**点乘和叉乘的几何意义**

点乘：a·b = |a||b|cosθ，可用于计算两向量夹角、计算一个向量到另一个向量的投影、判断两向量前后。

叉乘：a×b是a和b两向量所在平面的法线向量，大小为|a||b|sinθ。与a、b向量组成的平行四边形面积相等。可用于判断左右/内外（判断点在三角形内部）。

点乘和叉乘均可用矩阵表示：a·b = aTb，a×b = A\*b，其中A\*表示对偶矩阵。

**仿射变换/齐次坐标**

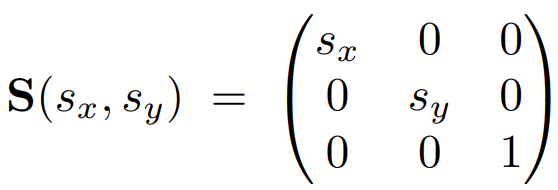
把点从n维(x, y, z)上升到n+1维(X, Y, Z, w)（x = X/w, y = Y/w, z = Z/w）。

n维线性变换无法实现**平移**操作但齐次坐标可以。

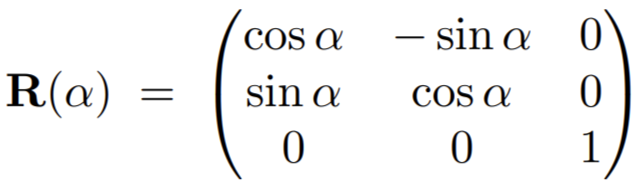
3维的点：(x, y, z, 1)T

3维的向量：(x, y, z, 0)T

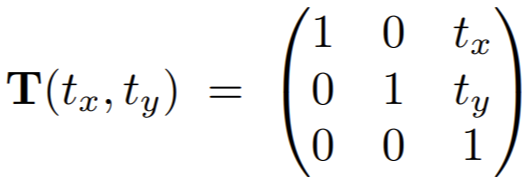
**缩放：**



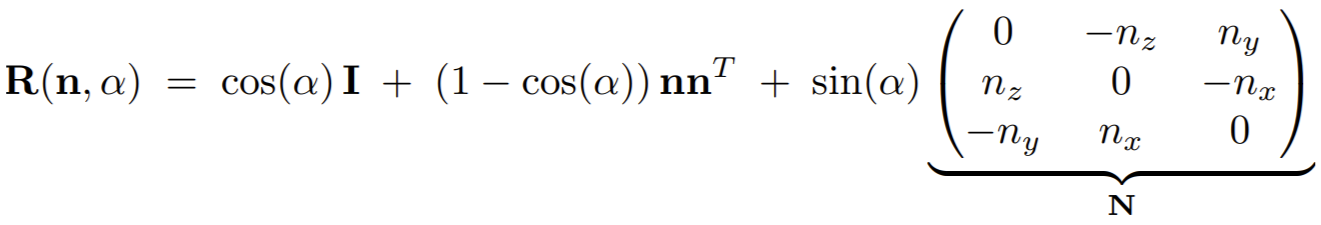
**旋转：**



**平移：**



关于任意向量n旋转α的旋转矩阵：（罗德里格旋转公式，将原向量分解为与n平行和与n垂直两部分然后推导可得）



由于矩阵运算的结合律，多个仿射变换可以合并为一个矩阵。

**欧拉角、矩阵与四元数**

**欧拉角**：三个值来表示旋转，按固定顺序旋转，（Unity：z-x-y）。非常直观，便于理解，但在某些情况下会使得两坐标轴重合，就会发生万向锁现象，丢失一个方向上的旋转能力。

**矩阵**：使用一个3\*3的矩阵表示独一无二的旋转，即实现线性变换，但是消耗空间大，运算繁杂。



**四元数**：用一个4维向量表示旋转，四元数相乘表示旋转叠加，可以平滑插值，缺点理解不直观。

**渲染管线**

① **应用阶段**

CPU准备渲染的数据，执行视椎剔除，设置渲染状态，绑定参数，将顶点数据发送给GPU。

② **几何阶段**

**顶点处理（MVP）**：

**模型变换**：从模型坐标转换为世界坐标。

**视图/相机变换**：将世界坐标转换为相机坐标。

**投影变换**：正交/透视变换，将相机坐标转换为**正规化可视空间**内的坐标。

**视口变换**：将坐标转换为屏幕坐标。

**三角形处理、裁剪。**

③ **光栅化阶段**：

对三角形进行遍历，进行着色处理。最后三大测试，输出图像。

其中顶点处理和像素（片段）处理阶段可以编程。（顶点着色器、片段着色器，Shader编程，GLSL）

Alpha测试：根据物体透明度决定是否渲染。

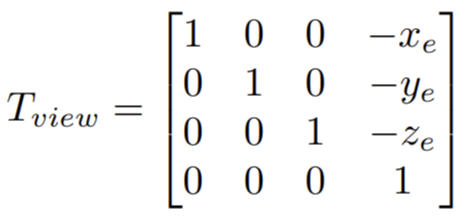
模板测试：根据物体位置范围决定是否渲染。

深度测试：根据物体深度决定是否渲染。

**视图变换**

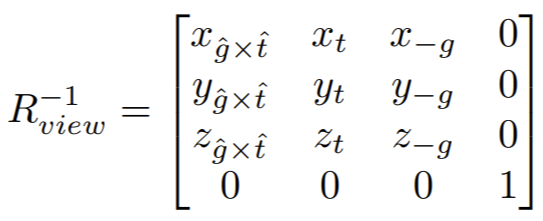
将相机移动到默认点/将坐标系转换为相机坐标系。

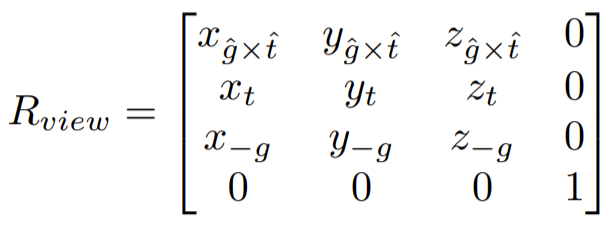
先**平移**



再**旋转**

将 g (注视方向)旋转到-z，将 t (向上方向)旋转到y，g×t旋转到x。





视图变换矩阵为

截图.png

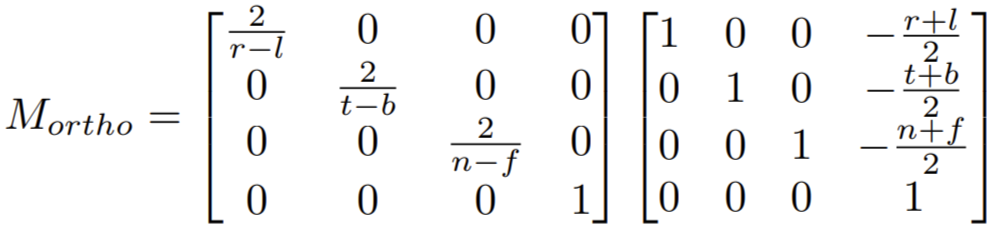
**投影变换**

将相机坐标转换为正规化可视空间内的坐标。

**正交投影**

将长方体[l, r] x [b, t] x [f, n]映射到[-1, 1]^3。

先**平移**后**缩放**：

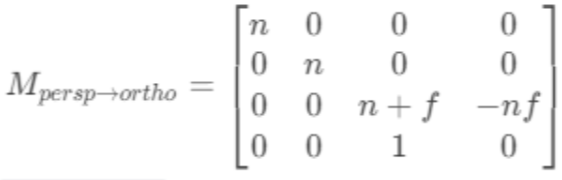


**透视投影**

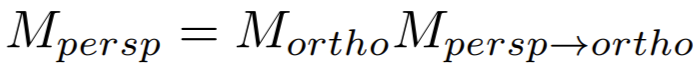
将视椎体映射到长方体[l, r] x [b, t] x [f, n]，在经过正交投影映射到[-1, 1]^3。

根据相似三角形 x' = (n / z)x，y' = (n / z)y。且在近平面和远平面时z不变，即 z = f 和 z = n 时 z' = z。

可得转换矩阵为



而透视投影变换矩阵为



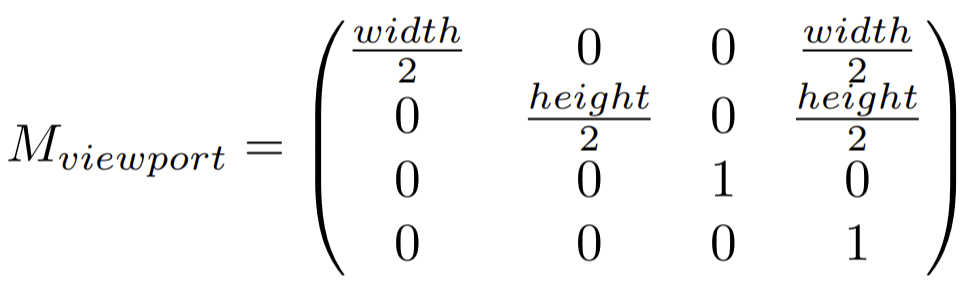
**视口变换**

将坐标转换为屏幕坐标。

将 [-1, 1]^2 转换到 [0, width] x [0, height]。

平移后缩放

视口变换矩阵为



**采样/判断点在三角形内**

采样就是对函数离散化的过程。

将三角形作为限定条件，对各像素中心进行采样，显示在像素中心的点，从而实现光栅化。

包围盒优化：利用三角形包围盒对一定不会包含三角形的像素进行优化。

**判断点在三角形内**：

**同侧法
：**点P同时在向量AB、BC、CA的同侧，PA×AB、PB×BC、PC×CA符号相同。

**方程法/线性组合法
：**AP=m AB+n AC， m > 0, n > 0, m + n <= 1。等价于P = (1 - m - n) A + m \* B + n \* C。

面积法：三角形PAB、PAC、PBC的面积之和与三角形ABC面积相等。计算面积用叉乘。

射线法：从点P发出一条射线，看与三角形交点，若为奇数则在三角形内，若为偶数则在三角形外。

**走样/反走样/抗锯齿**

相同的采样方法采样两种不同频率的函数得到完全相同的结果。本质上是采样频率跟不上信号频率。

锯齿、摩尔纹、车轮效应等。

减少走样的方法：

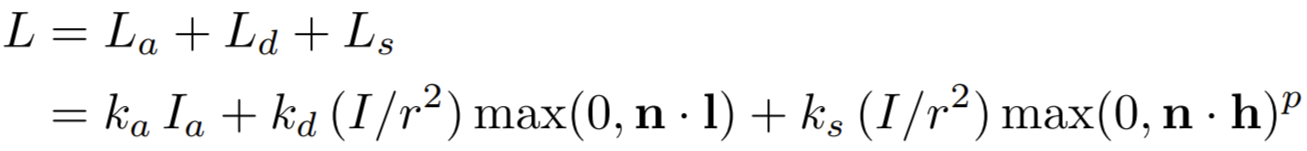
采样前进行一次模糊/低通滤波，去除高频信号，减少频谱重叠。

MSAA，对一个像素内部多增加采样点，并依据像素内采样点在三角形的比例对像素色值加权。缺点：计算量增加。

其他：FXAA（快速近似抗锯齿，图像层面上的后处理）、TAA（复用上一帧的信息）、DLSS（深度学习超采样）。

**着色/Blinn-Phong反射模型**

朗伯余弦定律：接收到的辐照度与入射光线和法线夹角的余弦值有关。



物体发出的光由**环境光**、**漫反射光**和**镜面反射光**组成。

I为光源强度，r为到光源的距离，n为法线，l为入射光线，h为半程向量，为入射方向和观察方向的中线。

Phong模型中没有半程向量，而是根据镜面反射方向和观察方向夹角，计算更为复杂。

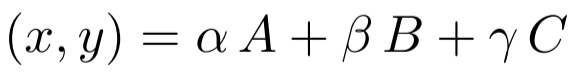
着色依赖于法线，着色频率分为**逐面**（依据三角面片法线）、**逐顶点**（依据顶点法线对三角形定点着色，再在三角形内部进行着色插值，顶点法线由共享顶点的平面的法线的平均或依据面大小的加权平均求得）、**逐像素**（根据三角形顶点法线插值各像素的法线）。

**纹理映射**

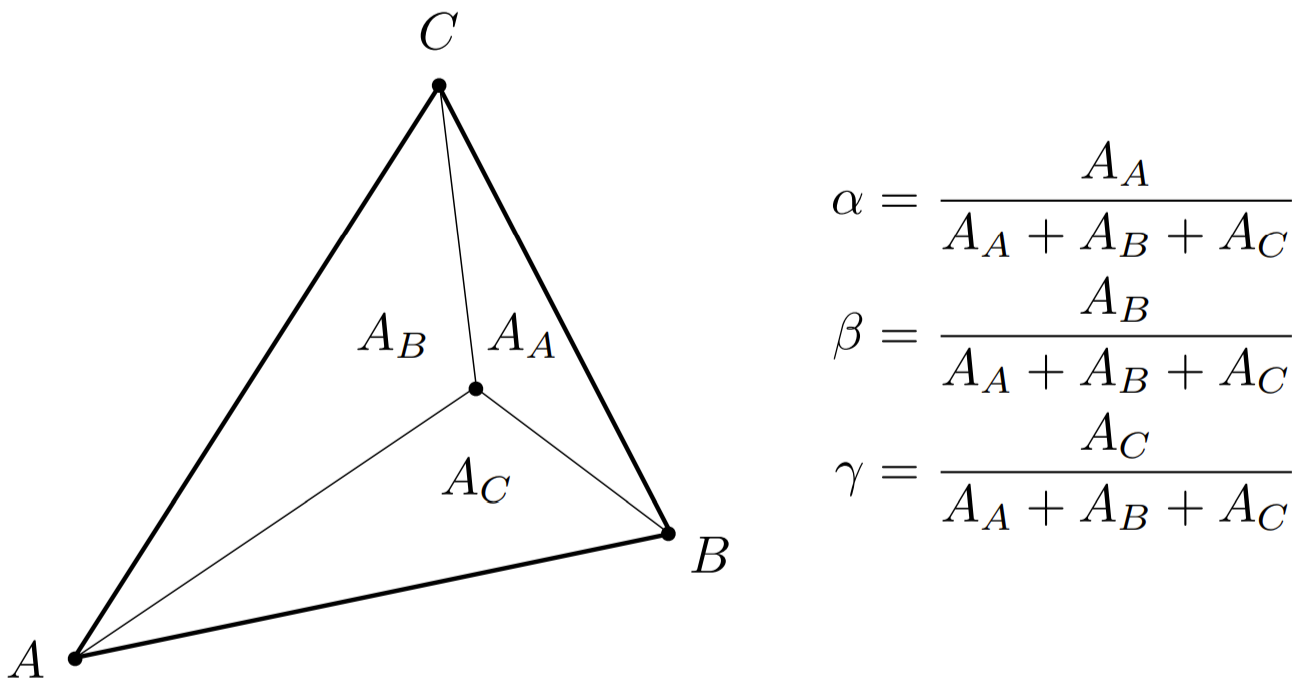
三维模型表面上一点对应纹理uv坐标上的某个点。

**三角形内部插值**

可用于纹理、着色：



截图.png



纹理较小时：锯齿。需要进行双线性插值。

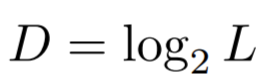
纹理较大时：摩尔纹（一个像素包含多个纹素，采样率更低）。使用Mipmap。

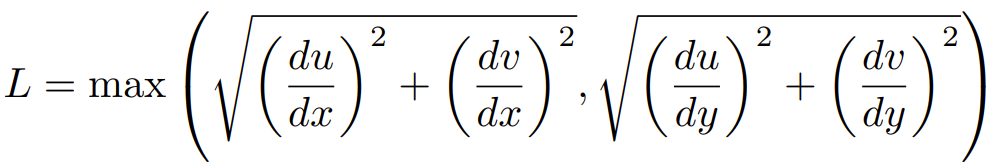
**Mipmap**

将原始贴图分为多个不同大小的层，相邻层之间边长减半，即分辨率越来越低。

多占用了1/3的空间。

计算Mipmap层数：计算与相邻像素的纹理坐标距离最大值L，log2L即为层数。





各向异性过滤：x和y分别进行压缩，耗费3倍空间。

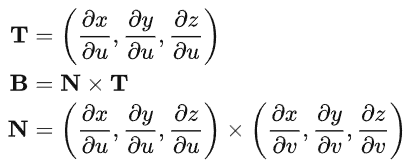
**纹理映射应用**

环境映射：球面映射、立方体映射。

法线贴图：将法线信息存储在贴图上，在不影响几何信息的情况下改变物体光照，从而达到凹凸的效果。

位移贴图：真实改变几何，模拟凹凸部分。

**切线空间/法线贴图**



切线空间：由法线、切线组成的空间。计算时将坐标转换为切线空间用于实现法线贴图。

**延迟渲染管线**

用于解决大量光照时的渲染，光栅化时先进行深度缓存找出所有能看到的像素，即离摄像机最近的片元。然后再进行光照着色。

优点：省去大量三角形的着色，减少不必要的光照计算量，提高了渲染性能。可用于处理大量光照的复杂场景。

缺点：更占用内存，无法直接支持透明物体的渲染，无法直接支持抗锯齿。

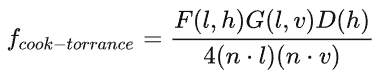
Early-Z：在前向渲染的基础上，对物体逐个渲染，对每个面片先进行深度剔除再进行光照计算。

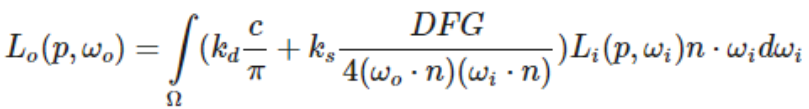
**基于物理的渲染PBR**

基于微表面的表面模型。（ **法线分布函数D**和**几何遮蔽函数G**）

能量守恒。

应用基于物理的BRDF，镜面反射、漫反射、折射（基于**菲涅尔方程F**，描述光线被折射部分比例）。





PBR材质：粗糙度、金属度。

**几何的表示方法**

隐式表示：函数表示、CSG（构造性立体几何，复杂的几何体通过简单几何体进行交并补等集合运算得到）、距离函数等。

显式表示：点云、多边形网格。

**贝塞尔曲线**

t从0取到1，对相邻控制点进行t的插值，递归进行直至点的数量为1。

常用三阶贝塞尔：四个控制点。利用分段贝塞尔表示任意曲线。

保证相邻贝塞尔曲线连续：上一段曲线最后一个控制点与下一段曲线第一个控制点相同，同时该控制点与上一段曲线倒数第二个控制点、下一段曲线第二个控制点在同一条直线上且间距相同。

贝塞尔曲面：与贝塞尔曲线类似，但参数为 u, v，变为两个。

**曲面细分/简化**

**Loop细分**

在每个边上添加新的顶点。

新顶点位置：V'=3/8 \* (A + B) + 1/8 \* (C + D)，其中 A 和 B 为两个面被共享边的的老顶点，C 和 D 为非共享边的老顶点。

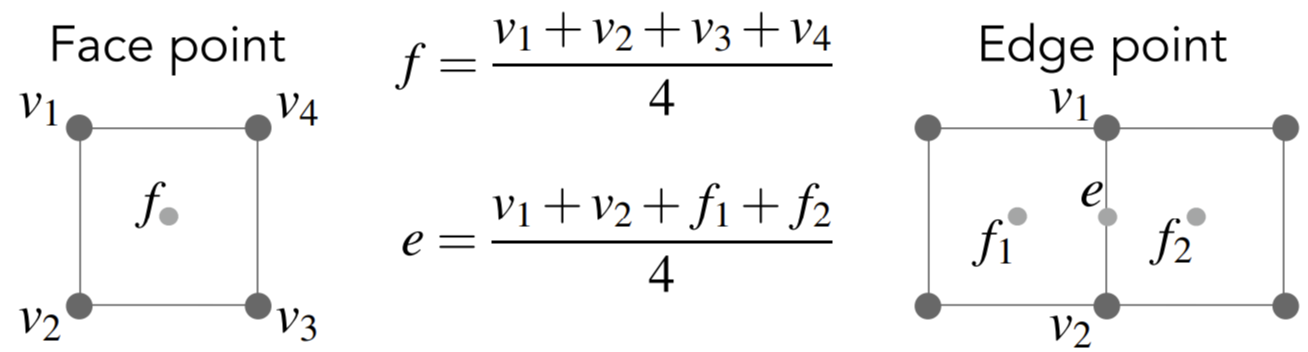
老顶点位置：V'=(1 - n\*u) \* V + original\_position \* neighbor\_position\_sum，其中 n 为顶点的度，当 n 为3时 u 为 3/16，否则 u 为 3/(8n)。

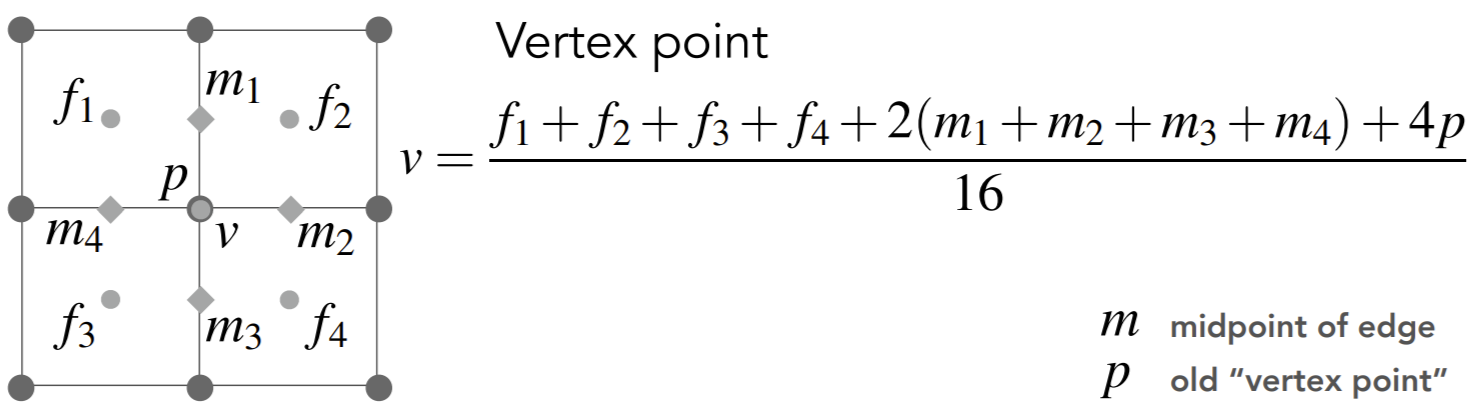
**Catmull-Clark细分**

可用于四边形。

增加网格数，取每条边的中点和每个方块的中点相连将区域划分为多个四边形。

调整顶点。





**曲面简化**

采取边坍缩的方法。利用二次度量误差使得新点到相关三角形面距离最小。

**Shadow Map**

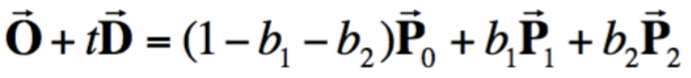
从光源看向场景并做深度测试，接着从观察点正常进行光栅化。若不能被观察点看到但可以被视椎体看到则为阴影。

缺点：只能产生硬阴影、分辨率问题可能出现锯齿、判定时的浮点精度问题（阴影抖动）。

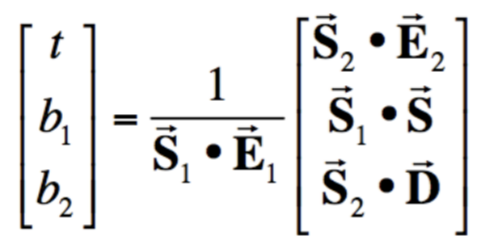
解决阴影锯齿/软阴影：百分比渐进过滤(Percentage Closer Filter，PCF)，从深度图进行多次采样，比如相邻的九个像素，然后进行加权平均。通过改变PCF的Kernel Size可以改变阴影模糊半径从而模拟出软阴影。

**光线追踪**

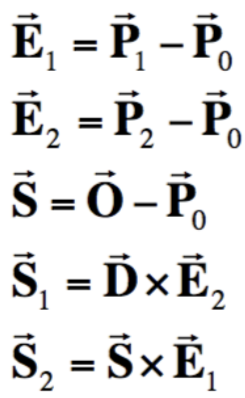
**判断射线与三角形相交**



根据**M**öller **T**rumbore算法，解得



其中



**判断射线与包围盒相交**

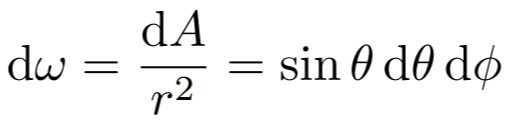
对每一对面，计算 tmin 和 tmax。

例如对 x = p 的平面来说，t = (p - o\_x) / d。

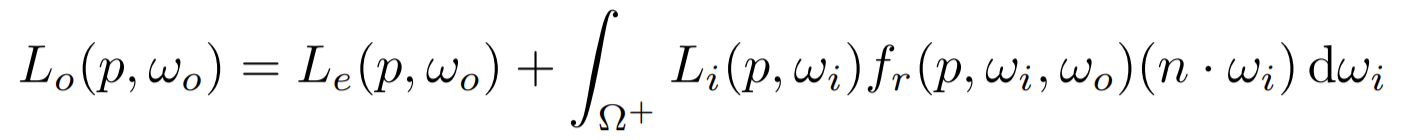
则t\_enter = max { tmin }，t\_exit = min { tmax }。

当t\_enter < t\_exit 且 t\_exit >= 0 时，认为射线与包围盒相交。

**立体角**



**渲染方程**



反射光等于发射光加上各个立体角内的入射光乘上对应BRDF的和。一种递归算法。

BRDF：出射光辐射率(Radiance)的微分和入射光辐照度(Irradiance)的微分之比。

**Whitted-style光线追踪**

只进行镜面反射、折射。遇见漫反射表面会停止。

**路径追踪**

蒙特卡洛采样。一次采样为一条光线，到达场景中一点时，首先计算来自光源的直接光照（从光源处采样），然后进行俄罗斯轮盘赌，若判定成立，则随机选取一个反射方向进行采样继续递归。

**光线追踪与光栅化**

解决了软阴影问题、间接光照。但速度较慢。

**空间划分算法**

**网格**

基于网格划分世界，二维用二维数组，三维用三维数组。（棋类游戏、MineCraft方块游戏）

**四叉树/八叉树**

将地理空间递归划分为不同层次的树结构，将已知范围空间等分成四个/八个相等的子空间，如此递归下去直至满足要求或树的层数达到一定深度。

**BSP树**

BSP树是一棵二叉树，其每个节点表示一个有向超平面的形状，其代表的平面将当前空间划分为前向和背向两个子空间，分别对应左儿子和右儿子。

**k-d树**

特殊的BSP树，每层对应一个划分维度，x轴和y轴交替划分，直至结点代表的空间内物体小于一定数目。

**层次包围盒树（BVH）**

包围盒：AABB（轴对齐包围盒，矩形/六面体）、OBB（有向包围盒）、包围球、k-DOP（k层离散方向多面体，k个面包含对象且所有面法向量都取自固定方向）

根据包围盒创建树，求出各物体的包围盒，将他们分为两部分，再重新求每组包围盒。最终每个叶子节点表示一个对象的包围盒。分割时选取最长的轴，将中间的物体作为分割线划分（快速排序可以实现O（n)复杂度寻找中位数）。

划分优化：SVH，考虑了物体分布的问题，物体的击中概率作为权重进行划分而不是以中位数。

**判断矩形相交**

判断两个轴对齐矩形相交：

1. 令矩形1和矩形2的最左下角的点分别为A1和A2，右上角的点分别为B1和B2，若max(x\_A1, x\_A2) <= min(x\_B1, x\_B2)且max(y\_A1, y\_A2) <= min(y\_B1, y\_B2)；

2. x轴方向上O1和O2的距离一定小于或等于矩形1的宽度+矩形2的宽度的一半；在y轴方向上O1和O2的距离一定小于或等于矩形1的高度+矩形2的高度的一半。

**物理模拟**

**质点弹簧系统**

认为物体是由一系列质点和弹簧组成的，绳子、布料（斜向弹簧、跳跃弹簧）等，注意需要加上摩擦力。

**粒子系统**

计算粒子之间的力（引力、斥力、摩擦力、碰撞）来进行模拟。

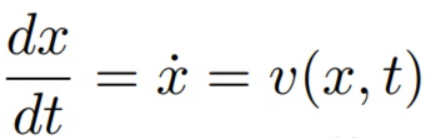
**正/逆向运动学**

正向运动学：正向计算各个骨骼位置，从而确定终点。

逆向运动学：逆向首先确定终点位置，再计算各个骨骼位置。使用解析法、数值法（雅可比逆矩阵、牛顿法）、启发式法（梯度下降等）解决。

**单粒子运动**

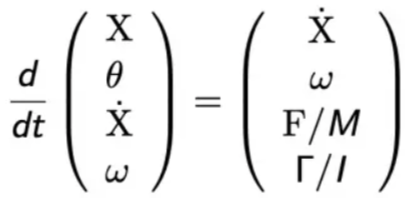
已知粒子速度场



求解每时刻粒子速度：

欧拉法、中点法、隐式欧拉法、龙格库塔法。

**刚体模拟**



**流体模拟**

质点法（拉格朗日方法）：考虑模拟规模群体中的每一个个体，并考虑其间的作用力。

网格法（欧拉方法）：将空间划分为一个一个的小格，模拟计算每一个网格内的运动。