3D графика и трасиране на лъчи v.6.0

http://raytracing-bg.net/

Тема 5

Lambert и Phong shading
Текстури
Пресичане с по-сложни примитиви
Булеви операции

Съдържание

- Lambert и Phong shading
- Текстури
- Пресичане с куб
- Булеви операции
  - Обединение
  - Сечение
  - Разлика
- Bitmap текстури
- UV координати

Новини

- Домашни
  - ДР4 – публикувани днес (вижте си и Moodle)
  - ДР5 – очакваме да публикуваме до четвъртък

Материали

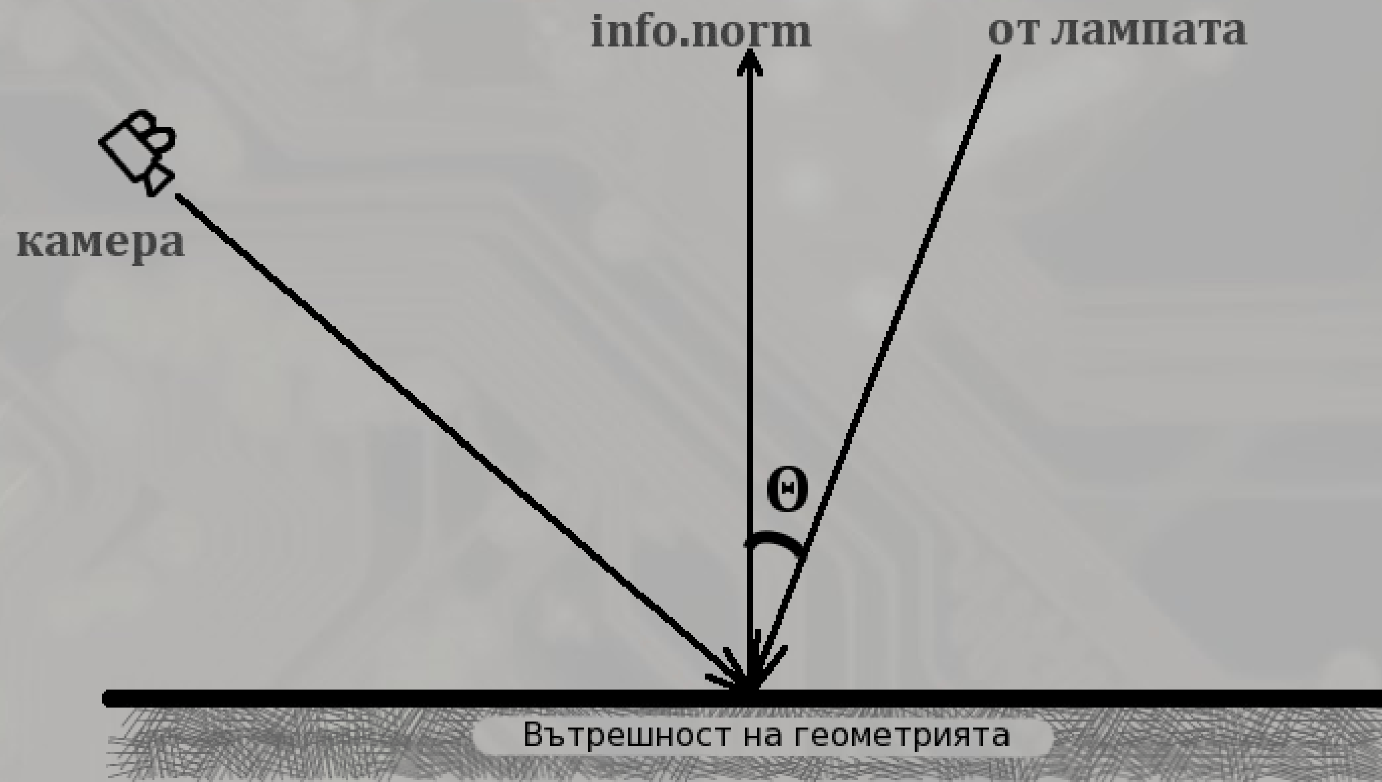
- Ще изнесем генерирането на дифузен цвят в отделен клас – Texture.
- Ще напишем два модела за повърхностните качества
  - Ламберт (или дифузен) модел
    - Написахме го (без много обяснения) още предната лекция
    - Симулира матови повърхности, които не хвърлят отблясъци
  - Фонг (Phong) модел
    - За „лъскави“ повърхности

Ламберт vs Фонг

Ламберт

- Осветлението зависи единствено от ъгъла, под който пада светлината спрямо повърхността
  - Нормалният вектор (info.norm) е перпендикуляр на геометрията в пресечната точка
    - т.е., ако имаме допирателна равнина в тази точка, това е нейният нормален вектор
- Яркост = cos(Θ)
  - Може да се изведе геометрично

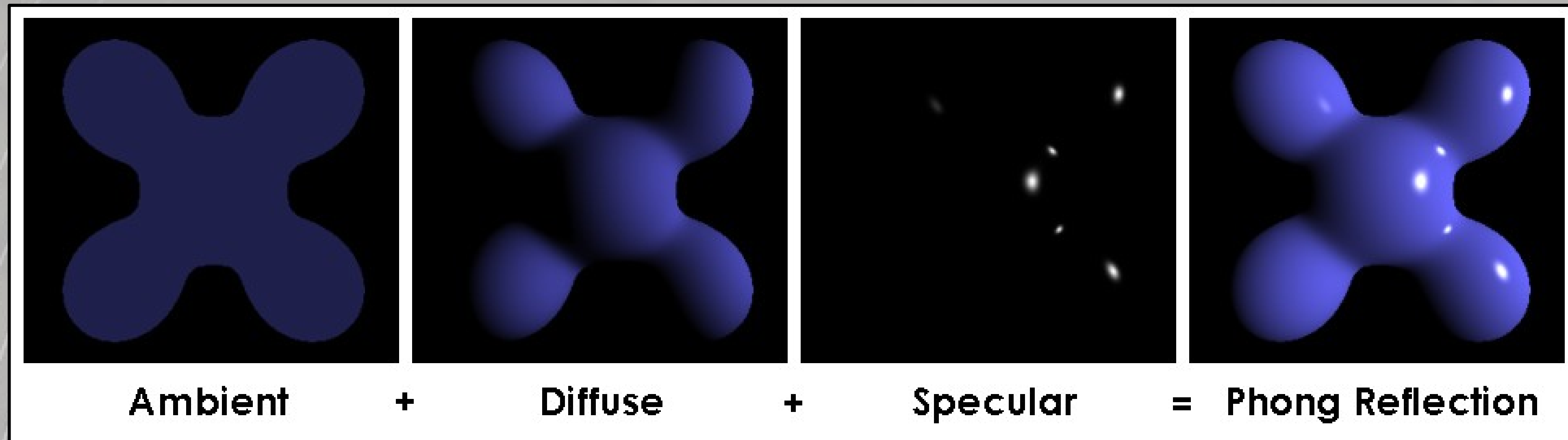
Ламберт



- OutColor = materialColor * max(0, cos(Θ)) * lightColor
- cos(Θ) = dot(lightVec, info.norm)

Фонг

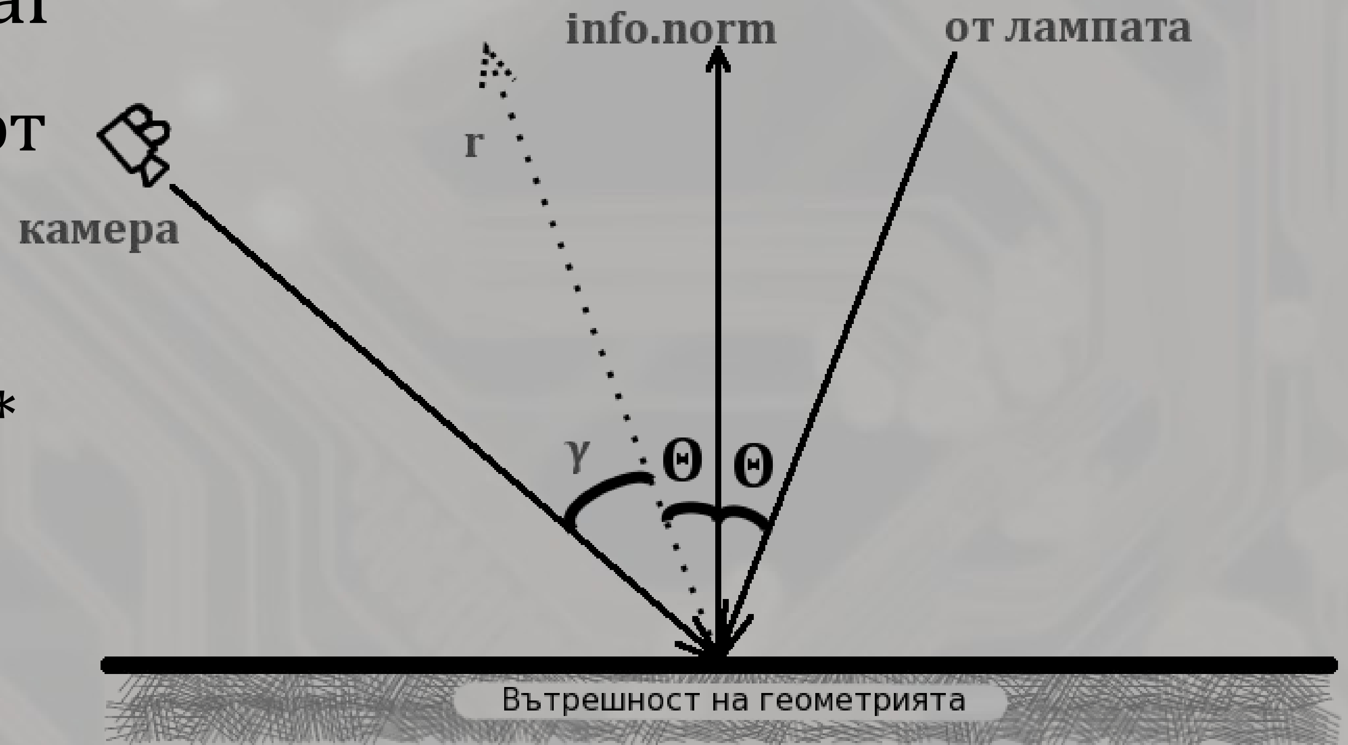
- Фонг е същото като Ламберт, само че добавяме и „лъскав“ (specular) компонент.



- Specular компонента се определя от това дали отражението на идващата светлина е близко до посоката на камерата

Фонг

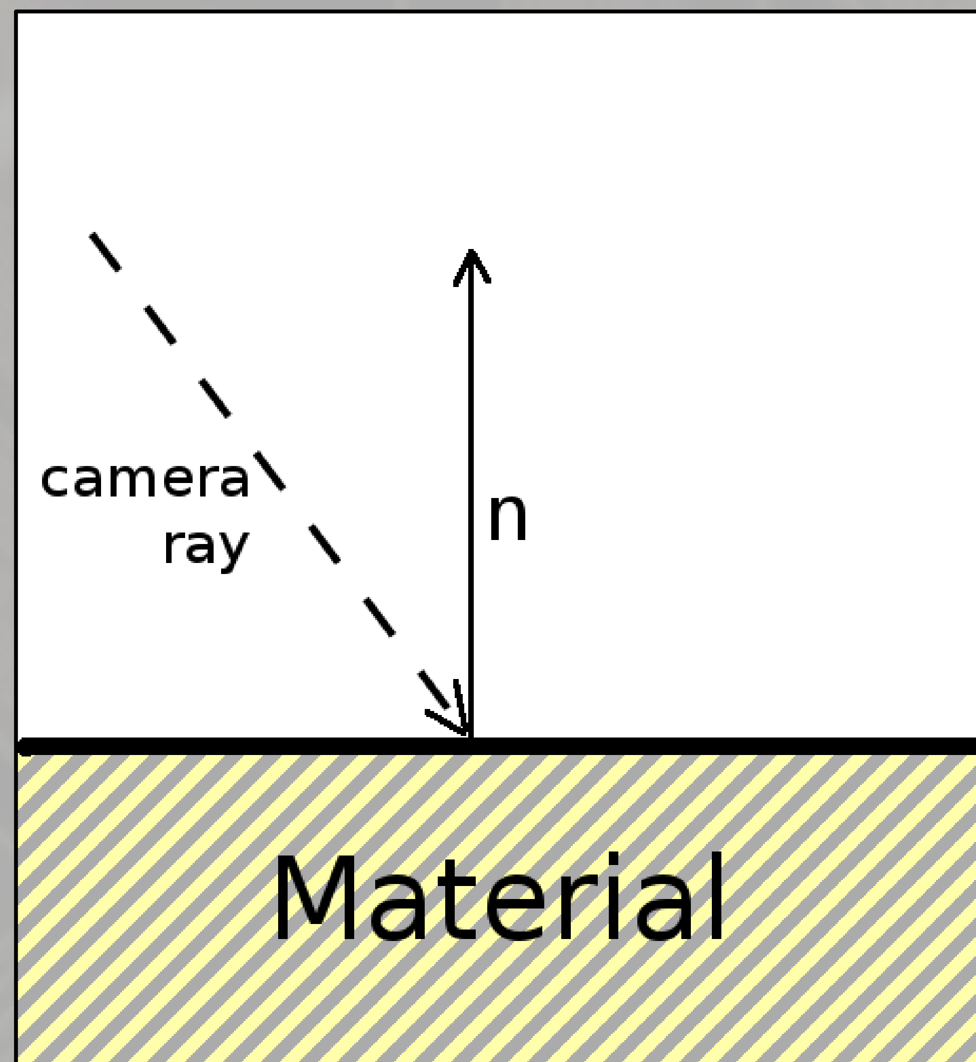
- Фонг = Ламберт + specular
  - r е отражението на лъча от лампата, спрямо нормалата info.norm
  - specular = max(0, cos(γ))$^{\eta}$ * lightColor
  - Параметърът η (обикновено наричан „exponent“) контролира „колко лъскав е материала“



r = reflect(lightVec, info.norm)
cos(γ) = dot(r, -cameraRay.dir)

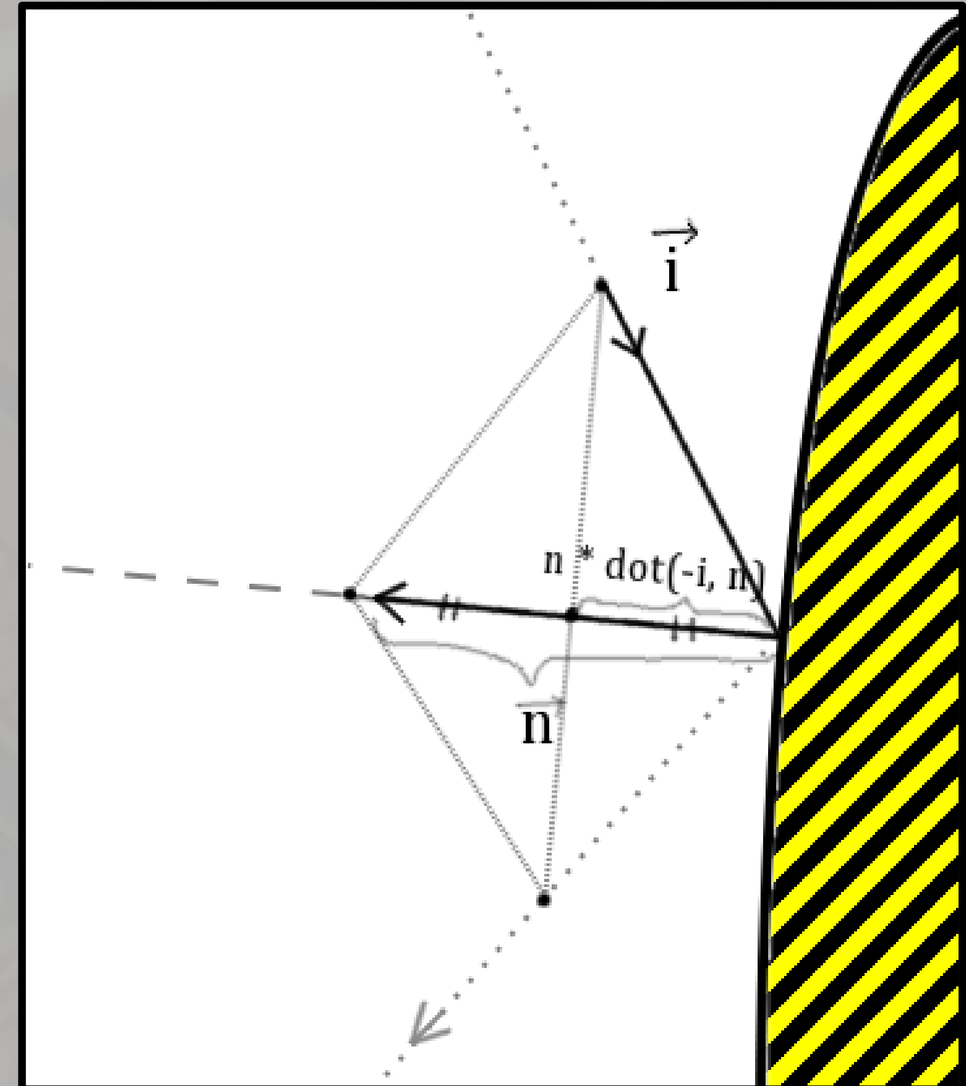
Ориентация на повърхнините

- Нормалите определят кое е „отвън“ и „отвътре“ на една геометрия
  - Отвън: dot(ray.dir, n) < 0
  - Отвътре: dot(ray.dir, n) > 0
- faceforward(): обръща нормал, така че да сочи „към нас“
  - dot(ray, faceforward(ray, n)) < 0
  - Ще го ползваме в най-различни шейдъри



Reflect

```
// i е входящия вектор (от камерата)
// n е нормалата
inline Vector reflect(const Vector& i,
                      const Vector &n)
{
    return i + 2 * dot(-i, n) * n;
}
```



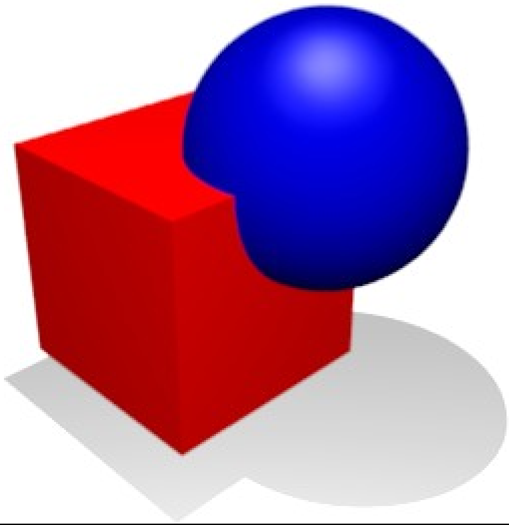
Пресичане с куб

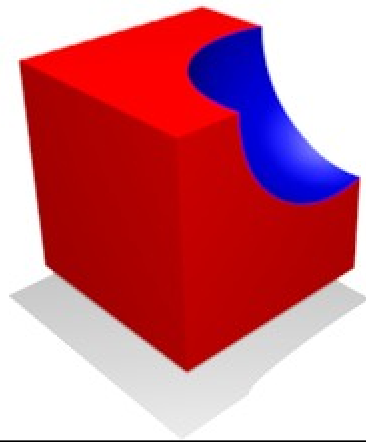
- За всяка страна приемаме, че пресичаме с равнина...
  - ... и проверяваме дали пресечната точка е в граници по другите две измерения
  - Подобно на пресичането със сферата, трябва да внимаваме за стени, намиращи се зад камерата
    - т.е. търсим най-близката положителна пресечна точка.
- Нормалите зависят единствено от стената
  - Например, за стената +X, нормалният вектор е (1, 0, 0)
- UV координатите могат да се генерират по много начини

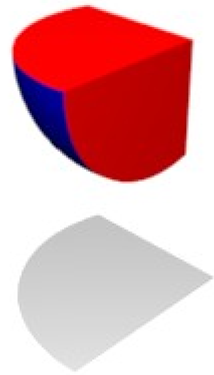
Пресичане с куб

- UV:
  - (u, v) := (x + y, z) – съвпада по ръбовете, но разтегля текстурата по две от стените
  - (u, v) := (p.x – cube.x, p.y – cube.y) / (2 * cube.halfSide) (за стена +/-Z)
    - Разкъсва текстурата по ръбовете, но изглежда добре
  - Сферични координати:
    - (u, v) := (atan2(p.z – cube.z, p.x – cube.x), asin(2*(p.y – cube.y)/cube.halfSide)
    - Изглежда доста странно

Булеви операции

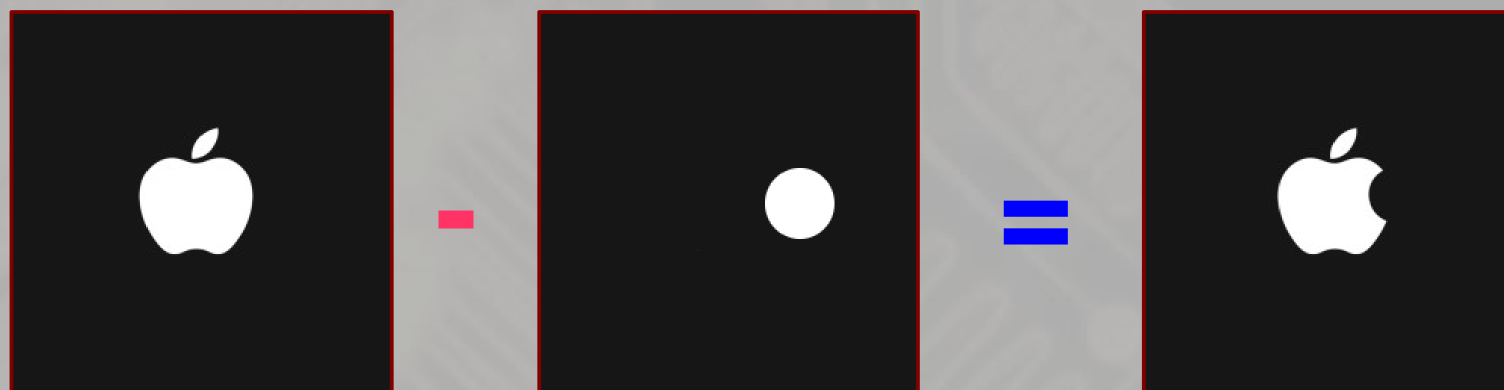






Булеви операции

- Често е полезно да създадем нов 3D обект, като комбинираме два съществуващи, и се интересуваме от някоя част на двата обекта, която да изпълнява някакво логическо (булево) условие
- На английски: CSG = Constructive Solid Geometry

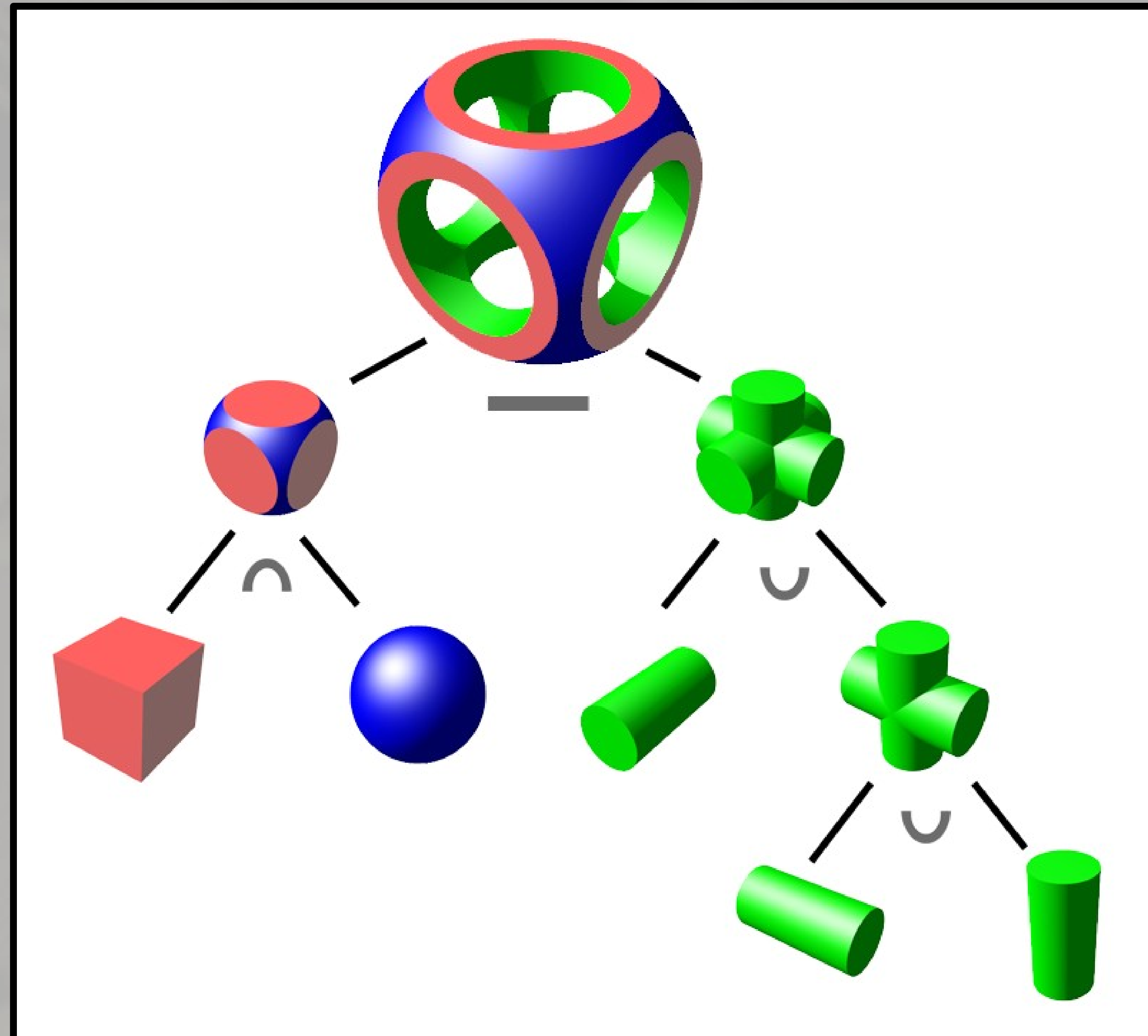


Apple's logo

Булева операция между два обекта

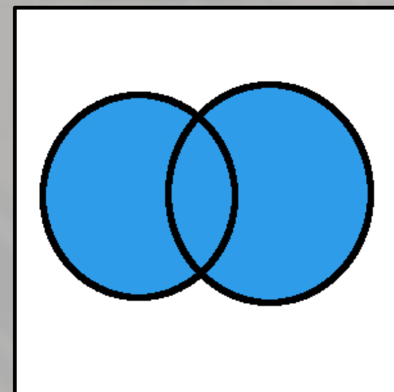
- Алгоритъмът за булеви операции е специфичен за Raytracing-a. При Z-buffer, например, аналогичен алгоритъм няма; булевите операции там се реализират много по-трудно
- Булевата операция между два обекта е фундаментална; при повече от един обект, може да реализираме дървовидна структура (двоично дърво), при което резултатът от някаква булева операция да се ползва за вход на друга булева операция
  - Ще дадем пример

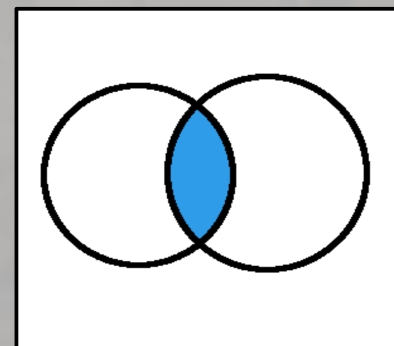
CSG trees

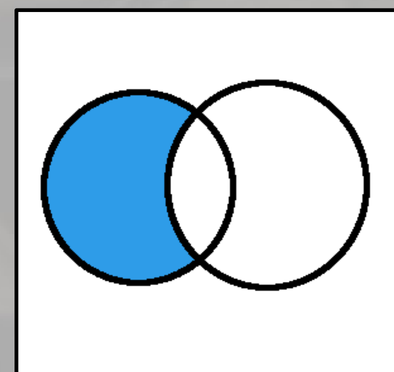


Видове булеви операции

- Обединение
- Сечение
- Разлика
  - Несиметрична операция!
- ...и всъщност, произволна булева функция на два аргумента – алгоритъмът е един и същ, променя се само самата булева функция
  - Булевата функция е израз от вида вътре_сме_в(A) && не_сме_вътре_в(B)







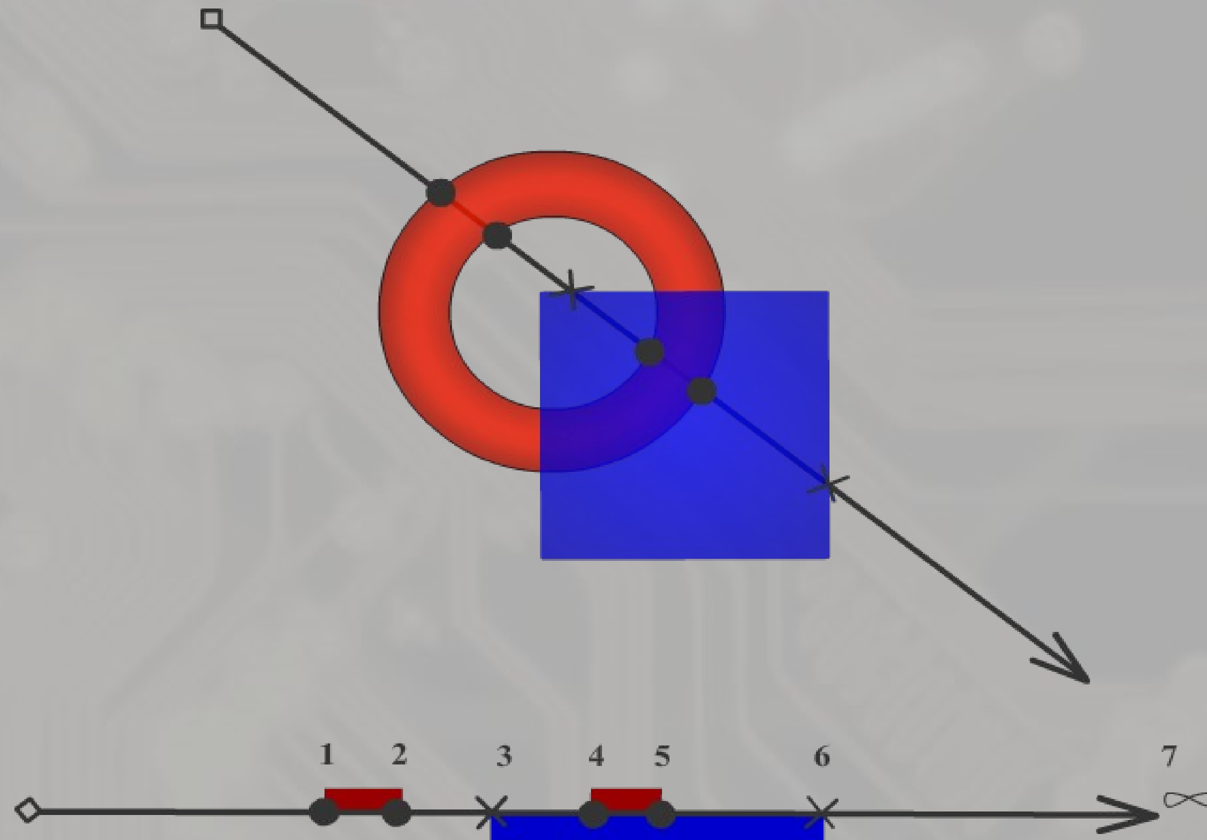
Алгоритъм за CSG с две геометрии

- Намираме всички пресечни точки на лъча с всяка от двете геометрии
  - Това включва всички пресечни точки на геометрия по лъча (например, за сфера, те са 1 или 2; за по-сложни примитиви, може да са повече)
- Обединяваме двата списъка с пресечни точки. Обединението сортираме по разстояние до пресечната точка
- Вървим по сортирания списък, като за всяка точка
  - Изчисляваме дали сме вътре във всяка от двете геометрии...

Алгоритъм за CSG с две геометрии

- … и ако изпълняваме условието на булевата функция, значи сме намерили пресичане – обявяваме пресечната точка за пресечна на лъча с булевия обект
- Проверката дали сме вътре в някоя геометрия е лесна
  - Може да определим дали началото на лъч е вътре или извън дадена геометрия, като преброим пресечните точки на лъча с геометрията
    - Нечетен, ако сме вътре; четен, ако сме извън нея
- При подминаването на пресечна точка, сменяме флага, показващ дали сме „вътре в геометрията“ (за тази геометрия, с която е пресечната точка)

Алгоритъм за CSG



- Операцията е сечение. Алгоритъмът ще намери точка 4 (това е първата точка, в която условието „вътре_сме_в_(A) && вътре_сме_в_(B)“ е изпълнено)

Приложение на CSG

- Често използвано в CAD системите за механика
  - Част трябва да пасне в друга, като болт и гайка
  - Металообработването често се прави с отнемане на материал (*CSGDiff* in the real world™)

Нормали

- Принципно, всяка точка от една повърхнина има нормален вектор; най-общо, той сочи „навън“ от „тялото“ на обекта
  - Пример за сфера: N := normalize(intersection.pos – sphere.pos)
- Обикновено, нормалите са ориентирани така, че лъчът R от пресечната точка към камерата да е в същото полупространство като N, т.е. N*R (= dot(N, R)) > 0
- Backface culling: това е оптимизация, при която пропускаме (части от) обекти, за които N*R < 0

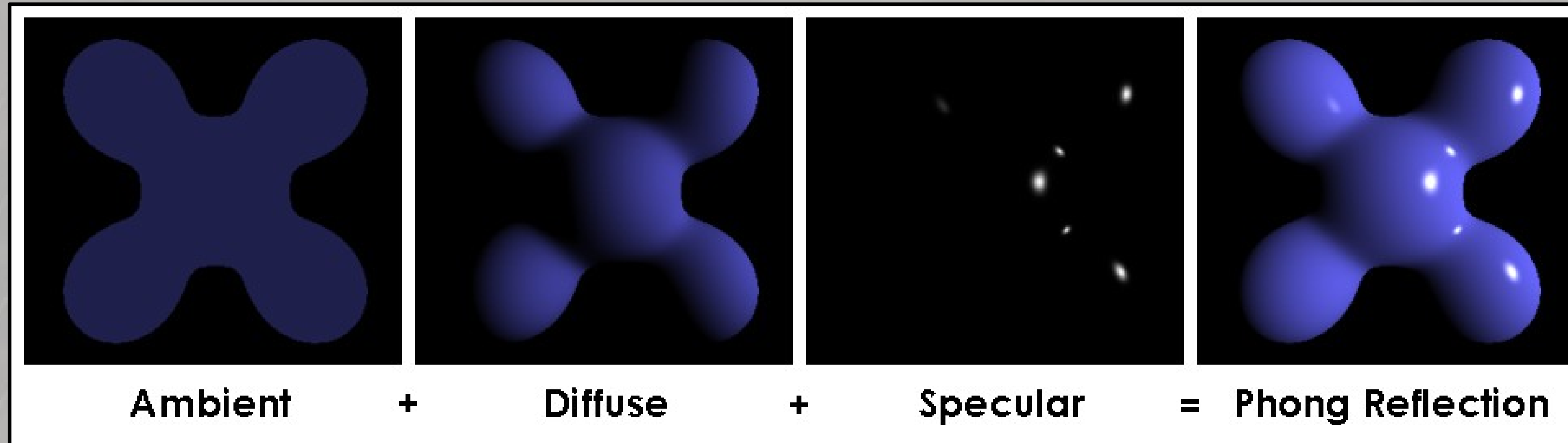
Нормали

- Когато лъчът тръгва от вътрешността на даден обект, обаче, нещата са по-различни
  - Например, ако тръгваме от средата на куб, backface culling-ът ще игнорира всички страни на куба
  - Geometric normal (gnormal) – истинският нормален вектор на повърхнината
  - Camera normal (normal) – нормален вектор, който е ориентиран в посока на камерата (rayFromCam * $N_c \le 0$)
    - Т.е., или normal = gnormal, или normal = -gnormal
    - Реализирано в нашия код от faceforward() функцията

Ambient light

- Светлината от обкръжението (ambient light) изразява факта, че една (малка) част от светлината не идва директно от лампите, а чрез отражения от други (неизлъчващи) предмети.
  - В природата, наситени сенки (Color = (0, 0, 0)) са рядкост
  - Досега ние смятахме само директната светлина; ambient light цели да симулира индиректната светлина
  - Ambient е просто някакъв константен цвят, който се добавя към цялото осветление
  - Груб „hack“, целящ да свърши работата на Global Illumination алгоритмите

Ambient light



- Ambient участва във Phong модела, като компонент, който не се влияе от косинуса между светлината и нормалния вектор или от наличието на сянка; просто константен цвят, умножен по текстурата или базовия цвят на обекта.

Ambient ≠ Global Illumination

- Ambient е еднакъв за всички точки от сцената
- При Global Illumination алгоритмите, именно това е трудният момент – да се сметне колко трябва да е ambient цвета, в зависимост от това къде по сцената се намираме
  - Например, в стая с една зелена и една червена стени, близо до зелената стена ще имаме по-зеленикав ambient. Този ефект се нарича Color bleeding.
  - Ambient **не** може да симулира този ефект!

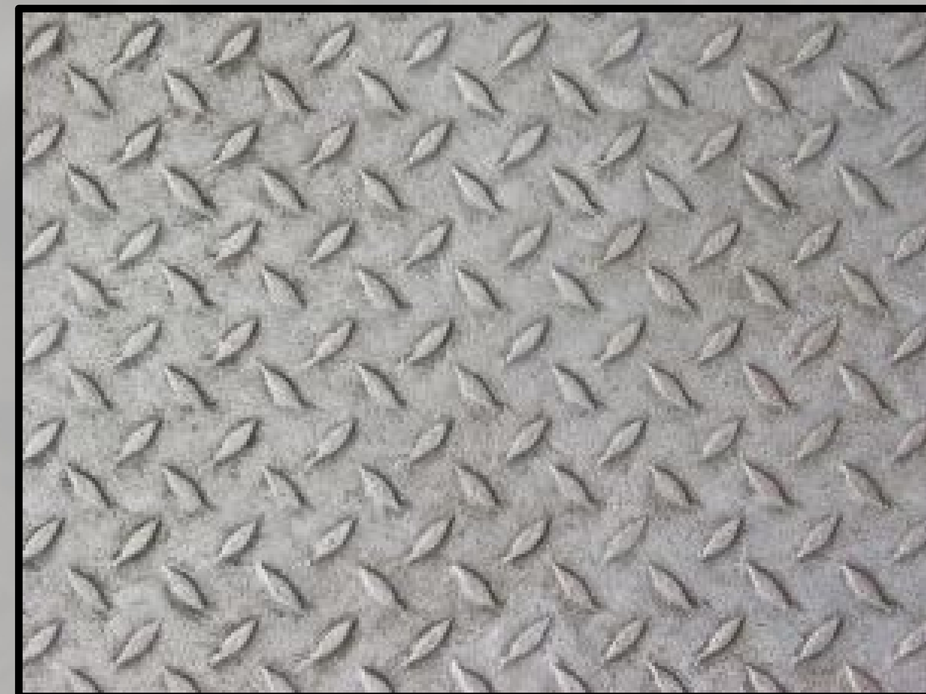
Color bleeding

Bitmap текстури

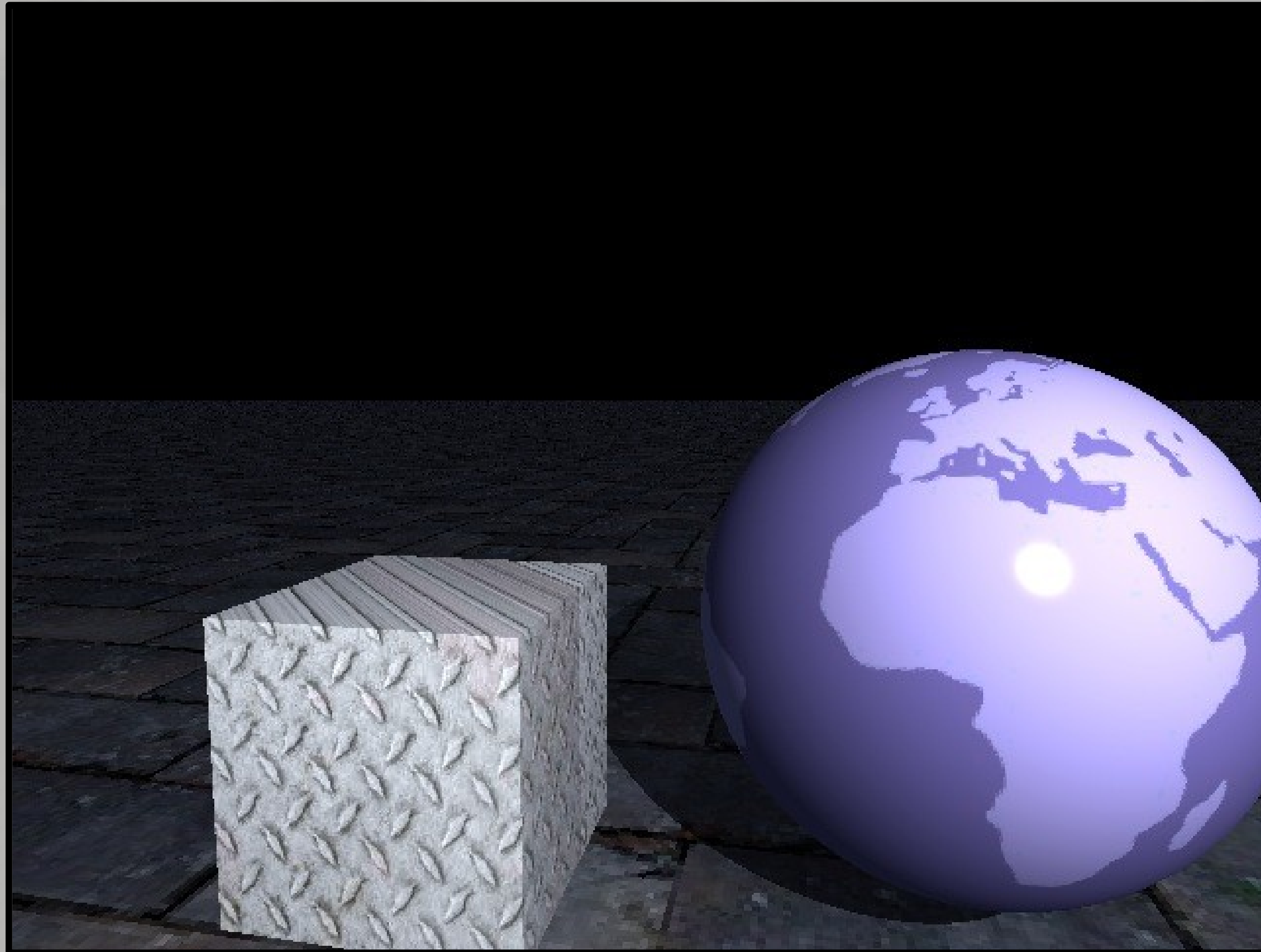
- При bitmap текстурите, цветът не се генерира процедурно, ами се взема от картинка
- Обикновено се абстрахираме от размера на картинката и приемаме, че нейните координати са в квадрата (0,0)-(1,1), като оставяме скалирането скрито в getColor() на текстурата
- Геометрията трябва да се грижи да създава удобни u,v координати, които да могат правилно да се облепят с текстури.

Bitmap текстури

- Като цяло, текстурите се ползват за да създадат усещане за допълнителен детайл, какъвто обектът няма геометрически
- Тук ще говорим за „дифузни“ текстури – такива, които променят цвета на обекта
    - По-нататък ще разгледаме и други видове текстури



Bitmap текстури



UV координати - пример

- В примера със сферата и картата на света, нямаме повторение (tiling) на текстурата, така че uv координатите трябва да са в [0, 1]
- *u* пробягва по ширината на картинката, т.е. *u* отговаря на географска дължина
  - $u = (\pi + \text{atan2}(\text{info.p.z} - \text{sphere.z}, \text{info.p.x} - \text{sphere.x}))/(2\pi)$
- *v* пробягва по дължината на картинката, т.е. *v* отговаря на географска ширина
  - $v = 1.0 - (\pi/2 + \text{asin}((\text{info.p.y} - \text{sphere.y}) / R))/\pi$