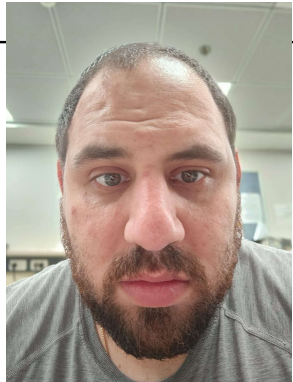




# Course: Image Processing 31651

## Assignment #13

### Synthetic Image Creation (Part 3)

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

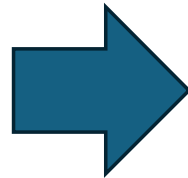
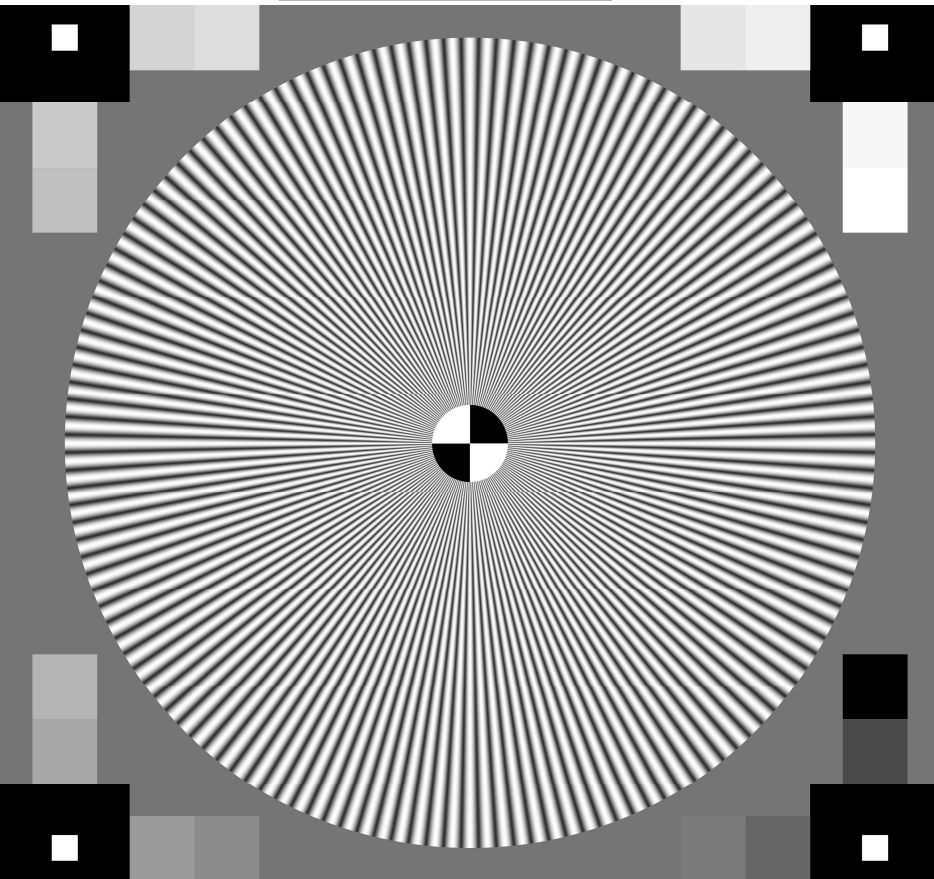
## 13.1 original TV Test image and our resulted image “grayImage13.bmp”

2

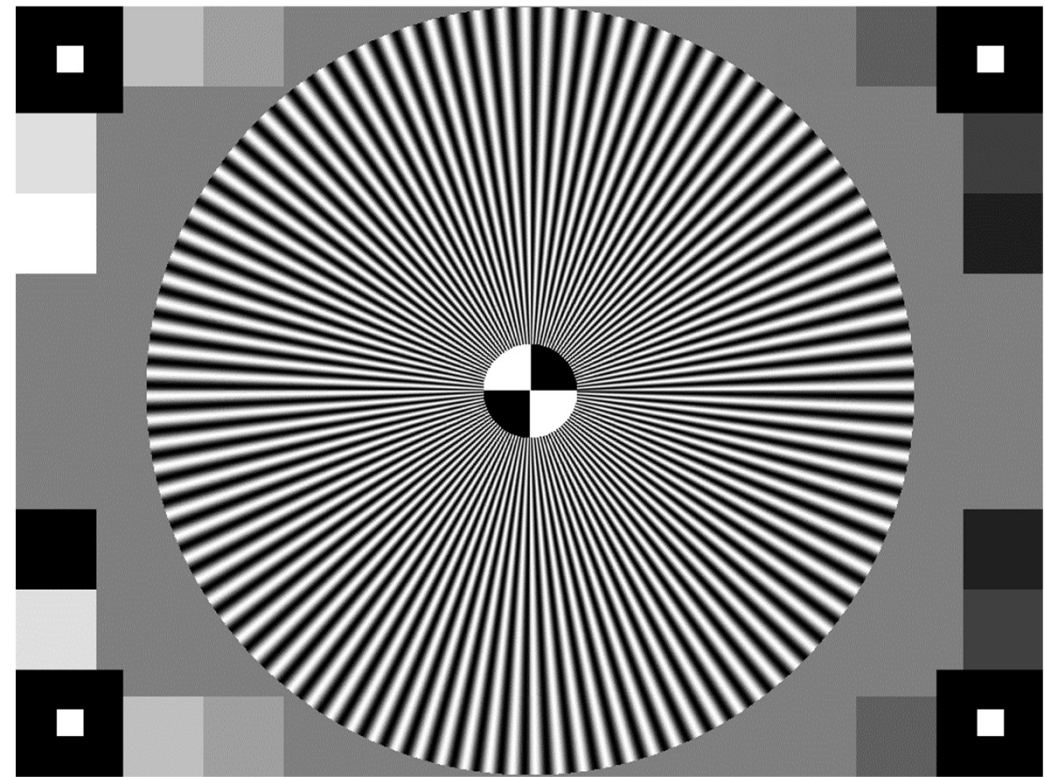
We can see that we have managed to create a TV image that is quite similar the original TV test image

**We know that the code that created the resulted image is not as the lecturer asked for. After consulting with the lecturer – he asked us to write that notation down in the presentation and in the mail!**

original TV Test  
image



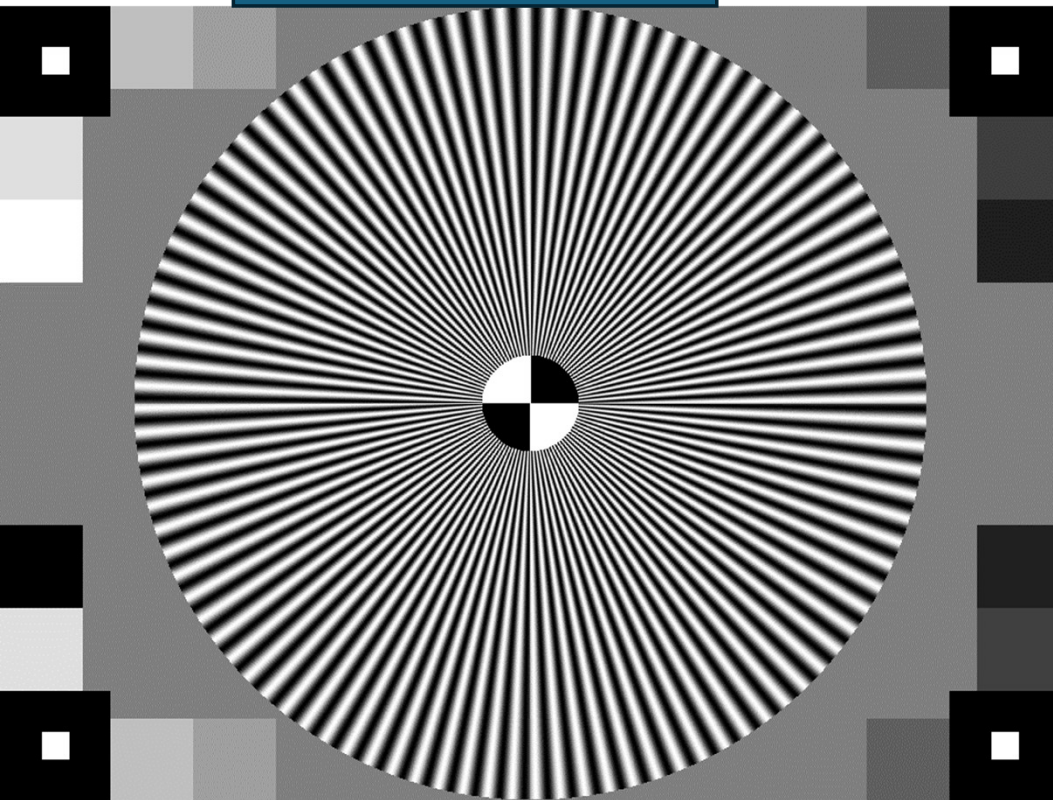
our resulted image  
“grayImage13.bmp”



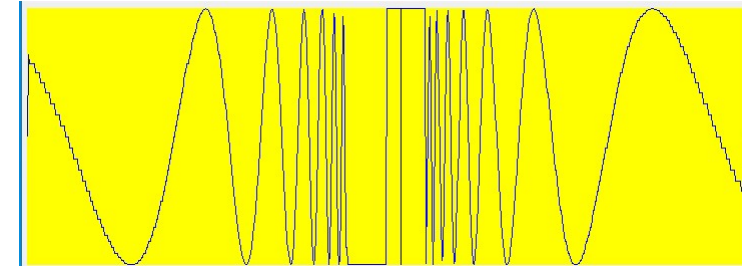
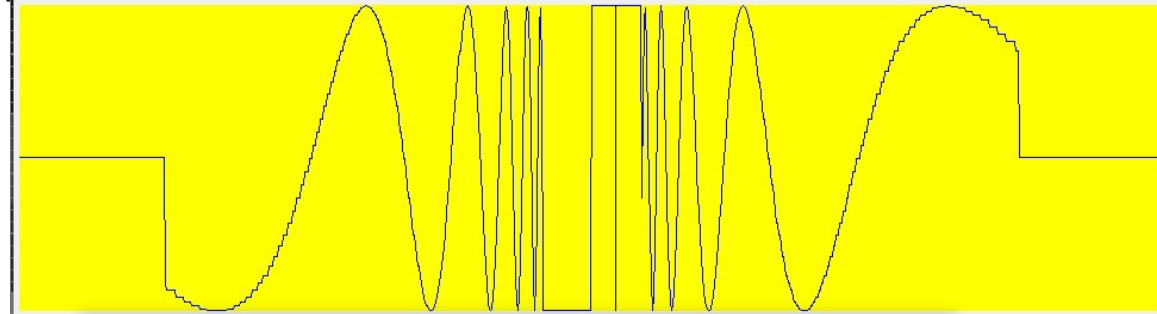
## 13.2 Relevant Profiles – **part 1**

(to prove that image “grayImage13.bmp” was created as required)

New-grayImage13.bmp



Row (left and right)



Column (up and down)

Point:	Mouse	Expl.	(M-E)
Row	360	360	0
Column	478	478	0
Red	0	0	0
Green	0	0	0
Blue	0	0	0

location

RGB values

Notation: I tried to analyze also the original given BMP image but the BMP viewer did not work (showed very wired results) for it also when the lecturer tried to use. So we present only the profile of the created image

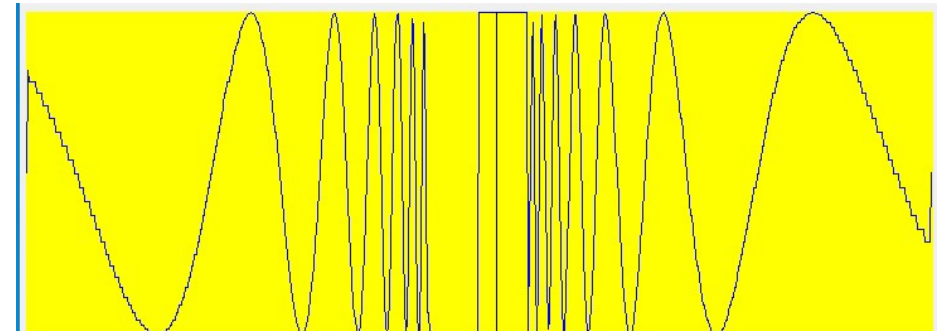


## 13.2 Relevant Profiles – part 2 – conclusions (to prove that image “grayImage13.bmp” was created as required)

The row and column profiles reveal symmetrical patterns, indicating that the original image has symmetrical features. **This is consistent with the radial pattern in the original image, where similar intensity variations occur at equidistant points from the center.**

The sharp transitions and periodic oscillations in the profiles suggest a central point with radially distributed patterns. In the original image, this is the center of the concentric circles and lines, where the intensity changes rapidly as you move away from the center.

The distinct steps or plateaus in the profiles indicate the presence of rectangular blocks in the image. These blocks are visible in the corners and edges of the original image



### 13.3 Code of the function “CreateTVtestImage” – part 1

```
45 void CreateTVtestImage(unsigned char img[][NUMBER_OF_COLUMNS])
46 {
47     InitializeImage(img, 128);
48
49     int centerX = NUMBER_OF_COLUMNS / 2;
50     int centerY = NUMBER_OF_ROWS / 2;
51     int radius = min(NUMBER_OF_COLUMNS, NUMBER_OF_ROWS) / 2;
52     for (double angle = 0; angle < 360; angle += 0.05) {
53         double radians = angle * M_PI / 180.0;
54         double sinValue = sin(radians * 100); // Sine function for smooth transition
55         int color = static_cast<int>((sinValue + 1) * 127.5); // Scale to [0, 255]
56
57         for (int r = 0; r < radius; r++) {
58             int x = centerX + static_cast<int>(r * cos(radians));
59             int y = centerY + static_cast<int>(r * sin(radians));
60
61             if (x >= 0 && x < NUMBER_OF_COLUMNS && y >= 0 && y < NUMBER_OF_ROWS) {
62                 img[y][x] = color;
63             }
64         }
65
66         for (int r = 0; r < radius / 8; r++) {
67             int x = centerX + static_cast<int>(r * cos(radians));
68             int y = centerY + static_cast<int>(r * sin(radians));
69
70             if (x >= 0 && x < NUMBER_OF_COLUMNS && y >= 0 && y < NUMBER_OF_ROWS) {
71                 if (angle < 90 || (angle > 180 && angle < 270))
72                     img[y][x] = 0;
73                 else img[y][x] = 255;
74             }
75         }
76     }
77 }
```

In the **CreateTVtestImage** function – there are 3 major components to be created:

- 1) Initialization
- 2) Central Circular Pattern
- 3) Grayscale Blocks

Each will be explained deeper after the presentation of the code

### 13.3 Code of the function “CreateTVtestImage” – part 2

```
77  // Add grayscale blocks at the corners and edges
78  int blockSize = 100;
79
80  // Corners
81  s2dPoint blackCorners[4] = {
82      {0, 0},
83      {NUMBER_OF_COLUMNS - blockSize, 0},
84      {0, NUMBER_OF_ROWS - blockSize},
85      {NUMBER_OF_COLUMNS - blockSize, NUMBER_OF_ROWS - blockSize}
86  };
87
88  for (auto& corner : blackCorners) {
89      s2dPoint oppositeCorner = { corner.X + blockSize, corner.Y + blockSize };
90      AddGrayRectangle(img, corner, oppositeCorner, 0, 0);
91  }
92
93  // White inner squares
94  int whiteInnerBlockSize = blockSize / 4;
95  s2dPoint whiteInnerCorners[4] = {
96      {blockSize / 2 - whiteInnerBlockSize / 2, blockSize / 2 - whiteInnerBlockSize / 2},
97      {NUMBER_OF_COLUMNS - blockSize / 2 - whiteInnerBlockSize / 2, blockSize / 2 - whiteInnerBlockSize / 2},
98      {blockSize / 2 - whiteInnerBlockSize / 2, NUMBER_OF_ROWS - blockSize / 2 - whiteInnerBlockSize / 2},
99      {NUMBER_OF_COLUMNS - blockSize / 2 - whiteInnerBlockSize / 2, NUMBER_OF_ROWS - blockSize / 2 - whiteInnerBlockSize / 2}
100  };
101
102  for (auto& corner : whiteInnerCorners) {
103      s2dPoint oppositeCorner = { corner.X + whiteInnerBlockSize, corner.Y + whiteInnerBlockSize };
104      AddGrayRectangle(img, corner, oppositeCorner, 0, 255);
105  }
106  int upperGrayS = blockSize * 3 / 4;
```



### 13.3 Code of the function “CreateTVtestImage” – part 3

```
107  // Middle of edges (top, bottom, left, right)
108  s2dPoint edgeMiddleBlocks[16] = {
109      /*bottom right side only*/
110      {NUMBER_OF_COLUMNS - upperGrayS, 0 + blockSize + upperGrayS},
111      {NUMBER_OF_COLUMNS - upperGrayS, 0 + blockSize},
112      {NUMBER_OF_COLUMNS - blockSize - upperGrayS, 0},
113      {NUMBER_OF_COLUMNS - blockSize - 2 * upperGrayS, 0},
114      /*bottom left side only*/
115      {0 + blockSize + upperGrayS, 0},
116      {0 + blockSize, 0},
117      {0, blockSize},
118      {0, blockSize + upperGrayS},
119      /*upper left side only*/
120      {0, NUMBER_OF_ROWS - (blockSize + 2 * upperGrayS)},
121      {0, NUMBER_OF_ROWS - (blockSize + upperGrayS)},
122      {blockSize, NUMBER_OF_ROWS - upperGrayS},
123      {blockSize + upperGrayS, NUMBER_OF_ROWS - upperGrayS},
124      /*upper right side only*/
125      {NUMBER_OF_COLUMNS - blockSize - 2 * upperGrayS, NUMBER_OF_ROWS - upperGrayS},
126      {NUMBER_OF_COLUMNS - blockSize - upperGrayS, NUMBER_OF_ROWS - upperGrayS},
127      {NUMBER_OF_COLUMNS - upperGrayS, NUMBER_OF_ROWS - blockSize - upperGrayS},
128      {NUMBER_OF_COLUMNS - upperGrayS, NUMBER_OF_ROWS - (blockSize + 2 * upperGrayS)}
129  };
130
131  };
```

### 13.3 Code of the function “CreateTVtestImage” – part 4

```
134 int gray_level[16] = { 0 };
135 for (int i = 0; i < 8; i++) {
136     gray_level[i] = (i + 1) * 32; // 8 levels of black (from 32 to 256)
137 }
138 for (int i = 8; i < 16; i++) {
139     gray_level[i] = 255 - (i - 8) * 32; // 8 levels of white (from 255 down to 32)
140 }
141
142 int i = 0;
143 for (auto& side : edgeMiddleBlocks) {
144     s2dPoint oppositeSide = { side.X + upperGrayS, side.Y + upperGrayS };
145     AddGrayRectangle(img, side, oppositeSide, 0, gray_level[i]);
146     i++;
147 }
148
149
150 }
```



### 13.3 Code of the function “CreateTVtestImage” – **part 5**

#### Initialization:

- 1 The function starts by calling **InitializeImage(img, 128)** to set all pixels to a medium gray value (128).
- 2 The CreateTVtestImage function initializes an image with a gray background.

#### Central Circular Pattern:

- 1 The center of the image is calculated, and a circular pattern is drawn using a sine function to create a smooth transition of colors.
- 2 A loop runs through angles from 0 to 360 degrees in small increments (0.05 degrees).
- 3 For each angle, the sine function is used to determine the color intensity.
- 4 Two nested loops draw circles: one for the full radius and one for a smaller radius (1/8th of the full radius) to create alternating black and white segments.

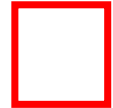
### 13.3 Code of the function “CreateTVtestImage” – part 6

#### Grayscale Blocks:

① Four black rectangles are added at the corners of the image.



Will be marked as:



② Four white inner squares are added near the corners.

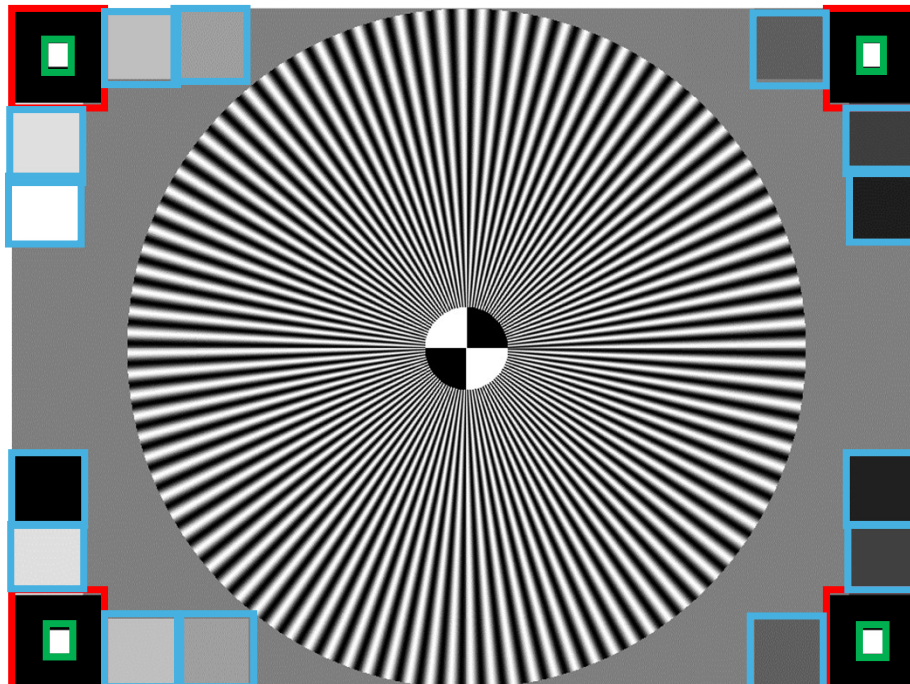


Will be marked as: 

③ 16 gray rectangles are added at the edges of the image, with varying gray levels.



Will be marked as:



## 13.4 Code of the main function

```
40  ✓int main() {  
41      CreateTVtestImage(img3);  
42      StoreGrayImageAsGrayBmpFile(img3, "grayImage13.bmp");  
43      return 0;  
44  }
```



## 13.5 What did we learned?

1. The code shows how to create complex patterns, such as a radial pattern with sinusoidal variations in pixel intensity, and how to draw rectangles with different gray levels. This involves understanding geometric transformations and basic trigonometry.
2. The code demonstrates how to represent and manipulate image data using a 2D array. Each element of the array corresponds to a pixel value in the grayscale image.
3. By working through this code, one can gain a deeper understanding of image processing techniques, geometric transformations, and file handling in C++.