

# COS-751 Computer Graphics

## Lecture 16

### Ray Tracing

Drs. Farias  
rfarias@cos.ufrj.br

# 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 to generate

# Ray Tracing

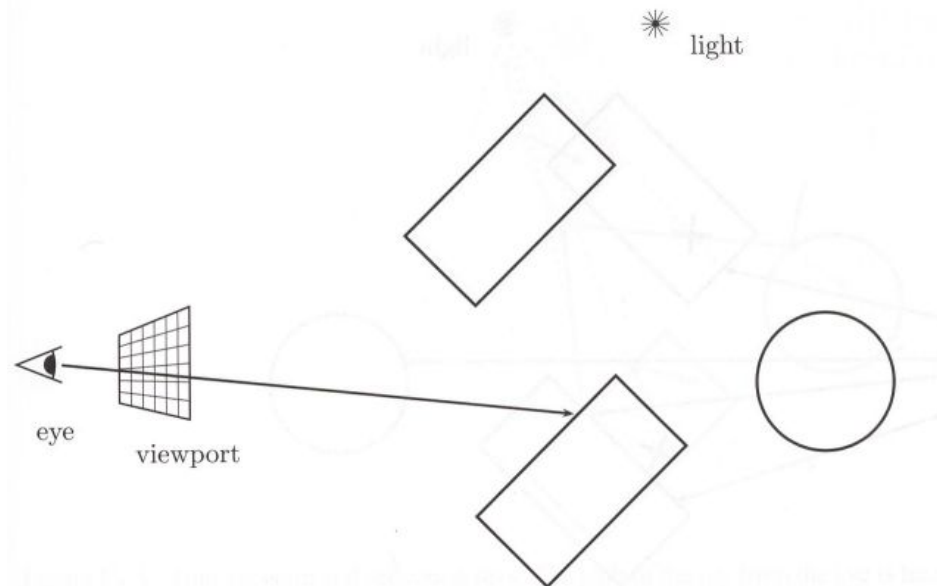
---

- Main idea (*conceptually*)
  - Send rays out from the light source(s)
  - Reflect the rays throughout the whole scene
  - Until they hit the image plane
  - Optionally continue ray propagation to achieve other effects
- However, most rays will probably not hit the image plane, so this is incredibly inefficient
- In practice, we do it the other way around: start with the image plane, and send rays into the scene
  - Result is equivalent

# Ray Tracing

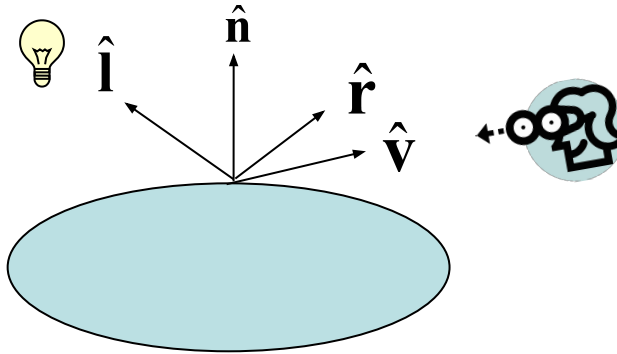
---

- 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)

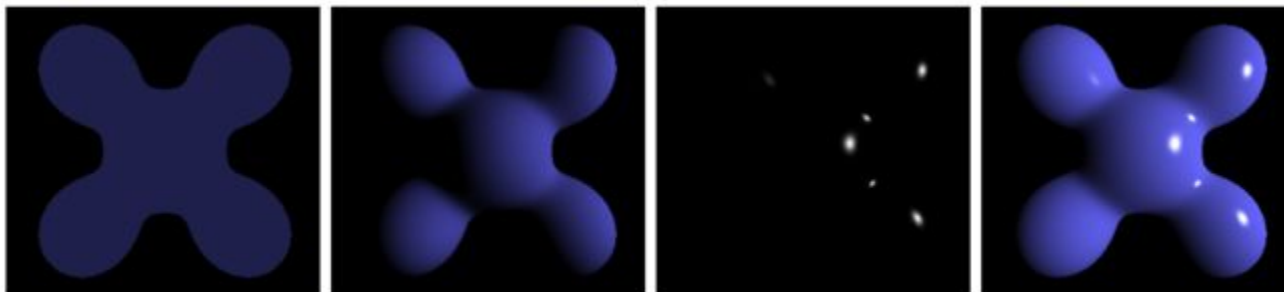


# Ray Tracing

- Remembering Phong



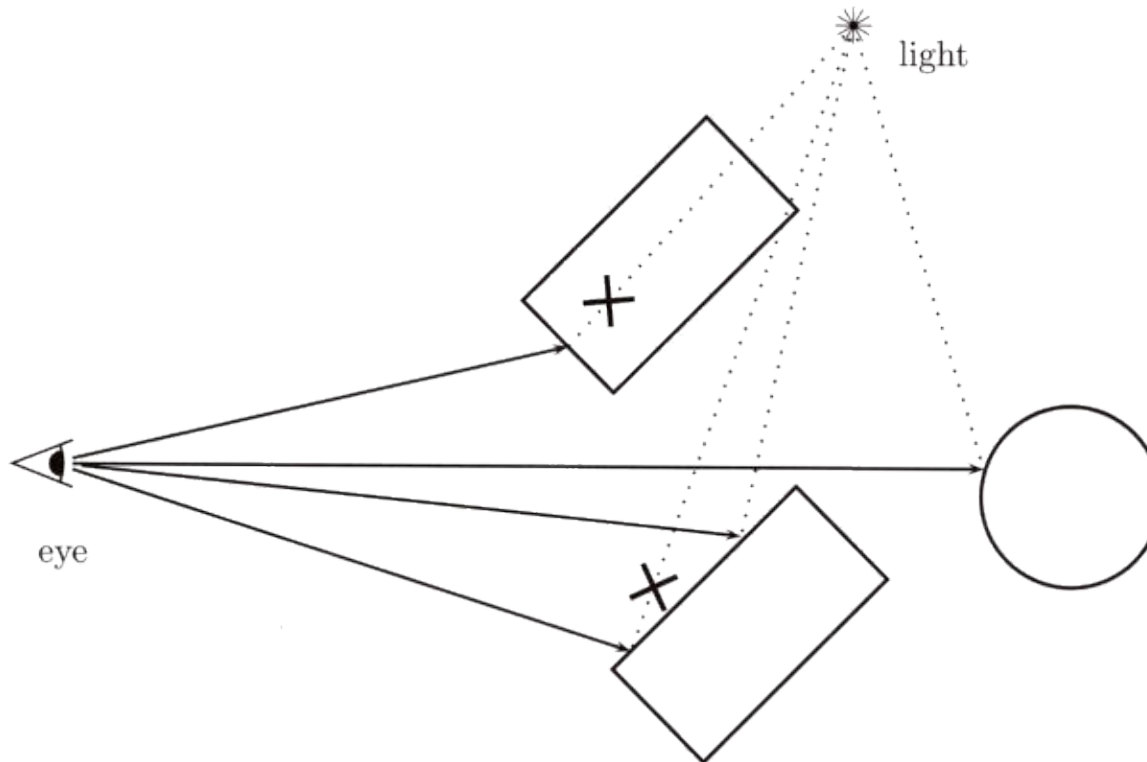
$$I = I_a k_a + I_d k_d (\hat{\mathbf{l}} \cdot \hat{\mathbf{n}}) + I_s k_s (\hat{\mathbf{v}} \cdot \hat{\mathbf{r}})^f$$



Ambient + Diffuse + Specular = Phong Reflection

# 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) ( $\delta$ ) to cancel or not (0 or 1) the effect of the direct light

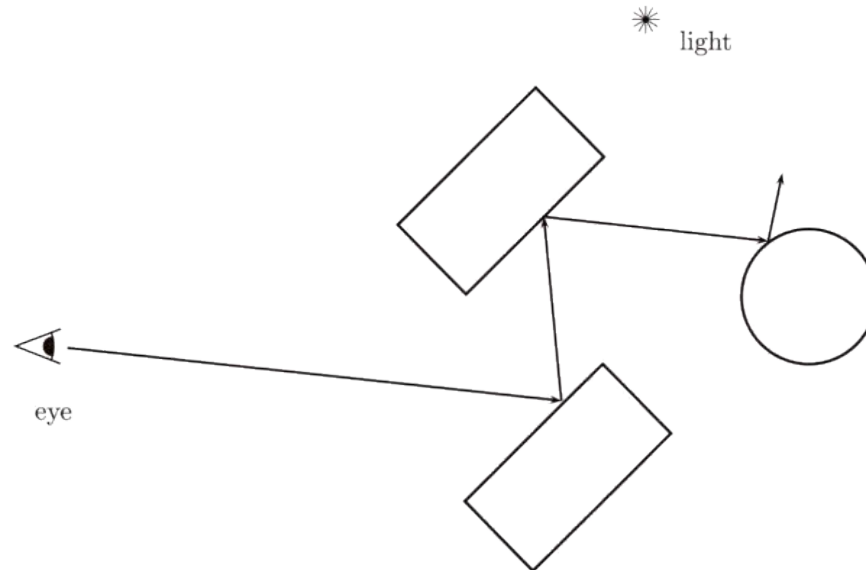


*(Shadow feelers are sent to all light sources)*

# 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":

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



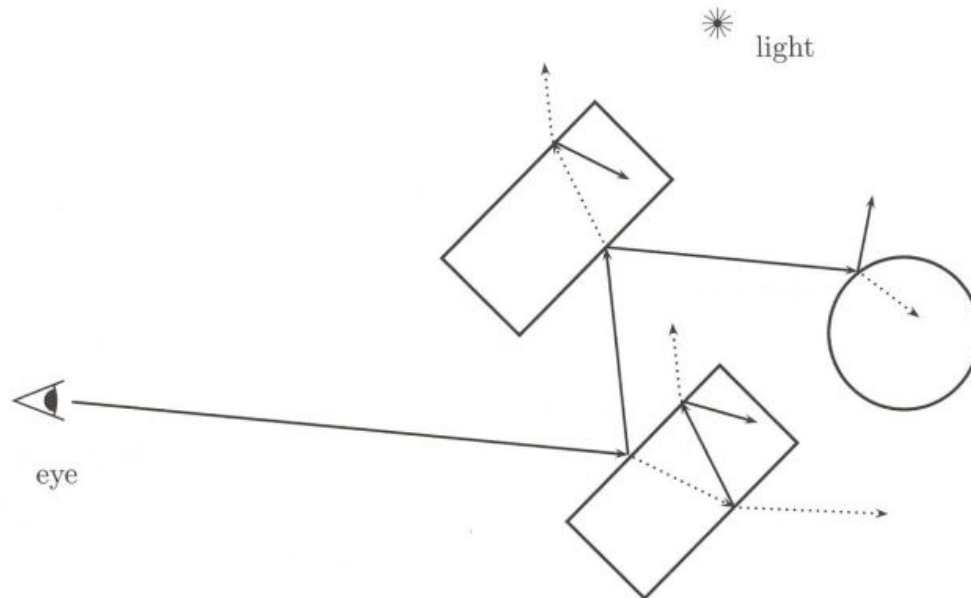
# Ray Tracing

- 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

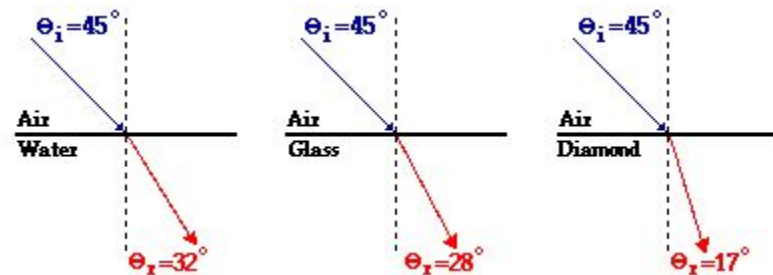
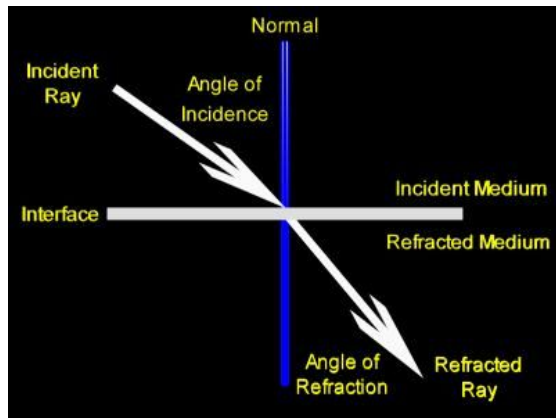




# Ray Tracing

- How to compute refraction
  - Refraction depends on the amount of change in light speed
  - Different materials have different indices of refractions
  - If  $\alpha$  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 ex.

---

```
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.  
    }  
}
```

# Ray Tracing: implementation ex.

```
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  $\delta_i$  and  $\delta'_i$  appropriately.  
    }  
    Set color =  $\mathbf{I}_{\text{local}}$ ; // Use equation IX.7
```

*Phong equation, modified by the shadow feeler coefficient*

# Ray Tracing: implementation ex.

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

# I 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).
```

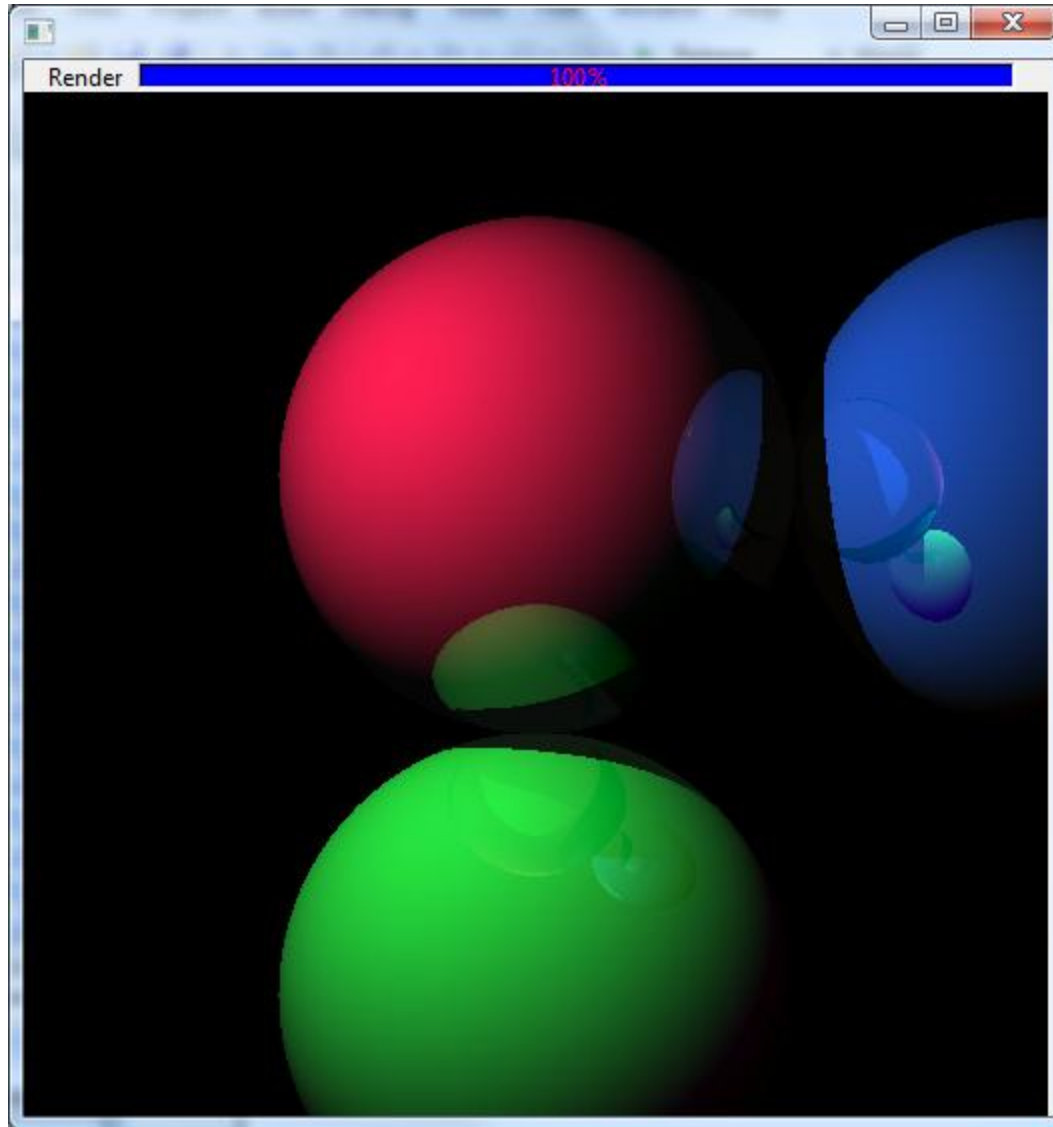
```
    // Part I - Nonrecursive computations  
    Check the ray with starting position s and direction u  
    against the surfaces of the objects in the scene.  
    If it intersects any point, let z be the first intersection point  
    and n be the surface normal at the intersection point.  
    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  $\delta_i$  and  $\delta'_i$  appropriately.  
    }  
    Set color =  $I_{\text{local}}$ ;           // 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 (  $\rho_{\text{rg}} \neq 0$  ) {           // if nonzero reflectivity  
        Set  $\mathbf{r} = \mathbf{u} - 2(\mathbf{u} \cdot \mathbf{n})\mathbf{n}$ ;           // Eq. IX.2 with  $\mathbf{v} = -\mathbf{u}$ .  
        Set color = color +  $\rho_{\text{rg}} * \text{RayTrace}(\mathbf{z}, \mathbf{r}, \text{depth}-1)$ ;  
    }  
    // Calculate transmission direction (if any) and add in transmitted color  
    If (  $\rho_{\text{tg}} \neq 0$  ) {           // if has transparency  
        // Let  $\eta$  be the index of refraction.  
        Set  $\mathbf{t} = \text{CalcTransmissionDirection}(-\mathbf{u}, \mathbf{n}, \eta)$ ;  
        If  $\mathbf{t}$  is defined {           // if not total internal reflection  
            Set color = color +  $\rho_{\text{tg}} * \text{RayTrace}(\mathbf{z}, \mathbf{t}, \text{depth}-1)$ ;  
        }  
    }  
    Return color;  
}
```

# Examples

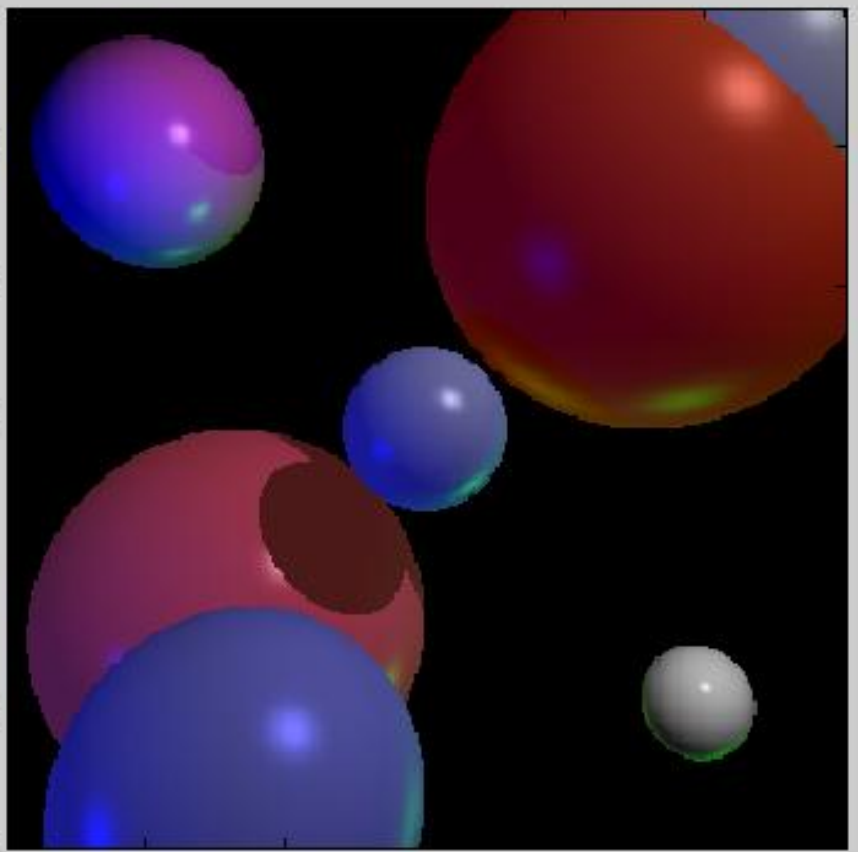
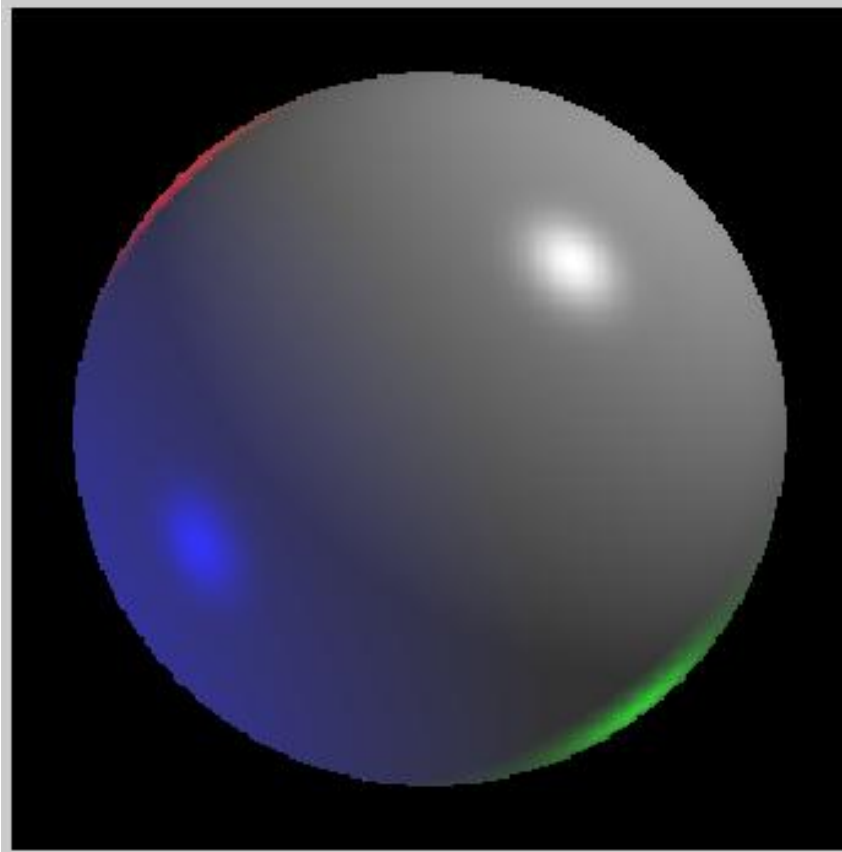
---

By Robert  
Backmann,  
CSE170 2009



# Examples

---





# Examples

---

- Persistence of Vision Raytracer ([povray.org](http://povray.org))





<https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel>