

3D графика и трасиране на лъчи v.3.0



<http://raytracing-bg.net/>

Тема 11

Стереоскопия

Глобално осветление

Фундаментална формула на Каджия

(транспортно уравнение на светлината)

Основни алгоритми за глобално осветление

Съдържание

- Анонси
- Стереоскопия
 - Видове
 - Реализация на анаглифни изображения
- Глобално осветление
 - BRDF-и
 - Фундаментална формула на Каджия (транспортно уравнение на светлината)

Съдържание (2)

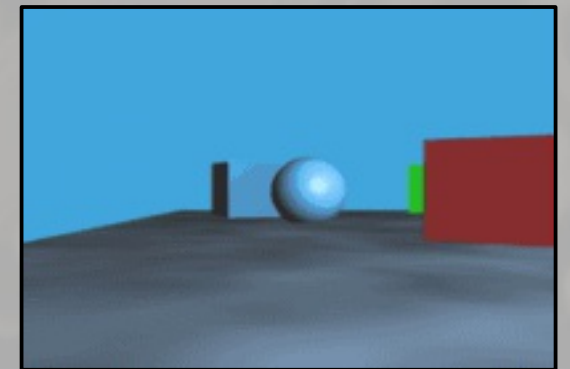
- Алгоритми за глобално осветление
 - Path tracing
 - Light tracing (накратко)
 - Bidirectional path tracing (накратко)

Анонси

- Следващата лекция е на 10^{ти} Януари
 - Весели празници :)
- Курсовите проекти са публикувани на сайта на курса
- 08.02, 22.02 – Дати за защиты
 - Подробности във форума
- 01.02, зала 02/ФМИ, 10:00 – дата за Тест 1 и Тест 2 за тези, които не са го правили през семестъра

Стереоскопия

- Стереоскопията е техника, с която можем да придадем усещане за обем в някакво изображение, като показваме различни картинки на двете очи
 - По подобие на това, което се случва в реалния свят
 - Паралакс ефект – при две различни гледни точки, по-близките обекти изглеждат сякаш са се изместили повече от по-далечните



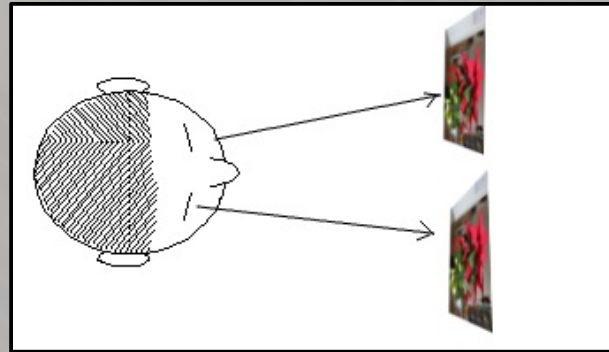
Стереоскопия

- Технически, стереоскопичните изображения са просто двойка кадри, единият от които е предназначен за лявото око, другият за дясното
- Разнообразие от начини за създаване на стереоскопични двойки:
 - Комбиниране на двойки снимки
 - Специални фотоапарати
 - Компютърно-генерирани изображения (тук се намесваме ние)



Стереоскопия

- Разглеждане на стереоскопични изображения
 - Още по-голямо разнообразие
 - Ще се спрем на част от методите тук
 - Основната идея на всички методи е: единият кадър да се вижда само от лявото око, другият – само от дясното



Двойка екрани (комплект)

- Представяват малки екрани, по един за всяко око
 - Недостатъци:
 - Изисква специализиран софтуер и хардуер
 - Относително скъпи
 - Ниска разделителна способност



LCD затъмнителни очила (shutter glasses)

- Пред всяко око има един LCD елемент, който може да закрие плътно гледката на окото
- CRT монитор показва редуващи се кадри (ляв-десен-ляв-десен-...). Затъмняването на очилата се редува по подобен начин и е синхронно с монитора.
- Така всяко око вижда само „полагащите“ му се кадри

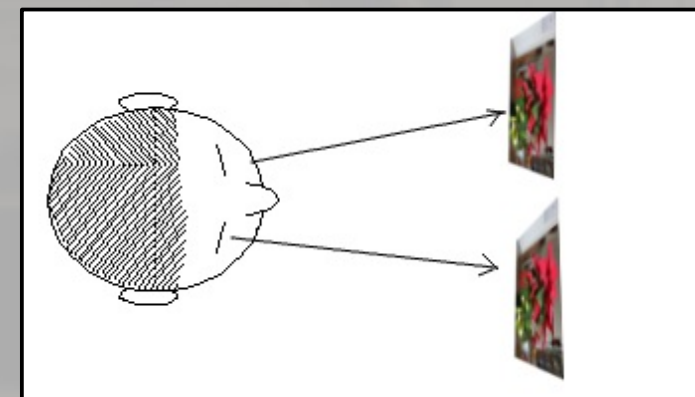


LCD затъмнителни очила (shutter glasses)

- Необходим е драйвер, който да реализира бързото сменяне на кадрите на екрана (на всяко опресняване) и да синхронизира затъмняванията на очилата
- Предимства
 - Относително евтино
 - Добро цвето предаване, добра разделителна способност
- Недостатъци
 - Трептене, монитора трябва да опреснява поне на 120 херца
 - Не работи с TFT монитори
 - Остатъчно изображение върху фосфора (ghosting)

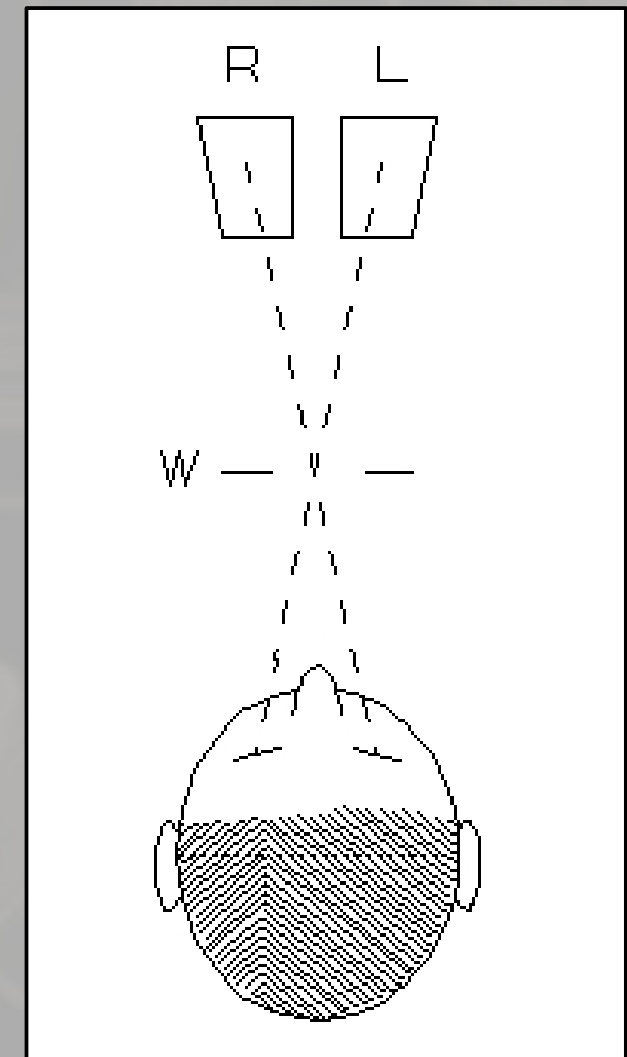
„Успореден“ метод

- Пред очите на човека се слагат две малки картинки на разстояние не повече от 65 мм една от друга
- Гледат се отблизо, очите трябва да гледат успоредно (в безкрайност), а фокусът трябва да е върху картинките
 - Това е трудната част; изисква малко тренировка



„Кръстосан“ метод

- Картинката за дясното око се поставя отляво, лявата – отдясно
- Погледът трябва да се кръстоса (сякаш гледаме нещо наблизо), а фокуса да е върху картинките (трудната част)
 - Човек трябва да „отдели“ функциите си за „следене“ и „фокусиране“ на очите
- По-добър метод от успоредния – позволява картинките да са много по-големи (и по-детайлни)



Примерна „кръстосана“ снимка



(картинките се различават много малко, но човешкото око усеща разликите много добре)

„Кръстосан“ метод

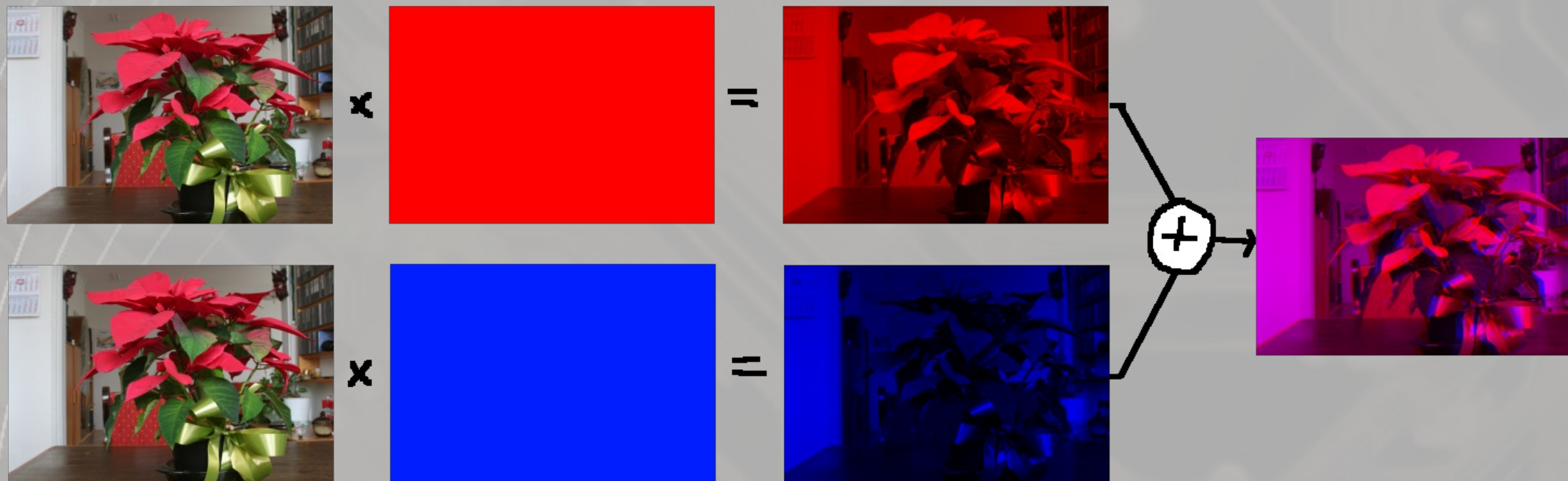
- Предимства
 - Не изисква никаква техника
 - Пълно качество: и откъм разделителна способност, и откъм цветовете
- Недостатъци
 - Изисква научаване, което за някои хора е доста трудно
 - Кара мускулите на очите да се напрягат по необичайни за тях начини

Анаглифни очила

- Идеята: всеки от двата кадъра да мине предварително през цветен филтър (например, червен или син).
Резултатите се смесват в един кадър.
Разглеждането става чрез очила със същия цвят филтри. Всеки филтър пропуска само „съответстващия“ си кадър, и поглъща другия (светлината, минала веднъж през син филтър, не минава през червен).



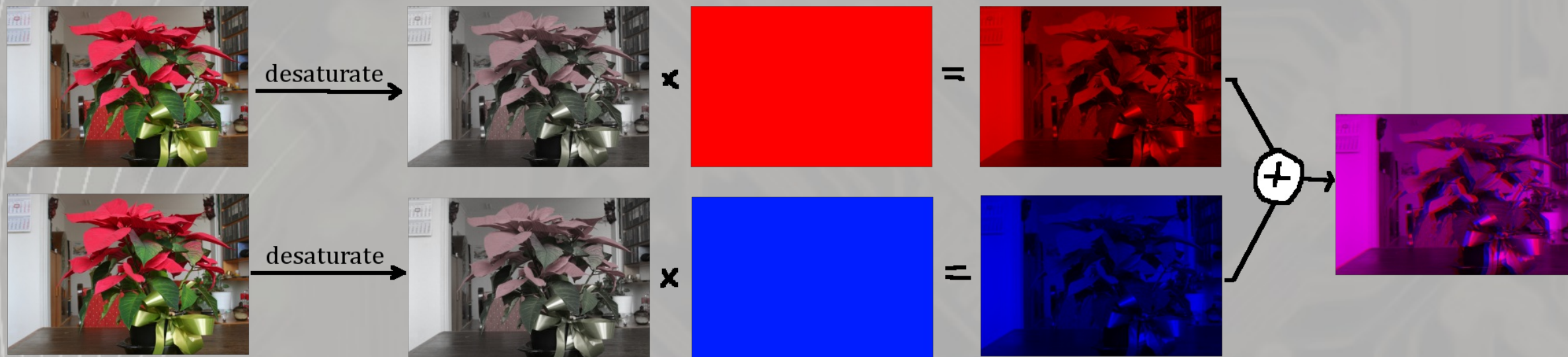
Анаглифни очила



При картинки с наситени цветове (като примерната), обаче, филтрирането отрязва доста информация (например, червените листа не се виждат в десния кадър)

Анаглифни очила

- Оказва се благоприятно да „убием“ малко цветовете на входните изображения (desaturation):

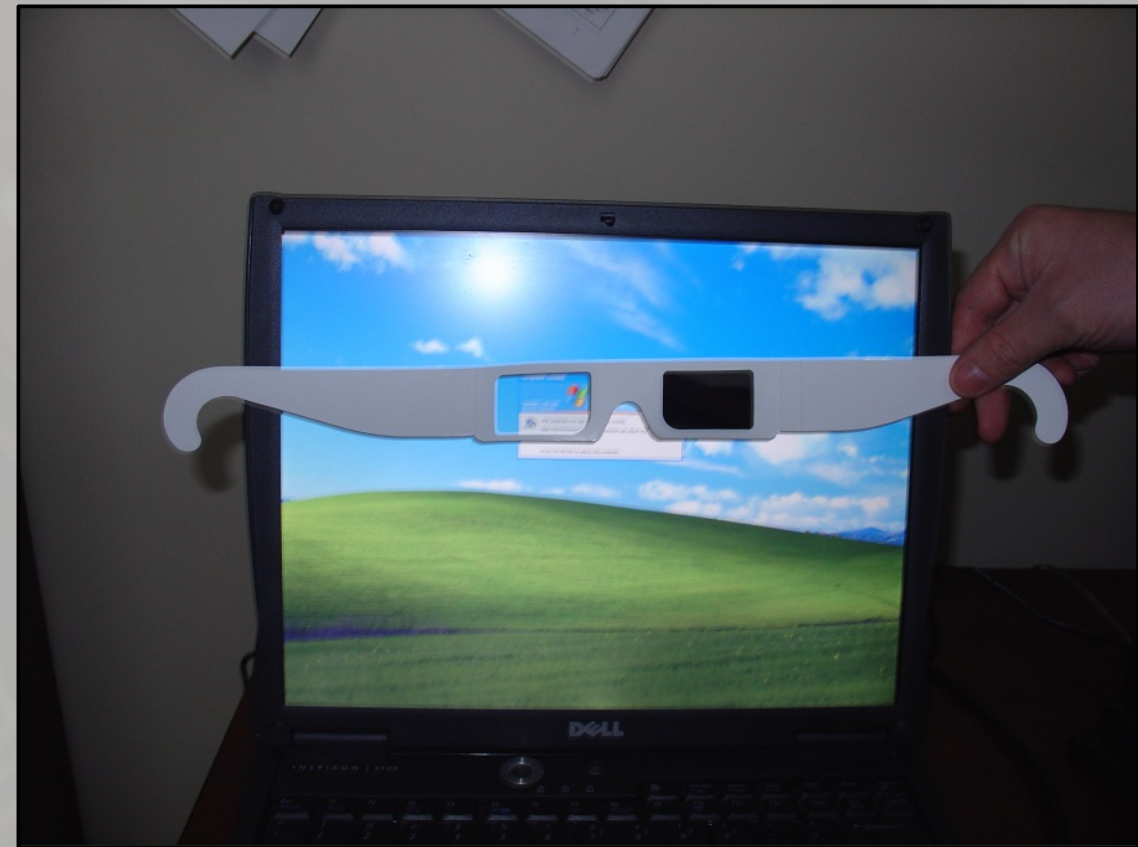


Анаглифни очила

- Освен червено-сини, често ползвани са червено-циан, жълто-циан, червено-зелени ...
- Недостатъци
 - Губи се много от усещането за цветовете. В повечето случаи, картинките изглеждат все едно са били черно-бели
 - Ако филтрите не са перфектни, може част от неправилния кадър да прозира (ghosting)

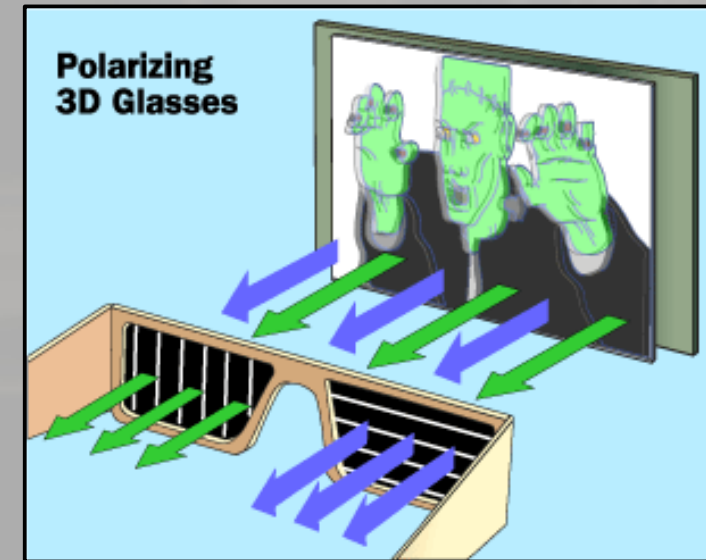
Очила с поляризационни филтри

- Използват се свойството „поляризация“ на светлината
 - Съществуват филтри, които пропускат светлина само с определена поляризация (например хоризонтална или вертикална)
 - Очилата са с два различни типа филтри пред всяко око



Очила с поляризационни филтри

- Кадърът се смесва от два прожектора, пред всеки от които също са поставени разнородни поляризирани филтри
- Тъй като всеки филтър блокира светлината с несъвпадаща поляризация, то лявото око вижда само светлината от „левия“ прожектор, а дясното – само от „десния“



Очила с поляризационни филтри

- Предимства
 - Не се губят цветове или разделителна способност
- Недостатъци
 - Ghosting
 - Заради недобри филтри
 - Ако главата на човек не е точно изправена. Дори при леко завъртане на главата, филтрите почват да пропускат и от другия кадър
 - Това може да се поправи, ако филтрите са с кръгова (вместо линейна) поляризация
 - Изисква или два проектора, или сложна технология за смяна на поляризацията на един

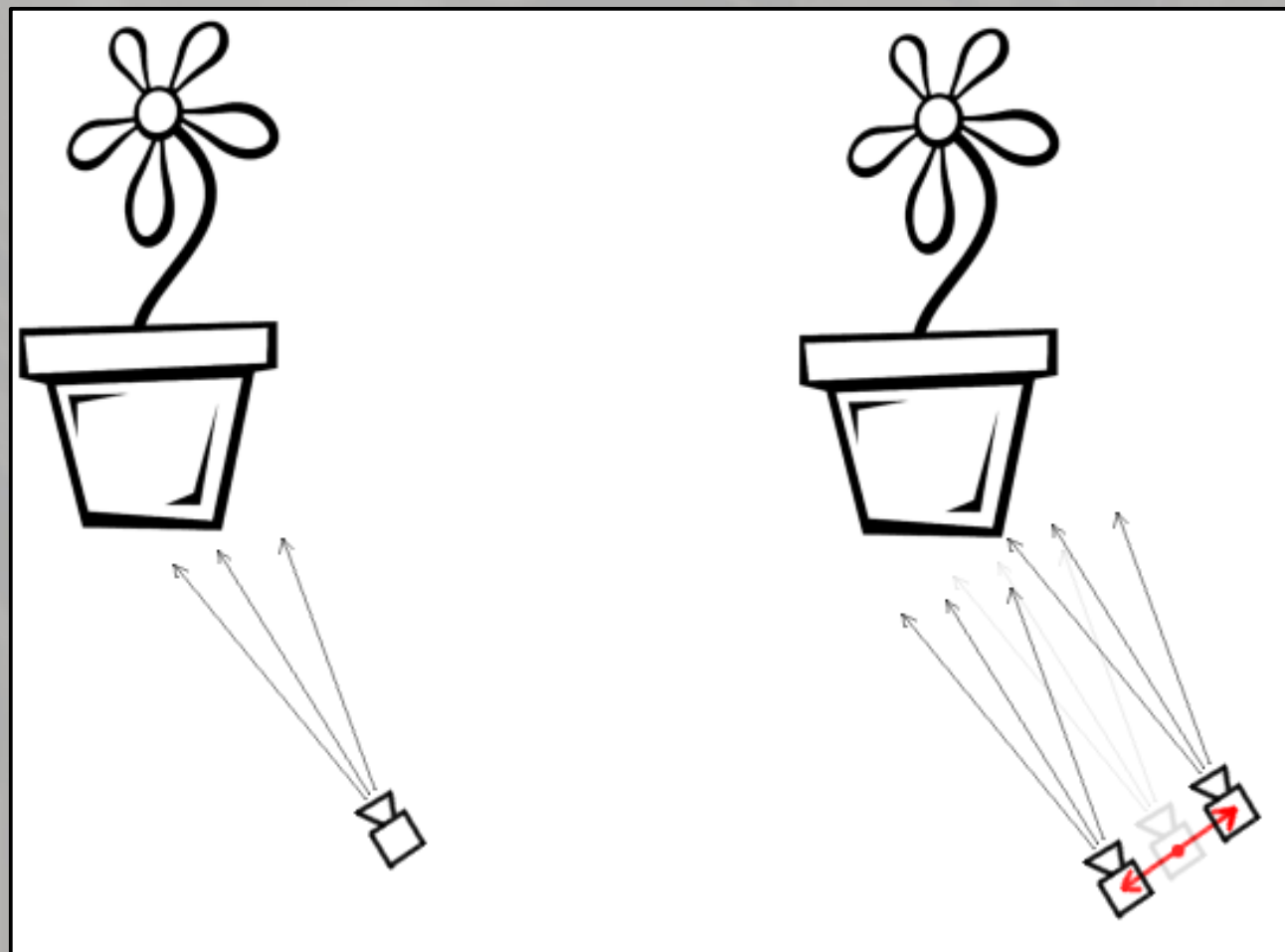
Спектрално-разделени очила

- Същата идея като при поляризиращите филтри, само че филтрите са спектрални – пропускат само различни (непресичащи се) части от спектъра
 - За разлика от поляризиращите филтри, няма проблеми при накланянето на главата
- Човек не усеща „нарязаността“ на спектъра; ако дразненето на R, G и B рецепторите му правилно, ще приеме изображението за пълноцветно

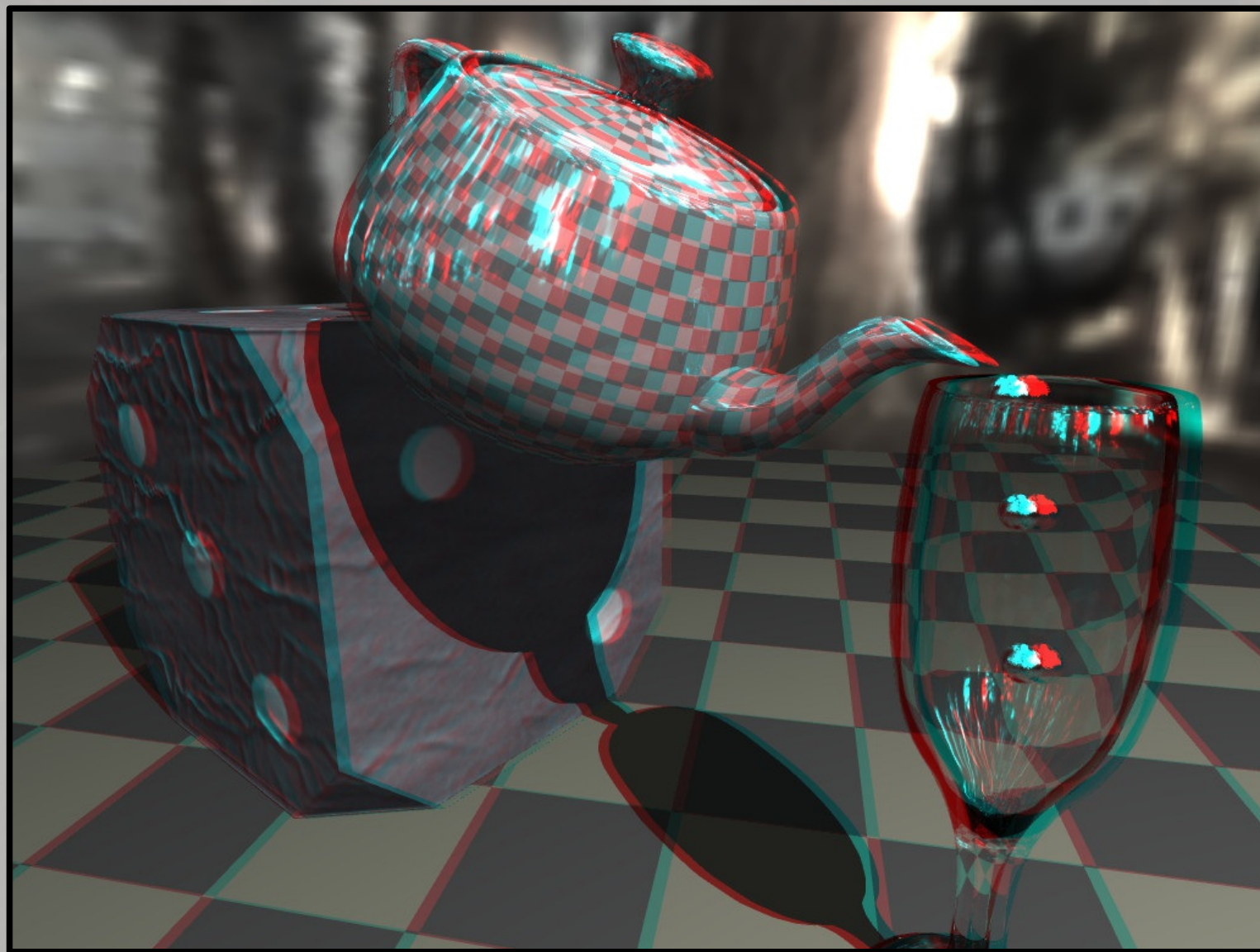


Стереоскопично рендериране

- Рендерираме два кадъра от две гледни точки, раздалечени по посоката `rightDir` на камерата
- За анаглифен образ, смесваме двата кадъра с подходящо маскиране



Результат



Стереоскопично рендериране

- Разделението между двете виртуални камери – `stereoSeparation` – е съществен параметър; то определя колко „голям“ е наблюдателя в сравнение със сцената
 - По-голям `stereoSeparation` допринася за по-силно усещане за обем. Възможно е да преувеличим с цел артистичен ефект
 - При прекалено голям `stereoSeparation`, картината изглежда неестествена и „сбъркана“
 - При прекалено малък `stereoSeparation`, стерео-ефекта се губи
 - Точното му калибриране е индивидуално за всяка сцена



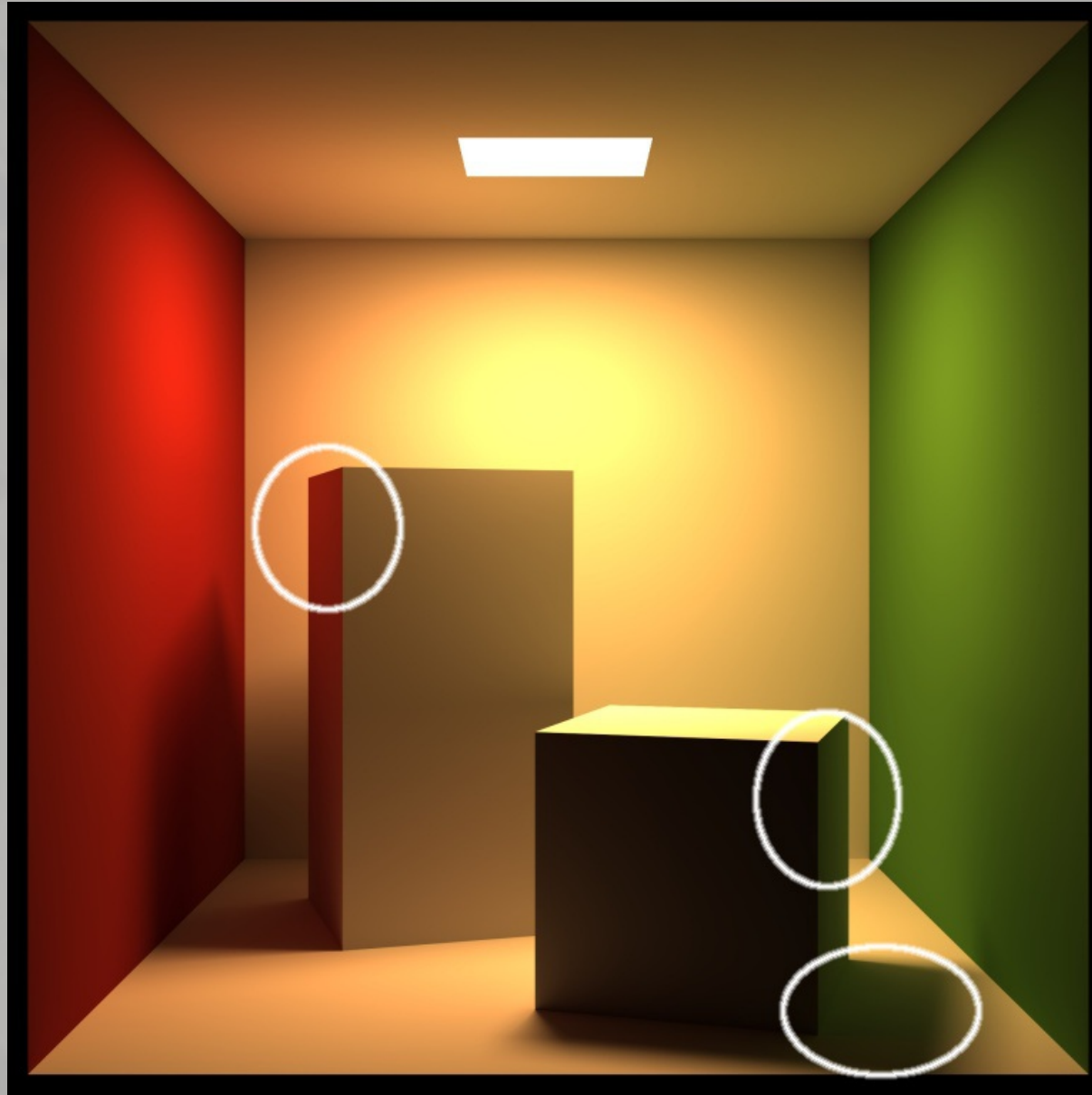
Глобално осветление

Глобално осветление

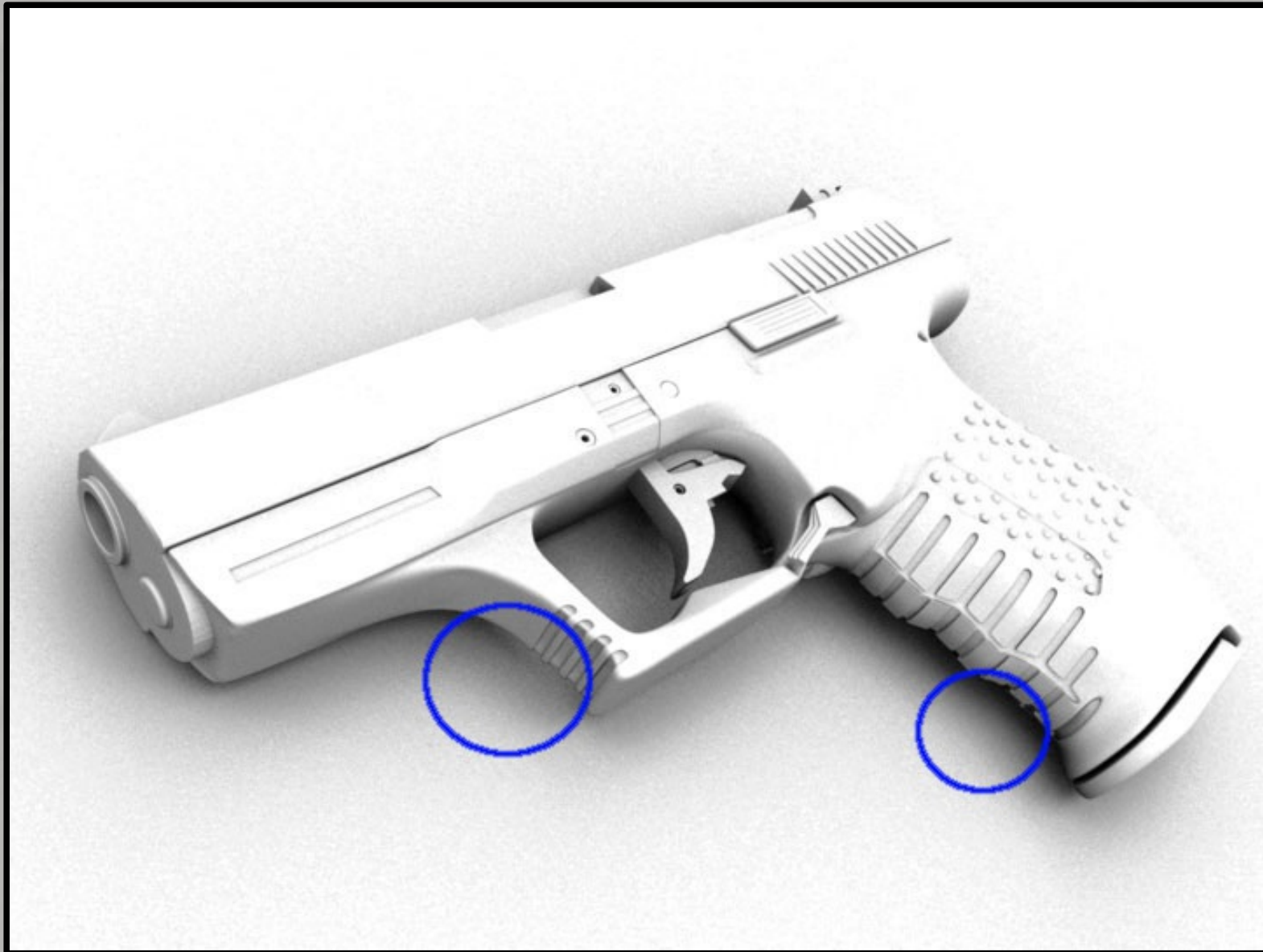
Глобално осветление = директно осветление + индиректно осветление

- Досега апроксимирахме индиректното осветление чрез ambient light – но това е много неточен подход
- Има редица ефекти, които не можем да пресъздадем само чрез директно осветление

Color bleeding



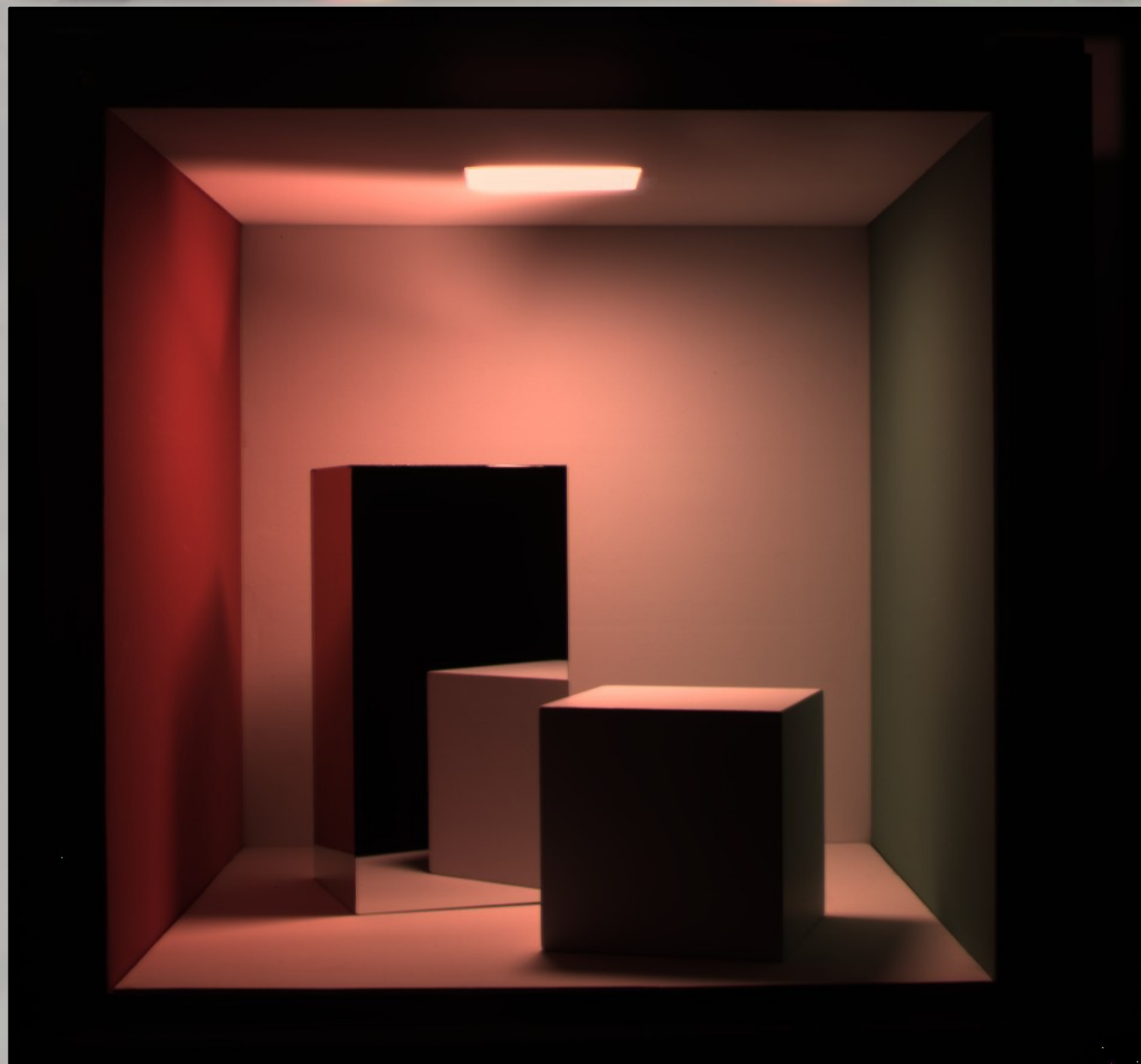
Ambient occlusion



Indirect illumination



The Cornell box

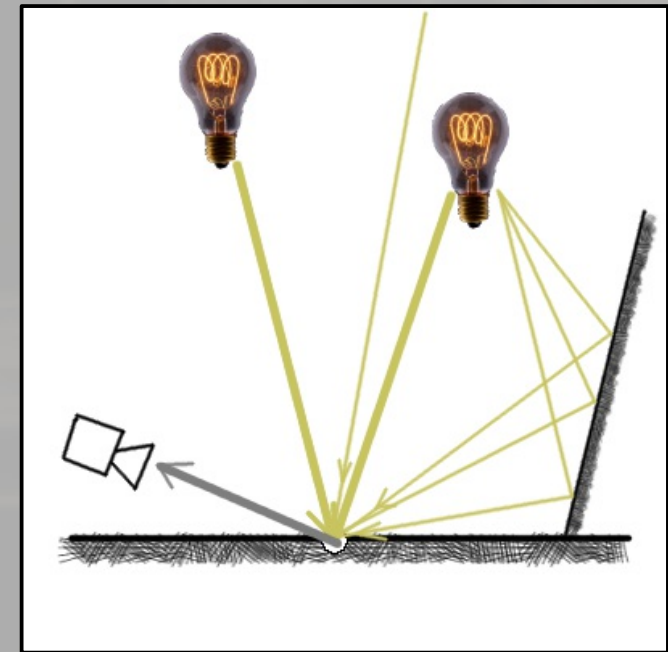


The Cornell box

- Това е стандартна тестова сцена, която са си сътворили от университета Cornell през 80-те за тестване на рендериращи алгоритми (вкл. GI такива)
- Действителен модел на сцената е заснет с фотоапарат; после се опитват да го пресъздадат компютърно
 - Cornell box не е предизвикателство за модерните GI алгоритми

Глобално осветление

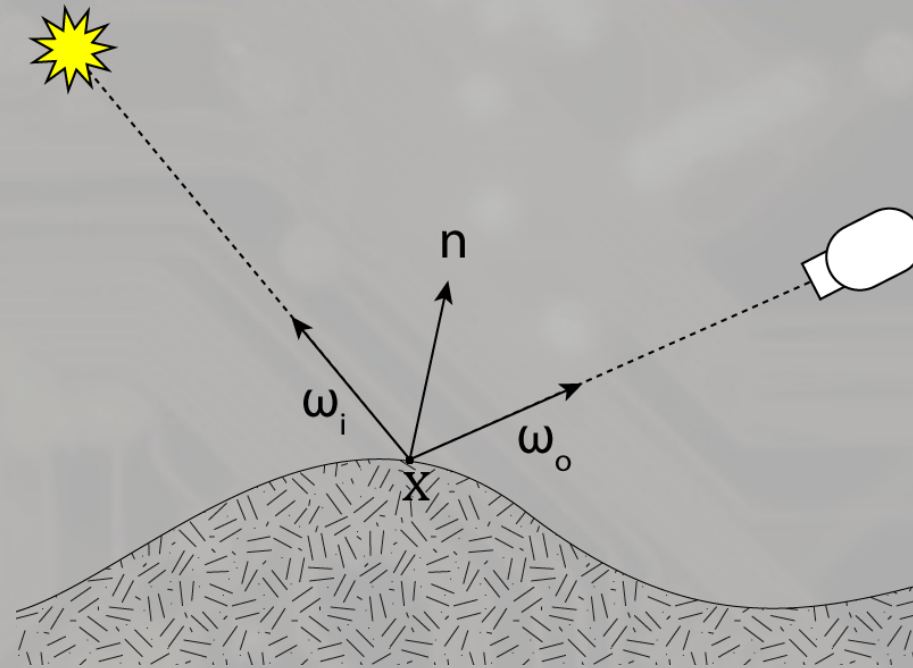
- Най-общо казано, за да пресмятаме глобалното осветление в дадена точка, трябва да:
 - Пресметнем светлината, идваща от цялата сцена, в тази точка (irradiance)
 - В зависимост от отражателните свойства на повърхността, да сметнем каква част от входната светлина се отразява по посока камерата
 - Тук ни трябва някакъв начин да моделираме отражателните свойства на повърхността (шейдърите не са подходящи)



Шейдъри vs BRDF-и

- Шейдърите не са подходящи за моделиране на отражателните свойства на повърхността
 - Те по-скоро задават алгоритъм, който описва какви други лъчи трябва да пуснем от точката, за да пресметнем осветлението; отражателните свойства са „вградени“ вътре в този алгоритъм, и са „недостъпни“ отвън
 - Иначе казано, пресмятането чрез шейдъри е ефективно, но не достатъчно общо, за да се ползва в GI алгоритмите
 - Необходимо е да използваме отделна функция, която моделира именно отражателните свойства.

Функция на отраженията (BRDF)



Bidirectional Reflectance Distribution Function или f_r :

$f_r(x, \omega_i, \omega_o) :=$ „каква е вероятността, лъч светлина, идващ от ω_i в точката x , да се отрази в посоката ω_o “

Примери за BRDF

- Дифузен BRDF (Lambertian), за бяла повърхност:

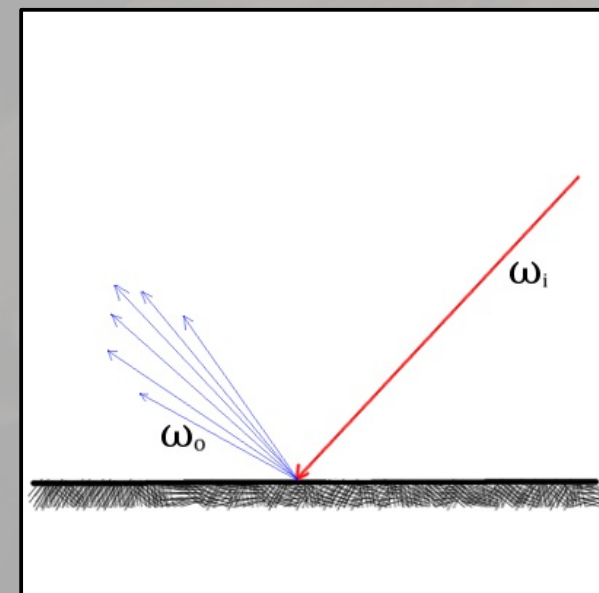
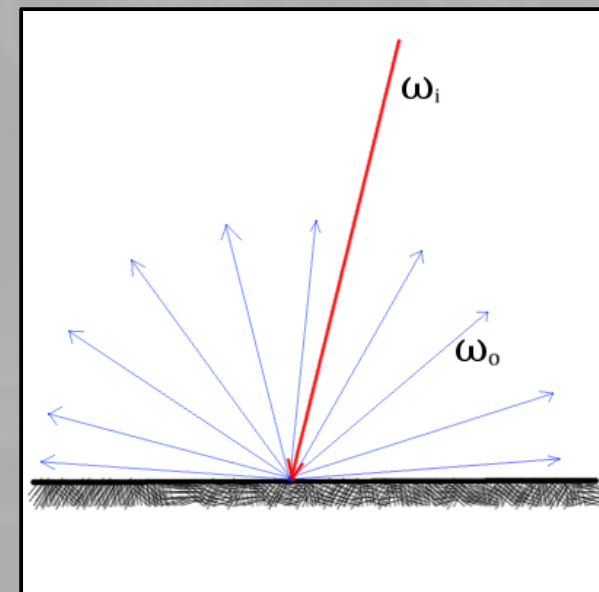
$$f_r(x, \omega_i, \omega_o) := 1/\pi$$

(ще обясним защо $1/\pi$ след малко)

- Glossy BRDF (за грапави отражения):

$$f_r(x, \omega_i, \omega_o) := S(n) * \max(0, \cos(\angle(\text{reflect}(\omega_i), \omega_o)))^n$$

(където $S(n)$ е нормираща константа, целяща запазване на енергията)

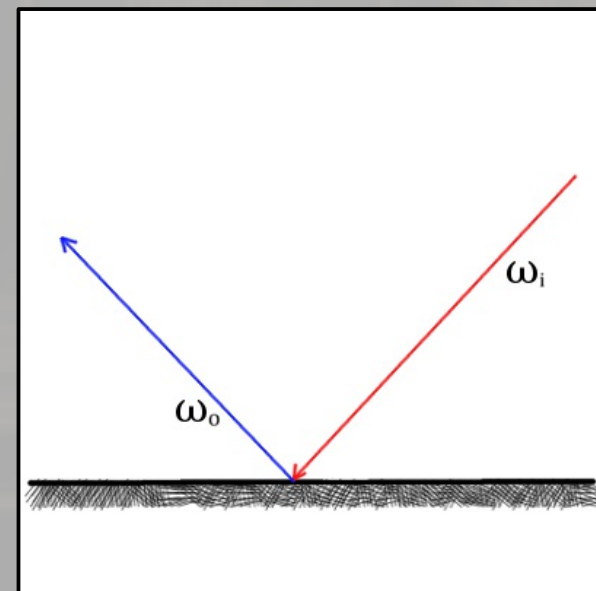


Примери за BRDF

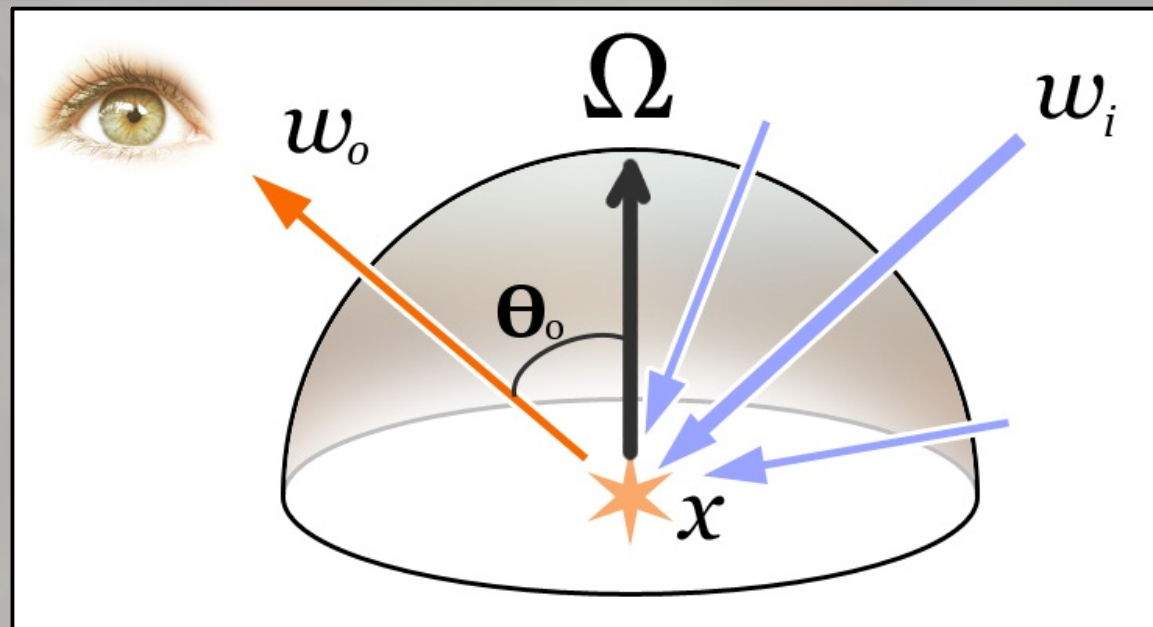
- Чисто отражение:

$$f_r(x, \omega_i, \omega_o) := \begin{cases} \infty, & \text{ако } \omega_o = \text{reflect}(\omega_i) \\ 0, & \text{иначе} \end{cases}$$

- Т.е., BRDF-ът може и да е прекъснатата функция



Свойства на BRDF-ите



- Запазване на енергията: $\int_{\Omega} f_r(x, \omega_i, \omega_o) \cos(\Theta_o) d\omega_o \leq 1$
 - Това гарантира, че не се „създава“ светлина
 - Заради това изискване имаме $1/\pi$ и $S(n)$ нормировките
- Симетричност: $f_r(x, \omega_i, \omega_o) = f_r(x, \omega_o, \omega_i)$

BRDFs: малко по-сложно

- Казахме, че дифузния BRDF е $1/\pi$ навсякъде, но това е само при бял обект без текстура. Ако има (дифузна) текстура, то връщаният резултат зависи от точката x :

$$f_r(x, \omega_i, \omega_o) := \text{texture.getTexColor}(x)$$

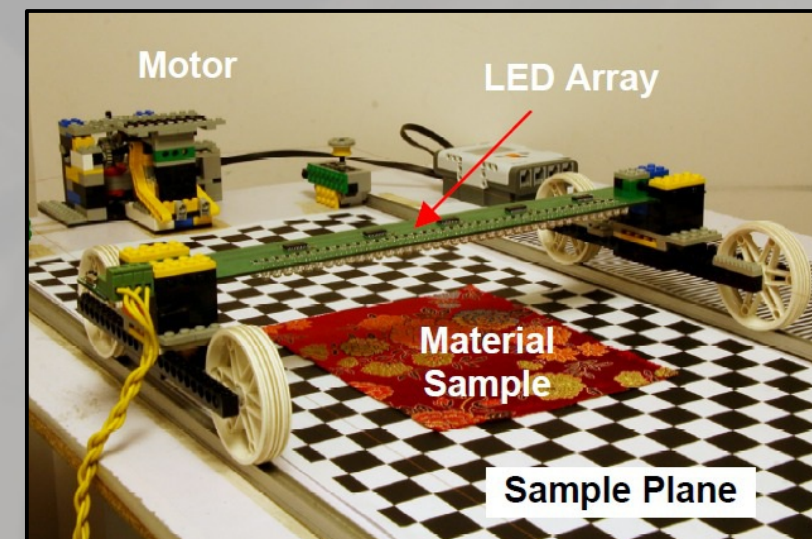
- Подобно и за останалите BRDF-и, които споменахме. Например, `reflect()` функцията изисква нормал, около който да отразява, а той зависи от точката x .

BRDFs: още по-сложно

- В някои статии, BRDF-ите имат и четвърти параметър – дължина на вълната. Това е полезно, ако се опитваме да реализираме BRDF на хроматично пречупване
 - Ние няма да го реализираме, но е хубаво да се знае

Непроцедурни BRDF-и

- В някои случаи, свойствата на BRDF-а може да не се изчисляват чрез формула, а да се взимат директно от текстура (или някакъв друг начин за справка с предварително известни данни)
- Съществуват устройства, които „сканират“ даден материал и извличат BRDF-то му [041-wang-video]
 - Вижте също talk-а от CG²2013



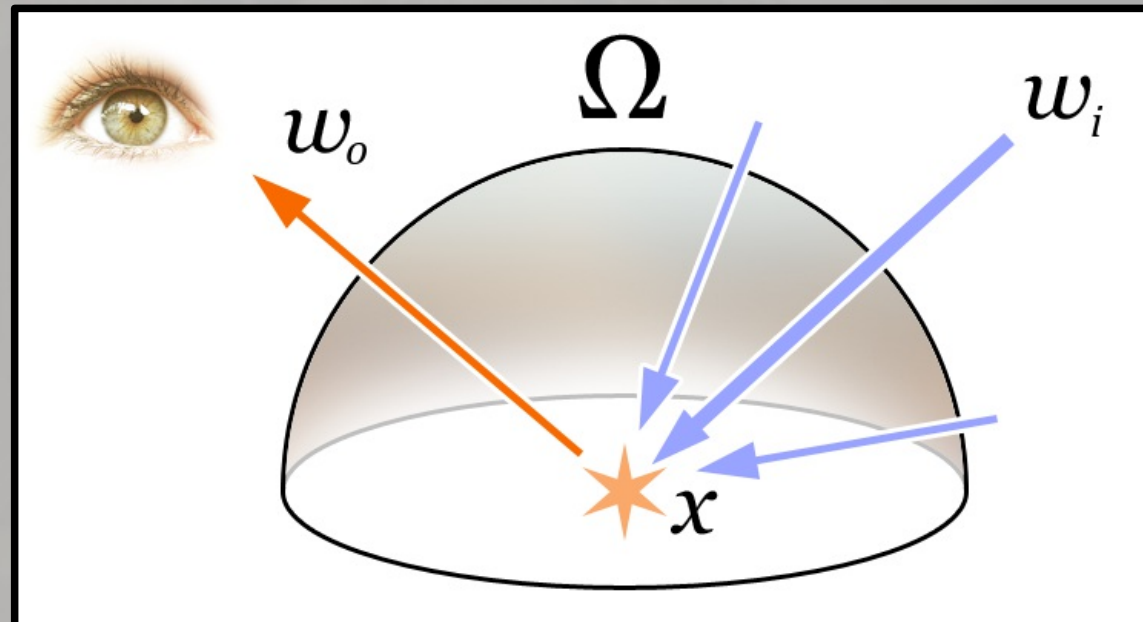
„Фундаменталната формула“ на Каджия

- Jim Kajiya през 1986 представя своя основополагащ труд в областта на глобалното осветление - „The Rendering Equation“ [SIGGRAPH-1986]
- Самата „фундаментална формула“ представлява транспортно уравнение на светлината, което ни дава израз за пресмятане на глобалното осветление във всяка точка от сцената



The Rendering Equation

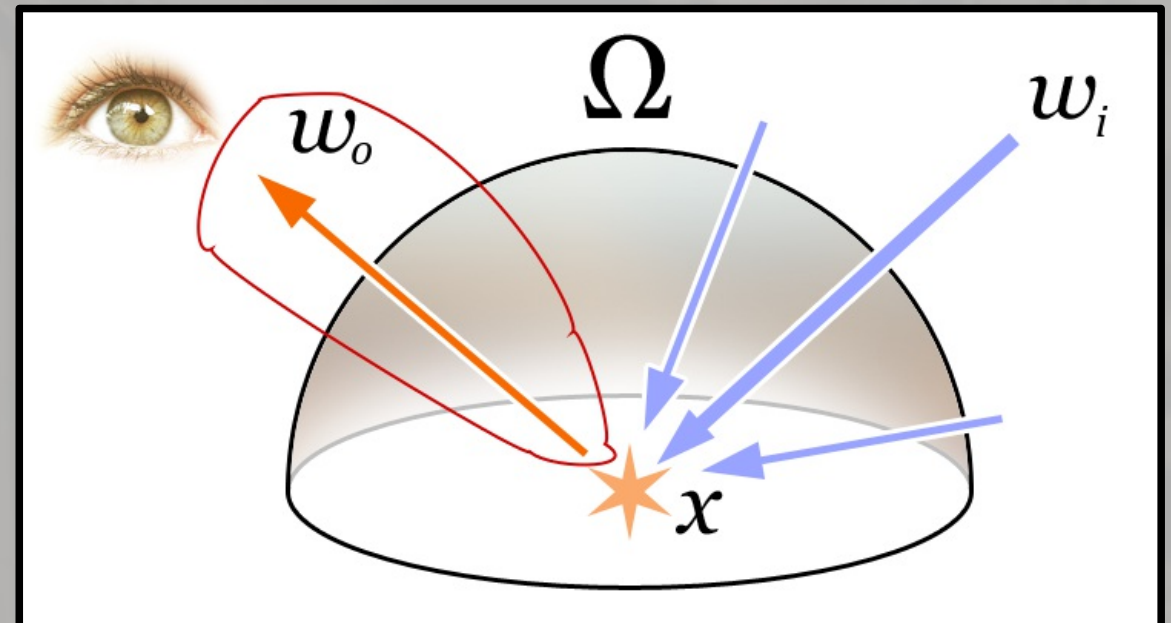
$$\mathbf{L}_o(\mathbf{x}, \omega_o) = \mathbf{L}_e(\mathbf{x}, \omega_o) + \int_{\Omega} \mathbf{f}_r(\mathbf{x}, \omega_i, \omega_o) \mathbf{L}_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$



The Rendering Equation

$$\mathbf{L}_o(\mathbf{x}, \omega_o) = \mathbf{L}_e(\mathbf{x}, \omega_o) + \int_{\Omega} \mathbf{f}_r(\mathbf{x}, \omega_i, \omega_o) \mathbf{L}_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

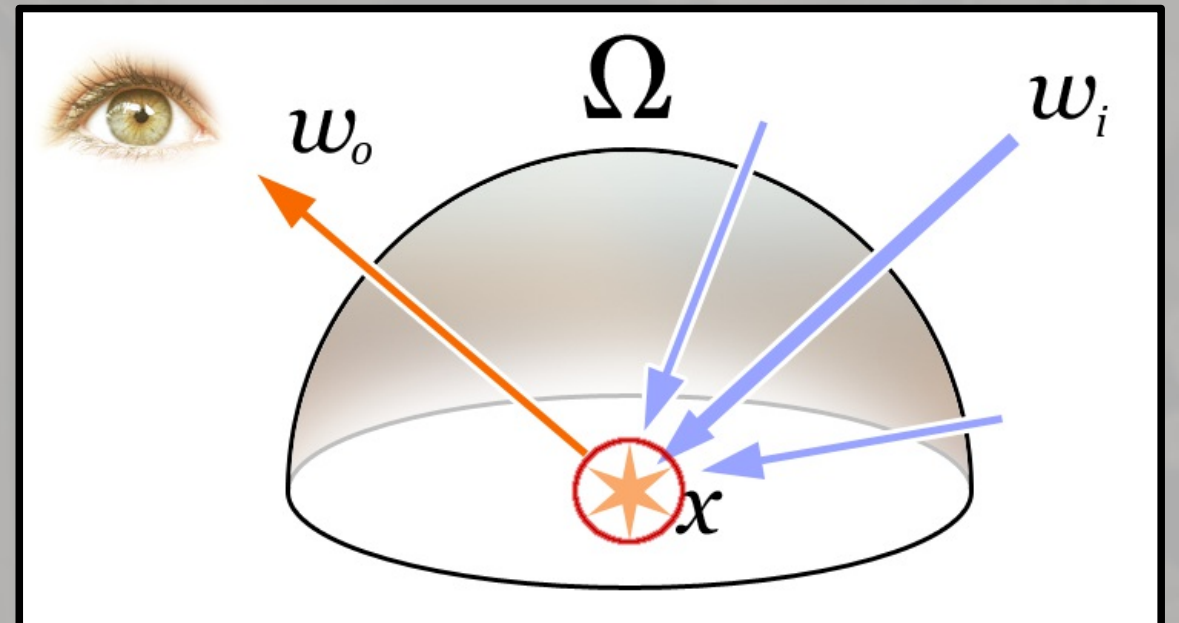
- $L_o(x, \omega_o)$ показва отразената светлина от точката x в посока ω_o
Например, към камерата:
- Ако от камерата сме пуснали лъч $-\omega_o$, който е пресякъл геометрията в точка x .



The Rendering Equation

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

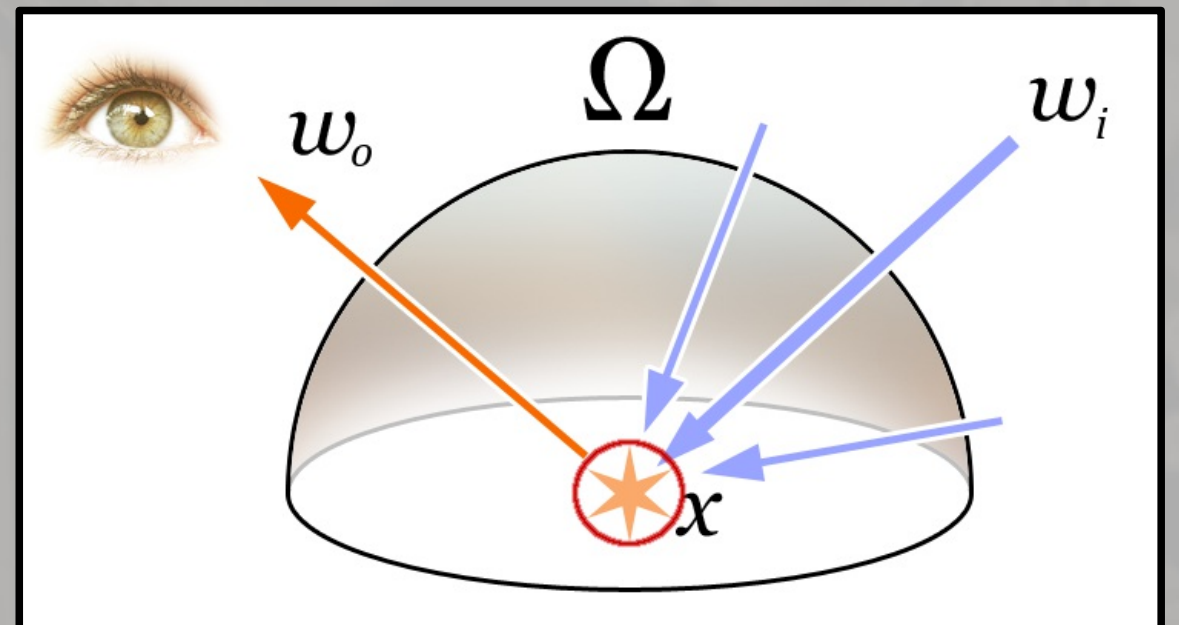
- Ако x е част от лампа, $L_e(x, \omega_o)$ показва излъчваната светлина от точката x в посока ω_o
- 0 за всички неизлъчващи обекти.



The Rendering Equation

$$\mathbf{L}_o(\mathbf{x}, \omega_o) = \mathbf{L}_e(\mathbf{x}, \omega_o) + \int_{\Omega} \mathbf{f}_r(\mathbf{x}, \omega_i, \omega_o) \mathbf{L}_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

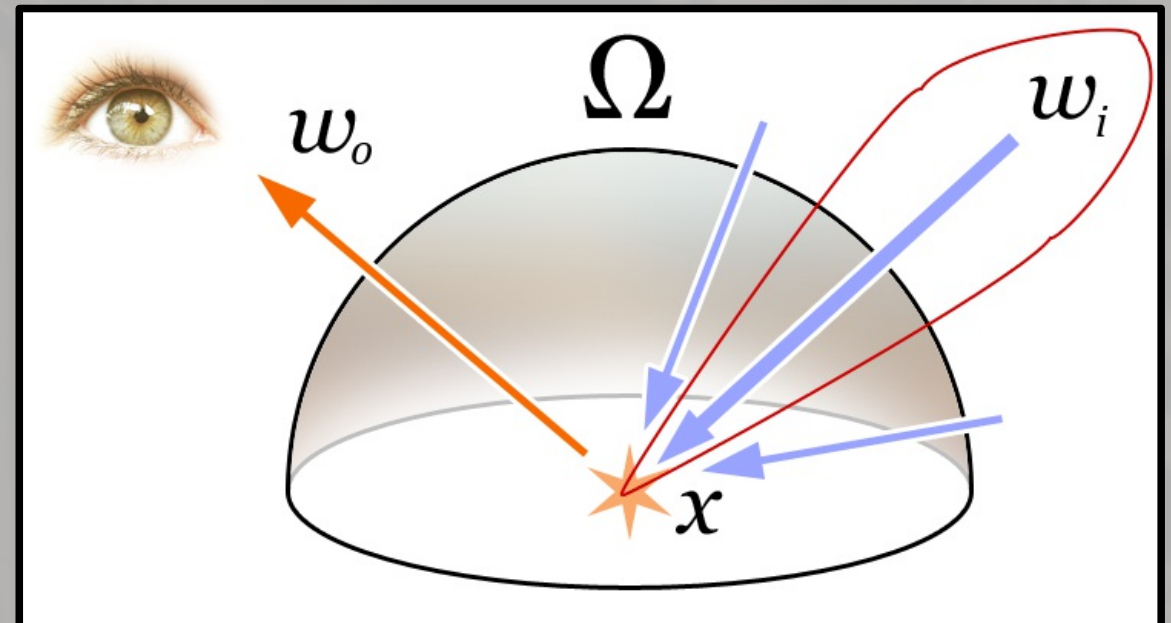
- $f_r(x, \omega_i, \omega_o)$ е BRDF-ът на материала в точката x



The Rendering Equation

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

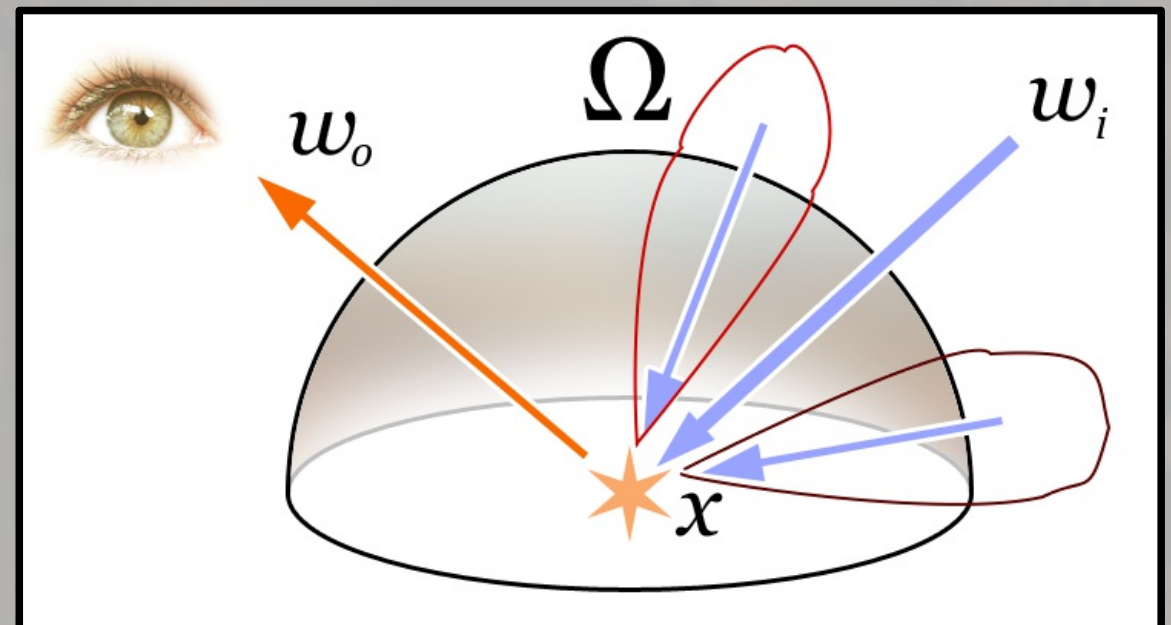
- $L_i(\mathbf{x}, \omega_i)$ показва входящата светлина в точката \mathbf{x} по някое направление ω_i . Тя може да идва както от лампа, така и от неизлъчващ обект (чрез отражение на светлината, т.е. от друг екземпляр на транспортното уравнение)



The Rendering Equation

$$\mathbf{L}_o(\mathbf{x}, \omega_o) = \mathbf{L}_e(\mathbf{x}, \omega_o) + \int_{\Omega} \mathbf{f}_r(\mathbf{x}, \omega_i, \omega_o) \mathbf{L}_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

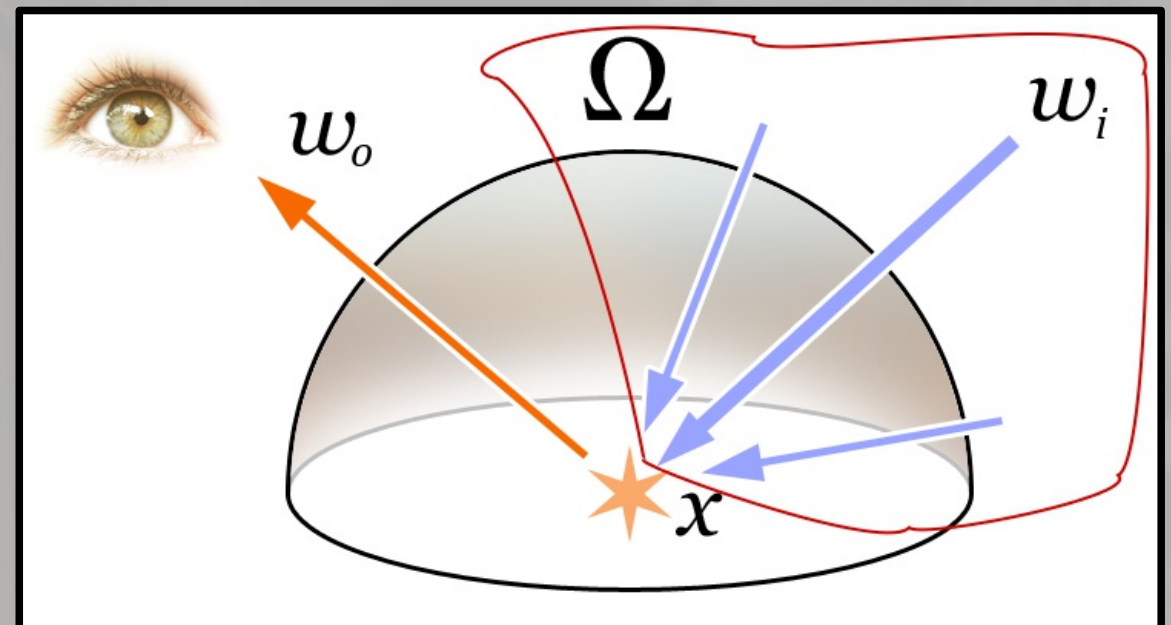
- Косинусът на ъгъла между ω_i и нормалата \mathbf{n} в точката \mathbf{x} участва, защото, под коси ъгли, входната светлина се разпределя на по-голяма площ. Така, на единица площ, осветяването е по-слабо, следователно и отражението трябва да е по-слабо.



The Rendering Equation

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

- Чрез интеграла ние сумираме цялата входяща светлина, отразена от BRDF-а в конкретната посока. Т.е. вътрешната сметка извършваме за всички входящи направления на светлината – което е цялата полусфера Ω над точката x .

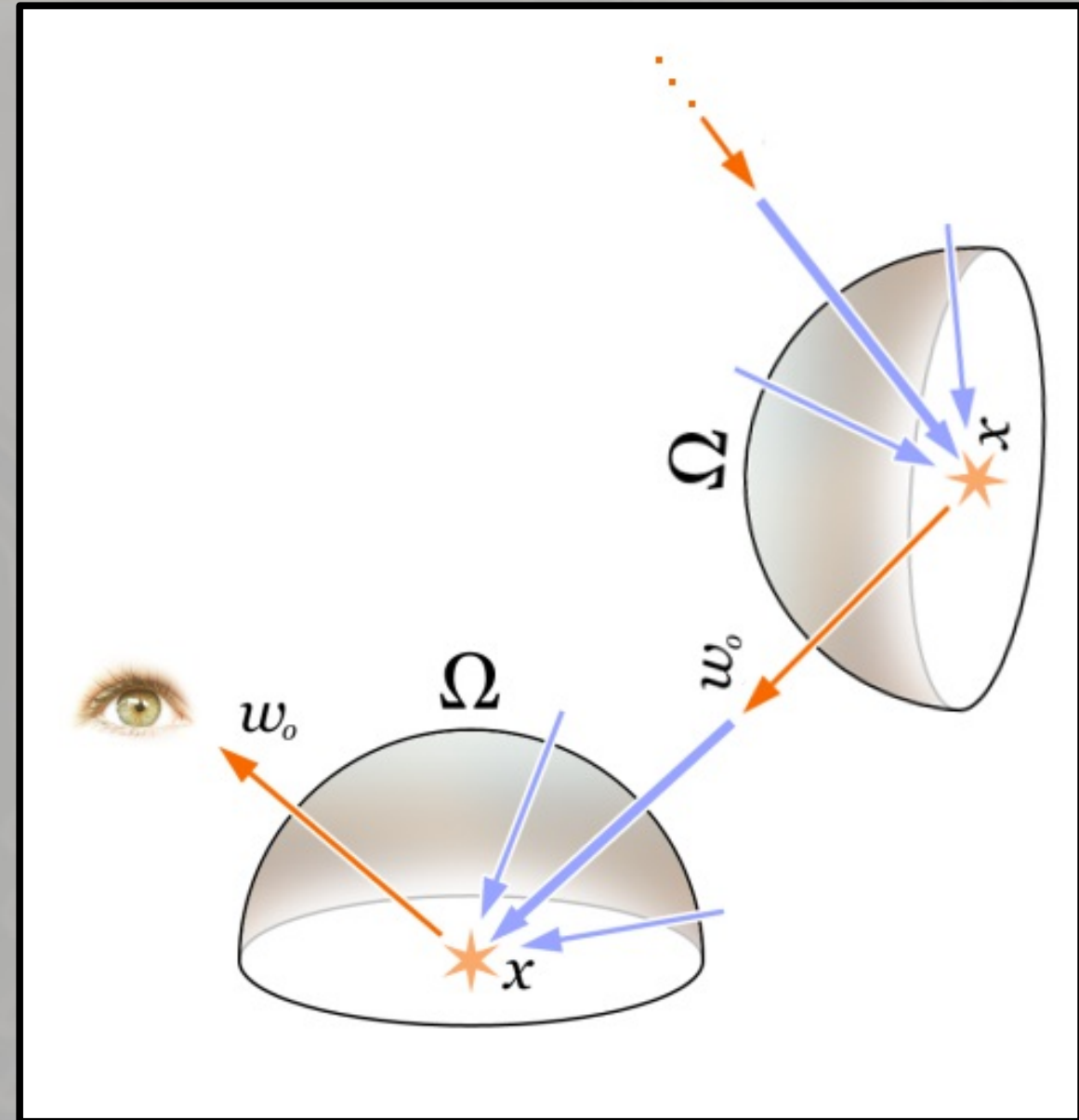


Особености на уравнението

- Уравнението е линейно – абстрахирайки се от BRDF-а, във формулата участват само събирания и умножения. Това дава възможност на практика да се правят разнообразни опростявания и апроксимации.
- Уравнението е рекурсивно – ако тръгнем да го решаваме с Monte Carlo метод, ще стигнем до друг екземпляр от уравнението, за решението на който отново ще ни трябва да решим друг екземпляр, и т.н.
- Изглежда, че всяка точка от сцената зависи от всяка друга точка от сцената!

Рекурсивност

- За да се получим повечето от GI ефектите, то обикновено трябва да преровим сцената поне няколко нива рекурсивно.
- Т.е., като от точката x пуснем случаен лъч ω_i , проследим къде се удря той в сцената (x') и после пресметнем транспортното уравнение в точката x'

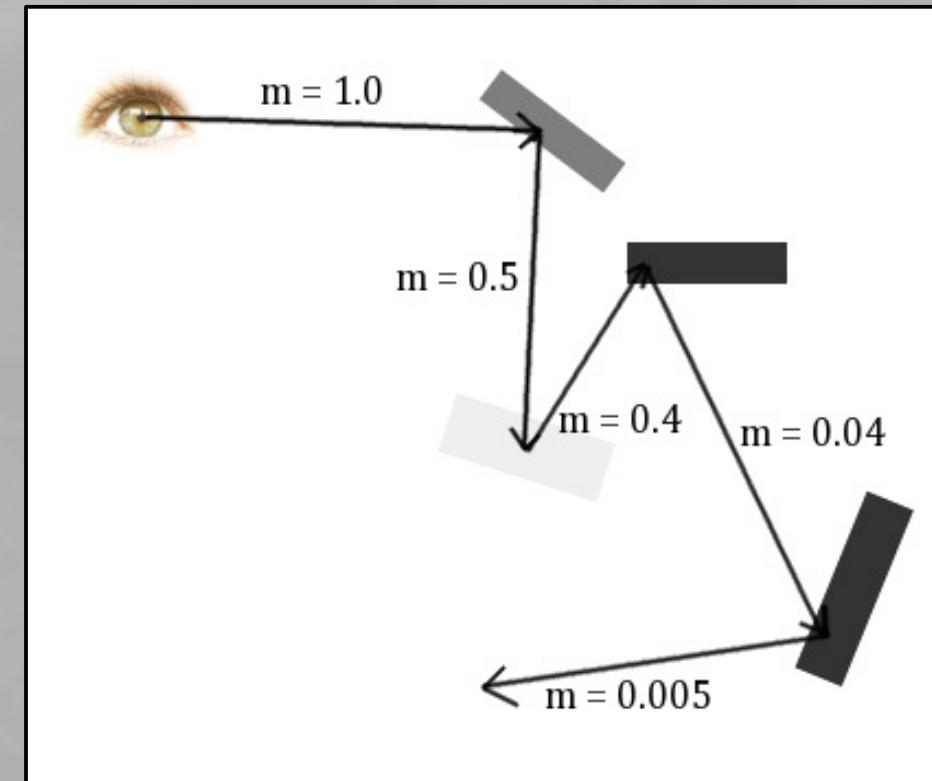


Рекурсивност

- Кога приключва рекурсията?
 - Ако ударим светлина
 - BRDF-а на светлината спокойно може да приемем за 0
 - Ако не ударим нищо (т.е. попаднем в Environment-а – тогава вземаме цвета от околната среда в тази посока)
 - Ако ударим слабо-отразяващ обект (т.е. BRDF-а е 0 или почти 0)

Рекурсивност

- Последната идея може да разширим така: може да пазим текущото произведение от всички BRDF-и по пътя на рекурсията. Ако това произведение стане почти 0, то може да прекратим рекурсията (приносът от обхождането още надолу ще е прекалено малък и може да го пренебрегнем)



Решаване чрез Monte Carlo метод

- Така зададеното уравнение може да сметнем чрез Monte Carlo метод
 - При пресмятането на интеграла, ще пуснем много случайни лъчи от точката x , ще сметнем осветлението от всеки от тях, и ще ги съберем, умножавайки по BRDF-а. Резултата ще разделим на броя на лъчите
 - Но така получаваме комбинаторна експлозия: за добро качество трябва да пускаме по много лъчи (над 20), и само няколко нива надолу вече трябва да трасираме милиони лъчи (за единствен пиксел от екрана!)

Решаване чрез Monte Carlo метод

- Можем да пускаме по много лъчи само на най-горното ниво на рекурсията, а по-долните да смятаме само с по няколко лъча; въпреки това, този метод не се справя добре и е много бавен
- По-добър алгоритъм предлага самият Каджия, в същата статия. Алгоритъмът се нарича Path tracing.

Rendering Equation (yet again =-)

$$\mathbf{L}_o(\mathbf{x}, \omega_o) = \mathbf{L}_e(\mathbf{x}, \omega_o) + \int_{\Omega} \mathbf{f}_r(\mathbf{x}, \omega_i, \omega_o) \mathbf{L}_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

- Как може да се сметне нещо подобно?

Само в сто реда код :)

```
1. #include <math.h> // smallpt, a Path Tracer by Kevin Beason, 2008
2. #include <stdlib.h> // Make : g++ -O3 -fopenmp smallpt.cpp -o smallpt
3. #include <stdio.h> // Remove "-fopenmp" for g++ version < 4.2
4. struct Vec { // Usage: time ./smallpt 5000 && xv image.ppm
5.     double x, y, z; // position, also color (r,g,b)
6.     Vec(double x_=0, double y_=0, double z_=0){ x=x_; y=y_; z=z_; }
7.     Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
8.     Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
9.     Vec operator*(double b) const { return Vec(x*b,y*b,z*b); }
10.    Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
11.    Vec& norm(){ return *this = *this * (1/sqrt(x*x+y*y+z*z)); }
12.    double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; } // cross:
13.    Vec operator%(Vec&b){return Vec(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);}
14. };
15. struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} };
16. enum Refl_t { DIFF, SPEC, REFR }; // material types, used in radiance()
17. struct Sphere {
18.     double rad; // radius
19.     Vec p, e, c; // position, emission, color
20.     Refl_t refl; // reflection type (DIFFuse, SPECular, REFRactive)
21.     Sphere(double rad_, Vec p_, Vec e_, Vec c_, Refl_t refl_):
22.         rad(rad_), p(p_), e(e_), c(c_), refl(refl_) {}
23.     double intersect(const Ray &r) const { // returns distance, 0 if nohit
24.         Vec op = p-r.o; // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
25.         double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
26.         if (det<0) return 0; else det=sqrt(det);
27.         return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
28.     }
29. };
30. Sphere spheres[] = { //Scene: radius, position, emission, color, material
31.     Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
32.     Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
33.     Sphere(1e5, Vec(50,40.8, 1e5), Vec(),Vec(.75,.75,.75),DIFF),//Back
34.     Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(), DIFF),//Frnt
35.     Sphere(1e5, Vec(50, 1e5, 81.6), Vec(),Vec(.75,.75,.75),DIFF),//Botm
36.     Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
37.     Sphere(16.5,Vec(27,16.5,47), Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
38.     Sphere(16.5,Vec(73,16.5,78), Vec(),Vec(1,1,1)*.999, REFR),//Glas
39.     Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12), Vec(), DIFF) //Lite
40. };
41. inline double clamp(double x){ return x<0 ? 0 : x>1 ? 1 : x; }
42. inline int toInt(double x){ return int(pow(clamp(x),1/2.2)*255+.5); }
43. inline bool intersect(const Ray &r, double &t, int &id){
44.     double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
45.     for(int i=int(n);i--;) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
46.     return t<inf;
47. }
48. Vec radiance(const Ray &r, int depth, unsigned short *Xi){
49.     double t; // distance to intersection
50.     int id=0; // id of intersected object
51.     if (!intersect(r, t, id)) return Vec(); // if miss, return black
52.     const Sphere &obj = spheres[id]; // the hit object
53.     Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
54.     double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
55.     if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
56.     if (obj.refl == DIFF){ // Ideal DIFFUSE reflection
57.         double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
58.         Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
59.         Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
60.         return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
61.     } else if (obj.refl == SPEC) // Ideal SPECULAR reflection
62.         return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
63.     Ray reflRay(x, r.d-n*2*n.dot(r.d)); // Ideal dielectric REFRACTION
64.     bool into = n.dot(nl)>0; // Ray from outside going in?
65.     double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;
66.     if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0) // Total internal reflection
67.         return obj.e + f.mult(radiance(reflRay,depth,Xi));
68.     Vec tdir = (r.d*nnt - n*((into?1:-1)*(ddn*nnt+sqrt(cos2t)))).norm();
69.     double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));
70.     double Re=R0+(1-R0)*c*c*c*c*c,Tr=1-Re,P=.25+.5*Re,RP=Re/P,TP=Tr/(1-P);
71.     return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ? // Russian roulette
72.         radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
73.         radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
74. }
75. int main(int argc, char *argv[]){
76.     int w=1024, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // # samples
77.     Ray cam(Vec(50,52,295.6), Vec(0,-0.042612,-1).norm()); // cam pos, dir
78.     Vec cx=Vec(w*.5135/h), cy=(cx%cam.d).norm()**.5135, r, *c=new Vec[w*h];
79.     #pragma omp parallel for schedule(dynamic, 1) private(r) // OpenMP
80.     for (int y=0; y<h; y++){ // Loop over image rows
81.         fprintf(stderr, "\rRendering (%d spp) %5.2f%%", samps*4, 100.*y/(h-1));
82.         for (unsigned short x=0, Xi[3]={0,0,y*y}; x<w; x++) // Loop cols
83.             for (int sy=0, i=(h-y-1)*w+x; sy<2; sy++) // 2x2 subpixel rows
84.                 for (int sx=0; sx<2; sx++; r=Vec()){ // 2x2 subpixel cols
85.                     for (int s=0; s<samps; s++){
86.                         double r1=2*erand48(Xi), dx=r1<1 ? sqrt(r1)-1: 1-sqrt(2-r1);
87.                         double r2=2*erand48(Xi), dy=r2<1 ? sqrt(r2)-1: 1-sqrt(2-r2);
88.                         Vec d = cx*( ( (sx+.5 + dx)/2 + x)/w - .5) +
89.                             cy*( ( (sy+.5 + dy)/2 + y)/h - .5) + cam.d;
90.                         r = r + radiance(Ray(cam.o+d*140,d.norm()),0,Xi)*(1./samps);
91.                     } // Camera rays are pushed ^^^ forward to start in interior
92.                     c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z))*25;
93.                 }
94.             }
95.     FILE *f = fopen("image.ppm", "w"); // Write image to PPM file.
96.     fprintf(f, "P3\n%d %d\n255\n", w, h, 255);
97.     for (int i=0; i<w*h; i++)
98.         fprintf(f, "%d %d %d ", toInt(c[i].x), toInt(c[i].y), toInt(c[i].z));
99. }
100. }
```

Връзка между L_i и L_o

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

Color $L_i(\mathbf{x}, w_i)$:

$x' = \text{trace}(\mathbf{x}, w_i)$

$w_o = -w_i$

return $L_o(x', w_o)$

Rendering equation in pseudocode

Color $L_o(\mathbf{x}, \omega_o)$:

Color sum = 0

for i in 1..num_samples:

$w', pdf = \text{generate_new_random_dir}(\mathbf{x}, w_o)$

$sum += Fr(\mathbf{x}, w_o, w') * Li(\mathbf{x}, -w') * \text{dot}(w', \mathbf{N}) * (1 / pdf)$

$sum /= \text{num_samples}$

return $Le(\mathbf{x}, w_o) + sum$

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o) L_i(\mathbf{x}, \omega_i) (-\omega_i \cdot \mathbf{n}) d\omega_i$$

Algorithms for Global Illumination

- Алгоритмите за GI могат основно да се разделят на
 - Неотместени (unbiased)
 - path tracing, light tracing, bidirectional tracing, metropolis light transport, etc.
 - Отместени (biased)
 - photon mapping, irradiance caching, instant radiosity, etc.
- Неотместен алгоритъм (грубо казано) е такъв, при който грешката се изразява само в шум
 - При отместените могат да се получат резултати, в които няма шум, но грешката да е произволно голяма

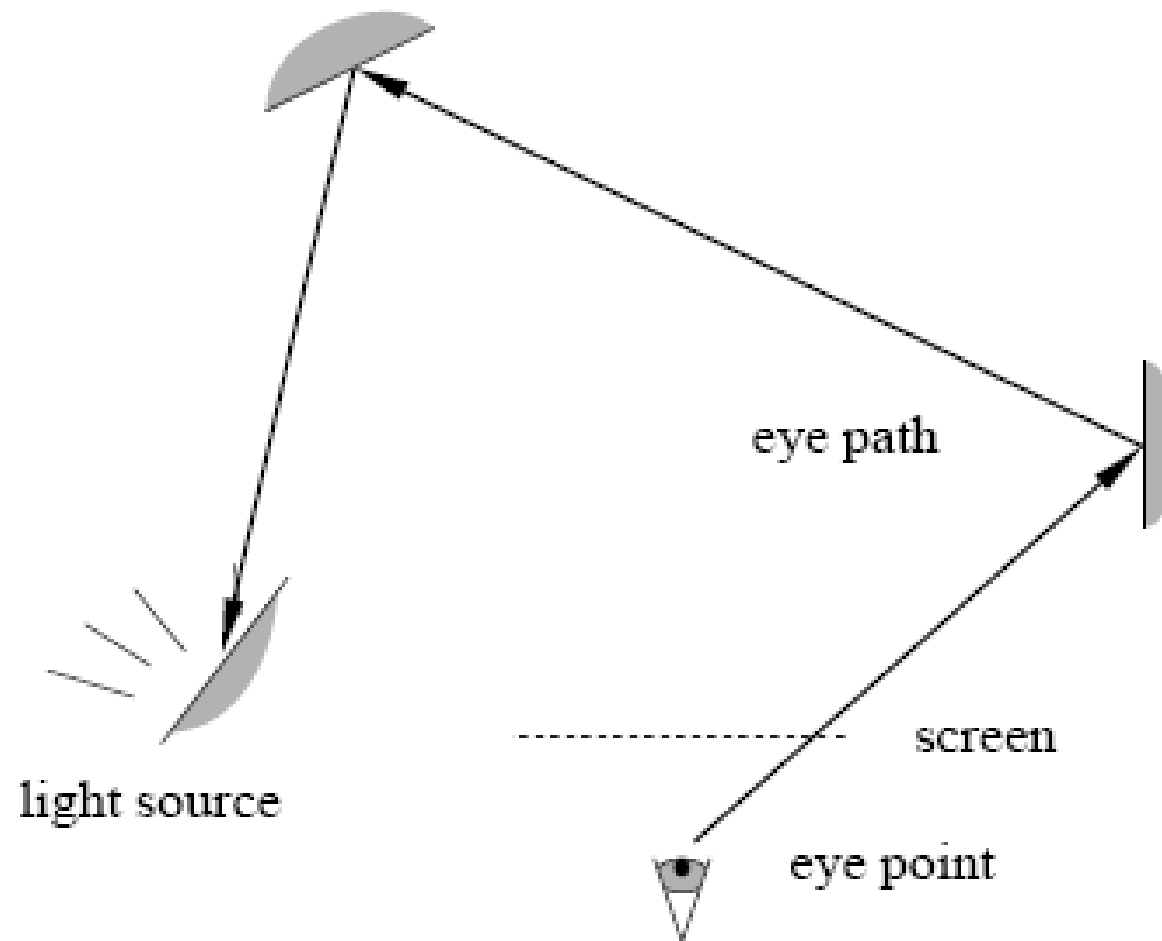
Algorithms for Global Illumination (II)

- Неотместени алгоритми са
 - + предвидими
 - и в математически смисъл на думата
 - + фотореалистични
 - но имат и ограничения, например path tracing не може да получи слънчеви зайчета от не-glossy пречупване и точкова светлина
- - бавни
 - т.е. отместените алгоритми могат да получат гладка, но грешна картинка по-бързо, отколкото неотместените могат да сметнат гладка, но вярна

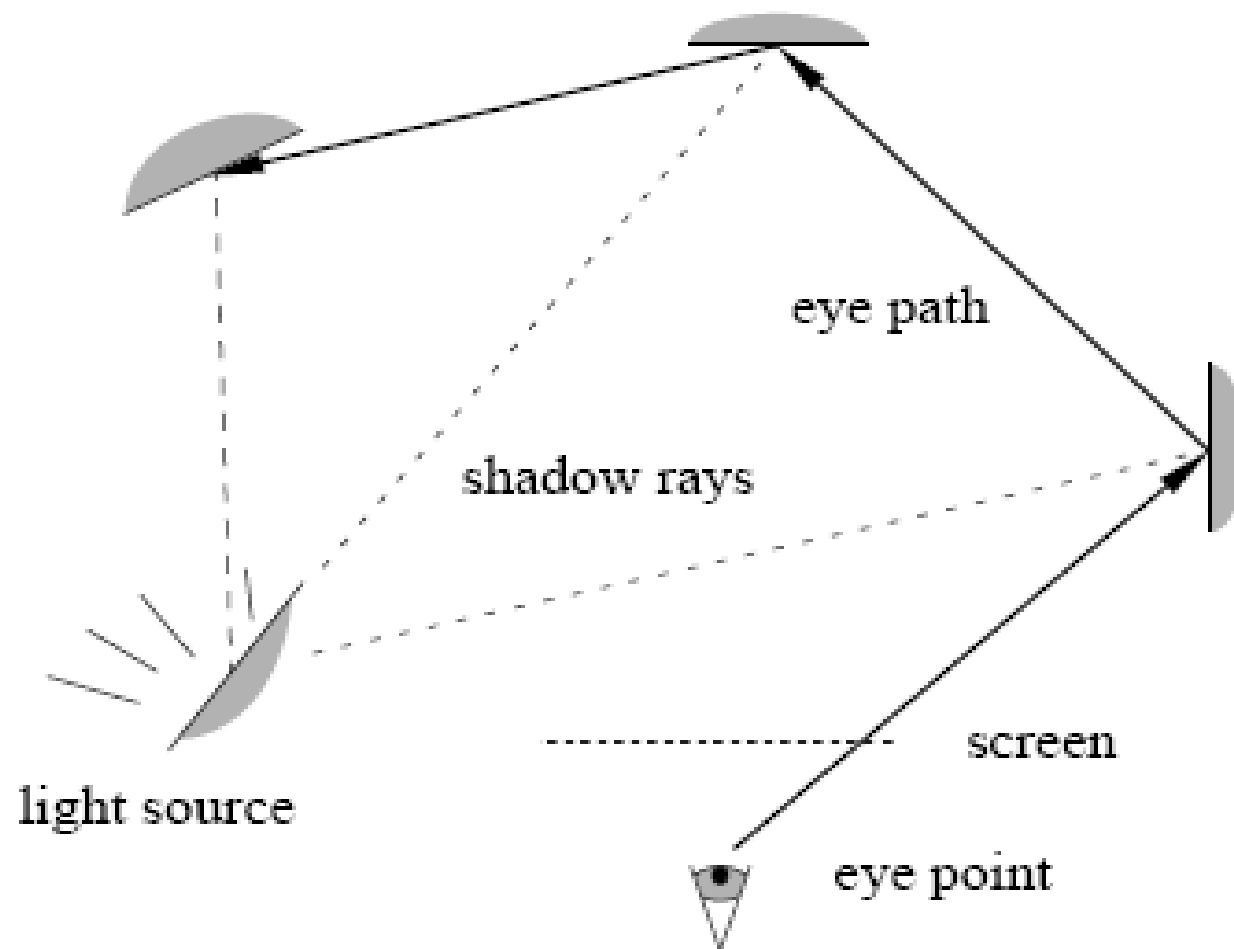
Algorithms for Global Illumination (III)

- Ще разгледаме path tracing, light tracing и bidirectional tracing
- „Път“ ще наричаме редица от точки, такива че
 - Първата точка е върху лещата на камерата
 - Последната точка е върху някоя светлина
 - Останалите точки са върху някоя от обектите в сцената
- И трите алгоритъм използват, че формулата на Каджия може да се разпише като интеграл над всички пътища от сцената
- Основната разлика между трите е в начина, по-който се генерират пътищата

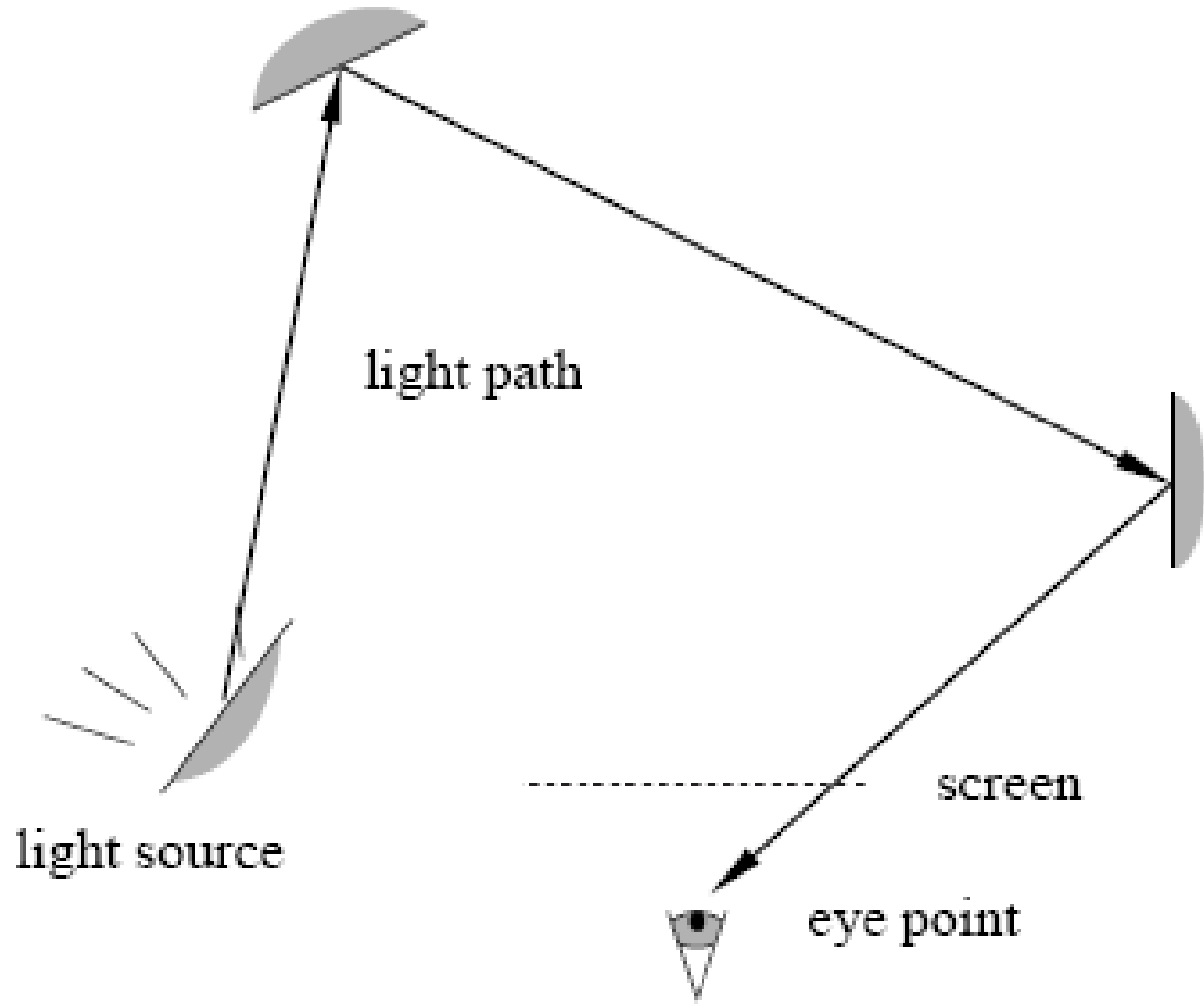
Path tracing



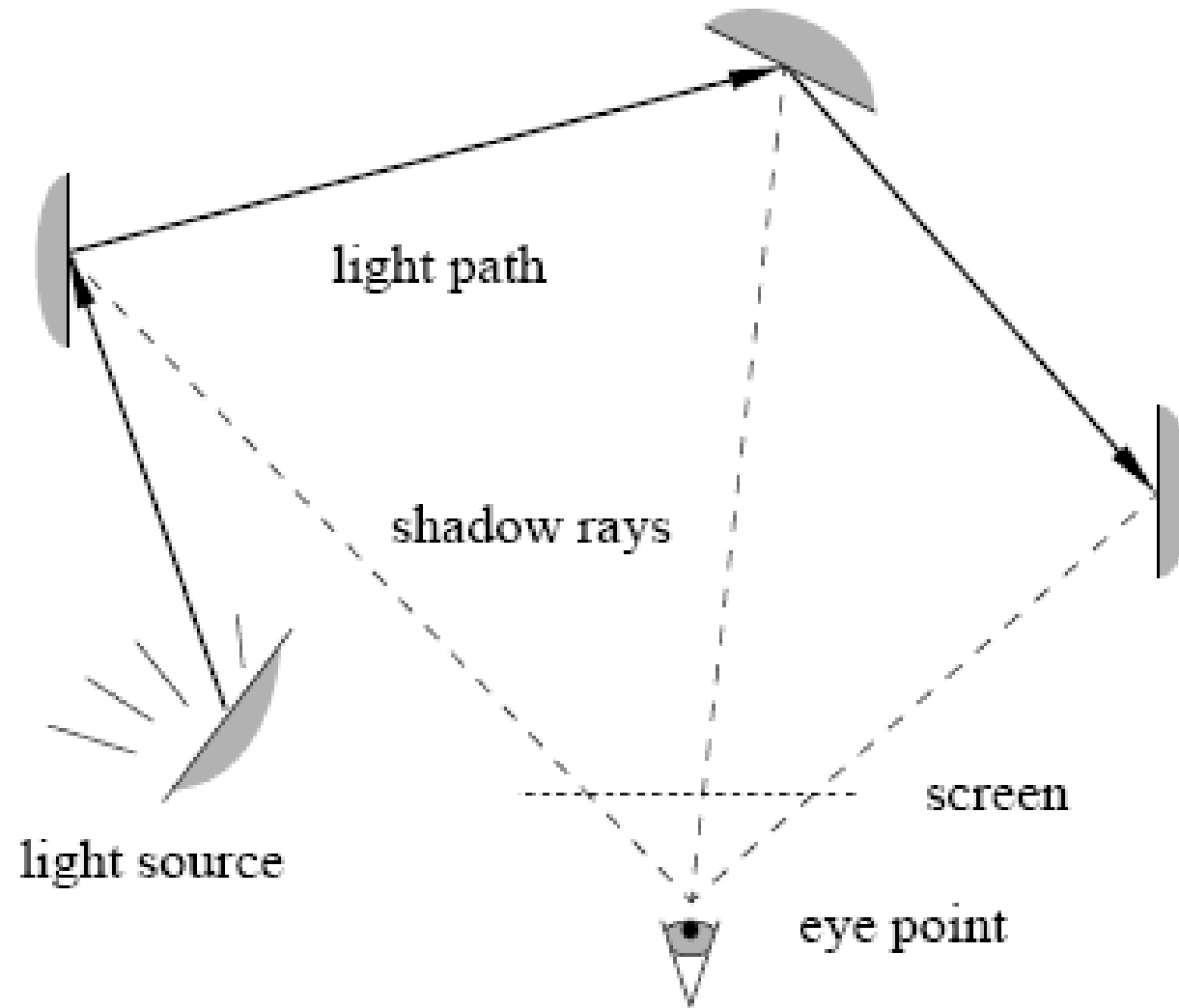
Path tracing (II)



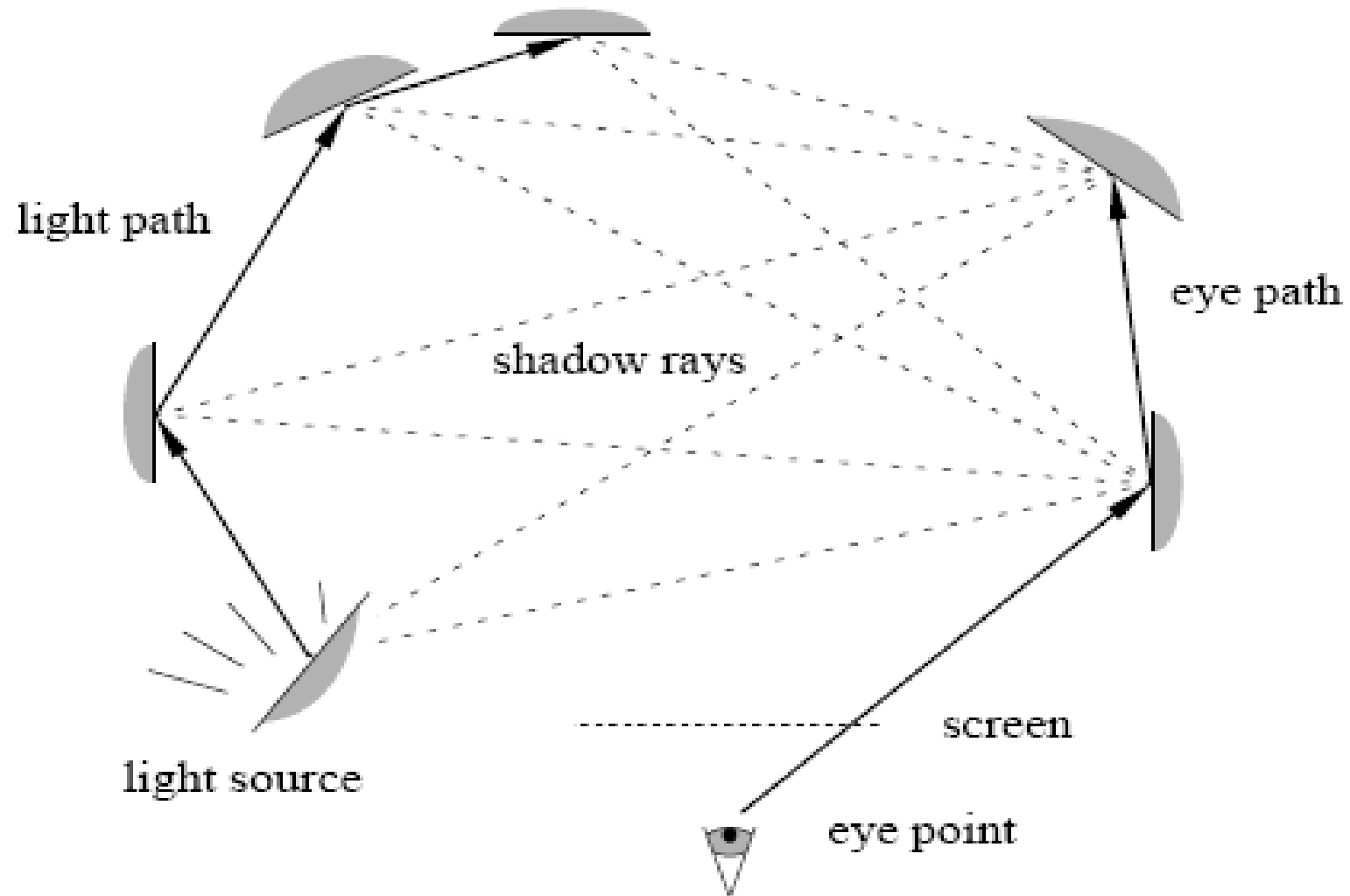
Light tracing



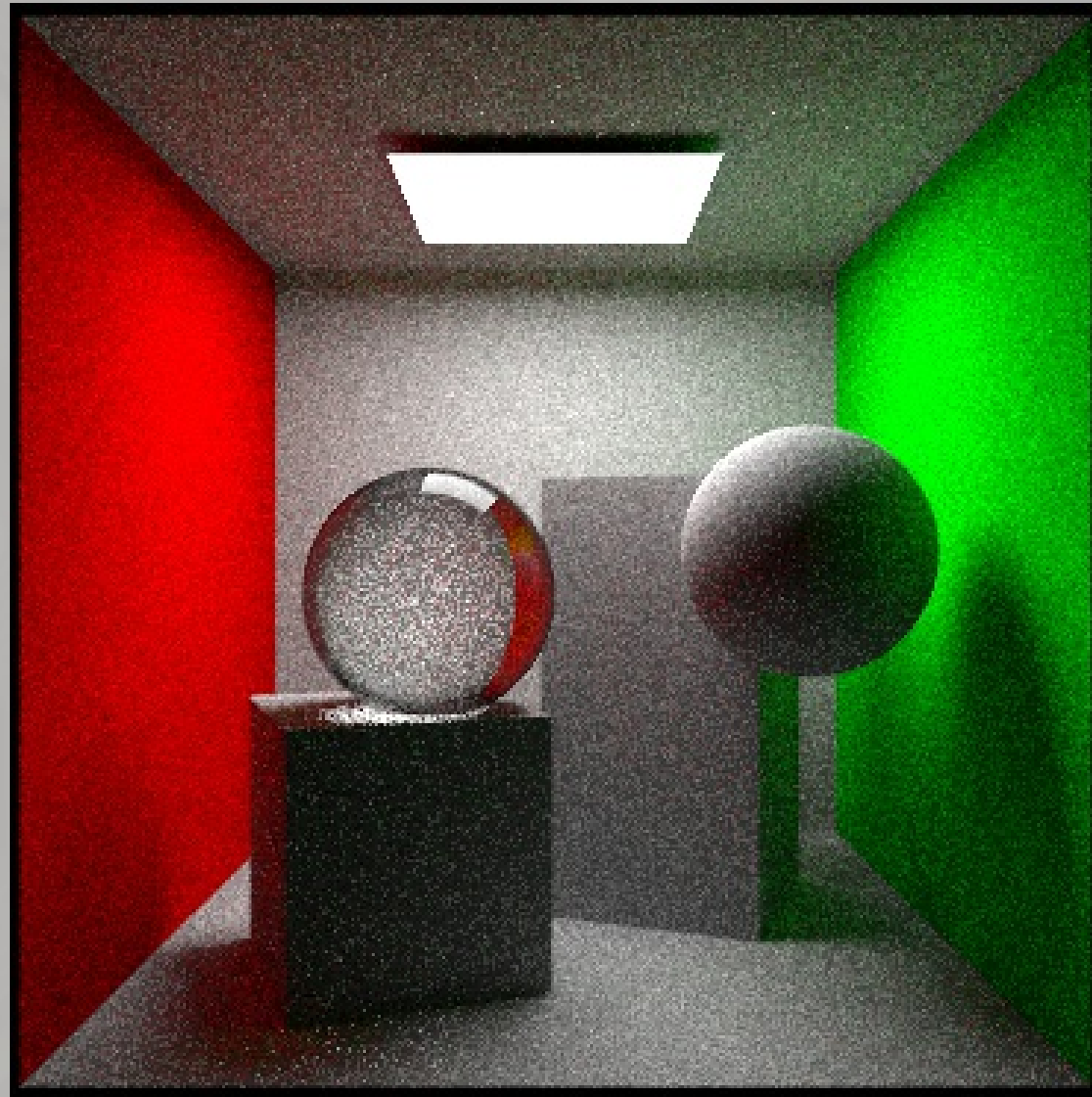
Light tracing (II)



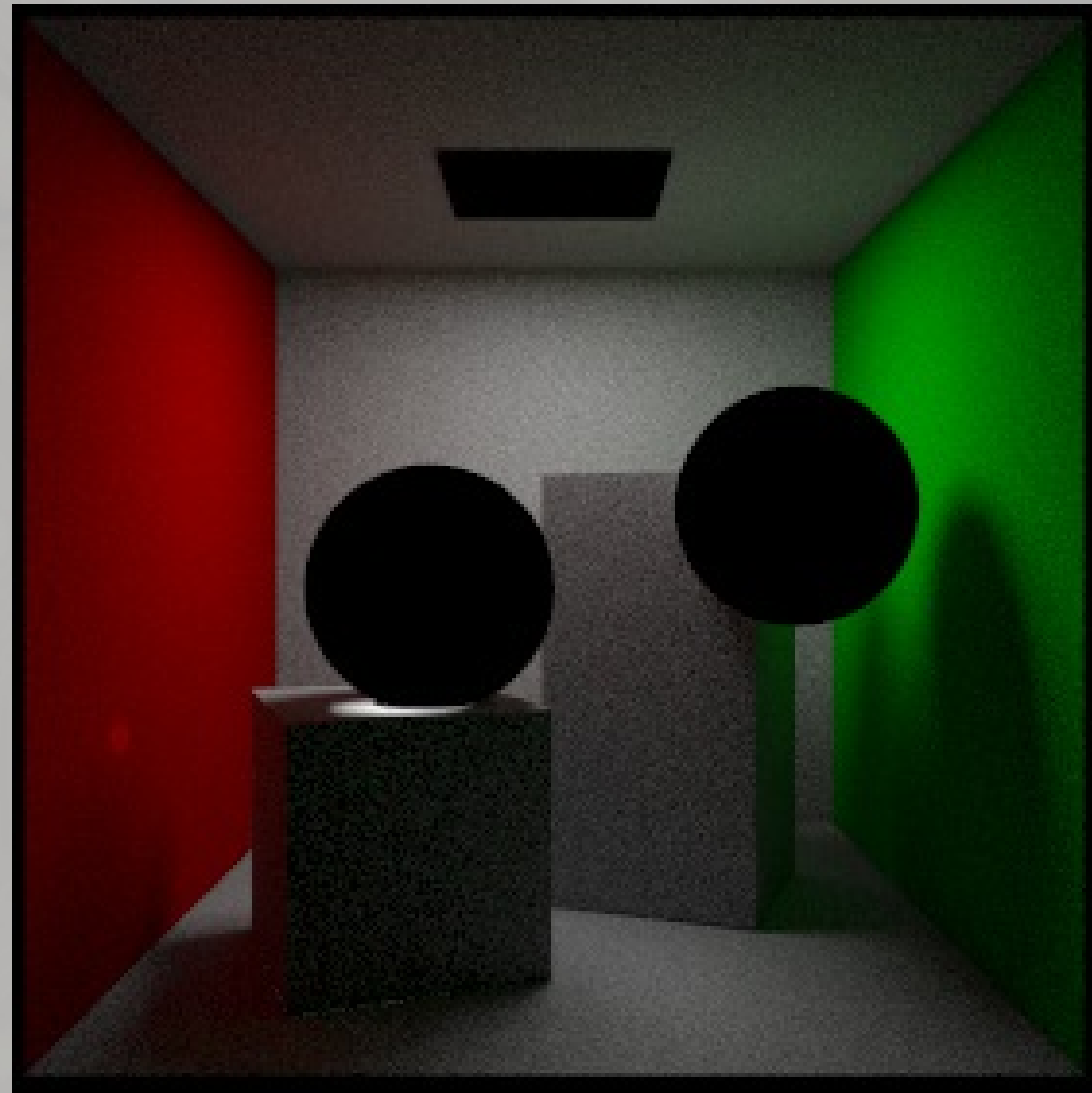
Bidirectional path tracing



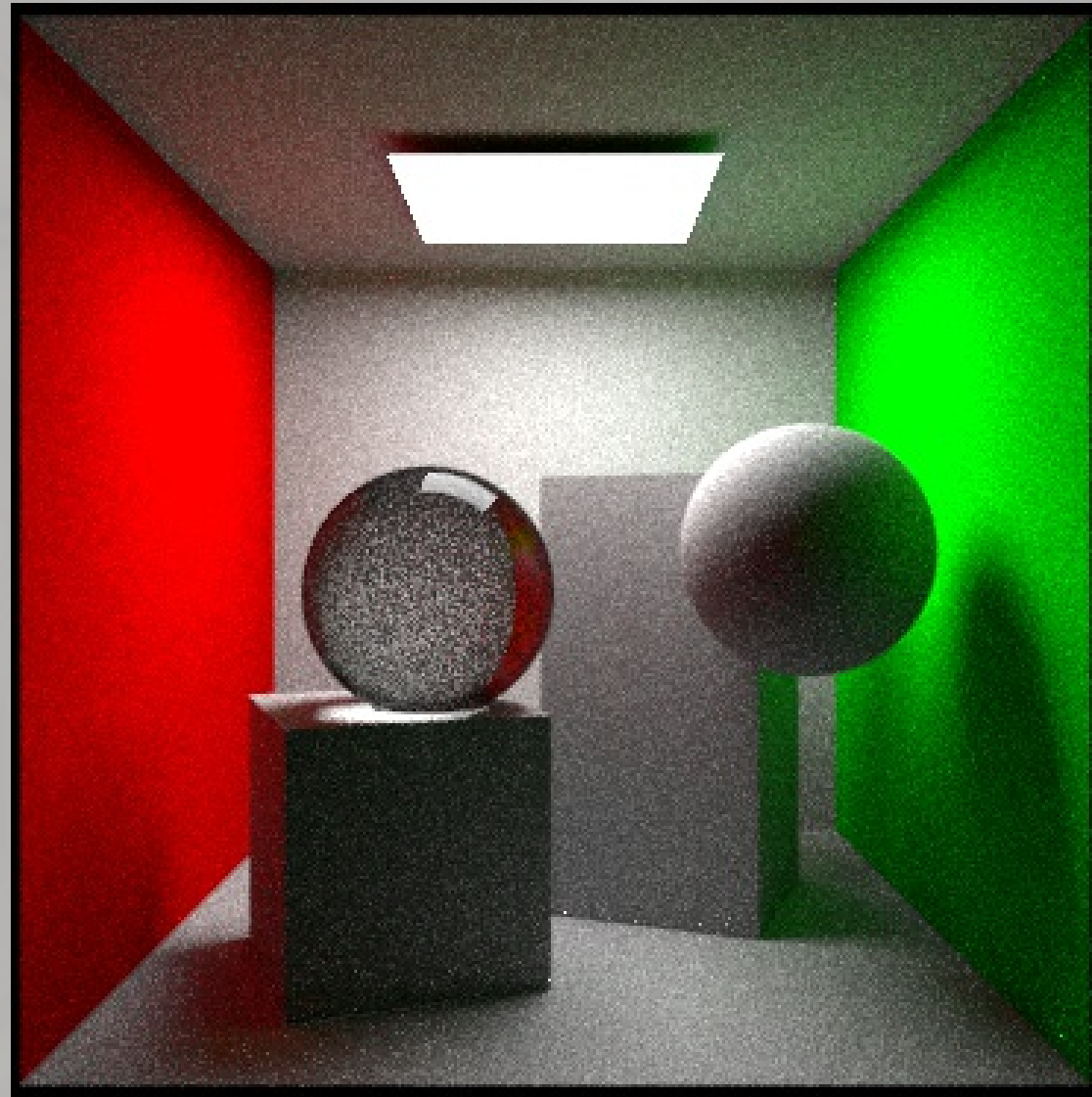
Path tracing result



Light tracing result



Bidirectional tracing result



Ресурси

- По-подробен разбор на различните алгоритми за глобално осветление, вижте лекцията от CG²2012
- The holy bible по BDPT е тезиса на Eric Veach