OpenCV

Optimizations with OpenCV

Agenda

- OpenCV
  - Brief overview
  - Basic structures
  - Example functions and methods
- Let’s write an algorithm
- Testing module
  - Accuracy test
  - Performance test
- Let’s optimize
- Practice / Homework

OpenCV

- Open source project on GitHub with >3M downloads per year
- The most popular computer vision library with 20 years of development history
- 4 major releases: `1.0`, `2.4.x`, `3.4.x`, `4.x`
- Modular structure: `core`, `imgproc`, `ts`, `dnn`, `stitching`, ...
- Written in C++ but has automatic wrappers in Python, Java, JavaScript, Matlab, GO, PHP, C#, etc.
- Cross-platform and well optimized for research and development

OpenCV basic structures

- **`cv::Mat`** - for images, masks, vector fields, complex values and custom data

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

- **`cv::Mat`** types: **`[depth | (i.e. CV_8U, CV_16F, CV_32F, CV_64F)] + channels`**
- **`std::cout << mat << std::endl`** - To print **`cv::Mat`** in console and watch values

OpenCV basic structures

```
01  cv::Rect rect;
02  int x = rect.x;
03  int y = rect.y;
04  int w = rect.width;
05  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;
```

OpenCV methods

Most of the methods work with basic OpenCV data structures as input and output

```
01 cv::Mat src, dst, mask;
02
03 cv::cvtColor(src, dst, COLOR_BGR2GRAY);
04
05 cv::resize(src, dst, cv::Size(1280, 960));
06
07 cv::Canny(src, dst, /*threshold1*/ 100, /*threshold2*/ 200);
08
09 cv::inpaint(src, mask, dst, /*inpaintRadius*/ 3, cv::INPAINT_TELEA);
10
11 std::vector<cv::Mat> images;
12 cv::Ptr<Stitcher> stitcher = cv::Stitcher::create(cv::Sticher::PANORAMA);
13 stitcher->stitch(images, dst);
```

BGR2Gray

```cpp
#include "algo.hpp"

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;
    }
}

void bgr2gray_u8(const cv::Mat& src, cv::Mat& dst) {
```

Reference implementation: 4.02ms @ 1920x1080

BGR2Gray

```
21      }
22  }
23
24  void bgr2gray_u8(const cv::Mat& src, cv::Mat& dst) {
25      const int w = src.cols;
26      const int h = src.rows;
27      dst.create(h, w, CV_8UC1);
28
29      const uint8_t* src_data = src.ptr<uint8_t>();
30      uint8_t* dst_data = dst.ptr<uint8_t>();
31      for (int y = 0; y < h; ++y) {
32          for (int x = 0; x < w; ++x) {
33              uint8_t b = src_data[x * 3];
34              uint8_t g = src_data[x * 3 + 1];
35              uint8_t r = src_data[x * 3 + 2];
36              dst_data[x] = (29 * b + 150 * g + 77 * r) >> 8;
37          }
38          dst_data += w;
39          src_data += w * 3;
40      }
41  }
42
43  // https://docs.opencv.org/master/d7/dff/tutorial_how_to_use_OpenCV_parallel_for_.html
44  void bgr2gray_u8_parallel(const cv::Mat& src, cv::Mat& dst) {
```

Fixed point: 2.39ms @ 1920x1080 (x1.68)

BGR2Gray

```
41 }
42
43 // https://docs.opencv.org/master/d7/dff/tutorial_how_to_use_OpenCV_parallel_for_.html
44 void bgr2gray_u8_parallel(const cv::Mat& src, cv::Mat& dst) {
45     const int w = src.cols;
46     const int h = src.rows;
47     dst.create(h, w, CV_8UC1);
48
49     parallel_for_(cv::Range(0, h), [&](const cv::Range& range) {
50         const uint8_t* src_data = src.ptr<uint8_t>(range.start);
51         uint8_t* dst_data = dst.ptr<uint8_t>(range.start);
52         for (int y = range.start; y < range.end; ++y) {
53             for (int x = 0; x < w; ++x) {
54                 uint8_t b = src_data[x * 3];
55                 uint8_t g = src_data[x * 3 + 1];
56                 uint8_t r = src_data[x * 3 + 2];
57                 dst_data[x] = (29 * b + 150 * g + 77 * r) >> 8;
58             }
59             dst_data += w;
60             src_data += w * 3;
61         }
62     });
63 }
```

Parallel implementation: 1.83ms @ 1920x1080 (x2.19)

OpenCV `parallel_for_`

Different backend depends on compilation options and target OS

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

Regression tests

One of OpenCV modules is named **`ts`**. It sonsists of

- Google Test based testing infrastructure
- OpenCV related extensions for regression and performance tests
- Python scripts for tests analysis

Regression test example

- Use **TEST** macro to define non-parametrized test
- Use **EXPECT_*** checks for numerical and logical tests

```
01 #include <opencv2/ts.hpp>
02
03 TEST(bgr2gray, u8)
04 {
05     cv::Mat src(10, 11, CV_8UC3), ref, dst;
06     randu(src, 0, 255);
07
08     bgr2gray_reference(src, ref);
09     bgr2gray_u8(src, dst);
10
11     double maxV;
12     minMaxLoc(abs(ref - dst), 0, &maxV);
13     EXPECT_LE(maxV, 1);
14 }
```

```
01 $ ./bin/test_algo --gtest_filter=bgr2gray.u8
02
03 [==========] Running 1 test from 1 test case.
04 [----------] Global test environment set-up.
05 [----------] 1 test from bgr2gray
06 [ RUN      ] bgr2gray.u8
07 [       OK ] bgr2gray.u8 (15 ms)
08 [----------] 1 test from bgr2gray (18 ms total)
09
10 [----------] Global test environment tear-down
11 [==========] 1 test from 1 test case ran. (21 ms total)
12 [  PASSED  ] 1 test
```

Parametrized regression test example

- Define a list of parameters and instantiate test with their combinations

```
01 typedef TestWithParam<tuple<int, int> > bgr2gray;
02 TEST_P(bgr2gray, parallel)
03 {
04     Mat src(/*rows*/ get<0>(GetParam()), /*cols*/ get<1>(GetParam()), CV_8UC3), ref, dst;
05     randu(src, 0, 255);
06
07     bgr2gray_u8(src, ref);
08     bgr2gray_u8_parallel(src, dst);
09
10     EXPECT_EQ(countNonZero(ref != dst), 0);
11 }
12 INSTANTIATE_TEST_CASE_P(/**/, bgr2gray, Combine( Values(3, 4), Values(2, 5) ));
```

```
01 [==========] Running 4 tests from 1 test case.
02 [----------] Global test environment set-up.
03 [----------] 4 tests from bgr2gray
04 [ RUN      ] bgr2gray.parallel/0, where GetParam() = (3, 2)
05 [       OK ] bgr2gray.parallel/0 (14 ms)
06 [ RUN      ] bgr2gray.parallel/1, where GetParam() = (3, 5)
07 [       OK ] bgr2gray.parallel/1 (0 ms)
08 [ RUN      ] bgr2gray.parallel/2, where GetParam() = (4, 2)
09 [       OK ] bgr2gray.parallel/2 (0 ms)
10 [ RUN      ] bgr2gray.parallel/3, where GetParam() = (4, 5)
11 [       OK ] bgr2gray.parallel/3 (0 ms)
12 [----------] 4 tests from bgr2gray (24 ms total)
13
14 [----------] Global test environment tear-down
15 [==========] 4 tests from 1 test case ran. (27 ms total)
16 [  PASSED  ] 4 tests.
```

Performance tests

- Use **PERF_TEST** to define performance test
- Wrap target code to a block **PERF_SAMPLE_BEGIN() - PERF_SAMPLE_END()**
- OpenCV does as much iterations as it's needed to have stable metrics

```
01 PERF_TEST(bgr2gray, u8_parallel)
02 {
03     cv::Mat src(480, 640, CV_8UC3), dst;
04
05     PERF_SAMPLE_BEGIN()
06         bgr2gray_u8_parallel(src, dst);
07     PERF_SAMPLE_END()
08
09     SANITY_CHECK_NOTHING();
10 }
```

```
01 $ ./bin/perf_algo --gtest_filter=bgr2gray.u8_parallel
02
03 [==========] Running 1 test from 1 test case.
04 [----------] Global test environment set-up.
05 [----------] 1 test from bgr2gray
06 [ RUN      ] bgr2gray.u8_parallel
07 [ PERFSTAT ]    (samples=100   mean=0.25   median=0.25   min=0.22   stddev=0.02 (9.5%))
08 [       OK ] bgr2gray.u8_parallel (28 ms)
09 [----------] 1 test from bgr2gray (29 ms total)
10
11 [----------] Global test environment tear-down
12 [==========] 1 test from 1 test case ran. (31 ms total)
13 [  PASSED  ] 1 test.
```

Universal intrinsics

Set of vectorized instructions that turn to platform specific operations at compile time

- AVX / SSE / SIMD (x86)
- NEON (ARM)
- VSX (PowerPC)
- MSA (MIPS)
- WASM (JavaScript)

```
01 #include <opencv2/core/hal/intrin.hpp>
02 // ...
03 std::vector<int> data = {1, 2, 3, 4, 5, 6, 7, 8};
04
05 cv::v_int32x4 twos = cv::v_setall_s32(2);
06
07 cv::v_int32x4 b0 = cv::v_load(&data[0]);
08 b0 *= twos;
09 v_store(&data[0], b0);
10
11 b0 = cv::v_load(&data[4]);
12 b0 -= twos;
13 v_store(&data[4], b0);
14 // data = {2, 4, 6, 8, 3, 4, 5, 6}
```

Example: edge detector

1. Sobel operator

```
      | -1  0  +1 |              | -1  -2  -1 |
Gx = | -2  0  +2 | * A,   Gy = |  0   0   0 | * A
      | -1  0  +1 |              | +1  +2  +1 |

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

1. Prewitt operator

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

1. Roberts cross

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

Prewitt operator implementation

```
01 #include "algo.hpp"
02
03 //      | -1  0  +1 |
04 // Gx = | -1  0  +1 | * A
05 //      | -1  0  +1 |
06 void prewitt_x(const Mat& src, Mat& dst) {
07     CV_Assert(src.type() == CV_8UC1);
08     Mat bsrc;
09     copyMakeBorder(src, bsrc, 1, 1, 1, 1, BORDER_REPLICATE);
10     dst.create(src.size(), CV_8UC1);
11     for (int y = 0; y < dst.rows; ++y)
12         for (int x = 0; x < dst.cols; ++x) {
13             dst.at<uchar>(y, x) = bsrc.at<uchar>(y    , x + 2) - bsrc.at<uchar>(y    , x) +
14                                   bsrc.at<uchar>(y + 1, x + 2) - bsrc.at<uchar>(y + 1, x) +
15                                   bsrc.at<uchar>(y + 2, x + 2) - bsrc.at<uchar>(y + 2, x);
16         }
17 }
18
19 void prewitt_x_parallel(const Mat& src, Mat& dst) {
20     Mat bsrc;
21     copyMakeBorder(src, bsrc, 1, 1, 1, 1, BORDER_REPLICATE);
22     dst.create(src.size(), CV_8UC1);
23     parallel_for_(Range(0, src.rows), [&](const Range& range) {
```

Reference implementation: 12.76ms @ 1920x1080

Prewitt operator implementation

```
14                                  bsrc.at<uchar>(y + 1, x + 2) - bsrc.at<uchar>(y + 1, x) +
15                                  bsrc.at<uchar>(y + 2, x + 2) - bsrc.at<uchar>(y + 2, x);
16          }
17  }
18
19  void prewitt_x_parallel(const Mat& src, Mat& dst) {
20      Mat bsrc;
21      copyMakeBorder(src, bsrc, 1, 1, 1, 1, BORDER_REPLICATE);
22      dst.create(src.size(), CV_8UC1);
23      parallel_for_(Range(0, src.rows), [&](const Range& range) {
24        for (int y = range.start; y < range.end; ++y)
25            for (int x = 0; x < dst.cols; ++x) {
26                dst.at<uchar>(y, x) = bsrc.at<uchar>(y    , x + 2) - bsrc.at<uchar>(y    , x) +
27                                      bsrc.at<uchar>(y + 1, x + 2) - bsrc.at<uchar>(y + 1, x) +
28                                      bsrc.at<uchar>(y + 2, x + 2) - bsrc.at<uchar>(y + 2, x);
29            }
30      });
31  }
32
33  #include <opencv2/core/hal/intrin.hpp>
34
35  void prewitt_x_parallel_vec(const Mat& src, Mat& dst) {
36      Mat bsrc;
```

Parallel implementation: 9.83ms @ 1920x1080 (x1.29)

Prewitt operator implementation

```
37      copyMakeBorder(src, bsrc, 1, 1, 1, 1, BORDER_REPLICATE);
38      dst.create(src.size(), CV_8UC1);
39      parallel_for_(Range(0, src.rows), [&](const Range& range) {
40          for (int y = range.start; y < range.end; ++y) {
41              const uint8_t* psrc0 = bsrc.ptr(y);
42              const uint8_t* psrc1 = bsrc.ptr(y + 1);
43              const uint8_t* psrc2 = bsrc.ptr(y + 2);
44              uint8_t* pdst = dst.ptr(y);
45              int x = 0;
46              for (; x <= dst.cols - v_uint8::nlanes; x += v_uint8::nlanes) {
47                  v_uint8 res = vx_load(psrc0 + x + 2) - vx_load(psrc0 + x) +
48                                vx_load(psrc1 + x + 2) - vx_load(psrc1 + x) +
49                                vx_load(psrc2 + x + 2) - vx_load(psrc2 + x);
50                  v_store(pdst + x, res);
51              }
52              for (; x < dst.cols; ++x) {
53                  pdst[x] = psrc0[x + 2] - psrc0[x] +
54                            psrc1[x + 2] - psrc1[x] +
55                            psrc2[x + 2] - psrc2[x];
56              }
57          }
58      });
59  }
60
```

Parallel vectorized implementation: 2.57ms @ 1920x1080 (x4.96)

Prewitt operator implementation

```
63      copyMakeBorder(src, bsrc, 1, 1, 1, 1, BORDER_REPLICATE);
64      dst.create(src.size(), CV_8UC1);
65      parallel_for_(Range(0, src.rows), [&](const Range& range) {
66          for (int y = range.start; y < range.end; ++y) {
67              const uint8_t* psrc0 = bsrc.ptr(y);
68              const uint8_t* psrc1 = bsrc.ptr(y + 1);
69              const uint8_t* psrc2 = bsrc.ptr(y + 2);
70              uint8_t* pdst = dst.ptr(y);
71              int x = 0;
72              for (; x <= dst.cols - v_uint8::nlanes; x += v_uint8::nlanes) {
73                  v_uint8 res = v_add_wrap(v_sub_wrap(vx_load(psrc0 + x + 2), vx_load(psrc0 + x)),
74                                v_add_wrap(v_sub_wrap(vx_load(psrc1 + x + 2), vx_load(psrc1 + x)),
75                                           v_sub_wrap(vx_load(psrc2 + x + 2), vx_load(psrc2 + x))));
76                  v_store(pdst + x, res);
77              }
78              for (; x < dst.cols; ++x) {
79                  pdst[x] = psrc0[x + 2] - psrc0[x] +
80                            psrc1[x + 2] - psrc1[x] +
81                            psrc2[x + 2] - psrc2[x];
82              }
83          }
84      });
85  }
86
```

Parallel vectorized implementation: 2.61ms @ 1920x1080 (x4.88)

Prewitt operator implementation

```
92          int y = range.start;
93          for (; y <= range.end - 2; ++y) {
94              const uint8_t* psrc0 = bsrc.ptr(y);
95              const uint8_t* psrc1 = bsrc.ptr(y + 1);
96              const uint8_t* psrc2 = bsrc.ptr(y + 2);
97              const uint8_t* psrc3 = bsrc.ptr(y + 3);
98              uint8_t* pdst0 = dst.ptr(y);
99              uint8_t* pdst1 = dst.ptr(y+1);
100              int x = 0;
101              for (; x <= dst.cols - v_uint8::nlanes; x += v_uint8::nlanes) {
102                  v_uint8 res = v_add_wrap(v_sub_wrap(vx_load(psrc1 + x + 2), vx_load(psrc1 + x)),
103                                           v_sub_wrap(vx_load(psrc2 + x + 2), vx_load(psrc2 + x)));
104                  v_store(pdst0 + x, v_add_wrap(res, v_sub_wrap(vx_load(psrc0 + x + 2),
105                                                                vx_load(psrc0 + x))));
106                  v_store(pdst1 + x, v_add_wrap(res, v_sub_wrap(vx_load(psrc3 + x + 2),
107                                                                vx_load(psrc3 + x))));
108              }
109              for (; x < dst.cols; ++x) {
110                  uint8_t res = psrc1[x + 2] - psrc1[x] + psrc2[x + 2] - psrc2[x];
111                  pdst0[x] = res + psrc0[x + 2] - psrc0[x];
112                  pdst1[x] = res + psrc3[x + 2] - psrc3[x];
113              }
114          }
```

Parallel vectorized implementation: 2.54ms @ 1920x1080 (x5.02)

Prewitt operator implementation

```
110                 uint8_t res = psrc1[x + 2] - psrc1[x] + psrc2[x + 2] - psrc2[x];
111                 pdst0[x] = res + psrc0[x + 2] - psrc0[x];
112                 pdst1[x] = res + psrc3[x + 2] - psrc3[x];
113             }
114         }
115         const uint8_t* psrc0 = bsrc.ptr(y);
116         const uint8_t* psrc1 = bsrc.ptr(y + 1);
117         const uint8_t* psrc2 = bsrc.ptr(y + 2);
118         uint8_t* pdst = dst.ptr(y);
119         int x = 0;
120         for (; x <= dst.cols - v_uint8::nlanes; x += v_uint8::nlanes) {
121             v_uint8 res = v_add_wrap(v_sub_wrap(vx_load(psrc0 + x + 2), vx_load(psrc0 + x)),
122                           v_add_wrap(v_sub_wrap(vx_load(psrc1 + x + 2), vx_load(psrc1 + x)),
123                                      v_sub_wrap(vx_load(psrc2 + x + 2), vx_load(psrc2 + x))));
124             v_store(pdst + x, res);
125         }
126         for (; x < dst.cols; ++x) {
127             pdst[x] = psrc0[x + 2] - psrc0[x] +
128                       psrc1[x + 2] - psrc1[x] +
129                       psrc2[x + 2] - psrc2[x];
130         }
131     });
132 }
```

Parallel vectorized implementation: 2.54ms @ 1920x1080 (x5.02)

Practice / Homework

- Go to https://github.com/dkurt/cv_winter_camp_2020 for slides and project source code
- Implement Roberts Cross operator:

```
input:  cv::Mat (single channel, uint8_t)
output: cv::Mat (single channel, uint32_t)

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

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

- Write regression tests
- Parallelize and vectorize algorithm
- Write performance test and compare efficieny against reference version