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)
Single scattering and multiple scattering

- Single scattering
- Multiple scattering

Attenuation
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

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

\[ \frac{dL(x, \omega)}{L(x, \omega)} = -\sigma_a(x, \omega)ds \rightarrow \int_x^{x+s_\omega} \frac{dL(x', \omega)}{L(x', \omega)} = -\int_0^s \sigma_a(x + s' \omega, \omega)ds' \]

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

Optical distance or depth

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

Homogeneous media: constant \( \sigma_a \)

\[ \sigma_a \rightarrow \tau(s) = \sigma_a s \]
Transmittance and opacity

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

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

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

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

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 $\omega$

\[
dL(x, \omega) = L_{ve}(x, \omega) ds
\]
Emission
Out-scattering

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

\[ L(x, \omega) \rightarrow \sigma_s(x, \omega) \rightarrow L + dL \]

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

**Scattering cross-section:** \( \sigma_s \)

**Probability of being scattered per unit length**
Extinction

\[ L(x, \omega) \xrightarrow{\sigma_t(x, \omega)} L + dL \]

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

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

\[ Tr(x \rightarrow x') = e^{-\int_0^s \sigma, (x+s'\omega, \omega) ds'} \]

\( s \): distance between \( x \) and \( x' \)

• Properties of \( Tr \):

• In vacuum \( Tr(x \rightarrow x') = 1 \)

• Multiplicative \( Tr(x \rightarrow x'') = Tr(x \rightarrow x') \cdot Tr(x' \rightarrow x'') \)

• Beer’s law (in homogeneous medium)

\[ Tr(x \rightarrow x') = e^{-\sigma_is} \]
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 \( \cos \theta = \omega \cdot \omega' \)

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

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

\[2\pi \int_0^\pi p(\cos \theta) \cos \theta \, d\theta = g\]

\(g\): average phase angle

\(g = -0.3\)

\(g = 0.6\)
Henyey-Greenstein approximation

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

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

\[
b_n = \langle p(\cos \theta), P_n(\cos \theta) \rangle = \int_{-1}^{1} p(\cos \theta)P_n(\cos \theta)d \cos \theta
\]

\[
P_0(x) = 1
\]
\[
P_1(x) = x
\]
\[
P_2(x) = \frac{1}{2}(3x^2 - 1)
\]
\[
P_3(x) = \frac{1}{2}(5x^3 - 3x)
\]
...
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

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

\[ \phi = 2\pi \xi \]

\[ \cos \theta = \begin{cases} 1 - 2\xi & \text{if } g = 0 \\ -\frac{1}{|2g|} \left(1 + g^2 - \left(\frac{1 - g^2}{1 - g + 2g \xi}\right)^2\right) & \text{otherwise} \end{cases} \]
Pbrt implementation

- `core/volume.*  volume/*`

```cpp
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)
  - $\sigma_s$ and $\sigma_a$
  - $g$ in phase function
  - Emission $L_{ve}$
  - Spatial extent

```c
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;
    Spectrum 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** \( v(s_j) = \text{trilinear}(v, i, j, k, x(s_j)) \)
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

```c
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

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

\[ L(x, \omega) = \int_0^\infty Tr(x' \to x)S(x', \omega)ds \]

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

\[ x' = x - s \omega \]
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

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

from the surface point \( x_0 \) from the participating media

\[ x' = x - s\omega \]
Simple atmosphere model

Assumptions

- Homogenous media
- Constant source term (airlight)

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

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

- Fog
- Haze
OpenGL fog model

\[ C = fC_{in} + (1 - f)C_{fog} \]

**GL_EXP**

\[ f(z) = e^{-(\text{density} \cdot z)} \]

**GL_EXP2**

\[ f(z) = e^{-(\text{density} \cdot z)^2} \]

**GL_LINEAR**

\[ f(z) = \frac{\text{end} - z}{\text{end} - \text{start}} \]

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

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

• Solution for the emission-only simplification

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

\[ L(x, \omega) = \text{Tr}(x_0 \to x)L(x_0, -\omega) + \int_0^d \text{Tr}(x' \to x)L_{ev}(x', -\omega)ds \]

• Monte Carlo estimator

\[ \frac{1}{N} \sum_{i=1}^N \frac{\text{Tr}(x_i \to x)L_{ev}(x_i, \omega)}{p(x_i)} = \frac{t_1 - t_0}{N} \sum_{i=1}^N \text{Tr}(x_i \to x)L_{ev}(x_i, \omega) \]
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
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;  // use larger steps for shadow and indirect rays for efficiency
    offset = rng.RandomFloat();
}
Spectrum tau = scene->volumeRegion->tau(ray, step, offset);
return Exp(-tau);

\[
T_\omega(s) = e^{-\tau_\omega(s)} \quad \tau_\omega(s) = \int_0^s \sigma_a(x + s' \omega, \omega) ds'
\]
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;
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;
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)

\[ L_o(p_o, \omega_o) = \int \int_{A H^2} S(p_o, \omega_o, p_i, \omega_i) L_i(p_i, \omega_i) |\cos \theta_i| d\omega_i dA \]
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 equilibrium 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

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

n=10

n=100

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 $\Phi$

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

fluence

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

diffusion coefficient

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

The fluence at $p_i$ from $p$ is

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

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

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

\[ z = z^- \]
\[ z = 2AD \]
\[ z = 0 \]
\[ z = z^+ \]

\[ d \]

\[ d \]
Dipole diffusion approximation

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

negative light at \((x_i, y_i, z^-)\)

positive light at \((x_i, y_i, z^+)\)

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

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

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

\[ L_o(p_o, \omega_o) \]

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

\[ L_o(p_o, \omega_o) \]

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

\[ \approx F_{dt}(\eta_i) E(p_i) \]

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.
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](image1.png) ![Bad, but dipole is dad for this anyway](image2.png)
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>
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>
        Calculate direct lighting only; it is possible to include indirect lighting but more expensive
        irradiancePoints.push_back(IrradiancePoint(sp, E));
    }
}

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

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

```cpp
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);
```

Computes representative position, irradiance and area for each node. Positions are weighted by irradiance values to emphasize the points with higher irradiance values.
\[
L_0(p_o, \omega_o) \approx \frac{1}{\pi} F_t(\eta, \omega_o) F_{dt}(\eta) M_o(p_o)
\]

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);
    }
}
L += UniformSampleAllLights(...);
if (ray.depth < maxSpecularDepth) {
    L += SpecularReflect(...);
    L += SpecularTransmit(...);
}
return L;
\[ M_o(p_o) = \sum_j R_d\left(\|p_j - p_o\|\right)E_j A_j \]
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.

<table>
<thead>
<tr>
<th>Material</th>
<th>$\sigma'_s \ [\text{mm}^{-1}]$</th>
<th>$\sigma_a \ [\text{mm}^{-1}]$</th>
<th>Diffuse Reflectance</th>
<th>$\eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>G</td>
<td>B</td>
<td>R</td>
</tr>
<tr>
<td>Apple</td>
<td>2.29</td>
<td>2.39</td>
<td>1.97</td>
<td>0.0030</td>
</tr>
<tr>
<td>Chicken1</td>
<td>0.15</td>
<td>0.21</td>
<td>0.38</td>
<td>0.015</td>
</tr>
<tr>
<td>Chicken2</td>
<td>0.19</td>
<td>0.25</td>
<td>0.32</td>
<td>0.018</td>
</tr>
<tr>
<td>Cream</td>
<td>7.38</td>
<td>5.47</td>
<td>3.15</td>
<td>0.0002</td>
</tr>
<tr>
<td>Ketchup</td>
<td>0.18</td>
<td>0.07</td>
<td>0.03</td>
<td>0.061</td>
</tr>
<tr>
<td>Marble</td>
<td>2.19</td>
<td>2.62</td>
<td>3.00</td>
<td>0.0021</td>
</tr>
<tr>
<td>Potato</td>
<td>0.68</td>
<td>0.70</td>
<td>0.55</td>
<td>0.0024</td>
</tr>
<tr>
<td>Skimmilk</td>
<td>0.70</td>
<td>1.22</td>
<td>1.90</td>
<td>0.0014</td>
</tr>
<tr>
<td>Skin1</td>
<td>0.74</td>
<td>0.88</td>
<td>1.01</td>
<td>0.032</td>
</tr>
<tr>
<td>Skin2</td>
<td>1.09</td>
<td>1.59</td>
<td>1.79</td>
<td>0.013</td>
</tr>
<tr>
<td>Spectralon</td>
<td>11.6</td>
<td>20.4</td>
<td>14.9</td>
<td>0.00</td>
</tr>
<tr>
<td>Wholemilk</td>
<td>2.55</td>
<td>3.21</td>
<td>3.77</td>
<td>0.0011</td>
</tr>
</tbody>
</table>
Marble: BRDF versus BSSRDF
Marble: MCRT versus BSSRDF

MCRT

BSSRDF
Milk

surface reflection  translucency
Skin

surface reflection  translucency