Distribution Ray Tracing

Distribution Ray Tracing

use many rays to compute average values over pixel areas, time, area lights, reflected directions, …

Antialiasing Origin

- compute average color subtended by a pixel

pixel center $C = \text{raytrace}(\mathbf{E}, \mathbf{Q} - \mathbf{E})$

pixel average $C = \frac{1}{A_P} \cdot \int_{\mathbf{Q} \in P} \text{raytrace}(\mathbf{E}, \mathbf{Q} - \mathbf{E}) \cdot dA_{\mathbf{Q}}$

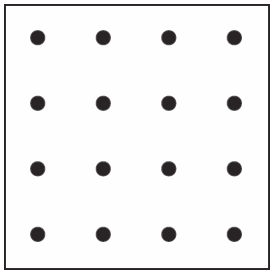
$\mathbf{E}$: camera origin

$\mathbf{Q}$: point on image plane

P : pixel of area A_P

Antialiasing by Deterministic Integration

- subdivide the pixel in squares
- cast rays through squares centers
- average result

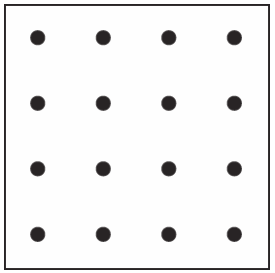


[Shirley]

Deterministic Antialiasing Pseudocode

- antialising pixel (i, j)

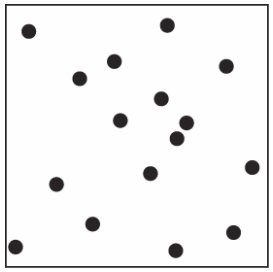
```
c = 0
for sx = 0 to ns
  for sy = 0 to ns
    u = (i + (sx+0.5)/ns) / width
    v = (j + (sj+0.5)/ns) / height
    Q = imagePlanePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns^2
```



[Shirley]

Antialiasing by Monte Carlo Estimation

- pick random points in pixel area
- cast rays through them
- average result

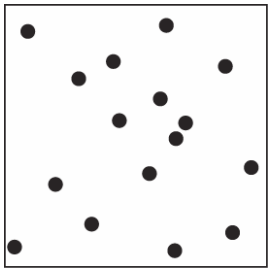


[Shirley]

Monte Carlo Antialiasing Pseudocode

- antialising pixel (i, j)

```
c = 0
for s = 0 to ns^2
    (rx,ry) = random2d();
    u = (i + rx) / width
    v = (j + ry) / height
    Q = imagePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns^2
```

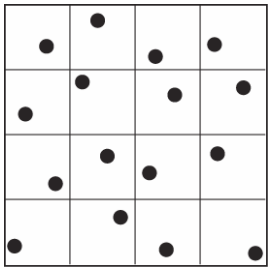


[Shirley]

Monte Carlo Antialiasing Pseudocode

- antialising pixel (i, j)

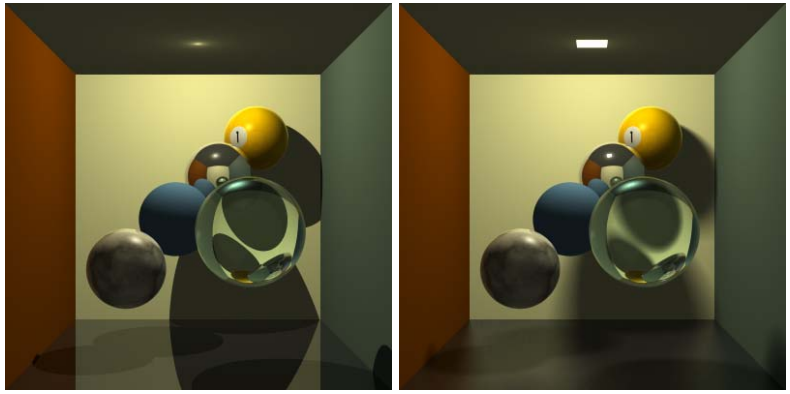
```
c = 0
for sx = 0 to ns
  for sy = 0 to ns
    (rx,ry) = random2d();
    u = (i + (sx+rx)/ns) / width
    v = (j + (sy+ry)/ns) / height
    Q = imagePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns^2
```



[Shirley]

Raytraced Images are too “Clean”

- soft shadows come from area light
  - raytracing only supports point lights



[Jason Waltman / jasonwaltman.com]

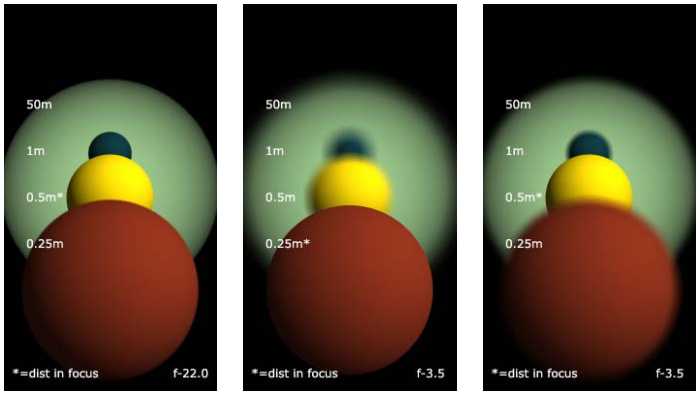
Raytraced Images are too “Clean”

- blurry reflections come from rough materials
  - raytracing only supports perfectly sharp mirrors

[Jensen]

Raytraced Images are too “Clean”

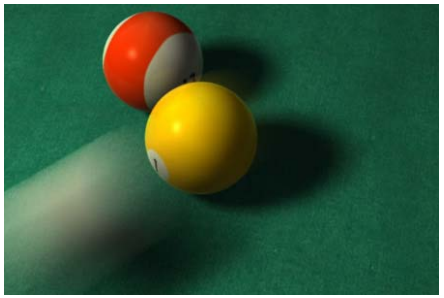
- depth of field come from lens system
  - raytracing only supports pinhole camera



[Jason Waltman / jasonwaltman.com]

Raytracer Images are too “Clean”

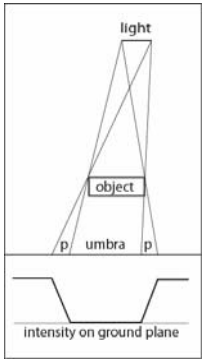
- motion blur come from shutter speed
  - raytracing only support infinitely fast shutter speed



[Jason Waltman / jasonwaltman.com]

Soft Shadows Origin

- area lights create penumbras
  - light is only partially visible from a given point
  - want to compute how much light hits the point



[Shirley]

Approximate Soft Shadows Principle

point light $$C = C_l \cdot V(\mathbf{P},\mathbf{S}) \cdot \text{shading}(\mathbf{P},\mathbf{S})$$

area light $$C = \frac{C_l}{A_L} \cdot \int_{\mathbf{S}\in L} V(\mathbf{P},\mathbf{S}) \cdot \text{shading}(\mathbf{P},\mathbf{S}) \cdot dA_{\mathbf{S}}$$

$\mathbf{P}$: point on the surface

$\mathbf{S}$: point on the light

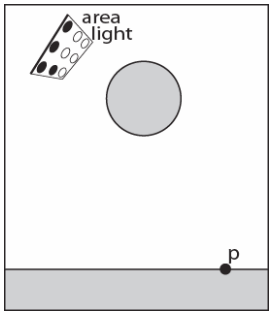
V : visibility function (0 or 1)

L : light of area A_L

C_l : total light intensity

Soft Shadows by Deterministic Integration

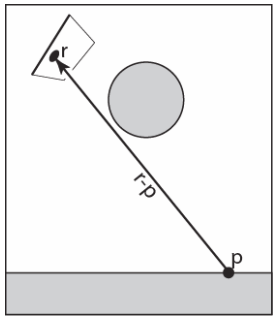
- approximate area light as a set of point lights
  - equivalent to quadrature rule
- for each point, compute shadows and lighting
- average results



[Shirley]

Soft Shadows by Monte Carlo Estimation

- use Monte Carlo integration
- pick random points on the light
  - easy for quad lights, hard (but possible) for others
- compute shadows and lighting
- average results



[Shirley]

Soft Shadows by Monte Carlo Estimation

for quads

$$\mathbf{S}_i = \mathbf{S}_c + (0.5 - r_{i,1})l\mathbf{u} + (0.5 - r_{i,2})l\mathbf{v}$$

$$\langle C \rangle = \frac{C_l}{N} \cdot \sum_i^N V(\mathbf{P}, \mathbf{S}_i) \cdot \text{shading}(\mathbf{P}, \mathbf{S}_i)$$

$\mathbf{S}_c$: light source center

$\mathbf{u}, \mathbf{v}$: light source tangent vectors

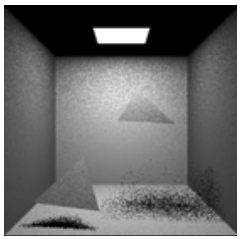
l : light source size

$\mathbf{r}_i$: uniformly sampled random 2d vector in $[0,1]^2$

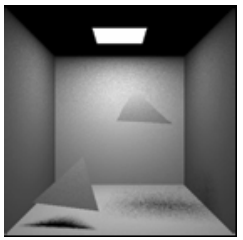
N : total number of samples

How Many Samples?

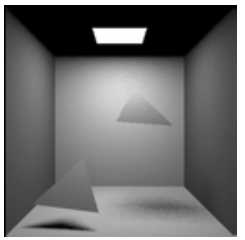
1 sample



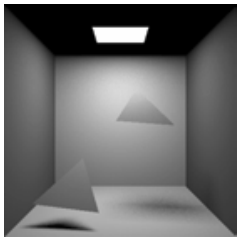
9 samples



36 samples



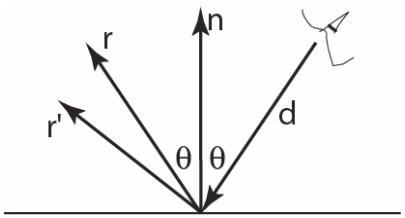
[Bala]



100 samples

Blurry Reflections Origin

- real materials often have blurred reflections
  - light is scattered around a set of directions
  - want to compute how much light reaches the surface along these directions



[Shirley]

Approximate Blurry Reflections Principle

mirror refl. $C = k_r \cdot \text{raytrace}(\mathbf{P}, \mathbf{R})$

blurry refl. $C = \frac{k_r}{A_\Omega} \cdot \int_{\mathbf{R} \in \Omega} \text{raytrace}(\mathbf{P}, \mathbf{R}) \cdot dA_{\mathbf{R}}$

$\mathbf{P}$: point on the surface

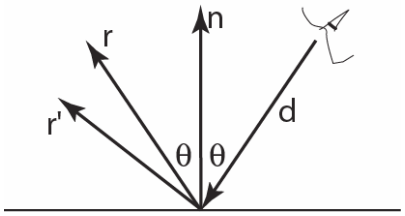
$\mathbf{R}$: reflected direction

Ω : solid angle where object is reflecting

k_r : reflection coefficient

Blurred Reflection by Monte Carlo Est.

- compute reflected direction **R**
- pick random direction **R'** around **R**
- compute reflected color along **R'**
- average results



[Shirley]

Blurred Reflection by Monte Carlo Est.

$$\mathbf{R}_i = \left[\mathbf{R} + (0.5 - r_{i,1})l\mathbf{u} + (0.5 - r_{i,2})l\mathbf{v}\right]/|\mathbf{R}_i|$$

for quads

$$\langle C \rangle = \frac{k_r}{N} \cdot \sum_i^N \text{raytrace}(\mathbf{P}, \mathbf{R}_i)$$

$\mathbf{R}$: reflected direction

$\mathbf{u}, \mathbf{v}$: vectors orthogonal to $\mathbf{R}$

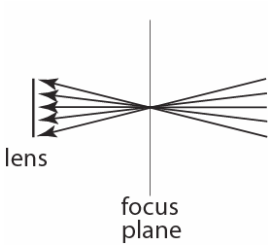
l : blur size

$\mathbf{r}_i$: uniformly sampled random 2d vector in $[0,1]^2$

N : total number of samples

Depth-of-Field Origin

- images in eyes and cameras are formed by lenses
  - not all rays converge to a point
  - so only some object appear sharp (in focus)
  - while all other objects do not (out of focus)



[Shirley]

Approximate Depth-of-Field Principle

pinhole $$C = \frac{1}{A_P} \int_{\mathbf{Q} \in A_P} \text{raytrace}(\mathbf{E}, \mathbf{Q} - \mathbf{E}) dA_{\mathbf{Q}}$$

dof $$C = \frac{1}{A_P A_L} \cdot \int_{\mathbf{Q} \in P} \int_{\mathbf{F} \in L} \text{raytrace}(\mathbf{F}, \mathbf{Q} - \mathbf{F}) \cdot dA_{\mathbf{F}} \cdot dA_{\mathbf{Q}}$$

$\mathbf{E}$: camera origin

$\mathbf{Q}$: point on image plane

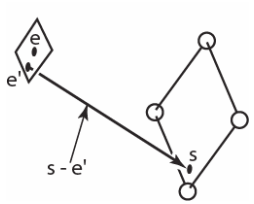
$\mathbf{F}$: point on "lens" plane around $\mathbf{E}$

P : pixel of area A_p

L : lens of area A_L

Depth-of-Field by Monte Carlo Estimation

- pick random points on film and image plane
- compute color from aperture point toward image one
- average results



[Shirley]

Depth-of-Field by Monte Carlo Estimation

$$\mathbf{F}_i = \mathbf{E} + (0.5 - s_{i,1}) l_a \mathbf{u} + (0.5 - s_{i,2}) l_a \mathbf{v}$$

for quads $$\mathbf{Q}_i = \mathbf{E} + (x + 0.5 - r_{i,1}) l_p \mathbf{u} + (y + 0.5 - r_{i,2}) l_p \mathbf{v} - n\mathbf{z}$$

$$\langle C \rangle = \frac{1}{N} \cdot \sum_i^N \text{raytrace}(\mathbf{F}_i, \mathbf{Q}_i - \mathbf{F}_i)$$

$\mathbf{u}, \mathbf{v}$: vectors parallel to image plane

l_P, l_L : pixel/aperture size

n : distance to image plane

$\mathbf{r}_i, \mathbf{s}_i$: uniformly sampled random 2d vectors in $[0,1]^2$

N : total number of samples

Motion Blur Origin

- takes time for images to form on camera/eye
  - during that time the camera/eye is open
  - sensors take the "average" of what's happening
  - but objects move around in that time

Approximate Motion Blur Principle

no blur $$C(t) = \text{raytrace}(\mathbf{E}, \mathbf{I}, S_t)$$

blurry refl. $$C(t, \Delta t) = \frac{1}{\Delta t} \cdot \int_{s \in [t, t+\Delta t]} \text{raytrace}(\mathbf{E}, \mathbf{I}, S_s) \cdot dl_s$$

$\mathbf{E}$: any ray origin

$\mathbf{I}$: any ray direction

S_t : scene at time t

t : time when shutter opens

Δt : time that shutter stays open

$\Delta t = t_{k+1} - t_k$: for animation, time between frames

Motion Blur by Monte Carlo Estimation

- for any ray (camera, shadow, reflection), pick random time in the shutter interval
- update object transformations for picked time
- compute intersection
- average results

Motion Blur by Monte Carlo Estimation

for aliased camera rays $$\langle C \rangle = \frac{1}{N} \cdot \sum_i^N \text{raytrace}(\mathbf{E}, \mathbf{Q} - \mathbf{E}, S_{t_i})$$

$\mathbf{E}$: camera origin

$\mathbf{Q}$: point on image plane

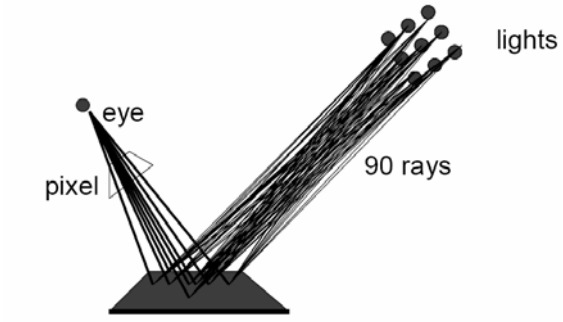
t_i : uniformly sampled random variable in [0,1]

N : total number of samples

Combining Estimations

- e.g. compute soft shadows and antialias the pixel

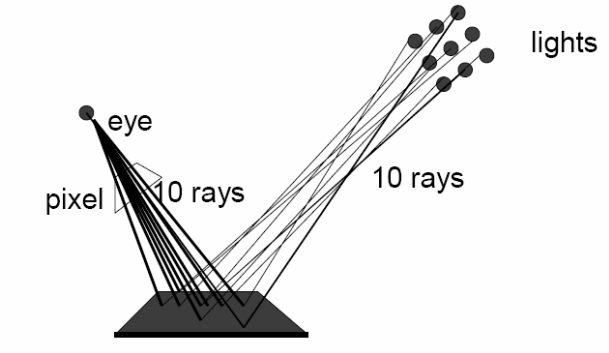
$$C = \frac{C_l}{A_P A_L} \cdot \int_{\mathbf{Q} \in A_P} \int_{\mathbf{S} \in A_L} V(\mathbf{P}(\mathbf{Q}), \mathbf{S}) \cdot \text{shading}(\mathbf{P}(\mathbf{Q}), \mathbf{S}) \cdot dA_{\mathbf{S}} \cdot dA_{\mathbf{Q}}$$



[Bala]

Combining Estimations: Path Tracing

- idea: *randomize the whole path*
  - compute all the integrals at once!



[Bala]