Real-Time Ray Tracing and Denoising

Real-time Ray Tracing

- Real-time vs. offline rendering
- Acceleration structures for dynamic scenes

Denoising

- Bilateral, joint-bilateral, NLM filters
- CNN-based denoising
Rasterization vs. Ray Tracing

Ray Tracing (primary visibility):

- For each pixel
  - For each triangle
    - Record closest visible point

Rasterization:

- For each triangle
  - For each pixel
    - Record closest visible point

Note: acceleration structures elided!
Accurate Incoherent Visibility

Rasterization

Ray Tracing
Ray tracing is a first-class member of the GPU pipeline
Tight integration between raster and compute
Shared buffers, textures, shader code
(Video)
Real-Time vs. Offline Rendering

Different constraints:

- **Offline**: render until quality achieved
- **Real-time**: render until time runs out

Implications:

- **Performance eval**: average vs. max time
- **Many fewer rays/image in real-time**
Implications: Acceleration Structures

Time to build BVH/kd-tree matters much more for real-time ray tracing

■ For real-time, can allow a few ms / frame: e.g. @10M tris, 60fps, need 600M tris / sec.
■ pbrot BVH construction: ~2.5M tris / second

⇒ Hierarchy construction efficiency really matters
⇒ Hierarchy quality is (a little) less important
Fast (Parallel) Builds

Trade off BVH quality for build performance

Spatial sort using Morton ordering

Build lower-level BVHs in parallel

Build (small) top-level BVH
Two-level Acceleration Structures

Top-level acceleration structure

Bottom-level acceleration structures
Refit BVH When Objects Move

[Kopta et al. 2012]
White Room, 4096 pixel samples
White Room, 1 pixel sample
White Room, 2 pixel samples
White Room, 4 pixel samples
White Room, 8 pixel samples
White Room, 16 pixel samples
White Room, 32 pixel samples
White Room, 64 pixel samples
White Room, 128 pixel samples
White Room, 256 pixel samples
White Room, 512 pixel samples
White Room, 1024 pixel samples
Gaussian Filter

\[ f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \]

Spatial Domain  Frequency Domain
64 pixel samples
19x19 Gaussian Blur
White Room, 4096 pixel samples
Separate Illumination & Reflection

Hemispherical directional reflectance:

\[ \rho_{hd}(\omega_o) = \int_{H^2} f_r(\omega_i \rightarrow \omega_o) \cos \theta_i \, d\omega_i \]

Recall the reflection equation:

\[ L_o(\omega_o) = \int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_i(\omega_i) \cos \theta_i \, d\omega_i \]

Ratio approximates incident radiance:

\[ \frac{L_o(\omega_o)}{\rho_{hd}(\omega_o)} = \frac{\int_{H^2} f_r(\omega_i \rightarrow \omega_o) L_i(\omega_i) \cos \theta_i \, d\omega_i}{\int_{H^2} f_r(\omega_i \rightarrow \omega_o) \cos \theta_i \, d\omega_i} \approx \text{avg } L_i \]
Hemi-Directional Reflectance (Albedo)
White Room, 64 pixel samples
General Pipeline

Noisy image

Noisy Illumination

Denoising

Denoised Illumination

Denoised image

Albedo

Albedo
Gaussian Blurred Illumination
Blurred Illumination * Albedo
White Room, 4096 pixel samples
Ideal Denoising Filters (via Brute Force)
Better Filter: Bilateral

Pixel intensity kernel

\[ G_p(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{d((0,0),(x,y))^2}{2\sigma^2}} \]

\[ d(p, p') = |I(p) - I(p')| \]

Spatial kernel

\[ G_s(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}} \]

[Durand and Dorsey 2002]
Better Filter: Bilateral

[Durand and Dorsey 2002]
Even Better Filter: Joint Bilateral

Include additional, non-visible features in the filter

- Pixel depth
- Surface normal
- BRDF features—roughness, ...
- Object id
Surface Normal
Camera Space “z”
Approximating Local Planar Surface

Given camera space $z$ at a pixel, can approximate the local planar surface as:

$$z(\Delta x, \Delta y) \approx z + \Delta x \frac{\partial z}{\partial x} + \Delta y \frac{\partial z}{\partial y}$$

Where $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ are partial derivatives of $z$ in terms of pixel coordinates $(x, y)$.

Given a depth $z'$ at a nearby pixel, can compute distance from planar approximation,

$$z' - z(\Delta x, \Delta y)$$
\[\frac{dz}{dx}\]
\( \frac{dz}{dy} \)
Joint Bilateral Filter Function

\[ G = G_s \cdot G_p \cdot G_n \cdot G_z \]

**Spatial:** \[ G_s(x, y) = \frac{1}{2\pi \sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}} \]

**Intensity:** \[ G_p(x, y) = \frac{1}{2\pi \sigma^2} e^{-\frac{d((0,0),(x,y))^2}{2\sigma^2}} \]

**Normal difference:** \[ G_n(x, y) = \max(\tilde{n}(x, y) \cdot \tilde{n}(0, 0), 0)^n \]

**Depth difference:** \[ G_z(x, y) = \frac{1}{2\pi \sigma^2} e^{-\frac{(\hat{z}(x, y) - z(0,0))^2}{2\sigma^2}} \]

\[ \text{with } \hat{z} = z + \Delta x \frac{\partial z}{\partial x} + \Delta y \frac{\partial z}{\partial y} \]
About all those sigmas...

Pixel intensity contribution

$G_p(x, y) = \frac{1}{2\pi\sigma^2}e^{-\frac{d((0,0),(x,y))^2}{2\sigma^2}}$

should be based on pixel's variance

**Uniform variance:**

$$d(p, p')^2 = \frac{\max(0, (I(p) - I(p'))^2 - 2\sigma^2)}{\epsilon + k^2 2\sigma}$$

**Non-uniform variance:**

$$d(p, p')^2 = \frac{\max(0, (I(p) - I(p'))^2 - (\text{Var}[p] + \min(\text{Var}[p], \text{Var}[p']))}{\epsilon + k^2(\text{Var}[p] + \text{Var}[p'])}$$

**Sample variance:**

$$\text{Var}[p] = \left(\frac{1}{n - 1} \sum_{i=1}^{n} (x_i - \bar{x})\right)/n$$
Indirect Illumination Sample Variance
Filtered Direct Variance
Filtered Indirect Variance
Revised Pipeline

Noisy direct

Denoising

Denoised direct

Albedo

Denoised indirect

Noisy indirect

Denoising

+
Multi-Level Filtering: À-Trous

Want wide filter kernels to eliminate noise

- Recall Gaussian 19x19 was still blotchy

Wide kernels are expensive...

Multi-scale filtering:

[Dammertz et al. 2010]
Direct Lighting / Albedo
One Iteration
Two Iterations
Three Iterations
Filtered Direct Lighting
Indirect Lighting (64 samples)
Indirect Lighting / Albedo
Filtered Indirect Lighting / Albedo
Filtered Indirect Lighting
64 denoised samples, $MSE = 2.16$
64 samples per pixel, MSE = 3.44
64 denoised samples, MSE = 2.16
128 samples, MSE = 1.28
64 denoised samples, MSE = 2.16
Reference (4096 samples)
Close-ups

64 samples

4096 samples

64 samples, denoised
Temporal Filtering

Spatiotemporal Variance-Guided Filtering [Schied et al. 2017]
(Video)
Summary

Estimate variance, then blur it
■ Using which filter? Blur how much?

Choose additional features for joint filter
■ Which ones? Measure distance how?
  How much weight does each one get?

Blur illumination using joint filter
■ Which filter? Blur how much?

Multiply by albedo again
Learning Filter Parameters

3-layer fully connected MLP
Computes per-pixel filter parameters

[Kalantari et al. 2015]
Deep Denoising (NVIDIA)

Features: illumination, normals, depth, roughness

3.2M trainable parameters

Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder
Results

(a) 1spp noisy input  
(d) Recurrent autoencoder  
(e) Reference
Denoising (Pixar/Disney)

Deep CNN, 8 layers, 100 5x5 kernels

Features: illumination, normals, depth, and their variances

Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings
Kernel Prediction

Network generates a stencil of 21x21 filter weights at each pixel

- Normalized to sum to one (~softmax)
- Weights are then applied to the noisy image

Advantages:

- Result always in convex hull of input
- Better scale invariance (HDR)
- 5-6x faster convergence than direct reconstruction
Results

\begin{itemize}
    \item Ours
    \item Input (32 spp)
    \item Ours
    \item Ref. (1K-4K spp)
\end{itemize}

relative $\ell_2$
\begin{align*}
1 - \text{SSIM} & \\
19.21 \times 10^{-3} & \\
1.16 \times 10^{-3}
\end{align*}

Noisy (32 spp)

TRANING

Noisy (32 spp)

TEST

Reference (1024 spp)

Denoised (32 spp)
64 samples per pixel, MSE = 3.44
64 CNN-denoised samples, MSE = 2.38
Reference (4096 samples)
Zoom-ins: CNN denoising

64 samples

4096 samples

64 samples, denoised
Noise2Noise: Training w/noisy images

Insight: neural networks can be trained using independent noisy images, without having high-quality reference images in the training set.

[Lehtinen et al. 2018]