

XÂY DỰNG PHẦN MỀM CHUYỂN ẢNH THÀNH TRANH VẼ

Môn: Xử lý ảnh số

Đề tài số 4

Sinh viên thực hiện:

Bùi Thiên Thái-B22DCCN776

Đặng Anh Tuấn-B22DCCN752

Giảng viên hướng dẫn: Phạm Hoàng Việt

Nhóm 12- Nhóm lớp 12

Đề tài & Yêu cầu đề bài

Nội dung đề tài 4:

- Xây dựng phần mềm chuyển ảnh thành tranh vẽ (vẽ tay).
- Sử dụng:
 - Chuyển đổi mức xám.
 - Kỹ thuật làm mịn ảnh.
 - Phát hiện biên.

Yêu cầu nghiêm thu:

- Cho phép tải ảnh từ file (ảnh y tế, tự nhiên, công nghiệp...).
- Xử lý ảnh và xem kết quả trực quan.
- Có thể lưu lại ảnh kết quả.
- Khuyến khích dùng Bilateral hoặc bộ lọc edge-preserving.

Mục tiêu & Phân chia công việc

Mục tiêu kỹ thuật:

- Xây dựng ứng dụng desktop bằng Tkinter.
- Tự cài đặt các thuật toán: Gaussian, Bilateral, Sobel, LoG, Canny.
- Tạo được 3 hiệu ứng:
 - Cartoon grayscale.
 - Pencil Sketch đen trắng.
 - Pencil Sketch màu.

Phân chia công việc:

Đặng Anh Tuấn

- Thiết kế giao diện GUI (Tkinter).
- Tổ chức module, app.py, gui.py.
- Pipeline Sketch (BW & Color).

Bùi Thiên Thái

- Cài đặt Gaussian separable, Median, Bilateral (filters.py).
- Cài đặt Sobel, LoG, Canny (edges.py).
- Cartoon grayscale (painting_pipeline).

• Tỉ lệ đóng góp: 50% – 50%.

Bài toán & Hướng tiếp cận

Đầu vào:

- Ảnh màu bất kỳ (JPG, PNG, BMP...)

Đầu ra:

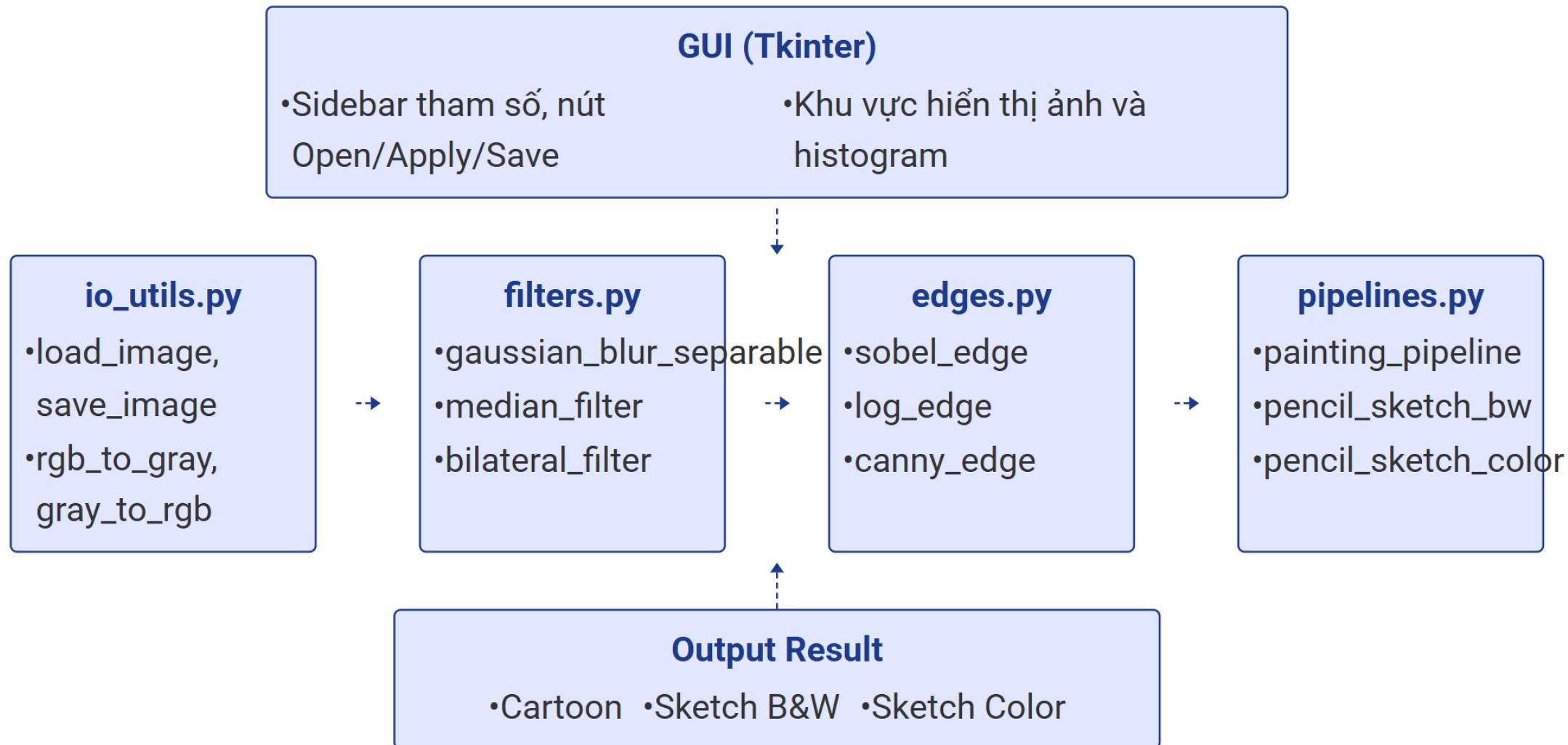
- Ảnh dạng tranh vẽ:

- Cartoon (mức xám phẳng + đường biên)
- Sketch đen trắng, Sketch màu

Hướng tiếp cận tổng quan:

- RGB → ảnh xám (Gray)
- Làm mịn để giảm nhiễu (Gaussian / Bilateral)
- Phát hiện biên (Sobel / LoG / Canny)
- Xử lý sau biên:
 - Biến bản đồ biên thành mask
 - Kết hợp với ảnh xám/quantization → tạo hiệu ứng tranh
- Hiển thị & lưu kết quả

Kiến trúc phần mềm (Block Diagram)



Tổ chức mã nguồn

Các file chính:

- app.py – Điểm vào, khởi tạo Tkinter.
- gui.py – Lớp ImageSketchApp: UI, sự kiện, preview.
- io_utils.py – Đọc/ghi ảnh, chuyển đổi màu.
- filters.py – Gaussian, Median, Bilateral.
- edges.py – Sobel, LoG, Canny.
- pipelines.py – Cartoon & Sketch pipelines.
- __init__.py – xuất ImageSketchApp.

Ưu điểm:

- Mỗi file phụ trách một nhóm chức năng.
- Dễ bảo trì, dễ thuyết trình chi tiết từng phần.

Chuyển đổi mức xám (RGB → Gray)

Lý do cần Gray:

- Giảm 3 kênh xuống 1 kênh → giảm tính toán
- Các thuật toán biên (Sobel, LoG, Canny) hoạt động tốt trên ảnh xám

Công thức sử dụng:

$$\text{Gray} = 0.299R + 0.587G + 0.114B$$

Ứng dụng:

- Tất cả mode Cartoon, Sketch đều bắt đầu từ Gray
- Gray cũng dùng để tính histogram

Tổng quan về Smoothing (Làm mịn ảnh)

Mục tiêu:

- Giảm nhiễu, tránh biên giả (false edges)
- Làm mịn nhẹ trước khi phát hiện biên / quantization

Các bộ lọc trong dự án:

- Gaussian blur (separable)
- Bilateral filter (edge-preserving)

Vai trò trong pipeline:

- Trước Canny, LoG
- Trước Cartoon (làm mịn để vùng phẳng đẹp hơn)

Gaussian Blur (Separable) – Chi tiết

Ý tưởng separable:

- Convolution 2D = convolution 1D theo hàng + 1D theo cột.

Lợi ích:

- Giảm độ phức tạp: từ $O(K^2)$ xuống $O(2K)$.
- Tăng tốc xử lý ảnh lớn.

Trong code:

- Hàm gaussian_blur_separable(gray, sigma, ksize)
- Dùng ở:
 - Smoothing cho Canny
 - Tiền xử lý cho LoG
 - Chuẩn bị ảnh cartoon/ sketch

Bilateral Filter – Lý thuyết & Cài đặt

Ý tưởng:

- Kết hợp:
 - Gaussian theo khoảng cách (spatial)
 - Gaussian theo chênh lệch cường độ (range)
- Pixel gần & giống nhau → ảnh hưởng mạnh

Tính chất:

- Làm mịn vùng phẳng
- Giữ rõ biên – rất phù hợp để tài "tranh vẽ"

Tối ưu trong code:

- Giới hạn kích thước cửa sổ (`window_size ≤ 11`)
- Nếu ảnh lớn → fallback Gaussian
- Cache Gaussian không gian để không tính lại

Phát hiện biên: Sobel

Nguyên lý:

- Dùng 2 mask Sobel:
 - G_x theo chiều ngang (phát hiện biên dọc)
 - G_y theo chiều dọc (phát hiện biên ngang)

Độ lớn gradient:

- Công thức: $mag = \sqrt{G_x^2 + G_y^2}$
 - Cách tính nhanh (đôi khi dùng): $mag = |G_x| + |G_y|$
 - Xác định hướng gradient: $\theta = \arctan(G_y/G_x)$

Cách dùng trong code:

- Áp trên ảnh xám đã làm mịn:
 - `edges = sobel_edge(smoothed_img)`
- So sánh với ngưỡng threshold để lấy biên:
 - `binary_edges = edges > threshold`
- Trong dự án: thường dùng cho Cartoon

Phát hiện biên: LoG (Laplacian of Gaussian)

Pipeline:

- Gaussian smoothing (sigma do người dùng chọn)
- Áp dụng Laplacian 3x3
- Tìm zero-crossing (vị trí đổi dấu)

Tối ưu:

- Gaussian separable → nhanh
- Kernel Laplacian nhỏ → tránh lag

Đặc điểm biên:

- Biên mượt, ít nhiễu hơn Sobel đơn thuần

Phát hiện biên: Canny (Full Pipeline)

1 Gaussian Smoothing

Làm mịn ảnh để loại bỏ nhiễu trước khi phát hiện biên

2 Tính Gradient

Sử dụng bộ lọc Sobel theo hướng X và Y để tính cường độ và hướng gradient

3 NMS

Non-Maximum Suppression làm mỏng biên, chỉ giữ lại pixel cường độ lớn nhất theo hướng gradient

4 Double Threshold

Phân loại pixel biên thành mạnh, yếu và loại bỏ dựa trên 2 ngưỡng cao và thấp

5 Hysteresis

Nối các đoạn biên yếu có liên quan tới biên mạnh, loại bỏ các biên yếu cô lập

Tham số trong GUI:

- Sigma: kiểm soát mức độ làm mịn của Gaussian
- Low threshold ratio: ngưỡng dưới (phần trăm so với ngưỡng trên)
- High threshold ratio: ngưỡng trên (phần trăm cường độ tối đa)

Kết quả:

- Biên mạnh, sắc nét (nhờ NMS)
- Ít nhiễu nhờ double threshold và hysteresis
- Phù hợp cho Sketch & Cartoon pipeline

Xử lý sau khi phát hiện biên (Post-Edge Processing) – Cartoon

1 Chuẩn hóa ảnh biên

Biến đổi mảng biên về giá trị từ 0.0 đến 1.0

$$e = \text{edges} / 255.0$$

Tại vị trí biên: $e \approx 1.0$

Tại vị trí không phải biên: $e \approx 0.0$

2 Tạo Edge Mask bằng Invert

Đảo ngược giá trị biên để tạo mask

$$\text{mask} = 1.0 - e$$

Tại vị trí biên: $\text{mask} \approx 0.0 \rightarrow$ tối (nét bút)

Tại vị trí không phải biên: $\text{mask} \approx 1.0 \rightarrow$ giữ sáng/phẳng

3 Áp mask lên ảnh quantized

Nhân ảnh xám đã lượng hóa (phân tầng) với mask

$$\text{cartoon} = \text{grayquantized} \times \text{mask} \times 255$$

Biên: nhân với mask gần 0 → tạo nét vẽ tối

Không phải biên: nhân với mask gần 1 → giữ giá trị lượng hóa

Tâm quan trọng trong đề tài:

- Đây là phần quan trọng mà ít đề tài làm rõ
- Mục đích: nhúng nét vẽ (edge) lên vùng phẳng (quantized)
- Tạo hiệu ứng nét vẽ trên các vùng màu đồng nhất

Kết quả cartoon grayscale:

- Biên rõ ràng, mảnh như nét vẽ bút chì
- Vùng bên trong phẳng (flat) do quantization
- Hiệu ứng giống tranh vẽ tay dạng poster/outline

Cartoon Grayscale Pipeline (Theo code)

1 Gray + Smoothing

Chuyển ảnh màu RGB sang ảnh xám và làm mịn để giảm nhiễu

```
gray = rgb_to_gray(img)  
smoothed = gaussian / bilateral
```



2 Edge Detection

Chọn 1 trong 3 thuật toán phát hiện biên: Sobel / LoG / Canny

```
edges = detect_edges(smoothed)
```



3 Quantization mức xám

Chia 256 mức xám thành số ít mức (3-12) để tạo hiệu ứng phẳng

```
bins = np.linspace(0, 255, quant_levels+1)  
quantized = np.digitize(gray, bins) * (255/quant_levels)
```



4 Mask theo biên (Post-edge)

Tạo mask từ biên và áp dụng lên ảnh quantized

```
edge_norm = edges / 255.0  
mask = 1.0 - edge_norm  
cartoon = quantized * mask
```



5 Hiển thị

Chuẩn hóa kết quả để hiển thị và lưu ảnh

```
result = gray_to_rgb(cartoon) (để hiển thị/lưu)
```

👉 *Cartoon grayscale = vùng phẳng mức xám + đường biên tối*

Pencil Sketch B&W Pipeline

① Gray

Chuyển ảnh màu RGB sang ảnh xám

```
gray = rgb_to_gray(img)
```



② Invert

Đảo ngược ảnh xám để chuẩn bị cho hiệu ứng sketch

```
inv = 255 - gray
```



③ Blur inv

Làm mờ ảnh đã đảo ngược bằng Gaussian blur

```
blur = gaussian_blur(inv, sigma=sigma)
```



④ Dodge blend

Kết hợp ảnh xám và ảnh làm mờ theo công thức dodge

Công thức: $\text{sketch} = \text{gray} \times 255 / (255 - \text{blur} + \epsilon)$

```
sketch = np.minimum(gray * 255 / (255 - blur + 1e-7), 255)
```

Kết quả: Hiệu ứng như vẽ chì trên nền giấy trắng, vùng bóng tự nhiên

Pencil Sketch Color Pipeline

1 Tạo sketch B&W

Tạo bản sketch đen trắng làm nền tảng như đã trình bày

```
gray = rgb_to_gray(img)  
inv = 255 - gray  
blur = gaussian_blur(inv, sigma=blur_strength)  
sketch_bw = dodge_blend(gray, blur)
```



2 Chuẩn hóa

Chuẩn hóa giá trị RGB và sketch về khoảng [0,1]

```
rgb_norm = img / 255.0  
sketch_norm = sketch_bw / 255.0
```



3 Nhân để giữ màu gốc

Nhân RGB với sketch để giữ màu gốc kết hợp với nét chì

```
result = rgb_norm * sketch_norm  
result = result * 255.0 (chuyển về thang 0-255)
```

Kết quả: ảnh giữ được tông màu gốc nhưng có texture nét chì tự nhiên

Histogram & Đánh giá mức xám

Tính bằng NumPy:

- `hist, _ = np.histogram(gray_img, bins=256, range=(0,256))`
 - Tính toán nhanh và hiệu quả hơn vòng lặp Python
 - 256 mức xám, phạm vi từ 0-255 (8 bit)

Hiển thị:

- Histogram ảnh gốc
- Histogram ảnh sau cartoon/sketch
 - So sánh trực quan trước/sau xử lý

Nhận xét ví dụ:

- Cartoon → các cột "bậc thang" rõ
 - Do quantization (giảm số mức xám)
- Bilateral → histogram mượt hơn ảnh gốc
 - Giảm nhiễu, giữ rõ cạnh quan trọng
- Sketch → histogram lệch về vùng sáng
 - Nền trắng (phông giấy) chiếm ưu thế
 - Đúng với đặc trưng của tranh vẽ chì thật

Tối ưu hiệu năng & Tránh lag UI

Vấn đề:

- Bilateral, Canny, LoG trên ảnh lớn dễ gây lag

Giải pháp trong code:

- Xử lý ảnh trong thread riêng → UI không "Not Responding"
- Giới hạn window_size Bilateral
- Dùng Gaussian separable
- Dùng np.histogram thay vì vòng lặp Python

Kết quả:

- Kéo slider, đổi tham số → phản hồi nhanh
- Ảnh Full HD xử lý vẫn sử dụng được

Kết quả thử nghiệm & Đánh giá

Trên ảnh y tế:

- Canny + Bilateral cho biên xương, cấu trúc rõ.
- Mạch máu và chi tiết mô mềm được phân biệt rõ ràng.

Trên ảnh công nghiệp:

- Sobel/LoG giúp thấy rõ cạnh chi tiết.
- Các chi tiết nhỏ, kết cấu kim loại và bề mặt vật liệu được thể hiện tốt.

Trên ảnh tự nhiên / chân dung:

- Sketch B&W và Color cho hiệu ứng vẽ tay đẹp.
- Cartoon grayscale làm nổi bật shape & biên.
- Đặc biệt hiệu quả với ảnh chân dung và phong cảnh có chi tiết đa dạng.

Đánh giá chung:

- | | |
|--|--|
| • Đáp ứng tốt yêu cầu đề tài. | • Kết quả trực quan, dễ thuyết phục. |
| • Xử lý được đa dạng loại ảnh đầu vào. | • Giao diện người dùng thân thiện, dễ sử dụng. |