CS148 Summer 2025 - Lecture 7

07/15/2025

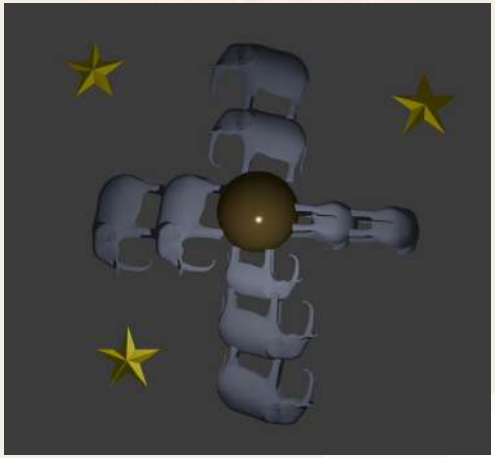
Raytracing II

cs148.stanford.edu

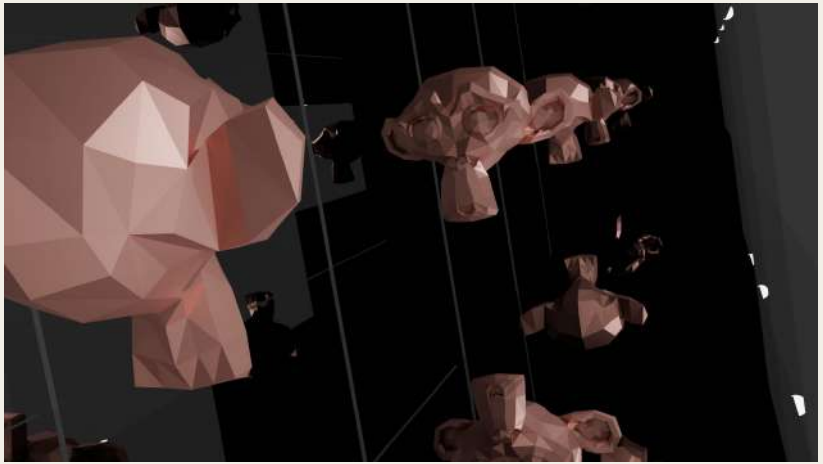
Abby chen

Felix Fan

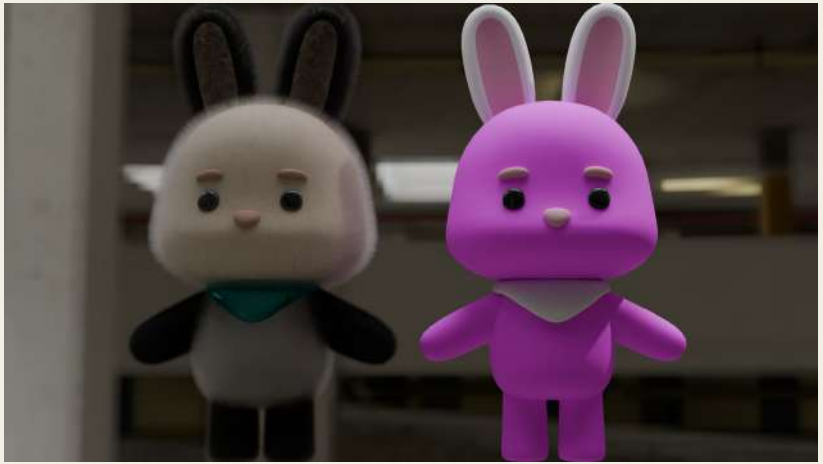
Crystal Chen



Jida Zhang



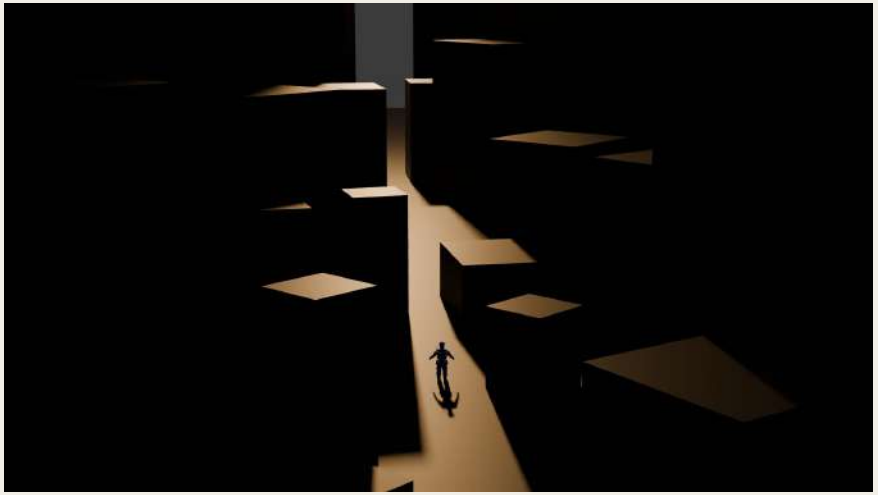
Daniela Barrera

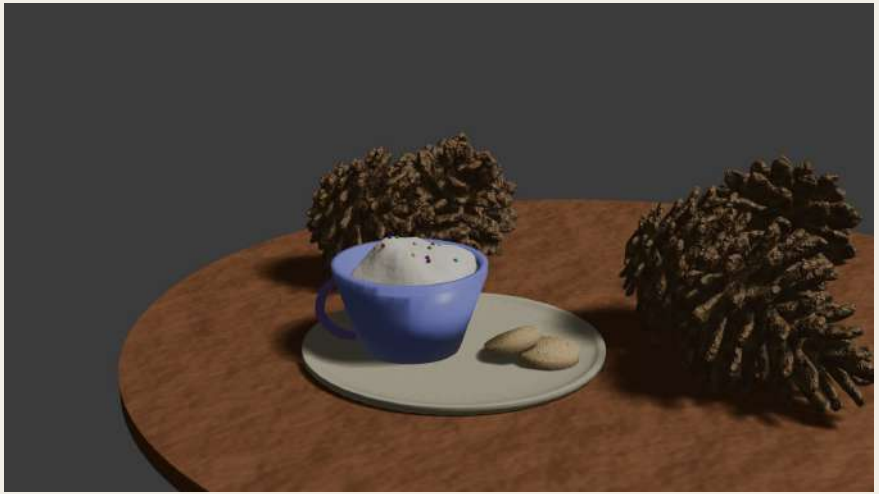


Seigo Hayami

Benjamin Zeidel, Calvin Sokk

Jericha Liu





Yicheng Wang
Hanjun Luo

Grace Zhao

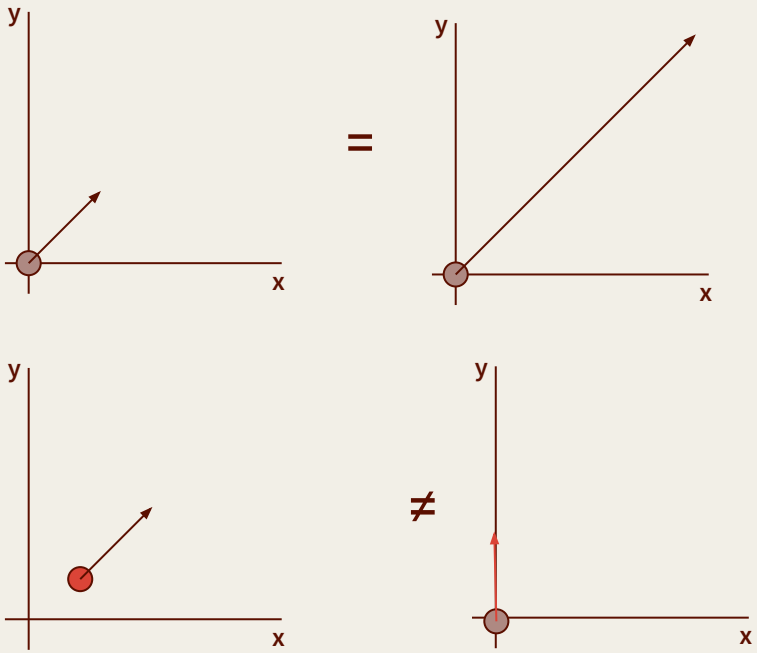
Solveig Jonsdottir

Announcements

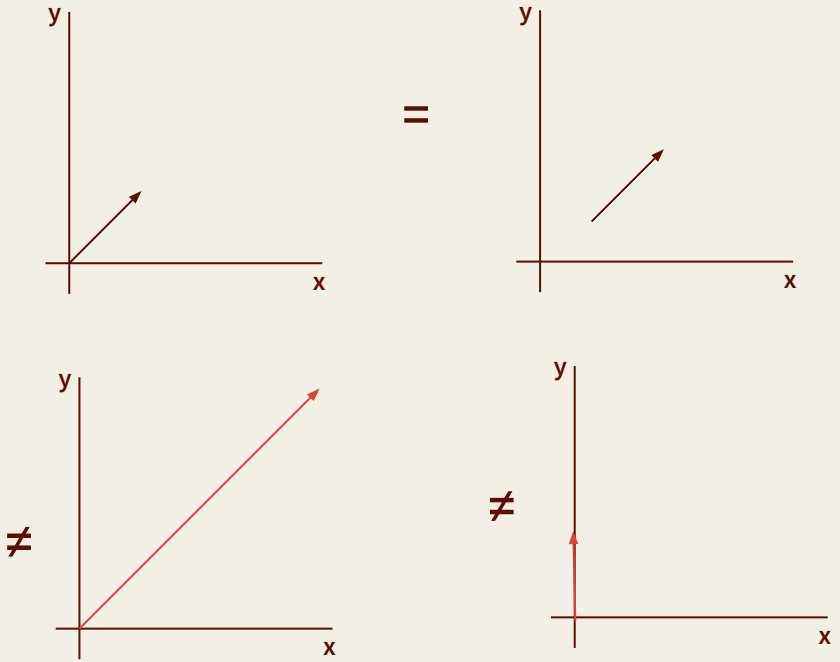
- Start thinking about the final project
- Proposal due next week together with the homework.

Vector Math Review

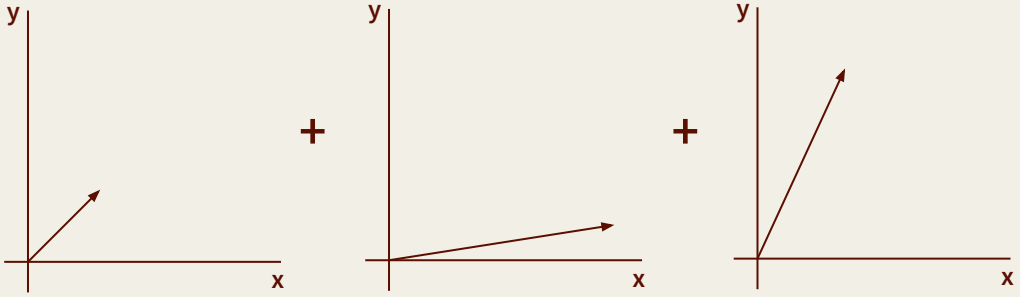
Ray = point + direction



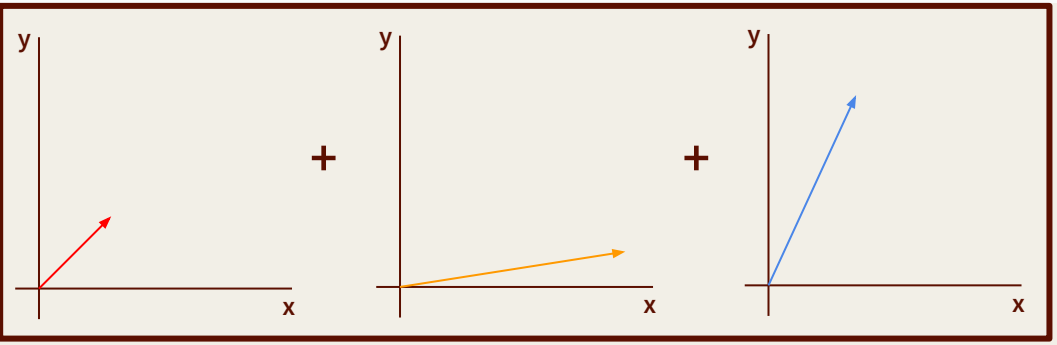
Vector = direction + length

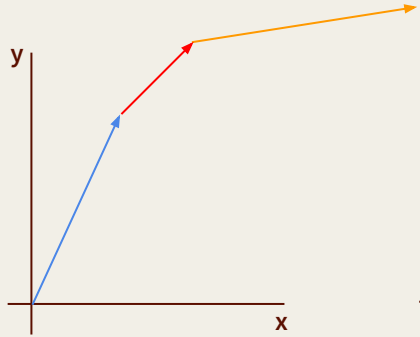


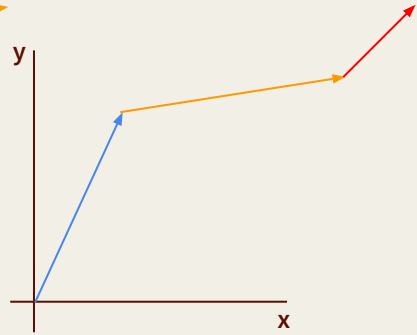
Vector Addition

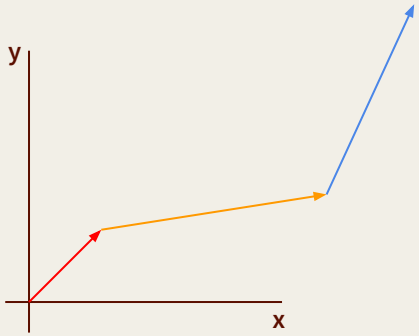


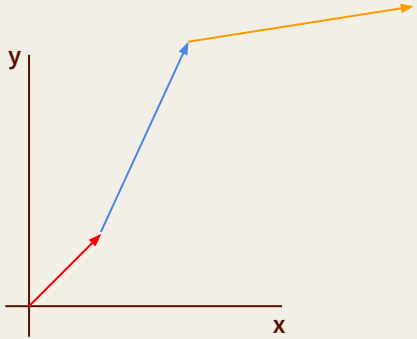
Vector Addition

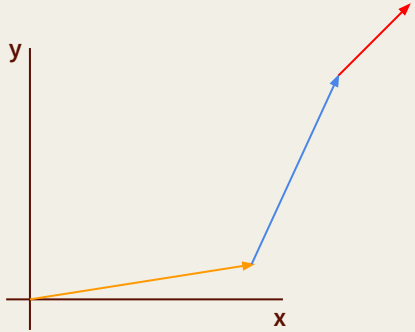


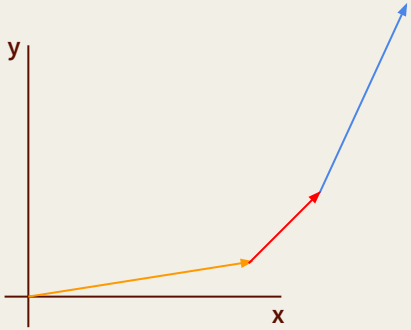




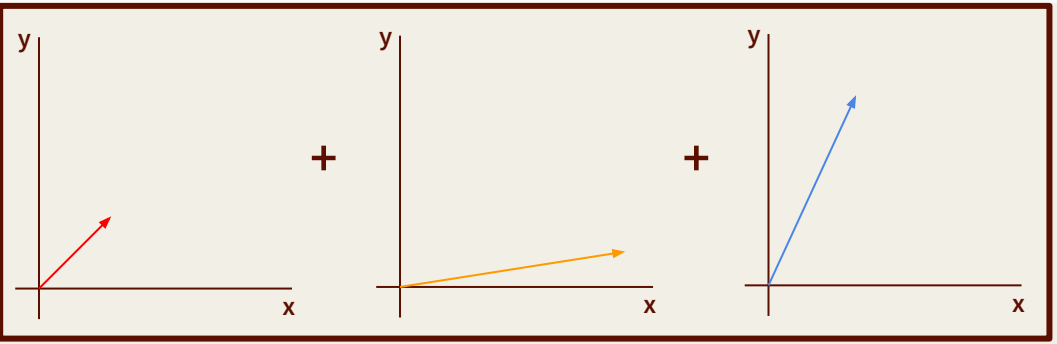


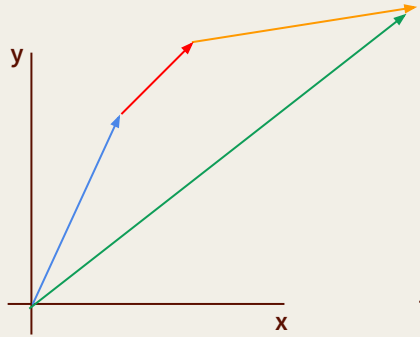


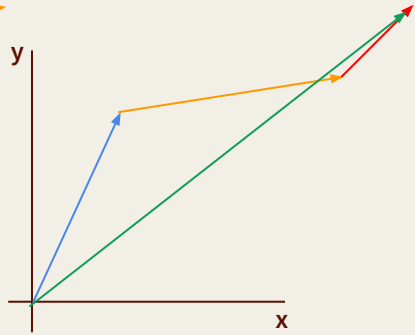


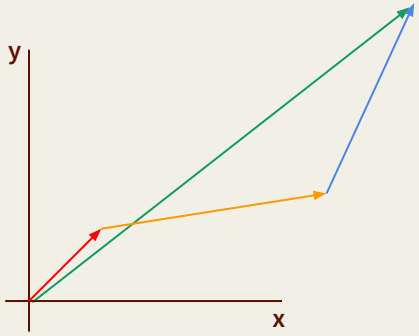


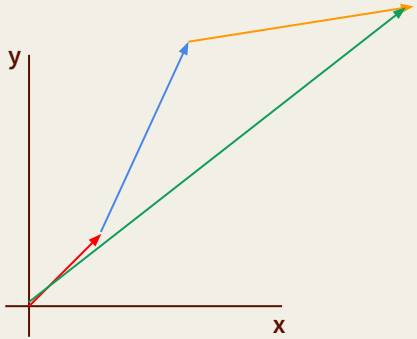
Vector Addition

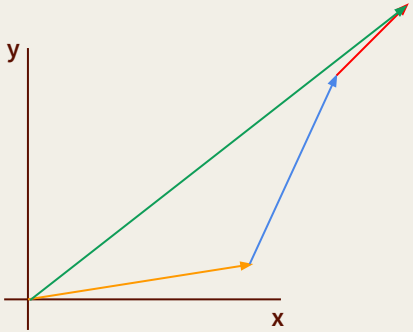




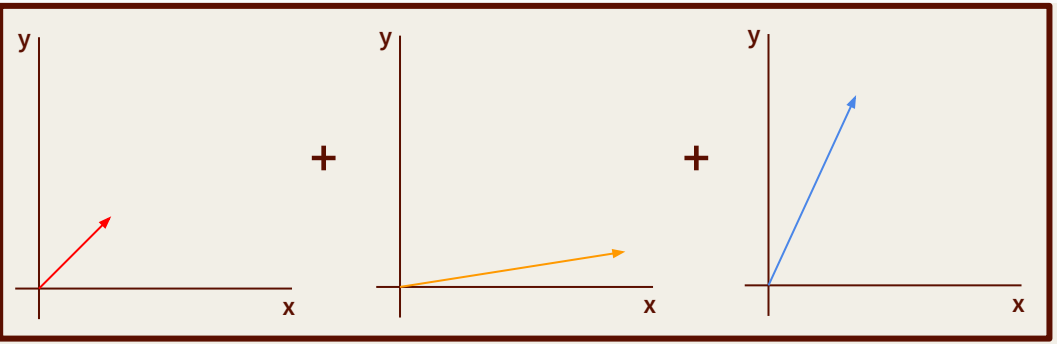


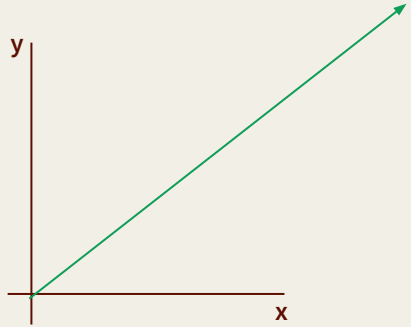


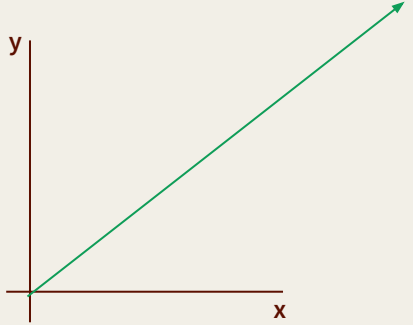


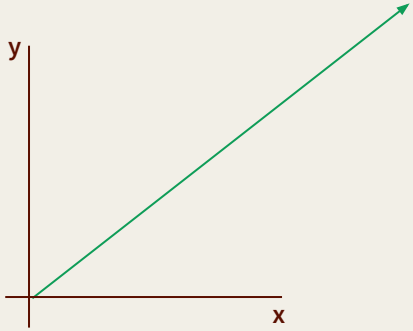


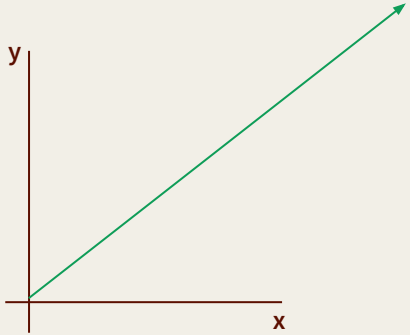
Vector Addition

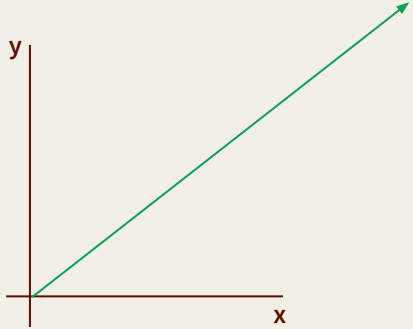


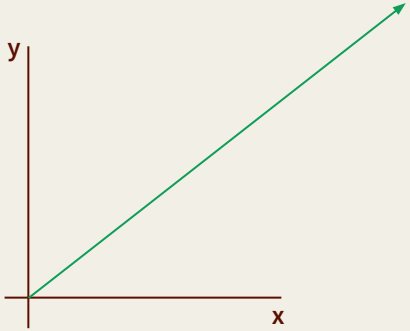




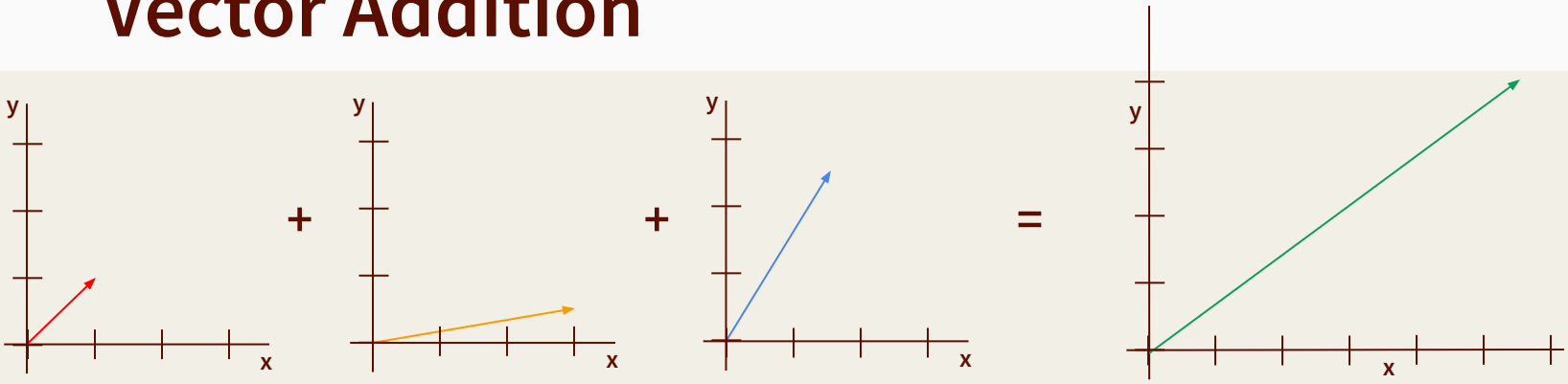






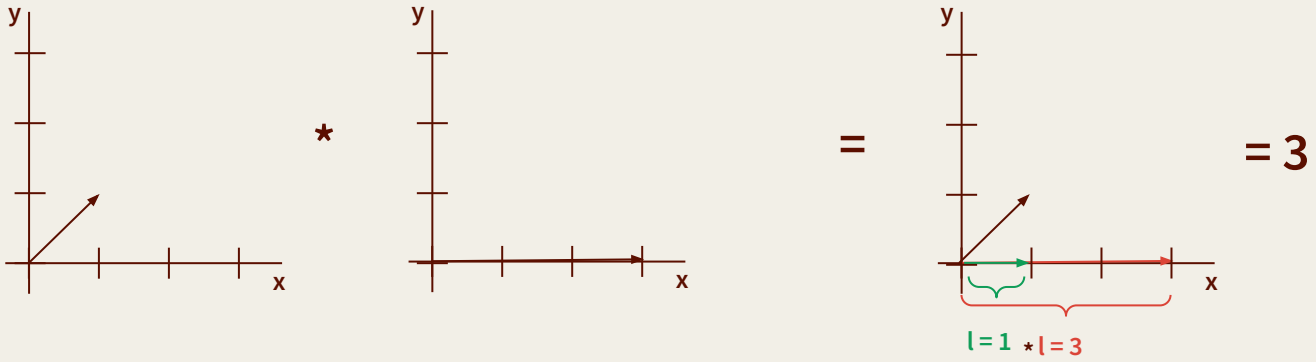


Vector Addition



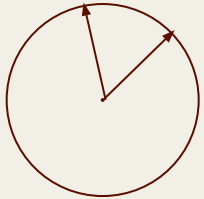
$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 3 \\ .5 \end{pmatrix} + \begin{pmatrix} 1.5 \\ 2.5 \end{pmatrix} = \begin{pmatrix} 1+3+1.5 \\ 1+.5+2.5 \end{pmatrix} = \begin{pmatrix} 5.5 \\ 4 \end{pmatrix}$$

Dot Products

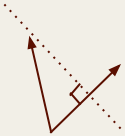


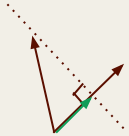
$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} * \begin{pmatrix} 3 \\ 0 \end{pmatrix} = (1 * 3) + (1 * 0) = 3$$

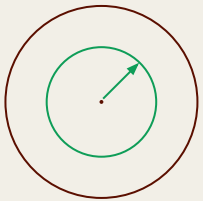
Dot Products



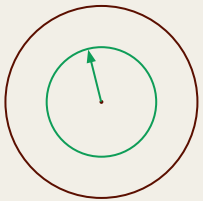
l = 1





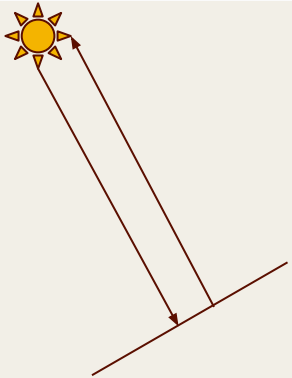


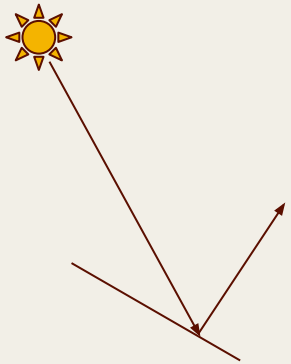
= .57

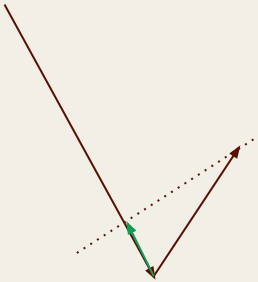


= .57

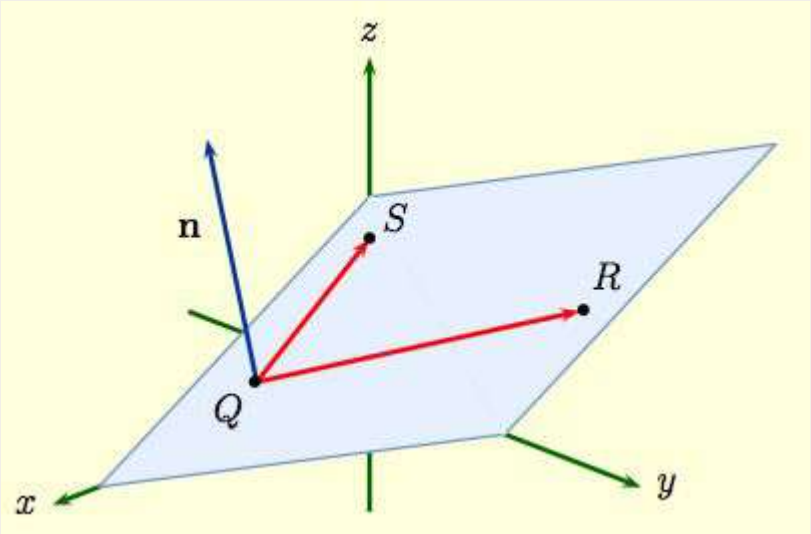
Dot Products







Cross Products



Ray Tracing

Scanline Renderer

Ray Tracing

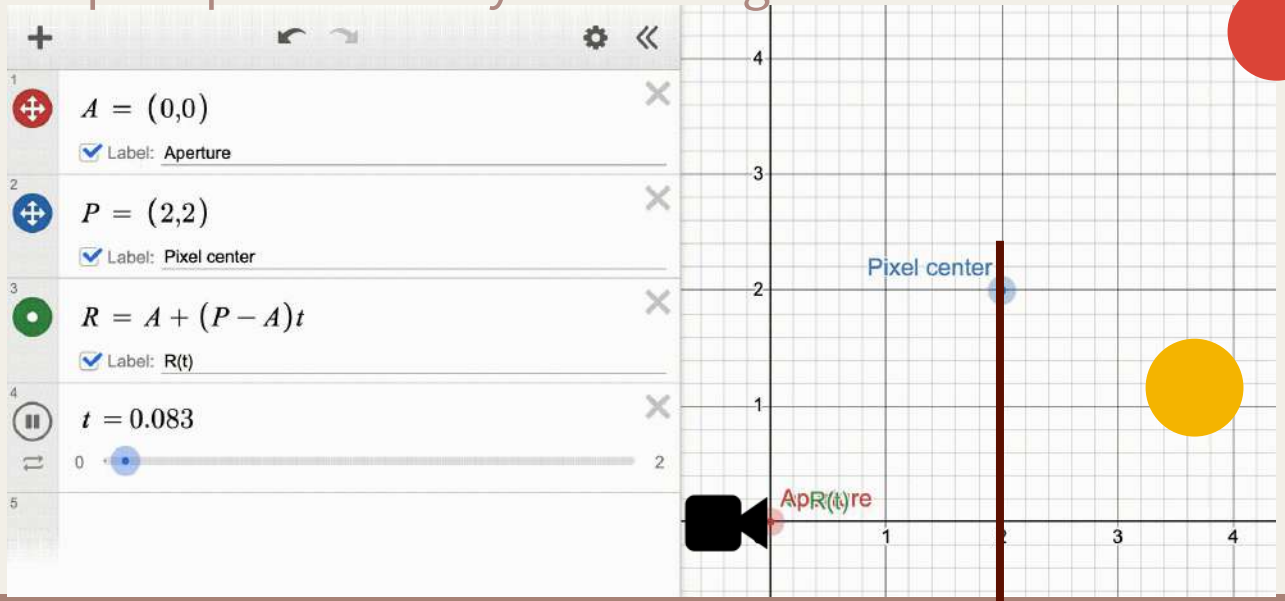
- Part 1: ray-object intersections & shadows
- **Part 2: reflections, transmissions, & other recursive concepts**

Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF

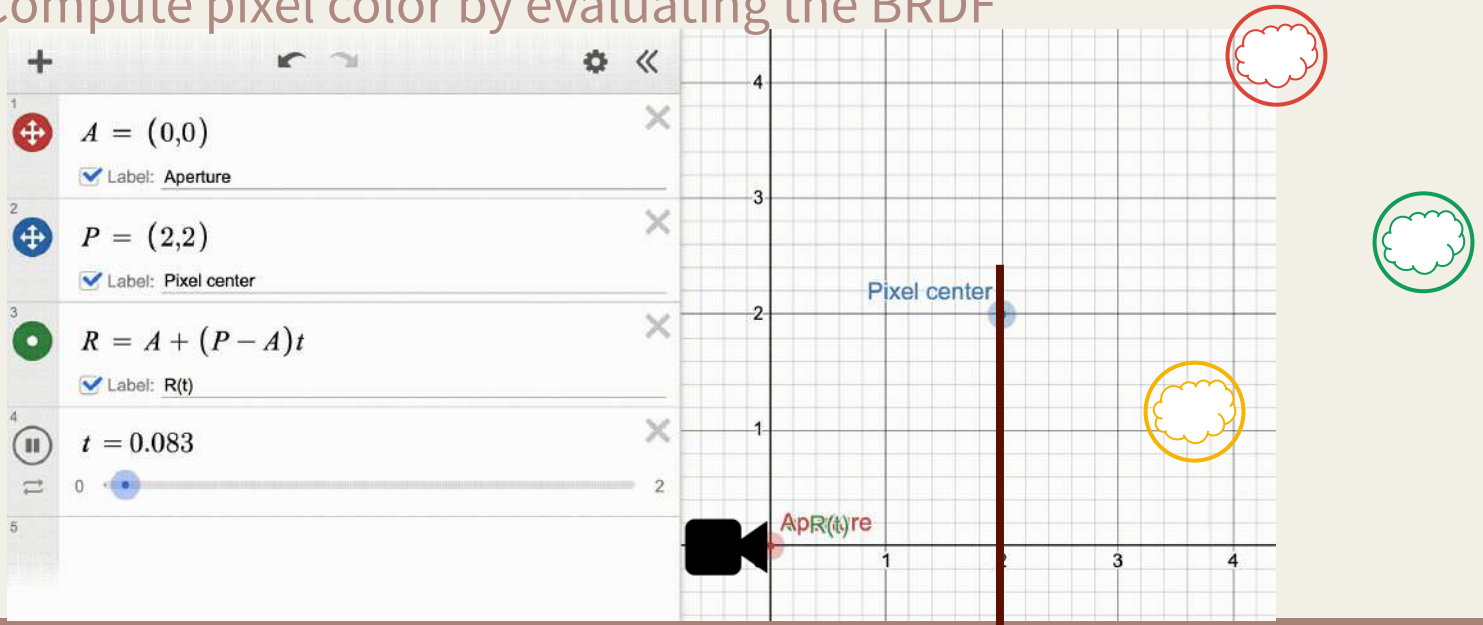
Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF



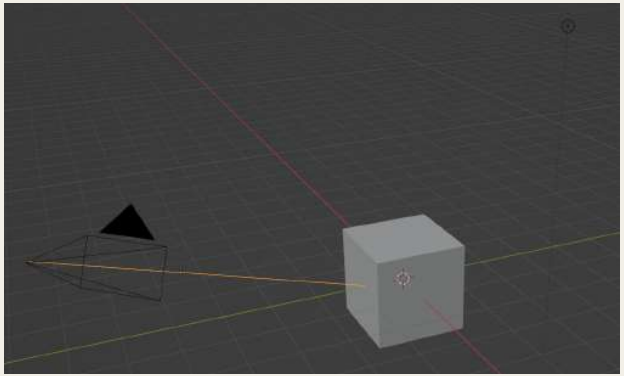
Raytracing Review

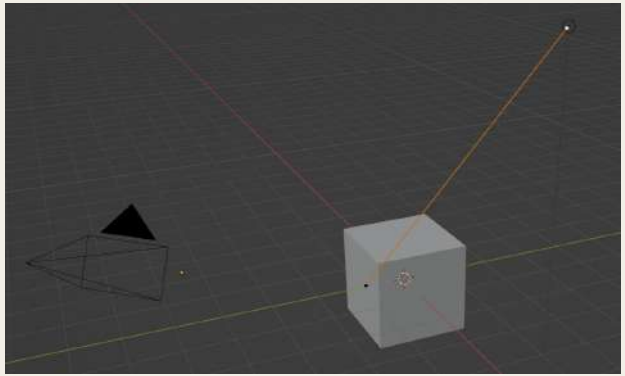
- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF

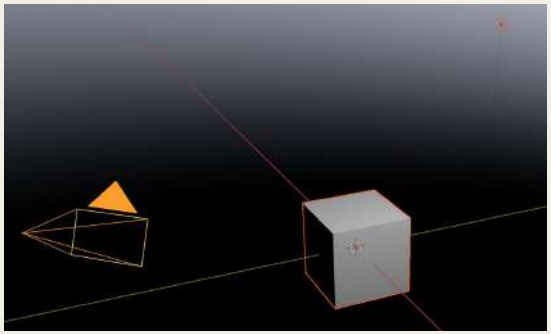


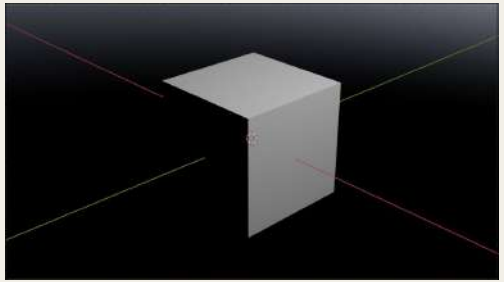
Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF



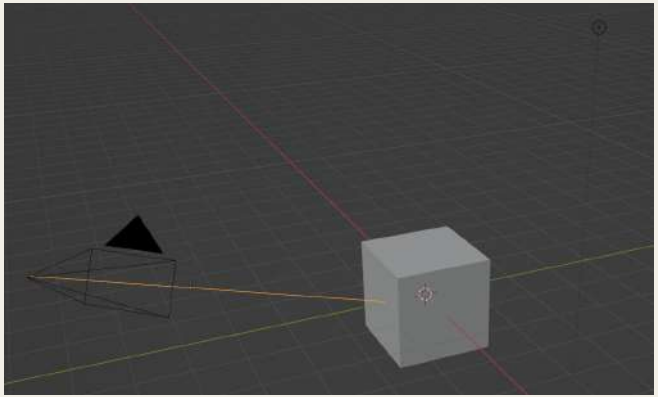


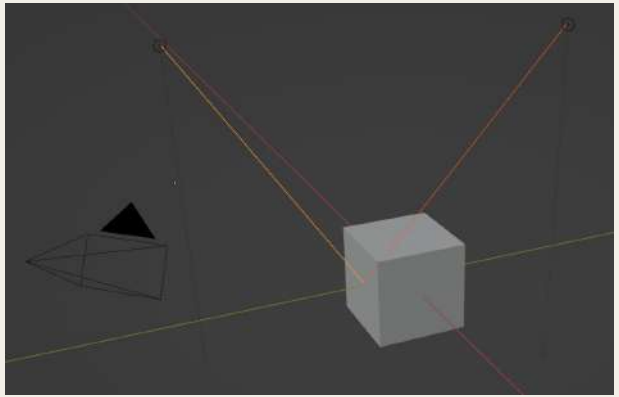


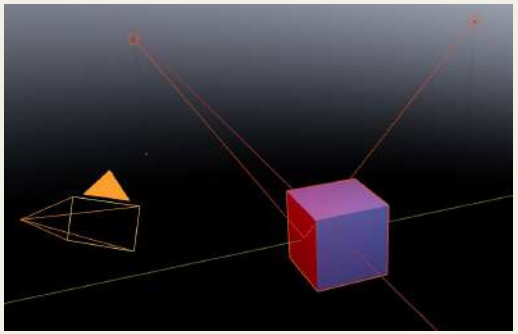


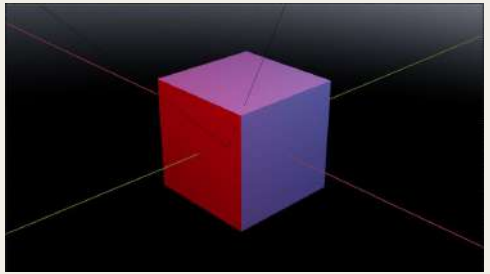
Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF



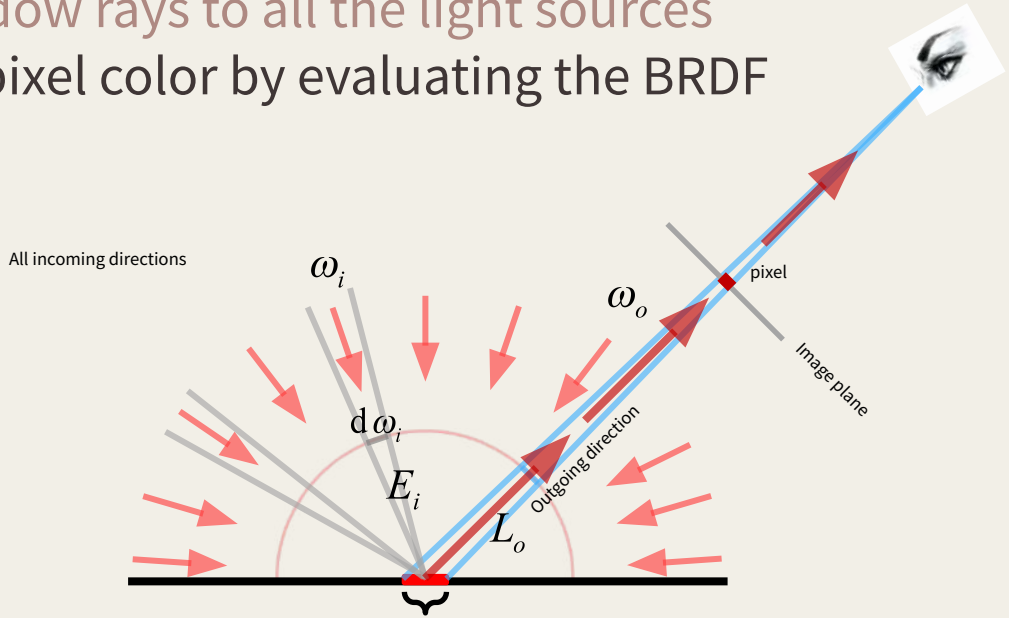






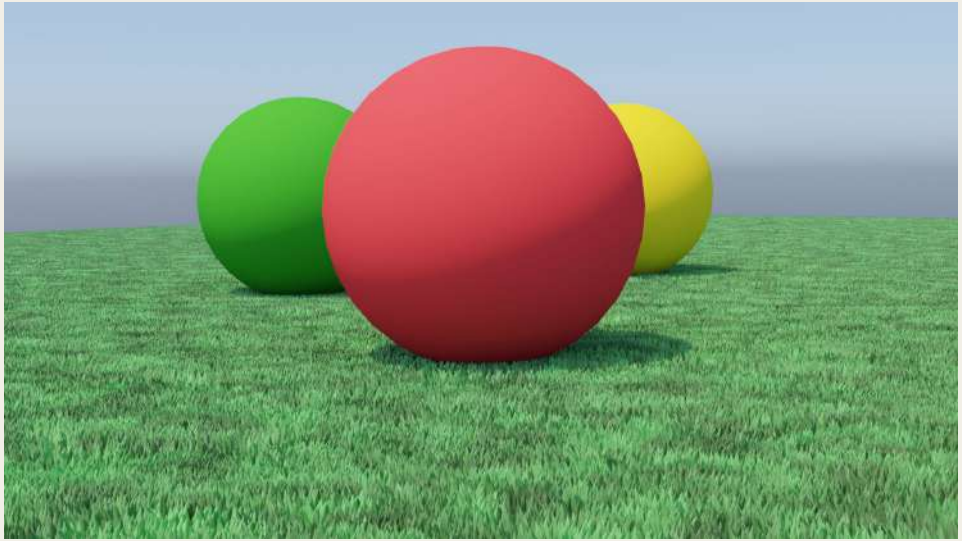
Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF



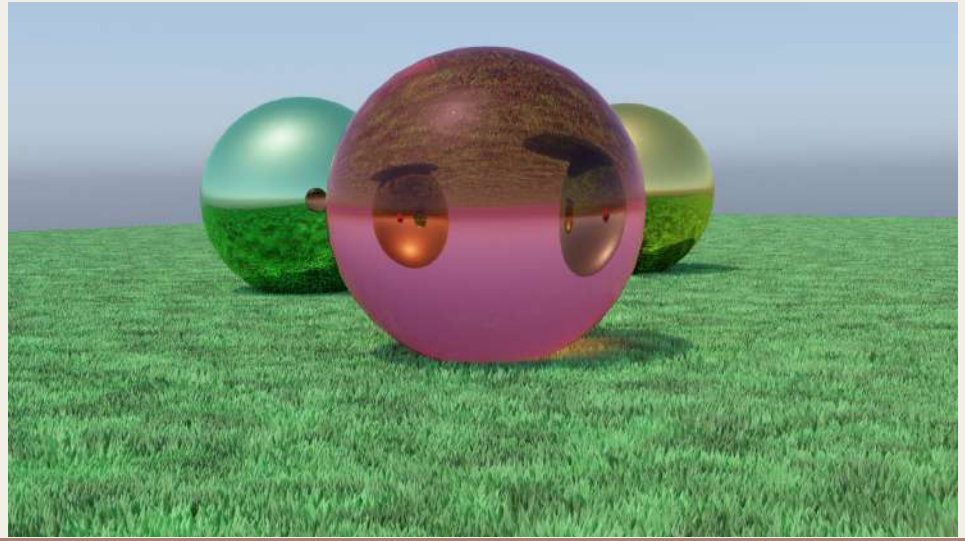
Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF



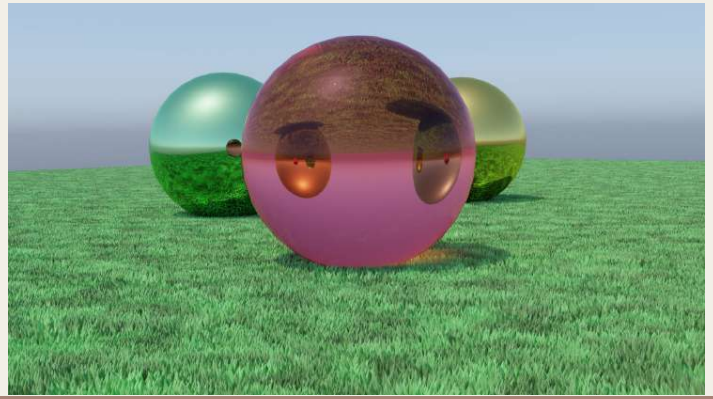
Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF



Raytracing Review

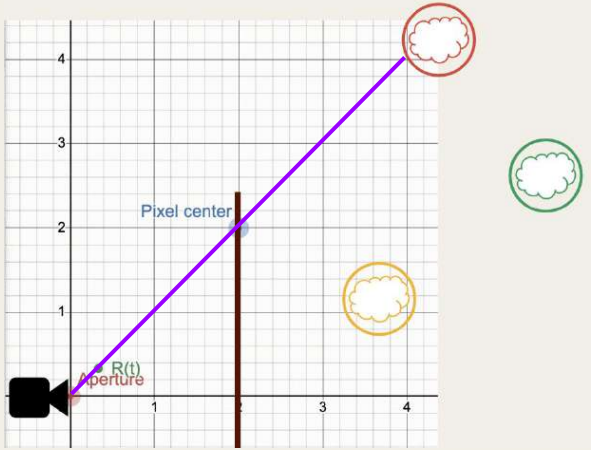
- For each pixel, shoot a camera ray
- Find the closest intersection of the camera ray with scene geometry
- Shoot shadow rays to all the light sources
- Shoot out new **transmission rays**
- Shoot out new **reflection rays**
- Compute pixel color by evaluating the BRDF



Raycasting - Review

```
function generate_image():
    for pixel_position on film_plane:
        camera_ray=generate_ray_through_pixel(pixel_position)
        color=trace(camera_ray, objects, lights)
        film[pixel_position]=color
```

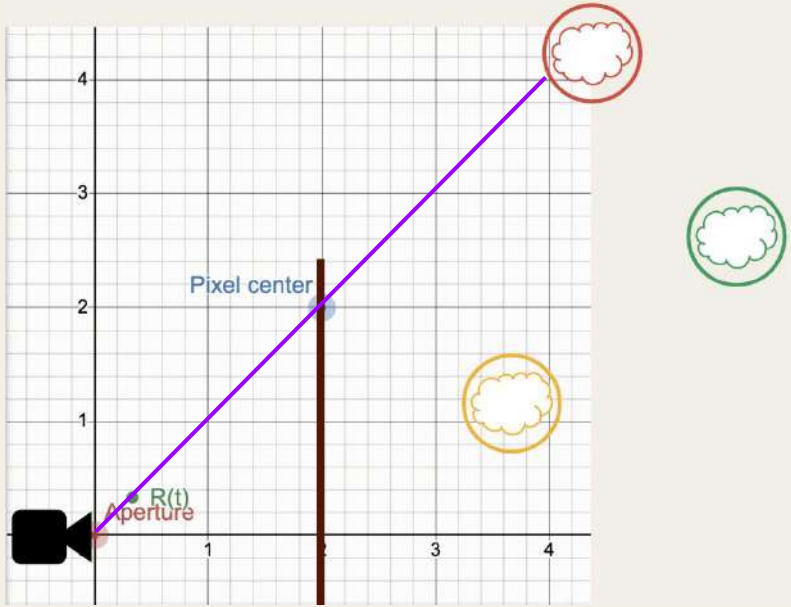
```
function trace(ray, objects, lights):
    intersection=ray_geometry_intersection(ray, objects)
    color=evaluate_rendering_equation(intersection, objects, lights)
    return color
```




```
function evaluate_rendering_equation(intersection, objects, lights):
    color=intersection.ambient
    for light in lights:
        shadow_ray=generate_ray_to_light(intersection, light)
        if does_not_hit_geometry(shadow_ray, objects):
            color+=BRDF(intersection.camera_ray,shadow_ray)*light.color*cos(theta)
    return color
```

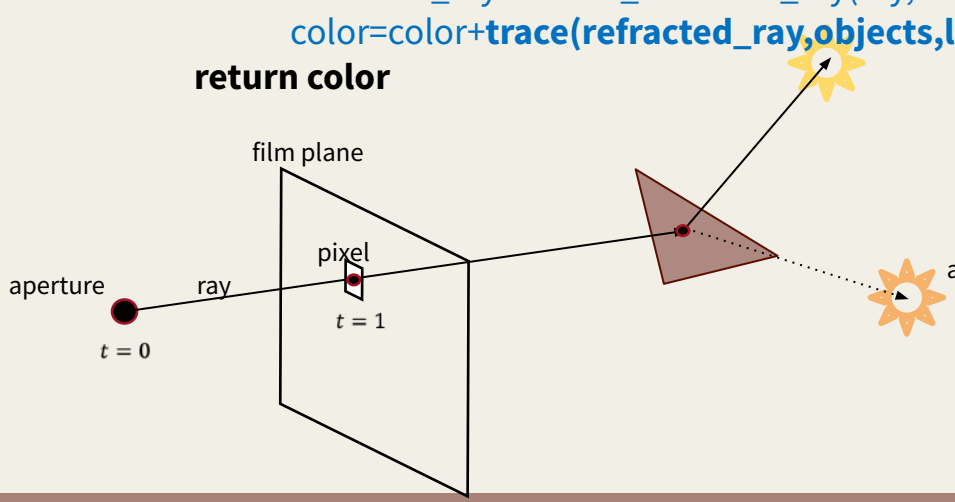
Recursive Raytracing

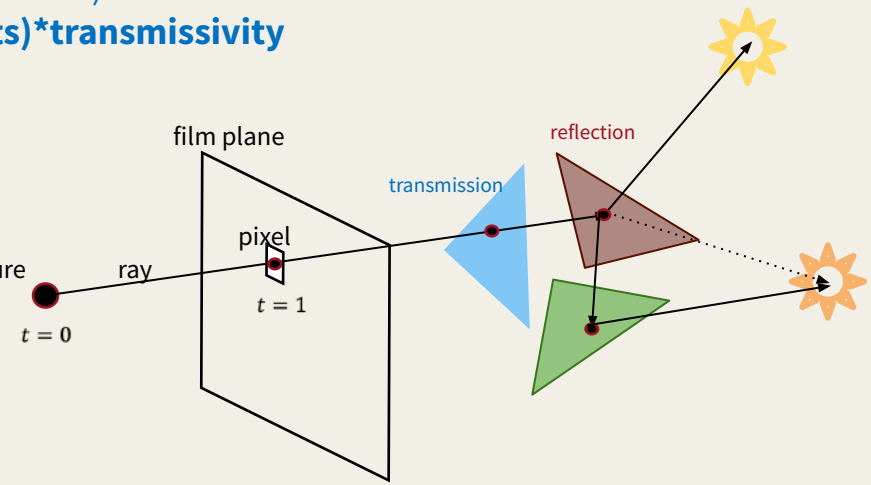
function trace(ray, objects, lights):
    intersection=ray_geometry_intersection(ray, objects)
    color=**compute_BRDF_at_intersection**(intersection, objects, lights)
    return color



Recursive Raytracing

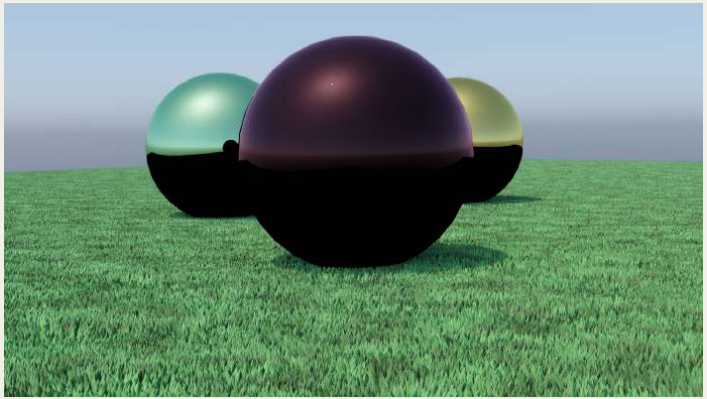
```
function trace(ray, objects, lights):
    intersection=ray_geometry_intersection(ray, objects)
    color=compute_BRDF_at_intersection(intersection, objects, lights)
    if (material at intersection has reflectivity>0)
        reflected_ray=create_reflected_ray(ray, intersection)
        color=color+trace(reflected_ray,objects,lights)*reflectivity
    if (material at intersection has transmissivity>0)
        refracted_ray=create _refracted_ray(ray, intersection)
        color=color+trace(refracted_ray,objects,lights)*transmissivity
    return color
```



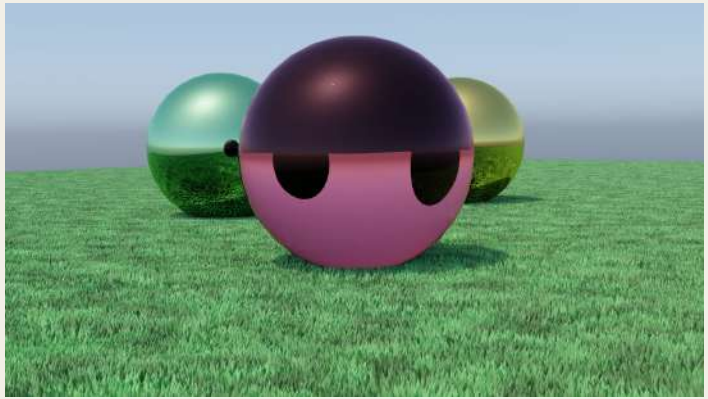


Recursive Raytracing

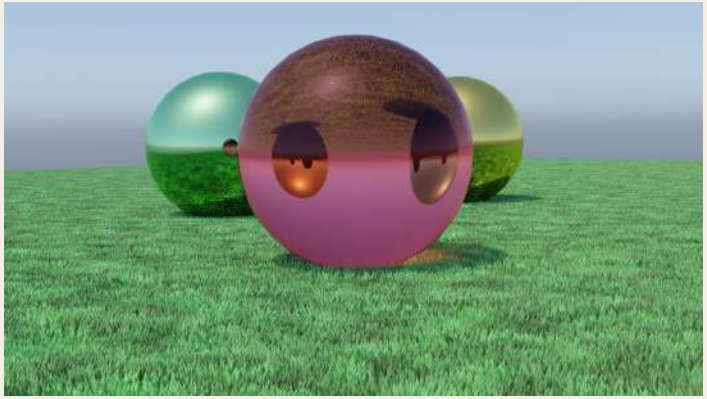
No recursions



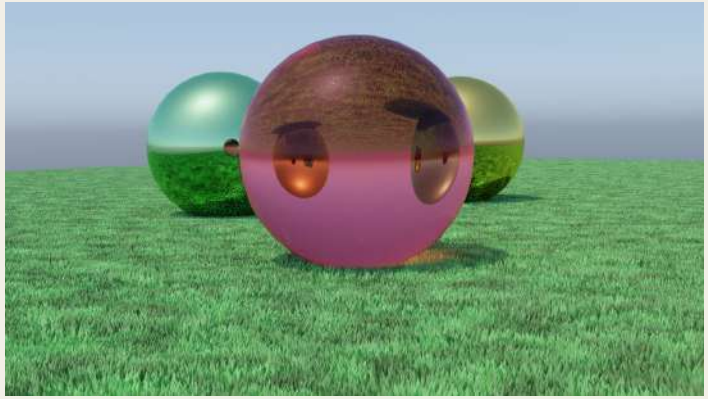
1 recursion bounce



2 recursion bounces

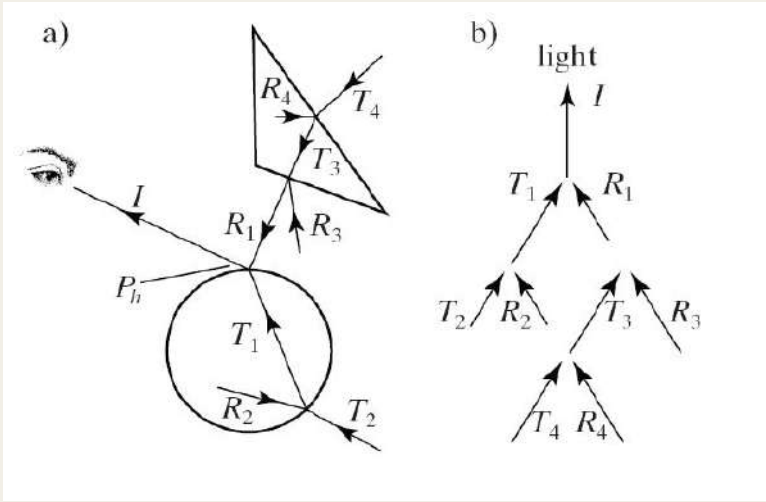


3 recursion bounces



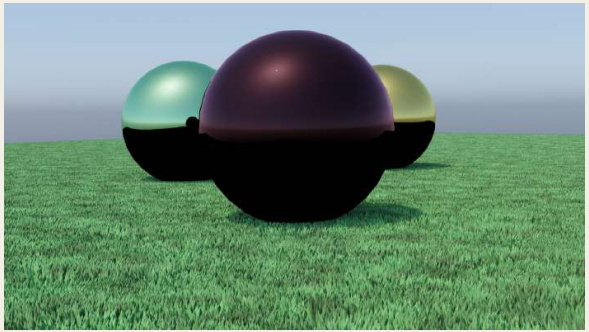
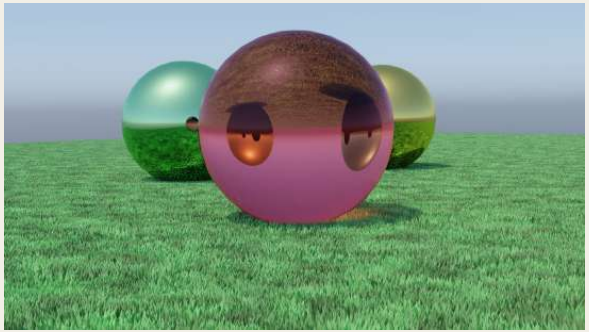
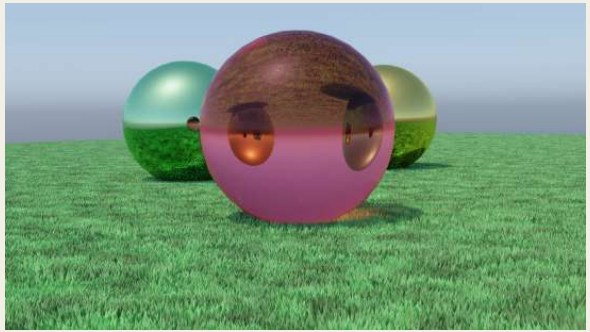
Implementation Detail - Termination

- To avoid infinite recursion/ stack overflow, keep track of how many bounces it's been.
- Terminate raytracing after certain recursion depth
- Large bounces doesn't just only happen with mirrors: Total internal reflection too!



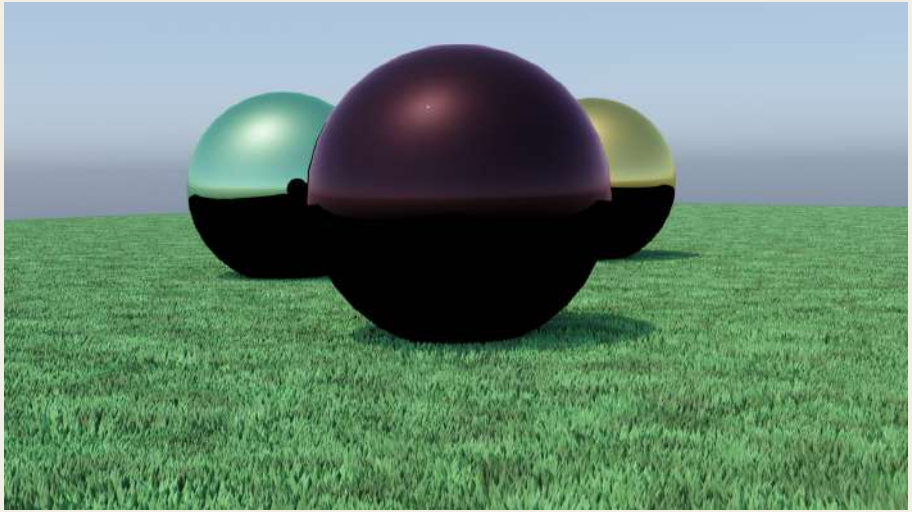
Implementation Detail - Termination

- To avoid infinite recursion/ stack overflow, keep track of how many bounces it's been.
- Terminate raytracing after certain recursion depth
- Large bounces doesn't just only happen with mirrors:
  Total internal reflection too!

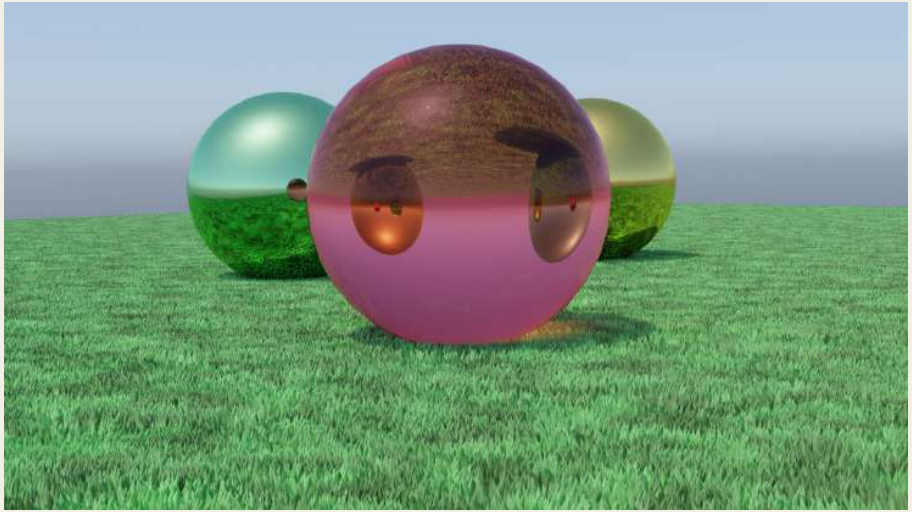




Recursive Raytracing

Max bounces = 0



Max bounces = 16



Lecture Outline

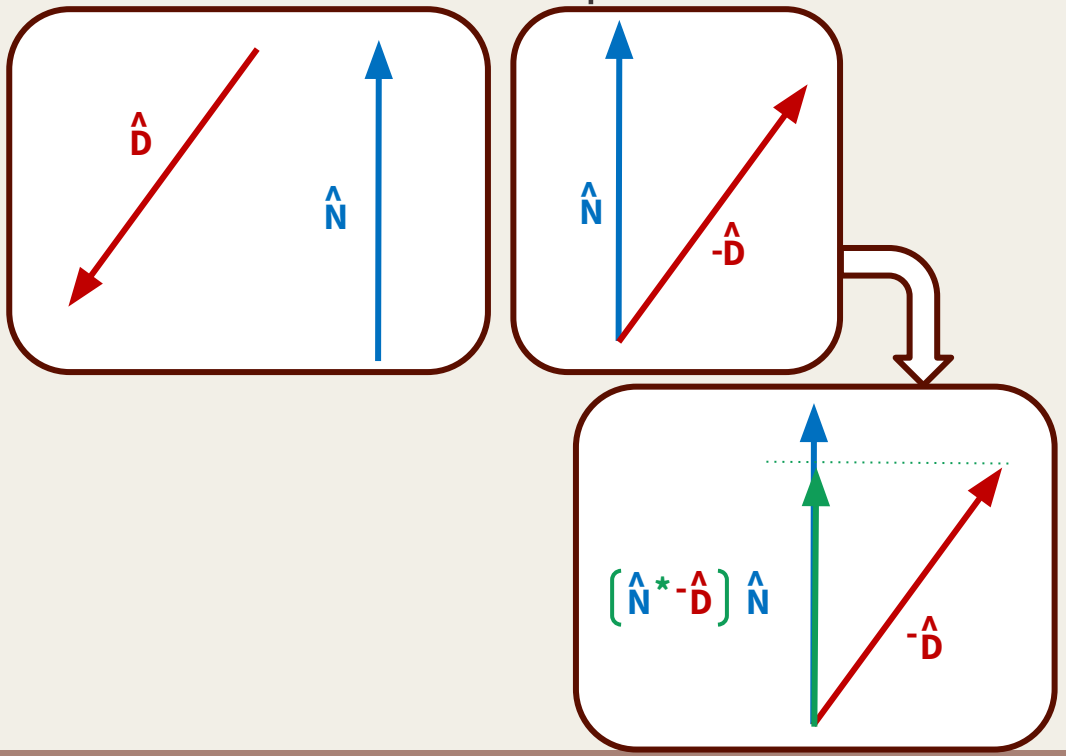
- Recursive Raytracing (Whitted raytracing)
  - Reflection
  - Refraction (Snell's Law)
  - Fresnel Equations
  - Attenuation(Beer's Law)

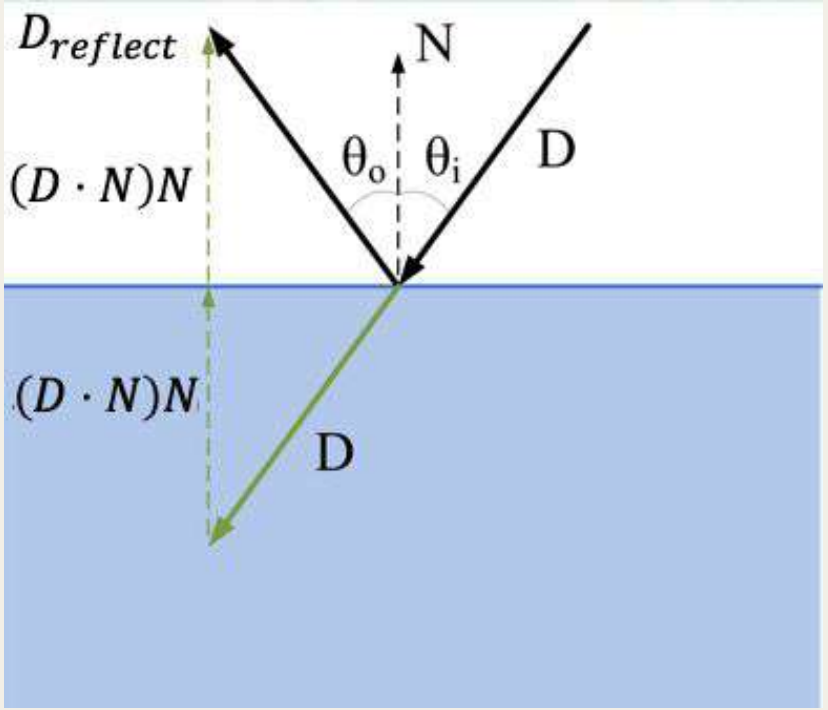
- Evaluating the Rendering Equation:
  - Beyond Blinn-Phong and point lights
  - Monte Carlo Methods
  - Stochastic raytracing (Area lights, depth of field, anti-aliasing)
  - Pathtracing (importance sampling, denoising)

Questions?

Reflected Rays

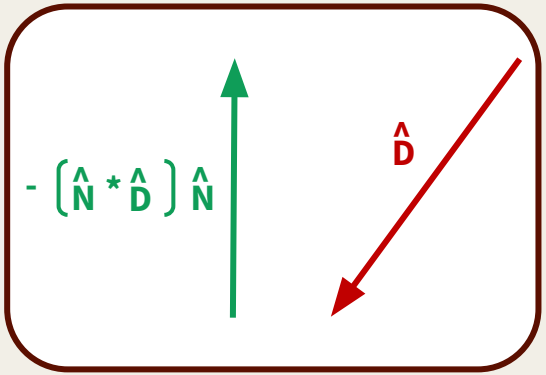
- How do we compute these?



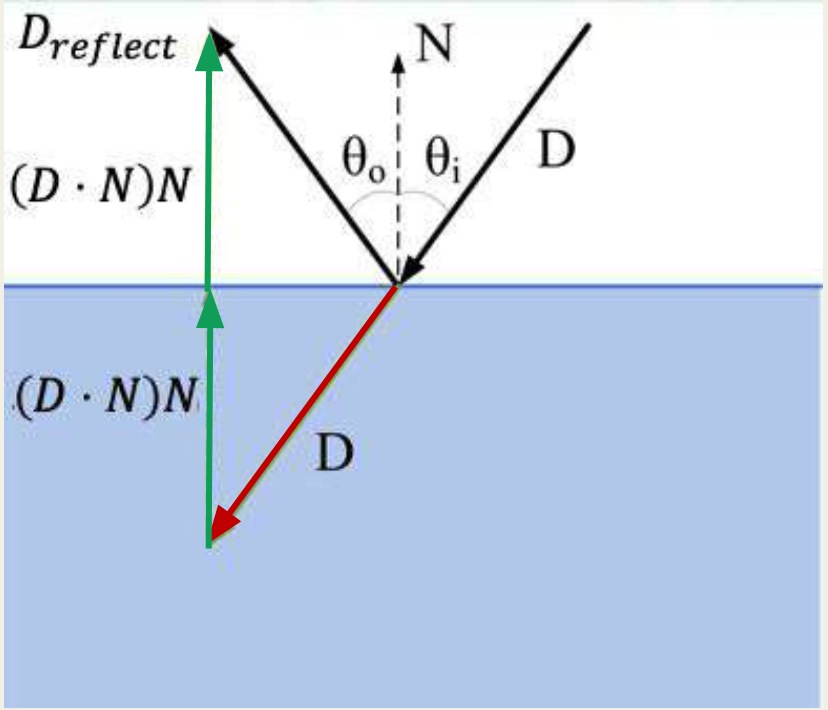


Reflected Rays

- How do we compute these?



$$\hat{D}_{reflect} = \hat{D} - 2\left(\hat{N} \cdot \hat{D}\right)\hat{N}$$

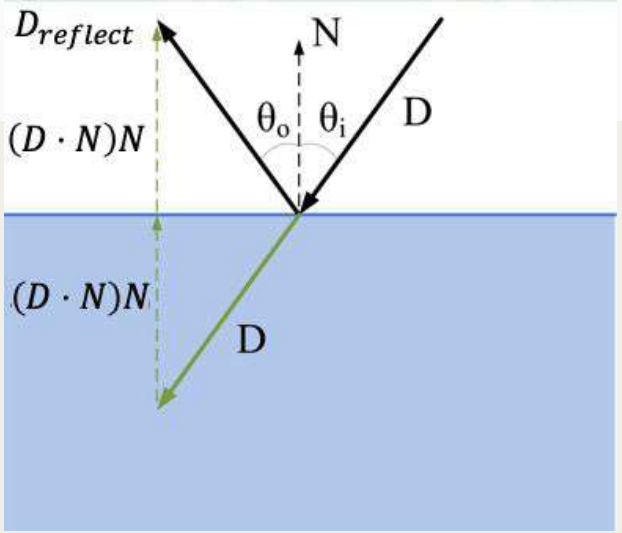


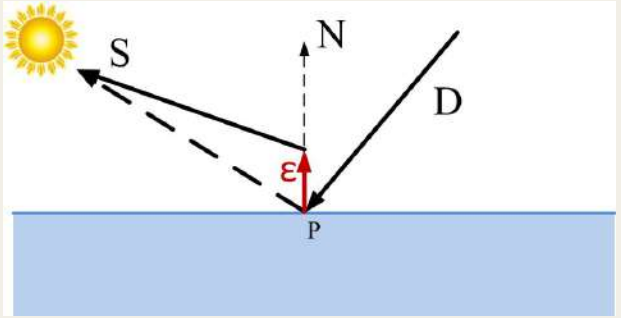
Reflected Rays

- Recall spurious self-intersection

$$\hat{D}_{reflect} = \hat{D} - 2\left(\hat{N} \cdot \hat{D}\right)\hat{N}$$

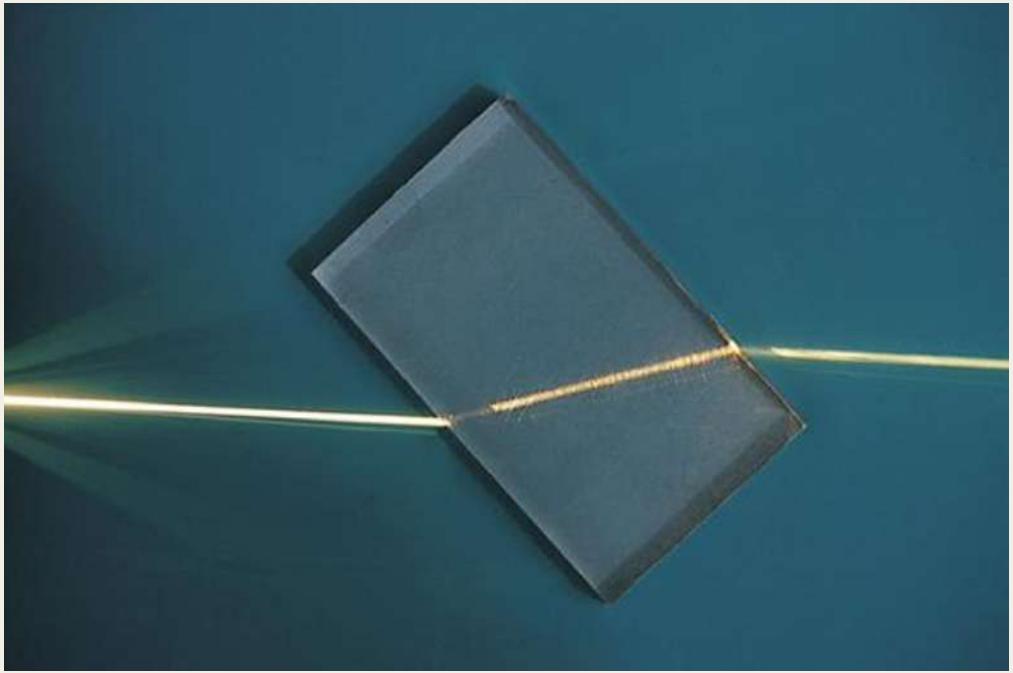
$$R_{reflect}(t) = \left(P + \epsilon\hat{N}\right) + \hat{D}_{reflect}\, t$$





Transmitted Ray

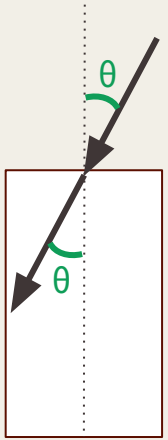
- Light bends as it passes through different materials



Transmitted Ray

- Light bends as it passes through different materials
- The relationship between the angle of incidence and transmittance is given by Snell's Law
- $\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$ where n is the index of refraction

Material	Index of Refraction (n)
Vacuum	1.000
Air	1.000277
Water	1.333333
Ice	1.31
Glass	About 1.5
Diamond	2.417

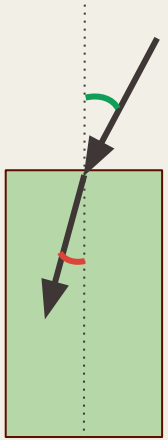


$$\frac{sin(\theta)}{sin(x)} = \frac{1}{1}$$

$$\frac{sin(\theta)}{sin(x)} = 1$$

$$sin(\theta) = sin(x)$$

$$x = \theta$$



$$\frac{sin(\theta)}{sin(x)} = \frac{1.5}{1}$$

$$\frac{sin(\theta)}{sin(x)} = 1.5$$

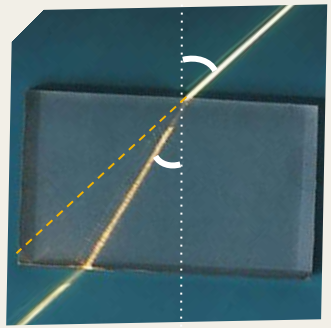
$$sin(x) = \frac{sin(\theta)}{1.5}$$

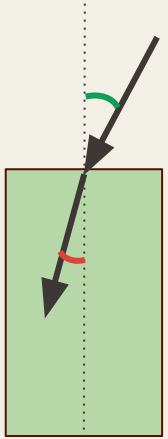
$$x = sin^{-1}(\frac{sin(\theta)}{1.5})$$

Transmitted Ray

- Light bends as it passes through different materials
- The relationship between the angle of incidence and transmittance is given by Snell's Law
- $\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$ where n is the index of refraction

Material	Index of Refraction (n)
Vacuum	1.000
Air	1.000277
Water	1.333333
Ice	1.31
Glass	About 1.5
Diamond	2.417





$$\frac{sin(\theta)}{sin(x)} = \frac{1.5}{1}$$

$$\frac{sin(\theta)}{sin(x)} = 1.5$$

$$sin(x) = \frac{sin(\theta)}{1.5}$$

$$x = sin^{-1}(\frac{sin(\theta)}{1.5})$$

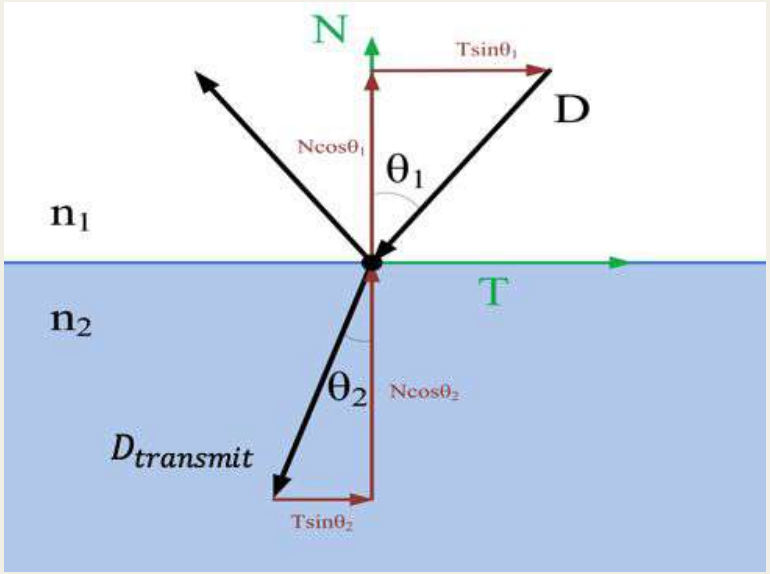
Transmitted Ray

$$\hat{D} = -\cos\theta_1 \hat{N} - \sin\theta_1 \hat{T}$$

$$\hat{D}_{transmit} = -\cos\theta_2 \hat{N} - \sin\theta_2 \hat{T}$$

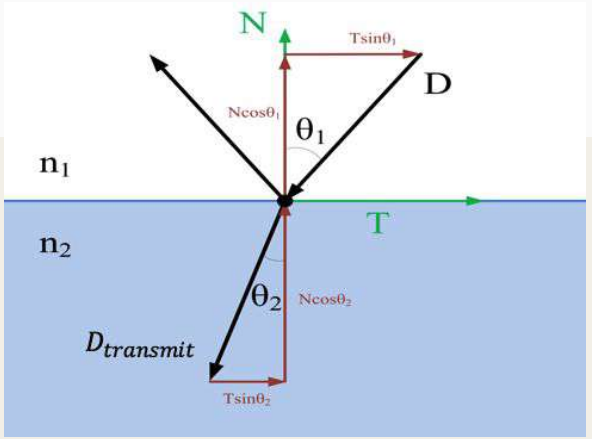
$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$$

All we have to do now is get rid of T and thetas - represent transmitted ray in terms of n1, n2, D, N



Transmitted Ray

$$\hat{D}_{transmit} = -\cos\theta_2 \hat{N} - \sin\theta_2 \hat{T}$$



- So $\hat{D}_{transmit} = -\hat{T}sin\theta_2 - \hat{N}cos\theta_2 = (\hat{D} + \hat{N}cos\theta_1)\frac{sin\theta_2}{sin\theta_1} - \hat{N}\sqrt{1 - \sin^2\theta_2}$
- Using Snell's Law, $\hat{D}_{transmit} = (\hat{D} + \hat{N}cos\theta_1)\frac{n_1}{n_2} - \hat{N}\sqrt{1 - \left(\frac{n_1}{n_2}sin\theta_1\right)^2}$
- $\hat{D}_{transmit} = \hat{D}\frac{n_1}{n_2} + \hat{N}\left(\frac{n_1}{n_2}cos\theta_1 - \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2(1 - \cos^2\theta_1)}\right)$
- Using $cos\theta_1 = -\hat{D}\cdot\hat{N}$ gives $\hat{D}_{transmit} = \hat{D}\frac{n_1}{n_2} - \hat{N}\left(\frac{n_1}{n_2}\hat{D}\cdot\hat{N} + \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2\left(1 - (\hat{D}\cdot\hat{N})^2\right)}\right)$

$$\hat{D} = -\cos\theta_1 \hat{N} - \sin\theta_1 \hat{T}$$

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$$

$$\cos\theta_1 = -\hat{N}\cdot\hat{D}$$

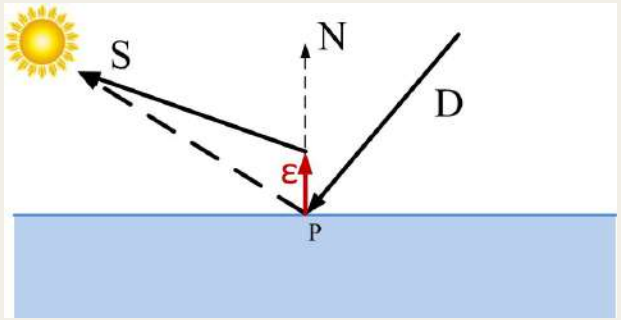
$$\hat{D}_{transmit} = \frac{n_1}{n_2}\hat{D} - \left(\frac{n_1}{n_2}\hat{N}\cdot\hat{D} + \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2\left(1 - \left(\hat{N}\cdot\hat{D}\right)^2\right)}\right)\hat{N}$$

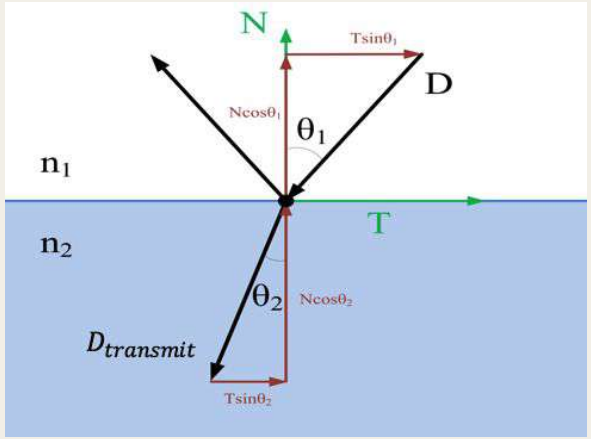
Transmitted Ray

- Recall spurious self-intersection (NOTE: flip the normals!)

$$\hat{D}_{transmit} = \frac{n_1}{n_2}\hat{D} - \left(\frac{n_1}{n_2}\hat{N}\cdot\hat{D} + \sqrt{1-\left(\frac{n_1}{n_2}\right)^2\left(1-\left(\hat{N}\cdot\hat{D}\right)^2\right)}\right)\hat{N}$$

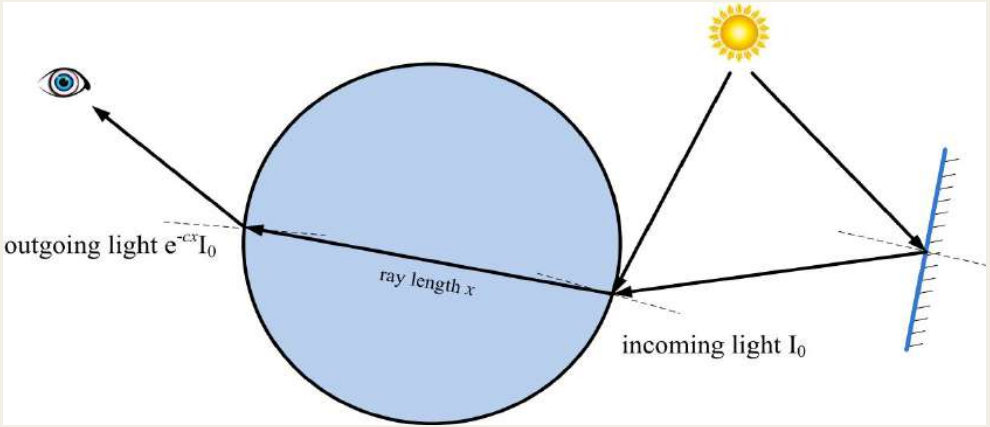
$$R_{transmit}(t) = \left(P - \epsilon\hat{N}\right) + \hat{D}_{transmit}\,t$$





Implementation Detail - Refraction

- Transmission: compute snell's law twice,
  once entering/ once exiting
- When computing intersection from inside an object:
- Flip the normals
- Make sure to check assumptions about intersection logic



Implementation Detail - Refraction

function trace(ray, objects, lights):

intersection=ray_geometry_intersection(ray, objects)

color=**compute_BRDF_at_intersection**(intersection, objects, lights)

if (material at intersection has reflectivity>0)

reflected_ray=create_reflected_ray(ray, intersection)

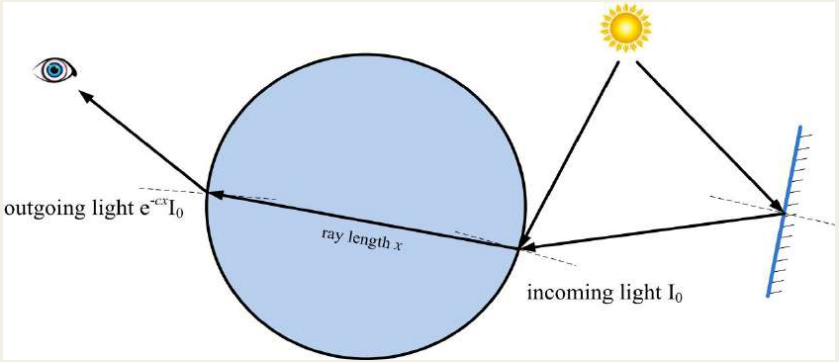
color=color+**trace(reflected_ray,objects,lights)*reflectivity**

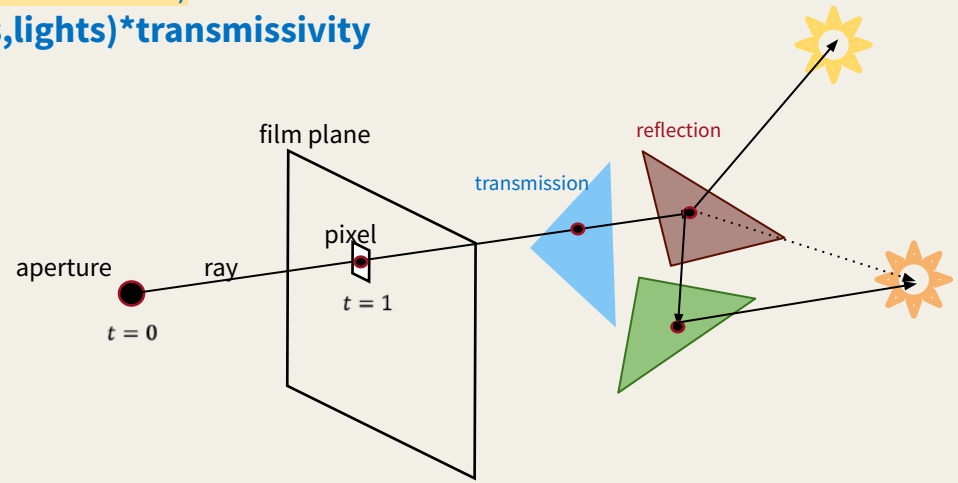
if (material at intersection has transmissivity>0)

refracted_ray=create _refracted_ray(ray, intersection) **<- compute exiting ray**

color=color+**trace(refracted_ray,objects,lights)*transmissivity**

return color





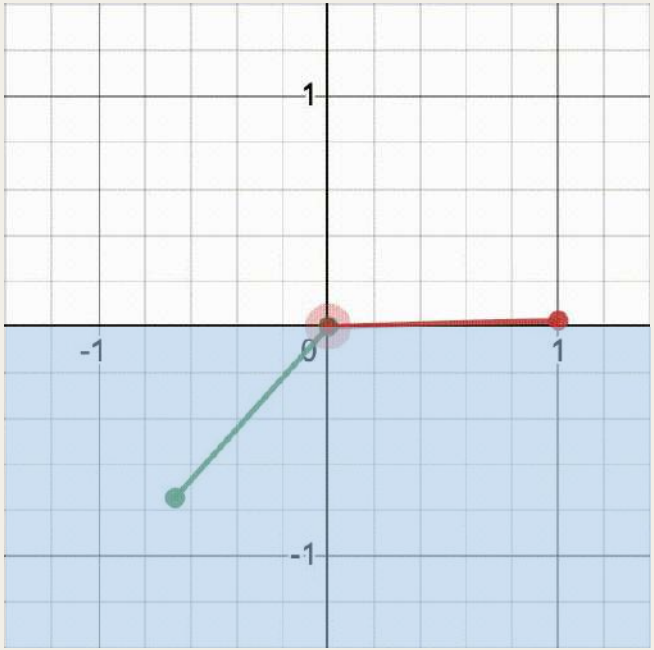
Transmitted Ray

$$\hat{D}_{transmit} = \frac{n_1}{n_2}\hat{D} - \left(\frac{n_1}{n_2}\hat{N}\cdot\hat{D} + \sqrt{1-\left(\frac{n_1}{n_2}\right)^2\left(1-\left(\hat{N}\cdot\hat{D}\right)^2\right)}\right)\hat{N}$$

- What happens when the square root term is negative?
- No transmission – **reflection** instead

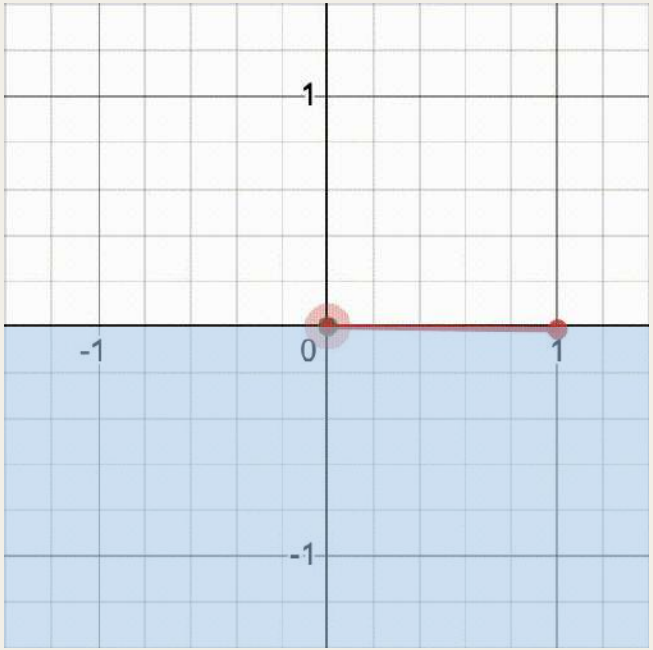
Total Internal Reflection

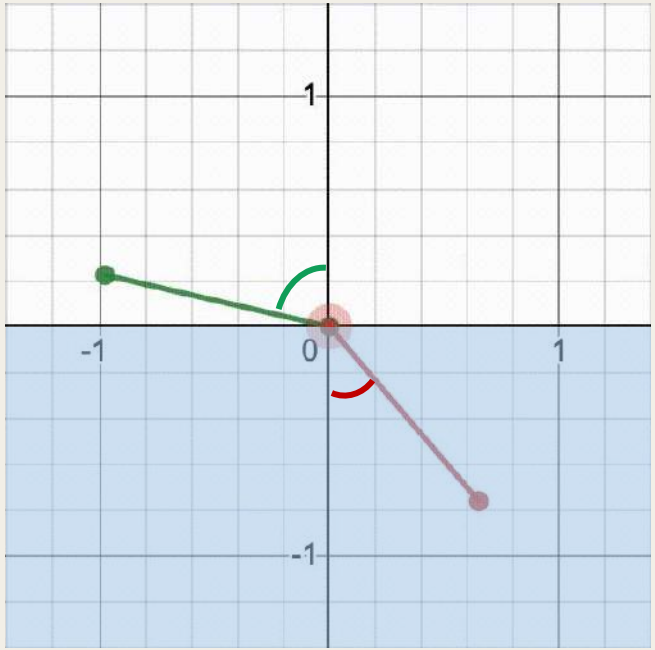
- Happens when we go from higher IOR to lower IOR
- The boundary angle is called the critical angle



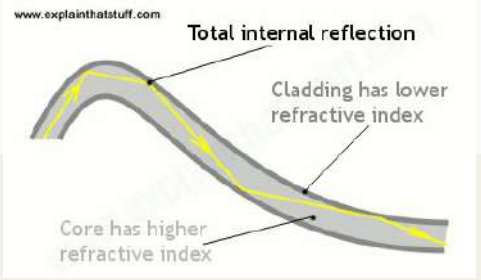
Total Internal Reflection

- Happens when we go from higher IOR to lower IOR
- The boundary angle is called the critical angle

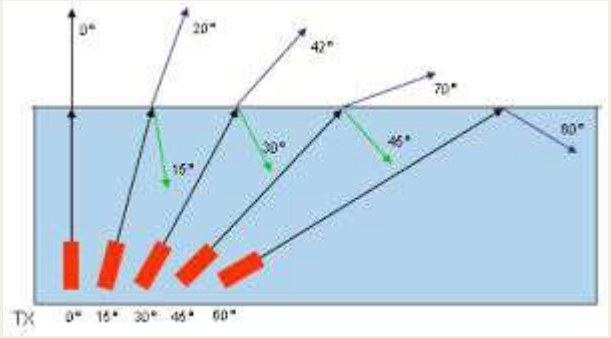


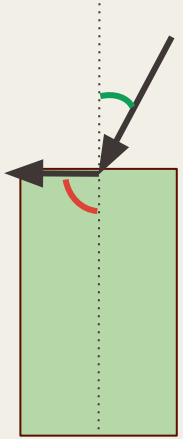


Total Internal Reflection



- Happens when we go from higher IOR to lower IOR
- The boundary angle is called the critical angle



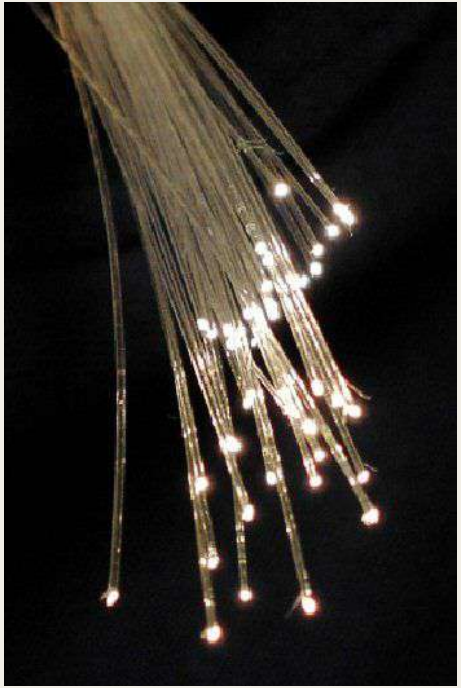


$$\frac{sin(\theta_1)}{sin(\theta_2)} = \frac{n_2}{n_1}$$

$$\frac{sin(\theta_1)}{sin(\frac{\pi}{2})} = \frac{n_2}{n_1}$$

$$sin(\theta_1) = \frac{n_2}{n_1}$$

$$\theta_1 \geq sin^{-1}(\frac{n_2}{n_1})$$



Transmitted Ray

$$\hat{D}_{transmit} = \frac{n_1}{n_2}\hat{D} - \left(\frac{n_1}{n_2}\hat{N}\cdot\hat{D} + \sqrt{1-\left(\frac{n_1}{n_2}\right)^2\left(1-\left(\hat{N}\cdot\hat{D}\right)^2\right)}\right)\hat{N}$$

- What happens when the square root term is negative?
- No transmission – **reflection** instead

Reflection vs Transmission

Objects are often simultaneously reflective and transmissive

Reflection vs Transmission

Objects are often simultaneously reflective and transmissive

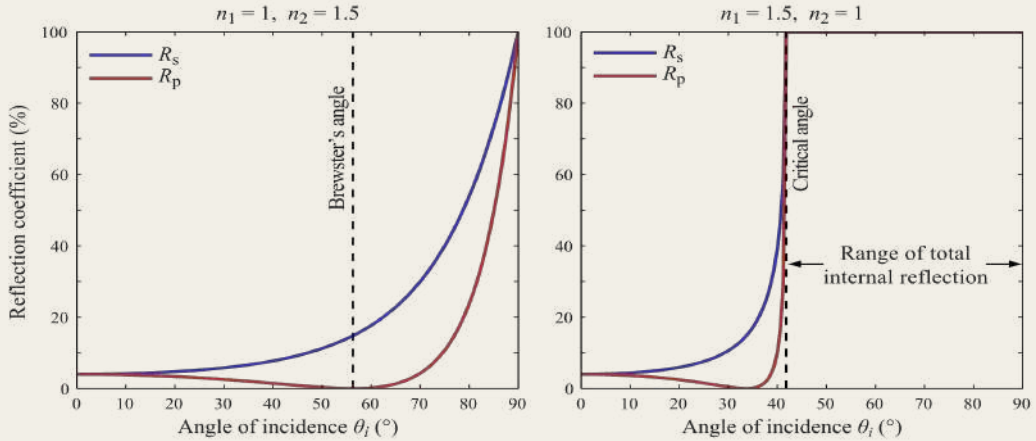
Amount of transmission vs. reflection decreases as the viewing angle goes from perpendicular (overhead) to parallel (grazing)

Reflection vs Transmission

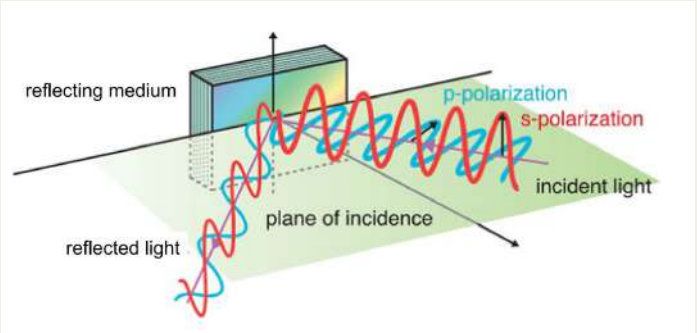
Reflection increases as the viewing angle goes from perpendicular (overhead) to parallel (grazing)

Reflection vs Transmission

- **reflection** increases as the viewing angle goes from perpendicular (overhead) to parallel (grazing) – this is determined by the **Fresnel equations**
- Interesting detail: light is polarized and behaves differently based on orientation relative to incident plane



Light entering a denser material (e.g. from air into water)



Reflection vs Transmission

- Light is polarized – behaves differently based on orientation relative to incident plane

$$R_p = \left| \frac{n_1 \cos\theta_t - n_2 \cos\theta_i}{n_1 \cos\theta_t + n_2 \cos\theta_i} \right|^2 \qquad R_s = \left| \frac{n_1 \cos\theta_i - n_2 \cos\theta_t}{n_1 \cos\theta_i + n_2 \cos\theta_t} \right|^2$$

- Transmission= usually amount of light that is not reflected

$$T_p = 1 - R_p \qquad T_s = 1 - R_s$$

- In rendering, we usually just take the average and treat light as unpolarized

$$R = \frac{R_p + R_s}{2} \qquad T = 1 - R$$

Schlick's Approximation

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

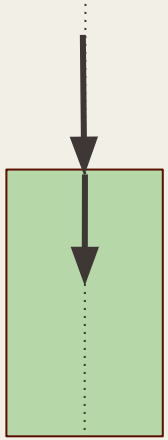
where

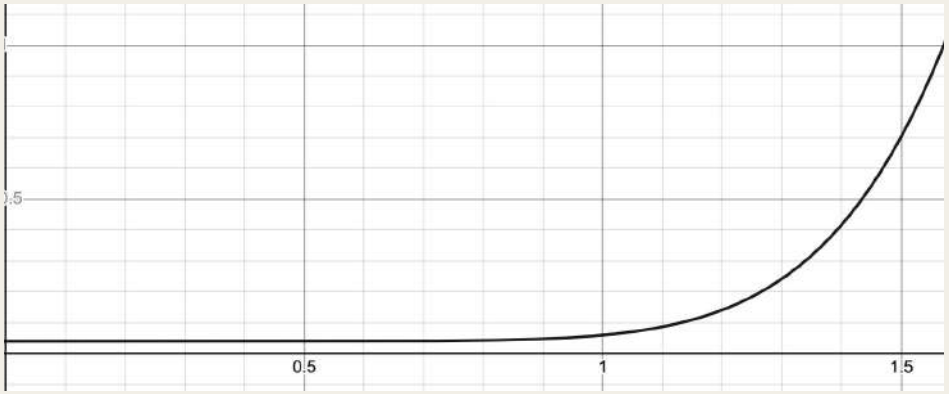
$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

$n_1 = 1.0$

$n_2 = 1.5$

$R_0 = .04$

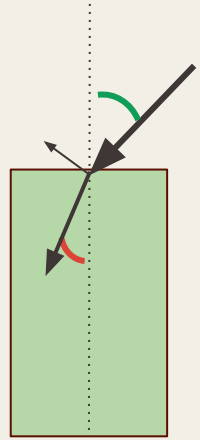




$n_1 = 1.0$

$n_2 = 1.5$

$R_0 = .04$



Reflection vs Transmission

Objects are often simultaneously reflective and transmissive

Amount of transmission vs. reflection decreases as the viewing angle goes from perpendicular (overhead) to parallel (grazing)

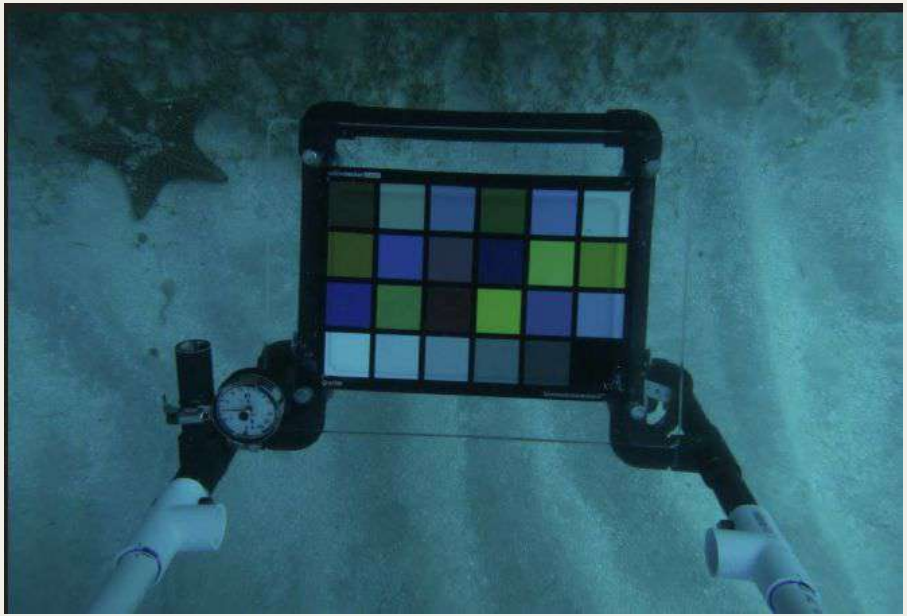
Attenuation

When light is transmitted through objects, some light may get absorbed/scattered

For example:

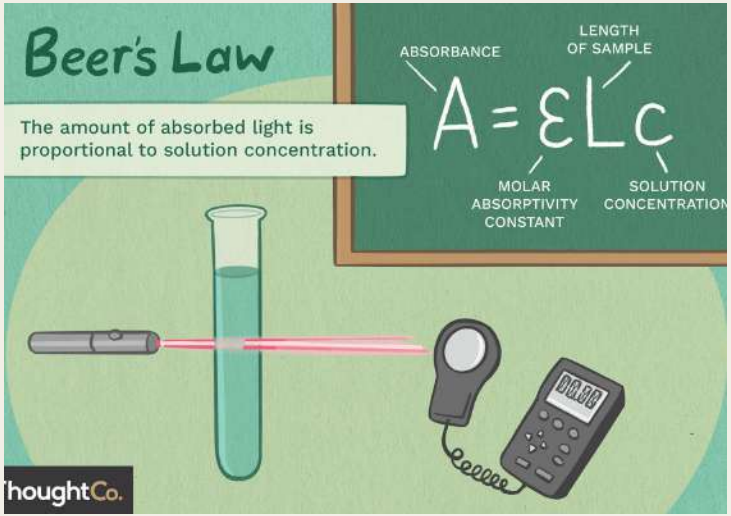
- Shallow water is clear (almost no attenuation)
- Deeper water attenuates all the red light and looks bluish-green
- Even deeper water attenuates all the green light too, and looks blue
- Eventually all the light attenuates, and the color ranges from blackish-blue to black

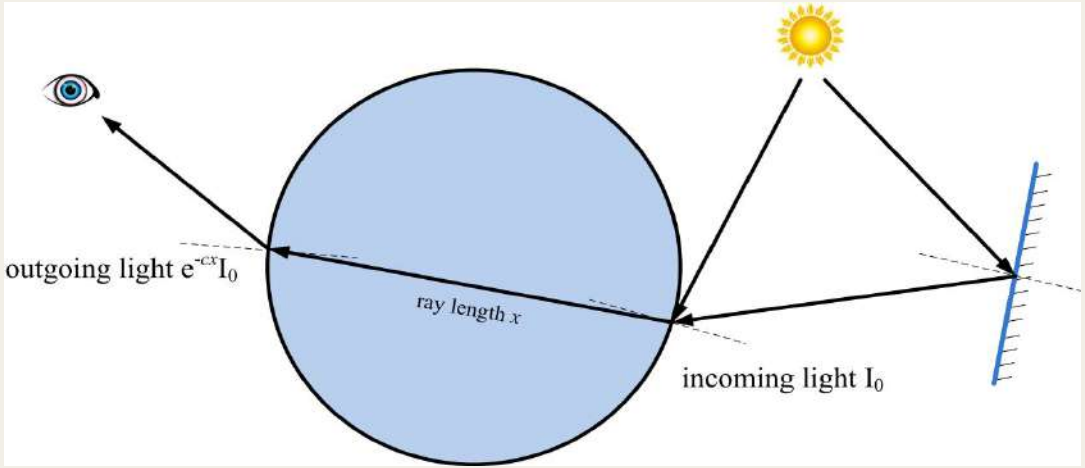
Attenuation



Attenuation

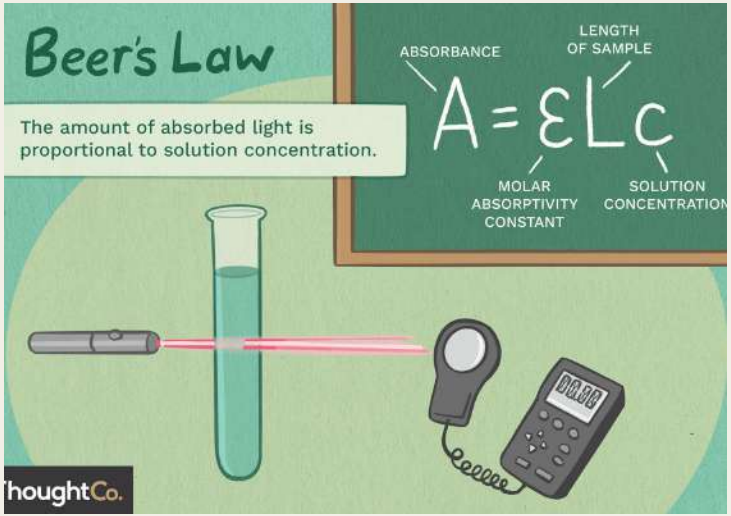
- If we assume the media is homogenous, attenuation along a ray can be described by Beer's Law

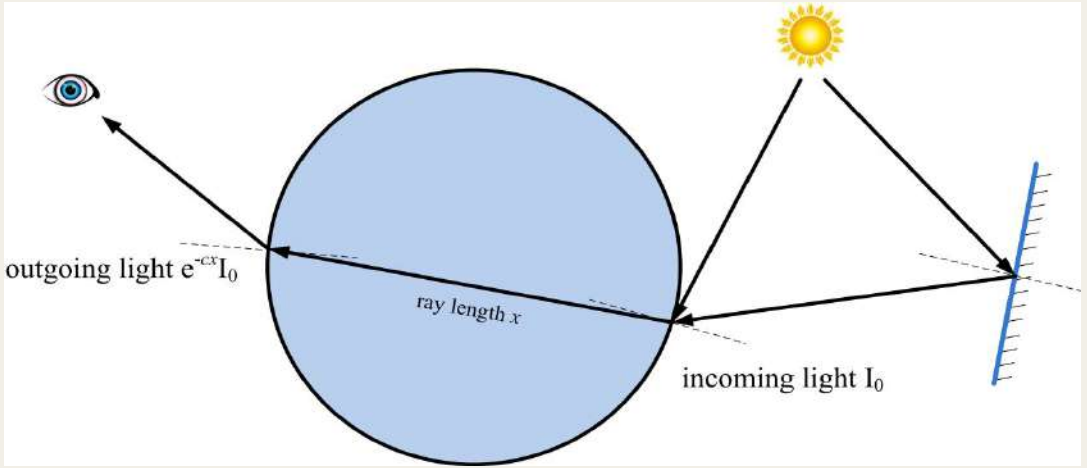




Attenuation

- c is 3d (rgb) coefficients of attenuation
- Note: For a blue object, what should the attenuation coefficients be?





Attenuation

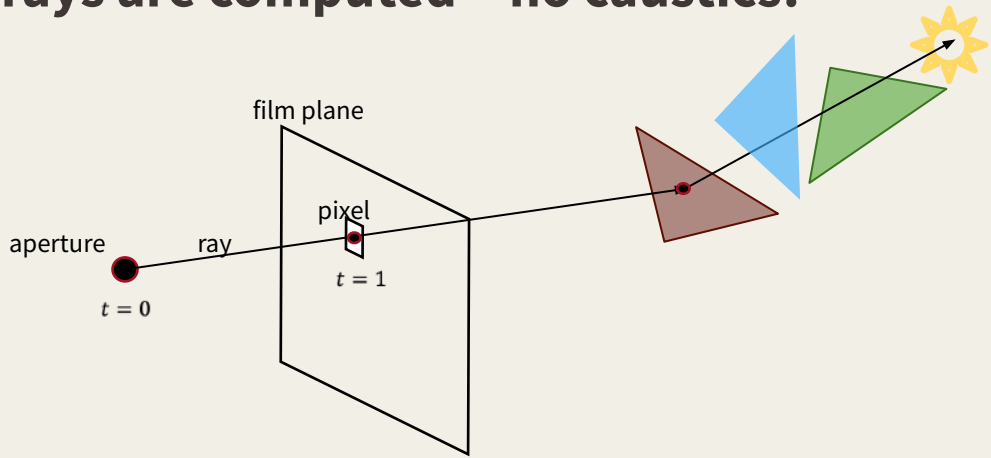
When light is transmitted through objects, some light may get absorbed/scattered

For example:

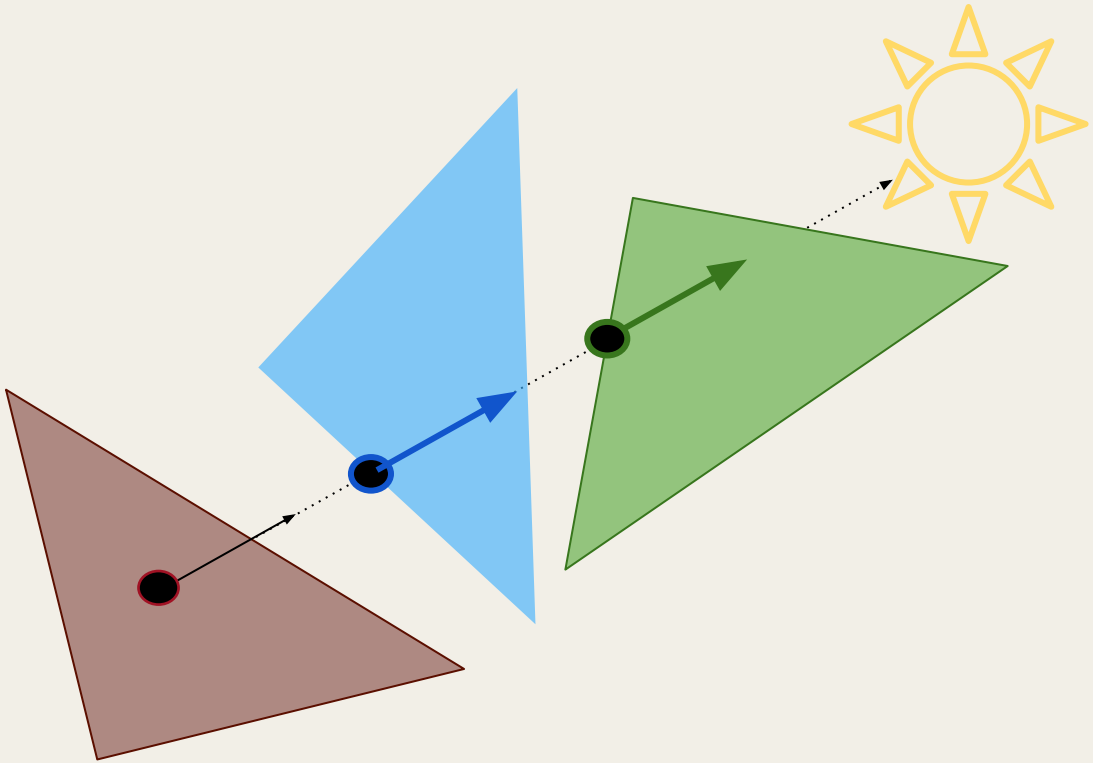
- Shallow water is clear (almost no attenuation)
- Deeper water attenuates all the red light and looks bluish-green
- Even deeper water attenuates all the green light too, and looks blue
- Eventually all the light attenuates, and the color ranges from blackish-blue to black

Attenuation: Shadows Rays…

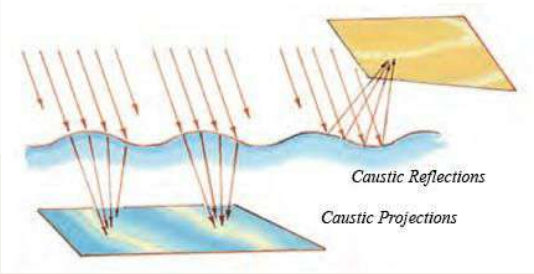
- Shadows of translucent objects can be approximated by attenuation.
- If we assume the media is homogenous, attenuation along a ray can be described by Beer's Law
- **But note: the way shadow rays are computed = no caustics!**



Attenuation: Shadows Rays…



Caustics

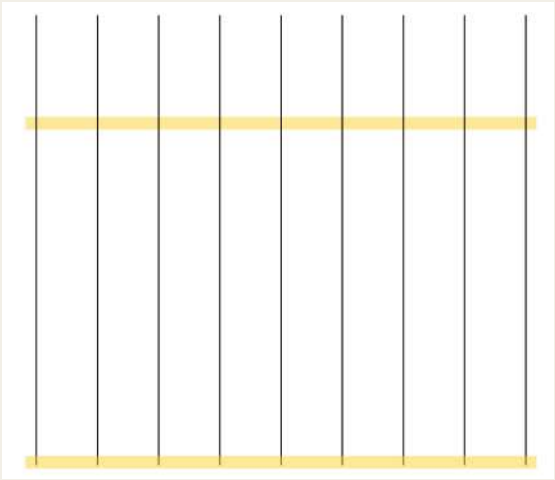


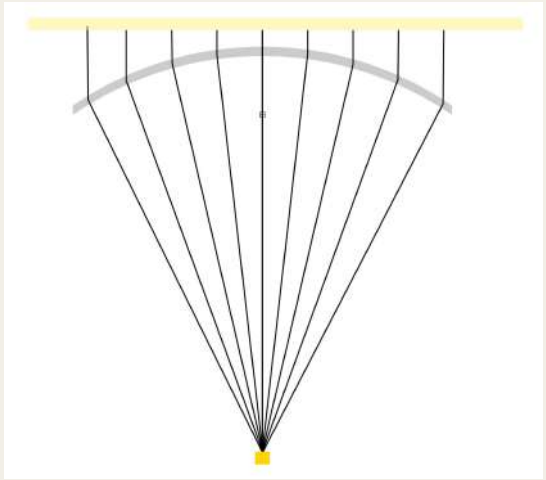
- Caused by refraction
- Require not just “straight line shadow rays” to each light source
- Light refracts before reaching the surface

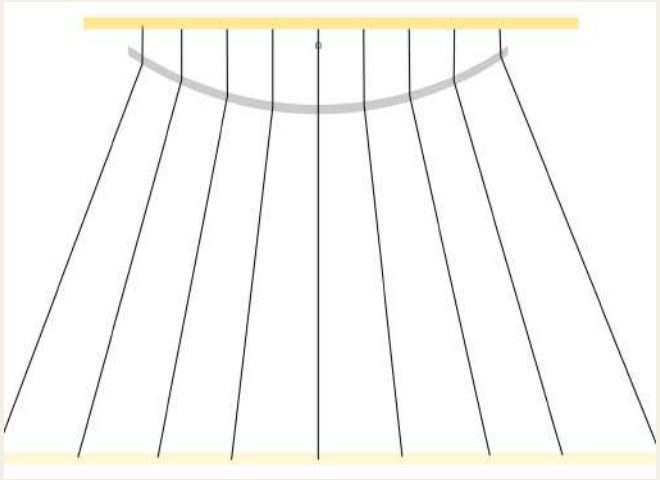
Caustics

- Caused by refraction
- Require not just “straight line shadow rays” to each light source
- Light refracts before reaching the surface

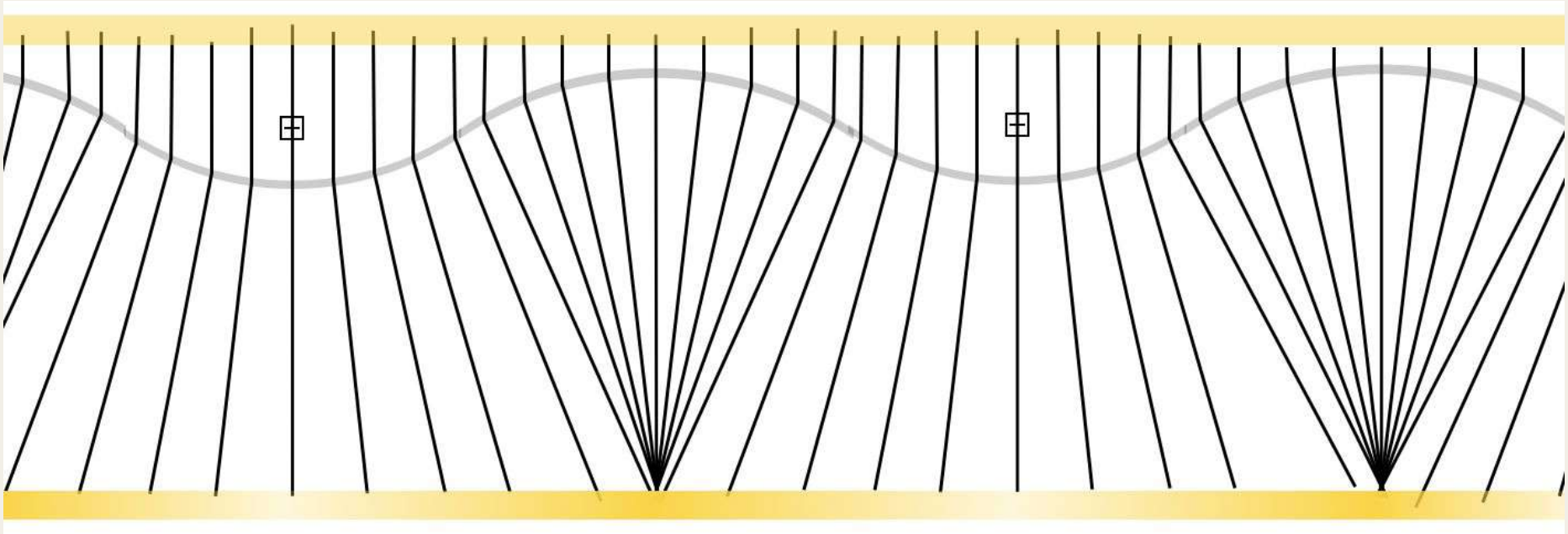
Caustics



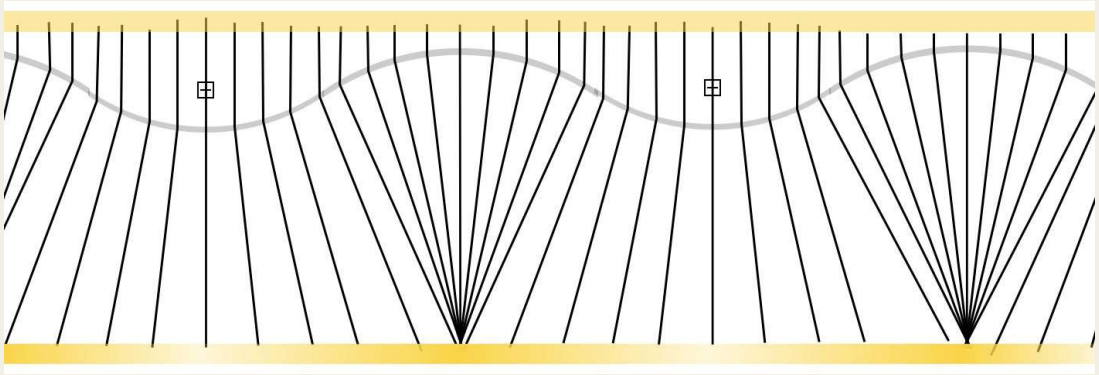


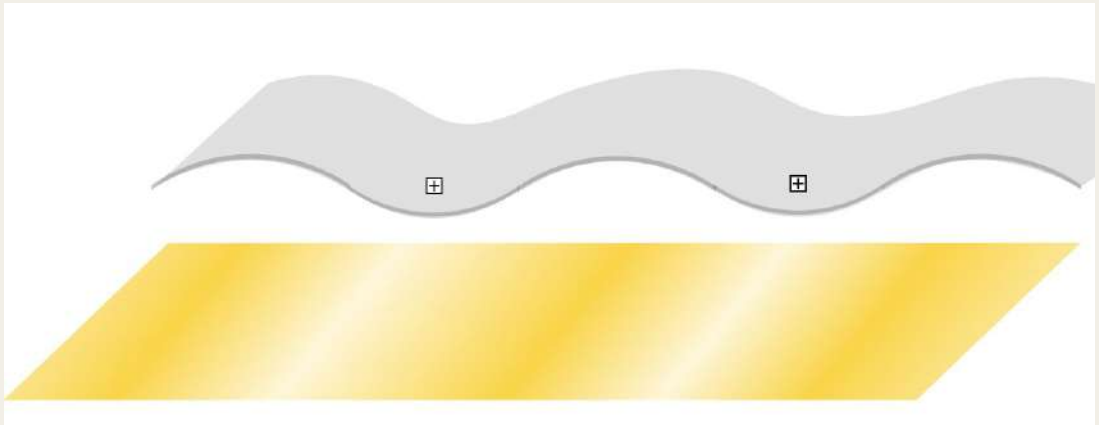


Caustics

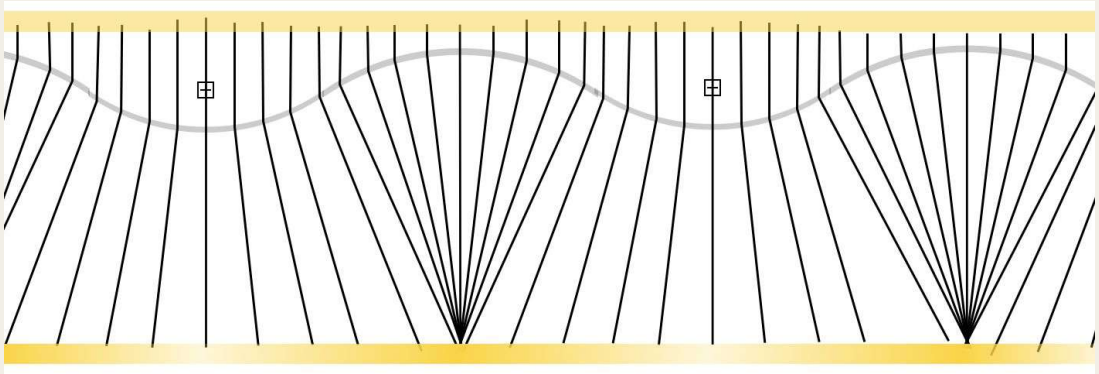


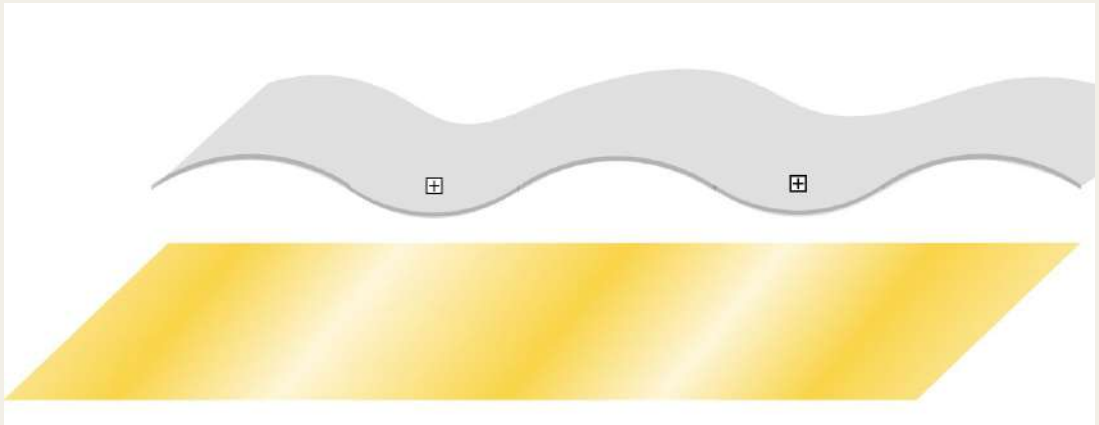
Caustics

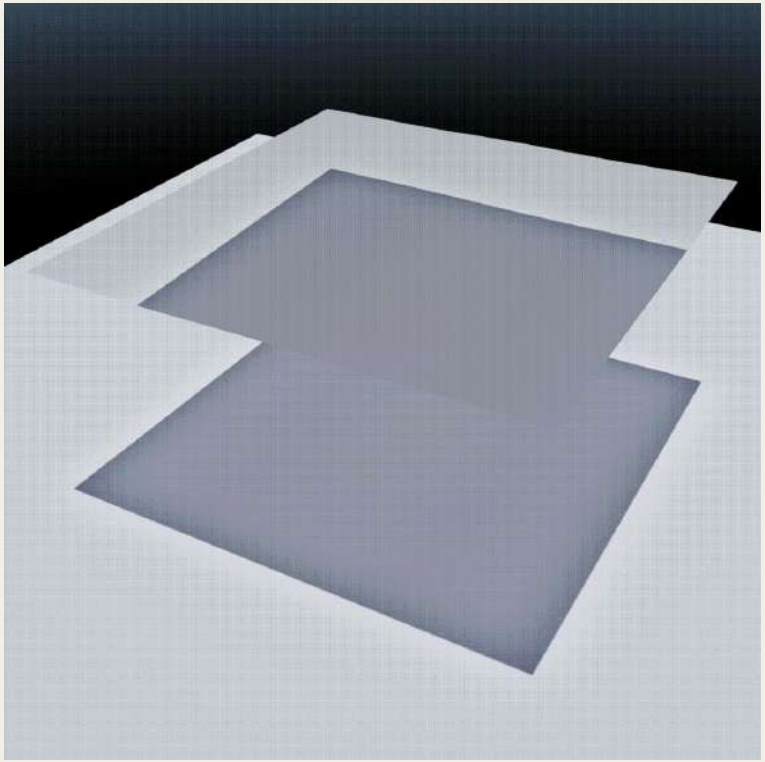




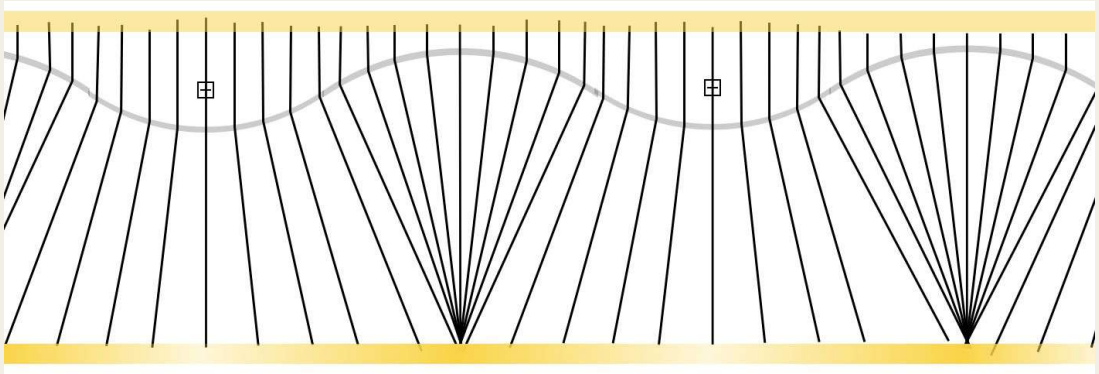
Caustics

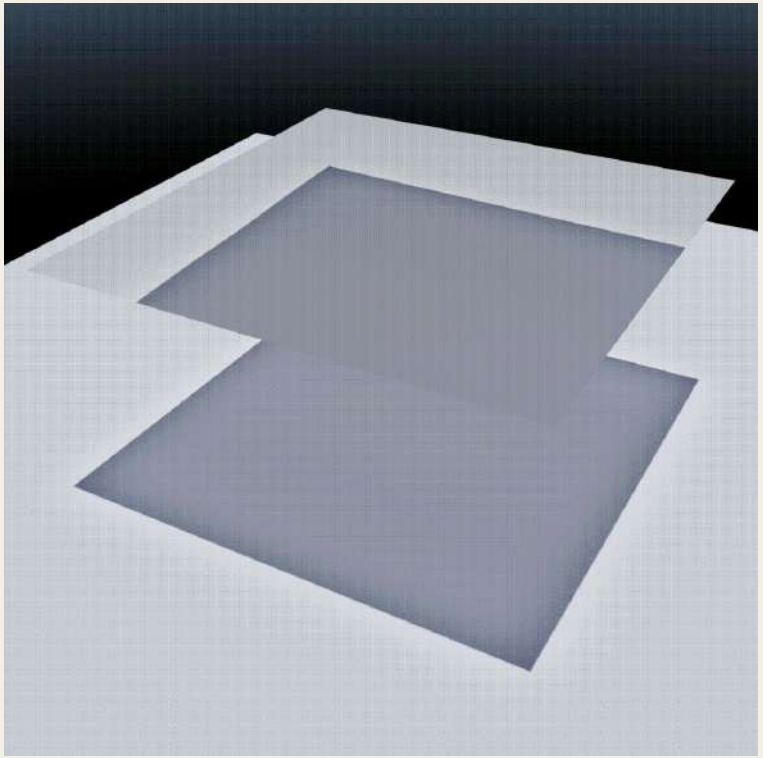




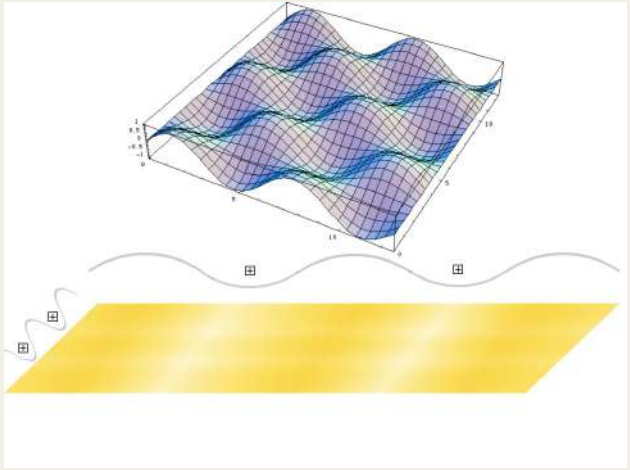


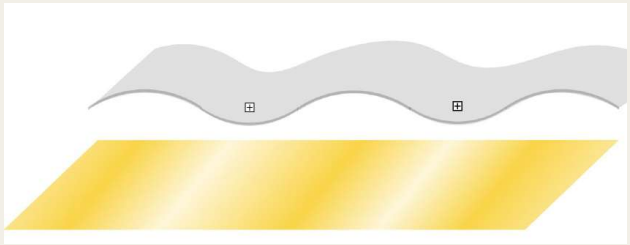
Caustics

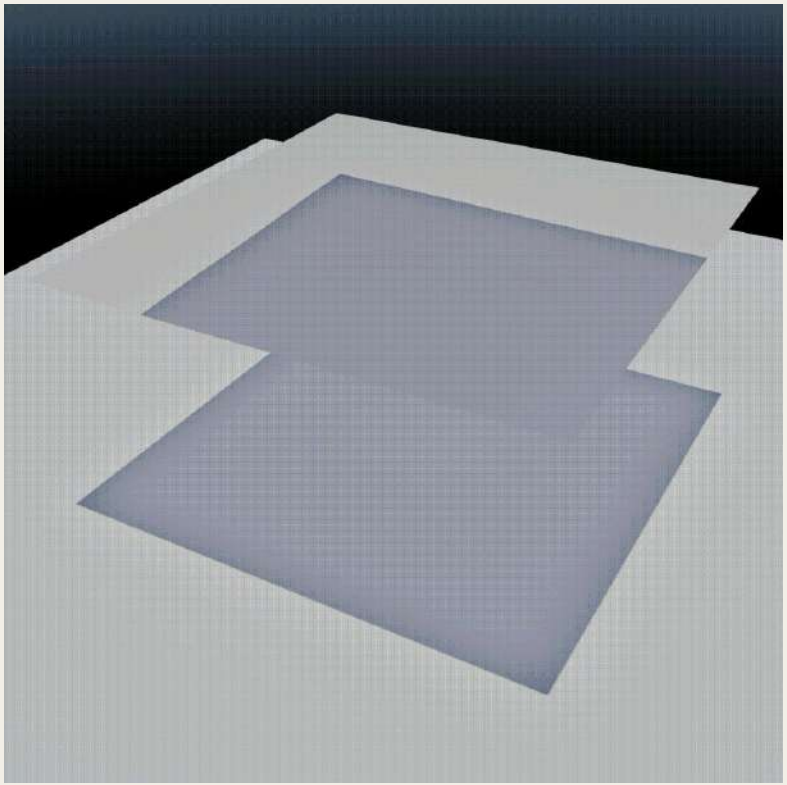




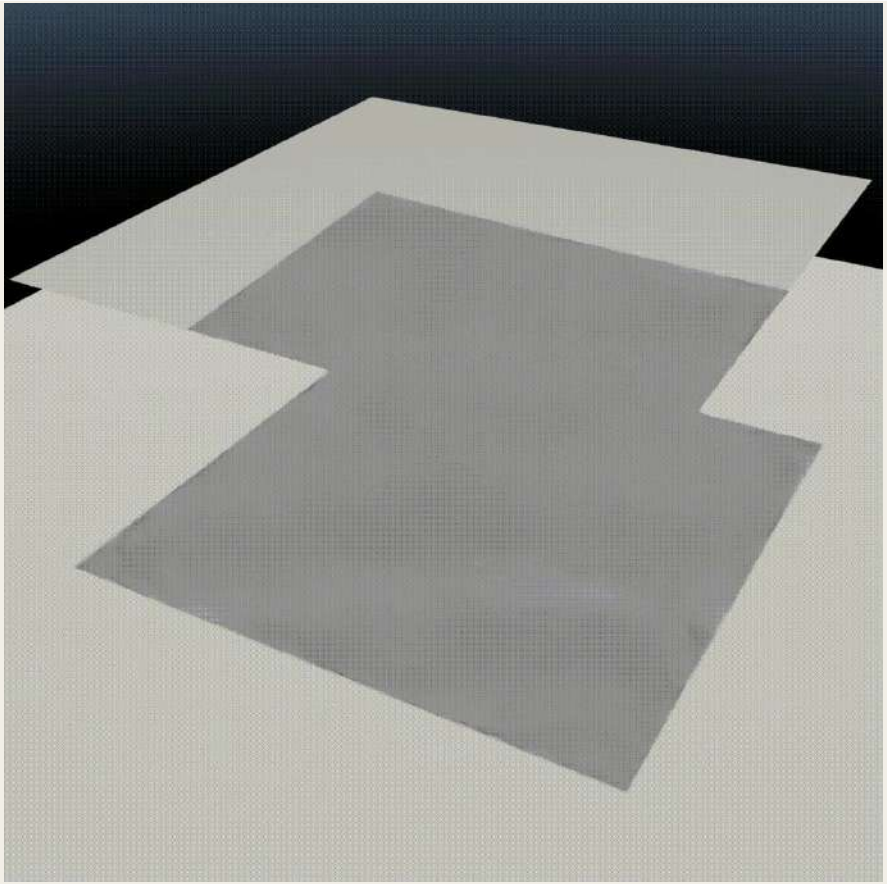
Caustics





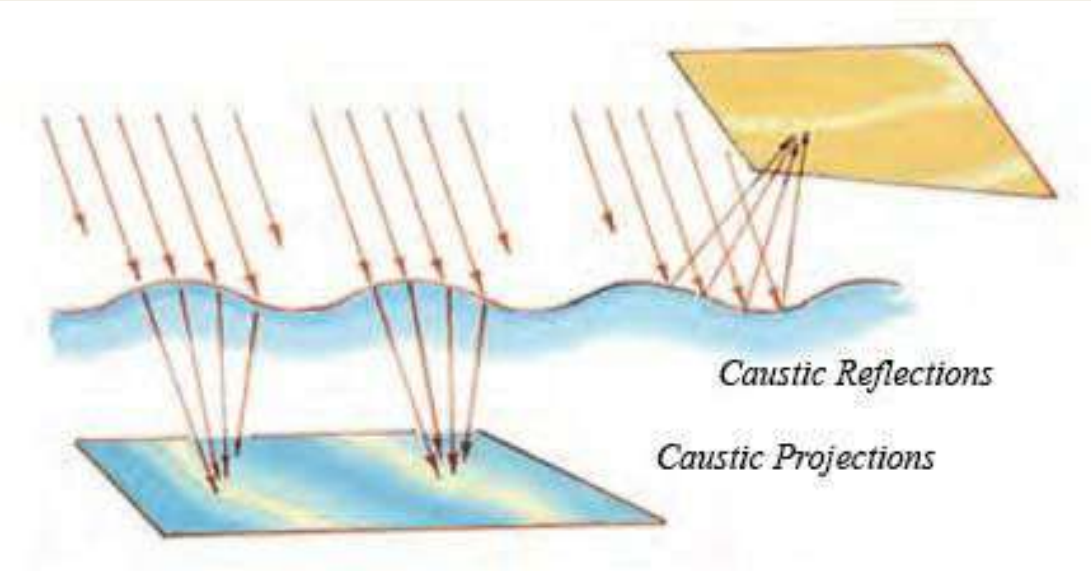


Caustics



Caustics

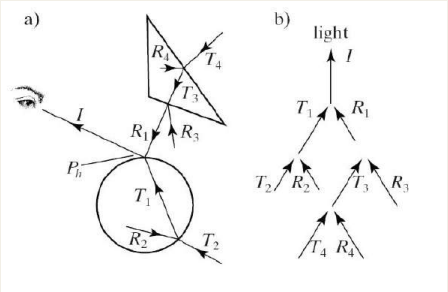
- Require not just “straight line shadow rays” to each light source
- Light refracts before reaching the surface

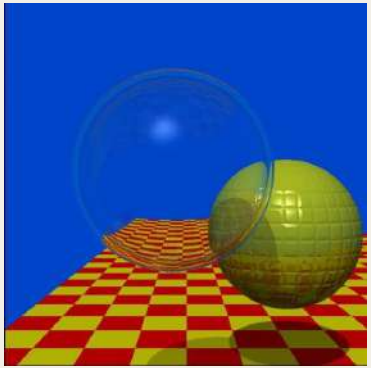


Raycasting -> Whitted Raytracer

- So far: Moved from a model that
  "casts a single camera ray and single shadow ray per pixel" (Appel)
  to "cast single camera ray -> up to 2 more branching rays" (Whitted)
- Better, but still has limitations (ex. Caustics)

- What about multiple camera rays? Even more branching?
- How does this tie back to the rendering equation?





Lecture Outline

- Recursive Raytracing (Whitted raytracing)
  - Reflection
  - Refraction (Snell's Law)
  - Fresnel Equations
  - Attenuation(Beer's Law)

- Evaluating the Rendering Equation:
  - Beyond Blinn-Phong and point lights
  - Monte Carlo Methods
  - Stochastic raytracing (Area lights, depth of field, anti-aliasing)
  - Pathtracing (importance sampling, denoising)

Questions?