Ralf Wondratschek | adorsys GmbH & Co. KG

Bildbearbeitung für Android mit OpenCV

Inhalt

OpenCV API

Pixel Zugriff

Effektbeispiele

Inhalt

Einbindung in Android

Kamerazugriff bei Android

Best Practices

```
cv::Scalar provideCode() {
    return cv::Scalar(CPP, Java);
}

void showDemo(cv::Mat& mat) {
    Android(mat);
    iOS(mat);
    Windows(mat);
}
```

OpenCV API

Open Source Computer Vision

BSD Lizenz

→ kommerzielle Produkte erlaubt

C++, C, Python und JAVA API

Fokus auf Performance und Echtzeitanwendungen

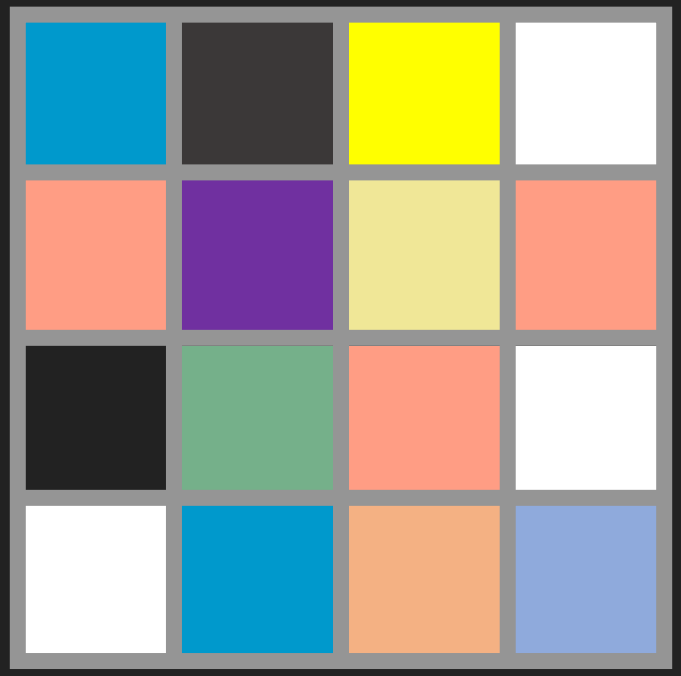
Entwicklung begann 1999 durch Intel

Version 1.0 erschien 2006

Version 2.0 mit C++ API erschien 2009

Seit 2012 durch OpenCV.org gepflegt und verwaltet

class **Mat**



class **Mat**

Repräsentiert ein n-dimensionales Array

Bei Bildern 2-dimensional
→ rows und cols Variablen

Überladene Operatoren
→ A+B, A-B, A*alpha, A*B, …

class **Mat**

Anzahl der Kanäle und Bittiefe frei wählbar
→ z.B. CV_8UC1, CV_8UC3, CV_32F

Konvertierung mittels cvtColor(src, dst, code) Funktion

Standardformat ist BGR und nicht RGB

Mat bgr(4, 4, CV_8UC3)

(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
(0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)

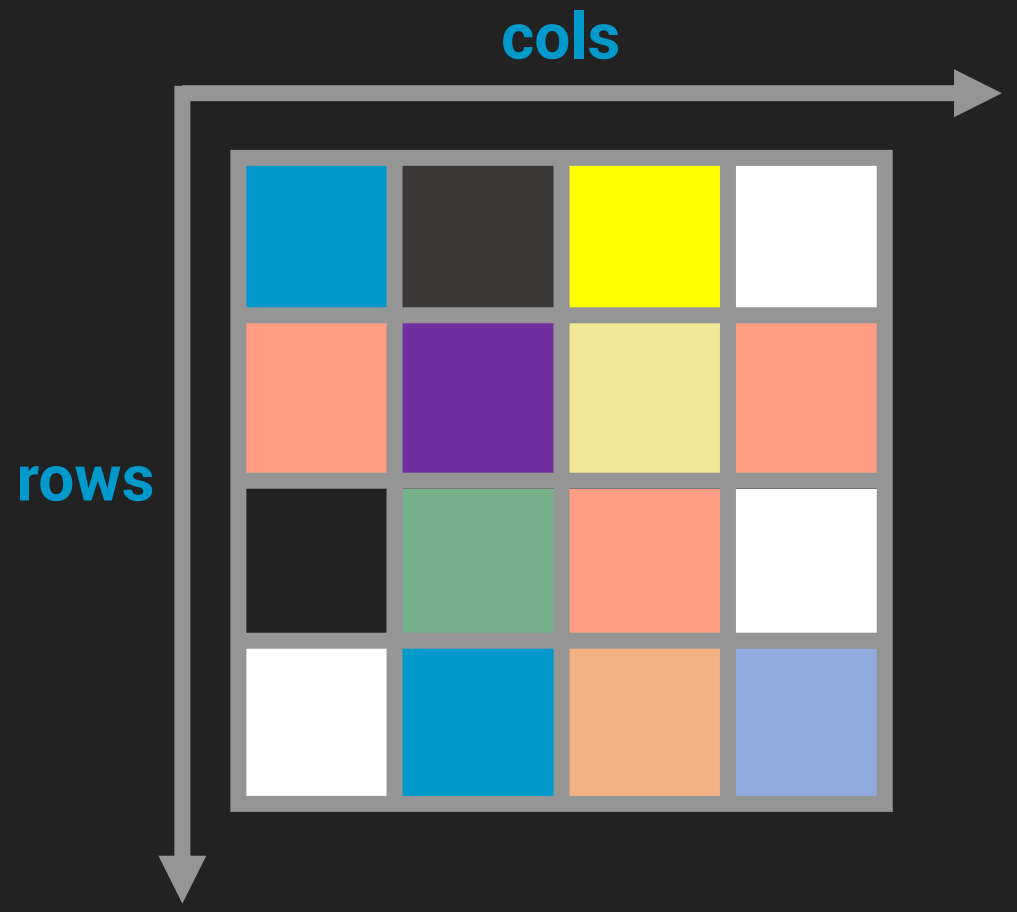
class **Mat_**

Template Klasse für Mat

Sinnvoll, wenn man zu Compile-Zeit den Typ des Mat Objektes kennt

```
Mat mat(24, 24, CV_8U);
…
Mat_<uchar>& other = (Mat_<uchar>&) mat;
```

class **Size**



class **Size**

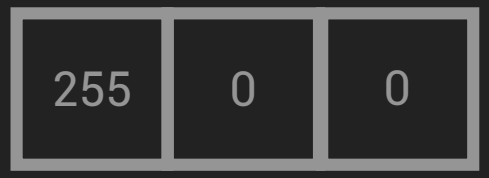
Gibt die Anzahl der Spalten und Reihen an
→ Mat::size()

Größe auch über cols und rows abfragbar

Bei höher dimensionalen Matrizen sind Werte (-1, -1)

class **Vec**

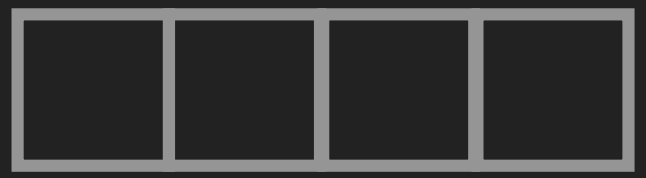
Klasse für kurze numerische Vektoren
→ Vec2b, Vec4i, Vec6d



Überladene Operatoren
→ v1+v2, v1-v2, v1*scale, …

```
Mat_<Vec3b> mat(240, 320, Vec3b(255, 0, 0));
```

class **Scalar**



Erbt von Vec und stellt Vec<type, 4> dar

4 dimensionaler Vektor

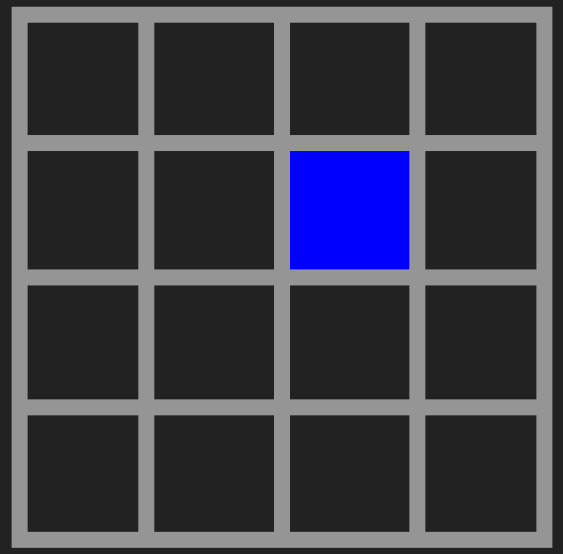
Pixel Zugriff

at(row, col)

```
Mat mat(240, 320, CV_8UC3);
mat.at<Vec3b>(row, col)[0] = 255;
mat.at<Vec3b>(row, col)[1] = 255;
mat.at<Vec3b>(row, col)[2] = 255;

Mat mat(240, 320, CV_8UC1);
mat.at<uchar>(row, col) = 255;
```

at(row, col)

```
Mat mat(4, 4, CV_8UC3);
mat.at<Vec3b>(1, 2)[0] = 255;
```



at(row, col)

Typ der Matrix muss zur Compile Zeit bekannt sein

Gut lesbar

Sinnvoll bei einzelnen Pixelzugriffen

Langsam beim Iterieren über das gesamte Bild

Iterator

```
Mat_<Vec3b>::iterator it = mat.begin<Vec3b>();
Mat_<Vec3b>::iterator itend= mat.end<Vec3b>();

for ( ; it != itend; ++it) {
      (*it)[0] = 255;
      (*it)[1] = 255;
      (*it)[2] = 255;
}
```

Iterator

Typ der Matrix muss zur Compile Zeit bekannt sein

Gut lesbar, objektorientiert

Sinnvoll, wenn die Bearbeitungszeit nicht ausschlaggebend ist

Langsam beim Iterieren über das gesamte Bild (aber schneller als at() Methode)

Pointer

```
int nr = mat.rows;
int nc = mat.cols;

if (mat.isContinuous()) {
        nc = nc * nr;
        nr = 1;
}

for (int row = 0; row < nr; ++row) {
        uchar* pointer = mat.ptr<uchar>(row);

        for (int col = 0; col < nc; ++col) {
                *pointer++ = 255; // B
                *pointer++ = 255; // G
                *pointer++ = 255; // R
        }
}
```

Pointer

Kann als Template Methode verwendet werden

Pointerarithmetik fehleranfällig

Effizient, Vorteile bei Speicherstruktur können genutzt werden

Sinnvoll beim Iterieren über das gesamte Bild

Beispiele

Allgemeines Vorgehen

1.) App lädt plattformspezifisch das Bild und stellt es dar

2.) Aus dem Bild wird ein Mat Objekt im BGR Format erstellt

3.) Im geteilten Code wird das Mat Objekt in-place bearbeitet

Schwarz-Weiß

```
void OpenCvDemo::convertGray (cv::Mat& mat) {
    cv::Mat gray(mat.rows, mat.cols, CV_8UC1);

    cvtColor(mat, gray, CV_BGR2GRAY);
    cvtColor(gray, mat, CV_GRAY2BGR);
}
```

Farbcontroller

```
void OpenCvDemo::controlColor(cv::Mat& mat, int r, int g, int b) {
        float red = (float) r / 100;
        float green = (float) g / 100;
        float blue = (float) b / 100;

        int nr = mat.rows;
        int nc = mat.cols;

        if (mat.isContinuous()) {
                nc = nc * nr;
                nr = 1;
        }

        for (int row = 0; row < nr; ++row) {
                uchar* pOriginal = mMatOriginal.ptr(row);
                uchar* pResult = mat.ptr(row);

                for (int col = 0; col < nc; ++col) {
                        *pResult++ = (uchar) (*pOriginal++ * blue);  // B
                        *pResult++ = (uchar) (*pOriginal++ * green); // G
                        *pResult++ = (uchar) (*pOriginal++ * red);   // R
                }
        }
}
```

Touchfocus

```
void OpenCvDemo::touchFocus(cv::Mat& matOverlay, cv::Mat& matResult, int centerX, int centerY, int fade, int distance) {
        int edge = distance + fade;

        int nr = matResult.rows;
        int nc = matResult.cols;

        for (int row = 0; row < nr; ++row) {
                uchar* pOriginal = mMatOriginal.ptr(row);
                uchar* pOverlay = matOverlay.ptr(row);
                uchar* pResult = matResult.ptr(row);

                for (int col = 0; col < nc; ++col) {
                        float calcDistance = (abs(centerY - row) + abs(centerX - col));
                        float opacity;

                        if (calcDistance < distance) {
                                opacity = 1;
                        } else if (calcDistance > edge) {
                                opacity = 0;
                        } else {
                                opacity = (fade - (calcDistance - distance)) / fade;
                        }

                        *pResult++ = (uchar) ((*pOverlay++ - *pOriginal) * opacity + *pOriginal++); // B
                        *pResult++ = (uchar) ((*pOverlay++ - *pOriginal) * opacity + *pOriginal++); // G
                        *pResult++ = (uchar) ((*pOverlay++ - *pOriginal) * opacity + *pOriginal++); // R
                }
        }
}
```

Zugriff auf Nachbarpixel

Möglich über weitere Pointer auf vorhergehende Reihe und Spalte

Sonderbehandlung von erster und letzter Reihe und Spalte

Funktioniert, ist aber fehleranfällig und schwer zu warten

Kernel Matrix

Zentrum referenziert momentanen Pixel

Pixelwerte werden mit Kernelwert multipliziert und anschließend addiert

Kernel Matrix

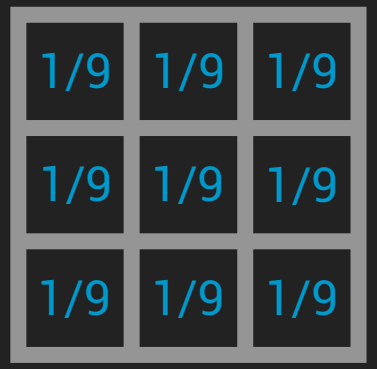
$$125 * 5 - 68 - 48 - 196 - 127 = 186$$

96	68	33		
127	125	48		
146	196	142		

0	-1	0
-1	5	-1
0	-1	0

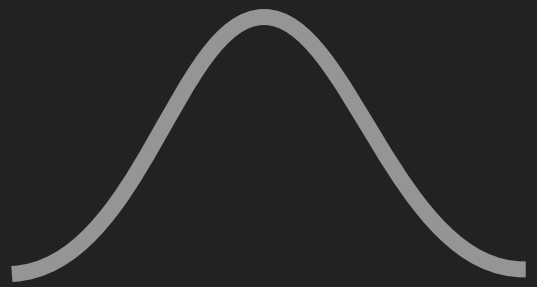
96	68	33		
127	186	48		
146	196	142		

Box-Blur




```
void OpenCvDemo::boxBlur(cv::Mat& mat, int size) {
        float kernelValue = 1.0 / (size * size);

        cv::Mat kernel(size, size, CV_32F, cv::Scalar(kernelValue));
        filter2D(mMatOriginal, mat, mMatOriginal.depth(), kernel);
}
```

Gaussian-Blur



```
void OpenCvDemo::gaussianBlur(cv::Mat& mat, int size) {
        if (size % 2 == 0) {
                size++;
        }

        // sigma in both directions computed
        GaussianBlur(mMatOriginal, mat, cv::Size(size, size), 0, 0);

}
```

Auflösung verringern

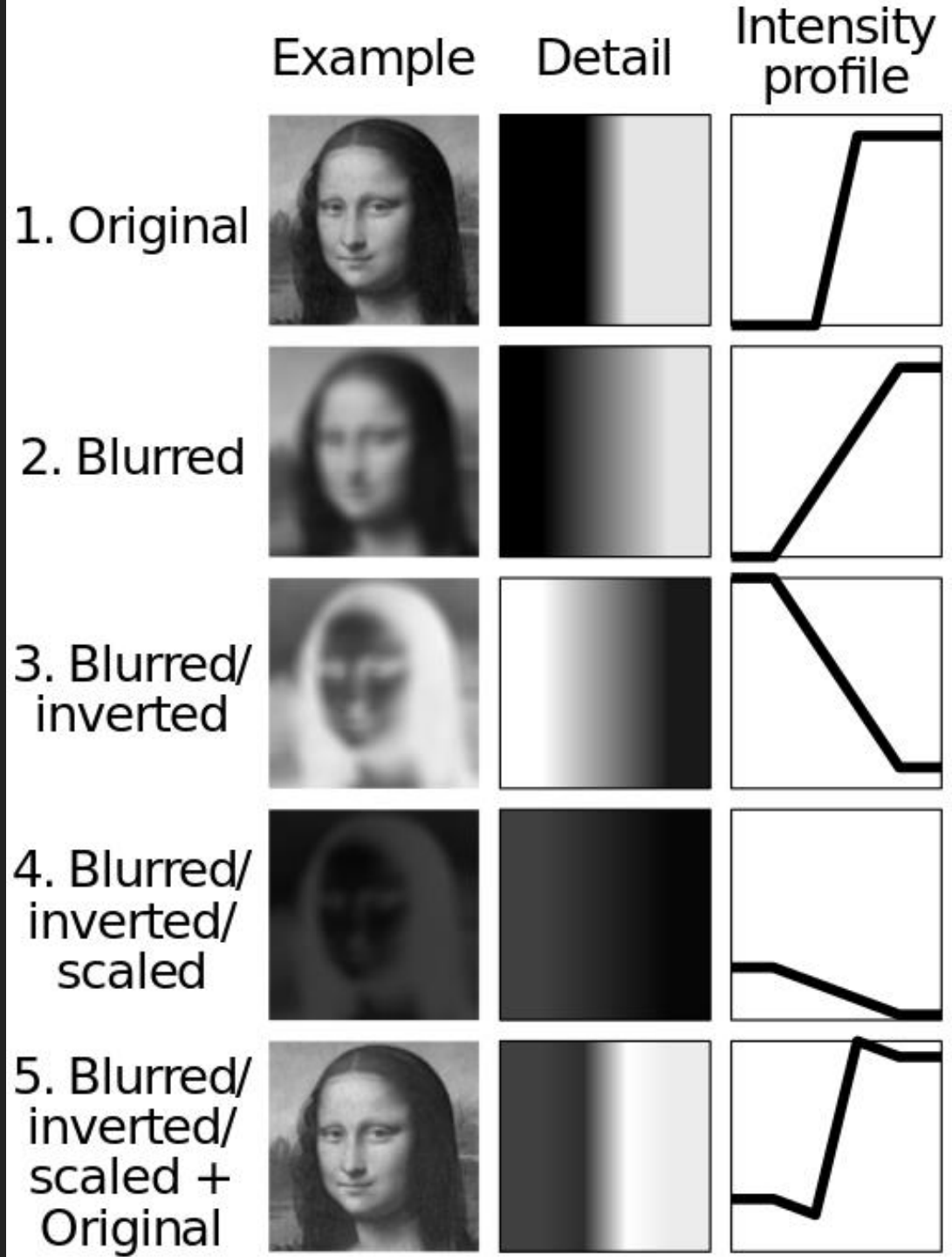
```
void OpenCvDemo::sizeDown(cv::Mat& mat, int step, bool bad) {
        if (bad) {
                int nr = mat.rows;
                int nc = mat.cols;

                for (int row = 0; row < nr; ++row) {
                        uchar* pOriginal = mMatOriginal.ptr(row * step);
                        uchar* pResult = mat.ptr(row);

                        for (int col = 0; col < nc; ++col) {
                                *pResult++ = *pOriginal++; // B
                                *pResult++ = *pOriginal++; // G
                                *pResult++ = *pOriginal++; // R
                                pOriginal += 3 * (step - 1);
                        }
                }
        } else {
                resize(mMatOriginal, mat, cv::Size(mMatOriginal.cols / step, mMatOriginal.rows / step));
        }
}
```

Arithmetik

```
void OpenCvDemo::arithmetic(cv::Mat& mat, int factor) {
      float f = (float) factor / 10;
      mat = mMatOriginal + (f * mMatOriginal);
}
```

Unsharp Masking

http://en.wikipedia.org/wiki/Unsharp_masking



Unsharp Masking

```
void OpenCvDemo::sharpen(cv::Mat& mat, int size) {
      gaussianBlur(mat, size);
      addWeighted(mMatOriginal, 1.5, mat, -0.5, 0, mat);
}
```

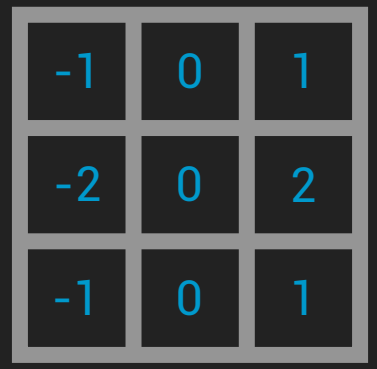
Kantendetektion

Gesucht ist der Gradient der Matrix I

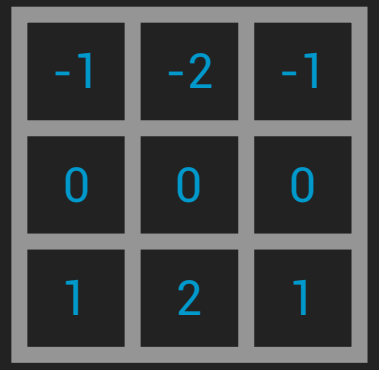
$$grad(I) = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]^T$$

$$G = \sqrt{G_x^2 + G_y^2}$$

Sobel

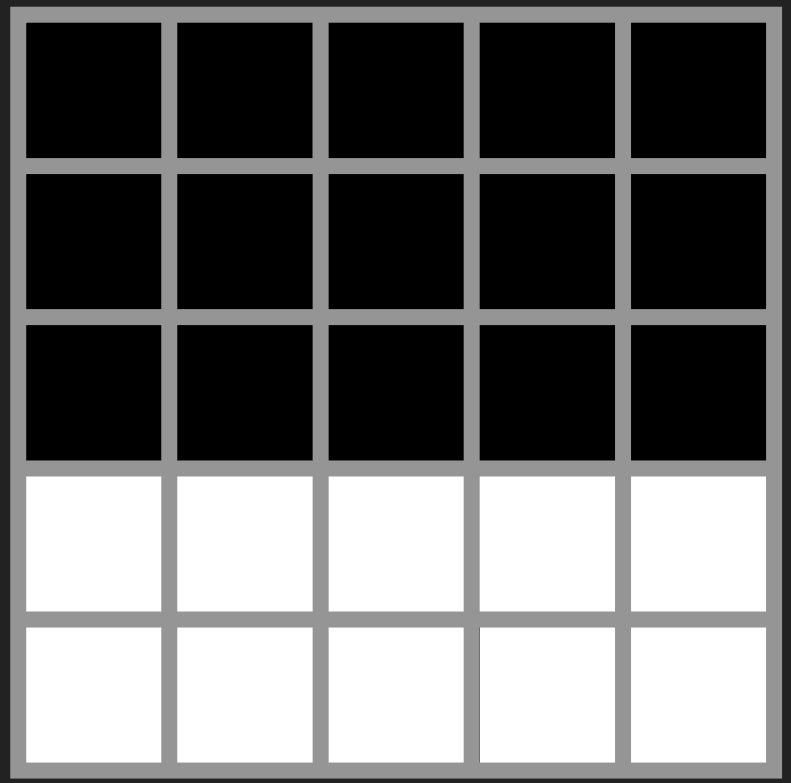


Gradient in X-Richtung



Gradient in Y-Richtung

Sobel Beispiel



0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
255	255	255	255	255
255	255	255	255	255

Sobel Beispiel

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
255	255	255	255	255
255	255	255	255	255

-1	-2	-1
0	0	0
1	2	1

0	0	0	0	0
0	0	0	0	0
1020	1020	1020	1020	1020
1020	1020	1020	1020	1020
0	0	0	0	0

Sobel

```
void OpenCvDemo::sobel(cv::Mat& mat) {
        cv::Mat originalGray(mMatOriginal.cols, mMatOriginal.rows, CV_8UC1);
        cvtColor(mMatOriginal, originalGray, CV_BGR2GRAY);

        cv::Mat gradX;
        cv::Mat gradY;
        cv::Mat absGradX;
        cv::Mat absGradY;
        cv::Mat grad;

        Sobel(originalGray, gradX, CV_16S, 1, 0);
        Sobel(originalGray, gradY, CV_16S, 0, 1);

        convertScaleAbs(gradX, absGradX);
        convertScaleAbs(gradY, absGradY);

        addWeighted(absGradX, 0.5, absGradY, 0.5, 0, grad);

        cvtColor(grad, mat, CV_GRAY2BGR);
}
```

Einbindung Android

Anforderungen

OpenCV Java API

OpenCV C++ API

NDK

Warum wir die vorgefertigten Klassen und Java API nicht verwenden

Java API nicht mit allen Plattformen nutzbar

Java API ist nur ein Wrapper

Flexibilität

Warum wir die vorgefertigten Klassen und Java API nicht verwenden

Eclipse nicht erwünscht

Gradle als Build System

Gradle Task zum Bauen von C++ Code

OpenCV einbinden

```
repositories {
        maven {
                url 'https://raw.github.com/vRallev/mvn-repo/master/'
        }
}

dependencies {
        compile 'org.opencv:opencv-android:2.4.8'
}
```

Gradle und das NDK

Rudimentäre Unterstützung

Makefile wird generiert, keine manuelle Anpassung möglich

Automatischer Build mit jni Ordner

→ Umweg über eigenen Build Task

https://code.google.com/p/android/issues/detail?id=65241

OpenCV initialisieren

```
static {
        if (OpenCVLoader.initDebug()) {
                System.loadLibrary("cv-module");
        } else {
                throw new IllegalArgumentException();
        }
}
```

Szene laden

```
ImageView imageView = (ImageView) findViewById(R.id.imageView);
Bitmap bitmap = createBitmap();

imageView.setImageBitmap(bitmap);

int[] data = new int[bitmap.getWidth() * bitmap.getHeight];
// übergib Array per JNI und manipuliere Daten

bitmap.setPixels(data, 0, width, 0, 0, width, height);
```

Schecht!

```
ImageView imageView = (ImageView) findViewById(R.id.imageView);
Bitmap bitmap = createBitmap();

imageView.setImageBitmap(bitmap);

int[] data = new int[bitmap.getWidth() * bitmap.getHeight];
// übergib Array per JNI und manipuliere Daten

bitmap.setPixels(data, 0, width, 0, 0, width, height);
```

Bitmap manipulieren

```
BitmapConverter::BitmapConverter (JNIEnv* env, jobject& bitmap) throw (int) {
        AndroidBitmapInfo info;
        int ret;
        void* pixels;

        if ((ret = AndroidBitmap_getInfo(env, bitmap, &info)) < 0) {
                LOGE("AndroidBitmap_getInfo() failed ! error=%d", ret);
                throw ret;
        }
        if (info.format != ANDROID_BITMAP_FORMAT_RGBA_8888) {
                LOGE("Bitmap format is not RGBA_8888 !");
                throw ret;
        }
        if ((ret = AndroidBitmap_lockPixels(env, bitmap, &pixels)) < 0) {
                LOGE("AndroidBitmap_lockPixels() failed ! error=%d", ret);
                throw ret;
        }

        width = info.width;
        height = info.height;

        Mat bitmapMat(height, width, CV_8UC4, pixels);
        mat8UC4 = bitmapMat;
        Mat matBGR(height, width, CV_8UC3);
        mat = matBGR;

        cvtColor(mat8UC4, mat, CV_RGBA2BGR);
}
```

Bitmap manipulieren

```
void BitmapConverter::release (JNIEnv* env, jobject& bitmap) {
        cvtColor(mat, mat8UC4, CV_BGR2RGBA);
        AndroidBitmap_unlockPixels(env, bitmap);
}
```

Bitmap manipulieren

```
ImageView imageView = (ImageView) findViewById(R.id.imageView);
Bitmap bitmap = createBitmap();

imageView.setImageBitmap(bitmap);
// übergib bitmap per JNI und manipuliere Daten

imageView.invalidate();
```

Kamerazugriff Android

Vorgehen

Kameraverbindung herstellen
- SurfaceView oder TextureView
- Parameter setzen
- Kamera öffnen

Byte Array als Datenpuffer erstellen

PreviewCallback setzen

Vorgehen

Zeichnen Daten in Bitmap, die von einer ImageView angezeigt wird

YUV420 muss in RGBA umgewandelt werden

Beispiel

```
void jni(JNIEnv* env, jobject, jbyteArray yuv, jobject bitmap) {
        BitmapConverter converterResult (env, bitmap);

        int width = converterResult.width;
        int height = converterResult.height;

        jbyte* _yuv = env->GetByteArrayElements(yuv, 0);
        Mat myuv = Mat(height + height/2, width, CV_8UC1, (uchar *)_yuv);

        cvtColor(myuv, converterResult.mat, CV_YUV420sp2BGR);

        circle(converterResult.mat, Point(width / 2, height / 2),
                        width / 4, Scalar(255, 0, 0), 20);

        env->ReleaseByteArrayElements(yuv, _yuv, 0);
        converterResult.release(env, bitmap);
}
```

Best Practices

Best Practices

Trennung von Plattform und OpenCV Code

Matrizen eignen sich gut für Parallelisierung

Verarbeitung in Workerthreads

Best Practices

Bei Analyse erst kleinere Bilder verwenden

Bei Typumwandlung Speicherstrukturen ausnutzen:

→ YUV420sp zu Gray ist günstig
→ BGR zu BGRA ist günstig
→ YUV420sp zu BGR ist teuer

Best Practices

Optimierung bei Arithmetik nutzen, falls Funktion zeitkritisch ist

API nutzen, falls Funktionen schon fertig programmiert sind

Ralf Wondratschek

rwo@adorsys.de

adorsys