计算机图形学

何中天

January 2021

目录

1 成果图

成果图的详细内容在最后一节场景设计中详细介绍。

2 算法选型

我实现了光线追踪和渐进式光子映射两种渲染算法，均使用 kd 树优化。同时实现了景深、贴图、软阴影和抗锯齿多种效果。

2.1 光线追踪

由渲染方程，每次在光线与表面的交点处都要做一次积分。光线追踪算法，使用蒙特卡洛方法来近似这个积分。在每个交点，依据当前表面的 BRDF 函数来进行反射与折射的采样，

$$L_r = \int_\Omega f(\omega_i \to \omega_r) L(\omega_i) \cos(\omega_i) d\omega_i$$

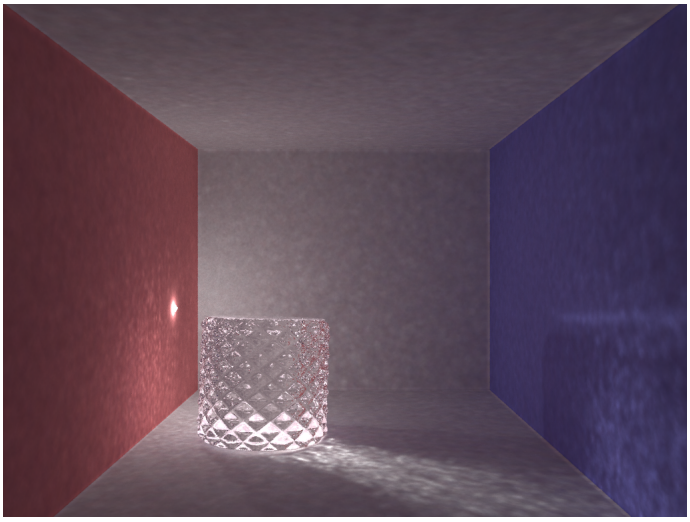
在学习光线追踪的过程中，我参考了smallpt。

2.2 渐进式光子映射

光子映射算法从光源出发追踪光子的路径，使用俄罗斯轮盘赌来决定光子在每个漫反射面是否被吸收，并记录它经过的漫反射面。通过随机大量的光子，我们就可以得到光子在漫反射面上的光子图，由于光子数量大，我们使用 kd 树来存储。在渲染时，从相机发射射线并追踪，碰到第一个漫反射面则停止，这时在交点处找光子图中 k 临近光子，同时设定一个最大半径 R。使用这些光子的能量来渲染对应点。

渐进式光子映射算法通过多轮迭代，逐渐减小 R 的大小来获得更高的精度。

与光线追踪算法相比，光子映射能够更好的处理由反射和透射引起的漫反射。一个典型的例子是 Caustic，我渲染了下图的玻璃杯来展示这个特性。

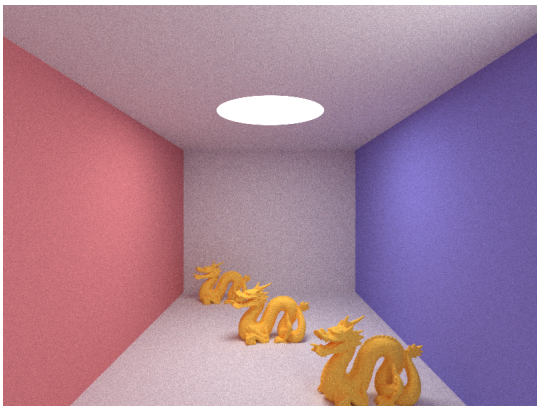


3 算法优化

在光线追踪和渐进式光子映射中我均使用了 kd 树进行优化。对于光线追踪的部分，主要的开销来源于求交的判断，对三角面片建 kd 树可以得到显著的优化，这部分代码实现在 `mesh.cpp` 中。在渐进式光子映射时，光子数量是百万级别的，使用 kd 树可以有效优化求解 k 近邻的时间。kd 树求 knn 的核心代码如下：

```
    numberstyle
1  float kdtDis(int x) {
2      Vector3f dis;
3      for (int i = 0; i < 3; i ++) {
4          if (kdt[x].mn[i] <= centr[i] && centr[i] <= kdt[x].
               mx[i]) dis[i] = 0;
5          else dis[i] = min(fabs(kdt[x].mn[i] - centr[i]),
               fabs(kdt[x].mx[i] - centr[i]));
6      }
7      return dis.length();
8  }
9
10 void kdtQry(int x, std::priority_queue< Photon, vector<
       Photon>, cmpKmin >* que) {
11     if (kdtDis(x) >= DisToCentr(tp)) return ;
12     if (que->size() < K || DisToCentr(pm[x]) <= DisToCentr(
           tp)) {
13         if (que->size() >= K) {
14             que->pop();
15         }
16         que->push(pm[x]);
17         tp = que->top();
18     }
19     int lc = kdt[x].lc, rc = kdt[x].rc;
20     if (lc && rc) {
21         if (kdtDis(lc) < kdtDis(rc)) {
22             kdtQry(kdt[x].lc, que);
23             kdtQry(kdt[x].rc, que);
24         } else {
25             kdtQry(kdt[x].rc, que);
26             kdtQry(kdt[x].lc, que);
27         }
28     } else if (lc) kdtQry(kdt[x].lc, que);
29     else if (rc) kdtQry(kdt[x].rc, que);
30 }
```

同时也使用了 OpenMP 利用多线程加速。



4 附加特性

4.1 景深

景深的光学效果来自于相机凸透镜的聚焦。首先求出相机射线与焦平面的交点，在凸透镜上蒙特卡洛采样并发出新的射线即可得到景深效果。

4.2 贴图

建立交点与图片坐标的对应关系，将交点对应到图片的像素点上。在我的成果图中，背景球场和天空都是使用贴图来实现的。

4.3 抗锯齿

计算某个像素的值时，通过扰动可以获得抗锯齿的效果，我使用了 tent 滤波来进行扰动的改良, 使得分布向中间集中。

$$f(x) = \begin{cases} \sqrt{x} - 1, x < 1 \\ 1 - \sqrt{2 - x}, x > 1 \end{cases}$$

4.4 软阴影

软阴影的原理是对面光源的随机采样，我的算法实现了这个效果。

5 代码框架

我的代码框架是基于 PA1 的。主要渲染算法包括：在 `main.cpp` 中，`PathTracing()` 和 `radiance()` 实现了光线追踪。`PPM()`, PhotonMapping(), RayTracing() 和 collectPhoton() 实现了 PPM，其中 `struct Kdtree` 是对光子图建立的 kd 树。

在 `mesh.cpp` 和 `mesh.hpp` 中，对三角面片建立了 kd 树，求交由原来的暴力改为在 kd 树上查询，这个 kd 树是对三角面片而不是点建立，所以与上一个结构不同，不能复用。

6 场景设计

成果图中展示的场景由我自己搭建，以足球为主题。正中是英超足球俱乐部阿森纳的 logo，使用的是折射效果，地面上的红色光晕就是对光源的折射。两边摆放了两座欧冠奖杯（其实阿森纳没有得过欧冠，但是网上免费的足球奖杯模型只有欧冠了，那就祝枪手早日捧起欧冠奖杯吧）。欧冠奖杯的表面采用金属高光效果。前面摆放着一个足球和一双金靴（枪手拿过 12 次英超顶级联赛金靴，位居第一，其中传奇亨利拿过四次）。金靴采用反射面。背景贴图是枪手主场——酋长球场，天空自然是北伦敦的天空了。地面采用 Phong 材质，光照效果是漫反射与反射的叠加。图中有四个点光源，位置使用小白点标出。具体参见 arsenal.txt。

成果图分辨率 1024×768, 采用 PPM 算法进行渲染，图中包含多种折射、反射模型及不同的材质，我选出了效果最好的材质组合，比如如下两图的对比。

图 1: 地面 Phong 材质

图 2: 地面不带反射