

Simple Monte Carlo Path Tracer

梁晨 计算机学院 本科生院 3180102160

● Introduction

我按照要求，实现了一个基于Monte Carlo方法的Path Tracer。我的作业搭建于在自学[MIT 6.837](#)时所实现的Whitted Style Ray Tracer之上，除了实现了**Monte Carlo Path Tracer**的基本框架算法外，我的主要工作还有：

- 实现了**Cook-Torrance**材质模型
- 实现了**Triangle**、**Sphere**、**Facet**、**Plane**等多种几何以及它们与光线的求交。
- 实现了**Uniform Grid+3DDDA Ray Marchig**加速结构
- 实现了**Bounding Volume Hierarchy(BVH)**加速结构，并可以根据情况进行**Surface Area Heuristic(SAH)**
- 实现了通过**Multiple Importance Sampling(MIS)**，同时考虑面光源、环境光、BSDF的Diffuse项和BSDF的Specular项对光线进行采样。
- 通过命令行参数，**为用户在不同场景下提供多种选择**。用户可以灵活选择进行Whitted Style Ray Tracing或Monte Carlo Path Tracing；在Monte Carlo Path Tracing的框架下，可以选择Phong Shading或Cook-Torrance Shading；加速结构可以选择3DDDA或BVH。
- 在不同选择下，做了多项对比评测，评价渲染的时间开销、空间开销和渲染质量。
- 通过OpenGL实现了**场景预览**，并可以调节相机位置，再进行软件光追，优化了交互体验。

● Arguments

```
//default setting is Whitted Style Ray Tracing with Phong Shading and
3DDDA, without OpenGL preview.

-input scenefile.txt
-size width height
-output outputname.tga
-gui //specify whether to use OpenGL to preview
-spp sample_per_pixel
-MonteCarlo //specify whether to use Monte Carlo Path Tracing
-cook_torrance //specify whether to use Cook-Torrance Shading
-RR_cutdown cutdown //set p for Russian Roulette
-Gamma gamma //realize gamma correction
-SpatialTree // specify whether to use BVH
-resolution resolution_num // when 3DDDA is applied, resolution_num is the
number of grids of an axis; when BVH is applied, resolution_num is the
minimal number of primitives in a box
-SAHeuristic heuristic_num //when BVH is applied, using SAH when # primitives in a
box <= 2*heuristic_num, default is 2000
```

▼ Arguments Passed On Launch

```
✓ -input
✓ diningroom.txt
✓ -size
✓ 800
✓ 800
✓ -output
✓ dining_cook_64.tga
 -gui
✓ -spp
✓ 64
✓ -MonteCarlo
✓ -RR_cutdown
✓ 0.6
✓ -Gamma
✓ 2.2
✓ -resolution
✓ 5
✓ -cook_torrance
✓ -SpatialTree
✓ -SAH
✓ 2000
+ -
```

scenefile.txt格式如下，需要指明Camera、Environment Light以及Geometry(Group)的具体情况：

```
PerspectiveCamera {
    center 0 0 2.5
    direction 0 0 -2.5
    up 0 1 0
    angle 60
}

Lights {
    numLights 1
    Skybox {
        environment.hdr
        R 1000
    }
}

Group {
    numObjects 6
    TriangleMesh {
        obj_file newscene.obj
    }
}

Sphere {
    center 0.5 0.5 0.5
    radius 0.15
```

```

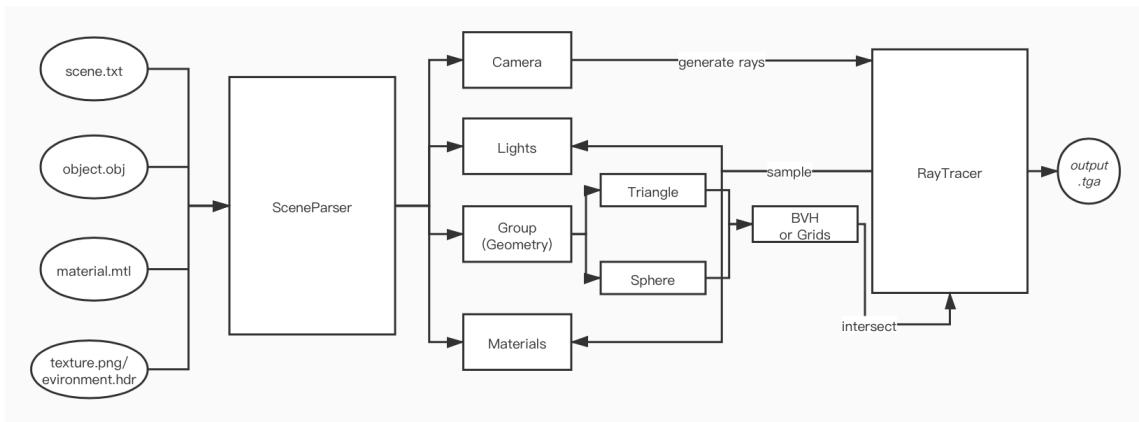
    mtl cornellbox_6_material
}
Sphere {
    center -0.3 0.3 0.5
    radius 0.15
    mtl cornellbox_7_material
}
Sphere {
    center 0.2 0.4 -0.5
    radius 0.1
    mtl cornellbox_8_material
}
Sphere {
    center -0.2 -0.4 0.9
    radius 0.16
    mtl cornellbox_6_material
}
Sphere {
    center -0.6 -0.2 0.1
    radius 0.18
    mtl cornellbox_9_material
}
}
}

```

- Key Algorithms & Data Structures

 - Pipeline and Relationship Between Key Classes

下图说明了各个类之间的关系，同时也说明了我们如何从scenefile.txt中的数据，生成outputfile.tga这张图片。



 - Camera

我们用center、direction、up、horizontal三个向量描述相机的位置与朝向，用angle描述相机的FOV。注意对三个方向向量做归一化。四个virtual函数用来在preview时调整相机。
generateRay()射出光线。

```

class PerspectiveCamera:public Camera{

```

```

public:
    PerspectiveCamera(Vec3f&, Vec3f&, Vec3f&, float);
    Ray generateRay(Vec2f point);
    virtual void glPlaceCamera(void);
    virtual void dollyCamera(float dist);
    virtual void truckCamera(float dx, float dy);
    virtual void rotateCamera(float rx, float ry) ;
    Vec3f getDir() const{return direction;}
    Vec3f getPos() const{return center;}
private:
    Vec3f center;
    Vec3f direction;
    Vec3f up;
    Vec3f horizontal;
    float angle;
};


```

- Light

在Monte Carlo Path Tracer中，我们实现了Area Light和Environment Light两种光源。
samplingLight用来对光线进行采样，getIllumination得到发光的Radiance， L_e 。

```

class AreaLight:public Light{
public:
    AreaLight(Triangle* shape, Vec3f color):shape(shape),color(color)
{lightType=2;}
    void samplingLight(Vec3f& pos);
    float getArea();
    void getIllumination(const Vec3f& objPos, Vec3f& lightPos, Vec3f&
col, float& dist) const;
    void glInit(int id);
private:
    Triangle* shape;
    Vec3f color;
};

class EnvLight:public Light{
public:
    EnvLight(Image* _img, Sphere* sphere):img(_img),shape(sphere);
    void samplingLight(Vec3f& container);
    void getIllumination (const Vec3f & pos, Vec3f &dir, Vec3f &col,
                          float &distanceToLight) const
    void glInit(int id);
    float getArea(){return 0.0;}
private:
    Image* img;
    float* pdfu;
    float* cdfu;
    float** pdfv;
    float** cdfv; //记录采样环境光的信息，需要precompute
};


```

```
    Sphere* shape;  
};
```

- Group

这个类中存取了场景的几何信息，包括BVH(SpatialTree)。

```
class Group:public Object3D{  
public:  
    Group(int _num);  
    void addObject(int index, Object3D* obj); //加入几何体  
  
    virtual bool intersect(const Ray&, Hit&, float);  
    virtual bool intersectShadowRay(const Ray&, float); //光线求交  
  
    void createTree(); //构造BVH  
private:  
    int num;  
    Object3D** arr; //记录几何体数量和它们的指针  
  
    float stepX;  
    float stepY;  
    float stepZ;  
    int resolution; //记录Grid信息  
  
    Vec3f maxPoint;  
    Vec3f minPoint; //记录场景的AABB  
  
    SpatialTree* tree; //记录BVH信息  
};
```

- Materials

为了简洁，我们将Phong Material和Cook-Torrance Material做成了一个类。Phong Material中只有kd、ks、exponent有意义，Cook-Torrance Material还包含mentalness和alpha两个参数来说明材质的金属性与粗糙度。同时texture也包含在材质信息中。

```
class CookMaterial:public Material{  
public:  
    CookMaterial(const Vec3f& d, const Vec3f& s, const Vec3f& e,  
    float exp, float _alpha, float _mentalness, Image*  
    map_Kd):Material(d),ks(s),emittance(e),exponent(exp),map_Kd(map_Kd),alp  
ha(_alpha),mentalness(_mentalness){}  
    Vec3f Shade(Vec3f wi, Hit& hit, Vec3f wo, Vec3f color, bool  
    cosineWeighted) const;  
    virtual void glSetMaterial(void) const;  
    Vec3f getReflectance() const{return ks;}  
    float getAlpha() {return alpha;}  
    float getMentalness() {return mentalness;}}
```

```

    Vec3f getEmittance() const {return emittance;}
    float getExponent (){return exponent;}
    Vec3f sampleKd(Hit&);

private:
    Vec3f albedo;
    Vec3f emittance;
    Vec3f ks;
    float exponent;
    float alpha;
    float mentalness;
    Image* map_Kd=NULL;
};

```

- Monte Carlo Ray Tracer

为了方便，以下略去部分细节，采取伪码形式叙述Monte Carlo Ray Tracing的算法：

```

MC Raytracer(w):
    Lo=(0,0,0)
    intersect(ray,group)
    if(does not intersect with any objects) return
    Radiance_of_Background_Light
    for every light in scene
        sampleLight(pdfLight,wo,wi)
        Lo+= Regularized_Radiance_of_Light*BSDF*cosine(wi,norm)/pdfLight
    sampleBSDF(pdfBSDF,wo,wi)
    if(random()<RR_cutdown)
        Lo+= MC Raytracer(wi)*BSDF*cosine(wi,norm)/pdfBSDF/cutdown
        Lo+=textureColor
    return Lo

```

- Multiple Importance Sampling

- Sample Diffuse Term

在对Diffuse Term做采样时，我们使用cosine weighted hemisphere：

```

void samplingDiffuseRay(float& theta, float& phi)
{
    float thetaCDF=(float)distribution(generator),
          phiCDF=(float)distribution(generator);
    theta=thetaCDF*2*M_PI;
    phi=0.5*acos(1-2*phiCDF);
}

```

- Sample Specular Ray

在对Specular Term做采样时，使用Phong模型和Cook-Torrance模型时采样方法有所不同。使用Phong模型时，我们把镜面反射出射光当成“法向量”，做一个类似于cosine weighted hemisphere的采样：

```

void samplingSpecRay(float& theta, float& alpha,int ns)
{
    float thetaCDF=(float)distribution(generator),
          alphaCDF=(float)distribution(generator);
    theta=thetaCDF*2*M_PI;
    alpha=acos(pow(alphaCDF,1.0/(ns+1.0)));
}

```

使用Cook-Torrance模型时，我参考了[BSDF Shader](#)，对GGX的概率密度函数进行采样。特别注意采样Diffuse Term和Specular Term时，得到的theta, phi是局部坐标系下的，通过toWorld()函数转为全局坐标系的wi:

```

Vec3f sampleGGX(Material* s,Vec3f norm, float& half, float& phi)
{
    float alpha2=s->getAlpha()*s->getAlpha();
    float x=distribution(generator),y=distribution(generator);
    phi=2*M_PI*x;
    half=acos(sqrt((1.0 - y) / (1.0 + (alpha2 - 1.0) * y)));
    return toWorld(half,phi,norm);
}

```

■ Sample Area Light

我的实现中，认为Area Light可以被细分成一个个小三角形，所以对Area Light做采样，就是对一个个小三角形做关于面积A的均匀采样，pdf=1/A:

```

void samplingLight(Vec3f& pos){
    Vec3f a=shape->a,b=shape->b,c=shape->c;
    float r1=(float)distribution(generator),
          r2=(float)distribution(generator);
    float coe_a=1-sqrt(r1),coe_b=sqrt(r1)*r2,coe_c=1-coe_a-
    coe_b;
    pos=coe_a*a+coe_b*b+coe_c*c;
}

```

■ Sample Environment Light

我参考了论文[Monte Carlo Rendering with Natural Illumination](#)中的方法，根据environment light预先计算出一个pdf[nu][nv]，环境光中(u,v)点的概率密度值，与(u,v)点的Luminance成正比，这样就完成了对环境光的importance sampling，再按照以下公式，根据(u,v)点的pdf得到(theta,phi)的pdf。同时实现(u,v)到(theta,phi)的转换，简单地有：theta=2 * u * PI, phi=v * PI。

$$p(\omega) = p(u, v) \frac{n_u n_v}{2\pi^2 \sin \theta}.$$

■ Multiple Importance Sampling Scheme to Combine

对于light和材质的结合，我们使用比较naive的策略，即采材质时我们忽略光源，如果采到面光源，则忽略。在结合Diffuse Term和Specular Term时，我们生成一个随机数，将随机数与对应的weight比较，决定是采Diffuse还是Specular。注意最后的pdf=weight * pdfSpecular+(1-weight) * pdfDiffuse。

```
//sampling Phong Material
if(d<weight){
    samplingSpecRay(theta,alpha,ns);
    wi=toWorld(alpha,theta,Perfect);
    phi=acos(norm.Dot3(wi));
}
else{
    samplingDiffuseRay(theta,phi);
    wi=toWorld(phi,theta,norm);
    alpha=acos(Perfect.Dot3(wi));
}
float pdfD=cos(phi)/M_PI;
float pdfS=0.5*(ns+1.0)*pow(cos(alpha),ns*1.0)/M_PI;
pdf=weight*pdfS+(1-weight)*pdfD;
```

- BVH的构建

我没有采用naive的中位数分割方法，而是使用了SAH。但SAH搜索最小Cost的代价过大，在树的每个节点都为 $O(n * n)$ 。所以我设定了一个阈值SAH_NUM，这个数值由用户给定。当节点中的几何体数小于等于SAH_NUM时，才进行Cost的搜索；否则采用naive的中位数分割法。

$$C = t_i + p_a N_a t_i + p_b N_b t_i$$

```
sort(shapeBuffer) according to center coordinate along the longest axis
unsigned int optimal=shapeBuffer.size()/2;
//search cost
if(optimal<=SAH)
for(int i=1;i<shapeBuffer.size();i++)
    split shapeBuffer into two groups, [0,i) in A, [i,size) in B
    calcualte cost=i*SA+(size-1-i)*SB
    if(cost<minCost){
        minCost=cost;
        optimal=i;
    }
}
```

- Scene Evaluations:

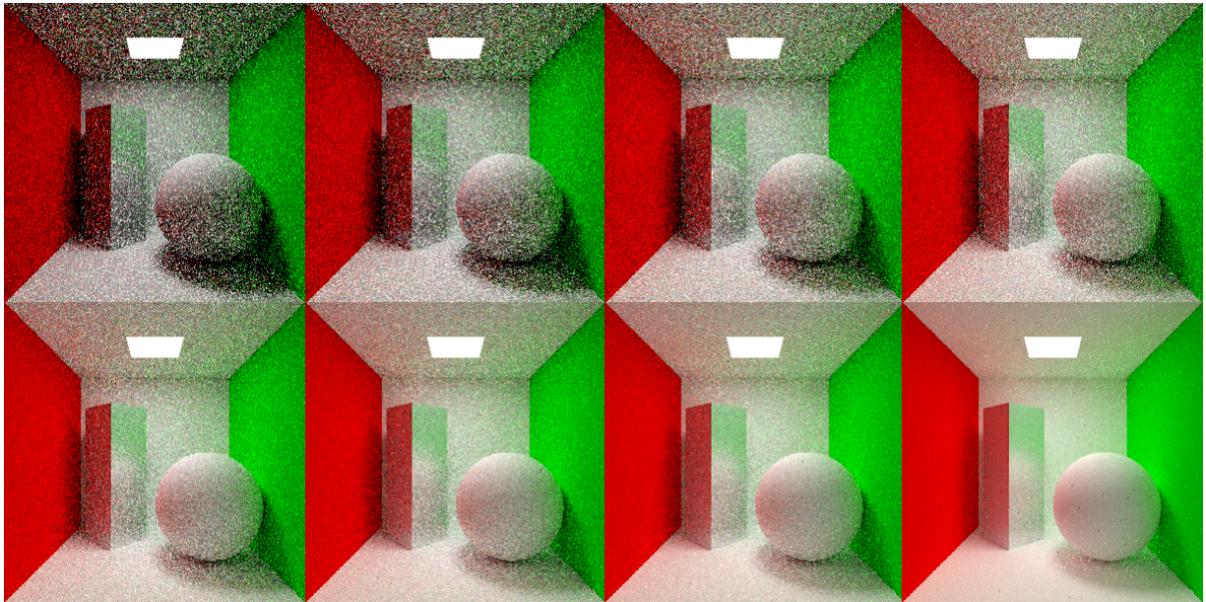
- 对于CornellBox，我用Phong模型和Cook-Torrance模型各做了1spp-64spp，1024spp八张（400x400），用来对比。又做了800x800，用Cook-Torrance的64spp图，来看最终效果。
- 对于CarDusk，我用Cook-Torrance模型做了1spp-64spp（400x400）共七张图。
- 对于CarDay，我用Phong模型和Cook-Torrance模型各做了1spp-32spp（400x400）六张

图，用来对比。又做了800x800，用Cook-Torrance的64spp图，来看最终效果。

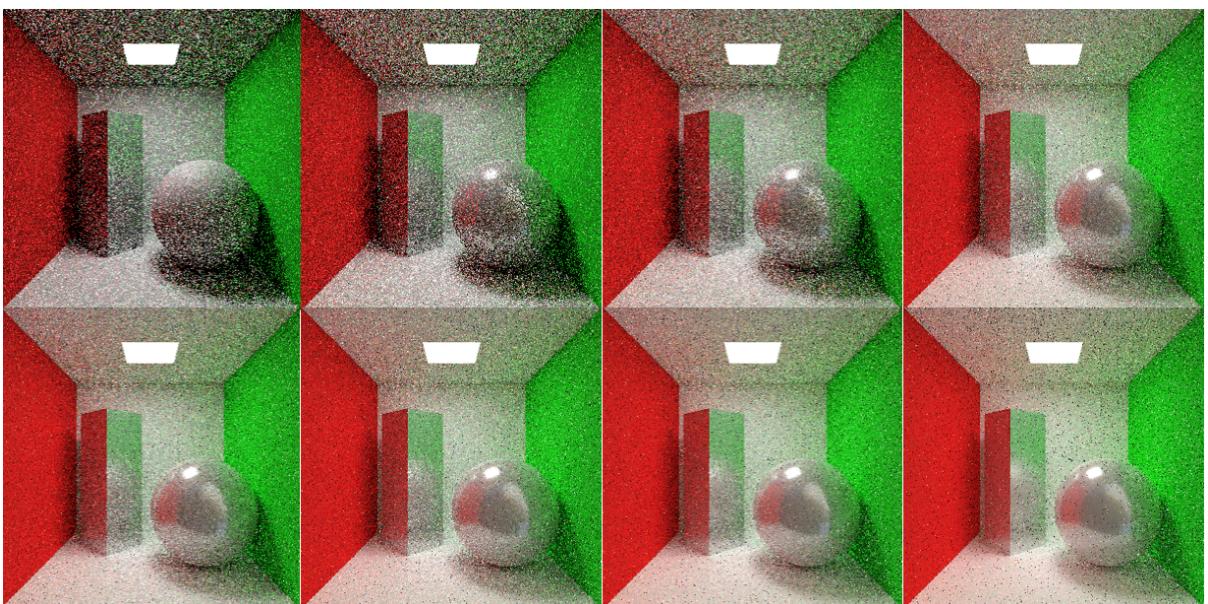
- 对于DiningRoom，我用Cook-Torrance模型做了1spp-32spp(800x800)共5张图。
- 我自己构建了newscene场景，来验证Cook-Torrance模型正确实现，同时测试其效果。我用Phong模型和Cook-Torrance模型各做了1spp-64spp, 1024spp八张 (400x400)，用来对比。又做了800x800，用Cook-Torrance的64spp图，来看最终效果。

1. CornellBox:

400x400 Phong Shading:

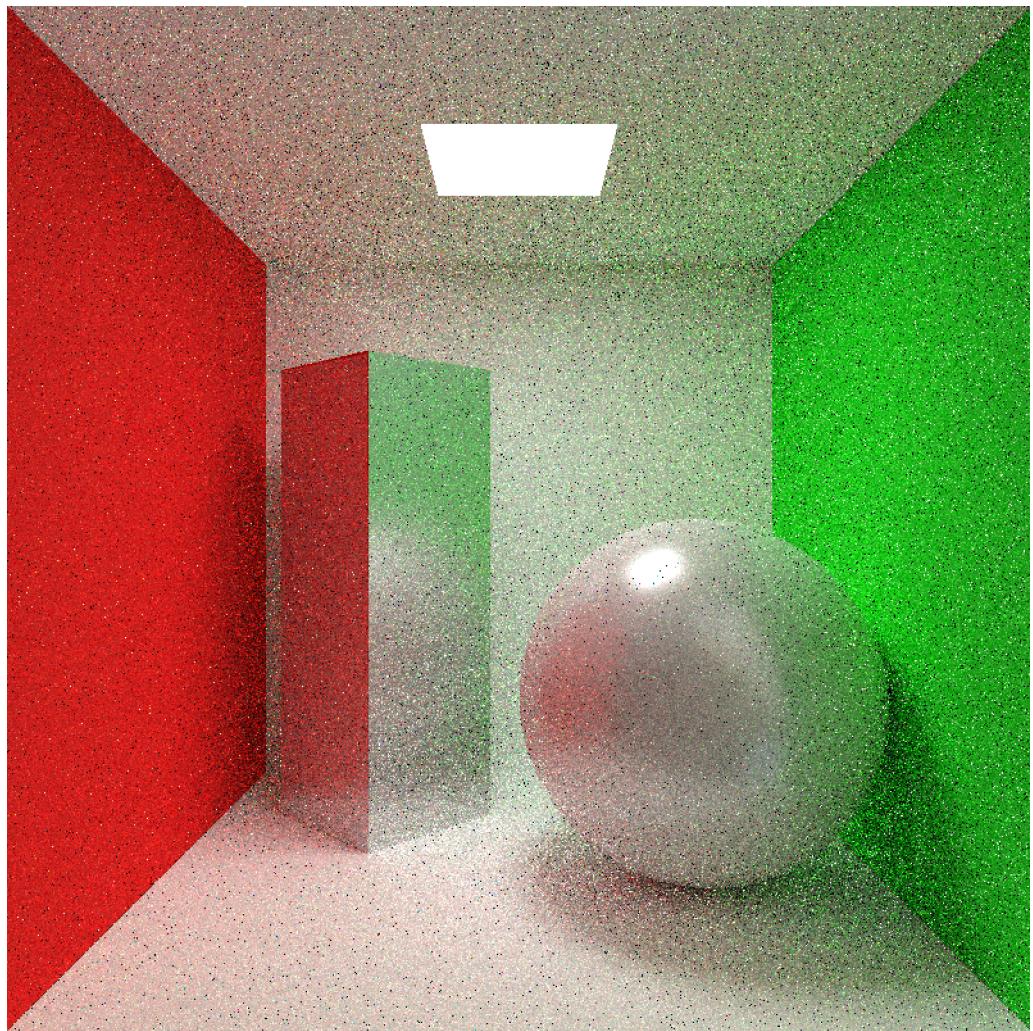


400x400 Cook-Torrance Shading:



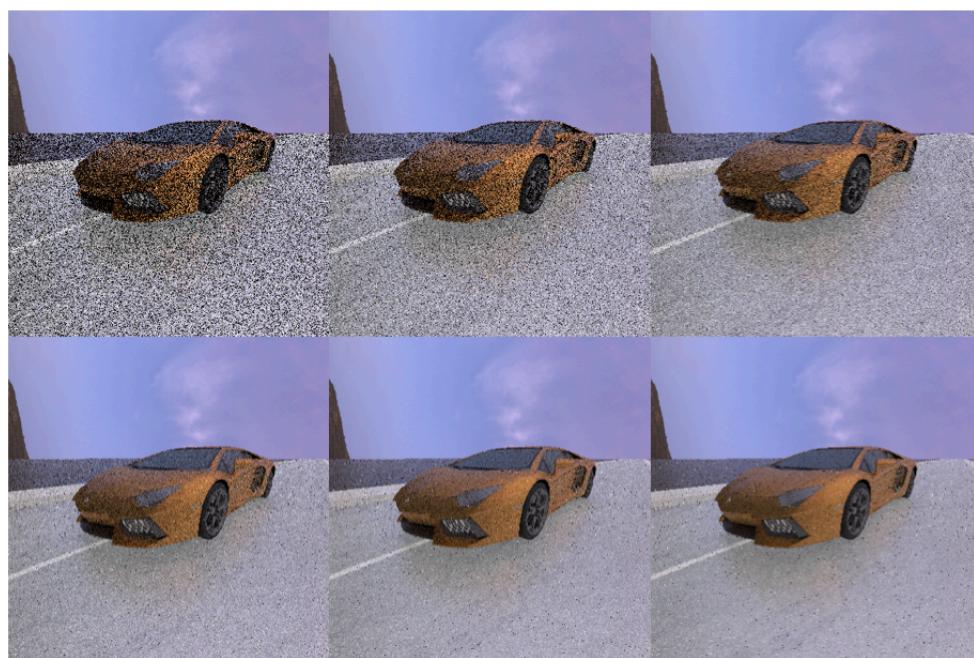
可以看到，使用Cook-Torrance Shading，反射光得以更好表现。同时，我们能更灵活调节表面材质，控制球体Mentalness=0.2,Roughness=0.3，使得球体在整体看起来比较diffuse的情况下，仍有一定部分的高光和反射。另外，我们观察到64spp和1024spp噪点相差不多，故接下来的场景都做到64spp。

800x800 Final Result:



2. CarDusk:

400x400 Cook-Torrance Shading:



以上为2spp-64spp的结果，可以看到，随着噪点降低，地面上车的倒影越发明显。这个效果得益于Cook-Torrance模型对反射光较好的表现效果，从以下CarDay场景的对比出，更易看到其比Phong Shading的优势所在。

3. CarDay:

400x400 Phong Shading和Cook-Torrance对比：



上图第一列为Cook-Torrance的效果，下图为Phong的效果，从左到右依次为4-32spp。可以看到，Cook-Torrance对地面的反光有更好的表现；其次，Phong模型车的RGB值更高，原因是Lobe集中在Diffuse。同数量spp的情况下，使用Cook-Torrance模型噪点明显更少。

800x800 Final Result:



4. DiningRoom

800x800 Cook-Torrance Shading:



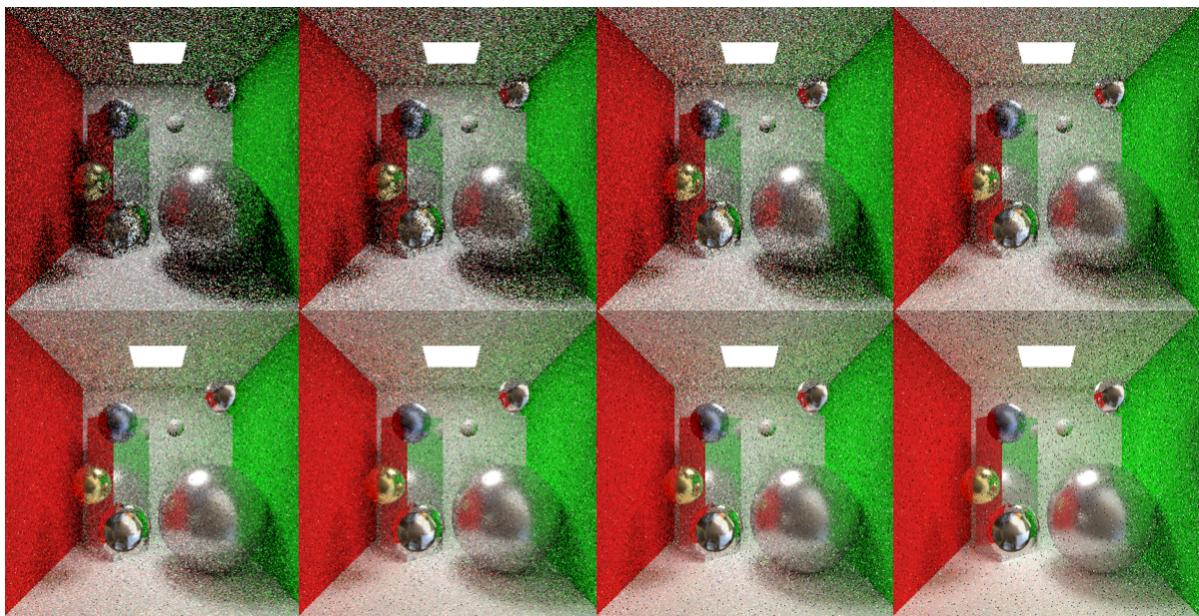


可以看到，在较高spp(16、32)下，对金属质感的花瓶、玻璃材质的桌子都有比较好的效果。

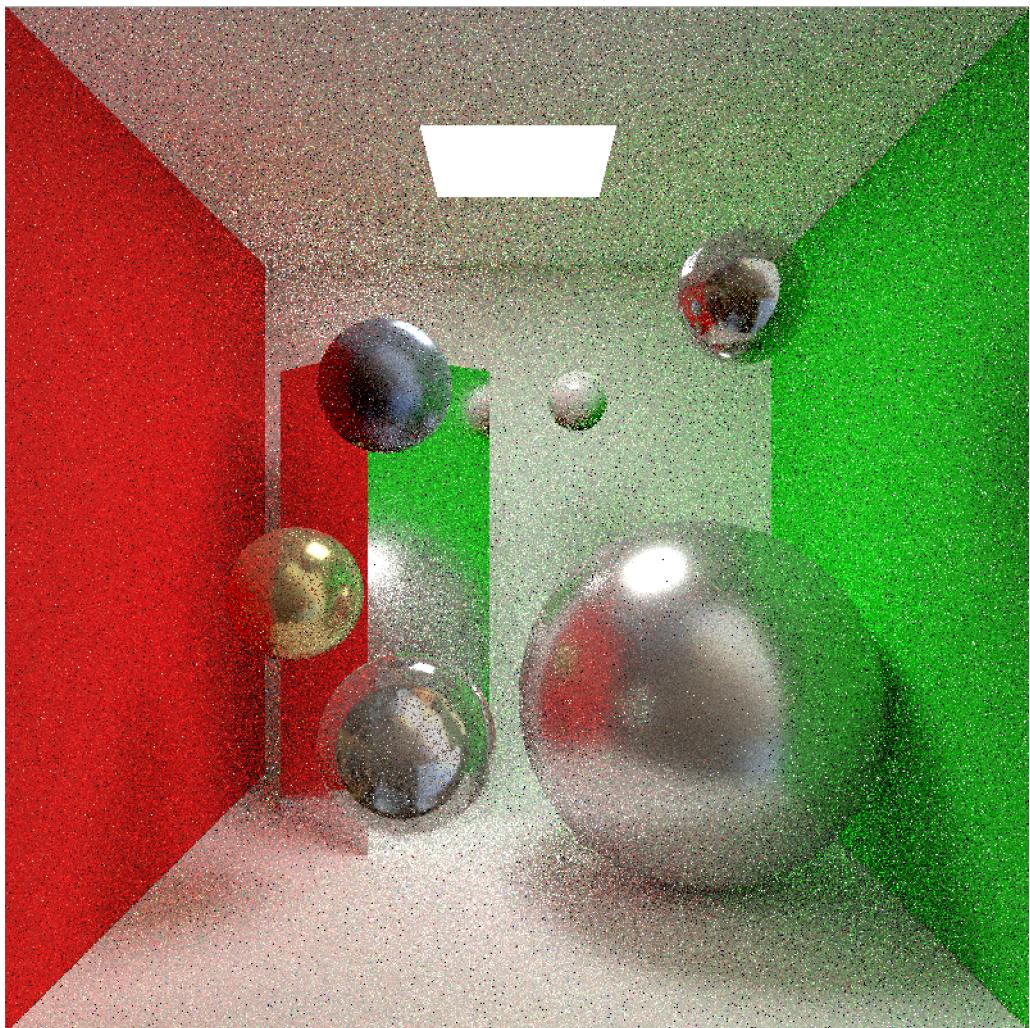
5. Newscene:

为了验证Cook-Torrance实现的正确性，我在cornellbox场景中新加了四个不同材质的球，看看效果：

400x400 Cook-Torrance Shading：



800x800 Final Result:



从左到右几个球的材质依次为:

```
Kd 0.9 0.75 0.3  
Roughness 0.2  
Mentalness 0.6
```

```
Kd 0.5 0.6 0.9
```

```
Roughness 0.3
```

```
Mentalness 0.4
```

```
Kd 1 1 1
```

```
Roughness 0.1
```

```
Mentalness 0.5
```

```
Kd 1 1 1
```

```
Roughness 0.3
```

```
Mentalness 0.6
```

```
Kd 1 1 1
```

```
Roughness 0.1
```

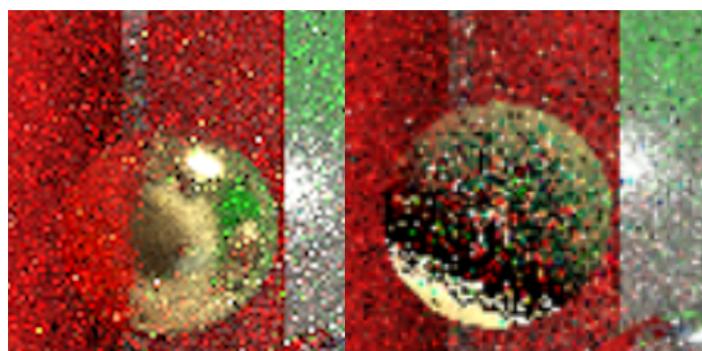
```
Mentalness 0.1
```

```
Kd 1 1 1
```

```
Roughness 0.1
```

```
Mentalness 0.5
```

可以看到，Roughness值越大，高光的范围越大，Mentalness值越大，材质越容易反射Specular光线。同时，我还发现了一个有趣的现象，即当Mentalness、Roughness都很高时，会产生极大的噪点。我们将金色球的Roughness、Mentalness都设为0.9，与原结果（同为32spp）进行对比：



原因可以解释为，两个值都很高，代表材质每一点都更倾向于反射Specular Ray，而且Specular Ray的Lobe覆盖角度很大，高强度的Radiance打向四面八方，也引起来高强度的噪声。

- Acceleration Structure Evaluation:

本来我只实现了3DDDA，但3DDDA是将场景平均分成格子，不能反映出场景的特点。每个格子都要记录包含的几何体的信息，在内存一定的情况下，加速效果不佳。渲染Dining Room场景时发现用3DDDA一帧要30分钟...无奈实现了BVH，并做了相关评测（对于Dining Room场景,600x600）：

	3DDDA	BVH(SAH_NUM=0)	BVH(SAH_NUM=2000)
时间	38min/frame	4min21s/frame	2min12s/frame
内存	8.7G	624.2M	625.4M

- Future Work:

- 移植到GPU，增加并行性。

采用并行编程，多个sample可以并行计算，提高效率。同时构建BVH也可以应用并行的方法。

- Adaptive SPP

采用不同的像素打不同SPP值的方法，改善难以收敛的区域的渲染效果。比如对于Roughness和Metalness都很高的空间区域、或是Caustics的空间区域，投更多sample，而对于Diffuse占主要的、光线情况简单的区域，则分配更少sample。

CornellBox的Reference图片左上角，有一块高光区域，由Diffuse-Specular-Diffuse的光路产生，类似于Caustics。我在做图片时没有做出来，勉强能看出来一点，但非常不明显，如果在这个区域投更多sample，相信可以改善。

- 其他算法用于降低图片噪点

如采用BDPT、Metropolis等等。

- 感想：

诚实来讲，这个作业还是非常难做的...想要出个差不多的效果比较容易，但要出物理上正确的、噪点较少的图片其实还是非常难的。我做作业时主要卡在做Multiple Importance Sampling的阶段，一方面不知道怎么采样不同的Lobe，另一面不知道采完后怎样结合在物理上才是正确的。非常感谢组里学长和好书pbr的帮助...作为一名本科生，基本从零开始搭建这样一个引擎，难度确实挺高的。不过收获颇丰，也认识到了自己的不足，有了更强的学习动力。