



Софийски университет „Св. Климент Охридски“
Факултет по математика и информатика

*Курсова работа по Обектно-ориентирано програмиране
специалност Компютърни науки*

Тема №17 Растерна графика

(Конзолно C++ приложение)

Съдържание

| | |
|---|----|
| 1. Увод..... | 3 |
| 1.1. Описание и идея на проекта | |
| 1.2. Цели и задачи на разработката | |
| 1.3. Връзка към хранилището в Github | |
| 2. Преглед на предметната област..... | 4 |
| 2.1. Основни дефиниции и концепции, използвани в проекта | |
| 2.2. Алгоритми | |
| 2.3. Подходи и методи за решаване на поставените проблеми | |
| 3. Проектиране..... | 7 |
| 3.1. Архитектура | |
| 3.2. Имплементация (най-важните извадки от кода) | |
| 4. Реализация и тестване..... | 10 |
| 4.1. Управление на паметта и сложност | |
| 4.2. Тестове | |
| 5. Заключение..... | 11 |
| 5.1. Бъдещо развитие и усъвършенстване | |

1. Увод

1.1. Описание и идея на проекта

Проектът „Растерна графика“ реализира картинен редактор, в който потребителят може да редактор трябва да може да прилага различни трансформации върху изображенията. Идеята е да се разработи приложение, което представлява конзолен редактор на растерни изображения, поддържа работа с различни файлове(от тип PPM, PGM и PBM).

Потребителят може да създава множество сесии, във всяка от които отваря изображения и задава операции – да направи всички снимки в сесията негатив (*negative*), едноцветни (*monochrome*) или в нюанси на сивото (*grayscale*). При въвеждане на командата *save* модифицираните изображения в сесията се запазват в съответните файлове. Съществува и опция за правене на колаж (*collage*), при което се подреждат две изображения от един и същи формат едно до друго (*horizontal*) или едно под друго (*vertical*) и ги записва в един файл. Когато потребителят иска да отмени последната зададена , това може да стане

1.2.Цели и задачи на разработката

Целта на проекта е да се създаде оптимална програма с максимално разнообразна функционалност, която да отговаря на поставената задача.

Проектът е изграден съгласно добрите принципи на ООП.

1.3. Връзка към хранилището в Github:

https://github.com/bobivk/Photoshop_netbpm

2. Преглед на предметната област

2.1. Основни дефиниции и концепции, използвани в проекта

Netbpm представлява три формата за запазване на изображения:

PBM (Portable BitMap) формат: всеки пиксел може да приема стойност 0 или 1, където 0 означава бял пиксел, а 1 - черен пиксел, за разлика от останалите формати, където по-високите стойности представляват по-светли пиксели.

PGM (Portable GrayMap) формат: пикселите приемат стойности цели числа, като преди матрицата с пиксели се изписва число, което играе ролята на максимална стойност – чисто черно – на пикселите във файла. Така нюансът на сиво на всеки пиксел се скалира според разликата на неговата стойност с максималната.

PPM (Portable PixMap) формат: всеки пиксел се състои от три стойности (цели числа), които отговарят съответно за червено, синьо и зелено (RGB). Както и при PGM преди матрицата се изписва максималното число, което може да заема всяка стойност.

В началото на файла се изписва т.нар. „магическо число“, което означава типа на файла:

| Формат | Магическо число | |
|--------|-----------------|---------------|
| | ASCII | Двоичен запис |
| PPM | P1 | P4 |
| PGM | P2 | P5 |
| PPM | P3 | P6 |

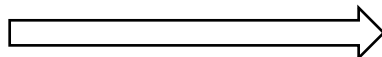
Примери:

P1

6 10

0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0

Представява изображението



```

P2
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 0 0 0 11 11 11 0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Представява изображението

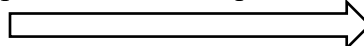


```

P3
3 2
255
255 0 0 0 255 0 0 0 255
255 255 0 255 255 255 0 0 0

```

Представява изображението



2.2. Алгоритми

Negative: заменяме всеки пиксел от картинката с обратния му такъв.

-PBM: пикселите със стойност 0 приемат стойност 1 и тези със стойност 1 приемат стойност 0

-PGM: всеки пиксел заема стойност, равна на максималната стойност на пиксел в картинката минус началната му стойност

-PPM: всяка стойност за основен цвят заема стойност, равна на максималната стойност за картинката минус началната му стойност.

Monochrome: правим картинката едноцветна – ако стойността на пиксела е по-малка от половината на максималната стойност, я зануляваме, а ако е по-голяма от половината, я правим равна на максималната стойност. Така получаваме картинка, състояща се само от нули или максимални стойности.

Grayscale: операция, приложима само върху изображения в PPM формат. За всеки пиксел правим средноаритметично на RGB стойностите му и след това правим всяка от стойностите равна на това средноаритметично.

Колаж: запазваме две снимки последователно в един файл или една до друга, или една под друга. Размерите на изходната снимка са както следва:

- Вертикално:

$$\begin{array}{ccc} (x_1, y_1) & \longrightarrow & (\max(x_1, x_2), y_1 + y_2) \\ (x_2, y_2) & & \end{array}$$

- Хоризонтално:

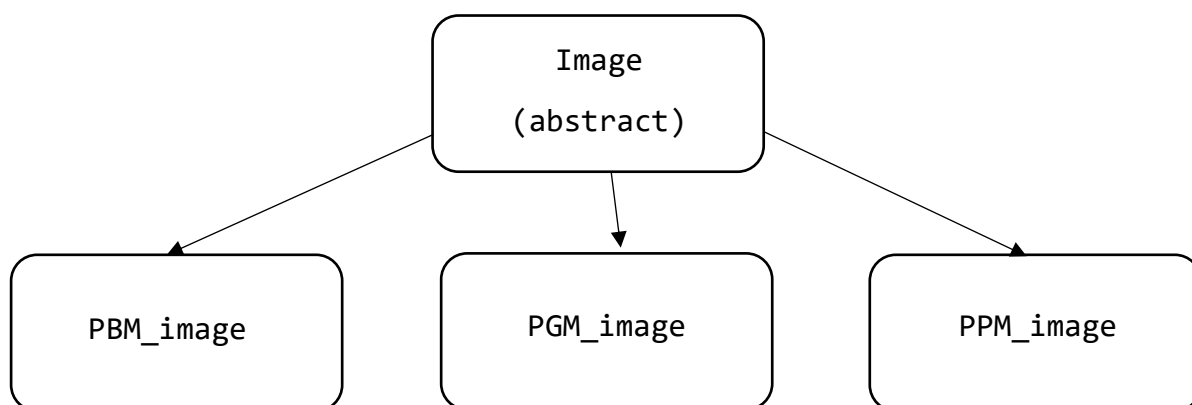
$$\begin{array}{ccc} (x_1, y_1) & \longrightarrow & (x_1 + x_2, \max(y_1, y_2)) \\ (x_2, y_2) & & \end{array}$$

2.3. Подходи и методи за решаване на поставените проблеми

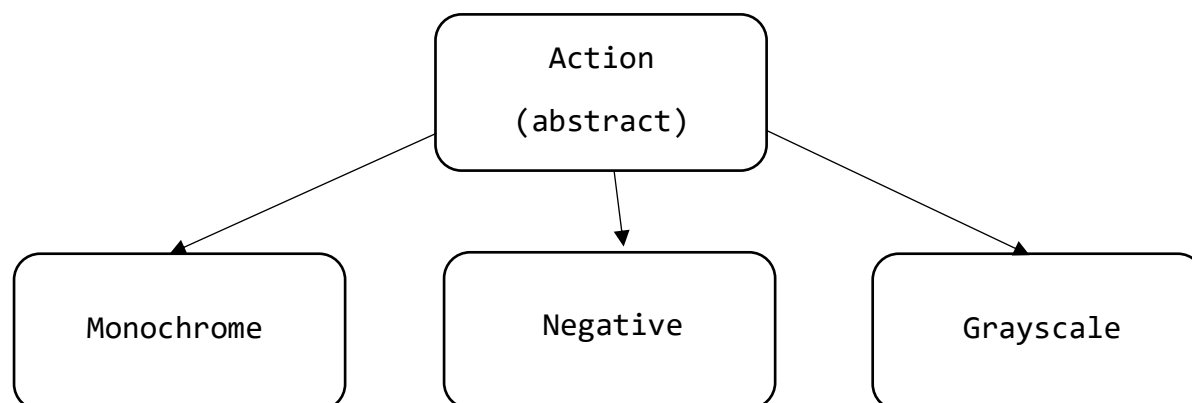
На страницата <http://netpbm.sourceforge.net/> са описани подробно трите формата, както и историята за появата им и функцията им. Преди започване на работа се запознах с тази документация, използвайки някои от техните примери за тестване на програмата си.

3. Проектиране

3.1. Архитектура и реализация на класовете



Фигура 3.1.1



Фигура 3.1.2

Освен тези йерархии са изградени и следните класове:

- PBM_pixel
- PGM_pixel
- PPM_pixel
- Dimensions
- Collage
- Session

3.2. Имплементация (най-важните извадки от кода)

Зареждане на снимки: след задаване на името на файла, който трябва да бъде зареден, програмата го отваря и разчита „магическото число“. След това се конструира обект от правилния тип изображение чрез неговия конструктор. Конструкторът чете размерите на подаденото изображение и ги запазва като обект от тип **Dimensions**, който е член-данна на изображението. Освен размер, всяко изображение съдържа двумерен

`std::vector` от съответния тип пиксели. Всеки от тях бива прочетен последователно от файла чрез предефиниран `operator>>`(фиг. 3.2.2) по следния начин:

```
//there may be fill bits at the end, so we need to read only dimensions.y characters on each row
for (unsigned row = 0; row < dimensions.y; ++row) {
    vector<PBM_pixel> row_pixels;
    while (in.peek() == '#') in.ignore(2048, '\n');
    for (unsigned col = 0; col < dimensions.x; ++col) {
        try {
            in >> current_pixel;
            row_pixels.push_back(current_pixel);
        }
        catch (Bad_pixel_exception & e) {
            cout << "Bad PBM_pixel exception caught: " << e.get_bad_pixel() << endl;
        }
    }
    input_pixels.push_back(row_pixels);
    cout << "\n";
}
set_pixel_matrix(input_pixels);
```

Фигура 3.2.1

```
istream& operator>>(istream& in, PBM_pixel& pixel) {
    char input_char;
    in.get(input_char);
    while (in.peek() == '#') in.ignore(2048, '\n');
    while (input_char == ' ' || input_char == '\n') in.get(input_char);
    if (input_char == '0' || input_char == '1') {
        pixel.value = input_char - '0';
    }
    else
    {
        string message;
        message += input_char;
        throw Bad_pixel_exception(message);
    }
    return in;
}
```

Фигура 3.2.2

Класът `Session` се грижи за всички изображения и действия в съответната сесия и притежава следните член-данни в секцията със спецификатор за достъп `private`:

```
std::vector<Image*> images;  
std::vector<Action*> actions;
```

Всяка сесия също така притежава уникален идентификационен номер – `id`.

Структурата `Dimensions` се грижи за правилната работа на размерите на една снимка. Съдържа две член-данни от тип `unsigned int`.

В класовете `PBM_pixel`, `PGM_pixel` и `PPM_pixel` се съдържа логиката за правилно четене и писане на всеки пиксел. Техните стойности могат да бъдат достъпени и променяни от класовете `Monochrome`, `Grayscale` и `Negative`. Тази връзка е направена чрез деклариране на класовете на действията като *friend* класове на пикселите.

4. Реализация и тестване

4.1. Управление на паметта и сложност

В класовете, където е необходимо, са предефинирани методите на „Голямата четворка“ : *Constructor*, *Copy constructor*, *Destructor* и *operator=* за правилно функциониране на програмата и паметта.

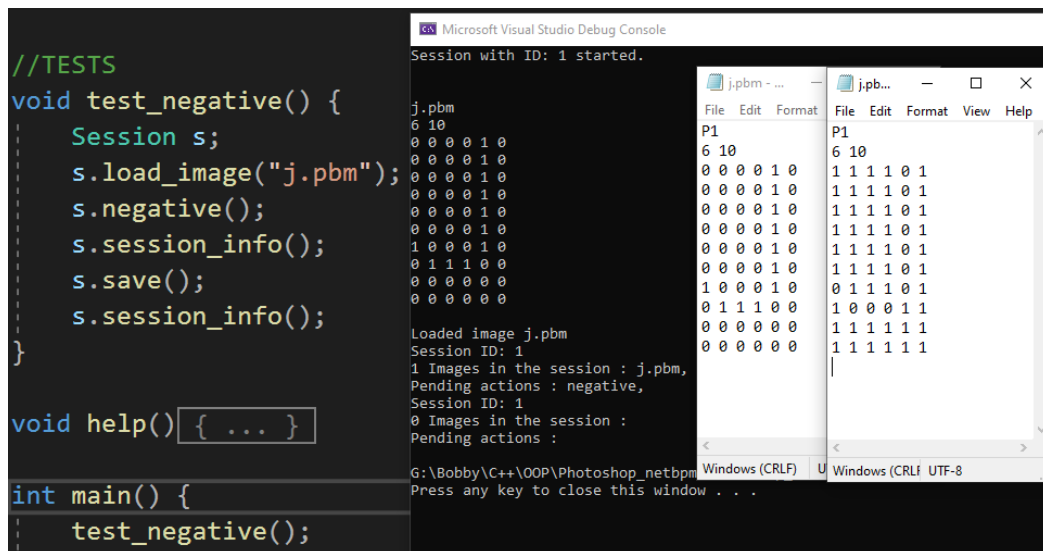
При четене всяко изображение се заделя динамично в паметта, като във сесията се съдържа указател към него. Същото се случва и с всяко действие. Когато бъде зададена командата *undo*, последното зададено действие бива премахнато от масива с действия и не бива извършвано. При задаването на команда *save* се извършват последователно всички действия от масива *actions* и всички изображения се запазват по съответните си файлове, от които са прочетени, а масивите *actions* и *images* биват изтрети.

Алгоритмите в програмата са със сложност $O(nxm)$, където n и m са размерите на даденото изображение.

4.2. Тестване

Програмата е тествана чрез въвеждане на файлове и ръчно проверяване дали алгоритмите са били правилно изпълнени върху тях, както и дали всичко е изписано както трябва в конзолата за потребителя.

Пример:



The screenshot displays the Microsoft Visual Studio Debug Console. On the left, C++ code is shown, including a test function `test_negative()` that loads an image, applies a negative effect, and saves it. The `main()` function calls `test_negative()`. The right side of the console shows the program's execution output. It starts with 'Session with ID: 1 started.', followed by the loading of 'j.pbm' (6x10 pixels). The output then shows the application of the 'negative' action and the saving of the image. The final output indicates that the session is complete and prompts the user to press a key to close the window.

```
//TESTS
void test_negative() {
    Session s;
    s.load_image("j.pbm");
    s.negative();
    s.session_info();
    s.save();
    s.session_info();
}

void help() { ... }

int main() {
    test_negative();
}
```

Microsoft Visual Studio Debug Console

Session with ID: 1 started.

j.pbm
6 10
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
0 0 0 0 1 0
1 0 0 0 1 0
0 1 1 1 0 0
0 0 0 0 0 0
0 0 0 0 0 0

Loaded image j.pbm
Session ID: 1
1 Images in the session : j.pbm,
Pending actions : negative,
Session ID: 1
0 Images in the session :
Pending actions :

G:\Bobby\C++\OOP\Photoshop_netbpm Windows (CRLF)
Press any key to close this window . . .

Фигура 4.2.1

5. Заключение

5.1. Бъдещо развитие и усъвършенстване

Проектът може да бъде разрастнат многократно, като се добавят още начини за модифициране на изображения и имплементиране на нови алгоритми. Тази програма може да служи за основа на някаква среда за модифициране на изображения, подобна на ImageMagick, ако към нея се напише красив потребителски интерфейс. Макар форматът *Netpbm* да е много стар и да не се ползва почти никъде, той би могъл да бъде полезен със своята простота и лесна разчетимост от човешко око, без нуждата от софтуер за визуализация. Така тази програма би помогнала на тези, които нямат знания за растерната графика и имат желание да научат повече за нея.