SGN-26006 Advanced Signal Processing Laboratory
Data Compression

Emanuele Palma (**Exercise Instructor**)
Petri Helin, Pekka Astola and Ioan Tabus
firstname.lastname@tut.fi

September 14, 2018

The data compression exercise is about studying the compression of image data using a simple causal predictor and Golomb-Rice coding. The image data is extracted from Point Cloud file format. The input data for the exercise are a matrix **RGB** containing RGB light intensity in a given projection and a matrix **Depth** containing the corresponding depth map.

1 Tasks

1. The process of extracting a given projection view (color view and depthmap) is described in the Annex starting after page 3 of this document. Follow the instructions and execute the tasks mentioned in the Annex. The extracted color image **RGB** is transformed into a grayscale matrix and is denoted as **A**.

2. Draw the histogram of the values found in the matrix **A**. Use the collected statistics to compute the empirical entropy.

3. Apply to **A** the predictor presented below using a row-wise scan order and compute the residual matrix **E**.

   The predictor is estimating the value $z = A(i,j)$ of the current pixel position (i,j) in **A**, using the values of the northern (N), western (W), northwestern (NW), and northeastern (NE) pixel position: $(i-1,j), (i,j-1), (i-1,j-1), (i-1,j+1)$, in the causal neighborhood of the current pixel. If we denote $n = A(i-1,j), w = A(i,j-1), nw = A(i-1,j-1), ne = A(i-1,j+1)$, then the predictor estimates z as

   $$\hat{z} = \text{median}\{n, w, n+w-nw, w-n+ne, w+\frac{ne-nw}{2}\}.$$

   The residual error is computed as $z - \hat{z}$ and is saved as $E(i,j)$.

   For some pixel positions the causal neighborhood does not contain all of the values used by the predictor, and hence the predictor is using only the currently available values.

   For the value found at the pixel position $(0,0)$ there is no prior information and it will be encoded using its 8 bits representation.

4. Divide the matrix **E** into blocks of size $b \times b$ and collect in a vector e all the values found in the block using a row-wise scan.

5. Each vector e is compressed using the class of codes known as Golomb-Rice codes. These codes are parametrized by an integer parameter p (a number of bits). Each integer value

is split into the sign, the p least significant bits, and the remaining most significant bits. The sign bit and the p least significant bits are coded raw, and the value represented by the most significant bits is coded in unary.

For example, if we have $p = 3$ and we want to code $x = 37$, we split 37 into the sign bit (0), the 3 least significant bits ($37\&(2^3-1) = 5$) and the most significant bits ($\lfloor 37/(2^3) \rfloor = 4$). The codeword associated with the value will therefore be 0, 101, 00001 (4 in unary).

For each block of $b \times b$ error samples, an optimal parameter p must be found. It is clear that the codeword length for coding a value x is $1+p+(\lfloor x/2^p \rfloor+1)$, therefore the optimum p may be computed by checking all the p values from 0 to 8, and choosing the one which produces the smallest length in bits for the block. For more efficient search, it may be used that the optimum parameter p is close to $\log_2$ of the mean of the absolute value in the block.

The bits should be written to the binary compressed file and read back with the predefined functions **fwrite()** and **fread()** using the ubit1 or ubitN (in some cases) precision specifiers.

6. Write two top-level MATLAB functions. The first is **lf_compress()** which takes as an arguments the input matrix **A**, the destination file (compressed file), and the block size for the Golomb-Rice coding (b, in the range $32 \ldots 128$). The second is **lf_decompress()** which takes as an argument the source file (compressed file) and returns the reconstructed matrix.

7. Use the **lf_compress()** function several times and try different block sizes and build a combined graph for the compressed size results. Explain the influence of the block size b on the compressed size.

8. Verify for each compressed file produced using the **lf_decompress()** function that the decompression process losslessly restores the original file.

9. Use the same procedure presented above (steps 2-8) and compress the matrix **Depth**, where this time at Step 2 use the following procedure to quantize the depth values. Quantize **Depth** as follows:

$$\mathbf{D} = \lfloor 255 \cdot \frac{\mathbf{Depth} - minD}{maxD - minD} \rfloor$$

where $maxD, minD$ are the maximum and minimum depth values found in the matrix **Depth**.

2 Submitting results

You must send by email a **ZIP** archive, including the report in .pdf format and commented MATLAB code used for solving the exercise, no later than **20th of December 2018**. The report should contain at least the following information:

- All plots required in the Annex
- Plots with the histograms at point 2
- A short explanation how you wrote the parameter p for each block into the compressed bit stream and what additional side information is also required in the compressed bitstream for successful lossless decoding of the compressed file.

- The graph with the compressed sizes obtained at point 7 and what block size $b \times b$ produces the best compression.
- Plot the residual error matrix **E**. What can you notice? How is this affecting compression results.
- Create a new algorithm by using the following procedure:
  - Encode the depth matrix using the best block size value.
  - Create a different segmentation by quantizing the depth matrix: divide the range $0, \ldots, 255$ into equal or unequal ranges e.g. instead of using 256 levels to represent the depth define a different set of n_l levels, where $n_l = 128, 64, 32, \ldots$. All the neighboring values in a 4-connectivity will be included in the same region if they have the same value
  - Encode the intensity matrix by using the previously generated segmentation instead of the block partition.
  - Test different segmentations and find the best option.
  - Is the new method better than the previous block based one? Why?
- Write a paragraph about your proposal to improve the compression algorithm proposed in this document.
- Compress the images with at least one other lossless image codec (JPEG2000, PNG, GIF, TIFF etc.) using MATLAB **imwrite(), imread()** and compare the results.

You must send also the commented MATLAB code for solving the exercise. It has to include the two .m files **lf_compress.m** and **lf_decompress.m** which may be called independently from the MATLAB prompt.

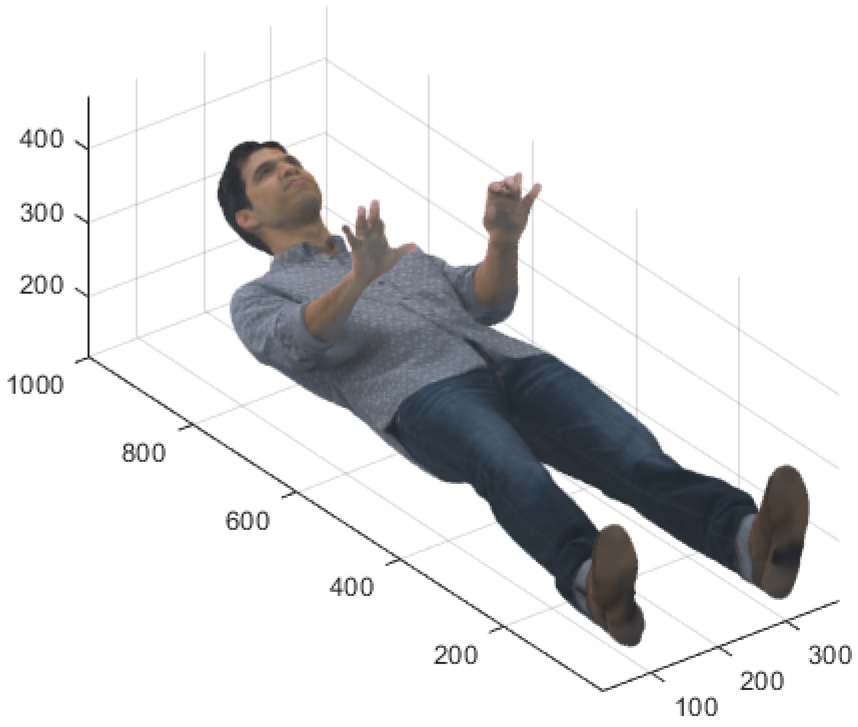
Table of Contents

```
clear all
close all
```

Task 1. Initialization and Acquisition of the Point Cloud

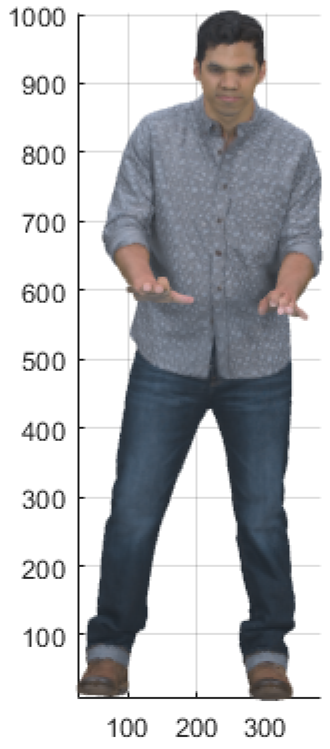
```
filename = 'S:\81201_IMSIM\POINT_CLOUDS\AdvSPLab\loot_vox10_1001.ply';
filename = 'loot_vox10_1001.ply'
URL = 'http://www.cs.tut.fi/~tabus/course/AdvSP/loot_vox10_1001.ply'
A = urlread(URL);
fid = fopen(filename,'w')
fwrite(fid,A,'char');
fclose(fid);

ptCloud = pcread(filename);
figure(1),PC1_img = pcshow(ptCloud);
```

```
filename =

    'loot_vox10_1001.ply'


URL =

    'http://www.cs.tut.fi/~tabus/course/AdvSP/loot_vox10_1001.ply'


fid =

     3
```



Task 1a. Orientation and Manipulation of the Point Cloud

```
% Use the figure toolbar (open it from Figure 1 Menu --> View -->
 Figure Toolbar, if it's not open already)
% Pick the Rotate 3D button and rotate as much as you think is
 necessary to
% understand the structure of the point cloud
% Extract the 3 major projections (Front, Top and 1 Side Projection of
 your
% choice [L/R]), show them, save them to your hard drive and include
 them
% in the report.

PC1_img.View = [0 90]; %First Projection (Front)


%Inspect the structure of the point cloud and then, subsequentially,
 save
%the geometry and the colors in 2 indipendent variables:
BB = ptCloud.Location(:,1:3); %Geometry
CC = ptCloud.Color(:,1:3);
```

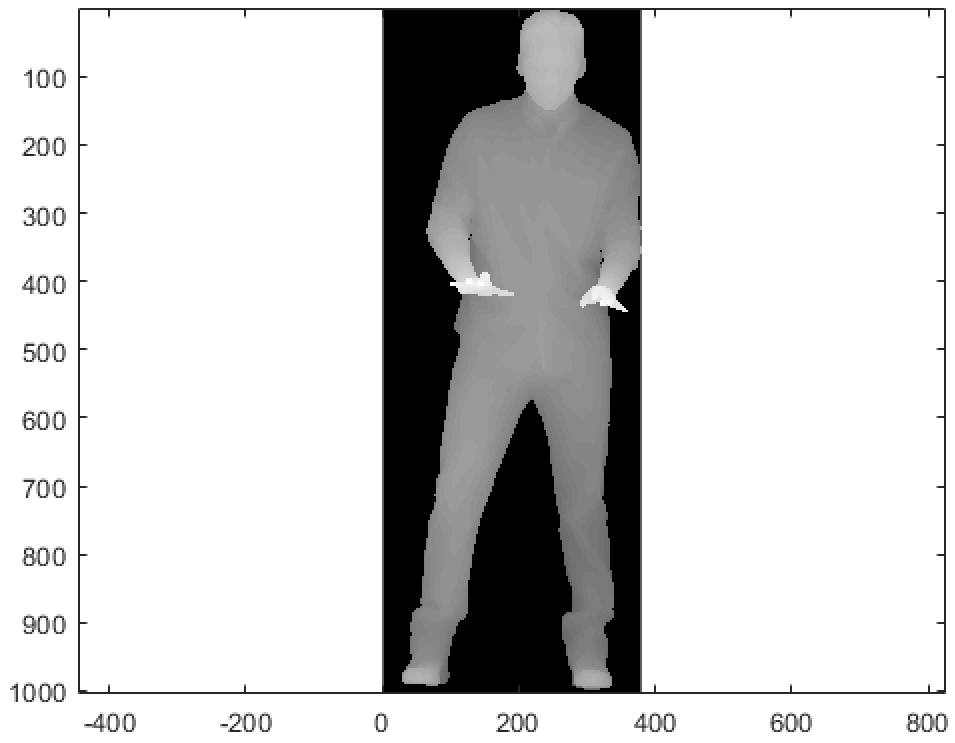


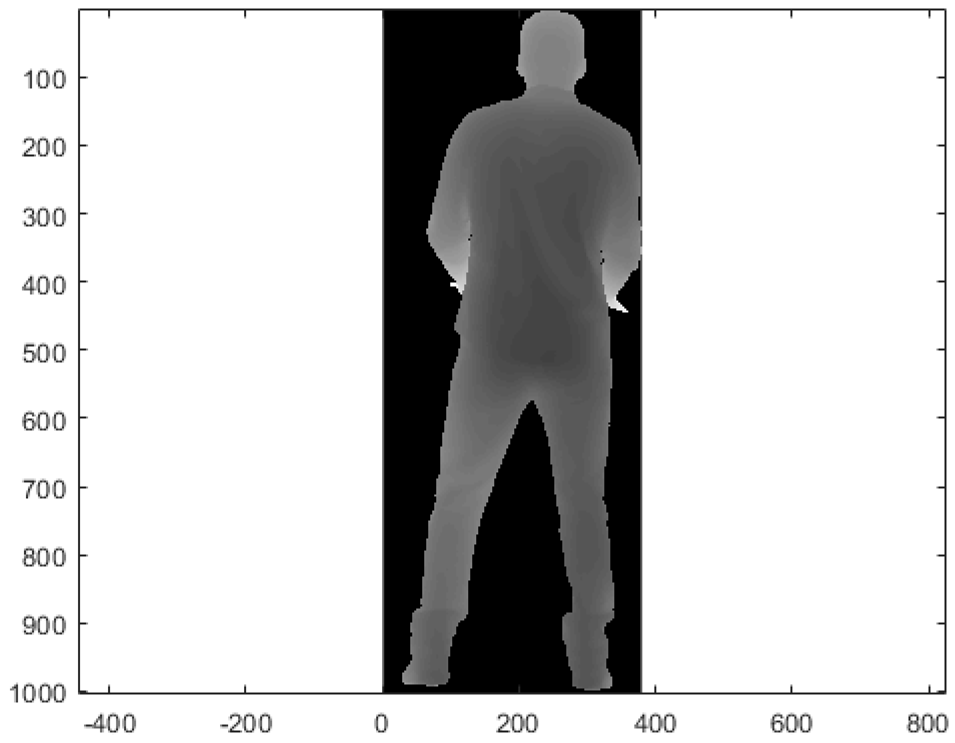
Task 2 Extract the Assigned Projection

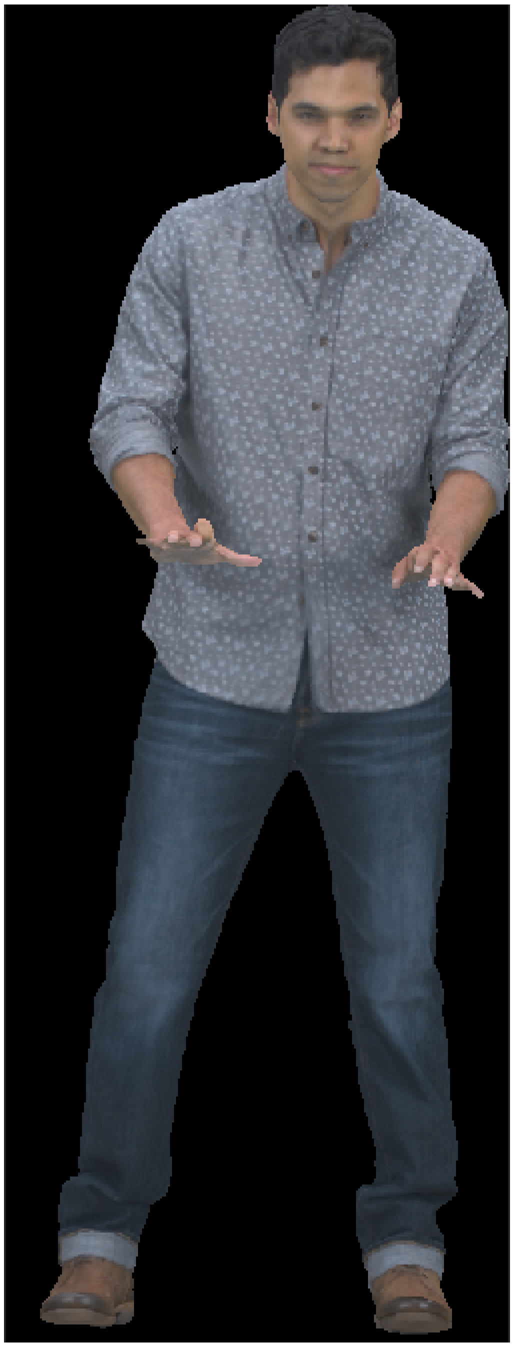
```
% If your Student Number's last digit is odd --> Left/Right Projection
% If your Student Number's last digit is even --> Top/Bottom
 Projection

% For both cases you also have to display them and save them to your
 hard drive and
% include them in the report.

% Example for the extraction of the front/back projection

FP_BB1 = BB(:,1);
FP_BB2 = BB(:,2);
FP_BB3 = BB(:,3);

MaskFP_1 = zeros(length(FP_BB1),1);
MaskFP_2 = zeros(length(FP_BB1),1);
MaskFP_3 = zeros(length(FP_BB1),1);

FP_ImRepMax = zeros(max(FP_BB2(:)),max(FP_BB1(:)));
FP_ImRepMin = 10^10*ones(max(FP_BB2(:)),max(FP_BB1(:)));
FP_ImRepCell = cell(max(FP_BB2(:)),max(FP_BB1(:)));
FP_ImRepCard = zeros(max(FP_BB2(:)),max(FP_BB1(:)));
FP_ImColForgr = zeros(max(FP_BB2(:)),max(FP_BB1(:)),3);
```

```
FP_ImColBackgr = zeros(max(FP_BB2(:)),max(FP_BB1(:)),3);
for i4 = 1:length(FP_BB1)
    if( FP_ImRepMax( FP_BB2(i4),FP_BB1(i4) ) < FP_BB3(i4) )
        FP_ImRepMax( FP_BB2(i4),FP_BB1(i4) ) = FP_BB3(i4);
        FP_ImColForgr( FP_BB2(i4),FP_BB1(i4),1:3) = CC(i4,:);
    end
     if( FP_ImRepMin( FP_BB2(i4),FP_BB1(i4) ) > FP_BB3(i4) )
        FP_ImRepMin( FP_BB2(i4),FP_BB1(i4) ) = FP_BB3(i4);
        FP_ImColBackgr( FP_BB2(i4),FP_BB1(i4),1:3) = CC(i4,:);
     end
    FP_ImRepCell{FP_BB2(i4),FP_BB1(i4)} =
 [FP_ImRepCell{FP_BB2(i4),FP_BB1(i4)} FP_BB3(i4)];
    FP_ImRepCard(FP_BB2(i4),FP_BB1(i4)) =
 length( FP_ImRepCell{FP_BB2(i4),FP_BB1(i4)} );
end
% Mark the used points
for i4 = 1:length(FP_BB1)
    if( FP_ImRepMax( FP_BB2(i4),FP_BB1(i4) ) == FP_BB3(i4) )
        MaskFP_1(i4) = 1;
    end
     if( FP_ImRepMin( FP_BB2(i4),FP_BB1(i4) ) ==  FP_BB3(i4) )
        MaskFP_1(i4) = 2;
     end
end


FP_ImRepCard = FP_ImRepCard(end:-1:1,:);
FP_ImRepMax= FP_ImRepMax(end:-1:1,:);
FP_ImRepMin = FP_ImRepMin(end:-1:1,:); FP_ImRepMin( FP_ImRepMin ==
 10^10) = 0;
FP_ImColForgr = FP_ImColForgr(end:-1:1,:,:);
FP_ImColBackgr = FP_ImColBackgr(end:-1:1,:,:);

% Saving and Displaying Images

U8ver_FP_ImColForgr = uint8(FP_ImColForgr);
U8ver_FP_ImColBackgr = uint8(FP_ImColBackgr);

figure(14), imagesc(FP_ImRepMax), colormap(gray), axis equal
figure(15), imagesc(FP_ImRepMin), colormap(gray), axis equal
figure(18), imshow(U8ver_FP_ImColForgr), colormap(gray)
figure(19), imshow(U8ver_FP_ImColBackgr), colormap(gray)

vv = [sum(FP_ImRepMax(:)>0) sum(FP_ImRepMin(:)>0)
 sum(FP_ImRepMax(:)>0)+sum(FP_ImRepMin(:)>0)  length(FP_BB1)];

imwrite(U8ver_FP_ImColForgr, 'U8ver_FP_ImColForgr.png');
imwrite(U8ver_FP_ImColBackgr, 'U8ver_FP_ImColBackgr.png');

%At the end of this taks, you'll need to work with only one projection
 until the end of the assignment:

% Top/Bottom has to work with only the Top Projection
% Left/Right has to work with only the Left Projection
```

```
Warning: Image is too big to fit on screen; displaying at 67%
Warning: Image is too big to fit on screen; displaying at 67%
```



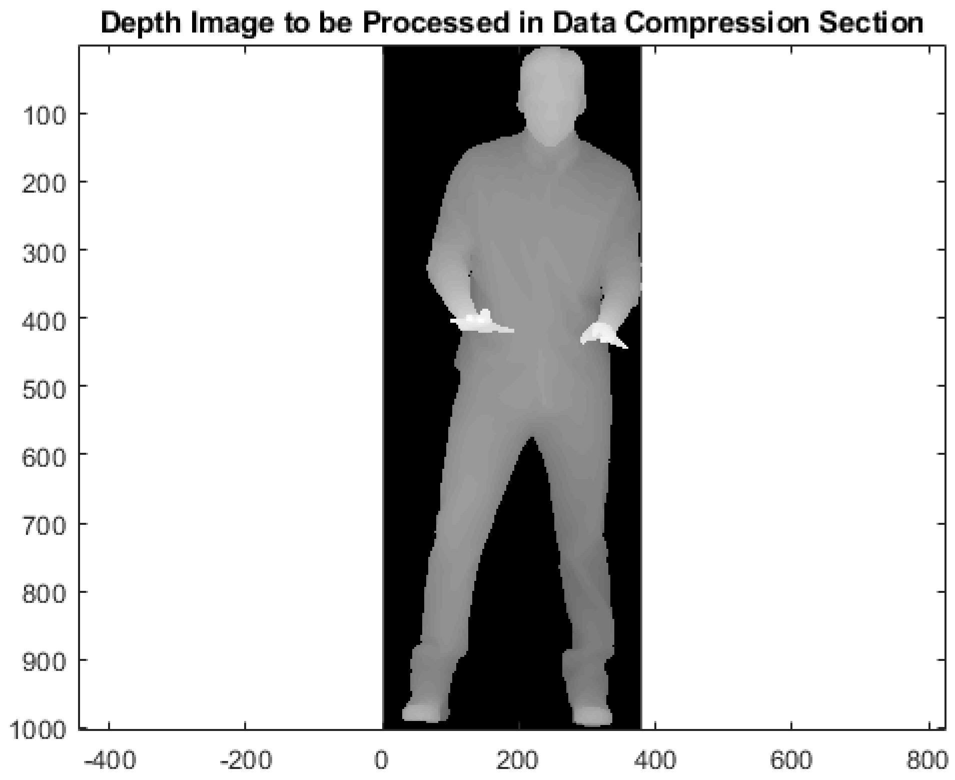

100
200
300
400
500
600
700
800
900
1000
-400
-200
0
200
400
600
800

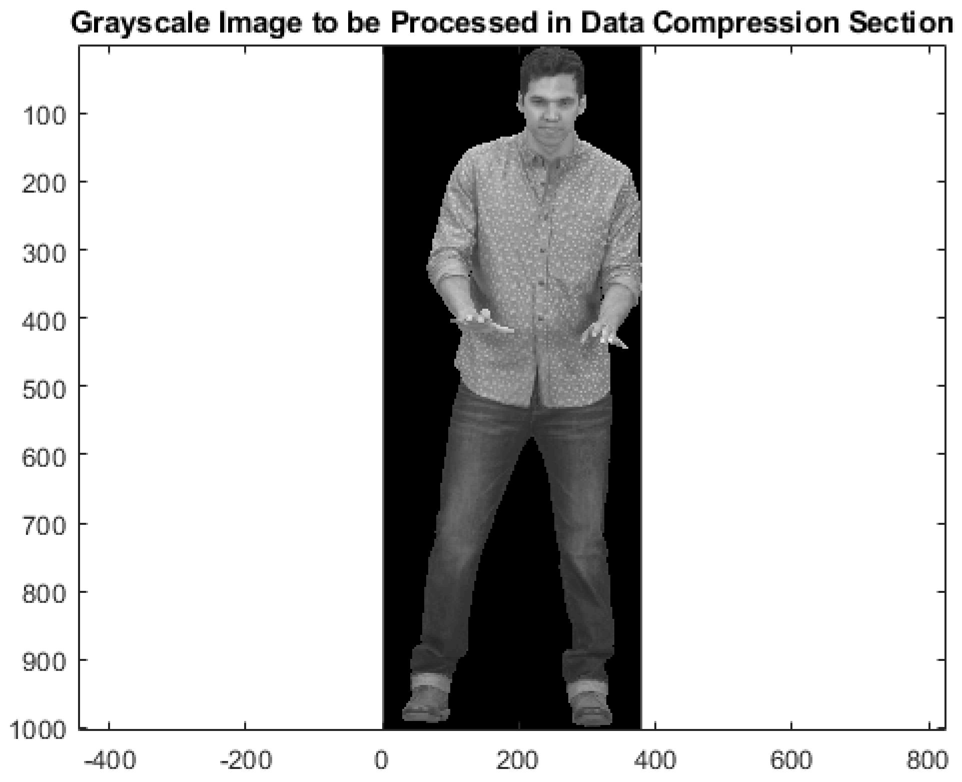


Task 3. Conversion in Grayscale

Save the Depth Image (ImRepMax) to a separate variable, display it and convert the acquired color image RGB into a grayscale matrix. Denote the new matrix A.

```
%Example for Front Projection (Change for your assigned Projection)

Depth = FP_ImRepMax;
figure(31), imagesc(Depth), colormap (gray), axis equal,title('Depth
 Image to be Processed in Data Compression Section')

Acolor = U8ver_FP_ImColForgr;

A = rgb2gray(Acolor);
figure(32),imagesc(A), colormap (gray), axis equal,title('Grayscale
 Image to be Processed in Data Compression Section')
```





Published with MATLAB® R2017b