蒙特卡洛光线追踪器程序报告

傅珂杰 （11824029）

目录

1.概述

本程序是基于 Qt 5.11.2 clang 64 bit 开发，经测试可以在 macOS Mojave 10.14.4 上正常运行，编译本程序需在安装 QT 的前提下，双击 MonteCarloPathTracer.pro 即可在 QT Creator 完成 build。

本程序运行界面如下图所示，有已经配置好相关信息的四个场景（但是导入*.obj 还是需要用户自己完成，以免用户改变*.obj 对应文件路径），此外，本程序支持用户导入自己的*.obj 文件，自行设置相关参数（光源参数只允许增加新的光源，不允许修改已经配置好的光源），支持用户将渲染好的场景保存为*.png。用户相关操作，都会有右下角的文本浏览器进行反馈。本程序对生成的图像进行了 Gamma 校正，Gamma 值为 2.2。

关于 GUI 的使用会在后面的章节中进行接收

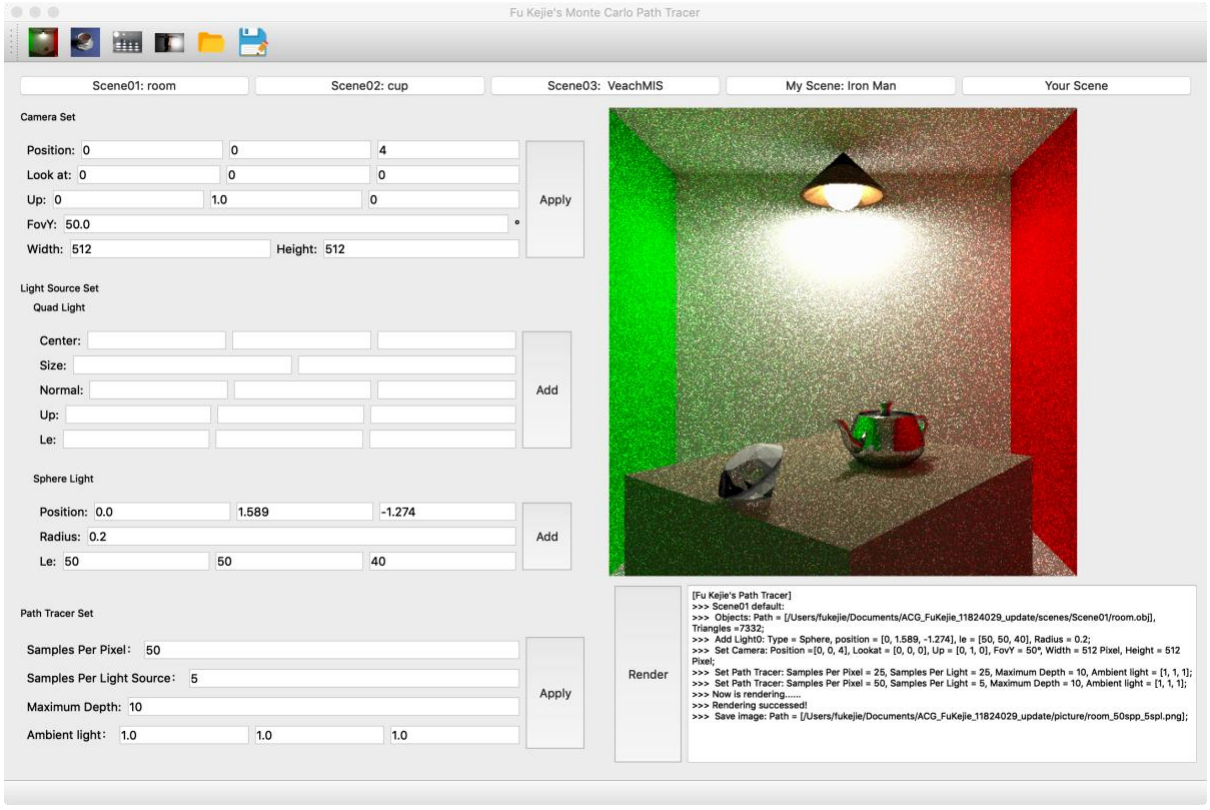


图 1 程序运行用户界面

本程序主要依赖了如下 QT 封装的库：
<QPainter> 用于可视化;
<QVector> 类似于 std 中 vector 容器;
<QVector3D> 和<QVector2D>三维和二维单精度浮点向量;
<QFileDialog> 用于获取“打开”“保存”操作的路径;
<QWidget>和<QApplication>用户图形界面相关的部件。
其他除了 c++标准库之外，没有使用其他第三方库。

本程序可以在非 GUI 下进行参数设置，如下图所示，在主函数 main（）中，注释掉下面的内容，可以设置像素采样，光源采样，以及最大深度，打开文件路径，保存

文件路径，进行编译运行，渲染生成的图片会在窗口中进行显示，同时会自动根据保存路径进行*.png 的保存。

```
int main(int argc, char *argv[])
{

    QApplication a(argc, argv);
    MainWindow w;
    PathTracer tracer;
    srand(time(nullptr));                                                    △ implicit conversion

/*------------------ Uncomment this, then you can render the scene without the gui-------------------------*/
//    string openpath="/Users/fukejie/Library/Mobile Documents/com~apple~CloudDocs/Computer Graphics/note&scenes/Scene02/room.obj";
//    string savepath="/Users/fukejie/Documents/ACG_FuKejie_11824029_update/scenes/Scene01/myRoom0413_pixel150s_light5s.png";
//    int ssp = 50;
//    int spl = 5;
//    int maxdepth =10;
//    scene01_room(tracer, w, ssp, spl, maxdepth, openpath, savepath);
//    scene02_cup(tracer, w, ssp, spl, maxdepth, openpath, savepath);
//    scene03_VeachMIS(tracer, w, ssp, spl,maxdepth, openpath, savepath);
//    myscene_IronMan(tracer, w, ssp, spl,maxdepth, openpath, savepath);
/*------------------ Uncomment this, then you can render the scene without the gui -------------------*/

    w.show();
    return a.exec();
}
```

图 2 主函数 main()直接设置渲染参数并可视化

2.主要数据结构

2.1 Triangle 类

Triangle 类存储的是从*.obj 中读取的三角形单元，*.obj 中多边形（四条边及以上）会被分解成多个 triangle 对象存储。每个 Triangle 可以都会有一个 AABBox(求交加速结构，后面会讲)，然后有一个判读与追踪光线 ray 相交的成员函数 intersect（），会返回是否相交，如相交，则会修改参数中引用的部分，交点信息。

```
class Triangle
{
public:
    Triangle();
    ~Triangle() {}

    Triangle(QVector3D *vertPos, QVector3D *vertNorm, Material *mtl);
    void initialize(QVector3D *vertPos, QVector3D *vertNorm, Material *mtl);


    /*
     * Update the AABBox of this triangle
    */
    void updateAABBox();

    /*
     *  Check whether or not the ray intersects the triangle
    */
    bool intersect(const Ray &r, Intersection &intersection);



    QVector3D vertexPosition[3];//The positions of the three vertices
    QVector3D vertexNormal[3];// The normal vectors of three vertices
    QVector3D origin;
    QVector3D e1;
    QVector3D e2;
    AABBox aabbox;// The AABBox of this triangle
    Material *material;

};
```

2.2 Ray 类

Ray 类是生成的追踪光线，里面包含了追踪光线的原点 origin 和光线的方向 direction,同时为了后续避免光线追踪过程中因为浮点误差造成自交和对光源求交，设置里 t_min 和 t_max 即光线路径长度的最小值和最大值。

```
class Ray
{
public:
    Ray();

    Ray(const QVector3D &i_origin, const QVector3D &i_direction)
    {
        initialize(i_origin, i_direction);
    }

    void initialize(const QVector3D &i_origin, const QVector3D &i_direction);

    //Ray = origin + t * direction
    QVector3D origin;// Origin
    QVector3D direction;// Direction
    float t_min = 0;//The intersect distance must larger than t_min
    float t_max = 3.4e38f;
    QVector3D inverseDirection;// inverseDirection = 1 / direction

};
```

2.3 Material 类

Material 类用于存储在读取*.mtl 文件时材料的属性

```
class Material
{
public:
    Material(){}
    QVector3D Ka;// The Ka statement specifies the ambient reflectivity
    QVector3D Kd;// The Kd statement specifies the diffuse reflectivitys
    QVector3D Ks;// The Ks statement specifies the specular reflectivity
    QVector3D Tf;// The Tf statement specifies the transmission filter
    float Ni;// The Ni specifies the optical density for the surface.
    float Ns;// The Ns specifies the specular exponent for the current material.

    QVector3D Ke;
};
```

2.4 Camera 类

Camera 类用于设置相机参数，屏幕尺寸，和对指定像素（i，j），在世界坐标系内生成采样光线。

```
class Camera
{
public:
    Camera();


    /*
     * Initialize the camera by inputing the position , the lookat, the up direction of the camera
    */
    void initialize(QVector3D myPosition, QVector3D myLookAt, QVector3D myUp);

    /*
     * Set view port
    */
    void setViewPort(float myFovY, float myWidth, float myHeight);

    /*
     *  Generate a simple ray at the (x,y) at the screen coordinate system
    */
    Ray generateSampleRay(int x, int y);

    float width;// The width of the view port
    float height;// The height of the view port

    QVector3D position;// The camera position
    QVector3D lookAt;// The position the camera look at
    QVector3D up;// The up direction of the camera
    QVector3D right;// The right direction of the camera


    float fovy;// Specifies the field of view angle, in degrees, in the y direction

    float halfClipPortWidth;//The half width of the clip port
    float halfClipPortHeight;// The half height of the clip port
};
```

2.5 LightSource 类

LightSource 基类包含纯虚的成员函数 generateSampleLightRay()，以及光源中心 center 和光源强度 emission。

```
class LightSource                                                    'LightSource' has no
{
public:
    LightSource() {}
    virtual ~LightSource() {}

    /*
     *  virtual function for generating a sample light ray from the point to the light source
    */
    virtual Ray generateSampleLightRay(QVector3D &point,float &length,float &cosTheta_O, float &area) = 0;

    QVector3D center;
    QVector3D emission;

};
```

本程序有两个继承的光源类：QuadLight 和 SphereLight

2.6 Scene 类

Scene 类，能够导入*.obj 文件，读取*.mtl 文件，并将其存储在 tiangles 的容器中，容器用的是 QT 封装的 Q

```
class Scene
{
public:
    Scene();

    ~Scene();


    /*
     * Read *.obj and *.mtl together to set the objects
    */
    void loadObjects(string filePath);

    /*
     *
    */
    void loadMaterials(string filePath);

    /*
     * Check whether or not current ray intersects the triangles, further find the nearest intersections
    */
    bool intersect(const Ray &ray, Intersection &intersection);

    /*
     * Construct KdTree
    */
    void constructKdTree();

    /*
     * Check whether or not the point can see the light source by a sample ray from sampling from the light source
    */
    bool seeLightSource(const Ray &ray, const float length);

    /*
     * Update AABBox
    */
    void updateAABBox();

public:
    QVector <Triangle*> triangles;
    QVector <LightSource*> lights;
    map<std::string, Material> materials;//The materials used in this scene ( map<material name, material attributes>)
    Camera camera;
    treeNode kdTree;
    AABBox aabbox;
};
```

2.7 PathTracer 类

PathTracer 类是本程序的核心，其中包含了关于像素采样数 numPixelSamples，直接光源采样数 numDirectLightSamples，追踪深度 maxdepth，以及最终生成的图像信息 image。每个 PathTracer 只能保存一个 Scene 对象，render()成员函数会逐像素，利用 scene 中的相机对象 camera 生成采样光线 sampleRay,对 scene 成员对象调用 traceRay()函数，进行光线追踪。每个追踪点根据相交情况会先计算直接光装，再根据反射还是折射计算间接光照，的出射量为直接光照+间接光照+环境光照之和。对于直接光照和间接关照的蒙特卡洛采样计算，在第四部分会进行介绍。

```
class PathTracer
{
public:
    PathTracer():scene(nullptr), numPixelSamples(1), numDirectLightSamples(1), ambientLight(1,1,1), terminateProbility(0.9f), maxDepth(5)
    {
    }

    Scene *scene;
    int numPixelSamples;// The number of simples at a pixel
    int numDirectLightSamples;// The numberof simples for Monte Carlo  Direct Illumination
    QVector3D  ambientLight; //The ambient light
    float terminateProbility;//The probility to terminate when current >maxDepth;
    int maxDepth = 5;//The max depeth when it needs to check whether to terminate or not
    QColor **image;

    /*
     * Render the scene, and save the color at the **image
    */
    void render();

    /*
     * Release the data stored at the **image
    */
    void releaseImage();

    /*
     *  Set the number of the sample rays per pixel
    */
    void setNumSamplesPerPixel(int n);

    /*
     * Set the number of sample rays per light source
    */
    void setNumSamplesPerLightSoure(int n);

    /*
     * Set the ambient light
    */
    void setAmbienLight(QVector3D l);

    /*
     * Set max trace depth
    */
    void setTraceDepeth(int n);

    /*
     * Key function:
     * Cenerate a samole ray from the current intersection,and trace the ray, calculate the illumintion!
     * Return the imllumination for the former path
    */
    QVector3D traceRay(Ray &ray, int depth);

    /*
     * Key function:
     * Calculate direct illumination from the light source
     *
    */
    QVector3D directIllumination(Ray &ray, Intersection &intersection);

};
```

2.8 Display 类

Display 类继承了 QT 的部件类 QWidget 类，可以进行可视化绘制，里面有从主函数 main（）中接收处理完的图像的接口成员函数 loadImage(), 也有自己的成员对象 pathTracer 可以直接在 GUI 中进行设置。

```
class Display : public QWidget
{
    Q_OBJECT
public:
    explicit Display(QWidget *parent = nullptr);

    void initialize();
    void paintEvent(QPaintEvent *event);
    void loadImage(QColor **&img);
    void save();
    void setSize(int width, int height);
    void setImageSavePath(string imgPath);

    QColor **image = nullptr;
    string imageSavePath;

    PathTracer pathTracer;

    bool pathTracerHaveSet = false;
    bool cameraHaveSet = false;
    bool objectsHaveLoad = false;
    bool canSave = false;
private:
    int width = 512;
    int height = 512;
};
```

3.加速说明

本程序针对光线和三角形面片求交做了加速，包括 AABB 包围盒和 SAH Kd-Tree

3.1 AABB 包围盒

```
class   AABBox // Axis-Aligned Bounding Box
{
public:
    AABBox();
    AABBox(QVector3D min, QVector3D max): bottom_left(min), top_right(max){}

    /*
     *  Initialize this AABBox
    */
    void initialize( const QVector3D min, const QVector3D max);


    /*
     * Expand to fit a vertex
     */
    void expand(const QVector3D &vertex);

    /*
     * Expand to fit another AABBox
    */
    void expand(const AABBox &aabbox);

    /*
     * Estimate whether the ray hit the AABBox
    */
    bool isHit(const Ray &ray);

    QVector3D bottom_left;//The vertex at the bottom-left of the AABBox (min)
    QVector3D top_right;//The vertex at the bottom-left of the AABBox (max)
};
```

当光线与 AABBox 内的对象（可能为 1 个或者多个三角形面片构成的包围盒）求交的时候，先对包围盒进行粗判，如果和包围盒相交，则对包围盒内的对象进行求交运算。包围盒求交的伪代码为老师在课堂上所讲的：

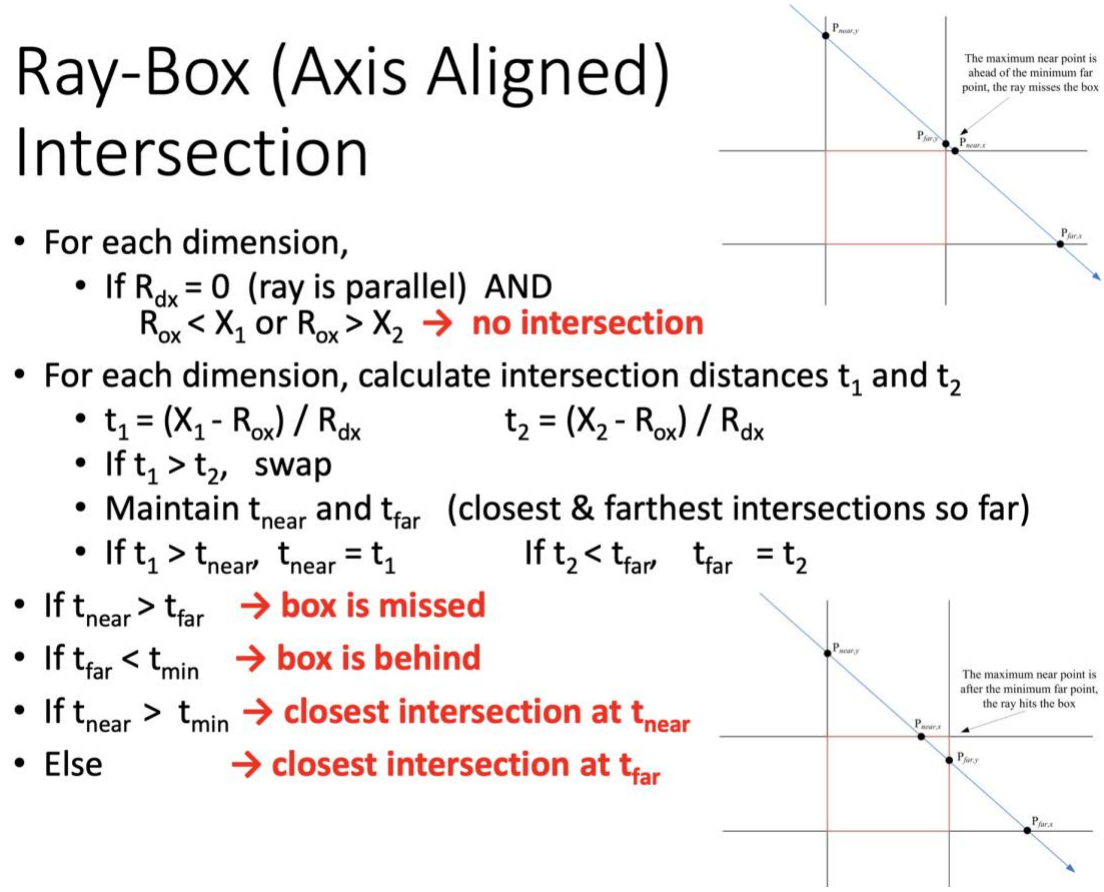


图 3 AABB 包围盒求交伪代码

3.2 SAH KD-tree

KD 数从顶开始递归构建，剖分的过程中，都通过计算剖分的代价，选择代价最小的进行剖分，这里设遍历时间为T_t，求交时间为T_i ,左边节点包围盒的表面积为S_A，其中三角形面片数为N_A，右边包围盒的表面积为S_B，其中三角形面片数为N_A，父节点包围盒的表面积为 S，非空子节点的数目 N（N=0,1,2）那么这次划分的代价为：

$$\text{Cost} = NT_t + \frac{S_A N_A + S_B N_B}{S} T_i$$

本程序中取$T_t = 1$，$T_i = 80$，那么代价函数可以化简为：

$$\text{Cost} = N + 80\frac{S_A N_A + S_B N_B}{S}$$

特别对于叶节点，即不对该节点进行剖分，其代价为：

$$\text{Cost} = 80(N_A + N_B)$$

剖分主要流程：

1. 判断当前节点的三角形面片数是否小于 2 或者当前深度大于 10，若是，则将当前节点作为叶节点，若不是，则进行解析去的步骤。

2. 将代价最小值 Cmin 设为当前节点为叶节点的时候的代价，即

$$\text{Cmin} = 80(N_A + N_B)$$

3. 然后分别对 x, y, z 方向，对该轴剖分成 10*2 段，计算每种剖分下，左右结点的表面积S_A和S_B，对于左右节点三角形个数的计算方式为：

若三角形包围盒在当前轴方向，最大值大于剖分处的值，N_B++;

若三角形包围盒在当前轴方向，最小值小于剖分处的值，N_A++;

4.判读非空节点的数目，若有一个节点没有三角形，则 N=1
5.计算代价

$$\text{Cost} = N + 80\frac{S_A N_A + S_B N_B}{S}$$

6.将计算得到的代价和 Cmin 比较，若更小，则记录下当前的剖分方式，继续精选剖分，直到 x,y,z 轴所有剖分情况都已经做完
7.根据最小代价的情况进行实际的剖分，并让非空的子节点进行剖分。

构建完 KD 树之后，剩下的光线与三角面片求交就变成先对 KD-tree 包围盒的求交粗判断。直到遍历到叶节点，才会对三角形进行求交计算，然后比较所有交点情形下的最小值，作为光线和三角面片实际的交点。

4.几个采样说明

本程序有一个 Sampler 的 namespace,里面包含了 0~1 随机数生成器，光线采样相关的采样器，先对几个比较关键的采样进行说明。

4.1 像素采样

对于宽为 w 像素，高位为 h 像素的窗口，坐标范围为 x 轴方向为 0，1，2……w-1，y 轴方向为 0，1，2……h-1，坐标（0， 0）点对应屏幕左下角，坐标（w-1， h-1）对应于屏幕右上角，屏幕中心的像素坐标为(w/2, h/2)，同时对应于相机的中心。

每次对一个像素采样，对于任意像素点(i， j)，会在 x 方向[i-0.5， i+0.5]生成随机数，y 方向[j-0.5， j+0.5]生成随机数，然后进行坐标变换，在相机 lookat 处的平面。获取该采样点在世界坐标系的坐标，然后再根据相机位置的世界坐标系下的位置 position，生成采样光线的方向。

4.2 光源采样

对于矩形光源（Quad Light）的采样示意图如下:

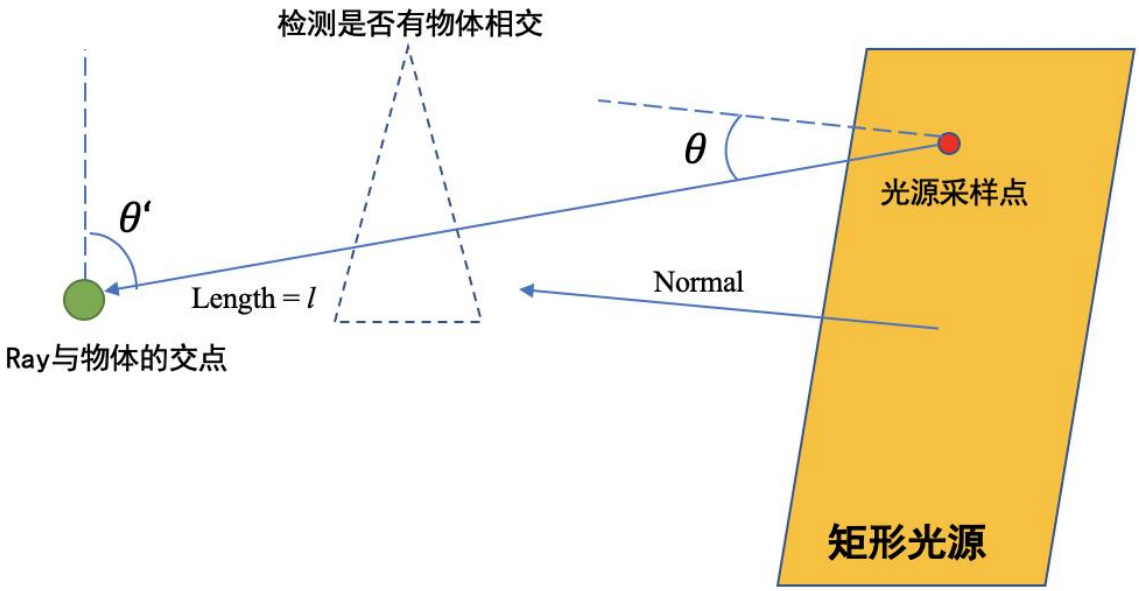


图 4 矩形光源采样示意图

1. 首先在矩形光源上均匀采样出一个点，计算处该点和追踪点连线与光源法向 normal 的夹角 θ，与追踪点处平面法向的夹角为 θ’；
2. 然后判断连线上是否有物体相交，若有，则 continue;若没有则执行下一步；
3. 这次采样的光源到追踪点入射量为：

$$L_i = \frac{cos\theta \cdot cos\theta'}{l^2} \cdot Le \cdot Area$$

这里 Le 为光源的辐射强度，Area 为光源的面积，l 为光源采样点到追踪点的距离。

对于球面光源采样示意图如下：

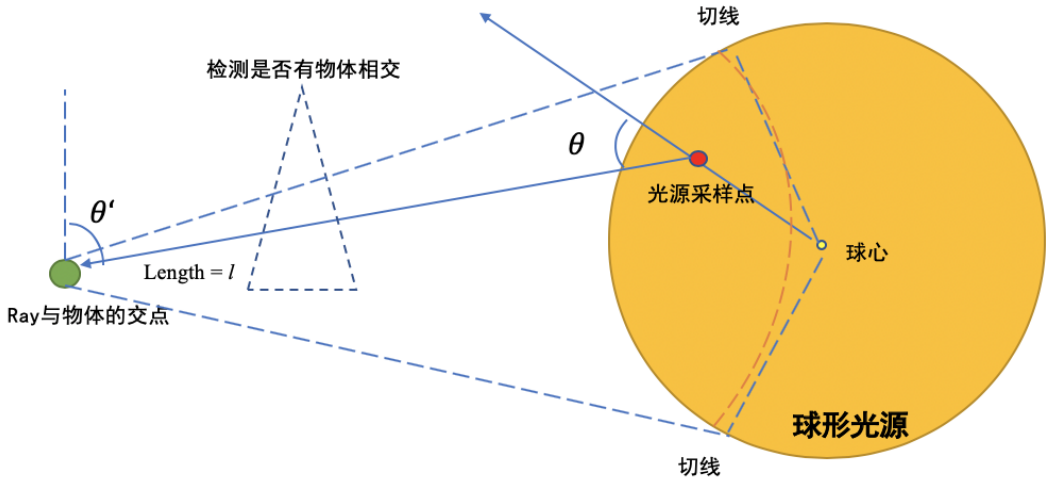


图 5 球形光源采样示意图

1. 首先求出追踪点和球形光源边界的切线，然后在切线范围的球冠上均匀采样出一个点。

2.计算采样点和球形连线的的夹角 θ，采样点与追踪点连线和追踪点平面法向的夹角为 θ’；
3.然后判断连线上是否有物体相交，若有，则 continue;若没有则执行下一步；
4.这次采样的光源到追踪点入射量为：

$$L_i = \frac{cos\theta \cdot cos\theta'}{l^2} \cdot Le \cdot Area$$

这里 Le 为光源的辐射强度，Area 为球冠的面积，l 为光源采样点到追踪点的距离。

最后，若每个光源采样 n 次，共有 k 个光源，则总的光源贡献为：

$$\mathrm{L} = \frac{\sum_{t=1}^{k}\sum_{i=0}^{n} L_{t,i}}{n}$$

4.3 漫反射采样

漫反射采样选择的是 cosinweighted 重要性采样,

$$\mathrm{pdf(direction)} = \frac{cos\theta}{\pi}$$

由于我在计算的时候令漫反射的 brdf 为

$$\mathrm{brdf(l,v)} = \frac{k_d}{\pi}$$

那么根据 Lambert 光照模型，对单条采样光线，其蒙特卡洛估计量为

$$L_{MonteCarlo} = \frac{brdf}{pdf} L_i cos\theta = k_d L_i$$

4.4 高光反射采样

Blinn-Phong 的 BRDF 为：

$$\mathrm{brdf(l,v)} = \frac{n_s + 8}{8\pi} cos\theta^{ns} k_s$$

这里 θ 是半角向量 H 和平面法向 N 之间的夹角。
这里采样的策略是，对理想反射方向以

$$\mathrm{pdf(direction)} = \frac{n_s + 1}{2\pi} cos\alpha^{ns}$$

对反射方向进行采样，这里 α 是采样方向与理想反射方向的夹角，同时如果采样的反射方向与平面法向夹角大于 90°，则对采样方向对理想反射方向镜像一下。

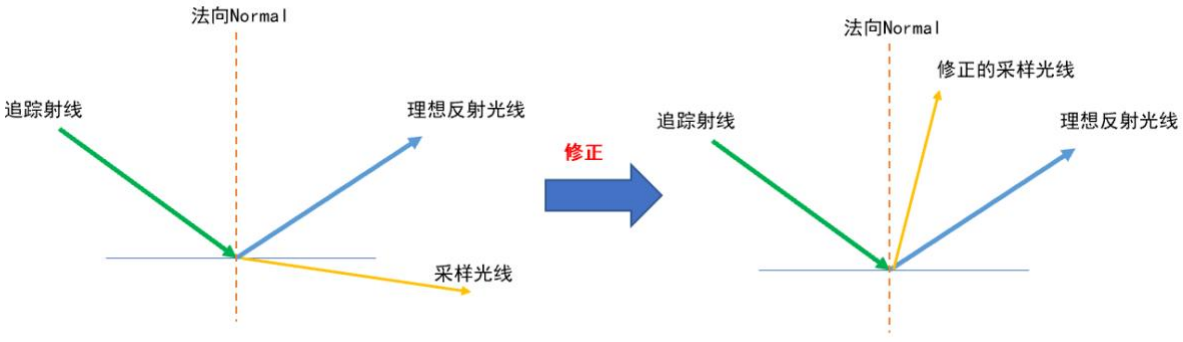


图 6 高光反射采样修正示意图

高光反射蒙特卡洛估计量为：

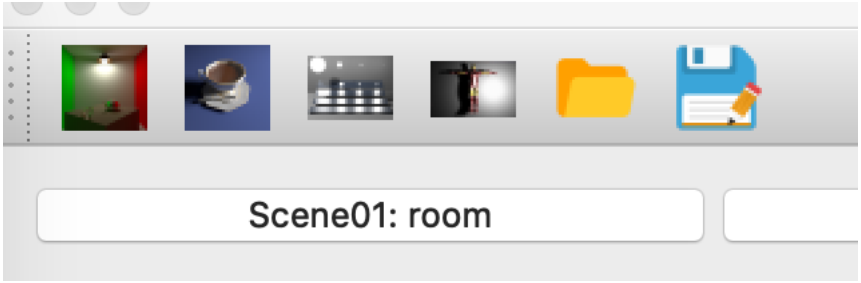
$$L_{MonteCarlo} = \frac{brdf}{pdf} L_i cos\theta$$

5.用户界面使用说明

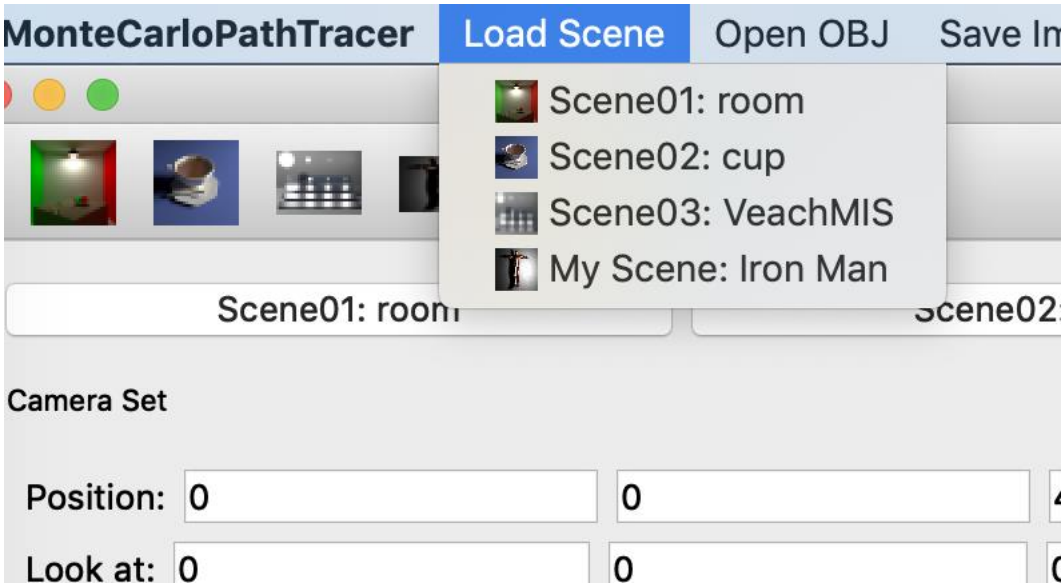
5. 1 导入固定场景

本程序有四个固定场景，分别为 room，cup，VeachMIS 还有我的场景 IronMan,导入对应*.obj 文件即会立即配置好相机，光源，追踪器信息，
方法 1：可以通过点击如下按钮选择：

方法 2：也可以选择工具栏上前四个图标进行 obj 文件的选择：



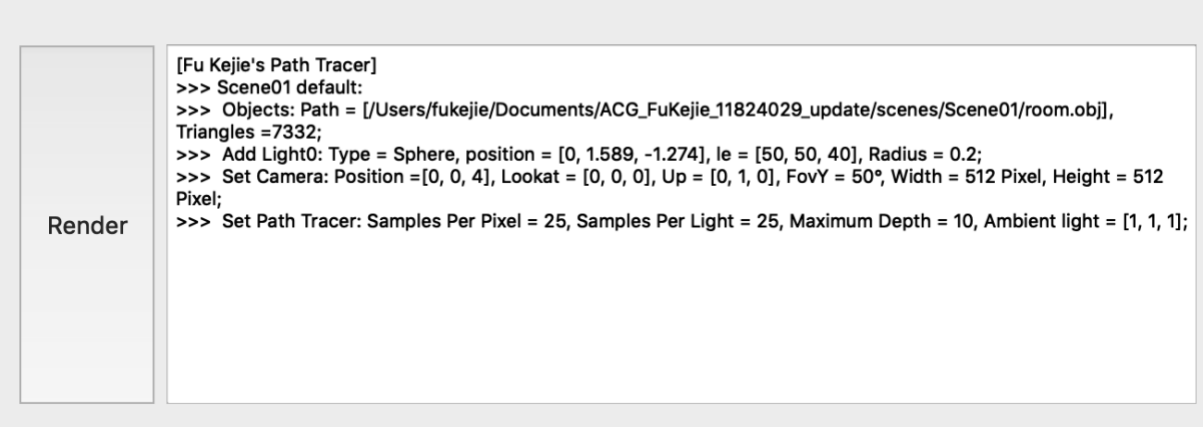
方法 3：也可以选择菜单栏的 Load Scene：



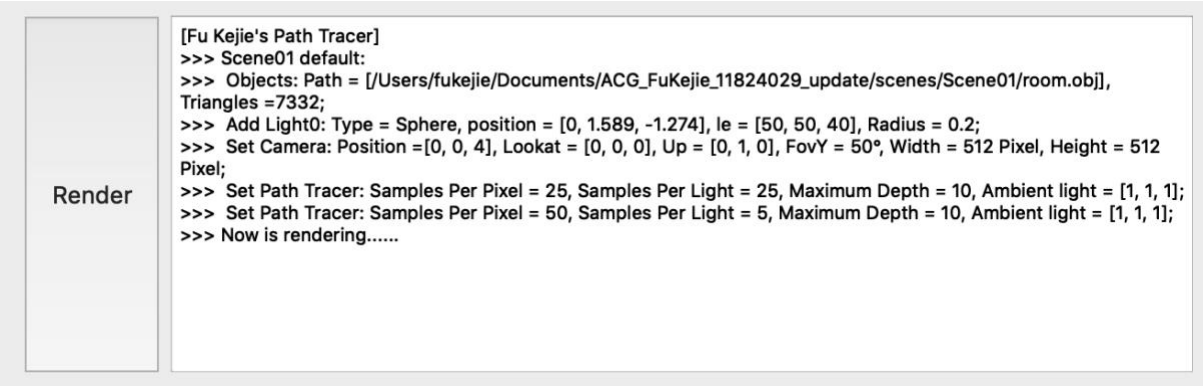
需要注意的是这四个场景的 obj 文件名字已经被写死了，即只能选择对应的文件，无法选择其他的 obj 文件！

导入场景之后依然可以修改相机信息，光源信息，追踪器信息，**但是需要注意的是，光源只能增加，不能撤销和在 Add 之后修改！**

导入固定场景之后，右下角的文本框浏览器会输出设置信息：

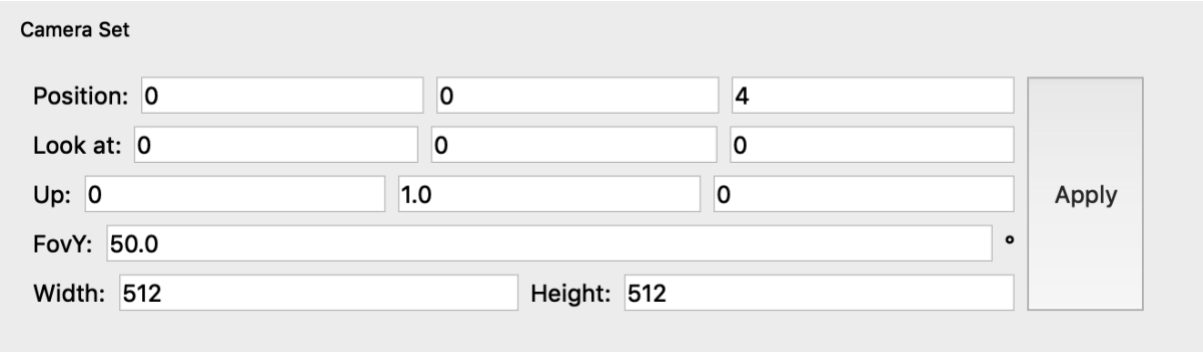


此时，可以直接点击 Render 按钮进行渲染，如需修改一些配置信息，修改后之后，点击 Render 按钮即可：



5.2 设置相机信息

如图，相机信息的界面如下：

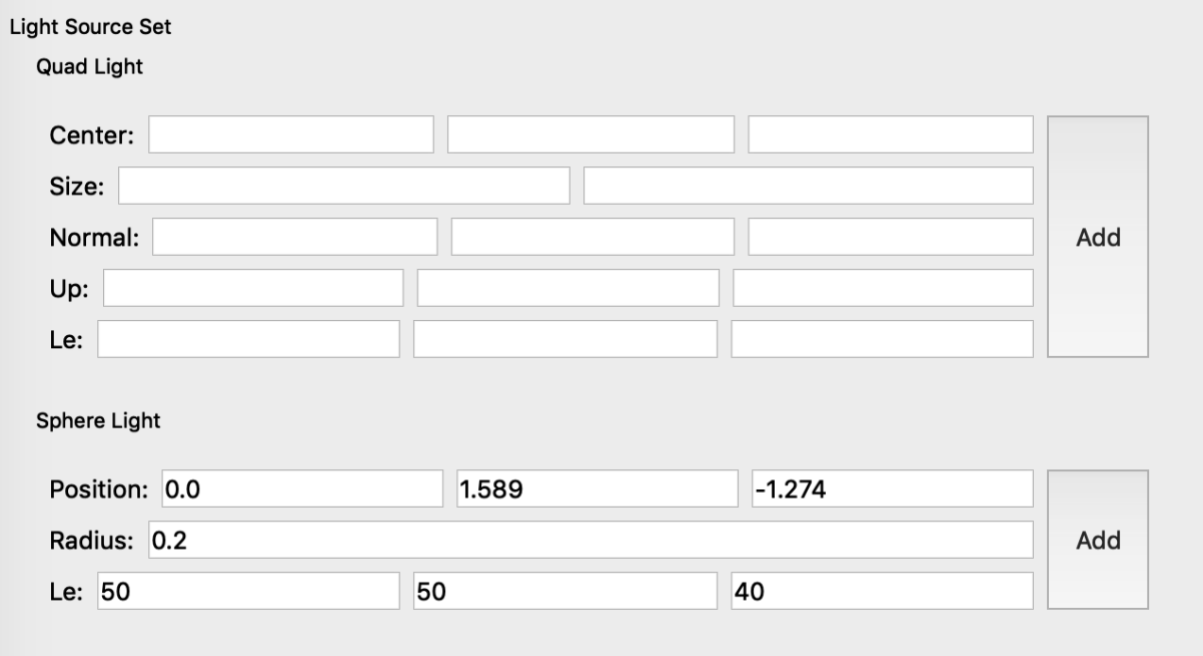


设置好相关信息之后，直接点击 Apply 按钮即可，如果需要修改，则重新修改好信息，再次点击 Apply 即可。

可以设置的信息与老师给的 note 上的内容一致。

5.3 设置光源信息

如下图所示，光源信息设置的界面如下：

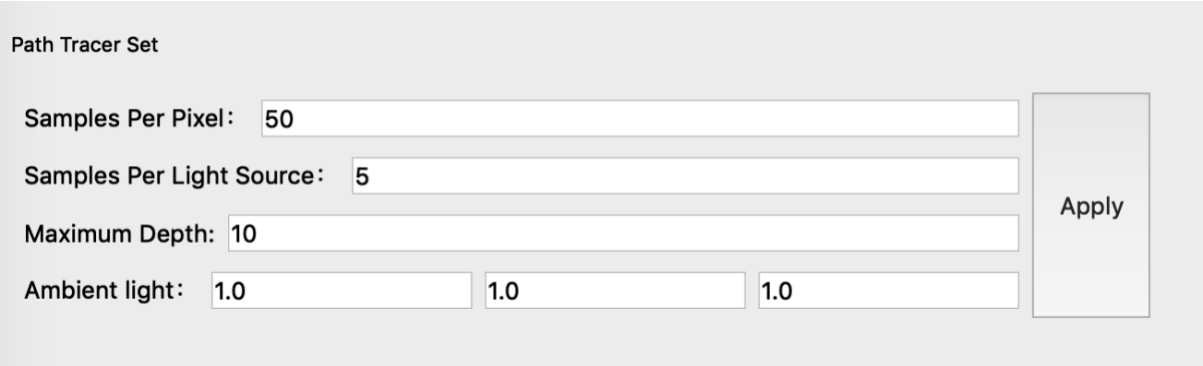


可以添加矩形光源和球形光源，设置完之后点击 Add 即可，右下角文本浏览器会输出相应的设置，并给每次设置的光源设置一个编号，可以任意添加光源。
关于可以设置的信息与老师所给的 note 中的一致，只不过，对于矩形光源，还需要指定一个 up 方向。

需要注意的是:本程序的光源设置只能进行添加，不能对已经添加的光源进行修改！

5.4 设置追踪器信息

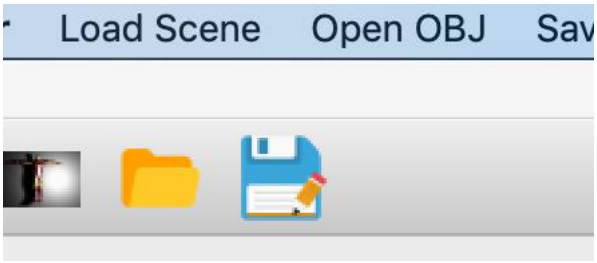
如下图所示，追踪器设置界面如下:



可以设置每个像素的采样点，每个光源的采样点，采样深度，还有环境光照，设置完点击 Apply,右下角文本浏览器会输出相应的设置信息。

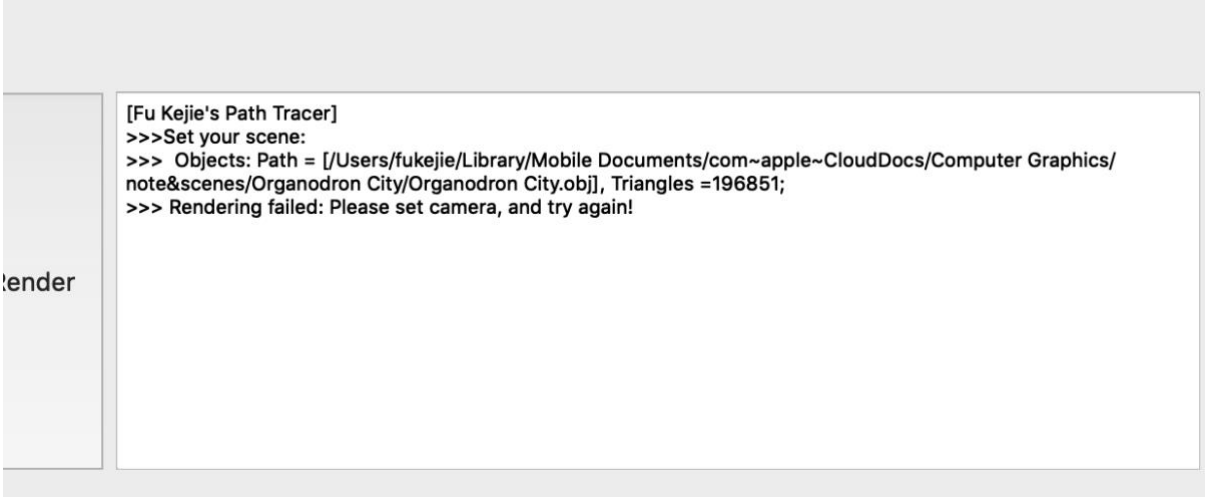
5.5 导入自定义场景

本程序支持导入自定义场景，



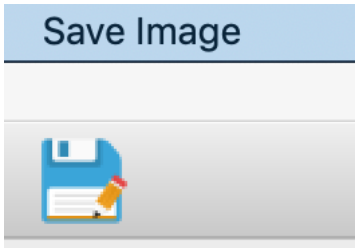
点击菜单栏 OpenOBJ 和工具栏文件夹图标即可以选择任意 obj 文件，请保证 obj 文件同目录下有对应同一名字的 mtl 文件。

需要注意的是：通过自定义导入的 obj 文件，不会自动配置相应的设置信息，只有在设置好相机信息，追踪器信息之后，才能点击 Render 进行渲染！误点 Render，不会进行渲染，文本浏览器会输出还没有设置的信息项，比如会出现如下的报错信息：

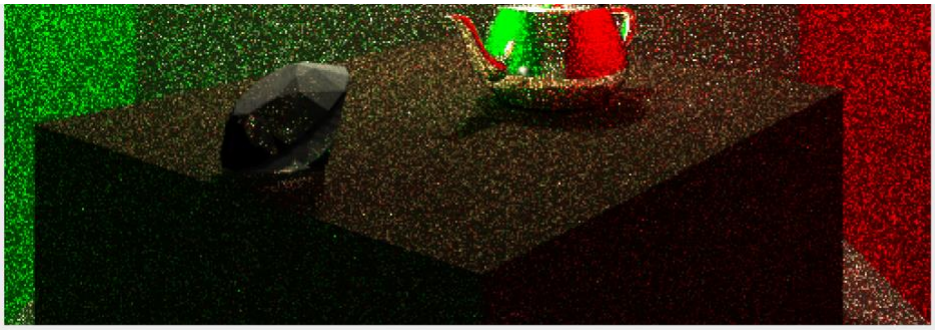


5.6 保存渲染结果到 png

本程序支持将渲染结果保存为 png 格式的图片：



点击任务栏的 Save Image 或者点击菜单栏的保存图标，可以根据需求进行保存，无需自己添加后缀.png。右下角文本浏览器会输出对应的保存路径信息。



Render

```
[Fu Kejie's Path Tracer]
>>> Scene01 default:
>>>  Objects: Path = [/Users/fukejie/Documents/ACG_FuKejie_11824029_update/scenes/Scene01/room.obj],
Triangles =7332;
>>>  Add Light0: Type = Sphere, position = [0, 1.589, -1.274], le = [50, 50, 40], Radius = 0.2;
>>>  Set Camera: Position =[0, 0, 4], Lookat = [0, 0, 0], Up = [0, 1, 0], FovY = 50°, Width = 512 Pixel, Height = 512
Pixel;
>>>  Set Path Tracer: Samples Per Pixel = 25, Samples Per Light = 25, Maximum Depth = 10, Ambient light = [1, 1, 1];
>>>  Set Path Tracer: Samples Per Pixel = 100, Samples Per Light = 1, Maximum Depth = 10, Ambient light = [1, 1, 1];
>>> Now is rendering.....
>>> Rendering successed!
>>>  Save image: Path = [/Users/fukejie/Desktop/saveexample.png];
```

6.渲染结结果

6.1 Scene01: room

图 7 我渲染出的单像素采样 50，单光源采样 5 的 room 图，gamma 校正值为 2.2

6.2 Scene02：cup

图 8 我渲染出的单像素采样 150，单光源采样 5 的 cup 图，Gamma 校正值为 2.2

6.3 Scene03: VeachMIS

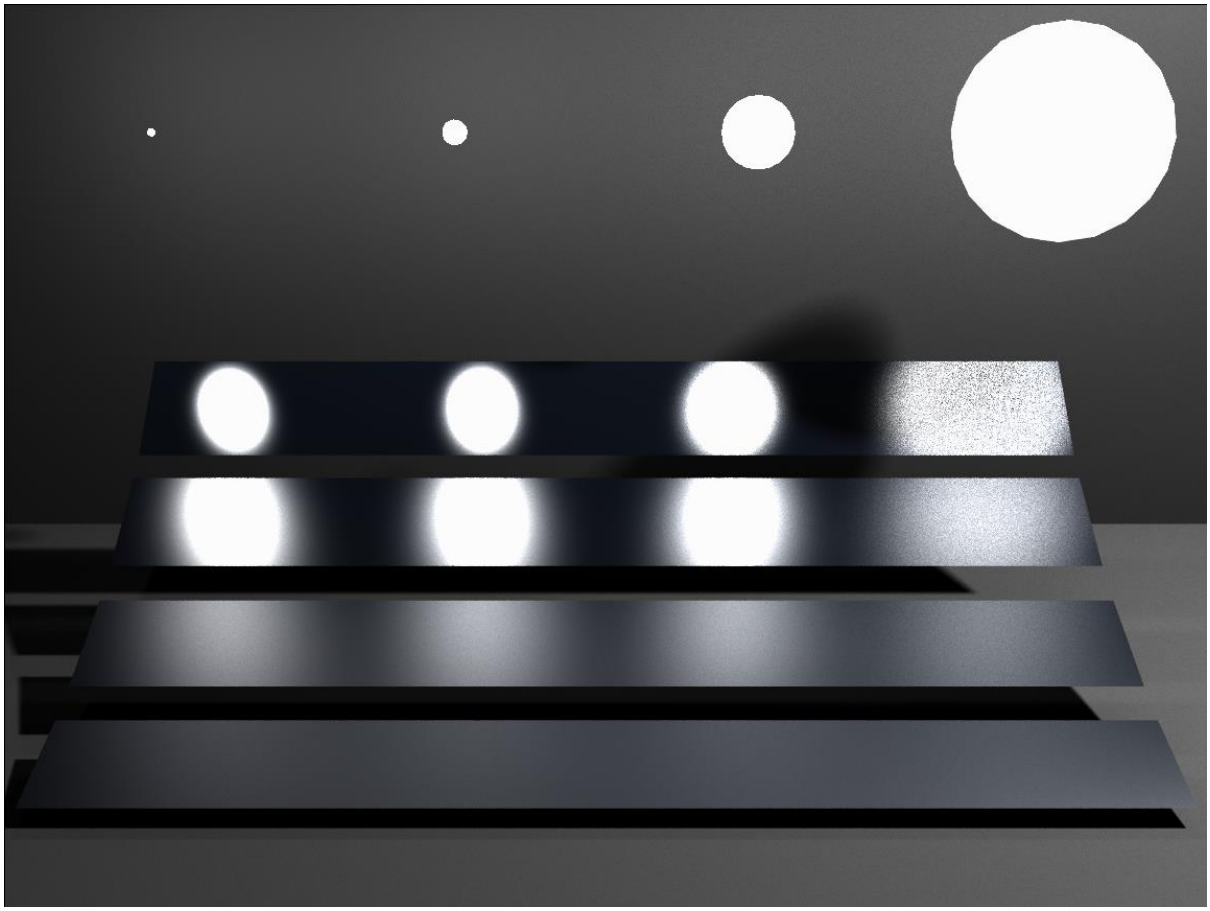


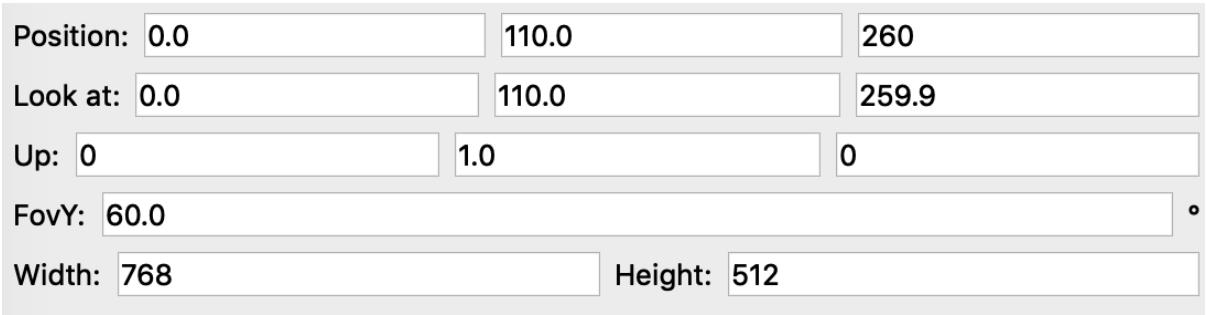
图 9 我渲染出的单像素采样 25 次光源采样 25 次的 VeachMIS 图

6.3 我的场景：Iron Man

图 10 我渲染的单像素采样 25 光源采样 25 次的 IronMan 图

这里的参数设置为:

相机设置：



光源设置：加了一个球形光源