

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Лабораторная работа №2

По дисциплине «Алгоритмы компьютерной графики»

Выполнил:

Студент группы Р3306

Михайлов Дмитрий

Андреевич

Преподаватель:

Потемин Игорь

Станиславович



Санкт-Петербург

2025 год

Оглавление

Общее описание приложения	2
Архитектура приложения	3
Визуализация данных	4
Технические особенности	9
Пользовательский интерфейс	10
Заключение	10

Общее описание приложения

Приложение представляет собой настольное GUI-приложение для применения различных фильтров к изображениям.

Используемые библиотеки:

- **Tkinter** — для построения графического интерфейса;
- **Pillow (PIL)** — для загрузки, обработки и сохранения изображений.

Функционал приложения:

- загрузка изображения с диска или автоматическая загрузка файла `test.png` из папки с программой;
- отображение исходного и обработанного изображения;
- применение фильтров:
 - Grey (оттенки серого),
 - Blur (размытие),
 - Contrast (контраст),
 - Brightness (яркость),
 - Invert Colors (инверсия цветов);
- сохранение результата обработки в формате PNG.

Архитектура приложения

1. Класс ImageEditorApp — основной класс приложения, в котором:

- инициализируются элементы интерфейса;
- хранятся ссылки на исходное и обработанное изображение;
- реализованы методы обработки и вспомогательные функции.

2. Интерфейс управления (панель сверху) содержит:

- кнопки:
 - Grey,
 - Blur,
 - Contrast,
 - Brightness,
 - Invert Colors,
 - Open Image,
 - Save PNG;
- ползунки (слайдеры):
 - Radius — для фильтра размытия;
 - Factor — для контраста;
 - Bright — для яркости.

3. Область отображения:

- левая часть — метка **Исходное**, показывает уменьшенную копию загруженного изображения;
- правая часть — метка **Результат**, показывает результат последнего применённого фильтра либо заглушку до обработки.

4. Методы обработки:

- `make_gray` — перевод в оттенки серого;
- `apply_blur` — размытие;
- `apply_contrast` — изменение контраста;
- `apply_brightness` — изменение яркости;
- `invert_colors` — инверсия цветов;
- `save_image` — сохранение результата.

5. Вспомогательные методы:

- `_load_default_image()` — попытка загрузить `test.jpg` при старте;
- `open_image_dialog()` — диалог выбора файла;
- `_load_image(path)` — безопасная загрузка изображения и подготовка превью;
- `_preview_size(image) / _resize_for_preview(image)` — расчёт размеров и масштабирование под окно;
- `_show_left(img), _show_right(img)` — обновление левой и правой картинок в интерфейсе;
- `_ensure_image_loaded()` — проверка, что изображение загружено перед обработкой;
- `_update_result(img)` — сохранение результата и обновление правого превью.

Подробное описание фильтров

1. Фильтр “Grey” (Оттенки серого)

Принцип работы. Цель фильтра — преобразовать цветное изображение в оттенки серого. В простейшем случае яркость пикселя можно вычислить как среднее от каналов R, G и B:

$$gray = \frac{R + G + B}{3}.$$

В приложении используется готовая функция `ImageOps.grayscale`, которая переводит изображение в одноканальный режим L, а затем результат приводится обратно к RGBA с учётом альфа-канала исходного изображения.

Пример кода:

```
rgb = self.orig_image.convert("RGB")
gray_l = ImageOps.grayscale(rgb)
gray_rgba = gray_l.convert("RGBA")

if self.orig_image.mode == "RGBA":
    alpha = self.orig_image.getchannel("A")
    gray_rgba.putalpha(alpha)

self._update_result(gray_rgba)
```

Особенности:

- при наличии прозрачности (формат RGBA) альфа-канал сохраняется;
- результат — изображение в оттенках серого с тем же уровнем прозрачности.

Результат. Цветное изображение превращается в чёрно-белое, при этом прозрачные области остаются прозрачными.

2. Фильтр “Blur” (Размытие)

Принцип работы. Размытие реализовано с помощью `ImageFilter.GaussianBlur` из Pillow. Гауссово размытие смешивает каждый пиксель с его соседями, используя гауссово распределение: чем ближе соседние пиксели, тем больший вклад они вносят.

Параметры:

- радиус размытия задаётся ползунком `Radius` (0–20);
- чем больше радиус, тем сильнее размытие.

Пример кода:

```
radius = int(self.radius_scale.get())
blurred = self.orig_image.filter(ImageFilter.GaussianBlur(radius))
self._update_result(blurred)
```

Результат. Изображение становится более «мягким», мелкие детали сглаживаются, текст и границы становятся менее резкими.

3. Фильтр “Contrast” (Контрастность)

Принцип работы. Контраст — это разница между тёмными и светлыми участками изображения. Фильтр регулирует эту разницу, делая изображение либо более «плоским», либо более «выразительным».

Общее математическое представление:

$$\text{новое_значение} = (\text{старое_значение} - \text{среднее}) \times \text{фактор} + \text{среднее},$$

где:

- *среднее* — некая средняя яркость;
- *фактор* — множитель контраста (в приложении от 0.0 до 3.0).

Техническая реализация:

- используется класс `ImageEnhance.Contrast` из Pillow;
- обрабатываются только RGB-каналы;
- альфа-канал извлекается отдельно и затем объединяется с обработанными каналами.

Пример кода:

```
factor = float(self.contrast_scale.get())

rgba = self.orig_image.convert("RGBA")
rgb = rgba.convert("RGB")
alpha = rgba.getchannel("A")

enhanced_rgb = ImageEnhance.Contrast(rgb).enhance(factor)
r, g, b = enhanced_rgb.split()
result = Image.merge("RGBA", (r, g, b, alpha))

self._update_result(result)
```

Влияние параметров:

- `factor = 0.0` — изображение становится полностью серым (всё стягивается к средней яркости);
- `factor = 1.0` — исходный контраст без изменений;
- `factor > 1.0` — контраст усиливается (тёмные участки темнеют, светлые светлеют);
- `factor < 1.0` — контраст уменьшается, изображение становится более «блеклым».

Результат. Регулируется общая выразительность изображения: можно сделать его более ярким и драматичным или, наоборот, приглушённым.

4. Фильтр “Brightness” (Яркость)

Принцип работы. Фильтр изменяет общую яркость изображения, делая его темнее или светлее.

Техническая реализация:

- используется класс `ImageEnhance.Brightness`;
- аналогично контрасту, RGB и альфа-канал обрабатываются отдельно.

Пример кода:

```
factor = float(self.brightness_scale.get())

rgba = self.orig_image.convert("RGBA")
rgb = rgba.convert("RGB")
alpha = rgba.getchannel("A")

enhanced_rgb = ImageEnhance.Brightness(rgb).enhance(factor)
r, g, b = enhanced_rgb.split()
result = Image.merge("RGBA", (r, g, b, alpha))

self._update_result(result)
```

Интерпретация значений фактора:

- `factor = 0.0` — изображение становится полностью чёрным;
- `factor = 1.0` — исходная яркость без изменений;
- `factor > 1.0` — изображение осветляется;
- `factor < 1.0` — изображение затемняется.

Результат. Изменяется общая освещённость изображения, что полезно для коррекции тёмных или пересвеченных фотографий.

5. Фильтр “Invert Colors” (Инверсия цветов)

Принцип работы. При инверсии каждый цвет заменяется на «противоположный» по формуле:

$$\text{новый_канал} = 255 - \text{старый_канал}.$$

Например:

- чёрный (0, 0, 0) превращается в белый (255, 255, 255);
- белый превращается в чёрный;
- красный (255, 0, 0) превращается в голубой (0, 255, 255) и т. д.

Реализация в коде. Вместо ручного прохода по пикселям используется функция `ImageOps.invert` для RGB-части изображения, при этом альфа-канал сохраняется:

```
rgba = self.orig_image.convert("RGBA")
rgb = rgba.convert("RGB")
alpha = rgba.getchannel("A")

inverted_rgb = ImageOps.invert(rgb)
r, g, b = inverted_rgb.split()
result = Image.merge("RGBA", (r, g, b, alpha))

self._update_result(result)
```

Особенности:

- прозрачность не меняется (альфа-канал не инвертируется);
- визуально эффект напоминает «фото-негатив».

Результат. Получаем негатив исходного изображения с сохранением прозрачных областей.

Технические особенности

Работа с альфа-каналом

Во всех фильтрах учитывается наличие прозрачности:

- исходное изображение приводится к режиму **RGBA**;
- альфа-канал (`getchannel("A")`) сохраняется отдельно;
- обработка (контраст, яркость, инверсия, серый) применяется только к цветовым каналам (RGB);
- в финальном результате каналы R, G, B объединяются с исходным альфа-каналом:

```
result = Image.merge("RGBA", (r, g, b, alpha))
```

Это позволяет корректно обрабатывать логотипы, иконки и другие изображения с прозрачным фоном.

Масштабирование изображения

Для корректного отображения изображений в окне используется система превью:

- максимальные размеры превью:
 - `PREVIEW_MAX_WIDTH = 500`,
 - `PREVIEW_MAX_HEIGHT = 400`;
- реальный размер превью вычисляется пропорционально:

```
ratio = min(  
    PREVIEW_MAX_WIDTH / w,  
    PREVIEW_MAX_HEIGHT / h,  
    1.0,  
)  
new_size = (int(w * ratio), int(h * ratio))
```

Масштабирование выполняется с использованием высококачественного ресэмплинга:

```
RESAMPLING = Image.Resampling.LANCZOS  
image.resize(new_size, RESAMPLING)
```

Обработка фильтрами всегда происходит над полной версией изображения, а не над уменьшенной копией.

Загрузка и сохранение

- при запуске приложение пытается загрузить файл `test.jpg` из той же папки, что и скрипт; при отсутствии файла пользователю показывается сообщение об ошибке;
- кнопка `Open Image` вызывает диалог выбора файла с фильтрами по типам изображений;
- кнопка `Save PNG`:
 - сохраняет обработанное изображение;
 - использует диалог `asksaveasfilename` с расширением `.png` по умолчанию;
 - выводит сообщение об успехе или ошибке.

Пользовательский интерфейс

1. Верхняя панель управления:

- 1-й ряд:
 - кнопка **Grey**;
 - кнопка **Blur**;
 - подпись **Radius:** и ползунок радиуса размытия;
- 2-й ряд:
 - кнопка **Contrast**;
 - подпись **Factor:** и ползунок контраста;
 - кнопка **Brightness**;
 - подпись **Bright:** и ползунок яркости;
- 3-й ряд:
 - кнопка **Open Image**;
 - кнопка **Invert Colors**;
 - кнопка **Save PNG**.

2. Область изображений:

- слева — подпись **Исходное** и превью оригинала;
- справа — подпись **Результат** и превью обработанного изображения или заглушка до применения фильтра.

1. При запуске:

- либо автоматически загружается **test.png**;
- либо пользователь вручную выбирает изображение через **Open Image**.

2. Пользователь настраивает параметры фильтра (радиус, контраст, яркость) с помощью ползунков.

3. Нажимает соответствующую кнопку фильтра (**Grey**, **Blur**, **Contrast**, **Brightness**, **Invert Colors**).

4. Результат отображается справа.

5. После получения нужного эффекта можно сохранить его с помощью **Save PNG**.

Заключение

Разработанное приложение демонстрирует:

- практическое использование библиотеки **Tkinter** для создания простого графического интерфейса;
- применение возможностей **Pillow** для обработки изображений:
 - использование встроенных фильтров (**GaussianBlur**, **ImageEnhance**);
 - работу с цветовыми каналами и альфа-каналом;
 - масштабирование изображений с сохранением пропорций.

Приложение позволяет наглядно экспериментировать с параметрами фильтров (радиус размытия, контраст, яркость) и сразу видеть результат. Благодаря тому, что обработка всегда выполняется от исходного изображения, пользователь может многократно пробовать разные настройки, не накапливая артефакты предыдущих преобразований.