边缘检测 LOG 与 DOG 原理与 OpenCV 代码实战

Dezeming Family

Dezeming

Art. No 0
ISBN 000–00–0000–00–0
Edition 0.0

Cover design by Dezeming Family

Published by Dezeming
Printed in China

目录

0.1 本书前言 5

1 Laplacian of Gaussian 6

1.1 图像微分 6

1.2 拉普拉斯算子 7

1.3 LOG 算子 8

1.4 实际代码测试 9

2 Difference of Gaussian 12

2.1 LOG 的近似 12

2.2 LOG 与 DOG 图像 12

2.3 程序测试 13

Literature 14

前言及简介

DezemingFamily 系列书和小册子因为是电子书，所以可以很方便地进行修改和重新发布。如果您获得了 *DezemingFamily* 的系列书，可以从我们的网站 *[https://dezeming.top/]* 找到最新版。对书的内容建议和出现的错误欢迎在网站留言。

0.1 本书前言

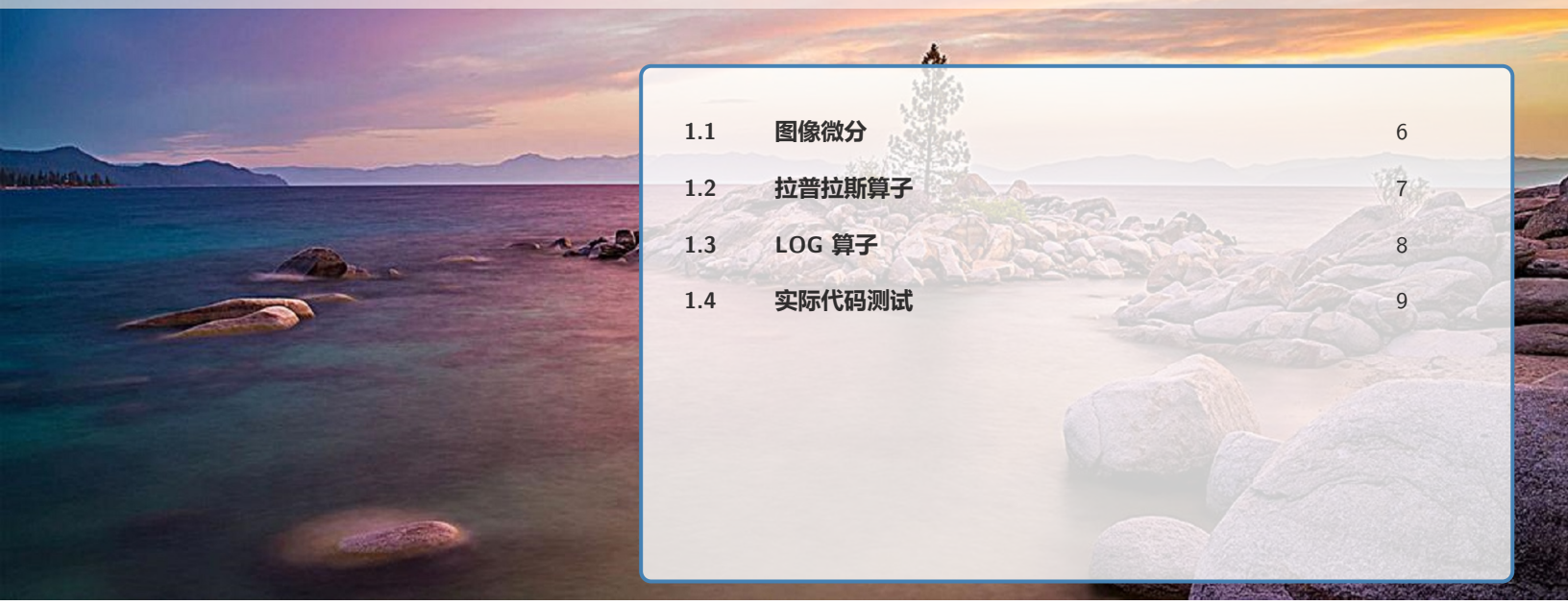
这次打算写个简单点的专题：边缘提取之 LOG 与 DOG。

图像的边缘特征提取是图像特征的一个很重要的方面，其中 LOG 与 DOG 是比较重要的算子。LOG 是基本方法，但为了降低复杂度我们会使用 DOG 去近似 LOG 算子。本书的内容就是深入讲解一下这两个算子的原理和一些测试，希望读者能够更好地掌握这两种算子的基本原理。

我们本书从图像的微分出发，讲解基于图像的基本数学知识，然后再介绍这两种方法。最后给出 OpenCV 的代码测试程序和测试效果，以及对这些效果进行一定的解释。

本书定价为 1.5 元，但用户可以免费获取并使用。如果您得到了本书，在学习中对自己有所帮助，可以往 DezemingFamily 的支付宝账户中进行支持。您的支持将是我们 DezemingFamily 发布更多电子书和努力提高电子书质量的动力！

1. Laplacian of Gaussian



1.1 图像微分 6

1.2 拉普拉斯算子 7

1.3 LOG 算子 8

1.4 实际代码测试 9

本章从数学原理的角度出发，讲解 *Laplacian of Gaussian* 算子的原理和意义，并对该算子从可视化的角度进行一定的分析。最后我们给出 *OpenCV* 实现的程序，来测试效果。

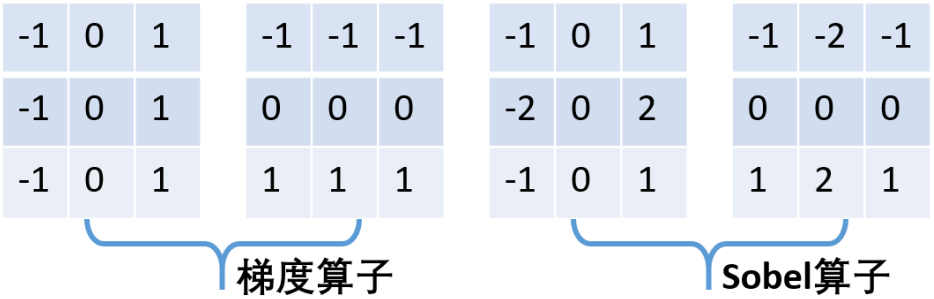
1.1 图像微分

我们通常了解的函数微分都是连续函数的形式，而图像确是离散值。但图像的微分应该满足一些性质：（1）在恒定值区域微分为 0；（2）出现灰度变化时微分为非 0；（3）灰度变化越大，微分值的绝对值越大。

对于一维离散信号，微分定义为：

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \tag{1.1.1}$$

一阶微分通常认为是梯度，在 2×2 的滤波模板中可以方便使用，但在一个 3×3 的模板中，考虑到中心像素的对称性，可以实现下面的梯度算子：



对中心像素的左右或上下两边分别加权，就可以得到上图右边的 Sobel 算子。

1.2 拉普拉斯算子

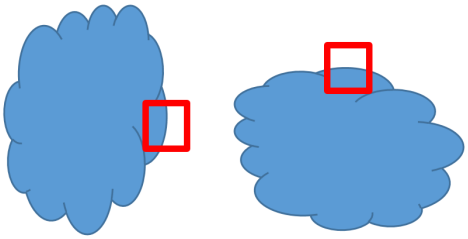
图像的二阶微分定义为：

$$\frac{\partial^2 f}{\partial x^2} = \Big(f(x+1) - f(x)\Big) - \Big(f(x) - f(x-1)\Big) \tag{1.2.1}$$

$$= f(x+1) + f(x-1) - 2f(x) \tag{1.2.2}$$

图像是二维的，因此微分需要表示成分别对 x 方向和 y 方向的偏导。

我们需要思考一种滤波器，这种滤波器无论图像的边缘方向如何，它的响应都是不变的。比如下面的红色滤波核检测的局部范围，两种边缘的响应应该是相同的：



这种滤波器叫做各向同性滤波器（具有旋转不变性），而二阶偏导构成的拉普拉斯算子就是一种各向同性微分算子：

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{1.2.3}$$

我们要注意的是拉普拉斯算子和拉普拉斯变换在一般人类的认知领域里并不存在任何关系（有些人问过我拉普拉斯变换怎么过渡到拉普拉斯算子，这就好比问豆浆怎么过渡成辣条。当然，从应用考虑，这是分析信号的两种工具，一个是可逆积分变换，一个是微分形式）。

因此使用拉普拉斯算子的图像变换就可以定义如下了：

$$\nabla^2 f(x,y) = \Big(f(x+1,y) - f(x,y)\Big) - \Big(f(x,y) - f(x-1,y)\Big) + \Big(f(x,y+1) - f(x,y)\Big) - \Big(f(x,y) - f(x,y-1)\Big) \tag{1.2.4}$$

$$= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{1.2.5}$$

该公式完全可以使用一个滤波模板表示出来。该滤波器可以扩展为多种不同形式的滤波核，比如扩展到 8 邻域以及模板值取反之类的：

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

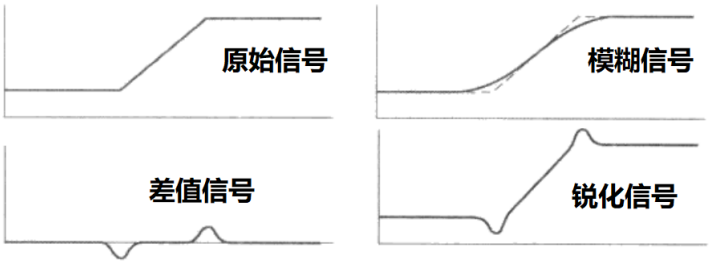
0	1	0
1	-4	1
0	1	0

1	1	1
1	-4	1
1	1	1

在工业界上一般会用三步骤的方法得到锐化图：

- 将原图像进行模糊
- 从原图像中减去模糊图像，得到差值图像

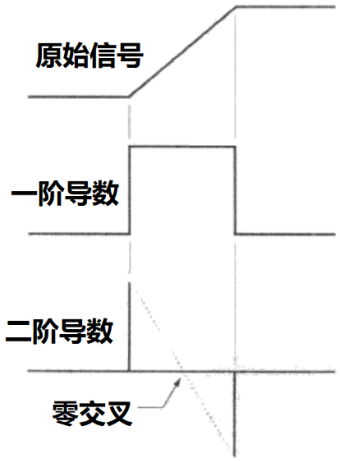
- 将差值图像加到原图像上



1.3 LOG 算子

一般的边缘检测算子如上，并没有那么“智能”。Marr 和 Hildreth 证明，图像的灰度变化与图像的尺寸是没有关系的，也就是说图像模板大小不应该只是小模板（比如上面的 3×3 模板）。

我们知道图像灰度的突然变化会在一阶导数中引起波峰或者波谷，在二阶导数中会引起零交叉：



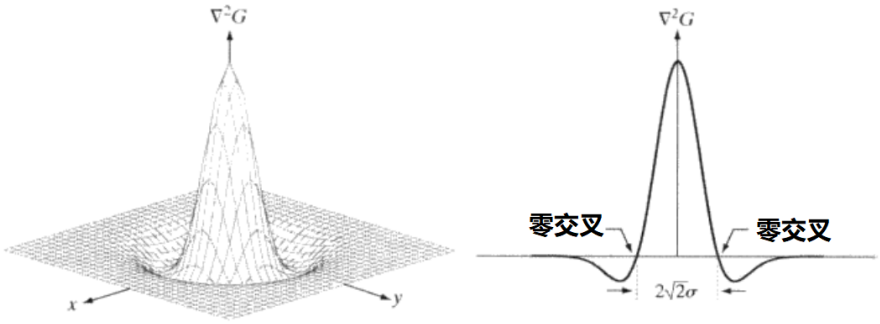
同时需要保证边缘检测算子滤波器模板的所有系数加起来为 0。

因此边缘算子应该具备能够检测图像每一个点的一阶导数和二阶导数，同时应该能在任何尺度下都起作用，即大的算子可以检测模糊边缘，小的算子检测锐利边缘。Marr 和 Hildreth 证明最好的算子就是 $\nabla^2 G$（其实是唯一的尺度函数），G 是二维高斯函数：

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.3.1}$$

$$\nabla^2 G(x,y) = \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.3.2}$$

二阶导的零交叉出现在 $x^2+y^2-2\sigma^2=0$ ，函数波形在二维和横截面上的表示如下：



该函数同样可以得到模板近似，例如 5×5 近似：

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

因此 LOG 算法的描述可以写为：

$$g(x,y) = [\nabla^2 G(x,y)] * f(x,y) \tag{1.3.3}$$
$$= \nabla^2 [G(x,y)] * f(x,y) \tag{1.3.4}$$

上面第二个等号成立的原因是因为对高斯求拉普拉斯和对图像高斯模糊后再求拉普拉斯的效果是一样的（都是线性运算，可以用矩阵向量运算实现）。也就是说，我们可以先对图像进行高斯滤波，然后计算图像的拉普拉斯，找到图像的零交叉即为边缘区域，

1.4 实际代码测试

我们分别用两种方法来做 LOG 测试。

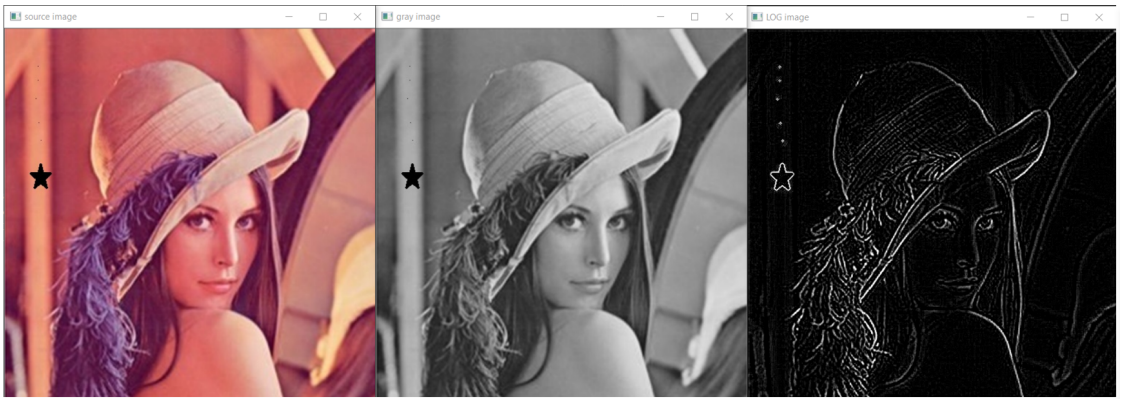
使用 LOG 模板

首先是直接使用 LOG 模板，例如上面的 5×5 模板：

```
1  //OpenCV3
2  #include <opencv2/highgui/highgui.hpp>
3  #include <opencv2/imgproc/imgproc.hpp>
4  #include <iostream>
5  using namespace cv;
6  int main(int argc, char *argv[]){
7    Mat srcImage = imread("p1.jpg", 1);
8    imshow("source image", srcImage);
9    Mat grayImage;
10   cvtColor(srcImage, grayImage, CV_BGR2GRAY);
```

```
11      imshow("gray image", grayImage);
12      Mat imageLOG;
13      Mat kernel = (Mat_<float>(5, 5) <<
14          0,  0, -1,  0,  0,
15          0, -1, -2, -1,  0,
16         -1, -2, 16, -2, -1,
17          0, -1, -2, -1,  0,
18          0,  0, -1,  0,  0);
19      filter2D(grayImage, imageLOG, CV_8UC3, kernel);
20      imshow("LOG image", imageLOG);
21      waitKey();
22      return 0;
23  }
```

就可以得到如下结果：



先模糊再微分

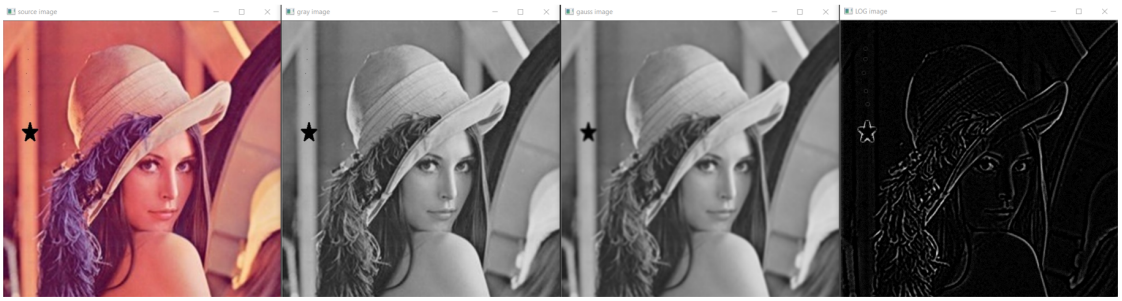
不同的标准差会产生不同的高斯核，这里我选择标准差为 4 的高斯滤波核。

程序如下：

```
1   #include <opencv2/highgui/highgui.hpp>
2   #include <opencv2/imgproc/imgproc.hpp>
3   #include <iostream>
4   using namespace cv;
5   int main(int argc, char *argv[]){
6     Mat srcImage = imread("p1.jpg", 1);
7     imshow("source image", srcImage);
8     Mat grayImage;
9     cvtColor(srcImage, grayImage, CV_BGR2GRAY);
10    imshow("gray image", grayImage);
11    Mat gaussImage;
```

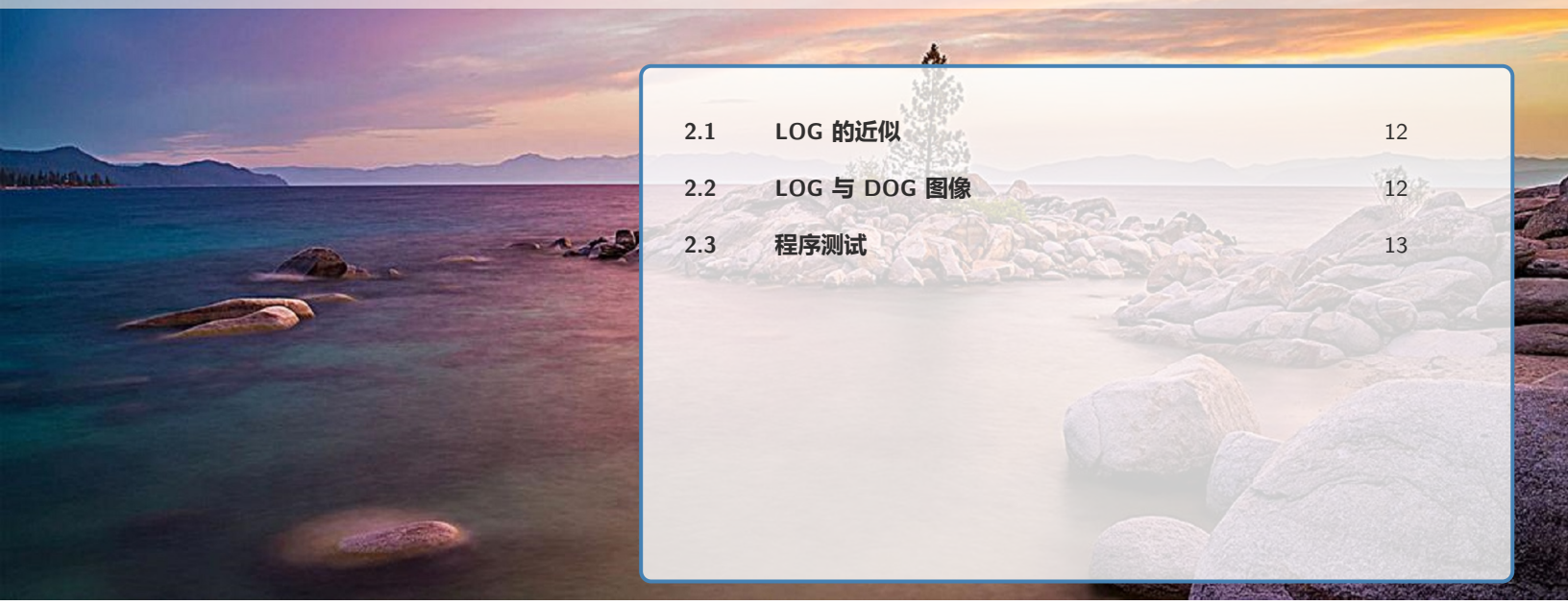
```
12      GaussianBlur(grayImage, gaussImage, cv::Size(5, 5), 4, 4);
13      imshow("gauss image", gaussImage);
14      Mat imageLOG;
15      Mat kernel = (Mat_<float>(3, 3) <<
16          0, -1,  0,
17         -1,  4, -1,
18          0, -1,  0
19         );
20      filter2D(gaussImage, imageLOG, CV_8UC3, kernel);
21      convertScaleAbs(imageLOG, imageLOG);
22      //倍乘一下来显示
23      imshow("LOG image", imageLOG * 10);
24      waitKey();
25      return 0;
26  }
```

测试得到输出结果：



但有时候，我们会认为 LOG 的计算复杂度会比较高，因为它是要计算高斯核拉普拉斯的，因此我们下一章介绍 LOG 的近似方法——DOG。

2. Difference of Gaussian



2.1 LOG 的近似 12

2.2 LOG 与 DOG 图像 12

2.3 程序测试 13

本章介绍 *LOG* 算子的近似方法——*DOG* 算子。我们会讲解近似的原理，并通过实际程序来测试该算子。

2.1 LOG 的近似

Marr 和 Hildreth 指出 LOG 算子可以使用 DOG 算子来近似：

$$G(x,y) = \frac{1}{2\pi\sigma_1^2} e^{-\frac{x^2+y^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{x^2+y^2}{2\sigma_2^2}} \tag{2.1.1}$$

我们要保证的是 $\sigma_1 > \sigma_2$，这是因为 σ 越大，图像越模糊，因此用比较锐利的图像减去比较模糊的图像就能得到边缘信息。

我们计算得到 DOG 算子的零交叉点：

$$G(x,y) = 0 \tag{2.1.2}$$

$$\frac{1}{2\pi\sigma_1^2} e^{-\frac{x^2+y^2}{2\sigma_1^2}} = \frac{1}{2\pi\sigma_2^2} e^{-\frac{x^2+y^2}{2\sigma_2^2}} \tag{2.1.3}$$

$$\frac{\sigma_2^2}{\sigma_1^2} = e^{\frac{x^2+y^2}{2\sigma_1^2} - \frac{x^2+y^2}{2\sigma_2^2}} \tag{2.1.4}$$

如果要保证 LOG 与 DOG 的零交叉点相同，则根据 LOG 的 $x^2+y^2=2\sigma^2$ 我们可以得到：

$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2-\sigma_2^2} \ln\left[\frac{\sigma_1^2}{\sigma_2^2}\right] \tag{2.1.5}$$

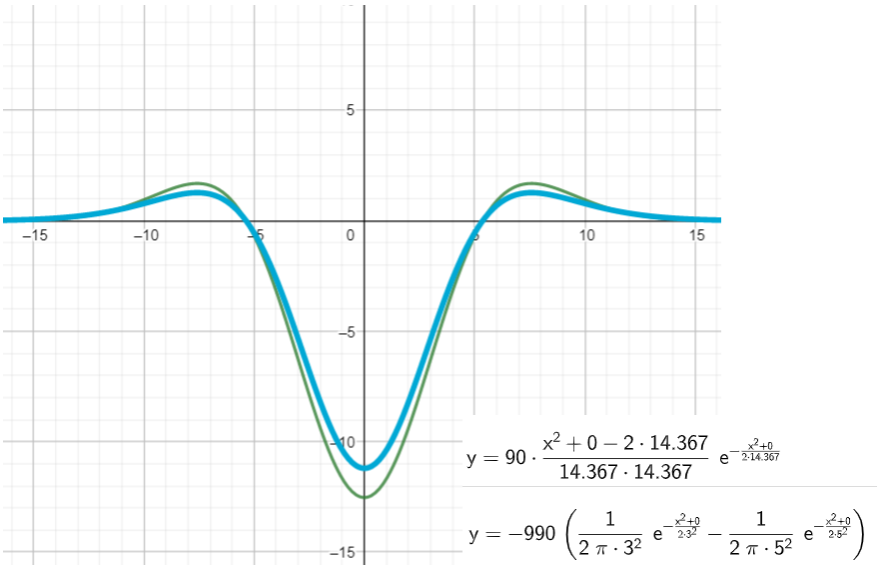
2.2 LOG 与 DOG 图像

我们本节显示一下 LOG 与 DOG 的图像，我们只看三维图的剖面图，即设 $y=0$，然后我们令 LOG 与 DOG 算子的零交叉相同，查看两种算子的函数图像的相似度。

设 $\sigma_1 = 3$，$\sigma_2 = 3$，则算出 σ^2 为：14.367。

```
1        float sigma1 = 3, sigma2 = 5;
2        float sigma1_2 = sigma1 * sigma1, sigma2_2 = sigma2*sigma2;
3        float sigma_2 = sigma1_2 * sigma2_2 / (sigma1_2 - sigma2_2) *
             log(sigma1_2 / sigma2_2);
4        std::cout << (sigma_2);
```

我们查看这两种波形（注意为了显示效果，将原函数都进行了倍增：



其中绿线表示 LOG 曲线，蓝线表示 DOG 曲线。

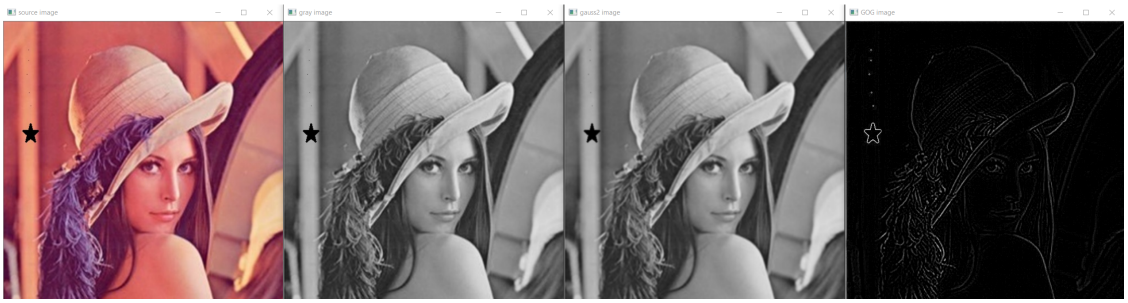
2.3 程序测试

我们取 $\sigma_1 = 0.3$, $\sigma_2 = 0.8$，然后进行处理：

```
1   #include <opencv2/highgui/highgui.hpp>
2   #include <opencv2/imgproc/imgproc.hpp>
3   #include <iostream>
4   using namespace cv;
5   int main(int argc, char *argv[]){
6     Mat srcImage = imread("p04.jpg", 1);
7     imshow("source image", srcImage);
8     Mat grayImage;
9     cvtColor(srcImage, grayImage, CV_BGR2GRAY);
10    imshow("gray image", grayImage);
11    Mat gaussImage1, gaussImage2;
12    GaussianBlur(grayImage, gaussImage1, cv::Size(5, 5), 0.3, 0.3);
13    GaussianBlur(grayImage, gaussImage2, cv::Size(5, 5), 0.8, 0.8);
```

```
14      imshow("gauss1 image", gaussImage1);
15      imshow("gauss2 image", gaussImage2);
16      Mat GODImage = gaussImage1 - gaussImage2;
17      convertScaleAbs(GODImage, GODImage);
18      //倍乘一下来显示
19      imshow("GOG image", GODImage * 10);
20      waitKey();
21      return 0;
22  }
```

得到结果如下：



关于 LOG 与 DOG 的讲解就到此结束了，本书是我写 SIFT 书时因为感觉介绍 LOG 与 DOG 放在 SIFT 书里会占用一些空间，不太好展开，所以就新开了这本小书。

Bibliography

[1] Gonzalez, R. C., & Woods, R. E.(1977). Digital image processing.

[2] https://blog.csdn.net/geduo_feng/article/