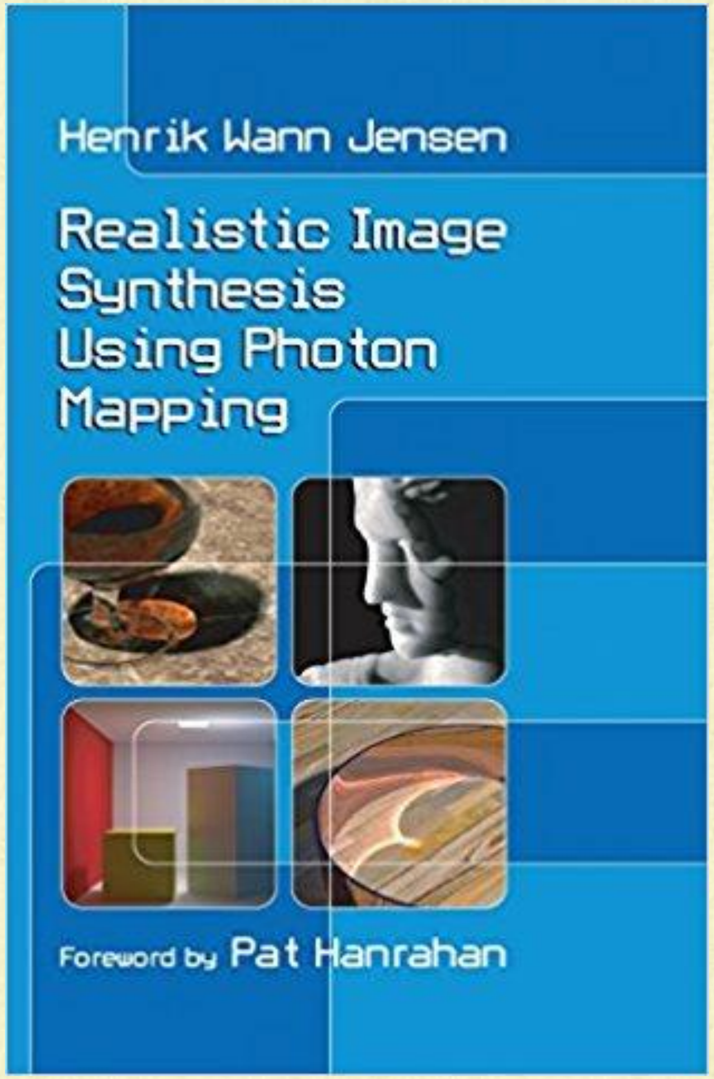
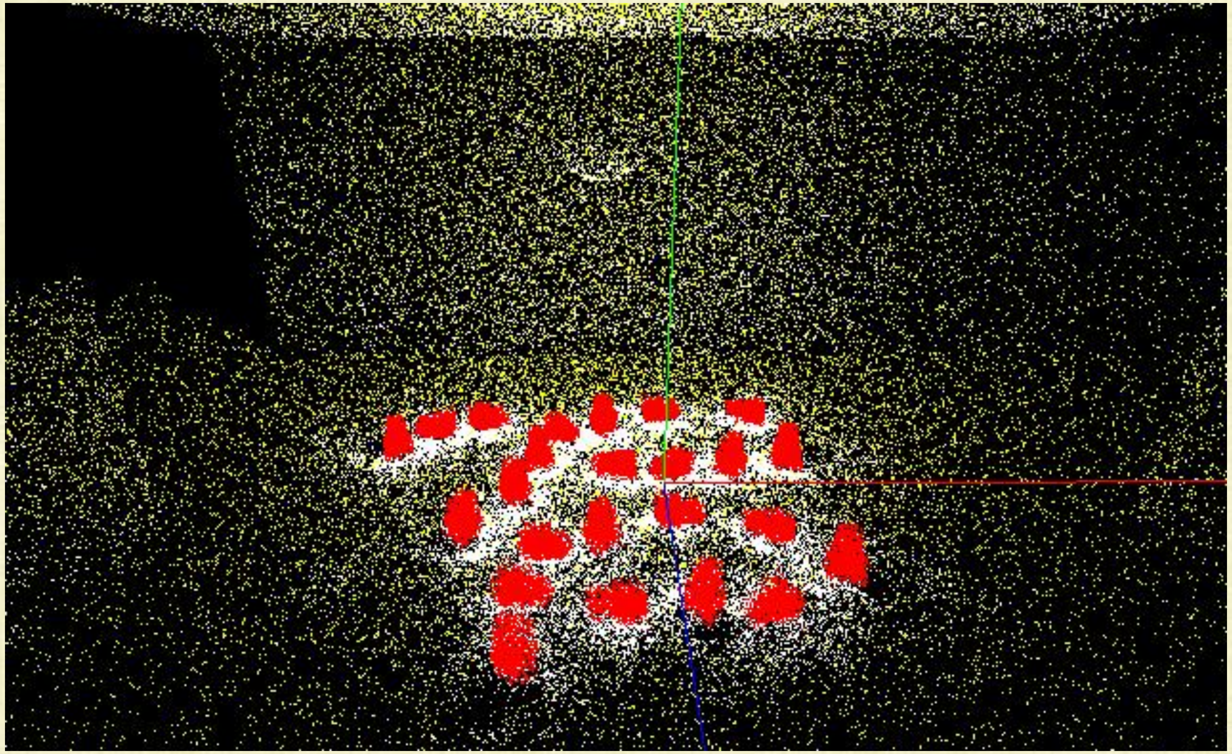
Photon Mapping



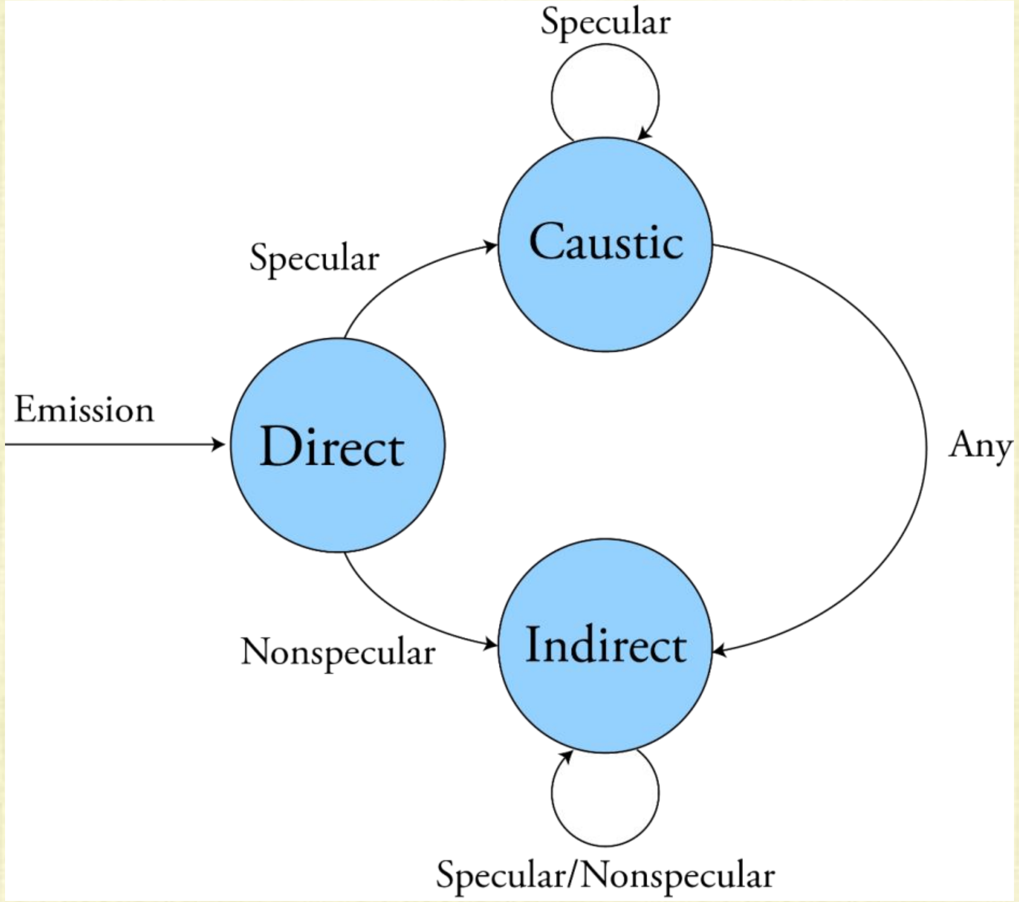
Photon Map (a kind of Light Map)

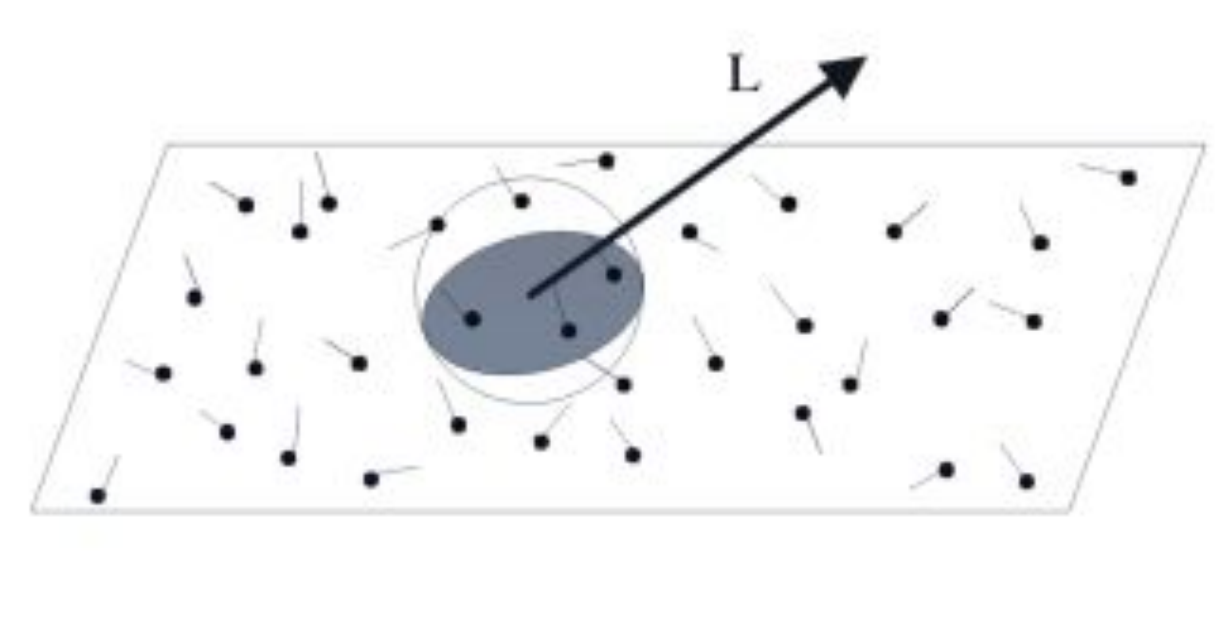
- Photon maps store lighting information on points or “photons” in 3D space
  - Stored either <u>on</u> or <u>near</u> 2D surfaces
- In the last lecture, we (instead) stored information on surfaces patches/triangles



Photon Maps

- Emit photons from light sources and bounce them around the scene, storing light information in the photon map (left image)
- Later (right image), query the photon map in order to estimate global illumination





Avoiding Radiosity

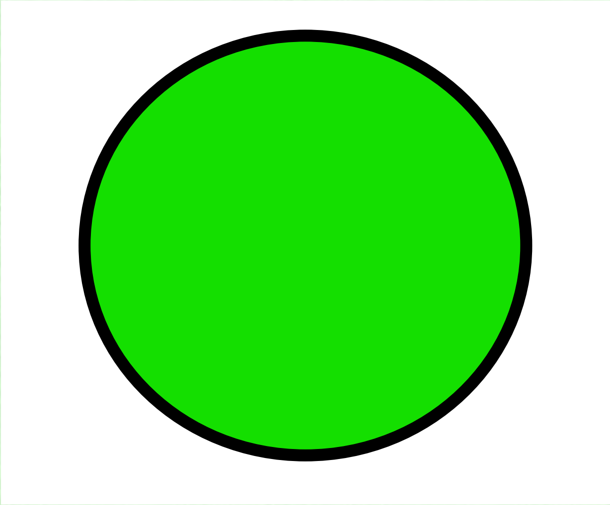
- In the last lecture, we discretized the surfaces and the directions on a hemisphere
- This discretization into "elements" is a Newton-Cotes style approximation to the integral
- 2D space + 2D angles = 4D (or 5D for participating media)
- Since Newton-Cotes approaches suffer from the curse of dimensionality, a diffuse (only) lighting assumption was used to reduce the dimensionality (for tractability)
- Integrating over angles (a radiosity approach) reduces the problem to 2D (or 3D for participating media)
- But specular lighting can no longer be addressed

- Monte Carlo integration (although less accurate than Newton-Cotes) scales well to higher dimensional problems (i.e., no curse of dimensionality)
- Monte Carlo integration can be used on the full 4D (or 5D) lighting equation
- The diffuse (only) lighting assumption is no longer required

A Simple Example

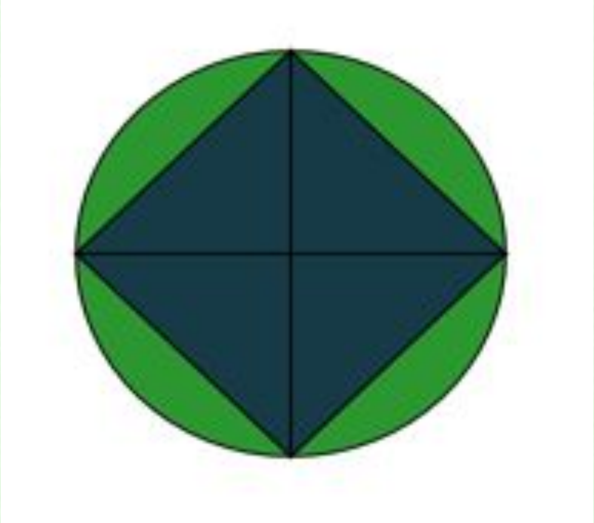
- Consider approximating $\pi = 3.1415926535\ldots$
- Use a compass to construct a circle with radius = 1
- Since $A = \pi r^2$, the area of the circle is π
- Setting $f(x, y) = 1$ gives $\iint_A f(x, y)\,dA = \pi$
- So, compute the integral...

$A = \pi$

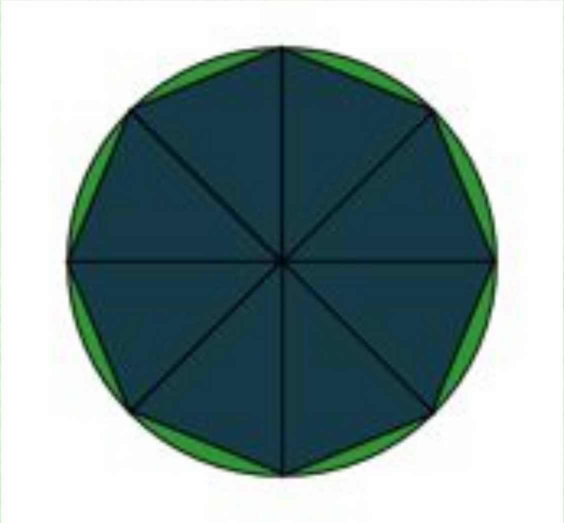


Newton-Cotes Approach

- Inscribe triangles inside the circle
- Sum of the area of all the triangles (no need to trivially multiply by the height = 1)
- The difference between A and its approximation with triangles leads to errors



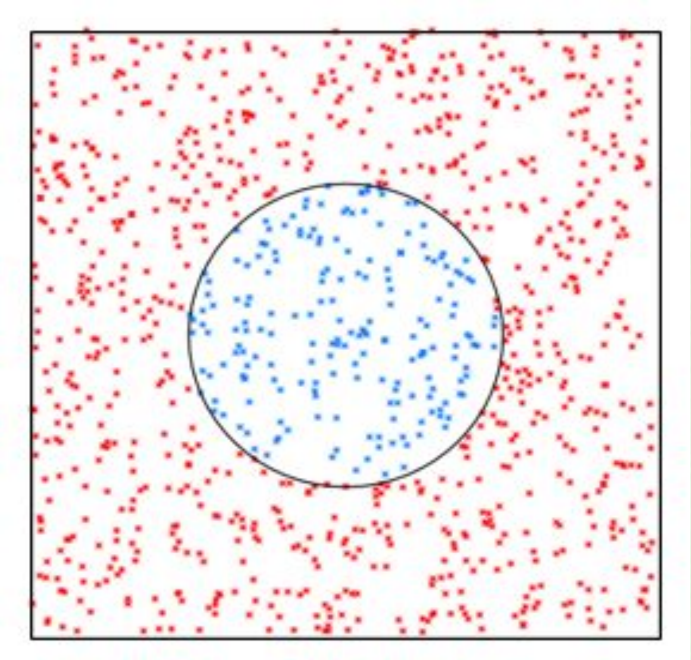
$\pi \approx 2$



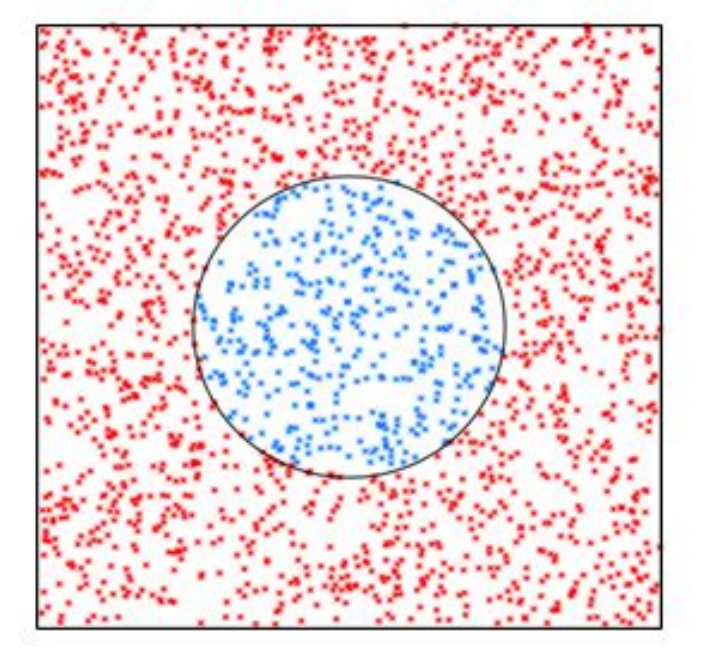
$\pi \approx 2.8284$

Monte Carlo Approach

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



$\pi \approx 3.136$



$\pi \approx 3.1440$

Review: Random Numbers

- **Random variables** – expressions whose value is the outcome of a random experiment
- **Sample space** – set of all possible outcomes
- **Probability distribution** - probability $p(x)$ of selecting an outcome x in the sample space
- **Sampling** – selection of a subset of a sample space (valid when it reflects $p(x)$)

- **Pseudo-Random Number Generator** (PRNG) - deterministic algorithm that generates a sequence of quasi-“random” numbers based on an initial **seed** (starting point in the pre-determined sequence)
  - PRNGs typically generate real numbers between 0 and 1 with equal (**uniform**) probability
  - The ability to uniformly sample from [0,1] enables sampling from other sample spaces that have non-uniform probabilities

see CS109

Monte Carlo

- Typically used in higher dimensions (5D or more)
- Random (pseudo-random) numbers are used to 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 low dimensional problems (i.e., 1D, 2D, 3D)
- But <u>tractable</u> for high dimensional problems

Monte Carlo Integration (in 1D)

- Consider: $\int_a^b f(x)dx$

- Generate N random samples X_i in the interval $[a, b]$

- A Monte Carlo estimate for the integral is:

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

- This is a simple averaging of all the sample results

Importance Sampling

(Trivial) Motivating Case:

- Suppose $f(x)$ is only nonzero in $[a_1, b_1] \subset [a, b]$, i.e. $\int_a^b f(x)\mathrm{d}x = \int_{a_1}^{b_1} f(x)\mathrm{d}x$
- Then, $X_i \notin [a_1, b_1]$ do not contribute to the integral
- It is more efficient to change $p(x)$ to a uniform distribution over $[a_1, b_1]$ (instead of $[a, b]$)

General Case:

- The probability distribution $p(x)$ should prefer samples from areas with higher contributions to (or higher **importance** to) the integral
- Given any $p(x)$ (with $\int_a^b p(x)\mathrm{d}x = 1$), the Monte Carlo estimate is:

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

- When $p(x) = \frac{1}{b-a}$ (i.e., uniform sampling), this reduces to: $F_N = \frac{1}{N}\sum_{i=1}^{N}(b-a)f(X_i)$

Importance Sampling

- Monte Carlo estimates for $\int_0^1 x^2 \mathrm{d}x$ with $N = 100$ samples:

		Relative Error

- Typically, the more $p(x)$ “resembles” $f(x)$, the lower the error
- So, choose $p(x)$ based on physical/known principles or an approximate solution
- Caution: importance sampling does not necessarily reduce error (and can make errors worse)

Photon Emission

- Choose some number of photons; divide them amongst the lights (based on relative power)
  - For efficiency/implementation, every photon has the <u>same strength</u>
  - So, brighter lights emit more (not stronger) 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 (randomly choosing the position within a cell)
- Emission Direction:
  - Randomly choose a direction on a sphere, a hemisphere, a subset of the sphere (for spotlights), etc.
- In some cases (e.g. consider the sun), a large number of photons will miss the scene entirely
  - Can ignore those photons (never emitting them)
  - Restrict the light to an appropriate sub-light
  - Scale down the light's energy to that of the sub-light (when dividing up photons)

Light Map

- Using the ray tracer, follow the photon's path (until it intersects scene geometry)
- Each time a photon intersects geometry, add its data to the light map (as **incoming light**)
- Make a copy of the photon to store in the light map
  - Don't delete the photon, or move it into the light map
  - The photon *may* still bounce around a bit more (if it doesn't get absorbed)
- Store (in the light map):
  - The point of impact (a location in 3D space)
  - The incoming direction (the ray direction from the ray tracer)
  - Don't need to store the energy (since all photons have the same energy)

Absorption

- After storing the photon's data in the light map, determine what happens next
- Objects absorb some incoming light (which is why they have a color)
- There is a chance that the photon is absorbed:
  - Absorbing a fraction of the photon's energy leads to unequal energy photons
  - Instead, use the fraction of light energy that would be absorbed to calculate a probability that the (entire) photon is absorbed
- Generate a random number (between 0 and 1), and compare it to the probability of absorption (Russian Roulette)
- If absorbed, the process stops (for this photon)
- Otherwise, the photon bounces

Bouncing

- Compute a new direction by mapping BRDF directions into probabilities
  - E.g. a purely diffuse BRDF would have equal probabilities for every hemisphere direction
- Generate a random number, and use it to determine the bounce direction
- Then, use the ray tracer to follow the photon’s path
- At the next intersection, (again) store the photon’s data in the light map
- Then (once again), check for absorption; if not absorbed, bounce again, etc.

- Use a pre-determined maximum number of bounces (before termination)
  - Can (usually) 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 a light map

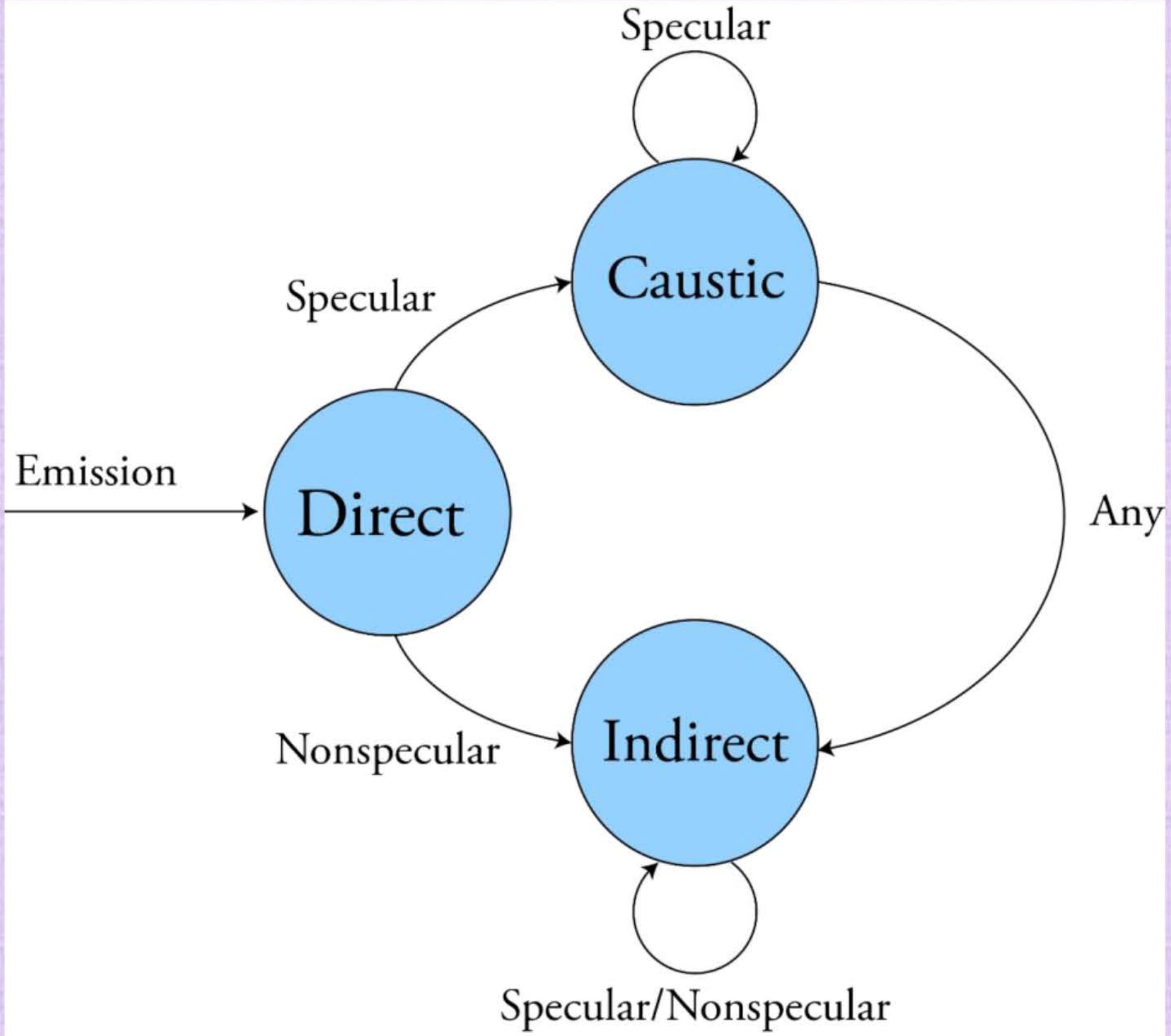
- Thus, the first time a photon emitted from a light source hits an object, it is not stored in the light map (since this is direct lighting)

- This also makes the light map a lot more efficient, since direct illumination information is not being stored

Separating Diffuse/Specular

- It’s more convenient/efficient to treat diffuse and specular lighting separately
- When bouncing a photon, first determine (randomly) if the photon undergoes:
  - absorption (deleted)
  - (or) a diffuse bounce
  - (or) a specular bounce
- If bouncing, randomly determine the (diffuse or specular) bounce direction
- Use 2 light maps:
  - <u>Caustic Map</u>: store photons that have had <u>specular bounces only</u> (prior to being stored in the map)
  - <u>Indirect Lighting Map</u>: store photons that have had <u>at least one diffuse</u> <u>bounce</u>

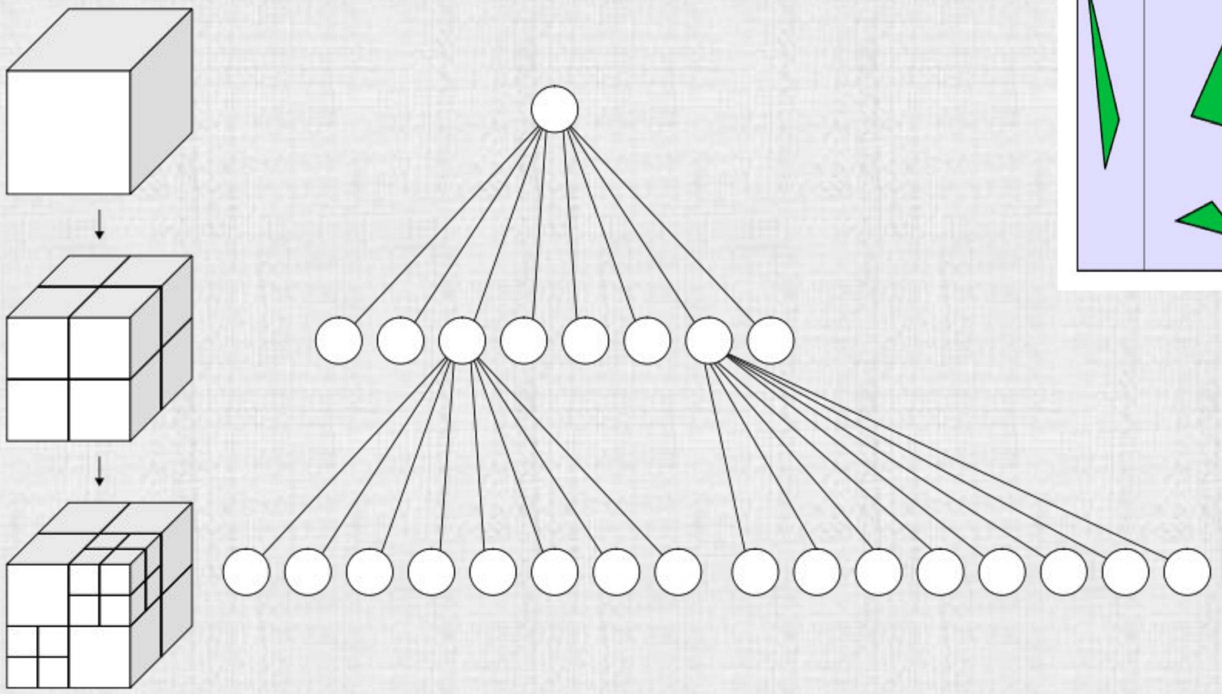
Separate Diffuse/Specular Photon Maps

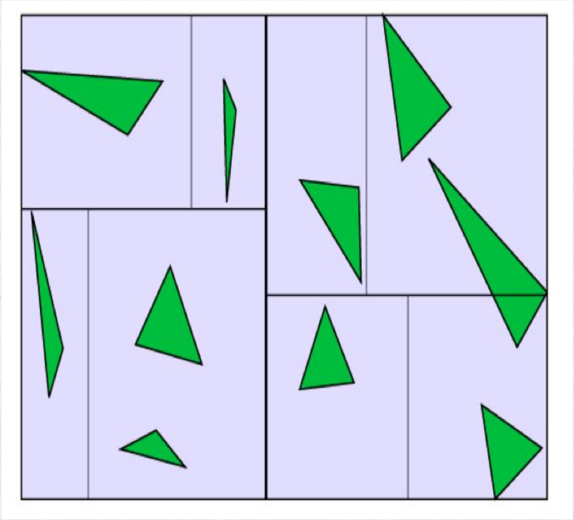


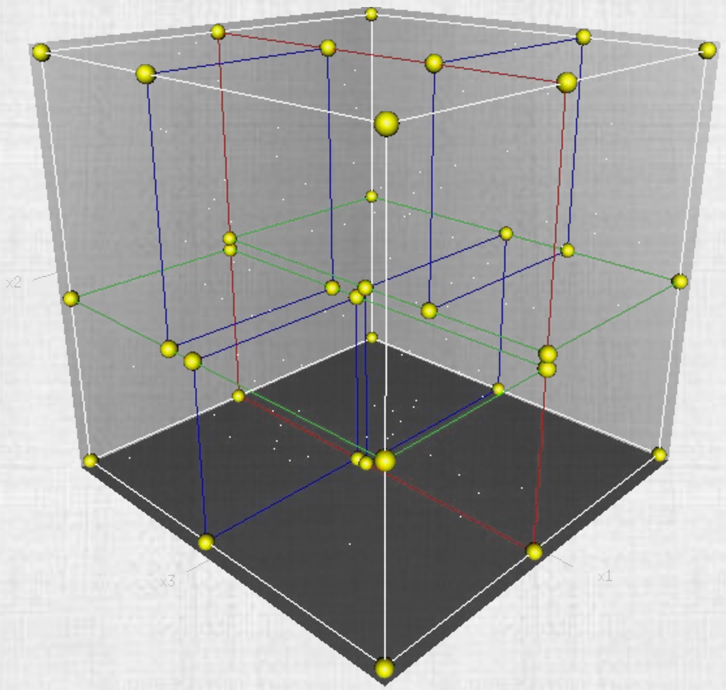
Caustics

Aside: Code Acceleration

• Photons are typically stored in an octree or K-D tree acceleration structure (so that the information they contain is more efficiently retrieved)

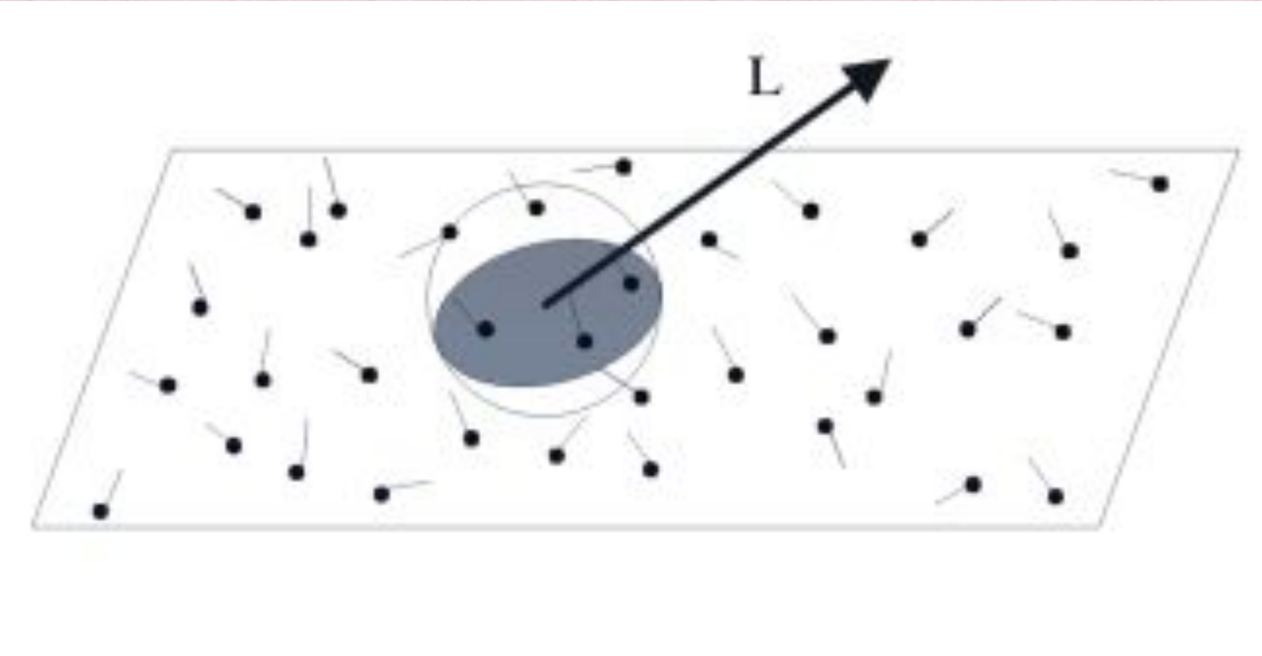






Gathering Radiance

- Trace rays from the camera and intersect with objects (as usual)
- Use shadow rays for <u>direct</u> lighting (as usual)
- Estimate the radiance contribution to the ray from <u>caustics</u> and <u>indirect</u> lighting using the respective light maps:
  - Use the N closest photons to the point of intersection (with the aid of an acceleration structure)



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