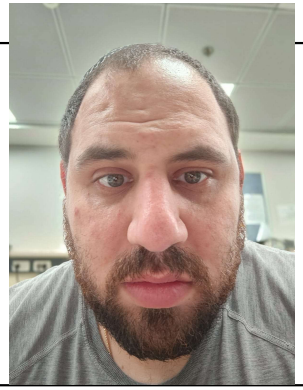
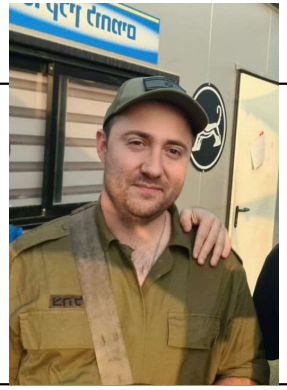
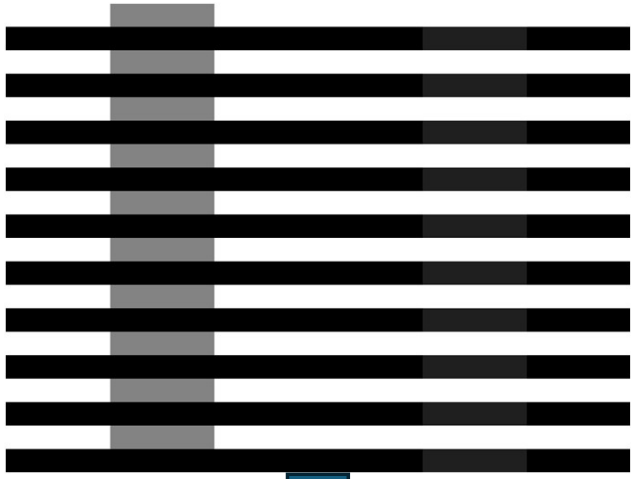
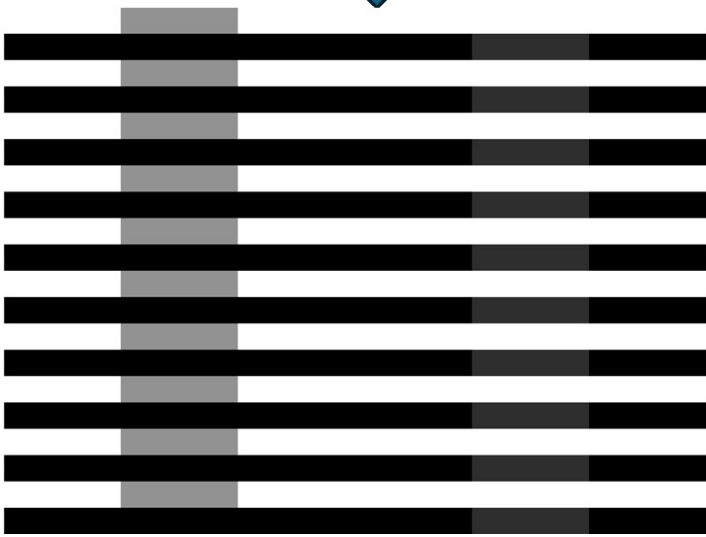
Course: Image Processing 31651
Assignment #12
Synthetic Image Creation (Part 2)

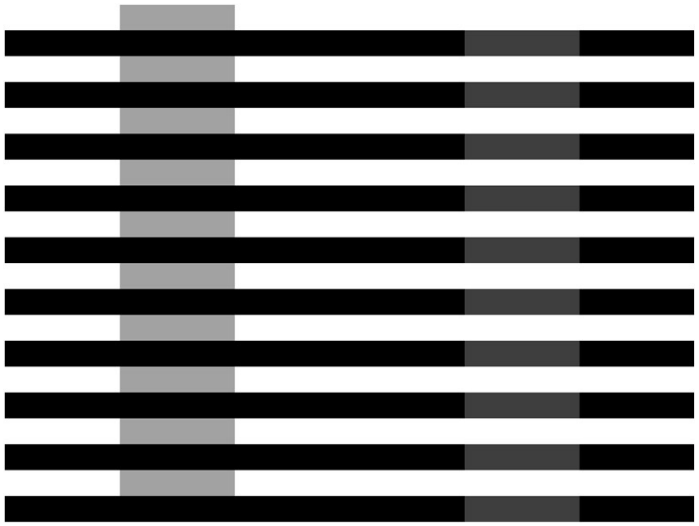
	ID (4 last digits)	Shorten Name	Photo of the student
Student #1	**1950**	shienfeld	
Student #2	**2210**	pony	
Student #3	**7939**	akimov	

12.1 our resulted images “grayImage12n.bmp” with add short comment– part 1

white_illusion0.bmp



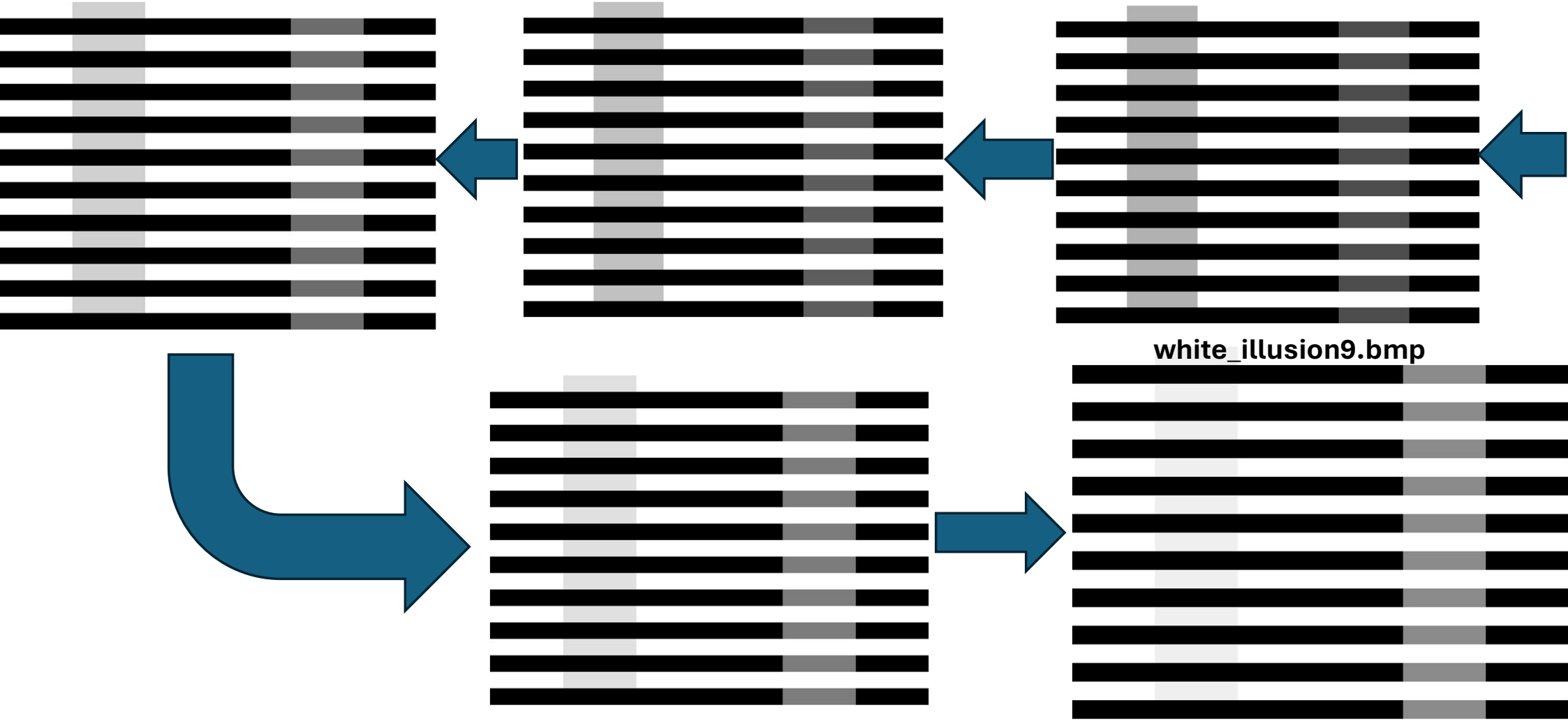




In that slide we can see the process (automate process) of creating a white illution – we decided to create a total of 10 pictures in the process (can be any number we would like)

The first image here is white_illusion0.bmp and the rest (until white_illusion9.bmp) is in the next slide

12.1 our resulted images “grayImage12n.bmp” with add short comment– part 2



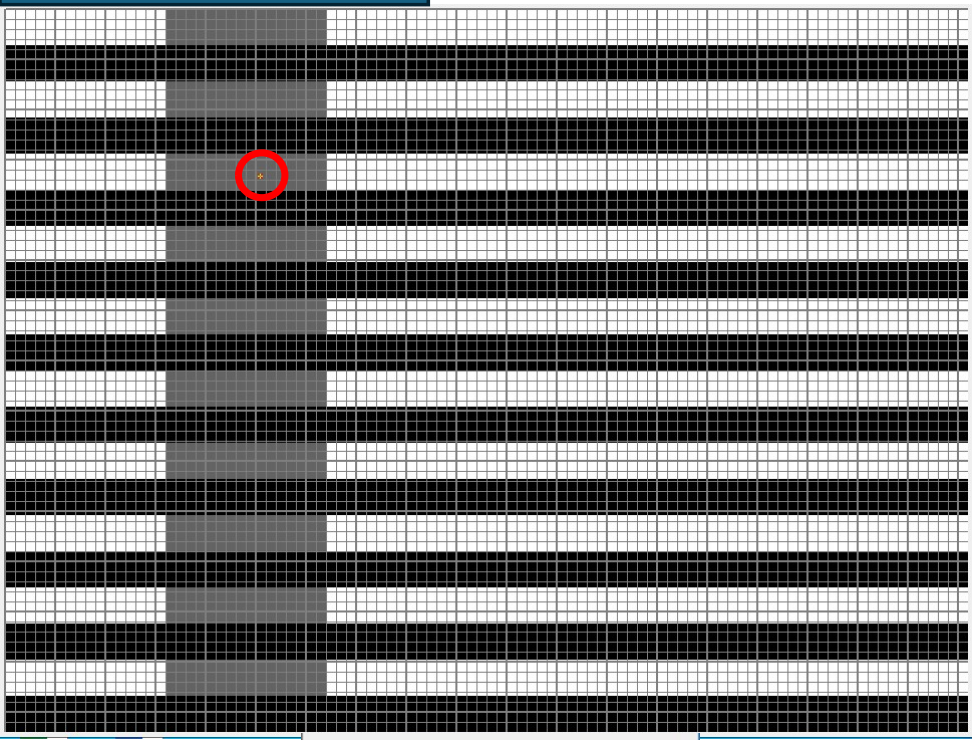
12.1 our resulted images “grayImage12n.bmp” with add short comment– part 3

white_illusion0.bmp

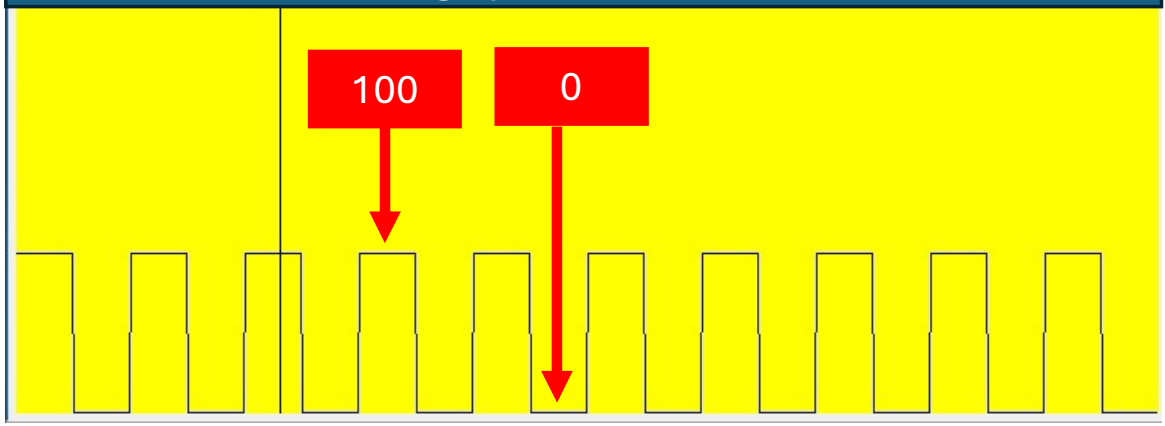
white_illusion9.bmp

12.2 Relevant Profiles – part 1

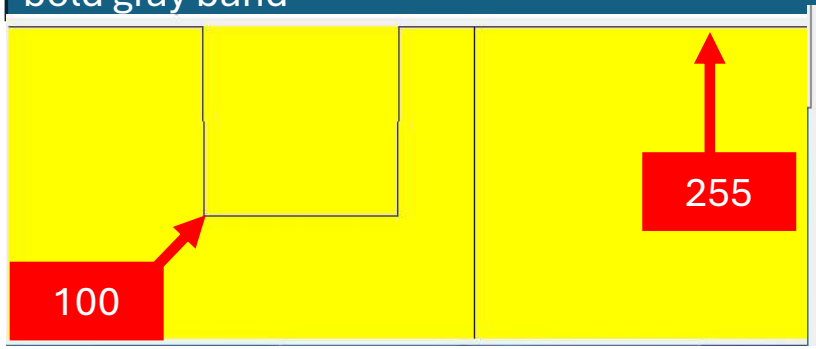
white_illusion0.bmp



In the columns view (AKA moving up and down on the same column) we can see a rectangular periodic function all over the column. That function is relevant for several columns – where we see the bold gray band!



In the rows view (AKA moving right and left on the same row) we can see a hole function (like a potential hole).
We understand that the low level of the hole represents the bold gray band



Point:	Mouse	Expl.	(M-E)
Row	409	409	0
Column	130	130	0
Red	255	255	0
Green	255	255	0
Blue	255	255	0

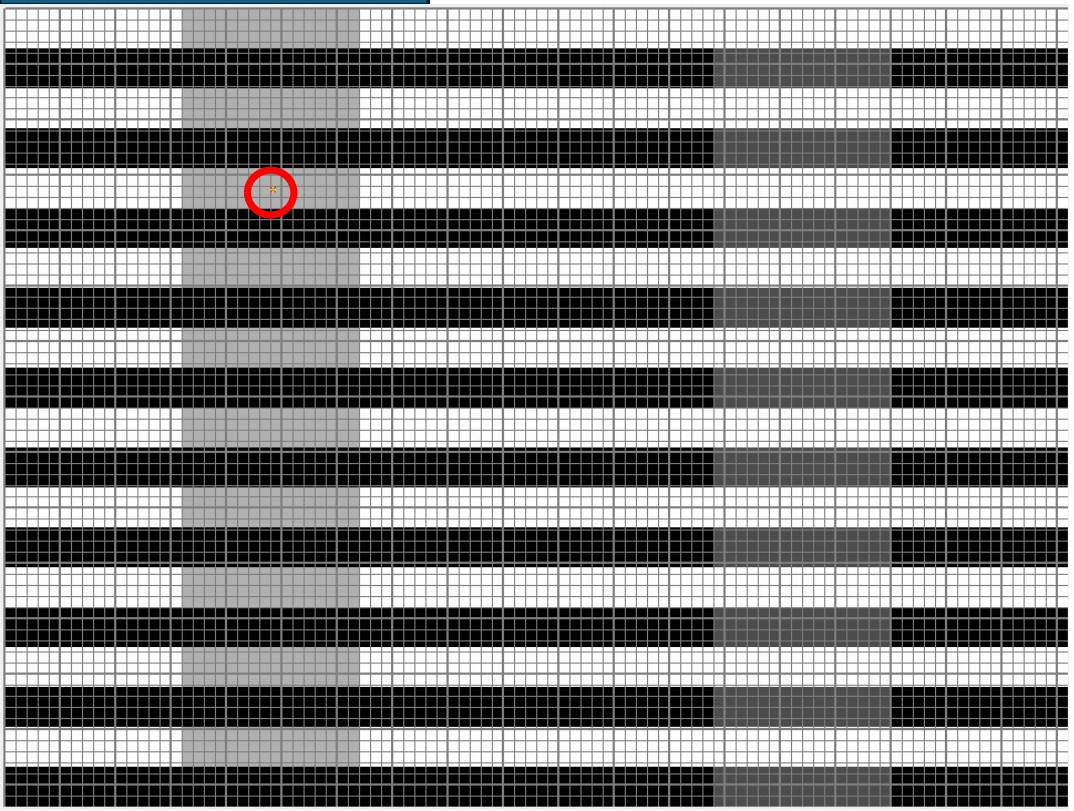
Regarding the column view

Point:	Mouse	Expl.	(M-E)
Row	372	372	0
Column	268	268	0
Red	0	0	0
Green	0	0	0
Blue	0	0	0

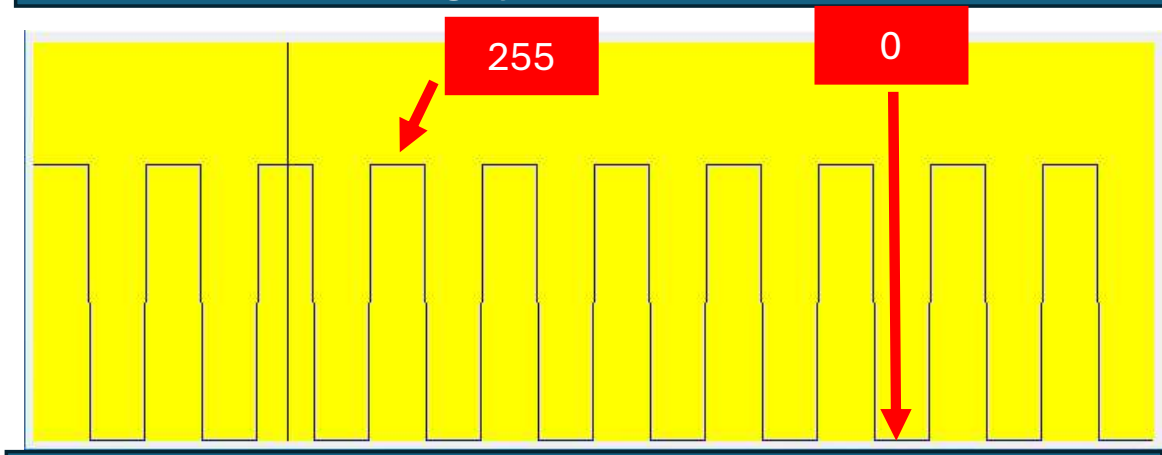
Point:	Mouse	Expl.	(M-E)
Row	407	407	0
Column	270	270	0
Red	100	100	0
Green	100	100	0
Blue	100	100	0

12.2 Relevant Profiles – part 2

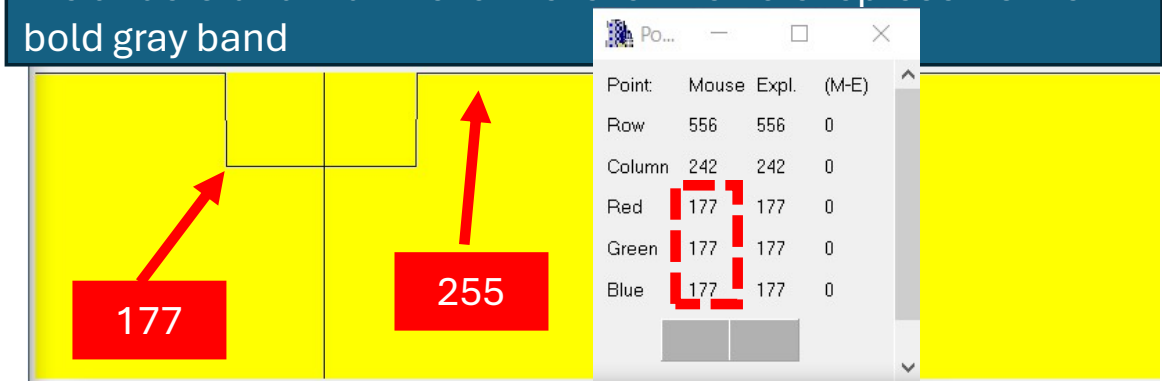
white_illusion5.bmp



In the columns view (AKA moving up and down on the same column) we can see a rectangular periodic function all over the column. That function is relevant for several columns – where we see the bold gray band!



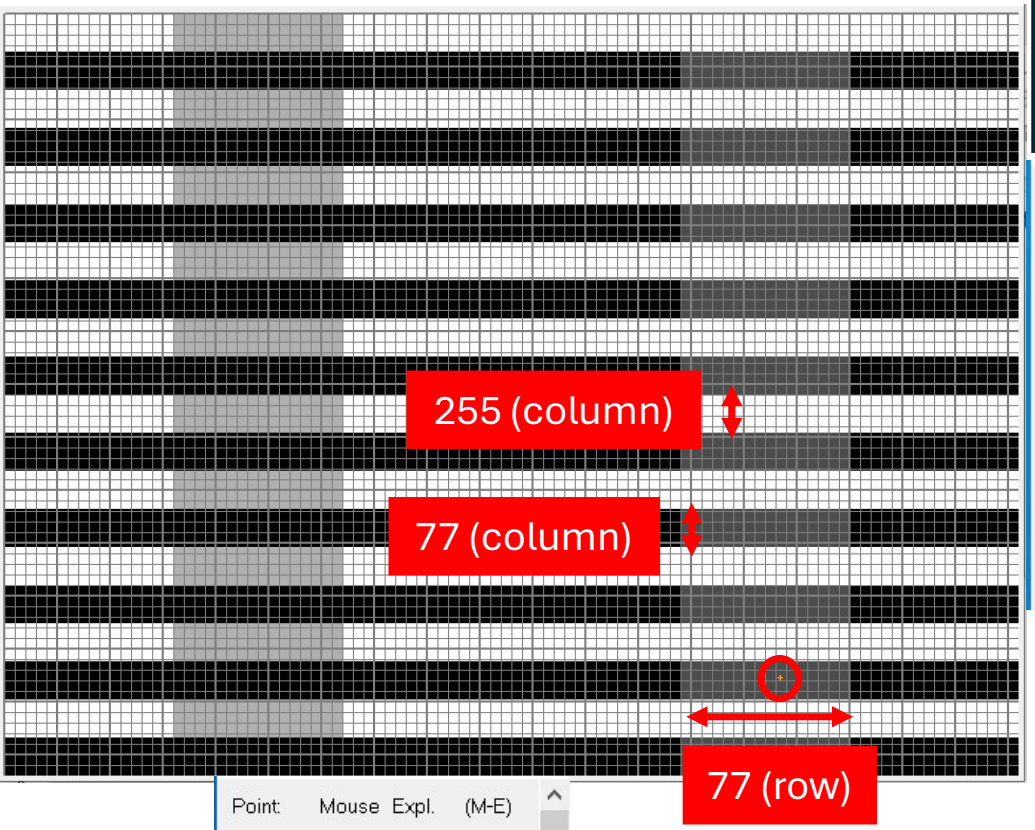
In the rows view (AKA moving right and left on the same row) we can see a hole function (like a potential hole).
We understand that the low level of the hole represents the bold gray band



white_illusion5.bmp – another example will be presented in the next slide

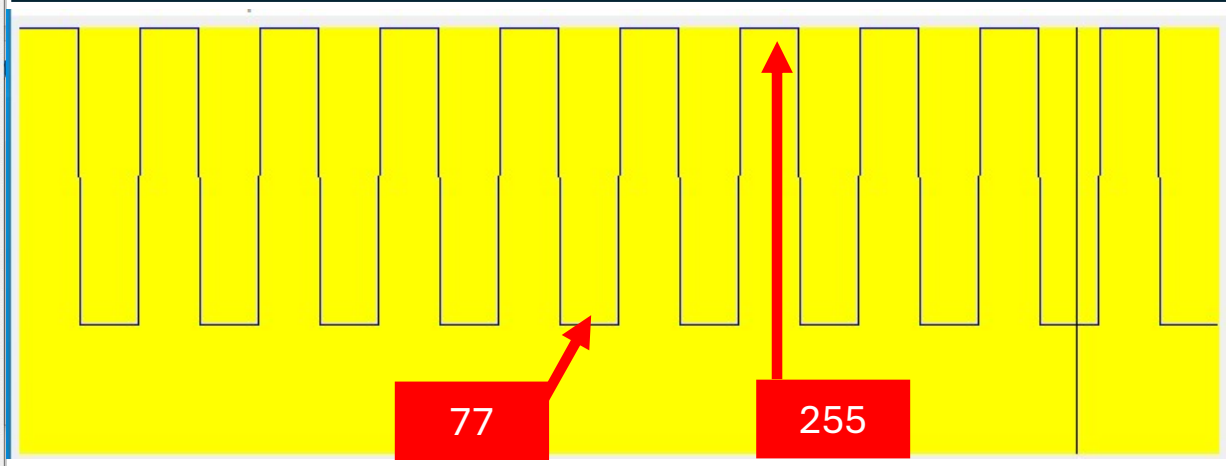
12.2 Relevant Profiles – part 3 (continue of part 2)

white_illusion5.bmp

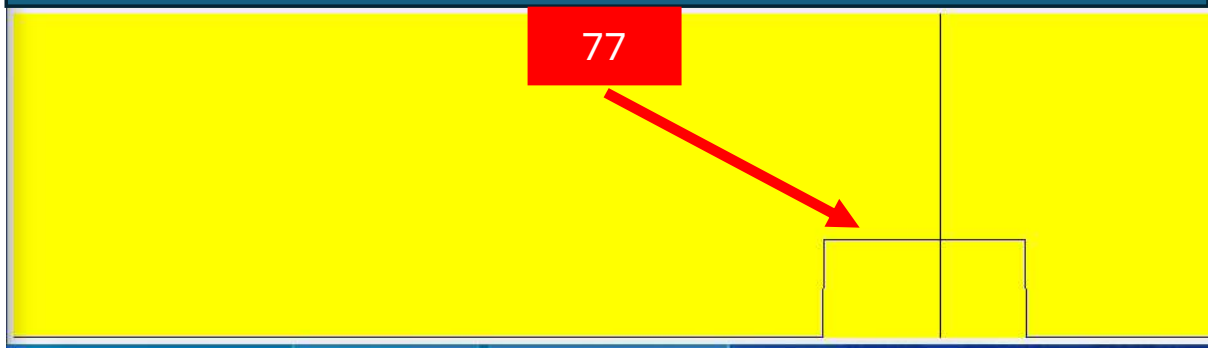


Point:	Mouse	Expl.	(M-E)
Row	98	98	0
Column	747	747	0
Red	77	77	0
Green	77	77	0
Blue	77	77	0

In the columns view (AKA moving up and down on the same column) we can see a rectangular periodic function all over the column. That function is relevant for several columns – where we see the bold gray band!

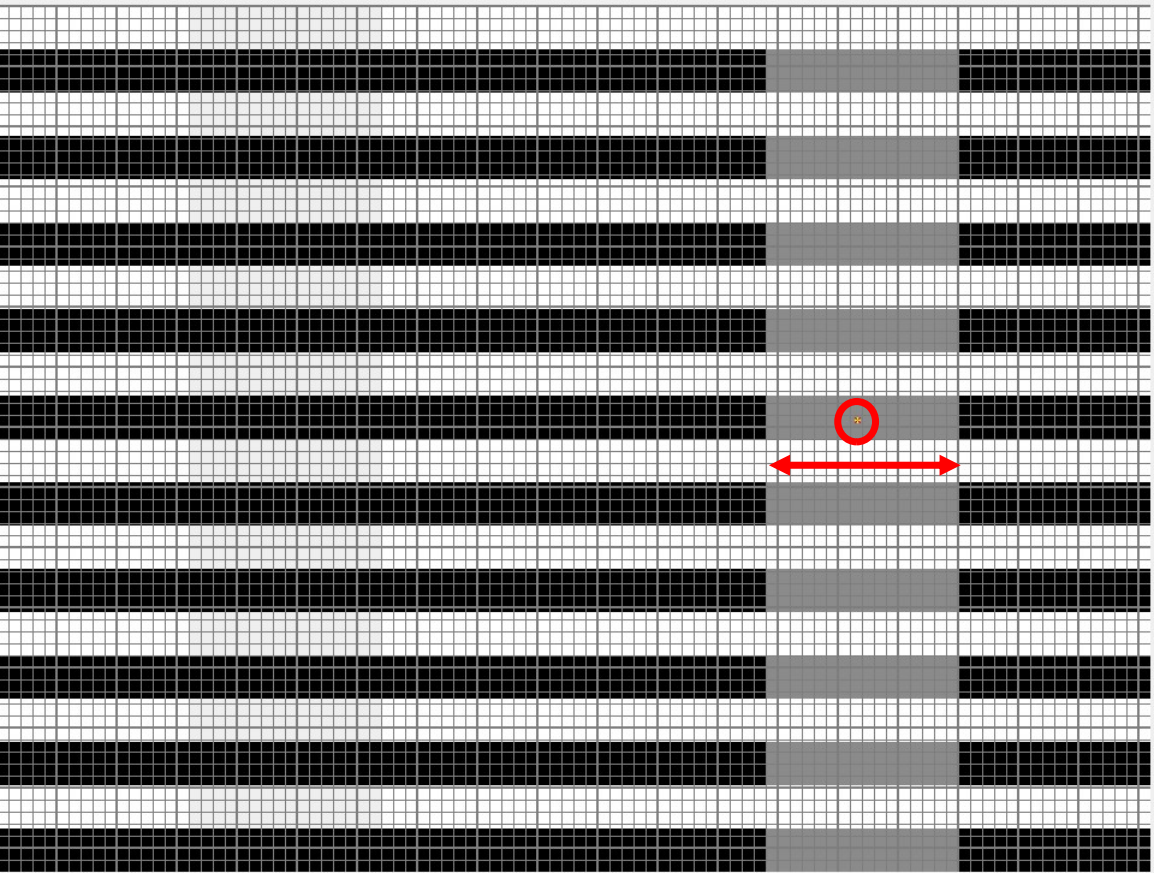


In the rows view (AKA moving right and left on the same row) we can see a step function.

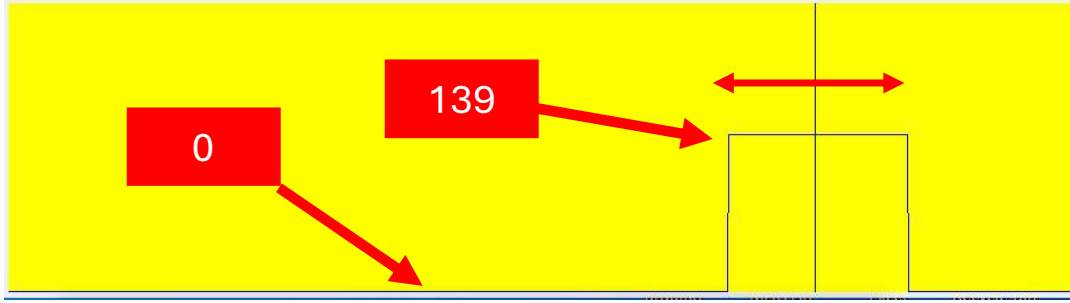


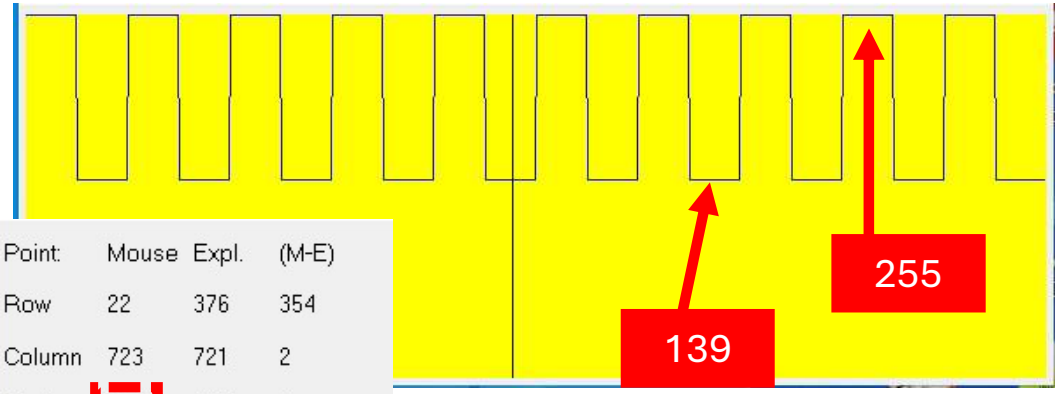
12.2 Relevant Profiles – part 4

white_illusion9.bmp



In the rows view (AKA moving right and left on the same row) we can see a step function.
We understand that the low level of the hole represents the non bold gray band



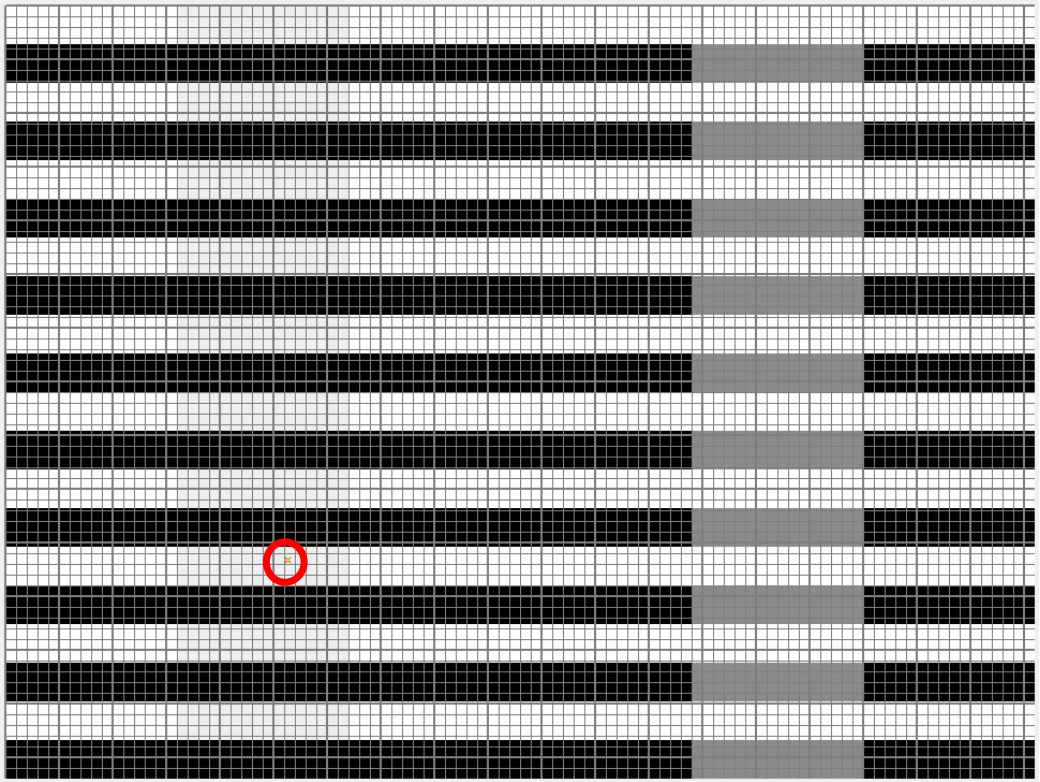


Point:	Mouse	Expl.	(M-E)
Row	22	376	354
Column	723	721	2
Red	139	139	0
Green	139	139	0
Blue	139	139	0

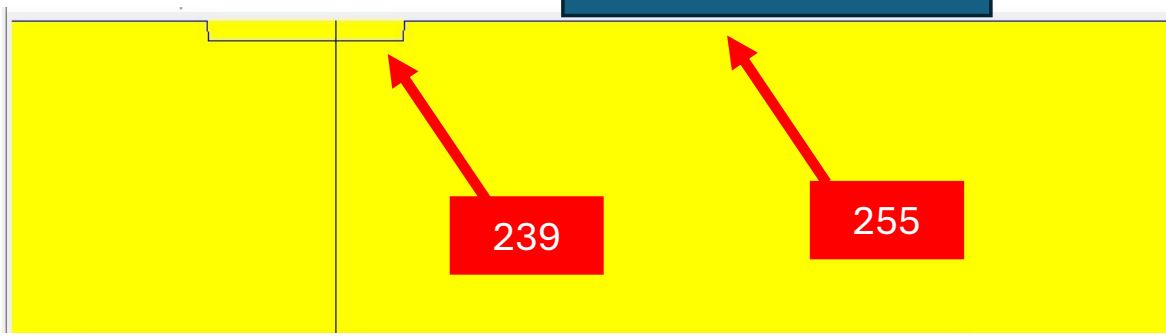
In the columns view (AKA moving up and down on the same column) we can see a rectangular periodic function all over the column. That function is relevant for several columns – where we see the bold gray band!

12.2 Relevant Profiles – part 5 (continue of part 4)

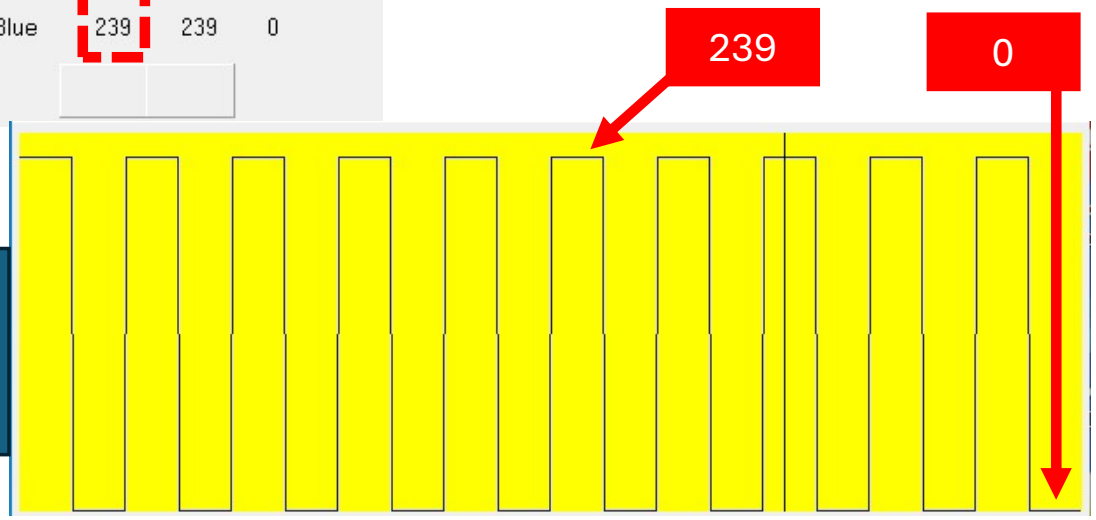
white_illusion9.bmp



Rows view



Point:	Mouse	Expl.	(M-E)
Row	200	200	0
Column	266	266	0
Red	239	239	0
Green	239	239	0
Blue	239	239	0



In the rows view (AKA moving right and left on the same row) we can see a very delicate hole function.
We understand that the very low level of the hole represents the non bold gray band (almost white = 239 ~255)

12.3 Code of the function "createWhitesIllusion" – part 1

```
89  void createWhitesIllusion(unsigned char img[][NUMBER_OF_COLUMNS],
90                            int numberOfBlackHorizontalStrips, unsigned char grayLevelOfGrayBars) {
91      InitializeImage(img, 255);
92      int stripHeight = NUMBER_OF_ROWS / numberOfBlackHorizontalStrips;
93      for (int i = 0; i < NUMBER_OF_ROWS; i += stripHeight) {
94          s2dPoint p1, p2;
95          int narrowStripWidth = NUMBER_OF_COLUMNS / 6;
96          if ((i / stripHeight) % 2 == 0) {
97              // Add black strip
98              p1.X = 0;
99              p1.Y = i;
100             p2.X = NUMBER_OF_COLUMNS;
101             p2.Y = i + stripHeight;
102             AddGrayRectangle(img, p1, p2, 0, 0);
103             // Second narrow strip
104             p1.X = 4 * narrowStripWidth;
105             p1.Y = i;
106             p2.X = 5 * narrowStripWidth;
107             p2.Y = i + stripHeight;
108             AddGrayRectangle(img, p1, p2, 100, grayLevelOfGrayBars);
109         }
110         else {
111             // Add narrow gray strips within the white strip
112
113             // First narrow strip
114             p1.X = narrowStripWidth;
115             p1.Y = i;
116             p2.X = 2 * narrowStripWidth;
117             p2.Y = i + stripHeight;
118             AddGrayRectangle(img, p1, p2, 100, grayLevelOfGrayBars);
119         }
120     }
121 }
```

The function createWhitesIllusion is designed to create an illusion pattern on a grayscale image by adding alternating black and gray strips to a white background.

First input parameter : **unsigned char img [] [NUMBER_OF_COLUMNS]**: A 2D array representing the image where the illusion will be created.

Second input parameter : **int numberOfBlackHorizontalStrips**: The number of black horizontal strips to be added to the image.

third input parameter : **unsigned char grayLevelOfGrayBars**: The gray level of the gray bars that will be added within the white strips.

12.3 Code of the function "createWhitesIllusion" – part 2

```
InitializeImage(img, 255);
```

The image is initialized to a white background (gray level 255).

```
int stripHeight = NUMBER_OF_ROWS / numberOfBlackHorizontalStrips;
```

The height of each strip is calculated by dividing the total number of rows by the number of black horizontal strips.

```
for (int i = 0; i < NUMBER_OF_ROWS; i += stripHeight) {
    s2dPoint p1, p2;
    int narrowStripWidth = NUMBER_OF_COLUMNS / 6;
    if ((i / stripHeight) % 2 == 0) {
        // Add black strip
        p1.X = 0;
        p1.Y = i;
        p2.X = NUMBER_OF_COLUMNS;
        p2.Y = i + stripHeight;
        AddGrayRectangle(img, p1, p2, 0, 0);
        // Second narrow strip
        p1.X = 4 * narrowStripWidth;
        p1.Y = i;
        p2.X = 5 * narrowStripWidth;
        p2.Y = i + stripHeight;
        AddGrayRectangle(img, p1, p2, 100, grayLevelOfGr
    }
```

The for loop iterates through the image rows in steps of stripHeight to add alternating strips.

For even-indexed strips (if ((i / stripHeight) % 2 == 0)), add a black strip and a narrow gray strip:

12.3 Code of the function "createWhitesIllusion" – part 3

```
else {
    // Add narrow gray strips within the white strip

    // First narrow strip
    p1.X = narrowStripWidth;
    p1.Y = i;
    p2.X = 2 * narrowStripWidth;
    p2.Y = i + stripHeight;
    AddGrayRectangle(img, p1, p2, 100, grayLevelOfGrayBars);
```

For odd-indexed strips (else) , add a narrow gray strip within the white strip

The function **AddGrayRectangle** is called to add rectangles with specified transparency and gray levels.
It first validates the coordinates using **checkValidation**, then applies the rectangle with blending techniques.

Final conclusion : the **createWhitesIllusion** function creates an illusion on a grayscale image by alternating black and gray strips on a white background. **Even** rows are filled with black strips, including a narrow gray strip, while **odd** rows only contain narrow gray strips. This pattern is generated using loops and helper functions to manipulate the pixel values of the image array.

12.4 Code of the main function – part 1

```
int main() {

    const int numIllusion = 10;
    unsigned char transparencies[numIllusion];
    unsigned char grayLevels[numIllusion];
    int numberOfBlackHorizontalStrips = 20;
    for (int i = 0; i < numIllusion; i++) {
        transparencies[i] = static_cast<unsigned char>((static_cast<float>(i) / numIllusion) * 255);
        grayLevels[i] = static_cast<unsigned char>((static_cast<float>(i) / numIllusion) * 255);

        createWhitesIllusion(img2
            , numberOfBlackHorizontalStrips, grayLevels[i]);
        char filename[30];
        sprintf(filename, "white_illusion%d.bmp", i);
        StoreGrayImageAsGrayBmpFile(img2, filename);

    }
    WaitForUserPressKey();


}
```

The main function in the provided code is responsible for setting up and creating multiple versions of a visual illusion pattern, saving each version as a BMP file.

Transparency and gray level values are calculated based on the iteration index i, resulting in a range of values from 0 to 255.

The **createWhitesIllusion** function is called to create the illusion pattern on the img2 array using the calculated gray level.

The main function initializes arrays for transparency and gray levels and then iterates to create multiple illusion images with varying gray levels

Each generated image is saved as a BMP file with a unique name.
Finally, the program waits for user input to ensure the user has a chance to review the output before the program finishes execution.

12.5 What did we learned ?

understand how grayscale images are represented and manipulated, using values from 0 (black) to 255 (white).

Learn how to blend different transparency levels and gray levels to create visual effects.

Apply geometric concepts to determine the positions and dimensions of shapes within an image.

Understand techniques for generating dynamic filenames to save multiple output files.