Photon Mapping

Photon Mapping

Photon mapping is a two-pass algorithm

1. Pass 1: Tracing of photons from light sources, and caching them in a *photon map*
2. Pass 2: Tracing from the eye and approximating indirect illumination using the photons
   It's similar to light tracing, but connects to eye subpaths using kernel density estimation.

The first pass is to generate a photon map, which mainly includes:

- Photon emission
- Photon scattering
- Photon storing

Pass 1

Photon Emission

A photon is generated by a randomly chosen light source.

- $\mathbf{x}_p$: position
- $\vec{\omega}_p$: incident direction
- Φ_p: photon power → flux not radiance, each photon carries multiple wavelengths

The photon power is given by

$$\Phi_p = \frac{1}{M} \frac{L_e(\mathbf{x}_p, \vec{\omega}_p) \cos\theta_p}{p(\mathbf{x}_p) p(\vec{\omega}_p | \mathbf{x}_p)}$$

where M is the number of emitted photons and

- $p(\mathbf{x}_p)$ is the pdf of sampling position $\mathbf{x}_p$ on surface area of light
- $p(\vec{\omega}_p | \mathbf{x}_p)$ is the pdf of sampling direction $\vec{\omega}_p$ conditioned on $\mathbf{x}_p$

An interesting fact is that, if the pdfs are proportional to emission, i.e.

$$p\left(\mathbf{x}_p\right) = \frac{\int_{H^2} L_e\left(\mathbf{x}_p, \vec{\omega}\right) \cos\theta_p d\vec{\omega}}{\int_A \int_{H^2} L_e(\mathbf{x}, \vec{\omega}) \cos\theta_p d\vec{\omega} d\mathbf{x}} \quad p\left(\vec{\omega}_p \mid \mathbf{x}_p\right) = \frac{L_e\left(\mathbf{x}_p, \vec{\omega}_p\right) \cos\theta_p}{\int_{H^2} L_e\left(\mathbf{x}_p, \vec{\omega}\right) \cos\theta_p d\vec{\omega}}$$

Then

$$\begin{aligned}\Phi_p &= \frac{1}{M} \frac{L_e\left(\mathbf{x}_p, \vec{\omega}_p\right) \cos\theta_p}{p\left(\mathbf{x}_p\right) p\left(\vec{\omega}_p \mid \mathbf{x}_p\right)} \\ &= \frac{1}{M} \frac{\cancel{L_e\left(\mathbf{x}_p, \vec{\omega}_p\right)} \cos\theta_p}{\frac{\cancel{\int_{H^2} L_e(\mathbf{x}_p, \vec{\omega}) \cos\theta_p d\vec{\omega}}}{\int_A \int_{H^2} L_e(\mathbf{x}, \vec{\omega}) \cos\theta_p d\vec{\omega} d\mathbf{x}}} \frac{\cancel{L_e\left(\mathbf{x}_p, \vec{\omega}_p\right)} \cos\theta_p}{\cancel{\int_{H^2} L_e(\mathbf{x}_p, \vec{\omega}) \cos\theta_p d\vec{\omega}}}} = \frac{\Phi \cos\theta_p}{M}\end{aligned}$$

where Φ is the total power of the light source.

If you perfectly importance sample the emitted radiance, just take the total power and divide by # of emitted photons

Photon Scattering

Photons can be:

- absorbed or scattered (reflected or refracted)
- BSDF sampling chooses either reflection or refraction
- the power of the scattered photon is lowered to account for absorption

Every time a photon hits a surface, it gets reflected or refracted by BRDF sampling. The power of the photon is lowered the same way as the radiance, i.e.

$$\Phi' = \Phi * \texttt{absDot}(n, \omega') * \texttt{BxDF}(x', -\omega, \omega') / \texttt{pdf()}$$

Photon Storing

Every time a photon hits a surface, its power, incident direction and position is stored. Note that only storing only happens on diffuse surfaces, specular surfaces need to be handled using path tracing from the camera.

```C++
struct Photon
{
        float position[3];
        float power[3];
        float direction[3];
};
```

The up above data structure takes 36 bits. We could make it more efficient using

```C++
struct Photon
{
        float position[3];
        char power[4]; // Packed RGBE format
        char phi, theta; // Packed direction
};
```

Now we can write the pseudocode for photon mapping:

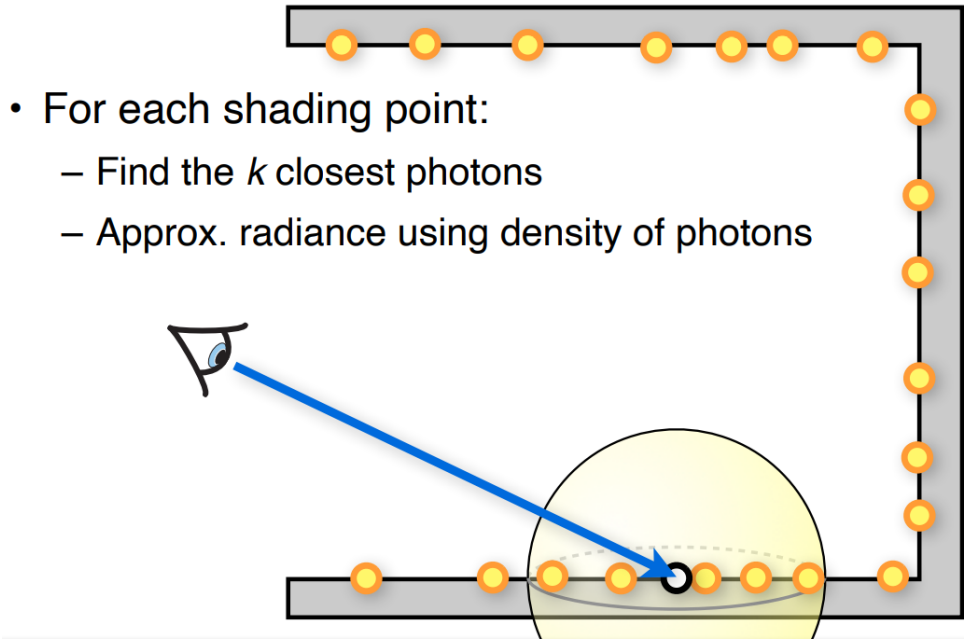
```C++
void generatePhotonMap()
{
        repeat:
                (l, Probl) = chooseRandomLight();
                (x, ω, Φ) = emitPhotonFromLight(l);
                tracePhoton(x, ω, Φ / Probl);
        until we have enough photons;
        divide all photon powers by number of emitted photons;
}

void tracePhoton(x, ω, Φ)
{
        (x', n) = nearestSurfaceHit(x, ω);
        possiblyStorePhoton(x', ω, Φ);
        (ω', pdf) = sampleBxDF(x', -ω);
        Φ' = Φ * absDot(n, ω') * BxDF(x', -ω, ω') / pdf ;
        if survivedRussianRoulette(Φ, Φ')
                return tracePhoton(x', ω', Φ');
}

void survivedRussianRoulette(Φ, Φ')
{
        p = 1 - min(1, Φ' / Φ);
        if rand() < p:
        // terminate
                return false;
        else:
        // continue with re-weighted power
                Φ' = Φ' / (1-p);
                return true;
}
```

In the code we probabilistically terminate the photon walk using Russian roulette. In this way, all photons in the photon map would have the same (or similar) power.

Pass 2



Radiance Estimate

The radiance is estimated by

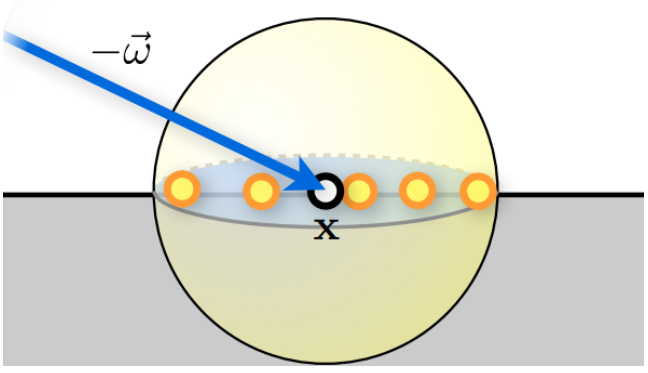
$$
\begin{aligned}
L_r(\mathbf{x}, \vec{\omega}) &= \int_{H^2} f_r\left(\mathbf{x}, \vec{\omega}', \vec{\omega}\right) L_i\left(\mathbf{x}, \vec{\omega}'\right) \cos\theta' d\vec{\omega}' \\
&= \int_{H^2} f_r\left(\mathbf{x}, \vec{\omega}', \vec{\omega}\right) \frac{d\Phi^2\left(\mathbf{x}, \vec{\omega}'\right)}{\cos\theta' d\vec{\omega}' dA} \cos\theta' d\vec{\omega}' \\
&= \int_{H^2} f_r\left(\mathbf{x}, \vec{\omega}', \vec{\omega}\right) \frac{d\Phi^2\left(\mathbf{x}, \vec{\omega}'\right)}{dA} \\
&\approx \sum_{p=1}^{n} f_r\left(\mathbf{x}, \vec{\omega}_p, \vec{\omega}\right) \frac{\Delta\Phi_p\left(\mathbf{x}, \vec{\omega}_p\right)}{\Delta A}
\end{aligned}
$$

Approach 1

First define area, then find photons

$$
L_r(\mathbf{x}, \vec{\omega}) \simeq \sum_{p=1}^{k} f_r(\mathbf{x}, \vec{\omega}_p, \vec{\omega}) \frac{\Phi_p}{\pi r^2}
$$

where r is the assumed disk region and k is the number of photons on the disk.



Approach 2

First find k nearest photons, then define area:

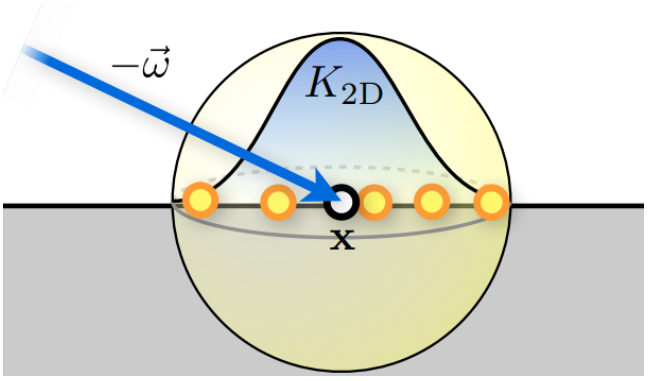
$$L_r(\mathbf{x}, \vec{\omega}) \simeq \sum_{p=1}^{k-1} f_r(\mathbf{x}, \vec{\omega}_p, \vec{\omega}) \frac{\Phi_p}{\pi r_k^2}$$

where r_k is the distance to the k-th photon and we discard the k-th photon. Note that it's not r_p

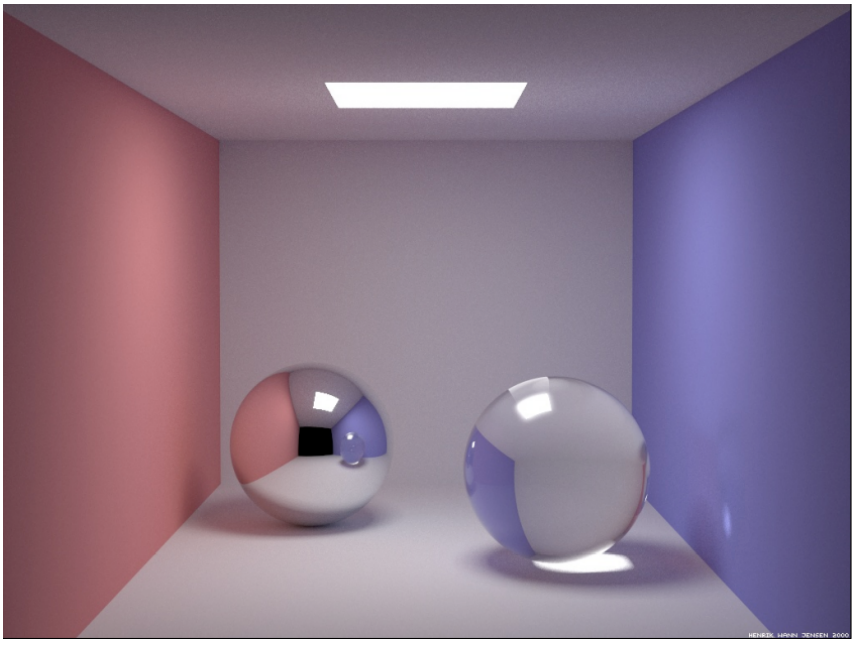
Approach 3

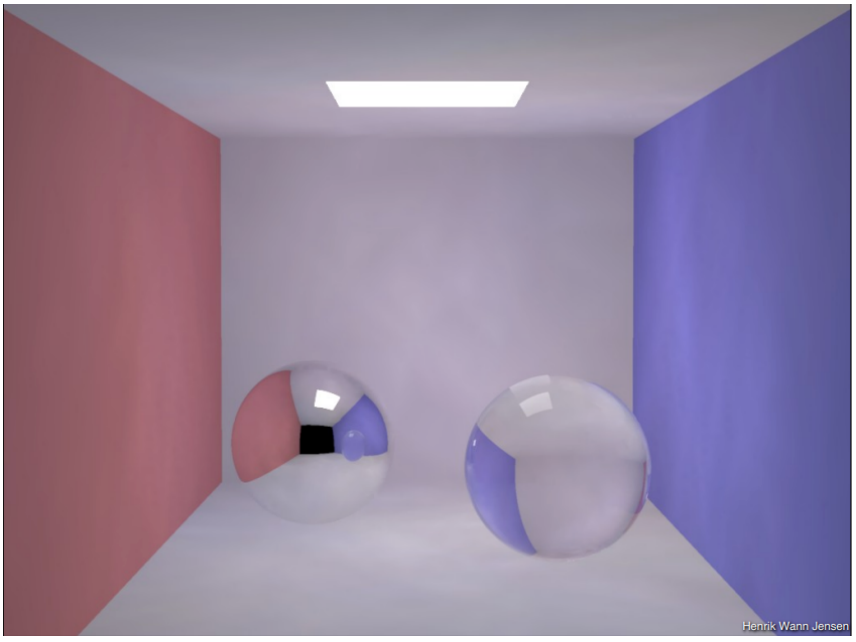
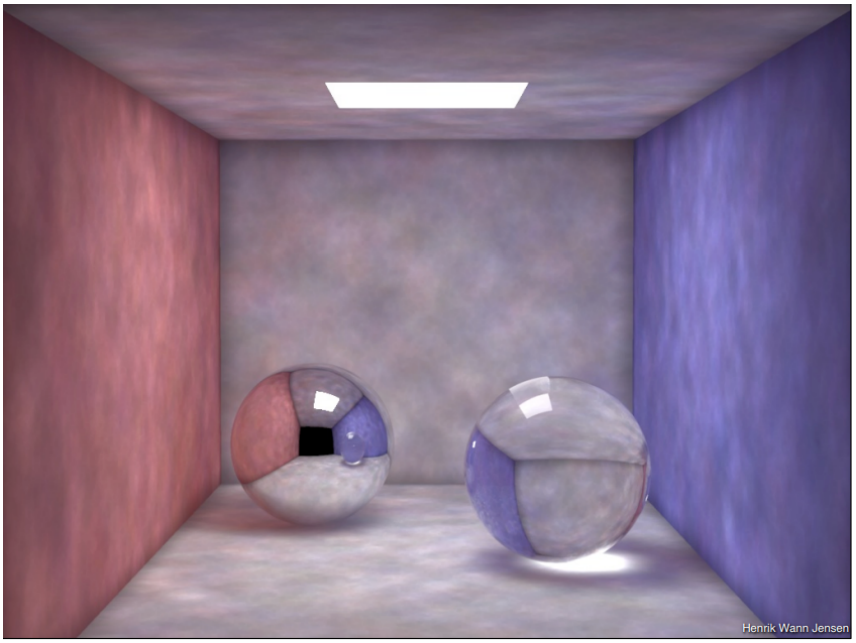
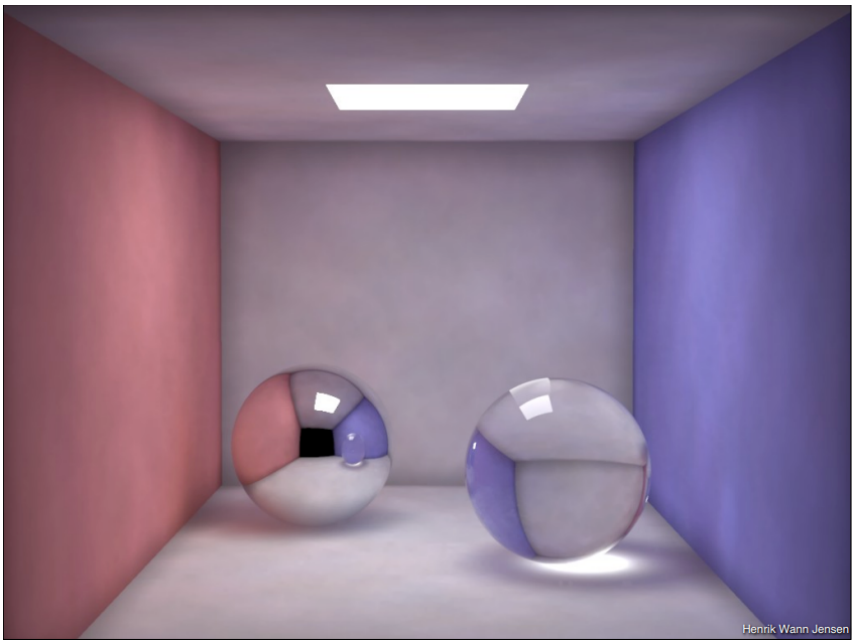
Use a non-constant kernel

$$L_r(\mathbf{x}, \vec{\omega}) \approx \sum_{p=1}^{k-1} f_r\left(\mathbf{x}, \vec{\omega}_p, \vec{\omega}\right) \Phi_p K_{2\mathrm{D}}\left(r_p, r_k\right)$$

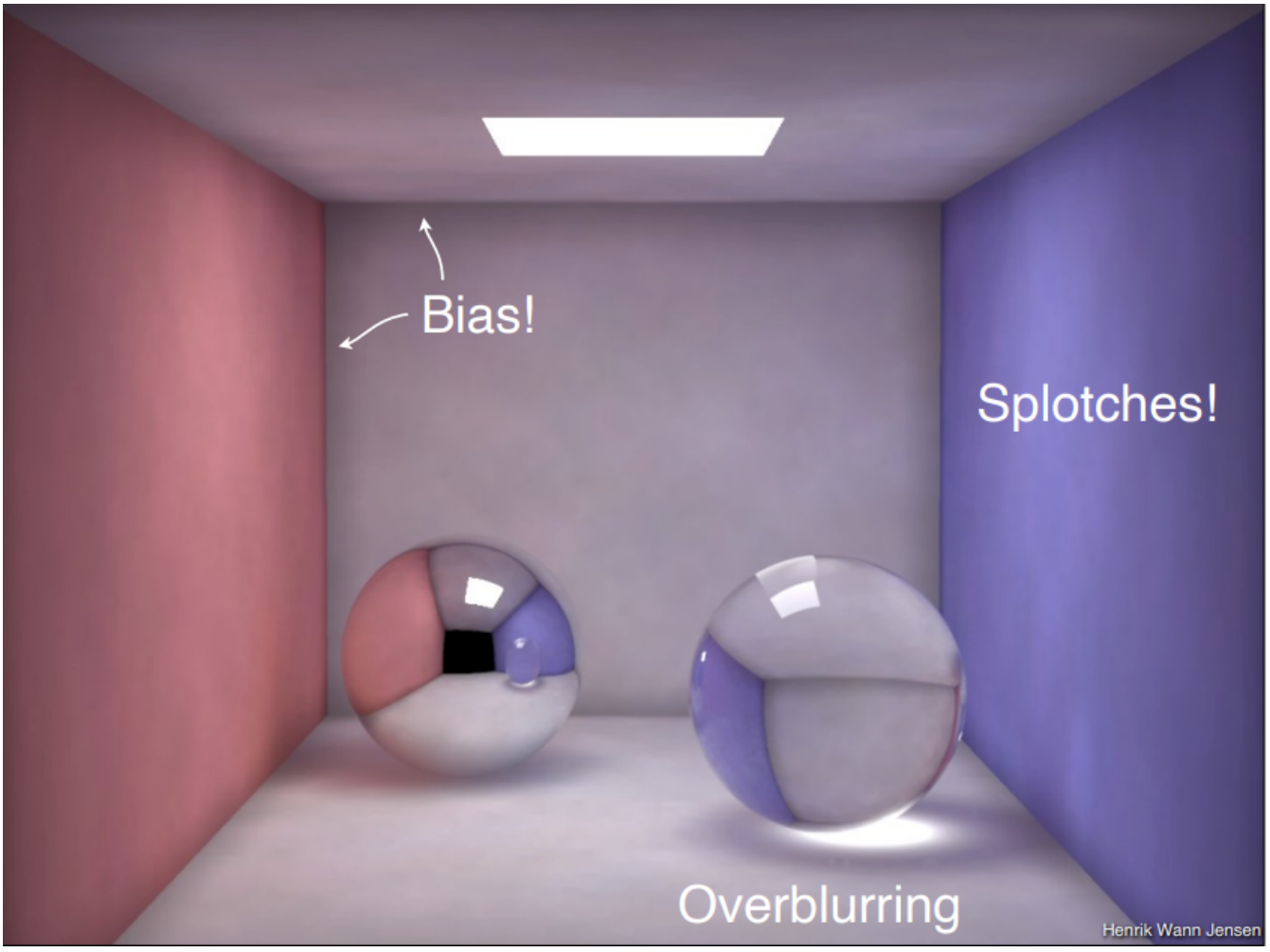


A naïve photon mapping example is given below, from which we could see its strengths and weaknesses:

Setting	Picture
Path Tracing	

Setting	Picture
200 photons / 50 photons in radiance estimate	
100,000 photons / 50 photons in radiance estimate	
500,000 photons / 500 photons in radiance estimate	

We see that photon mapping introduces a lot of bias, splotches and overblurrings.



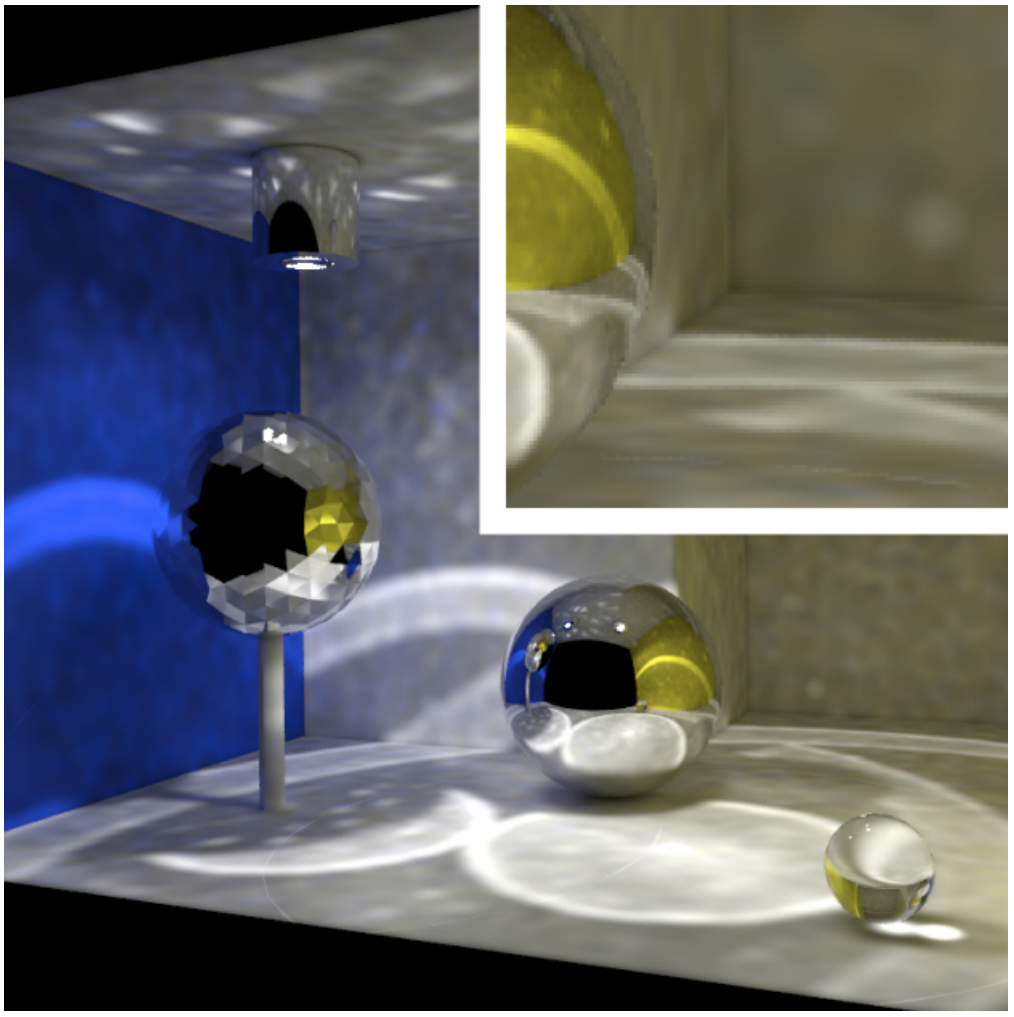
Progressive Photon Mapping

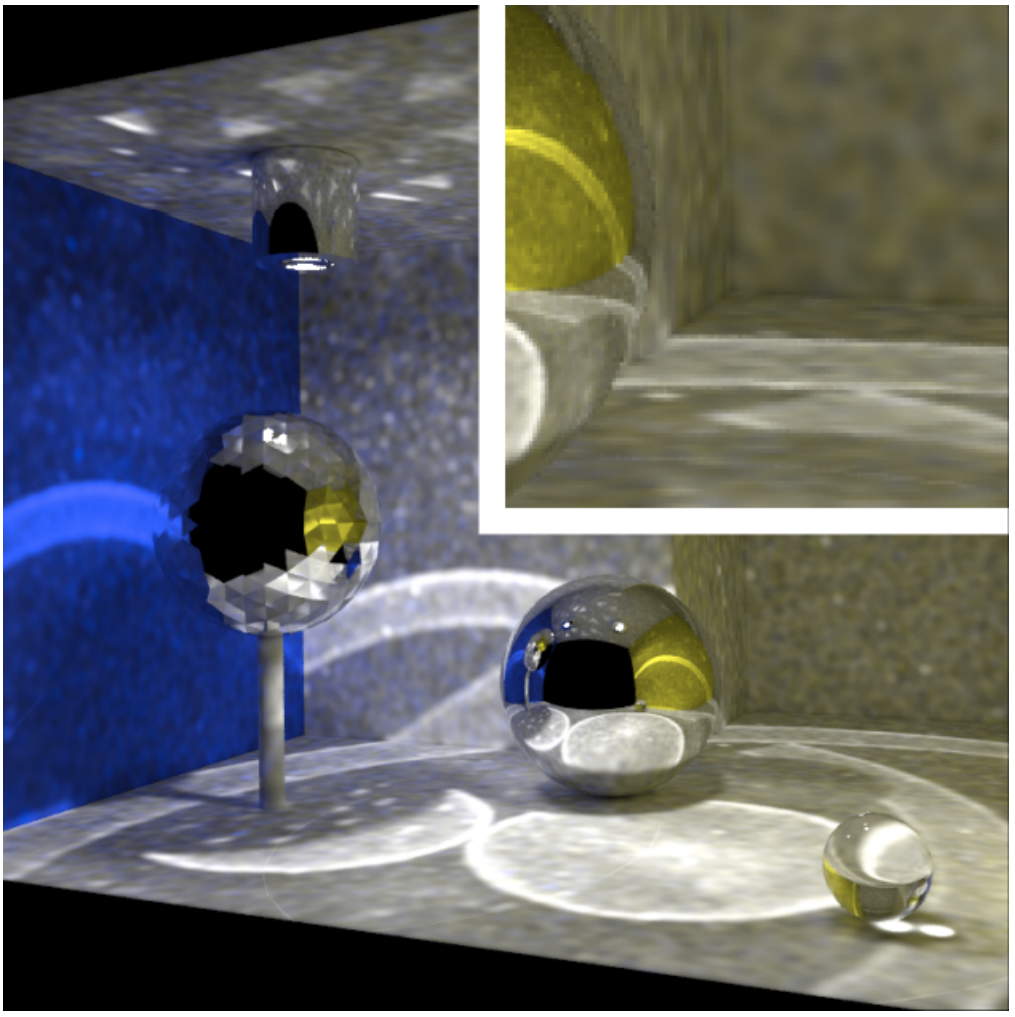
Main idea:
Progressively shrink the density estimation kernel.

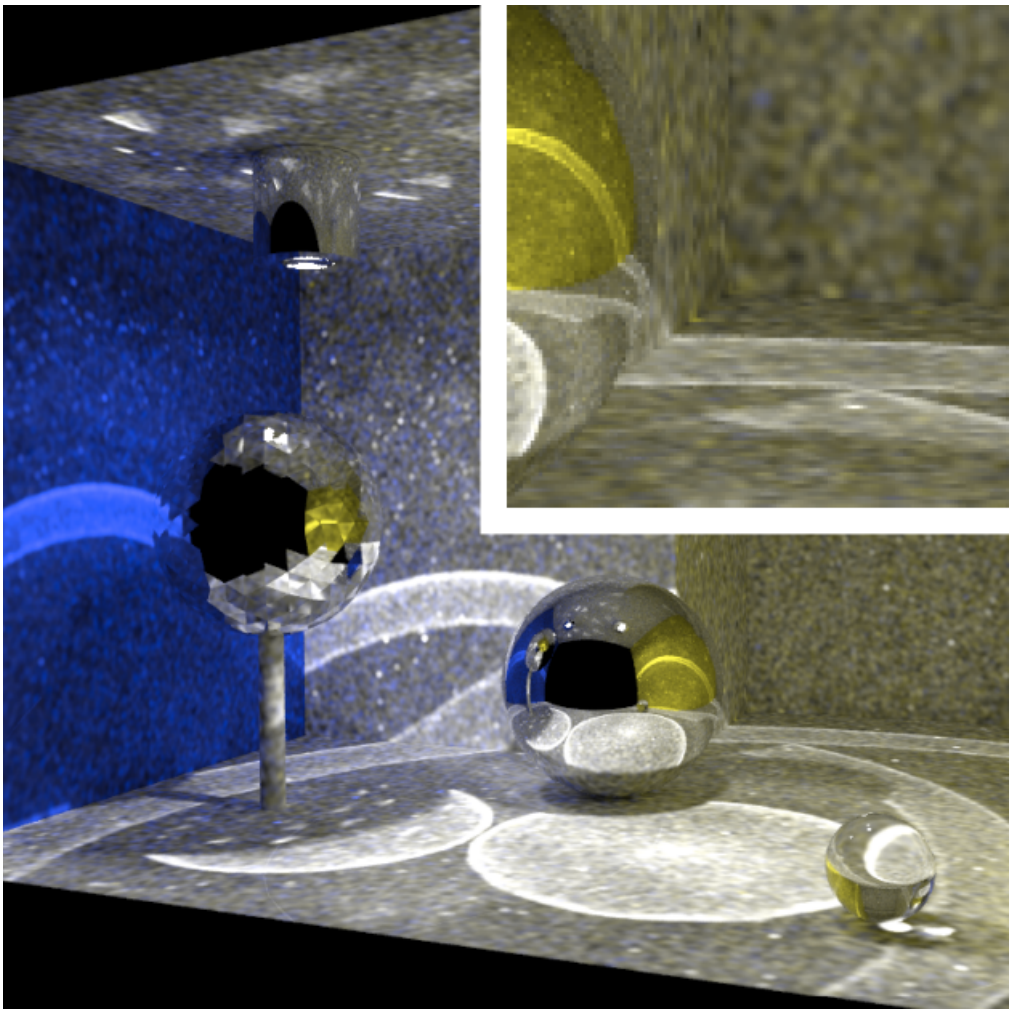
> Render independent images with smaller and smaller radius, and average. [Knaus & Zwicker 2011]

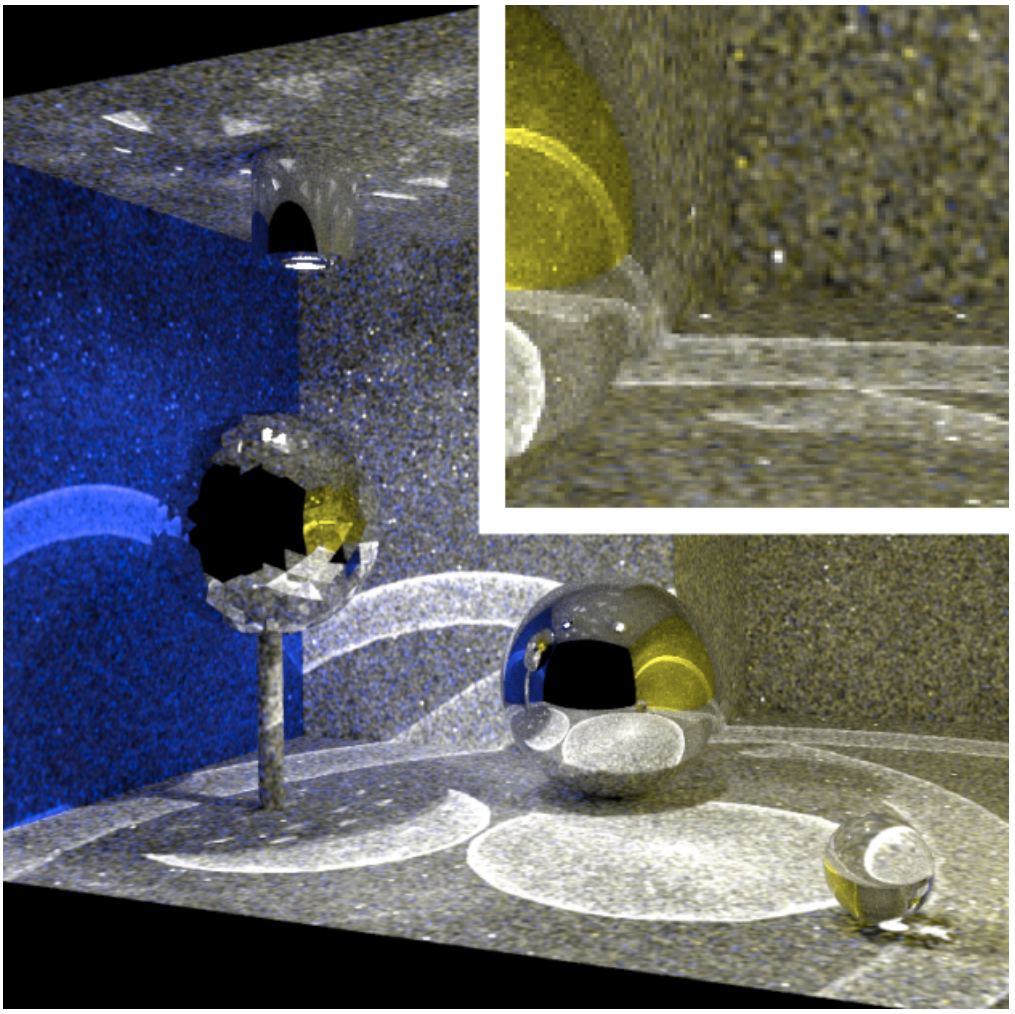
Example

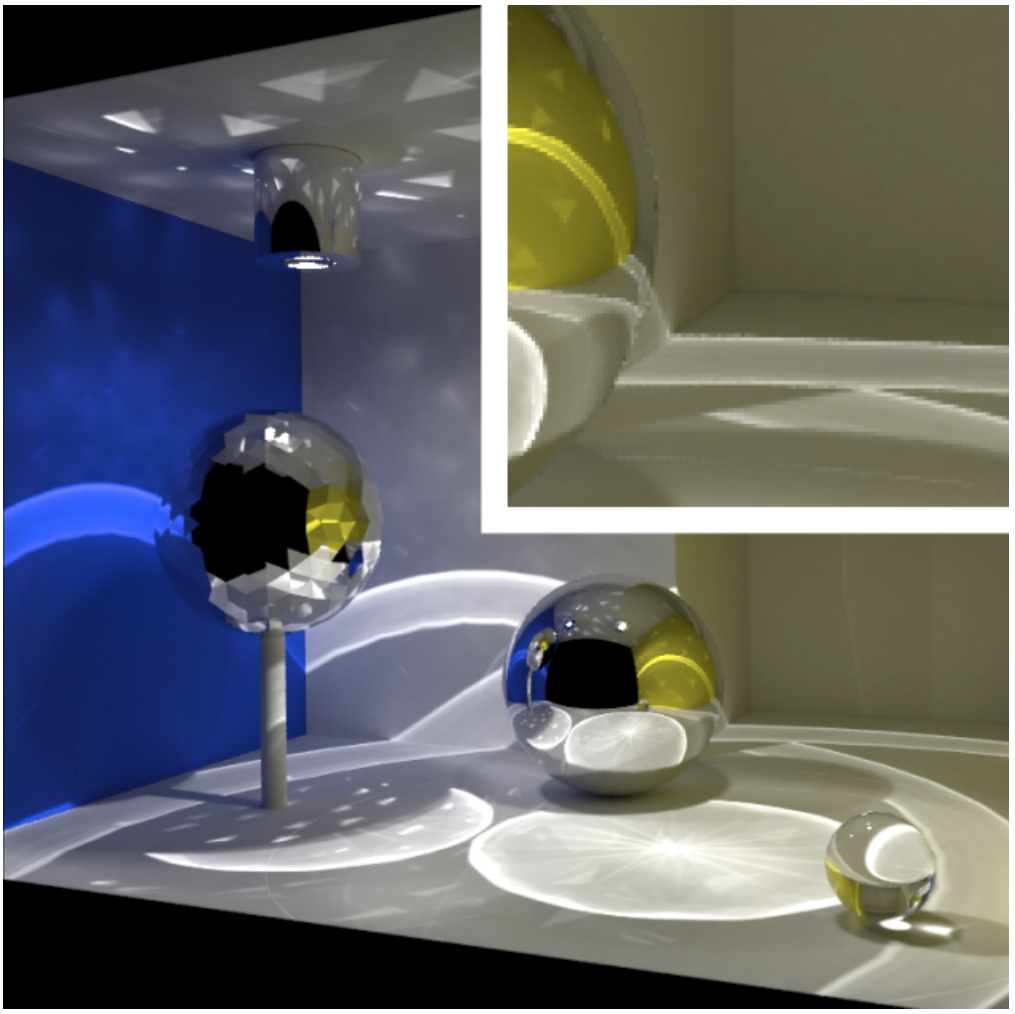
Setting	Rendering

Setting	Rendering
Image 1, $r = 20$	

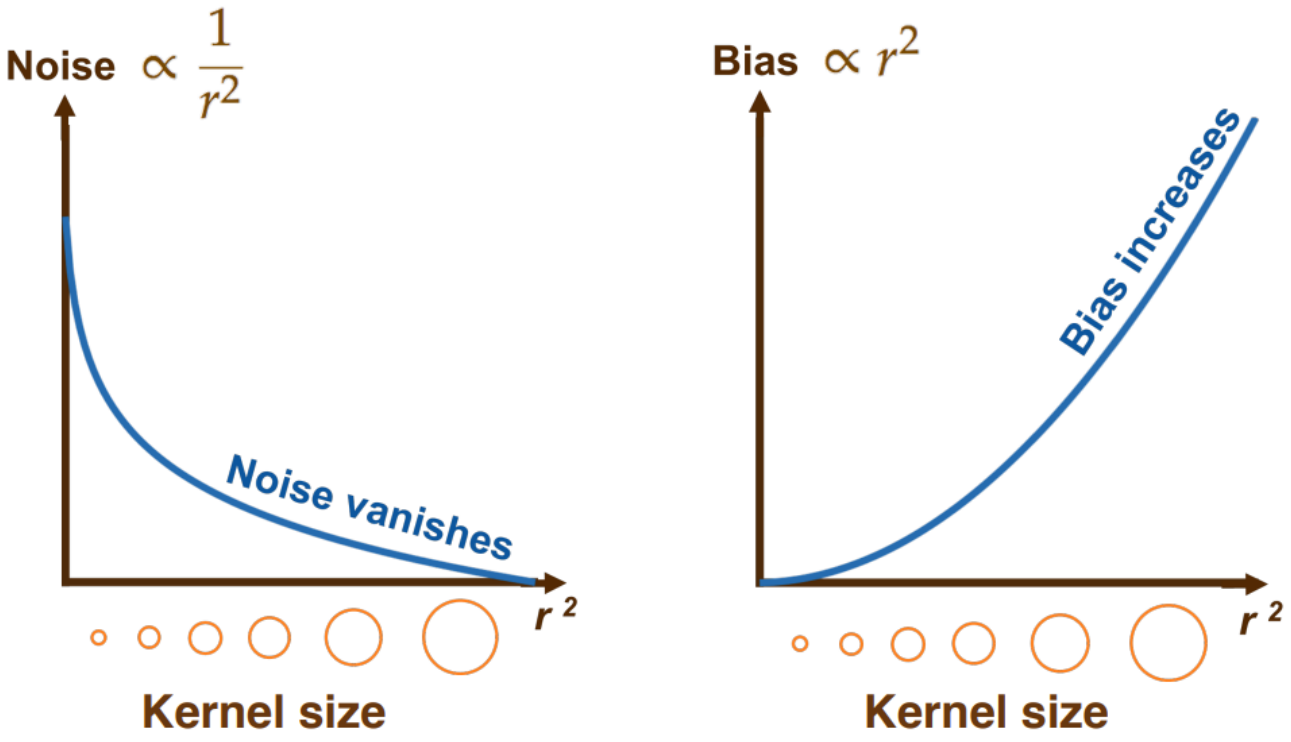
Setting	Rendering
Image 10, $r = 11.87$	

Setting	Rendering
Image 100, $r = 6.71$	

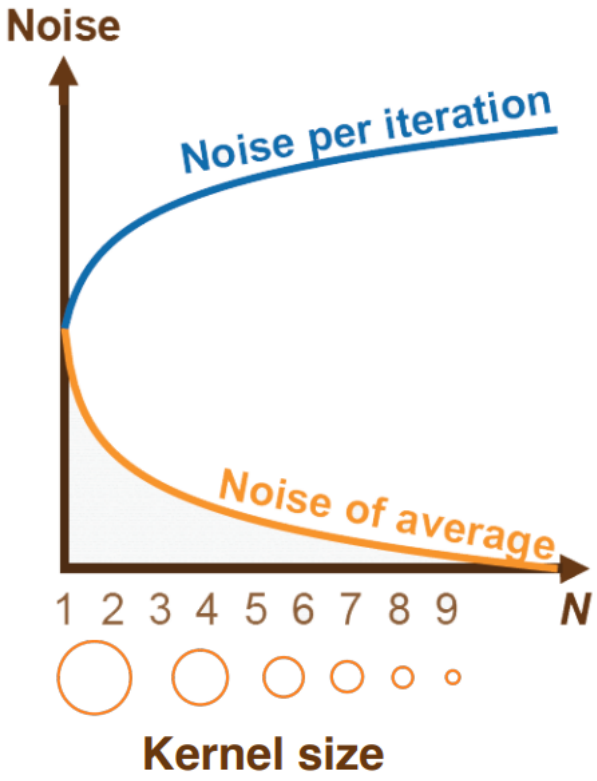
Setting	Rendering
Image 1000, $r = 3.78$	

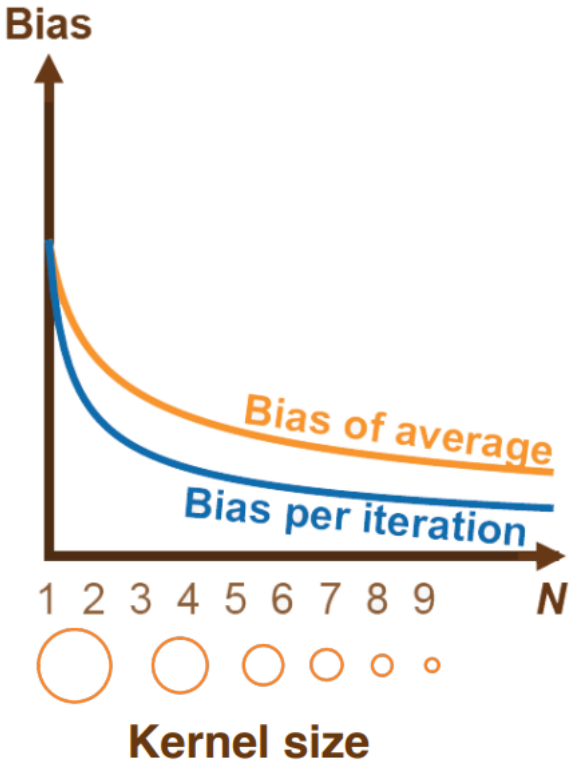
Setting	Rendering
Average image	

The bias and noise have a tradeoff on the kernel size



Using averaged image, both noise and bias decrease





Progressive Photon Mapping:

- Step 1: Photon tracing
- Step 2:
  - Trace camera paths
  - Evaluate radiance estimate using radius r_i
- Display running average
- Compute new radius $r_{i+1}^2 = \frac{i+\alpha}{i+1} r_i^2$ and repeat
  where $\alpha \in (0, 1)$ is the shrinking parameter