前言

高斯模糊是图像处理中几乎每个程序员都或多或少听过的名词，但是对其原理大家可能并不了解，只知道通过高斯模糊能实现图像毛玻璃效果。

本文首先介绍图像处理中最基本的概念：卷积；随后介绍高斯模糊的核心内容：高斯滤波器；接着，我们从头实现了一个Java版本的高斯模糊算法，以及实现RenderScript版本。由于我们自己实现的Java版本的高斯模糊算法的效率太低，因此最后介绍比较有名的高斯模糊的开源项目：Blurry以及BlurKit-Android。

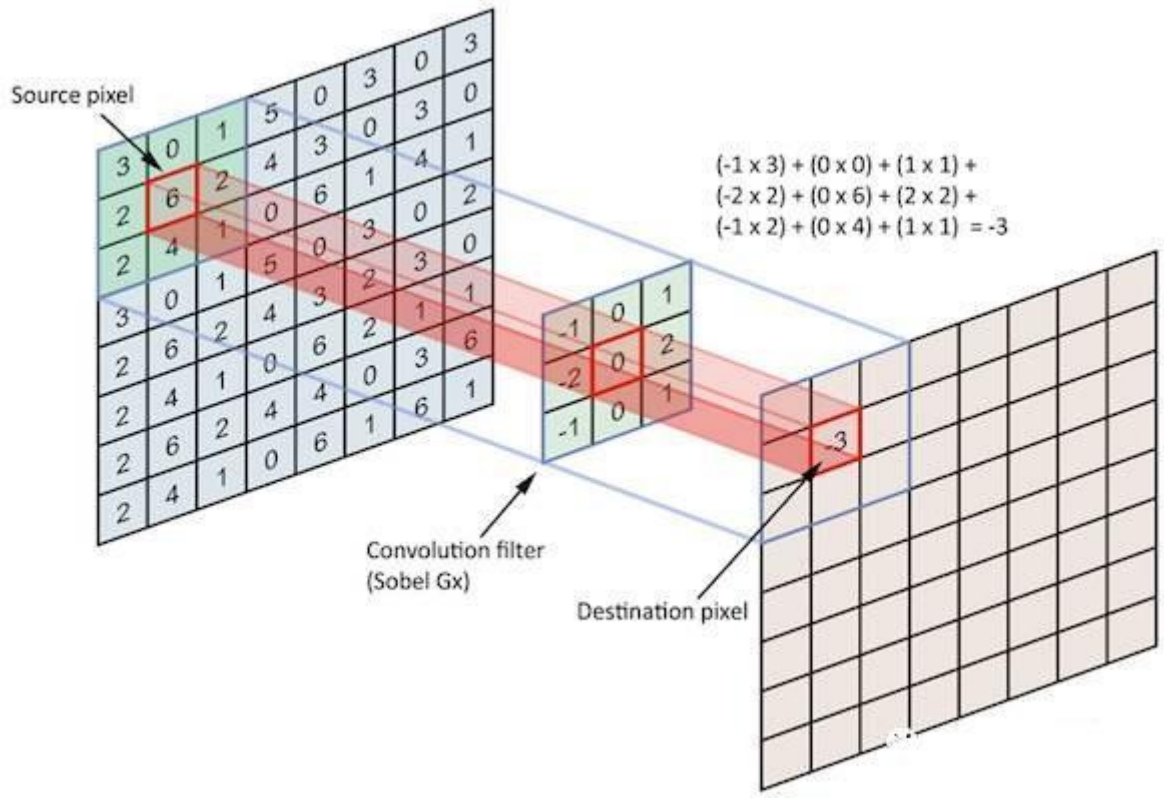
BlurDemo是本文的配套Demo：

- Demo1：Java版本的高斯模糊的简单实现。
- Demo2：RenderScript的高斯模糊实现。
- Demo3：BlurKit-Android的基本使用。
- Demo4：Blurry的基本使用。

卷积

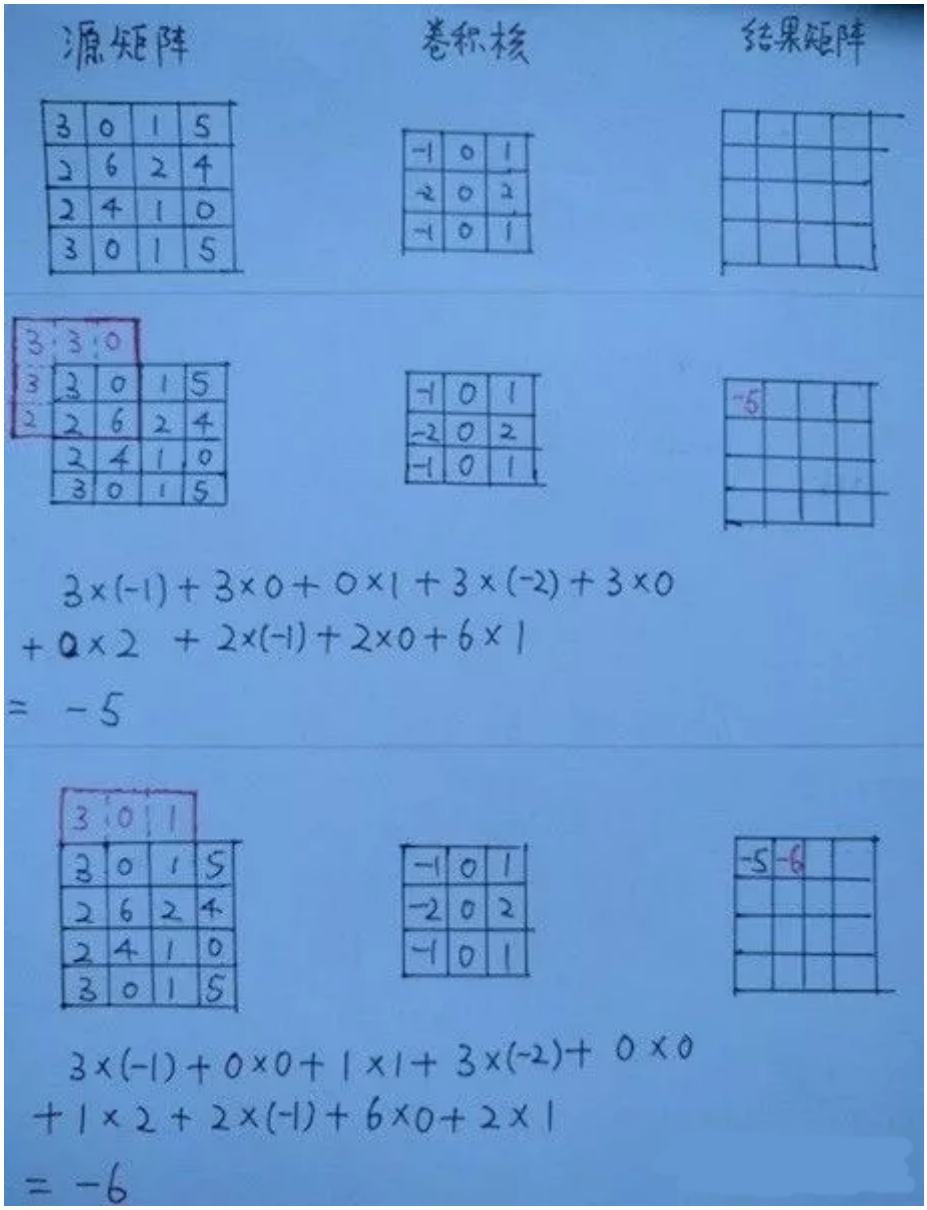
本文只讨论图像，而图像可以表示为二维矩阵，其中每个元素为ARGB像素值，因此这里讨论二维矩阵的卷积操作。卷积（Convolution）是图像处理中最基本的操作，就是一个二维矩阵A（M*N*）*和一个二维矩阵B（*mn）做若干操作，生成一个新的二维矩阵C（M*N），其中m和n远小于M和N，B称为卷积核（kernel），又称滤波器矩阵或模板。

这里举个卷积的例子，如图：



上图中，最左边的是源矩阵（*88*），*中间是卷积核（*33，半径为1），最右边是通过对前面两个矩阵做卷积生成的结果矩阵。图中，如果我们要求出结果矩阵中第二行第二列的元素的值，则把卷积核的中心元素（值为0）和源矩阵的第二行第二列（值为6）对齐，然后求加权和，即图中的公式，最后得到-3。

我们再举一个例子：



上图也展示了如何做卷积的过程，比如要求出结果矩阵中第一行第一列的值，则把卷积核的中心对准源矩阵的第一行第一列，发现部分区域超出源矩阵的范围了（图中红色部分），解决方法有很多，这里的方案是：用边界值填充。接着做加权和，结果为-5。接着用同样的方法依次计算结果矩阵的每个元素即可。

通常来说卷积核需要满足：

- 宽和高都为奇数，这样才会有半径和中心的概念。
- 元素总和为1。

滤波器

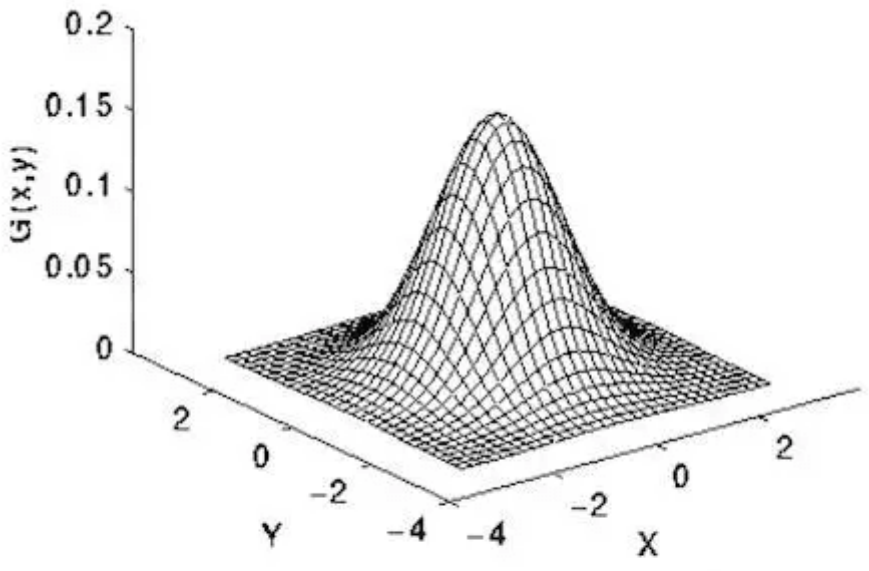
均值滤波器

均值滤波器（Mean Filter）是最简单的一种滤波器，它是最粗糙的一种模糊图像的方法，高斯滤波是均值滤波的高级版本。实际上不同的滤波器就是通过改变卷积核（滤波器），从而改变最后的结果矩阵，中间步骤都一样，都是求加权和。均值滤波器的卷积核通常是m*m的矩阵，其中每个元素为1/(m^2)，可以看出卷积核的元素总和为1。比如*3的均值滤波器，卷积核的每个元素就是1/9。

高斯滤波器

高斯滤波器是均值滤波器的高级版本，唯一的区别在于，均值滤波器的卷积核的每个元素都相同，而高斯滤波器的卷积核的元素服从高斯分布。

高斯滤波器是基于二维的高斯分布函数，因此首先介绍二维高斯分布函数。二维高斯分布函数和图如下：



$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

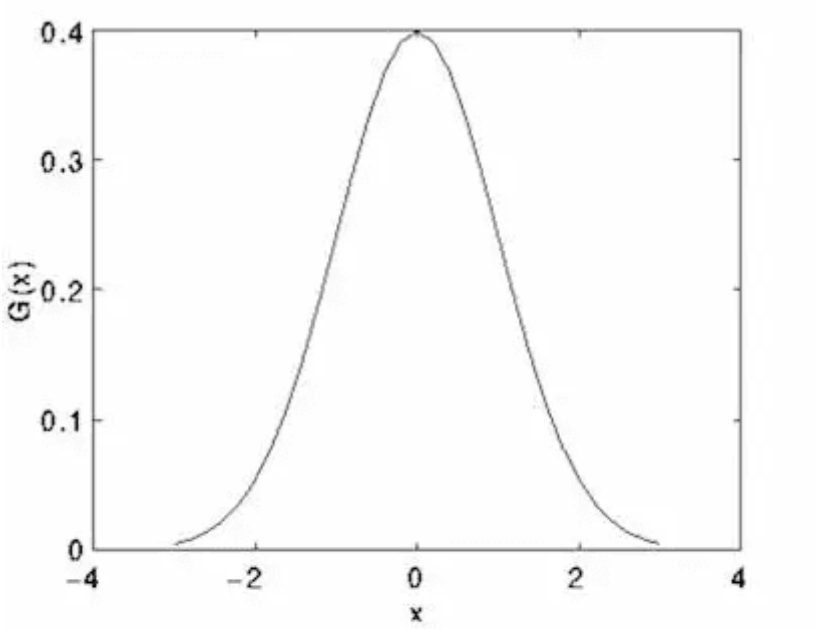
其中x和y表示卷积核中某个元素横坐标和纵坐标距离中心点的距离。sigma控制曲线的平缓程度，值越大，越平缓，最高点越低。我们可以轻易看出当x=0且y=0时值最大，即卷积核的中心点权重最大。

比如卷积核中一个元素距离中心点，横向距离2，纵向距离1，那么x=2,y=1，就能求出该元素的值。当然为了保证卷积核元素总和为1，最后每个元素都需要除以卷积核中所有元素之和。

怎么确定卷积核的大小呢？确定sigma之后，虽然不管距离中心点多远，该元素的高斯分布函数值总为非负数，但是根据经验，卷积核的半径定为3*sigma，因此宽高为*6sigma+1。

如果高斯滤波器的卷积核是二维的（m*n*），*则算法复杂度为O(m*n*M*N)，复杂度较高，因此接下来我们对算法复杂度进行优化。

一维的高斯分布函数和图如下：



$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

实际上，二维高斯分布函数可以分解为两个一维高斯分布函数相乘，如下：

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} * \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{y^2}{2\sigma^2}}$$

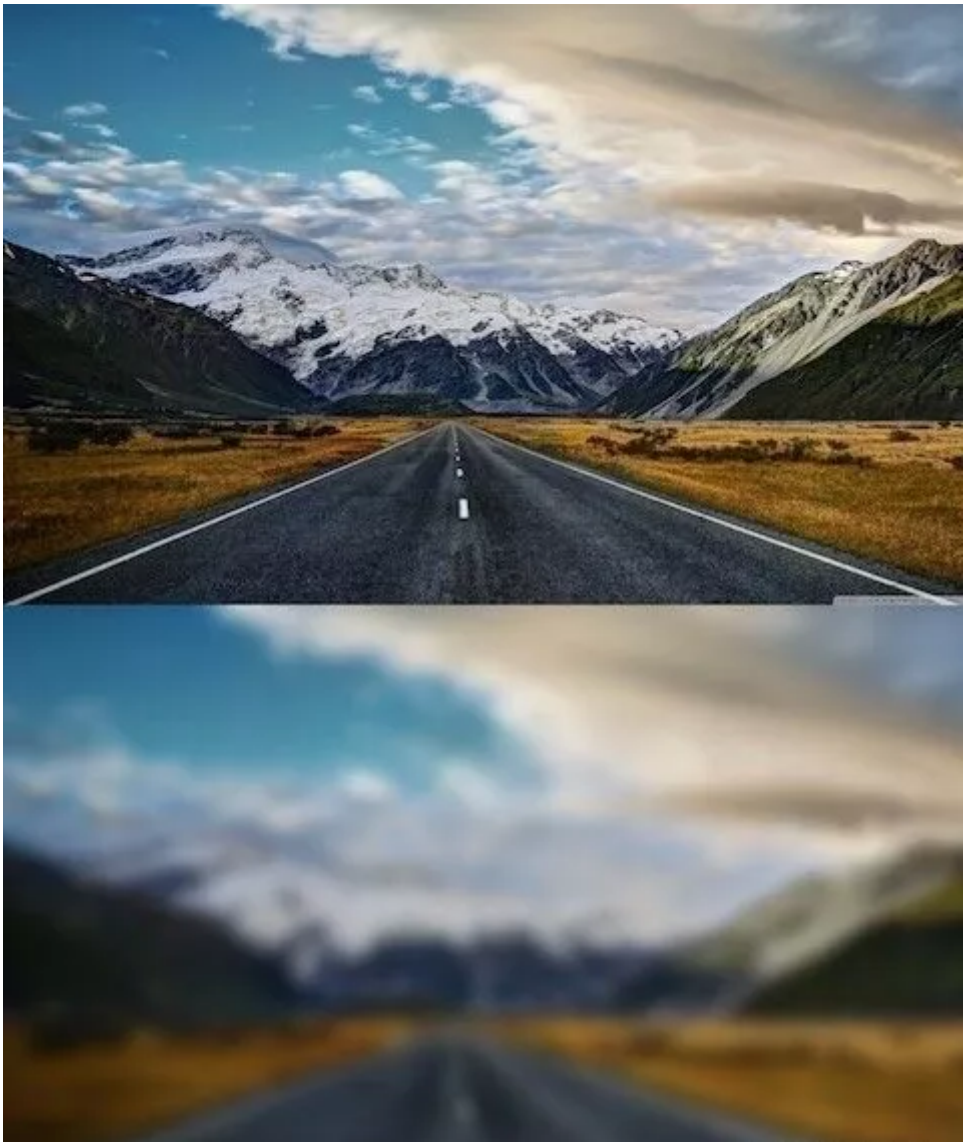
因此原本的源矩阵和二维卷积核做卷积等价于源矩阵先与1*m的一维卷积核做卷积，再与*m1的一维卷积核做卷积。一维卷积核的半径仍定为3*sigma。此时算法复杂度变为O(*2m*M*N)。

高斯模糊的实现

Java版本

这里实现了简单版本的高斯模糊，通过使用横向和纵向的一维高斯滤波器分别对源矩阵卷积，通过设置sigma的大小能控制图片的模糊程度，值越大越模糊。但是算法速度仍比较慢，建议直接使用RenderScript版本或直接使用成熟的开源项目。

效果如下：



RenderScript版本

RenderScript是Android提出的一个计算密集型任务的高性能框架，能并行的处理任务，他可以充分利用多核CPU和GPU，你不需要管怎么调度你的任务，只需要管任务具体做什么。这里不深入介绍RenderScript，因为RenderScript已经提供了一个实现高斯模糊的类：ScriptIntrinsicBlur。

实现起来非常简单：

```
public Bitmap blur(Bitmap bmp) {
    Bitmap result = Bitmap.createBitmap(bmp.getWidth(), bmp.getHeight(), Bitmap.Config.ARGB_8888);
    RenderScript rs = RenderScript.create(this); //创建RenderScript对象
    ScriptIntrinsicBlur blur = ScriptIntrinsicBlur.create(rs, Element.U8_4(rs)); //创建高斯模糊脚本对象
    Allocation in = Allocation.createFromBitmap(rs, bmp);  //输入
    Allocation out = Allocation.createFromBitmap(rs, result); //输出
    blur.setRadius(25f); //设置模糊半径
    blur.setInput(in); //把输入图像传进去
    blur.forEach(out); //执行，并写入到out
    out.copyTo(result); //拷贝到Bitmap中
    rs.destroy();
    return result;
}
```

开源项目

关于Android图像模糊的开源项目有很多，比如Blurry是专门针对Bitmap或View做模糊，可以设置模糊的基底色，而且还能对模糊操作异步化；BlurKit-Android也能对Bitmap做高斯模糊（内部通过RenderScript实现），但最吸引人的是实现了毛玻璃的遮罩，效果如下：

BlurKit-Android支持的最低版本是Android 4.1（API 16），因此如果应用需要支持的最低版本是4.0，则不能使用该库，Blurry支持的最低版本是3.0。

BlurKit-Android

配置过程如下：

- 在build.gradle中设置：`compile 'com.wonderkiln:blurkit:1.0.0'`，并在defaultConfig中设置`renderscriptTargetApi 24`和`renderscriptSupportModeEnabled true`。
- 在Application的onCreate()最开始处加入`BlurKit.init(this);`。

配置完成后，通过调用`BlurKit.getInstance().blur(Bitmap src, int radius);`实现高斯模糊，并会把高斯模糊的结果图写入src，其中0<radius<=25。

该库还提供了`fastBlur()`实现速度更快的高斯模糊，和`blur()`的区别在于，`fastBlur()`在高斯模糊之前对图片采样，使得图片大小缩小好几倍，从而加快高斯模糊的速度。这种加快速度的方法是合理的，因为高斯模糊并不需要原图像很精确的信息。

BlurKit-Android最吸引人的是提供高斯模糊的遮罩（BlurLayout），随着遮罩下面的内容的变化，高斯模糊效果也会随之改变。使用如下：

```
<com.wonderkiln.blurkit.BlurLayout
    android:id="@+id/blurLayout"
    android:layout_width="150dp"
    android:layout_height="150dp" />
```

该Layout能够实现实时的对该Layout下面的内容做高斯模糊。

Blurry

配置方法：在build.gradle中添加`compile 'jp.wasabeef:blurry:2.1.1'`。

使用方法如下：

```
Blurry.with(this)
    .radius(10)      //值越大越模糊
    .sampling(2)     //对原图像抽样
    .async()         //异步
    .from(Bitmap bmp)    //对bmp做高斯模糊
    .into(ImageView view); //把结果写入
```

总的来说，这两个库都使用起来非常方便。