Raytracing Pseudocode
function traceImage (scene):
    for each pixel (i,j) in image
        $S = \text{PointInPixel}$
        $P = \text{CameraOrigin}$
        $d = (S - P)/\|S - P\|$
        $I(i,j) = traceRay(scene, P, d)$
    end for
end function
function traceRay(scene, P, d):
    \((t, N, mtrl) \leftarrow \text{scene}.\text{intersect}(P, d)\)
    \(Q \leftarrow \text{ray}(P, d)\) evaluated at \(t\)
    \(I = \text{shade}(mtrl, \text{scene}, Q, N, d)\)
    \(R = \text{reflectDirection}(N, -d)\)
    \(I \leftarrow I + mtrl.k_r \times \text{traceRay}(\text{scene}, Q, R)\)
    if ray is entering object then
        \(n_i = \text{index}_\text{of}_\text{air}\)
        \(n_t = mtrl.\text{index}\)
    else
        \(n_i = mtrl.\text{index}\)
        \(n_t = \text{index}_\text{of}_\text{air}\)
        if \((mtrl.k_t > 0 \text{ and } \text{notTIR}(n_i, n_t, N, -d))\) then
            \(T = \text{refractDirection}(n_i, n_t, N, -d)\)
            \(I \leftarrow I + mtrl.k_t \times \text{traceRay}(\text{scene}, Q, T)\)
        end if
    return \(I\)
end function
Thinking About Refraction

Remember Snell’s law?
• \( \eta_i \sin \theta_i = \eta_t \sin \theta_t \)

When does light bend?
• Must account for entering and leaving!
• How do we know if we’re entering or leaving? (hint: all geometry has a “front face” and a “back face”)

\[ Q \]
\[ N \]
\[ R \]
\[ -d \]
\[ \eta_i \]
\[ \eta_t \]
\[ \theta_i \]
\[ \theta_r \]
\[ \theta_t \]
Calculating Refraction

\[ \vec{M} = \frac{(\vec{N}\cos\theta_i - \vec{I})}{\sin\theta_i} \]

unit vector perpendicular to N in plane of I and N

\[ \vec{T} = \sin\theta_t\vec{M} - \cos\theta_t\vec{N} \]
Calculating Refraction

\[ \vec{T} = \frac{\sin \theta_t}{\sin \theta_i} (\vec{N}\cos \theta_i - \vec{I}) - \cos \theta_t \vec{N} \]

Let \( \eta_r = \frac{\eta_i}{\eta_t} = \frac{\sin \theta_t}{\sin \theta_i} \) (Snell’s Law)

\[ \vec{T} = (\eta_r \cos \theta_i - \cos \theta t) \vec{N} - \eta_r \vec{I} \]

\[ \cos \theta_i = \vec{N} \cdot \vec{I} \]

\[ \cos \theta_t = \sqrt{1 - \sin^2 \theta_t} = \sqrt{1 - \eta_r^2 \sin^2 \theta_i} \]

\[ \cos \theta_t = \sqrt{1 - \eta_r^2 (1 - (\vec{N} \cdot \vec{I})^2)} \]

\[ \vec{T} = \left( \eta_r (\vec{N} \cdot \vec{I}) - \sqrt{1 - \eta_r^2 (1 - (\vec{N} \cdot \vec{I})^2)} \right) \vec{N} - \eta_r \vec{I} \]
Determining TIR

\[ \vec{T} = \left( \eta_r (\vec{N} \cdot \vec{I}) - \sqrt{1 - \eta_r^2 (1 - (\vec{N} \cdot \vec{I})^2)} \right) \vec{N} - \eta_r \vec{I} \]

- TIR occurs when index of refraction of current medium ($\eta_i$) > index of refraction of other medium ($\eta_t$)
  - going from more dense to less dense medium
- Critical angle is value of $\sin \Theta_i$ when $\sin \Theta_t$ is 1
- Critical angle $\Theta_c = \sin^{-1}(\eta_t/\eta_i)$
- TIR occurs when square root of expanded $\cos \Theta_t$ is imaginary
function shade(mtrl, scene, \( Q \), \( N \), \( d \)):
    I ← mtrl.\( k_e \) + mtrl. \( k_a \) * scene->\( I_a \)
    for each light source \( l \) do:
        atten = \( l \) -> distanceAttenuation( \( Q \) ) * \( l \) -> shadowAttenuation( scene, \( Q \) )
        I ← I + atten*(diffuse term + spec term)
    end for
    return I
end function
function PointLight::shadowAttenuation(scene, P)
    d = (l.position - P).normalize()
    (t, N, mtrl) ← scene.intersect(P, d)
    Q ← ray(t)
    if Q is before the light source then:
        atten = 0
    else
        atten = 1
    end if
    return atten
end function
Some Additional Notes

The raytracer skeleton code is extensive but largely undocumented

• Taking time to look through the code to understand what it does is essential

• Mathematical elegance doesn’t mean there’s a simple codebase
Passing by Reference

Many important values are passed by reference!

• Look carefully to determine where/how values are being updated
• Very common in C and C++ codebases
tmax and tmin

Parametric values that define the bounding box around the scene

- Returned t values are within this range

Scene can be further subdivided for additional intersect optimizations!
Debugging Visually: What Happened?
Casting Shadow Rays

at what $t$ does the ray hit an object?
Casting Shadow Rays

at what $t$ does the ray hit an object?

if lucky: $\{-1.2, 0.0\}$
if unlucky: $\{-1.2, 1e-12\}$
Shadow Rounding Error

Classic fix: move slightly in normal direction before shooting shadow ray

- RAY_EPSILON provided for this
But Shadows Don’t Look Like This!
Hard vs Soft Shadows
Calculate Penumbra

Use full lighting equation or calculate geometrically (not required for A1!)