10 Toward Photo-Realistic Ray Tracing

Up to this point, we are able to convincingly simulate a variety of materials and illumination effects using clever approximations. However, there are important limitations to our approach. First of all, our raytracer can only handle simple materials whose properties are well represented using the Phong model. Most real- world materials behave in much more complex ways and show visible appearance characteristics that can not be accurately rendered with our simple model. Secondly, the way our simple ray tracer handles light is too simple to accurately replicate most of the complex visual effects produced by light traveling through a scene. Our light propagation model accounts for very limited reflection and refraction effects (using only a single ray for each component), and does not account at all for secondary (indirect) illumination. Its ability to handle complex illumination effects such as caustics is also limited to what can be approximated reasonably well with photon mapping. Illumination effects such as self-occlusion, multi-bounce secondary illumination, and intervening media (such as fog or smoke) are not handled at all, and light dispersion effects such as those produced by prisms are non-existent. Finally, the illumination model is not physically consistent, meaning that the final rendered scene can easily result in objects that are un-realistically bright given the light sources in the scene.

This scene was rendered using a realistic light transport model. The different materials and the liquids contained inside some of the objects interact with light in complex ways that can not be accurately portrayed by our simple ray tracer based on the Phong model, even with the use of photon mapping. [Image by: Krivanek et al., 'Unifying Points, Beams, and Paths in Volumetric Light Transport Simulation', SIGGRAPH 2014]

So, if our goal is to create an accurate rendering of a scene; one that captures the subtle illumination effects we see in real world scenes, and that can handle realistic materials and light sources,

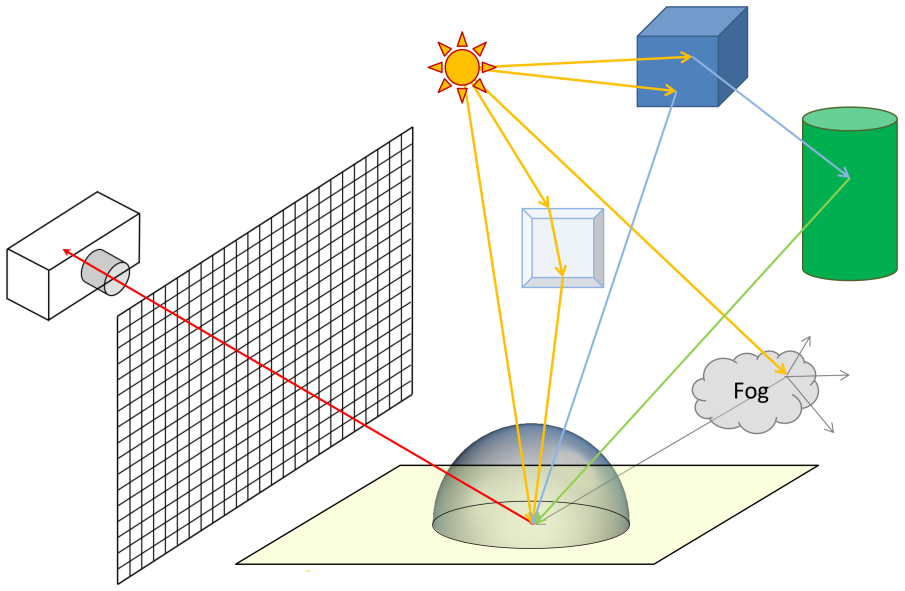
we need to build into our raytracer a physically consistent model for materials, and a physically accurate model for light transport. This chapter and the next will deal with two approaches for building a realistic, physically accurate ray tracing engine able to render accurate, realistic images of scenes with rich geometry, materials that interact with light in complex ways, and physically realistic light transport.

10.1 Light Transport at a Point: The Rendering Equation

The central issue in ray tracing is how to estimate the amount of light leaving a scene along a ray in the direction of the camera passing through a particular pixel. In our simple ray tracer, this amounts to determining the visible surface closest to the camera along the ray, and then using reasonable approximations to approximate the *radiance* from that surface back along the direction of the ray, toward the camera. Radiance is a measure of the amount of light emitted, received, transmitted, or reflected by a surface, by unit area, and unit solid angle.

> *Note:*
> For a detailed description of how *radiance* and the closely related *irradiance* are computed, please refer to the Appendix section on Radiometry.

The radiance from a given surface depends on the amount of light incident on that surface from the rest of the scene, plus any energy emitted by the surface itself. This complicates matters because light can arrive at a surface from any direction - if we assume for the moment an opaque material, this means that light can arrive at a given surface point from any direction over a hemisphere centered at that point.



To determine the radiance from point $\bar{p}$ back toward the camera along a ray, we have to account for light arriving at $\bar{p}$ along every possible direction over a hemisphere centered at the point. This accounts for direct illumination from light sources, as well as secondary illumination from light bounced (possibly multiple times) from other objects in the scene, reflected or refracted, or scattered by environmental factors such as fog, liquids, or smoke.

To understand what is involved in estimating radiance along the ray from the surface point to the camera, let's consider what the contribution to this radiance is due to light arriving at a *single* direction at point $\bar{p}$.

$$L_e(\vec{d_e}, \vec{d_i}(\theta, \phi)) = \rho(\vec{d_e}, \vec{d_i}(\theta, \phi)) L_i(\vec{d_i})(\vec{n} \cdot \vec{d_i}) \tag{1}$$

Here L_e is the *radiance* emitted from point $\bar{p}$ in the direction $\vec{d_e}$ of a ray leaving point $\bar{p}$ due to incoming light from direction $\vec{d_i}$. The incoming light direction is parameterized using spherical coordinates over a unit sphere, hence the angles θ and ϕ. L_i is the radiance arriving at $\bar{p}$ along $\vec{d_i}$, the term $\vec{n} \cdot \vec{d_i}$ is the foreshortening term, identical to what we use for the diffuse component of the Phong model, and the term $\rho(\vec{d_e}, \vec{d_i}(\theta, \phi))$ gives the proportion of the incoming radiance that is emitted reflected toward direction $\vec{d_e}$.
We can immediately see an analogy between the equation above, and the diffuse component of the Phong model $I_d = r_d \; I_d \; max(0, \vec{n} \cdot \vec{d_i})$. However, note that several components of the Phong model's diffuse term are constants, while in the radiance equation above they are a function of either the incoming, outgoing, or both directions of light transfer. This is essential for accurately simulating light transport.
The function $\rho(\vec{d_e}, \vec{d_i})$ is particularly important. It is called the *Bidirectional Reflectance Distribution Function* or *BRDF*. It characterizes the way a surface material interacts with light by specifying for each incoming direction, and every outgoing direction, how much of the incident light is reflected toward the outgoing direction. The BRDF of a material can be measured experimentally, by illuminating the material from many different directions, and recording the amount of light reflected toward multiple outgoing directions. BRDF tables are available for common materials, and it is not uncommon to find parametric approximations to BRDFs based on experimental data.
Now that we know how to estimate radiance due to a single incident light direction, we can proceed to estimate the total radiance along direction $\vec{d_e}$:

$$L_e(\vec{d_e}) = \int_\Omega \rho(\vec{d_e}, \vec{d_i}(\phi, \theta)) \; L_i(\vec{d_i}(\phi, \theta)) \; (\vec{n} \cdot \vec{d_i}) \; d\omega \tag{2}$$

The equation above is commonly referred to as the *rendering equation*. The integral is over the hemisphere Ω centered at $\bar{p}$, and accumulates the contribution to the total radiance along $\vec{d_e}$ from

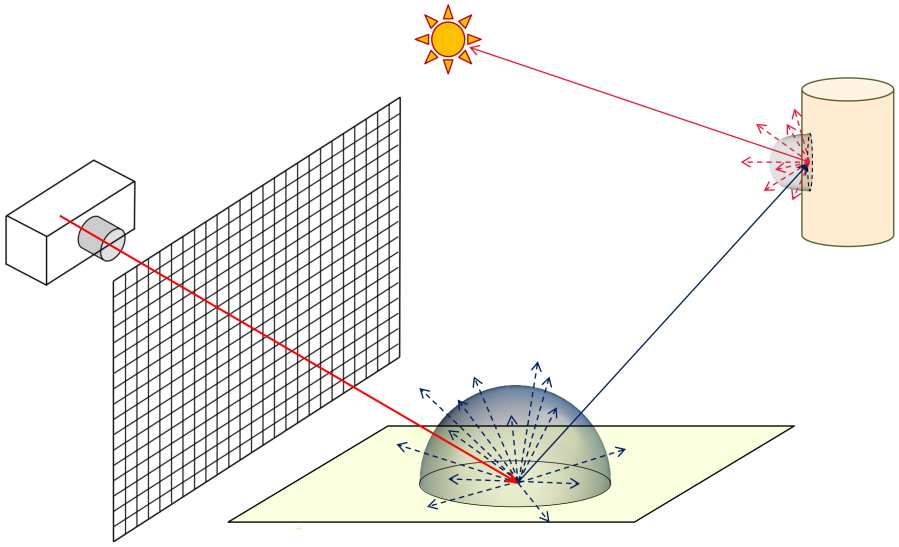
each possible direction over this hemisphere. This is the quantity we need to estimate in order to determine the amount of light traveling along a ray from any point in the scene. The integral itself can not be computed exactly, creating a photo-realistic render becomes a matter of developing algorithms to approximate the value of the rendering equation as closely as possible, while doing so in a computationally manageable way.
In what follows, we will study two common methods for approximating the radiance given by the rendering equation. They both rely on sampling, but, critically, differ in how the sampling is carried out and this has important implications in terms of computational efficiency and rendering accuracy.

Note:
The text above ignores the fact that most materials behave differently for light at different wavelenghts. The rendering equation is often parameterized also in terms of wavelength λ. The raytracer can then accurately simulate light transport by sampling over λ in addition to the sampling required to approximate the rendering equation. More often, radiance is represented as an $[R, G, B]$ triplet, and the BRDF for a specific material gives reflectance amounts for each colour component.

Note:
An additional but important detail is that we now need to account for the area of pixels on the image plane. Light will be received over the entire area of the pixel, not just a single ray through the pixel's center. This involves estimating the integral of (appropriately scaled) radiance over the area of the pixel. A derivation for this can be found in the appendix. The rendering algorithms described above also approximate this integral via carefully constructed sampling.

10.2 Distribution Ray Tracing

Distribution Ray Tracing (DRT) attempts to densely sample light bouncing off surfaces in the scene. The process is in fact fairly simple. As with our simple ray tracer, we start by casting a ray through a pixel and finding the closest intersection with some object in the scene.



Distribution Ray Tracing process. At every intersection point, the ray tracer must cast and trace rays that cover the hemisphere centered at that surface point. The number of rays cast, and the direction of the rays are usually chosen based on the BRDF of the material. A large number of rays need to be cast to accurately capture illumination at any given surface location.

At the intersection point, the distribution ray tracer approximates the rendering equation by casting and tracing rays over a hemisphere aligned with the surface normal. The number of rays cast should be sufficient to provide a good approximation to the integral of the rendering equation. The ray directions themselves are chosen by random sampling, typically influenced by the BRDF of the material the surface is made of. Of course, each of these cast rays may, in turn, hit other scene objects at which point another set of rays will have to be cast over the corresponding hemisphere. The recursive process continues up to a specified depth, and can end early of a ray misses all objects (in which case there is no light arriving at our original pixel along that particular light path), or when the ray hits a light source, at which point the colour contribution at the pixel is computed from the light source colour and the BRDFs of all objects encountered along the way.
There are two components to the sampling performed by a distribution ray tracer: Sampling light over the area of a pixel (which accounts for the the fact pixels are square and have a non-negligible area), and sampling over a hemisphere centered at an intersection point to evaluate the rendering equation. Both of these steps, and how their contributions are added up to determine the final pixel colour, are described in detail below.

10.3 Integration at a pixel

To compute the intensity of an individual pixel, we need to evaluate a 2D integral, so we need to determine K 2D points (α_i, β_i), and compute:

$$\Phi_{i,j} \approx \frac{(\alpha_{max} - \alpha_{min})(\beta_{max} - \beta_{min})}{K} \sum_{i=1}^{K} H(\alpha_i, \beta_i) \tag{3}$$

In other words, we pick N points within the pixel, cast a ray through each point, and then average the intensities of the rays (scaled by the pixel's area $(\alpha_{max} - \alpha_{min})(\beta_{max} - \beta_{min})$. These samples can be chosen randomly, or uniformly-spaced.

Example:
The simplest way to compute this is by uniformly-spaced samples (α_m, β_n):

$$\alpha_m = (m-1)\Delta\alpha, \quad \Delta\alpha = (\alpha_{max} - \alpha_{min})/M \tag{4}$$
$$\beta_n = (n-1)\Delta\beta, \quad \Delta\beta = (\beta_{max} - \beta_{min})/N \tag{5}$$

and then sum:

$$\Phi_{i,j} \approx \Delta\alpha\Delta\beta \sum_{m=1}^{M}\sum_{n=1}^{N} H(\alpha_m, \beta_n) \tag{6}$$

However, Monte Carlo sampling — in which the samples are randomly-spaced — will usually give better results.

10.4 Shading integration

Our goal in shading a point is to compute the integral:

$$L(\vec{d}_e) = \int_{\phi\in[0,2\pi]} \int_{\theta\in[0,\pi/2]} \rho(\vec{d}_e, \vec{d}_i(\phi,\theta))\, L(-\vec{d}_i(\phi,\theta))\, (\vec{n}\cdot\vec{d}_i)\, \sin\theta\, d\theta d\phi \tag{7}$$

The simplest way to to this is to choose uniformly-spaced values of ϕ and θ as follows:

$$\theta_m = (m-1)\Delta\theta, \quad \Delta\theta = (\pi/2)/M \tag{8}$$
$$\phi_n = (n-1)\Delta\phi, \quad \Delta\phi = 2\pi/N \tag{9}$$

This divides up the unit hemisphere into MN solid angles, each with area approximately equal to $\sin\theta\Delta\theta\Delta\phi$. Applying 2D numerical integration gives:

$$L(\vec{d}_e) \approx \sum_{m=1}^{M}\sum_{n=1}^{N} \rho(\vec{d}_e, \vec{d}_i(\phi,\theta))\, L(-\vec{d}_i(\phi,\theta))\, (\vec{n}\cdot\vec{d}_i)\, \sin\theta\, \Delta\theta\, \Delta\phi \tag{10}$$

A slightly better approximation uses Monte Carlo sampling, selecting sample directions randomly over the hemisphere. The BRDF of the material can be used to determine the directions along which incoming light contributes the most toward $L(\vec{d_e})$, and sampling can be concentrated along these directions (this is known as importance sampling, and we'll look at how to do it for diffuse surfaces in a moment).
Putting everything together: The ray-tracer, the BRDF model for the materials present in the scene, and the sampling process for light directions, results in images that very accurately simulate realistic surfaces, and global illumination effects. Unlike the simple ray tracer, we no longer have to worry about special cases such as specular, diffuse, mirror, etc., instead, the sampling process and the BRDF account for the contributions multiple light paths have toward the final value of pixels in the image.

10.5 Distribution Ray Tracer

for each pixel (i,j)
  $<$ choose N points $\bar{x}_k$ within the pixel's area $>$
  for each sample k
    $<$ compute ray $\vec{\mathbf{r}}_k(\lambda) = \vec{\mathbf{p}}_k + \lambda\vec{\mathbf{d}}_k$ where $\vec{\mathbf{d}}_k = \vec{\mathbf{p}}_k - \vec{\mathbf{e}}$ $>$
    I_k = rayTrace($\vec{\mathbf{p}}_k$, $\vec{\mathbf{d}}_k$, 1)
  end **for**
  setpixel(i, j, $\Delta i \Delta j \sum_k I_k / N$)
end **for**

The rayTrace and findFirstHit procedures are the same as for Basic Ray Tracing. However, the new shading procedure uses numerical integration:

distRtShade(OBJ, $\vec{\mathbf{p}}$, $\vec{\mathbf{n}}$, $\vec{\mathbf{d}}_e$, depth)
  $<$ choose N directions (ϕ_k, θ_k) on the hemisphere $>$
  for each direction k
    I_k = rayTrace($\vec{\mathbf{p}}$, $\vec{\mathbf{d}}_k$, depth+1)
  end **for**
  return $\Delta\theta\Delta\phi \sum_k \rho(\vec{\mathbf{d}}_e, \vec{d_i}(\phi_k, \theta_k)) I_k \sin\theta_k$

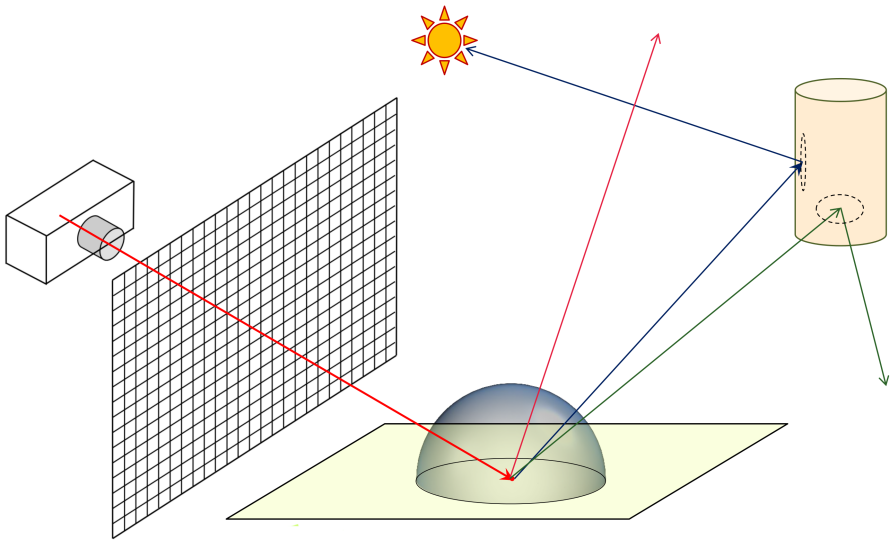
10.6 Computational Burden of Distribution Ray Tracing

Given sufficient sampling, DRT produces the most realistic scenes, limited only by the accuracy and completeness of the BDRF model used to represent surfaces in the scene, and by the modeling of light-sources. However, this comes at the cost of a significant computational expense.

Estimating the rendering equation takes the form of a breadth-first-search in the space of possible light paths through the scene. The number of rays that need to be traced in order to obtain a single colour value for a pixel grows exponentially with each additional level of recursion. Even with extensive use of importance sampling and acceleration structures for intersection testing, the computational expense of DRT may quickly grow beyond what is practical.
An alternate way of obtaining an approximation to the value of the rendering equation, but at a lower computational cost than DRT, is presented next.

10.7 Path Tracing

In Path Tracing (PT), the goal is to approximate the rendering equation with as little sampling as possible. The actual integration over the area of the pixel, and the hemisphere around a surface point is the same. However, the sampling order for possible light paths is quite different.
In path tracing, we define a fixed number of sample rays N to cast for each pixel. These are randomly chosen over the area of the pixel, as described above. When a ray hits an object, instead of sampling over the hemisphere centered at the intersection point, we choose a single direction from that hemisphere, in a way that is appropriate for the BRDF of the material, and trace *only that light direction*. The same process is applied at every intersection, selecting a single light path to trace from all available directions as shown below.
Sampling of light arriving at a surface from different directions happens because for every initial ray through a random location inside a pixel's area, a different random path will be chosen.



Path Tracing process. At every intersection point, the path tracer selects a single direction at random from among all possible paths light could bounce off the surface. It then traces the corresponding ray. When a ray hits a lightsource, we add its contribution to the corresponding image pixel's colour, suitably modulated by the BRDFs of objects encountered along the path to the pixel.

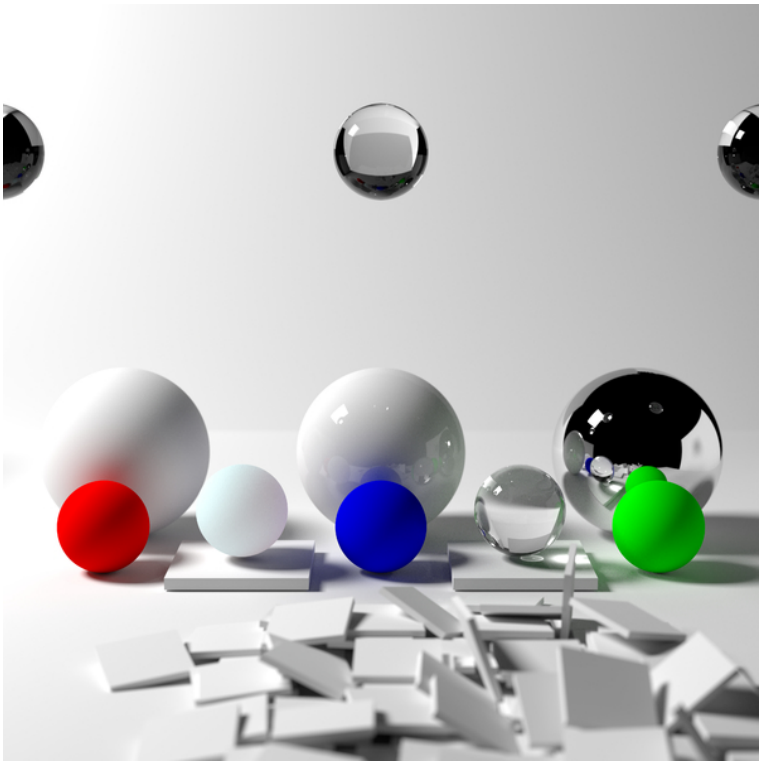
If at some point a ray leaves the scene, we know there is no light arriving along that specific light path. If, on the other hand, after a number of bounces we hit a light source, we compute the contribution of this particular light path to the pixel's colour using the light source's brightness modulated by the BRDF of each object that the light bounced off (or refracted through) before reaching the pixel.
This sampling pattern has significant advantages over DRT. It avoids the exponential growth in the number of rays cast, it can trade off computational expense for rendering accuracy - enabling us to quickly obtain an approximation to the rendering equation with only a few samples per pixel, and it keeps the computational expense of each sample constant - for each initial ray, we only cast an additional D rays, where D is the maximum recursion depth we are willing to explore.
While DRT does a more complete sampling of the light traveling through a scene, PT is able to produce excellent results at a fraction of the computational cost. Certain global illumination effects may require more sampling, but overall, PT has much better performance than DRT and is now the preferred rendering method for state-of-the-art rendering.
The path tracing algorithm is summarized below.

```
Path Tracer

1.0  For each pixel in the image

   1.1        For each of N samples

      1.1.2  Random sample a point within the pixel's area

      1.1.3  Cast a ray in the direction of the sampled point

          2.0   Trace ray to find intersections with objects
                in the scene

                2.1   If maxim recursion depth reached, return

                2.2   If the ray does not hit an object, return

                2.3   If the ray hits a lightsource, increment
                      brighness for that pixel by the light
                      source's intensity modulated by the BRDF
                      of any objects encountered along the
                      path.

                      else

                2.4   The ray hit a surface. Random-sample a
```



```
                    direction over the hemisphere centered at
                    the intersection point, form a ray in this
                    direction, and go back to 2.0.
```

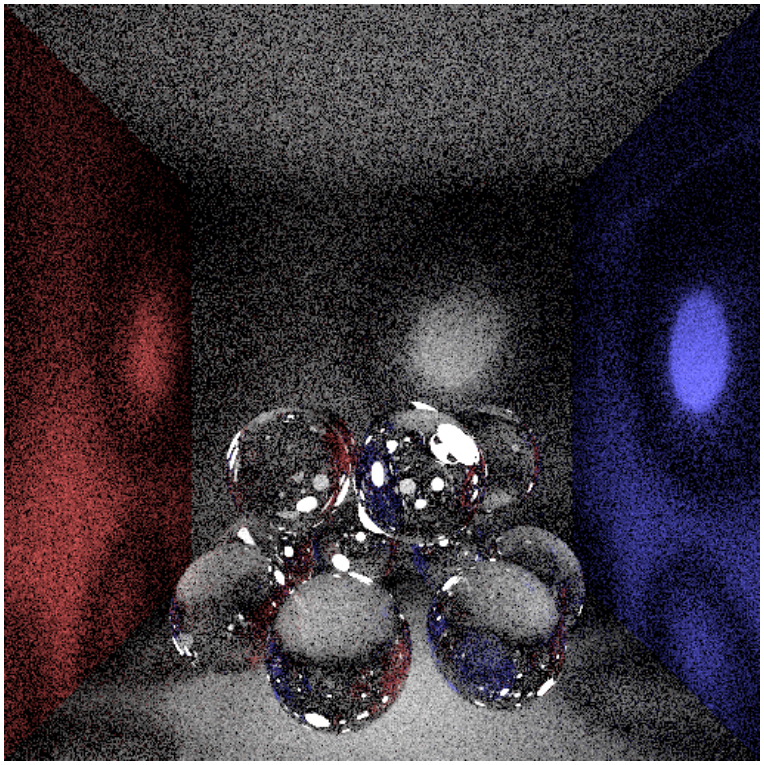
As we shall see below, with a few careful improvements, path tracing is able to render complex scenes that show the full range of global illumination effects at a reasonable computational cost.



Path traced scene showing global illumination effects including smooth shadows, caustics, and colour bleeding. [Source: Wikipedia, Author:Qutorial]

10.8 Importance Sampling

Path tracing does a good job of approximating the rendering equation with a reasonably low amount of sampling - at least compared to DRT. However, this comes at the price of noise in the rendered image. This is particularly noticeable for diffuse materials, which should produce smooth-looking surfaces. Unless a large amount of sampling is carried out, diffuse surfaces tend to look noisy in scenes where light sources are relatively small.



Standard path Tracing, 1000 samples per pixel. The lightsource is at the center of the scene. Notice how noisy the diffuse surfaces look - even at a 1000 samples per pixel, the probability that many of the sampled paths will reach the lightsource are relatively low, diffuse surfaces show high variance because of this

The standard path tracing sampling process is unbiased in that all possible sample directions have the same chance of being chosen. There is no preference for some specific directions, and therefore the resulting rendered scene is a fair sampling of the light traveling through the scene. It turns out that we sample in a smarter way while preserving the property that the rendered scene will still be a fair sampling of the illumination in the scene.

The process is called importance sampling. It is easiest to see how it works if we think about diffuse surfaces in the sample image above. Standard path tracing will choose with equal probability a direction that is very close to the normal for the surface, or one that is almost perpendicular. However, even if both of these sampled directions will result in rays that hit a light source, the latter sample direction will contribute almost nothing to the value of the rendering equation at the surface. This is because as we have seen, for diffuse materials, perceived brightness is scaled by the cosine of the angle between the incoming light direction and the normal.

For the sample direction that is almost perpendicular to the normal, this will result in a contribution close to zero. In some sense, we have wasted a sample in evaluating a direction that will end up contributing almost nothing to the appearance of the surface.

Importance sampling takes advantage of the fact that most material BRDFs do not reflect light

equally over all possible directions, and chooses directions to sample so as to give higher probability to those directions that contribute the most to the visual appearance of the material.

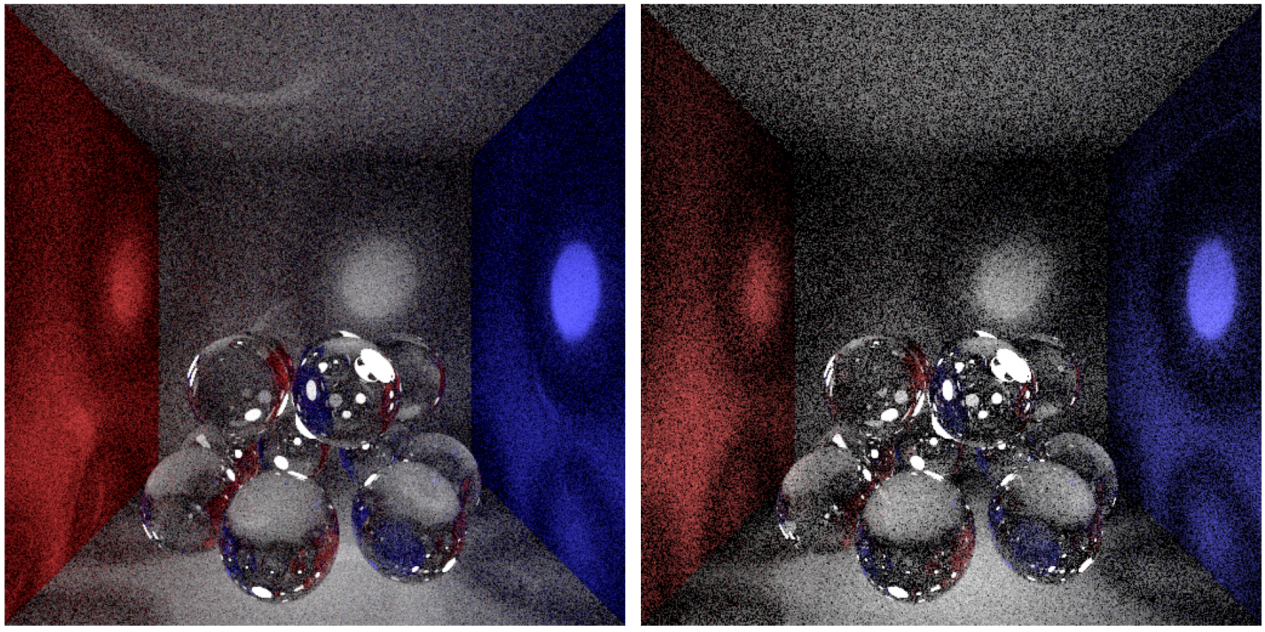
Example:

For perfectly diffuse surfaces, incoming light contributes to radiance from the surface by an amount proportional to $\vec{n} \cdot -\vec{s}$, where $\vec{n}$ is the normal at the surface, and $\vec{s}$ is the direction of the incoming light ray.
To use importance sampling with diffuse surfaces, the sampling directions have to be chosen with probability proportional to $\vec{n} \cdot -\vec{s}$ so as to choose more often those directions that have the greater contributions to the radiance at the surface. A direction vector $\vec{d}$ drawn from a cosine-weighted distribution can be obtained from the following equations: $r = \sqrt{u_1}$

$\theta = 2\pi u_2$
$\vec{d} = [r\cos(\theta) \quad r\sin(\theta) \quad \sqrt{(1.0 - x^2 - y^2)}]$

Here, u_1, and u_2 are two uniformly distributed random numbers in $[0, 1]$. Given direction $\vec{d}$, we continue the path tracing process as before, but now we are no longer sampling fairly over possible light directions, so in order to make sure our rendered image is still a fair sample of light transport in the scene, the contribution of this sample to the radiance estimate at the surface has to be weighted by a factor of $1/(\vec{n} \cdot \vec{d})$. This cancels out the bias introduced by sampling $\vec{d}$ with probability proportional to $\vec{n} \cdot \vec{d}$.

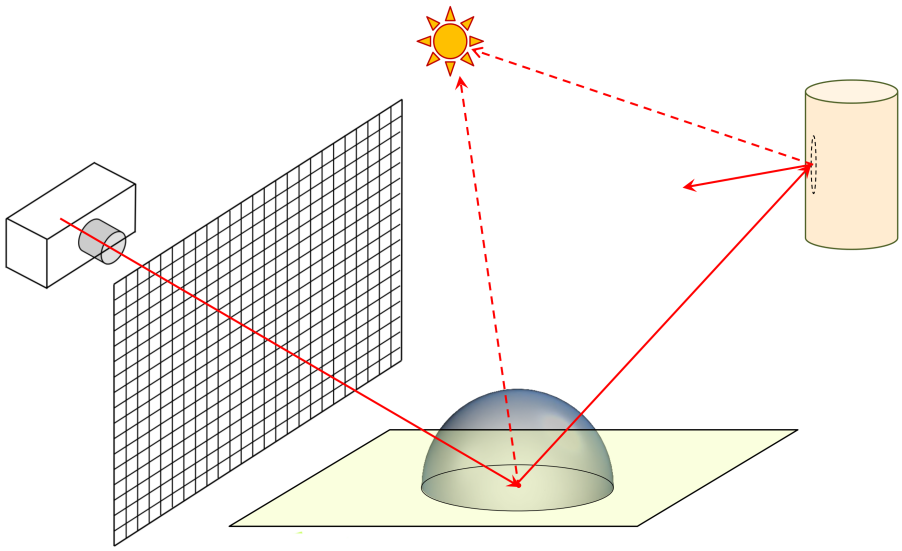
Importance sampling can be applied to any BRDF and its benefits will be greater the less isotropic (reflecting equally in all directions) the BRDF is. All that is needed is a procedure to draw a sample with probability proportional to the BRDF for each corresponding direction, and weighting of the sample by the inverse of the BRDF.



Comparison of path tracing with (left) and without (right) importance sampling, both images were rendered with 1000 samples per pixel. Note how the importance sampling process results in smoother diffuse surfaces for the same amount of sampling.

10.9 Explicit Light Sampling

Even with importance sampling, obtaining smooth estimates for diffuse surfaces may require more sampling than we are willing to do. While we can improve our estimates of the rendering equation via importance sampling, there is still high variance due to the probability that many of the sampled rays will not reach a light source, and therefore contribute nothing to radiance at the surface.
One technique used to improve rendering of diffuse surfaces for a small increase in computation is to use explicit light sampling. This consists of modifying the path tracing algorithm so that for diffuse surfaces, in addition to the random sampled direction which will be traced next, we cast one ray to a randomly sampled point on a randomly selected light source. If the ray is not blocked by objects in the scene, we accumulate brightness just as we would if a randomly sampled direction had happened to hit the same light source.

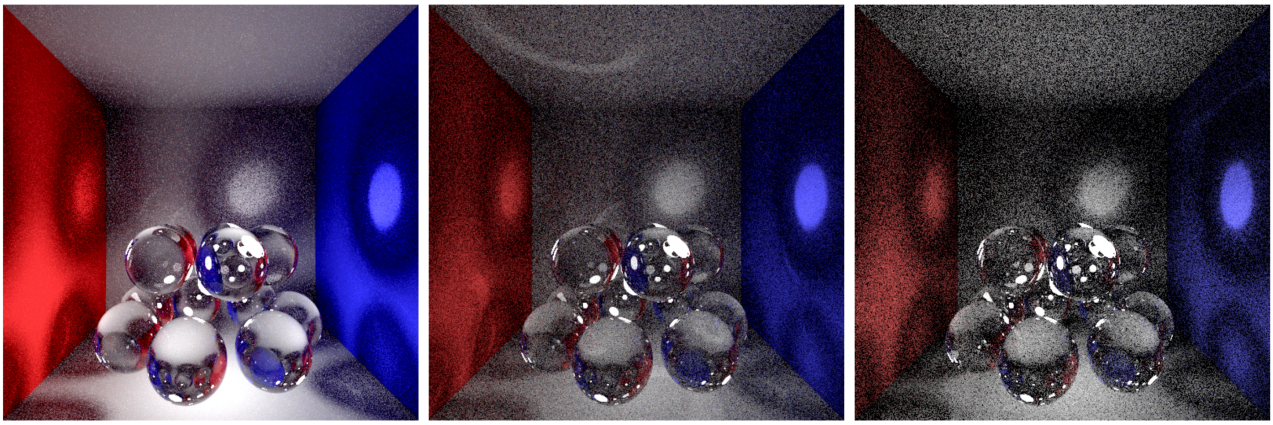


Explicit lightsource sampling. For diffuse surfaces, we cast two rays, the random sampled one (solid lines) and an additional ray in the direction of a lightsource (dashed lines). If the ray toward the lightsource is not blocked, we add a suitably weighted contribution from this ray to the radiance at the corresponding surface point. This is only done for diffuse surfaces

There are two important details to take care of here: The weight of the explicit light sample is not the same as for all other samples (it wasn't chosen randomly, so we have to account for the actual chance that we would actually hit the light source by random sampling). Secondly, since we now send out two rays at each intersection point, we have to ensure we do not double-count brightness from the same light source. This means that if the explicit ray is not blocked, then if the random sampled ray happens to hit the same light source, we do not accumulate any brightness for the sampled ray.

The weighting of the contribution from the light source sample accounts for the relative size and orientation of the light source with regards to the surface point. For fully random sampling this is not an issue since the chance of hitting the light source with a random direction is proportional to its relative size, but for explicit light sampling we need to account for these factors. Therefore, the contribution of the explicit sample ray is weighted by $w = min(1, \frac{A_{ls}(\vec{n}\cdot\vec{l})(\vec{n}_{ls}\cdot\vec{-l})}{d^2})$, where A_{ls} is the area of the light source, $\vec{n}$ is the surface normal, $\vec{l}$ is a vector in the direction from the surface to the light source, $\vec{n}_{ls}$ is the normal at the light source, and d is the distance from the surface to the light source.

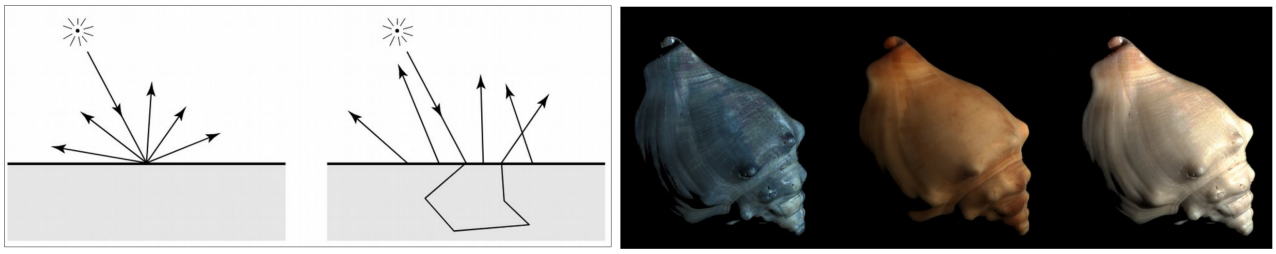
Note we do not perform explicit light sampling for reflective or refractive surfaces. This is only for diffuse materials.



Path tracing with explicit light sampling and importance sampling (left), with only importance sampling (middle), and withour eithe of these (right). All images generated with 1000 samples per pixel. Explicit sampling of light sources greatly enhances the rendering of smooth surfaces for the same amount of sampling

10.10 Further Extensions

There are many surfaces that behave in ways that are not well modeled with a regular BRDF. Certain types of materials are at least partially translucent, light arriving at the back of an object will travel through, get bounced around inside, and come out on a different location. Such materials are often modeled using sub-surface scattering BRDFs (BSSRDF).
Modeling sub-surface scattering is beyond the scope of this introductory course, but it is addressed in detail on many research papers that propose different models for simulating translucent materials. Examples of surfaces that require sub-surface scattering include porcelain, ceramics, and human skin.



(Right) Comparison of typical diffuse material with one that shows sub-surface scattering [Source: Jensen et al., 2001]. (Left) Rendering of a sea-shell combining a diffuse surface with sub-surface scattering. [Source: Wikipedia, Author: Meekohi]

Optical effects that can not be rendered using standard BRDFs also include scattering from intervening media such as smoke, fog, certain liquids like milk, or transparent solids for which light scattering occurs as it travels through the media. Approaches to rendering such intervening media are an active area of research in computer graphics.

