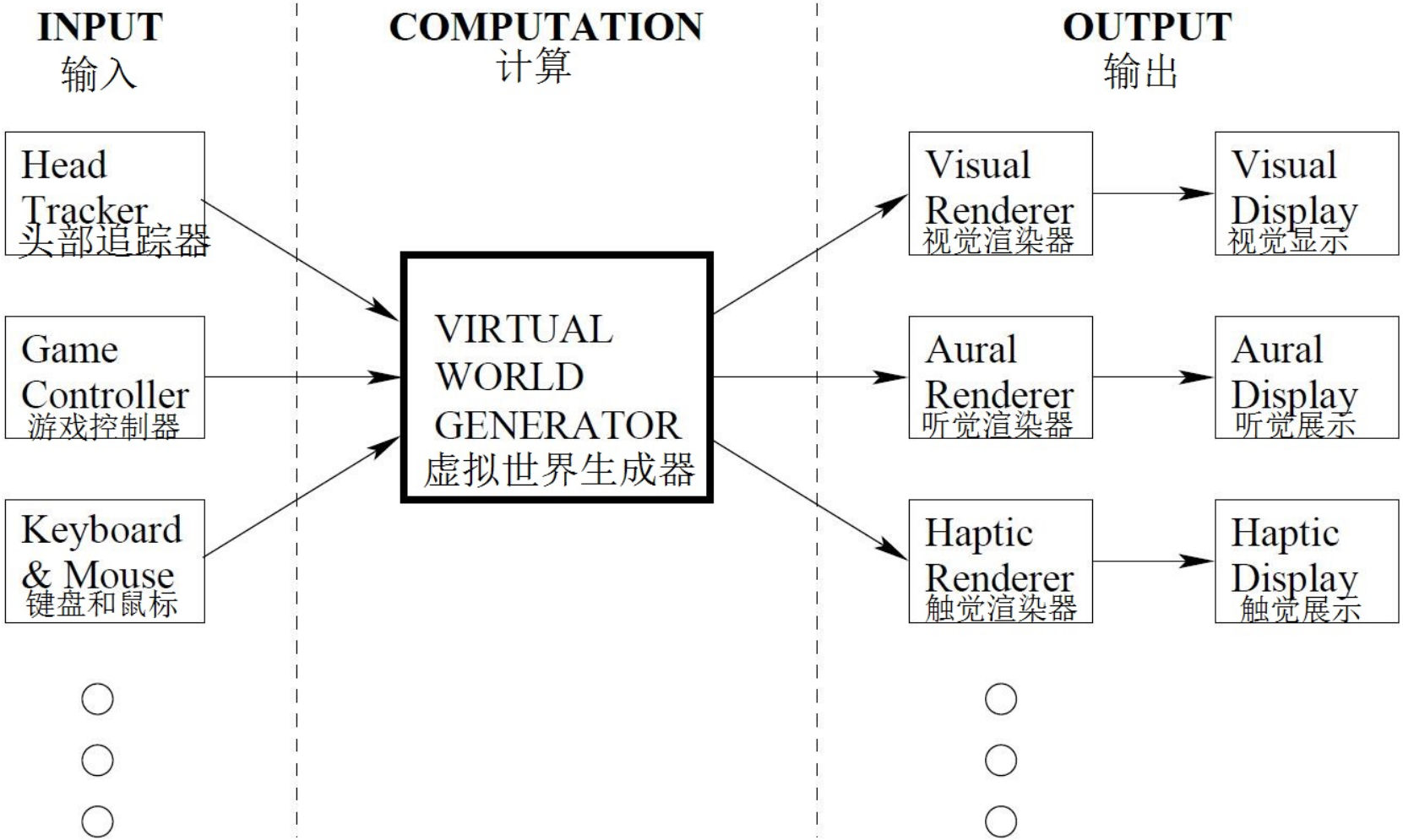
视觉渲染
Visual Rendering

2019.12.2

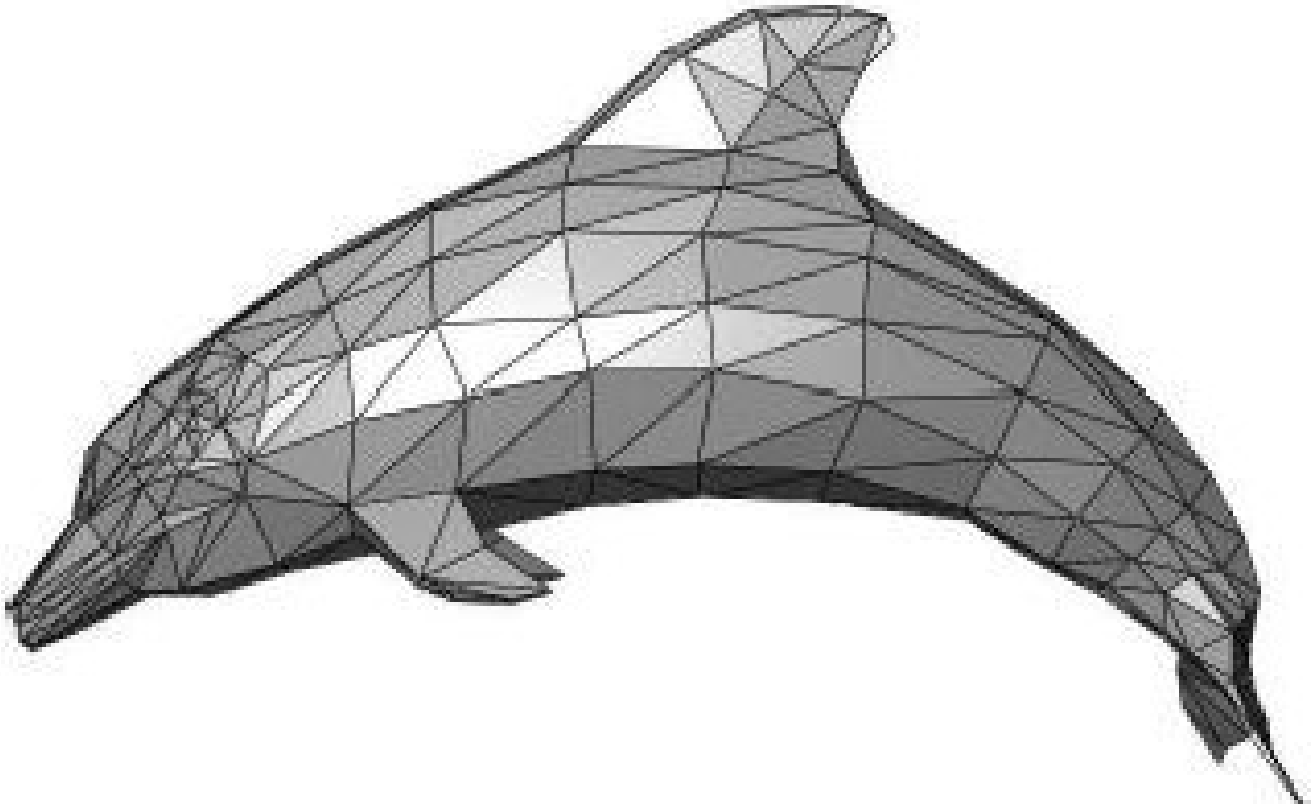
虚拟世界生成器

虚拟世界生成器 (Virtual World Generator, VWG)



图片来自 [3]

三角形网格组成的几何模型



- 经过变换，每个三角形都被正确地放置在虚拟屏幕上
- 接下来，确定哪些屏幕像素被变换后的三角形覆盖，然后更具虚拟世界的物理原理照亮它们

图片来自 [3]

视觉渲染

对于渲染，需要考虑物体和像素的所有组合

基于图像的渲染
Image-ordered rendering

- 一个像素接着一个像素
  Pixel by pixel
- 光线追踪
- 更容易实现
- 通常更慢

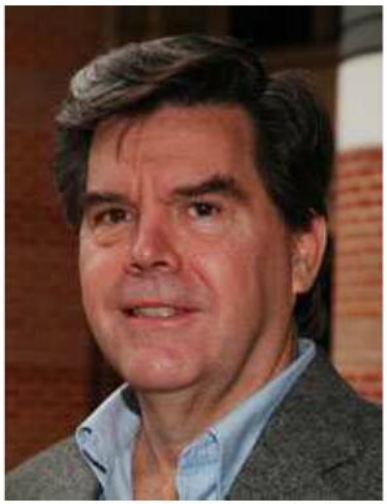
基于对象的渲染
Object-ordered rendering

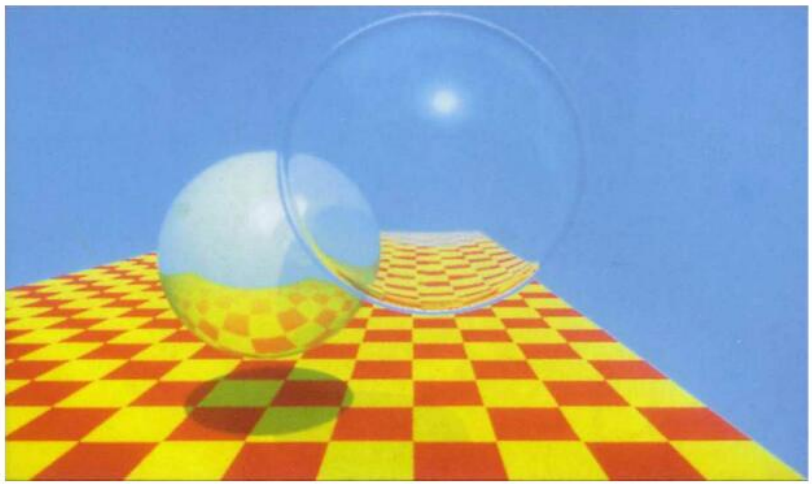
- 一个三角形接着一个三角形
  Triangle by triangle
- 光栅化 Rasterization
- GPUs 使用此方法
- 通常更快

Outline

光线跟踪 Ray tracing

Turner Whitted 于 1980 年首次提出一个包含光反射和折射效果的模型:Whitted 模型，并第一次给出光线跟踪的算法的范例，是计算机图形学历史上的里程碑。





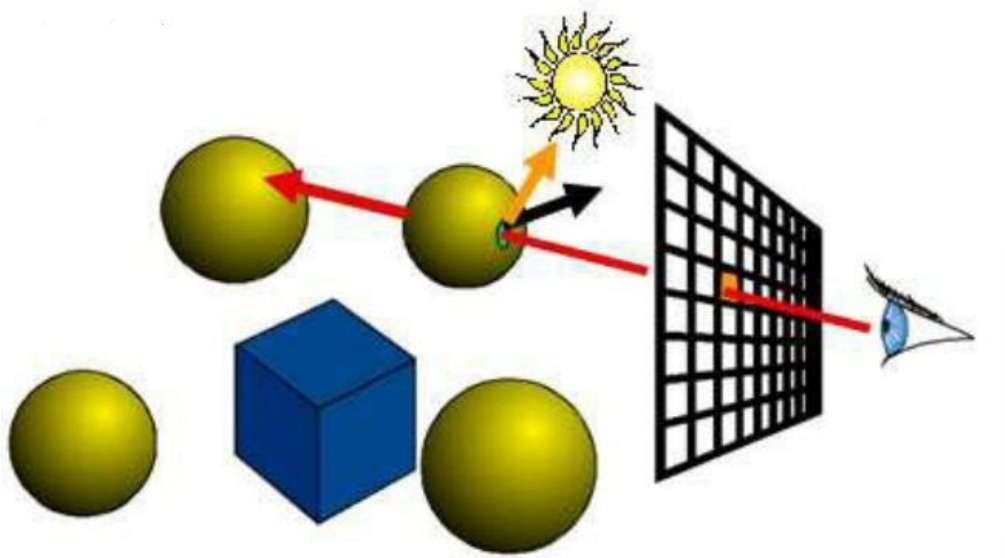
来自 icourses 胡事民

光线跟踪的思路和框架

将显示缓存区看成是由空间中的像素组成的矩阵阵列，人眼透过这些像素看到场景中的物体

对于每个像素 P 计算器色彩值 (逆向跟踪)

- 计算由视点连接像素 P 中心 (Ray) 延长后所碰到的第一个物体的交点。
- 使用局部光照模型 (如 Phong 模型) 计算交点处的颜色值。
- 沿交点处的反射和折射方向对光线进行跟踪。



来自 icourses 胡事民

光线跟踪 Ray tracing

为了计算像素处的 RGB 值，从放置在虚拟世界中的屏幕上的焦点通过像素的中心绘制观察光线。
该过程分为两个阶段：

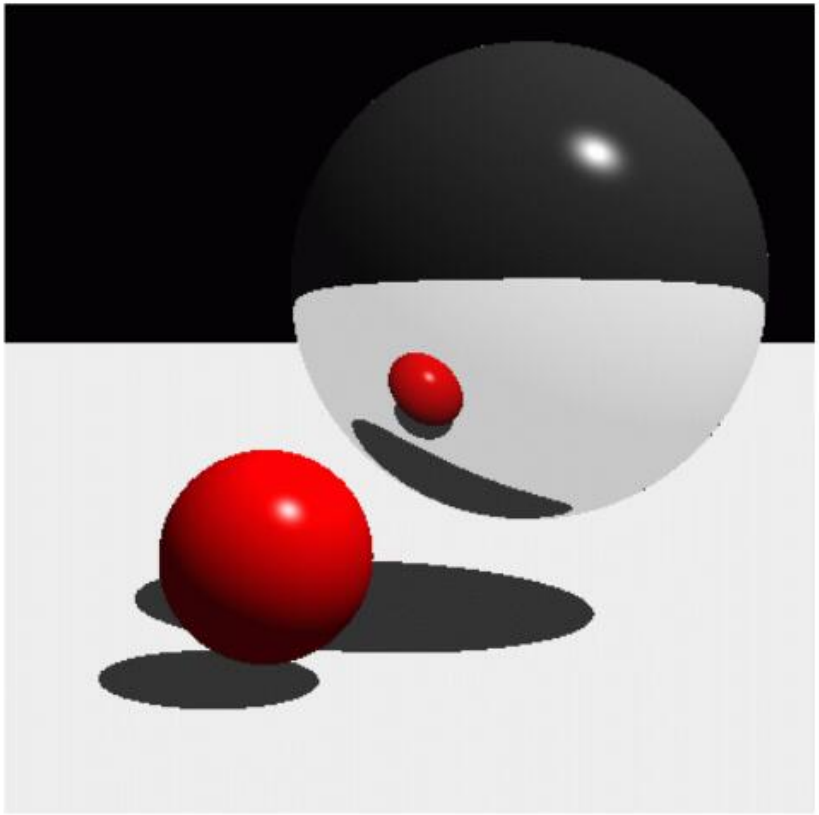
- 光线投射 Ray casting：定义了观察射线并计算虚拟世界中所有三角形之间的最近交点
- 光照和明暗处理 Shading：根据照明条件和交点处的材料属性计算像素 RGB 值

光线跟踪 Ray tracing

光线追踪算法：可实现其他算法很难达到的效果，作为一个有效的真实感图形绘制算法被广泛使用

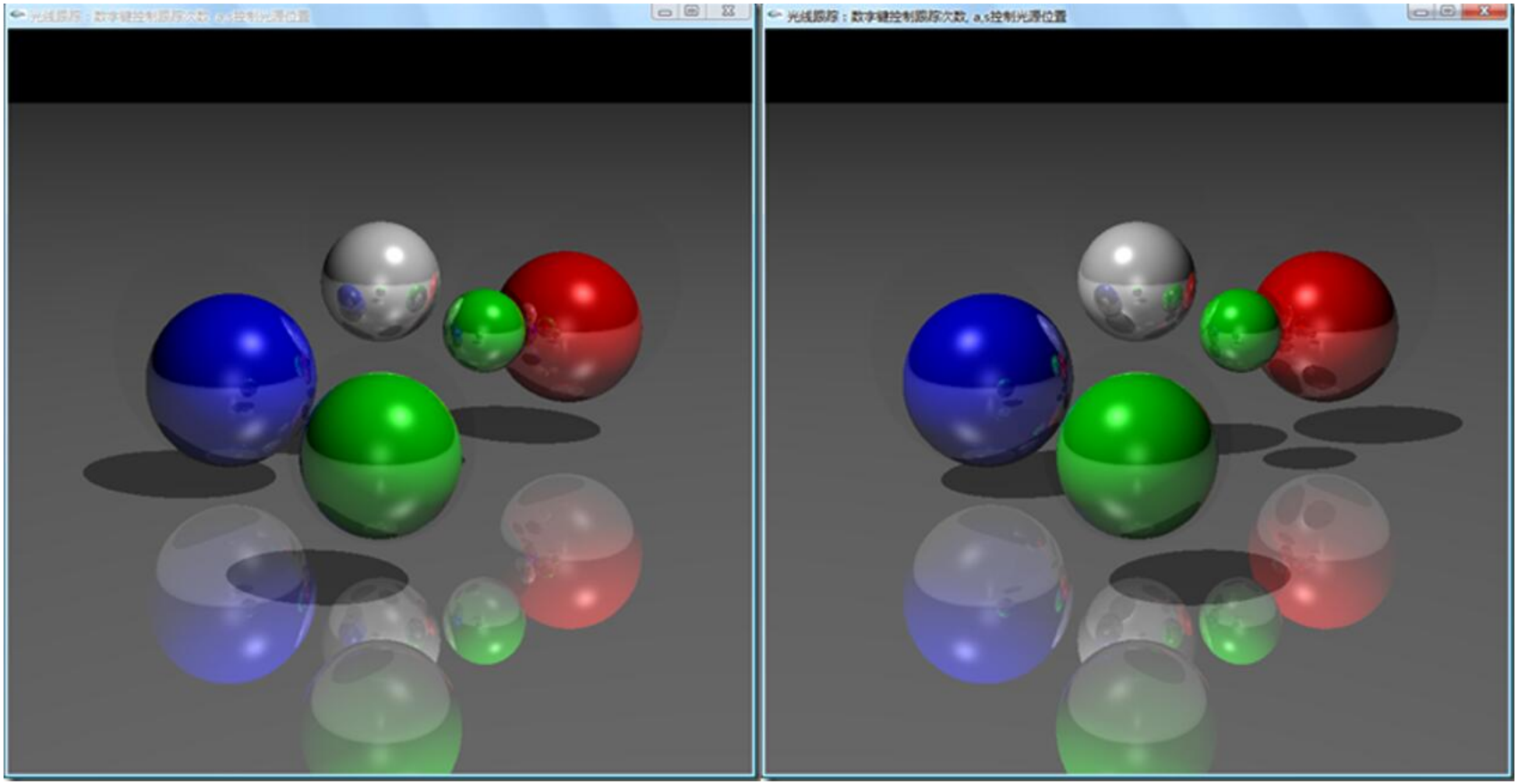
- 支持全局光照的方法。
- 光线追踪能简单直接地支持阴影、反射、折射，实现起来亦非常容易。

光线追踪 Ray tracing



来自 icourses 胡事民

光线追踪 Ray tracing



光线追踪

Outline in Code

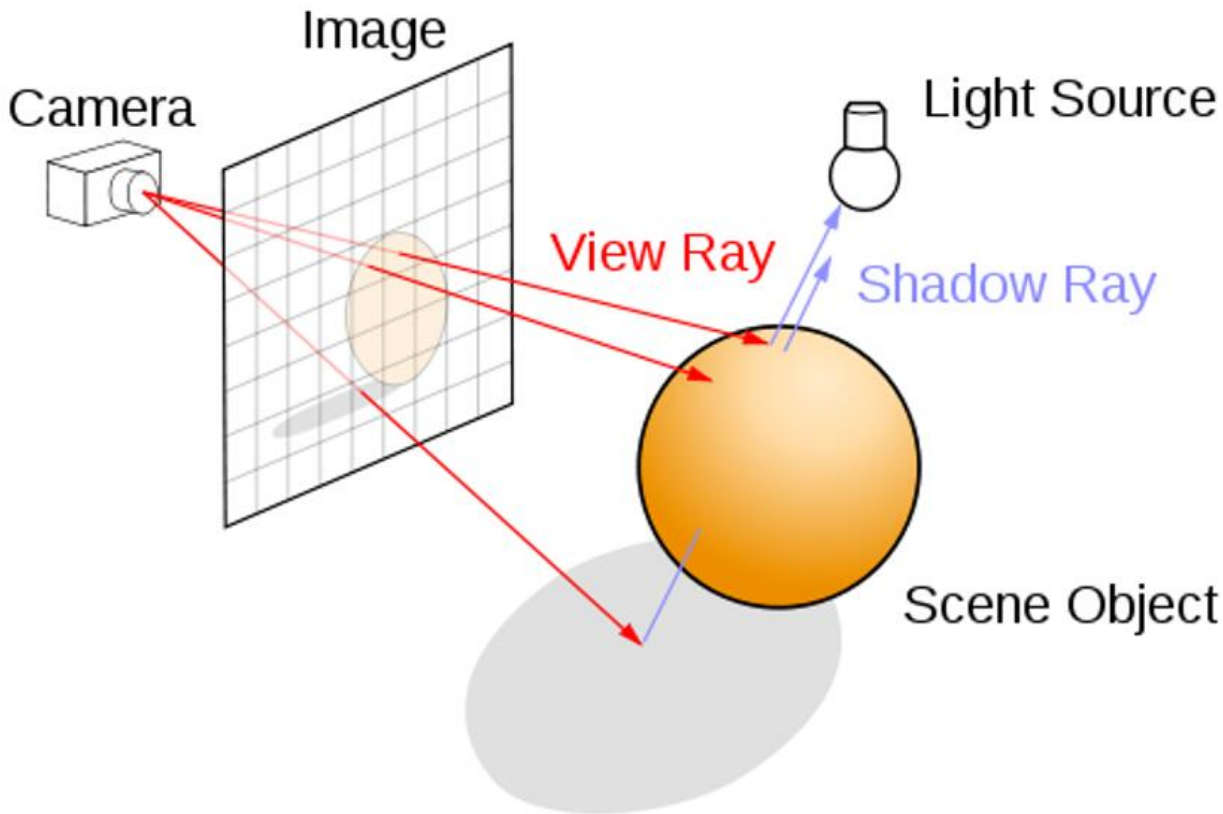
```
Image Raytrace (Camera cam, Scene scene, int width, int height)
{
    Image image = new Image (width, height) ;
    for (int i = 0 ; i < height ; i++)
            for (int j = 0 ; j < width ; j++) {
                    Ray ray = RayThruPixel (cam, i, j) ;
                    Intersection hit = Intersect (ray, scene) ;
                    image[i][j] = FindColor (hit) ;
                    }
    return image ;
}
```

图片来自 edX Ravi Ramamoorthi

光线追踪 Ray tracing

- 光线追踪，简单地说，就是从摄影机的位置，通过影像平面上的像素位置，发射一束光线到场景，求光线和几何图形间最近的交点，再求该交点的著色。
- 如果该交点的材质是反射性的，可以在该交点向反射方向继续追踪。
- 光线追踪除了容易支持一些全局光照效果外，亦不局限于三角形作为几何图形的单位。任何几何图形，能与一束光线计算交点，就能支持。

光线追踪



上图显示了光线追踪的基本方式。要计算一点是否在阴影之内，也只须发射一束光线到光源，检测中间是否存在障碍物。

光线追踪 Ray tracing

- 不仅考虑到光源的光照，而且考虑到场景中各物体之间彼此反射的影响，因此显示效果十分逼真。
- 每条光线的处理过程相同，结果彼此独立，因此可以在并行处理的硬件上快速实现光线跟踪算法。
- 光线跟踪算法的缺点是计算量非常大，因此，显示速度极慢。

光照和明暗处理的重要性

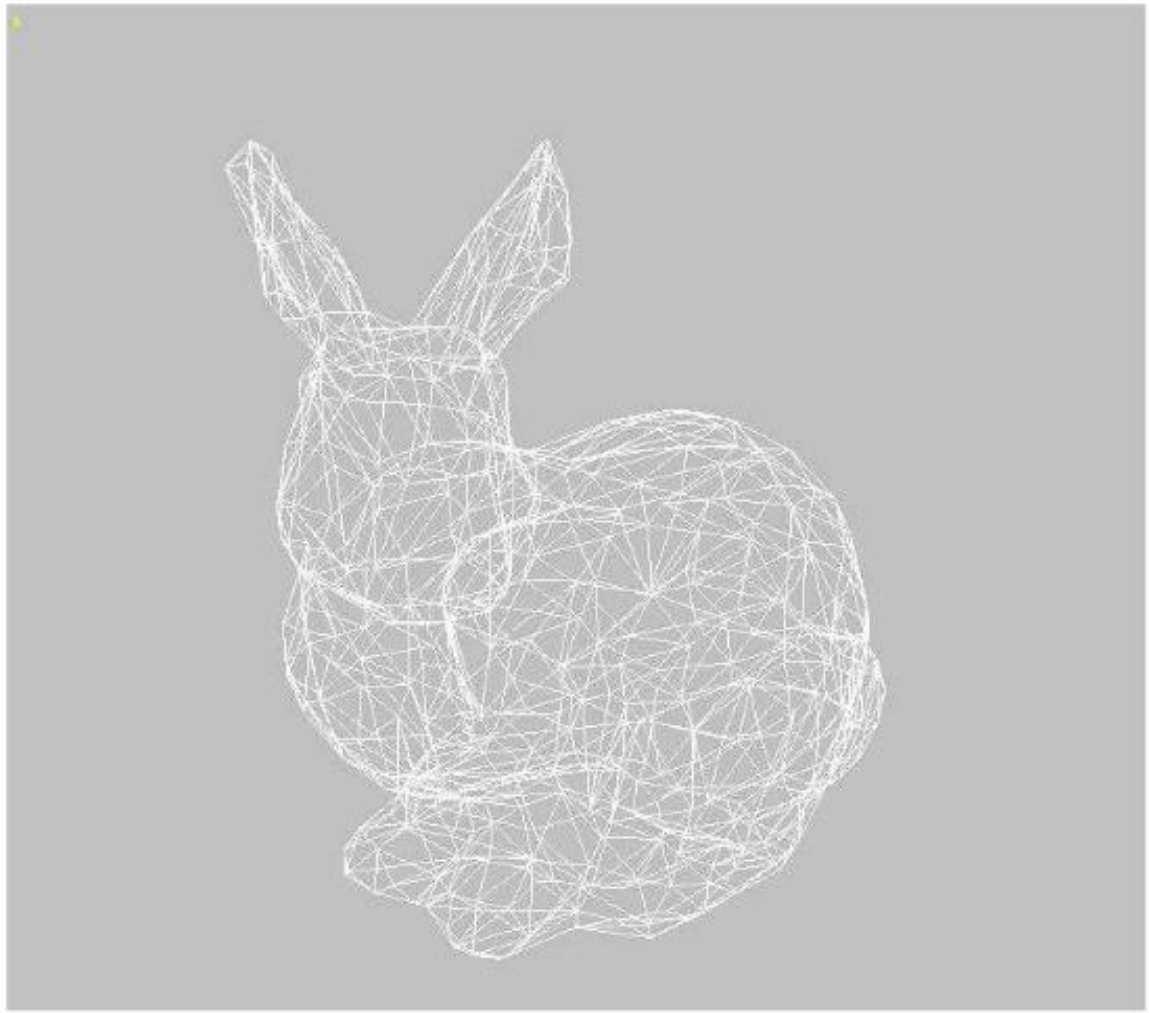
- 对于营造立体感非常重要
- 对于光照下得到正确的阴影很重要
- 怎么进行光照明暗处理？
  - Flat: 整个面值使用从一个端点得到的一种颜色
  - Gouraud or smooth: 使用多个端点的信息，插值得到颜色

三角网格的简单绘制

- 三角网格模型的每个顶点都需要指定一个颜色属性。
- 基于颜色的绘制
  - 模型表面的每个点的颜色通过其所在的三角面皮的顶点颜色插值得到
- 基于光照的绘制
  - 需要指定一个虚拟的光照环境
  - 如何计算光照对颜色的影响是最大的问题

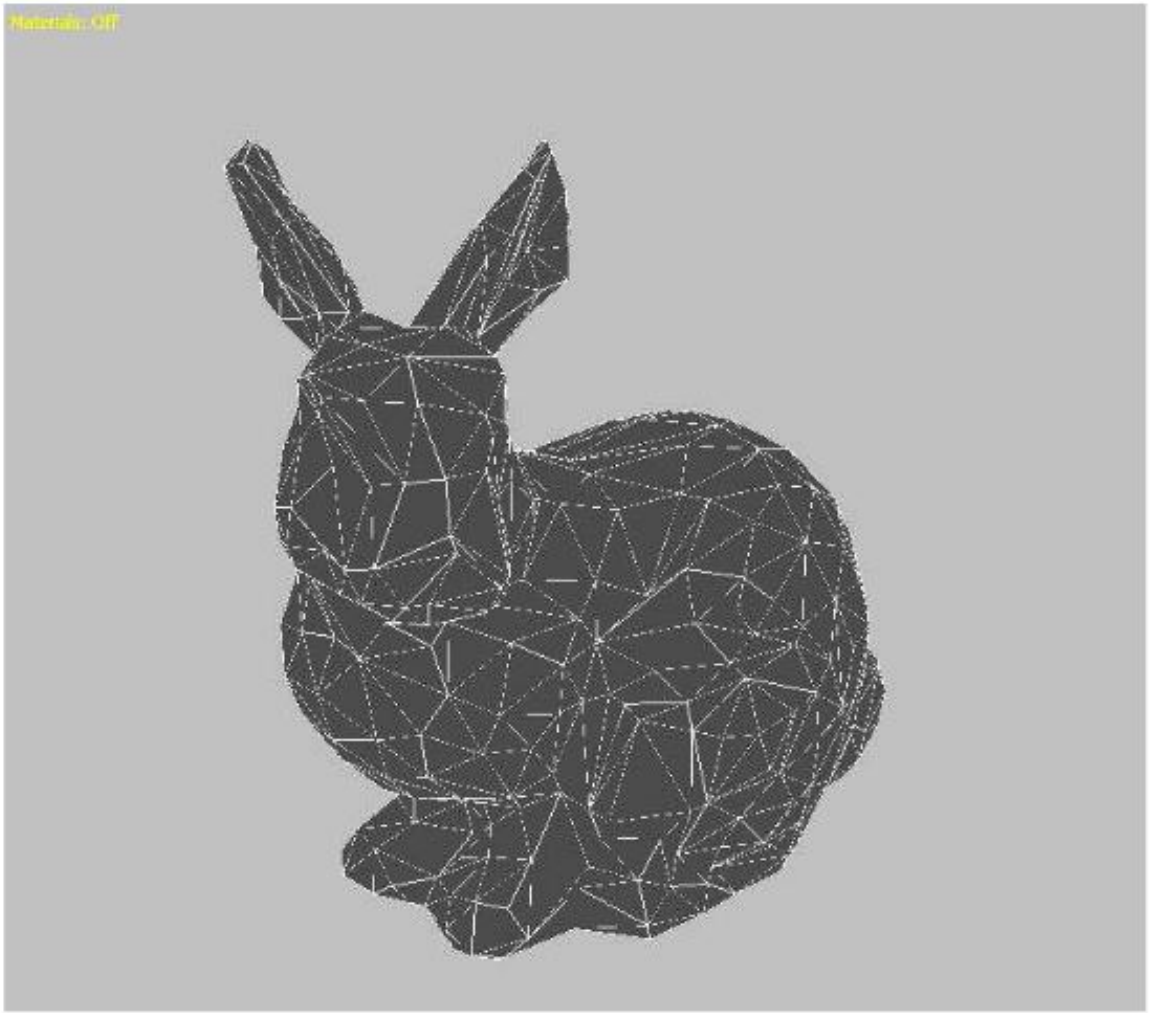
三角网格的简单绘制效果

线框效果



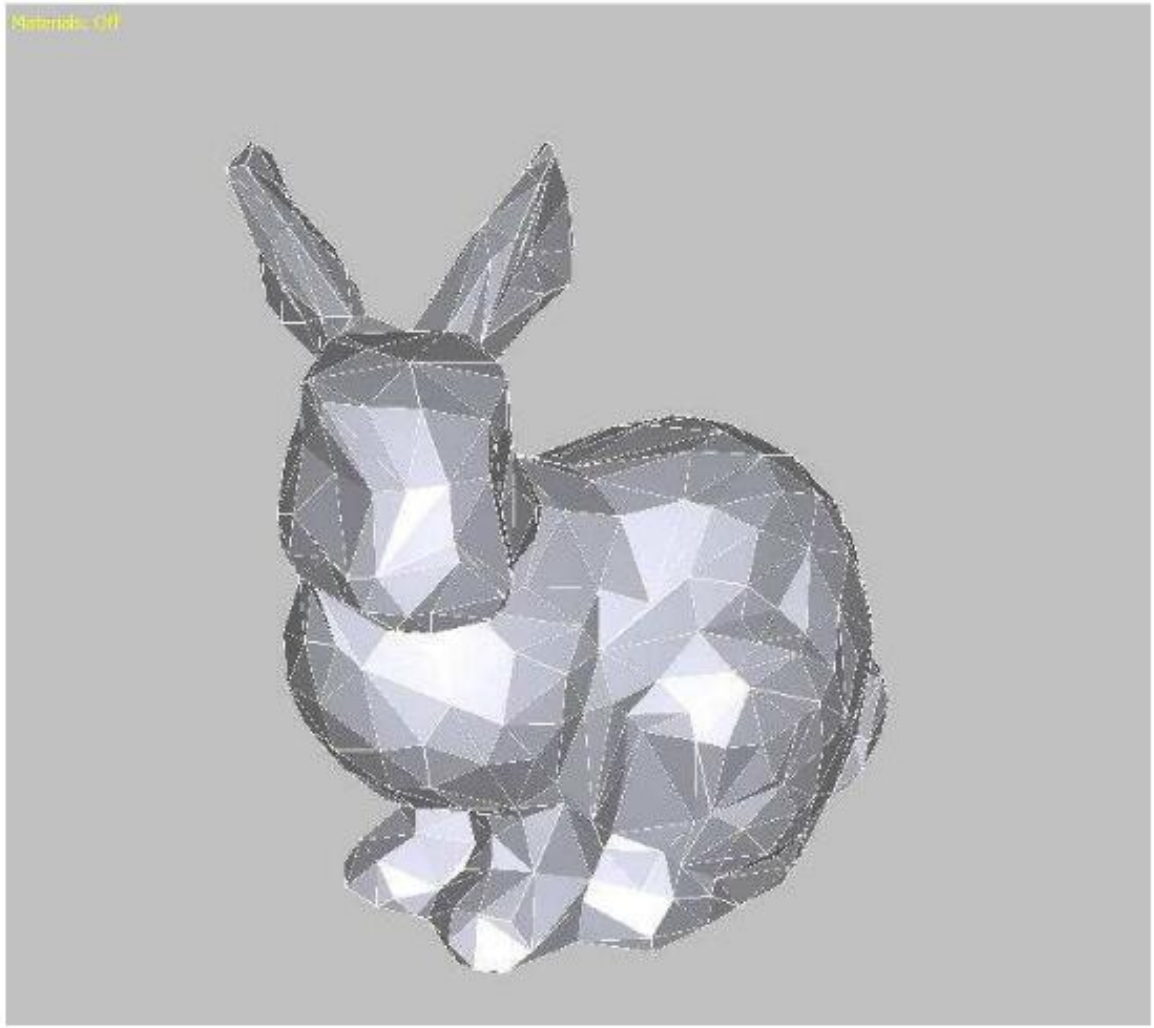
三角网格的简单绘制效果

颜色效果



三角网格的简单绘制效果

光照效果



光照模型

光照模型 (lighting model 或 illumination model) 用于计算光的强度

- 局部光照明 (Local Lighting)
  - 关注物体直接受到光源影响所产生的光照效果
- 全局光照明 (Global Lighting)
  - 关注阴影效果
  - 关注所有不是直接与光源位置相关的光照效果，例如反射和折射效果等

光照模型的历史

- 1967 年，Wylie 等人第一次在显示物体时加入了光照明效果，认为光的强度与物体到光源的距离成反比关系
- 1970 年，Bouknight 提出来第一个光反射模型：
  - Lambert 漫反射光 + 环境光
  - 发表于 Communication of ACM

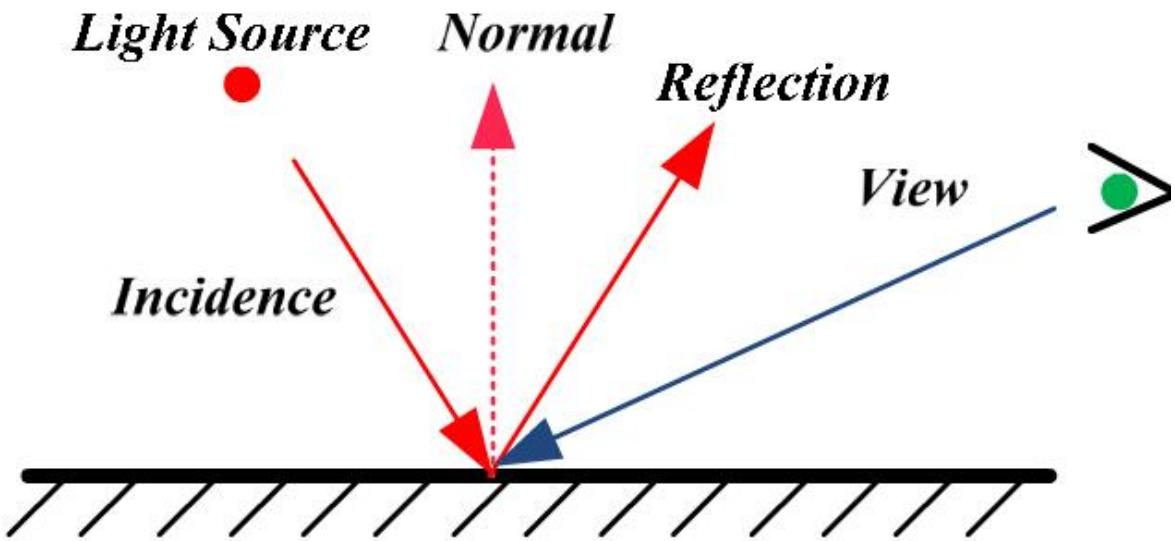
光照模型的历史

- 1971 年，Gouraud 提出来漫反射模型加差值的思想
  - Lambert 漫反射光 +Barycentric 差值
  - 发表于 IEEE transactions on Computers
- 1975 年，Phong 提出了图形学中第一个有影响也是最有影响的光照模型：Phong 模型
  - 漫反射 (diffuse light)+ 环境光 (ambient light)+ 高光 (specular light)
  - 发表于 Communication of ACM

光的传播

光的传播遵循反射定律

- 入射角等于反射角
- 入射光线、反射光线、以及反射面的法向量位于同一平面内。



光线传播的能量方程

光的传播遵循能量守恒定律:

$I_i = I_d + I_s + I_t + I_v$

- I_i: 入射光的能量
- I_d: 漫反射 (diffuse reflection) 光的能量
- I_s: 镜面反射 (specular reflection) 光的能量
- I_t: 折射（refraction）光的能量
- I_v: 被介质和物体所吸收的能量

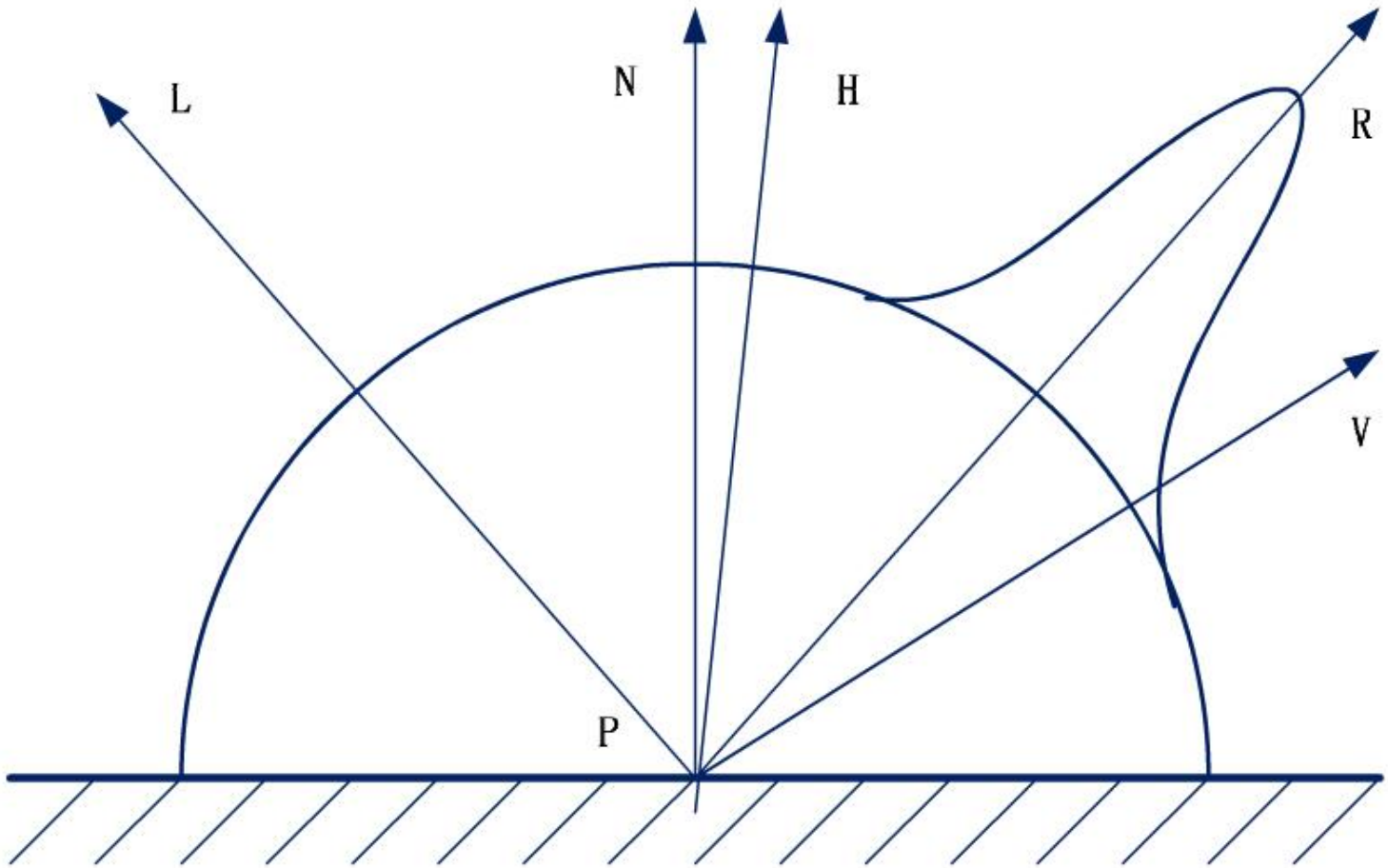
Phong 光照模型

- Phong 模型支持点光源和方向光源
- Phong 模型是局部光照模型，将局部光照明效果分解成三个部分：
  - 漫反射光效果
  - 镜面反射光效果
  - 环境光效果

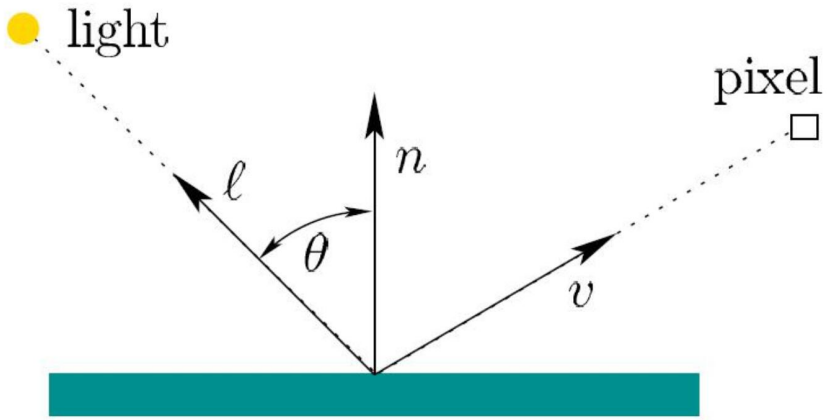
Phong 光照模型

Phong 光照模型

L 是入射光；R 是反射光；N 是物体表面的法向量；V 是视点方向；H 是 L 和 V 夹角的角平分线方向



Phong 光照模型



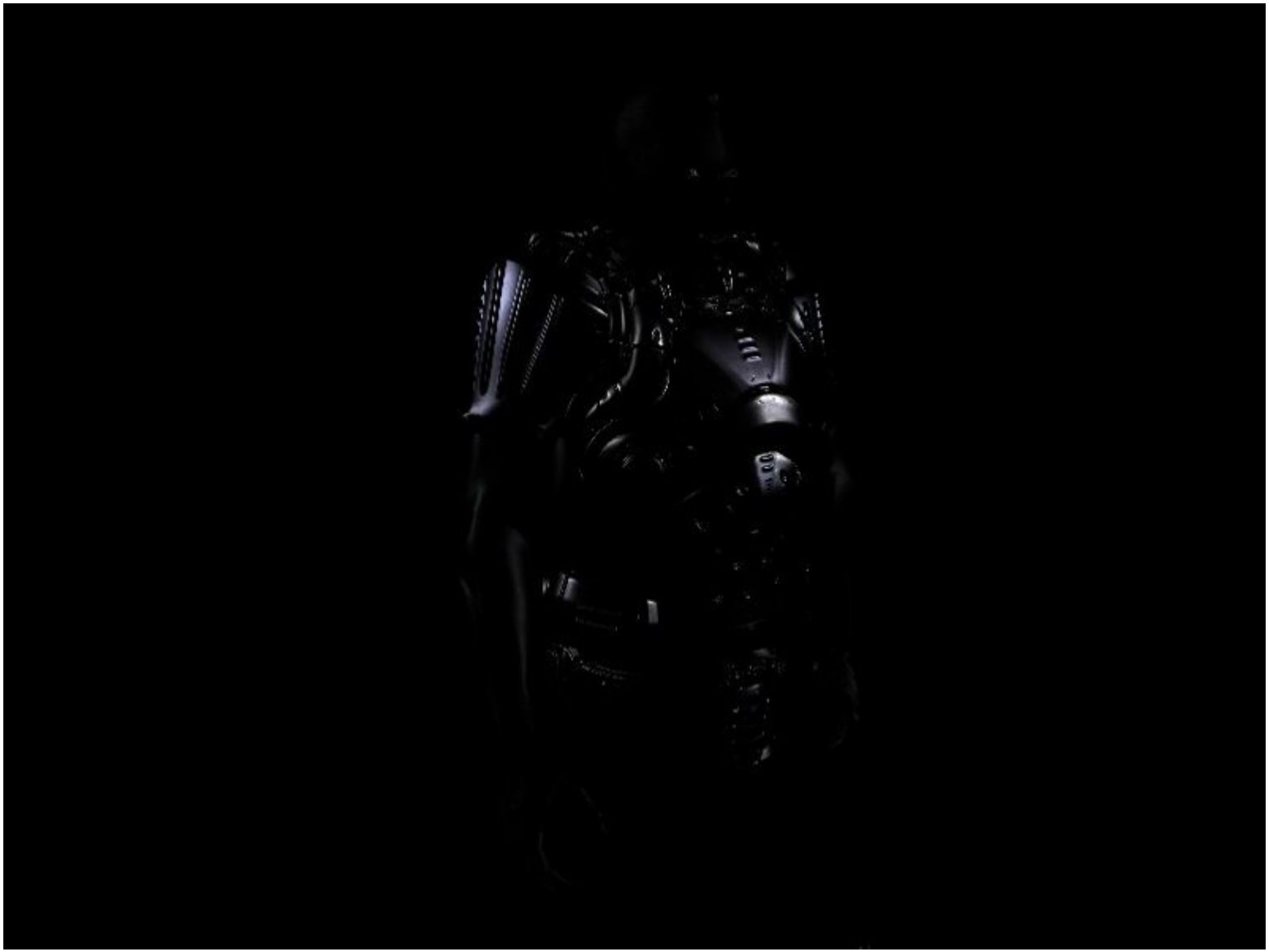
- 漫反射光效果: 对于表面比较粗糙的物体，基本表面的明暗就是漫反射效果，比如裤子的材质。某一个象素的明暗系数只取决于该点与光源的相对位置，而与眼睛的位置无关
  - 漫反射光的传播是各向同性的;
  - 漫反射光的强度为：$I_d = I_i K_d * max(0, L \cdot N)$
  - K_d 是漫反射系数。
  - K_d 具有三个分量 k_{dr}, k_{dg}, k_{db} 分别代表 R,G,B 三个通道的漫反射系数。
  - K_d 与模型自身的色彩紧密相关。

漫反射效果

Phong 光照模型

- 镜面反射光效果: 镜面反射是视点相关的，所以会随着眼睛位置的变化而“流动”
  - 对于光滑的平面，依据反射定律，反射光线往往集中在一个小的立体角内，这些反射光称为镜面反射光；
  - 镜面反射光的强度为：$I_s = I_i K_s * max(0, R \cdot V)^n$
  - K_s 是镜面反射系数，与物体表面光滑程度相关。
  - n 是反射指数；n 越大，则高光区域越集中。

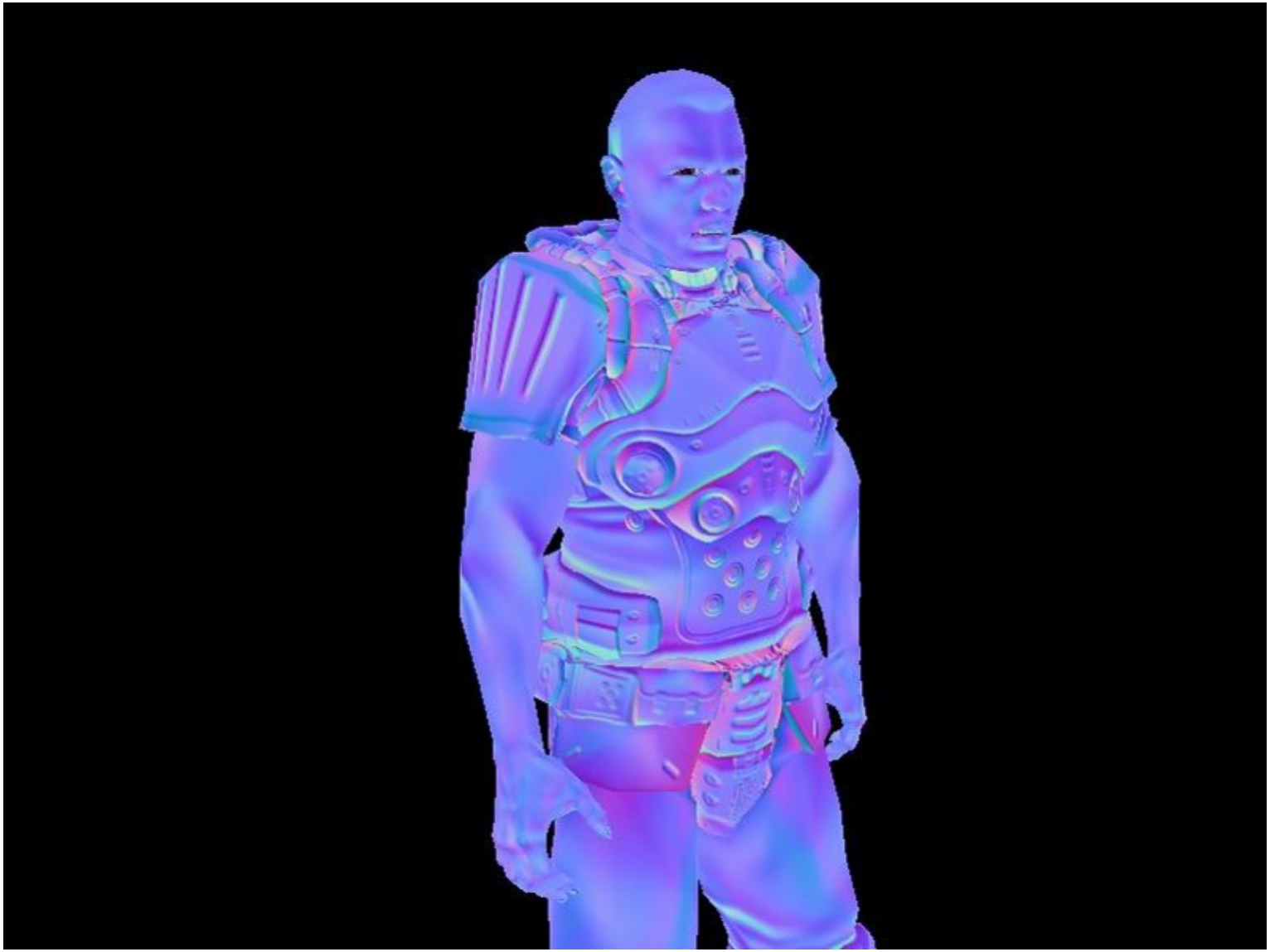
镜面反射效果



Phong 光照模型

- 环境光效果
  - 环境光的强度为：$I_a = I_i K_a$
  - K_a 是物体对环境光的反射系数。

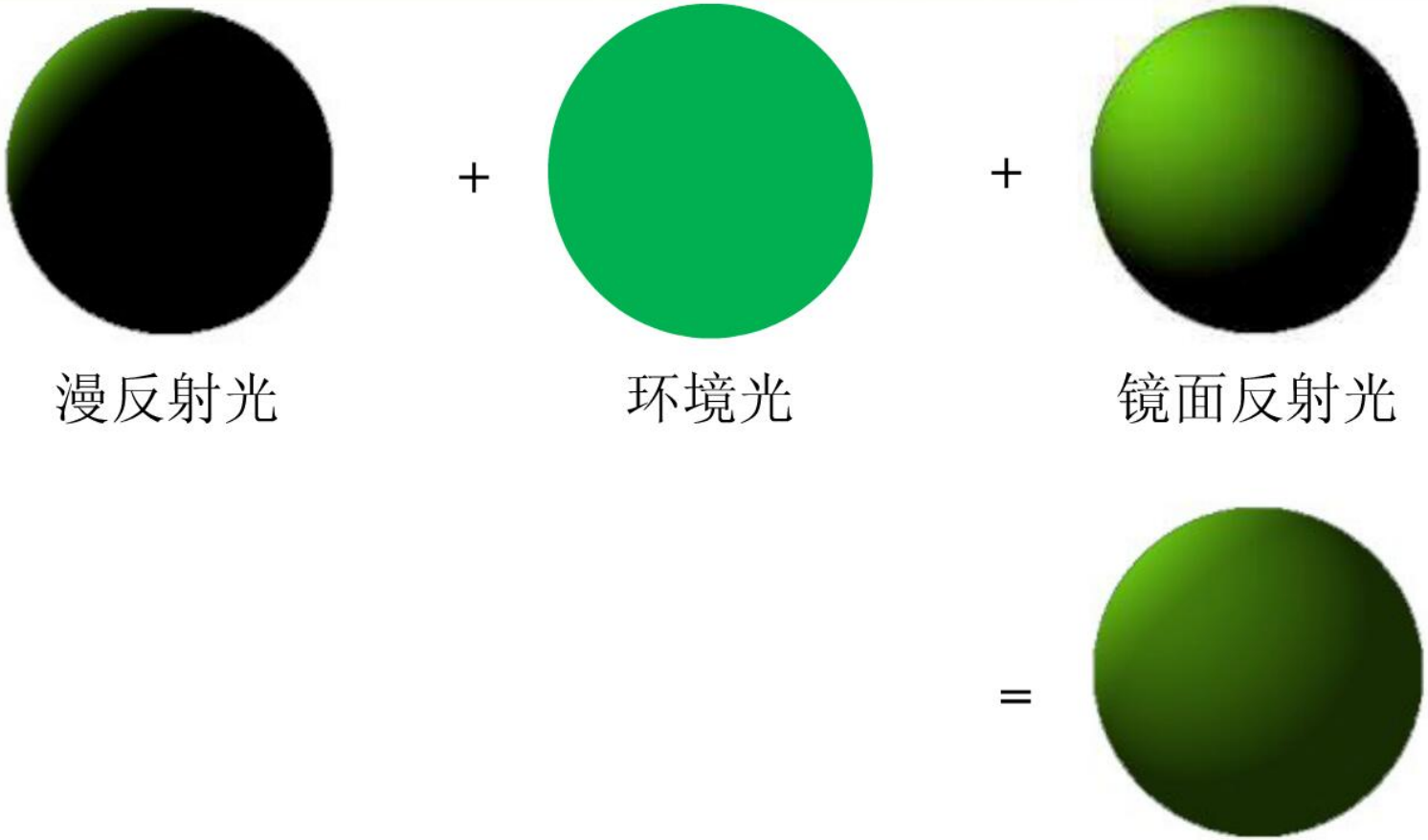
环境光效果



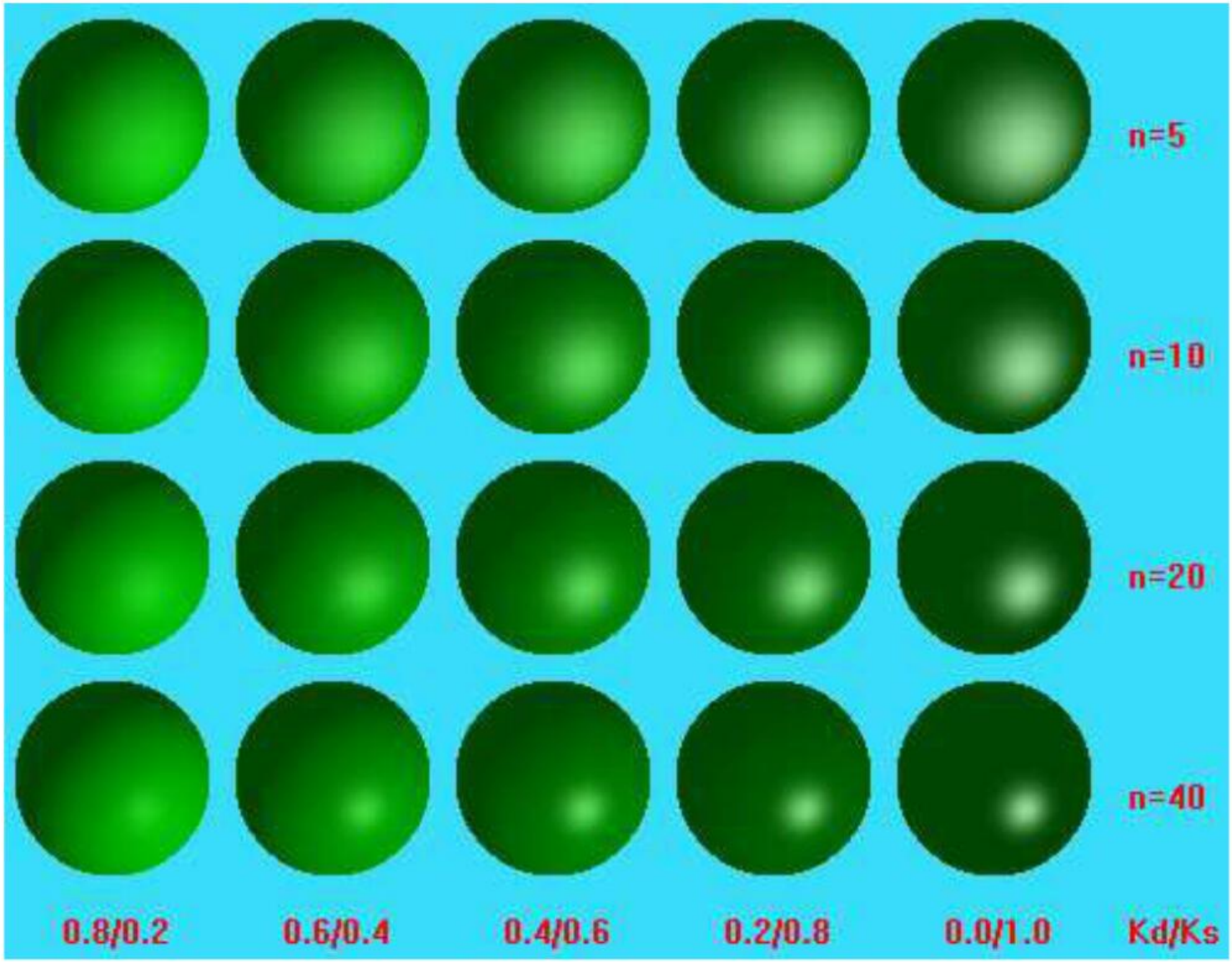
Phong 光照模型

- 视角方向的发光强度为漫反射光分量、镜面反射光分量，以及环境光分量的发光强度之和
  $I = I_i K_a + I_i K_s * max(0, R \cdot V)^n + I_i K_d * max(0, L \cdot N)$

Phong 模型示例



Phong 模型示例



明暗处理（Shading）

图片来自 edX Ravi Ramamoorthi

- 考虑到物体表面的几何细节往往不规则，为了减缓由模型离散化所导致的不光滑的色彩效果，通常的明暗处理除了使用光照模型外，还需要进行插值
- Gouraud 明暗处理，Phong 明暗处理
  - Gouraud 明暗处理是对色彩进行插值；
  - Phong 明暗处理是对法向进行插值。

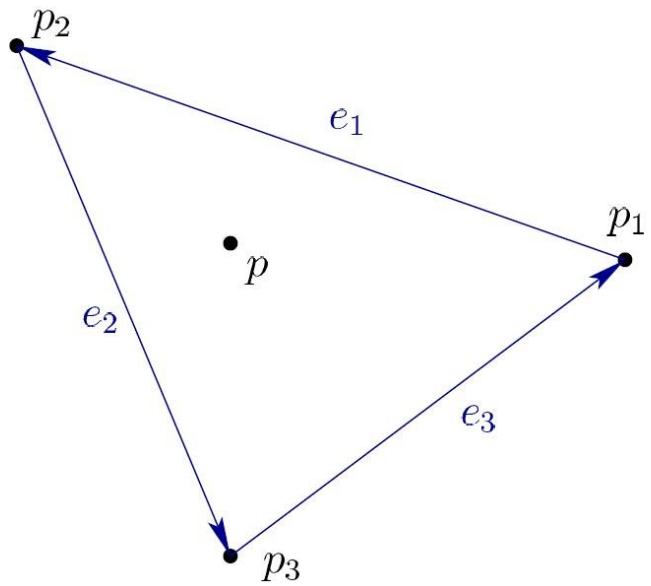
插值是在特点范围内通过一组离散的已知数据点来构建新的数据点的方法。

Gouraud 明暗处理

- 由 Gouraud 于 1971 年提出
  - 又被称为 Gouraud 插值；
- 计算方法：
  - 首先计算所有模型顶点的色彩值；
  - 对模型上的任何一点，按照其所在的三角面片上顶点的色彩值按重心差值 (Barycentric Interpolation) 的结果赋予其色彩值。

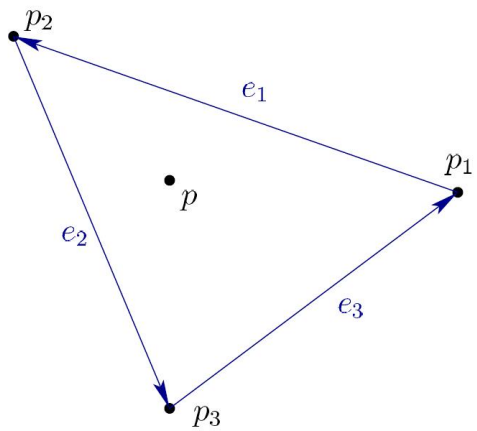
Barycentric 坐标

把颜色 RGB 值等一些性质从端点扩散到面片其余的点的有用工具：Barycentric 坐标



图片来自 [3]

Barycentric 坐标

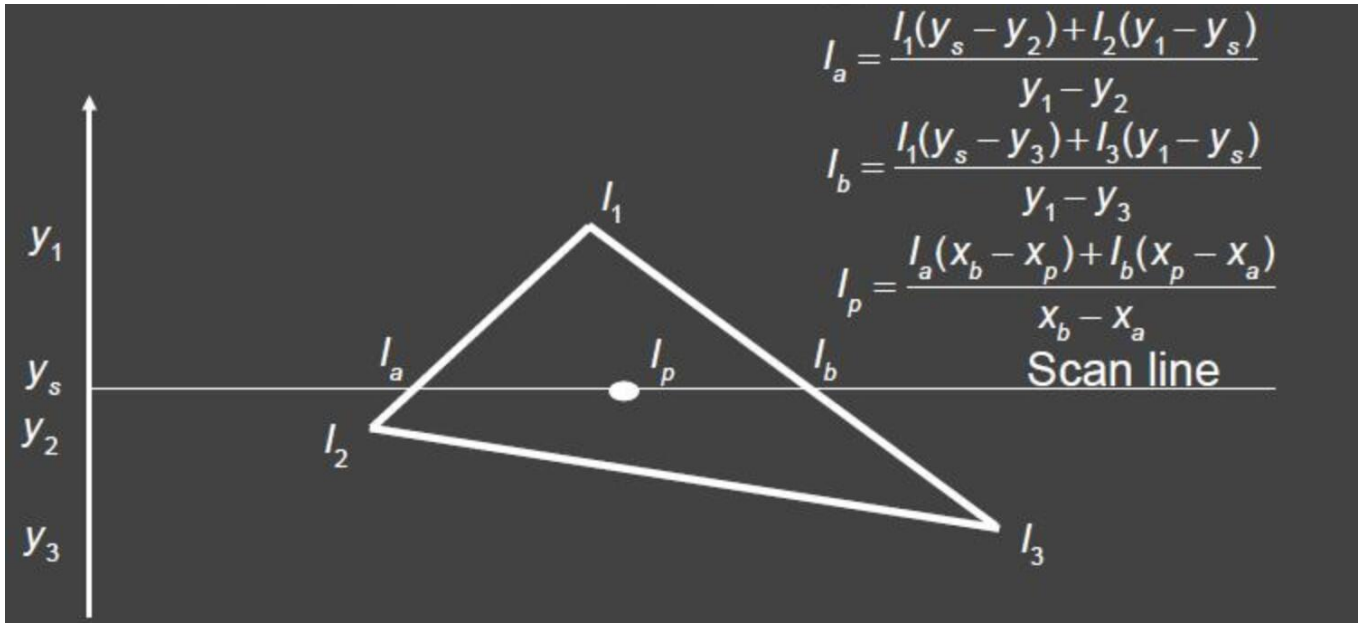


$p_i = (x_i, y_i)$

$p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3,$

for $0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1,\ \alpha_1 + \alpha_2 + \alpha_3 = 1$

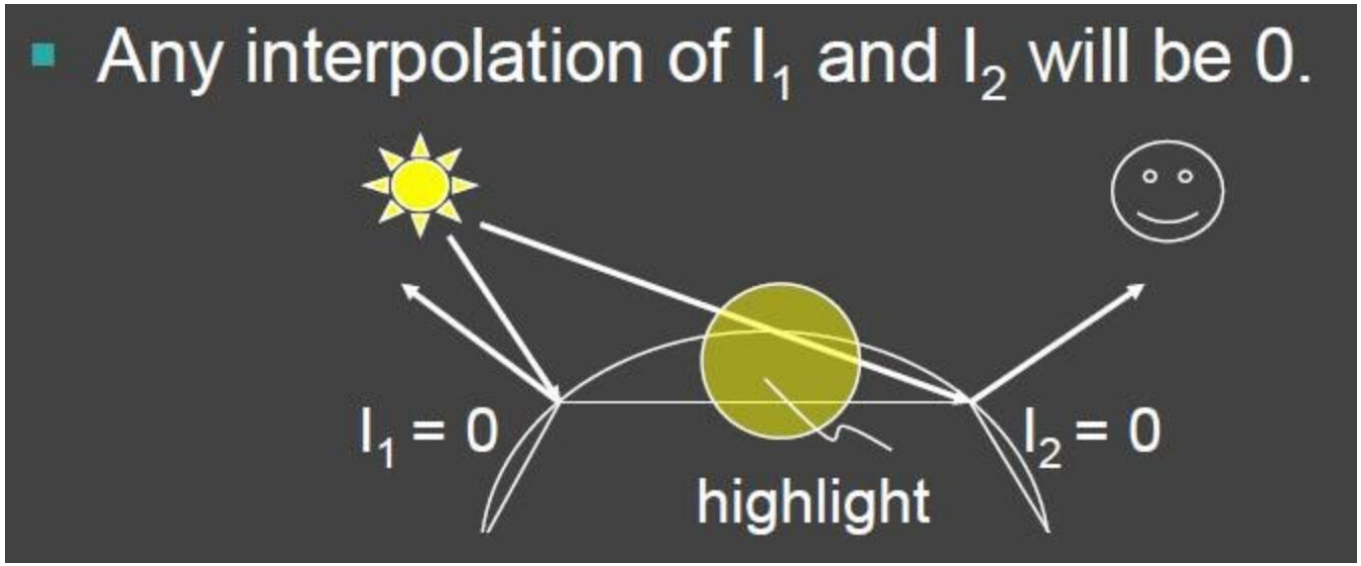
Gouraud 明暗处理



图片来自 edX Ravi Ramamoorthi

Gouraud 明暗处理只在多边形顶点处采用 Phong 局部反射模型计算光强，而在多边形内的其他点采用双向线性插值，这样做的优点是高效，但是无法很好的处理镜面高光问题，依赖于其所在多面形的相对位置.

Gouraud Shading and Errors



图片来自 edX Ravi Ramamoorthi

Phong 明暗处理

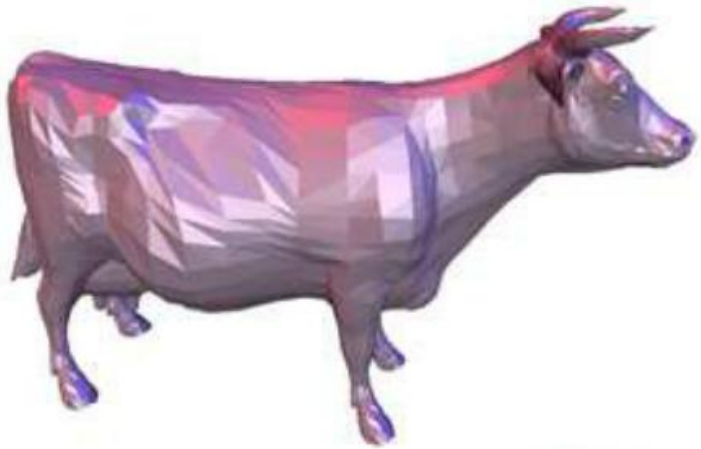
- 由 Bui Tuong Phong 于 1973 年在其博士论文中提出
  - 与 Phong 光照模型是不同的概念。
- 计算方法：
  - 与 Gouraud 明暗处理不同，Phong 明暗处理不是对颜色插值，而是对点的法向量进行插值，得到连续的法向量场，再利用该法向量场逐点使用光照模型进行色彩的计算。

Phong 明暗处理

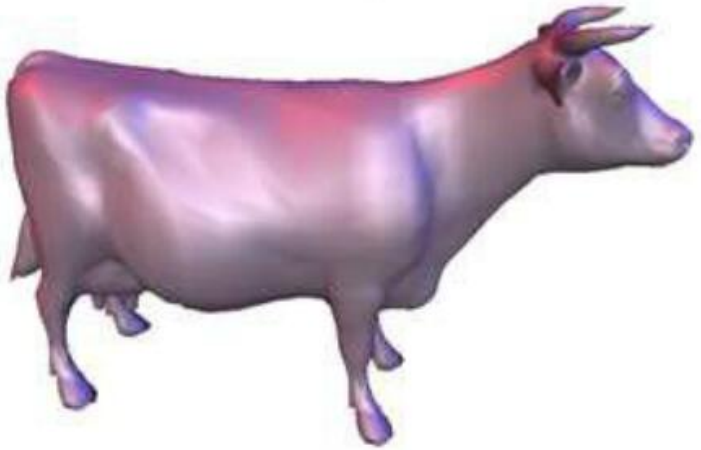
Phong 明暗处理，通过插值计算每个顶点的法向量（3 次插值，在 x，y，z 三个方向分别进行插值计算），然后计算每个点上的光强值，这样效果好，但计算复杂，需要付出比 Gouraud 4-5 倍的时间。

Phong 明暗处理示例

直接用光照明模型进行绘制



Phong 明暗处理

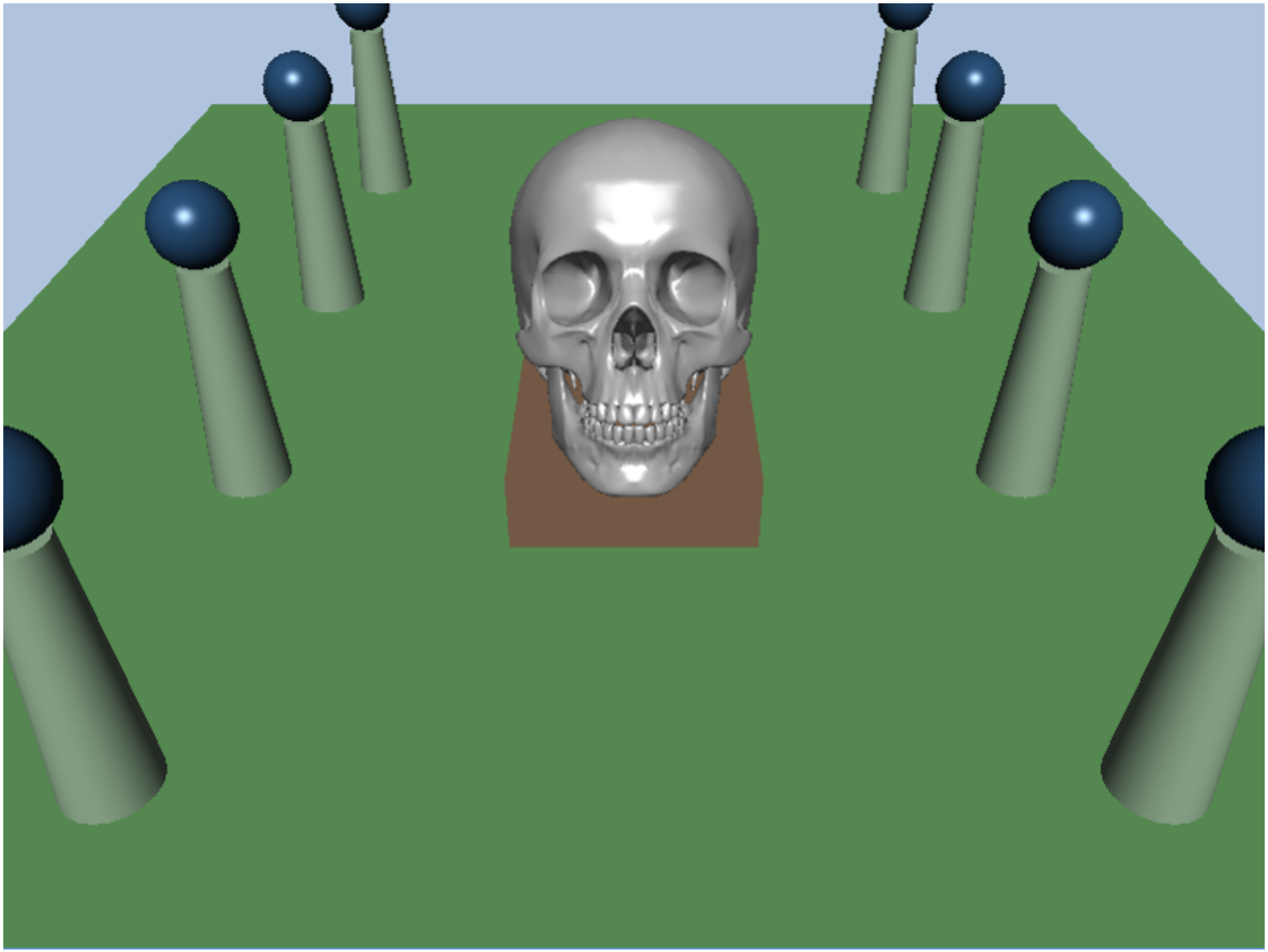


来自 icourses 胡事民

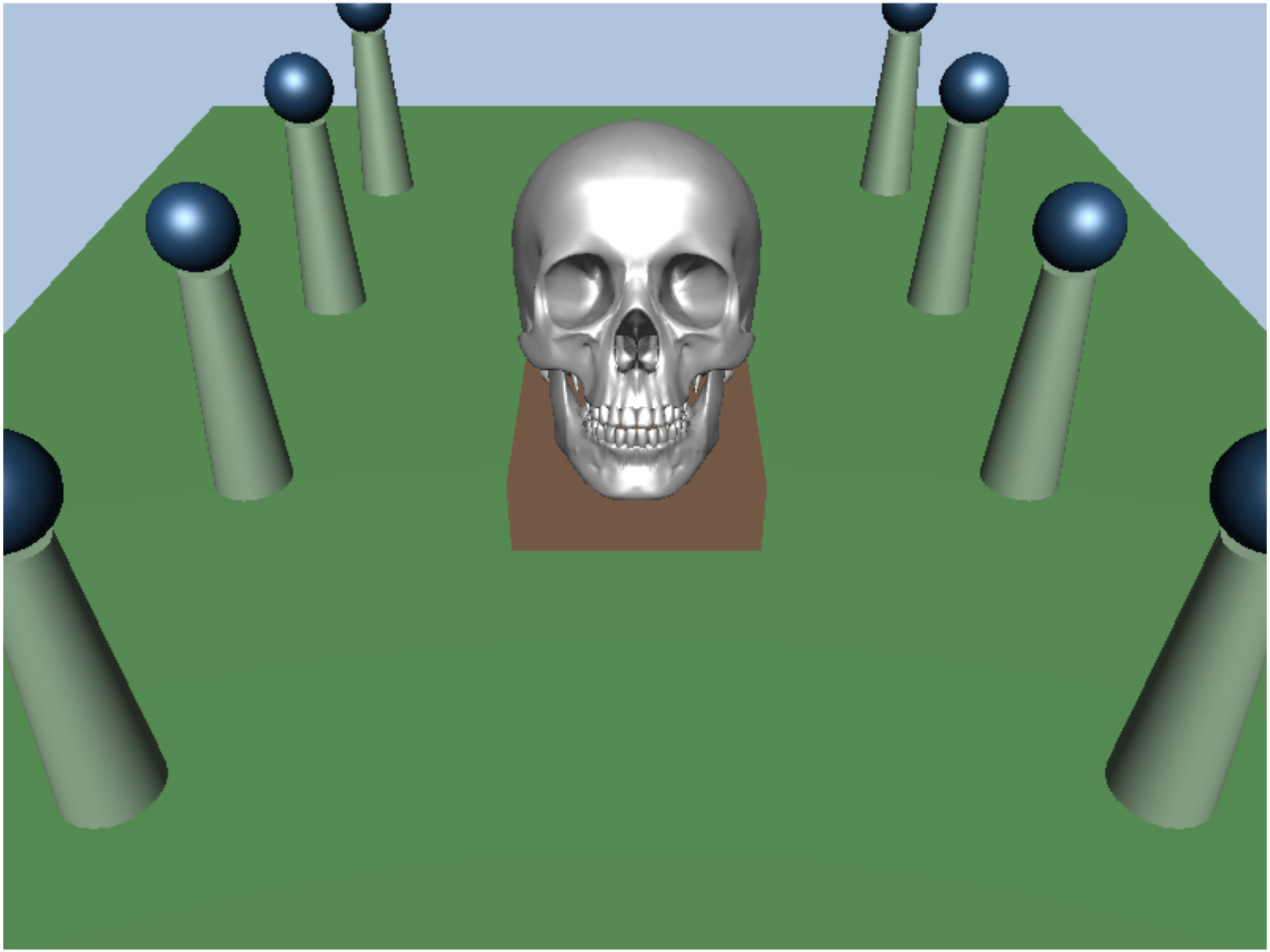
Shading 比较

通常，在一个比较复杂的场景中，当物体镜面发射很微弱时，我们对其采用 Gouraud 明暗处理，而对于一些镜面高光的物体，采用 Phong 明暗处理，这样既保证质量，又保证速度。

Gouraund 明暗处理示例

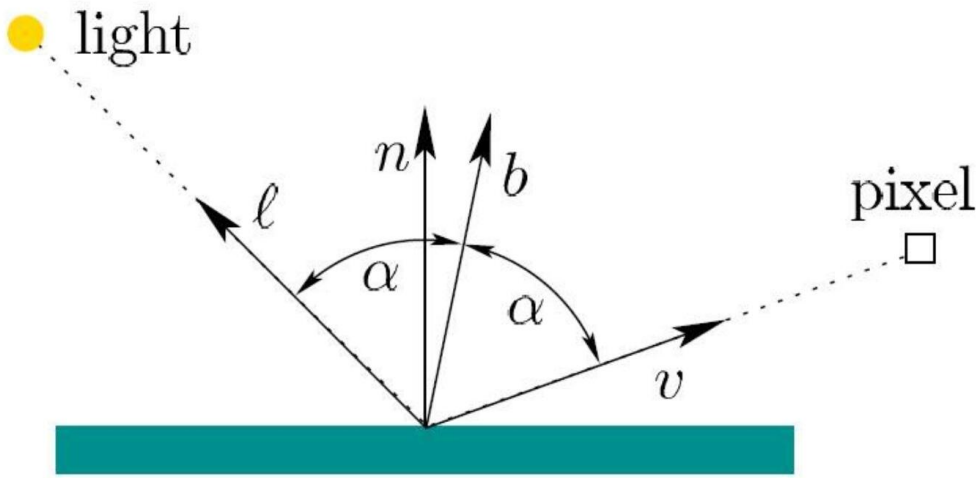


Phong 明暗处理示例



Blinn-Phong 明暗处理

相比较 Phong 模型，Blinn-phong 模型只适用 $N \cdot H$ 替换了 $V \cdot R$，但却获得了明显的提高，它能提供比 Phong 更柔和、更平滑的高光，而且速度上也更快，因此成为很多 CG 软件中默认的光照渲染方法，同时也被集成到大多数的图形芯片中，而且在 OpenGL 和 DirectX 3D 的渲染管线中，它也是默认的光照模型。



图片来自 [3]

$$L = k_d I max(0, N \cdot L) + k_s I max(0, N \cdot H)^p + k_a I_a$$

全局照明 Global Illumination

全局照明 Global illumination

- 处理多重反射: 阴影 (Shadow)、反射 (Reflection)、折射 (Refraction) 均为全局照明 (Global Illumination) 效果
- 计算更复杂：全局光照计算量大，一般也没有特殊硬件加速 (通常只使用 CPU 而非 GPU)，所以只适合离线渲染 (offline rendering)，例如 3D Studio Max、Maya 等工具。

Outline

1. 光线跟踪和明暗处理
2. 光栅化渲染
3. VR 中的渲染问题
4. 沉浸式照片和视频

基于对象的渲染

光线投射操作很快成为瓶颈。对于 90Hz 的 1080p 图像，每秒需要执行 180 多万次，并且将针对每个三角形执行光线-三角形相交测试。
在大多数情况下，从这种图像顺序渲染切换到物体顺序渲染效率更高。

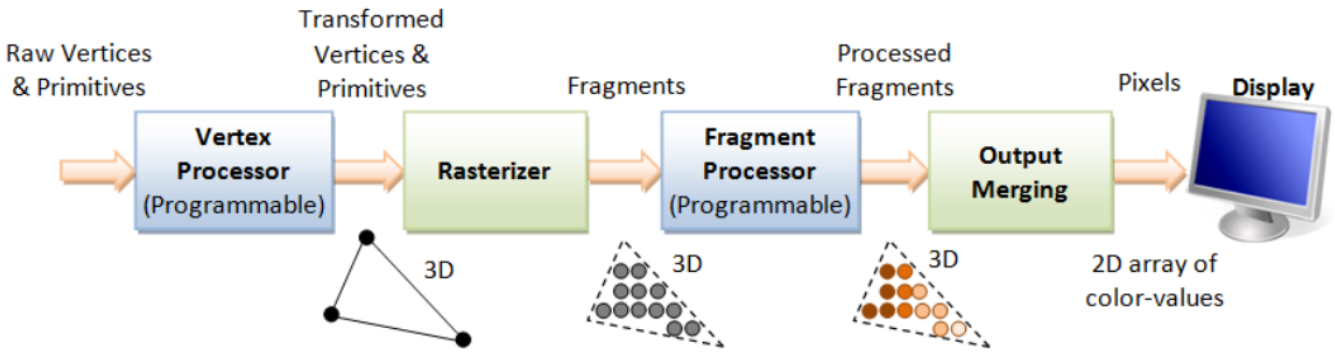
基于对象的渲染

- 光栅化：将输入的图元转换为屏幕坐标对齐的片元 (fragment) 的过程。
- 一个三角形接着一个三角形

主要问题在于该方法不可避免的必须解决确定哪个部分最接近焦点（大致为虚拟眼睛的位置）这个问题。

光栅化

渲染管线：



图片来自 3D Graphics with OpenGL Basic Theory)

- 输入是顶点以及由 gl 命令指定绘制的图元类型 (点、直线、三角形等)，顶点着色器输出经过坐标变换后的顶点，这一阶段由顶点着色器完成。
- 光栅处理：运用光栅扫描转换算法（scan-conversion）将输入的图元转换为片元（fragment）
- 片元着色器处理输入的片元，决定了最终屏幕上每个像素的颜色。

栅格化渲染

栅格化的最大优势是计算量比较小，适合实时渲染。

- 电脑游戏大多使用光栅化渲染器。事实上，这些大部分应用于游戏制作的 API 主要为实时渲染（Real-time Rendering）
- 只支持局部照明：局部照明关注物体直接受到光源影响所产生的光照效果。
- 阴影 (Shadow)、反射 (Reflection)、折射 (Refraction) 均为全局照明 (Global Illumination) 效果。在实际应用中，栅格化渲染系统可以使用预处理 (如阴影贴图 (shadow mapping)、环境贴图 (environment mapping)) 去模拟这些效果。

景深顺序

光栅化渲染把大量三角形画到屏幕上。哪个部分最接近焦点呢?

- Painter's 算法: 按照从远到近的顺序画出三角形。这个算法有什么问题呢？排序算法复杂、景深循环

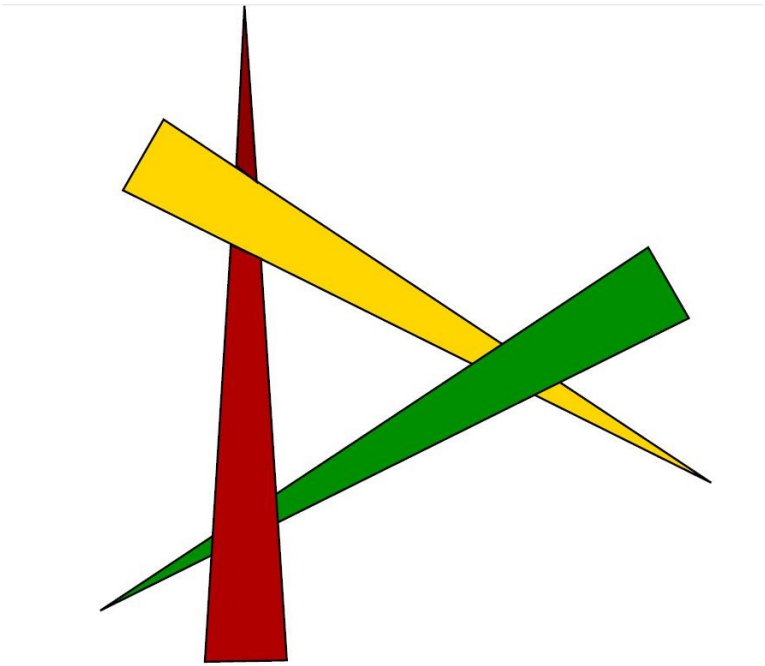


Figure: 由于景深循环的可能性，对于距观察者的距离，物体不能按三维排序。每个物体都部分位于一个物体的前面，部分位于另一个的后面。

深度缓冲 z-buffer

一种简单而有效的方法解决这个问题是通过维护一个深度缓冲区（也称为 z 缓冲区）来逐个像素地管理深度问题。

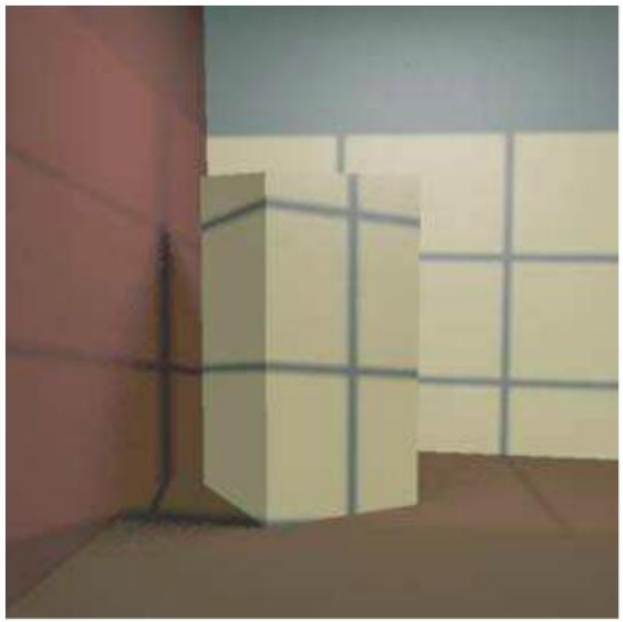
- 保存每个像素上最新渲染的点到视点的距离信息 (z)
- 只有在新的 z 值小于保存的 z 值的情况下渲染该像素

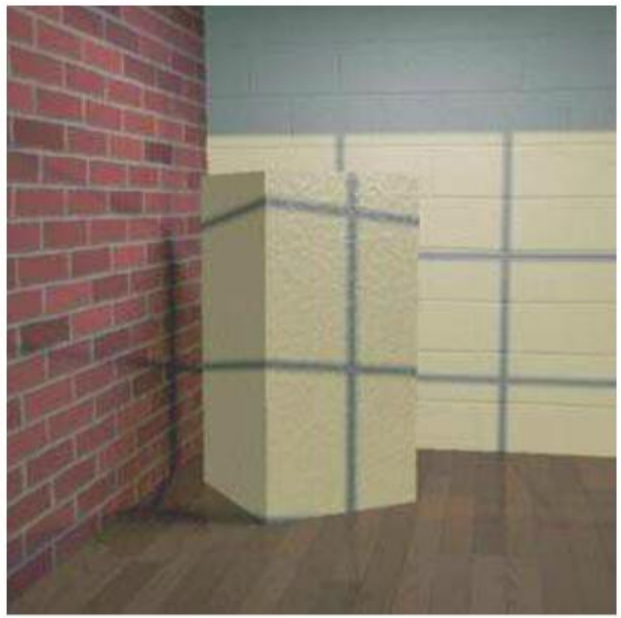
映射表面（Mapping the surface）

- RGB Mapping：重心坐标为在三角形上线性内插值提供了一种简单而有效的方法
- 纹理贴图 Texture Mapping: 可以在物体表面传播重复的图案
- 法向贴图 Normal Mapping:，通过在三角形上人工改变表面法线来改变明暗处理的过程。例如凹凸贴图，通过不规则地扰动法线使得平坦的表面变得粗糙。

纹理 Texture

- 纹理：近似真实细节的图像。
- 纹理映射 (Texture mapping) 或贴纹理：将纹理应用于物体表面的过程。
- 纹理映射的好处：
  - 提高建模效率，使得用户能不必对模型的几何细节进行建模来表示丰富的表面细节，可以极大地减少模型的网格数量。
  - 提高渲染的效率





来自 icourses 胡事民

纹理映射 Texture Mapping

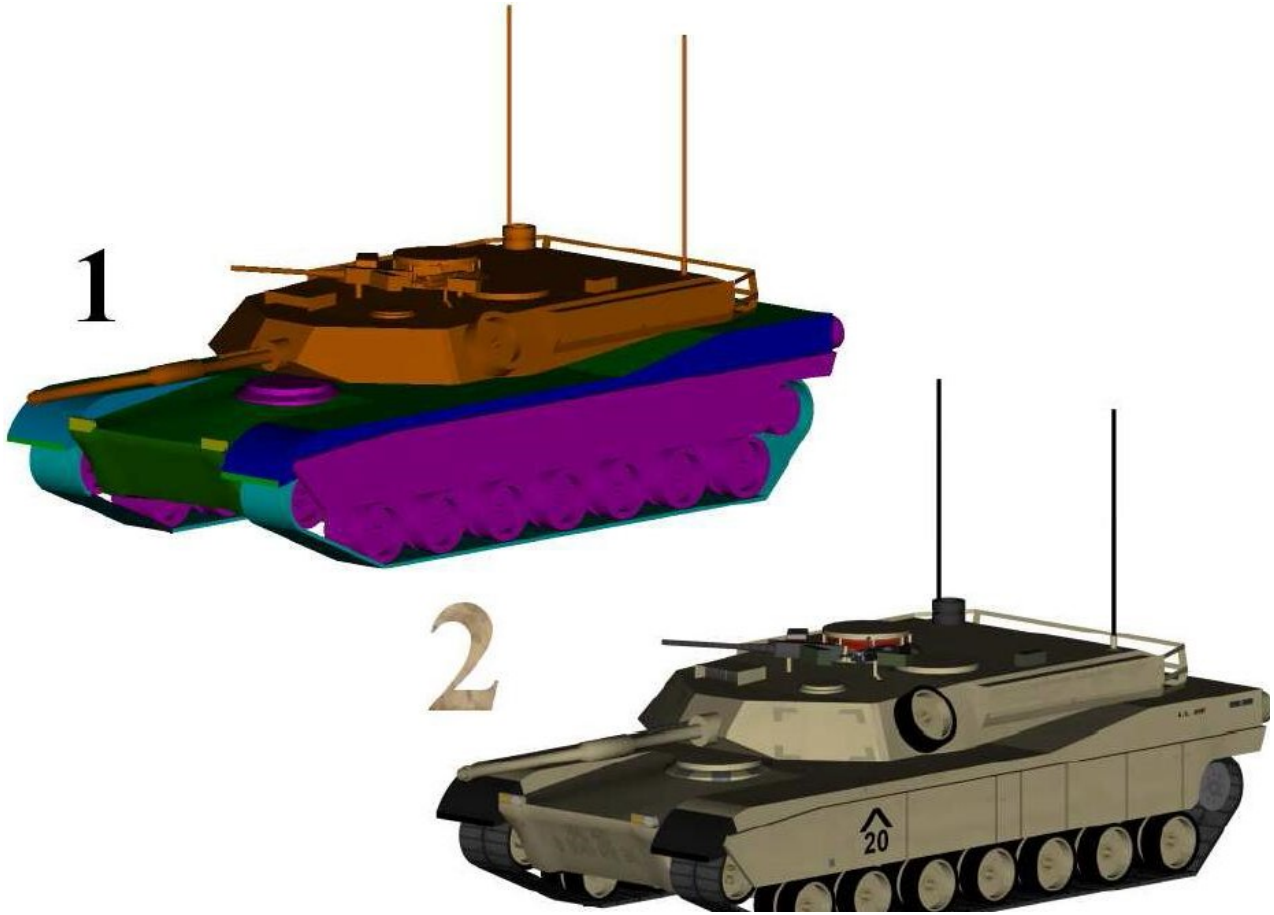
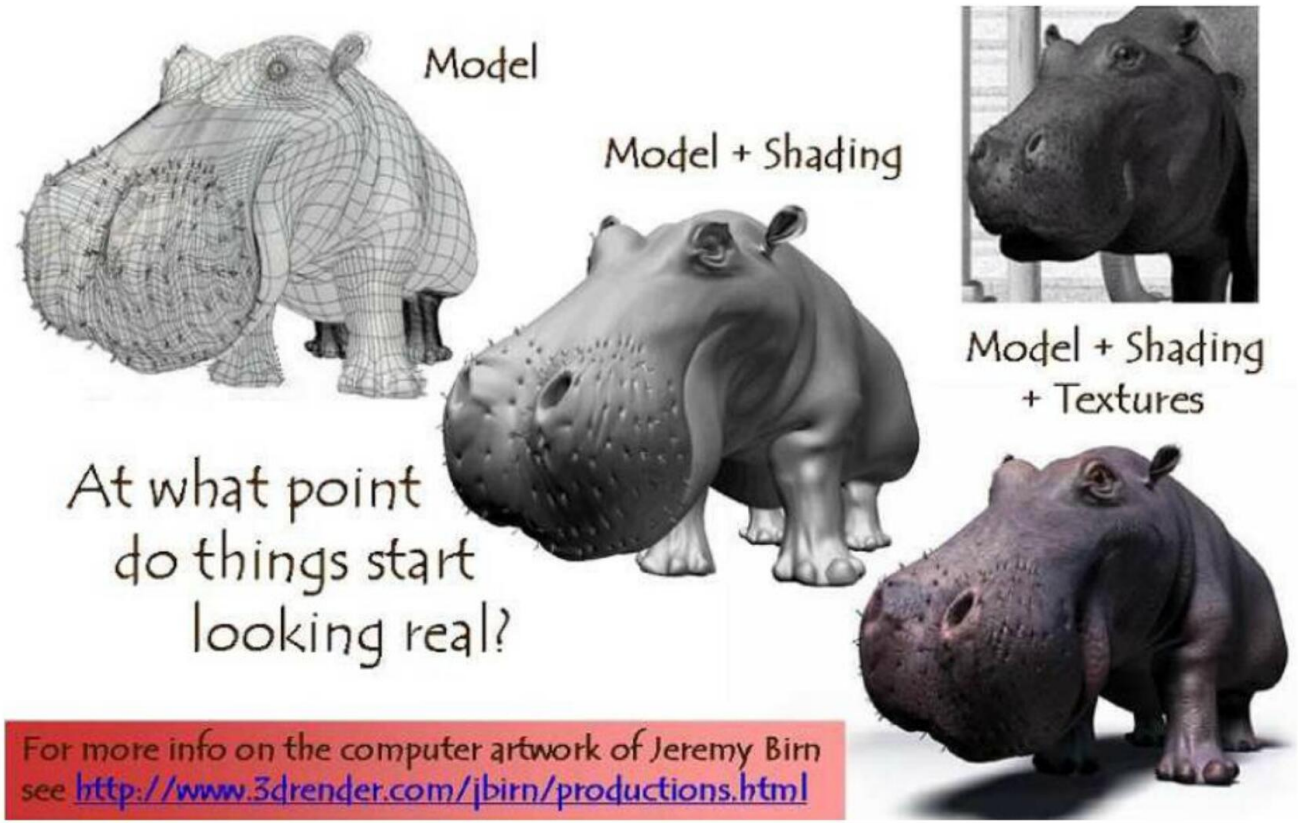


Figure: 纹理贴图：将简单的图案或是整幅图像映射在三角形纹理中，然后在图像上进行渲染，相比于直接使用模型中的三角形纹理，这样可以提供更多细节。（图源于 Wikipedia）

纹理映射


Model
Model + Shading
Model + Shading
+ Textures
At what point
do things start
looking real?
For more info on the computer artwork of Jeremy Birn
see http://www.3drender.com/jbirn/productions.html

图片来自 [3]

凹凸映射 Bump Mapping

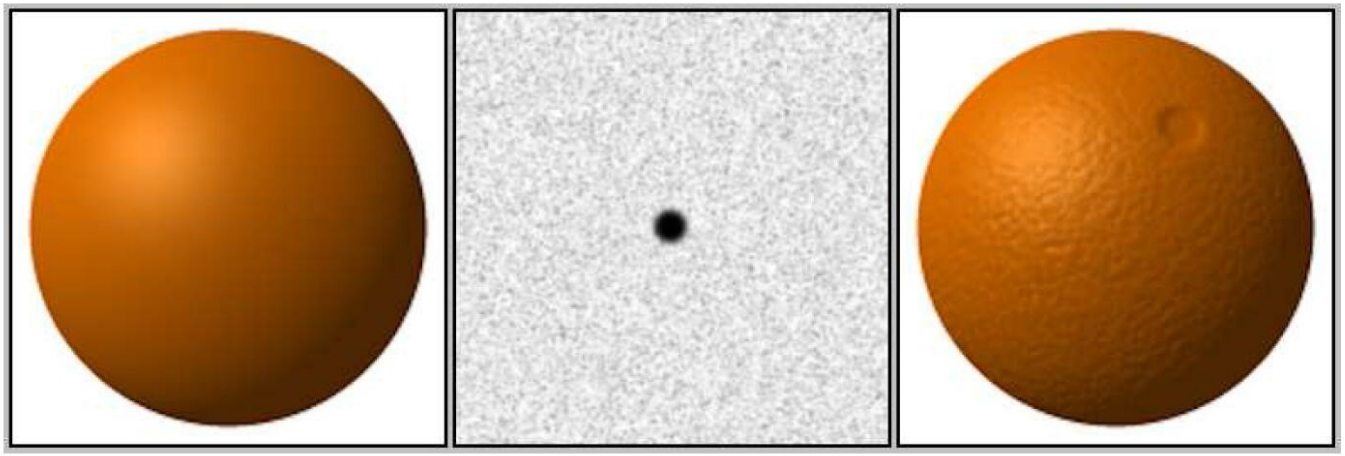


Figure: 凹凸贴图：通过人工改变表面法线，阴影生成算法会产生一个看上去粗糙的表面

图片来自 [3]

走样 Aliasing

由于数字图像的离散化，可能会出现一些缺陷。由于像素密度不足，导致一条直线看上去是阶梯状的。

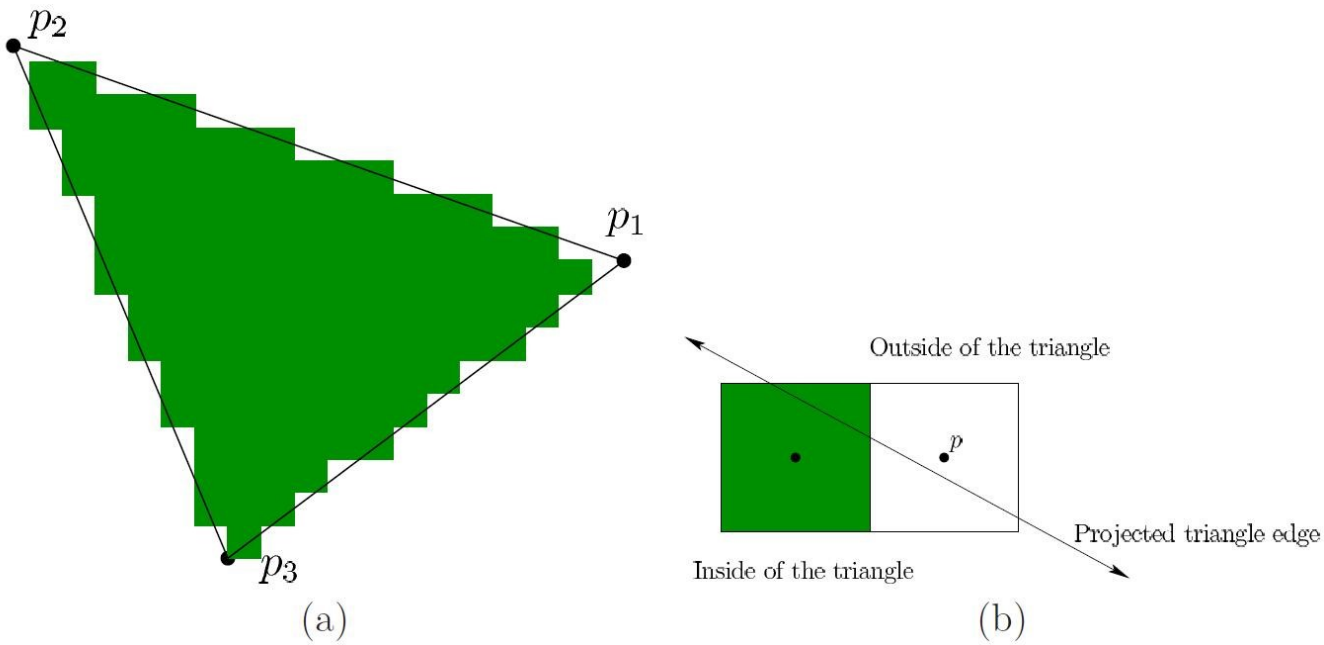
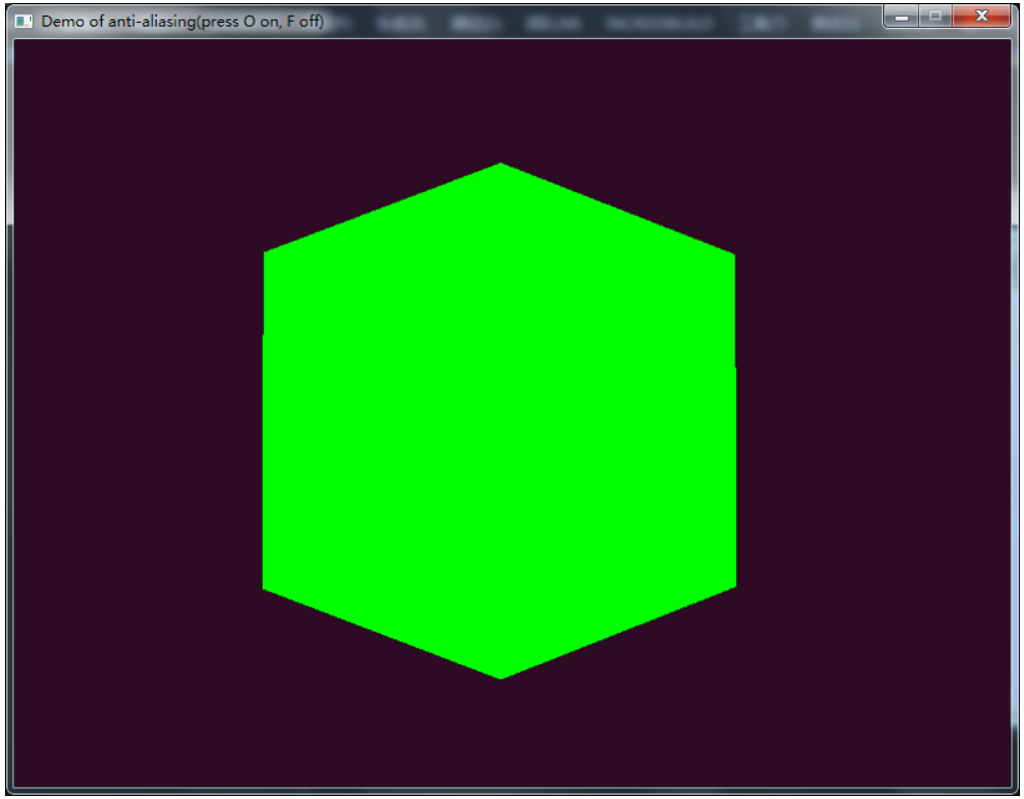


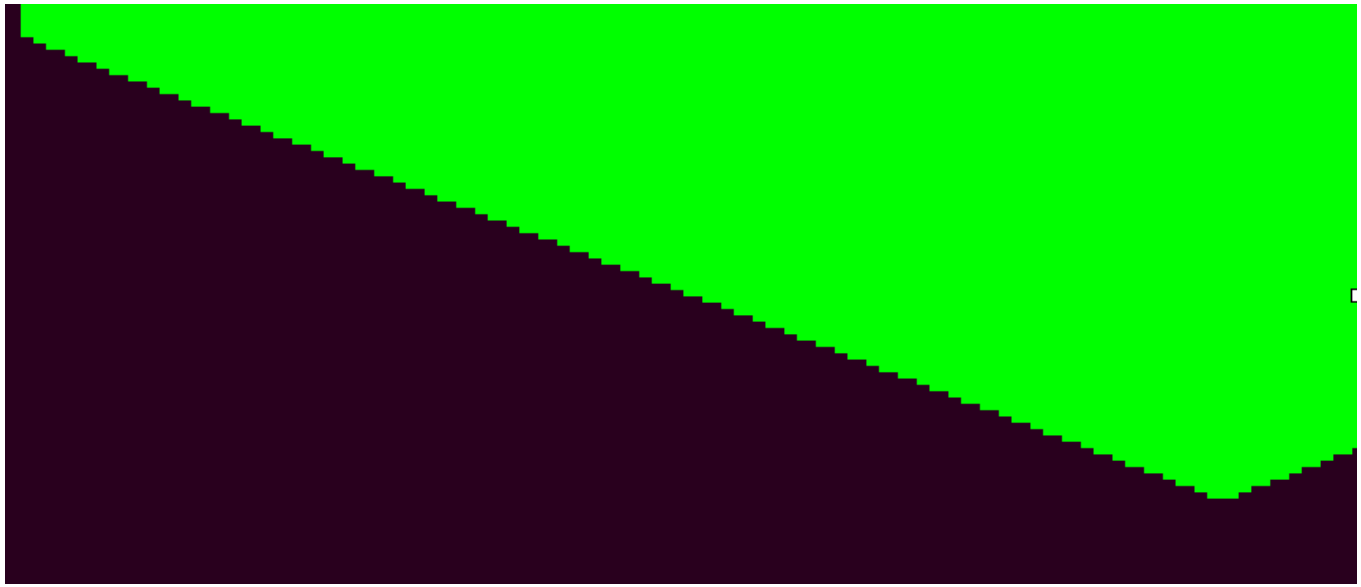
Figure: （a）光栅化导致走样现象：直线看上去像是阶梯状的（b）根据像素的中心点 p 是否在三角形内部来选取像素

图片来自 [3]

走样 Aliasing



走样 Aliasing



走样 Aliasing

- 这种绘制的物体边缘部分出现锯齿的现象称之为走样 (aliasing) 。
- 反走样 (抗锯齿，Anti-aliasing) 是减轻这种现象的方法。反走样本身比较复杂，深入了解需要信号处理背景知识.

走样 Aliasing

走样出现的原因

- 由于高分辨率下的来源信号或连续的模拟信号能够存储较多的数据，但在通过取样（sampling）时将较多的数据以较少的数据点代替，部分的数据被忽略造成取样结果有损，使机器把取样后的数字信号转换为人类可辨别的模拟信号时造成彼此交叠且有损。

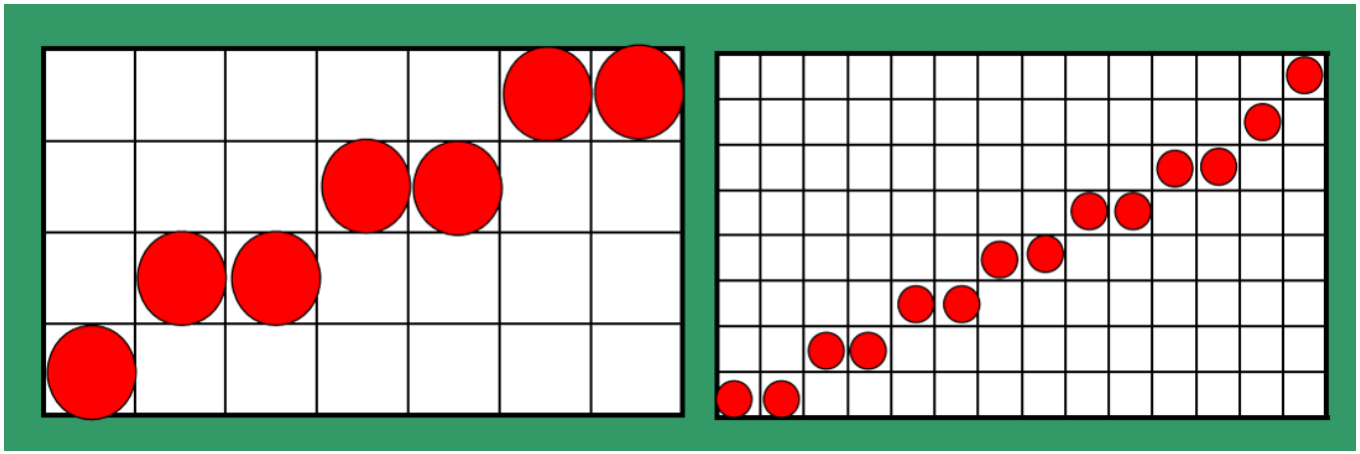
走样 Aliasing

走样出现的原因

- 在 3D 绘图时，每个图形由像素组成，每段瞬间画面由帧组成，因为屏幕上的像素有限，如果要表现出多边形的位置时，因技术所限，使用绝对坐标定位法是无法做到的，只能使用在近似位置采样来进行相对定位。由于没有足够的采样来表现出 3D 世界中的所有物品的图形，所以在最后图像显示上，这些现象便会造成在物品与物品中过渡的边缘就会产生波浪状、圆形、锯齿和闪烁等有损现象，严重影响了画面的质量。

超级采样 Super-sampling

仅仅通过点 p 的坐标来决定其是否完全包含在三角形中的话，阶梯效应是无法避免的。更好的方法是根据三角形覆盖的像素区域来渲染像素。这样的话，它的值可以由像素区域内多个三角形混合而成。



超级采样反走样方法 (Super-Sampled Anti-Aliasing ,SSAA)，是通过以更高的分辨率来采样图形，然后再显示在低分辨率的设备上，从而减少失真的方法。

超级采样 Super-sampling

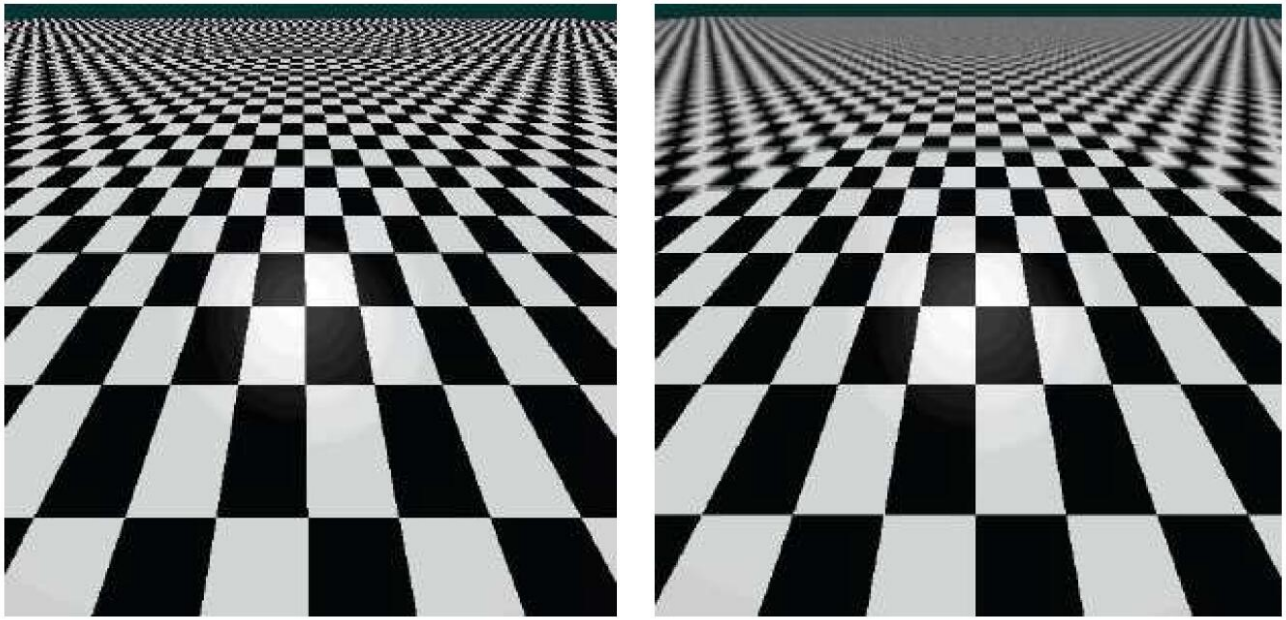


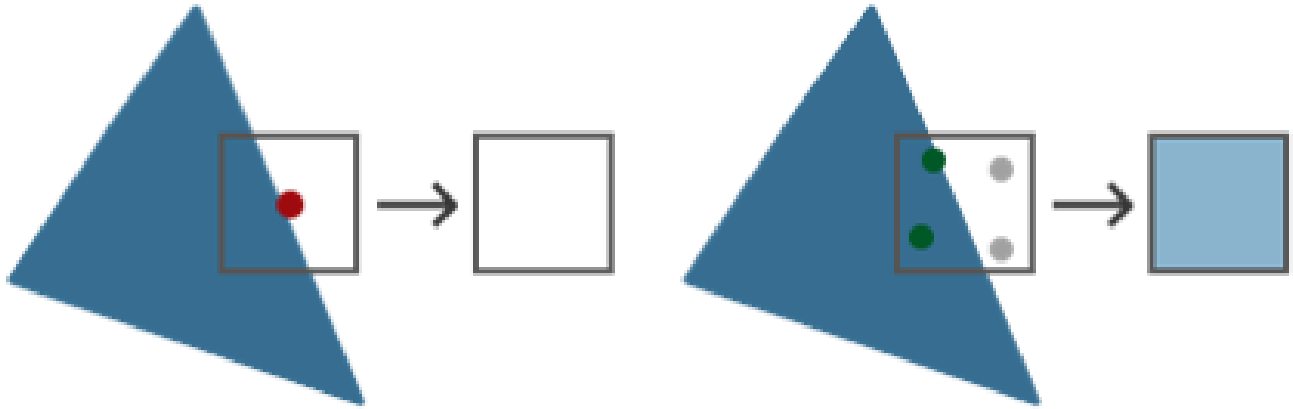
Figure: （a）由于透视变换，随着深度的增加，平铺的纹理会产生空间走样现象（b）可以通过超采样来解决该问题

超采样，意味着要以比像素密度高得多的密度来投射射线，从而估计三角形所覆盖的部分。这样的话代价会大大增加

图片来自 [3]

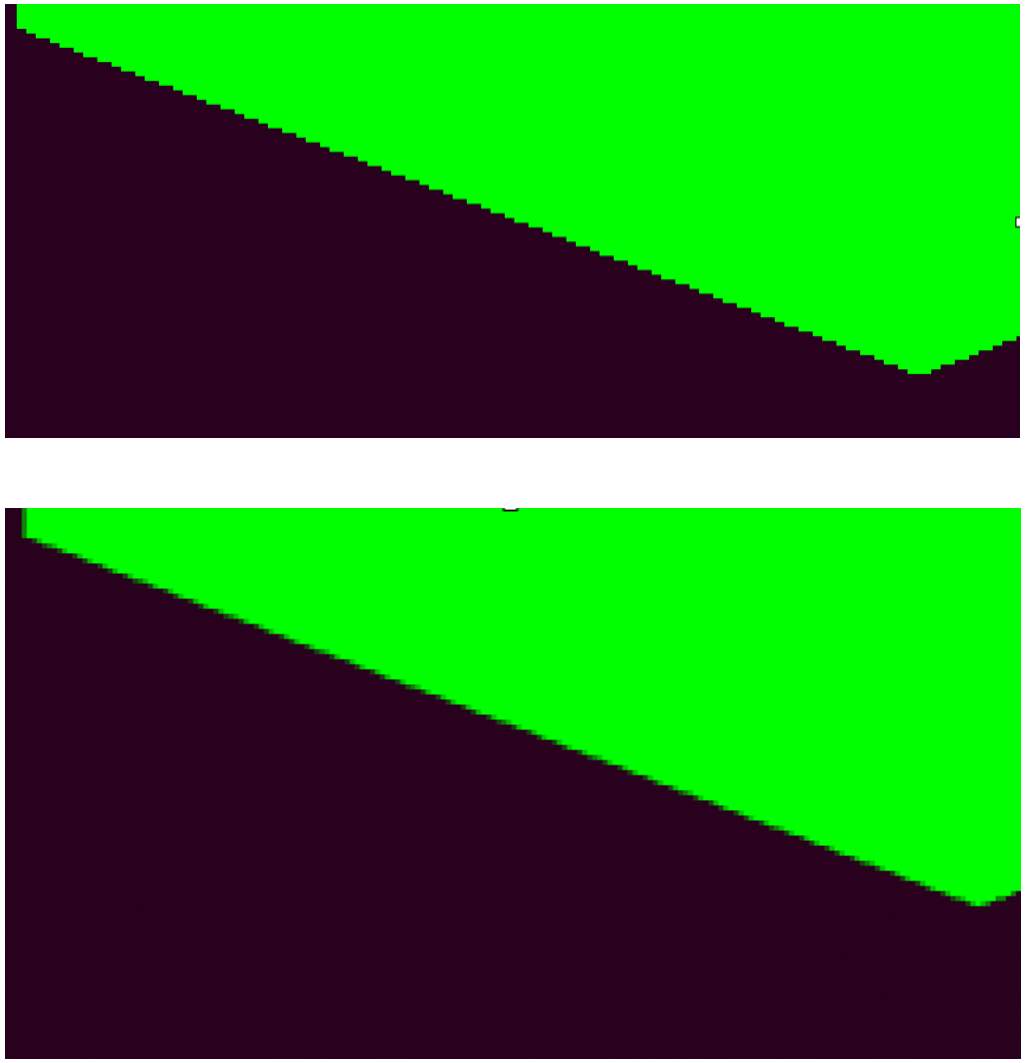
多采样 Multisampling

通常，可以使用一种折中的方案。



- 多采样技术 (Multi-Sampled Anti-Aliasing, MSAA) 是对 SSAA 的改进，改进之处在于执行片元着色器的次数并没有明显增加，对边缘部分却进行了很好的反走样。
- 例如上面的图中，三角形图元覆盖了 2 个采样点，那么这个像素的最终颜色由三角形覆盖的 2 个采样点的颜色和另外两个采样点的颜色 (例如这个颜色可能是 glClearColor 指定的颜色) 混合后的均值决定。

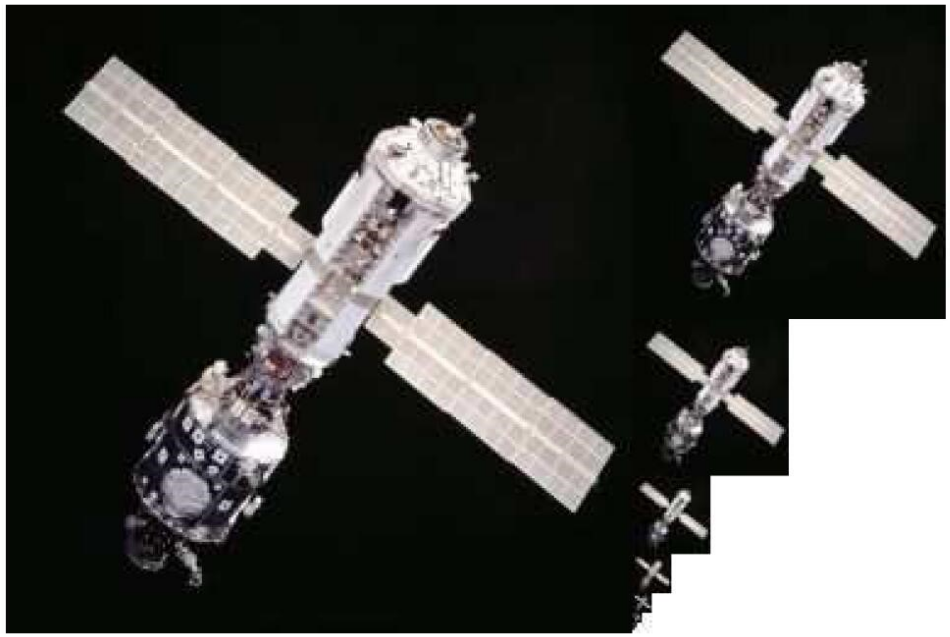
多采样效果 Antialiasing



Good but expensive.

Mipmapping

空间走样的问题是由纹理贴图引起的。将虚拟世界映射到屏幕上时，视变换会显著地减少原始纹理的大小和纵横比, 用来表示纹理中的重复图案的分辨率可能会有所不足。
在实践中，该问题通常通过预先计算和存储每个纹理的纹理贴图金字塔（mipmap）来解决。
基于屏幕上三角形的大小和视点，选择适当的缩放纹理图像并贴图到三角形上，可以减少走样缺陷现象。

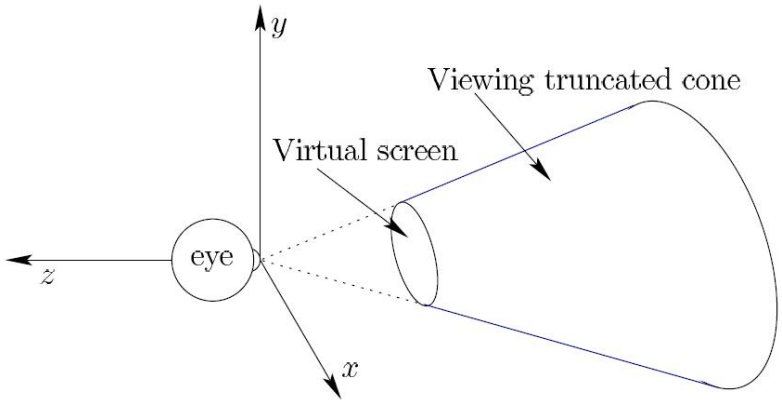


剔除

在实践中，许多三角形可以在渲染之前被快速地消除.。这是预处理阶段一种叫做剔除的方法，可以显著提高性能，也可以提高帧率.

视域体积剔除（View volume culling）：

- 可以消除完全位于视锥体之外的所有三角形
- 对于 VR 头显，由于光学系统的限制，视锥体可能会有弯曲的横截面，视锥体必须被替换为一个具有合适形状的区域
- 可以在 GPU 中使用模板缓冲来标记镜头视域外的所有像素



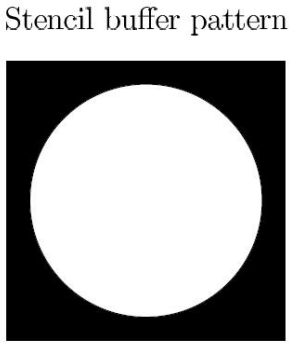


Figure: 由于屏幕前方的光学系统，在圆形对称视图的情况下，视体将

剔除

- 可以在 GPU 中使用模板缓冲来标记镜头视域外的所有像素

背面剔除（Backface culling）: 用于消除具有远离焦点的外表面法线的三角形。当模型持续生成时，“背面”的部分没有被渲染的必要。
遮蔽剔除 (Oculusion culling): 可用于消除模型中可能会被更近的物体遮挡住的部分。

Outline

VR 特有的问题

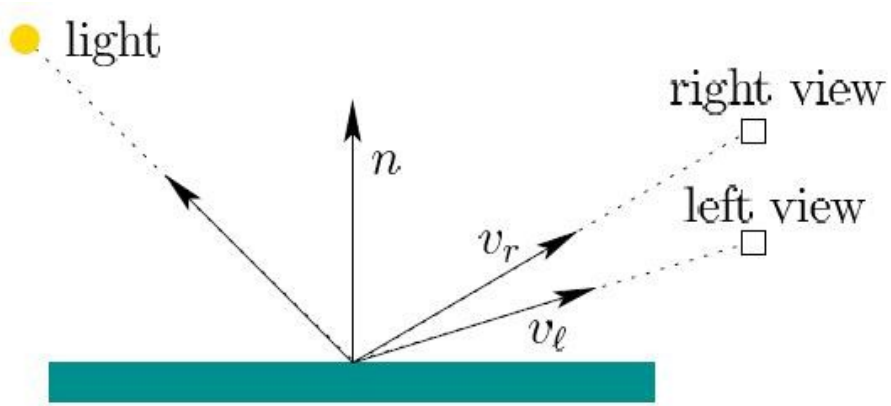


Figure: 由于右眼和左眼的视点不同，光泽表面变得复杂。使用 Blinn-Phong 阴影模型，镜面反射对于每只眼睛应具有不同的亮度级别。为了与现实世界的行为保持一致，效果可能难以匹配。

解决方案：

- 高光的部分需要立体视角

图片来自 [3]

VR 特有的光栅化问题

- 由于目前的分辨率远低于视网膜显示的下限，所以由于走样造成的阶梯状问题应该会更严重。
- 由于头部的运动导致视点不断改变，问题会变得更为严重。当用户试图注视边缘的时候，由于像素是否包含在三角形中是取决于视点的细微变化，“阶梯”似乎更像是“自动扶梯”。用户会被这种分散注意力的动作所吸引。
- 对于立体的视点来说，甚至更糟：左右侧图像中的“自动扶梯”通常并不匹配。当大脑尝试将两幅图像融合成一幅连贯视图时，走样缺陷会生成强烈的运动不匹配的感觉。

解决方案：

- 需要比以前更高的分辨率

VR 特有的光栅化问题

VR 系统带来的深度感知的增强会引起一个更严重的问题。头部运动和立体视域都使用户能感觉表面深度的微小差异。

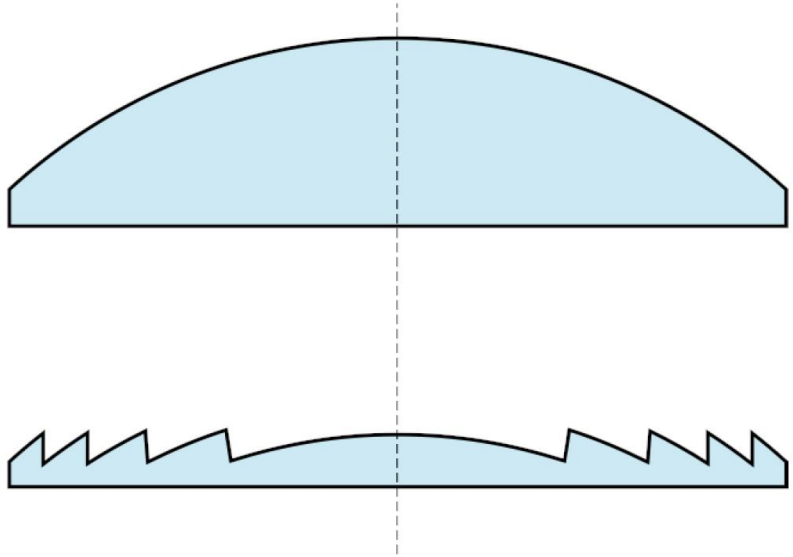
- VR 会使纹理贴图看上去并不真实。
- 凹凸/法向量贴图看起来假。

随着 VR 系统质量的提升，应该提高对渲染质量的要求，这意味着很多经典的方法应当被修改或舍弃。

光学畸变的修正

通过 VR 头显的镜头观看时，通常会产生枕型畸变。如果图像不经过任何修正的话，那么虚拟世界看上去就会出现扭曲的现象。
解决方案：

- 数字光处理（DLP）技术在不使用透镜的情况下直接将光投射到眼睛中
- 使用菲涅尔透镜，通过在较大的区域内用波纹或阶梯表面更精确地控制光线的弯曲；也可以设计成非球面的方案
- 通过软件进行修正



光学畸变的修正

通过变化半径 r 来表示镜头的畸变，而由于对称性，方向 θ 并不会被影响。
枕形和桶形畸变通常可以使用奇数次幂的多项式来近似表示，可定义 f 为：$r_d = f(r_u) = r_u + c_1 {r_u}^3 + c_2 {r_u}^5$
c1 和 c2 是常数, 用 r_u 表示未畸变的半径，r_d 表示畸变半径。
若 c1<0，则为桶形畸变；若 c1>0，则为枕形畸变.

光学畸变的修正包含两个阶段：

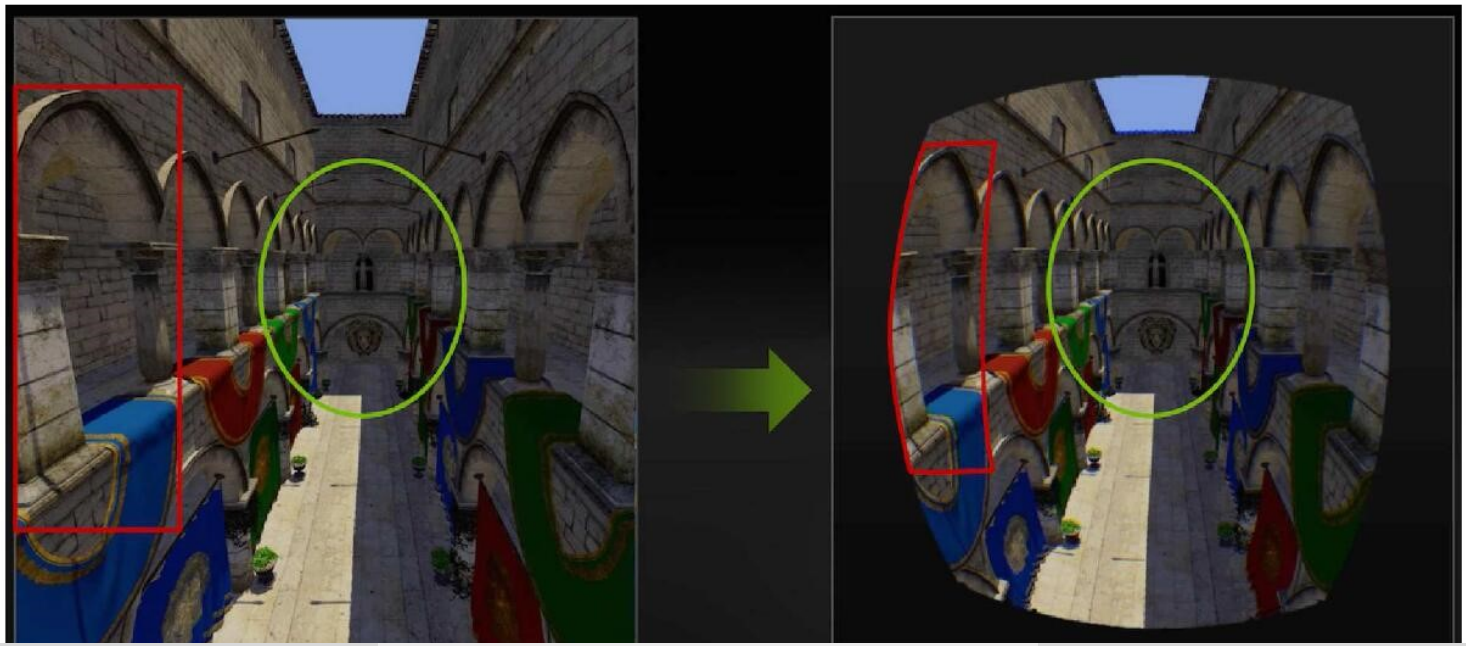
- 确定特定头显的径向畸变函数 f
- 确定 f 的逆函数，使得可以在镜头产生畸变前将其应用到图像渲染上

光学畸变的修正

多项式函数通常没有确定的或是闭合形式的逆函数，因此经常使用近似的结果，如：$f^{-1}(r_d) \approx \frac{c_1 {r_d}^2 + c_2 {r_d}^4 + {c_1}^2 {r_d}^4 + {c_2}^2 {r_d}^8 + 2 c_1 c_2 {r_d}^6}{1 + 4 c_1 {r_d}^2 + 6 c_2 {r_d}^4}$

变换 f^{-1} 可以直接应用于透视变换，从而用非线性运算代替 T_p 和 T_{can}。

在现有的图形渲染管线中，可作为一种后处理操作。图像变换的过程有时也被称为畸变着色，因为它可以作为 GPU 中的着色操作来实现；基于逐像素操作。



减小时延、提升帧率

VR 头显中的系统延时指的是从响应头部方向和位置运动到更新屏幕画面所需的时间。

- 例如，假设用户正在注视虚拟世界中的某一固定特征点。此时如果头部偏向右侧，屏幕内对应的图像特征必须要立刻向左侧移动。否则，如果图像没有变化，就会觉得特征点在移动。这破坏了平衡的感觉。

历史问题

- 在早期的 VR 系统中，系统延时经常会超过 100 毫秒。
- 在 20 世纪 90 年代，延迟减少到了 60 毫秒。时延被认为是导致 VR 疾病的重要原因之一，因此这也是过去几十年 VR 广泛运用的一大障碍。
- 由于最新一代的跟踪、GPU 和显示技术，时延不再是大多数 VR 系统的主要问题。现在的时延可以控制在 15 到 25 毫秒左右，甚至还可以通过跟踪系统的预测方法进行补偿，达到接近 0 的有效时延。现如今其他因素对 VR 疾病和 VR 疲劳的影响更大，如体感和光学像差等。

减少时延的方法概述

以下的方法可以一起使用，从而减少时延并最小化剩余时延带来的副作用：

- 降低虚拟世界的复杂度
- 提高渲染管线的性能
- 消除从渲染图像到切换像素之间的延迟
- 预测估计未来的视点和状态
- 移动或扭转渲染图以补偿上一时刻的视点错误和丢失的帧

简化虚拟世界

虚拟世界是由几何图元组成的，通常是三维网格中排列的三角形。对于每个三角形，变换和光栅化的过程是必须的，从而导致与三角形数量成正比的计算代价。
除了减少渲染时间之外，简化模型还会降低虚拟世界生成器（VWG）的计算需求。

Figure: 可使用网格简化算法来减少模型的复杂度，同时保留重要结构。

从渲染图像到切换像素

Figure: 如果在发生显示扫描时将新的帧写入显存，则会出现裂痕，其中两个或更多帧的部分同时可见

图片来自 [3]

后渲染图像扭曲 Post-rendering image warp

补偿时延的方法：使用预测的方法，进行后渲染图像扭曲 post-rendering image warp
四步：

- 读取最新的头部姿势（位置和方向）
- 用第 1 步读取的数据把场景渲染到缓存中
- 读取最新的头部姿势
- 调整渲染缓存来“伪造”出新的视点

后渲染图像扭曲

缓存调整：

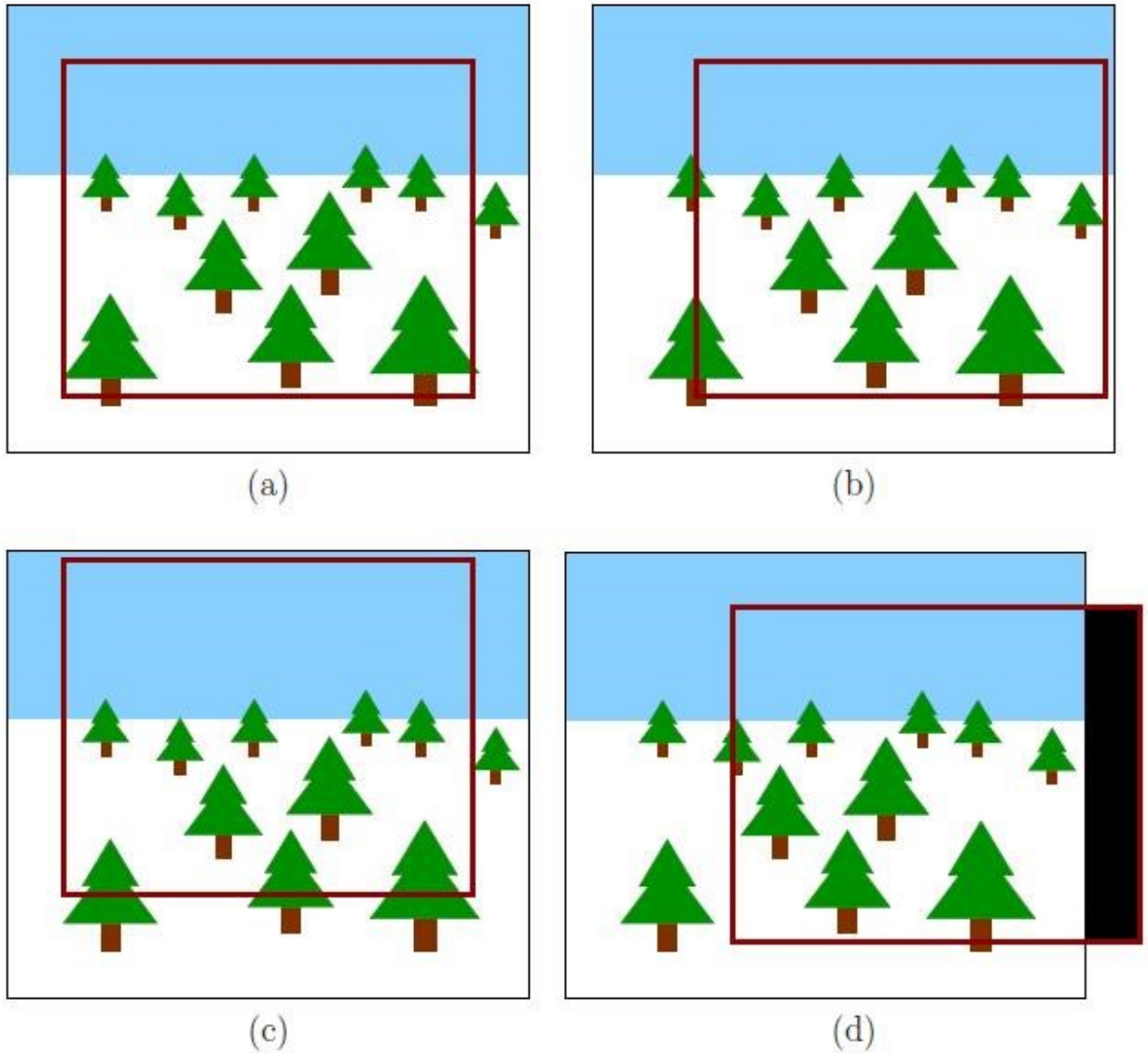
- 对于小角度的俯仰和偏航，进行图像的平移
- 对于小角度的翻滚，进行图像的旋转
- 对于 z 轴的平移（景深变化），对图像进行缩放
- 对于 x 轴和 y 轴的变化，对图像进行平移

Perturbation	Image effect
$\Delta\alpha$ (yaw)	Horizontal shift
$\Delta\beta$ (pitch)	Vertical shift
$\Delta\gamma$ (roll)	Rotation about image center
Δx	Horizontal shift
Δy	Vertical shift
Δz	Contraction or expansion

Figure: 基于视点变化的六种自由度渲染图像的扭曲。前三个对应于一个方向。剩下的三个对应一个位置变化。这些操作可以通过打开数码相机并观察图像在每个扰动下如何变化来可视化。

图片来自 [3]

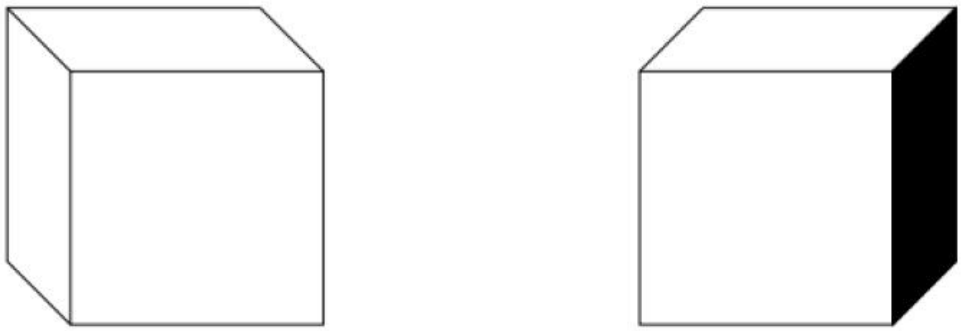
后渲染图像扭曲



扭曲图像中的缺陷

- x 和 y 中的扰动不考虑运动视差，但是需要知道物体的深度。
- z 中的变化会产生类似不正确的图像，因为附近的物体应该比其他物体扩展或收缩更多。
- 视点位置的变化可能导致可视性事件，其中物体的一部分可能仅在新视点中变为可见

图像变形的一种替代方法是使用并行处理来对未来的几个视点进行采样并为所有视图渲染图像。然后可以选择最正确的图像，以大大减少图像扭曲缺陷。



图片来自 [3]

Outline

沉浸式照片和视频

数十年计算机图形学研究所开发的方法已针对从几何模型综合构建的虚拟世界进行了研究。不过，这种趋势最近发生了变化，人们更想捕捉真实世界的图像和视频，然后轻松嵌入到 VR 体验中。原因:

- 智能手机行业人们随身携带高分辨率相机。
- 3D 相机技术不断前进，除了颜色和光线强度之外，还提供距离信息。

这些技术正迅速地融合到全景图的情况中，全景图包含从所有可能的观察方向获取的图像数据。当前的挑战是也捕获所有可能的观看位置和方向的区域内的数据。

捕捉更广阔的视野

相机无法在一个瞬间从单台相机捕捉全景照片，存在两个选择：

- 每次使用一台摄像机拍摄多张图像，每次指向不同方向，直至覆盖所有观看方向的整个球体。
- 使用多个摄像机，指向不同的观看方向，以便通过拍摄同步的图像覆盖所有方向。

Figure: （a）360Heros Pro10 HD 是一台可以在相反方向安装 10 台 GoPro 摄像机以捕获全景图像的装备。（b）理光 Theta S 仅使用两台摄像机拍摄全景照片和视频，每台摄像机均提供一个视场大于 180 度

映射到球体上

- 在 VR 中渲染光球时直接映射到虚拟世界中的球体上。所有可能的观察方向的球体映射到虚拟世界球体而没有畸变。然而，相邻像素的结构（上，下，左，右）并不清楚。
- 将光球表示为六个方形图像，每个图像对应一个立方体的面。

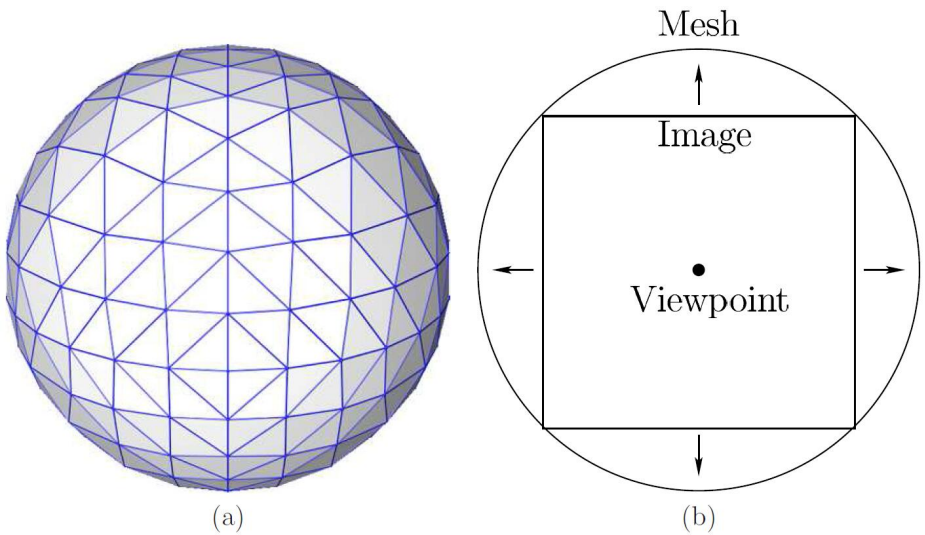


Figure: 7.24：（a）光斑纹理映射到球体的内部，被建模为三角形网格。（b）以六个图像的立方体形式存储的光球可以快速地映射到球体，分辨率损失相对较小; 这里显示了横截面。

映射到球体上

- 一旦光球（或电影球体）被渲染到虚拟球体上，渲染过程就非常类似于渲染后图像扭曲。
- 实际上，整个光栅化过程只能对整个球体执行一次，而渲染到显示器的图像则根据观察方向进行调整。
- 进一步优化可以通过绕过网格并直接从捕获的图像形成光栅化图像来完成。

全景的光场

- 全景图像没考虑因用户移动任意视角环绕世界将如何出现。
- 理想的情况是捕获用户允许移动任何观察体积内的整个光场能量。光场提供空间每个点处的光传播光谱功率和方向。

Figure: Figure Digital 的 Pantopticam 原型使用数十个相机来提高逼近更多视点的能力，从而可以模拟立体观看和位置变化引起的视差。

Any Questions?