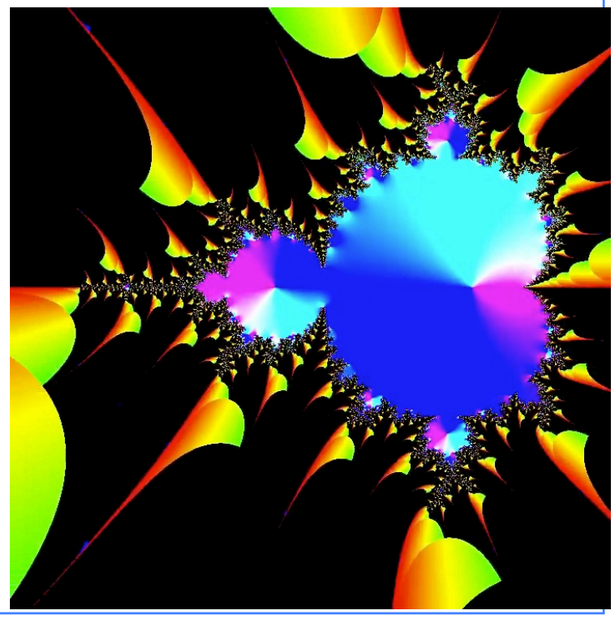
TSBK07/TSBK11

Computer Graphics

Ingemar Ragnemalm, ISY



Lecture 14

The final cut!

Advanced ray-tracing, global illumination

Low-level graphics (Bresenham etc)

GLES, WebGL, Vulkan

Lecture questions

1. What is the difference between photon mapping and path tracing?

2. What is the optimization done by Bresenham's line drawing algorithm?

3. What is the difference between the odd-even rule and the non-zero winding number rule?

4. When do you need to be careful with where the "hotspot" of a pixel is located?

Project specifications

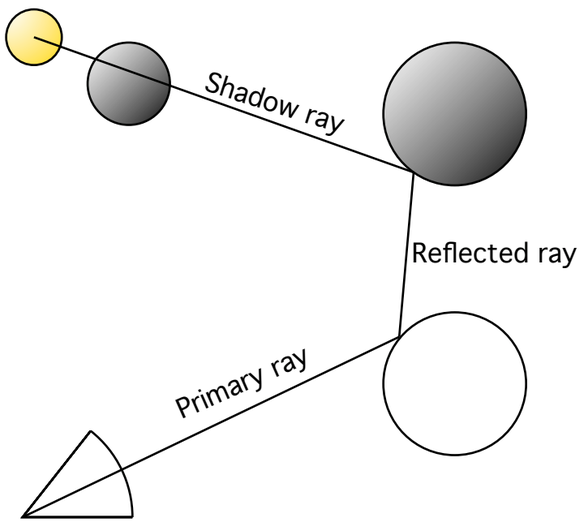
All who have proposed a project should have had a reply.

Something that looks like a final version should be submitted today.

Ray-tracing

Trace rays from the camera. Reflect, refract.

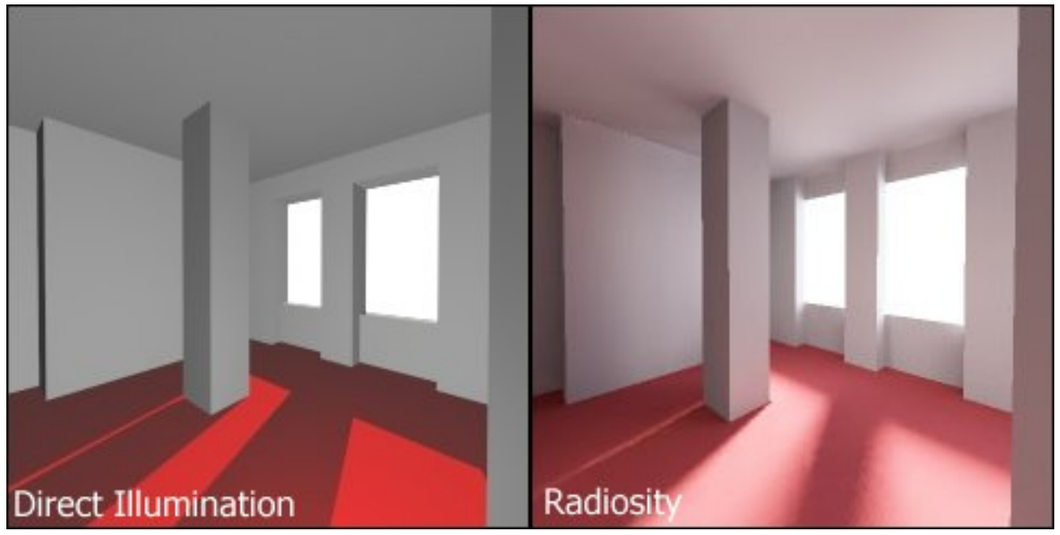
Shadow rays check visibility from surface to light source.



Only direct light can be handled this way!

Radiosity

A method for high-quality rendering of scenes with diffuse reflections and soft shadows.



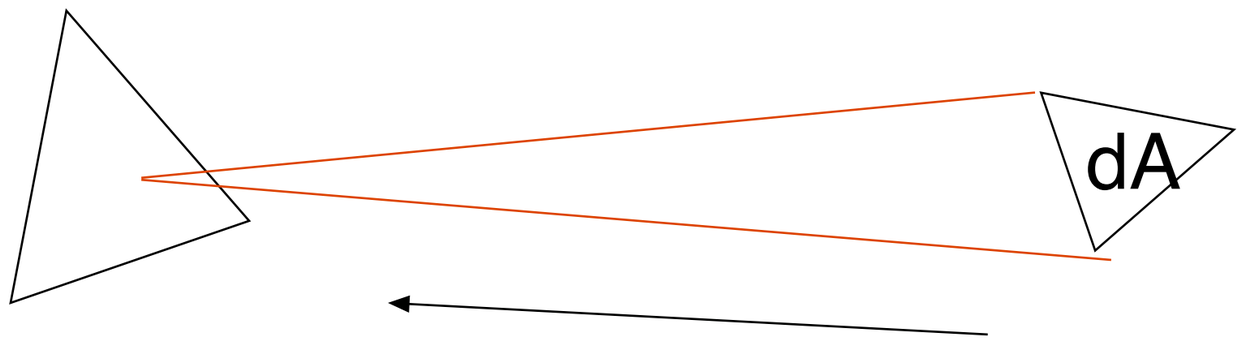
(image from Wikipedia)

Radiosity method

Models indirect light from surface to surface. Only diffuse surfaces.

Incoming light is summed to outgoing. Amount of light based on the solid angle, the form factor.

Accumulate, repeat to approach correct light.



Emission * form factor

The model

A surface emits energy that is a sum of reflected and emitted energy:

Energy * area = emitted + reflected

$$B_k * dA_k = E_k * dA_k + R_k * \int B_j * F_{kj} * dA_k$$

Emitted light (true light sources only)

Form factor between j and k

Outgoing energy from the surface element j

Reflectivity

Simplified to discrete patches

$$B_k = E_k + R_k * \sum B_j * F_{jk}$$

=> Equation system!

1) Solve equation system! Can be done incrementally.

2) We must interpolate between patches (e.g. Gouraud shading)

3) The form factors F_{jk} must be calculated

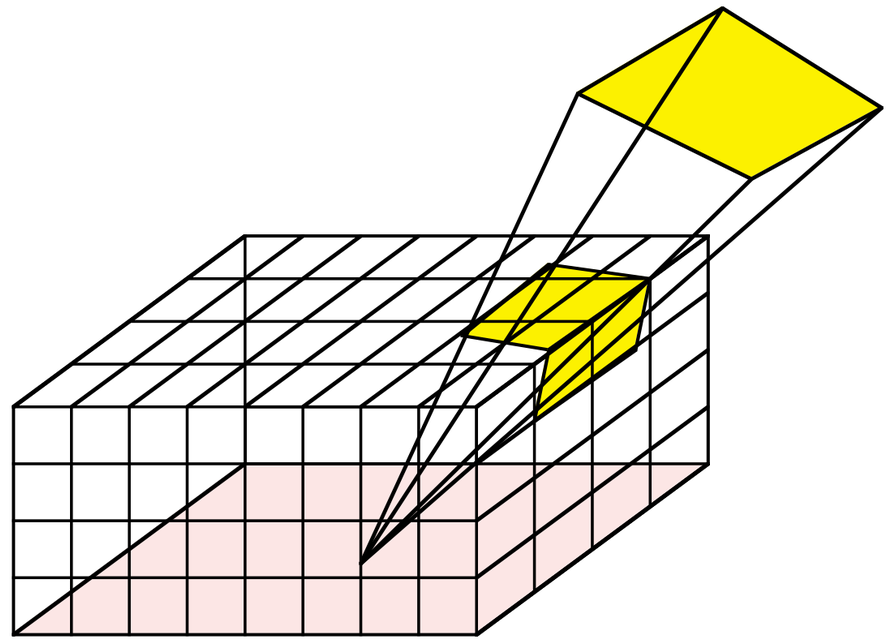
Calculating F_{jk}

Major problem of radiosity! F_{jk} calculations take most of the processing time!

F_{jk} = (Energy directly from j to k) / (Total energy from j)

This is calculated from the positions, angles and sizes of the two surfaces. All surfaces must be subdivided in parts. More parts give higher realism!

Hemicube for form factor calculation



Approximates f_{jk} by calculating projections

Progressive refinement radiosity

Step-wise refinement

A method for solving the equation system in real time

Make one step of emission at a time.

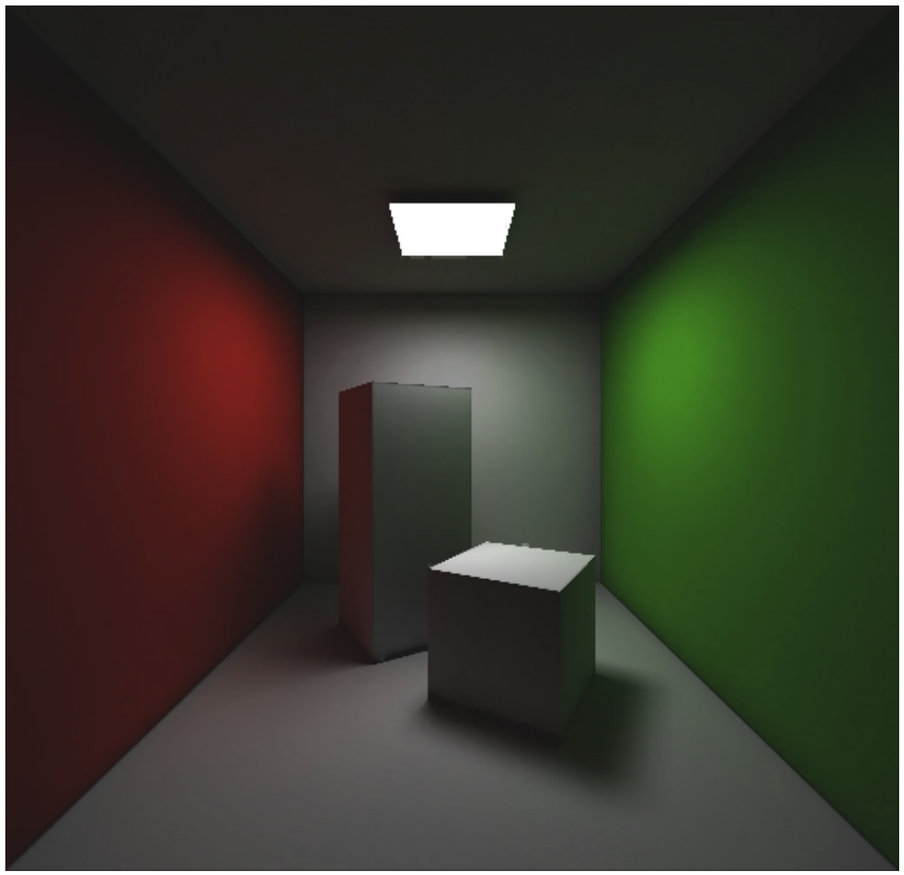
Exact solution takes an infinite number of iterations, but a good approximation is found after a few steps.

Preview can be shown instantly!

Example

TSBK03 project on progressive refinement

(Variant of Cornell Box, standard scene for illumination)



More global illumination models

Very important problem, much research!

- Photon mapping
- Path tracing
- Approximation by proximity measurements
- Hybrid models
- Many variants and parallel implementations

Photon mapping

"Backwards ray-tracing"

Applies ”backwards” light (from light sources) to measure how light is scattered over a scene

Gives a good measure of indirect light

Photon Mapping

Follow rays from light source with ray-tracing methods

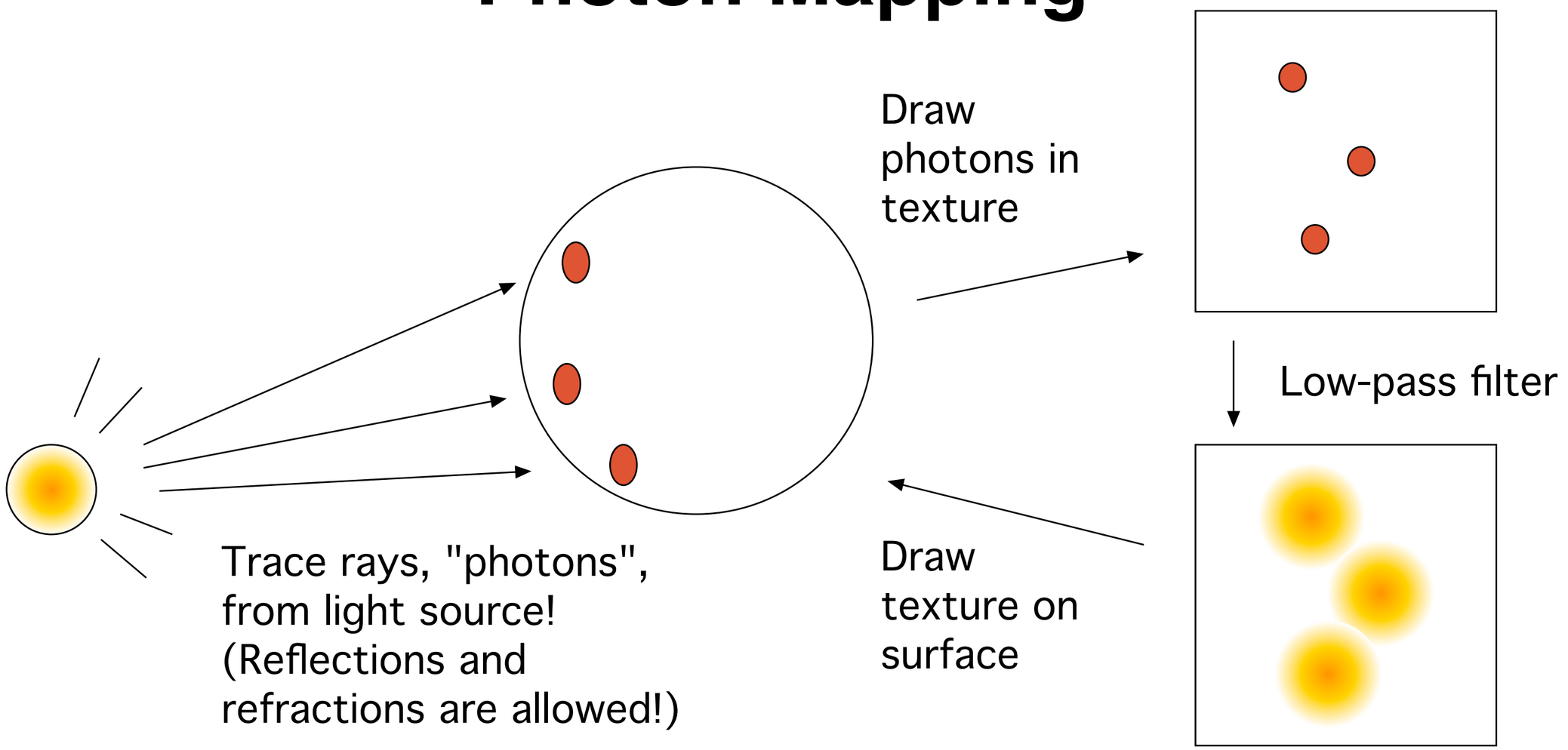
Accumulate light in ”photon maps”

Saves information about every photon - allows specular surfaces!

Low-pass filter

Then render scene using these maps as surfaces.
Handles both diffuse and specular reflections!

Photon Mapping

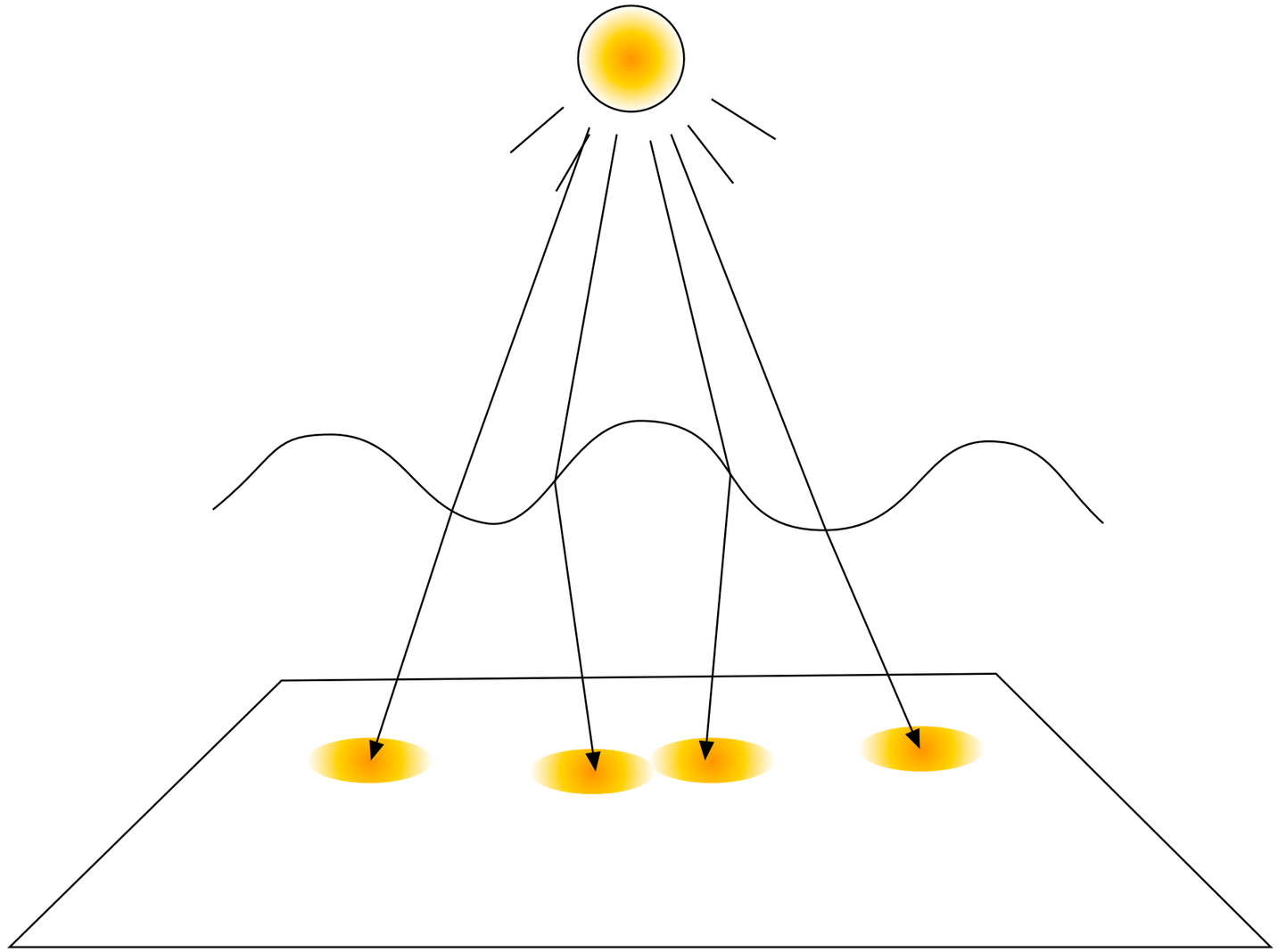


Caustics

Nice case for photon mapping

Bottom of lake only surface to illuminate

(Harder if the lake bottom is not flat or if there are several objects to illuminate.)



Photon Mapping Example

(The Schindler demo)

Typical features: Reflections, caustics and diffuse shadows

IMAGE REMOVED

Path tracing

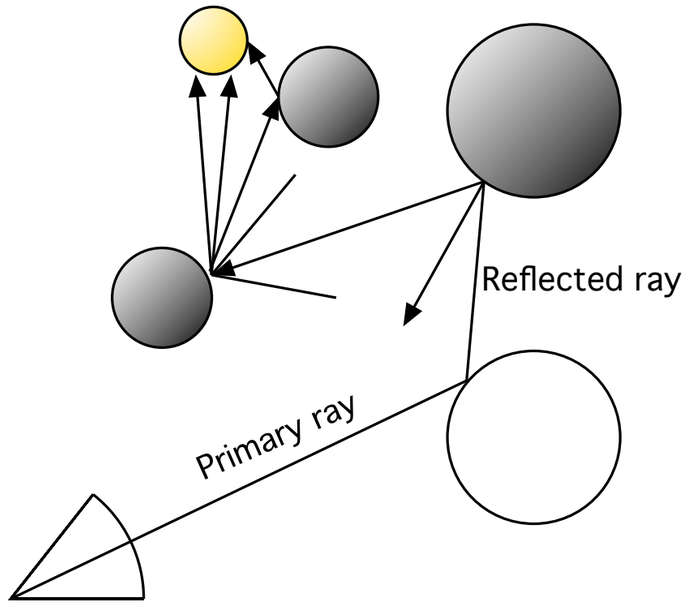
Similar to ray-tracing, but more rays.

Cast rays to search for "light paths".

Essentially cast multiple "shadow rays" to find indirect light.

"Backwards" path tracing

When hitting that opaque surface, cast many "shadow rays" recursively and see if they hit a light source! Search for light paths!



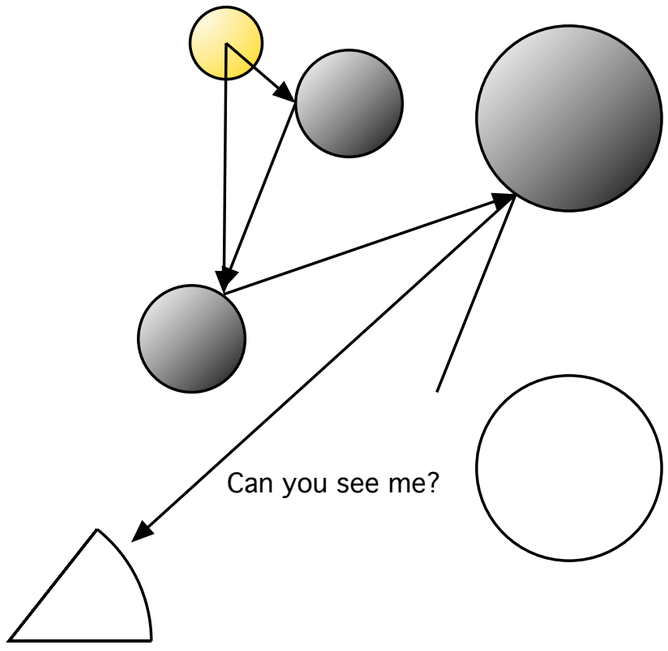
"Forwards" path tracing

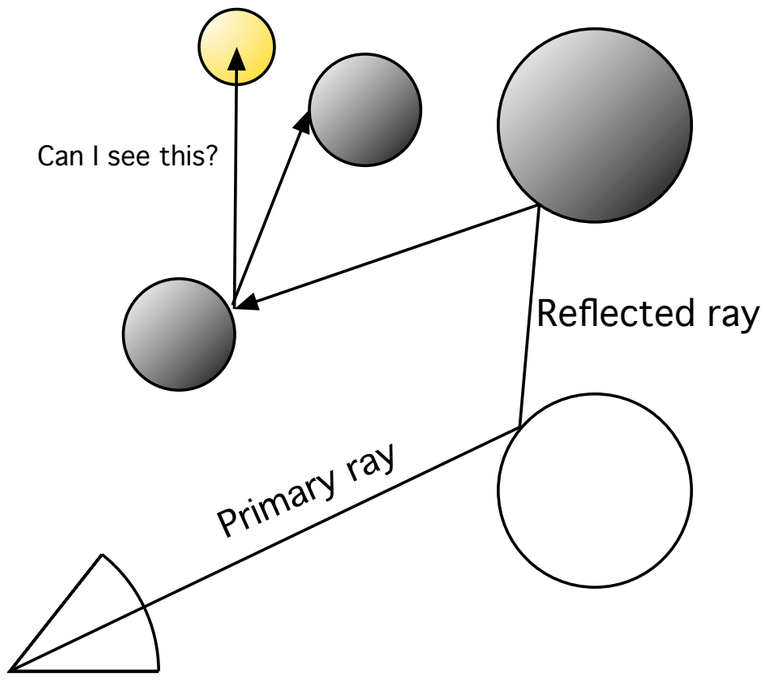
Instead of going from the camera, go from the light source, trace until you hit the camera.

Both cases can use traditional "shadow rays" to detect that a surface has direct light or a direct path to the camera.

Shadow rays still relevant

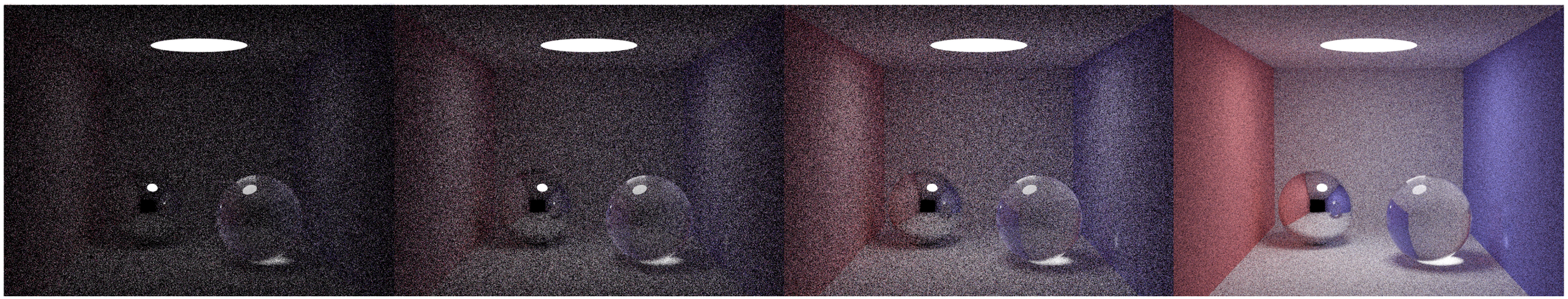
Both cases use traditional "shadow rays" to detect that a surface has direct light or a direct path to the camera.





Results (using Kevin Beason's "smallpt")

Small path tracer available on-line.



4 spp (21s) | 8 spp (42s) | 16 spp | 64 spp

We need *many* rays to get a good result!
Beason reports thousands!
Parallel computing vital!

Beason's path tracer

Casts multiple primary rays. Secondary rays make random bounces but never multiply!

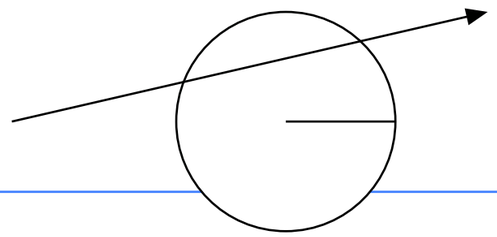
Uses shadow rays.

Uses light models for more realistic light.

Beason likes spheres

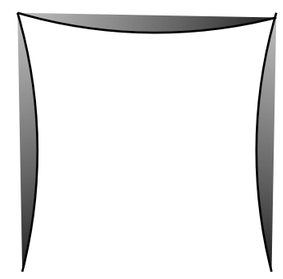
All parts of the scene are spheres! The walls are large spheres!

```
double intersect(const Ray &r)
const
{
    // returns distance, 0 if nohit
    Vec op = p-r.o;
    // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
    double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
    if (det<0)
        return 0;
    else
        det=sqrt(det);
    return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
}
```



As in the book but differently expressed.

```
// Scene description. Even the walls are spheres!
Sphere spheres[] =
{
    //Scene: radius, position, emission, color, material
    Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
    Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
    Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
    Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
    Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
    Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
    Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
    Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
    Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF)
    //Lite
};
```

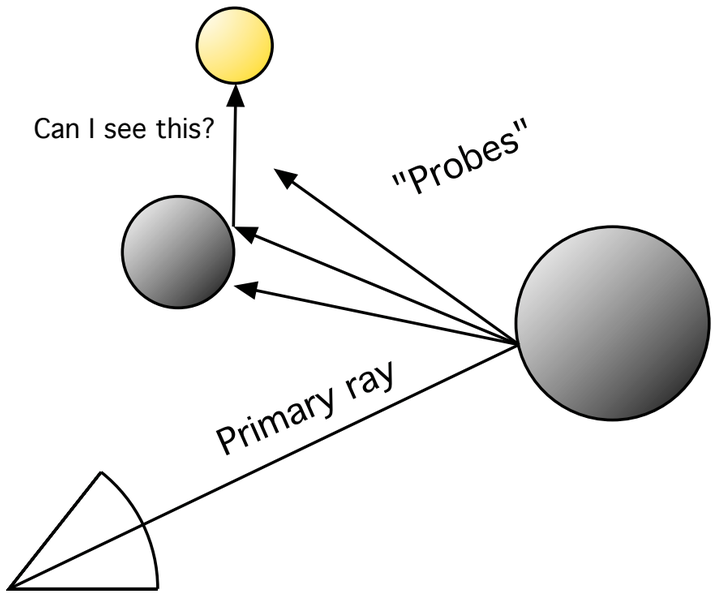


Path tracing in fragment shader

Hours to produce a noisy image? A bit better with openmp...?

But how about doing it on the GPU?

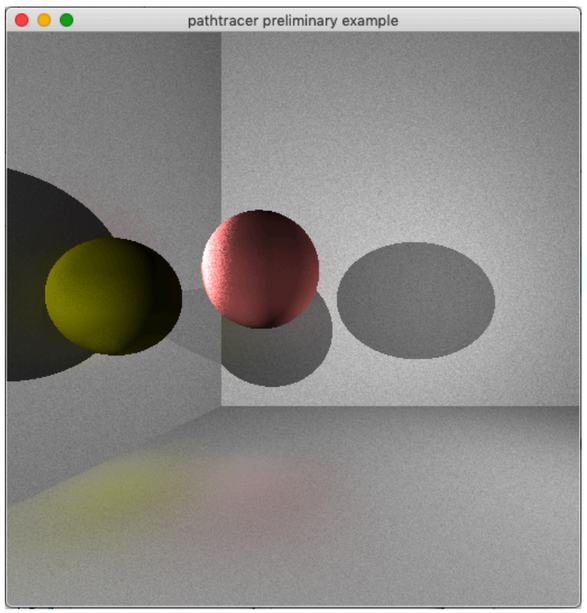
My first attempts: Simple path tracer:

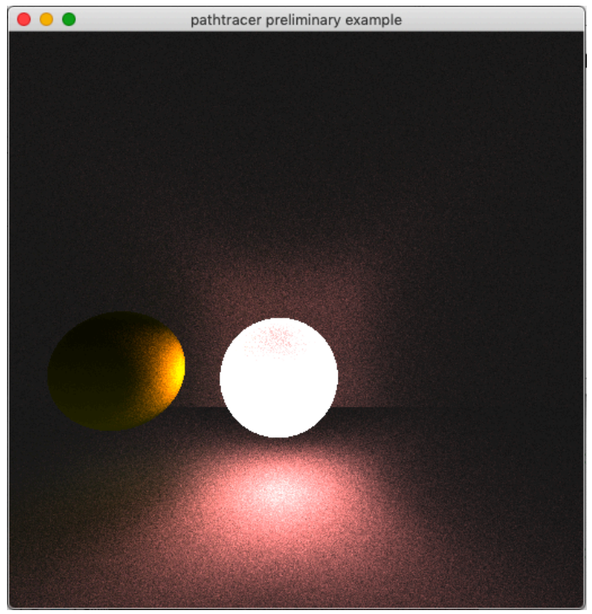


My simplified path tracer

Two variants: One with point lights, one with only emissive objects.

Pretty OK in real time (or almost) on my 650M. Cheating?





Summary

Ray-tracing:

Searches for surfaces and direct light. Good for shiny surfaces, transparency etc. “Hard” images.

Radiosity:

Models light patch to patch. Good for realistic images of diffuse surfaces. Can not handle specular reflections!

Photon Mapping: accumulates light in surfaces

Path tracing: searches in multiple directions for light paths