Introduction to Computer Graphics

10. Advanced Rendering

I-Chen Lin

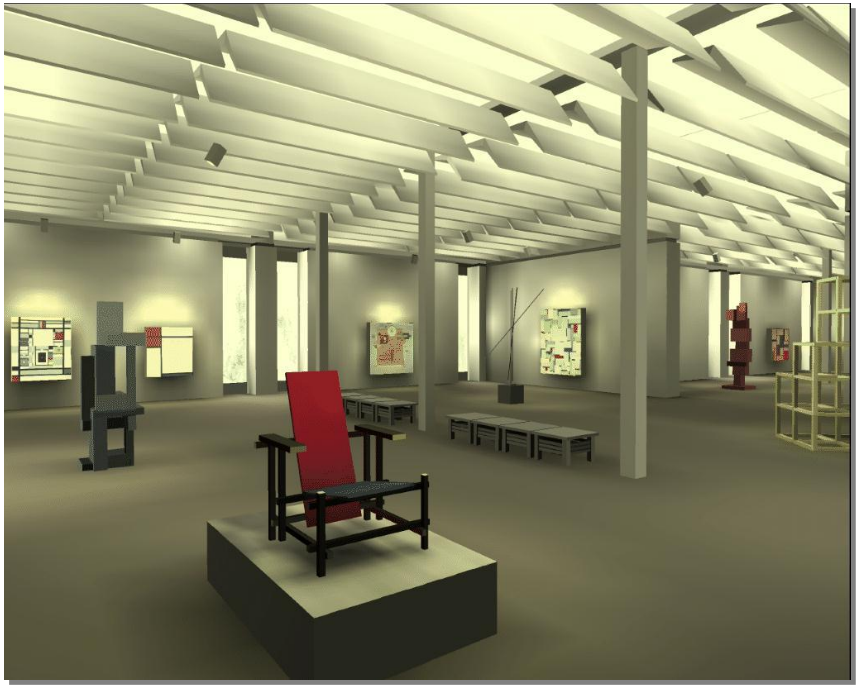
National Chiao Tung University

Textbook: E.Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson
Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson
John C. Hart, slides of Advanced Topics in Computer Graphics
H.W. Jenson, Realistic Image Synthesis using Photon mapping

Outline

- Going beyond pipeline rendering
- Ray tracing
- Rendering equation
- Radiosity 熱輻射法
- Photon mapping 光子映射法
- Real-time ray tracing 適合做鏡面的東西

Can We Render Images Like These?



玻璃部分反射部分折射

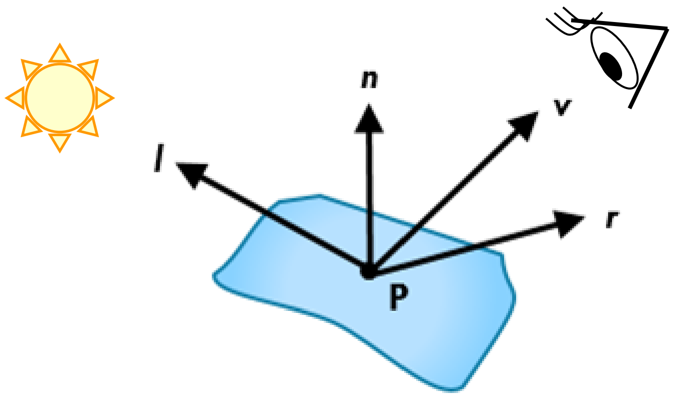
Pictures from http://www.graphics.cornell.edu/online/realistic/

Local Illumination

- The Phong model is a local illumination model
  - Shaded color depends only on
    - Surface normal, viewing direction, light direction
    - Ambient, diffuse, and specular reflectances

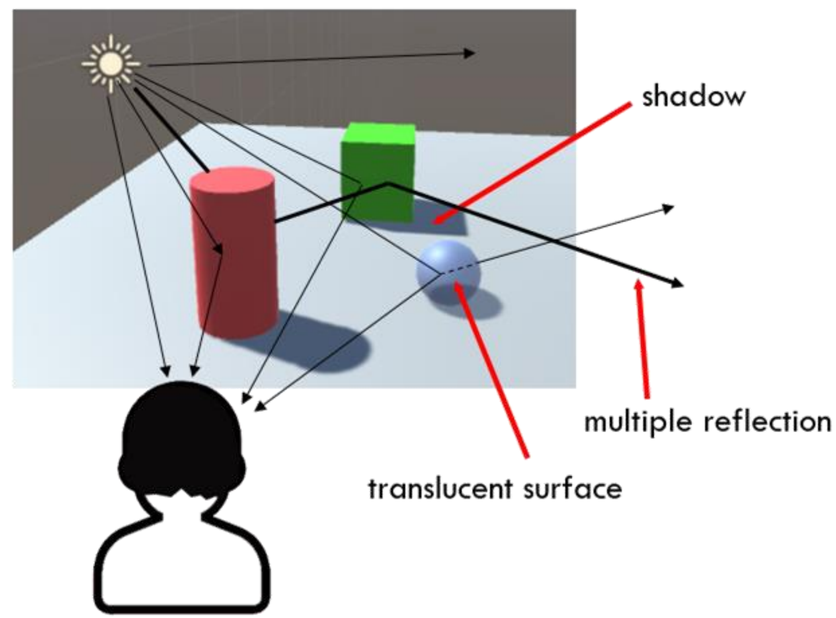
- $I = I_{ambient} + I_{diffuse} + I_{specular}$

  $= k_a I_a + k_d I_d(\mathbf{l} \cdot \mathbf{n}) + k_s I_s (\mathbf{v} \cdot \mathbf{r})^a$



Local Illumination (cont.) 局部光照：沒有考慮折射和反彈多次的光

- Don’t take other surfaces into account !
  - Other surfaces cannot block light (no shadows)
  - Omitting light from reflection or refraction of other objects.
- These interactions happen in reality!



光線追蹤 : trace ray pass

Forward Ray Tracing

global illumination : 全域光照

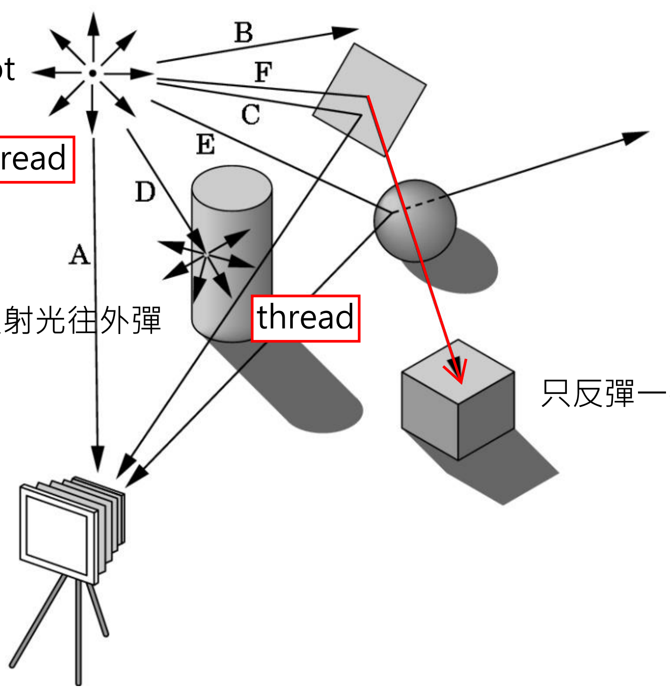
forward : 照著物理精神從光點往外打

放射

- Rays emanate from light sources and bounce around in the scene.
- Rays that pass through the projection plane and contribute to the final image.

以tree的角度 : 光點 : root

thread



會有很多額外的反射光往外彈

只反彈一次 : 完美的鏡子

What's wrong with this method?

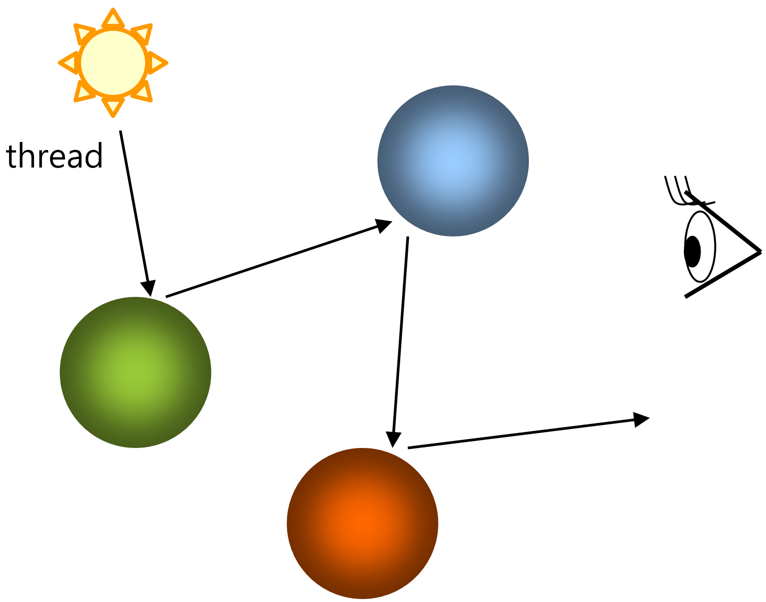
Forward vs. Backward

打的層數越多、畫面越擬真、但cost會越高

forward

Starting at the light

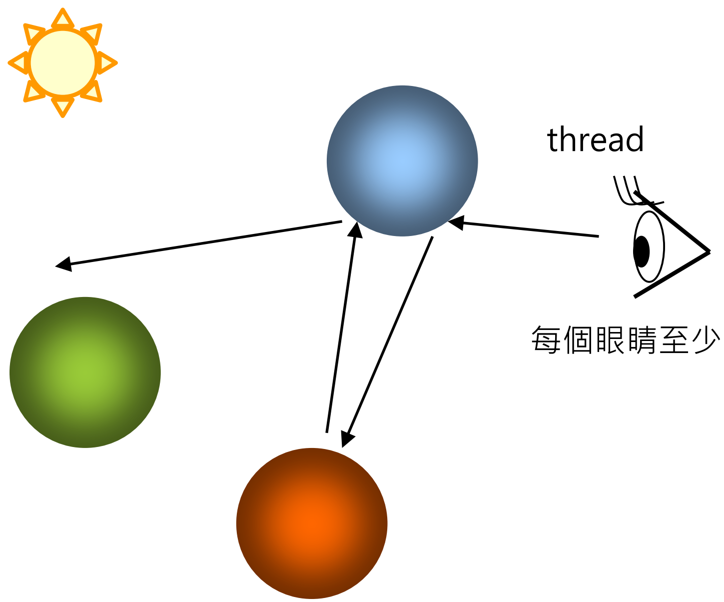
可能所有的光都打不到眼睛(看不到東西)



backward

Starting at the eye

彈到光源的機率也是低



每個眼睛至少一個射線

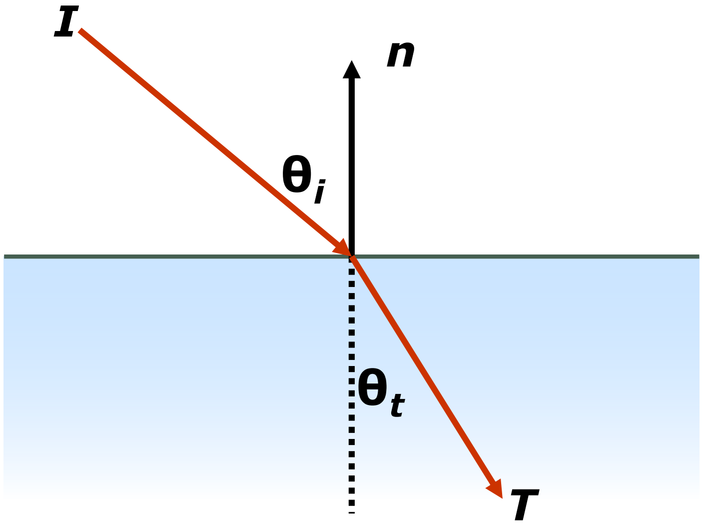
彈到某一個程度就可以先算phong lighting model
(backward ray trace + phong model)

折射
Refraction of Light

- Rays transitioning between materials are bent around normal
  - every material has an index of refraction
- Angles with surface normal obey Snell's Law

$$\boxed{\frac{\sin\theta_i}{\sin\theta_t} = \eta_{ti} = \frac{\eta_t}{\eta_i}}$$

Where η is the indices of refraction



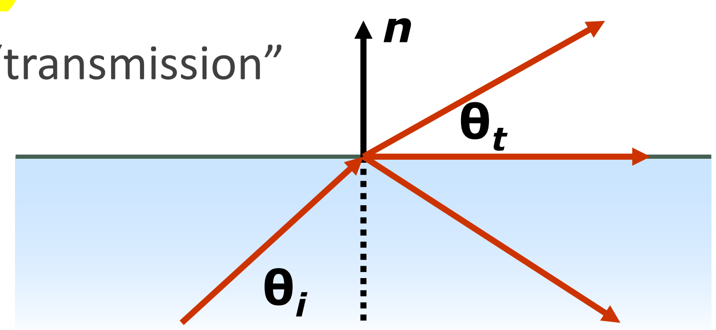
Material	Index of Refraction
Vacuum	1.0
Ice	1.309
Water	1.333
ethyl alcohol	1.36
Glass	1.5–1.6
Diamond	2.417

Refraction of Light (cont.)

- When entering material of lower index
  - Ray bends outward from normal
  - What if the angle is more than 90°?
    - Ray is actually reflected off the boundary
    - this is called total internal reflection (like fiber optics)
- Total internal reflection occurs when

$$\boldsymbol{\theta}_i > \theta_{critical}\text{, where} \qquad \theta_{critical} = \sin^{-1}\frac{\eta_t}{\eta_i}$$

  - just need to check for this critical angle
  - if above it, use specular reflection for "transmission"

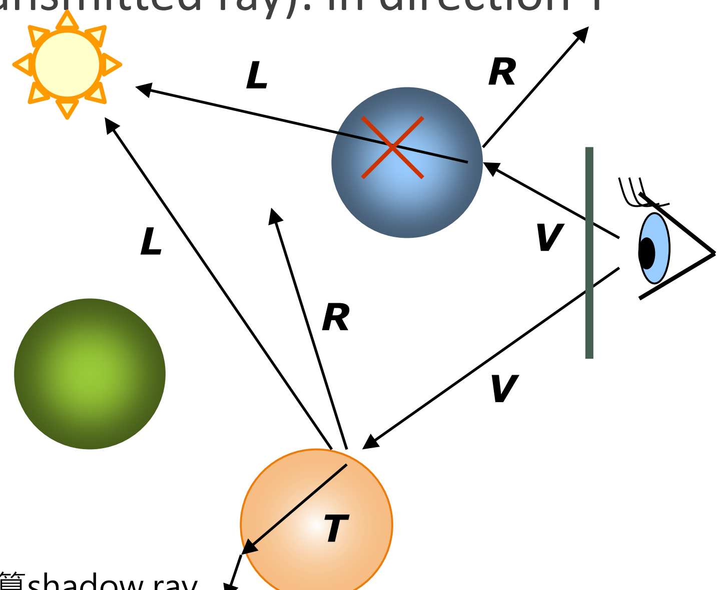


Whitted Ray-tracing

從眼睛出去的ray

1. For each pixel, trace a ***primary ray*** (eye ray) to the first visible surface.
2. For each intersection trace secondary rays: 每種可能的光都會再向下沿伸三種光

phong model
- ***Shadow rays***: in directions L to light sources 多一項檢測 : 途中會不會被物體擋住(不用多算)(occulted / visibility or not)=>cost不小
- ***Reflected ray***: in direction R 考慮物體的交互關係=>recursive call : 會生成新的、亮度較低的ray
- ***Refracted ray*** (transmitted ray): in direction T



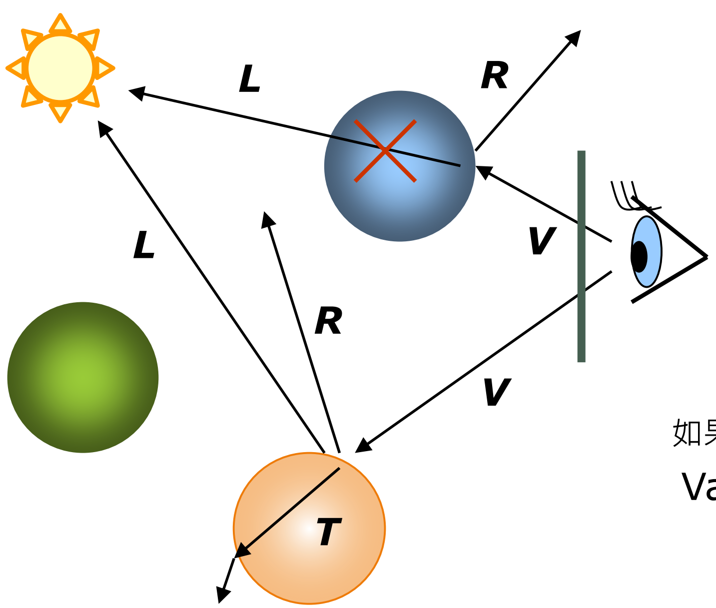
背面不會去算shadow ray

Whitted Ray-tracing (cont.)

- Every surface intersection spawns
  - 1 reflected ray
  - 1 transmitted ray
  - 1 shadow ray per light
  - Shaded color of $\boldsymbol{V_i}$ = Valid($\boldsymbol{L_i}$) x *PhongModel* + *ReflectedRay* + *TransmittedRay*

所有的點都會算反射光

玻璃型表面會需要計算



每個pixel都需要經過一個eye ray

如果眼睛看不到的話整條就會歸零

Valid(Li) = 1, if visible to the light source

0, otherwise

A Simple Ray Tracer

```
void raytrace()
    for all pixels (x,y)
      image(x,y) = trace(compute_eye_ray(x,y))

rgbColor trace(ray r)
  for all surfaces s {  =>for each triangle
      t = compute_intersection(r, s)
      closest_t = MIN(closest_t, t) 找最近的光打到的點距離
  }
                                  著色
   if( hit_an_object )
      return shade(closest_s, r, closest_t)
                        三角形
   else
      return background_color 沒有打到東西就給黑色
```

A Simple Ray Tracer (cont.)

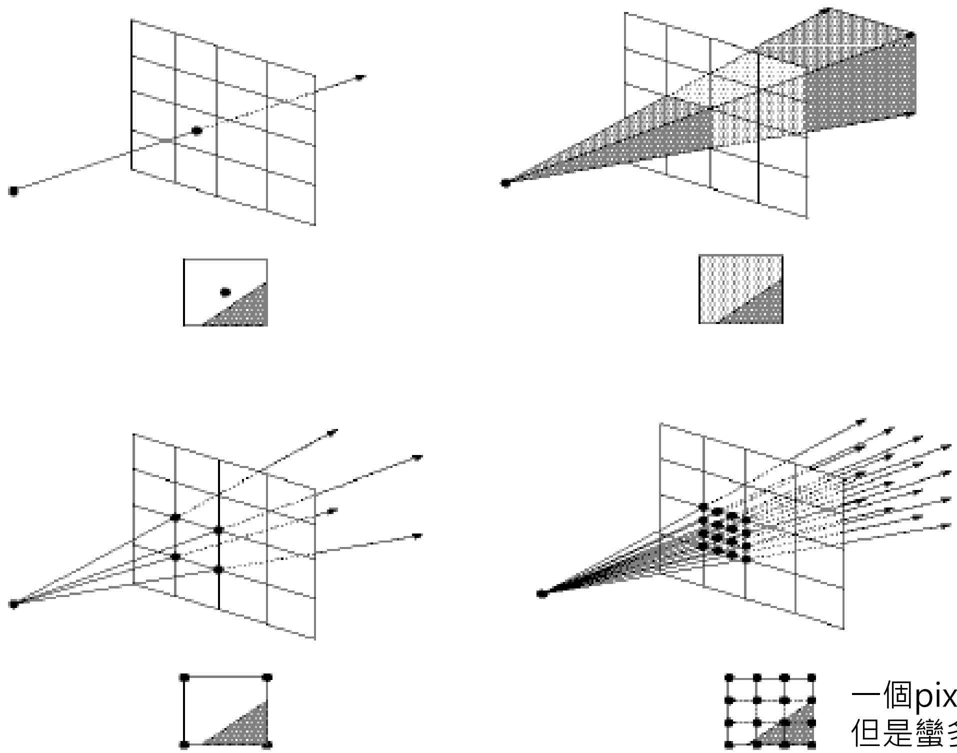
每道光每次三個分支(樹狀圖)，到phong的時候就停止，或設置最多打到幾層(會傳遞層數的參數)就停止

```
rgbColor shade(surface s, ray r, double t)
    point x = r(t)
    rgbColor color = black

shadow ray  for each light source L    對每個光源
      //Check whether there is no object on the line segment xL  中間沒有任何物體的情況下
      if( closest_hit(shadow_ray(x, L)) >= distance(x, L) ) {  撞到物體的距離大於等於光線的距離
            color += shade_phong(s, x)
      }
reflect ray   color += k_specular * trace(reflected_ray(s,r,x))  把新的光當作一般的光=>trace
refract ray   color += k_transmit * trace(transmitted_ray(s,r,x))
                       k係數：反射、折射光會被打折
    return color
```

Supersampling

- Aliasing problems. : FSAA(full screen anti-aliasing)
- We can approximate the average color of a pixel's area by firing multiple rays and averaging the result.



一個pixel多打一些點(拿多一點資料)
但是蠻多都只有打到空洞

Efficiency of Ray Tracing

- Consider this example
  - image resolution of 1024x768 = 786,432 pixels
  - 3x3 supersampling = 7 million eye rays
  - recursion depth 5 = 63*7 = 441 million rays 要去檢查有沒有遮蔽
    每道光有63條eye ray
  - each tested against 10,000 polygons
  - 4.4 trillion intersection tests (ignoring shadow rays)

10^3	10^6	10^9	10^12
K	M	Bi	Tri

Most of the time is spent in the calculation of intersection !

Efficiency of Ray Tracing (cont.)

- How to efficiently calculate intersections?
  - Efficient representation of an object.

    使用有效率的表達方式，EX : 球體以方程式來表達比較好retrace

  - Bounding boxes

    要畫很多小兵的話，就看ray有沒有打到包住全部小兵的bounding box

  - Space partitioning
    - Octree, BSP tree, etc.

  - Distributed ray tracing (non-uniform ray distribution)

  - ……

Efficiency of Ray Tracing (cont.)

- How to utilize more than 1 computer?

- The efficiency of realistic synthetic image rendering (in movie quality)

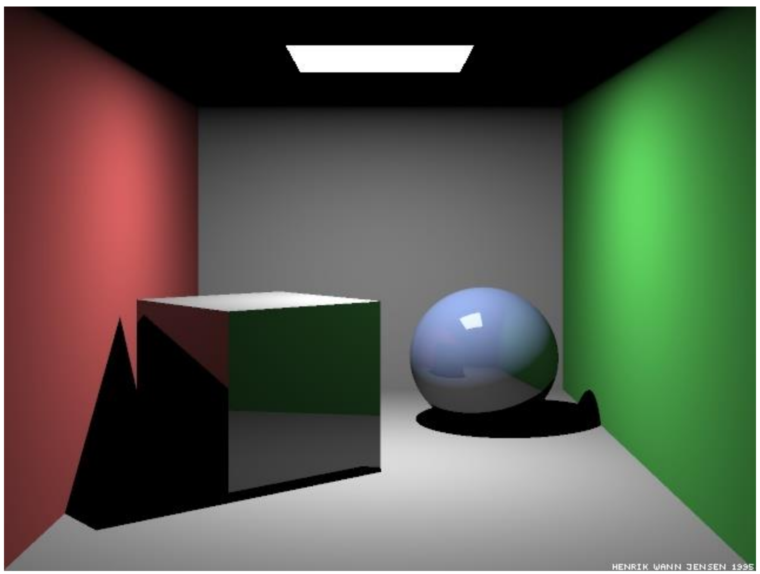
Ray Tracing Example

左邊shadow太強、右邊shadow比較真實

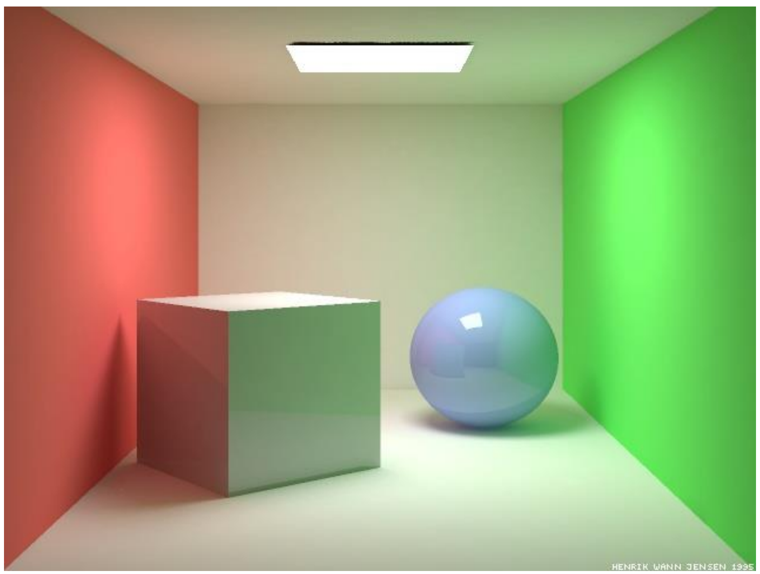
whitted raytrace做的

What’re missing?

How to handle these?



VS



Ray-traced Cornell box by Henrik Jensen,
http://www.gk.dtu.dk/~hwj

Ray Tracing vs. Radiosity

- Ray tracing
  - An image space algorithm 畫面一旦有更動就要重算
  - View-dependent
  - Rendering scenes with perfect specular reflection and refraction.
  - Point light sources.
  - Ideas from the path of light flow

每打出光會三條光有兩條會繼續往下長
可以offline計算、與view無關
熱輻射法：光會彈到熱平衡

- Radiosity
  - An object space algorithm
  - View-independent (can be pre-computed)
  - Rendering perfect diffuse scenes. 光會往四面八方彈
  - Light sources are polygonal patches.
  - Ideas from the conservation of energy.

The Rendering Equation

- Regarding the light as a form of energy.

- In a closed environment, we do not see how the rays have bounced around.

- What we see is at an equilibrium state.
  - [***outgoing***] = [***emitted***] + [***reflected***] + [***transmitted***]
    自發光 反射光 折射進來的光
  - (We usually omit the “transmitted” terms)

The Rendering Equation (cont.)

反射係數Kd

P'發到P能量

$$I(p, p') = v(p, p')\left[\varepsilon(p, p') + \int \boxed{\rho(p, p', p'')} \cdot I(p', p'')dp''\right]$$

- $I(p, p')$: intensity passing from p' to p.
- $\varepsilon(p, p')$: emitted light intensity from p' to p.
- $\rho(p, p', p'')$: reflection function at point p'.
- $v(p, p')$: visibility function

0: if p' is invisible from p.

$1/r^2$: if p' is visible from p.

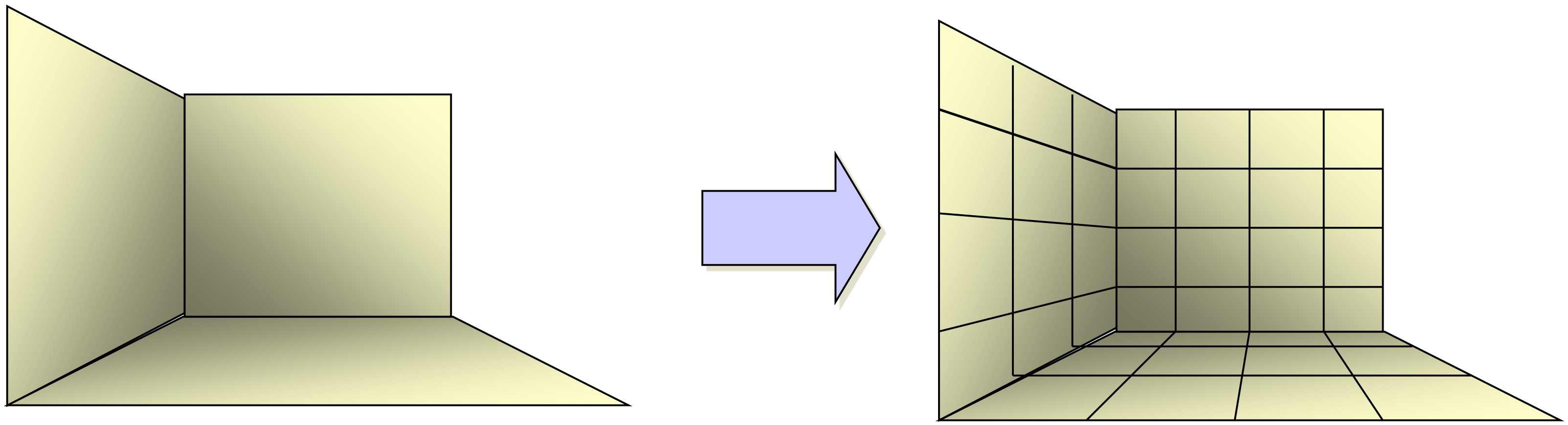
光會隨著距離分散掉

- r : distance between p and p'.

Radiosity

- One way to simplify the rendering equation.
  - All surfaces are perfectly diffuse reflectors.
  - Dealing with diffuse-diffuse interactions.
- A scene is divided into "patches".

一片一片分開算



Radiosity (cont.)

$$I(p,p') = v(p,p')\left[\varepsilon(p,p') + \int \rho(p,p',p'') \cdot I(p',p'')\,dp''\right]$$

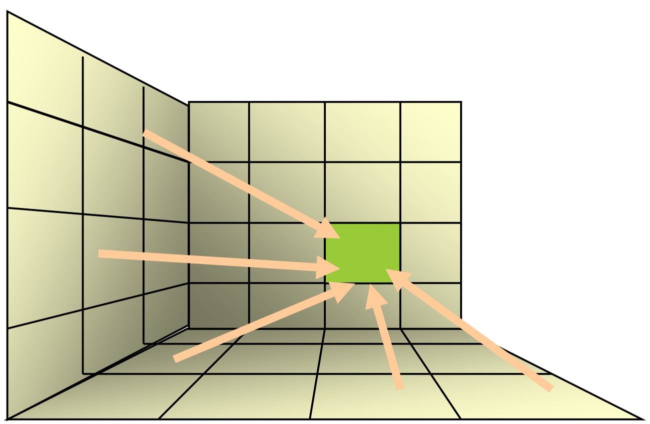
patch i $$b_i a_i = e_i a_i + \rho_i \sum_{j=0}^{n} f_{ji} b_j a_j$$

The light intensity of i

The emissive intensity

The reflective intensity due to intensity of all other patches (j)

b未知、其它都已知

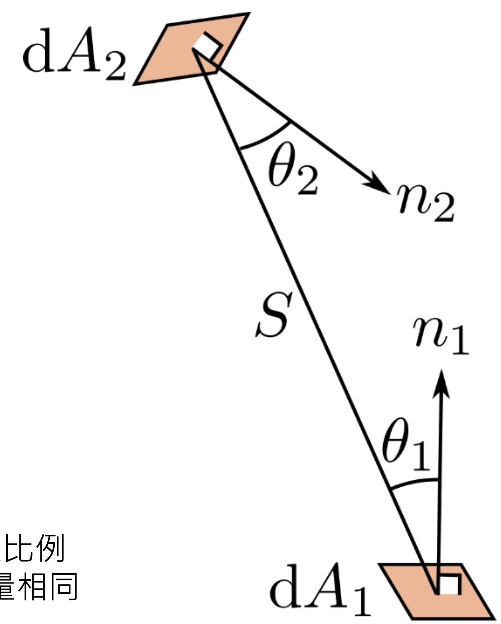


- ρ_i : reflectance of element *i* (given)
- b_i : the color of patch *i* (unknown) 要解出平衡狀態的顏色
- a_i : the area of patch *i* (computable)
- e_i : the emissive component (given)
- f_{ji} : the form factor (*j* -> *i*) (computable)

(熱力學)能量轉換的比例

Form Factor

- F_{ji}: Fraction of light leaving element *j* and arriving at element *i*
- Depends on
  - Shape of patches i and j
  - Relative orientation of both patches
  - Distance between patches
  - Visibility or occlusion by other patches



$$dF_{12} = \frac{\cos\theta_1 \cos\theta_2}{\pi S^2} dA_2$$

$$F_{12} = \frac{1}{A_1} \int_{A_1} \int_{A_2} \frac{\cos\theta_1 \cos\theta_2}{\pi S^2} dA_2 dA_1$$

$$\sum_j F_{ij} = 1$$

$$\boxed{A_1 F_{12} = A_2 F_{21}}$$

reciprocity

F12 : 1送到2的能量比例
與對面送過來的能量相同

與兩個角度有關

Fig. from: en.wikipedia.org/wiki/View_factor

Radiosity (cont.)

- The reciprocity equation
  - $\boxed{f_{ij}\, a_i = f_{ji}\, a_j}$

$$b_i a_i = e_i a_i + \rho_i \sum_{j=0}^{n} f_{ji} b_j a_j$$

想要置換成下面這樣

$$b_i a_i = e_i a_i + \rho_i \sum_{j=0}^{n} f_{ij} b_j a_i$$

就可以得到bi

$$b_i = \boxed{e_i} + \boxed{\rho_i} \sum_{j=0}^{n} \boxed{f_{ij}} b_j$$

人為假設 可計算

The radiosity equation

Radiosity (cont.)

- Put the equations in matrix form.

$$b_i = e_i + \rho_i \sum_{j=0}^{n} f_{ij} b_j$$

matrix inverse

$$\begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} - \begin{bmatrix} \rho_0 & & & 0 \\ & \rho_1 & & \\ & & \ddots & \\ 0 & & & \rho_n \end{bmatrix} \begin{bmatrix} f_{00} & f_{01} & \cdots & f_{0n} \\ f_{10} & f_{11} & & \\ \vdots & & \ddots & \vdots \\ f_{n0} & \cdots & & f_{nn} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

e* = *b* - *R* *F* *b

The solution is matrix很大、cost高(inverse的複雜度是O(n^3)，n > 10^4的畫memory會爆)

$$b = [I - RF]^{-1} e$$

⇦ **Is it feasible?**

Solving the Radiosity Equation

- Direct inverse: dimensional problem.
- Jacobi method

$$Ax = c, A = D + O$$
$$(\mathrm{D:diagonal\ matrix,O:residual})$$
$$(D + O)x = c$$
$$Dx = c - Ox$$
$$x^{t+1} = D^{-1}(c - Ox^{t})$$

- Gauss-Seidal method

$$Ax = c, A = L + U$$
$$(\mathrm{L:lower\ triangle\ plus\ diagonal\ matrix,U:upper\ triangle\ matrix})$$
$$(L + U)x = c$$
$$Lx = c - Ux$$
$$x^{t+1} = L^{-1}(c - Ux^{t})$$

$$[I - RF]b = e$$
$$A = [I - RF]$$
$$e = c$$
$$\mathrm{unknown\ b}$$

Solving the Radiosity Equation 不考

- Solving the equation by a direct method (e.g. Gaussian elimination) is infeasible.
  - **F** is too large.

- Solving by iterative numerical methods 兩個都跑不動
  - Jacobi's method

$$b_i^{k+1} = \frac{1}{1-\rho_i f_{ii}}\left(e_i + \sum_{j=1, j\neq i}^{n} \rho_i f_{ij} b_j^k\right)$$

  - the Gauss-Seidel method 但這個較節省memory

$$b_i^{k+1} = \frac{1}{1-\rho_i f_{ii}}\left(e_i + \sum_{j=1}^{i-1} \rho_i f_{ij} b_j^{k+1} + \sum_{j=i+1}^{n} \rho_i f_{ij} b_j^k\right)$$

Ref: https://en.wikipedia.org/wiki/Gauss-Seidel_method

Solving the Radiosity Equation

- **Jacobi's method**
  - need two copies of radiosity vector **B**
  - doesn't always converge quickly
- **the Gauss-Seidel method**
  - no additional copies
  - it converges more quickly

```
// Make an initial guess
for all i { b_i = e_i }

// Iteratively improve guess
while( not converged )
{
      for each i
      {
        sum = 0;
        for all j except i
        sum +=ρ_i b_j*f_ij;
         b_i = e_i + ρ_i sum;
      }
}
```

Calculating Form Factors

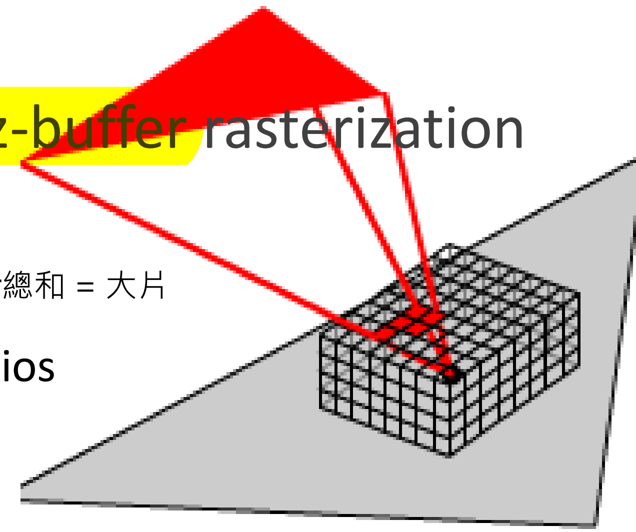
- One simple way uses ray tracing & point-to-area form factors

```
p = center of a_i;
f_ij = 0;
for k = 1 to N (separate a_j into N pieces)
{
        q = point on a_j;
        if( is_visible(p,q) )
         // Trace ray to test visibility
        f_ij += cos(…)*cos(…)/(π*r*r) * (a_j/N);

}
```

Hemicube Algorithm 加速計算的方法

- A hemicube is constructed around the center of each patch.
- Faces of the hemicube are divided into "pixels"
- Each patch is projected (rasterized) onto the faces of the hemicube.
- Each pixel stores its **pre-computed** form factor.
- The form factor for a particular patch is just the sum of the pixels it overlaps.
- Patch occlusions are handled similar to z-buffer rasterization



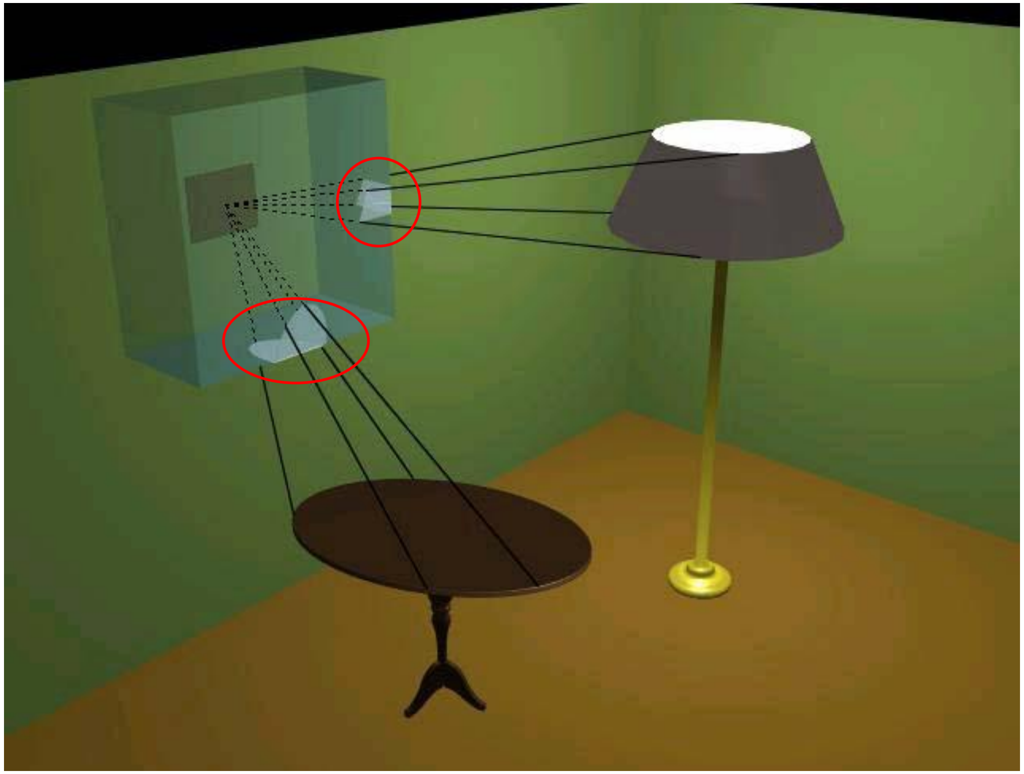
小片form vector總和 = 大片

In a closed scene, $\Sigma(j=1\sim n)f_{ij}=1$, because the radiosity-ratios affecting a patch must add up to 100%.

http://commons.wikimedia.org/wiki/File:Hemicube_Radiosity.png

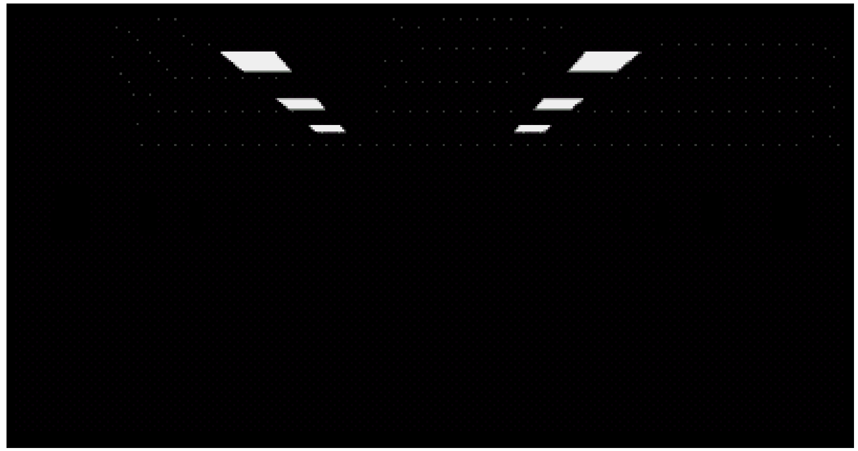
Hemicube Algorithm

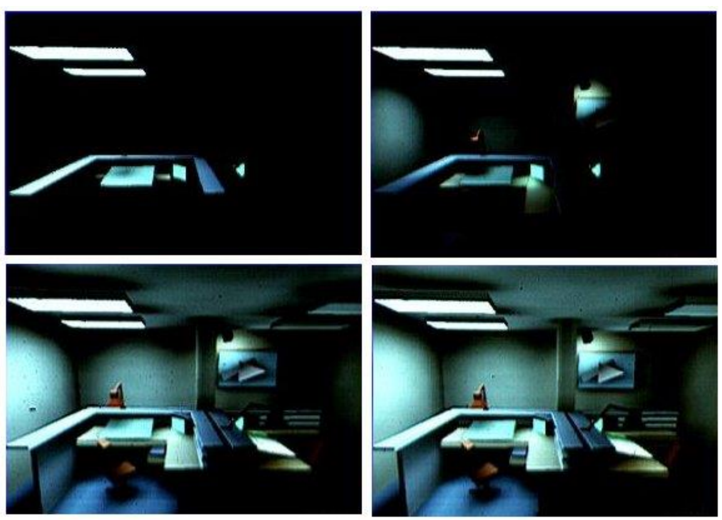
每個patch都罩一個box，把圖像直接投影到patch

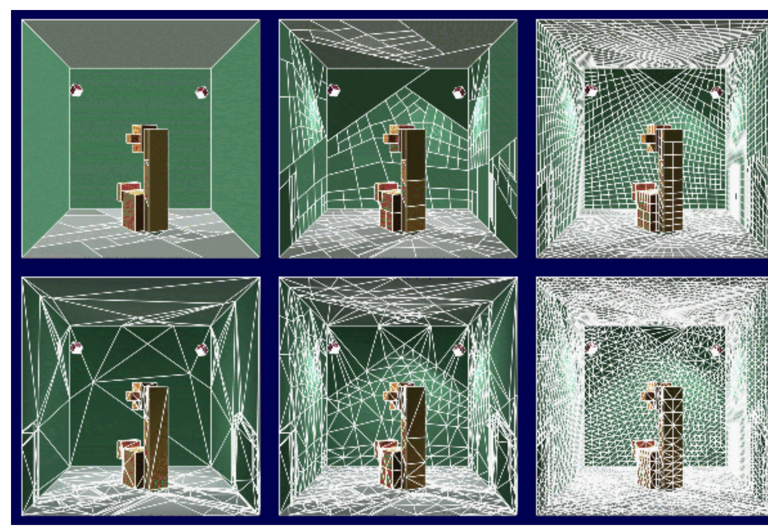


http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_2.htm

Radiosity diffuse不能處理折射和鏡反射







http://www.siggraph.org/education/materials/HyperGraph/radiosity/overview_3.htm

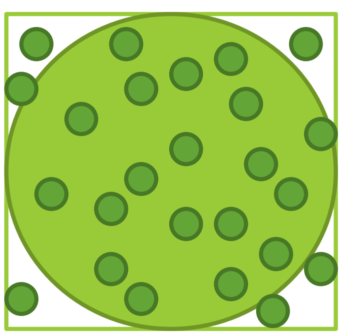
Monte-Carlo computation of π

- Take a random point (x,y) in unit square random sample看有哪些點在
- $\boxed{\text{Test if it is inside the disc } (x^2 + y^2 < 1)}$
- Probability of being inside disc 會有這個機率的點在園內

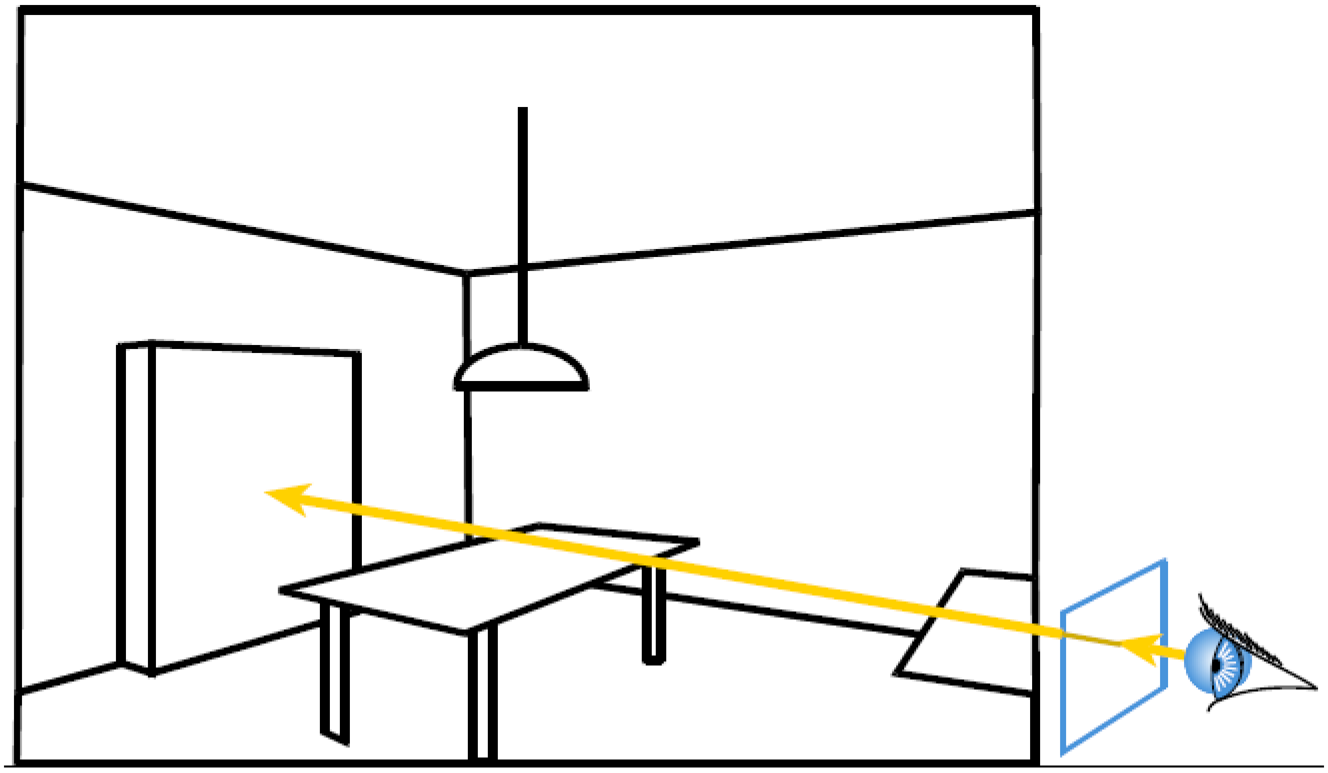
  = (area of unit circle)/ (area of 2x2 square) = **$\pi/4$**

  pi r^2

- $\boxed{\pi \approx 4 * \text{ number inside disc / total number}}$ 可以推算出pi
- The error depends on the number or trials



Ray casting

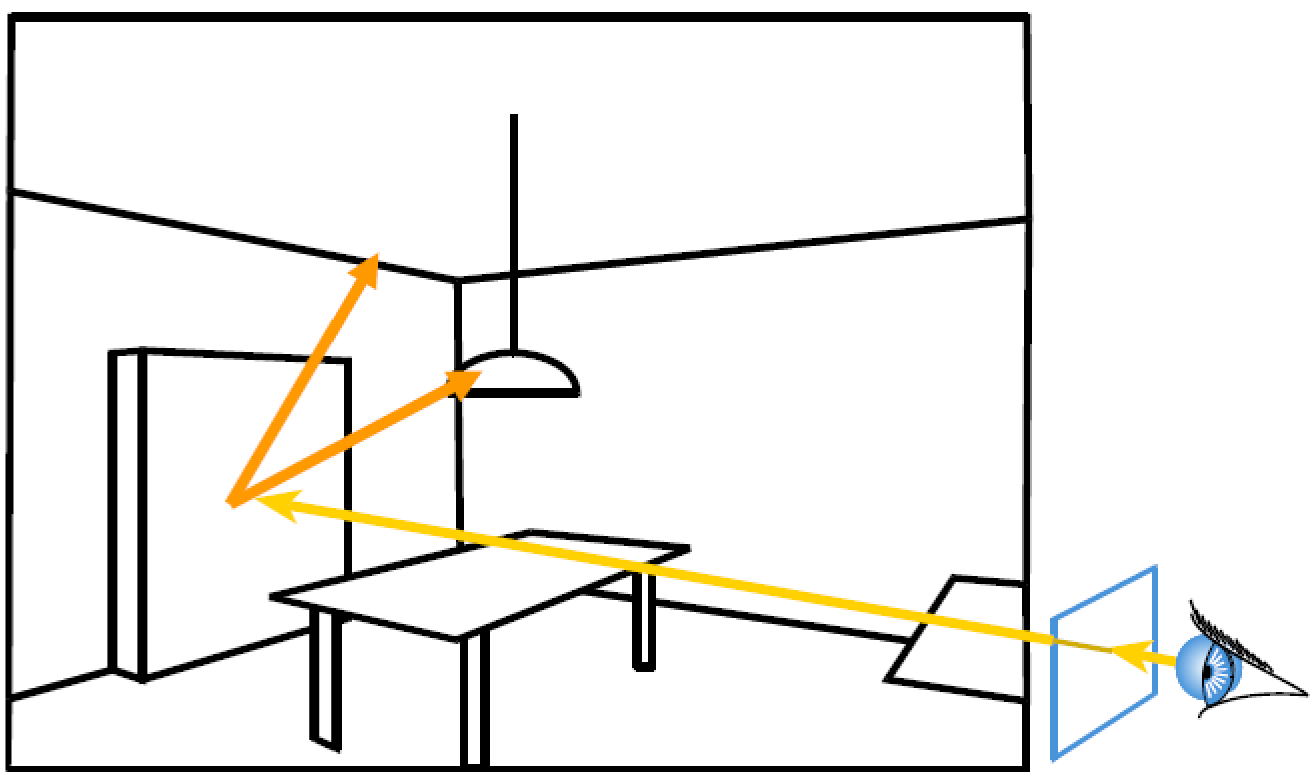


Slides from Cutler, Durand

whitted : 會做鏡反射但太過鏡面

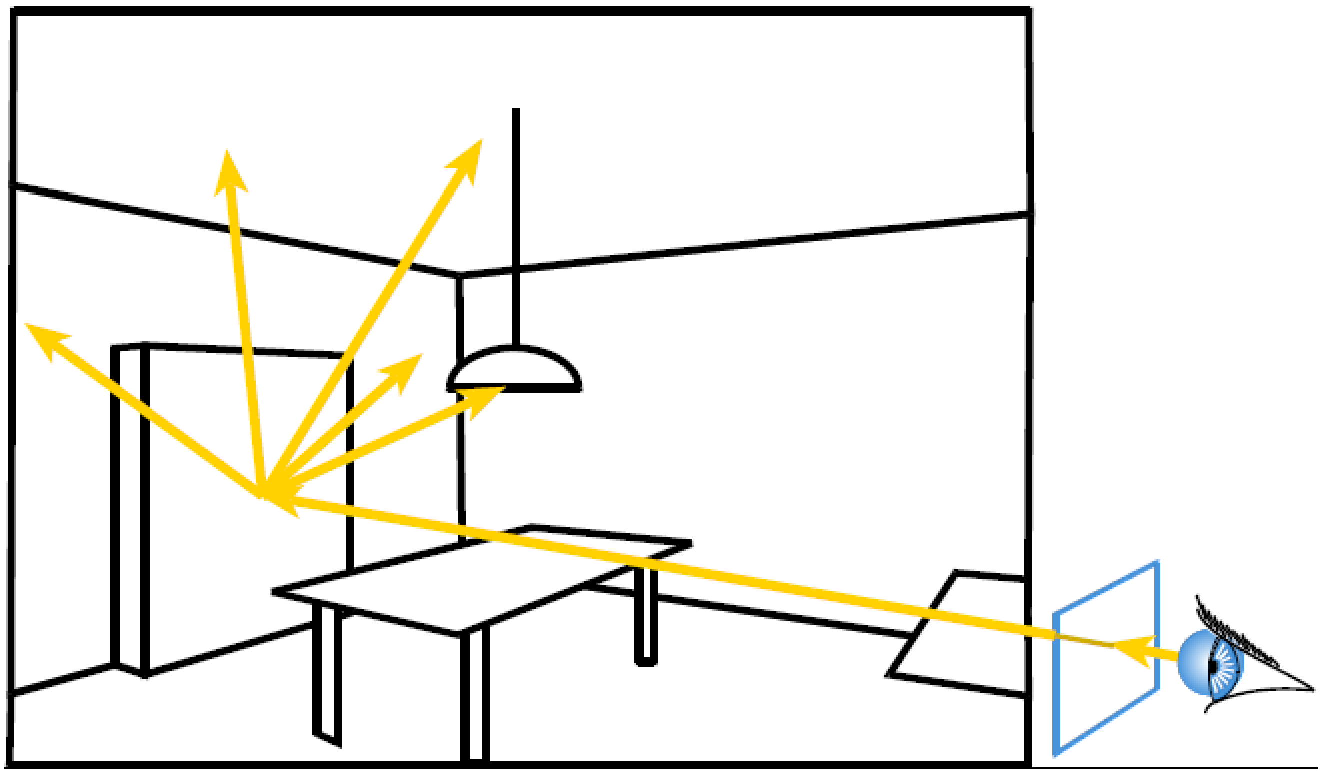
Ray tracing

- Cast a ray from the eye through each pixel
- Trace secondary rays (light, reflection, refraction) diffuse僅靠shadow ray去計算



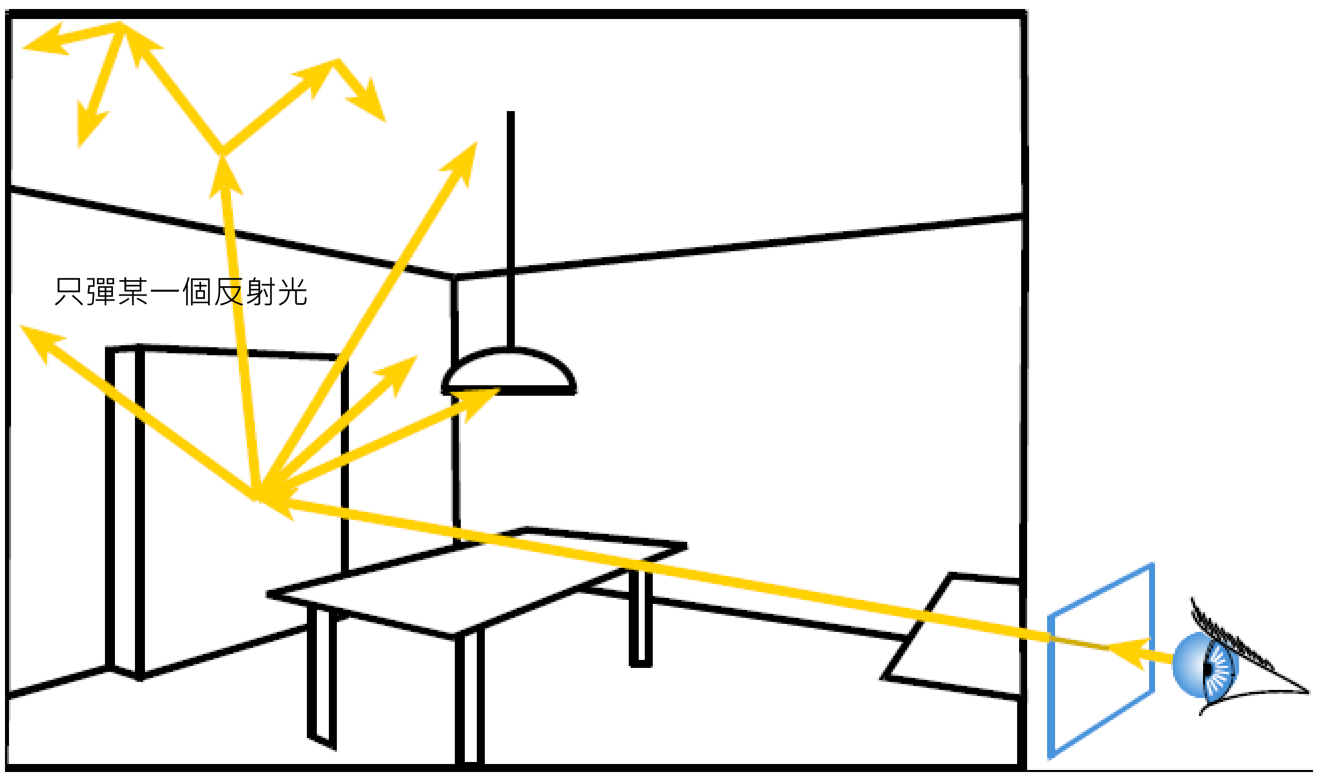
Monte-Carlo ray tracing

- Cast a ray from the eye through each pixel
- Cast random rays from the visible point 比較有機會還原光真實的狀況
  - Accumulate radiance contribution



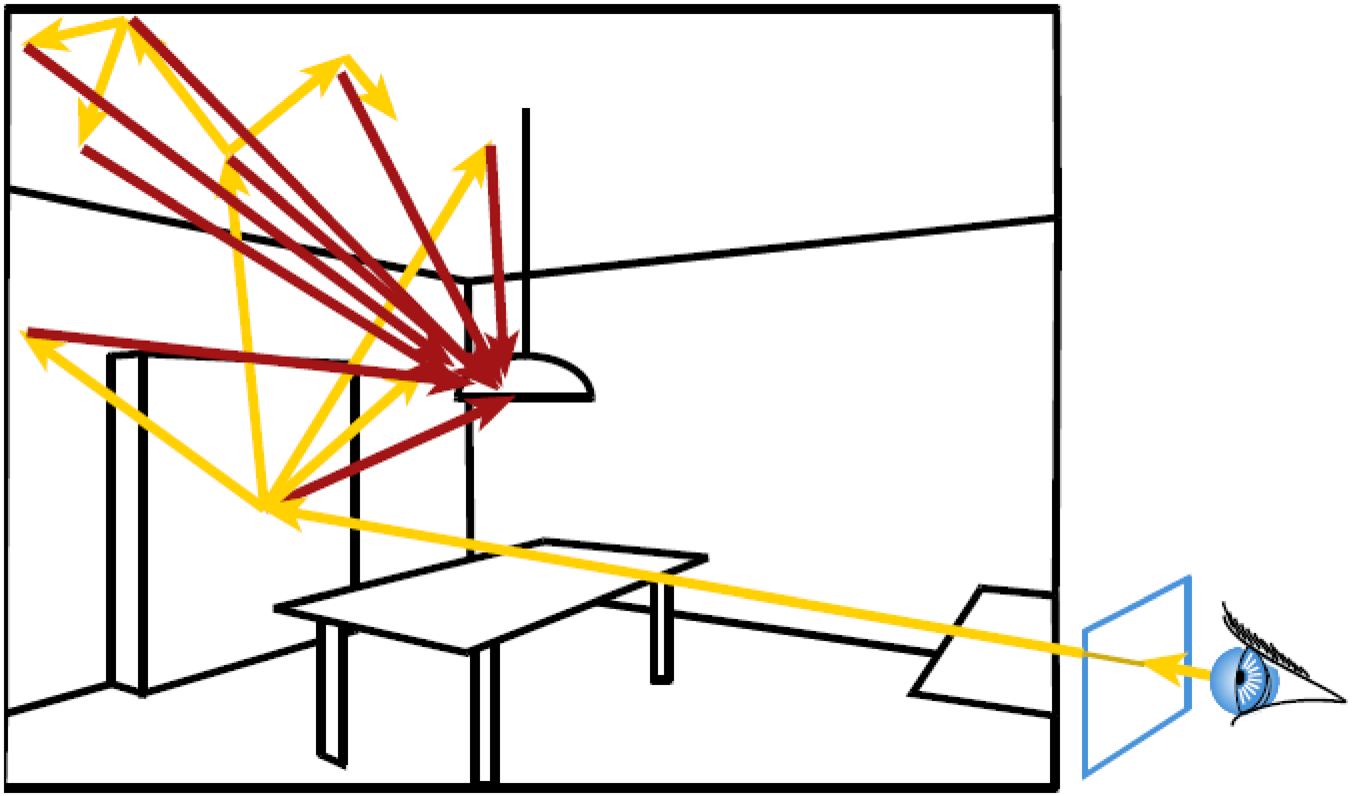
Monte-Carlo ray tracing

- Recursion



Monte-Carlo ray tracing 會產生很多branch的那種

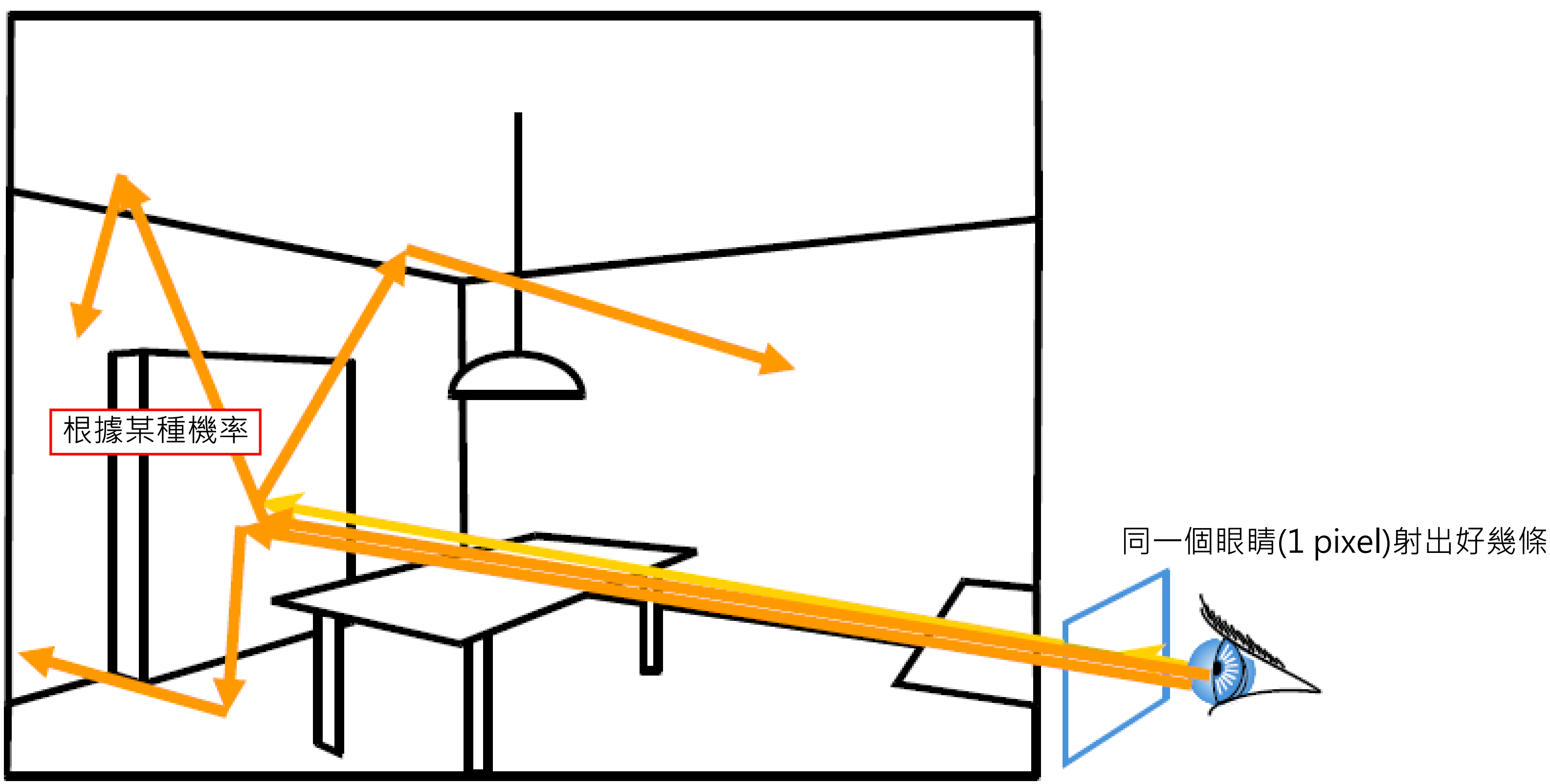
- Systematically sample primary light



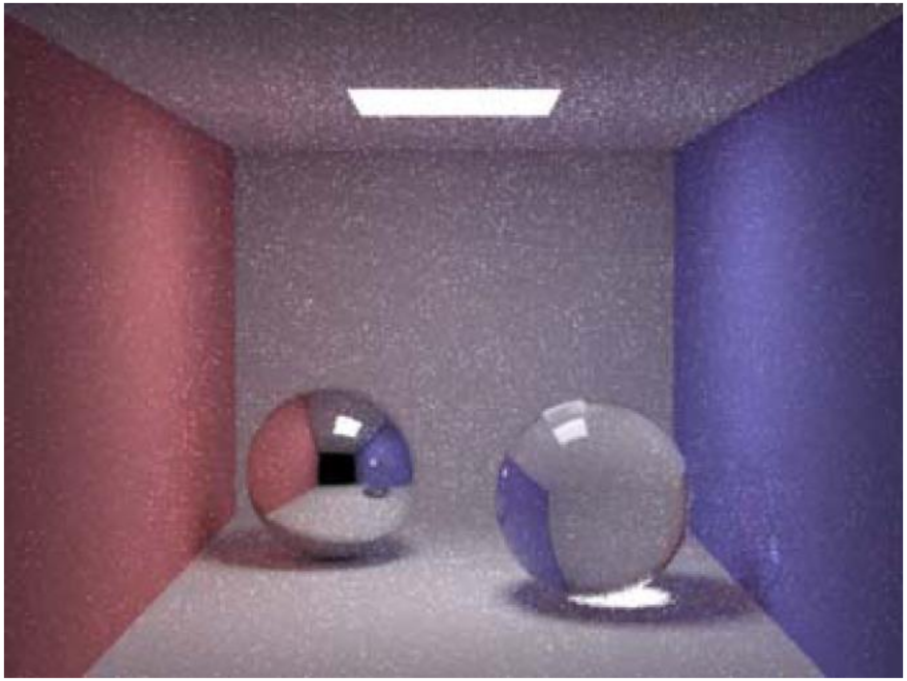
Monte Carlo path tracing

- Trace only one secondary ray per recursion
- But send many primary rays per pixel

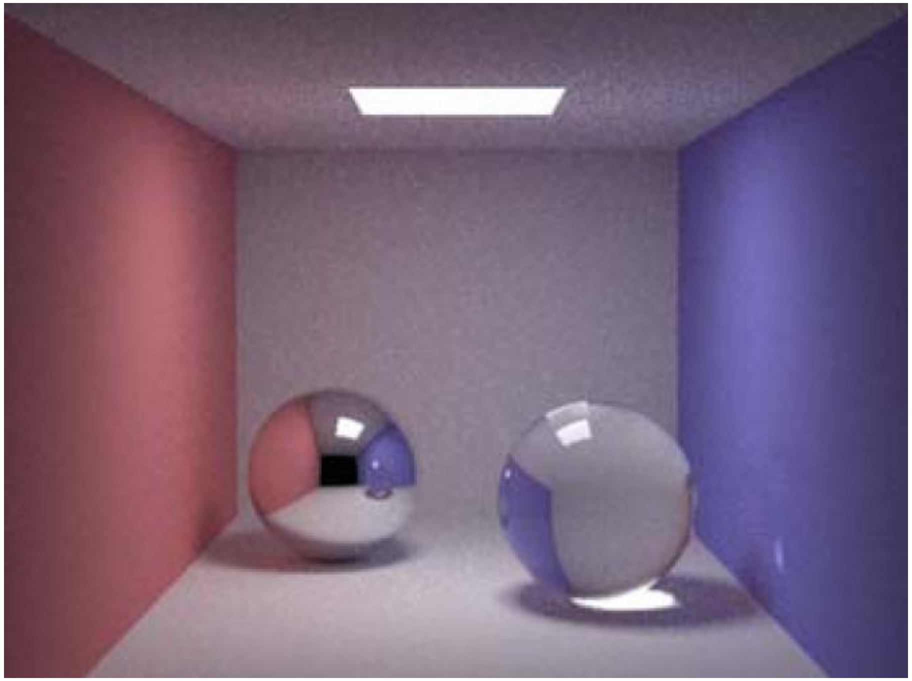
直接從起點控制ray的數量
出去之後random往外彈
這個方法現在比較常用
recursive比較難控制ray的數量



Monte Carlo path tracing



10 paths/pixel



100 paths/pixel

Problem of path tracing

1000 paths/pixel

Figure by H.W. Jensen

結合ray tracing和radiosity(cover眼睛和patch(光源))

Photon mapping

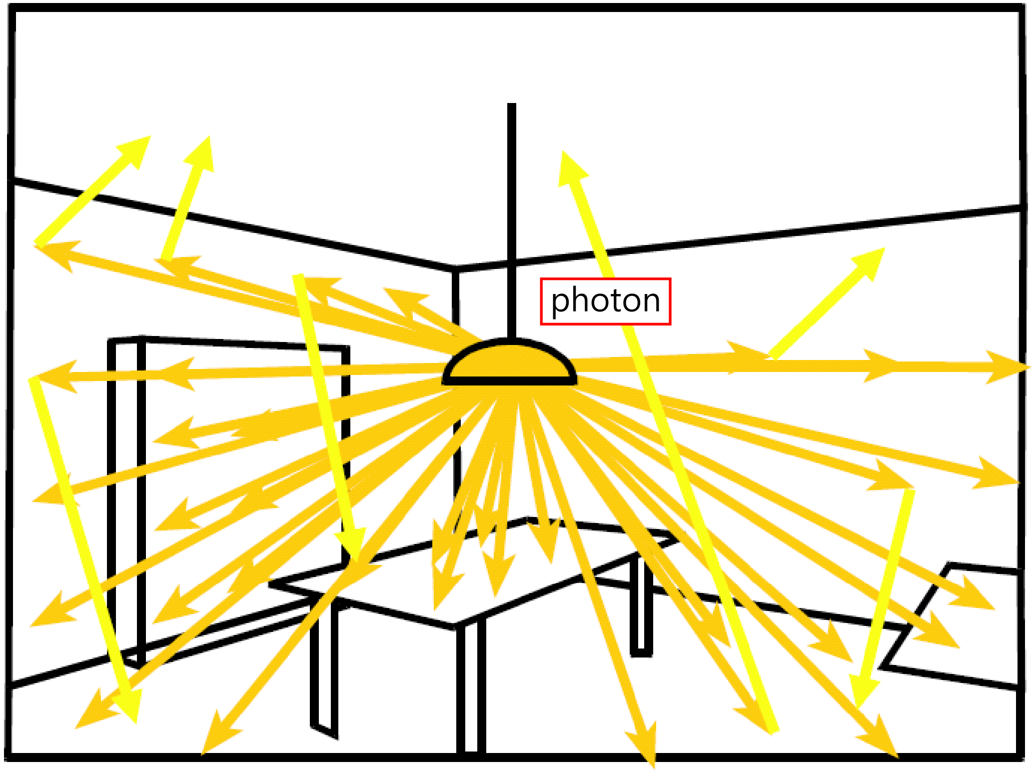
現在的做法

ray trace : 眼睛發出去的 vs.
photon mapping : 以patch為能量源亂彈

- Bi-directional paths
  - Construct paths not only from the eye, but also from the light sources
- Caching
  - Cache photons distributed along paths from the light sources
- Interpolation
  - Interpolate radiance from cached photons

Photon mapping 可以設定specular、catch、diffuse的比例

- Photon emission and transport

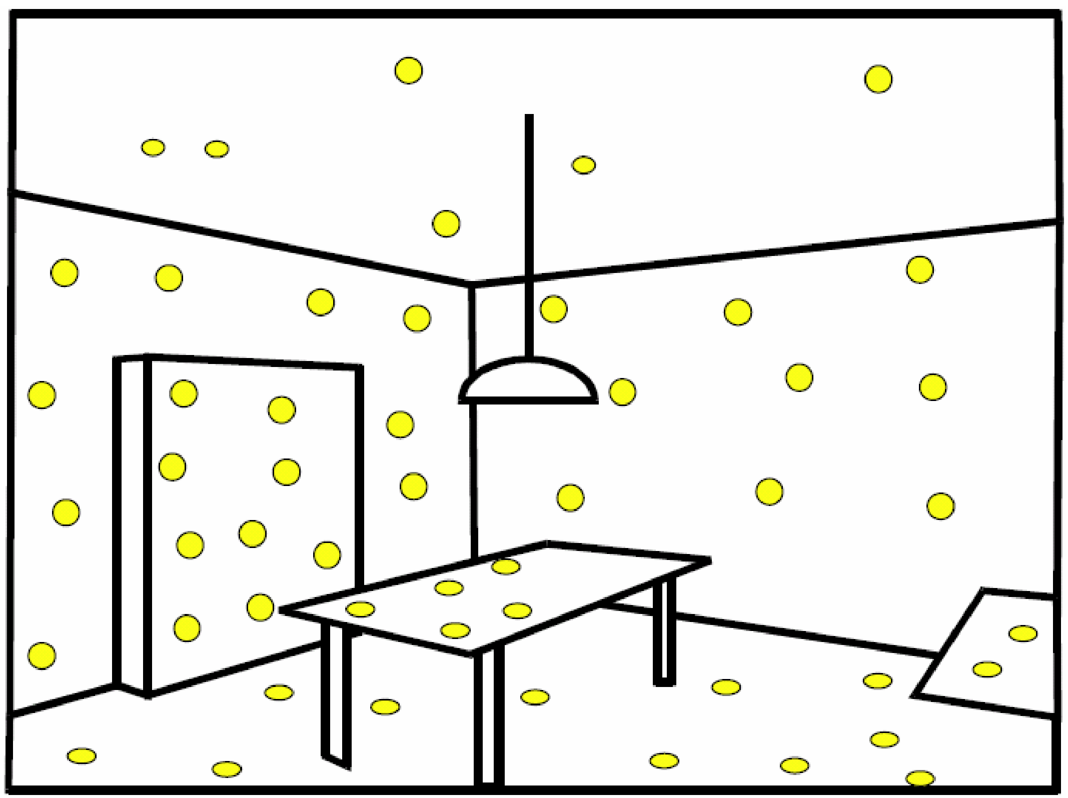


Slides from Cutler, Durand

Photon mapping

眼睛的射線考慮局部的photons的顏色

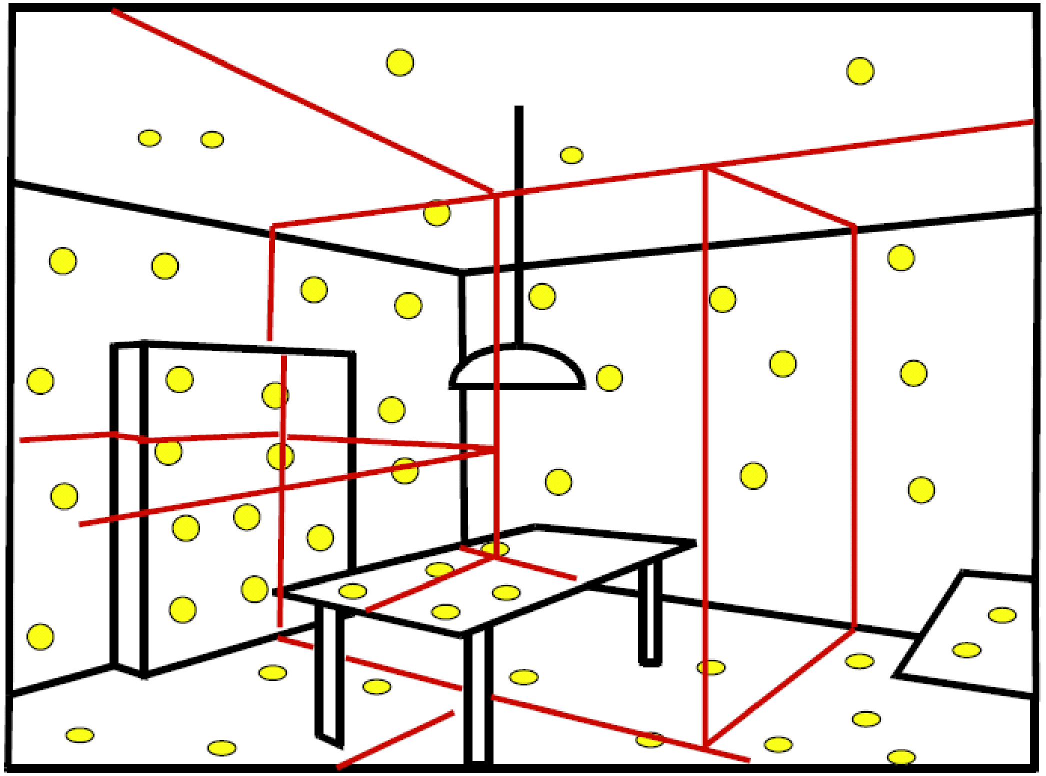
- Photon caching 彈了幾次留下光點



Slides from Cutler, Durand

Photon mapping

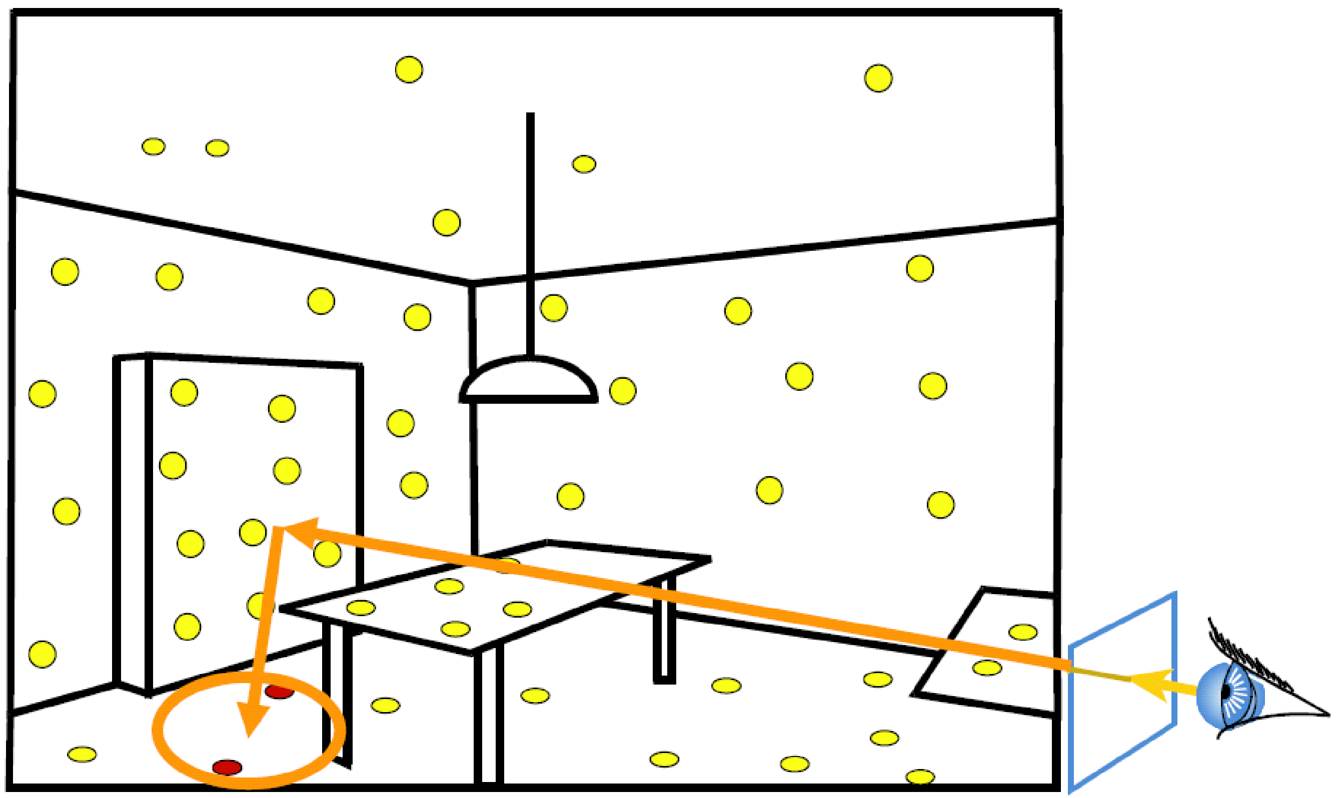
- Spatial data structure for fast access



[Cutler, Durand]

Photon mapping

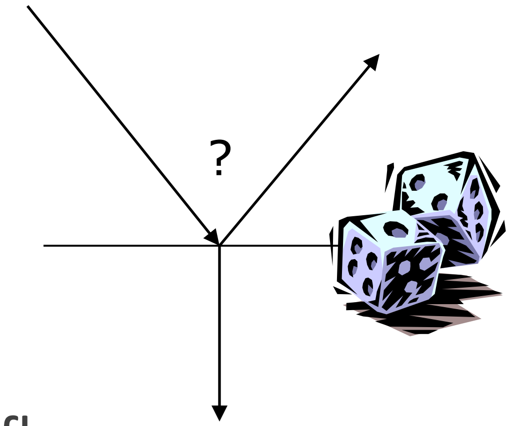
- Radiance estimation



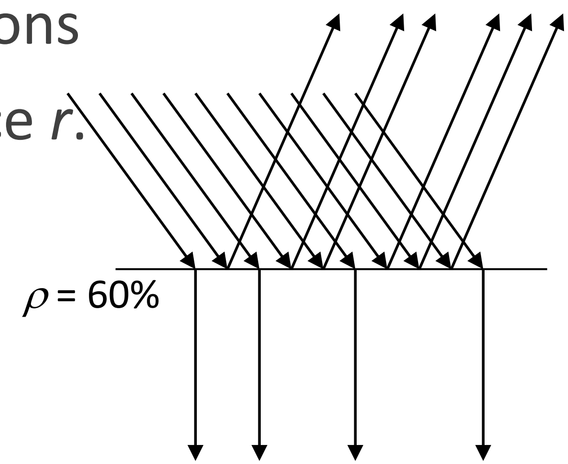
要考慮周圍的光(area filtering)
周圍顏色取平均

[Cutler, Durand]

Russian Roulette

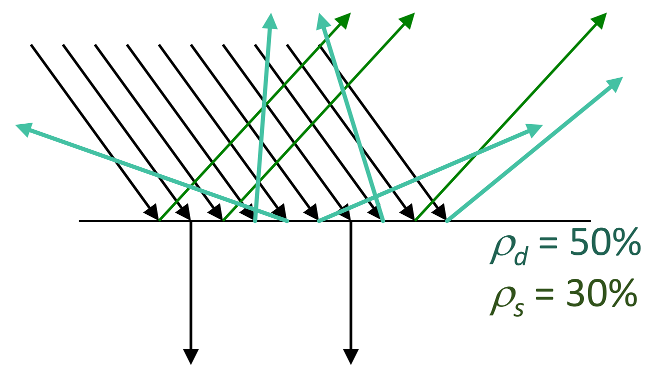


- Arvo & Kirk, S90
- Reflected flux only a fraction of incident flux
- After several reflections, spending a lot of time keeping track of very little flux
- Instead, completely absorb some photons and completely reflect others at full power
- Spend time tracing fewer full power photons
- Probability of reflectance is the reflectance *r*.
- Probability of absorption is 1 – *r*.



Distribution

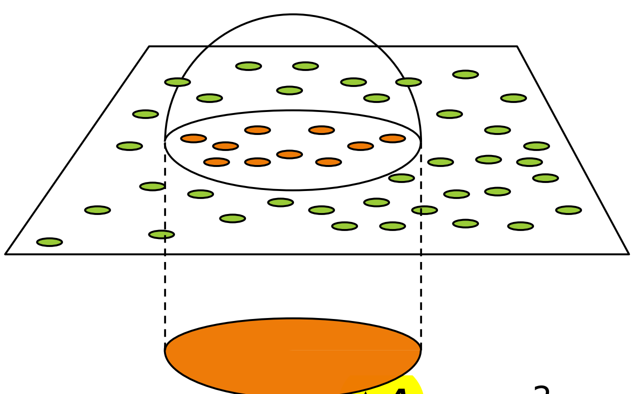
- Surfaces have specular and diffuse components
  - r_d – diffuse reflectance
  - r_s – specular reflectance
  - $r_d + r_s < 1$ (conservation of energy)
- Let z be a uniform random value from 0 to 1
- If $z < r_d$ then reflect diffuse
- Else if $z < r_d + r_s$ then reflect specular
- Otherwise absorb



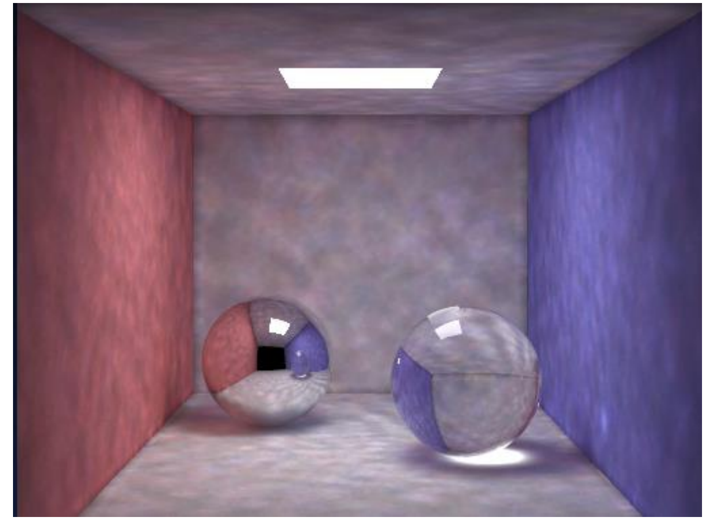
How many photons?

光點多area可以取小一點(可以做的較精細)；
光點少area就要取大一點

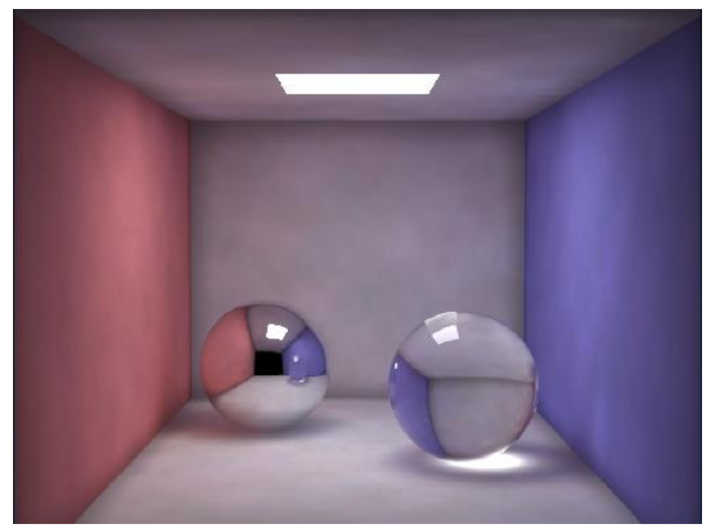
- How big is the disk radius r?
- Large enough that the disk surrounds the n nearest photons.
- The number of photons used for a radiance estimate n is usually between 50 and 500.



$\Delta A = \pi r^2$



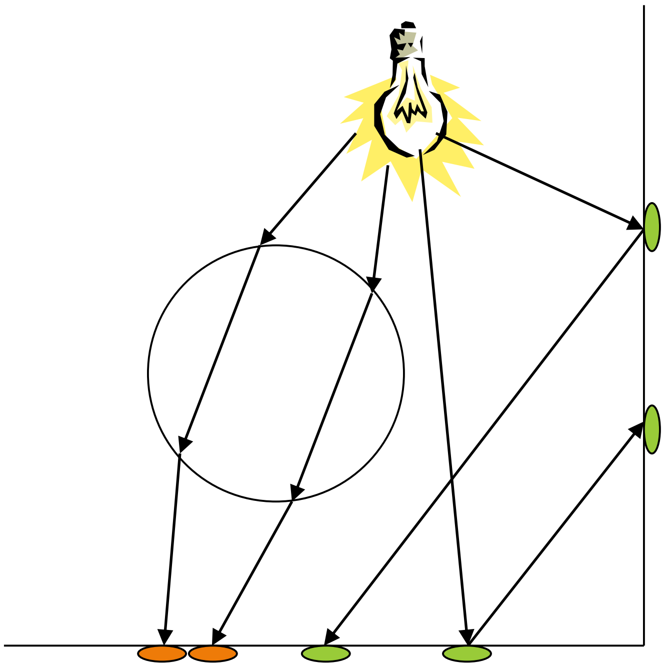
Radiance estimate using 50 photons



Radiance estimate using 500 photons

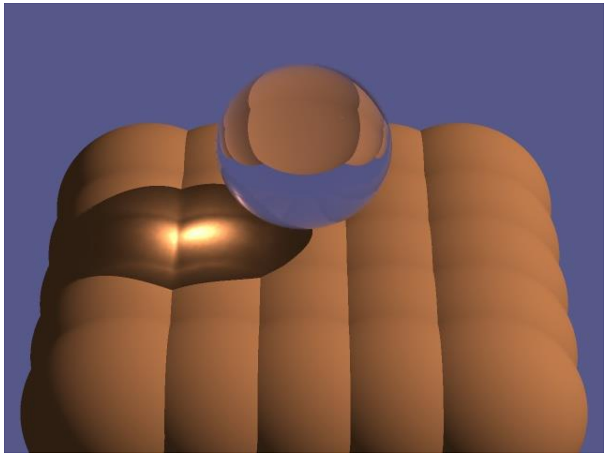
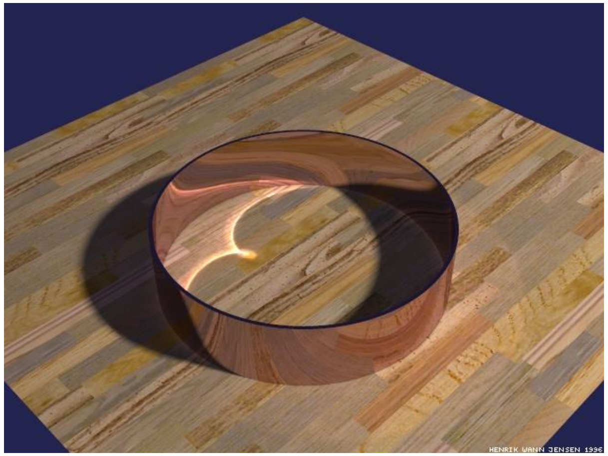
Multiple Photon Maps

- Global L(S|D)*D photon map
  - Photon sticks to diffuse surface *and* bounces to next surface (if it survives Russian roulette)
  - Photons don't stick to specular surfaces
- Caustic LSS*D photon map
  - High resolution
  - Light source usually emits photons only in directions that hit the thing creating the caustic



Photon mapping

- [Jensen EGRW 95, 96]
  - The lower-left scene below contains glossy surfaces, and was rendered in 50 minutes using photon mapping. The same scene took 6 hours for render with Radiance that used radiosity for diffuse reflection and path tracing for glossy reflection.

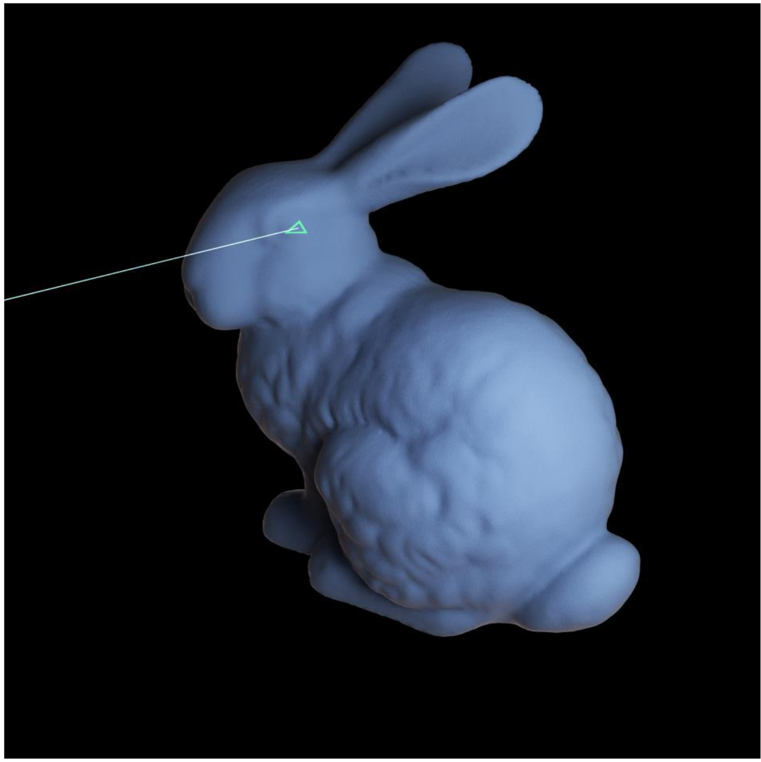



Real-time Ray Tracing

- Real-time ray tracing challenge

  運算最複雜：ray到三角形的距離、polygons之間的intersection
  =>用HW來做：用bounding box

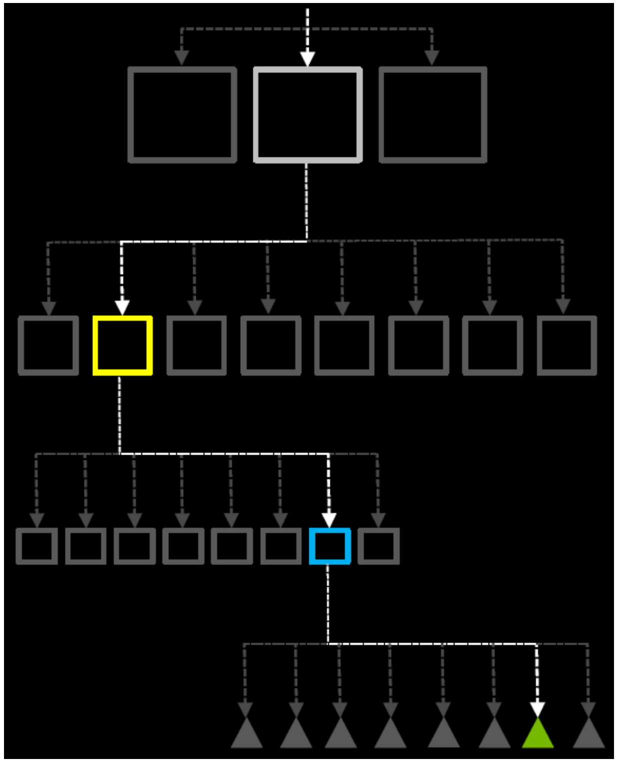
  - How to find the “needle” in the triangle data “haystack” ?

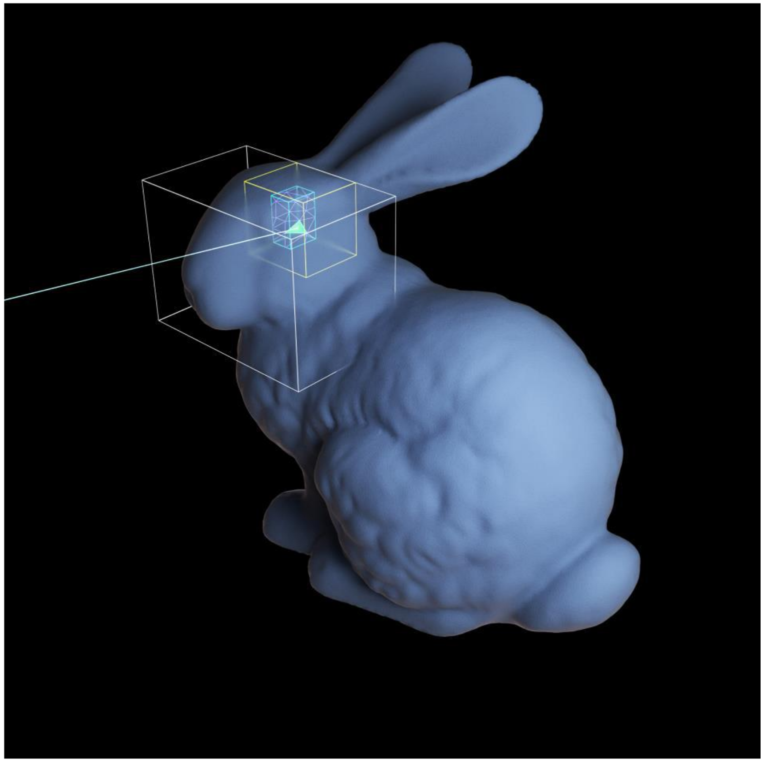


The following slides are extracted from Martin Stich, Real Time Raytracing with NVIDIA RTX.

Real-time Ray Tracing (cont.)

- Bounding volume hierarchy (BVH) traversal 像Octree

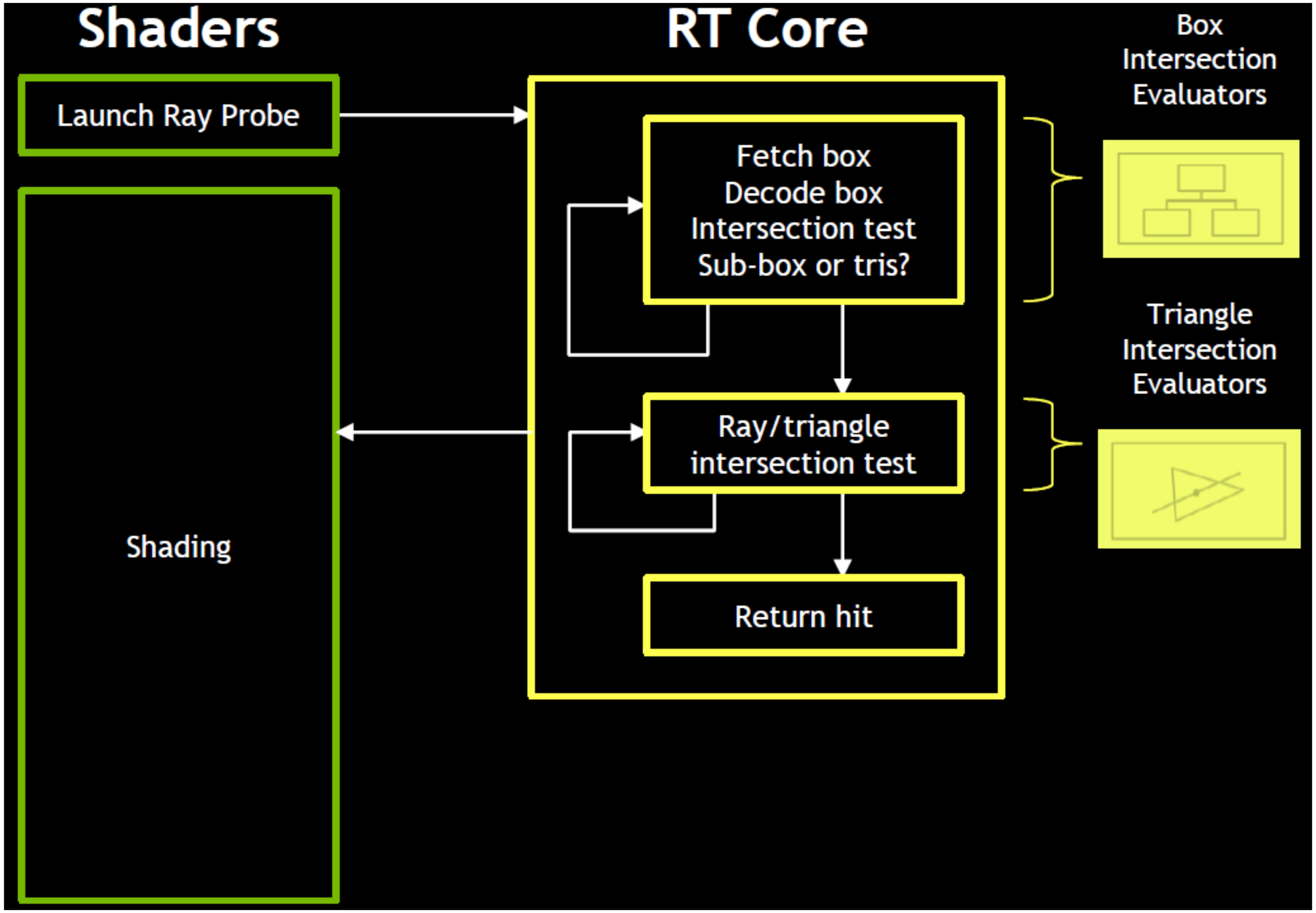




猜打到哪個三角形

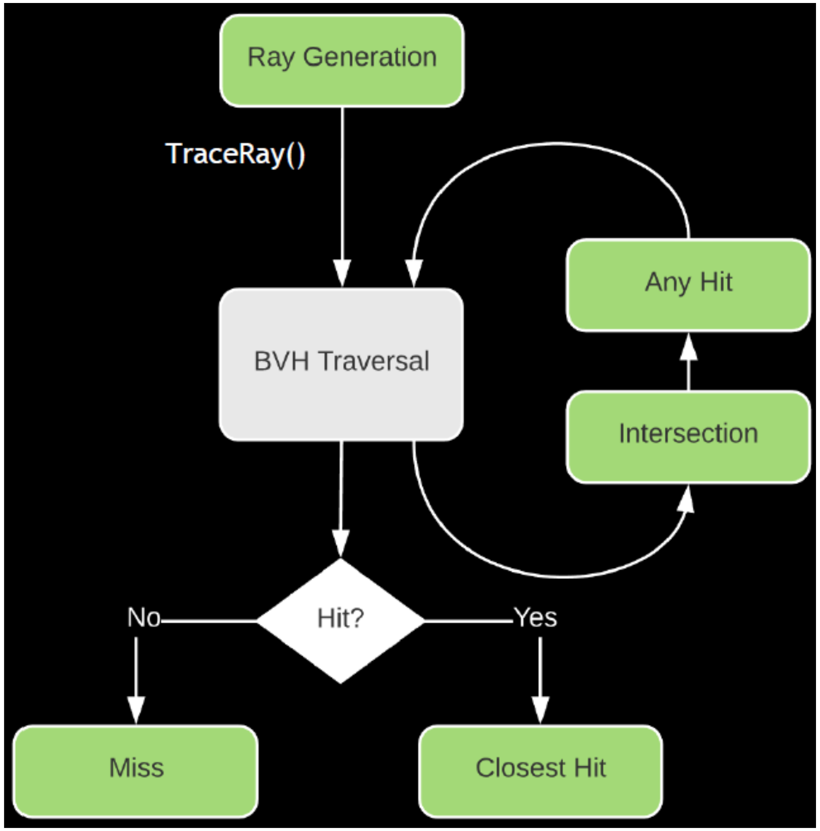
Real-time Ray Tracing (cont.)

- Ray tracing with RT cores ray trace core



Real-time Ray Tracing (cont.)

- Ray tracing pipeline
- New shaders for
  =>green part : 要寫的程式
  - Ray generation
  - Intersection
  - Any hit
  - Closest hit
  - ....



Real-time Ray Tracing (cont.)

每秒能處理的ray還是有限：每個畫面只射一個ray，
可以用train過的neural network、CNN(denoise)來補光線不夠的地方

- Even with hardware acceleration for evaluation, intersections, only a few rays can be casted for real time performance.
- How to generate realistic images with only a few rays?

The End of Chapter 10