GAMES202 高质量实时渲染笔记Lecture11：Real-Time Physically-Based Materials (Surface models cont.)

本文是闫令琪教授所教授的GAMES202 高质量实时渲染笔记Lecture11：Real-Time Physically-Based Materials (Surface models cont.)

本人属于新手上路暂无驾照，有错误欢迎各位大佬指正.

本节内容:

Linearly Transformed Cosines(LTC->线性变换的余弦)

LTC是为了解决microfacet models的shading问题,但是它有一些限制:

1. *主要是针对GGX模型,对于其他模型原理也同样适用.*

2. *做的是不考虑shadow的shading*

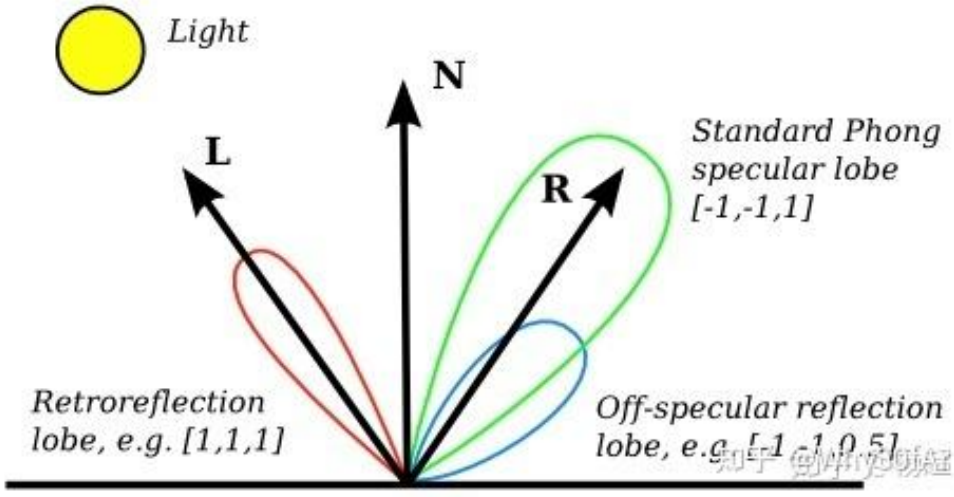
3. *光源是多边形光源,且发出的radiance时uniform的.*

LTC就是在多边形光源的照射下快速求出在ggx模型上不考虑shadow的任意一点的Shading.

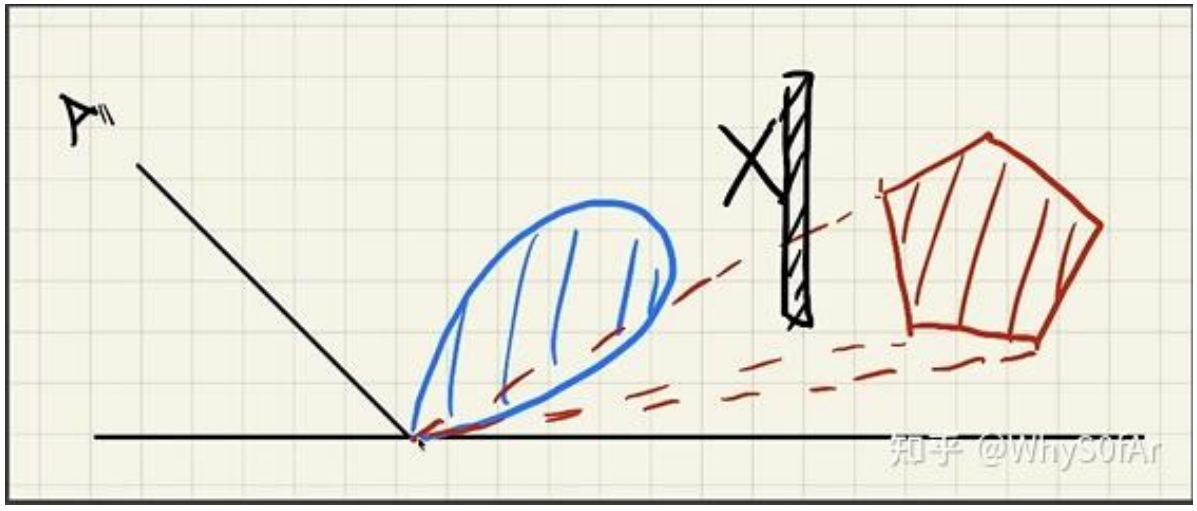
首先我们要解释一下什么是Lobe:

Lobe就是在固定一边方向下的brdf的函数图象,由于BRDF是一个4维的函数,在固定一边之后就变为了2维的函数.

大多数高光brdf，都是在半角向量接近宏表面法线时反射亮度最大,向上或向下时其值都会减小,使得其函数图象像叶片，故称为lobe。



如图在光源方向L固定的情况下，你移动反射方向R，当**L和R的半角向量=向量N** 时反射亮度最强，就是图中所示情况。如果继续向上或向下改变R，反射亮度都会减弱。就形成了那条绿色的极坐标曲线,也就是Lobe。



我们知道LTC是在多边形光源照射下求出微表面模型上一点的shading值

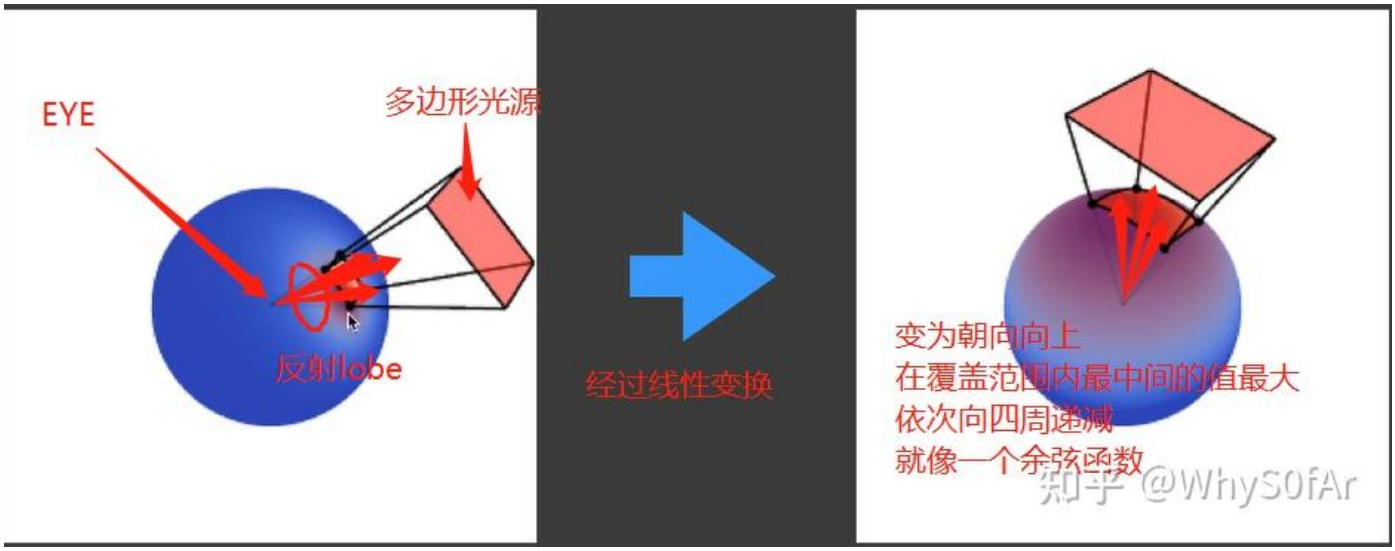
如果没有Ltc这个方法我们需要采样,在多边形光源上我们需要取很多采样点,并且与shaing point连线,如果不考虑连线是否与场景有交点则不考虑shadow,如果需要考虑shadow还需要做一下shadow test.但不论如何都需要采样,那么不可避免地会使速度变慢,因此为了不采样产生了LTC这个方法.

LTC和split sum本质上的区别是,split sum做的是环境光下的shading.

LTC核心方法:

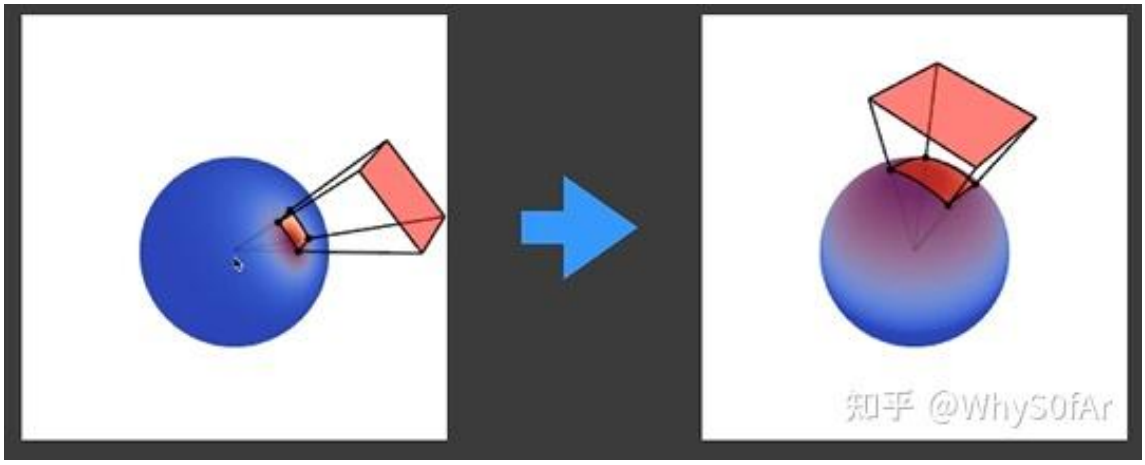
1.在固定入射方向后,我们将出射的lobe(brdf的lobe)转变为一个余弦函数

球面的所有方向 经过这个**线性变换** 后使得brdf的lobe朝向向上.



左边我们固定了入射方向,知道了反射方向的lobe,通过一个线性变换,我们将lobe内的所有朝向转变为右图中,其朝向向上的范围,并且它的覆盖是从正中间是最大值,依次向周围衰减,也就是像余弦函数一样.

2.在转换Brdf的lobe时,将多边形光源也进行变换



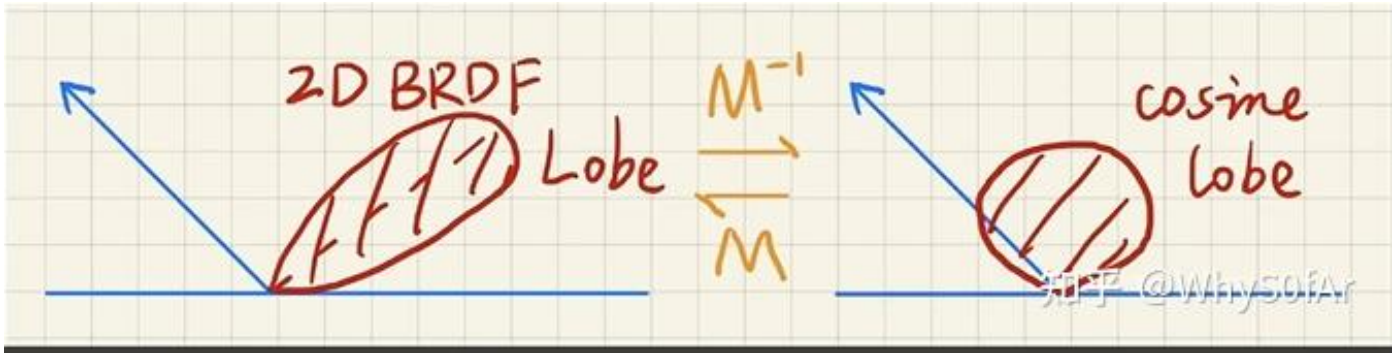
以图的四边形光源为例,将四边形光源的四个顶点与shading point相连得到四个方向(向量),让这四个向量也进行相同的线性变换,从而形成新的四边形光源,此时我们发现:

原光源与brdf结合去照亮shading point = 新光源与cos函数结合去照亮shading point

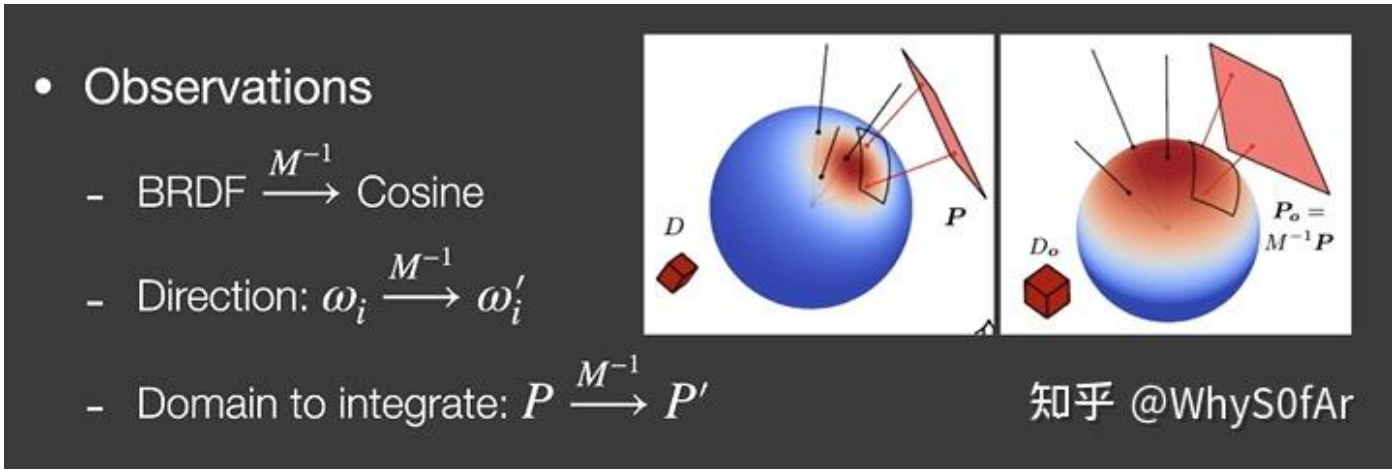
3.**我们将shading point点任意brdf的lobe在任意多边形光源下积分求shading的问题转变为在一个固定cos函数下对任意的多边形光源积分求shading**

因此我们要积分的东西只是cos,积分的范围虽然各不相同,但可以保证这个范围是一个多边形,而且在这个范围内积分是有解析解的,因此可以帮助我们快速求出shading.

思路:



原paper认为是任意一个cos lobe可以变换为brdf的lobe,因此brdf变为cos则需要的是逆矩阵.



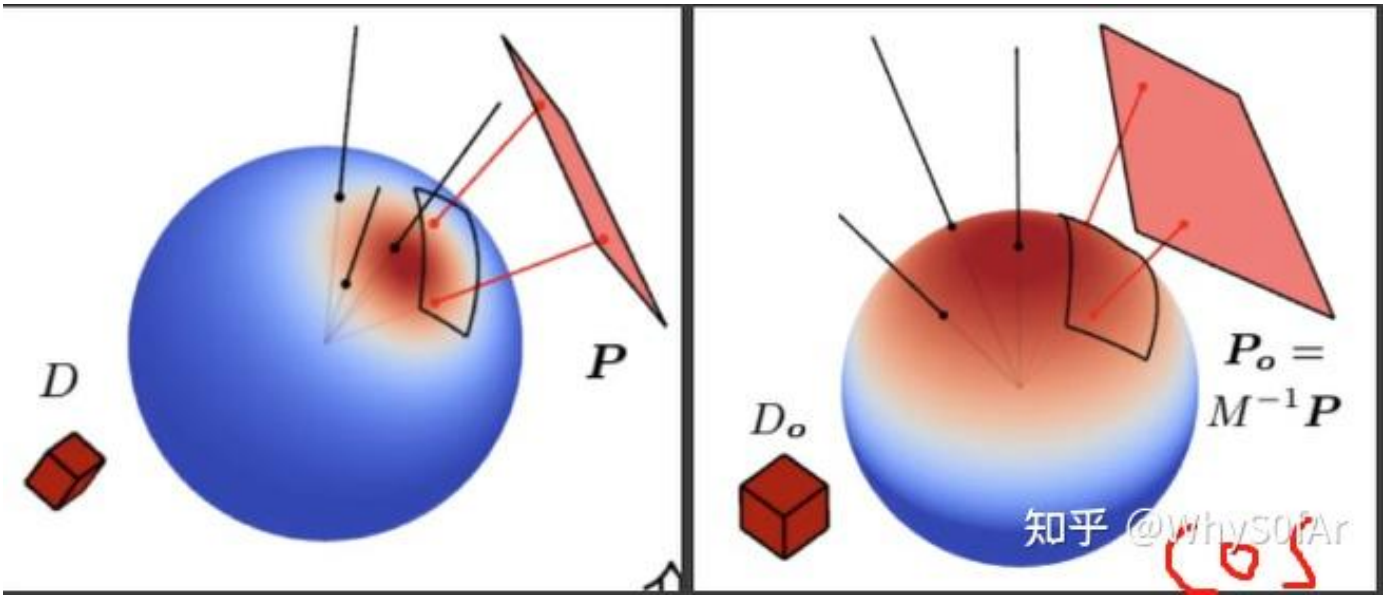
我们需要把

1.任意brdf的lobe变为固定的cos lobe

2.我们需要所有的方向ωi进行变换,变为新的ωi',从而使得brdf的lobe变为固定的cos.

3.原本的多边形光源所覆盖的方向也需要进行变换,从而产生新的多边形光源覆盖的区域,然后我们再新的区域内对cos进行积分求出shading值.

(个人理解,我本人数学知识很差所以这部分有点拿不准,如果有错误希望能大家指出来)

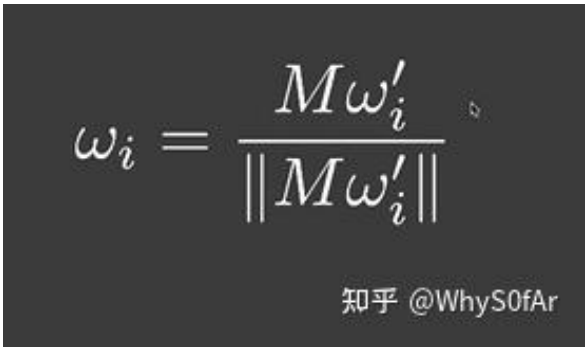


至此我们知道了整体思路,具体步骤我们需要做的就是改变变量:

我们从rendering equation开始:

$$L(\omega_o) = L_i \cdot \int_P F(\omega_i)\, \mathrm{d}\omega_i$$

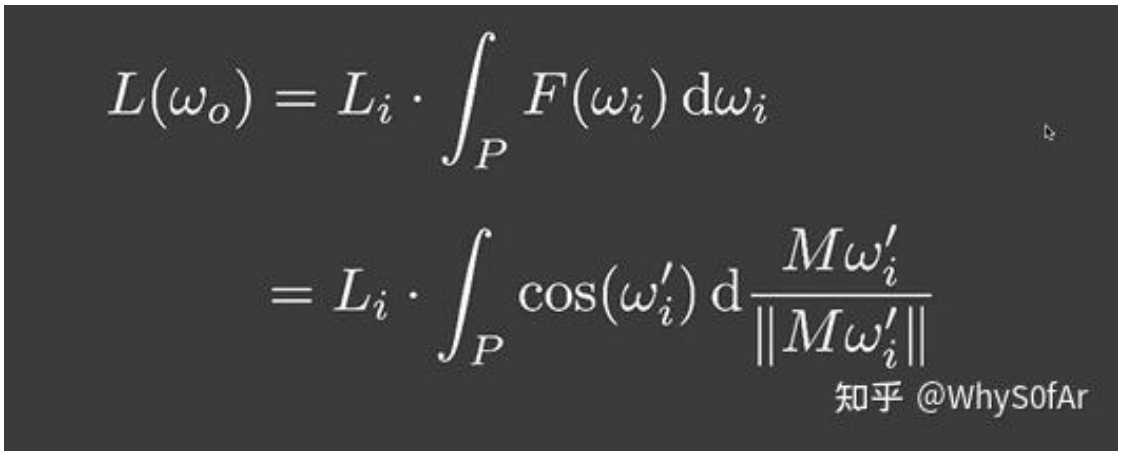
由于我们认为多边形光源内的任意radiance都是uinform的,因此lighting项可以拆出来,我们把剩下的brdf和cos合在一起机座F(ωi),我们需要把F(ωi)通过某种线性变换把所有的ωi变为新的方向ωi',从而使F(ωi)的形象变为cos.



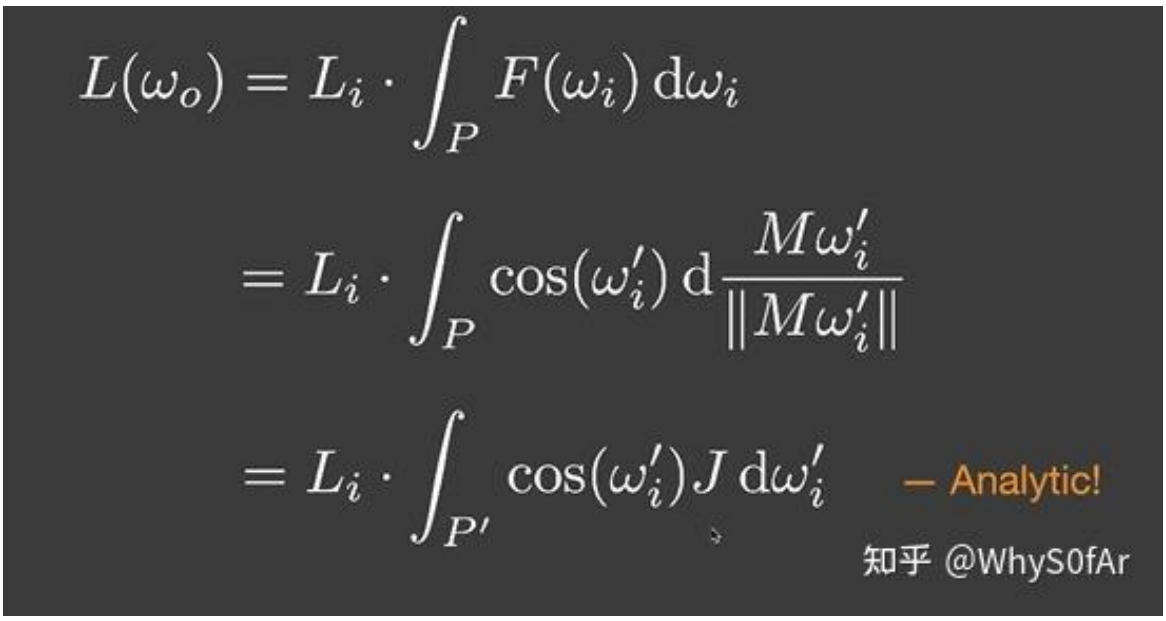
假设在单位球面上有一点我们经过了线性变换后,新的点可能不在单位球面上,也就是将一个方向经过单位变换后,它的长度可能会发生变化,因此如果像仍然让他是一个方向我们需要进行归一化处理.

由于我们是按照paper中来说的,paper中brdf是经过M的逆变换得到的,因此新的方向ωi'经过M才是ωi.

因此我们将其替换进去:



做完变量替换后我们会发现解不了积分,我们需要将 $d\frac{M\omega i'}{||M\omega i'||}$ 变为dωi',这部分是纯微积分,我们需要引入一个雅克布项-J.

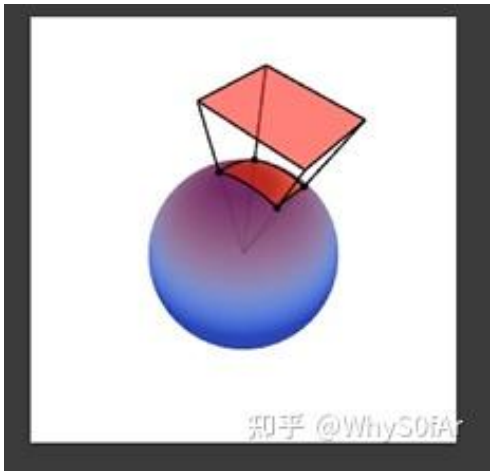


从而我们得到了在P'范围内对cos的一个积分,这个积分是有解析解的.这就是LTC方法的基本思路.

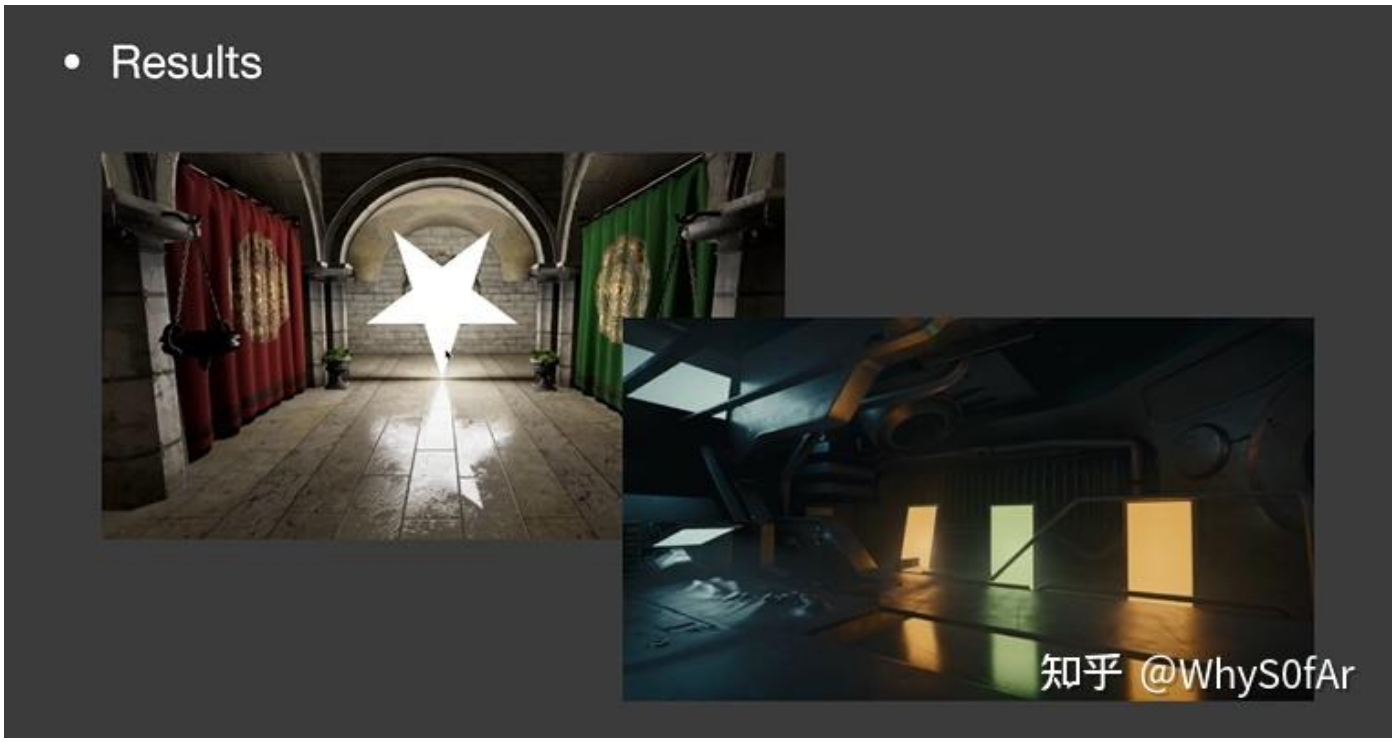
Q:cos不是一个一维的函数吗?

A:cos确实是一个一维的函数,我们把他认为是θ的函数而不是ϕ,但是我们还是可以把他以一个2D的形式显示在球面上,方便我们去理解.

$cos(\theta_s)$是一个球面分布函数（**余弦分布函数**），$f(p, w_i, w_o)$也是一个球面分布函数.因此其实我们是让一个球面分布函数通过线性变化变成了另一个球面分布函数.



我们可以看到LTC可以得到一个很好的结果:



由于LTC里面涉及的内容比较复杂,因此这里只讲了基本思路.

总结:LTC就是把 变化brdf 和 变化光源 通过线性变换 变成了在固定的brdf下变化的光源问题的一个方法.

Disney's principle BRDF

首先我们来讨论一下为什么还需要Disney's principle BRDF:

因为微表面模型是由一些问题的:

1.微表面模型的效果虽然很好,但是不能表示出所有的材质,比如真实材质就无法表示.

微表面模型大多都不diffuse.

我们来举个例子,我们有一个木头桌子,我们知道木头是diffuse的,在桌子的表面刷一层清漆.

清漆，又名凡立水，是由树脂为主要成膜物质再加上溶剂组成的涂料。由于涂料和涂膜都是透明的，因而也称透明涂料。涂在物体表面，干燥后形成光滑薄膜，显出物面原有的纹理。

从而我们的桌子变成了多层材质:清漆 + 木头.

清漆是无色的,由于清漆是平坦的,因此在光线打入时,一部分反射出去产生了高光现象.

另一部分打入内部并打到桌子上以diffuse发散出去,因此我们应该会看到高光和diffuse.

这是微表面模型无法做到的,因为微表面模型最多解释一层材质,而无法解释多层材质.

2.微表面模型对艺术家来说并不好用.

我们知道PBR,PBR,都是基于物理的,我们以金属反射率来说,金属反射率是一个复数 n - ik,这里要么理解成n - ik,要么理解成反射率由n和k这两个参数定义.

对于物理学家来说,他们知道如何去选择n和k的范围,但是对于artist来说,他们是不知道的,因此这是一个很苦恼的事,因为一旦定义不好就会出现严重的后果.因此PBR材质对于艺术家来说不好用.对于艺术家来说快速选择出材质去生成场景才是最重要的.

因此除了PBR材质,还有artist friendly材质,他的代表就是Disney's principle BRDF.

- Disney's principle BRDF诞生的首要目的就是为了让artist使用方便,因此它并不要求在物理上完全正确.
- 但是在RTR中我们认为Disney's principle BRDF也算是PBR材质.

OK,接下来我们来了解一下Disney's principle BRDF:

Disney's principle BRDF有几个重要的设计原则:

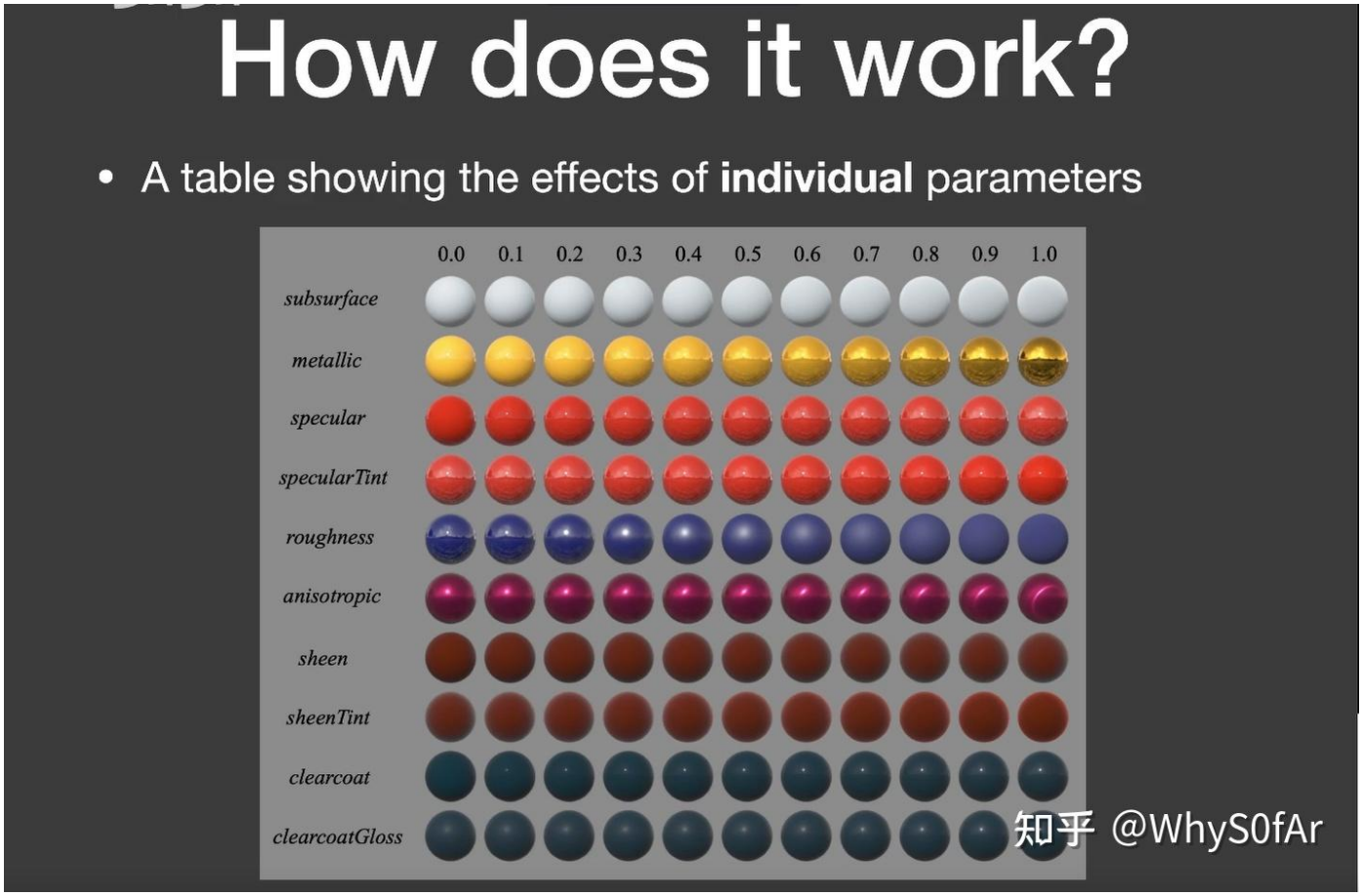
- The BRDF is designed with a few important principles
  - Intuitive rather than physical parameters should be used.
  - There should be as few parameters as possible.
  - Parameters should be zero to one over their plausible range.
  - Parameters should be allowed to be pushed beyond their plausible range where it makes sense.
  - All combinations of parameters should be as robust and plausible as possible.

知乎 @WhyS0fAr

1. 应该使用更直观的名词而不是使用物理名词参数,比如使用平缓,饱和度等
2. 让brdf框架不太复杂,也就是让参数数量少一点
3. 最好有一个拖动条左边最小值,右边最大值供艺术家们进行调整
4. 有时候为了特殊的效果允许将参数值超过范围,也就是允许小于0或大于1
5. 所有参数的组合应尽可能可靠和合理,也就是不论如何调整参数最后的结果应该是正常的.

我们不会将任何关于的公式,因为他很复杂,而且有开放的源码,如果有兴趣的可以去深入了解一下.

因此在这套设计原则下,artist可以根据自己需要去定义自己想得到的BRDF:

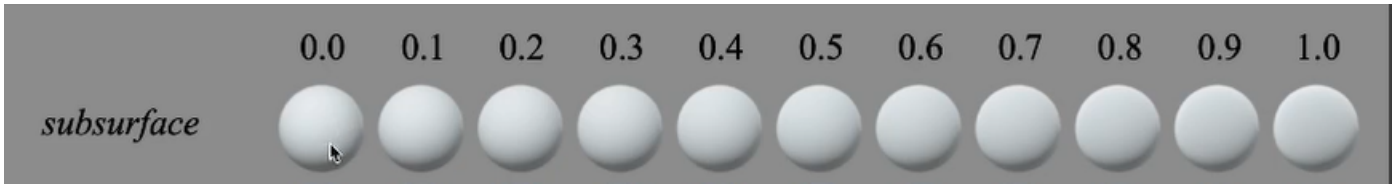


这张表格独立显示了各个参数在不同数值下的效果,在实际操作时这些参数是可以组合起来的,也就是叠加起来.

我们可以看到这些参数(属性)就不是基于物理的名词了,比如金属度(metallic),参数范围从0-1,0代表无金属,1代表就是金属.

因此我们将每个参数的定义给过一遍:

- subsurface:次表面反射,为了在BRDF中给你一种比diffuse还要平的效果.可以看出当subsurface为1时与0相比像是被压扁了一样.

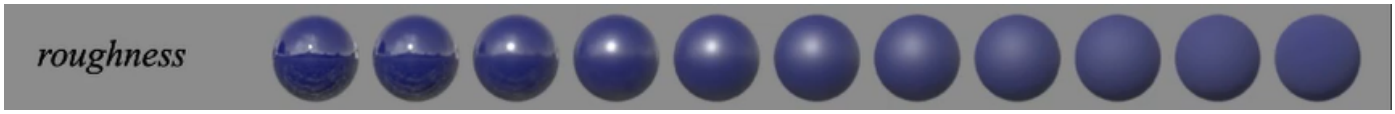


- metallic:金属性,顾名思义看起来像金属的程度.

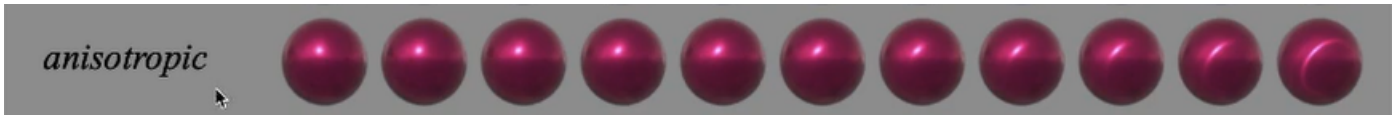
- specular:控制有多少镜面反射的内容,0为完全没有镜面反射内容,diffuse,1则表示全是镜面反射内容.

- specular tint:镜面反射出的颜色无色(为0),还是偏向于自己物体本身的颜色(1).

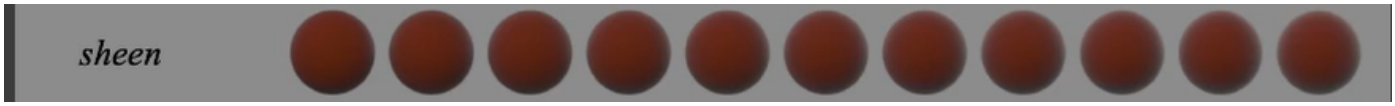
- roughness:粗糙度,为0表示全是镜面反射,为1表示没有镜面反射.



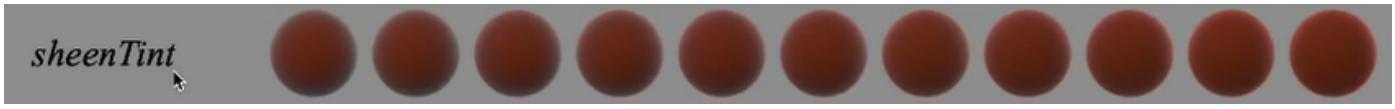
- anisotropic:各向异性程度,可以理解为当为1的时候带来一种像是被刷过一样的效果.



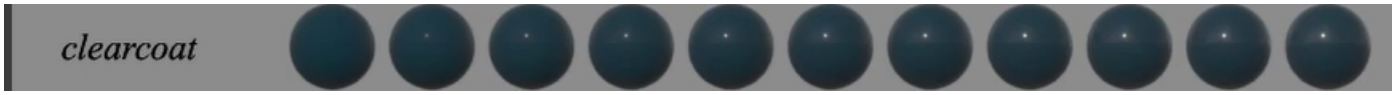
- sheen:可以理解为,在物体表面法线方向上长了绒毛,这让你在grazing angle(外圈)处看起来有一种雾化的感觉.



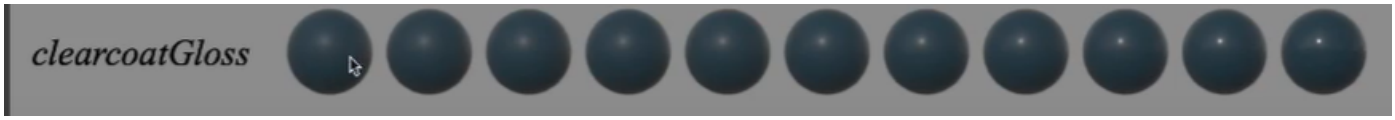
- sheen tint:可以理解为绒毛造成的雾化效果颜色是无色,还是偏向物体本身的颜色.



- clearcoat:可以理解为透明层的明显程度,0时表示没有透明层,1则表示有一层透明层(涂了一层清漆).



- clearcoat gloss:透明层的光泽层度,为0就像被磨砂了一样,为1则表示完全光滑.



所有的属性可以混合在一起使用.

Disney's principle BRDF的优点:

1.容易理解和使用各参数(属性)

2.参数的混合组合使得可以在一个模型上显示出很多不同的材质.

3.开源

Disney's principle BRDF的缺点:

1.并不是完全基于物理的

2.巨大的参数空间使得拥有强大的表示能力,但是会造成冗余现象.

个人觉得这部分只是了解一下即可,知道有这么个设计理念.

Non-Photorealistic Rendering(NPR)

首先我们需要知道NPR做的是什么,我们知道游戏的画风是各不相同的,有些游戏比如COD,战地等他们追求的时画面的真实从而让玩家有良好的代入感,而有些游戏比如原神,塞尔达的画风明显是与真实不相干的卡通风,也就是风格化(stylization)

NPR的任务就是为了把渲染出的结果偏向于这种卡通的风格.

闫神认为,在RTR中的NPR需要是一个**快速** 且 **可靠的** 风格化操作.

因此NPR通常是一些轻量级的处理,一般是在shader中做一些简单但是很聪明的处理从而完成风格化.

Photorealistic Rendering

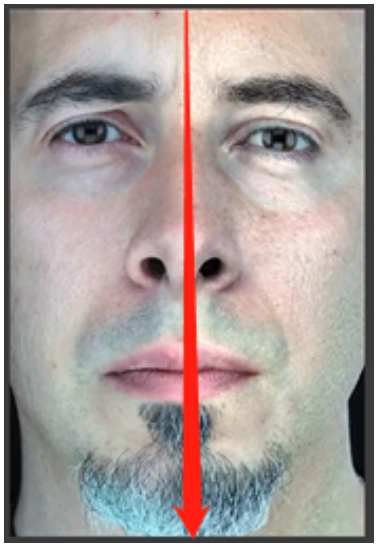
那么我首先要从Photorealistic Rendering开始了解

Photorealistic Rendering强调画面的真实感,因此他的目的很明确:

1.无法分辨是照片还是渲染出来的

2.光照,阴影,材质等效果需要无限接近于真实的.

渲染出的结果



一边是渲染的结果,一边是照片

Non-Photorealistic Rendering

NPR的目的是为了制造一种artistic的效果,从而使得渲染结果虽然远离真实感,但是有自己独特的风格和特点,且能够能清晰了解图想表达的是什么.

如图非常不真实,但是通过将区域变暗我们可以很清楚的知道暗下去的区域是阴影.

NPR思路:

1.我们要先得到真实渲染的结果

2.通过观察,把Photorealistic变为Non-Photorealistic

NPR的一些应用:

1.艺术

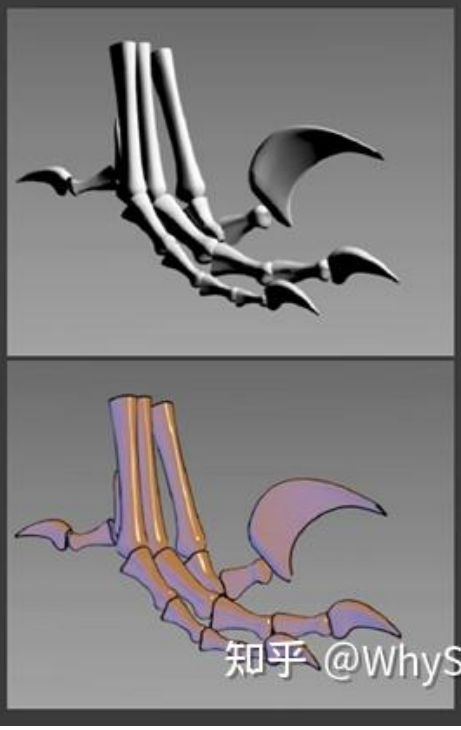
Applications of NPR

- Art

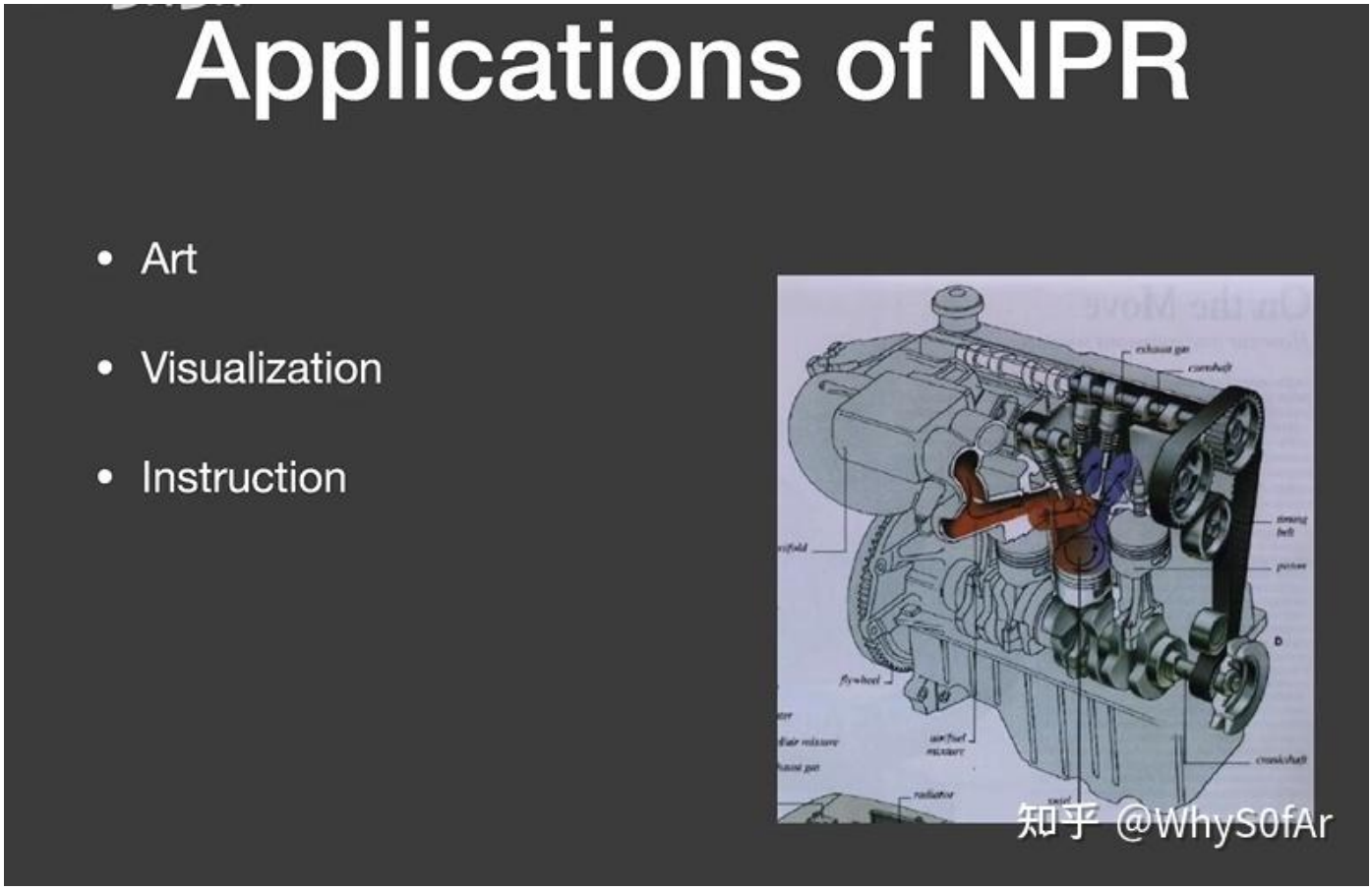
2.可视化

Applications of NPR

- Art
- Visualization
- Instruction



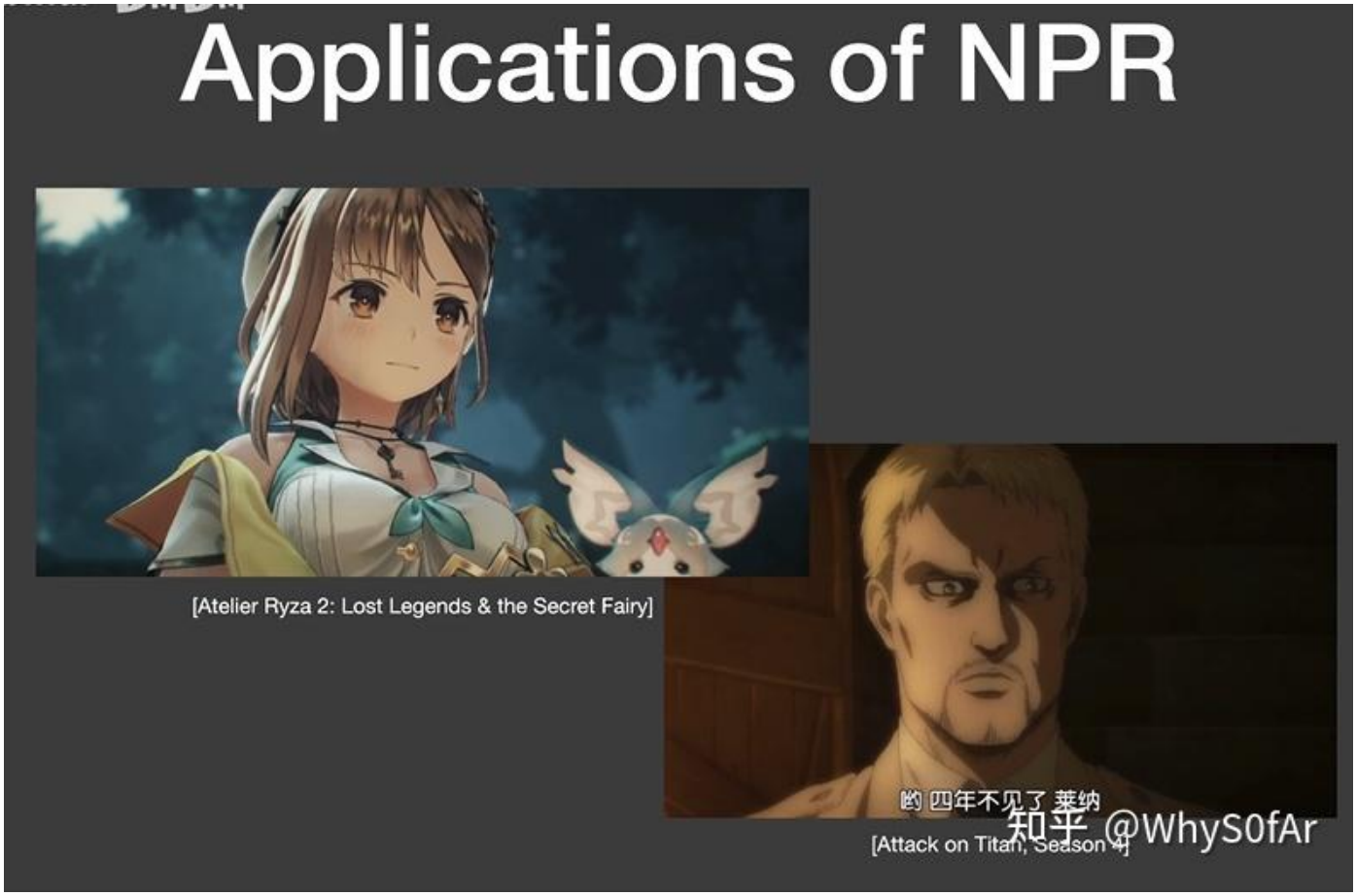
3.指示,示意图,在这里NPR可以比PR更直观的指出问题或者关键.



4.教育

5.娱乐

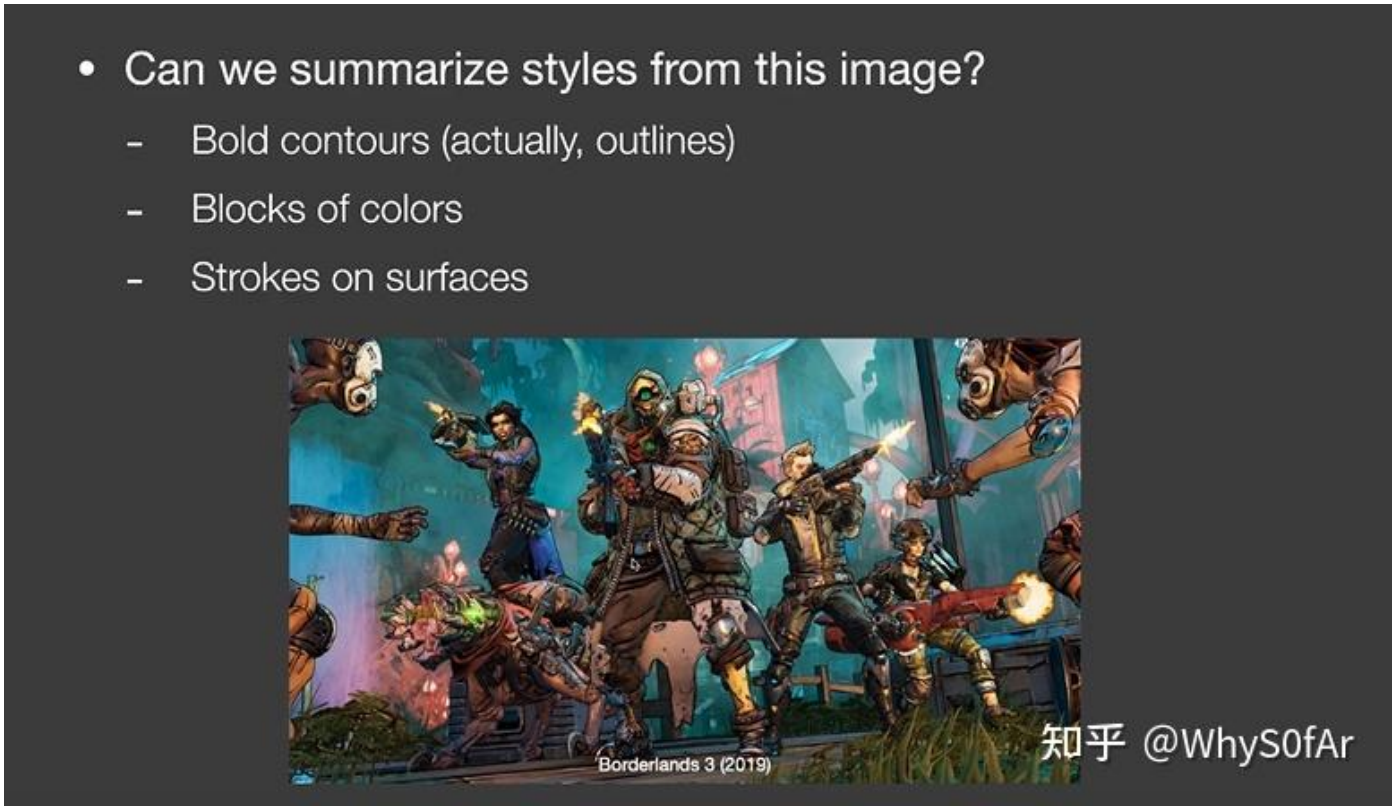
NPR在游戏领域运用的其实是很广泛的,比如之前提到的原神,塞尔达,以及大佬图中的莱莎的炼金工坊2(被种草了回头得去玩玩)



我们可以看到在大腿....啊不炼金工坊中,人物是3D的,光照,着色,阴影等都有,只是应用的不是真实感渲染而是非真实感.

同样的NPR在动漫领域,在经历过纯手绘的时期之后,现在的几乎所有动漫都是在通过渲染+部分人工工作这么一种协同来得到的.

我们可以看到图中莱纳的脸上有明显的由左侧点光源照亮而得到的阴影,也就是拥有全局光照,至此我们可以知道NPR保留了很多photorealistic的东西.

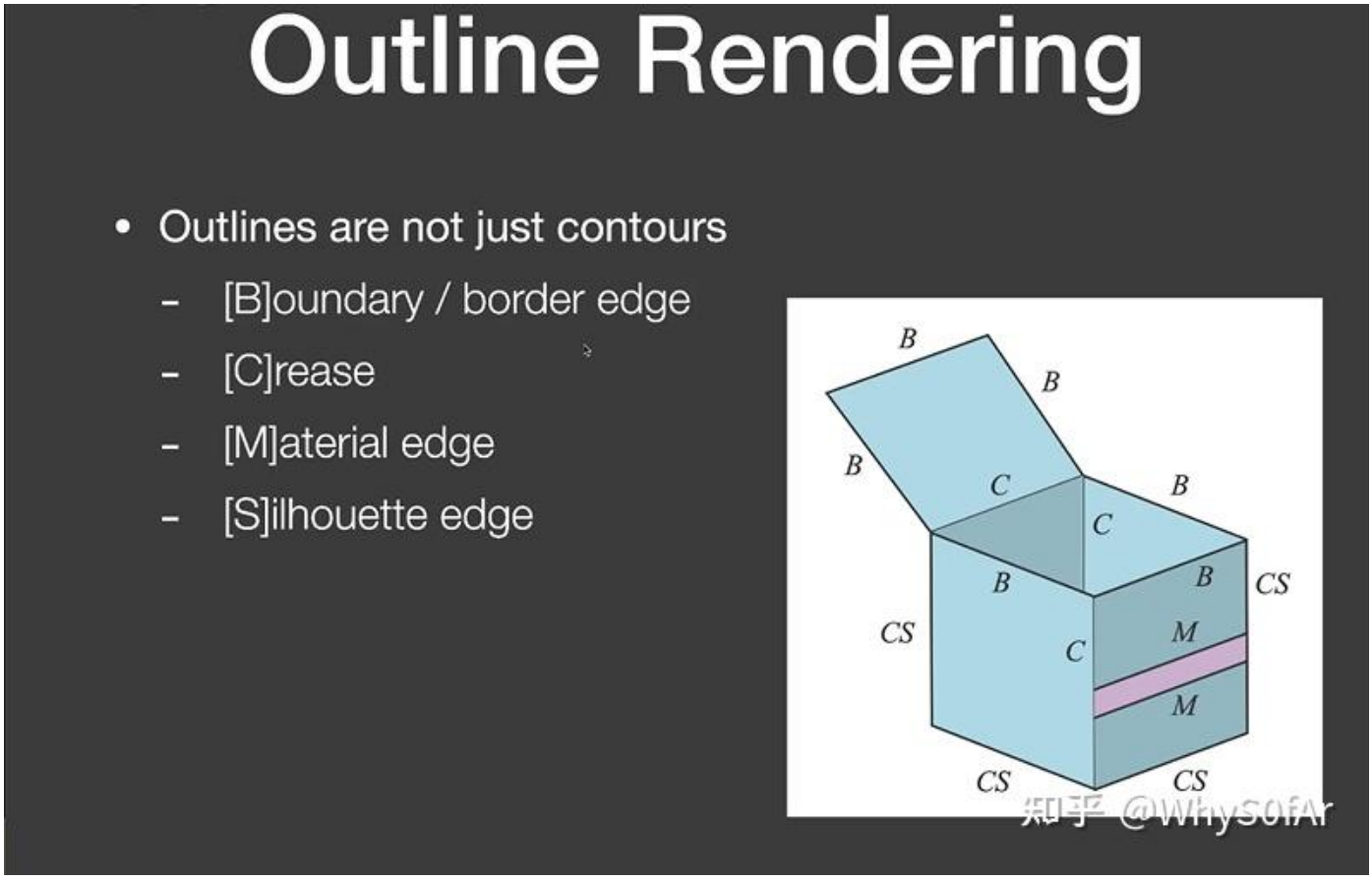


以无主之地系列为例,它具有很明显的美漫风格,以他们的身体旁的边缘轮框描边为例,我们想得到这个效果,在实际中需要思考这些边缘是什么,在NPR中如何找到这些边缘并且加粗从而产生一个轮廓描边的效果,不只是轮廓描边,在NPR中做任何一种效果都应该这么去分析,分析的好坏决定了结果.

outline rendering(轮廓渲染):

首先我们要说一下,平常我们说的边缘指的是contours:整个人物轮廓外围的一圈.

Outline的概念范围很大,contours是outline的子集.



如图,将一个盒子细分为各种类型的"边缘":

- 1Boundary/border edge->物体外边界上
- 2.Crease->折痕,通常是在两个表面之间的
- 3.material->材质边界
- 4.Sihouette edge->必须是在物体外边界上且由多个面共享的.

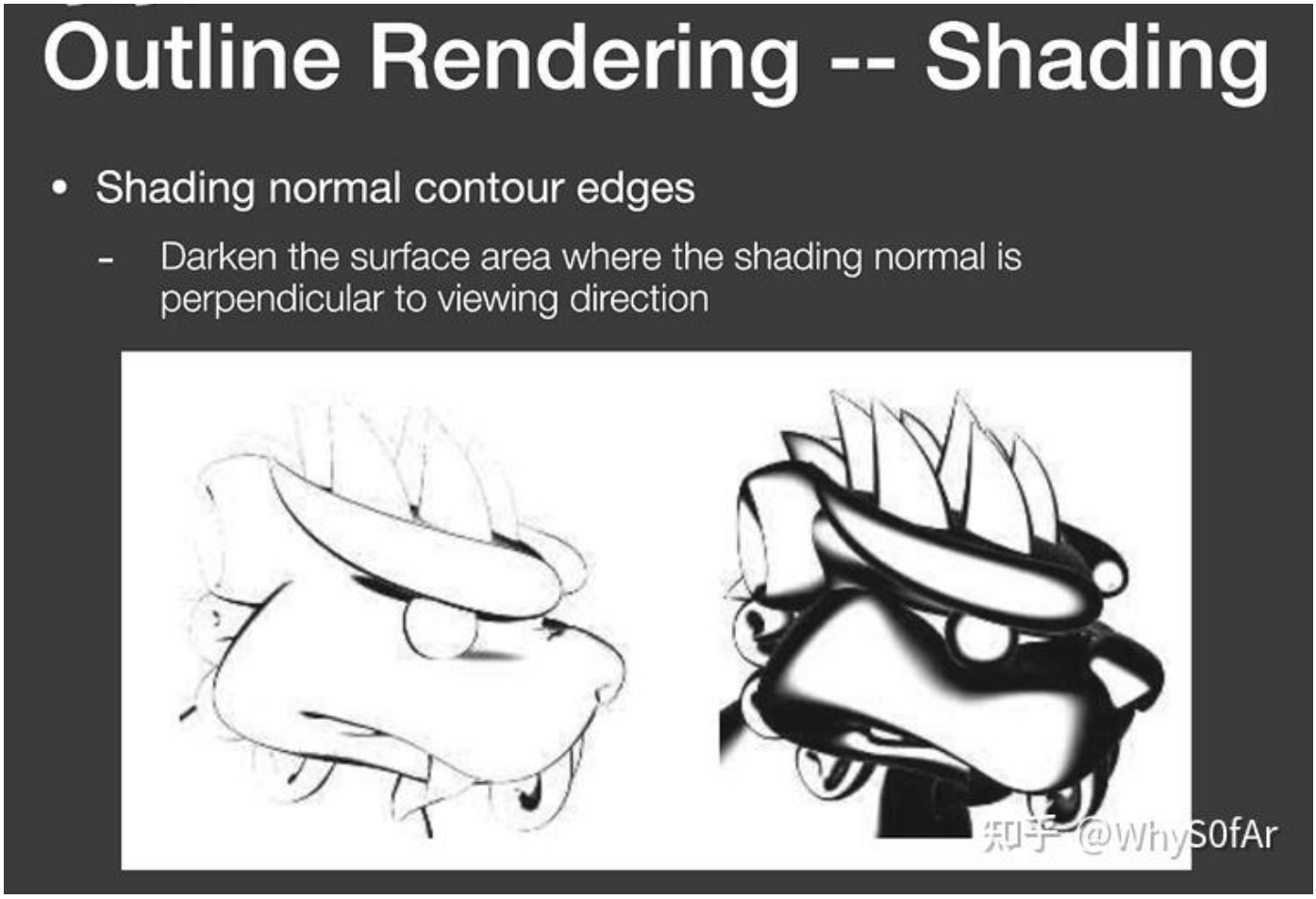
到此我们知道了边界的各种类型,接下来我们进行描边操作:

描边有两种做法:

- shading
- 后期处理

基于法线着色:

大体意思是在Pixel Shader中对观察方向向量和着色表面法线向量点乘,如果得到的值接近于0,说明着色区域是contour edges.



这里我们做的是Sihouette edge

正常情况下我们认为整个物件的不同物件是各自封闭的

以图为例,我们如何去判断哪些像素点在Sihouette edge上让我们去进行描边这个操作呢?

首先我们要先找到Sihouette edge,我们看向物体grazing angle的地方就是Sihouette edge所在的地方.

因此在渲染时我们只需要做一个判断,判断观察方向与 Shading point的法线的夹角是否接近90度,我们可以设定一个范围来认为是否在Sihouette edge上

比如超过89度小于90度,我们就认为这个shading point在Sihouette edge上,我们需要把这一点的颜色变为黑色从而达到描边效果.

我们可以更改设定的范围从而改变容忍度,比如超过60度的点都被认为在Sihouette edge上,那么容忍度就提高了,可想而知将会得到更多的黑色部分,也就是得到的效果变得更粗.

但是这样做有一个问题,不同的位置描述出来的边的粗细可能不一样,因为我们通过判断角度来进行描边操作的,无法固定粗细,因此法线变化平缓的地方就必须粗,法线变化尖锐的地方就会比较细.故而我们得到的效果可能不是很好

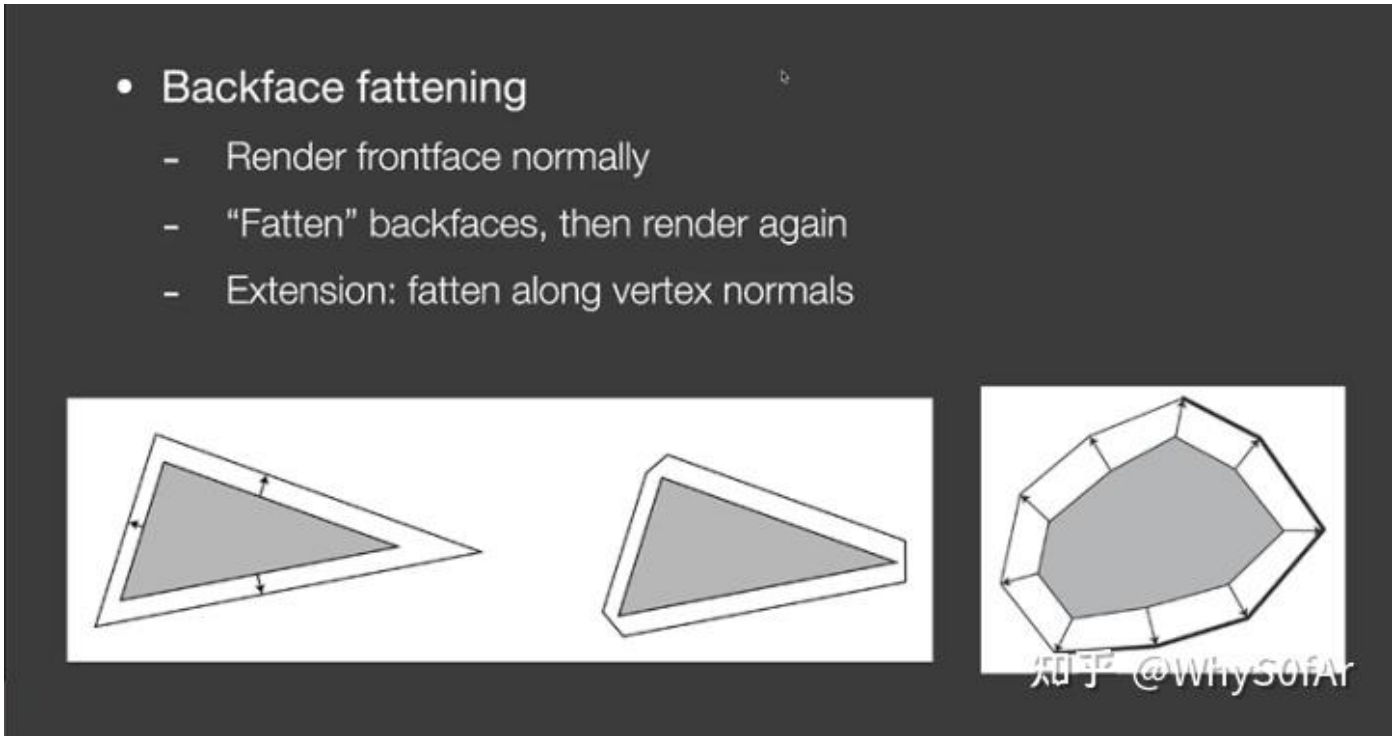
还有就是在一些类似正方体的模型上会失败,因为正方体的模型边缘是crease edges,这种方式无法识别. 在一些法线分布不均匀的模型上,也会产生奇怪的效果.

基于几何 :

我们这么去想,我们想得到的不就是一个轮廓分明点的描边效果嘛,那么我们不去区分什么Sihouette edge,Boundary/border edge之类的

当我们去渲染一个物体/模型时候,如果我们把模型加大一圈,然后大模型放在原模型背后并且把新模型完全渲染成黑,就像你背后站了一个就比宽一点高一点的替身使者,但这个替身使者是全黑的,这不就是描边了吗.

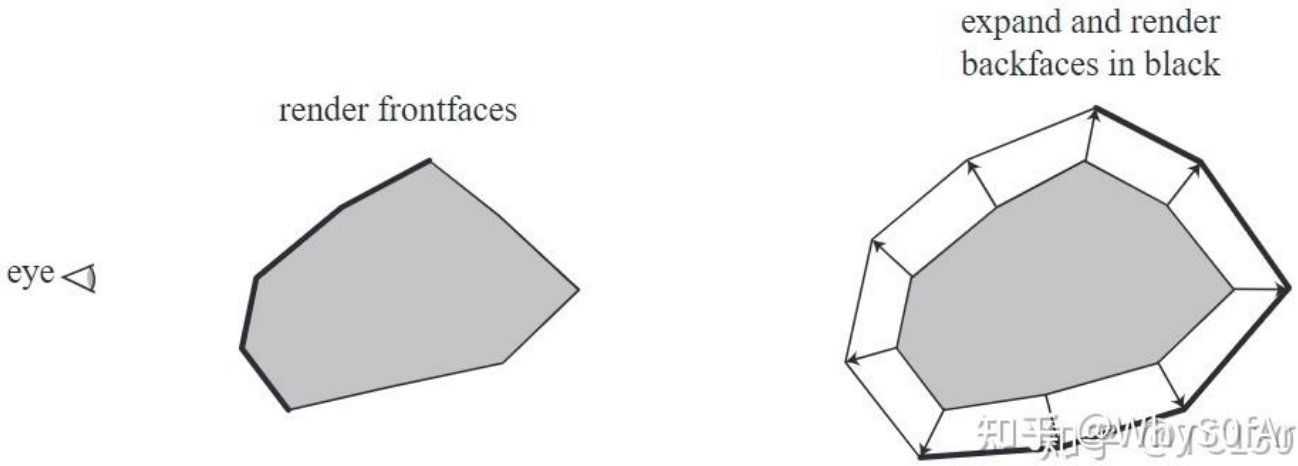
这是一个想法,实际操作中有更聪明的实现方法.



一个封闭模型分为看得见的面为正面,看不见的面为背面

我们刚才的思路不是将整个模型放大一圈放在身后嘛,那样太麻烦

封闭模型正面和背面所能看到的区域是完全一样的,也就是看不到模型背面的任何信息,因此我们将模型背面给扩大一圈,也就是把背面的每个三角形扩大了一圈,并渲染成黑色,从而得到描边的效果.



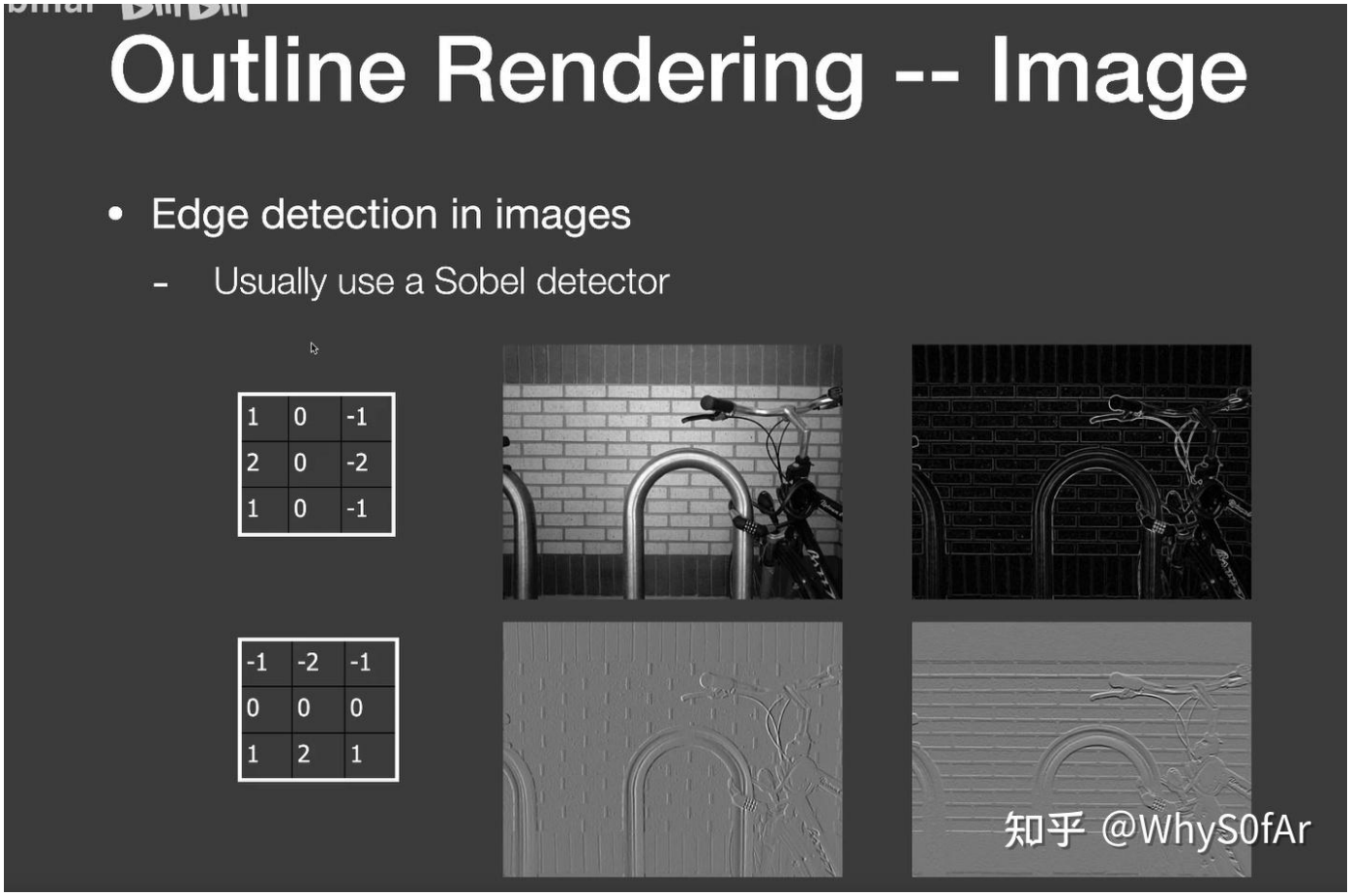
但是主流的方法是将我们将背面的每个顶点沿着顶点的法线方向向外偏移,从而达到背面外拓

这种实现方式简单稳定,不需要获取三角形的邻接信息,不过只能显示contour edge.

因为是需要将整个背部的面都渲染,所以这种方式可能造成一些资源浪费.对于半透明物体的渲染描边,也会产生影响.

基于图像的后期处理:

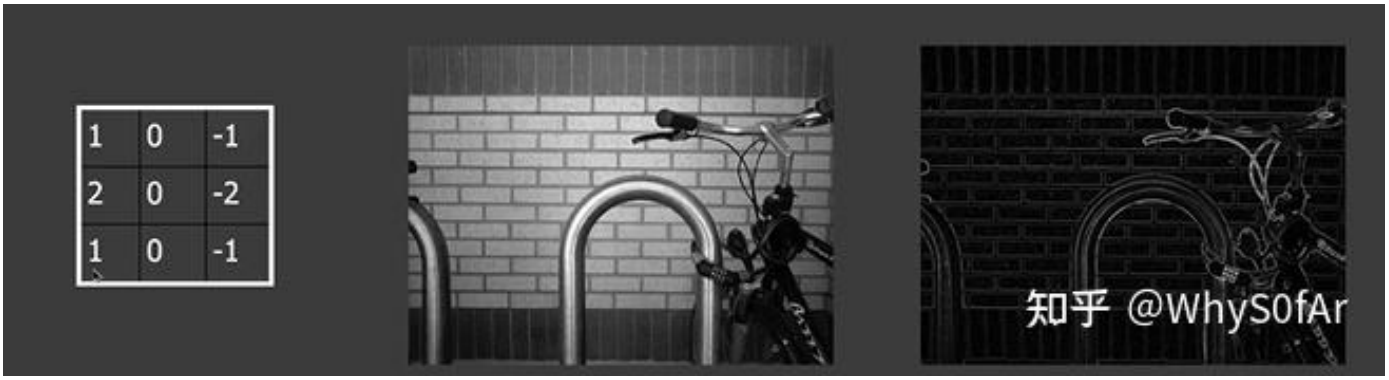
我们先生成未描边的正常渲染结果,再通过一些图像处理的方法从而得到这些边并且标上颜色.



我们以Sobel detector为例,他的做法是做了图像的filter(卷积).

我们知道卷积是在一个像素,取周围一圈的值做一个加权平均,因此我们可以自己设计filter kernel得到我们想要的边.

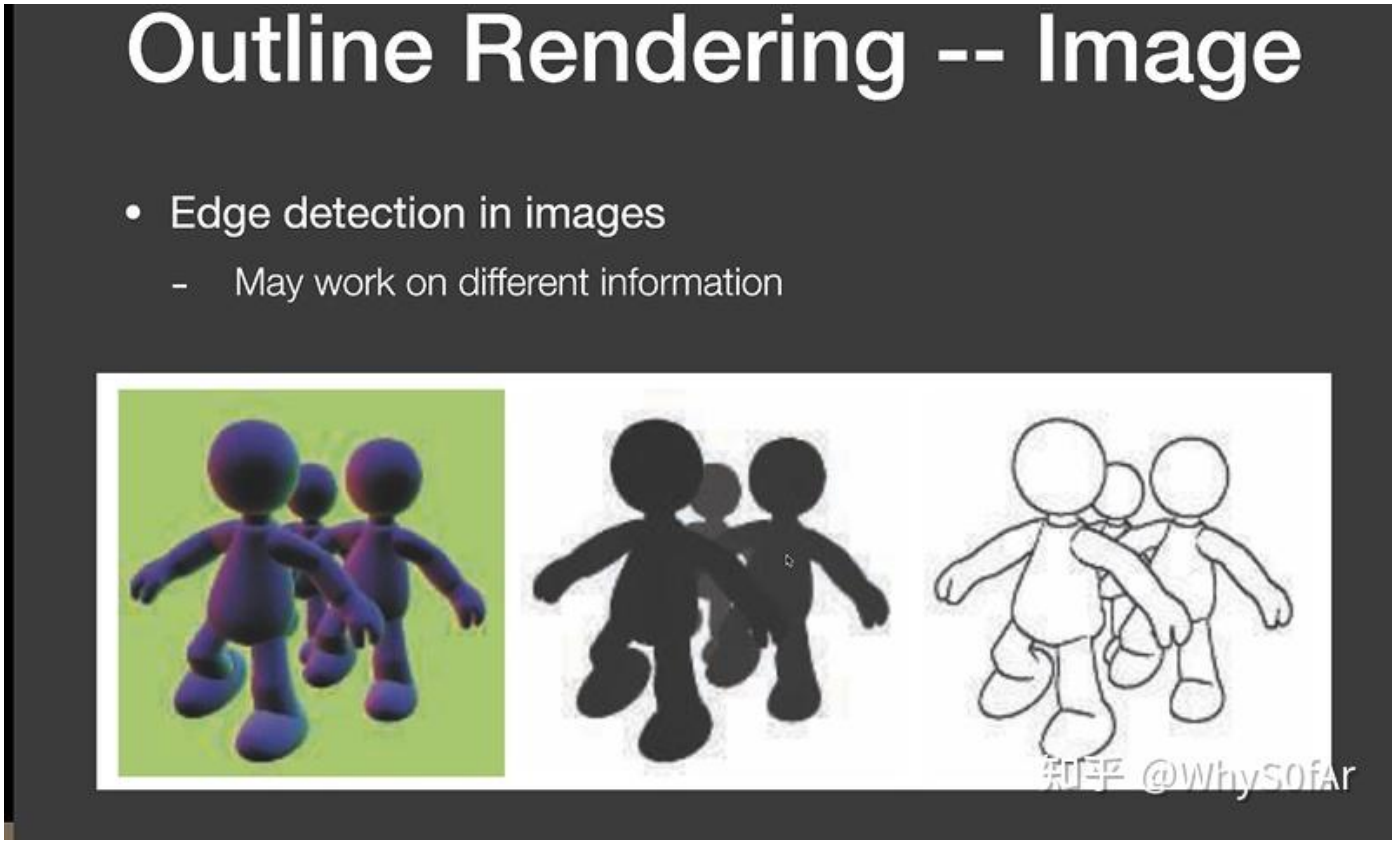
我们以left sobel为例:

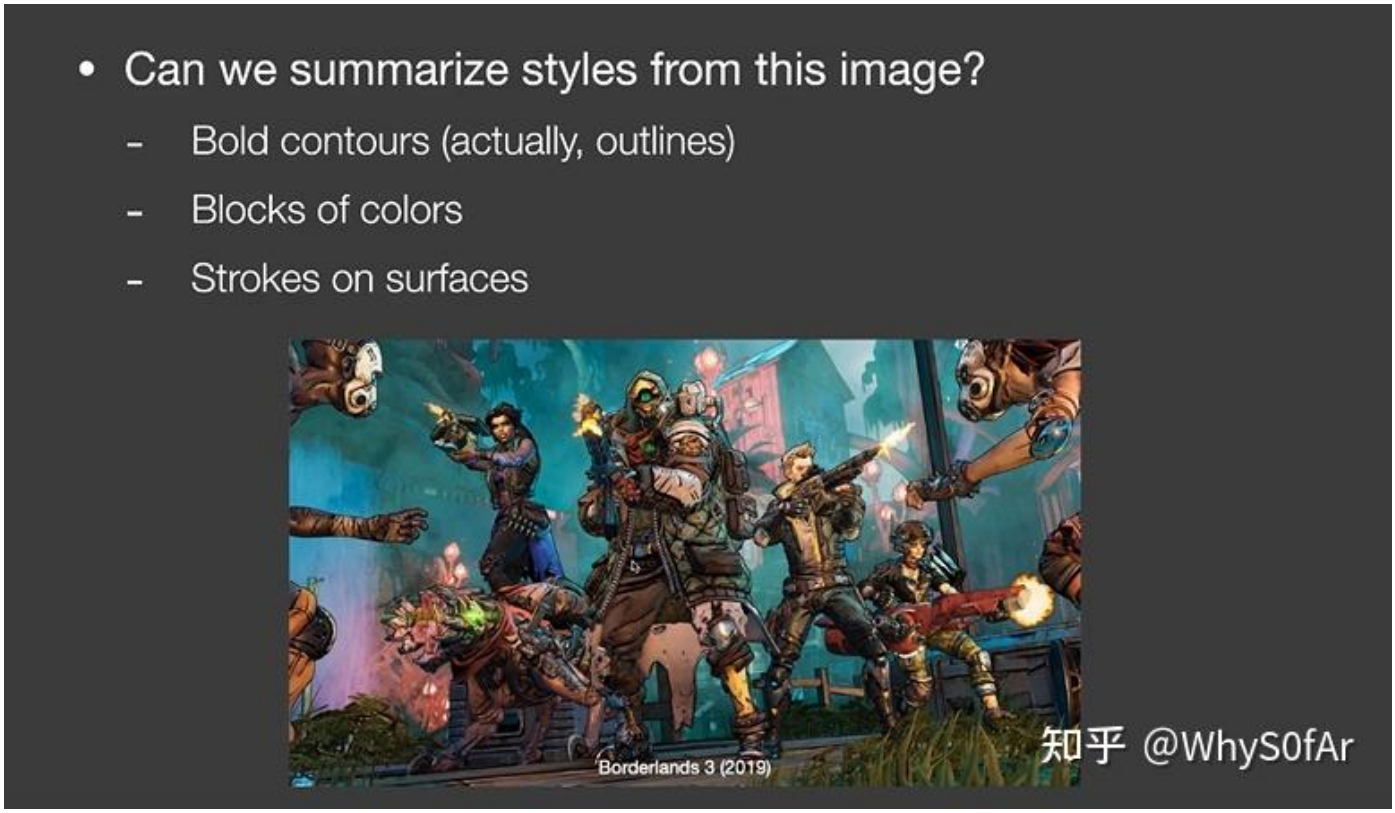


我们可以看到这个kernel竖着几乎没有变换而横着变化非常强烈,当一个像素周围是很平滑的,可以认为值几乎不变,因此在经过这个kernel后左右两边的值可以认为被抵消掉了.

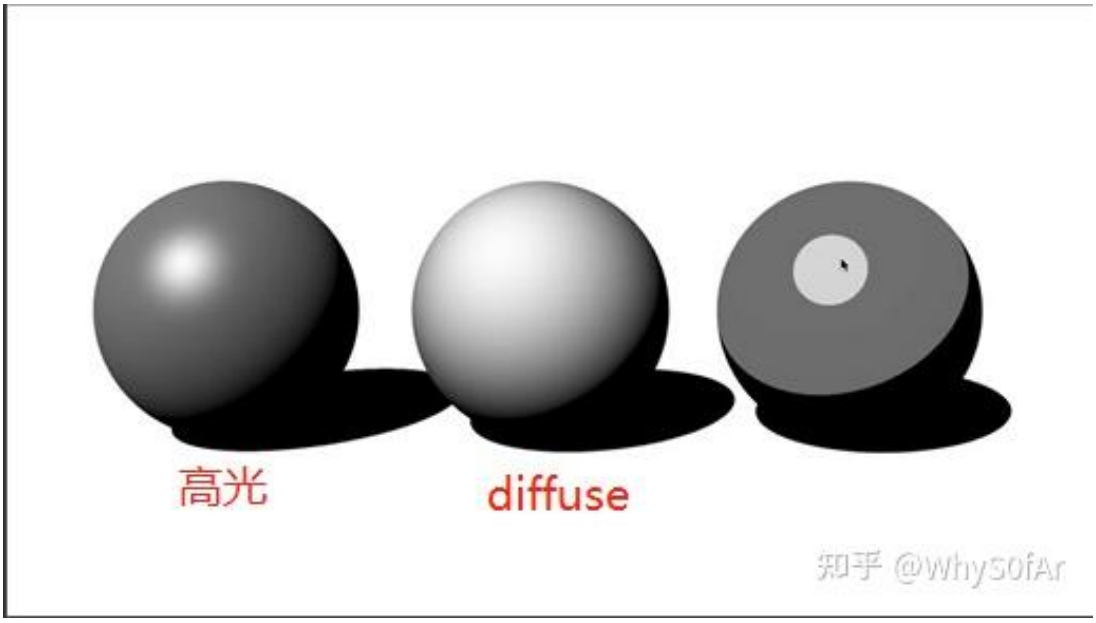
而如果在墙缝上我们可以知道,左边是白色的墙,右边是黑色的墙缝,因此他们之间的差距在经过kernel之后会被拉的更大,依次对整个图所有的进行处理,从而得到一个提取边的图.

除了在最后的渲染结果上进行图像处理,我们还可以在其他的图中进行处理:



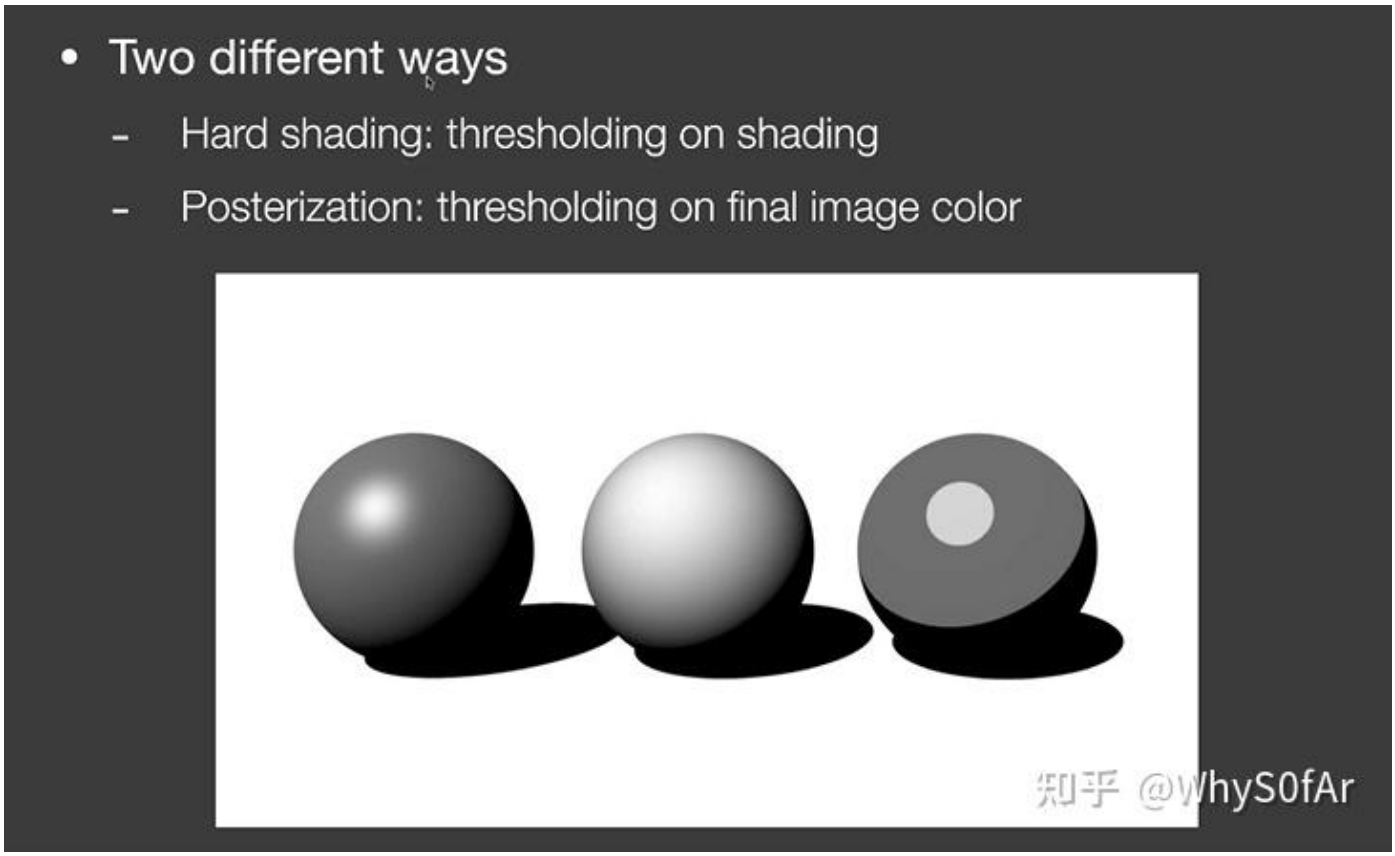


任然以无主之地系列为例,除了轮廓之外我么可以看到他们很多明显的色块,color blocks.



我们如果想要得到最右边的这种效果,我们该怎么得到呢?

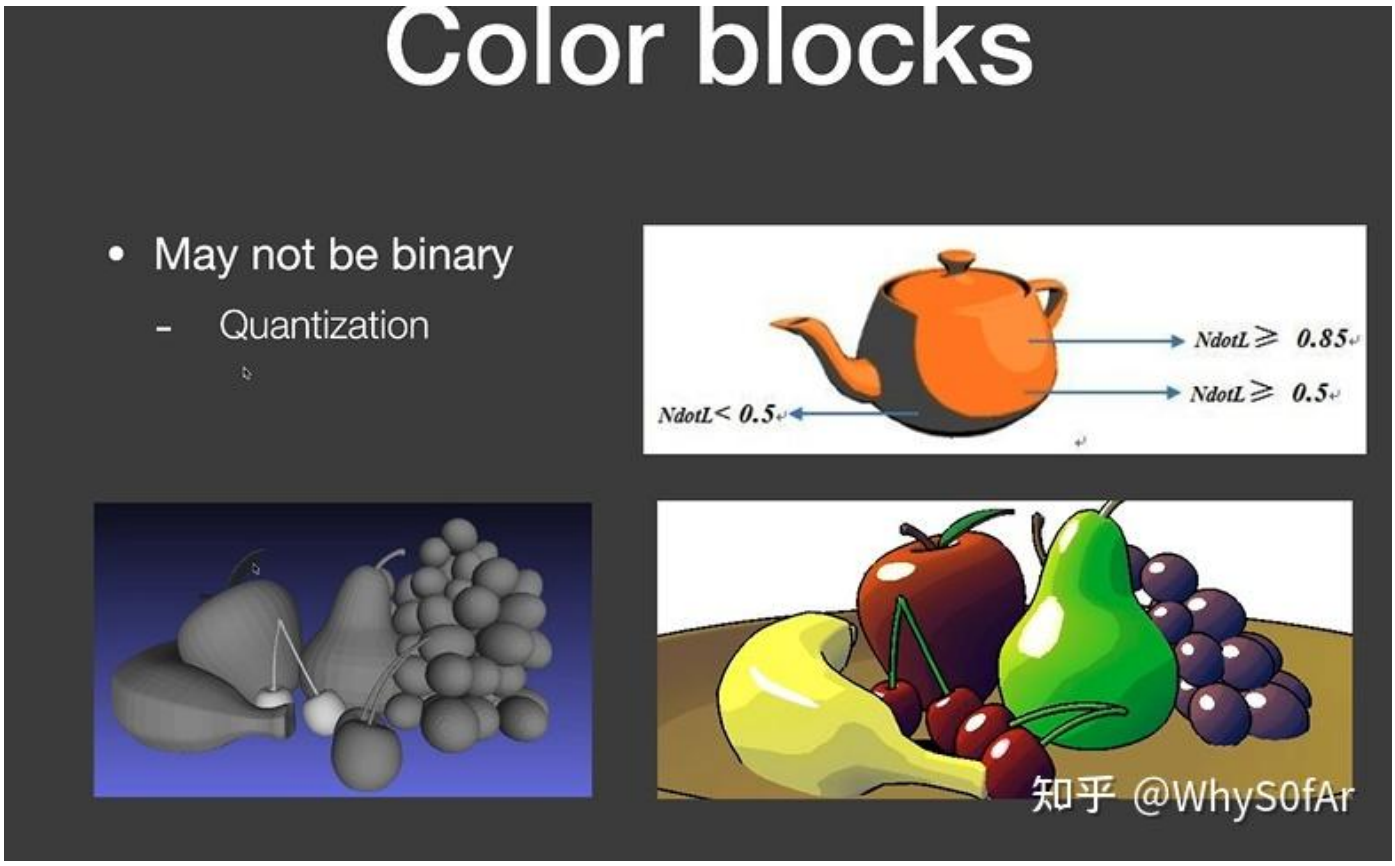
首先得到正常的Shading model算出来的结果,之后我们进行一个阈值化的操作(thresholding),



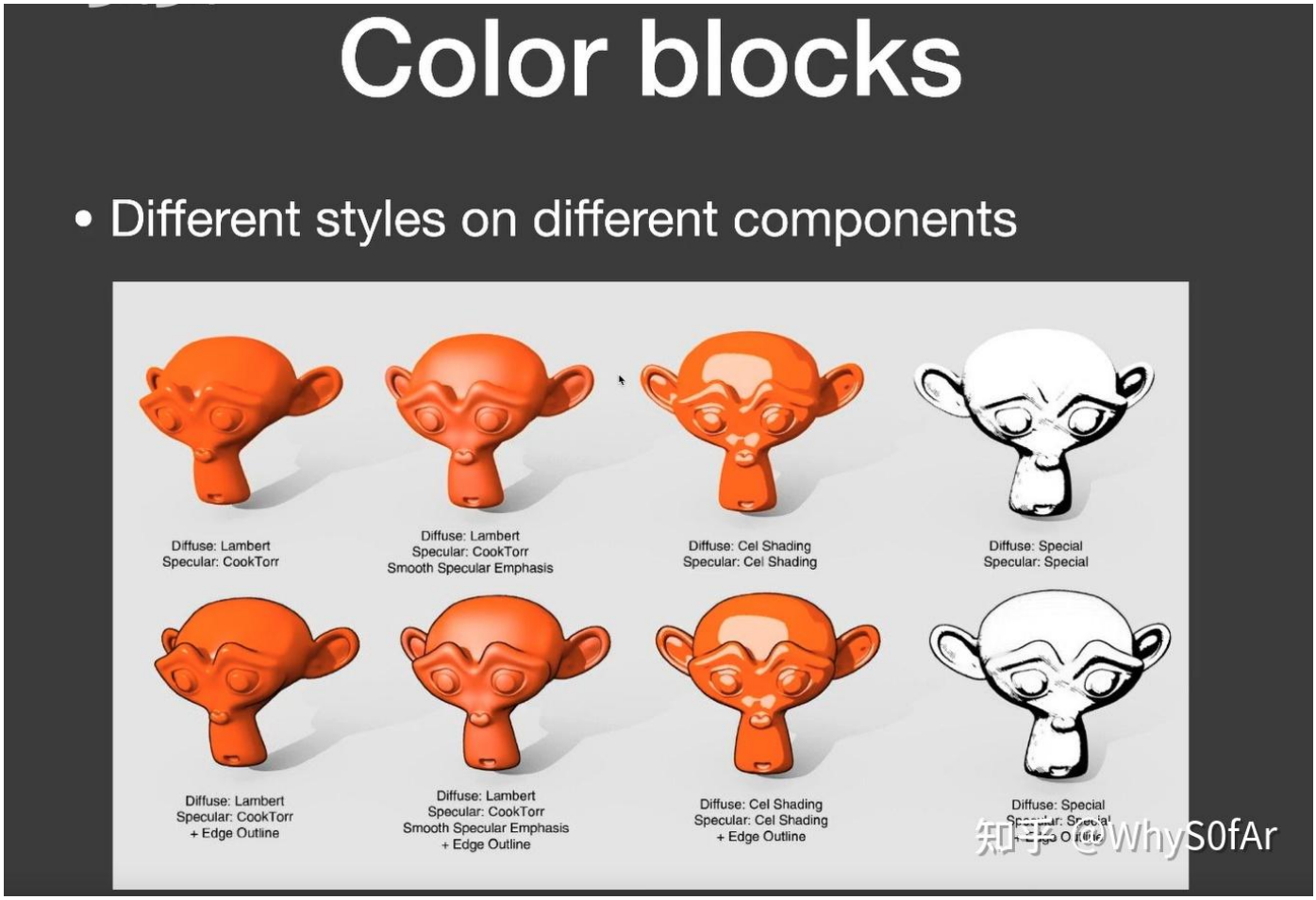
以高光为例,如果一些值值超过了1.5,则认为全是1.5,

没超过但大于0的,我们认为全都是0.8,

而那些本来就是0的依旧是0.



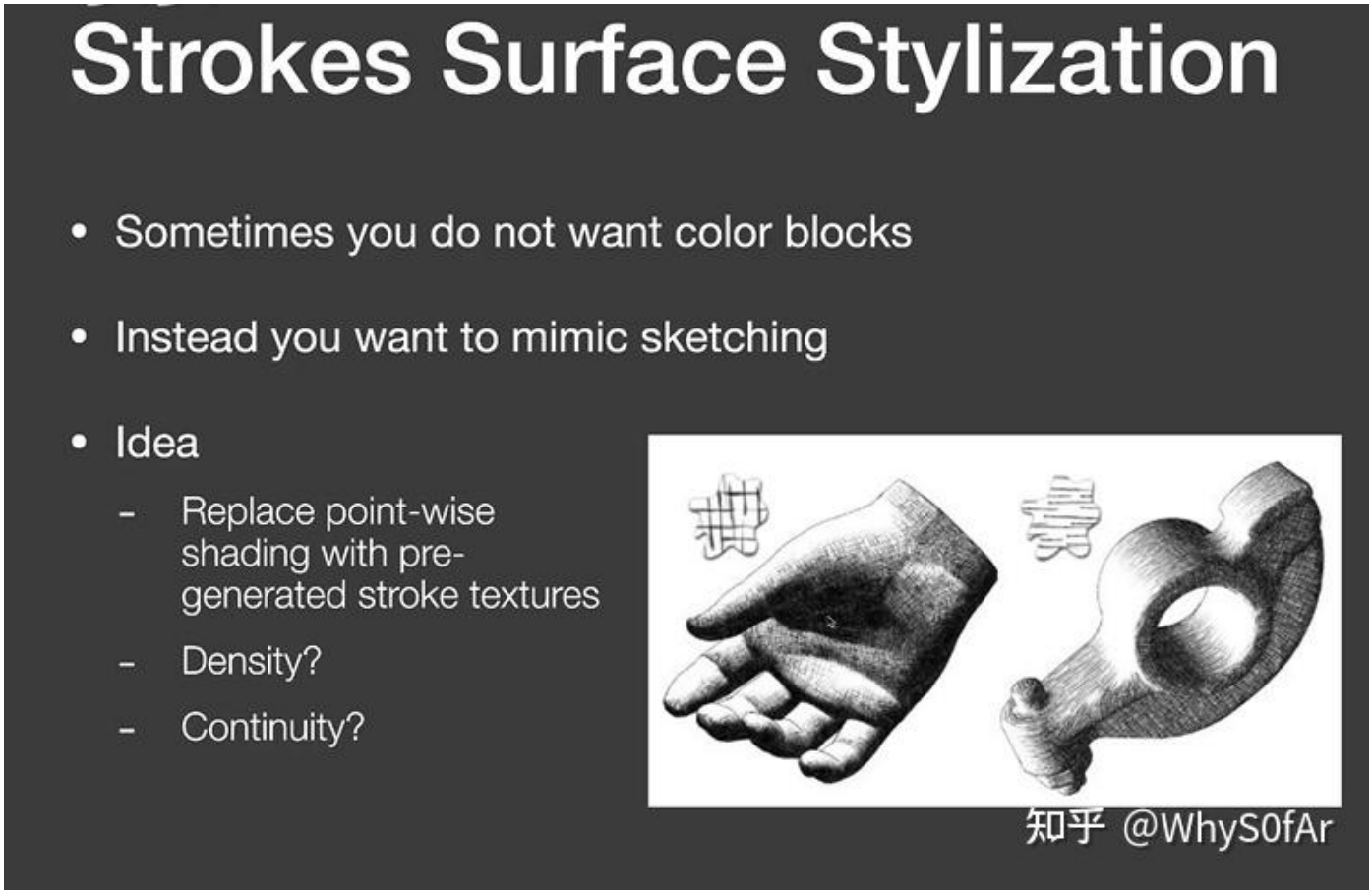
阈值化操作可以设定多个值,如图,小于0.5的是一个颜色,大于等于0.5的是一个颜色,大于等于0.85的又是一个颜色,从而拥有更多的色块.



我们可以根据自己需要去将不同风格的结果组合在一起,比如diffuse和specular都进行阈值化处理,或者只处理spcular不处理diffuse等组合.

Stroke surface stylization

但是有时候我们并不想得到色块的感觉,而是想得到素描一样的效果.



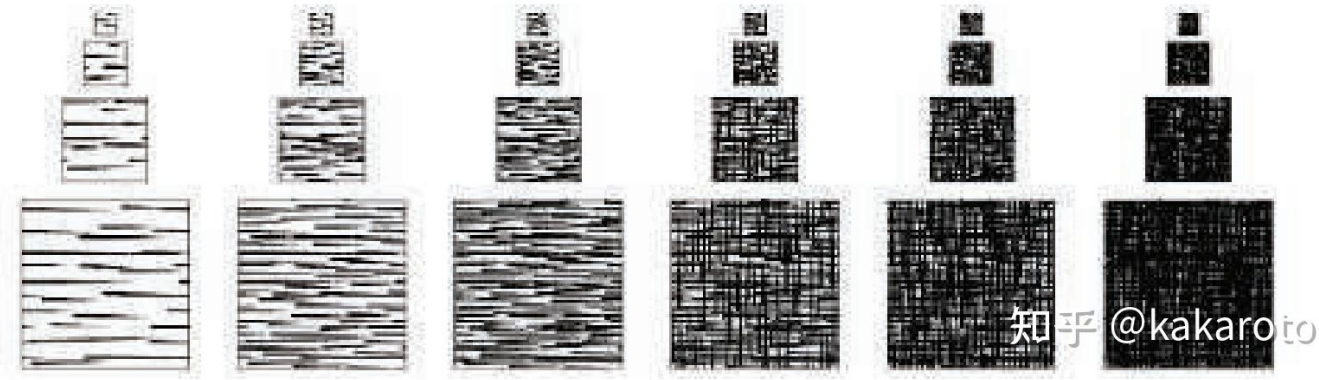
我们知道在素描上,往往越暗的地方需要更多的涂抹,我们把它理解成小格子,越暗的地方格子越多也就是密度越大,因此我们把格子密度与明暗度联系在一起

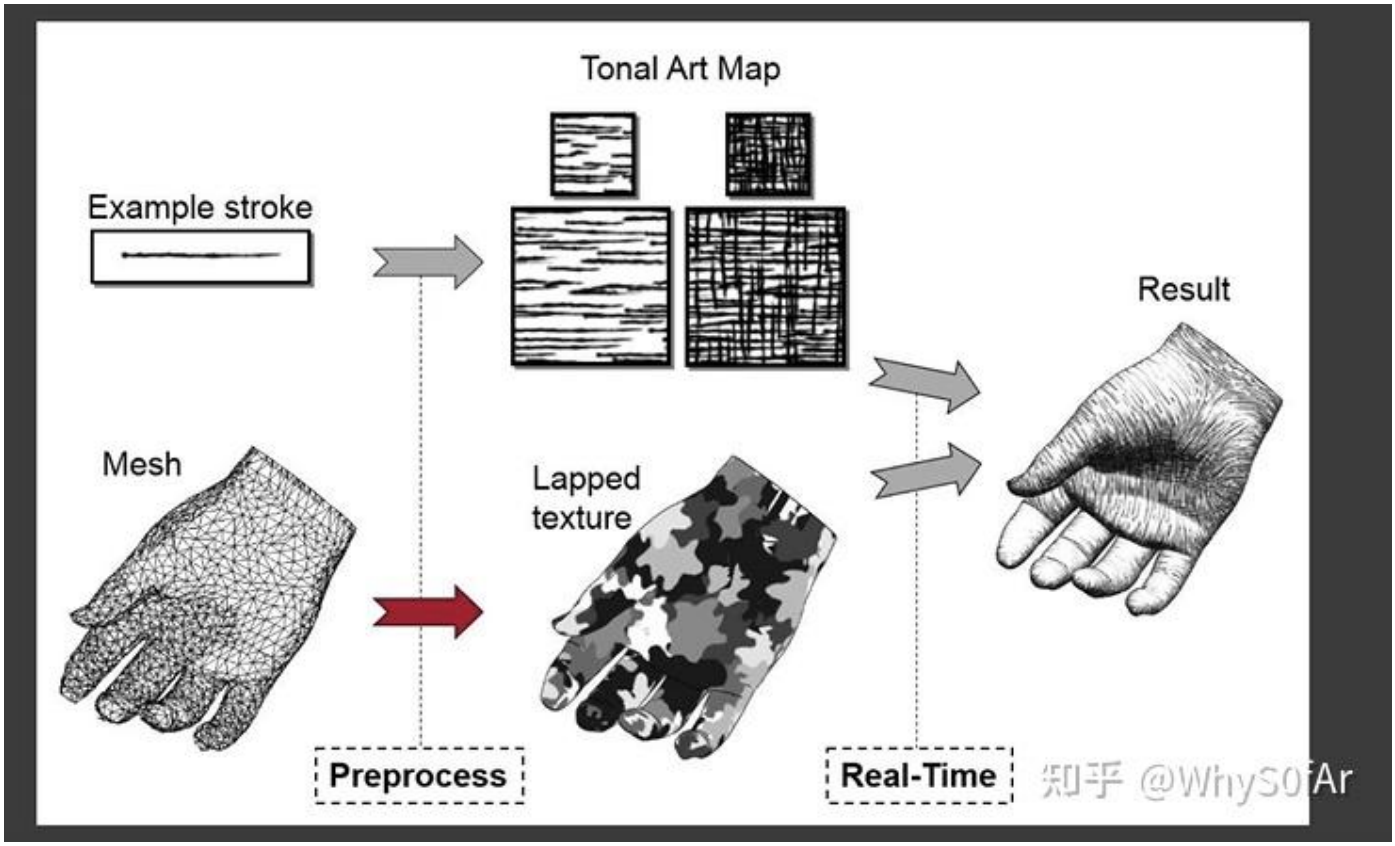
TAMs方法是一种实现类似手绘素描风格渲染的方法

主要分为两步

- 第一步是关于不同明暗设计不同的纹理
- 第二步是对于不同明暗的纹理设计其密度不变的mipmap,利用密度不变来保证距离远近不改变纹理的明暗,因为如果物体开始远离camera,由于纹理是贴在物体上的,由于距离越远导致图变小了,也就是密度变小了,为了让距离不影响到贴图,所以使用mipmap.

如下图所示：





总结:

NPR是一个art driven问题

因此我们要做出什么样的效果是artist们决定的,我们要思考和实现的是如何做出这种效果:比如边界

因此在这里沟通是最重要的,需要及时与美工他们沟通.

NPR做的效果好与坏,其实根本上取决于photorealistic模型,而不是后期的各种处理.