# 3D графика и трасиране на лъчи v.4.0


http://raytracing-bg.net/

# Тема 3

Основни концепции в рендерирането и raytracing-а
Базова инфраструктура на raytracer-а

# Съдържание

- Новини
- Основни понятия в рендерирането
  - Сцена
  - Камера
  - Геометрия
  - Материали
  - Светлини
  - Обкръжение
- Базова инфраструктура на рейтрейсъра

# Съдържание (2)

- Показване на графика на екрана чрез SDL
  - Подредба на екранния буфер в паметта (RGB32)
  - Писане на минимално SDL приложение

# Новини

- Сменена зала
- Reminder: $CG^2$ 2015 този уикенд
- Гласувайте за име на рейтрейсъра
  - Анкетата е във сайта, http://raytracing-bg.net/

# Основни понятия

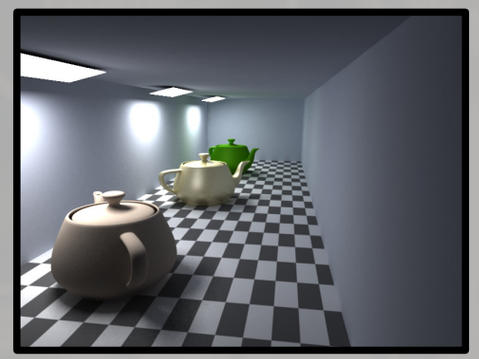
- Сцената включва всички останали елементи
- Сцената е „виртуалната реалност“, която рендерираме
- Говорим за „архитектурна сцена“, „интериорна сцена“, „тежка сцена“, „светла сцена“ и пр.
- Сцените обикновено се записват в някакъв контейнерен формат, който държи цялата сцена (с някои изключения) на едно място
  - Например, *.MAX файлове за 3ds Max, *.POV за POV-Ray

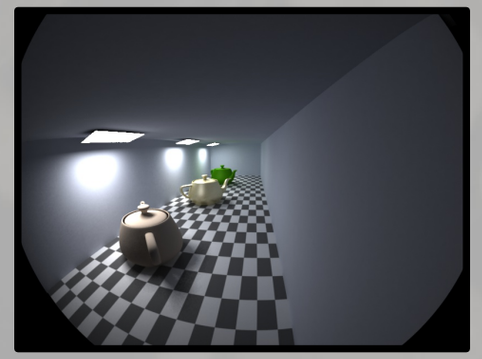
# Камерата

- Камерата представлява „окото“ на наблюдателя
  - Еквивалентна е на фотоапарата във фотографията
    - С тази разлика, че е невидима
- При анимация, може да имаме няколко камери в сцената, но само една е активна за даден кадър
- При raytracing-а, камерата е мястото, от което тръгват лъчите
- Има различни видове камери

# Видове камери

- Нормална (rectilinear)
- Ортографическа
- Рибешко око (fisheye)
- Други (изкуствени) – сферична, цилиндрична, ...



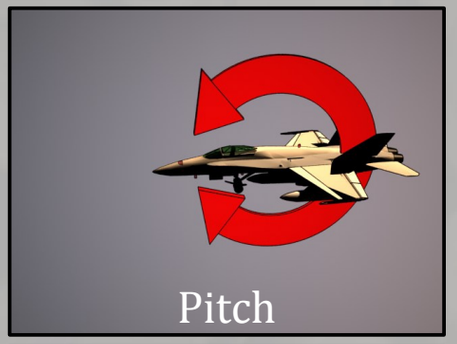


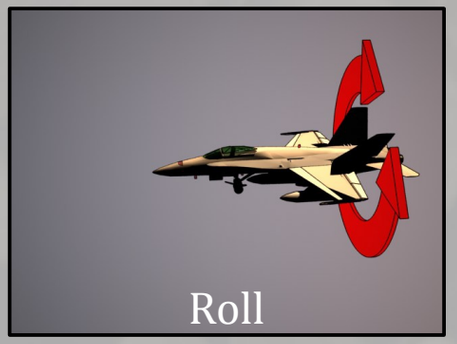
# Характеристики на камерата

- Позиция (3D вектор)
- Посока – може да се зададе по различни начини:
  - Чрез ъгли (yaw/pitch/roll система)
  - Чрез ос на въртене и ъгъл на завъртането около оста
  - Чрез позиция на целта (look-at система)
    - При нея се налага да се посочи и „up-vector“. Той показва в каква посока е „нагоре“, тъй като иначе не може да симулира ефекта на „roll“ ъгъла
- Зрителен ъгъл (FOV (Field-of-View))
- Aspect ratio
- Дълбочина на полето (DOF (Depth-of-Field))

# Yaw/pitch/roll система

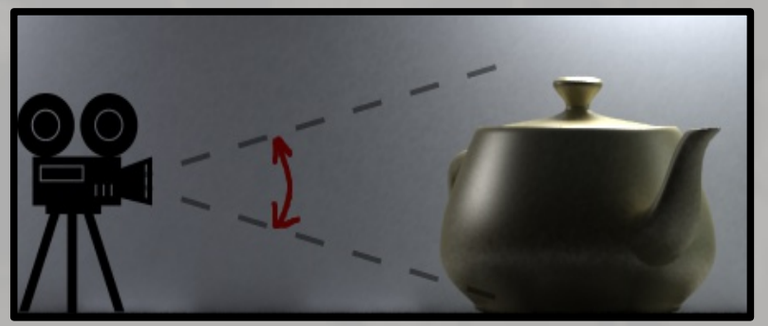
- По аналогия с контролите на самолет
  - Yaw ъгълът указва посоката „напред“ на самолета спрямо „земята“; [0° .. 360°)
  - Pitch ъгълът указва „изкачването“ на самолета спрямо земята; [-180° .. 180°]
  - Roll ъгълът указва завъртането на самолета спрямо надлъжната му ос; [0° .. 360°)
  - Прилагането на ротациите е в реда roll, pitch, yaw

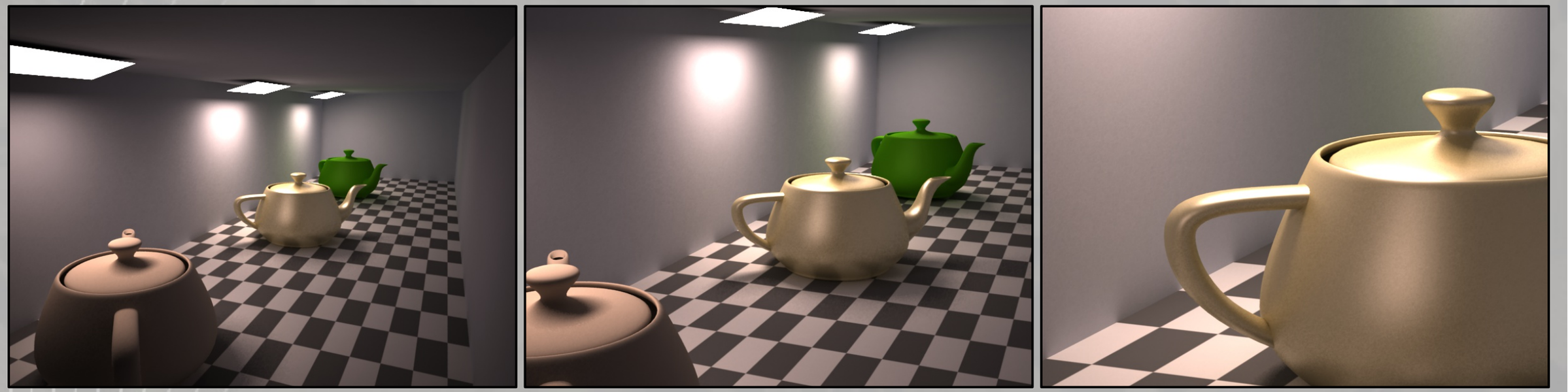



# FOV

- Зрителният ъгъл определя колко „широк“ е погледа на камерата
- Обикновено се дефинира като:
  - Диагонален ъгъл, в градуси
  - Фокусно разстояние на обектива, в милиметри (за класически фотоапарат с 35mm филм)
  - Двете са обратно-пропорционални: по-голямо фокусно разстояние = по-малък зрителен ъгъл



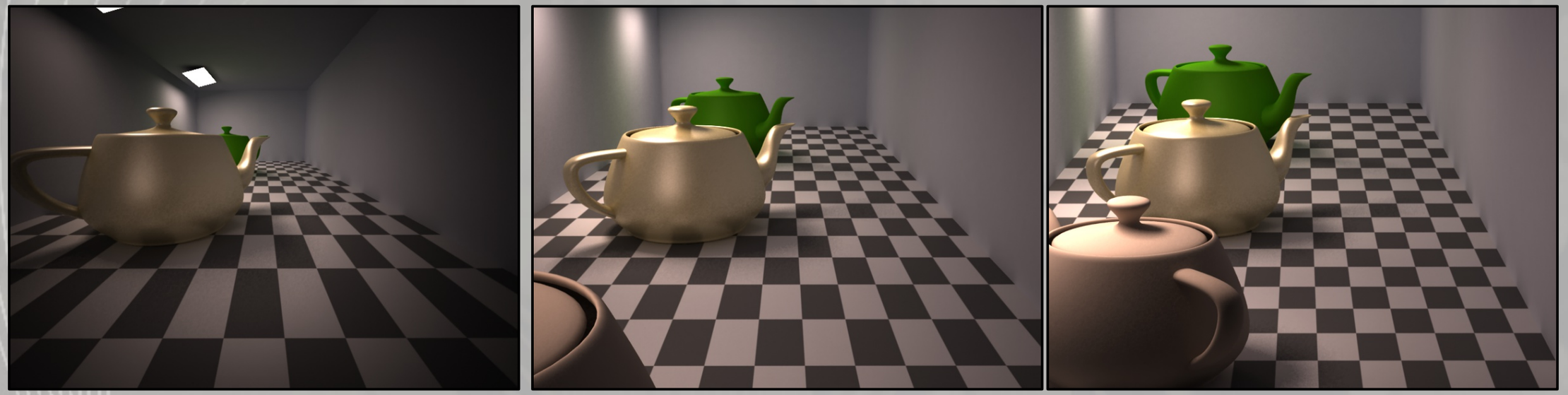
# FOV



- 30mm, 50mm, 135mm
- 72°, 47°, 18°

# FOV

- Усещането за перспектива също зависи от зрителния ъгъл, тъй като мени отношението между видимия размер на нееднакво отдалечените обекти
- В примера, средният чайник е еднакво голям (като пиксели); фокусните разстояния са 17, 45 и 105mm.



# FOV

- За „нормален“ FOV се счита ≈53° (около 50mm)
- При широките ъгли, изображението става много неестествено след 120°
  - 180° е математически невъзможно при правоъгълна камера; на практика всичко над 160° е неизползваемо
  - В реалния свят, най-широките обективи са до 14mm (114°)
  - Ако целим много широки ъгли, по-добре да се ползва fisheye камера

# Отношение (Aspect ratio)

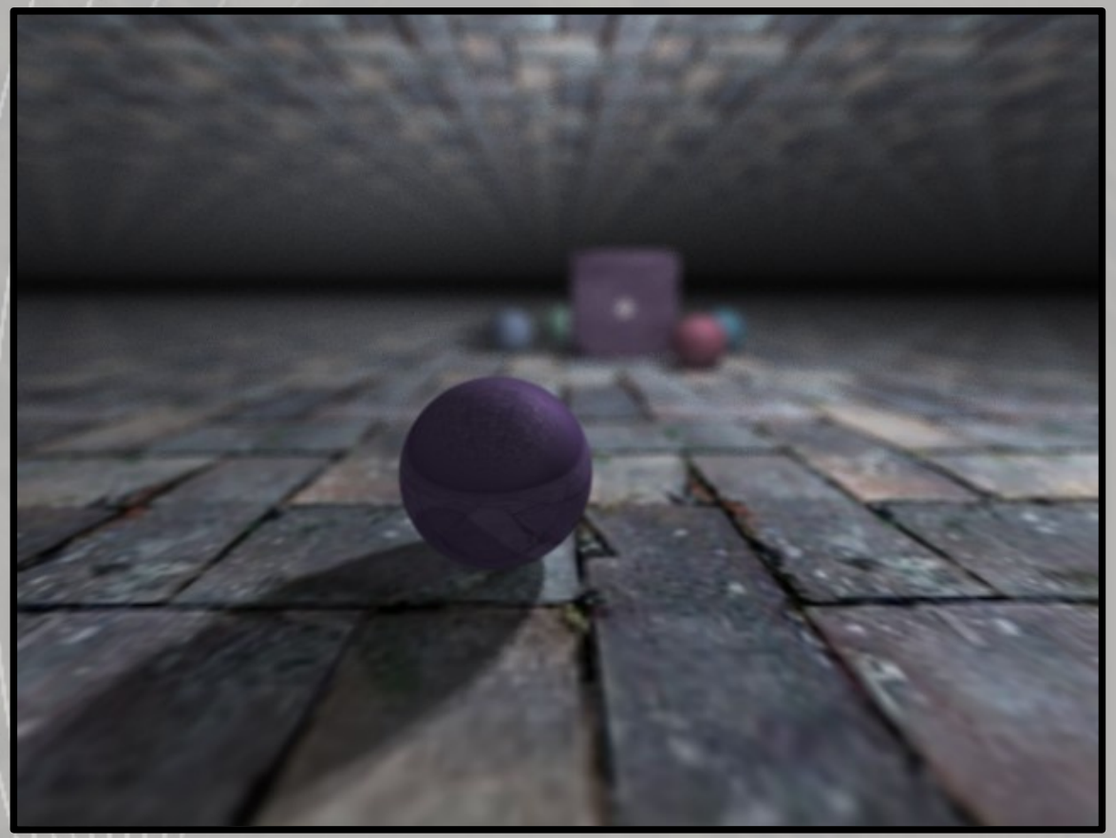
- Има се предвид отношението Ширина:Височина на кадъра
  - Това не е същото като отношението на изходната картинка в пиксели!
  - Трябва да съвпада с отношението на целевата медия (монитор, хартиена снимка...)
    - Например, 640x480 кадър (4:3 отношение) може да се покаже на wide монитор (16:10), и за да не изглежда разтеглено, трябва на камерата да се укаже aspect = 1.6
  - Разбира се, най-добре е всичко да съвпада (квадратни пиксели)

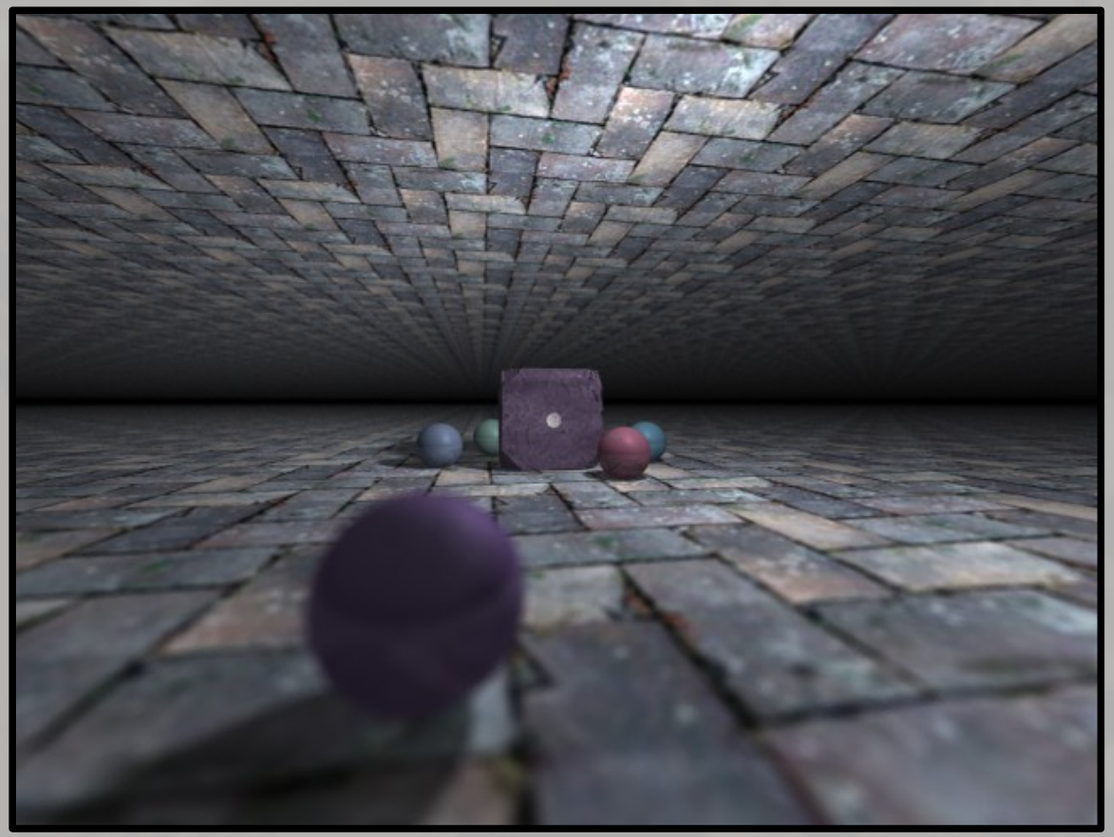
# Дълбочина на полето (DOF)

- Симулира фокусираните и разфокусираните области при фотографията
  - Имаме фокусна равнина, в която предметите са на фокус; колкото повече се отдалечаваме от нея, толкова повече се разфокусират
  - Дълбочината на полето се определя от областта, която човек възприема за „добре фокусирана“
  - Дълбочината зависи от редица фактори
    - Най-вече от избраната бленда на фото-обектива (aperture), но има и други фактори

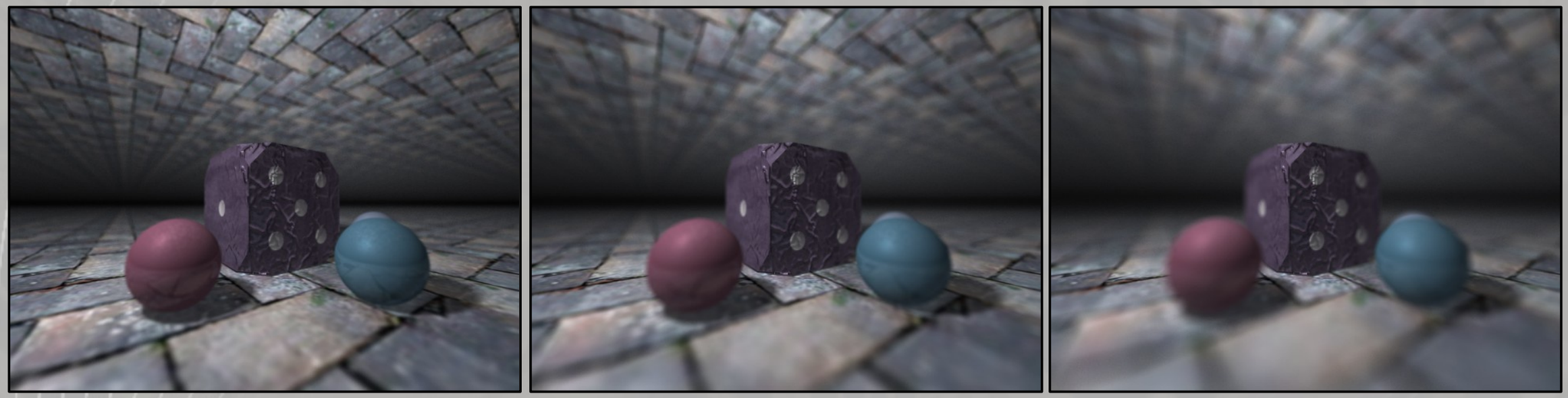
# Дълбочина на полето (DOF)

- Фокусиране на близък и далечен обект:





# Дълбочина на полето (DOF)



- Различните бленди (aperture) дават различни дълбочини при еднакво разстояние до фокуса.
- Тук са показани, от ляво надясно: f/8, f/4, f/2

# Дълбочина на полето (DOF)

- Дълбочината на полето зависи от:
  - Разстоянието до фокусната равнина: когато фокусът е далече, дълбочината е по-голяма
  - Блендата – ниските f-числа съответстват на плитък DOF
  - Зрителният ъгъл: при по-широк FOV, дълбочината расте
  - При фотоапаратите: големината на филма/сензора. По-голям филм/сензор съответства на по-плитък DOF
- По-нататък ще покажем как се симулират отделните свойства на фотокамерата от гледна точка на DOF

# Геометрия

- Под „геометрията“ ще разбираме описанието на формите на всички обекти в сцената
- Когато трасираме един лъч, ние търсим къде той се пресича с геометрията
- В raytracing-а, геометрията може да е съставена от разнородни по вид геометрични обекти
  - Триъгълни мрежи
  - Математически обекти (сфера, равнина, ...)
  - Релефни карти
  - и т.н.

# Геометрия

- Общото между всички тях е, че предоставят алгоритъм за пресичане с лъч:

```
class IntersectableInterface {
public:
   virtual
   bool intersect(Ray ray, IntersectionInfo &info) = 0;
};
```

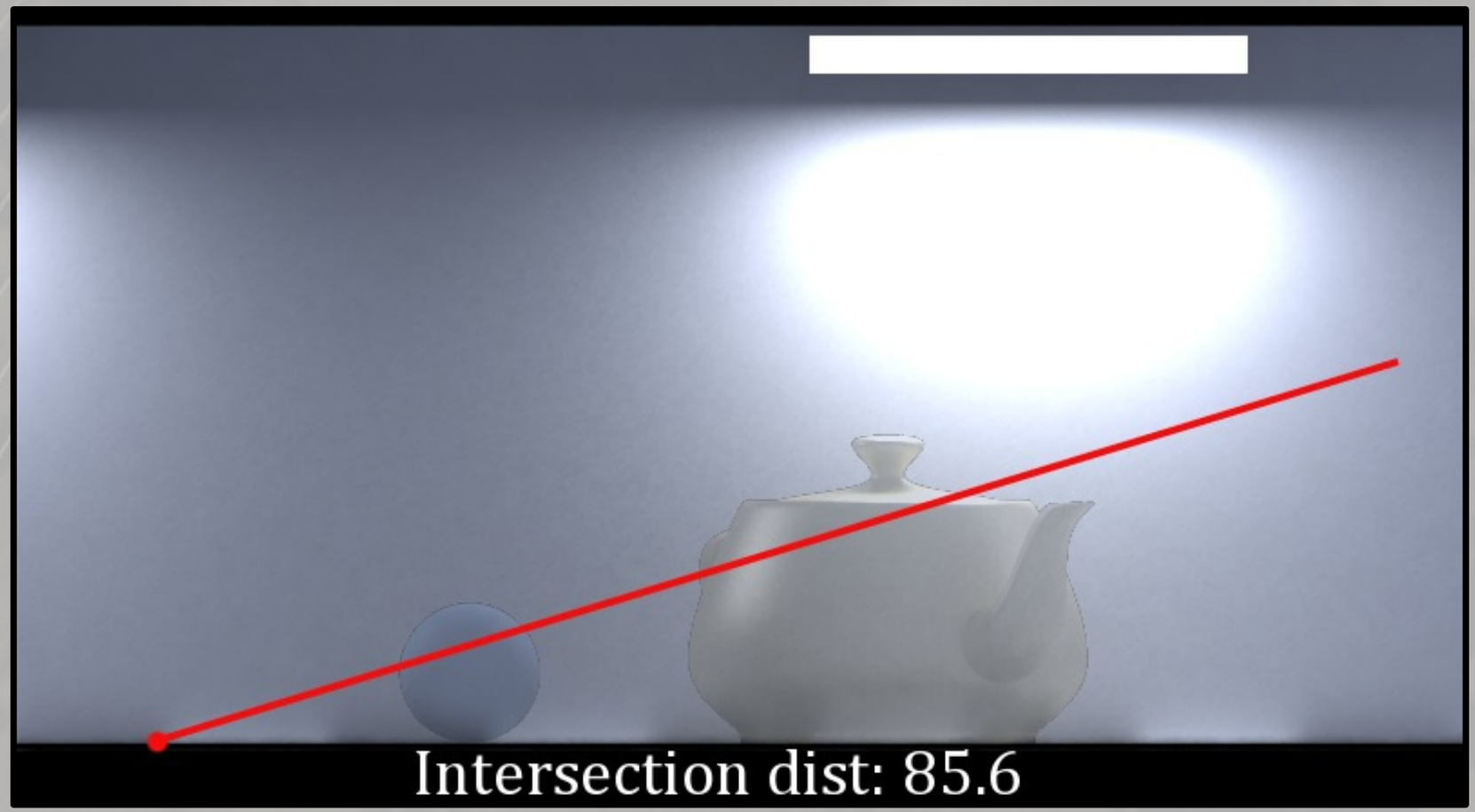
- Връща true, ако лъчът пресича геометрията, и попълва структурата *info* при пресичане
- В *info* се запазват допълнителни данни за пресечната точка: разстояние до нея, 3D позиция, нормали, ...

# Геометрия

- Наивният raytracing алгоритъм пресича даден лъч с геометрията, като обходи всички обекти и пресече лъча с всеки един от тях
- Търси се тази пресечна точка, която е най-близо до камерата
- Ще илюстрираме с пример

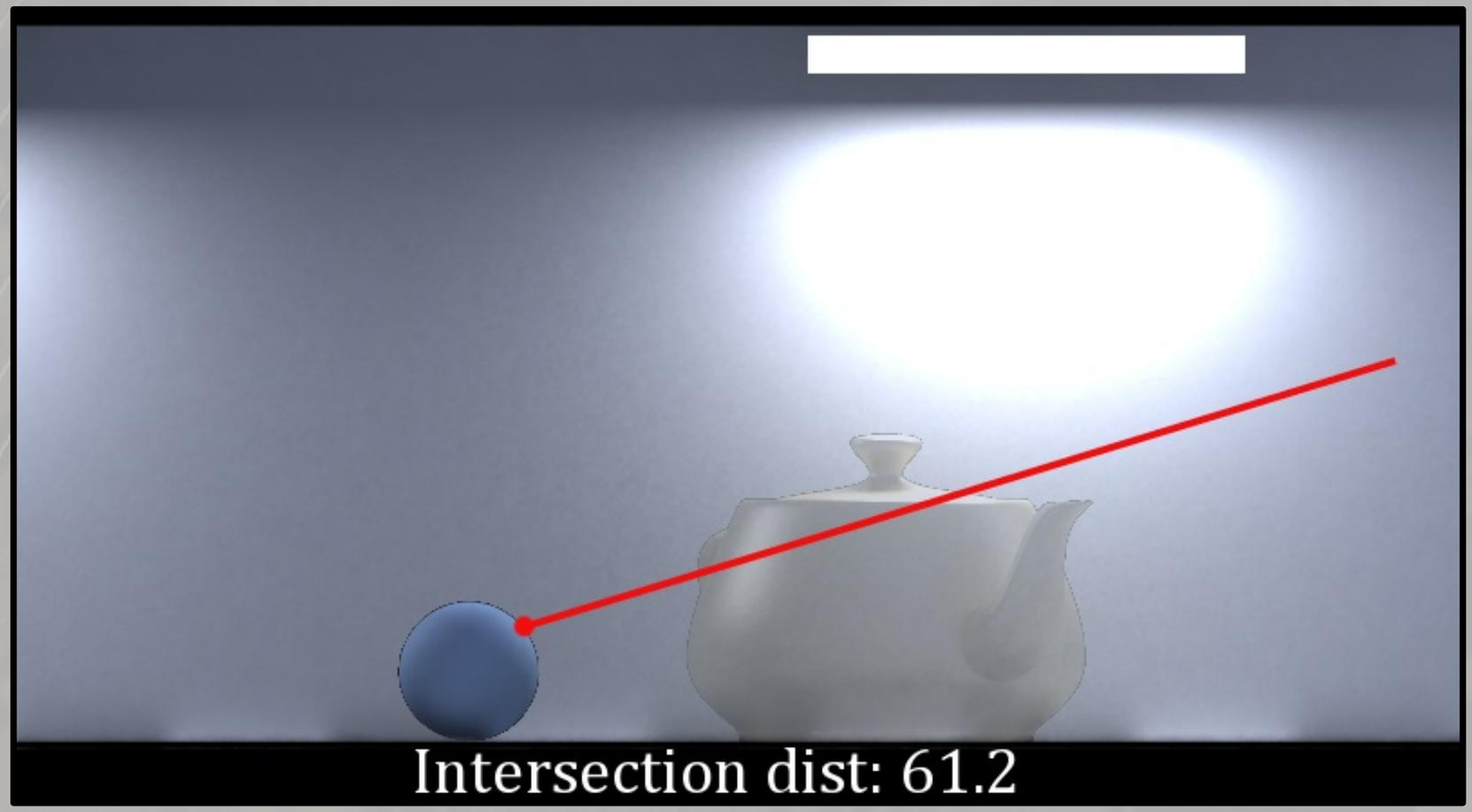
# Пресичане с геометрията

- Пресичане с равнината



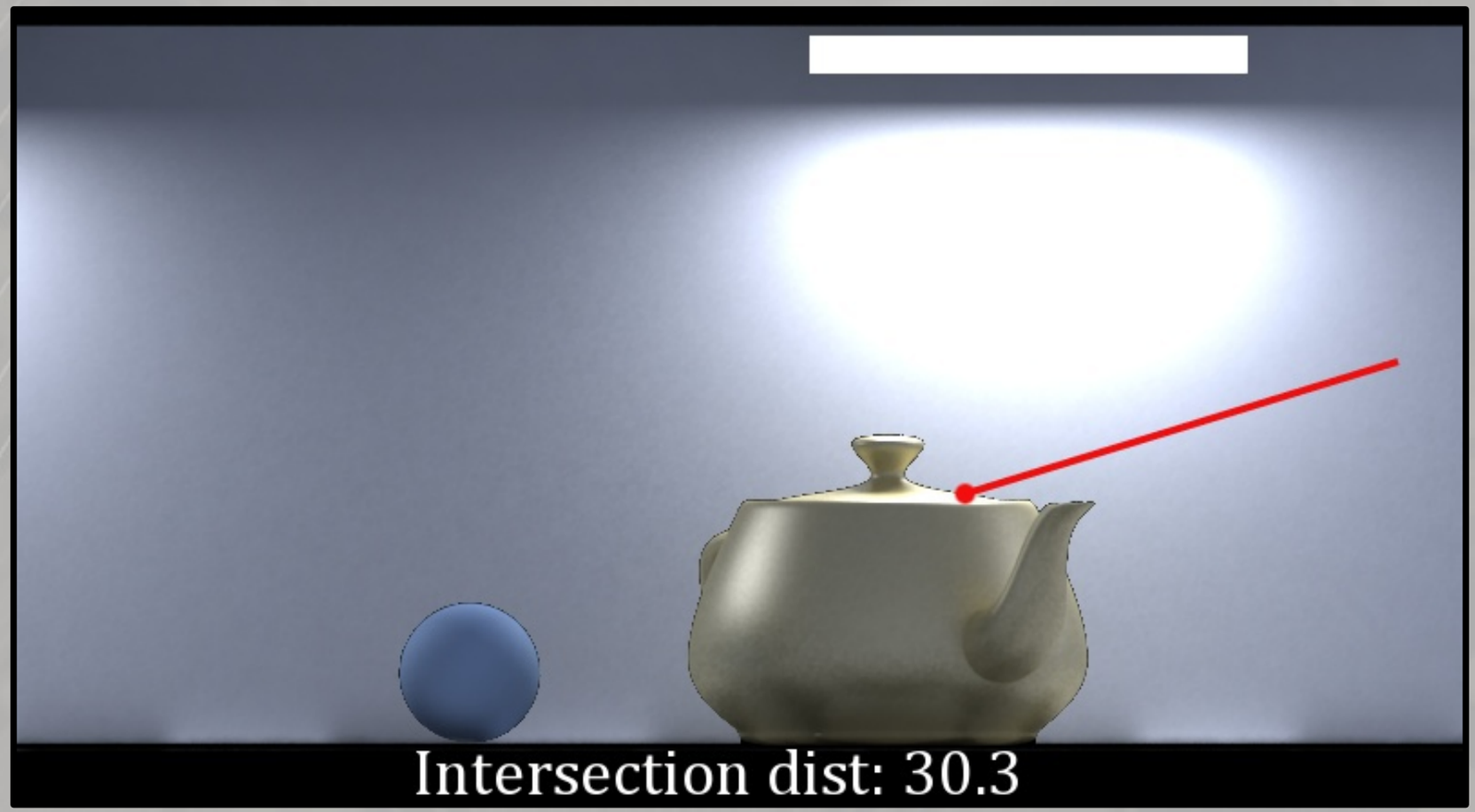
# Пресичане с геометрията

- Пресичане със сферата



# Пресичане с геометрията

- Пресичане с чайника

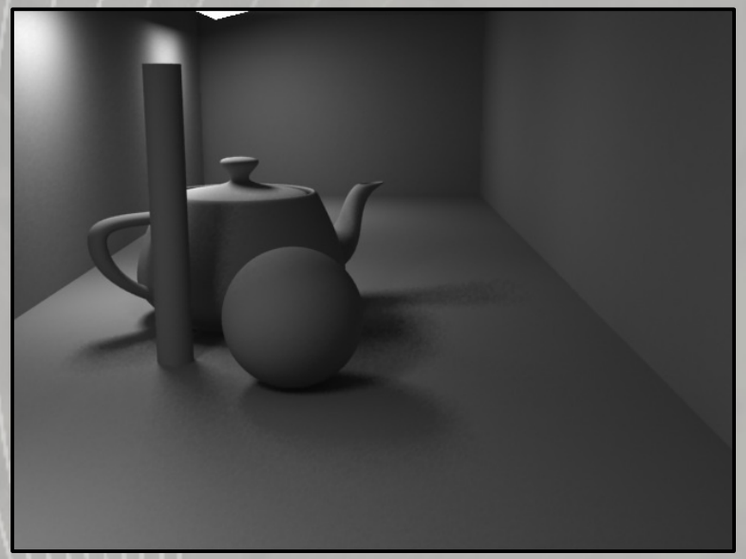


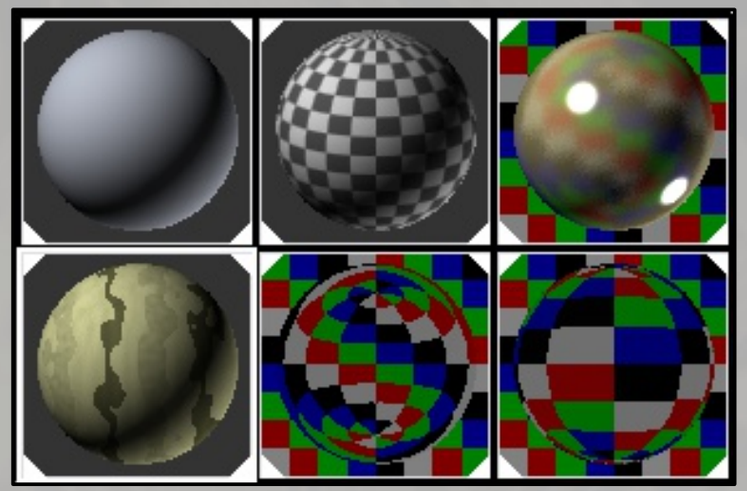
# Пресичане с геометрията

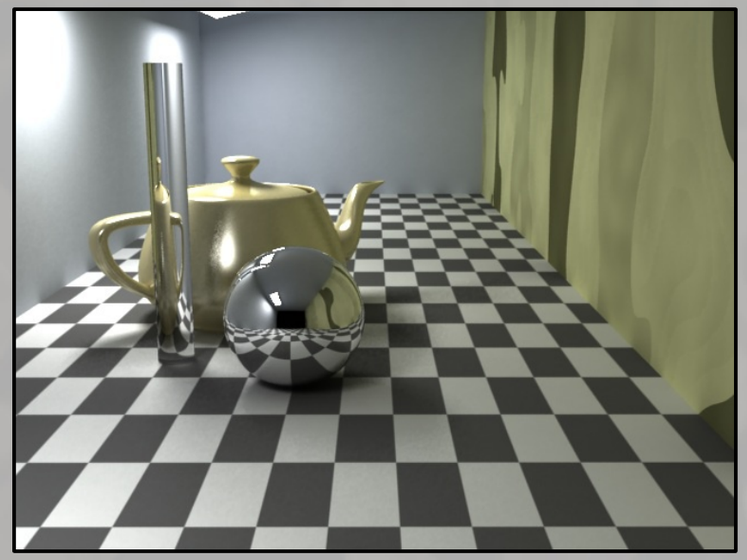
- Raytracing-а е много гъвкав алгоритъм, защото позволява да добавим поддръжка за нов тип обекти, като просто реализираме intersect() функцията
- Пресичането отнема основна част от времето на един raytracer (обикновено над 50%)
- Информацията от пресичането се ползва за изчисляване на осветлението; но само тя не е достатъчна
  - Как точно се получава осветлението зависи и от материалите

# Геометрия vs Материали

- Само геометрия
- Само материали
- Заедно







# Материали

- Материалите описват как повърхностите от геометрията отразяват светлината, т.е., как изглеждат самите повърхности
- Понятието „шейдър“ (shader) има подобен смисъл: шейдърът е математическа функция, която, по зададени позиция, нормали и светлини, изчислява цвета на обекта
  - Т.е., в зависимост от тази функция, обектите изглеждат по различен начин

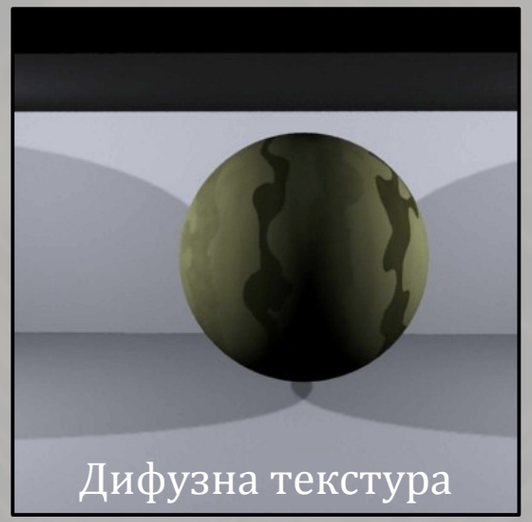
# Материали

- Примери за шейдъри:
  - Шейдър за стъкло / огледален шейдър
  - Шейдър, имитиращ шлифован метал
  - Шейдър, имитиращ лакирано дърво
  - Шейдър, имитиращ авто-боя
  - ...
- Материалите не винаги се описват само с шейдъри
  - В повечето 3D пакети, материалите включват описание и на вътрешността на обекта, не само на повърхността му

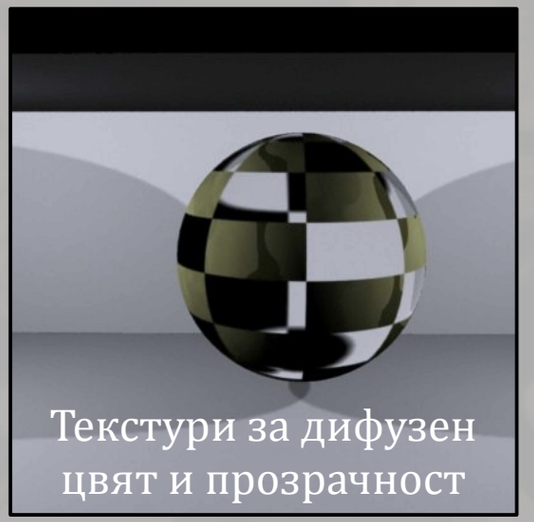
# Текстури

- Свойствата на материала (прозрачност, дифузен цвят, ...) не винаги са равномерни по целия геометричен обект. Често се ползват текстури, който „облепят“ геометрията

Плътен цвят



Дифузна текстура



Текстури за дифузен цвят и прозрачност

# Видове текстури

- Зададени чрез файл
- Процедурни
  - Т.е. чрез зададен алгоритъм за получаване на цвят по (x, y) координати
- 3D текстури
- Bump map текстури
- ...и други

# Материали vs Геометрия

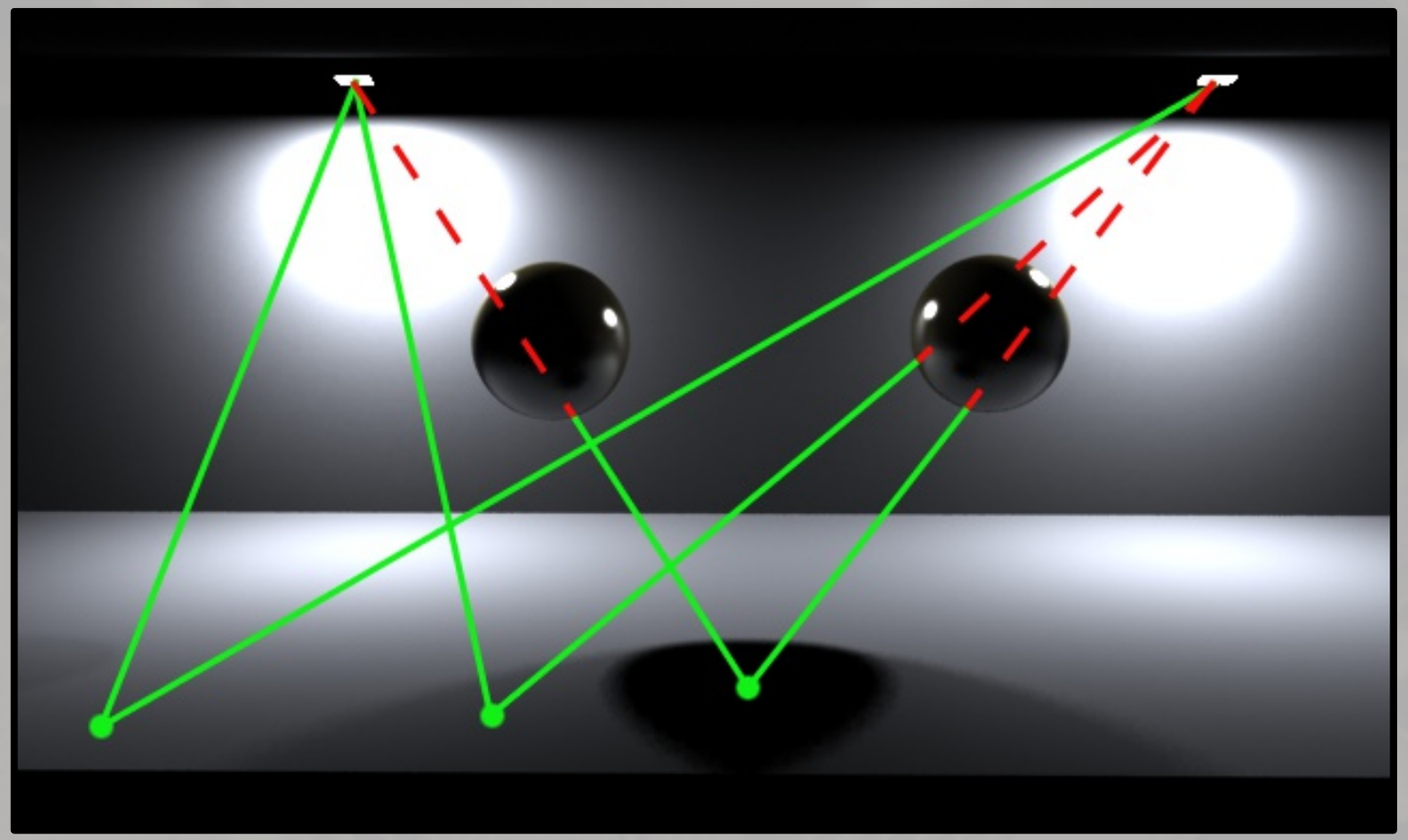
- Материалите и геометрията са два разделени свята
  - Можем да имаме два различни обекта с еднаква геометрия и различни материали
- Node: това е комбинация от геометрия и материал
  - За да се пести памет, обикновено имаме списък със материали и геометрии. В Node обекта се пазят указатели към един материал и една геометрия

# Светлини

- Във всяка сцена има поне една светлина
- Светлините са като лампите в реалния свят – определят колко ярки са отделните части от сцената
- Осветлението в дадена точка обикновено се получава като сума от светлините, видими от дадената точка
  - Но това си зависи от шейдъра

# Осветление

- Пример



# Осветлението

- От какво зависи осветлението, идващо от единична лампа?
  - Сила на лампата
  - Разстояние до нея
  - Ъгъл, под който пада лъча от лампата
  - Други (shader-specific) фактори

# Видове лампи

- Точкова (omni light)
- Насочена (spotlight)
- Сферична, правоъгълна и други обемни лампи
  - Даже и с произволна форма
- С постоянна посока (directional light)
  - Например, слънцето
  - Има специални Sun-Sky системи за реалистично осветление в открити пространства

# Видове лампи (2)

- Със зададено разпределение
  - Подобна на spot light, но интензитета на светлината около централното направление се задава с конкретна функция
  - Полезно за имитиране на истински лампи

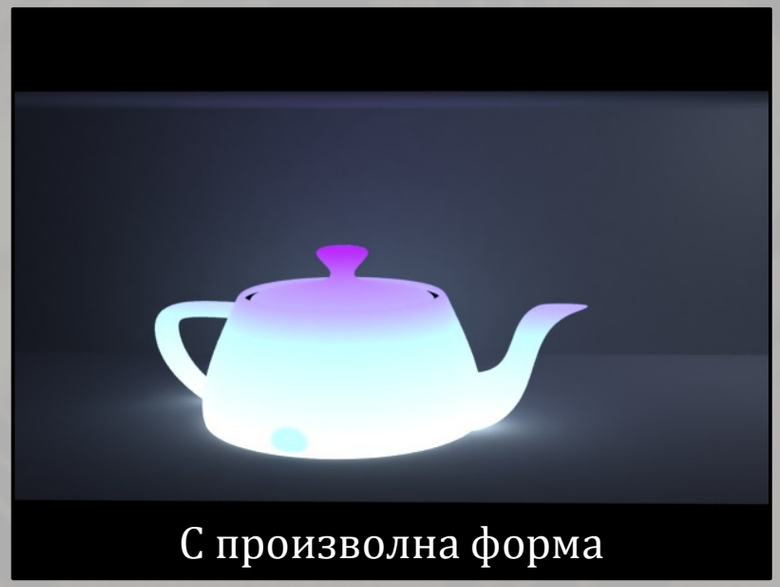
# Видове лампи (примери)

Точкова

Правоъгълна (area)

Насочена (spot)

- С произволна форма:



С произволна форма

# Обкръжение (environment)

- Обкръжението се грижи за частта от изображението, която не принадлежи на никой обект
  - При затворени сцени, нямаме нужда от обкръжение
- Обкръжението обикновено не се влияе от позицията на камерата, а само от посоката ѝ
  - Пример [humus-metaballs]

# Видове обкръжение

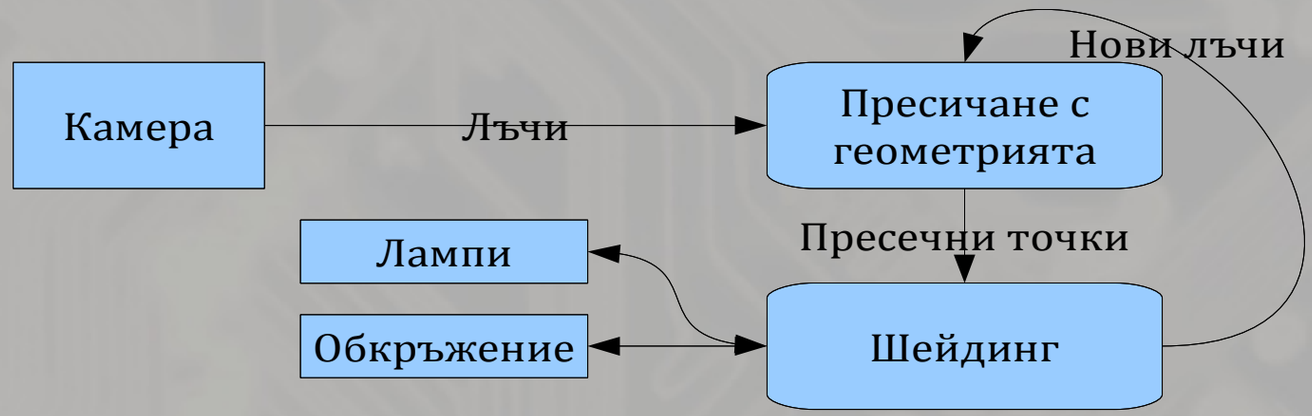
- Солиден цвят
  - При черен цвят, все едно нямаме обкръжение
- Градиент
- Environment map – използват се текстури
  - Spherical map – използва се една текстура, облепена сферично
  - Cube map – обкръжението е с формата на куб, ползват се 6 текстури за всяка стена на куба
- Sun/Sky environment – симулира свойствата на небето

# Влияние на обкръжението

- Обкръжението обикновено влияе на изчислението на осветлението
  - Например, сцена със сиво обкръжение ще е по-ярка от такава с черно
- При Image-based lighting, цялата светлина идва от обкръжението – нямаме лампи

# Базова инфраструктура на рейтрейсъра

- Камерата генерира първоначалните лъчи за определяне на цветовете на пикселите в картината
- Шейдърите могат да поискат да трасират нови лъчи (например, за изчисляване на сенки или отражения)



# Показване на графика на екрана

- За целите на нашия курс, ще ползваме библиотеката **SDL** (Simple Directmedia Layer)
- Компактна C библиотека с широка поддръжка
  - Windows, Linux, Mac OS X, FreeBSD
- Не изисква специална инсталация, достатъчно е просто да влачим един DLL с програмата ни
- Повечето неща стават лесно, без да четем тонове документация
- http://libsdl.org

# SDL

- Интеграцията на Code::Blocks с SDL е тривиална – под Windows трябва само да укажете къде сте разархивирали `SDL-devel-...-mingw32` архива, свален от libsdl.org
- Демонстрация – проста програма, показваща графика чрез SDL
  - Сорс-кода на програмата може да свалите от сайта на курса

# SDL екранен буфер