CS 563 Advanced Topics in Computer Graphics
Irradiance Caching and Particle Tracing

by Stephen Kazmierczak

Introduction

- Unbiased light transport algorithms can sometimes take a large number of rays to generate an image without noise
- To counter this, biased algorithms such as irradiance caching and photon mapping have been developed
  - Reuse previously computed results
  - Produce images without high-frequency noise artifacts
  - Produce good images using less additional computing power
  - Hard to do error estimation, however

Irradiance Caching

- Based on the notion that while direct lighting changes drastically from point to point, indirect lighting does not
- Compute indirect lighting at a sparse set of sample points and interpolate the rest

Indirect Illumination

IC Issues

1) When are new representations of indirect light computed, and how often are already existing ones interpolated?
2) How is the indirect lighting distribution represented and stored after being computed at a point?

Computing Points

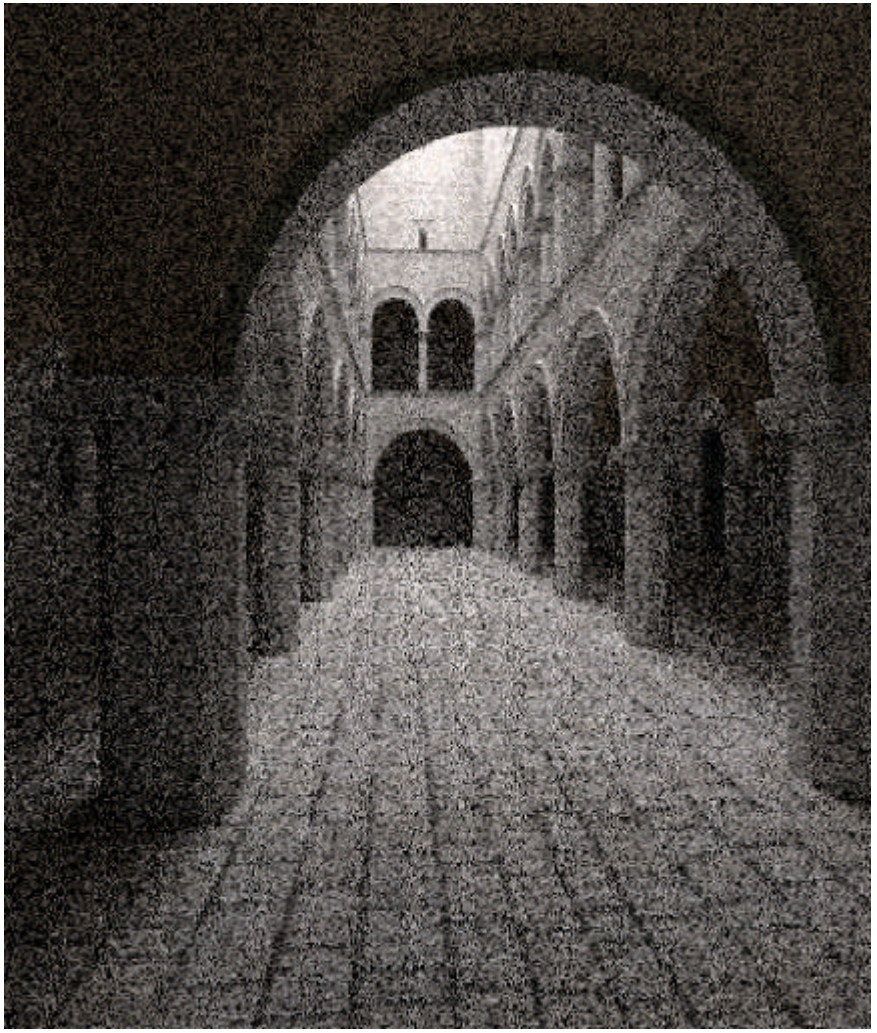
- Indirect lighting computed on-the-fly (as opposed to on a fixed set of points chosen ahead of time)
- First cache is searched for an acceptable point using a set of error metrics
- If a point cannot be found in the cache, a new point is created

Storing Information

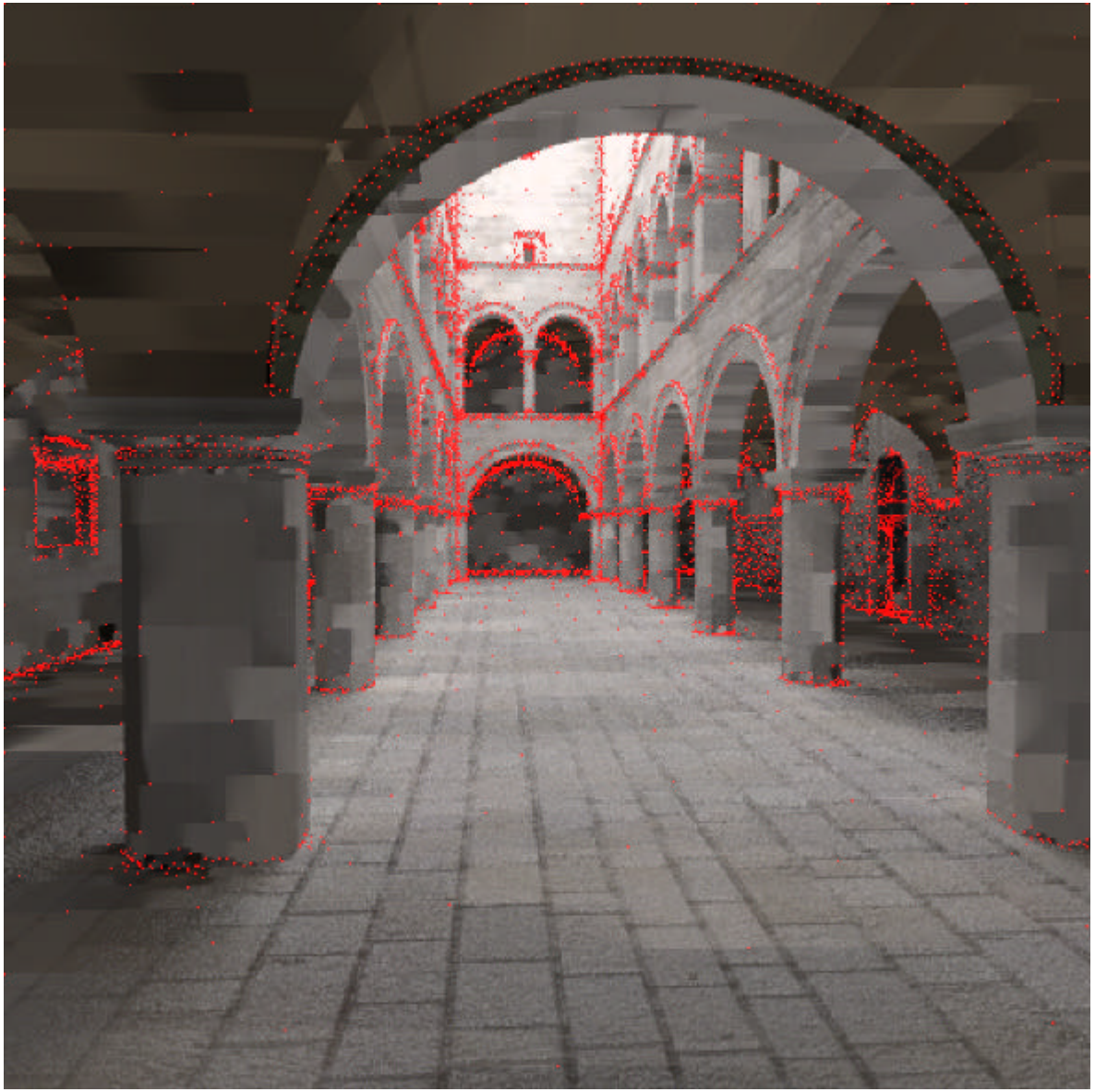
- Only irradiance is stored at each point (light representation is a single Spectrum object)
- As long as the surface is of a perfectly diffuse material, irradiance alone can exactly compute from the surface due to a particular incident lighting distribution

- A perfectly specular surface, however, can potentially introduce an arbitrarily large error

Caching vs. Path Tracing



Points for IC



Implementation Details

- Partitions BSDF for indirect lighting computation
  - Perfect specular reflection handled by sampling BSDF and recursively calling the integrator
- Implementation uses irradiance caching for both diffuse and glossy components
  - Glossy components introduce additional error
- Reflection and transmission handled separately
  - Reflective surface hemisphere completely different than transmissive surface hemisphere

New Irradiance Values

- Uses a cosine-weighted distribution of directions
- Uses the standard Monte Carlo estimator

Storing Values

- Uses an octree data structure to store computed estimates
- Each estimate has an axis-aligned bounding box associated with it, giving the overall area for which the sample is valid
- Bounding box clamped to make sure it is neither too large nor too small
- Boxes are smaller around groups of objects, as the more objects that are close to a sample point the greater potential for rapidly changing irradiance
- Estimate stored in IrradianceSample object

Computing Irradiance

- Traverse the octree to find appropriate box
- Using an irradiance sample from the tree and the point to be shaded, test
  - Reject if surface normals too different
  - Reject if too far from point
  - Reject if sample in front of point
  - Compute error term
- If everything passes, weight the sample based on error term and use
- Final interpolated irradiance

$$E = (?_i\ w_i E_i) / (?_i\ w_i)$$

Photon Mapping

- Photon Mapping is one of a family of *particle-tracing* algorithms
  - Construct paths from lights, and at each vertex the amount of incident illumination recorded
- Unlike irradiance caching, photon mapping handles both glossy and diffuse reflection well
- Can also handle perfectly specular reflection (handled separately with recursive ray tracing)

Photon Integrator

- Photon mapping integrator traces particles into the scene
- It interpolates among particles (called photons) to approximate the incident illumination at shading points
- Integrator uses a kd-tree to store photons, which allows quick access to the photons around the point being shaded

Photon Integrator (2)

- Adjusting the quality of results computed is easy with the photon mapper, as it partitions the LTE in a number of ways
  - Particles coming from lights are characterized: direct illumination, caustic illumination, and indirect illumination
- Partitioning allows flexibility in how reflected radiance is estimated
- Integrator also partitions the BSDF
  - Uses recursive ray tracing to handle perfectly specular components, and either photon maps or Monte Carlo ray tracing for the rest

Photon Integrator Config

- Integrator is highly configurable
  - Desired number of photons of each type (direct, caustic, and indirect)
  - Number of photons used for interpolation
  - Whether to do “Final Gathering” or not

Building the Photon Maps

- Particles begin at light sources and are traced through the scene until the integrator has accumulated the desired number of particle histories
- At each intersection of the path with an object a weighted particle contribution (of type *Photon*) is stored in the appropriate map
- A Halton sequence is used to generate particle rays, as they need to be well distributed but it is unknown ahead of time how many are needed

Following the Photon

- While the photon intersects with objects in the scene
  - Handle photon/surface intersection
  - Update photon weight and photon ray direction
  - Possibly terminate photon path
    - Russian roulette

Using the Photon Map

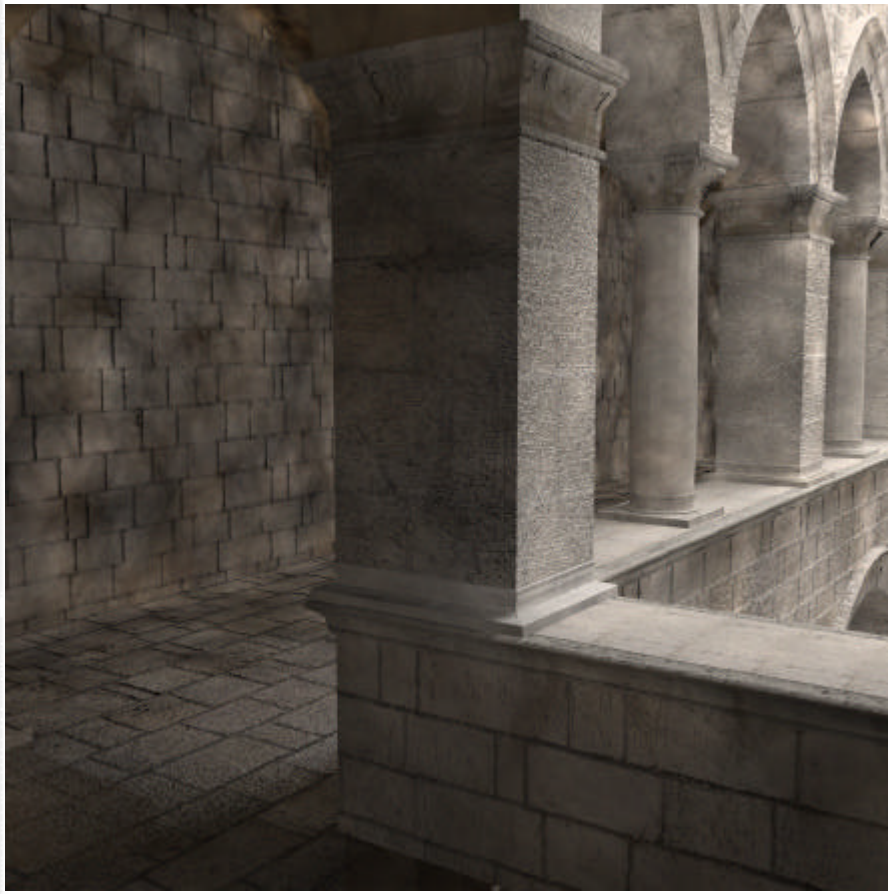
- At rendering time, the photon map is used to compute reflected light at each point being shaded
- Photon mapping interpolates information about illumination at the point from nearby photons
  - The more photons around the point and the higher their weights the more radiance is estimated to be incident at the point
  - Estimated radiance used in conjunction with surface BSDF to compute reflected light

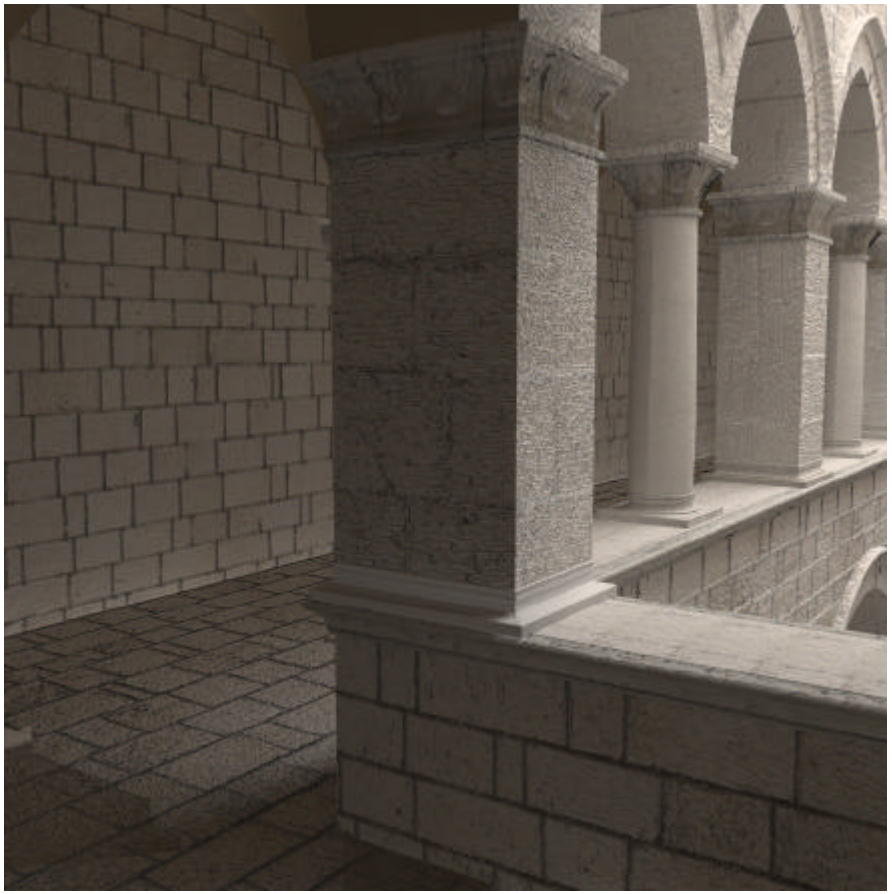
Photon Map in Action

Final Gathering

- Calculates the irradiance at selected sample points in the scene by shooting many rays and gathering light from other surfaces
- Very expensive for more than one bounce
- Can be combined with photon mapping:
  - Use photon mapping for N – 1 bounces
  - Use final gathering for the final bounce

Final Gathering Compare





Photon Interpolation and Density Estimation

- Search the photon list for photons near the point to be shaded, and keep track of photons close to the point
  - Photons stored in kd-tree
- Need both the local density of particles and their individual weights
  - Use a kernel method to estimate density (will use a constant function)
  - Nearest-neighbor techniques can adaptively choose soothing parameter based on local density

Photon Final

- Substituting everything into the measurement equation, the exitant radiance at point p in direction w is:

$$L_o(p, w_o) = p(p) ? a_j f(p, w_o, w_j)$$

Where the sum is over the *n* nearest photons

References

- Pharr, Humphreys, Physically Based Rendering, Sections 16.4 & 16.5
- Using Global Illumination in Turtle: http://www.illuminatelabs.com/support/tutorial-folder/advanced-global-illumination/