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

$E$ : camera origin; $Q$ : point on image plane; $P$ : pixel of area $A_p$
antialiasing by deterministic int.

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

[Shirley]
deterministic antialias pseudocode

// antialiasing pixel (i,j) - deterministic
c = 0
for sx = 0 to ns
  for sy = 0 to ns
    u = (i + (sx+0.5)/ns) / width
    v = (j + (sy+0.5)/ns) / height
    Q = imagePlanePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns*ns

[Shirley]
antialiasing by monte carlo estimation

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

[Shirley]
monte carlo antialiasing pseudocode

// antialiasing pixel (i,j) - monte carlo

\[ \text{c} = 0 \]  
\[ \text{for s = 0 to ns*ns} \]  
\[ (rx,ry) = \text{random2d()} \]  
\[ u = (i + rx) / \text{width} \]  
\[ v = (i + ry) / \text{height} \]  
\[ Q = \text{imagePoint}(u,v) \]  
\[ c += \text{raytrace}(E,Q-E) \]  
\[ c /= \text{ns*ns} \]
monte carlo antialiasing pseudocode

// antialiasing pixel (i,j) - stratified monte carlo

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) / ns*ns
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 mirror

[Jensen]
raytraced images are too "clean"

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

[Jason Waltman / jasonwaltman.com]
raytraced 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
  - $C = C_l \cdot V(P, S) \cdot \text{shading}(P, S)$

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

$P$: point on the surface; $S$: 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 int.

- 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 est.

- 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
soft shadows by monte carlo est.

for quads

- \( S_i = S_c + (0.5 - r_{i,1})lu + (0.5 - r_{i,2})lv \)
- \( \langle C \rangle = \frac{C_l}{N} \cdot \sum_i^N V(P, S_i) \cdot \text{shading}(P, S_i) \)

\( S_c \) : light source center; \( u, v \) : light source tangent vectors; \( l \) : light source size; \( r_i \) : 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]
approx. blurry reflections principle

- mirror reflection
  \[ C = k_r \cdot \text{raytrace}(P, R) \]
- blurry reflection
  \[ C = \frac{k_r}{A_\Omega} \cdot \int_{R \in \Omega} \text{raytrace}(P, R) \cdot dA_R \]

\( P \): point on the surface; \( R \): reflected direction; \( \Omega \): solid angle where object is reflecting; \( k_r \): reflection coefficient
blurred reflection by monte carlo est.

- compute reflected direction \( \mathbf{R} \)
- pick random direction \( \mathbf{R}' \) around \( \mathbf{R} \)
- compute reflected color along \( \mathbf{R}' \)
- average results
blurred reflection by monte carlo est.

for quads

- \( \mathbf{R}_i = \frac{\mathbf{R} + (0.5-r_{i,1})\mathbf{u} + (0.5-r_{i,2})\mathbf{v}}{|\mathbf{R}_i|} \)
- \( \langle C \rangle = \frac{k_r}{N} \cdot \sum_{i}^{N} \text{raytrace}(\mathbf{P}, \mathbf{R}_i) \)

\( \mathbf{R} \): reflected direction; \( \mathbf{u}, \mathbf{v} \): vectors orthogonal to \( \mathbf{R} \); \( l \): blur size; \( r_i \): 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)
approximate depth-of-field principle

- pinhole
  \[ C = \frac{1}{A_P} \cdot \int_{Q \in P} \text{raytrace}(E, Q - E) \cdot dA_Q \]

- dof
  \[ C = \frac{1}{A_P A_L} \cdot \int_{Q \in P} \int_{F \in L} \text{rt}(F, Q - F) \cdot dA_F \cdot dA_Q \]

\( E \): camera origin; \( Q \): point on image plane; \( F \): point on "lens" plane around \( E \); \( P \): pixel of area \( A_P \); \( L \): lens of area \( A_L \)
depth-of-field by monte carlo est.

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

[Shirley]
depth-of-field by monte carlo est.

for quads

- $F_i = E + (0.5 - s_{i,1})l_a u + (0.5 - s_{i,2})l_a v$
- $Q_i = E + (x + 0.5 - r_{i,1})l_p u + (y + 0.5 - r_{i,2})l_p v - nz$
- $\langle C \rangle = \frac{1}{N} \cdot \sum_{i}^{N} \text{raytrace}(F_i, Q_i - F_i)$

$u, v$ : vectors parallel to image plane; $l_p, l_a$ : pixel / aperture size; $n$ : distance to image plane; $r_i, s_i$ : 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 is happening
  - but objects move around in that time
approximate motion blur principle

- no blur
  - \( C(t) = \text{raytrace}(E, d, S_t) \)
- motion blur
  - \( C(t, \Delta t) = \frac{1}{\Delta t} \cdot \int_{s \in [t, t+\Delta t]} \text{raytrace}(E, d, S_s) \cdot dl_s \)

\( E \) : any ray origin; \( d \) : any ray direction; \( S_t \) : scene at time \( t \); \( t \) : time when shutter opens; \( \Delta t \) : 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

- $\langle C \rangle = \frac{1}{N} \cdot \sum_{i}^{N} \text{raytrace}(E, Q - E, S_{t_{i}})$

$E$ : camera origin; $Q$ : 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

\[ C = \frac{C_l}{A_{P \cdot A_L}} \cdot \int_Q \int_S V(P(Q), S) \cdot \text{sh}(P(Q), S) \cdot dA_S \cdot dA_Q \]

\[ Q \in A_P, S \in A_L \]
combining estimations: path tracing

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