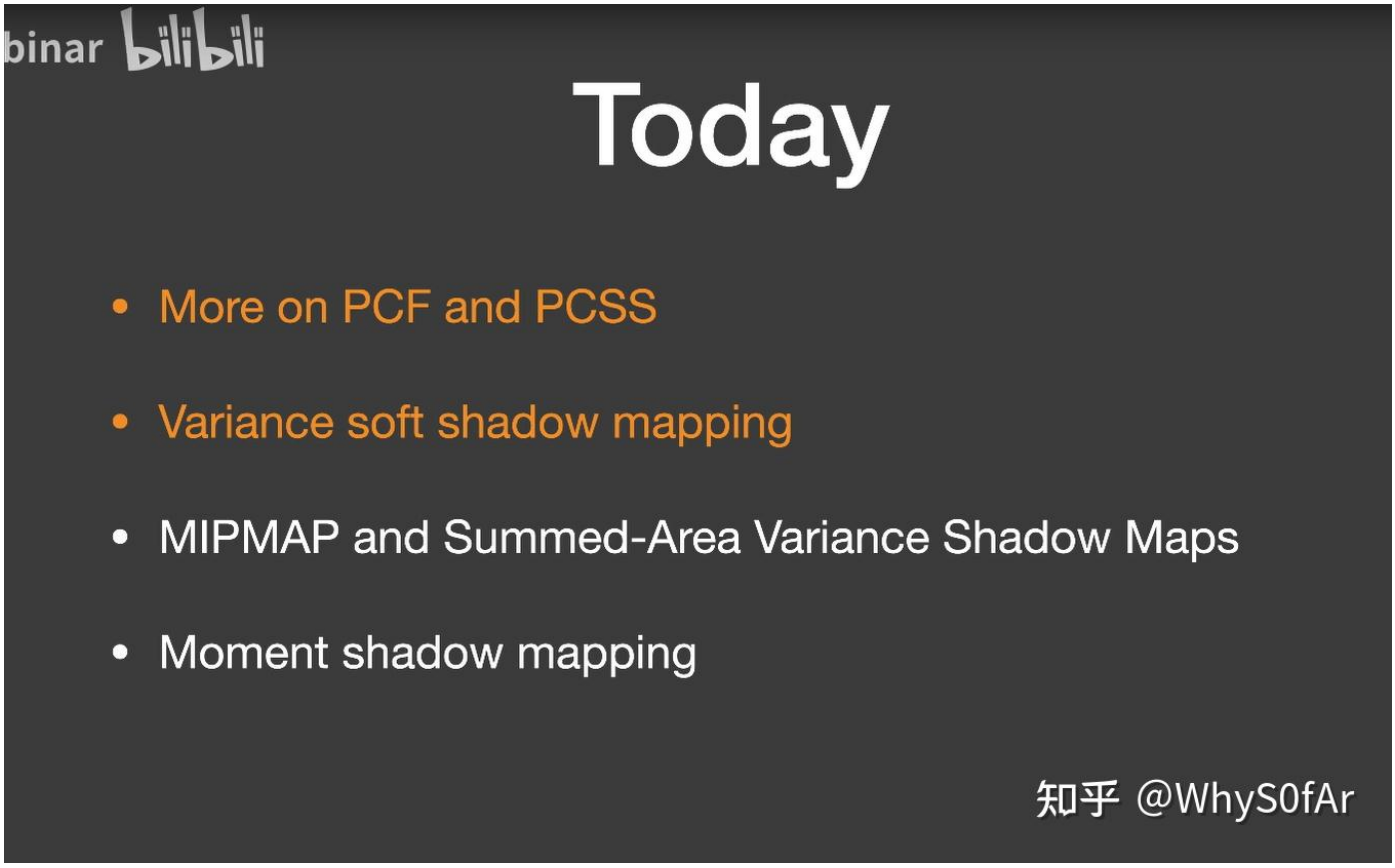
GAMES202 Real-Time High Quality Rendrting 高质量实时渲染课程笔记Lecture 4- Shadow 02

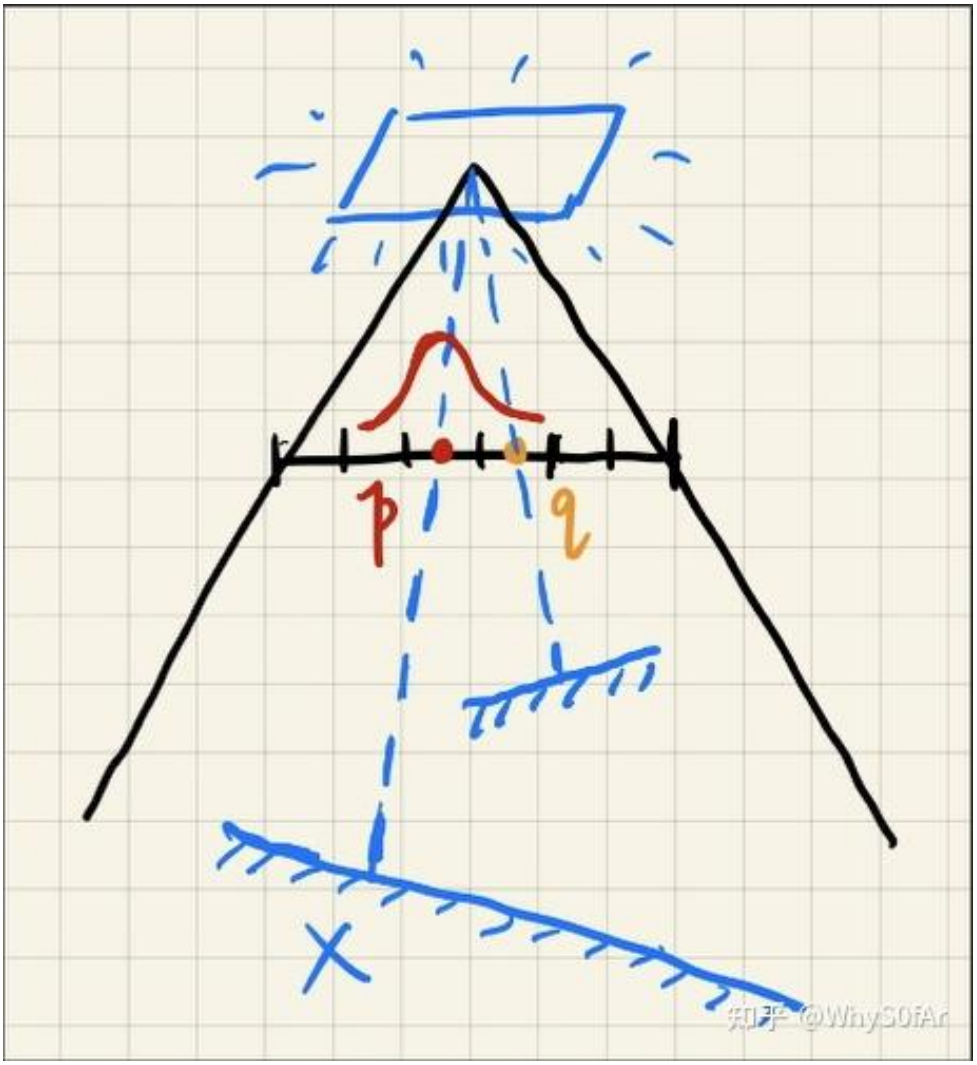
本文是闫令琪教授所教授的Real-Time High Quality Rendrting的第四讲阴影第二部分笔记，本人是计算机图形学菜鸟，如果有错误之处麻烦各位在评论区指出,谢谢大佬们.

GAMES202-高质量实时渲染_哔哩哔哩 (゜-゜)つロ 干杯~-bilibili



本节主要内容是关于pcf和pcss的深度了解以及vssm.

A Deeper Look at PCF



我们先来回顾一下PCF做了什么事：

对于实际渲染的点X,我们找到其投影到shadow map上对应的像素点---p,我们不只考虑点p,还要考虑其周围一圈范围内的像素,区域内各像素的根据到点p的距离进行加权平均,离得远贡献小,离得近贡献大.

区域内除点p的其他像素在shadow map上记录的最小深度,也与x的实际深度进行比较,从而判断区域内有多少fragments是遮挡物(Blocker),然后算出平均的visilibity,从而得到了一个在0到1之间的软阴影效果.

- Filter / convolution:

$$[w * f](p) = \sum_{q \in \mathcal{N}(p)} w(p,q) f(q)$$

知乎 @WhyS0fAr

这个公式的意义就是:

对于任意一点p,我们取其周围一圈作为邻域,也就是图中的N(p),在邻域中取一点q,我们考虑邻域中的所有点q,并将各点q的值,根据点q与点p间的距离做一个加权处理,最后把加权所有点q后的值写入点p的值,也就是做了一个filter.(也就是games101中的卷积).

PCF的公式:

$$V(x) = \sum_{q \in \mathcal{N}(p)} w(p,q) \cdot \chi^{+}[D_{\mathrm{SM}}(q) - D_{\mathrm{scene}}(x)]$$

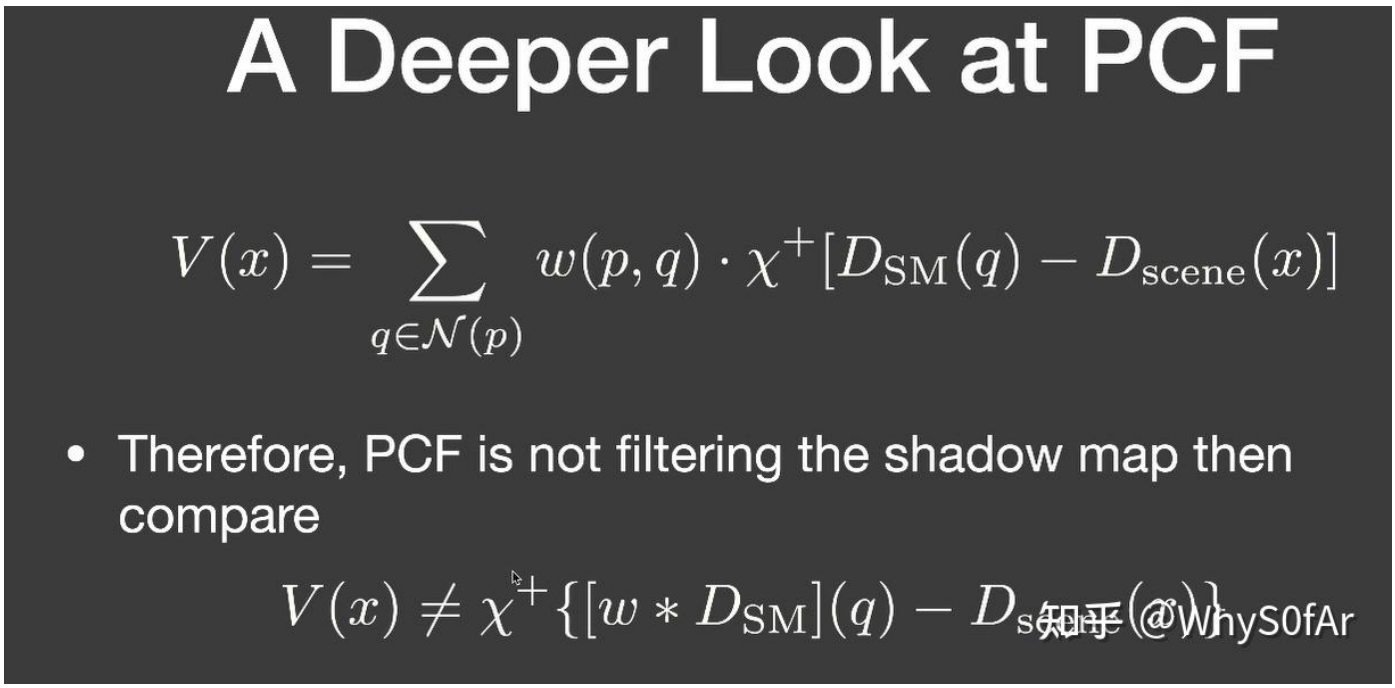
$$[D_{\mathrm{SM}}(q) - D_{\mathrm{scene}}(x)]$$

如果图中的公式值>0,那么

$$\chi^+[D_{\mathrm{SM}}(q) - D_{\mathrm{scene}}(x)]$$

的值为1,反之小于0,值为0.

将每一个点q与x的实际深度比较后,知道了各自的visilibity是0还是1,将所有点q对应的visilibity进行一个平均,就得到了点x在经过pcF后的效果.



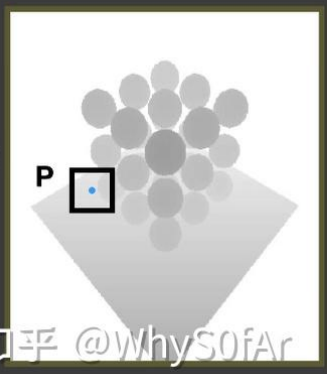
从图中公式可知,pcf并非在filter Shadow map,图中下方为filter shadow map,从公式中可知,最后我们得到的值仍然是非0即1的,得到的仍然是硬阴影.

以上就是关于pcf的数学公式方面的知识.

那么在PCSS里我们做了什么事呢?

Revisiting PCSS

- The complete algorithm of PCSS
  - Step 1: Blocker search
    (getting the average blocker depth in a certain region)
  - Step 2: Penumbra estimation
    (use the average blocker depth to determine filter size)
  - Step 3: Percentage Closer Filtering
- Which step(s) can be slow?
  - Looking at every texel inside a region (steps 1 and 3)
  - Softer -> larger filtering region -> slower



首先将shading point点x投应到shadow map上,找到其对应的像素点p.

a) 在点p附近取一个范围(这个范围是自己定义或动态计算的),将范围内各像素的最小深度与x的实际深度比较,从而判断哪些像素是遮挡物，把所有遮挡物的深度记下来取个平均值作为blocker distance。（Blocker search）

第二步：用取得的遮挡物深度距离来算在PCF中filtering的范围。

第三步：进行pcf操作.

步骤1和步骤3需要对整个区域的各个texels与点x的深度进行比较，所以会导致很慢。

如果觉得区域过大不想对每一个texels都进行比较,就可以通过随机采样其中的texels，而不是全部采样，会得到一个近似的结果,近似的结果就可能会导致出现噪声。

工业的处理的方式就是先稀疏采样得到一个有噪声的visibility的图,接着再在图像空间进行降噪。至于如何降噪在real-time ray tracing讲.

由于需要在一个范围内进行比较，那么步骤1和3的时间开销会决定整个算法的时间开销，此外为了得到越“软”的阴影意味着需要使用更大的filtering size，会导致速度越慢。为了解决这两步慢的问题,就有人提出了Variance Soft Shadow Mapping。

Variance soft shadow mapping

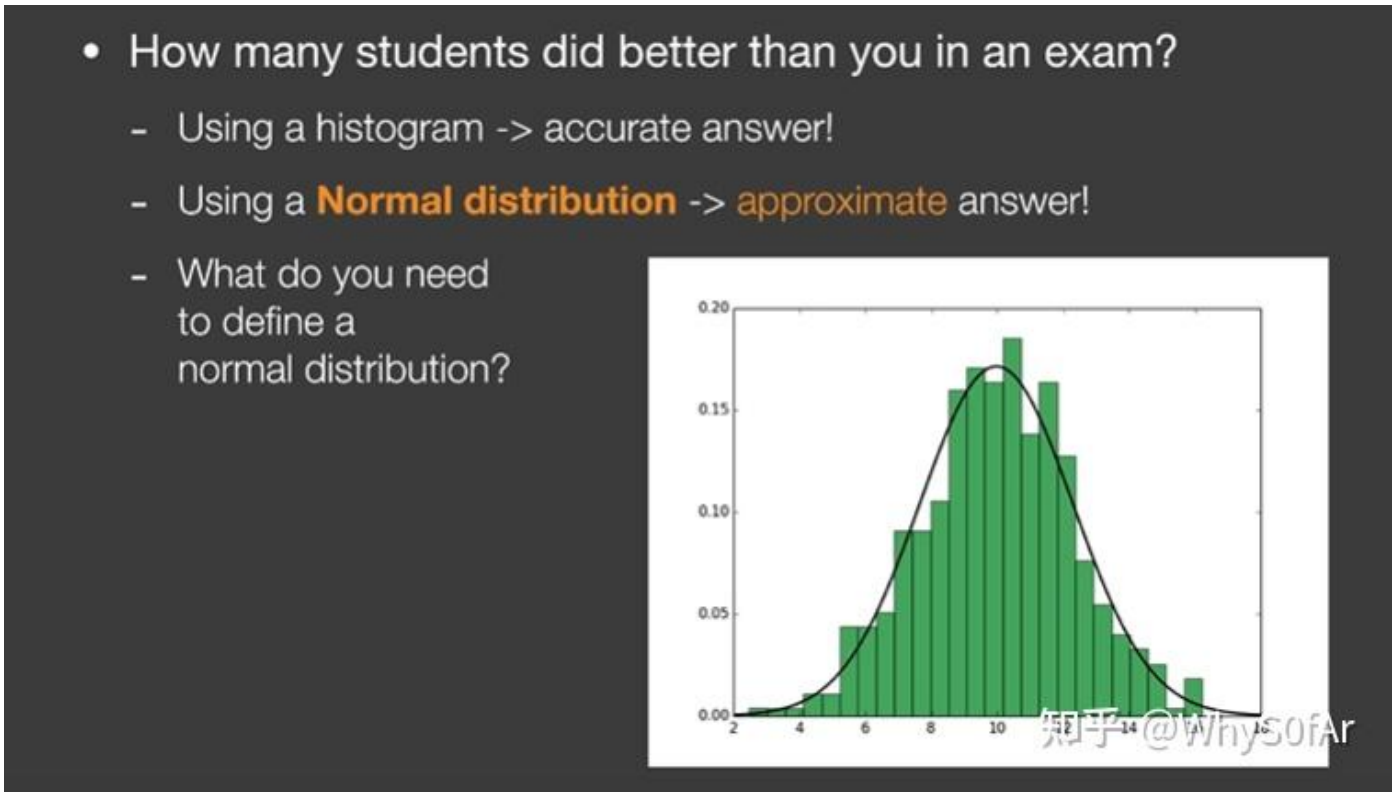
Variance soft shadow mapping主要解决了PCSS中第一步和第三步慢的问题.

我们回到第三步PCF之中,我们要在shadow map上对其周围的一圈像素的各个最小深度与Shading point比较,从而判断是否遮挡,也就是要求出范围内有百分之多少的像素比它浅.

这个过程很像在考试成绩出来后,你知道了自己的成绩,你想知道自己在班级中的排名,因此你需要知道班级中所有人的成绩从而进行比较来判断自己是百分之几,这就是PCF的做法.

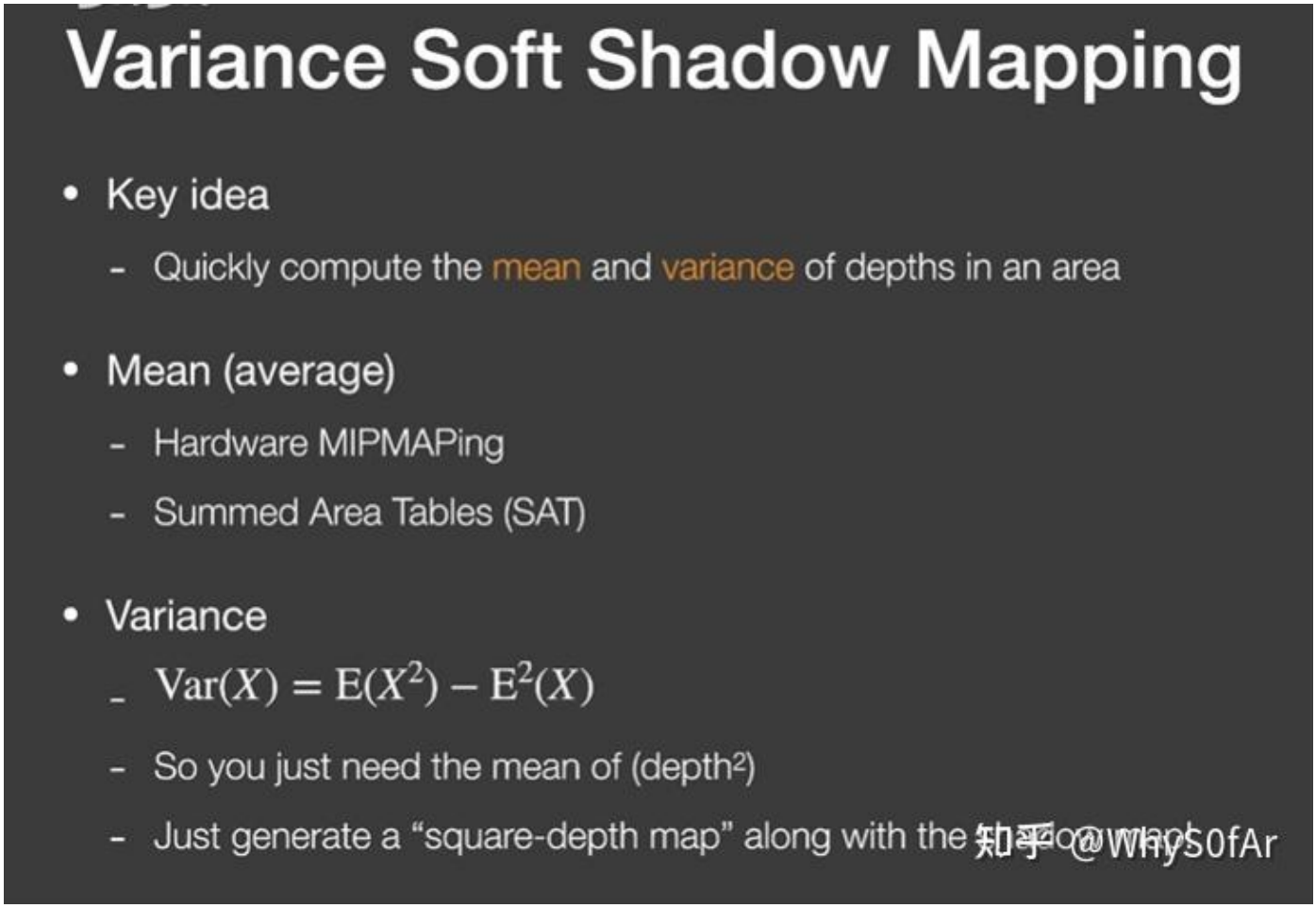
但现在我们就是为了避免这种时间消耗大的做法.

那么一个不错的办法就是,对班级所有人的成绩做成一个直方图,根据直方图我们可判断出自己的成绩排名.



如果我们不需要那么准的话就可以当做一个正态分布，正态分布就只需要方差和平均值就能得出,更加的方便快速,这也就是VSSM的核心思想,通过**正态分布** 来知道自己大约占百分之几.

VSSM的**key idea** 是快速计算出某一区域内的均值和方差.



均值:

对于快速的求一个范围内的求均值,我们可以想到在games101中学到的mipmap方法.但是**mipmap** 毕竟是**不准** 的,而且只能在正方形区域内查询.因此引入Summed Area Tables (SAT）.

方差:

VSSM用结合了**期望** 与**方差** 之间的关系的一个公式来得到方差:

$$Var(X) = E(X^2) - E^2(X) \tag{1}$$

这个公式的含义是用 **平方值的期望 - 期望值的平方** = 方差.

用这个公式的原因是:

在shadow map中我们存储的是depth,因此depth也就是公式中的x,在指定区域范围后,可以快速的求出区域范围的平均值(期望),因此也可以很快求出区域范围内平均值的平方,也就是求出了 $E^2(X)$.

那么求 $E(X^2)$,我们就需要额外生成一张shadow map,但是这张图上存的不是depth,而是 $depth^2$,然后再在指定范围区域内快速求出平均值,也就是求出了平方值的期望,求出了 $E(X^2)$,这张存储了 $depth^2$的shadow map叫做square-depth map.

也就是为了求方差,需要在生成shadow map时再存储一张square-depth map。

到此为止我们就快速获得了均值和方差。那么回到问题本身:

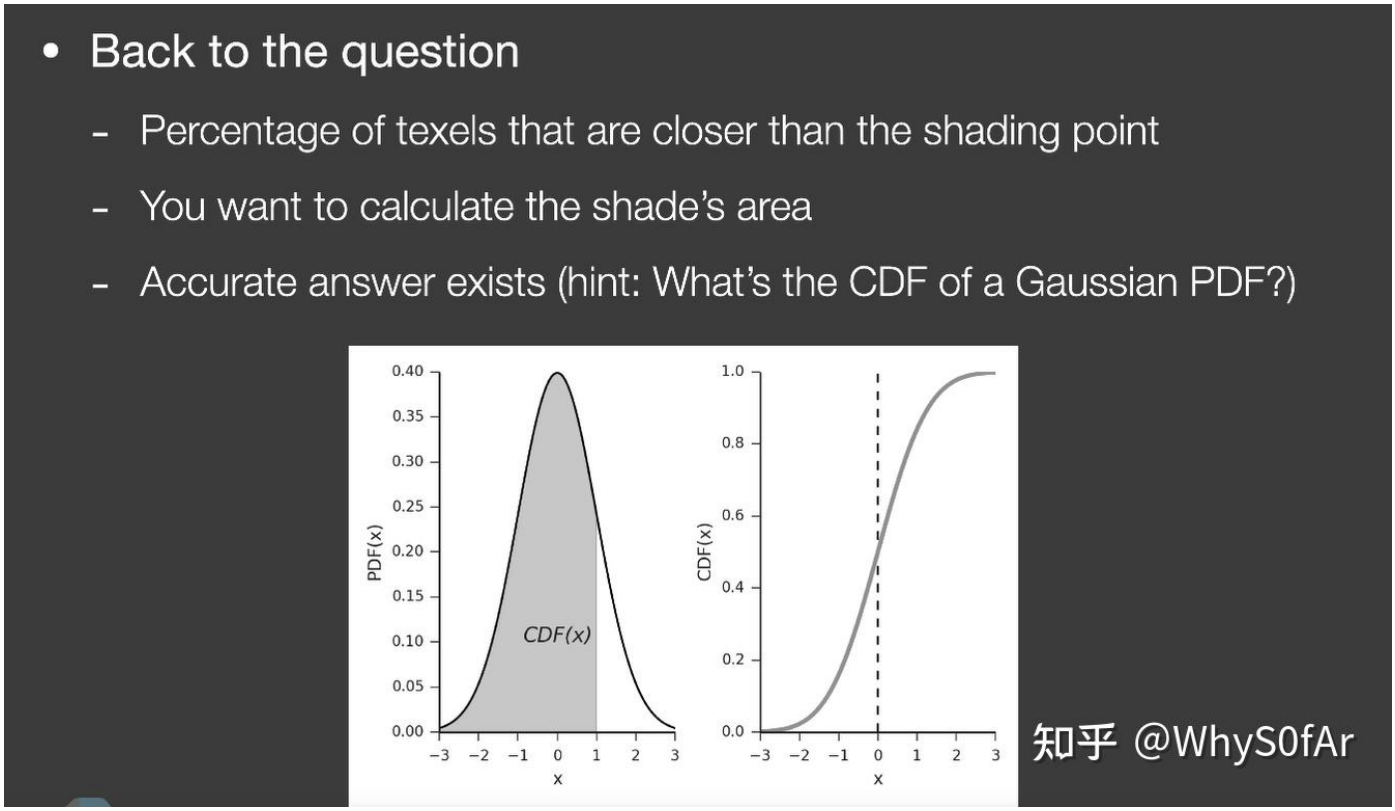
有多少百分比的像素是比Shading point **大** 也就是 **不会挡住** Shading point的只需要计算出下图 **PDF中白色面积** 的值就行了。

有多少百分比的像素是比Shading point **小** 也就是 **会挡住** Shading point的只需要计算出下图 **PDF中灰色面积** 的值就行了。

注:

PDF：概率密度函数（probability density function）, 在数学中，**连续型随机变量** 的概率密度函数是一个描述这个随机变量的输出值，在某个确定的取值点附近的可能性的函数。

CDF : 累积分布函数 (cumulative distribution function)，又叫分布函数，是**概率密度函数的积分**，能完整描述一个实随机变量X的概率分布。



其实想知道CDF，也就是求出PDF曲线下对应的面积.

对于一个通用的高斯的PDF，对于这类PDF，可以直接把CDF结果，输出为一个表，叫误差函数Error Fuction，误差函数有数值解，但是没有解析解，在C++ 中的函数ERF(lower_limit, [upper_limit])函数可以计算CDF。

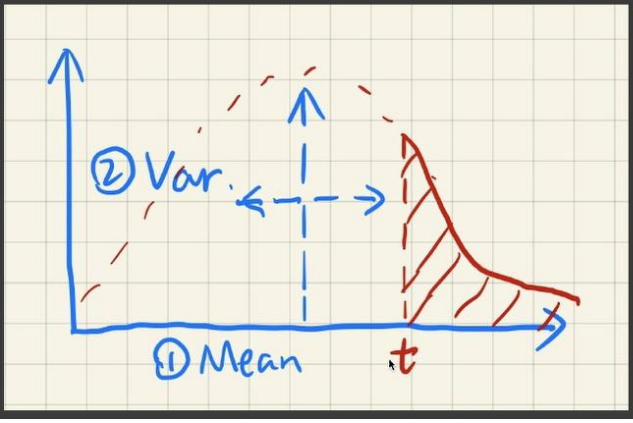
- It doesn't have to be too accurate!
  - Chebychev's inequality (one-tailed version, for t > mu)

$$P(x > t) \leq \frac{\sigma^2}{\sigma^2 + (t - \mu)^2}$$

μ : mean

σ^2 : variance

Doesn't even assume Gaussian distribution!



知乎 @WhyS0fAr

我们用切比雪夫不等式,来近似地求出在知道 **期望** 和**方差** 时候,不考虑是不是正态分布,图中红色面积不会超过等式右边,这里使用了一个Trick的方法，将这个**不等式近似为相等.**

那么就可以通过均值和方差获得图中红色面积的值。因此就可以不用计算CDF了，而是通过1 - 求出的x>t的面积值得到CDF.
但是**切比雪夫不等式有一个苛刻的条件：t 必须在均值右边，也就是t大于均值。**

加速第三步总结:

1.我们通过生成shadow map和square-depth map得到期望值的平方和平方值的期望再根据公式 得到方差

2.通过mipmap或者SAT得到期望

3.得到期望和方差之后,根据切比雪夫不等式近似得到一个depth大于shading point点深度的面积.,也就是求出了未遮挡Shading point的概率,从而可以求出一个在1-0之间的visilibity.

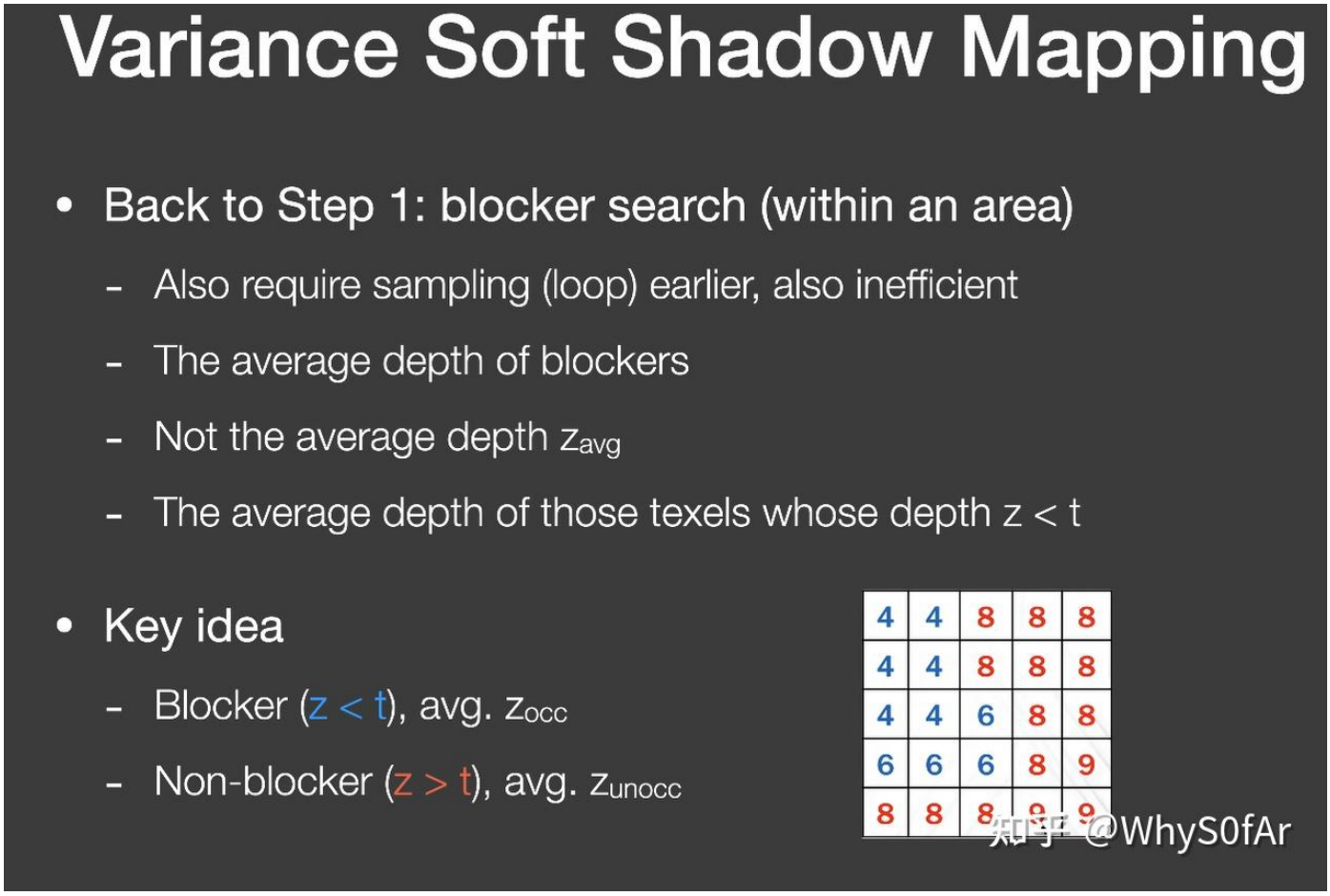
也就是省去了在这个范围内进行**采样** 或者**循环** 的操作,大大加速了第三步.

如果场景/光源出现移动 就需要更新MIPMAP，本身还是有一定的开销，但是生成MIPMAP硬件GPU支持的非常到位，生成非常快（几乎不花时间），而是SAT会慢一点，这个后面进行分析。

到目前为止VSSM也只解决了第三步PCF Filter的问题，PCSS在第一步要需要求在范围内将所有像素的深度走一遍从而求平均遮挡深度Average Blocker Depth的问题并未解决。

4	4	8	8	8
4	4	8	8	8
4	4	6	8	8
6	6	6	8	9
8	8	8	9	9

我们以图中的5*5范围为例,假设我们的Shading point的深度是7.



我们将其分为两个区域,**蓝色** 是深度小于shading point的遮挡区域,其平均深度为Zocc**红色** 是深度大于shading point的非遮挡区域.其平均深度为Zunocc.并且我们认为区域内的像素总数为N,非遮挡的像素为N1个,遮挡的像素为N2个.

- Key idea
  - Blocker (z < t), avg. z_{occ} (we want to compute)
  - Non-blocker (z > t), avg. z_{unocc}
  - $$\frac{N_1}{N} z_{unocc} + \frac{N_2}{N} z_{occ} = z_{Avg}$$
  - Approximation: N1 / N = P(x > t), Chebychev!
  - Approximation: N2 / N = 1 - P(x > t)
  - z_{unocc}, we really don't know
  - Approximation: z_{unocc} = t (i.e. shadow receiver is a plane)

4	4	8	8	8
4	4	8	8	8
4	4	6	8	8
6	6	6	8	9
8	8	8	9	9

核心思路:

我们需要去计算求出 Z_{unocc}和 Z_{occ},通过他们之间的关系我们可以得出一个数学公式:

$$\frac{N1}{N} * Z_{unocc} + \frac{N2}{N} * Z_{occ} = Z_{Avg}$$

非遮挡像素占的比例 * 非遮挡物的平均深度 + 遮挡像素占的比例 * 遮挡物的平均深度 = 总区域内的平均深度.

总区域内的平均深度我们用mipmap或者SAT去求,然后用shadow map和square-depth map方差,最后根据切比雪夫不等式近似求出 $\frac{N1}{N}$和 $\frac{N2}{N}$。

$$\frac{N1}{N} = P(x > t)$$

$$\frac{N2}{N} = 1 - P(x > t)$$

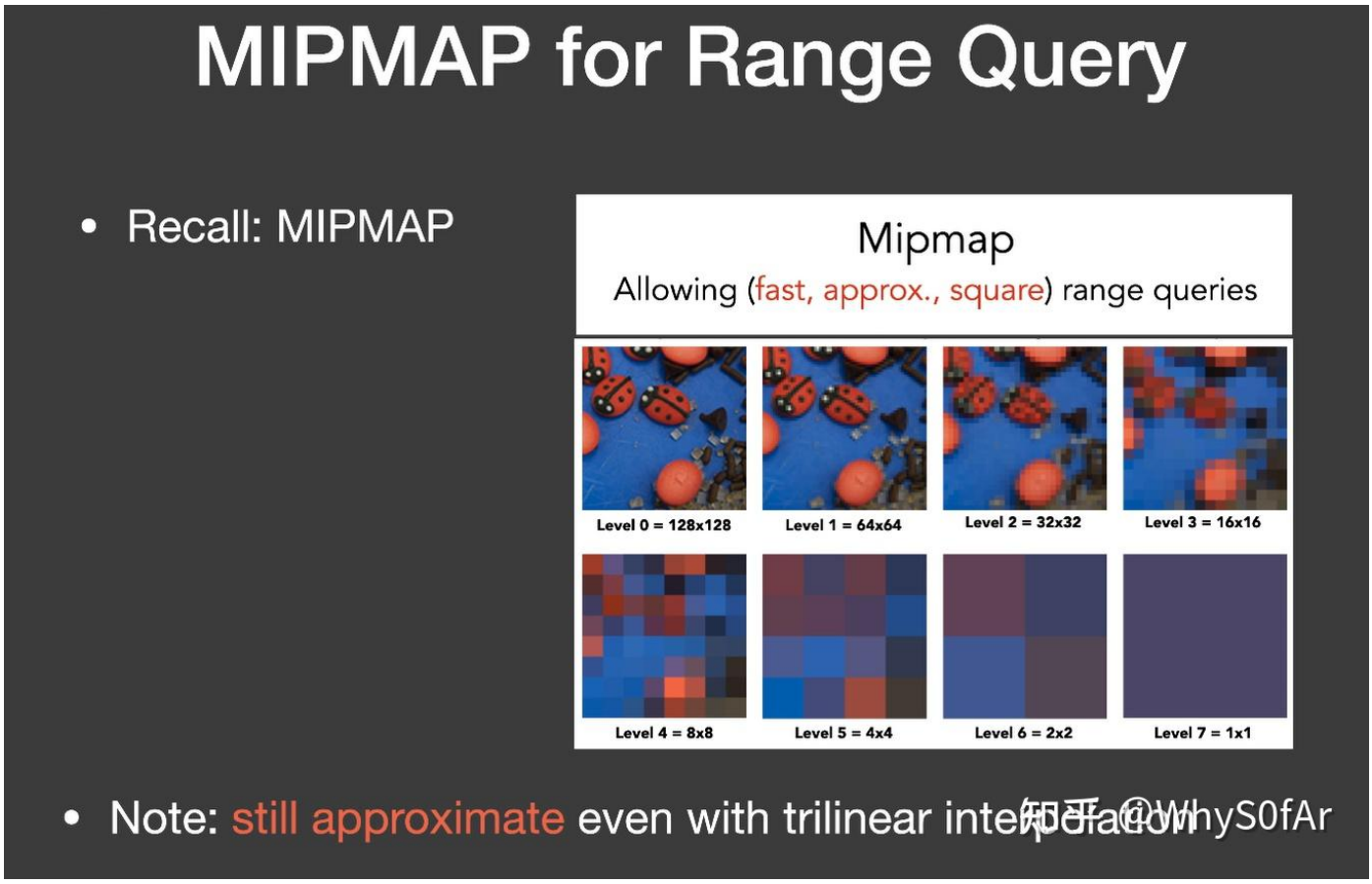
此时公式中的 Z_{unocc}和 Z_{occ}仍然是不知道的,我们做一个大胆的假设,我们认为非遮挡物的平均深度 = shading point的深度,至此我们只剩下Z_{occ}的深度,将所有值代入可求出遮挡物的平均深度,Z_{occ}.但是接受平面是曲面或者与光源不平行的时候就会出问题。

VSSM的做法实在是十分聪明,采用了非常多的大胆假设，同时非常的快，没有任何噪声，本质上其实也没有用正态分布，是直接用切比雪夫不等式来进行近似。但是现在最主流的方法仍然是PCSS,因为人们对噪声的容忍度变高加上降噪的技术越来越高明,因此大多数人采用PCSS.

MIPMAP和SAT

VSSM中如何加速第一步和第三步的我们知道了,那么如何在区域范围内快速的求出均值呢?

有两个方法:MIPMAP 和 SAT.

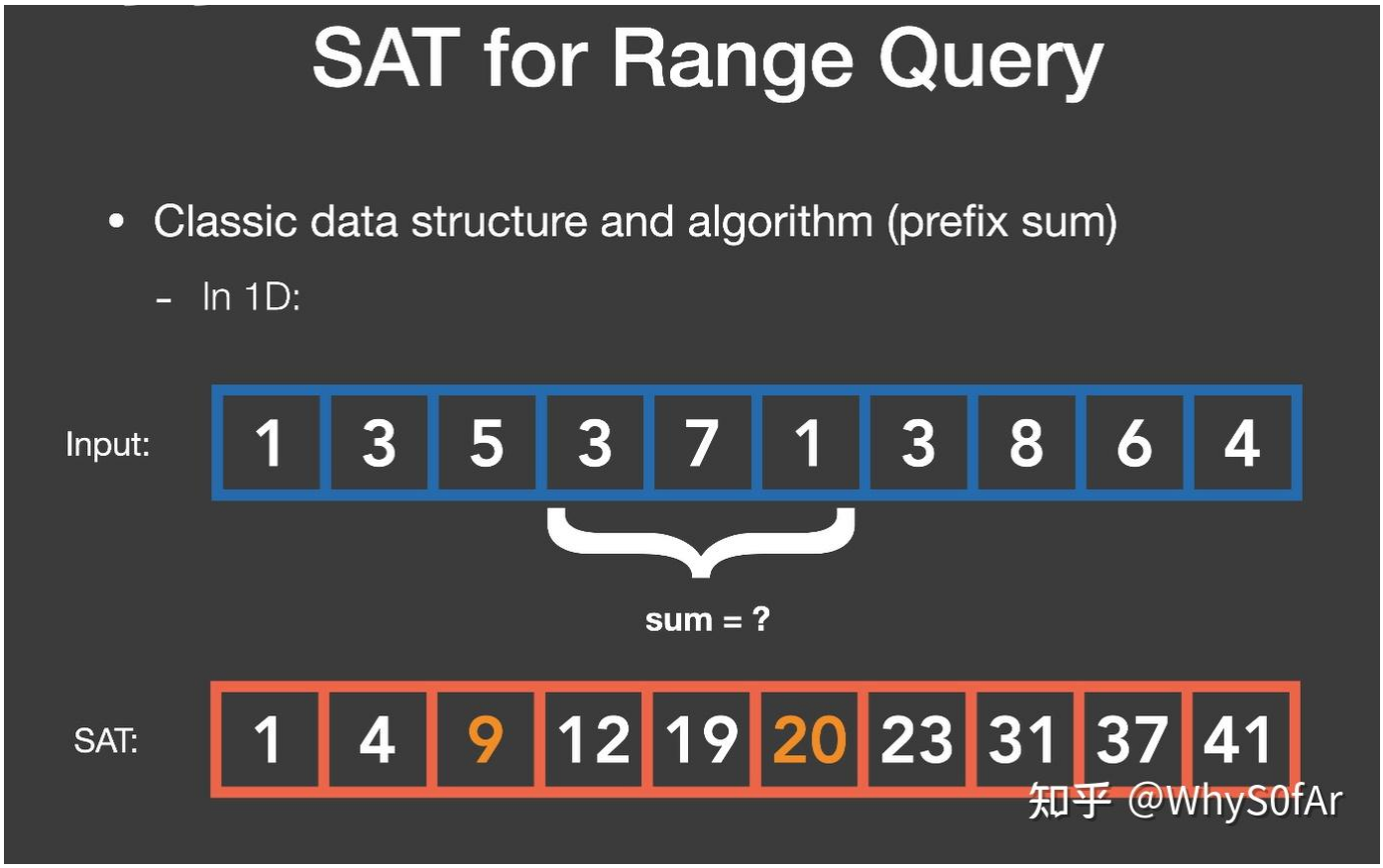


最简单的方法自然是MIPMAP,我们在GAMES101里学过,他是一个**快速的,近似的,正方形的范围查询,** 由于他要做插值,因此即便是方形有时也会不准确.同时当插值的范围不是2的次方时，也就是在两个MIPMAP之间时，还要再进行一次插值，也就是“三线性插值”，这样会让结果更加不准确,因此局限性太大且准确度也不算高.

但是SAT是百分百准确的一个数据结构.SAT的出现是为了解决范围查询(在区域内快速得到平均值),并且,范围内求平均值是等价于范围内求和的,毕竟总和除以个数=平均值.

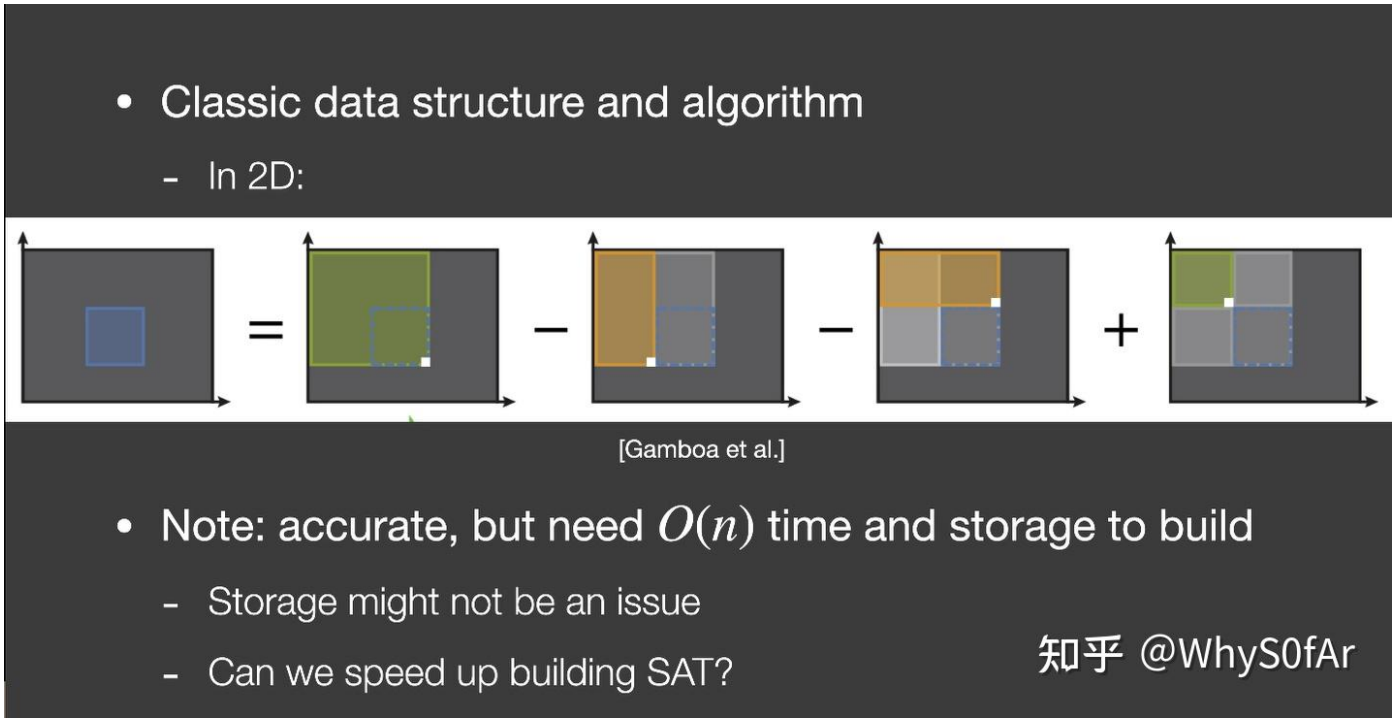
在1维情况下其实就是一维数组,SAT这种数据结构就是做了预处理，也就是在得到一维数组时,先花费O(n)的时间从左到右走一遍,并且在走的同时把对应的累加和存入数组SAT中，那么SAT上任意的一个元素就等于原来数组从最左边的元素加到这个元素的和,如图SAT数组第2个元素表示原数字前2个元素之和。

那么查询下图中SUM区域总和时候，就是SAT数组中第六个元素减去第三个元素。相当于使用了$O(n)$的时间，把预计算做了一遍。



那么在2维（二维数组）情况下 有一个任意矩形（横平竖直），蓝色矩形内的总和为两个绿色矩形内的总和减去两个橙色矩形内的总和。同样可以采用SAT 的方法，做一个表，做出从左上角到这个元素的和。因此只需要**查表4次** 就可以得出精准的区域求和。

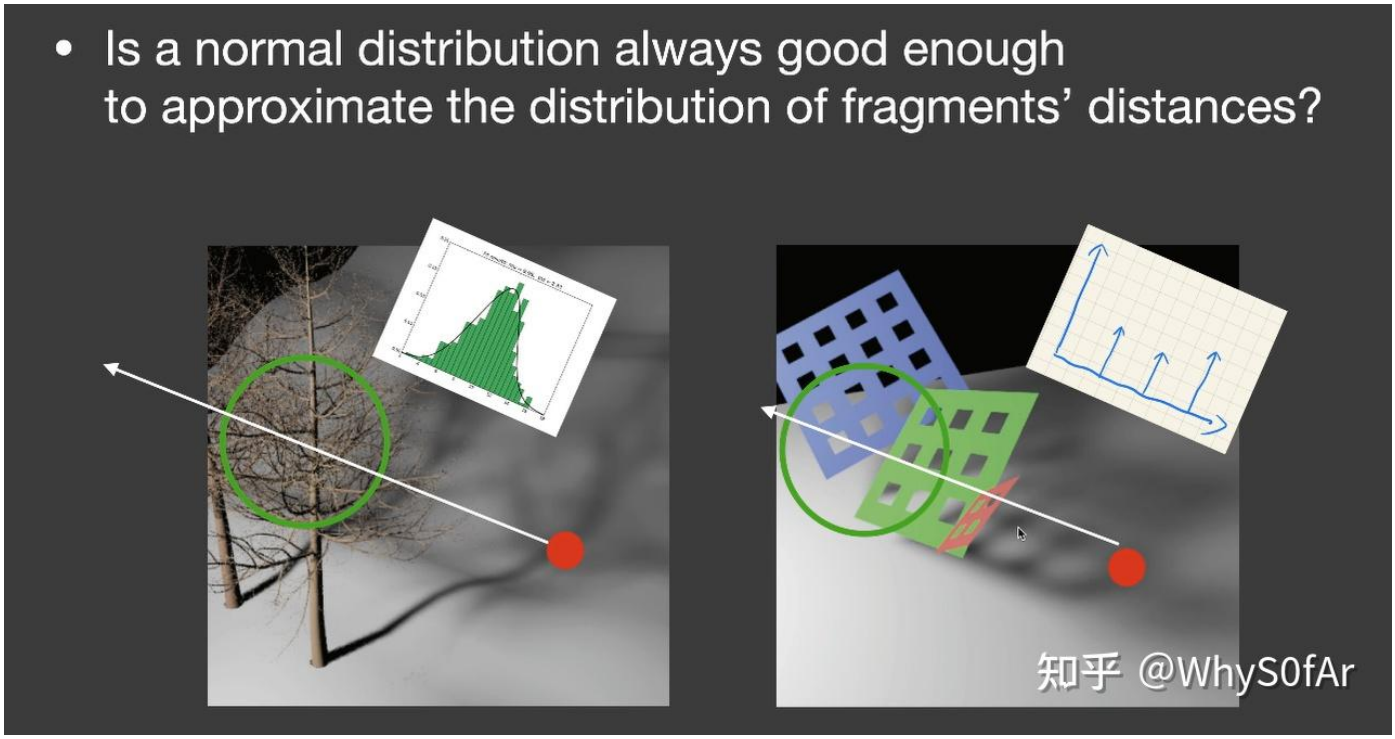
在2维情况下，可以通过建立m行中每行的SAT和在每行的sat基础上再建立n列中每列的SAT，最终可以获得一个2维的SAT因此最终的SAT,但是由于gpu的并行度很高,行与行或列与列之间的sat可并行,因此具有m×n的时间复杂度。



其实这个算法就是leetcode 1314 matrix block sum的算法.

Moment shadow mapping

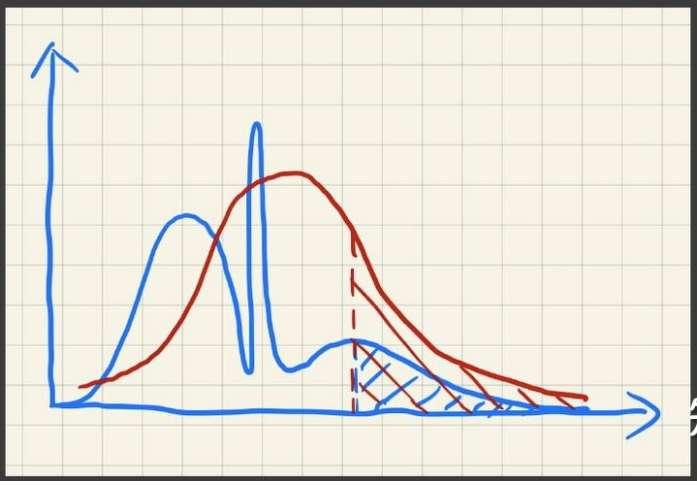
VSSM是为了解决PCSS的问题,但vssm由于做了很多假设，当假设不对的时候会有问题。



比如右图，只有三个片的遮挡的情况下，那么深度的分布就在这三个遮挡度深度周围，形成了三个峰值，自然就会出现假设描述的不准。

Revisit: VSSM

- Issues if the depth distribution is inaccurate
  - Overly dark: may be aceeptable
  - Overly bright: LIGHT LEAKING!

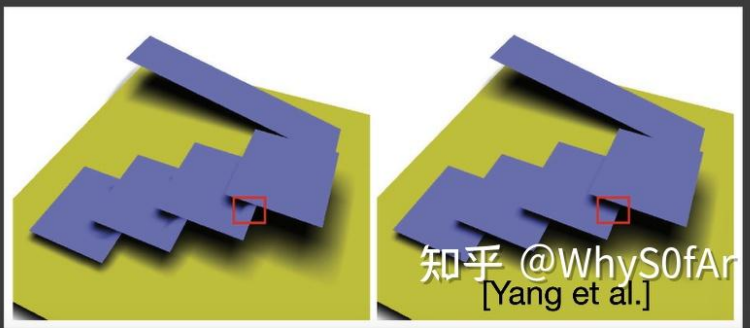


不是正态分布强行按正态分布算就会出现漏光和过暗的结果。
在阴影的承接面不是平面的情况下也会出现阴影断掉的现象。

Revisit: VSSM

- Limitations?
  - Light leaking
  - non-planarity artifact
- Chebychev is to blame?
  - Only valid when $t > z_{avg}$
- Can we do better?

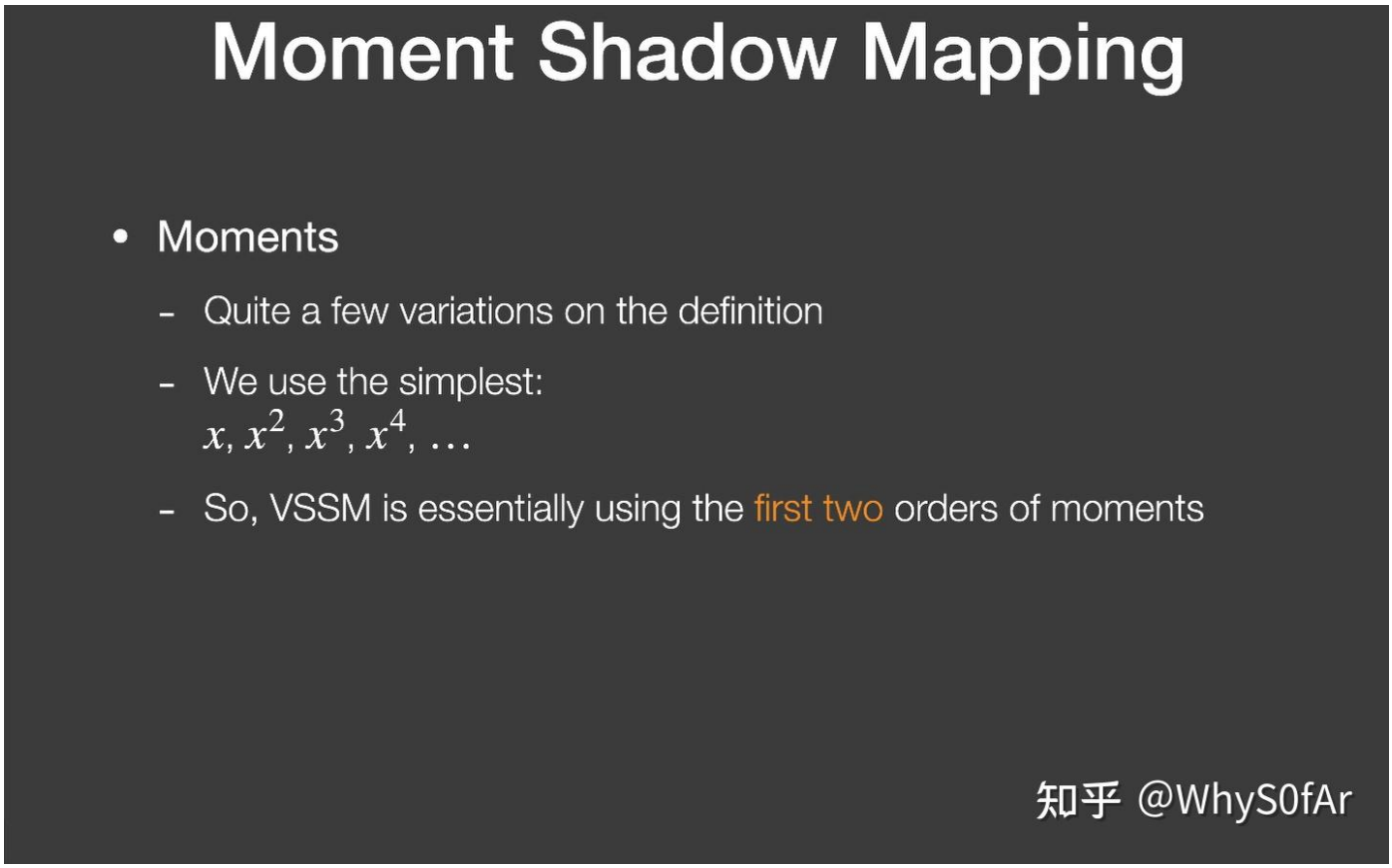
[https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch08.html]



[Yang et al.]

我们看图中小车可以发现,在车的底部阴影部分出现了一部分偏白的阴影,这是因为车是一个镂空的状态,如果从底部向LIGHT处看去会发现底板遮挡一部分,车顶附近会遮挡一部分,这就导致了不是正态分布情况,因此才出现light leaking这种情况.

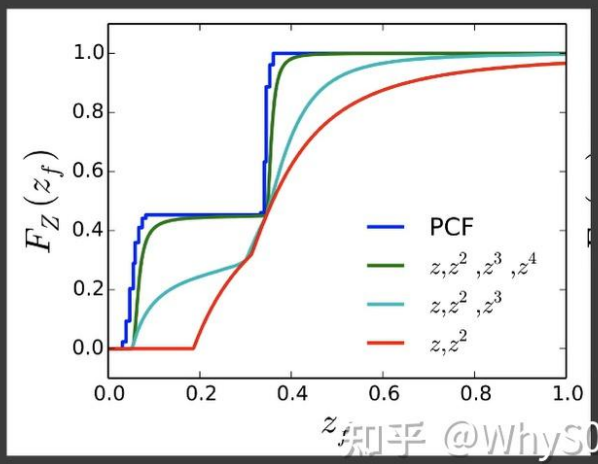
因此人们为了避免VSSM中不是正态分布情况下的问题,就引入了更高阶的moments来得到更加准确的深度分布情况.想要描述的更准确，就要使用更高阶的moment(矩)，矩的定义有很多，最简单的矩就是记录一个数的次方，VSSM 就等于用了前两阶的矩。这样多记录几阶矩就能得到更准确的结果。



如果保留前M阶的矩，就能描述一个阶跃函数，阶数等2/M,就等于某种展开。越多的阶数就和原本的分布越拟合。一般来说4阶就够用。

Moment Shadow Mapping

- What can moments do?
  - Conclusion:
    first m orders of moments can represent a function with $m/2$ steps
  - Usually, 4 is good enough to approximate the actual CDF of depth dist.
  - ~~How to restore a CDF whose moments match the given moments~~



[Peters et al., Moment Shadow Mapping]