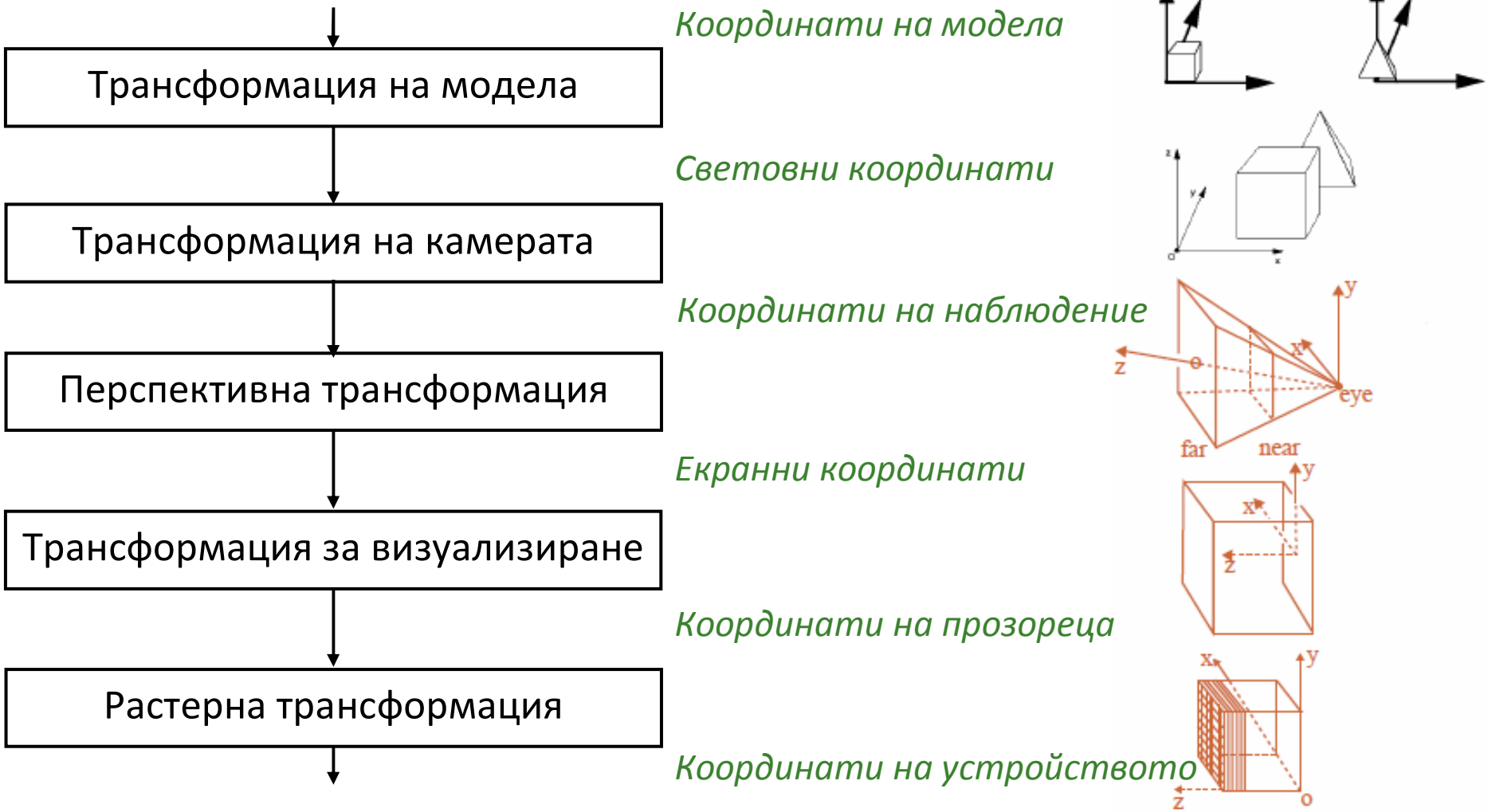
Компютърна графика

Визуализиране с трасиране на лъчи

доц. Милена Лазарова, кат. КС, ФКСУ

Основен графичен конвейер



Реалистично визуализиране

- (меки) сенки
- отражение (огледалност и лъскавост)
- прозрачност (вода, стъкло)
- взаимно отражение (color bleeding)
- сложна осветеност (area light)
- реалистични материали (кадифе, стъкло)
- и др.

Визуализиране

- Основен графичен конвейер
  - примитивите се визуализират последователно
    - ***за всеки обект***
  - няма сенки
  - няма взаимно отразяване
- Други подходи
  - рендериращо уравнение
  - рендиране с трасиране на лъчи (ray tracing)
    - ***за всеки пиксел***
  - radiosity

Основен графичен конвейер

- Допускания при визуализиране с използване на z-буфер и модел на Фонг
  - точкови източници на светлина
  - директно осветяване
    - светлината се излъчва от източника
    - пречупва се поне веднъж
    - достига до наблюдателя
  - непрозрачни повърхности
  - няма сенки

Трасиране на лъчи

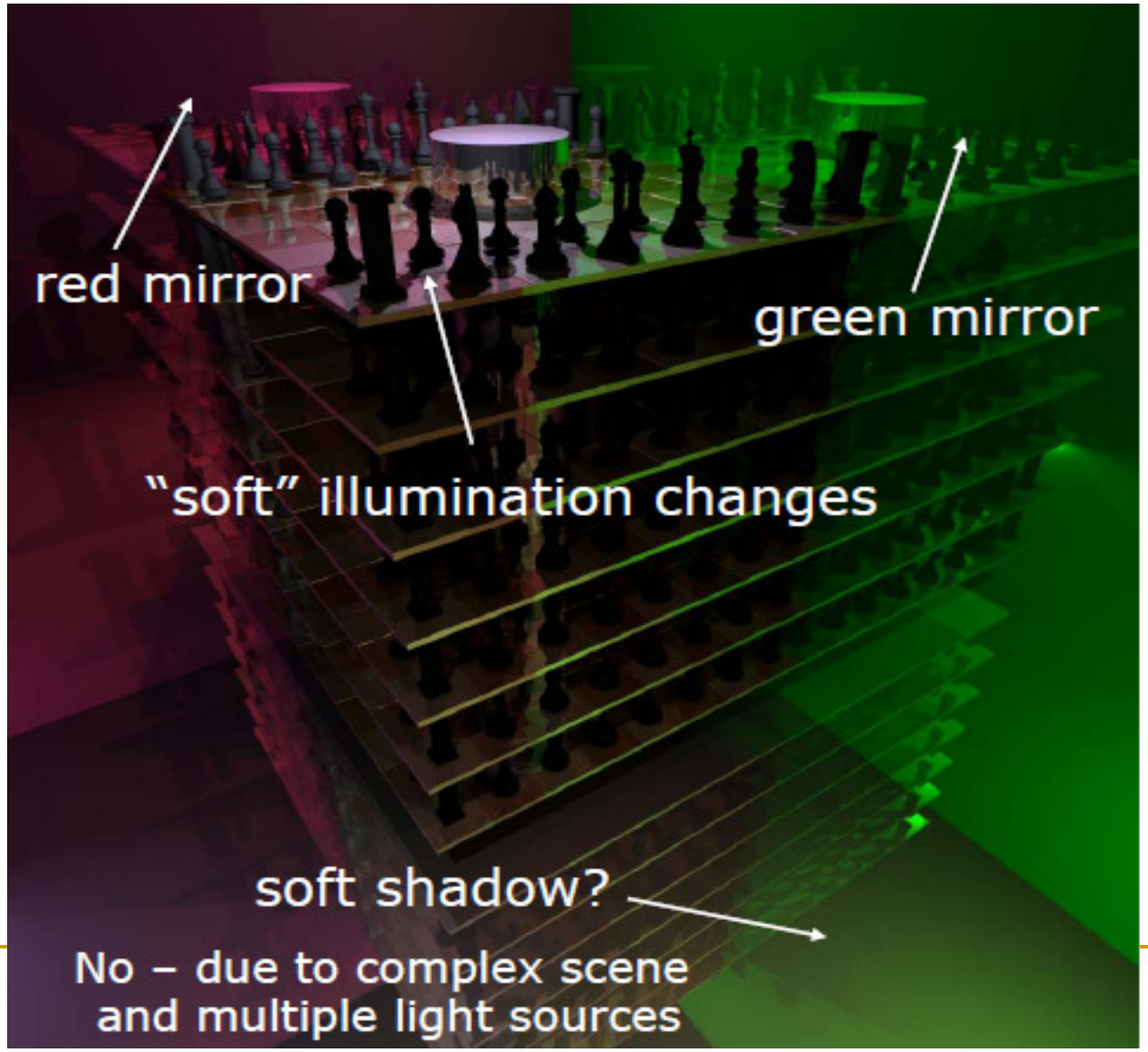
- Премахват се някои от тези допускания и могат да се симулират различни ефекти
  - огледално отразяване
  - сянка
  - прозрачни повърхности (преминаване с отразяване)
  - индиректно осветяване
  - *area* източници на светлина
  - моделиране на мъгла

Трасиране на лъчи

- ***Ray Tracing***
- Цел
  - генериране на реалистични изображения
- Предимства
  - огледално отражение
  - прозрачност
- Недостатъци
  - color bleeding (дифузно отражение)
  - изчислителна сложност

Ray Tracing

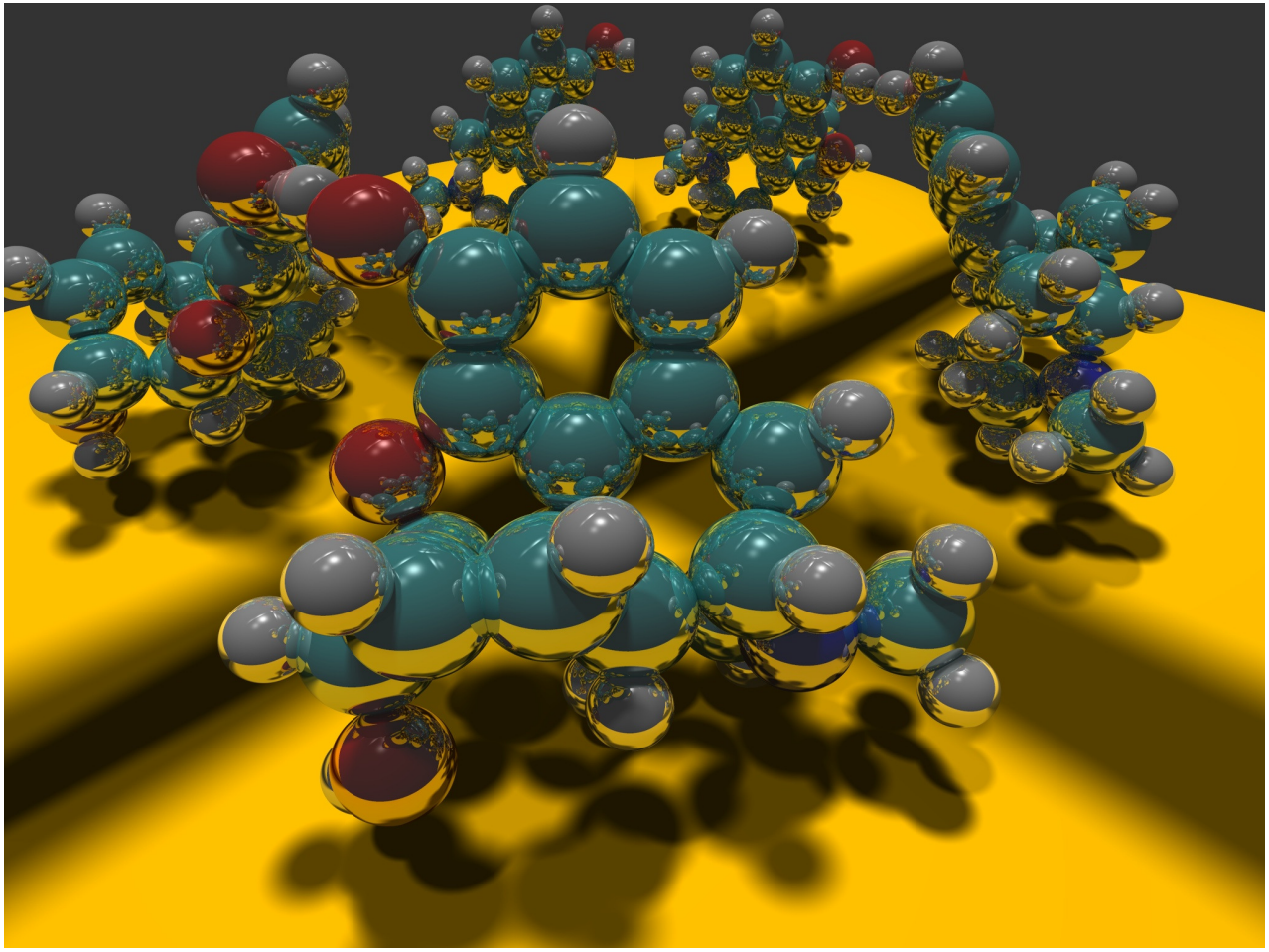
Ray Tracing – ефекти



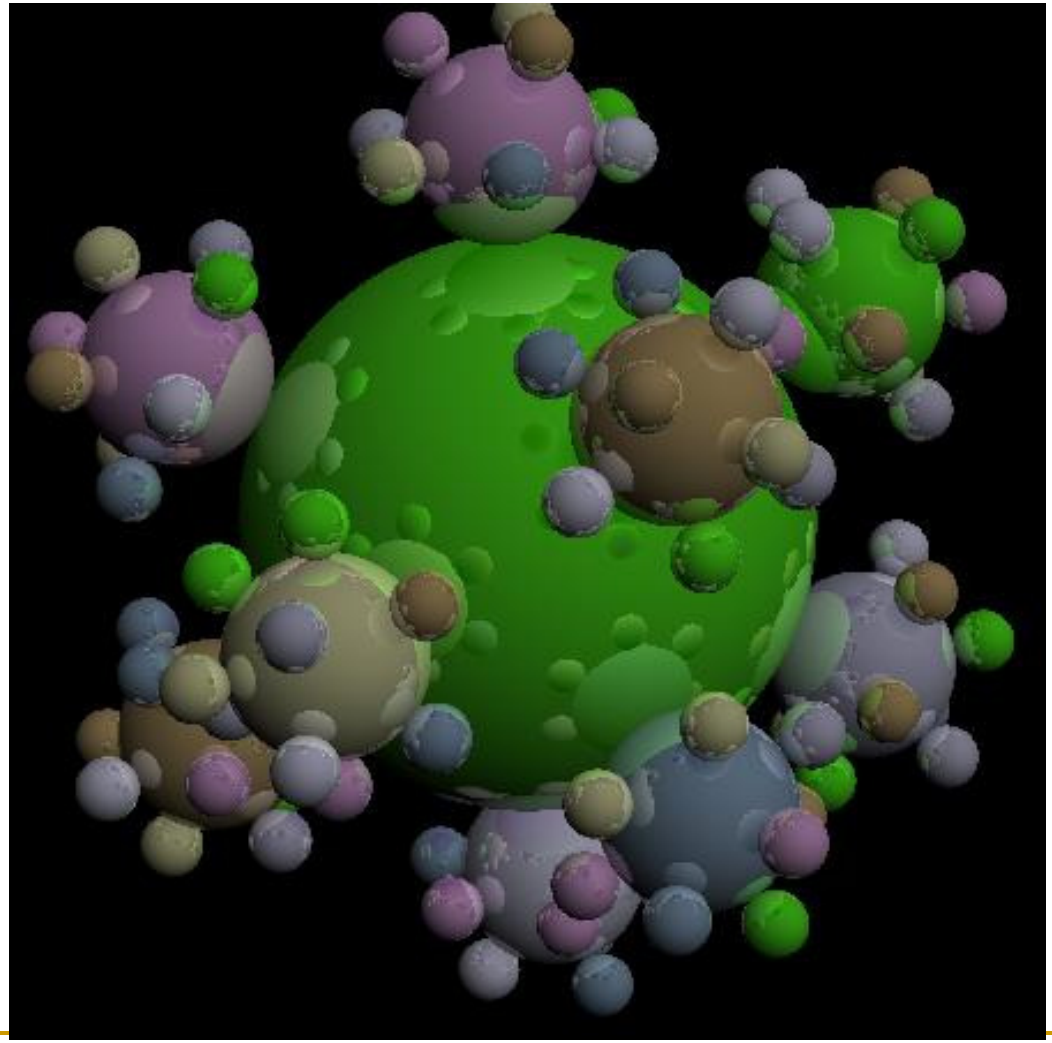
Възможности при трасиране на лъчи

- ***Основен алгоритъм***
  - премахване на скрити повърхности и обекти
    - както при алгоритъма със z-буфер
  - няколко светлинни източника
  - отразяване
  - прозрачност и пречупване
  - тъмни сенки (hard shadows)
- ***Разширения на алгоритъма***
  - меки сенки (soft shadows)
  - motion blur
  - размити отражения (glossiness)
  - дълбочина на наблюдение (finite apertures)
  - полупрозрачност
  - и др.

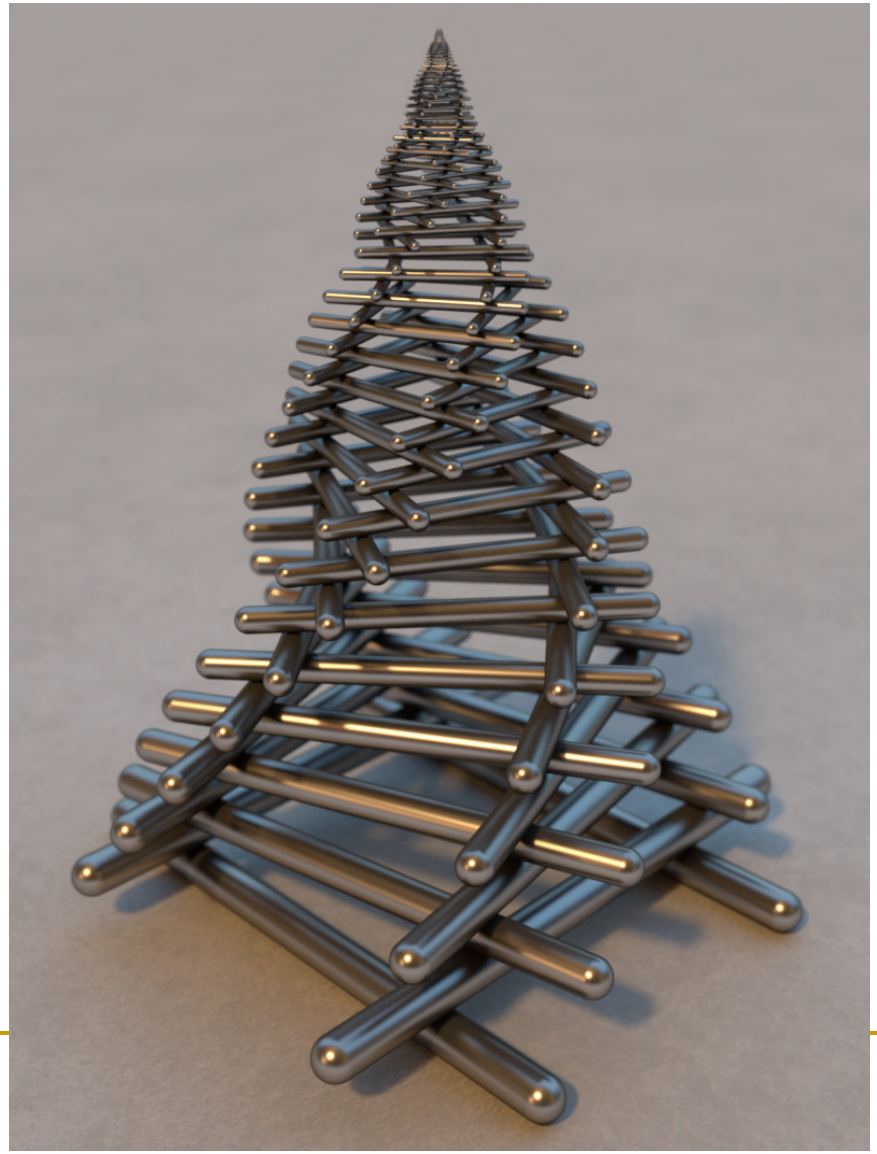
Ray Tracing – примери



Ray Tracing – примери

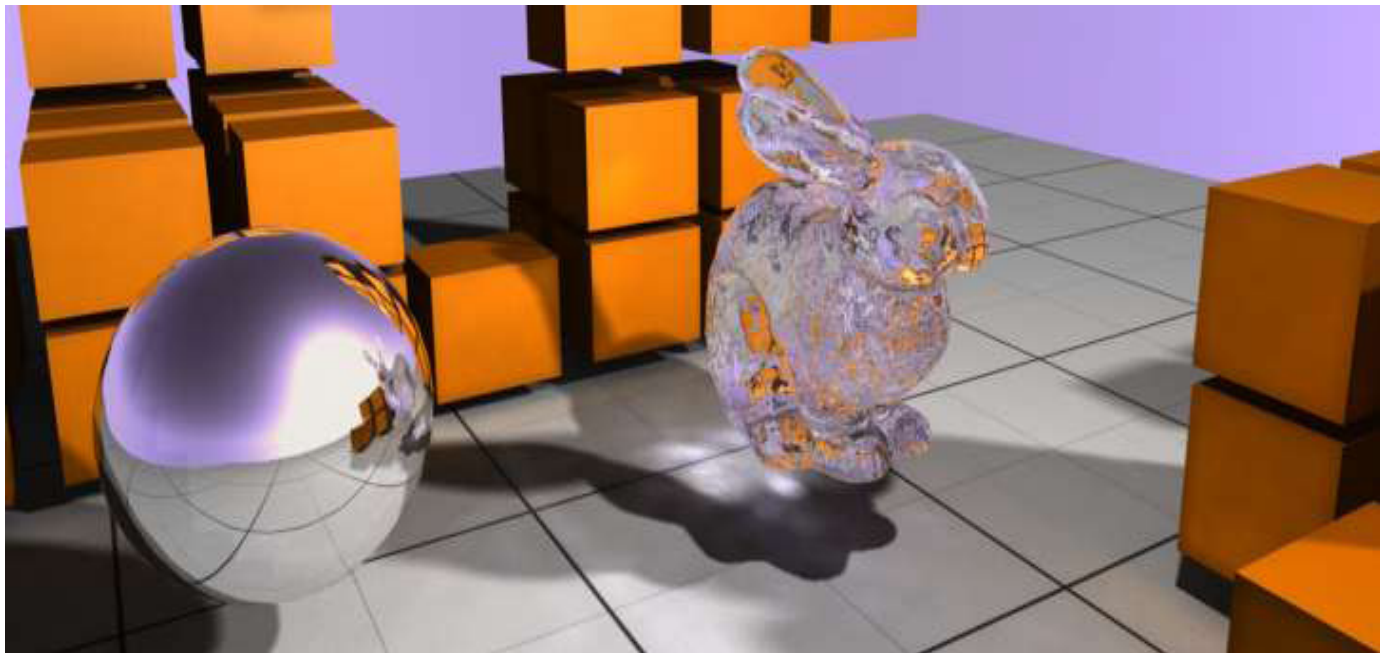


Ray Tracing – примери



Ray Tracing – примери

Ray Tracing – примери



Ray Tracing – примери

Ray Tracing – примери

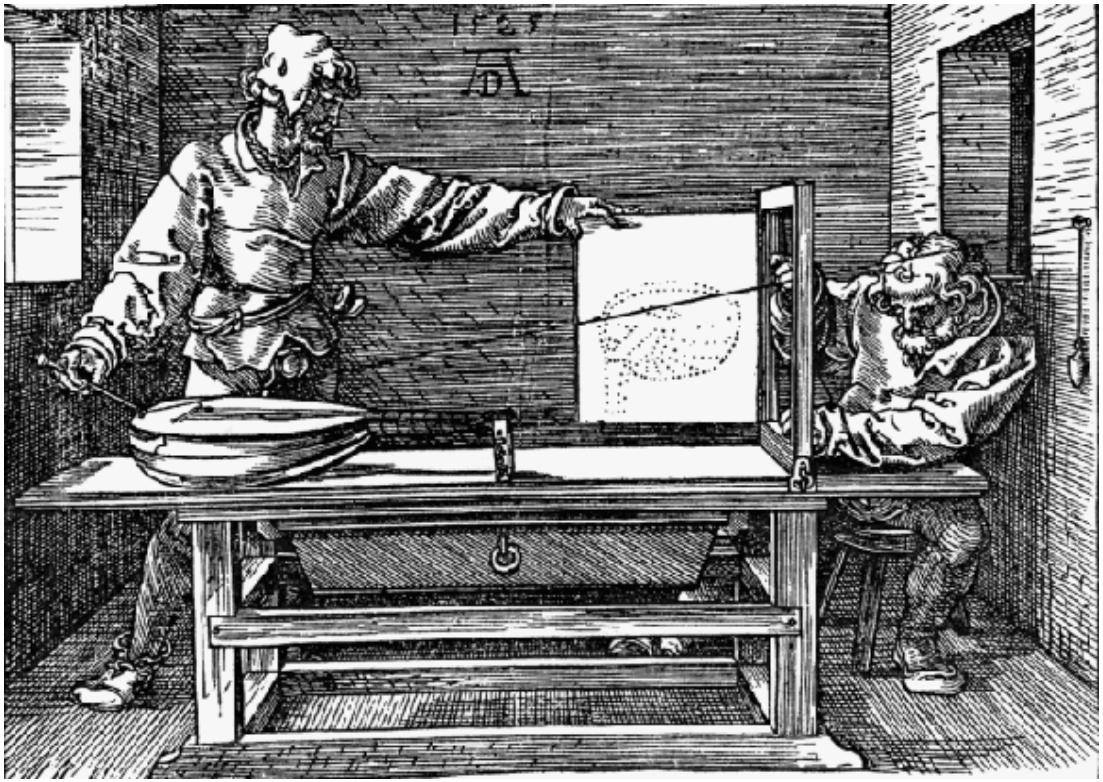
Ray Tracing – примери

Ray Tracing – примери

Ray Tracing – примери

Трасиране на лъчи

- конец от центъра на проекцията (точка на наблюдение) до обектите в сцената
- отбелязва точката на пресичане на конеца с 2D равнина
  - отбелязаните точки са перспективна проекция на 3D обект в 2D равнина



Dürer, 'Underweysung der messung', Nurenberg, 1525

Трасиране на лъчи

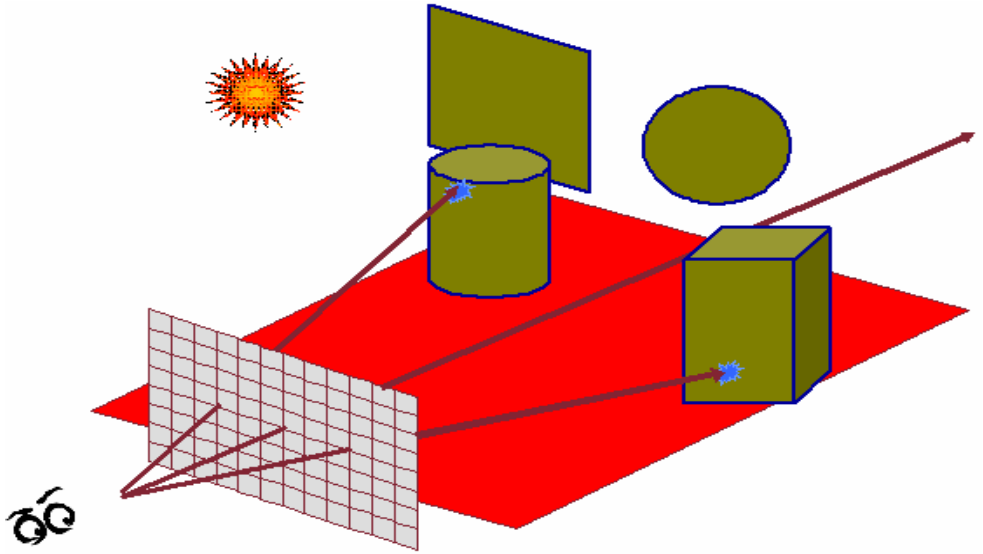
- Визуализиране с трасиране на лъчи
  - проста основна идея
  - вместо да се изобразяват в права посока *безкраен брой лъчи от източника на светлина към обекта и към наблюдателя*

    се изобразяват в обратна посока *краен брой лъчи*

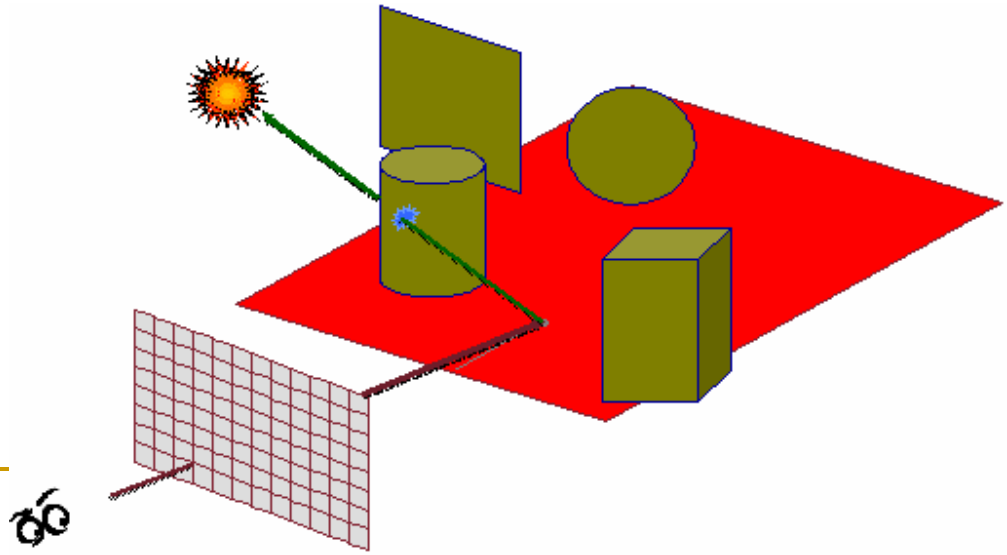
    от наблюдателя през всеки обект към източника на светлина (или други обекти)
- ***Ray Tracing***
  - “изстрелване” на лъчи от наблюдателя през точка (напр. център на пиксел) на виртуално изображение (или прозорец на визуализиране) и изчисляване на цвят/интензитет в точката на пресичане на лъча

Трасиране на лъчи – история

- ***Ray Casting***
  - Arthur Appel, 1968



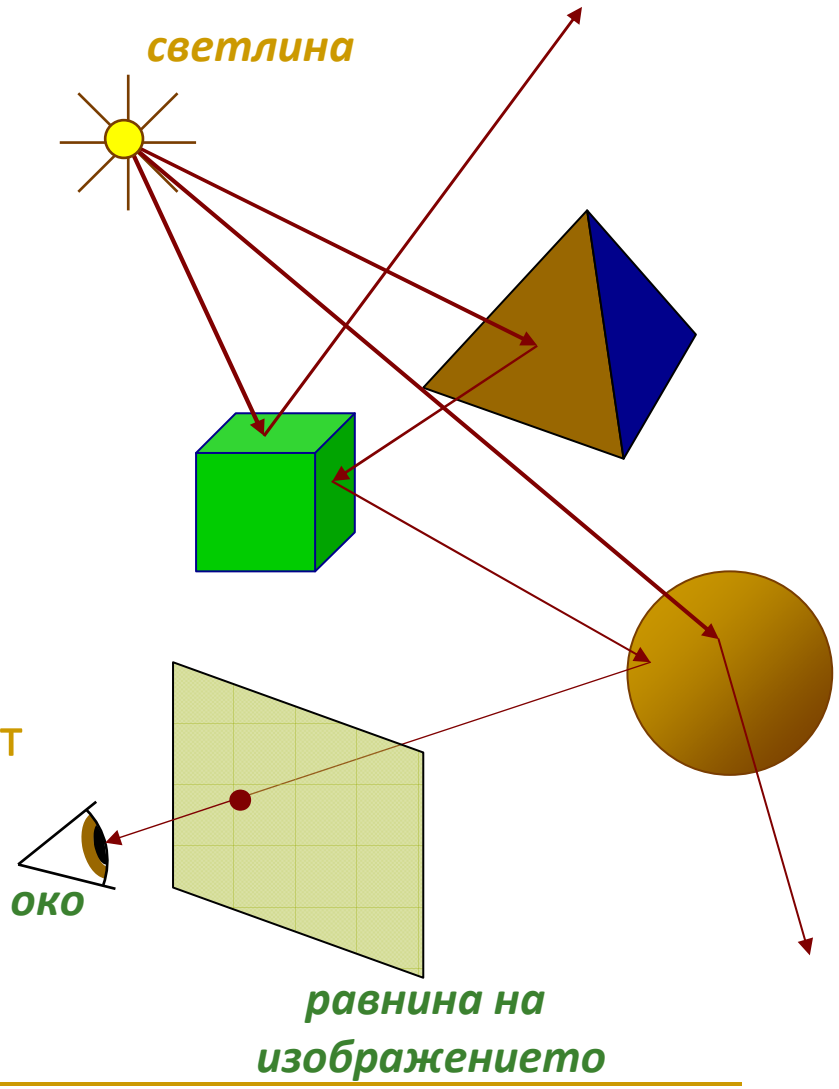
- ***Ray Tracing***
  - Turner Whitted, 1980



“Backward” трасиране на лъчи

- Трасиране на лъча ***напред***

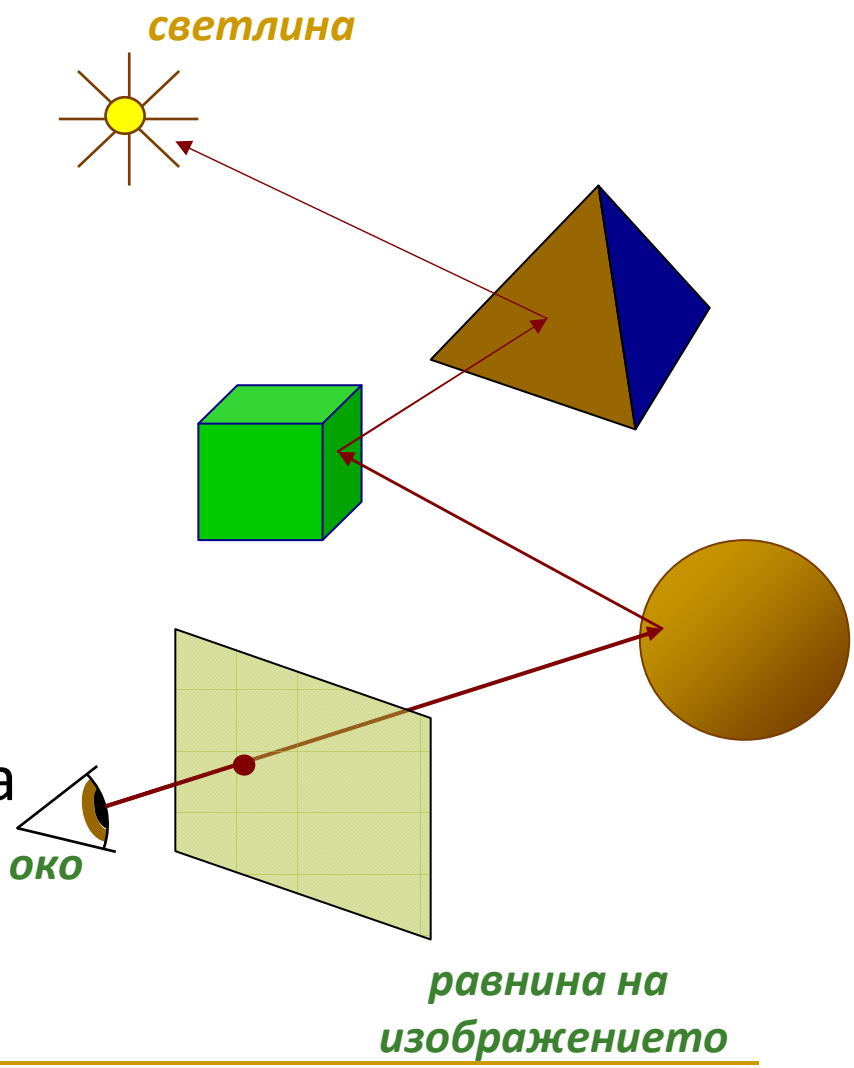
  (във времето) от източника на светлина към потенциалните (множество на брой) обекти в сцената, с които лъча взаимодейства
- Проблем
  - повечето лъчи никога не достигат в близост до наблюдателя
  - много неефективно, тъй като се изчисляват множество лъчи, които не се “виждат”



"Forward" трасиране на лъчи

- Трасиране на лъча ***назад***

  (във времето) от наблюдателя към точка на екрана
- По-ефективно
  - изчисляват се само видимите лъчи (тъй като се започва от наблюдателя)
- Обикновено ***алгоритъм с трасиране на лъчи*** се нарича алгоритъма с трасиране напред (*forward* ray tracing)



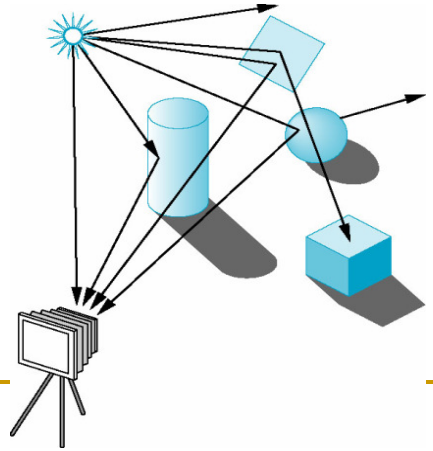
Трасиране на лъчи

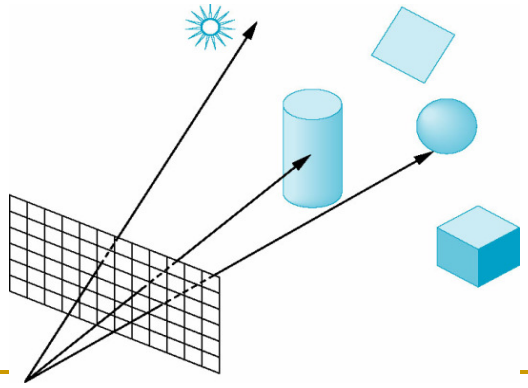
- Алгоритъм за генериране на прецизно реалистично изображение
- ***видимостта се определя пиксел по пиксел (per-pixel)***
  - трасират се един (или повече) лъчи за всеки пиксел
  - изчислява се позицията на най-близкия обект (триъгълник, сфера и др.) за всеки лъч
- Реалистични изображения
- Изчислителна сложност

Същност

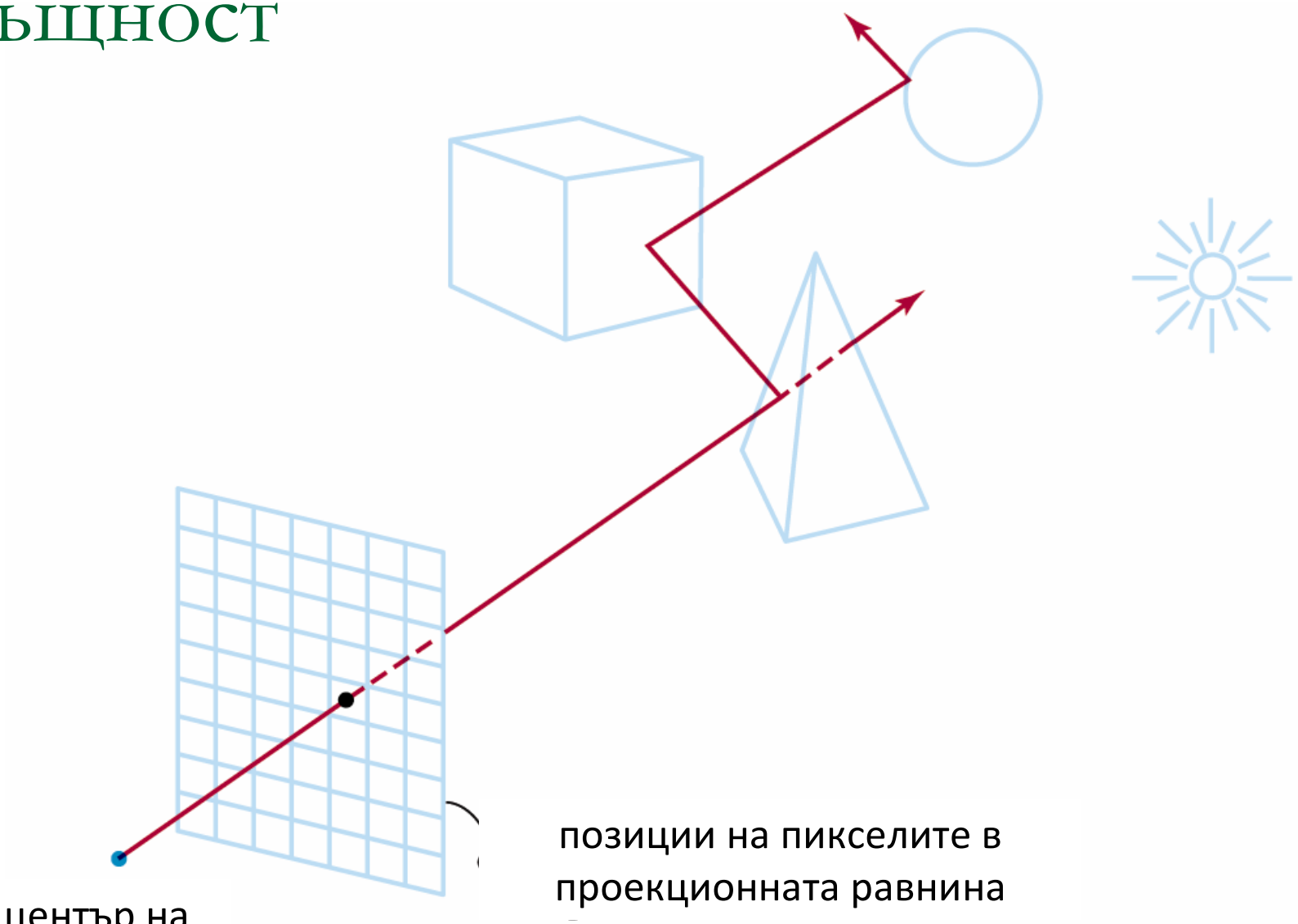
Алгоритъм с трасиране на лъчи

- Трасиране на придвижването на лъч светлина
- Моделиране на взаимодействията му с обектите в сцената
- Генериране на вторични лъчи при пресичане на лъча с обект и определяне на тяхното взаимодействие с обекти в сцената
  - ***пречупване (reflection), сянка (shadow), пропускане (transmission)***





Същност



позиции на пикселите в проекционната равнина

център на проекцията

Същност

- Алгоритъм с трасиране на лъчи
  - Излъчва се единствен лъч от всеки пиксел в проекционната равнина към сцената по протежение на проекционен лъч
  - Определя се с кои повърхности се пресича лъча и в какъв ред са разположени повърхностите спрямо разстоянието до пиксела
    - най-близката до пиксела повърхност е видимата повърхност за този пиксел
  - Отразява се лъча от видимата повърхност на ъгъла на огледално отражение
  - За прозрачни повърхности се излъчва лъч през повърхността в посоката на пречупване
  - Процесът се повтаря за вторичните лъчи

Същност

- Алгоритъм с трасиране на лъчи
  - Условия за терминиране
    - лъчът не пресича никакви повърхности
    - лъчът пресича повърхност, която не отразява
    - направени са максимален брой отражения на лъча

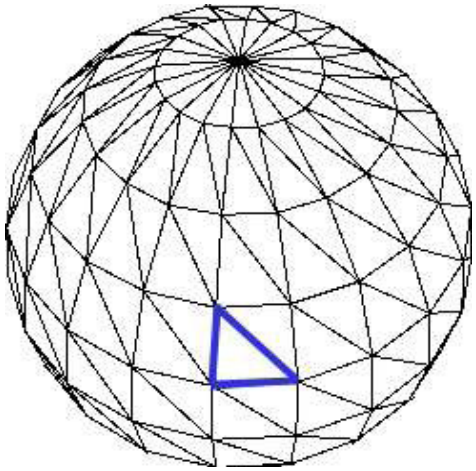
Трасиране на лъчи

- ***Ray Tracing – подзадачи***
  - Генериране на лъчи
    - лъчи от наблюдателя през точките в проекционната равнина към сцената
  - Определяне на най-близък обект по протежение на лъча
    - първото пресичане на лъч и обект от сцената
  - Изчисляване на осветеност
    - използва се модел на осветеност за определяне на интензитет/цвят в точката на пресичане
  - Рекурсивно генериране на вторични лъчи

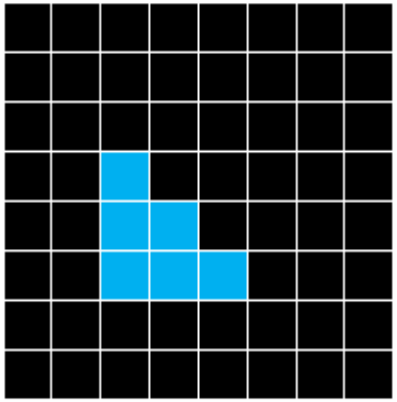
Трасиране на лъчи vs. растеризиране

- ***Ray Tracing*** vs. ***Scan Conversion***
  - Растеризиране на обектите с алгоритъм на сканиращата линия

```
for each object in the scene
    for each triangle in the object
        pass vertex geometry and colors to render them
```



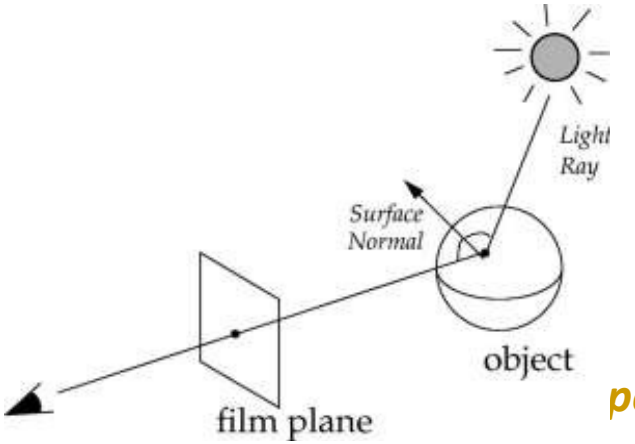
растеризиране на триъгълник на екрана



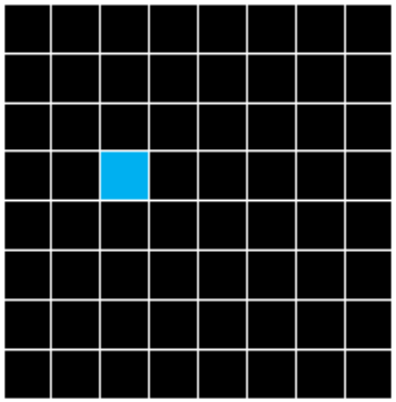
Трасиране на лъчи vs. растеризиране

- ***Ray Tracing*** vs. ***Scan Conversion***
  - Растеризиране на обектите с алгоритъм с трасиране на лъчи

```
for each sample in the film plane
    determine the closest object in the scene hit by
        a ray through that sample
    set the color based on the calculation of the
        illumination model for the intersected object
```



растеризиране на
ресичането на лъча на екрана



Трасиране на лъчи vs. растеризиране

- ***Ray Tracing*** vs. ***Scan Conversion***

  Разликата

  - трасиране на лъчи
    - итерира в пространството на ***пикселите***
  - сканираща линия
    - итерира в пространството на ***възлите***

Трасиране на лъчи

- Имплементацията на основен (минимален) алгоритъм с трасиране на лъчи

Paul Heckbert source code в 10×20см.

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>=1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){puts("P3\n32 32\n255");while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*minray!*/
```

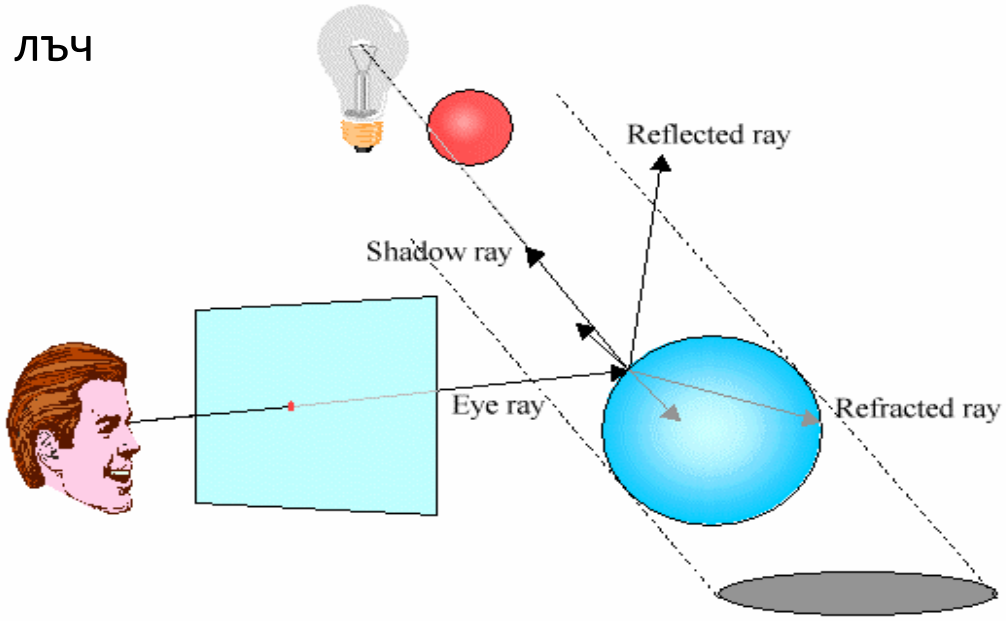
Трасиране на лъчи

- В тази имплементация са включени
  - множество сфери (с различни характеристики)
  - множество нива на рекурсивност
    - отражение
  - прозрачност
    - пречупване
  - един точков източник на светлина
    - твърди сенки
  - премахване на скрити повърхности
  - модел на осветеност на Фонг

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,7.,0.,0.,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx;double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u,tmin=u>=1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){puts("P3\n32 32\n255");while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*minray!*/
```

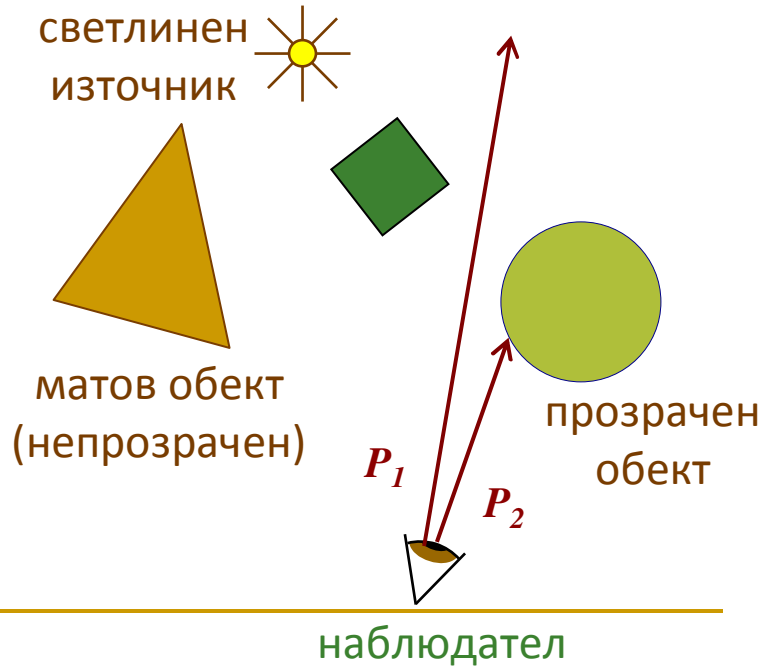
Видове лъчи

- ***Основни лъчи***
  - от точката на наблюдение през проекционната равнина към сцената
- ***Вторични лъчи***
  - от точката на пресичане на лъч и обект
    - преминаващ (пречупен) лъч
    - отразен лъч
    - лъч за сянка



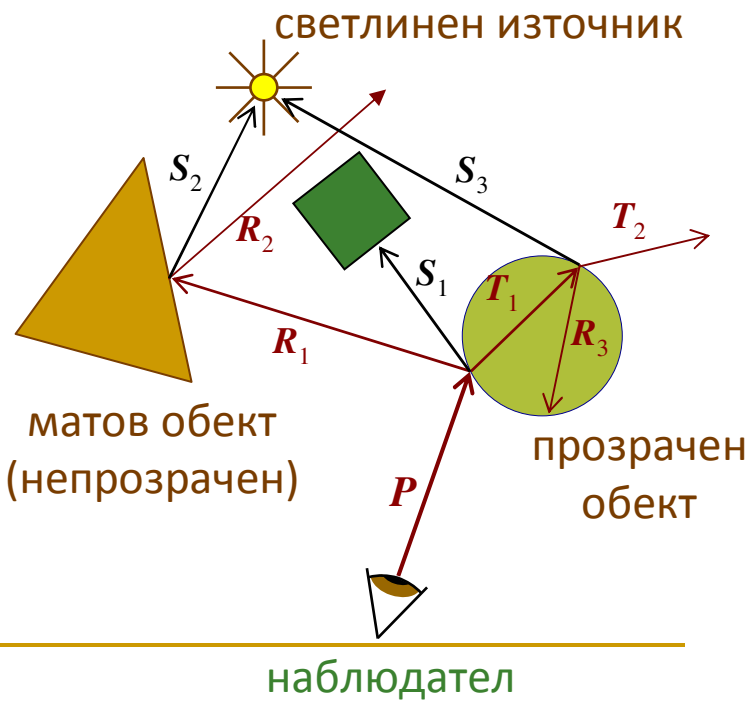
Видове лъчи

- ***Основни лъчи***
  - от точката на наблюдение през проекционната равнина към сцената
  - може да пресича или да не пресича обект в сцената
    - ***не пресича обекта***
      - пикселът има цвета на фона
    - ***пресича обекта***
      - генерират се вторични лъчи
      - изчислява се интензитет/цвят по модела на осветеност



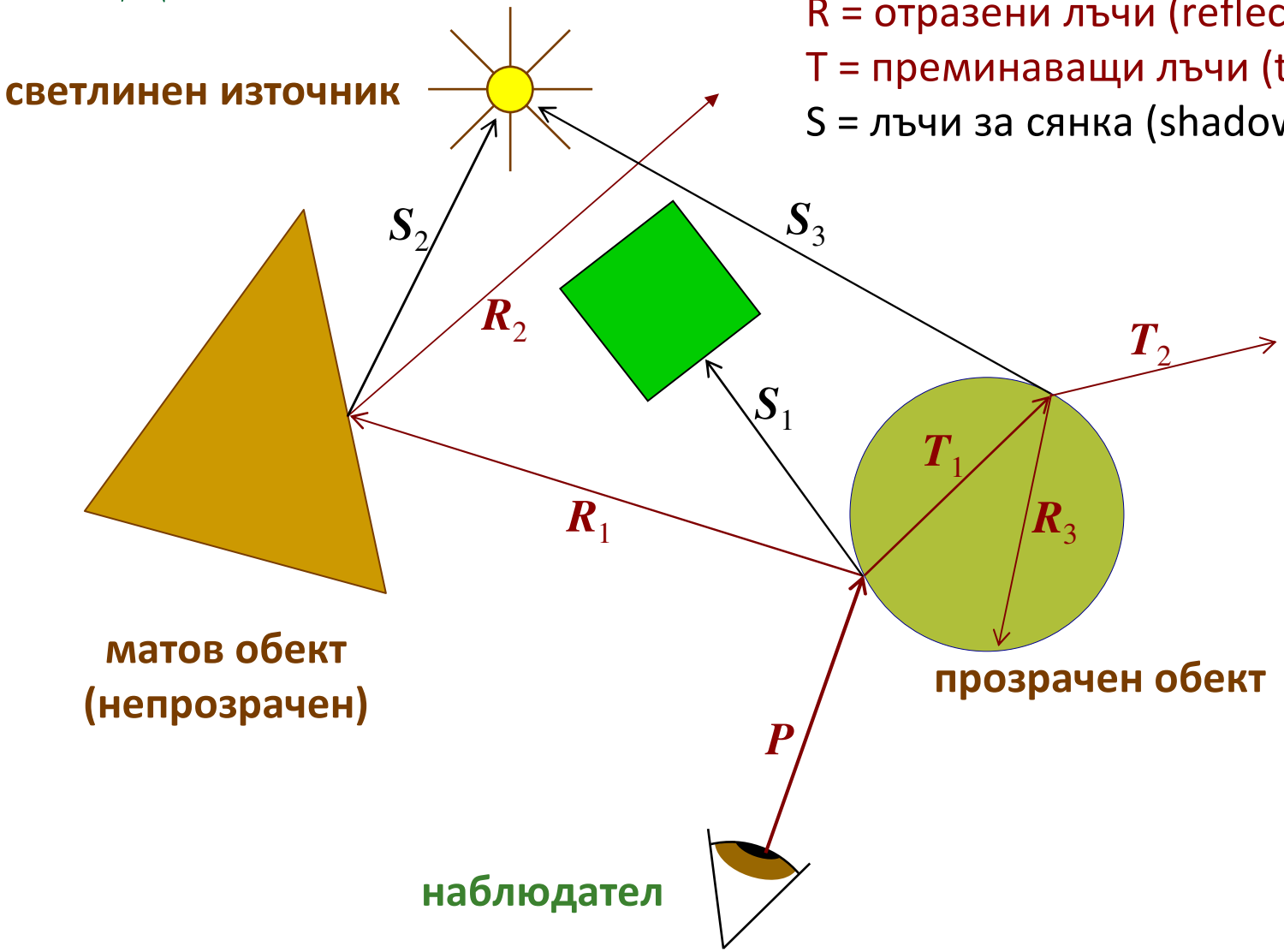
Видове лъчи

- ***Вторични лъчи***
  - от точката на пресичане на лъч и обект
  - ***преминаващ (пречупен) лъч*** ***(transmission – T)***
    - в посоката на пречупване
  - ***отразен лъч*** ***(reflection – R)***
    - в посоката на отражение
      - използва се в модела на Фонг
  - ***лъч на сянка*** ***(shadow – S)***
    - в посока на източника на светлина
      - за определяне дали точката е в сянка или не



Видове лъчи

P = основни лъчи (primary)
R = отразени лъчи (reflected)
T = преминаващи лъчи (transmitted)
S = лъчи за сянка (shadow)

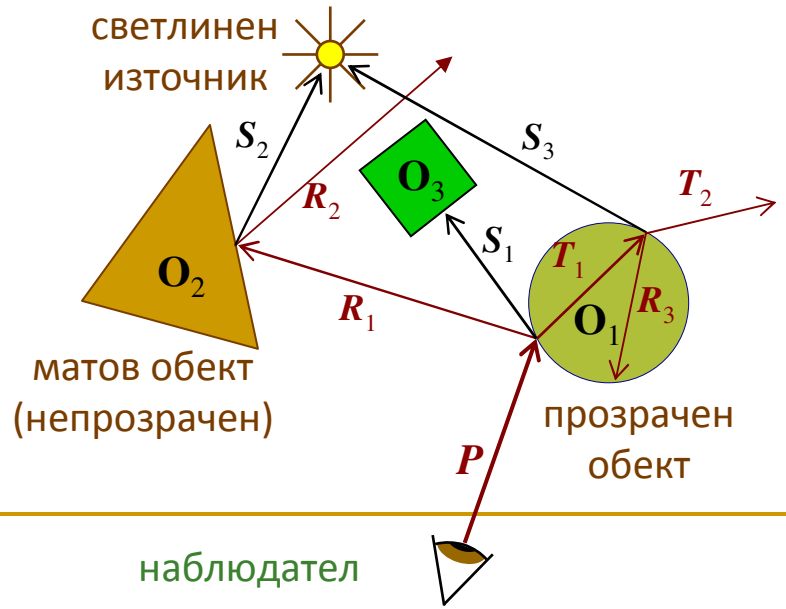


Дърво на лъчите

- ***Ray Tree***
  - всяко пресичане на лъч с обект може да генерира вторични лъчи
- Генерираните лъчи формират дърво на лъчите
  - възлите са пресечните точки
  - дъгите са вторичните лъчи
  - лъчите за сянка се генерират от всеки възел в дървото, но не пораждат други вторични лъчи
- Левите клони на дървото представят пътя на отражение на лъчите
- Десните клони на дървото представят пътя на преминаващите лъчи

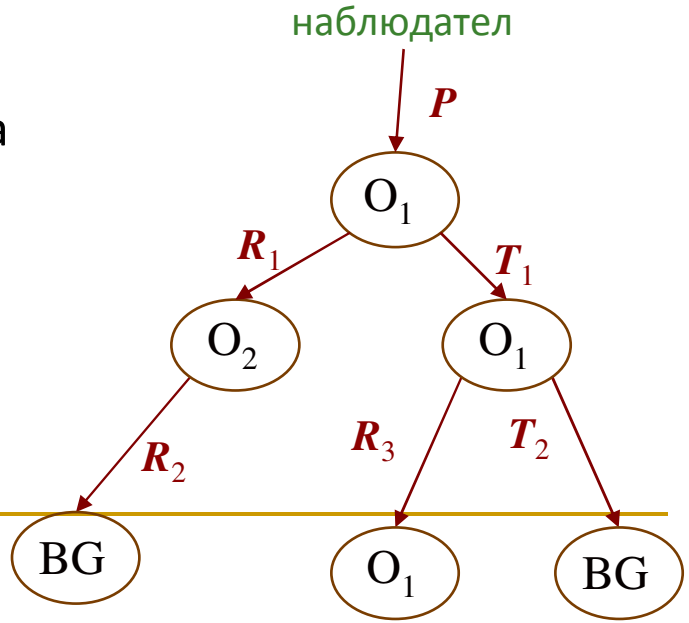
Дърво на лъчите

- Лъчите се генерират рекурсивно докато
  - лъчите не пресичат никакви обекти
  - дървото има максимална дълбочина
  - интензитетът на светлината има минимална стойност



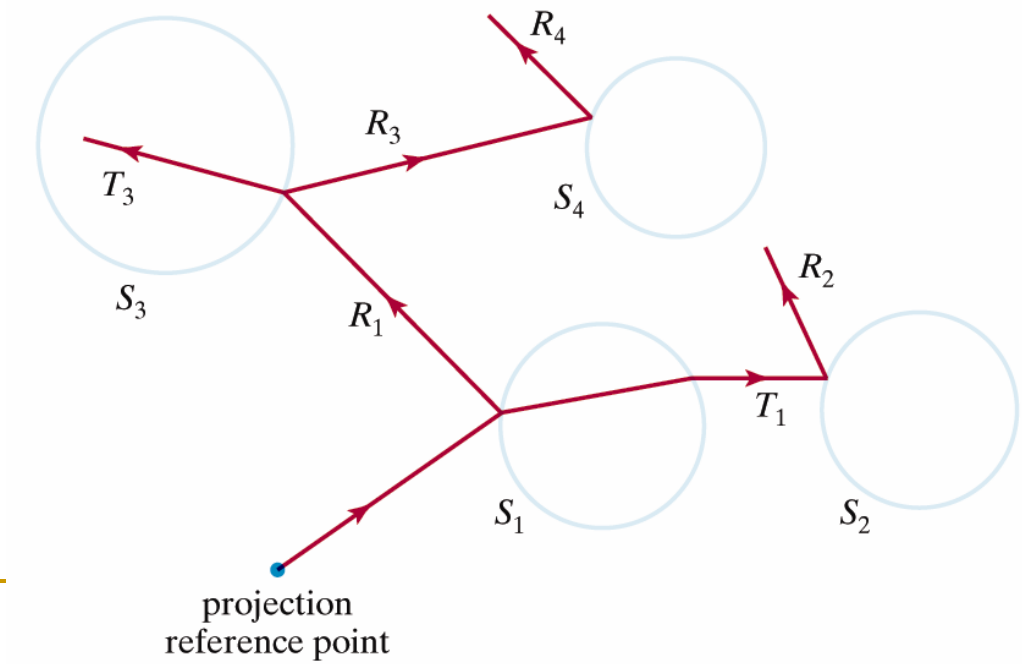
Дърво на лъчите

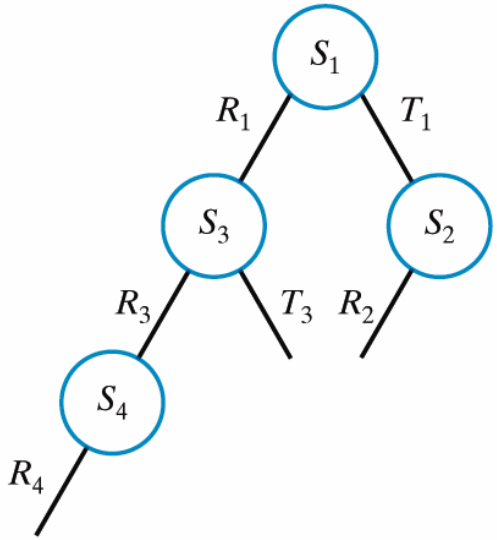
- Дървото на лъчите се използва за да се определи цвета на пиксел на базата на всички участващи при формирането му източници
- След изграждане на дървото на лъчите за даден пиксел се сумират интензитети, които определят цвета на пиксела
  - дървото се анализира отдолу нагоре
  - цветът на възел-родител се изчислява на базата на цветовете на възлите-деца



Дърво на лъчите

- За дървото на лъчите на даден пиксел
  - започва се от възел-лист
  - интензитета във всеки възел се намалява с разстоянието до повърхността на възела-родител и се добавя към интензитета на възела-родител
  - сумата на интензитетите в корена се определя като цвят на пиксела





Основен алгоритъм RayTracing

- Генерира се по един лъч за всеки пиксел
- За всеки лъч
  - определя се първият обект, които лъчът пресича
  - изчислява се цвета на пресечната точка с използване на модел на осветеност
  - ако повърхността е рефлективна се трасира отразен лъч
  - ако повърхността е прозрачна се трасира преминаващ лъч
  - трасира се лъч за определяне дали точката е в сянка
  - комбинират се резултатите за изчисления интензитет, отражение, пречупване и сянка
  - ако лъчът не се пресича с нито един обект се задава цвета на фона

Основен алгоритъм RayTracing

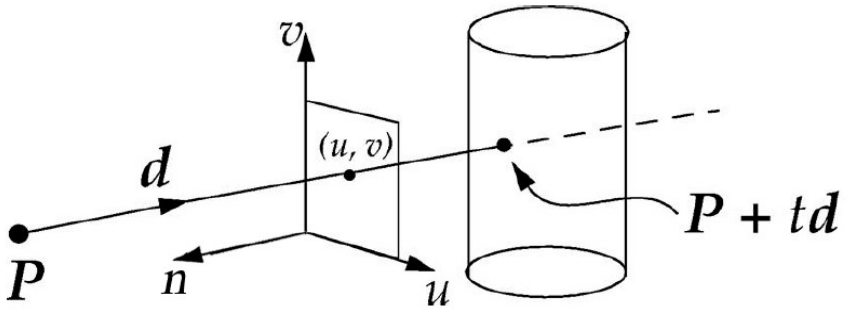
- Основен (*нерекурсивен*) алгоритъм
  1. Генерира се лъч от позицията на наблюдение през екрана на визуализиране
  2. Определя се кой е първият обект, които се пресича от лъча
  3. Изчислява се цвета на пиксела в пресечната точка

- По-голяма част от времето за изчисления е за стъпка 2 (около 75%)
  - прост метод
    - сравнява се всеки лъч с всеки обект за определяне на най-близкият обект, до който достига лъча
  - изчислително сложен и времеотнемащ
    - възможни са различни оптимизации

Основен алгоритъм RayTracing

- Основен (*нерекурсивен*) алгоритъм
  1. ***Генерира се лъч от позицията на наблюдение през екрана на визуализиране***
  2. Определя се кой е първият обект, които се пресича от лъча
  3. Изчислява се цвета на пиксела в пресечната точка

Основни лъчи



- ***Начало на основен лъч***
  - Геометрия в не-трансформирана световна координатна система с перспективна проекция
    - начало на лъча
      - позиция на наблюдение – точка P
    - посока на лъча
      - от т.P към точката в проекционната равнина, чиито цвят трябва да се определи – вектор d
    - точките върху проекционния лъч са P + td
      - P е позицията на наблюдение (на камерата)
      - d е единичен вектор в посока на лъч
      - t е неотрицателна реална стойност
- точката на наблюдение е центъра на проекцията в перспективния визуален обем (frustum)
- не се използва “де-перспективизиране” за да не се налага след това използване на обратна перспективна трансформация

Основни лъчи

- ***Представяне на основен лъч***
  - явно представяне (в параметрична форма) чрез начална точка и вектор на посока
  - начало на лъча

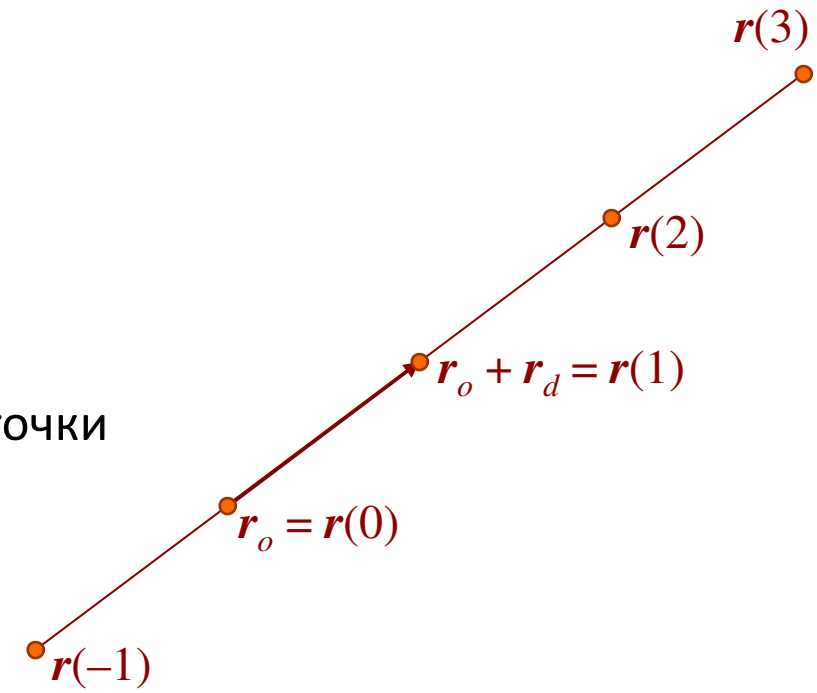
$$\mathbf{r}_o = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix}$$

  - посока на лъча

$$\mathbf{r}_d = \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$$

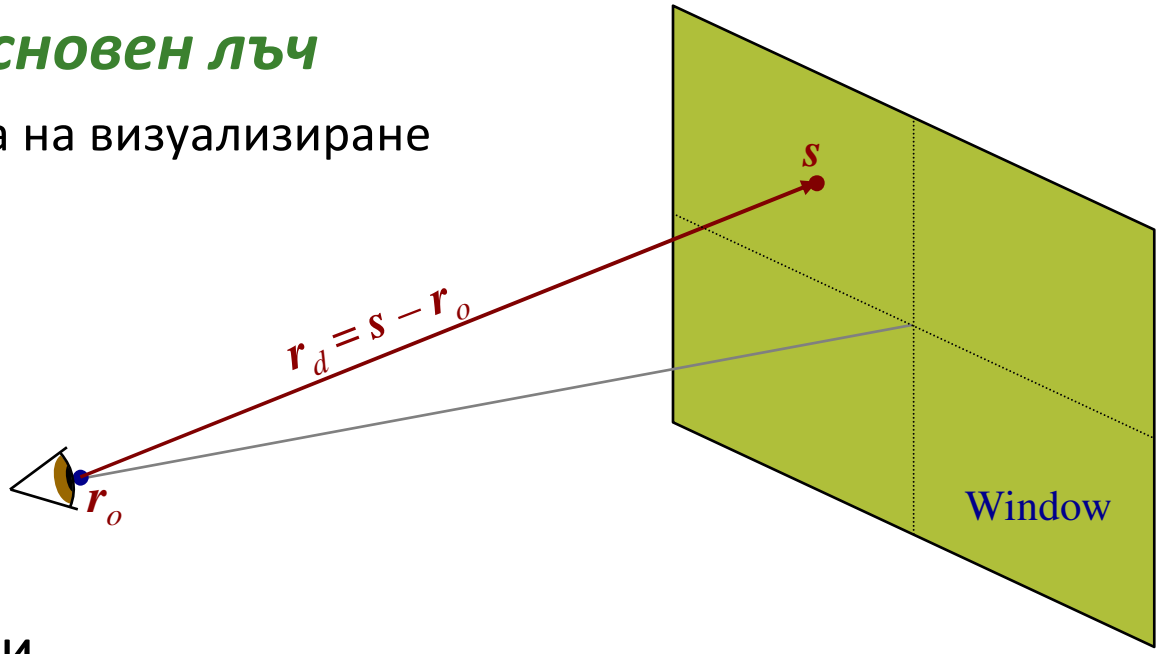
  - Лъчът се състои от безкраен брой точки

$$\mathbf{r}(t) = \mathbf{r}_o + \mathbf{r}_d t$$



Основни лъчи

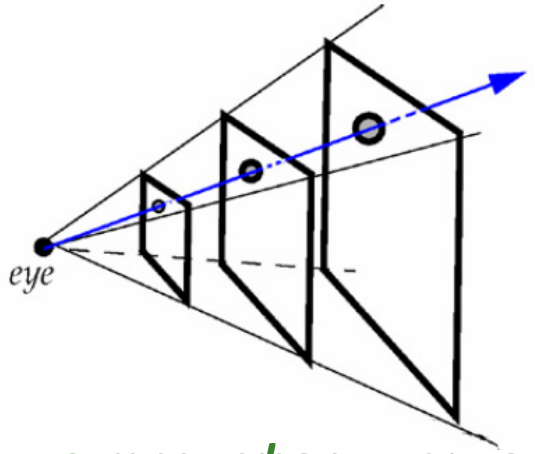
- ***Представяне на основен лъч***
  - за точка s в прозореца на визуализиране основният лъч е
    - начало: r_o
    - посока: $r_d = s - r_o$



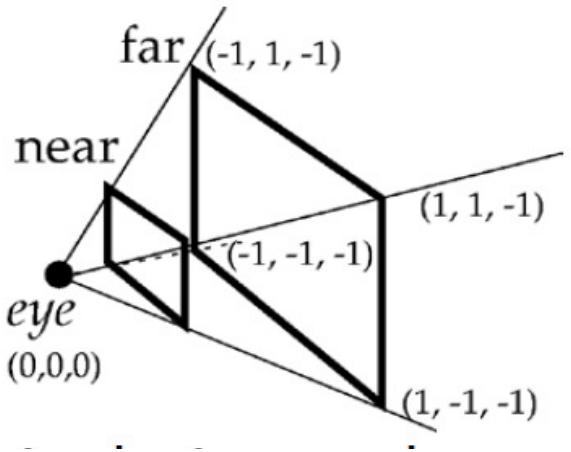
- Координатни системи
  - лъчът трябва да се дефинира в световна координатна система (x, y, z)
  - позицията на наблюдение е зададена със световни координати
  - пикселът от екрана s се определя най-лесно в прозорец с координати на визуализиране (u, v, w)

Основни лъчи

- ***Представяне на основен лъч***
  - за всяка 2D точка (пиксел) в прозореца на визуализиране
    - 2D точката трябва да се конвертира в 3D точка в пространството на проекционната равнина за да се създаде лъч от позицията на наблюдение през тази точка
  - генерираният лъч ще пресича обектите в не-трансформираната световна координатна система



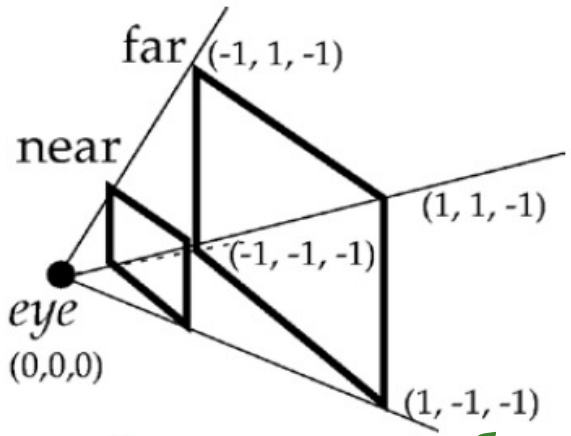
не-трансформирано пространство



каноничен визуален обем

Основни лъчи

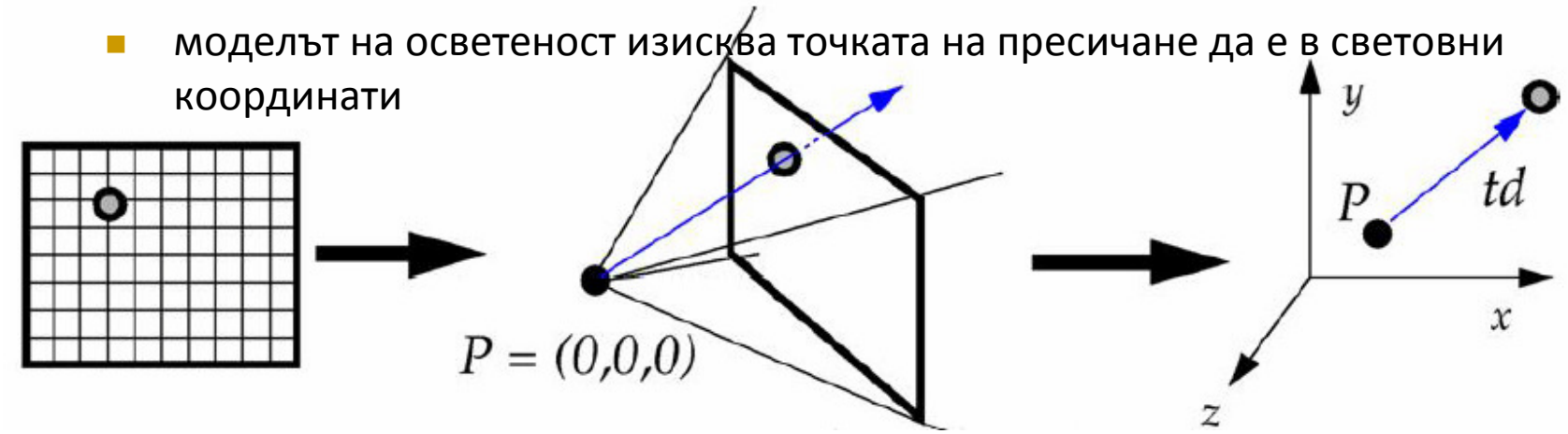
- ***Представяне на основен лъч***
  - Избира се проекционна равнина и се определя функция за преобразуване на екрана за визуализиране в тази равнина
    - всяка равнина z = k, -1<= k < 0, перпендикулярна на вектора на наблюдение може да бъде проекционна равнина
      - например проекционната равнина може да бъде далечната изрязваща равнина (z = - 1)
  - За да се конвертират координатите трябва да се преобразуват целочислените екранни координати в реални стойности между –1 и 1



каноничен визуален обем

Основни лъчи

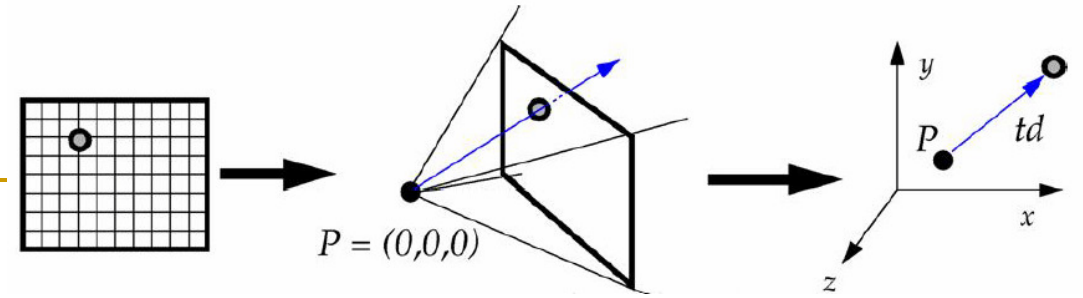
- ***Представяне на основен лъч***
  - След определянето на 3D точка в проекционната равнина, тя трябва да се трансформира в световната координатна система на сцената
    - определя се вектор на посоката на лъча между точката на наблюдение (в центъра на проекцията) и 3D точката в проекционната равнина
    - векторът трябва да бъде в световната координатна система за да се пресича с оригиналния обект в сцената в световна КС
    - моделът на осветеност изисква точката на пресичане да е в световни координати



точка от екрана на визуализиране ***точка в проекционната равнина от каноничния визуален обем*** ***не-трансформиран лъч в световна КС***

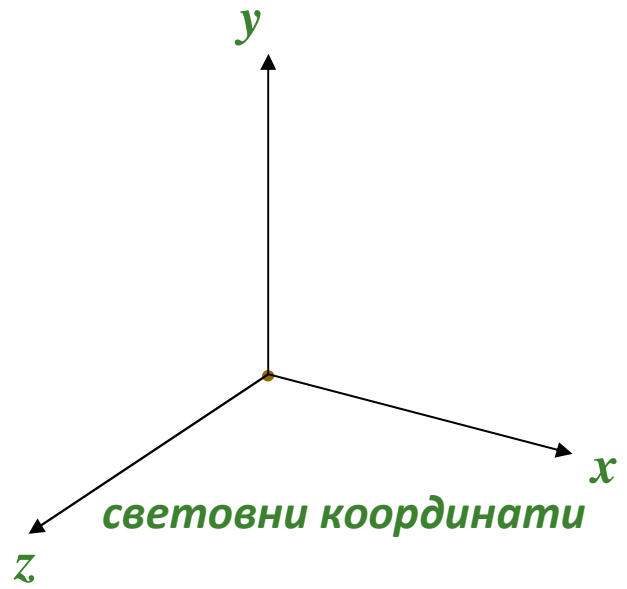
Основни лъчи

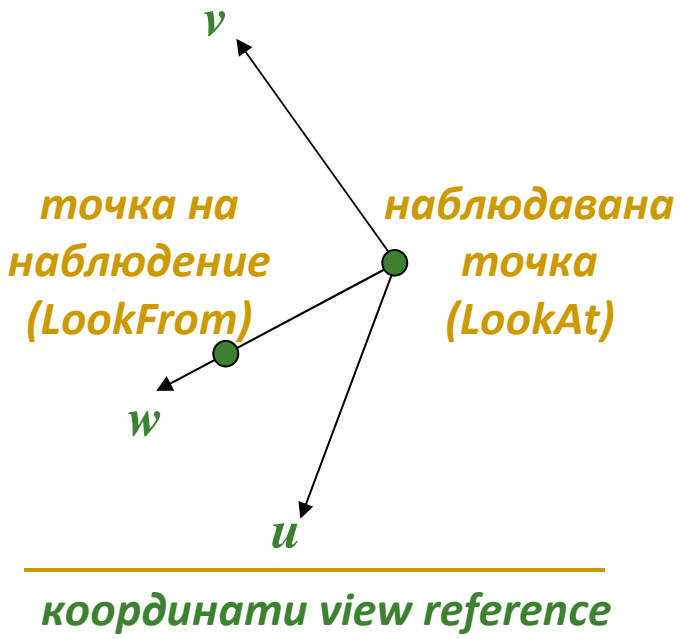
- ***Представяне на основен лъч***
  - Нормализиращата трансформация преобразува точките от световни координати в точки от каноничния визуален обем
    - транслира се началото на КС
    - ротира се така, че вектора на наблюдение *Look* да се ориентира по отрицателната посока на оста Z, а вектора *Up* да е по посока на оста Y
    - мащабира се по x и y за да се станат ъглите на визуализиране 45˚
      - мащабиране по z: [-1, 0]; мащабиране по x, y: [-1, 1]
  - За преобразуване на точка от каноничния визуален обем в не-трансформираната световна КС се прилага обратна на нормализиращата трансформация
    - визуализираща трансформация (**Viewing Transformation**)



Основни лъчи

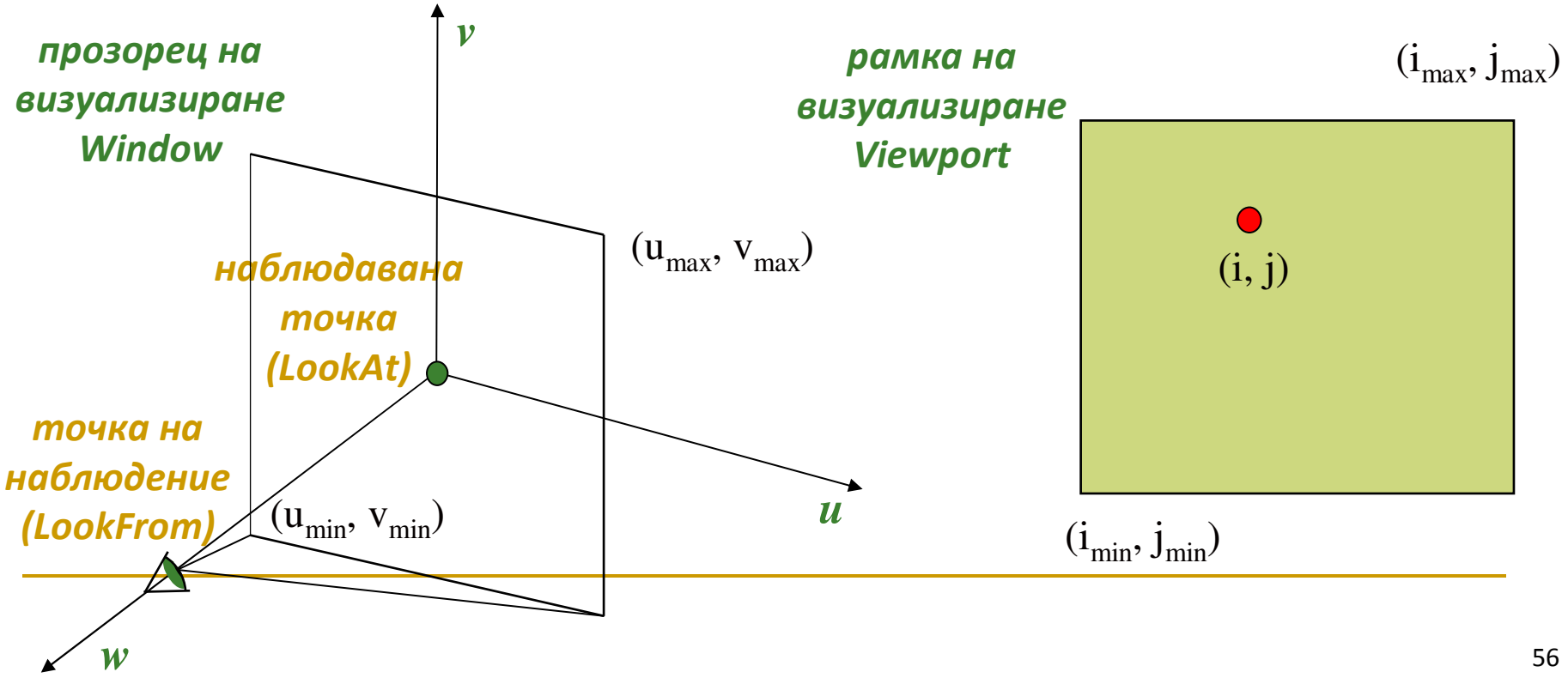
- ***Представяне на основен лъч***
- дадено
  - сцената в световни координати
  - позиция на наблюдение в световни координати (x, y, z)
  - пиксел от рамката на наблюдение с екранни координати (i, j)
- търси се
  - координати на точката в проекционната равнина, която съответства на пиксела с координати (i, j) в рамката на наблюдение (viewport)
  - трансформация на тази точка в световни координати





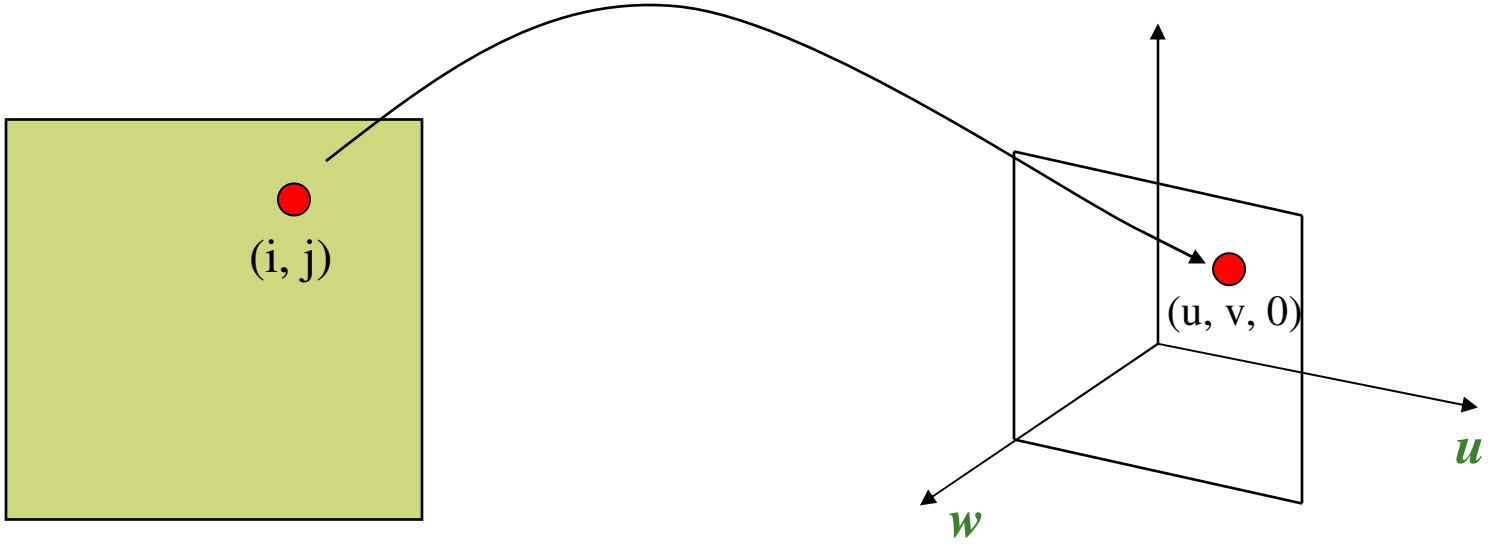
Основни лъчи

- Стъпка 1
  - да се определят координати на точката в проекционната равнина, която съответства на пиксел с координати (i, j) в рамката на наблюдение (viewport)



Основни лъчи

- Стъпка 1
  - да се определят координати на точката в проекционната равнина, която съответства на пиксел с координати (i, j) в рамката на наблюдение (viewport)



Основни лъчи

- Трансформация Window-to-Viewport

$$i = (u - u_{\min})\left(\frac{i_{\max} - i_{\min}}{u_{\max} - u_{\min}}\right) + i_{\min}$$

$$j = (v - v_{\min})\left(\frac{j_{\max} - j_{\min}}{v_{\max} - v_{\min}}\right) + j_{\min}$$

- Обратна трансформация Viewport-to-Window

$$u = (i - i_{\min})\left(\frac{u_{\max} - u_{\min}}{i_{\max} - i_{\min}}\right) + u_{\min}$$

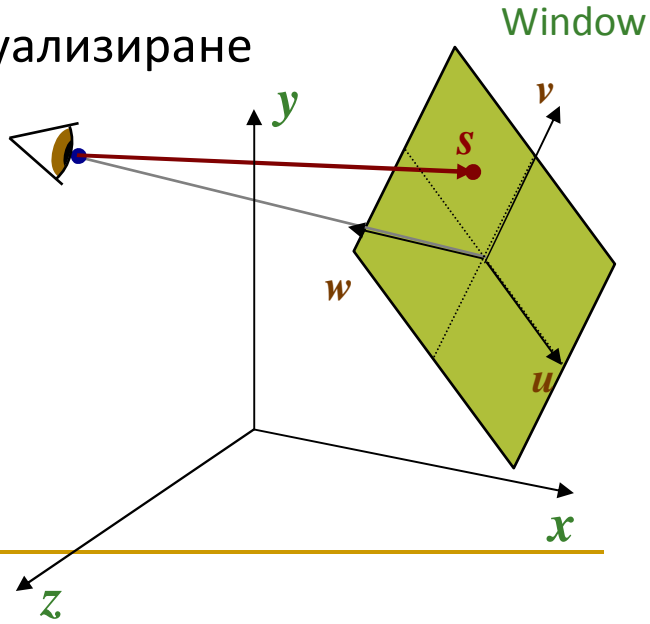
$$v = (j - j_{\min})\left(\frac{v_{\max} - v_{\min}}{j_{\max} - j_{\min}}\right) + v_{\min}$$

$$w = 0$$

Основни лъчи

- Стъпка 2
  - да се трансформира точката от проекционната равнина с координати на визуализиране (*u*, *v*, *w*) в световни координати (*x*, *y*, *z*):
    - трансформация View-reference-to-World
- Визуалната трансформация преобразува точка от световни координати в координати за визуализиране

$$\mathbf{M}_v = \mathbf{M}_{CoB}\mathbf{T} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -LookAt_x \\ 0 & 1 & 0 & -LookAt_y \\ 0 & 0 & 1 & -LookAt_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



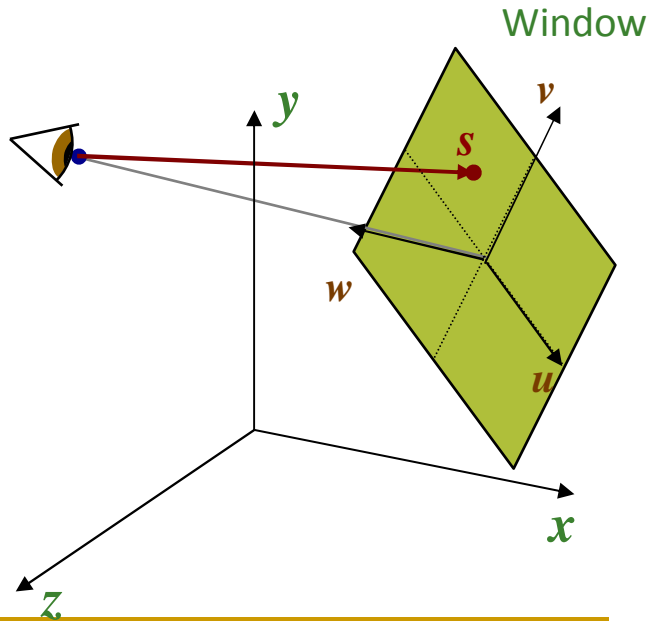
Основни лъчи

- За да се трансформира точката от проекционната равнина с координати на визуализиране (*u*, *v*, *w*) в световни координати (*x*, *y*, *z*) се използва обратна на визуалната трансформация

$$\mathbf{s}_{World} = \mathbf{M}_{v}^{-1}\begin{bmatrix} u_s \\ v_s \\ w_s \\ 1 \end{bmatrix} = \mathbf{T}^{-1}\mathbf{M}_{CoB}^{-1}\begin{bmatrix} u_s \\ v_s \\ w_s \\ 1 \end{bmatrix} =$$

$$\begin{bmatrix} u_x & v_x & w_x & LookAt_x \\ u_y & v_y & w_y & LookAt_y \\ u_z & v_z & w_z & LookAt_z \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} u_s \\ v_s \\ w_s \\ 1 \end{bmatrix}$$

$$\boldsymbol{s}_{World} = \boldsymbol{LookAt} + u_s\boldsymbol{u} + v_s\boldsymbol{v} + w_s\boldsymbol{w}$$



Основен алгоритъм RayTracing

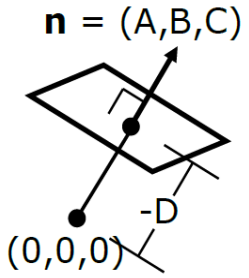
- Основен (*нерекурсивен*) алгоритъм
  1. Генерира се лъч от позицията на наблюдение през екрана на визуализиране
  2. ***Определя се кой е първият обект, които се пресича от лъча***
  3. Изчислява се цвета на пиксела в пресечната точка

Пресичане на лъч и обект

- ***Неявни обекти (Implicit objects)***
  - обект е дефиниран с неявно представяне чрез функция *f* такава, че *f(Q)* = 0
  - ако *Q* е точка от повърхността на обекта, то пресичането на лъча и обекта се определя лесно
    - много обекти могат да бъдат зададени с неявно представяне
    - неявните функции осигуряват практически безкрайна разделителна способност
    - разделянето на такива обекти (tessellation) е по-трудно отколкото при явни функции

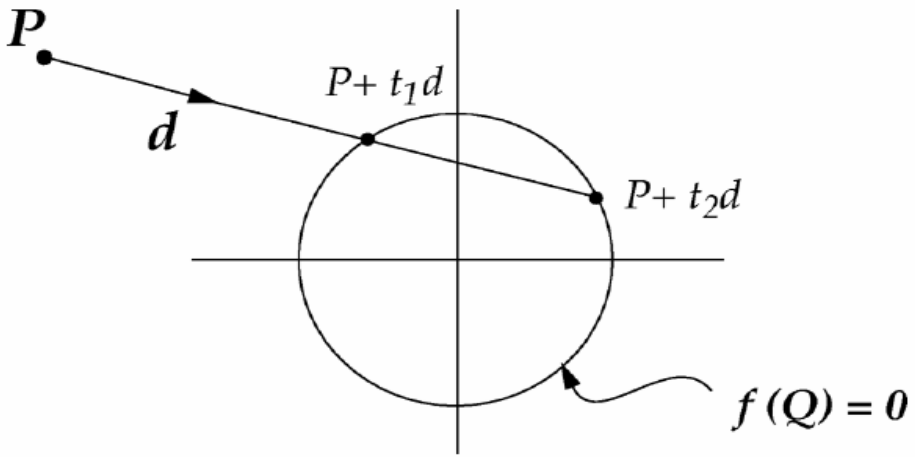
Примери

- Окръжност с радиус *R* е неявно зададен обект в равнина и има уравнение ***f(x,y)*** :
  - точката *(x,y) е върху окръжността когато f(x,y)* = 0
- Равнина се дефинира с неявно представяне чрез ***f(x,y,z) = Ax + By + Cz + D***
- Сфера с радиус *R* се дефинира неявно в 3D пространството чрез ***f(x,y,z) = $x^2 + y^2 + z^2 – R^2$***



Пресичане на лъч и обект

- ***Неявни обекти (Implicit objects)***
  - Определяне на пресечна точка на лъч и обект
    - точките от лъча са $P + t\mathbf{d}$
      - t е неотрицателна реална стойност
    - за точка Q от повърхността на обекта: $f(Q) = 0$
    - трябва да се определи за кои стойности на t: $f(P + t\mathbf{d}) = 0$
      - решава се система уравнение спрямо x, y (в 2D) или спрямо x, y, z (в 3D)



Пресичане на лъч и обект

- ***Пример***
  - ***пресичане на лъч с 2D окръжност***
    - дадени са точка на наблюдение P = (-3, 1), посока на лъча $\boldsymbol{d}$ = (0.8, -0.6) и единична окръжност $f(x,y) = x^2 + y^2 – R^2$
    - точка от лъча

      $Q = P + t\boldsymbol{d} = (-3, 1) + t\ (0.8,-0.6) = (-3 + 0.8t, 1–0\ .6t)$
    - замества се в уравнението на окръжността

      $f(Q) = f(-3 + 0.8t,1 – 0.6t) = (-3+0.8t)^2 + (1–0.6t)^2 – 1$
    - развива се: $9 – 4.8t + 0.64t2 + 1 – 1.2t + 0.36t2 - 1$
    - приравнява се на нула

      $t^2 – 6t + 9 = 0$
    - решава се квадратното уравнение

      $t_1 = t_2 = 3.3$
    - уравнението има двоен корен $\Rightarrow$ лъчът е тангента на окръжността

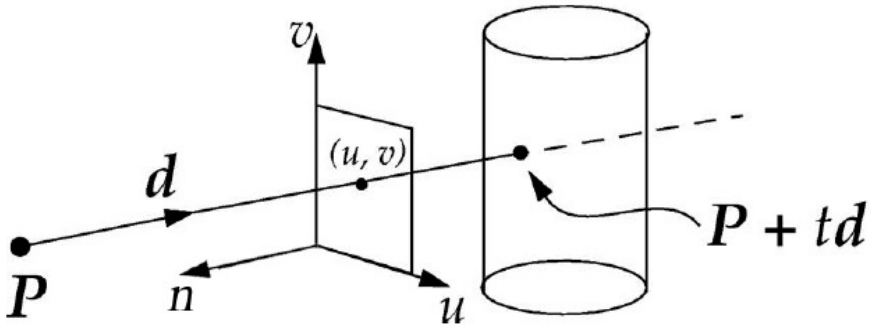
Пресичане на лъч и обект

- ***Пример***
  - ***пресичане на лъч с 2D окръжност***
    - по знака на дискриминантата D на квадратното уравнение може да се определи дали лъча пресича окръжността
      - ***ако D < 0***
        - лъчът не пресича окръжността (имагинерни корени)
      - ***ако D = 0***
        - лъчът пресича окръжността в една точка (един двоен корен, тангентен лъч)
      - ***ако D > 0***
        - лъчът пресича окръжността в две точки (два реални корена)
    - най-малката неотрицателна реална стойност на t представлява най-близката до позицията на наблюдение пресечна точка

Пресичане на лъч и обект

- ***Обобщение – пресичане на лъч с произволна повърхнина***
  - произволна повърхнина зададена с неявно представяне $f(Q) = 0$
    - точки от лъча $P + t\mathbf{d}$
  - замества се в представянето на повърхнината

    $f(P + t\mathbf{d}) = 0$
  - след еквивалентни преобразувания се свежда до уравнение спрямо неизвестната стойност t
  - уравнението се решава спрямо t
    - аналитично или числено



Пресичане на лъч и обект

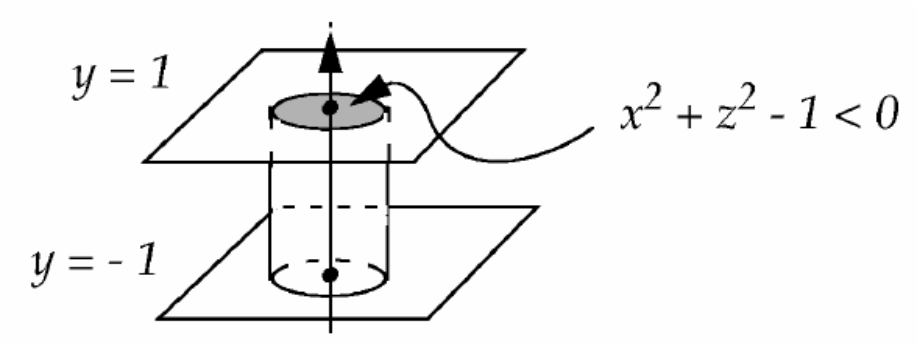
- ***Неявни обекти – множество условия***

  - Неявно представяне на безкраен цилиндър с радиус 1, ориентиран по оста y

    $f(x, y, z) = x^2 + z^2 - 1 = 0$

  - Краен цилиндър

    $x^2 + z^2 - 1 = 0,\ -1 \le y \le 1$



  - Горна стена на цилиндъра

    $x^2 + z^2 - 1 \le 0,\ y = 1$

  - Долна стена на цилиндъра

    $x^2 + z^2 - 1 \le 0,\ y = -1$

Пресичане на лъч и обект

- ***Неявни обекти – множество условия***

```
Ray_inter_finite_cylinder(P,d):
   t1,t2 = ray_inter_infinite_cylinder(P,d)
              // Check for intersection with infinite cylinder
   compute P + t1*d, P + t2*d
              // If there is an intersection, is it between “end caps”?
   if y > 1 or y < -1 for t1 or t2, toss it
   Compute ray_inter_plane(t3, plane y = 1)
              // Check for an intersection with the top end cap
   Compute P + t3*d
   if x2 + z2 > 1, toss out t3
              // If it intersects, is it within cap circle?
   Compute ray_inter_plane(t4, plane y = -1)
              // Check intersection with other end cap
   Compute P + t4*d
   if x2 + z2 > 1, toss out t4
              // If it intersects, is it within cap circle?
```

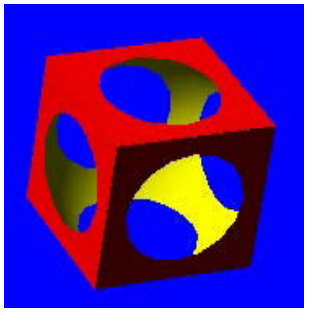
- от всички останали t1, t2, t3, t4 се избира най-малката не-отрицателна стойност

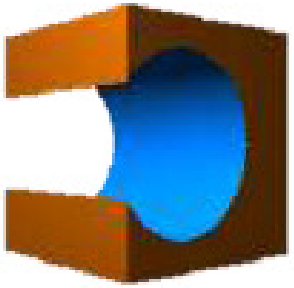
Пресичане на лъч и обект

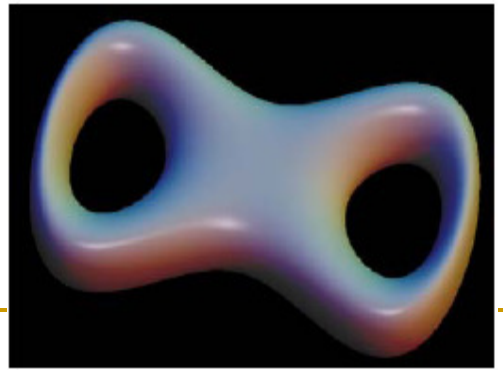
- ***Обобщение – пресичане на лъч и обект***
  - Обектът е зададен като повърхнина с неявно представяне
  - Замества се параметричното представяне на лъча (**P** + *t****d***) в уравнението на повърхнината и се решава спрямо *t*
    - най-малката неотрицателна стойност на *t* е най-близката до точката на наблюдение повърхнина от обект
  - За сложни обекти, които не се задават с единствено уравнение, се определят множество от равенства и неравенства и след това се анализират отделните повърхности

Пресичане на лъч и обект

- ***Обобщение – пресичане на лъч и обект***
  - Обобщението може да се направи за всички различни сложни комбинации от обекти
    - Конструктивна геометрия на твърди тела (Constructive Solid Geometry – CSG)
      - обектите се съхраняват като йерархични представяния на примитиви и тримерни теоретико-множествени операции
    - обекти, зададени чрез сума на уравнения за неявно представяне на криви и повърхнини
      - “blobby objects”







$F(x,y,z) = ((x^2*(1-x^2)-y^2)^2+0.5*z^2-f*(1+b*(x^2+y^2+z^2)) = 0$

Пресичане в световна КС

- За да се изчисли осветеност трябва да се определи пресичане на лъча с обектите в световна координатна система
  - необходимо е аналитично представяне на всеки обект в световна КС
- Пример
  - единична сфера, мащабирана с 2 по x и транслирана в (3, 4, 5) има уравнение

$$f(x,y,z) = \frac{(x-3)^2}{2^2} + (y-4)^2 + (z-5)^2 = 0.5^2$$

- Замества се параметричното представяне на лъча $P+t\boldsymbol{d}$ в f

$$f(P + t\boldsymbol{d}) = 0$$

- Решава се спрямо t
- Получената стойност за t се използва за не-трансформирания обект
  - единична сфера с център в началото на координатната система
- След това се използва трансформираната версия на обекта
  - не е лесно в общия случай за произволни трансформации
  - трансформираната версия на уравненията е по-сложна и изисква повече изчисления
- Вместо това може да се работи с обекта в неговата координатна система

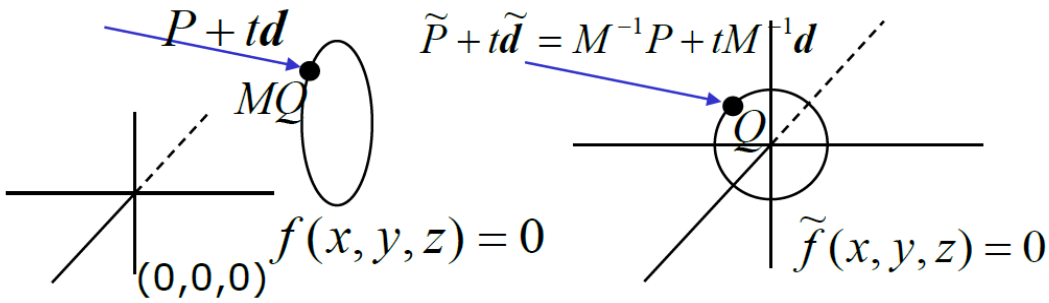
Пресичане в пространство на обекта

- Трансформиране на лъча в пространството на обекта
- *Q* е точка в пространството на обекта
- *MQ* е пресечната точка в световната координатна система

$$P + t\mathbf{d} = MQ$$

$$M^{-1}.(P + t\mathbf{d}) = Q$$

$$M^{-1}P + tM^{-1}\mathbf{d} = Q$$



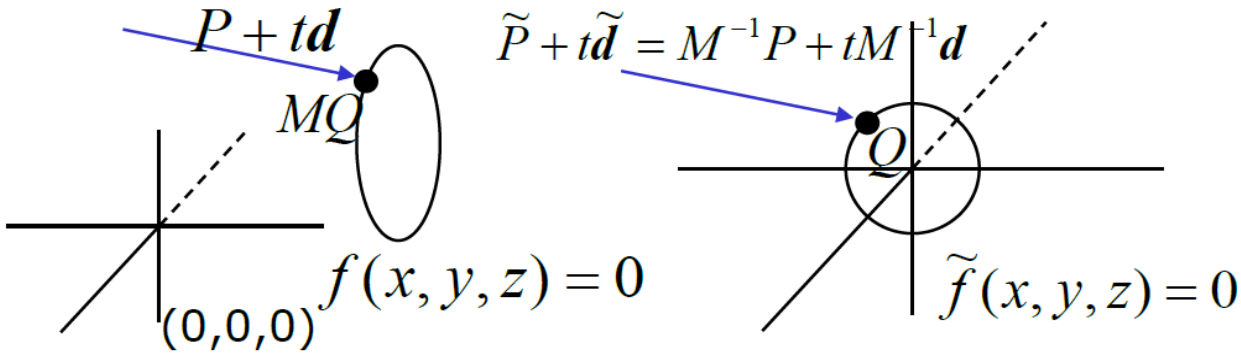
- Ако $\tilde{P} = M^{-1}P$ и $\tilde{d} = M^{-1}\mathbf{d}$
  - $\tilde{f}(x,y,z)$ е уравнението на не-трансформирания обект,
- то трябва да се реши $\tilde{f}(\tilde{P} + t\tilde{\mathbf{d}}) = 0$
  - $\tilde{d}$ обикновено не е единичен вектор
  - параметъра *t* има една и съща стойност за световната КС, и за пространството на обекта
  - $\tilde{d}$ не се нормализира за да не се промени това отношение

Пресичане на лъч и обект

- За изчисляване на пресичания в ***световната КС*** трябва да се трансформира неявното представяне на обектите
  - често е ***сложна*** задача
- За изчисляване на пресичания в ***пространството на обекта*** трябва да се приложи трансформация с матрица M^{-1} за P *и* $\mathbf{d}$
  - много ***по-проста*** задача
- Дали съществува обратна матрица M^{-1}?
  - матрицата M се състои от две части
    - кумулативни моделиращи трансформации, които позиционират обекта в световната КС
      - транслации, ротации, мащабирания – всичките се инвертират
    - нормализираща трансформация на визуализиране при проектиране
      - транслации, ротации и мащабирания, които се инвертират
      - трансформация за перспективна проекция, която ***не*** се инвертира

Пресичане на лъч и обект

- След определянето на стойността на *t* се използва по два начина
  - $P + t\boldsymbol{d}$ е позицията на пресичането на лъча с трансформирания обект в световна координатна система
  - $\tilde{P} + t\tilde{\boldsymbol{d}}$ е съответната точка от не-трансформирания обект в пространството на обекта



Основен алгоритъм RayTracing

- Основен (*нерекурсивен*) алгоритъм
  1. Генерира се лъч от позицията на наблюдение през екрана на визуализиране
  2. Определя се кой е първият обект, които се пресича от лъча
  3. ***Изчислява се цвета на пиксела в пресечната точка***

Нормални вектори

- За определяне на осветеността се изисква нормалата в точката на пресичане в световна КС
- За да се определи нормалата към повърхнината
  - определя се нормалата в пресечната точка в пространството на обекта
  - трансформира се нормалата в световни координати

- ***Стъпка 1: Определяне на нормалата в пресечната точка***
  - Ако повърхнината огражда твърдо тяло зададено с $f(x, y, z) < 0$
  - то нормален вектор в точка (*x*, *y*, *z*) се определя чрез *градиента* в тази точка $n = \nabla f(x, y, z)$

$$\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

  - Градиентът е вектор с компоненти частните производни на функцията

Нормални вектори

- Пример
  - Уравнение на сфера

$$f(x,y,z) = x^2 + y^2 + z^2 - 1$$

  - Частните производни

$$\frac{\partial f}{\partial x} = 2x, \qquad \frac{\partial f}{\partial y} = 2y, \qquad \frac{\partial f}{\partial z} = 2z$$

  - Градиентът е

$$n = \nabla f(x, y, z) = (2x, 2y, 2z)$$

  - Нормализира се нормалният вектор ***n*** преди да се използва в скаларно произведение

- В някои случаи градиентът може да е нула – тогава се използва най-близкия градиент в съседна точка

Нормални вектори

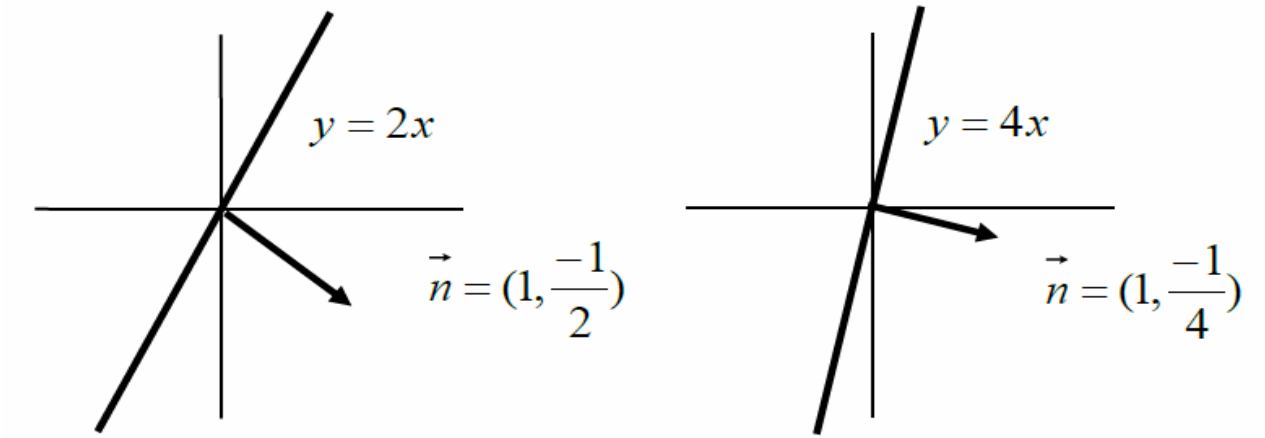
- ***Стъпка 2: Трансформиране на нормалата в световни координати***
  - Нормалният вектор ***не*** може да се трансформира с умножение с трансформиращата матрица

$$n_{world} \neq \mathrm{M}.n_{object}$$

  - Пример
    - M мащабира с коефициент 0.5 по x и 2 по y

Нормални вектори

- Транслацията и ротацията не променят нормалата
  - “rigid body” трансформации
- Мащабирането променя нормалата противоположно на промяната на повърхността на обекта
  - мащабиране с коефициент 2 мащабира нормалата с коефициент 0.5



Нормални вектори

- ***Стъпка 2: Трансформиране на нормалата в световни координати***
  - пример: полигон *H* с нормала в пространството на обекта $\boldsymbol{n}_{obj}$, която трябва да се трансформира в нормала $\boldsymbol{n}_{world}$ към трансформираната форма на *H* в световни координати *MH*
    - за всеки вектор $\boldsymbol{v}$ в световни координати, който лежи в равнината на полигона (например един от ръбовете му) нормалата към полигона е перпендикулярна на вектора

$$\boldsymbol{n}_{world} \cdot \boldsymbol{v}_{world} = 0$$

    - $\boldsymbol{v}_{world}$ е трансформирания вектор v_{obj} в пространството на обекта

$$\boldsymbol{n}_{world} \cdot \boldsymbol{M}\boldsymbol{v}_{obj} = 0$$

Нормални вектори

- ***Стъпка 2: Трансформиране на нормалата в световни координати***
  - транслацията не влияе на позицията на вектора

$$\boldsymbol{n}_{world} \cdot \boldsymbol{M}_3 \boldsymbol{v}_{obj} = 0$$

  - където $\boldsymbol{M}_3$ е матрица с размери 3×3 и стойности определени от горния ляв ъгъл на матрицата M (компонентите за ротация и мащабиране)

$$\boldsymbol{M}_3^t \, \boldsymbol{n}_{world} \cdot \boldsymbol{v}_{obj} = 0$$

  - тъй като

$$\boldsymbol{n}_{obj} \cdot \boldsymbol{v}_{obj} = 0$$

  - то $\boldsymbol{M}_3^t \, \boldsymbol{n}_{world} = \boldsymbol{n}_{obj}$ $\quad / \cdot (\boldsymbol{M}_3^t)^{-1}$

$$\boldsymbol{n}_{world} = (\boldsymbol{M}_3^t)^{-1} \boldsymbol{n}_{obj}$$

Нормални вектори

- ***Стъпка 2: Трансформиране на нормалата в световни координати***

$$\boldsymbol{n}_{world} = (\boldsymbol{M}_3^{t})^{-1}\boldsymbol{n}_{obj}$$

  - транспонирането и инвертирането могат да се разменят

$$\boldsymbol{n}_{world} = (\boldsymbol{M}_3^{-1})^{t}\,\boldsymbol{n}_{obj}$$

    - по-лесно се определят инверсните матрици на отделните трансформации, отколкото обратна на общата матрица на трансформацията

$$((RS...)^{-1})^{t} = (...S^{-1}R^{-1})^{t} = (R^{-1})^{t}\,(S^{-1})^{t}\,...)$$

  - при това

$$(R^{-1})^{t} = R$$

$$(S(s_x,s_y,s_z)^{-1})^{t} = S(s_x,s_y,s_z)^{-1} = S(1/s_x,1/s_y,1/s_z)$$

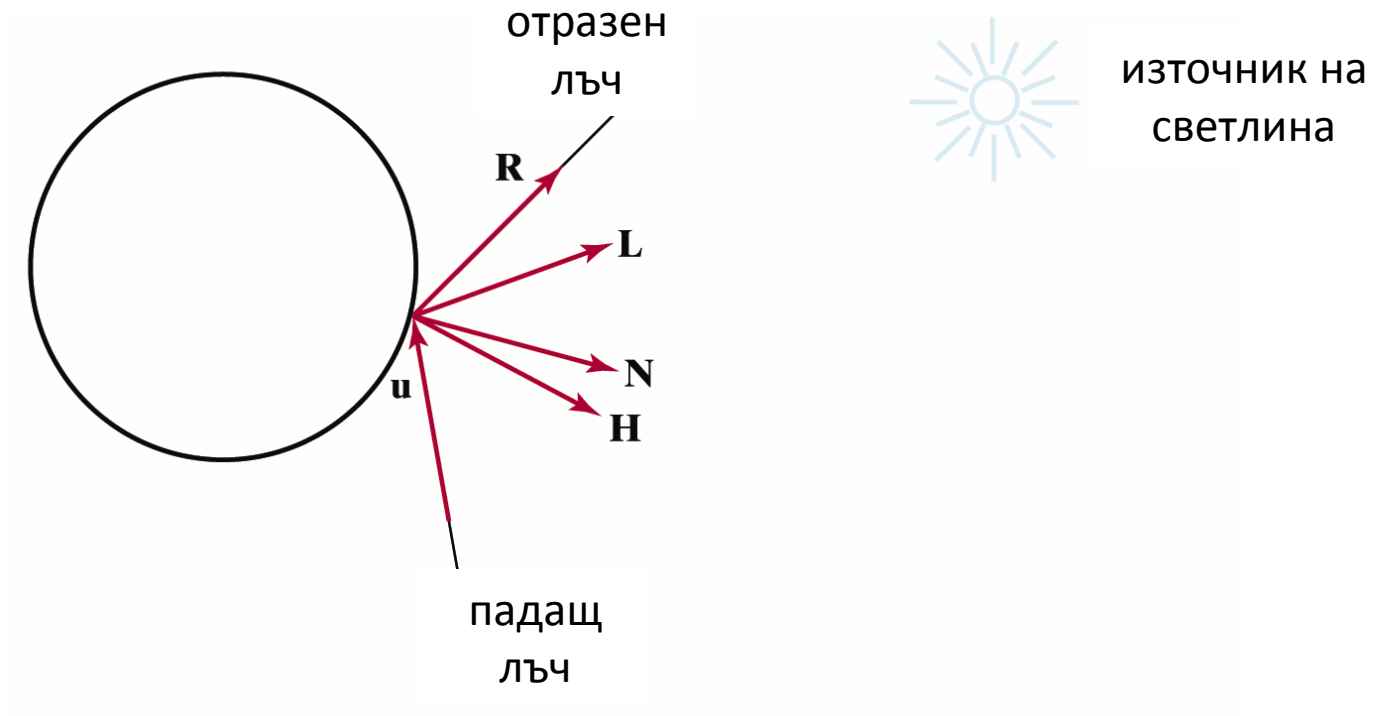
Нерекурсивен алгоритъм

- Прост нерекурсивен алгоритъм за трасиране на лъчи

```
P = eyePt
for each sample of image
   Compute d
   for each object
        Intersect ray P+td with object
        Select object with smallest non-negative t-value
        (visible object)
        For this object, find object space intersection point
        Compute normal at that point
        Transform normal to world space
        Use world space normal for lighting computations
```

Изчисляване на осветеност

- Във всяка пресечна точка се определя осветеност на повърхността съгласно модел на осветеност



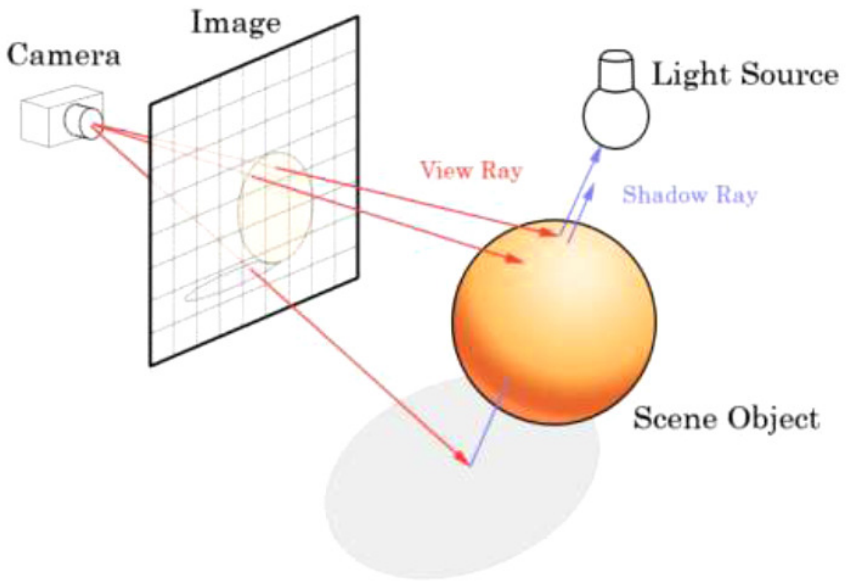
Сянка

- Цветът и интензитетът на елемент от повърхността на обекта се определя от всички светлинни източници в сцената

$$objectIntensity_\lambda = ambient + \sum_{light=1}^{numLights} attenuation \cdot lightIntensity_\lambda \cdot (diffuse + specular)$$

- ***АКО*** светлината достига до обекта!
  - обектът може да е закрит от други обекти в сцената
  - обектът може да се самозакрива

- Ако обект се пресича с лъч за сянка между повърхността и източника на светлина, то повърхността е в сянка спрямо този източник на светлина



Сянка

- За определяне на сянка
  - генерира се лъч за сянка
    - от пресечната точка върху повърхността на обекта към всеки източник на светлина
  - ако източникът на светлина ***е първият*** обект, който лъчът пресича, то за определяне на сянката се използва пълният интензитет на светлинния източник
  - ако първият обект, който пресича лъчът ***не е*** светлинен източник, то се игнорира интензитета на този светлинен източник

- С този метод се определят “твърди сенки” (hard shadows)
  - за определяне на “меки сенки” (soft shadows) са нужни повече изчисления
- При прозрачни или рефлективни (огледални) обекти се получава глобална осветеност
  - използва се рекурсивно трасиране на лъчи

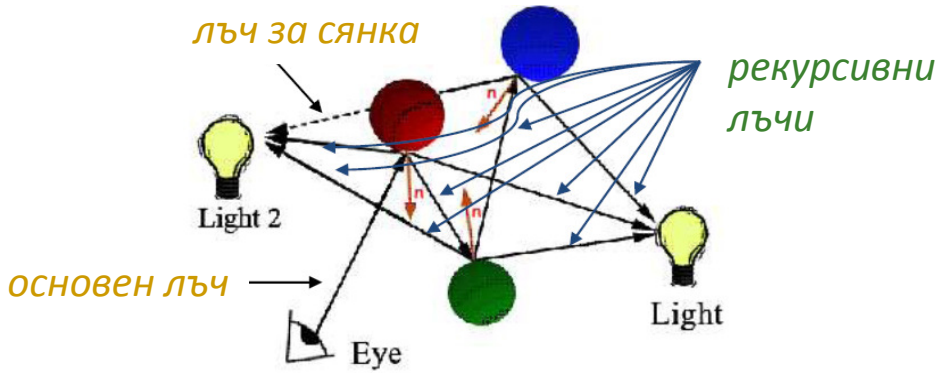
Рекурсивен алгоритъм

- ***Симулиране на ефекти на глобално осветяване (Whitted, 1979)***
- С рекурсивно генериране на нови лъчи се получава повече информация за осветеността в сцената
  - започва се от пресечна точка
  - трябва да се генерират лъчи във всички посоки
    - твърде сложно изчислително и времеотнемащо
  - генерират се лъчи в посоките, които е вероятно да доведат до промяна на осветеността
    - към източниците на светлина
      - пречките на светлината създават сенки за тези източници
    - огледално отражение към другите обекти за определяне на огледалните отношения между обектите
    - използва се обкръжаващата светлина (ambient) за определяне на дифузните (рефлективни) отношения между обектите
    - през обектите за определяне на прозрачност

Рекурсивен алгоритъм

- ***Симулиране на ефекти на глобално осветяване (Whitted, 1979)***
- Трасират се вторичните лъчи в пресечните точки
  - светлинен източник (***light***)
    - трасира се лъч до всеки източник на светлина
    - ако източника е блокиран от непрозрачен обект, то не участва в общата осветеност
  - огледално отражение (***specular reflection***)
    - трасира се отразен лъч (спрямо нормалата в точката на пресичане)
  - пречупване/прозрачност (***refractive transmission/transparency***)
    - трасира се преминаващия лъч (използва се закон на Snell)

- рекурсивно се генерират нови светлинни източници, пречупвания, преминавания във всяка пресечна точка докато интензитета стане пренебрежимо малък или се достигне максимална дълбочина на рекурсията



Рекурсивен алгоритъм

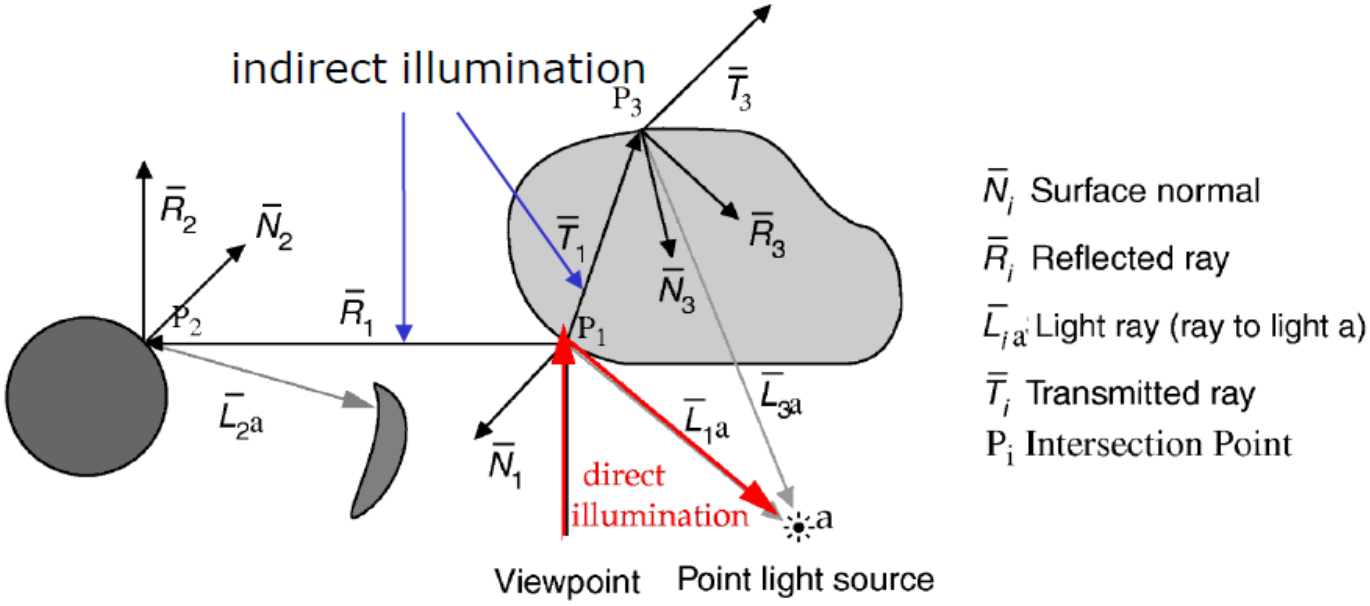
- ***Симулиране на ефекти на глобално осветяване (Whitted, 1979)***
- Модел на осветеност (модел на Фонг)

$$I_\lambda = \underbrace{I_a \lambda_{ka} O_{d\lambda}}_{ambient} + \sum_m f_{att} I_{p\lambda} [\underbrace{k_d O_{d\lambda} \vec{N} \bullet \vec{L}}_{diffuse}] + \underbrace{\underbrace{k_s O_{s\lambda} (\vec{R} \bullet \vec{V})^n}_{specular} + \underbrace{k_s O_{s\lambda}}_{reflected} + \underbrace{k_t O_{t\lambda} I_{t\lambda}}_{transmitted}}_{recursive}$$

  - интензитетите на рекурсивните лъчи се изчисляват с едно и също уравнение
  - източниците на светлина имат огледална и дифузна компонента
- Ограничения
  - рекурсивните отражения между обектите са рефлективни (огледални)
  - дифузните отражения между обектите се обработват отделно

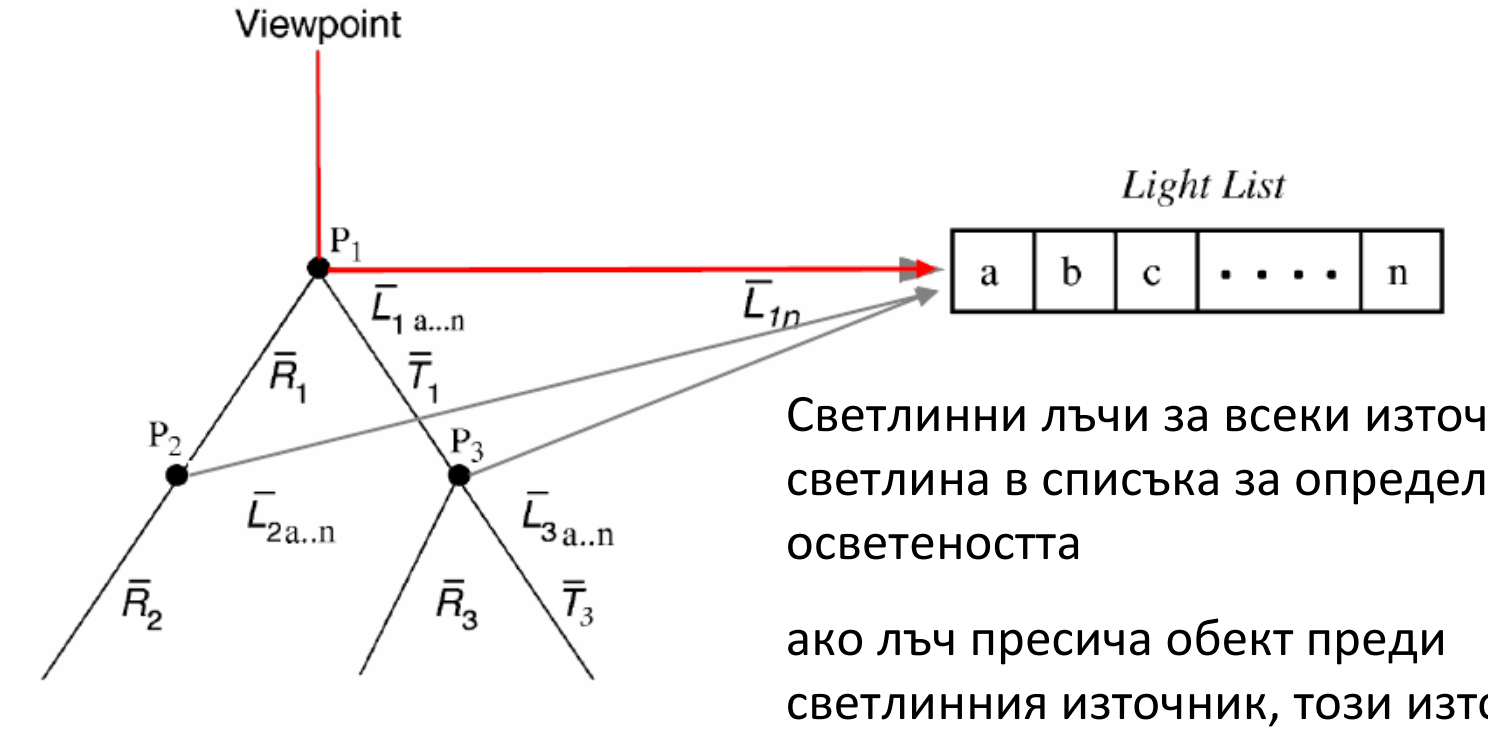
Рекурсивен алгоритъм

- ***Симулиране на ефекти на глобално осветяване (Whitted, 1979)***
- ***Дърво на лъчите***



Рекурсивен алгоритъм

- ***Симулиране на ефекти на глобално осветяване (Whitted, 1979)***
- ***Дърво на лъчите***

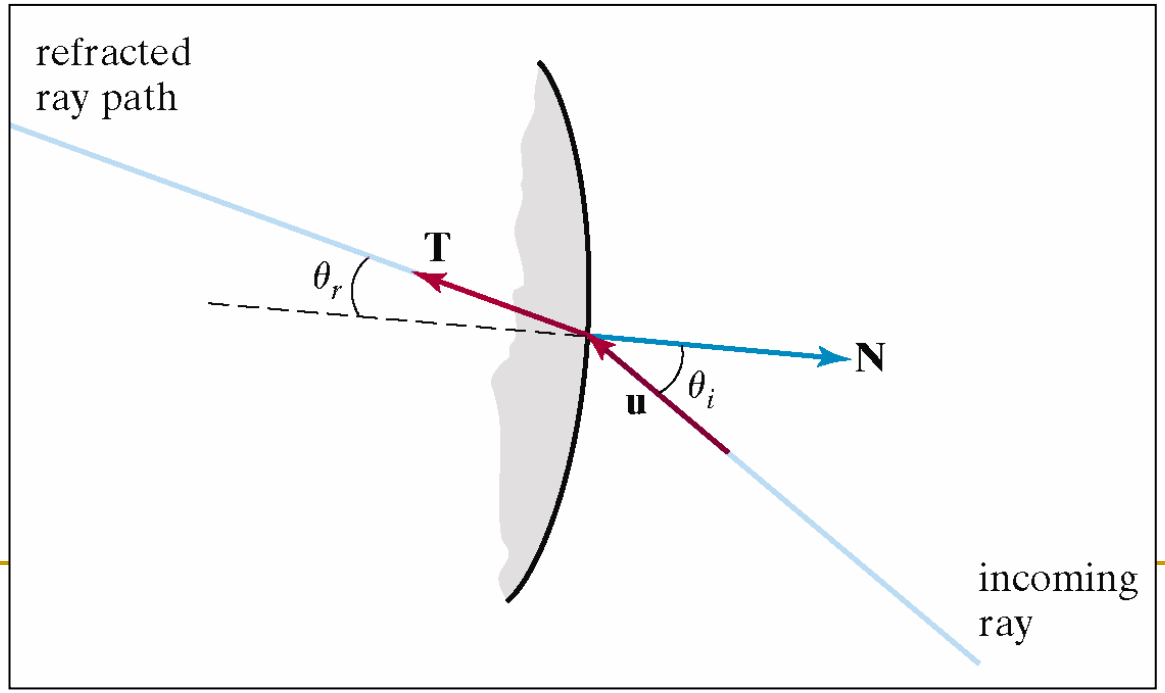


Светлинни лъчи за всеки източник на светлина в списъка за определяне на осветеността

ако лъч пресича обект преди светлинния източник, този източник не се отчита в модела на осветеността

Прозрачни повърхности

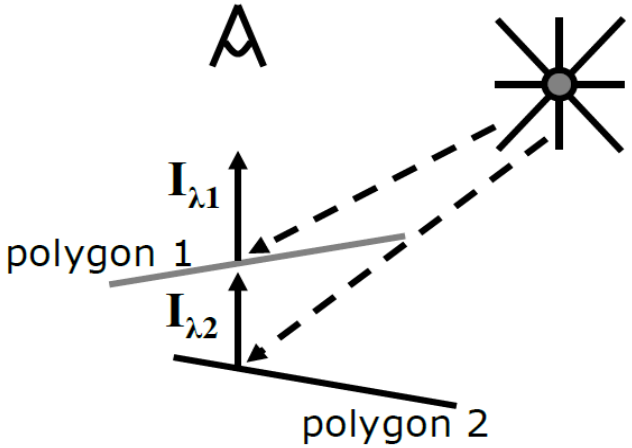
- За прозрачни повърхности се генерира лъч за светлината преминаваща през материала
- Посоката на преминаващия лъч се определя от индекса на пречупване на светлината за материала



Прозрачни повърхности

- ***Прозрачност без пречупване (Non-refractive transparency)***
  - частично прозрачни обекти

$$I_\lambda = (1 - k_{t1}) I_{\lambda 1} + k_{t1} I_{\lambda 2}$$



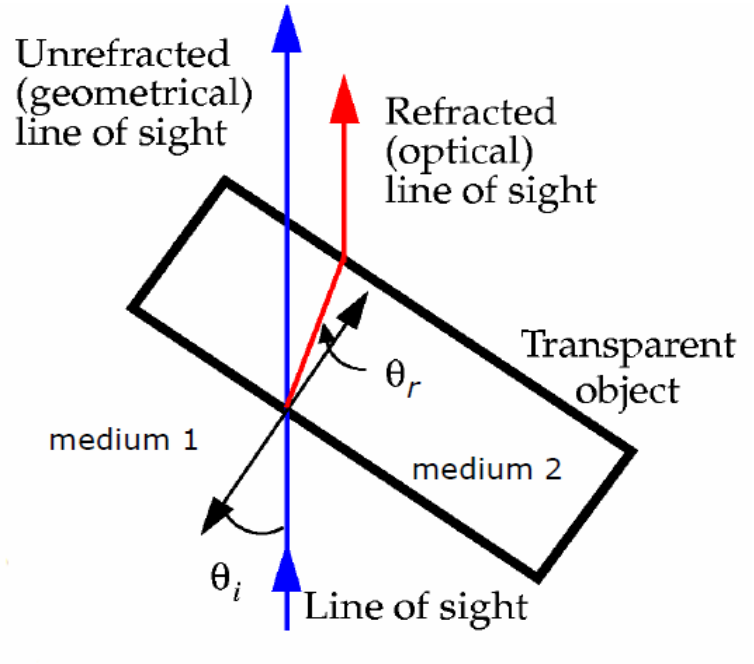
  - k_{t1} – прозрачност на полигон 1
    - 0 – матов, 1 – прозрачен
  - $I_{\lambda 1}$ – интензитет на осветеност за полигон 1
  - $I_{\lambda 1}$ – интензитет на осветеност за полигон 2

Прозрачни повърхности

- ***Прозрачност с пречупване (Refractive transparency)***
  - използва се закон на Snell

$$\sin\theta_r = \frac{\sin\theta_i \eta_{i\lambda}}{\eta_{r\lambda}}$$

  - $n_{i\lambda}$ – индекс на пречупване на материал 1
  - $n_{r\lambda}$ – индекс на пречупване на материал 2

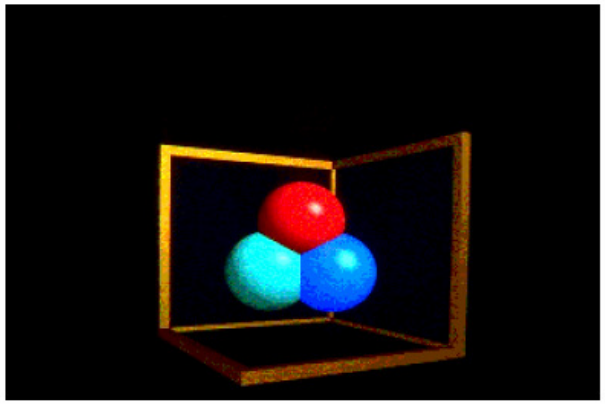


Прозрачни повърхности

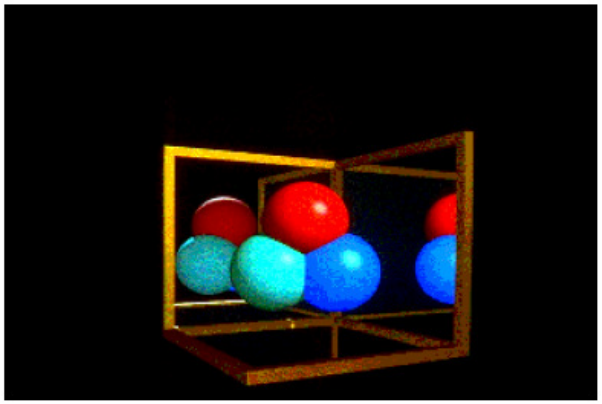
- ***Прозрачност с пречупване (Refractive transparency)***

Material	Index
Vacuum	1.0
Air	1.0003
Water	1.33
Alcohol	1.36
Fused quartz	1.46
Crown glass	1.52
Flint glass	1.65
Sapphire	1.77
Heavy flint glass	1.89
Diamond	2.42

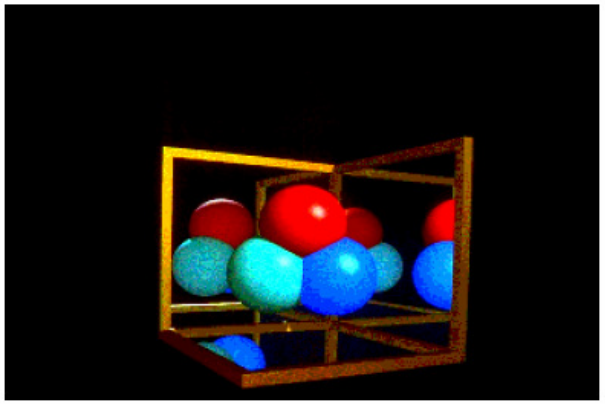
Ray Casting vs. Ray Tracing



Ray Casting -- 1 bounce

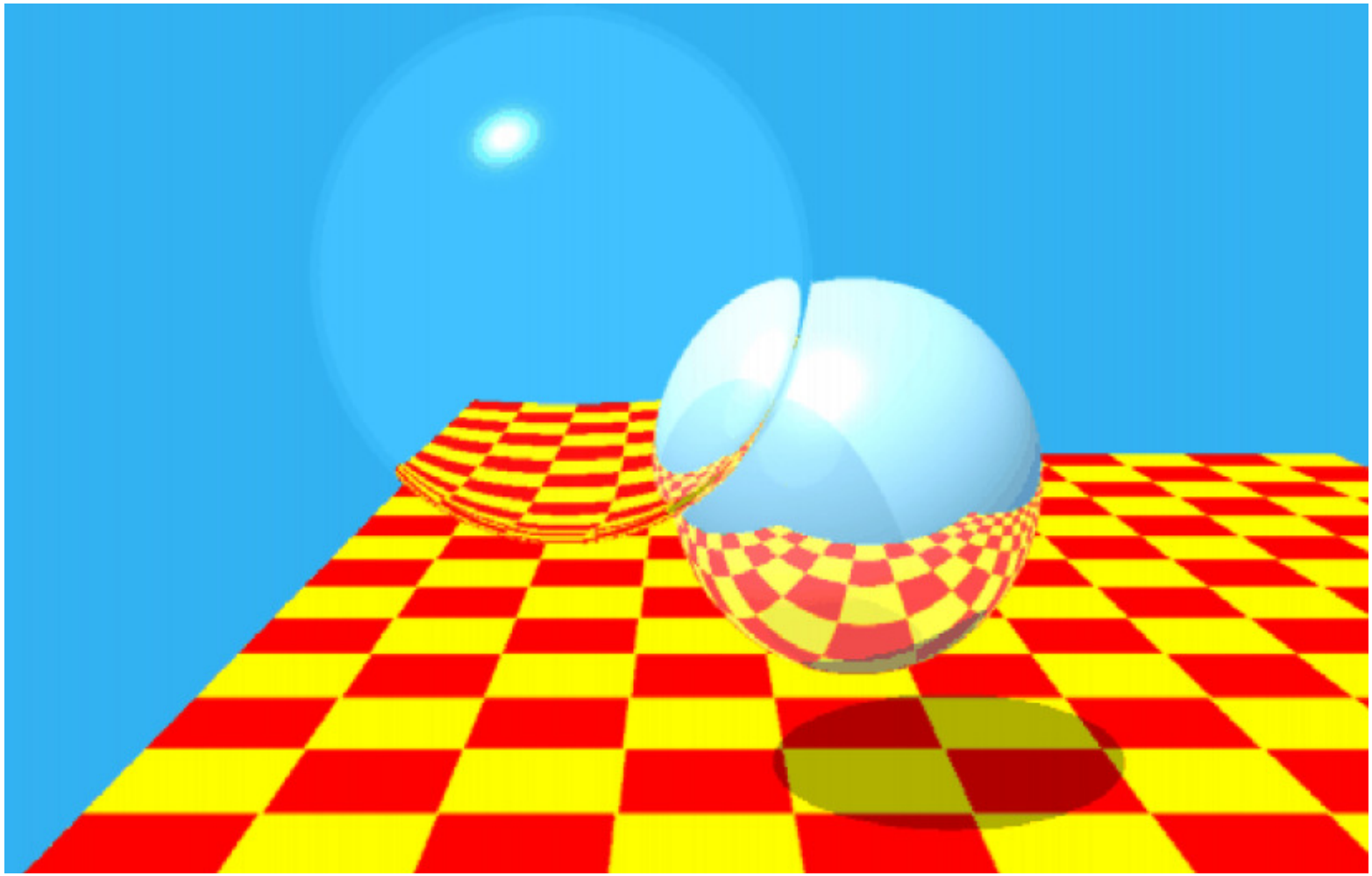


Ray Tracing -- 2 bounce



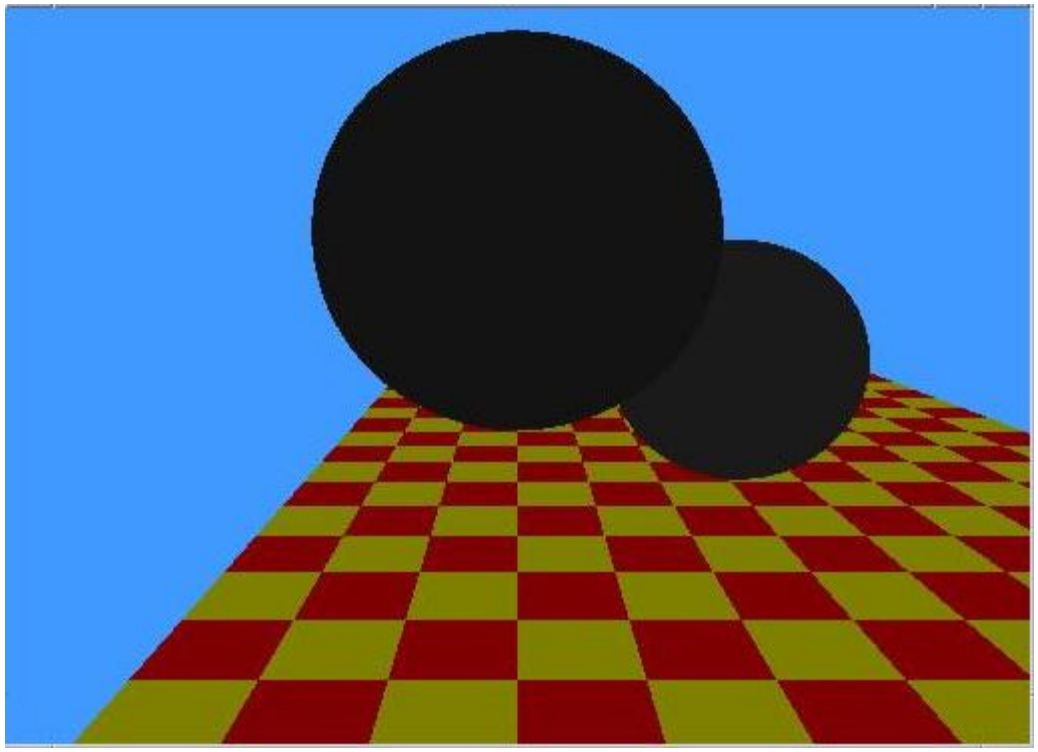
Ray Tracing -- 3 bounce

Рекурсивен алгоритъм



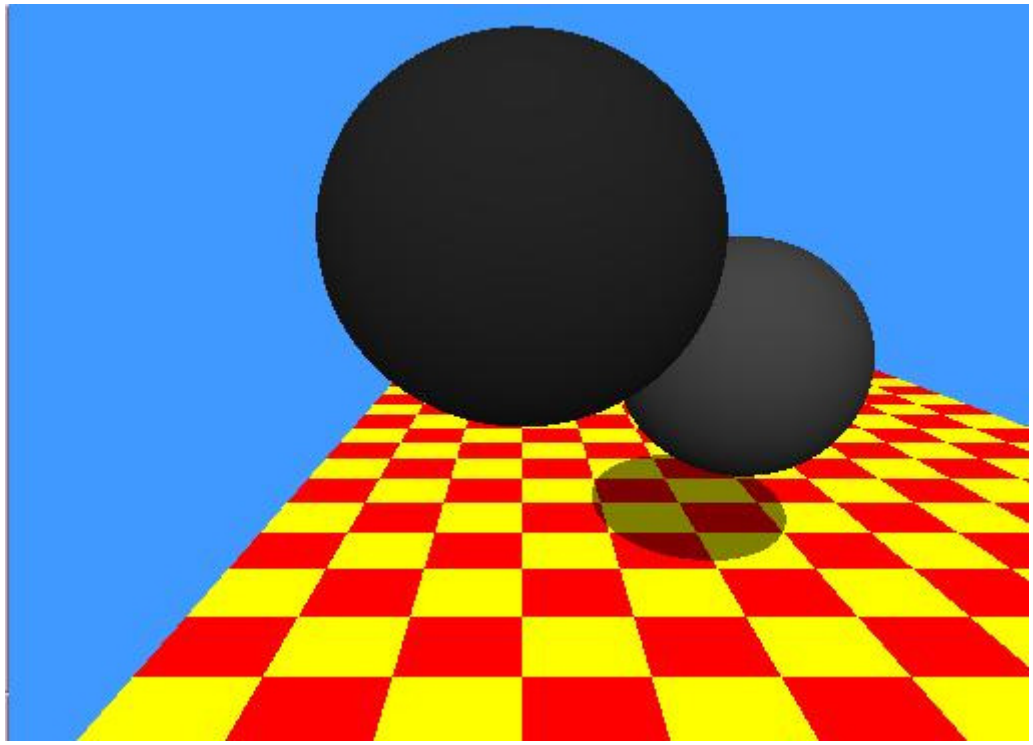
Ray Tracing

- Само фонова светлина



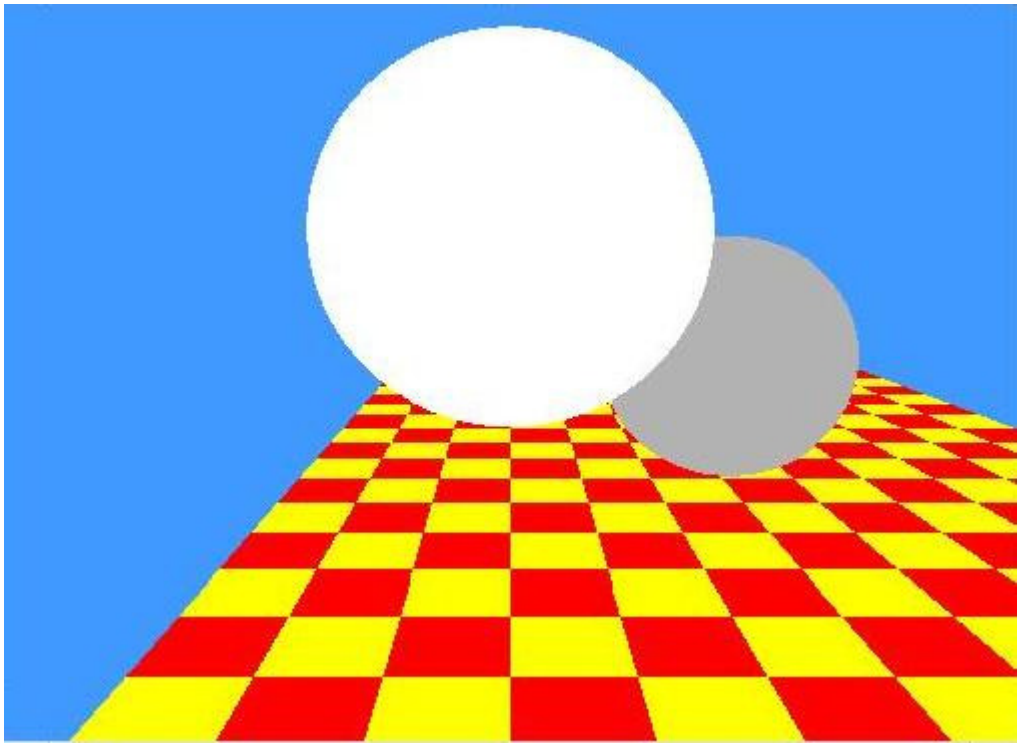
Ray Tracing

- Със сенки



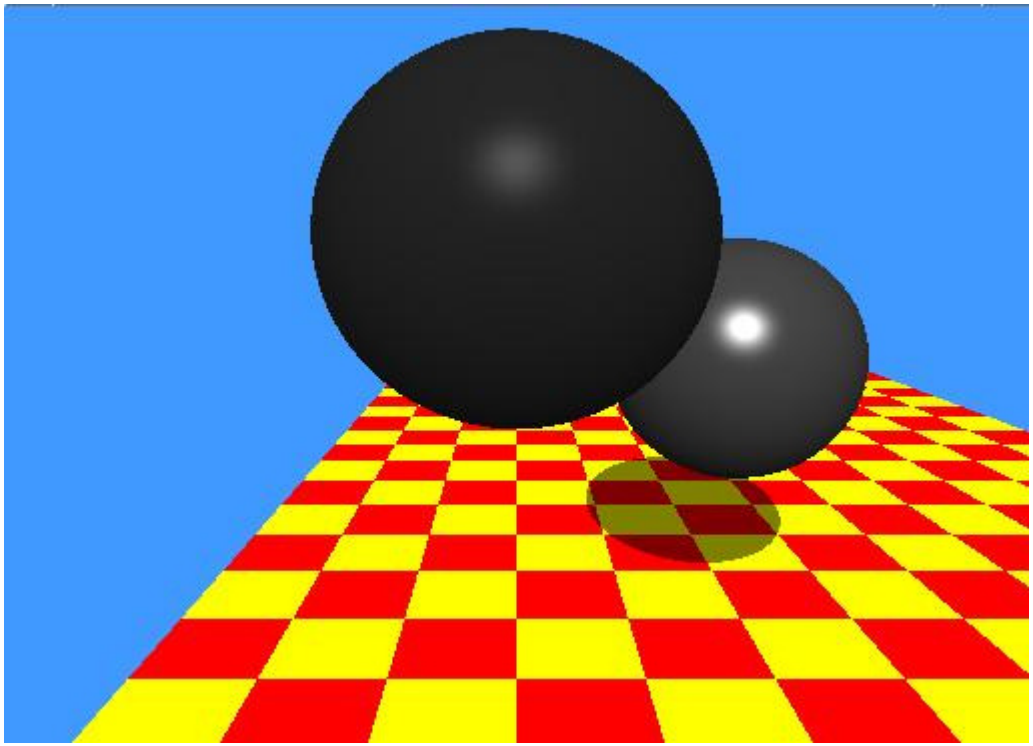
Ray Tracing

- Само основните лъчи



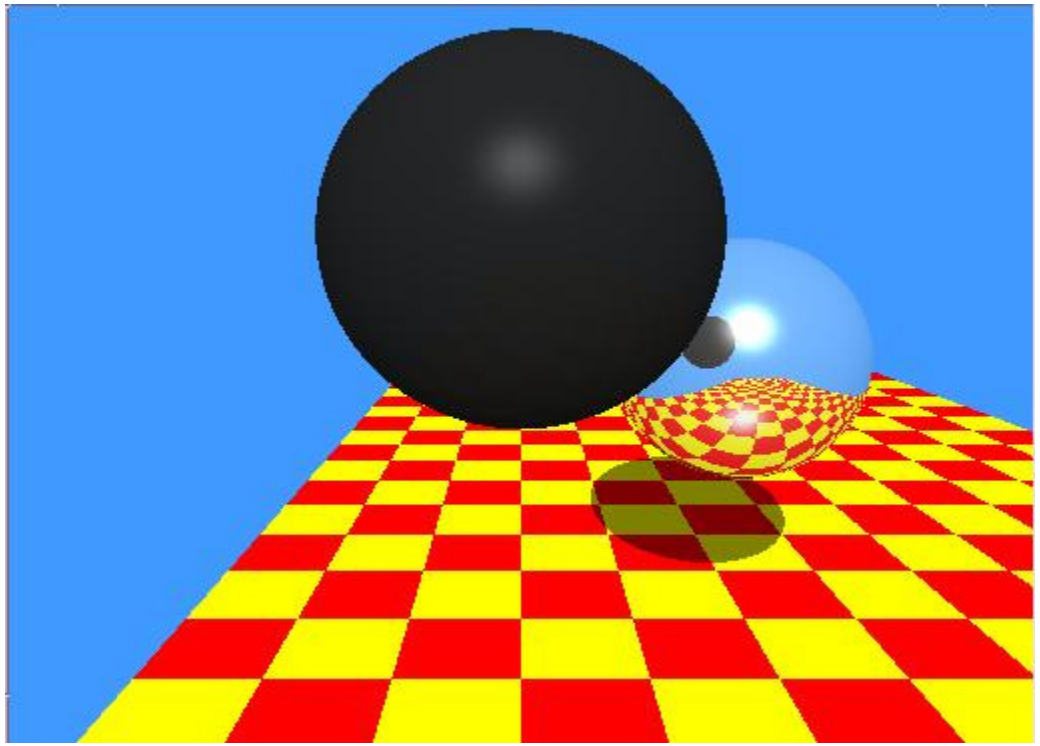
Ray Tracing

- С модел на осветеност (Фонг)



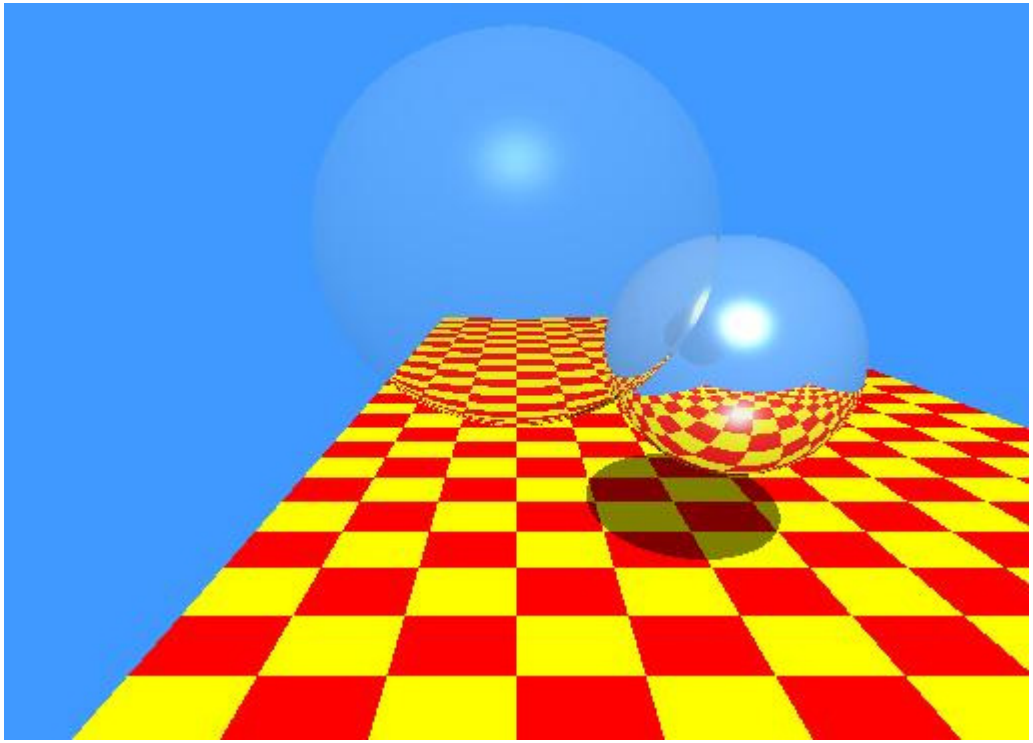
Ray Tracing

- С отразяване



Ray Tracing

- С пречупване и прозрачност

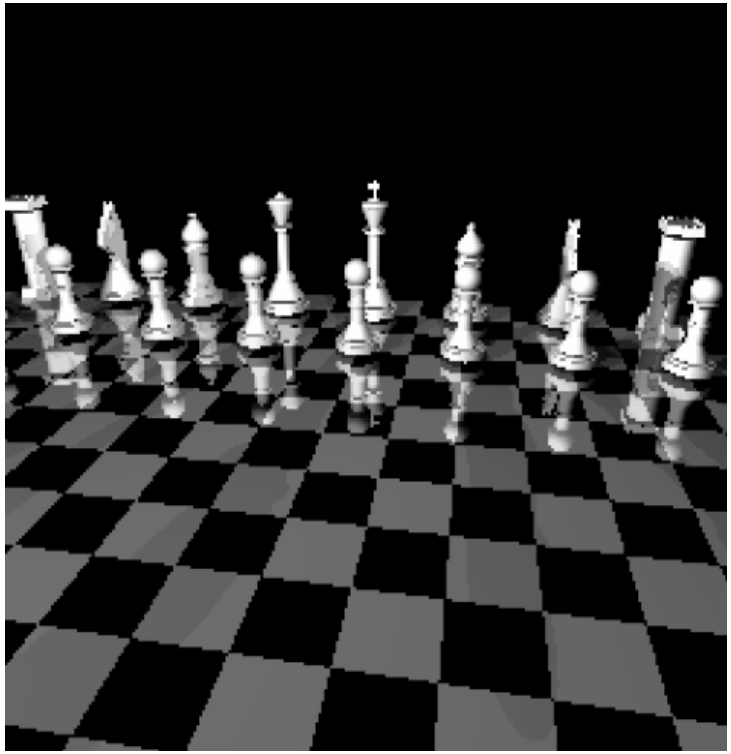


Aliasing

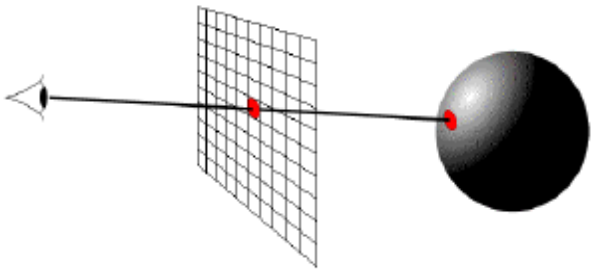
- Алгоритъмът с трасиране на лъчи определя цвят за всеки пиксел в изображението
  - квадратният пиксел съдържа *безкраен брой точки*
  - точките в един пиксел може да нямат еднакъв цвят

- Sampling (дискретизиране)
  - избира се цвят за всеки пиксел, който е цвета на точката в центъра на пиксела

- *Aliasing*
  - нащърбвания (jaggies)
  - Moire patterns



Aliasing

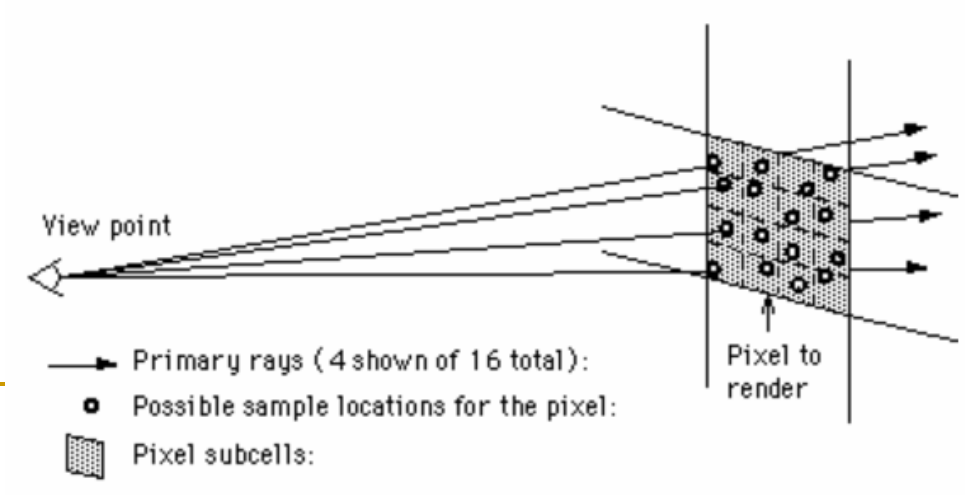


- ***Supersampling***
  - няколко лъча за всеки пиксел
    - например матрица 3×3 с лъчи в центъра и в ъглите на пиксела
    - осредняване на резултата
- ***Adaptive sampling***
  - увеличава се броят на лъчите в области с големи изменения
    - в геометрията или осветяването
- ***Stochastic sampling***
  - случаен избор на точките в пиксела
- ***Subsampling***
  - по-малко точки отколкото пиксели
    - толкова на брой, колкото времевото ограничение позволява
    - *beam tracing*: проследяване на сноп съседни лъчи заедно

- ***За преобразуване на резултатите от множество лъчи се прилагат изглаждащи (осредняващи) филтри***

Antialiasing

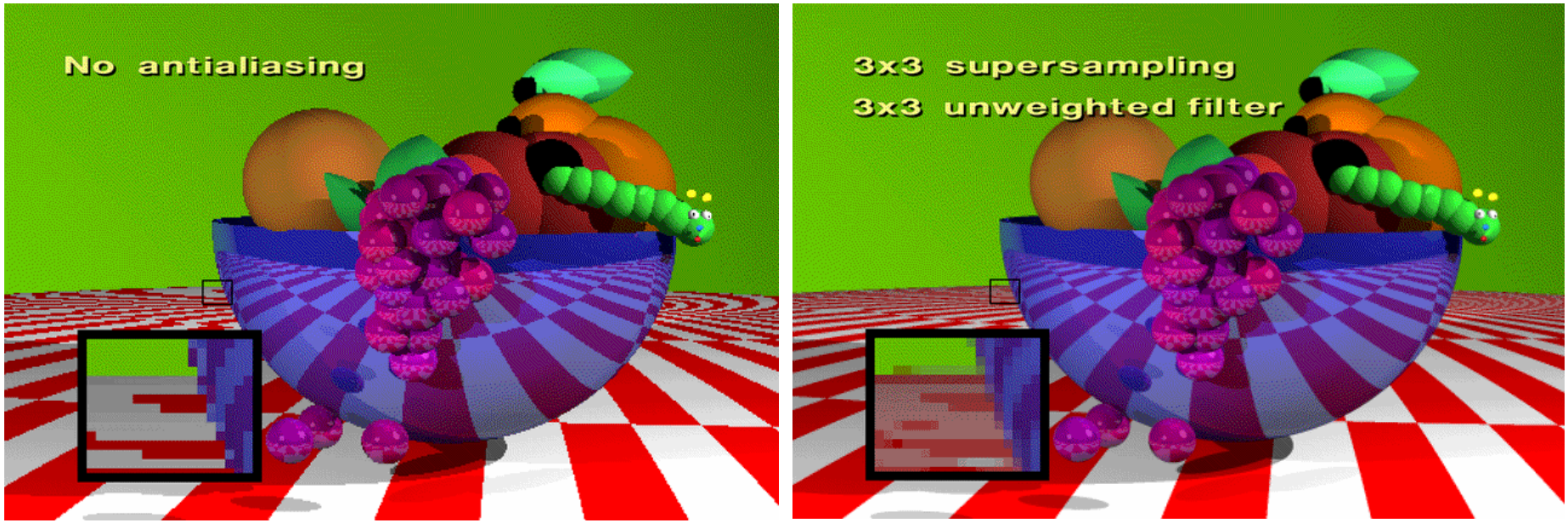
- ***Stochastic sampling***
  - пикселът се разделя на матрица от клетки с предварително определени размери
  - основният лъч се трасира през случайно избрани позиции във всяка от клетките
  - резултатите се осредняват с претеглена средна стойност
    - теглата зависят от разстоянието до центъра на пиксела



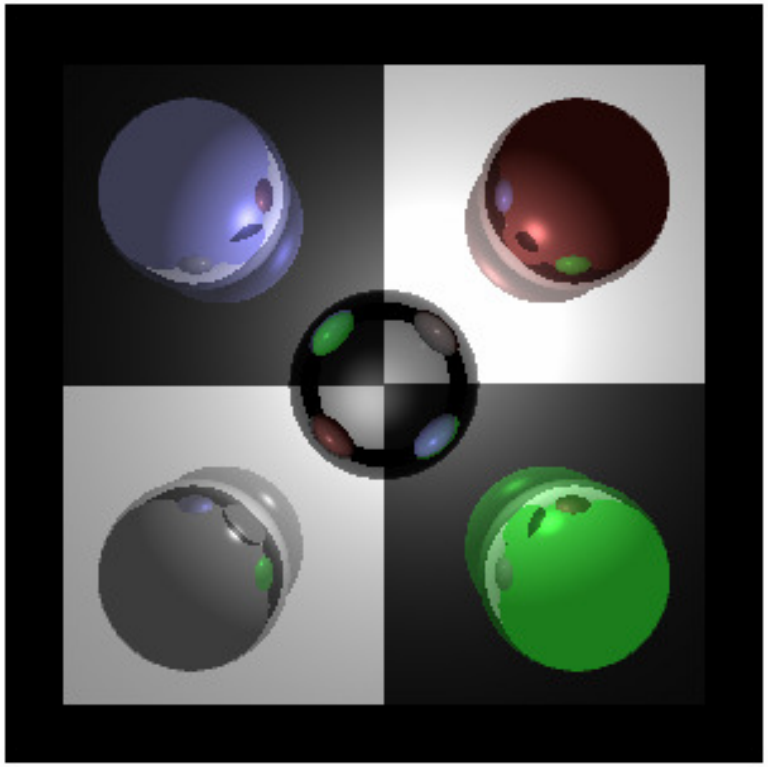
Motion Blur

- ***Temporal Aliasing***
  - Aliasing се наблюдава както в пространството, така и във времето
    - честотата на изчисляване е frame rate, 30Hz за NTSC video, 24Hz за филм
    - бързо движещите се обекти се преместват на голямо разстояние между кадрите
  - Anti-temporal aliasing
    - филтриране във времето
    - кадрите се изчисляват за 120Hz и се осредняват
    - изчислително сложно
  - Автоматичен temporal antialiasing
    - фото изображения интегрират стойностите за целия период на времето на експозиция
    - видео камерите осредняват съхранените в паметта стойности
    - резултатите в изображенията са *motion blur*

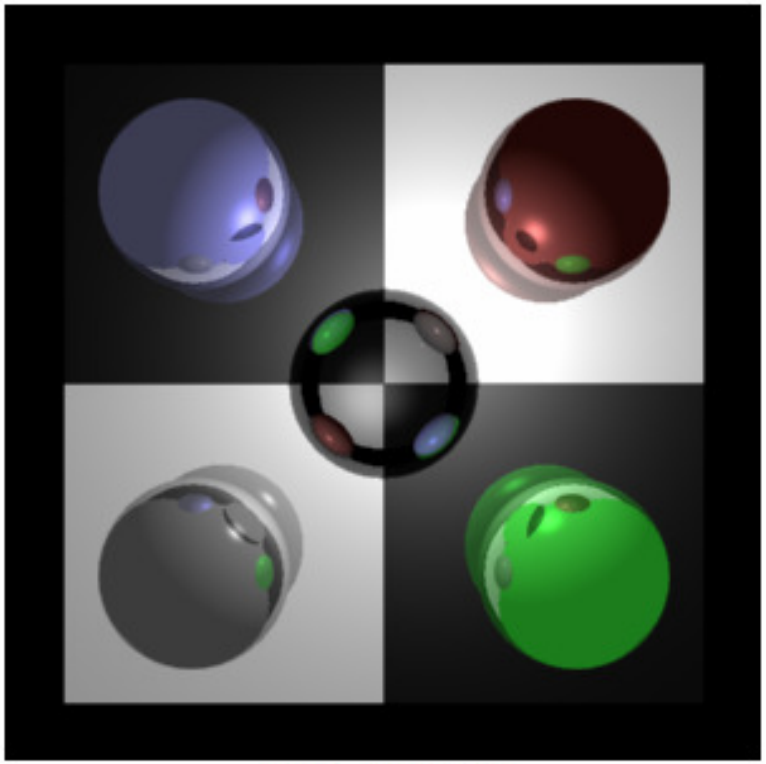
Antialiasing



Antialiasing



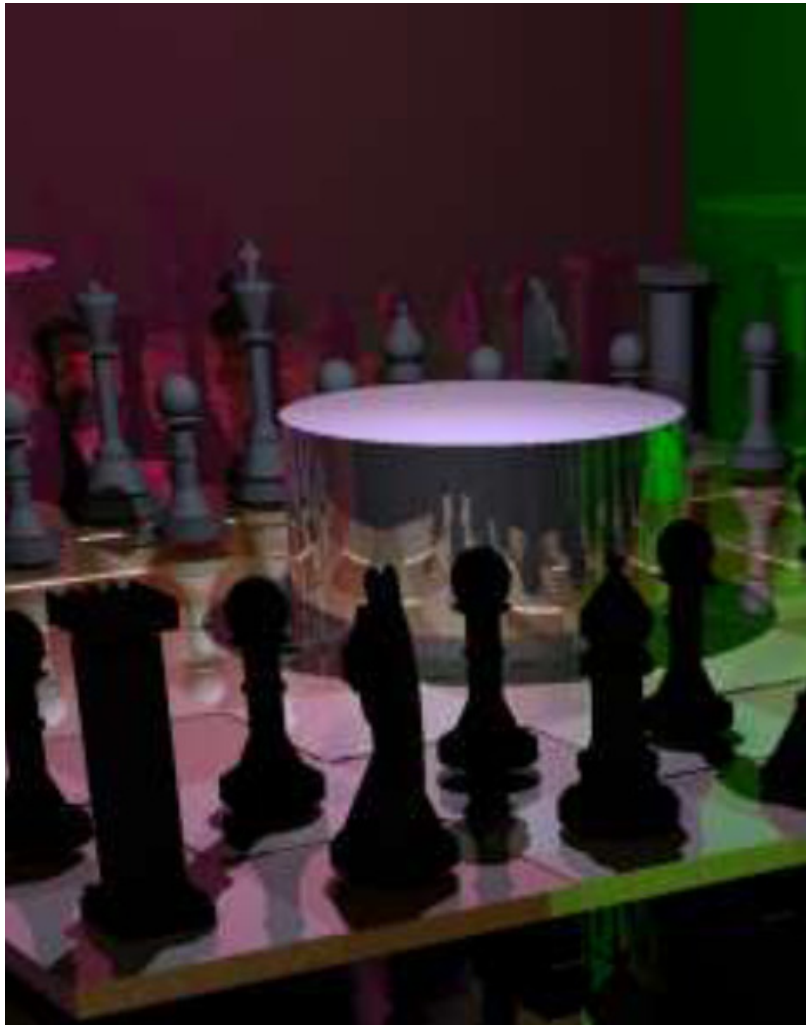
Aliased



Anti-aliased

Antialiasing

без supersampling



supersampling

Ускоряване на трасирането на лъчи

- ***Хардуерно ускорение***
  - използване на по-бързи компютри
  - използване на паралелни компютърни платформи
- ***Софтуерно ускорение***
  - използване на паралелни изчисления
  - прилагане на по-ефективни алгоритми

Ускоряване на трасирането на лъчи

- Прилагане на по-ефективен алгоритъм
  - Ускоряване на намирането на първа точка на пресичане
  - Използване на ограждащи обеми
  - Намаляване на броя на изчисляваните точки на пресичане
  - Задаване на дълбочина на рекурсията

Ускоряване на трасирането на лъчи

- ***Ускоряване на намирането на първа точка на пресичане***
  - използва се модифициран алгоритъм на Z-буфер
    - записва в буфера не на z-координатите на точките на пресичане, а указатели към обектите с най-близка точка на пресичане
    - трасирането на лъчи започва от тази точка

Ускоряване на трасирането на лъчи

- ***Ограждащи обеми***
- Всеки сложен обект се огражда с прост (ограждащо тяло)
  - сфера, куб
- Ако ограждащото тяло не е видимо, то и обектът вътре в него не е видим

- Допълнителна ефективност
  - няколко тела могат да са в едно ограждащо тяло

- ***Приложение при трасиране на лъчи***
  - бързо отхвърляне
    - лъчите най-напред се пресичат с ограждащото тяло
  - още по-бързо отхвърляне
    - групи лъчи (frustum) се пресичат с ограждащото тяло на обект

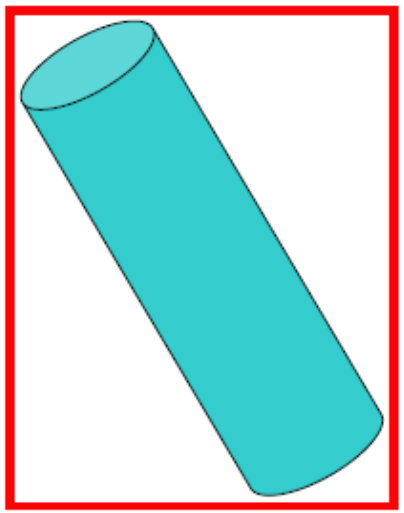
Ограждащи обеми

- Лесна имплементация
- Значително ускорение
- Ефективност при плътно обхващащ тялото ограждащ обем
  - по-плътно обхващащо тяло се изчислява по-трудно
    - например на стол

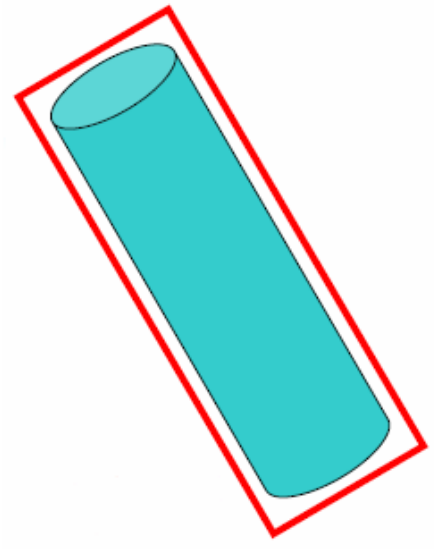
- ***Axis Aligned Bounding Boxes (AABB)***
  - куб с ръбове успоредни на някоя от осите x, y или z

- ***Йерархични ограждащи обеми (Bounding Volume Hierarchy)***
  - комбинират се съседни ограждащи обеми
  - изгражда се дървовидна структура
  - изгражда се отдолу нагоре

Ограждащи обеми

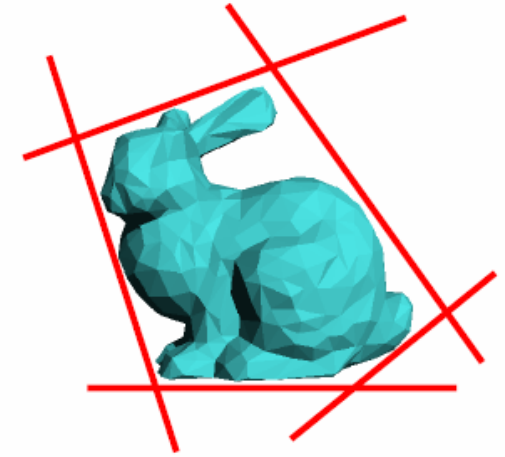


Ограждащ паралелепипед по координатните оси Axis Aligned Bounding Box



Ограждащ паралелепипед по осите на обекта Axis Aligned Bounding Box

Ограждаща сфера



Произволен изпъкнал регион (определен от разделящи линии)

Ограждащи обеми

- ***Йерархични ограждащи обеми***
  - итеративно групиране на ограждащите обеми на близки обекти докато не се огради цялата сцена
  - определянето на близко разположени обекти може да бъде доста трудно
    - тривиален алгоритъм $O(n^2)$, със сортиране може да бъде $O(n \log n)$
  - влошава се сложността на проблема с определяне на плътно обхващащи обеми
- Може да се използва графа на модела на сцената
  - ограждащите обеми във възлите са обединение на ограждащите обеми на възлите-наследници
    - лесно се конструира
    - графът на сцената е логически, но може да не е пространствено организиран

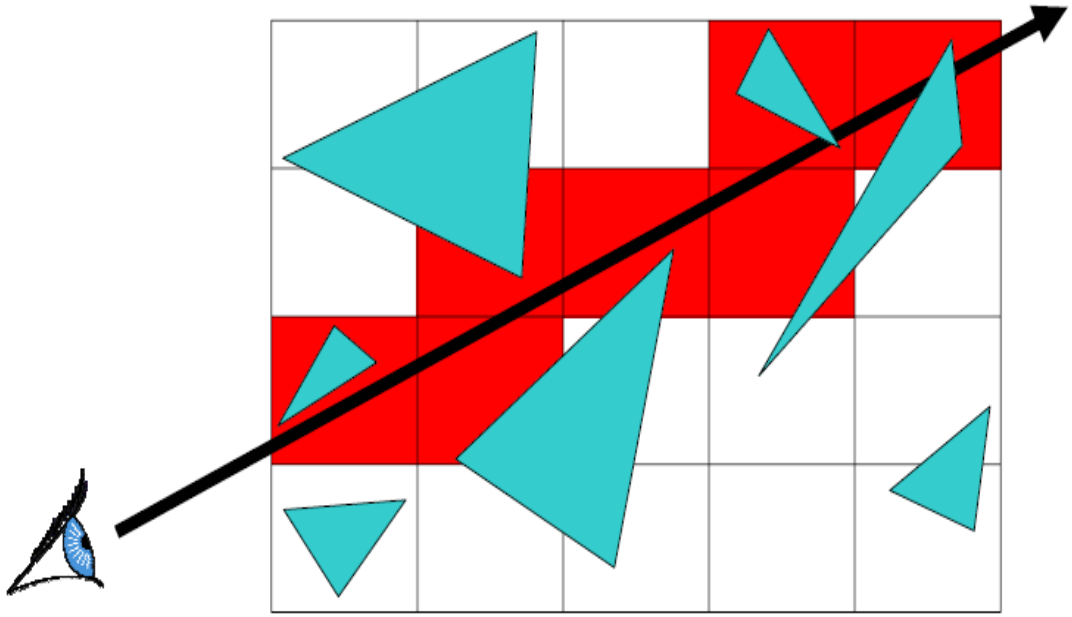
Ограждащи обеми

- Приложение в Ray Tracing
  - ако лъчът пресича родителя се проверява дали пресича децата
  - ако лъчът не пресича родителя не се разглеждат деца

- ***Недостатък на йерархичните ограждащи обеми***
  - не повишава ефективността за произволни сцени
    - например много детайлни мрежи или пейзаж с трева и дървета

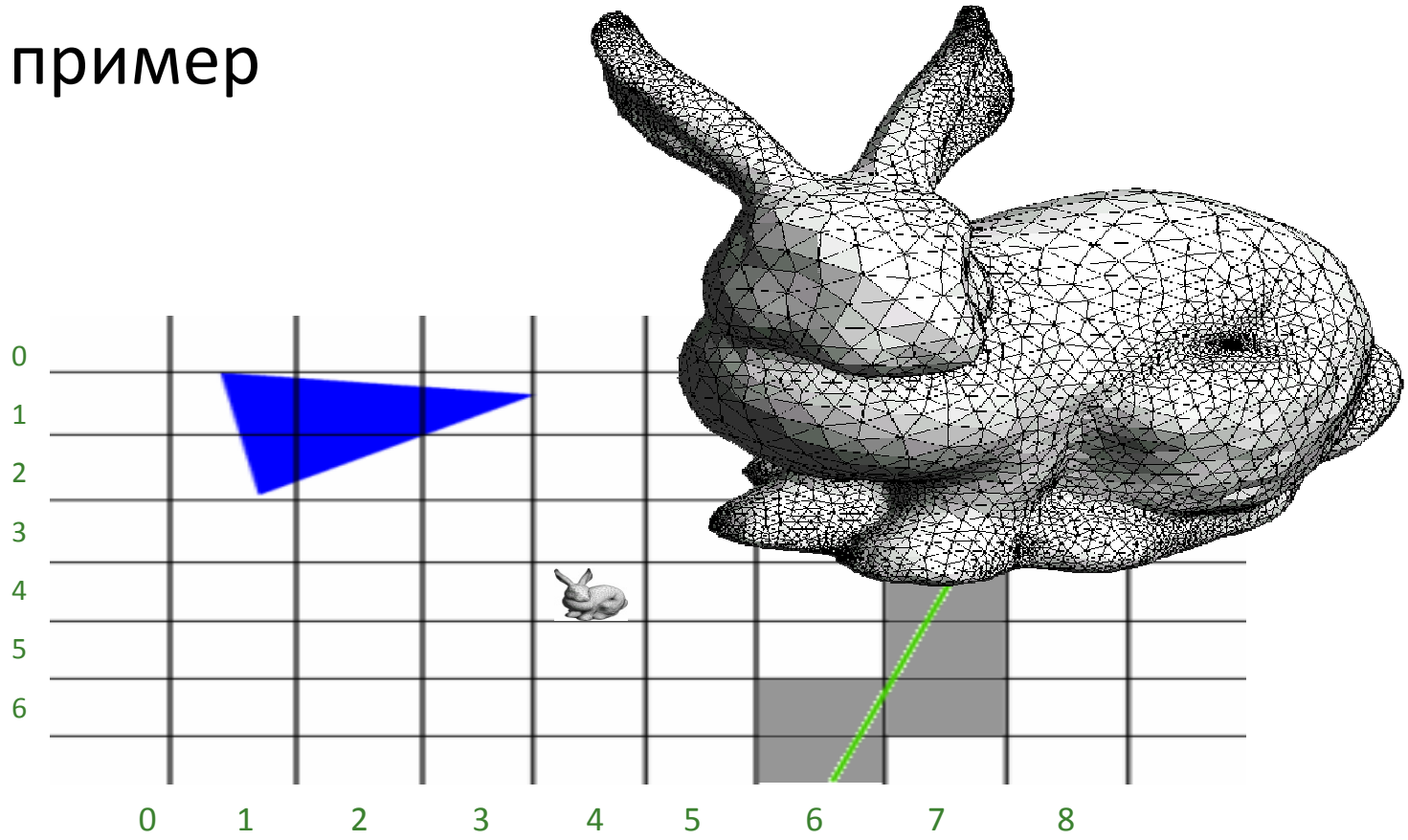
Решетка (Grid)

- ***Разделяне на пространството (Grid)***
  - вместо да се определят ограждащи обеми отдолу-нагоре за сцената
  - пространството се разделя на регулярни клетки



Решетка (Grid)

- 2D пример



Решетка (Grid)

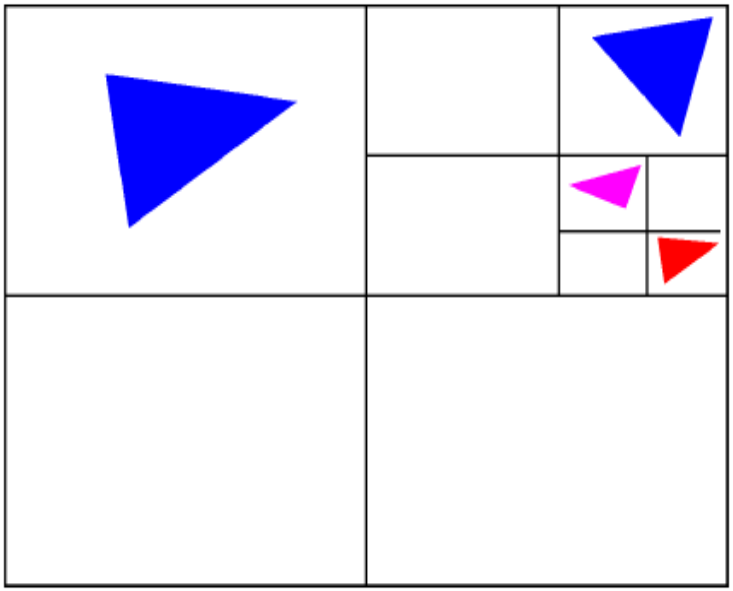
- ***Предимства***
  - лесно и бързо създаване
    - подходящо за анимирани сцени
    - преместването на един обект не влияе на другите обекти в решетката
  - лесно приложими са алгоритми със сканираща линия
    - клетките могат да се разглеждат като пиксели и трасирането на лъчи като растеризиране на линия
  - лесно се имплементират хардуерно

Решетка (Grid)

- ***Недостатъци***
  - Липсва балансиране
    - някои клетки са по-важни, но не може да се определи кои
    - много клетки не съдържат обекти, но се губят ресурси да се обработят
  - Липсва адаптивност на размера на клетките
    - ако се използват по-големи по размер клетки
      - може в една клетка да има твърде много обекти
    - ако се използват по-малки по размер клетки
      - увеличават се изчисленията за сканиране на всички клетки
        - аналогично на super-sampling при обработка на изображения
  - Липсва йерархия

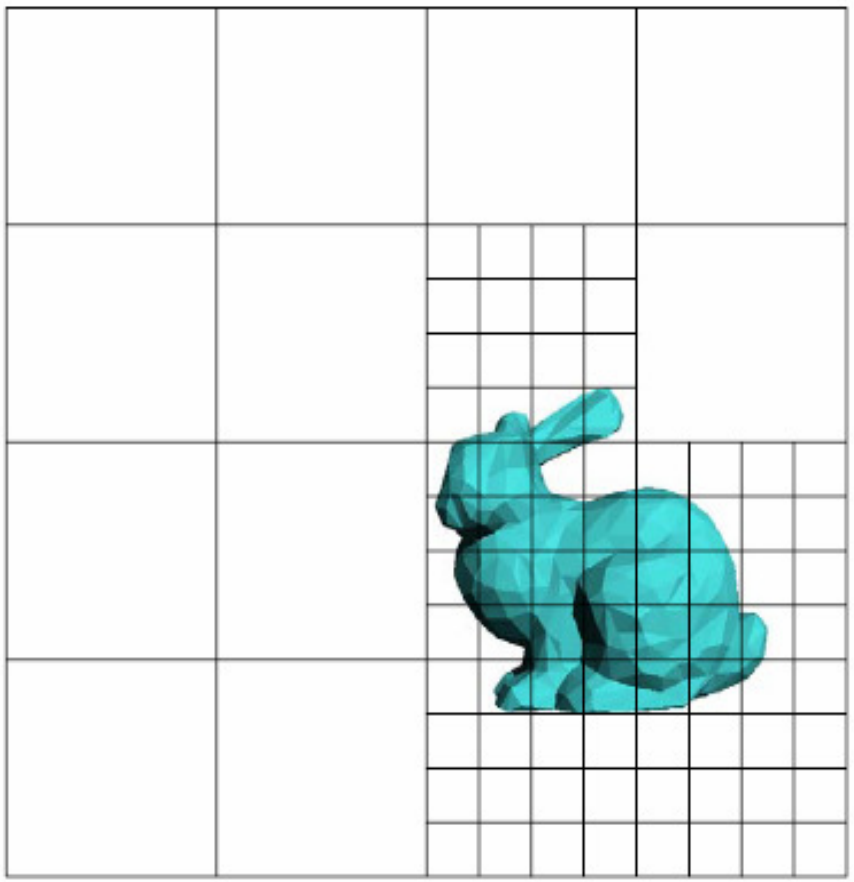
Осмични дървета

- ***Octrees***
  - комбинират предимствата на ограждащи обеми и решетки
    - директно описание на сцената (решетки)
    - адаптивни и йерархични (ограждащи обеми)
    - независими от позицията на наблюдение
- Подобни на решетка
  - няма изискване за еднакъв размер на вокселите (voxels)
    - повече воксели при сложна геометрия
- Всички възли са AABB
  - Axis Aligned Bounding Box

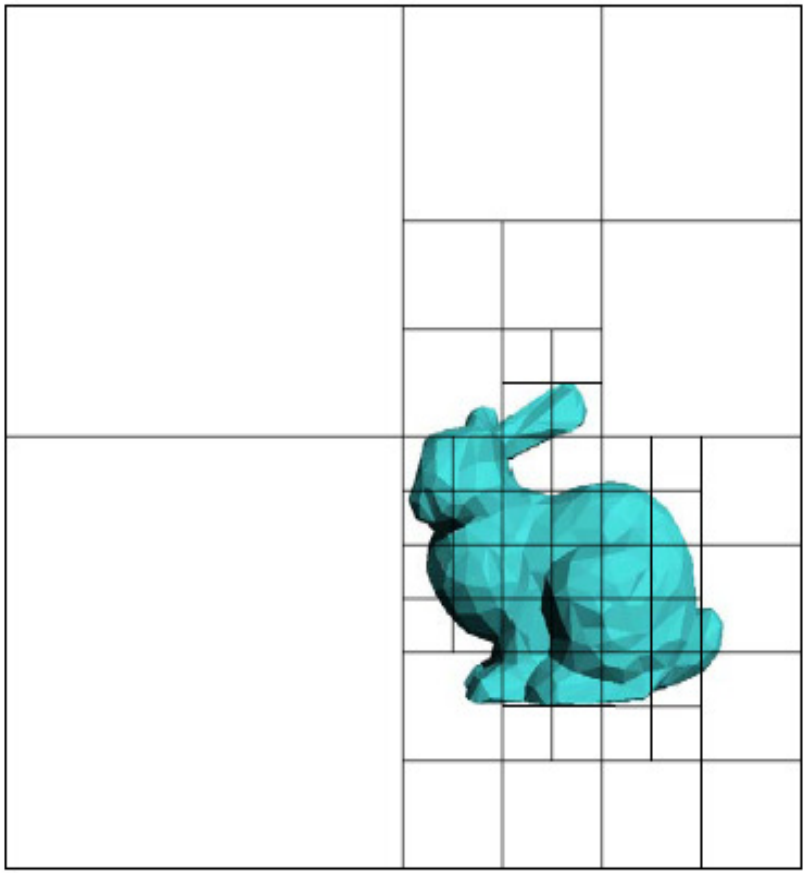


Осмични дървета

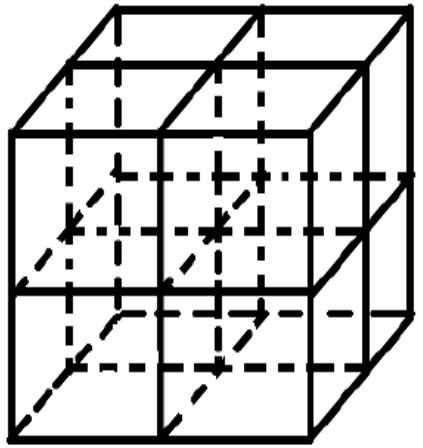
Nested grid



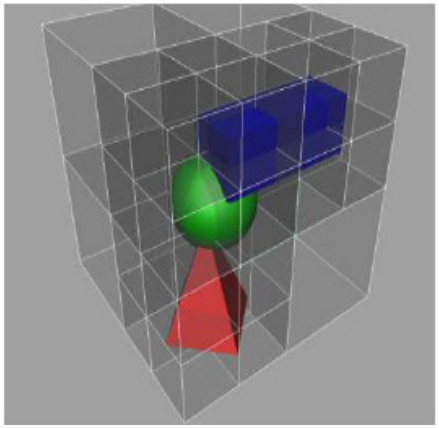
Octree (quadtree)



Осмични дървета

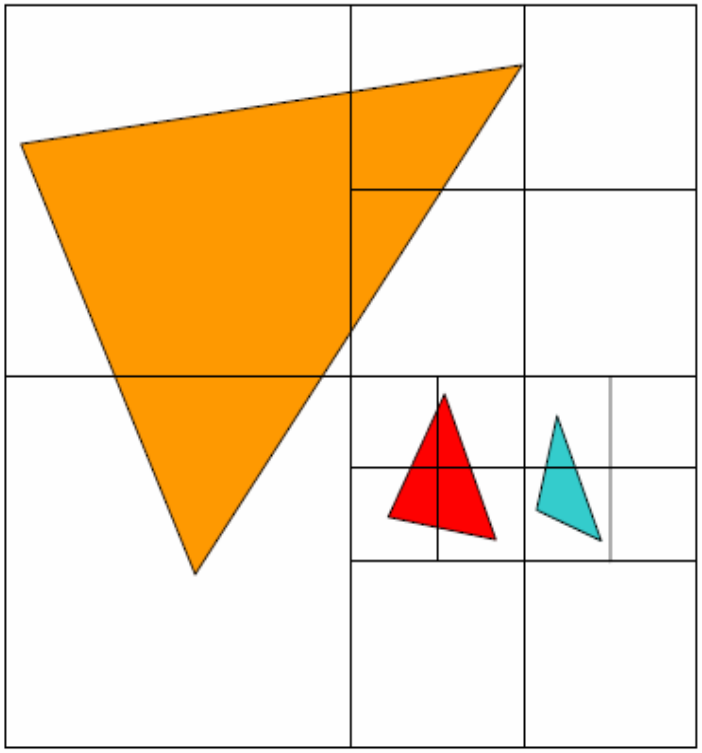


- ***Конструиране***
  - Рекурсивно, отгоре надолу
    - вместо отдолу нагоре както при ограждащите обеми
  - Определя се ограждащ обем на сцената
    - корен на дървото
    - съдържа всички примитиви
  - На всяка итерация текущият възел се разделя на 8 октанта
    - лесно се определят ако са успоредни на осите
    - примитивите в текущия възел се разделят в октантите
  - Новите възли в октантите се разделят рекурсивно
    - разделянето спира при максимална дълбочина или ако воксел съдържа достатъчно малко примитиви
  - Подобряване на ефективността
    - обединяване на празни съседни воксели

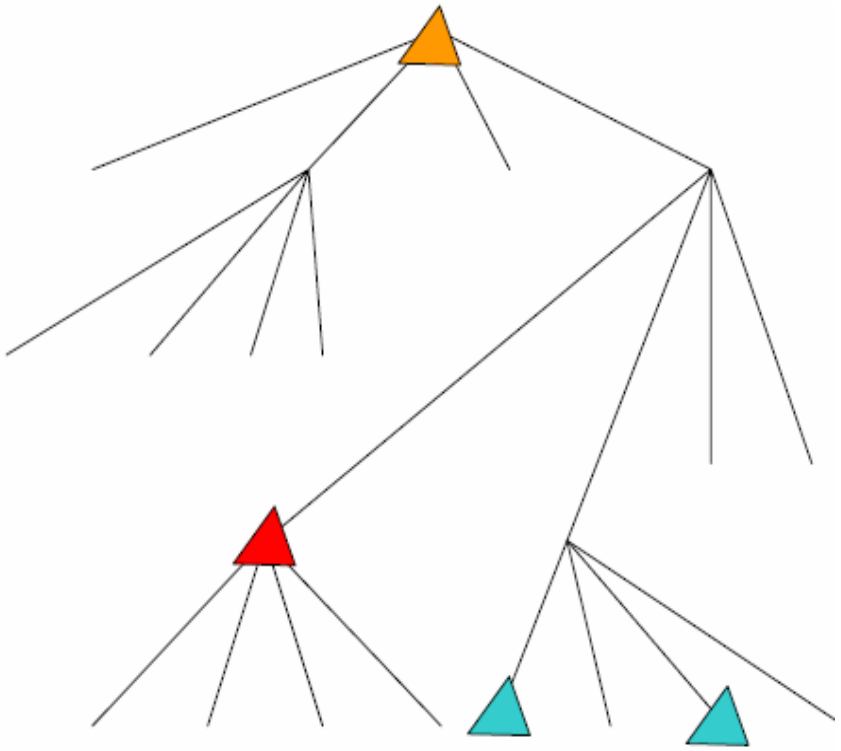


Осмични дървета

- ***Конструиране***

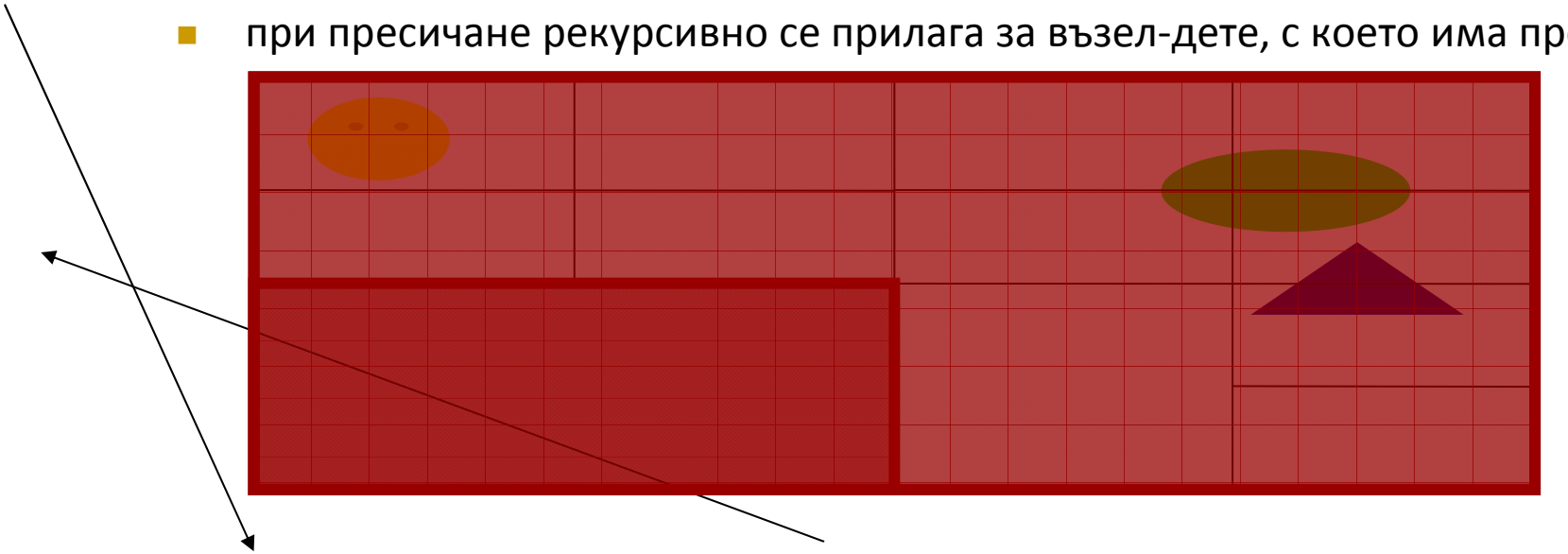


Octree/(Quadtree)



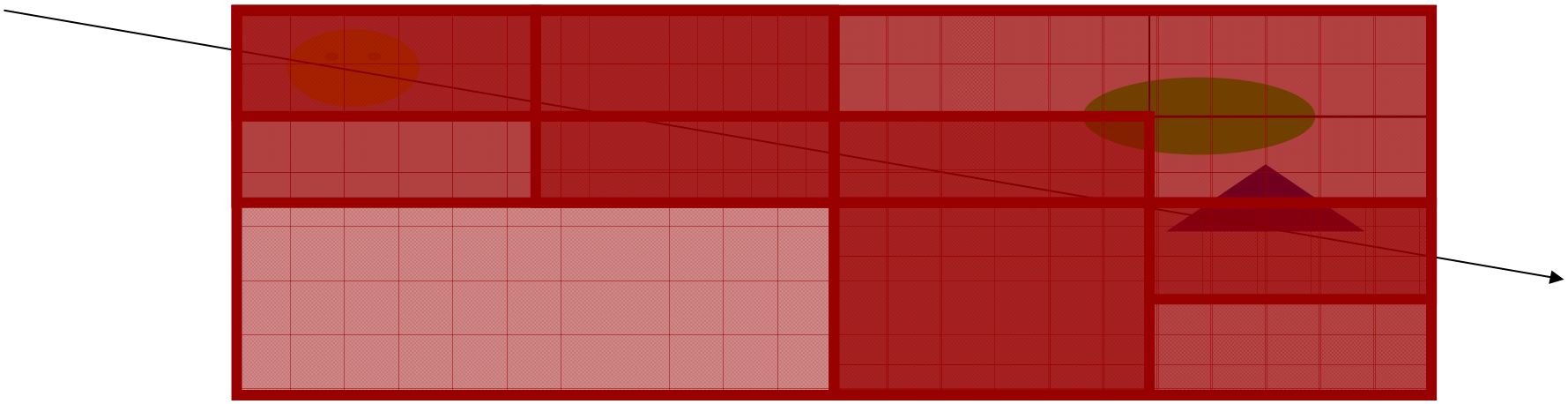
Осмични дървета

- ***Трасиране***
  - започва от корена
  - ако възел е лист се определя пресичане с лъча
  - ако възел не е лист се определя пресичане с лъча за ограждащите обеми на всички негови деца
    - при пресичане рекурсивно се прилага за възел-дете, с което има пресичане



Осмични дървета

- ***Трасиране***



Осмични дървета

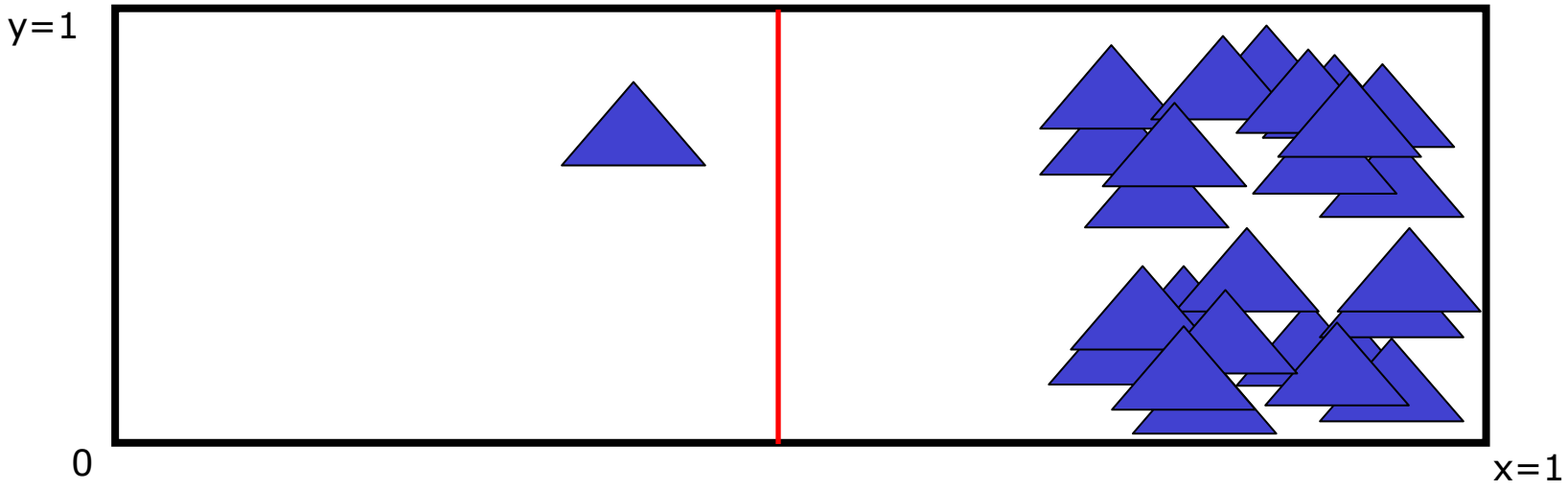
- ***Предимства***
  - могат да се използват за произволна сцена с добра производителност
  - очаквана сложност O(log n) за всеки лъч
- ***Недостатъци***
  - лоша производителност за сцени с много неравномерно разпределени примитиви
    - често срещани сцени
    - създават се неефективни дървета с голяма дълбочина и много разделяния в зони със сложна геометрия

kd-дървета

- ***Основна идея***
  - решение, което да отчита относителната цена за обхождане на клетките в решетка с извършване на тест за пресичане
  - бързо се изолират области с висока сложност и празни области
- ***Дефиниция***
  - kd-tree е k-мерно ориентирано по осите бинарно дърво
    - при k=2 се нарича четвъртично дърво (quadtree)
  - ориентирано по осите $\Rightarrow$ бързо трасиране
  - бинарно дърво $\Rightarrow$ избира се само една ос за разделяне на всеки възел
    - за разлика от осмичното дърво
- Основен проблем при пространствено разделяне с kd-дървета
  - определяне на позицията на разделящата равнина за всеки възел, в кл. по коя ос да е разделянето
    - разделящата равнина да съответства на геометрията на обектите

kd-Trees

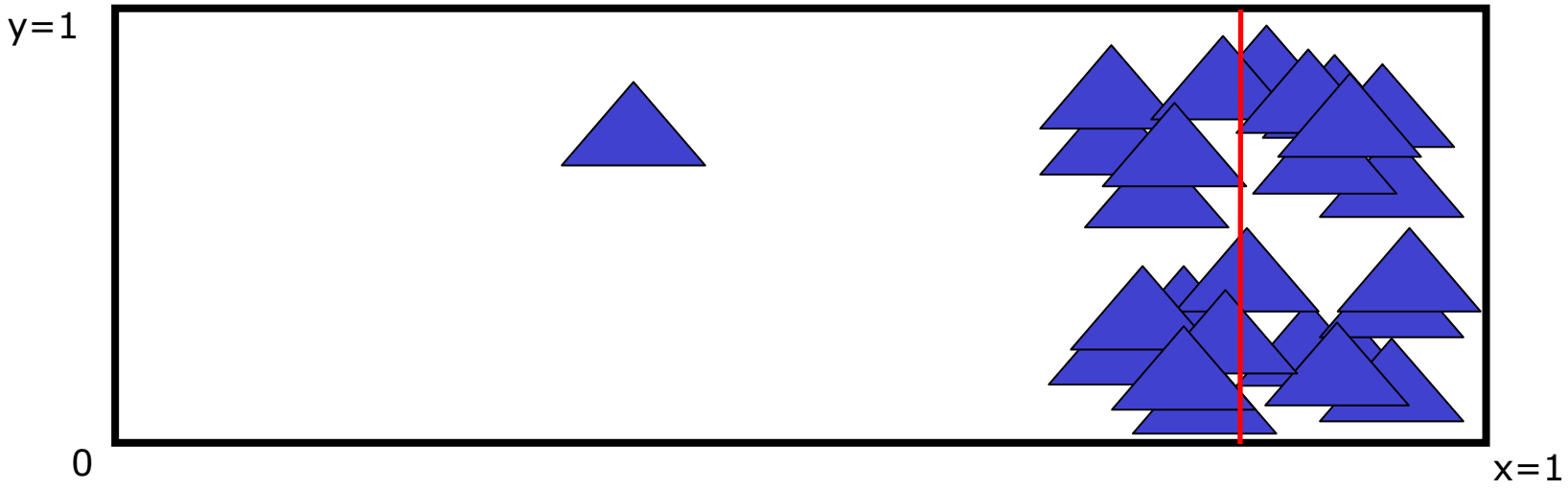
Определяне на разделяща равнина (2D primer)



- ***Къде да се раздели даден възел пространствено оптимално?***
- ***В средата***
  - при трасирането е еднакво вероятно лъчът да пресича и лявата, и дясната страна
    - но в случая цената на пресичане отдясно е много по-голяма

kd-Trees

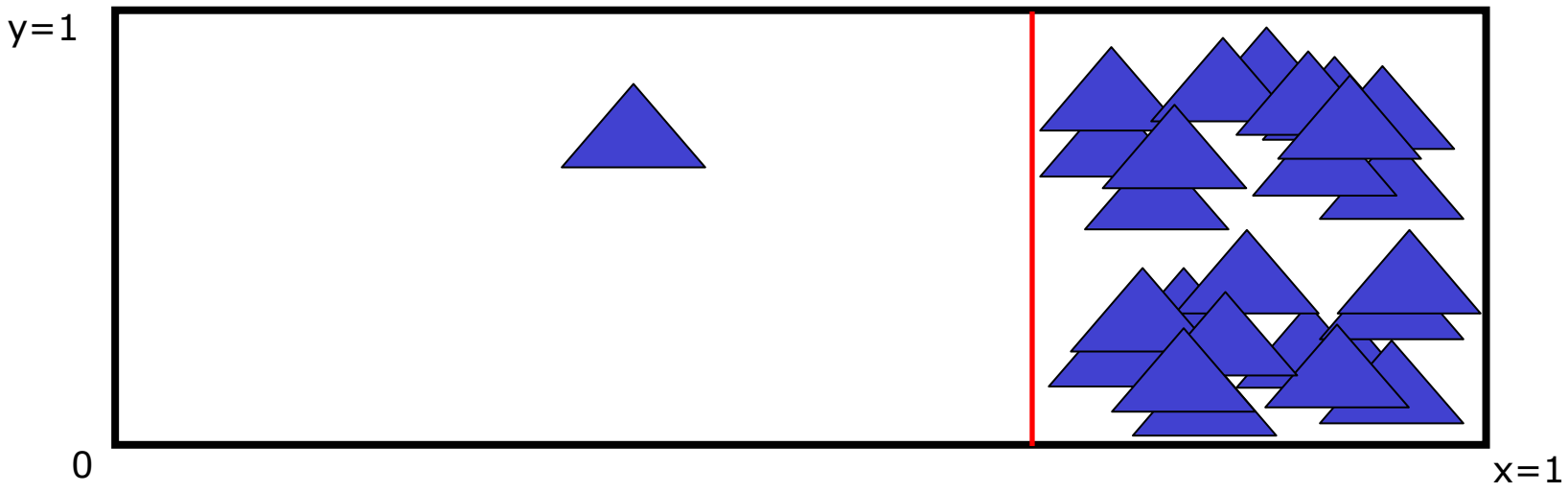
- ***Определяне на разделяща равнина (2D primer)***



- ***Къде да се раздели даден възел пространствено оптимално?***
- ***В медианата***
  - цената за трасиране на всяка страна в смисъл на възможни пресичания е приблизително еднаква
    - но лъчът е по-вероятно да премине през лявата страна, която е с много по-голяма площ

kd-Trees

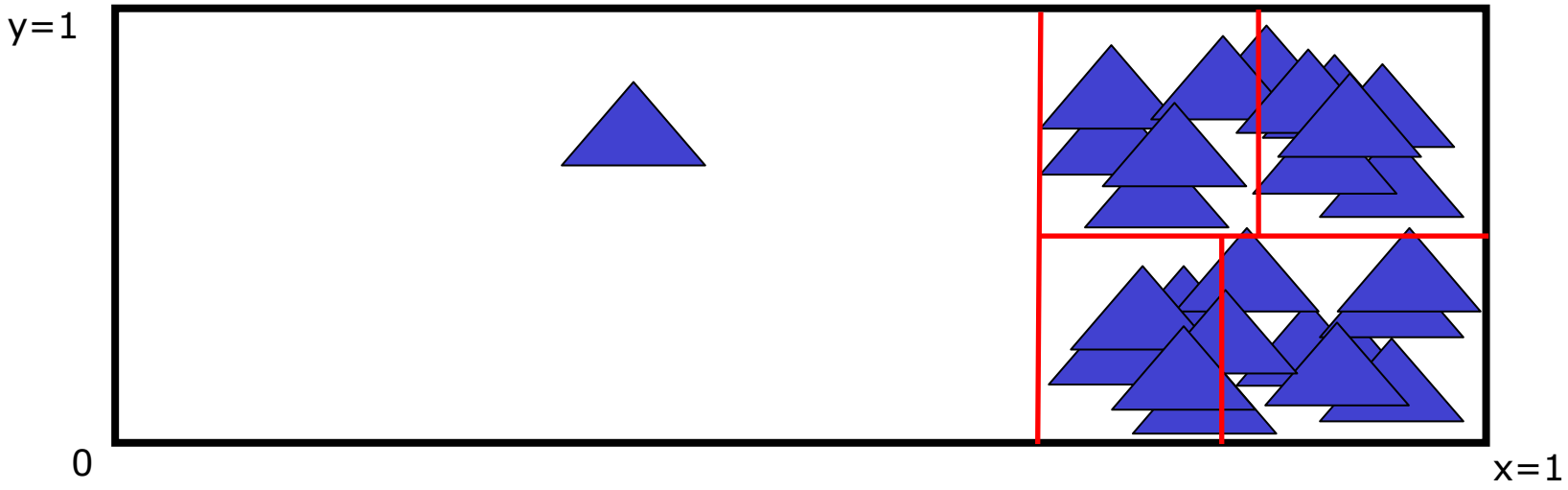
- ***Определяне на разделяща равнина (2D primer)***



- ***Къде да се раздели даден възел пространствено оптимално?***
- ***Cost-Optimized Split***
  - баланс между цената за преминаване на лъч (тестове за пресичане) и вероятността за преминаване през този възел
    - бързо се изолират геометрично сложните области и се създават големи празни възли, които през които лъчът не се трасира

kd-Trees

- ***Определяне на разделяща равнина (2D primer)***

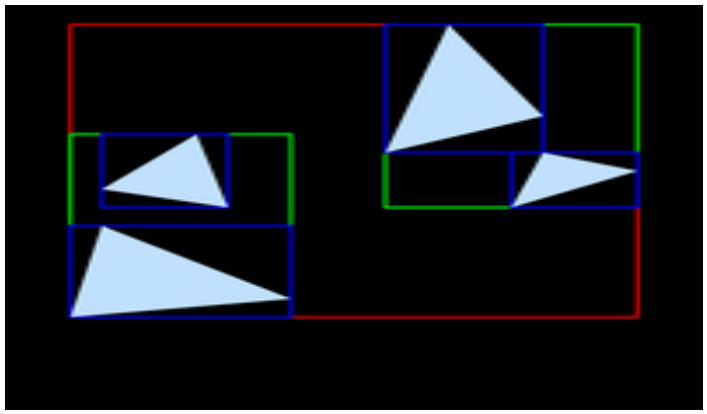


- ***Къде да се разделят възлите-деца?***
- аналогични стратегии
  - среда, медиана, оптимална цена

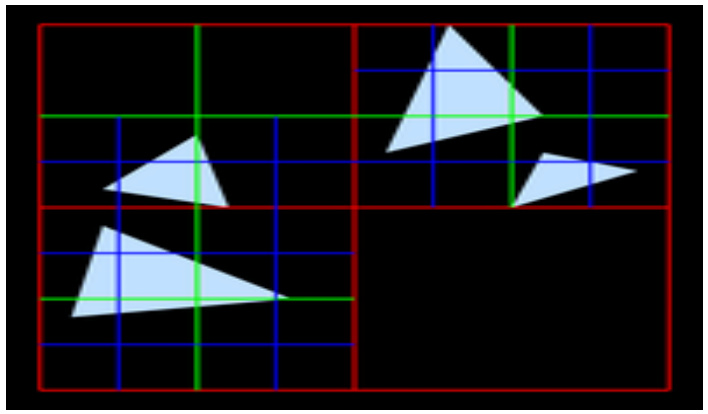
kd-Trees

- ***Трасиране***
  - може да се използва алгоритъм както при осмични дървета
  - може да се приложи ранно терминиране
  - двата възела-деца се трасират не в произволен ред, а най-напред първият възел-наследник, който се пресича от лъча
    - първият пресечен възел-дете е “frontside child”
    - вторият пресечен възел-дете е “backside child”
  - при пресичане за “frontside child”, което е близо от пресичането на ограждащия обем на “backside child”, то “backside child” не се трасира

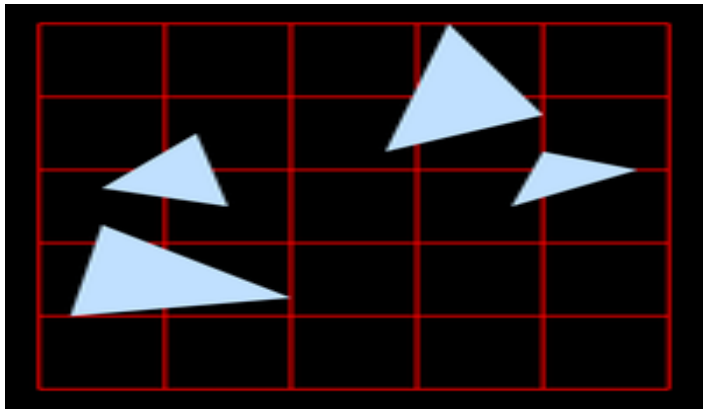
Ускоряване на трасирането на лъчи



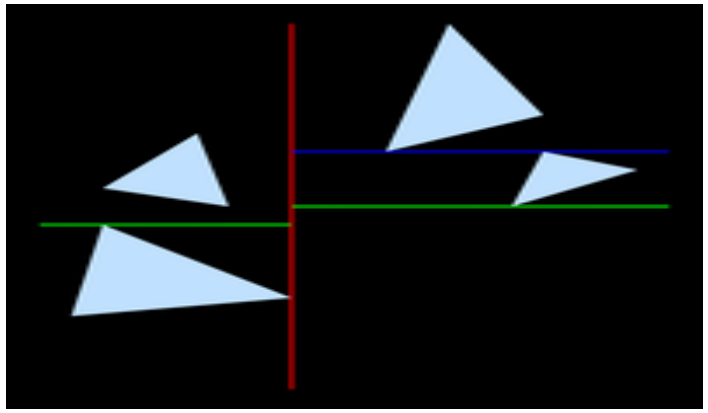
Bounding Volume Hierarchy



Octrees



Grids



kd-tree

Структури данни за ускоряване на трасирането на лъчи

RayTracing в реално време

- Традиционно изчислително невъзможно да се извърши в реално време
  - “embarrassing parallel”
    - независимост на всеки лъч
  - трудно се оптимизира хардуерно
    - много изчисления с плаваща запетая
    - сложно управление
    - сложен достъп до паметта за данните за сцената

RayTracing в реално време

- ***Паралелни софтуерни имплементации за клъстери с многопроцесорни възли***
  - OpenRT project (http://www.openrt.de)
    - рендиране на 5 дървета и 28 000 слънчогледа (35 000 триъгълника) на клъстер с 48 CPU

RayTracing в реално време

- ***Софтуерни имплементации за GPU***
  - трудности
    - GPU е традиционно специализиран за растеризиране
  - демо на HD ray tracing с 30fps от NVIDIA на SIGGRAPH 2010
    - 4 CUDA GPUs
    - три пъти по голяма производителност от SIGGRAPH 2008

  - най-бързото в момента решение за ray tracing в реално време

RayTracing в реално време

- ***Хардуерни имплементации за ray tracing***
  - специализиран чип с единствено предназначение за ray tracing
    - специализиран чип *RPU*
    - демонстриран на SIGGRAPH 2005
    - 66 MHz прототип с по-добра производителност от OpenRT на 2.66 GHz Intel Pentium 4

Сцена с 52 470 триъгълника от играта UT2003 рендирана с RPU

POV-Ray

- Пълно-функционален безплатен raytracer: povray.org
- Internet Ray Tracing Competition: irtc.org

КРАЙ

Следваща тема:

Визуализиране на текстури