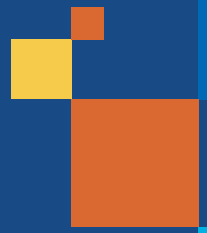
Winter Camp 2022

Оптимизация в OpenCV

Виталий Тузов, Дмитрий Куртаев



intel®

План лекции

- OpenCV
  - Краткий обзор
  - Основные структуры
  - Примеры функций и методов
- Пишем алгоритм
- Тестируем написанное
  - Тест корректности
  - Тест производительности
- Оптимизируем алгоритм
- Практика

OpenCV

- Проект с открытым исходным кодом на GitHub с >3M загрузок в год
- Самая популярная библиотека компьютерного зрения с 20-летней историей разработки
- 4 больших релиза: 1.0, 2.4.x, 3.4.x, 4.x
- Модульная структура: core, imgproc, ts, dnn, stitching, ...
- Написана на C++, имеет интерфейсные обертки для Python, Java, JavaScript, Matlab, GO, PHP, C#, etc.
- Кросплатформенная и хорошо оптимизированная библиотека подходит как для исследовательской деятельности так и для промышленной разработки

Базовые структуры OpenCV

- **`cv::Mat`** – для изображений, масок, векторов, комплексных значений и пользовательских данных

```
cv::Mat mat(480, 640, CV_8UC3);
int rows     = mat.rows;         // 480
int cols     = mat.cols;         // 640
int channels = mat.channels();   // 3
uint8_t* data = mat.ptr<uint8_t>();
```

- Типы **`cv::Mat`**: [глубина | (т.е. **`CV_8U, CV_16F, CV_32F, CV_64F`**)] + количество каналов

- **`std::cout << mat << std::endl`** – Для вывода матрицы в консоль и просмотра её значений

Базовые структуры OpenCV

```
cv::Rect rect;
int x = rect.x;
int y = rect.y;
int w = rect.width;
int h = rect.height;
```

```
cv::Point point;
int x = point.x;
int y = point.y;
```

```
cv::Size size;
int w = size.width;
int h = size.height;
```

Методы OpenCV

Большинство методов использует в качестве входов и выходов базовые структуры OpenCV

```
cv::Mat src, dst, mask;

cv::cvtColor(src, dst, COLOR_BGR2GRAY);

cv::resize(src, dst, cv::Size(1280, 960));

cv::Canny(src, dst, /*threshold1*/ 100, /*threshold2*/ 200);

cv::inpaint(src, mask, dst, /*inpaintRadius*/ 3, cv::INPAINT_TELEA);

std::vector<cv::Mat> images;
cv::Ptr<Stitcher> stitcher =
cv::Stitcher::create(cv::Sticher::PANORAMA);
stitcher->stitch(images, dst);
```

Пишем алгоритм

- Конвертация цветного изображения в оттенки серого
- Стандартная цветовая модель в OpenCV - RGB
- Преобразование в оттенки серого(яркость) задается весовыми коэффициентами R -> 0.299 G -> 0.587 B -> 0.114
- Преобразование применяется к каждой точке изображения
- OpenCV хранит точки в памяти непрерывно расположеными триплетами значений каналов с обратным порядком расположения цветов(BGR)

Пишем алгоритм

```
void bgr2gray_reference(const cv::Mat& src, cv::Mat& dst) {
    const int w = src.cols;
    const int h = src.rows;
    dst.create(h, w, CV_8UC1);

    const uint8_t* src_data = src.ptr<uint8_t>();
    uint8_t* dst_data = dst.ptr<uint8_t>();
    for (int y = 0; y < h; ++y) {
        for (int x = 0; x < w; ++x) {
            uint8_t b = src_data[x * 3];
            uint8_t g = src_data[x * 3 + 1];
            uint8_t r = src_data[x * 3 + 2];
            dst_data[x] = static_cast<uint8_t>(0.114f * b +
                                               0.587f * g +
                                               0.299f * r);
        }
        dst_data += w;
        src_data += w * 3;
    }
}
```

Регрессионные тесты

Для тестирования в OpenCV используется модуль ts.

Модуль состоит из:

- Инфраструктуры тестирования основанной на Google Test
- Специализированных OpenCV макросов для регрессионного тестирования и тестирования производительности
- Python-скриптов для анализа результатов тестирования

Регрессионные тесты: пример

- Непараметрический тест определяется с использованием макроса **TEST**
- Численные и логические проверки задаются макросам **EXPECT_***

```
#include <opencv2/ts.hpp>
TEST(bgr2gray, u8)
{
    cv::Mat src(10, 11, CV_8UC3), ref, dst;
    randu(src, 0, 255);
    bgr2gray_reference(src, ref);
    bgr2gray_impl(src, dst);
    double maxV;
    minMaxLoc(abs(ref - dst), 0, &maxV);
    EXPECT_LE(maxV, 1);
}
```

Регрессионные тесты: пример

```
$ ./bin/test_algo --gtest_filter=bgr2gray.u8

[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from bgr2gray
[ RUN      ] bgr2gray.u8
[       OK ] bgr2gray.u8 (15 ms)
[----------] 1 test from bgr2gray (18 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (21 ms total)
[  PASSED  ] 1 test.
```

Параметризованные регрессионные тесты

- Задаются набором параметров
- Генерируются как набор тестов со всевозможными комбинациями параметров

```
typedef TestWithParam<tuple<int, int> > bgr2gray;
TEST_P(bgr2gray, param)
{
    Mat src(/*rows*/ get<0>(GetParam()), /*cols*/ get<1>(GetParam()), CV_8UC3),
ref, dst;
    randu(src, 0, 255);

    bgr2gray_reference(src, ref);
    bgr2gray_impl(src, dst);

    EXPECT_EQ(countNonZero(ref != dst), 0);
}
INSTANTIATE_TEST_CASE_P(/**/, bgr2gray, Combine( Values(3, 4), Values(2, 5) ));
```

Параметризованные регрессионные тесты

```
[==========] Running 4 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 4 tests from bgr2gray
[ RUN      ] bgr2gray.param/0, where GetParam() = (3, 2)
[       OK ] bgr2gray.param/0 (14 ms)
[ RUN      ] bgr2gray.param/1, where GetParam() = (3, 5)
[       OK ] bgr2gray.param/1 (0 ms)
[ RUN      ] bgr2gray.param/2, where GetParam() = (4, 2)
[       OK ] bgr2gray.param/2 (0 ms)
[ RUN      ] bgr2gray.param/3, where GetParam() = (4, 5)
[       OK ] bgr2gray.param/3 (0 ms)
[----------] 4 tests from bgr2gray (24 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test case ran. (27 ms total)
[  PASSED  ] 4 tests.
```

Тесты производительности

- Тест производительности определяется с использованием макроса **PERF_TEST**
- Целевой код отмечается макросами **PERF_SAMPLE_BEGIN()** - **PERF_SAMPLE_END()**
- OpenCV автоматически выберет количество итераций необходимое для стабильности метрики

```
PERF_TEST(bgr2gray, u8_perf)
{
    cv::Mat src(480, 640, CV_8UC3), dst;

    PERF_SAMPLE_BEGIN()
        bgr2gray_u8_impl(src, dst);
    PERF_SAMPLE_END()

    SANITY_CHECK_NOTHING();
}
```

Тесты производительности

```
$ ./bin/perf_algo --gtest_filter=bgr2gray.u8_perf

[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from bgr2gray
[ RUN      ] bgr2gray.u8_perf
[ PERFSTAT ]    (samples=100   mean=0.25   median=0.25
min=0.22   stddev=0.02 (9.5%))
[       OK ] bgr2gray.u8_perf (28 ms)
[----------] 1 test from bgr2gray (29 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (31 ms total)
[  PASSED  ] 1 test.
```

Оптимизируем алгоритм

```
void bgr2gray_impl(const cv::Mat& src, cv::Mat& dst) {
    const int w = src.cols;
    const int h = src.rows;
    dst.create(h, w, CV_8UC1);

    const uint8_t* src_data = src.ptr<uint8_t>();
    uint8_t* dst_data = dst.ptr<uint8_t>();
    for (int y = 0; y < h; ++y) {
        for (int x = 0; x < w; ++x) {
            uint8_t b = src_data[x * 3];
            uint8_t g = src_data[x * 3 + 1];
            uint8_t r = src_data[x * 3 + 2];
            dst_data[x] = static_cast<uint8_t>(0.114f * b +
                                               0.587f * g +
                                               0.299f * r);
        }
        dst_data += w;
        src_data += w * 3;
    }
}
```

Оптимизируем алгоритм

```
void bgr2gray_impl(const cv::Mat& src, cv::Mat& dst) {
    const int w = src.cols;
    const int h = src.rows;
    dst.create(h, w, CV_8UC1);

    const uint8_t* src_data = src.ptr<uint8_t>();
    uint8_t* dst_data = dst.ptr<uint8_t>();
    for (int y = 0; y < h; ++y) {
        for (int x = 0; x < w; ++x) {
            uint8_t b = src_data[x * 3];
            uint8_t g = src_data[x * 3 + 1];
            uint8_t r = src_data[x * 3 + 2];
            dst_data[x] = (29 * b +
                           150 * g +
                            77 * r) >> 8;
        }
        dst_data += w;
        src_data += w * 3;
    }
}
```

Исходная версия: 4.02ms @ 1920x1080

Целочисленная версия: 2.39ms @ 1920x1080 (x1.68)

OpenCV `parallel_for_`

- Простой способ параллелизации для алгоритмов допускающих независимое деление входных данных между потоками
- Алгоритм описывается как функция обработки части данных определяемой входным параметром
- Есть руководство по использованию https://docs.opencv.org/4.x/dc/ddf/tutorial_how_to_use_OpenCV_parallel_for_new.html

OpenCV `parallel_for_`

Поддерживаются различные фреймворки в зависимости от используемой ОС и параметрров сборки

1. Intel Threading Building Blocks (TBB)
2. C= Parallel C/C++ Programming Language Extension
3. OpenMP
4. APPLE GCD
5. Windows RT concurrency
6. Windows concurrency
7. Pthreads

Оптимизируем алгоритм

```
void bgr2gray_impl(const cv::Mat& src, cv::Mat& dst) {
    const int w = src.cols;
    const int h = src.rows;
    dst.create(h, w, CV_8UC1);

    parallel_for_(cv::Range(0, h), [&](const cv::Range& range) {
        const uint8_t* src_data = src.ptr<uint8_t>(range.start);
        uint8_t* dst_data = dst.ptr<uint8_t>(range.start);
        for (int y = range.start; y < range.end; ++y) {
            for (int x = 0; x < w; ++x) {
                uint8_t b = src_data[x * 3];
                uint8_t g = src_data[x * 3 + 1];
                uint8_t r = src_data[x * 3 + 2];
                dst_data[x] = (29 * b + 150 * g + 77 * r) >> 8;
            }
            dst_data += w;
            src_data += w * 3;
        }
    });
}
```

Исходная версия: 4.02ms @ 1920x1080
Целочисленная версия: 2.39ms @ 1920x1080 (x1.68)
Параллельная версия: 1.83ms @ 1920x1080 (x2.19)

Универсальные интринсики OpenCV

- Реализуют концепцию SIMD
- Обеспечивающих ручную векторизацию с использованием платформозависимых операций
- Представлены набором функций оперирующих над специализированными типами данных
- Есть руководство по использованию https://docs.opencv.org/4.x/d6/dd1/tutorial_univ_intrin.html

Универсальные интринсики OpenCV

- Универсальные интринсики поддерживают большой набор архитектур:
  - AVX / SSE / SIMD (x86)
  - NEON (ARM)
  - VSX (PowerPC)
  - MSA (MIPS)
  - RISC-V Vector Instructions
  - WASM (JavaScript)

Универсальные интринсики OpenCV

```
#include <opencv2/core/hal/intrin.hpp>
// ...
std::vector<int> data = {1, 2, 3, 4, 5, 6, 7, 8};

cv::v_int32x4 twos = cv::v_setall_s32(2);

cv::v_int32x4 b0 = cv::v_load(&data[0]);
b0 *= twos;
v_store(&data[0], b0);

b0 = cv::v_load(&data[4]);
b0 -= twos;
v_store(&data[4], b0);
// data = {2, 4, 6, 8, 3, 4, 5, 6}
```

Пример оптимизации: детектор границ

1. Оператор Собеля

$$Gx = \begin{vmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{vmatrix} * A, \quad Gy = \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{vmatrix} * A$$

```
cv::Sobel(src, dst, CV_8U, 1, 0);  // d/dx
cv::Sobel(src, dst, CV_8U, 0, 1);  // d/dy
```

Пример оптимизации: детектор границ

2. Оператор Прюитт

$$Gx = \begin{vmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{vmatrix} * A, \quad Gy = \begin{vmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{vmatrix} * A$$

3. Крест Робертса

$$Gx = \begin{vmatrix} +1 & 0 \\ 0 & -1 \end{vmatrix} * A, \quad Gy = \begin{vmatrix} 0 & +1 \\ -1 & 0 \end{vmatrix} * A$$

Пример оптимизации: оператор Прюитт

```
void prewitt_x(const Mat& src, Mat& dst) {
    CV_Assert(src.type() == CV_8UC1);
    Mat bsrc;
    copyMakeBorder(src, bsrc, 1, 1, 1, 1, BORDER_REPLICATE);
    dst.create(src.size(), CV_8UC1);
    for (int y = 0; y < dst.rows; ++y)
        for (int x = 0; x < dst.cols; ++x) {
            dst.at<uchar>(y, x) = bsrc.at<uchar>(y    , x + 2) -
                                  bsrc.at<uchar>(y    , x)     +
                                  bsrc.at<uchar>(y + 1, x + 2) -
                                  bsrc.at<uchar>(y + 1, x)     +
                                  bsrc.at<uchar>(y + 2, x + 2) -
                                  bsrc.at<uchar>(y + 2, x);
        }
}
```

Пример оптимизации: оператор Прюитт

```
void prewitt_x_parallel(const Mat& src, Mat& dst) {
    Mat bsrc;
    copyMakeBorder(src, bsrc, 1, 1, 1, 1, BORDER_REPLICATE);
    dst.create(src.size(), CV_8UC1);
    parallel_for_(Range(0, src.rows), [&](const Range& range) {
      for (int y = range.start; y < range.end; ++y)
          for (int x = 0; x < dst.cols; ++x) {
              dst.at<uchar>(y, x) = bsrc.at<uchar>(y    , x + 2) -
                                    bsrc.at<uchar>(y    , x)     +
                                    bsrc.at<uchar>(y + 1, x + 2) -
                                    bsrc.at<uchar>(y + 1, x)     +
                                    bsrc.at<uchar>(y + 2, x + 2) -
                                    bsrc.at<uchar>(y + 2, x);
          }
    });
}
```

Исходная версия: 12.76ms @ 1920x1080
Параллельная версия: 9.83ms @ 1920x1080 (x1.29)

Пример оптимизации: оператор Прюитт

```
parallel_for_(Range(0, src.rows), [&](const Range& range) {
    for (int y = range.start; y < range.end; ++y) {
        const uint8_t* psrc0 = bsrc.ptr(y);
        const uint8_t* psrc1 = bsrc.ptr(y + 1);
        const uint8_t* psrc2 = bsrc.ptr(y + 2);
        uint8_t* pdst = dst.ptr(y);
        int x = 0;
        for (; x <= dst.cols - v_uint8::nlanes; x += v_uint8::nlanes) {
            v_uint8 res = vx_load(psrc0 + x + 2) - vx_load(psrc0 + x) +
                          vx_load(psrc1 + x + 2) - vx_load(psrc1 + x) +
                          vx_load(psrc2 + x + 2) - vx_load(psrc2 + x);
            v_store(pdst + x, res);
        }
        for (; x < dst.cols; ++x) {
            pdst[x] = psrc0[x + 2] - psrc0[x] +
                      psrc1[x + 2] - psrc1[x] +
                      psrc2[x + 2] - psrc2[x];
        }
    }
});
```

Исходная версия:
12.76ms @ 1920x1080
Параллельная версия:
9.83ms @ 1920x1080 (x1.29)
Параллельная векторизованная версия: 2.57ms @ 1920x1080 (x4.96)

Пример оптимизации: оператор Прюитт

```
for (; x <= dst.cols - v_uint8::nlanes; x += v_uint8::nlanes) {
    v_uint8 res = vx_load(psrc0 + x + 2) - vx_load(psrc0 + x) +
                  vx_load(psrc1 + x + 2) - vx_load(psrc1 + x) +
                  vx_load(psrc2 + x + 2) - vx_load(psrc2 + x);
    v_store(pdst + x, res);
}
```

```
for (; x <= dst.cols - v_uint8::nlanes; x += v_uint8::nlanes) {
    v_uint8 res = v_add_wrap(v_sub_wrap(vx_load(psrc0 + x + 2), vx_load(psrc0 + x)),
                  v_add_wrap(v_sub_wrap(vx_load(psrc1 + x + 2), vx_load(psrc1 + x)),
                             v_sub_wrap(vx_load(psrc2 + x + 2), vx_load(psrc2 + x))));
    v_store(pdst + x, res);
}
```

Исходная версия: 12.76ms @ 1920x1080
Параллельная векторизованная версия: 2.61ms @ 1920x1080 (x4.88)

Пример оптимизации: оператор Прюитт

```
int y = range.start;
for (; y <= range.end - 2; ++y) {
     const uint8_t* psrc0 = bsrc.ptr(y);
     const uint8_t* psrc1 = bsrc.ptr(y + 1);
     const uint8_t* psrc2 = bsrc.ptr(y + 2);
     const uint8_t* psrc3 = bsrc.ptr(y + 3);
     uint8_t* pdst0 = dst.ptr(y);
     uint8_t* pdst1 = dst.ptr(y+1);
     int x = 0;


     // Обработка двух рядов одновременно
     ....................................


}
```

Пример оптимизации: оператор Прюитт

```
for (; x <= dst.cols - v_uint8::nlanes; x += v_uint8::nlanes) {
    v_uint8 res = v_add_wrap(v_sub_wrap(vx_load(psrc1 + x + 2), vx_load(psrc1 + x)),
                             v_sub_wrap(vx_load(psrc2 + x + 2), vx_load(psrc2 + x)));
    v_store(pdst0 + x, v_add_wrap(res, v_sub_wrap(vx_load(psrc0 + x + 2),
                                                  vx_load(psrc0 + x))));
    v_store(pdst1 + x, v_add_wrap(res, v_sub_wrap(vx_load(psrc3 + x + 2),
                                                  vx_load(psrc3 + x))));
}
for (; x < dst.cols; ++x) {
    uint8_t res = psrc1[x + 2] - psrc1[x] + psrc2[x + 2] - psrc2[x];
    pdst0[x] = res + psrc0[x + 2] - psrc0[x];
    pdst1[x] = res + psrc3[x + 2] - psrc3[x];
}
```

Исходная версия: 12.76ms @ 1920x1080
Оптимизированная версия: 2.54ms @ 1920x1080 (x5.02)

Практика

- Реализовать крест Робертса:

```
input:  cv::Mat (single channel, uint8_t)
output: cv::Mat (single channel, int32_t)

out = (Gx)^2 + (Gy)^2, where

      | +1   0  |              |  0  +1 |
Gx =  |  0  -1  | * A,   Gy =  | -1   0 | * A
```

- Написать регрессионные тесты
- Паралелизовать и векторизовать реализацию
- Написать тесты производительности и сравнить эффективность оптимизированной и исходной версий


intel