Caustics

USING PHOTON MAPS

CS500 RAY TRACING

BORJA PORTUGAL MARTIN

Caustics using Photon maps

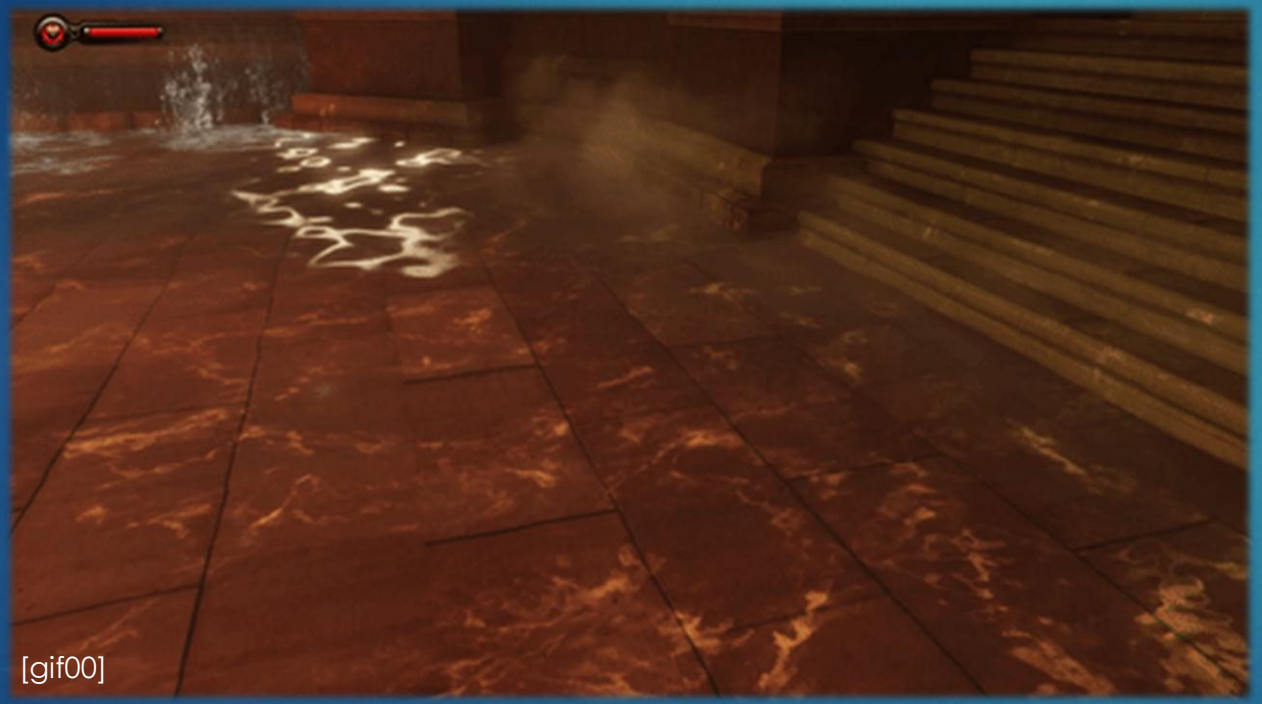
Contents

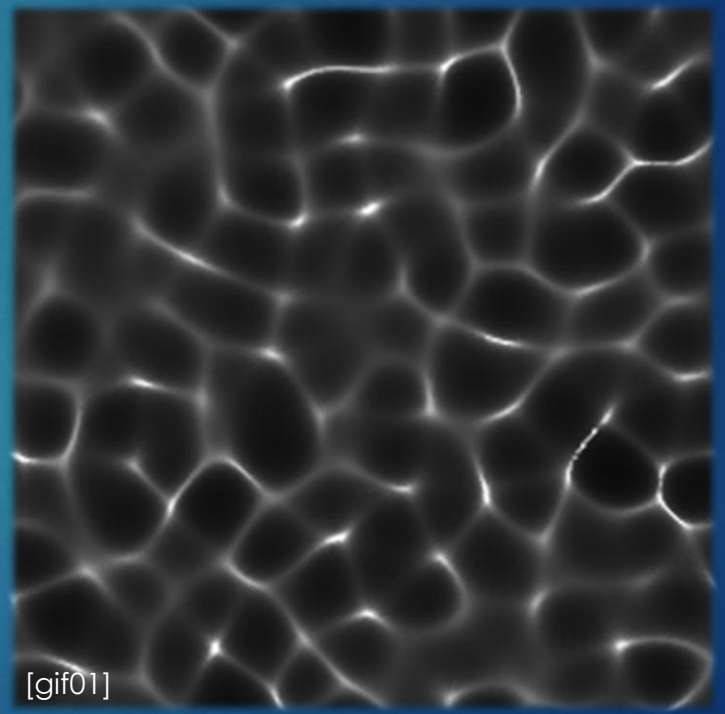
Motivation

Motivation

Caustics in Video Games

- Real time applications fake caustics

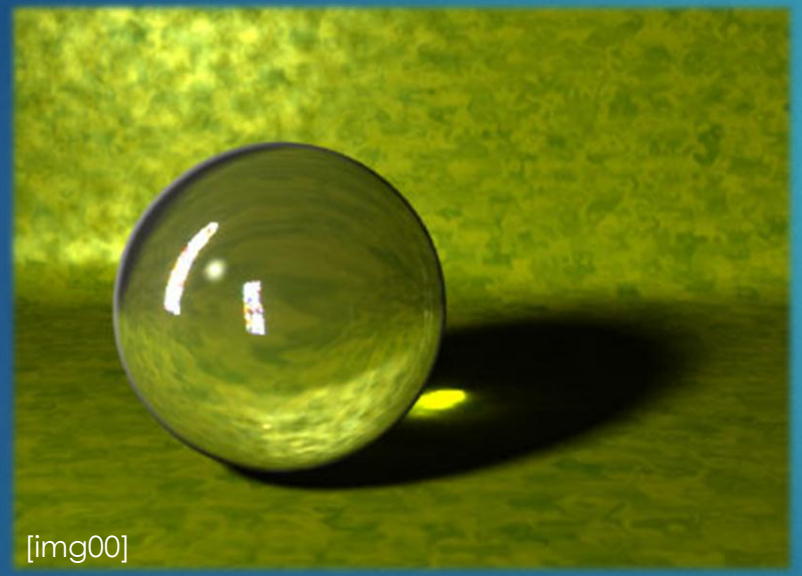

[gif00]


[gif01]

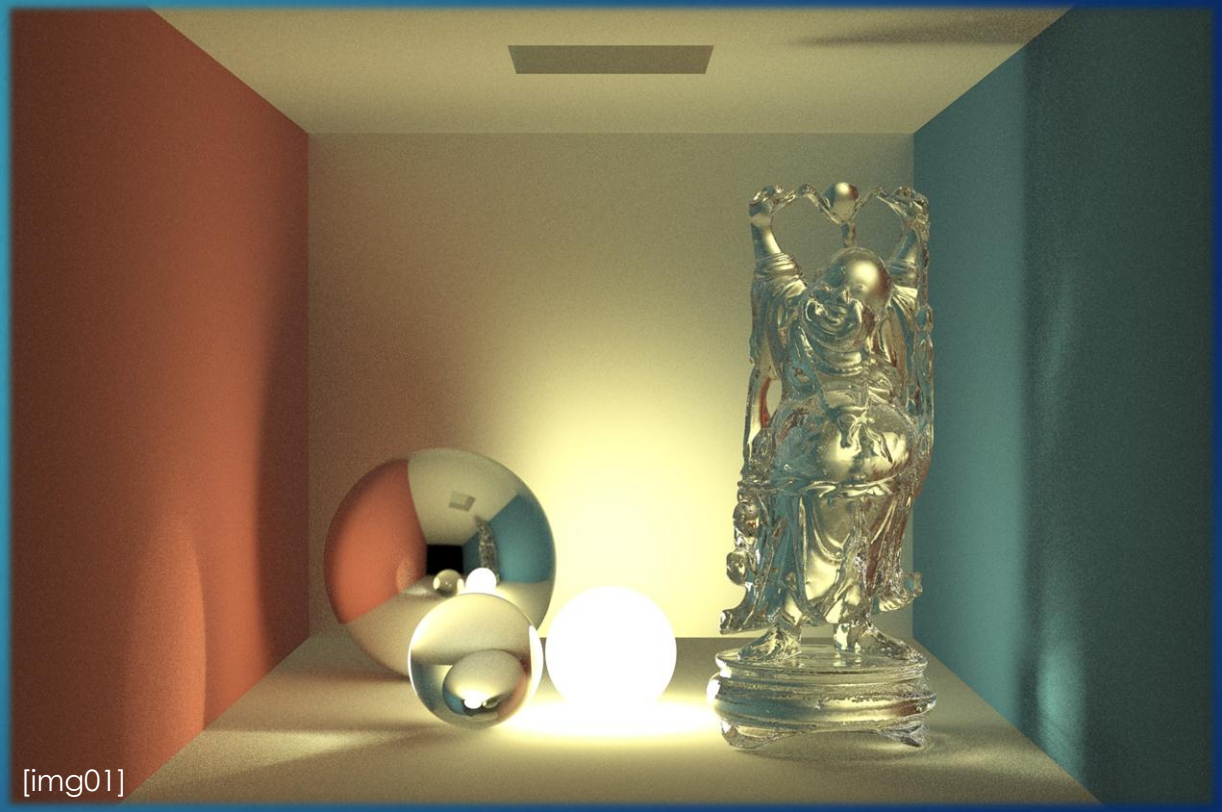
Motivation

Ray traced caustics

- Ray tracing is done offline, we can use all the computational power



[img00]

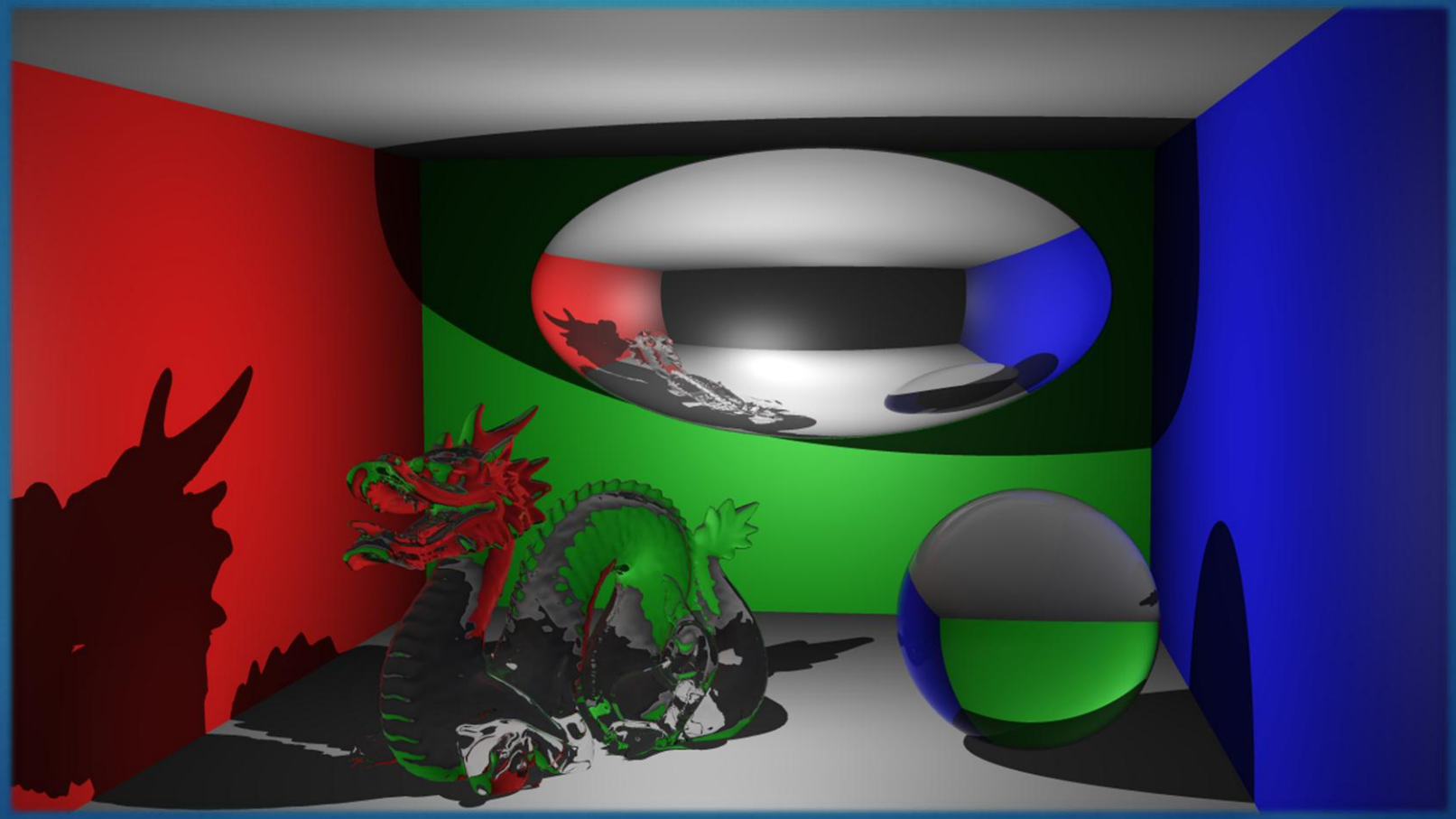


[img01]

Caustics

Motivation

- Glass in ray traced images without caustics do not seem correct



Approach

Approach

How can we do it?

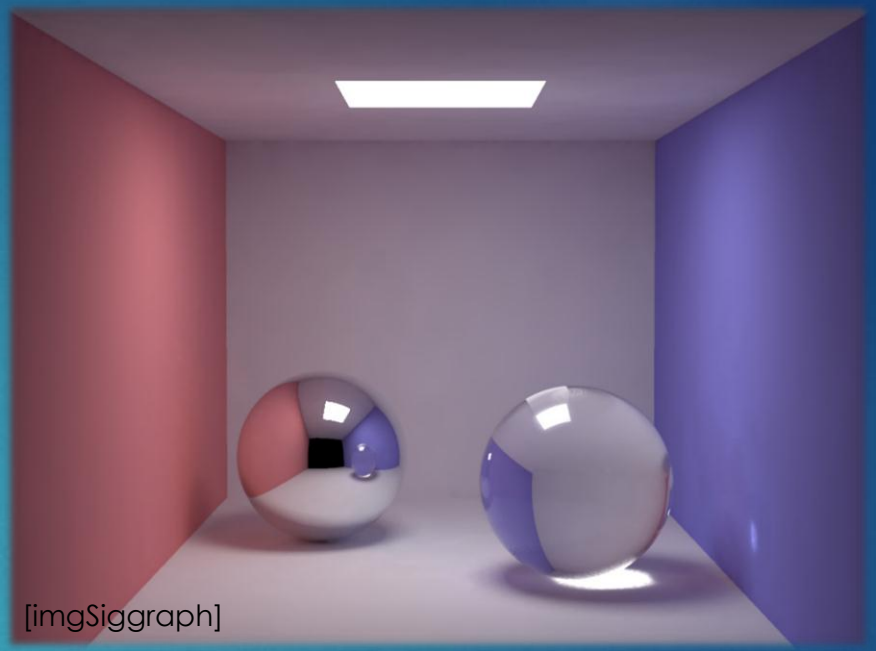
- We want to simulate what light does
  - Cast rays from the light source until we reach the camera (too expensive)
- Possible approaches:
  - Monte Carlo integration
  - Photon mapping: similar, but faster, approach to Monte Carlo

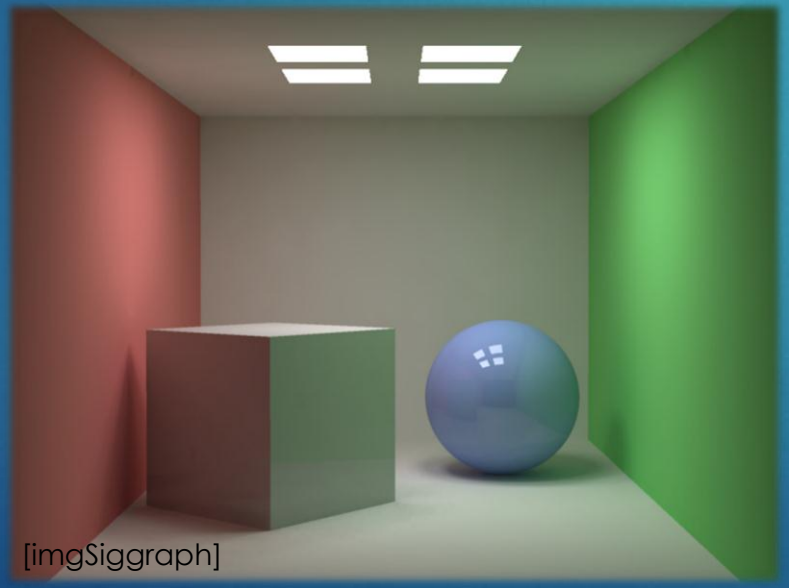
Photon maps

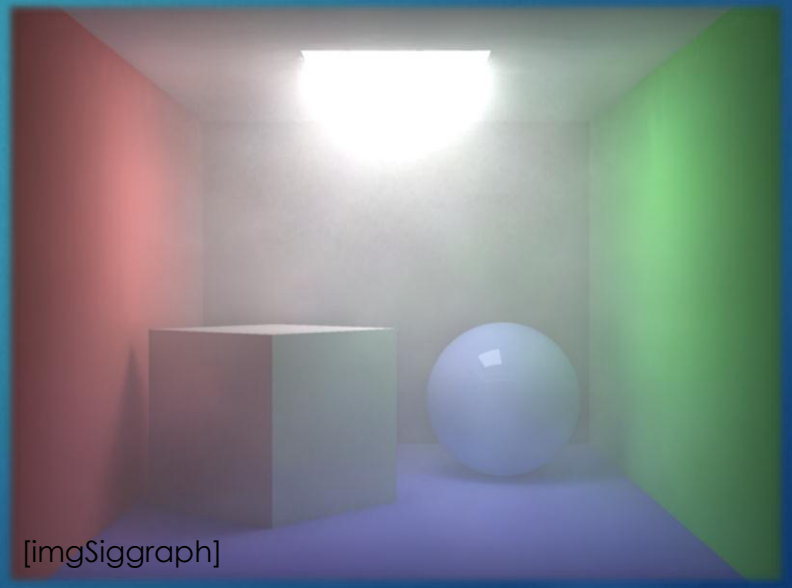
Photon maps

What are they used for?

- Caustics
- Global illumination
- Participating media


[imgSiggraph]

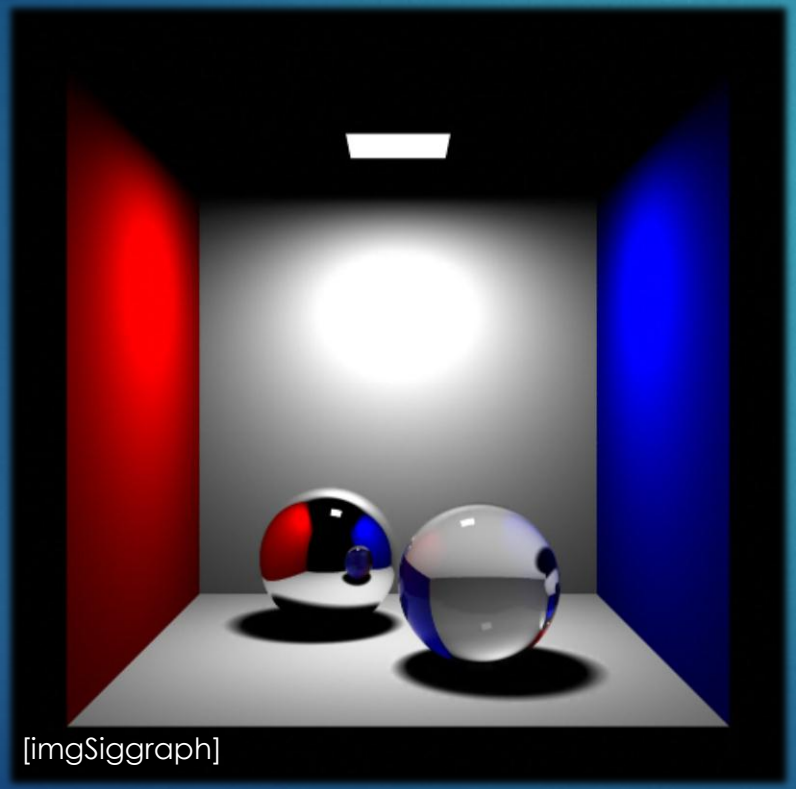

[imgSiggraph]


[imgSiggraph]

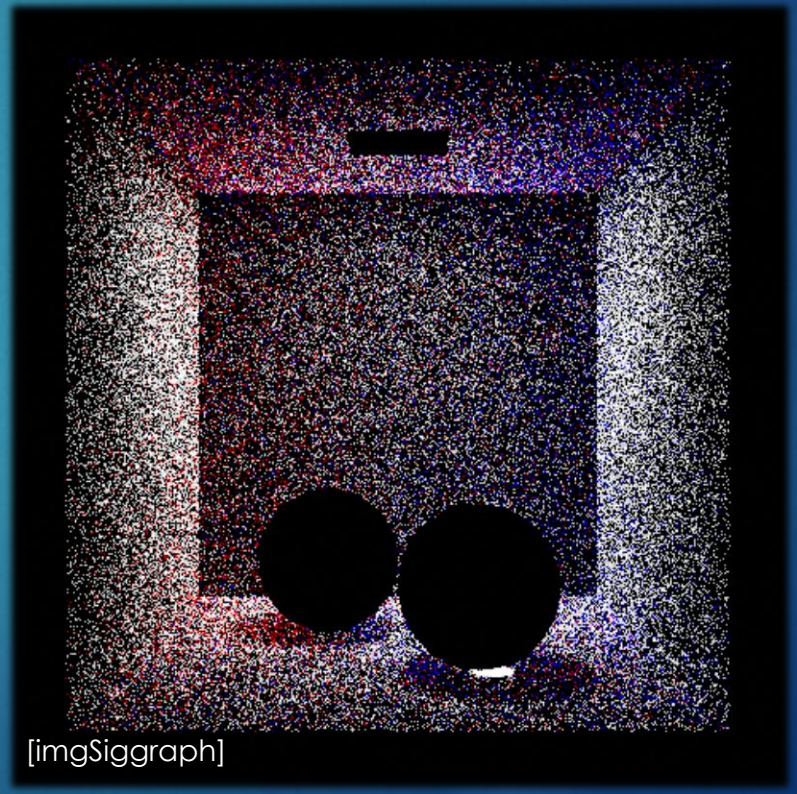
Photon maps

What are they?

- Collection of photons (light rays that reached a diffuse surface)
- 3D (its name made me, at first, think they were 2D)



[imgSiggraph]



[imgSiggraph]

Photon maps

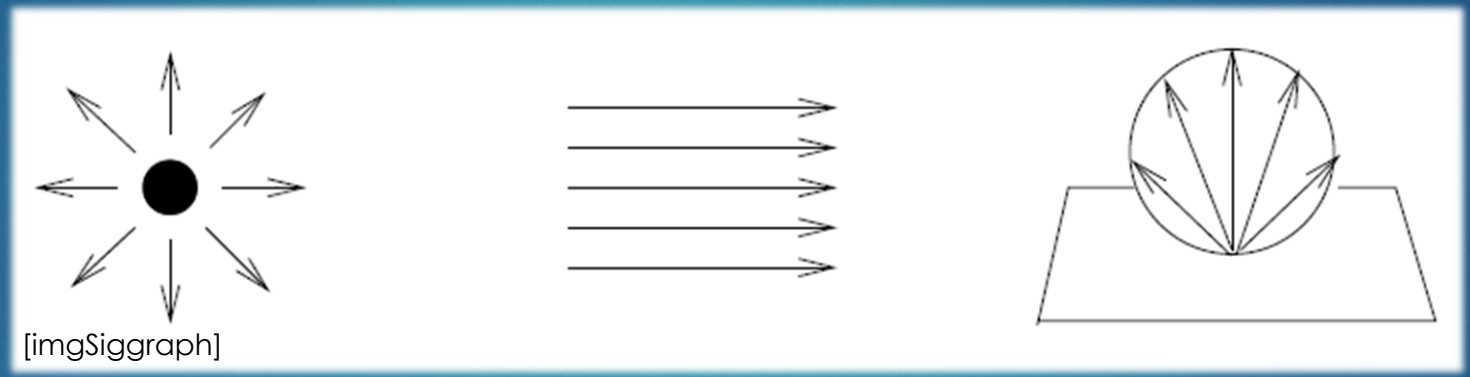
How are they generated?

1. Emit photons from the light sources
2. Trace photons until they reach a diffuse surface
3. Store them in a KD-Tree for fast access

Photon maps

Photon emission

- Photons are emitted from each light source
- Direction depends on the light type
  - Emission directions for point light, directional light and area light respectively



Photon maps

Photon emission

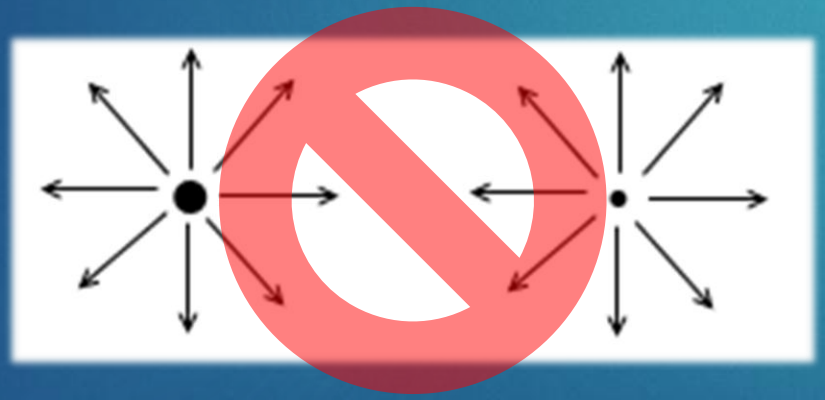
- Power of the photon
  - Information about the light source that emitted it
- Formula:
  - P_{photon} Power of each photon
  - P_{light} Power of the light
  - n_e Number of photons the light will emit

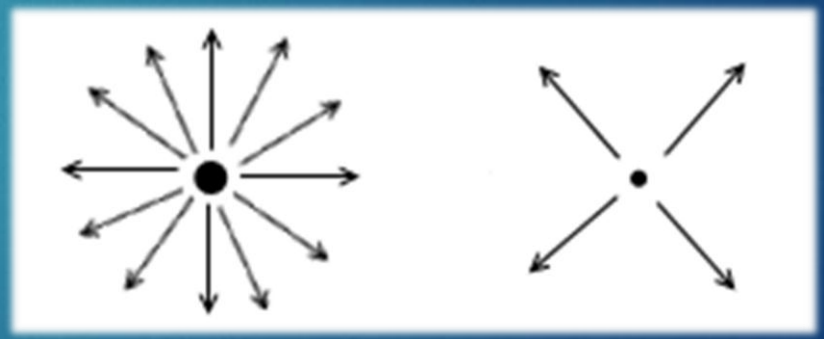
$$P_{photon} = \frac{P_{light}}{n_e}$$

Photon maps

Photon emission – Multiple lights

- We don't emit the same number of photons per light
- Lights with more intensity will emit more photons
  - Number of photons in the scene remains constant
  - Photons will have similar power
    - Makes the radiance estimate better





Photon maps

Photon emission – Multiple lights

	Light A	Light B	Total
Power	9	1	10
Contribution	90%	10%	100%
Number of photons	100	100	200
Photon power	9/100 = 0.09	1/100 = 0.01	10

	Light A	Light B	Total
Power	9	1	10
Contribution	90%	10%	100%
Number of photons	90	10	100
Photon power	9/90 = 0.1	1/10 = 0.1	10

Photon maps

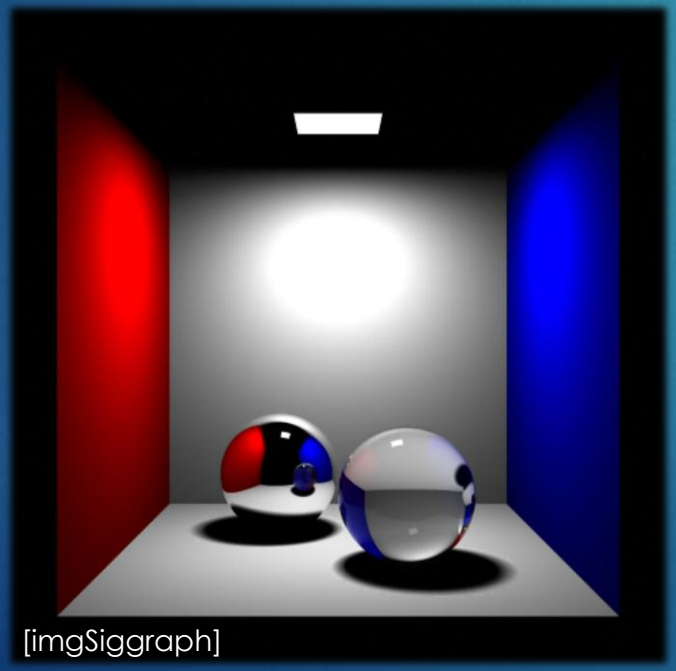
Photon tracing

- Each time we hit a surface we would need to generate 2 photons
  - Diffuse reflection
  - Specular reflection
- The 8^{th} bounce would generate $2^8 = 256$ photons (this is bad)
  - A lot of memory needed
  - Most photons will have very low power
- Solution: Russian roulette
  - Probabilistic approach

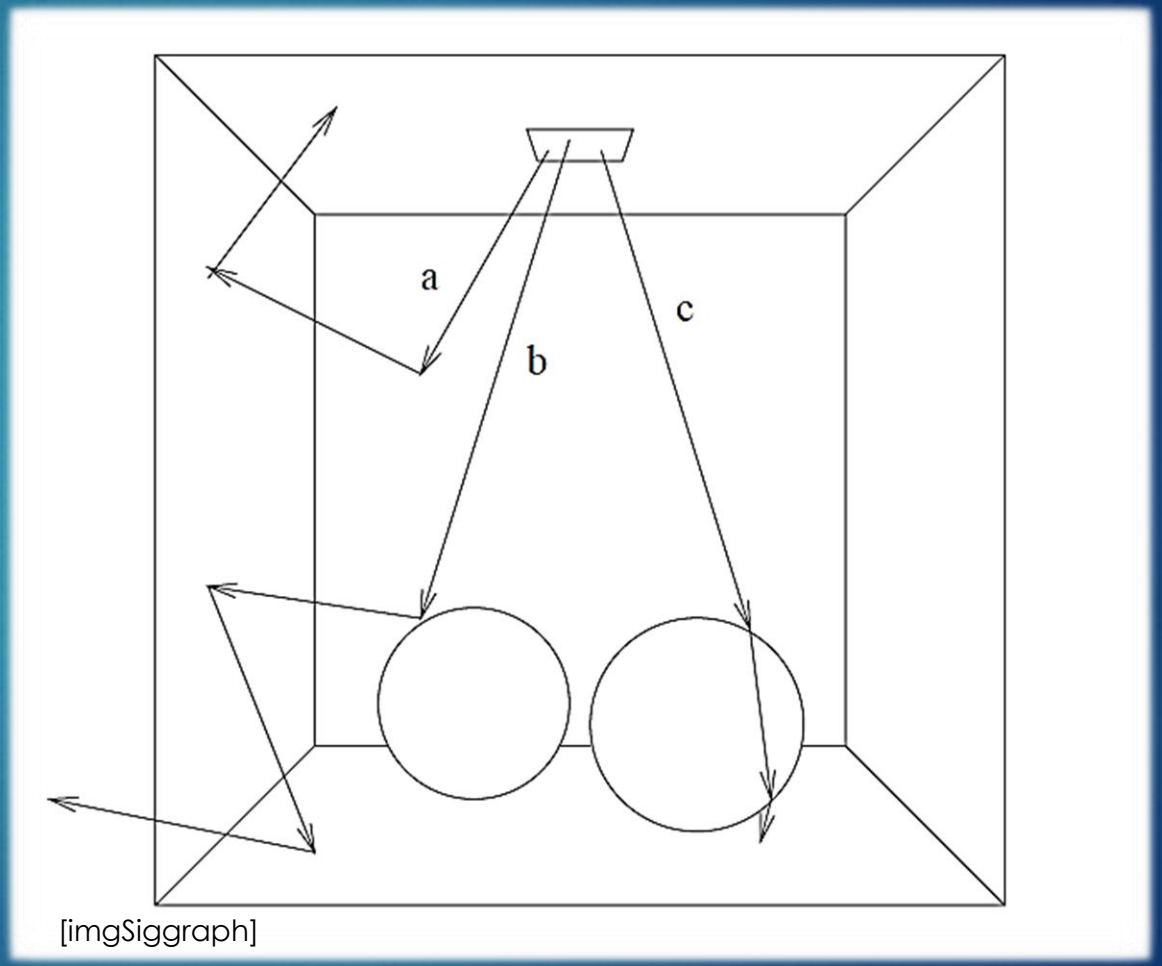
Photon maps

Photon tracing – Russian roulette

- Photon path examples:
  - a: x2 diffuse -> absortion
  - b: specular -> x2 diffuse
  - c: x2 specular -> absorption



[imgSiggraph]

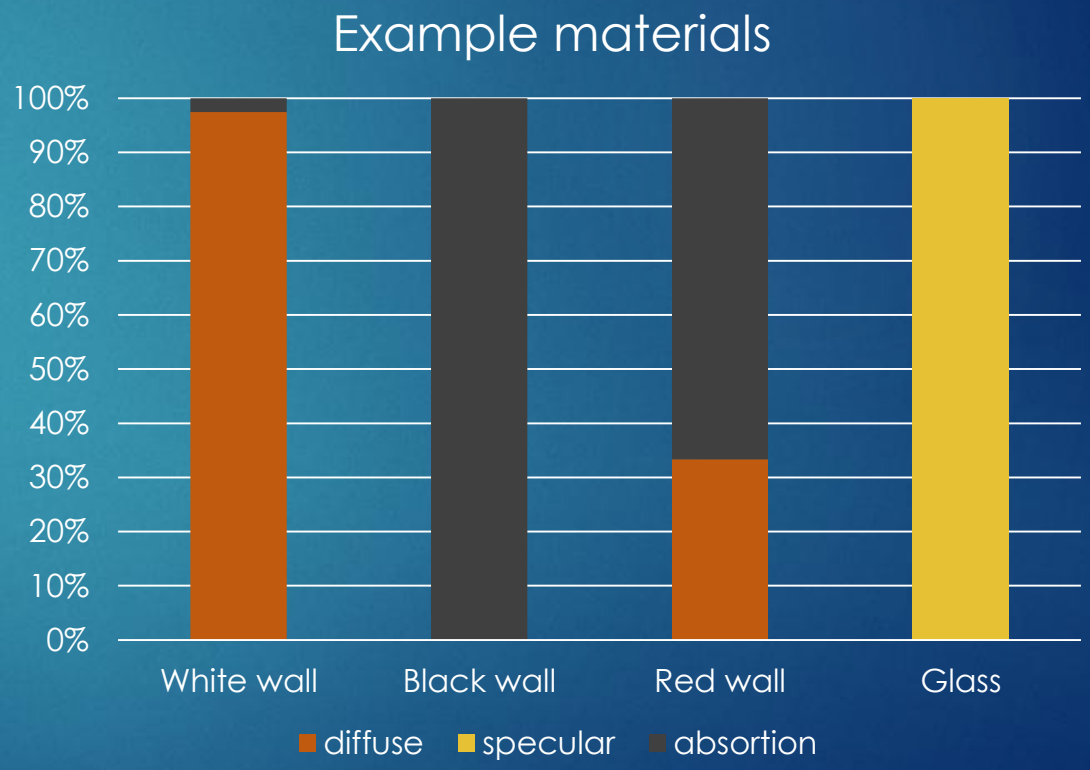


[imgSiggraph]

Photon maps

Photon tracing – Russian roulette

- Coefficients to determine the action of the photon
  - D Diffuse reflection coefficient
  - S Specular reflection coefficient
  - $D \geq 0,\ S \geq 0,\ D + S \leq 1$
- Take random x where $x \in [0, 1]$
  - $x \in [0, D] \rightarrow difuse\ reflection$
  - $x \in (D, D + S] \rightarrow specular\ reflection$
  - $x \in (D + S, 1] \rightarrow absortion$



Photon maps

Photon tracing – Russian roulette

- Compute coefficients out of RGB coefficients

$$P_r = \max(d_r + s_r, d_g + s_g, d_b + s_b)$$

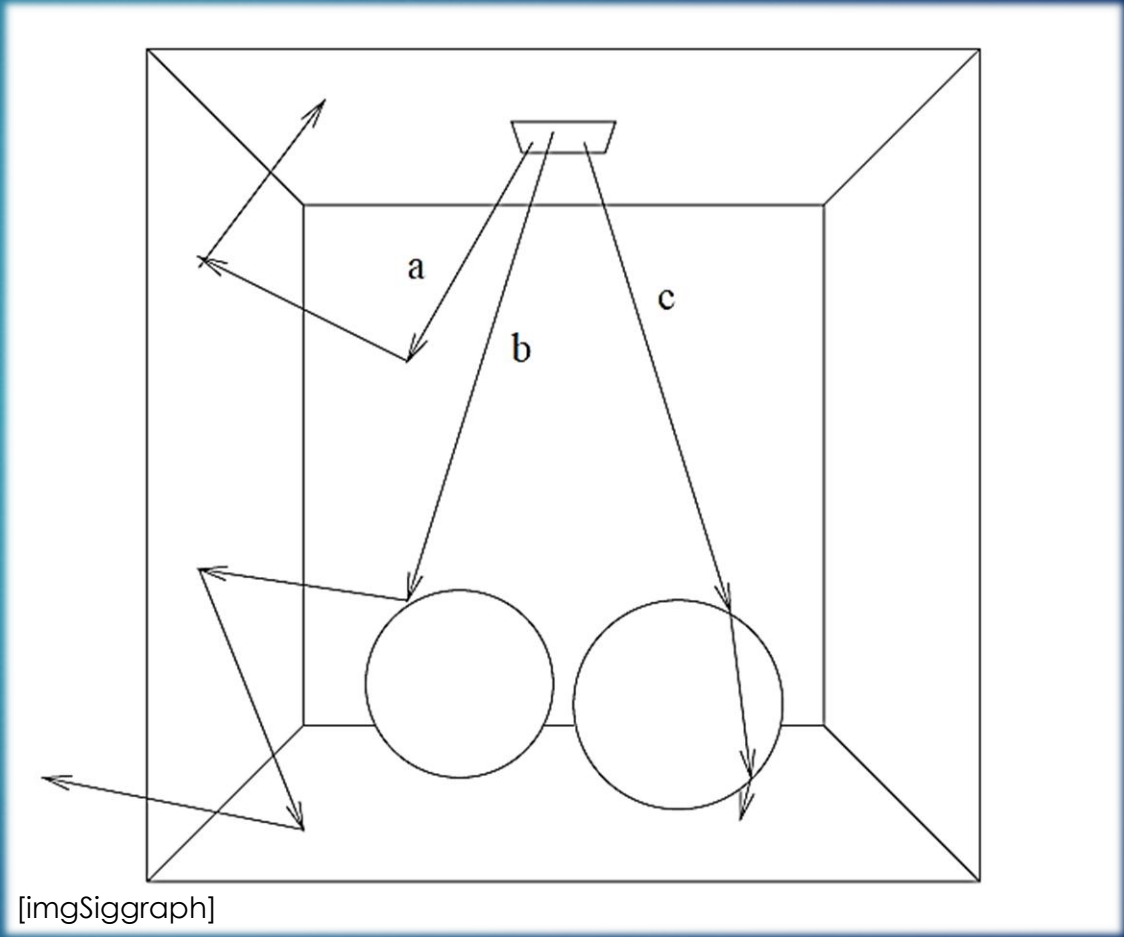
$$P_D = \frac{d_r + d_g + d_b}{d_r + d_g + d_b + s_r + s_g + s_b} P_r$$

$$P_S = \frac{s_r + s_g + s_b}{d_r + d_g + d_b + s_r + s_g + s_b} P_r = P_r - P_D$$

Photon maps

Photon tracing – Russian roulette

- Applying P_D and P_S
  - $x \in [0, P_D] \rightarrow$ difuse reflection
  - $x \in (P_D, P_D + P_S] \rightarrow specular\ reflection$
  - $x \in (P_D + P_S, 1] \rightarrow absortion$



[imgSiggraph]

Photon maps

Photon tracing – Russian roulette (Summary)

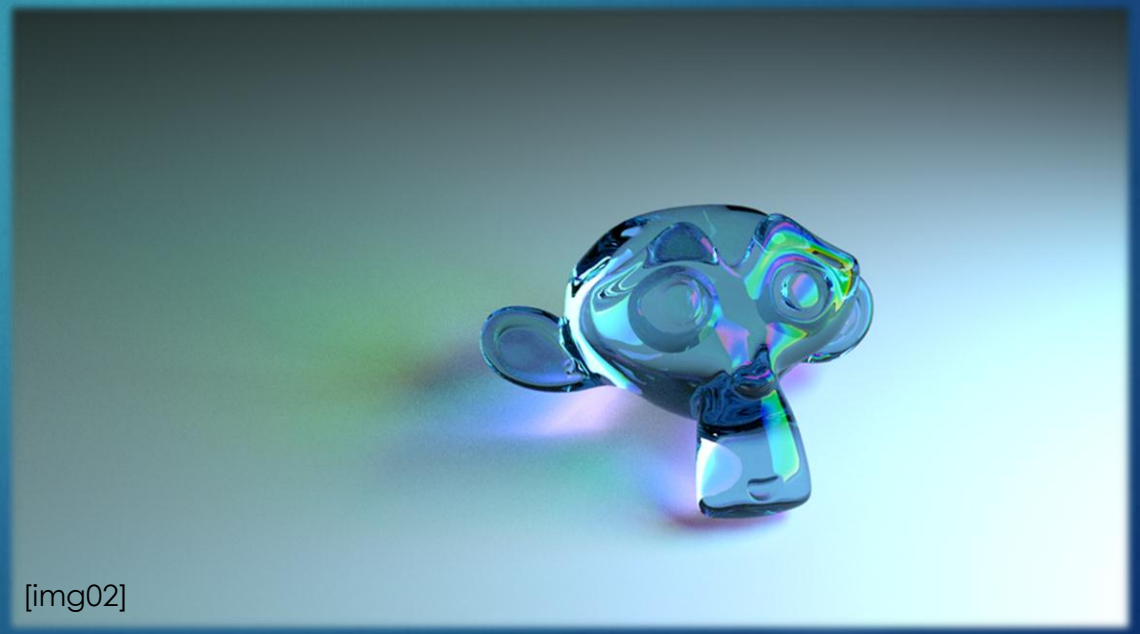
- Pros:
  - Reduces number of photon emission
    - Less computation & storage required
  - Keeps photon powers similar → Better radiance estimate
- Cons:
  - Increases variance on the solution, we need to emit a lot of photons
    - Not as many as without using Russian roulette

Photon maps

Photon tracing

- When a photon hits a surface we need to update its power
  - P_{inc} Power of the incident photon
  - P_{refl} Power of the reflected photon
  - S Specular color of the surface (if specular reflection happens)

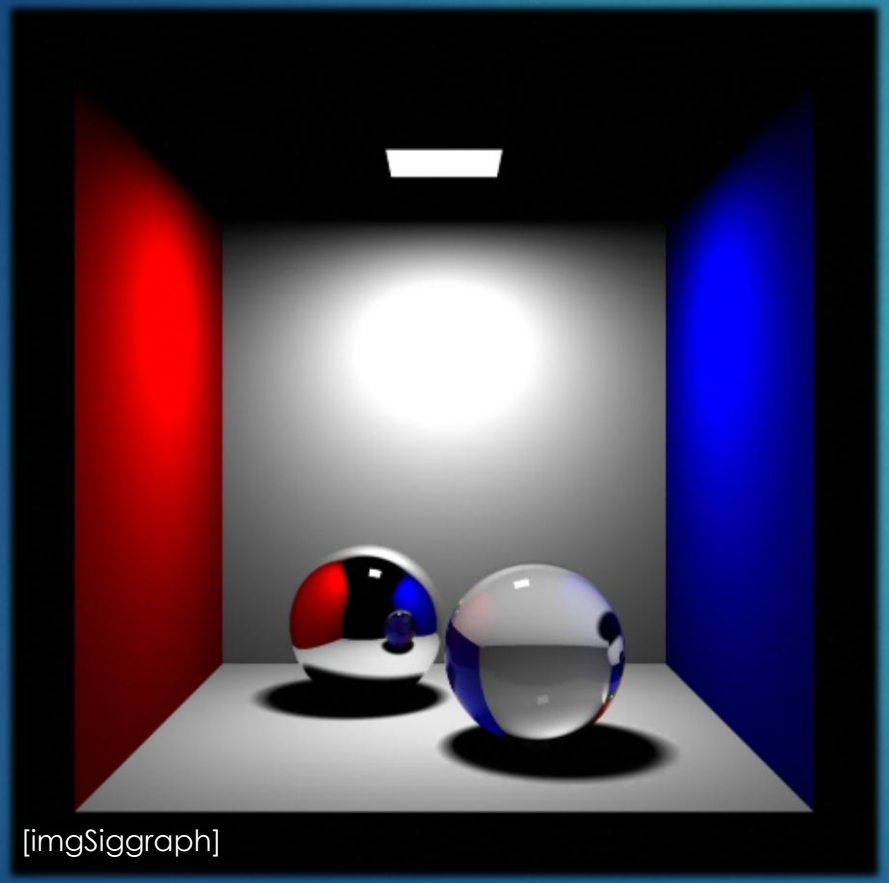
$$P_{refl} = \frac{P_{inc}\, S}{P_S}$$



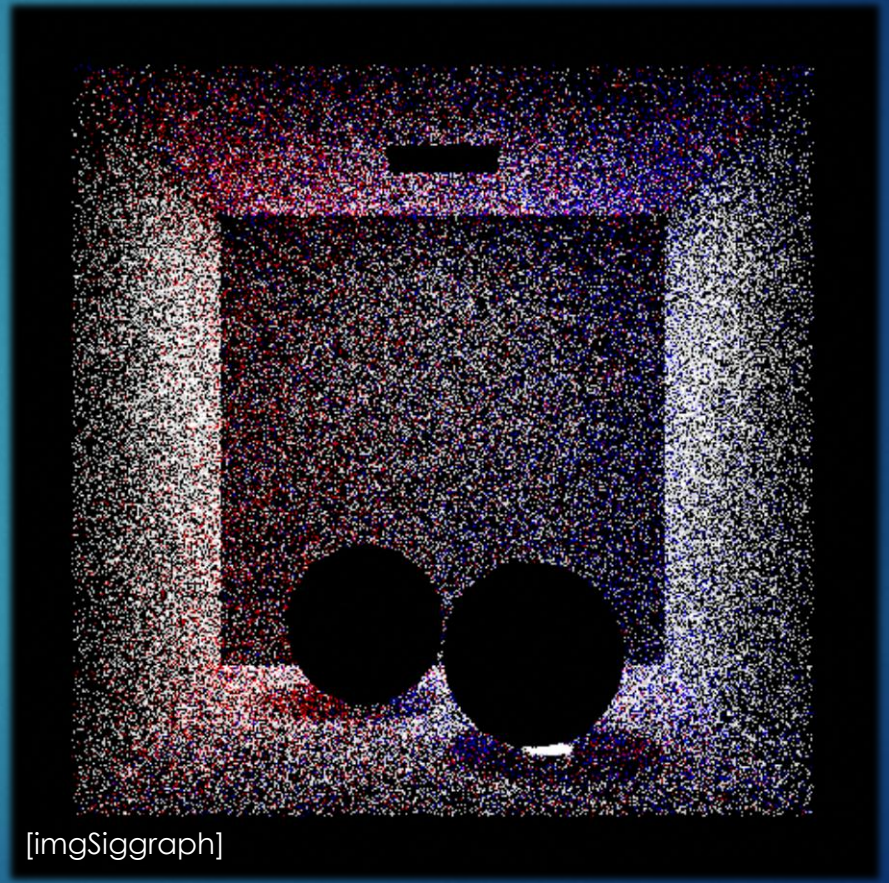
[img02]

Photon maps

Photon tracing



[imgSiggraph]



[imgSiggraph]

Photon maps

Photon storing – The photon

```
struct photon
{
   float x, y, z;      // position
   float r, g, b;      // power
   float dx, dy, dz;   // incident direction
};    // 36 bytes
```

Photon maps

Photon storing – The photon

- In case memory is a concern, compressed version

```
struct photon
{
   float x, y, z;   // position
   char p[4];       // power packed as 4 chars (Ward's RGBE)
   char phi, theta; // compressed incident direction
                    // (spherical coordinates)
   short flag;      // flag used in kd-tree
};    // 20 bytes
```

Photon maps

Photon storing – The photon

- Ward's RGBE packing
  - Same principle of mantissa and exponent in floating point values
  - Normalize RGB floats to chars, exponent gives us more precision
    - RGB → [0, 255] red, green, blue values
    - E → [0, 255] Exponent

Photon maps

Photon storing – The photon

- Incident direction packing
  - $2\ chars \rightarrow 16\ bits \rightarrow 2^{16} = 65536\ possible\ directions$

$$phi = 255 * \frac{atan2(dy, dx) + \pi}{2\pi}$$

$$[-\pi, \pi] \rightarrow [0, 2\pi] \rightarrow [0, 1] \rightarrow [0, 255]$$

$$theta = 255 * \frac{\text{acos}(dx)}{\pi}$$

$$[0, \pi] \rightarrow [0, 1] \rightarrow [0, 255]$$

Photon maps

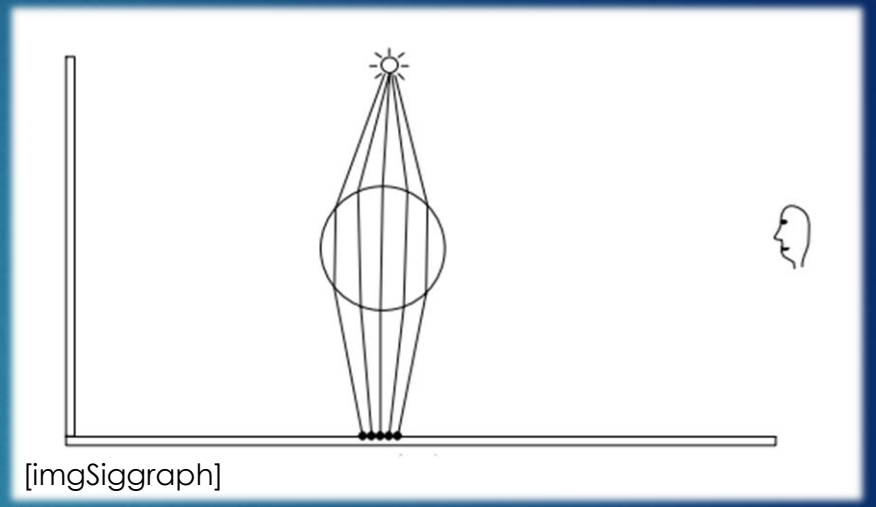
Photon storing – KD-tree

- Photons are stored when they hit diffuse surfaces
- Use a KD-Tree
  - $O(logN)$
  - Make sure is well balanced
    - Use an array to represent it

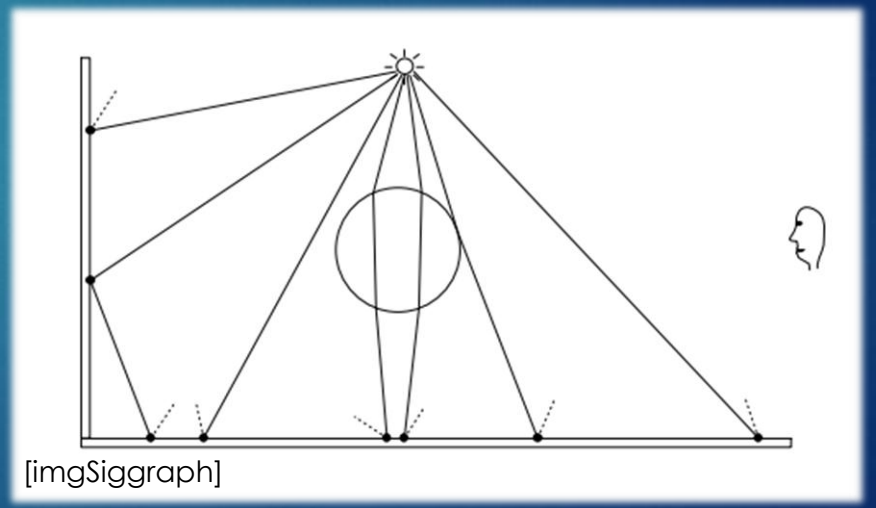
Photon maps

Photon storing – Three maps

- Photon map types:
  - Caustics photon map → Caustics
    - Photon went at least through an specular reflection
  - Global photon map → Global illumination
    - Photons that hit a diffuse surface
  - Volume photon map → Indirect illumination
    - Indirect illumination of a participating medium



[imgSiggraph]



[imgSiggraph]

Photon maps

Summary

- Emit photons from all light sources
- Let the photons bounce through the scene
  - Use Russian roulette to determine action
    - Diffuse reflection, specular reflection or absortion
- Store the photons in the corresponding photon maps

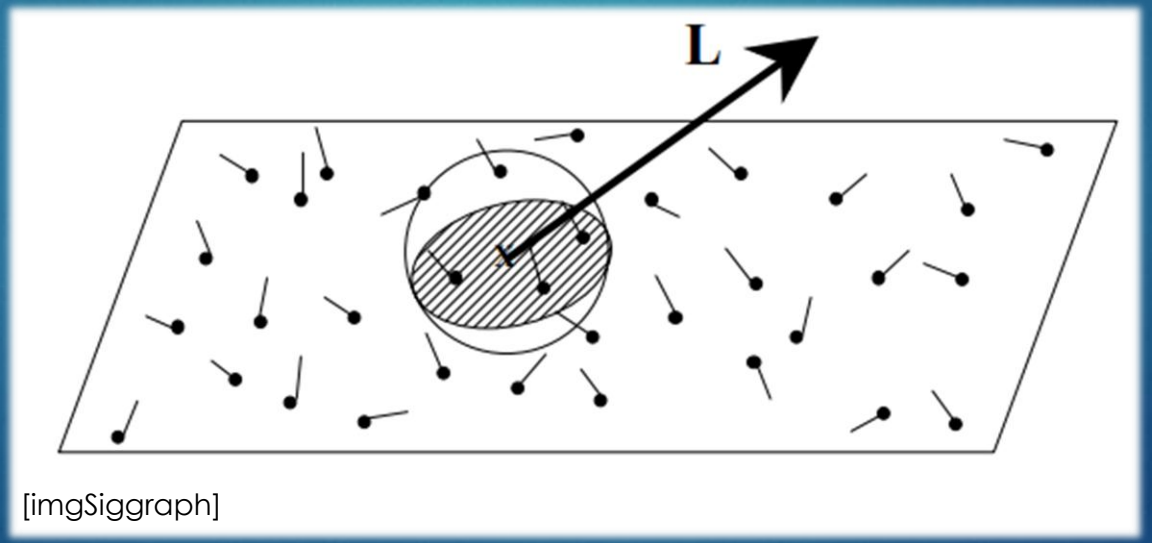
Rendering

Rendering

- Photon mapping is a 2 pass render technique
  1. Generate photon maps
  2. Use generated photon maps to get the radiance estimation on the surface

Rendering

- Use the generated photon maps to get the radiance estimation
  - Caustics photon map
  - Global photon map
  - Volume photon map
- Get nearest photons in the surface → KD-Tree traversal

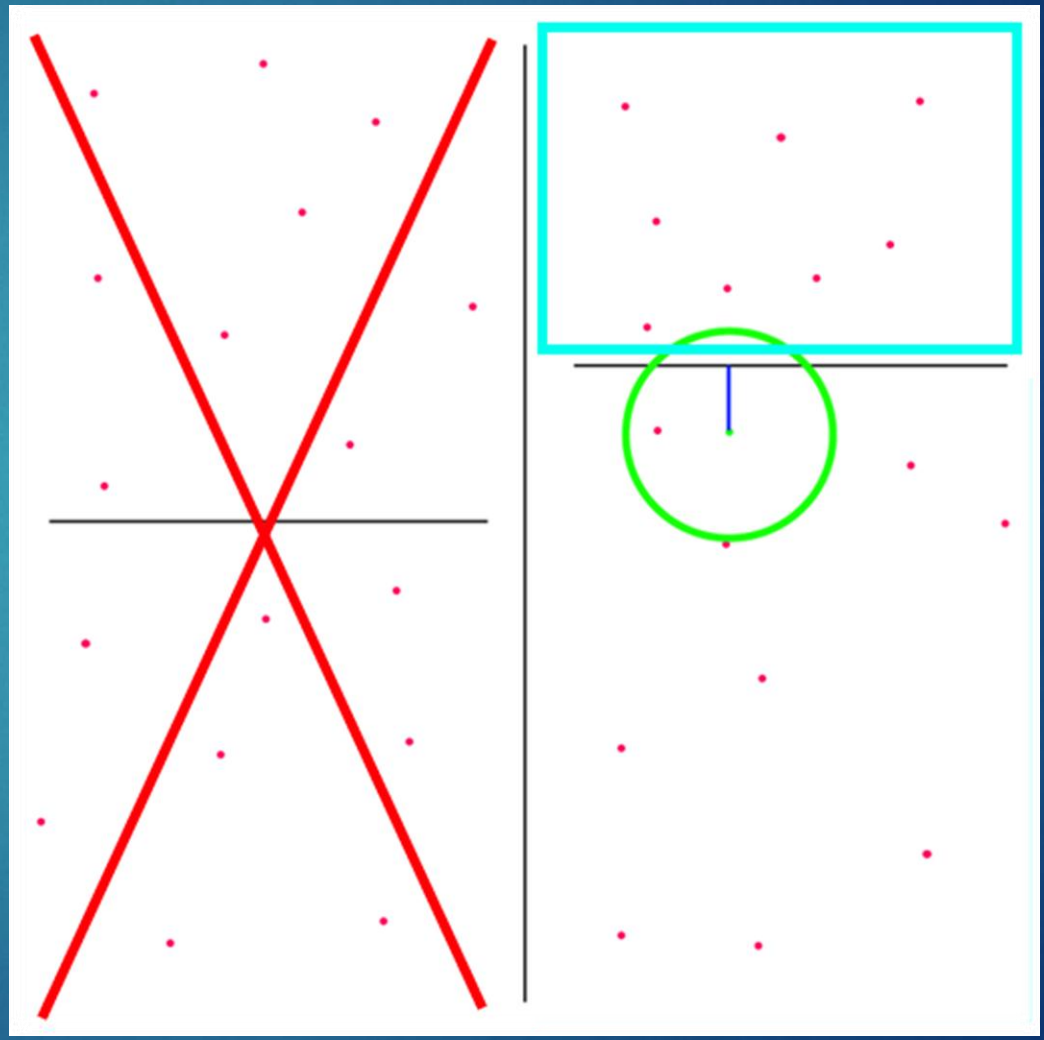


[imgSiggraph]

Rendering

KD-tree traversal

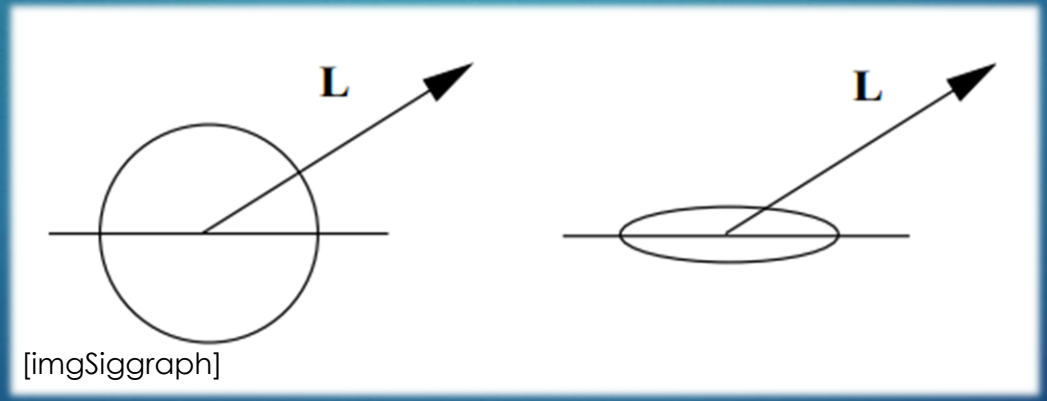
1. Check plane side
2. Traverse child node
3. Distance to plane < Threshold
    1. Traverse other child



Rendering

Sphere VS Disk

- Sphere
  - Fast to locate photons, just need squared distance
  - Can give wrong approximations on corners
- Disk
  - Slower
  - More accurate, takes surface into account



[imgSiggraph]

Rendering

Radiance estimate – Formula

- Outgoing = Emitted + Reflected
  - $L_o \rightarrow Outgoing\ radiance$
  - $L_e \rightarrow Emitted\ radiance$
  - $L_r \rightarrow Reflected\ radiance$
  - $x \rightarrow Surface\ position$
  - $\vec{w} \rightarrow Direction$ (Surface to camera)

$$L_o(x, \vec{w}) = L_e(x, \vec{w}) + L_r(x, \vec{w})$$

Rendering

Radiance estimate – Formula

- Reflected radiance is computed by integrating the incoming radiance
  - $f_r(x, \vec{w}', \vec{w}) \rightarrow Ilumination\ function\ (in\ my\ case\ phong)$
  - $L_i(x, \vec{w}') \rightarrow Incoming\ radiance$
  - $\Omega_x \rightarrow \text{hemi} - \text{sphere of incoming directions}$

$$L_r(x, \vec{w}) = \int_{\Omega_x} f_r(x, \vec{w}', \vec{w})\, L_i(x, \vec{w}')\, |\vec{n}_x \cdot \vec{w}'|\, dw'_i$$

Rendering

Radiance estimate – Formula

- Apply relation between the radiance and the flux
- The flux is represented by photons in the photon map
  - $N \rightarrow number\ of\ photons\ near\ x$

$$L_r(x, \vec{w}) = \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r\left(x, \overrightarrow{w_p}', \vec{w}\right) \Delta\Phi_p(x, \vec{w}_p)$$

Rendering

Radiance estimate – Formula

- Total incoming radiance $L_i(x, \vec{w})$ is the sum of all radiances
  - $L_{i,l}(x, \vec{w}) \rightarrow Direct\ ilumination$
  - $L_{i,c}(x, \vec{w}) \rightarrow Caustics$
  - $L_{i,d}(x, \vec{w}) \rightarrow Indirect\ illumination$

$$L_i(x, \vec{w}) = L_{i,l}(x, \vec{w}) + L_{i,c}(x, \vec{w}) + L_{i,d}(x, \vec{w})$$

Rendering

Radiance estimate – Formula

$$L_r(x, \vec{w}) = \int_{\Omega_x} f_r(x, \vec{w}', \vec{w})\, L_i(x, \vec{w}')\ \ |\vec{n}_x \cdot \vec{w}'|\, dw'_i$$

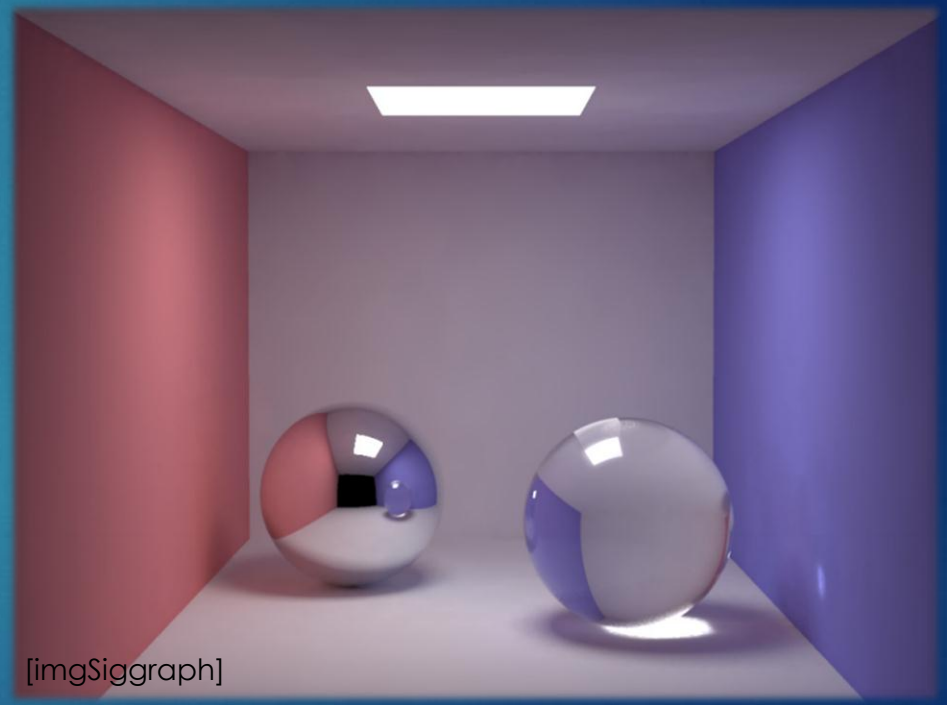
$$\frac{1}{\pi r^2} \sum_{p=1}^{N} f_r\left(x, \overrightarrow{w_p}', \vec{w}\right) \Delta\Phi_p(x, \vec{w}_p)$$

$$L_i(x, \vec{w}) = L_{i,l}(x, \vec{w}) + L_{i,c}(x, \vec{w}) + L_{i,d}(x, \vec{w})$$

Rendering

Radiance estimate - Filtering

- We average the flux in an area
- Special problem with Caustics
  - We want to preserve their sharp edges
- We use filtering to get better results
  - Increase weight of photons near the point of interest



[imgSiggraph]

Rendering

Radiance estimate - Filtering

- Gaussian filter
  - $d_p \rightarrow Distance\ from\ x\ to\ the\ photon$
  - $\alpha = 0.918$
  - $\beta = 1.953$

$$w_{pg} = \alpha \left[1 - \frac{1 - e^{-\beta \frac{d_p^2}{2r^2}}}{1 - e^{-\beta}} \right]$$

$$L_r(x, \vec{w}) = \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r\left(x, \vec{w_p}', \vec{w}\right) \Delta\Phi_p(x, \vec{w}_p)\, w_{pg}$$

Summary

Summary

- Photon mapping is a 2 pass rendering technique

1. Generate photon maps
   - Use Russian roulette (reflect or absorb)
2. Render scene
   - Use photon maps to get the radiance average
   - Apply filtering in case of caustics
     - Maintain sharp edges

Conclusion

Conclusion

Why photon mapping?

- Photon mapping is less expensive than Monte Carlo integration
  - Russian roulette
    - Probabilistic approach to reduce the number of photons
  - Number of lights does not affect photon number
  - While rendering just need to find the nearest photons
    - KD-Tree → $O(logN)$

Credits

- All the formulas and theoretical references of this presentation are from "Siggraph 2000 - A Practical Guide to Global Illumination using Photon Maps" (Order and arrangement of contents may not be the same)
  - https://graphics.stanford.edu/courses/cs348b-00/course8.pdf
- [imgSiggraph]: image taken from the document above
- [gif00]: Gif I created using giphy.com of the Bioshock Infinite game
  - https://giphy.com/gifs/bioshock-infinite-zrqsuyczat4U8/
- [gif01]: https://opengameart.org/content/water-caustics-effect-small
- [img00]: https://commons.wikimedia.org/wiki/File:Caustics.jpg
- [img01]: https://www.flickr.com/photos/snogglethorpe/2612616268
- [img02]: http://therealrichard.deviantart.com/art/Caustics-and-Coloured-Emission-277601210

Thanks for listening!

Questions?