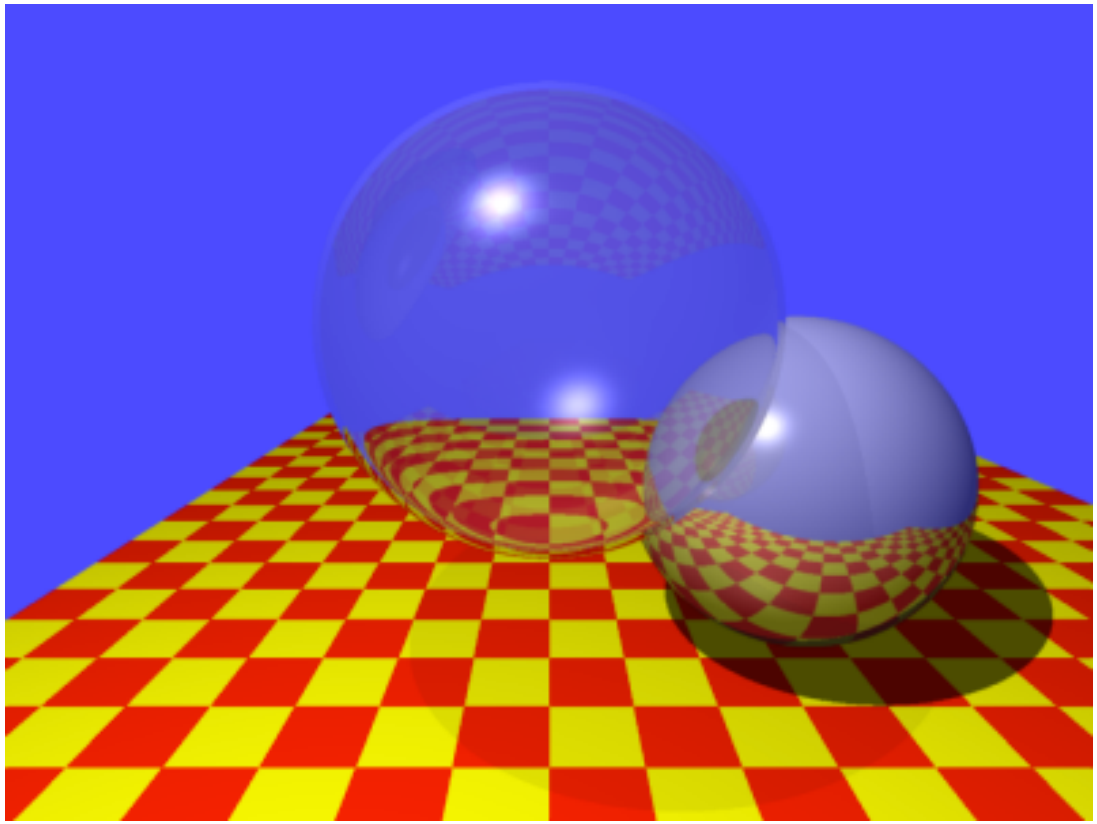
12 - raytracing

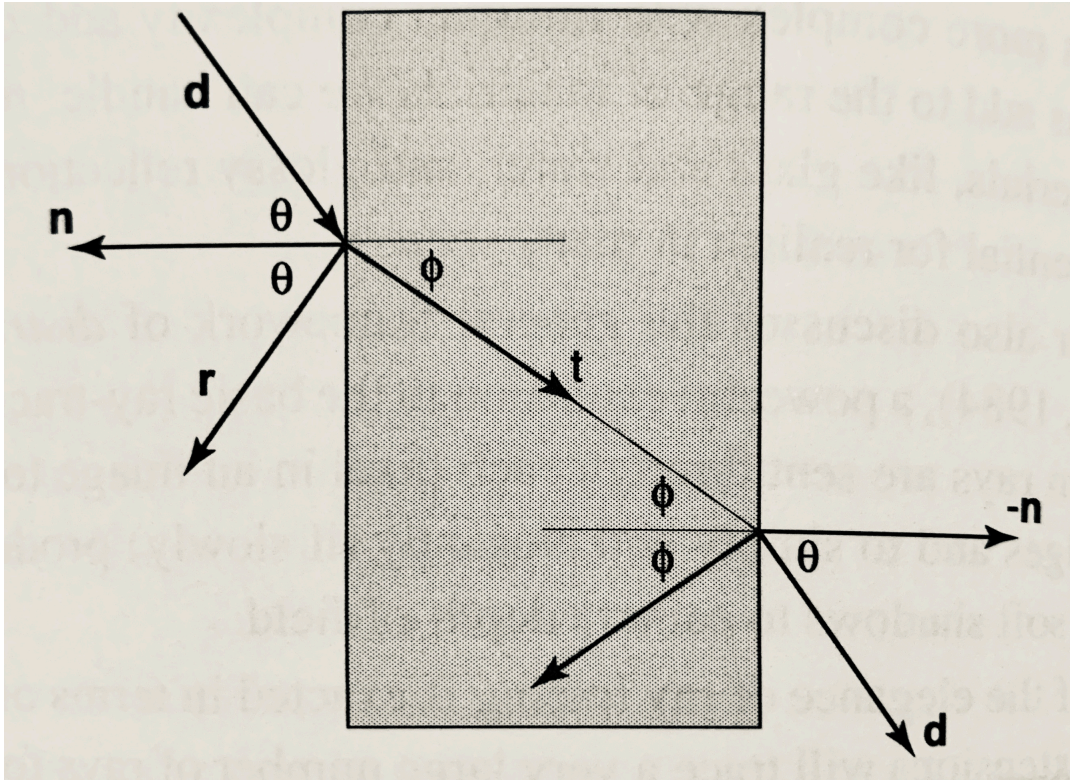
Making things look better

- Transparency and Refraction
- Constructive Solid Geometry
- Distribution Ray Tracing

Transparency and Refraction

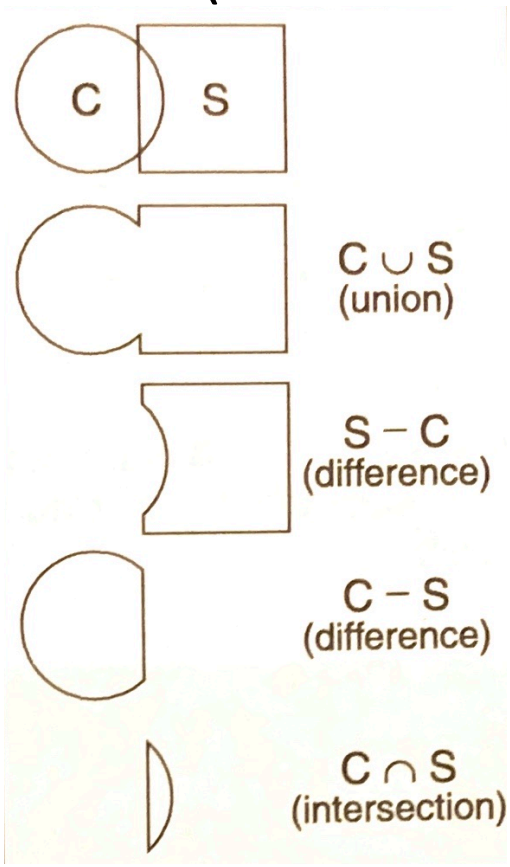


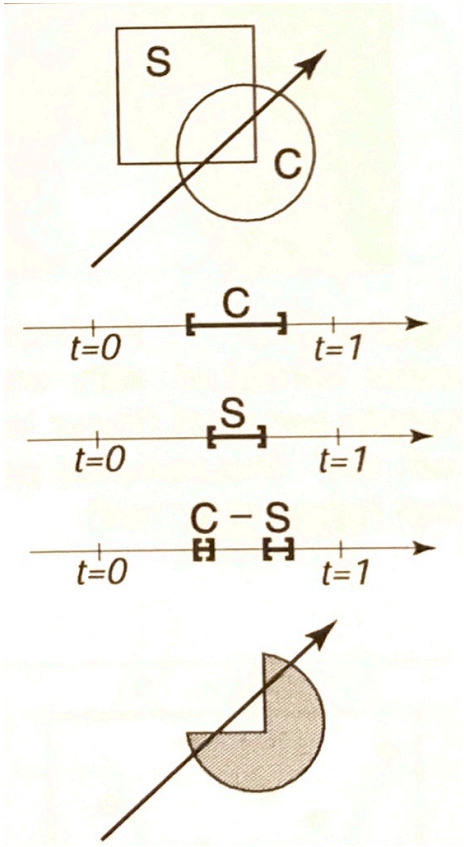
Diffraction and Reflection



from *Alan Watt, 3D Computer Graphics, 3rd Edition, 1999*

CSG (Constructive Solid Geometry)



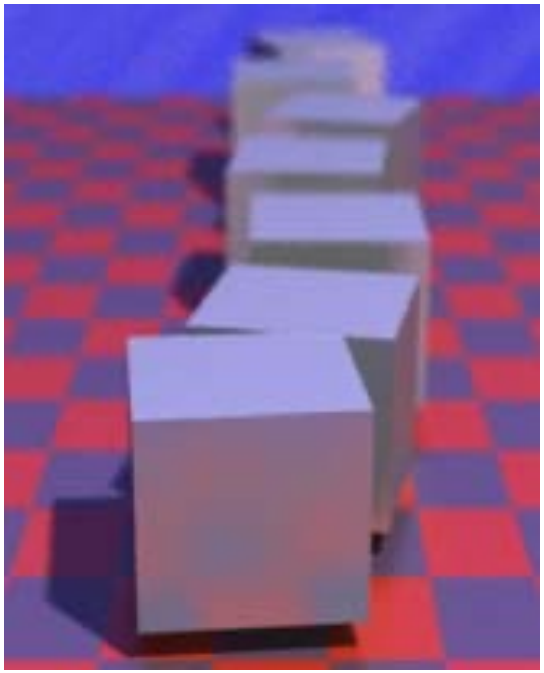


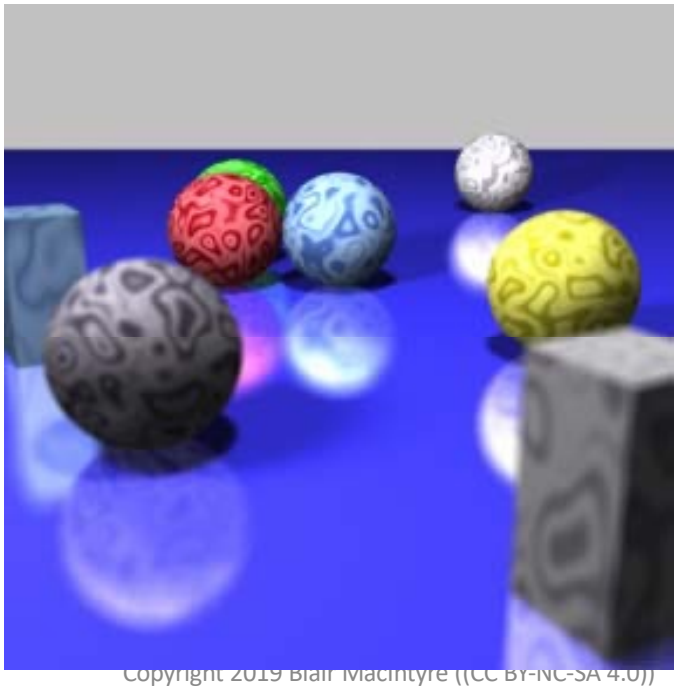
TinkerCad

https://www.tinkercad.com

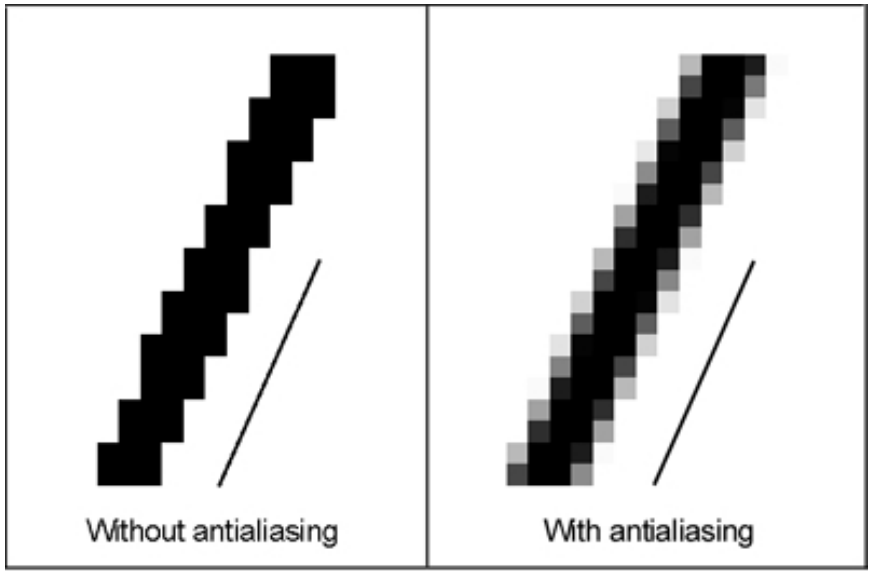
Distribution Ray Tracing

Use many rays to compute average values over pixel ares, time, area lights, reflected directions, …





Anti-Aliasing



antialiasing origin

- compute average color subtended by a pixel

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

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

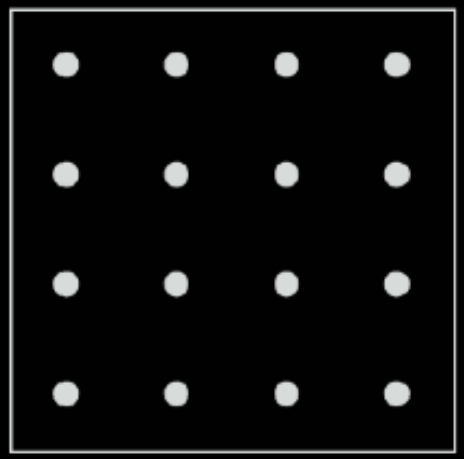
$\mathbf{E}$: camera origin

$\mathbf{Q}$: 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

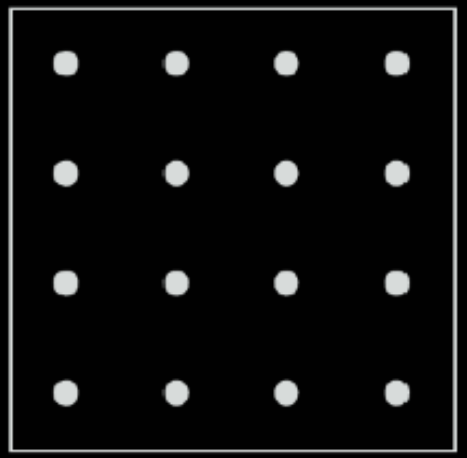


[Shirley]

deterministic antialiasing pseudocode

- antialising pixel (i, j)

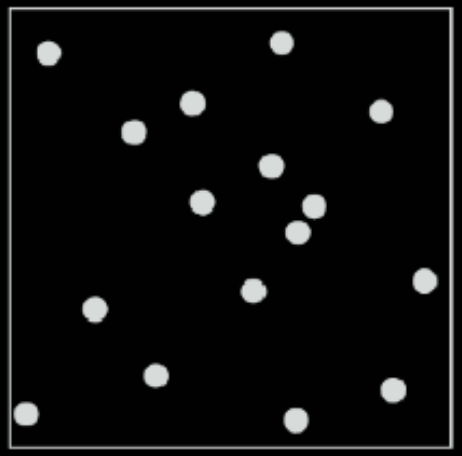
```
c = 0
for sx = 0 to ns
  for sy = 0 to ns
    u = (i + (sx+0.5)/ns) / width
    v = (j + (sj+0.5)/ns) / height
    Q = imagePlanePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns²
```



[Shirley]

antialiasing by Monte Carlo estimation

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

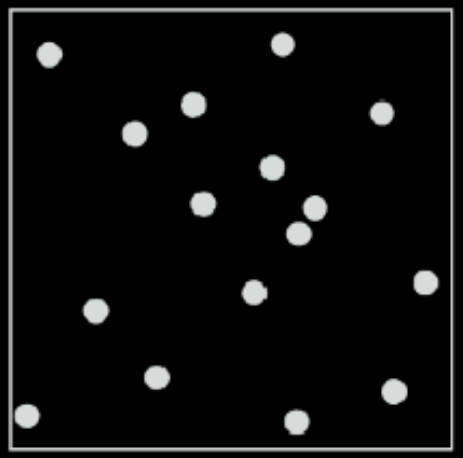


[Shirley]

Monte Carlo antialiasing pseudocode

- antialising pixel (i, j)

```
c = 0
for s = 0 to ns²
    (rx,ry) = random2d();
    u = (i + rx) / width
    v = (j + ry) / height
    Q = imagePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns²
```

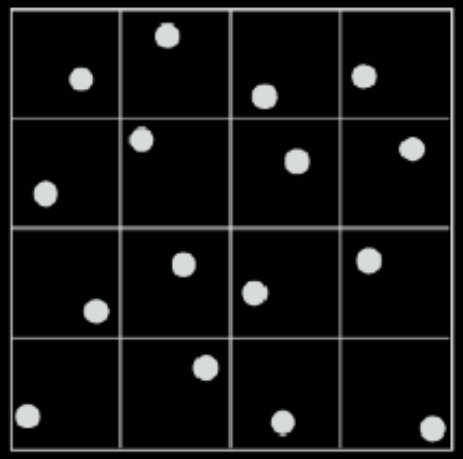


[Shirley]

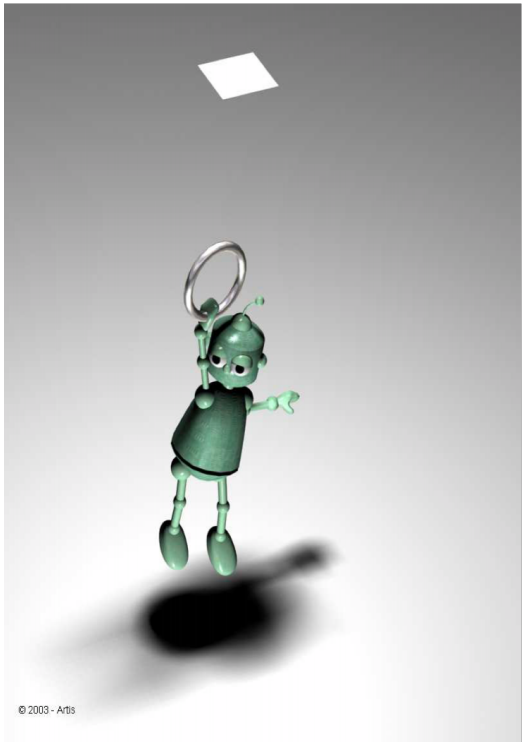
Monte Carlo antialiasing pseudocode

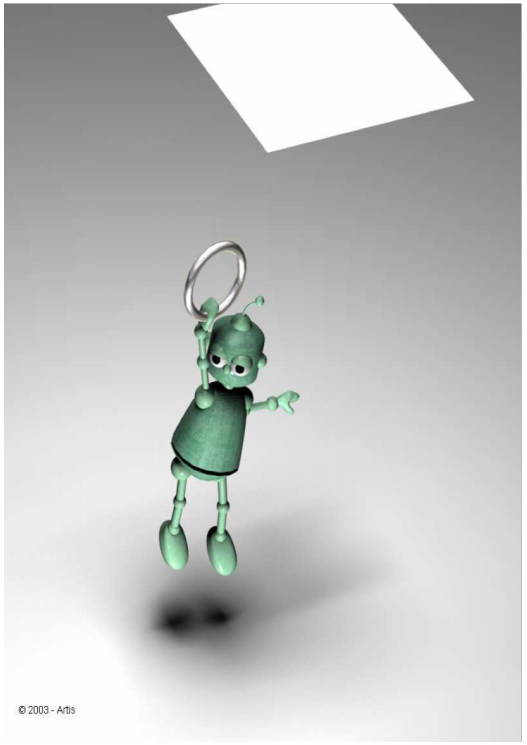
- antialising pixel (i, j)

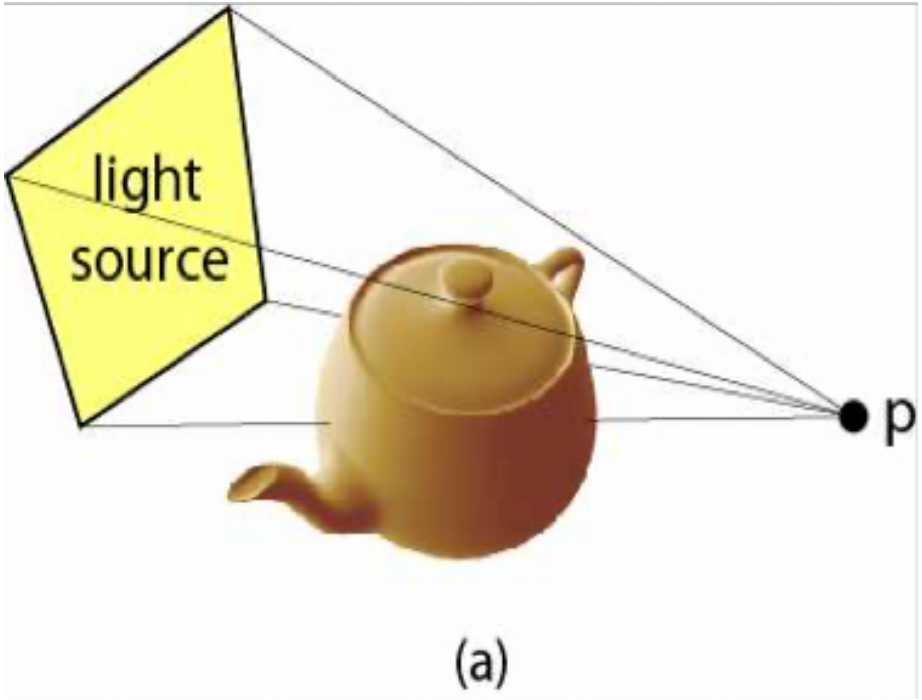
```
c = 0
for sx = 0 to ns
  for sy = 0 to ns
    (rx,ry) = random2d();
    u = (i + (sx+rx)/ns) / width
    v = (j + (sy+ry)/ns) / height
    Q = imagePoint(u,v)
    c += raytrace(E,Q-E)
c /= ns^2
```

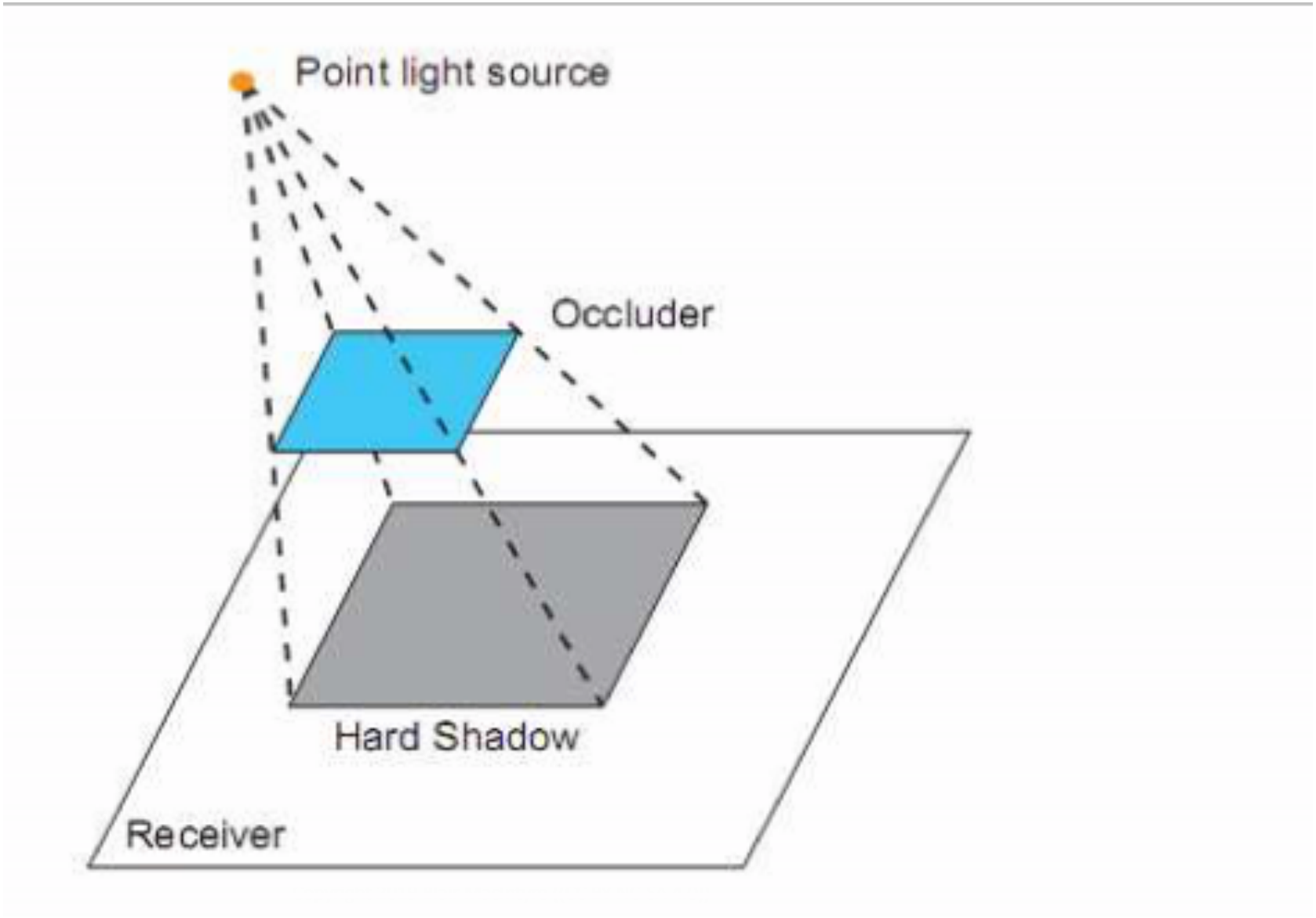


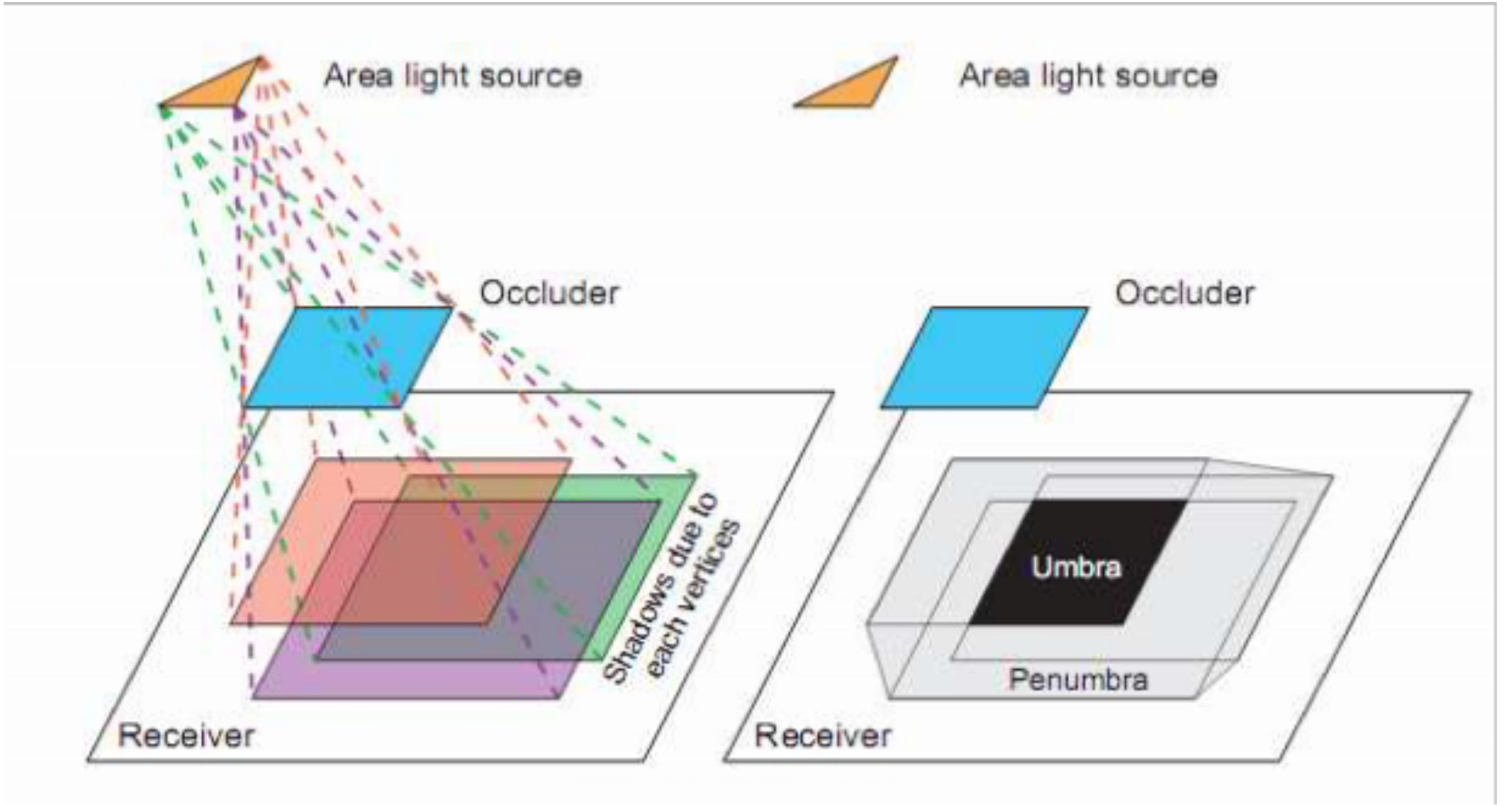
[Shirley]





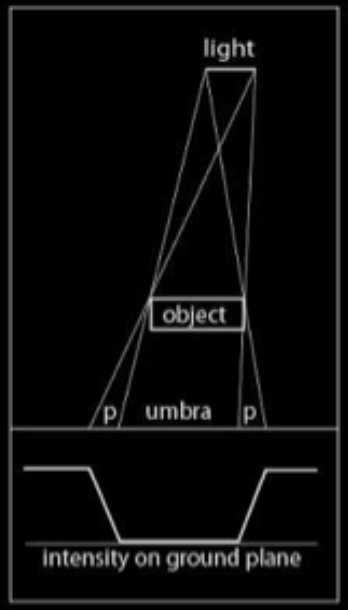



Point light source
Occluder
Hard Shadow
Receiver


Area light source
Occluder
Shadows due to each vertices
Receiver
Area light source
Occluder
Umbra
Penumbra
Receiver

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

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

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

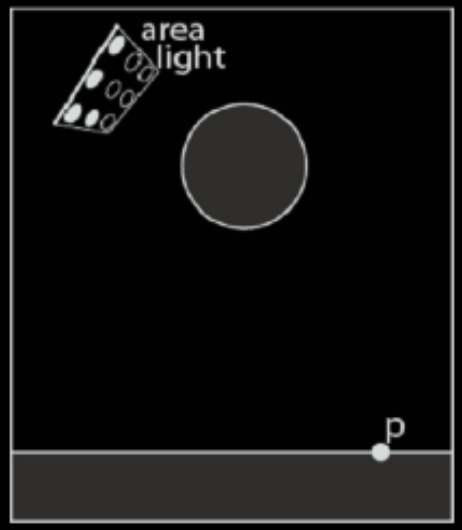
V : visibility function (0 or 1)

L : light of area A_L

C_l : total light intensity

soft shadows by deterministic integration

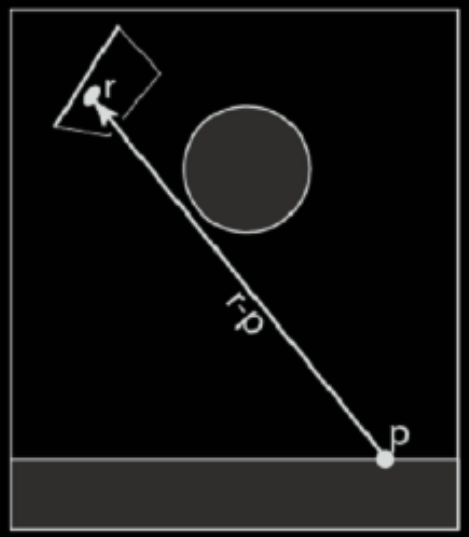
- approximate area light as a set of point lights
  - equivalent to quadrature rule
- for each point, compute shadows and lighting
- average results



[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

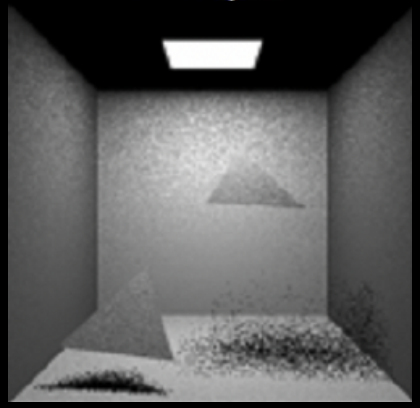
l : light source size

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

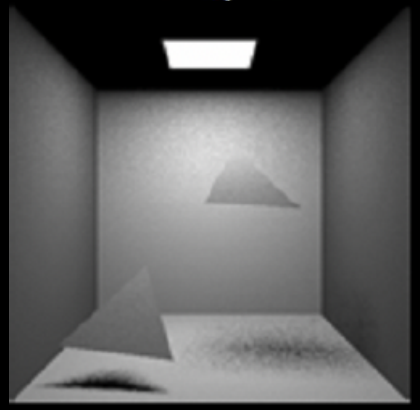
N : total number of samples

how many samples?

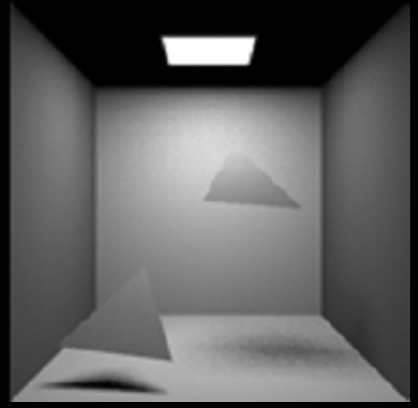
1 sample

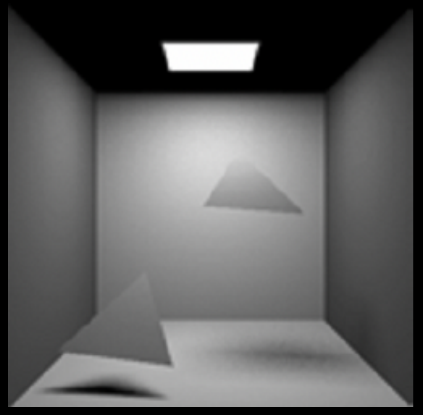


9 samples



36 samples





[Bala]