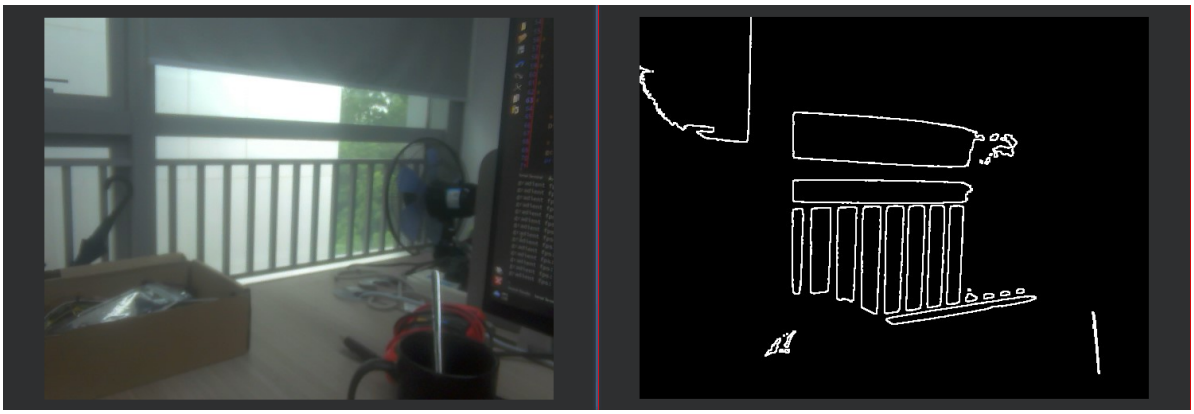
Gradient Transformation

Example Results

Run this section's example code [Source Code/06.cv_lite/18.rgb888_gradient.py]

In this section, we will use the `cv_lite` extension module to implement RGB888 image gradient processing on an embedded device.



Principle Explanation

What is an Image Gradient?

The image gradient is a morphological image processing operation used to extract boundary or edge information from objects in an image. It is achieved by calculating the difference between the results of the dilation and erosion operations. Gradient operations are often used for edge detection or contour extraction in image processing.

- **Operation Principle**: The morphological gradient is defined as the difference between the dilation and erosion results, i.e., `Gradient = Dilate(Image) - Erode(Image)`. The dilation operation expands the boundaries of white areas (foreground), while the erosion operation contracts them. Subtracting the two results in the boundary region, which is the edge of the white area in the original image.
- **Effect**: The gradient operation can highlight the outlines or boundaries of objects in an image and is suitable for binary or grayscale images. In the resulting image, boundary

regions typically appear white (high values), while other regions appear black (low values). This operation can be considered a simple edge detection method, especially effective in binary images.
- **Application Scenarios**: The morphological gradient is often used for edge extraction, object contour detection, and as an intermediate step in other complex morphological operations. For example, it is used to separate object boundaries in object detection and to extract structural edges in medical image processing.

In gradient operations, the size of the convolution kernel and the number of iterations significantly affect the results: larger kernels and more iterations result in wider boundary regions. Additionally, the binarization threshold (if any) determines which pixels are considered foreground (white) or background (black).

Code Overview

Importing Modules

```
import time, os, gc
from machine import Pin
from media.sensor import * # Camera interface
from media.display import * # Display interface
from media.media import * # Media manager
import cv_lite # AI CV extension module
import ulab.numpy as np
```

Setting the Image Size

```
image_shape = [480, 640]
```

Define the image resolution to 480x640 (height x width). The camera and display modules will be initialized based on this size.

Initialize the camera (RGB888 format)

```
sensor = Sensor(id=2, width=1280, height=960, fps=30)
sensor.reset()
sensor.set_framesize(width=image_shape[1], height=image_shape[0])
sensor.set_pixformat(Sensor.RGB888)
```

- Initialize the camera, set the resolution to 1280x960 and the frame rate to 30 FPS.
- Resize the output frame to 640x480 and set it to RGB888 format (three-channel color image, suitable for morphological operations on color images).

Initialize the display module

```
Display.init(Display.VIRT, width=image_shape[1], height=image_shape[0],
to_ide=True)
```

Initialize the display module, using the virtual display driver (`Display.VIRT`) with a resolution consistent with the image size. `to_ide=True` means the image will be simultaneously transferred to the IDE for virtual display.

Initialize the media manager and start the camera

```
MediaManager.init()
sensor.run()
```

Initialize the media resource manager and start the camera to capture the image stream.

Set gradient operation parameters

```
kernel_size = 3 # Kernel size (odd recommended)
iterations = 1 # Morphology iterations
threshold_value = 150 # Binarization threshold (0 = use Otsu automatic
thresholding)
clock = time.clock() # Start FPS timer
```

- `kernel_size`: Convolution kernel size, used for gradient operations (dilation and erosion). An odd number is recommended. Larger values result in wider borders.
- `iterations`: Number of gradient operation iterations. Larger values result in more pronounced border effects, but the computational effort increases.
- `threshold_value`: Binarization threshold, used for image processing before or after the gradient operation. A value of 0 uses the Otsu automatic thresholding algorithm.
- `clock`: Used to calculate the frame rate.

Image processing and gradient operation

```
while True:
    clock.tick()

    # Capture image and convert to ndarray
    img = sensor.snapshot()
    img_np = img.to_numpy_ref()

    # Call gradient operation (Gradient = dilate - erode)
    result_np = cv_lite.rgb888_gradient(image_shape, img_np, kernel_size,
iterations, threshold_value)

    # Construct grayscale image for display
    img_out = image.Image(image_shape[1], image_shape[0], image.GRAYSCALE,
                          alloc=image.ALLOC_REF, data=result_np)

    # Show image
    Display.show_image(img_out)

    # Garbage collection and FPS printing
    gc.collect()
    print("gradient fps:", clock.fps())
```

- **Image capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Gradient processing**: Call `cv_lite.rgb888_gradient()` to perform the morphological gradient operation (internal implementation is dilation minus erosion), returning the processed image as a NumPy array.
- **Image packaging and display**: Package the processed result into a grayscale image object and display it on the screen or in an IDE virtual window.

- **Memory management and frame rate output**: Call `gc.collect()` to clean up memory and print the current frame rate using `clock.fps()`.

Resource release

```
sensor.stop()
Display.deinit()
os.exitpoint(os.EXITPOINT_ENABLE_SLEEP)
time.sleep_ms(100)
MediaManager.deinit()
```

Parameter adjustment instructions

- `kernel_size`: Convolution kernel size. A larger value results in a wider extracted boundary area, which is suitable for processing larger objects or requiring more distinct edges. It is recommended to use an odd number (such as 3, 5, or 7) and start with 3 to balance details and boundary width.
- `iterations`: Number of iterations. A larger value results in a stronger gradient effect (boundary width), but may overemphasize the boundary and lose details. It is recommended to start with 1 and gradually increase it based on the image characteristics.
- `threshold_value`: Binarization threshold. Higher values require brighter pixels to be considered foreground. A value of 0 uses Otsu's automatic thresholding. It is recommended to test from 50 to 150 based on the image brightness, or set to 0 for automatic adaptation.