Images, Movies and Sound

- Reading, writing and displaying an image file can be accomplished using functions in the **Image Processing Toolbox**, `imread`, `imwrite` and `imshow` respectively.

- MatLab supports 3 types of image structures:

    1. An **indexed** image requires a colourmap and interprets the image data as indices into a colormap matrix. The colormap can be any $m \times 3$ matrix whose rows are valid RGB triplets. If `X` is an image and `cmap` is its associated colour map, then element `X(i,j)` has colour given by the $X(i,j)^{th}$ row of `cmap`, i.e. `cmap(X(i,j),:)` The values of `X` must be between 1 and `length(cmap)` (0:255

at most),. The image can be displayed as:

```
imshow(X,cmap);
```

This type of colour was common in the "old" days (1980's) when memory (frame buffers) was at a premium. A colour image could be displayed using the best 256 colours with reasonable good results and 1/3 the storage requirements (8 bit frame buffer instead of 24 bit frame buffer).

2. An **intensity** image is usually a grayvalue image (each element is a value between 0 and 255). Effectively, it can be displayed as:

```
imshow(X,[]);
```

`[]` indicates the scale argument is omitted, in which case

`[min(X(:)) max(X(:))]` is used.

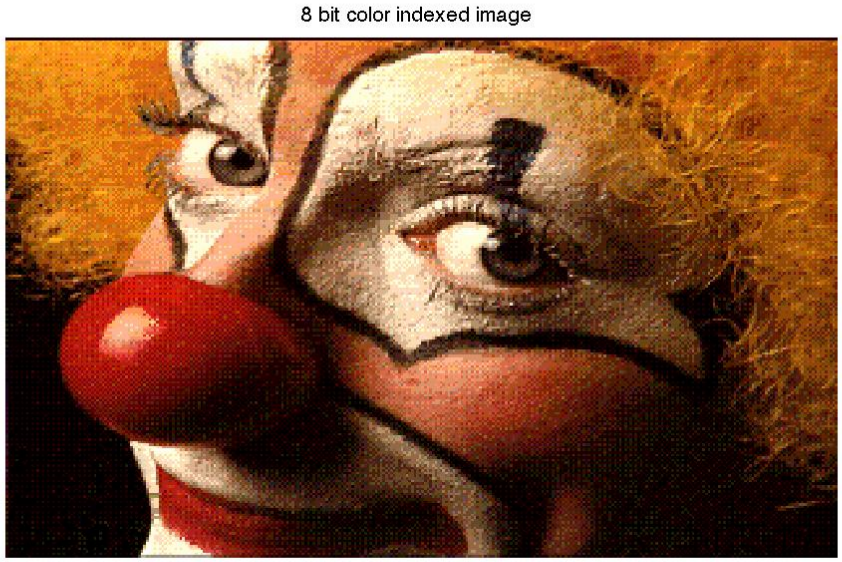
3. A **truecolor** or RGB image is created from a $m \times n \times 3$ array of data specifying valid RGB triplets. A colormap is not required as the colour data is stored in the data array for each pixel `(i,j)`. The name **pixel** is derived from **pic**ture **el**ement. It can be displayed as for a grayvalue image.

- `X` can be unsigned 8 or 16 bit data (`uint8` or `uint16`) or double data (`double`).

- Consider the following MatLab code (in **L10plotImage1.m**) that reads **clown.mat**, a 200×320 8 bit colour image, `X`, with a 81 colour colormap, `map`. **clown.mat** is one of the images provided by MatLab.

```
% 8 bit colour image
% clown sample data, image X and colormap map
load clown;
[r c]=size(X); % pixel dimensions
imshow(X,map)
title('8 bit color indexed image')
print displayed_clown.jpg -djpeg
```

The image is plotted as:

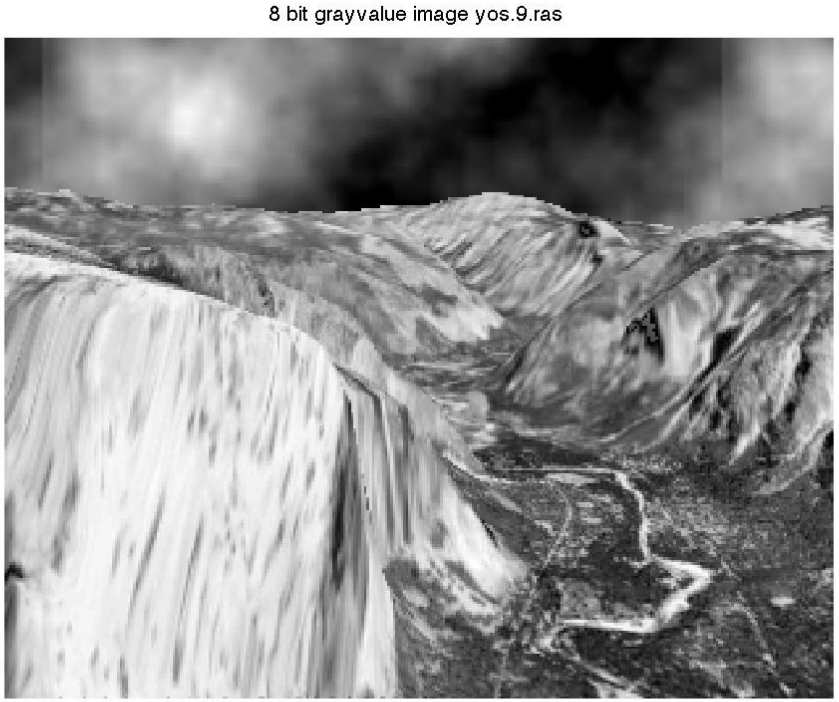


8 bit color image, clown.mat

- A grayvalue images is simply a 2D array, where each element is a number between 0 (black) and 255 (white). A SUN rasterfile grayvalue image of a Yosemite Fly-Through image can be read and displayed in MatLab (in **L10plotImage2.m**) by:

```
% Use imread Image Processing Toolbox function
image=imread('yos.9.ras');
imshow(image,[]);
title('Grayvalue image yos.9.ras');
print displayed_yos.jpg -djpeg
```

The image is plotted as:



8 bit grayvalue image, yos.9.ras

- Truecolor images are stored as 3 colour planes (or 2D arrays). The planes store the red, green and blue values. Thus, each colour plane can be viewed as a 8 bit grayvalue image. A colour is then a triplet of red, green and blue values. Usually these colour values are in [0,255] or normalized to be between [0,1]. Some colour triplets are given in the table below.

Color	Red	Green	Blue
Gray	0.5	0.5	0.5
Silver	0.75	0.75	0.75
Black	0.0	0.0	0.0
White	1.0	1.0	1.0
Red	1.0	0.0	0.0
Green	0.0	1.0	0.0
Blue	0.0	0.0	1.0
Yellow	1.0	1.0	0.0
Cyan	0.0	1.0	1.0
Magenta	1.0	0.0	1.0
Brown	0.65	0.15	0.15
Orange	1.0	0.5	0.0
Pink	1.0	0.5	0.5

Some common colour triplets

A 24 bit truecolor image of Lena can be read and displayed in MatLab (in **L10plotImage3.m**) by:

```
% Use imread Image Processing Toolbox function
image=imread('lena.bmp');
imshow(image,[]);
title('24 bit truecolor image lena.bmp');
print displayed_lena.jpg -djpeg
```

The image is plotted as:

24 bit truecolor image lena.tif

24 bit truecolor image, lena.bmp

Image Files

- The table below shows some image file types supported by MatLab.

Type	Description	Extension
BMP	Windows Bitmap	bmp
GIF	Graphics Interchange Format	gif
JPEG	Joint Photographic Experts Group	jpg, jpeg
JPEG2000	JPEG 2000	j2c, j2k, jp2, jpf, jpx
PBM	Portable Bitmap	pbm
PGM	Portable Graymap	pgm
PNG	Portable Networks Graphics	png
PNM	Portable Any Map	pnm
PPM	Portable Pixmap	ppm
RAS	Sun Raster	ras
TIFF	Tagged Image File Format	tif, tiff

Some common image file types supported by MatLab

Brief File Type Descriptions

- BMP 1,8 or 24 bit uncompressed images.
- GIF 8 bit images
- JPEG 8, 12, 16 bit JPEG images.
- JPEG2000 1, 8, 16 bit JPEG2000 images.
- PBM any 1 bit PBM image, ASCII (plain) or raw (binary) encoded image.
- PGM any standard PGM image. ASCII encoded with arbitrary colour depth and raw encoded with up to 16 bits per grayvalue.

- PNG 1, 2, 4, 8, 16 grayscale images, 8 or 16 bit grayvalue images with alpha channels, 1, 2 4 or 8 bit index (colour images), 24 or 48 bit true-color images with alpha channels.

- PNM any of PPM, PGM or PBM chosen automatically.

- PPM and standard PPM image. ASCII (plain) encoded with arbitrary colour depth (bits) and raw (binary) encoded with up to 16 bits per colour component.

- RAS any ras file including 1 bit bitmap, 8 bit index, 24 bit truecolor or 32 bit with alpha channel.

- TIFF 1, 8, 16 and 24 bit uncompressed images and images with various compression types.

Image I/O

- For example, the MatLab code (in **L10plot_chest_xray_image.m**)

```
% Read image
f=imread('chest-xray.jpg');
% display image
imshow(f,[]);
title('chest-xray.jpg');
% save image with title
print 'chest-xray1.jpg' '-djpeg'
% compute the number of columns and rows of f
[N,M]=size(f);
```

```
% print the number of columns and rows
fprintf('Image chest-xray.jpg is ');
fprintf('%d columns by %d rows\n',N,M);
% write png file
imwrite(f,'chest-xray.png');
figure % don't destroy the jpg display
imshow(f,[]);
title('chest-xray.png');
% save image with title
print 'chest-xray2.png' '-dpng'
% run a system command
system('ls -l chest-xray*');
```

reads a jpeg file and store it in `f` as a 2D array. `[N,M]=size(f)` retrieves the dimensions of `f` and saves them in variables `N` and `M`. `imread` understands the image file formats listed above. The webpage

```
http://www.mathworks.com/help/matlab/
    import_export/supported-file-formats.html
```

gives a complete list.

- The basic Image Processing Toolbox command for writing an image is `imwrite`. For example,

```
imwrite(f,'chest-xray.png')
```

writes this image to the current working directory as a `png` file (note that

we have converted a `jpg` file to a `png` file.

chest-xray.jpg

chest-xray.png

The jpg and png images of the chest xray image, yes they look identical but are different sizes!!!

Output to the MatLab command window consists of:

```
Image chest-xray.jpg is 430 columns by 1290 rows
/tmp/launch-HFZIE9/org.macosforge.xquartz:0
.tcshrc has executed successfully on newfie2
-rwxr--r--@ 1 barron  staff    8586 19 Aug 16:16 chest-xray.jpg
-rwxr--r--  1 barron  staff   78128 20 Aug  2013 chest-xray.png
-rwxr--r--  1 barron  staff   44974 20 Aug  2013 chest-xray1.jpg
-rwxr--r--  1 barron  staff  111728 20 Aug  2013 chest-xray2.png
```

First, Xquartz runs X windows on Mac OSX. Then the tcsh login file .tcshrc is executed. Note how much bigger chest-xray.png is than chest-xray.jpg (although the image qualities are the same); a png file is uncompressed while a jpg file is always compressed to some degree. Note that chest-xray1.jpg and chest-xray2.png are bigger than their corresponding non-figure files as they also have the MatLab figure additional stuff.

Movies

- One way to perform animation in MatLab is to draw the images and render them on top of the previous image. Figure and axes properties must be appropriately set to get a visually smooth animation and, of course, the computations must be simple enough to be computed in real-time.
- If the computations are complex enough and require significant computational resources, you must create a movie.
- The `getframe` command take a snapshot of the current image and `movie` plays the sequence of frames back after they have all been captured.
- Consider the following MatLab code (in **L10spinning_movie1.m**):

```
% rotate a 3D sphere plot
[X,Y,Z]=sphere(50);
% set the coloring using the X values
surf(X,Y,Z,X);
% turn off the axis display,
% keep the image tightly to the axes
% and freezes aspect ratio properties
% to enable rotation of the 3D object
axis vis3d tight off
for k=1:25 % make 25 images
  % change the viewing azimuth angle
  % by 15 degrees but keep the elevation
```

```
    % angle constant at 30 degrees
    view(-37.5+15*(k-1),30);
    % gcf returns the handle of
    % the current figure
    m(k)=getframe(gcf);
end
% display the movie as the images
% with handles in m(k)
movie(gcf,m)
% save the movie as an avi file
% with 5 frames per second
movie2avi(m,'spinning_sphere.avi','fps',5);
```

- This program display an image of the rotating sphere with the sphere colouring along the longitudinal axis determined by the `X` sphere coordinates. The sphere images are displayed at 5 frames per second (so a 5 second movie).

- The avi file for the movie is saved in **spinning_shere.avi**. To display this avi file on a max I use the program **vlc**[1]. To display this avi file on a Windows machine you could use the **mediaplayer**, although a version of vlc for windows is also available[2].

- Unfortunately, this `movie` capability will be depreciated and removed from future versions of MatLab. The alternative is an object oriented

[1] http://www.videolan.org/vlc/download-macosx.html
[2] http://www.videolan.org/vlc/download-windows.html

approach using `VideoReader` and `VideoWriter` classes. We'll wait until the objected oriented programming lecture to see that.

Sound

- MatLab can be used to record, manipulate and playback common sound files as NeXT/Sun audio files (file.au) or Microsoft wave files (file.wav).
- Lets look at code from the University of Dayton (USA) for playing the song "Long and Winding Road" (**L10long_and_winding_road1.m**):

```
% http://homepages.udayton.edu/
%      ~hardierc/ece203/sound.htm
% Loads "the long and winding road" clip"
```

```
% The array road now contains the stereo
% sound data and fs is the sampling frequency
% fs=44100 samples/second is used
% (same as for regular CDs)
% 606171 digital samples - takes time to read!
[road,fs]=wavread('road.wav');
fprintf('road is a %d row by %d column array\n',...
        size(road,1),size(road,2));

% The left and right channel signals are
% the two columns of the road array
left=road(:,1);
```

```
right=road(:,2);


% time of the clip
number_of_samples=length(left);
% time to pause for a sound playback
pause_time=(1/44100)*number_of_samples*1.05;
% 606171 samples too much
number_of_samples=2000;
time=(1/44100)*number_of_samples;
% time value for x dimension of plot
t=linspace(0,time,number_of_samples);
```

```
% plot left signal against t
plot(t,left(1:number_of_samples))
xlabel('time (sec)');
ylabel('left relative signal strength')
title({[num2str(number_of_samples)...
        ' left digital samples'];...
       ['of Long and Winding Road Song']});
print left_long_and_winding_road_signal.jpg -djpeg

figure
% plot left signal against t
plot(t,right(1:number_of_samples))
```

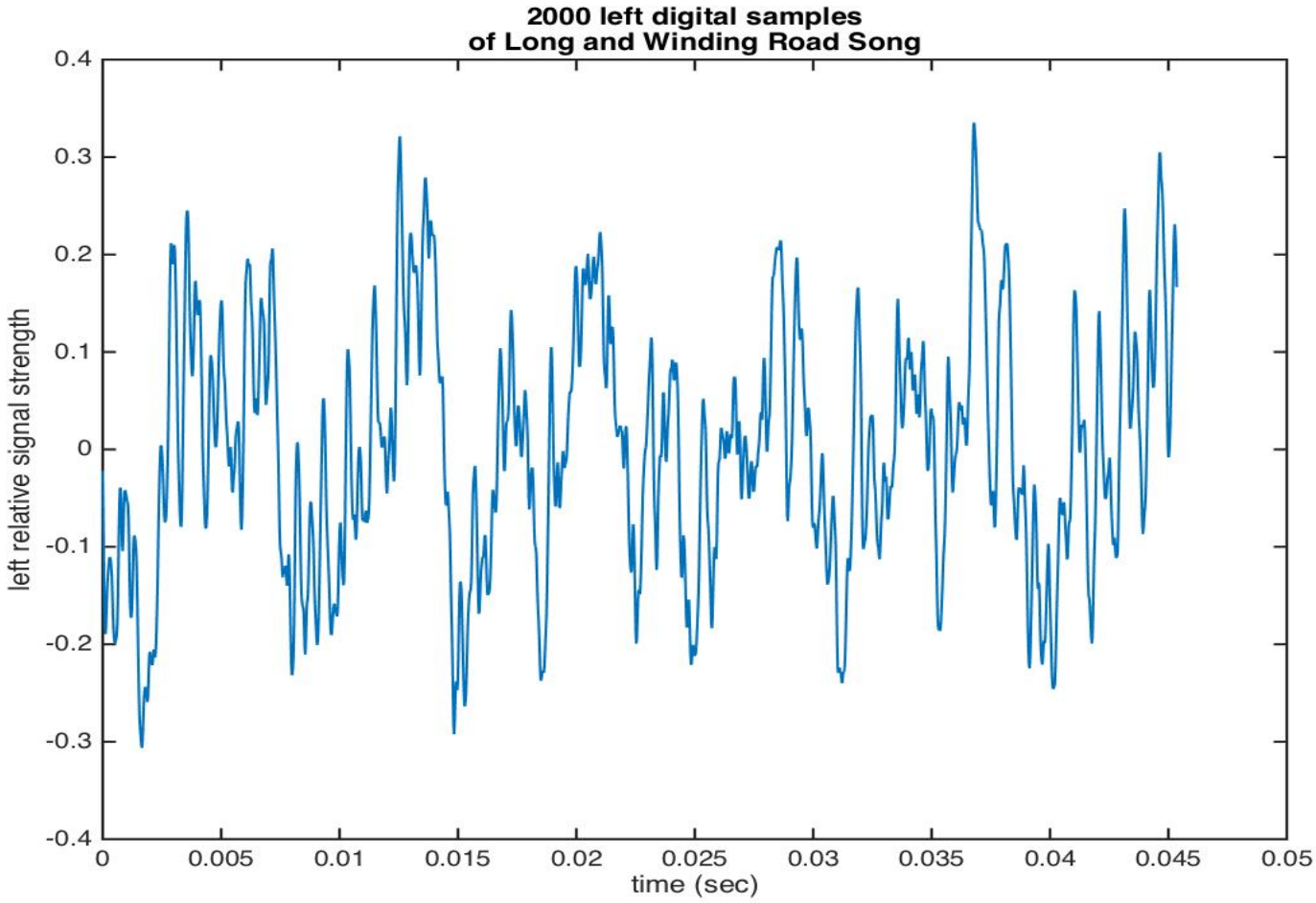
```
xlabel('time (sec)');
ylabel('right relative signal strength')
title({[num2str(number_of_samples)...
        ' right digital samples'];...
       ['of Long and Winding Road Song']});
print right_long_and_winding_road_signal.jpg -djpeg

% Make sure your speakers are turned on
fprintf('Left sound track:\n');
% Play left audio signal at sample rate fs
soundsc(left,fs)
% Wait for the left signal to play
```

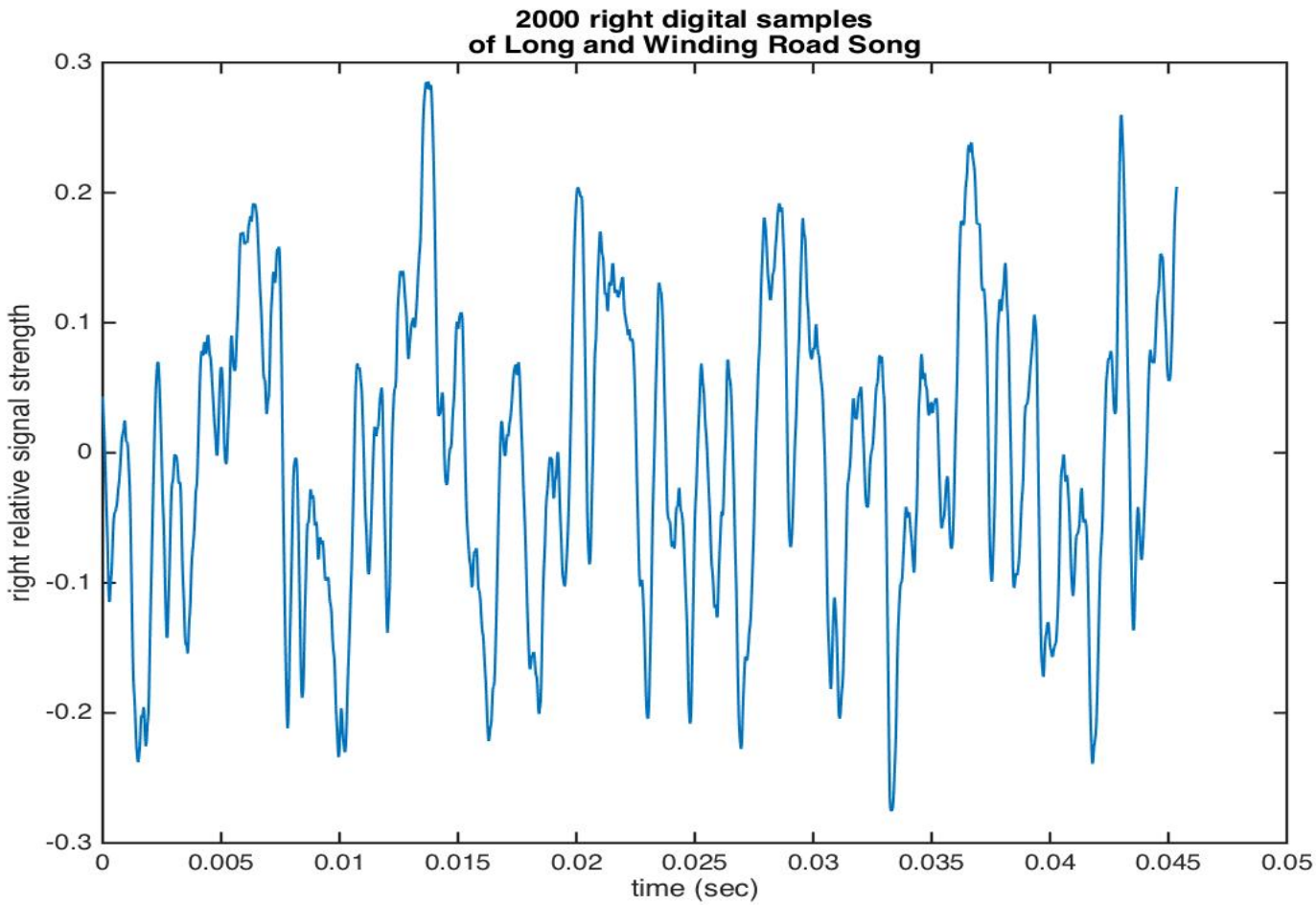
```
pause(pause_time);
fprintf('Right sound track:\n');
% Play right audio signal at sample rate fs
soundsc(right,fs)
% Wait for the right signal to play
pause(pause_time);
fprintf('Stereo sound track:\n');
% Play stereo audio signal at sample rate fs
soundsc(road,fs)
% Wait for the stereo signal to play
pause(pause_time);
fprintf('Left, right and stereo');
```

```
fprintf(' sound tracks played\n');
```

The following plots results:



Left 2000 digital samples of the Long and Winding Road song



Right 2000 digital samples of the Long and Winding Road song

- This is not a signal processing course so we won't cover the various processing options for possible for this signal. But lets consider 3 simple things we can do:

  1. playing the signal at a lower frame rate,
  2. playing the signal at a higher frame rate and
  3. playing "devil" music (playing the track in reverse order).

  The following MatLab code can accomplish this
  (in **L10long_and_winding_road2.m**):

```
[road,fs]=wavread('road.wav');
number_of_samples=size(road,1);
% Reverse the sound samples
```

```
% Flip matrix up to down
devil_road=flipud(road);
pause_time=(1/44100)*number_of_samples*1.05;

fprintf('Slow stereo sound track:\n');
% Play stereo signal at sample rate fs/1.5
soundsc(road,fs/1.5)
% Wait for the stereo signal to play
% 1.5 times slower, need 1.5 time longer time
pause(pause_time*1.5);

fprintf('Fast stereo sound track:\n');
```

```
% Play stereo signal at sample rate fs*1.5
soundsc(road,fs*1.5)
% Wait for the stereo signal to play
% 1.5 times faster, need 1.5 time shorter time
pause(pause_time/1.5);

fprintf('Devil stereo sound track:\n');
% Play stereo signal backwards at sample rate fs
soundsc(devil_road,fs)
% Wait for the stereo signal to play
pause(pause_time);
```

- Changing the frame rate, `fs`, speeds up or slows down the playback of the music. Were these any messages from the devil (when the playback was reversed)?

- Obviously many other things can be done, such as frequency analysis and manipulation of frequency data in the Fourier domain (for example, lowpass filtering, highpass filtering bandpass filtering).