3D графика и трасиране на лъчи v.4.0

http://raytracing-bg.net/

Тема 4

Въведение в камерата
Пресичане с равнини

Съдържание

- Реализация на правоъгълна камера
- Реализация на raytracing алгоритъма
- Реализация на пресичане с равнина
- Шейдъри и текстури
- Домашни :)

Новини

- Сорскод на проекта (вижте и във форума)
- Записът от лекция 3
- Календар

Разходка из кода на проекта

- http://code.raytracing-bg.net (намира се в github)
- 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()`), итензитет (`::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. A към т. B (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:
  - Скаларно произведение:

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

    - Ако a и b са нормирани, то 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;
```

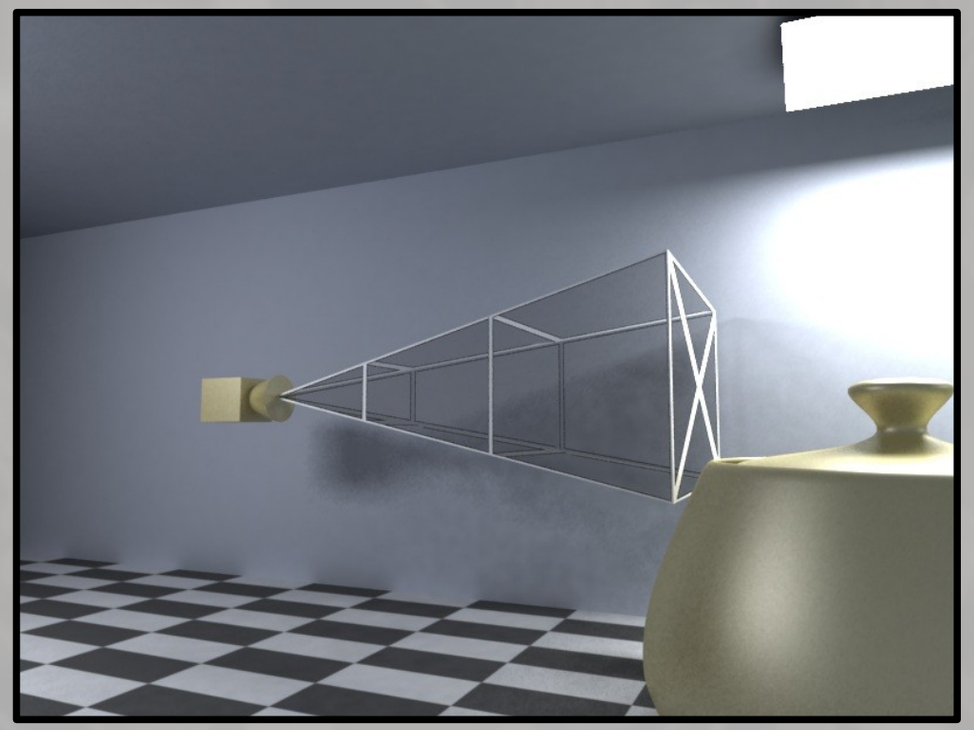
Raytracing cheatsheet

- 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 *= rotationAroundX(PI/6) * rotationAroundZ(PI/2)
```

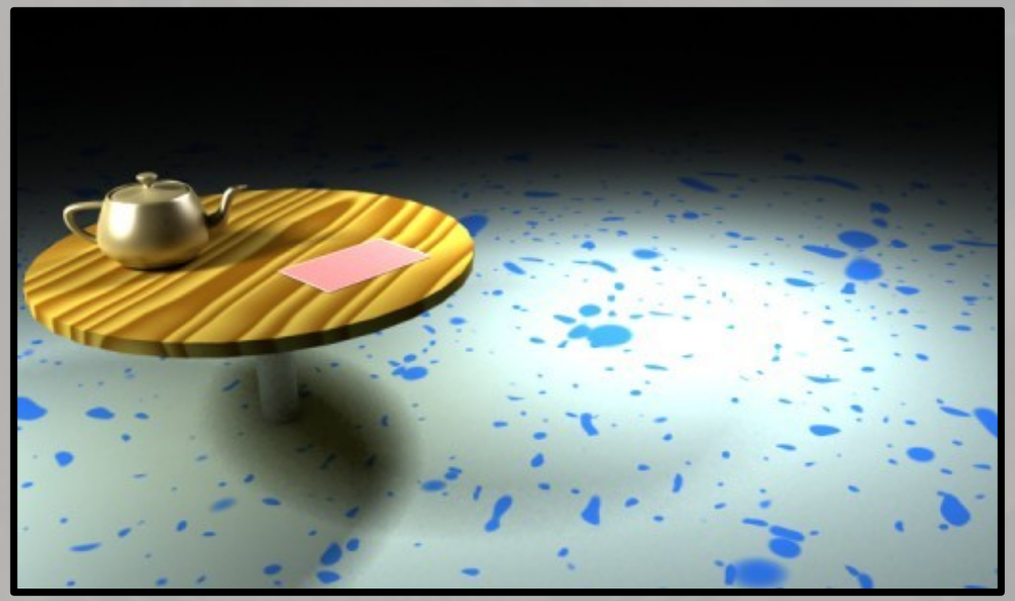
Реализация на камерата

- 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



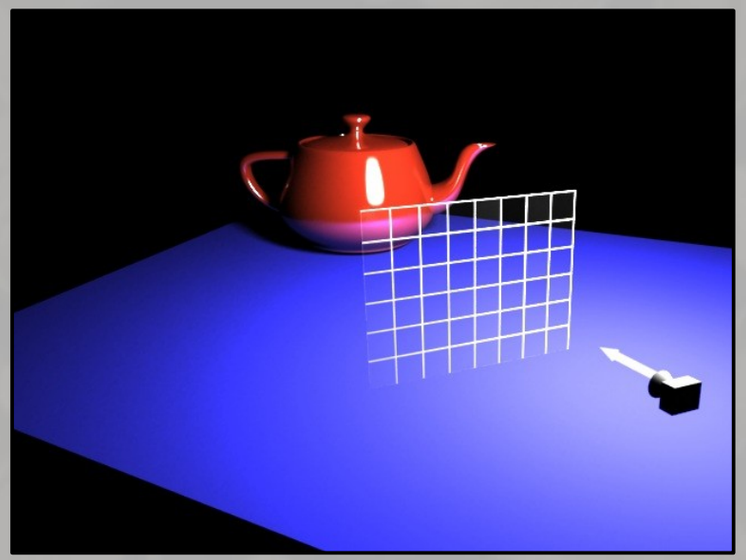
Реализация на камерата

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



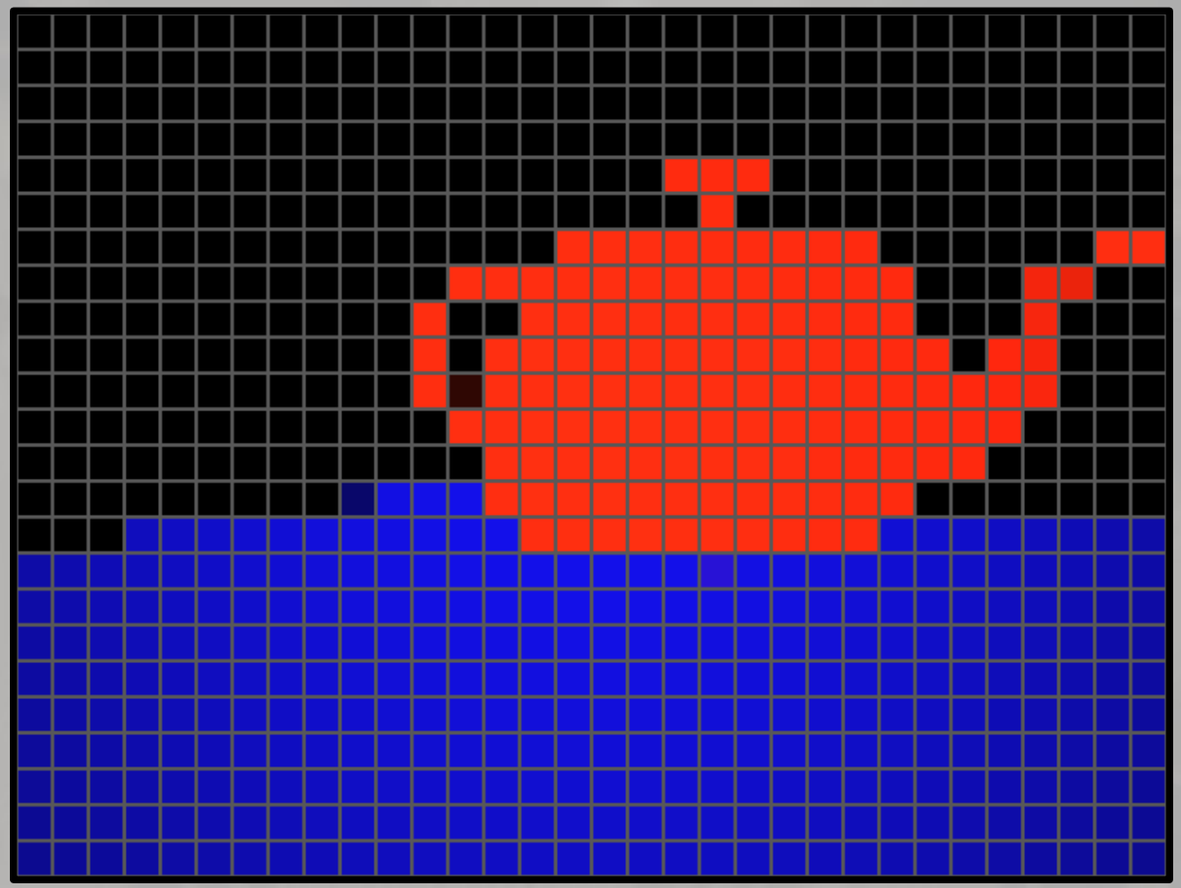
Реализация на камерата

- 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

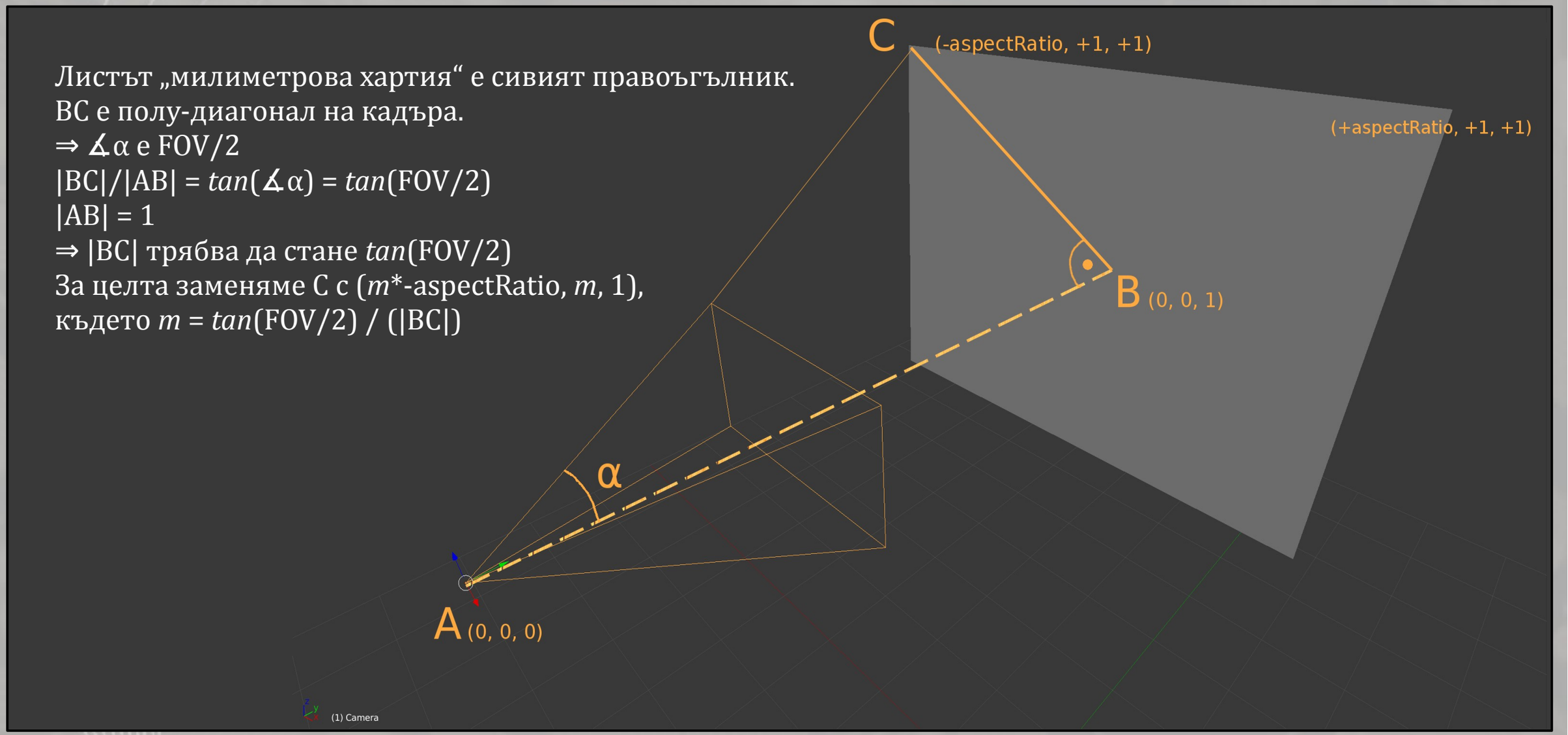
- Ще симулираме листа „милиметрова хартия“
- Трябват ни просто три от краищата на листа (горен ляв, горен десен, долен ляв)
- Намиране на произволен пиксел (по зададени X, Y)
  - За X направлението, интерполираме между горен ляв и горен десен край
  - За Y направлението, интерполираме между горен ляв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

Реализация на камерата

- Как генерираме трите краища?
  - Ще предполагаме, 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)
    - Т.е., с краища (±aspectRatio, ±1, +1)
  - Ще мащабираме листа, така че диагоналния ъгъл да стане колкото искаме (FOV)
  - За наше удобство, ще задаваме FOV-а в градуси

Реализация на камерата

Листът „милиметрова хартия“ е сивият правоъгълник.
BC е полу-диагонал на кадъра.
⇒ ∡α е FOV/2
|BC|/|AB| = *tan*(∡α) = *tan*(FOV/2)
|AB| = 1
⇒ |BC| трябва да стане *tan*(FOV/2)
За целта заменяме C с (*m**-aspectRatio, *m*, 1),
където *m* = *tan*(FOV/2) / (|BC|)



Реализация на камерата

- След скалирането по *m*, получаваме краищата на „лист милиметрова хартия“, задаващ правилния aspect ratio и FOV;
- Ще приложим трите ротации (roll, pitch и yaw) върху получените краища

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
  ```

Nodes

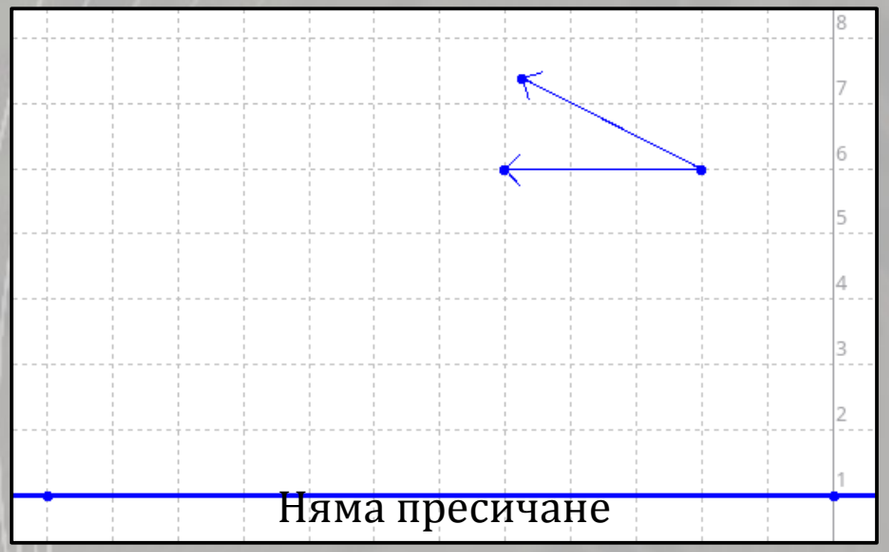
- Всеки обект в сцената има както геометрия, така и шейдър
- Намираме най-близката геометрия, пресичаща лъча, и викаме шейдъра на Node-а ѝ

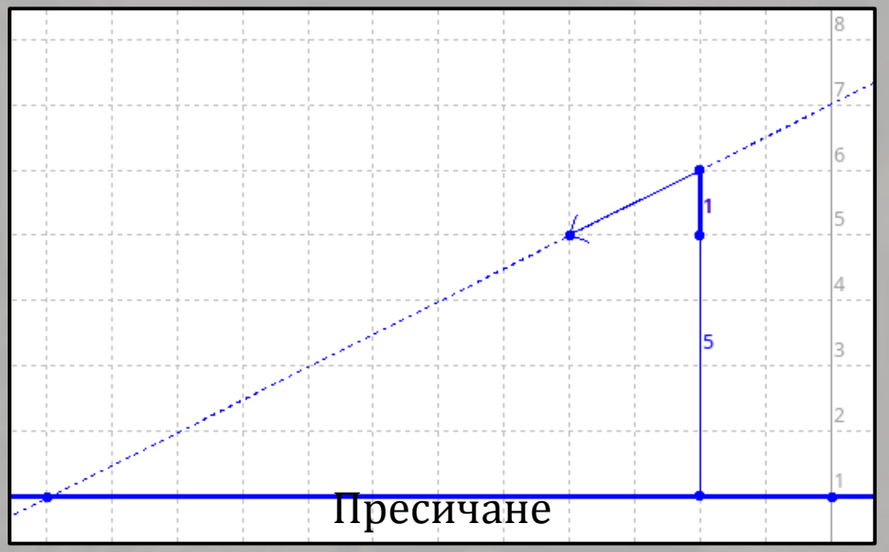
Raytracing алгоритъм

```
Function raytrace(ray):
    closestDist = +∞
    closestNode = None
    for each node in sceneNodes:
        intersectionInfo = intersect(ray, node)
        if intersectionInfo.distance < closestDist:
            closestDist = intersectionInfo.distance
            closestNode = intersectionInfo.node
    if closestDist == +∞:
        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:





Домашни

- Домашните ще са базирани на текущия сорс
- Подробности на сайта на курса