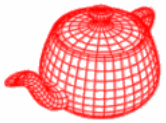
Volume and Participating Media

Digital Image Synthesis

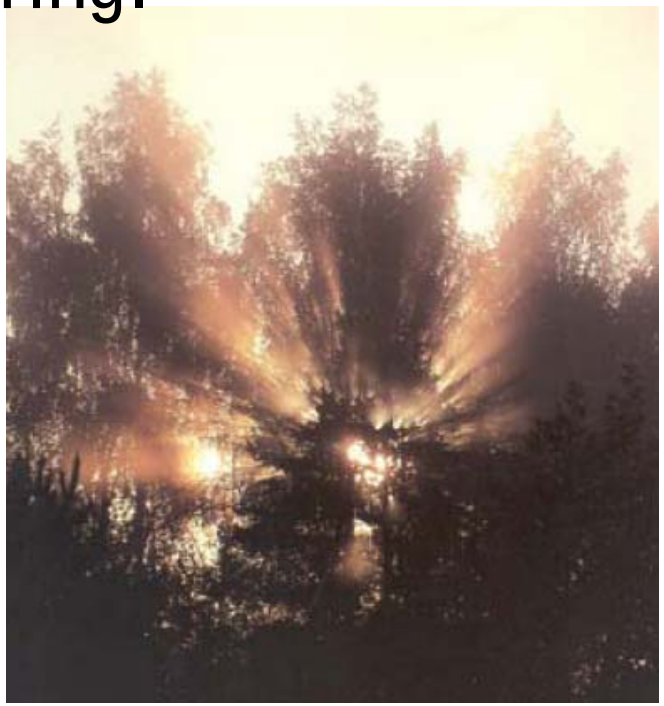
Yung-Yu Chuang

with slides by Pat Hanrahan and Torsten Moller

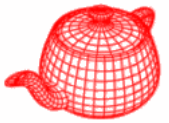
Participating media



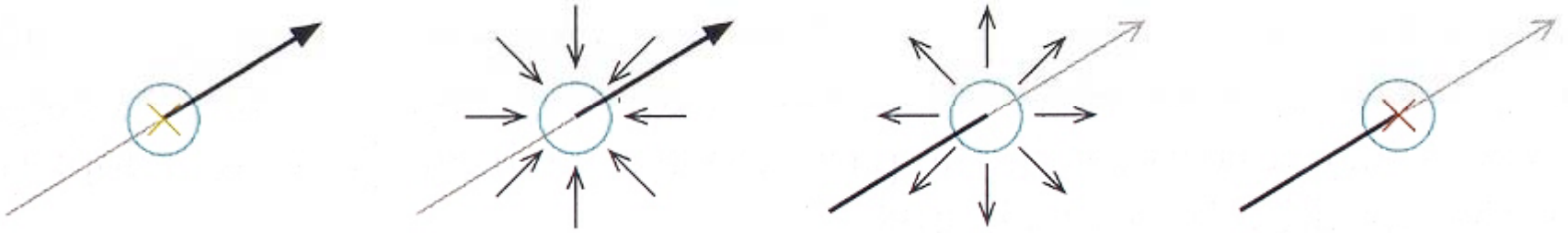
- We have by far assumed that the scene is in a vacuum. Hence, radiance is constant along the ray. However, some real-world situations such as fog and smoke attenuate and scatter light. They participate in rendering.
- Natural phenomena
  - Fog, smoke, fire
  - Atmosphere haze
  - Beam of light through clouds
  - Subsurface scattering



Volume scattering processes

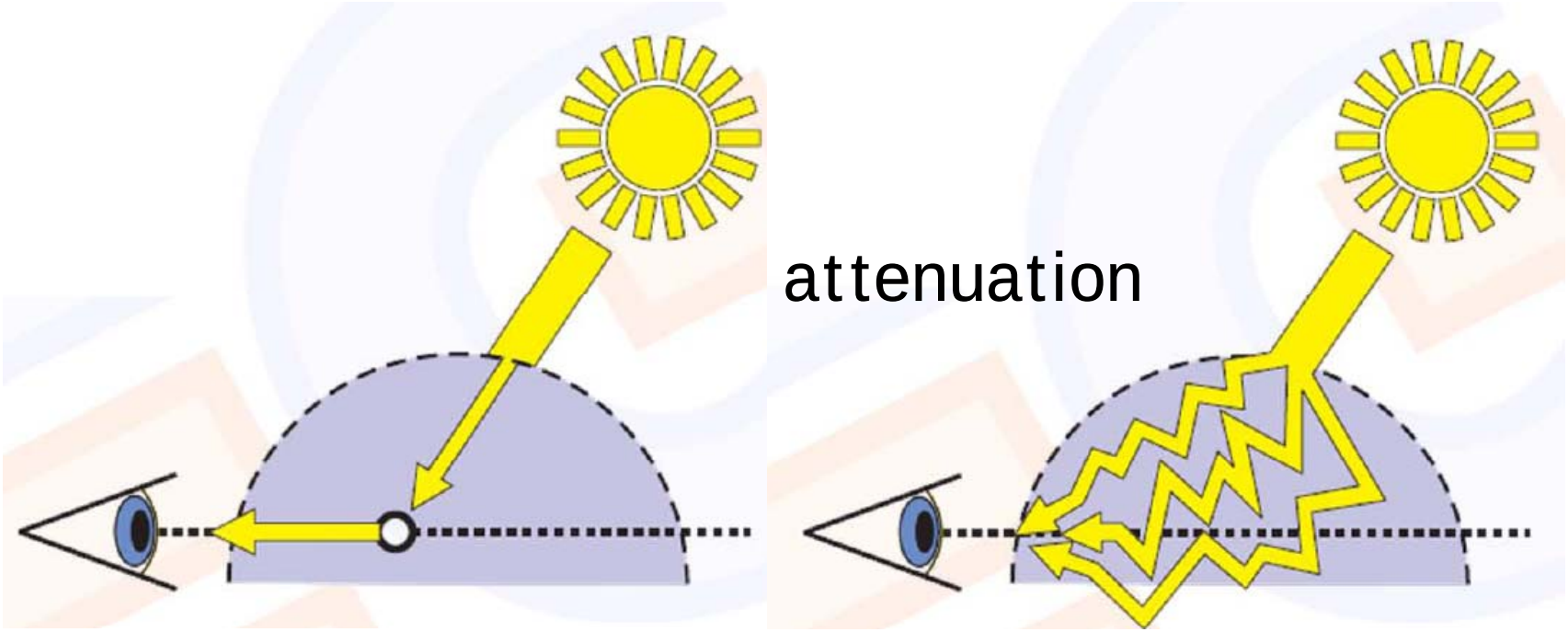


- Absorption (conversion from light to other forms)
- Emission (contribution from luminous particles)
- Scattering (direction change of particles)
  - Out-scattering
  - In-scattering
  - Single scattering v.s. multiple scattering
- Homogeneous v.s. inhomogeneous(heterogeneous)



emission in-scattering out-scattering absorption

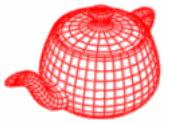
Single scattering and multiple scattering



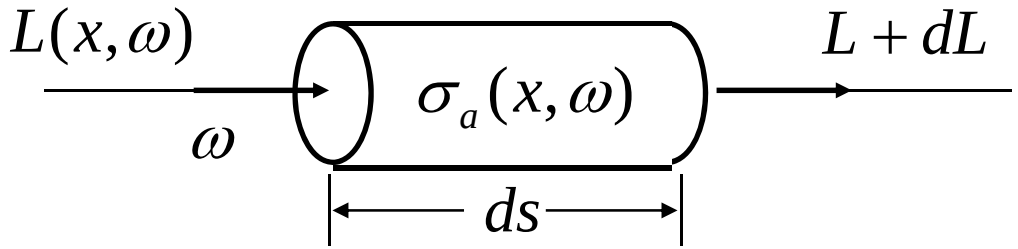
single scattering

multiple scattering

Absorption



The reduction of energy due to conversion of light to another form of energy (e.g. heat)

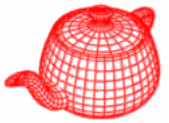


$$dL(x,\omega) = -\sigma_a(x,\omega)L(x,\omega)ds$$

Absorption cross-section: $\sigma_a(x,\omega)$

Probability of being absorbed per unit length

Transmittance



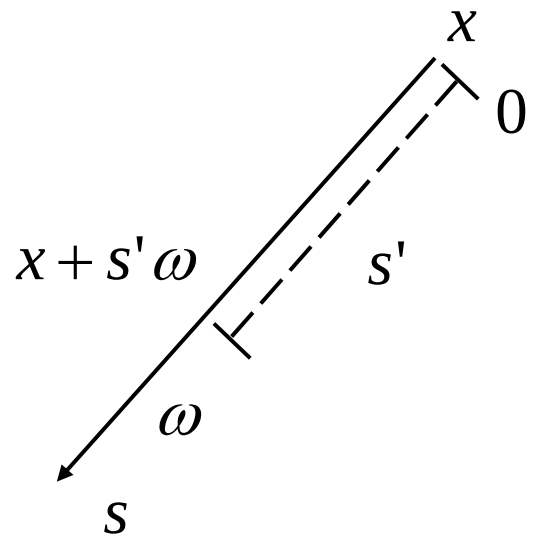
$$dL(x,\omega) = -\sigma_a(x,\omega)L(x,\omega)ds$$

$$\frac{dL(x,\omega)}{L(x,\omega)} = -\sigma_a(x,\omega)ds \longrightarrow \int_x^{x+s\omega} \frac{dL(x',\omega)}{L(x',\omega)} = -\int_0^s \sigma_a(x+s'\omega,\omega)ds'$$

$$\ln L(x+s\omega,\omega) - \ln L(x,\omega) = -\int_0^s \sigma_a(x+s'\omega,\omega)ds' = -\tau_\omega(s)$$

Optical distance or depth

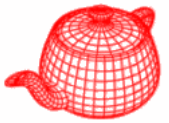
$$\tau_\omega(s) = \int_0^s \sigma_a(x+s'\omega,\omega)ds'$$



Homogenous media: constant σ_a

$$\sigma_a \rightarrow \tau(s) = \sigma_a s$$

Transmittance and opacity

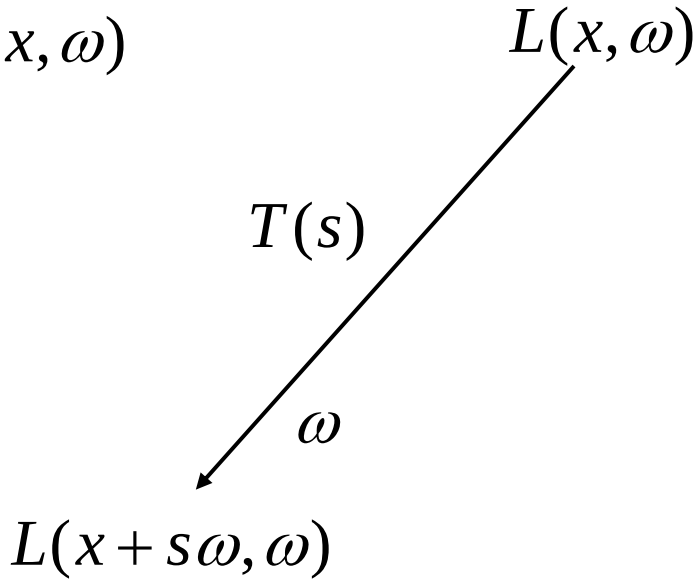


$$dL(x,\omega) = -\sigma_a(x,\omega)L(x,\omega)ds$$

$$\frac{dL(x,\omega)}{L(x,\omega)} = -\sigma_a(x,\omega)ds \longrightarrow \int_x^{x+s\omega} \frac{dL(x',\omega)}{L(x',\omega)} = -\int_0^s \sigma_a(x+s'\omega,\omega)ds'$$

$$\ln L(x+s\omega,\omega) - \ln L(x,\omega) = -\int_0^s \sigma_a(x+s'\omega,\omega)ds' = -\tau_\omega(s)$$

$$L(x+s\omega,\omega) = e^{-\tau_\omega(s)}L(x,\omega) = T_\omega(s)L(x,\omega)$$



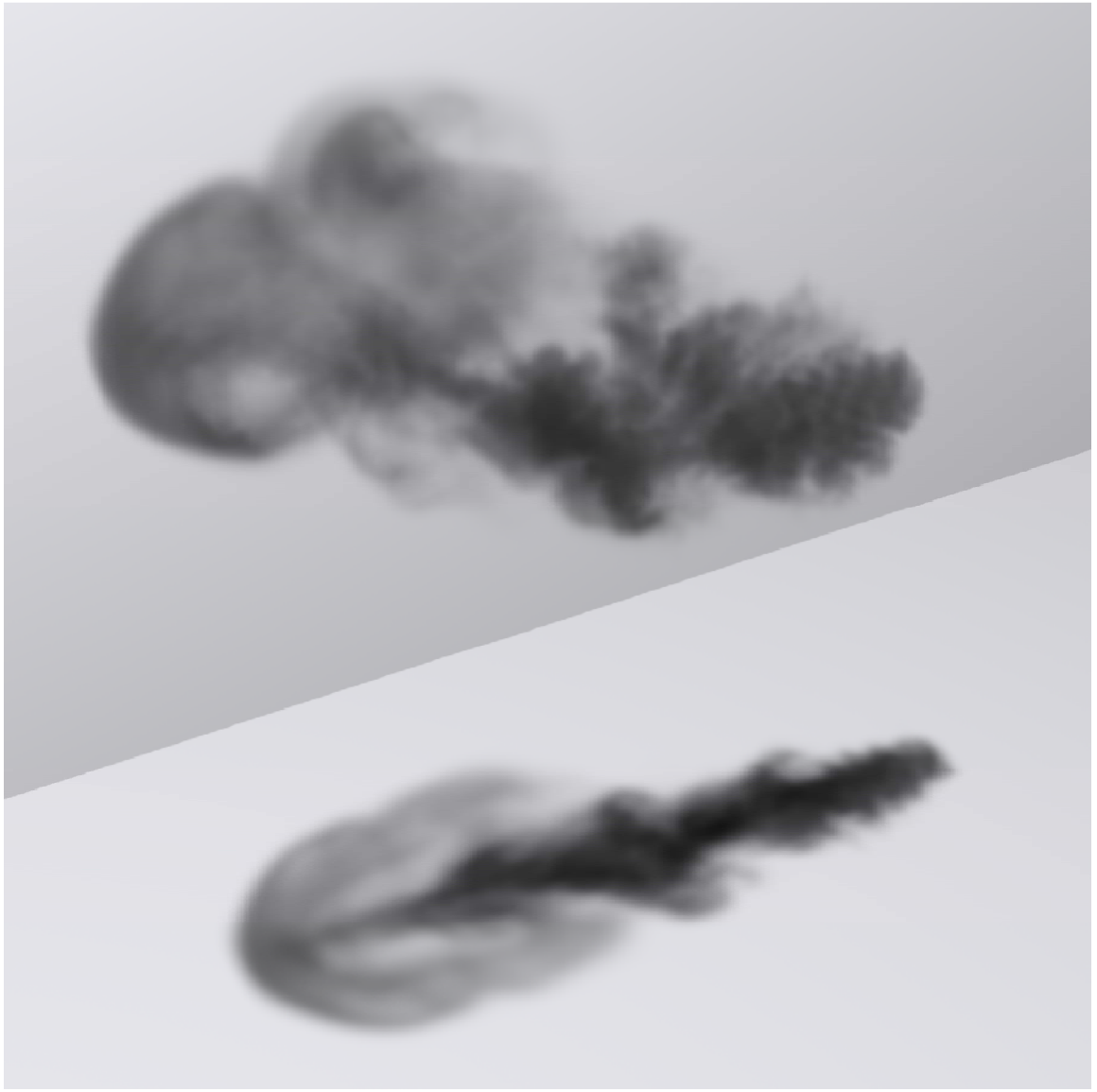
Transmittance

$$T_\omega(s) = e^{-\tau_\omega(s)}$$

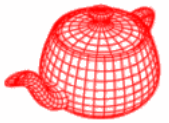
Opacity

$$\alpha_\omega(s) = 1 - T_\omega(s)$$

Absorption

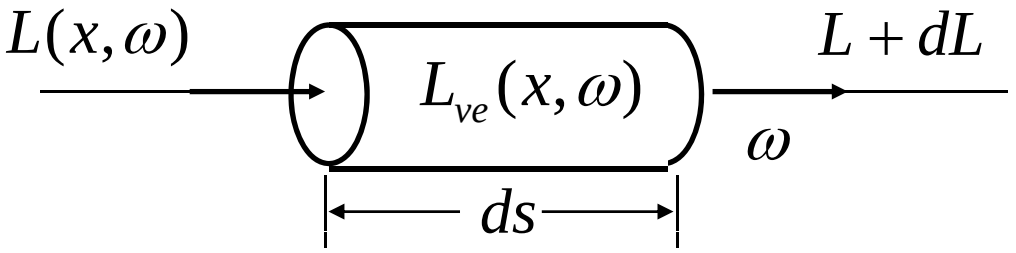


Emission

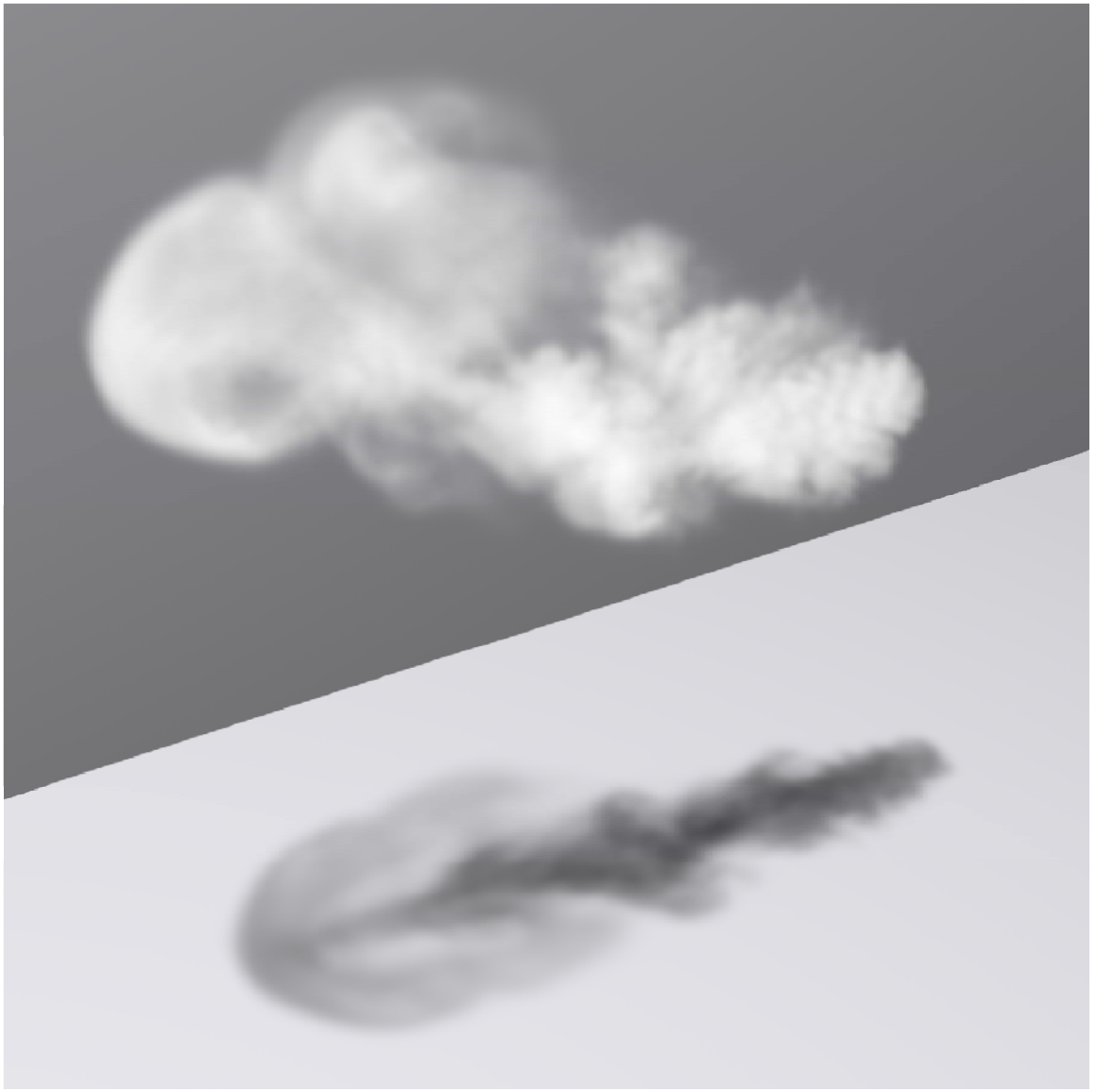


- Energy that is added to the environment from luminous particles due to chemical, thermal, or nuclear processes that convert energy to visible light.
- $L_{ve}(x,\omega)$: emitted radiance added to a ray per unit distance at a point x in direction ω

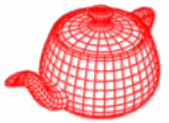
$$dL(x,\omega) = L_{ve}(x,\omega)ds$$



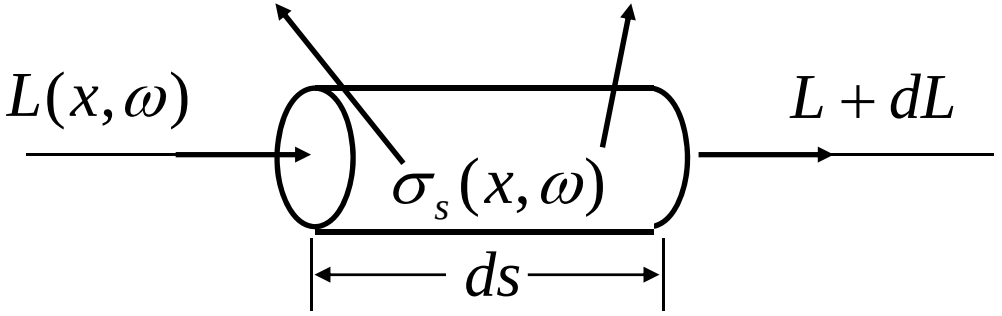
Emission



Out-scattering



Light heading in one direction is scattered to other directions due to collisions with particles

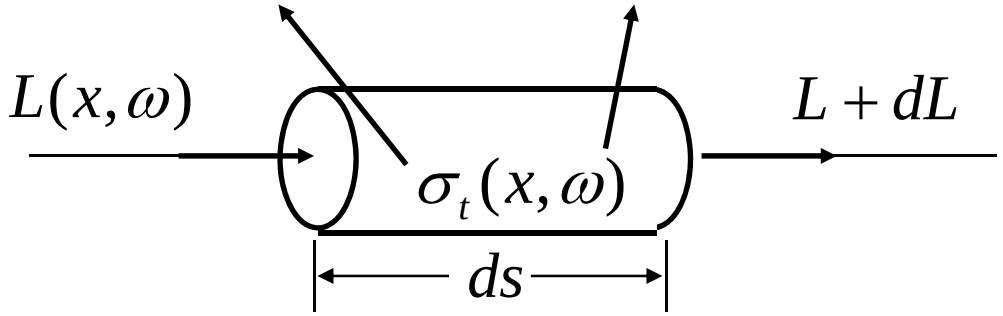


$$dL(x,\omega) = -\sigma_s(x,\omega)L(x,\omega)ds$$

Scattering cross-section: σ_s

Probability of being scattered per unit length

Extinction



$$dL(x,\omega) = -\sigma_t(x,\omega)L(x,\omega)ds$$

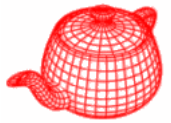
Total cross-section $\sigma_t = \sigma_a + \sigma_s$

Albedo $W = \frac{\sigma_s}{\sigma_t} = \frac{\sigma_s}{\sigma_a + \sigma_s}$

Attenuation due to both absorption and scattering

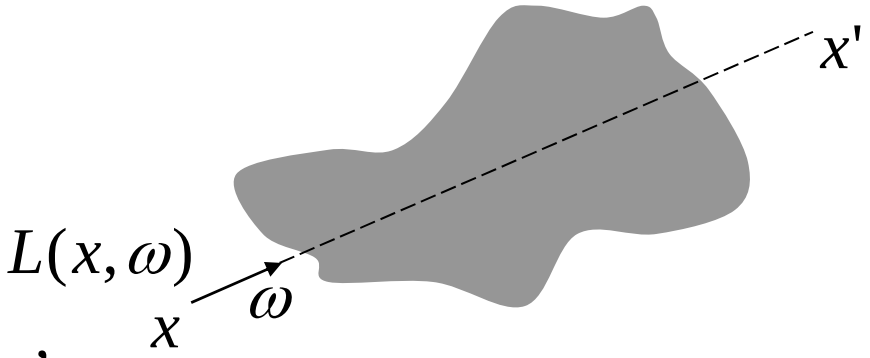
$$\tau_\omega(s) = \int_0^s \sigma_t(x + s'\omega, \omega)ds'$$

Extinction



- Beam transmittance

$$Tr(x \to x') = e^{-\int_0^s \sigma_t(x+s'\omega,\omega)ds'}$$

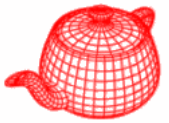


s: distance between x and x'

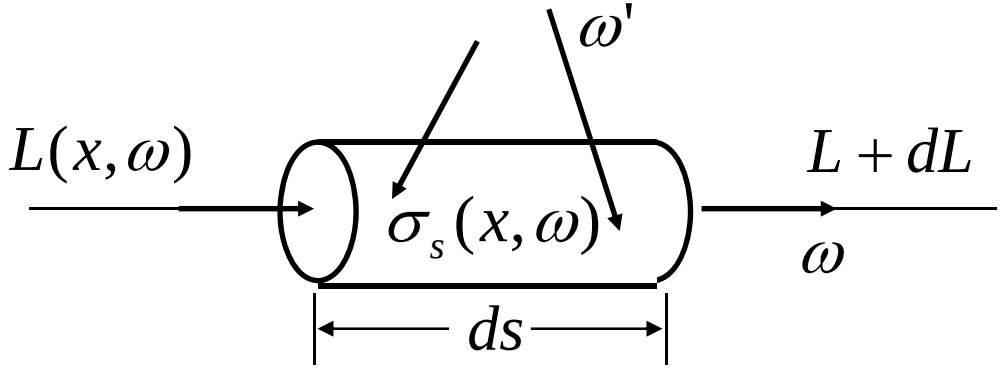
- Properties of Tr:
- In vaccum $Tr(x \to x') = 1$
- Multiplicative $Tr(x \to x'') = Tr(x \to x') \cdot Tr(x' \to x'')$
- Beer's law (in homogeneous medium)

$$Tr(x \to x') = e^{-\sigma_t s}$$

In-scattering



Increased radiance due to scattering from other directions



$$dL(x,\omega) = \left[\sigma_s(x,\omega) \int_{\Omega} p(x,\omega' \rightarrow \omega) L(x,\omega') d\omega' \right] ds$$

Phase function $p(\omega' \rightarrow \omega)$

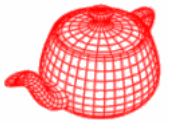
Reciprocity

$$p(\omega \rightarrow \omega') = p(\omega' \rightarrow \omega)$$

Energy conserving

$$\int_{S^2} p(\omega' \rightarrow \omega)\, d\omega' = 1$$

Source term

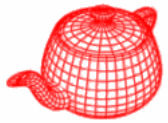


$$S(x,\omega) = L_{ve}(x,\omega) + \sigma_s(x,\omega)\int_{\Omega} p(x,\omega' \rightarrow \omega)L(x,\omega')d\omega'$$

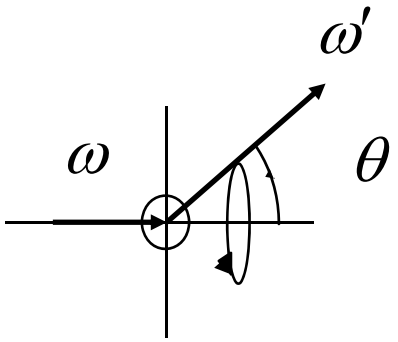
$$dL(x,\omega) = S(x,\omega)ds$$

- S is determined by
  - Volume emission
  - Phase function which describes the angular distribution of scattered radiation (volume analog of BSDF for surfaces)

Phase functions



Phase angle $\cos\theta = \omega \bullet \omega'$



Phase functions

(from the phase of the moon)

1. Isotropic $p(\cos\theta) = \frac{1}{4\pi}$

- simple

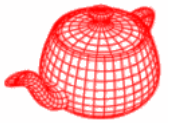
2. Rayleigh $p(\cos\theta) = \frac{3}{4}\frac{1+\cos^2\theta}{\lambda^4}$

- Molecules (useful for very small particles whose radii smaller than wavelength of light)

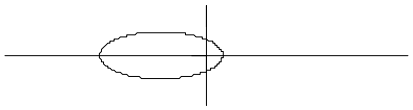
3. Mie scattering

- small spheres (based on Maxwell's equations; good model for scattering in the atmosphere due to water droplets and fog)

Henyey-Greenstein phase function

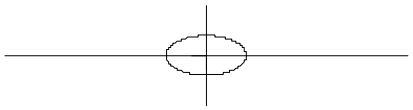


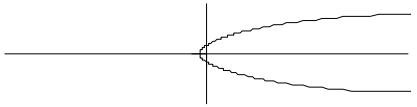
Empirical phase function



g = -0.3

$$p(\cos\theta) = \frac{1}{4\pi}\frac{1-g^2}{\left(1+g^2-2g\cos\theta\right)^{3/2}}$$



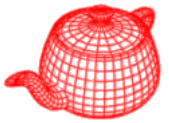


g = 0.6

$$2\pi\int_0^{\pi} p(\cos\theta)\cos\theta\, d\theta = g$$

g: **average phase angle**

Henyey-Greenstein approximation



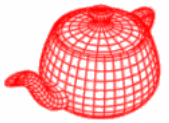
- Any phase function can be written in terms of a series of Legendre polynomials (typically, n<4)

$$p(\cos\theta) = \frac{1}{4\pi}\sum_{n=0}^{\infty}(2n+1)b_n P_n(\cos\theta)$$

$$b_n =< p(\cos\theta), P_n(\cos\theta) >$$
$$= \int_{-1}^{1} p(\cos\theta)P_n(\cos\theta)d\cos\theta$$

$$P_0(x) = 1$$
$$P_1(x) = x$$
$$P_2(x) = \frac{1}{2}(3x^2 - 1)$$
$$P_3(x) = \frac{1}{2}(5x^3 - 3x)$$
$$\ldots$$

Schlick approximation

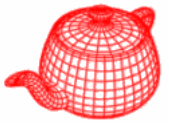


- Approximation to Henyey-Greenstein

$$p_{Schlick}(\cos\theta) = \frac{1}{4\pi}\frac{1-k^2}{(1-k\cos\theta)^2}$$

- K plays a similar role like g
  - 0: isotropic
  - -1: back scattering
  - Could use $k = 1.55g - 0.55g^2$

Importance sampling for HG

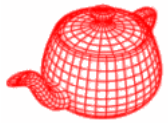


$$p(\cos\theta) = \frac{1}{4\pi}\frac{1-g^2}{\left(1+g^2-2g\cos\theta\right)^{3/2}}$$

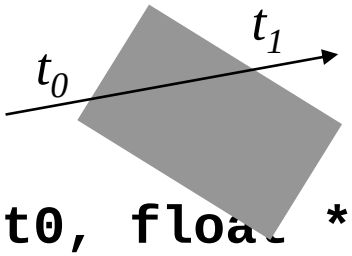
$$\phi = 2\pi\xi$$

$$\cos\theta = \begin{cases} 1-2\xi & \text{if } g = 0 \\ -\dfrac{1}{|2g|}\left(1+g^2-\left(\dfrac{1-g^2}{1-g+2g\xi}\right)^2\right) & \text{otherwise} \end{cases}$$

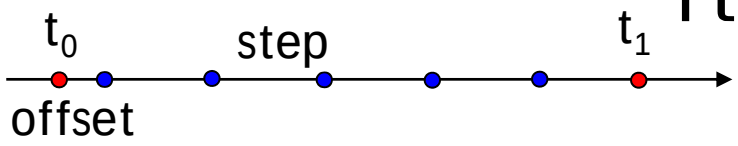
Pbrt implementation



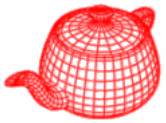
- **core/volume.* volume/***




```
class VolumeRegion {
public:
  bool IntersectP(Ray &ray, float *t0, float *t1);
  Spectrum sigma_a(Point &, Vector &);
  Spectrum sigma_s(Point &, Vector &);
  Spectrum Lve(Point &, Vector &);
  // phase functions: pbrt has isotropic, Rayleigh,
  // Mie, HG, Schlick
  virtual float p(Point &, Vector &, Vector &);
  // attenuation coefficient; s_a+s_s
  Spectrum sigma_t(Point &, Vector &);
  // calculate optical thickness by Monte Carlo or
  // closed-form solution
  Spectrum Tau(Ray &ray, float step=1.,
               float offset=0.5);
};
```



Homogenous volume

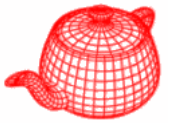


- Determined by (constant)
  - σ_s and σ_a
  - g in phase function
  - Emission L_{ve}
  - Spatial extent

```
Spectrum Tau(Ray &ray, float, float){
  float t0, t1;
  if (!IntersectP(ray,&t0,&t1))
    return 0.;
  return Distance(ray(t0),ray(t1)) * (sig_a + sig_s);
}
```

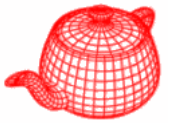
Homogenous volume

Varying-density volumes



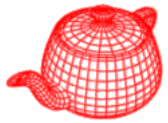
- Density is varying in the medium and the volume scattering properties at a point is the product of the density at that point and some baseline value.
- **`DensityRegion`**
  - 3D grid, **`VolumeGrid`**
  - Exponential density, **`ExponentialDensity`**

DensityRegion

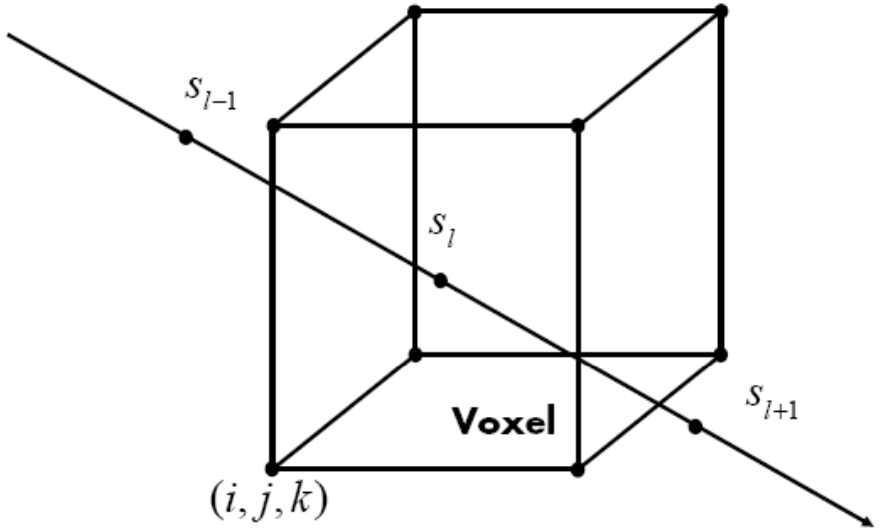


```
class DensityRegion : public VolumeRegion {
public:
  DensityRegion(Spectrum &sig_a, Spectrum &sig_s,
    float g, Spectrum &Le, Transform &VolumeToWorld);
  float Density(Point &Pobj) const = 0;
  sigma_a(Point &p, Vector &) {
    return Density(WorldToVolume(p)) * sig_a; }
  Spectrum sigma_s(Point &p, Vector &) {
    return Density(WorldToVolume(p)) * sig_s; }
  Spectrum sigma_t(Point &p, Vector &) {
    return Density(WorldToVolume(p))*(sig_a+sig_s);}
  Spectrum Lve(Point &p, Vector &) {
    return Density(WorldToVolume(p)) * le; }
  ...
protected:
  Transform WorldToVolume;
  Spectrum sig_a, sig_s, le;
  float g;
};
```

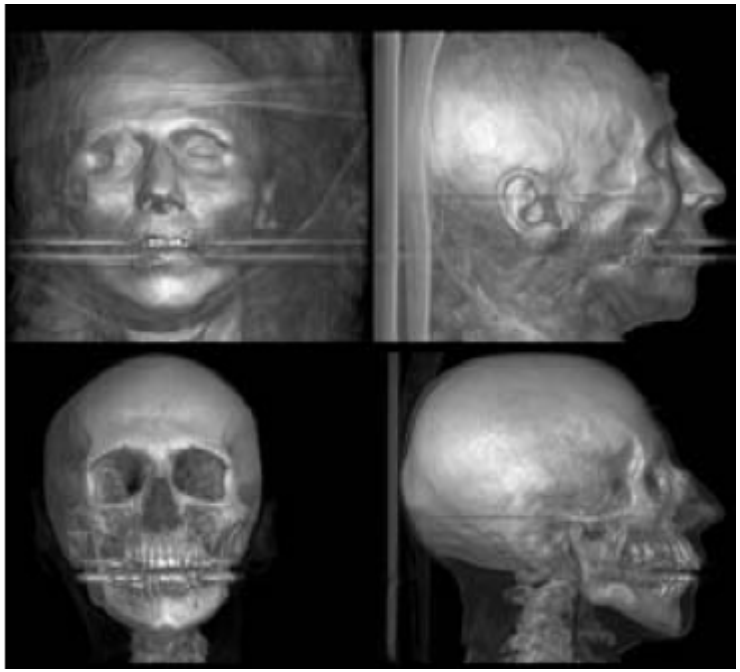
VolumeGrid



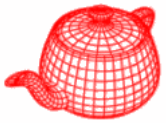
- Standard form of given data
- Tri-linear interpolation of data to give continuous volume
- Often used in volume rendering



Interpolation $v(s_l) = \text{trilinear}(v, i, j, k, x(s_l))$

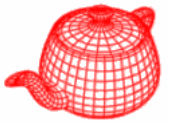


VolumeGrid



```
VolumeGrid(Spectrum &sa, Spectrum &ss, float gg,
           Spectrum &emit, BBox &e, Transform &v2w,
           int nx, int ny, int nz, const float *d);

float VolumeGrid::Density(const Point &Pobj) const {
  if (!extent.Inside(Pobj)) return 0;
  // Compute voxel coordinates and offsets
  float voxx = (Pobj.x - extent.pMin.x) /
      (extent.pMax.x - extent.pMin.x) * nx - .5f;
  float voxy = (Pobj.y - extent.pMin.y) /
      (extent.pMax.y - extent.pMin.y) * ny - .5f;
  float voxz = (Pobj.z - extent.pMin.z) /
      (extent.pMax.z - extent.pMin.z) * nz - .5f;
```

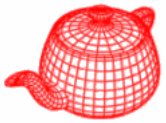
VolumeGrid



```
  int vx = Floor2Int(voxx);
  int vy = Floor2Int(voxy);
  int vz = Floor2Int(voxz);
  float dx = voxx - vx, dy = voxy - vy, dz = voxz - vz;
  // Trilinearly interpolate density values
  float d00 = Lerp(dx, D(vx, vy, vz),   D(vx+1, vy, vz));
  float d10 = Lerp(dx, D(vx, vy+1, vz), D(vx+1, vy+1, vz));
  float d01 = Lerp(dx, D(vx, vy, vz+1), D(vx+1, vy, vz+1));
  float d11 = Lerp(dx, D(vx, vy+1,vz+1),D(vx+1,vy+1,vz+1));
  float d0 = Lerp(dy, d00, d10);
  float d1 = Lerp(dy, d01, d11);
  return Lerp(dz, d0, d1);
}
```

```
float D(int x, int y, int z) {
  x = Clamp(x, 0, nx-1);
  y = Clamp(y, 0, ny-1);
  z = Clamp(z, 0, nz-1);
  return density[z*nx*ny+y*nx+x];
}
```

Exponential density

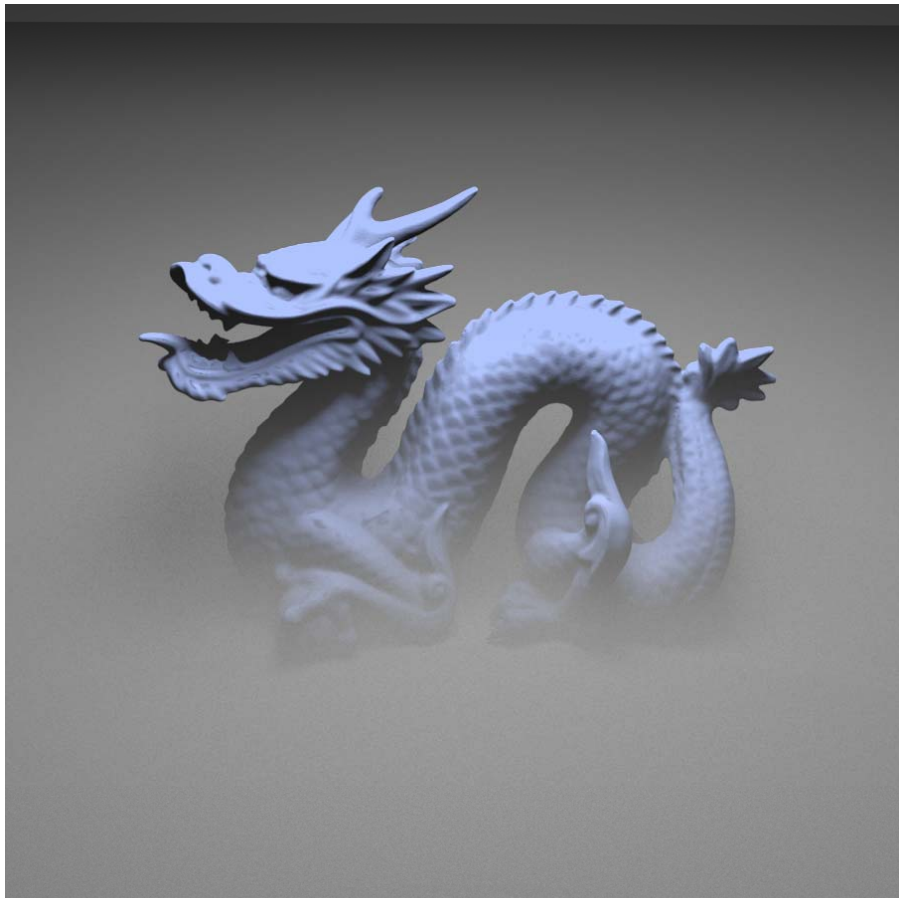


- Given by

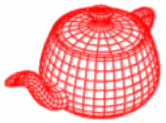
$$d(h) = ae^{-bh}$$

- Where h is the height in the direction of the up-vector

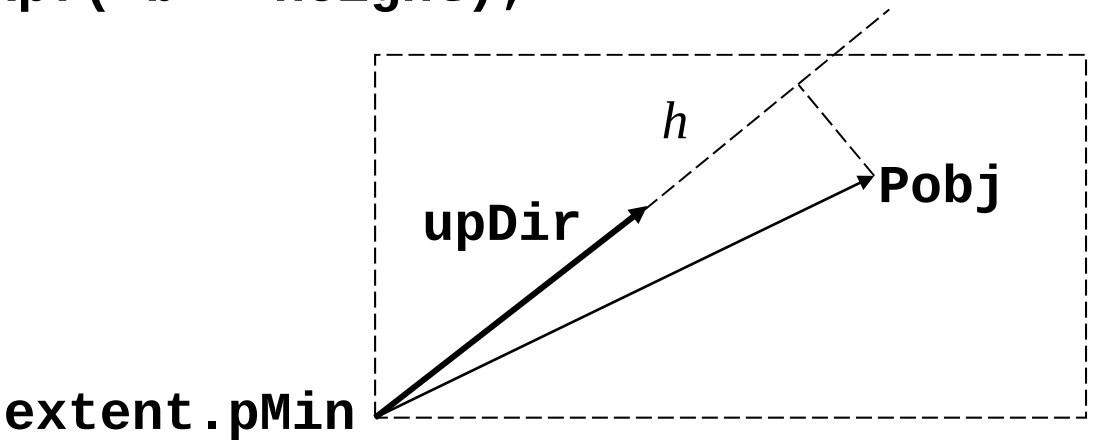
`ExponentialDensity`



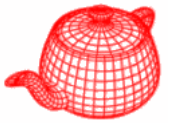
ExponentialDensity



```
class ExponentialDensity : public DensityRegion {
public:
   ExponentialDensity(Spectrum &sa, Spectrum &ss,
     float g, Spectrum &emit, BBox &e, Transform &v2w,
     float aa, float bb, Vector &up)
     ...

   float Density(const Point &Pobj) const {
       if (!extent.Inside(Pobj)) return 0;
       float height = Dot(Pobj - extent.pMin, upDir);
       return a * expf(-b * height);
   }
private:
   BBox extent;
   float a, b;
   Vector upDir;
};
```



Light transport



- Emission + in-scattering (source term)

$$S(x,\omega) = L_{ve}(x,\omega) + \sigma_s(x,\omega)\int_{\Omega} p(x,\omega' \rightarrow \omega)L(x,\omega')d\omega'$$

$$dL(x,\omega) = S(x,\omega)ds$$

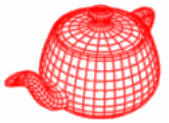
- Absorption + out-scattering (extinction)

$$dL(x,\omega) = -\sigma_t(x,\omega)L(x,\omega)ds$$

- Combined

$$\frac{dL(x,\omega)}{ds} = -\sigma_t(x,\omega)L(x,\omega) + S(x,\omega)$$

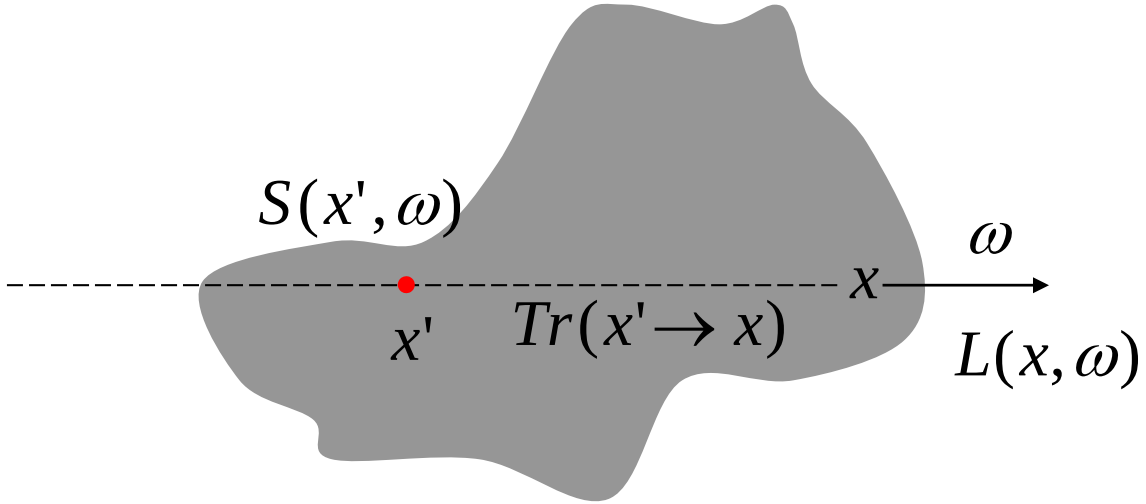
Infinite length, no surface



- Assume that there is no surface and we have an infinite length, we have the solution

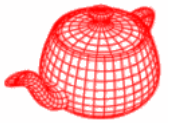
$$L(x,\omega) = \int_0^{\infty} Tr(x' \rightarrow x) S(x',\omega) ds$$

$$Tr(x' \rightarrow x) = e^{-\int_0^s \sigma_t(x+s'\omega,\omega) ds'}$$



$$x' = x - s\omega$$

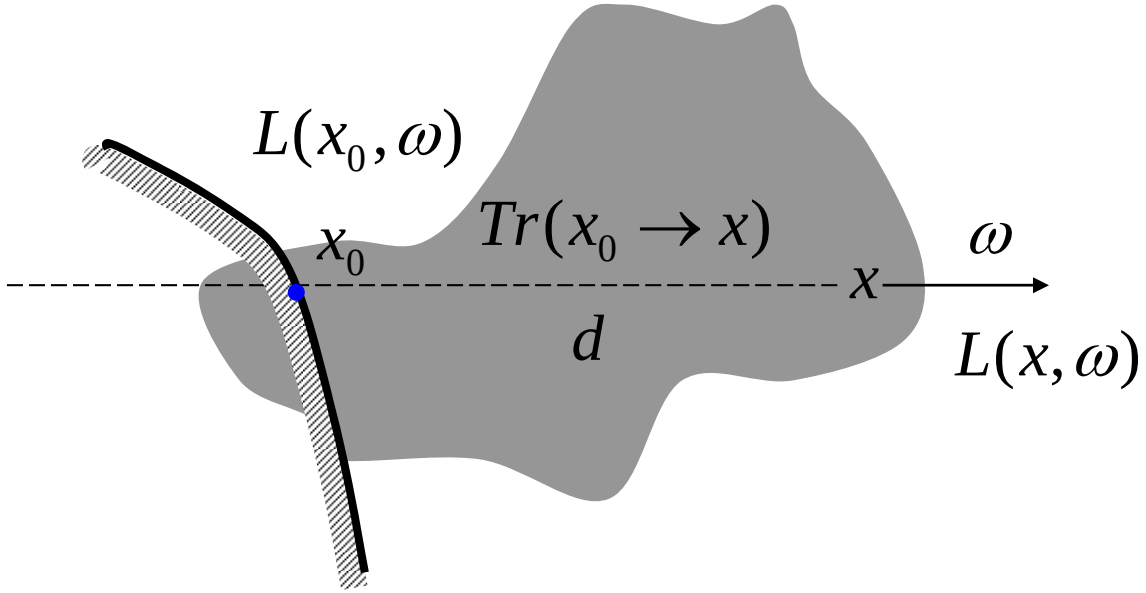
With surface



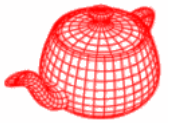
- The solution

$$L(x,\omega) = Tr(x_0 \rightarrow x)L(x_0,-\omega)$$

from the surface point x_0



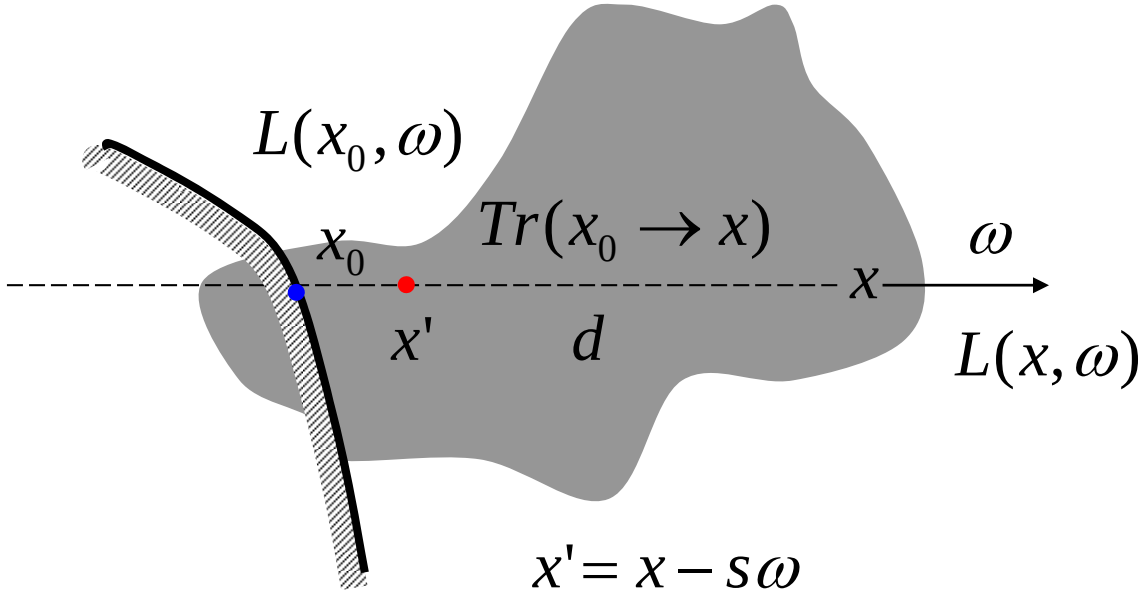
With surface



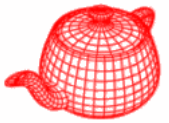
- The solution

$$L(x,\omega) = \boxed{Tr(x_0 \rightarrow x)L(x_0,-\omega)} + \boxed{\int_0^d Tr(x' \rightarrow x)S(x',-\omega)ds}$$

from the surface point x_0 from the participating media



Simple atmosphere model

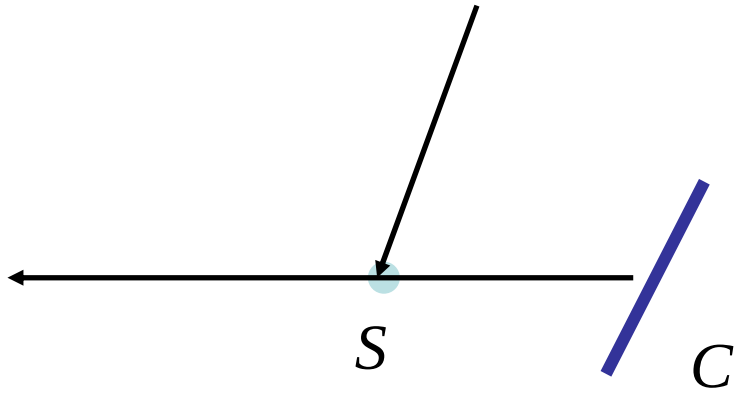


Assumptions

- Homogenous media
- Constant source term (airlight)

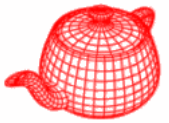
$$\frac{\partial L(s)}{\partial s} = -\sigma_t L(s) + S$$

$$L(s) = \left(1 - e^{-\sigma_t s}\right) S \quad + e^{-\sigma_t s} C$$

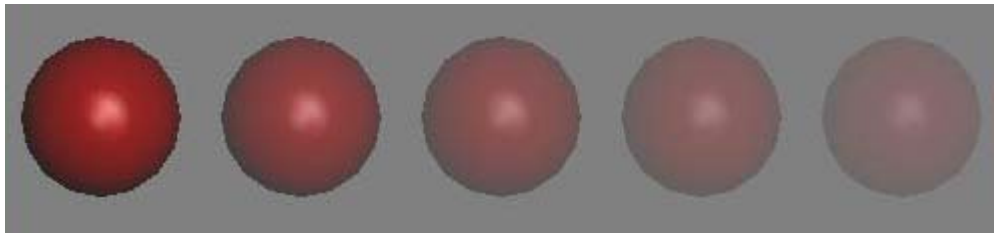


- Fog
- Haze

OpenGL fog model



$$C = fC_{in} + (1 - f)C_{fog}$$



GL_EXP

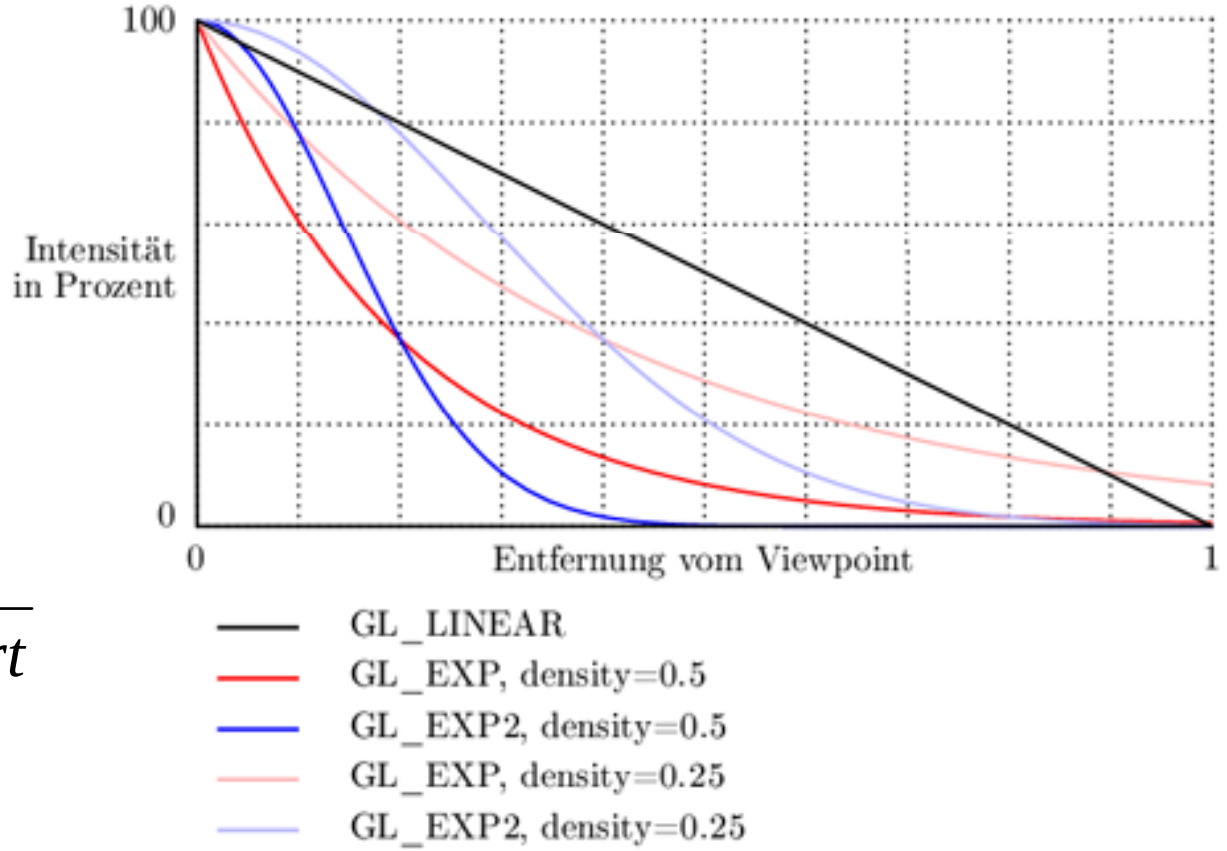
$$f(z) = e^{-(density \cdot z)}$$

GL_EXP2

$$f(z) = e^{-(density \cdot z)^2}$$

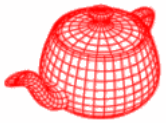
GL_LINEAR

$$f(z) = \frac{end - z}{end - start}$$



From http://wiki.delphigl.com/index.php/glFog

VolumeIntegrator

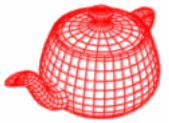


```
class VolumeIntegrator : public Integrator {
public:
  virtual Spectrum Transmittance(
    const Scene *scene,
    const Renderer *renderer,
    const RayDifferential &ray,
    const Sample *sample, …) const = 0;
};
```

Beam transmittance for a given ray from mint to maxt

Pick up functions **Preprocess()**, **RequestSamples()** and **Li()** from **Integrator**.

Emission only



- Solution for the emission-only simplification

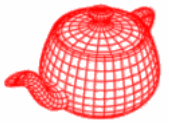
$$S(x',-\omega) = L_{ev}(x',-\omega)$$

$$L(x,\omega) = Tr(x_0 \rightarrow x)L(x_0,-\omega) + \int_0^d Tr(x' \rightarrow x)L_{ev}(x',-\omega)ds$$

- Monte Carlo estimator

$$\frac{1}{N}\sum_{i=1}^{N}\frac{Tr(x_i \rightarrow x)L_{ev}(x_i,\omega)}{p(x_i)} = \frac{t_1 - t_0}{N}\sum_{i=1}^{N} Tr(x_i \rightarrow x)L_{ev}(x_i,\omega)$$

Emission only

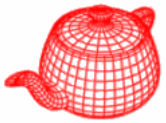


- Use multiplicativity of Tr

$$Tr(x_i \rightarrow x) = Tr(x_i \rightarrow x_{i-1}) \cdot Tr(x_{i-1} \rightarrow x)$$

- Break up integral and compute it incrementally by ray marching
- Tr can get small in a long ray
  - Early ray termination
  - Either use Russian Roulette or deterministically

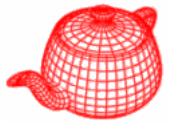
EmissionIntegrator



```
class EmissionIntegrator : public VolumeIntegrator {
public:
  EmissionIntegrator(float ss) { stepSize = ss; }
  void RequestSamples(Sampler *sampler, Sample
    *sample, Scene *scene);
  Spectrum Li(Scene *scene, Renderer *renderer,
    RayDifferential &ray, Sample *sample, RNG &rng,
    Spectrum *transmittance, MemoryArena &arena);
  Spectrum Transmittance(Scene *scene, Renderer *,
    RayDifferential &ray, Sample *sample, RNG &rng,
    MemoryArena &arena);
private:
  float stepSize;
  int tauSampleOffset, scatterSampleOffset;
};
```

single 1D sample for each

EmissionIntegrator::Transmittance



```
if (!scene->volumeRegion) return Spectrum(1);
float step, offset;
if (sample) {
  step = stepSize;
  offset = sample->oneD[tauSampleOffset][0];
} else {
  step = 4.f * stepSize;
  offset = rng.RandomFloat();
}
Spectrum tau = scene->volumeRegion->tau(ray,
                  step, offset);
return Exp(-tau);
```

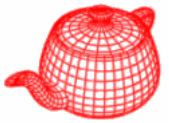
smaple is non-NULL only for camera rays

use larger steps for shadow and indirect rays for efficiency

$$T_\omega(s) = e^{-\tau_\omega(s)}$$

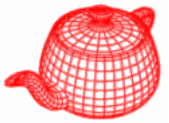
$$\tau_\omega(s) = \int_0^s \sigma_a(x + s'\omega, \omega) ds'$$

EmissionIntegrator::Li



```
VolumeRegion *vr = scene->volumeRegion;
float t0, t1;
if (!vr || !vr->IntersectP(ray, &t0, &t1)
    || (t1-t0) == 0.f) {
  *T = Spectrum(1.f);
  return 0.f;
}
// Do emission-only volume integration in vr
Spectrum Lv(0.);
// Prepare for volume integration stepping
int nSamples = Ceil2Int((t1-t0) / stepSize);
float step = (t1 - t0) / nSamples;
Spectrum Tr(1.f);
Point p = ray(t0), pPrev;
Vector w = -ray.d;
t0 += sample->oneD[scatterSampleOffset][0] * step;
```

EmissionIntegrator::Li

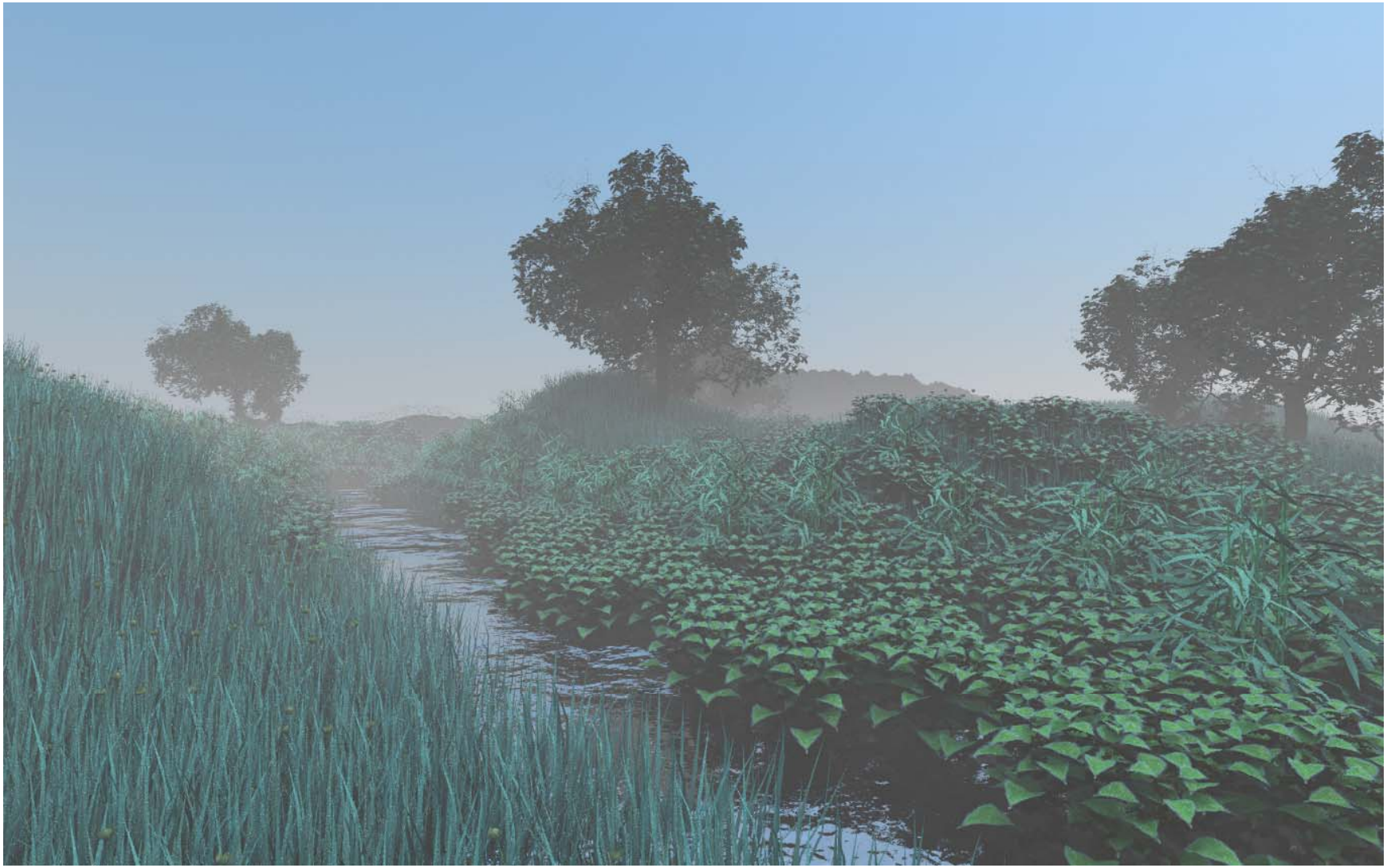


```
for (int i = 0; i < nSamples; ++i, t0 += step) {
  pPrev = p;  p = ray(t0);
  Ray tauRay(pPrev, p - pPrev, 0.f, 1.f, ray.time,
      ray.depth);
  Spectrum stepTau = vr->tau(tauRay,.5f * stepSize,
      rng.RandomFloat());
  Tr *= Exp(-stepTau);
  // Possibly terminate if transmittance is small
  if (Tr.y() < 1e-3) {
    const float continueProb = .5f;
    if (rng.RandomFloat() > continueProb) break;
    Tr /= continueProb;
  }
  // Compute emission-only source term at _p_
  Lv += Tr * vr->Lve(p, w, ray.time);
}
*T = Tr;
return Lv * step;
```

$$Tr(x_i \rightarrow x) = Tr(x_i \rightarrow x_{i-1}) \cdot Tr(x_{i-1} \rightarrow x)$$

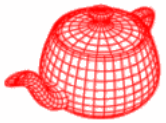
$$\frac{t_1 - t_0}{N} \sum_{i=1}^{N} Tr(x_i \rightarrow x) L_{ev}(x_i, \omega)$$

Emission only



exponential density

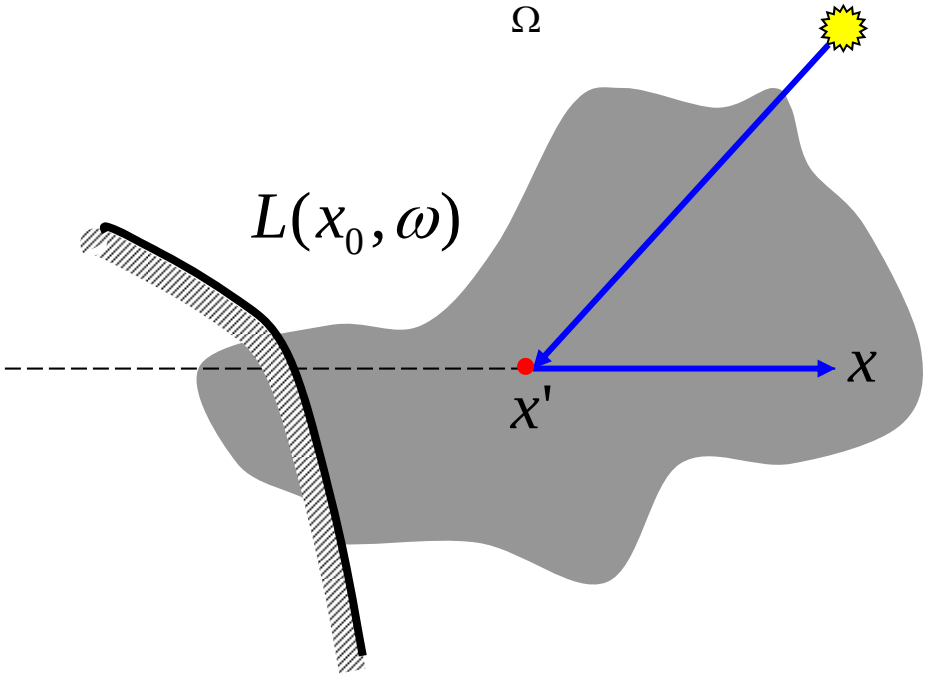
Single scattering



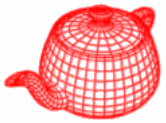
- Consider incidence radiance due to direct illumination

$$L(x,\omega) = Tr(x_0 \rightarrow x)L(x_0,\omega) + \int_0^d Tr(x' \rightarrow x)S(x',\omega)ds$$

$$S(x,\omega) = L_{ve}(x,\omega) + \sigma_s(x,\omega)\int_\Omega p(x,\omega' \rightarrow \omega)L_d(x,\omega')d\omega'$$



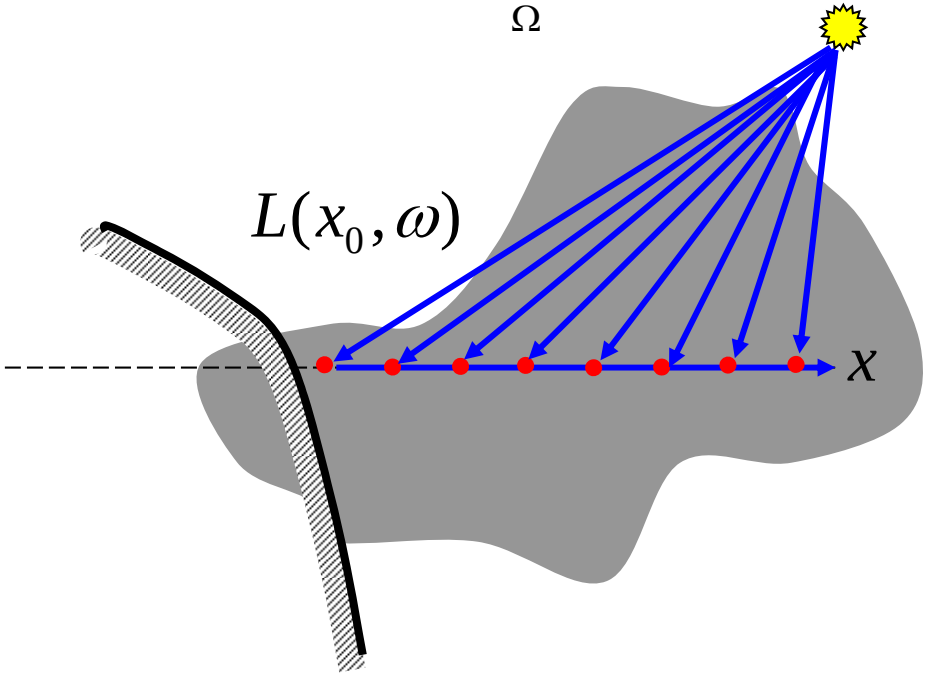
Single scattering



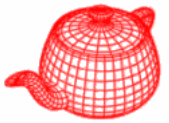
- Consider incidence radiance due to direct illumination

$$L(x,\omega) = Tr(x_0 \rightarrow x)L(x_0,\omega) + \int_0^d Tr(x' \rightarrow x)S(x',\omega)ds$$

$$S(x,\omega) = L_{ve}(x,\omega) + \sigma_s(x,\omega)\int_\Omega p(x,\omega' \rightarrow \omega)L_d(x,\omega')d\omega'$$



Single scattering



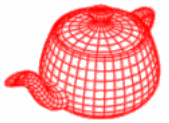
- L_d may be attenuated by participating media
- At each point of the integral, we could use multiple importance sampling to get

$$\sigma_s(x,\omega)\int_{\Omega} p(x,\omega' \rightarrow \omega)L_d(x,\omega')d\omega'$$

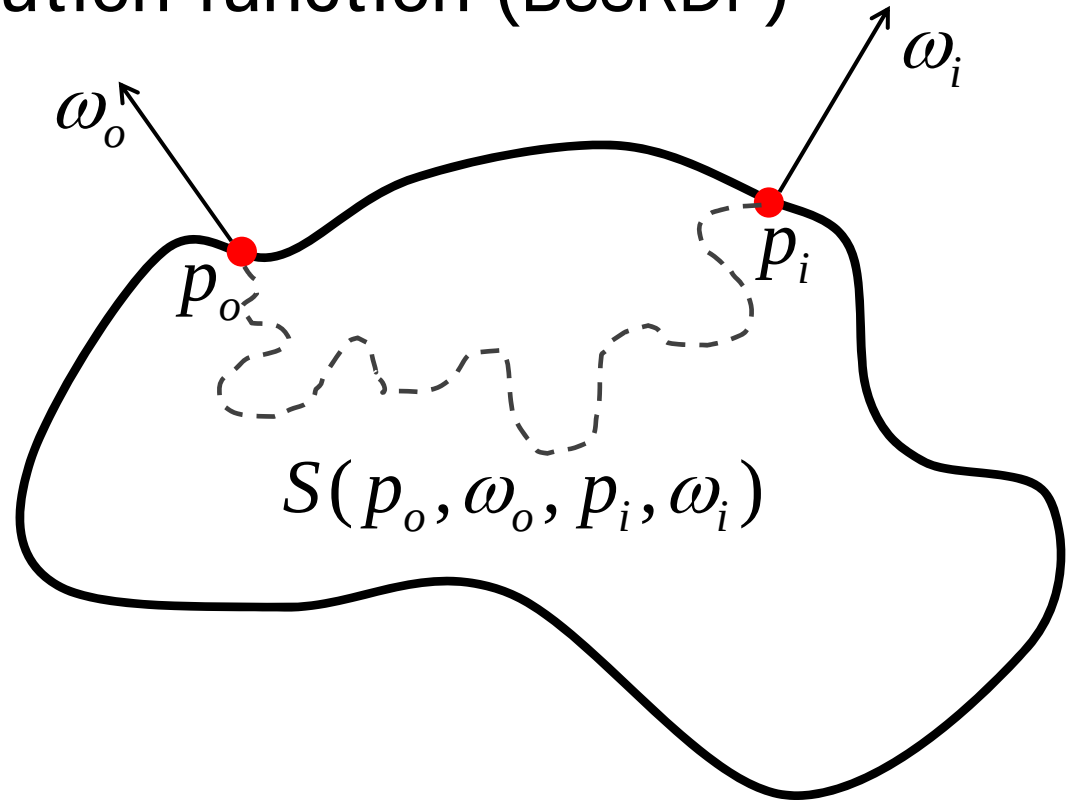
But, in practice, we can just pick up light source randomly.

Single scattering

Subsurface scattering

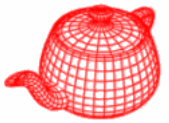


- The bidirectional scattering-surface reflectance distribution function (BSSRDF)



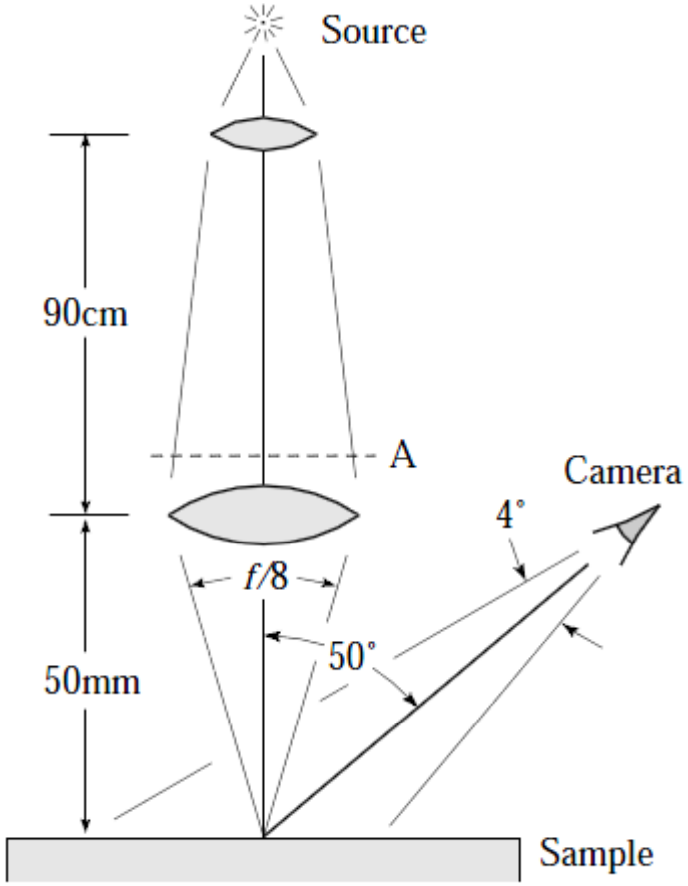
$$L_o(p_o, \omega_o) = \int_A \int_{H^2} S(p_o, \omega_o, p_i, \omega_i) L_i(p_i, \omega_i) |\cos\theta_i| d\omega_i dA$$

Subsurface scattering

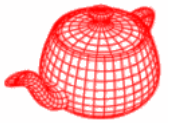


- Translucent materials have similar mechanism for light scattering as participating media. Thus, path tracing could be used.
- However, many translucent objects have very high albedo. Taken milk as an example, after 100 scattering events, 87.5% of the incident light is still carried by a path, 51% after 500 and 26% after 1,000.
- Efficiently rendering these kinds of translucent scattering media requires a different approach.

Highly translucent materials

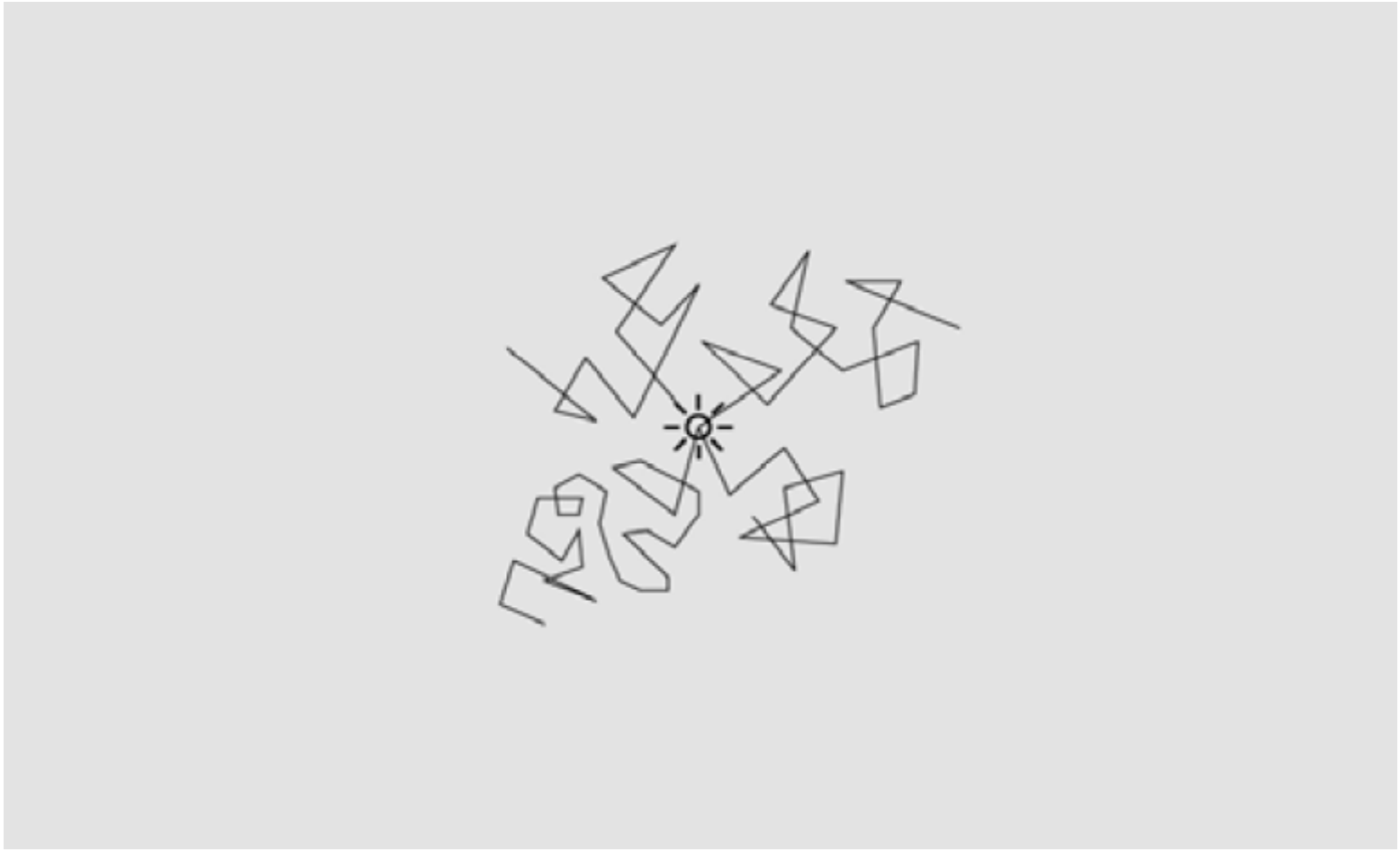


Main idea

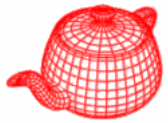


- Assume that the material is homogeneous and the medium is semi-infinite, we can use diffusion approximation to describe the equlibrium distribution of illumination.
- There is a solution to the diffusion equation by using a dipole of two light sources to approximate the overall scattering.

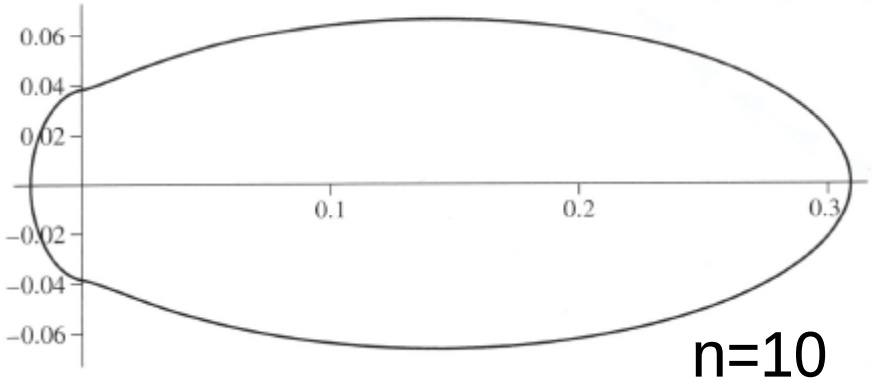
Highly scattering media



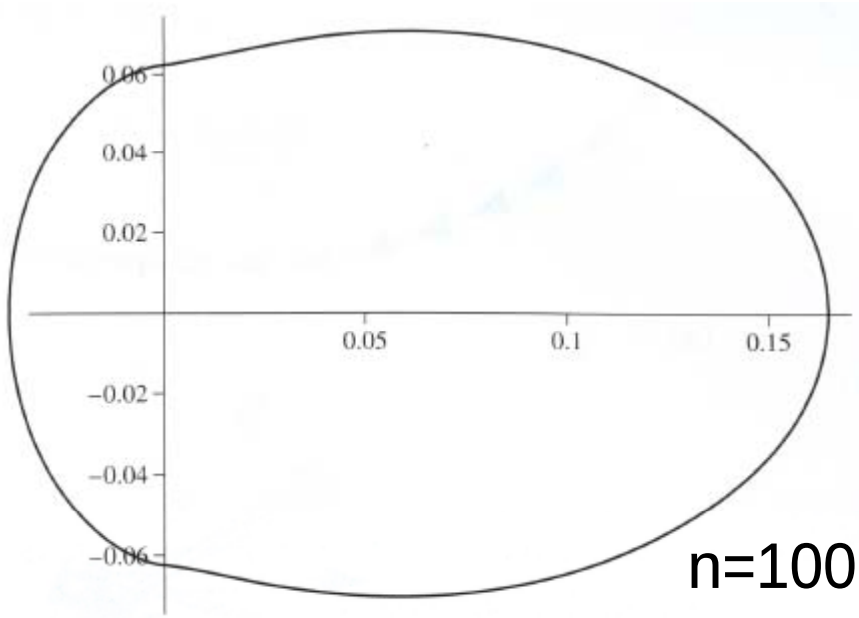
Principle of similarity



g=0.9

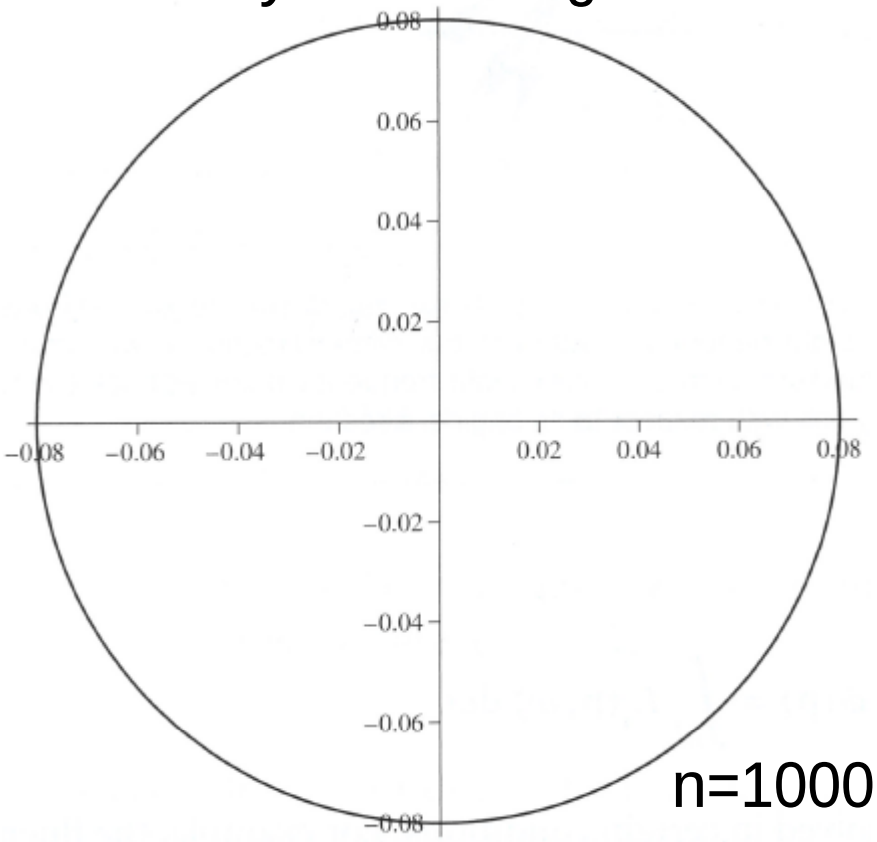


n=10



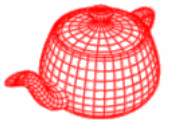
n=100

For high albedo objects, an anisotropically scattering phase function becomes isotropic after many scattering events.



n=1000

Diffusion approximation



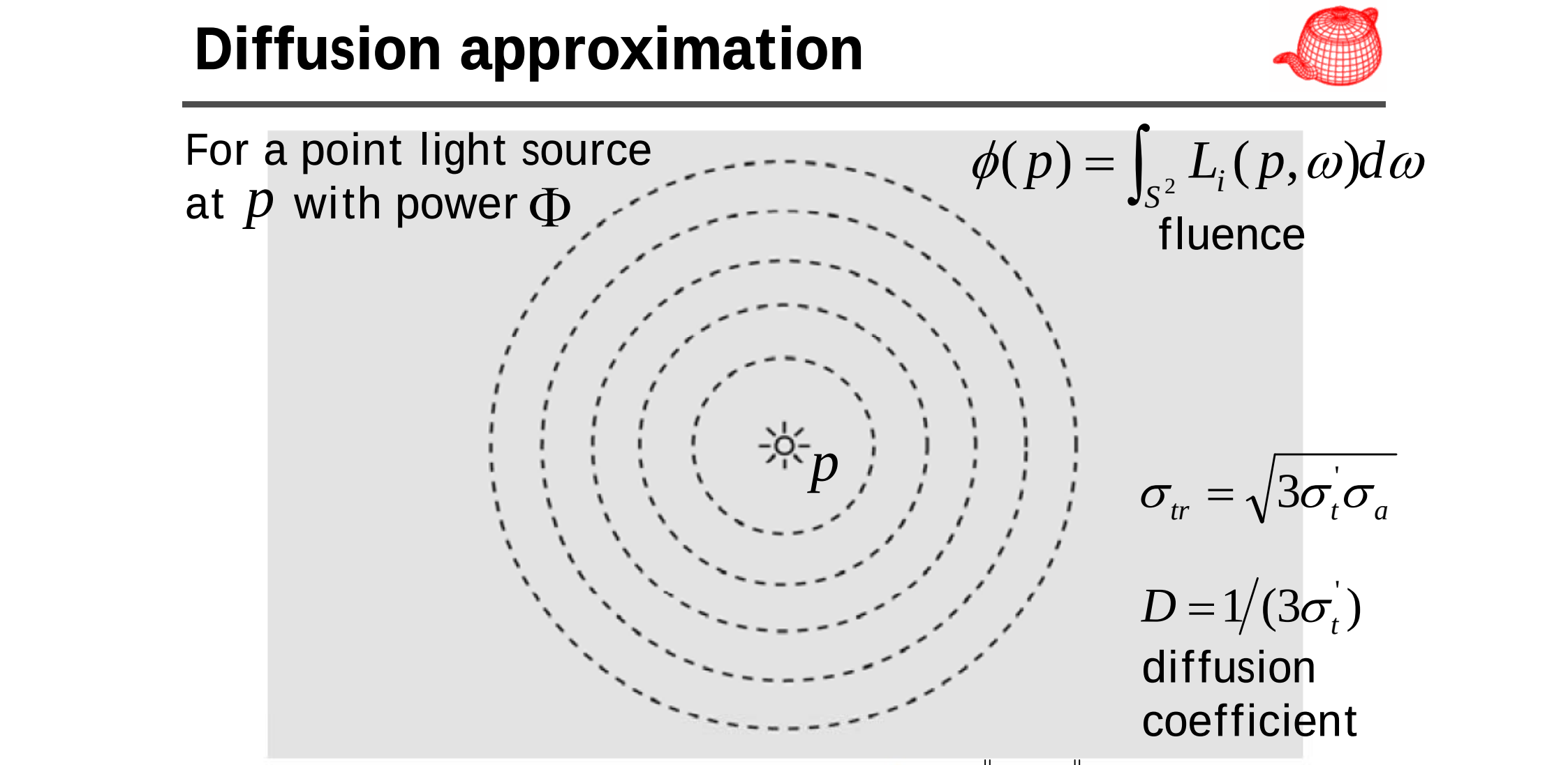
- The reduced scattering coefficient

$$\sigma_s^{'} = (1-g)\sigma_s$$

- The reduced extinction coefficient

$$\sigma_t^{'} = \sigma_a + \sigma_s^{'}$$

Diffusion approximation



For a point light source at p with power Φ

$$\phi(p) = \int_{S^2} L_i(p,\omega) d\omega$$

fluence

$$\sigma_{tr} = \sqrt{3\sigma_t' \sigma_a}$$

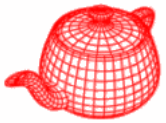
$$D = 1/(3\sigma_t')$$

diffusion coefficient

The fluence at p_i from p is

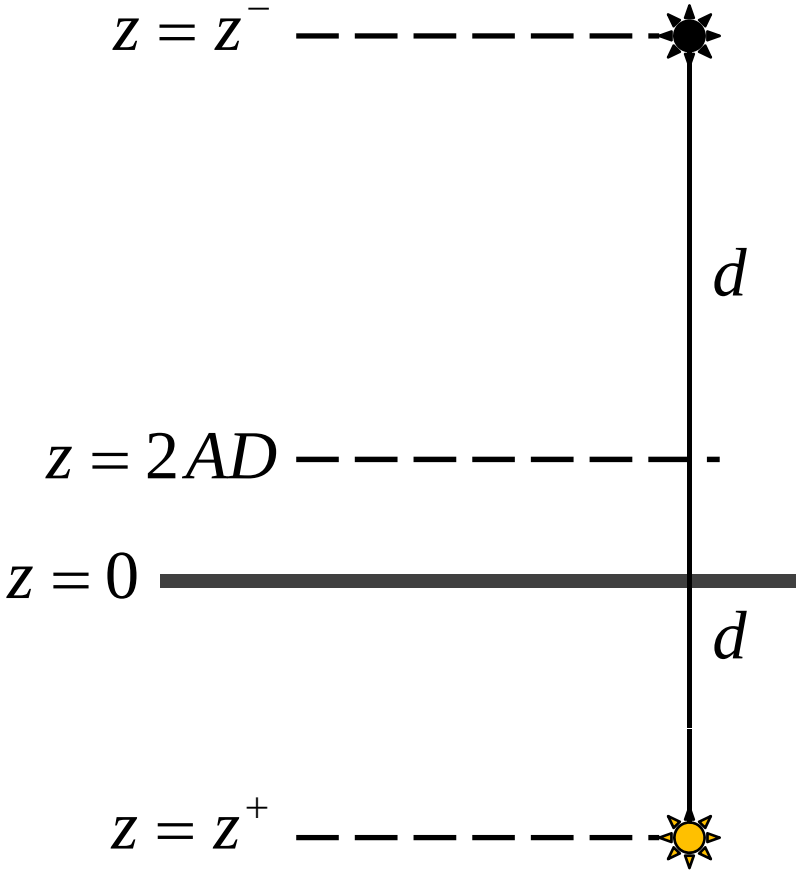
$$\phi(p, p_i) = \frac{\Phi}{4\pi D} \frac{e^{-\sigma_{tr}\|p - p_i\|}}{\|p - p_i\|}$$

Dipole diffusion approximation



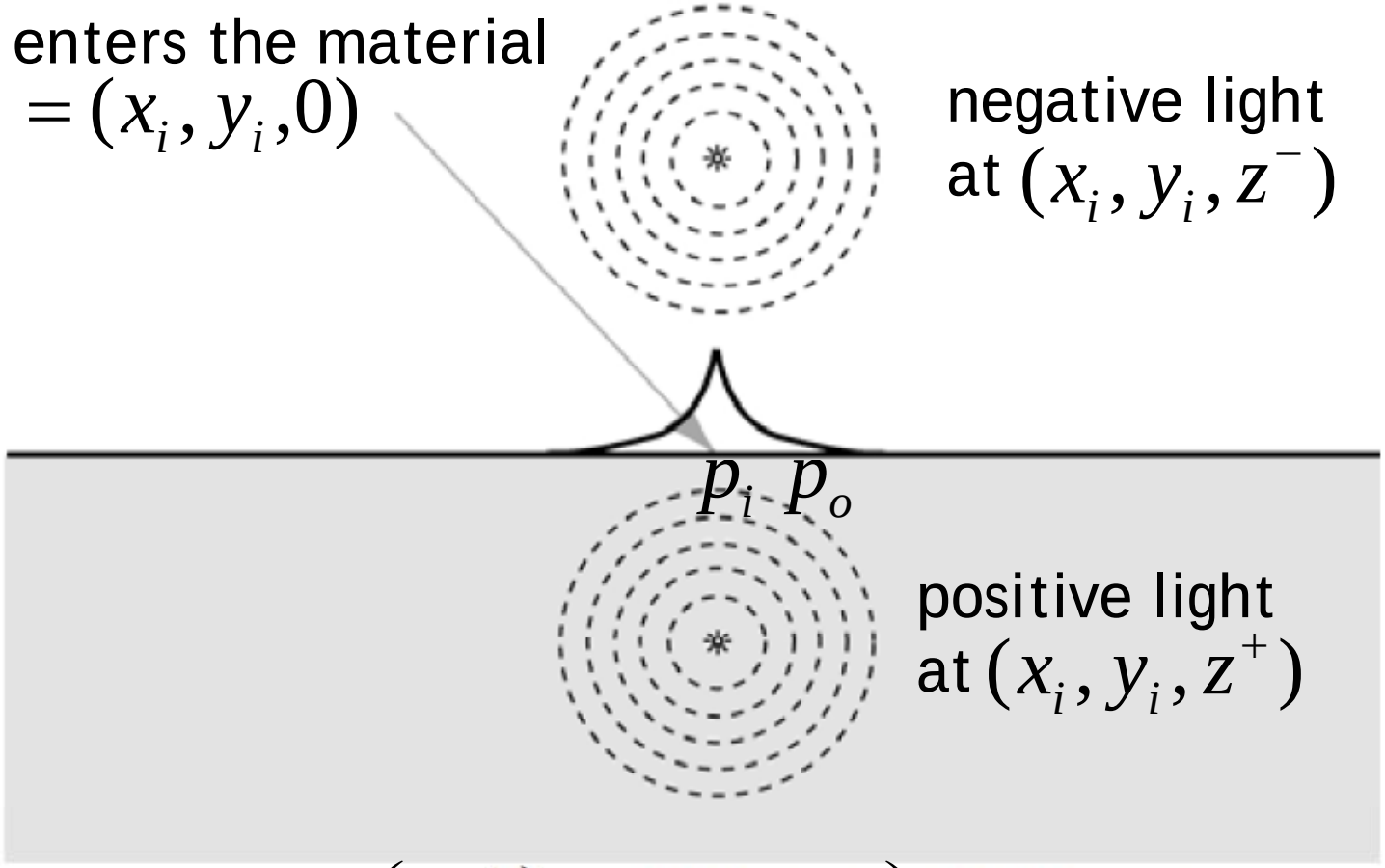
$$A = \frac{1 + F_{dr}(\eta)}{1 - F_{dr}(\eta)}$$

$$F_{dr}(\eta) = \int_{H^2} F_r(\eta, \omega_i) \left| \omega_i \cdot n \right| d\omega_i$$



Dipole diffusion approximation

Light enters the material at $p_i = (x_i, y_i, 0)$

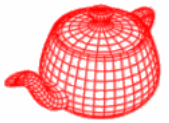


$$\phi(p_i, p_o) = \frac{\Phi}{4\pi D}\left(\frac{e^{-\sigma_{tr} d^+}}{d^+} - \frac{e^{-\sigma_{tr} d^-}}{d^-}\right)$$

$$d^+ = \left\|(x_i, y_i, z^+) - p_o\right\|$$

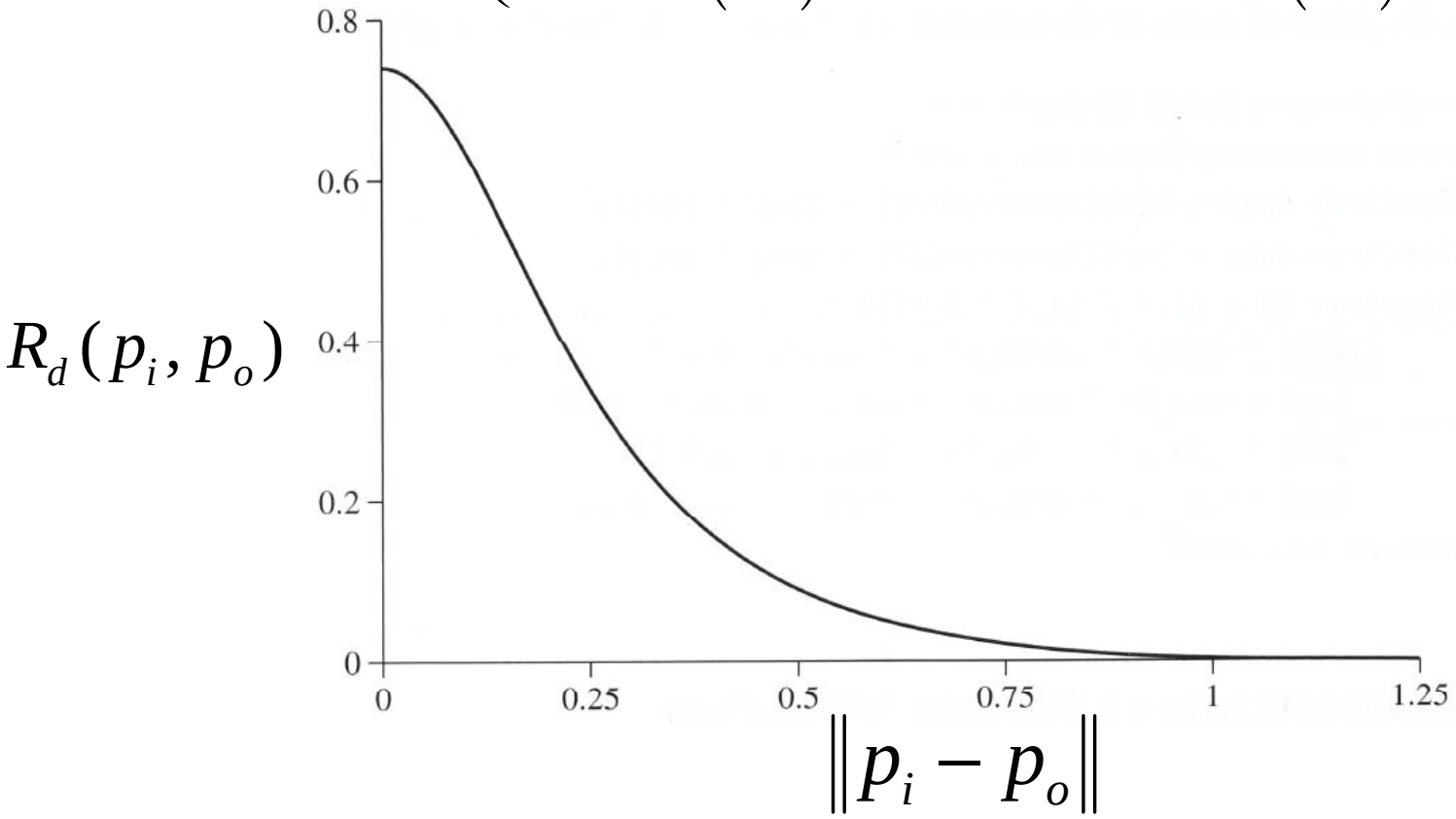
$$d^- = \left\|(x_i, y_i, z^-) - p_o\right\|$$

Dipole diffusion approximation

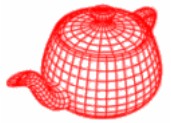


- Put all together

$$R_d(p_i, p_o) = \frac{1}{4\pi}\left(\frac{z^+(d^+\sigma_{tr}+1)e^{-\sigma_{tr}d^+}}{(d^+)^3} - \frac{z^-(d^-\sigma_{tr}+1)e^{-\sigma_{tr}d^-}}{(d^-)^3}\right)$$

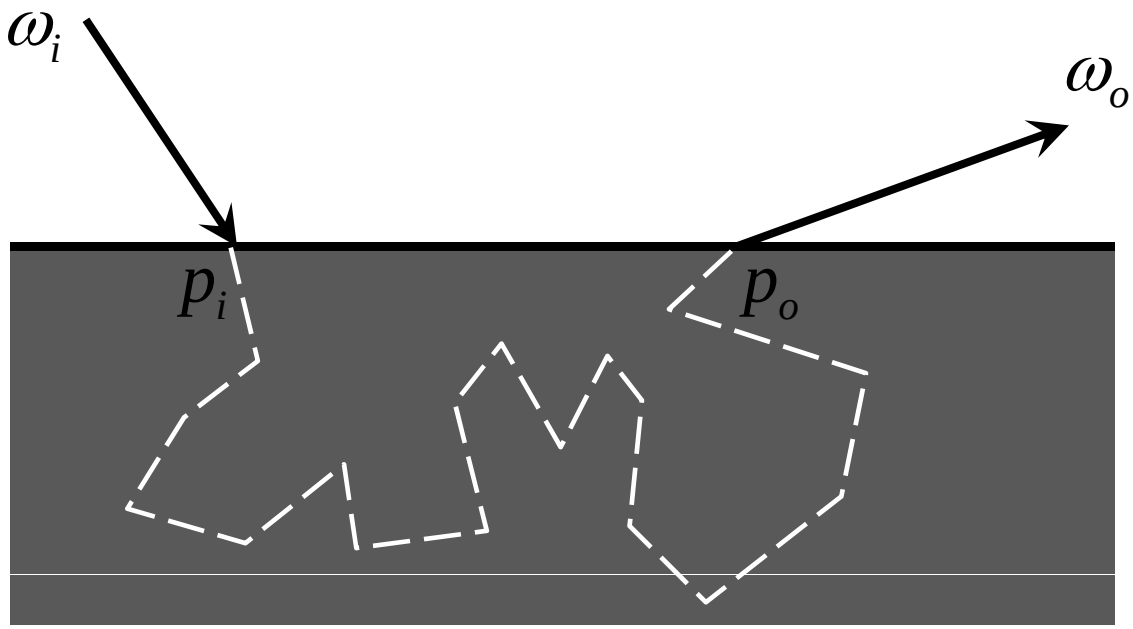


BSSRDF

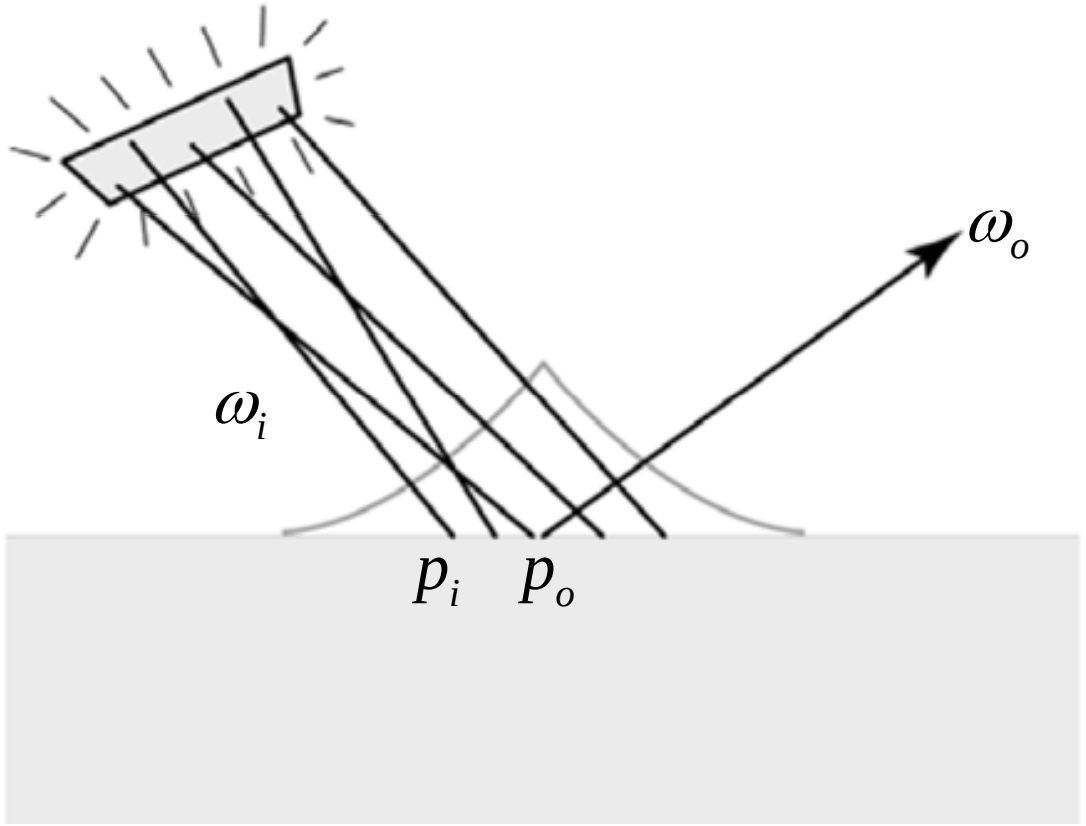


- BSSRDF based on the diffusion subsurface reflectance approximation

$$S(p_o, \omega_o, p_i, \omega_i) = \frac{1}{\pi} F_t(\eta_o, \omega_o) R_d(\|p_i - p_o\|) F_t(\eta_i, \omega_i)$$



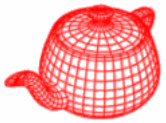
Evaluating BSSRDF



$$L_o(p_o, \omega_o)$$

$$= \int_A \int_{H^2} \left(\frac{1}{\pi} F_t(\eta_o, \omega_o) R_d(\|p_i - p_o\|) F_t(\eta_i, \omega_i) \right) L_i(p_i, \omega_i) |\cos\theta_i| d\omega_i dA$$

Evaluating BSSRDF



$$L_o(p_o,\omega_o)$$

$$= \frac{1}{\pi} F_t(\eta_o,\omega_o) \int_A R_d(\|p_i - p_o\|) \left[\int_{H^2} \left(F_t(\eta_i,\omega_i)\right) L_i(p_i,\omega_i) |\cos\theta_i| \, d\omega_i \right] dA$$

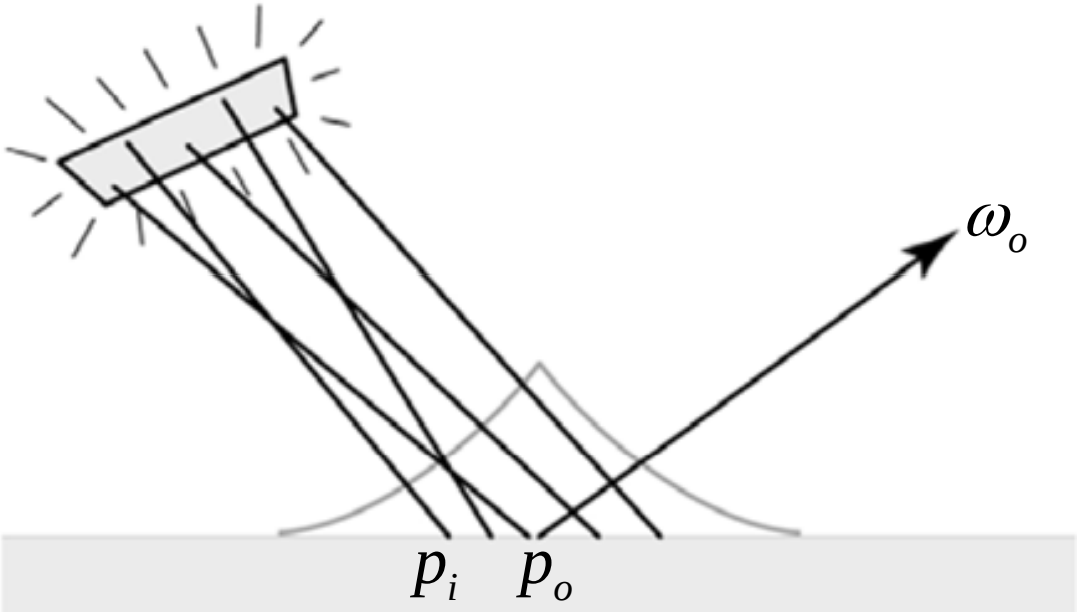
$$\approx F_{dt}(\eta_i) E(p_i)$$

For homogeneous materials, $\eta = \eta_o = \eta_i$

$$L_o(p_o,\omega_o)$$

$$\approx \frac{1}{\pi} F_t(\eta,\omega_o) F_{dt}(\eta) \int_A R_d(\|p_i - p_o\|) E(p_i) dA$$

Evaluating BSSRDF

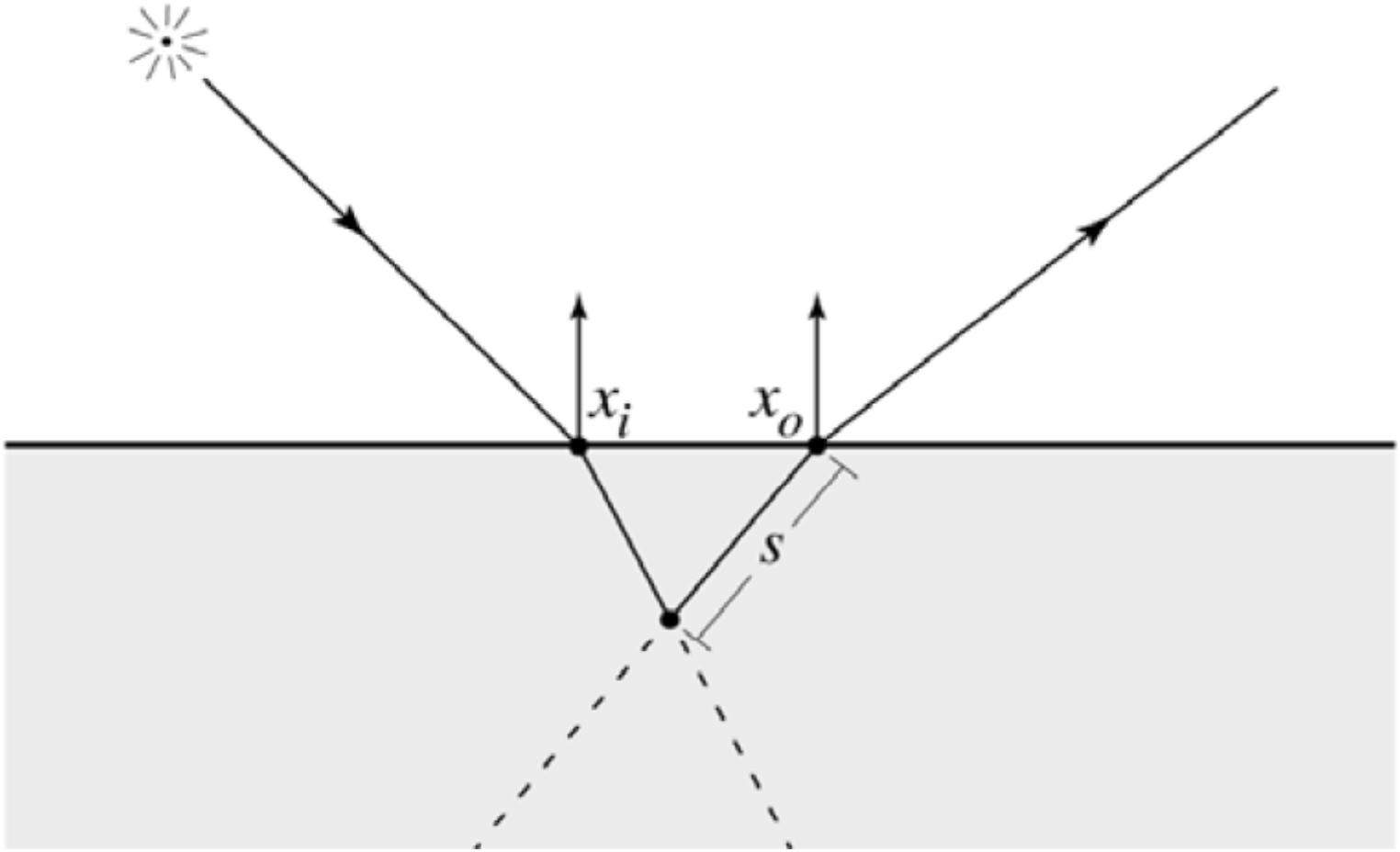


$$L_o(p_o, \omega_o)$$

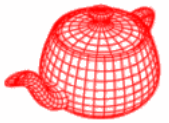
$$\approx \frac{1}{\pi} F_t(\eta, \omega_o) F_{dt}(\eta) \int_A R_d(\|p_i - p_o\|) E(p_i) dA$$

$$M_o(p_o) = \int_A R_d(\|p_i - p_o\|) E(p_i) dA \approx \sum_j R_d(\|p_j - p_o\|) E_j A_j$$

Single scattering

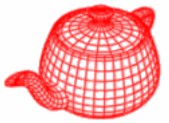


Solutions



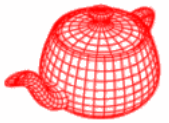
- Path tracing
- Photon mapping
- Hierarchical approach (Jensen 2002)

Three main components



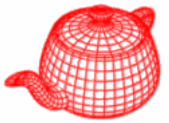
- Sample a large number of random points on the surface and their incident irradiance is computed.
- Create a hierarchy of these points, progressively clustering nearby points and summing their irradiance values.
- At rendering, use a hierarchical integration algorithm to evaluate the subsurface scattering equation.

DipoleSubsurfaceIntegrator

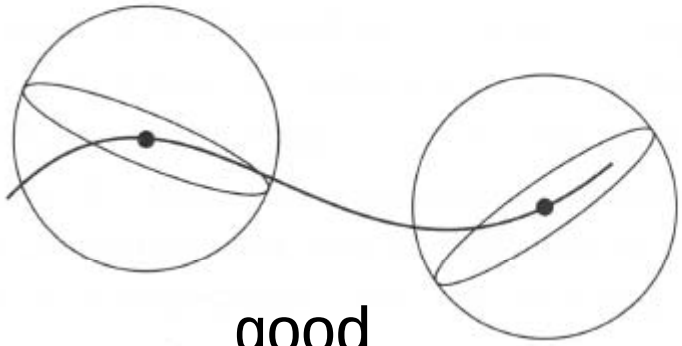


```
class DipoleSubsurfaceIntegrator : public
      SurfaceIntegrator {
  …
  int maxSpecularDepth;
  float maxError, minSampleDist;
  string filename;
  vector<IrradiancePoint> irradiancePoints;
  BBox octreeBounds;
  SubsurfaceOctreeNode *octree;
}
```

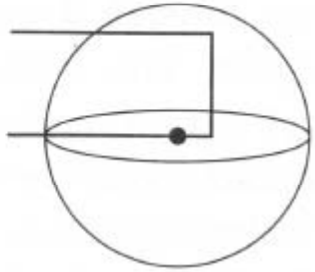
Sample points



- Samples should be reasonably uniformly distributed. A Poisson disk distribution of points is a good choice.
- There are some algorithms for generating such distributions. Pbrt uses a 3D version of dart throwing by performing Poisson sphere tests. The algorithm terminates when a few thousand tests have been rejected in a row.

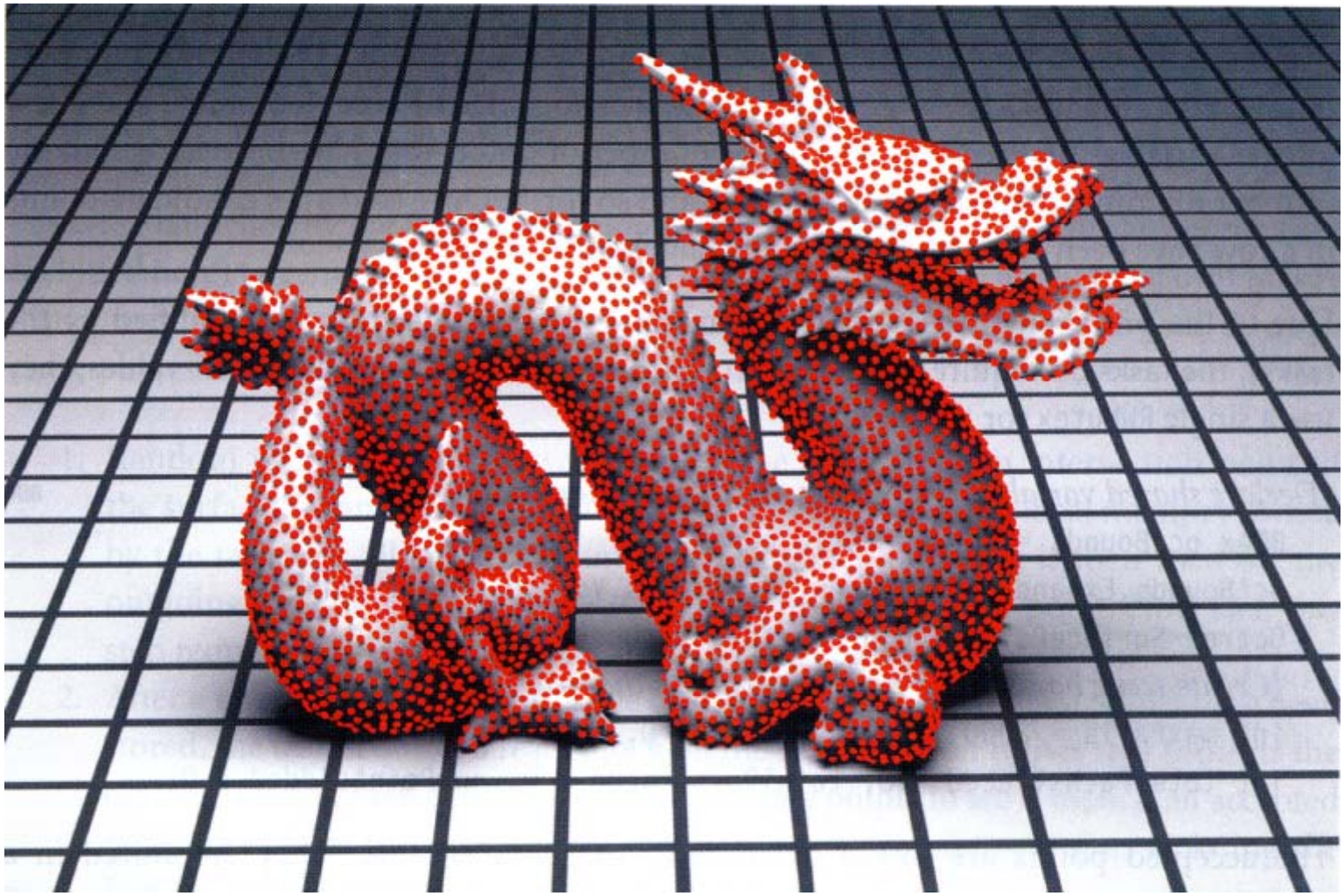


good

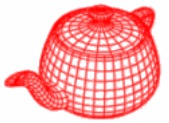


Bad, but dipole is dad for this anyway

Sample points

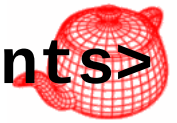


Preprocess



```
if (scene->lights.size() == 0) return;
<Get SurfacePoints for translucent objects>
<Compute irradiance values at sample points>
<Create octree of clustered irradiance samples>
```

<Compute irradiance values at sample points>

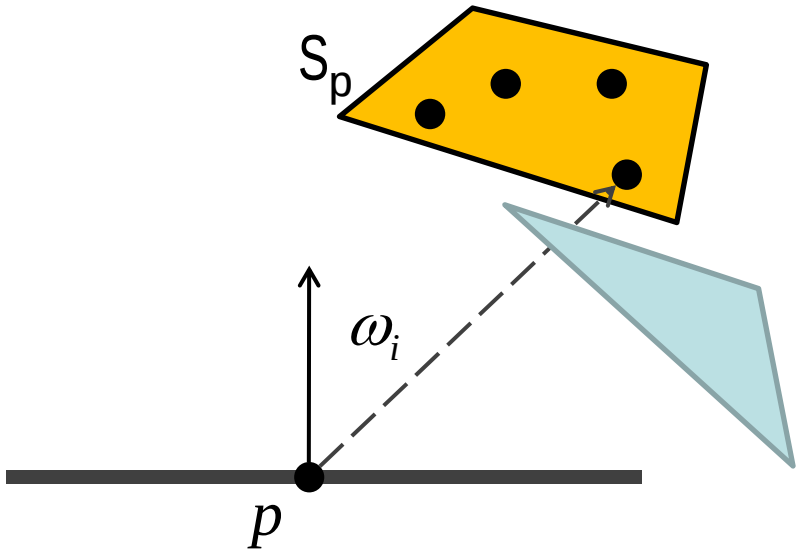


```
for (uint32_t i = 0; i < pts.size(); ++i) {
  SurfacePoint &sp = pts[i];
  Spectrum E(0.f);
  for (int j = 0; j < scene->lights.size(); ++j) {
    <Add irradiance from light at point>
  }
  irradiancePoints.push_back(IrradiancePoint(sp, E));
}
```

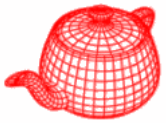
Calculate direct lighting only; it is possible to include indirect lighting but more expensive

$$E(p) = \int_{H^2} L_i(p, \omega_i) |\cos\theta_i| \, d\omega_i$$

$$E(p) = \frac{1}{N} \sum_j \frac{L_i(p, \omega_j) |\cos\theta_j|}{p(\omega_j)}$$

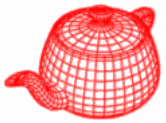


IrradiancePoint



```
struct SurfacePoint {
  …
  Point p;
  Normal n;
  float area, rayEpsilon;
};

struct IrradiancePoint {
  …
  Point p;
  Normal n;
  Spectrum E;
  float area, rayEpsilon;
};
```

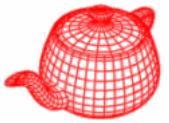
<Create octree of clustered irradiance samples>



```
octree = octreeArena.Alloc<SubsurfaceOctreeNode>();
for (int i = 0; i < irradiancePoints.size(); ++i)
  octreeBounds = Union(octreeBounds,
                       irradiancePoints[i].p);
for (int i = 0; i < irradiancePoints.size(); ++i)
  octree->Insert(octreeBounds, &irradiancePoints[i],
                 octreeArena);
octree->InitHierarchy();
```

Computes representative position, irradiance and area for each node. Positions are weighted by irradiance values to emphasize the points with higher irradiance values.

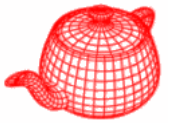
Li



$$L_o(p_o,\omega_o) \approx \frac{1}{\pi} F_t(\eta,\omega_o) F_{dt}(\eta) M_o(p_o)$$

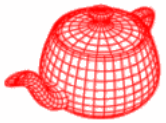
```
if (bssrdf && octree) {
  Spectrum sigma_a  = bssrdf->sigma_a();
  Spectrum sigmap_s = bssrdf->sigma_prime_s();
  Spectrum sigmap_t = sigmap_s + sigma_a;
  if (!sigmap_t.IsBlack()) {
    DiffusionReflectance Rd(sigma_a, sigmap_s,
                            bssrdf->eta());
    Mo = octree->Mo(octreeBounds, p, Rd, …);
    FresnelDielectric fresnel(1.f, bssrdf->eta());
    Ft = Spectrum(1)- fresnel.Evaluate(AbsDot(wo,n));
    float Fdt = 1.f - Fdr(bssrdf->eta());
    L += (INV_PI * Ft) * (Fdt * Mo);
  }
}
```

Li

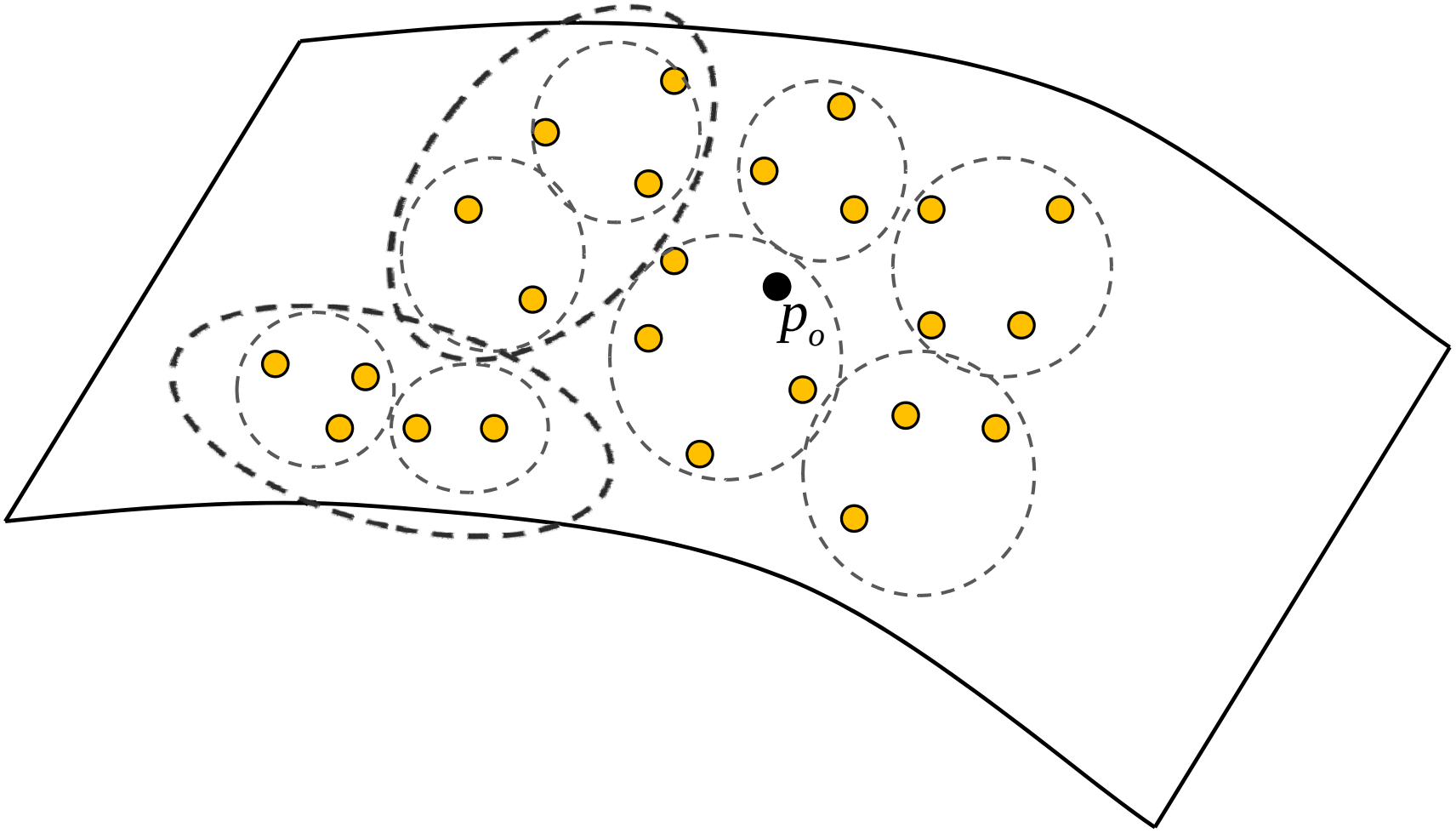


```
L += UniformSampleAllLights(…);
if (ray.depth < maxSpecularDepth) {
  L += SpecularReflect(…);
  L += SpecularTransmit(…);
}
return L;
```

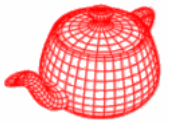
Mo



$$M_o(p_o) = \sum_j R_d(\|p_j - p_o\|) E_j A_j$$



SubsurfaceOctreeNode::Mo

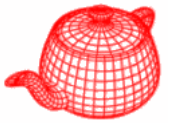


```
Spectrum SubsurfaceOctreeNode::Mo(BBox &nodeBound,
 Point &pt, DiffusionReflectance &Rd, float maxError
{
```

If extended solid angle of the node is small enough and the point is not inside the node, use the representative values of the node to estimate.

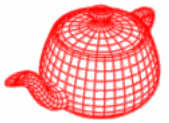
```
  float dw = sumArea / DistanceSquared(pt, p);
  if (dw < maxError && !nodeBound.Inside(pt))
    return Rd(DistanceSquared(pt, p)) * E * sumArea;

  Spectrum Mo = 0.f;
```

SubsurfaceOctreeNode::Mo



```
if (isLeaf)
  for (int i = 0; i < 8; ++i) {
    if (!ips[i]) break;
    Mo += Rd(DistanceSquared(pt, ips[i]->p))
              * ips[i]->E * ips[i]->area;
  }
else {
  Point pMid=.5f*nodeBound.pMin+.5f*nodeBound.pMax;
  for (int child = 0; child < 8; ++child) {
    if (!children[child]) continue;
    Bbox cBound=octreeChildBound(child,nodeBound,pMid);
    Mo+=children[child]->Mo(cBound, pt, Rd, maxError);
  }
}
return Mo;
}
```

Setting parameters



- It is remarkably unintuitive to set values of the absorption coefficient and the modified scattering coefficient.

Material	σ_s' [mm^{-1}]			σ_a [mm^{-1}]			Diffuse Reflectance			η
	R	G	B	R	G	B	R	G	B	
Apple	2.29	2.39	1.97	0.0030	0.0034	0.046	0.85	0.84	0.53	1.3
Chicken1	0.15	0.21	0.38	0.015	0.077	0.19	0.31	0.15	0.10	1.3
Chicken2	0.19	0.25	0.32	0.018	0.088	0.20	0.32	0.16	0.10	1.3
Cream	7.38	5.47	3.15	0.0002	0.0028	0.0163	0.98	0.90	0.73	1.3
Ketchup	0.18	0.07	0.03	0.061	0.97	1.45	0.16	0.01	0.00	1.3
Marble	2.19	2.62	3.00	0.0021	0.0041	0.0071	0.83	0.79	0.75	1.5
Potato	0.68	0.70	0.55	0.0024	0.0090	0.12	0.77	0.62	0.21	1.3
Skimmilk	0.70	1.22	1.90	0.0014	0.0025	0.0142	0.81	0.81	0.69	1.3
Skin1	0.74	0.88	1.01	0.032	0.17	0.48	0.44	0.22	0.13	1.3
Skin2	1.09	1.59	1.79	0.013	0.070	0.145	0.63	0.44	0.34	1.3
Spectralon	11.6	20.4	14.9	0.00	0.00	0.00	1.00	1.00	1.00	1.3
Wholemilk	2.55	3.21	3.77	0.0011	0.0024	0.014	0.91	0.88	0.76	1.3

Marble: BRDF versus BSSRDF

BRDF

BSSRDF

Marble: MCRT versus BSSRDF

MCRT

BSSRDF

Milk

surface reflection

translucency

Skin

surface reflection

translucency