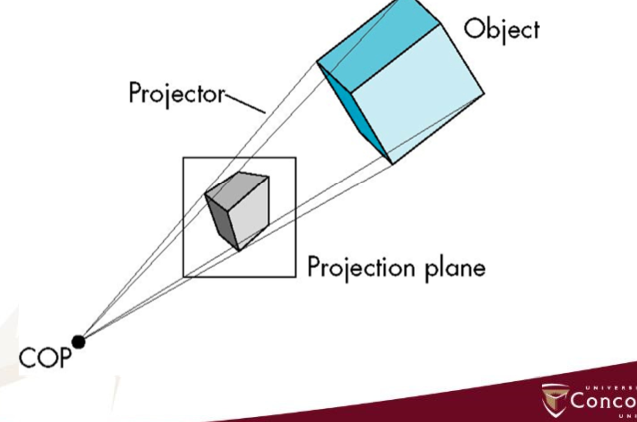
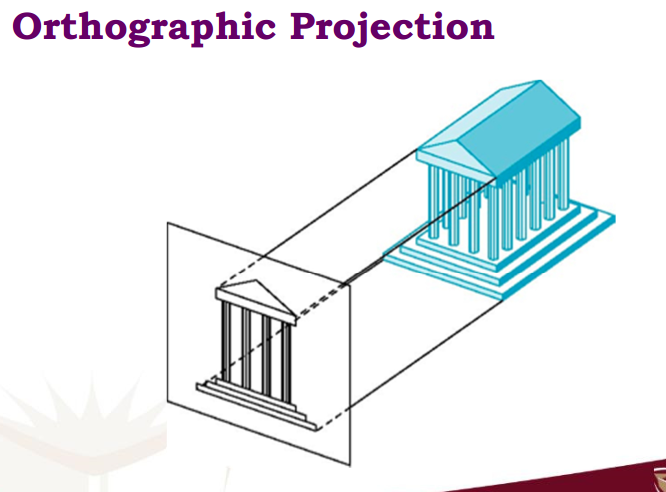
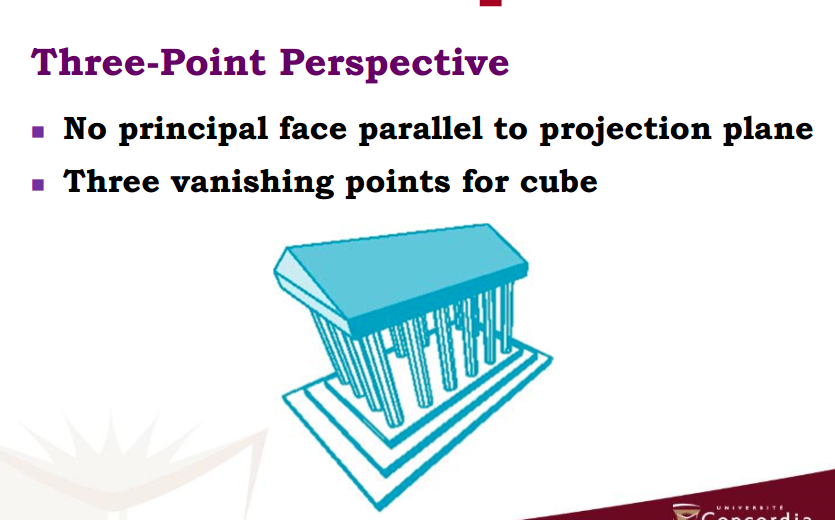
Projection

Perspective projection 中心投影法





垂直投影法



三点投影法，没有一个线与投射平面平行，三线交于一点

COMPUTER视角

一共有三个方面，在viewing process implement了这三个才会生成图片

Camera 位置

用model-view metrix设置

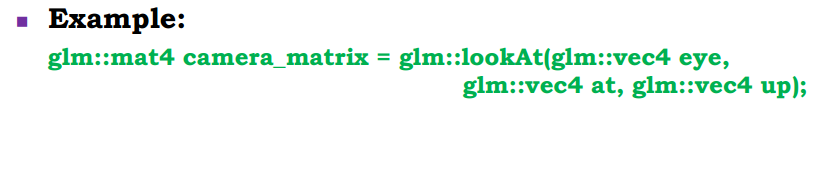
Lens 镜头

用projection matrix设置

Clipping

用视角容积设置

GLM::LOOKAT function

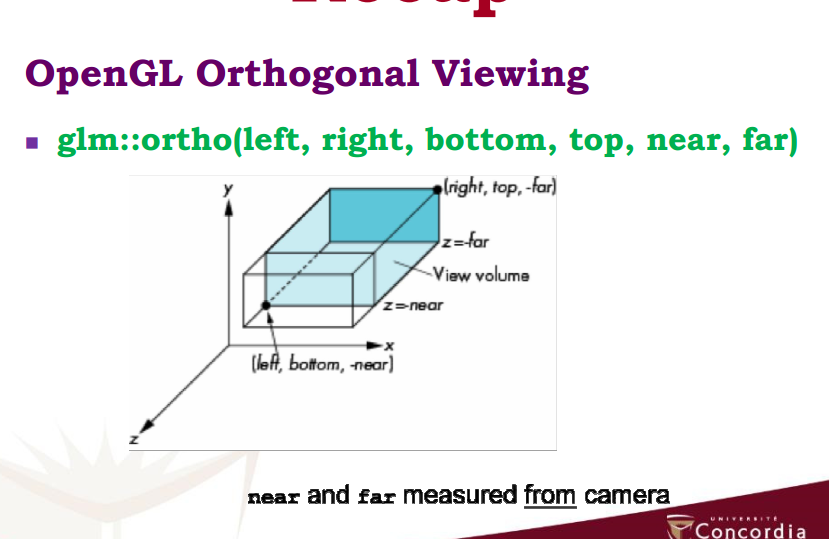


第一个位置是camera位置，at是视角方向（通常指向原点），up是010指向头顶

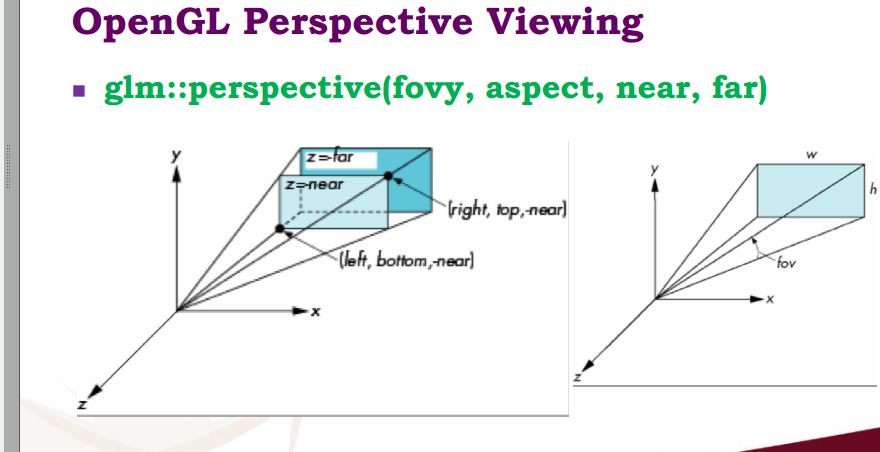
Projection与normalization正规化

默认的projection是orthogonal的，八角正方垂直

默认的view valume， xp=x, yp=y //就是长宽等于原长宽



left right bot top 取决于自己，Near填0，far填100



Opengl 还提供了perspective法

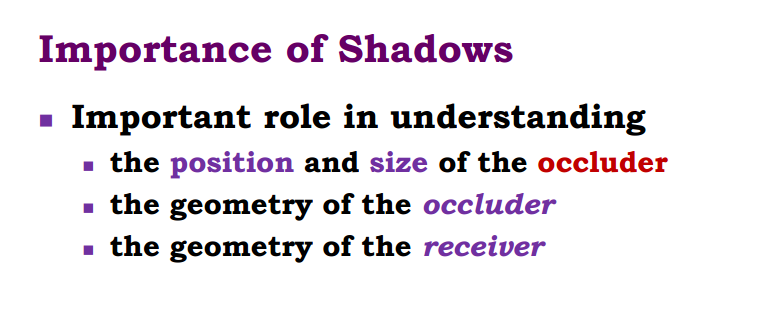
fovy就是角度，70°也行

aspect就是比例，1024.0f/768.0f

near填0.1,far填100

Fovy是xz平面与y轴的角度

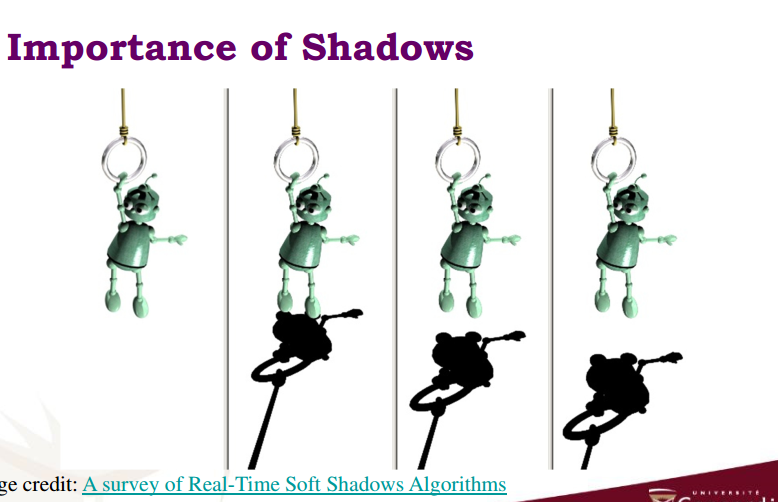
影子的重要性



影子最重要的几点，occuluder遮光器的位置以及形状以及大小

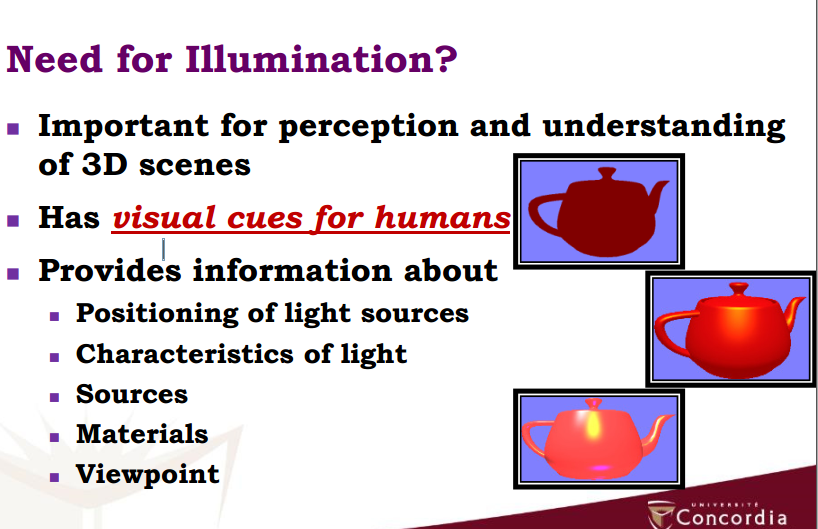
receiver的几何结构

Shadow让事物更真实



我们理解高度，光源都是通过shadow

Illumination照明



提供了信息

光源位置

光的特征

材质//木的不反光，金属反光

视角

光源

四种：

Point light

太阳（如果你把他想成太阳系）：

Isotropic:等方向性，向四面八方发出同样的光

所有光都是从一个点发出来的

容易计算（只要描述position就好）

不是很现实

会产生一个比较黑的影子

Spot light 聚光灯

Anisotropic:非均质的，释放一个锥形方向的光

是Point light的extension

容易计算（地点，方向，锥形角度）

产生比较黑的影子

Directional light

太阳（如果你想象成地球太阳）

所有阳光平行（因为足够远）

所以他的location是正无穷，你只需要描述方向

光源无限远

光的强度不随距离改变

产生比较黑的影子

Area light

现实，

光线由一个表面发出

难以计算（size, shape, position，intensity）

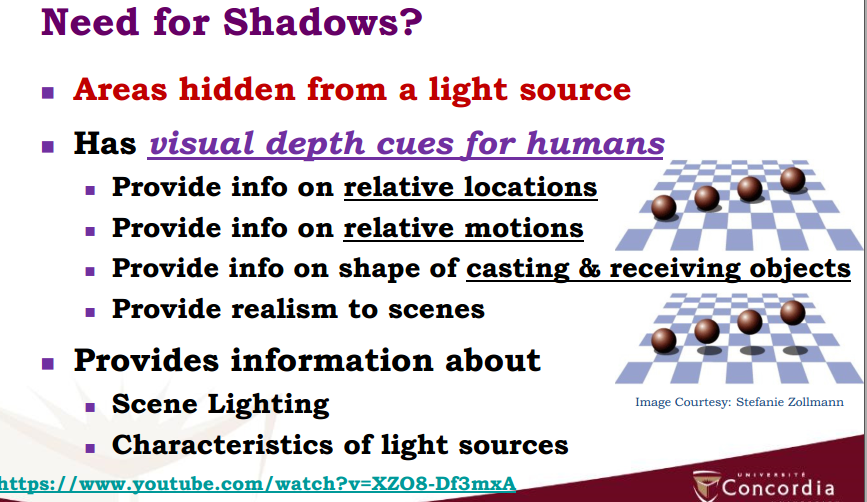
会产生更淡的shadow与更柔顺的光

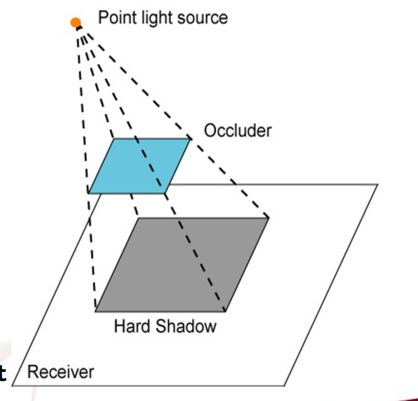
影子的重要性

影子是不受光源照射的地区

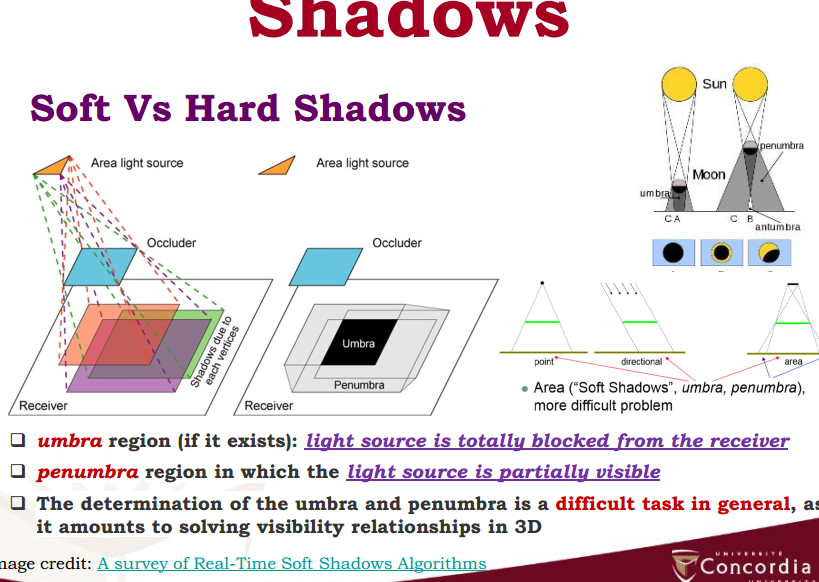
可以看出相对位置，运动，高度，让景色更真实

提供光源信息，场景照明





Occluder遮光板，遮光板直接把光源挡住了，产生hard shadow//比较实的shadow



Umbra,被completely block的区域

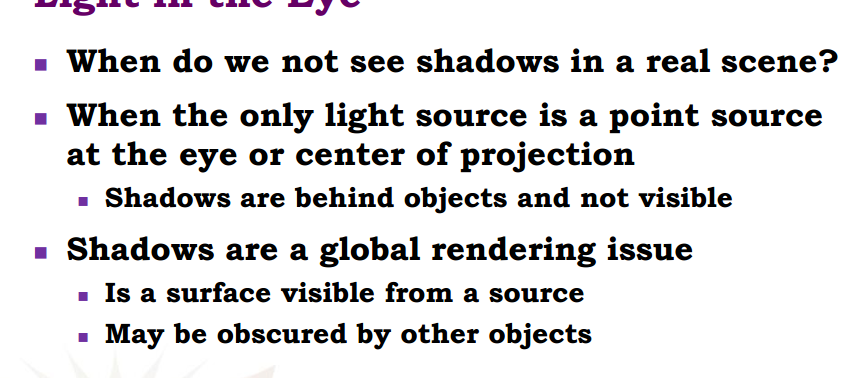
Penumbla，光源部分可见





Hard shadow，只考虑一个点or平行线的产物

Soft shadow，考虑一个面光源的产物，结合umbra与penumbra

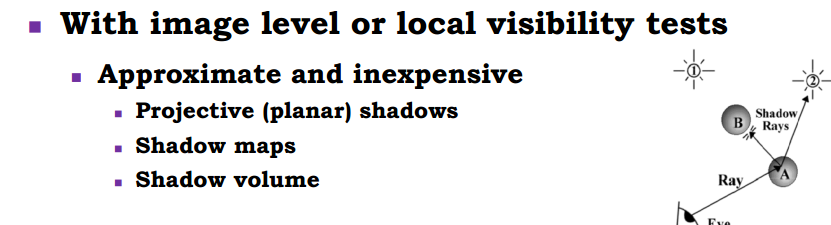
.什么时候我们看不见影子

只有一个点光源在眼睛里或在投影中心，那么物体背后的影子我们看不见

shadow属于global rendering问题

Light in the eye

渲染shadow的两种主要方式



1.image level

确定光源，确定camera,用camera作为主要视角来渲染（便宜）

1.投射的平面的影子

2.shadow map阴影图

3.shadow volume 阴影容积



World level

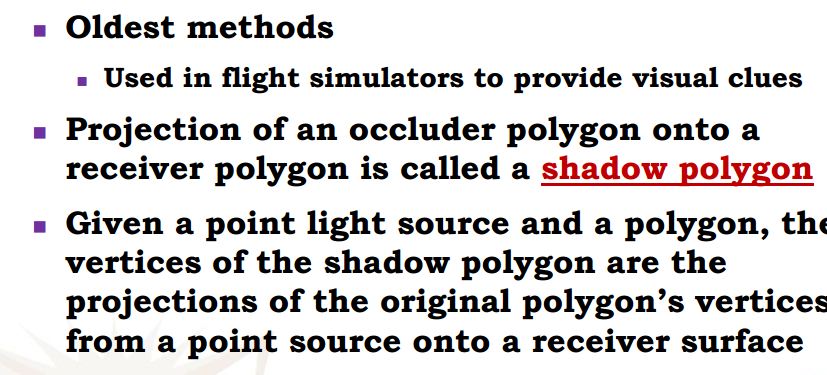
准确但贵

1.光线投射

2.辐射

3.montecarlo simulation蒙特卡洛模拟

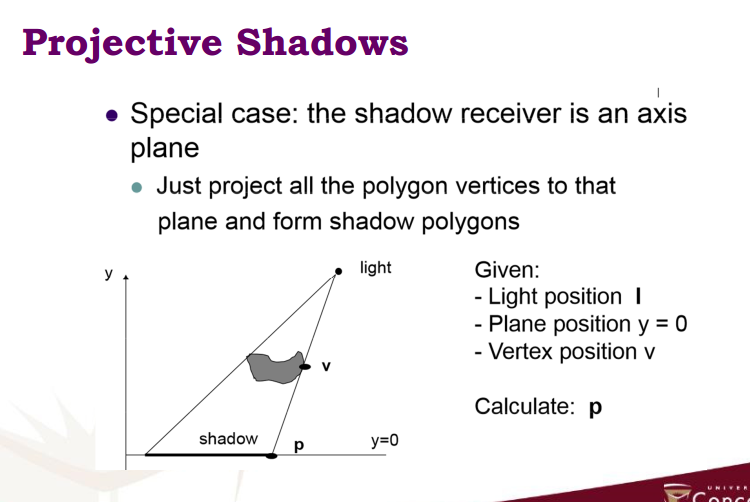
Projecttive shadows



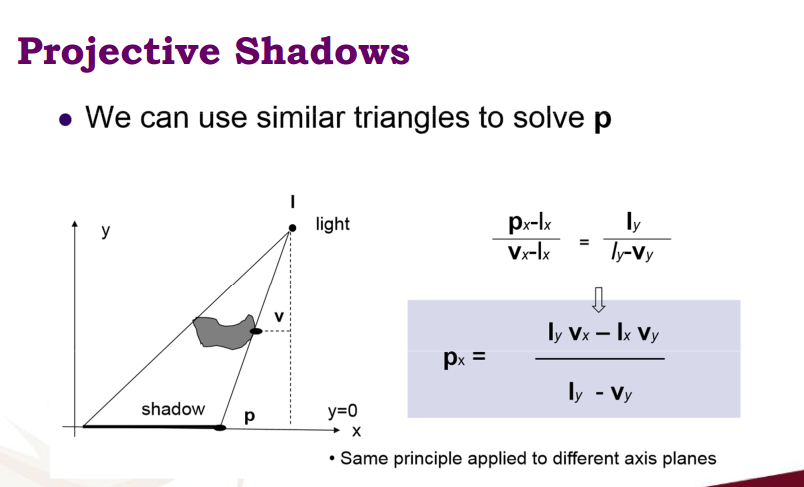
最老的方法

把待投射多边形投射到一个receiver polygon上，这个投影receiver polygon（的阴影部分）叫做shadow polygon

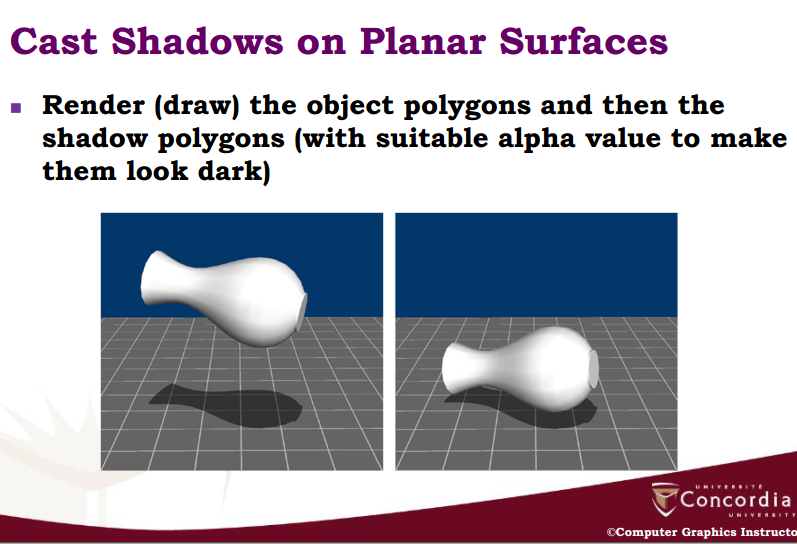
给一个点光源以及一个多边形，那么阴影部分的端点就是从点光源到receiver面的端点



特殊情况，shadow receiver是一个平面，那你直接把每个端点连起来投射到plane上就是shadow polygon了



相似三角形



在平面上画出阴影

画出这个object多边形以及shadow多边形（就是让object 多边形更暗）

Projective shadows(limitations)

shadow polygon generation阴影多边形的生成（z fighting）

解决方法：用glPolygonOffset 给shadow polygon加一个offset偏离量

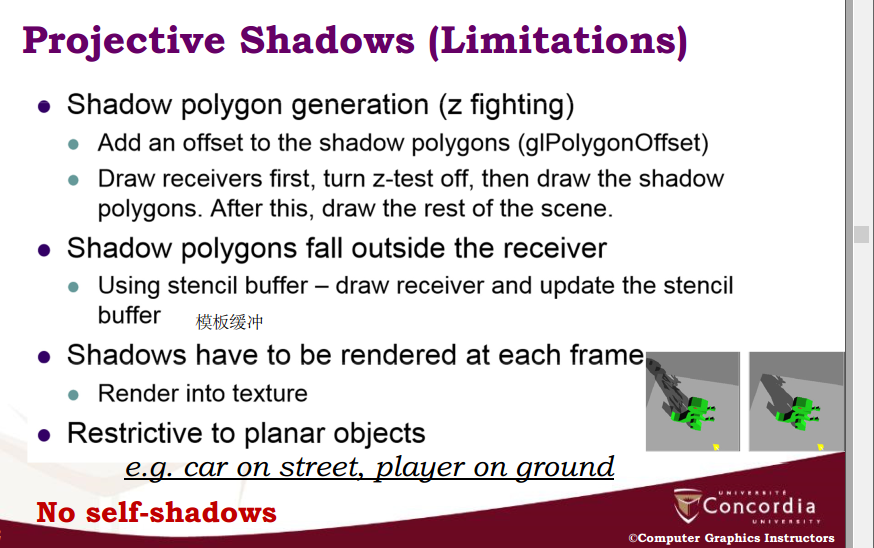
或者先画receiver，然后把z-test关掉，然后画shadow polygon，最后再画其他场景

Shadow polygon掉到receiver外面了

使用stencil buffer模板缓冲

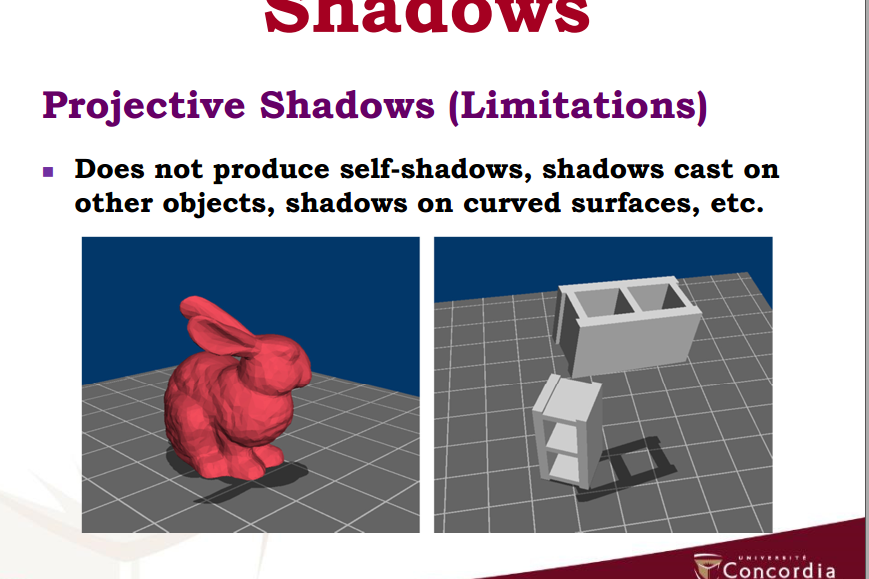
shadow必须在每个frame里都被画（视角frame等）

解决方案，在texture纹路里就渲染好影子

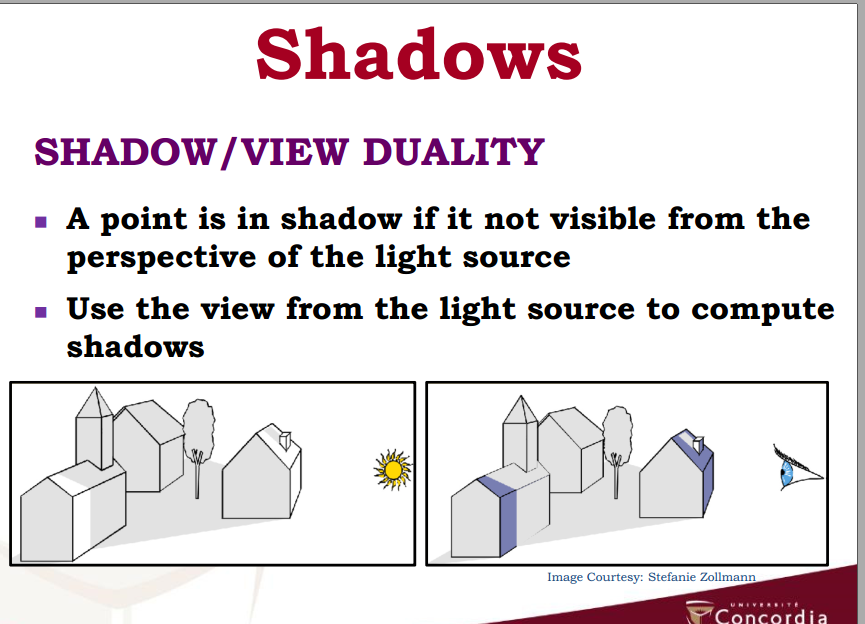


还有限制就是不能产生自己的影子

因为是投射在其他物体上的



一些凹凸面物体会自带阴影（图中是已经有阴影的情况）



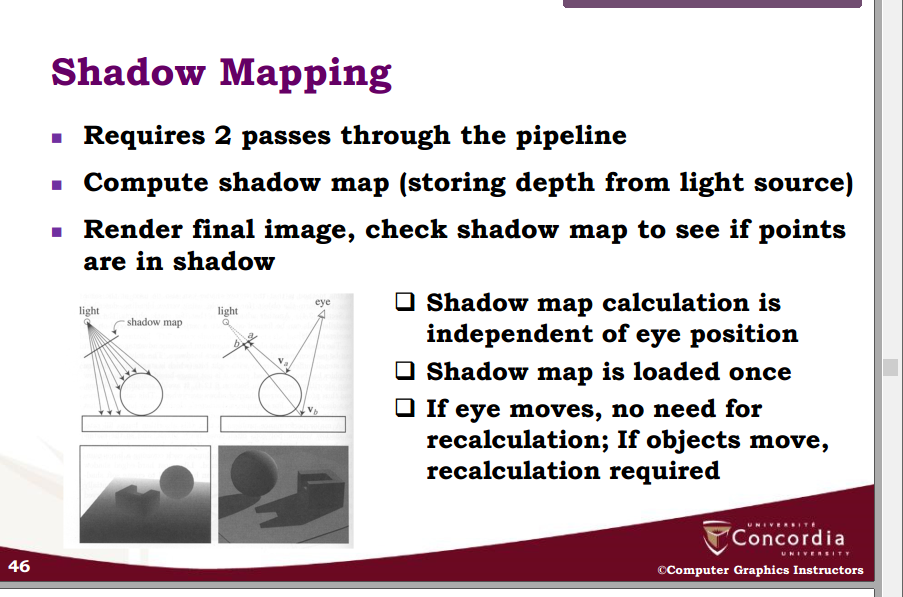
影子二相性

如果一个点从光源的地方看不见，那么就是shadow

因此我们可以用光源作为viewMatrix来计算影子

shadow mapping阴影贴图

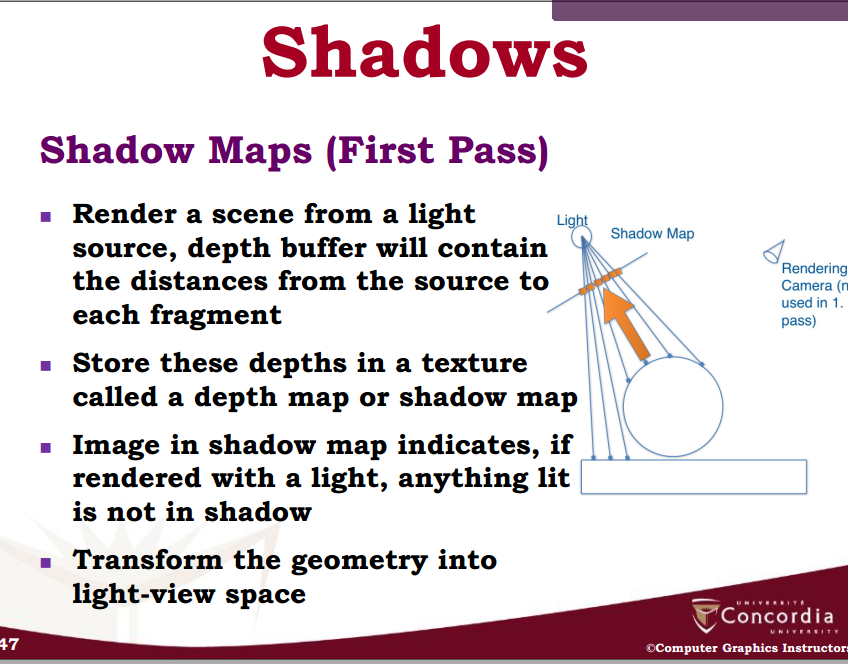




需要两次经过pipeline，第一次以光源为视角，计算阴影（存储在depth里），第二次以eye为视角，生成最终图片

shadow map的计算与眼睛位置无关

shadow map实际上只计算一次，如果眼睛移动了，没必要重新计算，如果object移动了，需要重新计算



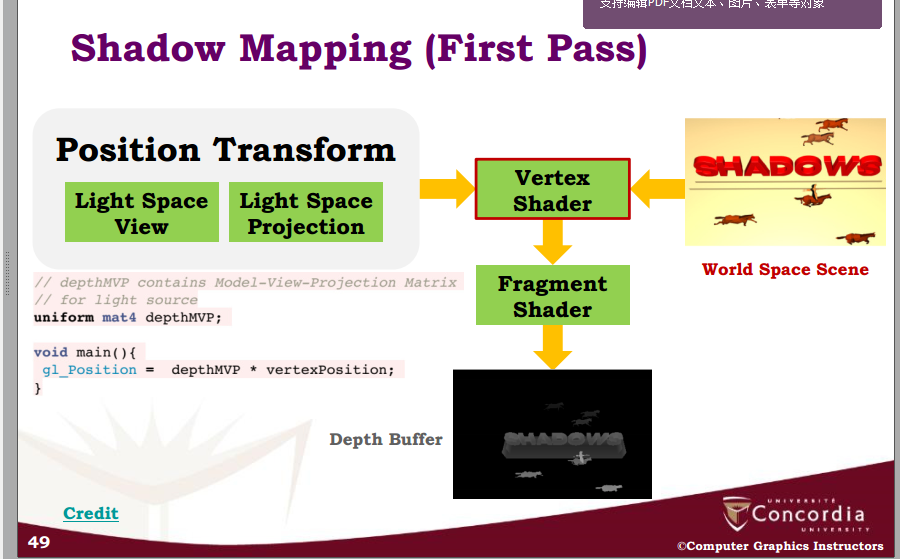
第一次：

从光源绘画场景，用depth buffer来存储每一个像素点到点光源的距离。

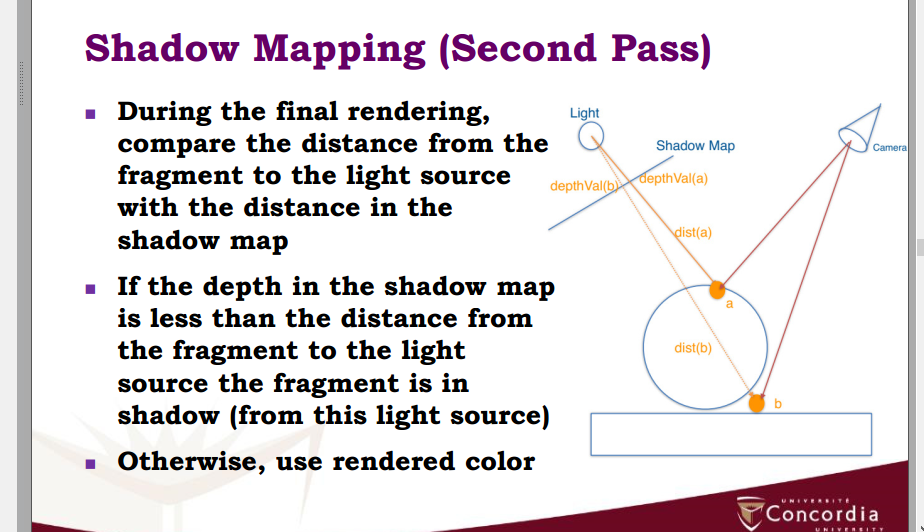
把这些距离存储在一个叫做depth map 或shadow map的纹理中

任何被照亮的物体都不在阴影中

把这个几何图形用light-view 光-视角 transform

.light space view加上light space projection进入vertex shader找到端点，进入fragment shader上色，然后存储到depth buffer

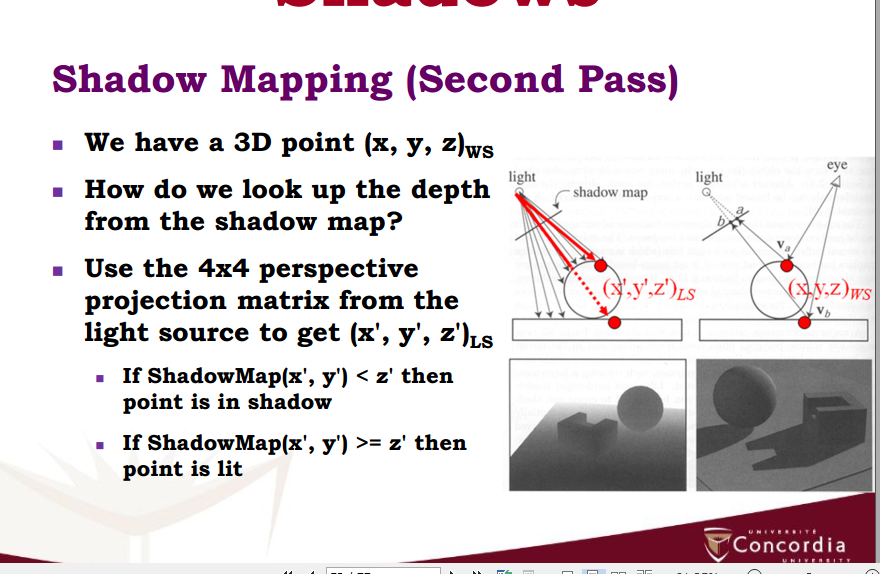
第二次观看



这是最终rendering，比较这个像素到光源的距离和他在shadow map的距离

如果shadowmap深度小于像素到光源距离，那么就在shadow里

不然就是平常颜色（没有shadow）



我们已经有了3D点 eye视角

我们怎么从shadow map得到depth

首先shadow map是一个2D图片，

用光源的4x4投射matrix来得到xyz ls

然后将xy带入 shadowmap纹理图，如果小于z距离，那么就暗

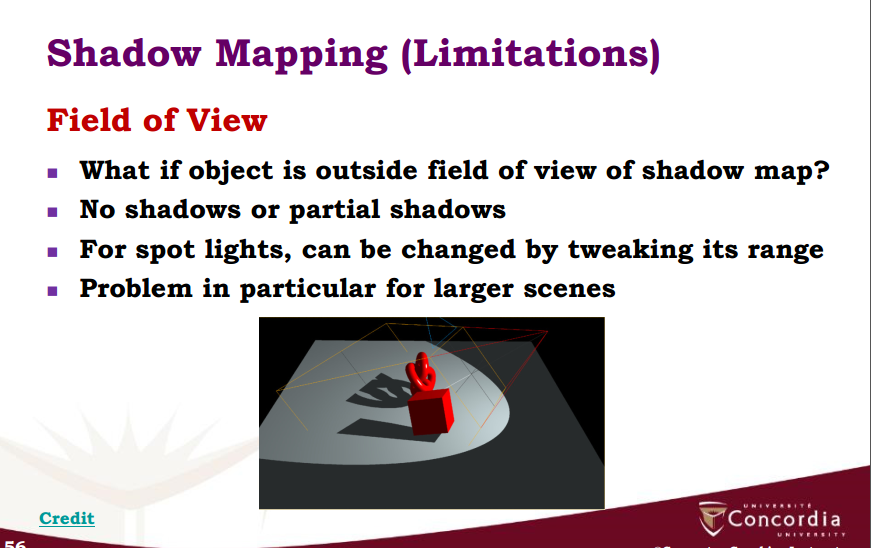
如果大于等于，亮





具体代码。 实际距离大于纹理距离，visibility=0,0

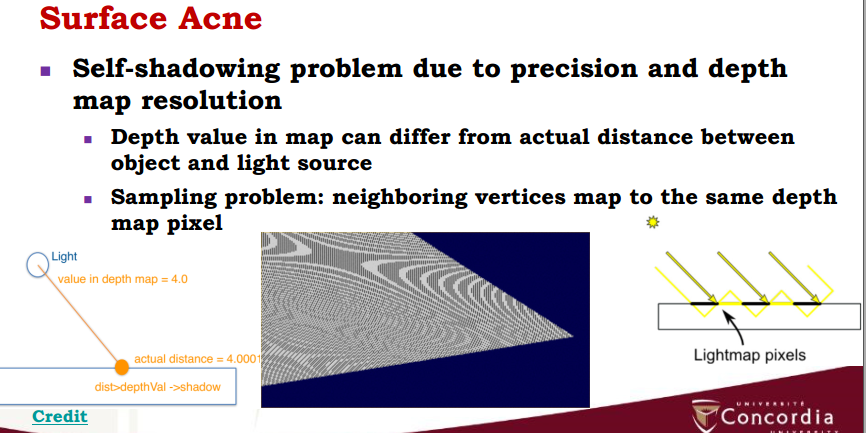
shadow mapping的缺陷



如果一个object在view field的外面 ，那就只有部分shadow或没有shadow

这个问题经常出现在大型场景中

surface acne表面粉刺

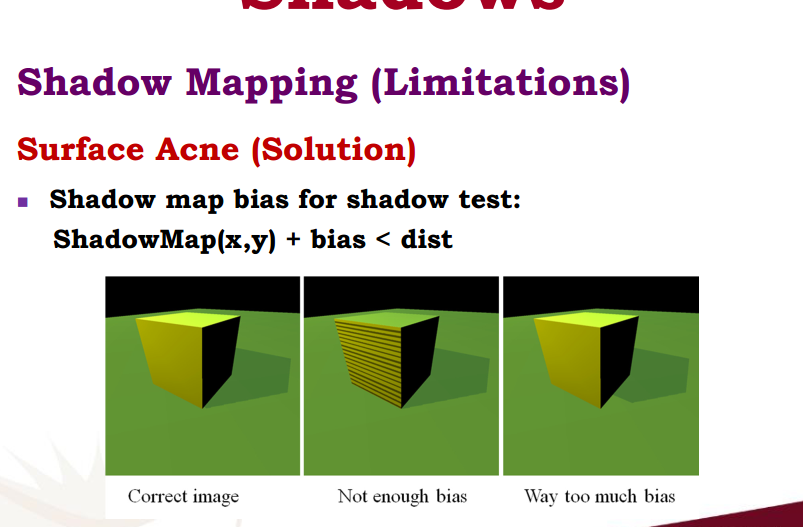


由于精度与depth map导致的self-shadowing问题

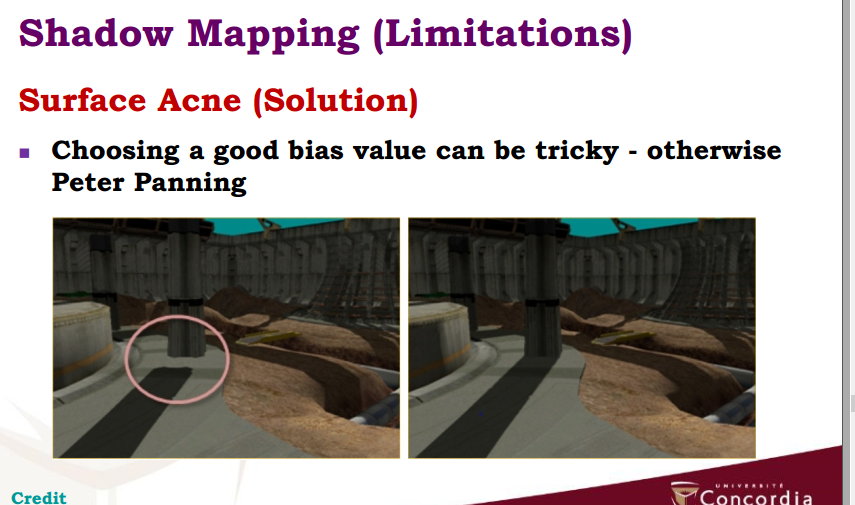
map里的depth value 可以和实际距离——object与光源 不同

例如：相邻的两个vertice投射到了同一个depth map pixel。

就压缩出来的shadow mapping可能一个pixel实际包含现实两个或多个点，导致他们share了同样的depth value

使用偏见法

shadow test的时候加一点偏见来模糊，只要偏见足够，就能使影子判断范围不那么精细



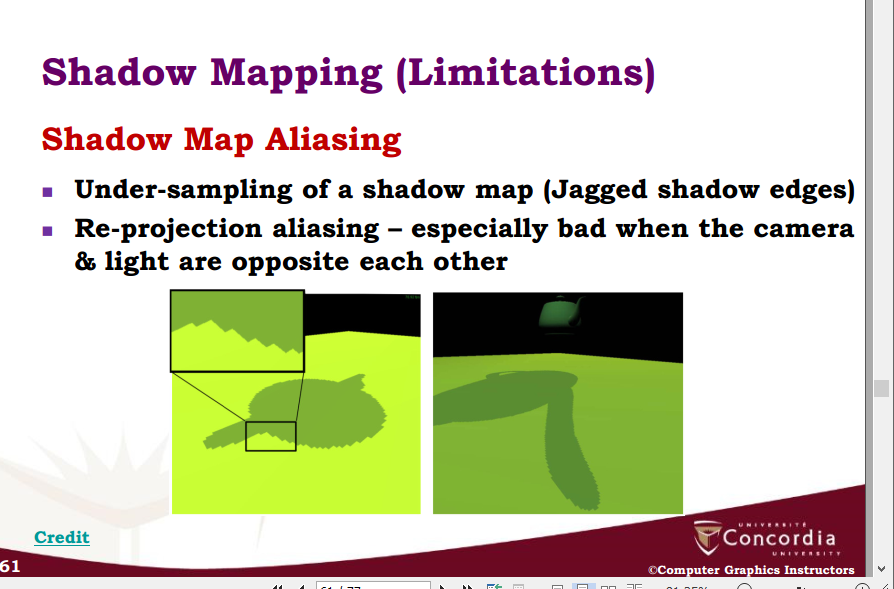
选取合适的bias很重要，不然就会让影子很奇怪

shadow map aliasing 锯齿

具体取决于shadow map resolution分辨率



更高的分辨率锯齿越小要的memory越多

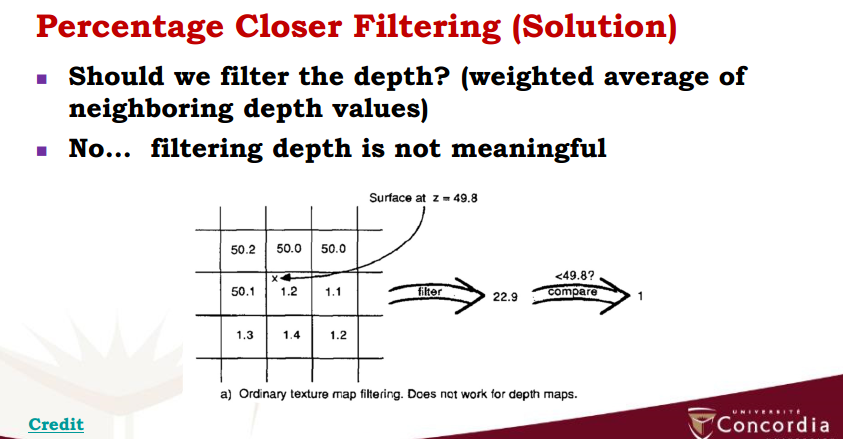


还有的锯齿是因为

under-sampling

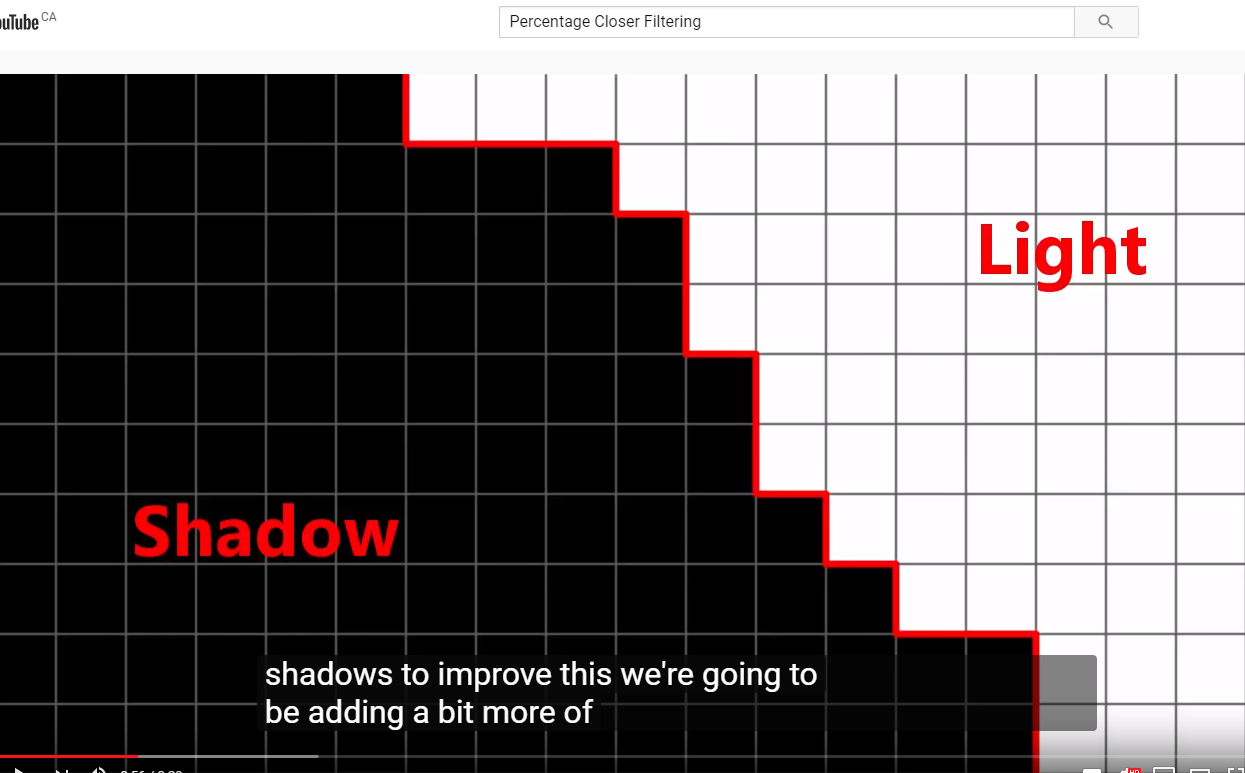
re-projection锯齿 //这两个背下来就行

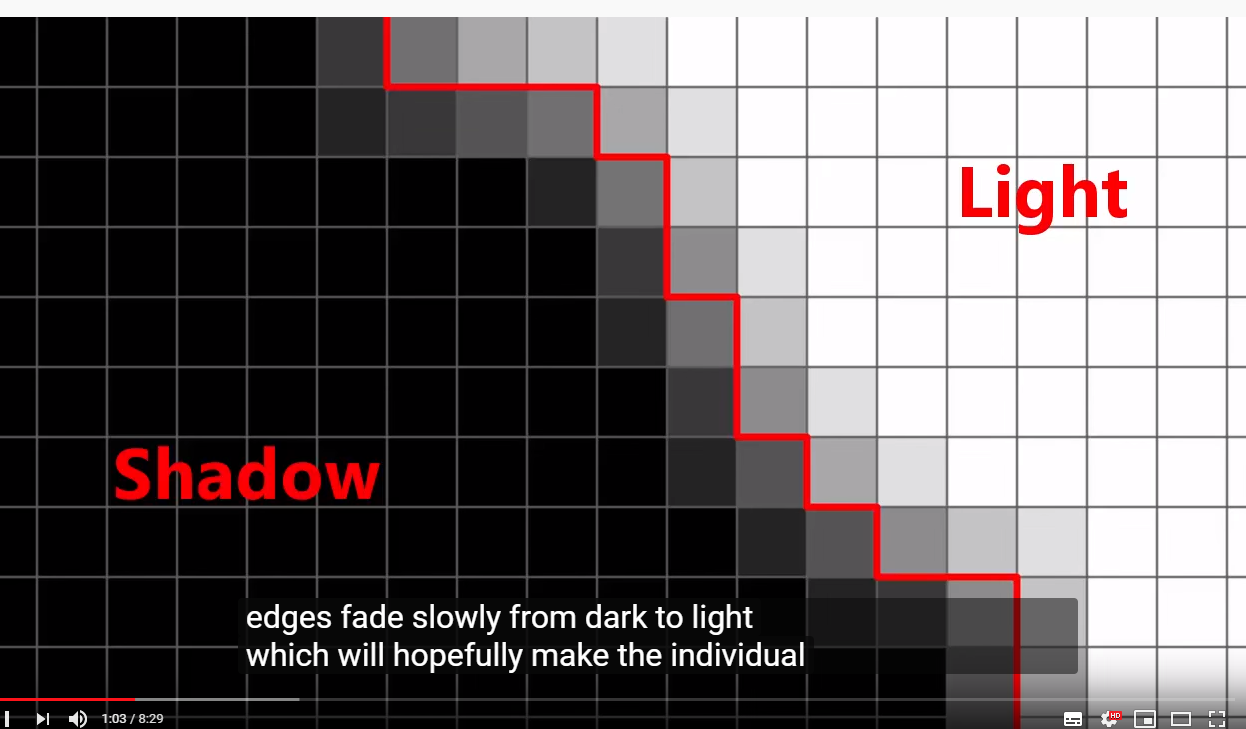
解决锯齿的通用方法



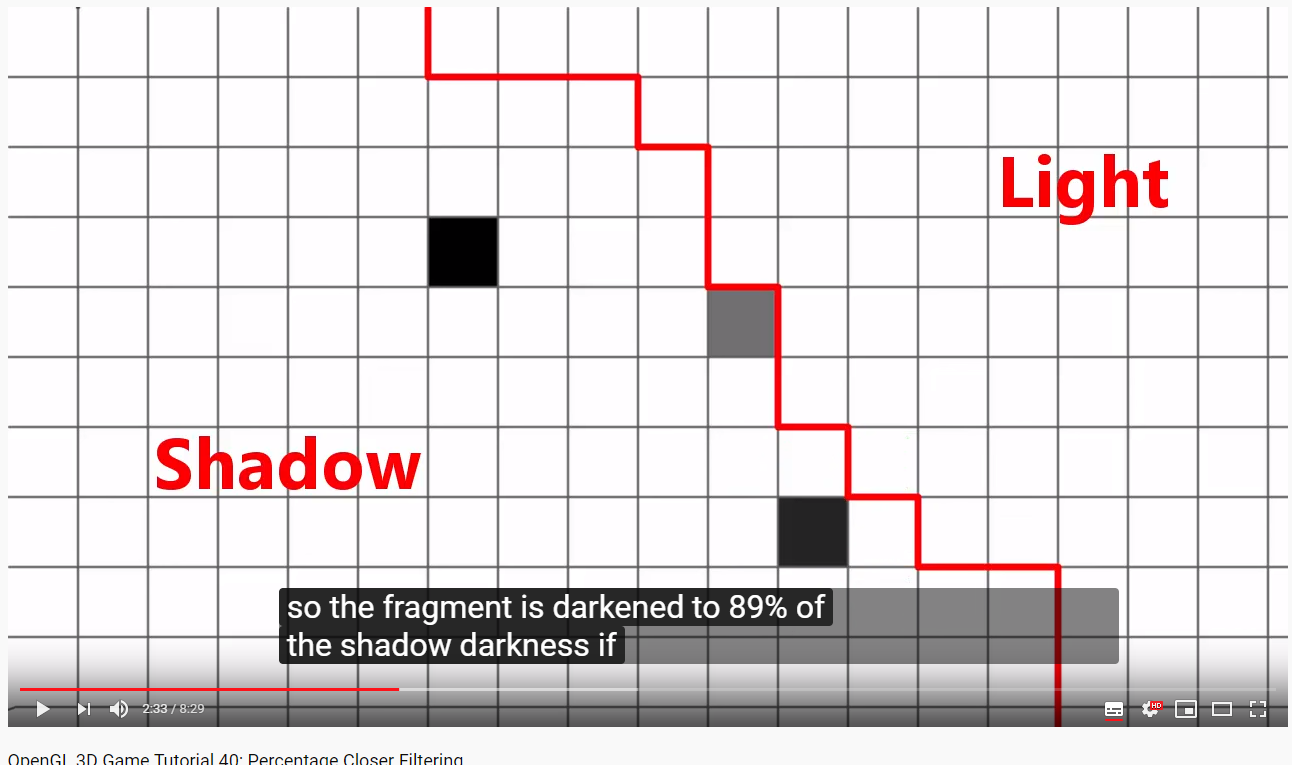
percentage closer filtering//过滤 百分比渐进过滤

这里50到1太突兀了



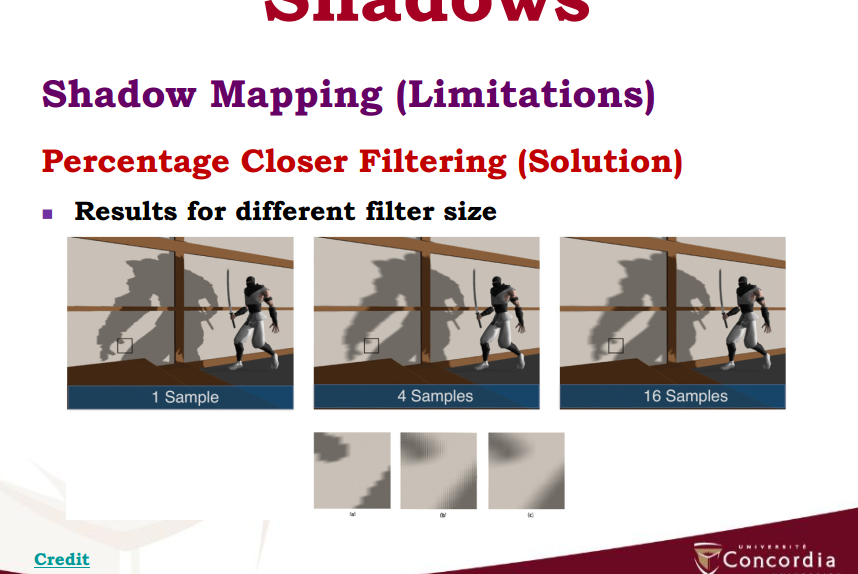


我们做一个渐变



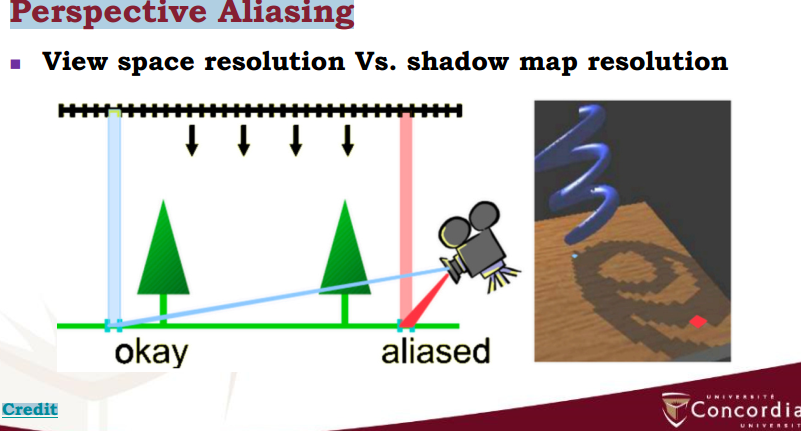
做一个九宫格，中间的那个56％在shadow里，所以darkness 56％

右下的那个89在shadow里，所以就更暗



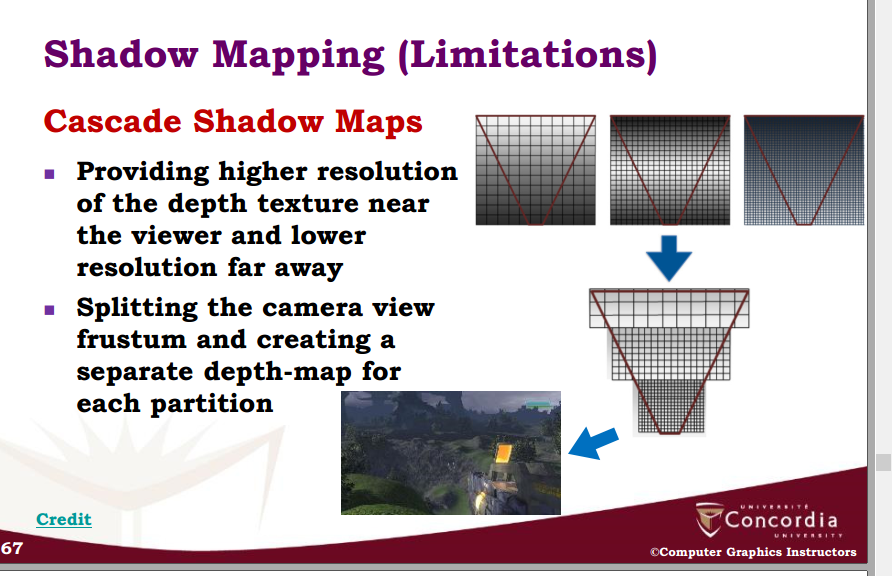
取得filter过滤器越大，越fake softer，看起来更顺更模糊

perspective aliasing 角度锯齿



离camera较近的角度锯齿更大

解决方案

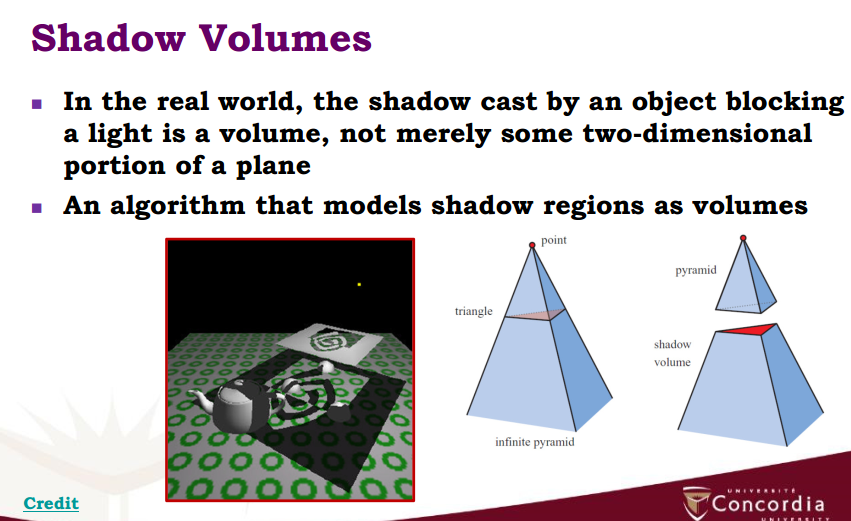


对离camera较近的地方提供高分辨率depth texture

Shadow volumes

我们现在是在平面上把点涂黑来创造shadow，然而实际生活中shadow不是一个二维图片，而是一块被遮挡的体积

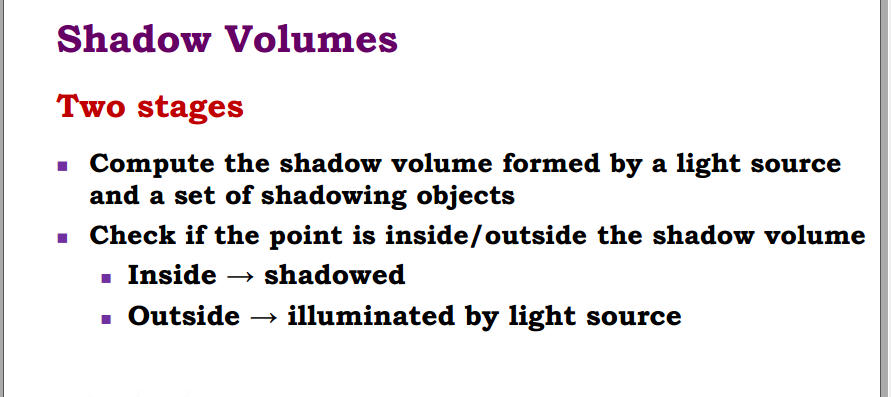
shadow valume就是将体积计算成shadow region的算法

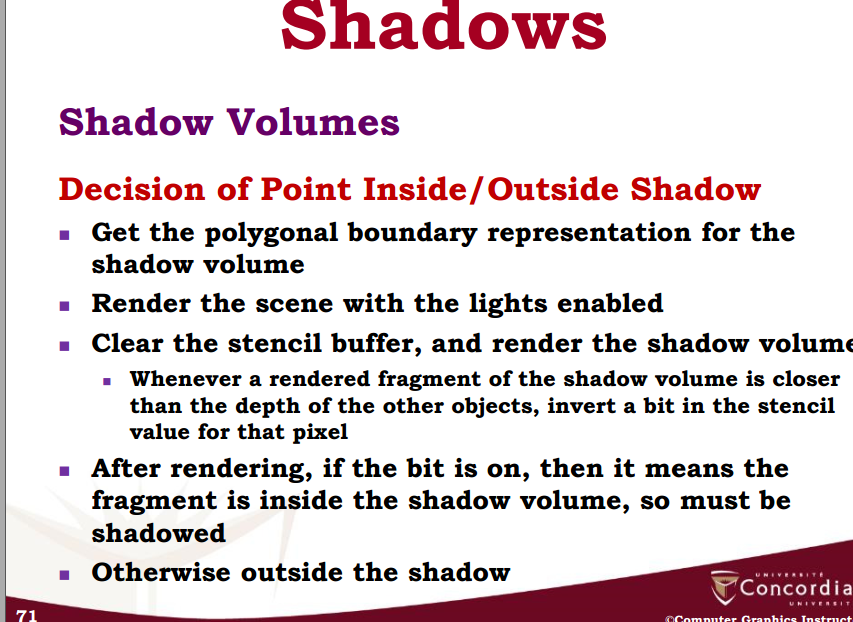


两大阶段：

计算由光源与一组遮挡物形成的shadow volume

查看点是否在shadow volume内，在就涂黑，不在就是有光照的





先得到一个多角bound代表shadow volume

绘制这个场面包含灯光

清除stenciLbuffer，绘制正式shadow volume

在绘制完以后，如果bit 是on,就意味着这个像素在shadow volume里，必须shadow，否则不shadow

优点：

可以用作常用图像硬件

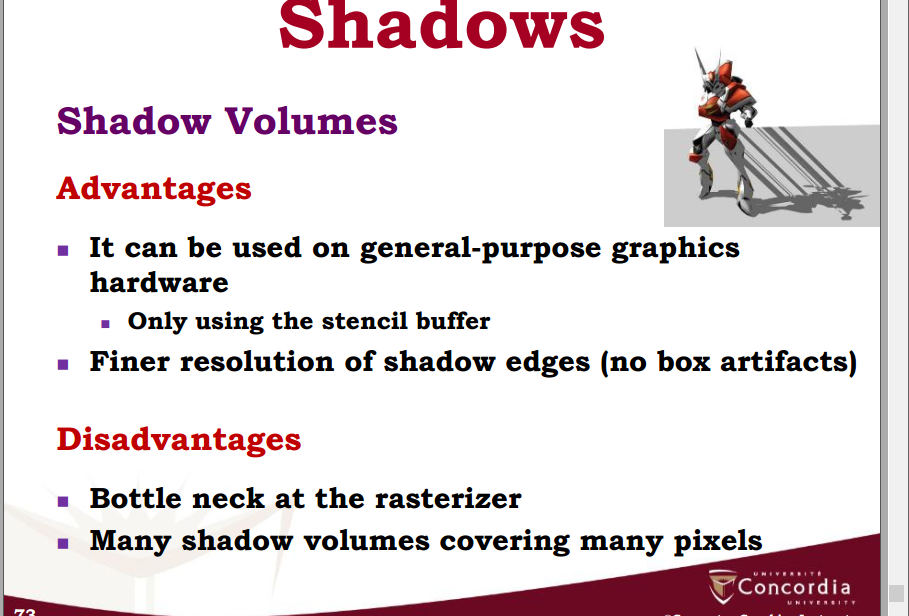
只用stencil buffer

更好的阴影边界分辨率

缺陷

光栅的瓶颈

shadow volume越多控制的pixel越多



ray tracing



射线追踪，把每一个可见面追踪一条射线到那个实际点再连向光源，如果中间有阻挡物，那就是暗的