Learn OpenCV by Examples

OpenCV simplified for beginners by the use of examples. Learn OpenCV with basic implementation of different algorithms.

Home | For Beginners | Table of Contents | Keywords

Basic drawing examples

Drawing a line

void **line**(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0)

Parameters:

- **img** – Image.
- **pt1** – First point of the line segment.
- **pt2** – Second point of the line segment.
- **color** – Line color.
- **thickness** – Line thickness.
- **lineType** – Type of the line:
  - **8** (or omitted) - 8-connected line.
  - **4** - 4-connected line.
  - **CV_AA** - antialiased line.
- **shift** – Number of fractional bits in the point coordinates.

Example 1: Drawing a line

```
#include <opencv2/core/core.hpp>
```

SEARCH CONTENTS OF THIS BLOG

Search

POPULAR POSTS

1 Find Contour

2 Basic drawing examples

3 Line Detection by Hough Line Transform

4 Face Detection using Haar-Cascade Classifier

5 Perspective Transform

6 Sobel Edge Detection

```
#include <opencv2/highgui/highgui.hpp>
using namespace cv;

int main( )
{
  // Create black empty images
  Mat image = Mat::zeros( 400, 400, CV_8UC3 );

  // Draw a line
  line( image, Point( 15, 20 ), Point( 70, 50), Scalar( 110, 220, 0
  imshow("Image",image);

  waitKey( 0 );
  return(0);
}
```

Drawing a Circle

void **circle**(Mat& img, Point center, int radius, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
Parameters:

- **img** – Image where the circle is drawn.
- **center** – Center of the circle.
- **radius** – Radius of the circle.
- **color** – Circle color.
- **thickness** – Thickness of the circle outline, if positive. Negative thickness means that a filled circle is to be drawn.
- **lineType** – Type of the circle boundary. See the `line()` description.
- **shift** – Number of fractional bits in the coordinates of the center and in the radius value.

The function `circle` draws a simple or filled circle with a given center and radius.

Example 2: Drawing a Circle

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;

int main( )
{
  // Create black empty images
  Mat image = Mat::zeros( 400, 400, CV_8UC3 );

  // Draw a circle
```

CATEGORIES

- Accessory
- Applications
- Basics
- Edge Detection
- Feature Extraction
- Filter
- Miscellaneous
- Morphological Operation

```
    circle( image, Point( 200, 200 ), 32.0, Scalar( 0, 0, 255 ), 1, 8
    imshow("Image",image);

    waitKey( 0 );
    return(0);
}
```

Drawing an Ellipse

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)
Parameters:

- **img** – Image.
- **center** – Center of the ellipse.
- **axes** – Length of the ellipse axes.
- **angle** – Ellipse rotation angle in degrees.
- **startAngle** – Starting angle of the elliptic arc in degrees.
- **endAngle** – Ending angle of the elliptic arc in degrees.
- **box** – Alternative ellipse representation via `RotatedRect` or `CvBox2D`. This means that the function draws an ellipse inscribed in the rotated rectangle.
- **color** – Ellipse color.
- **thickness** – Thickness of the ellipse arc outline, if positive. Otherwise, this indicates that a filled ellipse sector is to be drawn.
- **lineType** – Type of the ellipse boundary. See the `line()` description.
- **shift** – Number of fractional bits in the coordinates of the center and values of axes.

The functions `ellipse` with less parameters draw an ellipse outline, a filled ellipse, an elliptic arc, or a filled ellipse sector. A piecewise-linear curve is used to approximate the elliptic arc boundary.

If you use the first variant of the function and want to draw the whole ellipse, not an arc, pass `startAngle=0` and `endAngle=360`.

Example 3: Drawing an Ellipse

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;

int main( )
{
  // Create black empty images
  Mat image = Mat::zeros( 400, 400, CV_8UC3 );

  // Draw a ellipse
  ellipse( image, Point( 200, 200 ), Size( 100.0, 160.0 ), 45, 0,
  ellipse( image, Point( 200, 200 ), Size( 100.0, 160.0 ), 135, 0
  ellipse( image, Point( 200, 200 ), Size( 150.0, 50.0 ), 135, 0,
  imshow("Image",image);

  waitKey( 0 );
  return(0);
}
```

Drawing a Rectangle

void **rectangle**(Mat& img, Point pt1, Point pt2, const Scalar& color, int thickness=1, int lineType=8, int shift=0)
Parameters:

- **img** – Image.
- **pt1** – Vertex of the rectangle.
- **pt2** – Vertex of the rectangle opposite to `pt1` .
- **rec** – Alternative specification of the drawn rectangle.
- **color** – Rectangle color or brightness (grayscale image).
- **thickness** – Thickness of lines that make up the rectangle. *Negative values*, like **`CV_FILLED`** , mean that the function has to draw a *filled rectangle*.
- **lineType** – Type of the line. See the `line()` description.
- **shift** – Number of fractional bits in the point coordinates.

Example 4: Drawing a Rectangle

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;

int main( )
{
  // Create black empty images
  Mat image = Mat::zeros( 400, 400, CV_8UC3 );

  // Draw a rectangle ( 5th argument is not -ve)
  rectangle( image, Point( 15, 20 ), Point( 70, 50), Scalar( 0, 55,
  imshow("Image1",image);
  // Draw a filled rectangle ( 5th argument is -ve)
  rectangle( image, Point( 115, 120 ), Point( 170, 150), Scalar( 10
  imshow("Image2",image);

  waitKey( 0 );
  return(0);
}
```

Drawing a Filled Polygon

void **fillPoly**(Mat& img, const Point** pts, const int* npts, int ncontours, const Scalar& color, int lineType=8, int shift=0, Point offset=Point())
Parameters:

- **img** – Image.
- **pts** – Array of polygons where each polygon is represented as an array of points.
- **npts** – Array of polygon vertex counters.
- **ncontours** – Number of contours that bind the filled region.
- **color** – Polygon color.
- **lineType** – Type of the polygon boundaries. See the `line()` description.
- **shift** – Number of fractional bits in the vertex coordinates.
- **offset** – Optional offset of all points of the contours.

The function `fillPoly` fills an area bounded by several polygonal contours. The function can fill complex areas, for example, areas with holes, contours with self-intersections (some of their parts), and so forth.

Example 4: Drawing a Filled Polygon

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;

int main( )
{
  // Create black empty images
  Mat image = Mat::zeros( 400, 400, CV_8UC3 );

  int w=400;
  // Draw a circle
  /** Create some points */
  Point rook_points[1][20];
  rook_points[0][0] = Point( w/4.0, 7*w/8.0 );
  rook_points[0][1] = Point( 3*w/4.0, 7*w/8.0 );
  rook_points[0][2] = Point( 3*w/4.0, 13*w/16.0 );
  rook_points[0][3] = Point( 11*w/16.0, 13*w/16.0 );
  rook_points[0][4] = Point( 19*w/32.0, 3*w/8.0 );
  rook_points[0][5] = Point( 3*w/4.0, 3*w/8.0 );
  rook_points[0][6] = Point( 3*w/4.0, w/8.0 );
  rook_points[0][7] = Point( 26*w/40.0, w/8.0 );
  rook_points[0][8] = Point( 26*w/40.0, w/4.0 );
  rook_points[0][9] = Point( 22*w/40.0, w/4.0 );
  rook_points[0][10] = Point( 22*w/40.0, w/8.0 );
  rook_points[0][11] = Point( 18*w/40.0, w/8.0 );
  rook_points[0][12] = Point( 18*w/40.0, w/4.0 );
  rook_points[0][13] = Point( 14*w/40.0, w/4.0 );
  rook_points[0][14] = Point( 14*w/40.0, w/8.0 );
  rook_points[0][15] = Point( w/4.0, w/8.0 );
  rook_points[0][16] = Point( w/4.0, 3*w/8.0 );
  rook_points[0][17] = Point( 13*w/32.0, 3*w/8.0 );
  rook_points[0][18] = Point( 5*w/16.0, 13*w/16.0 );
  rook_points[0][19] = Point( w/4.0, 13*w/16.0) ;

  const Point* ppt[1] = { rook_points[0] };
  int npt[] = { 20 };

  fillPoly( image, ppt, npt, 1, Scalar( 255, 255, 255 ), 8 );
  imshow("Image",image);

  waitKey( 0 );
  return(0);
}
```

Putting Text in image

putText renders the specified text string in the image.

void **putText**(Mat& img, const string& text, Point org, int fontFace, double fontScale, Scalar color, int thickness=1, int lineType=8, bool bottomLeftOrigin=false)
Parameters:

- **img** – Image.
- **text** – Text string to be drawn.
- **org** – Bottom-left corner of the text string in the image.
- **fontFace** – Font type. One of `FONT_HERSHEY_SIMPLEX`, `FONT_HERSHEY_PLAIN`, `FONT_HERSHEY_DUPLEX`, `FONT_HERSHEY_COMPLEX`, `FONT_HERSHEY_TRIPLEX`, `FONT_HERSHEY_COMPLEX_SMALL`, `FONT_HERSHEY_SCRIPT_SIMPLEX`, or `FONT_HERSHEY_SCRIPT_COMPLEX`, where each of the font ID's can be combined with `FONT_HERSHEY_ITALIC` to get the slanted letters.
- **fontScale** – Font scale factor that is multiplied by the font-specific base size.
- **color** – Text color.
- **thickness** – Thickness of the lines used to draw a text.
- **lineType** – Line type. See the `line` for details.

- **bottomLeftOrigin** – When true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.

Example 5: Putting Text in image

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;

int main( )
{
  // Create black empty images
  Mat image = Mat::zeros( 400, 400, CV_8UC3 );

  putText(image, "Hi all...", Point(50,100), FONT_HERSHEY_SIMPLEX,
  imshow("Image",image);

  waitKey( 0 );
  return(0);
}
```

Source:
http://docs.opencv.org/modules/core/doc/drawing_functions.html?highlight=rectangle#void%20line%28Mat&%20img,%20Point%20pt1,%20Point%20pt2,%20const%20Scalar&%20color,%20int%20thickness,%20int%20lineType,%20int%20shift%29

+1 Recommend this on Google

Labels: Basics

5 comments:

Anonymous May 6, 2014 at 2:36 AM

Great tutorial. Thanks!

Reply

Anonymous May 26, 2014 at 8:14 PM