Обработка на PPM изображения

22.06.2019 София

Никола Тотев

ФN№ : 62271

Специалност: Софтуерно Инжинерство

Група: 6

Курс: 1

Тема

В рамките на този проект съм разработил приложение което обработва PPM, PGM, и PBM файлове. Операциите които се поддържат от приложението са: от дадено изображение да се генерира черно-бяло, монохромно изображение както и хистограма за червените, зелените и сини цветове.

Основни Класове

1. **Pixel** -
    - **Обобщение:**
      Клас който реализира най-малката съставна част на изображението. Сърдържа функции и член-данни които помагат за по-лесно представяне на информацията за изображението която се прочита от файла.
    - **Член-данни:**

      unsigned char pixelData[3] - държи трите компонента на един пиксел:

      -R - на позиция 0,

      -G - на позиция 1,

      -B - на позиция 2.

      int avrg_value - държи средната стойност на трите компонента на пиксела
    - **Конструктори:**

      Pixel() - конструктор по подразбиране, инициализира пиксела със стойност 0 на трите компонента, и 0 за средна стойност.

      Pixel(unsigned char R, unsigned char B, unsigned char G, int avrg) - конструктор с параметри, приема трите компонента R, G, B и средната има стойност avrg. Инициализира член-данните със съответните им стойности.
    - **Предефинирани оператори:**

      operator=

      operator <<

      operator==

- **Мутатори:**

  setPixel(byte R, byte G, byte B) - променя съответните стойности на трита компонента.

  setPixel(byte R, byte G, byte B, byte avrg) - променя съответните стойности на четирите член-данни.

  Където byte = unsigned char

- **Член-функции:**

  toGrayscale() - превръща пиксела от RGB в grayscale използвайки алгоритъма със средните стойности.

  toMonochrome(int limit=128) - превръща пиксела в бял или черен в зависимост от това колко е средната му стойност спрямо дадена граница (*По подразбиране е 128. След няколко теста стигнах до заключението че тази стойност е оптимална, но съм оставил опцията тя да може да се променя при извикването на функцията*).

2. **Image** -
   - **Обобщение:**
     Клас който реализира концепцията за изображение което отговаря на изискванията на Netpbm формата, задно с функциите и член-данните които помагат за реализацията.

   - **Член-данни:**

     Public:

     Enum formats {PPM, PBM, PGM, NA}

     Enum ops{monochrome, grayscale, histogram}

     Private:

     Typedef unsigned int size_int

     const int c_default_width = 256

     const int c_default_height = 256

     bool isGrayscale

bool isMonochrome

Header variables:

Тези променливи служат да пазят информация за изображението. Задават се след прочитането на хедъра на файла.

- ***format*** *format* - съдържа информация за типа изображение, задава се при извикване на функцията determine_format.
- *string file_name* - държи пътя към файла
- *string magic_number* - държи “магическото число” което казва с какъв файл се работи
- *size_int width* - ширината на изображението
- *size_int height* - височината на изображението
- *int bitDepth* - максималната стойност за компонентите на пиксела

Image data variables

Тези променливи държат информацията за картината на изображението.

- *vector<char> raw_data* - “Суровата” информация, като последователност от символи.
- *vector<Pixel> image_data* - Суровата информацията под формата на пиксели. Получава се след обработката на raw_data

Variables used for histogram calculations

- *vector<float> red_count* - броя на различните стойности на червеното 0-255
- *vector<float> green_count* - броя на различните стойности на зеленото 0-255
- *vector<float> blue_count* - броя на различните стойности на синьото 0-255

Operation execution variables

- *vector<ops> operations* - вектор който съдържа всички операции които потребителя е задал.

- *vector<Job::args_enum> op_args* - вектор който съжържа параметрите за зададените операциите. Ако някоя функция няма параметри на съответния индекс седи no_args

- **Конструктори:**

  Image(string fileName) - Има само конструктор с параметър, приема път към изображение. Може да е пълнен или относителен, като относителния се подава в следния формат "./image_name.ppm". ***(Кавичките са задължителни както за относителния, така и за пълния път.)***

  Image(const Image& rhs) -

- **Предефинирани оператори:**

  operator=

- **Мутатори:**

  set_file_name(string _fileName)

  set_magic_number(string _magicNumber)

  set_width(size_int _width)

  set_height(size_int _height)

  set_bit_depth(int _depth)

- **Селектори:**

  vector<Pixel> get_image_data()

  vector<float> getReds()

  vector<float> getGreens()

  vector<float> getBlues()

- **Член-функции:**

  Private:

initVectors() - Инициализира векторите които броят стойностите на R,G & B стойностите на пиксела. (*Колко пискела има със стойност 0, 1, 2... тн за всеки един цвят*)

determineFormat(string fileName) - Опеределя формата на изображението. Връща стойност от *enum formats.*

validHeader(string fileName) - Валидира хедъра на изображението. В тази функция също така се проверява дали файла съществува. Ако файла не съществува или хедъра е невалиден, програмата приключва изпълнение.

readImage(string fileName) - Чете изображението. Първо взима информацията за картината като символи и след това я превръща в пиксели. Докато чете и превръща данните, функцията също така изчислява средните стойности на всеки пикесел и следи дали изображението е монохромно или чернобяло.

updateRawData(formats format) - След промяна на пикселите, трябва да се промени “суровата” информация. Това е вектора от символи който държи информацията за изображението. Функцията превръща променения вектор от пиксели в нов вектор от символи. (актуализира суровата информация)

writeToFile(string fileName) - Създава и записва информацията във файл. Започва с информацията за хедъра, последвана от суровата информация за изображението.

copy(const Image& rhs) - Служи за копиране на изображението. Извиква се в оператора=, както и копи конструктора.

executeTasks() - Изпълнява операциите които са в масива от операции които трябва да се извършат - (*vector<ops> operations*).

Public:

to_grayscale() - Минава през всички пиксели и извиква функцията им toGrayscale(). След което извиква функцията и *updateRawData(formats format)* и след това се записва информацията във файл чрез извикване на *writeToFile(string fileName)*.

to_monochrome() - Минава през всички пиксели и извиква функцията им toMonochrome() с граница 128. След което извиква функцията и *updateRawData(formats format)* и след това се записва информацията във файл чрез извикване на *writeToFile(string fileName)*.

gen_histogram(Job::args_enum target_color) - Генерира хистограма. Първо изчислява колко процента от всеки нюанс имаге от всеки цвят и след това записва тази информация под формата на изображение.

add_operation(ops op, Job::args_enum = Job::args_enum::NA) - Добавя опереция към вектора за опереции заедно с параметрите които са нужни за изпълнението и. Ако няма параметри в вектора за параметри се записва “no_args”.

3. **Command Parser** -
   - **Обобщение:**

     Клас който превежда подадениете аргументи в изпълними команди, след което ги изпълнява. Повече информация за това как работи може да се намери в “Архитектура на проекта”

   - **Член-данни:**

     set<string> acceptedCommands = {“--monochrome”, “--grayscale”, “--histogram =RED”, “--histogram=GREEN”, “--histogram=BLUE”} -s

     string command_start = “-” - индентификатор за начало на команда.

     vector<string> raw_commands - Сурови команди (във формата “--histogram=BLUE”

     vector<string> filePaths - Списък от пътишата към файлове които ще се обработват.

     vector<Job> job_list* - Списък от операциите които трябва да се извършат за едно изображение.

   - **Член-функции:**

     *parse_input(int argc, char *input[])* - Функция кояото превежда поредицата от аргументи в използваеми функции и пътища.

     execute_jobs() - Минава през списъка с работи и за всяка работа извиква функцията *execute()* на нова нишка.

4. **Job** -
    - **Обобщение:**

      Всяко изображение което се подава се счита като “работа” която трябва да се извърши. Всяка работа има някакъв списък от операции и път към файл върху който трябва да се извършат съответните операции. Класа **Job** реализира тази идея.

    - **Член-данни:**

      set<string> allowed_args = {“RED”, “GREEN”, “BLUE”} - Списък от аргументите които програмата приема.

      vector<string> commands - Списък от “суровите” команди. Получават се при добавяне на команда чрез функцията *add_command(string cmd)*.

      vector<string> final_commands - Списък от командите които вече са форматирани и говори за изпълнение.

      vector<string> command_args - Списък с параметрите за командите. Ако някоя команда няма параметри на съответния индекс седи “no_args”

      string path - Път към файла върху който трябва да се изпълнят операциите.

    - **Член-функции:**

      gen_final_commands() - Функция която разделя суровите команди на команди и параметри.

      add_command(string cmd) - добавя команда към “суровите” команди

      set_path(string _path) - мутатор за пътя.

      execute() - Започва процеса за изпълнение на командите.

      Job copy()* - Функция която прави дълбоко копие на обект от тип **Job**.

Архитектура на приложението

Стартиране

При стартиране на приложението се създава инстанция на Command Parser и се извиква функцията *parse_input()*. Тук се подават аргументите от командния ред.

Обработване на аргументи от командния ред

Създава се инстанция на класът **Job.**

Във функцията *parse_input()* има цикъл който минава през всеки аргумент и определя дали той е команда или път.

Ако е команда проверява дали е валидна, ако не е, приложението приключва работа. При валидна команда тя се слага в списъка с операции на инстанцията на **Job**.

Когато цикълът засече път, той се подава на инстанцията на **Job** която след това се добавя в списъка с работи. След това ако има още аргументи се създава нова инстанция на **Job** и цикълът се повтаря.

Изпълняване на операциите

На инстанцията на **Command parser** се извиква функцията *execute_jobs()*. Това стартира цикъл който минава през списъка с *Jobs* и извиква *execute()* на нова нишка за всики *Jobs*.

Функцията *execute()* на **Job** минава през списъка с операциите, разделя командата от параметрите за нея на два други списъка. След това създава бавова инстанция на класът **Image** зарежда операциите и съответните им аргументи в него, след което извиква *begin_work()* на бавовата инстанция.

При конвертиране към монохромно или черно-бяло изображения, функцията създава собствено копие на базовия клас и извиква за всеки пиксел на копието съответно toMonochrome() или toGrayscale().

За хистограмата не се генерира копие.

След извършване на операциите, новите изображения се записват.

Чужд код

Няма.