OpenCV Tutorial C++

Smooth / Blur Images

In this lesson, I am going to explain how to smooth an image. Sometimes it is also called blurring. The main objective of smoothing is to reduce noise. Such noise reduction is a typical image pre-processing method which will improve the final result.
There are various ways to smooth or blur an image. I will explain you of most common smoothing techniques with OpenCV C++ examples.

1. Homogeneous Smoothing
2. Gaussian Smoothing
3. Median Smoothing
4. Bilateral Smoothing

Smoothing is done by sliding a window (kernel or filter) across the whole image and calculating each pixel a value based on the value of the kernel and the value of overlapping pixels of original image. This process is mathematically called as convolving an image with some kernel. The kernel is the only difference in all of the above types of smoothing (Blurring) methods.

For example, 5 x 5 kernel used in homogeneous smoothing (blurring) is as below. This kernel is known as "**Normalized box filter**".

$$\frac{1}{25}\begin{bmatrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{bmatrix}$$

whereas 5 x 5 kernel used in Gaussian smoothing (blurring) is as below. This kernel is known as "**Gaussian kernel**"

$$\frac{1}{273}\begin{bmatrix}1&4&7&4&1\\4&16&26&16&4\\7&26&41&26&7\\4&16&26&16&4\\1&4&7&4&1\end{bmatrix}$$

I will give you some important facts about smoothing kernels (filters)

- Number of rows and number of columns of a kernel should be odd (e.g. - 3x3, 11x5, 7x7, etc)
- When the size of the kernel is getting larger, processing time also becomes larger

Homogeneous Smoothing

"Homogeneous Smoothing" is also called as "Homogeneous Blurring", "Homogeneous Filtering" or "Box Blurring". This is the most simplest method of smoothing an image. It takes simply the average of the neighbourhood of a pixel and assign that value to itself.
You have to choose right size of the kernel. If it is too large, small features of the image may be disappeared and image will look blurred. If it is too small, you cannot eliminate noises of the image.

SITE MAP
Home
OpenCV Lessons
.. What is OpenCV?
.. Installing & Configuring v
.. Basics of OpenCV API
.. Read & Display Image
.. Capture Video from File o
.. Write Image & Video to Fi
.. Filtering Images
.....Change Brightness of In
.....Change Contrast of Imag
.....Historgram Equalization
.....Smooth / Blur Images
.. How to Add Trackbar
.. How to Detect Mouse Clic
.. Rotate Image & Video
.. Color Detection & Object
.. Shape Detection &Trackir
Reference Books
About Me

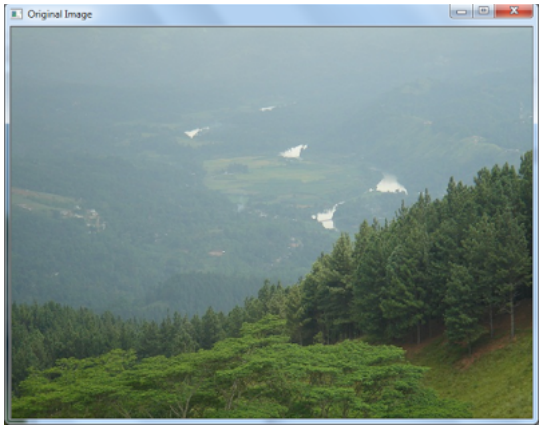
OpenCV code

In the following OpenCV code, kernel size is increasing from 1x1 to 29x29. Observe the smoothed image when increasing the kernel size. The size of the kernel by which the image is smoothed, is displayed in the middle of the image.

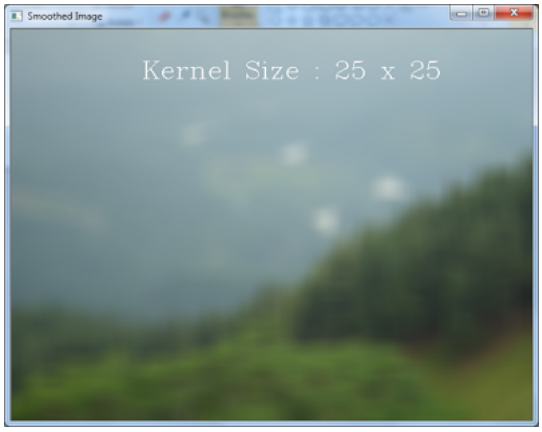
Note that in the following OpenCV code, kernel size is incremented by two in each steps to get odd numbers.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

int main( int argc, char** argv )
{
 //create 2 empty windows
 namedWindow( "Original Image" , CV_WINDOW_AUTOSIZE );
 namedWindow( "Smoothed Image" , CV_WINDOW_AUTOSIZE );

 // Load an image from file
 Mat src = imread( "MyPic.JPG", 1 );

 //show the loaded image
 imshow( "Original Image", src );

 Mat dst;
 char zBuffer[35];

 for ( int i = 1; i < 31; i = i + 2 )
 {
    //copy the text to the "zBuffer"
    _snprintf_s(zBuffer, 35,"Kernel Size : %d x %d", i, i);

    //smooth the image in the "src" and save it to "dst"
    blur( src, dst, Size( i, i ) );

    //put the text in the "zBuffer" to the "dst" image
    putText( dst, zBuffer, Point( src.cols/4, src.rows/8), CV_FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255) );

    //show the blurred image with the text
    imshow( "Smoothed Image", dst );

    //wait for 2 seconds
    int c = waitKey(2000);

    //if the "esc" key is pressed during the wait, return
    if (c == 27)
    {
        return 0;
    }
 }

 //make the "dst" image, black
 dst = Mat::zeros( src.size(), src.type() );

 //copy the text to the "zBuffer"
 _snprintf_s(zBuffer, 35,"Press Any Key to Exit");

 //put the text in the "zBuffer" to the "dst" image
 putText( dst, zBuffer, Point( src.cols/4,  src.rows / 2), CV_FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255) );

 //show the black image with the text
 imshow( "Smoothed Image", dst );

 //wait for a key press infinitely
 waitKey(0);

 return 0;

}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

You can download this OpenCV visual C++ project from here.



Original Image



Smoothed Image - Because of the kernel size is too large, the image looks blurry

Explanation of New OpenCV Functions

- **void blur(InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1), int borderType = BORDER_DEFAULT)**

This OpenCV function will smooth an image using normalized box filter (Homogeneous Blur). The normalized box filter (kernel) is explained in the beginning of this post. Every channel of the input image is processed independently.

  - **src** - Source image. (The depth of the image should be one of the following; CV_8U, CV_16S, CV_16U, CV_32F or CV_64F)
  - **dst** - Output image (It will have the same size and same depth as the input image)
  - **ksize** - Size of the kernel
  - **anchor** - Point(-1,-1) value means that the anchor is at the middle of the kernel. If you want, you can define your own point
  - **borderType** - You can define various border interpolation methods. This value only affects the pixels at the border. (You can use one of the following; BORDER_DEFAULT, BORDER_REFLECT, BORDER_REPLICATE, BORDER_TRANSPARENT, BORDER_REFLECT_101)

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

This OpenCV function renders text string in the image.

  - **img** - The image that you want to put your text
  - **text** - The text that you want to put into the image
  - **org** - Bottom-left corner of the text string in the image
  - **fontFace** - Font type (You can use one of the following; 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. Any one of them can be do "**bitwise or**" with FONT_ITALIC to italicize the font)
  - **fontScale** - Scaling factor (If you use 1, default font size will be drawn)
  - **color** - BGR color of the text
  - **thickness** - Thickness of the lines of the font
  - **lineType** - Line type
  - **bottomLeftOrigin** - If this is true, the origin is set at the bottom-left corner. Otherwise the origin is set at the top-left corner

- **static MatExpr zeros(Size size, int type)**

This OpenCV function will return an array of specified size and type with zero values

  - **size** - size of array (e.g - Size(no. of columns, no. of rows))
  - **type** - type of the array elements

Gaussian Smoothing

"Gaussian Smoothing" is also called as "Gaussian Blurring" or "Gaussian Filtering". This is the most commonly used smoothing method. This is also used to eliminate noises in an image. Gaussian kernel is slided across the image to produce the smoothed image.
Size of the kernel and the standard deviation of the Gaussian distribution in X and Y direction should be chosen carefully. For more information about the Gaussian smoothing, please refer here

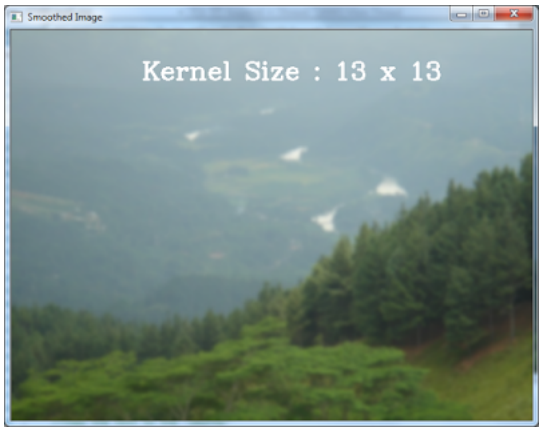
OpenCV Code

This OpenCV/C++ example is very much similar to the previous code except for a function.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

int main( int argc, char** argv )
{
 //create 2 empty windows
 namedWindow( "Original Image" , CV_WINDOW_AUTOSIZE );
 namedWindow( "Smoothed Image" , CV_WINDOW_AUTOSIZE );

 // Load an image from file
 Mat src = imread( "MyPic.JPG", CV_LOAD_IMAGE_UNCHANGED );

 //show the loaded image
 imshow( "Original Image", src );

 Mat dst;
 char zBuffer[35];

 for ( int i = 1; i  <  31; i = i + 2 )
 {
  //copy the text to the "zBuffer"
  _snprintf_s(zBuffer, 35,"Kernel Size : %d x %d", i, i);

  //smooth the image using Gaussian kernel in the "src" and save it to "dst"
  GaussianBlur( src, dst, Size( i, i ), 0, 0 );

  //put the text in the "zBuffer" to the "dst" image
  putText( dst, zBuffer, Point( src.cols/4, src.rows/8), CV_FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255), 2 );

  //show the blurred image with the text
  imshow( "Smoothed Image", dst );

  //wait for 2 seconds
  int c = waitKey(2000);

  //if the "esc" key is pressed during the wait, return
  if (c == 27)
  {
   return 0;
  }
 }

 //make the "dst" image, black
 dst = Mat::zeros( src.size(), src.type() );

 //copy the text to the "zBuffer"
 _snprintf_s(zBuffer, 35,"Press Any Key to Exit");

 //put the text in the "zBuffer" to the "dst" image
 putText( dst, zBuffer, Point( src.cols/4,  src.rows / 2), CV_FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255) );

 //show the black image with the text
 imshow( "Smoothed Image", dst );

 //wait for a key press infinitely
 waitKey(0);
```

```
 return 0;
}
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

You can download this OpenCV visual C++ project from here.



Smooth Image using Gaussian Filter

New OpenCV Function

There is only a new OpenCV function that we can find in the above example.

- **void GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0, int borderType=BORDER_DEFAULT)**

This OpenCV function smooths an image using Gaussian kernel. All channels of the image are processed independently.

- **src** - Source image. (The depth of the image should be one of the following; CV_8U, CV_16S, CV_16U, CV_32F or CV_64F)
- **dst** - Output image (It will have the same size and same depth as the input image)
- **ksize** - Size of the Gaussian kernel (Both dimension of the kernel should be positive and odd)
- **sigmaX** - Standard deviation in X direction. If 0 is used, it will automatically calculated from the kernel size
- **sigmaY** - Standard deviation in Y direction. If 0 is used, it will take the same value as the sigmaX.
- **borderType** - You can define various border interpolation methods. This value only affects the pixels at the border. (You can use one of the following; BORDER_DEFAULT, BORDER_REFLECT, BORDER_REPLICATE, BORDER_TRANSPARENT, BORDER_REFLECT_101)

Median Smoothing

"Median Smoothing" is also called as "Median Blurring" or "Median Filtering". This is also a common smoothing technique. The input image is convolved with a Median kernel. Median filtering is widely used in edge detection algorithms because under certain conditions, it preserves edges while removing noise. For more information about median smoothing, please refer to here.

OpenCV Code

Following OpenCV example is also as same as the previous except for one line.

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

int main( int argc, char** argv )
{
 //create 2 empty windows
 namedWindow( "Original Image" , CV_WINDOW_AUTOSIZE );
 namedWindow( "Smoothed Image" , CV_WINDOW_AUTOSIZE );

 // Load an image from file
 Mat src = imread( "MyPic.JPG", CV_LOAD_IMAGE_UNCHANGED );

 //show the loaded image
 imshow( "Original Image", src );

 Mat dst;
 char zBuffer[35];

 for ( int i = 1; i  <  31; i = i + 2 )
 {
  //copy the text to the "zBuffer"
  _snprintf_s(zBuffer, 35,"Kernel Size : %d x %d", i, i);
```

```
//smooth the image using Median kernel in the "src" and save it to "dst"
medianBlur( src, dst, i );

//put the text in the "zBuffer" to the "dst" image
putText( dst, zBuffer, Point( src.cols/4, src.rows/8), CV_FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255), 2 );

//show the blurred image with the text
imshow( "Smoothed Image", dst );

//wait for 2 seconds
int c = waitKey(2000);

//if the "esc" key is pressed during the wait, return
if (c == 27)
{
 return 0;
}
}

//make the "dst" image, black
dst = Mat::zeros( src.size(), src.type() );

//copy the text to the "zBuffer"
_snprintf_s(zBuffer, 35,"Press Any Key to Exit");

//put the text in the "zBuffer" to the "dst" image
putText( dst, zBuffer, Point( src.cols/4,  src.rows / 2), CV_FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255) );

//show the black image with the text
imshow( "Smoothed Image", dst );

//wait for a key press infinitely
waitKey(0);

return 0;
}
```

///

You can download this OpenCV visual C++ project from here.

Smooth Image using Median Filter (17 x 17)

New OpenCV Functions

- **void medianBlur(InputArray src, OutputArray dst, int ksize)**

This OpenCV function smooth the input image using a Median filter. All channels of the input image is processed independently. This method works in-place.

- **src** - Input image (images with 1, 3 or 4 channels / Image depth should be CV_8U for any value of "**ksize**". If "**ksize**" equals 3 or 5, image depths of CV_16U and CV_32F are also supported.
- **dst** - Output image (It will be of same size and depth as the input image)
- **ksize** - size of the filter (It should be odd and greater than 1) (Note - The resulting filter has the size of **ksize** x **ksize**)

Bilateral Smoothing

"Bilateral Smoothing" is also called as "Bilateral Blurring" or "Bilateral Filtering". This is the most advanced filter to smooth an image and reduce noise. All of the above filters will smooth away the edges while removing noises. But this filter is able to reduce noise of the image while preserving the edges. The drawback of this type of filter is that it takes longer time to process. For more information about bilateral smoothing, please refer to here.

OpenCV Code

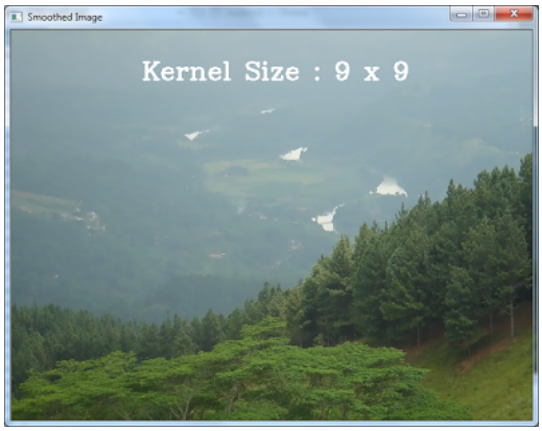
In the following OpenCV/C++ example code is same as the above examples except for one function.

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

int main( int argc, char** argv )
{
 //create 2 empty windows
 namedWindow( "Original Image" , CV_WINDOW_AUTOSIZE );
 namedWindow( "Smoothed Image" , CV_WINDOW_AUTOSIZE );

 // Load an image from file
 Mat src = imread( "MyPic.JPG", CV_LOAD_IMAGE_UNCHANGED );

 //show the loaded image
 imshow( "Original Image", src );

 Mat dst;
 char zBuffer[35];

 for ( int i = 1; i  <  31; i = i + 2 )
 {
  //copy the text to the "zBuffer"
  _snprintf_s(zBuffer, 35,"Kernel Size : %d x %d", i, i);

  //smooth the image using Bilateral filter in the "src" and save it to "dst"
  bilateralFilter( src, dst, i, i, i);

  //put the text in the "zBuffer" to the "dst" image
  putText( dst, zBuffer, Point( src.cols/4, src.rows/8), CV_FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255), 2 );

  //show the blurred image with the text
  imshow( "Smoothed Image", dst );

  //wait for 2 seconds
  int c = waitKey(2000);

  //if the "esc" key is pressed during the wait, return
  if (c == 27)
  {
   return 0;
  }
 }

 //make the "dst" image, black
 dst = Mat::zeros( src.size(), src.type() );

 //copy the text to the "zBuffer"
 _snprintf_s(zBuffer, 35,"Press Any Key to Exit");

 //put the text in the "zBuffer" to the "dst" image
 putText( dst, zBuffer, Point( src.cols/4,  src.rows / 2), CV_FONT_HERSHEY_COMPLEX, 1, Scalar(255, 255, 255) );

 //show the black image with the text
 imshow( "Smoothed Image", dst );
```

```
 //wait for a key press infinitely
 waitKey(0);

 return 0;
}
```

//

You can download this OpenCV visual C++ project from here.



Smooth Image using Bilateral Filter (9 x 9)

New OpenCV Functions

- **void bilateralFilter(InputArray src, OutputArray dst, int d, double sigmaColor, double sigmaSpace, int borderType=BORDER_DEFAULT)**

This OpenCV function filters an image using a Bilateral kernel. Each channel is processed independently. This method does not work in place. For real time applications, it is advised to use a value smaller than 5 for "**d**".

- **src** - Input image (Image with 1 or 3 channels)
- **dst** - Output image (It will have the same size and depth as the input image)
- **d** - Diameter of each pixel neighbourhood
- **sigmaColor** - sigma in the color space
- **sigmaSpace** - sigma in the coordinate space
- **borderType** - You can define various border interpolation methods. This value only affects the pixels at the border. (You can use one of the following; BORDER_DEFAULT, BORDER_REFLECT, BORDER_REPLICATE, BORDER_TRANSPARENT, BORDER_REFLECT_101)

I have discussed with you about most commonly used smoothing methods with OpenCV/C++ examples. Try all of the above smoothing techniques using different filter sizes and observe the output image and the processing time.

Next Lesson:

Previous Lesson: Histogram Equalization of Grayscale or Color Image

Posted by Shermal Fernando

8 comments:

website heatmap October 23, 2013 at 5:14 PM

Thank you so much for taking the time to share this exciting information. But, I would be grateful to you if you could provide some more details about heat map issue.

Reply

Anonymous February 26, 2014 at 6:04 PM

A note for anyone using this on a linux os use snprintf instead of _snprintf_s. Thanks for the code by the way.

Reply