真实感图形渲染

2015011333 计 54 秦岳

一、 代码设计框架

效果图和实验数据过程可见于文件夹中。

代码中除了使用 OpenCV 的 cvLoad 和 cvSave 读取和保存图片获取像素、使用 Egin 外，其他全部的代码均为独立实现(如 Vector3，Color 等)，共计代码 7586 行。

Bezier.h：提供 Bezier 曲线的一系列功能，如求点坐标，梯度等

BMP.hpp：图片数据存储类，支持读取图片像素颜色，双二次线性插值，拉普拉斯差分，球坐标映射等功能，主要用于各类贴图存储。

Camera.hpp：摄像机类，记录坐标张角等参数。提供投影坐标，产生视线和 Camera 特效(景深、抗锯齿)

Color.hpp：颜色类，重载+-之类的功能，支持 HSV->RGB 色彩空间转化。

Geometry.hpp：基本几何形状(在后期被废弃)

Matrix4.hpp：坐标变换矩阵，支持矩阵\向量乘，矩阵构造(绕轴旋转矩阵，位移矩阵等)

Mesh.hpp：网格物体类，以 Obj 格式存储的基本网格信息(基本功能已转移到 MeshObject 类)

Modeler.hpp：建模类：提供了大量物体和材质的构造方法(如 Bezier 曲面、玻璃材质、Obj 格式的茶壶龙等)

MousePaint.hpp：鼠标绘图类(从网上找的一个 OpenCV 绘图与交互代码、用于调试)

Object.h：所有的渲染物体信息。定义了材质、光线、碰撞点等基本结构。通过 Object 基类定义信息完全的物体(包含材质等一切渲染需要的信息)，如球体、长方体、点物体、圆环柱体、无穷大平面、三角面片、网格物体、Bezier 曲面体、物体组等。定义了光源基类、点光源、锥形光源、长方形光源、纹理光源等。并提供上述物体的一些函数如与光线求教，返回包围盒，返回光源光线集合等。

ObjParser.hpp：简易的 obj 格式解析器

RayTracing.hpp：朴素光线追踪渲染器

PathTracing.hpp：路径追踪渲染器

PhotonMapping.hpp：光子映射渲染器

ProgressivePhotonMapping.hpp：(随机)渐进式光子映射渲染器

Scene.hpp：场景类，包含场景必要的信息如环境光，摄像机，全部物体，全部光源等。包含测试过程中全部场景的生成过程。

Vector2.h/Vector3.h：向量类，提供向量基本功能

VolumetricLighting.hpp：体积光渲染插件

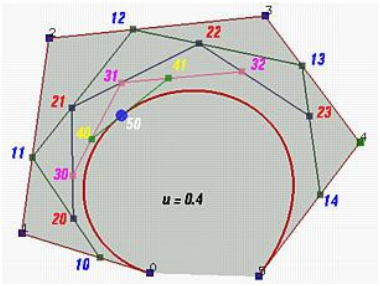
二、 Bézier 曲线造型与曲面建模

1. Bézier 曲线造型

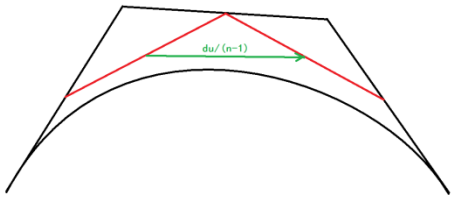
相关代码位于 Bezier.h(BezierCurve 类)

高次 Bézier 参数曲线由 n 个控制点构成，参数 t(t∈[0,1]。Bézier 曲线需要解决两个问题：①求参数 t 对应点的坐标②求参数 t 对应点的导数。Bezier 控制点可以为三维坐标以便构造复杂空间曲线。

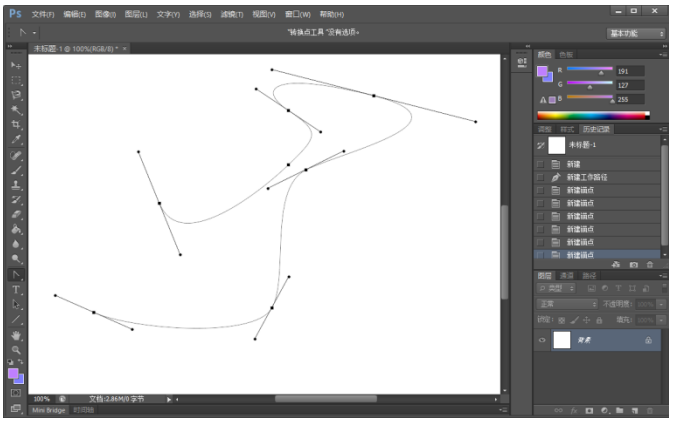
求 t 坐标：曲线上点的坐标可由德卡斯特里奥算法迭代计算，原理如下：每次将现有的线段的 t 比例的点取出来顺次连接构成新的 n-1 个线段，重复上述过程直到剩下 1 个点位置。(BezierCurve::Pos)

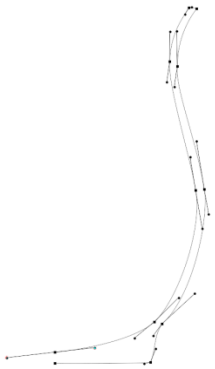


求 t 导数：将 Bézier 基函数做适当变形，可推导出 t 处的微分向量即为德卡斯特里奥算法最后一次迭代中，两个点所构成的向量的(n-1)倍即为 dt。(BezierCurve::Derivative)



构造漂亮的高次 Bézier 曲线较为困难，所以我在建模过程中主要使用分段 3 次 Bézier 曲线建模，常用的绘图工具(如 PhotoShop 的钢笔工具)均可支持，建模后通过画图工具读取控制点坐标并录入成文本。(建模代码位于 Modeler.hpp)





对于 Bézier 曲线类，附加了两个函数功能便于加速计算和后续工作：

1、 计算 Bézier 曲线长度 BezierCurve::Length（用于合理的细分网格），根据参数分段，每段用线段近似。

2、 曲线包围盒 BezierCurve::BoundingBox，计算曲线 x,y 坐标的最大值和最小值。以 x 维为例，先均分采样点，目标为函数最值。根据采样点的凸性找出极值存在区间，进行三分法求区间极值(也可以使用牛顿迭代)。

2. Bézier 曲面建模

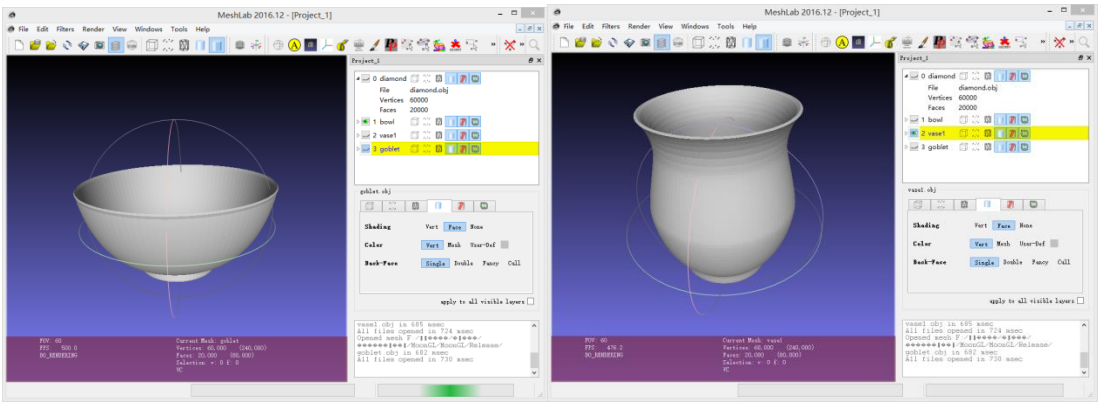
Bezier 曲面有两种建模方法：①用矩阵控制点描述曲面②用曲线绕轴旋转构成曲面

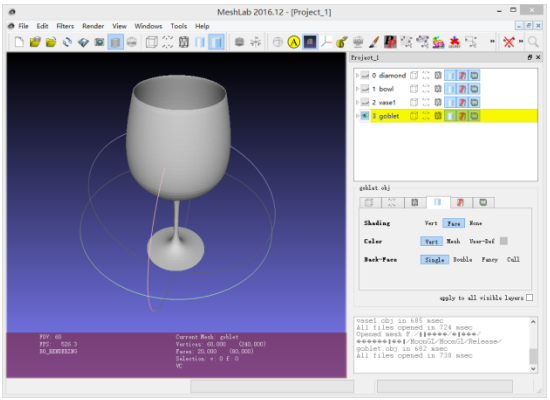
两种方法大体相同，均需提供计算参数(u,v)处的坐标和偏导数向量。

方法①可先根据参数 u 计算每一行控制点的 u 坐标构成一维控制点，再在一维控制点处计算 v 参数坐标(依然可由德卡斯特里奥算法计算)，偏导数即为德卡斯特里奥算法最后一步的向量 n-1 倍。

在代码中主要实现分段 Bezier 旋转体曲面(Object.hpp RotateBezierObject 类)

u,v 两参数描述曲面，u 为曲线方向，v 为旋转。根据计算几何方法计算坐标和法向，重点说明射线求教的计算方法：



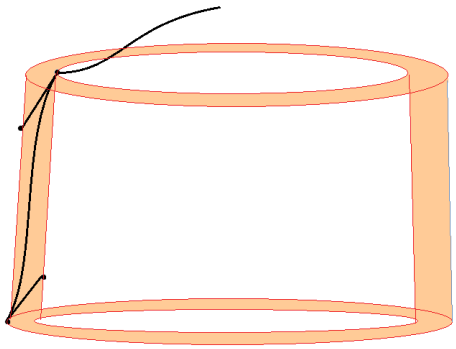


光线参数曲线为 L(t)，曲面为 B(u,v)，即为求三元非线性方程组的零点 F(t,u,v)=L(t)-B(u,v)，采用牛顿迭代法即可。对于 dt 即为光线方向单位长度，du 即为 Bezier 曲线导数根据 v 参数旋转，dv 即为切向乘 2πr。

迭代代码位于(Object.hpp 550 行左右)。

几点改进：

① 对于每一段旋转曲面用圆环柱体包围盒加速，若无交点则不必计算



② 由于牛顿迭代对初值极为敏感，一种方法是对 u,v 均匀划分 4×4，t 初值选取距离 B(u,v)最近的点。多次迭代取最近。

③ 迭代剪枝，由于牛顿迭代的局部收敛性，若 u,v 偏离[0,1]太大，或 t<0 太多则直接结束，或者迭代 10 次后结束(偏离参数选为 0.3)。

④ 启发式选取更有效的初值，均分 u 选取 v 平面与 t 直线的交点前后。参数对比如下：

实验对比(线段旋转构造了一个圆形曲面)：

左图为随机一个点进行迭代后的收敛域；右图为启发式选取一个点迭代的收敛域

事实证明启发式选取起点有着超强的收敛效果，对于简单曲面几乎一个点即可全部收敛。

三次曲面旋转体：

高次曲面旋转体(未抗锯齿)：

三、 真实感场景渲染

一、计算几何与三维物体求交

本节全部代码位于 Object.h 与 Object.cpp

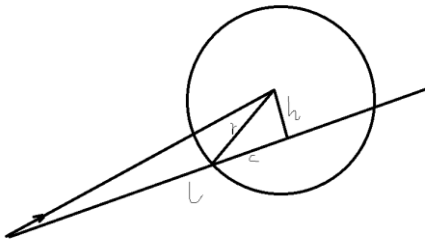
核心问题为射线与各种形态的几何物体求交，并计算出 交点坐标、几何法向量、纹理坐标。(交点信息类 Object.hpp：CollidePointData)，实现中所有物体均继承 Object 基类，基类为抽象类包含求交和返回包围盒两个纯虚函数。

1、 长方体(BoundingBox,CubeObject)

对于边与 xyz 轴平行的长方体可由区间法计算。例如射线起点为 pos，方向为 forward，参数方程 X(t)= pos+t*forward。以 x 方向为例，交区域的必要条件为 x 坐标位于长方体区间[ax,bx]即可计算出 t 的区间[x1,x2]，同理可得[y1,y2][z1,z2]，三个区间的交集即为光线与长方体相交的区域参数 t。由于射线只能向前，故要求 t>0。

2、 球体(SphereObject)

Forward 与圆心 O – pos 点积可算的 l，根据半径 r 可由勾股定理算出 h，判定是否相交，并由勾股定理算出 c，即可得到两个交点 l-c,l+c

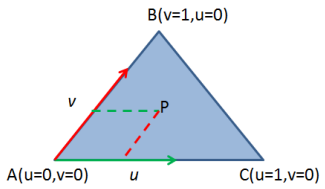


3、 无穷大平面(PlaneObject)

由点积计算点到平面距离，根据法向投影相除即可算出 t。

4、 三角面片(TriangleObject)

建立局部坐标系， $(1-u-v)V_0+uV_1+vV_2$



于是，求射线与三角形的交点也就变成了解下面这个方程-其中 t,u,v 是未知数，其他都是已知的 $O+Dt=(1-u-v)V_0+uV_1+vV_2$

根据克莱姆法则，可得 $\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{[D\times E_2 \bullet E_1]} \begin{bmatrix} T\times E_1 \bullet E_2 \\ D\times E_2 \bullet T \\ T\times E_1 \bullet D \end{bmatrix}$，E1=V1-V0，E2=V2-V0

5、 三角网格(MeshObject)

即为射线与每个三角面片求交取最近交点(KD-Tree 优化)

6、 物体组(GroupObject)

用 Vector<Object*>记录组中的所有物体，射线与组中每个 Object 分别求交，取最近交点。通过 GroupObject 即可实现物体的层级关系、组合 Transform 与集体 KD-Tree 优化。例如一个 Group 包含若干个 MeshObject 或其他 Group，将会形成 KD-Tree 的树套树。

仿射变换与射线求交：

除了对物体定义参数以外(例如球体圆心、长方体顶点)，通过 Matrix4 的空间变换可以实现任意角度物体的求交，被广泛用于层次结构物体中。例如通过绑定 Matrix4，即可实现倾斜的长方体，不同角度的 Bezier 旋转曲面，以及 GroupObject 的相对位置与层级变换。方法为求交之前先对射线的坐标，法向进行 Matrix4 变换(物体不用变)，即可在物体的相对坐标系下进行求交，求取交点后变换回原空间，还可进行组合物体树的递归。

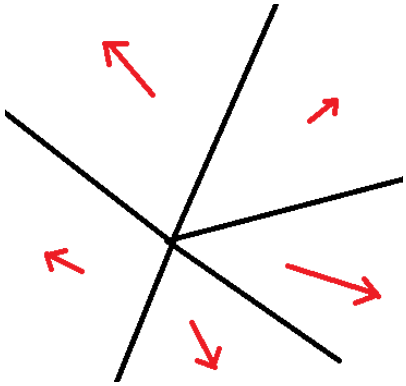
KD-Tree 加速物体求交(代码位于 Object.h，KD-Tree 中的物体全部为基类 Object*，支持 Collide 交点和 BoundingBox 返回包围盒)：

通过空间划分树将物体划分为树状结构，每个子树记录一个包围盒(BoundingBox)，求交时维护当前已算出的交点，由近到远先与盒求交，无交点或与盒的交点远于已有交点即可直接剪枝；启发式的递归进入近的一侧。

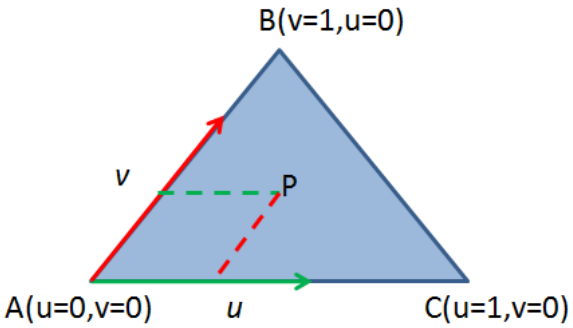
二、三角网格建模与渲染

通过导入 obj 网格来获得 MeshObject 获取空间坐标，法线方向和纹理坐标。

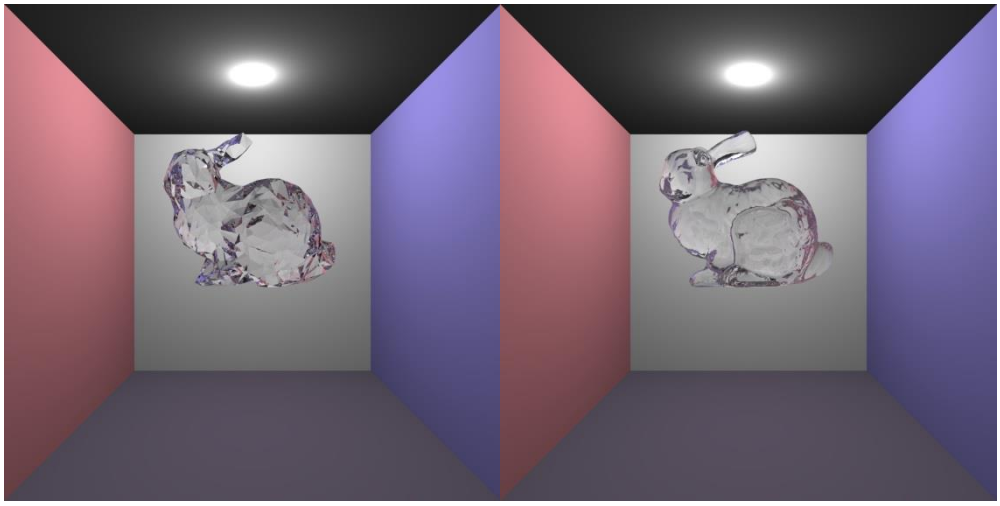
若 obj 模型未提供法向，实现对顶点法线进行可选的插值。常见的插值方法分为均值法、面积法、角度加权法等。代码中使用的均值法确定顶点法向：对于相邻的三角面片可由三个点大致确定面法向，取均值即为顶点法向。



法向插值：对于射线与三角面片的交点，根据重心坐标或局部坐标根据顶点法向插值。



未经过法线插值的低模斯坦福兔子；经过法线插值的高模斯坦福兔子



三、 基于物理的渲染与基本光线追踪(RayTracing)

本节全部代码位于 RayTracing.hpp

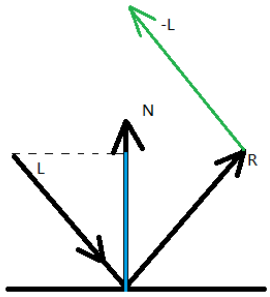
几个重要问题为光线在空间中的变化、光子的流动与光路，主要解决一些光的基本问题：

1、 基本漫反射

无差别的向法线半球方向均匀出射

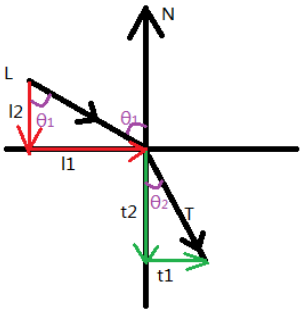
2、 基本镜面反射

光线经过镜面反射产生出射线方向为 R=L-(2*L • N)*N，根据材质参数可以改变出射的颜色比例(能量)。



3、 基本折射

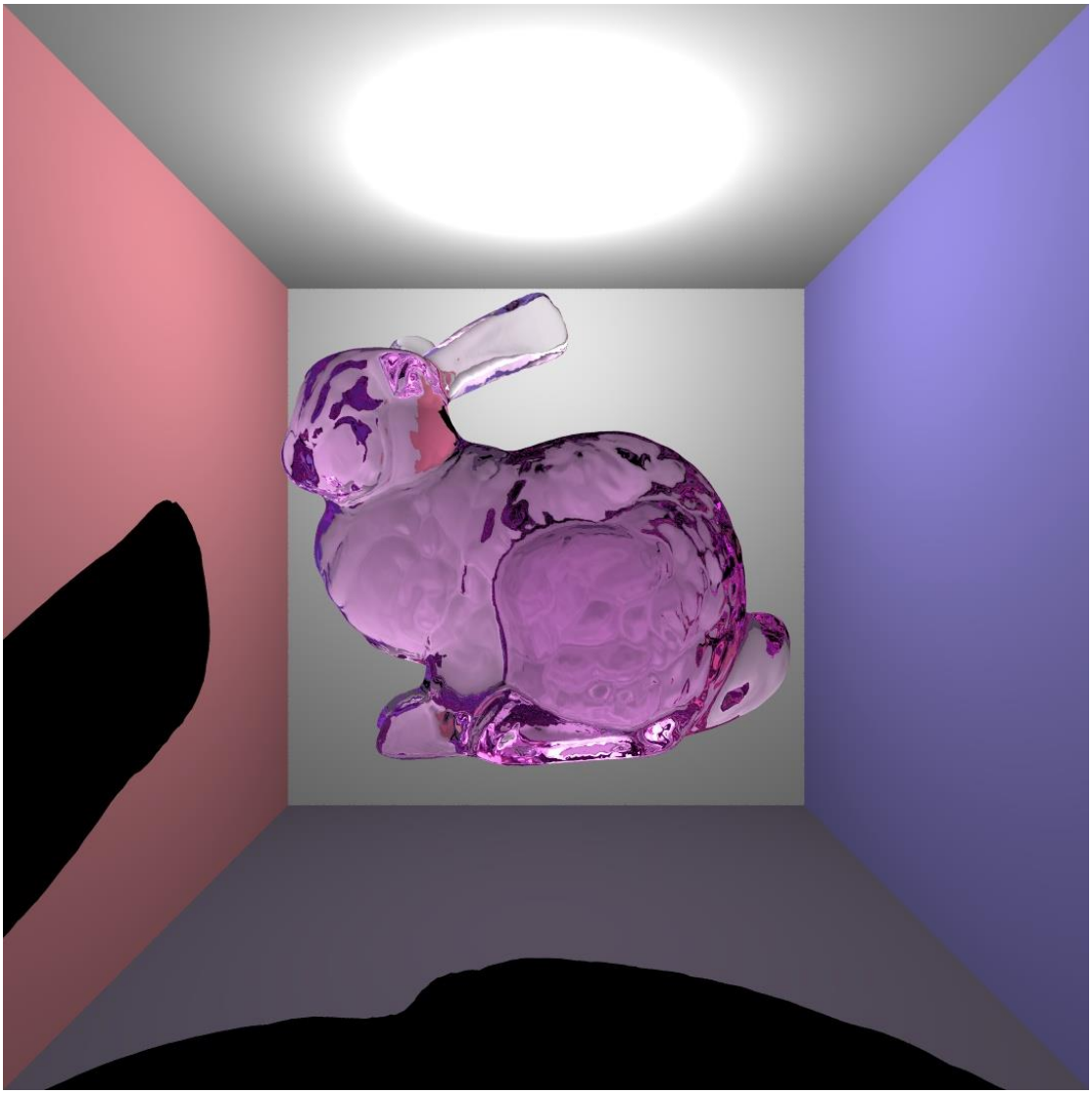
根据折射定律 $\sin\theta1/\sin\theta2=\eta2/\eta1$，可得 $\cos\theta1=-N\cdot L$，$\cos\theta2=\sqrt{1-(1/\eta^2)(1-\cos^2\theta1)}$，$L/\eta+((\cos\theta1)/\eta-\cos\theta2)*N$，根据材质参数可以控制折射光能比例。



朗伯比尔定律：

光穿过物体一部分能量会被吸收，表现为材质的光谱吸收率，单位长度颜色变化的比例。以

绿光为例，设 green 的吸收率为 l，则经过长度 d 后光线的绿光颜色分量将会变为原来的 l^d。



可以发现穿过耳朵的光线颜色变化远小于穿过身体：

4、 镜面漫反射(TODO)

在光线经过物体表面反射时并非完全漫反射或镜面反射两种，根据材质的反射分布还可产生各向同性和各向异性的镜面漫反射，基本解决方法为重复多次采样即可模拟出镜面漫反射效果

5、 菲涅耳效应

一些材质(比如玻璃、瓷器、水面)的折射光比例和反射光比例会随入射角度剧烈改变，所以

加入菲涅效应的材质渲染可以更加逼真的模拟材质的物理特性，并且避免全反射带来的突变边缘，特别是玻璃材质侧面的反射高光，全反射交界处的渐变。

$$k_r = \frac{1}{2}(r_{\parallel}^2 + r_{\perp}^2) \quad \text{其中} \quad r_{\parallel} = \frac{\eta\cos\theta_i - \cos\theta_t}{\eta\cos\theta_i + \cos\theta_t}$$
$$k_t = 1 - k_r \qquad r_{\perp} = \frac{\cos\theta_i - \eta\cos\theta_t}{\cos\theta_i + \eta\cos\theta_t}$$

上图为纯折射无菲涅耳，下图为有菲涅耳。

可以看出明显的侧视反射变化与全反射界面的渐变：

基本光线追踪的大体思路：

对于每一个像素发射一条光线(视线)，在最近物体的交点处结算反射率/透射率/漫反射率等光能分布。如反射率或透射率不为 0 则根据材质改变光的颜色递归，若漫反射不为 0 则计算交点的直接光照量 C，与光线当前的能量(颜色)相乘后将颜色叠加到像素上。

直接光照量 C 的计算为：从交点将每个光源的检测点发射光线(射线)判断是否有其他物体挡在前面，若没有则通过 BRDF 局部光照模型计算入射方向采集到的能量。

四、 PT,PM,PPM,SPPM 渲染器及渲染优化

1、基于路径追踪的渲染器(PT)

PathTracing 路径追踪是基本光线追踪的改良，有若干不同的版本，简单实现了 Basic Path Tracing 的改良版本(反射过程中直接统计直接光照)，因收敛太慢，故放弃转为光子映射思路(PathTracing.hpp)。

Basic Path Tracing 在 RT 中碰到漫反射面不终止而根据蒙特卡罗随机一个方向继续追踪直到碰到光源。可以采集到辉映效果，而焦散则难以采集(根源是对光源的追踪光路面积可能很小，随机难以打中)，收敛极慢。

Monte Carlo Light Tracing(TODO)：改为从光源发射射线，当与相机有连线时统计能量，比 BPT 略好。

Bidirectional Path Tracing(TODO)：BDPT 的基本思路为同时从光源和相机发射射线，当两者有连线时即找到了一条光路，由于借鉴双向搜索的思路比上述两者更好。

2、基于光子映射渲染器(PM)

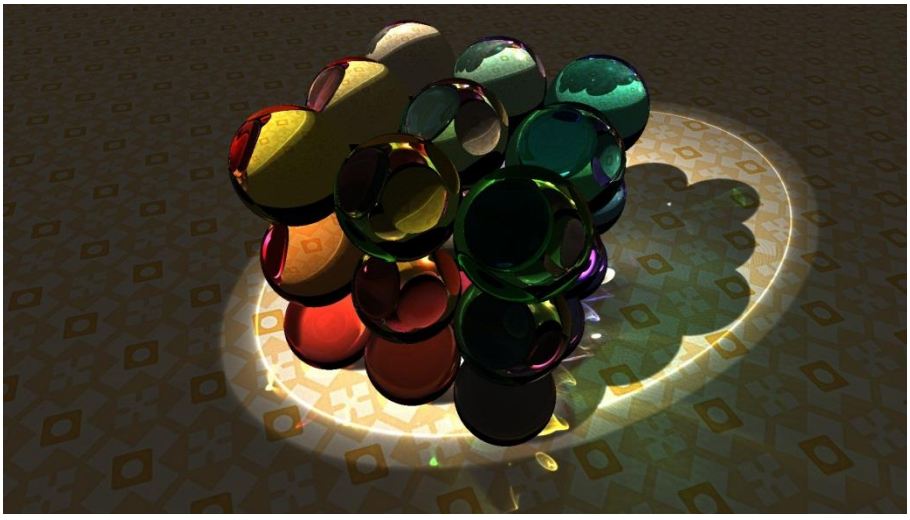
全部代码位于 PhotonMapping.hpp

光子映射的基本思路为从光源向外发射光子，根据光源的颜色和能量分布给予光子不同的能量。光子和光线追踪一样在场景中运动，并在漫反射面上记录为光子图并通过 KD-Tree 维护。通过统计的方法可以估计某点附近的光通量来进行渲染。构建完光子图后从相机进行普通光线追踪，与普通 RT 不同的是统计光亮的方法变为查询光子图给出光通量的估计值。

光子运动碰到面后有许多不同的处理方法：①根据反射/折射/漫反射比例蒙特卡罗发射②按比例拆分光子递归

统计光通量的方法也有许多：①查询 KNN 获得半径 R，则统计量为 K 近邻的能量和除以 π r^2②对固定的 R 统计光子

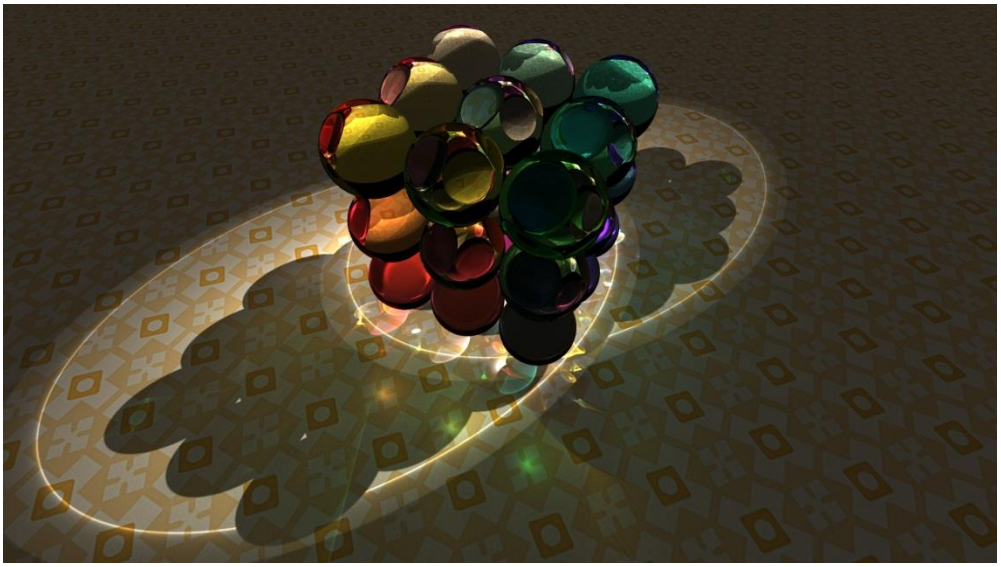
普通光子映射建立光子图可视化后可以表现非常明显的焦散。



3、渐进式光子映射

全部代码位于 ProgressivePhotonMappinga.hpp(代码实现的是 PPM 的改良版 SPPM)

与光子映射的不同在于，PPM 先从相机发射采样视线，并将采集点通过 KD-Tree 维护。然后在从光源发射光子，每个光子只需将能量累计到采样点中而不用存储，故可以源源不断的无穷无尽的发射光子。光子能量收集有若干种方法：①全局固定半径 R 或不断缩小，查询采样点 R 范围近邻来统计光能②根据论文公式 R'=(N+M*a)/(N+M)一轮一轮的修改每个采样点的半径。这两种方法各有好处①计算很快，对于细节收敛更快(非常锐利的焦散边缘)，而②则全局暗部渲染较好，能保证无偏的完全收敛，但要产生锐利的焦散边缘则需要大量的光子。代码中实现的是方法①。



可以看出比光子映射 PM 有着更高的准确度(噪声)。

4、 随机渐进式光子映射(SPPM)

在相机发射采样视线时加入一个像素内随机扰动，并在光子发射中不断进行此过程，目的在于对一个像素进行**超采样**以**抗锯齿**，超采样抗锯齿被广泛应用于光线追踪等算法中。锯齿和噪点不仅来源于物体边缘的像素内颜色变化，也来源于复杂光路中的角度敏感(左 1 左 2 玻璃杯底的红线光路)：

超采样 100 次：

实现的渲染器优化与加速技巧：

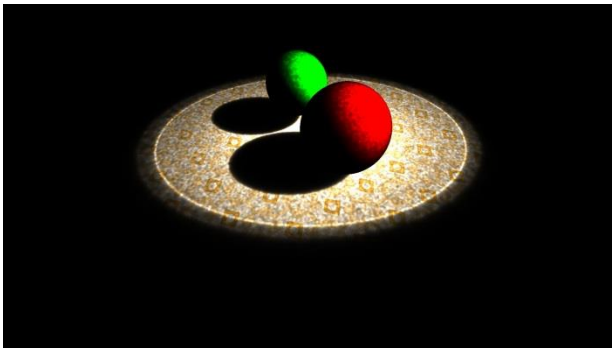
加速主要分为如下几个方面：常数优化或并行化加速计算，在尽可能少的采样下加速收敛、在相同的计算量下计算更多的采样；优化则体现在抗锯齿，更自然的纹理与效果等。

1、OpenMP 并行化(广泛分布于代码中)

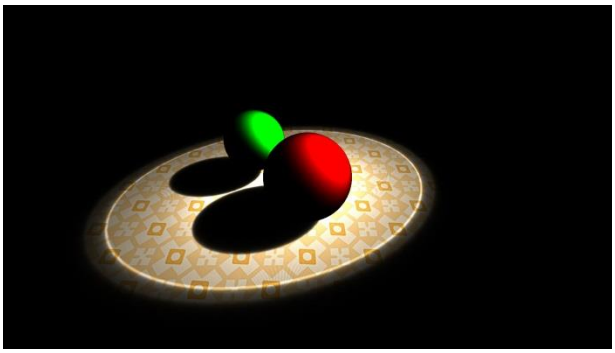
2、重要度采样加速：本来蒙特卡罗随机是 Uniform 的，但由于例如漫反射在平行方向采集到的能量几乎为 0，故通过改变采集分布与权重会加速积分的收敛(广泛应用于 PathTracing、PM 类)。

3、Stratified Sampling：由于图片渲染的积分收敛效率源于方差，而蒙特卡罗方法的随机化天然会叠加一个很大的方差导致收敛变慢，故一个朴素的思路为去除蒙特卡罗而改用确定式分层采样。

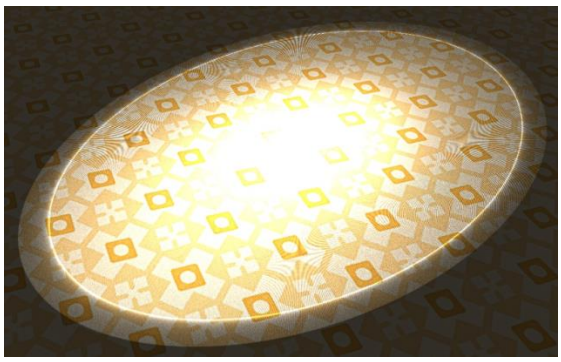
以锥形光源为例，蒙特卡罗发射 50000 个光子结果如下：

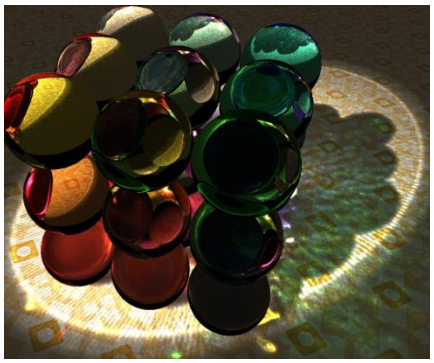


图中可见大量的低频噪声，而采用分层抽样(网格均分初始光子，格内随机)50000 光子结果如下：



可见改变采样可以大大提高收敛效率，但同时会引入一些新的问题，如光源网格形分层采样后会产生明显的摩尔干涉条纹：





解决方法即为在分布无关的维度上随机化，对于锥形光源即网格的 xy 方向是随机的。

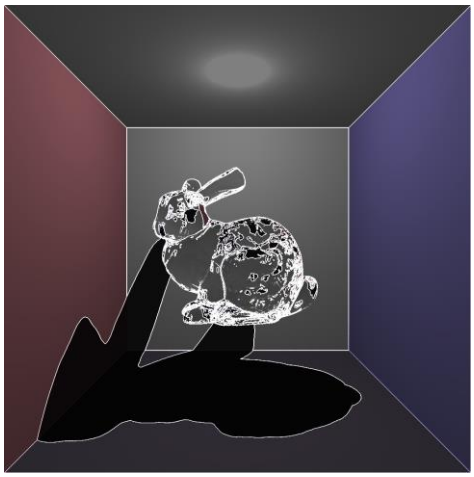
4、 启发式超采样

锯齿的根源有两钟，一种是物体交界面处，另一种是曲面反射折射引起的剧烈色彩变化。以 Bunny 为例

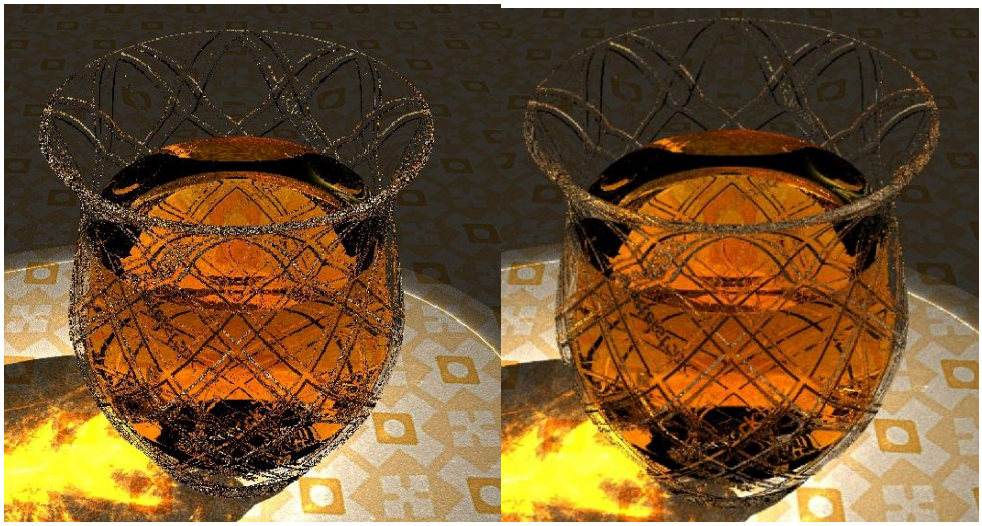
产生噪点和边缘锯齿的根源在于内部折射光路失之毫厘差之千里，边界也是如此。一般的全局超采样效率很低，对于普通漫反射面两三次就足够，而对于剧烈变化区域一千次可能才够，故启发式超采样即为判断哪些区域需要采样，以及达到收敛大致的采样次数来分布采样。

启发式超采样有很多种方法：①根据光路经过的物体 hash 决定是否超采样②基于边缘检测对剧烈区域超采样③基于单像素收敛性统计分析的超采样等(超采样估计颜色可以理解为区间估计，根据方差可以估计大约增加多少次采样可以达到平均水平)。

代码中实现的基于边缘检测超采样（该方法可以提高收敛效率约 50 倍，实验表明对高密度区域单次 PPM 采样约 50 次，特殊区域达到上千次）：



对于茶杯这种折射全反射变化极为剧烈的几何体，超采样极难收敛(噪点)，采用启发式采样后单点采样数增加了近 200 次(加速 200 倍收敛)，一次超采样迭代即可达到普通均匀超采样 200 次的效果：



四、特效渲染

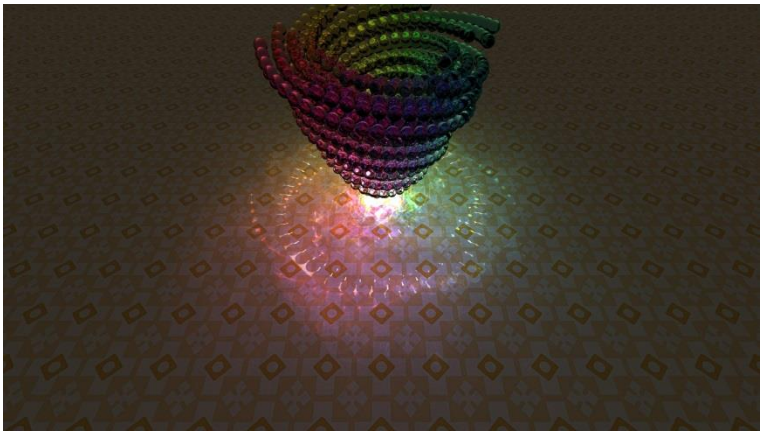
1、环境映射(Environment Mapping)

环境映射将环境贴图(全景图)作为空间边界，环境贴图有多种形式，如立方环境映射(天空盒)，球面环境映射(全景图)，双曲映射等。

代码实现的是球面环境映射(BMP::BallSpace)

2、纹理贴图(Texture)

纹理映射即为将射线交点的纹理坐标 uv 取出，用材质贴图的颜色作为该点颜色：

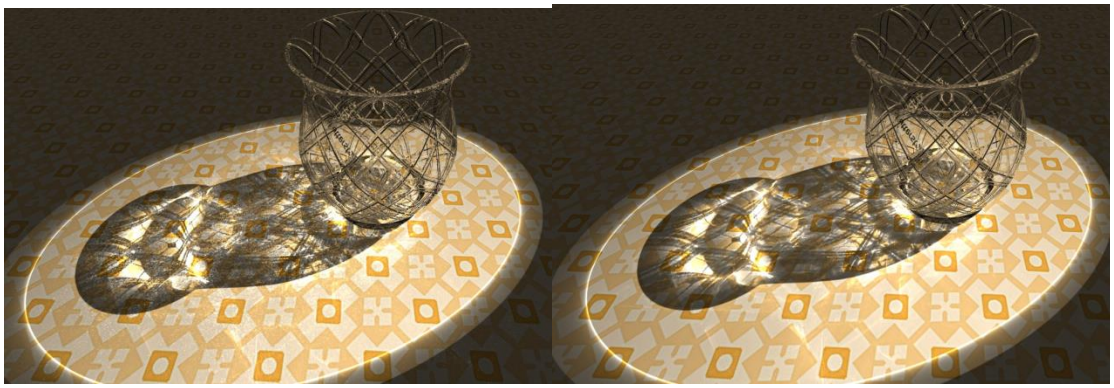


3、法线贴图(Normal Map)

贴图中 r,g,b 分别表示局部坐标系中的真实法向，在获得交点纹理坐标 uv 的同时，额外获得交点几何空间内纹理 u 增加的方向即可构建空间局部坐标系，叠加上法线贴图对应位置的法向即为真实法向(相关代码位于 CollidePointData::Normal())。

但若直接使用纹理的话对于透明物体会产生折射光斑噪声(因为法线贴图是像素化的，相当于鳞片附着在物体上)，解决方法为使用双二次线性插值进行贴图平滑(相关代码位于 BMP:: getPixelLinear)

左图为未平滑的茶杯；右图为平滑后的茶杯(注意焦散部分)



4、景深

景深属于摄像机特效，在产生光线时起点做局部随机化，根据焦平面确定射线方向重复采样即可产生景深效果(相关代码位于 Camera)

5、体积光

以光子映射产生体积光(相关代码位于 VolumetricLighting.hpp)，会在光子运动过程中根据距离衰减并保存空间光子图，在视线追踪过程中采集空间光子叠加亮度。优化：光子半径 R 光线射线 KD-Tree 加速获得碰到的光子，避免小步长迭代。但存在少许问题，无法捕获光源周围非光线方向的体积光。