Fast, Flexible, Physically-Based Volumetric Light Scattering

Nathan Hoobler, NVIDIA Developer Technology

March 16th, 2016
Polygonal Volumetric Lighting
Fast, Flexible, and Physically-based

- Fast: Scalable performance on all hardware
- Flexible: Minimally invasive to integrate into existing engines
- Physically-based: Easy mapping to real-world phenomenon

*Source will be available on GitHub to registered developers*
Background & Motivation
Algorithm Overview
Integration into Fallout 4
Background & Motivation
Light Propagation in a Vacuum
Light Propagation in a Vacuum

Directly Visible Light-Source

Direct Radiance:

\[ L_D = L(\omega_x) \]
Light Propagation in a Vacuum

Direct Illumination

Direct Radiance:
\[ L_D = L(\omega_s)\rho_S(l, s, \omega_x) \]
Light Propagation in a Vacuum

Shadows

Direct Radiance:

\[ L_D = L(\omega_s)\rho_s(l, s, \omega_x)V(s, l) \]
Participating Media
Transmittance & Extinction

Incident Light

Transmitted Light
Participating Media
Transmittance & Extinction

Incident Light

Absorption

Out-Scattering

Transmitted Light
Participating Media
Transmittance & Extinction

Direct Radiance:
\[ L_D = L(\omega_s)\rho_s(l, s, \omega_x)V(s, l) \]

Transmitted Radiance:
\[ L_T = L_D T(l, s)T(s, x) \]
Participating Media

Beer-Lambert Law

Transmittance:

\[ T = 1 - \frac{\phi_s}{\phi_i} \times \frac{\phi_a}{\phi_i} = 1 - \frac{\phi_{ex}}{\phi_i} \]

\[ T(a, b) = e^{-\tau|b-a|} \]

Optical Thickness:

\[ \tau_x = -\ln(1 - \frac{\phi_x}{\phi_i}) \]

\[ \tau_{ex} = \tau_s + \tau_a \]
Participating Media
Transmittance & Extinction

Direct Radiance:
\[ L_D = L(\omega_S)\rho_S(l, s, \omega_X)V(s, l) \]

Transmitted Radiance:
\[ L_T = L_D e^{-\tau_{ex}(\bar{l}s + \bar{s}x)} \]
Participating Media

In-Scattering

Incident Light

Absorption

Transmitted Light

Out-Scattering

gameworks.nvidia.com
Participating Media

In-Scattering

In-Scattering

Incident Light

Absorption

Out-Scattering

Transmitted Light
Participating Media
In-Scattering

Direct Radiance:
\[ L_D = L(\omega_S) \rho_l(l, s, \omega_x) V(s, l) \]

Transmitted Radiance:
\[ L_T = L_D e^{-\tau_{ex}(ls + sx)} \]

In-Scattered Radiance:
\[ L_S = \int_0^{sx} L(t, l) V(t, l) \rho_m(\overrightarrow{tl}, \omega_x) e^{-\tau_{ex}(lt + tx)} dt \]
Phase Functions

Overview

Volume analog of BRDF

Directional distribution of energy relative to incident direction

Determined by the ratio of medium particles and their size relative to the light

Phase function for atmosphere on a clear day
Phase Functions
Complex Phenomenon

Can be Wavelength-dependent

Can become arbitrarily complex depending on atmospheric conditions

We can approximate common conditions with simple models
Algorithm Overview
Interval Integration

Overview

In-Scattered Radiance:

\[ L_S = \int_{S_X}^{S_X} L(t, l)V(t, l)\rho_m(tl, \omega_x)e^{-\tau_{ex}(\overline{lt}+\overline{tx})}dt \]
Interval Integration
Non-Constant Visibility

In-Scattered Radiance:

\[
L_S = \int_{0}^{\overline{sx}} L(t, l) \mathcal{V}(t, l) \rho_m (\vec{tl}, \omega_x) e^{-\tau_{ex}(\overline{lt} + \overline{tx})} dt
\]
Interval Integration

Sum of Intervals

In-Scattered Radiance:

\[ L_{S'}(t) = L(t, l) \rho_m \left( \vec{t} l, \omega_x \right) e^{-\tau_{ex}(\vec{t} t + \vec{t} x)} \]

\[ L_S = \int_{t_1}^{t_2} L_{S'}(t) \, dt \]

\[ L_S = \int_x^{t_1} L_{S'}(t) \, dt + \int_{t_2}^{s} L_{S'}(t) \, dt \]
Interval Integration
Sum & Difference of Intervals

In-Scattered Radiance:
\[ L_{S'}(t) = L(t, l) \rho_m \left( \hat{t}l, \omega_x \right) e^{-\tau_{ex}(\bar{t}t + \bar{t}x)} \]

\[ L_{S'}(d) = \int_0^d L_{S'}(t)dt \]

\[ L_S = L_{S'}(t_1) - L_{S'}(t_2) + L_{S'}(s) \]
Interval Integration
Sum & Difference of Intervals

In-Scattered Radiance:
\[ L_{S'}(t) = L(t, l) \rho_m(t_l, \omega_x) e^{-\tau_{ex}(t_l+t_x)} \]

\[ L_{S'}(d) = \int_0^d L_{S'}(t) dt \]

\[ L_S = L_{S'}(t_1) - L_{S'}(t_2) + L_{S'}(s) \]

\[ L_S = \sum_{n \in G_f} L_{S'}(t_n) - \sum_{m \in G_b} L_{S'}(t_m) \]
Interval Integration

Intervals from Mesh

In-Scattered Radiance:

\[ L_S = \sum_{n \in G_f} L_{S'}(t_n) - \sum_{m \in G_b} L_{S'}(t_m) \]

Use front and back faces of light volume as integration intervals.
No Scattering
Scattering Only
Workflow
High-Level API

Begin Accumulation
- Scene Depth
- Medium Definition

Render Volume
- Light Shadow Map

Filter and Composite
- Scene Depth
- Scene Render

gameworks.nvidia.com
Workflow

Generic Pipeline Example

G-Buffer Render → Render Shadow Map → Apply Light → Alpha-Blend Render → Post-Process

Begin Accumulation → Render Volume → Filter and Composite

gameworks.nvidia.com
Medium Specification

Multiple Phase Terms

- **Density** - Optical depth of that term in <RGB>
- **Phase Function** - Directional scattering distribution (applied per-channel)
- **Phase Parameters** - Eccentricity (for Henyey-Greenstein)

**Absorption** - Optical depth of light absorption <RGB>

Terms are summed together per-channel to produce composite medium
Medium Specification

Rayleigh Scattering

Particles much smaller than wavelength of light (O2, N2, etc.)

Generally constant at a given altitude

Wavelength dependent

Relative optical depth for CIE-RGB:

\[
\rho(\theta) = \frac{3}{16\pi} (1 + \cos^2(\theta))
\]

R: 0.596x10^{-5}

G: 1.324x10^{-5}

B: 3.310x10^{-5}
Medium Specification

Mie-Lorenz Approximation

Mie-Hazy: Light fog

\[ \rho(\theta) = \frac{1}{4\pi} \left( \frac{1}{2} + \frac{9}{2} \left( \frac{1 + \cos \theta}{2} \right)^8 \right) \]

Mie-Murky: Dense fog

\[ \rho(\theta) = \frac{1}{4\pi} \left( \frac{1}{2} + \frac{33}{2} \left( \frac{1 + \cos \theta}{2} \right)^{32} \right) \]
Medium Specification
Henyey-Greenstein Function

Tunable phase function with a specified eccentricity $g = (-1, 1)$

- $g < 0$: back-scattering
- $g = 0$: isotropic scattering
- $g > 0$: forward scattering

Multiple terms can be combined to approximate complex functions
Volume Rendering

Overview

Convert light description + shadow map into geometry

Solve Integral at each intersection and add or subtract based on facing

Different solvers based on light type

- **Directional Light** - Analytical Solution
- **Omnidirectional Light** - Look-up Texture
- **Spot Light** - Look-up Texture or Numerical Integration
Volume Rendering

Directional Light

\[ L_{S'}(d) = L(d, l) \rho_m(xl, \omega_x) \frac{1 - e^{-\tau_exd}}{\tau_ex} \]

With some assumptions we can simplify the integral for directional light

- **Constant Direction** (parallel light-source)
- **Constant Power** (Light distance >> medium depth)

Reduces to analytic function evaluated in pixel-shader
Volume Rendering

Omnidirectional Light

Local lights too complex for analytical solutions

All terms can be expressed in with respect to direction and distance

Create a Look-up texture:
- Map to 2D space
- evaluate differential
- Numerically integrate with CS

\[
\cos^{-1}(\hat{v} \cdot \hat{l})
\]

\[d\]
Volume Rendering

Spot Lights

Spotlights are worse case than omnidirectional lights

Angular falloff complicates integral

Interval needs to be clamped to cone intersection

Can be simplified

• No Falloff
• Fixed Falloff
• Variable Falloff
Volume Rendering

Numerical Integration

No Falloff: Treat as point light

Fixed Falloff: Use modified point LUT

For other cases, integrate in the PS

1) Do cone intersection as before
2) Numerically integrate (Newton-Cotes)
3) Use the result as the integral
Apply Lighting
MSAA/Temporal Resolve

Use MSAA when down-sampling
  • Preserves shading rate savings
  • Improves edge quality

Resolve MSAA accumulation buffer before compositing
If temporal sampling, feed resolved accumulation through TAA system
Apply Lighting
Composite Results

Bilateral upsampling: helpful at \( \frac{1}{4} \)-resolution, situational at \( \frac{1}{2} \)-resolution

Additively blend resolved/filtered output with scene

Use dual-source blending to attenuate based on medium+depth
Integration into *Fallout 4*
Fallout 4 Integration

Feb: Development/Integration Begins
March: Artists authoring environments
April-June: Console Ports
July-August: Optimization (PC+Console)

Dedicated NVIDIA QA resources
Indoor, Dusty (Final Image)
Indoor, Dusty
(In-Scattering Only)
Outdoors, Clear
(No Scattering)
Outdoors, Clear (In-Scattering Only)
Outdoors, Foggy
(No Scattering)
Outdoors, Dusty (No Scattering)
## Gpu Performance

Concord, High Quality @ 1080p

<table>
<thead>
<tr>
<th></th>
<th>BEGIN</th>
<th>RENDER VOLUME</th>
<th>FILTER &amp; COMPOSITE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GTX 980 Ti</strong></td>
<td>0.037 ms</td>
<td>0.908 ms</td>
<td>0.264 ms</td>
<td>1.209 ms</td>
</tr>
<tr>
<td><strong>GTX 970</strong></td>
<td>0.049 ms</td>
<td>1.364 ms</td>
<td>0.389 ms</td>
<td>1.802 ms</td>
</tr>
<tr>
<td><strong>Radeon Fury X</strong></td>
<td>0.038 ms</td>
<td>2.581 ms</td>
<td>0.305 ms</td>
<td>2.924 ms</td>
</tr>
</tbody>
</table>

[gameworks.nvidia.com](http://gameworks.nvidia.com)
## Gpu Performance
Concord, Medium Quality @ 1080p

<table>
<thead>
<tr>
<th></th>
<th>BEGIN</th>
<th>RENDER VOLUME</th>
<th>FILTER &amp; COMPOSITE</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 980 Ti</td>
<td>0.035 ms</td>
<td>0.415 ms</td>
<td>0.394 ms</td>
<td>0.844 ms</td>
</tr>
<tr>
<td>GTX 970</td>
<td>0.049 ms</td>
<td>0.610 ms</td>
<td>0.565 ms</td>
<td>1.224 ms</td>
</tr>
<tr>
<td>Radeon Fury X</td>
<td>0.041 ms</td>
<td>1.074 ms</td>
<td>0.636 ms</td>
<td>1.751 ms</td>
</tr>
</tbody>
</table>

[gameworks.nvidia.com](http://gameworks.nvidia.com)
INTEGRATION TIPS

- Maximize Dynamic Range
- Make sure shadow maps are consistent
- Temporal filter low-res effects separately
- Be aware of worst-case scenes
Implementation Issues
Reduced Dynamic Range Limits Contrast

Problem: Hard to get intense effects without washing out scene
Intensity proportional to light source power
Real-world effects involve 10,000:1 contrast between source and scene!
Baked-in ambient normalizes intensity between bright and dark areas
Simply increasing medium density causes wash-out

Solution: Need HDR with real adaptation between dark and bright
Implementation Issues
Shadow Map/Scene Inconsistencies

Problem: Shadow Map inconsistencies become much more noticeable
Implementation Issues
Shadow Map/Scene Inconsistencies

Problem: Shadow Map inconsistencies become much more noticeable

“Bug” in art, but not noticeable because surface and shadows not usually visible

May only render front-faces to shadow map, but there may not have consistent geometry/alpha masks on both sides

May not render “distant” occluders to the shadow map

May use a separate, high-detail map for certain occluders

Solution: be consistent with your shadow map contents!
Implementation Issues
Temporal Jittering Causes Flicker

**Problem:** Temporal AA jitter causes flickering

Temporal AA jitters to increase effective resolution, then filters to smooth

Library runs at ½ - ¼ resolution to improve performance

A 1 px flicker at full-res could become 4x4 px in the down-sampled buffer!

Full-res temporal filter not designed to smooth artifacts that large

**Solution:** Added separate TAA resolve to down-sampled buffers
Implementation Issues
Perf Drop with High Frequency Occluders

Problem: Poor perf in specific views
Cost proportional to total pixel coverage
Dense occluders are no problem but create overhead at low angles
Ex: Sunset through the woods

Solution: Adjust tessellation factor based on view angle
Solution: Pre-filter shadow map
Gameworks Volumetric Lighting is...

- Fast enough for entire spec range
- Flexible enough for almost any engine
- Compatible with physically-based engines
- Currently available in DirectX 11 (with ports being added according to demand)

http://developer.nvidia.com
Questions?


