Recall

- can split illumination into direct and indirect
Recall

- surfaces respond to this illumination
Recall

- in reality, both direct and indirect light comes from many directions
Recall

- ... and surface response can vary drastically based on material!

Source: Mitsuba 0.5 documentation, https://www.mitsuba-renderer.org/docs.html
Recall

... and surface response can vary drastically based on material!

Source: Mitsuba 0.5 documentation, https://www.mitsuba-renderer.org/docs.html
Recall – path tracing

- Step 1. shoot ray from eye
  - ray attenuation = (1, 1, 1)
Recall – path tracing

- Step 2. accumulate direct contributions
  - keep track of material response per light

shadow ray
Recall – path tracing

- Step 3. scale ray attenuation by material
  - estimate indirect illumination by shooting a ray
Recall – path tracing

- Step 4. for new hit point, repeat
Path tracing algorithm

... 

for(number_samples)
    attenuation = Color(1.f, 1.f, 1.f);
    ray = generateNewRay( ... );
    while(depth < max_depth) {
        HitRecord hit;
        bvh.intersect(hit, ray);
        result += shade(...) * attenuation; // box filter
        attenuation *= mat_color;
        ray = hemiRay(...);
        depth++
    }

result /= number_samples; // box filter

// tone map!
image.set(pixel, result);
Path tracing considerations

- shoot many rays per pixel
  - samples pixel area = anti-aliasing
  - (effectively) samples material, (area) lights, indirect illumination = less noise in image

- stopping
  - max depth reached (5-6 good, scene-dependent)
  - when attenuation below threshold
    - must be careful about brightest light value

- selecting new shooting direction
  - based on material
Lambertian material

- \( L_{\text{Reflected}} = C_{\text{Light}} \cdot C_{\text{material}} \cdot \cos \theta \)
- \( \cos \theta = \hat{N} \cdot \hat{L} \), after normalization
Dielectric material

- Ex: glass, water, diamond, etc
- Incoming energy is split into \textbf{R}eflected and \textbf{T}ransmitted
- angular dependence based on indices of refraction – Snell’s law
Snell’s Law

- speed of light is different in different media
- light is an EM wave, different component will have different speed
- result: the light bends at the interface

\[
\frac{\sin \theta_1}{\sin \theta_2} = \frac{v_1}{v_2} = \frac{n_2}{n_1}
\]

Snell’s Law

- Total internal reflection
  - gives diamonds their shine
  - occurs beyond critical angle \( \theta_c = \sin^{-1} \left( \frac{n_2}{n_1} \sin \theta_2 \right) = \sin^{-1} \frac{n_2}{n_1} \)

## Examples of coefficients

<table>
<thead>
<tr>
<th>Material</th>
<th>Index of refraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacuum</td>
<td>1.0</td>
</tr>
<tr>
<td>Water</td>
<td>1.3330</td>
</tr>
<tr>
<td>Acetone</td>
<td>1.36</td>
</tr>
<tr>
<td>Ethanol</td>
<td>1.361</td>
</tr>
<tr>
<td>Silicone Oil</td>
<td>1.52045</td>
</tr>
<tr>
<td>Water Ice</td>
<td>1.31</td>
</tr>
<tr>
<td>Fused Quartz</td>
<td>1.458</td>
</tr>
<tr>
<td>Pyrex</td>
<td>1.470</td>
</tr>
<tr>
<td>Acrylic Glass</td>
<td>1.49</td>
</tr>
<tr>
<td>Amber</td>
<td>1.55</td>
</tr>
<tr>
<td>Diamond</td>
<td>2.419</td>
</tr>
</tbody>
</table>

Source: Mitsuba 0.5 documentation, https://www.mitsuba-renderer.org/docs.html
Fresnel Coefficients

- Power is reduced based on reflected and transmitted angles

Fresnel Coefficients

- power is reduced based on reflected and transmitted angles
- use Schlick's approximation (reflected amount)

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

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

\[
\cos \theta = \hat{H} \cdot \hat{V}
\]

\[
\vec{H} \text{ – half vector between incident light and view direction } \vec{V}
\]

- transmitted is then \( T(\theta) = 1 - R(\theta) \)

result = normal lambertian (or other) shading
Ray rray = reflect_ray( ray, ... );
if( tir(ray) ) {
    // kr = 1, kt = 0
    result += traceRay( rray, depth + 1, ... );
} else {
    // kr = R(theta) using Schlick’s approximation
    // kt = 1 - kr
    result += kr*traceRay( rray, depth + 1, ... );
}

Ray tray = transmit_ray( ray, ... );
result += kt*traceRay( tray, depth + 1, ... );
result = normal
Ray ray = reflect_ray(ray);
if(tir(ray)) {
  kr = 1, kt = 0
  result += traceRay(ray, depth + 1, ...);
} else {
  kr = R(theta) using Schlick's approximation
  kt = 1 - kr
  result += kr * traceRay(ray, depth + 1, ...);
  Ray tray = transmit_ray(ray, ...);
  result += kt * traceRay(tray, depth + 1, ...);
}

See supplemental slides for derivation of reflected and transmitted rays using Snell’s Law
bool tir( const Ray& ray ) {
    float cosTheta = -ray.direction() * normal;
    float eta;
    if( cosTheta > 0 ) {
        // ior_* - index of refraction, aka n
        eta = ior_from / ior_to;
    } else {
        eta = ior_to / ior_from;
    }
    return ( (1.f - (1.f - cosTheta*cosTheta) / (eta*eta)) < 0.f );
}
bool transmit_ray( const Ray& ray, ... ) {
    // compute eta and cosTheta the same as in tir
    // if cosTheta < 0: flip normal, eta, cosTheta

    tmp = 1.f - (1.f - cosTheta1*cosTheta1)/(eta*eta);
    cosTheta2 = sqrt(tmp);

    Ray tray( hit_point, ray.direction() /
               (cosTheta2 - cosTheta1/eta)*normal;
    return tray;
}
there are more efficient ways of doing it than the above pseudocode

should probably use a stack of indices

- nested refraction
- eye starts within media

directions of the normal matter!

Famous Material Models

- surfaces:
  - Lambertian
  - Cook-Torrence
  - Anisotropic Ward
  - Other microfacet Distributions
    - Ashikhmin-Shirley, 2000
    - Walter et. al, 2007

End