第一种, 直接把高斯模糊的原理翻译成代码, 即给定半径radius模糊系数sigma, 计算宽高都是radius\*2+1的高斯核矩阵, 计算每个像素的值.

计算过程中需要保存radius+1行的图片buffer, 因为把模糊过的值写到当前点的缓存中, 后面的点还会用到这一点的原始值.

整个图片都用memcpy复制过一遍.

复杂度图片宽\*图片高\*(半径\*2+1) \*(半径\*2+1)

第二种, 水平方向模糊一次, 垂直方向模糊一次. 给定半径radius, 模糊系数sigma, 计算长度为radius\*2+1的一维高斯核(向量?), 计算每个像素的值, 第一个方向模糊过的输出作为第二个方向模糊的输入.

计算过程中还是需要保存至少前面radius+1个点的原始值. 我直接保存一行了.

整个图片都复制过两遍. 垂直方向(不连续的方向)要在内存中跳转, 挺不好的.

复杂度图片宽\*图片高\*(半径\*2+1)\*2

第二种半, 这是想象出来的方法: 不想保存原始图片的buffer, 那么其实当前图片的每个模糊过的点都包含了原始图片的信息, 可以倒推出原始图片的像素值, 然后推了一下, 发现复杂度比第二种还大, 因为从第0个点到第i个模糊过的点都对第i个点有影响, 也就是说计算第i+1个点的时候要把前面i个点都加权求和.

又搜集了一通, 发现大部分中文博客, 都满足于, 把前两种的原理讲清楚......

第三种, 第一遍从前往后算, 第二遍从后往前算, 两个方向各算一次, 一共四次,

w(n) = a0\*x(n) + a1\*x(n-1) – b1\*w(n-1) – b2\*w(n-2), 从前往后

y(n) = a2\*x(n) + a3\*x(n+1) – b1\*y(n+1) – b2\*y(n+2), 从后往前

粗略估计一下信息是不会丢失的.