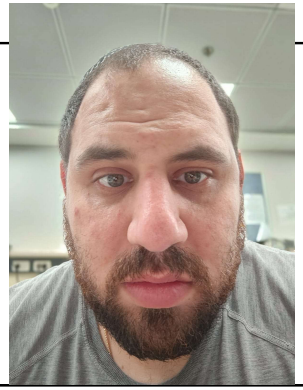
Course: Image Processing 31651
Assignment #14
Synthetic Image Creation (Part 4)

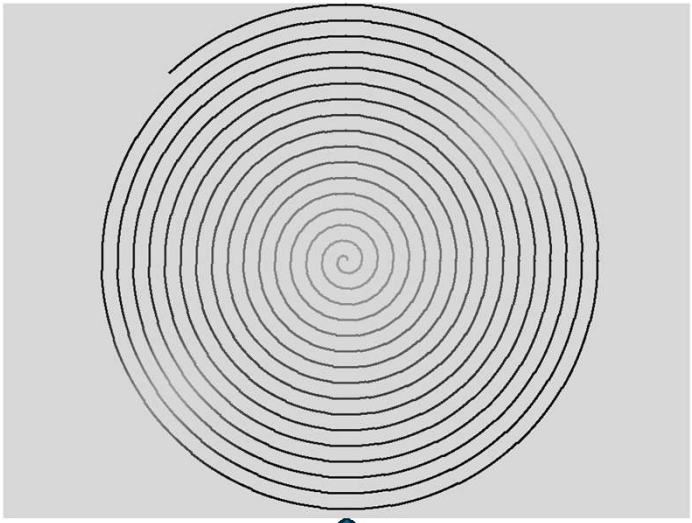
	ID (4 last digits)	Shorten Name	Photo of the student
Student #1	**1950**	shienfeld	
Student #2	**2210**	pony	
Student #3	**7939**	akimov	

Assignment #14: Create spiral blended with Gaussian

In the file ImProcInPlainC.h defined numerical values of:
NUMBER_OF_ROWS and NUMBER_OF COLUMNS

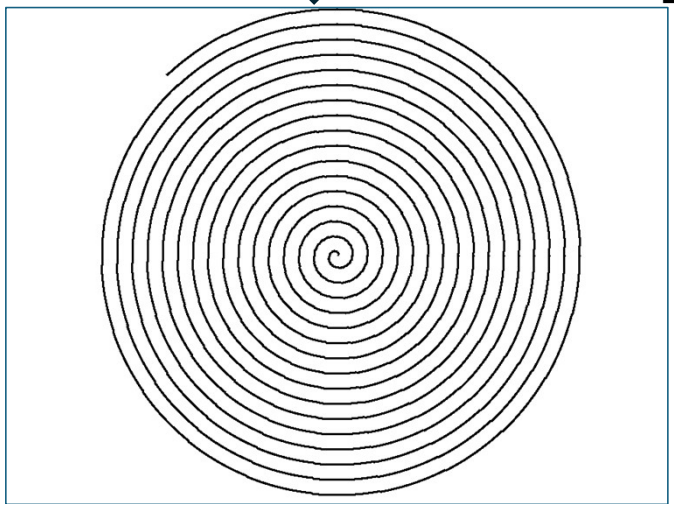
#	Image Description
1	Write function void DrawSpiral(unsigned char img[][NUMBER_OF COLUMNS]) Width of the spiral line is (3) pixels 
2	Write function void DrawGaussian(unsigned char img[][NUMBER_OF COLUMNS], int centerX, int sigmaX, int centerY, int sigmaY)
3	Blend Spiral with three Gaussians with different values of sigmaX and sigmaY. Select blending factors so that all Gaussians will be clearly seen Store resulted image as gray BMP file "grayImage14.bmp"
	For report use the following template. **Done: 14.1 14.2 14.3 14.4 14.5 14.6** **REMOVE ALL RED when the items are ready** **Important hint: Fill report pages while working.** **Do not complain "not enough time" in case you did not follow THIS rule.**

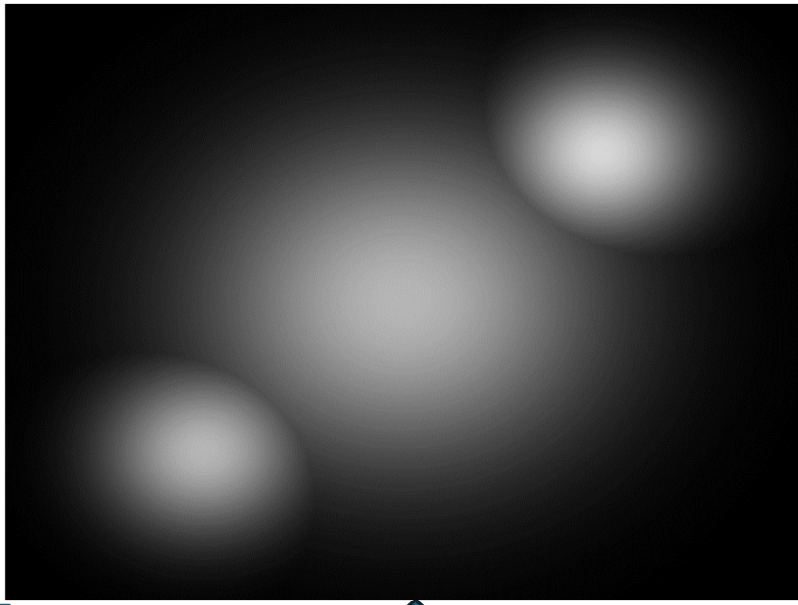
14.1 - our “image of spiral”, “images of Gaussian” and resulted blended image “grayImage14.bmp”



This image features a spiral pattern with gradually increasing radius, starting from the center and moving outward. The background contains the 3 gaussians patterns. The combined background is gray as a combination of the blended images. Pay attention (shown and approved by the lecturer) that the gaussians are not so easy to be seen but will be proved using profiles

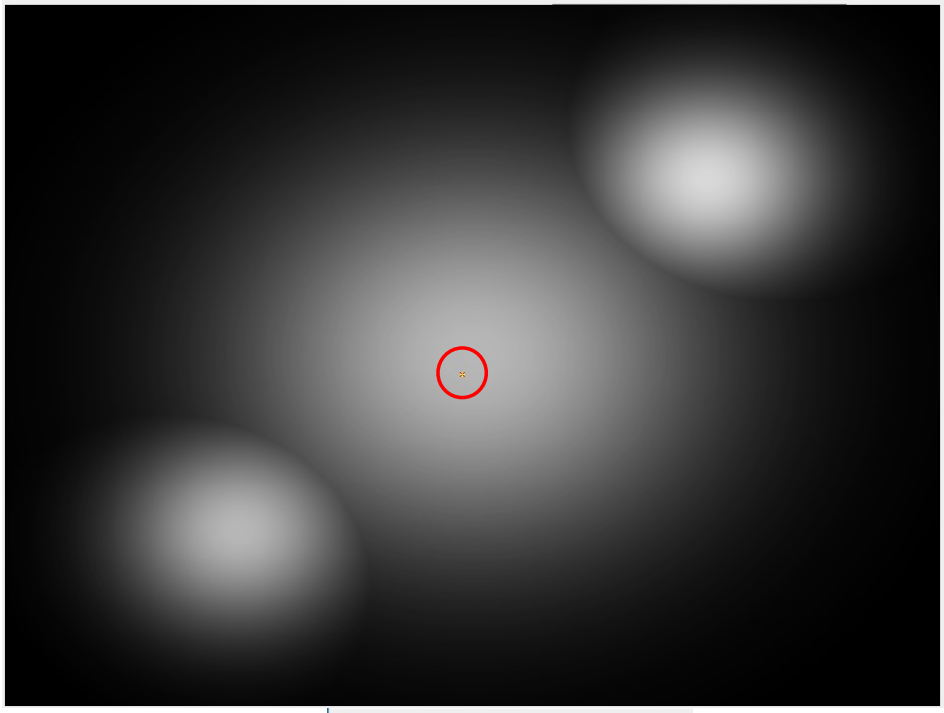
This image contains a spiral pattern. The background is uniformly white, making the black spiral lines more distinct and clearer.

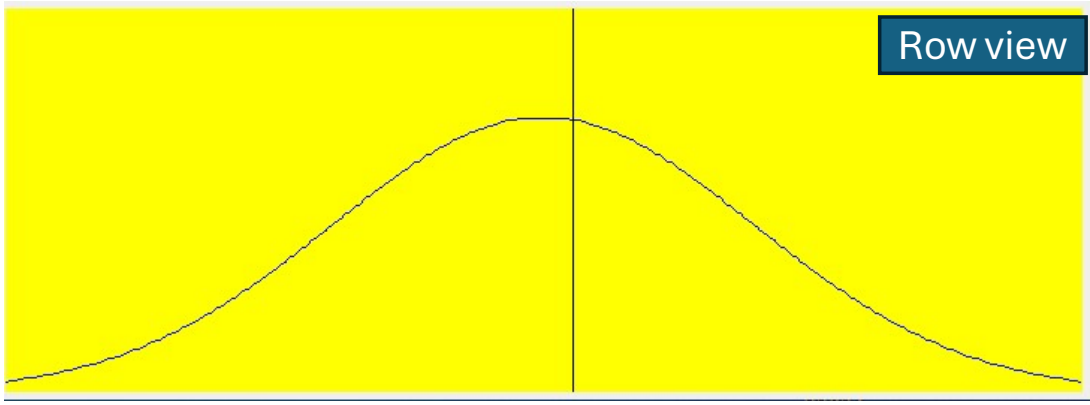




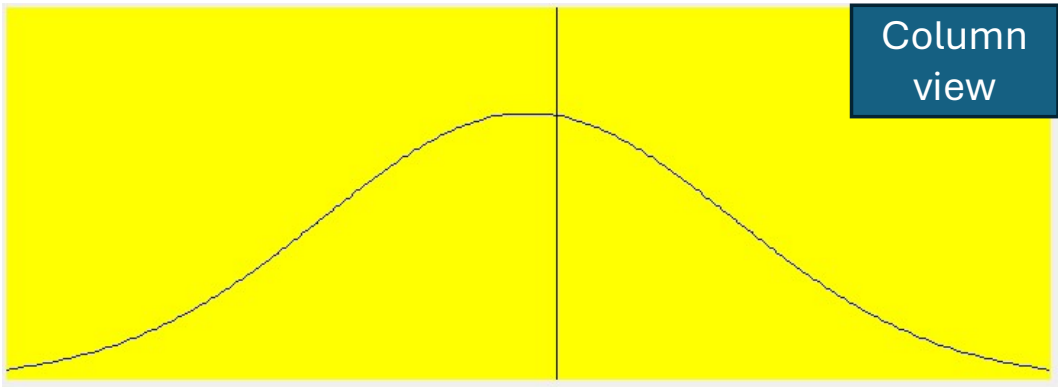
This image depicts 3 Gaussian patterns. In each gaussian pattern-the intensity of the pixels decreases smoothly from the center towards the edges, forming a bright spot in the middle that gradually fades to black.

14.2 - The Relevant Profiles (of all relevant images) (to prove that image "grayImage14.bmp" was created as required) - part 1/7

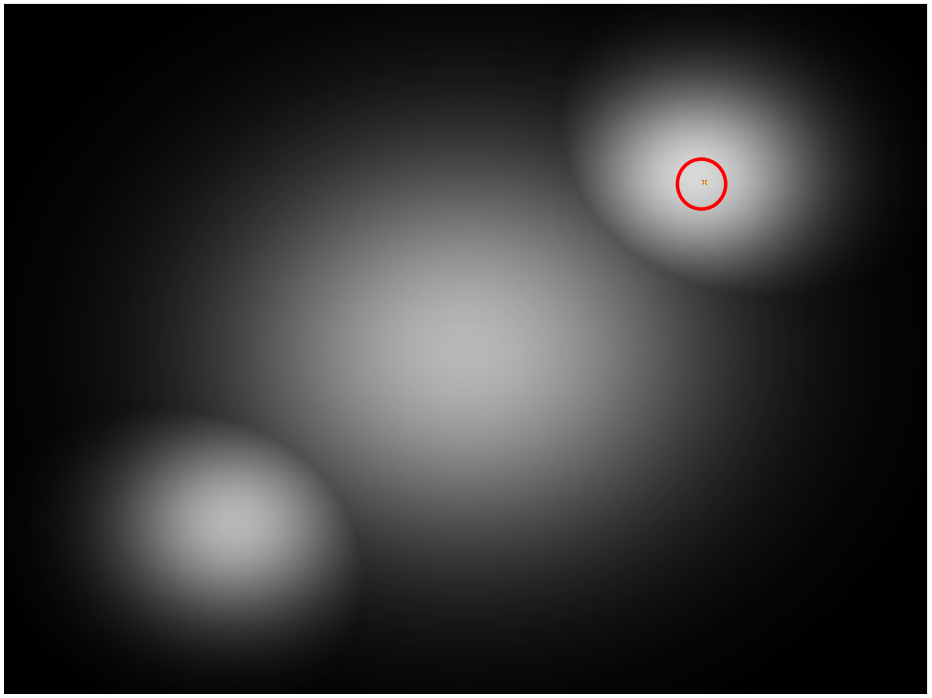




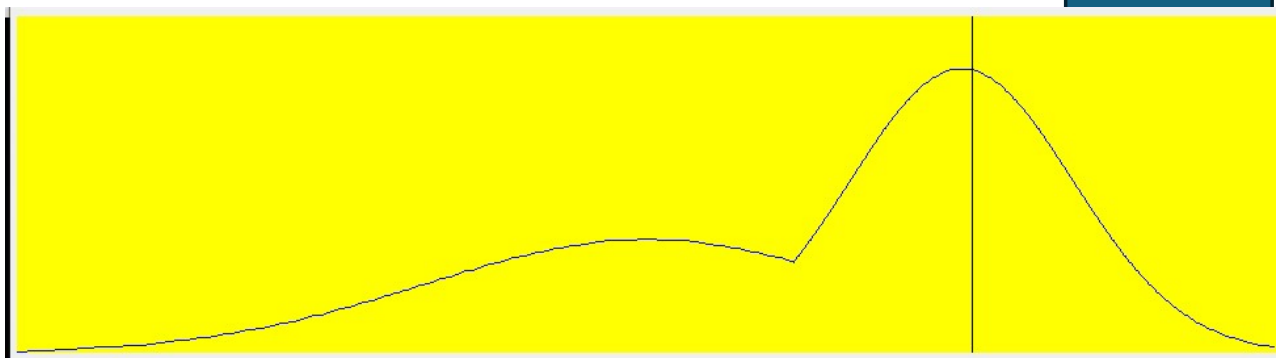
The point that we selected first to present is the "central point" of the central gaussian. The row and column views show the same profile which is a clear and perfect gaussian. The presented profiles approve the creation of the 1/3 gaussian as required.



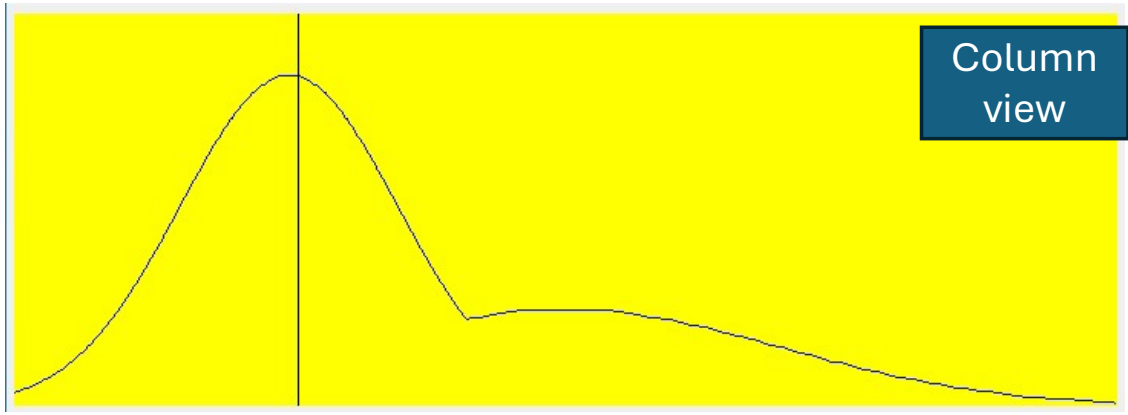
14.2 - The Relevant Profiles (of all relevant images) (to prove that image "grayImage14.bmp" was created as required) - part 2/7



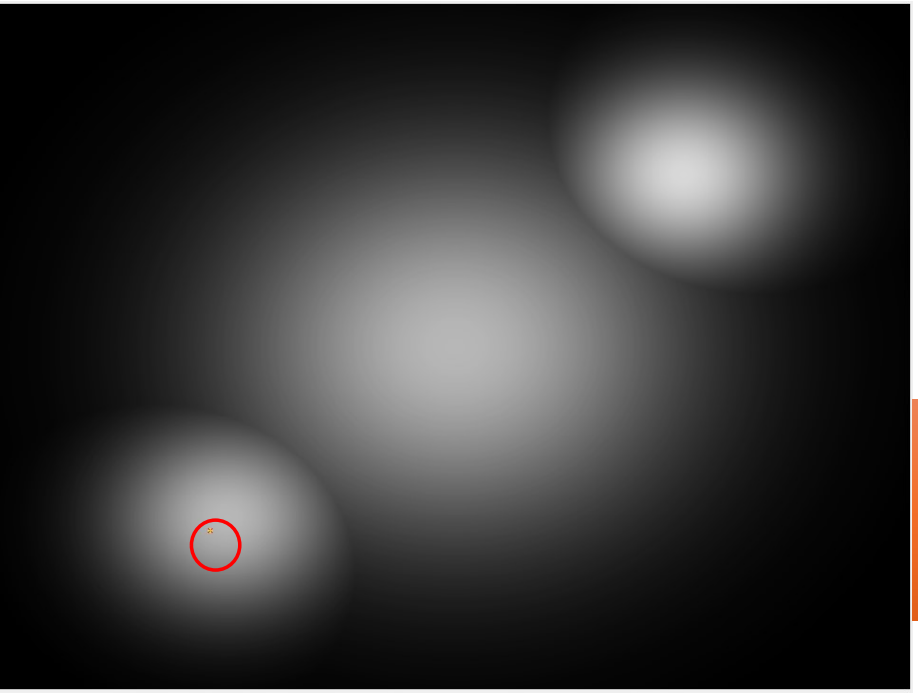
Row view

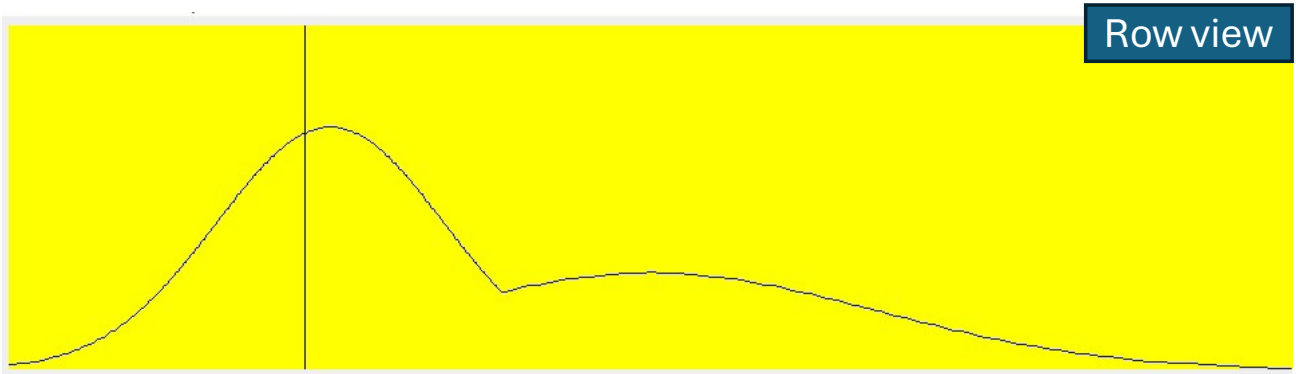


The point that we selected second to present is the "central point" of the upper right gaussian. The row and column views show a disrupted gaussian (pay attention to the reflected structure). The distruption in the gaussian pattern is due to the congruence with the central gaussian.

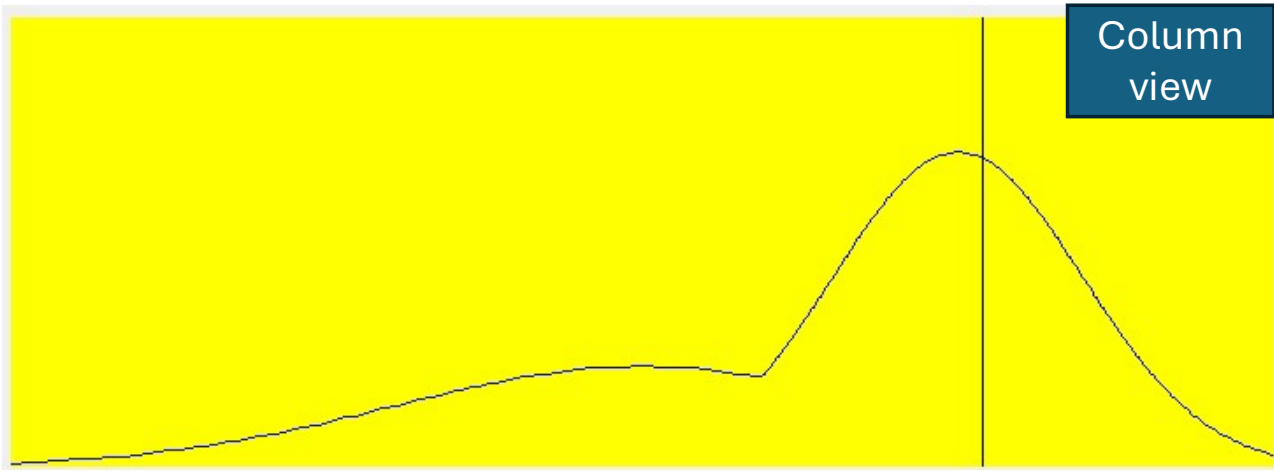


14.2 - The Relevant Profiles (of all relevant images) (to prove that image "grayImage14.bmp" was created as required) - part 3/7

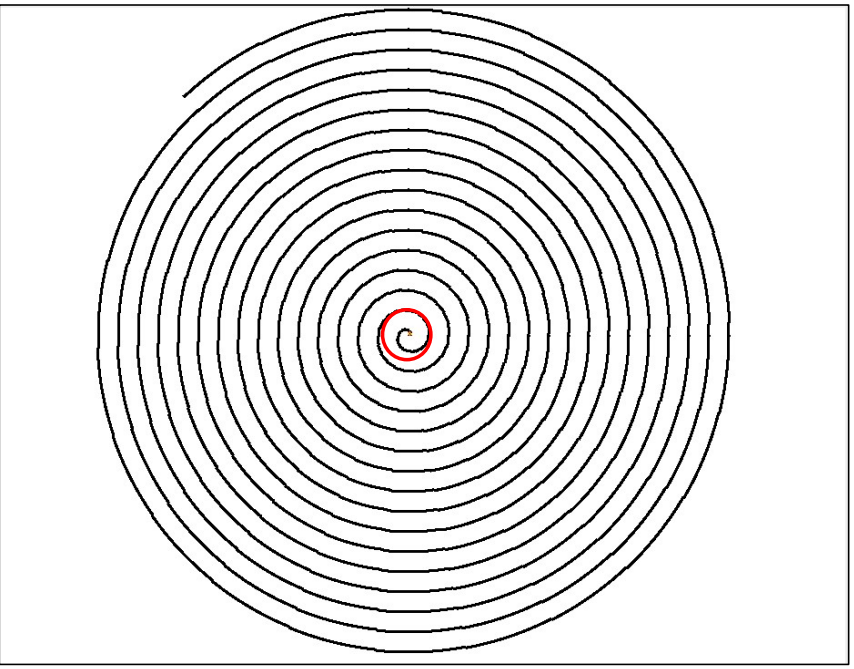


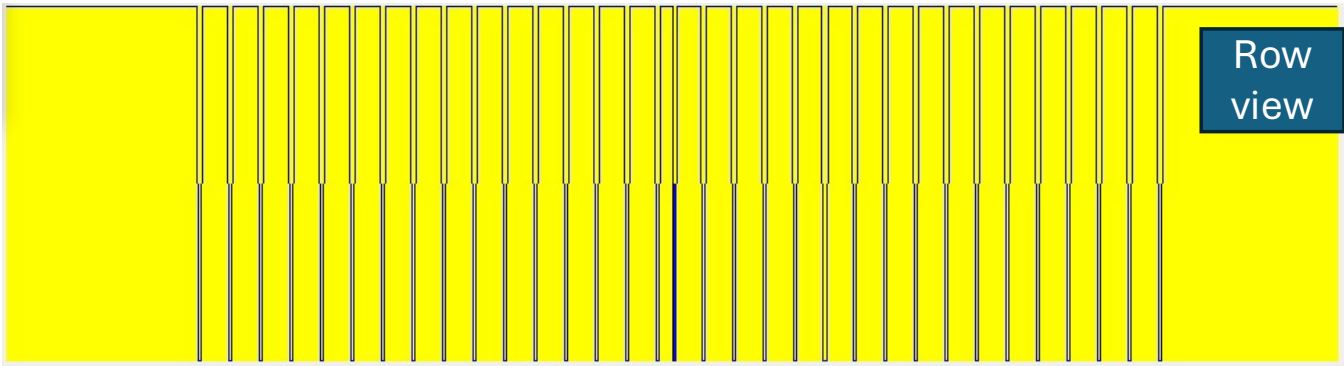


The point that we selected third to present is the "central point" of the lower left gaussian. The row and column views show a disrupted gaussian (pay attention to the reflected structure). The distruption in the gaussian pattern is due to the congruence with the central gaussian.

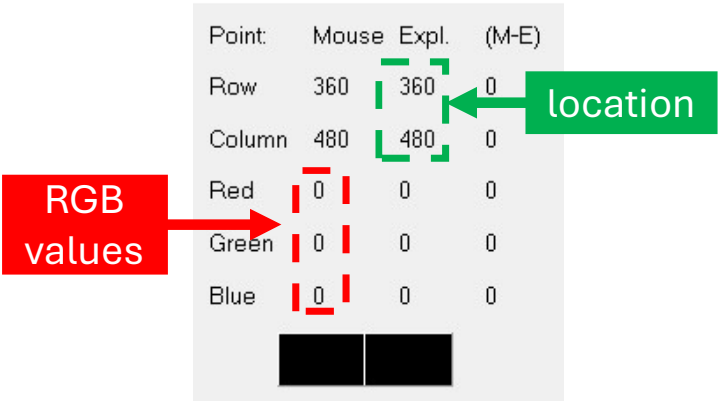


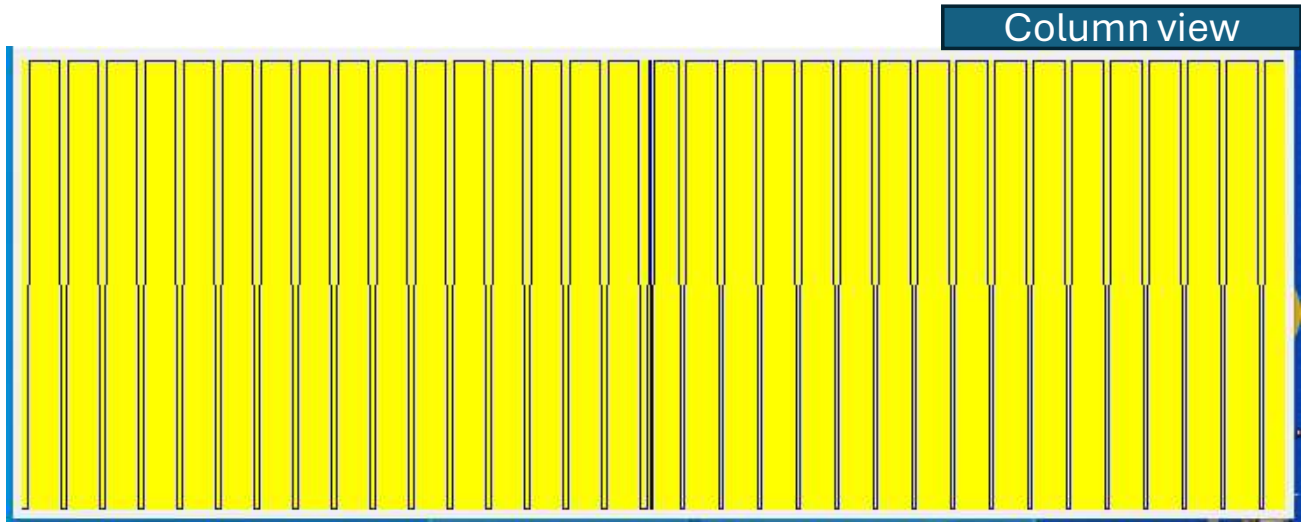
14.2 - The Relevant Profiles (of all relevant images) (to prove that image "grayImage14.bmp" was created as required) - part 4/7



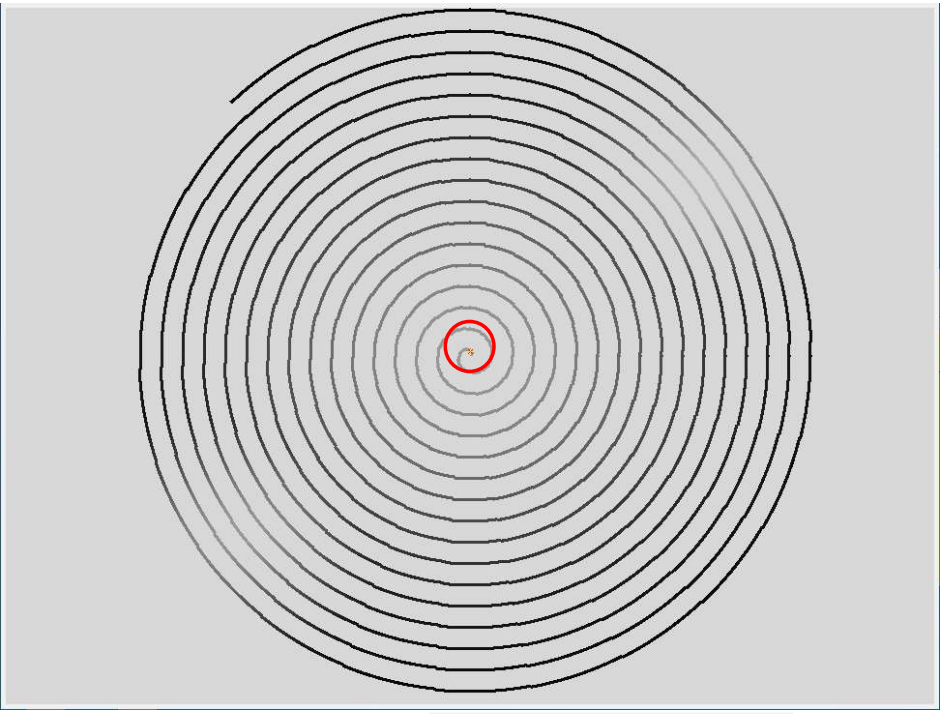


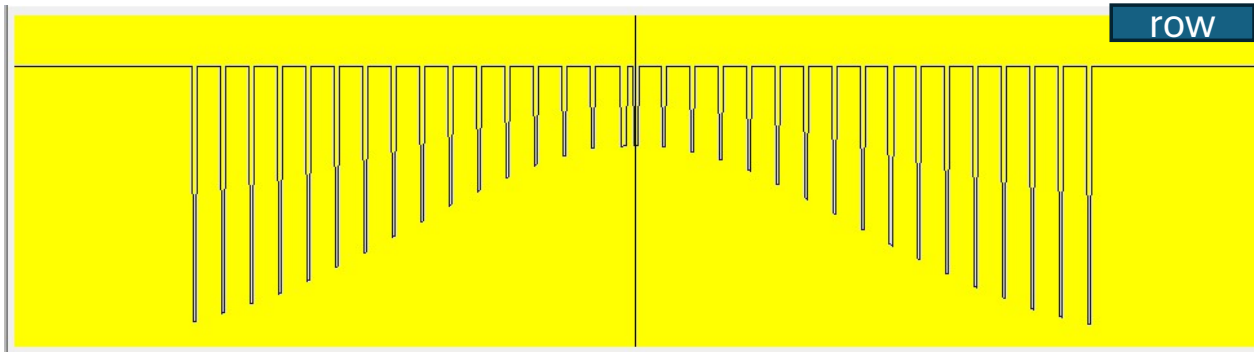
Again – the point we selected is the central point of the object – in this case it is the spiral. The profiles show a rectangular periodic function. Pay attention to the white space in the row view (level of 255) as the main difference between the row and the column profilers.



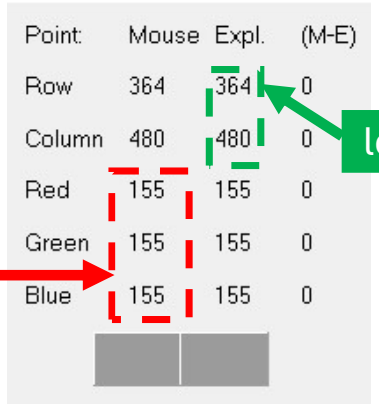


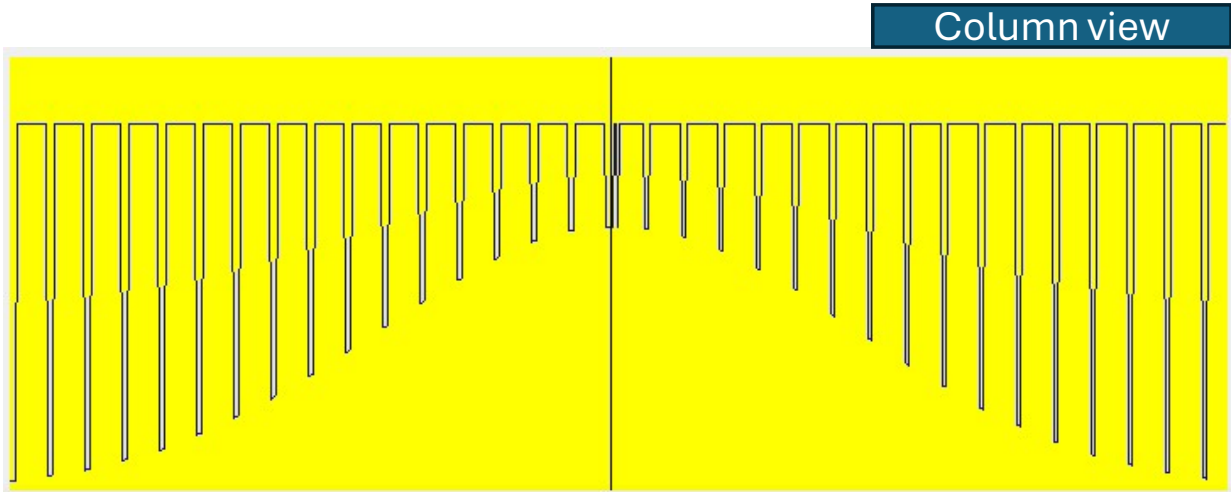
14.2 - The Relevant Profiles (of all relevant images) (to prove that image "grayImage14.bmp" was created as required) - part 5/7



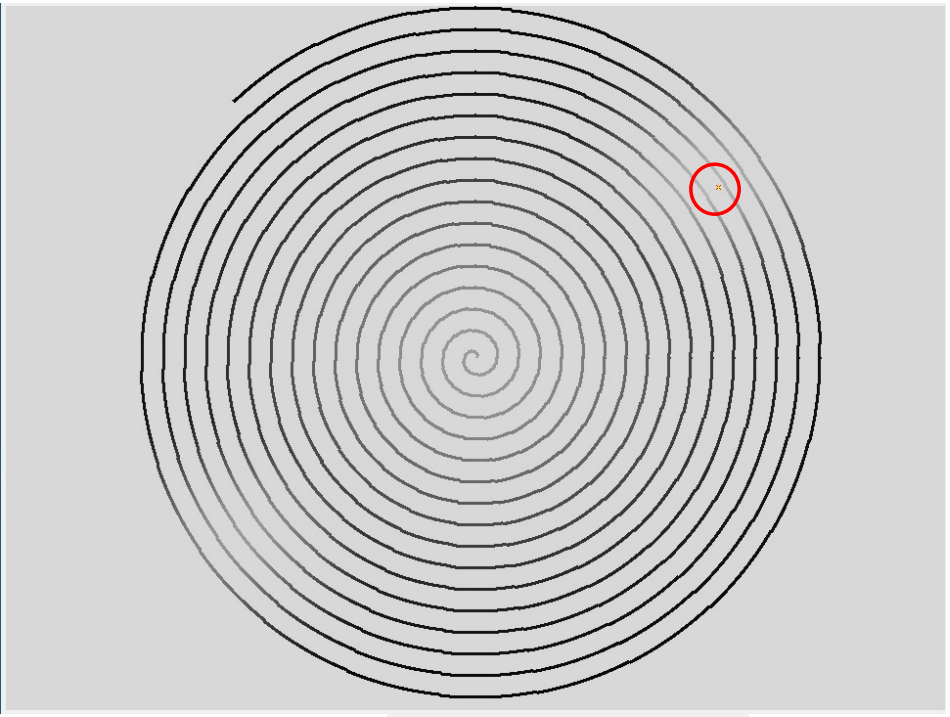


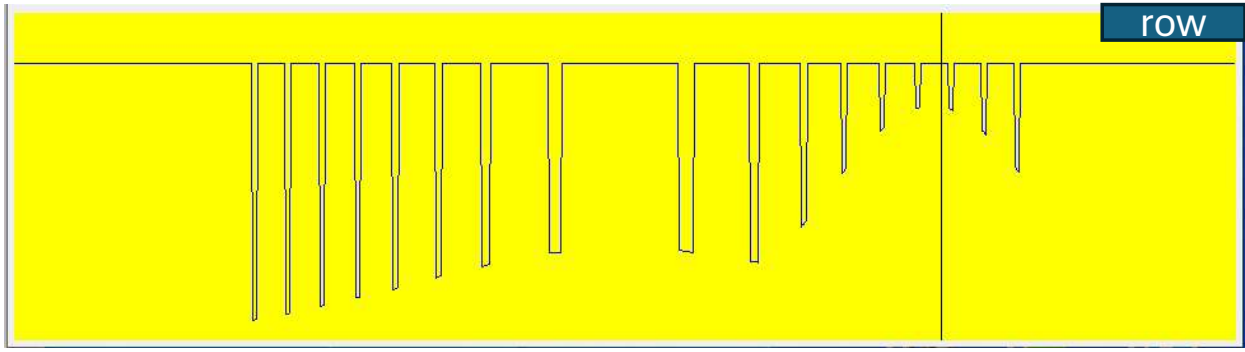
To consistent with part 1/7 of this section – we can see the same point but now the rectengualar pattern that "rides" on the gaussian pattern (like a signal and an envelop). The row and the column views present almost the same pattern.



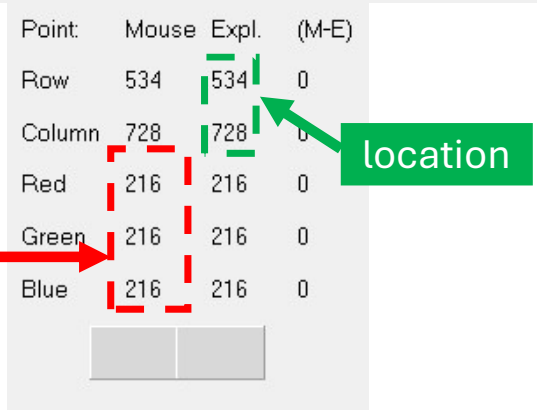


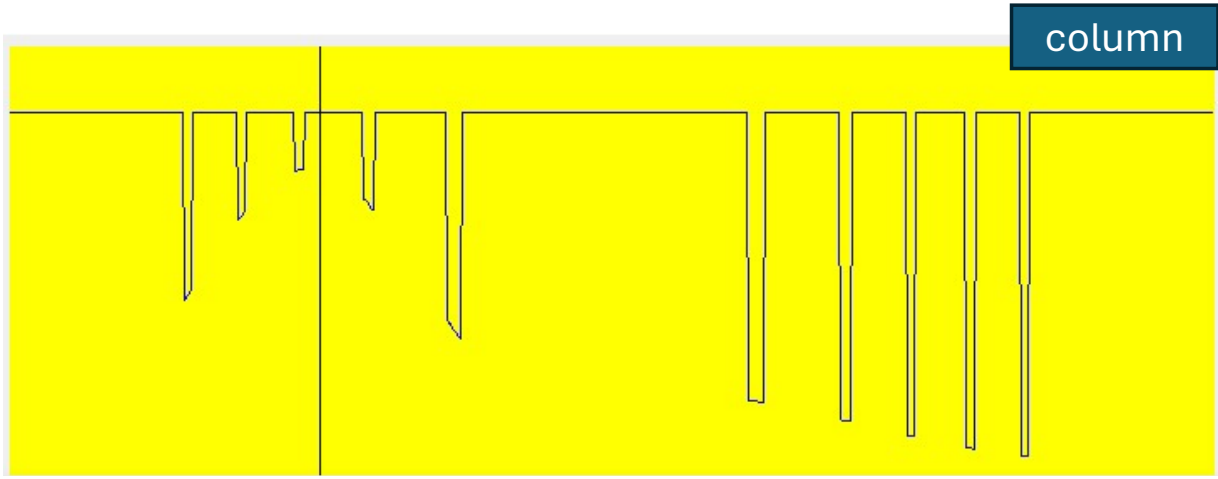
14.2 - The Relevant Profiles (of all relevant images) (to prove that image "grayImage14.bmp" was created as required) - part 6/7



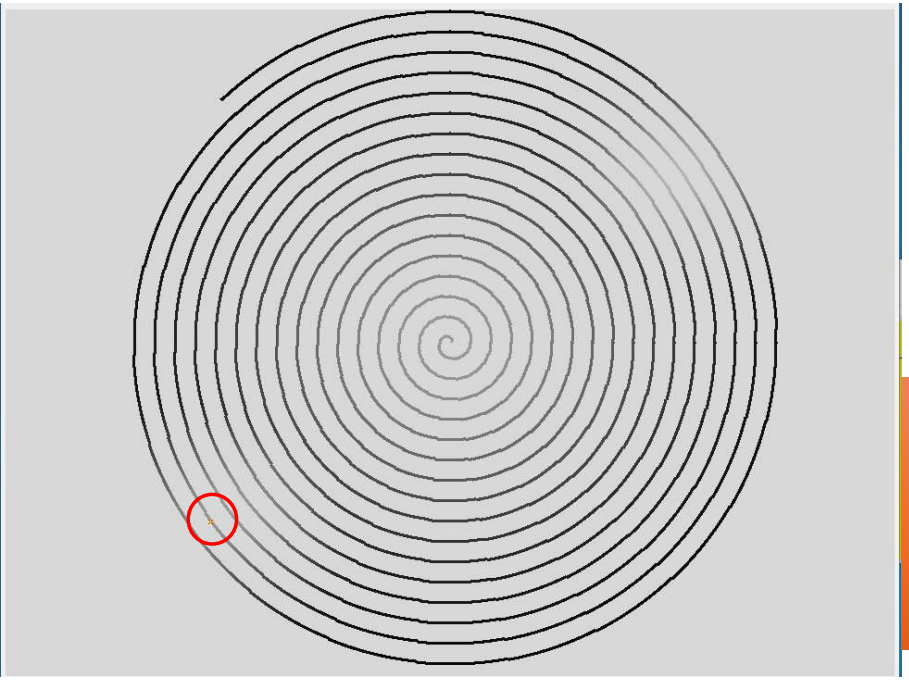


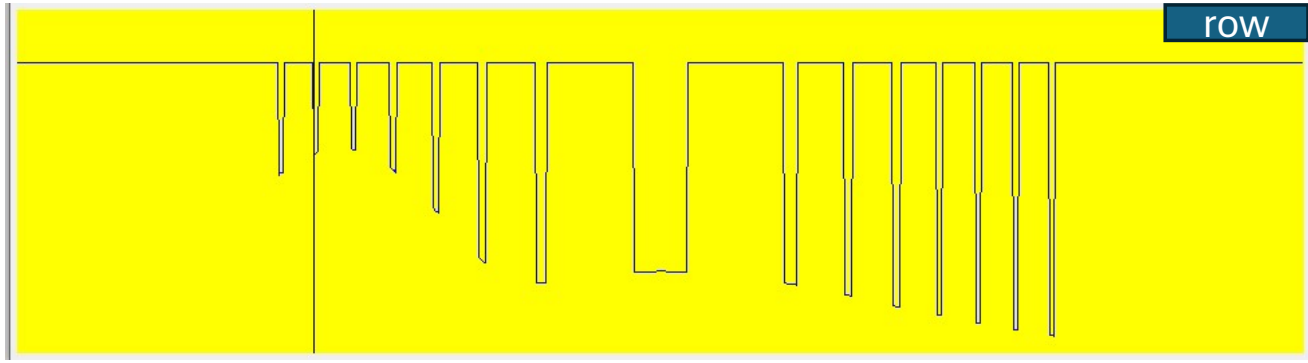
To consistent with part 2/7 of this section – we can see the same point but now the rectangular pattern is disrupted due to the gaussian and the congruence with the central gaussian.
If the gaussians were not congruence than we would have get the same perfect result as in part 5/7.



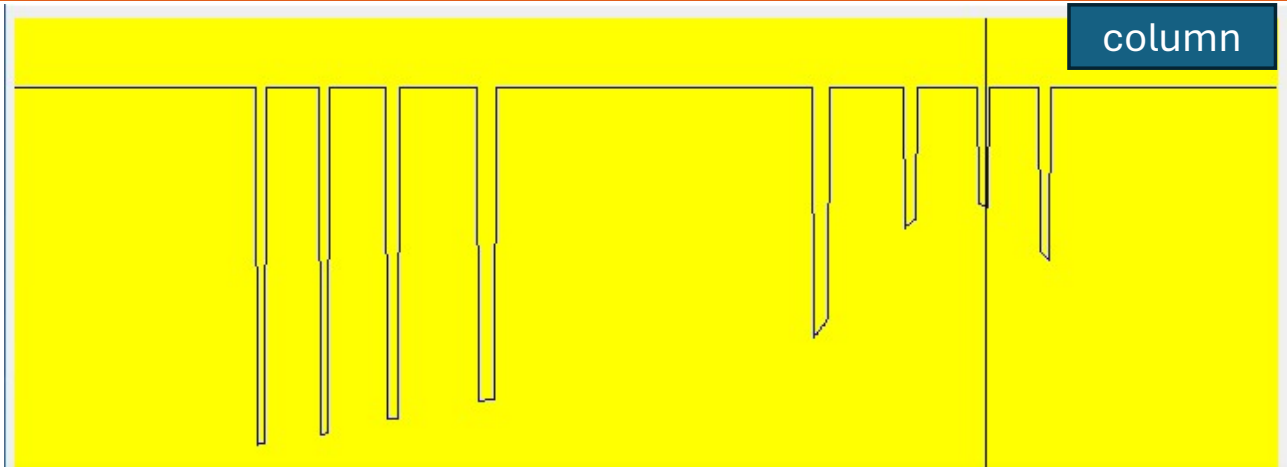


14.2 - The Relevant Profiles (of all relevant images) (to prove that image "grayImage14.bmp" was created as required) - part 7/7





To consistent with part 3/7 of this section – we can see the same point but now the rectangular pattern is disrupted due to the gaussian and the congruence with the central gaussian.
If the gaussians were not congruence than we would have get the same perfect result as in part 5/7. the result is a mirror of the result in part 6/7.



14.3 - Code of the function “DrawSpiral”


```
98  void DrawSpiral(unsigned char img[][NUMBER_OF_COLUMNS])
99  {
100     InitializeImage(img, 255);     // Initialize the image with a white background
101
102     float alfa = 0;
103     int rad = 0;
104     int centerX = NUMBER_OF_COLUMNS / 2;
105     int centerY = NUMBER_OF_ROWS / 2;
106
107     // Determine the maximum radius to ensure the spiral fits within the image boundaries
108     int maxRadius = std::min(centerX, centerY);
109
110     // Calculate the increment for alfa to control the density and length of the spiral
111     float increment = M_PI / 20000;
112
113     while (rad < maxRadius)
114     {
115         rad = static_cast<int>(3.5 * alfa);
116         for (int i = 0; i < 3; i++)
117         {
118             int x = static_cast<int>(centerY + (rad + i) * sin(alfa));
119             int y = static_cast<int>(centerX + (rad + i) * cos(alfa));
120             if (x >= 0 && x < NUMBER_OF_ROWS && y >= 0 && y < NUMBER_OF_COLUMNS)
121             {
122                 img[x][y] = 0;
123             }
124         }
125         alfa += increment;
126     }
127 }
```

alfa is the angle used in the spiral equation.
rad is the radius of the spiral.
centerX and centerY represent the center of the image.

The while loop continues until the radius rad exceeds the maximum radius. The radius rad is calculated as 3.5 * alfa. This equation defines how the spiral expands as alfa increases. For each step, a small segment of the spiral is drawn by incrementing the radius slightly (from rad to rad + 2).

The pixel coordinates x and y are calculated using polar-to-Cartesian conversion:

$$x = centerY + rad \times \sin(alfa)$$
$$y = centerX + rad \times \cos(alfa)$$

14.4 - Code of the function “DrawGaussian”

```
81  void DrawGaussian(unsigned char img[][NUMBER_OF_COLUMNS], int centerX, int centerY, float sigmaX, float sigmaY)
82  {
83      double a, b, c;
84      unsigned char d;
85      for (int row = 0; row < NUMBER_OF_ROWS; row++)
86      {
87          for (int column = 0; column < NUMBER_OF_COLUMNS; column++)
88          {
89              a = (column - centerX) / (sigmaX * 2);
90              b = (row - centerY) / (sigmaY * 2);
91              c = 255.0 * exp(-a * a - b * b);
92              d = static_cast<unsigned char>(c + 0.5);
93              img[row][column] = d;
94          }
95      }
96  }
```

a and b are variables for the normalized distances from the center in the x and y directions, respectively.
c holds the calculated intensity value.
d is the final pixel value after rounding.

Two nested for loops that iterate through each pixel in the image.
For each pixel at (row, column):

- Calculate a, the normalized distance from centerX in the x direction, scaled by sigmaX.
- Calculate b, the normalized distance from centerY in the y direction, scaled by sigmaY.
- Calculate the intensity c using the Gaussian function:

$$c = 255.0 \times e^{(-a^2-b^2)}$$

- Convert c to an unsigned char value d by rounding (+0.5 for correct rounding).
- Assign the value d to the pixel at (row, column) in the image.

14.5 - Code of the “main” function


```
17  int main() {
18
19      s2dPoint p1(0, 0);
20      s2dPoint p2(NUMBER_OF_COLUMNS, NUMBER_OF_ROWS);
21
22      AddGrayRectangle(grayImg14_1, p1, p2, 0, 255);
23      DrawSpiral(grayImg14_1);
24
25      // Create Gaussian pattern
26      DrawGaussian(gauss1, NUMBER_OF_COLUMNS / 4, NUMBER_OF_ROWS / 4, 60, 50);
27      DrawGaussian(gauss2, NUMBER_OF_COLUMNS / 2, NUMBER_OF_ROWS / 2, 120, 100);
28      DrawGaussian(gauss3, 3 * NUMBER_OF_COLUMNS / 4, 3 * NUMBER_OF_ROWS / 4, 60, 50);
29
30      // Blend the images
31      blend(gauss1, gauss2, gauss1);
32      blend(gauss1, gauss3, gauss1);
33      blend(gauss1, grayImg14_1, grayImg14);
34
35      // Store images for checking
36      StoreGrayImageAsGrayBmpFile(grayImg14_1, "grayImage14_1.bmp");
37      StoreGrayImageAsGrayBmpFile(gauss1, "gauss1.bmp");
38      StoreGrayImageAsGrayBmpFile(grayImg14, "grayImage14.bmp");
39
40      return 0;
41  }
```

p1 and p2 are instances of s2dPoint representing two corners of the rectangle. The struct is located in the header file and makes sure to automatically validate the validity of the points.

Adds a gray rectangle to grayImg14_1 covering the entire image with a gradient from transparency 0 to gray level of 255 (white). grayImg14_1 is the name of the stand alone spiral pattern.

Draws the 3 gaussians in different locations on the image – each with different standard deviations (x &y)

Blends gauss1 and gauss2, storing the result back in gauss1. Then , Blends the result in gauss1 with gauss3, again storing the result back in gauss1.
Finally, blends the result in gauss1 with grayImg14_1, storing the final blended image in grayImg14.

(+) – some additional functions we had to use – part 1


```
129  void blend(unsigned char img_src1[][NUMBER_OF_COLUMNS], unsigned char img_src2[][NUMBER_OF_COLUMNS], unsigned char img_dst[][NUMBER_OF_COLUMNS]) {
130      for (int row = 0; row < NUMBER_OF_ROWS; row++) {
131          for (int column = 0; column < NUMBER_OF_COLUMNS; column++) {
132              img_dst[row][column] = max(img_src1[row][column], img_src2[row][column])*0.85;
133          }
134      }
135  }
```

Two nested for loops iterate through each pixel in the images.
For each pixel at (row, column): computes the maximum value between the corresponding pixels from the two source images.
The result is then multiplied by 0.85 to apply a blending factor.

```
73  void InitializeImage(unsigned char image[][NUMBER_OF_COLUMNS], int gray_level) {
74      for (int row = 0; row < NUMBER_OF_ROWS; row++) {
75          for (int col = 0; col < NUMBER_OF_COLUMNS; col++) {
76              image[row][col] = gray_level;
77          }
78      }
79  }
```

Initialize image is a simple function for creating the basis for the task.

(+) – some additional functions we had to use – part 2


```
53  void AddGrayRectangle(unsigned char image[][NUMBER_OF_COLUMNS], s2dPoint A, s2dPoint B1, unsigned char transparency, unsigned char grayLevel) {
54      // Ensure coordinates are within bounds and place is not occupied
55      if (!checkValidation(A, B1, image)) {
56          return;
57      }
58
59      int top = max(A.Y, B1.Y);
60      int bottom = min(A.Y, B1.Y);
61      int left = min(A.X, B1.X);
62      int right = max(A.X, B1.X);
63
64      // Apply blending technique to the region of the rectangle
65      for (int row = max(bottom, 0); row < min(top, NUMBER_OF_ROWS); row++) {
66          for (int col = max(left, 0); col < min(right, NUMBER_OF_COLUMNS); col++) {
67              image[row][col] = static_cast<unsigned char>(transparency * (image[row][col] / 255.0)
68                  + (255 - transparency) * (grayLevel / 255.0));
69          }
70      }
71  }
```

We already used these functions in Assignment 11 & 12 so they will just be presented for the protocol without further explanation

```
43  bool checkValidation(s2dPoint p1, s2dPoint p2, unsigned char image[][NUMBER_OF_COLUMNS])
44  {
45      if ((0 > p1.X || NUMBER_OF_COLUMNS < p1.X) || (0 > p1.Y || NUMBER_OF_ROWS < p1.Y))
46      {
47          printf("Out of boundaries\n");
48          return false;
49      }
50      return true;
51  }
```

14.6 - What did we learned

1. How to Apply mathematical formulas in a programming context (e.g., Gaussian distribution, trigonometric functions for drawing spirals).
2. Drawing shapes (e.g., rectangles and spirals) by converting geometric formulas into pixel operations.
3. Implementing and understanding coordinate transformations (e.g., polar to Cartesian for the spiral).
4. Learning about the effects of different parameters (e.g., sigma values) on Gaussian distributions.
5. Designing and implementing blending algorithms to combine multiple images.