CSE-170 Computer Graphics

Lecture 18

Ray Tracing

Dr. Renato Farias
rfarias2@ucmerced.edu

Ray Tracing

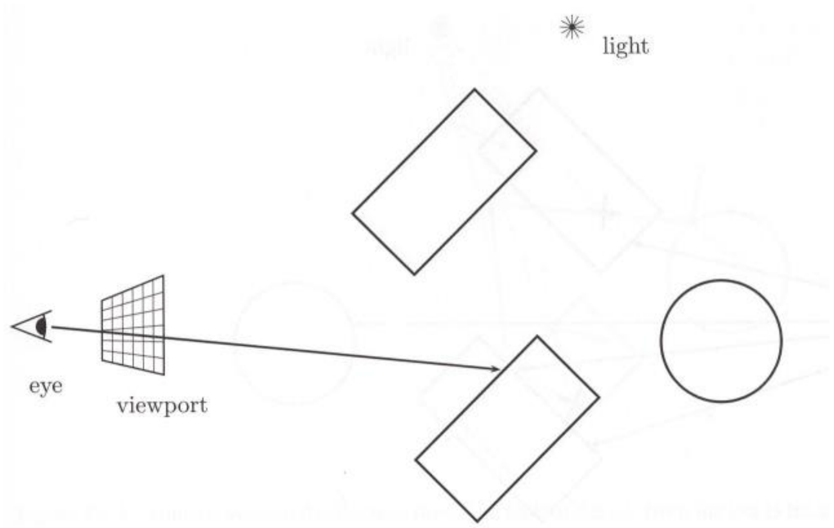
- Global technique for achieving
  - Lighting and shading
  - Hidden surface elimination
  - Reflection, refraction, transparency
  - Shadows
  - etc.
- Generates very realistic images
- ...but it is slow
  - Individual frames may take a long time

Ray Tracing

- Main Idea
  - Send a ray from the light source(s)
  - Reflect the ray through the scene
  - Until it hits the image plane
  - Continue ray propagation to achieve other effects

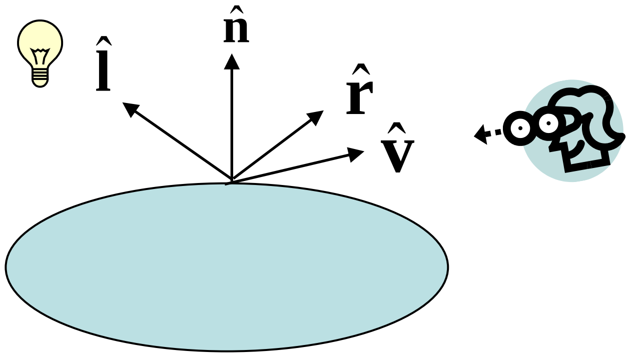
Ray Tracing

- Simplest Implementation
  - For every pixel in the image plane
    - Get closest intersection between the ray (sent from the center of the pixel) and the scene
    - Compute color for the intersection based on an illumination model (for example, Phong)

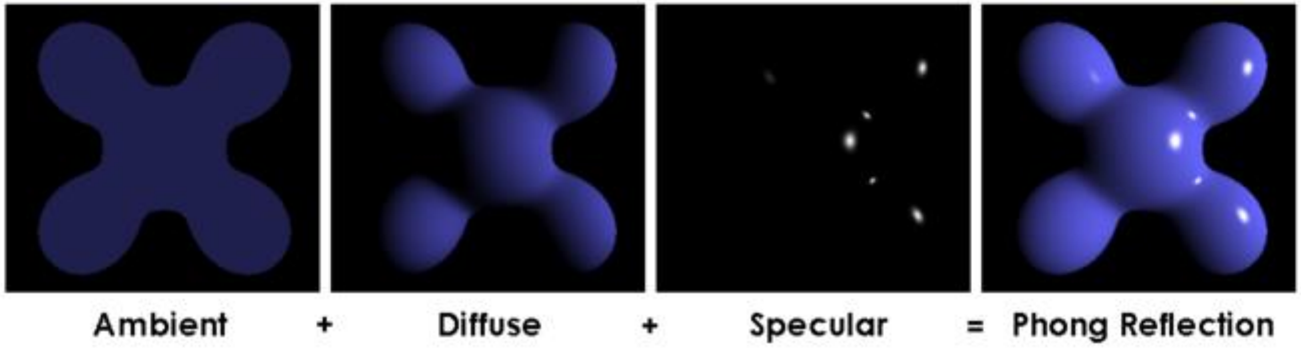


Ray Tracing

- Remembering Phong

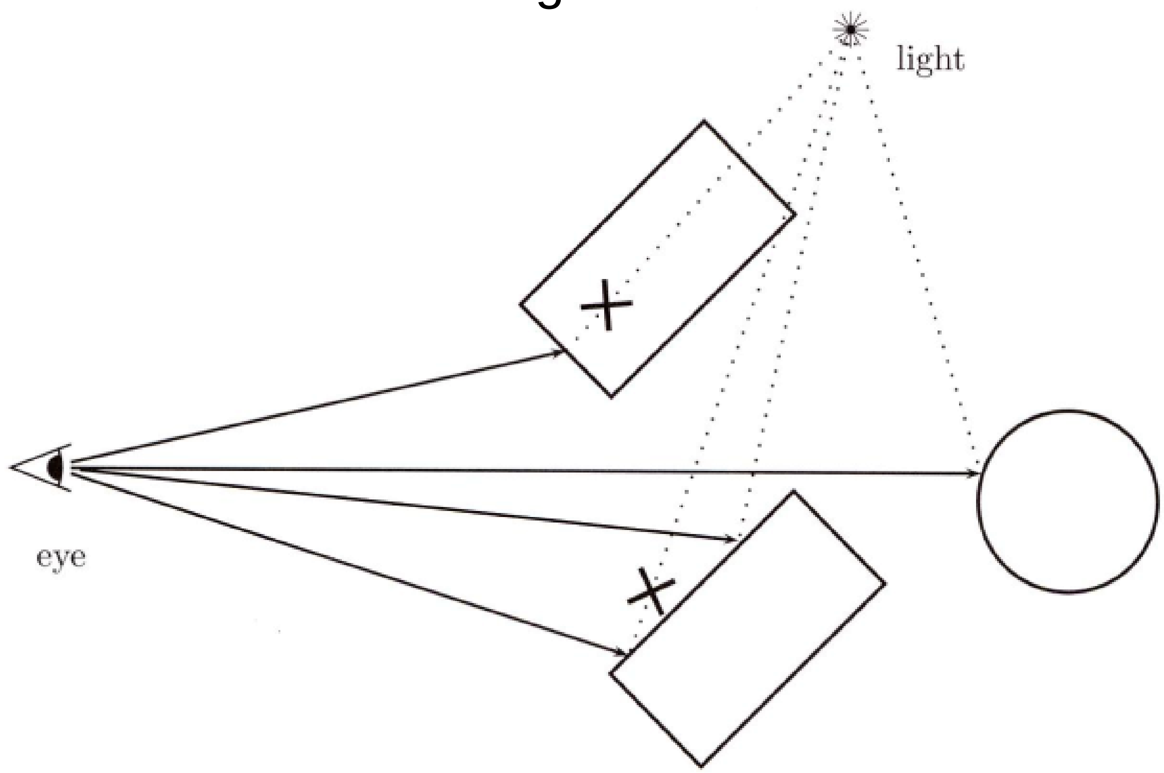


$$I = I_a k_a + I_d k_d (\hat{\mathbf{l}} \cdot \hat{\mathbf{n}}) + I_s k_s (\hat{\mathbf{v}} \cdot \hat{\mathbf{r}})^f$$



Ray Tracing

- Adding shadows:
  - "shadow feelers": for every hit, check if it can “see” the light source(s) with another ray
    - And update coefficient(s) (δ) to cancel or not (0 or 1) the effect of the direct light

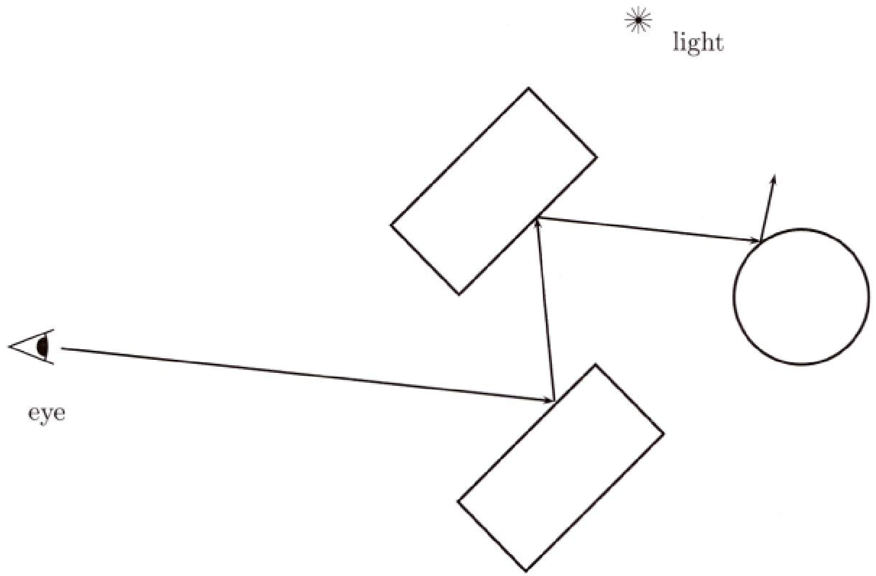


(Shadow feelers are sent to all light sources)

Ray Tracing

- Adding reflection rays:
  - Send a new ray in the reflection direction, which has the same angle with normal vector as the incoming ray"
  - Get the result of the reflected ray and add a portion of it (multiplying result by sigma in [0,1]) to the "first point hit":

$$I = I_{local} + \sigma_{rglob} I_{reflect}$$

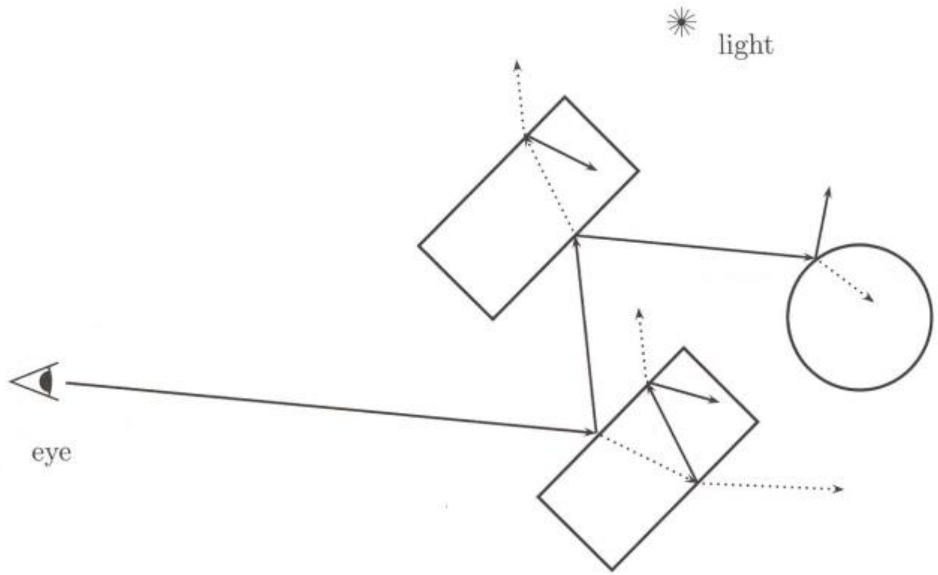


Ray Tracing

- Adding transmission rays:
  - Similar to reflected rays but for refractions
    - for generating transparency effects
  - Add its contribution to the equation:

$$I = I_{local} + \sigma_{rglob} I_{reflect} + \sigma_{tglob} I_{xmit}$$

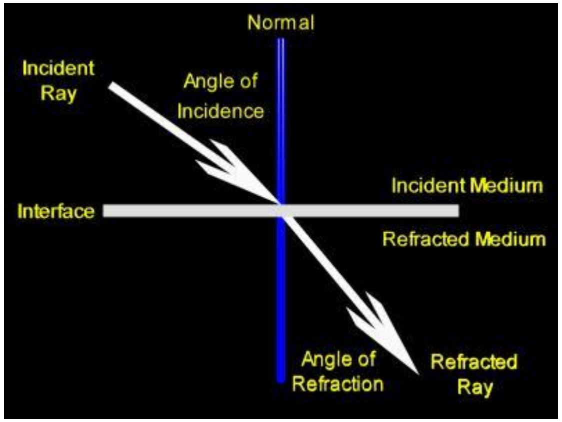
Example of
refraction effects

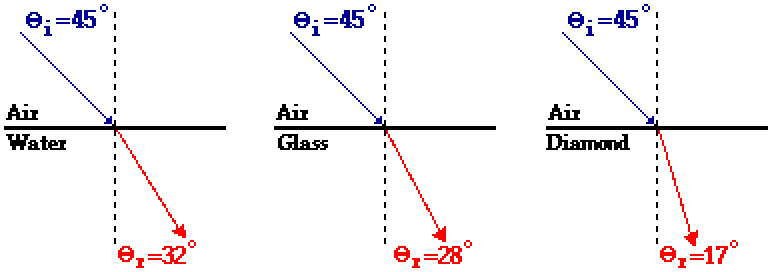


Ray Tracing

- How to compute refraction
  - Refraction depends on the amount of change in light speed
  - Different materials have different indices of refractions
  - If α is the "traversal cost in a material" $\Rightarrow \alpha_1 \sin\theta_i = \alpha_2 \sin\theta_r$

(You can try different values and see what the material looks like)





Ray Tracing: implementation

```
RayTraceMain() {
    // Let x be the position of the viewer.
    // Let maxDepth be a positive integer.
    For each pixel p in the viewport, do {
        Set u = unit vector in the direction from x to p.
        Call RayTrace( x, u, maxDepth );
        Assign pixel p the color returned by RayTrace.
    }
}
```

Reference: 3-D computer graphics, Samuel R. Buss

Ray Tracing

```
RayTrace( s, u, depth ) {
  // s is the starting position of the ray.
  // u is unit vector in the direction of the ray.
  // depth is the trace depth.
  // Return value is a 3-tuple of color values (R,G,B).
  // Part I - Nonrecursive computations
  Check the ray with starting position s and direction u
    against the surfaces of the objects in the scene.
    If it intersects any point, let z be the first intersection point
    and n be the surface normal at the intersection point.
  If no point was intersected {
    Return the background color.
  }
  For each light {
    Generate a shadow feeler from z to the light.
    Check if the shadow feeler intersects any object.
    Set δi and δ'i appropriately.
  }
  Set color = I_local;              // Use equation IX.7
```

Phong equation, modified by the shadow feeler coefficient

Ray Tracing

```
// Part II - Recursive computations
If ( depth==0 ) {
   Return color;                        // Reached maximum trace depth.
}
// Calculate reflection direction and add in reflection color
If ( ρ_rg ≠ 0 ) {                       // if nonzero reflectivity
   Set r = u − 2(u · n)n;               // Eq. IX.2 with v = −u.
   Set color = color + ρ_rg*RayTrace(z, r, depth-1);
}
// Calculate transmission direction (if any) and add in transmitted color
If ( ρ_tg ≠ 0 ) {                       // if has transparency
   // Let η be the index of refraction.
   Set t = CalcTransmissionDirection(−u, n, η);
   If t is defined {                    // if not total internal reflection
      Set color = color + ρ_tg*RayTrace(z, t, depth-1);
   }
}
Return color;
}
```

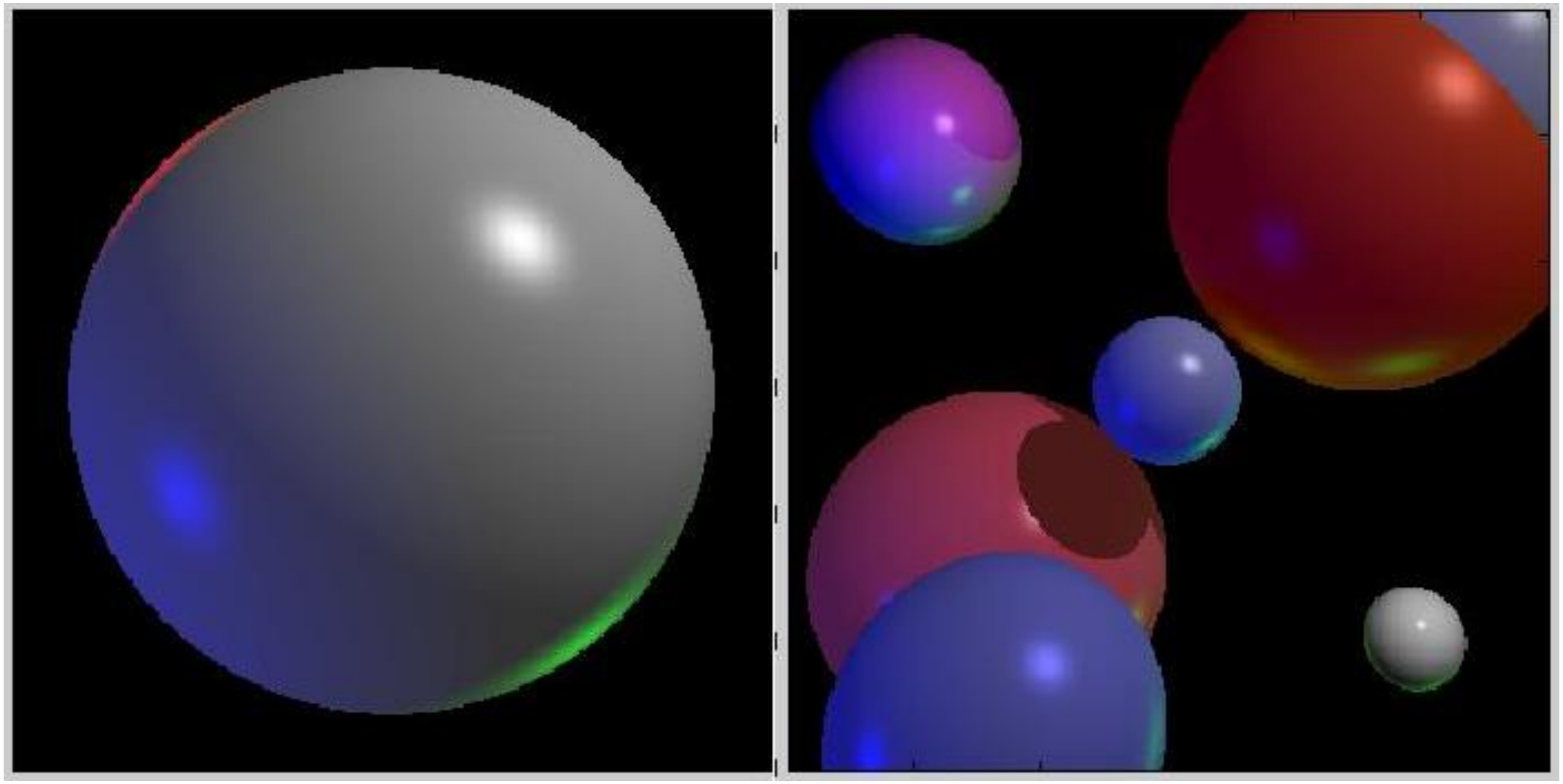
Full Algorithm

```
RayTrace( s, u, depth ) {
  // s is the starting position of the ray.
  // u is unit vector in the direction of the ray.
  // depth is the trace depth.
  // Return value is a 3-tuple of color values (R,G,B).

  // Part I - Nonrecursive computations
  Check the ray with starting position s and direction u
    against the surfaces of the objects in the scene.
    If it intersects any point, let z be the first intersection point
    and n be the surface normal at the intersection point.
  If no point was intersected {
    Return the background color.
  }
  For each light {
    Generate a shadow feeler from z to the light.
    Check if the shadow feeler intersects any object.
    Set δ_i and δ'_i appropriately.
  }
  Set color = I_local;                // Use equation IX.7

  // Part II - Recursive computations
  If ( depth==0 ) {
    Return color;                     // Reached maximum trace depth.
  }
  // Calculate reflection direction and add in reflection color
  If ( ρ_rg ≠ 0 ) {                   // if nonzero reflectivity
    Set r = u − 2(u · n)n;            // Eq. IX.2 with v = −u.
    Set color = color + ρ_rg*RayTrace(z, r, depth-1);
  }
  // Calculate transmission direction (if any) and add in transmitted color
  If ( ρ_tg ≠ 0 ) {                   // if has transparency
    // Let η be the index of refraction.
    Set t = CalcTransmissionDirection(−u, n, η);
    If t is defined {                 // if not total internal reflection
      Set color = color + ρ_tg*RayTrace(z, t, depth-1);
    }
  }
  Return color;
}
```

Examples



Examples

- Persistence of Vision Raytracer (povray.org)