Class Lab 8

Use two images for each operation to do the following operations and write down their advantages and disadvantages and explain your results:

1. Dilation (noisy_rectangle, noisy_fingerprint):

Algorithm:

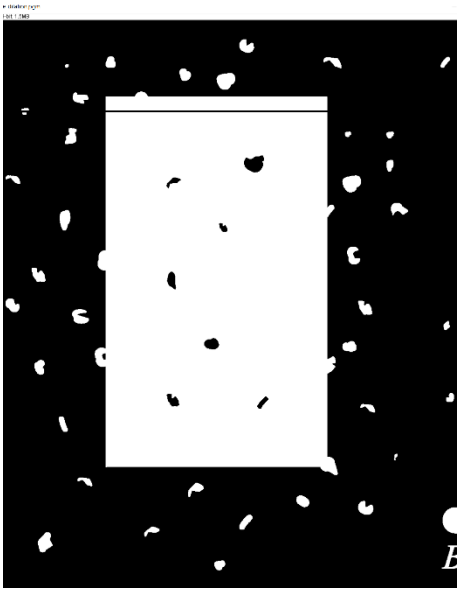
$$[f \oplus b](x,y) = \max_{(s,t) \in \hat{b}} \{f(x-s, y-t)\}$$

Results (including pictures):

Result of processing "noisy_rectangle.pgm":

Source Image:

Result after dilation:



Result of processing "noisy_fingerprint.pgm":

Source Image:

Result after dilation dilation:

Discussion:

Dilation “grows” or “thickens” objects in a binary image. The manner and extent of this thickening is controlled by the shape and size of the structuring element used.

Codes:

```
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        matrix[i * width + j] = tempin[i * width + j];
    }
}
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        if (i == 0 || j == 0 || i == height - 1 || j == width - 1) {
            if (flag == 1) {
                matrix[i * width + j] = 255;
            }
        } else {
            int max = 0;
            for (int k = -1; k <= 1; k++) {
                for (int l = -1; l <= 1; l++) {
                    if (tempin[(i + k) * width + j + l] > max) {
                        max = tempin[(i + k) * width + j + l];
                    }
                }
            }
            matrix[i * width + j] = max;
        }
    }
}
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        tempout[i * width + j] = matrix[i * width + j];
    }
}
free(matrix);
```

2. EROSION (noisy_rectangle, noisy_fingerprint):

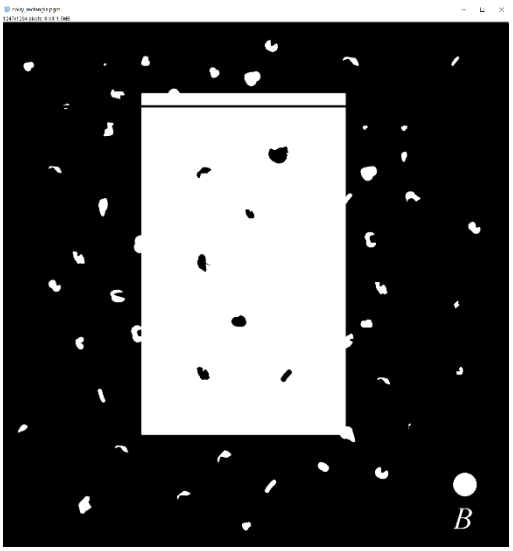
Algorithm:

$$[f \ominus b](x,y) = \min_{(s,t)\in b}\{f(x+s, y+t)\}$$

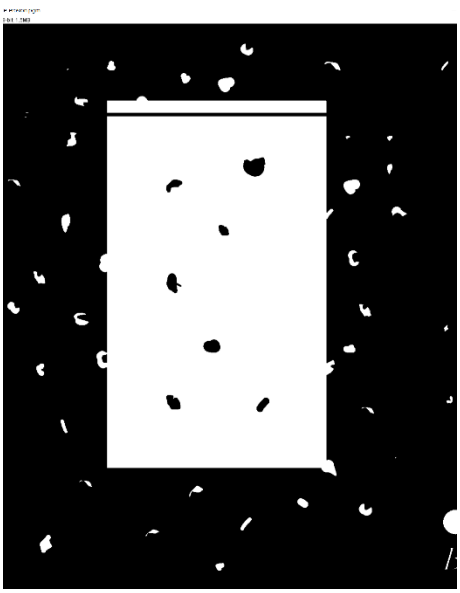
Results (including pictures):

Result of processing "noisy_rectangle.pgm":

Source Image:



Result after EROSION



Result of processing "noisy_fingerprint.pgm":

Source Image:

Result after EROSION:

Discussion:

Because grayscale erosion with a flat SE computes the minimum intensity value of f in every neighborhood of (,) x y coincident with b, we expect in general that an eroded grayscale image will be darker than the original, that the sizes (with respect to the size of the SE) of bright features will be reduced, and that the sizes of dark features will be increased.

Codes:

```
int height = image->Height;
int width = image->Width;
Image *outimage = CreateNewImage(image, height, width, (char *)"#Erosio
unsigned char *tempin = image->data;
unsigned char *tempout = outimage->data;
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        matrix[i * width + j] = tempin[i * width + j];
    }
}

for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        if (i == 0 || j == 0 || i == height - 1 || j == width - 1) {
            matrix[i * width + j] = tempin[i * width + j];
        } else {
            int min = 255;
            for (int k = -1; k <= 1; k++) {
                for (int l = -1; l <= 1; l++) {
                    if (tempin[(i + k) * width + j + l] < min) {
                        min = tempin[(i + k) * width + j + l];
                    }
                }
            }
            matrix[i * width + j] = min;
        }
    }
}
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        tempout[i * width + j] = matrix[i * width + j];
    }
}
free(matrix);
return outimage;
```

3. Opening (noisy_rectangle, noisy_fingerprint):

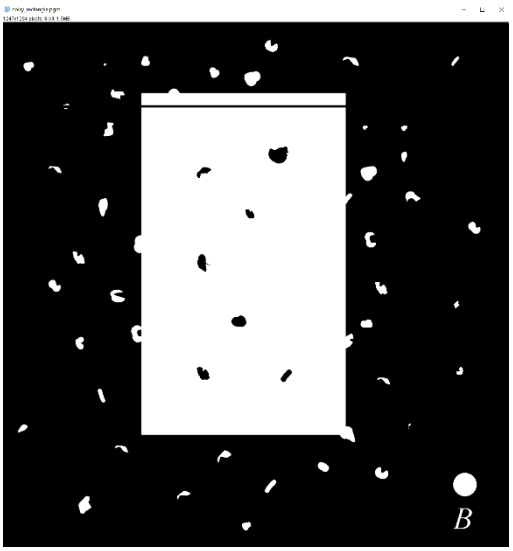
Algorithm:

1. Erosion
2. Dilation

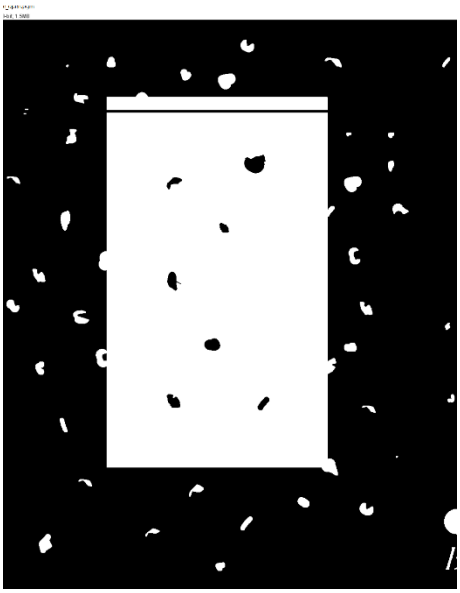
Results (including pictures):

Result of processing “noisy_rectangle.pgm”:

Source Image:



Result after Opening



Result of processing “noisy_fingerprint.pgm”:

Source Image:

Result after Opening:

Discussion:

(1) The open operation can remove isolated small points, burrs and small bridges, and the overall position and shape are inconvenient.

(2) Open operation is a filter based on geometric operation.

(3) Different sizes of structuring elements will lead to different filtering effects.

(4) The selection of different structural elements leads to different segmentations, that is, different features are extracted.

Codes:

```
Image *OpenImage(Image *image) {
    Image *outimage;
    outimage = ErosionImage(image);
    outimage = DilationImage(outimage);
    return outimage;
}
```

4. Closing (noisy_rectangle, noisy_fingerprint):

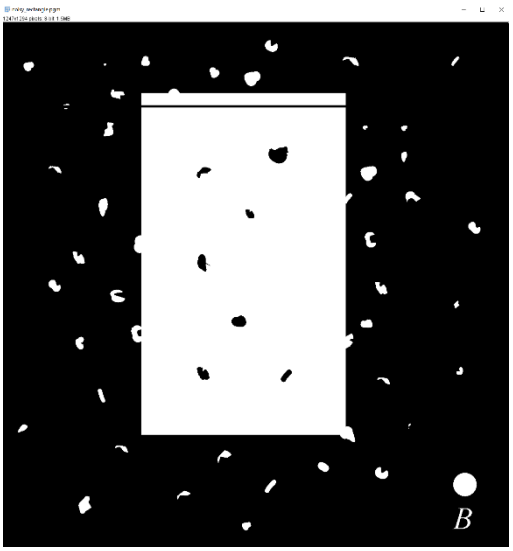
Algorithm:

1. Dilation
2. Erosion

Results (including pictures):

Result of processing “noisy_rectangle.pgm”:

Source Image:



Result after Closing

Result of processing “noisy_fingerprint.pgm”:

Source Image:

Result after Closing:

Discussion:

(1) The closing operation can fill in small lakes (ie small holes) and bridge small cracks, while the overall position and shape remain unchanged.

(2) The closing operation filters the image by filling the concave corners of the image.

(3) Different sizes of structuring elements will lead to different filtering effects.

(4) The choice of different structural elements leads to different segmentations.

Codes:

```
Image *CloseImage(Image *image) {
    Image *outimage;
    outimage = DilationImage(image);
    outimage = ErosionImage(outimage);
    return outimage;
}
```

5. Extract the Boundaries (licoln, U):

Algorithm:

1. Origin image – Erosion image

Results (including pictures):

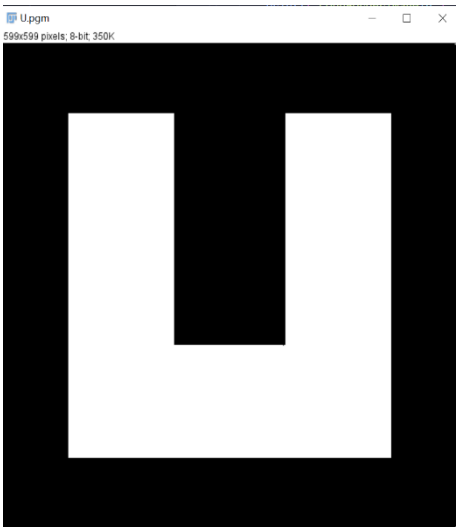
Result of processing “licoln.pgm”:

Source Image:

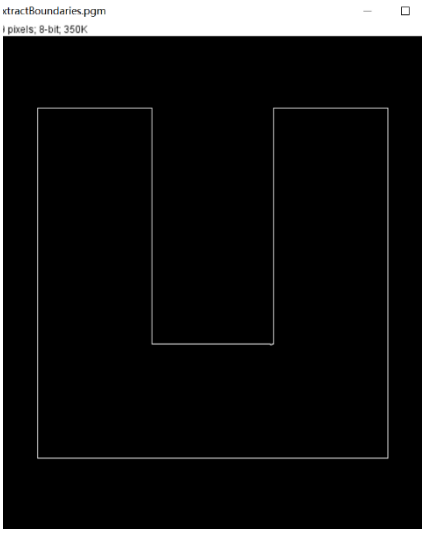
Result after Extract the Boundaries

Result of processing “U.pgm”:

Source Image:



Result after Extract the Boundaries:



Discussion:

When a 3x3 1-value struct is used, the extracted boundary value is 1 pixel.

Codes:

```
Image *ExtractBoundariesImage(Image *image) {
    Image *outimage;
    outimage = ErosionImage(image);
    int size = image->Height * image->Width;

    for (int i = 0; i < size; i++) {
        outimage->data[i] = image->data[i] - outimage->data[i];
    }

    return outimage;
}
```

6. Count the number of pixels in each white connected component (licoln, U):

Algorithm:

Suppose the original image is A0, the iteration matrix is X1, and B is the connected component storage matrix

1. A=A0
2. Find the point in A where the gray value is 1
3. Select the first point with a gray value of 1 as the initial point
4. Build a 3x3 1-value struct
5. Find the intersection after dilation
6. If X1 == Xp1, the iteration ends
7. B = B+X1
8. A = A-B

Continue looping until there are no white dots in A

Results (including pictures

```
The number of 0 th is 16
The number of 1 th is 2
The number of 2 th is 16
The number of 3 th is 1
The number of 4 th is 6
The number of 5 th is 16
The number of 6 th is 81
The number of 7 th is 2
The number of 8 th is 100
The number of 9 th is 16
The number of 10 th is 90
The number of 11 th is 20
The number of 12 th is 1
The number of 13 th is 2
The number of 14 th is 16
The number of 15 th is 16
The number of 16 th is 144
The number of 17 th is 380
The number of 18 th is 2
The number of 19 th is 2
The number of 20 th is 16
The number of 21 th is 2
The number of 22 th is 144
The number of 23 th is 20
The number of 24 th is 49
The number of 25 th is 16
The number of 26 th is 1
The number of 27 th is 16
The number of 28 th is 81
The number of 29 th is 42
The number of 30 th is 42
The number of 31 th is 23
The number of 32 th is 2
The number of 33 th is 12
The number of 34 th is 1
The number of 35 th is 16
The number of 36 th is 2
The number of 37 th is 2
The number of 38 th is 16
The number of 39 th is 100
The number of 40 th is 2
The number of 41 th is 2
The number of 42 th is 24
The number of 43 th is 1
The number of 44 th is 156
The number of 45 th is 90
The number of 46 th is 16
The number of 47 th is 90
The number of 48 th is 36
The number of 49 th is 42
The number of 50 th is 12
The number of 51 th is 2
The number of 52 th is 1
The number of 53 th is 1
The number of 54 th is 13
The number of 55 th is 16
The number of 56 th is 400
The number of 57 th is 144
The number of 58 th is 49
The number of 59 th is 144
The number of 60 th is 16
The number of 61 th is 144
The number of 62 th is 2
The number of 63 th is 42
```

The number of 58 th is 49
The number of 59 th is 144
The number of 60 th is 16
The number of 61 th is 144
The number of 62 th is 2
The number of 63 th is 42
The number of 64 th is 2
The number of 65 th is 1
The number of 66 th is 2
The number of 67 th is 16
The number of 68 th is 81
The number of 69 th is 42
The number of 70 th is 81
The number of 71 th is 16
The number of 72 th is 50
The number of 73 th is 100
The number of 74 th is 2
The number of 75 th is 42
The number of 76 th is 2
The number of 77 th is 20
The number of 78 th is 81
The number of 79 th is 1
The number of 80 th is 16
The number of 81 th is 49
The number of 82 th is 2
The number of 83 th is 7
The number of 84 th is 90
The number of 85 th is 16
The number of 86 th is 90
The number of 87 th is 16
The number of 88 th is 16
The number of 89 th is 2
The number of 90 th is 24
The number of 91 th is 1
The number of 92 th is 42
The number of 93 th is 16
The number of 94 th is 49
The number of 95 th is 1
The number of 96 th is 1
The number of 97 th is 144
The number of 98 th is 1
The number of 99 th is 1
The number of 100 th is 400
The number of 101 th is 90
The number of 102 th is 16
The number of 103 th is 1
The number of 104 th is 24
The number of 105 th is 1
The number of 106 th is 16
The number of 107 th is 16
The number of 108 th is 1
The number of 109 th is 2
The number of 110 th is 144
The number of 111 th is 2
The number of 112 th is 24
The number of 113 th is 16
The number of 114 th is 81
The number of 115 th is 1
The number of 116 th is 2
The number of 117 th is 2
The number of 118 th is 2
The number of 119 th is 16

Discussion:

Repeated iteration, the efficiency is too low

Codes:

```c
void CountConnPixel(Image *image, char *output) {
    int height = image->Height;
    int width = image->Width;
    int size = height * width;
    Pixel *pixel = (Pixel *)malloc(sizeof(Pixel) * size);
    unsigned char *A = (unsigned char *)malloc(sizeof(unsigned char) * size);
    memcpy(A, image->data, size);
    unsigned char *X1 = (unsigned char *)malloc(sizeof(unsigned char) * size);
    memset(X1, 0, sizeof(unsigned char) * size);
    unsigned char *B = (unsigned char *)malloc(sizeof(unsigned char) * size);
    memset(B, 0, sizeof(unsigned char) * size);
    int *num = (int *)malloc(sizeof(int) * size);
    memset(num, 0, sizeof(int) * size);

    int count = 0;
    FILE *fp = fopen(output, "w");

    while (find(A, 255, pixel, height, width)) {
        X1[pixel[0].y * width + pixel[0].x] = 255;
        while (1) {
            unsigned char *Xp1 = (unsigned char *)malloc(sizeof(unsigned char) * size);
            memcpy(Xp1, X1, size);
            X1 = Dilation(X1, height, width, 0);
            for (int i = 0; i < size; i++) {
                if (X1[i] > A[i]) {
                    X1[i] = A[i];
                }
            }
            if (memcmp(Xp1, X1, size) == 0) {
                break;
            }
            free(Xp1);
        }
        num[count] = find(X1, 255, pixel, height, width);
        fprintf(fp, "The number of %d th is %d\n", count, num[count]);
        for (int i = 0; i < size; i++) {
            if (B[i] < X1[i]) {
                B[i] = X1[i];
            }
        }
        memset(X1, 0, sizeof(unsigned char) * size);
        for (int i = 0; i < size; i++) {
            A[i] -= B[i];
            if (A[i] < 100) {
                A[i] = 0;
            }
        }
        count++;
    }
    free(pixel);
    free(A);
    free(X1);
    free(B);
    free(num);
    fclose(fp);
    printf("Finshed!\n");
}
```

7. Particles that only coincide with the edges of the image (bubbles_on_black_background):

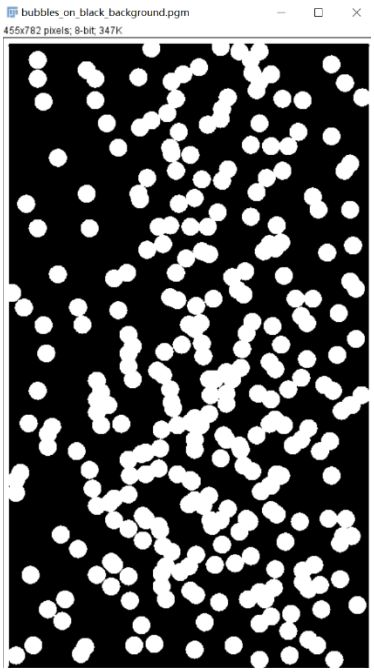
Algorithm:

Color the image border pixels the same color as the particles (white). Call the resulting set of border pixels B. Apply the connected component algorithm. All connected components that contain elements from B are particles that have merged with the border of the image

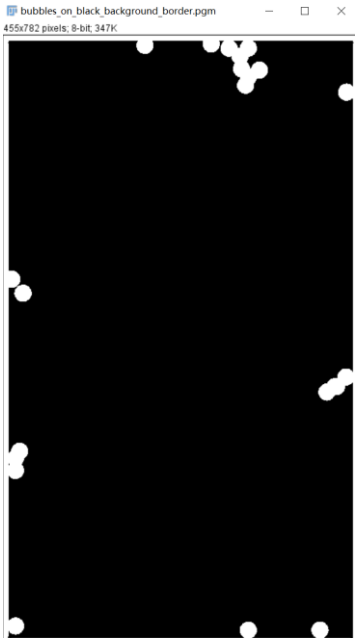
Results (including pictures):

Result of processing "bubbles_on_black_background.pgm":

Source Image:



Result:



Discussion:

Extraction of connected components for a fully automated process

Codes:

```
Image *ConnBorderImage(Image *image) {
    int height = image->Height;
    int width = image->Width;
    int size = height * width;
    Image *outimage = CreateNewImage(image, height, width, (char *)"#ConnBorder Image");
    unsigned char *tempin = image->data;
    unsigned char *tempout = outimage->data;
    unsigned char *B = (unsigned char *)malloc(sizeof(unsigned char) * size);
    memset(B, 0, sizeof(unsigned char) * size);

    B[0] = 255;
    while (1) {
        unsigned char *Xp1 = (unsigned char *)malloc(sizeof(unsigned char) * size);
        memcpy(Xp1, B, size);
        B = Dilation(B, height, width, 1);
        for (int i = 0; i < size; i++) {
            if (B[i] > tempin[i]) {
                B[i] = tempin[i];
            }
        }
        if (memcmp(Xp1, B, size) == 0) {
            break;
        }
        free(Xp1);
    }
    for (int i = 0; i < size; i++) {
        tempout[i] = B[i];
    }
    free(B);

    return outimage;
}
```

8. Only particles that overlap each other (bubbles_on_black_background):

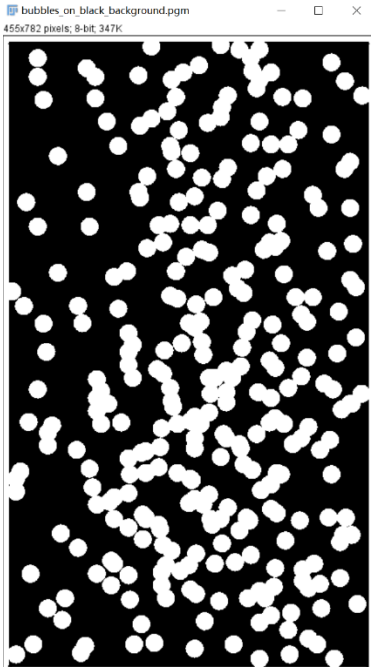
Algorithm:

It is given that all particles are of the same size. Determine the area (number of pixels) of a single particle; denote the area by A. Eliminate from the image the particles that were merged with the border of the image. Apply the connected component algorithm. Count the number of pixels in each component. A component is then designated as a single particle if the number of pixels is less than or equal to $A + \varepsilon$, where ε is a small quantity added to account for variations in size due to noise.

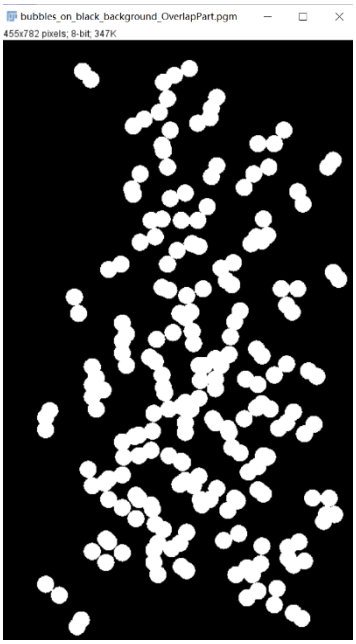
Results (including pictures):

Result of processing "bubbles_on_black_background.pgm":

Source Image:



Result:



Discussion:

Extraction of connected components for a fully automated process

Codes:

```
Image *outimage = CreateNewImage(image, height, width, (char *)"#OverlapPart Image");
outimage = ConnBorderImage(image);
unsigned char *tempin = image->data;
unsigned char *tempout = outimage->data;
for (int i = 0; i < size; i++) {
    tempout[i] = tempin[i] - tempout[i];
}

unsigned char *A = (unsigned char *)malloc(sizeof(unsigned char) * size);
memcpy(A, tempout, size);
unsigned char *X1 = (unsigned char *)malloc(sizeof(unsigned char) * size);
memset(X1, 0, sizeof(unsigned char) * size);
unsigned char *X2 = (unsigned char *)malloc(sizeof(unsigned char) * size);
memset(X2, 0, sizeof(unsigned char) * size);
unsigned char *B = (unsigned char *)malloc(sizeof(unsigned char) * size);
memset(B, 0, sizeof(unsigned char) * size);
int *num = (int *)malloc(sizeof(int) * size);
memset(num, 0, sizeof(int) * size);
int n = 400;
int count = 0;
while (find(A, 255, pixel, height, width)) {
    X1[pixel[0].y * width + pixel[0].x] = 255;
    while (1) {
        unsigned char *Xp1 = (unsigned char *)malloc(sizeof(unsigned char) * size);
        memcpy(Xp1, X1, size);
        X1 = Dilation(X1, height, width, 0);
        for (int i = 0; i < size; i++) {
            if (X1[i] > A[i]) {
                X1[i] = A[i];
            }
        }
        if (memcmp(Xp1, X1, size) == 0) {
            break;
        }
        free(Xp1);
    }
    num[count] = find(X1, 255, pixel, height, width);
    if (num[count] > n) {
        for (int i = 0; i < size; i++) {
            if (X2[i] < X1[i]) {
                X2[i] = X1[i];
            }
        }
    }
    count++;
    for (int i = 0; i < size; i++) {
        if (B[i] < X1[i]) {
            B[i] = X1[i];
        }
    }
    memset(X1, 0, sizeof(unsigned char) * size);
    for (int i = 0; i < size; i++) {
        A[i] -= B[i];
        if (A[i] < 100) {
            A[i] = 0;
        }
    }
}
for (int i = 0; i < size; i++) {
```

9. Only particles that not overlap each other (bubbles_on_black_background):

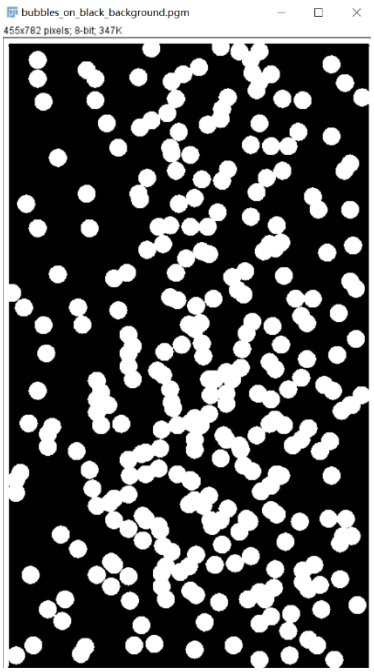
Algorithm:

Subtract from the image single particles and the particles that have merged with the border, and the remaining particles are overlapping particles.

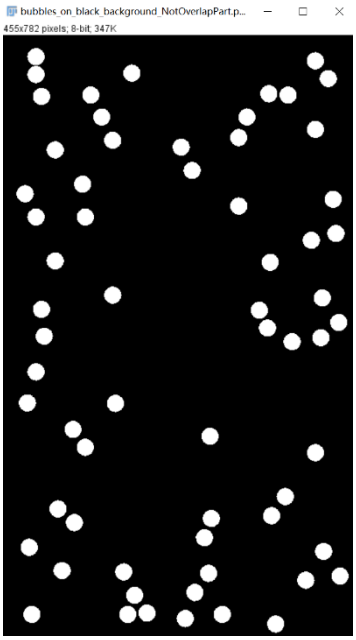
Results (including pictures):

Result of processing "bubbles_on_black_background.pgm":

Source Image:



Result:



Discussion:

When an operation is performed with this code, it cannot be performed at the same time as the previous one. The specific reason is that the *OverlapPartImage* function is called twice, resulting in some memory not being released when the C language manipulates the memory.

Codes:

```
Image *NotOverlapPartImage(Image *image) {
    int height = image->Height;
    int width = image->Width;
    int size = height * width;
    Image *tempimage1;
    Image *tempimage2;
    Image *outimage = CreateNewImage(image, height, width, (char *)"#NotOverlapPart Image");
    tempimage1 = OverlapPartImage(image);
    printf("OverlapPartImage\n");
    tempimage2 = ConnBorderImage(image);
    printf("ConnBorderImage\n");
    unsigned char *tempin = image->data;
    unsigned char *temp1 = tempimage1->data;
    unsigned char *temp2 = tempimage2->data;
    unsigned char *tempout = outimage->data;
    for (int i = 0; i < size; i++) {
        tempout[i] = tempin[i] - temp1[i] - temp2[i];
    }

    return outimage;
}
```