Sound Based Video Maker with Java

Although the author and publisher have made every effort to ensure that the information in this writing was correct at press time, the author and publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

paid link

Have you ever wanted to create videos where the images are displayed based on a sequence of musical notes? This tiny program could be a starting point. It uses the javax.sound.sampled package to handle digital (sampled) audio data and FFMPEG to generate the video. You can download FFMPEG video converter here: https://ffmpeg.org/download/html.

Here is the program:

SoundBasedVideoMaker.java

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.nio.ShortBuffer;
import java.util.ArrayList;
import javax.imageio.ImageIO;
import javax.sound.sampled.AudioFileFormat;
```

```
import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;
import javax.swing.text.BadLocationException;

public class SoundBasedVideoMaker extends JPanel{
    private JFrame frame;

    private PanelNote pnlNote;
    private JTextArea taNote;

    private PanelControl pnlControl;
    private JButton bCreate;

    private ArrayList<String> arrNote;
    private ArrayList<Double> arrPulse;

    private AudioFormat audioFormat;
    private AudioInputStream audioInputStream;

    private ByteBuffer byteBuffer;
    private ShortBuffer shortBuffer;
    private int byteLength;

    private final int sampleSize = 16;
    boolean signed = true;
    boolean bigEndian = true;
    private final int duration = 1;            //1 sec
    private byte audioData[];

    public SoundBasedVideoMaker(){
        setPreferredSize(new Dimension(420,360));

        pnlNote = new PanelNote();
        pnlNote.setPreferredSize(new Dimension(400,300));
        add(pnlNote);

        pnlControl = new PanelControl();
        pnlControl.setPreferredSize(new Dimension(400,100));
        add(pnlControl);
    }

    class PanelNote extends JPanel{
        public PanelNote(){
            taNote = new JTextArea(400,300);
            JScrollPane spNote = new JScrollPane(taNote,JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
                                                        JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
            spNote.setPreferredSize(new Dimension(400,290));
            add(spNote);
        }
    }

    class PanelControl extends JPanel implements ActionListener{
        public PanelControl(){
            bCreate = new JButton('Create');
            bCreate.addActionListener(this);
            add(bCreate);
        }

        @Override
        public void actionPerformed(ActionEvent ae) {
            if(ae.getSource() == bCreate){
                try {
                    createSoundFile();
                    createSoundText();
```

```
                createImageFile();
                createImageText();
                System.out.println('All fine');
                (new Thread(new VideoEncoder())).start();
            } catch (Exception ex) {

            }
        }
    }

    private ArrayList<Double> getNote(String note){
        ArrayList<Double> arrTemp = new ArrayList();
        switch(note.toLowerCase()){
            case 'c':
                for(int i=0;i<16000;i++){
                    arrTemp.add(Math.sin((524*3.14*i)/16000));
                }
                break;
            case 'd':
                for(int i=0;i<16000;i++){
                    arrTemp.add(Math.sin((588*3.14*i)/16000));
                }
                break;
            case 'e':
                for(int i=0;i<16000;i++){
                    arrTemp.add(Math.sin((660*3.14*i)/16000));
                }
                break;
            case 'f':
                for(int i=0;i<16000;i++){
                    arrTemp.add(Math.sin((700*3.14*i)/16000));
                }
                break;
            case 'g':
                for(int i=0;i<16000;i++){
                    arrTemp.add(Math.sin((784*3.14*i)/16000));
                }
                break;
            case 'a':
                for(int i=0;i<16000;i++){
                    arrTemp.add(Math.sin((880*3.14*i)/16000));
                }
                break;
            case 'b':
                for(int i=0;i<16000;i++){
                    arrTemp.add(Math.sin((988*3.14*i)/16000));
                }
                break;
        }

        return arrTemp;
    }

    private void createSoundFile() throws BadLocationException, IOException{
        arrNote = new ArrayList();
        for(String line: taNote.getText().split('\n')){
            arrNote.add(line);
        }

        for(int i= 0;i<arrNote.size();i++){
            audioData = new byte[(int)16000 * 4 * duration];     //4 = bytes per sample for
stereo
            byteBuffer = ByteBuffer.wrap(audioData);
            shortBuffer = byteBuffer.asShortBuffer();

            arrPulse = new ArrayList();
            arrPulse = getNote(arrNote.get(i));

            for(int j=0;j<duration;j++){
                for(int k=0;k<16000;k++){
                    shortBuffer.put((short)(arrPulse.get(k) * 16000));
```

```
                        shortBuffer.put((short)(arrPulse.get(k) * 16000));
                    }
                }

                InputStream byteArrayInputStream = new ByteArrayInputStream(audioData);
                audioFormat = new AudioFormat(16000,sampleSize,2,signed,bigEndian);
                audioInputStream = new
AudioInputStream(byteArrayInputStream,audioFormat,audioData.length/audioFormat.getFrameSize());

                AudioSystem.write(audioInputStream,AudioFileFormat.Type.WAVE,new
File('sound'+i+'.wav'));
            }
        }

        private void createSoundText(){
            String inputFile = 'sounds.txt';

            try {
                FileWriter fileWriter = new FileWriter(inputFile);
                BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
                bufferedWriter.write(''); //emptying the file first

                for(int i=0;i<arrNote.size();i++){
                    String st = 'file '+''+'sound'+i+'.wav'';
                    bufferedWriter.append(st);
                    bufferedWriter.newLine();

                    bufferedWriter.append('outpoint '+1);
                    bufferedWriter.newLine();
                }

                bufferedWriter.close();    //close file
            }catch(IOException ex) {
                ex.printStackTrace();
            }
        }

        private void createImageFile(){
            for(int i=0;i<arrNote.size();i++){
                int imgW = 600;
                int imgH = 400;

                BufferedImage bufimage = new
BufferedImage(imgW,imgH,BufferedImage.TYPE_INT_ARGB);
                Graphics g = bufimage.getGraphics();

                String theNote = arrNote.get(i).toLowerCase();
                System.out.println(theNote);
                if(theNote.equals('c')){
                    g.setColor(Color.MAGENTA);
                }else if(theNote.equals('d')){
                    g.setColor(Color.BLUE);
                }else if(theNote.equals('e')){
                    g.setColor(Color.GREEN);
                }else if(theNote.equals('f')){
                    g.setColor(Color.ORANGE);
                }else if(theNote.equals('g')){
                    g.setColor(Color.YELLOW);
                }else if(theNote.equals('a')){
                    g.setColor(Color.CYAN);
                }else if(theNote.equals('b')){
                    g.setColor(Color.GRAY);
                }

                g.fillOval(200, 100, 200, 200);

                try{
                    String filename = 'image'+i+'.jpg';
                    ImageIO.write(bufimage,'jpg',new File(filename));
                }
                catch(Exception ex){
                }
```

```
            }
        }

        private void createImageText() throws IOException{
            File file = new File('');
            String currentDirectory = file.getCanonicalPath();
            String inputFile = 'images.txt';

            try {
                FileWriter fileWriter = new FileWriter(inputFile);
                BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
                bufferedWriter.write(''); //emptying the file first

                for(int i=0;i<arrNote.size();i++){
                    String st = 'file '+''+'image'+i+'.jpg'';
                    bufferedWriter.append(st);
                    bufferedWriter.newLine();

                    bufferedWriter.append('duration '+1);
                    bufferedWriter.newLine();
                }

                bufferedWriter.close();    //close file
            }catch(IOException ex) {
                ex.printStackTrace();
            }
        }
    }

    private void createAndShowGUI() throws IOException{
        frame=new JFrame('Sound Based Video Maker');
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.add(this);

        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){
            @Override
            public void run() {
                SoundBasedVideoMaker ap;
                try {
                    ap = new SoundBasedVideoMaker();
                    ap.createAndShowGUI();
                } catch (Exception ex) {

                }

            }
        });
    }
}
```

VideoEncoder.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

class ProcessMessage extends Thread
{
    InputStream is;

    ProcessMessage(InputStream is)
    {
        this.is = is;
    }
```

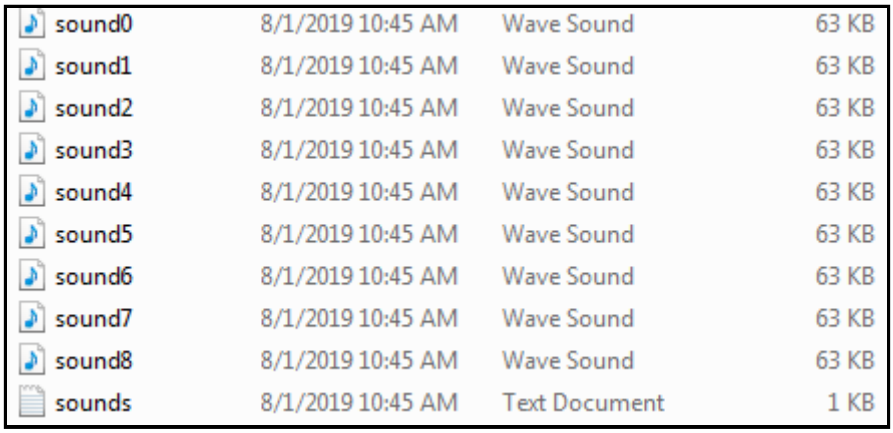
```
    public void run()
    {
        try
        {
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            String line=null;
            while ( (line = br.readLine()) != null)
                System.out.println(line);
            } catch (IOException ioe)
              {
                ioe.printStackTrace();
              }
    }
}

class VideoEncoder implements Runnable{
    private Process proc;

    public void run() {
        try {
            String FFMPEGFile = 'ffmpeg-win64\bin\ffmpeg.exe';
            String soundFile = 'sounds.txt';
            String imageFile = 'images.txt';
            String videoFile = 'video.mp4';

            proc = Runtime.getRuntime().exec(FFMPEGFile+' -f concat -safe 0 -i '+imageFile+' -f
concat -safe 0 -i '+soundFile+' -r 25 -pix_fmt yuv420p '+videoFile);

            ProcessMessage errorMessage = new ProcessMessage(proc.getErrorStream());
            ProcessMessage outputMessage = new ProcessMessage(proc.getInputStream());

            errorMessage.start();
            outputMessage.start();

            int exitVal = proc.waitFor();
            System.out.println('ExitValue: ' + exitVal);

            if(proc.isAlive()){
                proc.destroy();
            }
        }catch(Exception ex){

        }finally{

        }
    }
}
```

Explanation:

This is just a very simple program. For simplicity, user input checking and other important tasks are omitted. Basically, what the program does is to turn musical notes (input from user) into pulses (sampled sound wave/digital audio) and create a sound file for each of the note. Images files are then created for the video. The video generation process is performed with the help of FFMPEG.

```
createSoundFile();
createSoundText();
createImageFile();
createImageText();
(new Thread(new VideoEncoder())).start();
```

Every second, the generated video will display a different image and output a different sound. As you can see in the above block of code, there are two methods that create text files *createSoundText()* and *createImageText()*. The text files contain a list of the generated sounds and images.



sound0	8/1/2019 10:45 AM	Wave Sound	63 KB
sound1	8/1/2019 10:45 AM	Wave Sound	63 KB
sound2	8/1/2019 10:45 AM	Wave Sound	63 KB
sound3	8/1/2019 10:45 AM	Wave Sound	63 KB
sound4	8/1/2019 10:45 AM	Wave Sound	63 KB
sound5	8/1/2019 10:45 AM	Wave Sound	63 KB
sound6	8/1/2019 10:45 AM	Wave Sound	63 KB
sound7	8/1/2019 10:45 AM	Wave Sound	63 KB
sound8	8/1/2019 10:45 AM	Wave Sound	63 KB
sounds	8/1/2019 10:45 AM	Text Document	1 KB

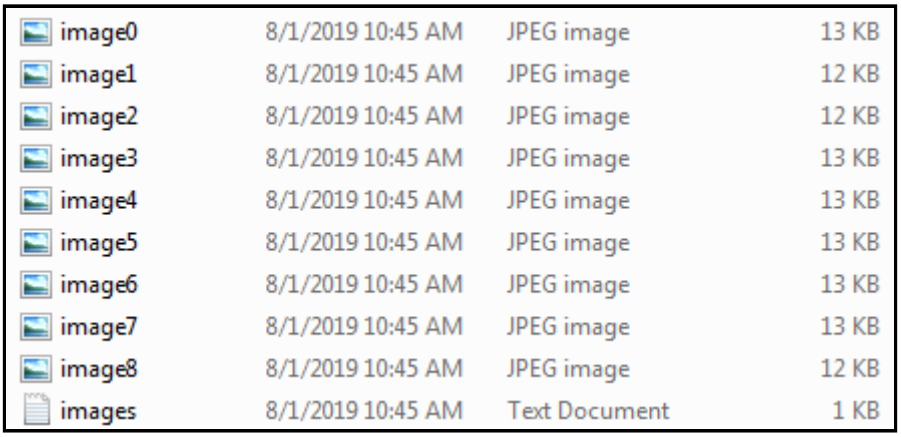
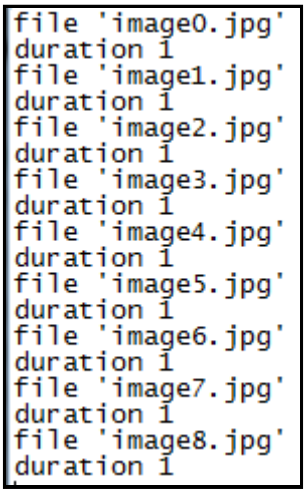


image0	8/1/2019 10:45 AM	JPEG image	13 KB
image1	8/1/2019 10:45 AM	JPEG image	12 KB
image2	8/1/2019 10:45 AM	JPEG image	12 KB
image3	8/1/2019 10:45 AM	JPEG image	13 KB
image4	8/1/2019 10:45 AM	JPEG image	13 KB
image5	8/1/2019 10:45 AM	JPEG image	13 KB
image6	8/1/2019 10:45 AM	JPEG image	13 KB
image7	8/1/2019 10:45 AM	JPEG image	13 KB
image8	8/1/2019 10:45 AM	JPEG image	12 KB
images	8/1/2019 10:45 AM	Text Document	1 KB

The content of the text files:

```
file 'sound0.wav'
outpoint 1
file 'sound1.wav'
outpoint 1
file 'sound2.wav'
outpoint 1
file 'sound3.wav'
outpoint 1
file 'sound4.wav'
outpoint 1
file 'sound5.wav'
outpoint 1
file 'sound6.wav'
outpoint 1
file 'sound7.wav'
outpoint 1
file 'sound8.wav'
outpoint 1
```



As the name implies, the method *createSoundFile()* generates sound files. The *AudioFormat* object specifies raw audio data. It contains a number of attributes that include sampling rate, sample size, and channel. As you can see, this program uses 16000 samples per second with 16 bits per sample.

```
audioFormat = new AudioFormat(16000,sampleSize,2,signed,bigEndian);
```

Note that it uses 2 channels (stereo). But as two speakers will output the same sound, the line *shortBuffer.put((short)(arrPulse.get(k) * 16000));* is written twice.

The *AudioInputStream* object represents the stream of audio data. The *AudioSystem* class then creates sound files based on the audio data and audio format. Put it simply, it stores a stream of audio data from the *AudioInputStream* into an WAV audio file.

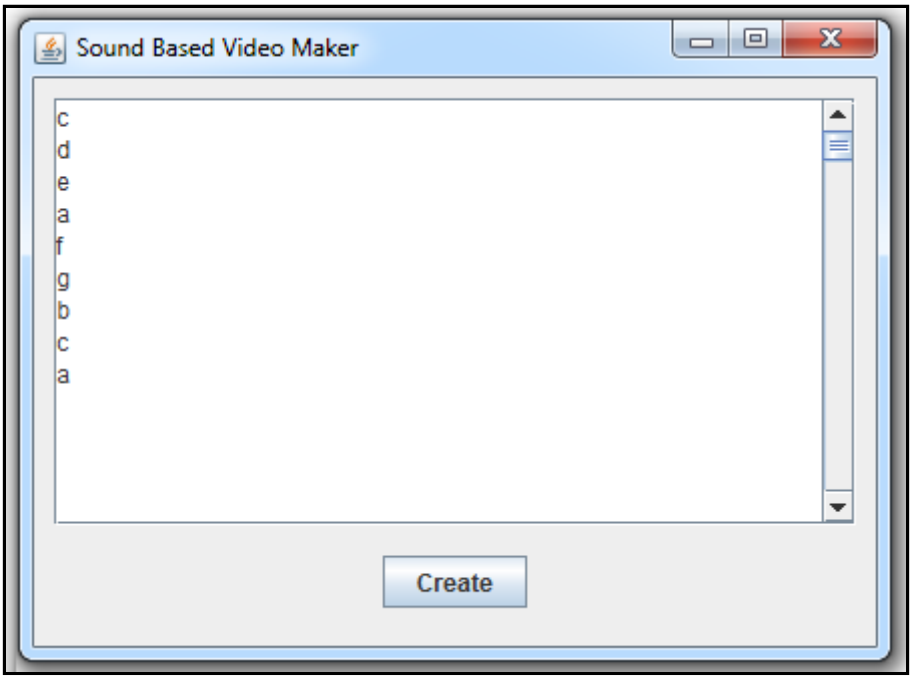
```
AudioSystem.write(audioInputStream,AudioFileFormat.Type.WAVE,new File('sound'+i+'.wav'));
```

The musical notes in this program range from middle C (frequency = 262) to B (frequency = 494). The ArrayList *arrPulse* stores the pulses of each musical note.

The formula to generate pulses is: y = sin[frequency * (2* pi * time). The generation process is conducted in the method *getNote*. The number of samples per second is 16000 and therefore to generate pulses for middle C we use:

```
for(int i=0;i<16000;i++){
    arrTemp.add(Math.sin((524*3.14*i)/16000));
}
```

The ArrayList *arrNote* stores user input. A text area is provided for user input.



The method *createImageFile()* creates image files based on user input (musical note).

```
if(theNote.equals('c')){
    g.setColor(Color.MAGENTA);
}
.
.
.
else if(theNote.equals('b')){
    g.setColor(Color.GRAY);
}
```

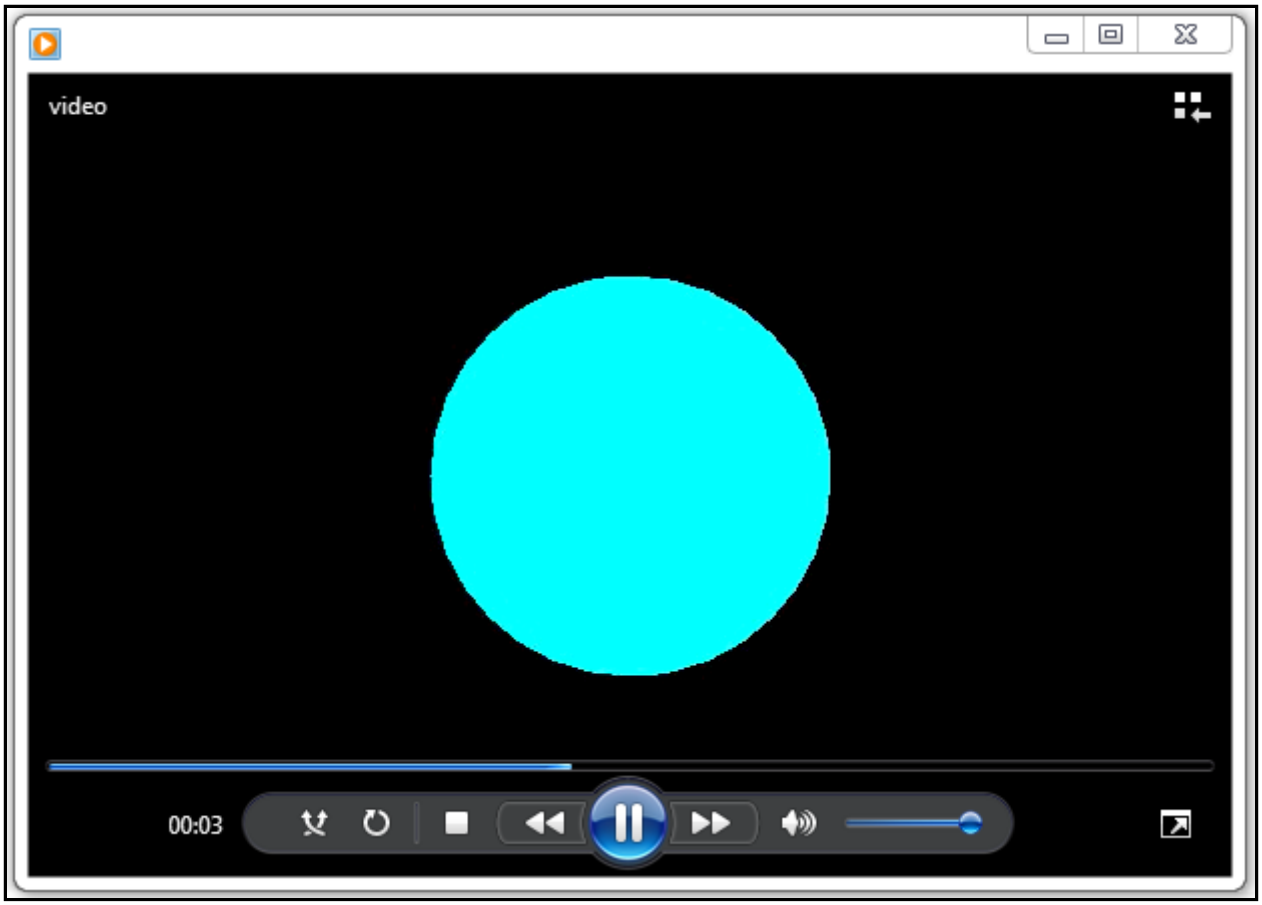
Video creation process is performed by *VideoEncoder.java*. Image files and sound files are concatenated as can bee seen in the following code:

```
proc = Runtime.getRuntime().exec(FFMPEGFile+' -f concat -safe 0 -i '+imageFile+' -f concat -safe
0 -i '+soundFile+' -r 25 -pix_fmt yuv420p '+videoFile);
```

A successful process will generate *ExitValue: 0* in the console.

```
[libx264 @ 00000000004685c0] mb B  I16..4:  0.0%  0.0%  0.0%  B16..8:  1.7%  0.0%  0.0%  direct:
[libx264 @ 00000000004685c0] 8x8 transform intra:19.6% inter:73.2%
[libx264 @ 00000000004685c0] coded y,uvDC,uvAC intra: 1.5% 3.6% 3.0% inter: 0.1% 0.1% 0.1%
[libx264 @ 00000000004685c0] i16 v,h,dc,p: 96%  2%  2%  0%
[libx264 @ 00000000004685c0] i8 v,h,dc,ddl,ddr,vr,hd,vl,hu: 19%  3% 75%  0%  0%  1%  1%  1%  0%
[libx264 @ 00000000004685c0] i4 v,h,dc,ddl,ddr,vr,hd,vl,hu: 16% 19% 42%  3%  3%  4%  6%  4%  3%
[libx264 @ 00000000004685c0] i8c dc,h,v,p: 93%  4%  3%  0%
[libx264 @ 00000000004685c0] Weighted P-Frames: Y:14.3% UV:12.5%
[libx264 @ 00000000004685c0] ref P L0: 60.1% 28.6%  9.9%  1.5%
[libx264 @ 00000000004685c0] ref B L0: 59.2% 40.8%
[libx264 @ 00000000004685c0] kb/s:24.86
[aac @ 000000000046b540] Qavg: 63900.879
ExitValue: 0
```

Here is the output (the video):



Feel free to change the images to anything you want.