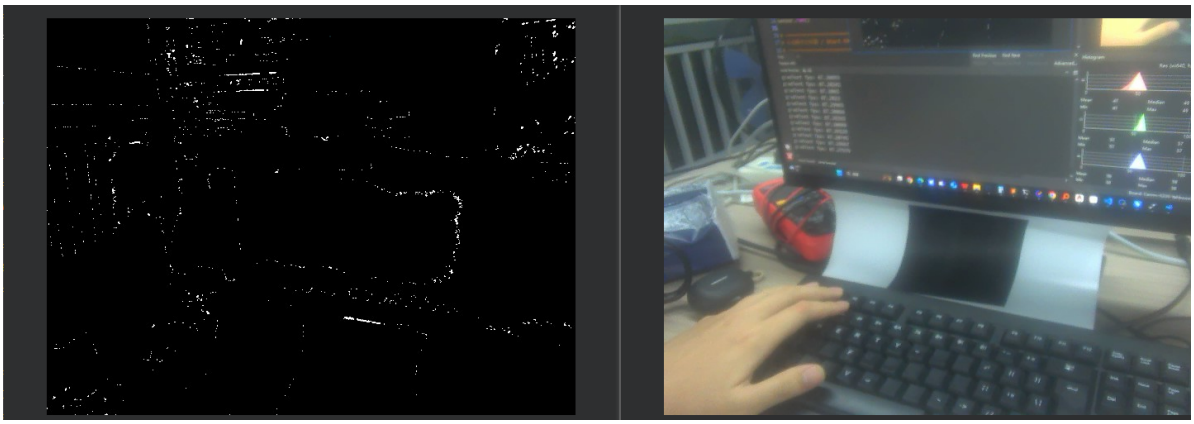
Image Top Hat Operation

Example Results

Run this section's example code [Source Code/06.cv_lite/22.rgb888_tophat.py]

In this section, we'll use the `cv_lite` extension module to implement a top-hat transform on an RGB888 image on an embedded device. This will be used to highlight bright areas and remove uneven lighting.



Principle Explanation

What is an Image Top-Hat Transformation?

The top-hat transform is a morphological image processing technique used to enhance local bright areas, remove uneven global lighting, or highlight small objects. Essentially, the top-hat transform is a difference calculation between the original image and the result of an opening operation. It is often used in preprocessing to improve image quality.

- **Operation Principle**: The top-hat operation is defined as the original image minus the result of the opening operation, i.e., `Top-Hat = Original Image - Opening(Original Image)`. The opening operation consists of dilation and erosion: first, erosion removes small objects and noise, then dilation restores the main structure. The top-hat operation retains pixels in the

original image that are brighter than the surrounding area, thereby highlighting small bright spots or details.

- **Effect**: The top-hat operation effectively removes uneven lighting in the background (such as shadows or light spots) and highlights bright details in the foreground, such as text, spots, or edges. It also helps enhance image contrast, but may introduce noise if the parameters are not set correctly. Unlike other morphological operations, the top-hat operation focuses more on brightness enhancement.
- **Application Scenarios**: The top-hat operation is commonly used for image enhancement, noise removal, text extraction (such as pre-processing for OCR), medical image processing (such as highlighting blood vessels or lesions), and industrial inspection (such as detecting surface defects). On embedded devices, it is suitable for real-time processing due to its relatively low computational overhead.

In the top-hat operation, the size of the convolution kernel and the number of iterations affect the results: larger kernels and iterations enhance global smoothing but may lose subtle details. Binarization thresholding is used in subsequent processing to further refine the results.

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 (Top-Hat function)
import ulab.numpy as np # NumPy-like ndarray for MicroPython
```

Setting the Image Size

```
image_shape = [480, 640] # Height x Width
```

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(width=1280, height=960)
sensor.reset()
sensor.set_framesize(width=image_shape[1], height=image_shape[0])
sensor.set_pixformat(Sensor.RGB888) # Set pixel format to RGB888
```

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

Initialize the display module

```
Display.init(Display.ST7701, width=image_shape[1], height=image_shape[0],
to_ide=True)
```

Initialize the display module, using the ST7701 driver chip, with a resolution consistent with the image size. `to_ide=True` indicates that the image will be simultaneously transmitted 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 the frame rate timer

```
clock = time.clock() # Start the frame rate timer / Start FPS timer
```

Initialize the frame rate timer, which is used to calculate and display the number of frames processed per second (FPS).

Set top-hat operation parameters

```
kernel_size = 3 # Kernel size (odd recommended)
iterations = 1 # Number of morphological passes
threshold_value = 100 # Binarization threshold (0 = Otsu)
```

- `kernel_size`: Kernel size, used for the opening operation in the top-hat operation. An odd number is recommended; larger values increase the processing range.
- `iterations`: Number of morphological operation iterations. A larger value results in a more pronounced enhancement effect, but increases the computational effort.
- `threshold_value`: Binarization threshold, used for processing the result. A value of 0 uses the Otsu thresholding algorithm.

Image processing and top-hat operation

```
while True:
    clock.tick()  # Start frame timing

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

    # Apply Top-Hat operation
    # Top-Hat = Original - Opening
    result_np = cv_lite.rgb888_tophat(
        image_shape,
        img_np,
        kernel_size,
        iterations,
        threshold_value
    )

    # Wrap result as image and display
    img_out = image.Image(image_shape[1], image_shape[0], image.GRAYSCALE,
                          alloc=image.ALLOC_REF, data=result_np)
    Display.show_image(img_out)
```

```
    # Cleanup and print FPS
    gc.collect()
    print("tophat fps:", clock.fps())
```

- **Image Capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Tophat Operation Processing**: Call `cv_lite.rgb888_tophat()` to perform the tophat operation, returning the processed grayscale image as a NumPy array.
- **Image Packaging and Display**: Pack 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() # Stop sensor
Display.deinit() # Deinit display
os.exitpoint(os.EXITPOINT_ENABLE_SLEEP) # Set safe exit point
time.sleep_ms(100) # Short delay
MediaManager.deinit() # Release media manager
```

Parameter adjustment instructions

- `kernel_size`: Convolution kernel size. Larger values (e.g., 3-7) provide a wider processing range and are suitable for removing background noise in a wider range, but may blur details. It is recommended to use an odd number and start testing with 3.
- `iterations`: Number of iterations. Larger values (e.g., 1-3) enhance the top-hat effect, but increase the computational effort and risk of losing edges. It is recommended to start with 1 and adjust based on image complexity.
- `threshold_value`: Binarization threshold. Higher values (e.g., 50-150) require brighter pixels to be retained. A value of 0 uses Otsu's automatic threshold. It is recommended to test based on image brightness, or set it to 0 for automatic adaptation.