Materials & Shadows

Steve Rotenberg
CSE168: Rendering Algorithms
UCSD, Winter 2017
In an earlier lecture, we discussed diffuse surfaces. We looked at the idealized *Lambertian* diffuse case using *Lambert’s Cosine Law*

\[ L_r = \frac{\rho}{\pi} \cdot L_i \cdot \cos \theta_i \]

- $L_r$ is the intensity of the reflected light
- $L_i$ is the intensity of the incident light
- $\rho$ is the albedo (essentially the ‘color’ of the material)
- $\theta_i$ is the angle between the incident light and the normal
We also discussed the Oren-Nayar model, which increases the reflection in the direction of the light source to account for shadow-hiding, inter-reflection, and other properties of rough surfaces.
Fresnel Surfaces

• We also discussed idealized Fresnel surfaces which are the perfectly smooth boundary between a dielectric (such as air, glass, or water) and another dielectric, or a dielectric and a metal.

• In the first case (dielectric-dielectric) an incoming beam of light will split into a reflected and transmitted (refracted) beam.

• The refracted beam bends according to Snell’s Law:

\[ n_i \sin \theta_i = n_t \sin \theta_t \]
Fresnel Equations (Dielectrics)

- We can use the *Fresnel Equations* to determine how much light is reflected and how much is refracted.

\[
\begin{align*}
    r_{\text{par}} &= \frac{n_t(n \cdot d) + n_i(n \cdot t)}{n_t(n \cdot d) - n_i(n \cdot t)} \\
    f_r &= \frac{1}{2} (r_{\text{par}}^2 + r_{\text{perp}}^2) \\
    r_{\text{perp}} &= \frac{n_i(n \cdot d) + n_t(n \cdot t)}{n_i(n \cdot d) - n_t(n \cdot t)} \\
    f_t &= 1 - f_r
\end{align*}
\]
Fresnel Equations (Metal)

• For a dielectric-metal surface, we can use the *Fresnel Equations for metals*:

\[
\begin{align*}
    r_{\text{par}}^2 &= \frac{(n_t^2 + k_t^2)(\mathbf{n} \cdot \mathbf{d})^2 - 2n_t(\mathbf{n} \cdot \mathbf{d}) + 1}{(n_t^2 + k_t^2)(\mathbf{n} \cdot \mathbf{d})^2 + 2n_t(\mathbf{n} \cdot \mathbf{d}) + 1} \\
    r_{\text{perp}}^2 &= \frac{(n_t^2 + k_t^2) + (\mathbf{n} \cdot \mathbf{d})^2 - 2n_t(\mathbf{n} \cdot \mathbf{d})}{(n_t^2 + k_t^2) + (\mathbf{n} \cdot \mathbf{d})^2 + 2n_t(\mathbf{n} \cdot \mathbf{d})} \\
    f_r &= \frac{1}{2}(r_{\text{par}}^2 + r_{\text{perp}}^2)
\end{align*}
\]
Fresnel Equations

• The Fresnel equations combined with Snell’s Law gives us all we need to know* when we have a beam of light (ray) hitting a perfectly smooth dielectric or metal surface

• What about rays hitting rough surfaces?

*OK, actually, we still haven’t talked about dispersion, which is caused by refractive index changing with the wavelength of the light, but we’ll get to that in a later lecture...
Specular Materials
Specular Materials

• The term *specular* refers to surfaces that show bright highlights
• It isn’t limited to perfectly smooth mirror surfaces however
• Rough metallic surfaces appear shiny, although they don’t act like perfect mirrors
Microgeometry

- Many of the macroscopic optical properties of materials are due to the microscopic geometry of the surface.
- Many materials are not smooth at a small scale; they have lots of little bumps.
- We can think of a surface as being made up of microfacets, whose normals are described by some sort of distribution function relative to the average surface normal.
Cook-Torrance Reflectance Model

• The Cook-Torrance reflection model is based on the assumption that the surface is made up of microfacets - each of which is an ideal Fresnel metal reflector
• It was proposed by Michael Cook and Kenneth Torrance in 1981
• The Oren-Nayar model was inspired by this model
• The Cook-Torrance model was based on an earlier model by Torrance and Sparrow from 1967 that evolved from research in radar reflections
Cook-Torrance Reflection Model

- They use a vector called $v$, which is the *view vector*, a vector pointing towards the viewer (this is generally going to be the negative of the incoming ray direction)
- They also use a vector $L$, which points towards the light ($I$’m using a capital $L$ because the lower case $l$ looks like a $l$)
- They introduce a vector $h$, called the *halfway vector* which lies halfway between $v$ and $L$
- $h$ refers to the normal of a hypothetical microfacet that would reflect the light directly towards the viewer

$$h = \frac{v + L}{|v + L|}$$
Cook-Torrance Model

\[ L_r = L_i \cdot \frac{F \cdot G \cdot D}{\pi (n \cdot L)(n \cdot v)} \]

- **F** = Fresnel term
- **G** = Geometric attenuation term
- **D** = Microfacet distribution function
Fresnel Term

• The Fresnel term $F$ can be the Fresnel equation for metals that we looked at earlier
• There are also various simplifications that have been proposed
Geometric Attenuation

- *Geometric attenuation* refers to the decrease in light reflection due to both shadowing and masking.

\[
G = \min \left( 1, \frac{2(n \cdot h)(n \cdot v)}{(v \cdot h)}, \frac{2(n \cdot h)(n \cdot L)}{(v \cdot h)} \right)
\]
Microfacet Distribution Function

- There have been various functions proposed that describe the distribution of microfacets around the average surface normal.

- **Gaussian:** \( D = ce^{-(\alpha/m)^2} \)

- **Beckmann:** \( D = \frac{1}{m^2 \cos^4 \alpha} e^{-\left(\frac{\tan^2 \alpha}{m^2}\right)} \)

where

\( \alpha = \cos^{-1}(n \cdot h) \)

\( m = \text{root mean square slope of microfacets} \)

\( c = \text{an arbitrary constant (?)} \)
Cook-Torrance Reflection Model
Anisotropic Materials
Isotropic vs. Anisotropic

• Lets say that we place a sample of a material flat on a table in front of us, and we have a light source in the room shining at the table.

• Then, without moving the light or changing our viewing angle, we rotate the material on the table.

• If the reflected color we see remains constant as the material rotates, we call the material \textit{isotropic} (isos=equal/same, tropos=turning/circle).

• If the reflected color changes as the material rotates, we call it \textit{anisotropic} (an=not).
Isotropic Materials

- Many common materials are isotropic due to the overall random distribution of surface microgeometry combined with random distribution of pigment particles in the medium.
- There is no inherent directionality at the microscopic scale which leads to no visible directionality at the macroscopic scale.
Anisotropic Materials

- Some materials do have some sort of inherent directionality at the microscopic level.
- A common example is brushed metals, where the metal surface is roughened along one particular direction.
Anisotropic Materials

• Cloth is another example of an isotropic material, due to the directionality of the threads in the weave
• Satin and velvet are two good examples of complex fabrics
Anisotropic Materials

- Wood and some other natural materials sometimes have an anisotropic appearance due to the directional alignment of cells.
Anisotropic Materials

- Hair and fur are also strongly anisotropic
Anisotropic Materials

• To render an anisotropic material, we need more information about a surface than just the position and normal
• We need some sort of information about the orientation of the material in the plane
• Typically, we use tangent vectors, which are in the plane and provide a reference frame for the material orientation
• We will look at this in more detail in a later lecture, as well as looking at some anisotropic reflection models
Other Material Properties
Rough Dielectrics

- We can derive models for rough dielectric surfaces, similar in concept to the Cook-Torrance model for rough metals.
Retroreflection

- **Retroreflection** refers to specular reflection back towards the light source
- It is not the same as the opposition effect, but it is another type of *backscatter* phenomenon
Iridescence

- Iridescence refers to the property of some materials changing color depending on the view direction.
- This can be caused by different phenomena such as constructive and destructive interference in thin films like bubbles, oil on water, or surface coatings.
Diffraction

- Diffraction of light on bumps near the wavelength of light can also cause iridescent effects.
Translucency

- Translucency and subsurface scattering are other common properties that can be captured
- We’ll look at these some more in a later lecture
Material Rendering
Materials

• Because there are a wide range of materials, it is nice to allow a flexible definition of materials, instead of just having one single material model
• This is a perfect place to take advantage of derived classes and virtual functions in C++
• We can create a base class Material and derive various specific material types from that
class Material {
public:
  Material();
  virtual void ComputeReflectance(Color &col, glm::vec3 &in, glm::vec3 &out, Intersection &hit)=0;
};
Colors

- The subject of color is actually quite complex and we will discuss it in a lot more detail in a later lecture.
- For now, I just want to mention that it is important to have a Color class that is used in all places where the renderer does operations on colors.
- It is tempting to just use a vec3, since we think of colors as having 3 components (red, green, blue), and we do a lot of similar operations as vectors (addition, scaling, etc.).
- However, as we will see later, colors really should be treated as spectral distributions across all visible wavelengths (not just 3!).
- If we use a Color class and just make it simple RGB for now, then we can later swap in a more sophisticated color class that uses the same interface (Add(), Scale()...) and upgrade to a more realistic color model with minimal effort.
class Color {
    public:
        Color();

        void Add(const Color c);
        void AddScaled(const Color c, float s);
        void Scale(float s);
        void Multiply(const Color c);

        void Exponent(); // Computes $e^c$
        void Gamma(float exp); // Computes pow(c,exp);

        int ToInt(); // Converts to 24 bit RGB
        void FromInt(int c); // Converts from 24 bit RGB
};
Shadows
Shadows

• Adding shadows to the ray tracer we have so far is very straightforward
• Instead of just adding the contribution from each light, we first want to check if the light in question is blocked by some other surface
• To do this, we can shoot a shadow ray towards the light source
• We want to make sure we don’t intersect with things past the light source, so we set the Intersection::HitDistance to be the distance to the light source before we call Scene::Intersect(ray, hit)
• The Light::Illuminate() routine sets a vec3 &toLight, which is the ray direction to the light source that can be used to generate a shadow ray
• Light::Illuminate() also sets the vec3 &ltPos, which is the position of the light source. We can use this to compute the initial HitDistance value
• Light::Illuminate() also returns the intensity of the light at the position, so don’t bother testing for shadows if the intensity is 0 or if the angle between the hit.Normal and toLight vector is greater than 90 degrees (this happens if hit.Normal.Dot(toLight)<0)
Shadow Rays
Shadow Ray Offset

- Note that due to floating point precision limitations the shadow ray could actually intersect the original surface itself. This could happen if the shadow ray origin actually ends up being slightly below the actual surface (due to round-off).
- A common approach to fixing this is to use the original position and just modify the ray intersection routines to reject any intersection where \( t < 0.001 \) (or some other small epsilon value).
- Also, the HitDistance to the light should be shortened by a similar small distance to avoid hitting the light surface (if there is one).
Shadow Ray Offset
Shadows

- This is project 1 with shadows added:
Area Lights
Soft Shadows

- Both point lights and directional lights generate very sharp shadows
- In the real world, we are used to seeing softer edges on shadows
- This is because real lights are not points (or perfectly unidirectional)
- Real lights have some finite area
- In computer graphics, we call these *area lights*
- Larger area lights cast very soft shadows, while smaller area lights cast sharper edged shadows
Umbra & Penumbra

- The fully shadowed area is called the *umbra*
- The partially shadowed area on the edge is called the *penumbra*
- During a total eclipse of the sun, you are in the umbra cast by the moon
- During a partial eclipse, you are in the penumbra
Soft Shadows

- Umbra
- Penumbra

Area light source

Occluding object

Ground

Penumbra, Umbra, Penumbra
Very Small Area Light
Small Area Light
Medium Area Light
Large Area Light
Ray Tracing Soft Shadows

- In order to render with soft shadows, we can modify our shadow casting routine.
- For non-area lights, we test a single shadow ray from the surface point to the light source.
- For area lights, we need to test a bunch of rays from the surface point to points scattered across the area of the light, and then average the results.
Random Sampling

• In order to generate a bunch of rays to the area light, we need to select several random points on the light
• Assuming that our area lights are modeled as triangles, this would mean that we have to generate a random point on a triangle
Barycentric Coordinates

- Recall our discussion about barycentric coordinates $\alpha$ and $\beta$ of a triangle, where:

$$q = a + \alpha(b-a) + \beta(c-a)$$

- $0 < \alpha < 1$
- $0 < \beta < 1$
- $\alpha + \beta < 1$
Triangle Sampling

• We need to generate random barycentric coordinates $\alpha$ and $\beta$

• Let’s say we have a random number generator that generates random numbers equally spaced from 0...1

• As we are sampling a two-dimensional area, we will need two random numbers

• However, we can’t just pick random values for $\alpha$ and $\beta$
Triangle Sampling

\[ u = \text{random number from 0...1} \]
\[ v = \text{random number from 0...1} \]

\[ \alpha = \sqrt{u} \times v \]
\[ \beta = 1 - \sqrt{u} \]

• We will take a closer look at this and some similar examples in a later lecture
Triangle Sampling

• Let’s say that we are going to shoot 25 shadow rays towards our area light.
• We choose 25 random points on the triangle and assume that each of them accounts for 1/25 of the area of the light.
• If the light source intensity is defined in terms of intensity/meter$^2$ so we’ll need to know the total area of the triangle to compute the intensity of one sample.
• Also, as we are assuming that the area light is a flat surface, the intensity is going to decrease as the normal of the light turns away from us.
• It will decrease by the cosine of the angle between the ray and the light’s normal.
• It will also decrease by the inverse square law.
How Many Samples?

• How many shadow rays should we shoot towards our area light?
• This is difficult but important question
• Too few samples will generate a noisy result
• Too many samples will take too long
1 Shadow Ray / Pixel
10 Shadow Rays / Pixel
100 Shadow Rays / Pixel
1000 Shadow Rays / Pixel
How Many Samples?

- Unfortunately, we can see that it may require quite a large number of samples to generate a clean image.
- A reasonable heuristic would be to generate a number of samples proportional to the size of the light from the point of view of the sample.
- We could also scale that by the intensity of the light, so darker lights would use fewer samples (because the variance/noise will increase with brightness).
- In addition, we could scale this by the albedo of the surface material (for the same reason).
- This gives us a reasonable guideline, but it’s not perfect.
Solid Angles

• How do we measure the size of a light source from some particular point of view?
• We can look at the solid angle that the area light covers.
• An angle represents the proportion of a circle (times $2\pi$), and a solid angle represents the proportion of a sphere (times $4\pi$).
• Solid angles are measured in steradians.
Solid Angle of a Triangle
Solid Angle of a Triangle

• To compute the solid angle of a triangle made up of points \( \mathbf{a}, \mathbf{b}, \) and \( \mathbf{c} \) (from the point of view of the origin):

\[
\Omega = 2 \cdot \arctan \left( \frac{\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})}{abc + (\mathbf{a} \cdot \mathbf{b})c + (\mathbf{a} \cdot \mathbf{c})b + (\mathbf{b} \cdot \mathbf{c})a} \right)
\]

where \( a = |\mathbf{a}| \), etc.

• Also note that the sign will get flipped if the triangle is facing away from the origin

• This formula comes from “The Solid Angle of a Plane Triangle” by A. Van Oosterom and J. Strackee
Many Light Sources

- Some scenes contain *many* lights
- An example would be a city scene at night
- Can we make some optimizations for situations like this?
Many Lights

• One thing we can do is to assume that each light has a finite range
• For example, we can find the distance where intensity drops below some threshold
• A reasonable threshold is $0.5/255$, as this would mean that the light’s contribution to the final pixel brightness is below the lowest intensity we can display (assuming a 24-bit display)
• We can also build spatial data structures around the volumes associated with the lights, so we can quickly query which lights affect a particular point
• Another option is to randomly select some of the lights weighted by their intensity at the point of interest