Gaussian Blur

Example Results

Run this section's example code [Source Code/06.cv_lite/17.rgb888_gaussian_blur.py]

In this section, we will use the `cv_lite` extension module to implement Gaussian filtering of RGB888 images on an embedded device.

Principle Explanation

What is Image Gaussian Filtering?

Gaussian Blur is a commonly used image smoothing technique used to reduce noise and blur details. It uses a filter kernel designed based on a Gaussian distribution (normal distribution) and achieves smoothing by performing a convolution operation on the image. Gaussian filtering is widely used in image processing, particularly in the preprocessing stage.

- **How it works**: Gaussian filtering uses a convolution kernel (or weight matrix) based on a Gaussian distribution to perform a weighted average of each pixel in an image and its surrounding pixels. The center of the kernel has the highest weight, and the weight gradually decreases with increasing distance from the center (in accordance with a normal distribution). This weighted average results in smoother pixel value variations, blurring details, and suppressing noise.
- **Effect**: Gaussian filtering effectively removes high-frequency noise (such as random noise) from an image while preserving low-frequency information (such as overall structure and large edges). Compared to mean filtering, Gaussian filtering is more natural because it is closer to human visual perception and does not introduce noticeable blocky artifacts.
- **Applications**: Gaussian filtering is commonly used for image denoising, preprocessing before edge detection (to reduce noise interference on edge detection), image downsampling (to prevent aliasing), and special effects (such as background blur).

In Gaussian filtering, the size of the convolution kernel (which must be an odd number) significantly affects the result: larger kernels result in stronger blurring, but also increase the computational effort. In addition, the parameters of the Gaussian filter may also include the standard deviation (sigma), which is used to control the weight distribution of the kernel (not explicitly specified in this code, it may be a fixed value or handled internally by the library).

Code Overview

Importing Modules

```
import time, os, sys, gc
from machine import Pin
from media.sensor import * # Camera interface
from media.display import * # Display interface
from media.media import * # Media manager
import _thread
import cv_lite # cv_lite extension (includes the gaussian_blur API)
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(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) # Set pixel format to 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 Gaussian filtering of color images).

Initialize the display module

```
Display.init(Display.ST7701, width=image_shape[1], height=image_shape[0],
to_ide=True, quality=100)
```

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, and `quality=100` sets the image transmission quality to the highest.

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 Gaussian filter parameters

```
kernel_size = 21
clock = time.clock() # Start FPS timer
```

- `kernel_size`: The convolution kernel size of the Gaussian filter. This must be an odd number (such as 3, 5, or 7). Larger values produce stronger blurring but increase the computational effort. In this code, it is set to 21, which is a larger kernel and produces a noticeable smoothing effect.
- `clock`: Used to calculate the frame rate.

Image processing and Gaussian filtering

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

    # Capture a frame
    img = sensor.snapshot()
    img_np = img.to_numpy_ref()  # Get RGB888 ndarray (HWC)

    # Apply Gaussian blur using cv_lite
    denoised_np = cv_lite.rgb888_gaussian_blur_fast(
        image_shape,
        img_np,
        kernel_size
    )

    # Wrap processed image for display
    img_out = image.Image(image_shape[1], image_shape[0], image.RGB888,
                          alloc=image.ALLOC_REF, data=denoised_np)

    # Display image Display the blurred image
    Display.show_image(img_out)
```

```
    # Clear memory and print the frame rate / Collect garbage and show FPS
    gc.collect()
    print("Gaussian blur fps:", clock.fps())
```

- **Image capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Gaussian filtering**: Call `cv_lite.rgb888_gaussian_blur_fast()` to perform the Gaussian filtering operation, returning the processed image as a NumPy array.
- **Image packaging and display**: Package the processed result into an RGB888 format image object and display it on the screen or in an IDE virtual window.
- **Memory management and frame rate output**: Call `gc.collect()` to clear 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`: The convolution kernel size of the Gaussian filter. A larger value results in a stronger blurring effect, making it suitable for processing higher levels of noise or for smoother images. An odd number (such as 3, 5, or 7) must be used. It is recommended to start with 3 or 5 and gradually increase to a larger value such as 21 to observe the effect and performance impact. A larger kernel (such as 21 in this code) will significantly increase the amount of computation and may reduce the frame rate. Adjustments should be made based on hardware performance.