

图形学基础导论及现代 GPU 架构介绍

了解认识现代计算机图形学

x

于 y

2024 年 11 月 2 日



Opening Minds • Shaping the Future
啟迪思維 • 成就未來

Contents

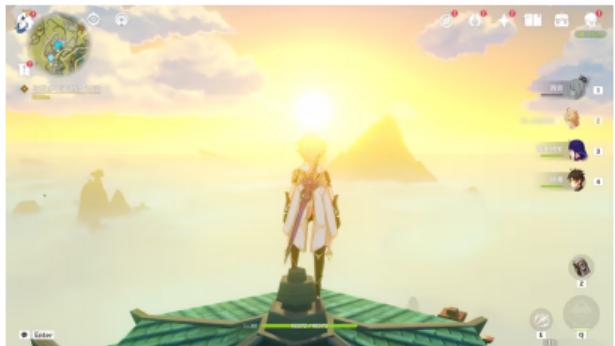


- ▶ 图形学简介
- ▶ 初步认识图形学
- ▶ 图形学背后的原理
- ▶ 光栅化渲染简介
- ▶ 光线追踪简介
- ▶ 结语

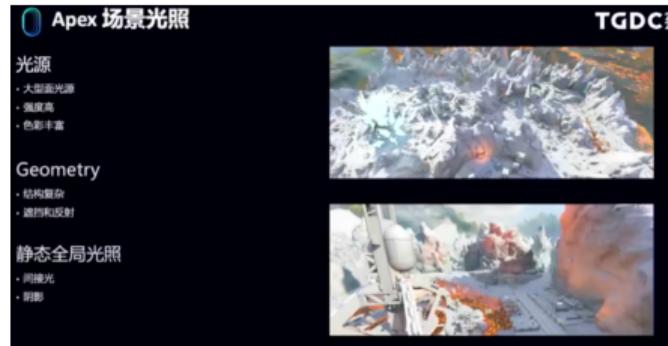
图形学的实际应用



当你在游玩你最爱的游戏的时，在赞叹美术能力的高超、策划能力的出众的同时，图形学也在背后起了无可替代的作用。事实上，如果没有图形学，那所有的游戏都将不复存在，我们也再也无法赞叹一款游戏“画质高”了。



(a) 游戏“原神”画面示意图



(b) 腾讯 TGDC 上对游戏“Apex”渲染原理讲述图

图：图形学在游戏娱乐产品中的运用

图形学的实际应用



其实，图形学不仅在游戏方面有广泛的应用。在影视、虚拟现实（VR）等领域都有被采用。



(a) 迪士尼将图形学应用到虚拟现实中



中

图：图形学在影视行业与动画

Contents



► 图形学简介

► 初步认识图形学

► 图形学背后的原理

► 光栅化渲染简介

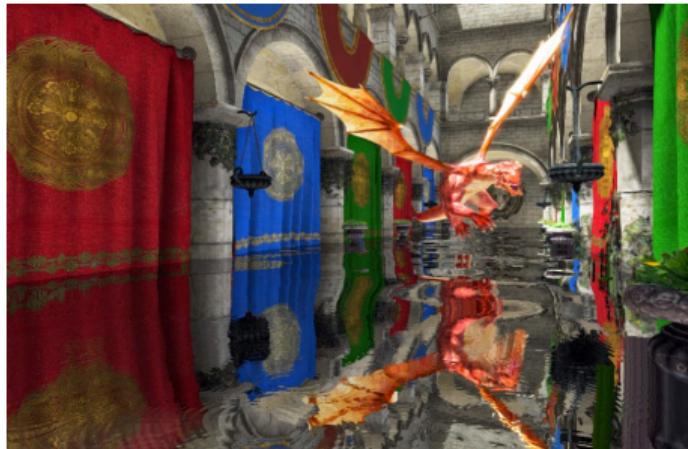
► 光线追踪简介

► 结语



根据维基百科的定义：

计算机图形学（Computer Graphics）是研究计算机在硬件和软件的帮助下创建计算机图形的科学学科，是计算机科学的一个分支领域，主要关注数字合成与操作视觉的图形内容。虽然这个词通常被认为是指三维图形，事实上同时包括了二维图形以及影像处理。



图：光线追踪效果图

我所认识的图形学



我认识的图形学，是一门极富有吸引力的学问。它就像是给予了人在虚拟世界内一只如意的画笔，通过数学几何和物理规律的推导一步一步将我们脑海中的图像用代码实现出来。

图形学并不是空想家的乌托邦，其在实际生产领域也有非常广阔的应用和行业前景。图形学对求学者做出了物理学和数学上的要求。学习图形学意味着要对数学代数几何具有比较深刻的理解，以及对现实物理规律有清晰地把握，只有扎实的基础知识储备，才能在图形学的海洋里游刃有余地遨游。

Contents

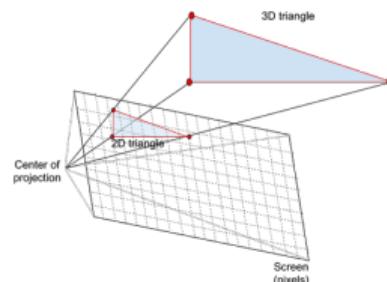


- ▶ 图形学简介
- ▶ 初步认识图形学
- ▶ 图形学背后的原理

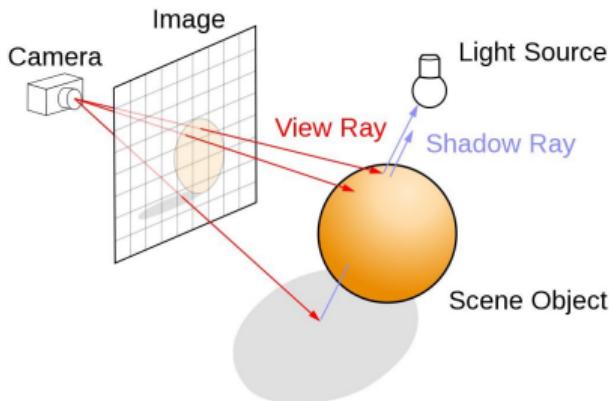
- ▶ 光栅化渲染简介
- ▶ 光线追踪简介
- ▶ 结语

两种不同的渲染方式

目前图形学图形学主要分为两大类渲染技术：光栅化渲染（Rasterization）和光线追踪（Ray Tracing）。[还有部分已经被淘汰的老技术如光线投射（Ray Casting）一种在 FC 红白机上用来渲染 3D 图像的技术，现已不再被使用]



(a) 光栅化渲染示意图



(b) 光线追踪示意图

图：两种不同的渲染方式比对

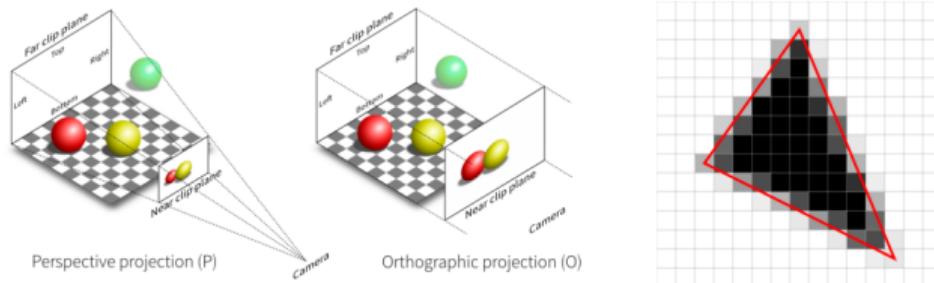
Contents



- ▶ 图形学简介
- ▶ 初步认识图形学
- ▶ 图形学背后的原理
- ▶ 光栅化渲染简介
- ▶ 光线追踪简介
- ▶ 结语

光栅化渲染简介

光栅化是一个关注于“投影”的渲染方法，其更加关心的是构建一个三维到二维的映射。什么叫做三维到二维的映射呢？其实就是画素描时常说的“透视”。但又与素描不同的是，光栅化是可以认为不考虑光照的。尽管人们发明了 Shadow Maps、SSAO、Phong、IBL、SSR 等算法来辅助光栅化算法做到光照效果，但是从本质上来说，光栅化生来不具有渲染光照的能力。

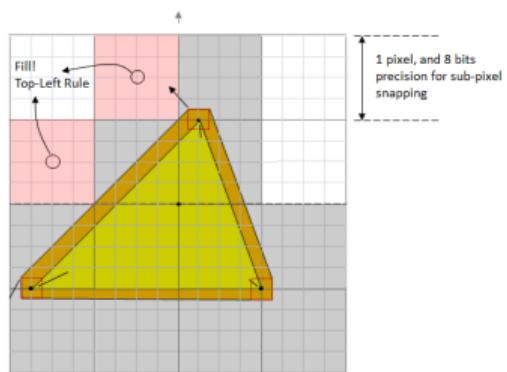


图：“透视”示意图

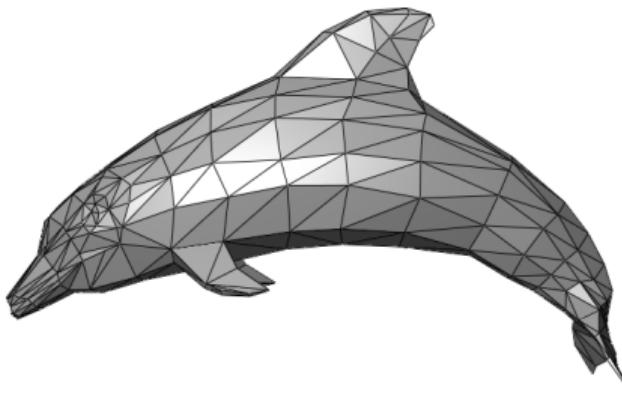
光栅化渲染基础知识



光栅化是一种基于图元可见性的算法，图元（primitive）在计算机图形学中是指被渲染的目标对象，如曲面、二次曲线、点、线段、面等。一般而言，图元都是三角形，一般的 3D 模型也是以多个三角形顶点储存的如图（b）。（想想，为什么？）



(a) 图元示意图



(b) 常见的 3D 模型一般以多个三角形顶点储存

图: 三角形图元展示



光栅化渲染简介

那么，光栅化是如何计算三维透视后的点的呢？其实，是依靠一个叫做**矩阵（Matrix）**的东西实现的。而负责投影的矩阵又称作**投影矩阵**，这个叫**投影矩阵**的东西长这样：

$$M = \begin{bmatrix} \frac{1}{r \cdot \tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{F}{F-N} & 1 \\ 0 & 0 & \frac{-NF}{F-N} & 0 \end{bmatrix}$$

其中 F, N, α 叫做**相机参数**，通俗地来讲就是调节我们视角角度的变量。 α 还有一个更为各位所熟知的名字，视场角（FOV），利用投影矩阵将三维的点投影到二维平面上的操作又叫做**离散化**。利用投影矩阵，我们就可以做到将三维的物体投影到我们二维的显示屏平面（或者说，相机平面）。



光栅化渲染总结

光栅渲染方法主要是利用数学代数计算方法计算三维空间 Ω 中的一个顶点 $P(x, y, z)$ 在二维平面（也就是我们的电脑屏幕） Ω' 的投影点 $P(x, y)$ 的坐标来实现渲染 3D 图像并呈现图形内容的。

光栅化作为一种快速、直观的渲染方法，一直在图像渲染领域有不可撼动的地位。经典的利用光栅渲染呈现画面的游戏有由 Valve 开发的“传送门 (Portal)”，还有由 Mojang 开发的“我的世界 (Minecraft)” (Java 原版)。



(a) 游戏“传送门”截图



(b) 游戏“我的世界”截图

Contents

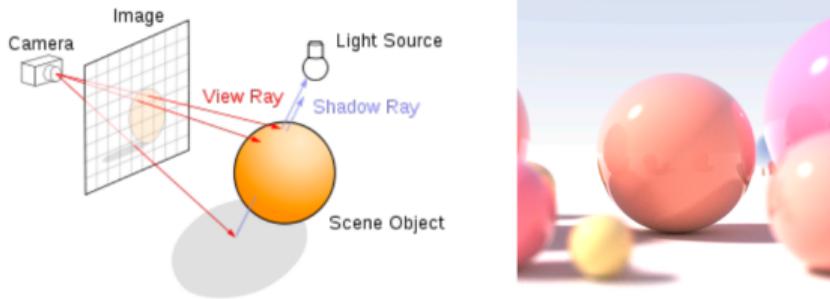


- ▶ 图形学简介
- ▶ 初步认识图形学
- ▶ 图形学背后的原理
- ▶ 光栅化渲染简介
- ▶ 光线追踪简介
- ▶ 结语

光线追踪简介



与光栅化方法截然不同，光线追踪基于对真实物理世界光线传播的模拟来实现图像的渲染。通过光线追踪技术渲染出的图像往往非常真实，因为其就是根据真实物理世界法则所制作的图像。光线追踪中也有一样的图元的概念。更与光栅化不同的是，光线追踪在数学方法上采用的是统计学方法来计算光线的入射反射。



图：光线追踪示意图

隐式几何与显式几何



而对于几何物体的数学处理方面，光线追踪又分为两个门派，一派是基于 SDF 距离函数和构造实体几何（CSG）隐式（Implicit）几何物体描述的光线步进（Ray Marching）。还有一种是基于点云或点映射进行显式（Explicit）几何物体描述的路径追踪（Path Tracing）。

Algebraic Surfaces (Implicit)

Surface is zero set of a polynomial in x, y, z



More complex shapes?

(a) 隐式几何 SDF 构造示意
图

Point Cloud (Explicit)

Easiest representation: list of points (x,y,z)

Easily represent any kind of geometry

Useful for LARGE datasets (>1 point/pixel)

Often converted into polygon mesh

Difficult to draw in undersampled regions



(b) 显式几何点云示意图

Polygon Mesh (Explicit)

Store vertices & polygons (often triangles or quads)

Easier to do processing / simulation, adaptive sampling

More complicated data structures

Perhaps most common representation in graphics



(c) 显式几何多边形 Mesh 示
意图

图：隐式、显式几何示意图

隐式几何与显式几何



简单地来说，隐式几何与显式几何的区别就在于：就相当于你问“这个圆长什么样？”，隐式几何会回答你“ $x^2 + y^2 = r^2$ （圆的标准方程）”，显式几何会告诉你“点 $P_1(x_1, y_1)$ 在圆上，点 $P_2(x_2, y_2)$ 在圆上，…，点 $P_n(x_n, y_n)$ 在圆上”。

想想：隐式几何与显式几何各有什么利弊呢？



光线追踪的基石

无论是基于何种方法的光线追踪方法，都基于一个最基础的公式，称之为渲染方程（Rendering Equation），其由 Jim Kajiya 和 David Immel et al 分别独立在 1986 被同时提出：

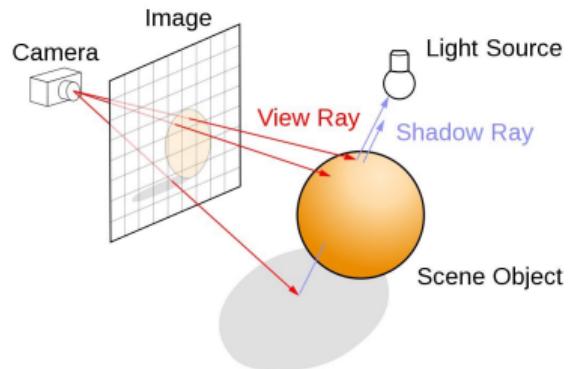
$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + \int_{\Omega} f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}') (\vec{w}', \vec{n}) d\vec{w}'$$

这个方程非常复杂，但他其实说的就是：

射出光线 = 特定角度和位置的出射光 + 物体自发光 - 光照衰减

光线追踪基本原理

事实上，光线追踪的最简原理就是高中数学学习过的“蒙特卡洛方法”，以及初中物理学习过的“光路可逆原理”。图形学中，光线追踪的原理是从“眼睛”中“发射射线”出去，再计算这条“眼睛射线” \vec{r} 的反射光线和折射光线，从而间接逆推出所有射入“眼睛”的光线从而渲染出图像。



(a) 光线追踪示意图



(b) NVIDIA 给出的光线追踪效果图

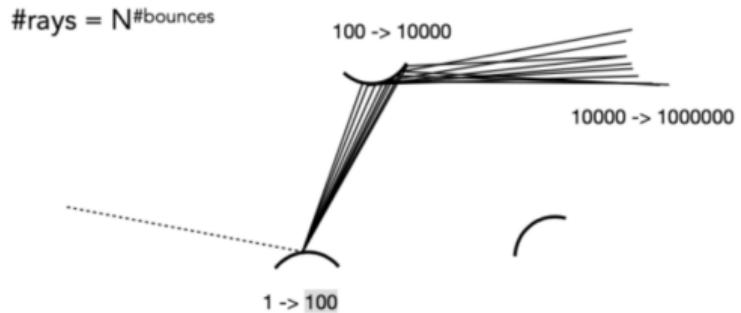
图: 光线追踪原理示意图（左图）及最终效果（右图）



光线追踪的较深剖析

从上文的介绍中，光线追踪似乎是一个“很当然”的过程，可能有人会有疑问“蒙特卡洛方法到底体现在哪里了？”。其实，如下图所示，每一条光线经过折射以后其实会产生 N 条光线（想想，为什么一定是 N 条光线？），然而，由于光线的反射可能有很多很多次，所以假如每一次计算反射就计算了 N 次反射，最终第 m 次反射将会最少产生 N^m 次计算！

这将是一个很大的值，例如 200^4 次计算，然而这只是针对一个点的计算量！那么，该如何解决这个问题呢？这就需要请出蒙特卡洛方法了。



图：一条光线经过折射以后其实会产生 N 条光线

蒙特卡洛方法在光线追踪的应用



上篇提到，既然会有 N 条折射光线，那么不妨每次就随机地计算一条，得到该点的“颜色”为 C ，显然，这个 C 与真实的“颜色”其实相差甚远，因此我们可以考虑多“试”几次，这个“试”的过程就叫做采样 (Sample)。设每次采样为函数 $f(\vec{x})$ ，每次采样 n 次，则最终得到的“颜色” C' 应为：

$$C' = \frac{1}{n} \sum_{i=1}^n f_i(\vec{x})$$

由蒙特卡洛方法可知，我们可近似认为：

$$\frac{1}{n} \sum_{i=1}^n f_i(\vec{x}) \approx L_e(x, \vec{w}) + \int_{\Omega} f_r(x, \vec{w}', \vec{w}) L_i(x, \vec{w}') (\vec{w}', \vec{n}) d\vec{w}'$$

注：此处有兴趣的同学可自行了解“大数法则”，由大数法则可知，当 $n \rightarrow \infty$ 时， $\frac{1}{n} \sum_{i=1}^n f_i(\vec{x})$ 一定会越接近真实值。

光线追踪的现状



前人的成就只是为了启迪后人更伟大的成就。渲染方程一度被认为是解决渲染问题的究极答案。直到华裔渲染学家闫令琪及其带领的团队，发表了论文 A Generalized Ray Formulation For Wave-Optical Light Transport，从新的物理层面开始着手渲染问题的解决，并给出了如下公式：

$$\Im\{g\} = \frac{1}{(2\pi)^2} \left| \int d\vec{r}' \psi_S(\vec{r}') \psi_{\beta,\rho}^*(\vec{r}'; \vec{r}_0, \vec{k}_0) \right|^2$$

以及另一个渲染公式，因篇幅所限此处不再做展开。但在已有领域还有新的发现无疑是一件令人兴奋的事情，这也在告诉我们，在未来科研的路上，不要拘于现状，看个角度看问题或许能够看到事情的另一面。

Contents



- ▶ 图形学简介
- ▶ 初步认识图形学
- ▶ 图形学背后的原理
 - ▶ 光栅化渲染简介
 - ▶ 光线追踪简介
- ▶ 结语



本想乘此机会介绍现代 GPU 的渲染架构，并且介绍其与 AI 人工智能应用的联系。但可惜根据时间预估安排，恐怕完成了 PPT 也无法有时间去讲，故只能在此深表遗憾。
如果下次还有机会能够再次站上这里，必会将此遗憾补全，请见谅。

Have you seen the new RTX 4090?



图: Jensen Huang 可爱捏



任何问题的 Q&A。