3D графика и трасиране на лъчи v.5.0

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)

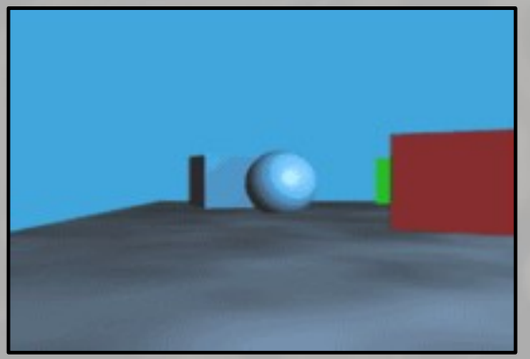
- Алгоритми за глобално осветление
  - Path tracing
  - Light tracing (накратко)
  - Bidirectional path tracing (накратко)

Анонси

- Трябва да завършим гласуването за датите тази седмица
  - Гласувайте **сега!**
  - Гласуването се затваря утре (**16.05**) в 13:00

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

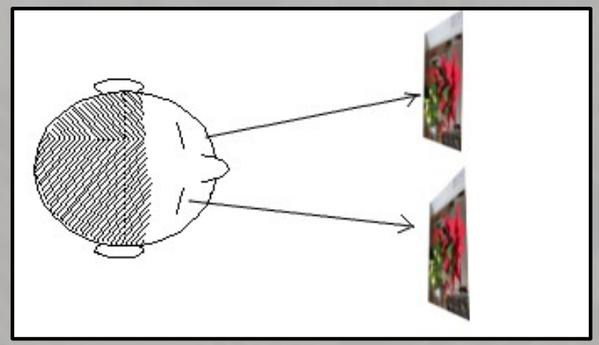


Стереоскопия

- 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



LCD затъмнителни очила (shutter glasses)

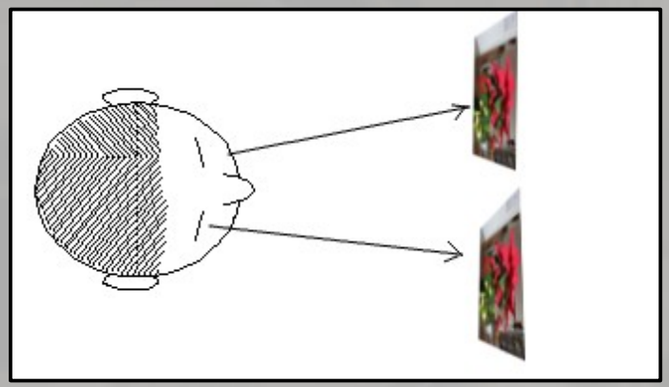
- Пред всяко око има един LCD елемент, който може да закрие плътно гледката на окото
- CRT монитор показва редуващи се кадри (ляв-десен-ляв-десен-...). Затъмняването на очилата се редува по подобен начин и е синхронно с монитора.
- Така всяко око вижда само „полагащите“ му се кадри

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 цветопредаване, добра разделителна способност
- Недостатъци
  - Трептене, монитора трябва да опреснява поне на 120 херца
  - Не работи с TFT монитори
  - Остатъчно изображение върху фосфора (ghosting)

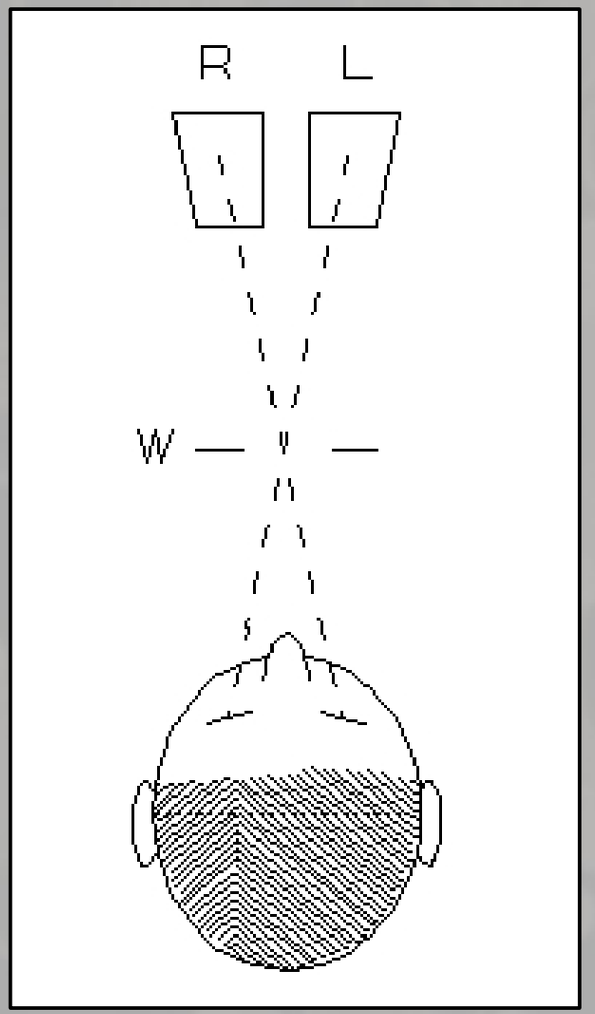
„Успореден“ метод

- 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)

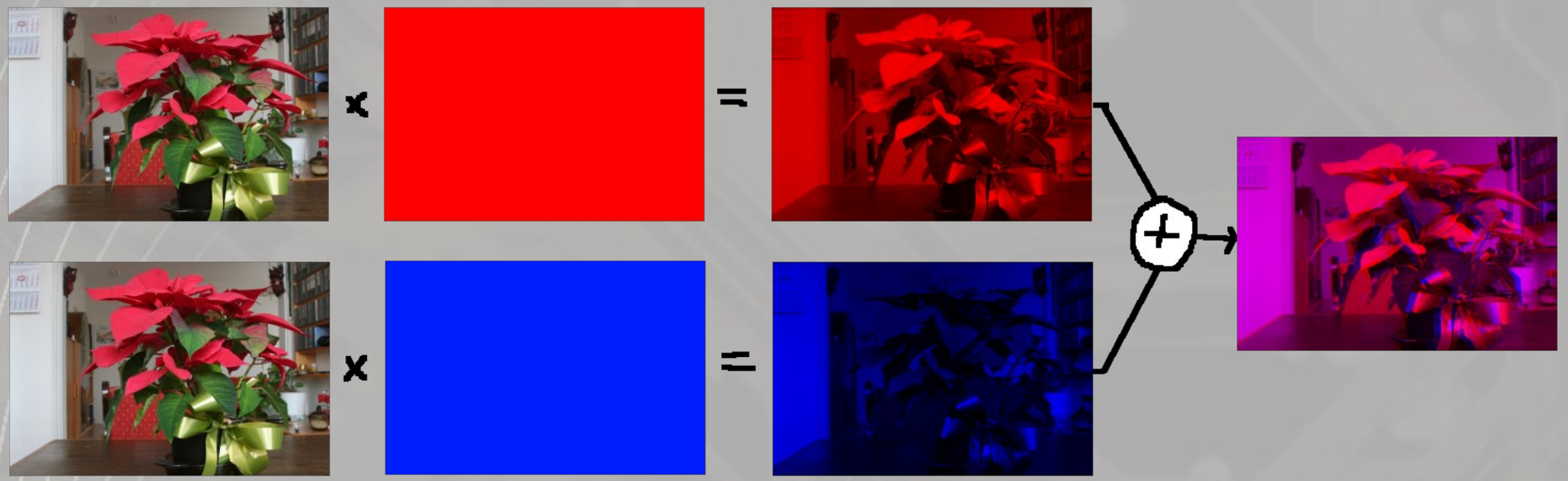
„Кръстосан“ метод

- Предимства
  - Не изисква никаква техника
  - Пълно качество: и откъм разделителна способност, и откъм цветове
- Недостатъци
  - Изисква научаване, което за някои хора е доста трудно
  - Кара мускулите на очите да се напрягат по необичайни за тях начини („Vergence-accomodation conflict“)

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).

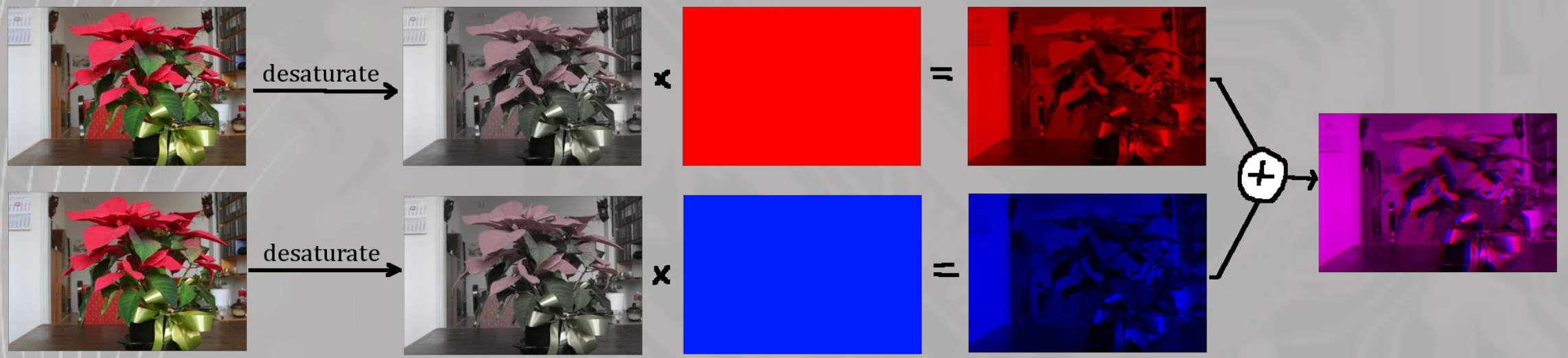
Анаглифни очила



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 благодатно да „убием“ малко цветовете на входните изображения (desaturation):

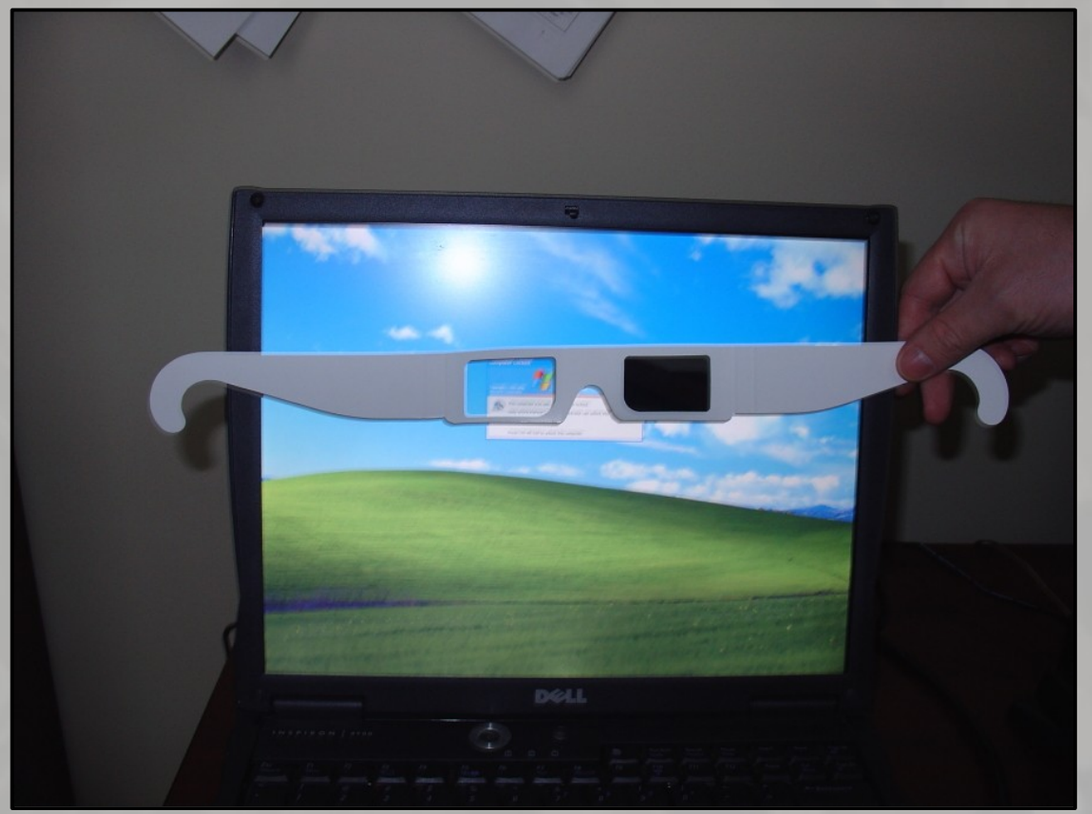


Анаглифни очила

- 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)

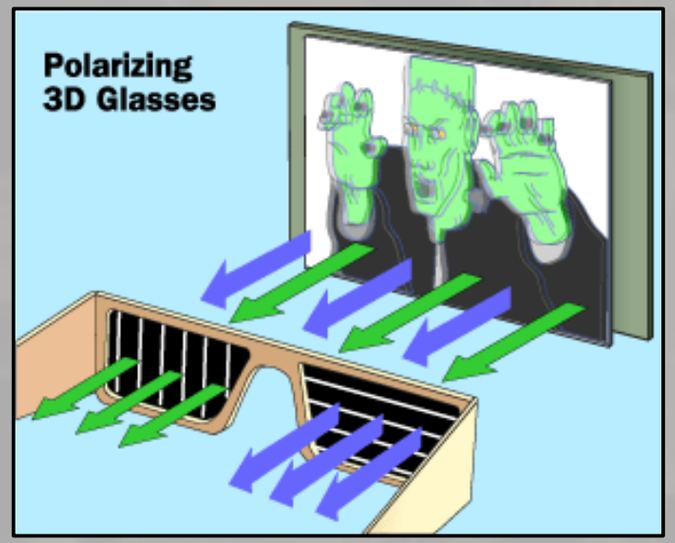
Очила с поляризационни филтри

- 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 поляризиращи 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 (примерна реализация в RealD 3D)
  - Изисква или два проектора, или сложна технология за смяна на поляризацията на един

Спектрално-разделени очила

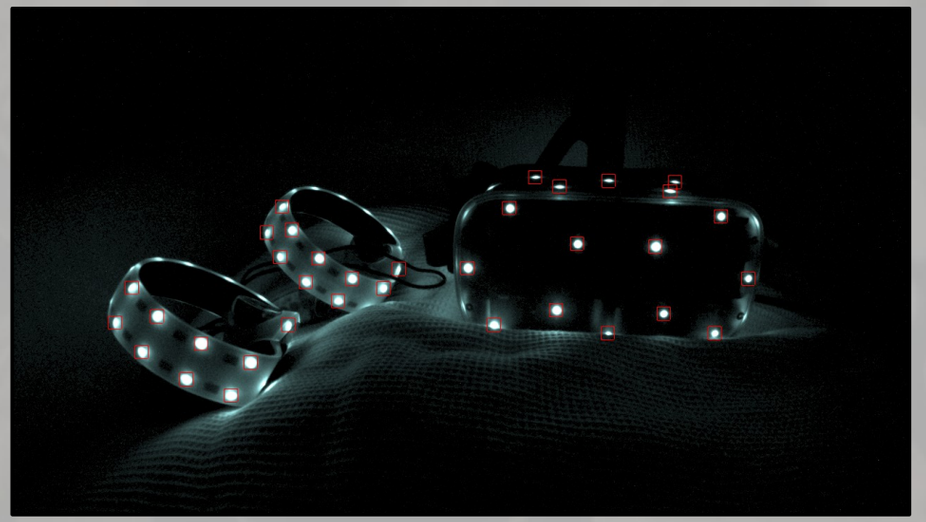
- Същата идея като при поляризиращите филтри, само че филтрите са спектрални – пропускат само различни (непресичащи се) части от спектъра
  - За разлика от поляризиращите филтри, няма проблеми при накланянето на главата
- Човек не усеща „нарязаността“ на спектъра; ако дразнението на R, G и B рецепторите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

Двойка екрани

- Представляват малки екрани, по един за всяко око
  - Недостатъци:
    - Изисква специализиран софтуер и хардуер
    - Относително скъпи
      - Но VR индустрията работи по въпроса
    - Ниска разделителна способност
    - Vergence-accomodation conflict

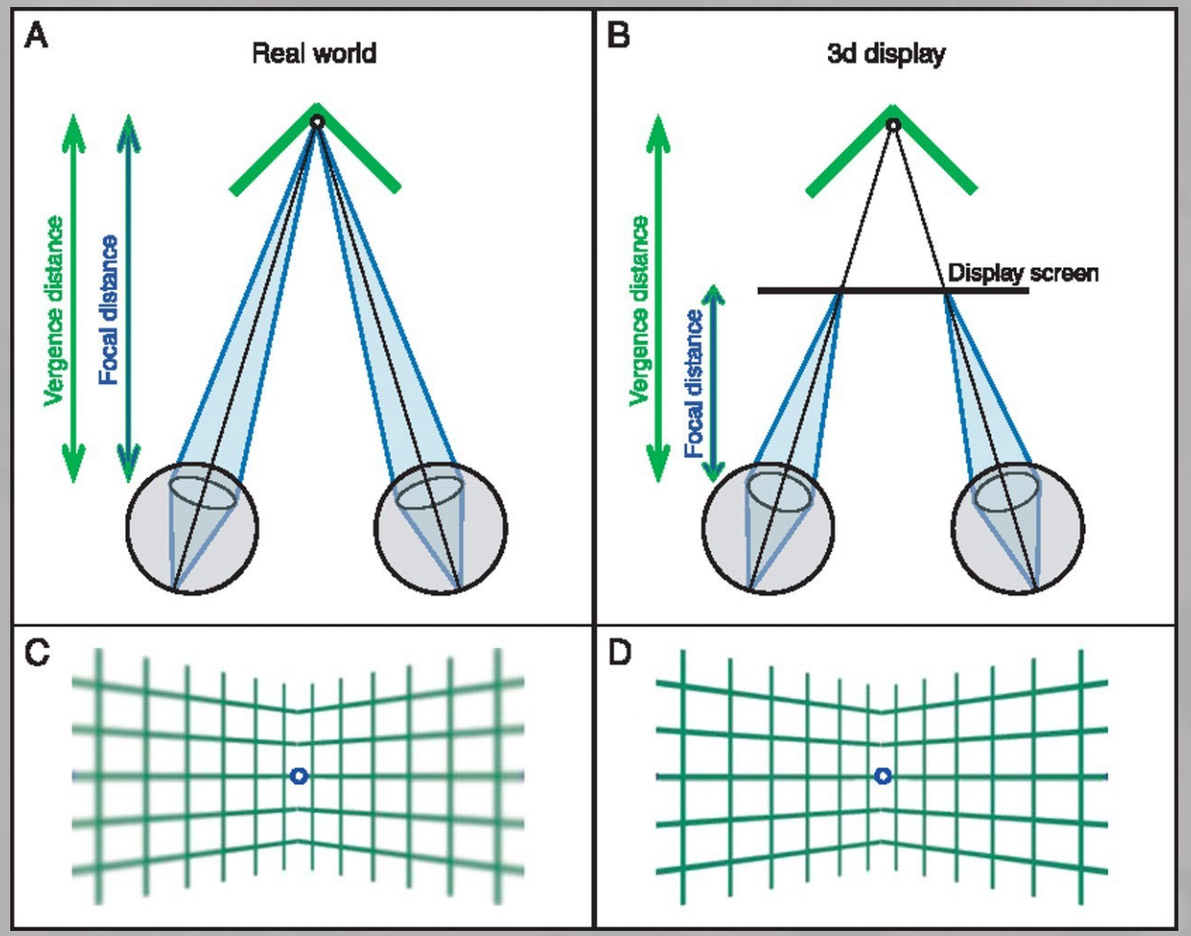
Очила за виртуална реалност (VR)

- Екранът е един, но чрез фокусиращи лещи, всяко око вижда отделна част от него
- Очилата съдържат система за следене („tracking“) на позицията на зрителя в пространството
  - Бърза (msec) и точна (mm)
  - Това позволява много бърз motion-render-display цикъл (90+ fps, под 20ms motion-to-photon)



Очила за виртуална реалност (VR)

- Oculus Rift, HTC Vive и т.н. се различават основно по вида на tracking решението, но дисплеите са много сходни
  - Дисплеят е много близо пред очите, затова има лещи, симулиращи отдалечен екран
  - Очите фокусират на това (фиктивно) разстояние



Vergence-accomodation conflict

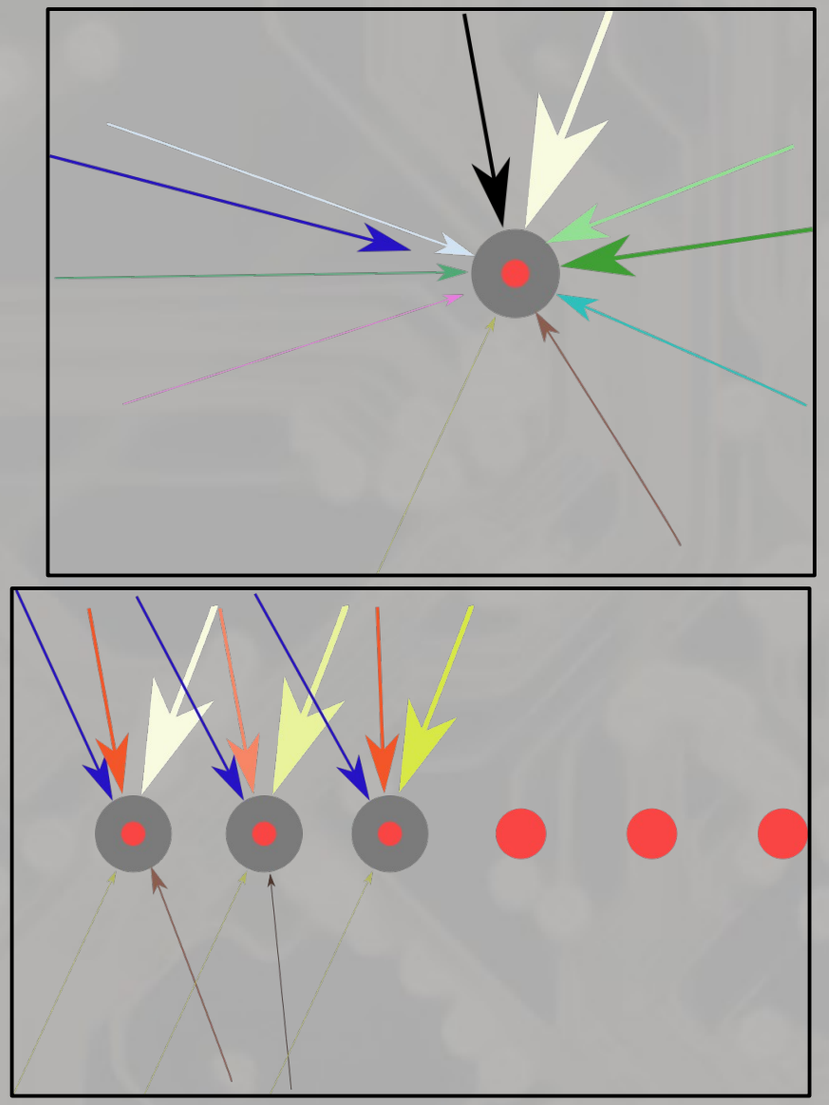
- Фокусирането на очите (*акомодация*) е фиксирано и не зависи от обектите 3D сцената
  - Обикновено се избира разстояние, близко до безкрайност
- Същевременно, различните 3D обекти са на различни разстояния (по паралакс), от 0m до безкрайност
  - Очите конвергират на тези разстояния (*конвергенция/vergence*)
- Несъответствието между двете се нарича *vergence-accomodation conflict* и е съществен проблем на тази технология

Vergence-accomodation conflict

- Конфликтът причинява различни неудобства на зрителите
  - Немалко хора чувстват главоболие, гадене и замаяност след употреба на VR очила
  - Създателите на игри и програми се стараят да няма много близки обекти, за да ограничат несъответствието

Light-field Images

- Ако в дадена точка измерим светлинните лъчи, идващи от *всички* посоки (2D карта, примерно чрез сферични координати), то все едно имаме pinhole камера
- Ако съберем тези данни за всички точки в дадена малка 2D област (примерно колкото челната леща на фотоапарат), то ние имаме *Light-field image* (4D карта)



Light-field Images

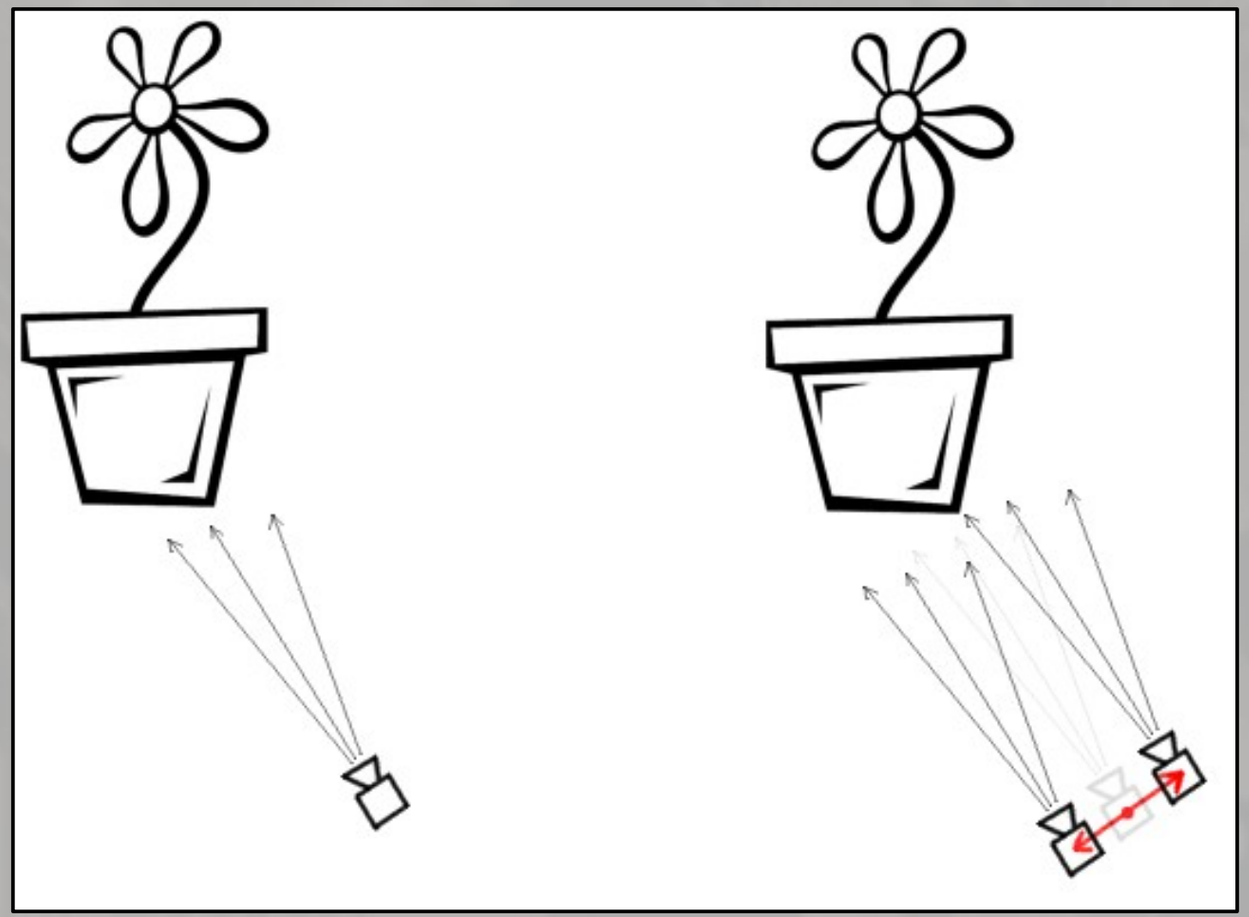
- Събраните данни са пълният комплект „светлинни лъчи“, които са минали пред очите на зрителя
  - Можем да симулираме фокусиране (вкл. произволно рефокусиране)
  - Можем да генерираме стереоскопични изображения
  - и т.н.
- Има камери (напр., Lytro), които „заснемат“ light-field снимки в ограничена форма (low-res, 4D ain't cheap)

Light-field displays

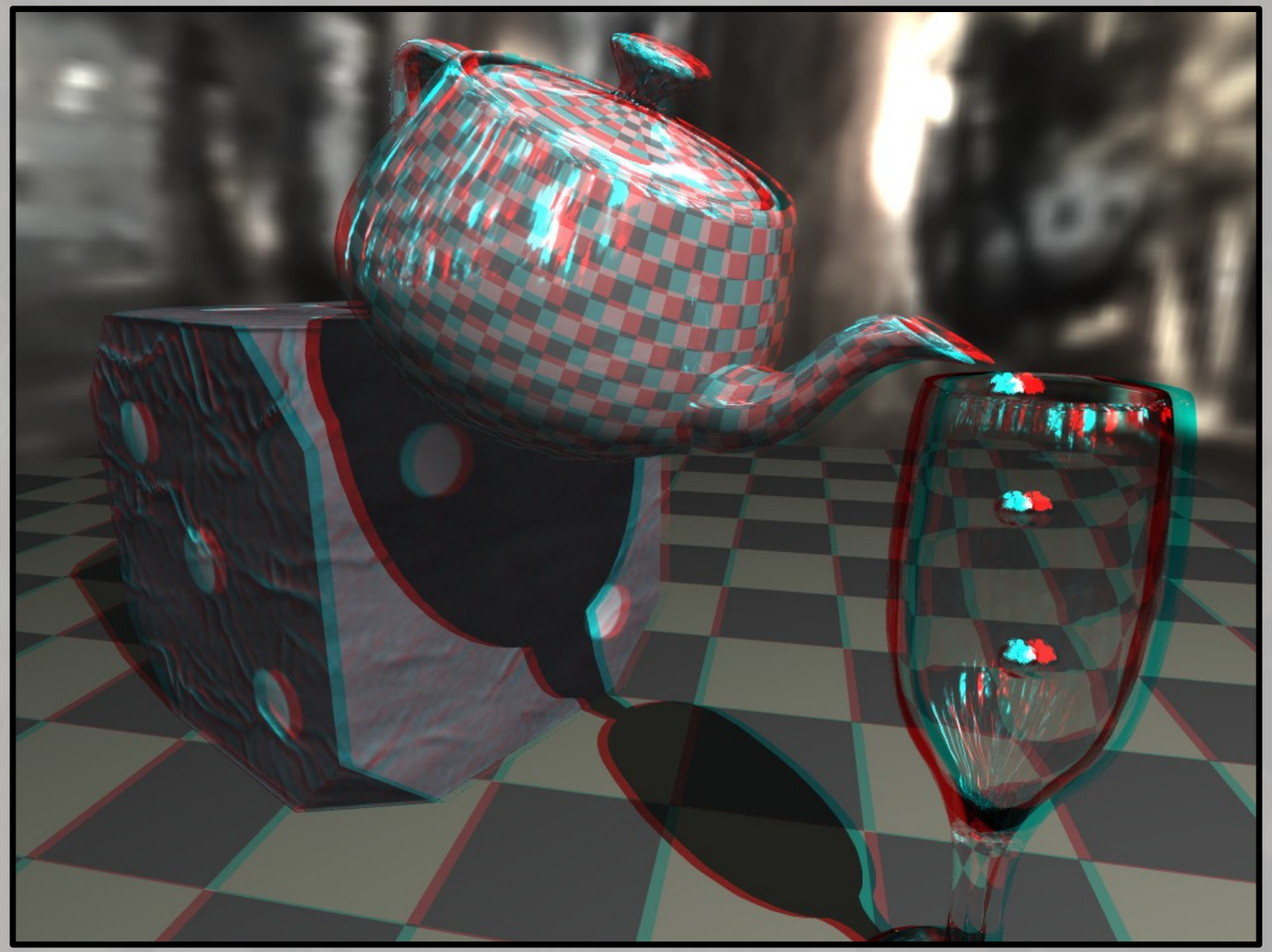
- Предизвикателството пред индустрията сега е да създаде light-field display
  - Екран с пиксели, които излъчват различно количество светлина в различните посоки
  - Засега има разнообразни multi focal plane дисплеи
  - Нещата се развиват *в момента*

Стереоскопично рендериране

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



Резултат



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

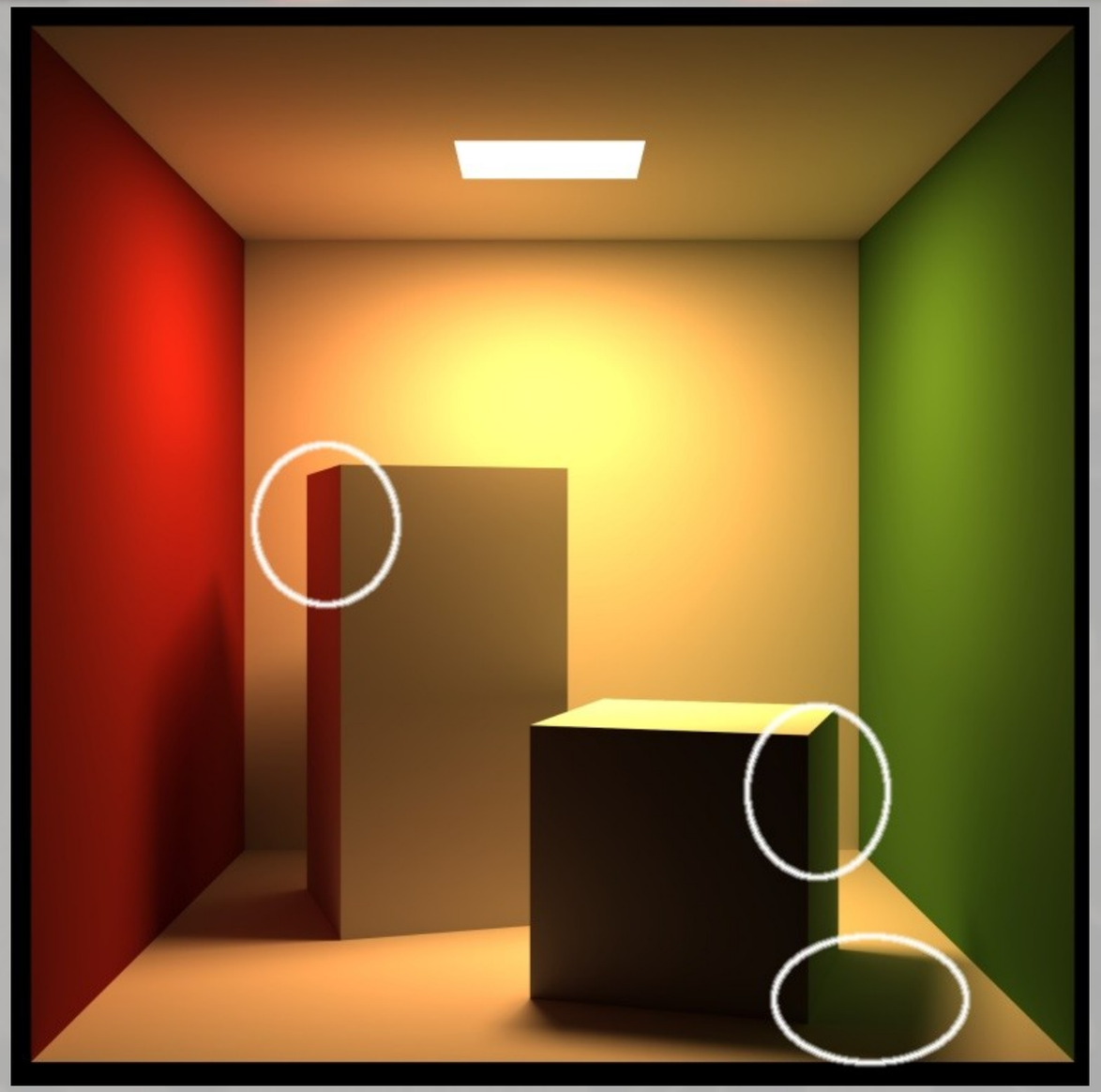
Глобално осветление

Глобално осветление

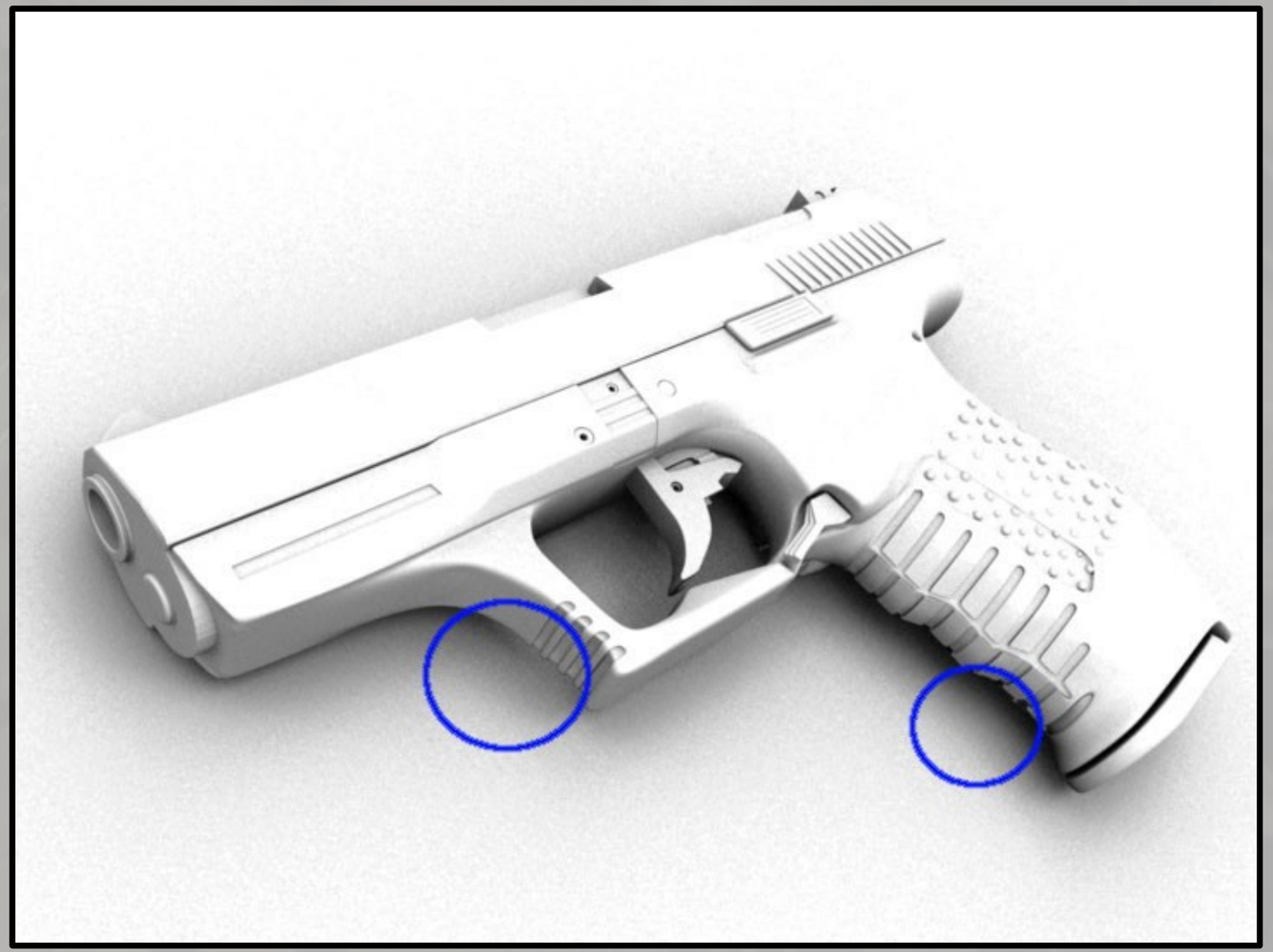
Глобално осветление = директно осветление + индиректно осветление

- 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

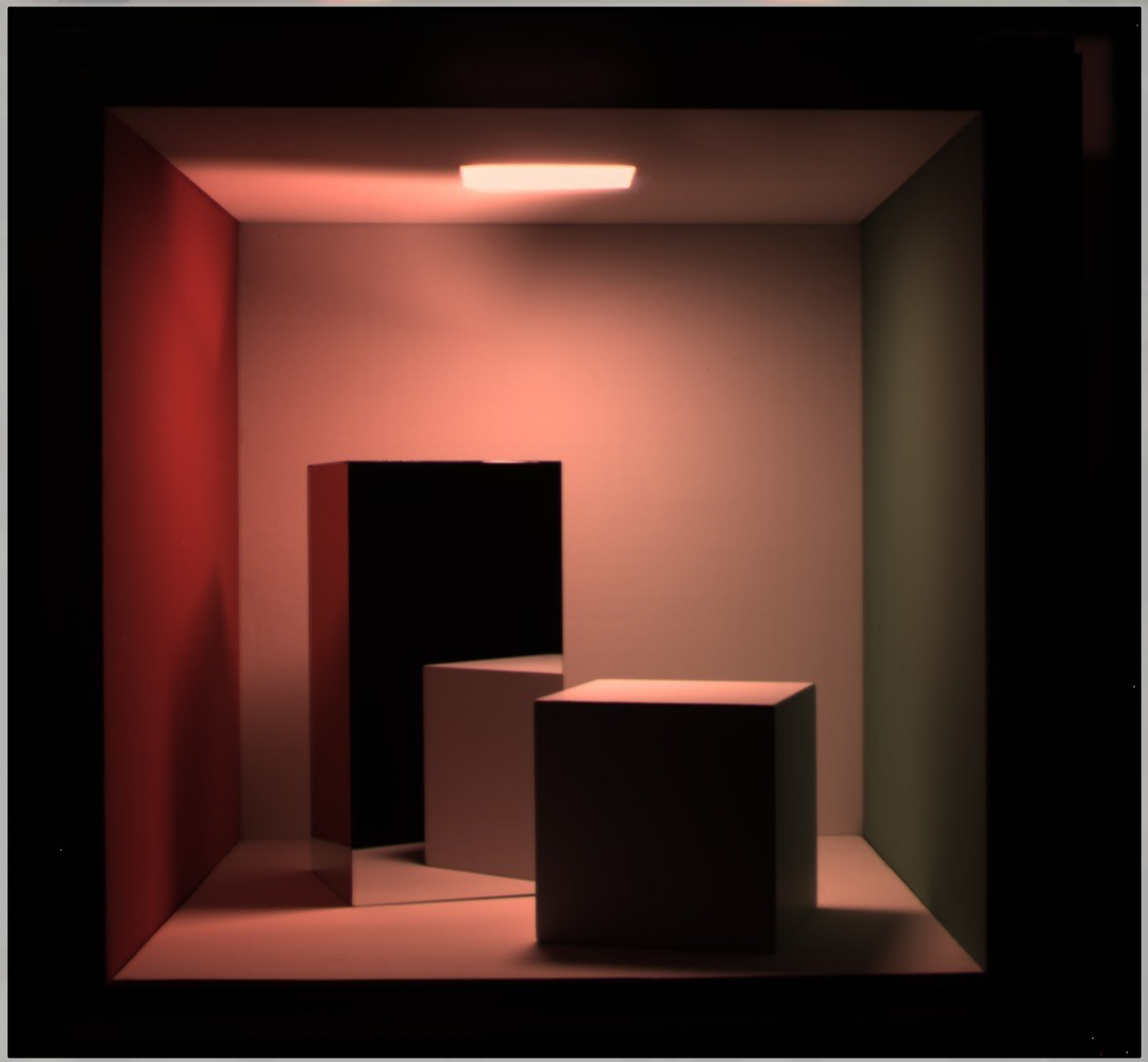


Indirect illumination

Indirect Illumination IRL

[Видеоклип от Блейчево]

The Cornell box

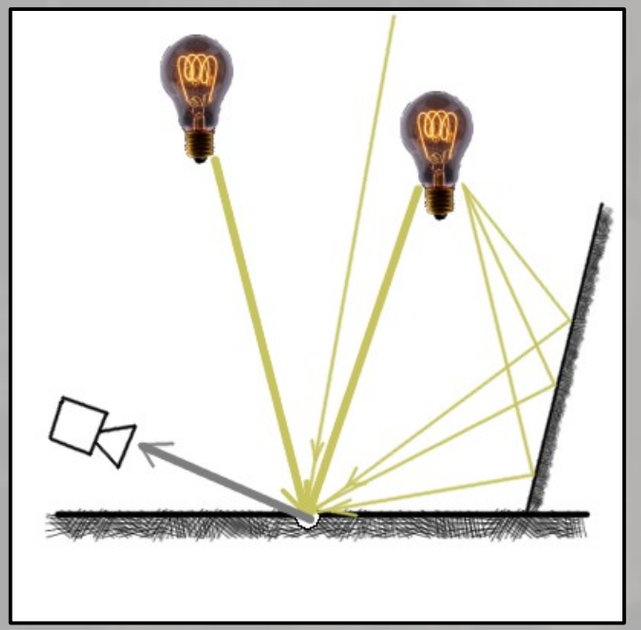


The Cornell box

- 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

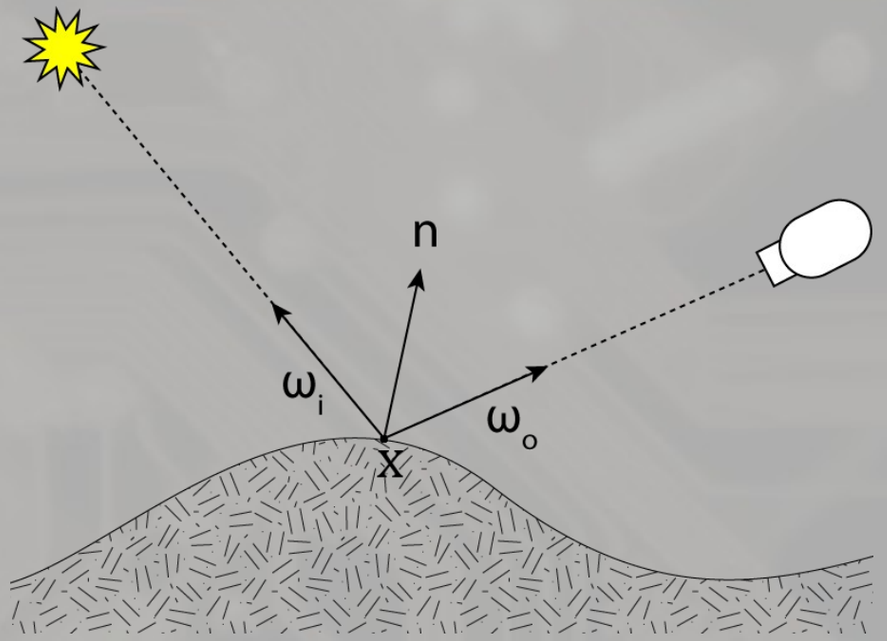
- Най-общо казано, за да пресмятаме глобалното осветление в дадена точка, трябва да:
  - Пресметнем светлината, идваща от цялата сцена, в тази точка (irradiance)
  - В зависимост от отражателните свойства на повърхността, да сметнем каква част от входната светлина се отразява по посока камерата
    - 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 **R**eflectance **D**istribution **F**unction 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:

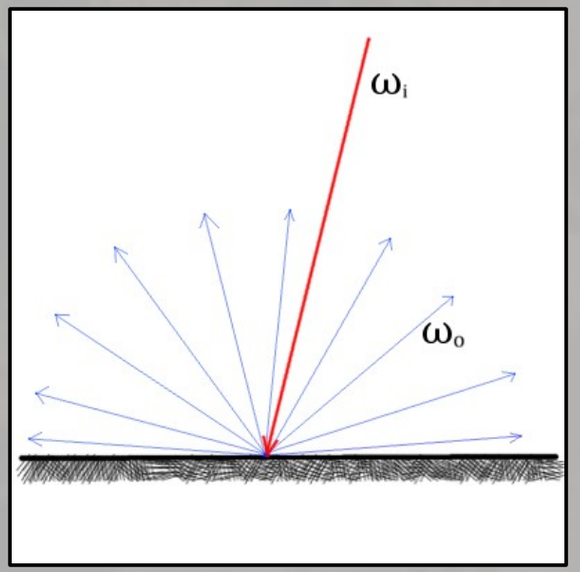
  $f_r(x, \omega_i, \omega_o) := \mathbf{1/\pi}$

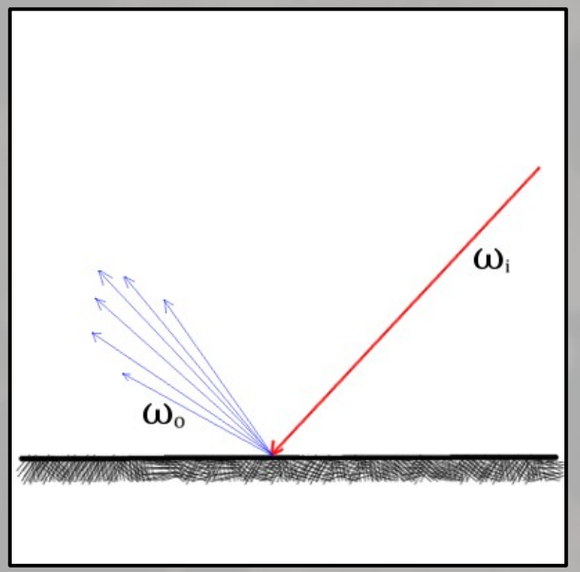
  (ще обясним защо 1/π след малко)

- Glossy BRDF (за грапави отражения):

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

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



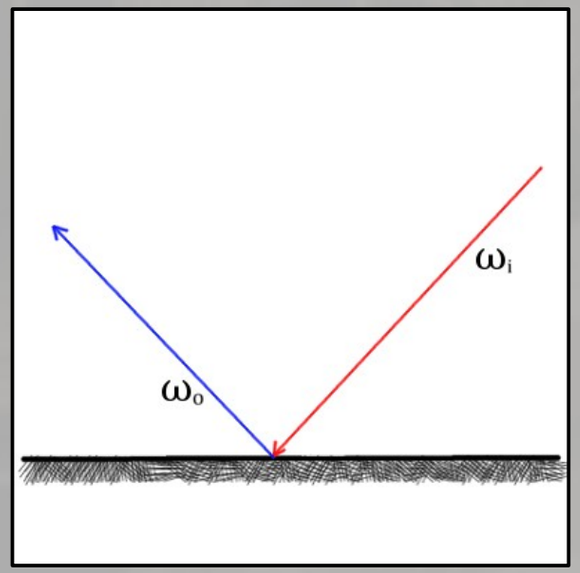


Примери за BRDF

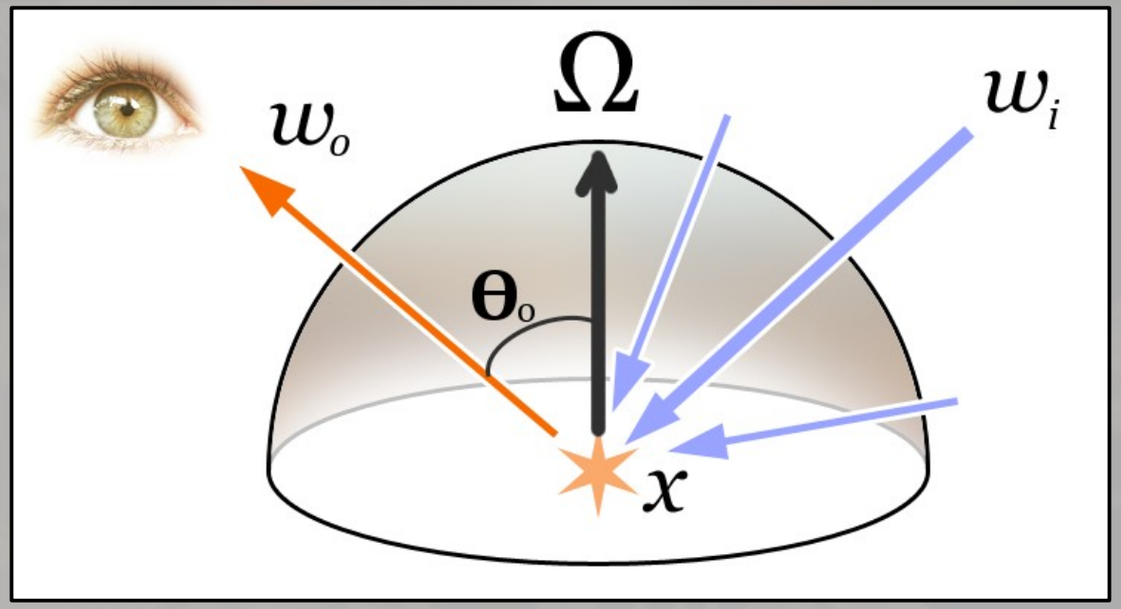
- Чисто отражение:

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

- Т.е., BRDF-ът може и да е прекъсната 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/π* 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/π навсякъде, но това е само при бял обект без текстура. Ако има (дифузна) текстура, то връщаният резултат зависи от точката *x*:

  $f_r(x, \omega_i, \omega_o)$:= texture.sample(x)

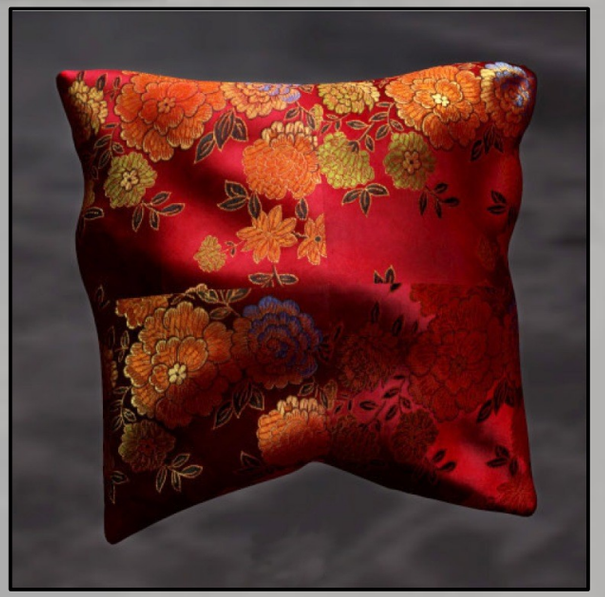
  - Подобно и за останалите BRDF-и, които споменахме. Например, reflect() функцията изисква нормал, около който да отразява, а той зависи от точката *x*.

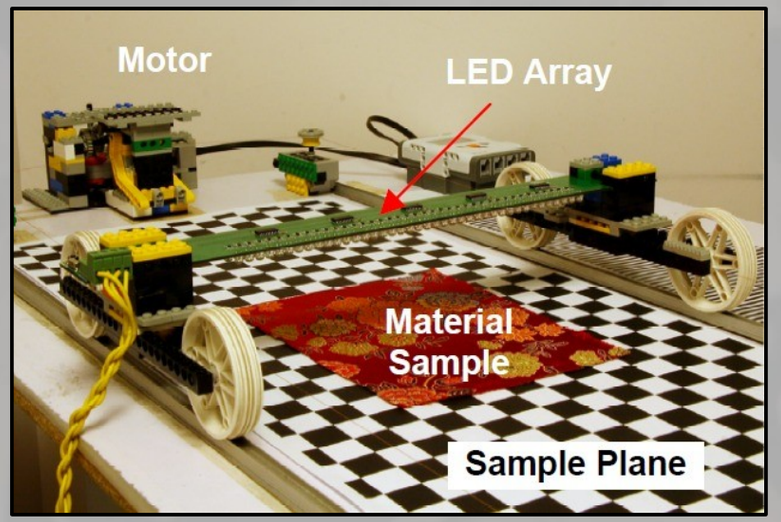
BRDFs: още по-сложно

- 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 някой случай, 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, който „сканират“ даден материал и извличат BRDF-то му [041-wang-video]
  - Вижте също talk-а от CG^{2}2013



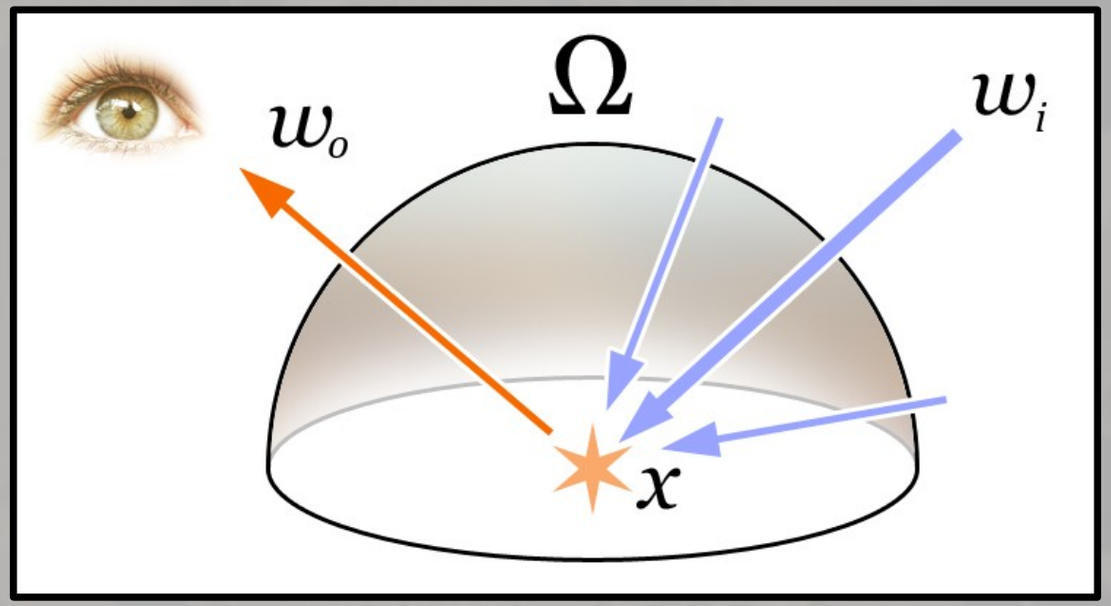


„Фундаменталната формула“ на Каджия

- 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

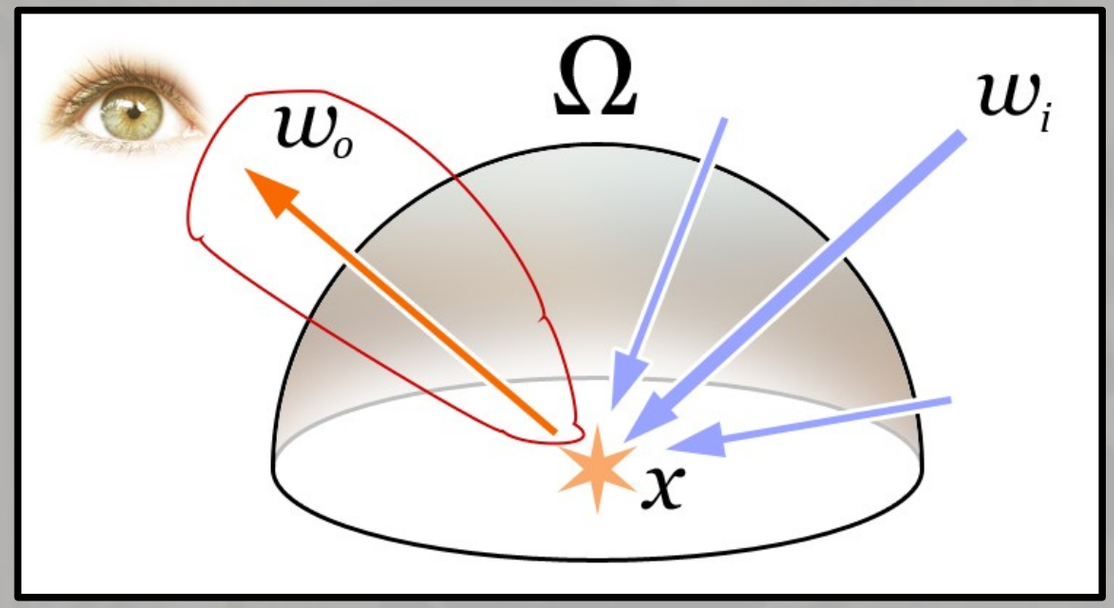
$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_r(x, \omega_i, \omega_o)\, L_i(x, \omega_i)\, (-\omega_i \cdot n)\, d\omega_i$$



The Rendering Equation

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

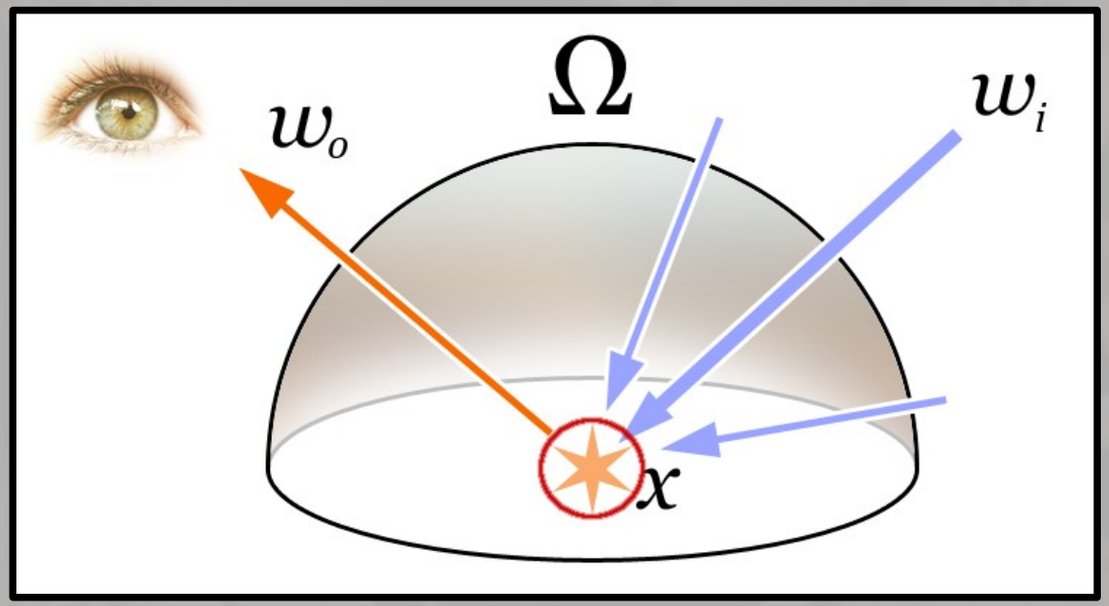
- $L_o(x, \omega_o)$ показва отразената светлина от точката x в посока ω_o
  Например, към камерата:
  - Ако от камерата сме пуснали лъч $-\omega_o$, който е пресякъл геометрията в точка x.



The Rendering Equation

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

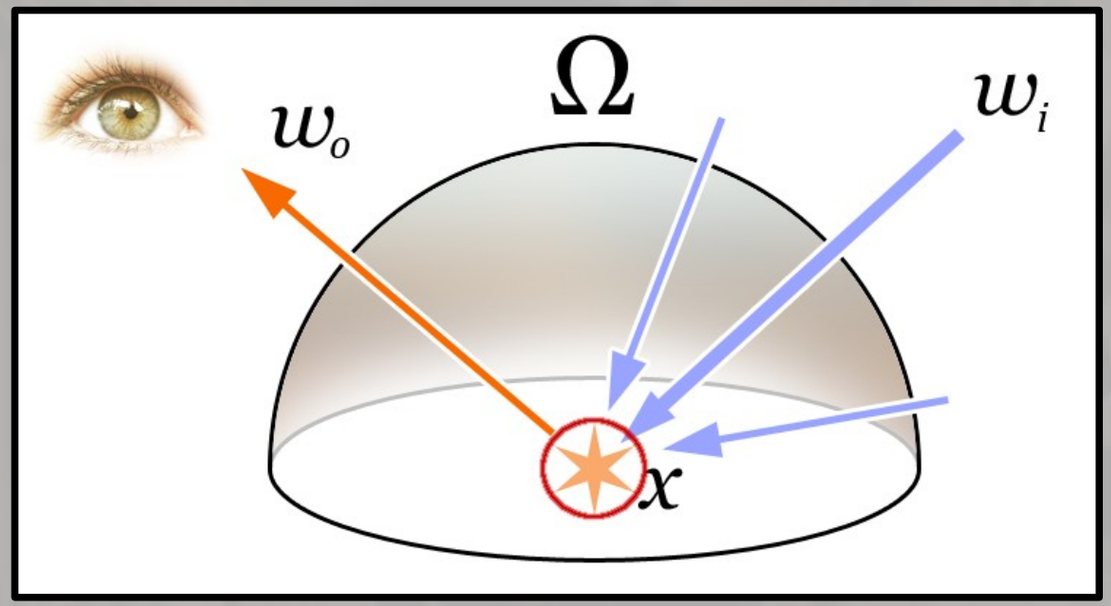
- Ако x е част от лампа, $L_e(x, \omega_o)$ показва излъчваната светлина от точката x в посока ω_o
- 0 за всички неизлъчващи обекти.



The Rendering Equation

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

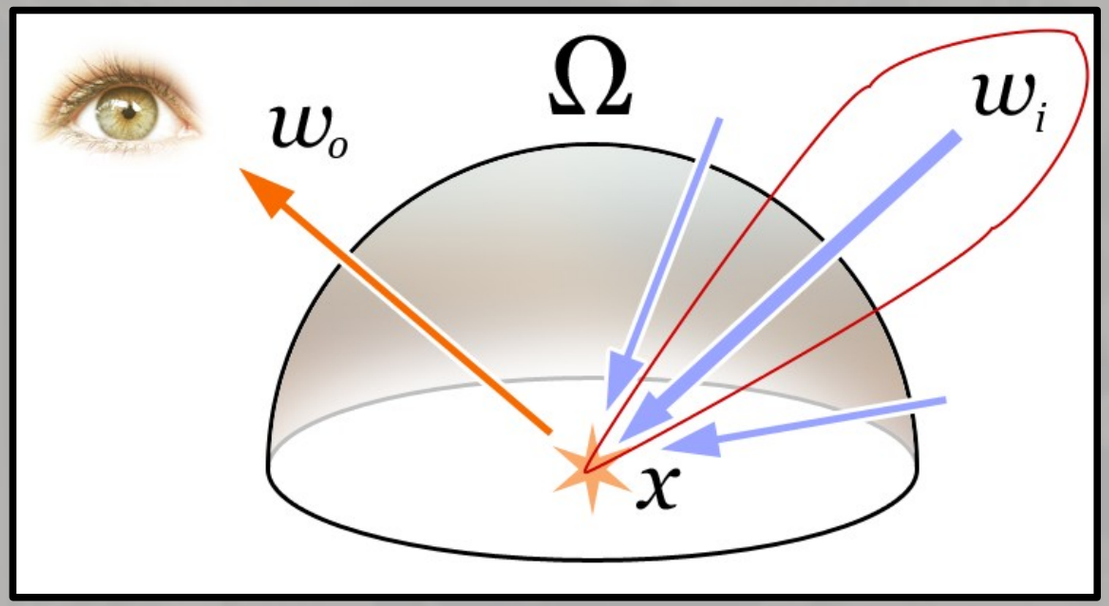
- $f_r(x, \omega_i, \omega_o)$ е BRDF-ът на материала в точката x



The Rendering Equation

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

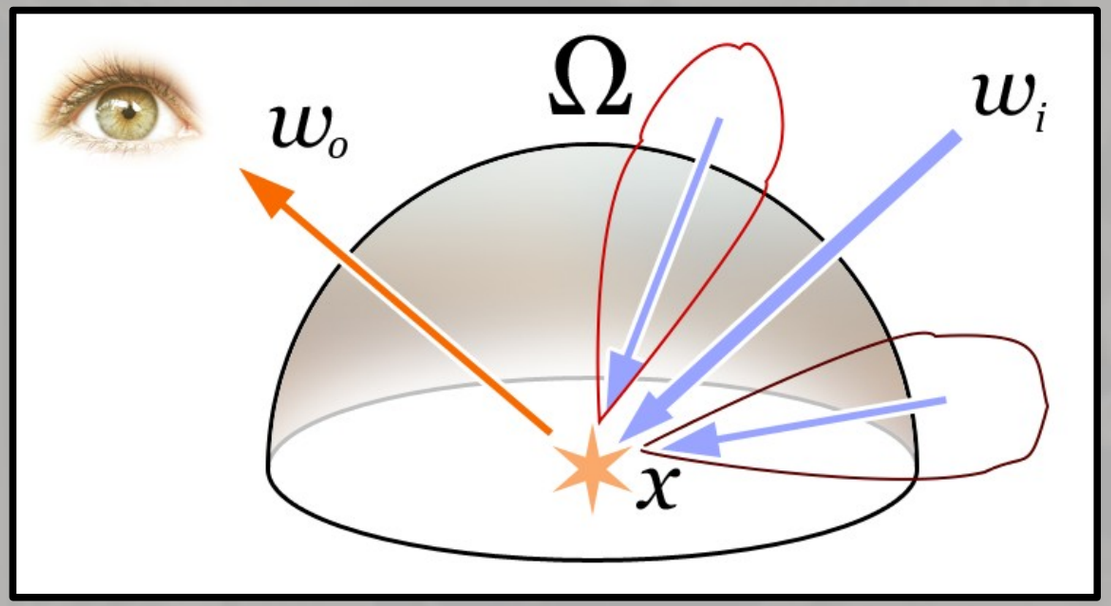
- $L_i(x, \omega_i)$ показва входящата светлина в точката x по някое направление ω_i. Тя може да идва както от лампа, така и от неизлъчващ обект (чрез отражение на светлината, т.е. от друг екземпляр на транспортното уравнение)



The Rendering Equation

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

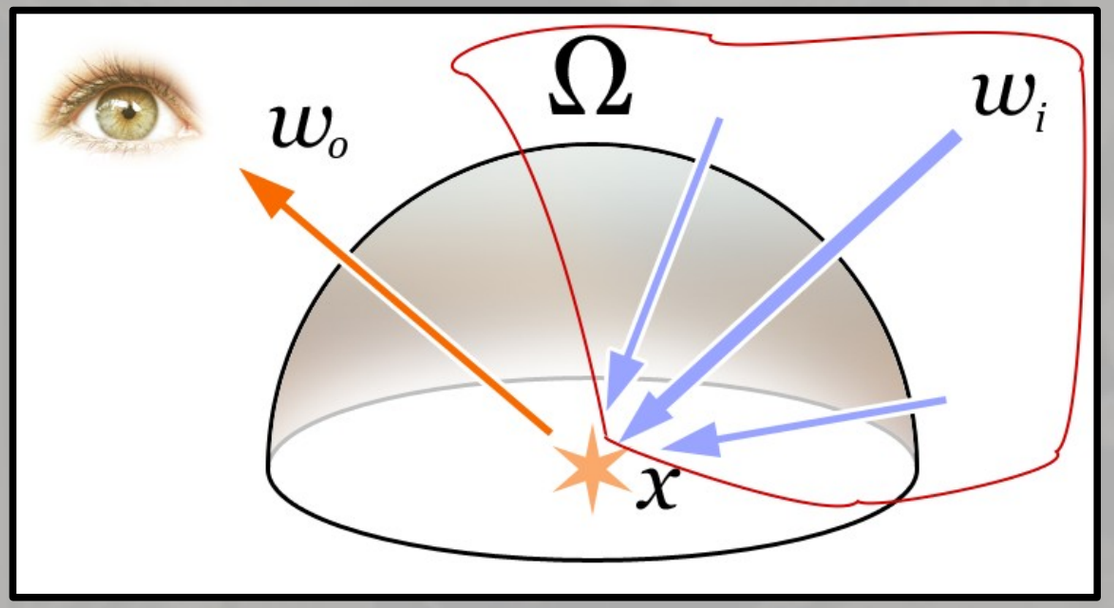
- Косинусът на ъгъла между ω_i и нормалата n в точката *x* участва, защото, под коси ъгли, входната светлина се разпределя на по-голяма площ. Така, на единица площ, осветяването е по-слабо, следователно и отражението трябва да е по-слабо.



The Rendering Equation

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

- 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*.

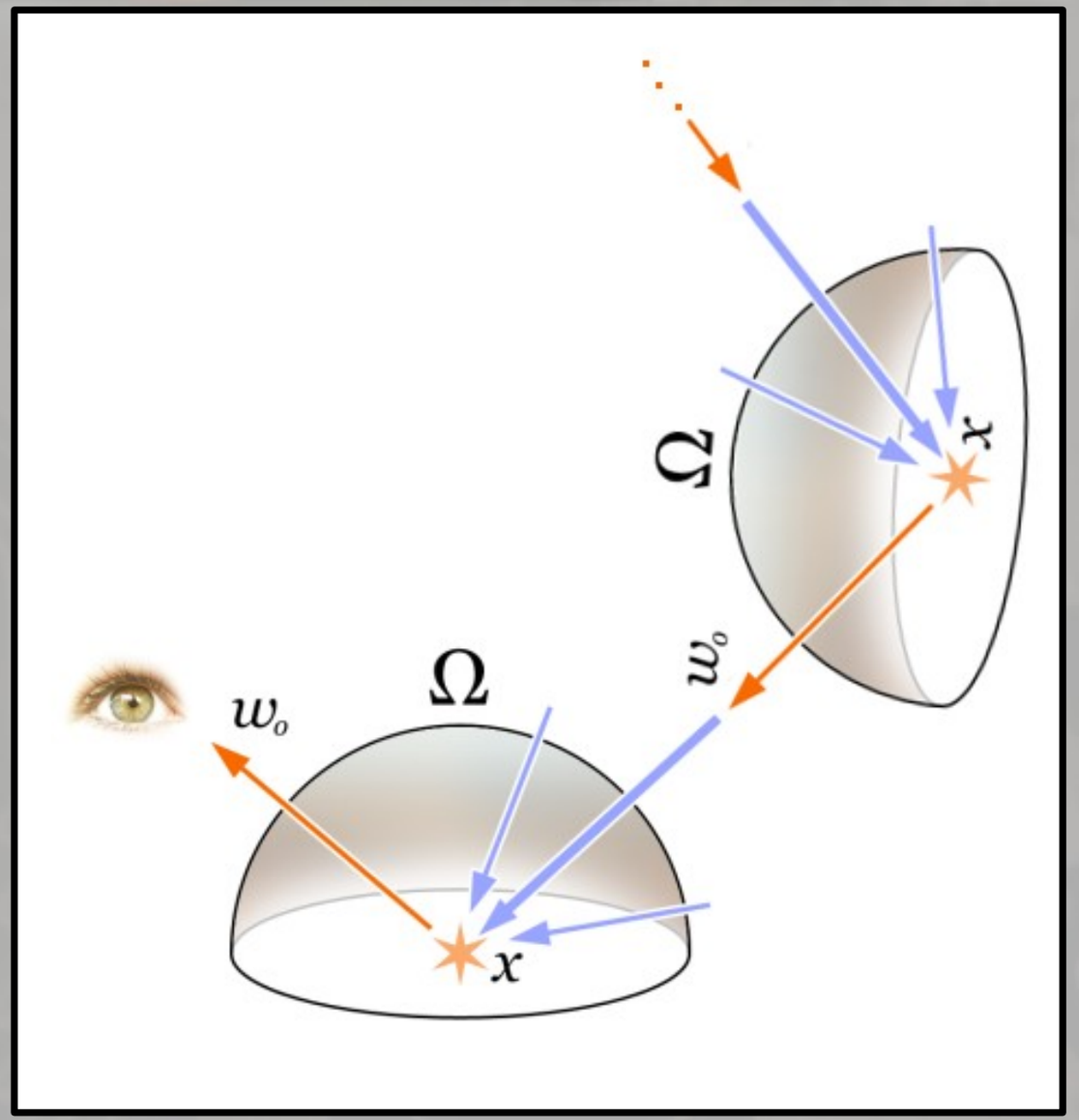


Особености на уравнението

- 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*'

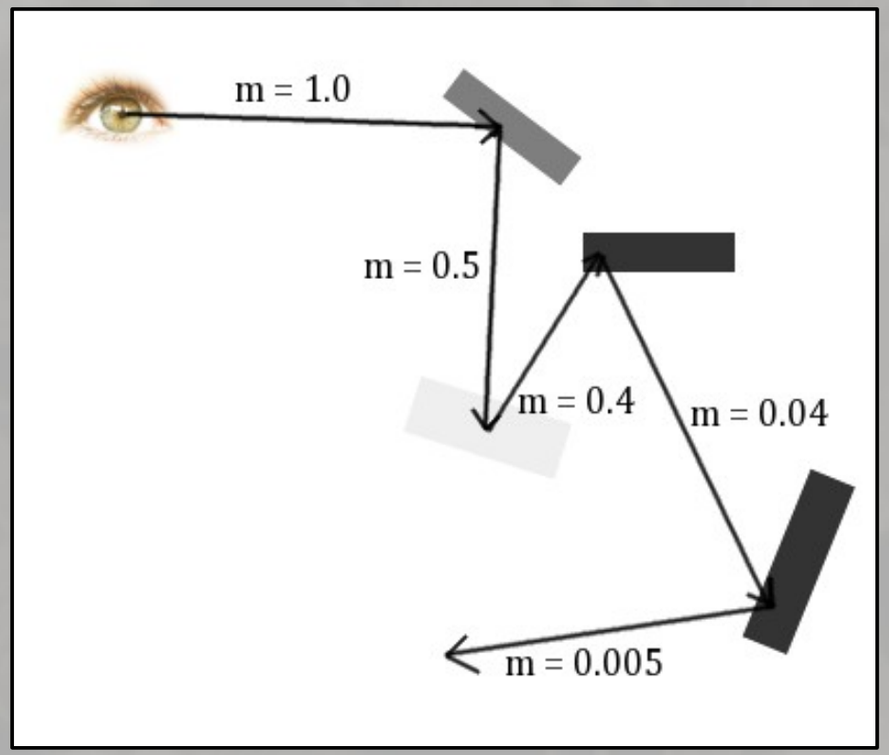


Рекурсивност

- 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 =-)

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

- Как може да се сметне нещо подобно?

Само в сто реда код :)

```
#include <math.h>   // smallpt, a Path Tracer by Kevin Beason, 2008
#include <stdlib.h> // Make : g++ -O3 -fopenmp smallpt.cpp -o smallpt
#include <stdio.h>  //        Remove "-fopenmp" for g++ version < 4.2
struct Vec {        // Usage: time ./smallpt 5000 && xv image.ppm
  double x, y, z;                  // position, also color (r,g,b)
  Vec(double x_=0, double y_=0, double z_=0){ x=x_; y=y_; z=z_; }
  Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
  Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
  Vec operator*(double b) const { return Vec(x*b,y*b,z*b); }
  Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
  Vec& norm(){ return *this = *this * (1/sqrt(x*x+y*y+z*z)); }
  double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; } // cross:
  Vec operator%(Vec&b){return Vec(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);}
};
struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} };
enum Refl_t { DIFF, SPEC, REFR };  // material types, used in radiance()
struct Sphere {
  double rad;       // radius
  Vec p, e, c;      // position, emission, color
  Refl_t refl;      // reflection type (DIFFuse, SPECular, REFRactive)
  Sphere(double rad_, Vec p_, Vec e_, Vec c_, Refl_t refl_):
    rad(rad_), p(p_), e(e_), c(c_), refl(refl_) {}
  double intersect(const Ray &r) const { // returns distance, 0 if nohit
    Vec op = p-r.o; // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
    double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
    if (det<0) return 0; else det=sqrt(det);
    return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
  }
};
Sphere spheres[] = {//Scene: radius, position, emission, color, material
  Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
  Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
  Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
  Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
  Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
  Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
  Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
  Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
  Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF) //Lite
};
inline double clamp(double x){ return x<0 ? 0 : x>1 ? 1 : x; }
inline int toInt(double x){ return int(pow(clamp(x),1/2.2)*255+.5); }
inline bool intersect(const Ray &r, double &t, int &id){
  double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
  for(int i=int(n);i--;) if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
  return t<inf;
}
Vec radiance(const Ray &r, int depth, unsigned short *Xi){
  double t;                               // distance to intersection
  int id=0;                               // id of intersected object
  if (!intersect(r, t, id)) return Vec(); // if miss, return black
  const Sphere &obj = spheres[id];        // the hit object
  Vec x=r.o+r.d*t, n=(x-obj.p).norm(), nl=n.dot(r.d)<0?n:n*-1, f=obj.c;
  double p = f.x>f.y && f.x>f.z ? f.x : f.y>f.z ? f.y : f.z; // max refl
  if (++depth>5) if (erand48(Xi)<p) f=f*(1/p); else return obj.e; //R.R.
  if (obj.refl == DIFF){                  // Ideal DIFFUSE reflection
    double r1=2*M_PI*erand48(Xi), r2=erand48(Xi), r2s=sqrt(r2);
    Vec w=nl, u=((fabs(w.x)>.1?Vec(0,1):Vec(1))%w).norm(), v=w%u;
    Vec d = (u*cos(r1)*r2s + v*sin(r1)*r2s + w*sqrt(1-r2)).norm();
    return obj.e + f.mult(radiance(Ray(x,d),depth,Xi));
  } else if (obj.refl == SPEC)            // Ideal SPECULAR reflection
    return obj.e + f.mult(radiance(Ray(x,r.d-n*2*n.dot(r.d)),depth,Xi));
  Ray reflRay(x, r.d-n*2*n.dot(r.d));     // Ideal dielectric REFRACTION
  bool into = n.dot(nl)>0;                // Ray from outside going in?
  double nc=1, nt=1.5, nnt=into?nc/nt:nt/nc, ddn=r.d.dot(nl), cos2t;
  if ((cos2t=1-nnt*nnt*(1-ddn*ddn))<0)    // Total internal reflection
    return obj.e + f.mult(radiance(reflRay,depth,Xi));
  Vec tdir = (r.d*nnt - n*((into?1:-1)*(ddn*nnt+sqrt(cos2t)))).norm();
  double a=nt-nc, b=nt+nc, R0=a*a/(b*b), c = 1-(into?-ddn:tdir.dot(n));
  double Re=R0+(1-R0)*c*c*c*c*c,Tr=1-Re,P=.25+.5*Re,RP=Re/P,TP=Tr/(1-P);
  return obj.e + f.mult(depth>2 ? (erand48(Xi)<P ?   // Russian roulette
    radiance(reflRay,depth,Xi)*RP:radiance(Ray(x,tdir),depth,Xi)*TP) :
    radiance(reflRay,depth,Xi)*Re+radiance(Ray(x,tdir),depth,Xi)*Tr);
}
int main(int argc, char *argv[]){
  int w=1024, h=768, samps = argc==2 ? atoi(argv[1])/4 : 1; // # samples
  Ray cam(Vec(50,52,295.6), Vec(0,-0.042612,-1).norm()); // cam pos, dir
  Vec cx=Vec(w*.5135/h), cy=(cx%cam.d).norm()*.5135, r, *c=new Vec[w*h];
#pragma omp parallel for schedule(dynamic, 1) private(r)       // OpenMP
  for (int y=0; y<h; y++){                       // Loop over image rows
    fprintf(stderr,"\rRendering (%d spp) %5.2f%%",samps*4,100.*y/(h-1));
    for (unsigned short x=0, Xi[3]={0,0,y*y*y}; x<w; x++)   // Loop cols
      for (int sy=0, i=(h-y-1)*w+x; sy<2; sy++)     // 2x2 subpixel rows
        for (int sx=0; sx<2; sx++, r=Vec()){        // 2x2 subpixel cols
          for (int s=0; s<samps; s++){
            double r1=2*erand48(Xi), dx=r1<1 ? sqrt(r1)-1: 1-sqrt(2-r1);
            double r2=2*erand48(Xi), dy=r2<1 ? sqrt(r2)-1: 1-sqrt(2-r2);
            Vec d = cx*( ( (sx+.5 + dx)/2 + x)/w - .5) +
                    cy*( ( (sy+.5 + dy)/2 + y)/h - .5) + cam.d;
            r = r + radiance(Ray(cam.o+d*140,d.norm()),0,Xi)*(1./samps);
          } // Camera rays are pushed ^^^^^ forward to start in interior
          c[i] = c[i] + Vec(clamp(r.x),clamp(r.y),clamp(r.z))*.25;
        }
  }
  FILE *f = fopen("image.ppm", "w");         // Write image to PPM file.
  fprintf(f, "P3\n%d %d\n%d\n", w, h, 255);
  for (int i=0; i<w*h; i++)
    fprintf(f,"%d %d %d ", toInt(c[i].x), toInt(c[i].y), toInt(c[i].z));
}
```

Връзка между Li и Lo

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

Color Li(x, w_i):

x' = trace(x, w_i)

$w_o = -w_i$

return Lo(x', w_o)

Rendering equation in pseudocode

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

```
Color Lo(x, w_o):
    Color sum = 0
    for i in 1..num_samples:
        w', pdf = generate_new_random_dir(x, w_o)
        sum += Fr(x, w_o, w') * Li(x, -w') * dot(w', N) * (1 / pdf)
    sum /= num_samples
    return Le(x, w_o) + sum
```

Algortihms 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

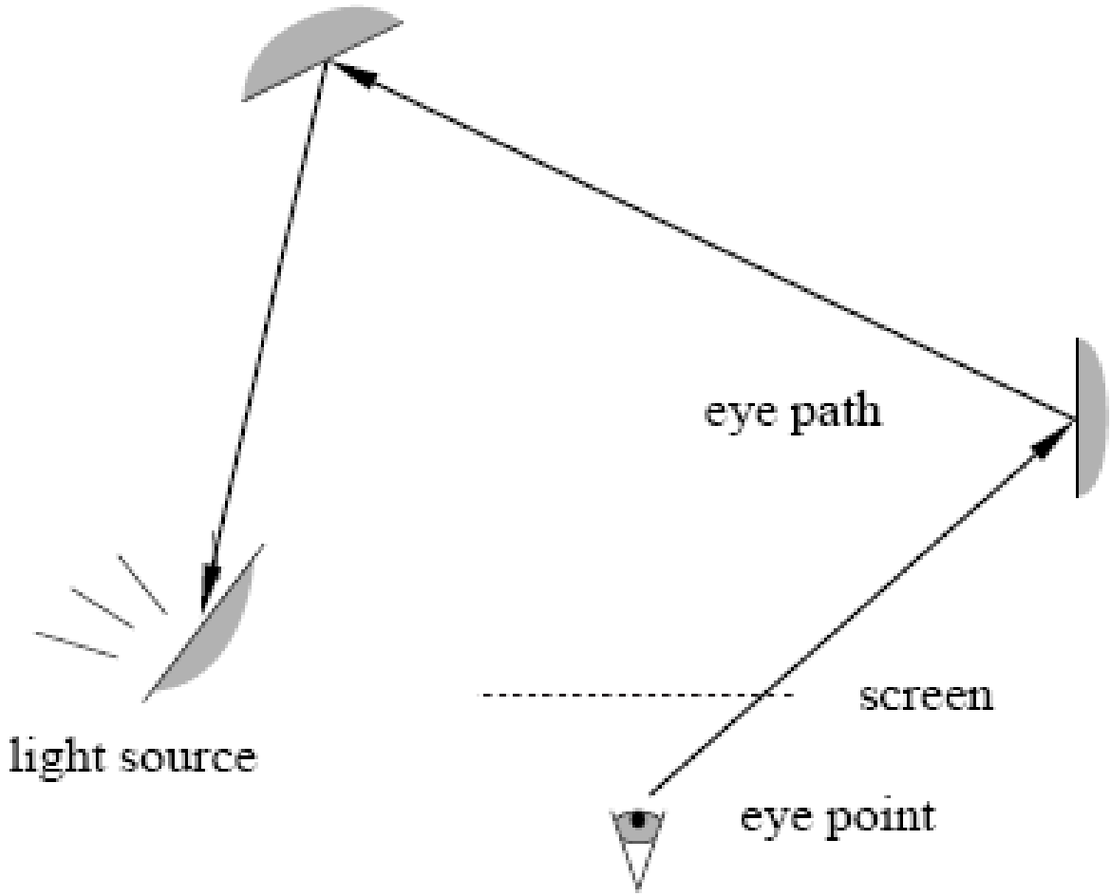
Algortihms for Global Illumination (II)

- 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

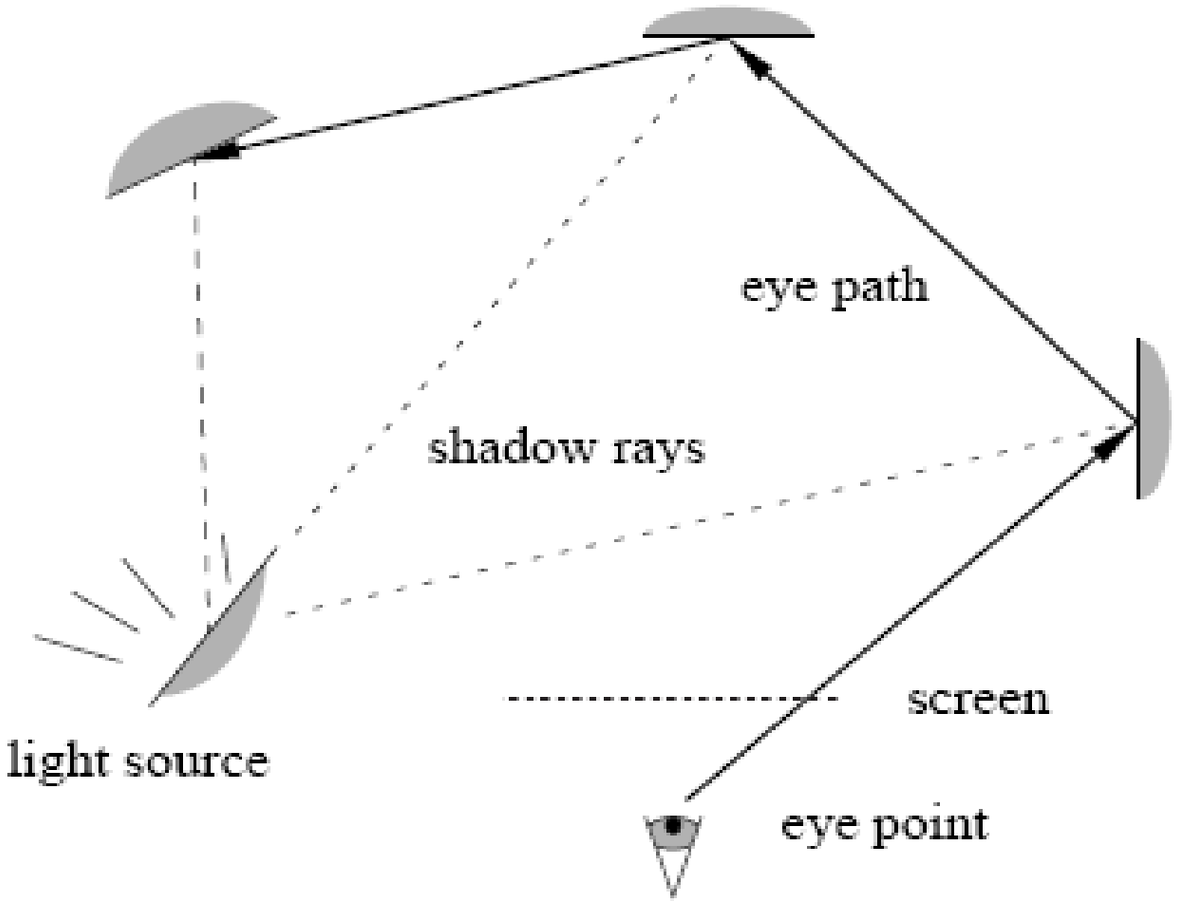
Algortihms for Global Illumination (III)

- 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

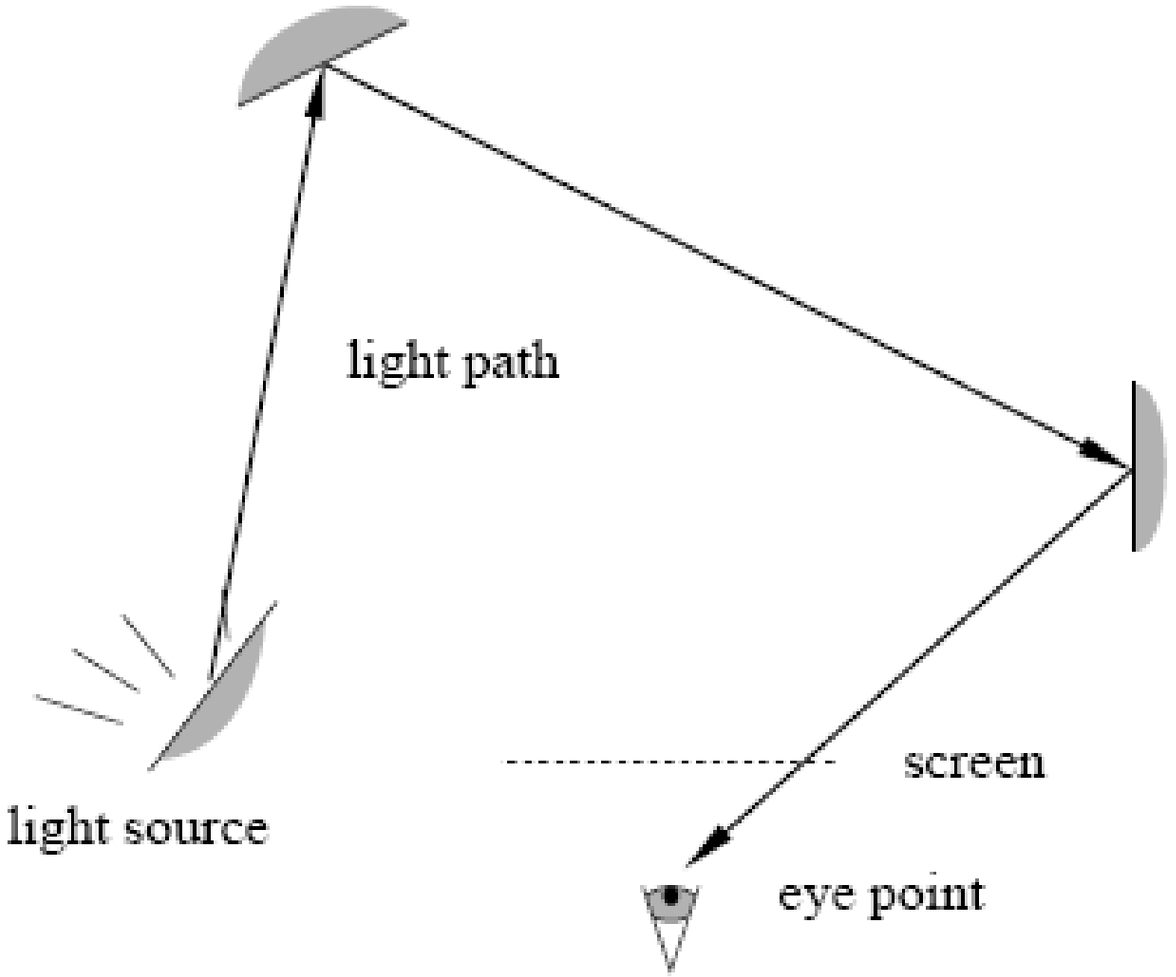
Path tracing



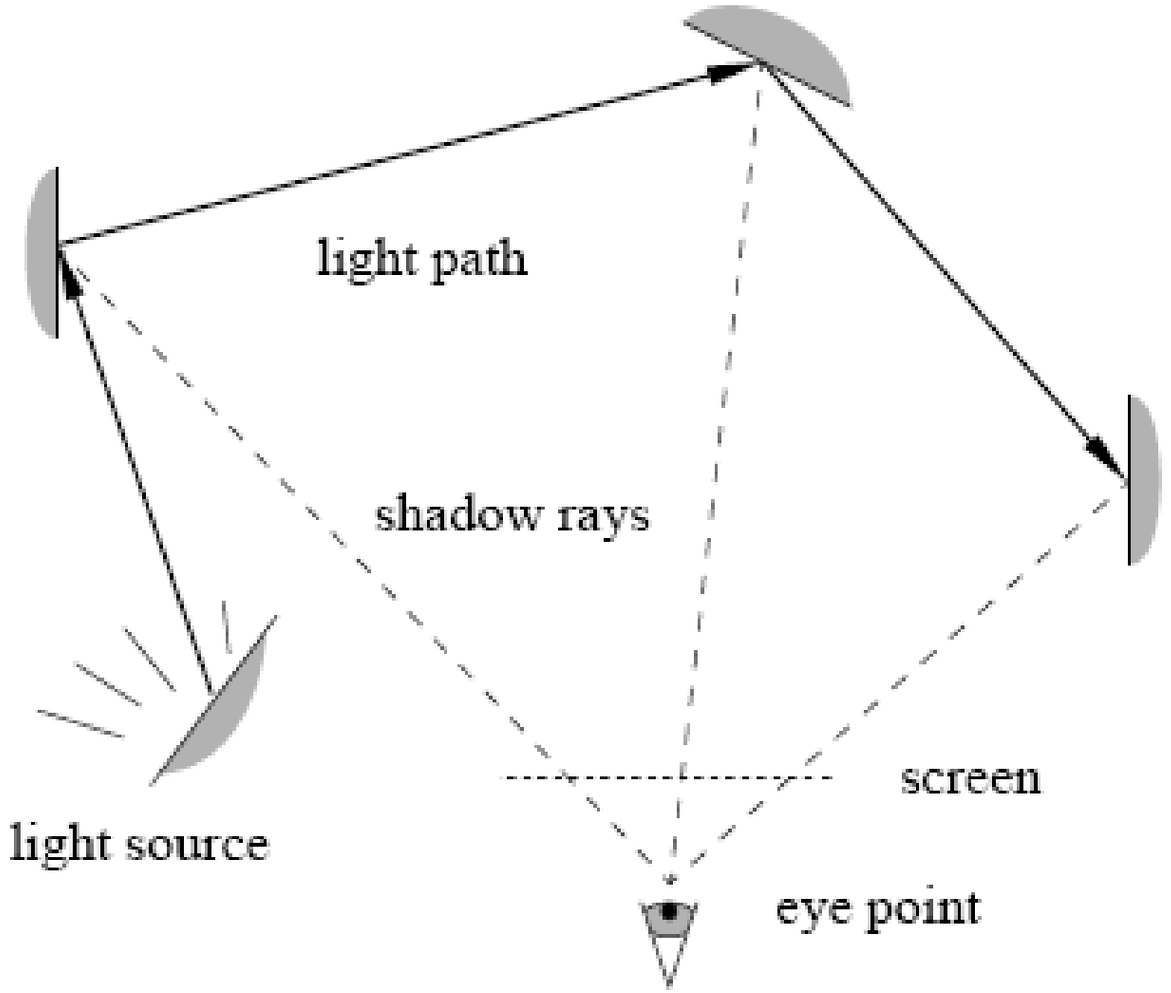
Path tracing (II)



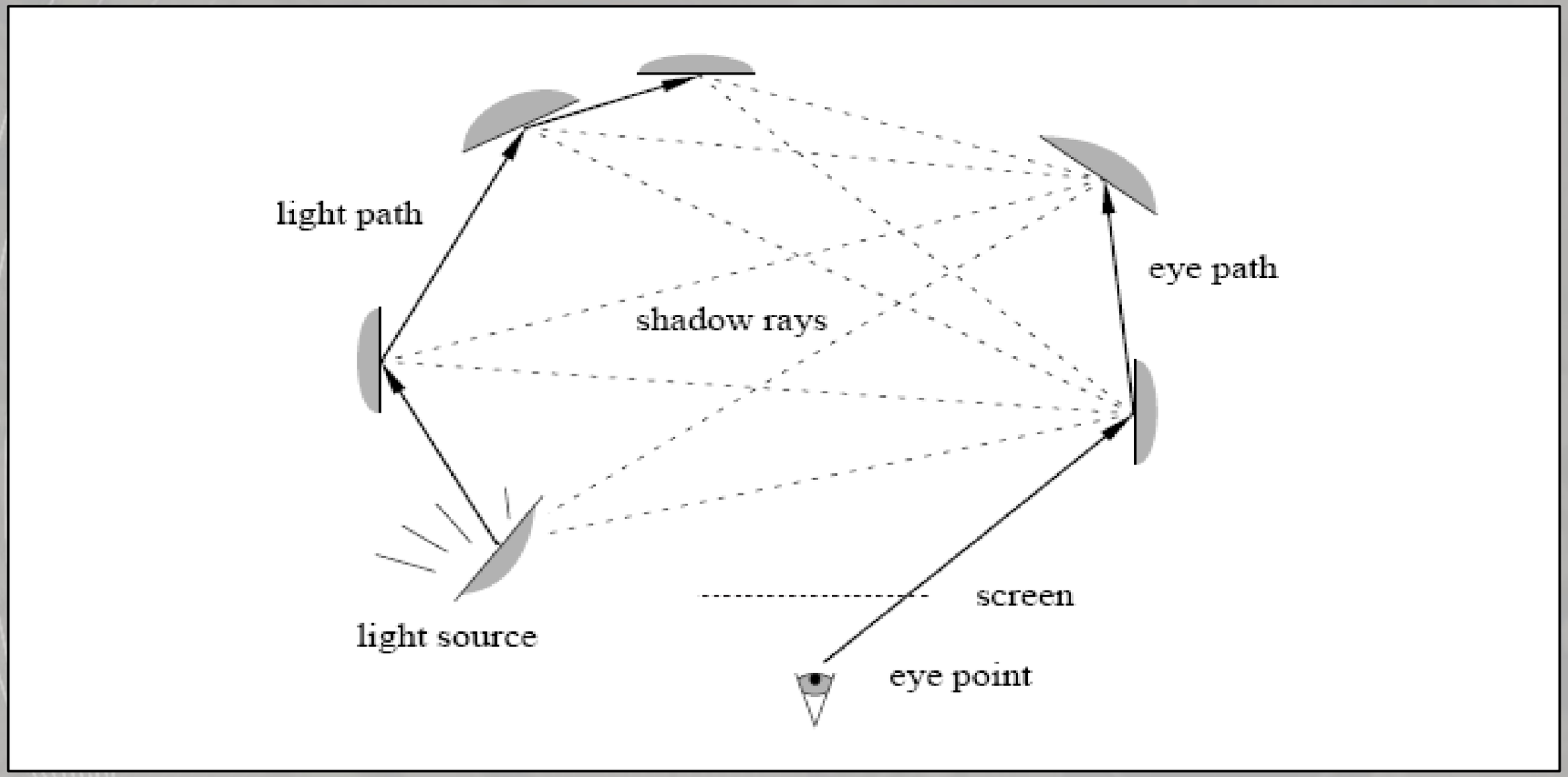
Light tracing



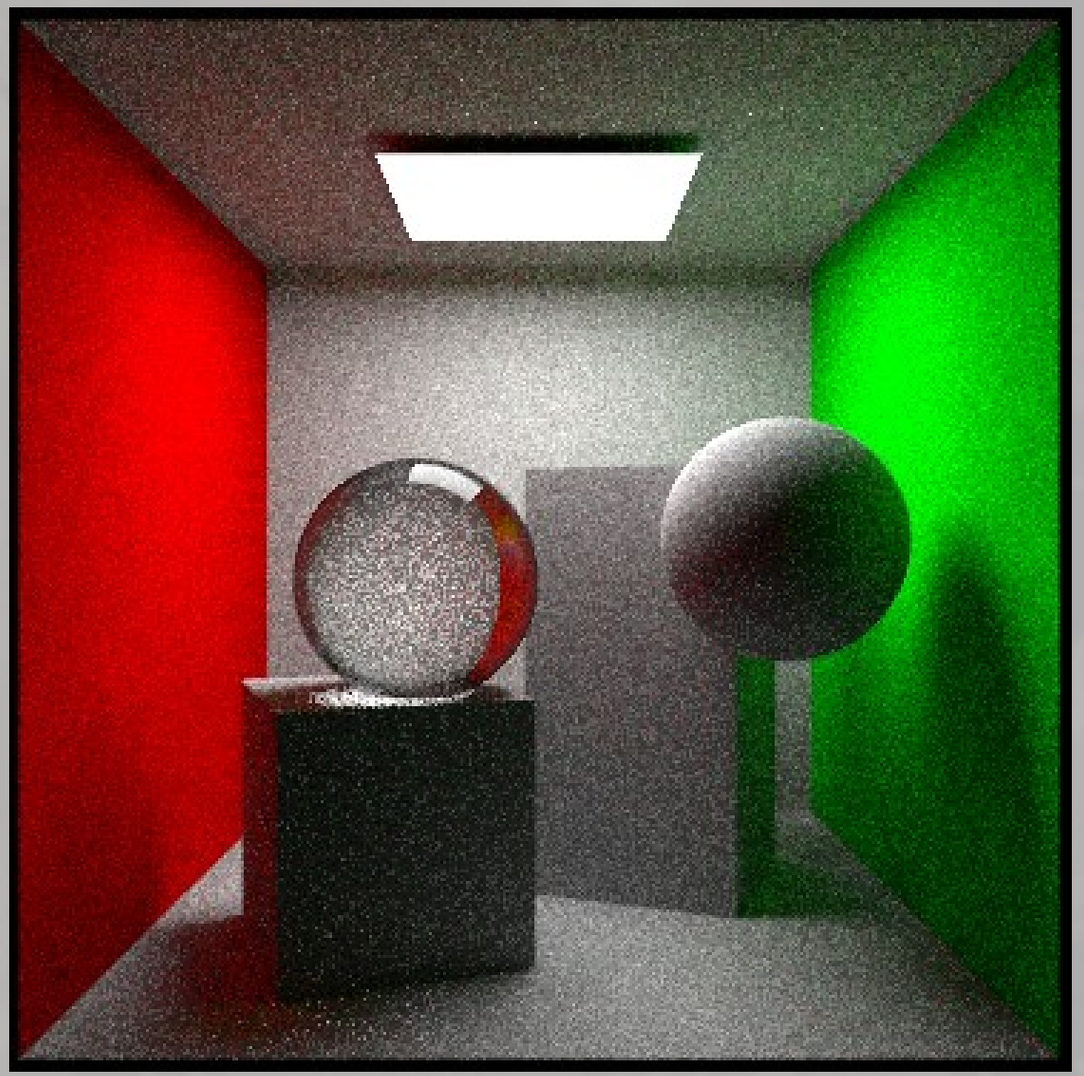
Light tracing (II)



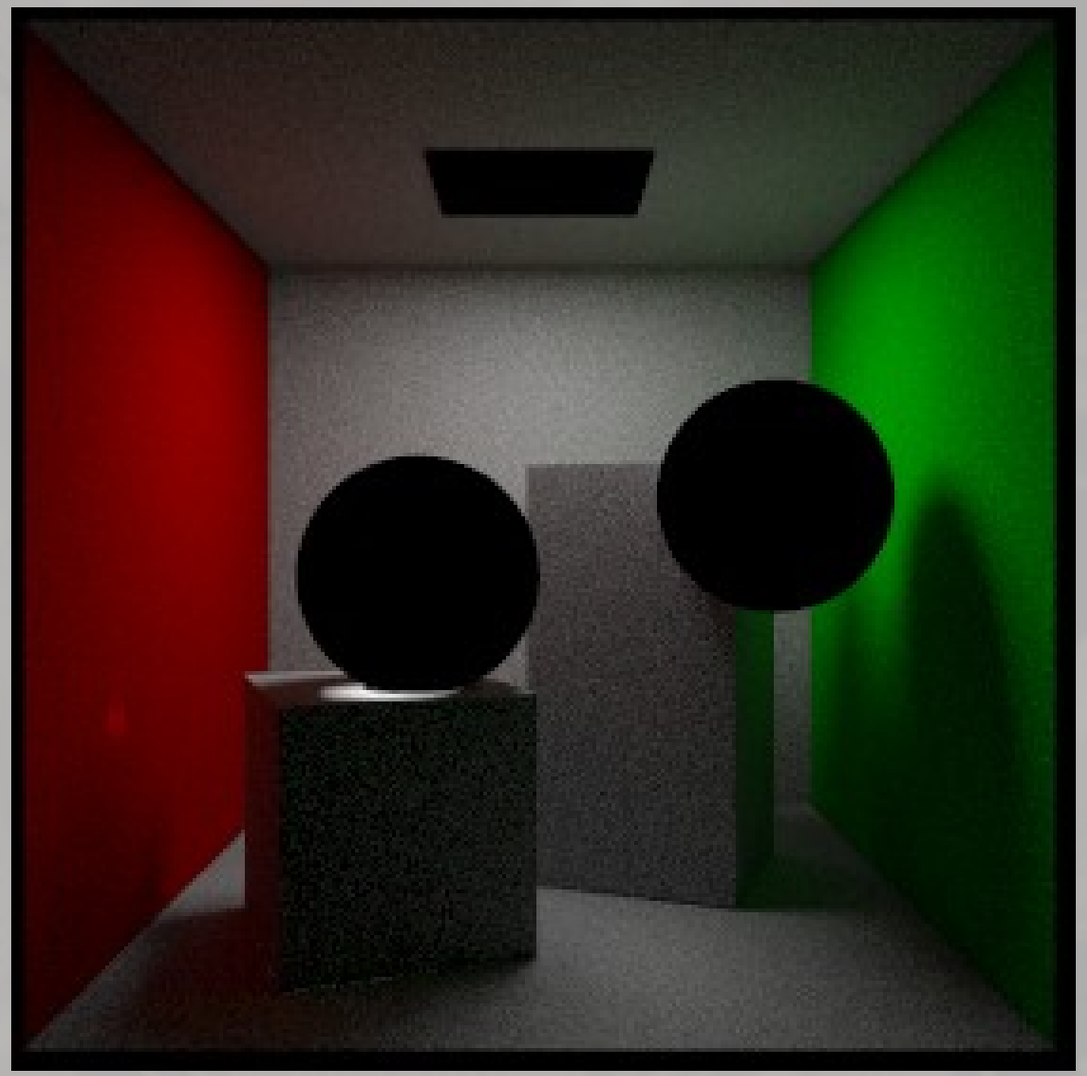
Bidirectional path tracing



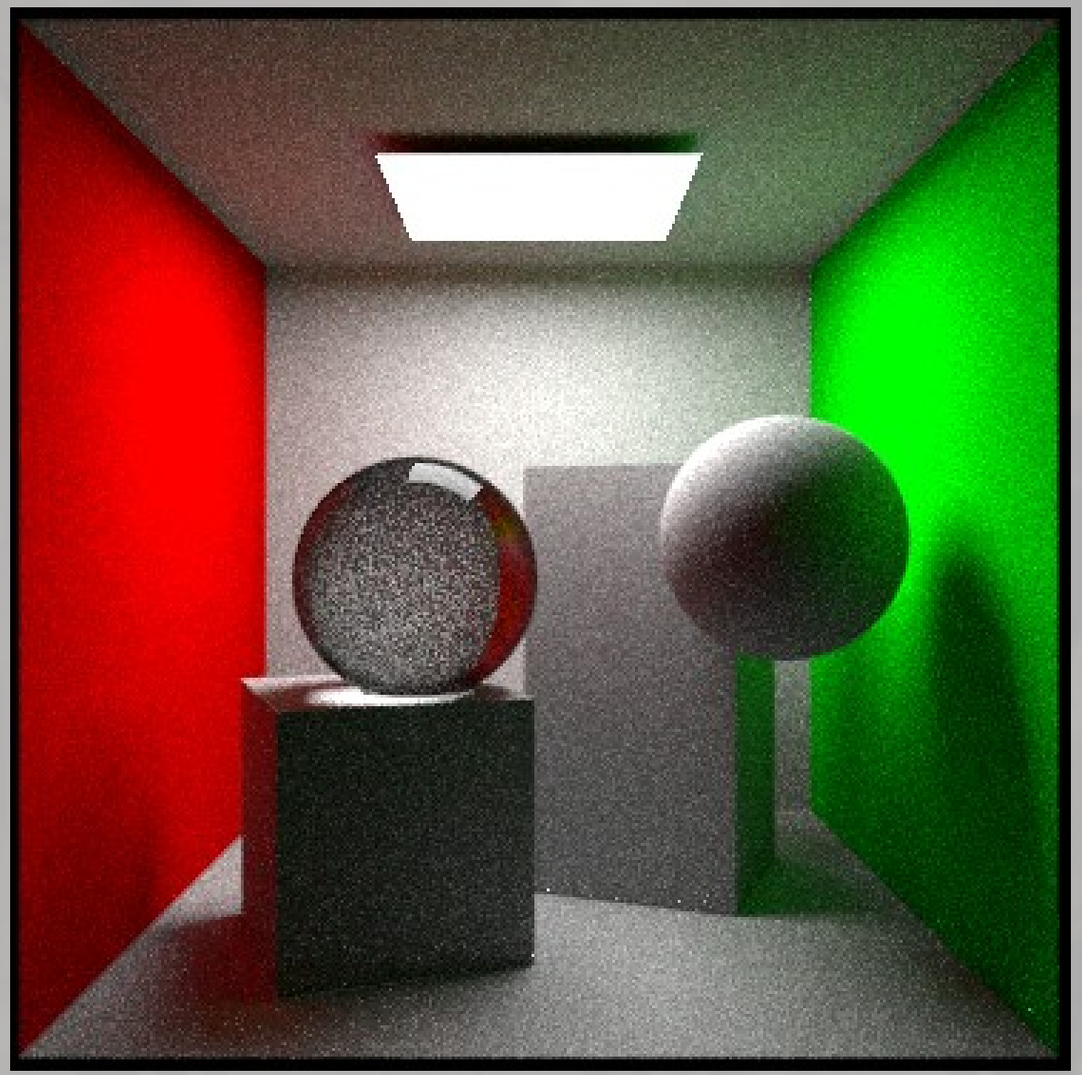
Path tracing result



Light tracing result



Bidirectional tracing result



Ресурси

- По-подробен разбор на различните алгоритми за глобално осветление, вижте лекцията от $CG^2$2012
- The holy bible по BDPT е тезиса на Eric Veach