NOTRE DAME UNIVERSITY- Louaize

Faculty of Engineering

ECCE Department

EEN 341

Signals and Systems Laboratory

Final Project

Image Compression and 3D Video Effect

Prepared by:

Firas Dimashki (20208022)

Presented to:

Dr. Nadine Bou Dargham

Date:

10-December-2022

Table of Contents

List of Figures

I. Abstract:

A signal is a quantity that can be measured for different points of time. This quantity can correspond to anything: force, potential, voltage… It can even correspond to audio and image values which are the main study of this project. A signal can be passed throughout a system to transform it into another signal. The aim of this four-part project is to design and implement different systems that can transform an input image or audio into different outputs with unique effects. In the first part, we develop an algorithm to compress using a signal processing technique called Fast Fourier Transform. This means that the output image would have a lower resolution than the original, and a simple blurring effect is added to enhance the result. In the second part, we apply a 3D effect with multiple layers to an image where the viewer can visualize it using a pair of 3D glasses, then we apply a continuous 3D effect to landscape images for the illusion of viewing the landscape in real life. In the third part, we apply a 3D audio effect (special type of audio panning) to a music file. In the fourth part, we combine the 3D audio effect with the 3D image effect in a video where the video frames depend on the intensity of the audio. Each of these effects is theorized and implemented using MATLAB [1] software.

II. Introduction:

A. Project Objectives:

The objectives of this project are to:

- Design and implement an image compression algorithm
- Design and implement a 3D image effect algorithm
- Design and implement a 3D audio effect algorithm
- Combine the 3D audio and image filters into a 3D video

B. Theory and Analysis:

1. Fast Fourier Transform (FFT):

The Fast Fourier Transform is an algorithm used in signal processing for countless applications. It aims to convert a signal from its original domain (time or space) to the frequency domain and show its frequency constituents. It is an advancement to the Discrete Fourier Transform (DFT) because while the DFT has $O(n^2)$ time complexity, the FFT only takes O(n.log(n)) to run (where n is the number of discrete samples) [2]. As seen in the figure below, the FFT algorithm enables us to transform a rather complicated signal into the frequency domain to easily visualize the data provided by the signal. The resulting graph is called the frequency spectrum of the signal.

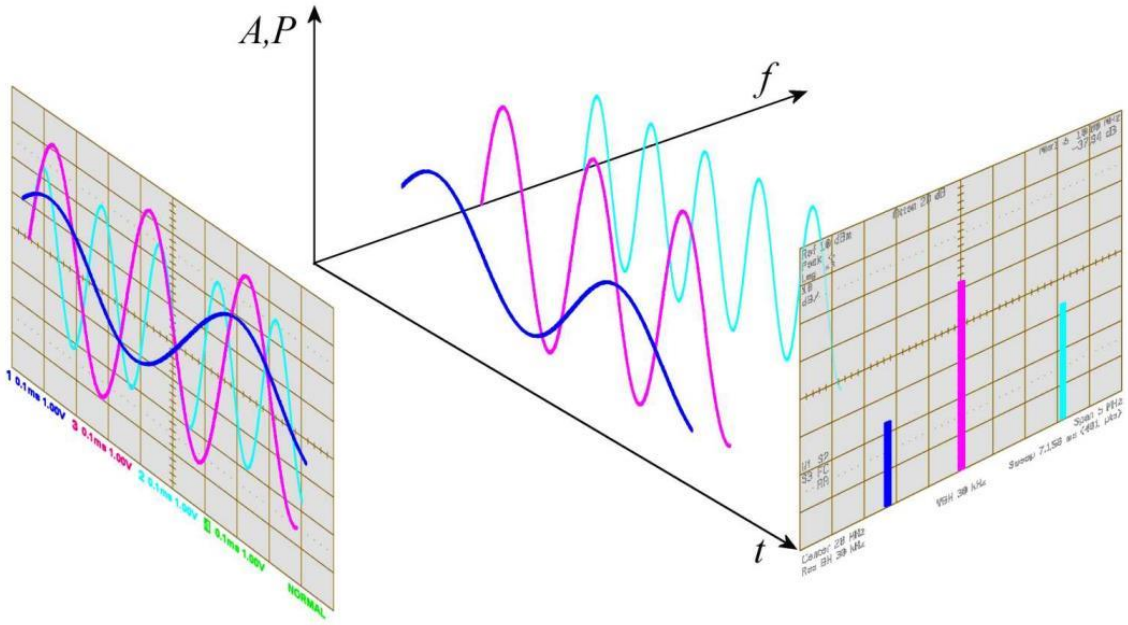

Figure 1: Fourier Transform of a Complicated Signal [3]

2. Images in the Frequency Domain:

Images can be interpreted as multidimensional matrices with values for every pixel. For grayscale images, the corresponding matrix is 2-dimensional (vertical and horizontal). However, colored images are interpreted as three overlapping 2-dimensional matrices, where each layer corresponds to a color (RGB). Observe the frequency spectrum of the following image:

Figure 2: Image Before and After Fourier Transform [4]

The frequency spectrum of the image has the same dimensions as the original image. This means that changing the number of pixels in the frequency domain results in the same change in the space domain. However, this does not mean that cropping the frequency spectrum will result in an image crop for the original image. Notice how the frequency domain contains most of its values in the center of the spectrum (the brighter the pixel, the higher value it contains). This is very critical in the process of compressing the image, as we will see shortly.

3. 3D Images:

Our brains are amazing organs for many reasons, one of which is the ability to adapt the senses, especially seeing and hearing, in order to understand what we sense. Because of this ability, it is easy to create visual and audio illusions that are very interesting to study and very fun to observe. One of these illusions is viewing a 3-dimensional image on a 2-dimensional digital screen. This is usually done by taking photographs with 3D cameras which have 2 or 3 lenses and editing them. The illusion is created by highlighting the blue and red colors of the 3D image, and it can be viewed using 3D glasses.

Figure 3: 3D Glasses

The left eye sees the red part of the image while the right eye sees the blue part. Since every eye sees a different part of the image, the brain tries to assemble what we see into a 3D image.

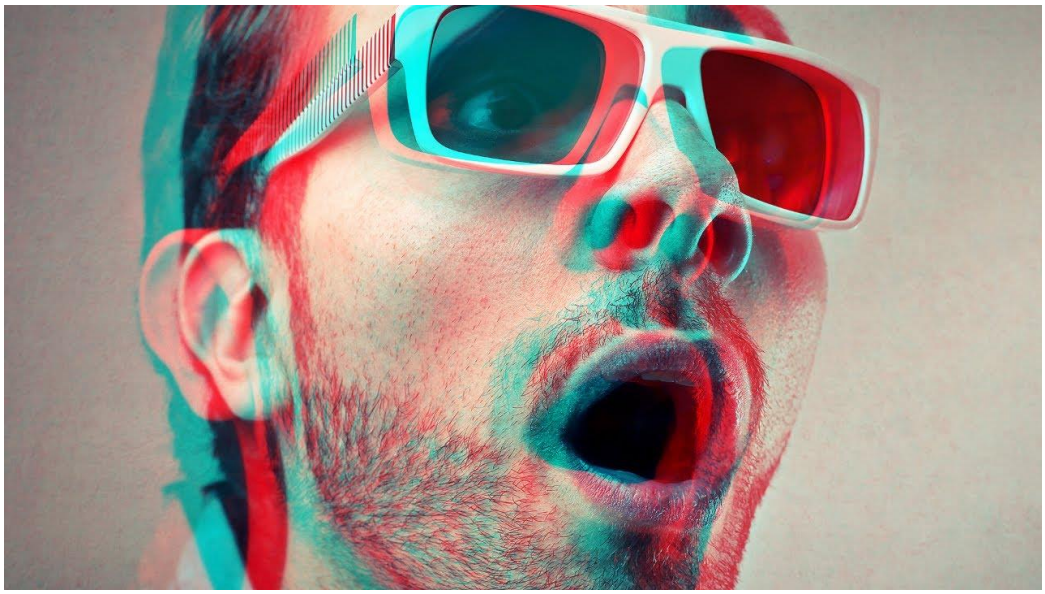

Figure 4: Example of 3D Image

The challenge however is to create a 3D image digitally without using a 3D camera.

4. Audio Panning:

Another interesting illusion that can be created using signal processing is the 3D audio effect. It is a special case of audio panning where the DJ or producer can manipulate the output of the left and right stereo speakers independently. The concept of 3D audio is that while the output of the left stereo increases, that of the right stereo decreases, and vice versa. The brain interprets this change as a sound source that moves around the head of the listener. Obviously to experience this effect the listener must be using headphones or stereo speakers.

III. Project Description and Procedure:

A. Image Compression:

As mentioned earlier, the image compression utilizes the FFT algorithm. Theoretically, if we crop the frequency spectrum of the image, we will get rid of the pixels that do not carry much value. However, the output image would not be cropped, instead, it would have a lower resolution while maintaining the greater portion of the original data. A blurry effect is added by combining the colors of adjacent pixels into the pixel at hand. The MATLAB script below is what we will use to apply this effect.

```
clc
clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%user defined variables:
image_filename = 'flower.jpg';
compression_ratio = 15;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%read image and properties
im1 = imread(image_filename); %read image
r = im1(:, :, 1); %red part of image
g = im1(:, :, 2); %green part of image
b = im1(:, :, 3); %blue part of image
s = size(im1);
rowsIn = s(1); %input number of pixel rows
colsIn = s(2); %input number of pixel columns
ratio = sqrt(compression_ratio);
rowsOut = ceil(rowsIn/ratio); %output number of pixel rows
colsOut = ceil(colsIn/ratio); %output number of pixel columns

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%calculate number of rows and columns to be removed
rows1 = floor((rowsIn-rowsOut)/2);
rows2 = rowsIn - rows1;
cols1 = floor((colsIn-colsOut)/2);
cols2 = colsIn - cols1;
rows1=rows1+1;
cols1=cols1+1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%compress image
%apply fft to red, green, and blue parts of image
R = fft2(r);
G = fft2(g);
B = fft2(b);
%shift fft for rgb parts
```

```
Rsh = fftshift(R);
Gsh = fftshift(G);
Bsh = fftshift(B);
%crop transform in frequency domain to reduce resolution
Rsh2 = Rsh(rows1:rows2, cols1:cols2);
Gsh2 = Gsh(rows1:rows2, cols1:cols2);
Bsh2 = Bsh(rows1:rows2, cols1:cols2);
%apply inverse fft shifting
R2 = ifftshift(Rsh2);
G2 = ifftshift(Gsh2);
B2 = ifftshift(Bsh2);
%apply inverse fft
r2 = ifft2(R2);
g2 = ifft2(G2);
b2 = ifft2(B2);
compressed = real(cat(3, r2, g2, b2)); %compressed image

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%blur image
s2 = size(compressed);
rows3 = ceil(s2(1)); %rows of compressed image
cols3 = ceil(s2(2)); %columns of compressed image
blurred = compressed; %intialize blurred image
%blur colors from adjacent pixels,
%an offset of 2 pizels from the edges are ignored
for i = 3:rows3-2
    for j = 3:cols3-2
        for k = 1:3
            blurred(i, j, k) = 0.25*compressed(i, j, k) ...
                + 0.09*(compressed(i-1,j,k)+compressed(i,j-1,k) ...
                +compressed(i,j+1,k)+compressed(i+1,j,k))...
                + 0.06*(compressed(i-1,j-1,k)+compressed(i-1,j+1,k) ...
                +compressed(i+1,j-1,k)+compressed(i+1,j+1,k))...
                + 0.02*(compressed(i-2,j,k)+compressed(i+2,j,k) ...
                +compressed(i,j-2,k)+compressed(i,j+2,k));
        end
    end
end
blurred = blurred(3:rows3-2, 3:cols3-2, :); %remove 2 pixels from edges

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%show images
compressed = rescale(compressed); %rescale compressed image
blurred = rescale(blurred); %rescale blurred image
%show original image
figure(1)
imshow(im1)
title('Original Image')
%show frequency spectra of original image (log transform)
%each color layer is shown alone then the total rgb spectrum is shown
figure(2)
title('Original Image')
subplot(2,2,1)
Rshl = log(1+abs(Rsh));
imshow(rescale(Rshl))
title('RED Frequency Spectrum')
subplot(2,2,2)
```

```
Gshl = log(1+abs(Gsh));
imshow(rescale(Gshl))
title('GREEN Frequency Spectrum')
subplot(2,2,3)
Bshl = log(1+abs(Bsh));
imshow(rescale(Bshl))
title('BLUE Frequency Spectrum')
subplot(2,2,4)
imshow(rescale(cat(3, Rshl, Gshl, Bshl)))
title('RGB Frequency Spectrum')
%show frequency spectra of compressed image
figure(3)
subplot(2,2,1)
Rsh2l = log(1+abs(Rsh2));
imshow(rescale(Rsh2l))
title('RED Frequency Spectrum')
subplot(2,2,2)
Gsh2l = log(1+abs(Gsh2));
imshow(rescale(Gsh2l))
title('GREEN Frequency Spectrum')
subplot(2,2,3)
Bsh2l = log(1+abs(Bsh2));
imshow(rescale(Bsh2l))
title('BLUE Frequency Spectrum')
subplot(2,2,4)
imshow(rescale(cat(3, Rsh2l, Gsh2l, Bsh2l)))
title('RGB Frequency Spectrum')
%show compressed image
figure(4)
imshow(compressed)
title('Compressed Image')
%show blurred image
figure(5)
imshow(blurred)
title('Blurred Image')
```

B. 3D Image Effect:

The theory of this effect is to convert the input image to grayscale and create a red and blue matrix from this grayscale. Then, these red and blue parts are each shifted in a direction (left or right) to create the illusion. Shifting the red part to the left and the blue part to the right creates the illusion of the image being deep inside the screen, while shifting the blue part right the red part left creates the illusion of the image coming out of the screen. The following MATLAB script tests this theory by creating this effect for squares of different dimensions, each shifted with a different offset, so that they would appear as multiple layers over each other.

```
clc
clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%user defined variable:
offset = 50;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%read square images
square1 = imread('Square1.jpg');
square2 = imread('Square2.jpg');
square3 = imread('Square3.jpg');
square4 = imread('Square4.jpg');
square5 = imread('Square5.jpg');
s = size(square1);
cols = s(2);
%placing read images in a list
input_images = cat(4, square1, square2, square3, square4, square5);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%applying the 3D effect to the square images
g = uint8(zeros([s(1), s(2)-offset])); %green part of output (all zeroes)
output_image = cat(3, g, g, g); %initialize output image to zeroes
for i = 1:5
    %convert each input image to grayscale
    gray = rgb2gray(input_images(:,:,:,i));
    val = (i-1)*(offset/5); %offset calculated for each image
    r = gray(:, offset-val+1:cols-val); %shift red part
    r = rescale(r)/2; %rescale values
    b = gray(:, val+1:cols-(offset-val)); %shift blue part
    b = rescale(b); %rescale values
    rgb = cat(3, r, g, b); %colored image with effect
    output_image = output_image + rgb; %combine 3D images to one image
end
output_image = uint8(rescale(output_image)*255); %rescale output image
imshow(output_image)
imsave()
```

Another case to try is the same image being shifted at different offsets depending on the row number of the image. This effect should work on landscape images where all the pixels at the bottom are closest and all the pixels at the top are the farthest, so it is a special case that does not work properly on any image.

```
clc
clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%user defined variables
offset = 30;
image_filename = 'Landscape1.jpg';
```

```
% image_filename = 'Landscape2.jpg';
% image_filename = 'Landscape3.jpg';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%read image
im = imread(image_filename); %read image
gray = rgb2gray(im); %convert to gray scale
s = size(gray);
rows = s(1); %number of rows
cols = s(2); %number of columns

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%calculate vertical and horizontal divisions
vertical_divisions = ceil(cols/offset);
horizontal_divisions = ceil(rows/vertical_divisions);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%calculating red and blue parts row by row
r = zeros(s-[0,vertical_divisions]); %initializing red part
b = zeros(s-[0,vertical_divisions]); %initializing blue part
for i = 1:rows
    %determine which section to apply corresponding 3D offset
   section = floor(i/horizontal_divisions/2);
   r(i,:) = gray(i,vertical_divisions+1-section:cols-section);
   b(i,:) = gray(i,1+section:cols-(vertical_divisions-section));
end
r = rescale(r)/2; %rescale red part
b = rescale(b); %rescale blue part
g = zeros(size(r)); %green part (all zeroes)
output = cat(3, r, g, b); %combine colors
imshow(output)
imsave()
```

C. 3D Audio Effect:

As mentioned before, the audio effect works by increasing the left side output while decreasing the right side and vice versa. To achieve this effect, we multiply the left and right parts of the audio with a positive triangular signal, with the right triangular signal time shifted by half a period to avoid increasing and decreasing the left and right sides simultaneously.

```
clc
clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%user defined variables
audio_filename = 'HealingRiver.mp3';
```

```
T_audio_effect = 8; %period of 3D audio effect

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[audio,fs] = audioread(audio_filename); %read audio file and its properties
N = T_audio_effect * fs; %number of samples in 3D period
tri = @(t) 0.5*sawtooth(2*pi/N*t,0.5) + 0.5 + 0.01; %3D audio effect signal
t = 1:length(audio); %samples of audio
%audio for left side
L = tri(t)'.*audio(:,1);
%audio for right side
R = tri(t-N/2)'.*audio(:,2); %delay of N/2 between two sides
out = [L, R]; %overall output signal
%play audio
player = audioplayer(out,fs);
play(player)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%plot input and output signals
subplot(2,1,1)
plot(audio) %input signal
hold on
plot(tri(t), 'c') %left triangular signal
plot(tri(t-N/2), 'y') %right triangular signal
hold off
ylim([-1.1 1.1])
title('Input Signal and 3D Audio Effect')
subplot(2,1,2)
plot(out) %output signal
ylim([-1.1 1])
title('Output Signal')
```

D. 3D Video:

For the 3D video, we combine the 3D audio and 3D image effects into a video. The image frames consist of a 3D background and a 3D object at the center which changes its offset according to the beat of the audio. The background is the output of the code above for the landscape 3D effect if the “Landscape2.jpg” file was chosen, but it was also externally edited for a more beautiful result. With 3D glasses, the viewer can see that the central object is going in and out of the screen according to the beat. The video writer object writes the video frame by frame, so for each video frame the image and audio frames should be computed. Also, to link the image frames to the audio, an array is used to hold the average volume of each audio frame. A temporary audio file is used to store the audio frame for each video frame, and it is deleted at the end. Note that the video will take up a lot of disk space if the video compressor is not activated.

```
clc
clear

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%user defined variables:
background_filename = 'back.jpg';
center_filename = 'note.jpg';
audio_filename = 'HealingRiver.mp3';
output_filename = 'video_out.avi';
fps = 20; %video frame rate
T_audio_effect = 15; %period of 3D audio effect
offset = 20; %maximum offset of pixels
hi_offset = 0; %offset from maximum (closest) image (to be ignored)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%read images and their properties
background = imread(background_filename); %read background image
center = imread(center_filename); %read center image
szb = size(background); %size of background image
szc = size(center); %size of center image
diffr = floor((szb(1)-szc(1))/2); %vertical(row) difference in size
diffc = floor((szb(2)-szc(2))/2); %horizontal(column) difference in size
gray = rgb2gray(center); %convert to grayscale
rows = szc(1); %number of pixel rows
cols = szc(2); %number of pixel columns
szc = szc -[0, offset,0]; %size of center after applying effect

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%create frames list
%initialize matrix to contain image for different distances
%frames list is 4 dimensional, 2 for rows and columns,
%1 for rgb layers, 1 for different frames
frames = cat(4);
g = zeros([rows, cols-offset]); %green part of output image is zero
%create images for different distances
for i = 1:offset-hi_offset
    r = gray(:, offset-i+1:cols-i); %shift red part
    r = rescale(r)/2; %rescale values
    b = gray(:, i+1:cols-(offset-i)); %shift blue part
    b = rescale(b); %rescale values
    rgb = cat(3, r, g, b); %colored image with effect
    rgb = uint8(rgb*255); %rescale to 8 bit integers
    %create output image that has the same size as background but contains
    %the center image
    output = uint8(zeros(szb)); %initialize output to zero
    %change center of output to hold center image
    output(diffr+1:szc(1)+diffr, diffc+1:szc(2)+diffc,:) = rgb;
    frames = cat(4, frames, output); %add image to frames list
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%create audio_offset which affects the image frames according to volume
```

```
[audio, fs] = audioread(audio_filename); %read audio
% audio = audio(1500000:3000000, 1); %crop audio (optional)
%rescale audio file values to positive numbers between 0 and 1
audio_rescale = rescale(abs(audio));
%create video writer object
videoWriter = ...
    vision.VideoFileWriter(output_filename, 'AudioInputPort', true);
%calculate number of frames in video
nFrames = floor(size(audio, 1)/fs*fps);
videoWriter.FrameRate = fps; %specify frame rate
% videoWriter.VideoCompressor = "DV Video Encoder"; %compress video
%calculate length of audio frame
audio_frame_length = floor(size(audio, 1)/nFrames);
%array that contains average volume values for every frame
audio_offset = zeros([nFrames, 1]);
%calculate average for every audio frame
for i = 0:nFrames-1
    audio_offset(i+1) = ...
        sum(audio_rescale((i*fs)/fps+1:(i+1)*fs/fps), 'all');
end
%rescale to get the indexes of images
audio_offset = uint8(rescale(audio_offset, 0, size(frames, 4)-1))+1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%apply 3D audio effect
N = T_audio_effect * fs; %number of samples in 3D period
tri = @(t) abs(sawtooth(2*pi/N*t,0.5)+0.2); %3D audio effect signal
t = 1:length(audio);
%audio for left side
L = tri(t)'.*audio(:,1);
%audio for right side
R = tri(t-N/4)'.*audio(:,2); %delay of N/4 between two sides
audio = [L, R]; %overall output signal

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%write video file frame by frame
for i = 1:nFrames
    %get image frame
    image_frame = 3*frames(:, :, :, audio_offset(i))+2*background;
    %get audio frame
    audio_frame = ...
        audio(audio_frame_length*(i-1)+1:audio_frame_length*i, :);
    %write audio frame to temporary file
    audiowrite('audio_frame.wav', audio_frame, fs)
    %write image frame and audio frame from temporary file
    step(videoWriter, image_frame, audioread('audio_frame.wav'));
end
release(videoWriter) %release output video
delete 'audio_frame.wav' %delete temporary audio frame file
```

IV. Results and Interpretation:

A. Image Compression:

After running the code, we chose a random image to try the filter on. Below are the original and its frequency spectrum.

Figure 5: Original Image for Compression

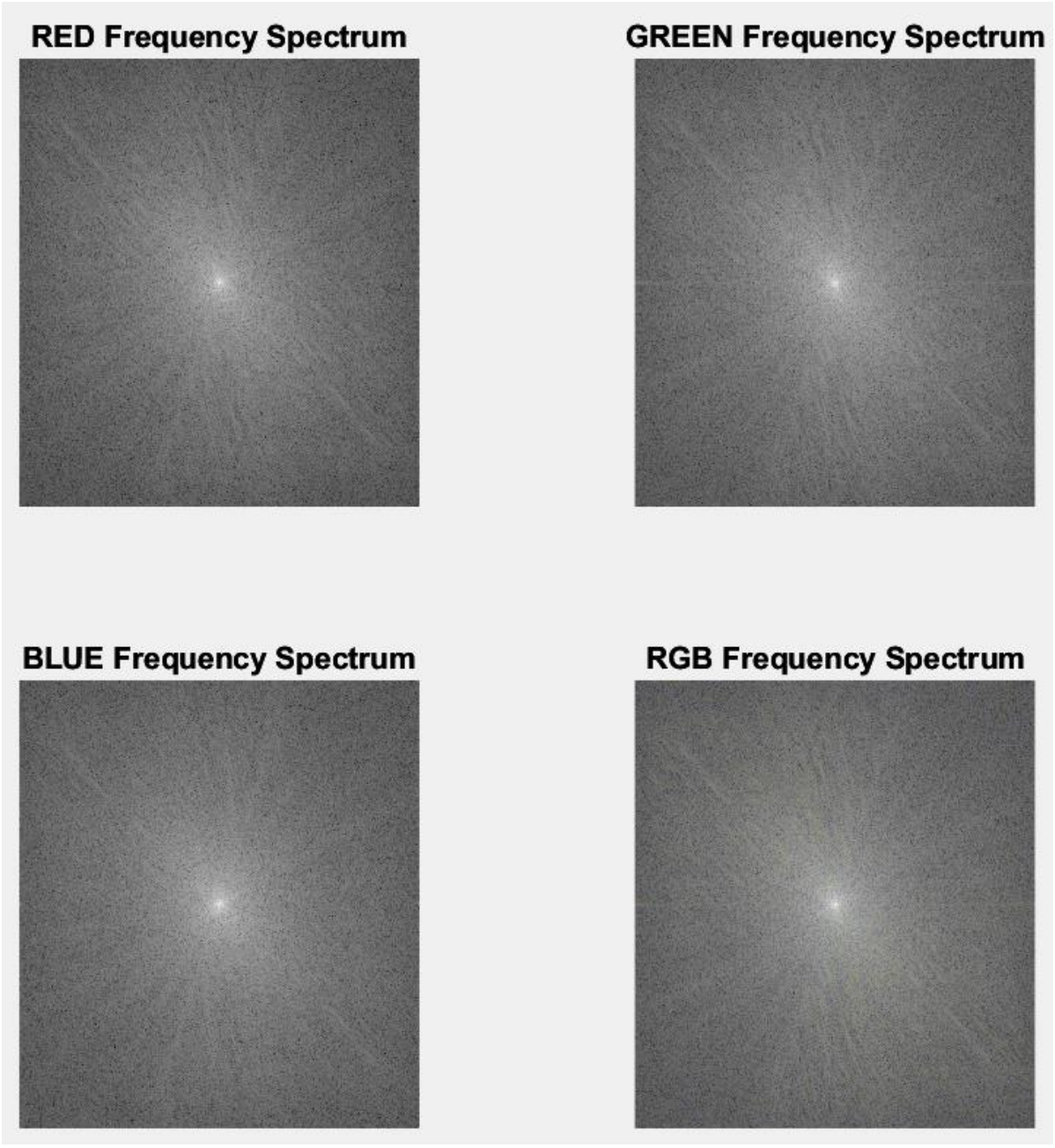

Figure 6: Original Frequency Spectrum

As expected, the frequency spectrum contains most of its data in the center. We now zoom in to the frequency spectrum to see what it looks like up close.

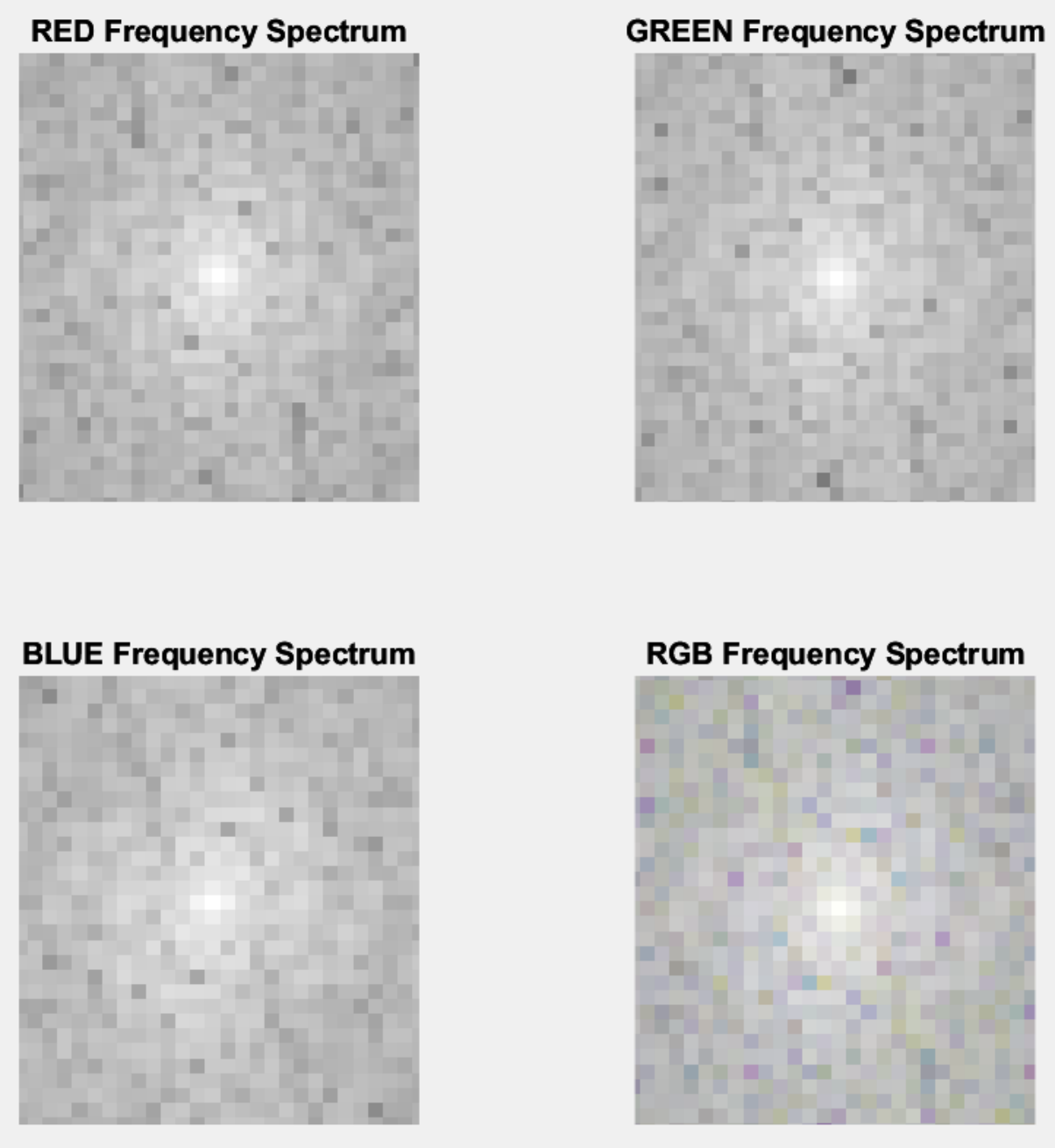

Figure 7: Original Frequency Spectrum Zoomed In

This is what the center of the spectrum looks like. This part holds the greatest amount of data in the spectrum. Notice that the spectra for the 3 color layers are very close, which is why the RGB spectrum does not have many colored pixels.

Next, we show the cropped frequency spectrum.

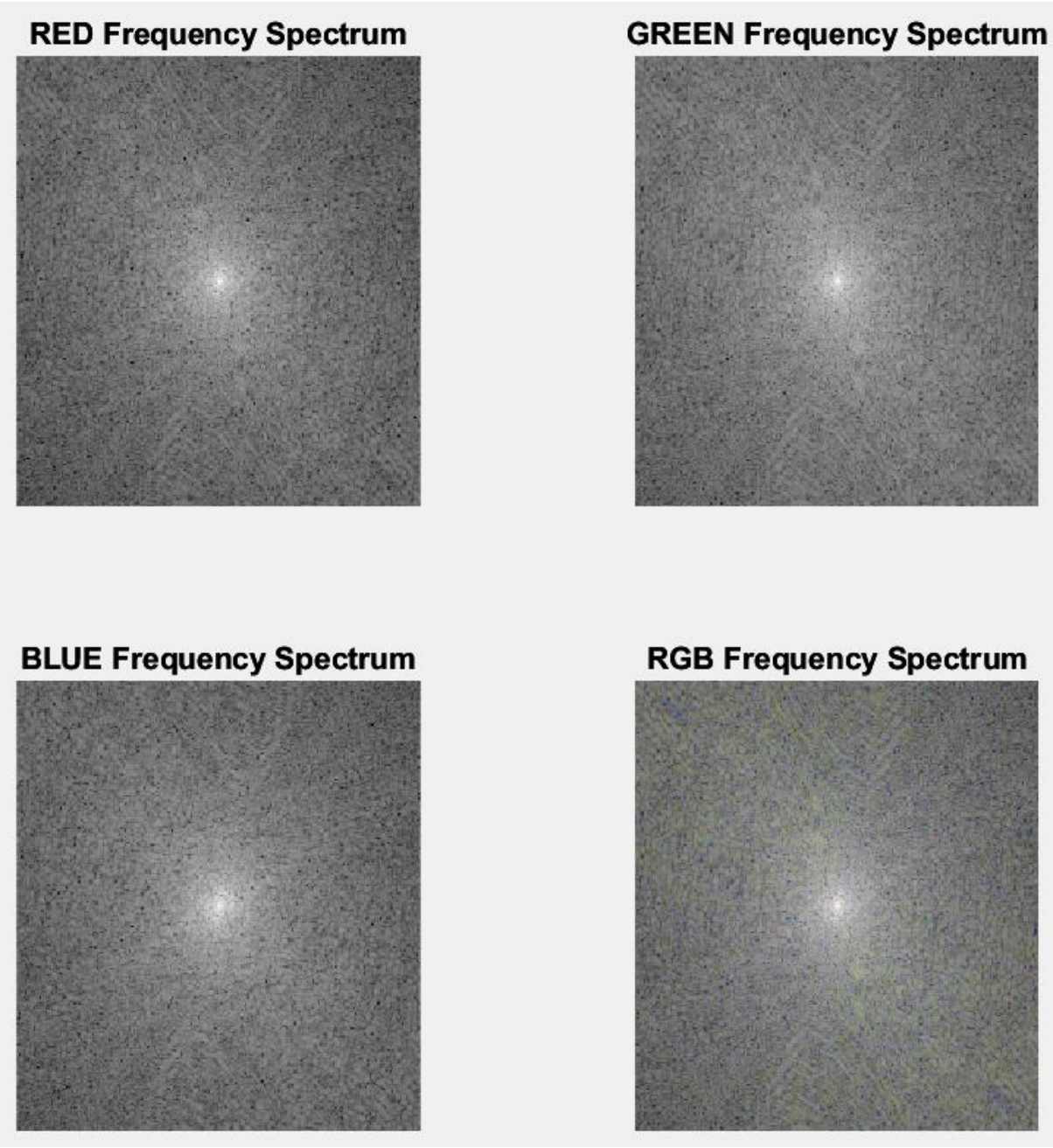

Figure 8: Cropped Frequency Spectrum

This is what the frequency spectrum of the compressed image looks like. It is simply a crop from the original frequency spectrum. It is clear that this spectrum has a lot less pixels than the original. After the inverse FFT is performed, the compressed image is obtained.

Figure 9: Compressed Image

As you can see, the compressed image clearly has a lower resolution than the original, so our theory is correct. You might notice that the colors are a little different than the original. Blurring the compressed image solves this problem.

Figure 10: Blurred Image

Here the colors are restored, and the blurry effect is obvious.

B. 3D Images:

Remember that to really visualize these 3D images you need a pair of 3D glasses.

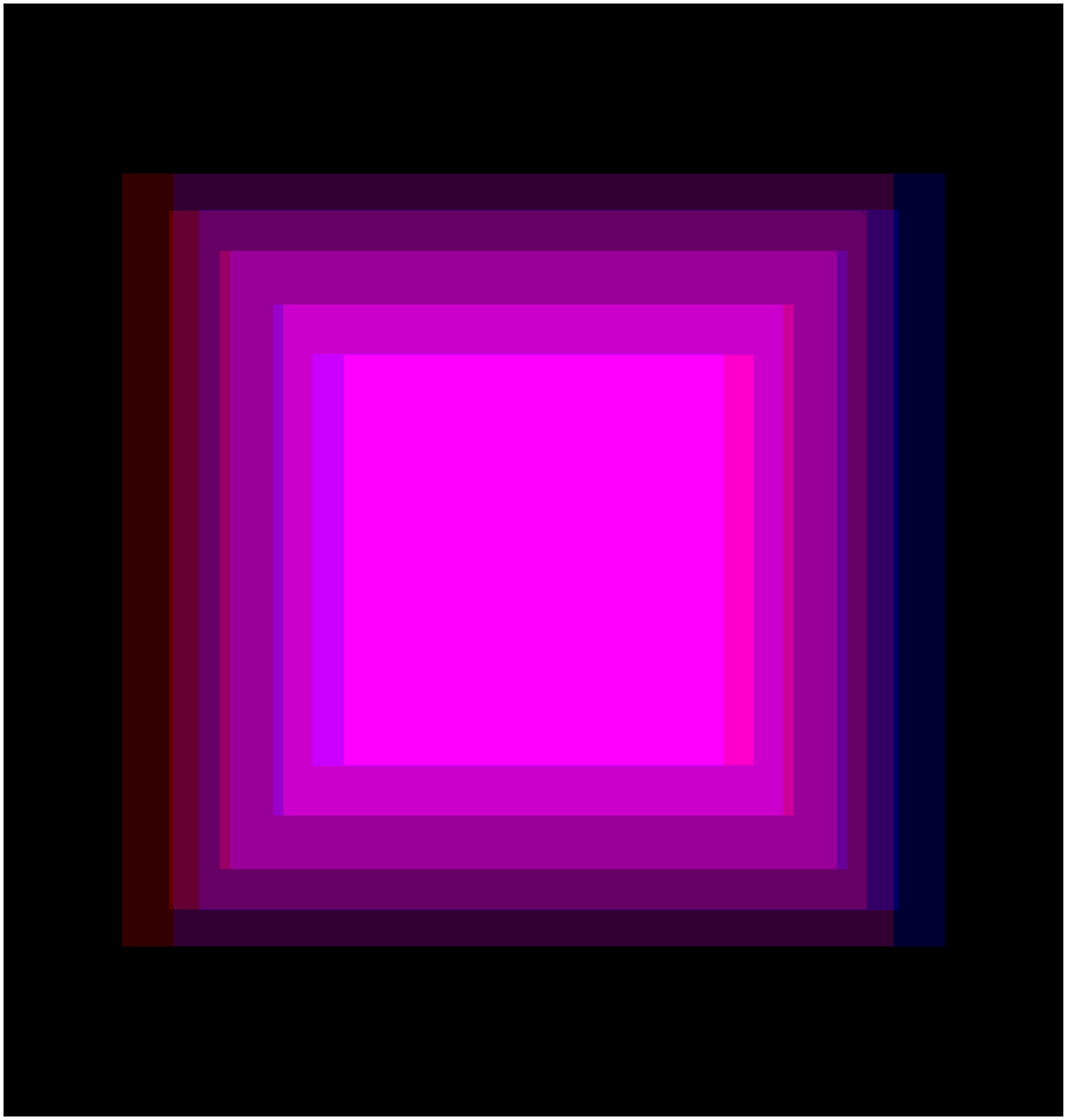

Figure 11: 3D Square Layers

Note that the input images are white squares of different sizes with a black background. The output image gives the illusion that there are multiple layers stacked over each other, some coming out of the screen. Next, the continuous 3D effect is applied to the following picture.

Figure 12: Original Image for 3D Effect

Figure 13: 3D Landscape

For this image, it seems like we are looking at a real scene because our brain processes it as if the road starts where we stand and continues to get farther and farther away. Even the hills in the back look like they are at different distances. Note that this effect does not work for any image. Two more images are provided to test, one of which is used in the last section.

C. 3D Audio:

As expected, the 3D audio effect worked, and the output audio seemed to be moving around from left to right. Below are the plots of the original and output audio.

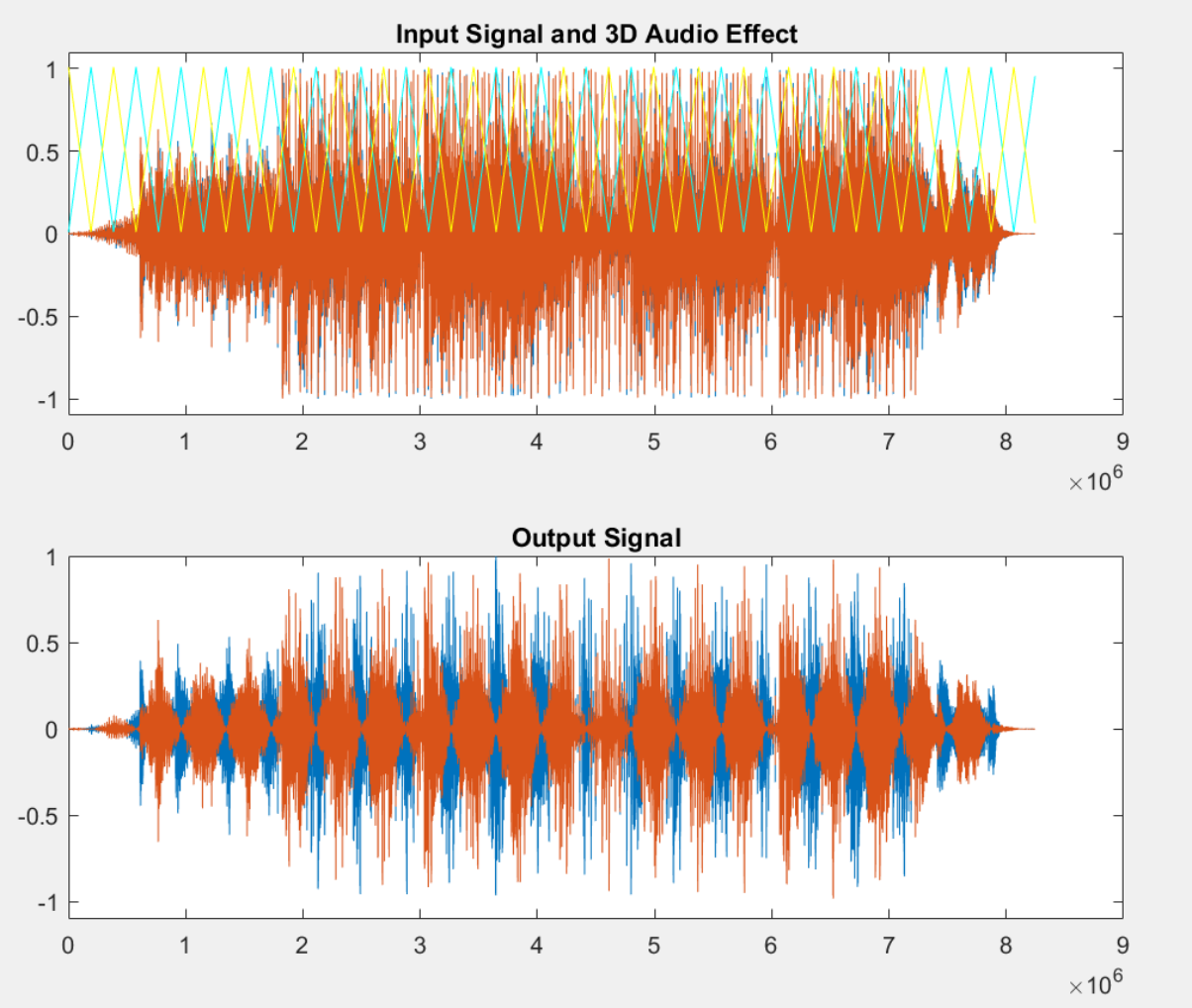

Figure 14: 3D Audio Input and Output

In the first subplot, you can see the original audio and the two triangular signals that are each multiplied with part of the original audio (left and right parts). The second subplot shows how one side increases as the other decreases (blue is left and red is right).

D. 3D Video:

The video also worked is it should; the center of the video moved inwards to and outwards from the screen according to the beat. The following are two frames from the video.

Figure 15: 1st Frame from the Video

Figure 16: 2nd Frame from the Video

V. Conclusion:

At the end, the project was successful and completely effective in all its parts.

The first part was to implement an image compression algorithm which used the Fast Fourier Transform to remove parts of the image in the frequency domain.

The second part was to create a constant and a continuous 3D effect. The results were as expected, and they verified the theoretical analysis.

The third part aimed to implement a 3D audio effect by multiplying the input with different triangular waves.

The fourth part was to combine the 3D audio and image filters in a video.

All results verified our theories and helped us understand new concepts in signal processing and image manipulation. The success of our project also means that one can advance these algorithms to achieve new systems that have outstanding outputs with only the sky as a limit.

VI. References:

[1]: MATLAB, https://www.mathworks.com/products/matlab.html

[2]: Wikipedia, https://en.wikipedia.org/wiki/Fast_Fourier_transform

[3]: Bou Dargham, N. (2022), EEN341 Lab Manual, Experiment 7, Fourier Series and Transforms

[4]: https://akshaysin.github.io/fourier_transform.html#.Y5N3F31Bw2w

To view or download the resources for this project, follow the link below:

https://nduedulb-my.sharepoint.com/:f:/g/personal/fbdimashki_ndu_edu_lb/Eq6gi8vBVapBr7TENBjK2nsBiFJwgzAKoj9nHtjlKMoIVA?e=GF5Llx