Course announcements

• Programming assignment 0 available on Canvas tonight.
  - Ungraded, no due date.
  - Used to set up rendering environment and github-based submission system.
  - Should take no more than 1-2 hours max.

• Programming assignment 1 will be posted on Friday 1/27, will be due two weeks later.

• Take-home quiz 1 available online, will be due Tuesday 1/31.

• All other logistics (office hours, recitations, reading groups) on course website.

• Office hours on Friday: Yannis, 2-4 pm.
Overview of today’s lecture

• Leftover from previous lecture: intersections, meshes, acceleration structures.
• Basics of shading.
• Basic reflection models.
Slide credits

Most of these slides were directly adapted from:

• Wojciech Jarosz (Dartmouth).
Recap: Raytracing

Ray Generation

Intersection

Shading
Shading

When ray hits a surface we perform \textit{lighting/shading}

Determine “what color/light should we see at this location?”

Surfaces can scatter and/or emit light

- Surface emits light? just return emitted color  
  (determined by the material)

- Surface scatters/reflects/refracts light?  
  (recursively) trace a ray in a scattering direction 
  (determined by the underlying material)
Overview

Diffuse shading
Specular reflection
Refraction
Diffuse emission
Light-material interactions
The reflection equation

Reflected radiance is a (hemi)spherical integral of incident radiance from all directions

\[ L_r(x, \vec{ω}_r) = \int_{H^2} f_r(x, \vec{ω}_i, \vec{ω}_r) L_i(x, \vec{ω}_i) \cos θ_i \, d\vec{ω}_i \]

This describes a local illumination model.
The BSDF

Bidirectional Scattering Distribution Function

- informally: how much the material scatters light coming from one direction \( \mathbf{l} \) into some other direction \( \mathbf{v} \), at each point \( \mathbf{p} \)
The BSDF

**Bidirectional Scattering Distribution Function**

- informally: how much the material scatters light coming from one direction \( l \) into some other direction \( v \), at each point \( p \)

You’ll also see **BRDF** \( R \) for reflectance
Real-world materials

Metals

Dielectric
Real-world materials

Metals

Dielectric
Idealized material models

Diffuse reflection
- light is reflected in all directions
- colored by surface color

Smooth specular reflection/refraction (e.g., chrome, glass, glaze/varnish)
- light reflected/refracted only in a single direction
- colored by source color
Idealized materials
Diffuse reflection
Diffuse reflection
Lambertian reflection

Also called ideal diffuse reflection
Basic Ray Tracing Pipeline

- Ray Generation
- Intersection
- Shading

What direction should we trace a new ray towards?
Basic Ray Tracing Pipeline

What direction should we trace a new ray towards?
• Pick a direction at random!
Sampling Lambertian scattering

From what distribution should we sample directions?
Sampling Lambertian scattering

From what distribution should we sample directions?

- Probability proportional to \( \cos(\hat{n} \cdot \hat{\omega}) \).
- Even though BSDF scatters to all directions uniformly, we need to account for foreshortening.

How do we sample directions based on this \textit{cosine-weighted} distribution?
Sampling Lambertian scattering

Generate points uniformly on sphere

(unit directions)
Sampling Lambertian scattering

Generate points uniformly on sphere (unit directions)

Add unit normal

unit normal
Sampling Lambertian scattering

Generate points uniformly on sphere (unit directions)

Add unit normal normalize
Rejection Sampling a Sphere

Vector3D v;

v.x = 1-2*randf();
v.y = 1-2*randf();
v.z = 1-2*randf();
Vector3D v;
do{
    v.x = 1-2*randf();
    v.y = 1-2*randf();
    v.z = 1-2*randf();
} while(v.length^2() > 1)
Rejection Sampling a Sphere

```
Vector3D v;
do
{
    v.x = 1-2*randf();
    v.y = 1-2*randf();
    v.z = 1-2*randf();
} while(v.length^2() > 1)
```
Vector3D v;
do {
    v.x = 1-2*randf();
    v.y = 1-2*randf();
    v.z = 1-2*randf();
} while(v.length2() > 1)
// Project onto sphere
v /= v.length();
Sampling a Sphere using normal samples

Vector3D v;
v.x = randnf();
v.y = randnf();
v.z = randnf();
// Project onto sphere
v /= v.length();

- No rejection sampling required (no while loop).
- Need to use normal, rather than uniform, samples.
Diffuse shading
Scene::trace(Ray ray)
    hit = surfaces.intersect(ray);
    if hit
        [col, sRay] = hit->mat->scatter(ray)
        return col * trace(sRay);
    else
        return backgroundColor;
Rounding errors

Don’t fall victim to one of the classic blunders:

What’s going on?

affectionately called “shadow acne”
Rounding errors

Don’t fall victim to one of the classic blunders:

- Intersection inside due to floating point error
- Ray blocked by self-intersection
Shadow rounding errors

Solution: recursive rays start a tiny distance from the surface

Do this by limiting the $t$ range
Specular/Mirror reflection
What direction should we trace a new ray towards?
Basic Ray Tracing Pipeline

Ray Generation

Intersection

Shading

specular object

light reflects towards only one direction

What direction should we trace a new ray towards?
• Just use law of mirror reflection, no need for random selection!
Mirror reflection

Consider perfectly shiny surface
- there’s a reflection of other objects

Can render this using recursive ray tracing
- to find out mirror reflection color ask:
  “what color is seen from surface point in reflection direction?”
Mirror reflection

Evaluated by tracing a new ray:

\[ L_r = k_r \text{trace}(p, r) \]

- reflected light
- reflection scaled down/tinted
- recursive call
Mirror reflection

Evaluated by tracing a new ray:

\[ L_r = k_r \text{trace}(p, r) \]

Implementation details:
- don’t self-intersect \((t_{\text{min}} > \epsilon)\)
- don’t recurse indefinitely
Same pseudo-code

```cpp
Scene::trace(Ray ray)
    hit = surfaces.intersect(ray);
    if hit
        [col, sRay] = hit->mat->scatter(ray)
        return col * trace(sRay);
    else
        return backgroundColor;
```
Diffuse & mirror spheres
Mirror reflection

Assume \( \mathbf{n} \) is unit length

What two properties defined reflection direction?
Mirror reflection

Assume \( \mathbf{n} \) is unit length

What two properties defined reflection direction?

- co-planar view direction, reflected direction, and normal direction
- equal angles between normal-view directions, and normal-reflected directions
Mirror reflection

Assume \( \mathbf{n} \) is unit length
Mirror reflection

- \( r \): reflected direction
- \( d \): view direction
- \( n \)
- \( \theta \)
- \( p \)
Mirror reflection

\[ r = -n(n \cdot d) \]

Reflected direction: \( r \)
View direction: \( d \)
Normal: \( n \)
Mirror reflection

\[ -2n(n \cdot d) \]
Mirror reflection

Assumes $\mathbf{n}$ is unit length

$$\mathbf{r} = -2\mathbf{n}(\mathbf{n} \cdot \mathbf{d}) + \mathbf{d}$$
Specular refraction
Refraction
Refraction
**Index of Refraction**

Speed of light in vacuum / speed of light in medium

<table>
<thead>
<tr>
<th>Material</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacuum</td>
<td>1</td>
</tr>
<tr>
<td>Air at STP</td>
<td>1.00029</td>
</tr>
<tr>
<td>Ice</td>
<td>1.31</td>
</tr>
<tr>
<td>Water</td>
<td>1.33</td>
</tr>
<tr>
<td>Crown glass</td>
<td>1.52 - 1.65</td>
</tr>
<tr>
<td>Diamond</td>
<td>2.417</td>
</tr>
</tbody>
</table>
Specular transmission/refraction

Materials like water, glass, etc., also refract/bend light

Trace a recursive ray in the refraction direction

\[ L_t = k_t \text{trace}(p, t) \]
Specular transmission/refraction

Snell’s law

\[ \eta_1 \sin \theta_1 = \eta_2 \sin \theta_2 \]
Snell’s law

\[ t = \frac{n_1}{n_2} \left( d - (d \cdot n) n \right) - n \sqrt{1 - \frac{n_1^2}{n_2^2} \left(1 - (d \cdot n)^2\right)} \]
Index of Refraction

Speed of light in vacuum / speed of light in medium

<table>
<thead>
<tr>
<th>Some values of $\eta$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacuum</td>
<td>1</td>
</tr>
<tr>
<td>Air at STP</td>
<td>1.00029</td>
</tr>
<tr>
<td>Ice</td>
<td>1.31</td>
</tr>
<tr>
<td>Water</td>
<td>1.33</td>
</tr>
<tr>
<td>Crown glass</td>
<td>1.52 - 1.65</td>
</tr>
<tr>
<td>Diamond</td>
<td>2.417</td>
</tr>
</tbody>
</table>

These are actually wavelength dependent!
Dispersion
Refraction in a Waterdrop
Double rainbow all the way across the sky!
Dispersion: “Halos” and “Sun dogs”
Halos and Sundogs

Sundogs are produced by hexagonal plate shaped ice crystals drifting with their large faces nearly horizontal.

Sundog rays pass through crystal faces inclined 60° to each other.

Rays are deviated by 22° or more. Red is deviated least, giving the 'dog' a red inner edge.

All crystals refract the sun's rays but we see only those that glint their light towards our eyes. They are the crystals that, to us, are 22° or more from the sun and at the same altitude. Their collective glints form the sundogs.
Dispersion
What is this dark circle?

source: mrreid.org
What is this dark circle?

Called “Snell’s window”

Caused by total internal reflection

source: mrreid.org
Recall...

Snell’s law

\[
t = \frac{\eta_1}{\eta_2} \left( d - (d \cdot n) n \right) - n \sqrt{1 - \frac{\eta_1^2}{\eta_2^2} \left(1 - (d \cdot n)^2\right)}
\]
When can total internal reflection happen?

Can only happen when the ray starts in the higher index medium.
Total Internal Reflection

source: mrreid.org
Total Internal Reflection
Total Internal Reflection
Total Internal Reflection
Reflection vs. Refraction

How much light is reflected vs. refracted?

- in reality determined by “Fresnel equations”
Fresnel Equations

*Reflection* and *refraction* from smooth *dielectric* (e.g. glass) surfaces

*Reflection* from *conducting* (e.g. metal) surfaces

Derived from Maxwell equations

Involves polarization of the wave
Fresnel Equations for Dielectrics

Reflection of light polarized parallel and perpendicular to the plane of refraction

\[
\rho_\parallel = \frac{\eta_2 \cos \theta_1 - \eta_1 \cos \theta_2}{\eta_2 \cos \theta_1 + \eta_1 \cos \theta_2}
\]

\[
\rho_\perp = \frac{\eta_1 \cos \theta_1 - \eta_2 \cos \theta_2}{\eta_1 \cos \theta_1 + \eta_2 \cos \theta_2}
\]

reflected:  \[ F_r = \frac{1}{2} \left( \rho_\parallel^2 + \rho_\perp^2 \right) \]

refracted:  \[ F_t = 1 - F_r \]
What’s happening in this photo?

source: flickr user neofob
Polarizing Filter
Polarization

Without Polarizer

With Polarizing Filter

source: photography.ca
Polarization

Without Polarizer

With Polarizing Filter

source: wikipedia
Effect of Polarization
Effect of Polarization
Fresnel Equations for Dielectrics

Reflection of light polarized parallel and perpendicular to the plane of refraction

\[
\rho_{\parallel} = \frac{\eta_2 \cos \theta_1 - \eta_1 \cos \theta_2}{\eta_2 \cos \theta_1 + \eta_1 \cos \theta_2}
\]

\[
\rho_{\perp} = \frac{\eta_1 \cos \theta_1 - \eta_2 \cos \theta_2}{\eta_1 \cos \theta_1 + \eta_2 \cos \theta_2}
\]

reflected: \[F_r = \frac{1}{2} \left( \rho_{\parallel}^2 + \rho_{\perp}^2 \right)\]

refracted: \[F_t = 1 - F_r\]

- The Shirley book uses a faster approximation (Schlick), but to get full accuracy you’d need to use these equations
Fresnel equations for glass

$n_1 = 1, \ n_2 = 1.5$

$R_s, R_p$

Brewster's angle

Range of total internal reflection

$n_1 = 1.5, \ n_2 = 1$

$R_s, R_p$

Critical angle

Range of total internal reflection
Fresnel reflection
Fresnel reflection/refraction
During ray tracing, how do we decide whether to reflect or refract?
During ray tracing, how do we decide whether to reflect or refract?

- Randomly! Using Fresnel coefficients as probabilities.
So Far: Idealized BRDF Models

Diffuse

Specular Reflection and Refraction
Real materials are more complex
Rough materials

In reality, most materials are neither perfectly diffuse nor specular, but somewhere in between

- Imagine a shiny surface scratched up at a microscopic level

We will look at a more principled way to handle this later.

For now, we can easily approximate one important characteristic: blurred reflections

- Compute reflection direction, then add a random offset to it
- Sample random offset from sphere. Scale it to increase/decrease fuzziness
Diffuse & mirror spheres
Diffuse & rough mirror spheres
Diffuse & rough mirror spheres
Diffuse & rough mirror spheres
Putting it together
Lighting
Lighting

So far, the sky or background has been the only source of emitted light

But we can easily make any surface a light source!

- Just return an emitted color when a ray hits that surface
- Add a function to material that returns emitted color
  - Returns black (all zeros) for regular (non-emissive) surfaces
  - Color will often be greater than (1,1,1)
- Also possible for surfaces to emit & scatter (but not common)
Pseudo-code

Scene::trace(Ray ray)
    hit = surfaces.intersect(ray);
    if hit
        [col, sRay] = hit->mat->scatter(ray)
        return col * trace(sRay);
    else
        return backgroundColor;
Pseudo-code

```cpp
Scene::trace(Ray ray)
    hit = surfaces.intersect(ray);
    if hit
        emit = hit->mat->emit(ray)
        [col, sRay] = hit->mat->scatter(ray)
        return emit + col * trace(sRay);
    else
        return backgroundColor;
```

100