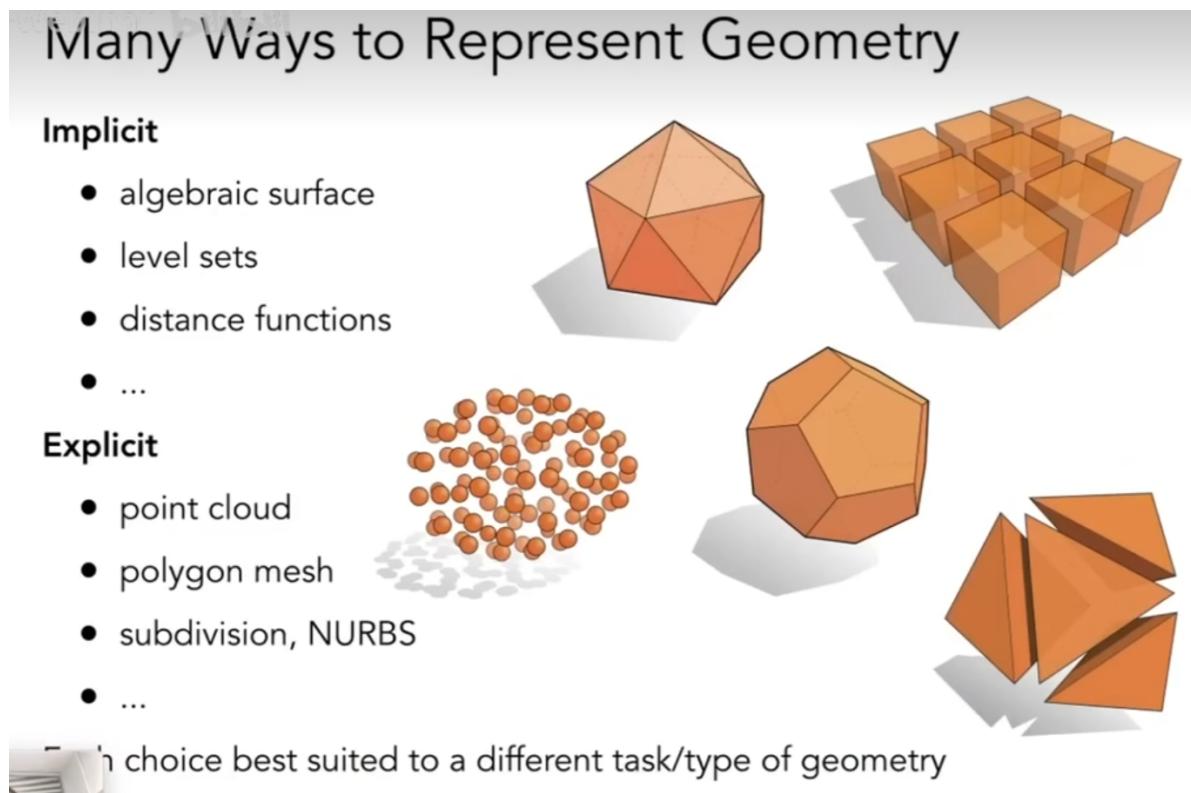


光栅化讲的比较好的：

(55条消息) 4、计算机图形学——光栅化、抗锯齿、画家算法和深度缓冲算法 (Z-buffer) Master Cui的博客 CSDN博客画家算法

10、geometry 几何

1、分类



implicit 隐式

explicit 显式

implicit 隐式

表示一定的关系（式子），并不给你直接的点。

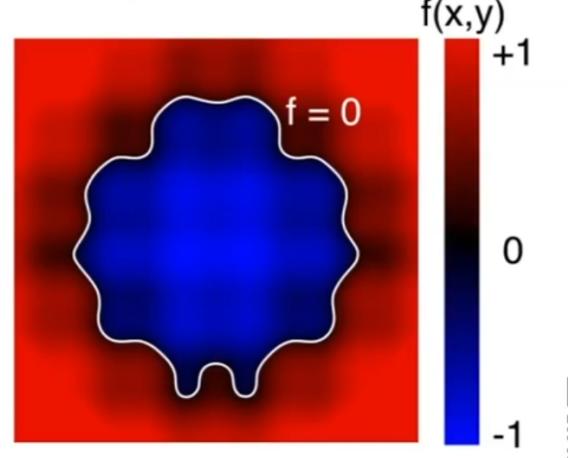
all points in 3D, where $x^2+y^2+z^2 = 1$

Based on classifying points

- Points satisfy some specified relationship

E.g. sphere: all points in 3D, where $x^2+y^2+z^2 = 1$

More generally, $f(x,y,z) = 0$



缺点：对于复杂的式子，难以看出其具体表示的图形。

优点：可以容易的判断某个点在不在这个表面上。

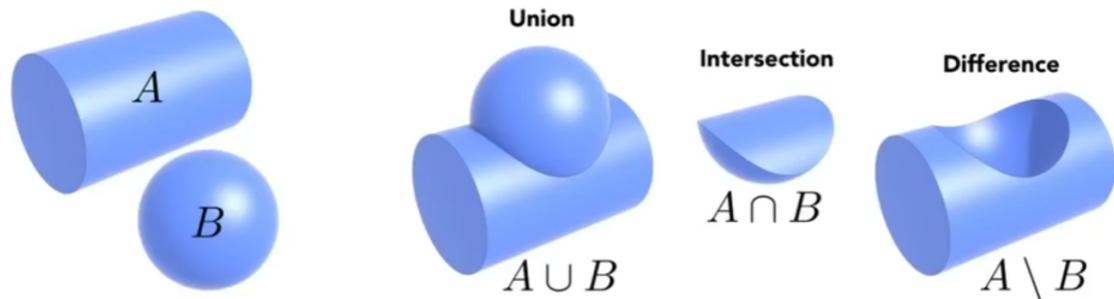
隐式额外的表示方法

1、CSG：

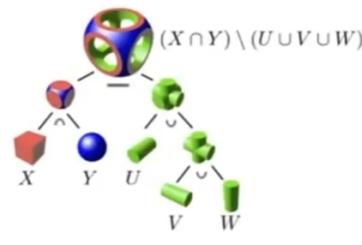
通过基本几何的基本运算，定义新的几何。

Constructive Solid Geometry (Implicit)

Combine implicit geometry via Boolean operations



Boolean expressions:



2、Distance Function

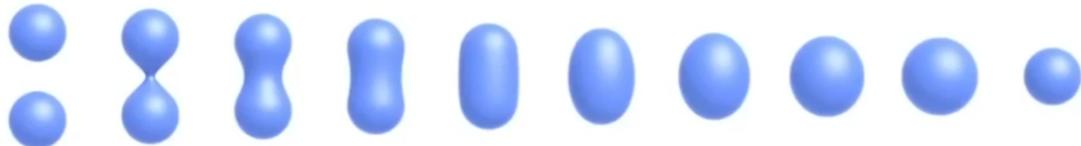
空间中任意一个点到某个图形的最小距离，外部的点是正的，内部的点是负的。

Distance Functions (Implicit)

Instead of Booleans, gradually blend surfaces together using

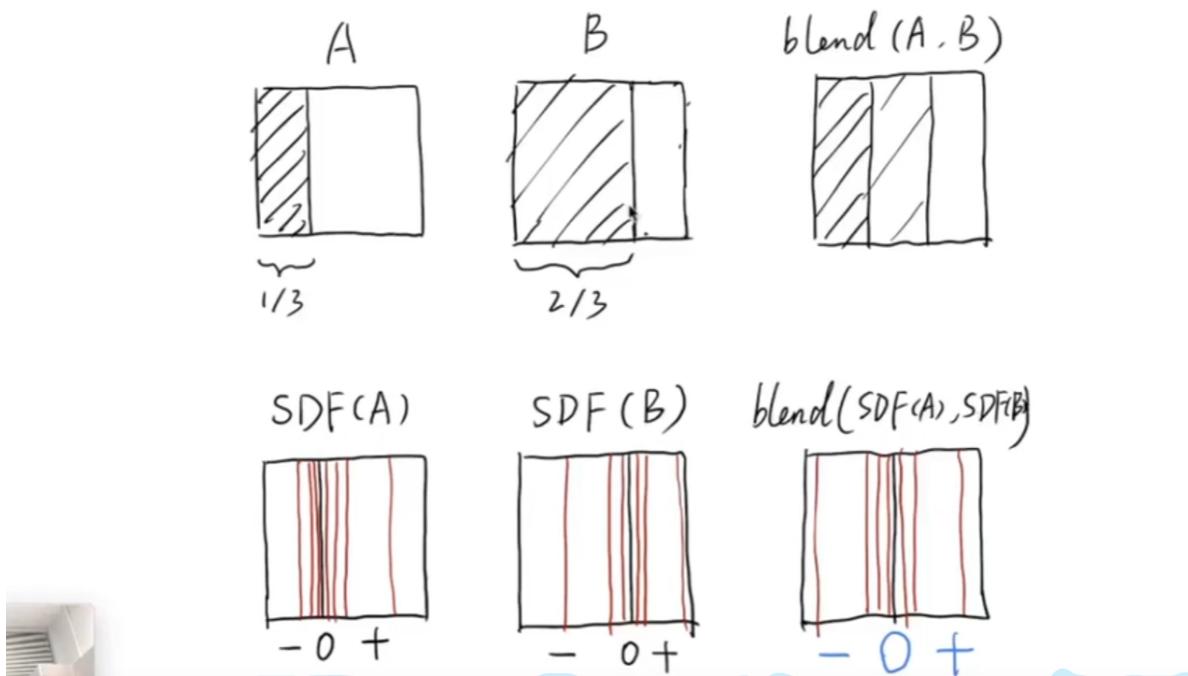
Distance functions:

giving minimum distance (could be **signed** distance)
from anywhere to object



Distance Functions (Implicit)

An Example: Blending a moving boundary



这种方式多用于在融合，求中间态时。

距离函数的具体过程：

对于两个状态，分别求出空间中点到面的距离，距离面比较近的绝对值小，远的绝对值大，由正负之分。把两个状态的进行融合，因为两个状态的正负位置以及每个位置的大小是不同的，所以会形成新的0表面。回复融合的表面就是把0的位置找出来就ok

Fractals 分形

Exhibit self-similarity, detail at all scales
“Language” for describing natural phenomena
Hard to control shape!

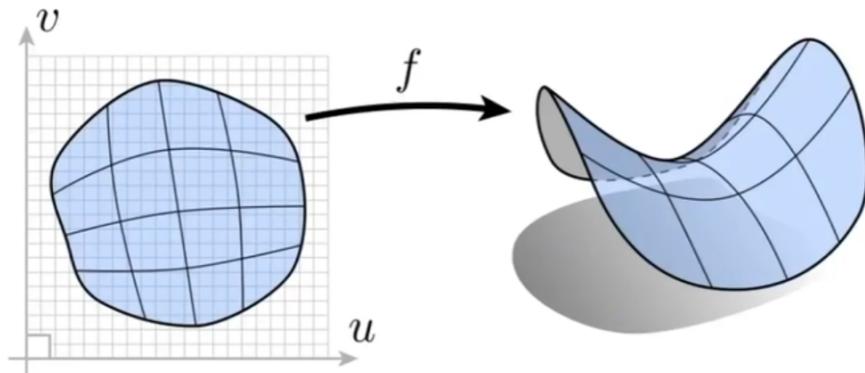


和cs里的递归类似。局部和整体很想，

explicit 显式

All points are **given directly or via parameter mapping**

Generally: $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3; (u, v) \mapsto (x, y, z)$



通过参数映射，遍历uv坐标，通过公式得出每个点的xyz坐标。

优点：显示形状容易

缺点：不方便判断点是否在面内，上，外。

11、几何2 曲线和曲面

curves and surfaces

显示表示的几种 (接上节课)

1、点云 pointCloud

用密集的点表示物体

2、多边形面 polygon mesh (最多)

用三角形面来表示物体

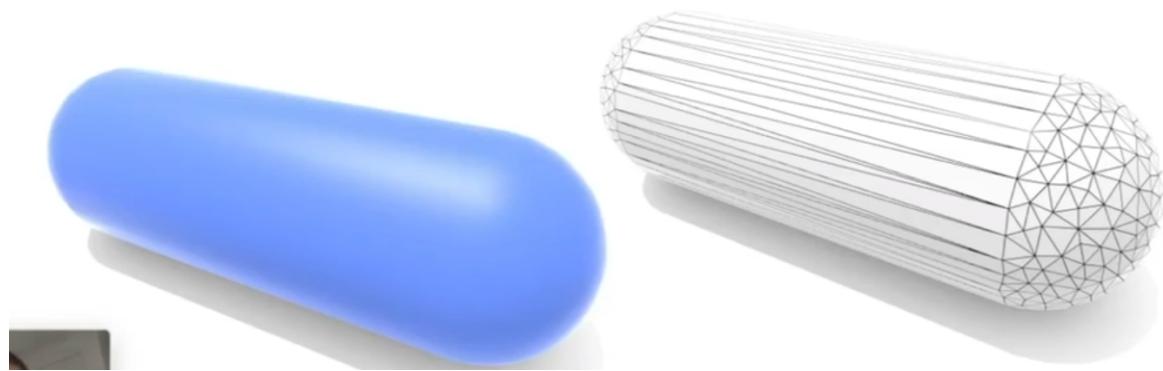
难点在于每个三角形之间的链接关系,

Polygon Mesh (Explicit)

Store vertices & polygons (often triangles or quads)

Easier to do processing / simulation, adaptive sampling

More complicated data structures



用这种方法表示的物体在计算机中如何表示的呢》下图

The Wavefront Object File (.obj) Format

Commonly used in Graphics research

Just a text file that specifies vertices, normals, texture coordinates **and their connectivities**



```
1 # This is a comment
2
3 v 1.000000 -1.000000 -1.000000
4 v 1.000000 -1.000000 1.000000
5 v -1.000000 -1.000000 1.000000
6 v -1.000000 -1.000000 -1.000000
7 v 1.000000 1.000000 -1.000000
8 v 0.999999 1.000000 1.000001
9 v -1.000000 1.000000 1.000000
10 v -1.000000 1.000000 -1.000000
11
12 vt 0.748573 0.750412
13 vt 0.749279 0.501284
14 vt 0.999110 0.501077
15 vt 0.999455 0.750380
16 vt 0.250471 0.500702
17 vt 0.249682 0.749677
18 vt 0.001085 0.750380
19 vt 0.001517 0.499994
20 vt 0.499422 0.500239
21 vt 0.500149 0.750166
22 vt 0.748355 0.998230
23 vt 0.500193 0.998728
24 vt 0.498993 0.250415
25 vt 0.748953 0.250920
26
27 vn 0.000000 0.000000 -1.000000
28 vn -1.000000 -0.000000 -0.000000
29 vn -0.000000 -0.000000 1.000000
30 vn -0.000001 0.000000 1.000000
31 vn 1.000000 -0.000000 0.000000
32 vn 1.000000 0.000000 0.000001
33 vn 0.000000 1.000000 -0.000000
34 vn -0.000000 -1.000000 0.000000
35
36 f 5/1/1 1/2/1 4/3/1
37 f 5/1/1 4/3/1 8/4/1
38 f 3/5/2 7/6/2 8/7/2
39 f 3/5/2 8/7/2 4/8/2
40 f 2/9/3 6/10/3 3/5/3
41 f 6/10/4 7/6/4 3/5/4
42 f 1/2/5 5/1/5 2/9/5
43 f 5/1/6 6/10/6 2/9/6
44 f 5/1/7 8/11/7 6/10/7
45 f 8/11/7 7/12/7 6/10/7
46 f 1/2/8 2/9/8 3/13/8
47 f 1/2/8 3/13/8 4/14/8
```

例如一个立方体：

有8个顶点，v

6个面，（用面的法线表示）vn

每个面有4个点，涉及到纹理坐标vt

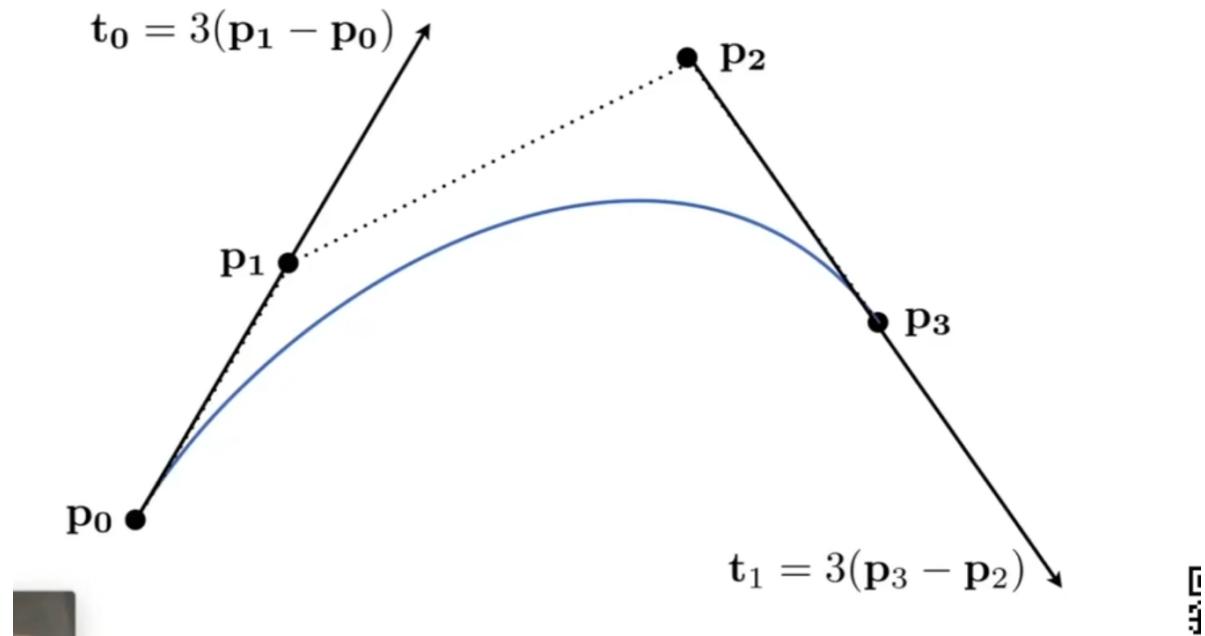
f表示三者的关系

CUvers 曲线

贝塞尔曲线 Bezier

显示的表示方法

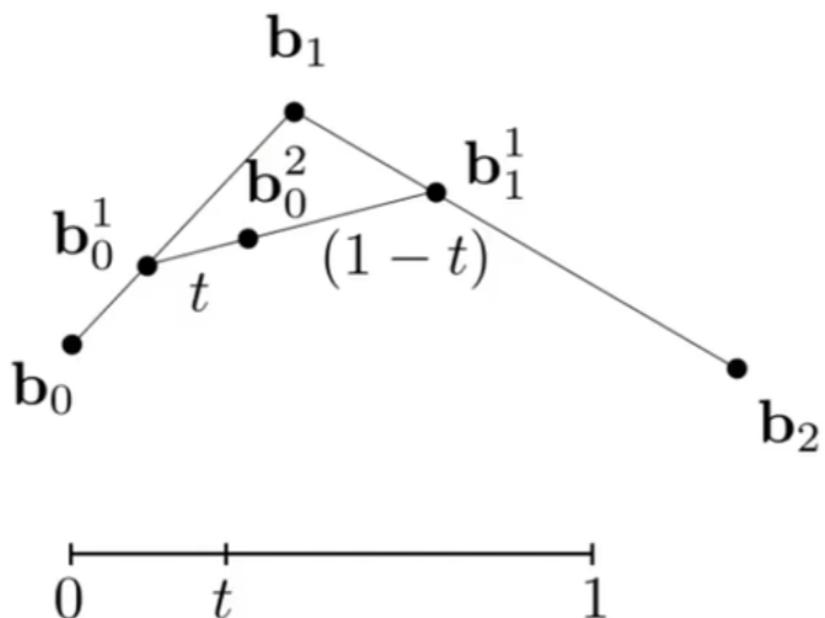
用一系列控制点表示曲线。

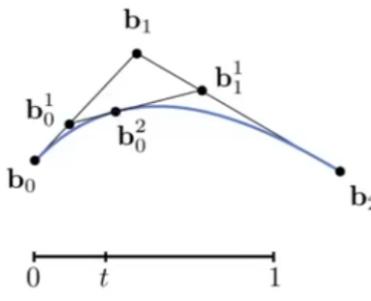


如何画贝塞尔曲线?

拿一个二次的贝塞尔曲线距离:

Repeat recursively





$$\mathbf{b}_0^1(t) = (1-t)\mathbf{b}_0 + t\mathbf{b}_1$$

$$\mathbf{b}_1^1(t) = (1-t)\mathbf{b}_1 + t\mathbf{b}_2$$

$$\mathbf{b}_0^2(t) = (1-t)\mathbf{b}_0^1 + t\mathbf{b}_1^1$$

$$\mathbf{b}_0^2(t) = (1-t)^2\mathbf{b}_0 + 2t(1-t)\mathbf{b}_1 + t^2\mathbf{b}_2$$

从时间0到1，求出t的位置即可

加入t为三分之一时刻，在两条线段都找出三分之一的位置，连线，形成的线的三分之一位置，即为此时的t点。

算法的过程

时间t从0->1,

在每条线段找到位置为t的点，连线。x线段会连成x-1, x-1连成x-2, 知道连成一个线段，找到对应的点即可。

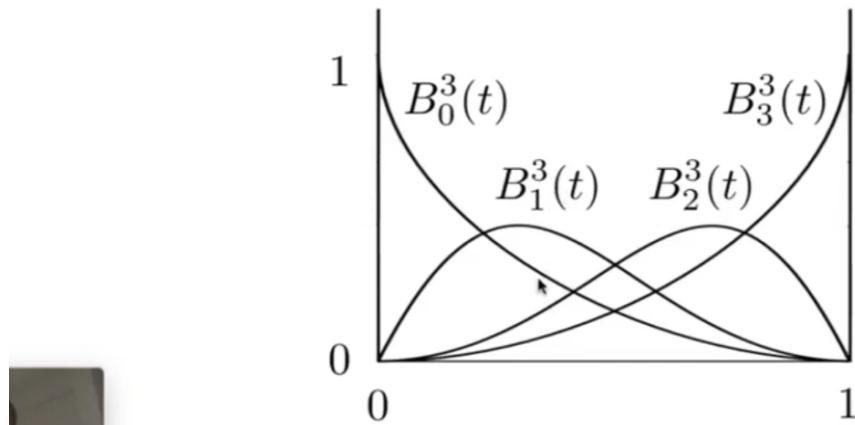
算法的公式：伯恩斯坦多项式

Bernstein Polynomials

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$



Sergei N. Bernstein
1880 – 1968



贝塞尔曲线性质：

1、规定起点0, 终点1.

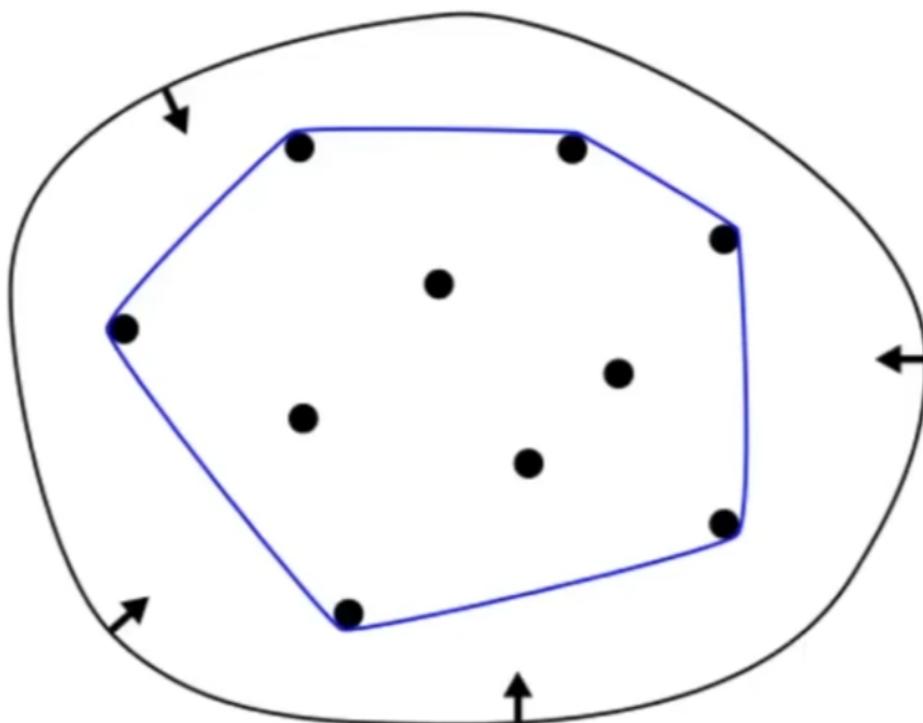
2、三次贝塞尔曲线：起点切线，终点切线是和这个曲线的阶数有关的，（四个点就是三阶）

3、在做仿射变换时得到的贝塞尔曲线不变（投影就不行）

4、凸包性质：画出来的曲线在几个点的凸包内

凸包：

可以包围形体最小的凸多边形



[from Wikipedia]

分段贝塞尔

问题映入：当点很多的时候，计算复杂而且不容易控制。而且一个点动了其他点都要动。

通常用四个控制点来决定一个别赛尔曲线，三次贝塞尔曲线那个切线的性质，

几个连续：

c0连续：每段的起始点一致

c1连续：每段的起始位置的切线连续

其他曲线

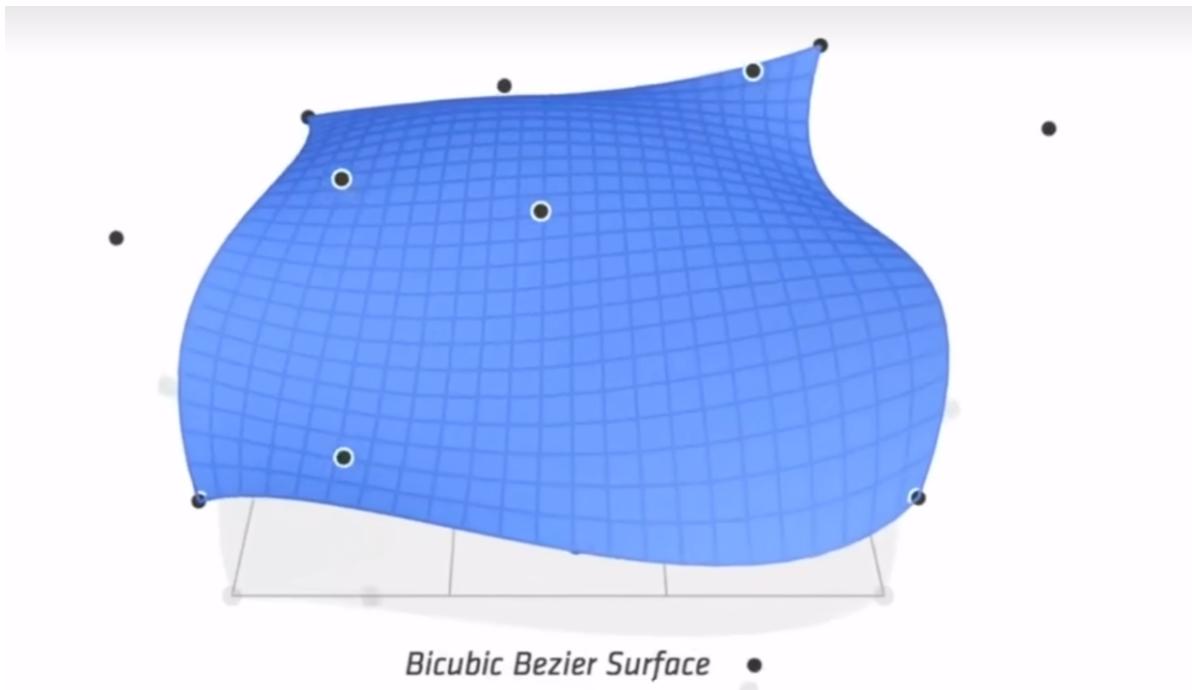
splines 样条

B-splines。 . . .

curfaces曲面

贝塞尔曲面：

不同的贝塞尔曲线形成的面



12、几何3

曲面细分 subdivision

在原来的基础分出更多的三角形，然后稍微改变一些形状。

Loop细分

在原来的基础分出更多的三角形，然后稍微改变一些形状。

新旧顶点进行不同区别的细分。

- Split each triangle into four



- Assign new vertex positions according to weights
 - New / old vertices updated differently

如何调整新旧顶点的位置 (关键):

Loop细分

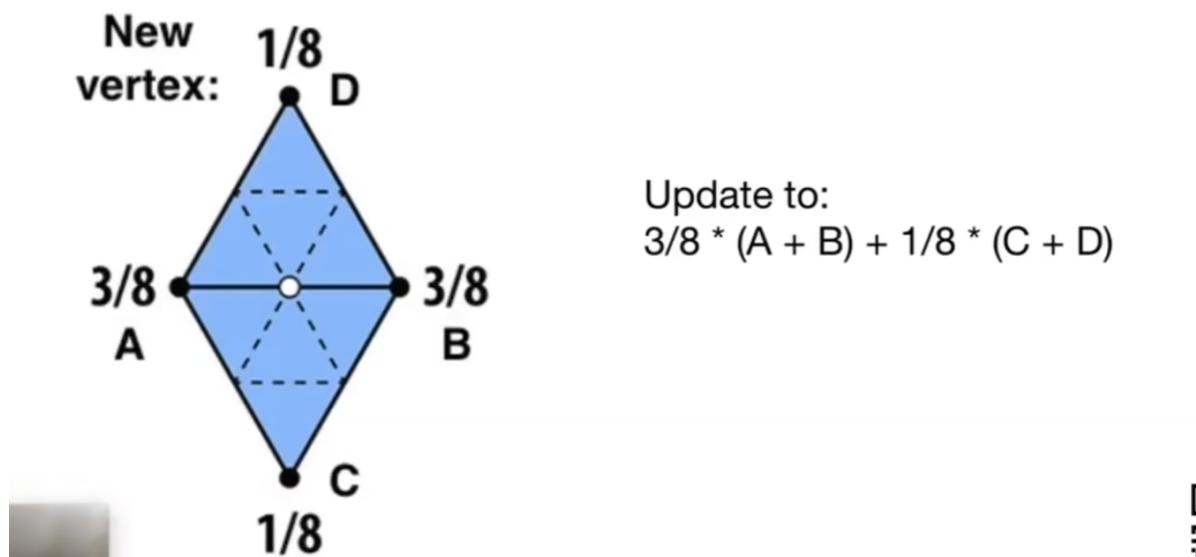
先细分在调整

(这里的Loop不是循环的意思，是人名)

新顶点：

Loop Subdivision — Update

For new vertices:



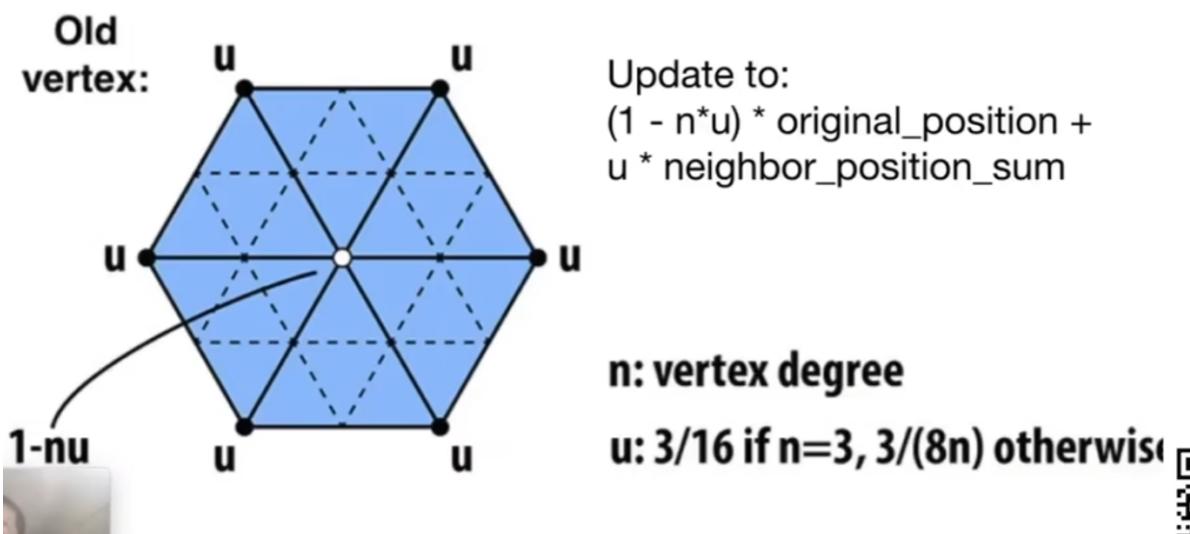
对于每个新的点，认为他所在的边恒为两个三角形的临边，具体位置计算公式：

Update to:

$$3/8 * (A + B) + 1/8 * (C + D)$$

|旧顶点：

For old vertices (e.g. degree 6 vertices here):



|旧顶点的细分:

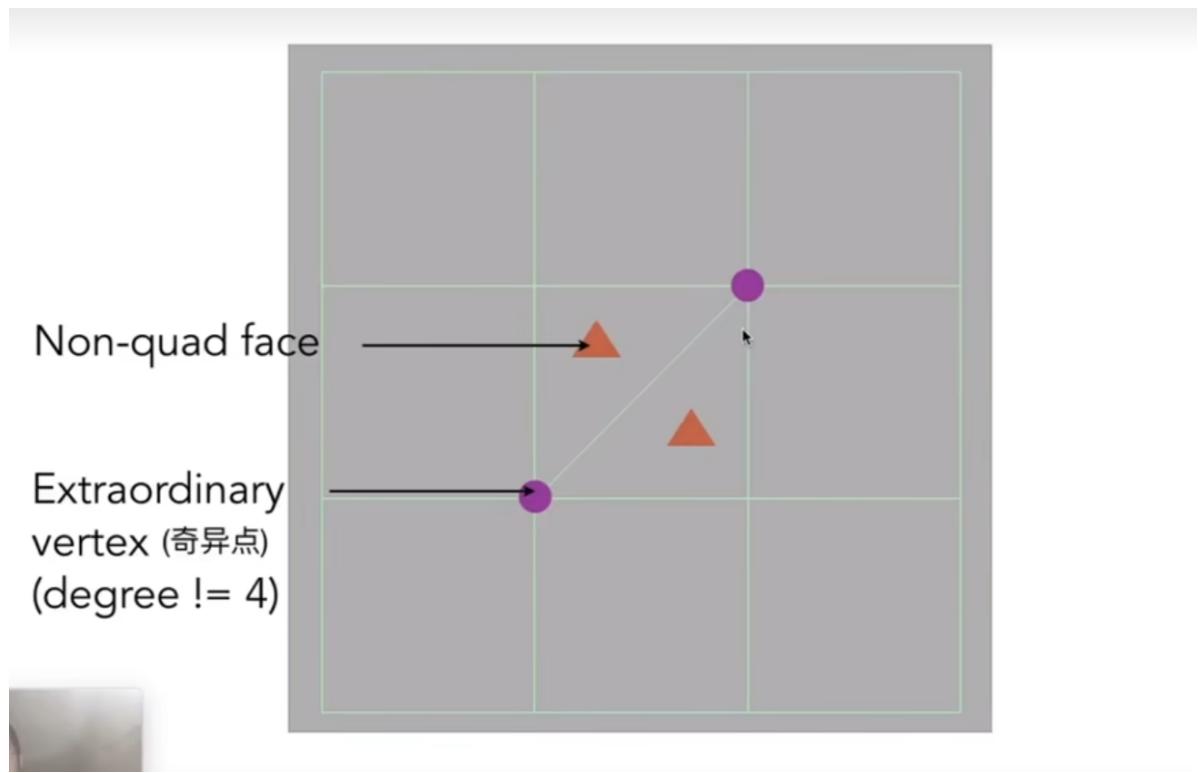
Update to:
 $(1 - n*u) * \text{original_position} + u * \text{neighbor_position_sum}$

n : vertex degree

u : $3/16$ if $n=3$, $3/(8n)$ otherwise

Catmull-Clark细分

(人名)



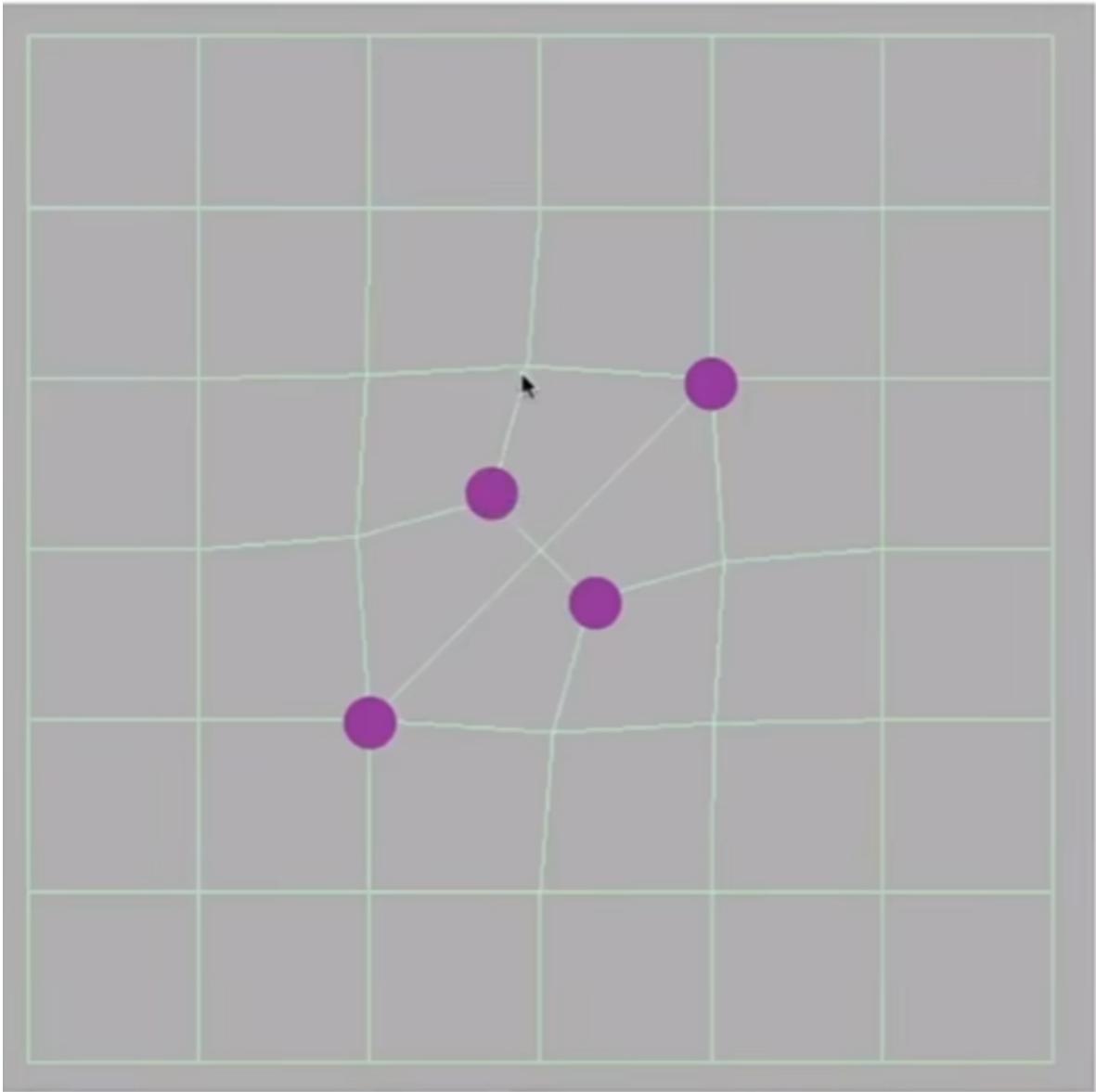
几个概念：

非四边形面：不是四边形的面

奇异点：度不是4的点

细分方法：

每条边取个中点，每个面也取个中点然后连接，如下图

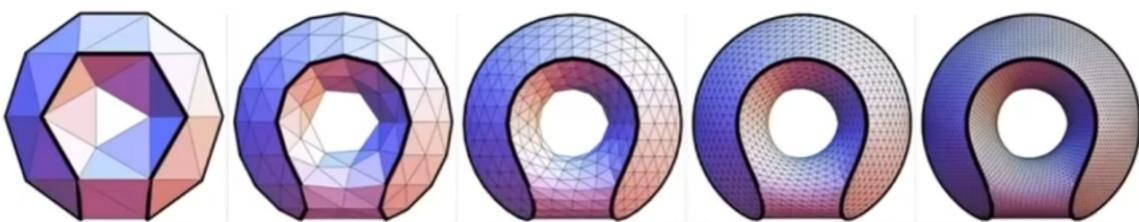


一次细分之后：

三角面消失，增加了奇异点，全部都是四边形面。原先奇异点的度数没变。

两种细分的对比

Loop with Sharp Creases



Catmull-Clark with Sharp Creases



Figure from: Hakenberg et al. Volume Enclosed by Subdivision Surfaces with Sharp Creases

loop细分仅用于三角形面，Catmull-clark用于各种形状的细分

曲面简化

Mesh Simplification

边坍缩方法：

Collapsing An Edge

把一条边的两个顶点缩成一个点。

坍缩完后如何减小误差？ 使用 二次误差度量

方法：找一个点，到原本相连的面的距离的平方和最小。

坍缩哪些边？

就是对每条边进行坍缩（打个分数），看看造成误差最小的那个。

问题：对某个边进行坍缩的时候，会影响到其他的边（的评分）。

解决：采用优先队列，每次去影响最小的边进行坍缩，然后更新他所影响的边，然后再去最小。

对于一个物体进行简化，坍缩的多的一般是较平整的面。

补充：光栅化过程中的阴影。

对于阴影，不要想成是某个物体投影出来的，而是想成这个区域是否可以被光源看到。

shadow mapping

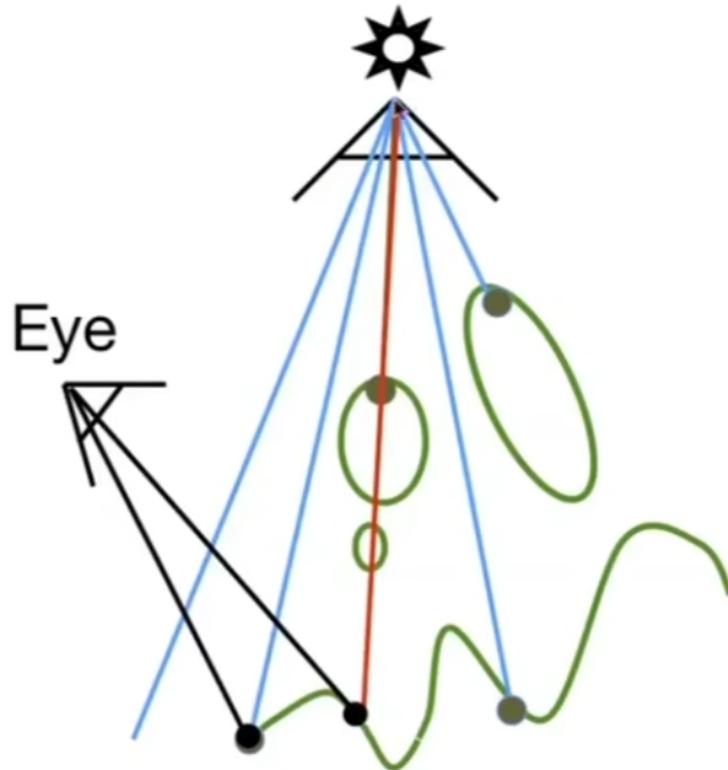
利用 相机，光源可否看到来处理

过程：

1、把相机移到点光源哪里，进行光栅化，看看光源可以看到什么，记录看的的点的深度值，不许要着色

2、从真正的眼睛出发，将看到的点像光源出做投影，对比现在观测的深度和之前记录的深度，如果相同，说明是可以（被相机和光源）看到的点。

如果深度不同，则说名光源看不到，则生成阴影。





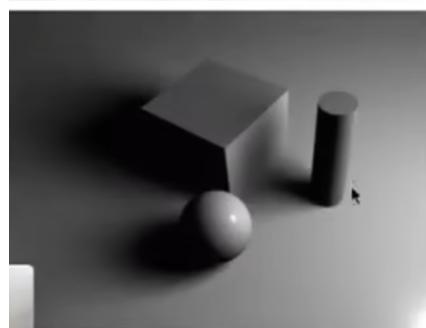
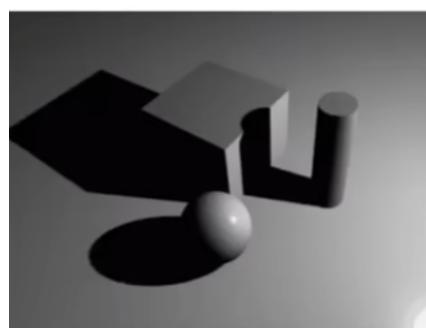
shadow mapping也存在小问题，比如上图中有些明显是可以被相机和眼睛看到的点，但是是黑色的。

因为距离是用浮点数表示的，而且测距离也会存在一些小的误差。

优化的方法：可以使我们的观测值要大于深度图（+小的偏差）就算可以看的。

还要问题就是要是测得深度图是保存在shadow map里，涉及到图的分辨率问题。

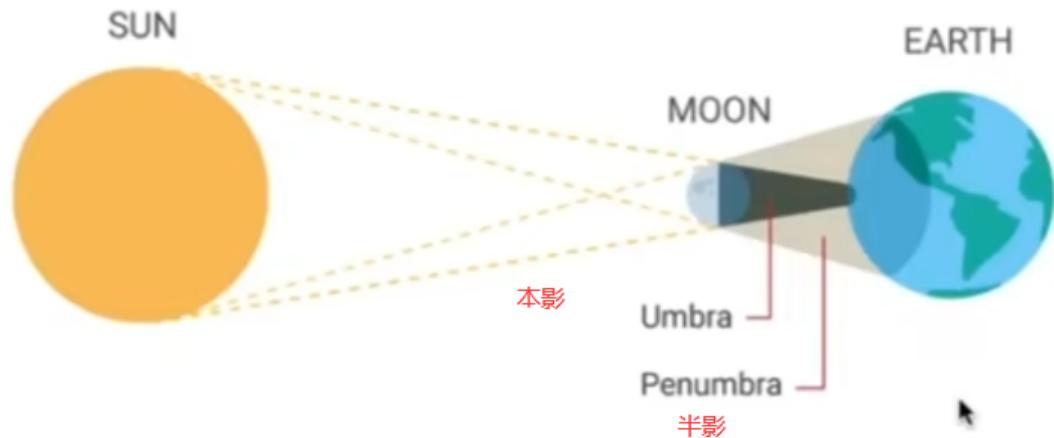
硬阴影和软阴影



RenderMan

硬阴影：边界很清晰。shadow mapping是硬阴影。（点光源是不会有关影的，应为点光源无大小。）

软阴影：是本影，半影，无阴影的一个过渡



13、光线追踪1

概括：

利用光的可逆性，从相机出发，看向一个物体的某个表面，那这个物体的表面反射的光可能来自光源的直接光照，也可能来自其他物体的反射。

不同物体的反射的光汇入一个像素，进而所有的像素进行成像。

涉及的问题：

1、传统的光追一般是指whited风格的光追，就是考虑了简单的反射和折射，但是没有对所有情况例如漫反射进行考虑。

2、定义了光线的方程表示，光线如何与隐式，显式物体判断相交，特别是显式物体，提出同MT算法优化和三角相交

但是和每个三角形判断太耗费，提出用（AABB）包围盒的方法

3、在包围盒内，又提出了“均匀的格子”“空间划分”“物体划分”“几种方法

4、具体的光追的实现--涉及到辐射度量学。

intensity, Irradiance, radiance的定义

BRDF用来求来自某个方向的power (Irradiance)，向某个方向反射时的power (randiance)。
反射方程，渲染方程。

蒙特卡洛积分来解渲染方程。解决：光线爆炸，递归终止，等问题 -->路径追踪

和光栅化成像方式不同。

光栅化的问题

在 阴影，类似于镜面反射，间接光照这几块做的不太好。

- Rasterization couldn't handle **global** effects well
 - (Soft) shadows
 - And especially when the light bounces more than once



Soft shadows



Glossy reflection



Indirect illumination

之前在光栅化的时候，着色只考虑一次反射的情况，对于多次反射处理的不好

- Rasterization is fast, but quality is relatively low



Buggy, from PlayerUnknown's Battlegrounds (PC game)

光栅化是快速的，但是质量要求较低。

对比于光线追踪，速度慢，但是质量高。多用于电影

- Ray tracing is accurate, but is **very slow**
 - Rasterization: **real-time**, ray tracing: **offline**
 - ~10K CPU core hours to render **one frame** in production



Zootopia, Disney Animation

定义光线追踪

Three ideas about light rays

1. Light travels in straight lines (though this is wrong)
2. Light rays do not “collide” with each other if they cross (though this is still wrong)
3. Light rays travel from the light sources to the eye (but the physics is invariant under path reversal - reciprocity).

1、光线直线传播

2、光线交叉不会碰撞

3、光线经过反射最终到达眼睛

光追利用了光的可逆性

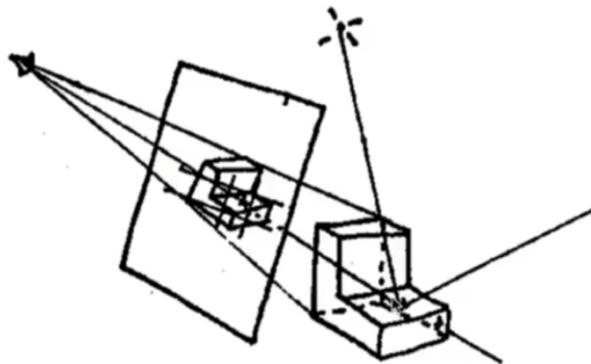
说明：认为光为点光源，折射为完美折射。

光线投射

有点类似于上面的shadow mapping的那个球阴影的步骤

Appel 1968 - Ray casting

1. Generate an image by **casting one ray per pixel**
2. Check for shadows by **sending a ray to the light**



最简单的光追就是从eye出发，看向场景，从看到的点像光源做投射，看看是否在阴影里面，然后返回进行着色

也就是反射一次，可想而知，这种方法显然是粗糙的，因为一些折射效果这类的就无法显示。

whitted 模型

考虑了光的多次反射和折射。将多次反射或者折射后到达的点的着色都计算会最初的那个像素，当然也会考虑光的衰减。

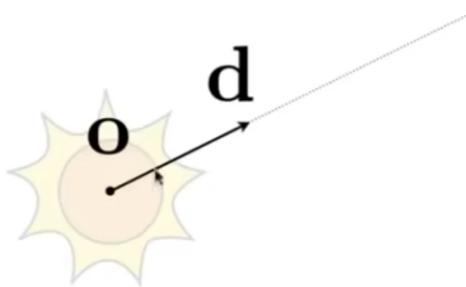
求光线和物体表面的交点

定义光线：

起点和一个方向

Ray is defined by its origin and a direction vector

Example:



Ray equation:

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \quad 0 \leq t < \infty$$

光线和球做交点

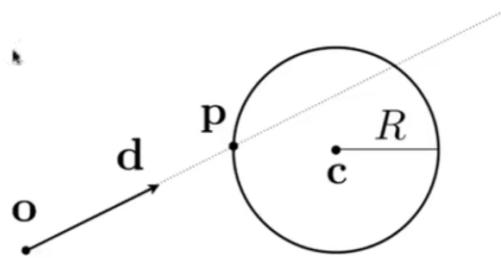
Ray Intersection With Sphere

Ray: $\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}, 0 \leq t < \infty$

Sphere: $\mathbf{p} : (\mathbf{p} - \mathbf{c})^2 - R^2 = 0$

What is an intersection?

The intersection \mathbf{p} must satisfy both ray equation and sphere equation



Solve for intersection:

$$(\mathbf{o} + t \mathbf{d} - \mathbf{c})^2 - R^2 = 0$$

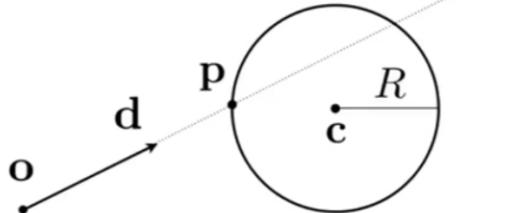
$$at^2 + bt + c = 0, \text{ where}$$

$$a = \mathbf{d} \cdot \mathbf{d}$$

$$b = 2(\mathbf{o} - \mathbf{c}) \cdot \mathbf{d}$$

$$c = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



求交点就是结合两个函数即可，把 $\mathbf{o}+t\mathbf{d}$ 替换 球公式中的 \mathbf{p} ， 带入后唯一的未知量就是那个 t 。

解出来就是有几重情况

由与球的交点得出与 隐式表示的物体的交点

Ray: $\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}, 0 \leq t < \infty$

General implicit surface: $\mathbf{p} : f(\mathbf{p}) = 0$

Substitute ray equation: $f(\mathbf{o} + t \mathbf{d}) = 0$

就是代入然后求 t , t 是实数, 正数。

光线与显式表示的物体的交点 (与三角形)

有个属性：就是判断光线射在封闭物体内还是外，如果是奇数个交点，就是在内部。

如何计算光线和物体交点：

对物体的每一个三角形，判断是否有交点，然后找到最近的那个t。这样很慢
具体的优化后面讲。

如何计算光线和三角形的交点：

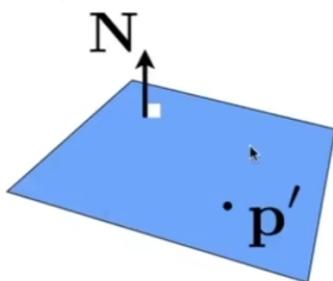
1、光线和三角形所在平面的交点，2、然后看看交点是否在三角形内。

定义平面：一个法线和一个点

平面内的点： $(p - p') \cdot N = 0$

Plane is defined by normal vector and a point on plane

Example:



Plane Equation (if p satisfies it, then p is on the plane):

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0 \quad ax + by + cz + d = 0$$

求交点：

光线公式和平面公式结合，求出t

Ray equation:

$$\mathbf{r}(t) = \mathbf{o} + t \mathbf{d}, \quad 0 \leq t < \infty$$

Plane equation:

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0$$

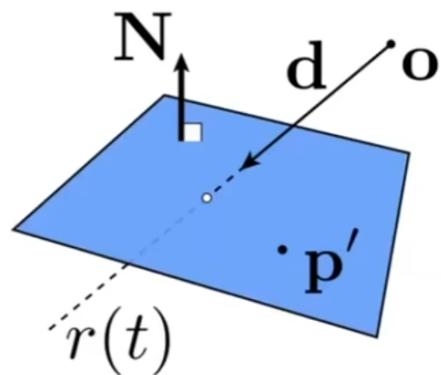
Solve for intersection

Set $\mathbf{p} = \mathbf{r}(t)$ and solve for t

$$(\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = (\mathbf{o} + t \mathbf{d} - \mathbf{p}') \cdot \mathbf{N} = 0$$

$$t = \frac{(\mathbf{p}' - \mathbf{o}) \cdot \mathbf{N}}{\mathbf{d} \cdot \mathbf{N}}$$

Check: $0 \leq t < \infty$



解出交点然后就判断是否在三角形内。

MT算法：

可以快算判断点是否在三角形内的方法

具体的方法就是如果光线的这个点在三角形内，拿他也可以写成三角形重心坐标表示的一种方法，又因为是三维空间，展开成三个方向的式子解出下图中的

t, b_1, b_2 . 看看 b 系数是不能为负数的。

A faster approach, giving barycentric coordinate directly

Derivation in the discussion section!

$$\vec{\mathbf{O}} + t \vec{\mathbf{D}} = (1 - b_1 - b_2) \vec{\mathbf{P}}_0 + b_1 \vec{\mathbf{P}}_1 + b_2 \vec{\mathbf{P}}_2$$

Where:

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{\mathbf{S}}_1 \cdot \vec{\mathbf{E}}_1} \begin{bmatrix} \vec{\mathbf{S}}_2 \cdot \vec{\mathbf{E}}_2 \\ \vec{\mathbf{S}}_1 \cdot \vec{\mathbf{S}} \\ \vec{\mathbf{S}}_2 \cdot \vec{\mathbf{D}} \end{bmatrix}$$

$$\vec{\mathbf{E}}_1 = \vec{\mathbf{P}}_1 - \vec{\mathbf{P}}_0$$

$$\vec{\mathbf{E}}_2 = \vec{\mathbf{P}}_2 - \vec{\mathbf{P}}_0$$

$$\vec{\mathbf{S}} = \vec{\mathbf{O}} - \vec{\mathbf{P}}_0$$

Recall: How to determine if the “intersection” is inside the triangle?

Hint:
($1 - b_1 - b_2$), b_1, b_2 are barycentric coordinates!

Cost = (1 div, 27 mul, 17 add)

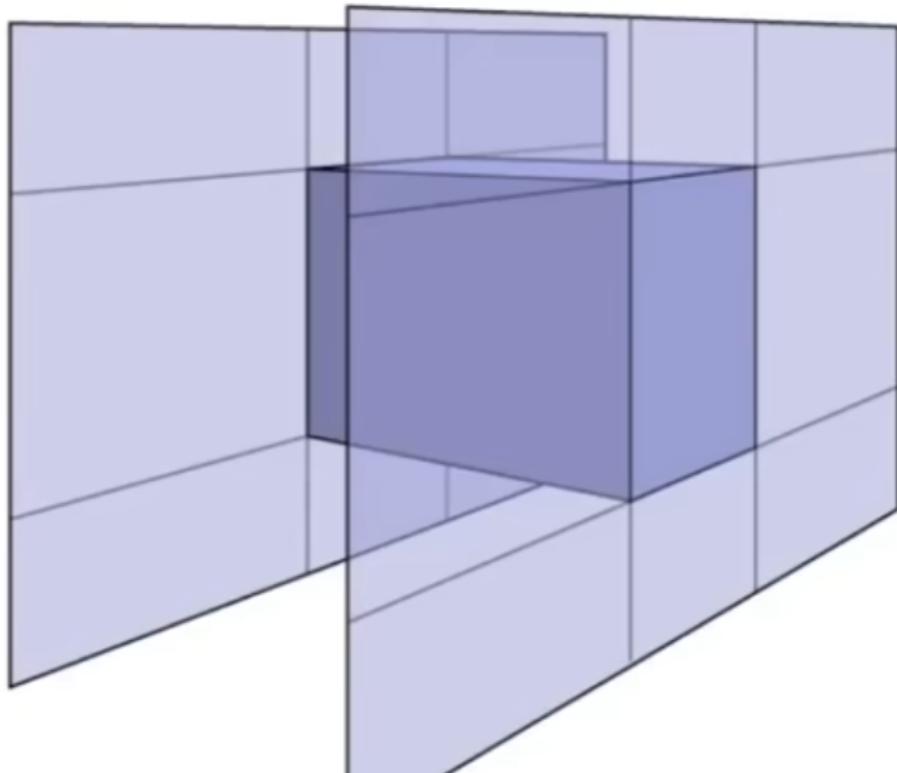
$$\vec{\mathbf{S}}_1 = \vec{\mathbf{D}} \times \vec{\mathbf{E}}_2$$

$$\vec{\mathbf{S}}_2 = \vec{\mathbf{S}} \times \vec{\mathbf{E}}_1$$

1、包围盒

就是把一个复杂的物体用简单的形状将其包围

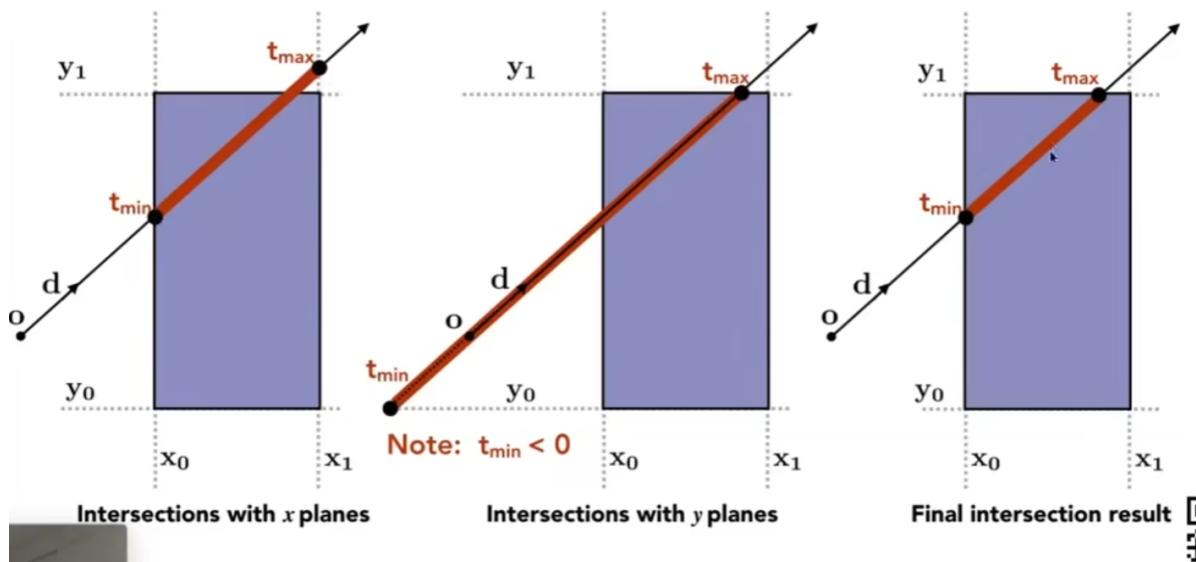
长方体：三个对面形成的交集



AABB包围盒：长方体的面是和某个平面平行

判断光线和包围盒是否相交

在二维中：



在二维中，包围盒由两对线来围城，用光线公式分别和每对求出交点，即算出四个点（可能有些点不合理），两对线段求交集算出真正的进出的两个点。

在三维中：

光线进入盒子：都进入三个对子面。

光线出盒子：光线出去任意一个对子面。

- For the 3D box, $t_{\text{enter}} = \max\{t_{\min}\}$, $t_{\text{exit}} = \min\{t_{\max}\}$

所以：

光线进入盒子：最晚进入某个对面的时间， T_{enter}

光线出去盒子：最近出去某个盒子、 T_{exit}

$T_{\text{enter}} < T_{\text{exit}}$ 说明光线是在盒子中带过一段时间

如果 $T_{\text{exit}} < 0$: 说明盒子在光的背后 无交点！

如果 $T_{\text{exit}} \geq 0 \&& T_{\text{enter}} < 0$: 光线起点就在盒子里面。一定有交点！

总结：光线和 AABB 盒子交点：满足如下：


$$- t_{\text{enter}} < t_{\text{exit}} \&& t_{\text{exit}} \geq 0$$

14、光线追踪2

上节讲了如何计算光线和AABB盒子求交，那如何利用AABB盒子来加速光线追踪？

均匀的格子

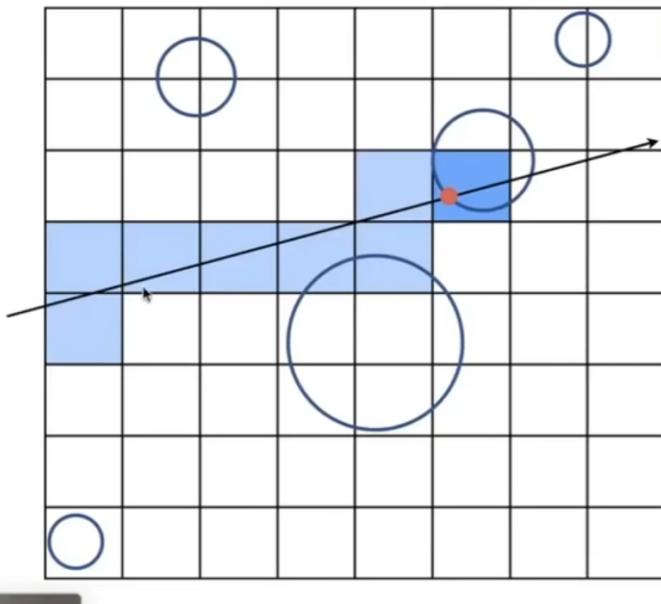
uniform grid

在AABB盒子中均匀的划分成几个格子，然后标记出那个格子有物体。

然后光线射入，和每个小格子做判断，看看这个格子是否有物体。

(这个认为判断光线和格子求交比光线和物体求交看很多。

而且，如何判断光线射入，和哪一个格子有交点：知道光线的大致方向，然后只要判断当前格子周围的那些格子即可)



Step through grid in ray traversal order

For each grid cell
Test intersection
with all objects
stored at that cell

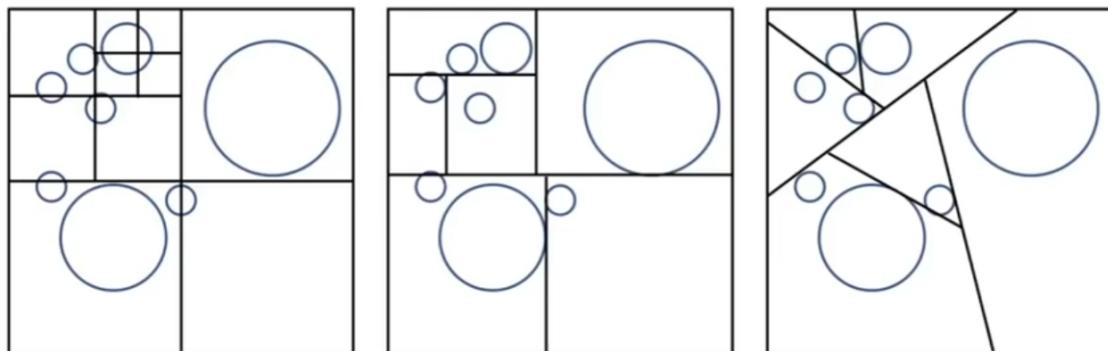
画格子的本质意向是让光线和格子判断求交，进而检测是否和物体有交点。比直接判断光线和AABB盒子内每一个物体有交点要快

还有就是格子的数量不应过于密集，

这种均匀格子方法不适用与场景分布不均匀，有些地方集中没有地方空旷。

空间划分

spatial partitions



Oct-Tree

KD-Tree

BSP-Tree

Oct Tree 八叉树

空间中把物体均匀的切成八块。每个子树在往下切成8块。

当子树空间内物体稀疏或者没有物体时，停止切割。

缺点：在二维叫四叉树，三维叫8叉，维数高就不容易了。

KD Tree

是每次只沿着一个轴划分一次，且水平，竖直划分支交替进行。

有二叉树的性质。

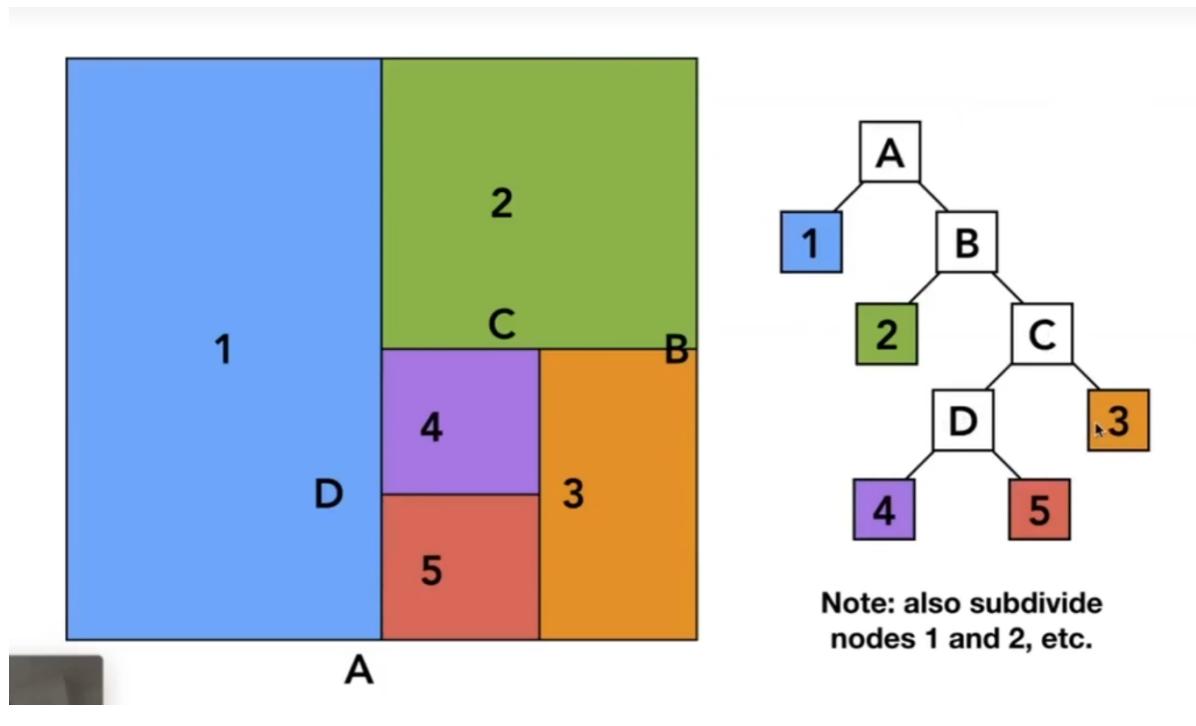
BSP Tree

和KD Tree的区别就是不是横平竖直来划分的。

在高维不好划分

KD-Tree

先建立KDTree，在做光线追踪



Internal nodes store

- split axis: x-, y-, or z-axis
- split position: coordinate of split plane along axis
- children: pointers to child nodes
- **No objects are stored in internal nodes**

Leaf nodes store

- list of objects

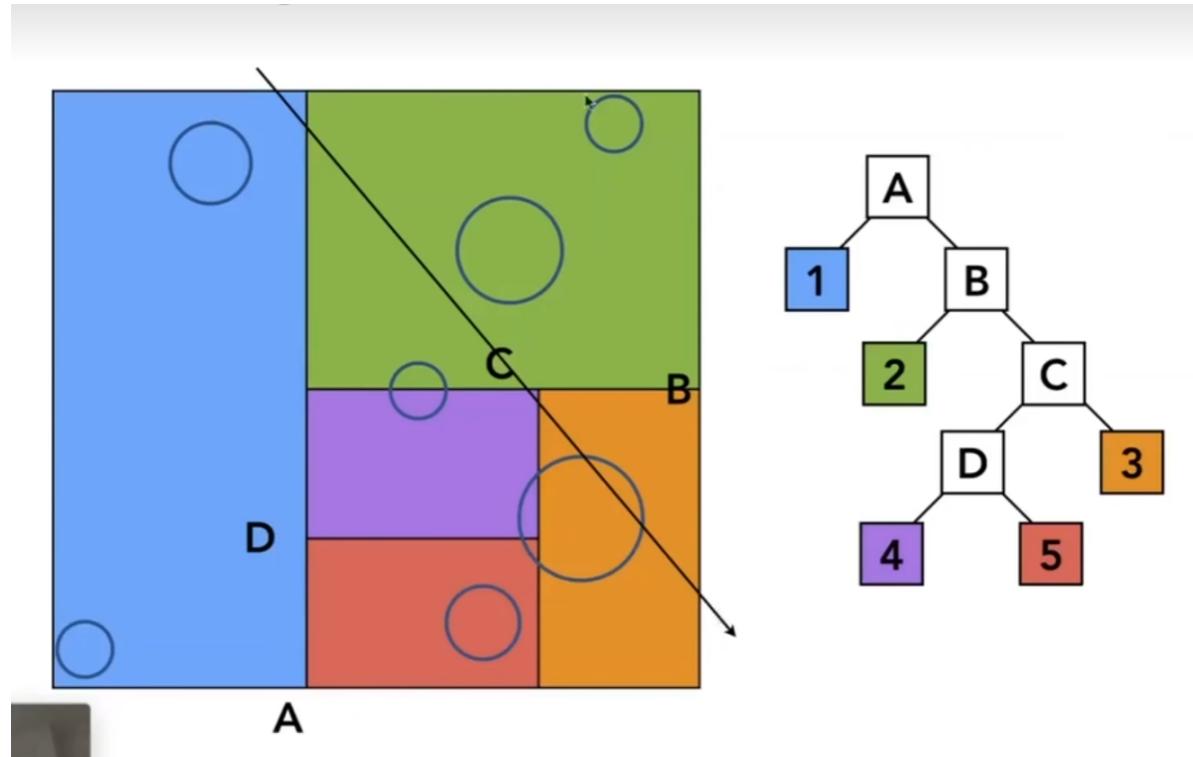
存储KDTree

按照x,y,z的顺序来划分

从哪里划分，可以时从中间切开

实际的三角形（物体）只存在叶子结点上。

过程：



首先是光线射过来，和AABB盒子有交点

然后从根节点开始，逐渐向下判断时候和划分区域有交点，直到到叶子结点，如果和区域有交点，则和区域内的物体判断是否有交点。

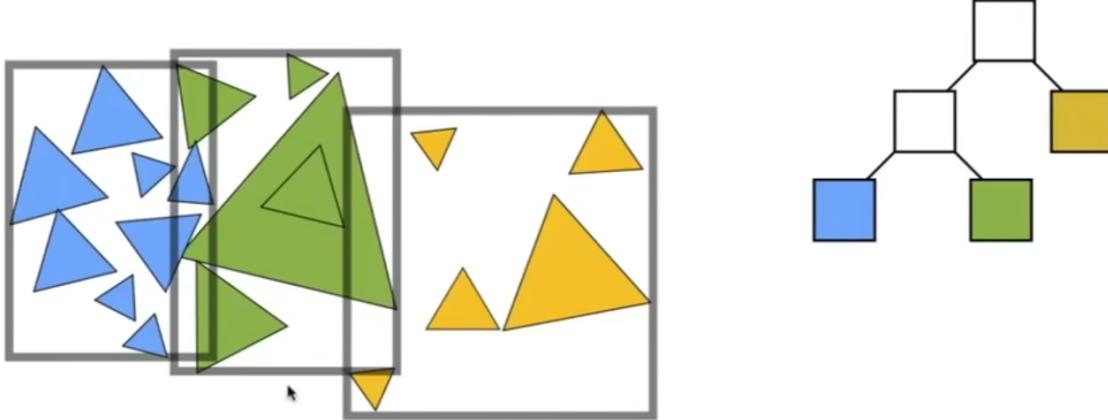
KD-Tree的问题：

- 1、包括AABB盒子，如果判断物体本身的三角形和我们划分的格子有交点（是否在我们的格子内），不好判断，有可能就是一个巨大的三角形把整个格子给包围住。
- 2、一个物体可能存在于对个格子里。

物体划分用的多！

可以解决上面KDTree的两个问题

Bounding Volume Hierarchy (BVH)



按照物体进行划分。

先把物体进行分开，然后求包围盒，不同的包围盒可能相交

这样一个物体只在一个盒子里

至于如何划分，有很多不同方法

划分：

划分的方向：可以时每次把最长的那部分分开，或者是按照xyz的顺序下去。

三角形分类：去中间的那个三角形，来左右分开

划分到区内数量少的时候停止划分

存储：

只在叶子结点存物体

```

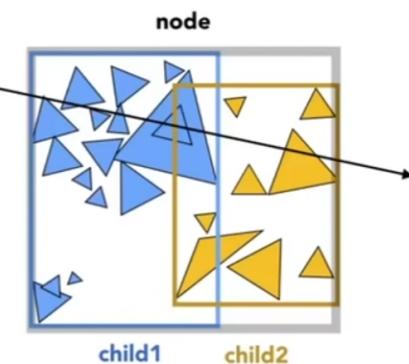
Intersect(Ray ray, BVH node) {
    if (ray misses node.bbox) return;

    if (node is a leaf node)
        test intersection with all objs;
        return closest intersection;

    hit1 = Intersect(ray, node.child1);
    hit2 = Intersect(ray, node.child2);

    return the closer of hit1, hit2;
}

```



具体的算法：

递归的从根节点往下算，

先判断光线是否和这个盒子有交点，然后再递归的去判断两个子盒子，直到到叶子结点，和物体进行判断。

以上是whitter风格的光追的一系列

从定义光线-->光线和隐式，显示的相交-->显示的相交无非是和三角形相交->利用AABB包围盒->AABB内的优化：均匀格子和空间划分和物体划分。

辐射度量学

Basic radiometry

概念

randiant energy 能量，单位是焦耳

randiant flux 单位时间的能量。 (功率，瓦特、流明)

flux：单位时间通过某个面的光的数量，表示光源射光的能力

Radient Intensity

能量/立体角 即每个立体角的能量

单位：瓦/角度 (sr) (下图)

立体角是个什么？？

弧度： $\Theta = l/r;$

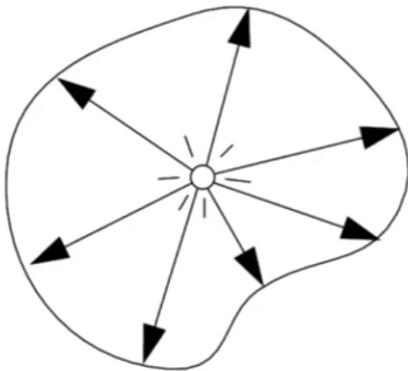
那 立体角： $= \text{面积} (\text{某个圆面}) / r^2 (\text{半径平方})$

立体角在整个球上是 4π 。

Definition: The radiant (luminous) intensity is the power per unit

solid angle (?) emitted by a point light source.

(立体角)

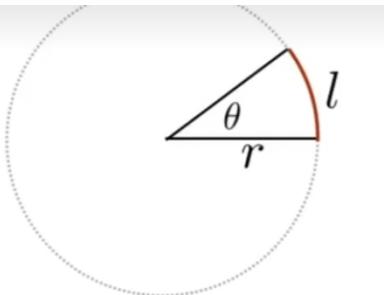


$$I(\omega) \equiv \frac{d\Phi}{d\omega}$$

$$\left[\frac{\text{W}}{\text{sr}} \right] \left[\frac{\text{lm}}{\text{sr}} = \text{cd} = \text{candela} \right]$$



The candela is one of the seven SI base units.



Angle: ratio of subtended arc length on circle to radius

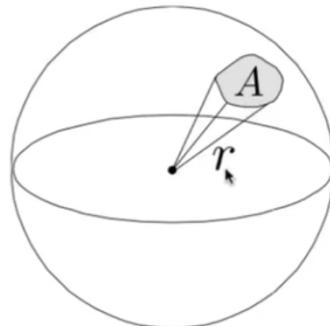
$$\bullet \theta = \frac{l}{r}$$

• Circle has 2π radians

Solid angle: ratio of subtended area on sphere to radius squared

$$\bullet \Omega = \frac{A}{r^2}$$

• Sphere has 4π steradians



对于一个比较均匀发光的物体，他的立体角为 4π 。

一直一流明，则其 intensity 为 流明/ 4π 。 (流明可以理解为那个瓦，大约1瓦是10流明)，即某一个方向上的强度。

Output: 815 lumens
(11W LED replacement
for 60W incandescent)

Radiant intensity?

Assume isotropic:
Intensity = $815 \text{ lumens} / 4\pi \text{ sr}$
= 65 candelas



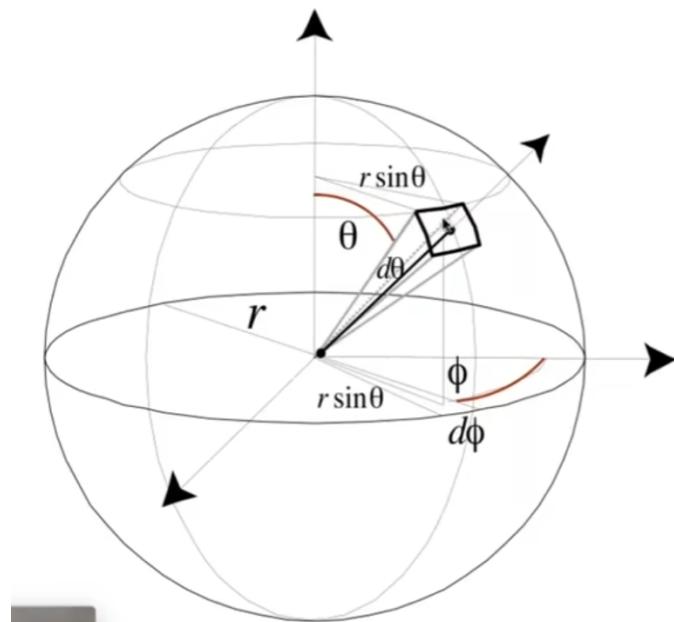
15、光线追踪3

补充：

关于立体角：

类似于求坐标， θ 为绕Z轴（竖直）转的角度， ϕ 为绕着这个竖直轴转了多少。

具体的计算看下图



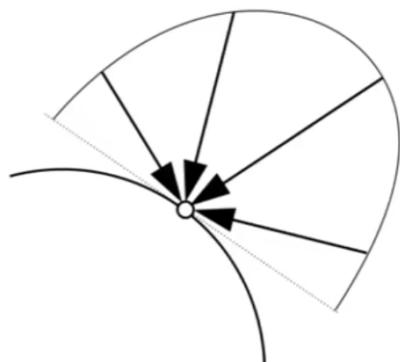
$$\begin{aligned}dA &= (r d\theta)(r \sin \theta d\phi) \\&= r^2 \sin \theta d\theta d\phi\end{aligned}$$

Irradiance

Definition: The irradiance is the power per (perpendicular/projected) unit area incident on a surface point.

$$E(\mathbf{x}) \equiv \frac{d\Phi(\mathbf{x})}{dA}$$

$$\left[\frac{\text{W}}{\text{m}^2} \right] \left[\frac{\text{lm}}{\text{m}^2} = \text{lux} \right]$$



是每个面积上对应的能量。

那个上节课的intensity类似，但是是单位立体角

Radiance

Definition: The radiance (luminance) is the power emitted, reflected, transmitted or received by a surface, **per unit solid angle, per projected unit area.**



$$L(p, \omega) \equiv \frac{d^2\Phi(p, \omega)}{d\omega dA \cos \theta} \quad \text{cos } \theta \text{ accounts for projected surface area}$$

$$\left[\frac{\text{W}}{\text{sr m}^2} \right] \left[\frac{\text{cd}}{\text{m}^2} = \frac{\text{lm}}{\text{sr m}^2} = \text{nit} \right]$$

定义：

在单位立体角，单位面积上的power (能量，流明)

So

- Radiance: Irradiance per solid angle
- Radiance: Intensity per projected unit area

radiance和Irradiance: 后者是接受来自四面八方的power，而前者是考虑某一个立体角的power。

Irradiance vs. Radiance

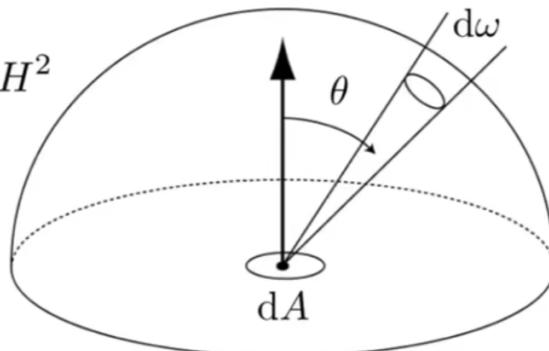
Irradiance: total power received by area dA

Radiance: power received by area dA from "direction" $d\omega$

$$dE(p, \omega) = L_i(p, \omega) \cos \theta d\omega$$

$$E(p) = \int_{H^2} L_i(p, \omega) \cos \theta d\omega$$

Unit Hemisphere: H^2



BRDF

Bidirectional Reflectance Distribution Function (BRDF)

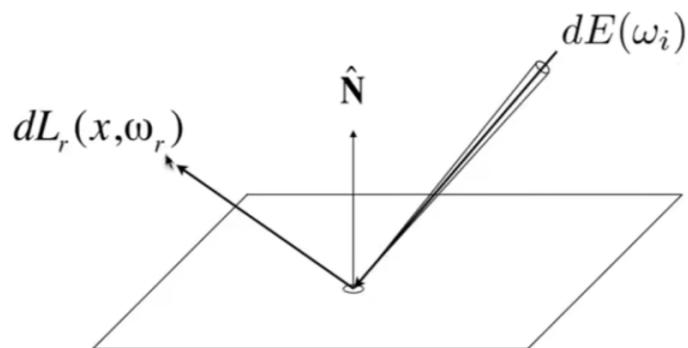
双向反射分布函数

讲述一个光线射进来，往不同的方向反射不同的能量。

可以看做物体表面先把射进来的power吸收，然后再向各个方向反射。

Radiance from direction ω_i turns into the power E that dA receives

Then power E will become the radiance to any other direction ω_o .



Differential irradiance incoming: $dE(\omega_i) = L(\omega_i) \cos \theta_i d\omega_i$

Differential radiance exiting (due to $dE(\omega_i)$): $dL_r(\omega_r)$

某一小块接受到的Irradiance, 如何反射到各个方向的radiance, 并分配power。有BRDF完成。

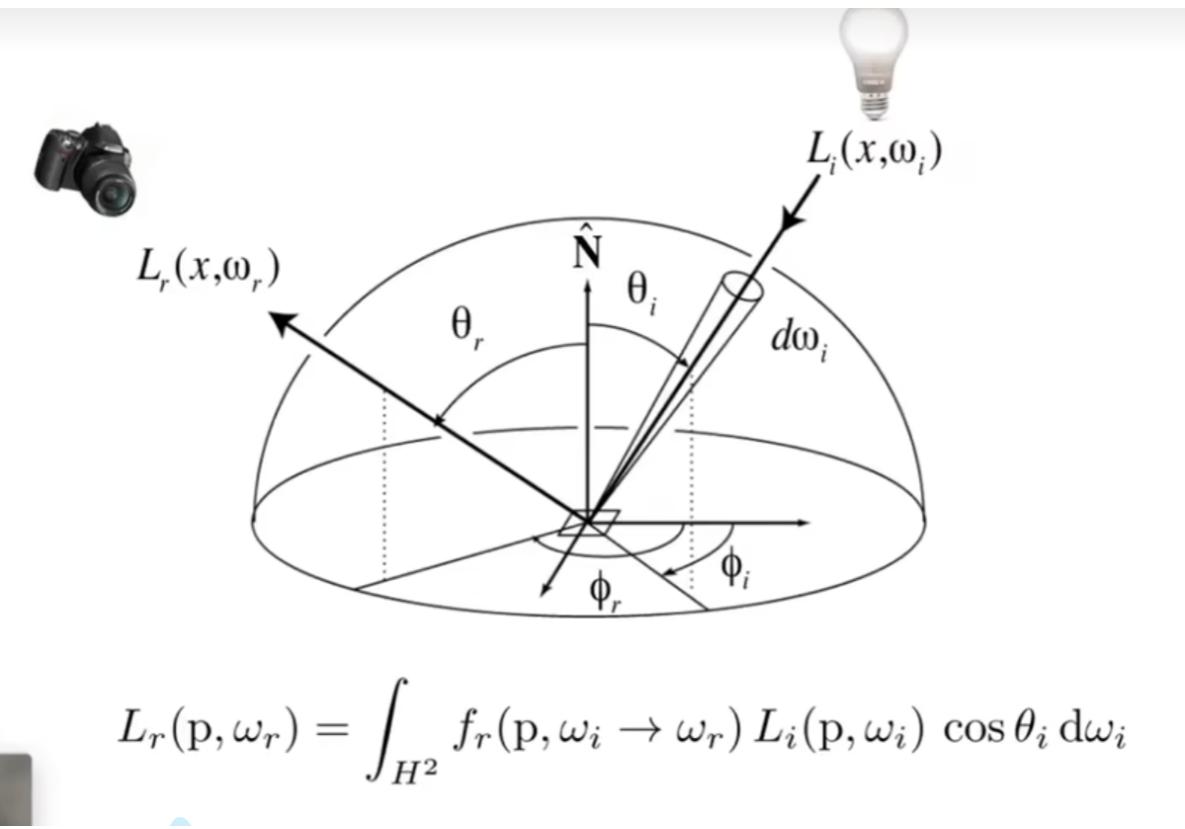
$$f_r(\omega_i \rightarrow \omega_r) = \frac{dL_r(\omega_r)}{dE_i(\omega_i)} = \frac{dL_r(\omega_r)}{L_i(\omega_i) \cos \theta_i d\omega_i} \left[\frac{1}{sr} \right]$$

一个方向上的randiance/表面的Irradiance

对于镜面反射, all power 分给反射方向。

漫反射, power是平均分配

反射方程



$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos \theta_i d\omega_i$$

对一个观测方向，把所有的入射光对此出射方向的贡献加和。

即可得到真正的场景。

难点：难点在于要考虑到所有入射光radiance的可能，如点光源的直接照射，其他物体的反射的光等等。

渲染方程

一个物体向某个观测点的光由 自身的发光 + 反射方程组成

渲染方程描述了光线的传播方式

The Rendering Equation

Re-write the reflection equation:

$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos \theta_i d\omega_i$$

by adding an Emission term to make it general!

The Rendering Equation

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i) d\omega_i$$

Note: now, we assume that all directions are pointing **outwards**!

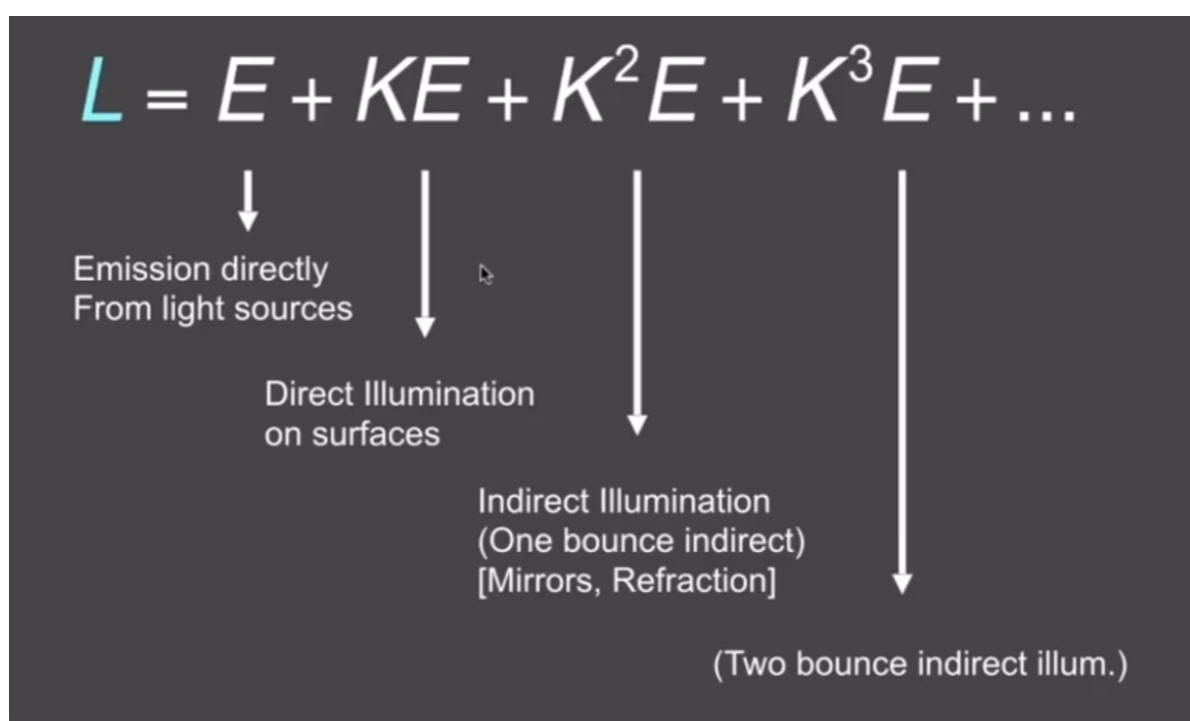
解渲染方程

简化：视为 表面向某个方向反射的光 = 自身的发光 + 其他表面反射的光*一个系数

$$\begin{aligned} L &= E + KL \\ IL - KL &= E \\ (I - K)L &= E \\ L &= (I - K)^{-1}E \end{aligned}$$

Binomial Theorem

$$\begin{aligned} L &= (I + K + K^2 + K^3 + \dots)E \\ L &= E + KE + K^2E + K^3E + \dots \end{aligned}$$



根据泰勒展开 会有不同系数项，对应了不同的反射次数

这些不同的反弹次数相加 = **全局光照**

在光栅化中，中的着色就是直接光照，只涉及到直接光照和光源自己，后面的就比较困难

但是光线追踪相对容易，

全局光照就是把直接光照，一次，两次。。。间接光照都算上。

概率论相关知识

用于解接下来的 渲染（全局光照）方程

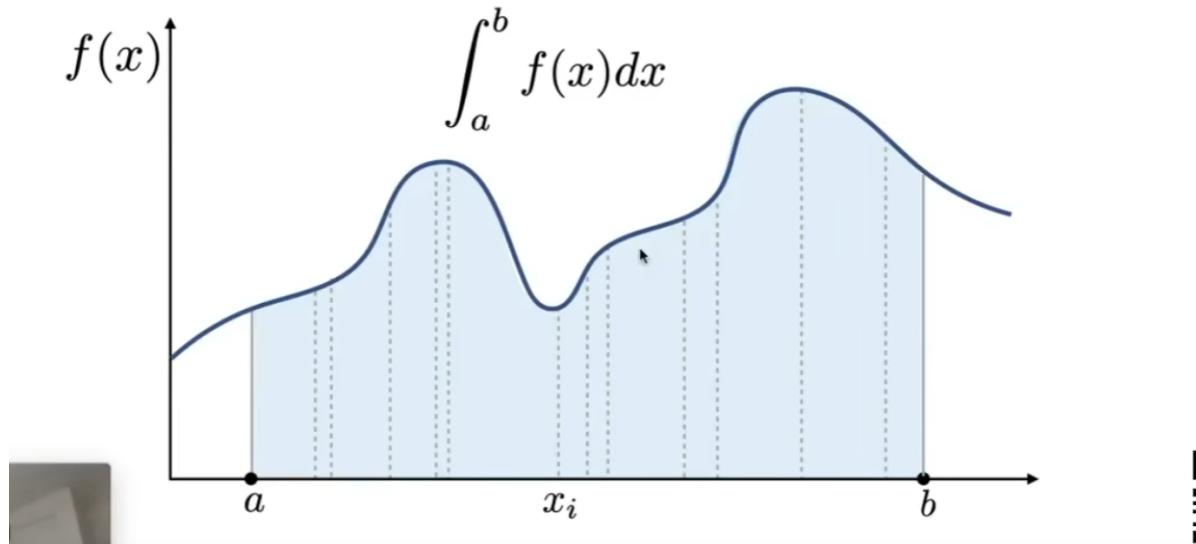
16、光线追踪4

蒙特卡洛积分

(在数学层面) 干什么的?

求定积分, 对于那些式子很复杂的。

类似于黎曼积分的办法, 在定义域内, 分成很多分长方形, 求那些长方形的面积。



Definite integral

$$\int_a^b f(x) dx$$

Random variable

$$X_i \sim p(x)$$

Monte Carlo estimator

$$F_N = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)}$$

这个 $p(x)$ 就是采样点的高度(概率), 如果是均匀采样, 那 $p(x)$ 恒为 $1/(b-a)$

特点:

采样点越多, 结果越准

积分域为 x ,

路径追踪

首先是光线追踪: (whited风格) 从相机出发向场景投射光线, 光线会发生发射, 折射, 每个点再向光源连线

只有达到打到镜面的面发生反射, 折射。达到发生漫反射的面停止, 忽略那些漫反射。

路径追踪解决whited风格光线追踪在物理上的一些错误,

whitted光线追踪使错误的，但是渲染方程式正确的

解渲染方程

使用蒙特卡洛方程来解

只考虑直接光照：

So, in general

$$L_o(p, \omega_o) = \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i) d\omega_i$$
$$\approx \frac{1}{N} \sum_{i=1}^N \frac{L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i)}{p(\omega_i)}$$

```
shade(p, wo)
    Randomly choose N directions wi~pdf
    Lo = 0.0
    For each wi
        Trace a ray r(p, wi)
        If ray r hit the light
            Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
    Return Lo
```

考虑间接光照：

射向表面的光源除了光源的直接光照，还有来自其他表面反射的光。处理这个反射的光，可以类似其他表面向这个方向的直接光照

```

shade(p, wo)

Randomly choose N directions wi~pdf
Lo = 0.0
For each wi
    Trace a ray r(p, wi)
    If ray r hit the light
        Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
    Else If ray r hit an object at q
        Lo += (1 / N) * shade(q, -wi) * f_r * cosine
        / pdf(wi)

Return Lo

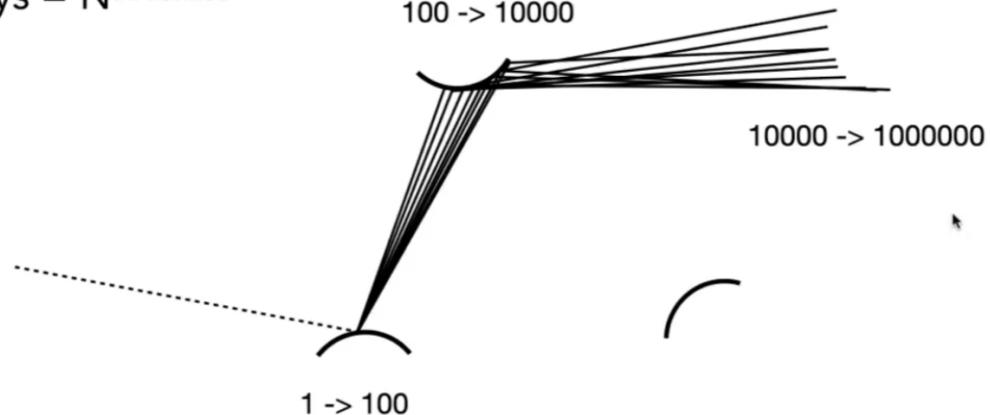
```

问题：

1、这样光线很多，会造成计算爆照。

Problem 1: Explosion of #rays as #bounces go up:

$$\#rays = N^{\#bounces}$$



解决：每次打出一根光线。即打向另一个物体的光线是1根。

```

shade(p, wo)

Randomly choose ONE direction wi~pdf(w)
Trace a ray r(p, wi)
If ray r hit the light
    Return L_i * f_r * cosine / pdf(wi)
Else If ray r hit an object at q
    Return shade(q, -wi) * f_r * cosine / pdf(wi)

```

这种方法有光源浪费的情况，因为打的光源很稀疏，有可能达不到某个物体，那就无法计算来自这个物体的反射光，但是打的很密，在一些空旷的地方会产生浪费（性能损失）

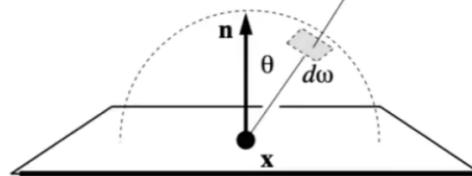
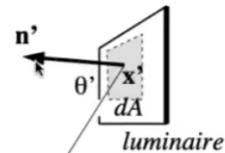
之前是均匀的采样，现在不采用均匀采样

Easy! Recall the alternative def. of solid angle:

Projected area on the unit sphere

$$d\omega = \frac{dA \cos \theta'}{\|x' - x\|^2}$$

(Note: θ' , not θ)



方法是不对表面均匀积分，直接对来自光线的表面进行采样。积分从原先的W变为物体表面的。代入蒙特卡洛方程。

这里还有一种可能，就是光源可能会被其他物体挡到

需要打一条连线看看是否有物体

2、算法是递归的，何时终止？

即光线弹射次数是无限的。

如果设定了一定的弹射次数，那会有一定的能量损失。

所以采用 RR算法（俄罗斯轮盘赌）

即用一定的概率来停止递归

```
shade(p, wo)
    Manually specify a probability P_RR
    Randomly select ksi in a uniform dist. in [0, 1]
    If (ksi > P_RR) return 0.0;

    Randomly choose ONE direction wi~pdf(w)
    Trace a ray r(p, wi)
    If ray r hit the light
        Return L_i * f_r * cosine / pdf(wi) / P_RR
    Else If ray r hit an object at q
        Return shade(q, -wi) * f_r * cosine / pdf(wi) / P_RR
```



将光源分为直接光源的照射和其他物体的光源。

直接的光源不用RR法，其他的使用RR法

```
shade(p, wo)

# Contribution from the light source.

Uniformly sample the light at x' (pdf_light = 1 / A)
L_dir = L_i * f_r * cos θ * cos θ' / |x' - p|^2 / pdf_light

# Contribution from other reflectors.

L_indir = 0.0

Test Russian Roulette with probability P_RR

Uniformly sample the hemisphere toward wi (pdf_hemi = 1 / 2pi)

Trace a ray r(p, wi)

If ray r hit a non-emitting object at q
    L_indir = shade(q, -wi) * f_r * cos θ / pdf_hemi / P_RR

Return L_dir + L_indir
```

点光源不好做，可以把点光源视为面积光源

17、材质与外观

材质 表现出 外观！



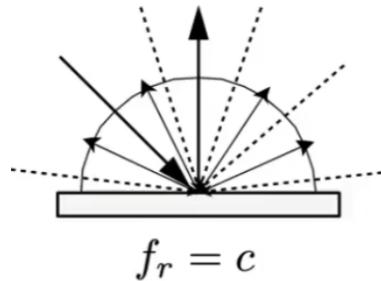
3D coffee mug model

Rendered

Rendered

在渲染方程中，和材质有关的一项是 BRDF，因为他决定了光如何被反射

Material == BRDF



Suppose the incident lighting is uniform:

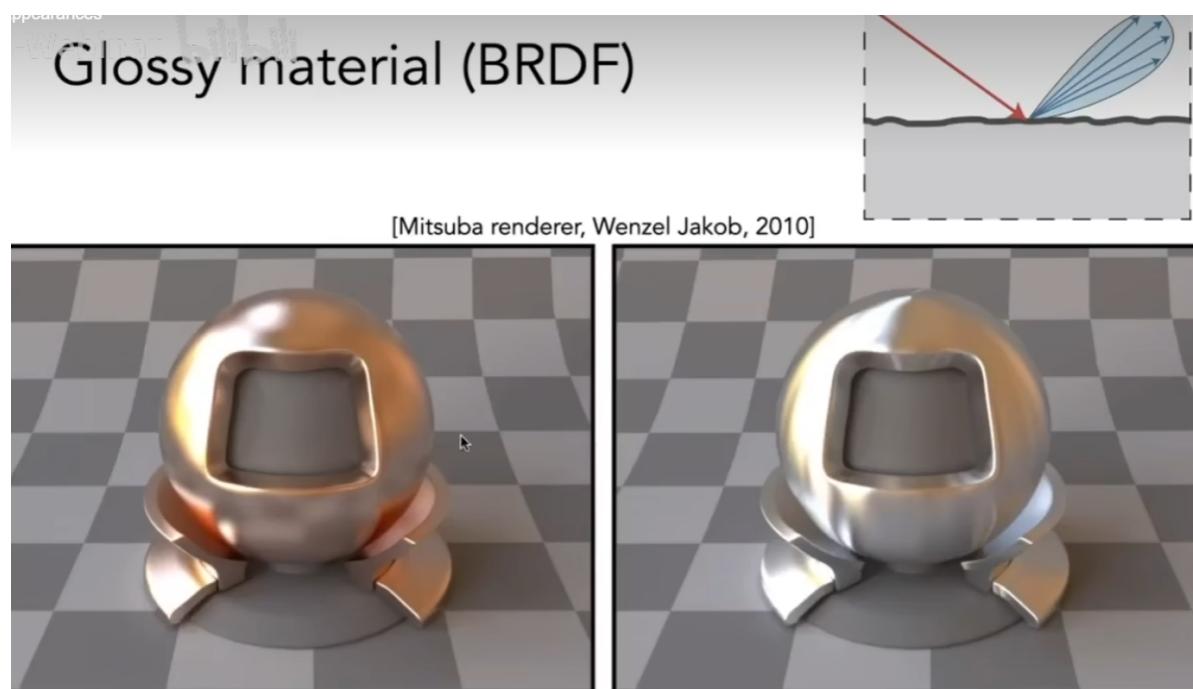
$$\begin{aligned} L_o(\omega_o) &= \int_{H^2} f_r L_i(\omega_i) \cos \theta_i d\omega_i \\ &= f_r L_i \int_{H^2} \cancel{(\omega_i)} \cos \theta_i d\omega_i \\ &= \pi f_r L_i \end{aligned}$$

$$f_r = \frac{\rho}{\pi} \quad \text{— albedo (color)}$$

漫反射的材质：

对渲染方程，设：入射的Radiance和BRDF是常数。提出来，可以解出 BRDF是 $1/\pi$ (在漫反射里假设入射和出射的radiance相同)

反射率：albedo。在0 - 1之间

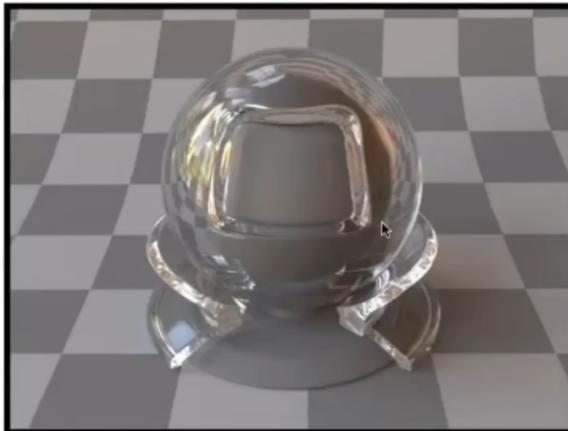
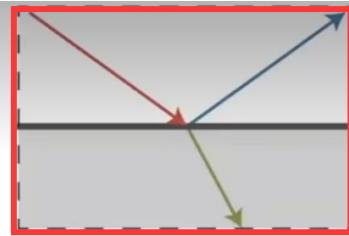


glossy materials 类似镜面材质

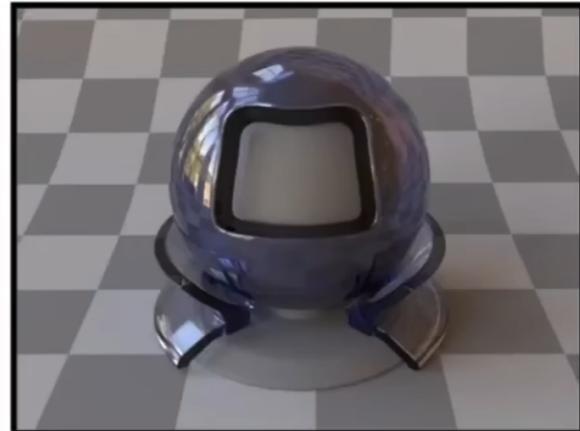
Appearances

Ideal reflective / refractive material (BSDF*)

[Mitsuba renderer, Wenzel Jakob, 2010]



Air <-> water interface

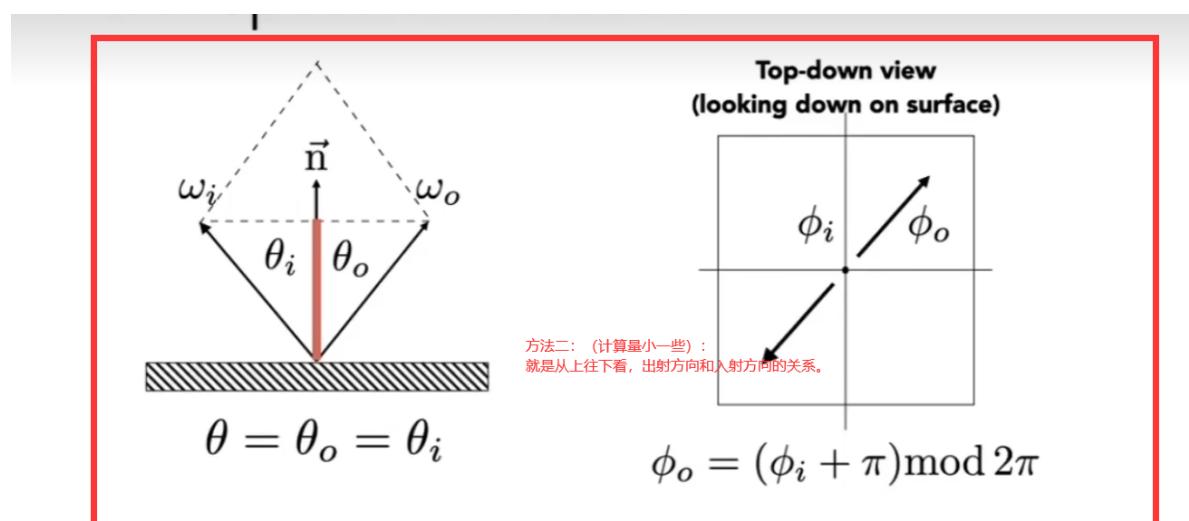


Air <-> glass interface
(with absorption)



玻璃或者水材质

求出射方向和出射角的两种方法：



$$\omega_o + \omega_i = 2 \cos \theta \vec{n} = 2(\omega_i \cdot \vec{n}) \vec{n}$$

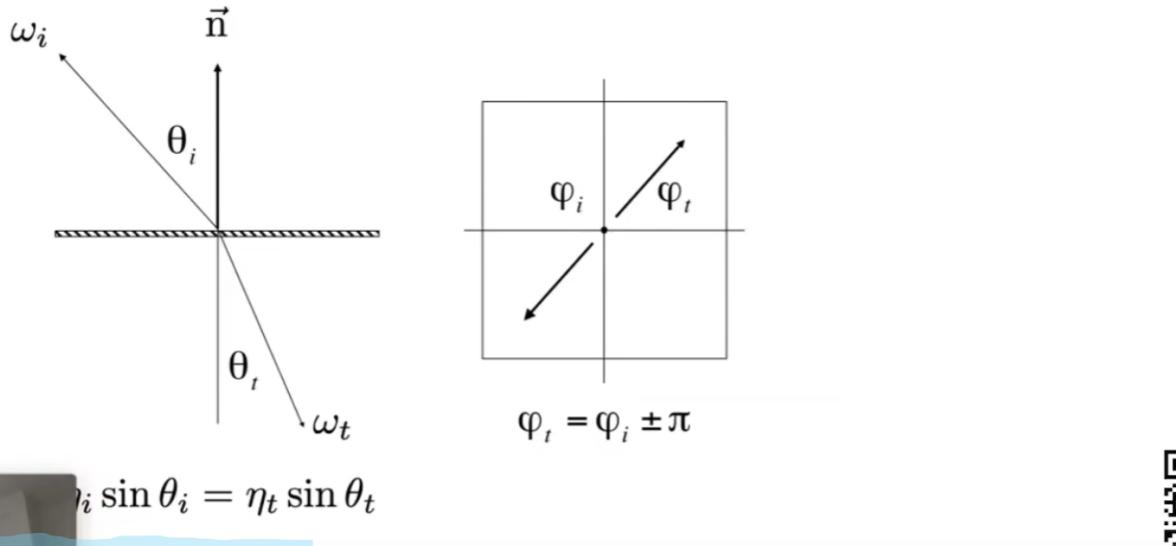
方法一：就是有反射的性质，进行一次点积然后移项。

$$\omega_o = -\omega_i + 2(\omega_i \cdot \vec{n}) \vec{n}$$

算折射方向：

Snell's Law

Transmitted angle depends on
 index of refraction (IOR) for incident ray
 index of refraction (IOR) for exiting ray



森尼尔定律：

规定一个平面有折射率，

折射率* 入射角的sin是个定值。

当折射率较大时，可能不会发射折射（全反射现象）

一个面的两端对应不同的折射率。

菲尼尔项

特点就是垂直看向某个物体，发现反射的比较少，斜着看，反射的比较多。

$$R_s = \left| \frac{n_1 \cos \theta_i - n_2 \cos \theta_t}{n_1 \cos \theta_i + n_2 \cos \theta_t} \right|^2 = \left| \frac{n_1 \cos \theta_i - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}}{n_1 \cos \theta_i + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2}} \right|^2,$$

$$R_p = \left| \frac{n_1 \cos \theta_t - n_2 \cos \theta_i}{n_1 \cos \theta_t + n_2 \cos \theta_i} \right|^2 = \left| \frac{n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2} - n_2 \cos \theta_i}{n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i \right)^2} + n_2 \cos \theta_i} \right|^2.$$

$$R_{\text{eff}} = \frac{1}{2} (R_s + R_p).$$

Rs和Rp就是真实值的两端的极化（波动）

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5$$

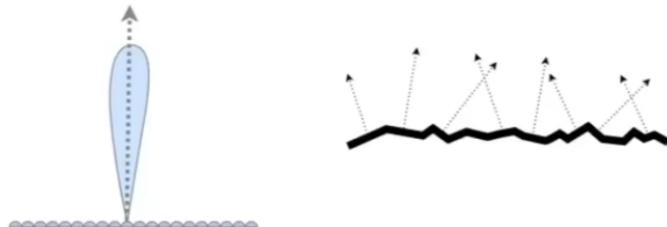
$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2} \right)^2$$

简化算法用以上公式简化。

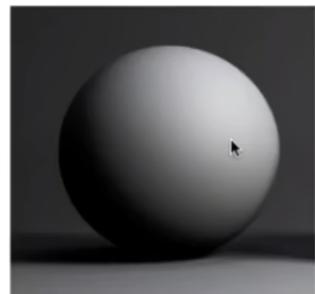
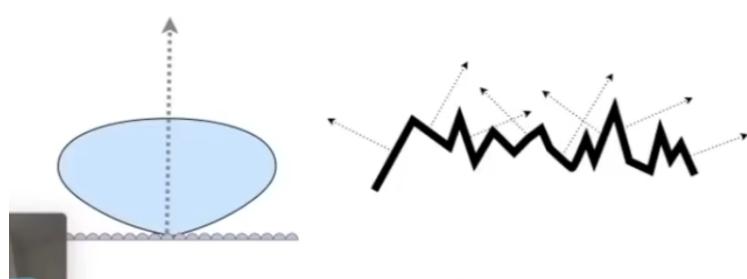
微表面

- Key: the distribution of microfacets' normals

- Concentrated <=> glossy



- Spread <=> diffuse

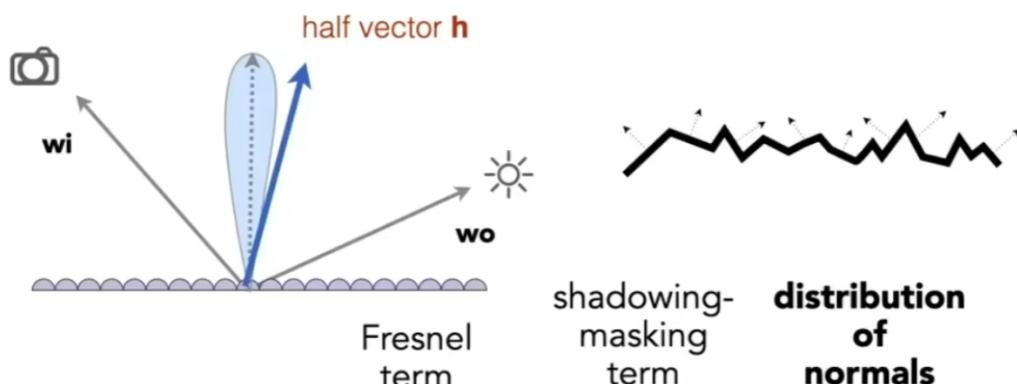


材质的粗糙程度可以用微表面的法线分布来描述

BRDF

brdf是反映一个面是如何反射的，还可以描述折射的。

- What kind of microfacets reflect w_i to w_o ?
(hint: microfacets are mirrors)



$$f(\mathbf{i}, \mathbf{o}) = \frac{\mathbf{F}(\mathbf{i}, \mathbf{h}) \mathbf{G}(\mathbf{i}, \mathbf{o}, \mathbf{h}) \mathbf{D}(\mathbf{h})}{4(\mathbf{n}, \mathbf{i})(\mathbf{n}, \mathbf{o})}$$

$F(ih)$ 菲尼尔项。

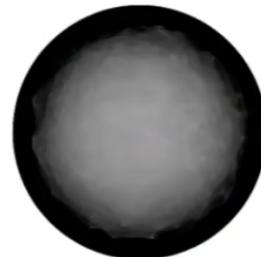
$D(h)$ 那些微表面可以把入射光反射到出射光

$G(ioh)$ 修正就是当光平行着表面射来的时候，产生特别亮的情况。

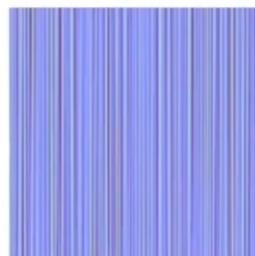
各向同性/各项异性材质

- Key: **directionality** of underlying surface

Isotropic



Anisotropic



Surface (normals)

BRDF (fix w_i , vary w_o)

各向同性：没哟明显的方向性。

各项异性：有明显的方向性。

在原理上：

$$f_r(\theta_i, \phi_i; \theta_r, \phi_r) \neq f_r(\theta_i, \theta_r, \phi_r - \phi_i)$$

就是他们的入射角，出射角在BRDF上，进行旋转是，各向同性是一样的。

测量BRDF

General approach:

```
foreach outgoing direction wo
    move light to illuminate surface with a thin beam from wo
    for each incoming direction wi
        move sensor to be at direction wi from surface
        measure incident radiance
```

就是基于物理的那些公式和实际有很多出入，所以采用直接测量的方法。

一个机器就是利用上图公式，360度固定一个相机位置，360度采用一个光源位置，然后测量，但是数据量太大，

由于各向同性的材质在方向上是一样的，从四维降到三维，
有因为brdf可逆性，又去掉一半。

18、

19、相机，棱镜，光场

21、动画

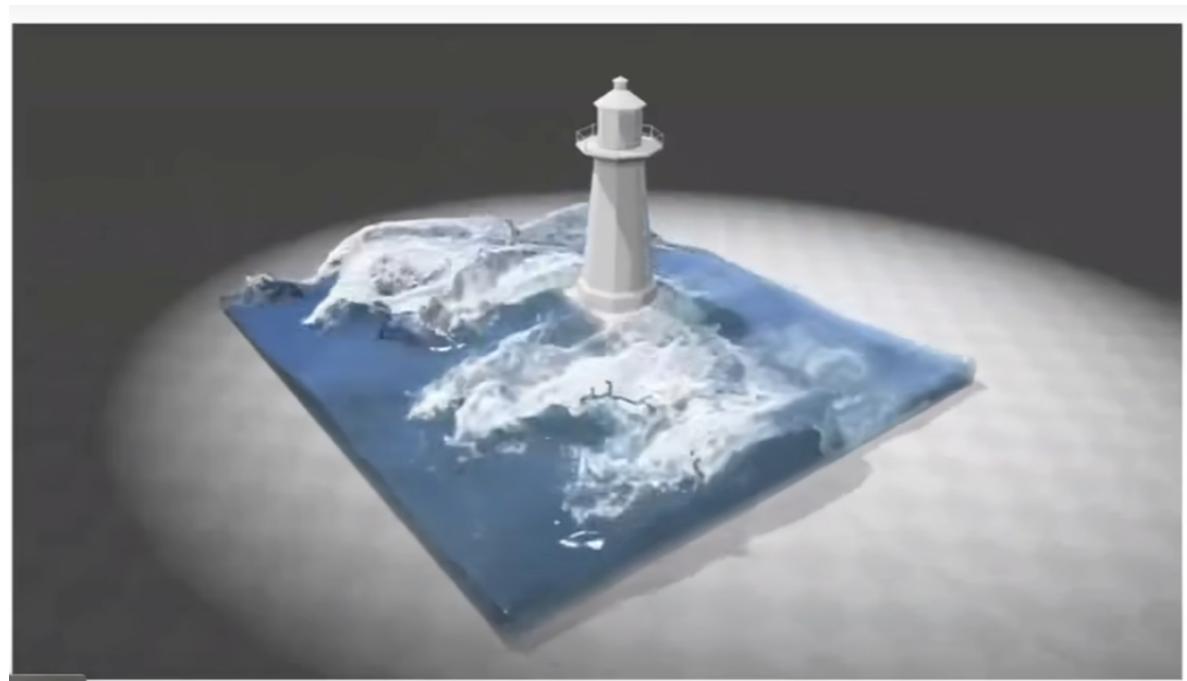
关键帧 KeyFrame



选出几个关键帧，中间用不同的方法去插值

物理模拟

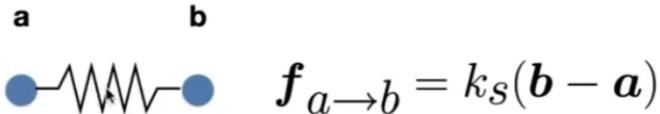
赋予物体（三角形）真实的物体特性，重力，收到的来自其他物体的牵引，速度，以及表面的摩擦力等等。



质点弹簧系统

是一系列相互链接的质点 和 弹簧

Idealized spring



$$\mathbf{f}_{b \rightarrow a} = -\mathbf{f}_{a \rightarrow b}$$

Force pulls points together

Strength proportional to displacement (Hooke's Law)

k_s is a spring coefficient: stiffness

胡克定律 (上图)

$$\mathbf{f}_{a \rightarrow b} = k_s \frac{\mathbf{b} - \mathbf{a}}{||\mathbf{b} - \mathbf{a}||} (||\mathbf{b} - \mathbf{a}|| - l)$$

↑
Rest length

x

$$\dot{\mathbf{x}} = \mathbf{v}$$

速度

$$\ddot{\mathbf{x}} = \mathbf{a}$$

加速度

考虑摩擦力，加一个和速度相反的力 (下图)



这个力和速度还有关系，而且沿着ab方向的速度 (下图)



a

b

Relative velocity of b,
assuming a is static (vector)

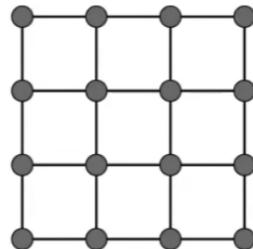
Damping force
applied on b

$$f_b = -k_d \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|} (\dot{\mathbf{b}} - \dot{\mathbf{a}}) \cdot \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|}$$

Relative velocity projected to
the direction from a to b (scalar)

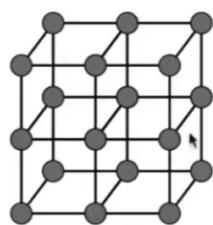
Direction from
a to b

Sheets



紙

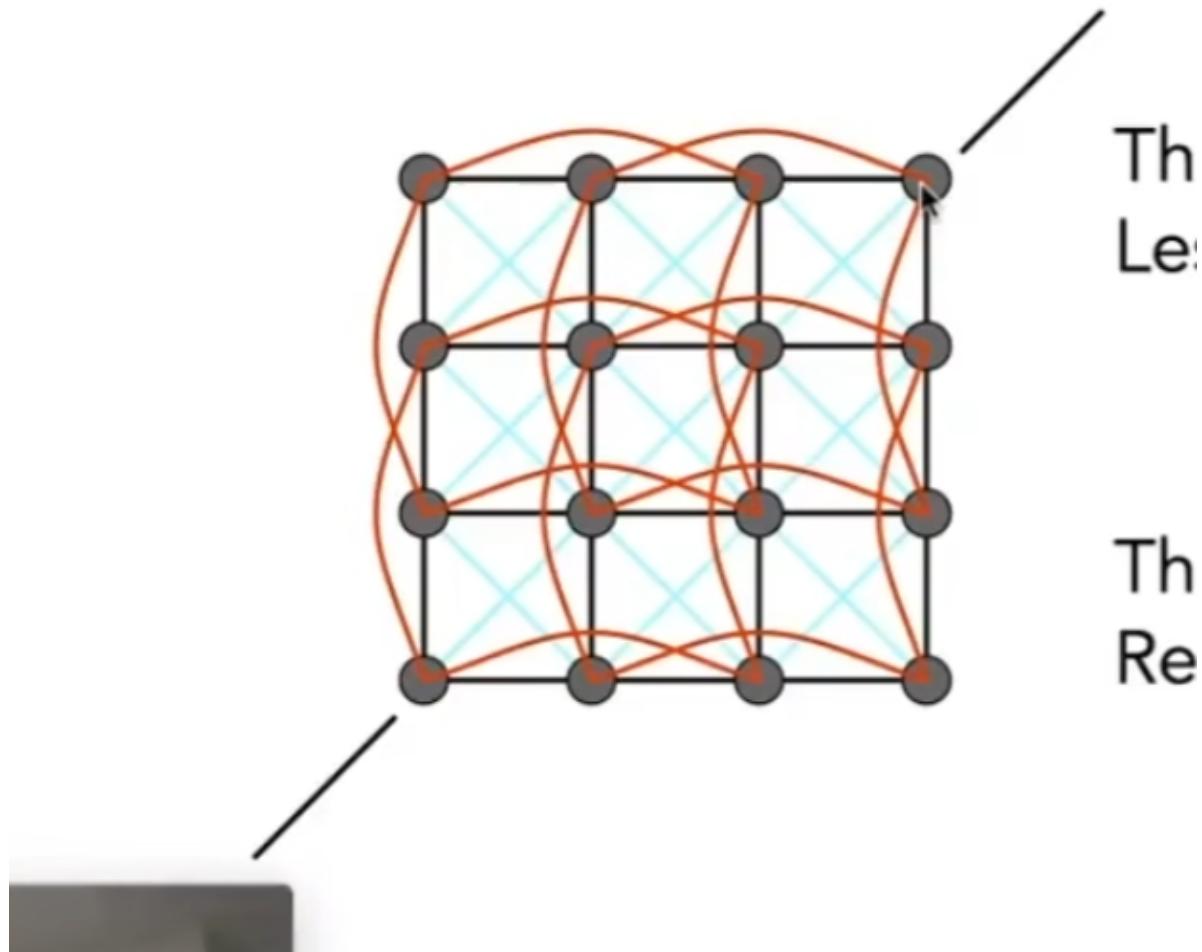
Blocks



三维物体

Others





表示一块布：沿对角线拉的时候要有地方切变的力，所以在对角线加弹簧，然后为了抵抗沿着竖直边对折的力，每隔一个加一个弱弹簧。

粒子系统

用很多的小粒子，赋予他们重力，相互之间的碰撞，风力，引力等等。

粒子越多，越精细。

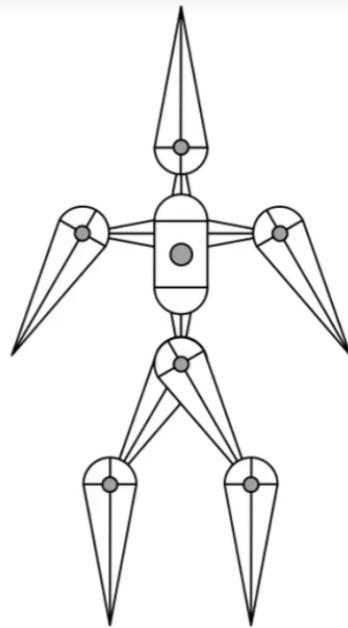
运动学

分正向和逆向

多用于处理骨骼。

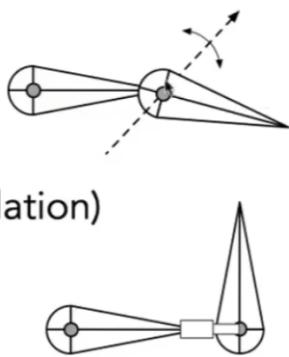
Articulated skeleton

- Topology (what's connected to what)
- Geometric relations from joints
- Tree structure (in absence of loops)



Joint types

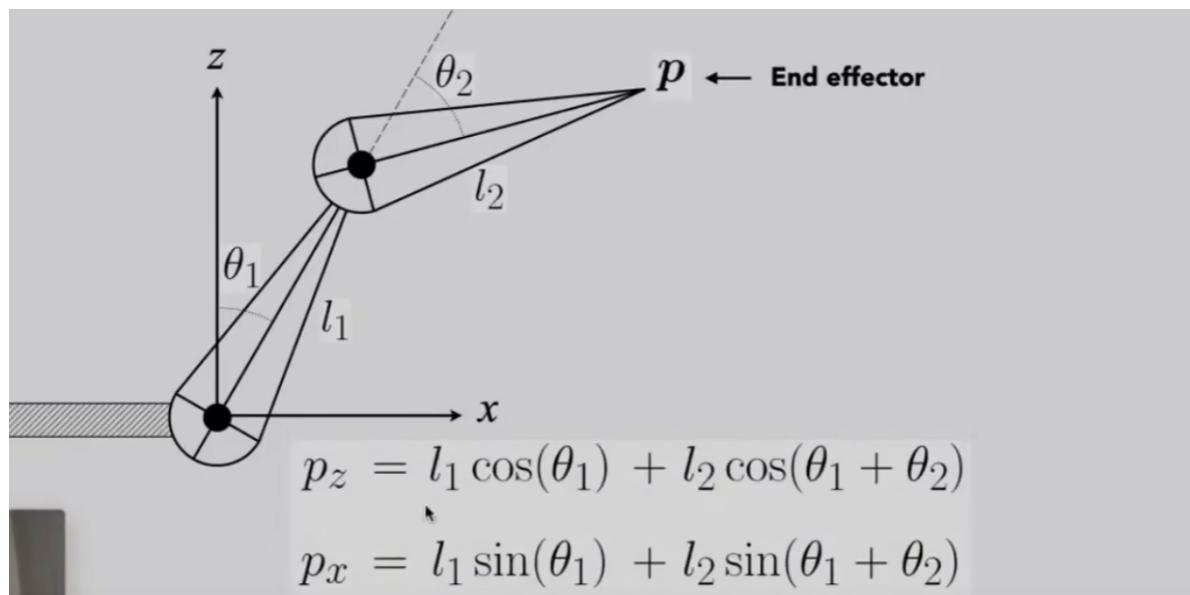
- Pin (1D rotation)
- Ball (2D rotation)
- Prismatic joint (translation)



Pin, 在平面内

Ball, 在空间中

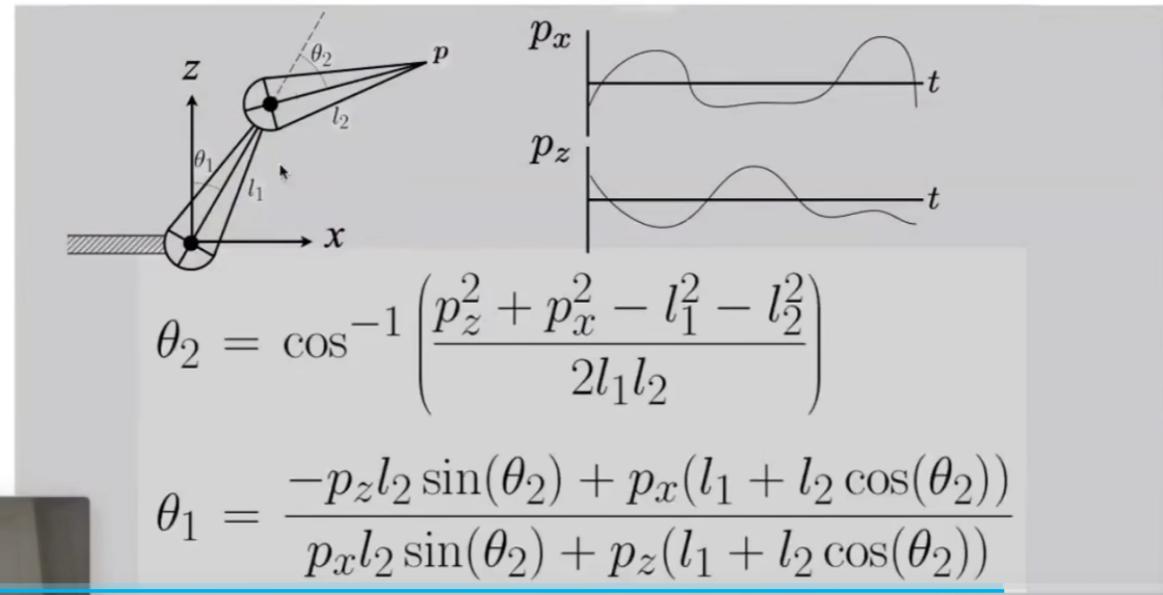
Prismatic joint 可以拉伸



某个关节转了几度，下一个关节转了几度，然后关节的位置。

逆运动学

Direct inverse kinematics: for two-segment arm, can solve for parameters analytically



先知道最后某个结点的位置，去求算上级的结点的位置（旋转，移动等等）

一些缺点：求算困难，解不唯一，无解

Rigging

控制，类似于控制木偶

采用逆向运动学

Monton Capture 动作捕捉

在真人上放上控制点，让他们去运动，来捕捉这些动作，更加真实

22、动画 (2)

欧拉方法

前向欧拉，显示欧拉

Euler's Method (a.k.a. Forward Euler, Explicit Euler)

- Simple iterative method
- Commonly used
- Very inaccurate
- Most often goes **unstable**

$$x^{t+\Delta t} = x^t + \Delta t \dot{x}^t$$

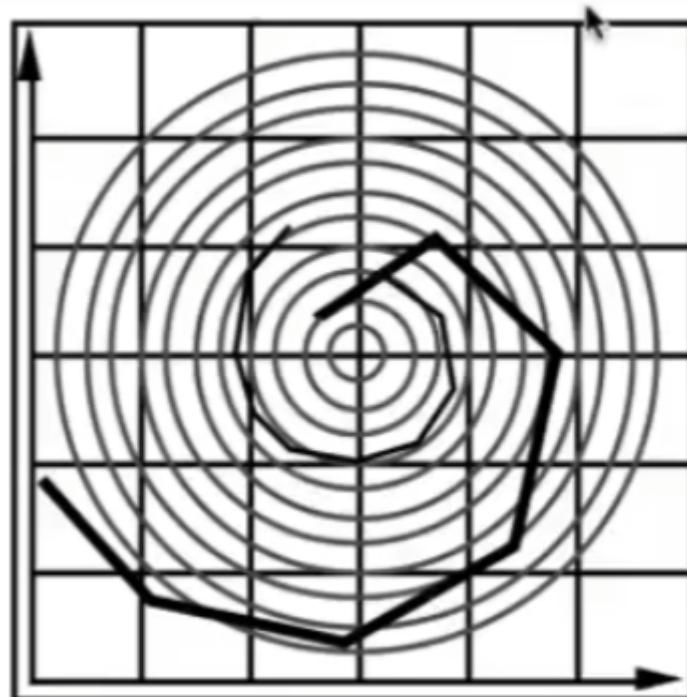
$$\dot{x}^{t+\Delta t} = \dot{x}^t + \Delta t \ddot{x}^t$$

就是由上一帧的速度推下一帧，上一帧的位置推下一帧。

问题：

误差：十分不准，有误差。

不稳定：对于处理螺旋形的速度场，模拟的久了，一定会离开场。（这个和误差无关，一定会偏离场）

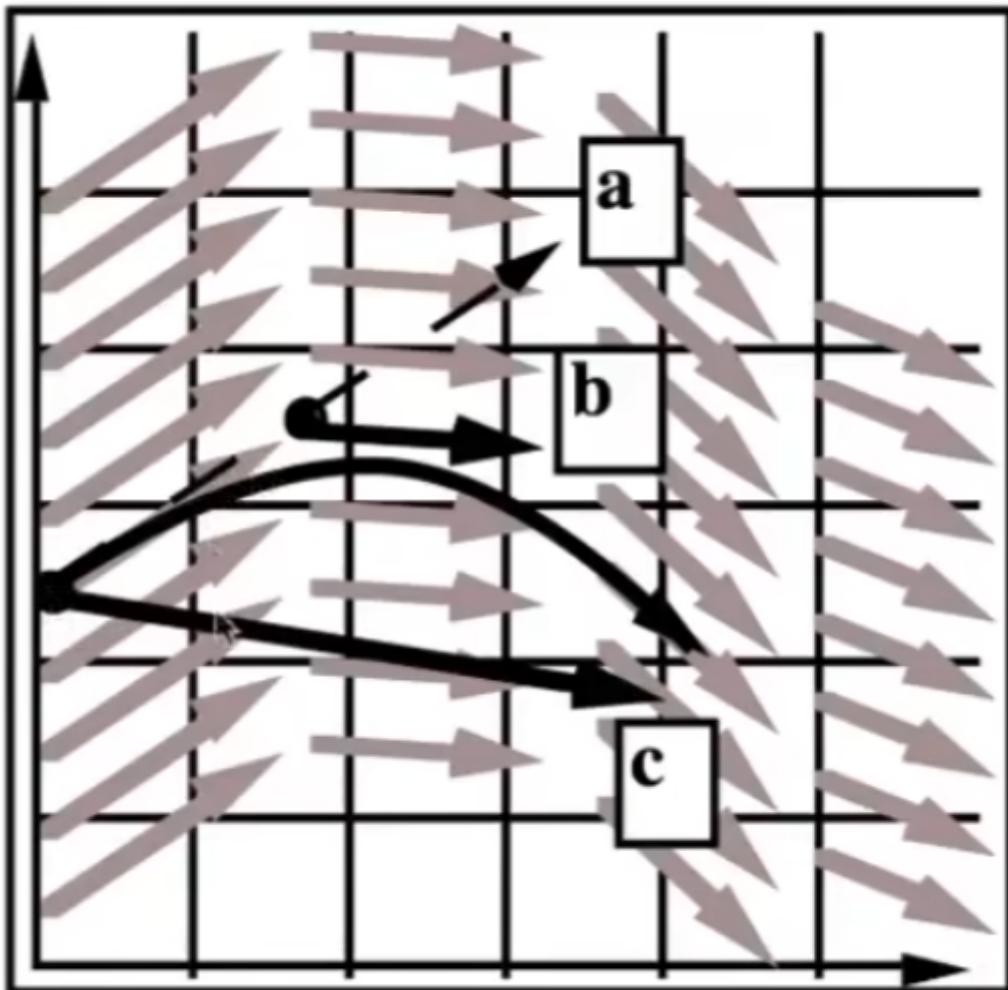


比如：pubg汽车撞到东西后，会有直接起飞的情况，在实际中很少有汽车直接起飞的情况

如何对抗不稳定性

中点法

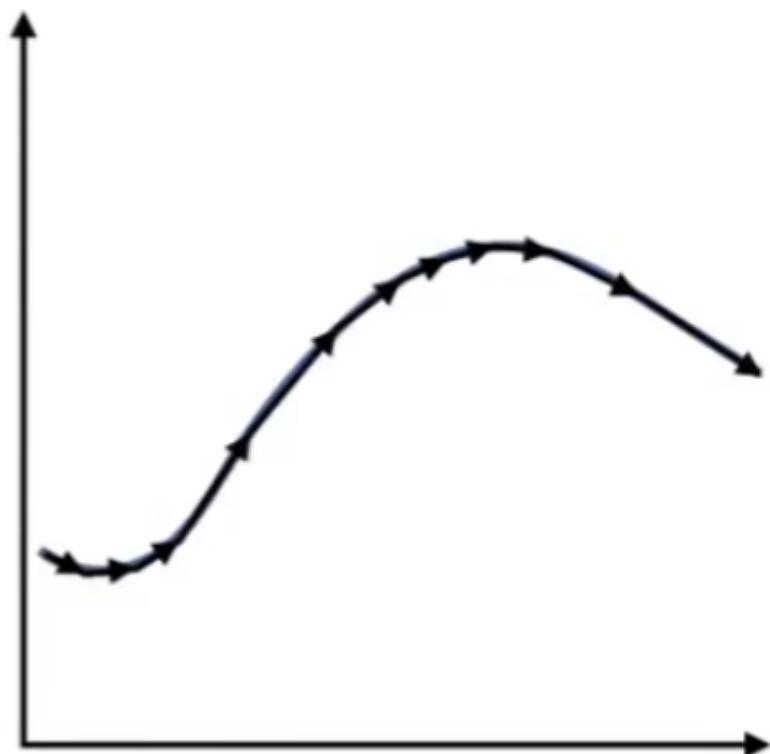
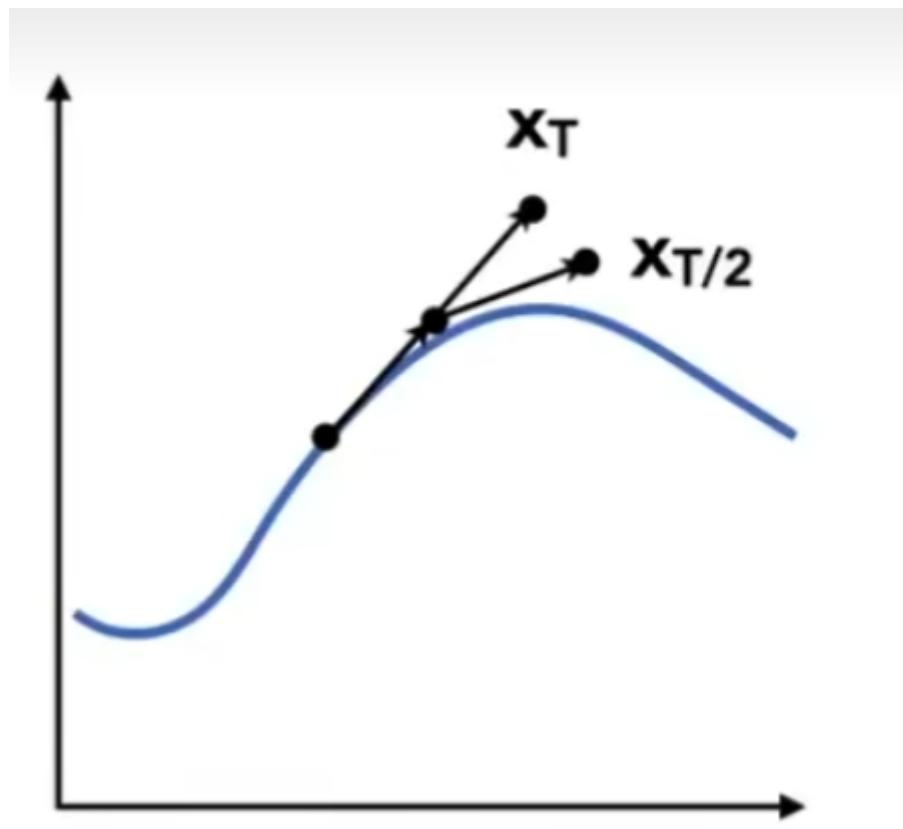
先使用欧拉方法计算出一个位置，然后找到起始点和这个位置的中点，求出在中点位置的速度，然后用中点的速度在起点重新开始运动。



Witkin and Baroff

自适应方法

先使用欧拉方法求一段，然后把这一段分成两端，分别求一次欧拉方法，看看最终结果的差距，如果差的很多，那就在二分，知道结果差的不多，就往下进行。



隐式欧拉方法

(后项欧拉)

使用下一时刻的速度和加速度

应为要用到未来的量，需要用公式解，但是不好解

$$\boldsymbol{x}^{t+\Delta t} = \boldsymbol{x}^t + \Delta t \dot{\boldsymbol{x}}^{t+\Delta t}$$

$$\dot{\boldsymbol{x}}^{t+\Delta t} = \dot{\boldsymbol{x}}^t + \Delta t \ddot{\boldsymbol{x}}^{t+\Delta t}$$

荣格库塔方法

解常微分方程很有利