Lecture 18:
Primer on Ray Tracing Techniques

6.172: Performance Engineering of Software Systems

Joshua Slocum

November 16, 2010

A Little Background

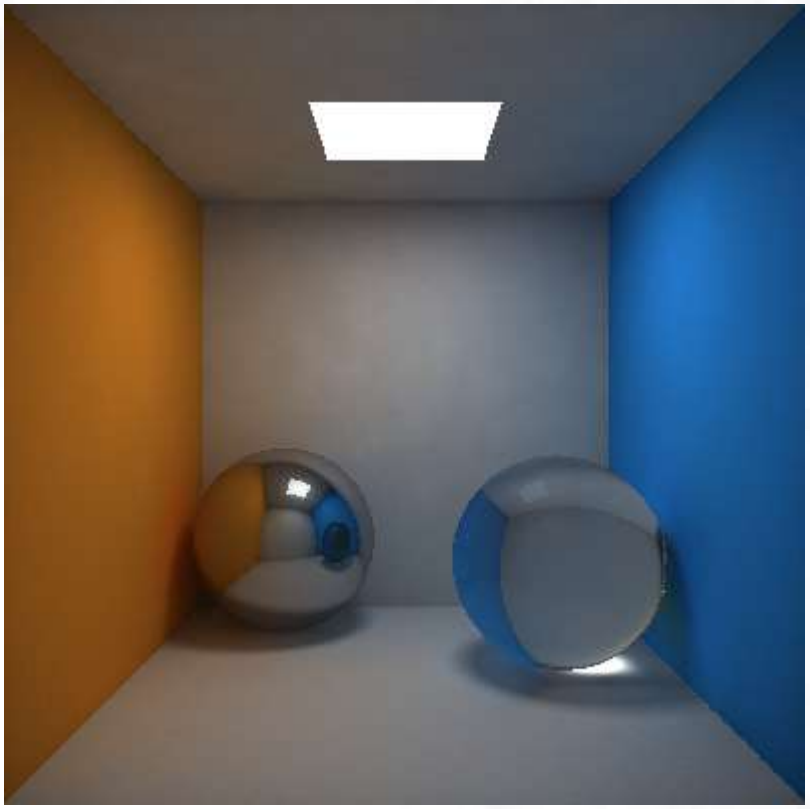
- Image rendering technique
- Simulate rays of light - “ray casting”
- Capacity for photorealism
- “Embarrassingly” parallel
- Your final project!

This photograph is in the public domain, and available here: http://en.wikipedia.org/wiki/File:Glasses_800_edit.png.

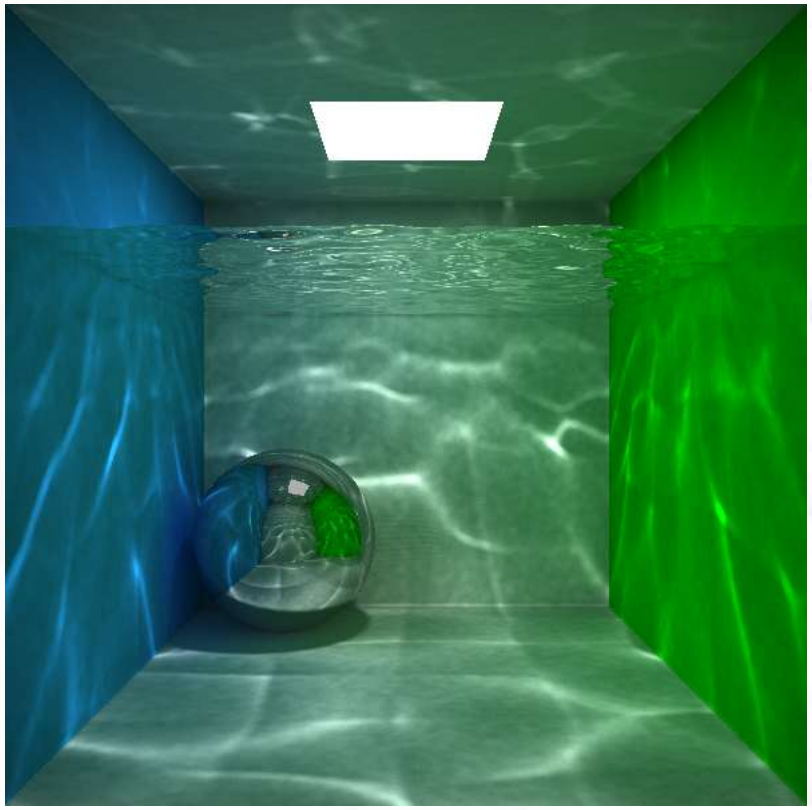
The Final Project

- Groups of 2-3
- We give you a working ray tracer; you make it go fast
- Lots of opportunities for optimization
  - Extreme parallelism
  - Algorithmic improvements
  - And others

Two Built-in Scenes



./raytracer -s1



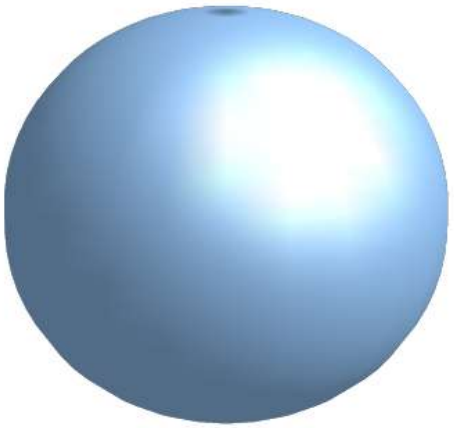
./raytracer -s2

What You Can't Do

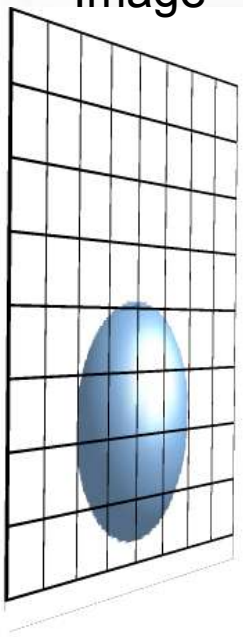
- Change the parameters defined in config.h
  - We will write over it when testing
- Compromise the functionality of the raytracer.
  - We may render other scene with your raytracer to make sure it still works properly
- Change the behavior of the code based on command line arguments

Rendering

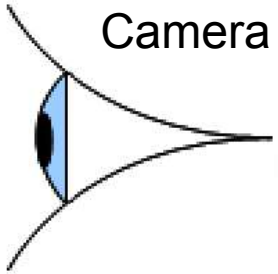
- Model a scene with objects
- Project the scene onto pixel grid



Scene



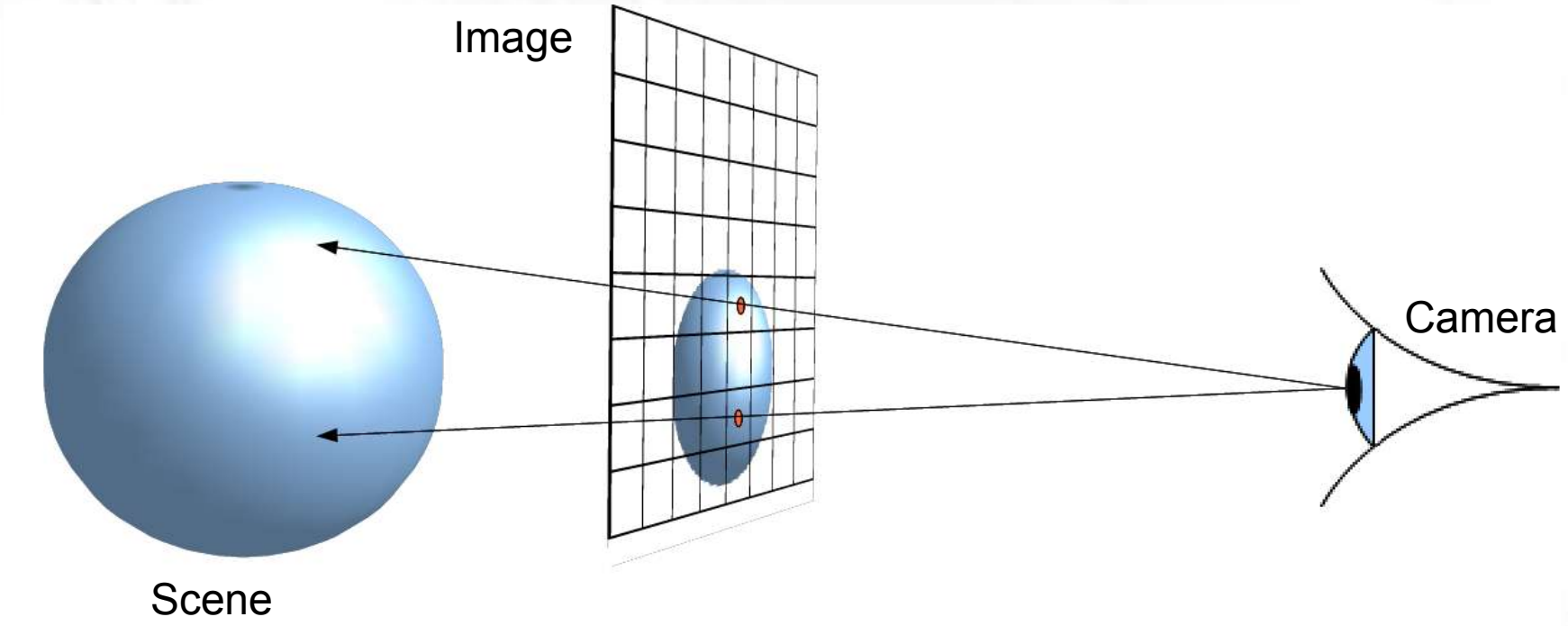
Image



Camera

Ray Casting

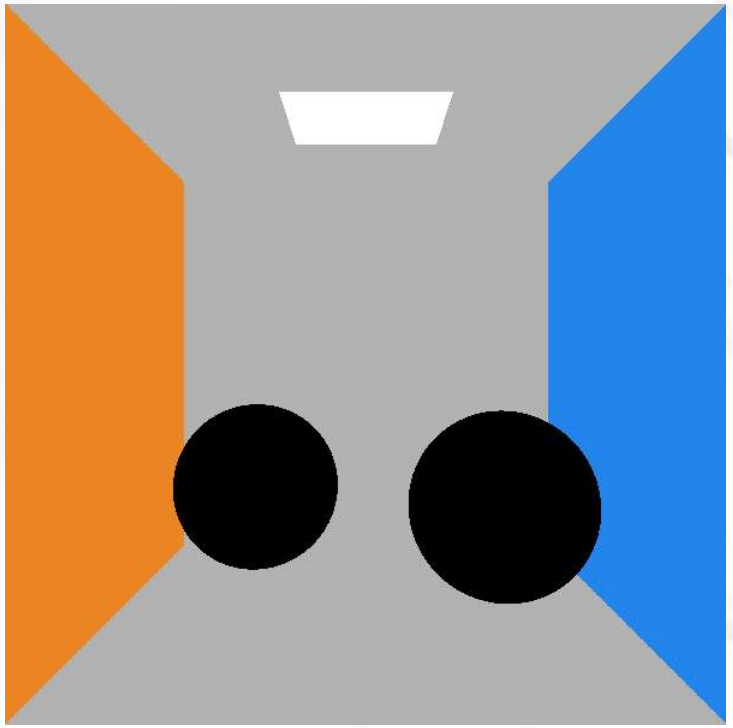
- Cast rays from the camera through each pixel
- Find intersection with the closest object
- Shade pixel with object's color



Ray casting

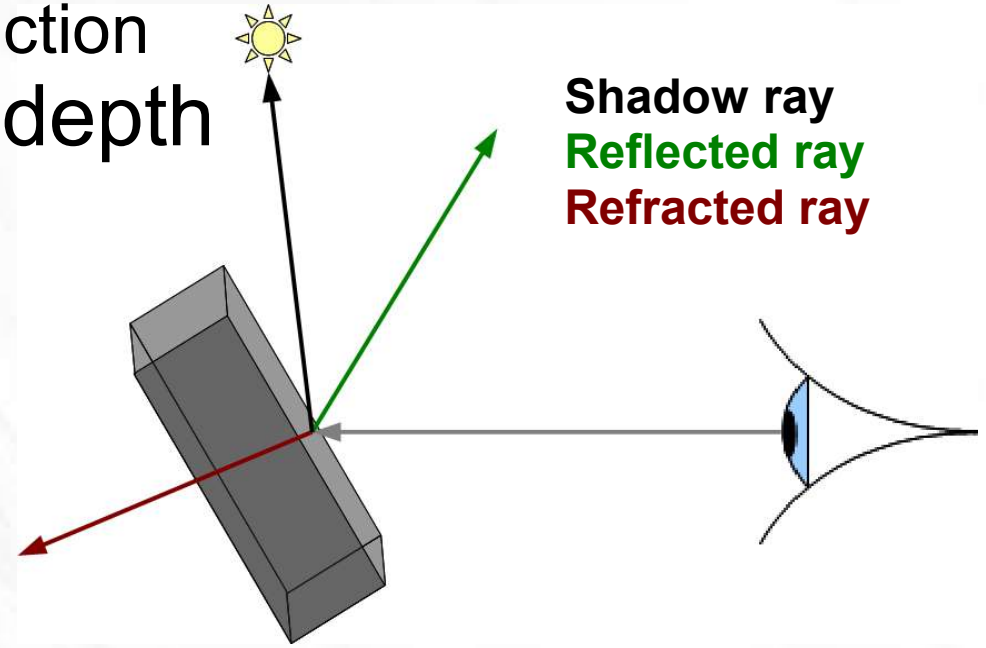
Very little detail

- Illumination is constant
- No reflections or refractions



Recursive casting: ray tracing

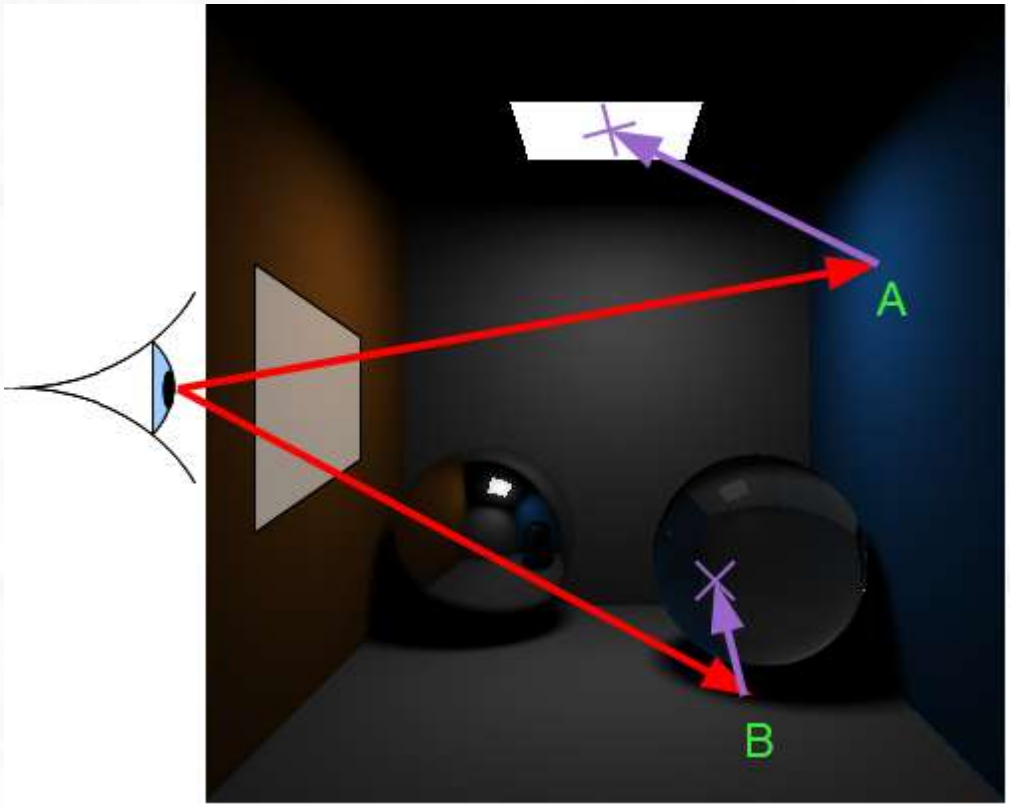
- Improved shading technique
- Recursively cast rays to simulate optical effects:
  - Cast rays at light sources to detect shadows
  - Trace rays to see reflection
  - Trace rays to see refraction
- Must limit recursion depth



Improved lighting

Calculate illumination of diffuse surfaces based on distance from light sources

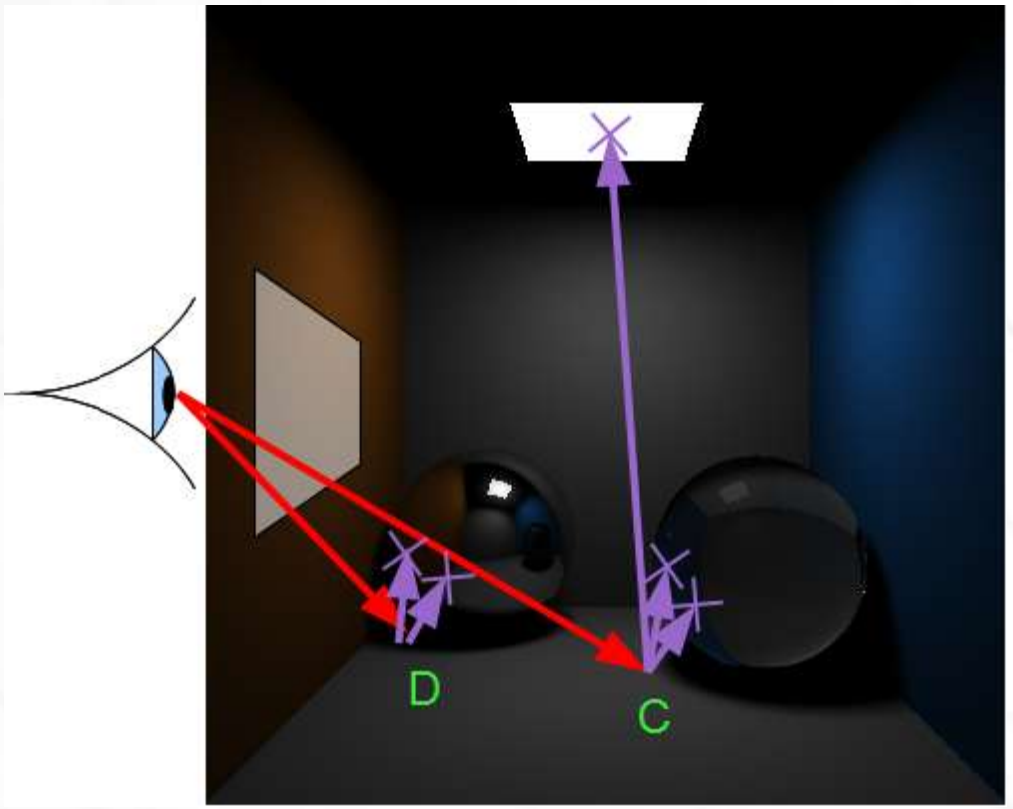
- Simulate effects of direct illumination on *diffuse* materials
- Cast a shadow ray from intersection point to each light source
- Amount of illumination based on relative position of light and intersection
- If a shadow ray intersects an object before reaching the light source, no illumination from that source (e.x. point B)



Soft shadows

Shadows from area lights are “soft” on the edges

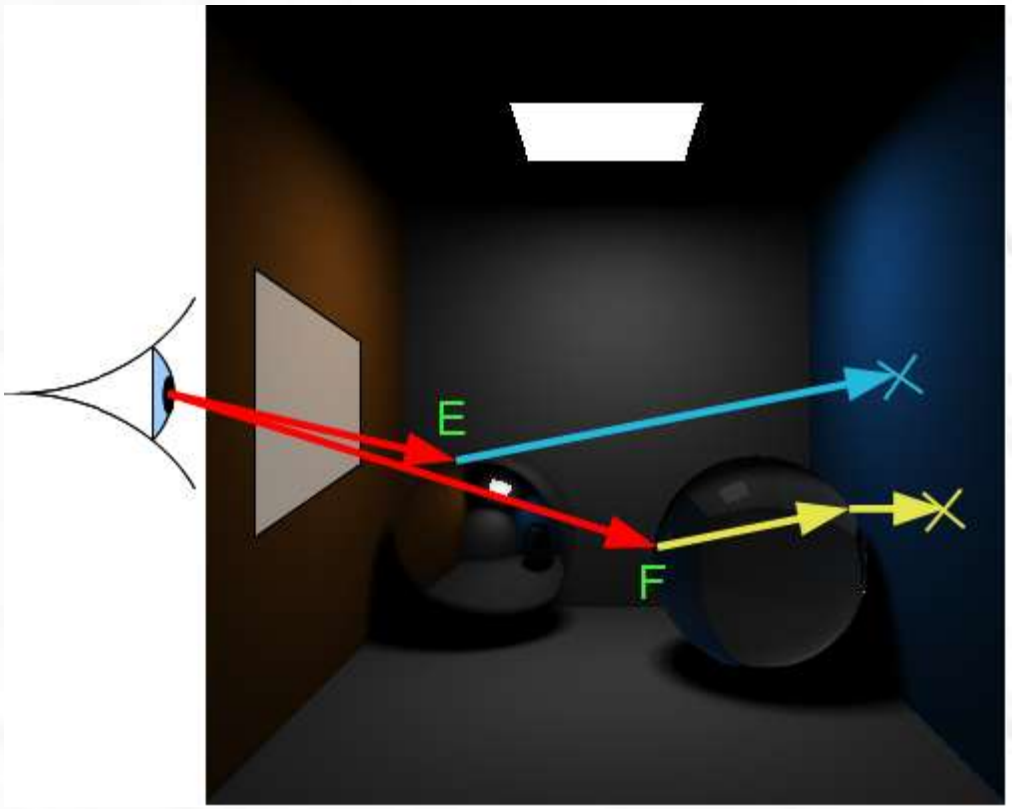
Solution: cast multiple shadow rays per light source



Optical Effects

Reflect rays off of *specular* (shiny) objects

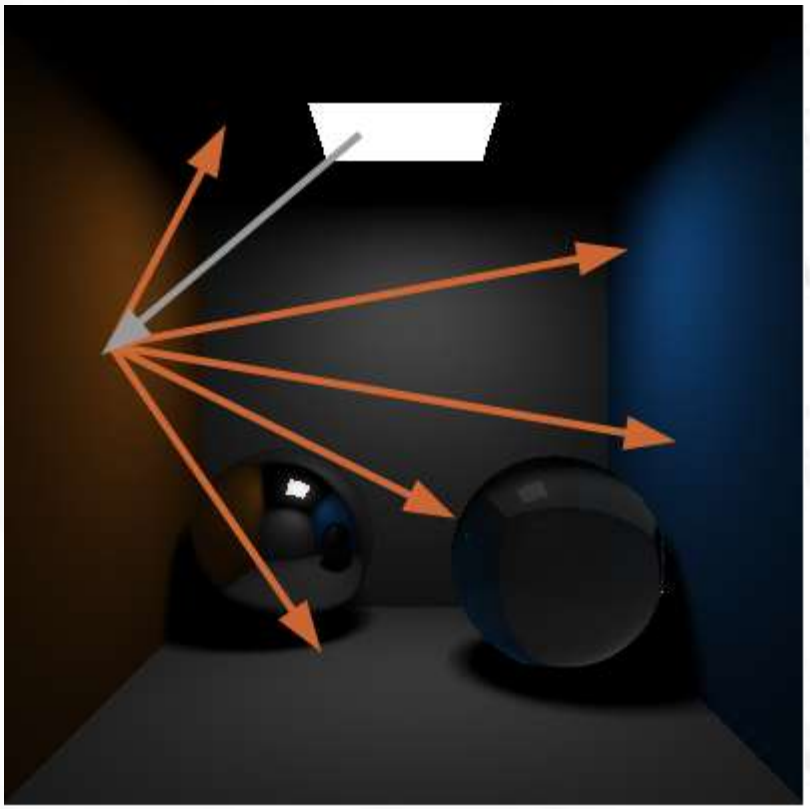
Refract rays through refractive (transparent) objects using Snell's Law



Global illumination

Notice the black ceiling and shadows!

Diffuse objects don't really absorb all light; they scatter some of it.



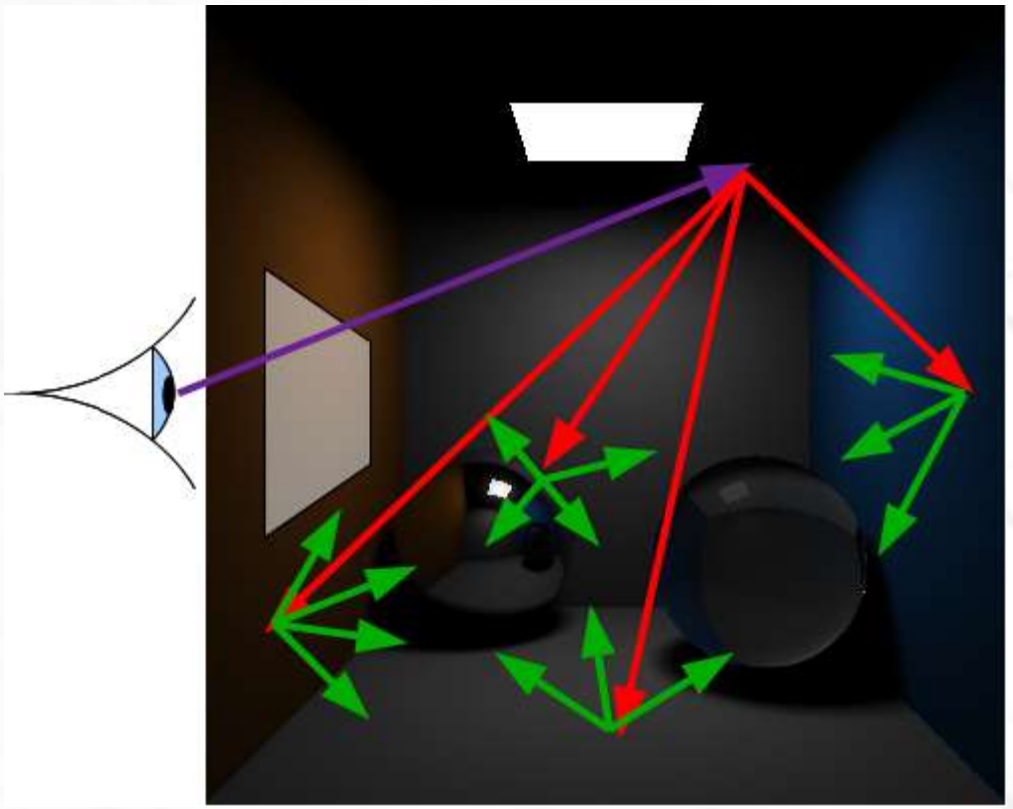
Scattering

Could use random sampling (Monte Carlo method)

Recursively sample multiple reflected rays in random directions

Tracing backwards results in exponential search

No guarantee that any particular branch will ever reach a light source



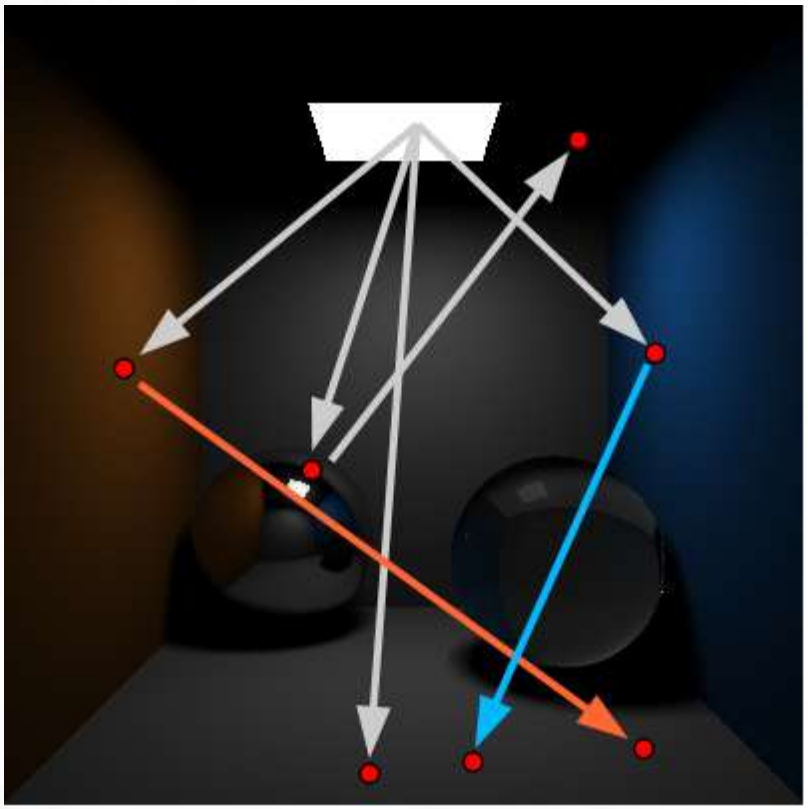
More Photon Mapping

Like with caustics, but in all directions

When photons hit a diffuse surface, some energy is absorbed, and some is reflected

Photons “bounce” until completely absorbed - expensive

Remember each bounce in photon map

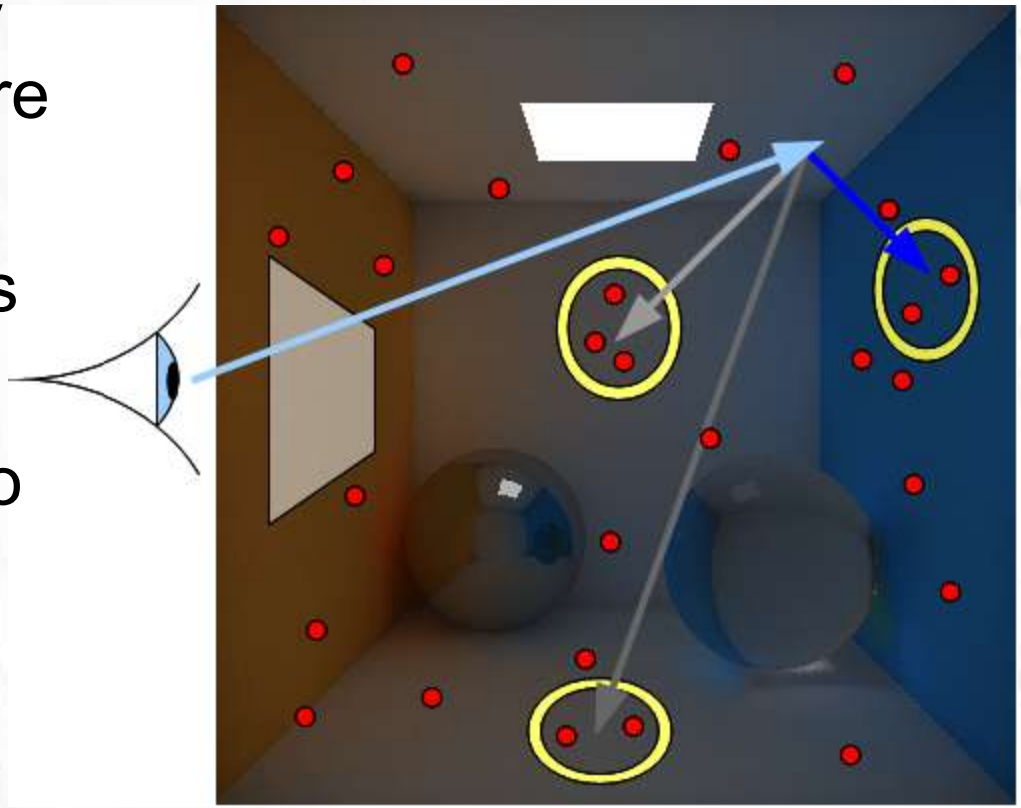


Hybridized Method

Photons will not be evenly distributed unless many are simulated – too costly

Uneven distribution results in blotchiness

Smooth using Monte Carlo Method



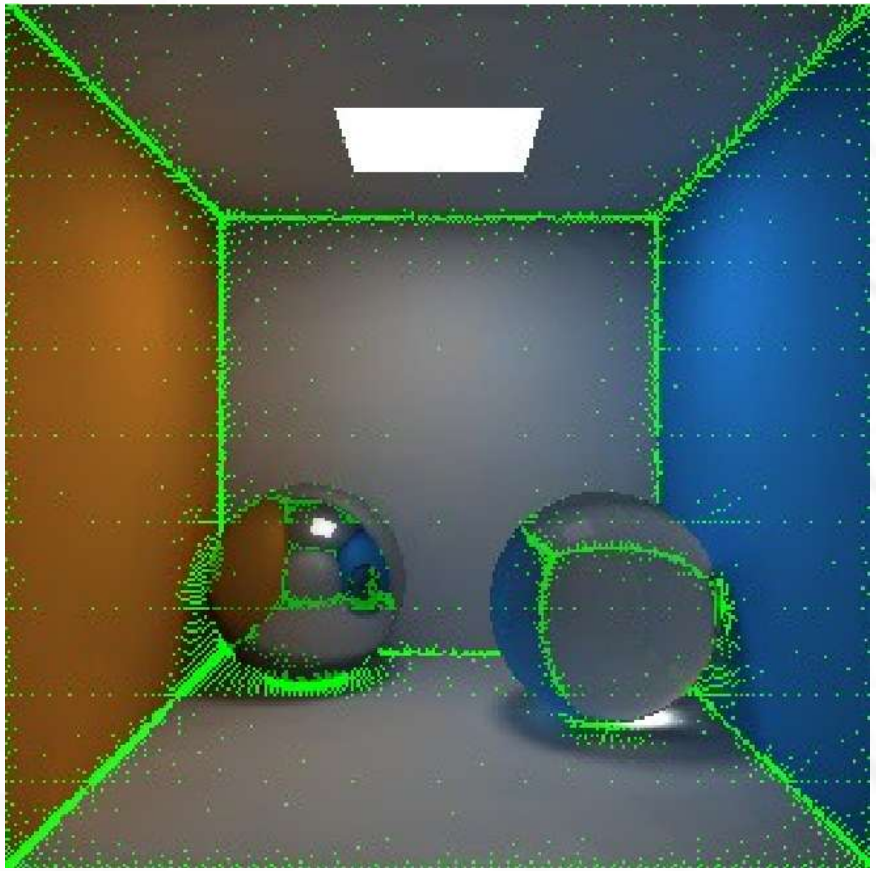
The Irradiance Cache

- Finding indirect illumination is still expensive
- In some situations, interpolation is likely to be very accurate
- While rendering, cache results of sampling and interpolate when possible
- A cached irradiance calculation can be used to interpolate results for nearby points with similar normal vectors.
- The icache is thus built while rendering occurs.

The Irradiance Cache

Green dots are points where samples were cached.

Note that caching is most frequent where normal vectors are changing.

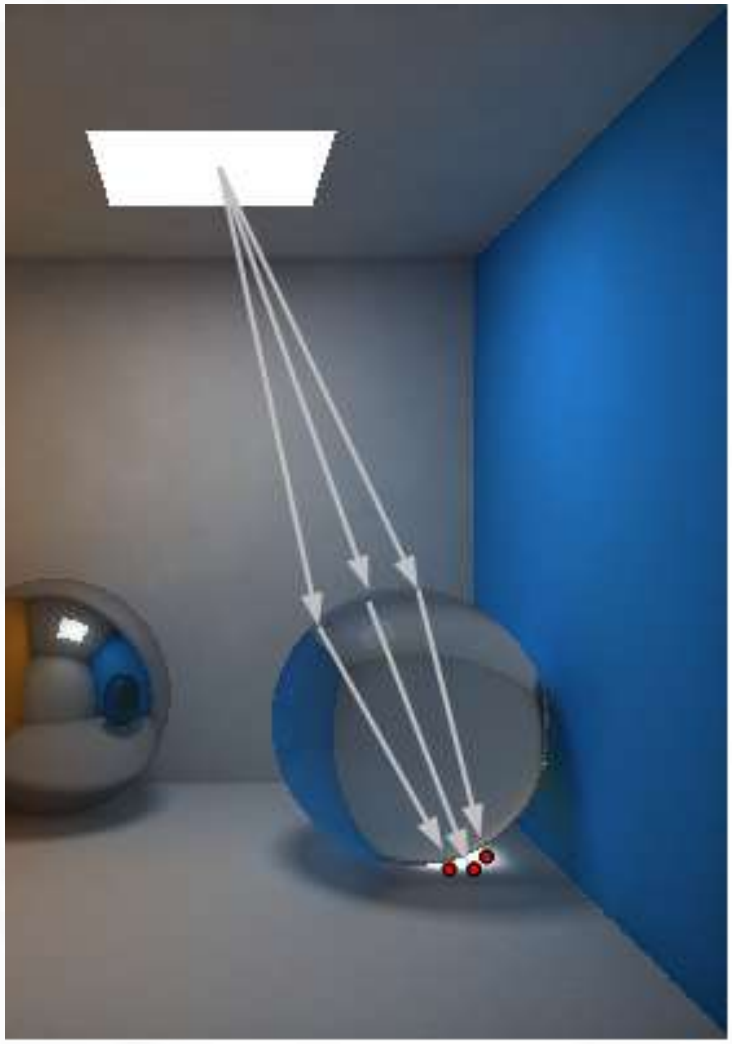


Caustics

Refractive objects often create patterns of bright and dark by focusing light.

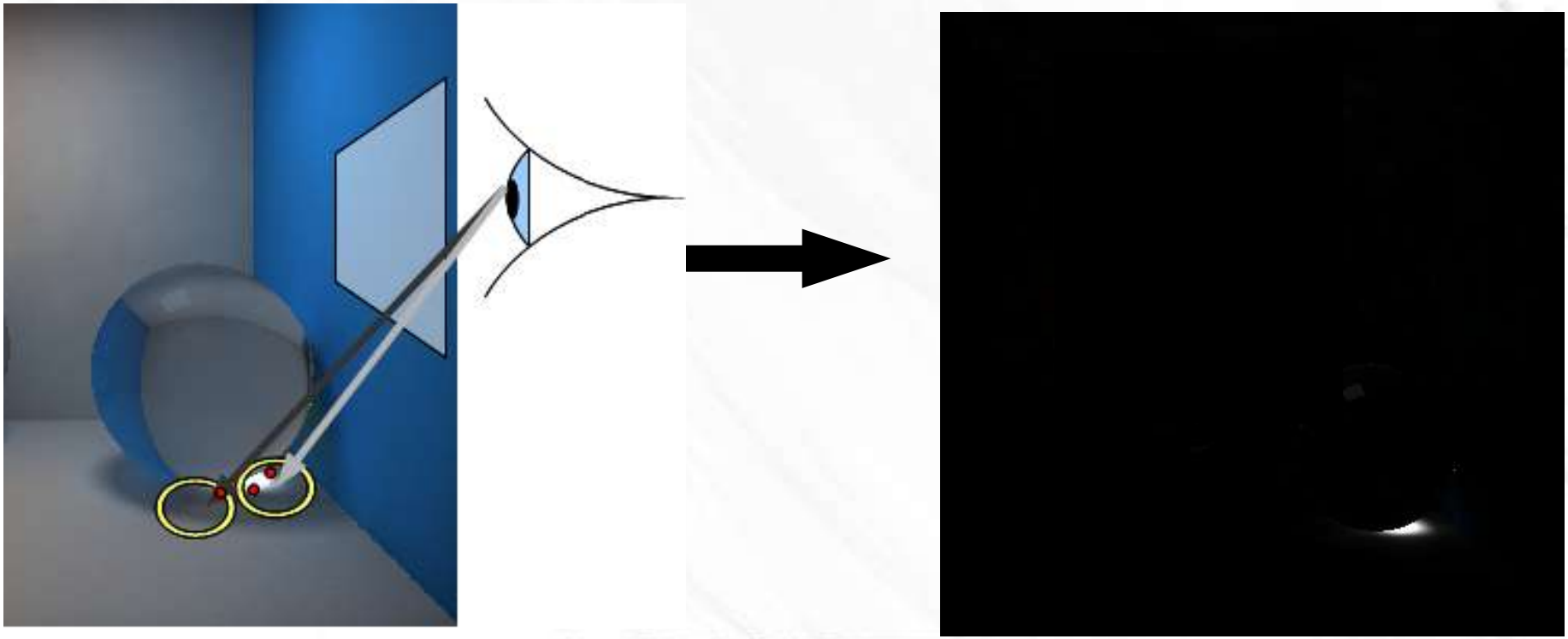
Cast rays from light towards refractive objects to simulate the paths of photons

Trace rays until they hit a diffuse surface; store hits in a *photon map*

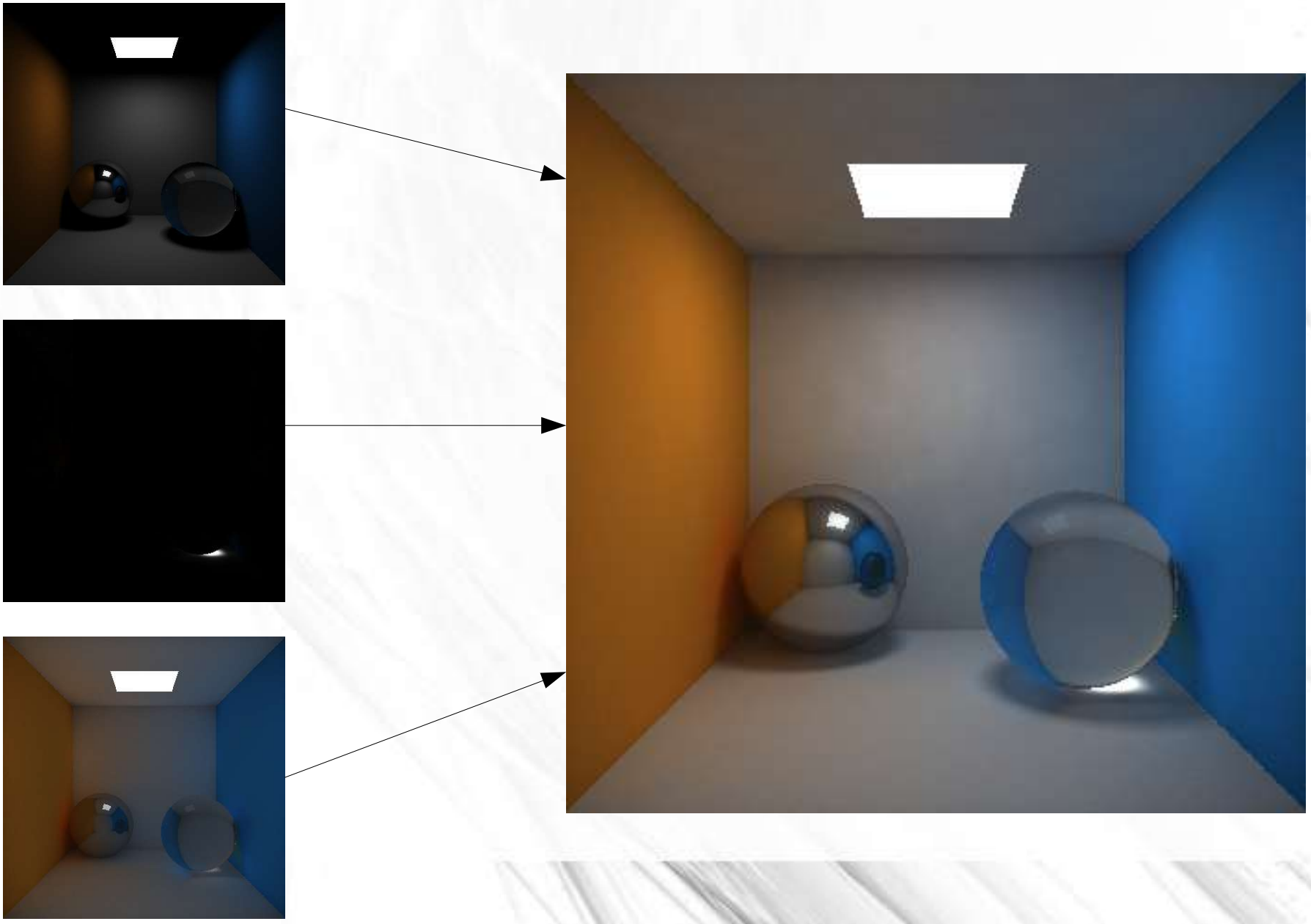


Rendering With a Photon Map

When rendering, the luminance from caustics is based on how many photons are nearby on the photon map



Putting it All Together



Code Overview - Classes

- SceneObject
  - Inherited by Square, Cube, Sphere, DisplacedSurface
  - Each object has its own ray intersection method

- Raytracer
  - Contains “main”
  - Methods for construction of scene space
  - Methods for rendering scene

Code Overview - Classes

- LightSource
  - Base class
- SquarePhotonLight
  - Used in both scenes
  - Contains seperate methods for each type of illumination

- ICache
  - Contains methods for constructing the ICache one sample at a time
  - Use find() to get illuminance at a certain point

Execution (high level)

```
Raytracer::main() {
    Parse command line args
    Construct scene to be rendered
    for each light source:
        light.tracePhotons() //construct global/caustic photon maps
    Render
}

Raytracer::render() {
    for each pixel cast ray:
        Find nearest intersection
        computeShading(ray)
        //cast recursive rays and sum up total illumination
}
```

Execution (high level)

```
SquareLightSource::tracePhotons() {
    //global illumination
    while (i< global_num)
        cast ray in random direction
        //bounce each ray until it is absorbed
        while ray power is non-negligible
            If ray.intersection.mat->isDiffuse()
                Store intersection in photon map
                Decrease ray power
                chance of diffuse reflection in random direction
                j++
            Chance of refraction/reflection/absorption of ray as
            appropriate for non-diffuse mats
}
```

Execution (high level)

```
SquareLightSource::tracePhotons() {
    //caustic illumination
    while (i< caustic_num)
        cast ray in random direction
        //bounce each ray until it is absorbed or hits a diffuse mat
        while(1)
            If ray.intersection.mat->isDiffuse()
                Store intersection in photon map
                I++
                break
            Chance of refraction/reflection/absorption of ray as
            appropriate for non-diffuse mats
}
```

Resources

More in depth explanations of the math and concepts behind ray tracing can be found at:

- http://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html
  (a good introduction to some of the basic concept and math)
- http://www.codermind.com/articles/Raytracer-in-C++-Introduction-What-is-ray-tracing.html
  (also a good introduction, and covers some more advanced concepts, too)
- http://www.cs.mtu.edu/~shene/PUBLICATIONS/2005/photon.pdf
  (A good explanation of photon mapping, with some nice pictures, too)
- *Note: there are many different variations of the algorithms used for ray tracing. The implementations found in the final project may differ from those you find in these or other resources.

6.172 Performance Engineering of Software Systems

Fall 2010