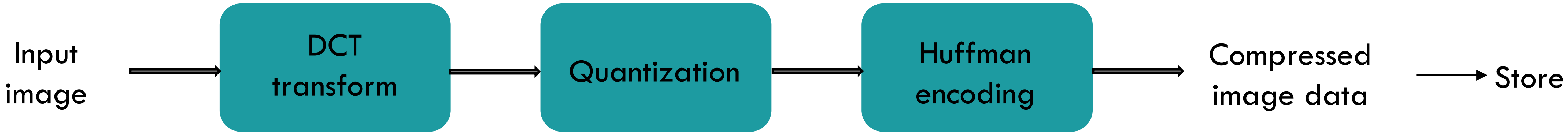
# VIDEO AND IMAGE CODEC

Ketheesram.N

# INTRODUCTION

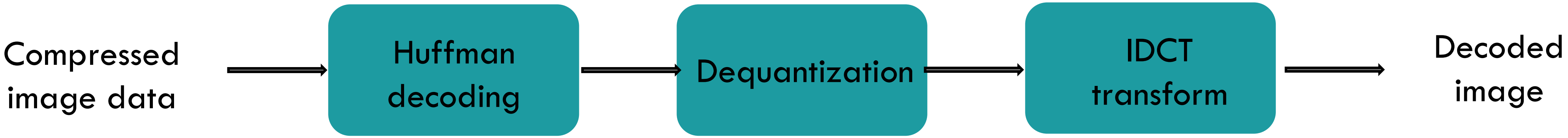
This is a simple hybrid image & video codec  implemented using Matlab with coding tools DCT2, iDCT2, quantization, motion estimation/compensation & entropy coding. The performance of the codec is optimized by adjusting those tools’ parameters.

# ENCODER FOR IMAGE CODEC



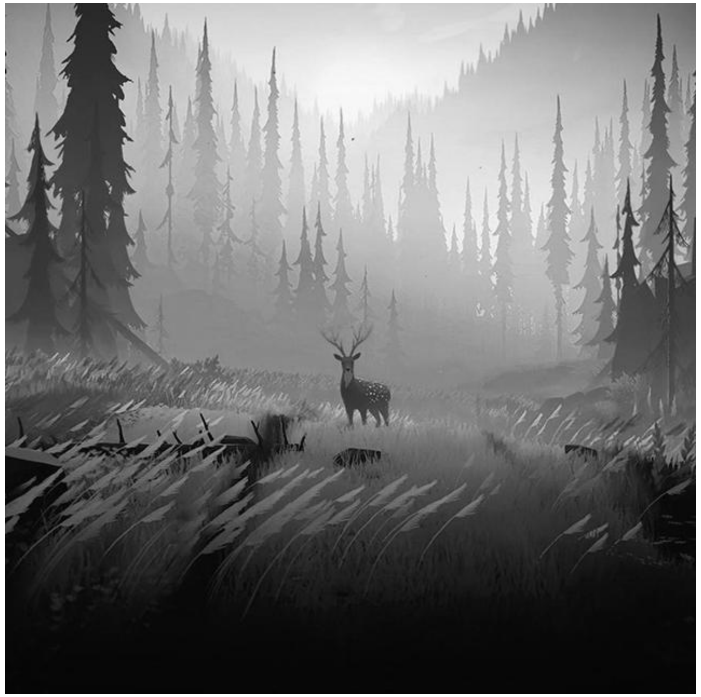
1. Load the input image and convert it to grayscale.
2. Define the 3 level quantization matrix.
3. Perform the forward transform (DCT) on the grayscale image.
4. Quantize the DCT coefficients using the quantization matrix.
5. Perform entropy encoding (Huffman coding) on the quantized coefficients.
6. Save the binary encoded data to a file.

# DECODER IMAGE CODEC



1. Load the Huffman encoded data and dictionary
2. Perform entropy decoding (Huffman decoding) on the binary encoded data.
3. Reconstruct the quantized coefficients.
4. Perform the inverse transform (IDCT) on the reconstructed quantized coefficients.
5. Calculate the compression ratio and PSNR (Peak Signal-to-Noise Ratio) between the original and reconstructed images.

# RESULTS OF IMAGE CODING



Gray Image(70.1kB)

Image After DCT applied

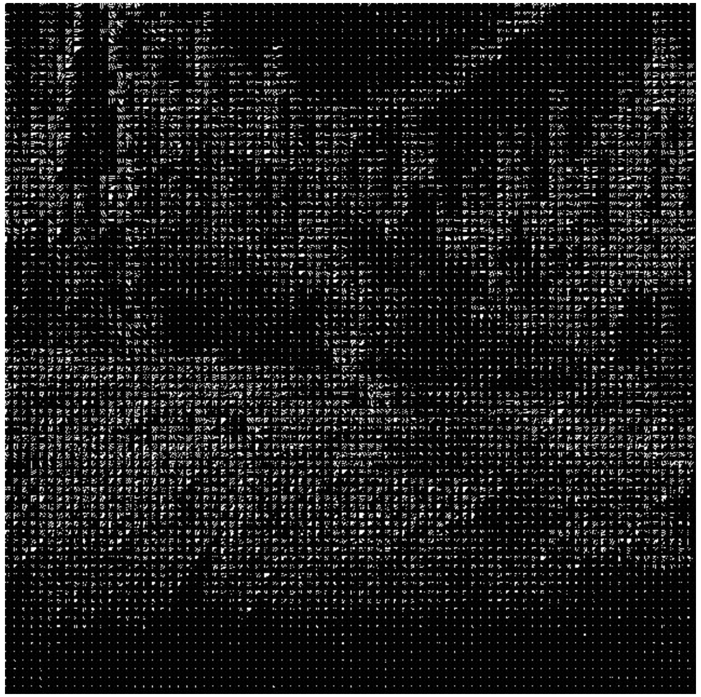
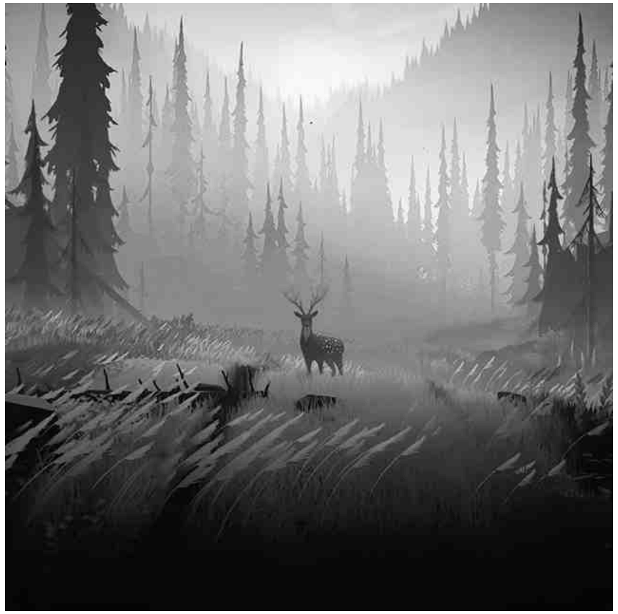
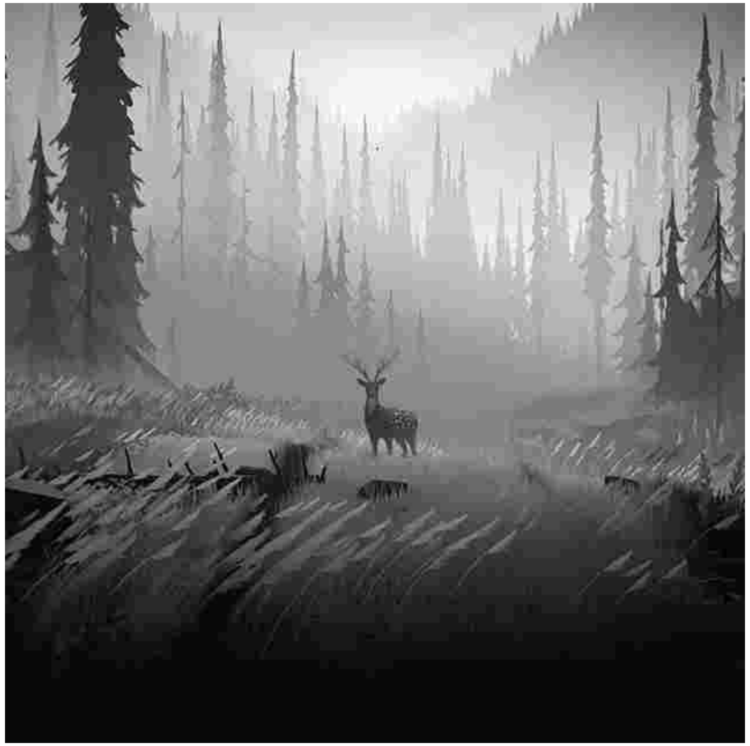


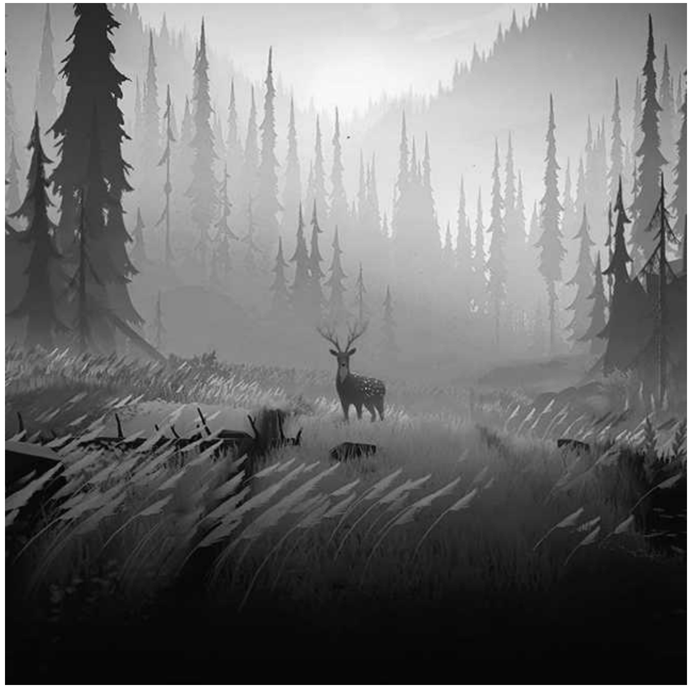
Image After quantization



MIN QUANT
Image size=68 kB



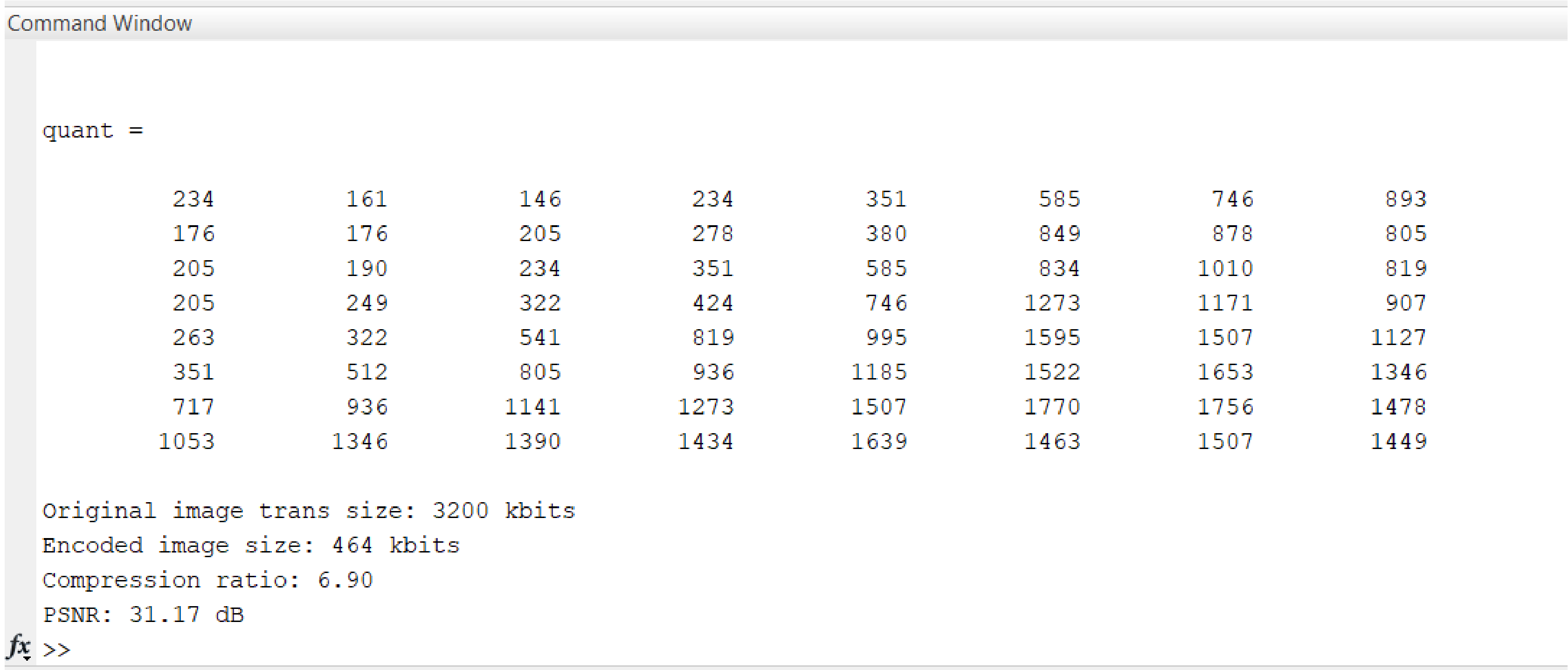
MAX QUANT
Image size=21.7 kB



MID QUANT
Image size= 44.1 kB

Setting the image bits size to transmit

Bit rate of the channel = 465 kbps



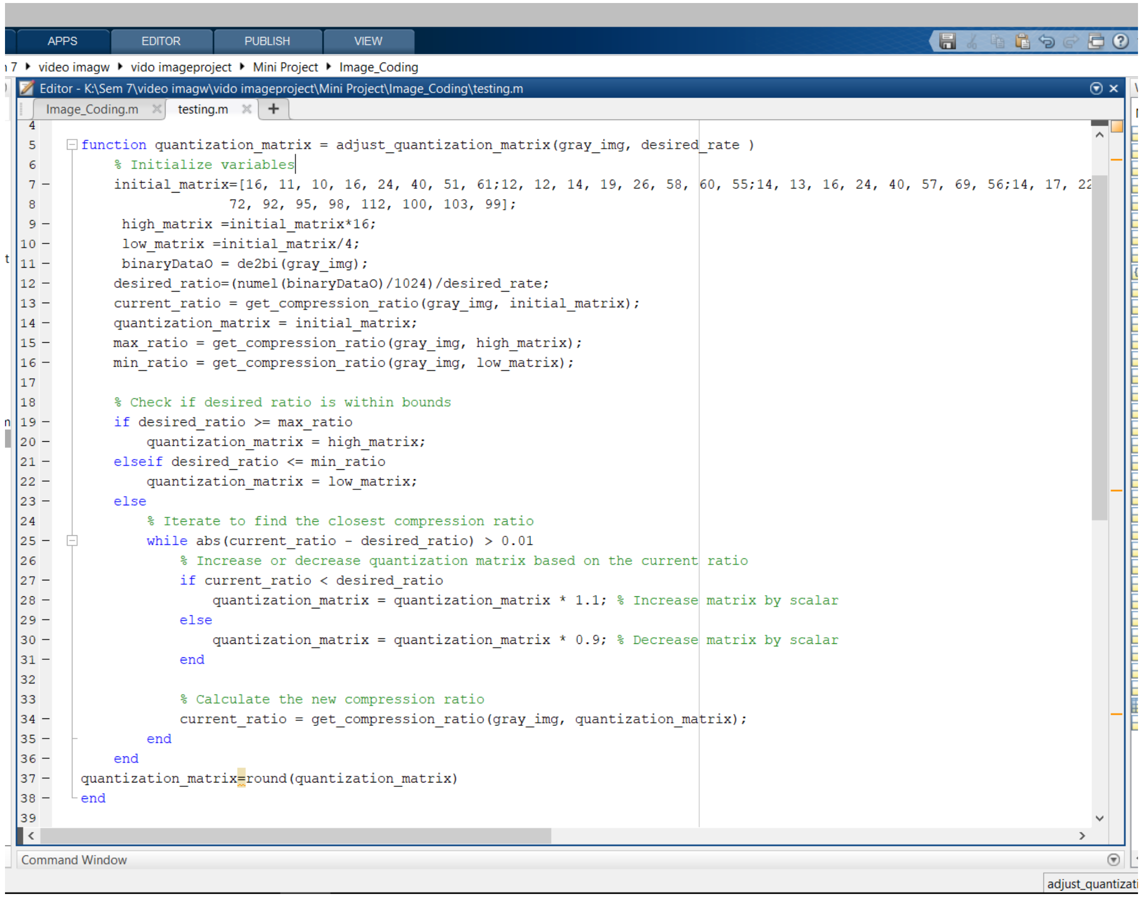
```
quant =

         234         161         146         234         351         585         746         893
         176         176         205         278         380         849         878         805
         205         190         234         351         585         834        1010         819
         205         249         322         424         746        1273        1171         907
         263         322         541         819         995        1595        1507        1127
         351         512         805         936        1185        1522        1653        1346
         717         936        1141        1273        1507        1770        1756        1478
        1053        1346        1390        1434        1639        1463        1507        1449

Original image trans size: 3200 kbits
Encoded image size: 464 kbits
Compression ratio: 6.90
PSNR: 31.17 dB
>>
```

# ITERATIVE METHOD

```
Initialize variables
Calculate desired,current compression ratio
If desired ratio is greater than or equal to maximum ratio
        quantization_matrix = high_matrix
    Otherwise
        quantization_matrix = low_matrix
Else
    While current ratio is not close enough to desired ratio
        If current ratio is less than desired ratio
            quantization_matrix = quantization_matrix * 1.1
        Otherwise
            quantization_matrix = quantization_matrix * 0.9
        Calculate new compression ratio
    End while
End if
Round quantization matrix
```


APPS
EDITOR
PUBLISH
VIEW
7 ▸ video imagw ▸ vido imageproject ▸ Mini Project ▸ Image_Coding
Editor - K:\Sem 7\video imagw\vido imageproject\Mini Project\Image_Coding\testing.m
Image_Coding.m
testing.m
Command Window
adjust_quantizati

```
function quantization_matrix = adjust_quantization_matrix(gray_img, desired_rate )
    % Initialize variables
    initial_matrix=[16, 11, 10, 16, 24, 40, 51, 61;12, 12, 14, 19, 26, 58, 60, 55;14, 13, 16, 24, 40, 57, 69, 56;14, 17, 22
                  72, 92, 95, 98, 112, 100, 103, 99];
     high_matrix =initial_matrix*16;
     low_matrix =initial_matrix/4;
     binaryDataO = de2bi(gray_img);
    desired_ratio=(numel(binaryDataO)/1024)/desired_rate;
    current_ratio = get_compression_ratio(gray_img, initial_matrix);
    quantization_matrix = initial_matrix;
    max_ratio = get_compression_ratio(gray_img, high_matrix);
    min_ratio = get_compression_ratio(gray_img, low_matrix);

    % Check if desired ratio is within bounds
    if desired_ratio >= max_ratio
        quantization_matrix = high_matrix;
    elseif desired_ratio <= min_ratio
        quantization_matrix = low_matrix;
    else
        % Iterate to find the closest compression ratio
        while abs(current_ratio - desired_ratio) > 0.01
            % Increase or decrease quantization matrix based on the current ratio
            if current_ratio < desired_ratio
                quantization_matrix = quantization_matrix * 1.1; % Increase matrix by scalar
            else
                quantization_matrix = quantization_matrix * 0.9; % Decrease matrix by scalar
            end

            % Calculate the new compression ratio
            current_ratio = get_compression_ratio(gray_img, quantization_matrix);
        end
    end
quantization_matrix=round(quantization_matrix)
end
```

# VIDEO CODING

**Video compression** is the process of reducing the total number of bits needed to represent a given image or **video** sequence.

Here I used 10 frames I and 9  p frames

Redundancy remove

- ✓Spatial redundancy – up sampling, down sampling
- ✓Temporal redundancy
- ✓Source coding redundancy

# ENCODER

```
Start
 Read video frames ,resize ,gray conversion and Macroblock division
      For each frame:
            Calculate motion vectors using SAD and stored
            Calculate residual by subtracting previous frame

     For each frame:
            apply down sampling
            Encode the residuals using quantization and huffman
 Store the dictionary and encoded residual
End
```

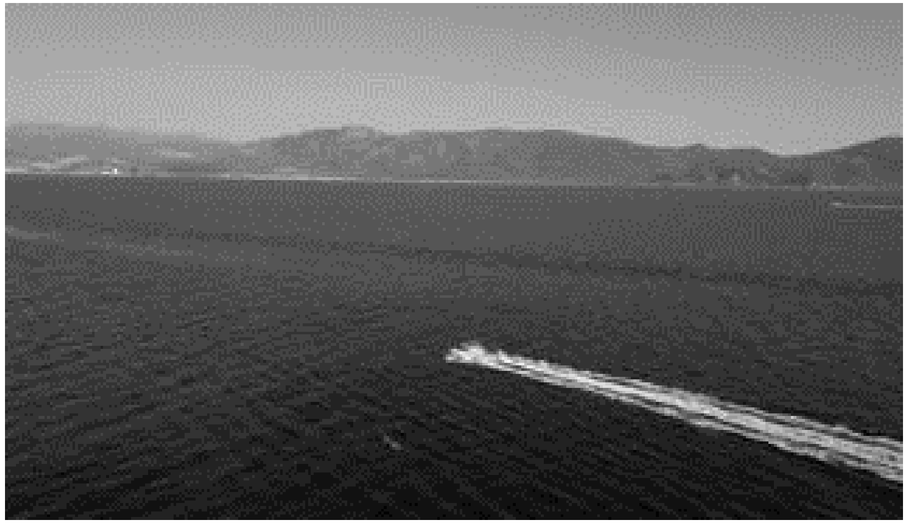
Residual frame_2

# DECODER

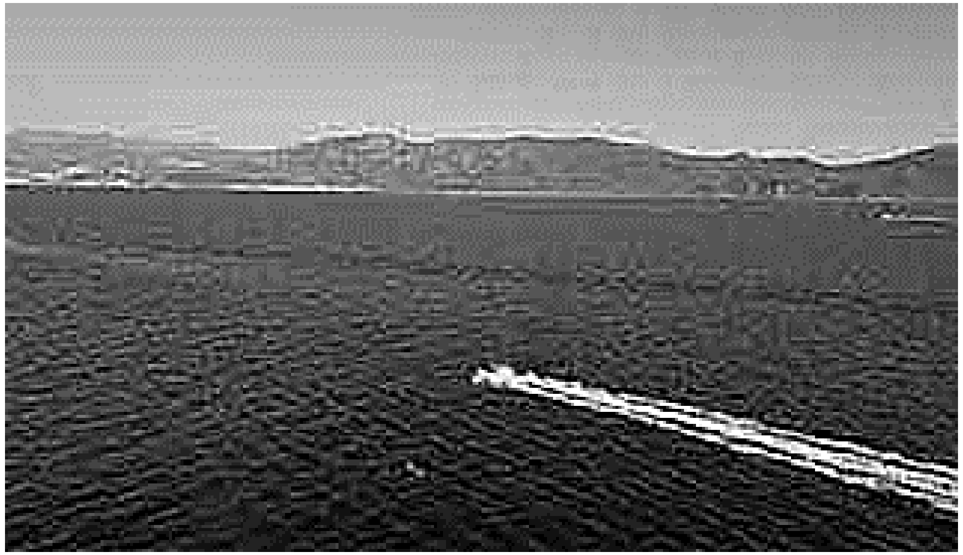
Predicted frame_2

```
Start
   Load encoded data, dictionary and motion vectors
   Initialize variables and parameters
   Read encoded residuals from text file
        For each frame:
                 Decode the residual & I using dictionary and quantization table
                 Store the decoded residual,I
                 Apply up sampling
For each frame:
            Calculate predict frame using motion vectors and previous frame
            Store the predicted frame
            Reconstruct final frames by adding decoded residuals to predicted frames
 Apply deblocking filter to reconstructed images
 Save filter applied reconstructed frames as a video
End
```

# RESULTS



Original Video(10 Frames)

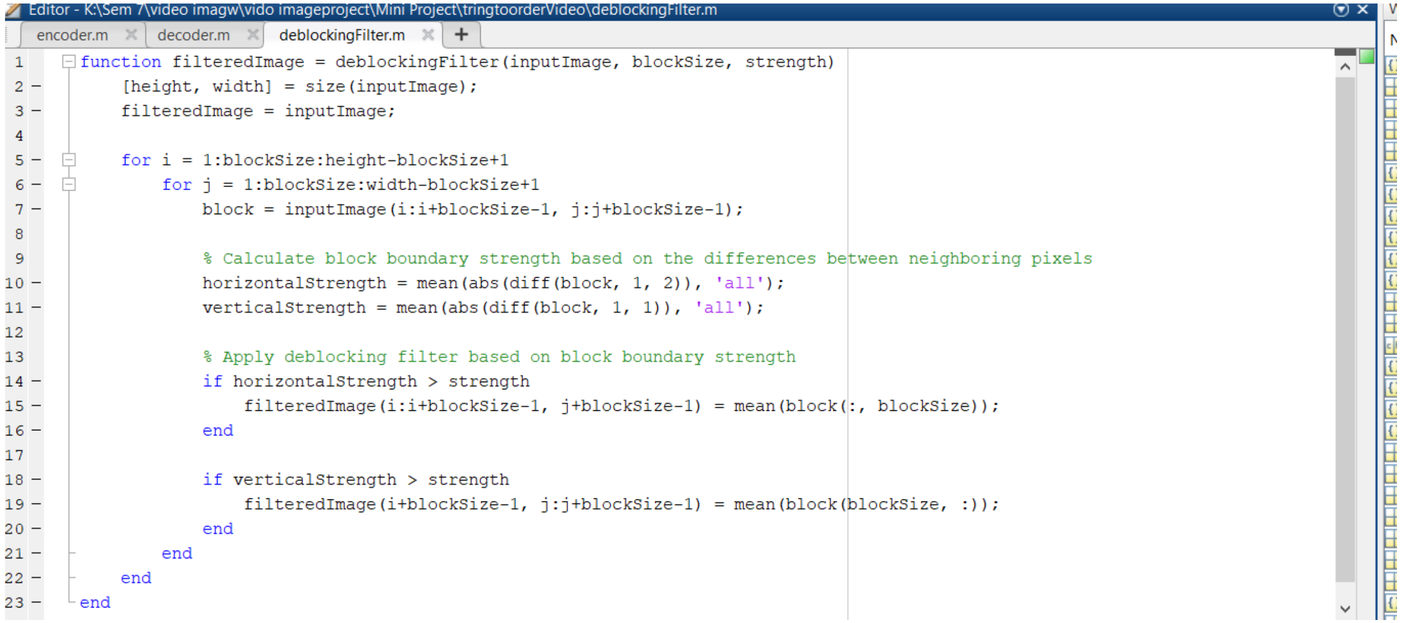


Compressed Video(10 Frames)

# OPTIMIZATION

- To remove spatial redundancy did downsampling(encoder) and upsampling(decoder)
- Improve quality using Deblocking filters

  A deblocking filter is a video filter applied to decoded compressed video to improve visual quality and prediction performance by smoothing the sharp edges



Editor - K:\Sem 7\video imagw\vido imageproject\Mini Project\tringtoorderVideo\deblockingFilter.m
encoder.m
decoder.m
deblockingFilter.m
+

```
function filteredImage = deblockingFilter(inputImage, blockSize, strength)
    [height, width] = size(inputImage);
    filteredImage = inputImage;

    for i = 1:blockSize:height-blockSize+1
        for j = 1:blockSize:width-blockSize+1
            block = inputImage(i:i+blockSize-1, j:j+blockSize-1);

            % Calculate block boundary strength based on the differences between neighboring pixels
            horizontalStrength = mean(abs(diff(block, 1, 2)), 'all');
            verticalStrength = mean(abs(diff(block, 1, 1)), 'all');

            % Apply deblocking filter based on block boundary strength
            if horizontalStrength > strength
                filteredImage(i:i+blockSize-1, j+blockSize-1) = mean(block(:, blockSize));
            end

            if verticalStrength > strength
                filteredImage(i+blockSize-1, j:j+blockSize-1) = mean(block(blockSize, :));
            end
        end
    end
end
```

# THANK YOU