《高等计算机图形学》

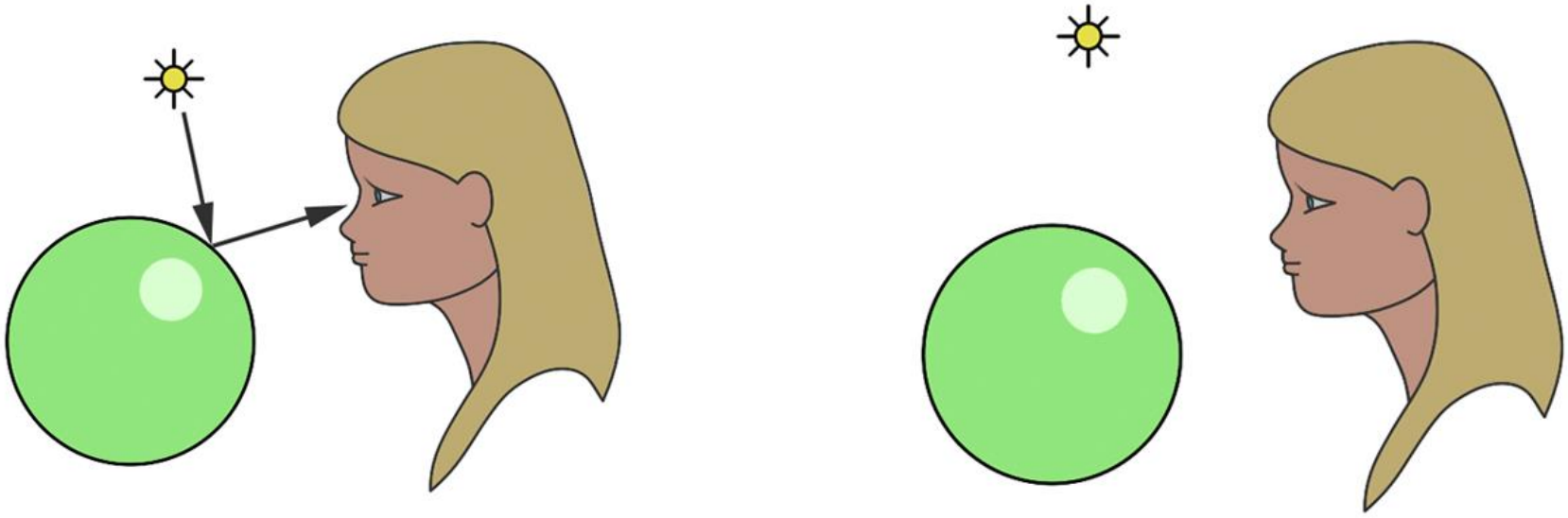
习题课2

助教　杨国烨

2022年3月27日

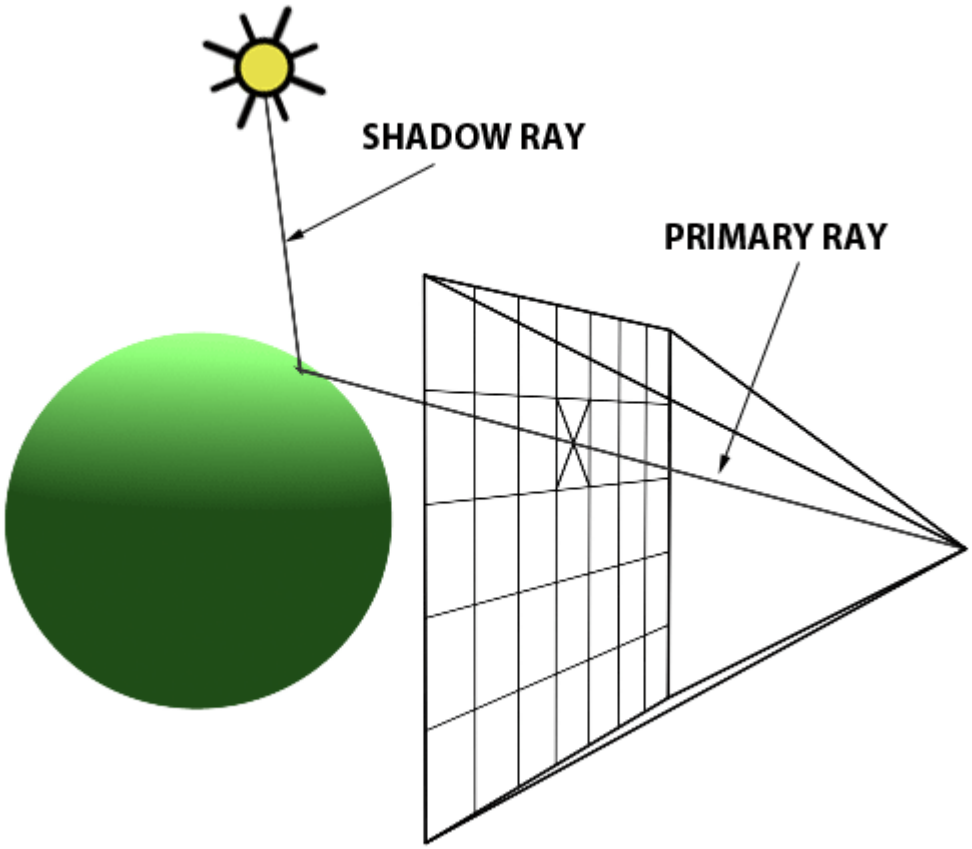
PA1：光线投射

光线投射



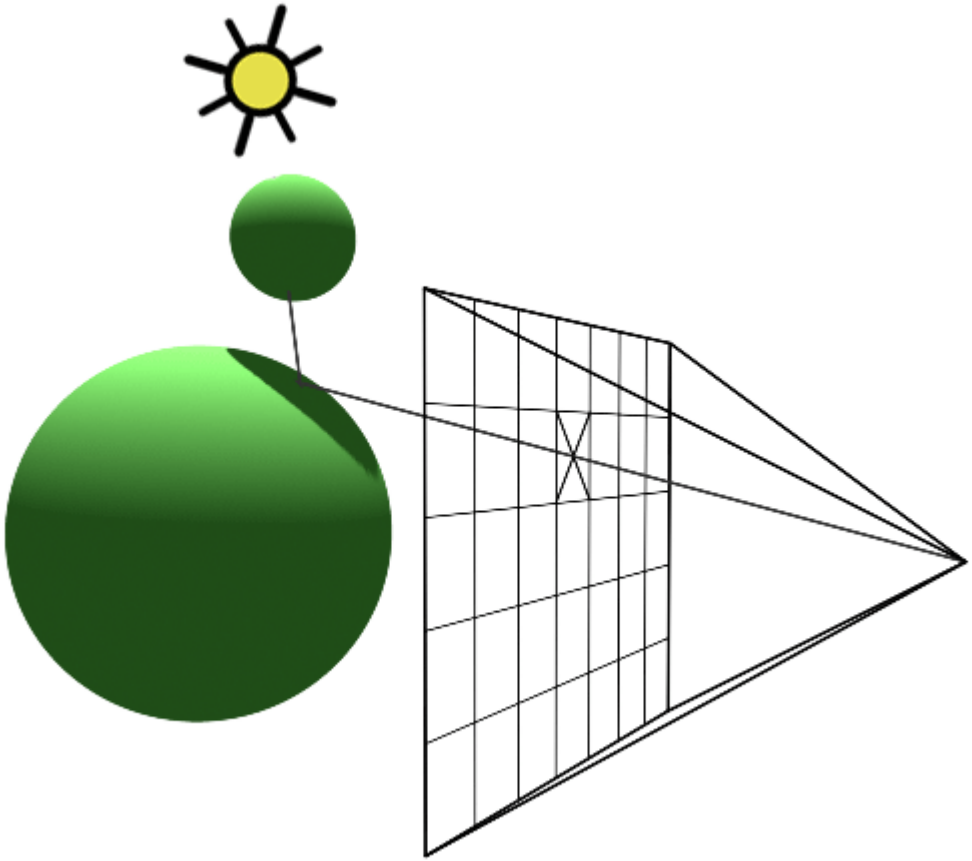
光线投射

- 从视点出发逆向追踪光路



光线投射

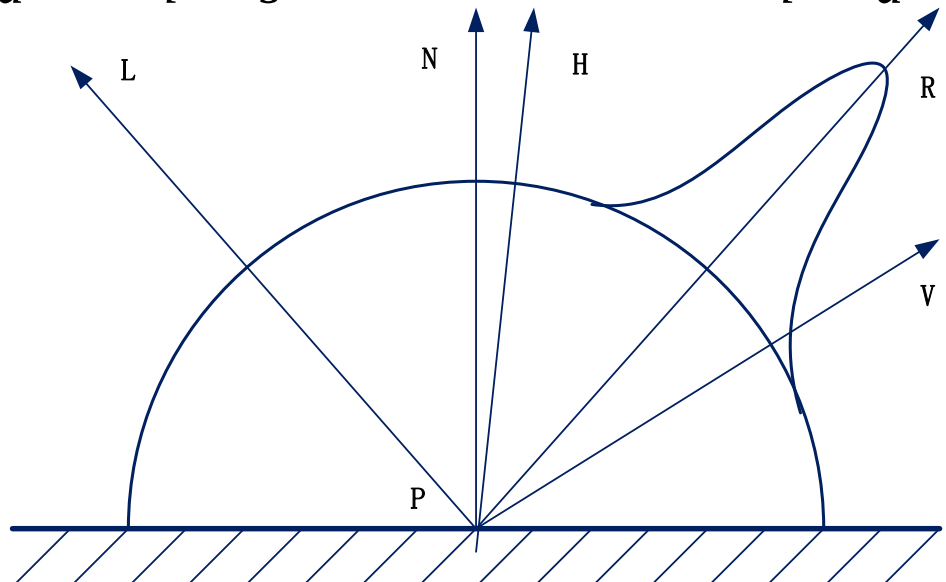
- 只计算光源对交点的直接贡献
- 作业中不考虑光源被遮挡的问题



光线投射

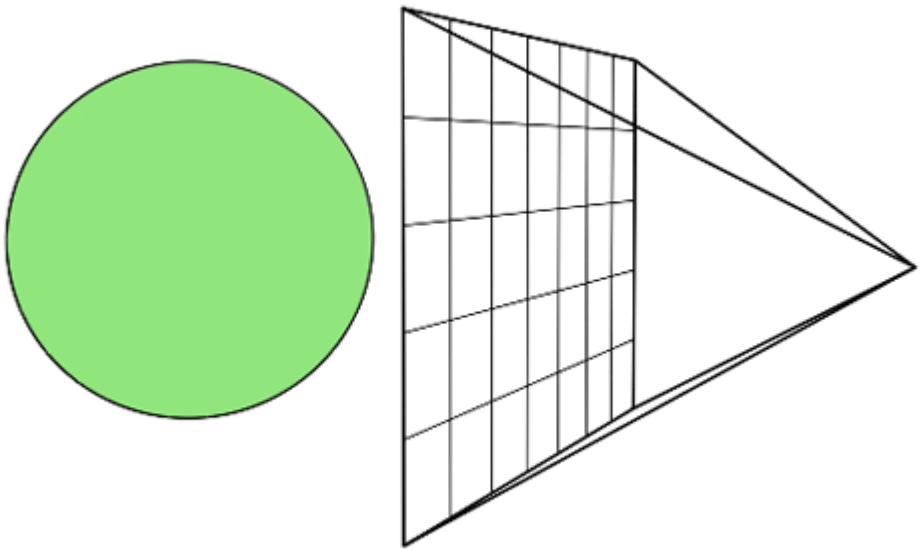
- Phong模型
- 视角方向接收的发光强度为环境光分量、镜面反射光分量和漫反射光分量之和
- 这次作业不要求实现环境光

$$I = I_i K_a + I_i K_s * (R \cdot V)^n + I_i K_d * (L \cdot N)$$



光线投射

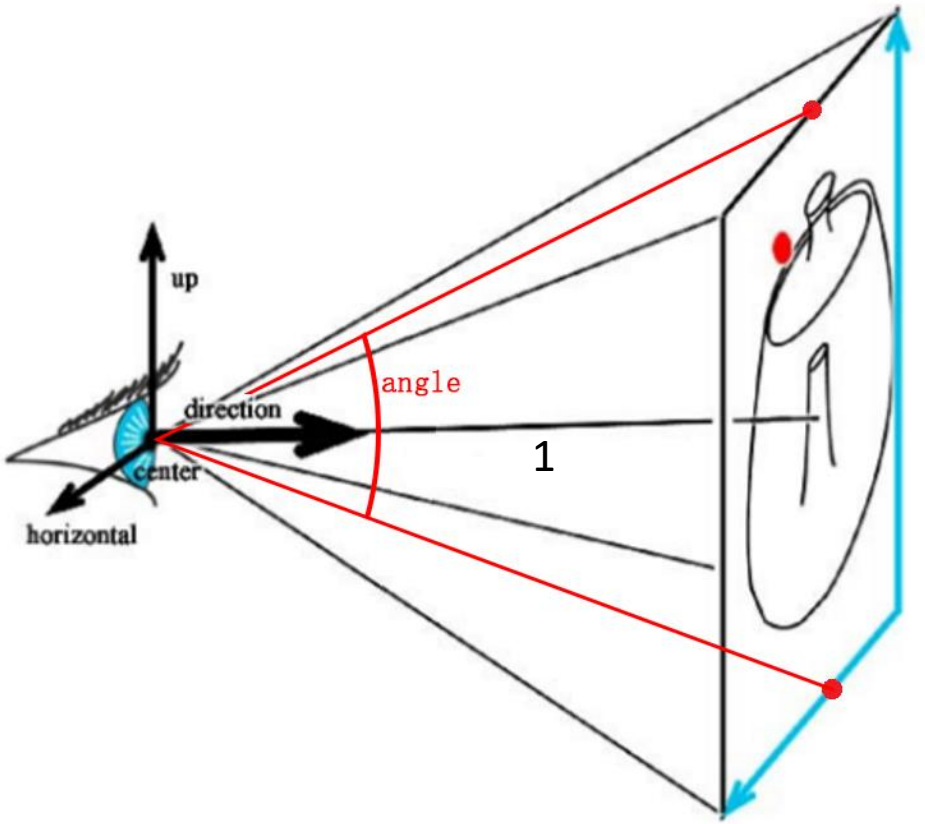
- 扫描所有像素，求交点并通过Phong模型求出对应颜色
- 没有交点的位置设为背景色



透视相机

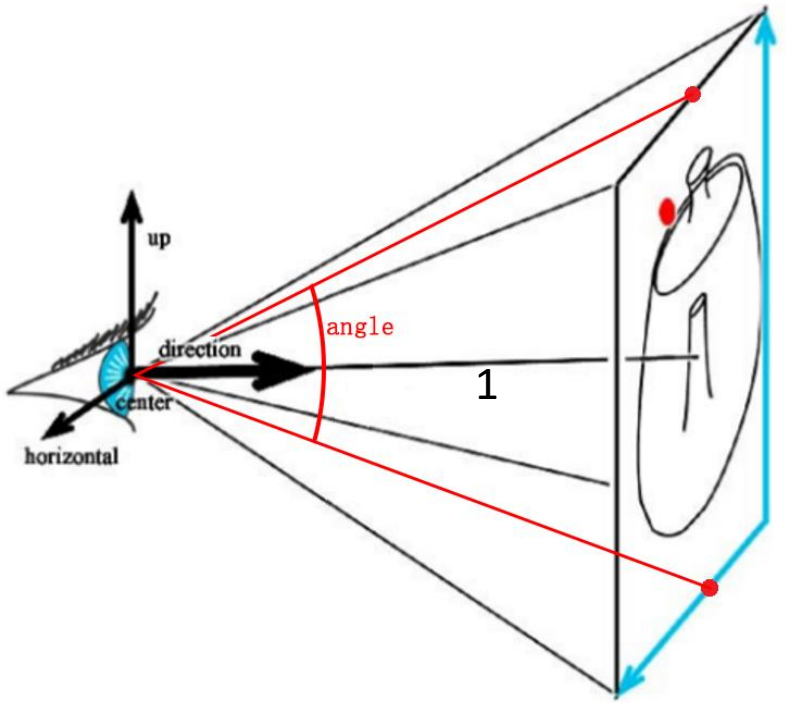
透视相机

- 在视点前放置一块画布，划分成h*w的均匀网格

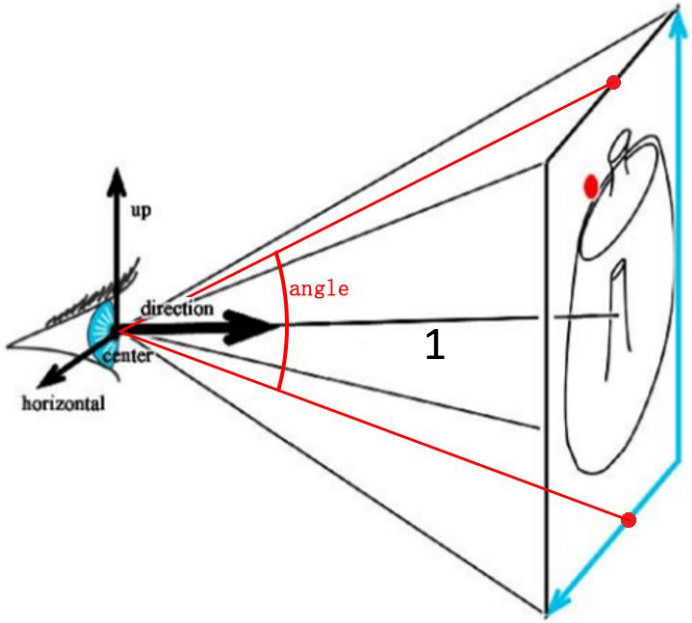


透视相机

- 相机外参：
  - 视点位置t
  - 视点朝向（指向光心）direction
  - 画布的水平轴horizontal和竖直轴up



透视相机



- 相机内参：
  - 画布（图像）大小(w, h)
  - 光心位置(c_x, c_y)，一般为(w/2, h/2)
  - 视场角$angle$
- 给定图像坐标(u, v)，求视线方向$\overrightarrow{d_{R_c}}$：
  - f_x, f_y表示真实空间中画布上1单位距离对应图像中的像素数

$$f_x = f_y = \frac{h}{2 * \tan\left(\frac{angle}{2}\right)}$$

$$\overrightarrow{d_{R_c}} = \text{normalized}\left(\frac{u - c_x}{f_x}, \frac{c_y - v}{f_y}, 1\right)^\top$$

代码讲解

代码讲解

- vecmath库
- 主要用到的是Vector3f类
- 重载了+-*/和[]访问
- 一些常用的函数

```
float length() const;
float squaredLength() const;

void normalize();
Vector3f normalized() const;

Vector2f homogenized() const;

void negate();

// ---- Utility ----
operator const float* () const; // automatic type conversion for OpenGL
operator float* (); // automatic type conversion for OpenGL
void print() const;

Vector3f& operator += ( const Vector3f& v );
Vector3f& operator -= ( const Vector3f& v );
Vector3f& operator *= ( float f );

static float dot( const Vector3f& v0, const Vector3f& v1 );
static Vector3f cross( const Vector3f& v0, const Vector3f& v1 );
```

代码讲解

- Image类
- 数组data存储颜色值

```
const Vector3f &GetPixel(int x, int y) const {
    assert(x >= 0 && x < width);
    assert(y >= 0 && y < height);
    return data[y * width + x];
}

void SetAllPixels(const Vector3f &color) {
    for (int i = 0; i < width * height; ++i) {
        data[i] = color;
    }
}

void SetPixel(int x, int y, const Vector3f &color) {
    assert(x >= 0 && x < width);
    assert(y >= 0 && y < height);
    data[y * width + x] = color;
}
```

代码讲解

- Object3D类
- 所有物体类型的基类
- 需要在派生类中实现intersect求交函数

```
// Base class for all 3d entities.
class Object3D {
public:
    Object3D() : material(nullptr) {}

    virtual ~Object3D() = default;

    explicit Object3D(Material *material) {
        this->material = material;
    }

    // Intersect Ray with this object. If hit, store information in hit structure.
    virtual bool intersect(const Ray &r, Hit &h, float tmin) = 0;
protected:

    Material *material;
};
```

代码讲解

- 需要在派生类中实现intersect求交函数：
  - r表示视线射线，包含原点和方向
  - h用于返回求交结果，返回(t, 材质, 法向)
  - tmin用于避免返回视线后方的交点（及光追反射时的误差）

```
bool intersect(const Ray &r, Hit &h, float tmin) override {
    float t = ...;
    if (t > tmin && t < h.getT()) {
        h.set(t, material, norm);
        return true;
    } else
        return false;
}
```

代码讲解

- Material类
- 定义了材质属性
- 需要根据Phong模型实现Shade函数

```
Vector3f Shade(const Ray &ray, const Hit &hit,
               const Vector3f &dirToLight, const Vector3f &lightColor) {
    Vector3f shaded = Vector3f::ZERO;
    //
    return shaded;
}

protected:
    Vector3f diffuseColor;
    Vector3f specularColor;
    float shininess;
```

代码讲解

- 根据Phong模型实现Shade函数：
  - ray表示视线射线
  - hit表示交点，含有法向等信息
  - dirToLight表示光线向量
  - lightColor表示光线颜色
  - 返回视线观察到的颜色值

```
Vector3f Shade(const Ray &ray, const Hit &hit,
               const Vector3f &dirToLight, const Vector3f &lightColor) {
    Vector3f shaded = Vector3f::ZERO;
    //
    return shaded;
}
```

代码讲解

- SceneParser类，读取输入的场景文件和模型
- 场景文件（scene*.txt）和模型文件（*.obj）

```
Materials {
    numMaterials 1
    PhongMaterial {
        diffuseColor 0.79 0.66 0.44
    specularColor 1 1 1
    shininess 20

    }
}

Group {
    numObjects 1

    MaterialIndex 0
    TriangleMesh {
        obj_file mesh/bunny_200.obj
    }
}
```

```
v -1 -1 -1
v 1 -1 -1
v -1 1 -1
v 1 1 -1
v -1 -1 1
v 1 -1 1
v -1 1 1
v 1 1 1
f 1 3 4
f 1 4 2
f 5 6 8
f 5 8 7
f 1 2 6
```

代码讲解

- 对经过transform的物体求交
- 对一般的物体，我们希望求到视线$\vec{o}+t\vec{d}$与物体的交点$\vec{x}$，即$\vec{x}=\vec{o}+t\vec{d}$
- 现在我们对物体上每个点做一个仿射变换，即$\vec{x}\rightarrow A\vec{x}+\vec{b}$

代码讲解

- 对经过transform的物体求交
- 问题转变为求$A\vec{x}+\vec{b}=\vec{o}+t\vec{d}$
- 实际上我们不需要真的对物体上每个点进行变换，因为上述式子可写成

$$\vec{x}=\left(A^{-1}\vec{o}-A^{-1}\vec{b}\right)+t(A^{-1}\vec{d})$$

- 换言之，我们只需要把视线起点改为$A^{-1}\vec{o}-A^{-1}\vec{b}$，方向改为$A^{-1}\vec{d}$对原物体求交

代码讲解

- 在Transform类中，我们正是对视线施加了逆变换来计算交点的。
- 注意，对视线施加了逆变换后，其方向向量**不一定是单位向量**，不能直接对其单位化，需要求这个非单位射线的对应t值：
- $\vec{x} = \left(A^{-1}\vec{o} - A^{-1}\vec{b}\right) + t(A^{-1}\vec{d})$
- 可以单位化求交后将t除以原向量长度。

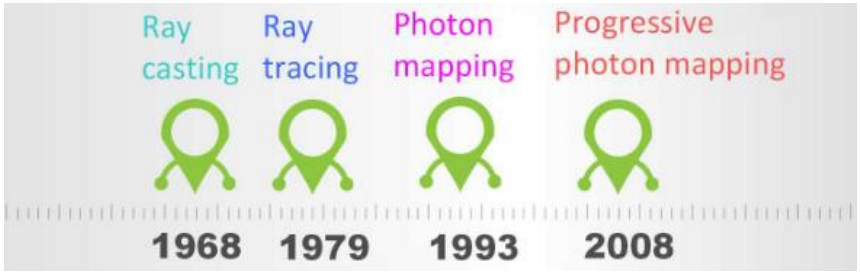
大作业说明

主要考核点

- 基本：实现带有反射、折射、支持三角网格模型导入的光线追踪引擎
- 其他：参数曲面的渲染（Bezier曲面，B样条曲面），光子映射，分布式光线跟踪（景深、运动模糊），算法加速，体积光等等

光线跟踪类别

- 光线投射(Ray casting)　光线跟踪(Ray Tracing)
- 路径追踪(Monte Carlo Ray Tracing / Path Tracing)
- 分布式光线跟踪(Distributed Ray Tracing)
- 光子映射(Photon mapping)
- 渐进式光子映射(Progressive Photon mapping)
- 随机渐进式光子映射(Stochastic Progressive Photon Mapping)

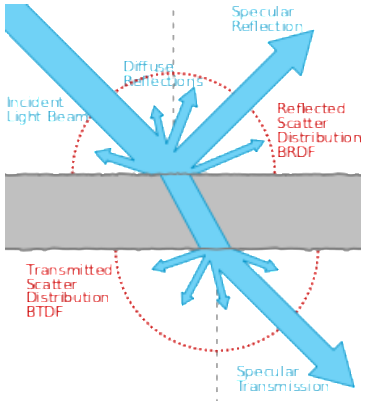


经典的光线跟踪算法(单点光源)

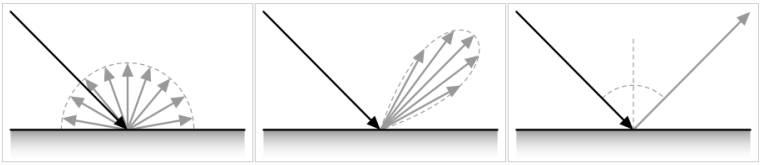
```
RayTracer(Ray, Depth, weight, &Color) {
  Color = 0;
  if ( weight衰减到小于给定值 ) return;  // 递归终止：给定衰减阈值.
  Color = Background;
  计算Ray与场景中最近的物体的交点P;          // 可以采用多种加速方法.
  if ( 没有交点 ) return;                  // 递归终止：返回背景色.
  if ( P非阴影点 )                         // P与光源间是否有物体阻挡.
    Color = 局部光强;                      // 如：书P149，局部光强计算公式.
  if ( Depth > 1 ) {                       // 递归终止：一定次数.
    if ( 当前面是镜面 ) {
      计算反射光线ReflectedRay;
      RayTracer(ReflectedRay, Depth-1, weight*w_r, &RefColor);
      Color += w_r * RefColor;
    }
    if ( 当前面是透射面 ) {
      计算透射光线TransmittedRay;
      RayTracer(TransmittedRay, Depth-1, weight*w_t, &TransColor);
      Color += w_t * TransColor;
    }
  }
}
```

其他算法选择

- Path Tracing
  - Color Bleeding
  - Physically Unbiased



- Distributed Ray Tracing
  - AA, Depth, Glossy, Motion Blur

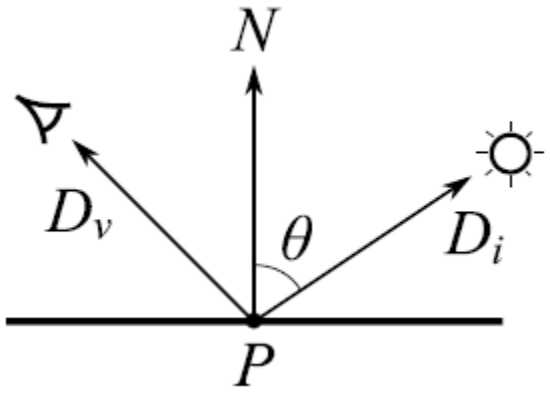


- Photon Mapping
  - Caustics

Path Tracing

- http://www.kevinbeason.com/smallpt/
- 路径追踪不在漫反射表面终止，而是在击中光源时终止
- 光线击中漫反射表面后随机选择出射方向，根据BRDF函数近似计算亮度积分

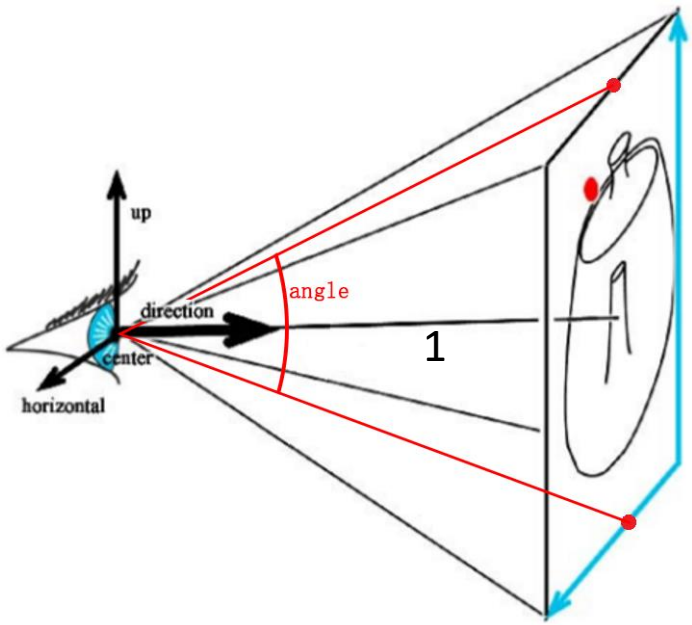
Render Equation



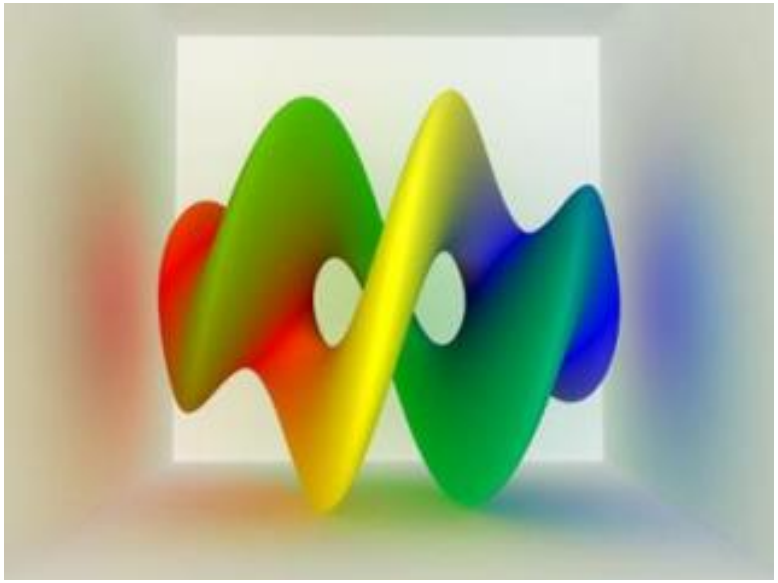
$$L(P \to D_v) = L_e(P \to D_v) + \int_\Omega F_s(D_v, D_i) |\cos\theta| L(Y_i \to -D_i)\, dD_i$$

Path Tracing Algorithm

```
1:  for each pixel (i,j) do
2:      Vec3 C = 0
3:      for (k=0; k < samplesPerPixel; k++) do
4:          Create random ray in pixel:
5:              Choose random point on lens P_lens
6:              Choose random point on image plane P_image
7:              D = normalize(P_image − P_lens)
8:              Ray ray = Ray(P_lens, D)
9:          castRay(ray, isect)
10:         if the ray hits something then
11:             C += radiance(ray, isect, 0)
12:         else
13:             C += backgroundColor(D)
14:         end if
15:     end for
16:     image(i,j) = C / samplesPerPixel
17: end for
```



Path Tracing



Photon Mapping

- http://graphics.ucsd.edu/~henrik/papers/
- 从光源出发发射一定数量的光子（photon tracing），在漫反射表面吸收
- 从视点出发追踪光路（ray tracing），收集终点处一个邻域内的光子以估计该点颜色

Photon Mapping: Construct Photon Map

- 传播
  1. 反射
  2. 折射
  3. 漫反射: 按照 BRDF (对于 Lambert 反射体就是 cos) 的概率分布随机选一个方向
  4. 每次碰到漫反射物体储存位置, 方向跟颜色
  5. 按照概率在反射, 折射, 漫反射以及被吸收中随机选一种.

Photon Mapping: Rendering

- 光线追踪

进行正常的光线跟踪. 但是碰到漫反射物体时, 使用光子图来计算颜色: 寻找碰撞点邻域内的光子, 根据 BRDF 计算这些光子产生的平均色光.

$$L_r(x,\omega) = \frac{1}{\pi r^2}\sum_{p=1}^{N} f(x,\omega_p,\omega)\Delta\Phi_p(x,\omega_p)$$

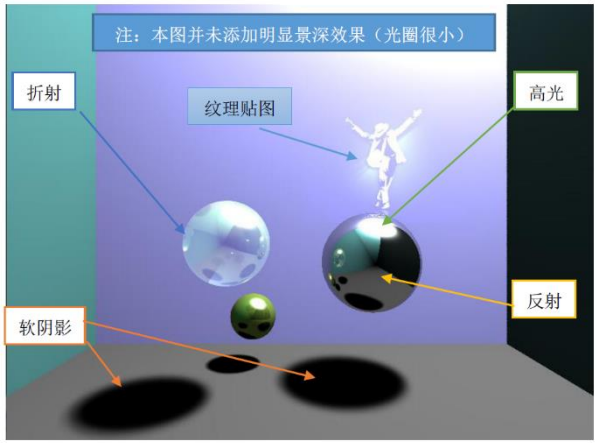
Photon Mapping

- 储存 photon mapping 的数据结构: kd 树 (其他空间数据结构也可)
- 如果场景中漫反射表面特别少, 光子一直反射怎么办: 设置最大撞击次数, 或者俄罗斯轮盘赌
- 需要很多的光子才能达到足够好的效果: Progressive photon mapping (PPM) 和 Stochastic progressive photon mapping (SPPM)

Photon Mapping

要求

- 按时完成，可查阅开源资料和往届文档，不允许裸抄
- CPU上实现
- 至少实现光线跟踪（反射+折射+阴影）
- 图片需要大于480P （640x480），建议使用无损png等格式进行存储。
  - 不建议在多张图上分别实现多个效果
- 严禁使用PhotoShop及其他画图工具进行后处理。
- 不要在最终结果上疯狂注释。
- 不要构造难以辨认的场景。
  - 例如使用纹理贴图贴一张光线跟踪结果。



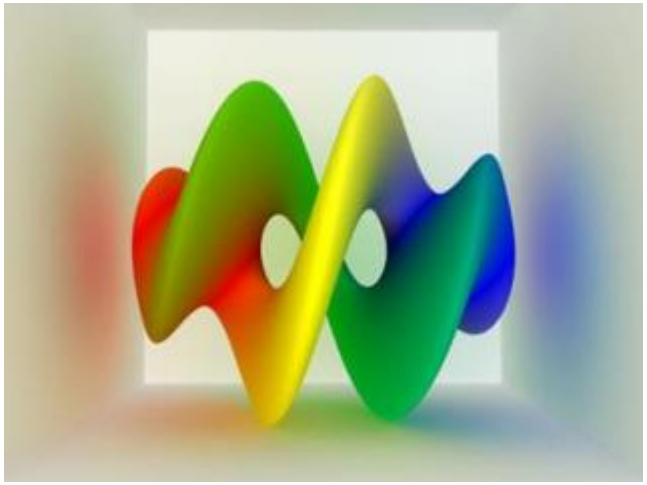
得分项总述

- 算法选型（RT，PT，PM，PPM，……）

- 求交加速（包围盒、层次包围盒、kd-tree、octree、hash）
- 抗锯齿（边缘超采样、DRT等方法，主要针对模型边缘）
- 景深（Adaptive Blur、DRT等方法，模拟相机焦距）
- 软阴影（面光源采样）
- 纹理贴图（UV纹理映射、凹凸贴图）
- 复杂网格模型/场景（网格读写、求交加速、法向插值等）
- 其他（体积光等）

1. 算法选型

- 仅需实现一种渲染法。
  - 算法难度和运算量逐步提高。
  - 如果选择了较新的算法，应该体现出比老算法强在哪里。
    - 做Path Tracing： 漫反射，特殊材质下的Color Bleeding。
    - 做Photon mapping：Caustics – 由反射/透射引起的漫反射。
    - 做Progressive Photon mapping ： 高精准度。



2. 时间代价

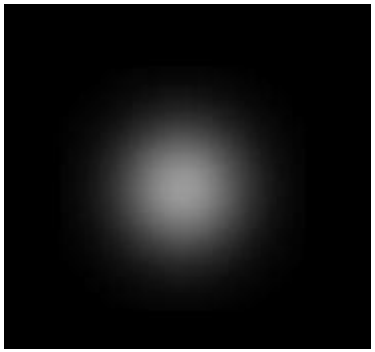
- 最终评分主要看效果，做的加速(包围盒、包围球、Octree、Kd-tree)请在报告中注明。

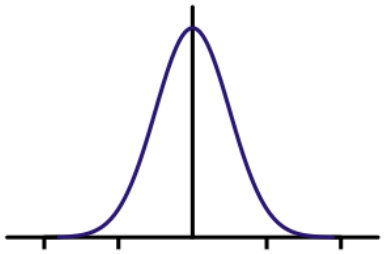
- 参数曲面（尤其是高次）和复杂网格求交的时间代价比一般曲面要高很多，完成作业时注意留足渲染时间。

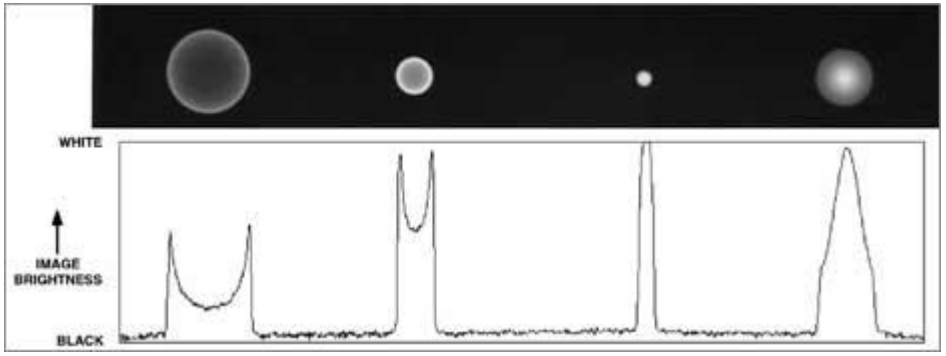
- 多项附加功能会导致渲染时间乘法式叠加，夜里挂程序请保护计算机的安全。

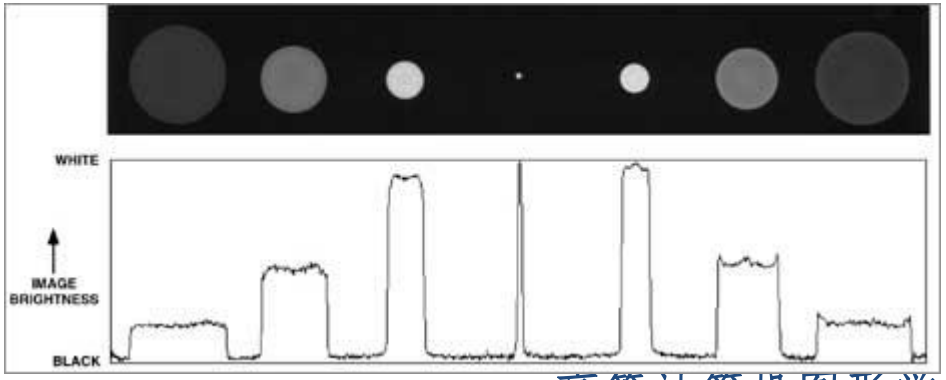
3. 景深

- 景深三要素：光圈、焦距、物距



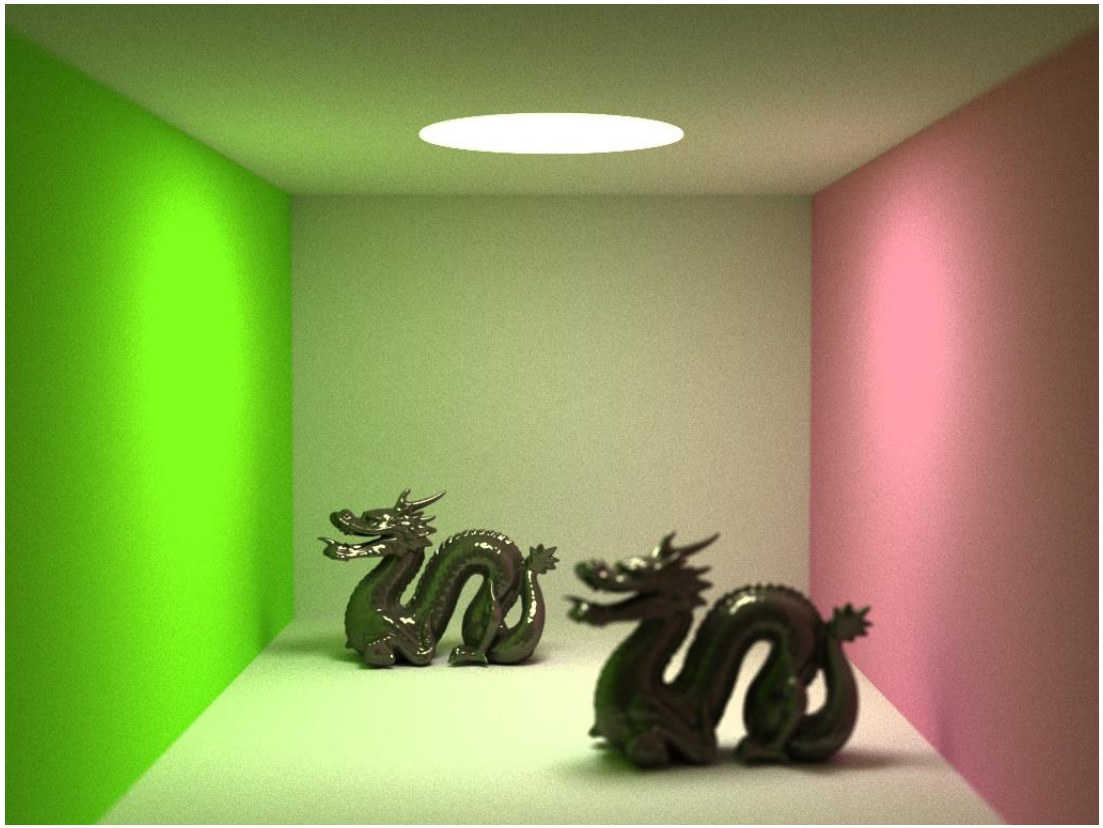






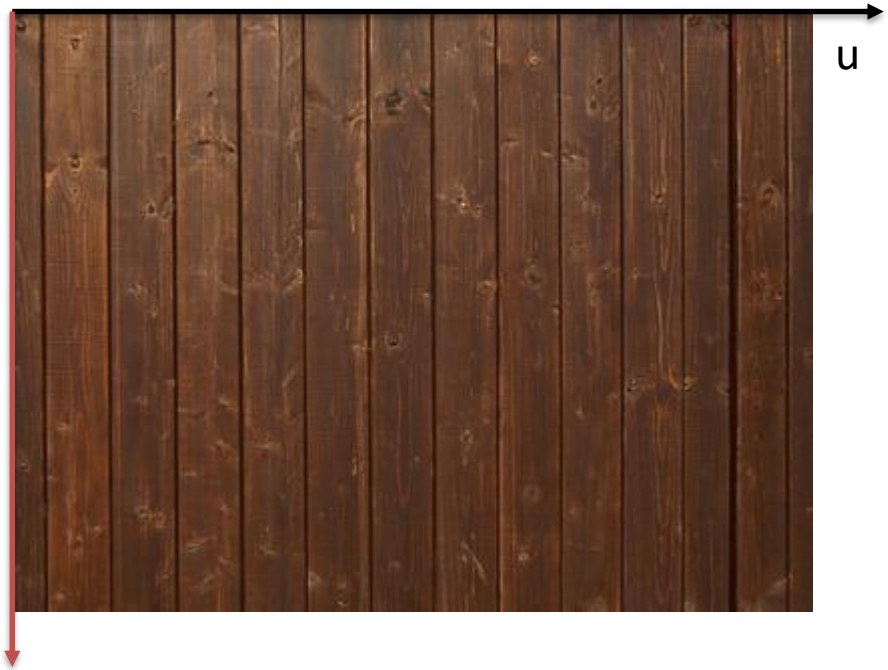
3. 景深

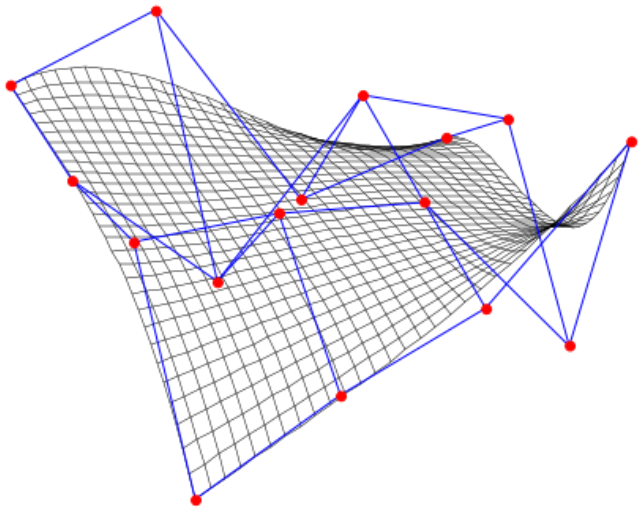
- Path Tracing带景深



4. 纹理映射

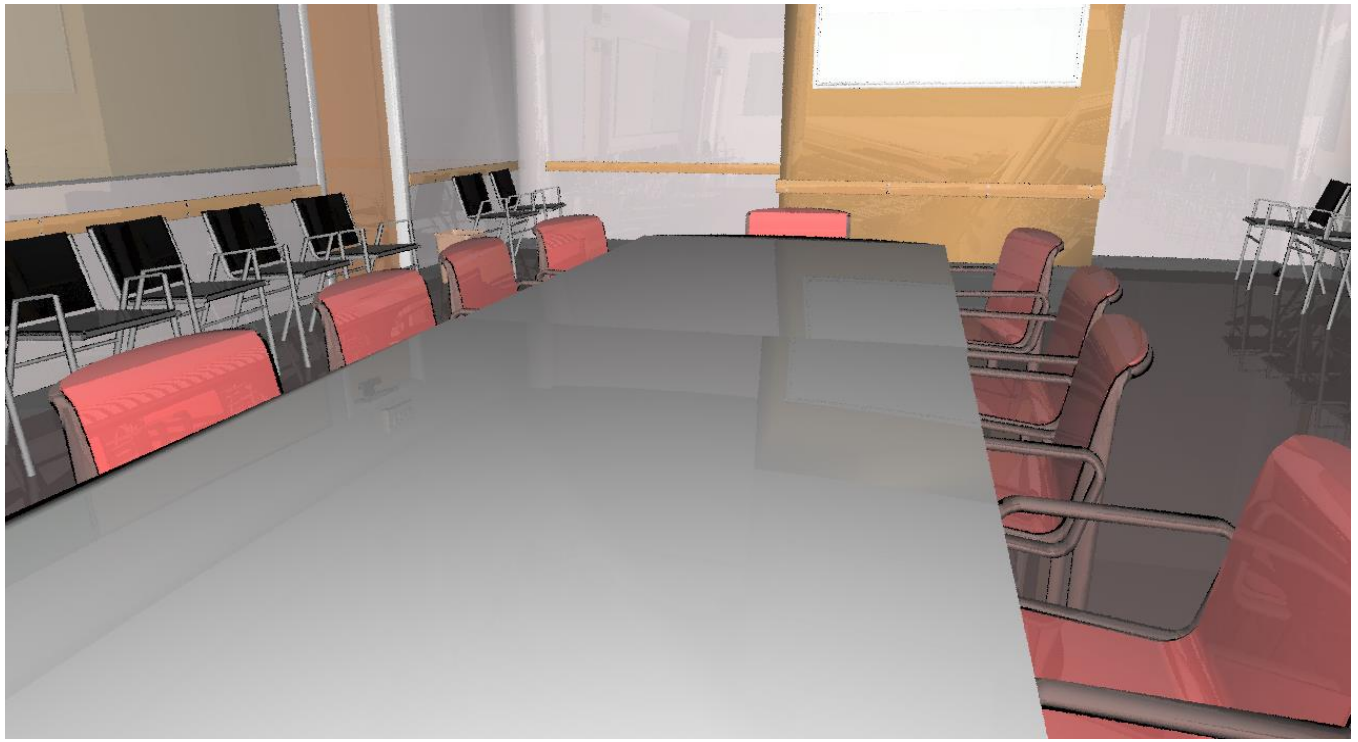
- 与参数曲面求交时，UV纹理坐标和求出来的参数值完美对应。
- 一般的面片，也可以将求交结果转化为参数坐标后读出需要取的纹理颜色值。





5. 复杂网格/场景

- 建议做1~2个参数曲面或复杂网格模型即可。



X. 体积光

- 纹理与体积光

最终提交和验收

- Result = 一张/多张渲染好的图片
- Code
- Report
  - 开头应明确列出自己的所有得分点，避免遗漏。
  - 再分别列出每个得分点对应的代码段，还可以辅以说明，方便查验。
  - 报告最后再附上实验结果图片。

- 第18周工作日进行当面验收，原则上不允许远程验收和推迟。
  - 验收以交流为主，验收时用的图像不作为最终评分标准
  - 之后可以补交最终结果。
- 第18周周日晚上23:59前提交。

评分细则

- 占总评45分。
  - 实现光线追踪（反射、折射、三角网格，有明显bug扣分）
  - 实现光子映射 / 渐进式光子映射
  - 景深 / 软阴影 / 抗锯齿 / 贴图 / 凹凸贴图/运动模糊等（路径追踪得分含在内）
  - 实现参数曲面解析法求交
  - 复杂网格模型
  - 主观分: 设计和构图
  - 其他额外效果: 体积光、色散、体渲染等

- 评分
  - 实现的功能：多写不扣分
  - 功能的完成度和呈现质量：影响每项功能的具体得分
  - 整体渲染效果
  - 实验报告：按要求描述清楚即不会扣分

Thank You ！
Any Questions?