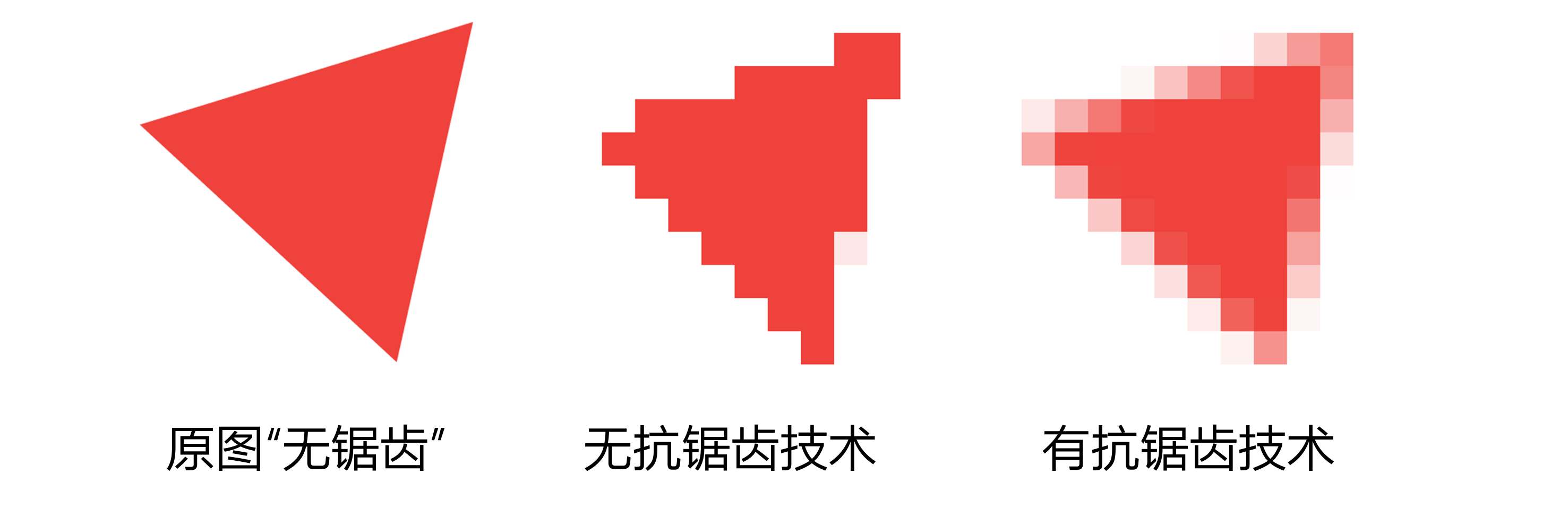
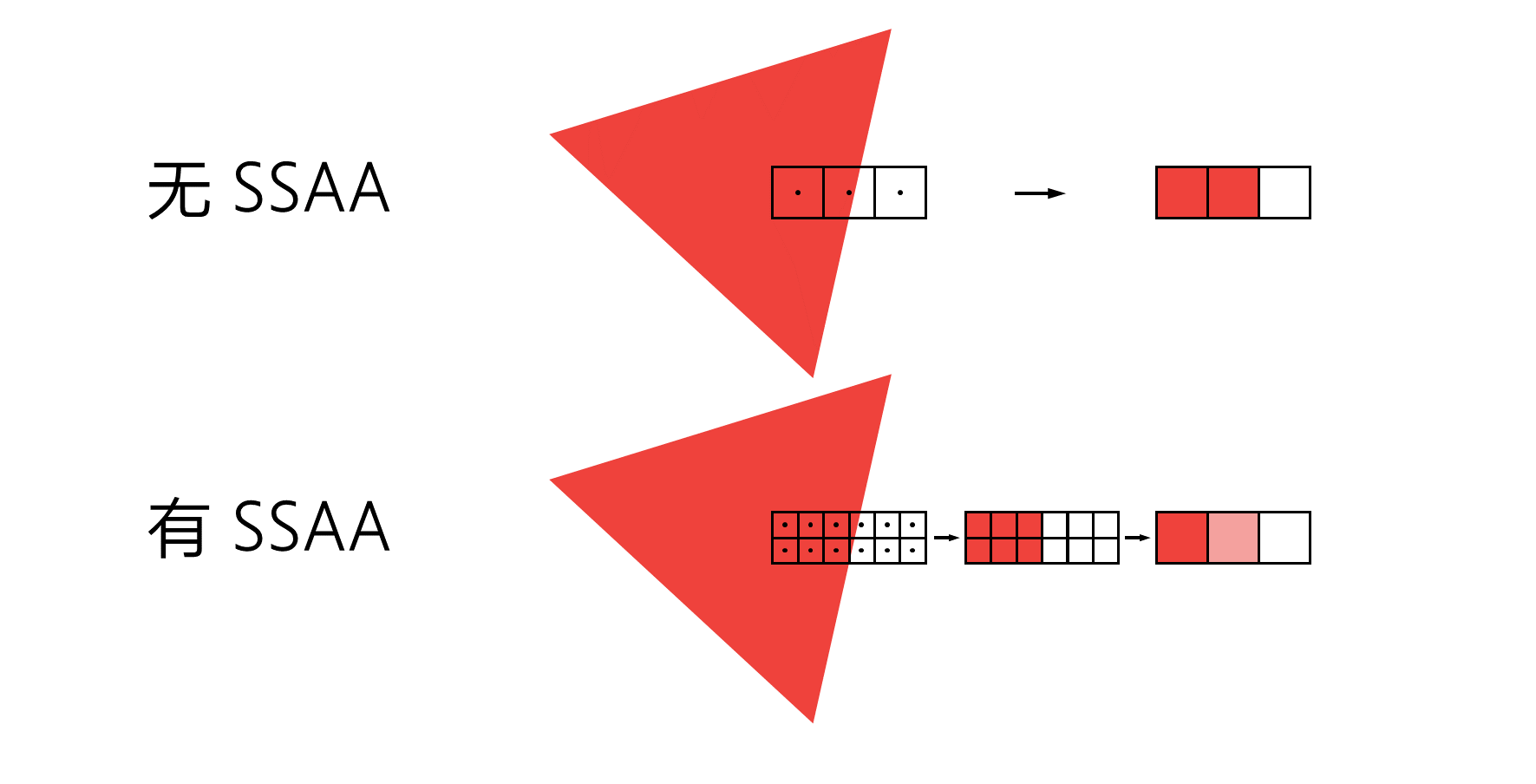
笔者最近在回顾一些图形学基础知识，遂整理在此，此文涉及图形学中的着色。

抗锯齿技术在渲染着色中十分重要，本文介绍一些空间上的抗锯齿技术，如：SSAA、MSAA、CSAA、DEAAA、MLAA、SRAA、FXAA、SMAA。有无抗锯齿的效果图如下所示：

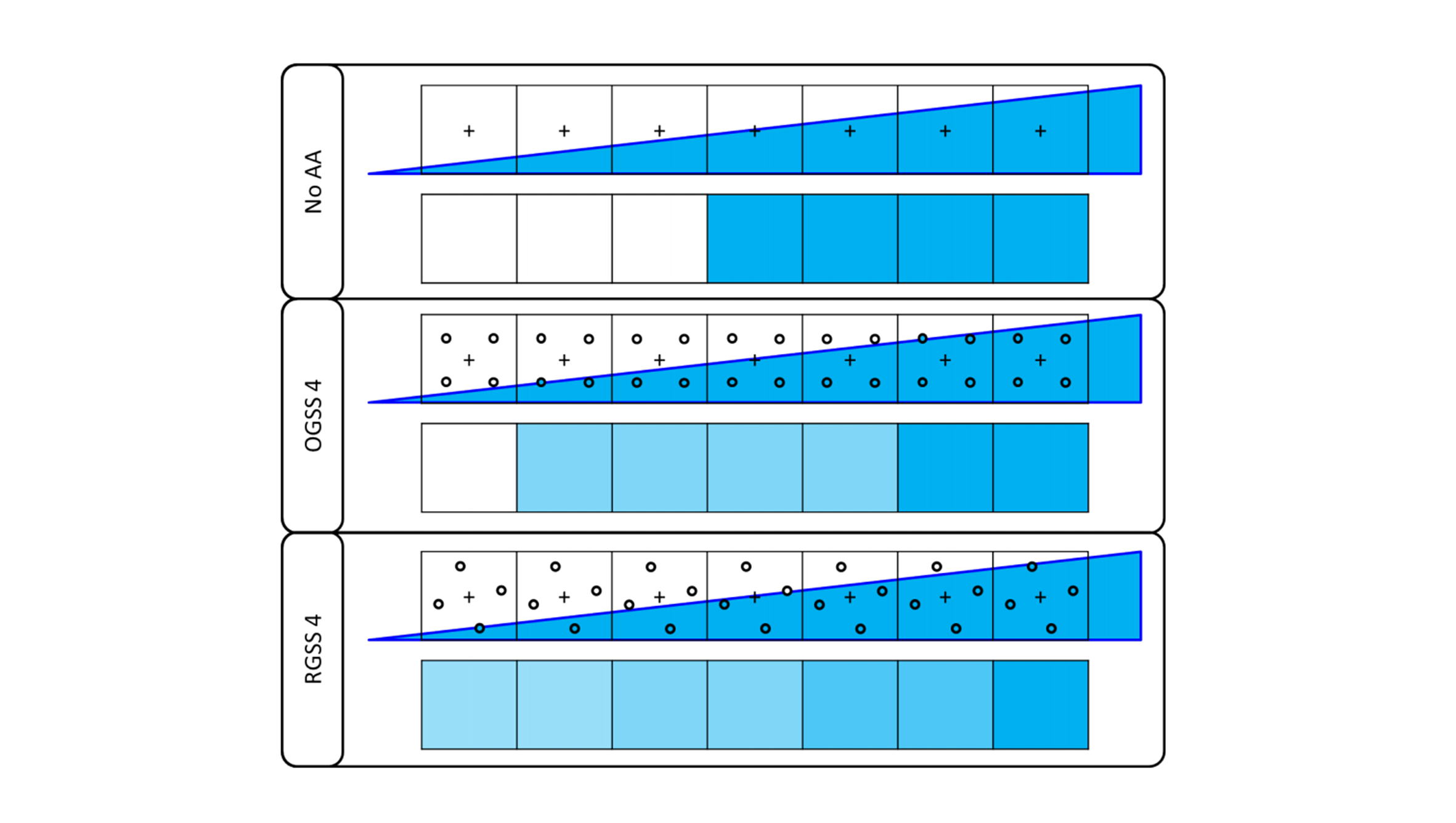


SSAA(Super-Sampling AA)

这是最简单粗暴的抗锯齿技术。基本思想是：先将场景渲染到一个更高分辨率的帧缓存上，然后分别局部计算多个点的均值得到每个点的颜色。比如目标是得到1280\*1024的图像，就先渲染3D场景到2560\*2048的帧缓冲上(4\*SSAA)，然后再滤波解析(Resolve，常用的是Box滤波器)，得到1280\*1024的图像，如下图所示：



从上图中可以看到，相当于将一个像素细化为四个采样点，此处的四个采样点分布有序且均匀，此类超采样称之为OGSS(Ordered Grid Super-Sampling)。有研究者发现OGSS对于竖直和横向方向的超采样表现不太好（考虑到很多边缘都是竖直或者横向的），于是将四个采样旋转一个角度，得到RGSS(Rotated Grid Super-Sampling)，下图比较了OGSS与RGSS的表现：

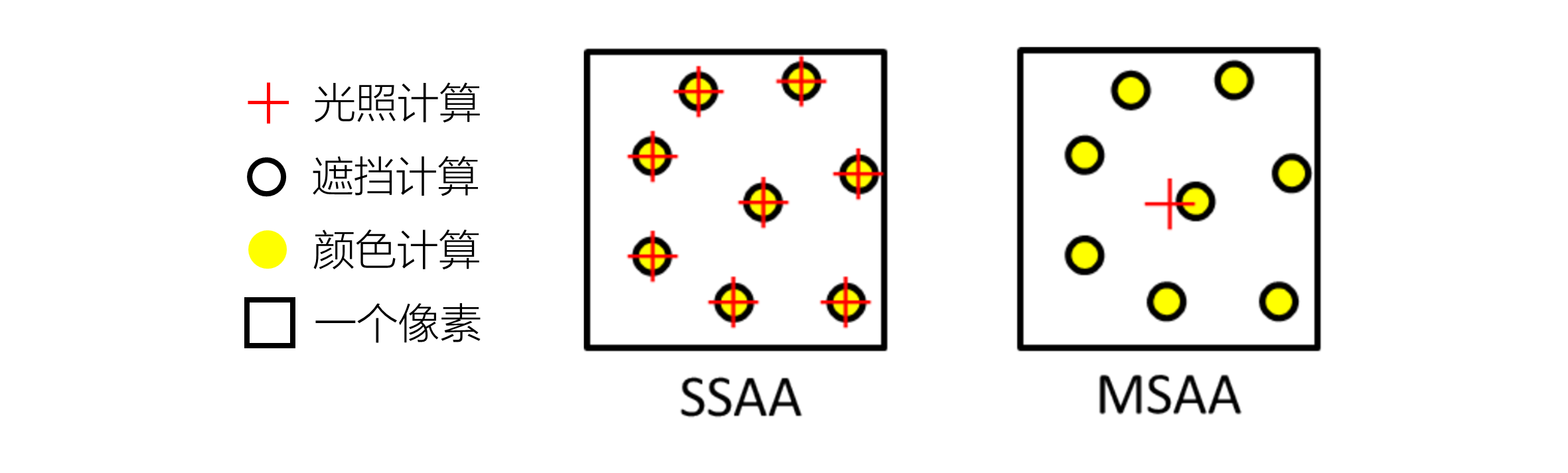


此外，还有其他的一些采样点分布方式，统称为采样模板(Sampling Pattern)，此处不详细展开。

SSAA的原理十分简单，只要提高超采样的倍数，效果也是最好的，但是其计算量往往不能被实时渲染所接受，所以经常作为GroundTruth来对其他方法进行验证

MSAA(Multi-Sampling AA)

相比SSAA，MSAA的优势是把采样点的深度、遮挡和光照计算分开，减少光照计算量。SSAA是对每一个采样点都计算一次深度、遮挡和光照，最后才进行滤波解析，而MSAA只对每一个采样点计算深度、遮挡等，然后就滤波解析，最后对一个像素内的多个采样点只计算一次光照。下图对比了两者的区别：



从上图可以看到，MSAA相比SSAA少了很多的光照计算，所以速度提升不少，是使用最普遍的一种抗锯齿技术。

此处MSAA对一个像素计算一次光照的时候，选取的是像素的中心点，当然也可以对此点进行偏移，分别称之为Center Sampling和Centroid Sampling，后者的效果更好但计算量更大，关于两者的区别可参考[1]，此处不展开。

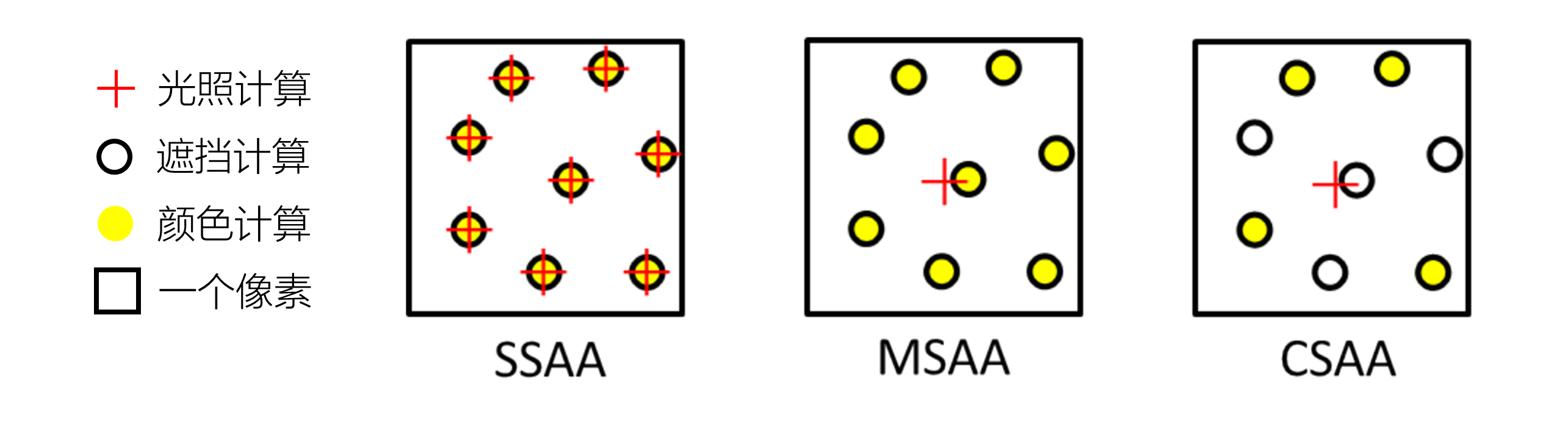
MSAA虽然相比SSAA提高的性能，但是存储占用与SSAA相同，下面的CSAA则进一步改进了这一点。

CSAA(Coverage Sampling AA)

CSAA是NVIDIA提出的，其主要思想是：进一步减少每个采样点的信息量。

回顾前面：SSAA是对每一个采样点计算深度、遮挡、颜色和光照，然后解析。

MSAA是对每一个采样点计算深度、遮挡、颜色，先解析，然后一个像素只计算一次光照，CSAA与MSAA的区别在于，进一步减少计算颜色的采样点，如下图所示：



CSAA相比MSAA少计算了四个点的颜色，所以存储空间减少了，但抗锯齿效果仍然不错，CSAA参考NVIDIA的白皮书[2]

EQAA(Enhanced Quality AA)则是AMD对标NVIDIA的CSAA开发的，两者原理一致，不赘述

DAEAA(Directionally-Adaptive Edge AA)

DAEAA技术对比MSAA，则是对边缘处的像素又做了不同的处理。其主要思想为：首先找到位于边缘处的像素，然后使用最小二乘法将边缘拟合为线段，对线段边缘两侧的采样点（像素内和像素外）进行加权求和，权重与采样点到边缘的距离呈负相关，其他处理和MSAA一致。DAEAA与MSAA的对比如下：



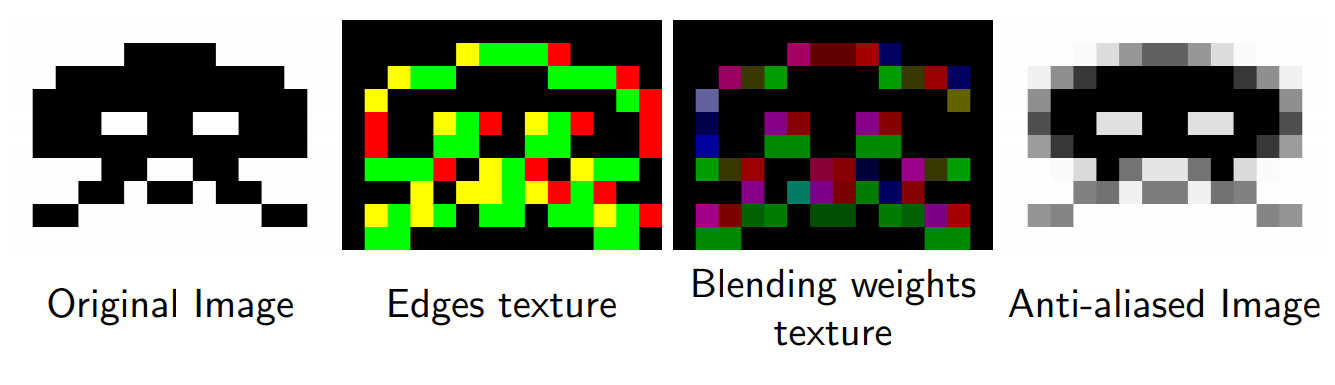
虽然此技术的效果好于MSAA，但有严重的性能问题：

1. 需要四次shader pass：分别用来1）确定边缘像素；2）用边缘模板筛选边缘像素，使边缘在一个像素内没有转角；3）计算边缘像素的梯度；4）根据梯度对采样点进行加权求和
2. 由于此技术着重处理了边缘像素，所以当画面的边缘数量变动较大时，处理时间也会随之波动，可能会导致帧率不稳定

DAEAA技术可参考[3]

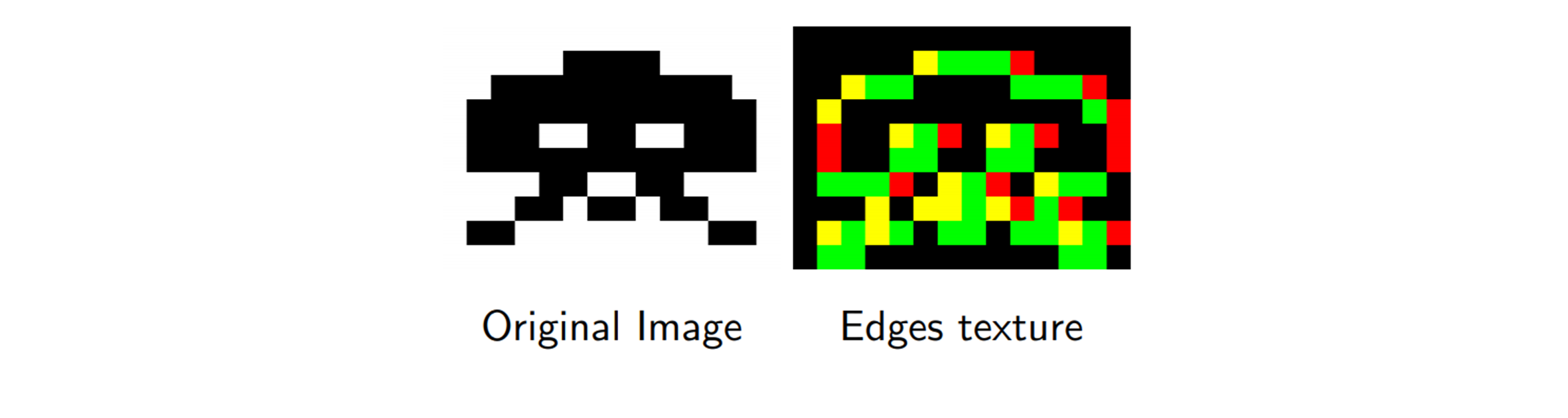
MLAA(Morphological AA)

形态学上的抗锯齿技术，尝试根据边缘的形态学信息进行抗锯齿。这类技术本质上是一种后处理，所以可以用在延迟渲染，弥补了MSAA不能用于延迟渲染的短板。最早的MLAA技术是Reshetov在2009年提出的，是在CPU上实现的为了解决光线追踪的抗锯齿技术。而后Jimenez在2011年提出了在GPU上实现的MLAA，此处我们主要介绍后者，即Jimenez’s MLAA，下图展示了这一技术的大致过程：

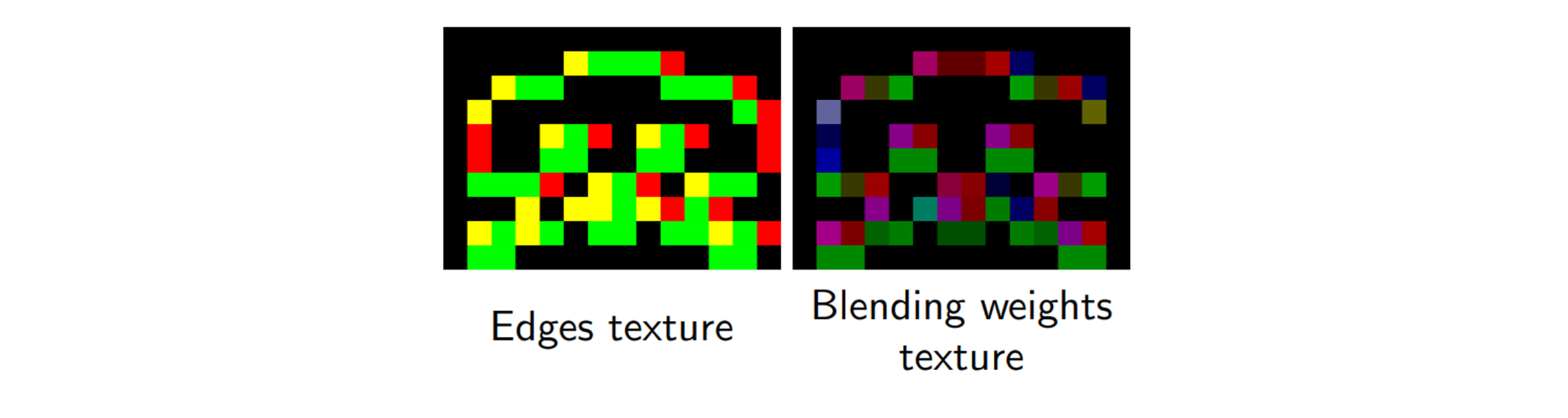


此技术有三个步骤：

1. 边缘检测。有两种方法，一是根据深度图检测边缘，二是根据RGB值的加权得到的亮度(luminance)来检测边缘。具体实现是在Pixel Shader里面，用当前点的颜色分别减去左边像素和上边像素的值，取绝对值后分别存在一张纹理的R和G通道，所以得到的Edges texture里面的红色点代表其左边为边缘，绿色点代表其上边为边缘，黄色点代表其左边和上边为边缘，如下图所示：



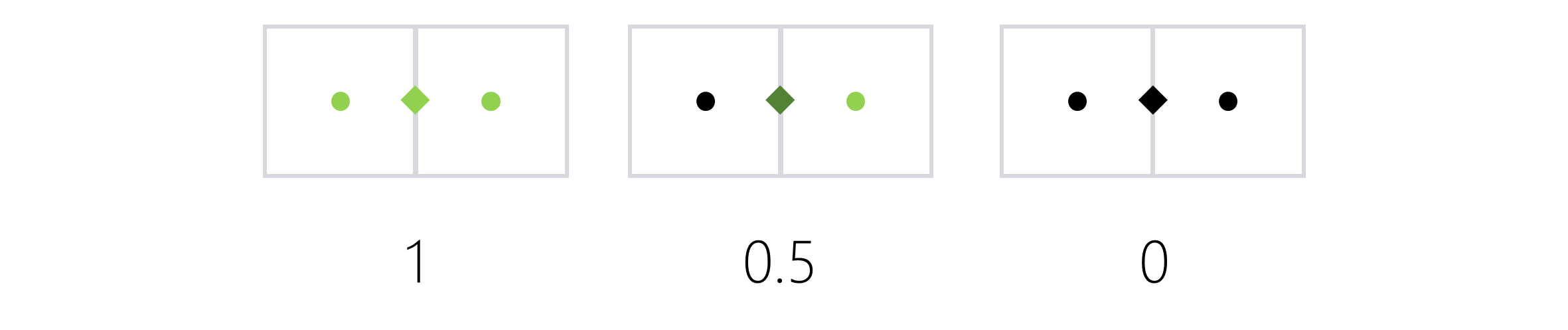
1. 计算混合权重。如下图所示：



来分析这一过程，我们知道，锯齿主要发生在边缘的转角处，所以比如一条横着的长边缘，中间的点的混合程度就要比两端的点混合程度要弱，并且不同的端点处的形状的混合程度也不一样。混合程度体现为混合权重，所以需要根据边缘的：a）长度 b）形状，来确定混合权重

1）确定Edges texture中边缘的长度和形状。

因为边缘的纵向和横向信息存在R和G分量，所以为了简化计算量，只在R分量为1的点计算此点到边缘上下端点的距离，只在G分量为1的点计算此点到边缘左右端点的距离。一个简单的思路就是沿着当前点向左或者向右逐个检查，直到发现G分量突然由1变成0就到达了横向边缘的端点。在GPU里面，Jimenez则利用了硬件双线性插值(Hardware bilinear)的特性。以向左查找端点为例，先将这个点向左偏移1.5个像素，新的一点值就是旁边两个像素的插值，有如下对应关系：



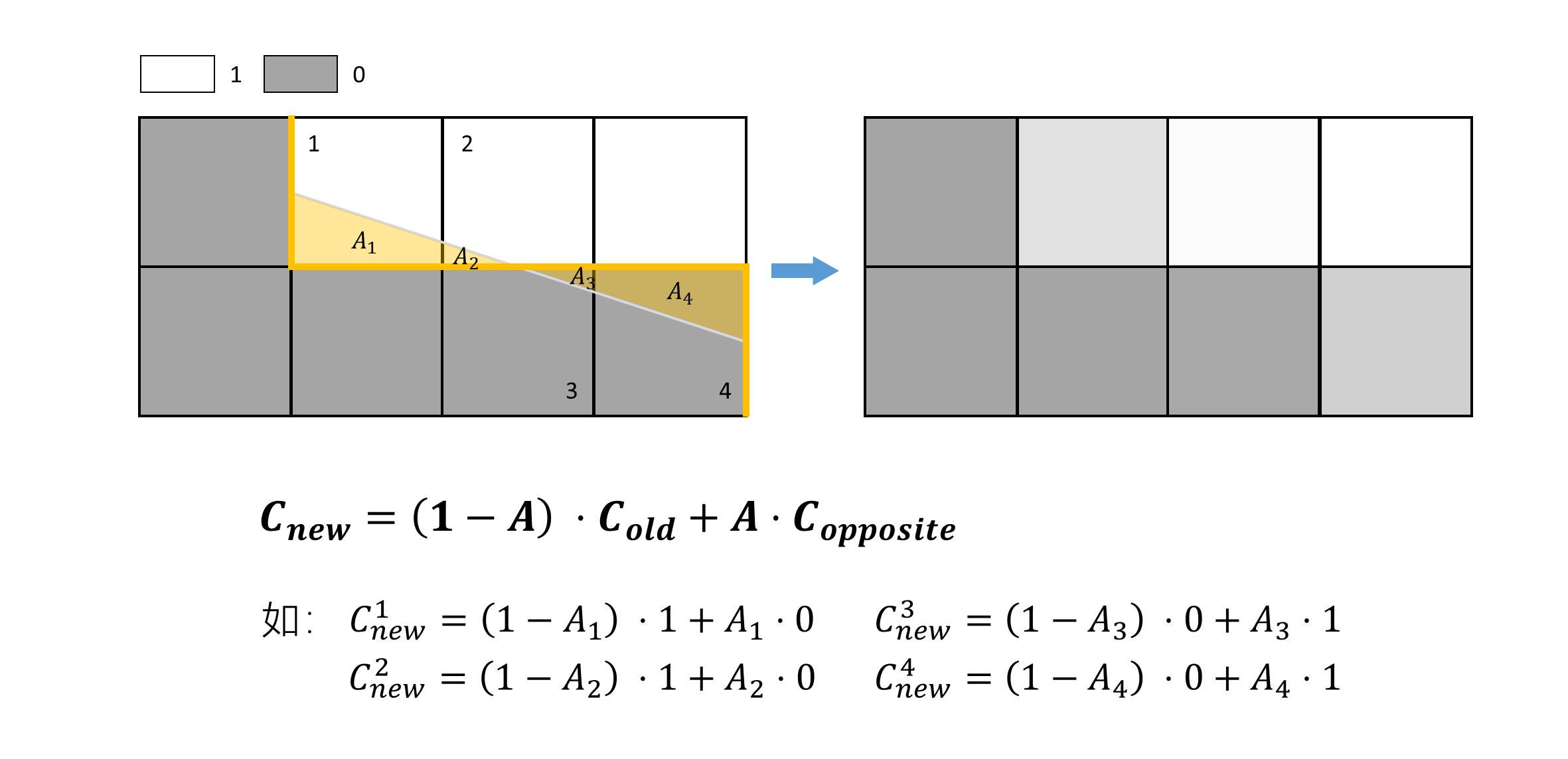
然后以2个像素的步长，并设置最大步数为8，向左查找，直到发现这个中间值小于1或者达到最大步数为止。于是可以得到此点与边缘左端点的距离，同理也可得到与右端点的距离，以及对R分量执行上下查找操作，可得到此点与边缘上下端点的距离。

对于边缘形状的确定，Jimenez的策略是在边缘端点处沿垂直方向偏移0.25个像素，根据此处的值来判断形状。比如正在向右查找边缘的端点，根据G分量找到端点后，向下偏移0.25个像素，再判断此点的R分量，如下图有四种对应关系：

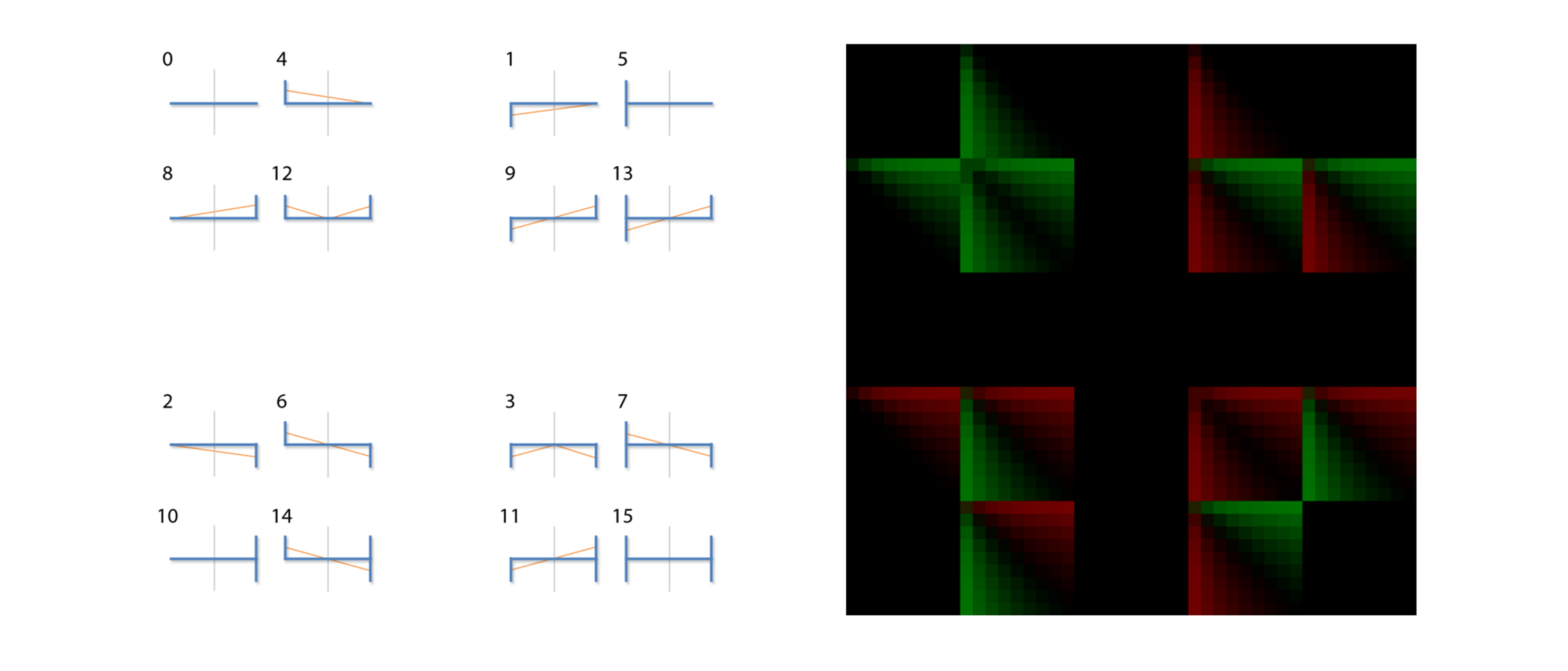


2）根据边缘的长度和形状计算权重。

确定了边缘的长度和形状，就可以跟这两个因素来确定边缘上每一点混合的权重，在Reshetov’s MLAA中是计算面积确定的权重：

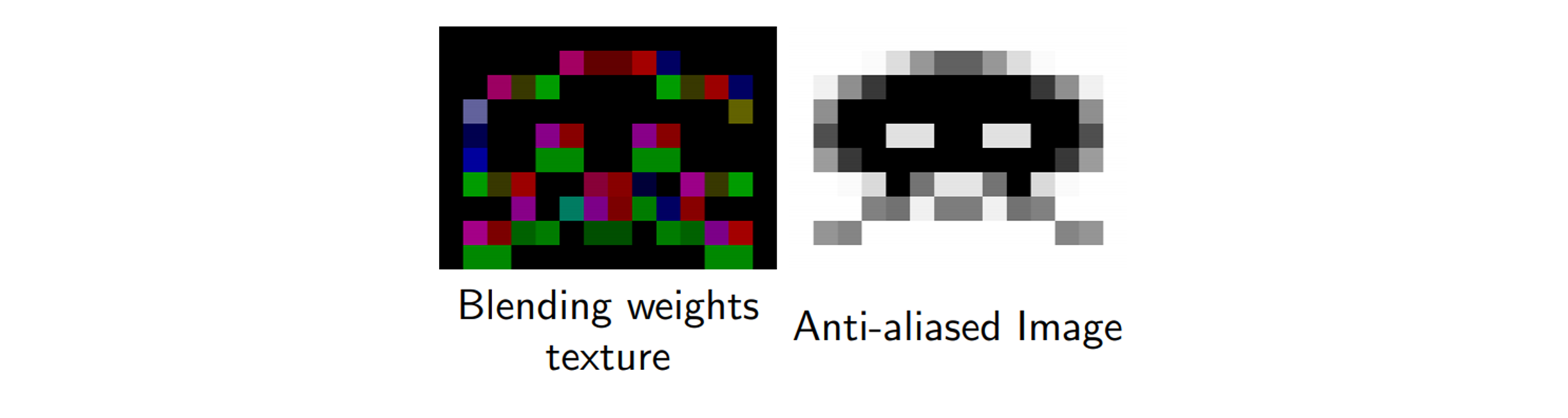


而Jimenez不是在运行时计算权重值，则是事先计算好。因为边缘的长度由于计算步数的限制，故边缘长度最大为8，而边缘端点处形状则限制为4种，故所有长度和形状的边缘对应的权重值，都可以保存在一张纹理中，并实时查找对应的值：

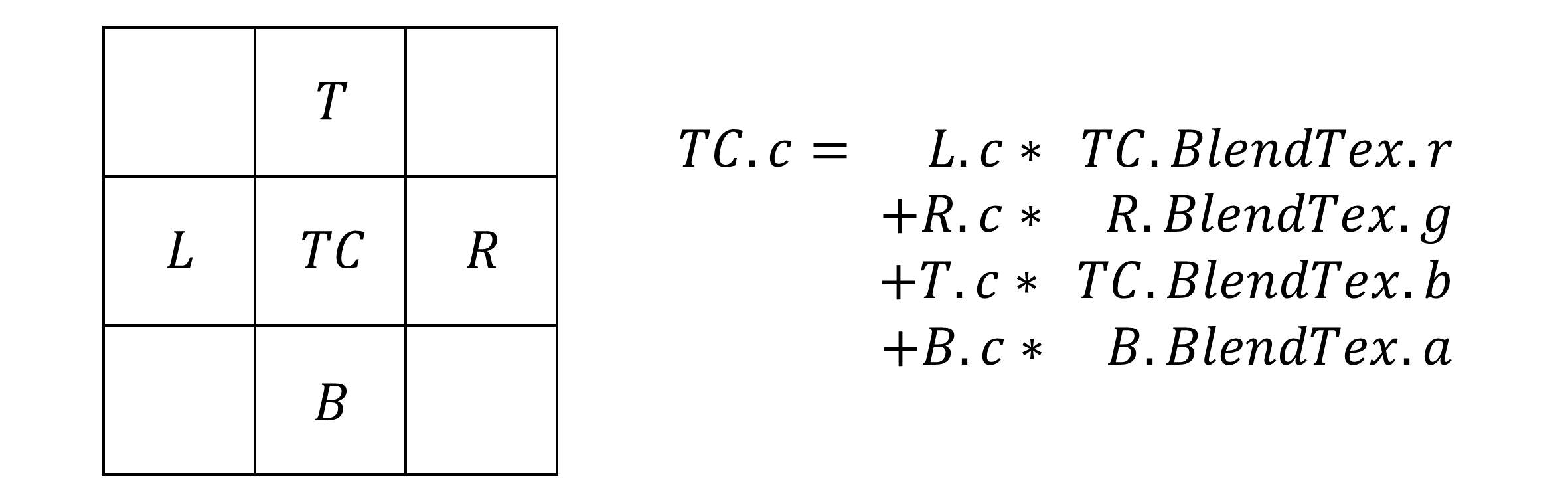


分别对左右型边缘和上下型边缘进行权重计算，并分别将四个权重存储在Blending weights texture里的RGBA通道里面。

1. 根据权重对颜色进行混合。



论文里用了周围四个点结合权重进行了混合，我理解的代码意思如下：

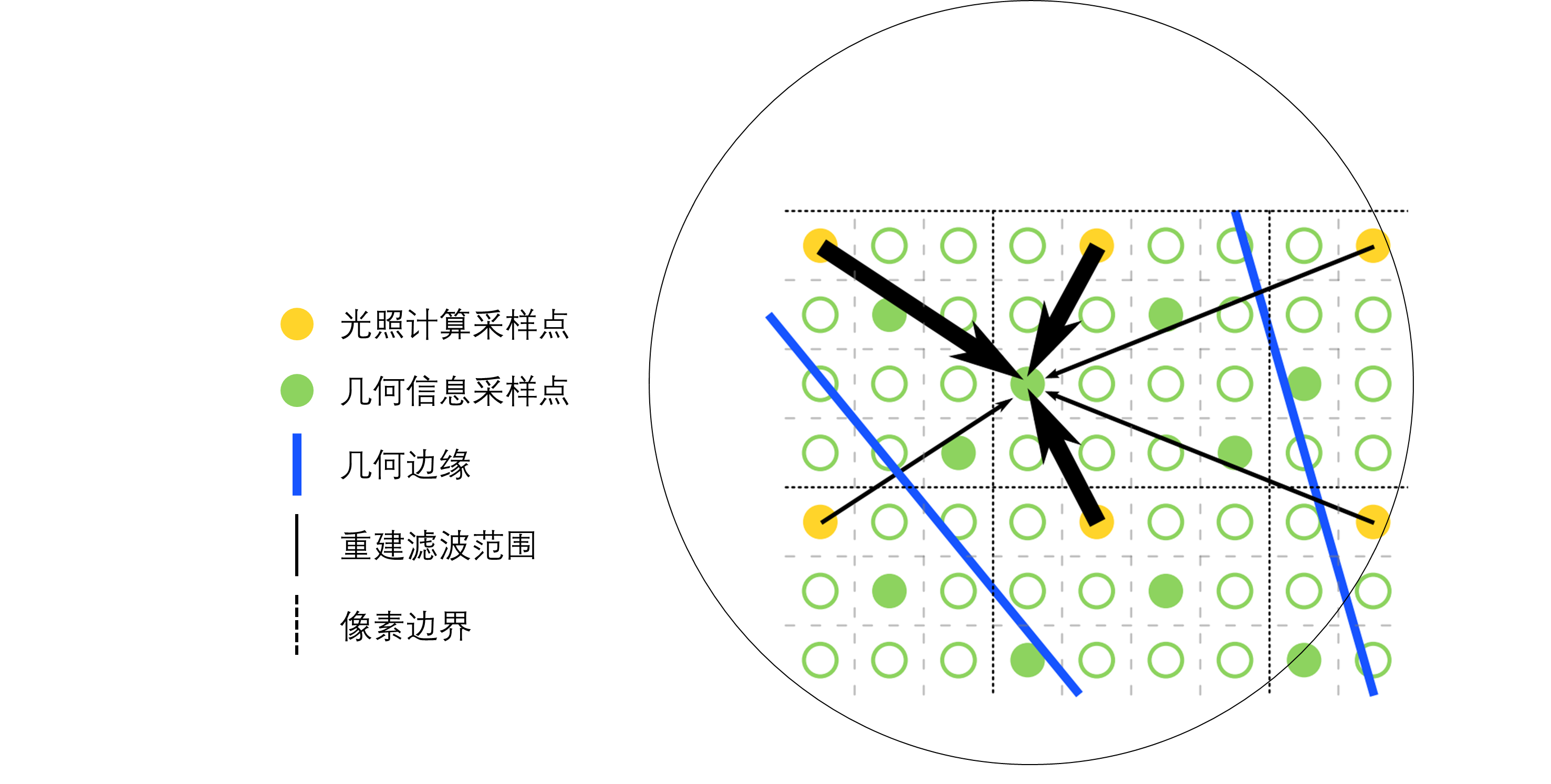


Reshetov’s MLAA与Jimenez’s MLAA的具体细节分别可参考[4][5]

SRAA(Subpixel Reconstruction AA)

子像素重建的抗锯齿，这也是适用于延迟渲染的技术，此技术相比于标准渲染管线，有两个额外的开销：

1. 必须在子像素层面上生成深度、法线和位置信息的帧缓冲；
2. 在逐像素计算光照后，还需要对子像素进行重建，并滤波降采样到屏幕分辨率



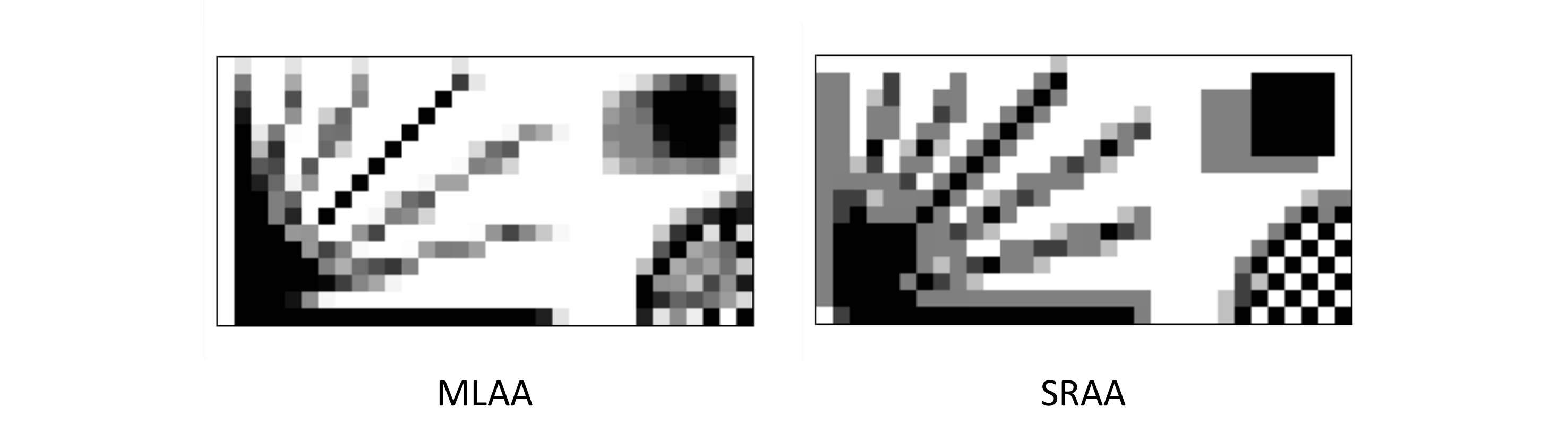
基本思想如上图所示：

1、先通过对投影矩阵设置微小偏移量，快速得到子像素层面的几何信息采样点的法线和位置信息，即图中的绿色采样点。

2、然后逐像素计算光照后，再划定一个圆形范围，用双边滤波器(bilateral filter)对子像素进行重建。此处一个很重要的点就是：几何边缘外部的光照采样点的权重小于内部的采样点，这是由于权重是由距离结合深度算出来的，而深度图一般也蕴含了边缘信息，并且双边滤波器对于边缘和非边缘处有不同的权重处理，所以相比MLAA更能够保留边缘。

3、重建子像素后，再借助简单的单个像素范围内的box滤波器实现降采样，以得到屏幕分辨率的图像

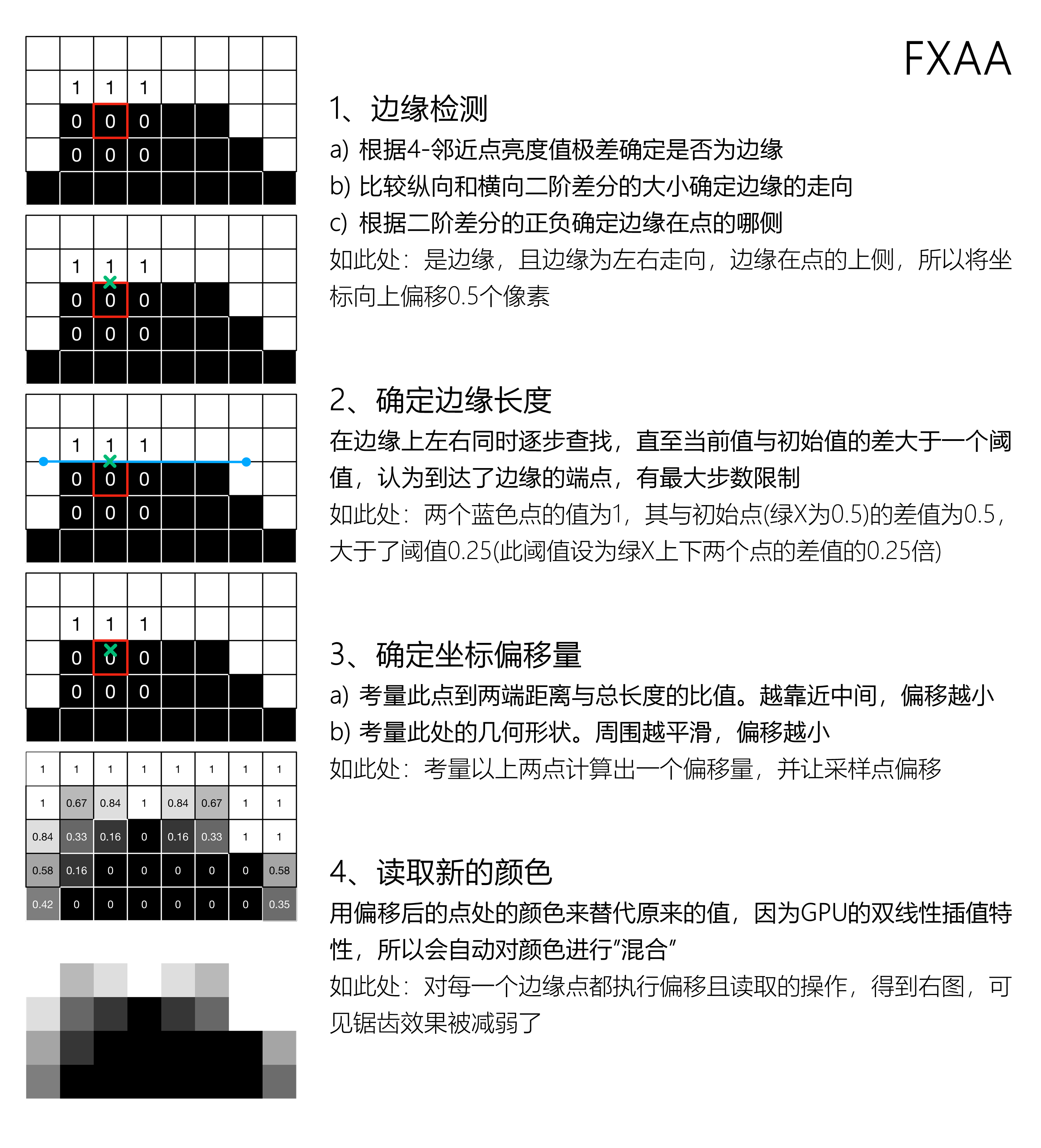
SRAA与MLAA的对比见下图：



可见，SRAA没有让边缘过于模糊，很好的保留了几何信息，更多SRAA的细节可参考[6]

FXAA(Fast Approximate AA)

FXAA是一个后处理技术，只需要一张渲染好的屏幕分辨率大小图像即可。一张图过一下FXAA的大致流程：



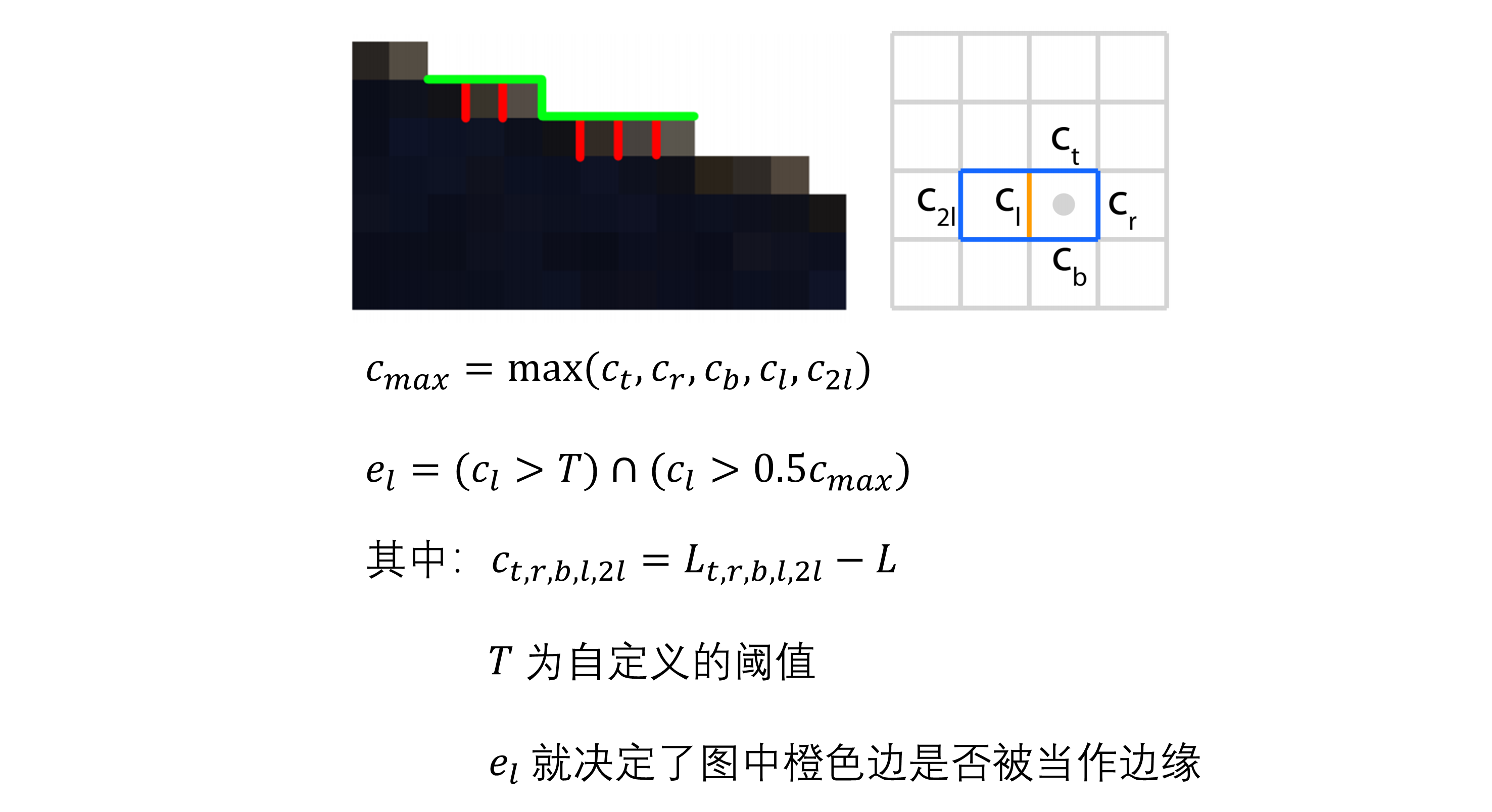
关于FXAA的具体实现细节，参考[7][8]

SMAA(Enhanced Subpixel Morphological AA)

SMAA其实是Jimenez对他自己提出的MLAA的改进，主要有以下几处：

1. 边缘检测的改进：

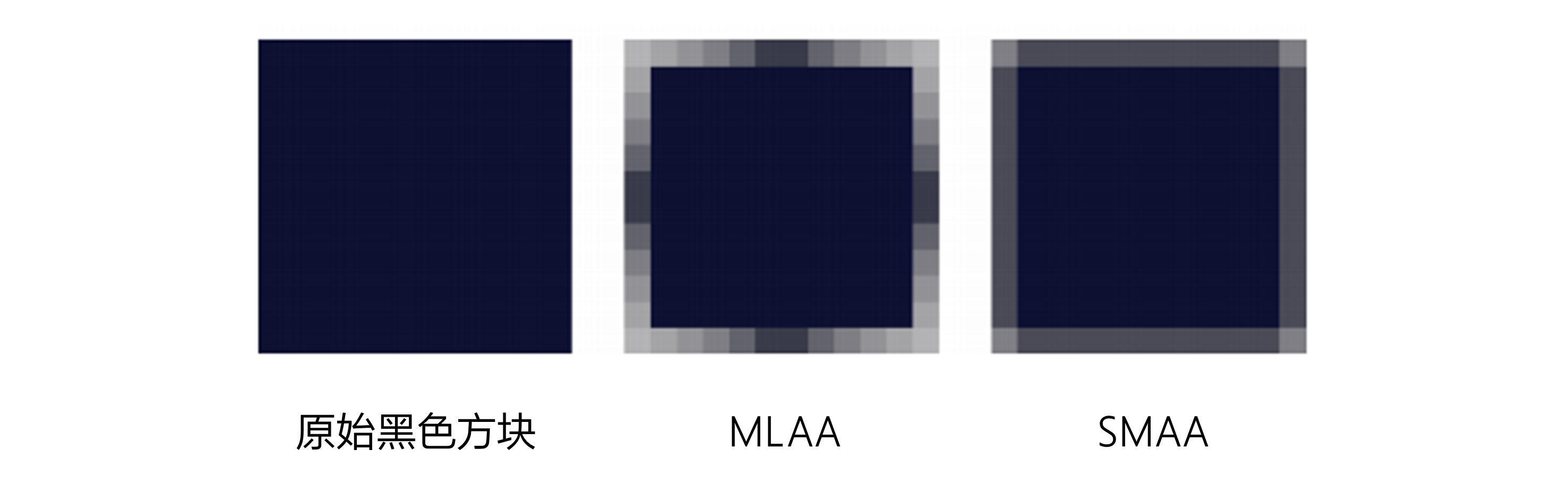
都是对像素左侧和上侧的边进行边缘检测，但又考虑了局部的像素对比，比如在下图中，红色边缘相比绿色边缘就不那么明显，所以在边缘检测的时候引入了额外的条件：



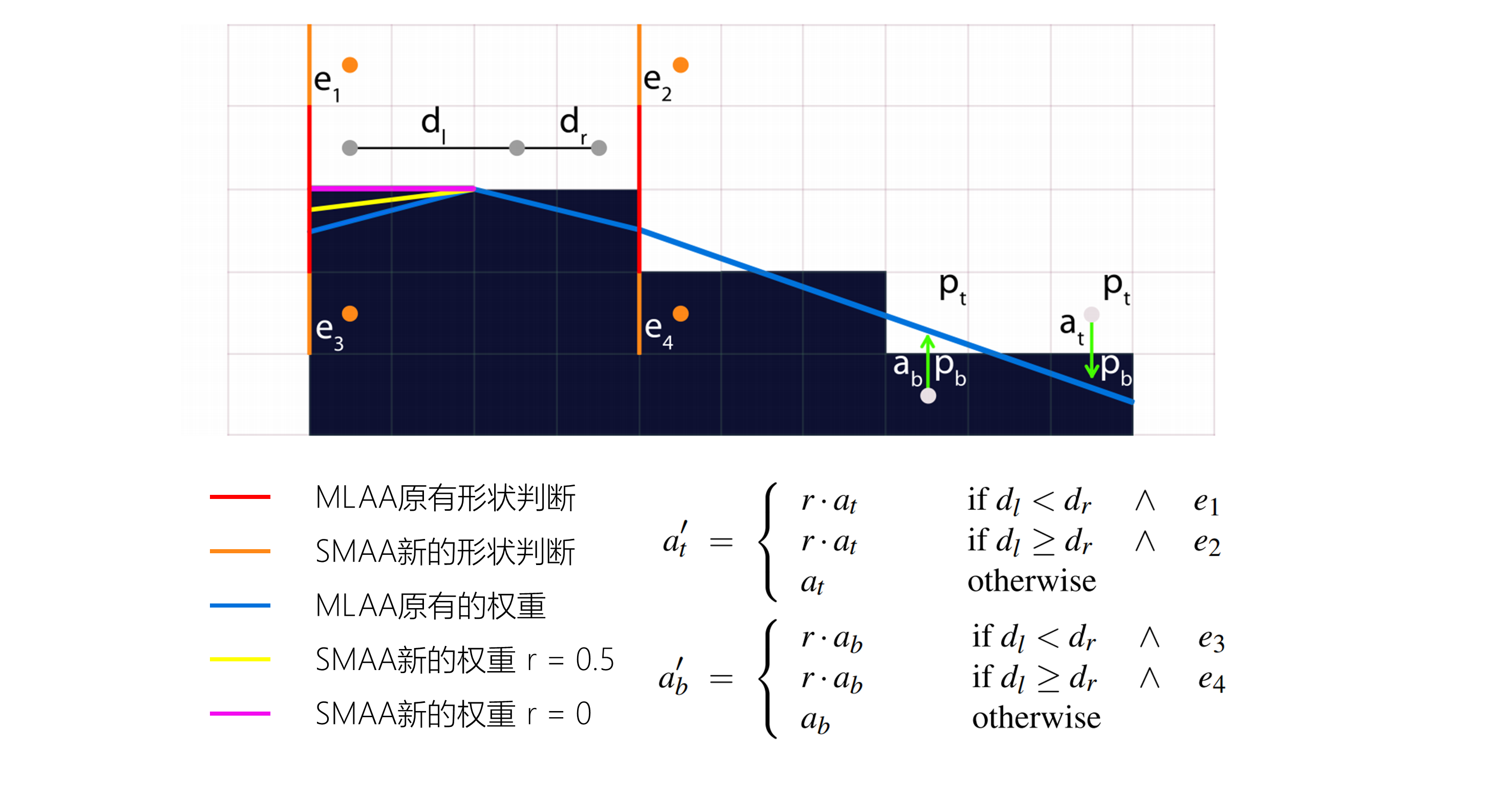
此外还有像素上侧的边缘检测，与之类似。

1. 提取更多几何信息，保留不该模糊的边缘。

如下图所示，一个方方正正的黑色方块，按理说是不该被抗锯齿技术模糊的，MLAA没有考虑到这一点，可见其边缘仍然被模糊了，而SMAA改进了这个细节。



改进的操作如下：在检测边缘端点处形状的时候，MLAA里面只对邻近的一个像素做了判断，而SMAA里面则考虑了两个像素。如下图所示，左边e3处连续三个像素下来都是直线，没有像右边e4处，才下到一个像素就向右呈现阶梯状，所以把左边认为不是锯齿，右边认为是锯齿。



上图红线代表原来MLAA中对于形状的检测，而橙线则是SMAA引进的对于是否为锯齿的检测，蓝线代表原来MLAA中的面积(权重)计算，而黄线和粉线则是SMAA中根据不同的因子r重新确定的面积(权重)。可见在黄线或者粉线处，本就不该是锯齿，所以也就减弱了混合权重。

1. 引入了对角线的边缘检测。

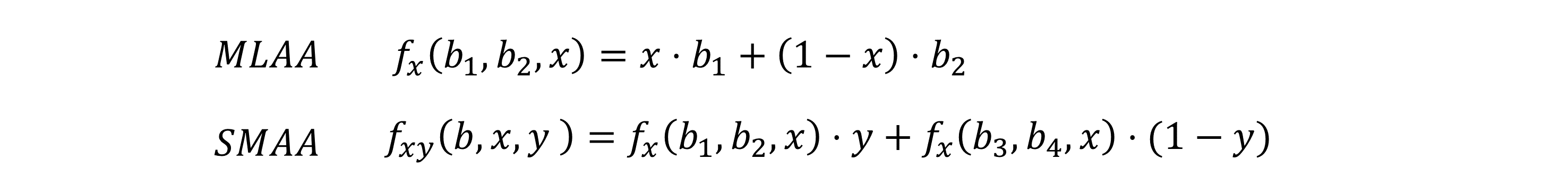
在MLAA中只有横竖两个方向的边缘检测，而在SMAA中则引入了对角线方向的边缘检测：



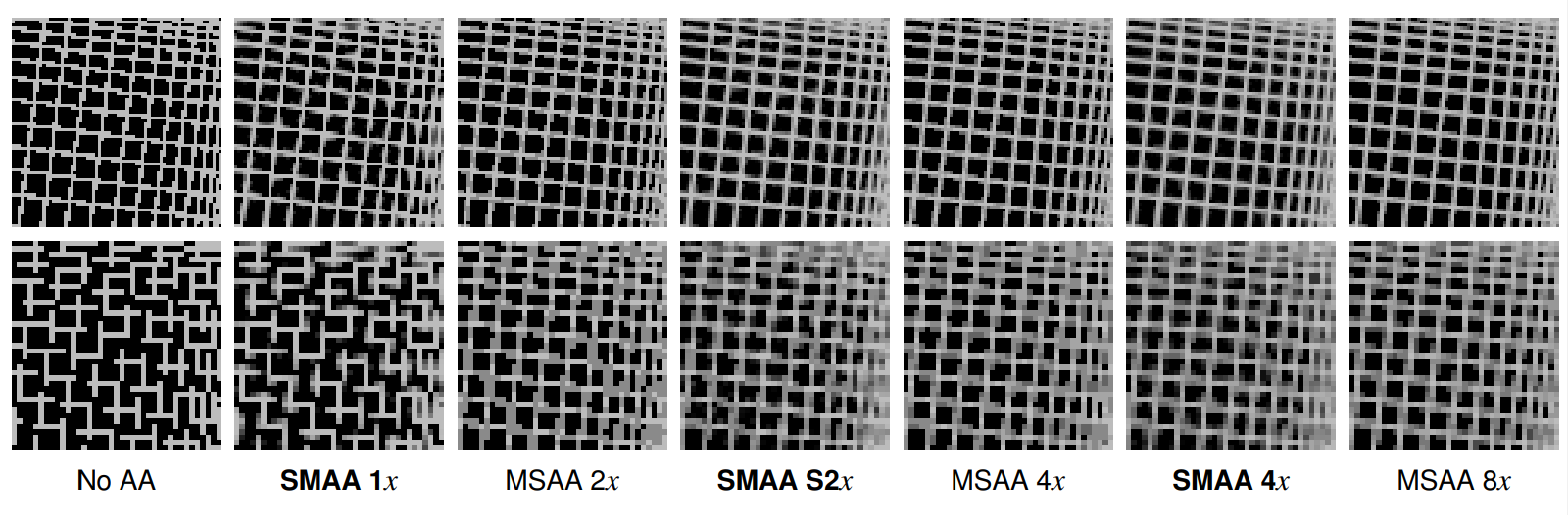
比如对于上图这种边缘则适合被当作是一条“斜边缘”，SMAA中增加了对角线边缘的检测，也类似地用一张纹理来保存相关的混合权重。

1. 更改了由面积计算权重的公式。

MLAA里面在颜色混合时，对于一条边缘，只考虑边缘两侧颜色的混合，而SMAA里面对于一条边缘，则考虑了边缘两侧以及垂直方向上另外两个颜色的混合。



除了这四点改进之外，SMAA还与MSAA及TAA进行了结合，实现了更为细腻的抗锯齿效果，效果展示如下：



更多具体细节可参考[9]

本文从最简单的SSAA出发，介绍了许多种空间抗锯齿技术，除此之外，还有时间上的抗锯齿技术和基于深度学习超分辨的抗锯齿技术，会在下一篇文章讲述。抗锯齿的技术很多，我也在学习阶段，文中有所错误还请多多指教！

[1] <https://www.opengl.org/pipeline/article/vol003_6/>

[2] <https://developer.download.nvidia.cn/SDK/10/direct3d/Source/CSAATutorial/doc/CSAATutorial.pdf>

[3] <https://dl.acm.org/doi/pdf/10.1145/1572769.1572789>

[4] http://www.cs.cmu.edu/afs/cs/academic/class/15869-f11/www/readings/reshetov09\_mlaa.pdf

[5]<https://graphics.unizar.es/papers/JimenezMLAA.pdf>

[6] <https://research.nvidia.com/sites/default/files/pubs/2011-02_Subpixel-Reconstruction-Antialiasing/I3D11.pdf>

[7] <http://blog.simonrodriguez.fr/articles/30-07-2016_implementing_fxaa.html>

[8]<https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf>

[9] <http://www.iryoku.com/smaa/downloads/SMAA-Enhanced-Subpixel-Morphological-Antialiasing.pdf>