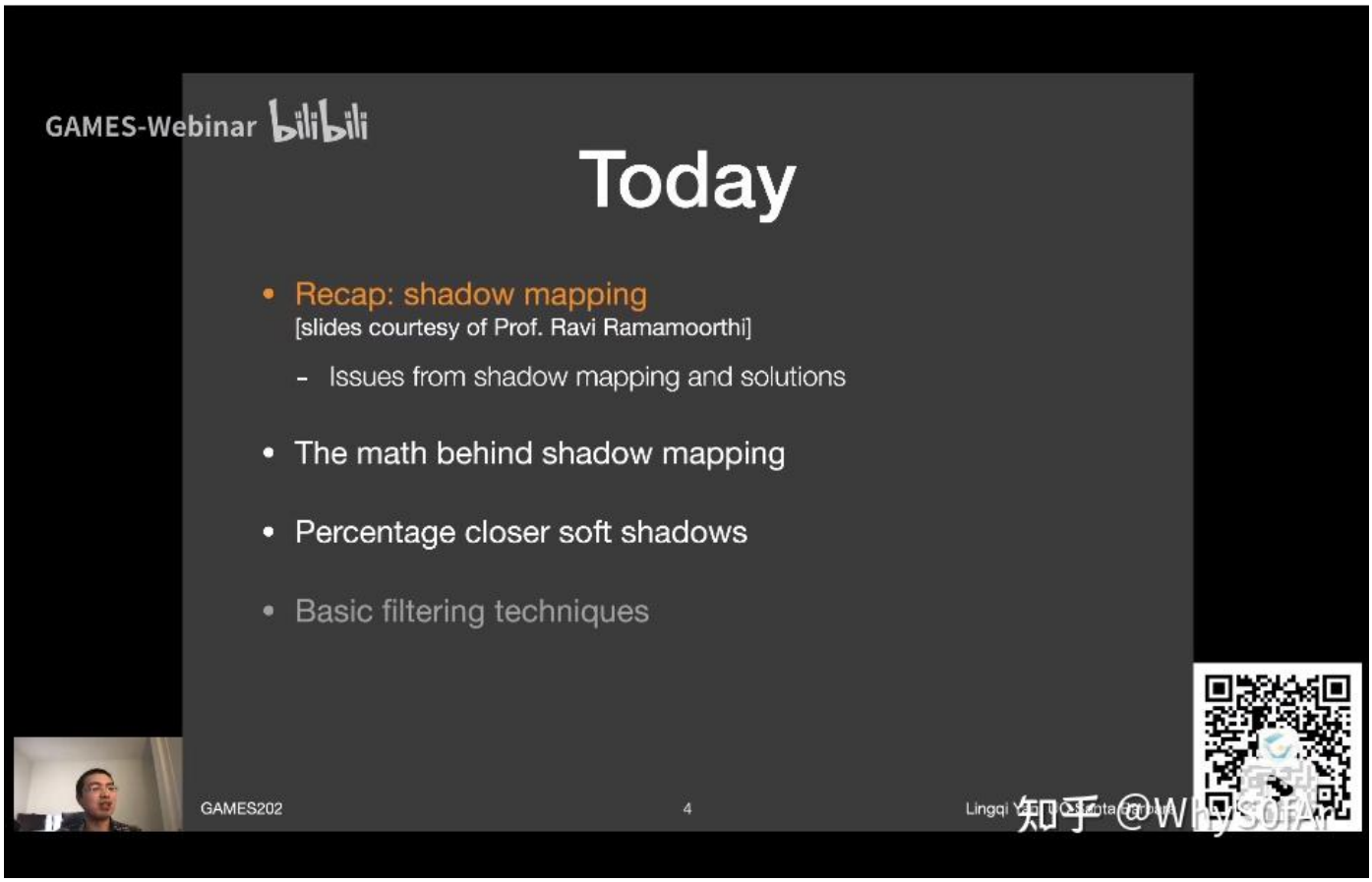
GAMES202 Real-Time High Quality Rendrting 高质量实时渲染课程笔记Lecture 3- Shadow 01

本文是关于B站闫老师的GAMES202课程的第三讲：实时阴影 PART 01：

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



课程目录

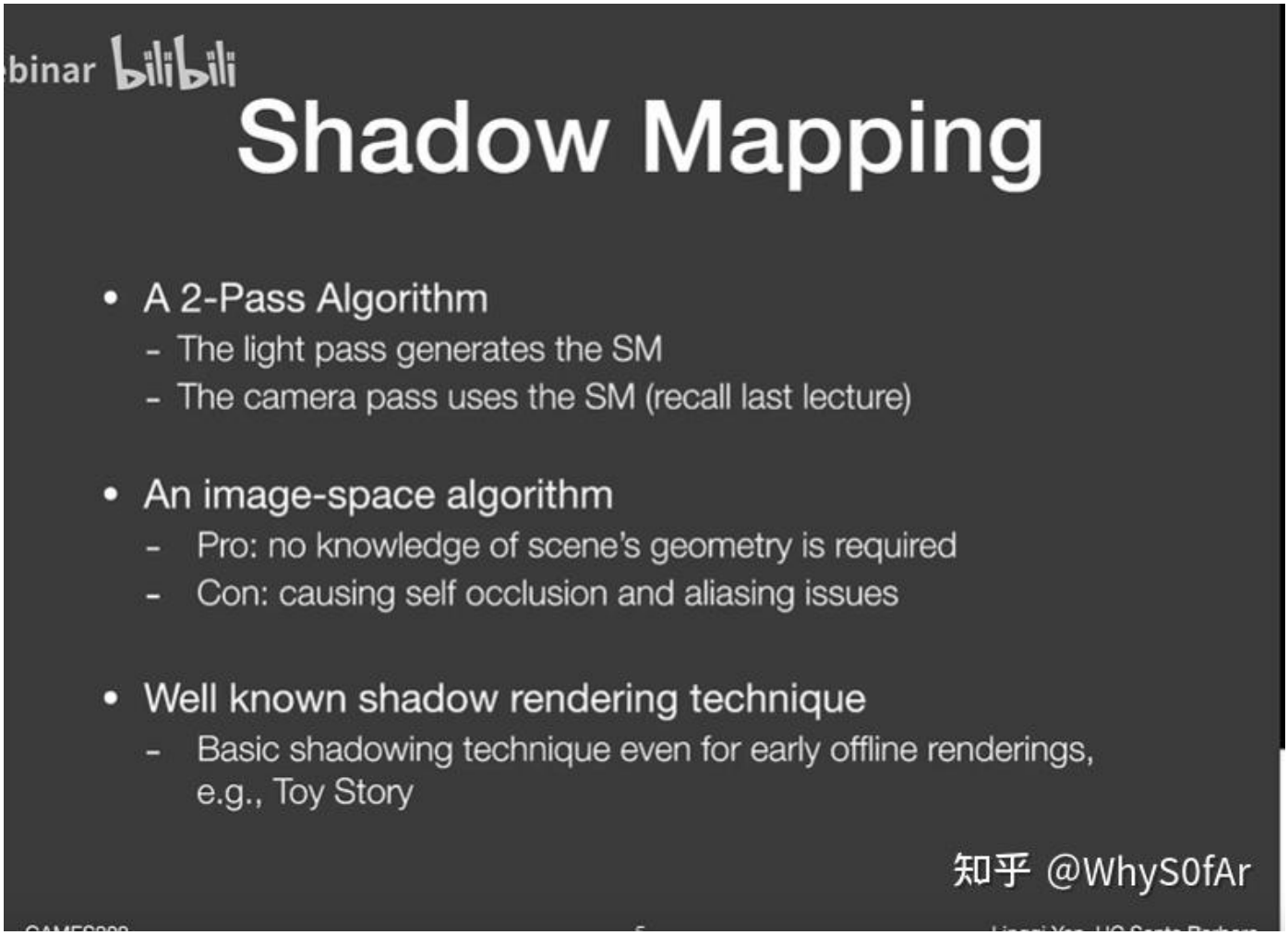
本节课的内容主要分为三部分：

1.Shadow mapping的回顾

2.Shadow mapping的数学基础

3.PCSS和PCF

i - > Shadow Mapping



要渲染一个点光源在场景中投射出的阴影,我们要用shadow mapping技术,在101中我们介绍过shadow mapping,它是一个2-pass的算法,也就是我们会对场景渲染两次,1-pass我们从Light处看向场景并输出一个从light处看向场景所生成的深度图,也就是所谓的shadow map.

2-pass我们从camera处看向场景渲染一遍将,并参考1-pass生成的shadow map去判断物体是否在阴影中.

shadow mapping是一个完全在图像空间中的算法,**其优点:**

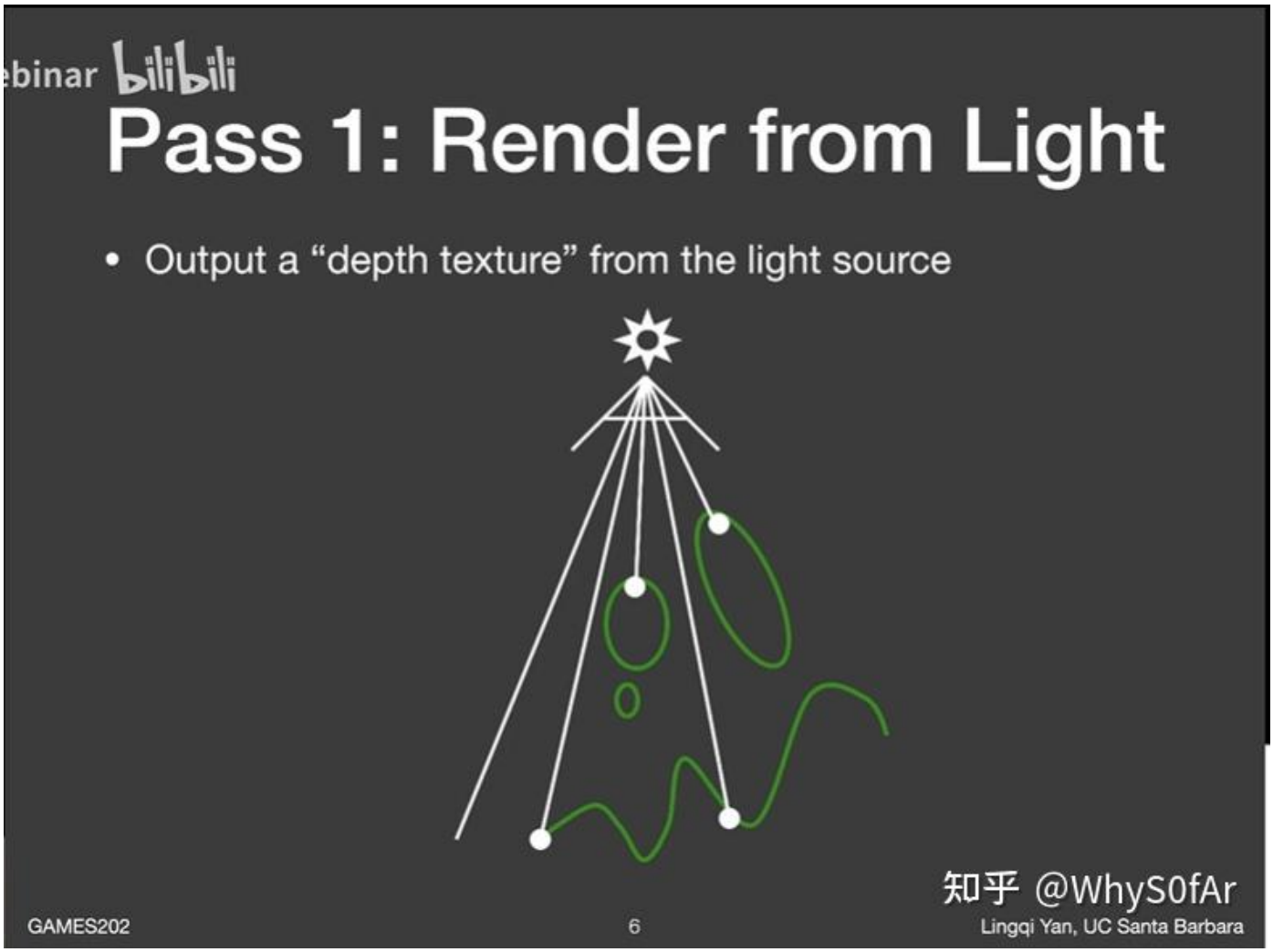
一旦shadow Map已经生成,就可以利用shadow map来获取场景中的几何表示而不是场景中的几何.

其缺点:

会产生自遮挡和走样现象.

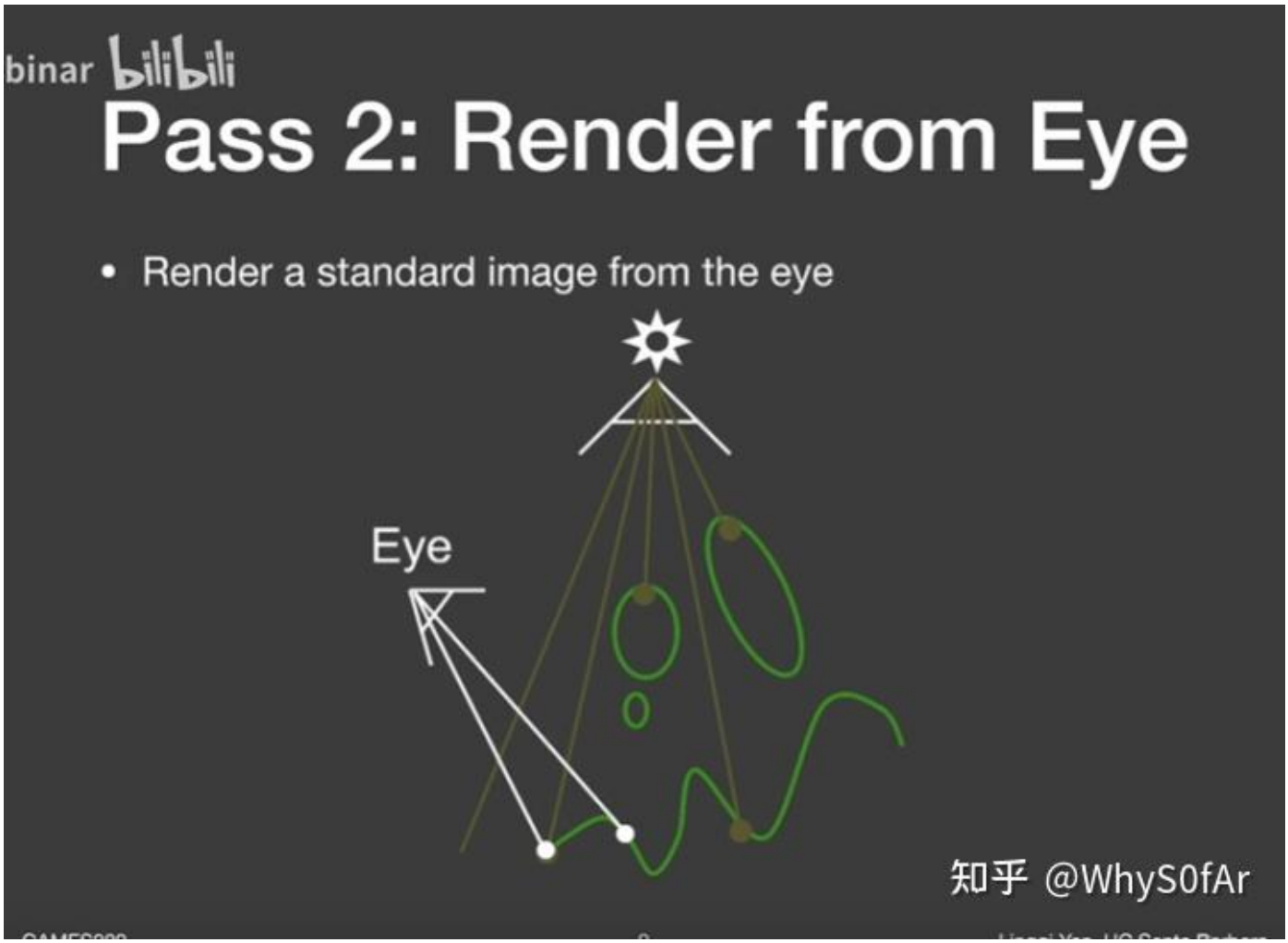
步骤:

1.从light处出发看向场景生成一张记录每个像素中最近物体深度的一张图,如图:

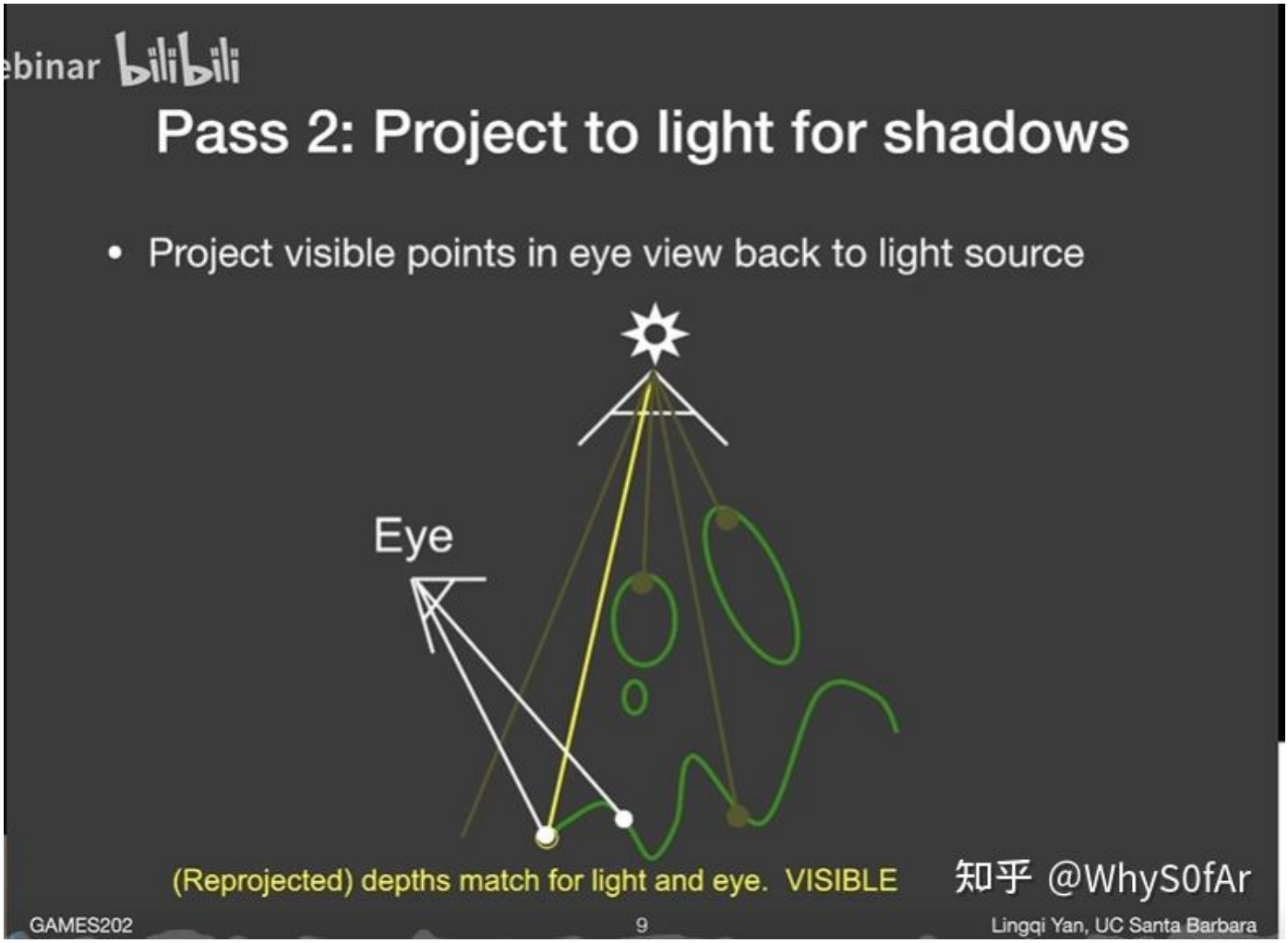


图中有很多像素,我们记录下每个像素中他们各自看到的最浅的深度或是最近的物体他们的位置在哪,存下来从而得到一张texture.

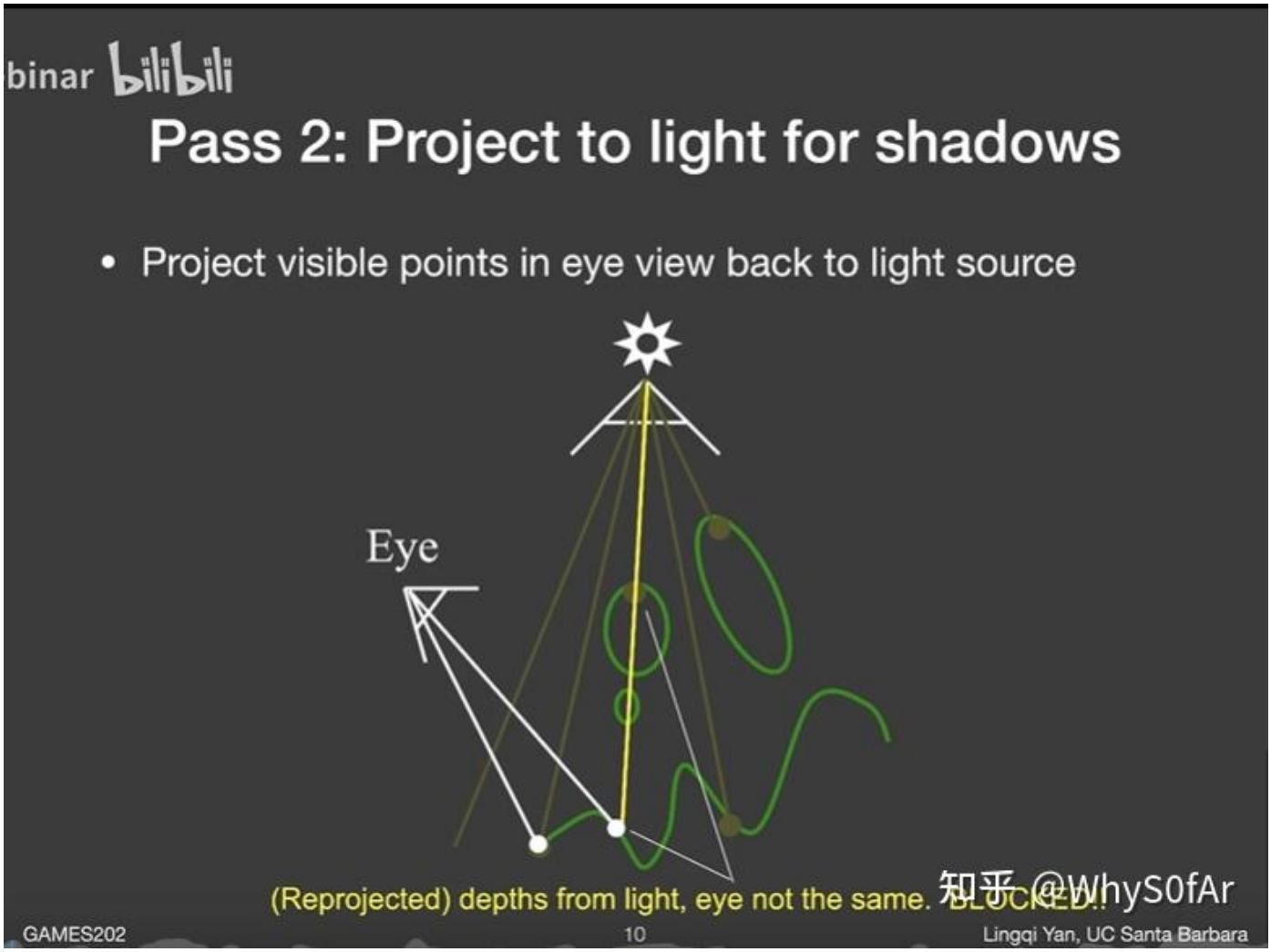
2.我们从camera(眼睛)出发,在渲染一遍场景.



对于这次我们来看渲染中的每个像素中都要来判断是否能被light照到,如果被照到则不在阴影中,反之,则存在于阴影中.

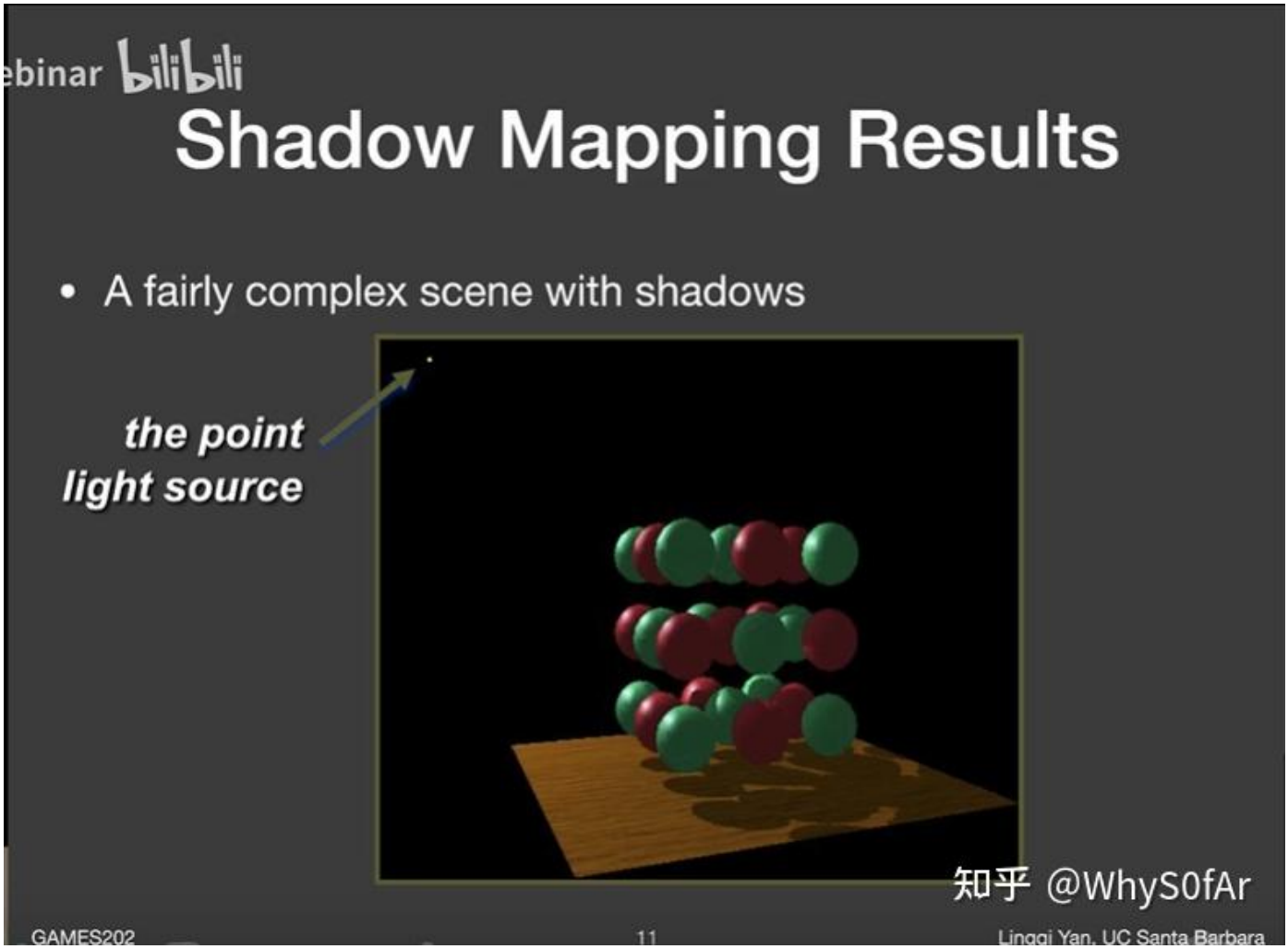


对于这个点来说,如果将这点连向light处会发现在1-pass得到的shadow map中,这一点的最浅深度 = 2-pass中从点连到light处的深度(2-pass中得到的点是可以投影回light处求出距离的),因此这一点是可见的,不在阴影中.

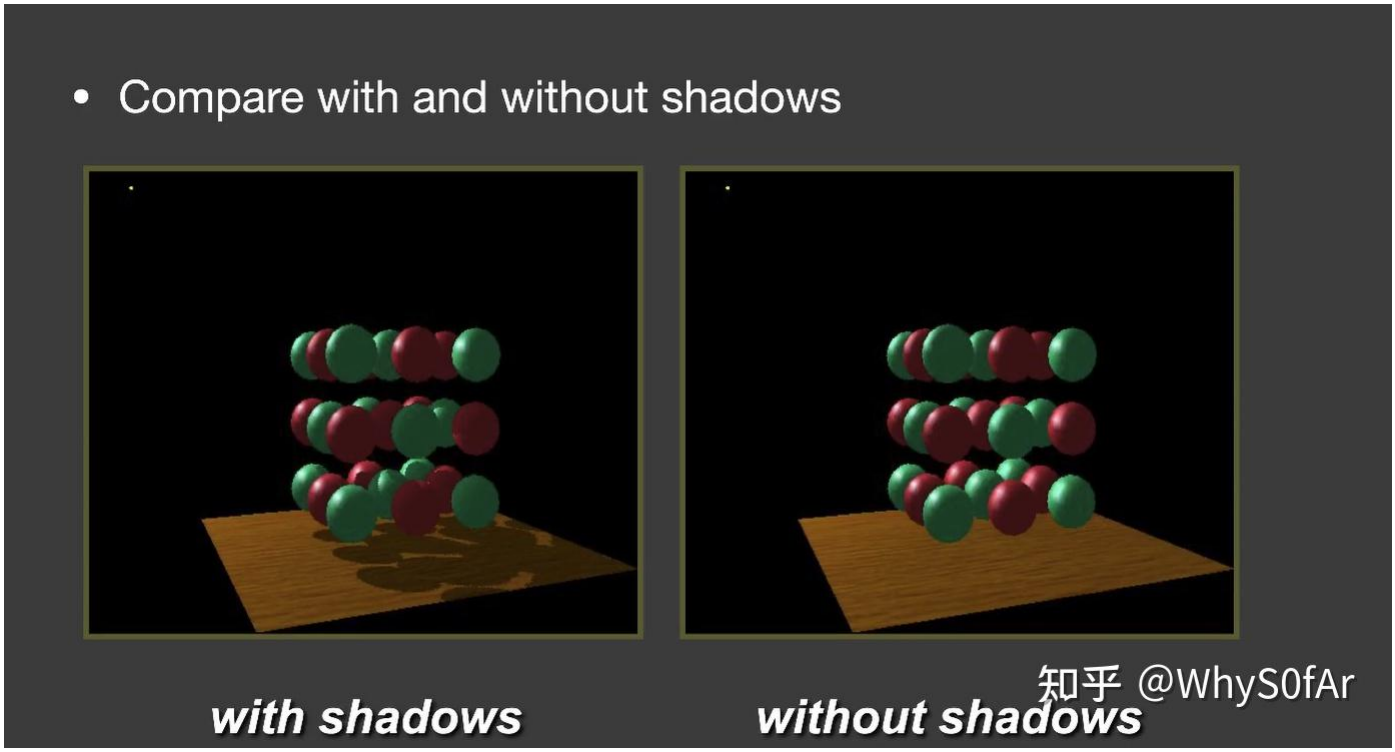


而对于这个点来说,仍然连向light处会发现:

1-pass的shadow map中这一点的最浅深度 < 2-pass中从点连到light处的深度,因此这一点是被遮挡的,在阴影中.



从这图可以看到,light在左上方,场景中的遮挡关系和阴影表示出的效果是很不错的.

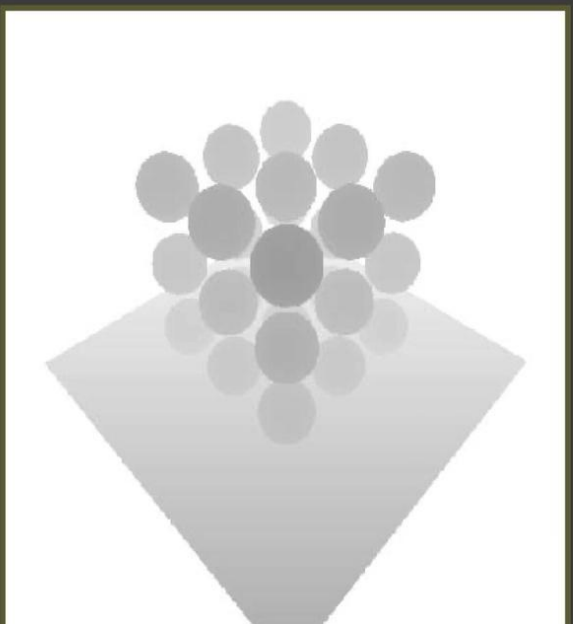


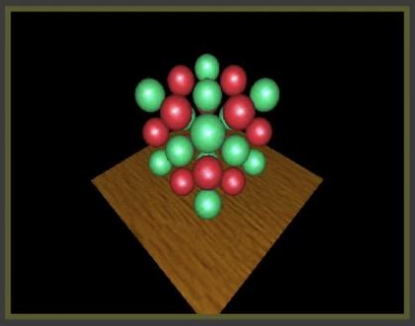
有阴影 VS 无阴影

从Light处看向场景生成的是shadow map,并不是Shading的结果,而是在light的pass中生成一张深度的buffer,如图中,颜色深的表示值比较小,也就是离light近,颜色浅的就是离light远,值大.

Visualizing Shadow Mapping

- The depth buffer from the light's point-of-view





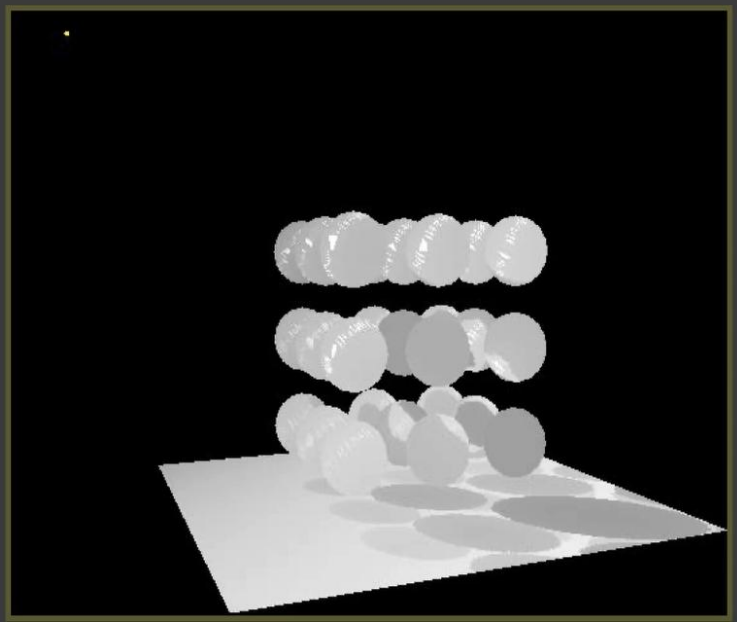
FYI: from the light's point-of-view again

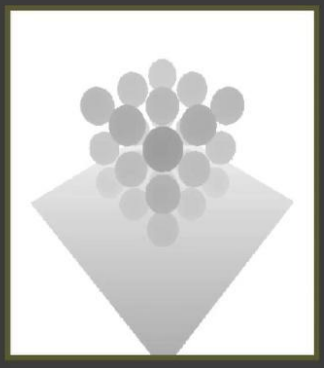
知乎 @WhyS0fAr

1-pass生成的深度buffer

Visualizing Shadow Mapping

- Projecting the depth map onto the eye's view

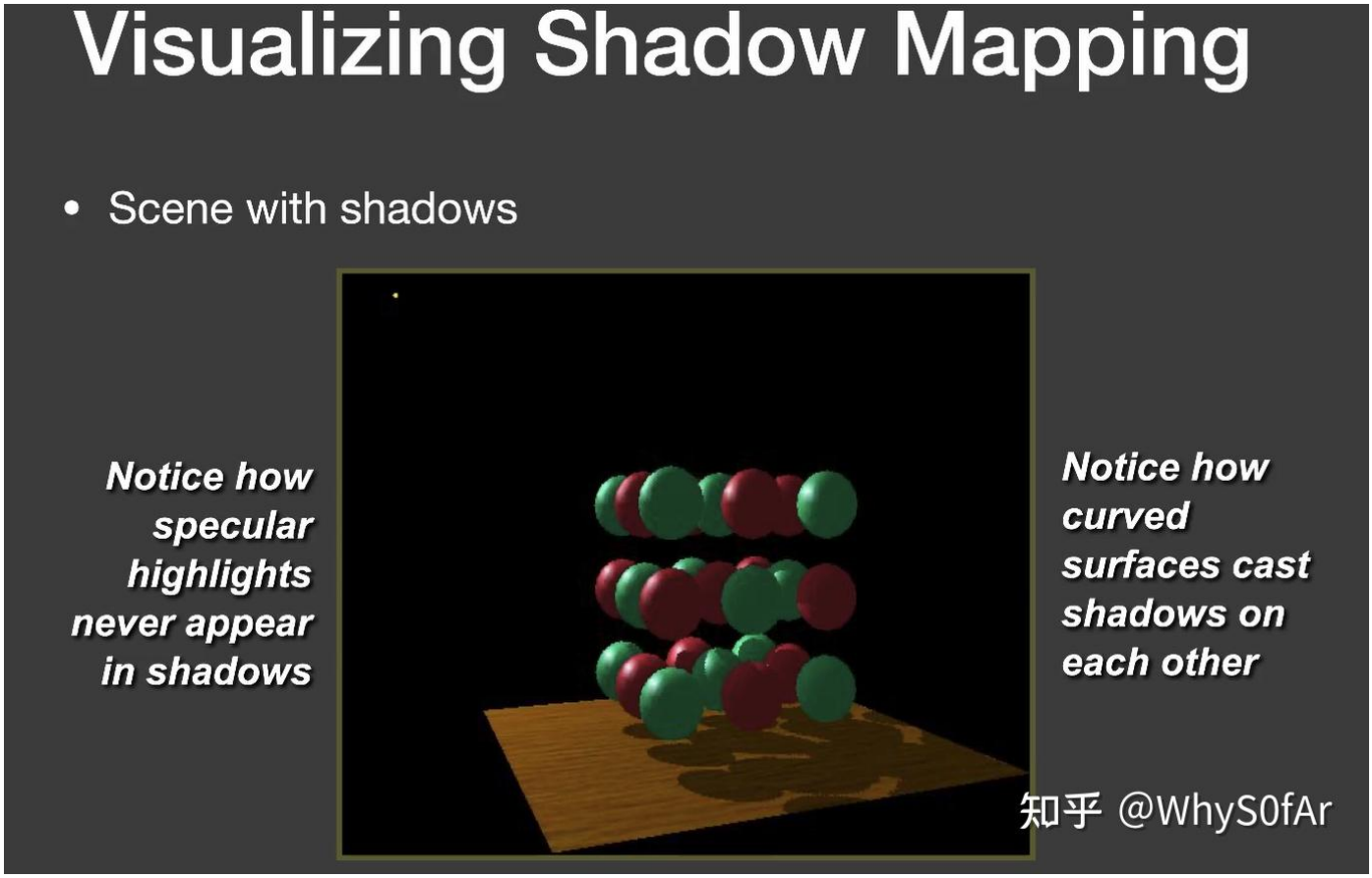




FYI: depth map for light's point-of-view again

知乎 @WhyS0fAr

2-pass生成的深度buffer

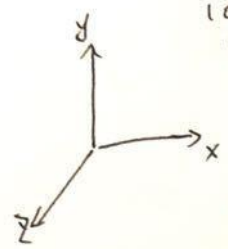


在Opengl中,1-pass真就是在light处放置一个相机,然后往某一方向看去,定义framebuffer写到某个texture上,然后在fragment shader中定义写的是一个深度而非shading的结果,在2-pass中则只需要用1-pass得到的texture即可.

关于深度,我们在101中讨论过一个问题,在做透视投影时,我们是将透视投影挤压成正交投影,然后再拍平,在这个过程中中间的点是会向近平面移动还是向远平面移动?

答案是中间的所有点会被推向远平面.

http://games-cn.org/forums/topic/guanyulecture-04-kehousikaoti-xiangduigedaan/



(0,0,0) Origin look at $-z$.

So. $0 > n > z > f$.

$$M_{persp\to ortho} = \begin{bmatrix} n,0,0,0 \\ 0,n,0,0 \\ 0,0,n+f,-nf \\ 0,0,1,0 \end{bmatrix}$$

Special: $z = \frac{n+f}{2}$

$$z\begin{bmatrix} 0 \\ 0 \\ \frac{f+n}{2} \\ 1 \end{bmatrix} \longrightarrow \begin{bmatrix} n,0,0,0 \\ 0,n,0,0 \\ 0,0,n+f,-nf \\ 0,0,1,0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ \frac{f+n}{2} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \left(\frac{f+n}{2}\right)^2 - nf \\ \frac{n+f}{2} \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \\ \frac{n^2+f^2}{n+f} \\ 1 \end{bmatrix} z'$$

$f < z < n < 0$

$$\frac{n^2+f^2}{n+f} - \frac{f+n}{2} = \frac{(n-f)^2}{2(f+n)} \quad \begin{matrix} >0 \\ <0 \end{matrix}$$

$$\frac{n^2+f^2}{n+f} < \frac{f+n}{2} < 0.$$

$z' < z$ closer to far.

More General $z\begin{bmatrix} 0 \\ 0 \\ z \\ 1 \end{bmatrix} \longrightarrow M_{persp} = \begin{bmatrix} 0 \\ 0 \\ z(n+f)-nf \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ n+f-\frac{nf}{z} \\ 1 \end{bmatrix} z'$

$f < z < n < 0$. $z-f > 0$ $z-n < 0$.

$$z - z' = z - \left(n+f-\frac{nf}{z}\right)$$

$$= (z-n) - f\left(1-\frac{n}{z}\right)$$

$$= (z-n) - \frac{f}{z}(z-n)$$

$$= (z-n)\left(1-\frac{f}{z}\right)$$

$$= \frac{1}{z}(z-n)(z-f) > 0.$$

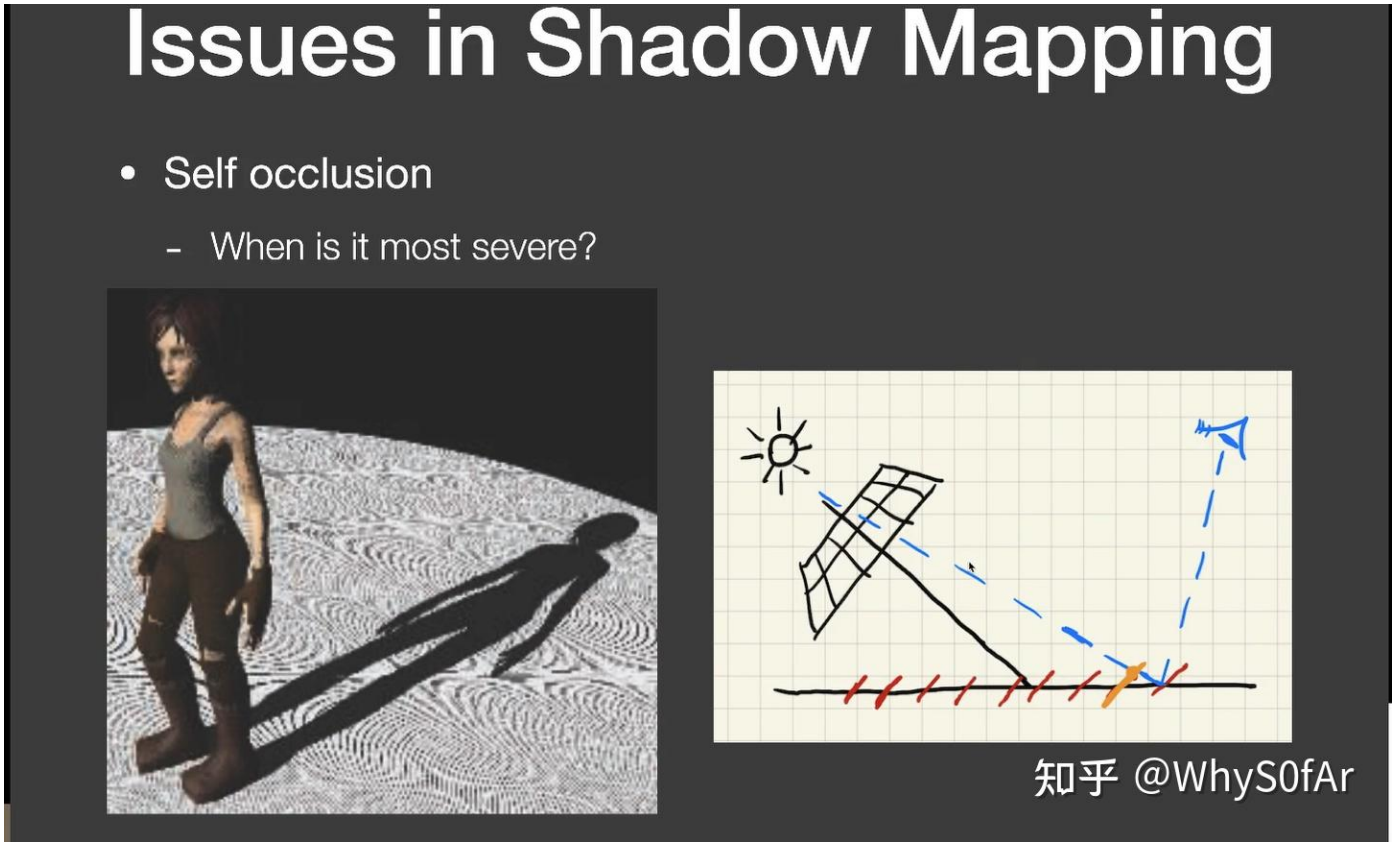
<0 <0 >0.

So. $z > n+f-\frac{nf}{z} = z'$ closer to far.

我们想说的是,在经过投影之后得到的Z其实不是实际上几何上的点到Light的距离,因此再真正生成阴影时比较两个pass中的depth时需要一致,也就是要么都用**投影后的Z值** 比较,要么通过**两点的位置得一向量算实际距离** .

Shadow Mapping也是存在一些问题的:

A： 自遮挡



学过101的看见地板上的东西会感觉像摩尔纹,但并不是,它是由数值精度造成的一种现象.

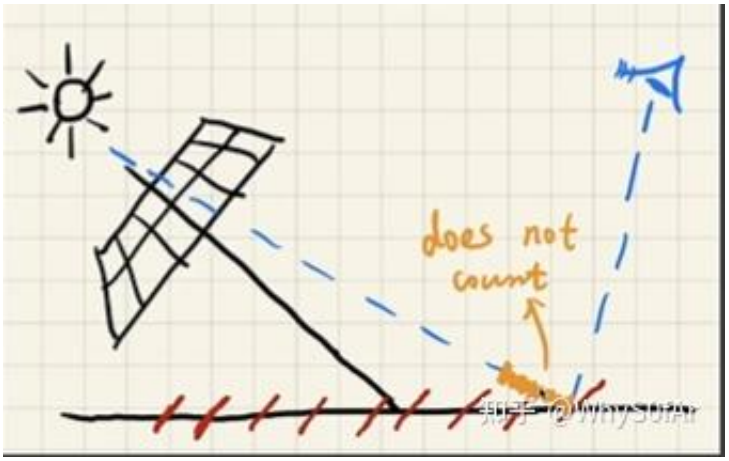
我们生成了一个Shadow map,那么这个Shadow map肯定有自己的分辨率,其每个像素要记录它所看见的最浅深度,可以理解为,每个像素内部他的深度是一个**常数**

那么从Light处看向场景时,沿某一像素看过去我们看到的位置就认为是像素所代表的深度,也就是认为这个场景在像素覆盖区域内都是一个常数的深度.

在Shadow map看来,场景被离散化为一系列红色小片形成的场景而不是直接的平面.

因为每个红色小片所代表的深度不一样,因此在2-pass,也就是从camera处看向场景时,后面的红色小片在连接light时,会被误认为被前面的红色小片遮挡住,从而产生了错误的阴影,这个现象在light与平面趋于平行时候最严重.

有了问题自然要去解决问题:



我们只需要把在黄色区域内的遮挡关系给舍弃掉就可以了,而且是有技巧的,当Light处垂直于场景时,我们可以让这个遮挡区域尽可能小一点,当light趋近于平行场景时,我们让这个区域尽可能大一点,我们可以引入一个bias的概念(黄色区域),来降低自遮挡情况,bias是根据角度调整的,并非常数.

具体方式就是当一个点深度大于记录深度的值超过一个阈值时，我们才认为这个点在阴影内。这也是工业界使用较多的一个办法。

但是这样也引出了另一个问题:

detach shadow

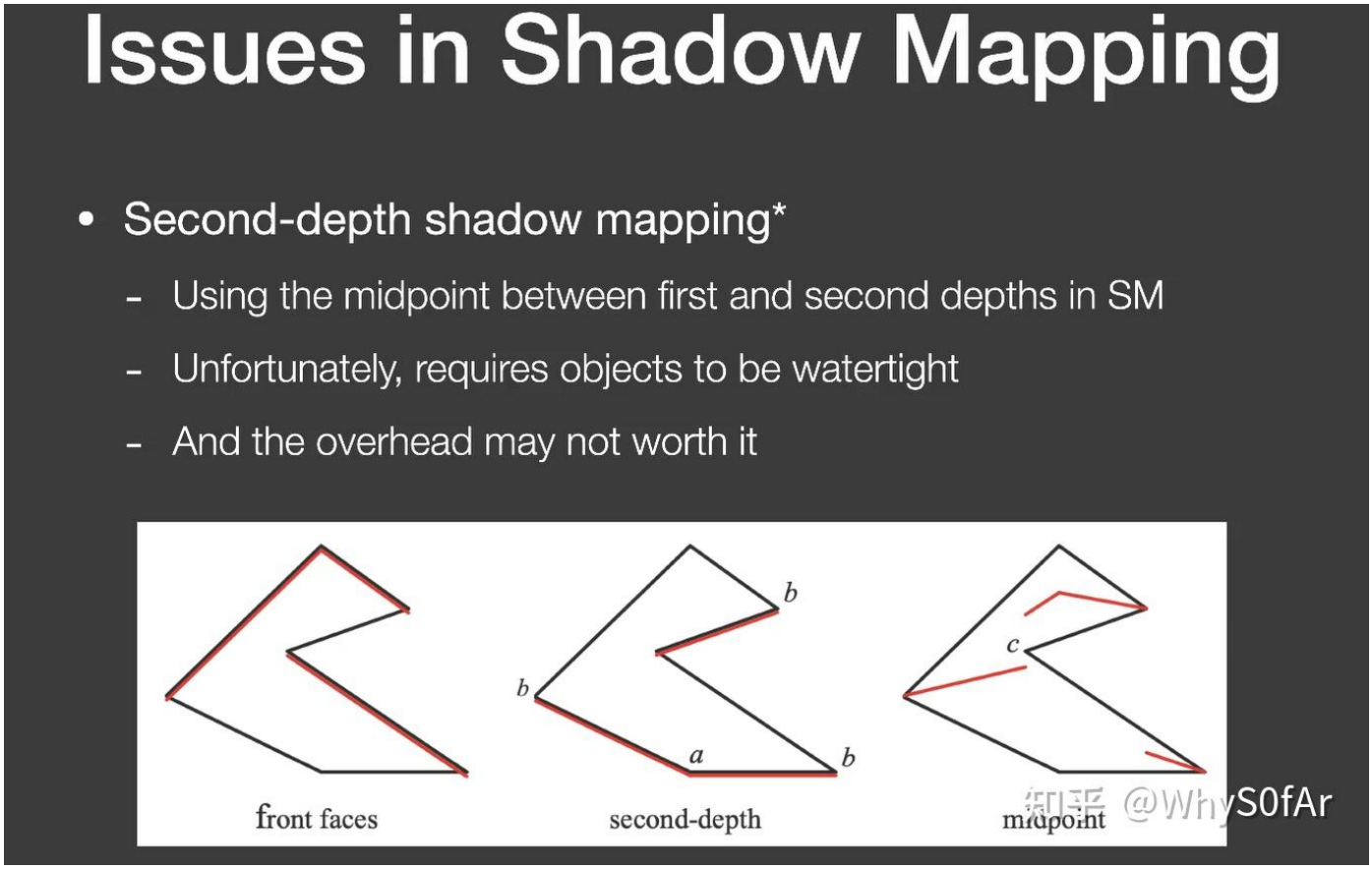
从图中我们可以看到,有一段的阴影被舍弃了,因为当Bias调的过大时,我们会丢失原本应该存在的阴影.

解决方案:

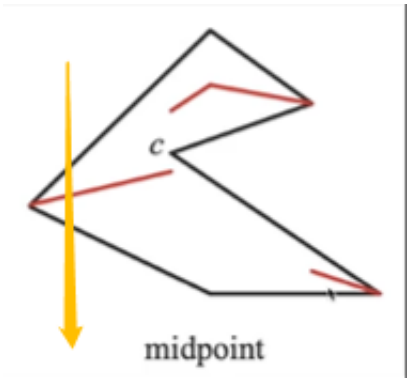
Second-depth shadow mapping

此时我们就舍弃biasd的概念,而是在渲染时不仅存最小的深度,我们还要存第二小的深度,然后我们用最小深度和第二小深度中间的深度来作比较.

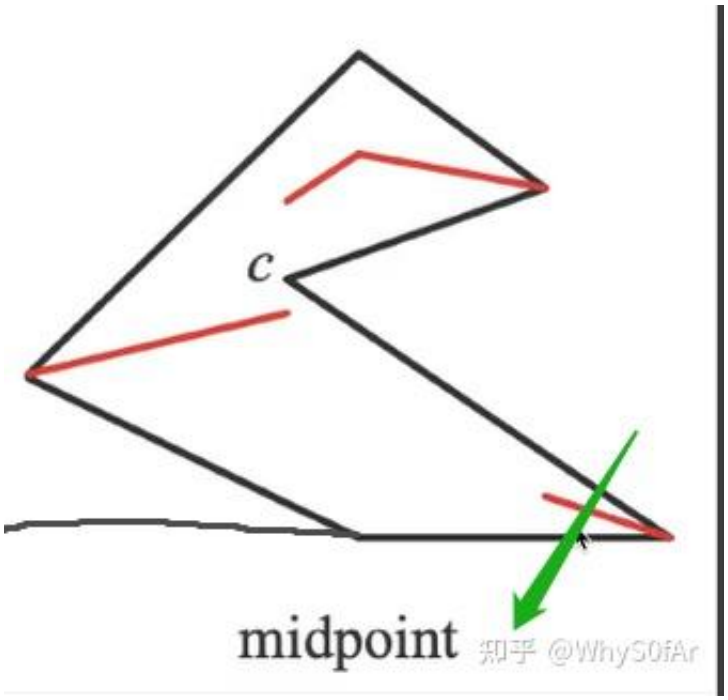
(个人理解为,我们在渲染时生成两张深度图,一张存储的是最浅,一张存储的是第二浅,之后将两张图平均从而得到最后的深度图,然后用这张平均得来的深度图和物体算遮挡关系)



以这个为例:



假设一根光线照过来,我们不用最小深度来比较,而是用由最小和第二小深度所得到的红色线来做后续的阴影,此处就没有Bias的事情了.



假设图中是人物的鞋,鞋底是接着一个平面的,我们以右边最极端的部分为例,当一根光线从右边鞋头部分打向鞋底,就算它底部紧贴着平面,我们能得到一个明显的遮挡关系.

然后实际中并没有人去使用这个技术,因为场景内的物体必须都是watertight（非面片），还有就是算的复杂，开销太大，实时渲染不相信复杂度。

只相信绝对的速度!

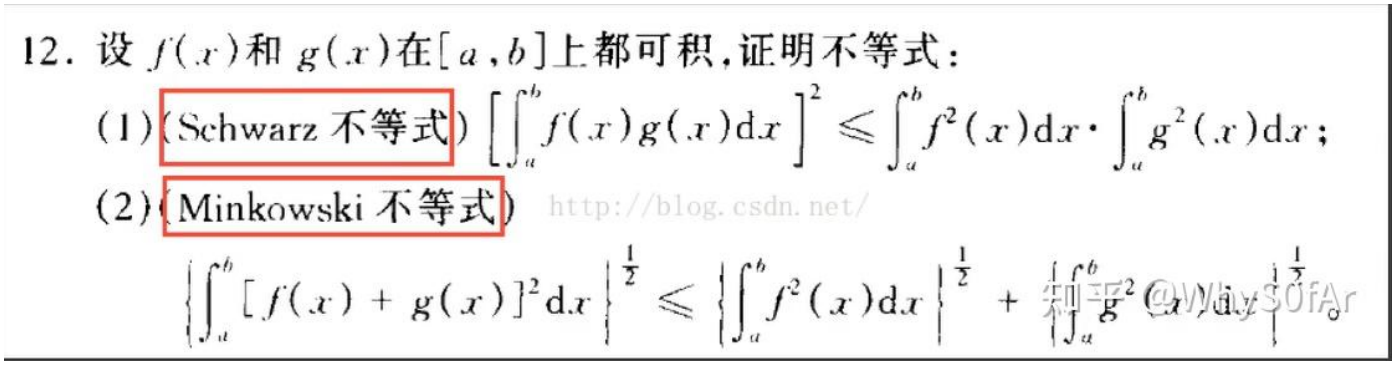
B : 走样

Shadow mapping的第二个问题就是走样。

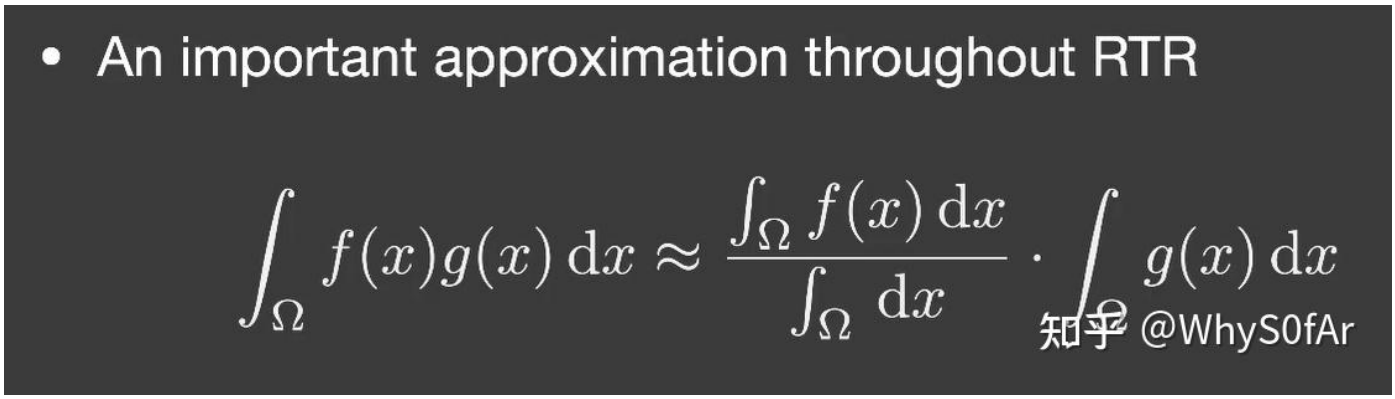
shadow map本身就存在分辨率，当分辨率不够大自然会看到锯齿,因为shadow map上每一个像素都可以看为小片,那么投出来的阴影自然会存在锯齿.

i i - > The math behind shadow mapping

在微积分中有很多有用的不等式,如图中的两个不等式为例:



但是在实时渲染中,我们只关心近似约等,我们不考虑不等的情况,因此我们将这些不等式当约等式来使用.



实时渲染中重要的一个约等式

如果你有两个函数的乘积,你又想把他们的乘积积分起来,你可以将其拆出来,也就是:

两个函数乘积的积分 ≈ 两个函数积分的乘积

首先我们来解释一下为什么右边第一个函数多了个分母.

分母这一项的作用是为了保证左右能量相同而做的归一化操作。

我们来用一个例子来解释这个归一化操作。我们假设f(x)是一个**常值函数** ,也就是f(x) = 2,我们的积分域恒为0-3.

那么约等式左边,把f(x) = 2代入,则可以提出来变为2倍的g(x)积分

而等式右侧第一个函数代入f(x)的积分是2 * 3 =6，分母的积分是3，结果也正好是2.正好也是2倍的g(x)积分.

接下来我们来讨论在什么情况下约等式结果更加准确:

一般需要以下两个条件:

1. **g(x)积分的support较小。这里的support我们可以暂时理解为积分域。**

2.g(x)在积分域上足够光滑（变化不大）。

此时我们把rendering equation代入这个约等式中:

$$L_o(\mathrm{p},\omega_o) = \int_{\Omega^+} L_i(\mathrm{p},\omega_i) f_r(\mathrm{p},\omega_i,\omega_o)\ \cos\theta_i\, V(\mathrm{p},\omega_i)\,\mathrm{d}\omega_i$$

我们把visibility看作是f(x),提取出来并作归一化处理:

- Approximated as

$$L_o(\mathrm{p},\omega_o) \approx \frac{\int_{\Omega^+} V(\mathrm{p},\omega_i)\,\mathrm{d}\omega_i}{\int_{\Omega^+}\mathrm{d}\omega_i} \cdot \int_{\Omega^+} L_i(\mathrm{p},\omega_i) f_r(\mathrm{p},\omega_i,\omega_o)\ \cos\theta_i\,\mathrm{d}\omega_i$$

知乎 @WhyS0fAr

红色区域部分时visibility,那么剩下的g(x)部分,也就是shading的结果.

因此其表示的意义就是,我们计算每个点的shading，然后去乘这个点的visibality得到的就是最后的渲染结果。

这也就是shadow mapping的基本思想。

那么什么时候这个约等式比较正确呢?

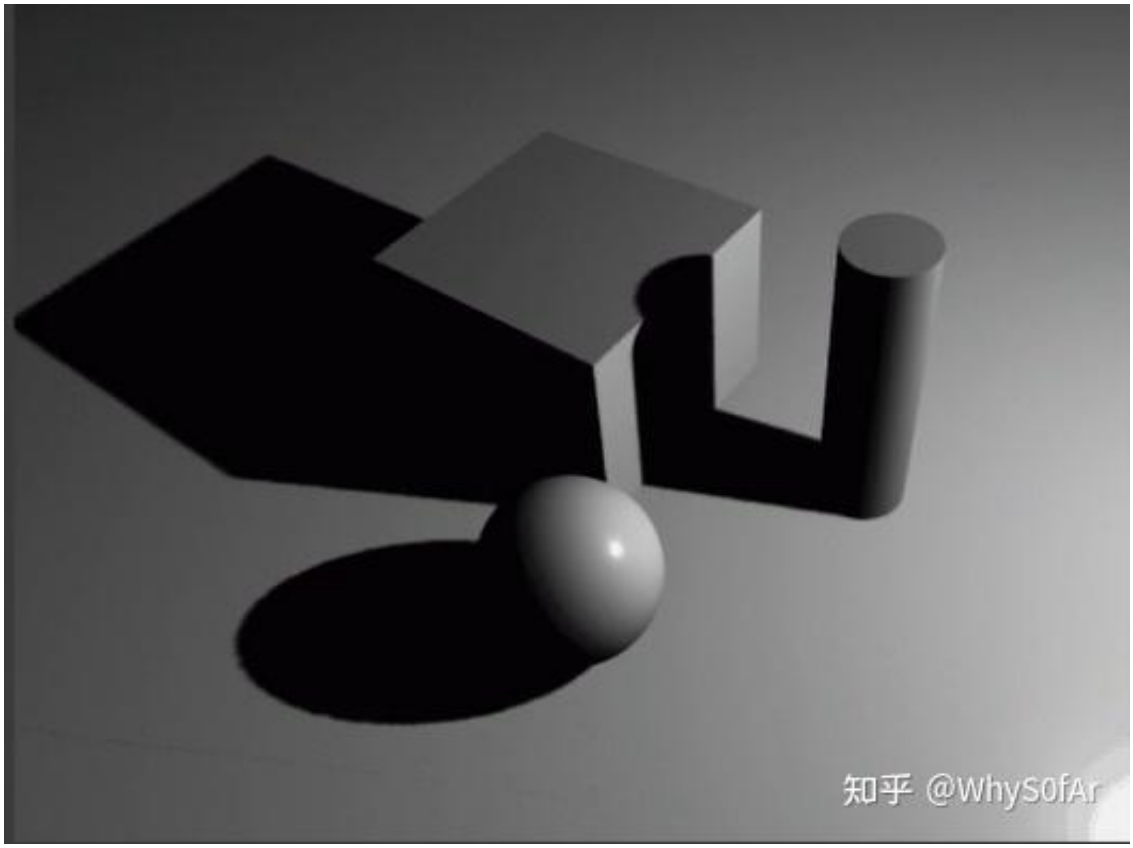
a) 我们要控制积分域足够小，也就是说我们只有一个点光源或者方向光源。

b) 我们要保证shading部分足够光滑，也就是说brdf的部分变化足够小，那么这个brdf部分是diffuse的。

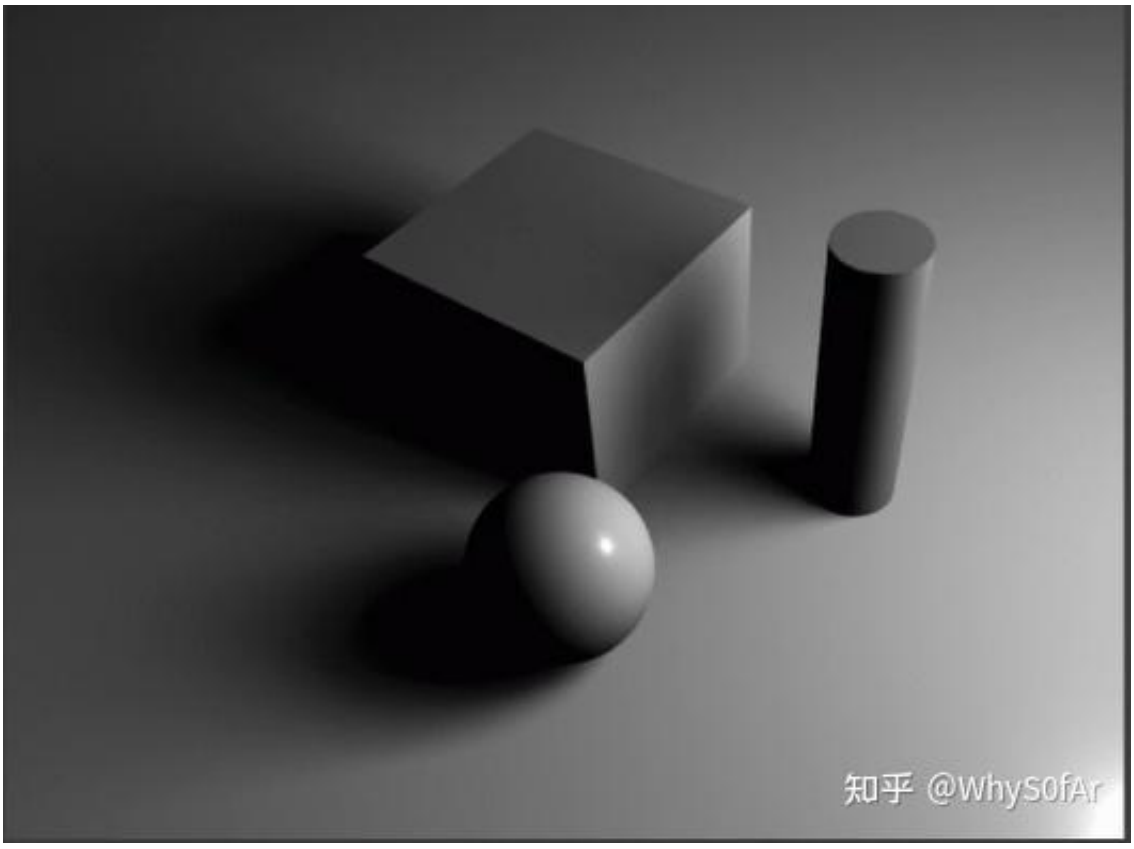
c) 我们还要保证光源各处的radience变化也不大，类似于一个面光源。

i i i - > PCF(Percentage Closer Filtering) 与 PCSS(Percentage Closer Soft Shadows)

首先我们来说一下软阴影和硬阴影:



硬阴影



软阴影

我们可以看出,软阴影的效果要更加真实也更自然,强于硬阴影的效果.(闫老师在课堂上甚至来了段RAP,233333333)

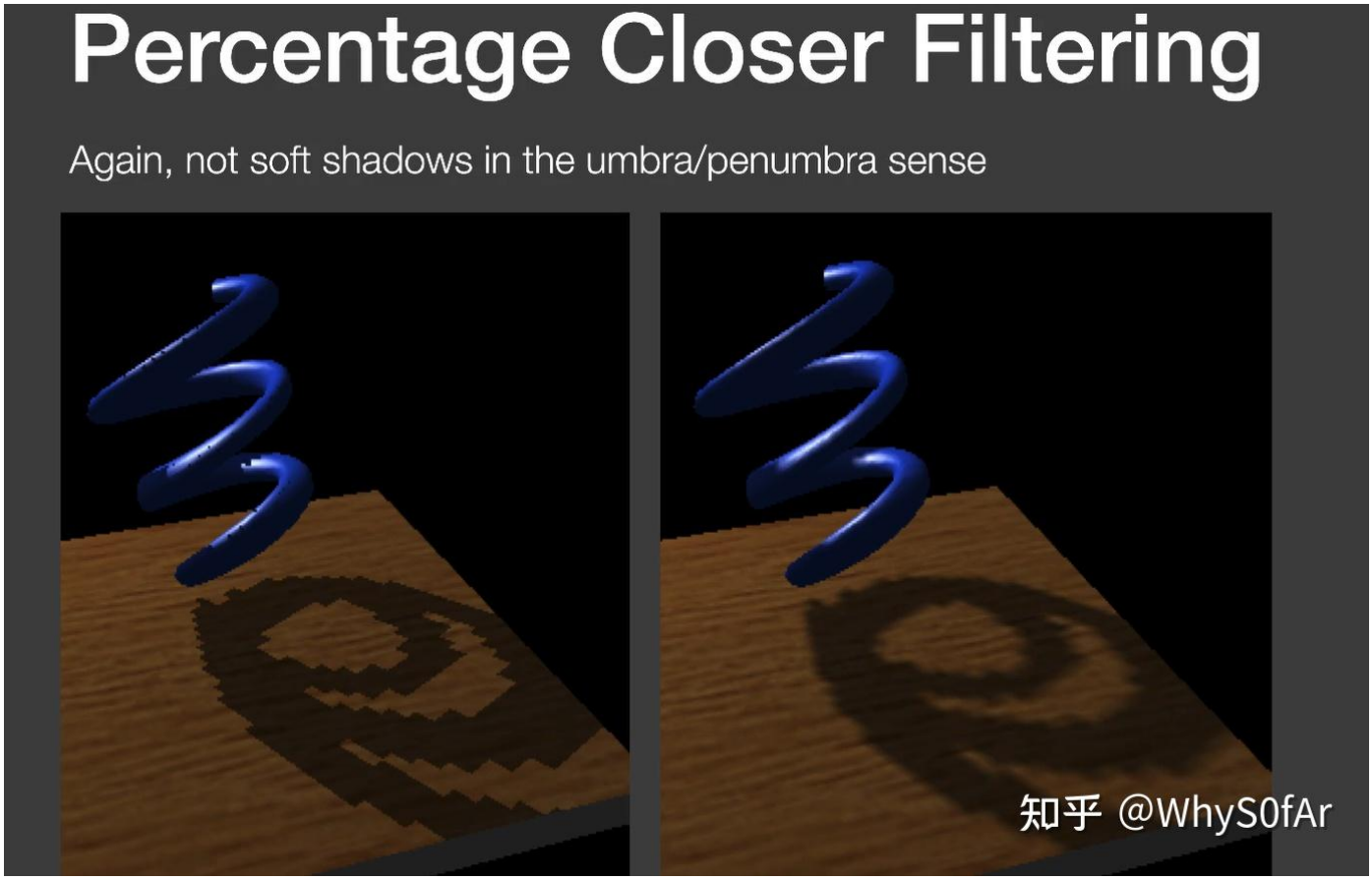
为了实现软阴影的效果,我们首先会用一个工具PCF------percentage closer filtering:

PCF的初衷是为了抗锯齿,反走样，比如上面的忍者阴影出现的锯齿状,是为了解决这个现象.

后来发现可以用在软阴影上，通过把shadow结果求一个加权平均(或者叫filtering).

首先我们要强调一点:

1.PCF不是直接在最后生成的结果上模糊,而是在你做阴影判断时进行filtering. 如忍者那张图,并不是在他的基础上去进行模糊处理.



在有锯齿的结果上进行filtering

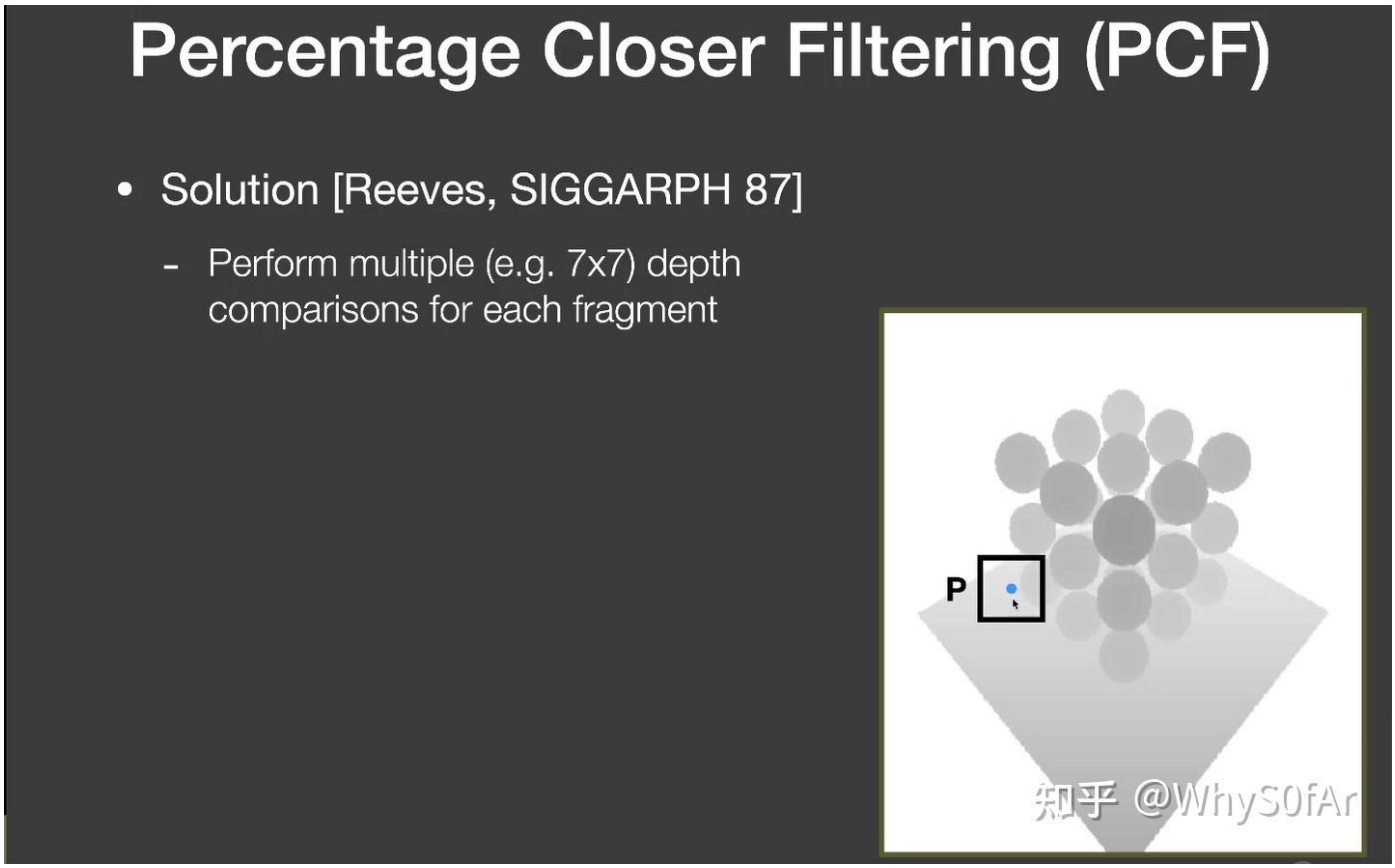
就跟我们在反走样时一样,我们不能先得到一个走样的结果再去做在这个走样的结果上进行模糊.

2.也不是Filter the shadow map,

如果直接在shadow map上filtering就会造成阴影和物体交界直接糊起来，而且在第二个pass上做深度测试还是非0即1的结果,最后得到的仍然是硬阴影。

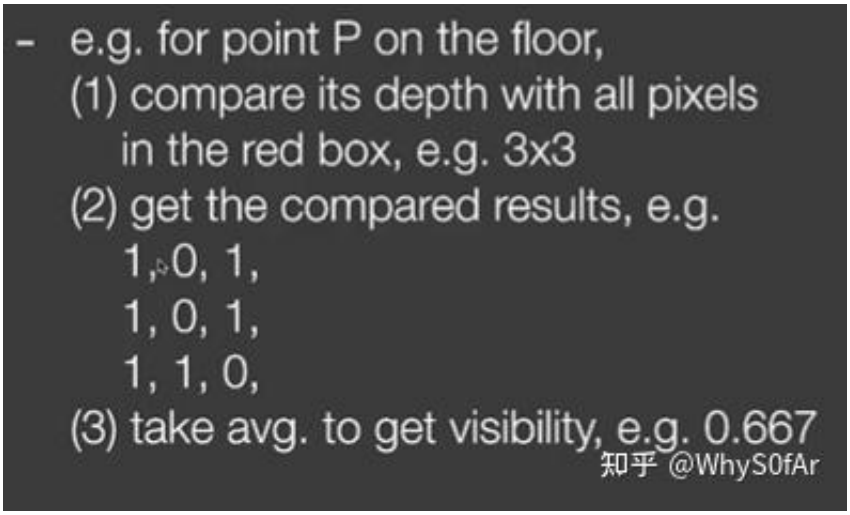
我们之前在做点是否在阴影中时,把shading point连向light然后跟Shadow map对应的这一点深度比较判断是否在阴影内,之前我们是做一次比较,这里的区别是,对于这个shading point我们仍要判断是否在阴影内,但是我们把其投影到light之后不

再只找其对应的单个像素,而是找其周围一圈的像素,把周围像素深度比较的结果加起来平均一下，就得到一个0-1之间的数，就得到了一个模糊的结果。



如图,蓝点是本来应该找的单个像素,现在我们对其周围3 * 3个像素的范围进行比较,由于是在Shadow map上,因此每个像素都代表一个深度,我们让在shadow map上范围内的每个像素都与shading point的实际深度进行一下比较,如果shadow map上范围内的像素深度小于shading point的实际深度,则输出1,否则输出0.

从而得到9个非0即1的值:



最终我们用得到的加权平均值0.667作为shading point的可见性。在计算阴影的时候我们就拿这个作系数来绘制阴影。

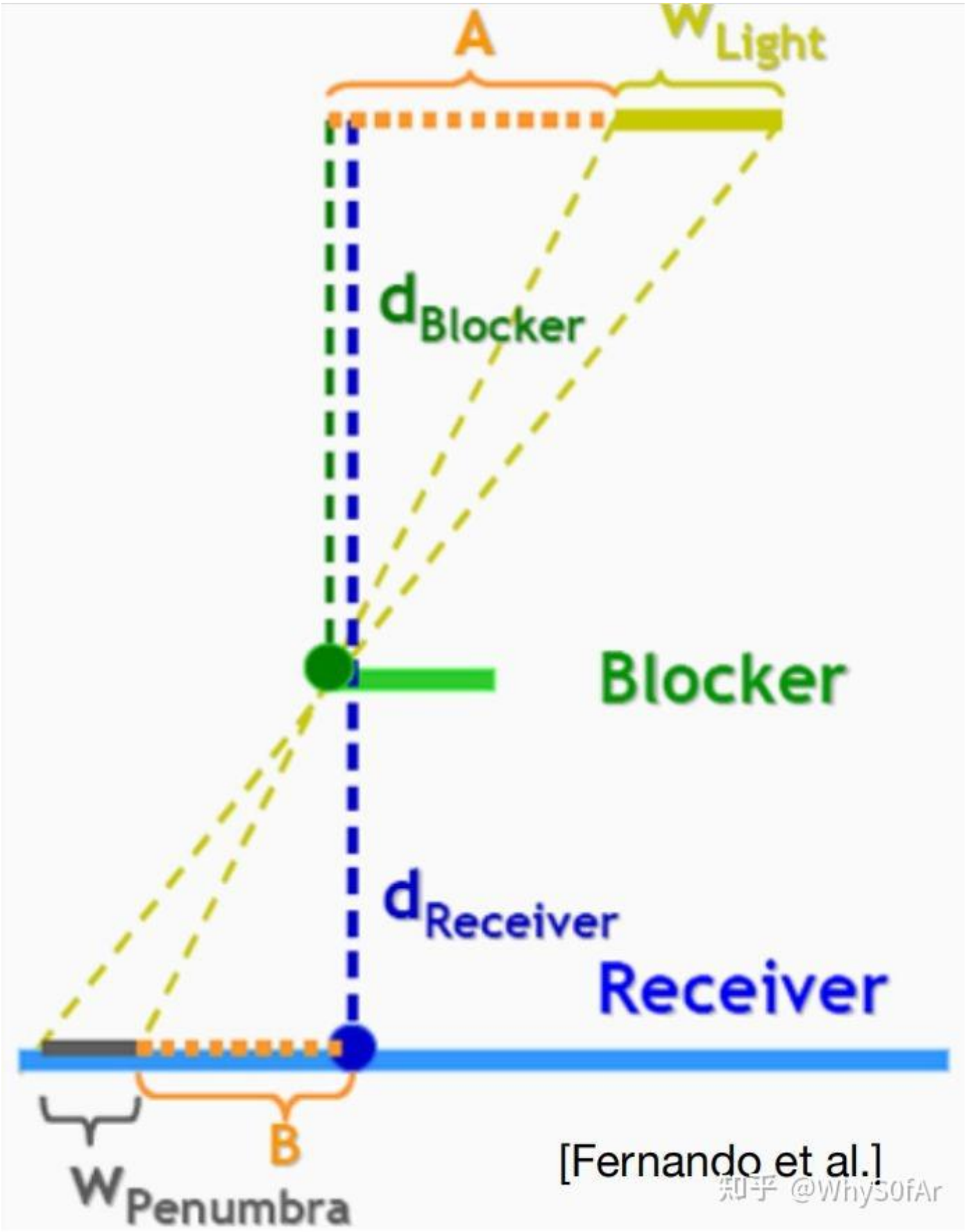
可以看到抗锯齿效果很好

我们会发现如果filter size越大，阴影本身越软，所以这个方法也就可以去绘制软阴影，也就是pcss技术。

在这幅图中我们可以看见,笔尖的阴影十分锐利(硬),因为我们可以认为

阴影接受物与阴影投射物的距离越小,阴影越锐利.

因此,我们要解决的一个问题是我们如何决定一个软阴影的半影区。换句话说，就是filter size 有多大的问题:



我们可以看到左下和右上两个黄色虚线形成的三角是两个相似三角形。

如果我们将blocker的位置移动一下,比如越靠近receiver,我们会发现（$W_{Penumbra}$）也就会越小.

用数学来表示半影区（$W_{Penumbra}$）：

$$w_{Penumbra} = (d_{Receiver} - d_{Blocker}) \cdot w_{Light} / d_{Blocker}$$

这里的 $d_{Recevier}$和 W_{Light}的大小我们是知道的，所以我们只需要拿到blocker的深度即可。

所以,在PPT中写道: **filster size < - > blocker distance**

但是这里又有一个问题了，如何确定一个blocker距离光源的位置?

不能直接使用shadow map中对应单个点的深度来代表blcoker距离,因为如果该点的深度与周围点的深度差距较大（遮挡物的表面陡峭或者对应点正好有一个孔洞），将会产生一个错误的效果,我们选择使用平均遮挡距离来代替，所以平常我们指的blocker depth其实是Average blocker depth.

blocker上的每个点距离光源的距离是不同的，深度也是不一样的。这里我们采用取平均深度的方式来表示blocker的深度。

求blocker距离的方法如下：

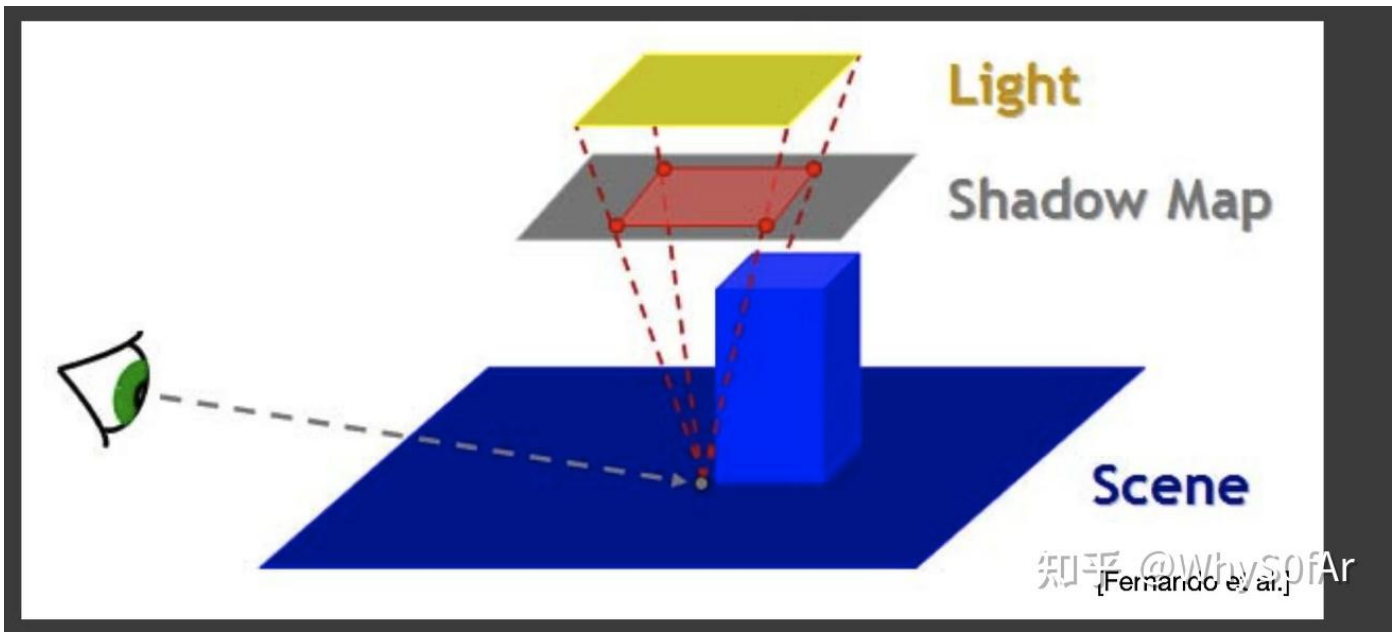
首先，我们把目标shading point 转换到light space 找到shading point在shadow map上对应的像素。

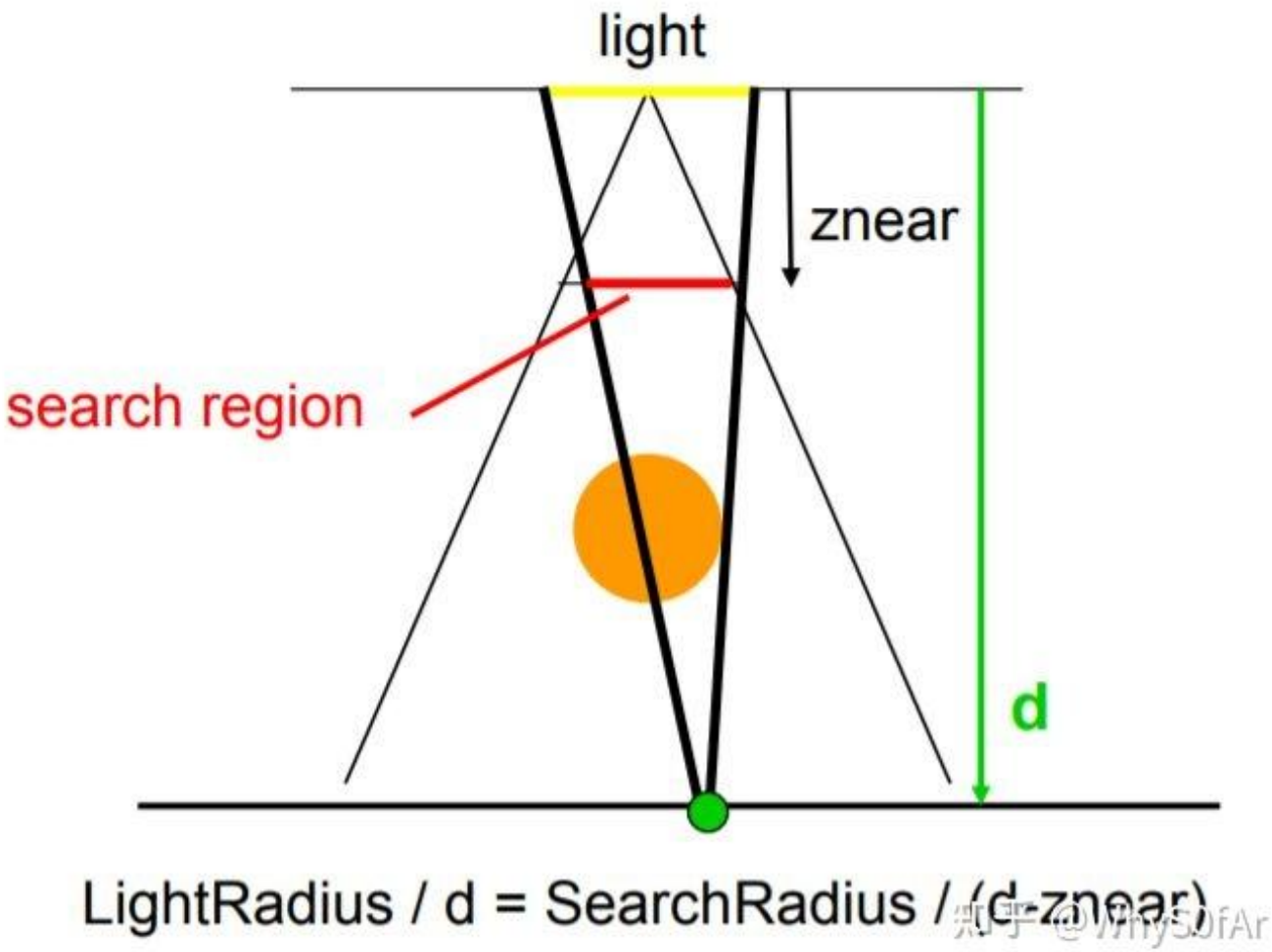
如果shading point的深度大于这个shadow map上点对应的深度,则说明shadow map上的点就是一个Blocker,然后我们取shadow map上这个点(像素)周围的一些像素，找出能够挡住shading point的点的像素,并求出他们的深度平均值作为blocker的深度。

这个方法还是有一个问题，这个问题就是我们虽然找出了filter size的大小。但是我们需要知道寻找blocker之一步骤中，我们需要找到周围的一些像素，那这个范围又是多少呢?一般我们有两种方法可以解决这个问题。

第一种，就是自己规定一个,比如 4 * 4, 16 * 16,比较简单但不实用.

第二种，是通过计算得到一个范围大小:





我们计算shadow map的时候在光源处设置过相机，如图所示，我们把shadow map放在由相机看向场景形成的视锥中的近截面上,然后将光源shading point相连，在shadow map上截出来的面就是要查询计算平均遮挡距离的部分.这部分的深度求一个均值，就是Blocker到光源的平均遮挡距离。

离光源越远，遮挡物也会更多,所以需要在Shadow map上的一个小区域内查找blocker.

离光源越近，遮挡物会少,所以需要在Shadow map上的一个大区域内查找blocker.

这样我们就得到了PCSS的三个步骤：

1. 寻找blocker，并计算平均深度。
2. 通过blocker 深度计算filter size。
3. 按照PCF方式绘制软阴影。

PCSS本质上就是求出了阴影中需要做PCF的半影部分后再进行PCF的计算，这样动态调节了半影范围，也就是动态设置了PCF的搜索范围，这样我们的硬阴影部分清晰，软阴影部分模糊，动态的实现了不错的软阴影效果。

说实话在写笔记过程中给自己写迷糊了,可能有些错误之处希望指出,在写完lecture 04之后我们再来看一遍视频对着改正一下,希望这篇笔记能给您带来帮助.