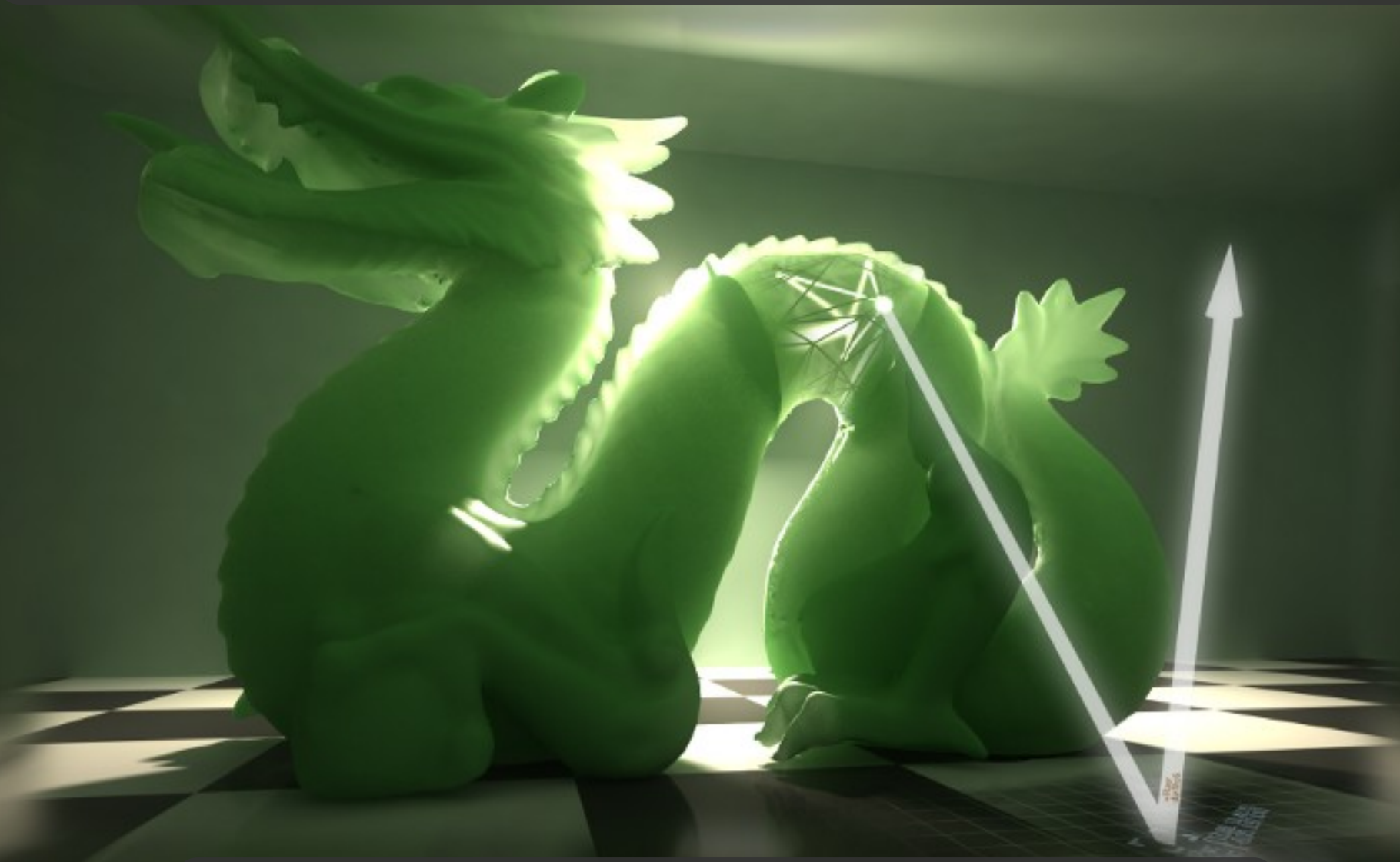


3D графика и трасиране на лъчи v.2.0



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



Тема 10

Осветление

Не-точкова светлина

Насочена светлина

Монте-карло методи за изчисляване на осветление

Дълбочина на полето

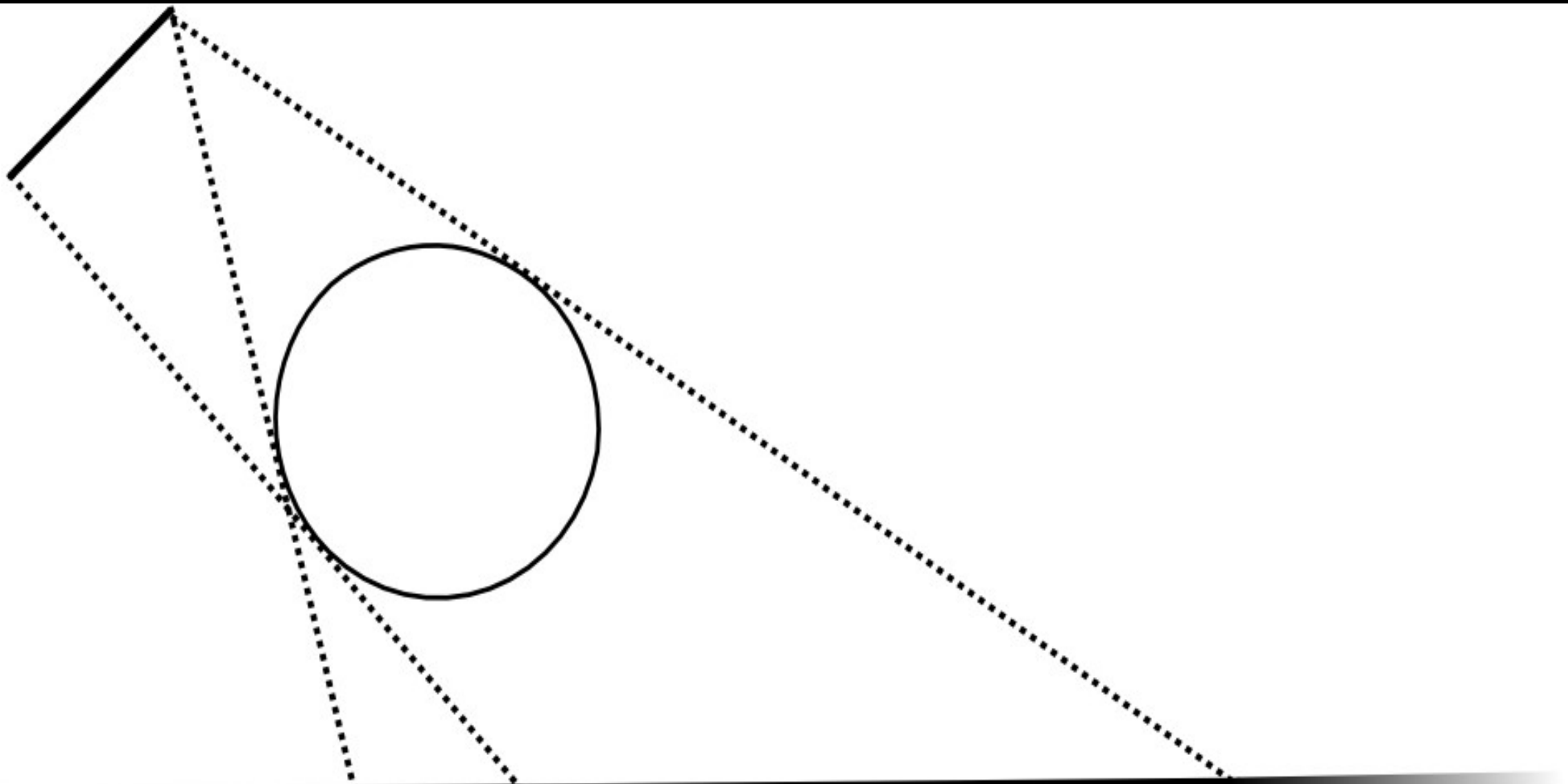
Съдържание

- Осветление в raytracing-a
- Не-точкова светлина
 - Меки сенки
 - Правоъгълна светлина
 - Реализация чрез Monte Carlo метод
- Дълбочина на полето
 - Реализация чрез Monte Carlo метод

Меки сенки

- Досега използвахме само един вид лампи – точкови
 - Те са идеализация, в реалния свят няма „точкови лампи“
- При точковите лампи, дадена точка може да бъде или изцяло огряна от дадена лампа, или изцяло в сянка
 - Следователно има рязка граница на ръба на сенките
- В реалния свят, лампите имат крайно лице
 - Така дадена точка може да бъде изцяло огряна, изцяло в сянка, или огряна само от част от лампата
 - Вместо рязка граница, на ръба на сянката се получават светлосенки (ака penumbra)

Меки сенки (в картинки)



Меки сенки (прод.)

- За да сметнем осветеността на дадена точка трябва да изчислим каква област от лампата е видима от точката
- В общия случай това е нетривиално
 - т.е. няма затворена формула с която може да я пресметнем
- Ще сумираме осветеността до дадената точка, от всяка точка върху лампата
 - Или в математически термини ще интегрираме...

Монте Карло интегриране

- Тук на помощ отново идва Монте-Карло интегрирането
 - Искаме да интегрираме цялото осветление, идващо от лампата, до дадена точка в сцената
 - Генерираме случайни точки върху лампата, и за всяка от тях проверяваме за засенчване, и смятаме Lambert/Phong осветлението за видимите
 - Грешката се изразява в шум (при полусенките)
 - При достатъчен брой точки (семпли), шумът става незабележим

Монте Карло за меки сенки

- Прилагаме дефиницията:

$$\int_{\Omega} f(x) dx \approx \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{\rho(x_i)}$$

- Ω ще бъде единичният квадрат (и следователно $\rho(x_i) = 1$ навсякъде). „Каноничната“ правоъгълна лампа ще е квадрат със страна 1, центриран в $(0, 0, 0)$ и светещ в посока $-Y$.
 - Ще имаме и трансформация (по подобие на Node); така ще можем да слагаме и ориентираме лампи където и както си искаме. Със scaling ще можем да направим лампите и не-квадратни

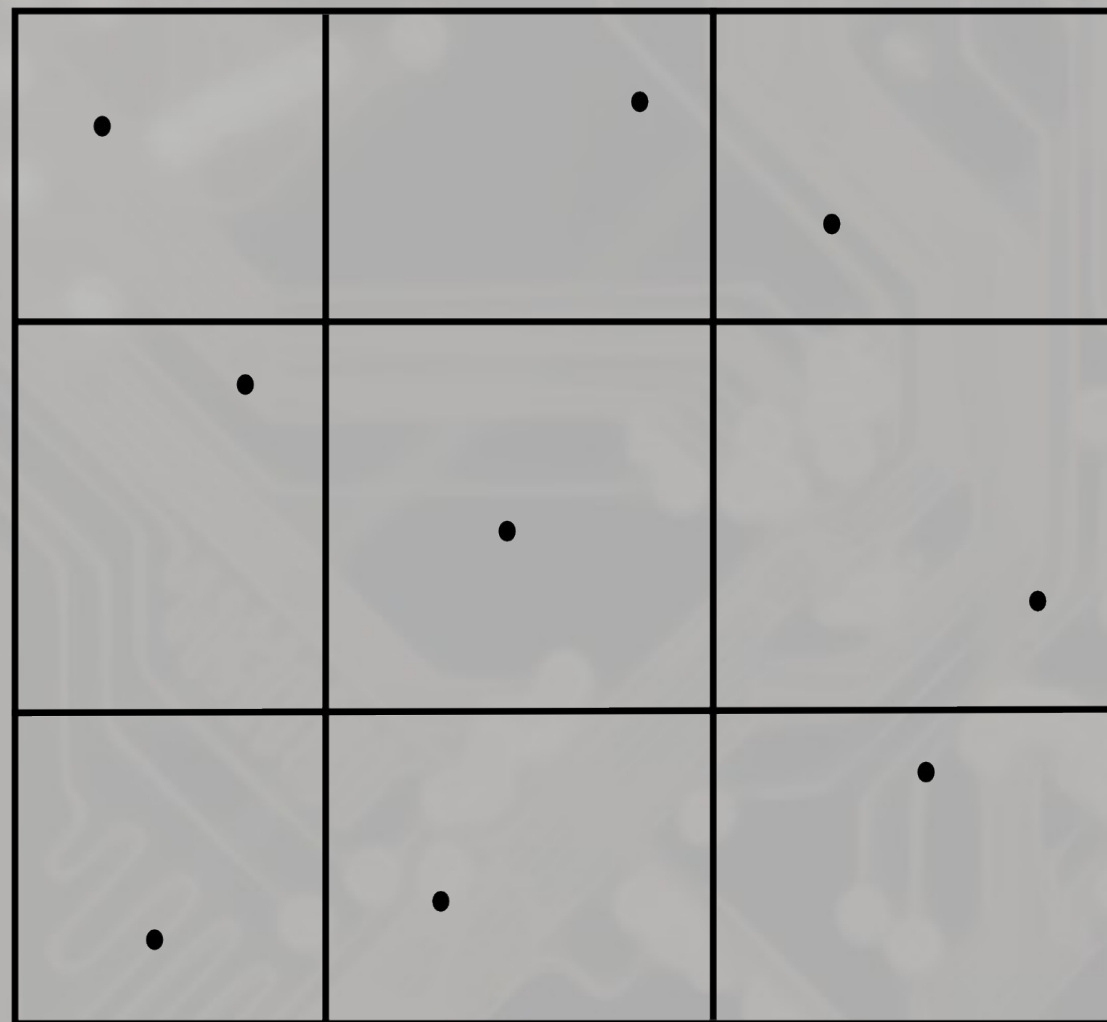
Генериране на точките

- $p[i].x = \text{randfloat}();$
 $p[i].y = \text{randfloat}();$
- При това наивно решение нямаме гаранция, че точките ще са добре разпределени в пространството
 - Получават се клъстери от точки, а голяма част от пространството може да остане непокрита



Генериране на точките

- Едно решение на проблема е stratified sampling
- Разделяме пространството на $n \times m$ области и генерираме по една точка за всяка област
- $p[i].x = (i \% n + \text{randfloat}()) / n$
 $p[i].y = (i / n + \text{randfloat}()) / m$



Интерфейс за светлини

```
class Light {  
protected:  
    Color col;  
public:  
    virtual int getNumSamples(void) = 0;  
    virtual void getNthSample(int sampleIdx, const Vector& hitPos, Vector& samplePos, Color& color) = 0;  
};
```

Добрия стар PointLight

```
class PointLight: public Light {  
    Vector pos;  
public:  
    int getNumSamples(void) { return 1; }  
    void getNthSample(int sampleIdx, const Vector& hitPos, Vector& samplePos, Color& color)  
    {  
        color = col;  
        samplePos = pos;  
    }  
};
```

Промени по шейдърите

```
Color Foo::computeColor(Ray ray, IntersectionInfo& info)
{
    Color result = ....;
    for (each light) {
        Color sum(0,0,0);
        for (each sample for light)
            sum += ....;
        result += sum / nSamples;
    }
    return result;
}
```

Правоъгълна лампа

- След всичко това, вече можем имплементираме правоъгълна лампа (а и не само =-)
- Трансформация, която позиционира лампата в пространството
- `width` и `height` на лампата са следствие на трансформацията
- За семплирането на лампата ще използваме `stratified sampling`
 - `xSubd` и `ySubd` ще са параметрите, които определят по колко семпла да се вземат съответно хоризонтално и вертикално от лампата

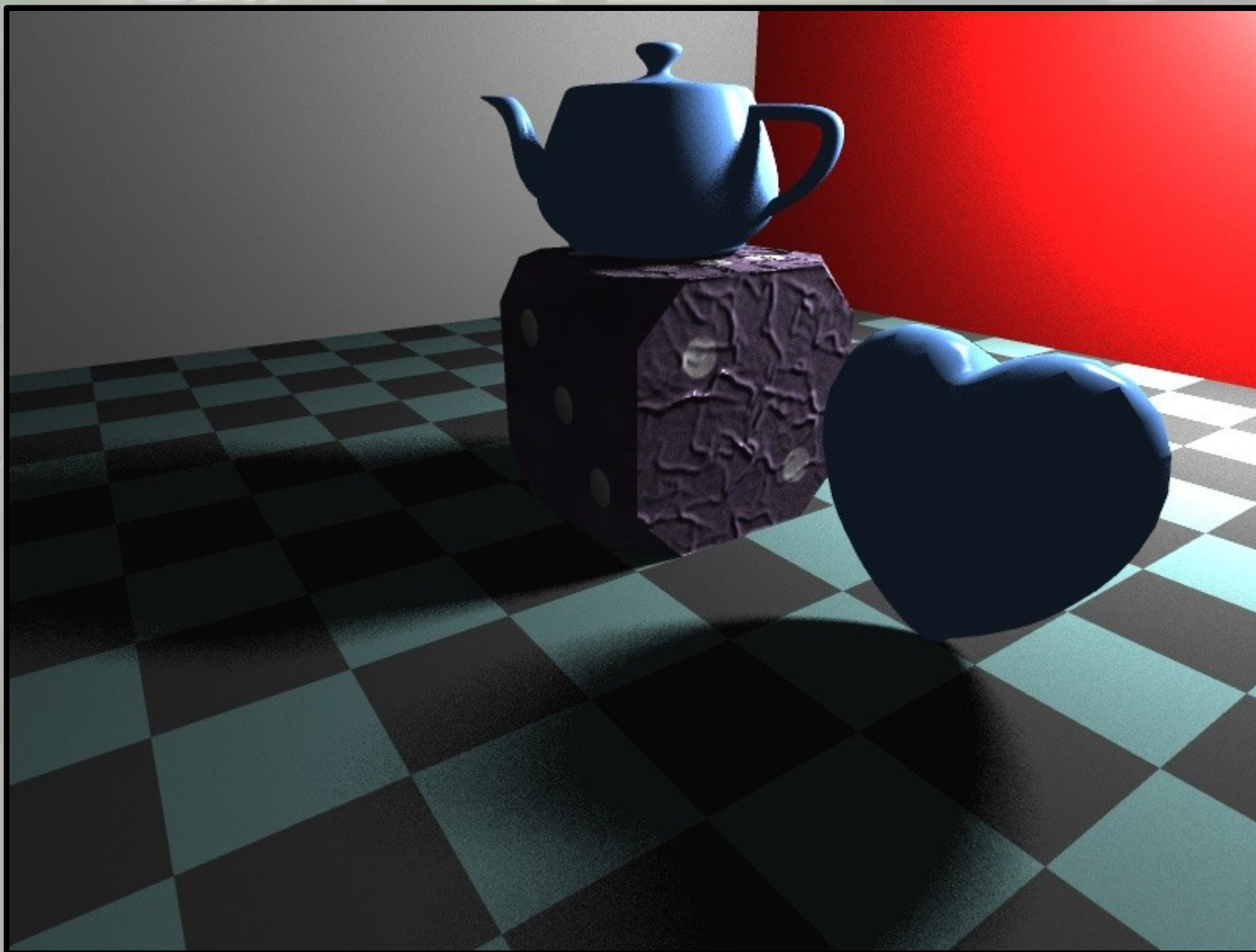
Правоъгълна лампа (прод.)

```
class RectLight: public Light {  
    float width, height;  
    int xSubd, ySubd;  
    Transform T, IT;  
public:  
    int getNumSamples(void) { return xSubd * ySubd; }  
    void getNthSample(int sampleIdx, const Vector& hitPos, Vector& samplePos, Color& color);  
}
```

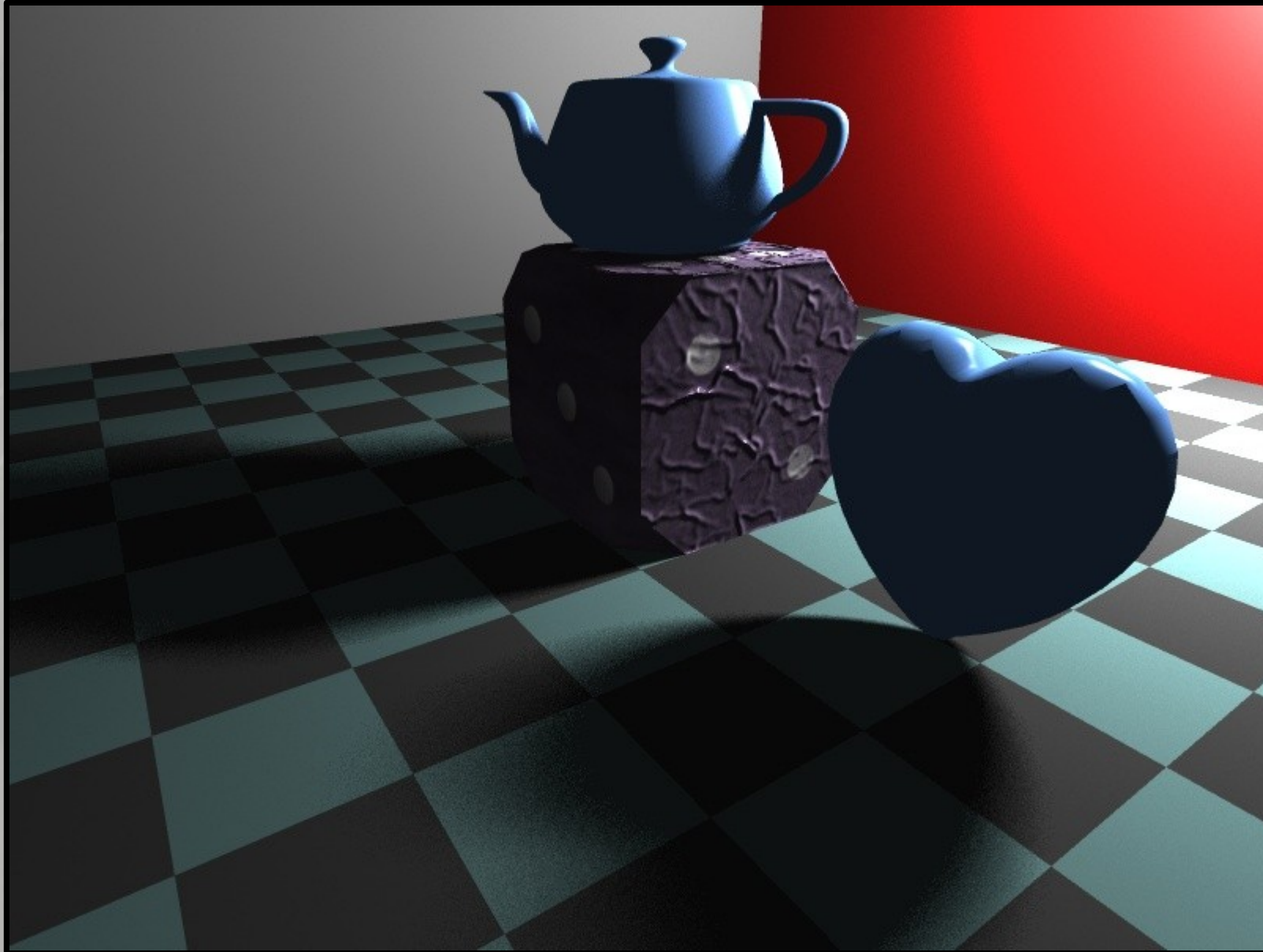
Правоъгълна лампа (прод.)

```
void RectLight::getNthSample(int sampleIdx, const Vector& hitPos, Vector& samplePos, Color& color)
{
    float x = (float) (sampleIdx % xSubd + randomFloat()) / xSubd;
    float y = (float) (sampleIdx / xSubd + randomFloat()) / ySubd;
    samplePos = Vector(x - 0.5, 0.0, y - 0.5);
    Vector hitPos_LS = IT.transformPoint(hitPos);
    if (hitPos_LS.y < 0) color = col;
    else                color = Color(0, 0, 0);
    samplePos = T.transformPoint(samplePos);
}
```

Правоъгълна лампа (random sampling)



Правоъгълна лампа (stratified sampling)





Идеи и идейки

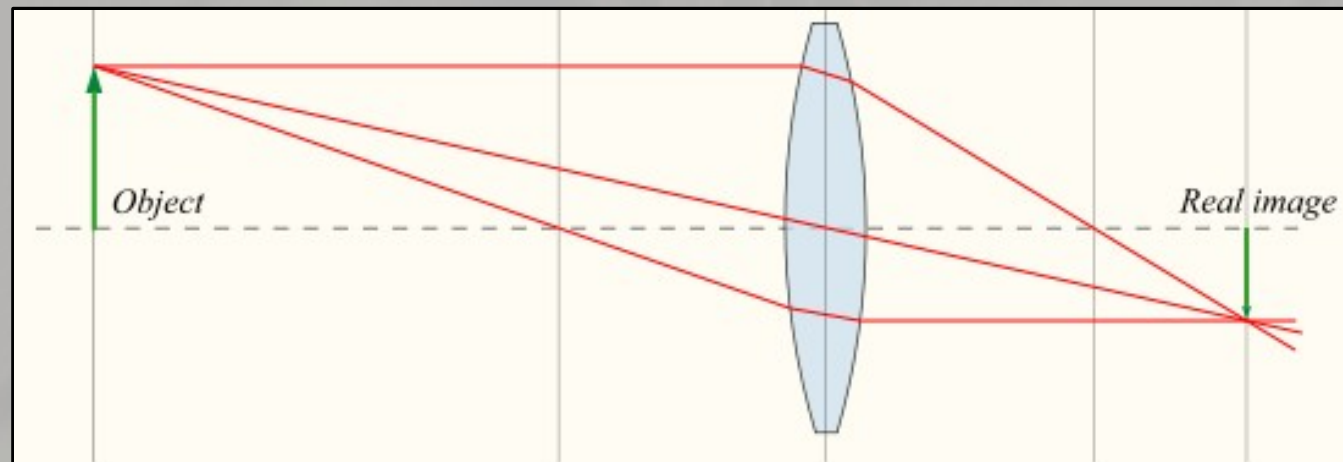
- Shape lights
 - Лампа, която може да се „закачи“ за произволна геометрия
- Spot light
 - Подобен на Point light, но посоките в които свети са ограничени чрез параметри
- Directional light
 - Светлината идва само от една посока от безкрайно далечен обект
- Environment Light
 - Може да семплираме environment-а като лампа (aka HDR lighting)

Дълбочина на полето

- Дълбочина на полето = частта от видимата сцена, която изглежда рязка (не е размита)
- Ефектът е свързан с фокусирането при фотоапаратите и филмовите камери – част от обектите са на фокус, други са извън и изглеждат размазани
- Досега работихме с pinhole camera, при която DOF-ът е безкраен – всичко е на фокус
 - ... и, ако не целим друго, това е идеално – човек предпочита всичко да му е на фокус

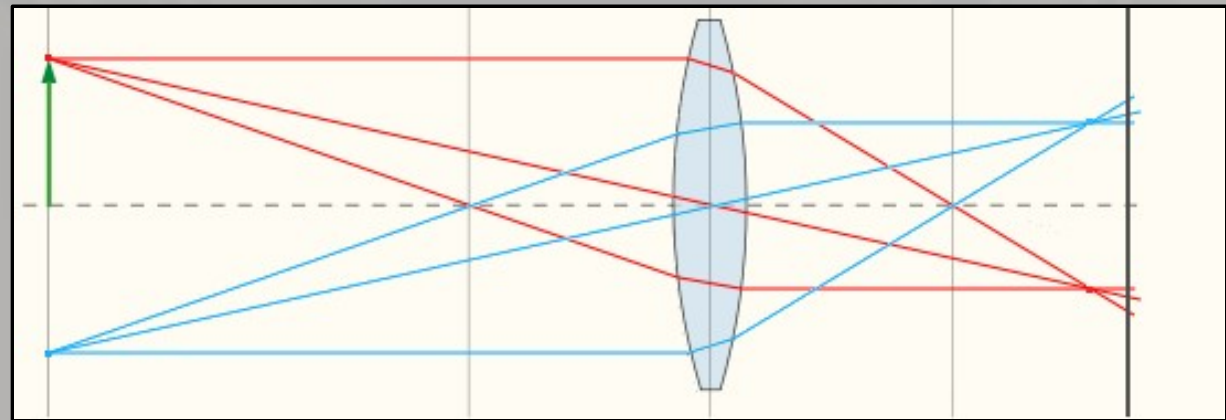
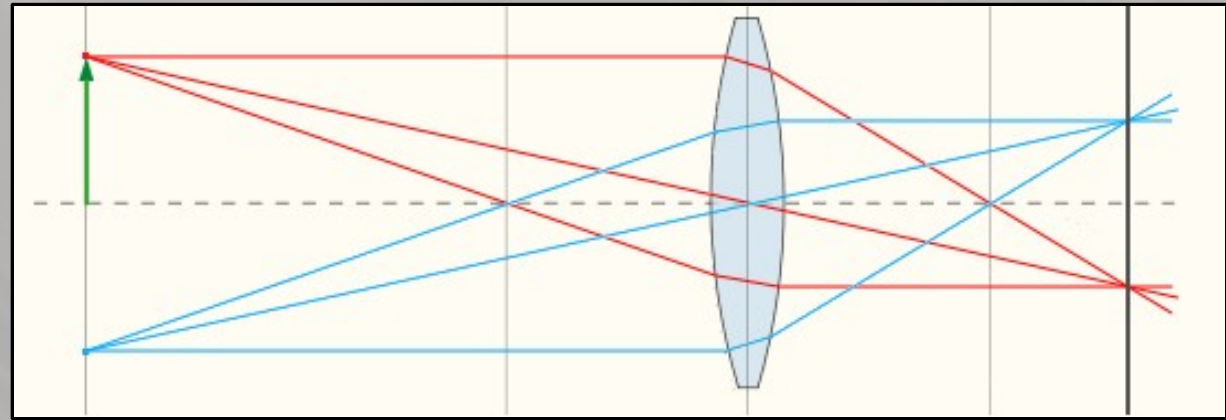
Какво предизвиква размиването на образите (focal blur)?

- Всяка оптична система може грубо да се наподобява чрез една-единствена събирателна леща (лупа)
- Обектът, който искаме да наблюдаваме, излъчва (или отразява) светлина във всички посоки
 - Дадена негова точка също излъчва във всички посоки
 - Ролята на лещата е да събере тези лъчи в една точка в образа



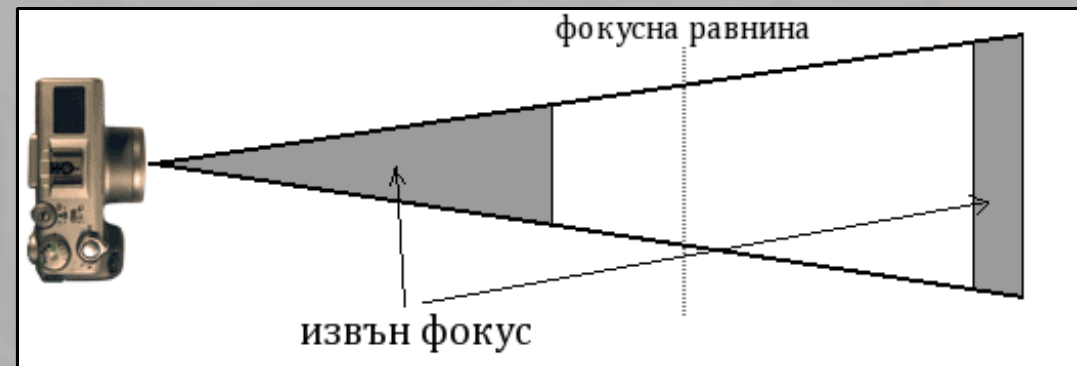
Какво предизвиква размиването на образите (focal blur)?

- Ако преместим лещата, лъчите вече няма да се събират в една точка, а ще се разпръскват в едно малко кръгче; точката ще изглежда разфокусирана
- Чрез местенето на лещата, различни части от сцената попадат на точен фокус



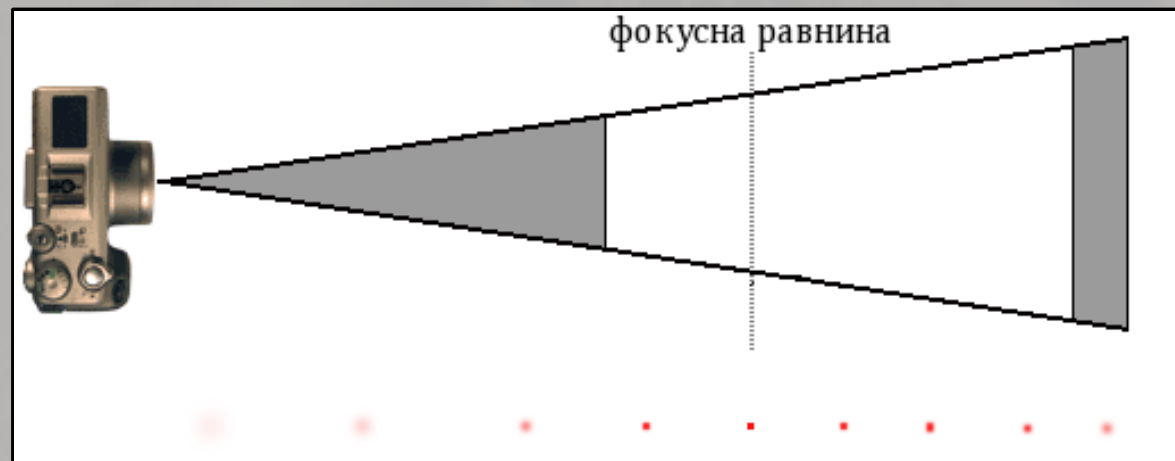
Фокусна равнина

- Разположението на оптиката в обектива (т.е. разстоянието до сензора) определя **фокусната равнина**. Всички обекти на тази равнина са с перфектен фокус
- Колкото повече се отдалечаваме от нея, толкова по-размазани стават обектите



Дълбочина на полето

- На практика, точките близо до фокусната равнина също приемаме за „фокусирани“, защото тяхното размазване (точката се размазва до малко петънце) е недостатъчно за да се забележи от окото (или сензора на камерата)
- Цялото поле, в което приемаме точките за достатъчно резки, представлява **дълбочината на полето**

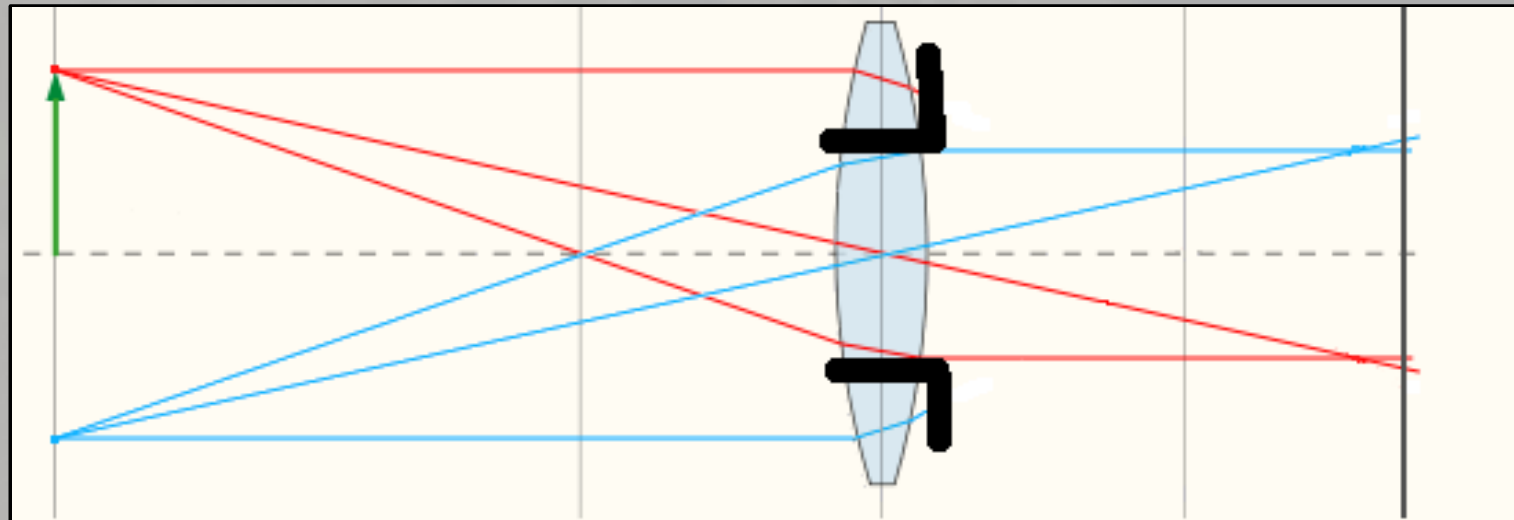


Апертура (бленда)

- В обективите има специален елемент (обикновено е диафрагма), който ограничава светлината, постъпваща към филма/сензора.
 - В човешкото око, зеницата играе ролята на апертура
- Апертурата може да бъде свита („затворена“), с цел да се стесни снопа от светлинни лъчи, минаващ през оптиката
 - В нашата едно-лещова аналогия, това е все едно да намалим диаметъра на лещата

Апертура

- В най-разпръснатата си точка (при лещата), лъчите вече са в по-тесен сноп, затова разфокусирането е по-малко. Т.е., при по-затворена апертура, размазването не е толкова голямо. Затварянето на апертурата увеличава Depth of field



Относителна апертура

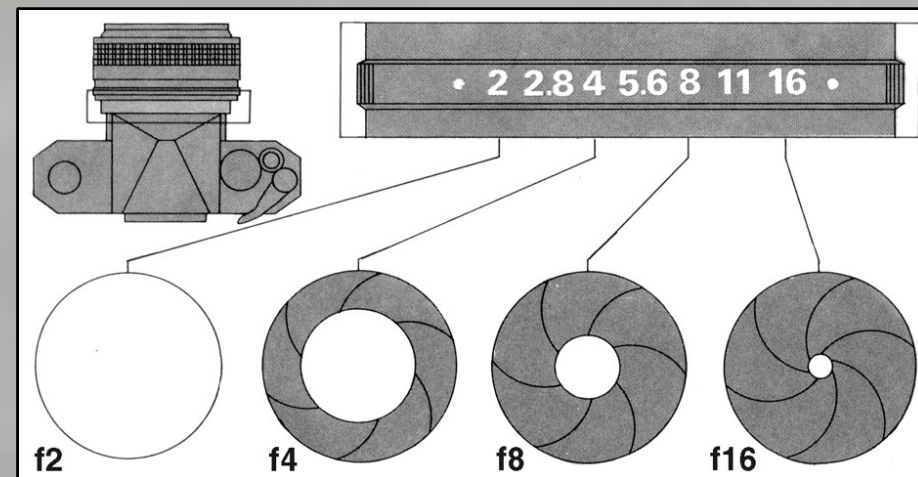
- Тъй като големината апертурата в обектива зависи от големината на самия обектив, обикновено се използва отношението между фокусното разстояние на обектива и диаметъра на входната зеница:

$$k = f/D$$

- Колкото по-голямо е това число, толкова по-“отворен“ е обектива (повече светлина, по-плитък DOF)
- Обикновено под f -число (бленда) се има предвид $f/\text{число}$
 - Бленда 4 ще рече $f/4$, т.е. $k = 1/4 = 0.25$

Апертура

- Във фотографията се използват почти изключително f -числа. По-ниско f -число означава по-отворен обектив
- Има максимална апертура (минимално f -число) – при нея обектива пропуска най-много светлина и дава най-плитък DOF



Примерни f -числа:

- $f/1.0$ - $f/1.2$: най-светлосилните обективи; изключително плитък DOF
- $f/3.5$: нормална бленда на компактен цифров фотоапарат
- $f/2.1$ - $f/8$: човешкото око (на тъмно и на светло)

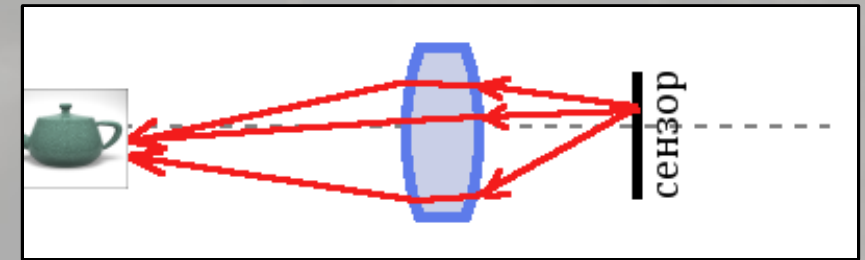
Апертура

- Ниските f -числа се ползват с артистична цел – да се отдели субектът от фона ($f/5.6$ и $f/32$)



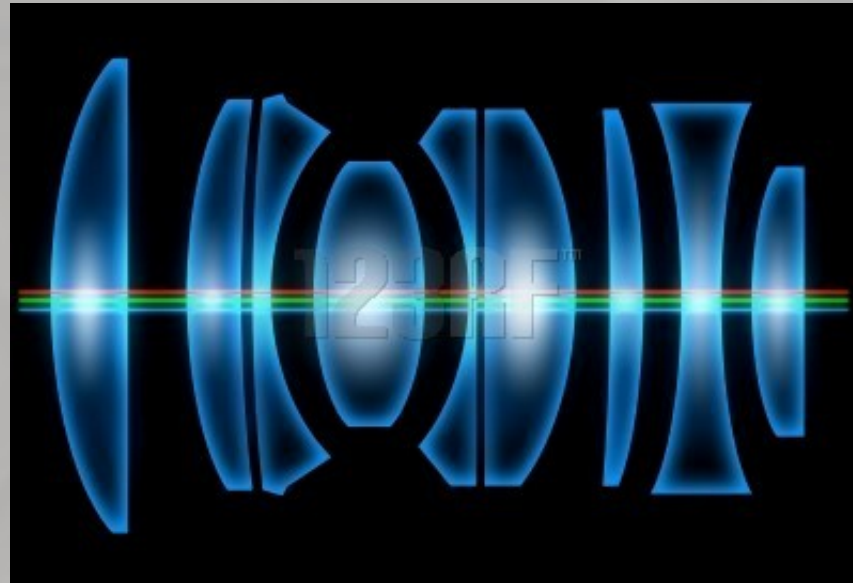
Симулиране на focal blur в raytracing-a

- Можем, за всеки пиксел, да изстрелваме по много лъчи от сензора, да ги пречупим през лещата, и да видим къде ще отидат в сцената
- За съжаление, лещата е най-простият, но и най-лошият обектив – притежава всички старателно отбягвани недостатъци
 - [\[demo\]](#)



Симулиране на focal blur

- Да трасираме през истински обектив?



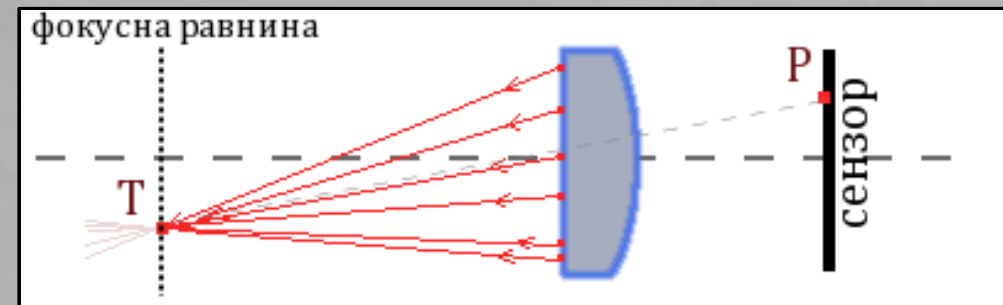
- За съжаление, истинските обективи са с много елементи, някои от които са и с доста сложни форми (например, асферичните елементи). Ще е ужасно бавно!

Симулиране на focal blur

- Ще се върнем на идеята с единична леща, само че, вместо да пречупваме лъчите, директно ще ги изстрелваме от повърхността на лещата
 - Зафиксираме си определен пиксел (P), и намираме образа му T върху фокусната равнина
 - Изстрелваните лъчи започват от случайни точки от челната повърхност на лещата, и са насочени към T.
 - За удобство, ще приемам, че челната повърхност на лещата е плоска

Симулиране на focal blur

- Т.е., така симулираме „събирането“ на лъчите от лещата, само че в обратната посока
- Обектите, които не са точно на фокус, ще участват (ще бъдат удряни от лъчи) в различни (съседни) пиксели
- С този алгоритъм, не изстрелваме излишни лъчи



Симулиране на focal blur

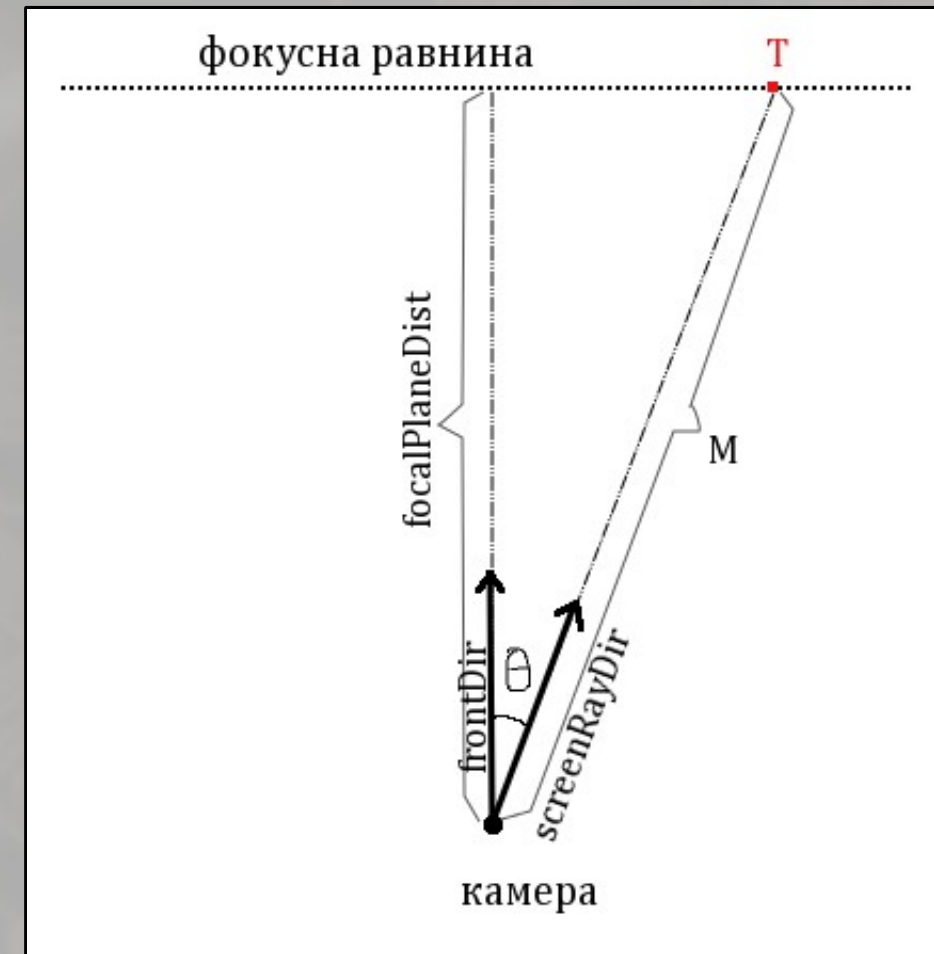
- Реално не е необходимо изображението върху сензора да е обърнато наопаки. Можем да ползваме `getScreenRay()`, както и досега, и този лъч да пресечем с фокусната равнина. Това ще ни бъде точката T . После стартовите точки върху лещата остават същите (тя е симетрична)
- Не е необходимо да сечем лъча с фокусната равнина. Достатъчно е просто да мащабираме `ray.dir` с подходящ множител M :
 - $T = \text{camera.pos} + \text{ray.dir} * M$

Симулиране на focal blur

- Ако `frontDir` е посоката „напред“ на камерата, а `focalPlaneDist` е разстоянието до фокусната равнина, то

$$M = \text{focalPlaneDist} / \cos(\Theta)$$

$$\cos(\Theta) = \text{frontDir} \cdot \text{screenRayDir}$$



Симулиране на focal blur

- Ако `rightDir` и `upDir` са посоките „надясно“ и „нагоре“ на камерата, и приемем, че `camera.pos` е в средата на лещата, то стартовите точки по повърхността на лещата може да генерираме с

$$\text{ray.start} = \text{camera.pos} + u * \text{rightDir} + v * \text{upDir},$$

където u , v са случайни числа, равномерно разпределени в единичния кръг (аналог на апертурата тук е радиуса на кръга: по-малък радиус = по-свита апертура = по-голямо f -число)

Алгоритъм за симулиране на focal blur

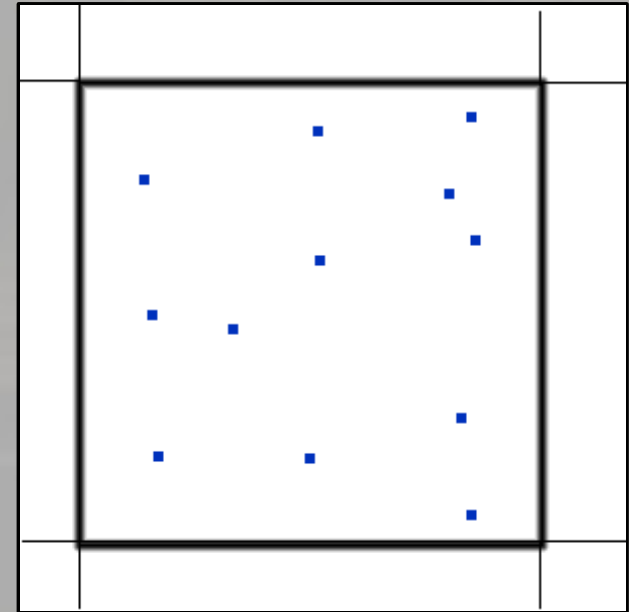
```
function computeFocalBlurredPixelXY(x, y):  
    sum = Color(0, 0, 0)  
    screenRayDir = camera.getScreenRay(x, y).dir  
    M = focalPlaneDist / (frontDir * screenRayDir)  
    T = camera.pos + screenRayDir * M  
    for i = 1 to numDOFSamples:  
        (u, v) = getUniformUnitDisc() * apertureSize  
        ray.start = camera.pos + u * camera.rightDir + v * camera.upDir  
        ray.dir = T - ray.start  
        sum += raytrace(ray)  
    return sum / numDOFSamples
```

Алгоритъм за симулиране на focal blur

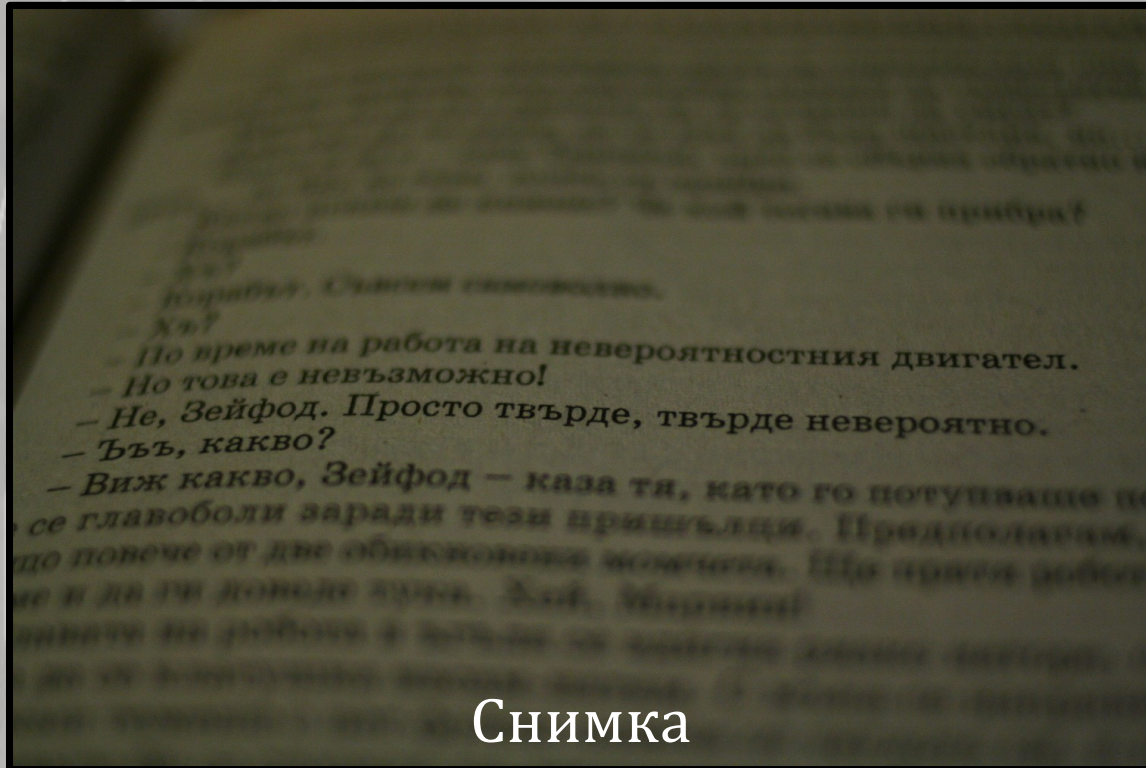
- Изникват няколко въпроса
 - Как да определим focalPlaneDist?
 - Може просто да се задава от потребителя, а може и да реализираме автофокус – да намерим най-близката пресечна точка на frontDir с геометрията
 - Как да определим apertureSize?
 - Това свойство симулира големината на отвора на диафрагмата. Трябва да е обратнопропорционално на f-числото. Тук може да направим пълно и детайлно превръщане от f-число до размер (като вземем под внимание fov, sensor size и други (физически акуратна камера)), или просто като ползваме $1/f$ -числото (quick'n'dirty)

Алгоритъм за симулиране на focal blur

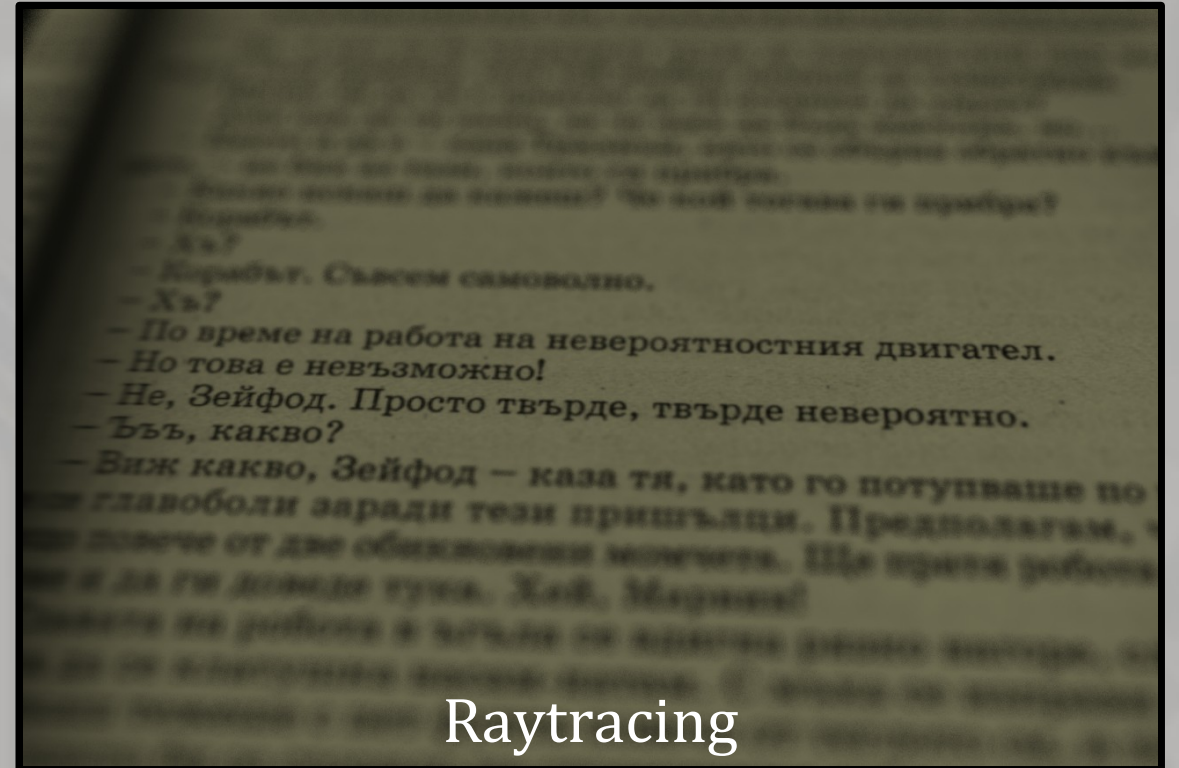
- И какво стана с anti-aliasing-a?
 - В `computeFocalBlurredPixelXY()` така или иначе вземаме доста голям брой семпъли за пиксел; досегашната `antialiasing` логика става излишна. Вместо нея, можем, за всеки семпъл, да сменяме x и y като им добавяме по едно случайно число от $[0..1]$; Така отместваме целта на произволна дробна позиция вътре в пиксела



Резултати



Снимка



Raytracing

Результати

