

计算机图形学：大作业报告

2019012338 范致远

January 14, 2022

1 代码框架

include/ 文件夹下包含了渲染器的所有逻辑：

- bxdf.hpp: BxDF 类，定义了双向散射函数的接口。
- bxdfs/: 实现了若干双向散射函数，包括 Lambert 与 GGX。均继承自 BxDF 类。
- camera.hpp: Camera 类，定义了相机逻辑的接口。
- cameras/perspective.hpp: 实现了带景深的透视相机。继承自 Camera 类。
- graph.hpp: 形状组，实现了 K-D 树以加速形状求交。
- light.hpp: Light 类，定义了光源的接口。
- lights/: 实现了均匀光源与聚光灯式光源。均继承自 Light 类。
- medium.hpp: Medium 类，定义了介质的接口。
- mediums/: 实现了均匀介质以及散射介质（体积光效果）。均继承自 Medium 类。
- models.hpp: 实现了 obj 模型导入的逻辑。
- object.hpp: Object 类。定义了实体的接口，用以描述了参与光线追踪的物体。
- particle.hpp: Particle 类。实现了用于散射介质的粒子。继承自 Object 类。
- sampler.hpp: 调用 erand 函数，实现了一个线程安全的采样器。
- scene.hpp: 实现了场景实例的逻辑，包含了天空盒的逻辑。
- shape.hpp: Shape 类。定义了形状的接口。继承自 Object 类。
- shapes/: 实现了若干形状，包括三角面片、球、旋转曲面。均继承自 Shape 类。
- tracers/: Tracer 类。定义了渲染器的接口。
- tracers/bdpt.hpp: 实现了双向光路追踪，使用 fopenmp 加速。继承自 Tracer 类。
- utils.hpp: 常用物件，声明了必要的标准库以及宏。
- vecmath.hpp, vecmath/: 向量计算的定义与实现，也包含了多项式与样条曲线的实现。

以上代码均为独立实现，总计 2940 行。此外，deps/ 下包含了两个用以处理外部文件的开源库：

- stb_image.h, stb_image_write.h: stb 图像库，用于读写图片。
- tiny_obj_loader.h: obj 文件解析器，用于读入模型。

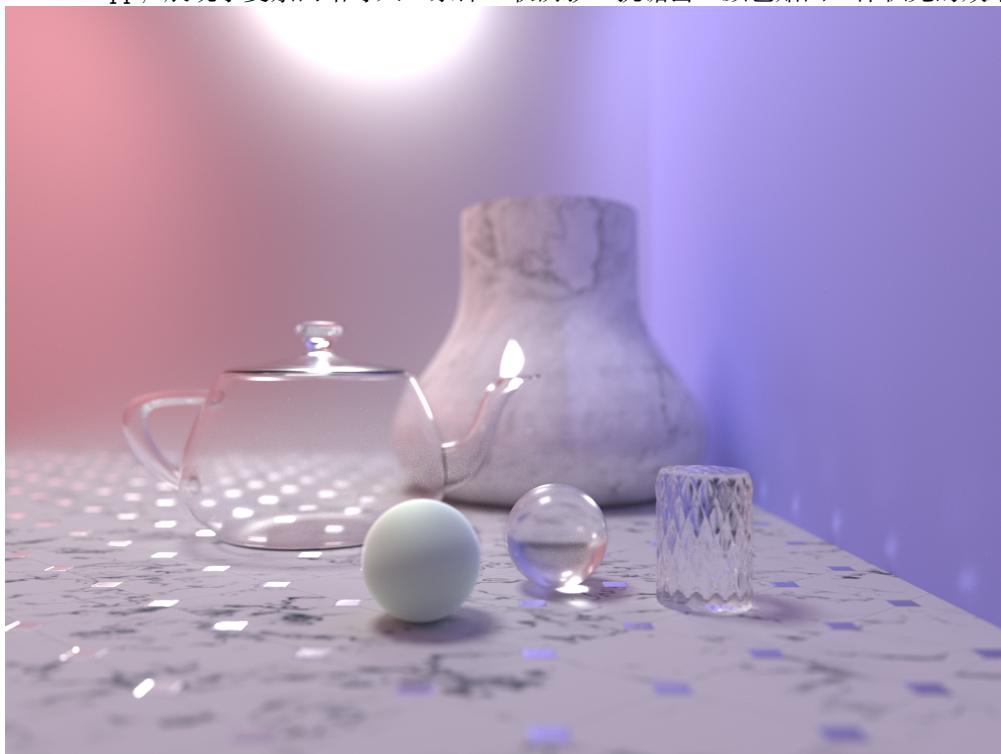
渲染图片时通过引用各种部件构建场景。详见 samples/ 下的各种示例。

2 渲染结果

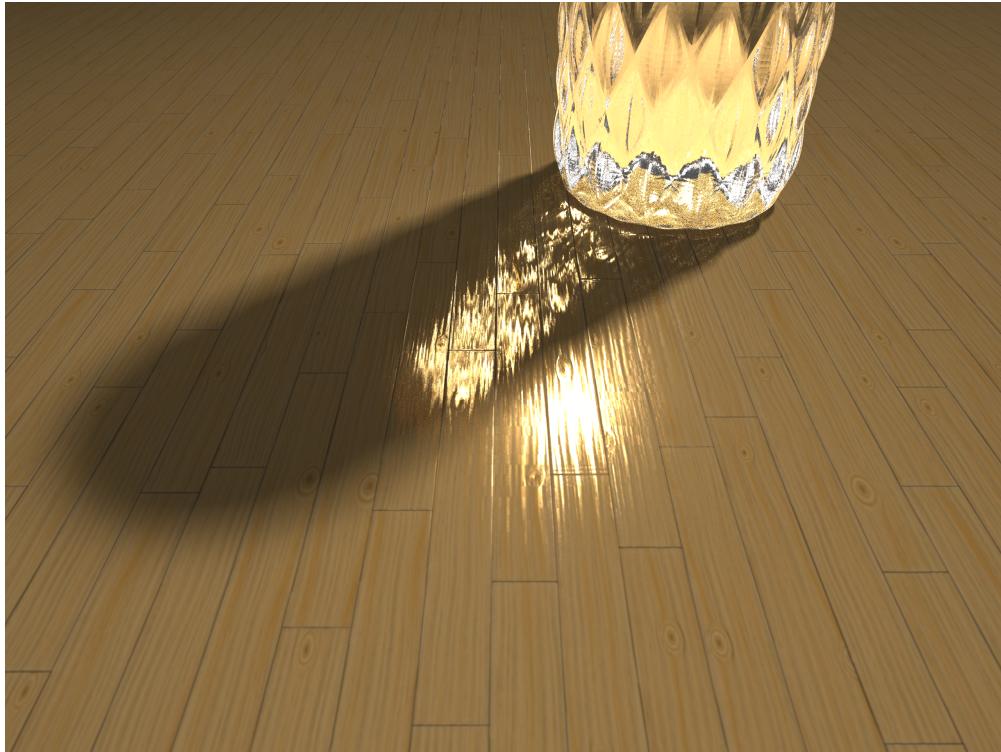
`samples/` 文件夹下包含了调用渲染器渲染具体场景的代码：



见 `classroom.cpp`, 展现了复杂网络导入、景深、软阴影、抗锯齿、颜色贴图、体积光的效果。



见 `corner.cpp`, 展现了参数曲面、景深、软阴影、抗锯齿、颜色贴图、法向贴图的效果。



见 `caustics.cpp`, 展现了焦散、景深、软阴影、抗锯齿、颜色贴图、法向贴图的效果。
以上结果的原图均位于 `results/` 文件夹下。

3 得分点汇总

这里只列举了得分点与对应的文件，具体实现方案与解释请参考后文。

- 算法选型：双向光路追踪见 `tracers/bdpt.hpp`。
- 参数曲面：样条曲线见 `vecmath/spline.hpp`, 基于其的旋转曲面见 `shapes/revolved.hpp`。
- 复杂网络与加速算法：导入模型处理见 `model.hpp`, K-D 树见 `graph.hpp`, OpenMP 见双向光路追踪。
- 景深、软阴影与抗锯齿：均见 `cameras/perspective.hpp`。
- 贴图：U-V 映射见 `shapes/` 下的各个形状，调用贴图见 `texture.hpp`, 颜色贴图见 `bxdfs/mtl.hpp` 与 `bxdfs/mtlExt.hpp`, 法向贴图见 `bxdfs/mtlExt.hpp`。
- 体积光：宏观介质部分见 `mediums/scatter.hpp`, 微观粒子部分见 `particle.hpp`。
- 焦散：由于渲染器基于物理，因此其自然有焦散效果。

4 双向光路追踪

见 `tracers/bdpt.hpp`。

为了能适应复杂场景中光滑场景不同的表面，渲染器使用双向光路追踪作为渲染算法。通过幕启发式方法，渲染器会倾向于按照方差最小的方式连接视点光路与光源光路，计算贡献。相较于普通的光路追踪，其能更为有效的在较为的光滑的表面上进行采样，并能处理从视点出发难以看到光源的情况。相较于渐进式光子映射，其能更为有效的对体积光效果进行渲染。

不仅如此，由于双向光路追踪追踪了从光源出发的路径，其也能轻易的处理焦散。

4.1 实现逻辑

假设连接出的光路段数为 l , 其从相机处追踪了 s 步, 从光源处追踪了 t 步, 则有 $l = s + t + 1$ 。特殊的, $t = 0$ 的光路为在光源上采样, 连线形成的光路, $t = -1$ 的光路为从相机光路一直追踪, 直到遇到光源表面而形成的光路。由于实现中相机大小非常小, 因此渲染器没有按照 $s = -1$ 进行采样。

对于每一次采样, 渲染器会以亮度作为权重, 随机一个光源从其开始追踪一个光子(见 `revTrace`)至一定深度, 记录下光源光子遇到的每一个表面(保存为一个结构体 `Photon`)。之后从相机处随机采样一条光线进行追踪(见 `trace` 函数)。每遇到一个表面, 若其是光源, 则对于这条从相机出发的光路计算贡献(见第 208 至 214 行, 统计 $s = l$ 的光路), 并对每个光源采样一个点连线, 若可见则对该光路贡献(见第 216 至 238 行, 统计 $s = l - 1$ 的光路), 再与每个记录下来的光源光子进行连线, 若可见则两端相连计算贡献(见第 240 至 262 行, 统计 $1 \leq s \leq l - 2$ 的光路)。特殊的, 对于每一个光子, 我们将其与相机直接相连, 若其可以被看见且会被捕获, 则在该像素上计算贡献(见第 104 至 135 行, 统计 $s = 0$ 的光路)。这样便考虑了双向光路追踪中的每一种连接方案。

4.2 幂启发式方法

不同的连接方法事实上给出了不同的采样方法。注意到若是一条光路很难被某种采样方法采样到, 则其以该种方法计算时会有较大的方差, 故为了能有效减少方差, 渲染器使用幂启发式方法减少此类采样的权重。

具体来说, 对于段数为 l 的光路, 渲染器一共有 $l + 1$ 种方法采样出它。对于从相机处追踪了 s 步的采样方法, 其权重 w_s 为采样出这条光路的密度函数(包括从相机采样光线的概率, 光源采样出光源光线的概率, 每个 BSDF 采样出射光线的概率)乘以连接时的系数(呈连接距离的反比)。而若是以这种采样方式采样出了这条光线, 则其的权重为 $w_s^2 / \sum_{i=0}^{l+1} w_i^2$ 。容易发现, 这样合并采样方法是无偏的。

为了避免在统计概率时重新算每次采出每个样的概率, 渲染器使用了前缀和优化以 $O(1)$ 计算权重。(见结构体 `BiPdf`)

5 相机相关

`cameras/perspective.hpp` 实现了带景深的透视相机。 o, x, y, z 四个向量描述了相机的摆放方式。 R 为相机的透镜半径。 f_0 描述为相机的透镜焦距。 f 为相机的聚焦距离。当 $R = 0$ 时, 为小孔成像无景深效果。`sample` 函数实现了给定像素坐标从相机采样的逻辑。`evaluate` 函数实现了给定场景中的点计算其对应相机像素坐标的逻辑。

5.1 抗锯齿

见 `cameras/perspective.hpp` 的 `evaluate` 函数与 `sample` 函数。

相机某像素处的亮度为通过其对应正方形的平均光强。每次采样时随机从中选取一条光线进行追踪以实现边缘抗锯齿。

5.2 景深

见 `cameras/perspective.hpp` 的 `evaluate` 函数与 `sample` 函数。

由成像规律, 距离为 z 的点会成像在 $-\frac{1}{1/f_0 - 1/z}$ 处。在透镜平面上采样作为镜头外光线的终点, 连接其与像点构成相机内的光路。通过在相机感光平面 $-\frac{1}{1/f_0 - 1/f}$ 计算光子贡献, 以物理的实现景深效果。

6 物体相关

通过在 `Shape` 实例上绑定 `BxDF` 实例, 便可以得到一个物体。其中, `intersect` 函数定义了物体与光线求交的逻辑, `normal` 函数定义了物体表面的宏观法向量方向, `tangent` 函数定义了物体表面纹理贴图的 x

轴方向, `sampleBxDF` 函数定义了给定入射光后对出射光进行采样并计算贡献的逻辑, `evaluateBxDf` 函数定义了入射光与出射光之间的贡献, `outline` 函数定义了用于加速的物体包围盒。

6.1 法向插值

见 `shapes/triangle.hpp` 的 `normal` 函数。

三角面片提供了法向插值功能, 在定义了顶点处的法向量后, 面片内的法向量会通过插值的方式计算。以此平滑导入模型的表面。

6.2 纹理映射

见 `shape.hpp` 的 `intersect` 函数与 `texture.hpp`, `bxdfs/mtl.hpp` 与 `bxdfs/mtlExt.hpp`。

光线与物体求交时, 不仅会返回交点的三维场景坐标 `pos`, 还会返回表面的二维纹理坐标 `texPos`。二维纹理坐标将三维物体表面映射到了平面上, 以方便实现贴图。支持此功能的双向散射函数时会访问材质图片用作表面的亮度。

注意: 出于性能上的考量, 仅有 `bxdfs/mtl.hpp` 与 `bxdfs/mtlExt.hpp` 两个较慢的双向散射函数支持了纹理贴图。

6.3 法向贴图

见 `shape.hpp` 的 `intersect` 函数与 `texture.hpp`, `bxdfs/mtlExt.hpp`。

基于微表面模型的双向散射函数定义了物体表面的法向量分布。支持此功能的双向散射函数会在使用法向量分布时根据法向贴图的信息旋转法向量。

注意: 出于性能上的考量, 仅有 `bxdfs/mtlExt.hpp` 这一个最慢的双向散射函数支持了纹理贴图。

6.4 旋转参数曲面

见 `shapes/revolved.hpp` 的 `intersect` 函数与 `normal` 函数。

支持参数曲面仅需要实现直线与旋转参数曲面的求交, 以及法向量计算。旋转参数曲面的纹理坐标将表面映射为对应的参数, 故法向量可以直接通过参数曲面的导数求出。对于直线与旋转参数曲面的求交, 渲染器使用了 G-N 迭代法, 将参数对应物体表面点与直线的距离作为目标, 对参数进行优化。通过选取多个参数作为初始值尝试迭代, 算法便能有效的找到第一个零点。

7 光源相关

通过在 `Shape` 实例上绑定 `Light` 实例, 便可以得到一个光源。其中, `intersect` 函数定义了光源与光线求交的逻辑, `normal` 函数定义了光源表面的宏观法向量方向, `tangent` 函数定义了光源表面纹理贴图的 `x` 轴方向, `sampleSurface` 函数定义了在光源表面采样的逻辑, `sampleLight` 函数定义了采样出射光并计算光强的逻辑, `evaluateBxDf` 函数定义了给定出射光计算对应光强的逻辑, `outline` 函数定义了用于加速的物体包围盒。

7.1 软阴影

使用面光源后, 所有的阴影都自然是软阴影。

8 介质相关

`Medium` 类定义了介质的各种行为。其中, `evaluate` 函数定义了光子在通过介质若干距离时的光强变化, `sample` 函数定义了光子在通过介质若干距离后遇到粒子的概率。光子在均匀介质中不会遇到任何粒子。

8.1 体积光

见 `mediums/scatter.hpp` 的 `sample` 函数与 `particle.hpp`。

而当光子在散射介质中穿行时，每单位距离会以一定距离遇到粒子（遇到粒子的距离呈指数分布）。若光子在遇到宏观物体时便以撞到粒子，光子会在该点调用粒子实例的散布函数计算对应的出射方向。

9 加速方法

9.1 K-D 树

见 `graph.hpp`。

为了加速光线与场景的交点计算，渲染器使用 K-D 树进行加速。复杂模型会被分解为三角面片通过 K-D 树加以维护。`Shape` 类的每一个物体都需要定义包围盒 `box` 的计算方法，其应为包含它的最小的棱平行于坐标轴的长方体。基于包围盒，`load` 函数实现了启发式建立 K-D 树。具体来说，枚举被切割的坐标轴，排序并枚举作为根的物体，以分离后两侧的包围盒的体积之和作为该分割方案的代价，选取代价最小的方案进行分割，并在两侧递归调用 `load` 函数。`intersect` 函数实现了询问光线与 K-D 树中物体的最近交点：首先判断光线与包围盒是否相交，且交点是否近于之前求出的交点，若相交则检查光线与根处物体的交点，并递归调用两孩子。值得注意的是，`intersect` 函数会尝试先递归包围盒与光线交点近的孩子。通过这样的方法，实践上，对于 10^5 个面片左右的模型，平均每次询问只需要访问 20 个左右的节点便能找到最近交点。

9.2 OpenMP

见 `tracers/bdpt.hpp` 第 74 至 75 行。

渲染器通过 OpenMP 实现了多线程渲染，对于每行的采样作为一个调度单位进行分配。通过隔离每次采样使用的资源（包括使用 `erand`，分离光源采样时的光子记录），最大化的减少了线程之间的内存冲突。两个线程仅有在修改结果数组时需要加锁。在第一张结果图上，多线程比起单线程在 64 核的机器上速度快了大约 31 倍（从 219 秒优化到了 7 秒）。