

Volume and Participating Media

Digital Image Synthesis

Yung-Yu Chuang

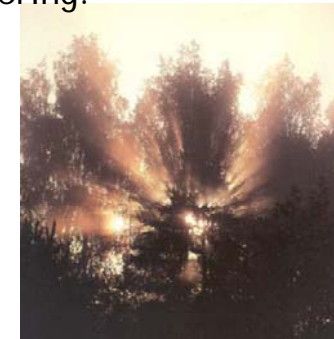
12/31/2008

with slides by Pat Hanrahan and Torsten Moller

Participating media



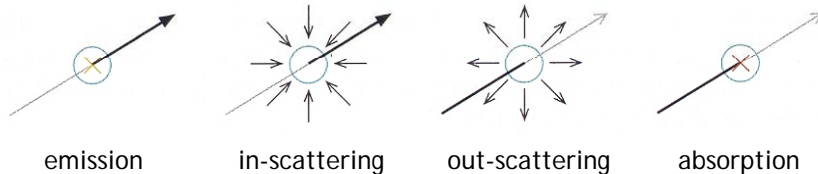
- We have by far assumed that the scene is in a vacuum. Hence, radiance is constant along the ray. However, some real-world situations such as fog and smoke attenuate and scatter light. They participate in rendering.
- Natural phenomena
 - Fog, smoke, fire
 - Atmosphere haze
 - Beam of light through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



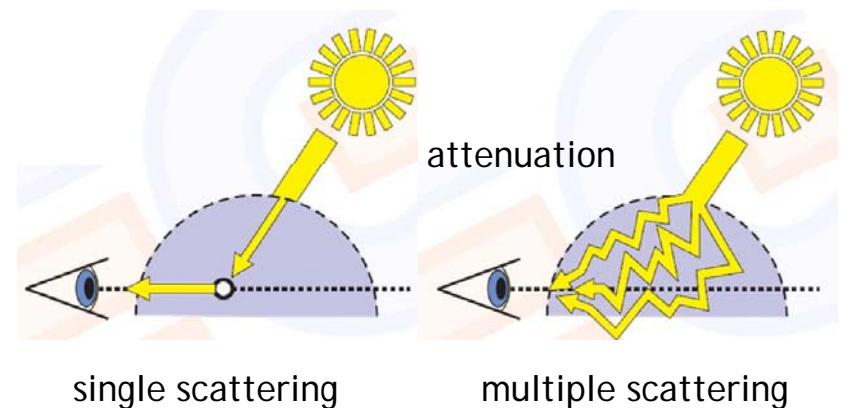
Volume scattering processes



- 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)



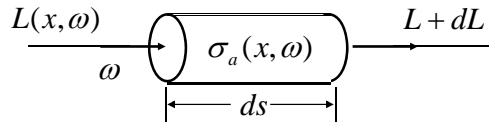
Single scattering and multiple scattering



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



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

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

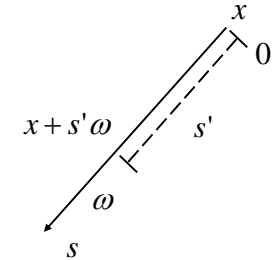
$$\ln L(x+s\omega, \omega) - \ln L(x, \omega) = -\int_0^s \sigma_a(x+s'\omega, \omega)ds' = -\tau_\omega(s)$$

Optical distance or depth

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

Homogenous media: constant σ_a

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



Transmittance and opacity



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

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

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

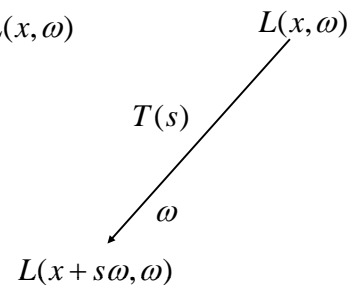
$$L(x+s\omega, \omega) = e^{-\tau_\omega(s)}L(x, \omega) = T_\omega(s)L(x, \omega)$$

Transmittance

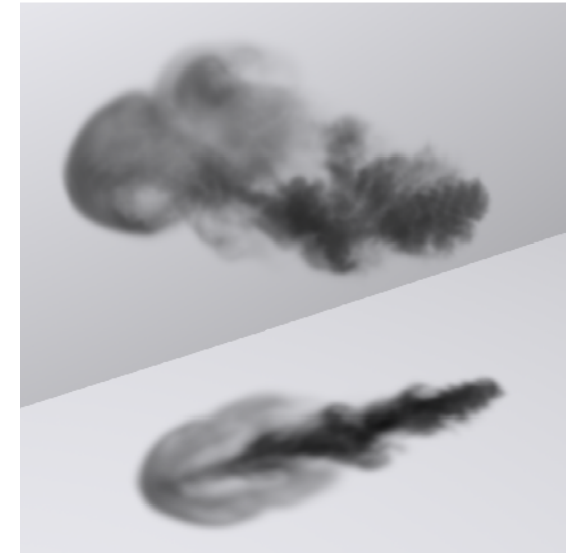
$$T_\omega(s) = e^{-\tau_\omega(s)}$$

Opacity

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



Absorption

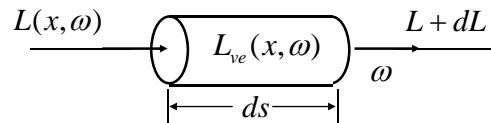


Emission

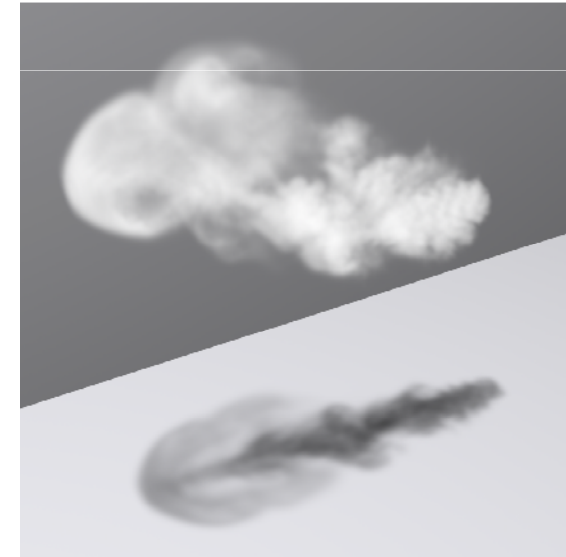


- Energy that is added to the environment from luminous particles due to chemical, thermal, or nuclear processes that convert energy to visible light.
- $L_{ve}(x, \omega)$: emitted radiance added to a ray per unit distance at a point x in direction ω

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



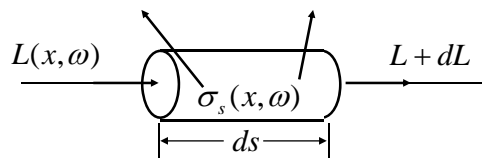
Emission



Out-scattering



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

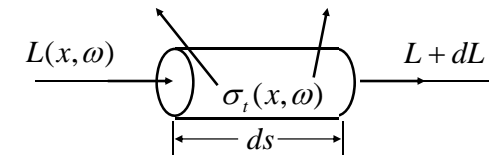


$$dL(x, \omega) = -\sigma_s(x, \omega) L(x, \omega) ds$$

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

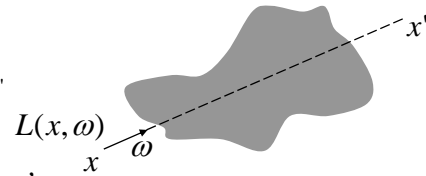
$$\tau_\omega(s) = \int_0^s \sigma_t(x + s', \omega) ds'$$

Extinction



- Beam transmittance

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



s : distance between x and x'

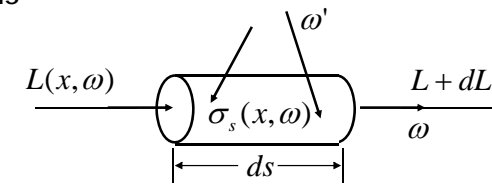
- Properties of Tr :
- In vacuum $Tr(x \rightarrow x') = 1$
- Multiplicative $Tr(x \rightarrow x'') = Tr(x \rightarrow x') \cdot Tr(x' \rightarrow x'')$
- Beer's law (in homogeneous medium)

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

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)

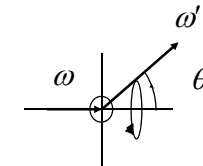
Phase functions



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

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)

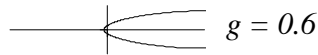
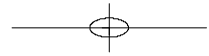
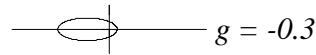
3. Mie scattering

- 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

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



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

g: average phase angle

Henyey-Greenstein approximation

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

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

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

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

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/*
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

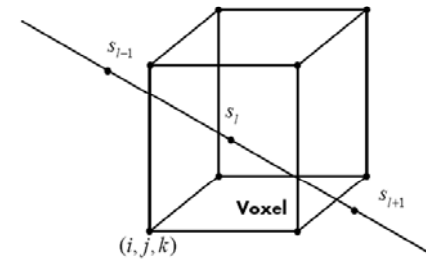


```
class DensityRegion : public VolumeRegion {
public:
    DensityRegion(Spectrum &sig_a, Spectrum &sig_s,
        float g, Spectrum &le, Transform &VolumeToWorld);
    float Density(Point &Pobj) const = 0;
    sigma_a(Point &p, Vector &) {
        return Density(WorldToVolume(p)) * sig_a; }
    Spectrum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

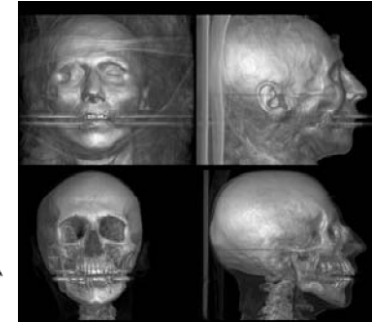
VolumeGrid



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



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



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

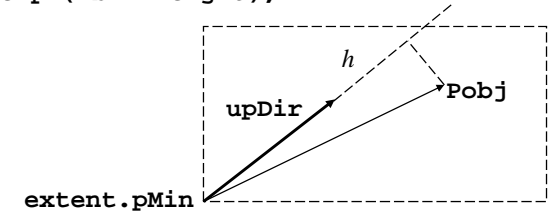


ExponentialDensity



```
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

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

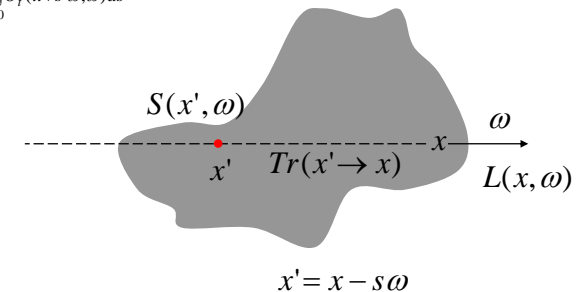
Infinite length, no surface



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

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

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



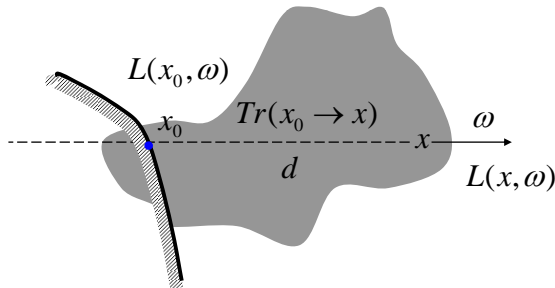
With surface



- The solution

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

from the surface point x_0



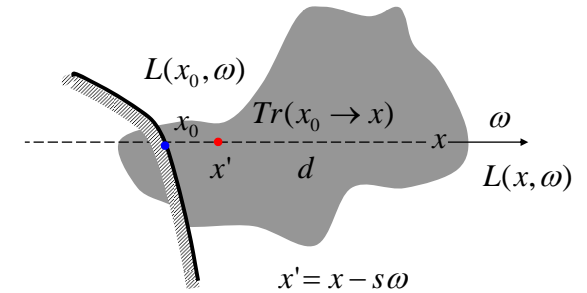
With surface



- 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

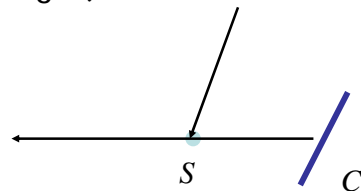


Assumptions

- Homogenous media
- Constant source term (airlight)

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

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

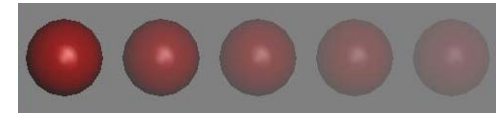


- Fog
- Haze

OpenGL fog model



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



GL_EXP

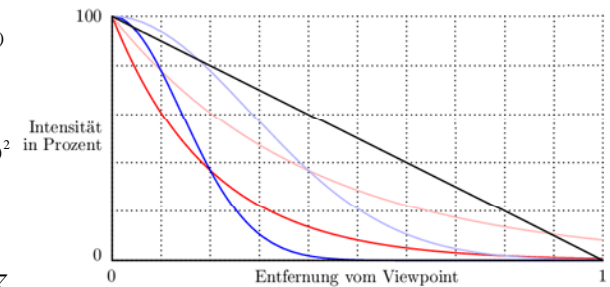
$$f(z) = e^{-(density \cdot z)}$$

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/gIFog>

VolumeIntegrator



```
class VolumeIntegrator : public Integrator {  
public:                                Beam transmittance for a given  
    virtual Spectrum Transmittance(ray from mint to maxt  
        const Scene *scene,  
        const Ray &ray,  
        const Sample *sample,  
        float *alpha) const = 0;  
};
```

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

Emission only



- Solution for the emission-only simplification

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

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

- 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(Sample *sample, const Scene *scene);  
    Spectrum Transmittance(const Scene *, const Ray &ray, const Sample *sample, float *alpha) const;  
    Spectrum Li(const Scene *, const RayDifferential &ray, const Sample *sample, float *alpha) const;  
private:  
    float stepSize;  
    int tauSampleOffset, scatterSampleOffset;  
}; single 1D sample for each
```

EmissionIntegrator::Transmittance



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

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

EmissionIntegrator::Li



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

EmissionIntegrator::Li



```
for (int i = 0; i < N; ++i, t0 += step) {
    // Advance to sample at t0 and update T
    pPrev = p;
    p = ray(t0); Tr(xi → x) = Tr(xi → xi-1) · Tr(xi-1 → x)
    Spectrum stepTau = vr->Tau(Ray(pPrev, p-pPrev, 0, 1),
        .5f * stepSize, RandomFloat());
    Tr *= Exp(-stepTau);
    // Possibly terminate if transmittance is small
    if (Tr.y() < 1e-3) {
        const float continueProb = .5f;
        if (RandomFloat() > continueProb) break;
        Tr /= continueProb;
    }
    // Compute emission-only source term at _p_
    Lv += Tr * vr->Lve(p, w);
}
return Lv * step;
```

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

Emission only



exponential density

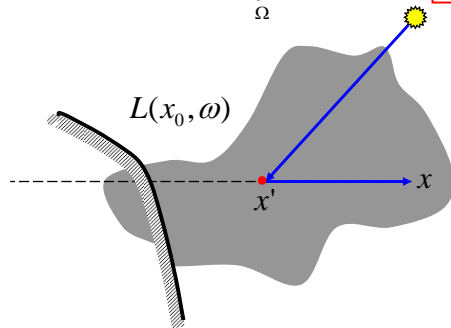
Single scattering



- Consider incidence radiance due to direct illumination

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

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



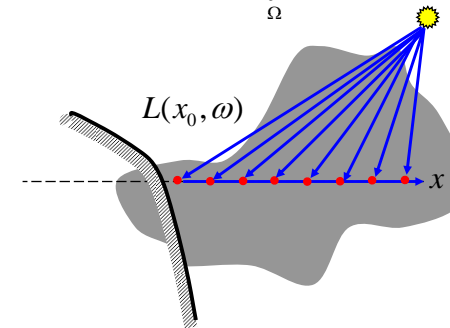
Single scattering



- 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

