Antialiasing

- Basic raytracing shoots one ray through the center of each pixel, so is subject to aliasing
  – Staircase edges, beating, etc
- Simple solution: Cast multiple rays per pixel
  – Sub-pixel sampling, can be adaptive
  – Sub-samples can be regular, or random
- Random sub-sampling replaces aliasing with noise

Distributed Ray Tracing

- Cook, Porter, Carpenter 1984
- Addresses the inability of ray tracing to capture:
  – non-ideal reflection/transmission
  – soft shadows
  – motion blur
  – depth of field
- Basic idea: Don't just vary the direction of each initial ray, also vary time, reflection, etc
- Rays are distributed, not the algorithm. Should probably be called distribution ray tracing.

Specific Cases

- Sample several directions around the reflected direction to get non-ideal reflection, and specularities
- Sample over the area of extended light sources to get soft shadows
- Sample over time to get motion blur
- Sample lens and aperture paths to get depth-of-field
- Must take care not to correlate choices

What’s Good? What’s Bad?

- Easy to implement - standard ray tracer plus simple sampling
- Which $L(D|S)^*E$ paths does it get?
- Which previous method could it be combined with to good effect?

Path Tracing

- Path tracing algorithms determine the intensity of each pixel by tracing light transport paths
  – Paths that start at light sources and carry energy
- A path of length *k* is a sequence of vertices, $<x_0,\ldots,x_{k-1}>$ where every x_i and x_{i+1} is mutually visible, and x_0 is on a light
- Clearly, we are most interested in “important” paths

“Important” Paths

- Consider only paths that go from a light source to the eye
  – Other useful paths are sub-paths of these
  – Paths that miss the image plane contribute nothing, so are not important
- Paths that carry more energy are more important
- Why is that?

Return to the Rendering Equation

$$L(\boldsymbol{x}' \to \boldsymbol{x}'') = L_e(\boldsymbol{x}' \to \boldsymbol{x}'')$$
$$+ \int_{\mathrm{M}} L(\boldsymbol{x} \to \boldsymbol{x}') f_s(\boldsymbol{x} \to \boldsymbol{x}' \to \boldsymbol{x}'') G(\boldsymbol{x} \leftrightarrow \boldsymbol{x}') dA(\boldsymbol{x})$$

- Can express value at each pixel as a sum of integrals, each one integrating over a different path length (Veach 97)

$$m_j = \int_{M^2} L_e(\boldsymbol{x}_0 \to \boldsymbol{x}_1) G(\boldsymbol{x}_0 \leftrightarrow \boldsymbol{x}_1) W_e^{(j)}(\boldsymbol{x}_0 \to \boldsymbol{x}_1) dA(\boldsymbol{x}_0) dA(\boldsymbol{x}_1)$$
$$+ \int_{M^3} \begin{array}{l} L_e(\boldsymbol{x}_0 \to \boldsymbol{x}_1) G(\boldsymbol{x}_0 \leftrightarrow \boldsymbol{x}_1) f_s(\boldsymbol{x}_0 \leftrightarrow \boldsymbol{x}_1 \leftrightarrow \boldsymbol{x}_2) \\ G(\boldsymbol{x}_1 \leftrightarrow \boldsymbol{x}_2) W_e^{(j)}(\boldsymbol{x}_1 \to \boldsymbol{x}_2) dA(\boldsymbol{x}_0) dA(\boldsymbol{x}_1) dA(\boldsymbol{x}_2) \end{array}$$
$$+ \ldots$$

Sampling Important Paths

- We can evaluate that integral using importance sampling
  - Sample paths of various lengths
  - Weight their contribution to pixel intensity by their importance
- We wish to sample important paths
  - those for which integration kernel is large
- The big question: How are those paths found?

Naïve Path Tracing (version 1)

- Start at light
- Build a path by randomly choosing a direction at each bounce, and adding point hit by ray in that direction
- Join last point to eye
- What is the basic problem? What paths does it get?

Naïve Path Tracing (version 2)

- Start at eye
- Build a path by randomly choosing a direction at each bounce, and adding point hit by ray in that direction
- (optional) Join last point to light
- What is the basic problem? What paths does it get?

Path Tracing (Kajiya): Description

- Start at eye
- Build a path by, at each bounce, sampling a direction according to some distribution
- At each point on the path, cast a shadow ray and add direct lighting contribution at that point
- Multiple paths per pixel, average contributions to get intensity

Path Tracing (Kajiya): Sampling Strategies

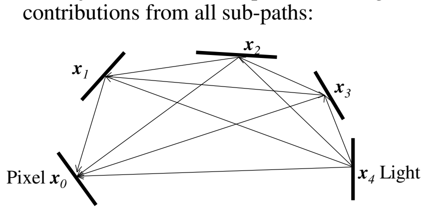
- The method of choosing the direction at each bounce is important
- Can choose using:
  - *stratified sampling*
    - Break possible directions into sub-regions, and cast one sample per sub-region
    - Various ways to be adaptive
  - Importance sampling
    - Sample according to BRDF

Path Tracing (Kajiya): Analysis

- Doesn’t waste time on things that aren’t visible
- Unlike ray tracing, spends equal time on all path lengths (ray tracing spends more time on longer paths)
- Downsides:
  - Little information gain for each ray cast
  - Not easy to get good (important) samples
  - Spends equal time on slow-varying diffuse components and fast varying specular components

Pure Bi-Directional:Approach

- Veach 94; Lafortune and Willems 94(?)
- Build a path by working from the eye and the light and join in the middle
- Don’t just look at overall path, also weigh contributions from all sub-paths:



Pure Bi-Directional: Analysis

- Advantages:
  - Each ray cast contributes to many paths
  - Building from both ends can catch difficult cases
    - All specular paths
    - Caustics
  - Extends to participating media (anisotropic, heterogeneous)
- Disadvantages:
  - Still using lots of effort to catch slow varying diffuse components
  - May not sample difficult to find paths

Combining Estimators

- Veach 95 describes a way to combine the results of various estimators
- Useful when:
  - different methods are suited to different aspects of the scene
  - It is not known a-priori which method will work

Metropolis Light Transport: Approach

- Other algorithms generate independent samples
  - Easy to control bias
- Metropolis algorithms generate a sequence of paths where each path can depend on the previous one
- For each sample:
  - Propose a new candidate depending on the previous sample
  - Choose to accept or reject according to a computed probability (if reject, re-use the old sample)
- Can prove the estimates for pixel intensities are correct

Metropolis Proposal Strategies

- Task: Given the previous sample, generate a new one
  - Should be very different, but should also be good
- Methods:
  - Randomly chop out some part of the path and replace it with a new piece
  - Randomly perturb a vertex on the path
  - Less randomly change the pixel that is affected
  - Other choices possible

Metropolis: Analysis

- Easy to implement basic algorithm
  - Some of the details for good results are difficult
  - Easy to parallelize
- Can do difficult scenarios:
  - Light through a crack, almost impossible any other way
  - Caustics from light reflecting off the bottom of a wavy pool
- But, still computes diffuse illumination on a per point basis

Still to come…

- Computing the diffuse component efficiently
- Various algorithms:
  - Radiosity textures
  - Radiance
  - Photon-maps