

3D графика и трасиране на лъчи v.2.0



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



Тема 4

Въведение в камерата Пресичане с равнини и кълбета

Съдържание

- Реализация на правоъгълна камера
- Реализация на raytracing алгоритъма
- Реализация на пресичания с равнина и сфера
- Шейдъри и текстури
- Сенки
- Домашни :)

Разходка из кода на проекта

- `svn://anrieff.no-ip.org/retrace`
- Изчакване в SDL (SDL events)
- Код, който ще ползваме наготово: Color, Vector, Matrix

Raytracing cheatsheet

- class Color:

- Създаване на цвят: `Color c(0.9, 0.6, 0.9)`
 - Web-формат: `Color c(0xe699e6)`
- Събиране на два цвята: `Color a, b, c; a = b+c;`
- Скалиране на цвят: `Color a; a = a*0.3; a /= 2;`
 - и т.н.
- Нулиране (`::makeZero()`), интензитет (`::intensity()`)

- Пример (осредняване на N измервания):

```
Color sum(0, 0, 0);  
for (int i = 0; i < N; i++) sum += getResult(i);  
sum /= N;
```

Raytracing cheatsheet

- class Vector:
 - Точка в 3D или посока според контекста
 - Посоките обикновено са нормирани, т.е. `::length() == 1`
 - Създаване
 - Като координати: `Vector v(0.3, -2, 0.8)`
 - Като посока от т. А към т. В (също вектори): `Vector dir = B - A;`
 - Скалиране: `v *= 1.5;` `x = a*0.5 - b*1.3;`
 - Дължина (`length()`, `lengthSqr()`), нормиране (`normalize()`)

Raytracing cheatsheet

- class Vector:

- Скалярно произведение:

```
double result = a*b;
```

```
double result = dot(a, b);
```

- Ако a и b са нормирани, то $\text{dot}(a, b)$ е косинуса от ъгъла между тях

- Векторно произведение:

```
Vector result = a^b;
```

Нетривиален пример

- При дадена посока v (нормирана), намира ортонормиран базис $\{a, b\}$, перпендикулярен на v :

```
Vector a, b, t;  
do {  
    t = Vector(randomFloat(), randomFloat(), randomFloat());  
    if (t.length() > 1e-4) t.normalize();  
} while (t.length() < 1e-4 || fabs(t * v) > 0.9);  
a = t^v;  
a.normalize();  
b = a^v;
```


Raytracing cheatsheet

- Class Matrix

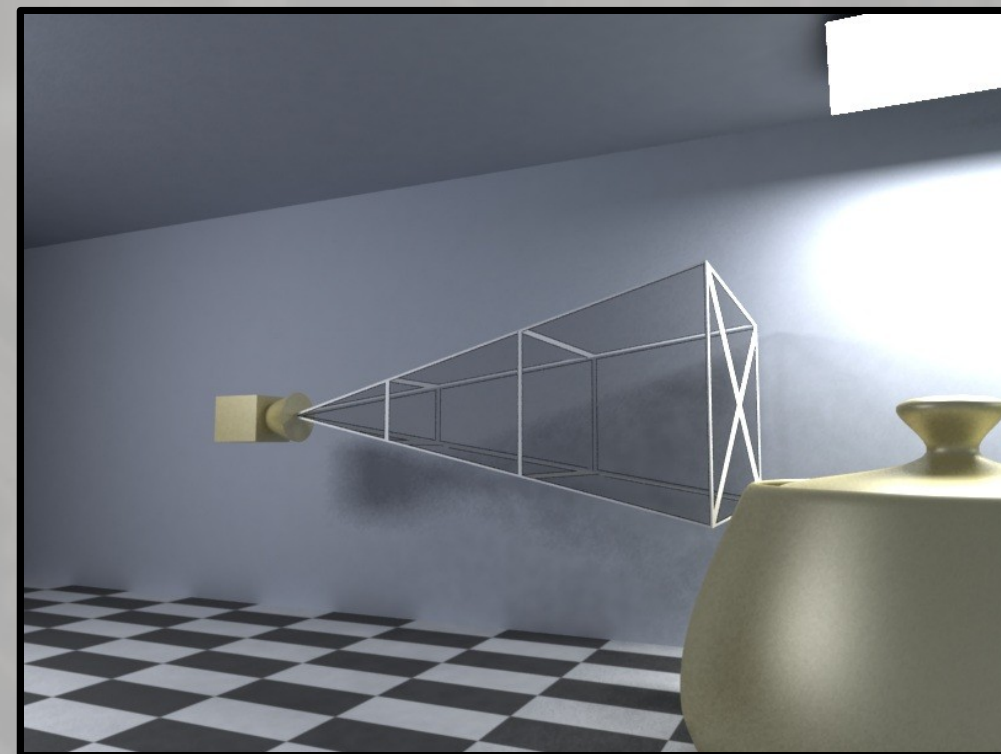
- 3x3 матрица с обичайните действия
 - Умножение, `determinant()`, `inverseMatrix()`
- Умножението на `Vector` с матрица ще е отдясно ($v = v * m$)
- Генериране на ротационни матрици:
 - Функциите `rotationAround«ос»(«ъгъл»)`, където «ос» е X, Y или Z, а «ъгъл» е в радиани
- Например, да завъртим точката p спрямо $(0, 0, 0)$, първо с 30° около оста X и после с 90° около оста Z:
$$p *= \text{rotationAroundX}(PI/6) * \text{rotationAroundZ}(PI/2)$$

Raytracing cheatsheet

- class Ray:
 - Начало и посока; посоката винаги е нормирана

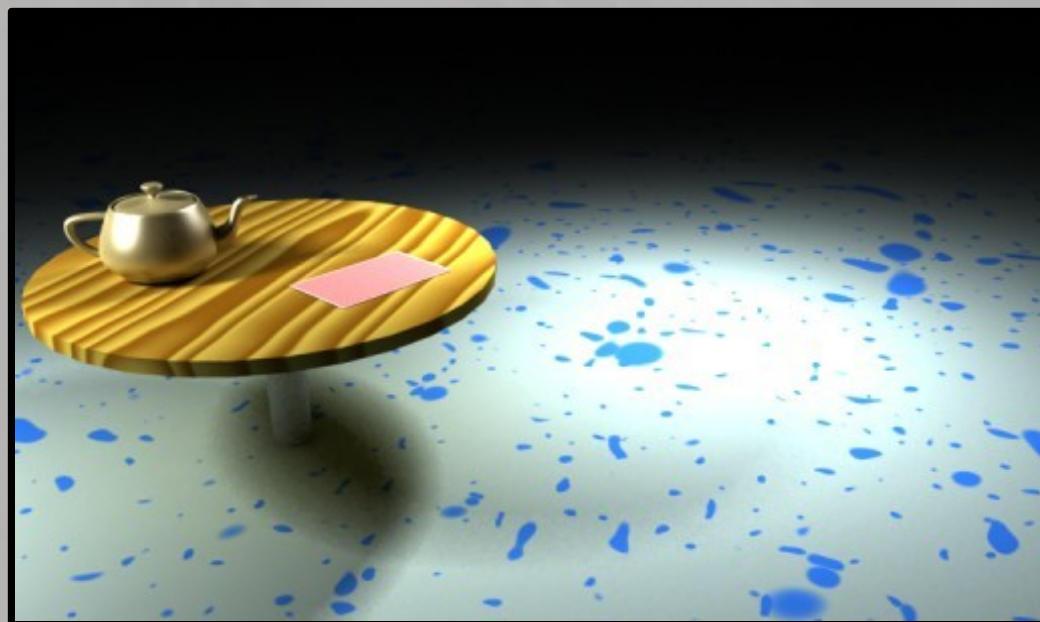
Реализация на камерата

- Ще реализираме стандартна, правоъгълна (rectilinear) камера
- Зрителното поле на камерата е във формата на правоъгълна пирамида
- Нарича се още pinhole camera
 - Няма фокус, DOF-ът е безкраен



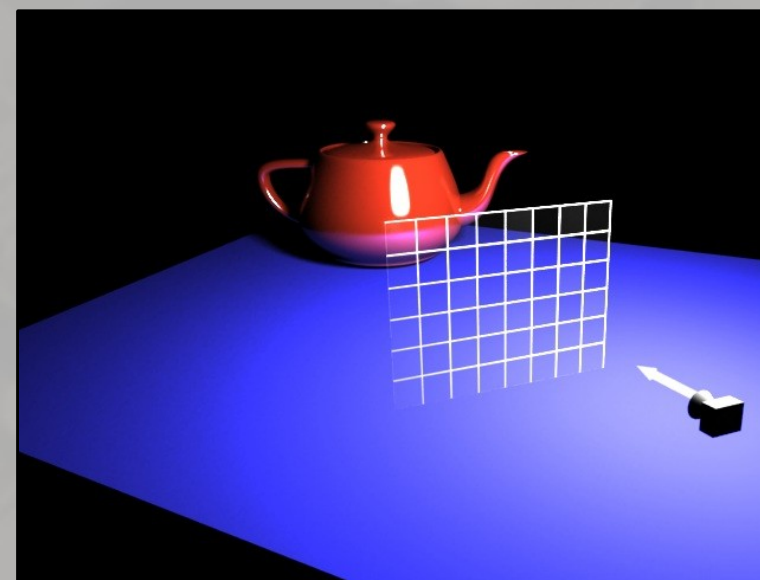
Реализация на камерата

- Ако разположим лист милиметрова хартия пред камерата, при подходяща разделителна способност, всяко квадратче ще отговаря на един пиксел
- Демонстрация [millimetre.m4v]



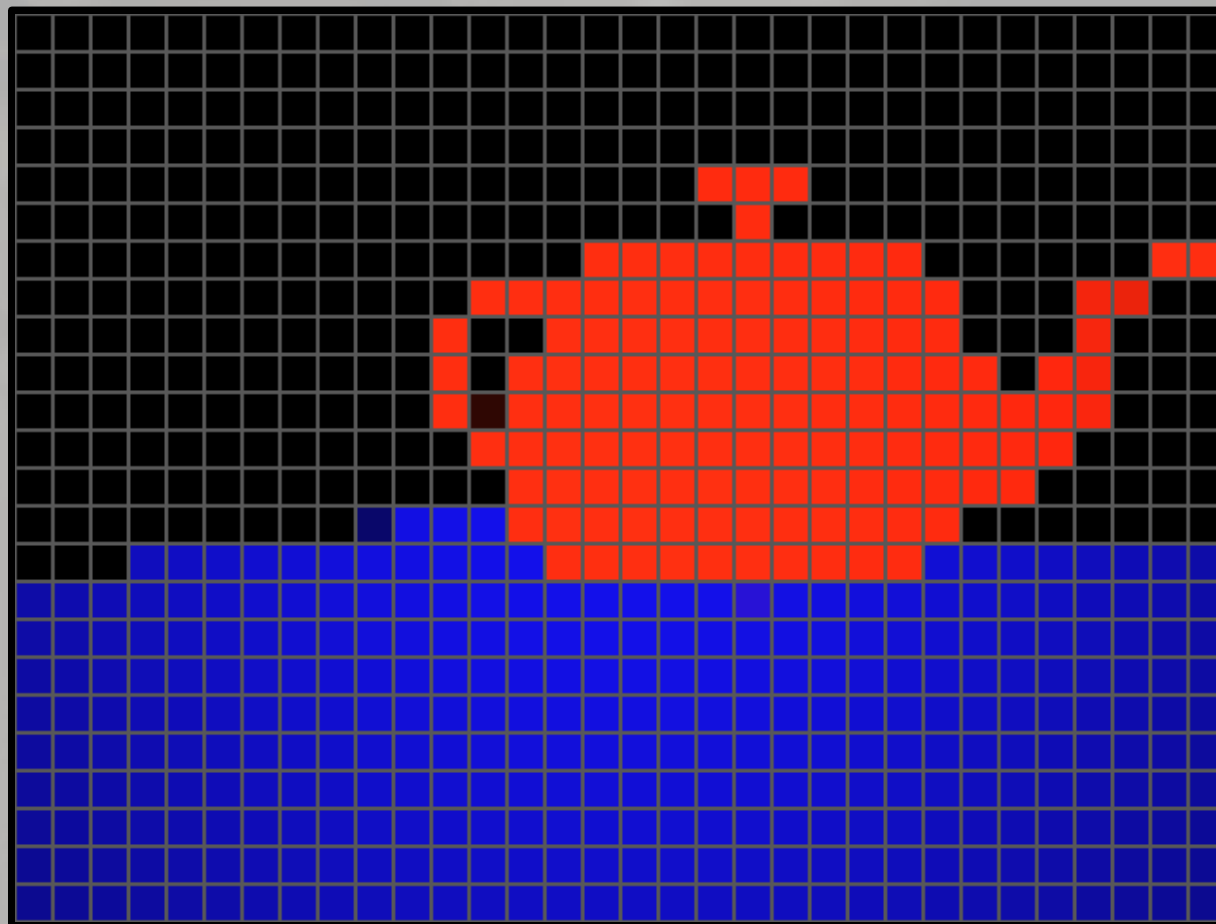
Реализация на камерата

- Най-простият възможен рейтрейсър: изстрелваме лъч през всеки пиксел и оцветяваме пиксела в зависимост от ударения обект
 - Например: черно, ако не ударим нищо; червено за чайник; синьо за равнина
- Демонстрация [primitive.m4v]



Реализация на камерата

- Резултатната картинка:



Реализация на камерата

- Ще симулираме листа „милиметрова хартия“
- Трябват ни просто три от краищата на листа (горен ляв, горен десен, долен ляв)
- Намиране на произволен пиксел (по зададени X, Y)
 - За X направлението, интерполираме между горен ляв и горен десен край
 - За Y направлението, интерполираме между горен ляв и долен ляв край
 - $$\text{DestP} = \text{TopLeft} + (\text{TopRight} - \text{TopLeft}) * (X / \text{screenW}) + (\text{BottomLeft} - \text{TopLeft}) * (Y / \text{screenH})$$

Реализация на камерата

- Как генерираме трите краища?
 - Ще предпологаме, че камерата се намира в $(0,0,0)$ и гледа в посока на оста Z
 - Ще разположим лист със зададеното отношение (aspect ratio), на разстояние 1 от камерата ($Z = 1$)
 - Ще мащабираме листа, така че диагоналния ъгъл да стане колкото искаме (FOV)
 - За наше удобство, ще задаваме FOV-а в градуси
 - Ще приложим трите ротации (roll, pitch и yaw) върху краищата

Генериране на лъчи

- Генерирането на лъчи след това става по споменатата формула, като трябва да транслираме лъча да тръгва от позицията на камерата
 - Самия лъч задаваме като двойка (начало, посока), като посоката е единичен вектор.

Рендерирането на сцената

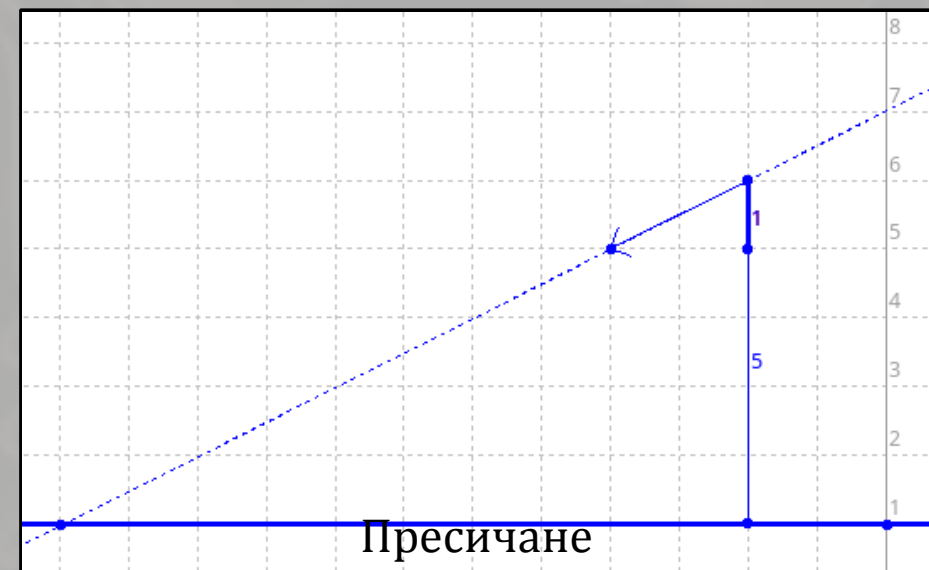
- **for each** *y* **in** rows:
 for each *x* **in** columns:
 ray = camera.getRay(*x*, *y*)
 vfb[*x*, *y*] = raytrace(*ray*)
- raytrace() е функция, която, по даден лъч, намира цвета на първия ударен обект по протежение на лъча

Raytracing алгоритъма

- **Function** raytrace(ray) :
closestDist = $+\infty$
closestNode = None
for each node **in** sceneNodes:
 intersectionInfo = intersect(ray, node)
 if intersectionInfo.distance < closestDist:
 closestDist = intersectionInfo.distance
 closestNode = intersectionInfo.node
if closestDist == $+\infty$:
 return backgroundColor
return closestNode.computeColor()

Пресичане с равнина

- Ще реализираме обекта „равнина“. Равнината ще е успоредна на XZ координатната равнина, ще е зададено само разстоянието по y от XZ
- Имаме два случая:



Пресичане със сфера

- Разстоянието от центъра на сферата O до произволна точка T е: $\text{sqrt}((T_x - O_x)^2 + (T_y - O_y)^2 + (T_z - O_z)^2)$
- Позицията на всяка точка по лъча e : $\text{start} + p * \text{dir}$
 - Където $p > 0$ е параметър
 - Разстоянието от всяка точка на лъча до сферата се получава по горната формула, като заместим $T = \text{start} + p * \text{dir}$, за някое конкретно p
 - Търсим такова p , че разстоянието да е точно R (радиуса на сферата)

Пресичане със сфера

- $\text{sqrt}(((\text{start}_x + p \cdot \text{dir}_x) - O_x)^2 + ((\text{start}_y + p \cdot \text{dir}_y) - O_y)^2 + ((\text{start}_z + p \cdot \text{dir}_z) - O_z)^2) = R \Leftrightarrow$
 - (нека $H = \text{start} - O$):
- $\text{sqrt}((H_x + p \cdot \text{dir}_x)^2 + (H_y + p \cdot \text{dir}_y)^2 + (H_z + p \cdot \text{dir}_z)^2) = R \Leftrightarrow$
- $(H_x + p \cdot \text{dir}_x)^2 + (H_y + p \cdot \text{dir}_y)^2 + (H_z + p \cdot \text{dir}_z)^2 = R^2 \Leftrightarrow$
- $p^2(\text{dir}_x^2 + \text{dir}_y^2 + \text{dir}_z^2) + 2 \cdot p(H_x \text{dir}_x + H_y \text{dir}_y + H_z \text{dir}_z) + (H_x^2 + H_y^2 + H_z^2 - R^2) = 0 \Leftrightarrow$
- $p^2 \cdot \text{dir.length}^2 + p \cdot (2 \cdot \text{dir} \cdot H) + (H.length^2 - R^2) = 0$
- Квадратно уравнение!
 - 0 решения – лъчът не пресича сферата
 - 1 или 2 решения – лъчът допира или пресича сферата

Осветление

- Ще използваме най-простия модел за лампа – точкова
- Точковата лампа има две характеристики
 - Позиция
 - Интензитет
- Точковата лампа разпръсква светлина във всички посоки
- Осветеността на дадена точка намалява с квадрата на отдалечеността ѝ от лампата ($1/r^2$)
 - Защо?

Материали

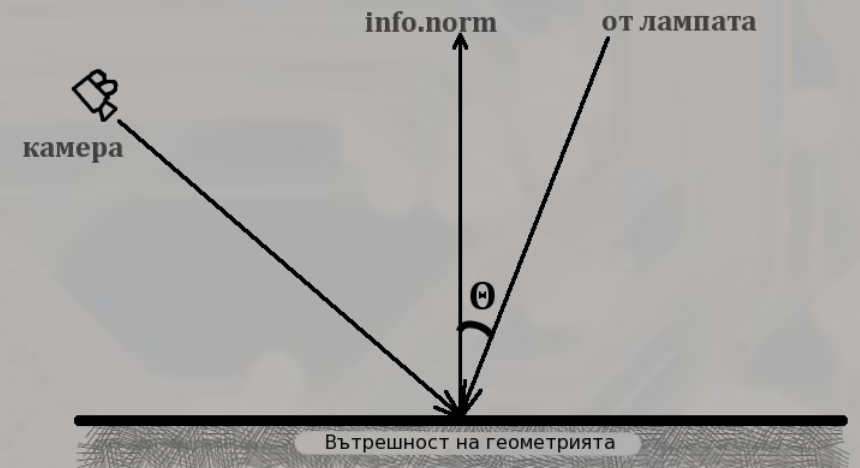
- Ще напишем 2 прости материала
 - Ламберт (или дифузен) модел
 - За матови повърхности, които не хвърлят отблясъци
 - Фонг (Phong) модел
 - За „лъскави“ повърхности

Ламберт vs Фонг



Ламберт

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



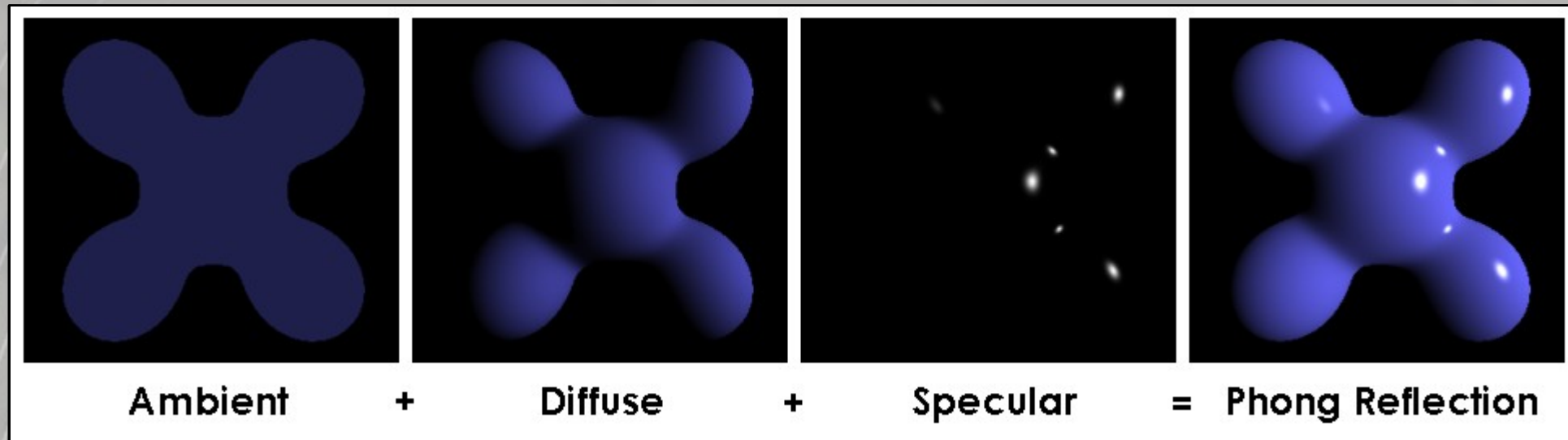
Ламберт



- $\text{OutColor} = \text{materialColor} * \max(0, \cos(\Theta)) * \text{lightColor}$
- $\cos(\Theta) = \text{dot}(\text{lightVec}, \text{info.norm})$

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

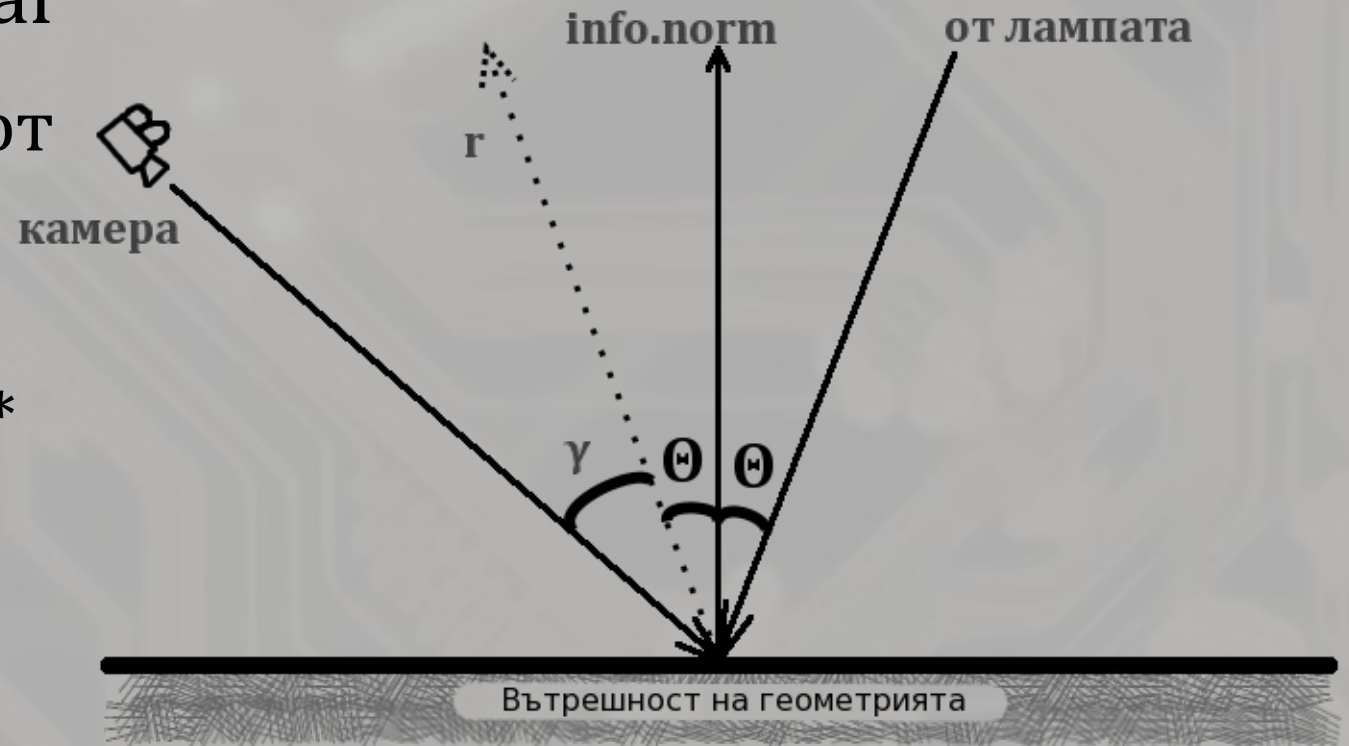
Фонг

- Фонг = Ламберт + specular

- r е отражението на лъча от лампата, спрямо нормалата `info.norm`

- $\text{specular} = \max(0, \cos(\gamma))^{\eta} * \text{lightColor}$

- Параметърът η (обикновено наричан „exponent“) контролира „колко лъскав е материала“



$$r = \text{reflect}(\text{lightVec}, \text{info.norm})$$
$$\cos(\gamma) = \text{dot}(r, -\text{cameraRay.dir})$$