Path Tracing

Steve Rotenberg
CSE168: Rendering Algorithms
UCSD, Spring 2017
Irradiance
Circumference of a Circle

• The circumference of a circle of radius $r$ is $2\pi r$
• In fact, a *radian* is defined as the angle you get on an arc of length $r$
• We can represent the circumference of a circle as an integral

$$circ = \int_{0}^{2\pi} r \, d\varphi$$

• This is essentially computing the length of the circumference as an infinite sum of infinitesimal segments of length $rd\varphi$, over the range from $\varphi=0$ to $2\pi$
Area of a Hemisphere

• We can compute the area of a hemisphere by integrating ‘rings’ ranging over a second angle $\theta$, which ranges from 0 (at the north pole) to $\pi/2$ (at the equator)

• The area of a ring is the circumference of the ring times the width $rd\theta$

• The circumference is going to be scaled by $\sin \theta$ as the rings are smaller towards the top of the hemisphere

\[
\text{area} = \int_0^{\pi/2} \int_0^{2\pi} \sin \theta \ r^2 \ d\phi \ d\theta = 2\pi r^2
\]
Hemisphere Integral

\[ \text{area} = \int_0^{\pi/2} \int_0^{2\pi} \sin \theta \ r^2 \, d\phi \, d\theta \]

- We are effectively computing an infinite summation of tiny rectangular area elements over a two dimensional domain. Each element has dimensions:

\[ \sin \theta \ r \, d\phi \times r \, d\theta \]
Irradiance

• Now, let’s assume that we are interested in computing the total amount of light arriving at some point on a surface
• We are essentially integrating over all possible directions in the hemisphere above the surface (assuming it’s opaque and we’re not receiving translucent light from below the surface)
• We are integrating the light intensity (or radiance) coming in from every direction
• The total incoming radiance over all directions in the hemisphere is called the irradiance
• We will say that the incoming radiance from any one direction is a function of the direction: $L_i(\theta, \varphi)$
• The incoming radiance from each direction is scaled by $\cos \theta$ to account for the fact that light coming in from a horizontal direction is more spread out than light coming from directly above
Irradiance

- We can express this irradiance as an integral over the hemisphere of incoming light directions:

\[
\text{irrad} = \int_0^{\pi/2} \int_0^{2\pi} L_i(\theta, \varphi) \cos \theta \sin \theta \, d\varphi \, d\theta
\]
Irradiance

\[ \text{irrad} = \int_{0}^{\pi/2} \int_{0}^{2\pi} L_i(\theta, \varphi) \cos \theta \sin \theta r^2 d\varphi \ d\theta \]

- We can simplify the notation a bit. Instead of writing it as a double integral over two angles, we can write it as a single integral over the domain \( \Omega \) of all possible directions \( \omega \)

\[ \text{irrad} = \int_{\Omega} L_i(\omega) \cos \theta \ d\omega \]
Irradiance

\[ \text{irrad} = \int_{\Omega} L_i(\omega) \cos \theta \, d\omega \]

- \( \omega \) is a unit vector for some particular direction. To be consistent with convention, these vectors will always point away from the surface regardless of whether they represent incoming or outgoing light.
- \( \Omega \) is the hemisphere domain, which can be thought of as a (infinite) set of all possible directions in a hemisphere.
- \( d\omega \) is an infinitesimal area element (around direction \( \omega \)) on the surface of the hemisphere. We don’t really care about it’s exact shape, just that it represents an infinitesimal portion of the total surface area.
- We know that the surface area of a unit hemisphere is \( 2\pi \), so by definition:

\[ \int_{\Omega} d\omega = 2\pi \]
Irradiance

\[ \text{irrad} = \int_\Omega L_i(\boldsymbol{\omega}) \cos \theta \, d\boldsymbol{\omega} \]

- If we write the integral like this, it doesn’t look so bad, but the notation hides a lot of details.
- We were able to compute the area of a hemisphere by solving the integrals and computing an analytical result \((2\pi r^2)\).
- Can we compute an analytical solution to the irradiance integral?
  - Doubtful, except in very simple situations.
- Consider that \(L_i(\boldsymbol{\omega})\) represents the incoming light from direction \(\boldsymbol{\omega}\) and this can vary significantly based on scene geometry, light properties, material properties, etc.
- Realistically, we have to accept that we can’t possibly calculate this analytically in any situation of interest.
- We therefore have to resort to numerical methods.
Monte Carlo Integration

\[ \text{irrad} = \int_{\Omega} L_i(\omega) \cos \theta \, d\omega \]

- We talked about using Monte-Carlo integration to estimate the area of a circle.
- We also applied the same approach to estimating the shadowing from an area light source, and estimating the color of a rectangular pixel.
- Let’s say that \( L_i(\omega) \) is given to us as some pre-defined function - perhaps in the form of an environment map.
- Instead of integrating over an infinite number of directions, we can estimate by summing over a finite number of randomly selected directions.

\[ \text{irrad} \approx \frac{2\pi}{N} \sum_{k=1}^{N} L_i(\omega_k) \cos \theta_k \]
Monte-Carlo Integration

\[ \text{irrad} \approx \frac{2\pi}{N} \sum_{k=1}^{N} L_i(\omega_k) \cos \theta_k \]

- We need to generate N random directions in a hemisphere. We can use the techniques from the previous lecture (i.e., generate two random numbers from 0 to 1 and then apply the hemispherical mapping formula).
- For each sample, we can then evaluate the incoming light \( L_i(\omega_k) \) from the environment map, multiply it by \( \cos \theta_k \) and then add it to a running total, which finally gets scaled by \( \frac{2\pi}{N} \).
- We accept that \( L_i(\omega_k) \) will vary unpredictably, so some samples will end up contributing a lot and some will contribute less.
- The \( \cos \theta_k \) term however, is not unpredictable, yet it will still cause some samples to contribute a lot and some to contribute almost nothing.
- We would like to eliminate this by generating a cosine-weighted distribution (instead of using a uniform distribution that gets scaled by the cosine).
Cosine-Weighted Hemisphere

• We can accomplish this by simply changing from the hemispherical mapping formula to the cosine weighted hemisphere formula.
• We also have to adjust the total scale factor because the area of a hemisphere is $2\pi$, but the $\cos \theta$ scaled area of a hemisphere is $\pi$.
• This gives us a better way to estimate the irradiance:

\[
irrad \approx \frac{\pi}{N} \sum_{k=1}^{N} L_i(\omega_k)
\]

• (Note that this assumes that $\omega_k$ are randomly distributed over the cosine weighted hemisphere)
Hemispherical Distributions

\[ u = 2\pi s \]
\[ v = \sqrt{1 - t^2} \]

\[ \omega_x = v \cdot \cos(u) \]
\[ \omega_y = t \]
\[ \omega_z = v \cdot \sin(u) \]

Note: \( s \) and \( t \) are random numbers between 0 and 1
BRDFs
BRDFs

• A bidirectional reflectance distribution function is a four dimensional function that describes how light is reflected off a surface.

• It relates light coming from the incident direction $\omega_i$ to light reflected in some outward direction $\omega_r$

$$f_r(\omega_i, \omega_r)$$

• Note that because the two vectors are unit length direction vectors, they really only have two independent variables each (the two polar angles describing the direction). Sometimes, BRDFs are written like this:

$$f_r(\theta_i, \varphi_i, \theta_r, \varphi_r)$$
BRDFs

Side view, where $\mathbf{n}$ is surface normal

Top view, where $\mathbf{s}$ and $\mathbf{t}$ are surface tangents
BRDFs

• We can think of the BRDF as returning a color, so it is really more of a vector function, but sometimes, it’s written as a scalar function of wavelength $\lambda$

$$f_r(\lambda, \omega_i, \omega_r)$$

• Also, as surface properties vary based on position $x$, sometimes, it is written as:

$$f_r(x, \omega_i, \omega_r)$$

• Also, some authors prefer to use the symbol $\rho$ for the BRDF function instead of $f_r$
BRDF Properties

• There are two important properties that are required for BRDFs to be physically valid:
  – Conservation of Energy
  – Reciprocity
Conservation of Energy

- Conservation of energy is a fundamental principle in physics.
- It says that energy is never created or destroyed - only changed from one form to another.
- With respect to light scattering from materials, it tells us that the total light reflected off of the material must be less than the light incident on the material.

\[ \forall \omega_i, \int_{\Omega} f_r(\omega_i, \omega_r) \cos \theta_r d\omega_r \leq 1 \]
Reciprocity

• Hermann von Helmholtz (1821-1894) was a German physicist who made many contributions to optics, theory of vision, color, thermodynamics, electrodynamics and more

• In a famous treatise on physiological optics, published in 1856, he formulated the *Helmholtz reciprocity principle*

• The principle states that if we follow a beam of light on any path through an optical system, the loss of intensity is the same as for a beam traveling in the reverse direction

\[
f_r(\omega_i, \omega_r) = f_r(\omega_r, \omega_i)
\]
Isotropic & Anisotropic

- A BRDF can be either isotropic or anisotropic
- An isotropic BRDF doesn’t change with respect to surface rotation about the normal
- Brushed metals and animal fur are good examples of anisotropic materials that do change with respect to orientation
- For an isotropic BRDF, there is no fixed frame of reference for the $\varphi$ angles. The absolute angles are irrelevant, but the difference between the two is still important. This effectively reduces the isotropic BRDF to a 3 dimensional function:

$$f_r(\theta_i, \theta_r, |\varphi_i - \varphi_r|)$$
Lambert Diffuse BRDF

• The Lambert diffuse BRDF can be written as:

\[ f_r(\omega_i, \omega_r) = \frac{c_{\text{diff}}}{\pi} \]

where \( c_{\text{diff}} \) is the diffuse color

• Note that the BRDF itself is constant. The cosine term usually associated with Lambert reflection comes from the cosine term in the radiance equation itself
Radiance

- *Radiance* is a measure of the quantity of light radiation reflected (and/or emitted) from a surface within a given *solid angle* in a specified direction
- It is typically measured in units of $W/(sr\cdot m^2)$ or Watts per steradian meter squared
- It is the property we usually associate as ‘light intensity’ and is what we measure when we are computing the color to display
- Technically, radiance refers to a single wavelength, and *spectral radiance* refers to the radiance across a spectrum
- Spectral radiance is what we associate as the perceived color (including the intensity), and is what we use to determine the pixel color
Reflected Radiance

• Let’s say that $L_i(\omega_i)$ is a function that measures the incident radiance from direction $\omega_i$.

• If light is shining on the surface from some particular direction $\omega_i$, then the light we see coming off in direction $\omega_r$ is

$$f_r(\omega_i, \omega_r)L_i(\omega_i)\cos \theta_i$$

• The cosine term is due to the fact that the incident light on the surface is spread out by $1/\cos \theta$, reducing the amount of light that can be reflected.

• Note that $\cos \theta_i = \omega_i \cdot n$. 
Radiance Equation

• The radiance equation computes the reflected radiance $L_r$ in direction $\omega_r$.
• It integrates over all of the possible incident light directions $\omega_i$ in a hemisphere $\Omega$.
• The total reflected radiance is the integral (sum) of all of the incident radiances $L_i$ from different directions, times the BRDF function for that direction (and times the cosine of incident angle):

$$L_r(\omega_r) = \int_{\Omega} f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \theta_i d\omega_i$$
BRDF Variations

• Technically, BRDFs are limited to describing light reflecting off of a surface
• If we want to talk about light transmitted (refracted) through the surface, we need the bidirectional transmission distribution function or BTDF
• Some authors combine these into a single scattering function or BSDF
• The BSDF is the most general as it includes reflection and scattering
• There is another term called bidirectional subsurface scattering distribution function or BSSRDF, which is more complex and is used for modeling translucency
Global Illumination
Global Illumination

• For photoreal rendering, we are interested in simulating *global illumination*

• Global illumination refers to the modeling of all of the light interaction within a scene, including the effects of multiple diffuse and specular bounces

• Light enters the scene from the light sources, bounces around, and reaches a global equilibrium

• Global illumination is described by the *rendering equation*
Rendering Equation

\[ L_{\text{out}} = L_{\text{emitted}} + L_{\text{reflected}} \]

\[ L(\omega_r) = L_e(\omega_r) + L_r(\omega_r) \]

\[ L(\omega_r) = L_e(\omega_r) + \int_{\Omega} f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \theta_i d\omega_i \]
Integral Equations

• A equation that contains a function and an integral of that function is called an integral equation
• As with differential equations, there are many different categories of these and many different solution techniques
• The rendering equation is an inhomogeneous, linear, Fredholm integral of the second kind
• The general form of this type of equation is:

\[ x(t) = g(t) + \lambda \int_{a}^{b} k(t, u)x(u)du \]
• Equations of this form were studied long before the rendering equation was identified
Recursive Integrals

• The reason the rendering equation is challenging to solve is that in order to evaluate the outgoing radiance $L$ at a point, we need to know the incoming radiance $L_i$ from every other point.

• This leads to an infinite recursion.
Neumann Expansion

\[ L(\omega_r) = L_e(\omega_r) + \int_{\Omega} f_r(\omega_i, \omega_r)L_i(\omega_i)\cos \theta_i d\omega_i \]

\[ L = L_e + \int f_r L \]

\[ L = L_e + \int f_r L_e + \int f_r \int f_r L \]

\[ L = L_e + \int f_r L_e + \int f_r \int f_r L_e + \int f_r \int f_r \int f_r L \]

\[ L = \ldots \]

\[ L = L_e + TL_e + T^2L_e + T^3L_e \ldots = \sum_{m=0}^{\infty} T^m L_e \]
Light Paths

• In order to classify and compare rendering algorithms, it is convenient to introduce a notation to describe light paths.
• In the real world, light starts at a light source and eventually ends up in our eye, after possibly several bounces.
• We make a distinction between diffuse bounces and specular bounces, as they bring different challenges and sometimes require different approaches to handle correctly.
• A light path is described using the following notation:
  – L Light source
  – D Diffuse reflection
  – S Specular reflection
  – E Eye
• So the direct lighting on a diffuse surface is LDE.
• If we are seeing a directly lit diffuse surface through a mirror, we would be seeing the LDSE paths.
Light Paths

- We can also use some operators in our notation:
  - $|$ or
  - $*$ 0 or more
  - $+$ 1 or more
  - $()$ parenthesis

- For example, $L(S|D)E$ refers to a path that starts at the light, hits either a diffuse or specular surface, and then hits the eye.

- $LS+DE$ refers to paths that have at least one specular bounce before hitting a diffuse surface and then hitting the eye.

- Global illumination models all light paths (in theory), so would include $L(S|D)*E$ paths.
Light Paths
Direct Illumination
Direct + Indirect Illumination
Indirect Illumination
Ray Based Algorithms

- Ray-based rendering algorithms are all evolved from the original ray tracing algorithm and make use of the concept of ray intersection.
- They can all be built off of the same foundation of ray intersection algorithms, spatial data structures, and BRDF models.
- It is even possible to combine all of these techniques and use each for its strengths.
Classical Ray Tracing

• *Classic ray tracing* traces one or more primary rays per pixel from the camera
• If a ray hits a diffuse surface, then additional shadow rays are traced to the lights and the pixel is shaded
• If a ray hits a specular surface, it reflects and/or refracts recursively until a diffuse surface is hit
• Traces LDS*E paths
Ray Tracing
Distribution Ray Tracing

- *Distribution ray tracing* added the concept of averaging over a distribution of rays to render softer effects like glossy reflections, soft shadows, camera focus, and motion blur.
- The camera tests multiple rays per pixel, distributed across the lens and in time.
- Reflections will spawn multiple additional rays and diffuse surfaces will test multiple shadow rays per area light.
- This approach can lead to an exponential growth in the number of rays per pixel.
- It traces the same LDS*E paths as classic ray tracing, but supports a wider range of material types and visual effects.
Distribution Ray Tracing
Path Tracing

• *Path tracing* traces one or more primary rays per pixel
• When a ray hits a surface, it traces one or more shadow rays towards light sources, and also spawns off one additional reflection ray
• The reflected rays form a path, where each node of the path connects to the light source
• Paths are randomly generated and their length is determined through probabilistic means
• Path tracing is capable of tracing all types of light paths \( L(S|D)^*E \), but can suffer from excessive noise in some situations
• It is generally good for LD*S*E paths, but is bad at LS*(S|D)*E paths (i.e., paths with caustics) and fails for any LS(S|D)*E paths if the light source is a singularity (i.e., a point or single direction)
Path Tracing
Path Tracing
Path Tracing

- There are different variants on path tracing, but at its simplest, it works like this:
- For each pixel, we trace multiple paths starting from the camera
- The first ray in the path is the same as in standard ray tracing, and because we know we’re going to shoot multiple paths, we might as well do pixel antialiasing
- When that ray hits a surface, we first compute an estimate of the direct illumination by tracing a single shadow ray to a single light source. The light source is randomly selected, weighted based on the potential (unshadowed) light hitting the surface. The shadow ray may additionally be randomly distributed over the light source in case of an area light
- We then compute an estimate of the indirect illumination by tracing a single ray to estimate the reflected radiance back in the incoming ray direction. This ray is generated in a random cosine-weighted direction around the normal of the surface (at least in the case of Lambert diffuse surfaces. For specular surfaces, we can do better).
- This reflected ray then shoots off into the scene and then hits a new surface. We test a single shadow ray, and then generated a new reflected ray, etc.
- The reflected rays form a random path through the scene
- We eventually terminate the path either when a preset number of bounces has occurred (or according to a better process to be described later)
Path Tracing

• How and why does this work?
• We’re essentially doing the same thing as ray tracing or distribution ray tracing but we’re not generating an exponential explosion of rays
• Essentially, we accept that we will be averaging out lots (100’s or 1000’s) of paths, so we can allow for each path to be a high variance estimate
• This is how we get away with estimating the light coming in from an entire hemisphere by just picking a single direction
• Consider that the light is dimmer after each bounce, so the earlier rays in the path contribute more to the total pixel color
• Also, consider distribution ray tracing. If we generate 4 primary rays for each pixel, then each of those rays will contribute 25% to the final value. If each bounce generates 4x new rays, then after a few bounces, we’re doing 100’s of rays that are each worth a tiny amount
• With path tracing, we trace the same number of rays per bounce (2), so each bounce has the same cost (rather than distant bounces costing a ton more, while contributing the least)
10 Paths / Pixel
10000 Paths / Pixel
Path Length

• Some lighting situations do not involve a lot of bounces, such as outdoor lighting, and may look fine with only two or three bounces
• Indoor scenes with bright lights and bright walls will have a lot of indirect light and will require more bounces (maybe 5 or more)
• Scenes with shiny objects and dielectrics may require many bounces to render correctly (10+)
• It is always possible to construct a situation where we need even more bounces (such as a hall of mirrors)
• How do we determine how long our paths should be without resorting to just picking some number?
Russian Roulette

- If we limit the number of bounces to a fixed value, then we are introducing bias (consistent error) to the image as we will underestimate the total brightness.
- Russian roulette is a strategy that randomly determines whether or not to terminate a path at each bounce, based on the total contribution of the path.
- Above a certain brightness threshold, all bounces are accepted. Below that threshold, we assign a probability of acceptance.
- If, for example, we determine that there is a $1/N$ chance of accepting a particular bounce, then we scale its contribution by $N$ if it is accepted.
Path Tracing

- Path tracing is well suited for rendering bounced light between diffuse surfaces.
- It can also handle specular surfaces and arbitrary BRDFs (provided they are physically valid).
- It can be extended to handle volumetric scattering (fog), translucency, and basically every relevant optical effect.
- It does however tend to require lots of paths to converge, and more specular surfaces and bounced light situations make the problem worse.
- It should eventually converge on the correct answer, but may take VERY long in certain problem situations.
Strengths of Path Tracing

• Well suited for primarily diffuse scenes with area lighting
• Relatively easy to implement
• Flexible: easy foundation to add new features
• Reference to compare other algorithms
• Becomes more and more appealing as GPUs get faster
Limitations of Path Tracing

• Poorly suited for highly specular scenes and caustics
• Not well suited for highly indirect light
• Can be slow and take many paths to converge
Diffuse vs. Specular Surfaces

• So far, the approach we discussed is suited best for diffuse surfaces
• This is because we are using a cosine-weighted distribution to fit the expected incoming radiance
• This works great for diffuse surfaces because the light reflected off is based on the incoming irradiance directly
• For specular surfaces, the BRDF is more complex, and we can generate sampling distributions that take the material BRDF into account
• For example, for a shiny surface, this would generate rays distributed around the reflection direction rather than over the entire hemisphere
• We will talk about this more in the next lecture
Project 3

- Project 3 will be to add path tracing and antialiasing to your existing renderer.
- The path tracer must support different material types, and in particular must handle diffuse surfaces and Fresnel metals (we will add more complex materials in project 4).
- Details of project 3 will be posted on the web page.
• I suggest adding a new virtual function to the Material base class:

```cpp
virtual void GenerateSample(Color &col, const Vector3 &in, Vector3 &out, const Intersection &hit);
```

• This generates the sample ray direction based on the material BRDF. For a diffuse material, this would use the cosine-weighted weighted hemisphere
Recursive Reflections

• I had previously suggested creating a RayTrace class to handle all shading computations

• The camera should only need to call RayTrace::TraceRay(ray, hit) to get a fully shaded result (the hit.Shade contains the color)

• This allows the camera to have a RenderPixel() function that handles antialiasing and everything up to shooting the actual primary rays

• The RayTrace class can handle the ray intersection, all shading & shadow ray testing, and all recursive ray reflections