Programmation Avancée

OpenCV

Daniel Felipe González Obando
dgonzale@pasteur.fr

3 Decembre 2018

(Support de Bertrand Cannelle)

OpenCV

- license BSD
  - gratuit pour la recherche et l'éducation
  - gratuit pour un usage commercial

OpenCV

- license BSD
  - gratuit pour la recherche et l'éducation
  - gratuit pour un usage commercial
- interface possible pour :
  - C++
  - C
  - Python
  - Java

OpenCV

- license BSD
  - gratuit pour la recherche et l'éducation
  - gratuit pour un usage commercial
- interface possible pour :
  - C++
  - C
  - Python
  - Java
- multi-plateforme
  - Windows
  - Linux
  - Mac OS
  - iOS
  - Android

OpenCV

- license BSD
  - gratuit pour la recherche et l'éducation
  - gratuit pour un usage commercial
- interface possible pour :
  - C++
  - C
  - Python
  - Java
- multi-plateforme
  - Windows
  - Linux
  - Mac OS
  - iOS
  - Android
- écrit en C++

OpenCV

- license BSD
  - gratuit pour la recherche et l'éducation
  - gratuit pour un usage commercial
- interface possible pour :
  - C++
  - C
  - Python
  - Java
- multi-plateforme
  - Windows
  - Linux
  - Mac OS
  - iOS
  - Android
- écrit en C++
- multi-processeur

OpenCV a une structure modulaire :

core structures de bases (point, matrice...) et des fonctions de base utilisées par les autres modules

OpenCV a une structure modulaire :

core structures de bases (point, matrice...) et des fonctions de base utilisées par les autres modules

imgproc contient des fonctions de filtrage, transformation d'image, changement d'espace de couleur, calcul d'histogramme

OpenCV a une structure modulaire :

core structures de bases (point, matrice...) et des fonctions de base utilisées par les autres modules

imgproc contient des fonctions de filtrage, transformation d'image, changement d'espace de couleur, calcul d'histogramme

video module qui contient des fonctions d'estimation de mouvement, du tracking d'objet...

OpenCV a une structure modulaire :

core structures de bases (point, matrice...) et des fonctions de base utilisées par les autres modules

imgproc contient des fonctions de filtrage, transformation d'image, changement d'espace de couleur, calcul d'histogramme

video module qui contient des fonctions d'estimation de mouvement, du tracking d'objet...

calib3d module de calibration de caméras, d'estimation de pose, de reconstruction

OpenCV a une structure modulaire :

core structures de bases (point, matrice...) et des fonctions de base utilisées par les autres modules

imgproc contient des fonctions de filtrage, transformation d'image, changement d'espace de couleur, calcul d'histogramme

video module qui contient des fonctions d'estimation de mouvement, du tracking d'objet...

calib3d module de calibration de caméras, d'estimation de pose, de reconstruction

features2d module d'extraction de caractéristiques et de mise en correspondance

OpenCV a une structure modulaire :

core structures de bases (point, matrice...) et des fonctions de base utilisées par les autres modules

imgproc contient des fonctions de filtrage, transformation d'image, changement d'espace de couleur, calcul d'histogramme

video module qui contient des fonctions d'estimation de mouvement, du tracking d'objet...

calib3d module de calibration de caméras, d'estimation de pose, de reconstruction

features2d module d'extraction de caractéristiques et de mise en correspondance

objdetect détection d'objets (visage, yeux...)

OpenCV a une structure modulaire :

core structures de bases (point, matrice...) et des fonctions de base utilisées par les autres modules

imgproc contient des fonctions de filtrage, transformation d'image, changement d'espace de couleur, calcul d'histogramme

video module qui contient des fonctions d'estimation de mouvement, du tracking d'objet...

calib3d module de calibration de caméras, d'estimation de pose, de reconstruction

features2d module d'extraction de caractéristiques et de mise en correspondance

objdetect détection d'objets (visage, yeux...)

highgui IHM

OpenCV a une structure modulaire :

core structures de bases (point, matrice...) et des fonctions de base utilisées par les autres modules

imgproc contient des fonctions de filtrage, transformation d'image, changement d'espace de couleur, calcul d'histogramme

video module qui contient des fonctions d'estimation de mouvement, du tracking d'objet...

calib3d module de calibration de caméras, d'estimation de pose, de reconstruction

features2d module d'extraction de caractéristiques et de mise en correspondance

objdetect détection d'objets (visage, yeux...)

highgui IHM

gpu module d'algorithm GPU

Conventions

- toutes les classes et fonctions sont placées dans le namespace cv::
- il faut donc utiliser cv::MaFonction

Type de données :

- 8 bit unsigned integer (uchar) CV_8U
- 8 bit signed integer (schar) CV_8S
- 16 bit unsigned integer (ushort) CV_16U
- 16 bit signed integer (short) CV_16S
- 32 bit signed integer (int) CV_32S
- 32 bit floating-point number (float) CV_32F
- 64 bit floating-point number (double) CV_64F

Type d'image multicanaux :

- 8 bit unsigned integer 1 canaux CV_8UC1 ou CV_8U
- 8 bit unsigned integer 2 canaux CV_8UC2
- 8 bit unsigned integer 3 canaux CV_8UC3
- 8 bit unsigned integer 4 canaux CV_8UC4
- 8 bit unsigned integer n canaux $CV\_MAKETYPE(CV\_8U, n)$
- (The number of bits per item)(Signed or Unsigned)(Type Prefix)C(The channel number)

```
// matrice 3x3 de float
Mat mtx(3, 3, CV_32F);
// matrice 10*1 a 2 canaux en double
Mat cmtx(10, 1, CV_64FC2);
// image 1920*1080 sur 3 canaux en unsigned char
Mat img(Size(1920, 1080), CV_8UC3);
```

Gestion des erreurs

```
 try
{
    ... // call OpenCV
}
catch( cv::Exception& e )
{
    const char* err_msg = e.what();
    std::cout << "exception caught: " << err_msg <<
   std::endl;
}
```

Structures de base

Point 2D

```
typedef cv::Point_<int> cv::Point2i;
typedef cv::Point2i cv::Point;
typedef cv::Point_<float> cv::Point2f;
typedef cv::Point_<double> cv::Point2d;

  cv::Point2f a(0.3f, 0.f), b(0.f, 0.4f);
  cv::Point2f ptA = (a + b)*2.5;
  std::cout << ptA.x << ", " << ptA.y << std::endl;
  cv::Point   ptB = (a + b)*2.5;
  std::cout << ptB.x << ", " << ptB.y << std::endl;
```

```
0.75, 1
1, 1
```

Point 3D

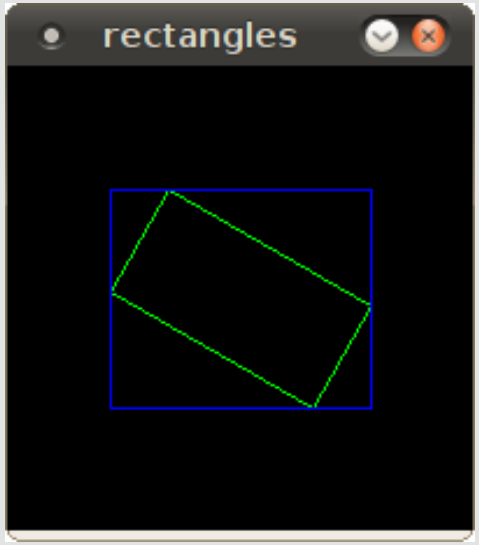
```
typedef cv::Point3_<int> cv::Point3i;
typedef cv::Point3_<float> cv::Point3f;
typedef cv::Point3_<double> cv::Point3d;

cv::Point3f a(0.3f, 0.f, 2.f), b(0.f, 0.4f, 4.f);
cv::Point3i ptA = (a + b)*5.f;
cout << ptA.x << ", " << ptA.y << ", " << ptA.z <<
    endl;
cv::Point3f ptB = (a + b)*5.f;
cout << ptB.x << ", " << ptB.y << ", " << ptB.z <<
    endl;
```

```
2, 2, 30
1.5, 2, 30
```

Rectangle

```
#include <iostream>
#include <cv.h>
#include <highgui.h>

using namespace cv;
int main ( int argc, char **argv )
{
  Mat image(200, 200, CV_8UC3, Scalar(0));
  RotatedRect rRect = RotatedRect(Point2f(100,100), Size2f(100,50), 30);

  Point2f vertices[4];
  rRect.points(vertices);
  for (int i = 0; i < 4; i++)
    line(image, vertices[i], vertices[(i+1)%4], Scalar(0,255,0));

  Rect brect = rRect.boundingRect();
  rectangle(image, brect, Scalar(255,0,0));

  imshow("rectangles", image);
  waitKey(0);
  return 0;
}
```

Rectangle



Matrice

```
typedef cv::Matx<float, 1, 2> Matx12f;
typedef cv::Matx<double, 1, 2> Matx12d;

template<typename _Tp, int m, int n> class Matx
    {...};
```

- addition, soustraction... : A+B, A-B, A+s, A-s, s+A, s-A, -A
- echelle : A*alpha
- multiplication et division par element : A.mul(B), A/B, alpha/A
- multiplication: A*B
- transposition: A.t()
- inversion et pseudo-inversion
- solveur de systeme linéaire et moindre carrés


MATIS
METHODES D'ANALYSE ET DE TRAITEMENT
D'IMAGES POUR LA STEREO-RESTITUTION

Mémoire

Gestion de la mémoire

Attention

```
// Creation d'une grosse matrice
Mat A(10000, 10000, CV_64F);
// Creation d'une autre matrice mais qui pointe vers les memes donnees
Mat B = A;
```

- Permet de réduire la taille en mémoire

Gestion de la mémoire

Solution

```
// Creation d'une grosse matrice
Mat A(10000, 10000, CV_64F);
// Creation d'une autre matrice
Mat B = A.clone();
```

- Effectue une copie complète

Entrée / Sortie

Lire

```
cv::Mat img = cv::imread("linux_icon.png");
```

Écrire

```
// fabrique une image a 3 canaux
// 1920 colonnes et 1080 lignes.
cv::Mat img(Size(1920, 1080), CV_8UC3);
cv::imwrite("c:/toto.png", img);
```


MATIS
METHODES D'ANALYSE ET DE TRAITEMENT
D'IMAGES POUR LA STEREO-RESTITUTION

Accesseurs


```
#include <iostream>
#include <cv.h>
#include <highgui.h>

using namespace cv;
int main ( int argc, char **argv ){
  Mat imgT = imread("linux_icon.png");
  for( int l = 0; l < imgT.rows; l++ ){
    for( int c = 0; c < imgT.cols; c++ ){
      std::cout<<c<<" "<<l<<" "<< (unsigned
    int)imgT.at<Vec3b>(l,c)[0]<<" "<<(unsigned
    int)imgT.at<Vec3b>(l,c)[1]<<" "<<(unsigned
    int)imgT.at<Vec3b>(l,c)[2]<<"\n";
    }
  }
  return 0;
}
```

Region

```
cv::Mat img = cv::imread("c:/toto.png");
cv::Rect r(10, 10, 100, 100);
cv::Mat smallImg = img(r);
```

Creation

```
cv::Mat M(5,3, cv::CV_8UC3, cv::Scalar(0,0,255));
std::cout << "M = " << std::endl << " " << M <<
    std::endl << std::endl;
```

```
cv::Mat M;
M.create(4,4, CV_8UC(2));
std::cout << "M = "<< std::endl << " "  << M <<
    std::endl << std::endl;
```

```
M =
[0, 0, 255, 0, 0, 255, 0, 0, 255;
0, 0, 255, 0, 0, 255, 0, 0, 255;
0, 0, 255, 0, 0, 255, 0, 0, 255;
0, 0, 255, 0, 0, 255, 0, 0, 255;
0, 0, 255, 0, 0, 255, 0, 0, 255]
```

Creation

```
cv::Mat E = cv::Mat::eye(4, 4, cv::CV_64F);
cout << "E = " << endl << " " << E << endl << endl;

cv::Mat O = cv::Mat::ones(2, 2, cv::CV_32F);
cout << "O = " << endl << " " << O << endl << endl;

cv::Mat Z = cv::Mat::zeros(3,3, cv::CV_8UC1);
cout << "Z = " << endl << " " << Z << endl << endl;
```

Creation

```
cv::Mat E = cv::Mat::eye(4, 4, cv::CV_64F);
cout << "E = " << endl << " " << E << endl << endl;

cv::Mat O = cv::Mat::ones(2, 2, cv::CV_32F);
cout << "O = " << endl << " " << O << endl << endl;

cv::Mat Z = cv::Mat::zeros(3,3, cv::CV_8UC1);
cout << "Z = " << endl << " " << Z << endl << endl;
```

```
E =
[1, 0, 0, 0;
0, 1, 0, 0;
0, 0, 1, 0;
0, 0, 0, 1]
```

Creation

```
cv::Mat E = cv::Mat::eye(4, 4, cv::CV_64F);
cout << "E = " << endl << " " << E << endl << endl;

cv::Mat O = cv::Mat::ones(2, 2, cv::CV_32F);
cout << "O = " << endl << " " << O << endl << endl;

cv::Mat Z = cv::Mat::zeros(3,3, cv::CV_8UC1);
cout << "Z = " << endl << " " << Z << endl << endl;
```

```
O =
[1, 1;
1, 1]
```

Creation

```
cv::Mat E = cv::Mat::eye(4, 4, cv::CV_64F);
cout << "E = " << endl << " " << E << endl << endl;

cv::Mat O = cv::Mat::ones(2, 2, cv::CV_32F);
cout << "O = " << endl << " " << O << endl << endl;

cv::Mat Z = cv::Mat::zeros(3,3, cv::CV_8UC1);
cout << "Z = " << endl << " " << Z << endl << endl;
```

```
Z =
[0, 0, 0;
 0, 0, 0;
 0, 0, 0]
```


MATIS
METHODES D'ANALYSE ET DE TRAITEMENT
D'IMAGES POUR LA STEREO-RESTITUTION

Dessin

Primitive

```
void circle(Mat& img, Point center, int radius, const Scalar& color, int thickness=1, int
      lineType=8, int shift=0);
bool clipLine(Size imgSize, Point& pt1, Point& pt2);
void ellipse(Mat& img, Point center, Size axes, double angle, double startAngle, double
      endAngle, const Scalar& color, int thickness=1, int lineType=8, int shift=0);
void ellipse2Poly(Point center, Size axes, int angle, int startAngle, int endAngle, int
      delta, vector<Point>& pts);
void fillConvexPoly(Mat& img, const Point* pts, int npts, const Scalar& color, int
      lineType=8, int shift=0);
void fillPoly(Mat& img, const Point** pts, const int* npts, int ncontours, const Scalar&
      color, int lineType=8, int shift=0, Point offset=Point());
Size getTextSize(const string& text, int fontFace, double fontScale, int thickness, int*
      baseLine);
void line(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int
      lineType=8, int shift=0);
void rectangle(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int
      lineType=8, int shift=0);
void polylines(Mat& img, const Point** pts, const int* npts, int ncontours, bool isClosed,
      const Scalar& color, int thickness=1, int lineType=8, int shift=0);
void putText(Mat& img, const string& text, Point org, int fontFace, double fontScale, Scalar
      color, int thickness=1, int lineType=8, bool bottomLeftOrigin=false);
```

Exemples

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;


int main( int argc, char** argv )
{
  Mat imgI = imread("linux_icon.png");
  ellipse(imgI, Point(50,50), Size(40,40), 45, 0, 270, Scalar(255,255,0), 2, CV_AA);
  imwrite("linux_iconDe.png", imgI);
  return 0;
}
```

Exemples

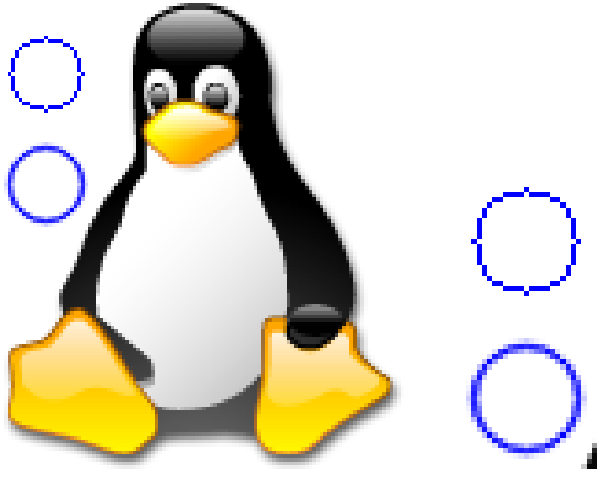
```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;


int main( int argc, char** argv )
{
  Mat imgI = imread("linux_icon.png");
  ellipse(imgI, Point(50,50), Size(40,40), 45, 0, 270, Scalar(255,255,0), 2, CV_AA);
  imwrite("linux_iconDe.png", imgI);
  return 0;
}
```

Exemples

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;


int main( int argc, char** argv )
{
  Mat imgI = imread("linux_icon.png");
  circle(imgI, Point(20,20),10, Scalar(255,0,0),1,4);
  circle(imgI, Point(20,50),10, Scalar(255,0,0),1,CV_AA);
  imwrite("linux_iconDe.png", imgI);
  return 0;
}
```

Exemples


```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;


int main( int argc, char** argv )
{
  Mat imgI = imread("linux_icon.png");
  circle(imgI, Point(20,20),10, Scalar(255,0,0),1,4);
  circle(imgI, Point(20,50),10, Scalar(255,0,0),1,CV_AA);
  imwrite("linux_iconDe.png", imgI);
  return 0;
}
```



Traitement d'image

Traitement d'image

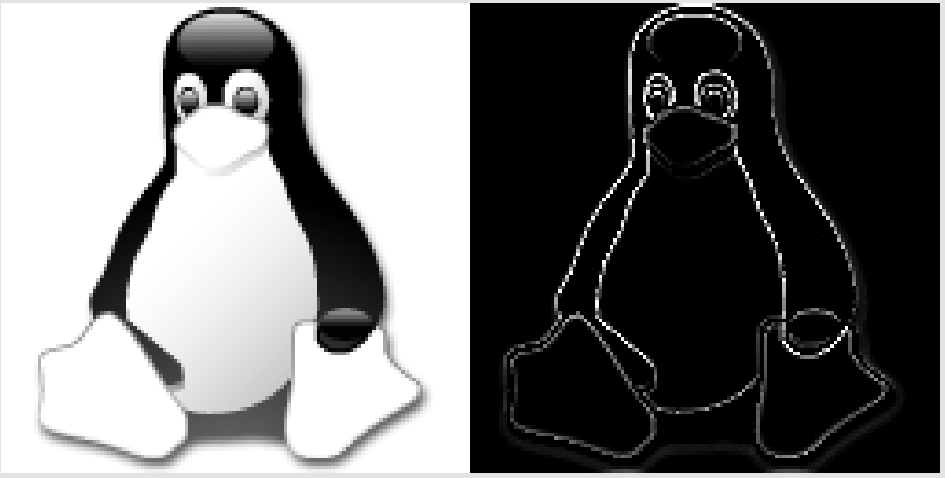
Noyau

Filtrage

```
cv::Mat imgI = cv::imread("linux_iconG.png");
cv::Mat imgO ;
cv::Mat kern = (cv::Mat_<char>(3,3) <<  0, -1,  0,
                                       -1,  4, -1,
                                        0,-1,  0);

cv::filter2D(imgI, imgO, imgI.depth(), kern );
cv::imwrite("linux_iconS.png", imgO);
```

Filtrage

```
cv::Mat imgI = cv::imread("linux_iconG.png");
cv::Mat imgO ;
cv::Mat kern = (cv::Mat_<char>(3,3) <<  0, -1,  0,
                                       -1,  4, -1,
                                        0,-1,  0);

cv::filter2D(imgI, imgO, imgI.depth(), kern );
cv::imwrite("linux_iconS.png", imgO);
```



Traitement d'image

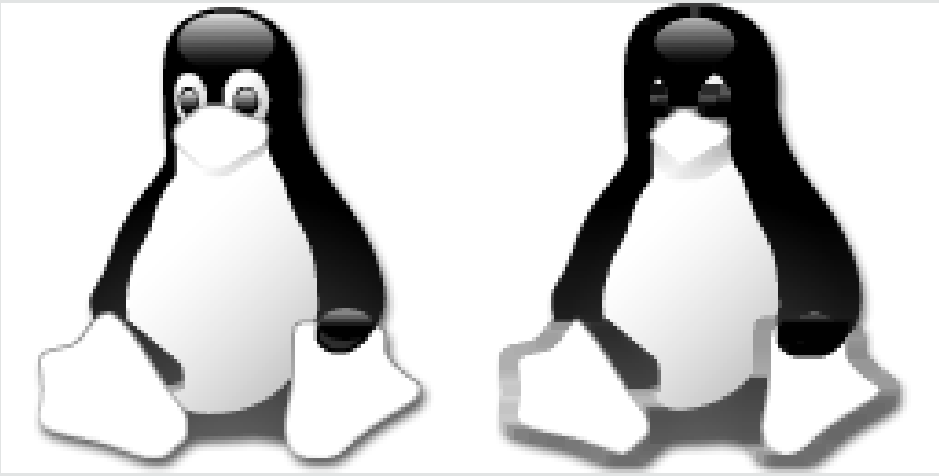
Morphologie

Erosion

```
cv::Mat imgI = cv::imread("linux_iconG.png");
cv::Mat imgO;
// MORPH_RECT - a rectangular structuring element:
// MORPH_ELLIPSE - an elliptic structuring element, that is, a filled ellipse inscribed into
      the rectangle Rect(0, 0, esize.width, 0.esize.height)
// MORPH_CROSS - a cross-shaped structuring element:

cv::Mat element = cv::getStructuringElement( MORPH_RECT,
                                             Size( 5, 3 ));

cv::erode( imgI,imgO, element );

cv::imwrite("linux_iconE.png", imgO);
```

Erosion

```
cv::Mat imgI = cv::imread("linux_iconG.png");
cv::Mat imgO;
// MORPH_RECT - a rectangular structuring element:
// MORPH_ELLIPSE - an elliptic structuring element, that is, a filled ellipse inscribed into
      the rectangle Rect(0, 0, esize.width, 0.esize.height)
// MORPH_CROSS - a cross-shaped structuring element:

cv::Mat element = cv::getStructuringElement( MORPH_RECT,
                                             Size( 5, 3 ));

cv::erode( imgI,imgO, element );

cv::imwrite("linux_iconE.png", imgO);
```

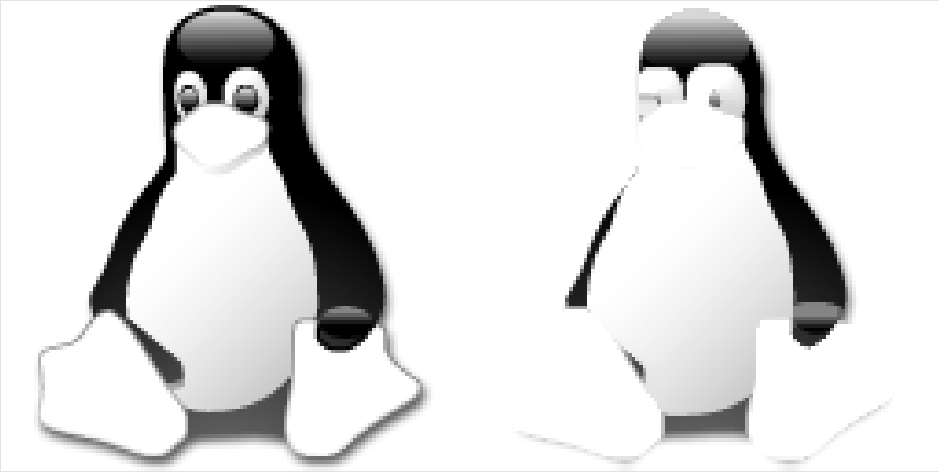


Dilatation

```
cv::Mat imgI = cv::imread("linux_iconG.png");
cv::Mat imgO;
// MORPH_RECT
// MORPH_ELLIPSE
// MORPH_CROSS

cv::Mat element = cv::getStructuringElement( MORPH_RECT,
                                             Size( 5, 3 ));

cv::dilate( imgI,imgO, element );

cv::imwrite("linux_iconD.png", imgO);
```

Dilatation

```
cv::Mat imgI = cv::imread("linux_iconG.png");
cv::Mat imgO;
// MORPH_RECT
// MORPH_ELLIPSE
// MORPH_CROSS

cv::Mat element = cv::getStructuringElement( MORPH_RECT,
                                             Size( 5, 3 ));

cv::dilate( imgI,imgO, element );

cv::imwrite("linux_iconD.png", imgO);
```



Traitement d'image

Couleurs

Attention

Une image couleurs est par défaut ouverte en BGR, c'est à dire que le canal 0 est le bleu, le 1 est le vert et le 2 le rouge !

```
#include <iostream>
#include <cv.h>
#include <highgui.h>

using namespace cv;
int main ( int argc, char **argv ){
  Mat imgT = imread("linux_icon.png");
  for( int l = 0; l < imgT.rows; l++ ){
    for( int c = 0; c < imgT.cols; c++ ){
      std::cout<<c<<" "<<l<<" "<< (unsigned
    int)imgT.at<Vec3b>(l,c)[0]<<" "<<(unsigned
    int)imgT.at<Vec3b>(l,c)[1]<<" "<<(unsigned
    int)imgT.at<Vec3b>(l,c)[2]<<"\n";
    }
  }
  return 0;
}
```

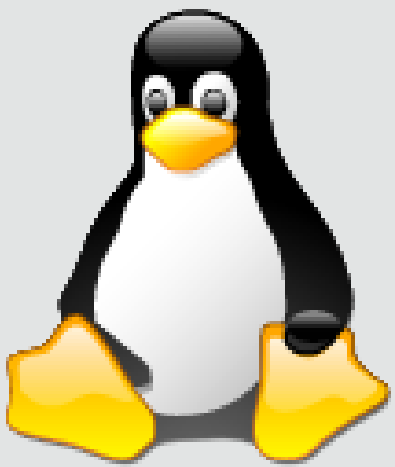
Conversion de couleur

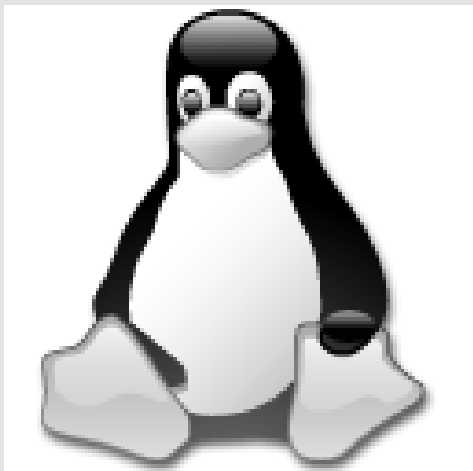
- RGB $->$ gris (CV_RGB2GRAY)
- RGB HSV (CV_BGR2HSV...)
- BGR XYZ (CV_BGR2XYZ...)
- BGR2YCrCb (CV_BGR2YCrCb...)
- BGR2HLS (CV_BGR2HLS...)
- BGR2Lab (CV_BGR2Lab...)
- BGR2Luv (CV_BGR2Luv...)
- BayerBG2BGR (...)

```
#include <cv.h>
#include <highgui.h>
using namespace cv;
int main( int argc, char** argv )
{
  Mat src, hsv;
  src=imread("linux_icon.png");
  cvtColor(src, hsv, CV_BGR2GRAY);
  imwrite("linux_iconG.png",hsv);
  return 0;
}
```

Conversion de couleur

```
#include <cv.h>
#include <highgui.h>
using namespace cv;
int main( int argc, char** argv )
{
  Mat src, hsv;
  src=imread("linux_icon.png");
  cvtColor(src, hsv, CV_BGR2GRAY);
  imwrite("linux_iconG.png",hsv);
  return 0;
}
```





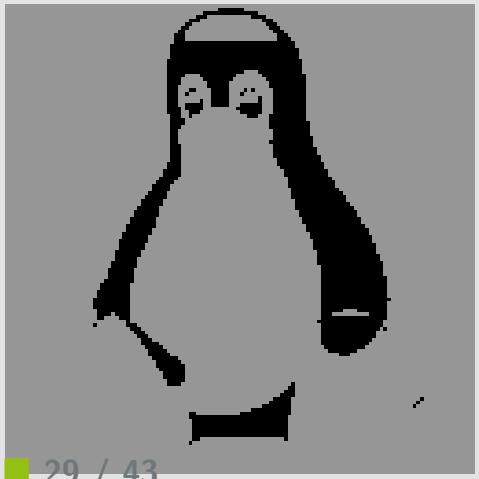
Traitement d'image

Seuillage

Seuillage automatique

- THRESH_BINARY

```
#include <cv.h>
#include <highgui.h>
#include <iostream>

using namespace cv;

int main( int argc, char** argv )
{
  Mat src = imread( "linux_icon.png" );
  Mat dst;
  threshold(src,dst, 100, 150, THRESH_BINARY);
  imwrite( "linux_icon_.png" ,dst);
 return 0;
}
```



Seuillage automatique

- THRESH_BINARY
- THRESH_BINARY_INV

```
#include <cv.h>
#include <highgui.h>
#include <iostream>

using namespace cv;

int main( int argc, char** argv )
{
  Mat src = imread( "linux_icon.png" );
  Mat dst;
  threshold(src,dst, 100, 150, THRESH_BINARY);
  imwrite( "linux_icon_.png" ,dst);
 return 0;
}
```

Seuillage automatique

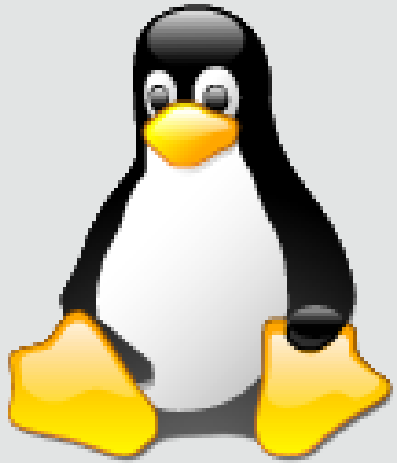
- THRESH_BINARY
- THRESH_BINARY_INV
- THRESH_TRUNC

```
#include <cv.h>
#include <highgui.h>
#include <iostream>

using namespace cv;

int main( int argc, char** argv )
{
  Mat src = imread( "linux_icon.png" );
  Mat dst;
  threshold(src,dst, 100, 150, THRESH_BINARY);
  imwrite( "linux_icon_.png" ,dst);
 return 0;
}
```


MATIS
METHODES D'ANALYSE ET DE TRAITEMENT
D'IMAGES POUR LA STEREO-RESTITUTION

IHM

Visualiser une Image



Visualiser une Image

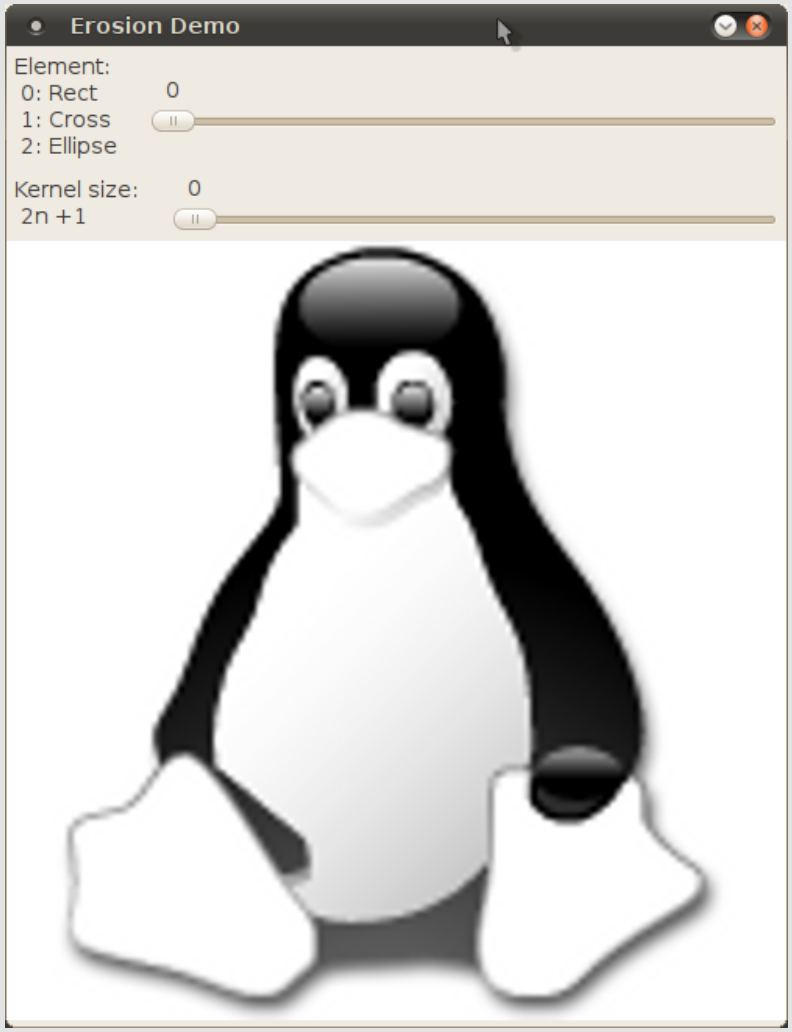
```
#include <iostream>
#include <cv.h>
#include <highgui.h>

using namespace cv;
int main ( int argc, char **argv )
{
  Mat img = cv::imread("./linux_icon.png");
  imshow("Mon image", img);
  waitKey();
  return 0;
}
```

Visualiser une Image

Ajouter une trackbar

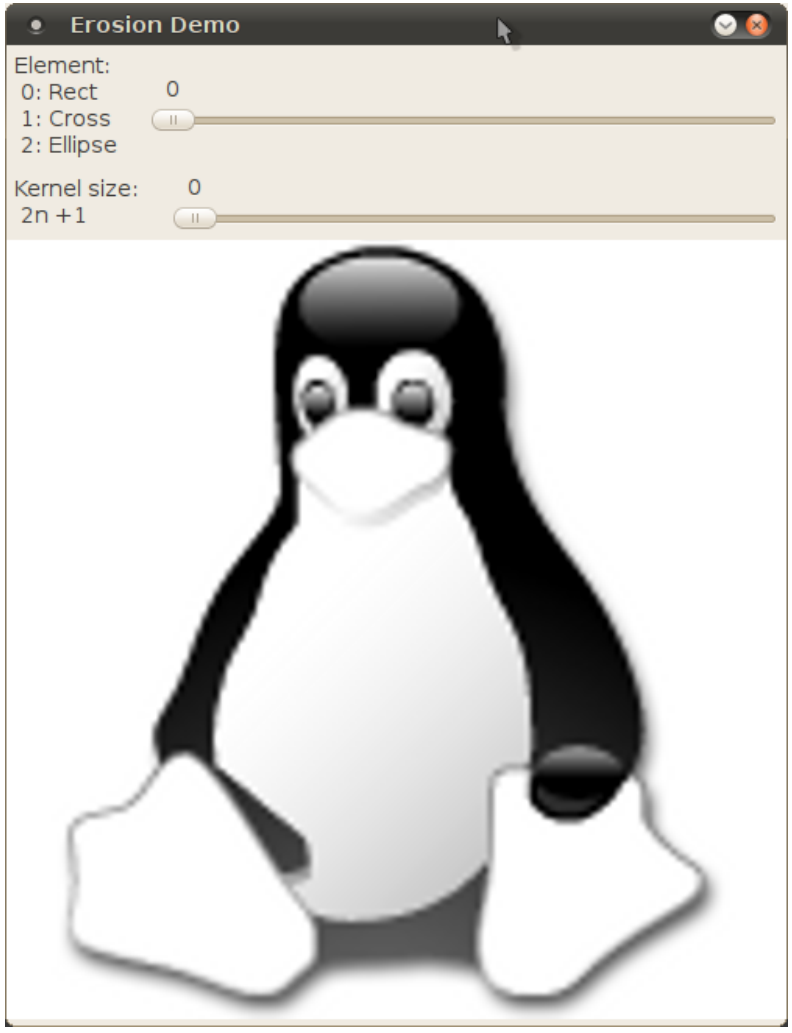

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>
using namespace cv;
Mat src, erosion_dst;
int erosion_elem = 0;int erosion_size = 0;

void Erosion( int, void* )
{
  int erosion_type;
  if( erosion_elem == 0 ){ erosion_type = MORPH_RECT; }
  else if( erosion_elem == 1 ){ erosion_type = MORPH_CROSS; }
  else if( erosion_elem == 2) { erosion_type = MORPH_ELLIPSE; }
  Mat element = getStructuringElement( erosion_type,Size( 2*erosion_size + 1,
      2*erosion_size+1 ),Point( erosion_size, erosion_size ) );
  erode( src, erosion_dst, element );
  imshow( "Erosion Demo", erosion_dst );
}

int main( int argc, char** argv )
{
  src = imread( "linux_iconGG.png" );
  if( !src.data ) return -1;
  namedWindow( "Erosion Demo", CV_WINDOW_AUTOSIZE );
  createTrackbar( "Element:\n 0: Rect \n 1: Cross \n 2: Ellipse", "Erosion
      Demo",&erosion_elem, 2,Erosion );
  createTrackbar( "Kernel size:\n 2n +1", "Erosion Demo",&erosion_size, 21, Erosion );
  Erosion( 0, 0 );
  waitKey(0);
  return 0;
}
```

Visualiser une Image



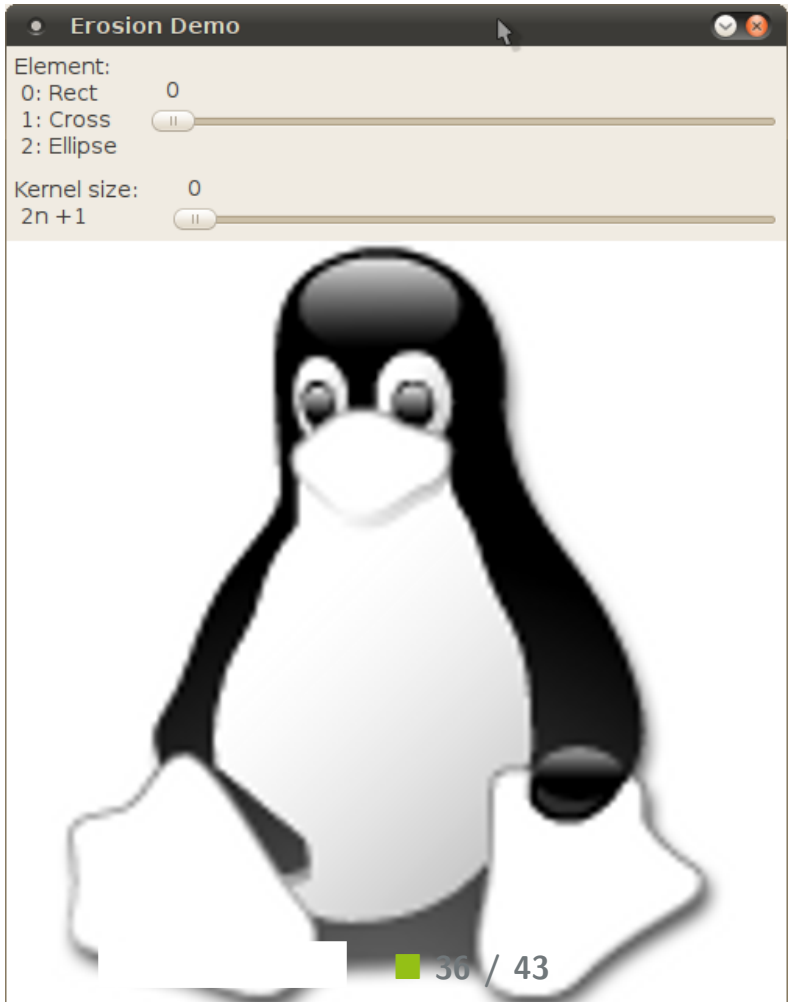
Ajouter une trackbar

```
namedWindow( "Erosion Demo", CV_WINDOW_AUTOSIZE );
```



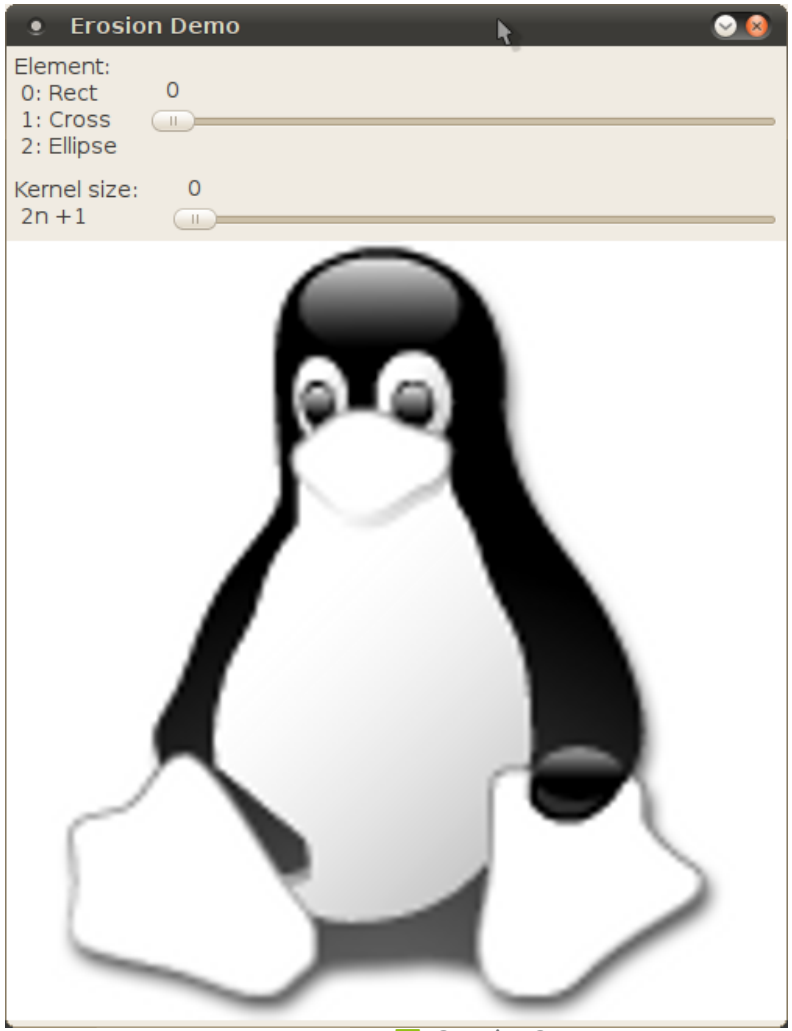
Ajouter une trackbar

```
createTrackbar( "Element:\n 0: Rect \n 1: Cross \n
    2: Ellipse", "Erosion Demo",&erosion_elem,
    2,Erosion );
```

Ajouter une trackbar

```
createTrackbar( "Kernel size:\n 2n +1", "Erosion
    Demo",&erosion_size, 21, Erosion );
```

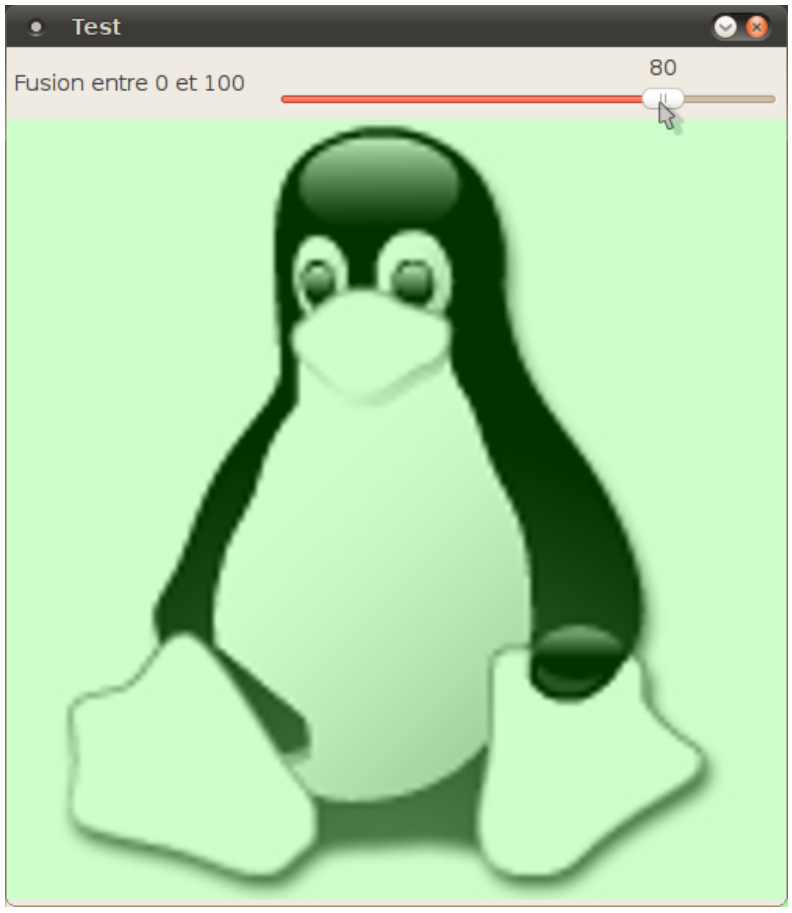



Visualiser plusieurs images

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;


int main( int argc, char** argv )
{
  Mat srcA=imread( "linux_icon.png" );
  Mat srcB=imread( "linux_iconG.png" );

  /// Create windows
  namedWindow( "Image A", CV_WINDOW_AUTOSIZE );
  namedWindow( "Image B", CV_WINDOW_AUTOSIZE );
  cvMoveWindow( "Image A", 0, 0 );
  cvMoveWindow( "Image B", srcA.cols+50, 0 );
  imshow( "Image A", srcA );
  imshow( "Image B", srcB );
  waitKey(0);
  return 0;
}
```

Visualiser plusieurs images

```
#include <cv.h>
#include <highgui.h>
#include <iostream>
using namespace cv;
Mat src1, src2, dst;
int fusion;
void Fusion( int , void* )
{
  double alpha=fusion/100.0;
  double beta = ( 1.0 - alpha );
  addWeighted( src1, alpha, src2, beta, 0.0, dst);
  imshow( "Test", dst );
}

int main( int argc, char** argv )
{
  fusion=50;
  src1 = imread("linux_iconGG.png");
  src2 = imread("vert.png");
  if( !src1.data ) { printf("Error loading src1 \n"); return -1; }
  if( !src2.data ) { printf("Error loading src2 \n"); return -1; }

  namedWindow("Test", 1);
  createTrackbar(  "Fusion entre 0 et 100","Test",&fusion, 100,Fusion );
  Fusion(0,0);
  waitKey(0);
  return 0;
}
```




MATIS
METHODES D'ANALYSE ET DE TRAITEMENT
D'IMAGES POUR LA STEREO-RESTITUTION

CMake

Pour utiliser OpenCV dans un projet c++, on peut utiliser CMake. Voici un exemple de fichier CMakeLists :

CMakeLists.txt

```
PROJECT( helloworld_proj )
FIND_PACKAGE( OpenCV REQUIRED )
ADD_EXECUTABLE( helloworld main.cpp )
TARGET_LINK_LIBRARIES( helloworld ${OpenCV_LIBS} )
```

Voici un exemple de fichier main.cpp :

main.cpp

```
#include <cv.h>
#include <highgui.h>
#include <iostream>
using namespace cv;
Mat src1, src2, dst;
int fusion;
void Fusion( int , void* )
{
  double alpha=fusion/100.0;
  double beta = ( 1.0 - alpha );
  addWeighted( src1, alpha, src2, beta, 0.0, dst);
  imshow( "Test", dst );
}

int main( int argc, char** argv )
{
  fusion=50;
  src1 = imread("linux_iconGG.png");
  src2 = imread("vert.png");
  if( !src1.data ) { printf("Error loading src1 \n"); return -1; }
  if( !src2.data ) { printf("Error loading src2 \n"); return -1; }

  namedWindow("Test", 1);
  createTrackbar(  "Fusion entre 0 et 100","Test",&fusion, 100,Fusion );
  Fusion(0,0);
  waitKey(0);
  return 0;
}
```