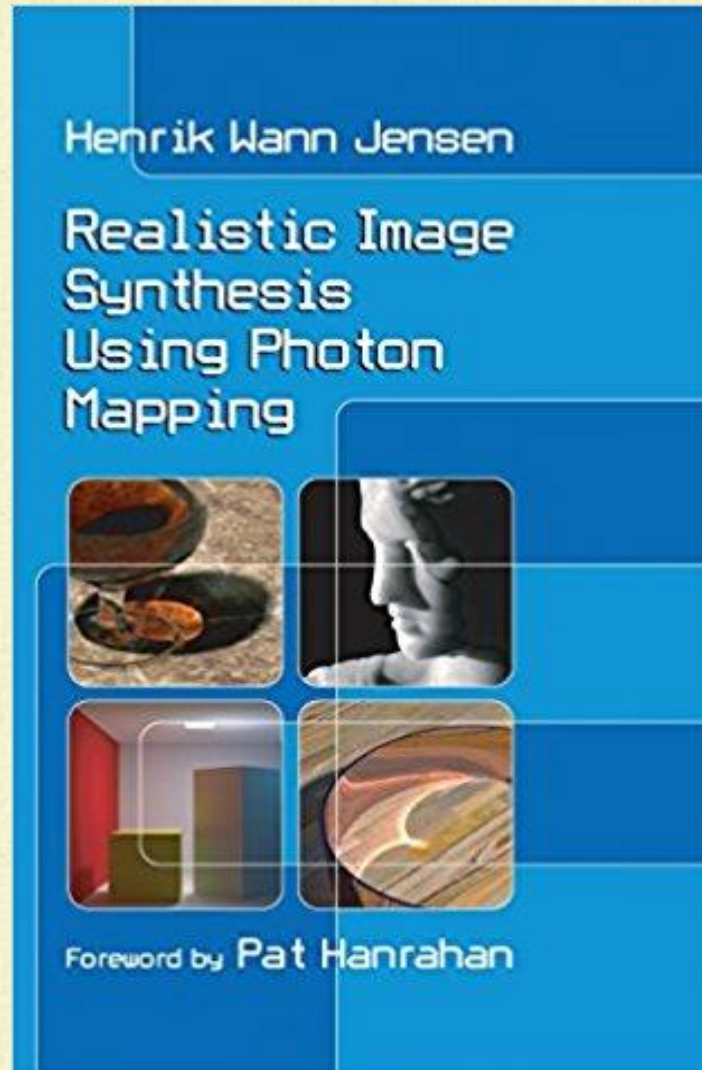
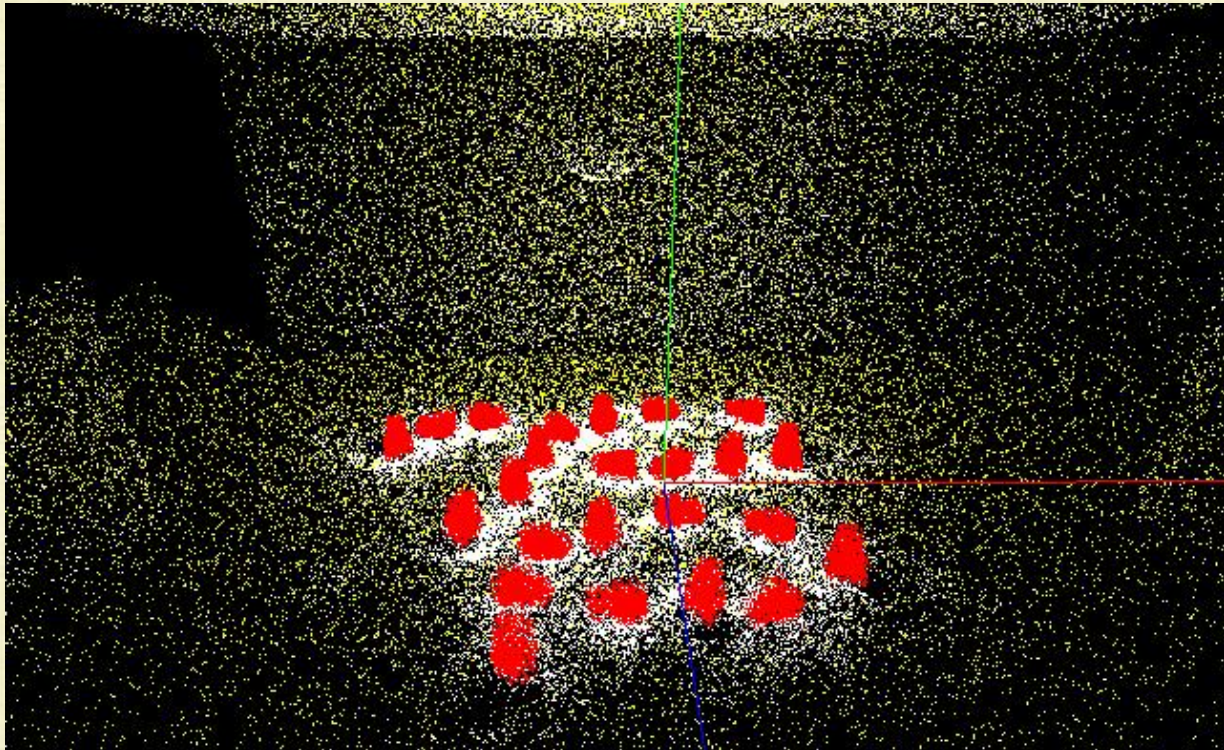


Photon Mapping



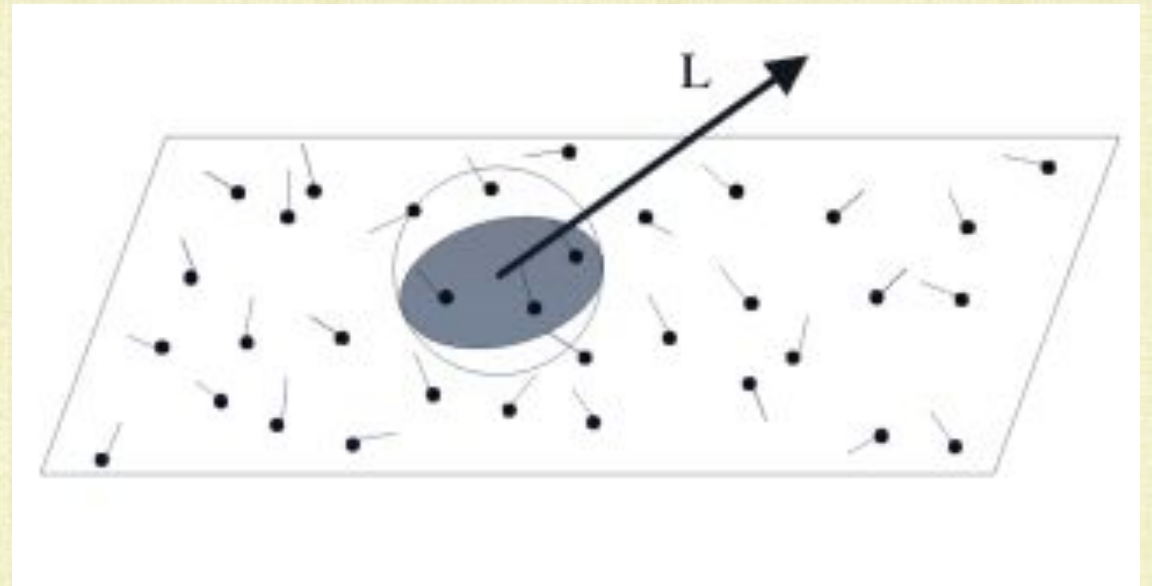
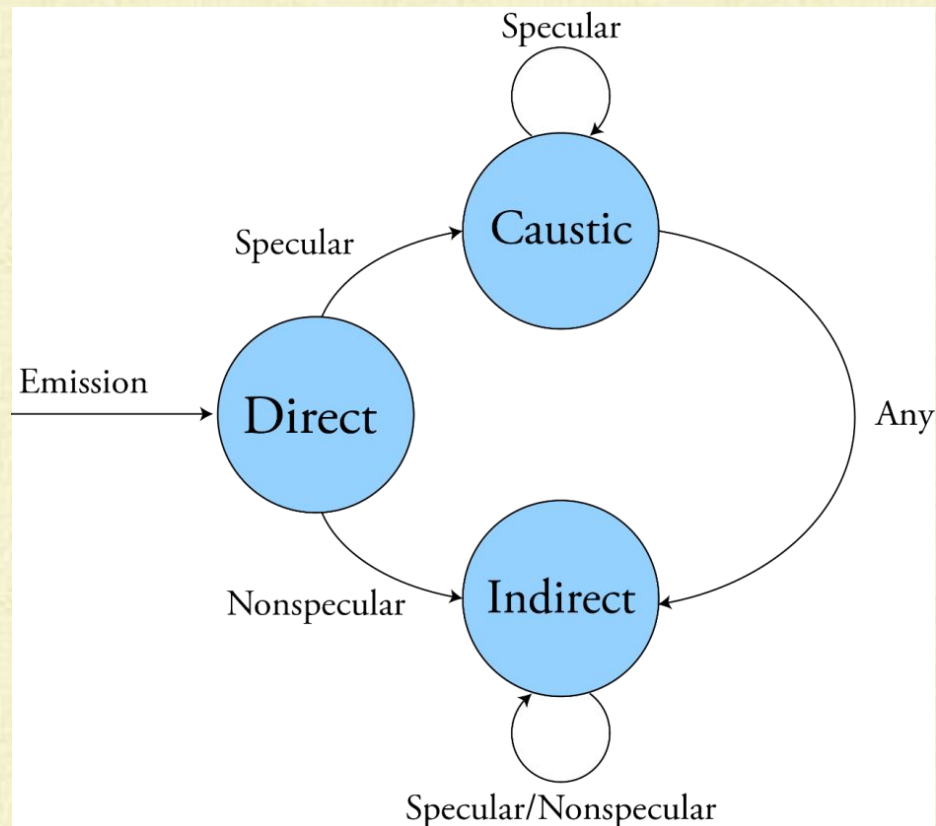
Photon Maps

- Photon maps store lighting information on points or “photons” in 3D space
 - Stored either on **or** near 2D surfaces
- In the last lecture, we (instead) stored information on surfaces patches/triangles



Photon Maps

- Photons are emitted from light sources and bounce around the scene creating lighting information (left image), which is stored in the photon map
- Later (right image), one queries the photon map in order to estimate global illumination

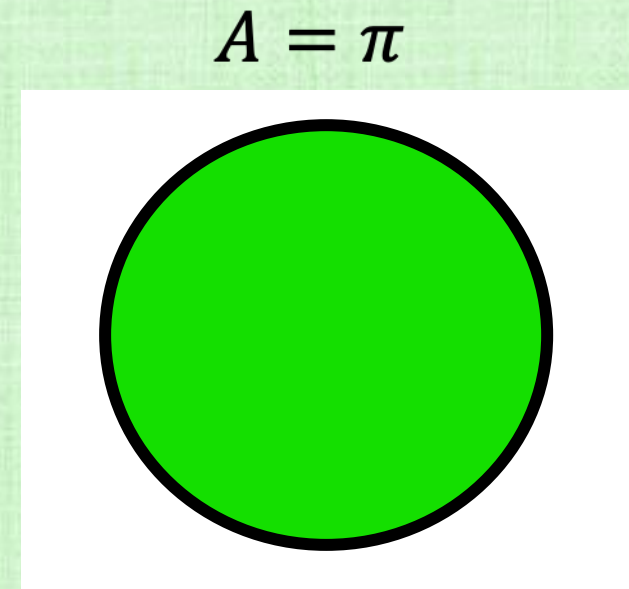


Avoiding Radiosity Simplifications

- In the last lecture, we discretized surfaces into triangles, and discretized the hemisphere as well
 - This discretization into “elements” is a finite element method (FEM) approach to the integral, and similar to Newton-Cotes quadrature
 - 3D space + 2D angles = 5D (or, ignoring participating media, 2D space + 2D angles = 4D)
 - Newton-Cotes quadrature suffers from **the curse of dimensionality**, and **thus we assumed purely diffuse lighting** to reduce the dimensionality (**for tractability**)
 - Integrating over angles (the radiosity approach) reduced the problem to 2D (or 3D for participating media)
 - But then one cannot address the specular term!
-
- Alternatively, **Monte Carlo integration** (which is less accurate than Newton-Cotes quadrature) scales well on higher dimensional problems (**no curse of dimensionality**)
 - Monte Carlo allows one to tackle the full lighting equation in 4D (and 5D), without assuming purely diffuse lighting

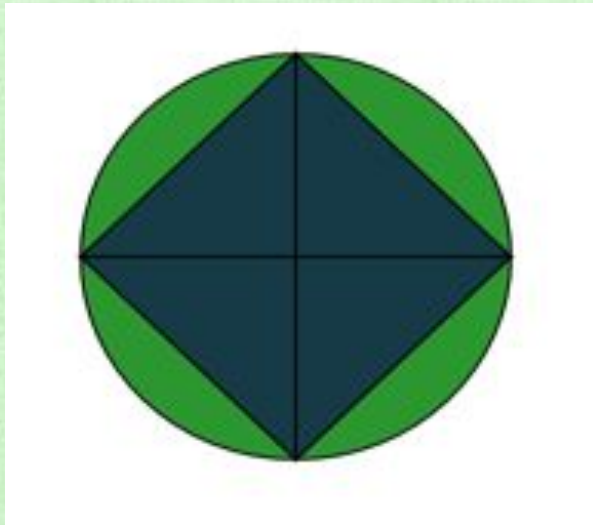
A Simple Example

- Consider approximating $\pi = 3.1415926535 \dots$
- Use a compass to construct a circle with radius = 1
- Since $A = \pi r^2$, the area of the circle is π
- Setting $f(x, y) = 1$ gives $\iint_A f(x, y) dA = \pi$
- So, compute the integral..

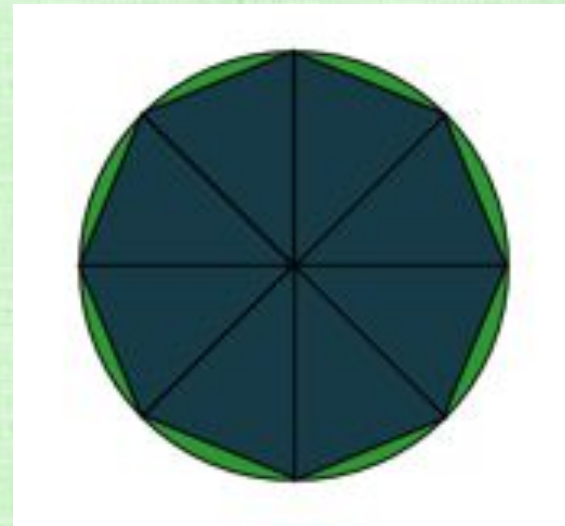


Newton-Cotes Approach

- Inscribe triangles inside the circle
- The function $f(x, y) = 1$ dictates computing the area of each triangle (and trivially multiplying by the height = 1)
- The difference between A and its approximation with triangles leads to errors



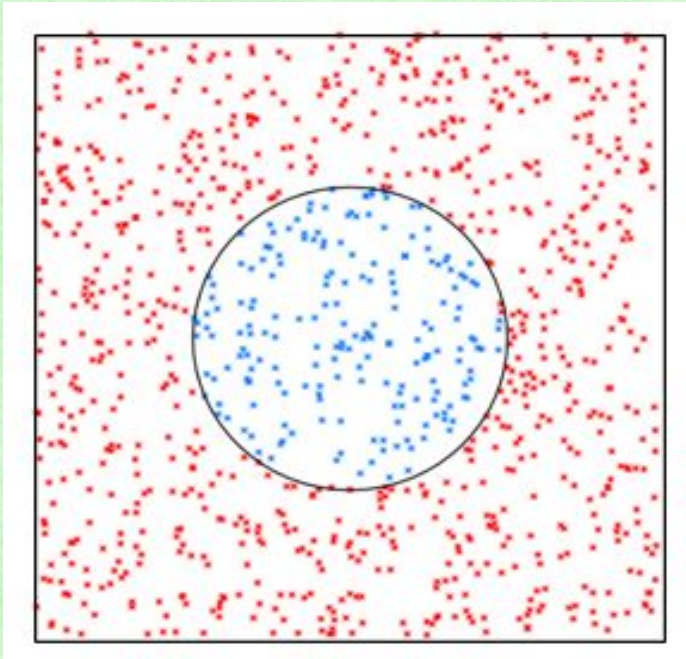
$$\pi \approx 2$$



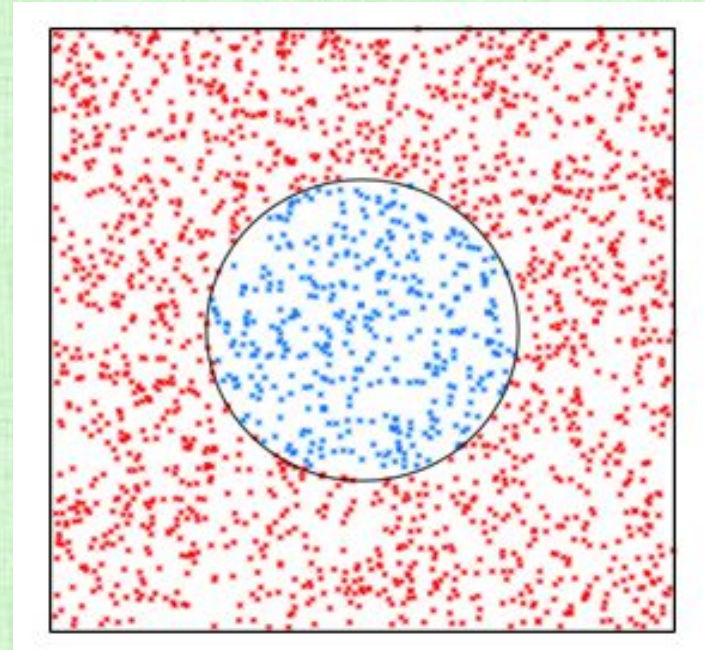
$$\pi \approx 2.8284$$

Monte Carlo Approach

- Construct a square with side length 4 containing the circle
- Randomly generate N points in the square, and color points inside the circle blue
- Since $\frac{A_{circle}}{A_{box}} = \frac{\pi}{16}$, one can approximate $\pi \approx 16 \left(\frac{N_{blue}}{N_{blue} + N_{red}} \right)$



$$\pi \approx 3.136$$



$$\pi \approx 3.1440$$

Monte Carlo Methods

- Typically used in higher dimensions (5D or more)
- Random (pseudo-random) numbers generate sample “points” that are multiplied by element “size” (e.g. length, area, volume, etc.)
- Error decreases like $\frac{1}{\sqrt{N}}$ where N is the number of samples (1/2 order accurate)
 - E.g. 100 times more sample points are needed to gain one more digit of accuracy
- **Very slow convergence, but independent of the number of dimensions!**
- Not competitive for lower dimensional problems (i.e., 1D, 2D, 3D), but the only tractable alternative for higher dimensional problems

Review: Random Numbers

- **Random variables** – expressions whose value is the outcome of a random experiment
- **Sample space** – the set of all possible outcomes
- **Probability distribution** $p(x)$ - probability of selecting each outcome in the sample space
- **Sample** - value of a random variable chosen from the sample space, with probability determined by $p(x)$
- **Pseudo-Random Number Generator** (PRNG) - deterministic algorithm that generate sequences of quasi-“random” numbers based on an initial **seed** (starting point in the pre-determined sequence)
 - PRNGs typically generate a (pseudo) random real number between 0 and 1 with equal (**uniform**) probability
 - Uniformly sampling [0,1] enables the sampling of other sample spaces that have non-uniform probabilities

Monte Carlo Integration (in 1D)

- Consider: $\int_a^b f(x)dx$
- Generate N ^(random) samples X_i uniformly in the interval $[a, b]$
- A Monte Carlo estimate for the integral is then defined as:

$$F_N = \sum_{i=1}^N \frac{b-a}{N} f(X_i) = (b-a) \frac{\sum_{i=1}^N f(X_i)}{N}$$

- This is a simple averaging of all the sample results

Importance Sampling

Trivial (motivating) Case:

- Suppose $f(x)$ is only nonzero in a subset $[a_1, b_1]$ of $[a, b]$, i.e. $\int_a^b f(x)dx = \int_{a_1}^{b_1} f(x)dx$
- Then only samples $X_i \in [a_1, b_1]$ matter, since those outside $[a_1, b_1]$ do not contribute to the integral
- Change $p(x)$ from a uniform distribution over $[a, b]$ to a uniform distribution over $[a_1, b_1]$ in order to be more prudent/efficient with sampling

General Case:

- More generally, the probability distribution $p(x)$ should prefer samples in areas with higher contributions (**importance**) to the integral
- Given a $p(x)$ (with $\int_a^b p(x)dx = 1$), the Monte Carlo estimate is given by:
$$F_N = \frac{1}{N} \sum_{i=1}^N \frac{1}{p(X_i)} f(X_i)$$
- When uniformly sampling with $p(x) = \frac{1}{b-a}$, this reduces to $F_N = \frac{1}{N} \sum_{i=1}^N (b-a) f(X_i)$

Importance Sampling

- Monte Carlo estimates for $\int_0^1 x^2 dx$ with $N = 100$ samples:

		Relative Error

- Caution: importance sampling does not necessarily reduce error (and can give worse results)
- Typically, the more $p(x)$ “resembles” $f(x)$, the lower the error
- Choose $p(x)$ based on physical principles or an approximate solution

Photon Emission

- Choose some number of photons; divide them amongst the lights (based on relative power)
 - For efficiency/implementation, every photon is set to be the same strength
 - Brighter lights (simply) emit more photons instead of higher energy photons
- Emission Position:
 - Point light - all photons are emitted from a single point
 - Area light - randomly select a point to emit each photon from
 - Semi-random: Divide a rectangular light into a uniform 2D grid; emit a set number of photons from each grid cell (choosing the position randomly within each cell)
- Emission Direction:
 - randomly choose a direction on a sphere, hemisphere, subset of the sphere (for a spotlight), etc.
- For some scenes (e.g. consider the sun), many/most photons will miss the scene entirely
 - As an optimization, ignore those photons (never emit them)
 - Only generate/emit photons for the sub-light region that hits your scene
 - Scale down the energy of the light (to be that of the sub-light) when dividing up photons

Photon Storage

- Use the ray tracer to find the first piece of geometry a photon intersects
- Every time a photon intersects a surface, its data is added to the photon map to represent **incoming light**
- Create a copy of the photon to store in the photon map
 - Don't delete the original photon, or move it into the photon map
 - The photon *may* still bounce around a bit more (if it's not being absorbed)
- Store the point of impact (a location in 3D space) along with the incoming direction (ray direction from the ray tracer)
 - No need to record the energy (since all photons have equal energy)

Photon Absorption

- After storing the photon's data in the photon map, randomly determine what happens next
- Objects absorb some incoming light, which is why they have a color
- There is some chance that the photon is absorbed by the surface
 - Absorbing a fraction of the photon's energy would result in unequal energy photons
 - Instead, use the fraction of light energy that would be absorbed to state a probability that the photon is absorbed
- Generate a uniform random number and compare it to the probability of absorption to see if the photon is absorbed or not (Russian Roulette)
- When absorbed, the process stops (for this photon)
- Otherwise, the photon bounces

Photon Bouncing

- A bouncing photon needs a new direction
- Compute a new direction by mapping BRDF directions into probabilities
 - E.g. a purely diffuse BRDF has equal probability for all directions on the hemisphere
- Generate a random number and use it to determine the bounce direction (from the BRDF probability table)
- Then, use the ray tracer to find the first piece of geometry the bounced photon intersects
- The new intersection location and the incoming direction are stored in the photon map
- Then, check for absorption; if not absorbed, bounce again, etc.
- Set a maximum number of bounces before termination
 - Can be set rather high as photons typically have a diminishing chance of avoiding absorption (as the number of bounces increases)

Photon Map



Physically Based Rendering by Pharr and Humphreys

Rendered Image



Physically Based Rendering by Pharr and Humphreys

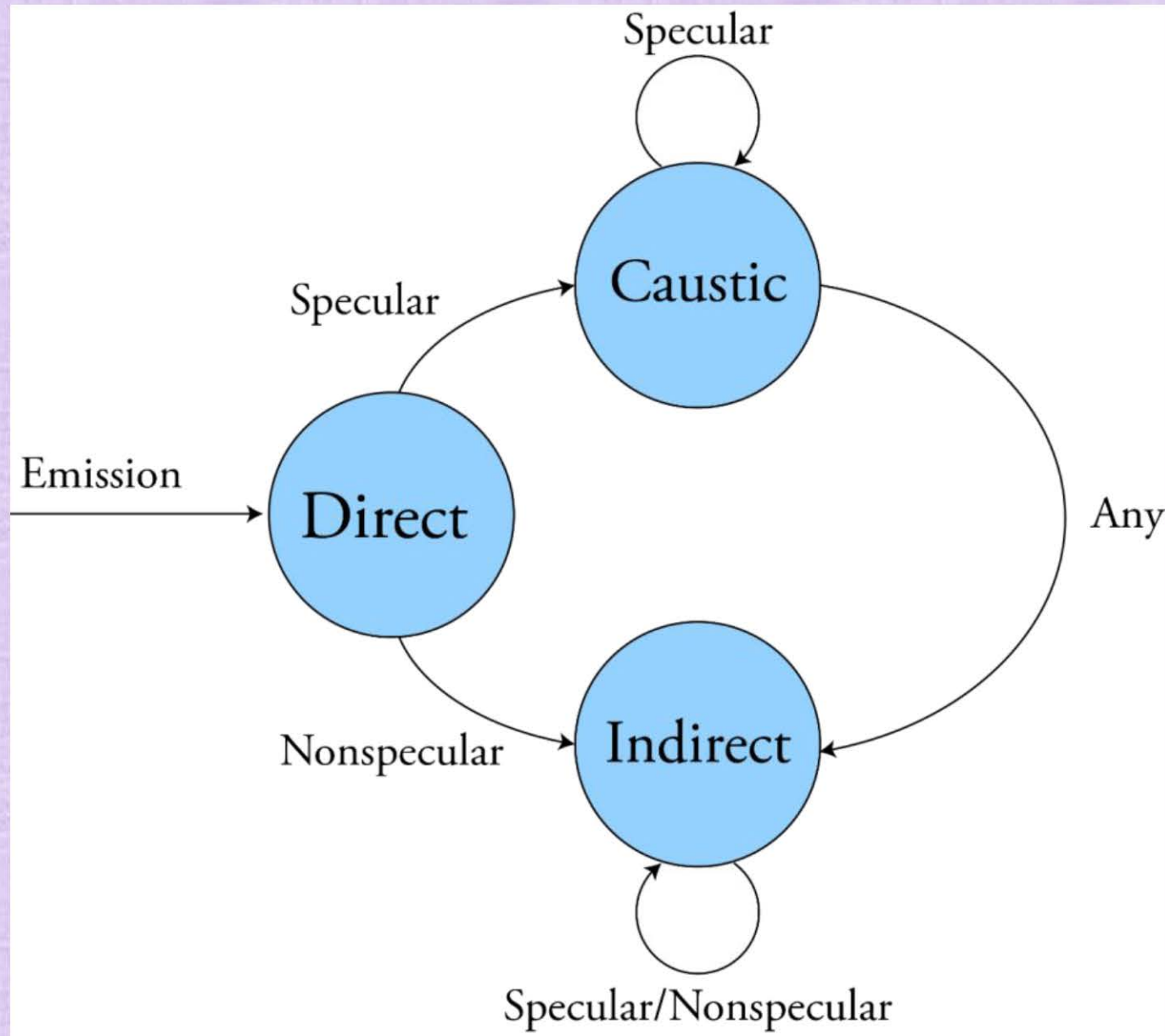
Direct Lighting

- It's more accurate to evaluate direct lighting using shadow rays, rather than interpolating lighting from the photon map
- Thus, the first time a photon emitted from a light source hits an object, it is not stored in the photon map (since this is direct lighting)
- This makes the photon map a lot more efficient, since one doesn't need to store any photons for direct illumination

Separate Diffuse/Specular Photon Maps

- It's more convenient/efficient to treat diffuse and specular lighting separately
- When bouncing a photon, first randomly determine if the photon undergoes:
 - absorption (deleted)
 - (or) a diffuse bounce
 - (or) a specular bounce
- When bouncing, randomly determine the (diffuse or specular) bounce direction
- Create two photon maps:
 - Caustic photon map stores photons that have undergone only specular bounces (up to the point at which they are stored in the map)
 - Indirect lighting map stores any photon that has ever undergone at least one diffuse bounce

Separate Diffuse/Specular Photon Maps

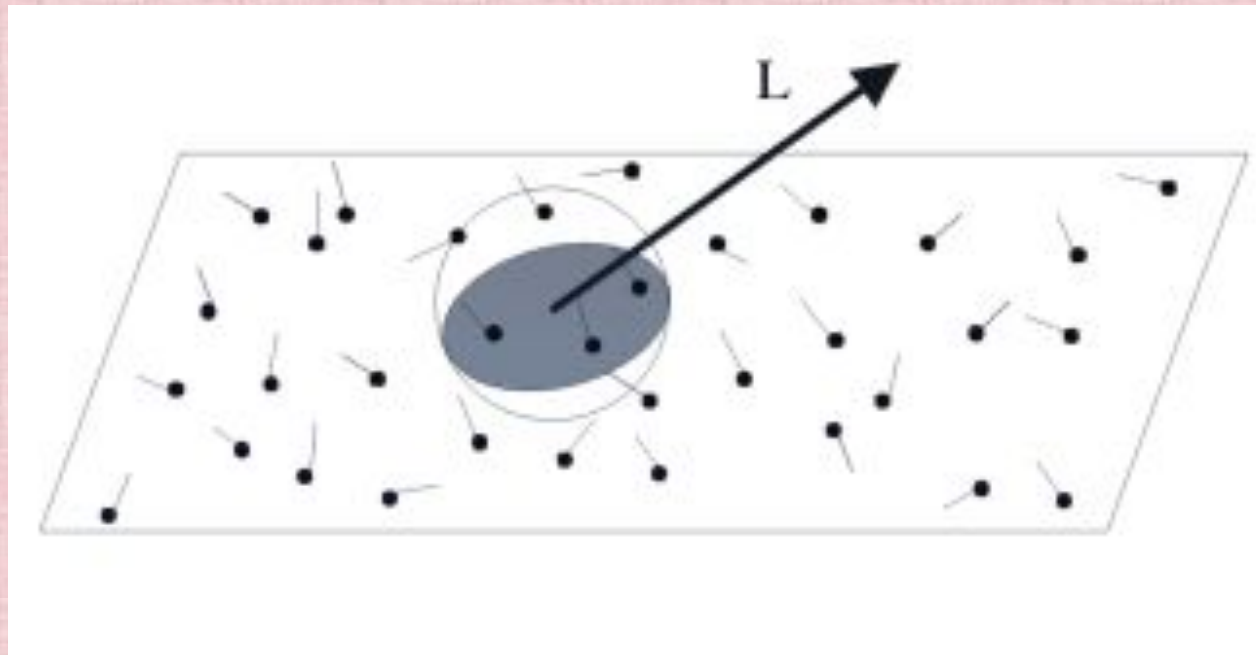


Specular Photon Map for Caustics



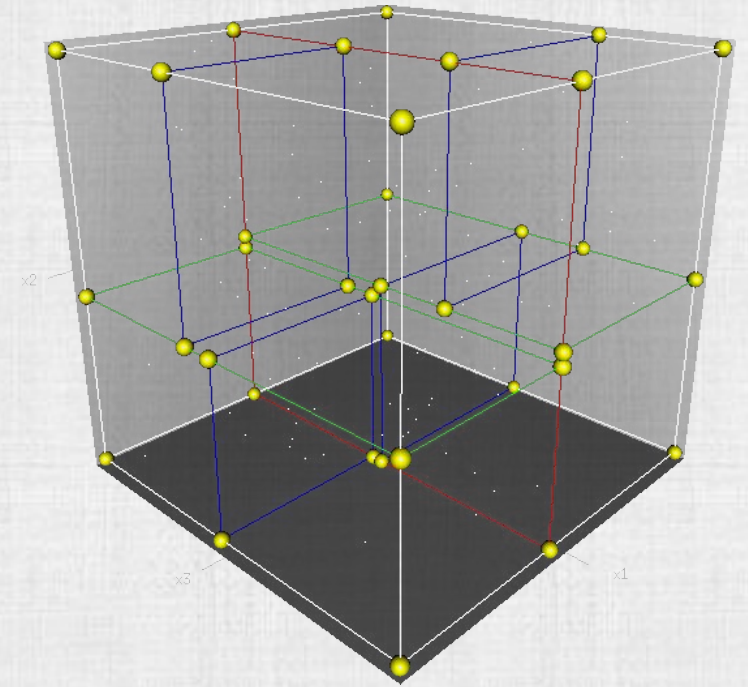
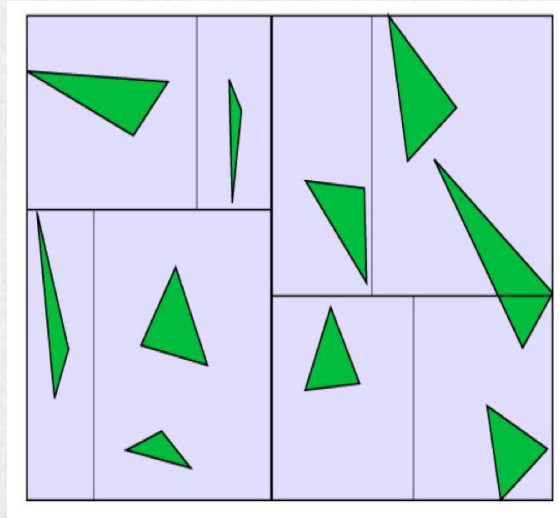
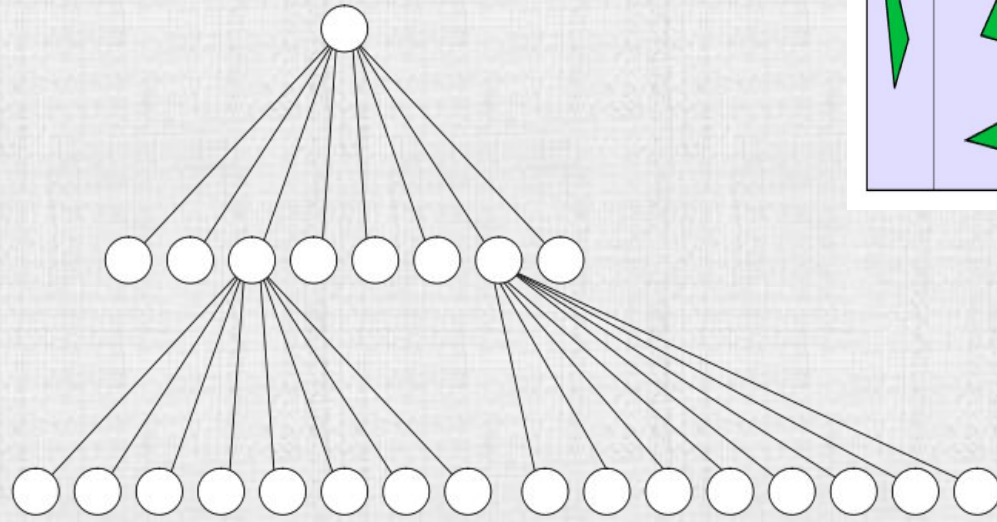
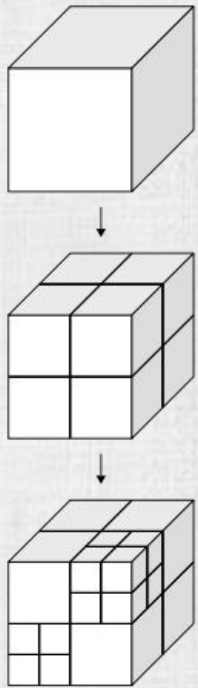
Gathering Radiance

- Trace rays from the camera and intersect with objects (as usual)
- Use shadow rays for direct lighting (as usual)
- Estimate the radiance contribution to the ray from caustics and indirect lighting using the respective photon maps:
 - Use the N closest photons to the point of intersection (with the aid of an acceleration structure to store the photons: an octree or KD tree)



Aside: Code Acceleration

- Photons are typically stored in octree or K-D tree acceleration structures, so that the information they contain is more efficiently retrieved



Color

- Create 3 photon maps, one for each color channel: Red, Green, Blue
- Objects of a certain color better absorb photons of differing colors, creating differences in the photon maps
- This gives color bleeding and related effects

