# Počítačová grafika III (NPGR010)
# Prednáška: Pathtracing I

Peter Hmíra
21. november, 2012

## 1 Úvod

Počas minulej prednášky bolo vysvetlené využitie metódy integrovania Monte Carlo. Medzi výhody tejto metódy patrí hlavne jej jednoduchá implementácia a robustnosť pre rôzne funkcie. Preto sa využíva na niekoľko renderovacích algoritmov, z ktorých teraz popíšeme *distribution ray-tracing* a *pathtracing*. Na začiatok si zopakujeme, ako bude vyzerať estimátor pre *rovnicu odrazu*, ktorá má tvar

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_H L_i(x,\omega_i) f_r(x,\omega_i \to \omega_o) \cos\theta_i d\omega_i \tag{1}$$

Pre ňu príslušný estimátor pri rovnomernom vzorkovaní smerov bude

$$\hat{L_o}(x,\omega_o) = L_e(x,\omega_o) + \frac{2\pi}{N} \sum_{k=1}^{N} L_i(x,\omega_{i,k}) f_r(x,\omega_{i,k} \to \omega_o) \cos\theta_{i,k} \tag{2}$$

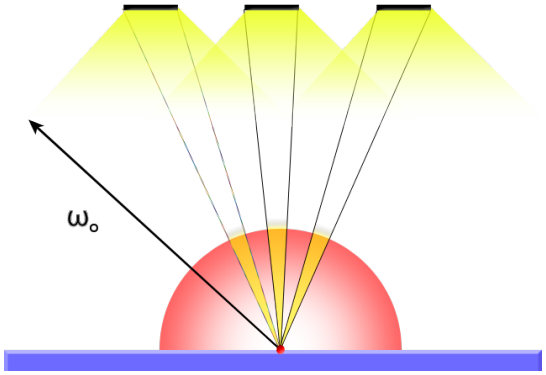
Kde

- $x$ je bod na scéne, v ktorom chceme vypočítať radianci
- $\omega_o$ je priestorový uhol medzi smerom odchádzajúcej radiancie a normálou na povrchu v bode $x$
- $\omega_i$ je priestorový uhol medzi smerom prichádzajúcej radiancie a normálou na povrchu v bode $x$
- $L_o(x,\omega_o)$ je výsledná odchádzajúca radiance z bodu $x$ v smere $\omega_o$
- $L_e(x,\omega_o)$ je radiance, ktorá vyžarovaná(*emitovaná*) z bodu $x$ v smere $\omega_o$
- $L_i(x,\omega_i)$ je radiance, ktorá dopadá na bod $x$ pod uhlom $\omega_i$
- $f_r(x,\omega_i \to \omega_o)$ je príslušná hodnota *BRDF* v bode $x$
- $\theta_i$ je elevácia - vertikálna zložka uhla $\omega_i$
- $N$ je počet vzorkov pri výpočte estimátoru
- $2\pi$ vyplýva z hustoty pravdepodobnosti $p(x)$, ktorá sa v prípade uniformného vzorkovanie hemisféry rovná $p(x) = \frac{1}{2\pi}$ a po úprave estimátoru hodnotou $\frac{1}{p(x)}$ dostávame v menovateli $2\pi$

ešte si pripomenieme tvar *zobrazovacej rovnice*, ktorá má tvar

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_H L_o(r(x,-\omega_i),\omega_i) f_r(x,\omega_i \to \omega_o) \cos\theta_i d\omega_i \tag{3}$$

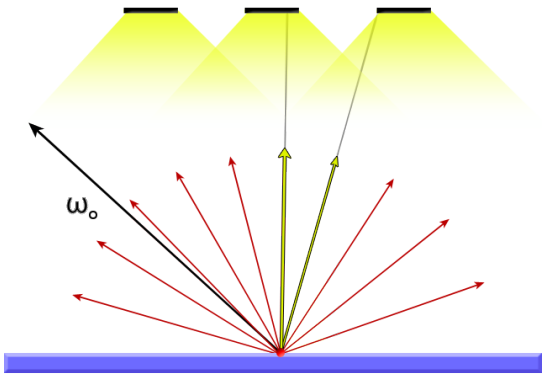
Načrtneme v 2D, ako môžu uvedené estimátory pomôcť pri odhadovaní odrazenej radiance.




**Obr. 1:** *hemisféricko-smerový odraz*

Na obrázku č. 1 máme 3 plošné zdroje svetla, povrch je vyznačený modrou a spomínaný bod $x$ je vyznačený červenou farbou. Červený gradient predstavuje pomyselnú jednotkovú guľu, na ktoré sa premietne plocha svetiel zo scény a na nej žltou vyznačený úsek, odkiaľ na bod $x$ dopadá nejaké svetlo, ktoré je následne odrazené v smere $\omega_o$. Na výpočet radiance odrazenej v smere $\omega_o$ by sme potrebovali spočítať (v 3D plošný) integrál žltého úseku na guli, čo môže byť v niektorých scénach príliš komplikované počítať analyticky.

V takom prípade je efektívnejšie použiť metódu vzorkovania, vygenerovať niekoľko vzorkov do hemisféry a pomocou metódy *Monte Carlo* vypočítať odhad hodnoty odrazenej radiance z bodu $x$ v smere $\omega_o$ čo je $L_o(x,\omega_o)$, ktoré chceme dostať. V našom 2D príklade si vygenerujeme uniformne 10 smerov a ak trafíme nejaký svetelný zdroj, dosadíme do vzťahu (2) a pričítame príspevok.



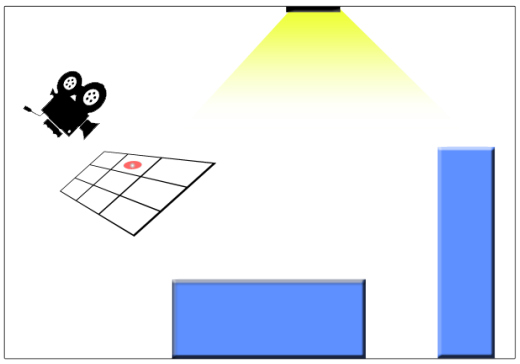

**Obr. 2:** *uniformné vzorkovanie hemisféry*

Na obrázku č. 2 je 10 vygenerovaných smerov *(červenou)* z čoho 2 sa trafili do svetla *(žltou)*, ostatné ich nezasiahli. Ako je vidno, pre nízky počet vzorkov je metóda veľmi nepresná.

## 2 Distribution ray-tracing

Predchádzajúci náčrt počíta čisto len s priamym osvetlením, neuvažuje radianci, ktorá dopadá do bodu $x$ po odraze od scény (prípadne viac odrazoch).

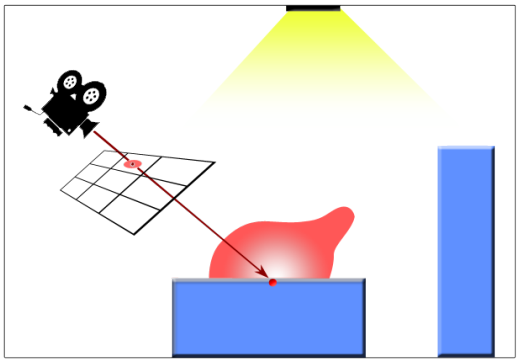
Predstavme si scénu, kde je daná pozícia kamery, objektov a svetiel v scéne. Pre každý pixel nás zaujíma hodnota radiance, ktorá ide v smere od bodu $x$ skrz pixel do kamery.



**Obr. 3:** *scéna*

Predpokladajme, že naša obrazovka má len $3 \times 3$ pixelov a scéna je nakonfigurovaná ako na obrázku č. 3. Na strope je jedno plošné svetlo, modrou farbou sú objekty a mriežka nech je naša obrazovka a na nej červenou je vyznačený pixel, skrz ktorý vedieme smer. Celá scéna nech je vo vnútri uzavretého boxu.

Veď me najprv smer od kamery skrz pixel.



**Obr. 4:** *prvý odraz*

Nájdeme najbližší priesečník so scénou, to bude hľadaný bod $x$ na obrázku č. 4 zobrazený červenou. Červený lalok vyjadruje priebeh funkcie BRDF v bode $x$ pre vstupný smer. Priebeh funkcie BRDF budeme neskôr potrebovať pri generovaní ďalších smerov.

V tomto prípade nás zaujíma hodnota radiance, ktorá prichádza z bodu $x$ do kamery. Jej hodnotu algoritmus *distribution raytracing* vyhodnocuje ako súčet radiance, ktorá je z bodu $x$ emitovaná a radiance, ktorá je z bodu $x$ odrazená. Algoritmus odhaduje zvlášť hodnotu iradiance, prichádzajúcej do bodu $x$ priamo zo svetiel v scéne a zvlášť, hodnotu, ktorá dopadá do bodu $x$ nepriamo po odraze.

### 2.1 Priame osvetlenie na ploche s obecnou BRDF

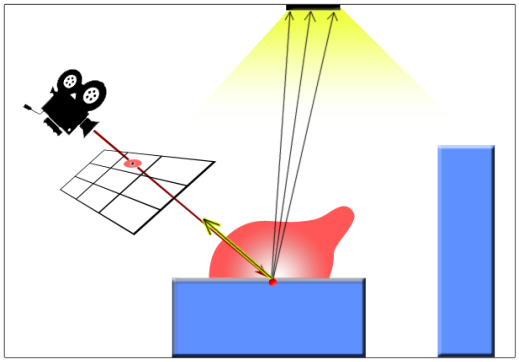
Odhadovaný integrál

$$L_o(x, \omega_o) = \int_A L_e(x \to y) f_r(y \to x \to \omega_o) V(x \leftrightarrow y) G(x \leftrightarrow y) dA \tag{4}$$

Príslušný estimátor pre uniformné vzorkovanie plochy zdroja svetla bude mať tvar

$$\hat{L_o}(x, \omega_o) = \frac{|A|}{N} \sum_{k=1}^{N} L_e(y_k \to x) f_r(y_k \to x \to \omega_o) V(x \leftrightarrow y_k) G(x \leftrightarrow y_k) \tag{5}$$

Kde $y_k$ je $k$-ta vzorka na plošnom svetle. Zvoľme si na našej scéne $N := 3$ tj. vygenerujeme na svetle v scéne (uniformne) 3 vzorky $y_1, y_2, y_3$.



**Obr. 5:** *vzorkovanie svetla*

Na obrázku č. 5 je žltou šipkou vyznačený odchádzajúci smer $\omega_o$, pre ktorý vypočítame odrazenú radianci.

### 2.2 Nepriame osvetlenie na ploche s obecnou BRDF

Odhadovaný integrál

$$L_o^{ind}(x, \omega_o) = \int_{H(x)} L_r(r(x, \omega_i) - \omega_i) f_r(x, \omega_i \to \omega_o) cos\theta_i d\omega_i \tag{6}$$

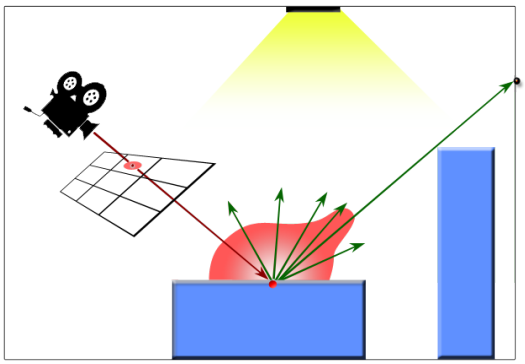
Príslušný estimátor pre vzorkovanie smeru s hustotou $p(\omega)$ ktorá je podobná BRDF

$$\hat{L}_o^{ind}(x, \omega_o) = \frac{1}{N} \sum_{k=1}^{N} \frac{L_r(r(x, \omega_{i,k}) - \omega_{i,k}) f_r(x, \omega_{i,k} \to \omega_o) cos\theta_{i,k}}{p(\omega_{i,k})} \tag{7}$$

Kde $r(x, \omega_i)$ je priesečník so scénou v smere $\omega_i$ s počiatkom v bode $x$. $-\omega_i$ je smer opačný k smeru $\omega_i$, teda $L_r(r(x, \omega_i), -\omega_i)$ je radiance *odrazená* z bodu $r(x, \omega_i)$ v smere k bodu $x$.

Algorimus *distribution raytracing* nájde príslušné body $r(x, \omega_i)$ tak, že náhodne vygeneruje $N$ smerov $\omega_i$ a nájde najbližší priesečník so scénou.

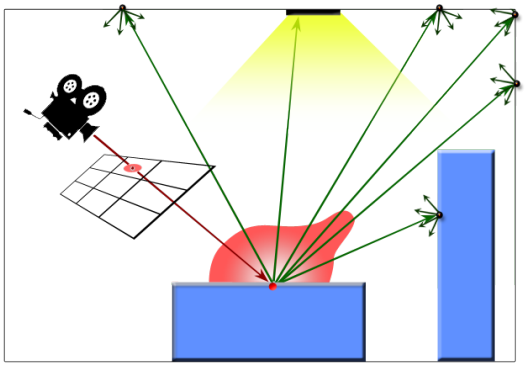
Zvoľme si $N := 6$. Vygenerujeme si 6 smerov $\omega_{i,1}$ až $\omega_{i,6}$ s hustotou pravdepodobnosti podľa BRDF. Pre každý smer vypočítame nový priesečník so scénou, odkiaľ zoberieme príspevok radiance, ktorý dopadá do bodu $x$.



**Obr. 6:** *nepriame osvetlenie*

Na obrázku č. 6 sú zelenou vyznačené smery $\omega_{i,k}$, ktoré generujeme od bodu $x$ s hustotou podľa BRDF (podľa červeného laloku). Tie pretneme so scénou a nájdeme bod $r(x, \omega_{i,k})$, nazvime ho bod $y_k$.

Aby sme vypočítali hodnotu $\hat{L}_o^{ind}(x, \omega_o)$, potrebujeme v každom bode $y_k$ vypočítať radianci $L_r(y_k, -\omega_{i,k})$, ktorá dopadá *do* bodu $x$ z $y$. Tú odhadneme presne tak, ako sme odhadovali $L_o(x, \omega_o)$ pre priame osvetlenie, plus pre bod $y$ vypočítame, ďalej nepriame osvetlenie - tiež vygenerujeme $N$ ďalších smerov a rekurzívne pokračujeme ďalej až po pevne zvolenú maximálnu hĺbku rekurzie.



**Obr. 7:** *2. level rekurzie*

Na obrázku č. 7 je znázornená 2. úroveň rekurzie. Všimneme si, že ak v každom bode vygenerujeme $N$ paprskov a nech hĺbka rekurzie je $m$, tak zložitosť algoritmu bude narastať *exponenciálne* vzhľadom k $m$.

### 2.3 Implementácia

**Algorithm 1** Distribution raytracing

```
Require: scene, pixels, camPos, N
1:  function RENDER()
2:      for each pixel in pixels do
3:          for k = 1 to N do
4:              ω_k := náhodný smer skrz pixel
5:              pixelCol += GetLi(camPos, ω_k)
6:          end for
7:      end for
8:  end function
1:  function GETLI(x, ω_i)
2:                  ▷ x je v len prvej úrovni rekurzie pozícia kamery
3:      hit := najbližší priesečník so scénou(x, ω_i)
4:      ω_o := -ω_i
5:      y := hit.pos
6:      if žiadny priesečník then return Farba Pozadia
7:      else
8:          L_o := (0, 0, 0)
9:          for k = 1 to N do
10:             ω_k := SAMPLEDIR(hit)
11:                 ▷ ω_k vygenerujeme s nejakou hustotou pdf(ω_k)
12:             L_o +=
13:             getLi(y, ω_k) * f_r(y, ω_k, ω_o) * dot(hit.n, ω_k) / pdf(ω_k)
14:         end for
15:         return L_o / N + DIRECTLIGHTING(y, ω_o)
16:     end if
17: end function
```

Ak by sme to mali celé zhrnúť, obrázok č. 4 odpovedá riadkom 4-5 v metóde RENDER() a 3-5 v metóde GETLI(). Výpočet priameho osvetlenia, ktorý je popísané v odseku 2.1 a naznačený na obrázku č. 5 je zahrnutý v metóde DIRECTLIGHTING() (riadok 15 v metóde GETLI()). Výpočet nepriameho osvetlenia, ktorý je popísaný v odseku 2.2 v implementácií odpovedá riadkom 9-14 v metóde GETLI().

### 2.4 Ukončenie rekurzie

Ako vidíme z implementácie, z každého bodu algoritmus vyberie $N$ vzorkov z ktorých následne rekurzívne pokračuje ďalej, z čoho vyplýva exponenciálna zložitosť v závislosti na hĺbke rekurzie.

Ďalší problém nastáva pri ukončení rekurzie. Jedna z možností je, že fixne obmedzíme algoritmu hĺbku rekurzie a druhá, že si nastavíme minimálny príspevok, pri ktorom sa ďalší krok vykoná. Obe možnosti nám spôsobujú, že algoritmus **nie je** nestranný.

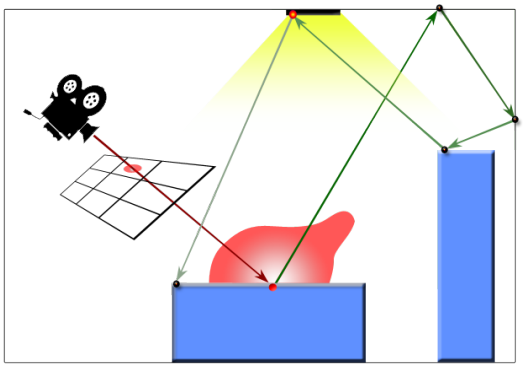
Oba tieto problémy rieši algoritmus *pathtracing*.

## 3 Pathtracing

Na rozdiel od *distributed ray-tracing*u algoritmus sleduje len jednu cestu s odrazmi až k svetelnému zdroju. Pri odrazoch si môže zvoliť z interakcií s povrchom (lesklý odraz, lom svetla) a patrične im nastavovať váhu. Od váh nakoniec závisí aj moment ukončenia rekurzie z čoho plynie nestrannosť algoritmu. Pre každý pixel vygeneruje niekoľko stoviek ciest, ktoré nakoniec spriemeruje.

### 3.1 Implementácia

Na rozdiel od *distribution raytracingu* sa rekurzia zmení



**Obr. 8:** *1 iterácia pathtracing-u pre hĺbku 6*

Metóda Render() bude vyzerať rovnako ako v prípade *ray tracingu*, líšiť sa bude v metóde GetLi(). Následujúca metóda nebude rekurzívna, nakoľko neprebehne vetvenie cesty, tak sa zmení na jednoduchý *while* cyklus.

**Algorithm 2** Pathtracing

```
1:  function GetLi(x, ω_i)
2:                  ▷ x je v len prvej úrovni rekurzie pozícia kamery
3:      Color thrput := (1, 1, 1)
4:      Color accum := (0, 0, 0)
5:      y := hit.pos
6:      while True do
7:          if žiadny priesečník then
8:              return accum + thrput * Farba Pozadia
9:          end if
10:         if hit je na svetle then
11:             accum += thrput * L_e(hit.pos, −ω_k)
12:         end if
13:         if RAND() < ρ then
14:             ▷ ρ je použitá ako pravdepodobnosť prežitia cesty
15:                 ▷ RAND() vygeneruje náhodne číslo od 0 do 1
16:             ω_k := SampleDir(hit)
17:                 ▷ ω_k vygenerujeme s nejakou hustotou pdf(ω_k)
18:             thrput *=
19:             f_r(y, ω_k, ω_o)*dot(hit.n, ω_k)/(ρ * pdf(ω_k))
20:             x := hit.pos
21:             ω_i := ω_k
22:         else
23:             break
24:         end if
25:     end while
26: end function
```

### 3.2 Ukončenie rekurzie - ruská ruleta

Ruská ruleta je nestranný algoritmus, ktorý pokračuje s pravdepodobnosťou $q$ a váhu upraví faktorom $\frac{1}{q}$, čo nám zaručuje nestrannosť.

$$E[Z] = \frac{E[Y]}{q}.q + 0.\frac{1}{q-1} = E[Y] \tag{8}$$

Za pravdepodobnosť prežitia cesty $q$ môžeme použiť čokoľvek, algoritmus bude vždy nestranný. V praxi za $q$ dosadzujeme práve odrazivosť, ako je uvedené v pseudokóde. Ak plocha odrazí napr. 30%, rekurzia skončí s pravdepodobnosťou 70% a s pravdepodobnosťou 30% bude pokračovať.

## 4 Výber náhodného smeru

V oboch algoritmoch sa nachádza metóda SampleDir($hit$) [pathtracing - 16. riadok]. Doteraz sme vedeli len, že je to nejaká metóda, ktorá vygeneruje z bodu $x$ paprsek v náhodnom smere $\omega_i$. Pre nás sú dôležité v prvom rade tie, odkiaľ je možno očakávať vysoký príspevok svetla, preto jednotlivé príspevky generujeme s nejakou hustotou, ktorá nám pomôže sa skôr dopracovať k výsledku. Ideálne by bolo, keby sme vedeli generovať vzorky s hustotou, ktorá je úmerná $L_i f_r(\omega_i, \omega_o)\cos\theta_i$, čo ale nedokážeme nakoľko nepoznáme $L_i$. Najčastejším riešením je preto generovat smer $\omega_i$ z rozdelenia, ktorého hustota je podobná členu $f(\omega_i, \omega_o)\cos\theta_i$, alebo len $f(\omega_i, \omega_o)$.

### 4.1 BRDF importance sampling

Pre *ideálne* vzorkovanie BRDF je PDF pre vygenerovaný smer $p(\omega_i)$ platí

$$p(\omega_i) = \frac{f_r(\omega_i \to \omega_o).cos\theta_i}{\int_{H(x)} f_r(\omega_i \to \omega_o).cos\theta_i dx} \tag{9}$$

Všimneme si však, ze normalizačný člen je presne rovný veličine, ktorú už poznáme: hemisfericko-smerovej odrazivosti $\rho(\omega_o)$. Rovnicu (9) môžeme prepísať ako

$$p(\omega_i) = \frac{f_r(\omega_i \to \omega_o).cos\theta_i}{\rho_{\omega_o}} \tag{10}$$

V obecnom prípade je váha (throughput) cesty pri každom odraze násobena členom (viď pseudokód path traceru 18-19. riadok):

$$thrput* = \frac{f_r(\omega_i \to \omega_o).cos\theta_i}{\rho * p(\omega_i)} \tag{11}$$

Čo ak spojíme s rovnicou (10), dostávame

$$thrput* = 1 \tag{12}$$

Pokiaľ teda použijeme „ideálne“ BRDF importance sampling, tj. ak budeme generovať smery z rozloženia s hustotou danou rovnicou (9), a pokiaľ budeme aplikovat ruskú ruletu s pravdepodobnosťou prežitia rovnou odrazivosti plochy, potom platí rovnica (12). Teda, váha cesty sa pri jej trasovani scénou nijak nemení, všetky spočítané príspevky prispievajú k výslednému pixelu rovnakou váhou. To je dobrá stratégia, pretože by sme určite nechceli investovať výpočet do vrhania paprsku, ktoré budú k pixelom prispievať s malinkou váhou - to by bolo plytvanie výpočetným časem. V praxi je často obtiažné dosiahnuť ideálneho BRDF importance samplingu s hustotou presne podľa rovnice (9), no použitie hustoty 'podobnej' BRDF zaručí, že váha cesty sa nebude príliš meniť a path tracer bude fungovať dobre.

### 4.2 Multiple importance sampling

V prednáške o Monte Carlo metódach sme si uviedli, že jedným z nejdôležitejších spôsobov zníženia rozptylu estimátorov je vzorkovanie podle dôležitosti (importance sampling). Podstatou tejto metódy je použiť pre generovanie náhodných vzorkov rozdelenie s hustotou, která blízko kopíruje tvar integrovanej funkcie.

Metóda *Monte Carlo* nám umožňuje aproximovať integrály v tvare $\int f(x)dx$. Často sa ale stretávame s integrálmi tvaru $\int f(x)g(x)dx$, pričom nemáme techniku, ktorá vypočíta *PDF* ktorá bude úmerná $f(x).g(x)$ a máme možnosť použiť vzorkovaciu techniku len podľa $f(x)$ alebo len podľa $g(x)$.

V prípade rovnice odrazu:

$$L_o(\omega_i, x) = L_e(x, \omega_o) + \int_{H(x)} L_i f_r(x, \omega_i \rightarrow \omega_o) cos\theta_i dx \quad (13)$$

Potrebujeme vypočítať integrál:

$$\int_{H(x)} L_i f_r(x, \omega_i \rightarrow \omega_o) cos\theta_i dx \quad (14)$$

Ak máme možnosť vzorkovať podľa $L_i$ alebo podľa $f_r$ sa stretneme s mnoho prípadmi, kde len jedna z týchto techník nebude stačiť, alebo bude vraciať výsledky s veľkou odchýlkou (v prípade obrazu šum).

Predstavme si prípad, kde materiál je „takmer zrkadlový“ a svetlo s veľmi veľkou plochou. Ak budeme vzorkovať podľa $L_i$, tj. vyberať náhodne z plochy svetla, je veľmi nízka pravepodobnosť, že sa trafíme práve do úzkeho intervalu smerov $U(\omega_i)$, ktorý je materiál schopný odraziť v smere $\omega_o$ s nejakou nezanedbateľnou radianciou. Ak budeme mať ešte silnejší predpoklad, a to že materiál je „dokonale zrkadlový“, budeme mať len jeden prípustný smer $\omega_i$. Ten však nami zvolená technika nikdy nevygeneruje. V týchto prípadoch bude lepšie použiť vzorkovanie podľa $f_r$.

Naopak, predpokladajme, že materiál je „dokonale difúzny“ a svetlo s veľmi malou plochou. Ak budeme vzorkovať podľa $L_i$, tj. vyberať náhodne smery z celej hemisféry, opäť dostávame veľmi nízku pravdepodobnosť, že trafíme zdroj svetla. Aj v tomto prípade môžeme mať silnejší predpoklad, a to keď svetelný zdroj bude *zanedbateľne* malý - môžeme ho zjednodušiť až na bodový svetelný zdroj. V tomto prípade technika vzorkovania nikdy nevygeneruje smer, kde sa trafí práve do bodu, kde je umiestnené svetlo, preto bude lepšie použiť techniku z prvého prípadu.

Presne tieto problémy rieši *Multiple importance sampling*. Ak sú dve distribúce $p_f$ a $p_g$ použité na aproximáciu integrálu $\int f(x)g(x)dx$, nový estimátor bude vyzerať

$$\frac{1}{n_f}\sum_{i=1}^{n_f}\frac{f(X_i)g(X_i)w_f(X_i)}{p_f(X_i)} + \frac{1}{n_g}\sum_{i=1}^{n_g}\frac{f(Y_i)g(Y_i)w_g(Y_i)}{p_g(Y_i)} \quad (15)$$

Kde $n_f$ je počet vzorkov vygenerovaných s hustotou $p_f$, $n_g$ počet vzorkov vygenerovaných s hustotou $p_g$ a nakoniec $w_f$ a $w_g$ sú špeciálne váhové funkcie zvolené tak, aby estimátor vo výsledku mal strednú hodnotu $\int f(x)g(x)dx$.

Overíme nestrannosť. Nech $w_i(x)$ je daná váhová funkcia pre zvolenú stratégiu $i$ (tj. v predchádzajúcom prípade $i \in f, g$):

$$E[F] = ... = \int \Big[\sum_{i=1}^{n} w_i(x)\Big] f(x)g(x)dx \equiv \int f(x)g(x) \quad (16)$$

Za predpokladu, ak váhové funkcie sú zvolené tak aby

$$\sum_{i=1}^{n} w_i(x) = 1 \quad (17)$$

Táto rovnosť musí platiť pre ľubovolné $x$ vo vnútri integračného oboru integrálu $\int f(x)g(x)dx$.

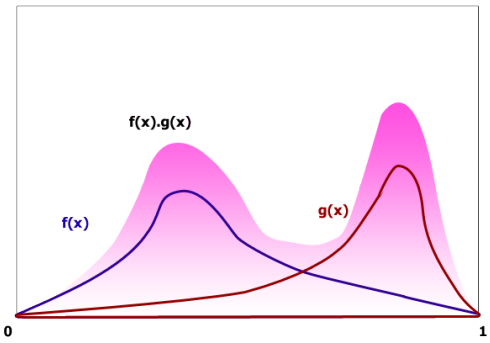
### 4.3 Vyrovnaná heuristika

Akákoľvek voľba váh, ktorá nám dáva nestranný výsledok je vhodná pre *Multiple importance sampling*. Ak využijeme aritmetický priemer, dostávame jeden z najhorších možných výsledkov, nakoľko sa dokonale prejavia nedostatky oboch techník.

Pre toto je lepšia *vyrovnaná heuristika*, kde sa zoberú do úvahy všetky ostatné techniky a ich pravdepodobnosti, ktoré by vygenerovali. Predpis pre váhy vo vyrovnanej heuristike je

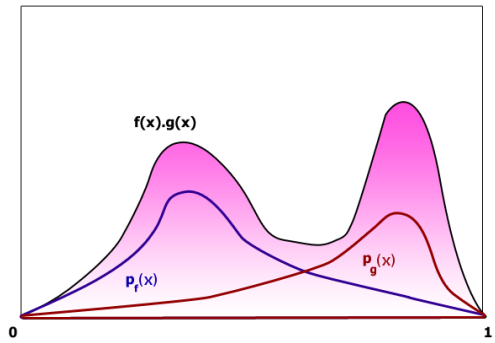
$$w_i(x) = \frac{p_i(x)}{\sum_k p_k(x)} \quad (18)$$

Predpokladajme teda funkciu $f(x)g(x)$.



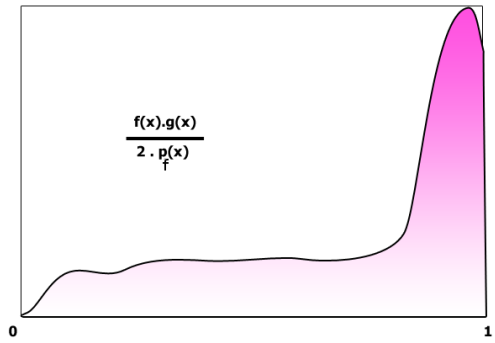

**Obr. 9:** *priebeh funkcie $f(x)g(x)$ ako súčin funkcií $f(x)$ a $g(x)$*

Ďalej máme k funkcii $f(x)$ príslušnú vzorkovaciu funkciu $p_f(x)$a k funkcii $g(x)$ vzorkovaciu funkciu $p_g(x)$

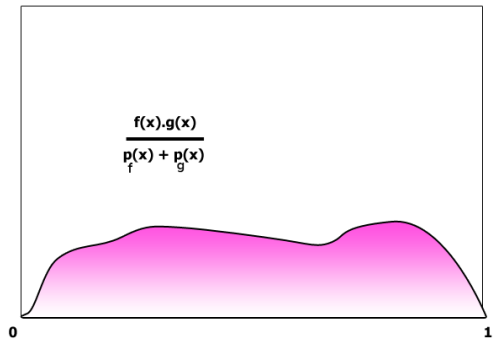



**Obr. 10:** *priebeh funkcie* $f(x)g(x)$ *vzorkovacie funkcie* $p_f(x)$ *a* $p_g(x)$ *úmerné príslušným funkciam, aby zároveň platilo* $\int p_f(x)dx = 1$ *a* $\int p_g(x)dx = 1$

Zvoľme váhu ako aritmetický priemer funkcií.




**Obr. 11:** *výsledok po použití heuristiky, ktorá využíva hodnoty z aritmetického priemeru*




**Obr. 12:** *výsledok pri použití vyrovnanej heuristiky*