Ray Tracing

- Ray Tracing
  - Shadow Mapping
    - definition
    - idea
    - Hard Shadows vs. Soft Shadwos
      - remark
  - Ray Tracing Introduction
    - Why Ray Tracing?
    - Basic Ray-Tracing Algorithm
      - idea
      - Ray Casting
  - Recursive (Whitted-Style) Ray Tracing
    - Algorithm
    - Ray-Surface Intersection
      - remark
      - Ray Intersection with Triangle
        - plane equation
        - Moller Trumbore Algorithm
      - Accelerating Ray-Surface Intersection
        - idea
        - remark
  - Global Illumination
    - algorithm
    - remark
    - Ray Generation
      - algorithm
    - Russian Roulette
      - algorithm
      - remark
    - Sampling the Light
      - algorithm

Shadow Mapping

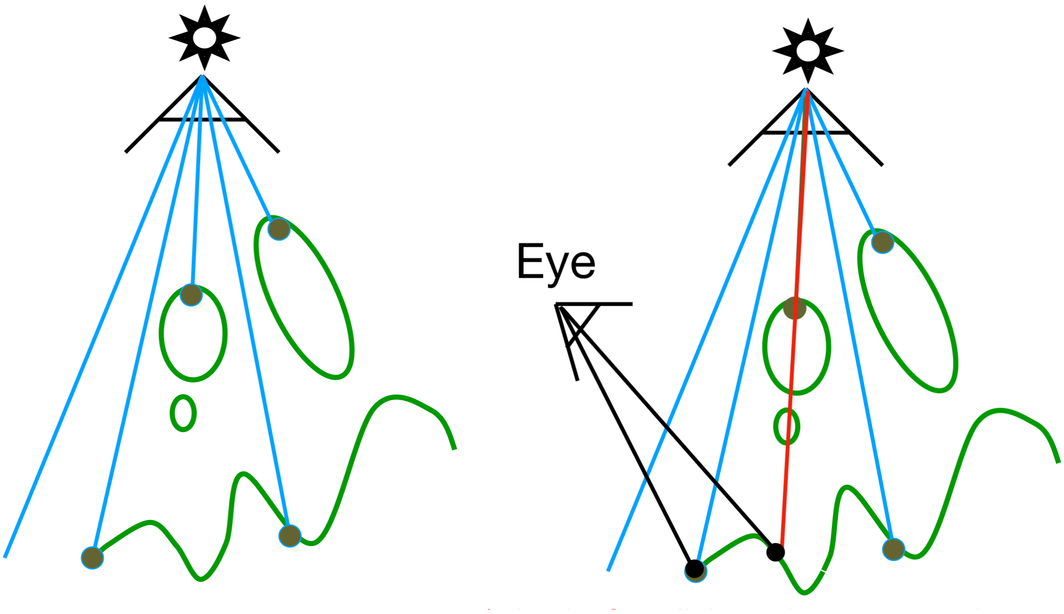
definition

an image-space Algorithm

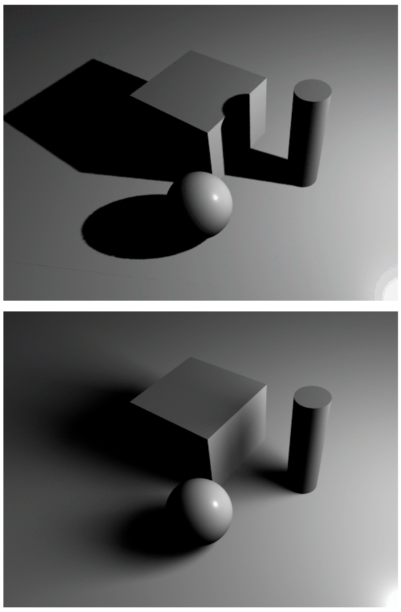
- no knowledge of scene's geometry during shadow computation
- must deal with aliasing artifacts

idea

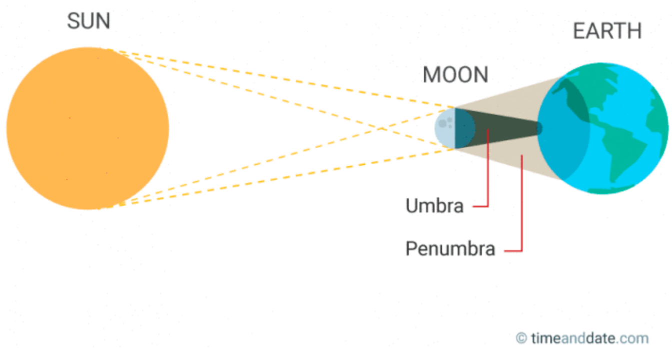
- the points NOT in shadow must be seen both by the light ans by the camera



Hard Shadows vs. Soft Shadwos



[RenderMan]



[https://www.timeanddate.com/eclipse/umbra-shadow.html]

remark

- 点光源不可能得到软阴影

Ray Tracing Introduction

Why Ray Tracing?

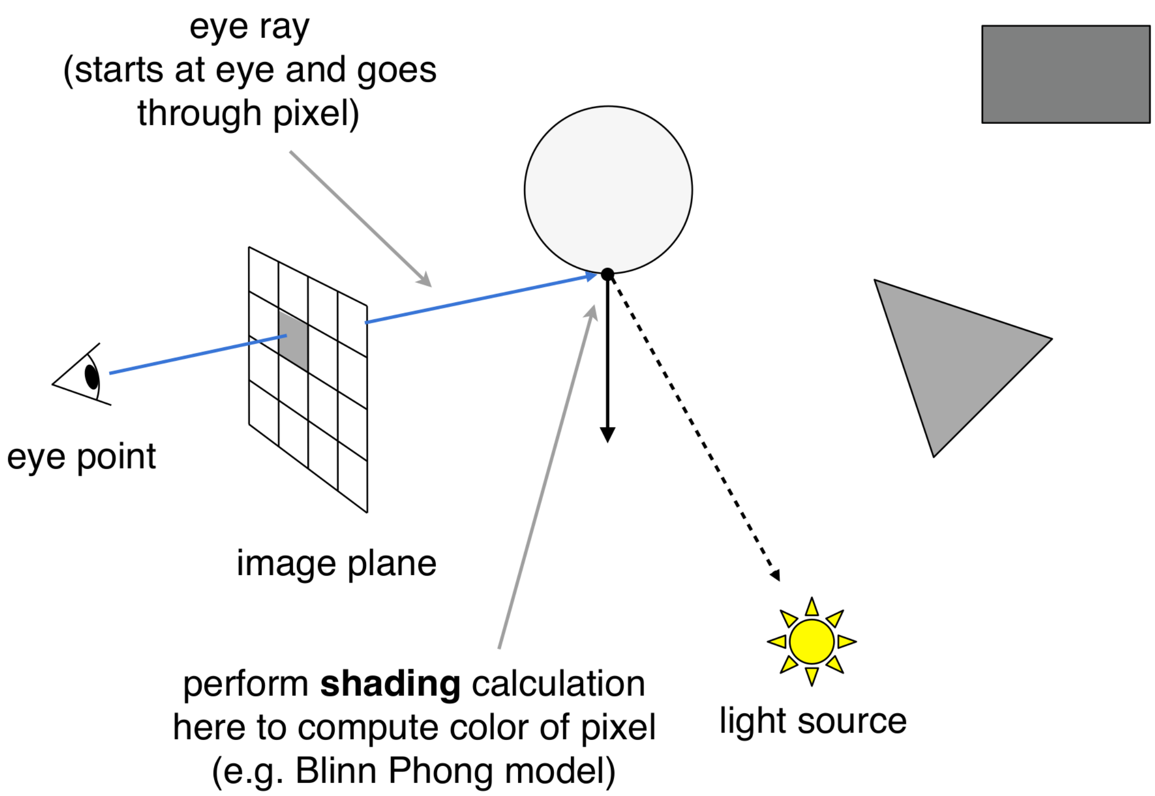
- rasterization couldn't handle global effects well
  - (soft) shadows
  - when the light bounces more than once
- rasterization is fast, but quality is relatively low
- ray tracing is accurate, but is very slow
  - rasterization: real-time, ray tracing: offline

Basic Ray-Tracing Algorithm

idea

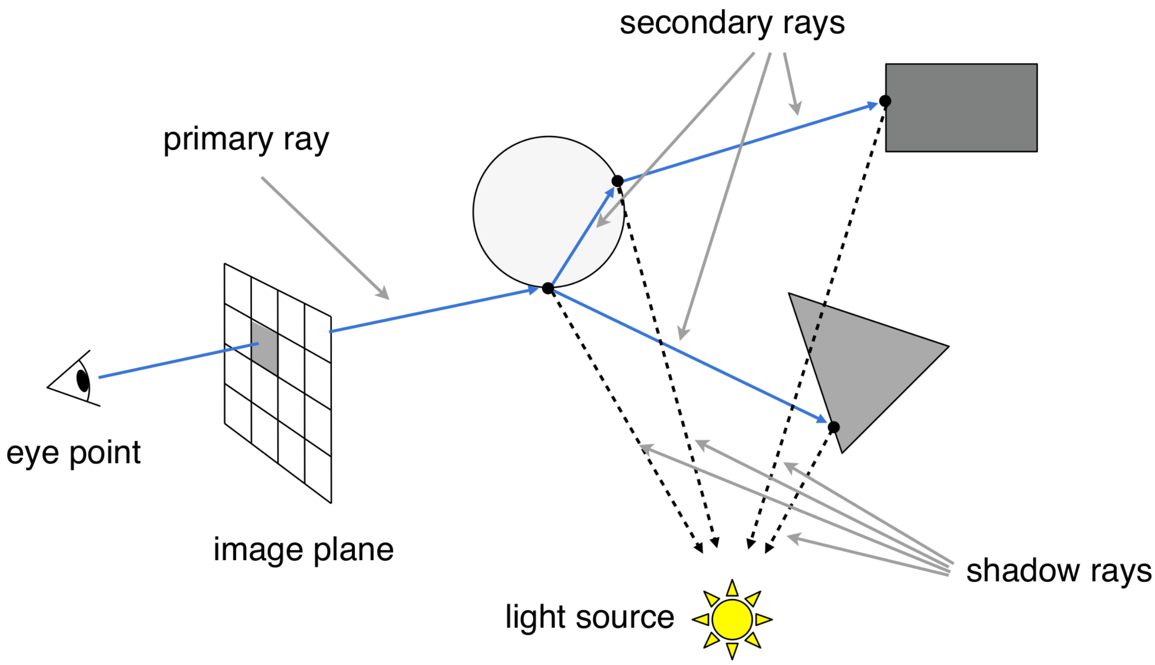
- Light travels in straight lines
- Light rays do not “collide” with each other if they cross
- Light rays travel from the light sources to the eye (but the physics is invariant under path reversal - reciprocity)

Ray Casting



Recursive (Whitted-Style) Ray Tracing

Algorithm



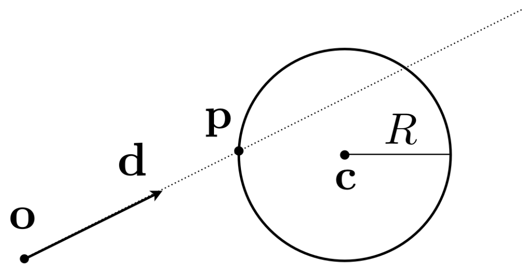
Ray-Surface Intersection

Ray: $\mathbf{r}(t) = \mathbf{o} + t\,\mathbf{d}, \ 0 \leq t < \infty$

Sphere: $\mathbf{p} : (\mathbf{p} - \mathbf{c})^2 - R^2 = 0$

What is an intersection?

The intersection p must satisfy both ray equation and sphere equation



Solve for intersection:

$$(\mathbf{o} + t\,\mathbf{d} - \mathbf{c})^2 - R^2 = 0$$

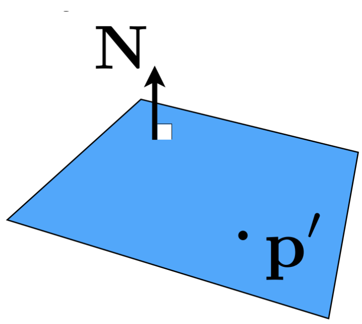
remark

- 光线和物体的交点个数是奇数时，光源在物体内部，偶数时光源在物体外部

Ray Intersection with Triangle

把问题转化为光线与平面的交点

plane equation



$\mathbf{p}:(\mathbf{p}-\mathbf{p}')\cdot\mathbf{N}=0$, where

- $\mathbf{p}$ any point on plane, can be represented by $\mathbf{o}+t\mathbf{d}$
- $\mathbf{p}'$ one point on plane
- $\mathbf{N}$ normal vector

Moller Trumbore Algorithm

$$\vec{\mathbf{O}}+t\vec{\mathbf{D}}=(1-b_1-b_2)\vec{\mathbf{P}}_0+b_1\vec{\mathbf{P}}_1+b_2\vec{\mathbf{P}}_2$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix}=\frac{1}{\vec{\mathbf{S}}_1\bullet\vec{\mathbf{E}}_1}\begin{bmatrix} \vec{\mathbf{S}}_2\bullet\vec{\mathbf{E}}_2 \\ \vec{\mathbf{S}}_1\bullet\vec{\mathbf{S}} \\ \vec{\mathbf{S}}_2\bullet\vec{\mathbf{D}} \end{bmatrix}$$

Where:

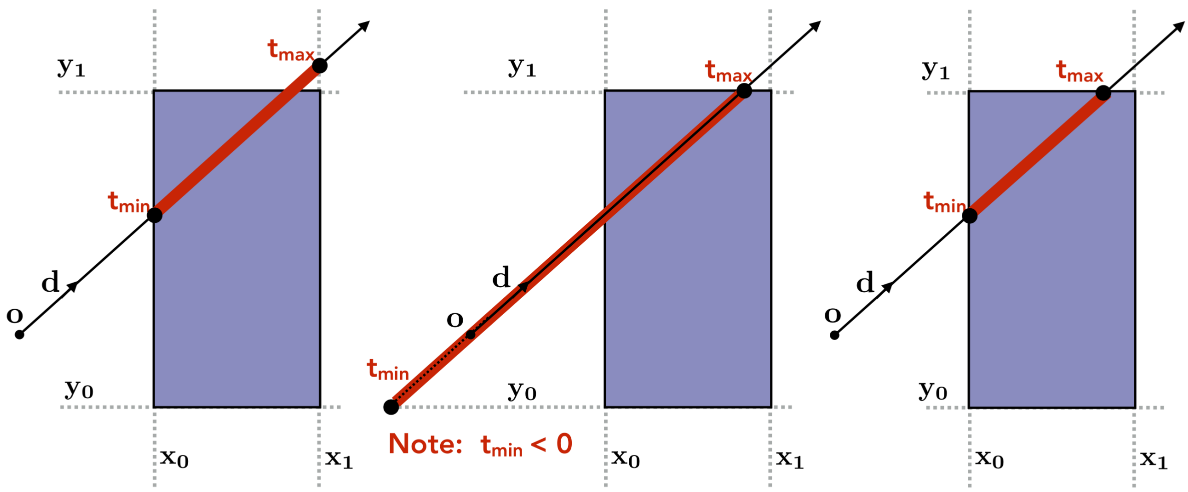
$$\vec{\mathbf{E}}_1=\vec{\mathbf{P}}_1-\vec{\mathbf{P}}_0$$
$$\vec{\mathbf{E}}_2=\vec{\mathbf{P}}_2-\vec{\mathbf{P}}_0$$
$$\vec{\mathbf{S}}=\vec{\mathbf{O}}-\vec{\mathbf{P}}_0$$
$$\vec{\mathbf{S}}_1=\vec{\mathbf{D}}\times\vec{\mathbf{E}}_2$$
$$\vec{\mathbf{S}}_2=\vec{\mathbf{S}}\times\vec{\mathbf{E}}_1$$

Cost = (1 div, 27 mul, 17 add)

Accelerating Ray-Surface Intersection

idea

using **bounding volumes**. e.g. **axis-aligned bounding box(AABB)**



Intersections with *x* planes **Intersections with *y* planes** **Final intersection result**

remark

- 3维的情况与上图类似，多算一个维度，同样取$t_{enter}=\max t_{min}$和$t_{exit}=\min t_{max}$
- $t_{exit}<0\ \rightarrow$ no intersection
- $t_{exit}>=0$ and $t_{enter}<0\rightarrow$ have intersection

Global Illumination

algorithm

```
shade(p, wo)
    randomly choose N directions wi~pdf
    Lo = 0.0
    for each wi
        trace a ray r(p,wi)
        if ray r hit the light
            Lo += (1 / N) * L_i * f_r * cosine / pdf(wi)
        else if ray hit an object at q
            Lo += (1 / N) * shade(q, -wi) * f_r * cosine / pdf(wi)
    return Lo
```

remark

- 计算量太大，为了避免指数增长，上述算法的`for`循环只循环一次($N = 1$)，此时的算法被称为==路径追踪==
- 上述解决方案噪声过大，因此选择增加单个像素中通过的"眼光"数量加以补偿
- 算法递归的终止条件用Russian Roulette（俄罗斯轮盘赌）解决

Ray Generation

algorithm

```
ray_generation(camPos,pixel)
    uniformly choose N sample positions within the pixel
    pixel_radiance = 0.0
    for each sample in the pixel
        shoot a ray r(camPos,cam_to_sample)
        if ray r hit the scene at p
            pixel_radiance += 1 / N *shade(p, sample_to_cam)
    return pixel_radiance
```

Russian Roulette

algorithm

```
shade(p, wo)
    manually specify a probaility P_RR
    Randomly select ksi in a uniform dist. in[0, 1]
    if (ksi > P_RR) return 0,0

    randomly choose 1 directions wi~pdf
    trace a ray r(p,wi)
    if ray r hit the light
        return L_i * f_r *cosine / pdf(wi) / P_RR
    else if ray hit an object at q
        return shade(q, -wi) * f_r * cosine /pdf(wi) / P_RR
```

remark

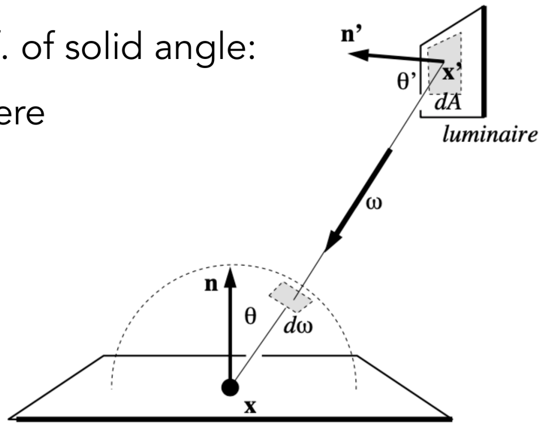
- 均匀地往四面八方采样经常会浪费光线，因此改进蒙特卡洛采样，将其写成在光源上的积分

Easy! Recall the alternative def. of solid angle:

Projected area on the unit sphere

$$d\omega = \frac{dA \cos\theta'}{\|x' - x\|^2}$$

(Note: θ’, not θ)



因此可以重写渲染方程

$L_o(x,\omega_o) = \int_{\Omega^+} L_i(x,\omega_i) f_r(x,\omega_i,\omega_o) \cos\theta \mathrm{d}\omega_i = \int_A L_i(x,\omega_i) f_r(x,\omega_i,\omega_o) \frac{\cos\theta \cos\theta'}{\|x'-x\|^2} \mathrm{d}A$

Sampling the Light

algorithm

```
shade(p, wo)
    // Contribution from the light source
    Uniformly sample the light at x‘ (pdf_light = 1 / A)
    L_dir = L_i * f_r * cos\theta * cos\theta’ / (x‘ - p)^2 / pdf_light

    // Contribution from other reflectors
    L_indir = 0.0
    Test Russian Roulette with probaility P_RR
    Uniformly sample the hemisphere toward wi (pdf_hemi = 1 / 2pi)
    Trace a ray r(p, wi)
    if ray r hit a non-emitting object at q
        L_indir = shade(q, -wi) * f_r * cos\theta / pdf_hemi / P_RR
    return L_dir + L_indir
```