

计算机图形学

learning note For reading translation

我真的不懂忧郁



计算机图形学

learning note For reading translation

by

我真的不懂忧郁

Student Name Student Number

First Surname 1234567

Instructor: I. Surname
Teaching Assistant: I. Surname
Project Duration: Month, Year - Month, Year
Faculty: Faculty of Aerospace Engineering, Delft

Cover: Canadarm 2 Robotic Arm Grapples SpaceX Dragon by NASA under
CC BY-NC 2.0 (Modified)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld



Preface

A preface...

我真的不懂忧郁
Delft, May 2024

Summary

A summary...

目录

Preface	i
Summary	ii
Nomenclature	v
1 光栅化	1
1.1 线性插值	1
1.2 重心坐标	1
1.3 直线的光栅化	3
1.4 三角形光栅化	5
1.5 反走样	5
1.6 对图形深度的处理	8
2 着色	10
2.1 平方反比定律	10
2.2 Blin-Phong 反射模型	11
2.3 着色频率	14
3 图形渲染管线	17
4 纹理映射	18
4.1 投影映射	19
4.2 变换函数	19
4.3 用重心坐标做插值	20
4.4 纹理放大	21
4.5 帧插值技术	24
4.6 环境光映射	24
4.7 纹理的其他应用	25
5 几何	26
5.1 几何的表示	26
5.2 不同表示方法的意义	26
5.3 曲线	27
5.4 曲面	27
5.5 Mesh Operations	27

6 光线追踪	29
6.1 Shadow Mapping	29
6.2 为什么需要光线追踪	29
6.3 光线	29
6.4 Ray Casting	29
6.5 Recursive(Whitted-Style) Ray Tracing	30
6.6 光线与物体表面交点怎么求?	30
6.7 Accelerating Ray-Sufaces Intersection	30
6.8 Uniform Spatial Partitions(Grids)	31
6.9 Spatial Partitions	31
6.10 Object Partition and Bounding Volume Hierarchy	32
7 基于辐射度量学的光线追踪	33
7.1 Radiant Energy and Flux(Power)	33
7.2 立体角 (soild angle)	34
7.3 辐照强度、辐射照度和辐射亮度	34
7.4 Bidirectional Reflectance Distribution Function(BRDF)	37
7.5 Monte Carlo Integeration	40
7.6 Path Tracing	40
7.7 Sampling the Light	42
7.8 Some Side Notes	42
8 基于辐射度量学的光线追踪	43
References	44
A Source Code Example	45
B Task Division Example	46

Nomenclature

If a nomenclature is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.

Abbreviations

Abbreviation	Definition
ISA	International Standard Atmosphere
...	

Symbols

Symbol	Definition	Unit
V	Velocity	[m/s]
...		
ρ	Density	[kg/m ³]
...		

Chapter 1

光栅化

Keyword. 重心坐标

虎书中有这么一段话: The process of finding all the pixels in an image that are occupied by a geometric primitive is called rasterization; 即光栅化就是找到所有被几何原型所占据的所有像素点 (找到这些像素点之后再进行逐个渲染)。其中三角形就是最常用的 geometric primitive

1.1. 线性插值

二维或者三维的情况, 插入一个位置形成线段, 由 a, b 两点构成以 t 为参数的直线性质¹,

$$p = (1 - t)a + tb \quad (1.1)$$

这就是插值, 因为系数 t 和 $1 - t$ 是 t 的线性多项式, 所以称为线性插值。

另一种线性插值的情况, 对于 x 轴上一组点, x_1, x_2, \dots, x_n , 我们想建立一个连续函数 $y = f(x)$ 来插入这些点。, 使得 f 能通过每一个数据点, 即 $f(x_i) = y_i$ 。在线性插值中, 各点 (x_i, y_i) 由直线相连。

$$f(x) = y_i + \frac{x - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i) \quad (1.2)$$

1.2. 重心坐标

数学中, 重心坐标是由单形 (如三角形或四面体等) 顶点定义的坐标。重心坐标是齐次坐标的另一种。

设 v_1, \dots, v_n 是向量空间 \mathbf{V} 中一个单形的顶点, 如果 V 中某点 p 满足

$$(\lambda_1 + \lambda_2 + \dots + \lambda_n)p = \lambda_1v_1 + \lambda_2v_2 + \dots + \lambda_nv_n \quad (1.3)$$

¹这是一个 0 单纯形

那么我们称系数 $(\lambda_1, \dots, \lambda_n)$ 是 p 关于 v_1, \dots, v_n 的重心坐标。

重心坐标不是唯一的：对任何不等于零的 k , $(k\lambda_1, \dots, k\lambda_n)$ 也是 p 的重心坐标。但总可以取坐标满足 $\lambda_1 + \dots + \lambda_n = 1$, 称为正规化坐标。注意到定义式在仿射变换下不变, 故重心坐标具有仿射不变性。

三角形的重心坐标

在三角形情形中, 重心坐标也叫面积坐标, 因为 P 点关于三角形 ABC 的重心坐标和三角形 PBC , PCA 及 PAB 的 (有向) 面积成比例

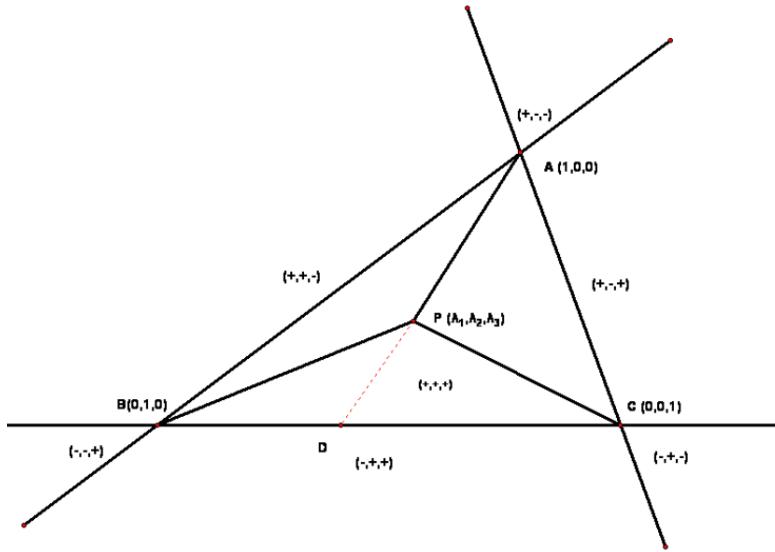


图 1.1: 三角形重心坐标

笛卡尔坐标系中的三角形面积

数学中常常用向量之间的“矩”来量化几何体的面积。考虑三角形 ABC 的面积

$$S = \frac{1}{2} \|\overrightarrow{AB} \times \overrightarrow{AC}\| \quad (1.4)$$

其中 $\overrightarrow{AB} = (x_1, x_2, x_3)$, $\overrightarrow{AC} = (y_1, y_2, y_3)$ 表示成行列式

$$S = \frac{1}{2} \left| \begin{array}{ccc} i & j & k \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{array} \right| \quad (1.5)$$

即

$$S = \frac{1}{2} \left| \begin{array}{ccc} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{array} \right| \quad (1.6)$$

重心坐标系

重心坐标系是一种仿射坐标系，在图形学中，经常希望为三角形的每个顶点赋予性质，如颜色，并将该性质的值插值到三角形中，实现这种要求的方法是利用重心坐标系

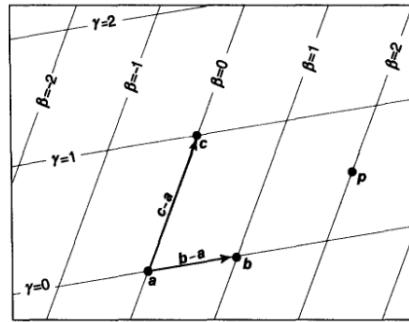


图 1.2: 重心坐标系

如上图所示，期坐标原点为 a ，从 a 到 b 和 c 的向量为基向量。于是 p 点就可以表示成

$$p = a + \beta(b - a) + \gamma(c - a) \quad (1.7)$$

重新安排顺序

$$p = (1 - \beta - \gamma)a + b + c \quad (1.8)$$

令 $\alpha = 1 - \beta - \gamma$ ，于是得到方程

$$p = \alpha a + \beta b + \gamma c \quad (1.9)$$

其中 $\alpha + \beta + \gamma = 1$ 。

重心坐标系的一大特点是，对于由 a, b, c 形成的三角形，当且仅当下列条件满足时，点 p 位于三角形内部

$$\begin{aligned} 0 < \alpha < 1 \\ 0 < \beta < 1 \\ 0 < \gamma < 1 \end{aligned} \quad (1.10)$$

1.3. 直线的光栅化

基于隐式方程绘制直线的常用算法是中点算法。直线的一般方程是

$$f(x, y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0 \quad (1.11)$$

我们先来考虑直线斜率 $k \in (0, 1)$ 的情况。在这种情况下，直线是以沿 x 轴方向的变化要快于沿 y 轴方向的变化。此时的直线是一条从左向右平缓上升的直线。不难想象的是，这时我们要光栅化一条直线的话，从左下角的起点出发，每次要“点亮”的像素的 x 坐标都是上一个像素的 x 坐标加 1，而 y 坐标则是根据某种条件，保持不变或者加 1，如此反复直至终点。

现在的关键就在于条件是什么？那么中点算法就是来解决这个问题的。

中点算法

中点算法的核心内容是：当我们要确定下一个要“点亮”的像素时，我们就把位于右侧和右上侧的这两个待选择的像素中心点的中点带入到直线方程中去，根据这个中点位于该直线的上侧还是下侧来决定“点亮”右侧的像素还是右上侧的像素。在保证 y 是正系数的情况下²，如果中点位于直线上方，则 $f(x, y) > 0$ ，如果中点位于下方，则 $f(x, y) < 0$ 。

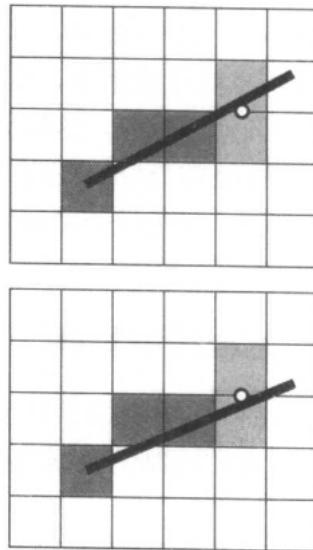


图 1.3: 中点算法的两种情况

证明

proposition 1.3.1: 在保证 y 是正系数的情况下³，如果中点位于直线上方，则 $f(x, y) > 0$ ，如果中点位于下方，则 $f(x, y) < 0$ 。

proof. 直线的参数方程为

$$t = \frac{x - x_0}{a} = \frac{y - y_0}{b} = \frac{z - z_0}{c} \quad (1.12)$$

其中 (a, b, c) 是和直线平行的一个向量。要证明

中点算法的改进—Bresenham 算法

中点算法有个很大的弊端就是每循环一次就要将中点带入方程求解并判断一次。我们有必要每次都求解方程吗？能不能只求解一次方程就可以呢？这就引入了中点算法的增量形式—Bresenham 算法。

因为中点算法首先是将两个像素点中点代入方程，然后计算出 $f(x, y)$ 的正负，但是我们发现，中点之间的增量都是固定的，这意味着我们只要求出来一个中点相对直线的位置，下一个中点的

²如果 y 不是正系数，则情况相反，例如 $f(x, y) = x - y + 1$ ，点 $f(0, 0) = 1 > 0$ 但是位于直线下方

³如果 y 不是正系数，则情况相反，例如 $f(x, y) = x - y + 1$ ，点 $f(0, 0) = 1 > 0$ 但是位于直线下方

位置只需要加上固定增量即可。增量存在如下关系

$$\begin{aligned} f(x+1, y) &= f(x, y) + (y - y_0) \\ f(x+1, y+1) &= f(x, y) + (y - y_0) + (x - x_0) \end{aligned} \quad (1.13)$$

1.4. 三角形光栅化

v、为什么要以三角形的光栅化为例呢，因为三角形是最基本的多边形，大部分的模型都是用一个个三角形表示，任意的其它多边形其实都可以转化成多个三角形的形式。

如下图所示，如何将三角形绘制到屏幕上？

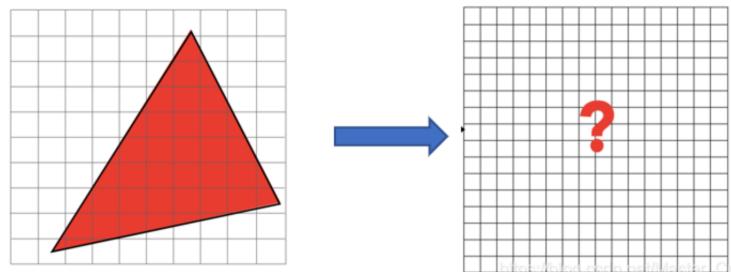


图 1.4: 三角形光栅化

首先需要进行采样，采样的意思就是对给定的函数进行离散化，比如上图中，要实现一个判断屏幕上的像素是否在三角形内的函数，然后将三角形顶点构成的最大长方形区域内所有点作为函数的输入，判断这些点是否在三角形内，如果在三角形内，就将屏幕上的像素点进行点亮。这个过程就是采样。其他采样的例子包括对视频进行时间上的采样，这样就可以得到视频中的某几帧画面

判断像素点是否在三角形内部

利用同侧向量积可以判断像素点是否在三角形内部

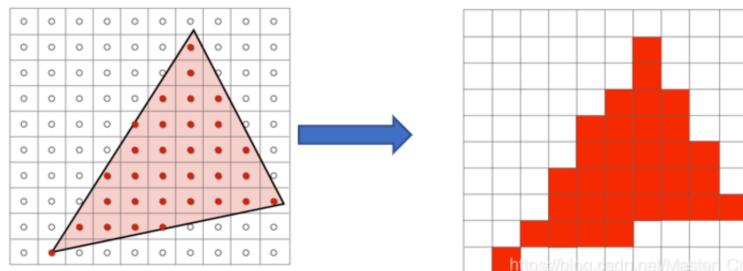


图 1.5: 采样后的三角形

1.5. 反走样

抗锯齿

锯齿产生的原因就是因为信号的变化频率高，而相应的采样频率低。就三角形边缘不规则的情况来说，因为三角形的边上是无限多个点，而用有限个方块去逼近无限多个点的三角形的边，所以当然会产生不规则的锯齿，硬件的解决办法一是可以加大屏幕的分辨率，使得像素变小，从而可以得到更多个有限的方块去逼近三角形。而软件的方法就是加入抗锯齿算法。无论是硬件方法还是软件方法，都不能完全解决锯齿问题，只能缓解锯齿问题，直到人眼察觉不出来

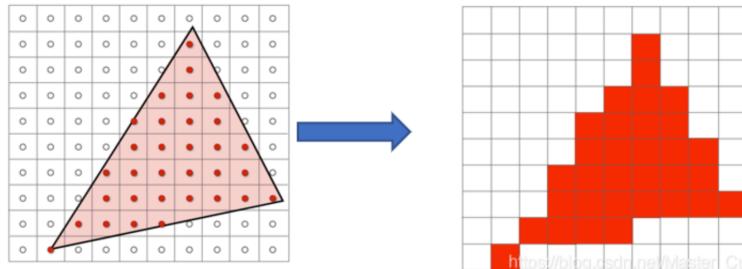


图 1.6: 采样后的三角形

利用平滑算子进行模糊三角形边界

平缓算子和图像卷积，在时域或频域进行。常用的平滑算子是计算邻域内的平均值。

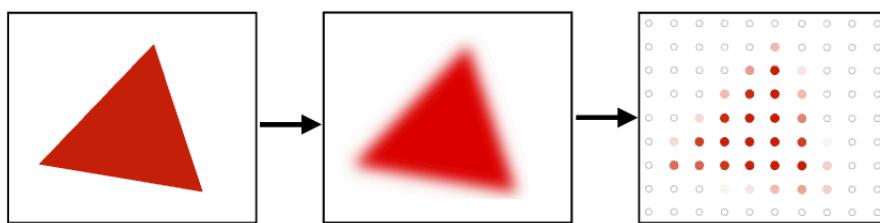


图 1.7: 经过平滑算子之后在进行光栅化

SSAA (super sampling antialiasing, 超采样抗锯齿算法)

它的原理非常简单，就是在渲染时将要输出的分辨率提升 x 倍，比如要输出 1920x1080 的分辨率到屏幕上，开启 SSAA 2x，那么内部渲染时的分辨率就是 3840x2160，然后 DownSampling 到 1920x1080 上，自然在许多纹理边缘上就显得平滑许多。

SSAA 的思想就是将一个像素分解成 $2*2$, $3*3$, $4*4$，然后判断分解后像素中的 4 个或者 9 个或者 16 个像素点是否在三角形内，如果在像素内，就将点绘制成红色，然后将这 4 个或者 9 个或者 16 个像素点的像素值求平均值，最后作为整个像素点的像素值。

例如一个 $2*2$ 的过程

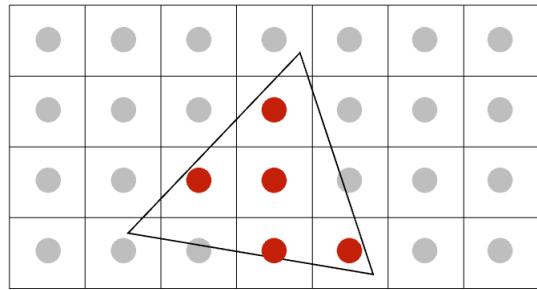


图 1.8: 原始采样

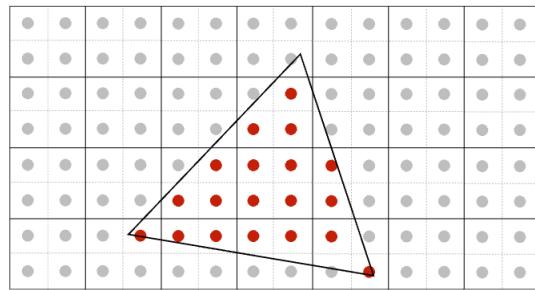


图 1.9: 细分每个像素的子像素

如果子像素在三角形内部则赋予红色。

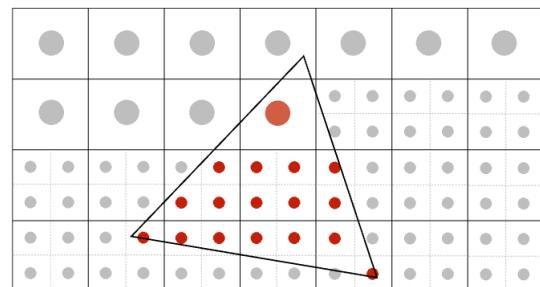


图 1.10: 求大像素内每个小像素平均值

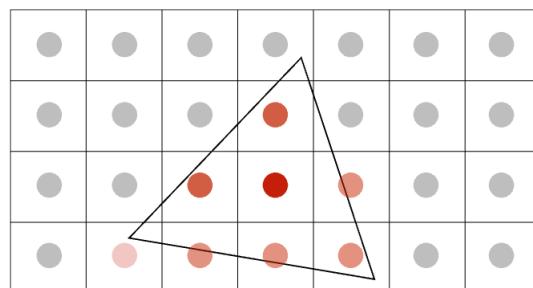


图 1.11: 平均每个像素

MSAA (multi-sampling antialiasing, 多采样抗锯齿算法)

MSAA 只对于多边形的边缘进行抗锯齿处理。比如一个红色的圆，只对圆周作抗锯齿多重采样计算，但是圆周以内的部分则不会处理。这种方式下的画面锯齿得到了一定的抑制，而抗锯齿需要的资源也大幅下降到可接受的范围中。所以 MSAA 也逐渐成为目前被使用的最多的抗锯齿技术。

MSAA 的思想就是如果有子采样像素在三角形内，那么，就认为像素点需要着色，采样后，看有多少个子采样像素点在三角形内，然后计算在三角形内的子采样像素占所有采样点的比重，最后把整体的像素值（比如红色）乘以该比重，就得到了最终绘制的像素值。

1.6. 对图形深度的处理

画家算法

“画家算法”表示头脑简单的画家首先绘制距离较远的场景，然后用绘制距离较近的场景覆盖较远的部分。画家算法首先将场景中的多边形根据深度进行排序，然后按照顺序进行描绘。这种方法通常会将不可见的部分覆盖，这样就可以解决可见性问题。在有些场合下，画家算法可能无法解决可见性问题。例如下面重叠三角形

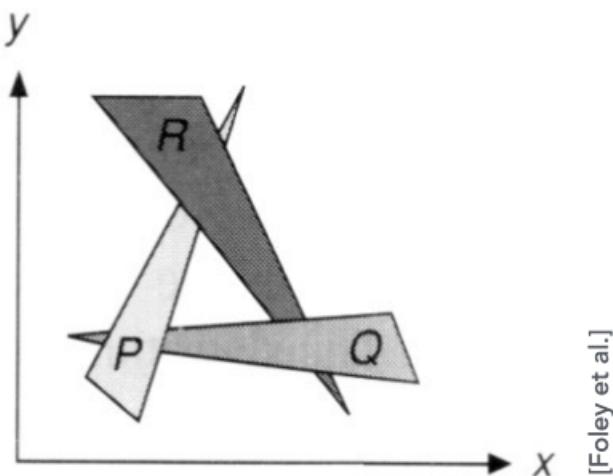


图 1.12: 画家算法无法按照先后顺序表示图中重叠物体

深度缓冲算法 (Z-Buffer)

算法的过程是：

1. 首先分配一个数组 $buffer$ ，数组的大小为像素的个数，数据中的每个数据都表示深度，初始深度值为无穷大。
2. 随后遍历每个三角形上的每个像素点 $[x,y]$ ，如果该像素点的深度值 $z < zbuffer[x,y]$ 中的值，则更新 $zbuffer[x,y]$ 值为该点深度值 z ，并更新该像素点 $[x,y]$ 的颜色为该三角形上像素点上的颜色。

说白了就是每个像素点去对比深度，伪代码表示

```

1   for(each triangle T)
2       for(each sample point (x,y,z) in T)
3           if(z<zbuffer[x,y])
4               framebuffer[x,y]=rgb;
5               zbuffer[x,y]=z;
6           else
7               ;

```

例如下面图， R 表示无穷大，红色和紫色为需要绘制的三角形。

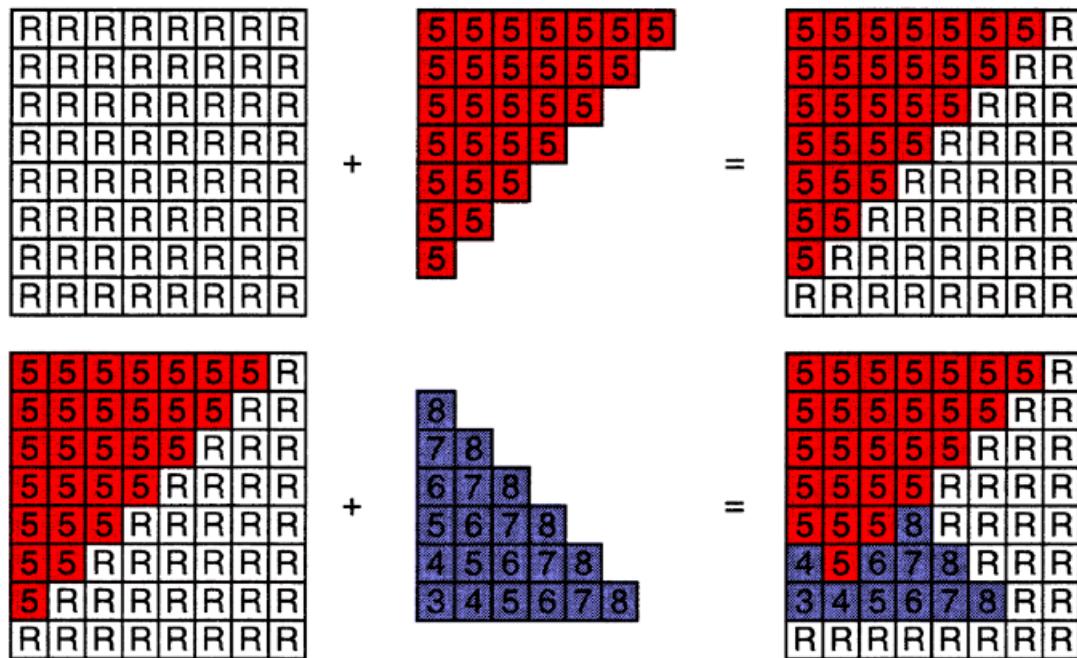


图 1.13: zbuffer algorithm

Chapter 2

着色

2.1. 平方反比定律

光线从光源到 r 位置光的强度满足平方反比定律

$$\text{Intensity} = \frac{I}{r^2} \quad (2.1)$$

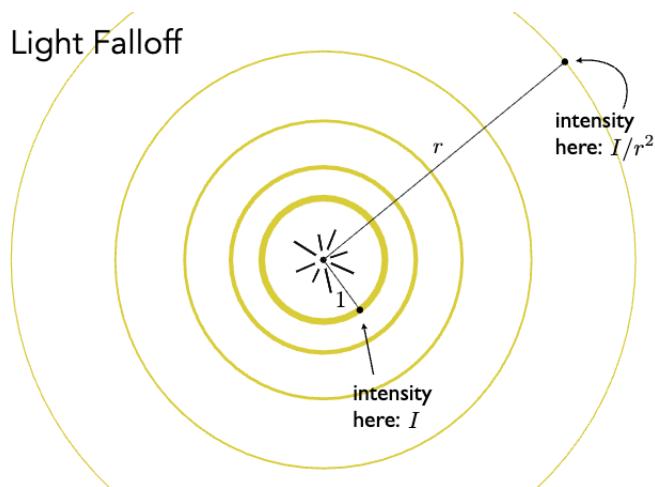


图 2.1: 光强的平方反比定律

光线经过反射被人眼观测，但是传播到 r 处的光并非所有能量都被人眼接收，由于接受光线的角度不一样，被物体吸收的光强也会衰减。

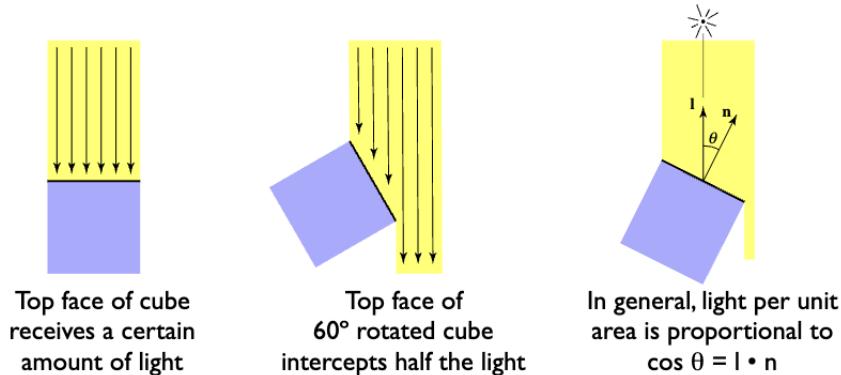


图 2.2: 光强的吸收

2.2. Blin-Phong 反射模型

Blin-Phong 反射模型描述了由漫反射 (*Diffuse*)、镜面反射 (*Specular*)、环境光反射 (*Ambient*) 组成。主要的表述形式如下

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + k_d (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p \end{aligned} \quad (2.2)$$

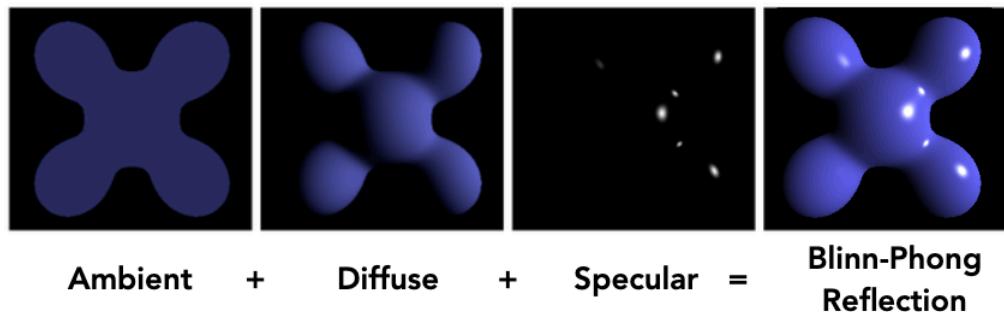


图 2.3: Blinn-Phong 反射模型

我们下面分别解释每一项的意思，假设 I 是入射光的单位向量， n 是平面法向量， v 是观测方向的向量， k_d 系数在 Blinn-Phong 光照模型中是描述材质表面对漫反射光的吸收程度的参数，它影响了表面在漫反射光照下的亮度和颜色 k_s 通常与光源的位置、观察者的位置以及材质的光泽程度等因素相结合，用来计算镜面反射光的强度。增加 k_s 的值会使得材质表面的镜面反射更加明显，反之则会减弱镜面反射的效果。 k_a 是 Blinn-Phong 光照模型中的环境光系数，用于描述材质表面对环境光的反射程度。环境光是场景中所有光源的综合效果，它无处不在，没有特定的入射方向，而是均匀地照亮了整个场景。环境光系数 ka 表示了材质表面对环境光的反射程度。

漫反射 L_d

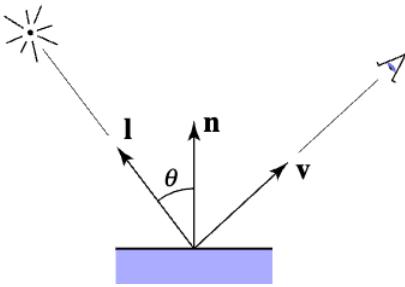


图 2.4: 反射模型

当光线垂直射向物体表面，接收到的光线亮度或者强度是 100，当光线和物体表面成角 60 度时，原来的六根光线只有三根光线被接收到，原因是因为物体表面的法线向量和入射方向的夹角为 60 $^\circ$ ， $\cos 60^\circ \times 6$ 就是三根光线。因此，接收到的光线的亮度或者强度与入射光线与平面法线的夹角有关，所以接收到的光照强度为

$$(I/r^2)\max(0, \mathbf{n} \cdot \mathbf{l}) \quad (2.3)$$

L_d 是漫反射的光强， $I/r^2\max(0, \mathbf{n} \cdot \mathbf{l})$ 为到达着色点 (*shading point*) 的光照能量， k_d 是物体吸收光照的系数，

镜面反射

镜面反射就是高光，光线经过完全反射的向量和观测者方向向量存在一定的角度，如下图所示

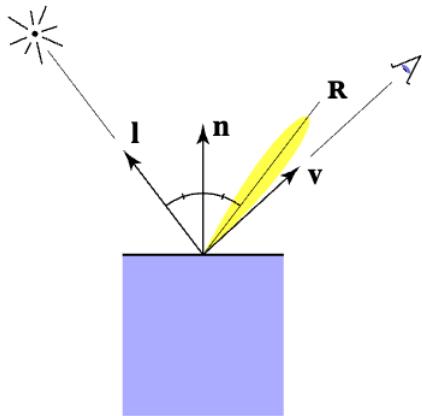


图 2.5: 高光

根据入射光线和法线向量，可以得到反射光线的方向，根据反射光线的方向与视角方向的夹角计算人眼接收到光线的强度，这种高光模型称为 *phong* 模型

Blinn-Phong 对该模型进行了改进，根据光线方向和视角方向引入了半程向量，如下图所示，半程向量与法线向量的夹角的 cos 就间接表示了视角和反射光线的夹角的 cos

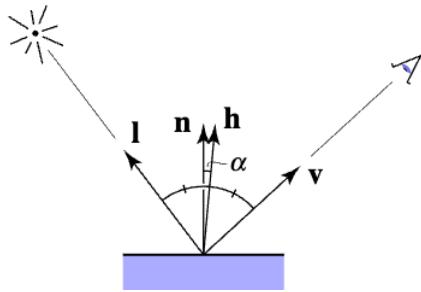


图 2.6: 半程向量

所谓半程向量定义如下

$$h = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|} \quad (2.4)$$

直观理解就是 \mathbf{l} 向 \mathbf{v} 方向旋转夹角的一半后进行归一化得到的向量。这时 \mathbf{n} 和 \mathbf{h} 的夹角等于 \mathbf{R} 和 \mathbf{v} 的夹角。

所以镜面反射的光强可以表示为

$$\begin{aligned} L_s &= k_s (I/r^2) \max(0, \cos\alpha)^p \\ &= k_s (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p \end{aligned} \quad (2.5)$$

为什么要加上一个指数 p 呢？之所以要加一个指数项，也是为了符合人们的直觉，可以这样想，比如当我们用镜子去晃别人的眼睛，镜子稍微偏转几度，就可能让被晃的人感觉光线强度变化很大，所以，加了一个指数，就是让角度稍微一变化，光线强度就会由剧烈的波动， \cos 的指数函数图如下

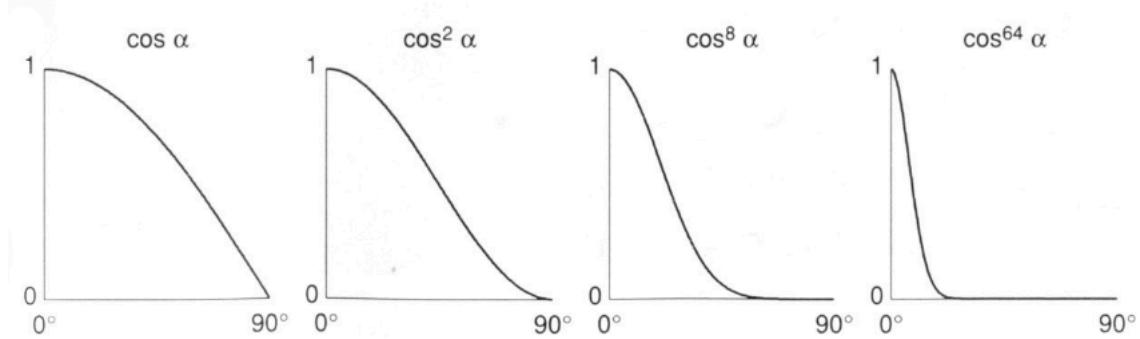


图 2.7: 镜面反射幂的意义

环境光反射

由于环境光和入射光和观测方向都没关系，因此只需乘一个常数 k_a

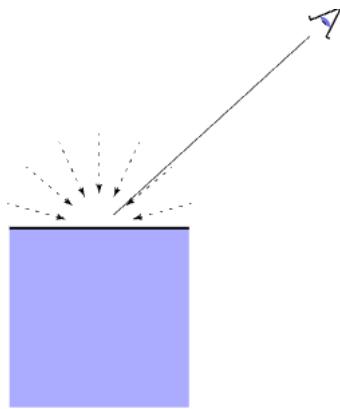


图 2.8: 环境光反射

2.3. 着色频率

通过光照模型可知，最终的光照结果和光照点的法线向量关系很大。所以，根据不同的法线向量，就有不同的着色方法。在图形学中，法线分为：面法线、顶点法线和像素法线。光照和这三种法线相互作用，就有了三种不同的渲染频率。

面着色 (Flat shading)

面着色就是利用物体面表上的每个三角形的法线向量进行光照强度的计算，进而决定物体表面的颜色。该方法效果较差（物体表面不光滑，棱角较多）

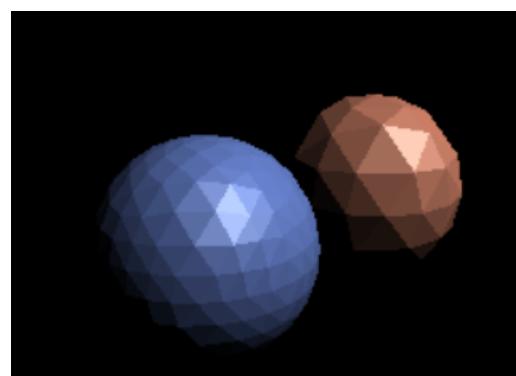


图 2.9: 面着色

顶点着色 (Gouraud shading)

顶点着色就是利用每个顶点的法向量来计算光照强度，得到三个顶点的颜色后，通过插值的方式来对面内的点的像素进行计算

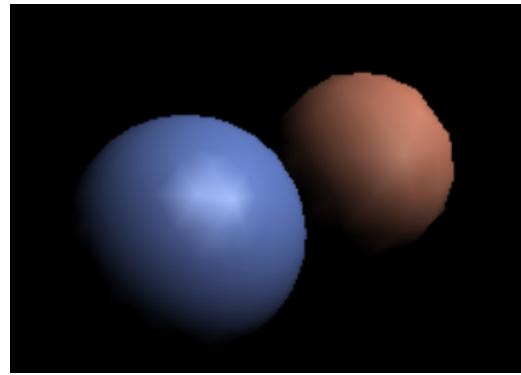


图 2.10: 顶点着色

首先如何计算每个顶点的法线向量? 方法就是将所有共享该顶点的三角形面的法线向量计算出来, 之后, 将这些法线向量相加求平均, 如下图所示

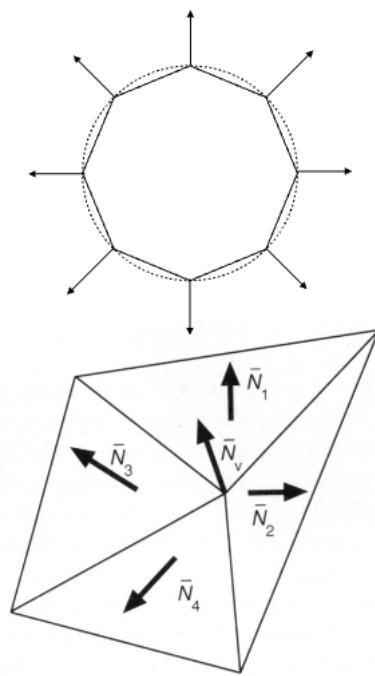


图 2.11: 顶点法线

$$N_v = \frac{\sum_i N_i}{\|\sum_i N_i\|} \quad (2.6)$$

这样, 就能得到每个顶点的法线向量, 继而根据光照模型就能算出每个顶点的像素,

插值

假设经过光照模型, 已经得到了顶点的像素, 我们通过插值的方法补充顶点之间像素。两个顶点之间直线的插值

$$p = (1 - t)a + tb \quad (2.7)$$

只要把 a, b 两点的顶点像素数据代入，通过调节 t 的变化，便能生成一条线的像素数据。
对于三角形，如何进行插值呢？

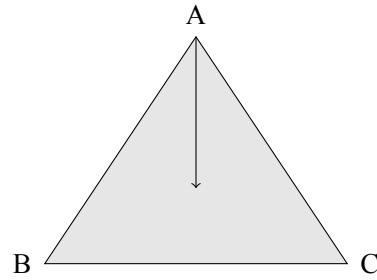


图 2.12: 三角形

对于三角形 $\triangle ABC$ 内部任意一个点 P ，由于三角形是一个 2- 单形，必然存在 α, β

$$\overrightarrow{AP} = \alpha \overrightarrow{AB} + \beta \overrightarrow{AC} \quad (2.8)$$

展开有

$$P - A = \alpha(B - A) + \beta(C - A) \quad (2.9)$$

设 $\gamma = 1 - \alpha - \beta$ 整理得到

$$P = \gamma A + \alpha B + \beta C \quad (2.10)$$

像素着色 (Phong shading)

像素着色是着色频率最高的方式，直接对每个像素点进行着色。

Chapter 3

图形渲染管线

Chapter 4

纹理映射

纹理映射表示物体表面细节的一幅或几幅二维图形，也称纹理贴图 (Texture Mapping)，是将纹理空间中的纹理像素映射到屏幕空间中的像素的过程，即将纹理贴图贴到三维物体表面的过程。作用：模拟物体表面细节。

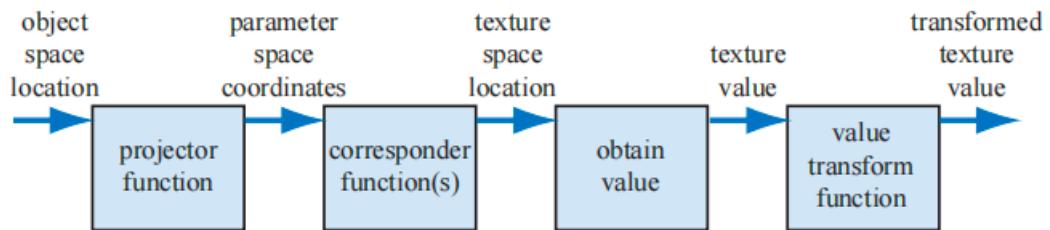


图 4.1: 纹理映射管线 1

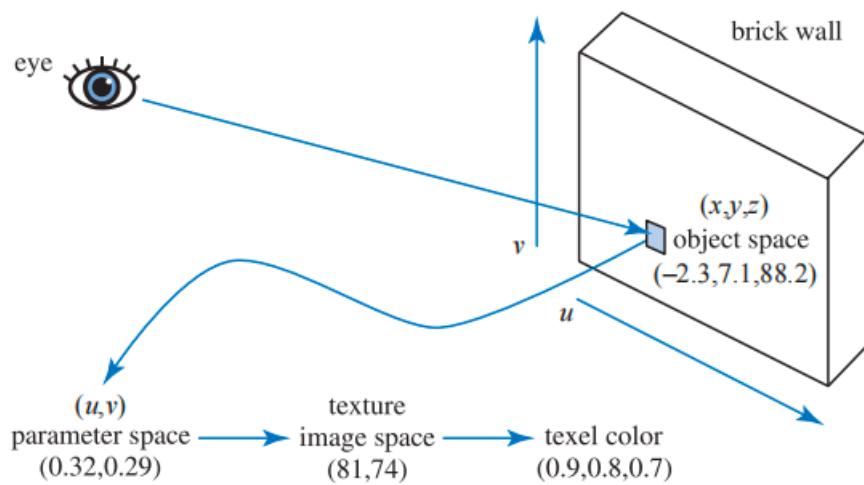


图 4.2: 纹理映射管线 2

4.1. 投影映射

之所以叫投影映射，是因为主要有两种方法可以将三维的空间坐标点转化为二维的纹理坐标点：Projector 和 UV Mapping。

$$\phi : S \rightarrow T \quad (4.1)$$

$$(x, y, z) \mapsto (u, v)$$

集合 T 称为纹理空间， $(u, v) \in [0, 1]^2$ ， uv 也称为纹理坐标， S 是物体的坐标系。

Projector

对于一些简单的几何体，通常用投影的方式

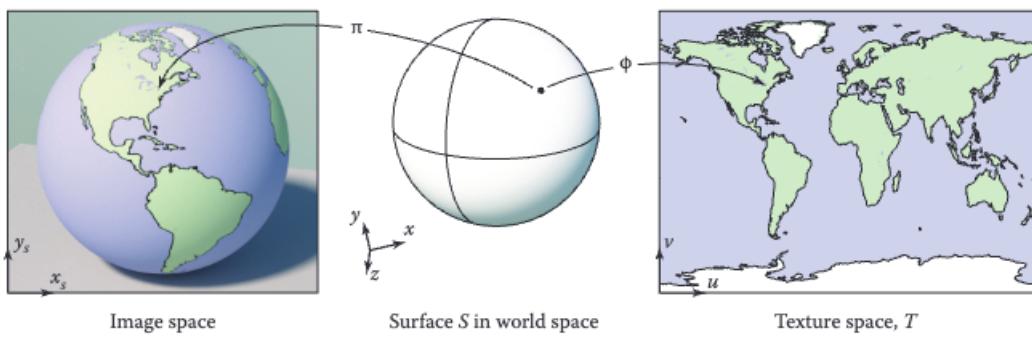


图 4.3: 纹理映射

这种将矩形地图纹理均匀贴到球表面的投影函数称之为：Spherical 形式

此外还有 Plane, Cubic 和 Cylindrical，下图总结了四者差异（一张红绿相间的纹理贴到不同简单几何体的方式）：

uv mapping

Projector 只适用于简单情况，对于更复杂的几何体贴图，往往需要用到 UV Mapping：用于将 3 维模型中的每个顶点与 2 维纹理坐标一一对应。UV map 则需要建模师精心制作。

也就是说一般情况下物体到纹理空间的纹理映射是确定好的。

参数化

自动化地把复杂纹理和物体一一对应，在几何上是一个研究方向。

4.2. 变换函数

4.3. 用重心坐标做插值

为了兼顾功耗，很多操作都在三角形上的顶点做计算，对于三角形内部的点则通常采用插值的方法，重心坐标的作用就是脱离某个特定坐标系去描述，方便去做插值。插值的内容可以是纹理坐标 uv 、颜色、法向量等等。

为了描述一个点的位置，只需要通过下面方式描述

$$(x, y) = \alpha A + \beta B + \gamma C \quad (4.2)$$

其中如果点在三角形内，则满足 $\alpha + \beta + \gamma = 1$ ，这样就不需要通过坐标系去描述。

如何计算重心坐标

首先计算重心坐标表达式中的系数¹

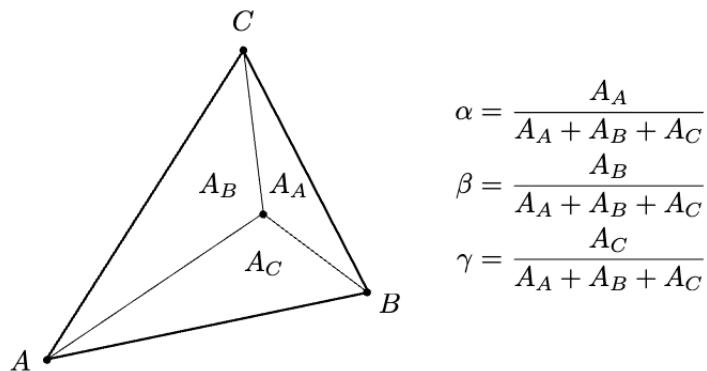


图 4.4: 重心坐标划分三角形区域

代入线性组合

$$\alpha = \frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)}$$

$$\beta = \frac{-(x - x_C)(y_A - y_C) + (y - y_C)(x_A - x_C)}{-(x_B - x_C)(y_A - y_C) + (y_B - y_C)(x_A - x_C)}$$

$$\gamma = 1 - \alpha - \beta$$

图 4.5: 重心坐标计算

¹三角形重心可以把三角形分成三个等面积的三角形。

利用重心坐标做插值

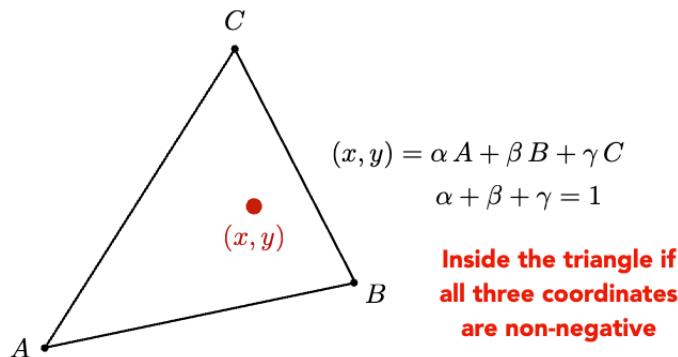


图 4.6: 重心坐标 1

重心坐标不具有投影不变性

4.4. 纹理放大

纹理放大 (Texture Magnification), 纹理本身太小, 纹理的分辨率不够, 图像上的多个像素在渲染时取纹理映射上取到了同一个点, 会有明显的方块状。我们将纹理上的每个像素称为 texel

纹理小的情况

小纹理到大分辨率屏幕之间映射会出现问题。例如 256*256 的纹理向 4k 分辨率屏幕映射的时候, 原本整数的像素被压缩成小数。

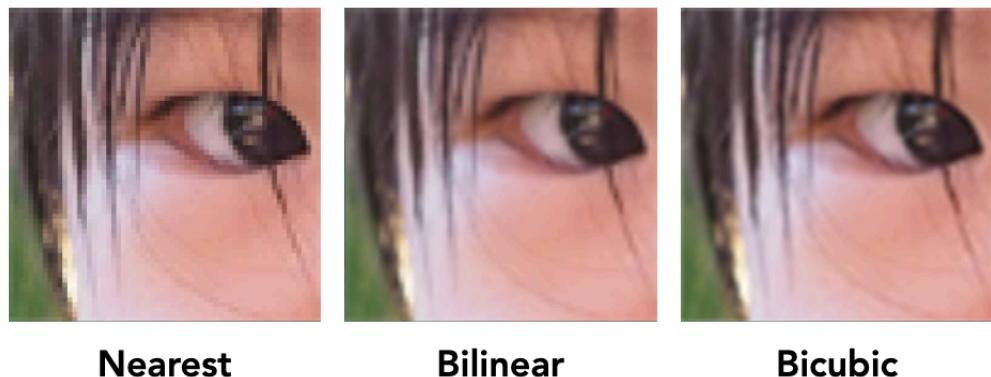


图 4.7: 纹理放大及各种解决方式的效果

小纹理到大像素空间之间的映射本质上是高频采样低频的纹理元素, 多次在一个纹理元素上进行采样造成的, 因此需要对每个纹理元素之间进行插值做补正。

双线性插值

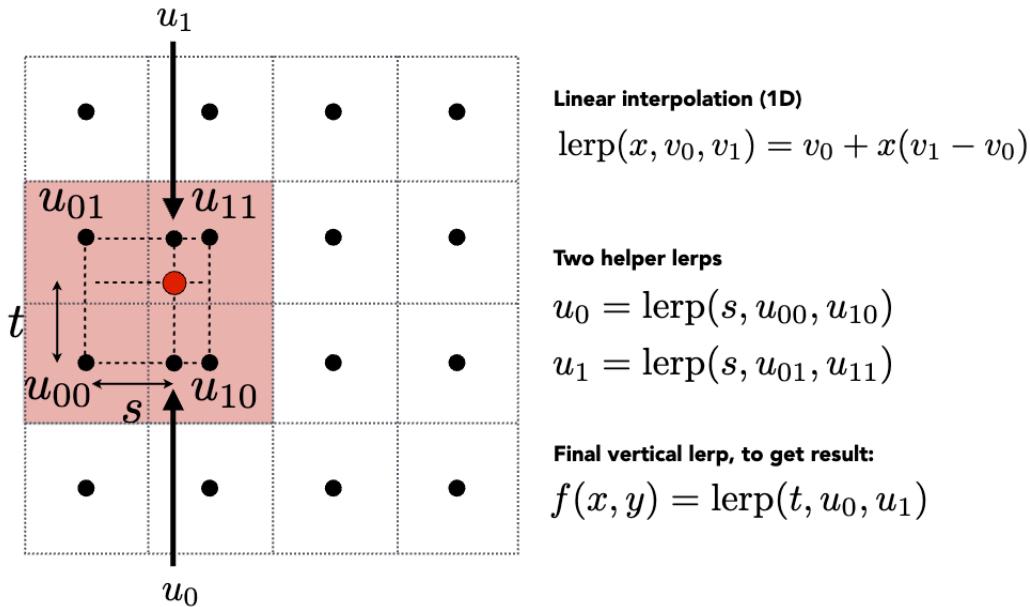


图 4.8: 双线性插值补正

所谓双线性插值就是在 xy 两个方向上做独立地做线性插值操作。

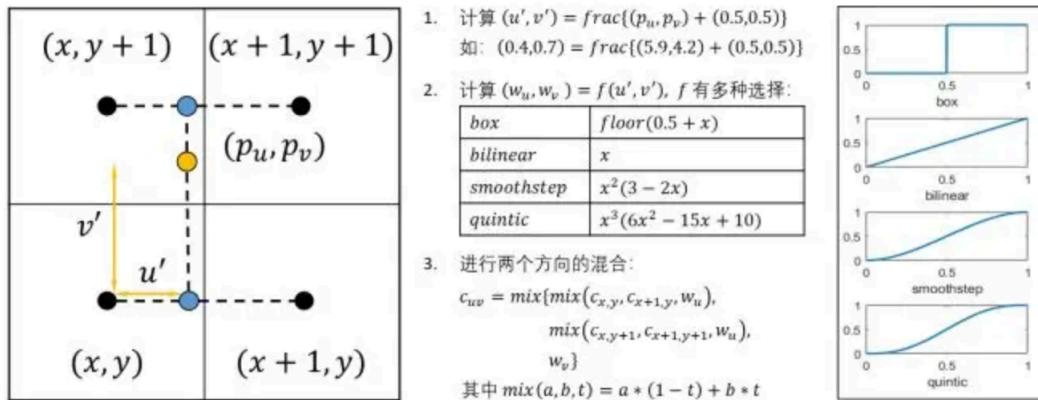


图 4.9: 双线性插值流程

纹理大的问题

纹理过大，导致一个 pixel 覆盖了多个 texel，会使生成的图像产生更明显的失真。近处产生锯齿，远处摩尔纹²。为什么远处会产生摩尔纹呢？这种现象是光栅化的算法导致的。我们知道，一个三角形有顶点坐标和纹理坐标，纹理坐标范围是 [0-1][0-1][0-1]。光栅化的过程就是把三角形在

²在光学中，摩尔纹（Moire）是一种视觉上的干涉现象，通常在两个或多个周期性结构相互叠加时出现。当纹理较大或者具有明显的周期性结构时，往往会导致摩尔纹的产生。

摩尔纹的产生是由于两个周期性结构之间的相对位移或角度差异导致的光学干涉。当两个结构叠加在一起时，它们的周期性纹理会相互干扰，产生了一系列新的干涉纹。这些干涉纹可以表现为亮暗条纹、彩色条纹或其他形式的图案。

在纹理较大的情况下，由于纹理的周期性结构比较明显，当它们与另一个周期性结构相叠加时，干涉现象更加明显，从而产生了更为明显的摩尔纹效应。

屏幕上以一个个像素的形式显示出来，插值计算三角形内部每个像素的顶点的数据，包括常见的深度值与纹理坐标。如果这个三角形距离 camera 近，也就是说在屏幕上占了较多的像素，那么相邻两个像素的纹理坐标是接近的，这样通过纹理坐标获得纹理贴图上的纹素值也是接近的，这样这两个像素看起来比较平滑，视觉上不突兀，同时 gpu 读取也快速，因为大部分纹素读取是在 cache 中。而如果这个三角形距离 camera 较远，也就是在屏幕上只占了很少的像素，这种情况就是一个小物体应用了一个大纹理，光栅化后，相邻两个像素的纹理坐标会差别很大，读出来的纹素也会差别很大，会很突兀，尤其是 camera 移动时特别明显，产生闪烁，火花现像，除此之外，gpu 读取性能也很低效，因为两个相邻的像素所对应的纹素，一个可能在 cache 中，另一个还没有加载到 cache 中。

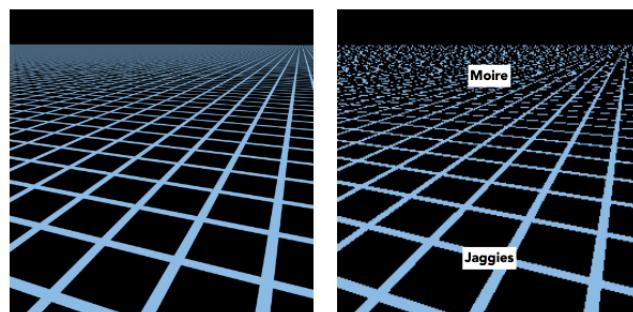


图 4.10: 纹理大的情况

超采样

如果一个像素不足以代表纹理的性质，那么很自然地方式是可以用超采样去解决，但是超采样是一种点查询解决方式，硬件开销大。

所以能否避免采样？方式就是取这个区域内的平均值。

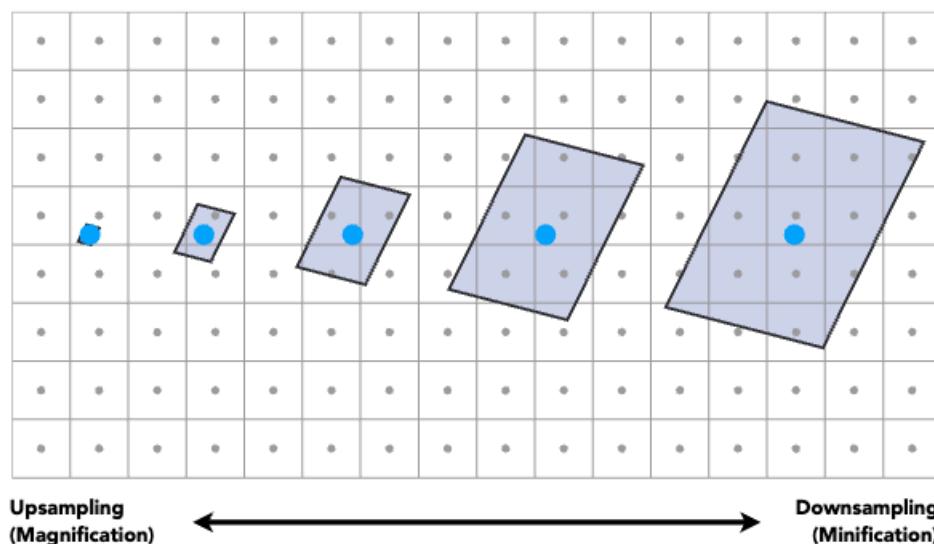


图 4.11: footprint

Mip-Map

Mip-Map 是一种范围查询技术，其思想是基本思想就是，建立一系列不同尺寸的多级纹理，纹理采样时，计算对应的细节等级，再利用三线性插值计算。

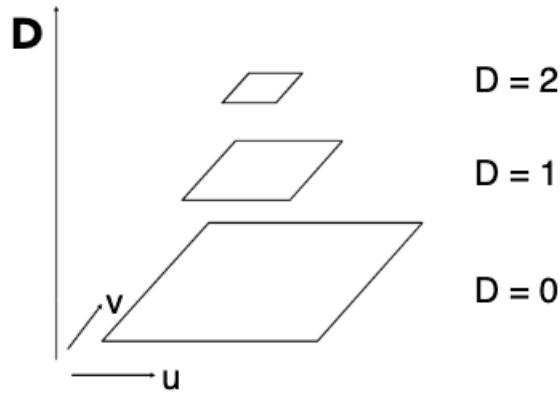


图 4.12: Mip-Map 纹理分层

各向异性过滤

4.5. 帧插值技术

帧插值

帧采样

光流法

4.6. 环境光映射

可以用纹理去描述环境光

Spherical Environmental map

环境光记录在球上会产生扭曲现象。

Cube mapping

Bump mapping

凹凸贴图，人为地制造法线，几何形体不变通过复杂的纹理体现物体表面的凹凸性。

如何获取这样的法线？差分获取梯度，法线垂直于梯度。

Displacement mapping

改变三角形顶点的位置，代价是三角形足够细。

4.7. 纹理的其他应用

三维空间中的噪声函数

提前做着色

环境光遮蔽

体渲染

Chapter 5

几何

5.1. 几何的表示

隐式

显式

参数化的显式表示

5.2. 不同表示方法的意义

Constructive Solid Geometry

基本几何体的交并差来组合描述。

距离函数

不描述表面，描述最近距离

SDF

blend

水平集

分型

点云

最简单的显式表示方法。

1

¹Object file：顶点，法线，纹理坐标分开表示，组成模型，.obj 格式

5.3. 曲线

三维模型沿着曲线去运动

贝塞尔曲线与 de Casteljau 算法

贝赛尔曲线代数形式

$$\mathbf{b}^n(t) = \mathbf{b}_0^n(t) \sum_{j=0}^n$$
(5.1)

伯恩斯坦多项式

其实就是二项分布

贝塞尔曲线的性质

凸包性质贝塞尔曲线上任何点一定都在给定控制点形成的凸包之内。

Piecewise Bezier Curves

逐段的贝塞尔曲线

连续性

函数的连续性和几何的连续性有区别么?

C^1 连续, C^∞ 连续

B-splines

局部性

非均匀有理 B-splines——NURBS

5.4. 曲面

Bezier Surfaces

双线性插值的思想，在两个方向分别应用贝塞尔曲线。

5.5. Mesh Operations

网格操作有三种

1. 网格细分 (*Mesh subdivision*);
2. 网格简化 (*Mesh simplification*);
3. 网格正规化 (*Mesh regularization*);

网格细分

我们介绍两种网格细分方法

Loop Subdivision

Loop 细分是一种流行的曲面细分方法，用于创建平滑的曲面模型。它是细分曲面技术的一种，由 Charles Loop 于 1987 年提出，并在图形学领域得到广泛应用。

Loop 细分的基本思想是通过迭代地对初始多边形网格进行细分，逐步增加网格的细节和平滑度。在每一次细分步骤中，原始多边形被分割成更小的子多边形，而新的顶点则是通过对原始顶点进行加权平均来生成的。

基本思想是通过迭代地对初始多边形网格进行细分，逐步增加网格的细节和平滑度。在每一次细分步骤中，原始多边形被分割成更小的子多边形，而新的顶点则是通过对原始顶点进行加权平均来生成的。

Loop 细分的算法包括以下步骤：

1. 计算新顶点位置：对于每个顶点，计算其新的位置，通常是其相邻顶点的加权平均。
2. 更新边和面：根据新的顶点位置更新原始网格的边和面。
3. 细分：根据新的边和面，生成新的细分网格，以便下一次迭代。

通过多次迭代上述步骤，可以逐渐增加网格的细节，并使曲面逼近更加光滑。Loop 细分通常用于创建高质量的曲面模型，如角色模型、汽车外壳等，以及在计算机动画和游戏开发中实现细致的渲染效果。

²

Catmull-Clark Subdivision

奇异点：顶点的度不为 4 的点。

Convergence : Overall Shape and Creases

曲面简化

边坍缩 二次误差度量

从二次度量误差最小的边做边坍缩

²图论：顶点的度

Chapter 6

光线追踪

6.1. Shadow Mapping

经典的 shadow mapping 只能处理点光源
点光源下如何生成阴影?

1. Render from light 从光源看向场景记录看到的深度。
2. Projec to light 投影回光源

阴影图的问题，只能做硬阴影，做不了软阴影，物理上区分阴影：本影和半影

6.2. 为什么需要光线追踪

光栅化着色只考虑光线至弹射一次，难以解决光线弹射很多次的情况。
光栅化快但是质量低

6.3. 光线

基本假设：光线沿直线传播，光路可逆

6.4. Ray Casting

从摄影机连一条线穿过像素到达物体上的点，在连接光源和点，由于光路可逆性，这就是光源到眼睛的光的路径。

Pinhole Camera Model

6.5. Recursive(Whitted-Style) Ray Tracing

弹射后的能量损失

6.6. 光线与物体表面交点怎么求?

Ray Equation

$$\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \quad 0 \leq t \leq \infty \quad (6.1)$$

Ray Intersection With Sphere

Ray Intersection With Implicit Surfaces

Ray Intersection With Triangle Mesh

在图形内发射一条光线，和图形边界的交点一定是奇数，在图形外交点一定是偶数。

由于三角形一定在一个平面内，首先求光线和平面求交，然后判断这个点是否在三角形内部。

平面方程形式如下

$$\mathbf{p} : (\mathbf{p} - \mathbf{p}') \cdot \mathbf{N} = 0 \quad (6.2)$$

其中 \mathbf{N} 为平面的法线。

下面求与平面的交点

完成了上面一步，下面就是怎么判断这个点在三角形内部，怎么判断这个点在三角形外部呢？

Möller Trumbore Algorithm，用重心坐标

$$\vec{\mathbf{O}} + t\vec{\mathbf{D}} = (1 - b_1 - b_2)\vec{\mathbf{P}}_0 + b_1\vec{\mathbf{P}}_1 + b_2\vec{\mathbf{P}}_2 \quad (6.3)$$

写成矩阵形式

6.7. Accelerating Ray-Surfaces Intersection

怎么样去加速？

Bounding Volumes

思路是如果一个光线连包围盒都碰不到，那根本不可能碰到里边的物体

轴对齐包围盒 (*Axis-Aligned Bounding Box AABB*)

如何判定光线和盒子有没有交点呢？

只有当光线进入盒所有的三个对面，那就是进入了盒子，只要离开了任意一个对面，那就算离开。

光线和 AABB 有交点当且仅当

$$t_{enter} < t_{exit} \quad and \quad t_{exit} >= 0 \quad (6.4)$$

6.8. Uniform Spatial Partitions(Grids)

Ray-Scene Intersection

只需要做若干次光线和盒子求交
 格子的数量不能太稀疏，也不能太密集
 “Teapot in a stadium” problem

6.9. Spatial Partitions

在稀疏的地方不需要这么多个格子。

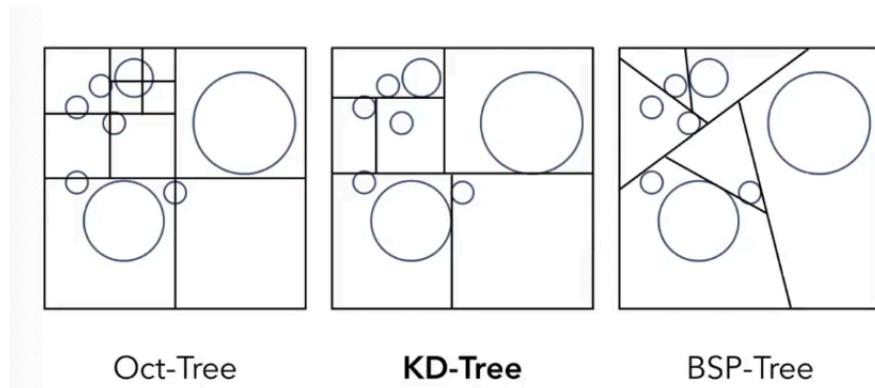


图 6.1: 空间划分方法

KD-Tree

kd-tree 是在光线追踪之前做好。
 每次找到格子，总是沿着某一个轴砍一刀，就砍一刀。

KD-tree Pre-Processing

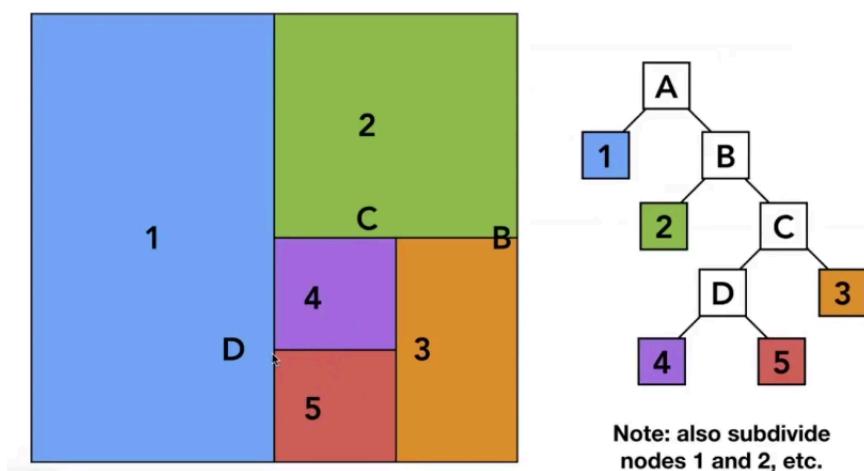


图 6.2: KD-tree

对于任意一个节点，沿着哪个轴划分，然后划分在哪儿。

Traversing a KD-Tree

KD-Tree 的问题

怎么判定三角形的交集

6.10. Object Partition and Bounding Volume Hierarchy

Bounding Volume Hierarchy(BVH)

解决 KD-Tree 三角形的问题。完全省去三角形与包围盒求交点
把三角形分为两部分，重新设置包围盒。

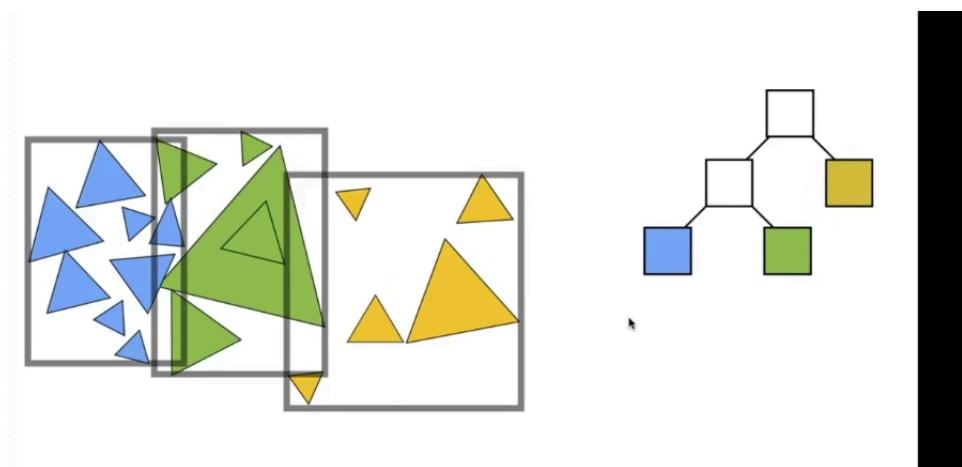


图 6.3: BVH

BVH 的性质：一个物体只在一个盒子里

BVH 对空间的划分不是完全划分开，Box 可以相交。

Summary:building BVH

技巧：快速划分算法，受到快速划分启发。

技巧 2：永远沿着最长的轴，中位数

BVH Traversal

Chapter 7

基于辐射度量学的光线追踪

Whitted style ray tracing 并不能精确地给到正确的结果，精准给光物理量的方法。

7.1. Radiant Energy and Flux(Power)

辐射能量 (radiant energy)

在辐射度量学中，最基本的单位是辐射能量，辐射能量 Q 是以辐射的形式发射、传播或者接收的能量。每个光子都携带一定的能量，这个能量正比于频率 v

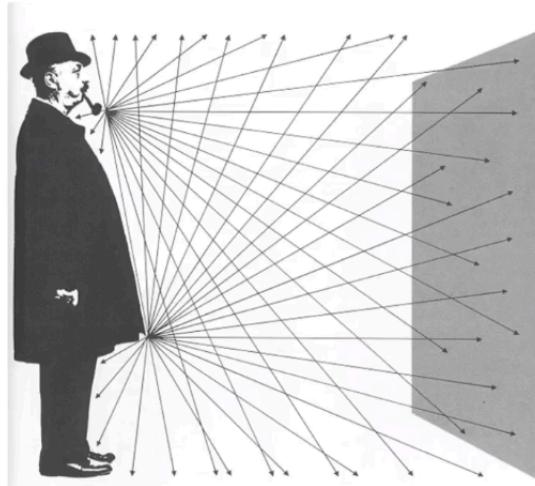
$$Q[J = Joule] = hv \quad (7.1)$$

其中 $h = 6.62620 \times 10^{-34} J \cdot s$ 为普朗克常数。

辐射流量 (Radiant flux)

辐射通量 (radiant flux) 记为，表示光源每秒钟发射的功率 (dQ/dt)，其单位为瓦特 (watt) W，例如一个灯泡可能发射 100 瓦特的辐射通量。在辐射测量中，都是基于这个辐射通量来测试能量，而不是使用总的能量 Q，所以以下这些度量都是在单位时间（每秒）下发生的。

$$\phi = \frac{dQ}{dt} [W = Watt][lm = lumen] \quad (7.2)$$



From London and Upton

图 7.1: flux

7.2. 立体角 (soild angle)

怎么去定义一个角度?

用弧度定义一个角

$$\theta = \frac{l}{r} \quad (7.3)$$

立体角是三维空间中的延伸

$$\Omega = \frac{A}{r^2} \quad (7.4)$$

微分立体角

整个球面上的立体角积分

$$\Omega = \int_{S^2} d\omega = 4\pi \quad (7.5)$$

$$dA = (rd\theta)(rsin\theta d\phi) = r^2 sin\theta d\theta d\phi \quad (7.6)$$

$$d\omega = \frac{dA}{r^2} = sin\theta d\theta d\phi \quad (7.7)$$

Isotropic Point Source

$$\phi = \int_{S^2} Id\omega = 4\pi I \quad (7.8)$$

7.3. 辐照强度、辐射照度和辐射亮度

“辐照强度 (Radiant Intensity)”形容是光子从源发生出来的能量，辐照度 (Irradiance) 形容光子打到一个表面上的能量，”辐射度 (Radiance)”形容光沿着特定方向传播的强度。

辐射亮度

在辐射度量学中，基本上考虑的是从面 A 的一部分射出的光能量。这个面可能是虚构的，也可能就是光源的真实的辐射面，或固体的一个受照面。如果固体是不透明的，则考虑的是反射光；如果固体是透明或半透明的（这时有一部分光被吸收或散射），通常测量的是透射光。

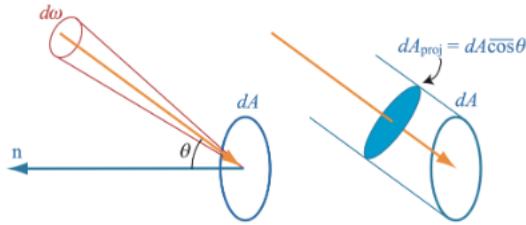


图 7.2: 辐射亮度表示的是某个点在某个方向上的亮度，在计算机图形学中它是一束光的亮度，是渲染方程最终要计算的量

设 $P(\xi, \eta)$ 是 A 面上的一个点，以面上任意一组方面的曲线为参考坐标系，现在在 P 点取一面元 dA ，并围绕极角 (α, β) 方向去一立体角 $d\omega$ ，另外设 $d\omega$ 方向与 dA 法线夹角为 θ ，则单位时间内由面元 dA 发射到 $d\omega$ 的能量值 $d\phi$ 可以表示为

$$d\phi = L \cos \theta \, dA \, d\theta \quad (7.9)$$

其中 $L = L(\xi, \eta; \alpha, \beta)$ 是一个因子。

辐射亮度测试的是单束光的度量，它也正是感应器（例如人的眼睛，或者场景中的虚拟摄像机）测量的度量，所以它在光照计算中特别重要。计算渲染方程的目标就是计算出表面上的点到摄像机所在方向上的辐射亮度。另外值得注意的是， L 的值不随距离发射点距离的变化而变化。

通常用两种不同的方式把 $d\phi$ 分解成两个量的乘积，以表示它对 $d\omega$ 和 dA 的显式关系

$$d\phi = dId\omega = dEdA \quad (7.10)$$

在接下来的两小节将分别讲述这两个新的度量 I 和 E 。

辐照强度 I

辐照强度是指光源在特定方向上发射的光功率，单位立体角上的辐射能量。它描述了光源在某个方向上的辐射能力。可以理解成从光源发出来通过某个面元的“光线的数量”的多少。

$$I(\omega) \equiv \frac{d\phi}{d\omega} \quad (7.11)$$

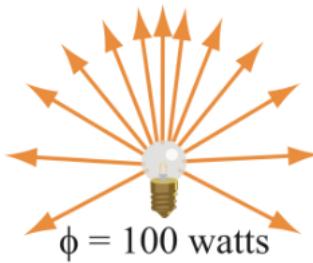


图 7.3: 一个灯泡在不同的方向上可能具有不同的辐射强度, 辐射强度也是计算机图形学中表示各种光源光照的度量

比较辐射光照可以发现

$$dI = \frac{d\phi}{d\omega} = L \cos\theta dA \quad (7.12)$$

对一个面 A 积分得到

$$I(\alpha, \beta) = \int L \cos\theta dA \quad (7.13)$$

其中 I 称为面积 A 在方向 (α, β) 辐照强度, 其单位为 W/sr , I 是与距离无关的量, 但是他是与发射面积有关的, 由于光源通常有一定的形状和面积, 所以图形学通常使用 I 来表示一个光源的辐射强度分布。

辐射照度

定义: 每一个面积上定义的能量

$$dE = \frac{d\phi}{dA} = L \cos\theta d\omega \quad (7.14)$$

对某个立体角积分有

$$E(\xi, \eta) = \int L \cos\theta d\omega \quad (7.15)$$

称为点 (ξ, η) 的辐射照度 (irradiance), 其单位为 W/m^2 , 它是点 (ξ, η) 沿各个方向对辐射亮度 L 的积分, 值得注意的是, 前面提到过, 在辐射度量学中, 基本上考虑的是从面的一部分射出或者接收能量, 所以辐射照度虽然表述的是面上一个点的度量, 但它实际上是通过该点所在的单位面积来测量的, 因为辐射亮度 L 也是通过单位面积来测量的。

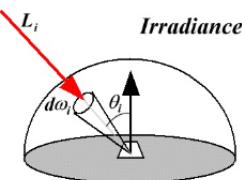


图 1.14 一个面上某点接收来自各个方向的辐射亮度形成辐射照度

图 7.4: 一个面上某点接收来自各个方向的辐射亮度形成辐射照度

回顾在布林冯模型中提到的 Lamber's Cosine Law

$$E = \frac{\phi}{A} \cos\theta \quad (7.16)$$

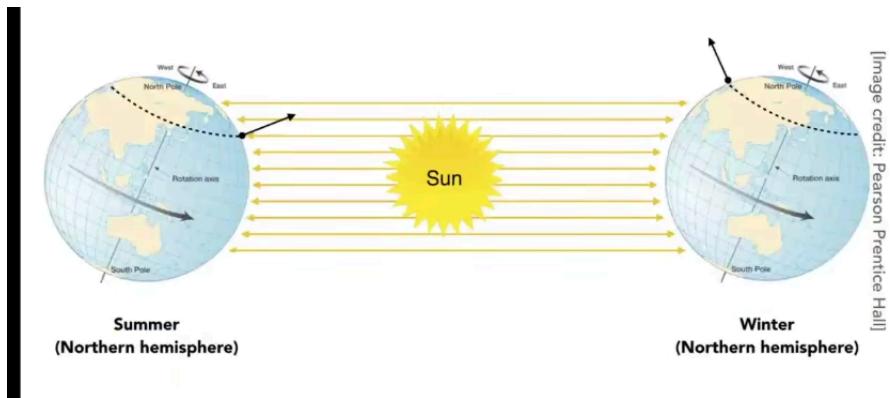


图 7.5: 为什么地球上四季分明?

辐射度量学基本方程

设 dA 是 P 处的面元, $QP = r$, 又设 θ 是 QP 与 dA 的法线夹角, 则光源在单位时间内射过 dA 的能量是 $Id\omega$, 其中 I 是光源沿 QP 方向的辐射强度, $d\omega$ 是 Q 所张立体角, 如下图

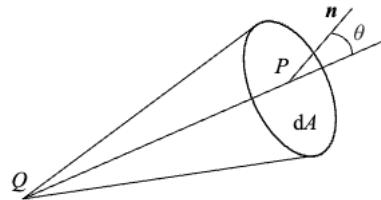


图 7.6: 点光源所产生的辐射照度

由基础几何学

$$\cos\theta \, dA == r^2 \, d\omega \quad (7.17)$$

根据 $d\phi = dId\omega = dEdA$, 所以有

$$E = \frac{I \cos\theta}{r^2} \quad (7.18)$$

上式即辐射度量学的基本方程, 它表达了所谓的照度余弦定理以及平方反比定律。

7.4. Bidirectional Reflectance Distribution Function(BRDF)

双向反射分布函数说明不同反射方向上分布的能量。试图理解反射到底是什么?

Reflecton at a Point

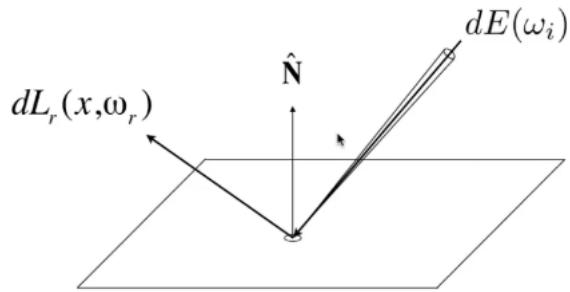


图 7.7: Reflecton at a Point

反射有两部分：

BRDF 会如何被分配到不同的立体角上？

definition 7.4.1: (BRDF)

$$f_r(\omega_i \rightarrow \omega_r) = \frac{dL_r(\omega_r)}{dE_i(\omega_i)} \quad (7.19)$$

The Reflection Equation

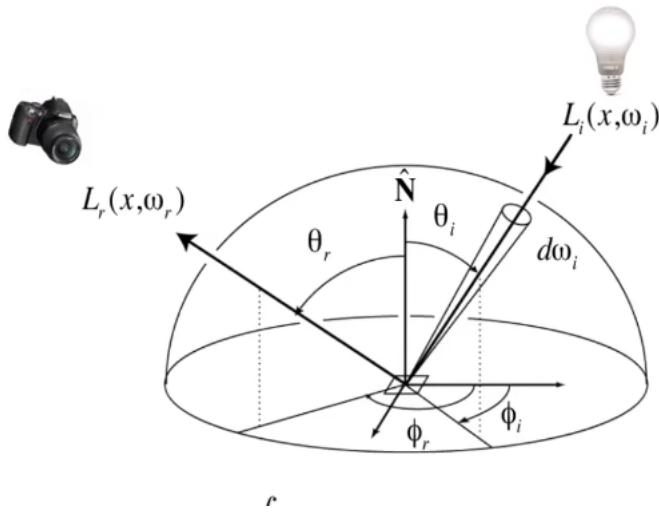


图 7.8: Reflecton at a Point

$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i d\omega_i \quad (7.20)$$

Challenge:Recursive Equation

任何出射的 radiance 都有可能成为任何点入射点 radiance。

The Rendering Equations

$$L_r(p, \omega_r) = \int_{H^2} f_r(p, \omega_i \rightarrow \omega_r) L_i(p, \omega_i) \cos\theta_i d\omega_i \quad (7.21)$$

增加一个发射项使得其变得广义

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega_+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i) d\omega_i \quad (7.22)$$

渲染方程可以让光线弹射多次。

渲染方程积分形式

Linear Operator equation

渲染方程算子形式。

$$l(u) = e(u) + \int l(v) K(u, v) dv \quad (7.23)$$

$$L = E + KL \quad (7.24)$$

General class numerical Monte Carlo methods

Approximate set of all paths of light in scene

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

Binomial Theorem

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

图 7.9: Ray Tracing and Extensions

可以把能量分解成经过弹射次数的分解

$$L = E + KE + K^2E + \dots \quad (7.25)$$

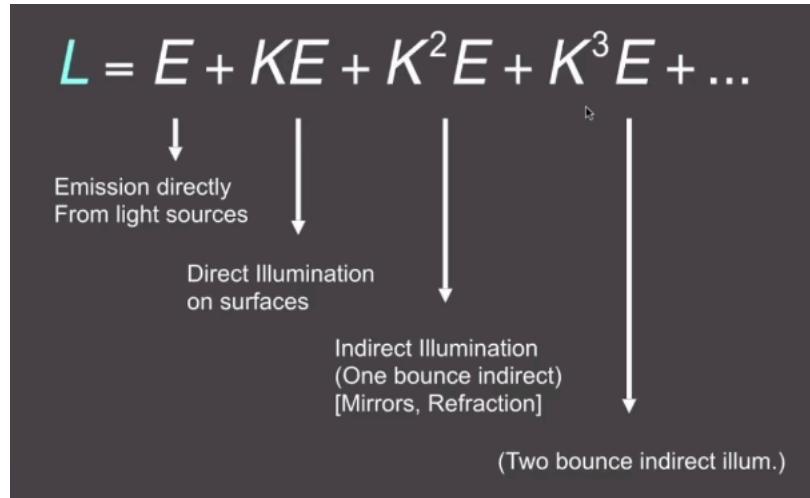


图 7.10: 弹射次数的分解

7.5. Monte Carlo Integration

为什么需要蒙特卡洛积分? 他是为了解决一个定积分

$$\int_a^b f(x)dx \quad (7.26)$$

相对于黎曼积分, 蒙特卡洛积分是基于一种随机的采样的方法。

定积分服从随机分布

$$X_i \sim p(x) \quad (7.27)$$

则蒙特卡洛积分

$$\int f(x)dx = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)} \quad (7.28)$$

Example:Uniform Monte Carlo Estimator

$$X_i \sim p(x) \equiv \frac{1}{b-a} \quad (7.29)$$

$$F_N = \frac{b-a}{N} \sum_{i=1}^N f(X_i) \quad (7.30)$$

7.6. Path Tracing

Motivation:Whitted-Style Ray Tracing

Whitted Style 的问题在于

1. 无法解决 Glossy reflection; 仍然认为沿着镜面反射是不对的;
2. 漫反射物体

怎么解决 Whitted-Style Ray Tracing 的问题

渲染方程是基于物理的，所以绝对正确，采用蒙特卡洛方法解决渲染方程

$$L_o(p, \omega_o) = \int_{\Omega_+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i) d\omega_i \quad (7.31)$$

结论得到

$$L_o(p, \omega_o) \approx \frac{1}{N} \sum_{i=1}^N \frac{L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i)}{p(\omega_i)} \quad (7.32)$$

Introducing Global illumination

```

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
            /
        Return Lo
    
```

图 7.11: 全局光照算法

以这种方式来做的问题, Problem1: 开销巨大

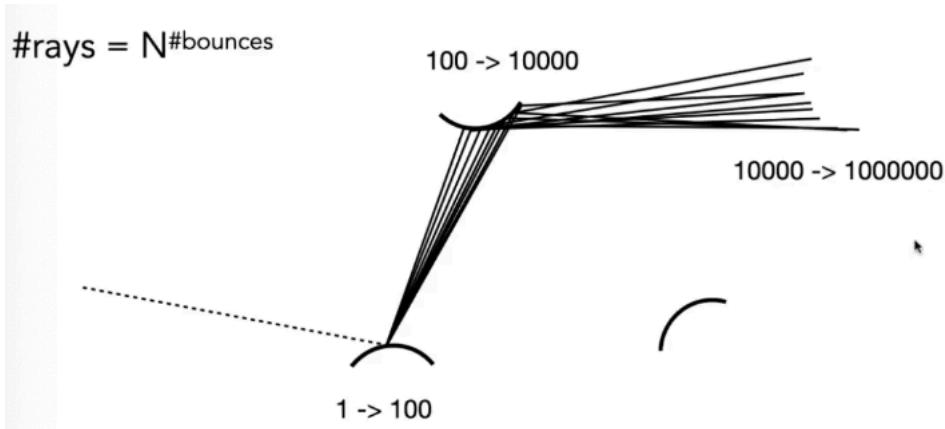


图 7.12: 路径追踪开销巨大

用 $N = 1$ 来做蒙特卡洛就是路径追踪

Ray Generation.

Problem2: 算法怎么停止?

俄罗斯轮盘法：以一定的概率停止往下走

7.7. Sampling the Light

浪费现象

pure math

把渲染方程写成对于光源上的一个对积分的面积。

$$d\omega = \frac{dA \cos\theta'}{\|x' - x\|} \quad (7.33)$$

则渲染方程可以写成

$$L_o(x, \omega_o) = \int_A L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \frac{\cos\theta \cos\theta'}{\|x' - x\|} dA \quad (7.34)$$

7.8. Some Side Notes

Chapter 8

基于辐射度量学的光线追踪

References

- [1] I. Surname, I. Surname, and I. Surname. “The Title of the Article”. In: *The Title of the Journal* 1.2 (2000), pp. 123–456.

Chapter A

Source Code Example

Adding source code to your report/thesis is supported with the package listings. An example can be found below. Files can be added using \lstinputlisting[language=<language>]{<filename>}.

```
1 """
2 ISA Calculator: import the function, specify the height and it will return a
3 list in the following format: [Temperature,Density,Pressure,Speed of Sound].
4 Note that there is no check to see if the maximum altitude is reached.
5 """
6
7 import math
8 g0 = 9.80665
9 R = 287.0
10 layer1 = [0, 288.15, 101325.0]
11 alt = [0,11000,20000,32000,47000,51000,71000,86000]
12 a = [-.0065,0,.0010,.0028,0,-.0028,-.0020]
13
14 def atmosphere(h):
15     for i in range(0,len(alt)-1):
16         if h >= alt[i]:
17             layer0 = layer1[:]
18             layer1[0] = min(h,alt[i+1])
19             if a[i] != 0:
20                 layer1[1] = layer0[1] + a[i]*(layer1[0]-layer0[0])
21                 layer1[2] = layer0[2] * (layer1[1]/layer0[1])**(-g0/(a[i]*R))
22             else:
23                 layer1[2] = layer0[2]*math.exp((-g0/(R*layer1[1]))*(layer1[0]-layer0[0]))
24     return [layer1[1],layer1[2]/(R*layer1[1]),layer1[2],math.sqrt(1.4*R*layer1[1])]
```

Chapter B

Task Division Example

If a task division is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.

表 B.1: Distribution of the workload

Task	Student Name(s)
Summary	
Chapter 1 Introduction	
Chapter 2	
Chapter 3	
Chapter *	
Chapter * Conclusion	
Editors	
CAD and Figures	
Document Design and Layout	