta_y$ [degree], the level and phase values for $f_x$ [Hz] are read from the subset corresponding to $\theta_y$.

The Temporal HRTF can be obtained by arranging the level and phase values read from the HRTF database in order of increasing frequency. The Temporal HRTFs of the left and right ears must be estimated independently, and the corresponding binaural signals can be given by superimposing the estimated Temporal HRTFs onto the spectrum of the reference signal.

### 3.2.1 HRTF database

The HRTF databases in the project are the binary files named ‘hrtf512.bin’ and ‘phase512.bin.’ The former contains the amplitude spectra and the latter contains the phase spectra of the HRTF. Each of these files consists of a
binary data stream of 16-bit integers based on HRIR data recorded at a sampling rate of 44100 Hz.

Fig. 6 shows the data stream in ‘hrtf512.bin.’ As can be seen in the figure, the file contains 72 subsets of the data, each of which corresponds to one of 72 directions. Each subset contains 512 sequences of data, each of which corresponds to a frequency from $f_0$ to $f_{n-1}$ [Hz]. Fig. 7 shows one of these subsets. The subset is the HRTF amplitude spectrum for a particular angle represented in a linear scale and multiplied by 2048 to fit in the 16-bit scale. Since it is based on the data sampled at 44100 Hz, the horizontal axis represents the frequencies between 0 and 44100 Hz.

‘Phase512.bin’ also consists of 72 subsets of data, as shown in Fig. 8. The phase values range from $-\pi$ to $\pi$ (in units of radians). To fit them into the 16-bit linear scale, the values are multiplied by 10000. Therefore, an angle
of $\pi$ rad is represented by 31415 instead of 3.1415. Fig. 9 shows a single subset, where 256 on the X-axis corresponds to the Nyquist frequency, which is 22050 Hz.

In these databases, 16-bit integers are arranged in big-endian order. In the present version of our programs, HRTF data originally provided by MIT Media Lab are used.

### 3.3 Direction of sound-image sources

Temporal HRTF is estimated by using directional data in the SOPA file. Each value in the directional data represents the direction of a sound-image source for each frequency component. The question is how to determine the direction of sound-image sources.

Although not included in the project, in this section we describe two methods for determining the directions of sound-image sources. The information in this section is not necessary for understanding how to reproduce SOPA data as it only provides guidance for recording SOPA files.

The direction of a sound-image source must be determined for every frequency from $f_0$ to $f_{n-1}$. While we introduce methods that have worked well in our programs, there may be other efficient algorithms, and programmers can invent their own methods.

#### 3.3.1 Method using an equilateral triangle

One of the methods is to use at least three microphones. The Tohoku Institute of Technology and the National Institute of Advanced Industrial Science and Technology have applied for a patent on this method in Japan. Although we describe the method in the following paragraphs, this does not mean that the method may be used for free.

If we place two microphones at a certain distance from each other in the horizontal plane, we can obtain the inter-channel phase difference and the inter-channel level difference, of which the former is much more useful for estimating the direction of the sound-image source since it is less affected by the distance between the sound sources and the microphones, while the latter depends not only on the directions, but also on the distances of the sound sources in relation to the microphones.

Since the speed of sound in the air is known, we can convert the inter-channel phase difference into a difference in distance between the source and the microphones. As the microphones are on the same horizontal plane, the difference of distance depends on the direction of the source. When the source is in front of a microphone (the angle of the source is 0 rad) or behind
Three microphones are placed at the vertices of an equilateral triangle in the horizontal plane. The phase difference grows larger as the angle between the source and the microphones approaches $\pm \frac{\pi}{2}$ rad. Therefore, inter-channel phase difference can be used to estimate how close the angle of the direction of the source is to $\pm \frac{\pi}{2}$ rad. However, it is not sufficient for determining whether the source is in the front or behind the microphones. It is impossible, for example, to determine whether the angle of the source direction is $\frac{3\pi}{4}$ rad or $\frac{5\pi}{4}$ rad.

Three microphones (a, b, c) placed at the vertices of an equilateral triangle in the horizontal plane as shown in Fig. 10 define three pairs of microphones (a and b, b and c, and c and a), which can be used to unambiguously determine the direction to the source. This simple strategy is actually used in the Miniature Head Simulator (Fig. 1).

### 3.3.2 Method using cardioid microphones

Another strategy for determining the direction of a sound-image source involves the use of cardioid microphones. We have not applied for a patent on this method, and we are unaware of whether any other research groups have applied for such a patent.

The reason why two microphones are not enough to uniquely determine the direction of a sound-image source is illustrated in Fig. 11. The upper row...
Fig. 11: Problem encountered when determining the direction of the source

The inter-channel phase difference is shown in the upper column as a function of the direction of the source. Since sound from directions ‘a’ and ‘b’ in the lower column yields the same inter-channel phase and level difference, it is impossible to distinguish between directions ‘a’ and ‘b’.

in this figure shows the inter-channel phase difference as a function of the direction of the sound image. A top view of the two microphones can be seen at the bottom of the figure. When the observed phase difference is $\phi$, the direction of the sound image can be either ‘a’ or ‘b’ in the figure. Since both of ‘a’ and ‘b’ produce the same inter-channel level difference, the direction cannot be determined unequivocally.

If the inter-channel level difference and the inter-channel phase difference are out of phase by $\frac{\pi}{2}$ rad, as shown in the upper panel of Fig. 12, the sound images corresponding to directions ‘a’ and ‘b’ produce the same inter-channel phase difference, but the inter-channel level difference for direction ‘a’ is opposite that of ‘b.’ We can, therefore easily determine whether the
If the inter-channel level difference and the inter-channel phase difference are out of phase by $\frac{\pi}{2}$, sounds from the directions ‘a’ and ‘b’ share the same inter-channel phase difference but exhibit mismatching inter-channel level differences.

The characteristics shown in the upper panel of Fig. 12 can be obtained by using two cardioid microphones, and a schematic illustration of this setup is shown in Fig. 13. In this case, the sensitivity of the microphone on the left reaches its maximum at $0^\circ$ and its minimum at $180^\circ$, and vice versa for the microphone on the right. The distance between the microphones may be several centimeters. If sound sources are close to the microphones, the distance between the microphones affects the inter-channel level difference and the estimation of the direction of sound image is generally less accurate. When the sound sources are far enough (farther than 100 cm, for example),
Fig. 13: Arrangement of cardioid microphones

Two cardioid microphones, where maximum sensitivity is reached at $0^\circ$ for one microphone and at $180^\circ$ for the other, can be used for recording SOPA files.

inter-channel level difference caused by the small distance of several centimeters between the microphones becomes negligible. In such cases, we can estimate how close the angle of the direction to the source is to $\pm \frac{\pi}{2}$ rad from the inter-channel phase difference and whether the source is in the front or behind the microphones from the inter-channel level difference.

This strategy, however, has a potential problem. When components of the same frequency come from different directions, one from the front another from the back, for example, the left microphone in Fig. 13 catches the component from the front while the right microphone catches the component from the back. In such a case, measuring the phase difference between these microphones does not make sense. If there is a solution to this problem, the strategy can be used for the Miniature Head Simulator.

Whatever the method used, a SOPA file can be prepared if the direction of the sound image for each frequency can be exclusively determined.

### 3.4 Miniature Head Simulator

Sample SOPA files are provided at the panoramic audio demonstration page. Some of these files were recorded by using a Miniature Head Simulator. The Miniature Head Simulator is a system consisting of a microphone system and a signal processor. An outline of the microphone system is shown in
Fig. 14: HATS and Miniature Head Simulator

A Miniature Head Simulator (right) is compared to the head of a HATS (left).

Fig. 1, where three electret condenser microphones are placed at the vertices of an equilateral triangle in the horizontal plane. The sides of the triangle are 30 mm.

These microphones must be uniform in characteristics. One of the microphones is expediently defined as a reference microphone and the others are defined as comparison microphones.

The direction of the sound-image source can be determined for each frequency by using the inter-channel phase difference between the reference microphone and the comparison microphones, and the direction is given in the form of an angle in units of radians. Next, each angle is matched to one of the 72 ranges of angles displayed in Fig. 5, and the results are subsequently written to the SOPA file. The audio signal recorded by the reference microphone (reference signal) is also written to the SOPA file.

In the process of reproduction of the data, the Temporal HRTF is estimated from the directional information contained in the SOPA file and is superimposed onto the spectrum of the reference signal. Since the Temporal HRTF is a time-variant function, the sequence must be processed frame by frame.

Conventional dummy heads have been used exclusively for the research purposes due to their size and weight. As can be seen in Fig. 14, the Miniature
Head Simulator is far smaller than a conventional dummy head, which allows it to be used in various environments, even in narrow ducts or other confined areas, where the standard dummy head cannot be used.

Although it is not necessary for recording SOPA files, the Miniature Head Simulator is a powerful tool for recording spatial audio data. The arrangement of microphones shown in Fig. 10 is one of a number of possible arrangements. Spatial audio data captured by the Miniature Head Simulator with three microphones is two-dimensional (horizontal). Three-dimensional spatial audio data can be captured by using four microphones placed at the vertices of a tetrahedron, although this requires considerably more complex calculations and a three-dimensional HRTF database.

3.5 Panning on demand

Conventional stereo audio formats work well for transmitting binaural audio signals over the Internet. So why do we recommend the SOPA format? Transmission of SOPA data has a unique advantage in terms of interactive manipulation of the sound, namely ‘panning on demand.’

When binaural signals are recorded with a dummy head and reproduced through headphones, the listener experiences the sound as if it originates from the virtual source. In this case, the direction of the virtual source has been determined in advance by the relative direction of the sound source from the dummy head. For example, if the direction of the actual sound source is $45^\circ$ to the dummy head, the listener will perceive the sound as arriving from the same direction, and that direction will not be adjustable.

This problem arises since in the case of a dummy head the front ($0^\circ$) and back ($180^\circ$) directions are physically determined in addition to the left and right directions. In the case of the Miniature Head Simulator, the direction correspond to $0^\circ$ is not physically determined but can be arbitrary determined in the process of estimating the Temporal HRTF.

In the real-world environments, a listener will frequently turn their head in various situations, for instance, when they wish to face a talker directly or to improve the detectability of an audible signal. SOPA brings the same functionality to online communication. Panning on demand is available on the panoramic audio demonstration page. In our programs, the Temporal HRTF is estimated immediately before the binaural signal is generated, which allows for the panning to be changed with a small delay.

Panning on demand can make applications appear more realistic and interactive when using binaural reproduction of sound. This is why we recommend the SOPA format.
4 Structure of a SOPA file

The structure of a SOPA file must be clearly specified to allow programmers to produce original applications capable of reproducing SOPA data. The structure of a SOPA file is similar to that of a stereo WAV file, and the structure of a 16-bit stereo WAV file is schematically presented in Fig. 15. It begins with a header followed by a stream of 16-bit PCM data. The data are stored in temporal order, and since stereo files contain two channels, the arrangement of data represents the left and right channels in alternating order.

The header of a SOPA file is mostly identical to that of a stereo WAV file. A SOPA file contains a stream of PCM data and directional information, which are arranged in the way illustrated in Fig. 16. The audio data stream in this figure is in the form of 16-bit PCM data arranged in temporal order.

N at the top of the figure represents the size of the frame (remember that estimation of the Temporal HRTF is a frame-by-frame process). Frames are also shown at the bottom of the figure. For example, the 0th frame consists of the PCM data from sample 0 to sample $N - 1$, and the next frame starts from sample $\frac{N}{4}$ as frames overlap by $\frac{3}{4}$ of their size.

Directional data are in the form of 8-bit integers, each of which represents the direction of the corresponding frequency as an integer between 1 and 72, inclusive ($f_0$ to $f_{\frac{N}{2} - 1}$ in the figure represent the frequencies). Note that directional data for frequencies above $f_{\frac{N}{2}}$ are not written to the file since the components above $f_{\frac{N}{2}}$ are aliases of the components below $f_{\frac{N}{2} - 1}$. Since the direction of a component is expected to be the same as that of its alias, it is not specified in the SOPA file (for example, the direction of frequency...
Fig. 16: Structure of a SOPA file

The header is followed by PCM data and directional information.

component $f_{N-1}$ should be the same as that of $f_1$. As a result, directional data and audio data for each frame start together but do not end together.

Since the direction of the sound-image source is matched to one of the 72 directions shown in Fig. 5, it can be written in the file as an 8-bit integer. However, since we use only 72 out of the 256 available integers, we can look forward to the extension of the format to support three-dimensional directional information without increasing the data size.

Note that the range of integers representing directions is between 1 and 72 instead of between 0 and 71. We have reserved ‘0’ for the particular purpose of letting the program know the size of the frame since the size of the frame is not specified in the header of the SOPA file. In the case of the $f_0$ (dc) component, the direction of the sound-image source has no importance. The addresses corresponding to the $f_0$ component (areas shown in yellow in the figure) can contain any integer, and we put 0s there. Since it is not used for any other frequencies, 0s appear only at addresses corresponding to $f_0$ and can be used as markers that indicate the beginning of the frames. By counting the number of addresses between the first ‘0’ and the next ‘0,’ the program automatically extracts the size of the frame ($N$).

After opening a SOPA file, the program reads the data stream. Then the program estimates the Temporal HRTF of the frame by using the directional data. The spectrum of the corresponding frame is obtained by FFT, and
after adding the Temporal HRTF to the spectrum, it is transformed back into
the time domain by inverse FFT. The resulting wave is then added to the
previous frame by the overlap-add method. This sequence is repeated frame
by frame until the data stream ends and the binaural signals are generated.

As an example, the header and the first several data sequences in a SOPA
file are shown in Fig. 17. The integer at the offset of 0x16 (4’ in this figure) is
the overlap factor representing the number of overlapping frames (this value
should be either 2 or 4). The data stream begins at offset of 0x2C, and $f_0,
1$ $\cdots$ in the figure represent the frequencies. The first data sequence in the
stream, (18’ in this example) indicates that the direction of a sound-image
source of the component at $f_1$ Hz is $85^\circ$, and $S_0, S_1, \cdots$ represent audio
(PCM) data.

\section{Java source code}

We have written a Java applet capable of streaming spatial audio data online.
The executable applet (version 1.0.0) is provided on the \textit{panoramic audio
demonstration page}, and the corresponding JAR (Java ARchive) files are
provided on the \textit{SOPA archive page}. In this section, we present the source
code of a simpler program, named SimpleSOPAPlayer, which simply decodes
the SOPA file and reproduces the binaural signals. Although panning control
is not available in this program, it demonstrates the reproduction of binaural
signals from SOPA files.

The following files are needed to reproduce sound with SimpleSOPAPlayer.

- A SOPA file (*.sopa)
- The FFT class file (fft.class)
- The HRTF level database (hrtf512.bin)
- The HRTF phase database (phase512.bin)

To execute the program, enter the following command in a command line (terminal) and press ‘Enter’ or ‘Return’.

```java
> java SimpleSOPAPlayer *.sopa
```

Here, ‘*.sopa’ represents the name of a SOPA file to be reproduced.

## 5.1 WAV player

Before presenting the source code of SimpleSOPAPlayer.java, we introduce ‘WAVPlayer.java’ which simply plays WAV files. The program source code is as follows:

```java
import java.io.File;
import java.io.IOException;
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.SourceDataLine;

public class WAVPlayer
{
    private static final int EXTERNAL_BUFFER_SIZE = 131072;
    public static void main(String[] args)
    {
        if (args.length != 1)
        {
            System.out.println("WAVPlayer: usage: java WAVPlayer <WAV file>");
        }
    }
}
```
String strFilename = args[0];
File soundFile = new File(strFilename);
AudioInputStream audioInputStream = null;
byte[] abData = new byte[EXTERNAL_BUFFER_SIZE];
int nBytesRead = 0;
int nBytesWritten = 0;
try {
    audioInputStream = AudioSystem.getAudioInputStream(soundFile);
}
catch (Exception e) {
    e.printStackTrace();
    System.exit(1);
}
AudioFormat audioFormat = audioInputStream.getFormat();
SourceDataLine line = null;
DataLine.Info info = new DataLine.Info(SourceDataLine.class, audioFormat);
try {
    line = (SourceDataLine) AudioSystem.getLine(info);
    line.open(audioFormat);
}
catch (LineUnavailableException e) {
    e.printStackTrace();
    System.exit(1);
}
line.start();
while (nBytesRead != -1) {
    try {
        nBytesRead = audioInputStream.read(abData, 0, abData.length);
    } catch (IOException e) {
        e.printStackTrace();
    }
    if (nBytesRead > 0) {
    
*/
After calling the method ‘line.start,’ the data are read into the array ‘abData[ ]’ in a loop, and additional code can be added to modify ’abData[ ]’ before calling the method ‘line.write().’

5.2 Opening a SOPA file

While the source code provided above is rather simple, it is more complex in the case of a SOPA file. Since the SOPA format is not supported by the Java sound API, ‘AudioInputStream’ method cannot be used for reading SOPA files. Instead of using ‘AudioInputStream,’ the following code is inserted into SimpleSOPAPlayer.java.

```
int[] nByte = new int[4];
int[] nFmt = new int[3];
int nTerm0[] = {82,73,70,70}; // RIFF
int nTerm1[] = {83,79,80,65}; // SOPA
int nTerm2[] = {102,109,116}; // fmt
int nCnt;
String strFilename = args[0];
File soundFile = new File(strFilename);
FileInputStream inStream = null;
AudioFormat audioFormat = null;

try
{
```
inStream = new FileInputStream(soundFile);
for(nCnt = 0;nCnt < 4;nCnt++)
{
    nByte[nCnt] = inStream.read();
    if(nByte[nCnt] == -1)
    {
        inStream.close();
        System.exit(1);
    }
}
if(!Arrays.equals(nByte,nTerm0))
{
    System.out.println("File format error! This is not RIFF");
    inStream.close();
    System.exit(1);
}
System.out.println("RIFF OK");
for(nCnt = 0;nCnt < 4;nCnt++)
{
    nByte[nCnt] = inStream.read();
    if(nByte[nCnt] == -1)
    {
        inStream.close();
        System.exit(1);
    }
}
for(nCnt = 0;nCnt < 4;nCnt++)
{
    nByte[nCnt] = inStream.read();
    if(nByte[nCnt] == -1)
    {
        inStream.close();
        System.exit(1);
    }
}
if(!Arrays.equals(nByte,nTerm1))
{
    System.out.println("File format error! This is not SOPA");
    inStream.close();
    System.exit(1);
}
System.out.println("SOPA OK");
for(nCnt = 0; nCnt < 3; nCnt ++)
{
    nFmt[nCnt] = inStream.read();
    if(nFmt[nCnt] == -1)
    {
        inStream.close();
        System.exit(1);
    }
}
if(!Arrays.equals(nFmt, nTerm2))
{
    System.out.println("File format error! Fmt chunk not found");
    inStream.close();
    System.exit(1);
}
System.out.println("fmt OK");
inStream.read();
int nBit = inStream.read();
if(nBit != 16)
{
    System.out.println("Data are not 16-bit!");
    inStream.close();
    System.exit(1);
}
for(nCnt = 0; nCnt < 3; nCnt ++)
{
    inStream.read();
    if(inStream.read() != 1) { 
        System.out.println("Data are not PCM!");
        inStream.close();
        System.exit(1);
    }
}
inStream.read();
int nOverlap = inStream.read();  // How many frames overlap
if(nOverlap != 2 &amp;&amp; nOverlap != 4)
{
    System.out.println("Wrong value (nOverlap)!");
    inStream.close();
    System.exit(1);
int nSampleRate = inStream.read();
nSampleRate += inStream.read() * 256;
for(nCnt = 0;nCnt < 10;nCnt ++)
{
    inStream.read();
}
for(nCnt = 0;nCnt < 4;nCnt ++)
{
    nByte[nCnt] = inStream.read();
    if(nByte[nCnt] == -1)
    {
        inStream.close();
        System.exit(1);
    }
}
}
catch (Exception e)
{
    e.printStackTrace();
    System.exit(1);
}
int nChannels = 2; // binaural audio is stereo
audioFormat = new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,nSampleRate,nBit,nChannels,
nChannels × nBit / 8,nSampleRate,false);
SourceDataLine line = null;
DataLine.Info info = new DataLine.Info(SourceDataLine.class,audioFormat);
try
{
    line = (SourceDataLine) AudioSystem.getLine(info);
    line.open(audioFormat);
}
catch (LineUnavailableException e)
{
    e.printStackTrace();
    System.exit(1);
}
line.start();
/**************************************************************/
5.3 Preparation of HRTF database

After calling the method line.start, the data stream in the SOPA file must be decoded and converted into a binaural audio data stream. Before this, the HRTF database must be prepared in the following source code.

```java
int iNum = 0;
short[] sHrtf = new short[36864];
short[] sPhase = new short[36864];
DataInputStream din = null;

// hrtf512.bin and phase512.bin have to be in the same directory
// as SimpleSOPAPlayer.class

try {
    din = new DataInputStream(new FileInputStream("hrtf512.bin"));
    for (iNum = 0; iNum < 36864; iNum++) {
        // BigEndian to LittleEndian
        sHrtf[iNum] = (short)((din.readByte() & 0xff) + (din.readByte() << 8));
    }
    din.close();
} catch (Exception e) {
    System.out.println("HRTF data error! : " + e);
    System.exit(1);
}

try {
    din = new DataInputStream(new FileInputStream("phase512.bin"));
    for (iNum = 0; iNum < 36864; iNum++) {
        // BigEndian to LittleEndian
        sPhase[iNum] = (short)((din.readByte() & 0xff) + (din.readByte() << 8));
    }
    din.close();
} catch (Exception e) {

```
Since ‘hrtf512.bin’ and ‘phase512.bin’ contain 72 subsets of data and each subset contains 512 data sequences, each of the arrays sHrtf[] and sPhase[] contains $512 \times 72 = 36864$ data sequences. These arrays are used in the estimation of the Temporal HRTF.

5.4 Decoding SOPA files and reproducing binaural signals

Here, we outline the procedure for reproducing binaural signals from the data stream.

```java
final int iFIN=16384;
final int iByte=nBit/8;

// initialize the values for variables
// the initial values are dummy

int nRatio=44100/nSampleRate;
int nSize=2048; // FFT window size
int nProc=nSize / nOverlap; // frame increment
int nRem=nSize-nProc;
int nBytesRead=0;
int nBytesWritten=0;

// prepare the data arrays

byte[] abData=new byte[EXTERNAL_BUFFER_SIZE]; // data array
byte[] bRet=new byte[2];
int[] nAngle=new int[nSize]; // sound-image source direction
short[][] sData=new short[2][EXTERNAL_BUFFER_SIZE/2];
short[][] sVal=new short[2][EXTERNAL_BUFFER_SIZE/4];
short[][] sDum =new short[2][iFin];
double[] dRealL=new double[nSize]; // Real part (left)
double[] dImageL=new double[nSize]; // Imaginary part (left)
double[] dRealR=new double[nSize]; // Real part (right)
double[] dImageR=new double[nSize]; // Imaginary part (right)
```
double[] dPow=new double[nSize]; // array for amplitude values
double[] dPh=new double[nSize]; // array for phase values

// other variables
int nSect, nInt, nSamplesWritten, nOffset, nTmp;
int nNumImage;
short sSample, sTmp;
double dSpL, dSpR, dSpImageL, dSpImageR; // amplitude values
double dPhaseL, dPhaseR, dPhaseImageL, dPhaseImageR; // phase values
double dTmp;
double dWindow; // window function

// initialize the fft class

fft test=new fft(); // fft class

nSamplesWritten=0;
try {
    for(nCnt = 0; nCnt < 4; nCnt++)
        inStream.read(); // skip chunk size information
} catch (IOException e) {
    e.printStackTrace();
    System.exit(1);
}

while(nBytesRead != -1) // if there are data
{
    // read out the data and put them into the array ‘abData[]’

    try {
        nBytesRead = inStream.read(abData, 0, abData.length);
    } catch (IOException e) {
        e.printStackTrace();
        break;
    }

    if(nSamplesWritten == 0) {
        nInt = 5;
        sSample = 1;
    }
We have to know the frame size by counting the bytes between
the first and second 0s in the directional data stream
The frame size should be either 512, 1024 or 2048 in the present
version of SOPA.

```java
while(sSample > 0)
{
    sSample = (short)abData[nInt];
    nInt += 4;
}
nSize = nInt - 5;
if(nSize!=512&&nSize!=1024&&nSize!=2048)
{
    System.out.println("Something is wrong in the SOPA file!");
    break;
}
System.out.println("Frame size is "+nSize);
nProc = nSize / nOverlap; // frame increment
nRem = nSize - nProc;
nRatio *= nSize / 512;

Each subset in the HRTF database has 512 data that were sampled at the
sampling rate of 44100 Hz. If the sampling rate of the PCM audio data is
not 44100 Hz or the frame size is not 512, we have to adjust the data.
The variable ‘nRatio’ is used for this adjustment.

```int nHlfSize = nSize / 2; // half of FFT window size
```

A group of frames in succession compose a section.

```java
nSect = nBytesRead / iBYTE / nChannels; // samples per section
if(nBytesWritten>0)
nSect += nRem;
nFrameNum = nSect / nProc; // number of frames in a section
if(nBytesRead==EXTERNAL_BUFFER_SIZE)
nFrameNum -= nOverlap-1;
test.iTap = nSize; // FFT window size
if(nBytesRead >= 0)
{
    rearrangeSOPA(abData,sVal,false);
}
```

rearrangeSOPA is the method that extracts the directional data
and the PCM data from the array ‘abData[]’ and puts them to
the arrays ‘sVal[0]’ and ‘sVal[1].’
// 'sVal[0]' consists of the directional data.
// 'sVal[1]' consists of the PCM data.

for(nInt = 0;nInt < nFrameNum;nInt ++)
{

// Preparation of the PCM data array 'dRealR[ ]' for FFT

nCnt = nProc * nInt;
if(nBytesWritten == 0)
{
    for(nNum = 0;nNum < nSize;nNum ++)
    {
      dRealR[nNum] = (double)sVal[1][nCnt + nNum];
    }
}
else
{
    for(nNum = 0;nNum < nSize;nNum ++)
    {
      if(nCnt + nNum < nRem)
        dRealR[nNum] = (double)sDum[1][nCnt + nNum];
      else
        dRealR[nNum] = (double)sVal[1][nCnt + nNum - nRem];
    }

// fft.fastFt() method transforms the time wave ('dRealR[ ]' and 'dImageR[ ]')
// to the spectral data ('dPow[ ]' and 'dPh[ ]').
// If the last parameter is TRUE, it does the inverse FFT
// otherwise the FFT.

if(test.fastFt(dRealR,dImageR,dPow,dPh,false))
{
    nAngle[nHlfSize] = 0;    // angle of the Nyquist frequency
    for(nNum = 0;nNum < nHlfSize;nNum ++)
    {
      int iBer = nNum / 2;
      int iFreq = nNum / nRatio;
      if(nBytesWritten == 0)
      {
        sTmp = sVal[0][nCnt + iBer];
      }
      else if(nInt < nOverlap - 1)
      {
        sTmp = sDum[0][nCnt + iBer];
      }
      else
      {

sTmp = sVal[0][nCnt - nRem + iBer];
}

// The directional data (8-bit integers) have to be extracted
// from the 16-bit integer 'sTmp.'
if(nNum % 2 == 0)
{
    nAngle[nNum] = sTmp / 256;
}
else
{
    nAngle[nNum] = sTmp % 256;
}
if(nAngle[nNum] <= 0 || nAngle[nNum] > 72)
{
    // in the case of f0 (Hz)
    dSpR = dPow[nNum];
    dSpL = dPow[nNum];
    dSpImageL = dPow[nNum];
    dSpImageR = dPow[nNum];
    dPhaseL = dPh[nNum];
    dPhaseR = dPh[nNum];
    dPhaseImageL = dPh[nNum];
    dPhaseImageR = dPh[nNum];
}
else
{

    // shift the numbers 1 to 72 to 0 to 71
    nAngle[nNum] -= 1;
    if(nAngle[nNum] > 71)
    {
        nAngle[nNum] -= 72;
    }
    else if(nAngle[nNum]<0)
    {
        nAngle[nNum] += 72;
    }

    // Prepare the Temporal HRTF of the left channel
    nOffset = 512 * (71 - nAngle[nNum]) + iFreq;
    nNumImage = 512 * (71 - nAngle[nNum]) + nSize - iFreq;
    if(nNumImage >= 36864)
    nNumImage -= 36864;
    else if(nNumImage < 0)
    }
nNumImage += 36864;
if(nOffset >= 36864)
    nOffset -= 36864;
else if(nOffset < 0)
    nOffset += 36864;

// read out the values from the HRTF databases
// and add them to ‘dPow[]’ and ‘dPh[]’

dTmp = (double)sHrtf[nOffset];
dSpL = dPow[nNum] * dTmp / 2048;
dTmp = (double)sPhase[nOffset];
dPhaseL = dPh[nNum] + dTmp / 10000;
dTmp = (double)sHrtf[nNumImage];
dSpImageL = dPow[nSize - nNum] * dTmp / 2048;
dTmp = (double)sPhase[nNumImage];
dPhaseImageL = dPh[nSize - nNum] + dTmp / 10000;

// Prepare the Temporal HRTF of the right channel

nOffset = 512 * nAngle[nNum] + iFreq;
nNumImage = 512 * nAngle[nNum] + nSize - iFreq;
if(nNumImage >= 36864)
    nNumImage -= 36864;
else if(nNumImage < 0)
    nNumImage += 36864;
if(nOffset >= 36864)
    nOffset -= 36864;
else if(nOffset < 0)
    nOffset += 36864;

// read out the values from the HRTF databases
// and add them to ‘dPow[]’ and ‘dPh[]’

dTmp = (double)sHrtf[nOffset];
dSpR = dPow[nNum] * dTmp / 2048;
dTmp = (double)sPhase[nOffset];
dPhaseR = dPh[nNum] + dTmp / 10000;
dTmp = (double)sHrtf[nNumImage];
dSpImageR = dPow[nSize - nNum] * dTmp / 2048;
dTmp = (double)sPhase[nNumImage];
dPhaseImageR = dPh[nSize - nNum] + dTmp / 10000;
}

// Prepare the real and imaginary parts for inverse FFT

dRealL[nNum] = dSpL * Math.cos(dPhaseL);
dRealR[nNum] = dSpR * Math.cos(dPhaseR);
dImageL[nNum] = dSpL * Math.sin(dPhaseL);
dImageR[nNum] = dSpR * Math.sin(dPhaseR);
if(nNum != 0)
{
    dRealL[nSize - nNum] = dSpImageL * Math.cos(dPhaseImageL);
    dRealR[nSize - nNum] = dSpImageR * Math.cos(dPhaseImageR);
    dImageL[nSize - nNum] = dSpImageL * Math.sin(dPhaseImageL);
    dImageR[nSize - nNum] = dSpImageR * Math.sin(dPhaseImageR);
}
}
dRealL[nHlfSize] = dRealR[nHlfSize];
dImageL[nHlfSize] = dImageR[nHlfSize];

// inverse FFT (left channel)

if(test.fastFt(dRealL,dImageL,dPow,dPh,true))
{

// inverse FFT (right channel)

if(test.fastFt(dRealR,dImageR,dPow,dPh,true))
{

// To make the edges of the frame smooth, the han window is used.

    for(nNum = 0;nNum < nSize;nNum++)
    {
        dWindow = (1 - Math.cos(2 * Math.PI * (double)nNum / (double)nSize)) / 4;
        dRealL[nNum] *= dWindow;
        dRealR[nNum] *= dWindow;
        sData[0][nCnt + nNum] += dRealL[nNum];
        sData[1][nCnt + nNum] += dRealR[nNum];
    }

} else
    System.out.println(“inverse FFT (right) Error\n”);
} else
    System.out.println(“forward FFT Error\n”);
else
    System.exit(0);
bRet[0] = bRet[1] = 0;

// extract byte data from integers and put them to the array 'abData[]'

for(nInt = 0; nInt < nSect - nRem; nInt++)
{
    intToByte((int)sData[0][nInt], bRet);
    abData[nInt * 4] = bRet[0];
    abData[nInt * 4 + 1] = bRet[1];
    intToByte((int)sData[1][nInt], bRet);
    abData[nInt * 4 + 2] = bRet[0];
    abData[nInt * 4 + 3] = bRet[1];
}

if(nBytesRead < EXTERNAL_BUFFER_SIZE)
    nBytesWritten += line.write(abData, 0, nBytesRead + nRem * 4);
else if(nBytesWritten != 0)
    nBytesWritten += line.write(abData, 0, nBytesRead);
else
    nBytesWritten += line.write(abData, 0, nBytesRead - nRem * 4);

nTmp = nSect - nRem;
for(nInt = 0; nInt < nSect; nInt++)
{
    if(nInt < nRem)
    {
        sData[0][nInt] = sData[0][nInt + nTmp];
        sData[1][nInt] = sData[1][nInt + nTmp];
        if(nSamplesWritten == 0)
        {
            sDum[0][nInt] = sVal[0][nInt + nTmp];
            sDum[1][nInt] = sVal[1][nInt + nTmp];
        }
        else
        {
            sDum[0][nInt] = sVal[0][nInt + nTmp - nRem];
            sDum[1][nInt] = sVal[1][nInt + nTmp - nRem];
        }
    }
    else
    {
        sData[0][nInt] = sData[1][nInt] = 0;
    }
}

nSamplesWritten = nBytesWritten / iBYTE / nChannels;

System.out.println(nSamplesWritten + “samples were played.
”);
line.drain();
The source code above is for a program that simply plays a SOPA file. To control the panning during reproduction, adaptively change the variable 'nAngle[]' before it is used to estimate the Temporal HRTF. Additional source code, including rearrangeSOPA(), intToByte(), and fft.class, is required to compile and execute the program. The source code of the necessary files can be downloaded from the SOPA archive page.

6 Binaural rendering of virtual sound sources

As mentioned previously, by using SOPA, the listener can interactively control the panning during the reproduction of sound recorded in a real-world environment. This technology can be used not only for real sound, but also for artificial sounds, for example, in the production of the computer-generated music. In fact, the sample at top of the project web site was produced with MIDI, where musical instruments were placed in the virtual horizontal plane around the reference point. The audio signals and the spatial information at the reference point were then encoded to a SOPA file. Producing binaural signals from virtual sources is sometimes referred to as ‘spatial audio rendering’ or ‘binaural rendering.’

Binaural rendering requires that sound sources be prepared in advance. Different sounds produced by, for example, musical instrument, barking dogs, car horns and other sources, should be in the form of a monaural audio data file. Then the sources can be arranged in the horizontal plane of the reference point, as illustrated in Fig. 18. The sound that should be captured at the reference point can be estimated by the delay-sum method, in which a time delay and an amplitude attenuation corresponding to the path travelled
Fig. 18: Virtual sound sources in the horizontal plane

A top view of the virtual horizontal plane, in which several musical instruments are placed. The sound that should be captured at the reference point can be estimated by the delay-sum method.

by the sound are assigned to the original signals and the latter are simply summed. The signal obtained by the delay-sum method can be used as the reference signal.

Assume that there is a microphone system at the reference point consisting of two cardioid microphones, as shown in Fig. 13. Although the signals that should be captured by the microphones can also be estimated by the delay-sum method, in this case the direction of the sources must be considered in addition to the distance since the sensitivity of the microphones depends on the direction.

Since the inter-channel phase and level difference can be obtained for every frequency from the estimated signals, the direction of each sound-image source can be determined for each frequency. Temporal HRTF can be obtained as described in Section 3.

Once the reference signal and the Temporal HRTF are obtained, they can be used to generate a binaural signal, or they can be encoded into a SOPA file.

The advantage of the SOPA lies in its low requirements for computational resources. Conventional spatial audio rendering is performed by using the HRIR convolution method, and binaural signals are generated by convoluting the HRIR of the corresponding direction to the source signal. If there are
more than two sound sources, HRIR convolution must be performed for each source. This type of rendering is illustrated in Fig. 19, and its requirements in terms of computational resources are rather high, especially when there are many virtual sources.

To make panning on demand available to the listener, every convolution must be performed while the sound is being reproduced since the HRIR to be convoluted depends on the panning.

The delay-sum method used in SOPA rendering is simple and requires fewer computational resources. Since the delay-sum process is completely independent of the panning control, it can be performed in advance to the reproduction. Therefore, during the reproduction of sounds, SOPA rendering requires fewer computational resources than conventional binaural rendering.
Concluding remarks

In this document, we presented the objectives of our project and described the principles of the underlying technology together with the different parts of the project and the structure of SOPA data. Binaural rendering of virtual sound sources in this project was also mentioned.

Despite its relative compactness, the SOPA data format can encode sufficient information for reconstructing a spatial audio scene. The technology can be used in mobile applications allowing the utilization of various user interfaces. The concept of panoramic audio appears to be compatible with the concept of head-mounted displays. SOPA rendering can be performed by plugins in the production of computer-generated music and virtual audio scenes.

As a result of its simple structure and the interactivity it allows, we expect the SOPA format can find wide application in various fields, including the production of qualitatively new audio and audiovisual works, in online communication and for transmission of spatial audio information over networks.

Copyright (c) 2012, AIST

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.