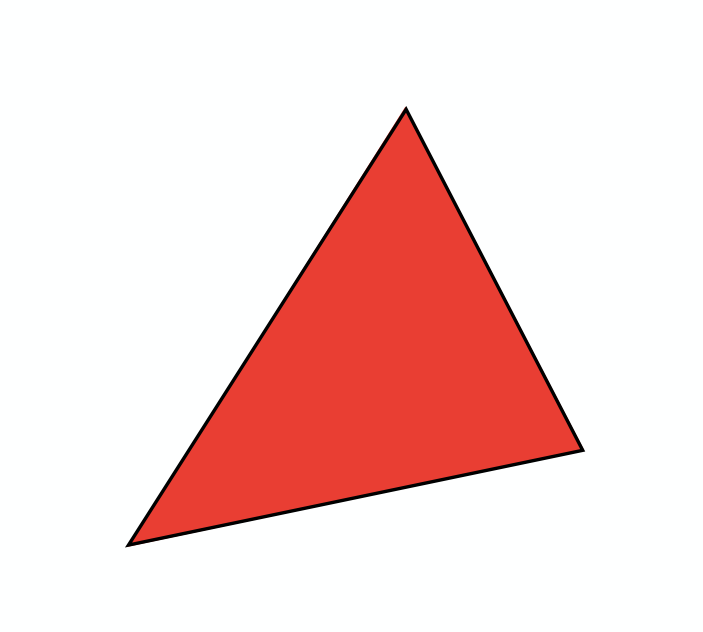
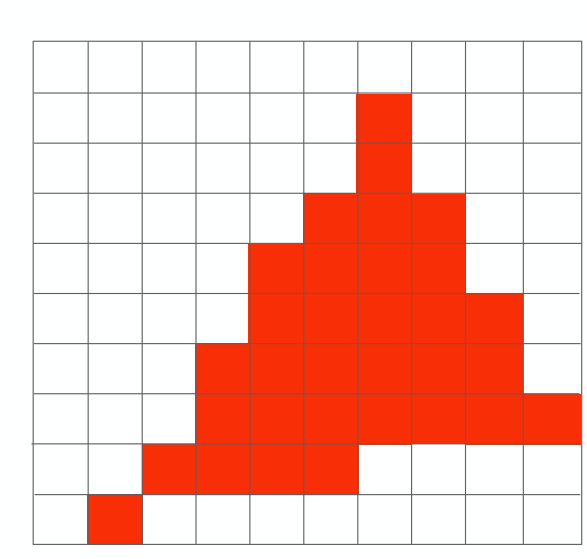
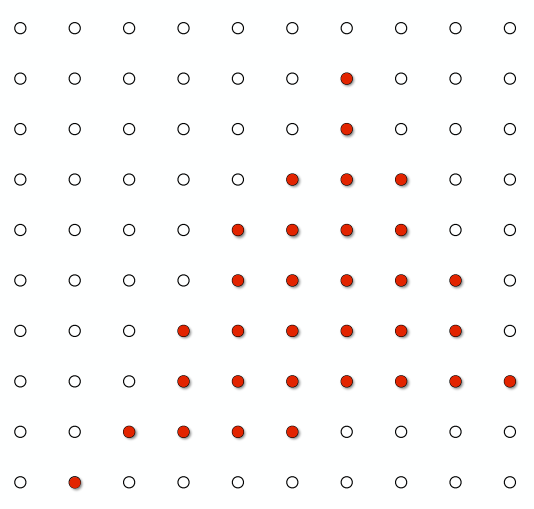
# （一）、Aliasing and Antialiasing(锯齿(走样)和抗锯齿(反走样)）

在上一讲我们做完采样工作以后，得到了如下图左边的图片，将所有红点所在的像素填充颜色后，我们得到了中间的图片，但是与最右边我们想要得到的效果相对比，还是差了很远，出现了严重的锯齿（走样）问题。

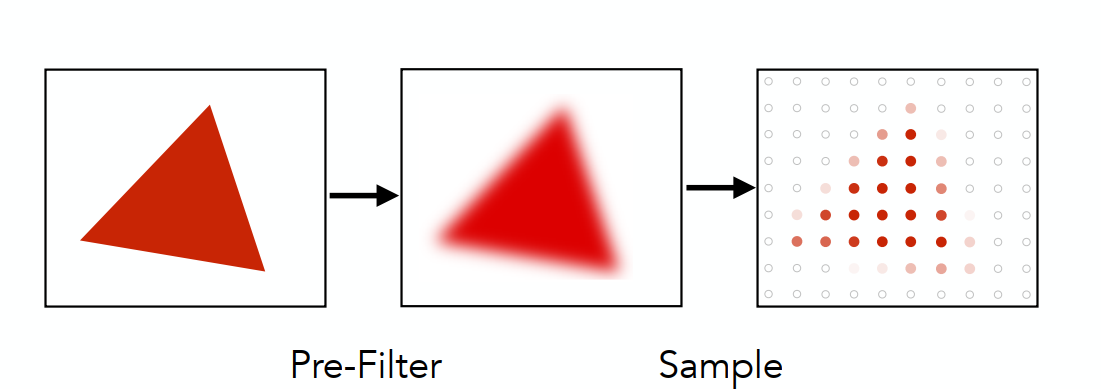


要解决上述问题，就需要用到反走样技术，在这里先给出操作的方法：先模糊后采样

Blurring (Pre-Filtering) Before Sampling

Why undersampling introduces aliasing?

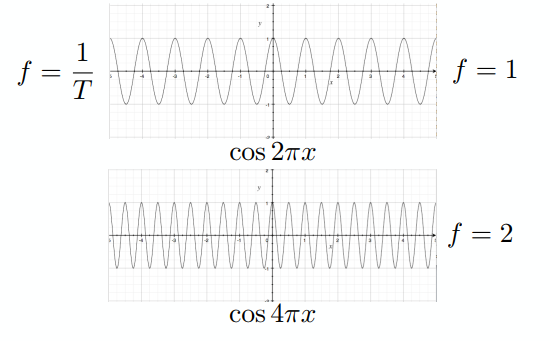
Why pre-filtering(模糊) then sampling can do antialiasing?



下面我们来分析原理

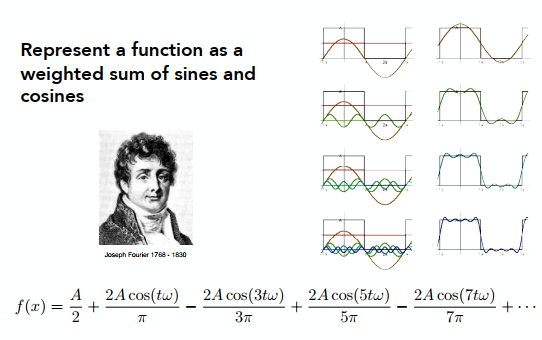
# （二）、Frequency Domain

## 1、Fourier Transform



通过这张图我们可以看到，影响这个余弦波变化快慢的因素是f，即为频率

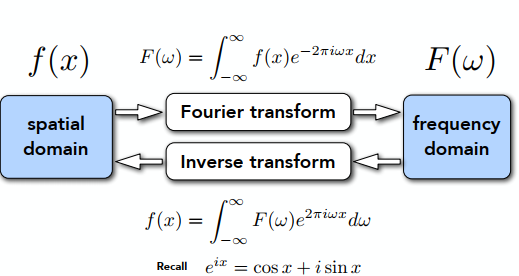
傅里叶级数展开：任何一个周期函数都可以写成一系列正弦和余弦函数的线性组合以及一个常数项



上图描述了阶跃函数被一系列正余弦函数和一个常数项表示的过程。

从f（x）=A/2（最低的频率）开始，每次不断地向式子中加入一些正余弦函数（函数的频率不断增高），然后画出新的图像，会发现当加入的项越多时，整个f（x）函数的图像越接近于阶跃函数。

### Fourier Transform Decomposes A Signal Into Frequencies

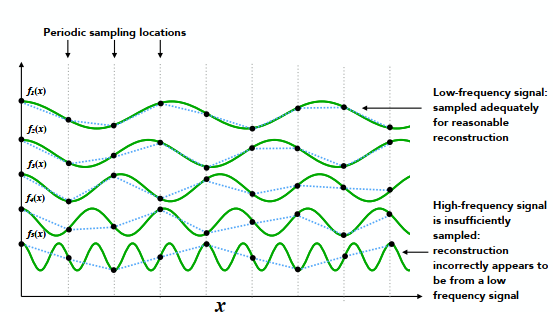


傅里叶变换：给定任何一个函数，都可以通过一系列复杂的函数变换变成另一个函数，变换后的函数还可以通过一系列操作变回原函数。这个操作就叫傅里叶变换和逆傅里叶变换。

## 2、Aliases

不同的正弦和余弦函数有不同的频率，通过傅里叶级数展开，我们可以知道任何一个函数都可以分解成不同的频率。我们可以根据把任何一个f（x），按照频率从低到高分解。（从刚刚傅里叶级数展开的图中，我们也可以看出分解开的函数频率是不断增高的）

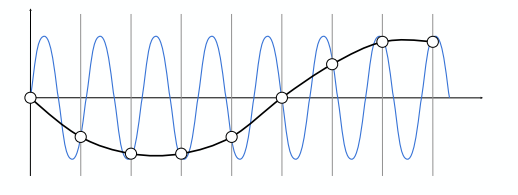
### Higher Frequencies Need Faster Sampling



我们把一个函数分解成不同频率的函数，从上到下频率由低到高，我们用相同的采样方法（间隔相同的时间段）对这些函数进行采样。将采样后的点连起来。我们会发现，低频的函数采样后通过采样点相连依然可以大致还原出原来的函数样子，而当频率逐渐变高时，采样点相连就慢慢无法看出原来函数的样子。

通过这个例子我们可以看出，当函数频率很低的时候，我们用低频的采样就可以大致得出原来的函数，而当函数频率变高时，如果再用低频的采样，那么采样频率跟不上函数的变化，就无法恢复出原始的函数。因此我们必须用相对高的采样频率，才能还原出原来的函数。

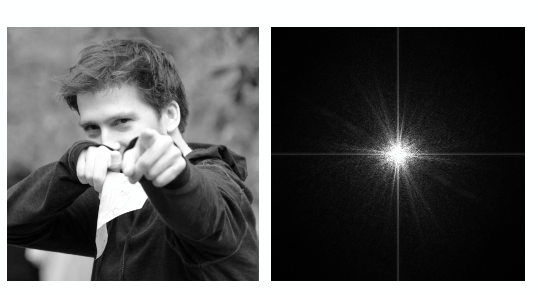
### Undersampling Creates Frequency Aliases



高频信号(蓝线)由于采样频率不足，得到的样本信号(空心采样点)错误地看起来像来自低频信号(黑线)，因而在给定采样频率下无法区分两个原始信号(不能知道空心采样点到底是来自于黑色还是蓝色线。)这称为“走样”

## 3、Filtering（Filtering = Getting rid of certain frequency contents）(滤波)

Visualizing Image Frequency Content



左侧的图片，通过离散傅里叶变换，得到了右边这张图(频谱)。那么这张图如何去分析？

我们定义，中间为低频区域，四周为高频区域（这里的低频和高频指的是灰度图中相邻像素（假设黑是0白是1）的差，差越大频率越高）

右图中的每一个点：

1）它到中点的距离描述的是平面波频率

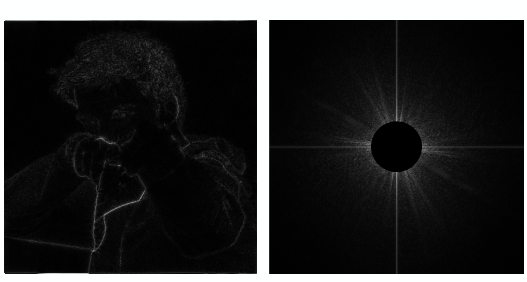
2）中点到它的方向，是平面波的方向

3）那一点的灰度值描述的是平面波的幅值

（补充：为什么会出现横竖两条线？因为相当于水平竖直各平铺了无数张图片，这样左右边界和上下边界相交的时候因为像素之间灰度差别过大会产生一个极大的频率）

知乎上一篇文章对于[图像傅里叶变换](https://zhuanlan.zhihu.com/p/99605178)作了清晰的回答： 建议参考

### Filter Out Low Frequencies Only (Edges)(高通滤波器)



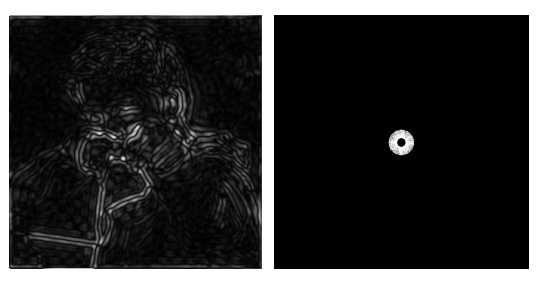
我们如果去掉刚刚那副图的低频区域，那么高频区域就是图像的边缘（可以想想原因：我们平时在判断物体的边缘时，就是看颜色的突变）

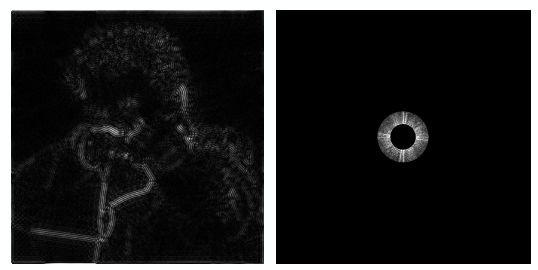
### Filter Out High Frequencies (Blur)(低通滤波器)



而如果去掉刚刚那副图的高频区域 ，那么留下的低频区域就是模糊的图像。(去掉了边界导致图像不清晰)

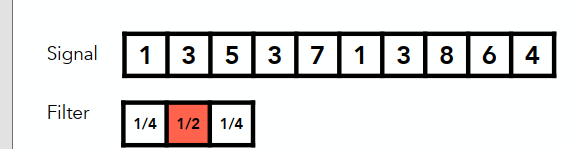
### Filter Out Low and High Frequencies

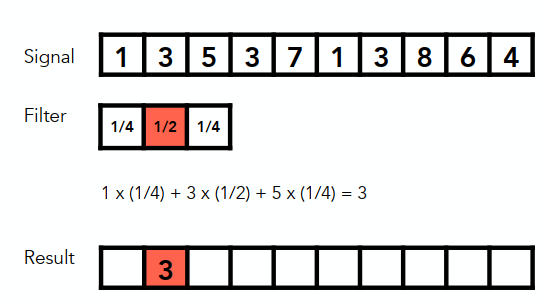


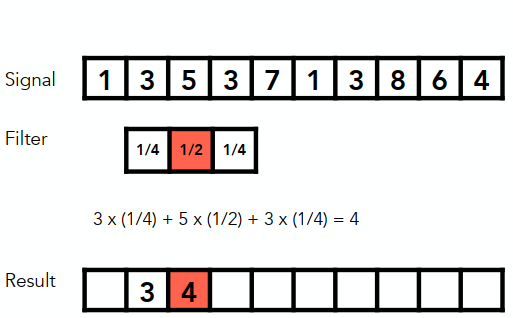


而如果去掉高频和低频，留下中频区域，则得到的图像如上所示。对应一些不太明显的高频特征

## 4、Convolution（Filtering = Convolution (= Averaging)）







通过这个例子来理解卷积的概念：假设有一个滤波器Filter，每次用他中间的位置对应要处理的像素，周围的方块也会分别对应一个像素，用滤波器（卷积核）和对应的像素作点乘（用相应的比例去乘以对应的像素值），得到的结果写回中间位置，即为卷积。

### Convolution Theorem(卷积定理)

空间域的卷积等于频域的乘法，反之亦然

方法1：

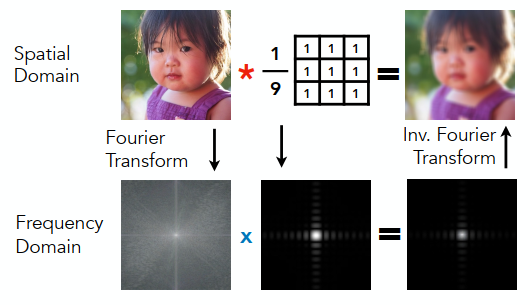
图像在空间域中与卷积核计算卷积

方法2：

图像变换到频域频谱（傅里叶变换）

卷积核变换到频域频谱(傅里叶变换)

二者相乘频谱经傅里叶逆变换回原始空间

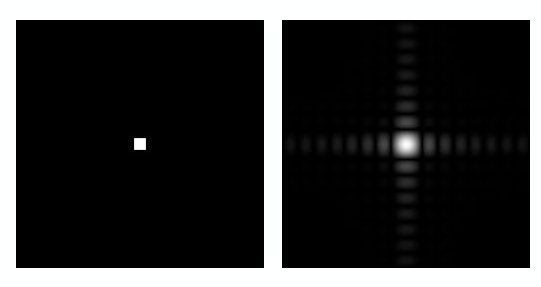


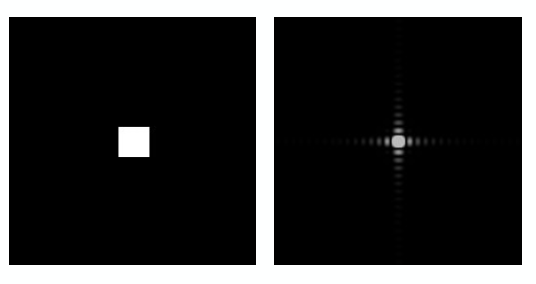
应用方法一:在这里这个例子中我们可以看到，上左与上中一个3×3的卷积核进行卷积，得到一个模糊的图像上右。

应用方法2:图像和卷积核都可以通过傅里叶变换在频域上表示出来(下左下中)，二者相乘。最后得到的频谱(下右)再做傅里叶逆变换成上右。

### Wider Filter Kernel = Lower Frequencies

如果用一个更大的卷积盒，得到的频域中，图像中间部分将会变小。

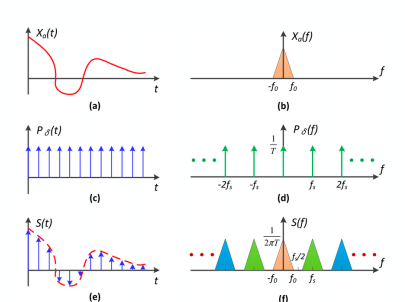




理由如下：用一个小的卷积盒（假设3×3），模糊的程度可能不是很大，但是要用一个很大的卷积盒（假设63×63）去对这个图像进行卷积操作，得到的图像肯定会越来越模糊，那么也就只能留下更低的频率。

### 5、Sampling = Repeating Frequency Contents

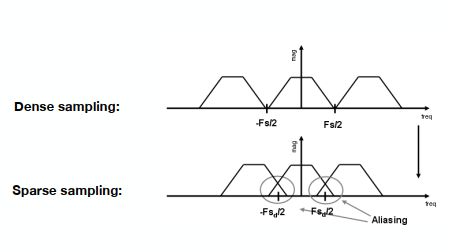
采样就是在重复频率上的内容



（a）是某个连续的函数，假设他通过傅里叶变换反应在某个频域上是（b），假如要采样这个函数，就要在这个函数上乘以另外一个只在固定位置上有值的函数（冲激函数，如（c）），用（a）乘以（c），得到的就是（a）函数上的一系列连续的点（e）。（b）和（d）分别是（a）和（c）傅氏变换之后的结果

由于时域上的乘积等于频域上的卷积，所以（f）就是（b）卷积（d）的结果。我们可以发现（f）不过是（b）复制粘贴的重复过程。

### 6、Aliasing = Mixed Frequency Contents



如果采样率合适，如图上方的图，则没有什么问题，如果采样率不足（采样不够快），那么信号“复制粘贴”中间的间隔就会非常小（采样间隔时间大，换算成频率的话就是频率间隔小（两者是倒数关系）），所以原始信号和“复制粘贴”的信号就会出现叠加，这种情况就叫发生了走样。所谓走样在频率的角度上来讲就是频率的频谱在经过“复制粘贴”的情况下发生了混合（混叠）。

# （三）、Antialiasing

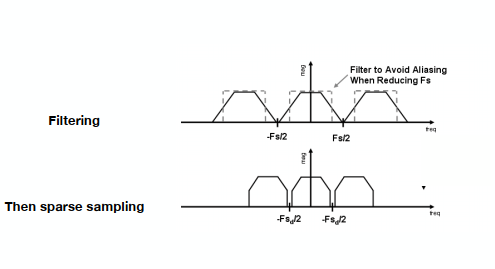
## How Can We Reduce Aliasing Error?

Option 1: Increase sampling rate（增加采样率）

本质上增加了频域中副本之间的距离，但是需要更高分辨率的显示器、传感器、帧缓冲器

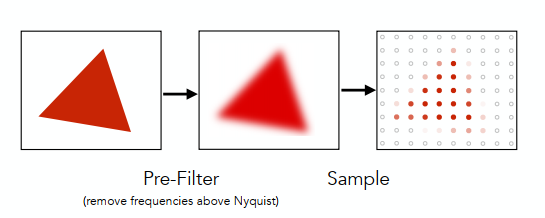
Option 2: Antialiasing

先模糊，后采样。先通过一个低通滤波，把信号的高频的信息拿掉，然后再采样。如图，我们就会发现，信号对应的频谱就不会发生混叠了。



## Antialiased Sampling

回到这节课一开始的问题，我们如何将这个三角形变模糊，答案也就很明确了。



用一个合适的低通滤波器。

## Antialiasing by Computing Average Pixel Value

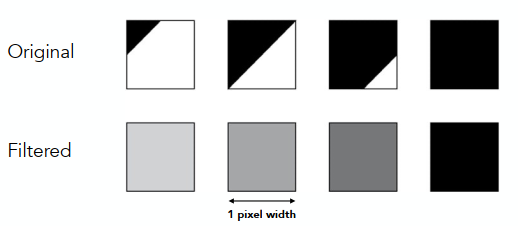
那么这个低通滤波器应当怎样选取呢？一般选择一个像素大小(1\*1)的卷积核

Solution:

Convolve f(x,y) by a 1-pixel box-blur(对每个像素做1\*1卷积)

​ --Recall: convolving = filtering = averaging

Then sample at every pixel’s center(对每个像素中心做采样)

在光栅化一个三角形时，f(x,y) = inside(triangle,x,y) 即像素区域的平均值等于三角形覆盖的像素区域

如图，假设三角形覆盖了一个像素的最左上边(覆盖了1/8)，有7/8没覆盖，也就是有87.5%的白色,12.5%的黑色，它的卷积核的值就取0.125(也就是12.5%的黑色)。这样这个像素的灰度经过卷积运算就变成了0.125。又假设最右边的这个像素被三角形全部覆盖，也就是有0%的白色（纯黑色）。那么它的卷积核就取1。这个像素的灰度值经卷积运算就是1。这样就会把每个被三角形影响的像素灰度值重新分配，以达到三角形区域模糊的目的。

# （四）、Antialiasing By Supersampling (MSAA)

所以应该怎样去判断三角形覆盖了某个像素的面积呢？

对于任何一个像素里面，我再把这个像素划分成好多小的像素，每个小的像素也有各自的中心，再判断这些小中心是否在三角形覆盖的区域内，根据每个大像素中小中心在与不在的三角形区域内的比例近似估算三角形覆盖像素的面积。

## Supersampling: (超采样)

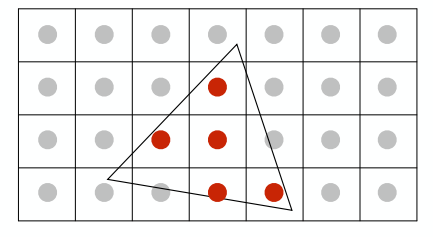
Step 1

Take NxN samples in each pixel.

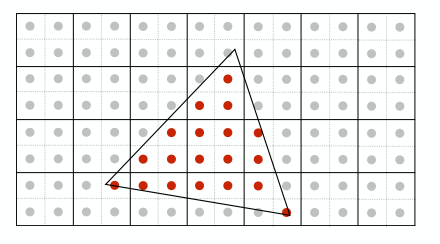
Step 2

Average the NxN samples “inside” each pixel.

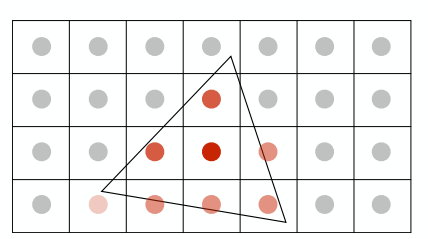
例子：以下是一个三角形覆盖的像素区域，可见在一个4\*7屏幕里，三角形内部有5个像素



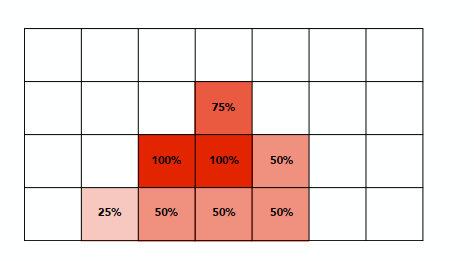
从左到右从上到下遍历，先发现影响的是(2,4)像素，对这个像素进行细分，变成了四个小像素，可见覆盖了三个小像素中心。可以近似这个三角形覆盖了(2,4)像素0.75的面积。同理，覆盖了(3,3)像素0.75面积，(3,4)1.00面积，(4,4)0.5面积 (4,5)0.5面积



以刚才覆盖面积百分比作为卷积核，对这5个大像素的灰度重新定义。可以看到这5个像素的灰度值变了。(灰度值分别为0.75 0.75 1 0.5 0.5 0.5)



也就是，说MSAA不是通过直接提升屏幕分辨率（采样率）来解决走样问题，通过增加那么多采样点，只是为了近似一个合理的三角形的覆盖率，并没有真的提高屏幕分辨率。



## What’s the cost of MSAA?

为了引入MSAA，代价是增大了计算量，但是不是成指数倍增加的，在工业界中，人们并不是规则的划分网格，而是用一些更加有效的图案，甚至有一些点还会被临近的像素所复用。因此如果打游戏启用抗锯齿，假设用4倍，帧率并不会掉到原来的1/4，就是这个原因。

除了MSAA还有别的抗锯齿方法，推荐以下两种：

FXAA (Fast Approximate AA) （图像的后期处理，先得出有锯齿的图，再去用图像处理的方式，把边界找到，并且把这些边界换成没有锯齿的边界）

TAA (Temporal AA) （找上一帧的信息，复用上一帧的像素值）

Super resolution（和抗锯齿不太一样，但都是解决样本不足的问题。把一张小图拉大，但又不想看到锯齿或者说有一张高分辨率图，但是采样率不够，想把高分辨率的图恢复出来）

From low resolution to high resolution

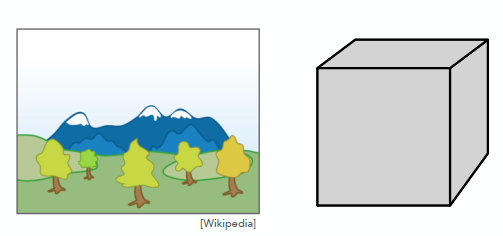
Essentially still “not enough samples” problem

DLSS (Deep Learning Super Sampling) （深度学习=猜！）

# （五）、Z-buffering(深度缓存||深度缓冲)

Painter’s Algorithm

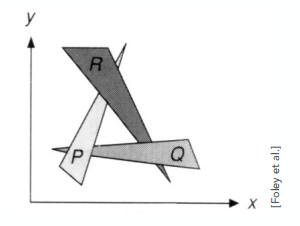
Inspired(启发) by how painters paint Paint from back to front, overwrite in the framebuffer



假设此处有一幅画，画家是从后往前画，先画最后面的山，然后画前面草地，遮挡住了一部分山，然后又画前面的树，遮挡住了一部分山和草地，一层一层遮挡。同理，当有一个3D场景，先把远处的东西印带屏幕上，做光栅化处理，然后再往近处慢慢把光栅化做完。

Requires sorting in depth (距离观察者的距离，称为深度信息)(O(n log n) for n triangles

Can have unresolvable depth order(画家算法难以解决以下问题)



但是会出现一个问题：一旦有三个三角形，互相覆盖互相部分遮挡（如上图R、P、Q），那么就无法判断谁在前谁在后（深度上存在互相遮挡关系）。因此我们引入了Z-buffer（深度缓存）的概念

## Z-buffer

Store current min. z-value for each sample (pixel)

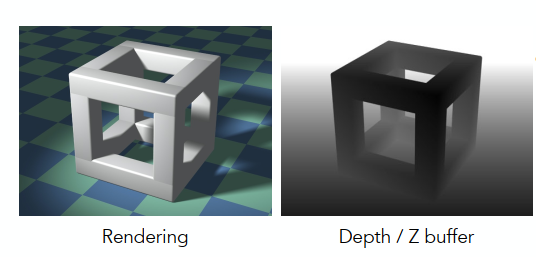
Needs an additional buffer for depth values

–frame buffer stores color values(最终视图存储颜色值)

–depth buffer (z-buffer) stores depth(深度视图存储深度值)

IMPORTANT: 由于我们是将照相机摆在原点，朝向-z。所以正常情况下z值越小应该是离原点越远，越大离原点越近。但是这里为了方便假设z值为正值，也就是z越大离得越远，越小离得越近。

对每一个像素来看，永远去记录像素表示的几何离我们的最浅（最靠前，最近）深度。



在渲染生成图像（上方左侧图）的同时，也会生成另一张图像，这张图只存任何一个像素所看到的几何物体最浅的深度信息，叫做深度缓存图（上方右侧图，离观察视点越近则越黑，越远则越白），利用这张图去维护遮挡关系的信息。

特别注意的是，这张Depth/Z-buffer图中，颜色的灰度代表点距离摄像机的远近，黑色代表近，白色代表远。

假设左边这幅图一开始渲染地板，右边的Depth/Z-buffer图中先将各个像素点的地板深度信息存储进去，后来在渲染左边的立方体的时候，一些像素点的地板被立方体遮挡，这时候右边更新深度信息，将那些遮挡地板的立方体所在的各个像素点的的深度信息储存进去。

Z-Buffer Algorithm算法

Initialize depth buffer to ∞ During rasterization:

for (each triangle T)

for (each sample (x,y,z) in T)

if (z < zbuffer[x,y]) // 比之前存储的像素深度更近

{

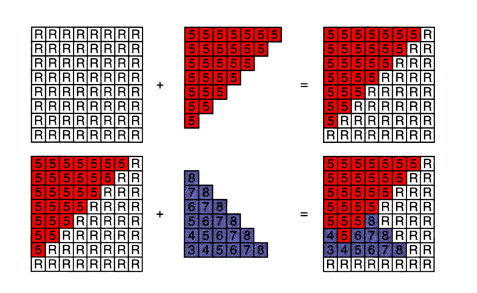
framebuffer[x,y] = rgb; // update color

zbuffer[x,y] = z; // update depth

}

else

; // 新三角形的像素深度比之前的要更远，不更新



如图，一开始认为所有像素的深度无限大，后来加上了红色三角形，其深度都为5，这些像素深度原来是R（无限大inf），因此新加进来的像素深度5比一开始的小，于是将其替换。后来加上了蓝色三角形，将这些区域的像素深度与刚刚同位置的像素深度比较，小的替换掉大的。

## Complexity(复杂度)

O(n) for n triangles (assuming constant coverage)

How is it possible to sort n triangles in linear time?

因为我们要知道，我们并没有提前排序，对于任意一个像素只是记录当前看到的最小值。

Most important visibility algorithm

Implemented in hardware for all GPUs