基于 Bézier 曲线的三维造型与渲染

2015011308 计 53 唐适之

概要

本程序实现了三次 Bézier 曲线生成的旋转曲面的造型与渲染，渲染算法是光子映射，并附加有纹理贴图和景深效果。此外，使用了一些数据结构优化了性能，以及通过随机化视线发射方向实现了一定程度的反锯齿。

效果展示

场景的三角网格表示

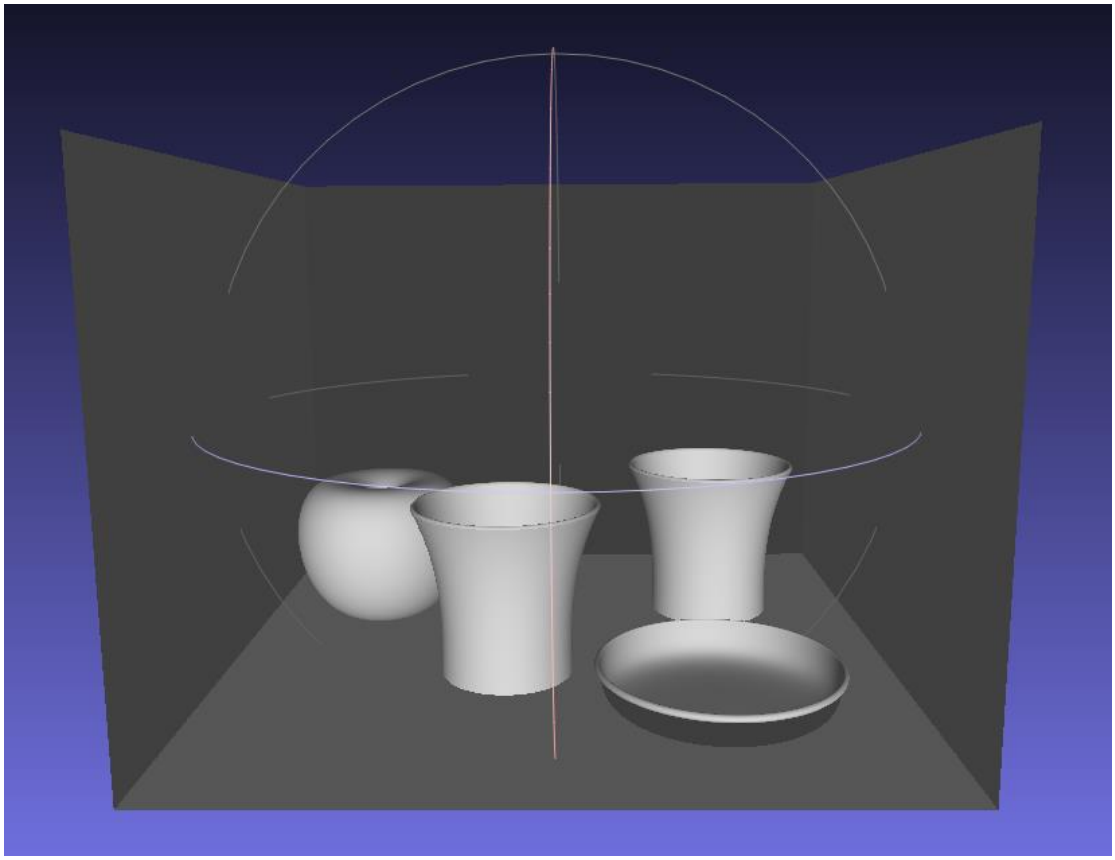

fig. 1 输出到.obj 文件并用 MeshLab 打开的截图

最终渲染结果

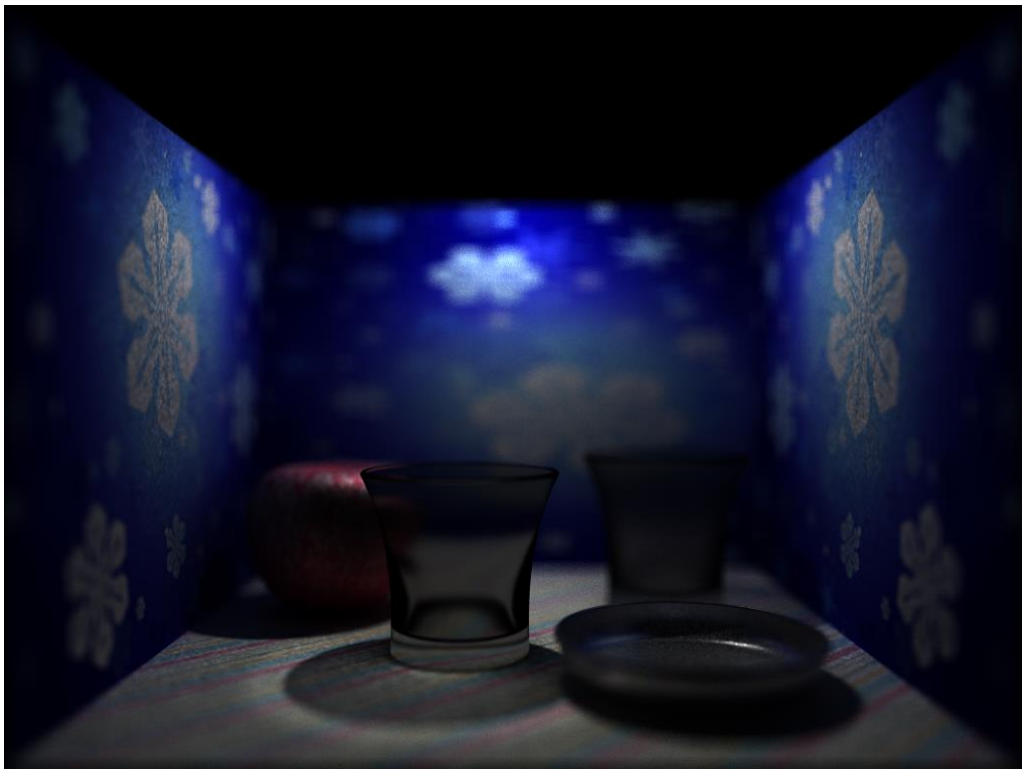

fig. 2 渲染结果（光圈=f/9）

此图带有景深效果，为了看清局部细节，又渲染了一张无景深（设光圈=0，等价于小孔成像）的输出如下：

fig. 3 渲染结果（无景深）

图中含有如下效果：

1. 从右侧的碟子可以看出镜面反射效果，包括反射像和高光；
2. 从较靠近视点的玻璃杯底部可以看出折射成像；
3. 从玻璃杯的阴影中可以看出光子映射做出的焦散效果；
4. 两个玻璃杯的漫反射分量和 Phong 模型高光项的指数不同，分别做出了磨砂玻璃和相对细腻的玻璃的效果；
5. 三面墙壁是纹理贴于平面的效果，苹果是纹理贴于 Bézier 曲线旋转面的效果。

绘制算法

算法首先要解决光线/视线与曲面求交的问题。曲面表示为参数形式$\mathbf{S}(\mathbf{u},\mathbf{v})$，对于每个曲面，将其按 u,v 分为 25*25 共 625 个小格，对于每个小格求出其三维包围盒。求交时，先在树上找出离光源/视点最近的与光线/视线相交的包围盒，以此包围盒作为交点初值，用曲面的导数进行牛顿迭代，迭代 5 次后求出精确解或确认无解。如果以最近的包围盒作为初值是无解的，还应尝试次近的、第三近的等。

具体实现中，Surface 类（Surface 类位于同名的 surface.h、surface.cpp 文件中，其子类位于 surface 文件夹中，以下类似）提供了各类曲面的基类接口，子类分别实现了求 Bézier 曲线旋转面和平面等曲面的坐标、导数等的算法。其中 Surface/Axisymmetric 类实现了对任意曲线构造旋转面的算法，Curve/Bezier3 类实现了 Bézier 曲线的相关算法。求交算法实现在 intersection.cpp 中。

其次是光子映射。本程序中，从光源发射10^7个光子，每个光子迭代 6 次；由视点向每个像素（1024*768）分别发射 250 条视线，每条视线迭代 6 次。

为了避免场景中出现完全漫反射体或完全镜面反射体而导致不够真实，每个物体都可以兼有漫反射、镜面反射、漫透射、镜面透射（折射）的性质，按权重表示各分量大小。光线/视线与曲面交汇后，以这四种分量各自的权重为概率，决定反射/折射的模式。对于每一种模式，均以 Phong 模型的权重作为概率密度函数，以相应分布向各方向随机发射下一次迭代的光子/视线。包括镜面反射/透射也是以 Phong 模型的权重为概率密度的，这意味着可以通过 Phong 模型高光项的指数调整镜面反射/透射光的聚集程度，即反射/透射面的光滑程度，例如上图中的磨砂玻璃和较细腻玻璃的对比。实践表明，这一指数要设得很大，例如几百时，物体才会显得比较光滑。

视线在每次迭代时，需要找出与其“交汇”的光子。具体地，即视线所交平面上距其最近的 100 个光子，对于每个光子，关于此交汇处所在的曲面再算一次 Phong 模型的权重，以光子的总亮度除以光子分布面积的大小作为光子的平均亮度，再与视线所剩亮度相乘作为视点可见的亮度。对于每条视线，取各次迭代亮度的均值作为最终亮度。

由于场景是有颜色的，除上述亮度外，光子和视线还记录了各颜色分量所占比例。最终各分量的亮度由总亮度与该分量比例相乘而得。

光子映射的控制流程实现在 main.cpp 中，具体追踪过程实现在 trace.cpp 中。

其他功能

纹理贴图

此程序中纹理是以图片形式输入的。因为所有曲面或平面都已有参数表示$S(u,v)$，直接将(u, v)线性映射到图片坐标即可。

具体实现中，纹理是作为“材质”的一部分定义的。“材质”还包括物体的各类反射率、透射率等参数。Material 类负责指定纹理，Texture 类负责读入纹理图。

性能优化

1. 如上文所述，求交时所用的初始解来自于距光源/视点最近（或第二近、第三近等）的包围盒。为高效地找到此包围盒，程序中将每个平面的所有包围盒分别组织成二叉树（实践证明二叉树效率比四叉树高）。每个节点按 u 或 v 均分生成两个子节点，并求出子树两个包围盒构成的总包围盒。事实上是先按 u 分还是先按 v 分是对性能有影响的。为了提高树的性能，应该让最终不与任何包围盒交的光线/视线尽早被判断为无解，那么由子树两个包围盒构成的总包围盒的体积应该尽量小。按这种思路，程序会尝试按 u 分或按 v 分两种分法，选择总包围盒体积较小的一种作为最终切分方法。此项优化请参见 BoxTree 类。
2. 为了找到与视线“交汇”，即与视线和平面的交点最近的 100 个的光子，程序中将每个平面上的光子组织成 k-d 树。若总共要投射 n 个光子，可在$O(100\ n^{\frac{2}{3}})$的时间内查询最近的 100 个光子。请参见 KDTree 类。
3. 多线程。本程序通过 OpenMP 实现多线程，使不同光子的追踪和不同视线的追踪均可以在各自线程中并行处理。具体实现见 main.cpp 的 emit 和 collect 函数。为了做到线程安全，各线程的随机数发生器也独立运行。实践中，每个 CPU 核心都能保持满负荷运行，证明各线程间不存在过度竞争。

景深

为了渲染出景深效果，视线是通过凸透镜成像的方式投射的。即，对于每个像素，在凸透镜后有多种可能的出射路径。每个像素投射的 250 根视线均在这些可能路径中随机选择。通过调整焦距和光圈即可调整视场深度及景深对比度。具体代码见 main.cpp:84~89。

反锯齿

光子映射是一个耗时较长的算法，而一般的超采样抗锯齿需要成倍地消耗时间，故不可取。但本程序中，对于每个像素点本来就要发射 250 根视线。这 250 根视线并不一定需要沿该像素中点射出，令其在对应像素的范围内随机选取方向，有效地缓解了锯齿问题。具体实现见 main.cpp:84~85。

附

关于代码具体组成的说明，可参见位于 doc/doxygen/html/index.html 的由 Doxygen 生成的文档。