Raytracing (Part 2)

Slides by Sarah Jobalia and Kevin Li, with modifications by Kate Baker

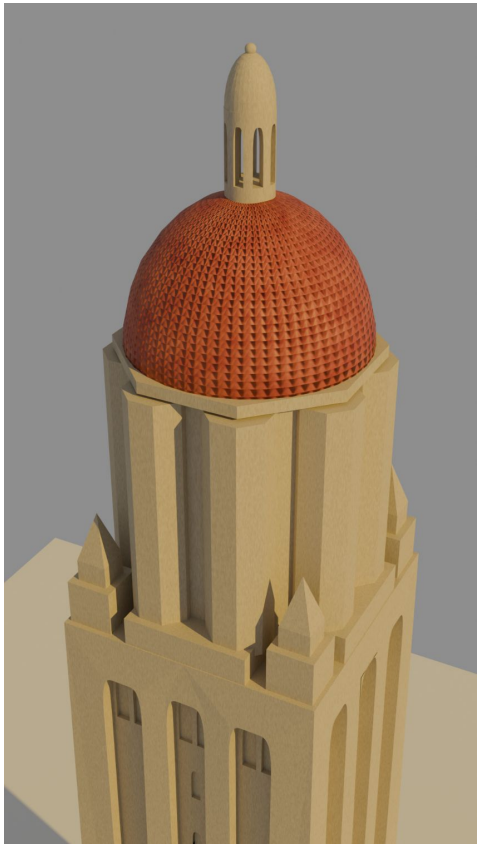
Housekeeping

- HW3 released, due Thursday
- Fill out the final presentation absence form!
- Thank you all for submitting HRCF feedback!
- It’s okay (and expected!) to be a bit confused about the derivations during lecture
- Guest speakers… stay tuned :)

Custom Texture Student Highlights!

Rio Suzuki

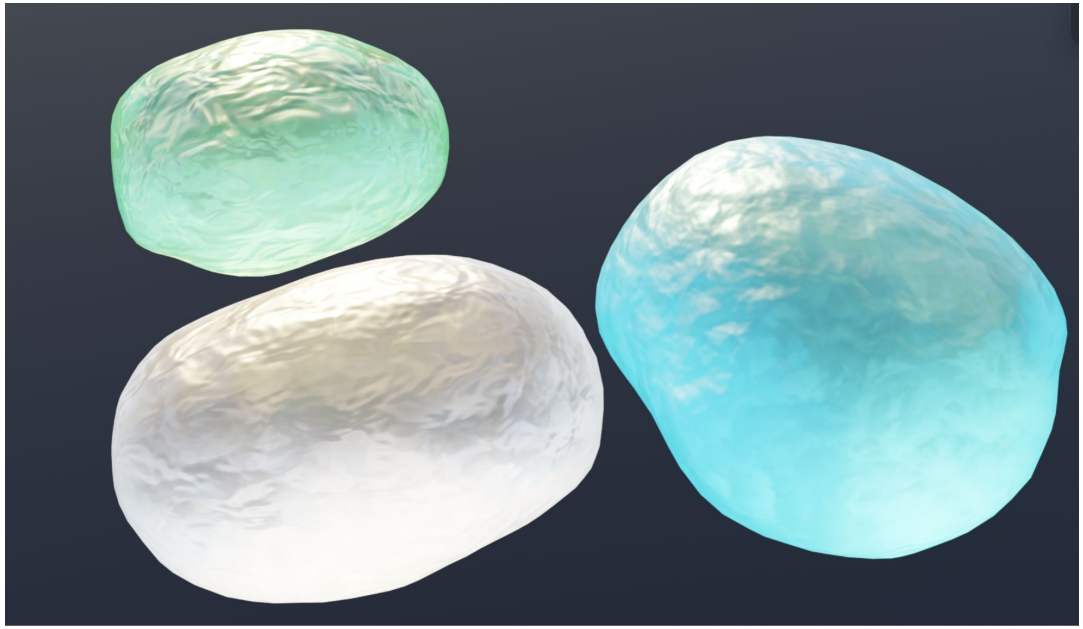
Ji Won Ryoo



Andrea Di Matteo

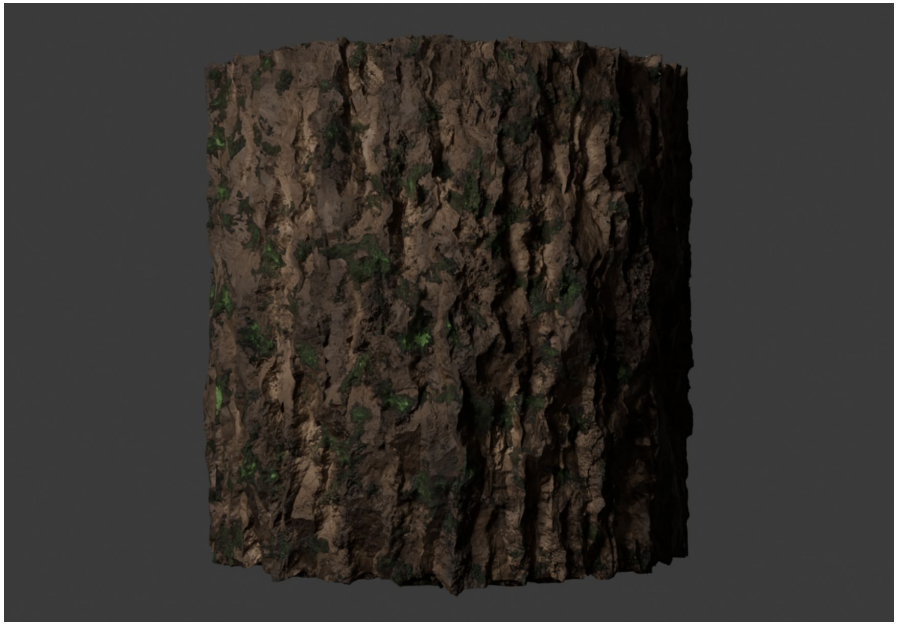
Custom Texture Student Highlights!

Silvia Festa



Custom Texture Student Highlights!

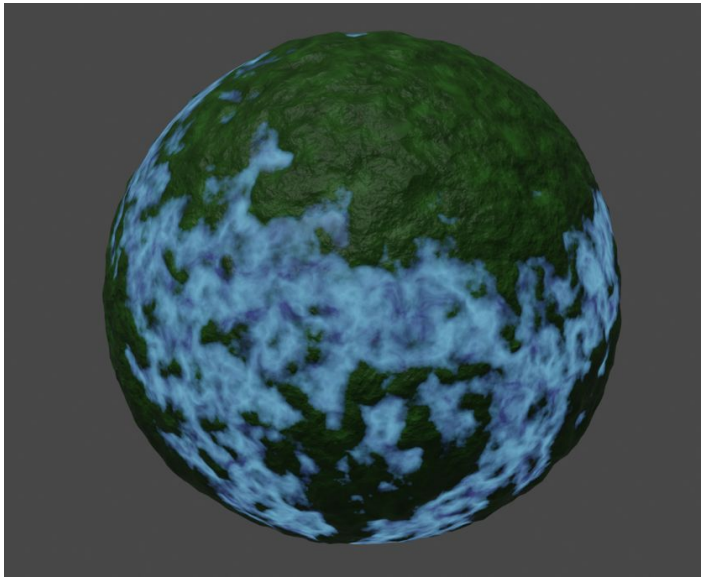
Siole Mayeski



Ya Wen

Custom Texture Student Highlights!

Jiuxing Chen



Daniel Li

Revisiting Questions

- In direct 3D ray-object intersection, how do we know if we've even hit the triangle that we care about?

- What if we can't solve this^ matrix?

Revisiting Questions

- In direct 3D ray-object intersection, how do we know if we've even hit the triangle that we care about?

- What if we can't solve this^ matrix?

Recall: Ray-Triangle Intersection

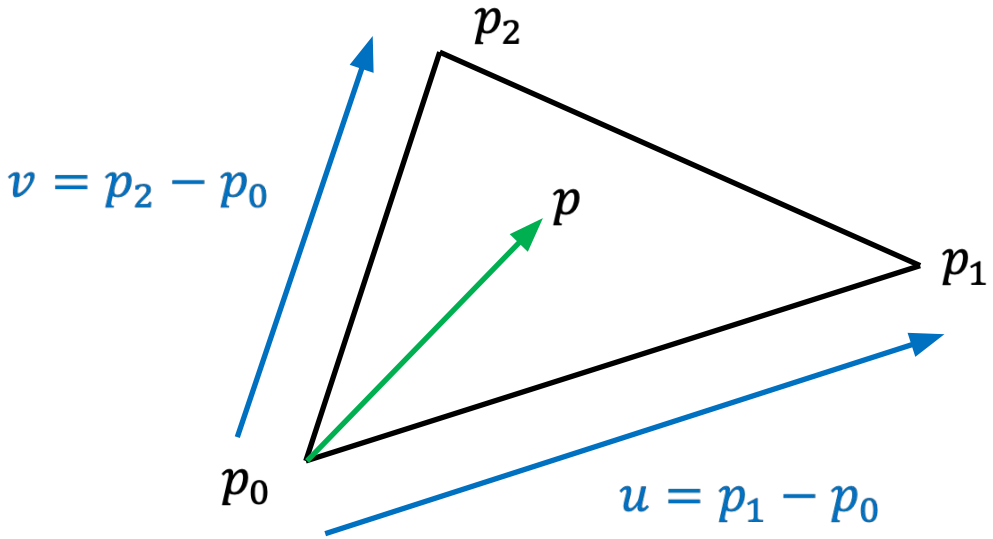
- Recall: most of our object will be **triangle meshes**
  - We need to intersect our ray with a triangle
  - Triangles are **planar** (3 points are guaranteed to make a plane)

- One technique:
  - ~~1) Determine how a ray intersects a **plane** for an intersection point~~ (Done!)
  - ~~2) Then, check if this intersection point is **inside the triangle**~~
- Another approach:
  - Consider 3D ray-object intersection directly

Recall: Triangle Basis Vectors

- Any point inside a triangle can be written as a sum of **one of the vertices + scalings of the edge vectors**

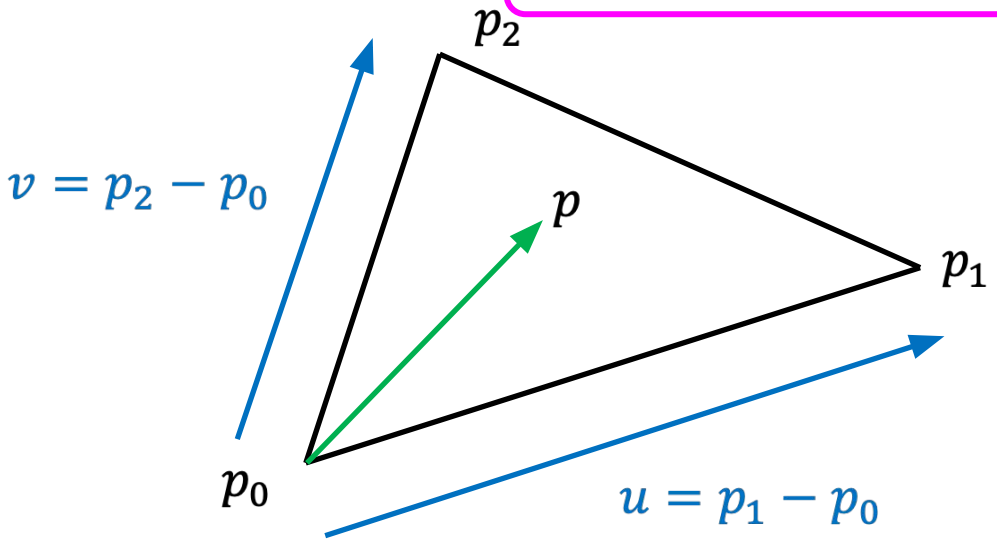
$$p = p_o + \beta_1 u + \beta_2 v \quad \text{with} \quad \beta_1, \beta_2 \in [0, 1], \beta_1 + \beta_2 \leq 1$$



Recall: Triangle Basis Vectors

- Any point inside a triangle can be written as a sum of **one of the vertices + scalings of the edge vectors**

$$p = p_o + \beta_1 u + \beta_2 v \quad \text{with} \quad \beta_1, \beta_2 \in [0, 1], \beta_1 + \beta_2 \leq 1$$



Revisiting Questions

- In direct 3D ray-object intersection, how do we know if we've even hit the triangle that we care about?

- What if we can't solve this^ matrix?

Recall: Direct Ray-Triangle Intersection

- A point inside a triangle is given by: $p = p_o + \beta_1 u + \beta_2 v$
- Substitute our ray: $R(t) = A + (P - A)t$

$$A + (P - A)t = p_o + \beta_1 u + \beta_2 v$$

$$(u, v, A - P) \begin{pmatrix} \beta_1 \\ \beta_2 \\ t \end{pmatrix} = A - p_o$$

- Solve matrix equation for: $\beta_1, \beta_2 \in [0, 1], \beta_1 + \beta_2 \leq 1$
- And: $t \in [1, t_{far}]$

Recall: Direct Ray-Triangle Intersection

- A point inside a triangle is given by: $p = p_o + \beta_1 u + \beta_2 v$
- Substitute our ray: $R(t) = A + (P - A)t$

$$A + (P - A)t = p_o + \beta_1 u + \beta_2 v$$

$$(u, v, A - P) \begin{pmatrix} \beta_1 \\ \beta_2 \\ t \end{pmatrix} = A - p_o$$

No solution: ray parallel to triangle's plane

- Solve matrix equation for: $\beta_1, \beta_2 \in [0, 1], \beta_1 + \beta_2 \leq 1$
- And: $t \in [1, t_{far}]$

Recall: Direct Ray-Triangle Intersection

- A point inside a triangle is given by: $p = p_o + \beta_1 u + \beta_2 v$
- Substitute our ray: $R(t) = A + (P - A)t$

$$A + (P - A)t = p_o + \beta_1 u + \beta_2 v$$

$$(u, v, A - P) \begin{pmatrix} \beta_1 \\ \beta_2 \\ t \end{pmatrix} = A - p_o$$

- Solve matrix equation for: $\beta_1, \beta_2 \in [0, 1], \beta_1 + \beta_2 \leq 1$
- And: $t \in [1, t_{far}]$

No solution: ray parallel to triangle's plane

Infinite solutions: ray lies in plane

Recall: Direct Ray-Triangle Intersection

- A point inside a triangle is given by: $p = p_o + \beta_1 u + \beta_2 v$
- Substitute our ray: $R(t) = A + (P - A)t$

$$A + (P - A)t = p_o + \beta_1 u + \beta_2 v$$

$$(u, v, A - P) \begin{pmatrix} \beta_1 \\ \beta_2 \\ t \end{pmatrix} = A - p_o$$

- Solve matrix equation for: $\beta_1, \beta_2 \in [0, 1], \beta_1 + \beta_2 \leq 1$
- And: $t \in [1, t_{far}]$

No solution: ray parallel to triangle's plane

Infinite solutions: ray lies in plane

If determinant of matrix is nonzero: a solution!

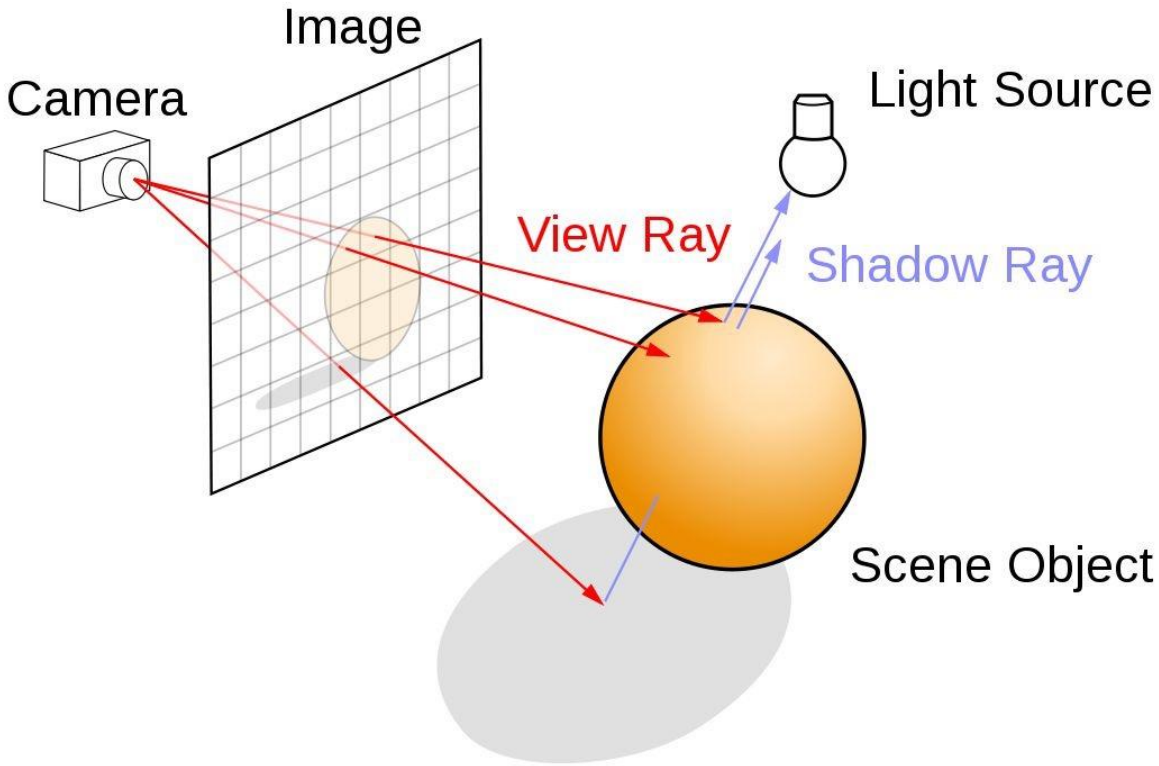
Lecture Outline

- Ray tracing review
- Raycasting & recursive raytracing
- Reflected & transmitted rays
  - Refraction
  - Total internal reflection
- Attenuation & caustics

Lecture Outline

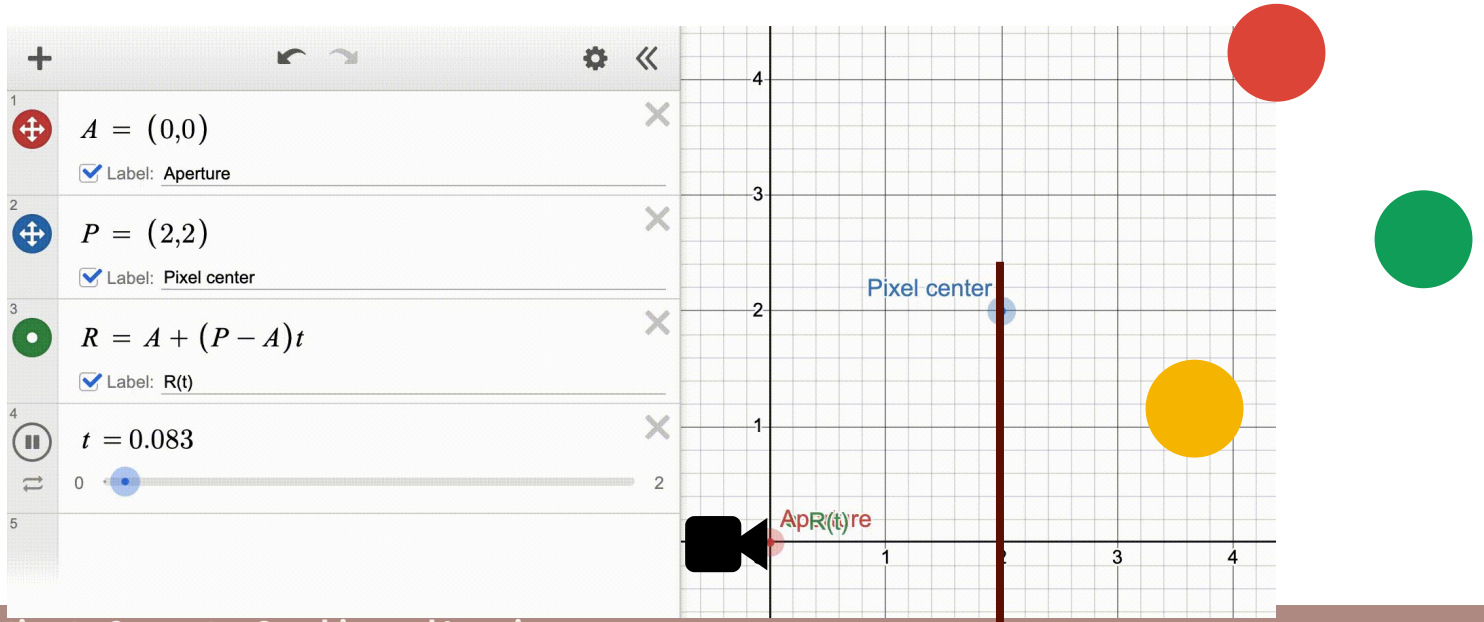
- Ray tracing review
- Raycasting & recursive raytracing
- Reflected & transmitted rays
  - Refraction
  - Total internal reflection
- Attenuation & caustics

Raytracing Review



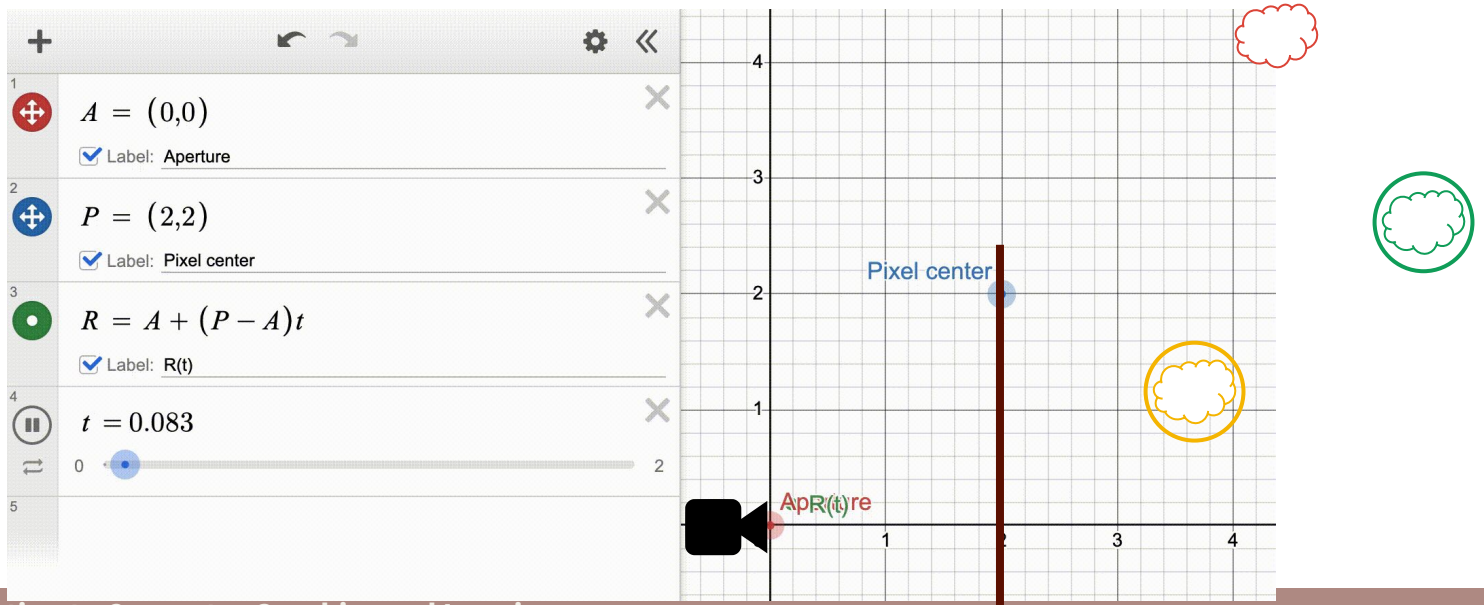
Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the ray with scene geometry
- Once we've found the intersection point, shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF (the lighting equation)



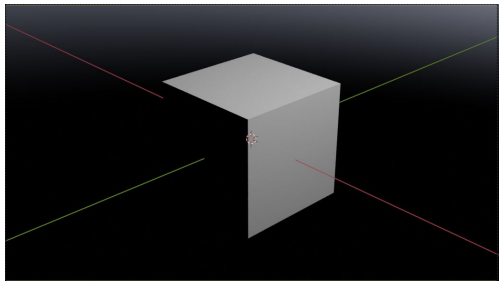
Raytracing Review

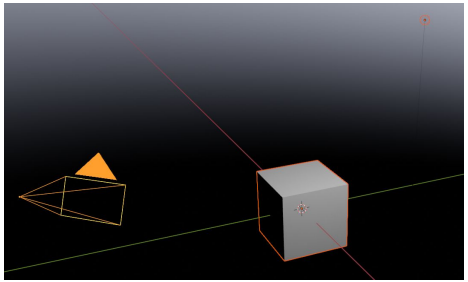
- For each pixel, shoot a camera ray
- Find the closest intersection of the ray with scene geometry
- Once we've found the intersection point, shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF (the lighting equation)

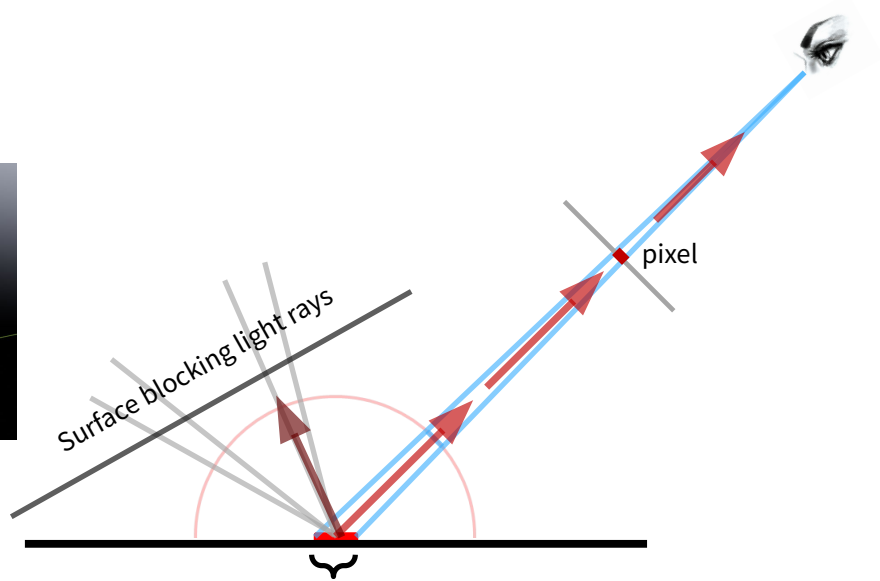


Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the ray with scene geometry
- Once we've found the intersection point, shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF (the lighting equation)

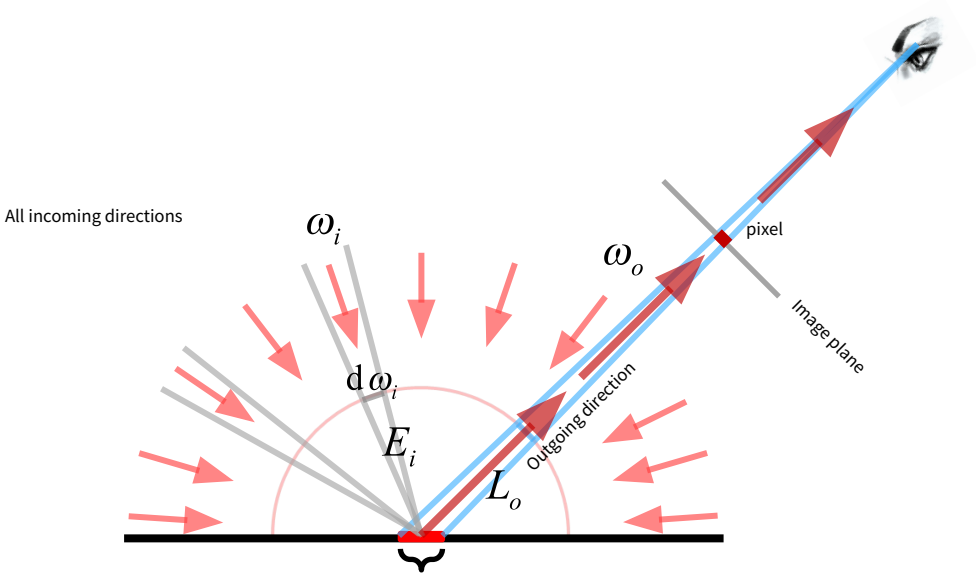






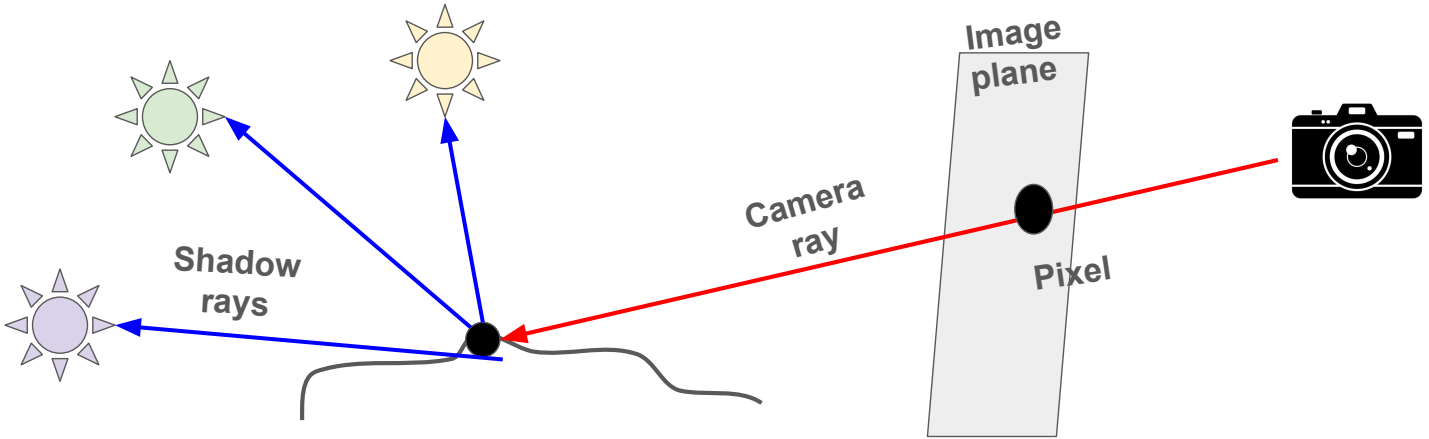
Raytracing Review

- For each pixel, shoot a camera ray
- Find the closest intersection of the ray with scene geometry
- Once we've found the intersection point, shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF (the lighting equation)



Raytracing Review

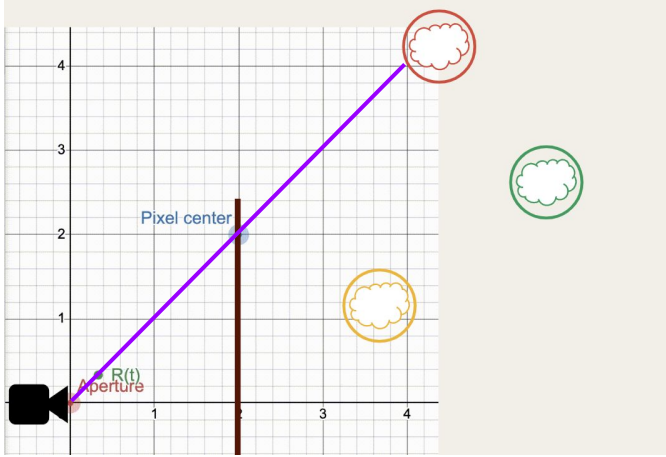
- For each pixel, shoot a camera ray
- Find the closest intersection of the ray with scene geometry
- Once we've found the intersection point, shoot shadow rays to all the light sources
- Compute pixel color by evaluating the BRDF (the lighting equation)



Raycasting

```
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
```



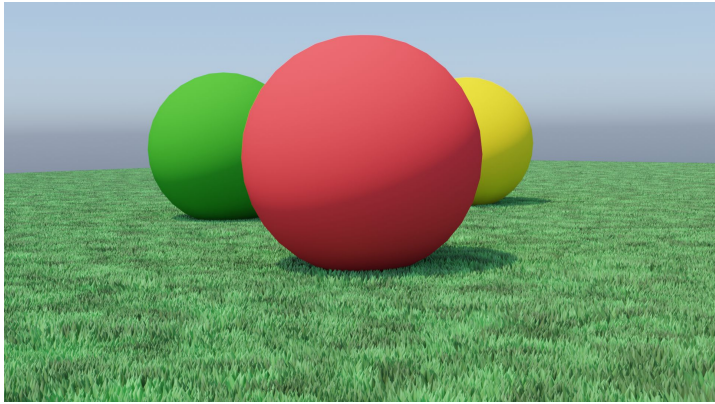

```
function evaluate_rendering_equation(intersection, objects, lights):
    color=intersection.ambient
    for light in lights:
        shadow_ray=generate_ray_to_light(intersection, light)
        if does_not_hit_geometry(shadow_ray, objects):
            color+=BRDF(intersection.camera_ray,shadow_ray)*light.color*cos(theta)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

Lecture Outline

- Ray tracing review
- Raycasting & recursive raytracing
- Reflected & transmitted rays
  - Refraction
  - Total internal reflection
- Attenuation & caustics

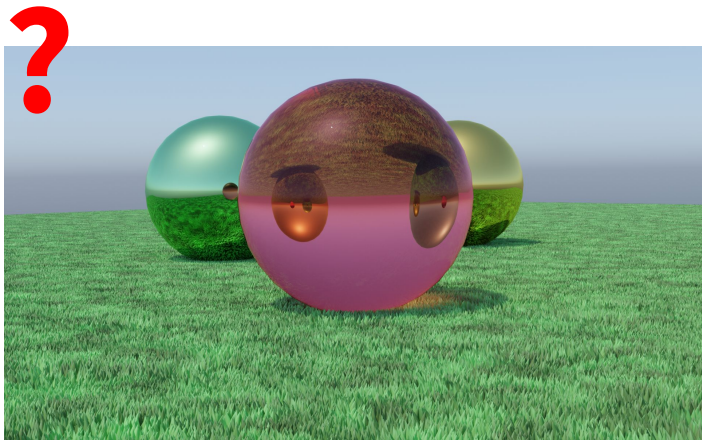
Raycasting

- 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



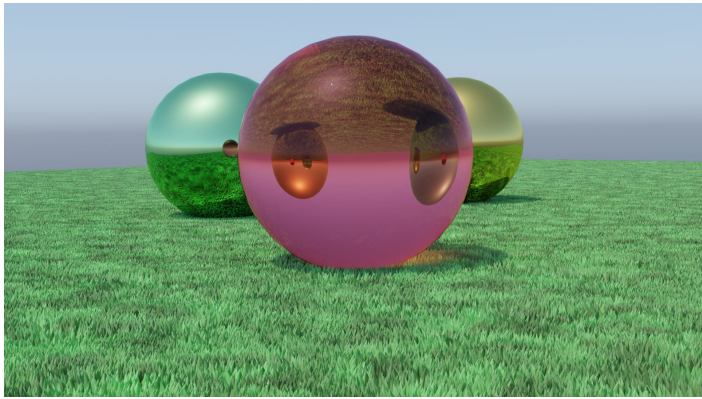
Raycasting

- 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



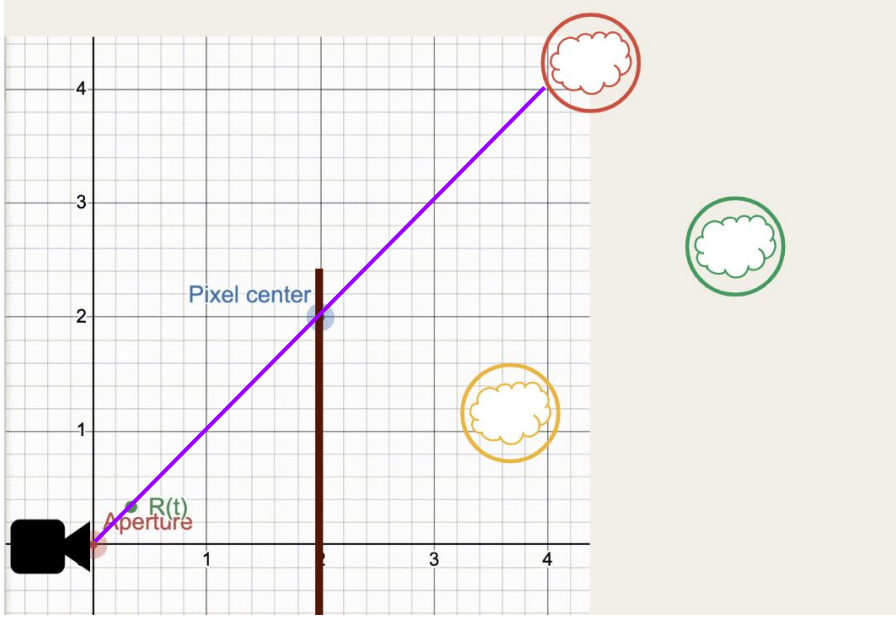
Raycasting

- 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



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)

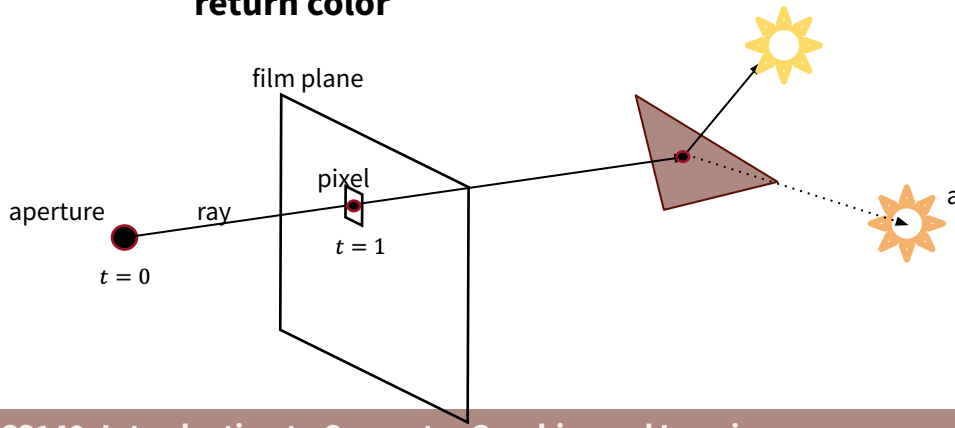
color=color+**trace(reflected_ray,objects,lights)*reflectivity**

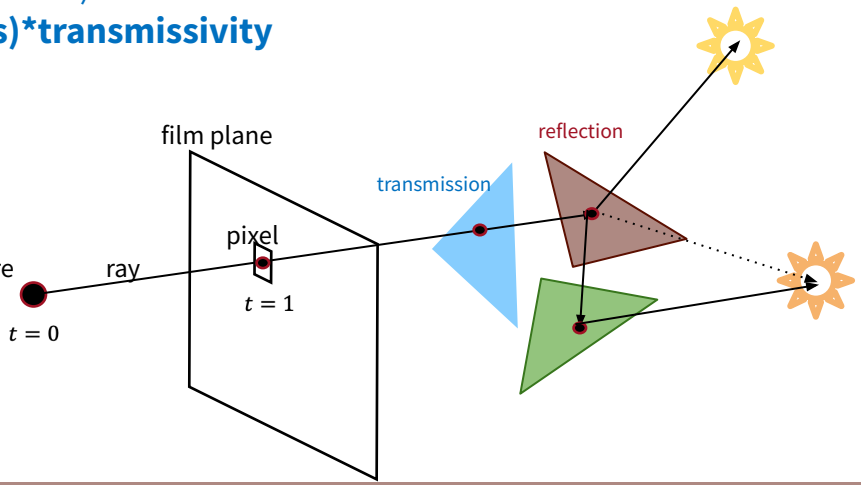
if (material at intersection has transmissivity>0)

refracted_ray=create _refracted_ray(ray, intersection)

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

return color



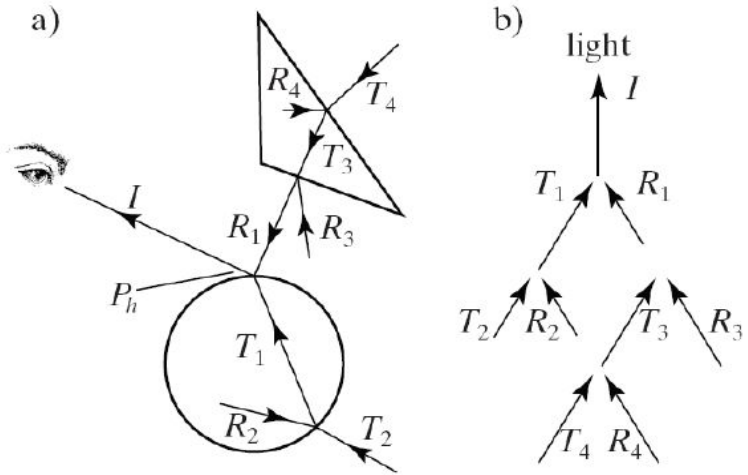


Recursive Raytracing

- How do we avoid infinite recursion (stack overflow)?
  - E.g. what if a transmission/reflection ray hit another transmissive/reflective object? And then another transmissive/reflective object? And then another…

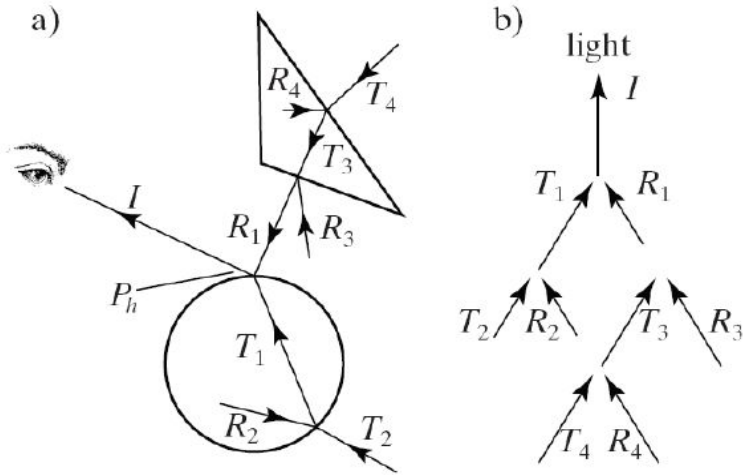
Recursive Raytracing: Termination

- To avoid infinite recursion (stack overflow), we need to keep track of how many bounces it's been
- Terminate raytracing after certain recursion depth



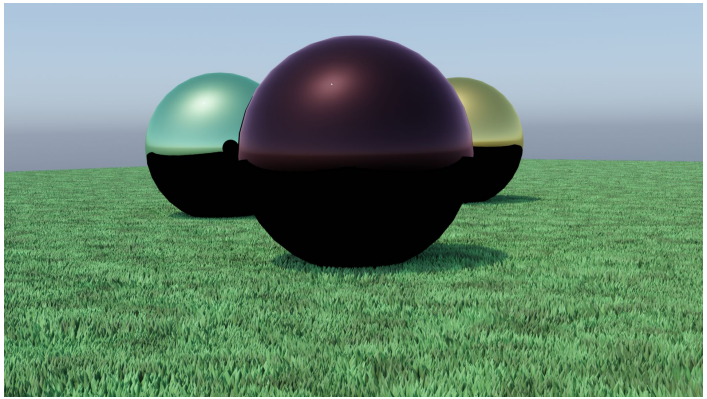
Recursive Raytracing: Termination

- To avoid infinite recursion (stack overflow), we need to keep track of how many bounces it's been
- Terminate raytracing after certain recursion depth
- This can happen with other materials besides mirrors
  - E.g. total internal reflection (later today)

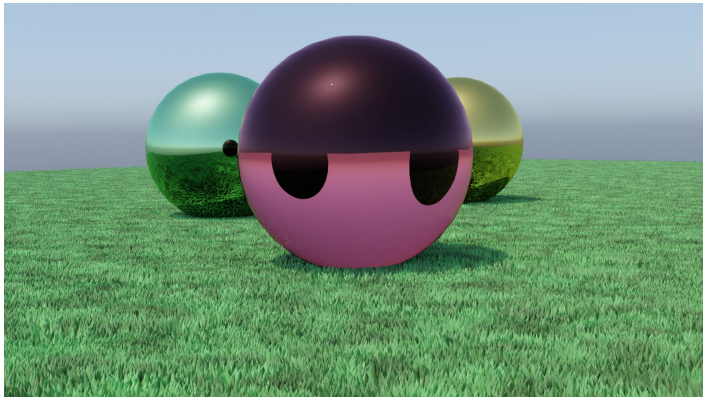


Recursive Raytracing

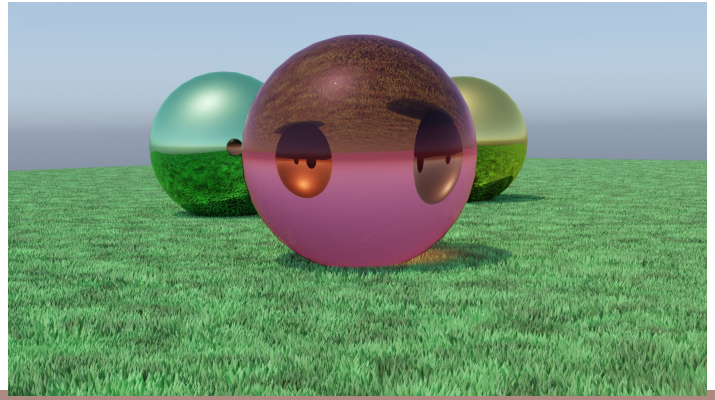
No recursions



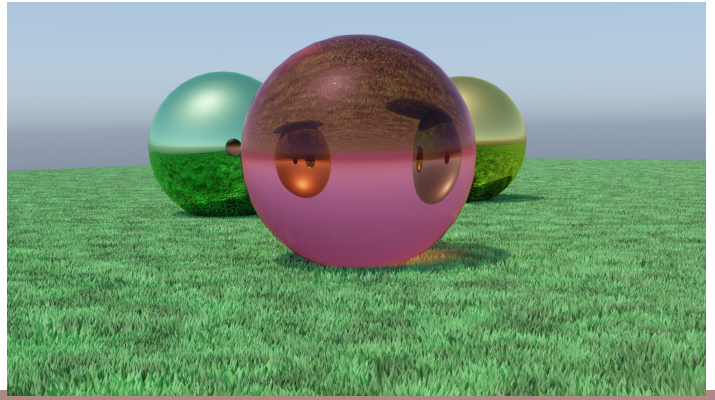
1 recursion bounce



2 recursion bounces

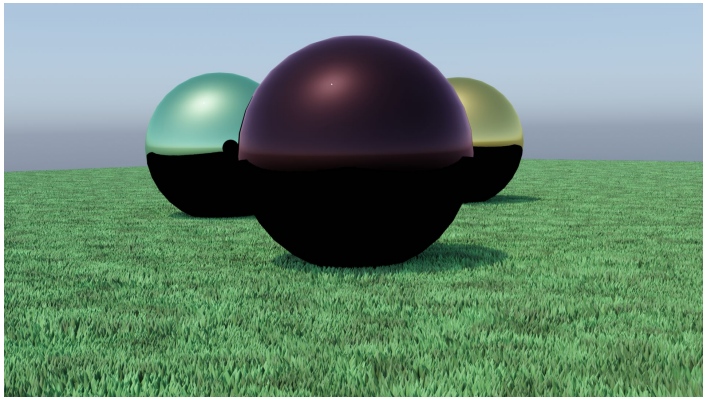


3 recursion bounces

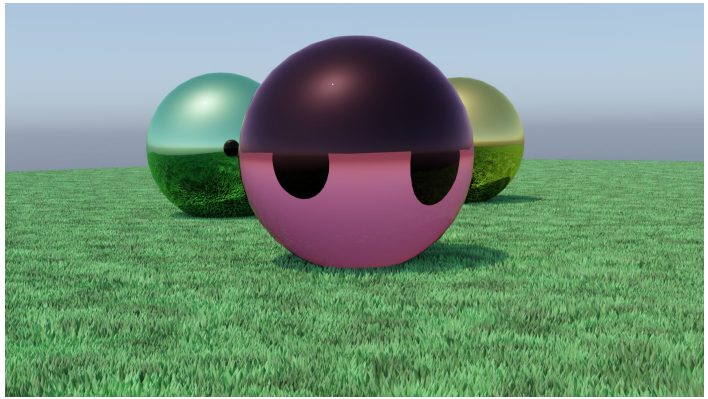


Recursive Raytracing

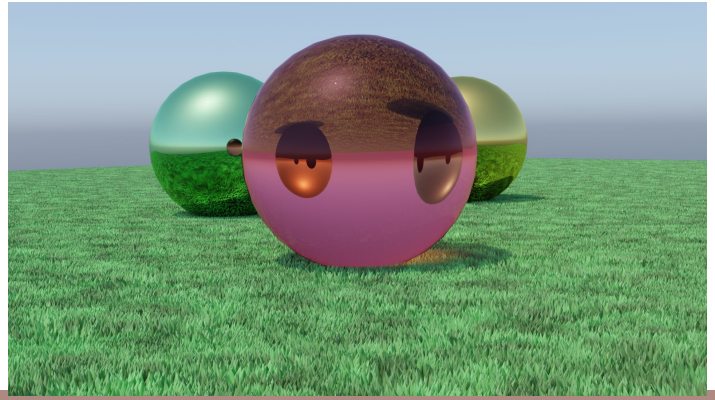
No recursions



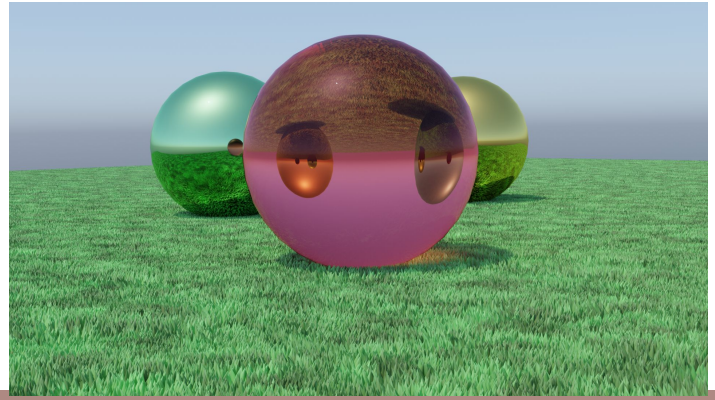
1 recursion bounce



2 recursion bounces

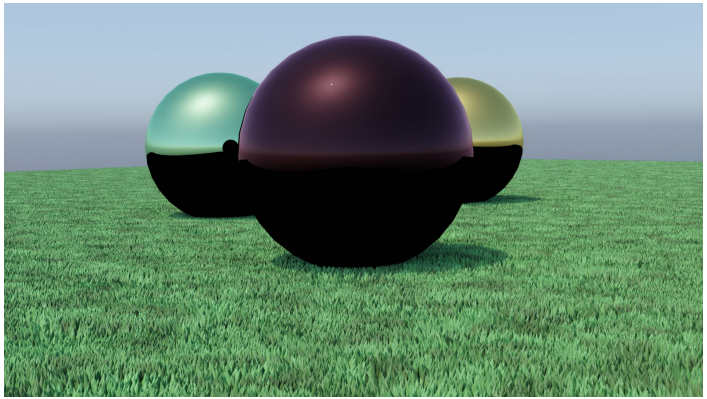


3 recursion bounces

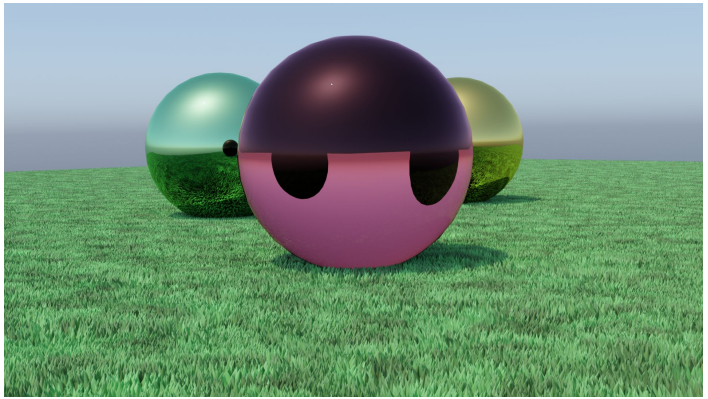


Recursive Raytracing

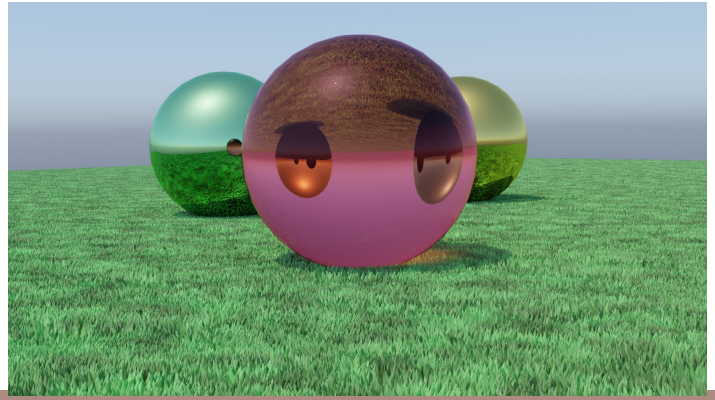
No recursions



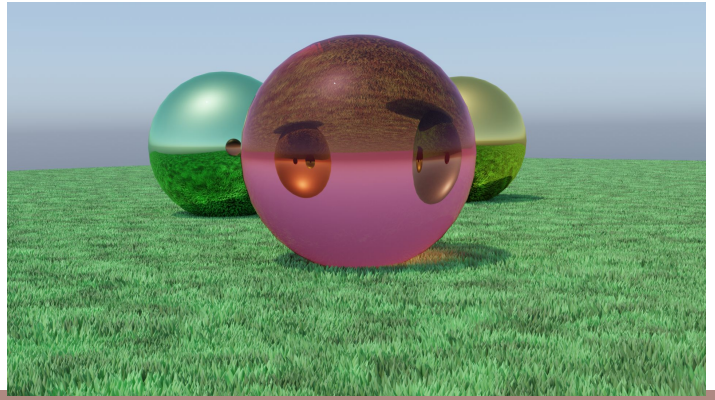
1 recursion bounce



2 recursion bounces

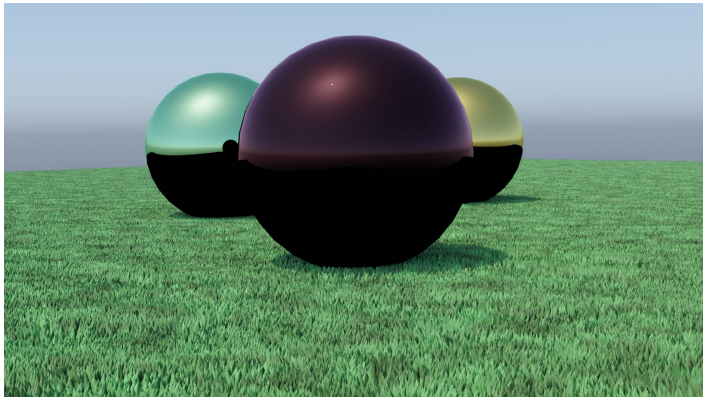


3 recursion bounces

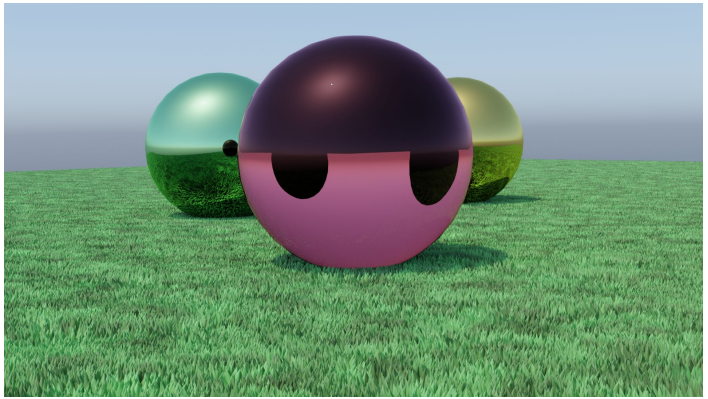


Recursive Raytracing

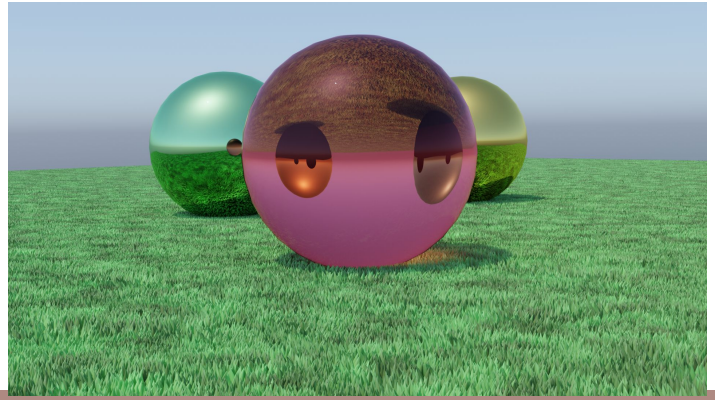
No recursions



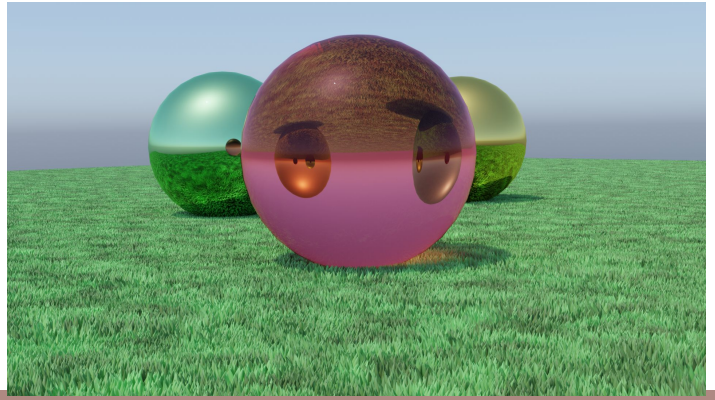
1 recursion bounce



2 recursion bounces

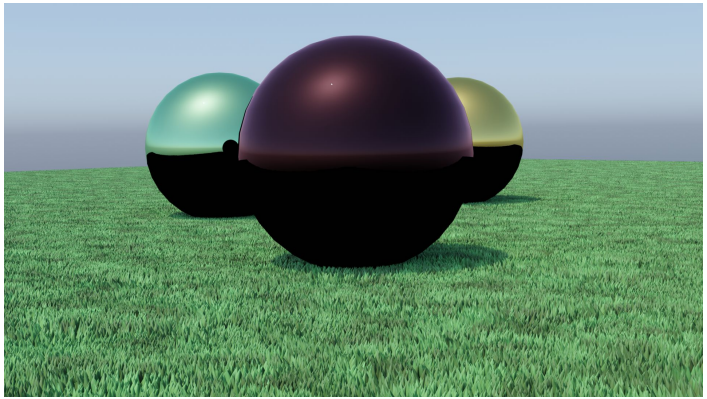


3 recursion bounces

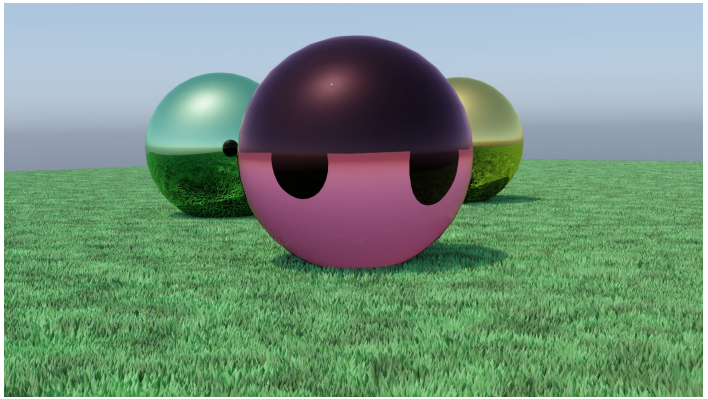


Recursive Raytracing

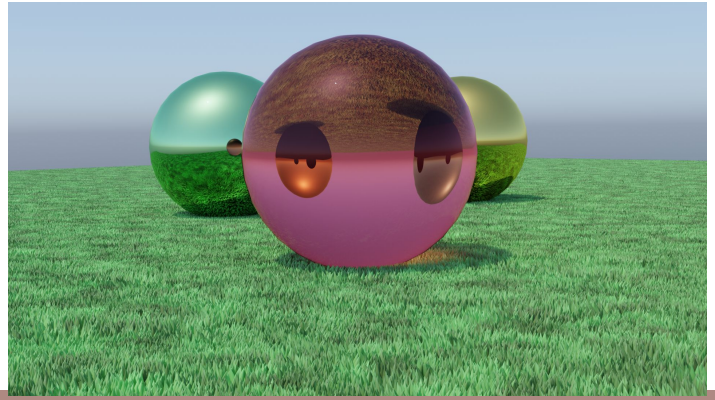
No recursions



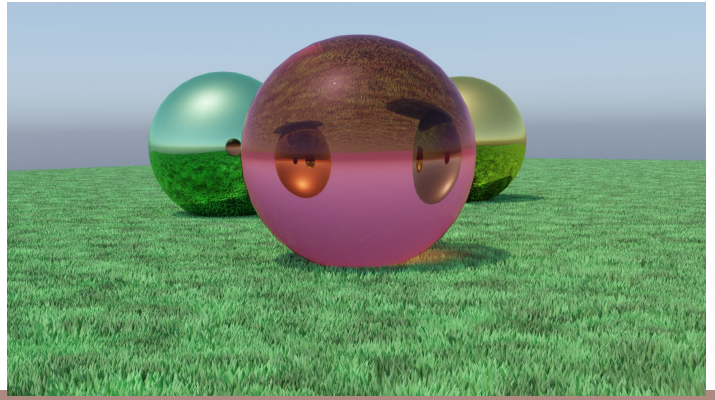
1 recursion bounce



2 recursion bounces

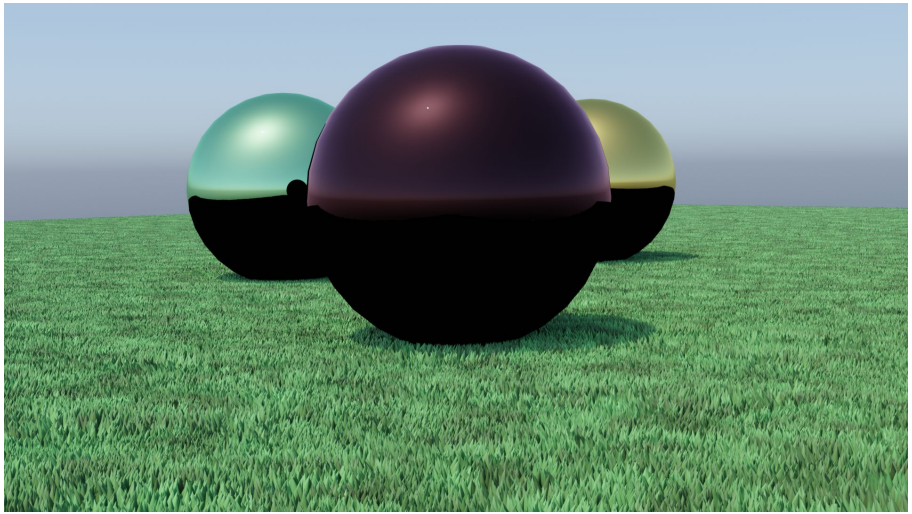


3 recursion bounces

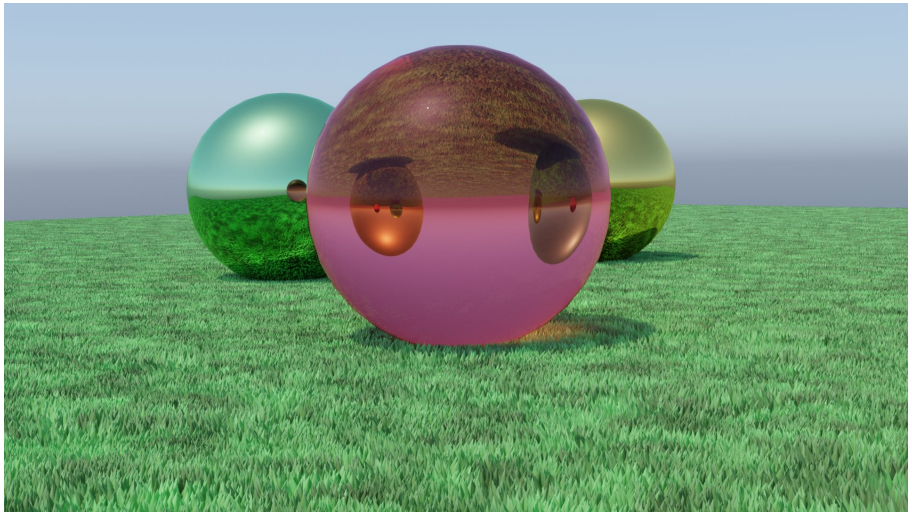


Recursive Raytracing: Termination

Max bounces = 0



Max bounces = 16

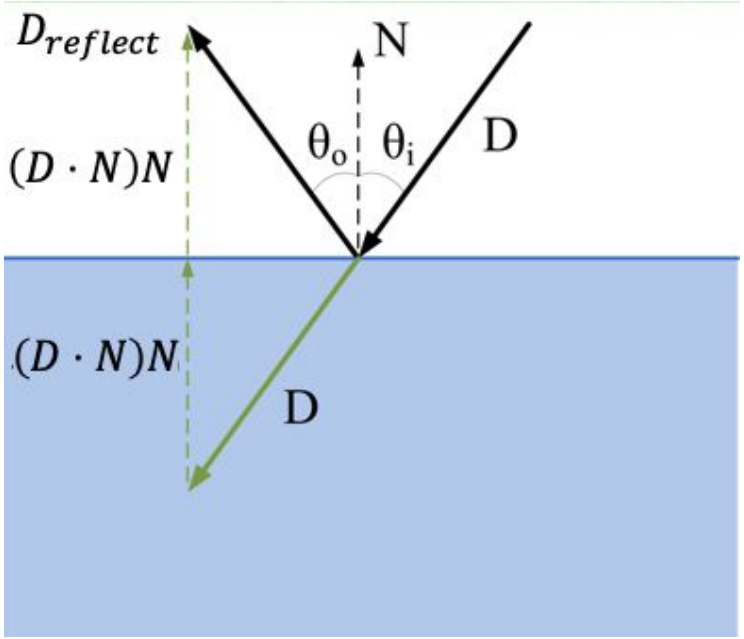


Lecture Outline

- Ray tracing review
- Raycasting & recursive raytracing
- Reflected & transmitted rays
  - Refraction
  - Total internal reflection
- Attenuation & caustics

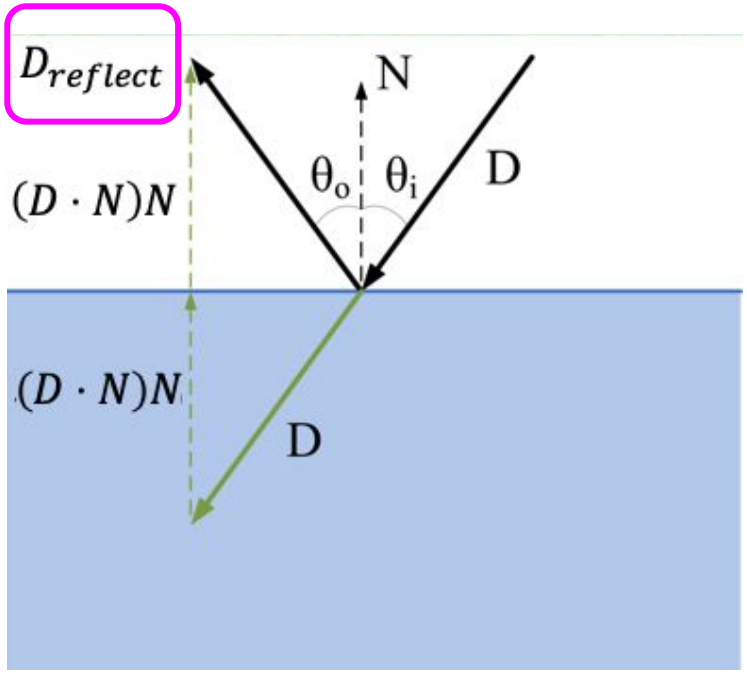
Reflected Rays

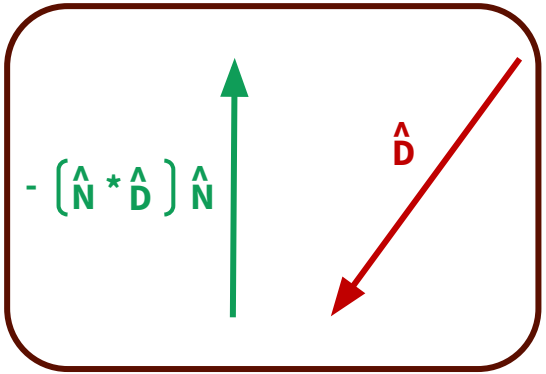
- Reflected rays bounce off of an object at the same angle the ray arrived



Reflected Rays

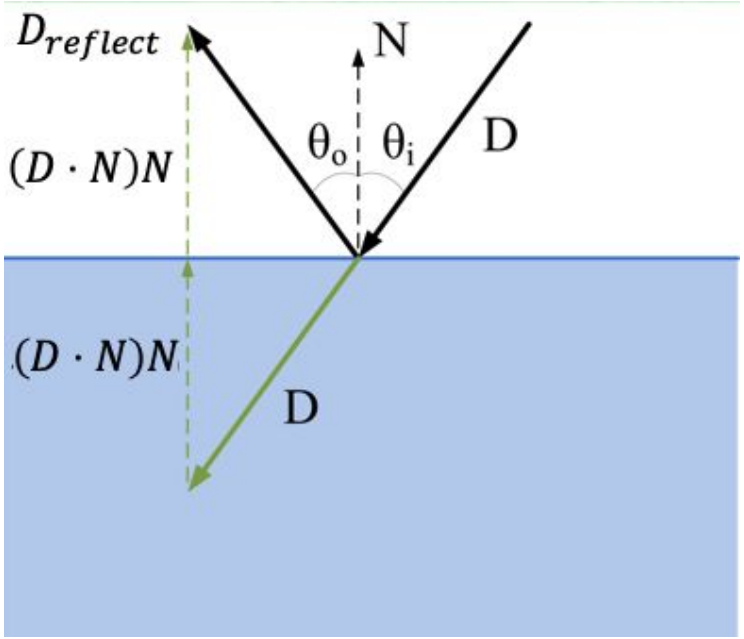
- Reflected rays bounce off of an object at the same angle the ray arrived



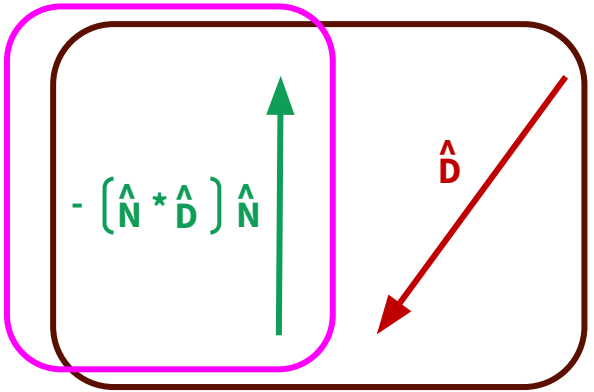


Reflected Rays

- Reflected rays bounce off of an object at the same angle the ray arrived



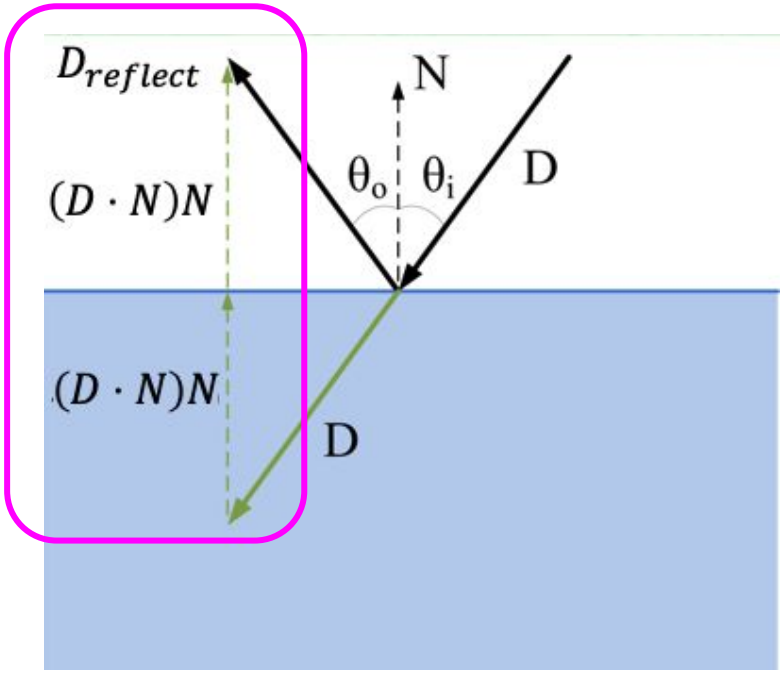
Decompose *D* into parallel and perpendicular (to *N*) components



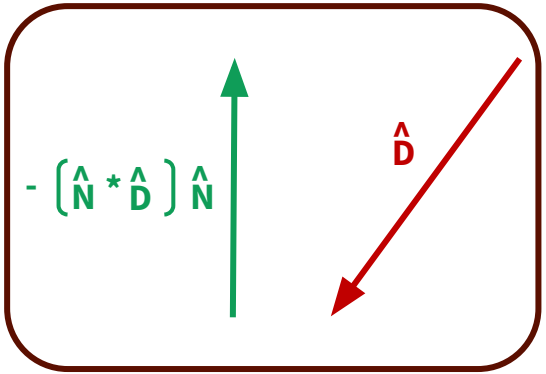
$$\hat{D}_{reflect} = \left(\hat{N} \cdot \hat{D}\right)\hat{N}$$

Reflected Rays

- Reflected rays bounce off of an object at the same angle the ray arrived



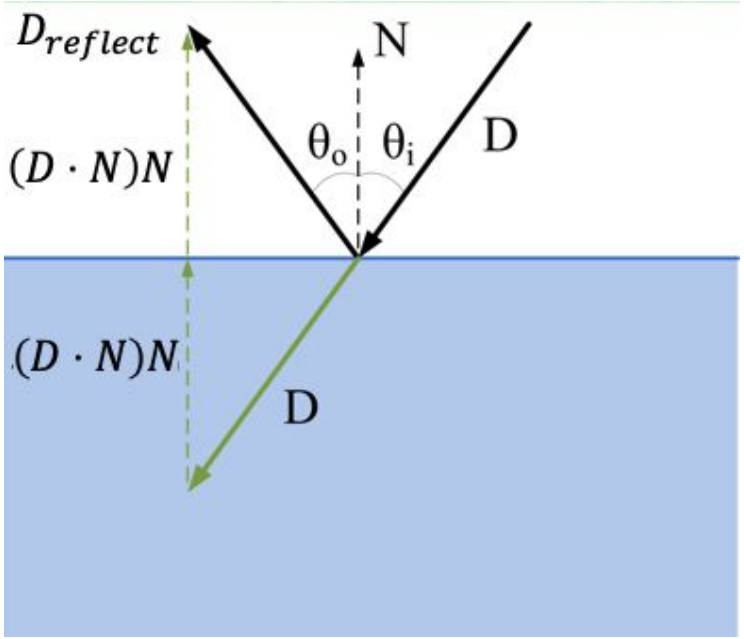
Component of *D* perpendicular to *N* stays the same, parallel component gets flipped



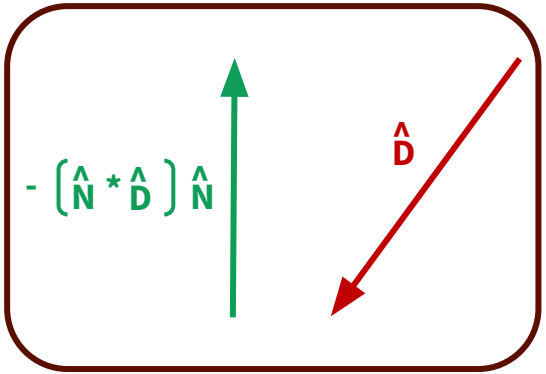
$$\hat{D}_{reflect} = \hat{D} - \left(\hat{N} \cdot \hat{D}\right)\hat{N}$$

Reflected Rays

- Reflected rays bounce off of an object at the same angle the ray arrived



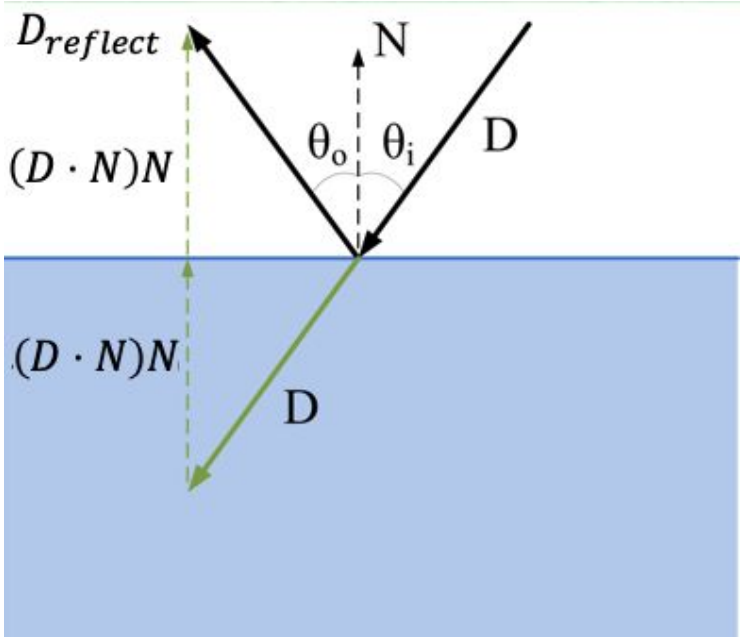
Subtract the parallel to *N* part of *D* twice; first to cancel the inward component, second to build the outward one

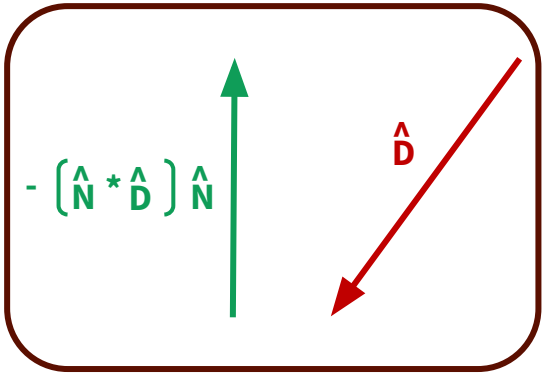


$$\hat{D}_{reflect} = \hat{D} - 2\left(\hat{N} \cdot \hat{D}\right)\hat{N}$$

Reflected Rays

- Reflected rays bounce off of an object at the same angle the ray arrived

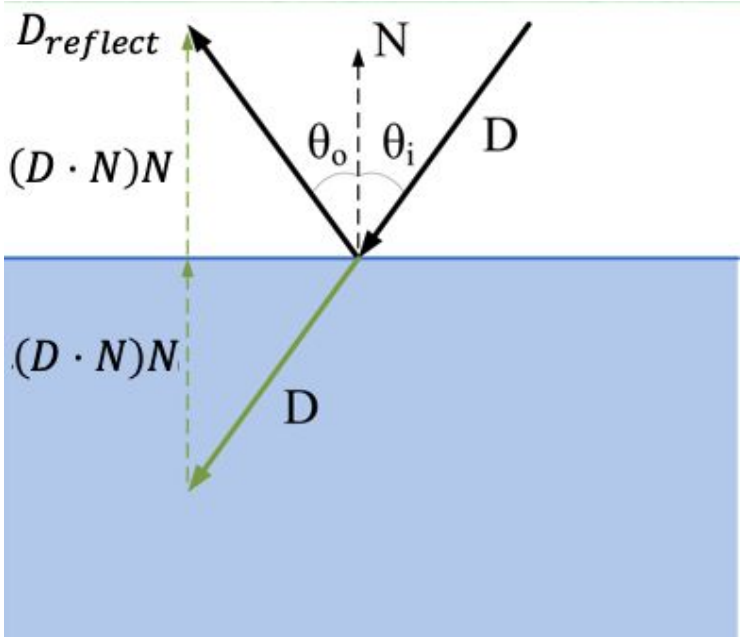


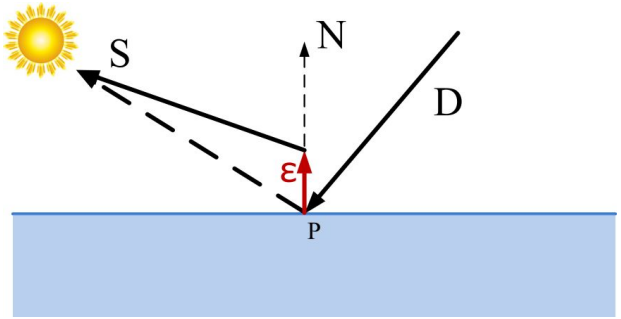


$$\hat{D}_{reflect} = \hat{D} - 2\left(\hat{N} \cdot \hat{D}\right)\hat{N}$$

Reflected Rays

- Recall: spurious self-occlusion!



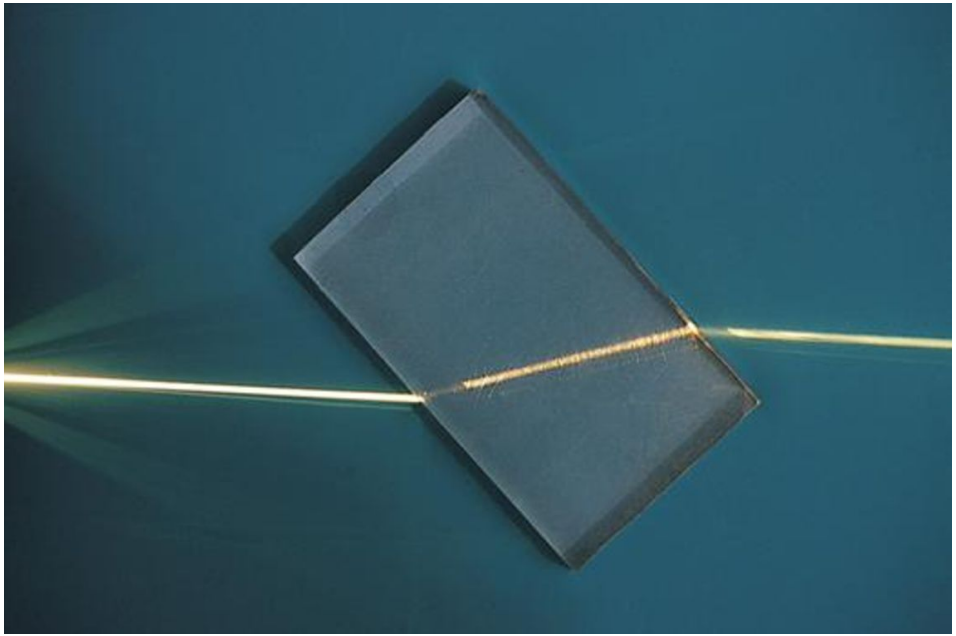


$$\hat{D}_{reflect} = \hat{D} - 2\left(\hat{N} \cdot \hat{D}\right)\hat{N}$$

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

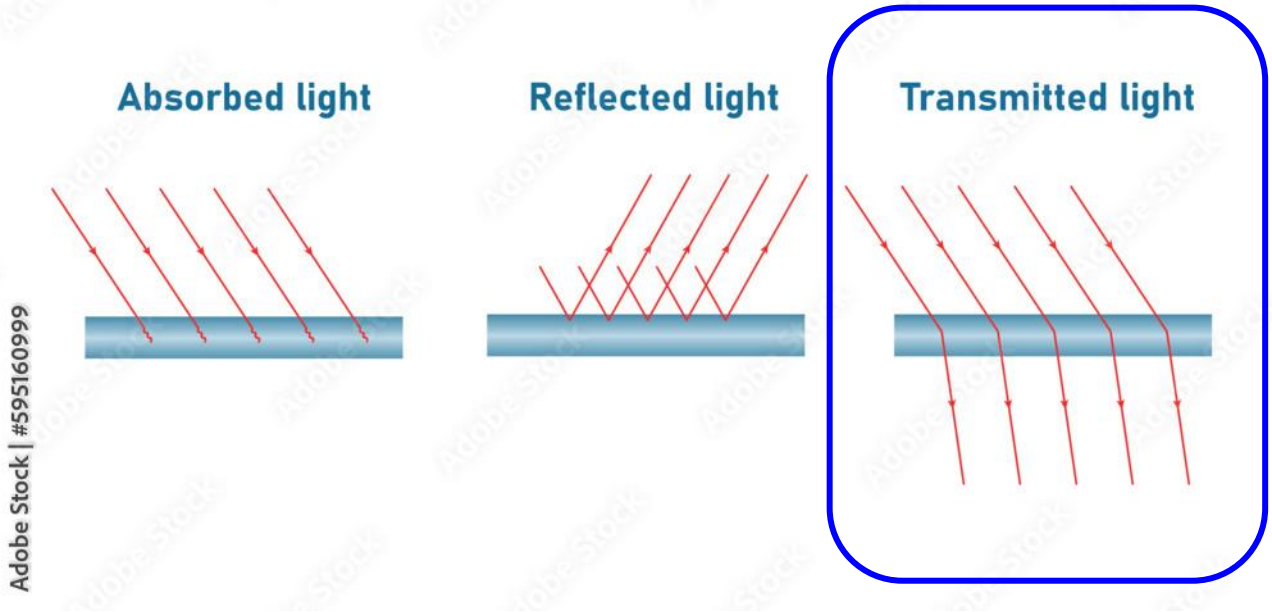
Transmitted Rays

- Light bends as it passed through **different materials**



Transmitted Rays

- Light bends as it passed through **different materials**



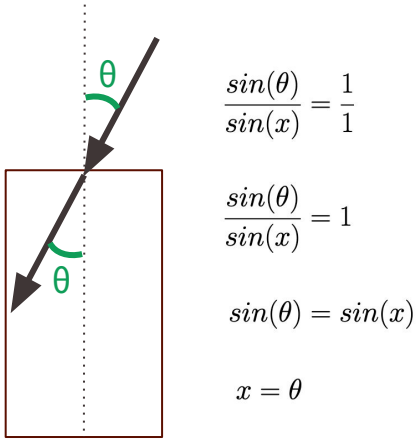
Transmitted Rays

- Light bends as it passed through different materials
- **Snell's law** ⇒ gives the relationship between angle of incidence and transmittance
- $\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$ where n1 the index of refraction for the first medium

Transmitted Rays

- Light bends as it passed through different materials
- **Snell's law** ⇒ gives the relationship between angle of incidence and transmittance
- $\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$ where n1 the index of refraction for the first medium

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

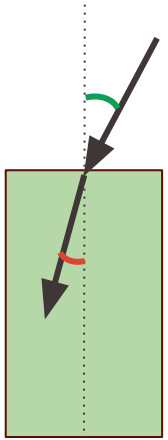


$$\frac{sin(\theta)}{sin(x)} = \frac{1}{1}$$

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

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

$$x = \theta$$



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

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

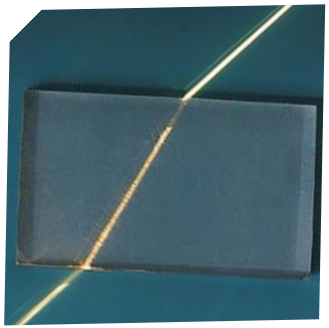
$$sin(x) = \frac{sin(\theta)}{1.5}$$

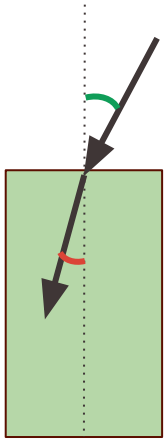
$$x = sin^{-1}(\frac{sin(\theta)}{1.5})$$

Transmitted Rays

- Light bends as it passed through different materials
- **Snell's law** ⇒ gives the relationship between angle of incidence and transmittance
- $\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$ where n1 the index of refraction for the first medium

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





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

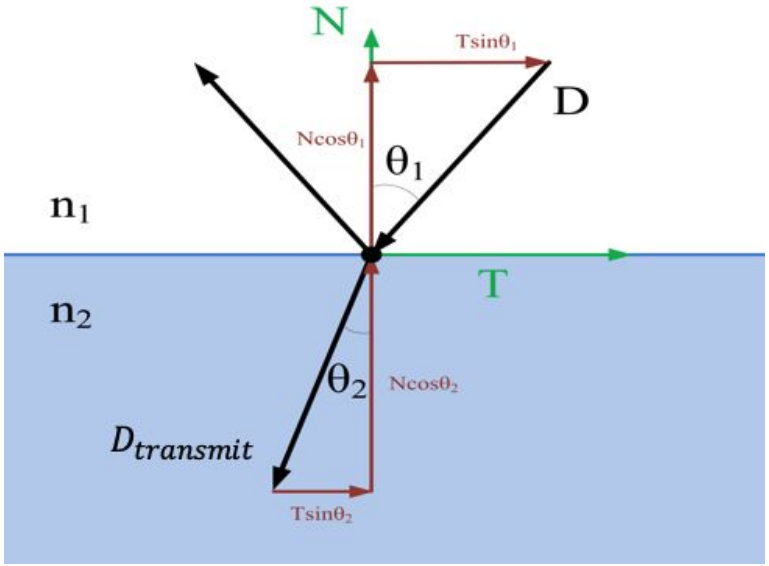
$$\frac{sin(\theta)}{sin(x)} = 1.5$$

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

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

Transmitted Rays

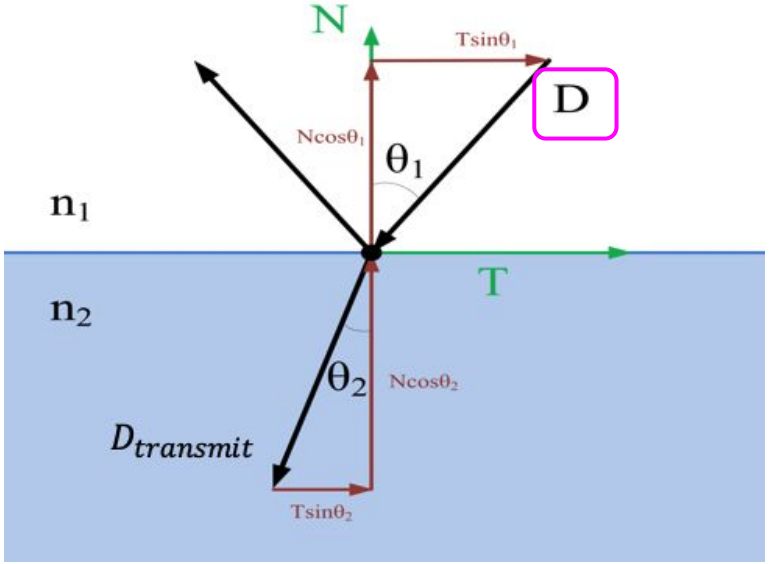
- We want to represent a transmitted ray in terms of n1, n2, D, N



Transmitted Rays

- We want to represent a transmitted ray in terms of n1, n2, D, N

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

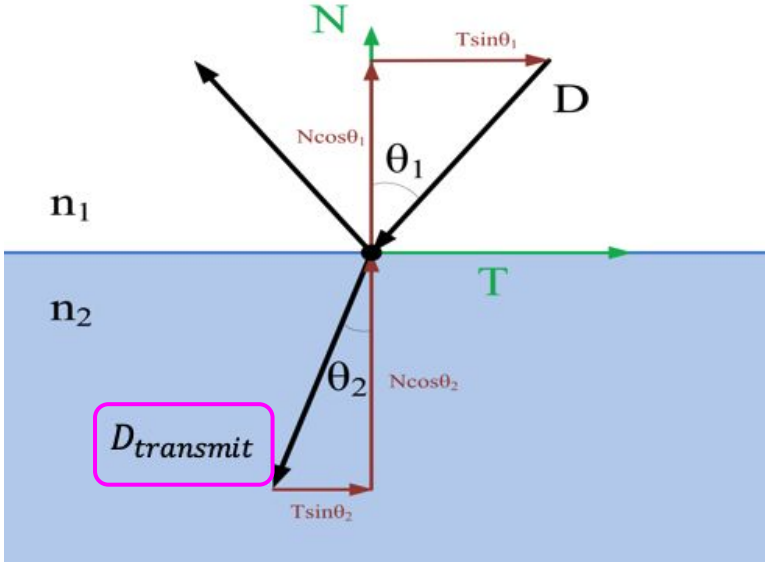


Transmitted Rays

- We want to represent a transmitted ray in terms of n1, n2, D, N

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

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



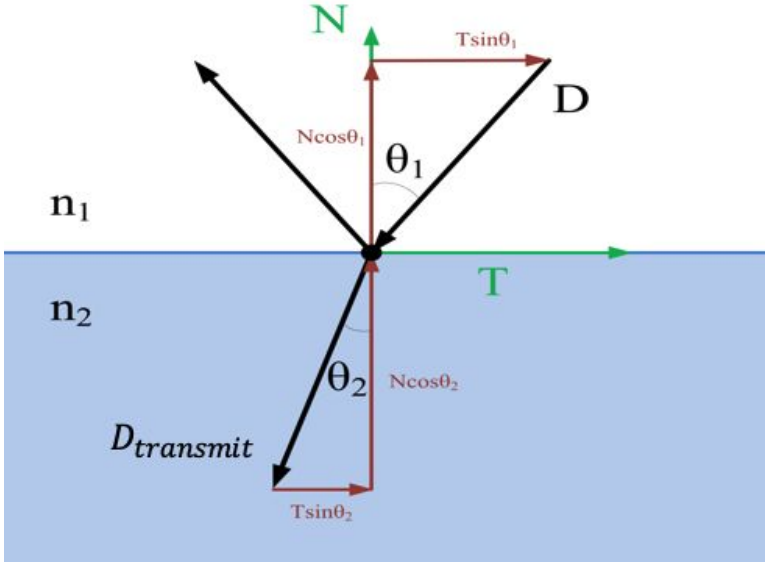
Transmitted Rays

- We want to represent a transmitted ray in terms of n1, n2, D, N

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

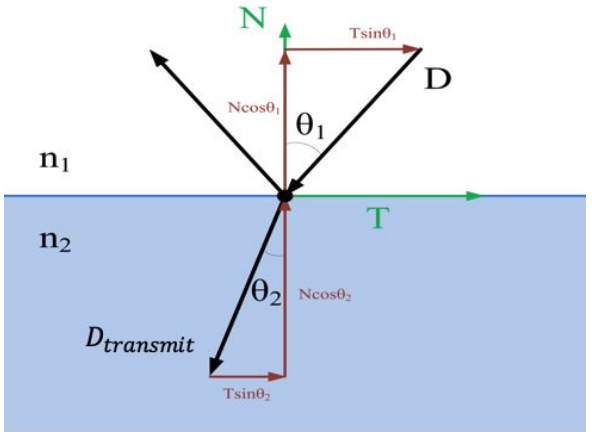
$$\hat{D}_{transmit} = -\cos\theta_2 \hat{N} - \sin\theta_2 \hat{T}$$

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



Transmitted Rays

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



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

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

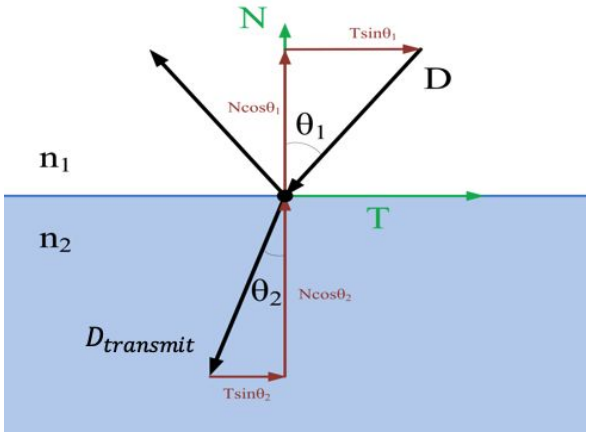
$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$$

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

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

Transmitted Rays

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



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

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

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

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

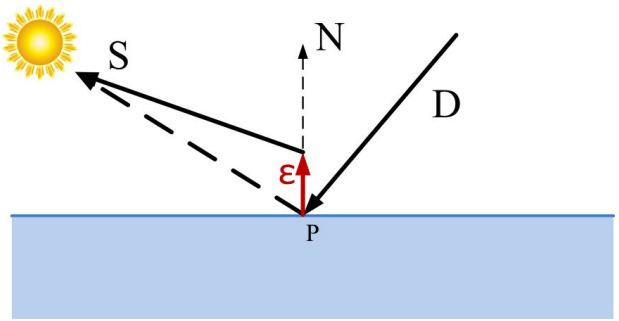
$$\hat{D}_{transmit} = \frac{n_1}{n_2}\hat{D} - \left(\frac{n_1}{n_2}\hat{N}\cdot\hat{D} + \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2\left(1 - \left(\hat{N}\cdot\hat{D}\right)^2\right)}\right)\hat{N}$$

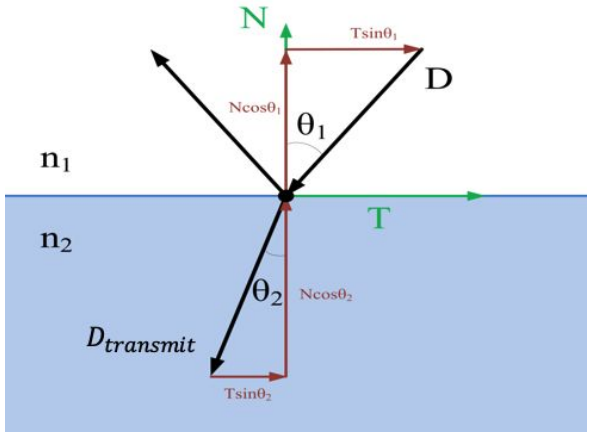
Transmitted Rays

- Don’t forget spurious self-intersection (flip the normals)

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

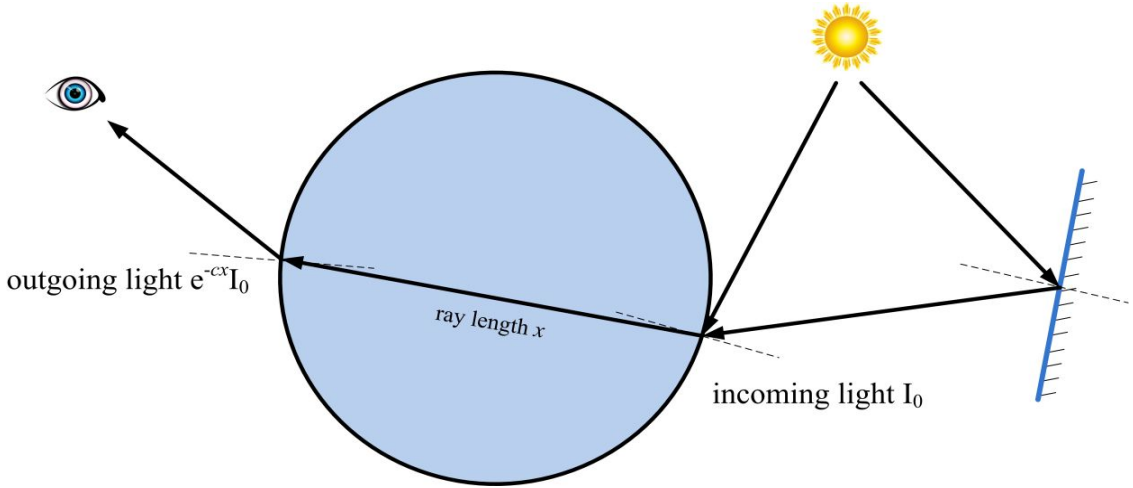
$$R_{transmit}(t) = \left(P - \epsilon\hat{N}\right) + \hat{D}_{transmit}\,t$$





Refraction

- Transmission: compute snell's law twice (once for entering, once for exiting)
- When computing intersection from inside an object, **flip the normals**



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)

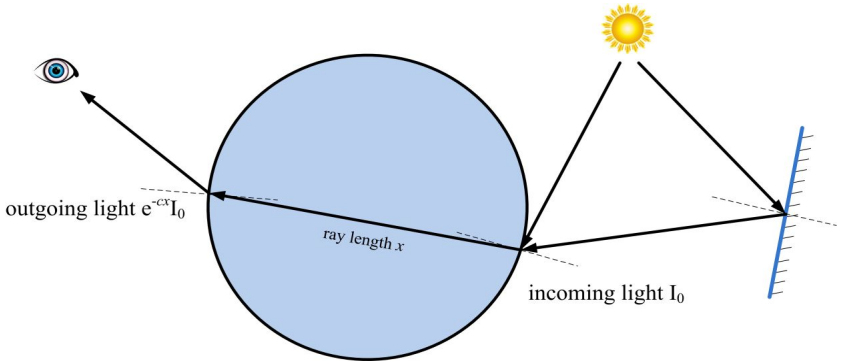
color=color+**trace(reflected_ray,objects,lights)*reflectivity**

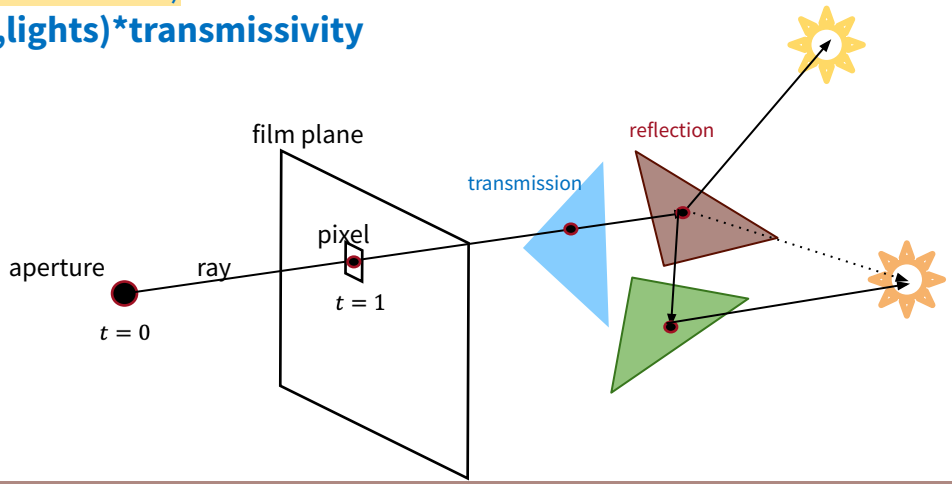
if (material at intersection has transmissivity>0)

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

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

return color





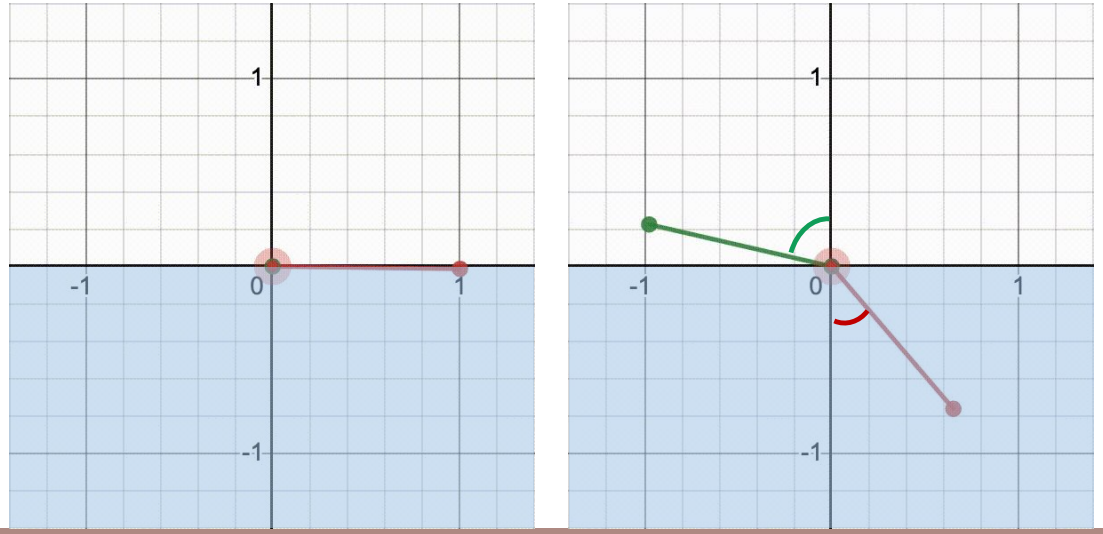
Total Internal Reflection

- Occurs when the term under the square root is negative
- No transmission – instead, we have reflection on the other side of the object

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

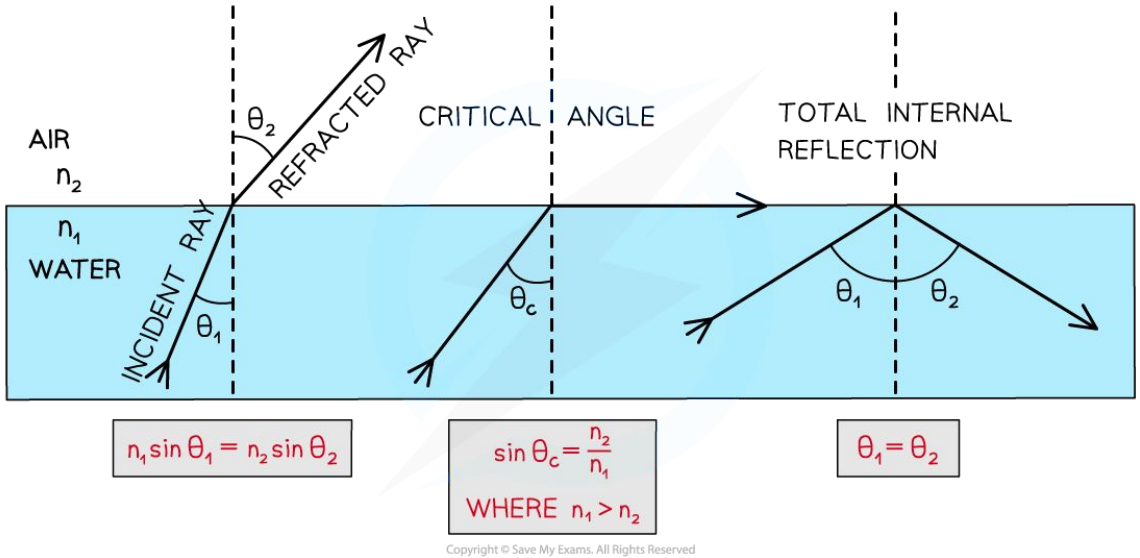
Total Internal Reflection

- Occurs when we go from a higher index of refraction (IOR) to a lower IOR
  - We use the boundary angle to determine this (the **critical angle**)



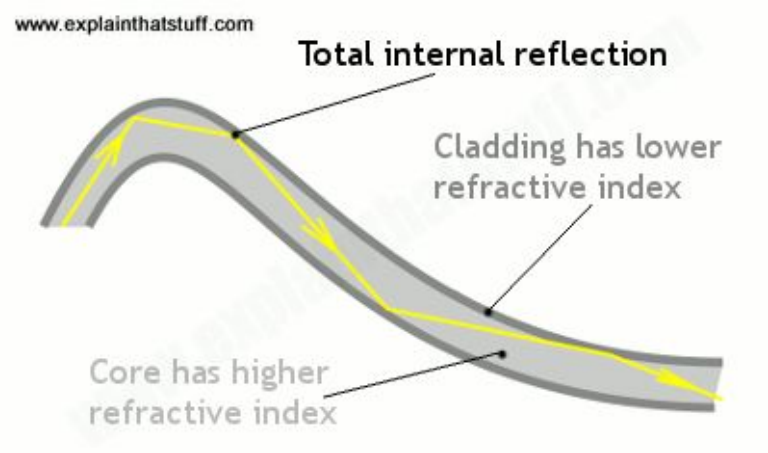
Total Internal Reflection

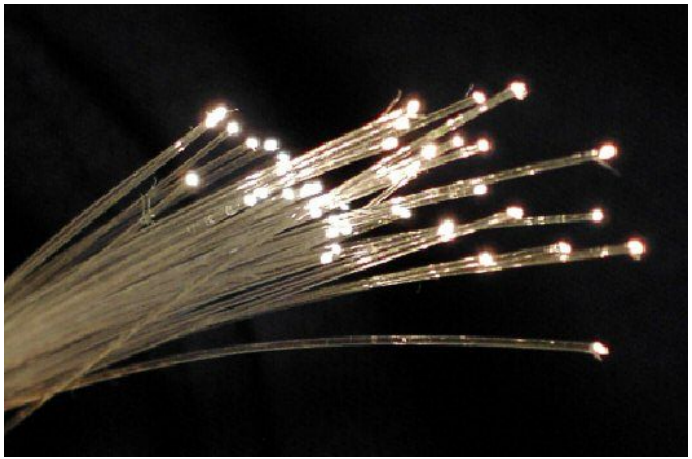
- Occurs when we go from a higher index of refraction (IOR) to a lower IOR
  - We use the boundary angle to determine this (the **critical angle**)



Total Internal Reflection

- Occurs when we go from a higher index of refraction (IOR) to a lower IOR
  - We use the boundary angle to determine this (the **critical angle**)



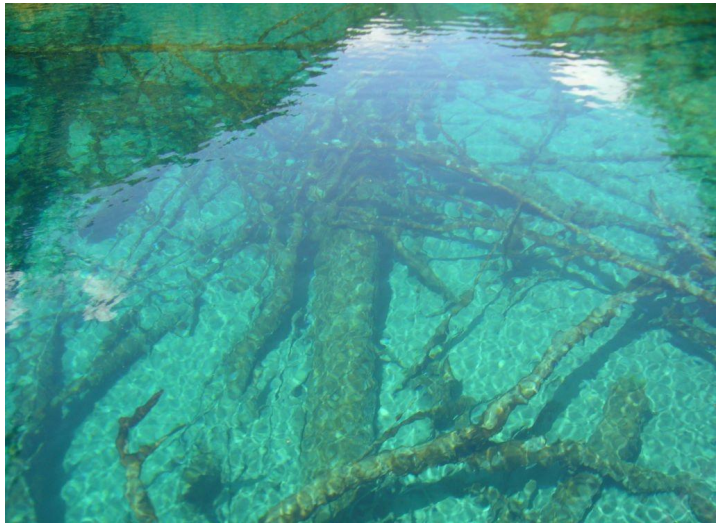


Reflection vs Transmission

- Objects can be (and often are) simultaneously reflective and transmissive

Reflection vs Transmission

- Objects can be (and often are) simultaneously reflective and transmissive
- Amount of transmission vs. reflection **decreases** as the viewing angle goes from **perpendicular** (overhead) to **parallel** (grazing)

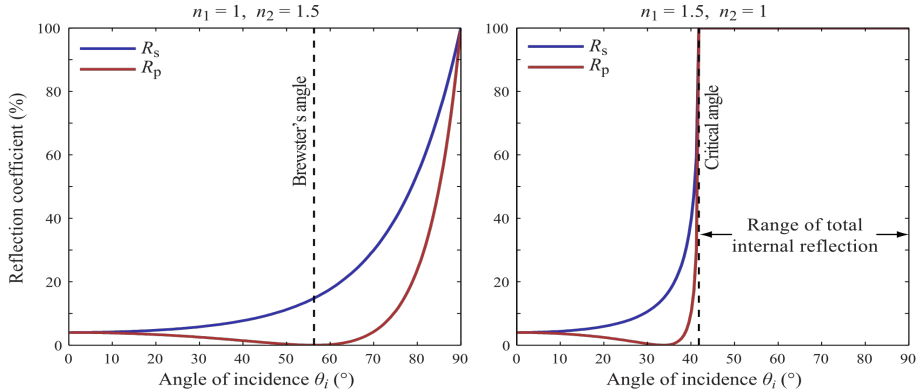


Reflection vs Transmission

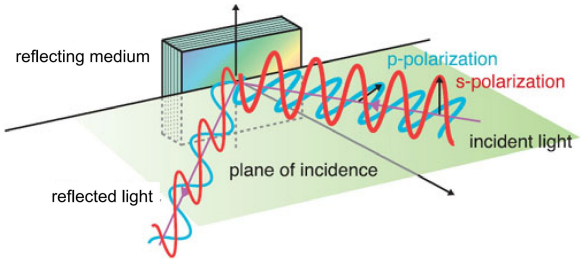
- Objects can be (and often are) simultaneously reflective and transmissive
- Amount of transmission **decreases** with respect to reflection as the viewing angle goes from **perpendicular** (overhead) to **parallel** (grazing)

Reflection vs Transmission

- Amount of transmission vs. reflection decreases 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 its orientation relative to the 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

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

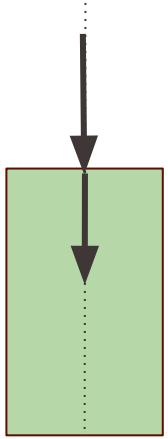
where

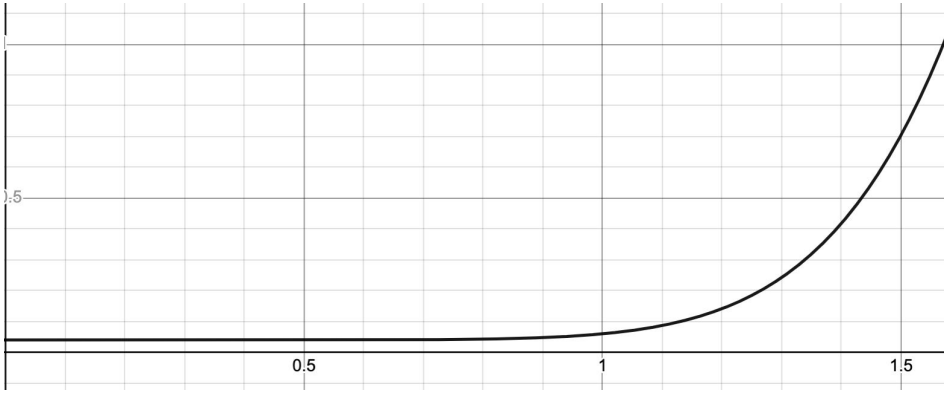
$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

$n_1 = 1.0$

$n_2 = 1.5$

$R_0 = .04$

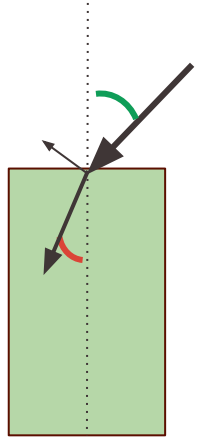




$n_1 = 1.0$

$n_2 = 1.5$

$R_0 = .04$



Lecture Outline

- Ray tracing review
- Raycasting & recursive raytracing
- Reflected & transmitted rays
  - Refraction
  - Total internal reflection
- Attenuation & caustics

Attenuation

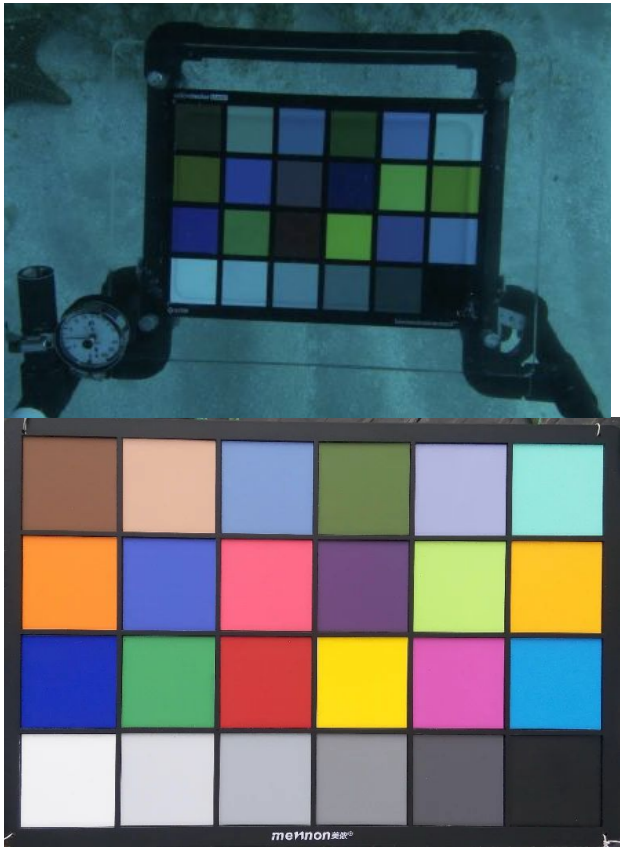
- When light is transmitted through objects, some light may get absorbed

Attenuation

- When light is transmitted through objects, some light may get absorbed
- Examples:
  - 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

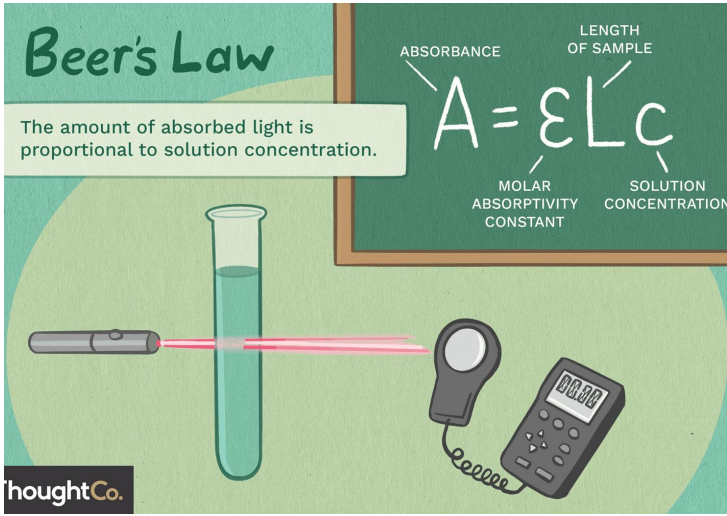


Attenuation



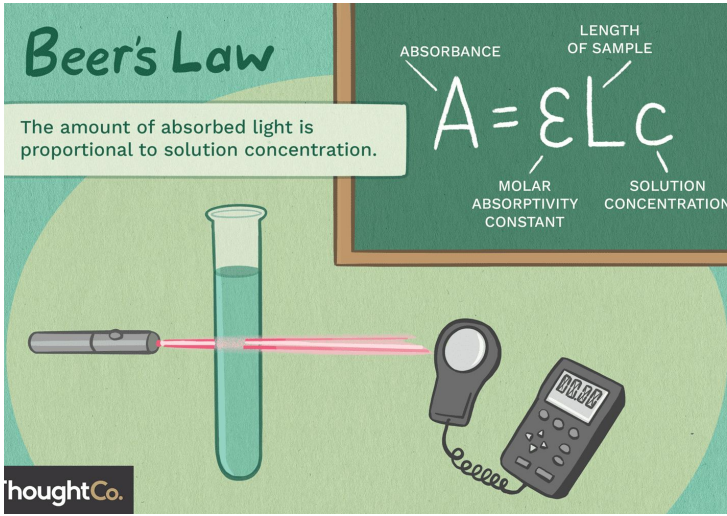
Attenuation

- **Beer's law** ⇒ describes the relationship between the absorbance of light by a solution and its concentration



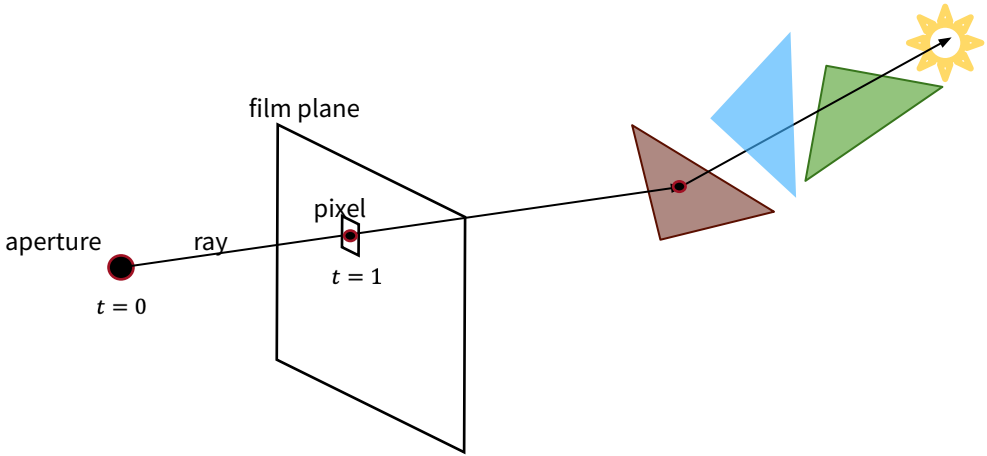
Attenuation

- **Beer's law** ⇒ describes the relationship between the absorbance of light by a solution and its concentration
  - Chemistry: given what the light looks like, what is the material
  - Graphics: given the material, what does the light look like through it



Attenuation: Shadow Rays

- Attenuation can approximate shadows of translucent objects
- Assuming the media is **homogeneous**, attenuation along a ray can be described by Beer's law



Caustics

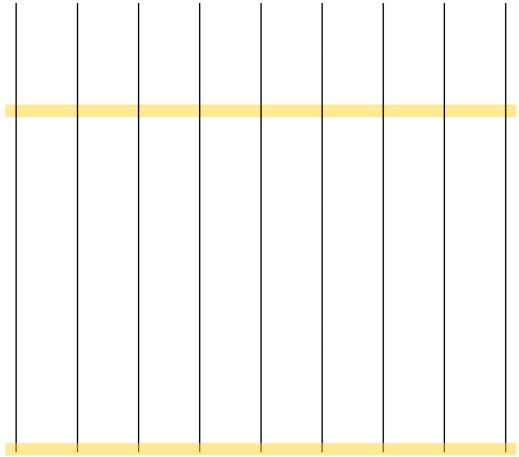
- Caused by refraction
- Light refracts before reaching the surface
  - Therefore, we can’t just use straight line shadow rays to each light course

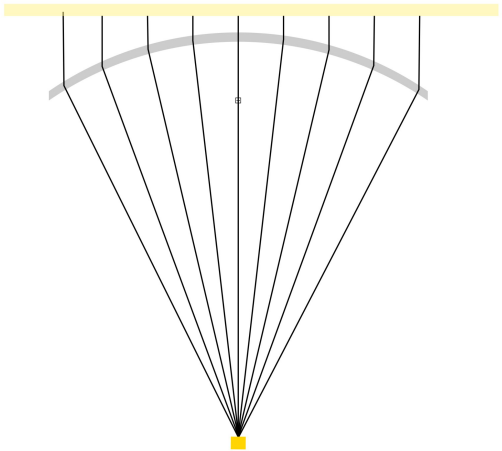
Caustics

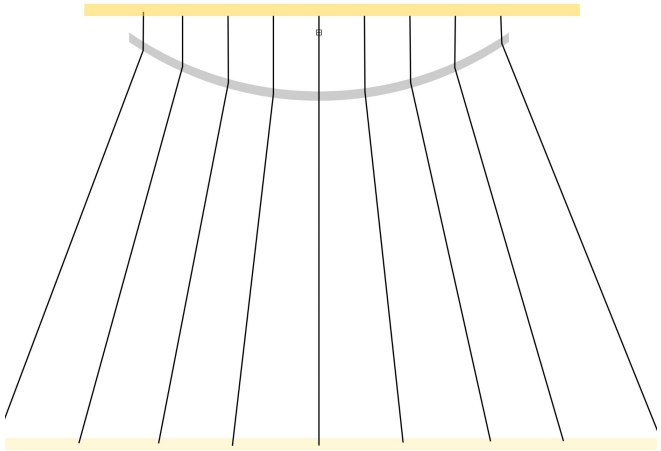
- Caused by refraction
- Light refracts before reaching the surface
  - Therefore, we can’t just use straight line shadow rays to each light course

Caustics

- (Shadow) rays leave an object at an angle depending on the property and **shape** of the material

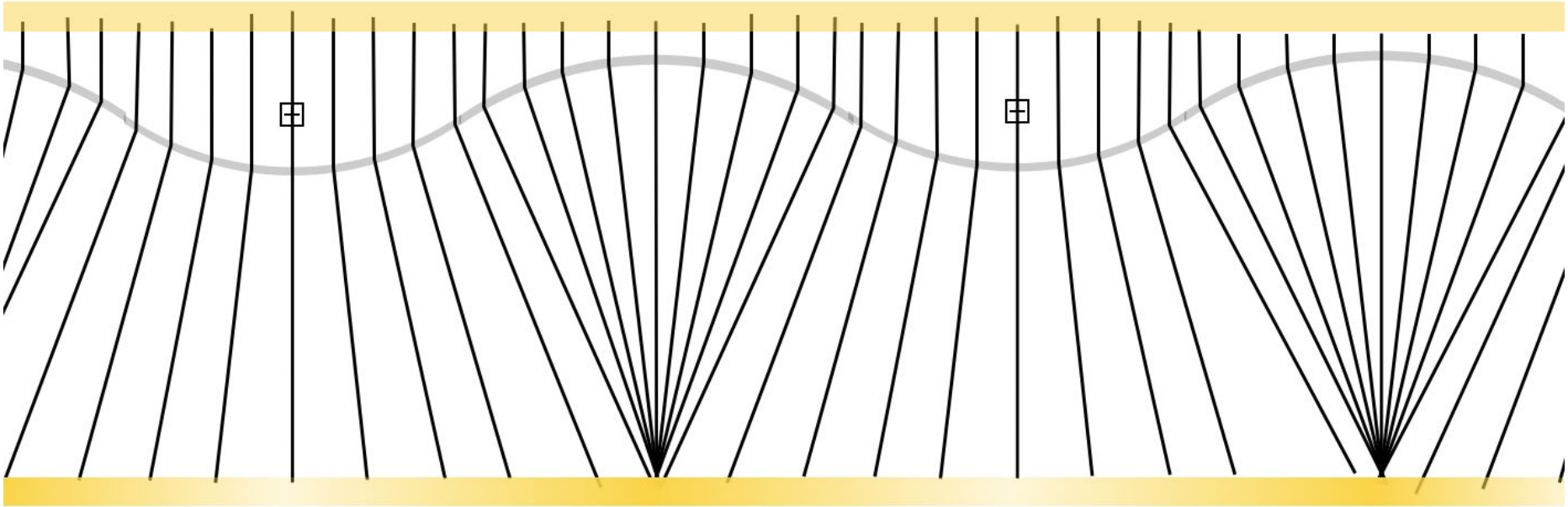




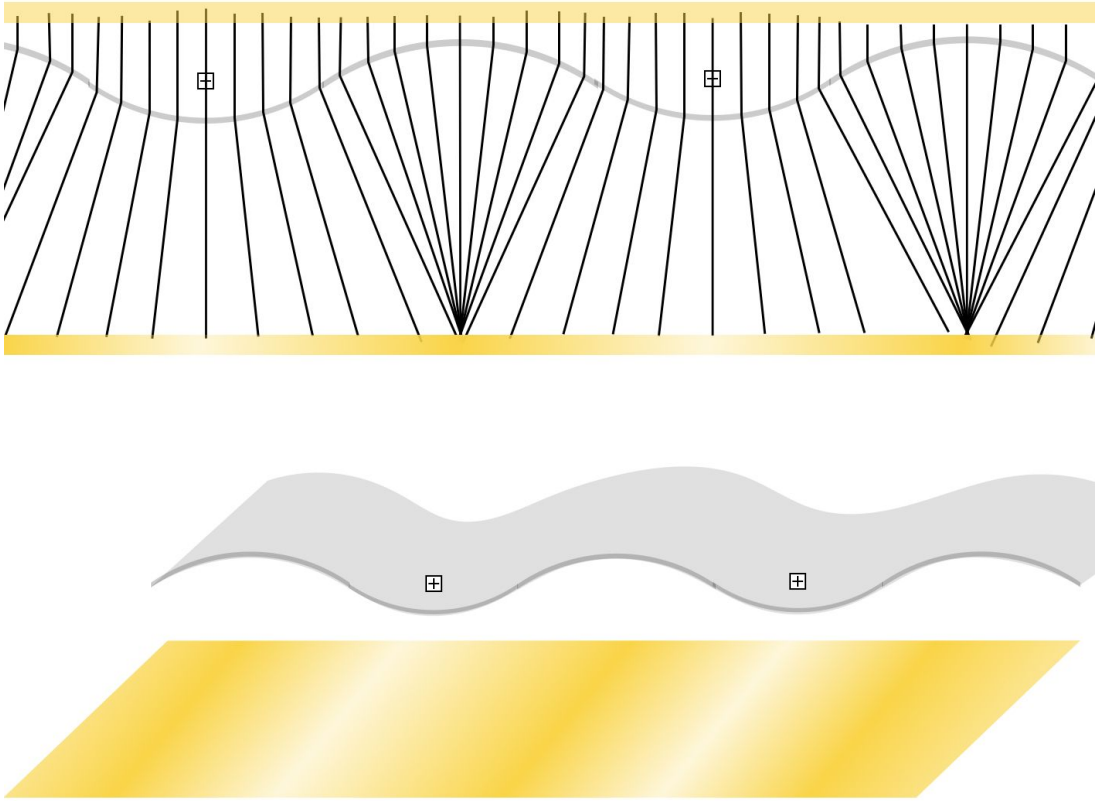


Caustics

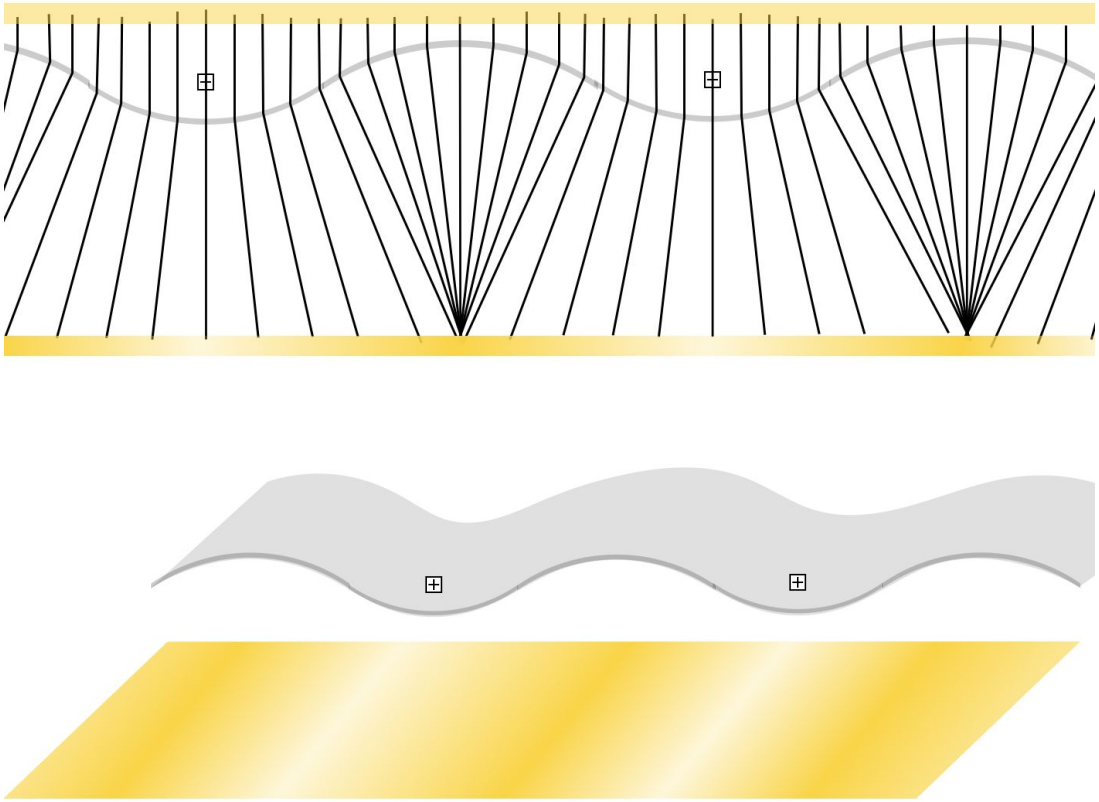
- (Shadow) rays leave an object at an angle depending on the property and **shape** of the material

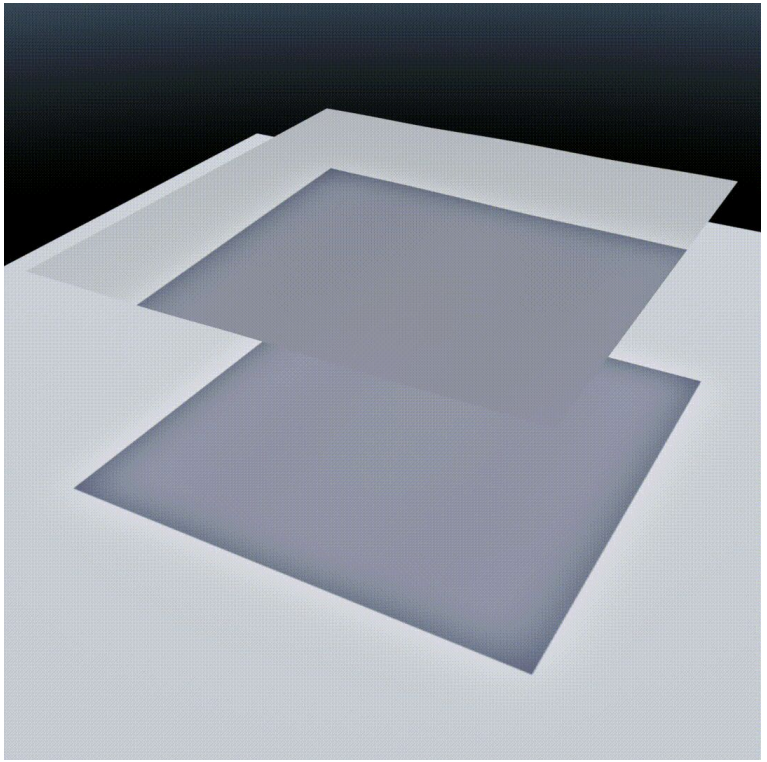


Caustics

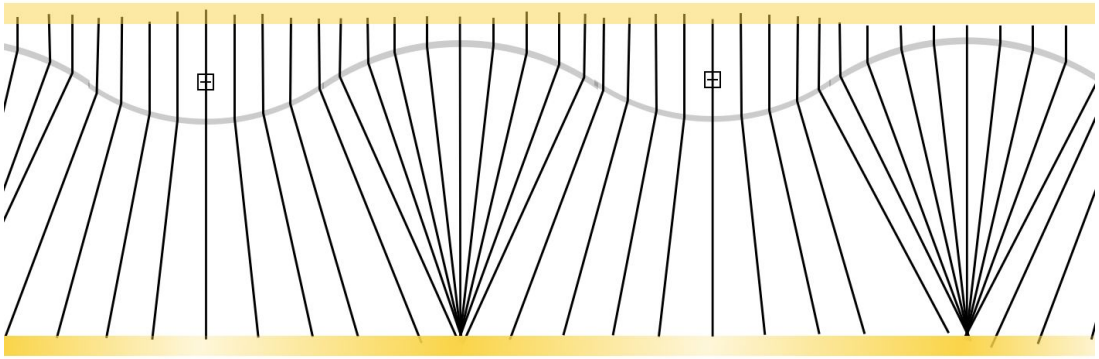


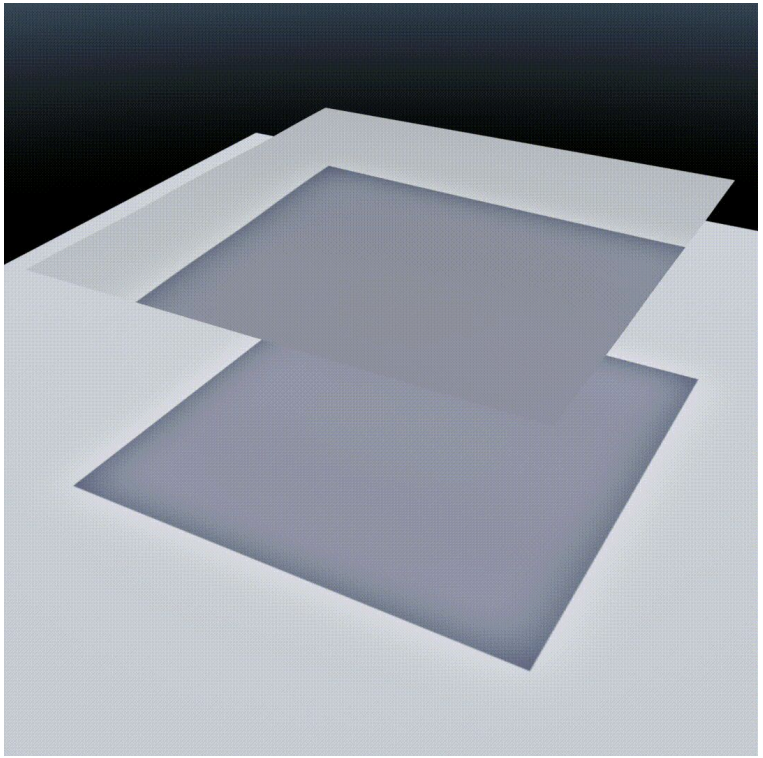
Caustics



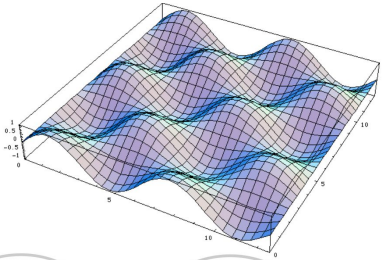


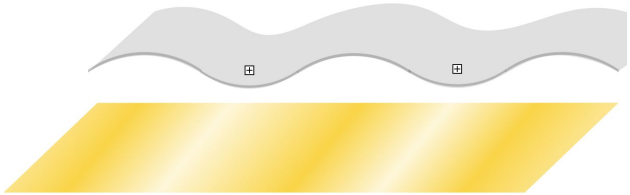
Caustics

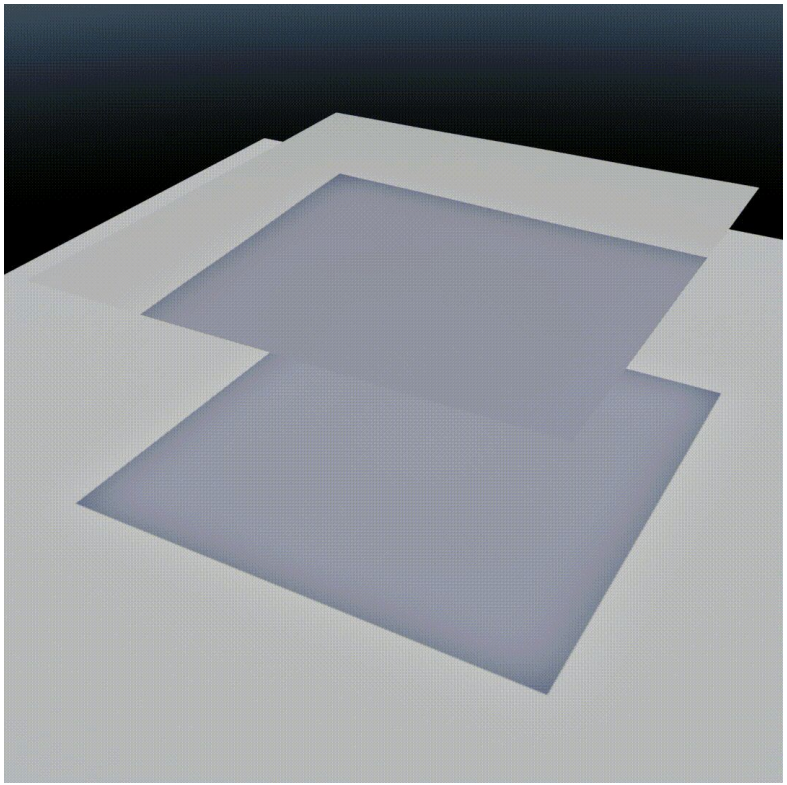




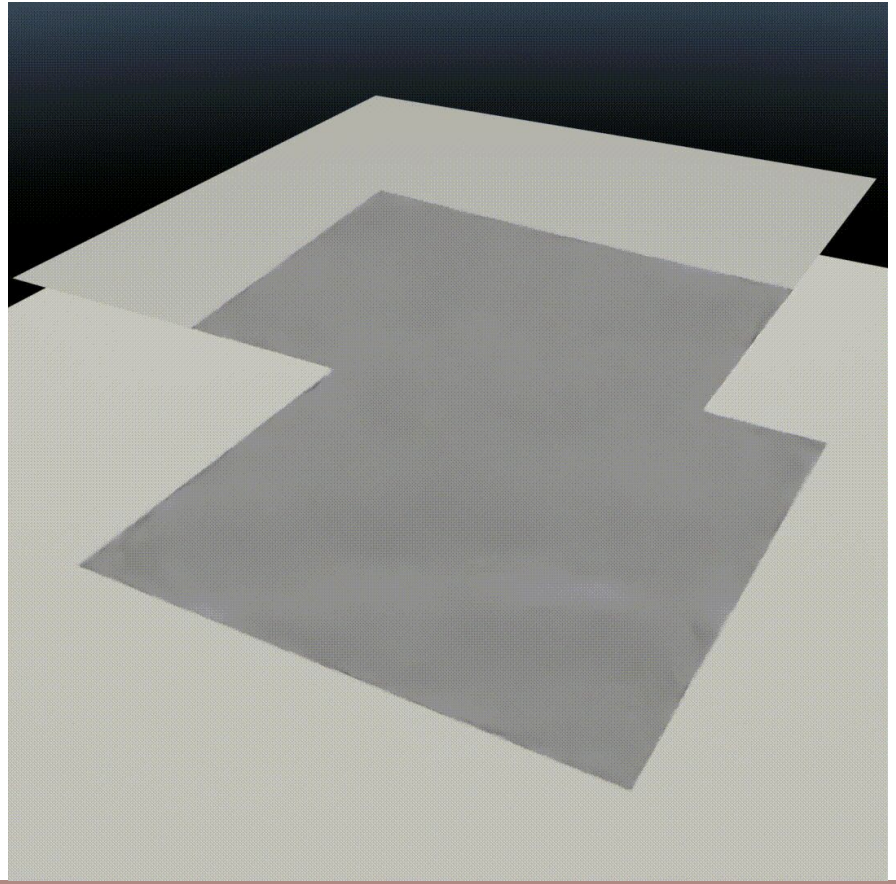
Caustics





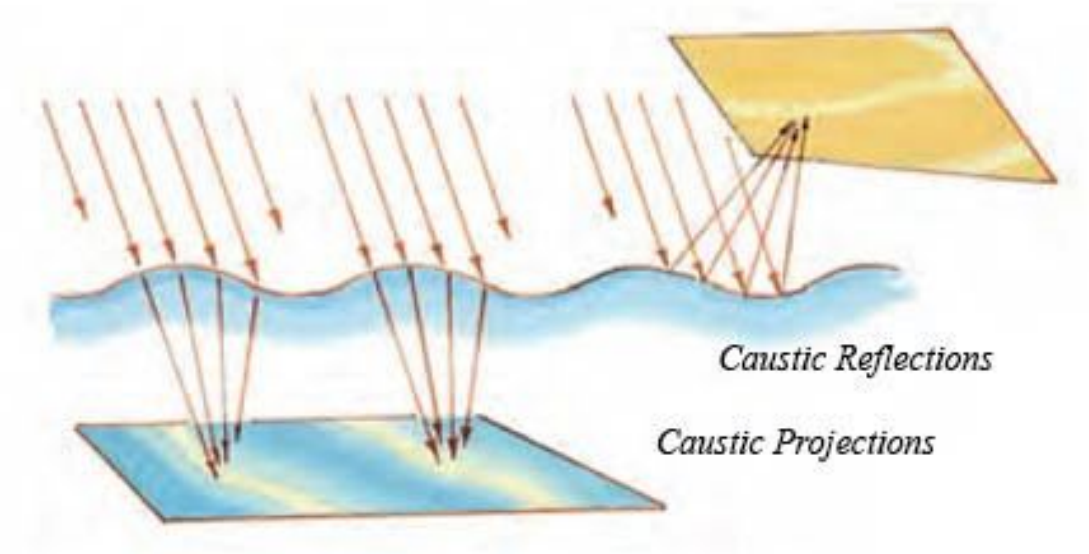


Caustics



Caustics

- (Shadow) rays leave an object at an angle depending on the property and **shape** of the material



Raytracing Summary

- **You did it!**

Raytracing Summary

- **You did it!** You know how to:
  - Send rays out from a camera and through a pixel

Raytracing Summary

- **You did it!** You know how to:
  - Send rays out from a camera and through a pixel
  - Find the closest intersection point with an object in the scene and this ray

Raytracing Summary

- **You did it!** You know how to:
  - Send rays out from a camera and through a pixel
  - Find the closest intersection point with an object in the scene and this ray
  - Send out more rays (shadow, reflected, transmissive)
    - With computer error correction
    - And considering material properties

Raytracing Summary

- **You did it!** You know how to:
  - Send rays out from a camera and through a pixel
  - Find the closest intersection point with an object in the scene and this ray
  - Send out more rays (shadow, reflected, transmissive)
    - With computer error correction
    - And considering material properties
- Think about making this cheap for the computer while maintaining high fidelity
  - Limiting number of rays (depth)
  - Prioritizing objects over one another for raytracing

Additional Resources + Exploring!

- ExplainThatStuff: Fiber Optics
- Fresnel Equations Derivations and Explanations
- Polarization
- Schlick's Approximation
- Underwater Image Restoration via Maximum Attenuation Identification
- Caustic light patterns inspire new glass artwork

- Fundamentals of Computer Graphics, Steve Marschner and Peter Shirley
  - Preview, Book