Основы работы с матрицами в OpenCV (C++ интерфейс)

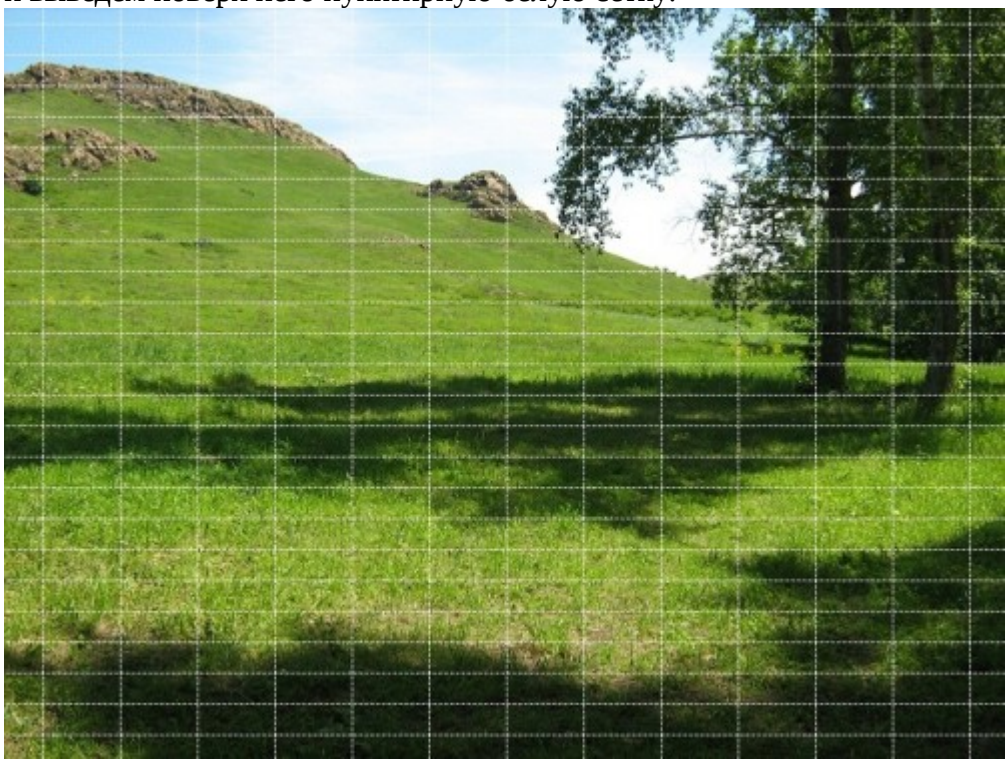
- OpenCV

Чтобы разрабатывать собственные алгоритмы компьютерного зрения, необходимо уметь получать доступ к пикселям. Изображение представляет собой двумерную матрицу, которая представлена в виде класса cv::Mat. Каждый элемент матрицы представляет собой один пиксель. Для изображений в градациях серого элемент матрицы представлен 8-битным числом без знака (от 0 до 255). Для цветного изображения в формате RGB таких чисел 3, по одному на каждую компоненту цвета. Формат класса cv::Mat следующий.

```
class CV_EXPORTS Mat
{
public:
// ... много методов ...
...
/*! включает в себя несколько битовых полей:
- сигнатура
- флаг непрерывности
- глубина
- количество каналов
*/
int flags;
//! размерность массива, >= 2
int dims;
//! количество строк и столбцов или (-1, -1)
int rows, cols;
//! указатель на данные
uchar* data;
//! указатель на счетчик ссылок; когда массив указан
// на выделенные пользователем данные, то указатель равен NULL
int* refcount;
// другие члены
...
};
```

Данные храниться могут в разных форматах, и это не только 1 и 3 байтные элементы матрицы. Чтобы показать, как осуществляется запись пикселей, возьмем для примера цветное изображение:

и выведем поверх него пунктирную белую сетку:



Код примера:

```
Mat img = imread( "image01.jpg"); // Открытие файла
// Отобразить на изображении белую пунктирную сетку
for( int i = 0; i < img.rows; i++ )
        for( int j = 0; j < img.cols; j++ )
                if ( ( i % 20 == 10 && j % 2 == 1 ) ||
                         ( j % 50 == 25 && i % 2 == 1 ) )
                {
                        img.at<Vec3b>(i,j)[0]= 255;
                        img.at<Vec3b>(i,j)[1]= 255;
                        img.at<Vec3b>(i,j)[2]= 255;
                }
```

```
        imwrite( "image01_res.jpg", img );
```

Цикл перебирает строки изображения (i) и столбцы (j). Условие вывода на изображения белого пикселя: если остаток от деления на 20 номера строки равен 10 и столбец нечетный, если остаток от деления на 50 номера столбца равен 25 и строка нечетная. Здесь видно, что для вывода белого пикселя закрашивается каждый компонент. Если бы изображение было 8 битным 1 канальным, т.е. градации серого, тогда можно было бы ограничиться одной строкой:

```
img.at<uchar>(i,j)= 255;
```

Очевидно, что если есть возможность записывать пиксели, то можно и считывать информацию о них. В следующем примере показано, что каждый пиксель анализируется на принадлежность зеленому цвету – уровень G компоненты должен быть более 64, а уровни B и R меньше компоненты G. Если пиксель принадлежит зеленому цвету, то он заменяется на ярко красный цвет.

```
        Mat img = imread( "image01.jpg"); // Открытие файла
        for( int i = 0; i < img.rows; i++ )
                for( int j = 0; j < img.cols; j++ )
                        if ( img.at<Vec3b>(i,j)[0] < img.at<Vec3b>(i,j)[1] - 10
&&
                                img.at<Vec3b>(i,j)[2] < img.at<Vec3b>(i,j)[1] -
10 &&
                                img.at<Vec3b>(i,j)[1] > 64 )
                        {
                               img.at<Vec3b>(i,j)[0]= 0;
                               img.at<Vec3b>(i,j)[1]= 0;
                               img.at<Vec3b>(i,j)[2]= 255;
                        }
        imwrite( "image01_res2.jpg", img );
```

Использование метода at класса cv::Mat может быть громоздким, поэтому есть более простой способ доступа к изображению:

```
cv::Mat_<uchar> img2 = img; // Ссылка на img
img2(10,20) = 0; // Доступ к строке 10 и столбцу 20
```

Естественно, что одними такими примитивными операциями не обойтись при разработке какого-либо серьезного программного продукта, поэтому необходимо знать существующие операции над матрицами.
Не всегда матрицы создаются при открытии изображения из файла или видеопотока. Очень часто приходится создавать изображения самостоятельно для этого в классе cv::Mat различные конструкторы, например, следующие.

```
Mat::Mat(int rows, int cols, int type);
Mat::Mat(Size size, int type);
```

Здесь Size – это класс из пространства имен cv, которым можно задавать размеры матрицы, например, так:

```
Mat mat( Size( 200, 200 ), CV_8UC1 );
```

CV_8UC1 – означает тип изображения: 8-битное 1 канальное unsigned char. Типы матриц определяются по следующей формуле:
CV_<S|U|F>CГде bit depth характеризует количество бит на глубину. S=signed, U=unsigned, F=float. A number of channels – количество каналов.
Удалять изображение не надо, поскольку в конце части программы, где определена матрица, будет вызван деструктор для Mat (когда счетчик ссылок на матрицу становится равным нулю). Важно помнить, что следующая строка

```
Mat mat2 = mat;
```

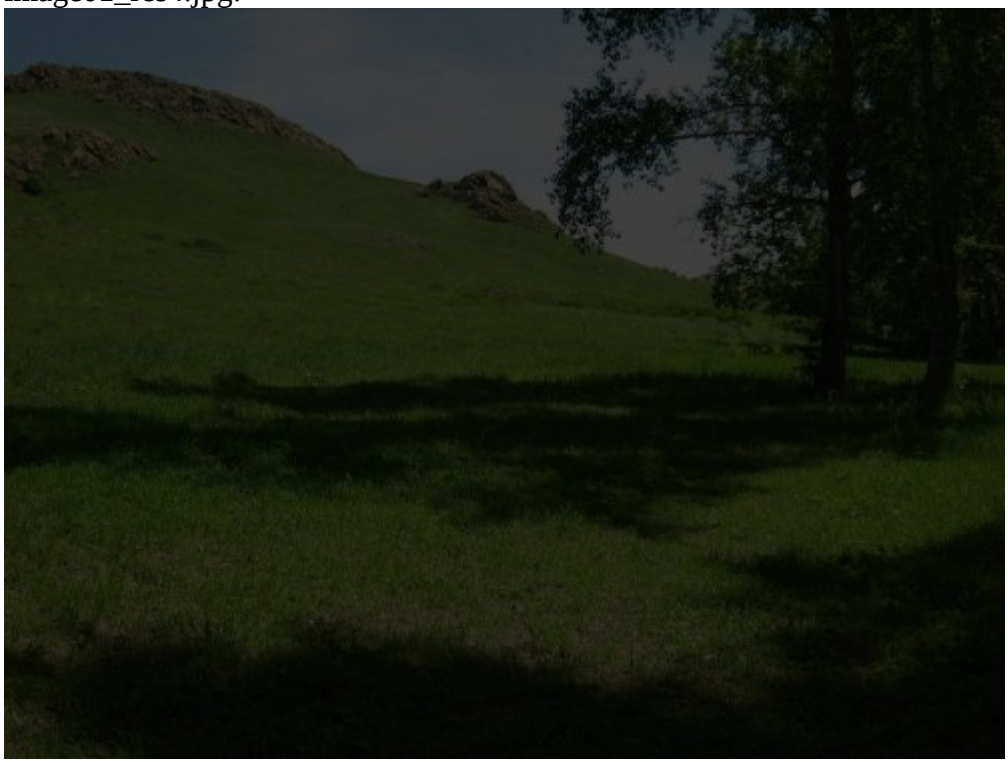
не осуществляет копирование матрицы, а лишь увеличивает счетчик ссылок на матрицу. Для того, чтобы матрицу скопировать нужно вызвать метод copyTo (следующий листинг). Над матрицами можно осуществлять действия без вызова функций, а именно: проводить математические операции матрицы с числом и матрицы с матрицей. Далее приведен пример математических операций над матрицами.

```
Mat img = imread( "image01.jpg"); // Открытие файла

Mat img2;
img.copyTo( img2 );
img2 = img2 * 2; // Увеличение яркости в 2 раза
imwrite( "image01_res3.jpg", img2 );
Mat img3;
img.copyTo( img3 );
img3 = img3 * 0.25; // Уменьшение яркости в 4 раза
imwrite( "image01_res4.jpg", img3 );
```

image01_res3.jpg:

image01_res4.jpg:

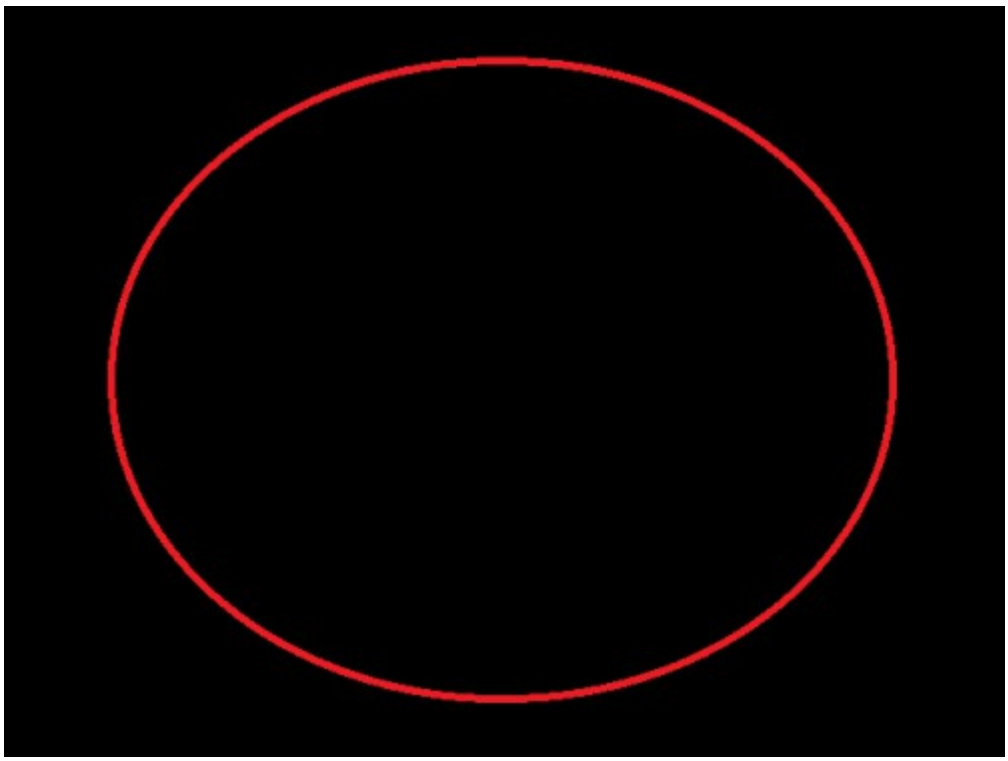


Если необходимо объединить изображения, то это очень просто. Пример представлен ниже:

```
Mat img = imread( "image01.jpg");
Mat img2 = imread( "image02.png");

img2 = img2 + img; // Сложение матриц
imwrite( "image01_res5.jpg", img2 );
```

Объединяется самое первое изображение со следующим:



Результат:

Самый естественный вопрос, который может возникнуть после рассуждений об операциях над матрицами – можно ли те же операции применить для части изображения? Да, можно. Это называется регионом интереса (ROI). Задается регион с помощью класса Rect, основное назначение которого – это поддержка данных о размерах региона: x, y, width, height. В следующем листинге приведен пример копирования региона в отдельное изображение и работы с регионом внутри изображения.

```
Mat img = imread( "image01.jpg");
Rect r( 100, 50, 200, 150 ); // Создание региона
Mat img2;
// Копирование региона в отдельное изображение
img( r ).copyTo( img2 );
```

```
img2 += CV_RGB(0,0,255); // Добавляем синего к изображению

// Изменение части изображения
img( r ) *= 0.5; // Понижение яркости
imwrite( "image01_res6.jpg", img2 );
imwrite( "image01_res7.jpg", img );
```

image01_res6.jpg:

image01_res7.jpg:

Это далеко не все операции над матрицами, поэтому далее приведены возможные операции с матрицами.

Список операций над матрицами, которые могут быть использованы в произвольно сложных выражениях (A, B – матрицы, s – скаляр, alpha – вещественный скаляр):

• сложение, вычитание, отрицание: A+B, A-B, A+s, A-s, s+A, s-A, -A;
• масштабирование: A*alpha;
• поэлементное умножение/деление: A.mul(B), A/B, alpha/A;
• умножение матриц: A*B;
• транспонированная матрица: A.t();
• обращение матрицы и псевдо-инверсии, решения линейных систем и наименьших квадратов: A.inv([метод]), A.inv([метод])*B;
• сравнивание: A op B, A op alpha, alpha op A, где op это одно из следующих: >, >=, ==, !=, <=, <. Результат сравнения в одноканальной 8-битной матрице;
• битовые логические операции: A op B, A op s, s op A, ~A, где op это одно из следующих: &, |, ^;
• нахождение минимума/максимума: min(A, B), min(A, alpha), max(A, B), max(A, alpha);
• abs(A);
• векторное и скалярное произведение: A.cross(B) A.dot(B);
• любая функция матрицы или матриц, которая возвращает скаляр, например, norm, mean, sum, countNonZero, trace, determinant, repeat.

Помимо этих операций существует ряд функций, которые в том или ином виде предназначены для обработки матриц. Одна из наиболее часто востребованных функций предназначена для конвертации матриц.

```
void cvtColor(
        InputArray src,
        OutputArray dst,
        int code,
        int dstCn=0
);
```

Параметры:
src – входная матрица;
dst – выходная матрица, размер матрицы должен быть таким же, как и в src;
code – код конвертации;
dstCn – количество каналов в конечной матрице, если указано 0, то количество каналов определяется автоматически.
Данная функция преобразует изображение (двумерная матрица) из одного цветового пространства в другое.
Внимание: Цветовой формат в OpenCV по умолчанию не RGB, а BGR!

Обычные диапазоны для R, G и B компонент:
• 0 — 255 для CV_8U изображений,
• 0 — 65535 для CV_16U изображений,
• 0 — 1 для CV_32F изображений.
В случае линейных преобразований, диапазон не имеет значения. Но в случае нелинейной трансформации, входное RGB изображение должно быть нормализовано для надлежащего диапазона значений, чтобы получить правильные результаты (см. документацию).
Функция поддерживает следующие трансформации:
• RGB в GRAY и обратно – CV_BGR2GRAY, CV_RGB2GRAY, CV_GRAY2BGR, CV_GRAY2RGB;
• RGB в CIE XYZ.Rec 709 (и обратно) – CV_BGR2XYZ, CV_RGB2XYZ, CV_XYZ2BGR, CV_XYZ2RGB;
• RGB в YCrCb JPEG (YCC) (и обратно) – CV_BGR2YCrCb, CV_RGB2YCrCb, CV_YCrCb2BGR, CV_YCrCb2RGB;
• RGB в HSV (и обратно) – CV_BGR2HSV, CV_RGB2HSV, CV_HSV2BGR, CV_HSV2RGB;
• RGB в HLS (и обратно) – CV_BGR2HLS, CV_RGB2HLS, CV_HLS2BGR, CV_HLS2RGB;
• RGB в CIE L*a*b* (и обратно) – CV_BGR2Lab, CV_RGB2Lab, CV_Lab2BGR, CV_Lab2RGB;
• RGB в CIE L*u*v* (и обратно) – CV_BGR2Luv, CV_RGB2Luv, CV_Luv2BGR, CV_Luv2RGB;
• Bayer в RGB – CV_BayerBG2BGR, CV_BayerGB2BGR, CV_BayerRG2BGR, CV_BayerGR2BGR, CV_BayerBG2RGB, CV_BayerGB2RGB, CV_BayerRG2RGB, CV_BayerGR2RGB.
Пример вызова:

```
cvtColor( Image24, Gray, CV_BGR2GRAY);
```

Здесь показан перевод из RGB 24-битного изображения в 8-битное градаций серого.

Поделитесь с друзьями
ВКонтактеОдноклассникиTwitterFacebookМой МирLiveJournalGoogle PlusЯндекс