Opencv图像处理平滑锐化操作

图像平滑算法

图像平滑与图像模糊是同一概念，主要用于图像的去噪。平滑要使用滤波器，为不改变图像的相位信息，一般使用线性滤波器。

几种不同的平滑方法：

1. 归一化滤波器
Blurs an image using the normalized box filter.

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

其中ksize为核窗口大小，
Point(-1, -1):
Indicates where the anchor point (the pixel evaluated) is located with respect to the neighborhood.If there is a negative value, then the center of the kernel is considered the anchor point.

2. 高斯滤波

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

sigmaX: The standard deviation in x. Writing 0 implies that x is calculated using kernel size.
sigmaxY: The standard deviation in y. Writing 0 implies that y is calculated using kernel size.

3. 中值滤波

```
void medianBlur(InputArray src, OutputArray dst, int ksize)
```

Size of the kernel (only one because we use a square window). Must be odd.因为其核窗口为正方形，所以他只有一个。

中值滤波对椒盐噪声的去噪效果最好。

Opencv加椒盐噪声

椒盐噪声是由图像传感器，传输信道，解码处理等产生的黑白相间的亮暗点噪声。椒盐噪声往往由图像切割引起。

我们用程序来模拟椒盐噪声，随机选取一些像素，把这些像素设为白色。

```
void salt(Mat& image, int n) {
    for (int k = 0; k<n; k++) {
        int i = rand() % image.cols;
        int j = rand() % image.rows;

        if (image.channels() == 1) {    //判断是一个通道
            image.at<uchar>(j, i) = 255;
        }
        else {
            image.at<cv::Vec3b>(j, i)[0] = 255;
            image.at<cv::Vec3b>(j, i)[1] = 255;
            image.at<cv::Vec3b>(j, i)[2] = 255;
        }
    }
}
```

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

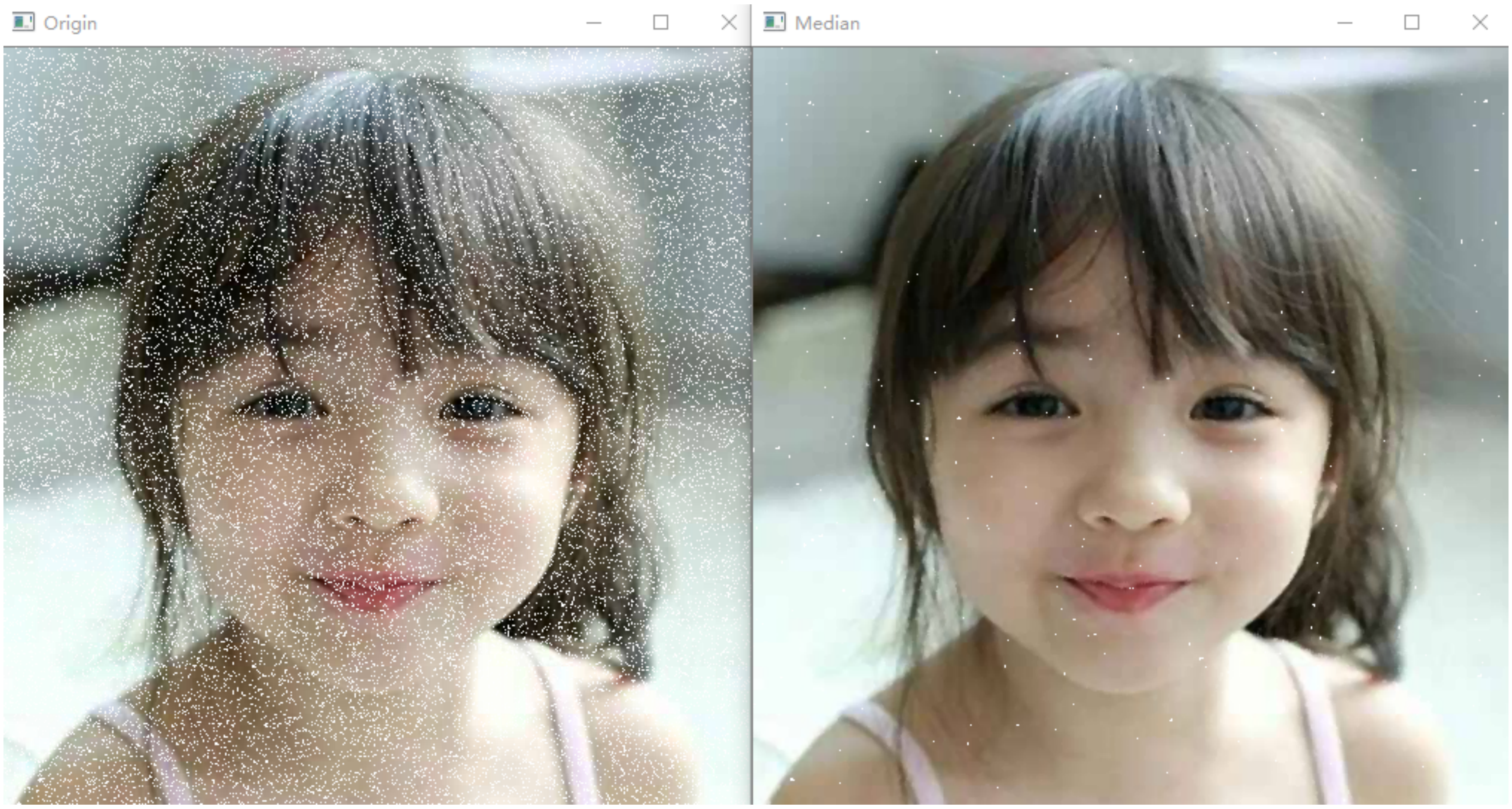
//测试程序

```
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
#include <iostream>
#include <string>
using namespace std;
using namespace cv;
void salt(Mat &image, int n ); //椒盐噪声产生函数
int main(void)
{
    Mat src; Mat dst;

    /// Load the source image
    src = imread("cute.jpg", IMREAD_COLOR);
    salt(src, 30000);
    dst = src.clone();
    medianBlur(src, dst, 3);
    string window_origin = "Origin";
    string window_median = "Median";
    imshow(window_origin, src);
    imshow(window_median, dst);
    waitKey(0);

    return 0;
}
```

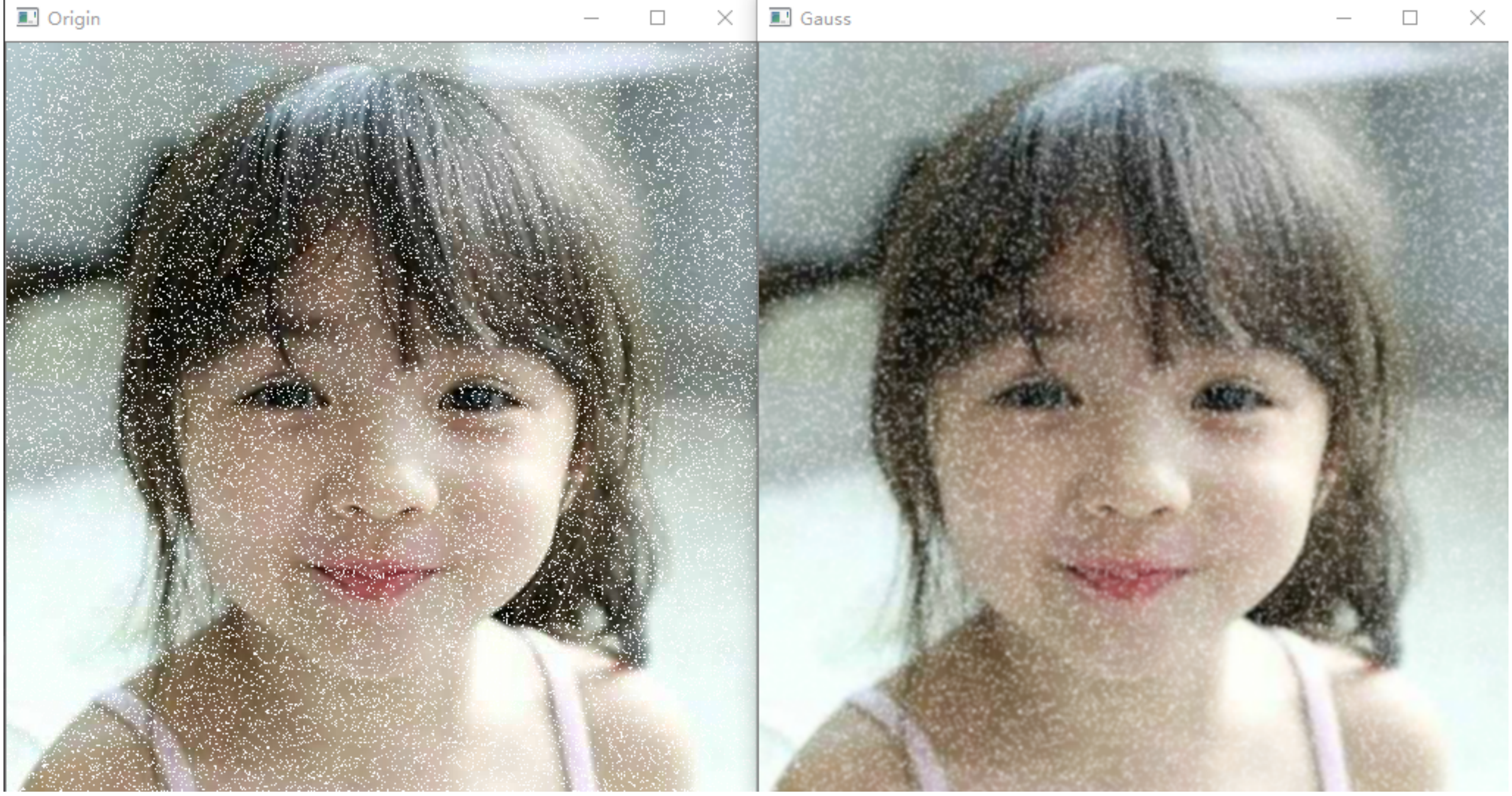
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

-
-
-
-
-
-

可以看到中值滤波对椒盐噪声简直是好的逆天了，这里加入了30000个噪声点。



这里放一个高斯滤波的效果图，可以看到在对椒盐噪声的处理上，高斯是比不过中值滤波的。



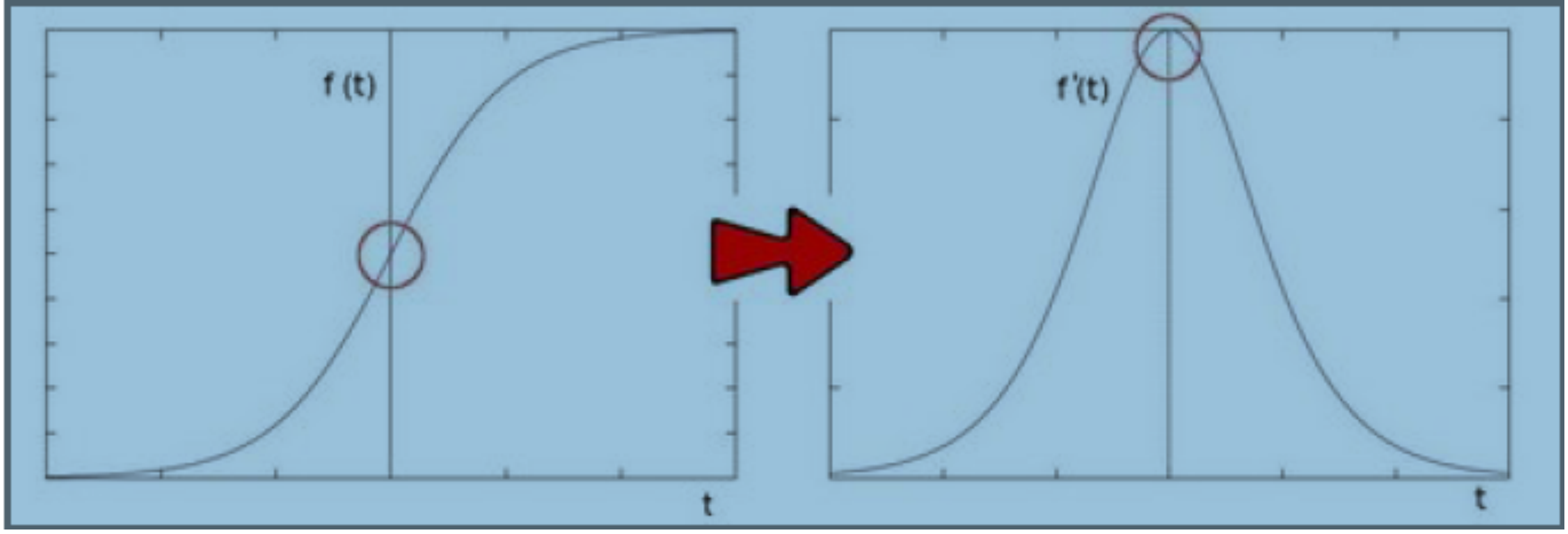
锐化操作

锐化滤波器是为了突出显示图像的边界和其他的细节，这些锐化是基于一阶导数和二阶导数的。
一阶导数可以产生粗的图像边缘，并广泛的应用于边缘提取，二阶导数对于精细的细节相应更好，常被用于图像增强。
常用的算子为Sobel和Laplacian

Sobel算子

关于sobel算子可参考相关书籍或者：
http://blog.csdn.net/caoenze/article/details/46699923?locationNum=2

导数求出的是变化最大的一部分，即突变：



可以看到在圆圈的区域的导数最大。

下面给出具体求解步骤:

> 步骤:
> 1.首先进行对图像高斯平滑消除噪声
> **GaussianBlur(src, src, Size(3,3), 0, 0, BORDER_DEFAULT);**
> 2.将彩色的图像转换成灰度图像
> **cvtColor(src, src_gray, CV_RGB2GRAY);**
> 3.分别计算x方向和y方向的导数，ddepth为图像的深度，应该避免溢出的情况，因此设置CV_16S
> **Sobel(src_gray, grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT);**
> **Sobel(src_gray, grad_y, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT);**
> 4.将其转成CV_8U
> **convertScaleAbs(grad_x, abs_grad_x);**
> **convertScaleAbs(grad_y, abs_grad_y);**
> 5.用两个方向的倒数去模拟梯度
> **addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad);**

应用实例:

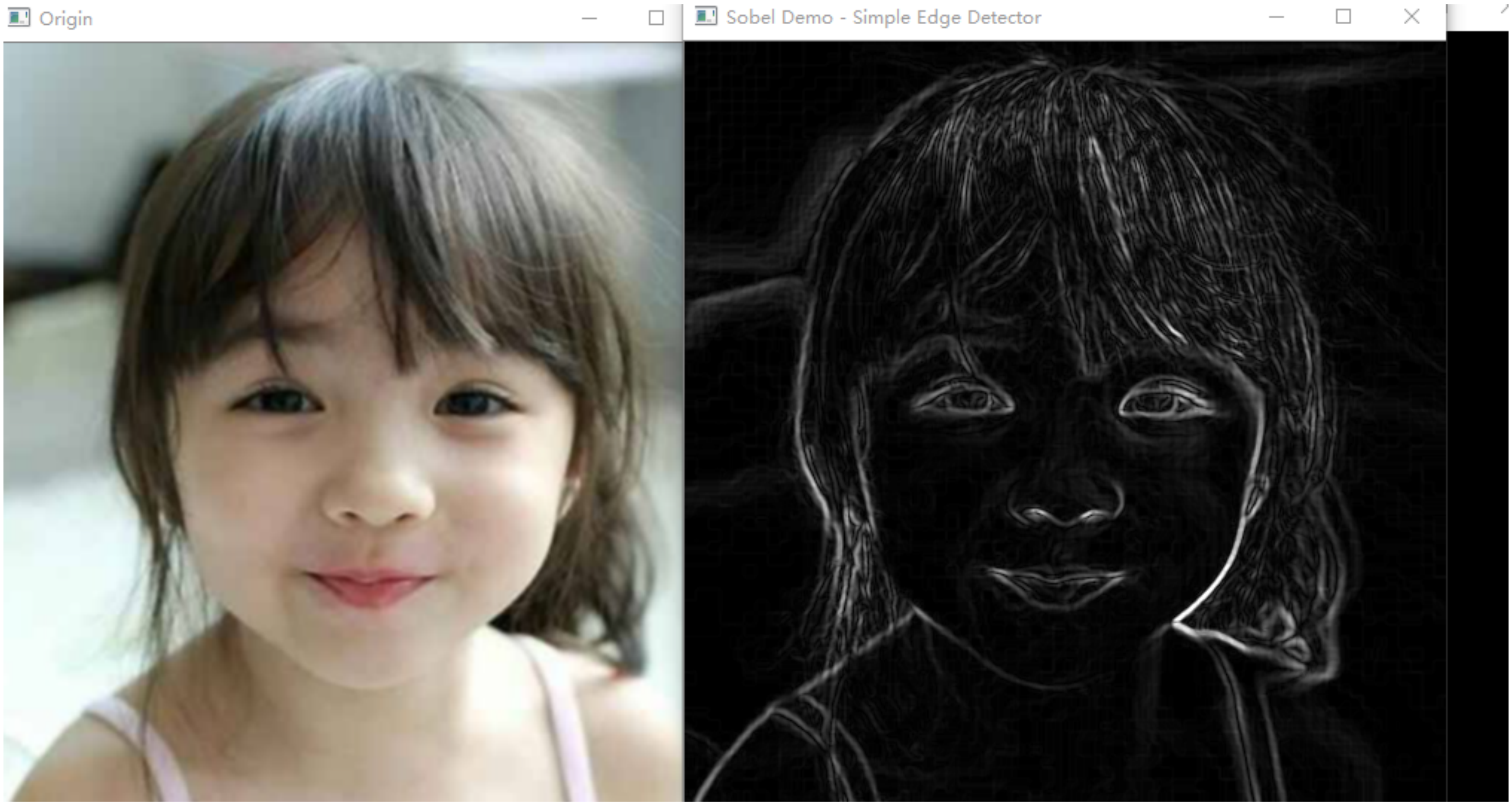
```
#include "opencv2/imgproc.hpp"
#include "opencv2/imgcodecs.hpp"
#include "opencv2/highgui.hpp"
```

```
using namespace cv;

/**
* @function main
*/
int main(int, char** argv)
{
    //![variables]
    Mat src, src_gray;
    Mat grad;
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;
    //![variables]

    //![load]
    src = imread("cute.jpg", IMREAD_COLOR); // Load an image

    if (src.empty())
    {
        return -1;
    }
    //![load]

    //![reduce_noise]
    GaussianBlur(src, src, Size(3, 3), 0, 0, BORDER_DEFAULT);
    //![reduce_noise]

    //![convert_to_gray]
    cvtColor(src, src_gray, COLOR_BGR2GRAY);
    //![convert_to_gray]

    //![sobel]
    /// Generate grad_x and grad_y
    Mat grad_x, grad_y;
    Mat abs_grad_x, abs_grad_y;

    /// Gradient X
    //Scharr( src_gray, grad_x, ddepth, 1, 0, scale, delta, BORDER_DEFAULT );
    Sobel(src_gray, grad_x, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT);

    /// Gradient Y
    //Scharr( src_gray, grad_y, ddepth, 0, 1, scale, delta, BORDER_DEFAULT );
    Sobel(src_gray, grad_y, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT);
    //![sobel]

    //![convert]
    convertScaleAbs(grad_x, abs_grad_x);
    convertScaleAbs(grad_y, abs_grad_y);
    //![convert]

    //![blend]
    /// Total Gradient (approximate)
    addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad);
    //![blend]

    //![display]
    const char* window_name = "Sobel Demo - Simple Edge Detector";
    const char* window="Origin";
    imshow(window,src);
    imshow(window_name, grad);
    waitKey(0);
    //![display]

    return 0;
}
```

- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50
- 51
- 52
- 53
- 54
- 55
- 56
- 57
- 58
- 59
- 60
- 61
- 62
- 63
- 64
- 65
- 66
- 67
- 68

- 69
- 70

结果如图所示：



Laplacian算子

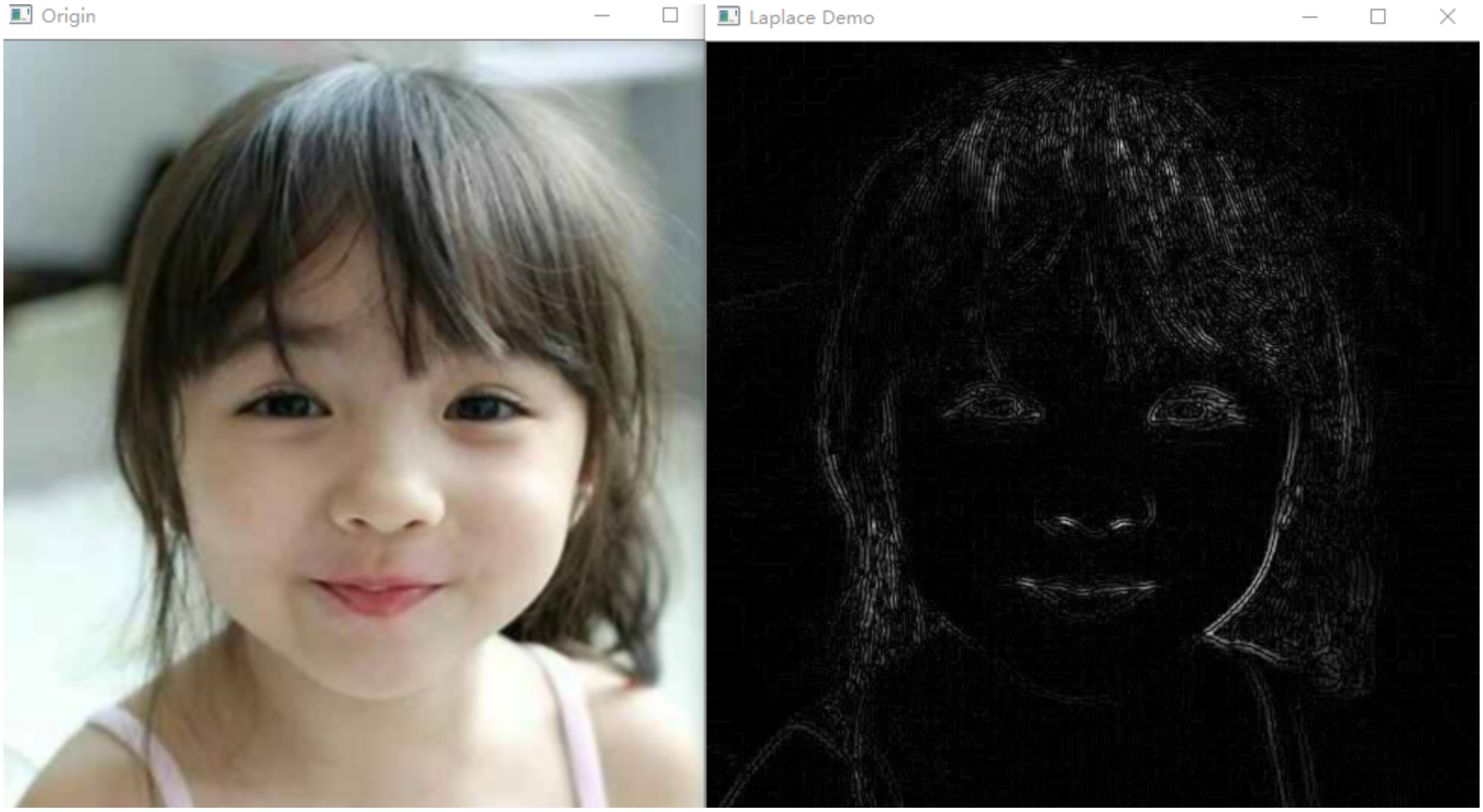
代码实现：

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

using namespace cv;

/**
* @function main
*/
int main(int, char** argv)
{
    //![variables]
    Mat src, src_gray, dst;
    int kernel_size = 3;
    int scale = 1;
    int delta = 0;
    int ddepth = CV_16S;
    //![variables]

    //![load]
    src = imread("cute.jpg", IMREAD_COLOR); // Load an image

    if (src.empty())
    {
        return -1;
    }
    //![load]

    //![reduce_noise]
    /// Reduce noise by blurring with a Gaussian filter
    GaussianBlur(src, src, Size(3, 3), 0, 0, BORDER_DEFAULT);
    //![reduce_noise]

    //![convert_to_gray]
    cvtColor(src, src_gray, COLOR_BGR2GRAY); // Convert the image to grayscale
                                             //![convert_to_gray]

                                             /// Apply Laplace function
    Mat abs_dst;
    //![laplacian]
    Laplacian(src_gray, dst, ddepth, kernel_size, scale, delta, BORDER_DEFAULT
    //![laplacian]

    //![convert]
    convertScaleAbs(dst, abs_dst);
    //![convert]

    //![display]
    const char* window_name = "Laplace Demo";
    const char* window = "Origin";
    imshow(window, src);
    imshow(window_name, abs_dst);
    waitKey();
    //![display]

    return 0;
}
```

- 1
- 2
- 3
- 4

- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49

- 50
- 51
- 52
- 53
- 54
- 55
- 56
- 57


Origin
Laplace Demo