Матэматыка

УДК 004.9

Оптимизация поэлементной обработки растровых изображений на языке программирования C++

Е.А. Краснобаев
Учреждение образования «Витебский государственный университет имени П.М. Машерова»

В статье сравнивается быстродействие алгоритмов попиксельной обработки растровых изображений на языке программирования C++ с различными способами оптимизации. Актуальность этой задачи связана с необходимостью повышения быстродействия алгоритмов преобразования изображений для режима реального времени при обработке видеопотоков.

Целью данной работы является определение наилучших способов оптимизации программного кода на языке C++ применительно к задачам цифровой обработки изображений.

Материал и методы. *Измерение быстродействия алгоритма инверсии полноцветного растрового изображения выполнялось с использованием класса System::Diagnostics::Stopwatch статистическим методом с применением пяти способов оптимизации. Для тестирования использовался компьютер с процессором AMD Sempron LE-1200 2,1 ГГц, одноядерный. При компиляции применялась среда программирования Microsoft Visual C++ 2008, CLR-проект, с опцией оптимизации /O2.*

Результаты и их обсуждение. *Для сравнения выбраны способы оптимизации алгоритма обработки изображения: с использованием класса Bitmap, с блокировкой изображения в памяти, с применением методов at, ptr и указателя data класса cv::Mat библиотеки компьютерного зрения OpenCV.*

Заключение. *Наилучший результат достигнут путем обработки изображения через указатель cv::Mat::data с оптимизацией цикла. Скорость обработки полноцветного изображения с разрешением 800x600 пикселей достигла 10 мс. Полученные способы оптимизации могут использоваться для решения задач потоковой обработки изображения в режиме реального времени.*

Ключевые слова: *оптимизация алгоритмов, быстродействие алгоритмов, цифровая обработка изображений, OpenCV.*

Optimization of Per-Pixel Processing of Raster Images in the Programming Language of C++

E.A. Krasnobaev
Educational establishment «Vitebsk State P.M. Masherov University»

In the article the algorithm speed for raster image processing in the C++ programming language with different methods of optimization is compared. The urgency of this task is determined by the need to improve the performance of algorithms to convert images to real time video stream processing.

The purpose of this article is to determine best ways to optimize the code in C++, with reference to problems of digital image processing.

Material and methods. *Measuring the performance of the inversion algorithm of full-color bitmap image was conducted using the class of System::Diagnostics::Stopwatch, using five methods of optimization. For testing computer with an AMD Sempron LE-1200 2,1 GHz single core was used. For compiling Microsoft Visual C++ 2008 CLR-project with the option of optimization /O2 was used.*

Findings and their discussion. *To compare methods of optimization using: class Bitmap, with blocking images in memory were selected, using the methods at, ptr and pointer data of cv::Mat class computer vision library OpenCV.*

Conclusion. *The best result is achieved by processing the image through a pointer cv::Mat::data with the optimization cycle. Processing speed of full-color image with a resolution of 800x600 pixels reached 10 ms. The obtained optimization methods can be used for solving the problems of streaming image processing in real time.*

Key words: *optimization algorithms, the speed of algorithms, digital image processing, OpenCV.*

Многие задачи цифровой обработки изображений и распознавания образов основываются на поэлементных преобразованиях изображений. Эффективность алгоритмов, выполняющих такие преобразования, во многом зависит от их быстродействия. Цифровое изображение, например, стандартного разрешения 1024x768, содержит чуть менее миллиона пикселей, которые

участвуют в преобразовании. В связи с этим незначительная оптимизация способов доступа к пикселям изображения, организации цикличности, способа передачи параметров в функцию позволяет существенно ускорить преобразование изображения.

Целью данной работы является определение наилучших способов оптимизации программного кода на языке C++ применительно к задачам цифровой обработки изображений.

Материал и методы. Измерение быстродействия алгоритма инверсии полноцветного растрового изображения выполнялось с использованием класса System::Diagnostics::Stopwatch статистическим методом с применением пяти способов оптимизации. Для тестирования использовался компьютер с процессором AMD Sempron LE-1200 2,1 ГГц, одноядерный. При компиляции, применялась среда программирования Microsoft Visual C++ 2008, CLR-проект, с опцией оптимизации /O2.

Результаты и их обсуждение. В данном исследовании предлагается сравнить наиболее распространенные и специфичные подходы к организации обработки растрового изображения на языке C++ и определить наиболее быстродействующий из них. Актуальность этой задачи связана с необходимостью повышения быстродействия алгоритмов преобразования изображений для режима реального времени при обработке видеопотоков.

Для проведения исследования применялась среда программирования Microsoft Visual C++ 2008, CLR-проект. В качестве типовой задачи цифровой обработки изображения выбрана процедура инверсии (негатив) полноцветного изображения формата RGB, выполняемая по следующей формуле:

$$s(x,y) = L - 1 - r(x,y),$$

где $s(x,y)$ и $r(x,y)$ – значения яркостей результирующего и исходного пикселя (x,y) изображения (для каждой цветовой компоненты); диапазон яркостей – $[0, L-1]$. Подобный переворот уровней яркости изображения создает эквивалент фотографического негатива.

Рассмотрим пять способов решения данной задачи, исходя из цели нахождения наиболее быстродействующего.

1. Первый способ базируется на использовании для хранения и доступа к пикселям изображения стандартного класса System::Drawing::Bitmap. Данный класс инкапсулирует точечный рисунок GDI+, состоящий из пикселей графического изображения и атрибутов рисунка. Объект Bitmap применяется для работы с изображениями, определяемыми его пикселями. Класс Bitmap содержит методы:

– public void SetPixel(int x, int y, Color color) – задает цвет указанного пикселя в данном изображении;

– public Color GetPixel(int x, int y) – получает цвет указанного пикселя в данном изображении.

В связи с этим метод Negative1, тестового класса Image, выполняющий инверсию изображения, примет следующий вид (листинг № 1):

```
void Image::Negative1(Bitmap^
image){
for (int x = 0; x < image->Width;
x++){
for (int y = 0; y < image->Height;
y++){
image->SetPixel(x, y,
Color::FromArgb(255 - image-
>GetPixel(x, y).R, 255 - image-
>GetPixel(x, y).G, 255 - image-
>GetPixel(x, y).B));}}
```

Для работы применялось изображение формата Format24bppRgb. Данный формат сохраняет цвета пикселей с помощью 24 бит, по 8 каждого компонента: красного, зеленого и синего цвета. Этот подход является наиболее простым, однако медленное исполнение функций SetPixel и GetPixel заставляет искать другие способы решения задачи.

2. Второй способ предполагает использование блокировки объекта Bitmap в системной памяти (как и рекомендуется в справочной документации), позволяющей ускорить его обработку. Для этого применяются два метода класса Bitmap:

– public BitmapData LockBits(Rectangle rect, ImageLockMode flags, PixelFormat format) – блокирует объект Bitmap в системной памяти;

– public void UnlockBits(BitmapData bitmapdata) – выполняет разблокировку изображения Bitmap из системной памяти.

Блокировка изображения в памяти позволяет получить указатель на массив пикселей изображения и дает возможность самостоятельно перегрузить функции чтения и записи значений в пиксели изображения, что и выполнено в специально разработанном классе Image. Процесс блокировки изображения в памяти приведен ниже:

```
Imaging::BitmapData^ imageData =
image->LockBits( rect,
Imaging::ImageLockMode::ReadWrite,
image->PixelFormat );
unsigned char* imagep = (unsigned
char*)imageData->Scan0.ToPointer();
```

Особенностью рассмотренного подхода является необходимость перерасчета индекса пикселя одномерного массива в двумерные координаты. Реализация примера инверсии изображения (листинг № 2) будет во многом аналогична листингу № 1:

```
void Image::Negative2(Image^ result){
for (int x = 0; x < width; x++){
for (int y = 0; y < height; y++){
result->SetPixel(x, y,
255 - this->GetPixelR(x, y),
255 - this->GetPixelG(x, y),
255 - this->GetPixelB(x, y)); }}}
```

3. Третий способ предполагает использование для обработки изображения сторонней библиотеки OpenCV 2.4. OpenCV – библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов с открытым исходным кодом. Она реализована на языке C/C++ и призвана стандартизировать разработку приложений в данной области. Множество функций библиотеки позволяет выполнять базовые операции обработки больших числовых массивов и изображений. В частности – выполнять фильтрацию изображений, нахождение отличительных признаков изображений, анализ движения, сравнение изображений, обнаружение объектов, например лиц, восстановление изображений, морфологический анализ и многое другое [1–2].

Библиотека компьютерного зрения OpenCV реализована на языке Си. Изначально, начиная с версий 1.0, она поддерживала только Си интерфейс, а с версии 2.0 включена поддержка объектно-ориентированного C++ интерфейса, который в версии 2.4 практически вытеснил устаревшие функции. C++ интерфейс более удобен, имеет большую функциональность и активно развивается. Поэтому разработанная программа реализована полностью на C++ интерфейсе.

Начиная с версии 2.0 осуществляется переход от структуры IplImage для хранения изображений к классу cv::Mat – многомерный массив, причем все больше функций обработки изображений отказывается от устаревшего формата.

Для реализации негатива изображения вначале выполняется загрузка изображения в формат cv::Mat с помощью функции cv::imread. Изображение хранится в формате CV_8UC3, соответствующем цветовым каналам RGB по 8 бит каждый.

В OpenCV есть несколько способов доступа к пикселям изображения. Они различны по степени безопасности (контроль типов и выхода за границы), по скорости работы и удобству. Один из способов доступа к пикселям изображений, у которых известен тип, – это использование метода at класса cv::Mat (листинг № 3):

```
void Image::Negative3(cv::Mat* img){
for (int y = 0; y < img->rows; y++) {
for (int x = 0; x < img->cols; x++) {
img->at<cv::Vec3b>(y, x) =
cv::Vec3b(255, 255, 255) - img->
at<cv::Vec3b>(y, x);}}}
```

4. В четвертом способе [3] используется метод ptr(), который возвращает указатель на первый пиксель нужной строки. Как известно, если выполнять цикл не по возрастанию, а по убыванию, то цикл всегда будет выполняться до нуля, и компилятору не нужно будет применять ассемблерную операцию сравнения cmp для проверки условия остановки цикла (листинг № 4):

```
void Image::Negative4(cv::Mat* img){
for(int y(img->rows - 1); y >= 0; --y){
cv::Vec3b* line (img->
ptr<cv::Vec3b>(y) );
for(int x(img->cols - 1); x >= 0;--x){
line[x] = cv::Vec3b(255, 255, 255) -
scanLine[x];}}}
```

5. Пятый способ предполагает использование указателя на данные изображения cv::Mat::data. Одна из причин того, почему лучше применять указатель data, заключается в том, что он хранит адрес начала массива данных изображения, и затем над этим указателем можно совершать арифметические операции. Также важно помнить, что в многоканальной матрице каналы являются смежными. Например, в трехканальной двумерной матрице, представляющей собой красный, зеленый и синий байты изображения, элементы отсортированы так: r g b r g b r g b… Следовательно, чтобы переместиться на следующий канал, необходимо увеличить указатель на 1. Если необходимо переместиться к следующему пикселю, нужно прибавить к указателю количество каналов (в данном случае 3) [4].

Как видно из предыдущих примеров для перебора пикселей изображения использовались вложенные циклы for. В предлагаемом примере (листинг № 5) выбран цикл do-while, и цветовые составляющие изображения обрабатываются последовательно, в соответствии с их расположением в памяти, а не по порядку следования их двумерных координат (листинг № 5).

```
void Image::Negative5(cv::Mat* img){
int len = img->rows * img->cols * 3 - 1;
uchar * p = img->data;
do{
p[len] = 255 - p[len];}
while(--len);}
```

Таблица

Результаты 3-х измерений быстродействия алгоритмов

№ п/п	Способ оптимизации	Время обработки полноцветного изображения с указанным разрешением, мс	
		2560x1920	800x600
1.	Использование класса Bitmap, со стандартными методами доступа SetPixel(), GetPixel(), листинг № 1	22423 22160 22874	2072 2066 2080
2.	Применение класса Bitmap с блокировкой изображения в памяти, перегруженными методами доступа к пикселям, листинг № 2	2470 2441 2500	102 102 106
3.	Использование класса cv::Mat с методом доступа at(), листинг № 3	480 452 452	47 50 46
4.	Применение класса cv::Mat с доступом через метод ptr() и оптимизацией цикла, листинг № 5	179 179 175	19 18 19
5.	Использование класса cv::Mat с доступом через указатель data и оптимизацией цикла, листинг № 5	100 105 100	9 10 10

Изображение опять же имеет формат CV_8UC3, в связи с этим выполнено преобразование указателя cv::Mat::data в тип uchar.

Для сравнения быстродействия приведенных программных реализаций алгоритма применялся класс System::Diagnostics::Stopwatch. Данный класс содержит набор методов и средств, которые можно использовать для точного измерения затраченного времени исполнения программного кода. Для тестирования применялся компьютер с процессором AMD Sempron LE-1200 2,1 ГГц, одноядерный. При компиляции CLR-проекта использовалась опция оптимизации /O2. Результаты измерения приведены в табл.

Заключение. В результате исследования выполнено преобразование тестовых изображений в негатив пятью способами. Первые два способа (листинги № 1 и № 2) используют для работы с изображением класс System::Drawing::Bitmap. Стандартные методы SetPixel и GetPixel класса Bitmap выполняются крайне медленно, что подтверждается измерением их быстродействия – около 22 с затрачено на обработку изображения размером 2560x1920 пикселей. Применение способа блокировки изображения в памяти позволяет получить указатель на массив данных изображения и перегрузить методы доступа к пикселю. Это на порядок уменьшает время обработки такого изображения, но, тем не менее, выполняется недостаточно быстро.

Применяя библиотеку компьютерного зрения OpenCV и класс для работы с изображением cv::Mat можно добиться ускорения обработки изображения, используя метод доступа к пикселям at() и метод построчного доступа ptr(). Применение векторов типа cv::Vec3b позволяет дополнительно оптимизировать обработку цветовых компонент пикселей изображения.

Анализируя данные, приведенные в табл., можно увидеть, что наилучший результат достигнут путем обработки изображения через указатель cv::Mat::data с оптимизацией цикла. Скорость обработки полноцветного изображения с разрешением 800x600 пикселей достигла 10 мс, при выполнении на откровенно слабом процессоре. Известно, что стандартная видеопоследовательность поступает с видеокамеры в режиме реального времени со скоростью порядка

20–25 кадров/с. Следовательно, временная задержка между кадрами составляет около 40–50 мс именно этим временем и должны ограничиваться алгоритмы обработки кадра видеопоследовательности. Полученные алгоритмы удовлетворяют этому требованию и могут использоваться для решения задач потоковой обработки изображения в реальном времени. Полученный результат не исчерпал всех способов оптимизации и, вероятно, может быть улучшен.

ЛИТЕРАТУРА

1. Bradski, G. Learning OpenCV / G. Bradski, A. Kaehler. – O'Reilly, 2008. – 576 p.
2. Краснобаев, Е.А. Разработка программного обеспечения счетчика посетителей с использованием библиотеки компьютерного зрения OpenCV 2.4 // Электроника инфо. – 2013. – № 1(91). – С. 22–24.
3. OpenCV: цикл по всем пикселям изображения и совмещение указателей // Блог Image Processing [Электронный ресурс]. – 2014. – Режим доступа: http://iproc.ru/image-processing/iterating-cv-mat-and-pointer-aliasing/. – Дата доступа: 28.11.2014.
4. Структура – CvMat // Learning OpenCV [Электронный ресурс]. – 2014. – Режим доступа: http://locv.ru/wiki/3.3_Структура_-_CvMat. – Дата доступа: 28.11.2014.

REFERENCES

1. Bradski, G. Learning OpenCV / G. Bradski, A. Kaehler. – O'Reilly, 2008 – 576 p.
2. Krasnobayev E.A. *Elektronika info* [Electronics Info], 2013, 1(91), pp. 22–24.
3. *OpenCV: tsikl po vsem pikseliam izobrazheniya i sovmeshcheniye ukazatelei* [OpenCV: Cycle on all Image Pixels and Correlation of indicators], 2014, http://iproc.ru/image-processing/iterating-cv-mat-and-pointer-aliasing.
4. *Struktura CvMat//Learning OpenCV* [Structure – CvMat // Learning OpenCV], 2014, http://locv.ru/wiki/3.3_Структура_-_CvMat.

Поступила в редакцию 08.12.2015
Адрес для корреспонденции: e-mail: krasnobaev@tut.by – Краснобаев Е.А.