Volumentric Rendering with Homogeneous media

Volume Scattering · Phase Functions · Media · Volumetric Path Tracer · Notes

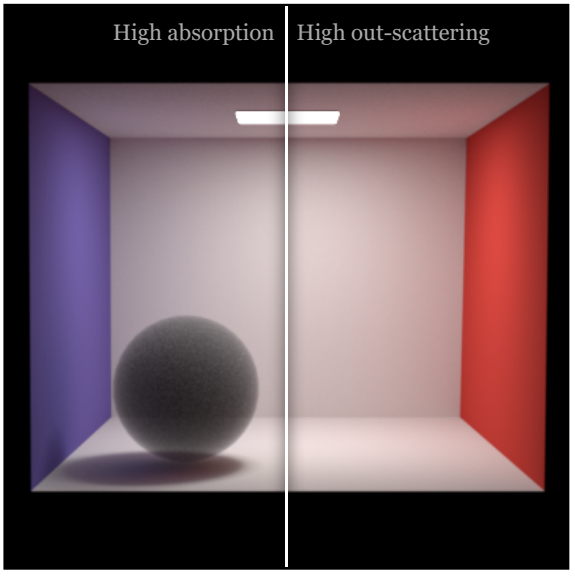
Author:Kyle Beebe

1 Volume Scattering

The three main processes affecting light transport within media are absorption, emission, and scattering. -Absorption represents a loss of radiance within a medium. -Emission represents a gain in radiance due to a medium producing light. -Scattering represents either light gained from rays bouncing into the medium in a same direction as another light ray or the loss of radiance due to light rays leaving a medium.

In regards to this implementation and what is accounted for is, absorption via an absorption coefficient σ_a, out-scattering via σ_s, a single in-scattering event, and a total reduction in radiance due to outscattering and absorption via $\sigma_t = \sigma_s + \sigma_a$ Both absorption and out-scattering coefficients can be in the range of 0 to infinity.

http://www.pbr-book.org/3ed-2018/Volume_Scattering/Volume_Scattering_Processes.html



High absorption:

- samples:2000
- homogeneous sphere medium, isotropic, $\sigma_s = 1$ $\sigma_a = 5$
- 400×400
- 8.2 min

High out-scattering:

- samples:2000
- homogeneous sphere medium, isotropic, $\sigma_s = 5$ $\sigma_a = 1$
- 400×400
- 9.4 min

2 Phase Functions

Similar to using BSDF models the describe the scattering events at surfaces, volume scattering can be described using a phase function. There exists isotropic and anisitropic phase functions which determine the manner in which light is scattered at a point in a medium.

Implementation: (phasefunction.h)

The PhaseFunction class is an abstract class containing a struct definition PhaseQueryRecord, which is passed to a phase function's functions containing and being popualted with incoming and outgoing light directions needed and resulting from a phase function evaluation or sampling. The virtual fucntions outlined are:

- eval() - Evaluates the phase function depending on the incoming and outgoing directions and returns a color value representing the probability evaluation.
- pdf () - returns the probability density sampling an outgoing direction
- sample() - samples the phase function, populates the outgoing direction in the PhaseQueryRecord parameter, and returns the pdf.

.

2.1 Isotropic

An isotropic phase function is defined by equal probability of scattering in any direction from a point. This phase function is useful for modeling such things as a uniform fog with no ideal direction of scattering. For a pair of outgoing and incoming light directions, this probability is:

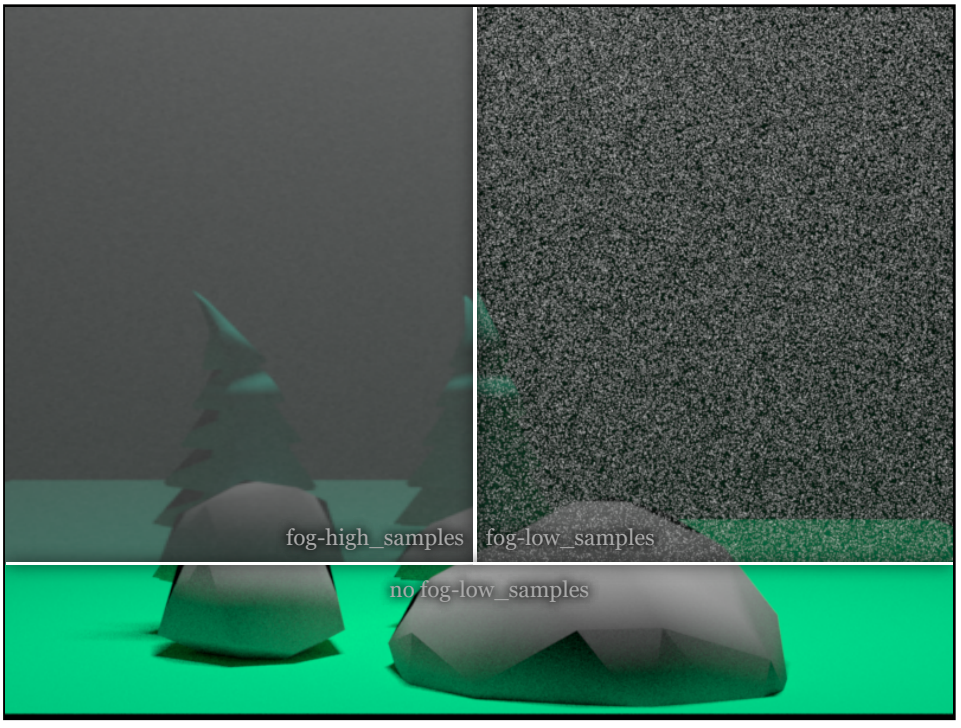
$$p(\omega_o, \omega_i) = \frac{1}{4\pi}$$

This is because the chance to scatter light is the same through the whole uniform sphere of directions.

Implementation: (isotropic.cpp)

- eval() - returns an rgb color value containing $\frac{1}{4\pi}$ for each channel
- pdf () - returns $\frac{1}{4\pi}$
- sample() - samples an outgoing direction using the sample warping function Warp::squaretoUniformSphere() and returns the pdf $\frac{1}{4\pi}$

Results:



Fog high samples:

- samples:50000
- homogeneous scene medium, isotropic, $\sigma_s = \sigma_a = 0.1$
- 400×400
- 1.6 hours

Fog low samples:

- samples:100
- homogeneous scene medium, isotropic, $\sigma_s = \sigma_a = 0.1$
- 600×800
- 38sec

No fog low samples:

- samples:100
- homogeneous scene medium, isotropic, $\sigma_s = \sigma_a = 0.1$
- 600×800
- 27sec

2.2 Henyey-Greenstein

The Henyey-Greenstein model was used for media requiring an anisitropic phases function. This model has a floating point parameter (g) in the range of (−1,1) representing the direction of scattering. Negative values represent back scattering while positive values represent forward scattering. The closer to −1 or 1 the value the more scattering occurs towards incoming or outgoing directions.

This model can evaluate the same as isotropic phase functions when the (g) parameter is set to 0. As in, scattering is not directed more either forward or backwards. The need for an explicit isotropic function helps lessen the use of wasted floating point computations involving zero.

The pdf of this model being:

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

Where the cos θ is the cosine of the angle between the incoming and outgoing light directions.

Implementation: (henyey-greenstein.cpp)

- eval() - returns an rgb color value containing the pdf for each channel
- pdf () - returns the probability density based on parameter g
- sample() - samples an outgoing direction returns the pdf

Sampling:

Since the the pdf can be seprated into two spherical components θ and ϕ where

$$p(\phi) = \frac{1}{2\pi}$$

We just need to sample $cos(\theta)$ to get theta and convert the spherical coordinate $(1, \theta, \phi)$ to cartesian.

$$cos(theta) = \frac{1}{2g}(1+g^2-(\frac{1-g^2}{1-g+2g\xi})^2)$$

if g $\neq$ 0

otherwise $cos(\theta) = 1 - 2\xi$

Where ξ is a uniforma random sample between 0 and −1.

The sampling method populates the PhaseQueryRecord with the sampled outgoing direction and returns the pdf.

http://www.pbr-book.org/3ed-2018/Light_Transport_II_Volume_Rendering/Sampling_Volume_Scattering.html

Results:

3 Media

Participating media can be separated into homogeneous and heterogeneous media. Homogeneous media being a volume consisting of the same density throughout it, and heterogeneous having different densities within the volume.

Multiple media can be contained within a scene and can be bound or unbound by a surface. In my implmentation all media must be attatched to a Mesh object (mesh.h) which gives it a defineable boudnary to do ray intersect calculations. This is true for all media except for a single scene medium which represents an unbounded medium that spans the whole scene and is attached to the scene itself. Each mesh has a refrence to both an internal and external medium in order to successfully account for media within another medium. Additionally, media can have a mesh boundary with a defined surface via a bsdf that has been assigned to account for surface reflections along with transmittance through the internal medium.

Implementation: (medium.h)

Abstract class representing either a homogeneous or heterogeneous medium.

Defines a MediumQueryRecord struct containing information needed for medium sampling and transmittance calculations such as the phase function, absorbtion and outscattering coefficients, and the evaluated point p along a ray.

Contains fields for the:

- σ_s outscattering coefficient
- σ_a absorbtion coefficient
- σ_t extinction coefficient
- The assigned phase function

Functions include:

- Tr() which calculates the transmittance value along a ray
- distSample() which randomly samples a distance along a ray and popualtes the Medium with the relevant sampled point and medium information if a point withing the medium was sampled. If a surface or point outside the medium was sampled, a surface interaction is assumed.

3.1 Homogeneous Medium

This type of medium has the same σ_a and σ_s throughout. Due to this, Beer's law

$$T_r(p \rightarrow p') = e^{-\sigma_t d}$$

That is, the trasmittance along a line segment from p to p'. Where d is the length of the this line segment.

Implementation: (homogeneous.cpp)

tr() the function for transmittance returns the evaluation using Beer's law with a passed in ray. The distance d is computed by subtracting the ray's min t value from the max. The return value is in the form of a rgb color of this evaluation.

distSample() distance sampling function

Sampling: The idea is to sample a point along the passed in ray that can either generate a point still in the medium or beyond the medium, which is essentially a surface. First, randomly sample a rgb channel from the extinction coefficient. This is opposed to having to randomly sample three distances accounting for all three channels then averaging them. It saves computations. The sampling method for an exponential distribution (Beer's law equation)

$$f(t) = e^{-\sigma_t t} \rightarrow t = -\frac{ln(1-\xi)}{\sigma_t}$$

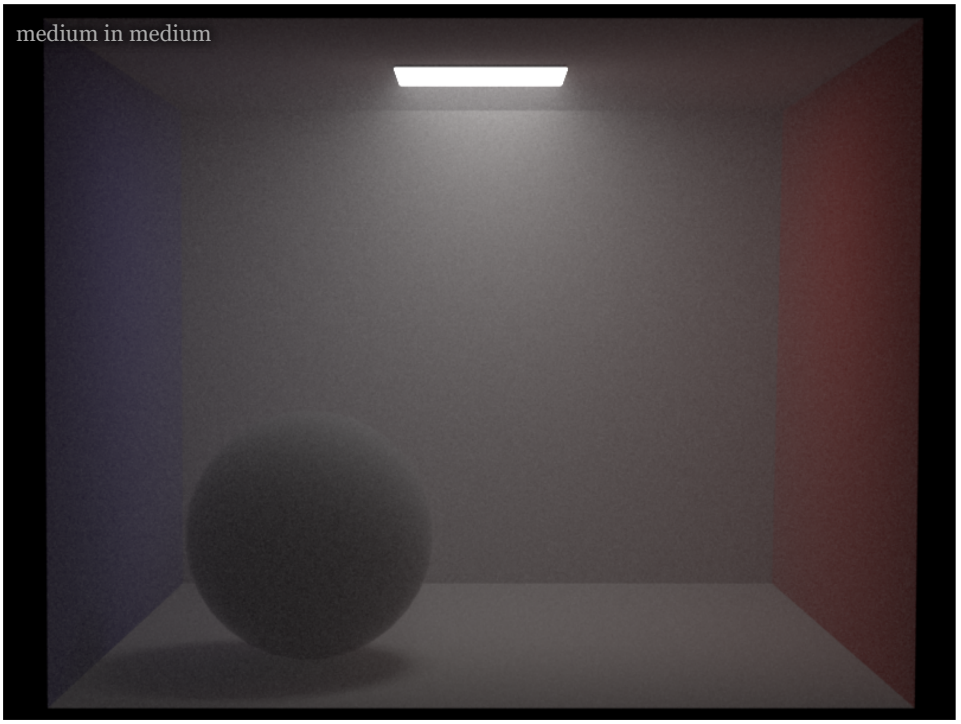
with a pdf

$$p(t) = \sigma_t e^{-\sigma_t t}$$

Once this t value is evaluated, if it is less than the passed in ray's max t value, then it is distance sampled in the medium and all of the appropriate fields in the MediumRecordQuery are filled out, including the point on the ray at time t. Else it represents a surface sampling. Either way, the pdf of either sampling a medium point or surface point is filled in, and the return value is the computed transmittance multiplied by σ_s divided by the pdfin the case of a medium sample or just the transmittance divided by the pdf for a surface sample.

http://www.pbr-book.org/3ed-2018/Light_Transport_II_Volume_Rendering/Sampling_Volume_Scattering.html

Results:



Medium in medium:

- samples:20000
- homogeneous scene medium $\sigma_s = 0.3$ $\sigma_a = 0.1$, henyey-greenstein g = 0.0
- homogeneous sphere medium (no brdf) $\sigma_s = 5$ $\sigma_a = 5$, henyey-greenstein g = 0.0
- 600×800
- 40min

4 Volumetric Path Tracer

The path tracer for volumetric light transport in my implementation accounts for the direct lighting at a given point and a single scattering event at that point within a medium or surface.

Implementation:

(ray.h)

A medium pointer was added to each ray to identify if the current ray being tested is wholly contained in a medium. It is assumed to always be entirely in a single medium or not.

(mesh.h)

Now contains pointers to internal and external mediums. A mesh can be surfaced, in which it has a bsdf, or not. It can contain a medium (internal) and be surrounded by a medium (external). For this implementation, mesh mediums can only be contained within a scene medium, and not withing other mesh mediums.

(scene.h)

A function rayIntersectTr() was added to account for volumetric transport that allows a ray to completely pass through non-surfaced mediums when testing for mesh surface instersection. The current ray's medium will be changed each time it passes into and out of a medium. This is based on the mesh objects internal and external medium pointers It accumulates a transmittance computed from each passed through medium along the way.

(volume_single.cpp)

This is the actual path tracing integrator.

Li() function returns the total lighting value starting with the camera ray and the accumulation from bounces For a passed in camera ray to the Li() function, there is a looping to account for each time a bounce occurs with a tracking variable for the current depth of bounces ,the ray being bounced, and the accumulated light being transferred along with the attenuation coefficient. A light ray can only stop bouncing after 3 bounces and has a 5% chance before computing the next bounce (.95 survival value). At depth 0, the camera ray will have its medium set to the scene medium.

The first stage of the loop tests for a ray intersection in the scene. An intersection will change the ray's max t value to shorten the rays for transmittance calculations. A non intersection means the ray exited the scene and thus the accumulated light should be returned and the evaluation of a light ray should end. Otherwise the ray's medium is checked to see if it is currently in a medium. Being in a medium means the medium is distance sampled. A distance sampled will either be in the medium or beyond at the surface of the previously found intersection.

In the case of a sampled point along the ray being in the medium, the multiple importance sampled direct lighting will be calculated via sampleLiDirect() at the sampled point. This will be factored into the current incident light weighted by the attenuation coefficient, then a new direction will be sampled using the medium's phase function and a new ray will be created at the current point in the new direction signifying in scattering to the medium.

In the case of a sampled point along the ray being on the intersected surface, if the intersected mesh doesn't have a material (no bsdf), it is a unmaterialed medium. The ray medium will change and a new ray will be created at the point in the same direction. Otherwise, it is a materialed surface intersection. Likewise with the medium, the MIS direct lighting is calculated via the sampleLightDirect() function. This is factored into the current total incident light weighted by the attenuation. For a surface, the bsdf is sampled for a direction. A new ray is created from this direction signifying a bounce.

sampleLightDirect() function takes a ray intersection structure and a ray and randomly samples a light(emitter mesh) in the scene then passes everything to the LiDirect() function to compute the MIS direct lighting.

LiDirect() function is given an intersection event, a ray, and a randomly sampled light.

It first randomly samples a point on the given light source. If the intersection event is for a surface intersection, the bsdf is evaluated using the incoming ray direction and direction towards the emitter and importance sampled based on probability if sampling the given light using the bsdf. If the intersection event is instead a medium distance sampled event, the phase fucntion is instead evaluated.

Next, either the bsdf or phase function is sampled based on the passed in insection event. A ray is tested for intersection in the sampled direction and the probability weighted against whether the what is intersected in the direction was the sampled light or not.

In both sampling events, the direct lighting is MIS weighted after appropriate conversions between surface areas and solid angles and weighted by trasmittance when checking for intersections to a light source.

http://www.pbr-book.org/3ed-2018/Light_Transport_II_Volume_Rendering/Volumetric_Light_Transport.html

5 Notes

- Heterogeneous mediums (grid.cpp) were not fully implemented. The most of the code exists but I couldn't determine a way to place a heterogeneous medium in the scene

without a mesh bounding the volume itself.

- non-zero g values for henyey-greenstein phase functions sometimes throw NaN radiance exceptions that I didn't handle
- I couldn't determine a way to test the forward and back scattering due to positive and negative g values in the henyey-greenstein phase function.