Edge Detection

Dr Frazer Noble

Introduction

In this presentation, I will describe:

- How to use OpenCV to detect edges in an image.

Requirements

To follow along with this tutorial, you will need the following tools:

- Python 3.8.6.
- Visual Studio Code 1.53.1.

You will also need to install the following Python packages:

- OpenCV.
- NumPy.

It is assumed that you are using Windows; however, these instructions should be easily adapted to Linux.

Getting Started

Open Visual Studio Code. To open the app: Open the Start menu, type `Visual Studio Code`, and then select the app.

Open the Explorer tab. To display the tab: Left click `View > Explorer` or press `Ctrl` + `Shift` + `E`. This will display the Explorer tab.

Left click on the `Open Folder` button. This will display the Open Folder prompt. Browse to the following directory:

```
C:/Users/%USER%/Documents
```

Note: Replace `%USER%` with your own username. My username is fknoble; hence, the path is `C:/Users/fknoble/Documents`.

In `C:/Users/%USER%/Documents` create a new folder named `opencv_03`. To create a new folder: Right click in the Explorer tab, left click `New Folder`, and rename it.

In `C:/Users/%USER%/Documents/opencv_03` create a new folder named `data`. Download `apples.PNG` from here; save it in `C:/Users/%USER%/Documents/opencv_03/data`.

In `C:/Users/%USER%/Documents/opencv_03` create new files named `sobel.py` and `canny.py`. To create a new file: Right click on `/opencv_03` in the Explorer tab, left click `New File`, and rename it. The file will open automatically.

`/opencv_03` should contain the following files and folders:

```
/opencv_03
    /data
        apple.PNG
    sobel.py
    canny.py
```

`sobel.py`

Type the following code into `sobel.py`:

```
import cv2 as cv
import numpy as np
```

OpenCV's Python module `cv2` is imported as `cv` and NumPy's Python module `numpy` is imported as `np`.

Type the following code into `sobel.py`:

```
def main():

    img = cv.imread('data/apples.PNG')

    if img is None:
        print('ERROR::CV::Could not read image.')
        return 1
```

This begins `main()`'s definition. `imread()` reads an image from a directory and assigns the results to array `img`. If the array is empty, a message is displayed and `main()` returns 1.

Type the following code into `sobel.py`:

```
    rows, cols, channels = img.shape

    rows = rows // 2
    cols = cols // 2

    img = cv.resize(img, (cols, rows))

    cv.imshow('img', img)
    cv.waitKey(1)
```

`img`'s shape is assigned to integers `rows`, `cols`, and `channels`. `rows` and `cols` are divided by 2 (rounded down) and the results assigned to themselves. `resize()` resizes `img` to shape `cols` x `rows` and the result is assigned to itself. The array is then displayed in the `img` window.

Figure: The `img` array.

Type the following code into `sobel.py`:

```
    img_blur = cv.GaussianBlur(img, (3, 3), 0)

    img_gray = cv.cvtColor(img_blur, cv.COLOR_BGR2GRAY)

    cv.imshow('img_gray', img_gray)
    cv.waitKey(1)
    cv.imwrite('data/img_gray.PNG', img_gray)
```

`GaussianBlur()` applies a Gaussian filter to `img` and assigns the results to array `img_blur`. `cvtColor()` converts `img_blur` from a colour image to a grayscale image and assigns the results to array `img_gray`. The array is then displayed in the `img_gray` window and saved as `img_gray.PNG` in `/data`.

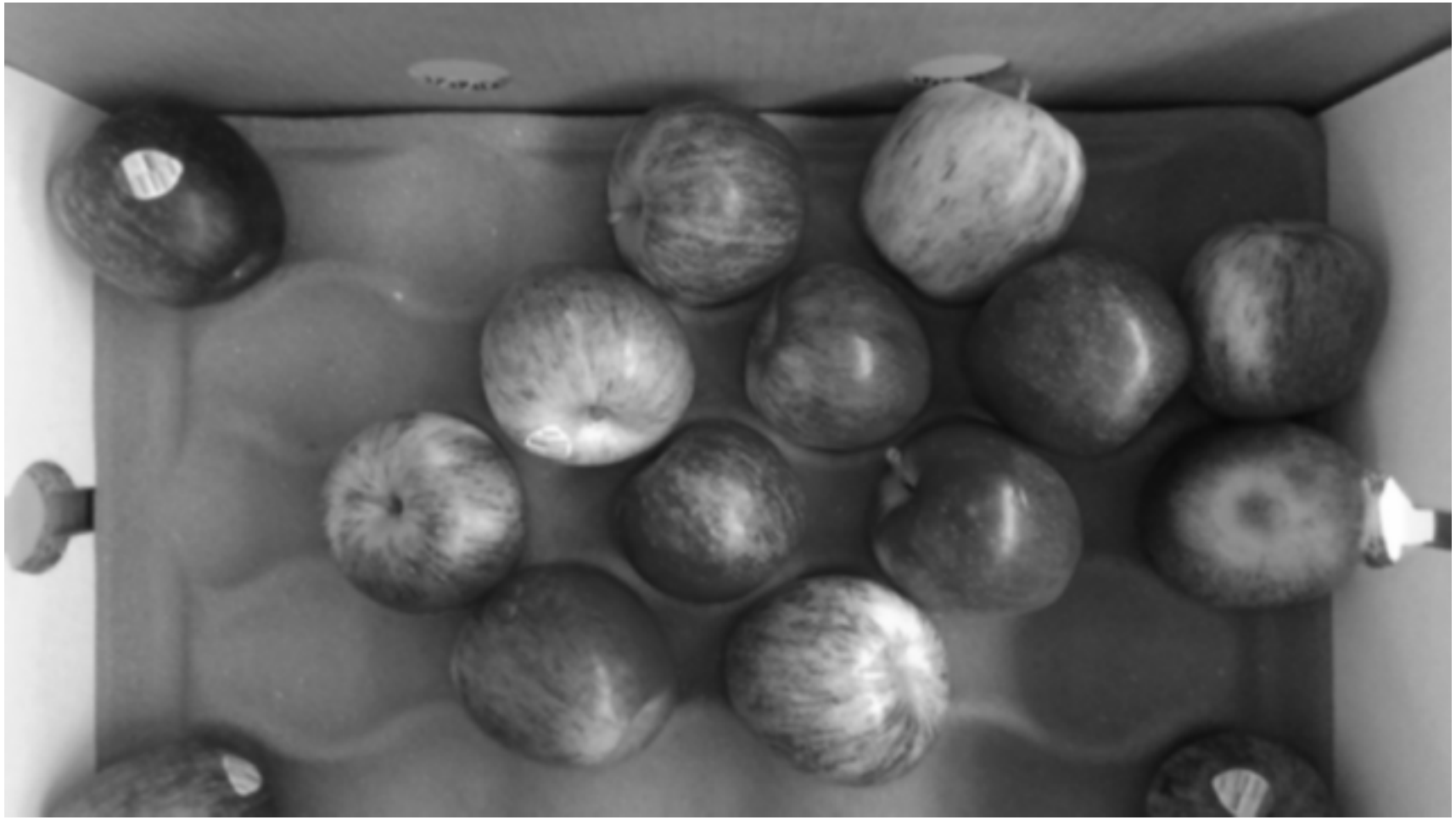

Figure: The `img_gray` array.

Type the following code into `sobel.py`:

```
    vertical = np.float32([-1, 0, 1, -2, 0, 2, -1, 0, 1])
    vertical = vertical.reshape((3, 3))

    vertical_edges = cv.filter2D(img, cv.CV_32FC1, vertical)
    vertical_edges = np.abs(vertical_edges)
    G_x = vertical_edges/vertical_edges.max() * 255
    G_x = np.uint8(G_x)

    cv.imshow("G_x", G_x)
    cv.waitKey(1)
    cv.imwrite("data/G_x.PNG", G_x)
```

This snippet defines an array named `vertical`, which is assigned a sobel kernel's values. `reshape()` reshapes `vertical` into a 3 x 3 array. `filter2D()` applies the kernel to `img_gray` and assigns the results to `vertical_edges`. `vertical_edges`'s values are normalised and assigned to `G_x`. The array is then displayed in the `G_x` window and saved as `G_x.PNG` in `/data`.

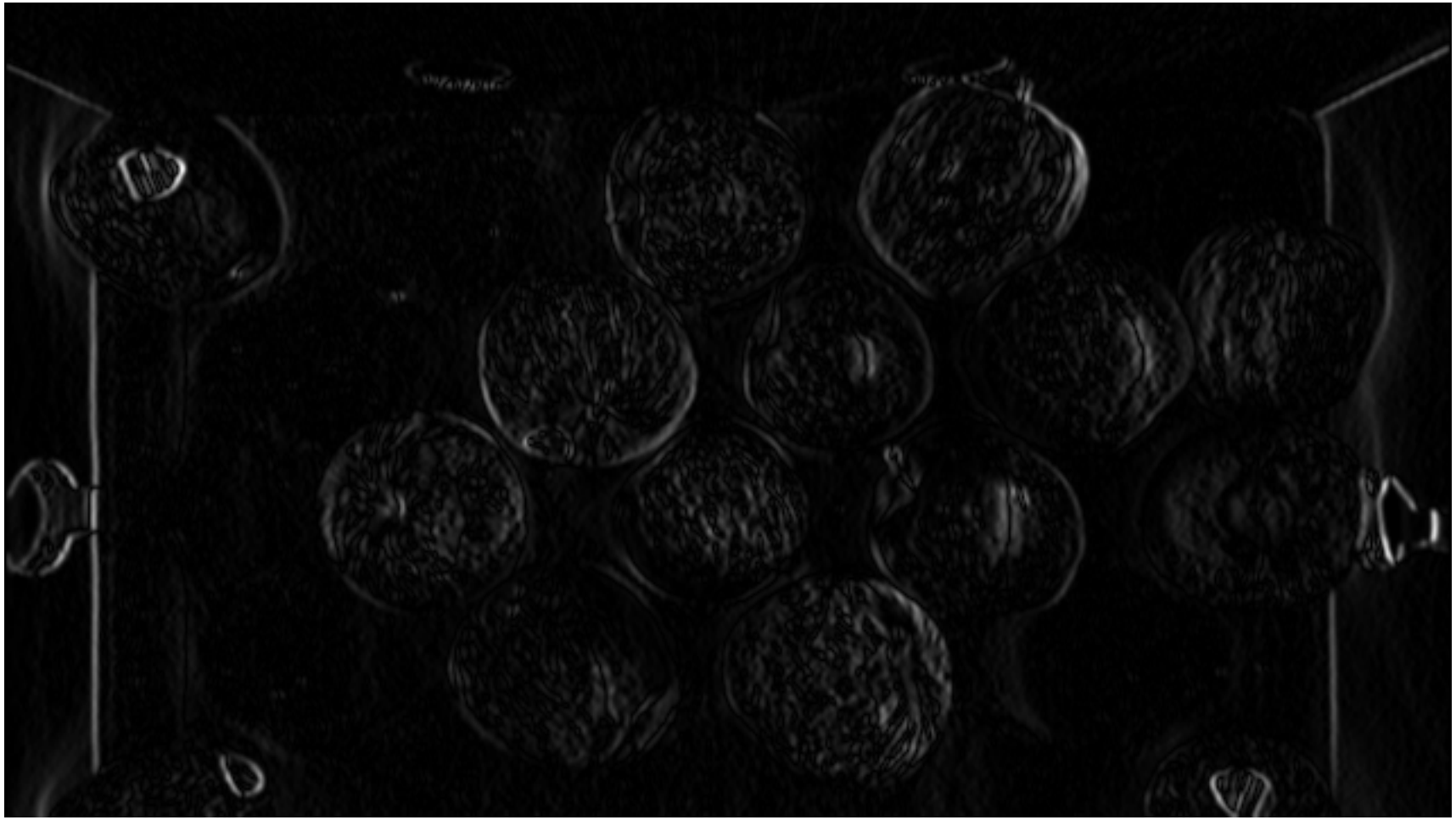

Figure: The `G_x` array.

Type the following code into `sobel.py`:

```
    horizontal = np.float32([-1, -2, -1, 0, 0, 0, 1, 2, 1])
    horizontal = horizontal.reshape((3, 3))

    horizontal_edges = cv.filter2D(img, cv.CV_32FC1, horizontal)
    horizontal_edges = np.abs(horizontal_edges)
    G_y = horizontal_edges/horizontal_edges.max() * 255
    G_y = np.uint8(G_y)

    cv.imshow("G_y", G_y)
    cv.waitKey(1)
    cv.imwrite("data/G_y.PNG", G_y)
```

This snippet defines an array named `horizontal`, which is assigned a sobel kernel's values. `reshape()` reshapes the `horizontal` into a 3 x 3 array. `filter2D()` applies the kernel to `img_gray` and assigns the results to `horizontal_edges`. `horizontal_edges`'s values are normalised and assigned to `G_y`. The array is then displayed in the `G_y` window and saved as `G_y.PNG` in `/data`.

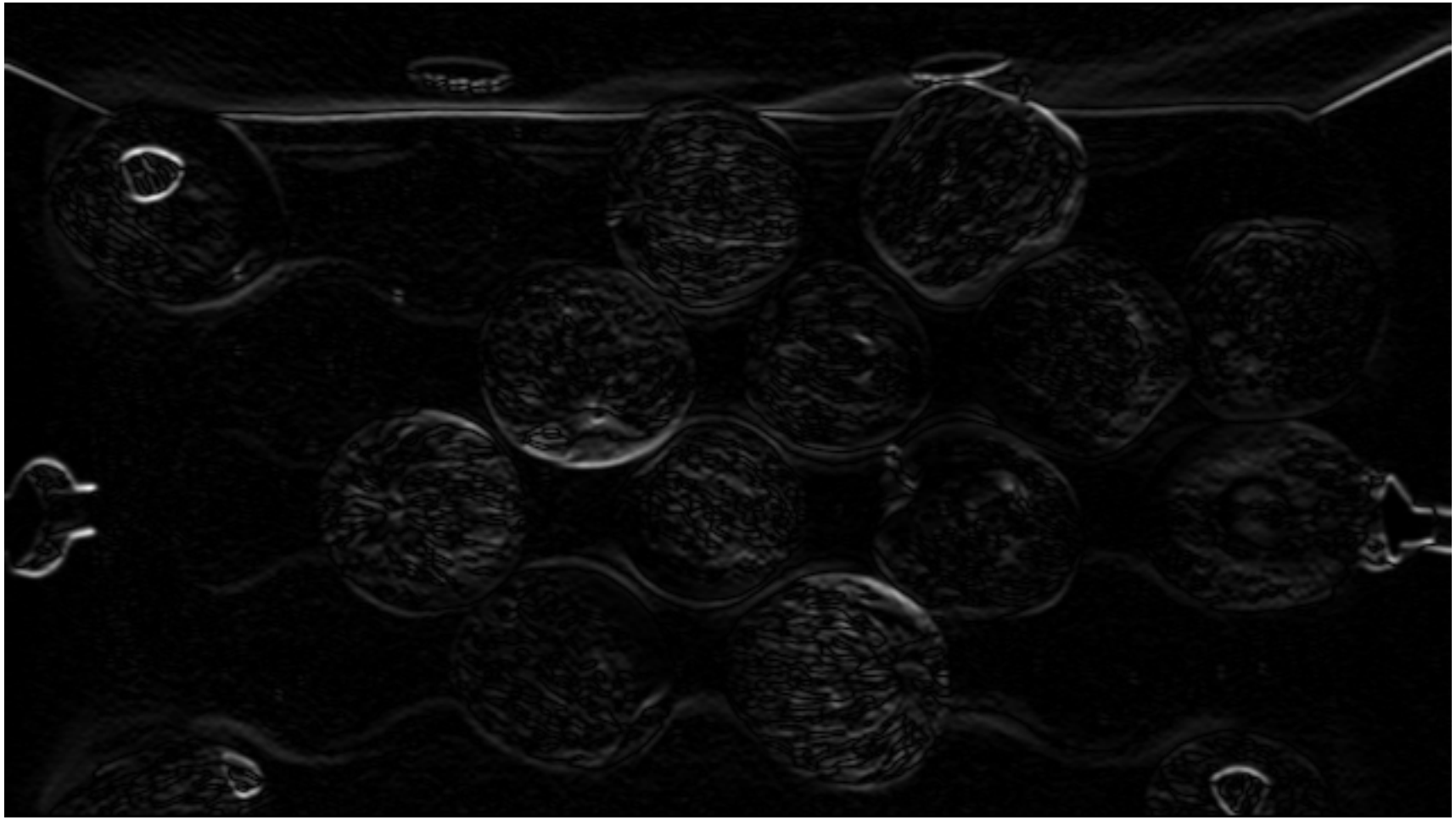

Figure: The `G_y` array.

Type the following code into `sobel.py`:

```
    G = np.hypot(G_x, G_y)
    G = G/G.max() * 255
    G = np.uint8(G)

    cv.imshow("G", G)
    cv.waitKey(0)
    cv.imwrite("data/G.PNG", G)

    cv.destroyAllWindows()

    return 0
```

This snippet defines an array named `G`, which is assigned the hypotenuse of `G_x` and `G_y`. `G` is normalised and converted to a `uint8` data type. The array is then displayed in the `G` window and saved as `G.PNG` in `/data`.

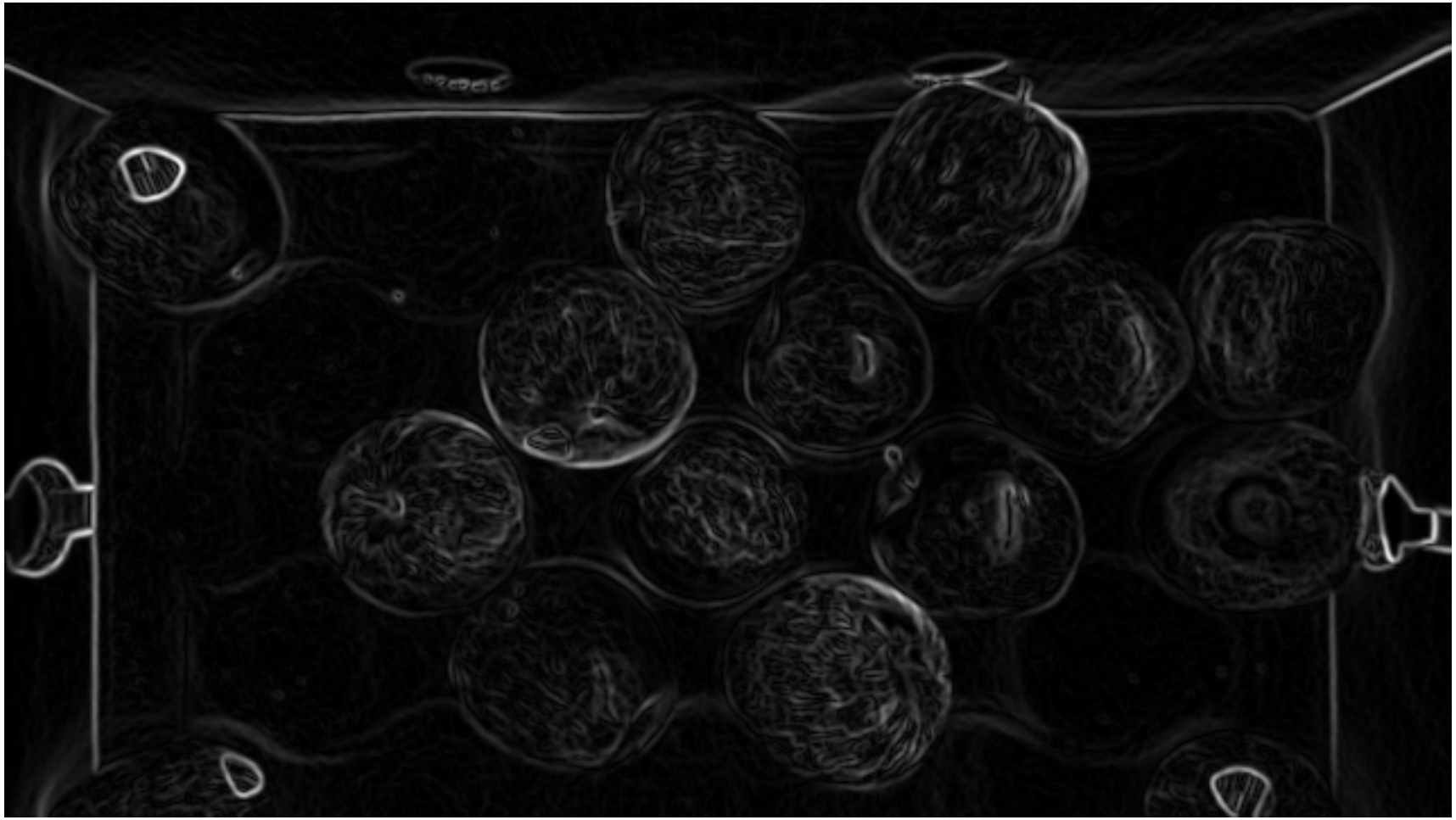

Figure: The `G` array.

Type the following code into `sobel.py`:

```
if __name__ == '__main__':

    main()
```

`main()` will be called when the `sobel.py` is run.

Run `sobel.py`

Open a new terminal in Visual Studio Code. To open a new terminal: Left click `View > Terminal` or press `Ctrl`+`` ` ``.

Type the following commands into the terminal and then press `Enter` after each one:

```
cd ./opencv_03
python sobel.py
```

This will change the current directory to the `/opencv_03` sub-directory and then run `sobel.py`.

Press any key to close the windows and stop `sobel.py`.

`canny.py`

Type the following code into `canny.py`:

```
import cv2 as cv
import numpy as np
```

OpenCV's Python module `cv2` is imported as `cv` and NumPy's Python module `numpy` is imported as `np`.

Type the following code into `canny.py`:

```
def main():

    img = cv.imread('data/apples.PNG')

    if img is None:
        print('ERROR::CV::Could not read image.')
        return 1
```

This begins `main()`'s definition. `imread()` reads an image from a directory and assigns the results to array `img`. If the array is empty, a message is displayed and the `main()` returns 1.

Type the following code into `canny.py`:

```
    rows, cols, channels = img.shape

    rows = rows // 2
    cols = cols // 2

    img = cv.resize(img, (cols, rows))

    cv.imshow('img', img)
    cv.waitKey(1)
```

`img`'s shape is assigned to integers `rows`, `cols`, and `channels`. `rows` and `cols` are divided by 2 (rounded down) and the results assigned to themselves. `resize()` resizes `img` to shape `cols` x `rows` and the result is assigned to itself. The array is then displayed in the `img` window.

Figure: The `img` array.

Type the following code into `canny.py`:

```
    img_blur = cv.GaussianBlur(img, (3, 3), 0)

    img_gray = cv.cvtColor(img_blur, cv.COLOR_BGR2GRAY)

    cv.imshow('img_gray', img_gray)
    cv.waitKey(1)
    cv.imwrite('data/img_gray.PNG', img_gray)
```

`GaussianBlur()` applies a Gaussian filter to `img` and assigns the results to array `img_blur`. `cvtColor()` converts `img_blur` from a colour image to a grayscale image and assigns the results to array `img_gray`. The array is then displayed in the `img_gray` window and saved as `img_gray.PNG` in `/data`.

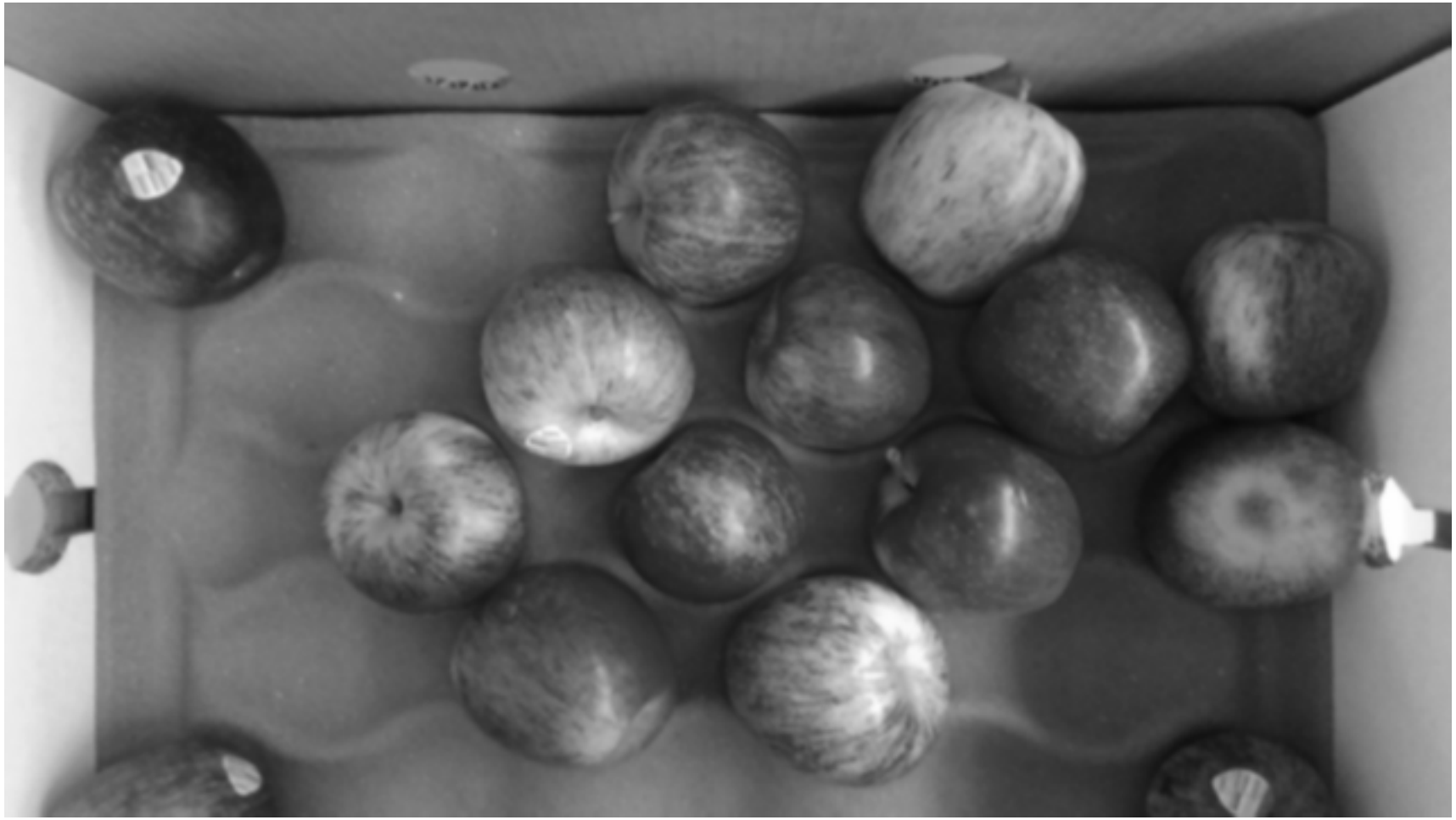

Figure: The `img_gray` array.

Type the following code into `canny.py`:

```
    canny = cv.Canny(img_gray, 20, 75)

    cv.imshow("canny", canny)
    cv.waitKey(0)
    cv.imwrite("data/canny.PNG", canny)

    cv.destroyAllWindows()

    return
```

`canny()` detects edges in `img_gray` and assigns the results to array `canny`. The array is displayed in the `canny` window and saved as `canny.PNG` in `/data`.

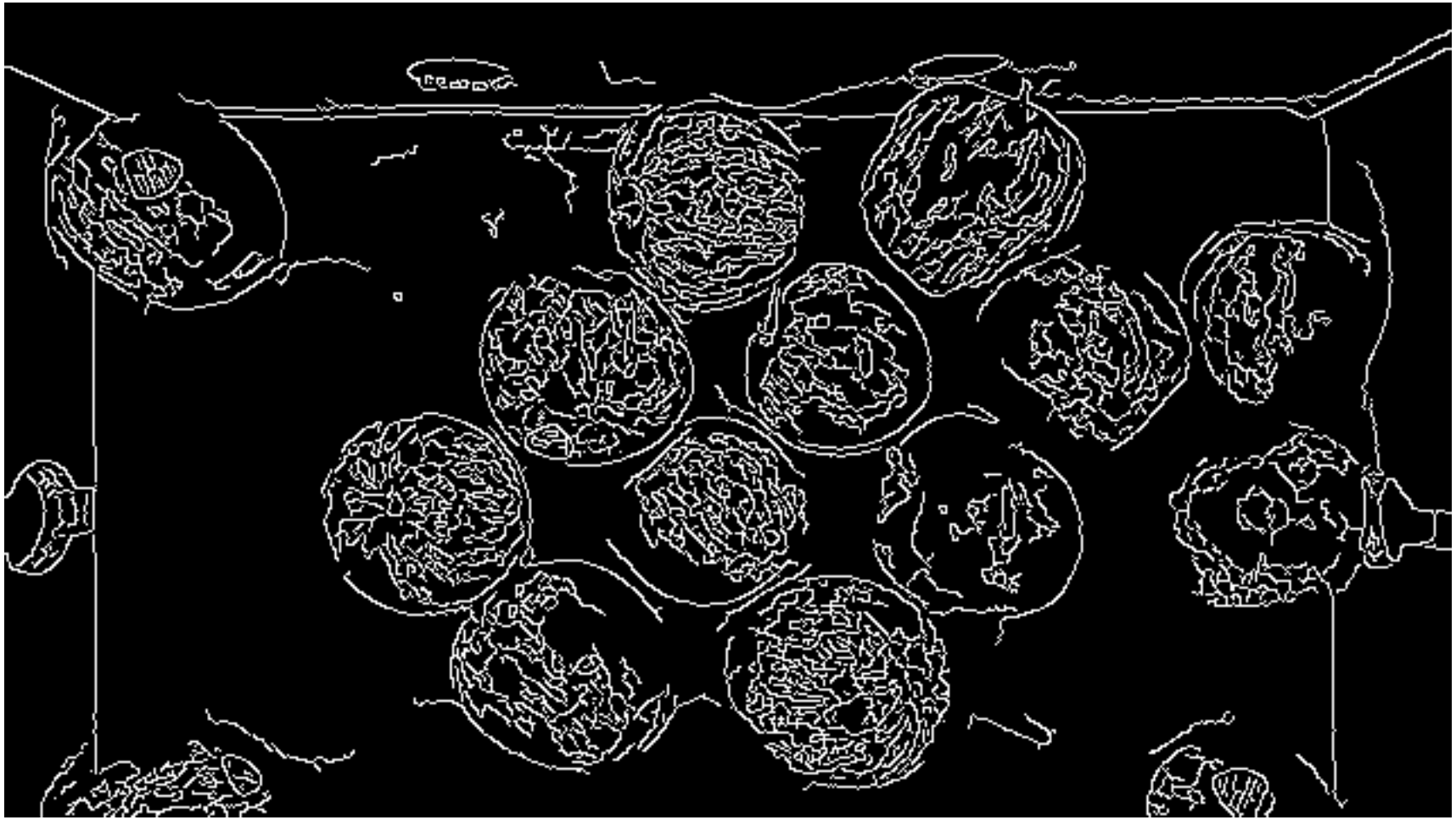

Figure: The `canny` array.

Type the following code into `canny.py`:

```
if __name__ == '__main__':

    main()
```

`main()` will be called when the `canny.py` is run.

Run `canny.py`

Open a new terminal in Visual Studio Code. To open a new terminal: Left click `View > Terminal` or press `Ctrl`+`` ` ``.

Type the following commands into the terminal and then press `Enter` after each one:

```
cd ./opencv_03
python canny.py
```

This will change the current directory to the `/opencv_03` sub-directory and then run `canny.py`.

Press any key to close the windows and stop `canny.py`.

Conclusion

In this presentation, I have described:

- How to use OpenCV to detect edges in an image.

References

1. https://docs.opencv.org/.