General Guidelines:

⇒ notice how the text in the body, captions and equations avoids superfluous white space

⇒ don't overcomplicate your:
- ↳ text/exposition
- ↳ diagrams
- ↳ math/equations/terminology

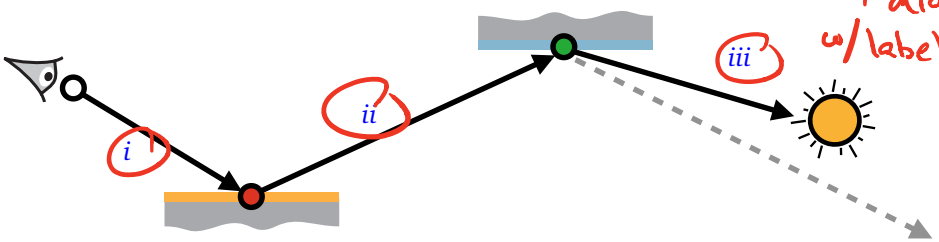
IMPLICIT PATH TRACING

ECSE 546 / COMP 599 – Scribe Note

⇒ Text, math & diagrams should be designed together, not separately

Derek Nowrouzezahrai ← include your name(s) & ID(s)

1.1 Implicit Path Tracing



Implicit path tracing traces out paths starting from the eye (*i*) – scattering at surfaces in the scene (*ii*) – in the blind hope that they will eventually (*iii*) hit a light source.

Overview Implicit path tracing (IPT) is the simplest algorithm capable of accurately computing global illumination effects in a virtual scene. This note assumes *surface-only* transport (i.e., no participating media) and fully opaque objects (i.e., no transmission or refraction – all scattering is due to reflection), although the algorithm can be generalized to handle a variety of effects.

IPT is a "forward" ray-tracing technique[1], meaning that paths are traced starting **from the eye**, with the hope of eventually hitting a light, and thus forming a *full light transport path*. The final image is generated by averaging the contribution of many such paths, through each pixel. The main drawback of IPT is that it forms light transport paths somewhat blindly, in the sense that a (potentially long, unfinished) path can eventually end without ever hitting a light source (e.g., gray dotted arrow in the overview figure).

The IPT algorithm is a Monte Carlo (MC) estimator of the rendering equation [Kajiya86]

$$L(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_{\Omega_{\vec{\mathbf{n}}}} L(\mathbf{x}',-\omega_i)\, f_r(\mathbf{x},\omega_i,\omega_o)\,(\vec{\mathbf{n}}\cdot\omega_i)\,\mathrm{d}\omega_i\,, \tag{1}$$

where $L(\mathbf{x},\omega_o)$ is the outgoing radiance at a surface point $\mathbf{x}$ with normal $\vec{\mathbf{n}}$ (e.g., the closest visible point from a pixel) in the direction ω_o towards the viewer. The outgoing radiance is the sum of the emitted radiance L_e at $\mathbf{x}$ and the **reflected radiance**: an integral, over all incident lighting directions $\Omega_{\vec{\mathbf{n}}}$ in the upper hemisphere around $\mathbf{x}$, of the product of incident radiance $L(\mathbf{x}',-\omega_i)$ from the point $\mathbf{x}' = \texttt{ray}(\mathbf{x},\omega_i)$ nearest to $\mathbf{x}$ in the direction ω_i, the *bidirectional reflectance distribution function* (BRDF) f_r at $\mathbf{x}$, and a cosine foreshortening factor.

Basic Idea At each pixel, the IPT algorithm averages the value of many 1-sample Monte Carlo integral estimates $\mathrm{E}^{1\mathrm{spp}}_{\Omega}$ of Equation 1 :

$$\mathrm{E}^{1\mathrm{spp}}_{\Omega}\left[L(\mathbf{x},\omega_o)\right] = L_e(\mathbf{x},\omega_o) + L(\mathbf{x}',-\omega_s)\, f_r(\mathbf{x},\omega_s,\omega_o)\,(\vec{\mathbf{n}}\cdot\omega_s)/pdf(\omega_s), \tag{2}$$

where, here, each 1-sample estimate forms a light path by tracing rays through the scene, evaluating the terms in Equations 1 and 2 along the way (see Algorithm Details below.) In Equation 2 we sample a **direction** $\omega_s \sim pdf(\omega)$ from a valid sampling probability distribution function pdf over the solid angle measure. We form a full path by recursively evaluating the incident radiance $L(\mathbf{x}',-\omega_s)$. Note that, while Equation 2 uses the *solid angle form* of the rendering equation (as written in Equation 1), we could just as easily have used a *surface area form*; in doing so, the integration domain in Equation 1 would become the set of all scene surfaces $\mathcal{M}$ and the corresponding 1-sample MC integral estimator $\mathrm{E}^{1\mathrm{spp}}_{\mathcal{M}}$ would sample **surface points** in the scene:

$$\mathrm{E}^{1\mathrm{spp}}_{\mathcal{M}}\left[L(\mathbf{x}\to\mathbf{x}_o)\right] = L_e(\mathbf{x}\to\mathbf{x}_o) + L(\mathbf{x}\leftarrow\mathbf{x}_s)\, f_r(\mathbf{x}_s\to\mathbf{x}\to\mathbf{x}_o)\, G(\mathbf{x},\mathbf{x}_s)/pdf(\mathbf{x}_s), \tag{3}$$

[1] Some sources use the term "backward" ray-tracing in this context, since light flows *from* lights *to* the eye in reality.

where arrows denote directional relationships between points, the *pdf* is one for choosing *points on scene surfaces*, and the geometry term $G(\mathbf{x}, \mathbf{x}_s) = (\cos\vec{\mathbf{n}}_{\mathbf{x}} \cos\vec{\mathbf{n}}_{\mathbf{x}_s})/|\mathbf{x}-\mathbf{x}_s|^2$ includes the original cosine foreshortening factor and an *additional* cosine foreshortening about the normal of the "emitting" surface point (and the squared-distance between the two points; see Step *ii* b). These additional terms are due to the Jacobian mapping between differential solid angles (i.e., when integrating over incident directions $\Omega_{\vec{\mathbf{n}}}$) and differential surface elements (i.e., when integrating over scene surfaces $\mathcal{M}$). Specifically, by substituting $\mathrm{d}\omega_i = G(\mathbf{x}, \mathbf{x}_i)\,\mathrm{d}\mathbf{x}_i / \cos\vec{\mathbf{n}}_{\mathbf{x}}$ into Equation 1 we can arrive at the surface-area form where the reflected radiance is an integral over $\mathcal{M}$ instead of $\Omega_{\vec{\mathbf{n}}}$: $L(\mathbf{x}\to\mathbf{x}_o) = L_e(\mathbf{x}\to\mathbf{x}_o) + \int_{\mathcal{M}} L(\mathbf{x}\leftarrow\mathbf{x}_i)\, f_r(\mathbf{x}_i\to\mathbf{x}\to\mathbf{x}_o)\, G(\mathbf{x},\mathbf{x}_i)\,\mathrm{d}\mathbf{x}_i$.

Perhaps the most important concept in IPT is that of a *light transport* ***path*** and, specifically, one beginning at the eye and ending at the light. Both Equations 2 and 3 naturally lead to the formation of such paths due to the **recursive** incident radiance term $L(\mathbf{x}', -\omega_i) \equiv L(\mathbf{x}\leftarrow\mathbf{x}_i)$; we (recursively) apply an MC estimator to evaluate this term since the *incident* radiance at $\mathbf{x}$ from direction ω_i (or, equivalently, from surface point $\mathbf{x}_i$) is equal to the *outgoing* radiance at $\mathbf{x}_i$ back towards $\mathbf{x}$. In doing so, we can arrive at the *path space formulation* of the rendering equation as

$$L(\mathbf{x}_0\leftarrow\mathbf{x}_1) = \int_{\mathcal{P}} W_e(\mathbf{x}_0,\mathbf{x}_1)\, L_e(\mathbf{x}_k\to\mathbf{x}_{k-1})\, G(\mathbf{x}_0,\mathbf{x}_1) \underbrace{\prod_{j=1}^{k-1} f_r(\mathbf{x}_{j+1}\to\mathbf{x}_j\to\mathbf{x}_{j-1})\, G(\mathbf{x}_j,\mathbf{x}_{j+1})}_{T(\overline{\mathbf{x}})}\,\mathrm{d}\overline{\mathbf{x}}\,, \tag{4}$$

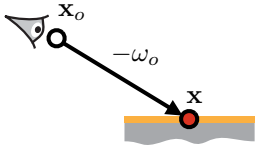
where $\mathcal{P}$ is the space of **all** light paths $\overline{\mathbf{x}} = \{\mathbf{x}_0 \ldots \mathbf{x}_k\}$, W_e models the sensor's importance response[2], only those paths that "end" at a vertex $\mathbf{x}_k$ on a light contribute non-zero radiance to $\mathbf{x}_0$, and $\mathrm{d}\overline{\mathbf{x}} = \prod_{i=0}^{k}\mathrm{d}\mathbf{x}_i$. The path's **throughput** $T(\overline{\mathbf{x}})$ combines evaluations of the geometry and BRDF terms at each path vertex. We can write a 1-sample MC estimator of Equation 4 as:

$$\mathrm{E}^{1\mathrm{spp}}_{\mathcal{P}} = L_e(\overline{\mathbf{x}}_{s,k}\to\overline{\mathbf{x}}_{s,k-1})\, T(\overline{\mathbf{x}}_s)/pdf(\overline{\mathbf{x}}_s)\,, \tag{5}$$

with a *pdf* over the distribution of light paths.

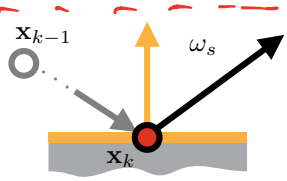
Remarks: In Algorithm Details we present the IPT algorithm in the form of a solution to Equation 2, but it is important to understand that both the solid angle and surface area formulations can be generalized into the path space formulation. Indeed, one can readily form a *pdf* over paths by combining (i.e., even during the process of path construction) the product of *pdf*s over surface points and *pdf*s over directions, as long as the proper care is taken to incorporate the (resp. inverse) Jacobian that maps $\mathrm{d}\omega_i$ to (resp. from) $\mathrm{d}\mathbf{x}_i$.

Algorithm Details As with most rendering algorithms, IPT relies on the ability to query the geometry of a scene. For the purposes of our implementation, we will assume that a ray tracer is used here, where rays are traced to compute (nearest-surface) object intersections. To begin, we assume that camera rays are generated and traced through each pixel. For each of these rays, the core IPT algorithm evaluates a 1-sample MC estimate of the rendering equation (i.e., Equation 2) at the nearest surface point $\mathbf{x}$ intersected by the ray and, ideally, the average of many such estimates is used to compute the final image. As such, IPT is trivially parallelizable over both pixels and samples (i.e., paths), and one can easily incorporate distribution effects such as pixel anti-aliasing, depth of field, or motion blur by simply assigning different spatial, lenticular, or time coordinates to each camera ray traced through a single pixel. The `Lo` function in our code listing implements the 1-sample MC estimate in Equation 2.



Step *i*: Rays are traced through each pixel, scattering off surfaces, in order to form paths in IPT.

[2] Not to be confused with *importance sampling*, *importance* is the radiometric dual of *radiance*.



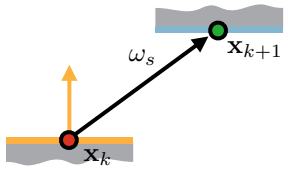
Step *ii* a: Paths are built recursively, by shooting a ray at each path vertex, to sample the incident radiance.

At each (non-emissive[3]) surface interaction, the IPT algorithm *incrementally* forms a path and accumulates an MC estimate of its throughput T. In the solid angle formulation, first an "outgoing" ray (with direction $\omega_s \sim pdf(\omega)$) evaluates the incident radiance $L(\mathbf{x}', -\omega_s)$ at the current vertex $\mathbf{x}_k$, where $\mathbf{x}' = \mathbf{x}_{k+1} = \texttt{ray}(\mathbf{x}_k, \omega_s)$ is the surface point nearest to $\mathbf{x}_k$ in direction ω_s. We then multiply the accumulated throughput $T(\overline{\mathbf{x}})$ by the cosine-weighted BRDF at $\mathbf{x}_k$ and divide by $pdf(\omega_s)$ to form our estimate. We repeat the process recursively at the next vertex. Concretely, in the solid angle formulation, we multiplicatively accumulate

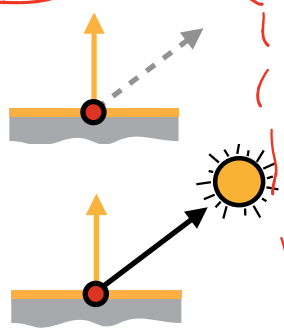
$$f_r\Big(\mathbf{x}_k, \omega_s, (\mathbf{x}_{k-1} - \mathbf{x}_k)/|(\mathbf{x}_{k-1} - \mathbf{x}_k)|\Big)(\vec{\mathbf{n}}_{\mathbf{x}_k} \cdot \omega_s)\Big/ pdf(\omega_s) \tag{6}$$

until path termination (see below). The choice of sampling distribution pdf influences the variance of our estimator but, with a valid[4] pdf, the estimator remains unbiased. Our implementation uses a cosine distribution about the normal, $pdf(\omega) = (\vec{\mathbf{n}} \cdot \omega)/\pi$ (implemented as `sampleCosine` in the code listing). To draw samples according to this pdf, we rotate random cosine-distributed directions $\widehat{\omega} = (\theta_s, \phi_s) = (\arccos(\sqrt{\epsilon_0}), 2\pi\epsilon_1)$ about the z-axis to align about $\vec{\mathbf{n}}$, obtaining ω_s, where ϵ_0 and ϵ_1 are uniform random numbers in $[0, 1)$.

For completeness, using the surface area formulation and *explicitly* sampling $\mathbf{x}_{k+1}$ according to a pdf over surfaces (instead of *implicitly* sampling it as a function of ω_s, $\texttt{ray}(\mathbf{x}_k, \omega_s)$), we would accumulate the geometry-weighted BRDF product divided by the pdf: $G(\mathbf{x}_k, \mathbf{x}_{k+1})\, f_r(\mathbf{x}_{k+1} \to \mathbf{x}_k \to \mathbf{x}_{k-1})/pdf(\mathbf{x}_s)$.

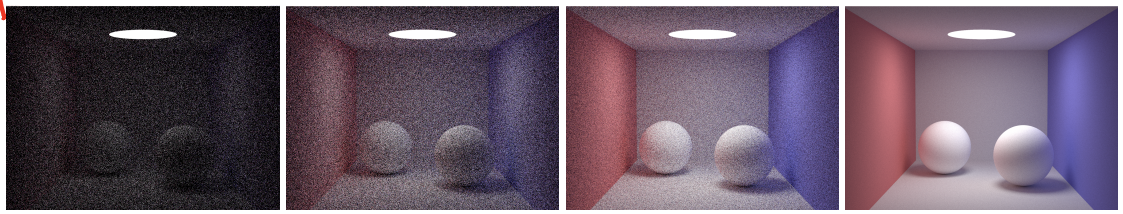


Step *ii* b: We build paths by incrementally sampling vertices.



Step *iii*: In IPT, paths are only terminated once they escape the scene (top) or hit a light source (bottom).

This procedure of sampling vertices along the path and multiplicatively accumulating the (inverse pdf-weighted) throughput, one ray-intersection at a time, continues until we meet a *termination criterion*. In IPT, paths can terminate in two ways: either the "last" ray fails to intersect a surface (i.e., it "leaves the scene"), or it hits a light source (Step *iii*). When hitting a light, the final value of the 1-sample estimator $\mathrm{E}_{\Omega}^{1\mathrm{spp}}$ is the product of the emitted radiance at the light and the accumulated throughput estimate; otherwise, the estimate's final value is 0. Repeating this entire process over many paths per pixel, averaging the results of the individual 1-sample MC estimates of each such path, eventually leads to a converged image. With low sample rates, the principal high-frequency noise artifact of IPT is due exclusively to the *variance* in the MC estimator (see convergence figure).

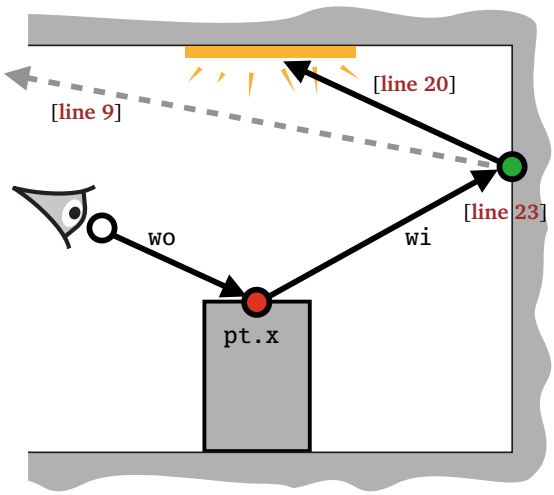


Convergence : As we increase the MC sample rate, variance decreases and IPT convergences to the correct solution.

[3] We assume that luminaires do not reflect light and vice-versa (i.e., that reflective surfaces have no emission).
[4] A pdf is valid if it is non-zero in all regions of the domain where the integrand is non-zero.

Implementation Details The pseudocode below assumes that we have access to a ray tracer that can `intersect` `ray`s with the scene geometry and return `surface` intersection data, such as the intersection point `x`, the normal `n` at `x`, and a function `fr` for the BRDF at `x`. The `Lo` function returns an RGB color corresponding to the 1-sample MC estimate in Equation 2, and so it should (ideally) be called and averaged *several times* per pixel; eye rays `wo` are generated from the camera through each pixel before calling `Lo`.

```
1  rgb Lo(const ray &wo){
2   // variable to store intersection data
3   surface pt;
4
5   // trace the ray wo into the scene
6   bool hit = intersect(wo, &pt);
7
8   // if ray exits the scene
9   if (!hit) return rgb(0.0,0.0,0.0);
10
11  // sample cosine-weighted direction about the normal
12  vec3 wi = sampleCosine(pt.n);
13
14  // Russian Roulette termination
15  real rr = 1.0, p = magnitude(pt.fr(wi,wo));
16  if (rand() < p) rr = f * (1.0/p);
17  else return pt.e;
18
19  // hit a light, return emission...
20  if( pt.e != rgb(0.0, 0.0, 0.0) ) return pt.e;
21
22  // otherwise, compute recursive integral
23  return rr * pt.fr(wi,wo) * Lo(ray(pt.x, wi));
24 }
```



The code listing above treats the general case of arbitrary BRDFs, assuming that we have access to an `fr(wi,wo)` function at any intersected point. Our convergence renderings all assume fully diffuse surfaces, where $f_r(\mathbf{x}, \omega_i, \omega_o) = \rho_\mathbf{x}/\pi$ and $\rho_\mathbf{x} \in [0,1]^3$ is the (RGB) *diffuse reflectivity*.

Our code also implements an **unbiased** stochastic *Russian roulette* process for terminating (potentially infinite[5]) Monte Carlo processes (lines 15–17). To do so, we choose an *acceptance probability* `p` proportional to the (spectrally- and energy-normalized `magnitude` of the) BRDF: the logic here is that we want to favour paths that have vertices with high reflectance since, if they eventually hit a light, they will more likely have a non-negligible throughput magnitude. Conversely, if we hit a "dark" surface (i.e., one with a low BRDF magnitude), the product-term contribution of that vertex's BRDF to the throughput will reduce the eventual contribution of a light to the path's pixel; as such, we favour terminating the recursion in these cases.

If a Russian roulette trial results is the decision to continue the recursion (and, so, continue the path for another vertex; line 16), we weigh the Monte Carlo estimate by the reciprocal of the acceptance probability to not bias its expected value; otherwise, we terminate with the accumulated path contribution (i.e., the throughput times the current vertex's emission). The acceptance probability heuristic (line 15) still holds if our implementation assumes that emissive surfaces do not reflect light (i.e., have $f_r = 0$); however, we explicitly enforce this assumption on line 20. The final estimate (line 23) is simply Equation 2 with $pdf(\omega_s) = \vec{\mathbf{n}} \cdot \omega_s/\pi$.

Implicit Path Tracing Reference

[Kajiya86] James T. Kajiya. *The Rendering Equation*. Proceedings of the 13th annual conference on Computer graphics and interactive techniques (SIGGRAPH). 1986

[5] If, for example, the scene was a closed volume and the light source was very small, one could imagine requiring very long paths (and, thus, many recursive evaluations) before hitting the light to terminate the recursion.