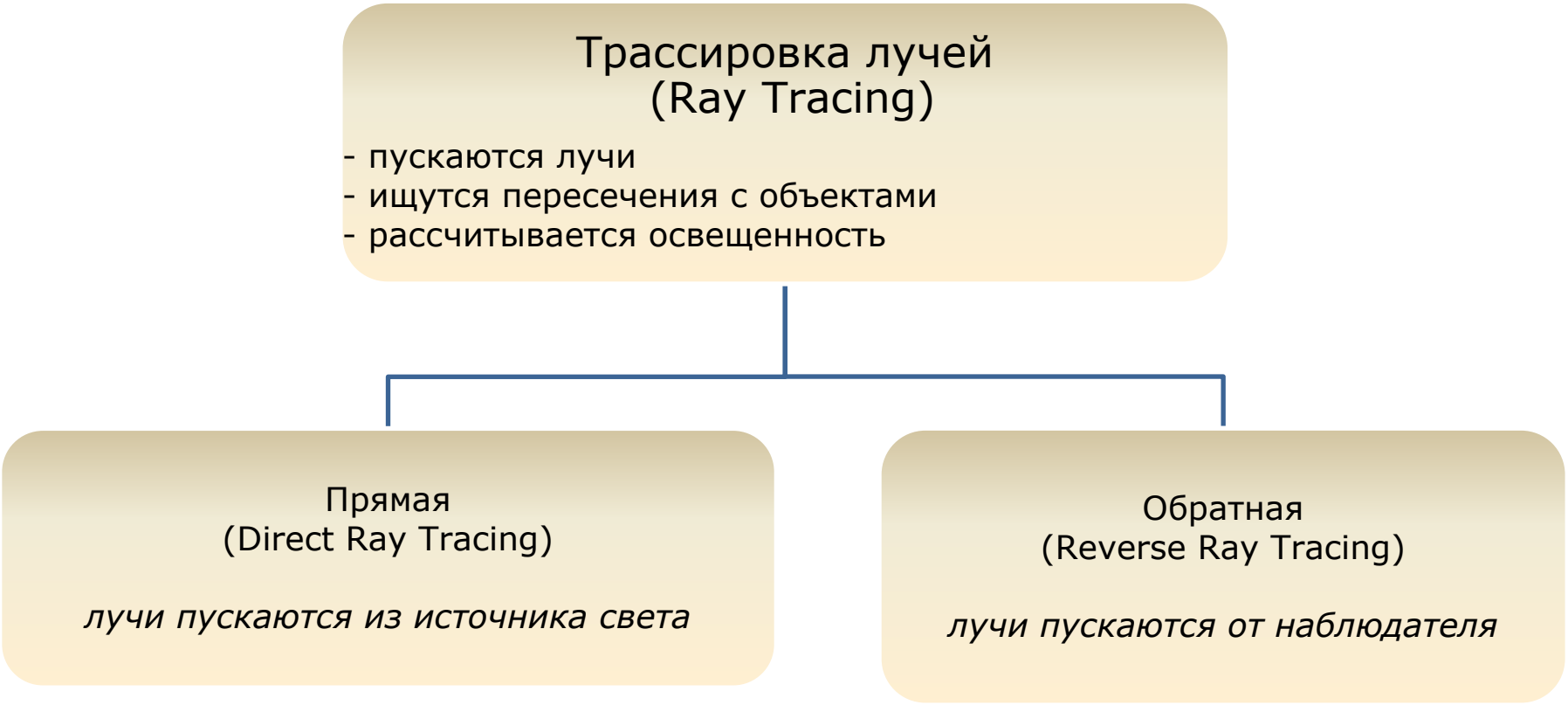
Интерактивная Компьютерная Графика

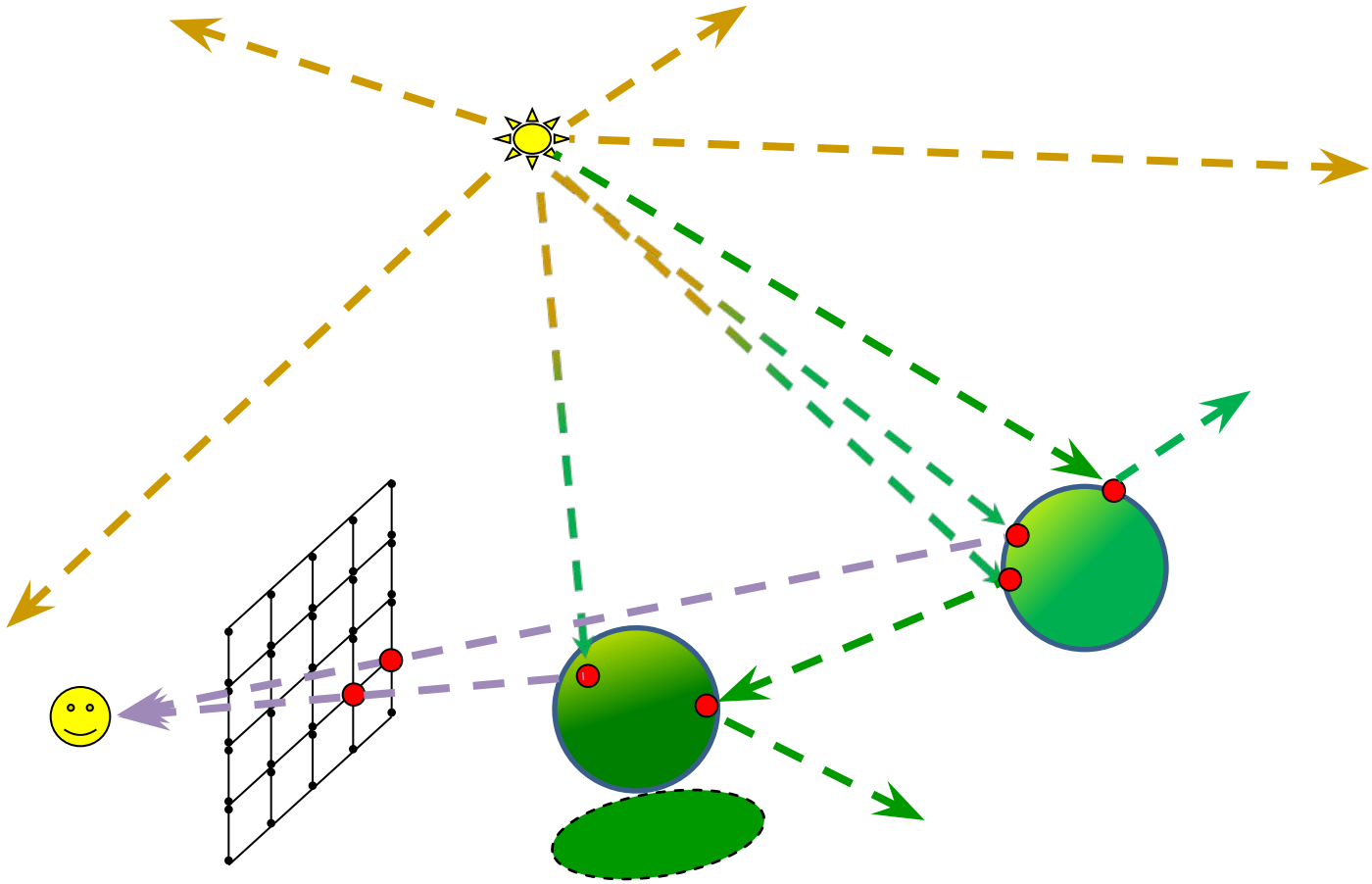
Часть 6-3

(трассировка лучей)

Трассировка лучей

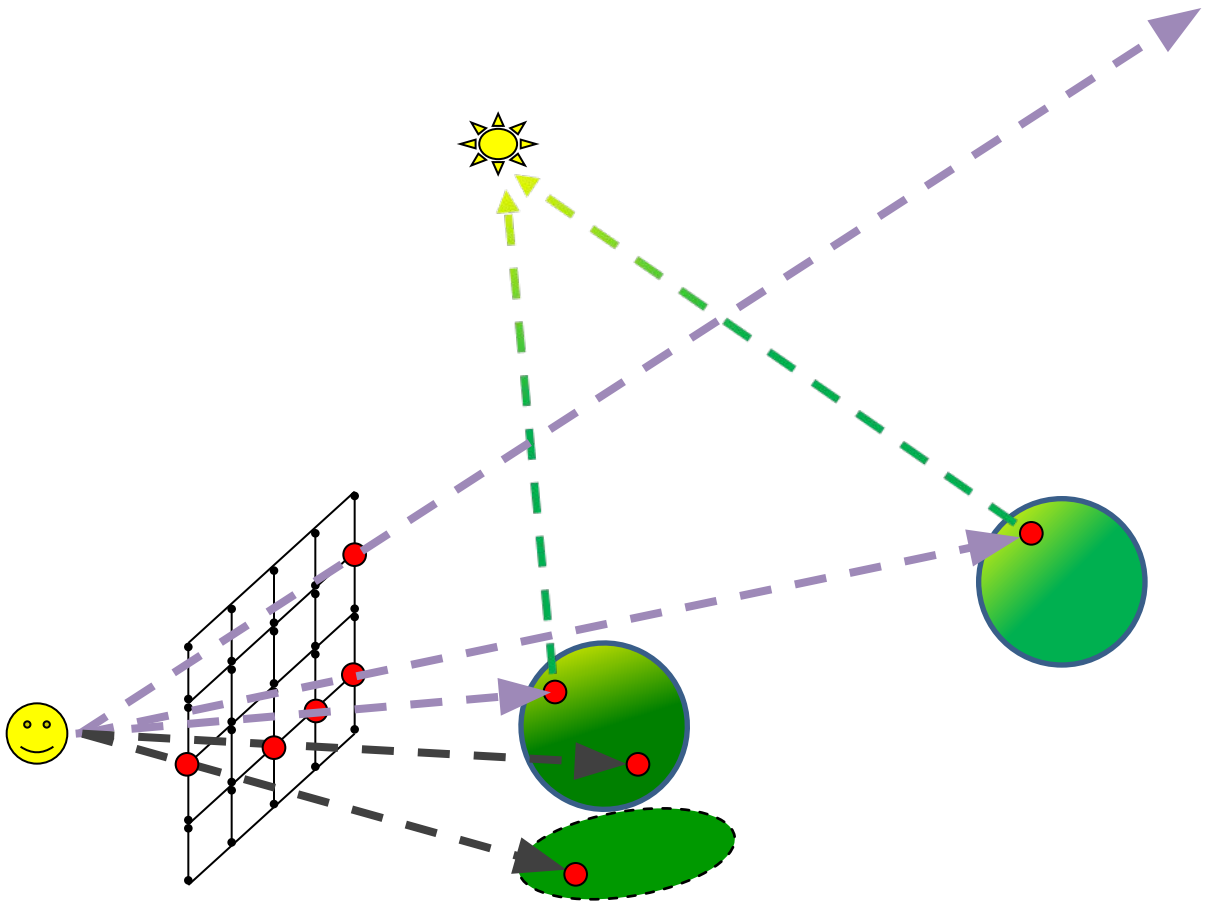


Прямая трассировка
(Direct Ray Tracing)



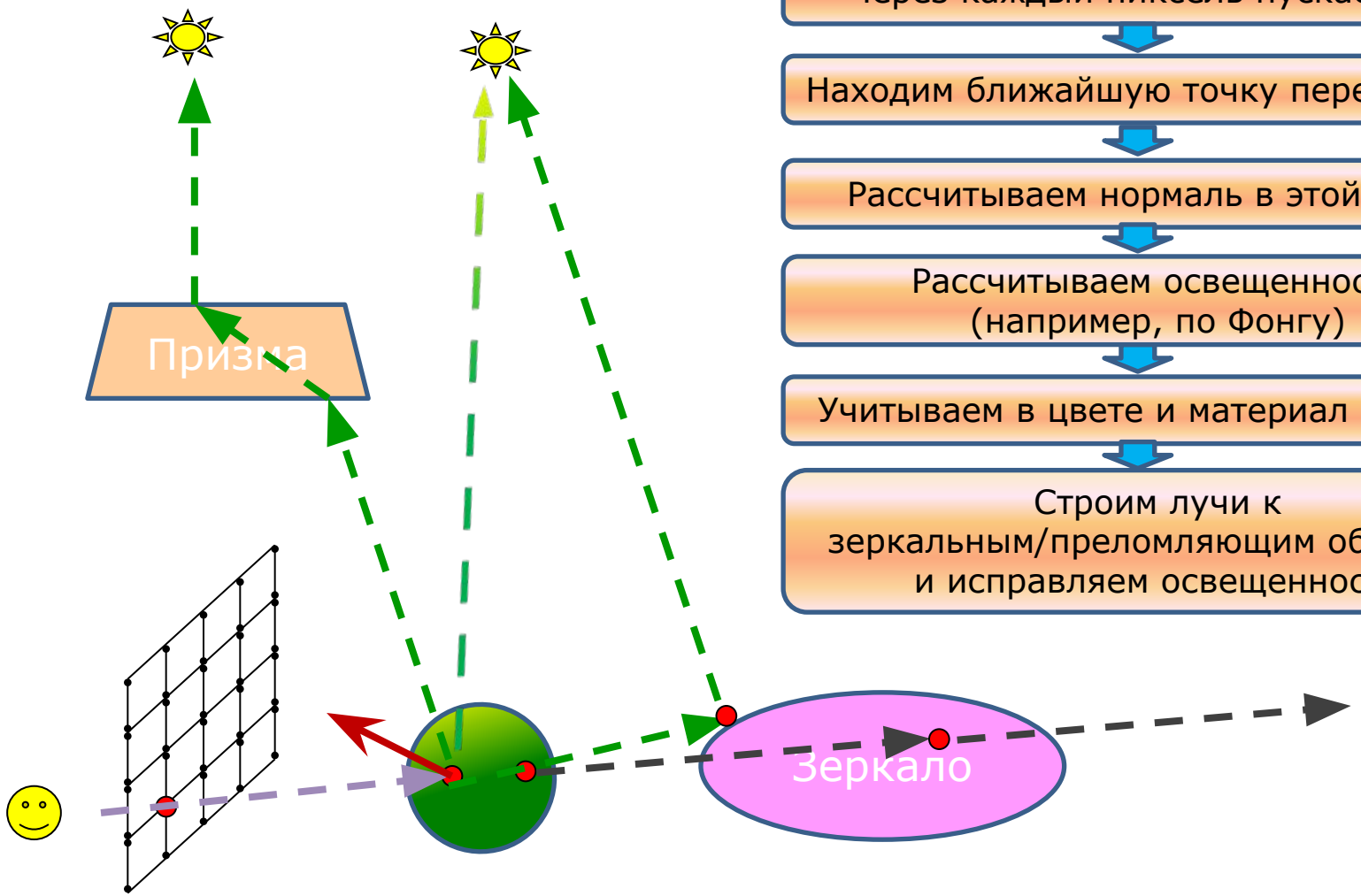
(-) много лишних лучей

Обратная трассировка
(Reverse Ray Tracing)



(+) просчет только нужных лучей

Обратная трассировка
(Reverse Ray Tracing)



Через каждый пиксель пускаем луч

Находим ближайшую точку пересечения

Рассчитываем нормаль в этой точке

Рассчитываем освещенность (например, по Фонгу)

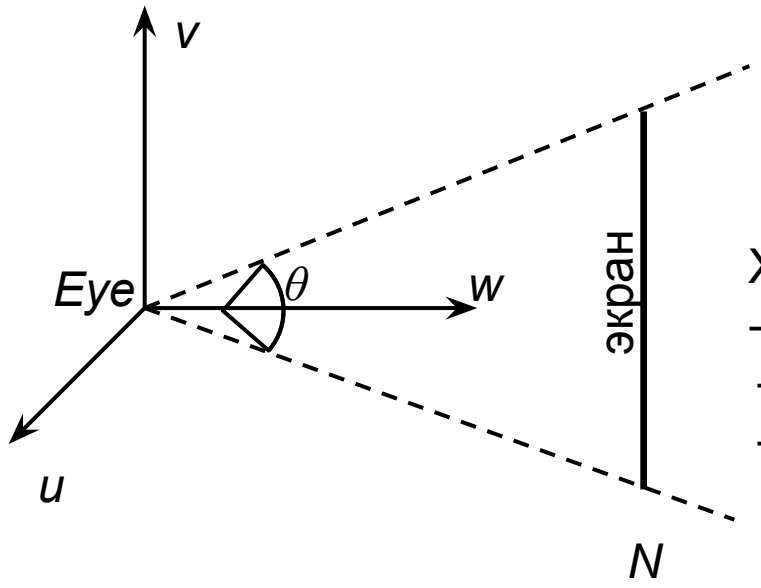
Учитываем в цвете и материал объекта

Строим лучи к зеркальным/преломляющим объектам и исправляем освещенность

Обратная трассировка
(минимальный код на визитке, Paul Heckbert)

```
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(){printf("%d %d\n",32,32);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!*/
```

Задание наблюдателя



Характеристики наблюдателя:
- положение: *Eye*
- система координат: (*u,v,w*)
- угол зрения: θ

Характеристики экрана:
- разрешение: n_c n_r
- форматное соотношение сторон: *Aspect*
- расстояние до экрана: *N*

Параметрическое уравнение луча:

$$r(t) = Eye + t \cdot Dir_{rc}$$

$$Dir_{rc} = N \cdot w + u_c \cdot u + v_r \cdot v$$

t<0: объект за наблюдателем

0<*t*<1 – объект перед ближней плоскостью отсечения

координаты (r,c)-пикселя

$$u_c = -W + W \cdot \frac{2 \cdot c}{n_c}$$

$$v_c = -H + H \cdot \frac{2 \cdot r}{n_r}$$

$$H = N \cdot tg\left(\frac{\theta}{2}\right)$$

$$W = H \cdot Aspect$$

Пересечение луча с объектом

Вариант задания объекта

Полигональной сеткой:

1. Ищем пересечение с плоскостью грани
2. Проверяем, что точка принадлежит многоугольнику

Аналитической неявной функцией *F(x,y,z)=0*:

1. Решаем уравнение

$F\left(x(t), y(t), z(t)\right) = 0$

относительно t

В каноническом случае:

$$F\left(M^{-1} \cdot r(t)\right) = 0$$

$$F\left(M^{-1} \cdot Eye + M^{-1} \cdot t \cdot Dir\right) = 0$$

Пересечение луча *r(t)*
со сферой

Общее уравнение сферы: $(x-x_0)^2+(y-y_0)^2+(z-z_0)^2=r^2$

Каноническое уравнение сферы: $x^2+y^2+z^2=1$

Параметрическое уравнение базовой сферы: $F(P)=|P|-1$

1. Подставляем уравнение луча: $|e+dt|-1=0$

2. Получаем квадратное уравнение: $ax+2bt+c=0$

$$где: a=(d,d),\ b=(e,d),\ c=(e,e)-1,$$

$$дискриминант\ s=b^2-4ac$$

3. Находим корни: $t=\begin{cases}\otimes,\ если\ s<0\ (нет\ пересечений)\\ -\dfrac{b}{a},\ если\ s=0\ (одно\ пересечение)\\ \dfrac{-b\pm\sqrt{s}}{a},\ если\ s>0\ (два\ пересечения)\end{cases}$

Пересечение луча *r(t)*
с плоскостью

Уравнение базовой плоскости (xy-плоскости): $F(x, y, z) = z$

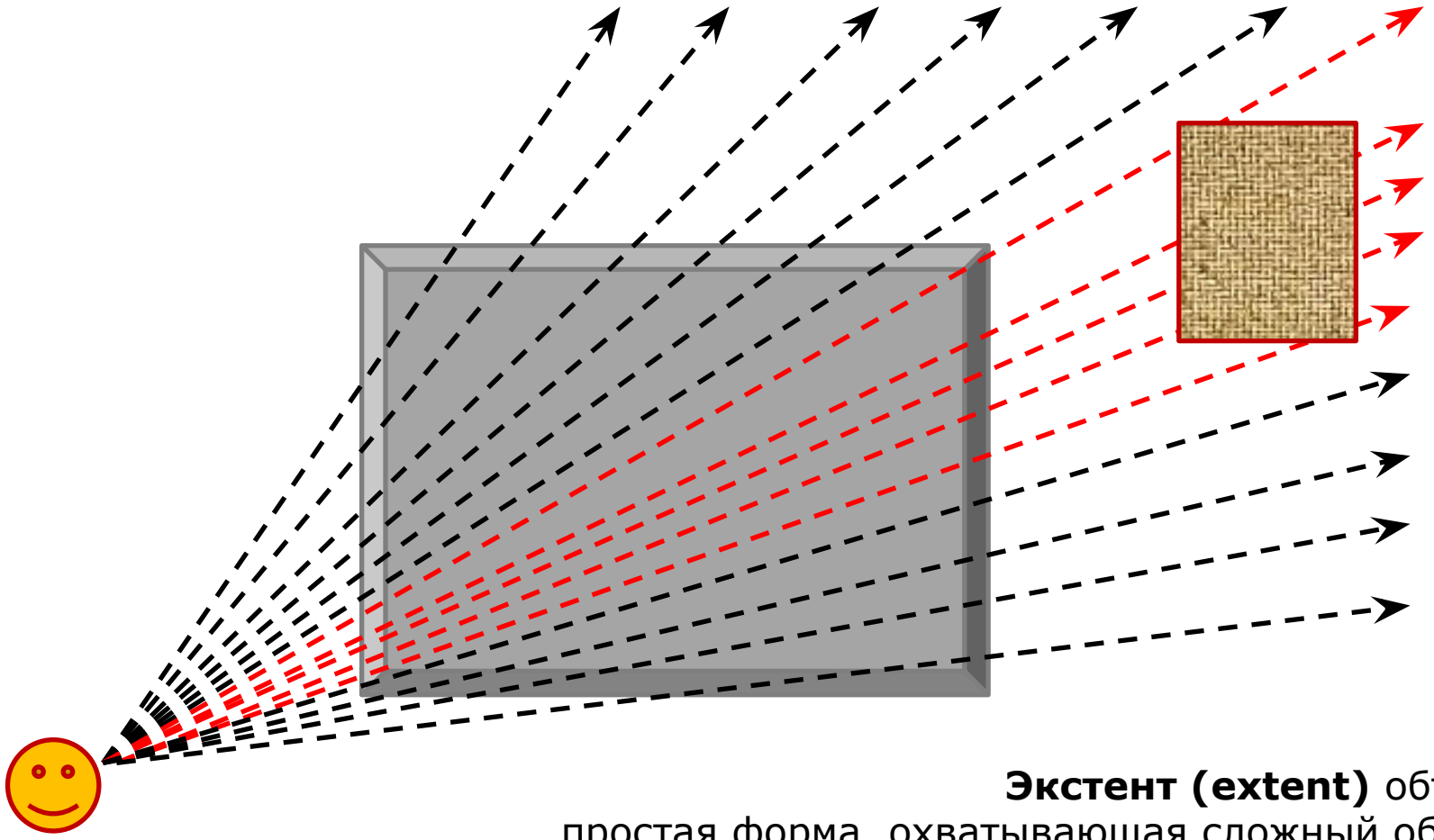
Луч *r(t)* пересекает плоскость z=0 когда: $e_z + d_z t = 0$

Решение уравнения:

$$t = \begin{cases} \otimes, \text{ если } d_z = 0 \ (\text{нет пересечений, луч параллелен плоскости}) \\ -\dfrac{e_z}{d_z}, \text{ если } d_z \neq 0 \end{cases}$$

Точка пересечения с плоскостью: $T = e - d\dfrac{e_z}{d_z}$

Использование экстентов



Экстент (extent) объекта
простая форма, охватывающая сложный объект,
с которым ищется пересечение только тогда,
когда луч пересекает экстент

Достоинства и недостатки
обратной трассировки лучей

(+) Корректно обрабатывает:

- ✔ тени
- ✔ отражения
- ✔ преломление
- ✔ полупрозрачные объекты

(+) Может работать с неполигональными объектами

(–) Затратность

Пример расчета в лесу с водой (особенности)

✔ Солнца скрыто за облаками → освещение почти полностью рассеянное

✔ Теней вообще нет (только темнота чащи леса)

✔ Рябь на воде

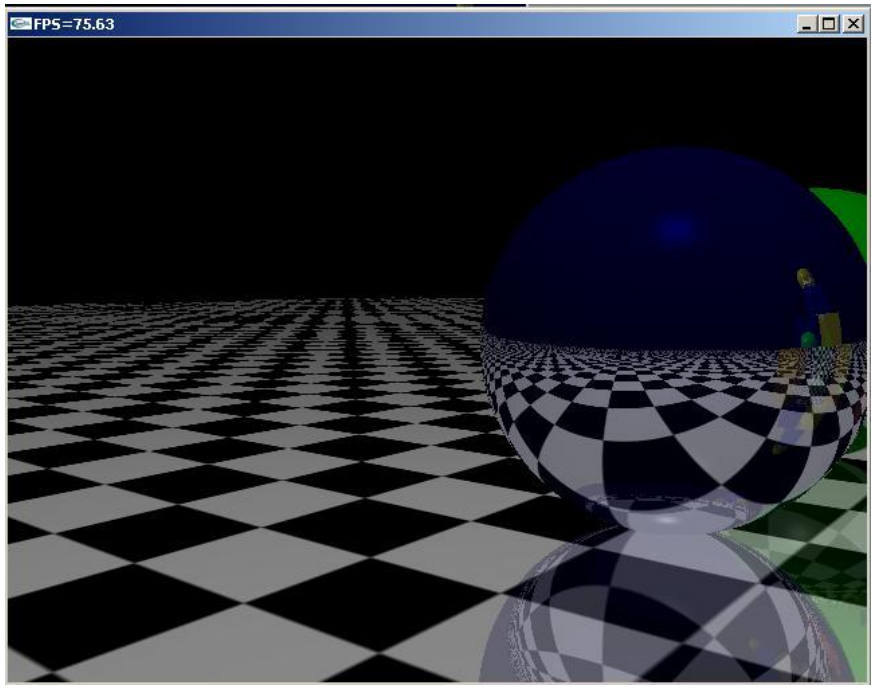
Пример расчета в лесу с водой
(стандартная растеризация)

1. Деревья:
   - ✔ модели стволов и основных веток изобразить примитивами
   - ✔ ветки изобразить спрайтами
   - ✔ дальние деревья скрыть туманом
2. Вода:
   - ✔ отрендерить сцену с точки зрения воды – и наложить как текстуру
   - ✔ рябь на воде сымитировать рельефной текстурой
3. Освещение:
   - ✔ Фоновое + диффузное

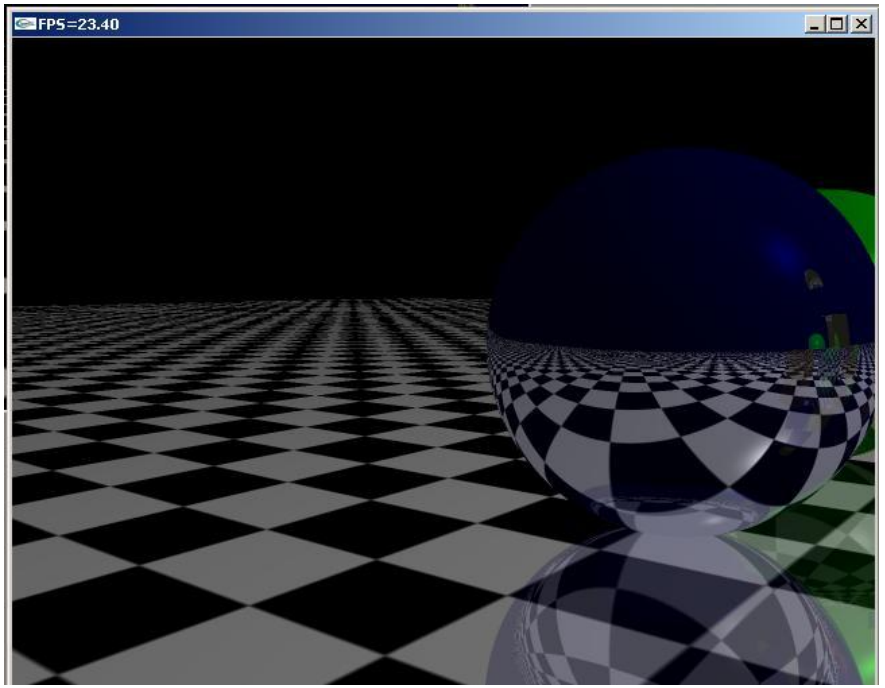
Пример расчета в лесу с водой
(трассировка лучей)

1. Деревья:
   - ✔ каждый луч пересекается с массой мелких веточек, листвой,…
2. Вода:
   - ✔ Из-за кривизны придется пускать много лучей через пиксель
3. Освещение:
   - ✔ нельзя провести луч к Солнцу и проверить на затенение
   - ✔ масса отражений/рассеиваний/преломлений от деревьев и воды

Пример сглаживания SSAA

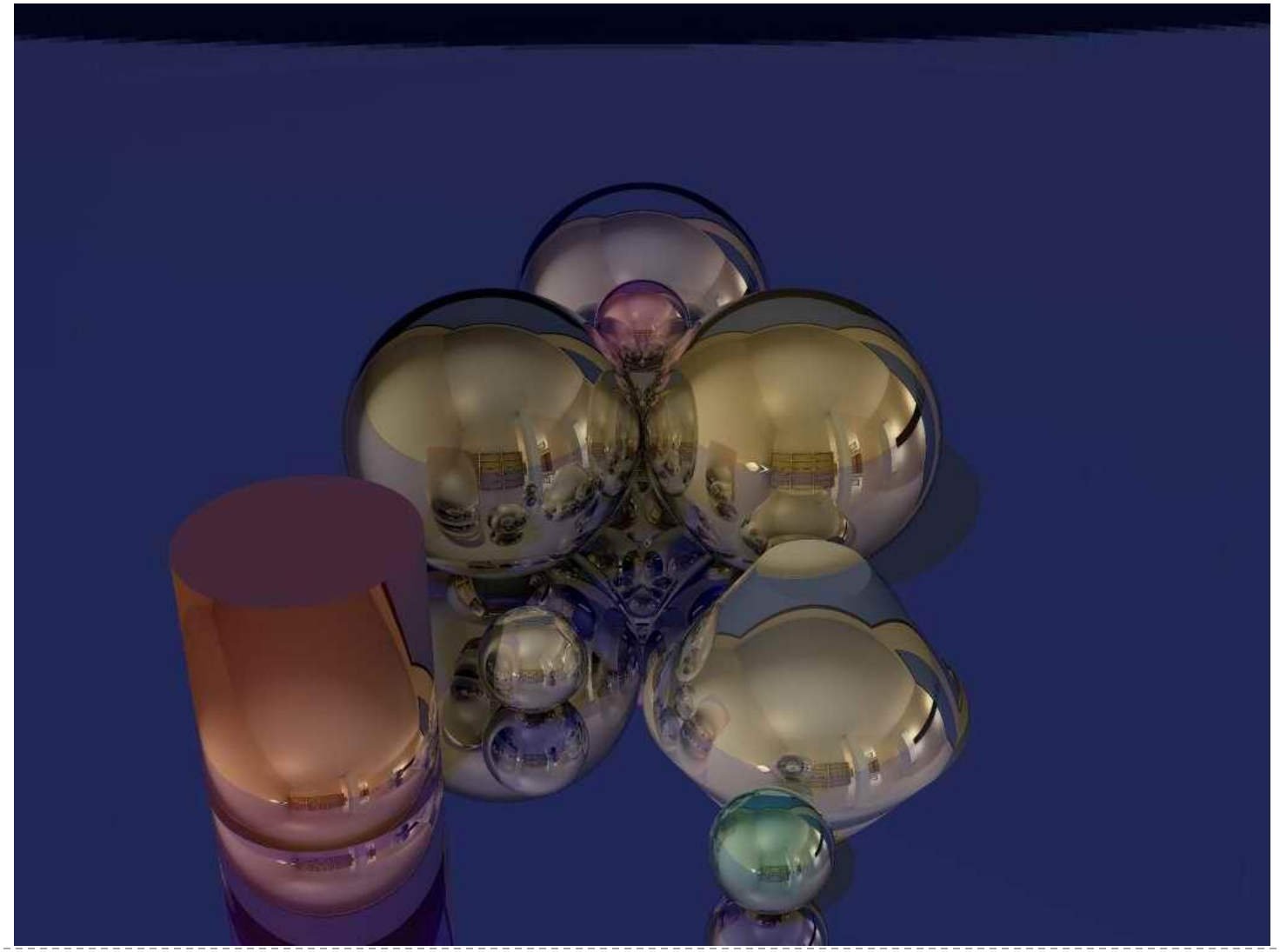


1 луч на пиксел
FPS = 75

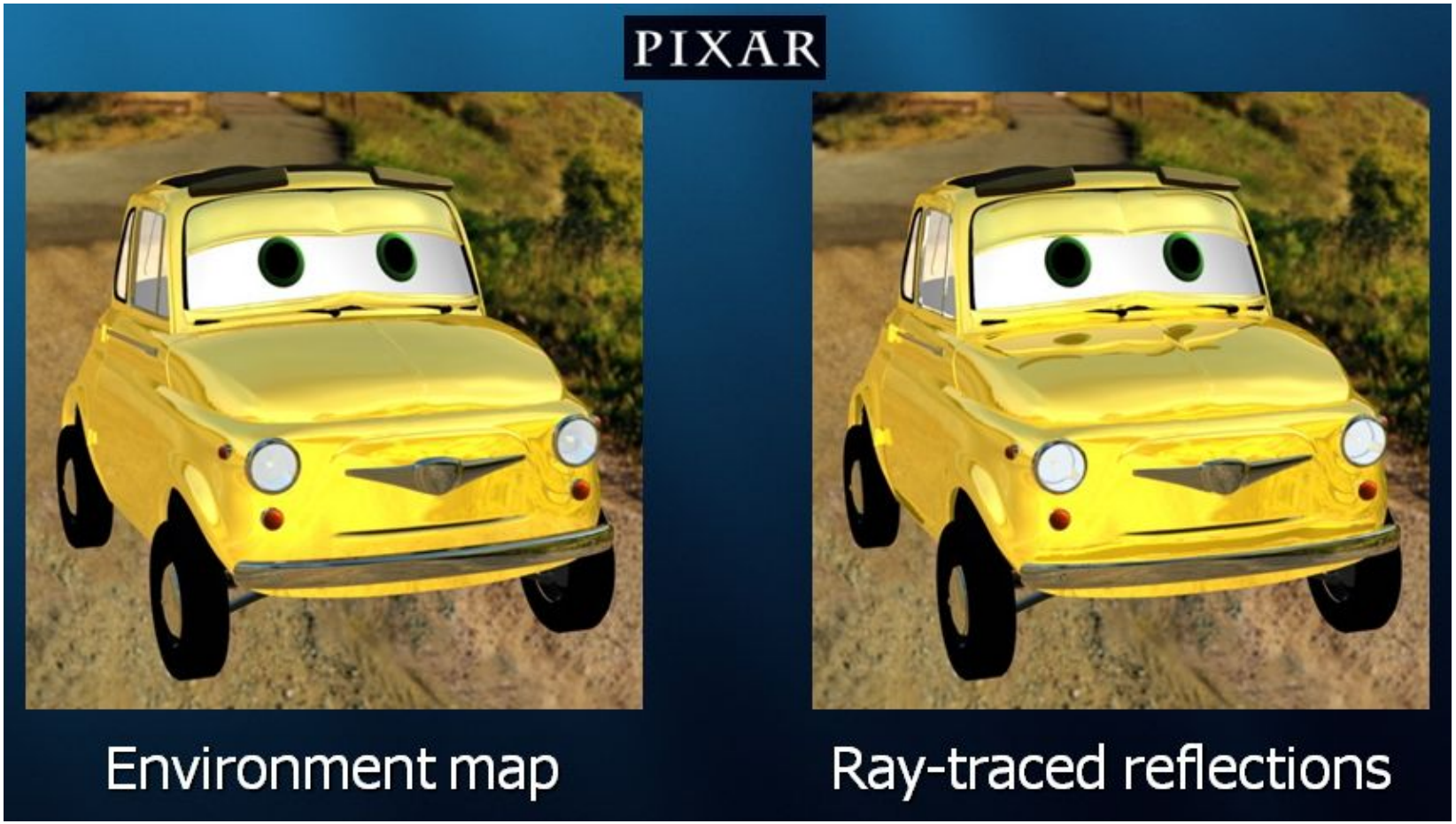


4 луча на пиксел
FPS = 24

Пример расчета отражения

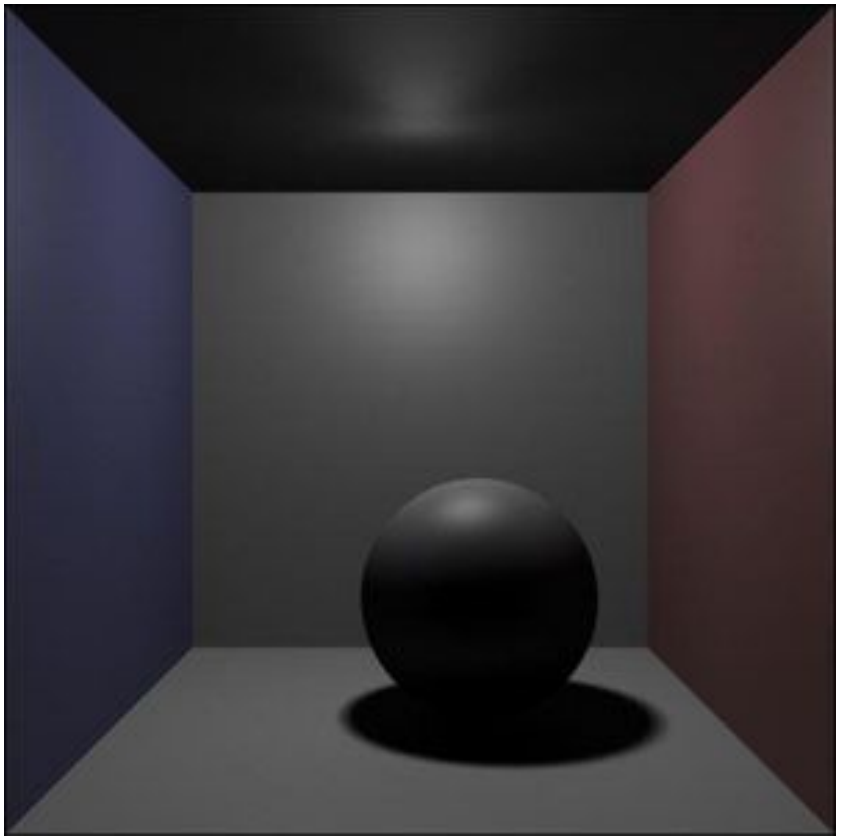


Пример расчета отражения
(Луиджи из м/ф «Тачки»)

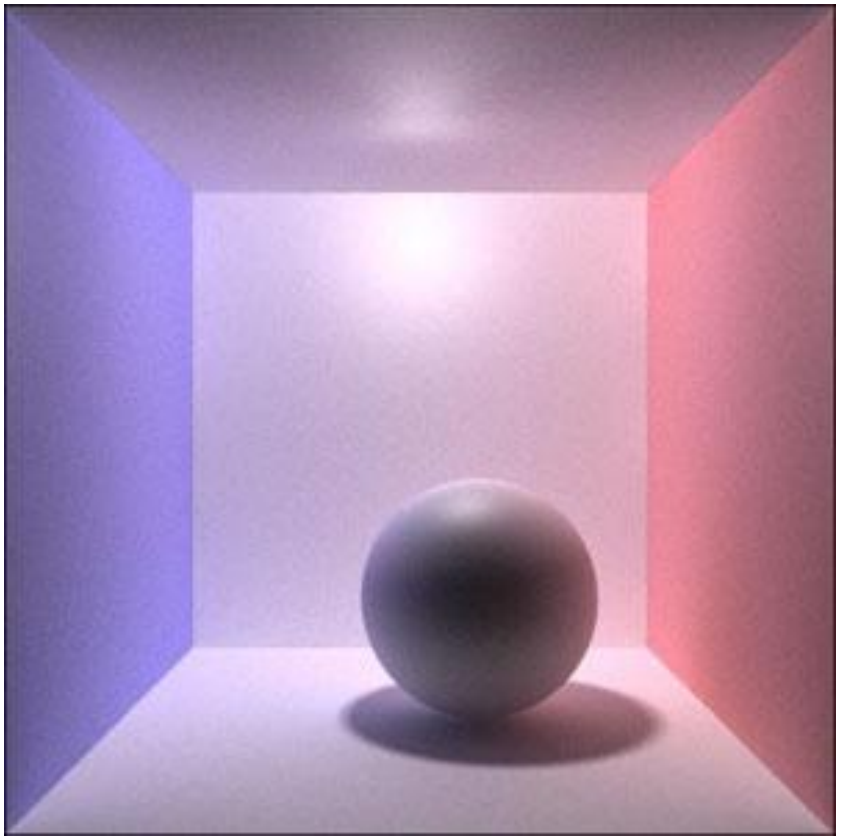


Студия Pixar использует *гибридный движок*
(карты окружения + трассировка лучей)
для учета отражения глаз на капоте и т.п.

Трассировка первичных лучей
(ray casting vs ray tracing)

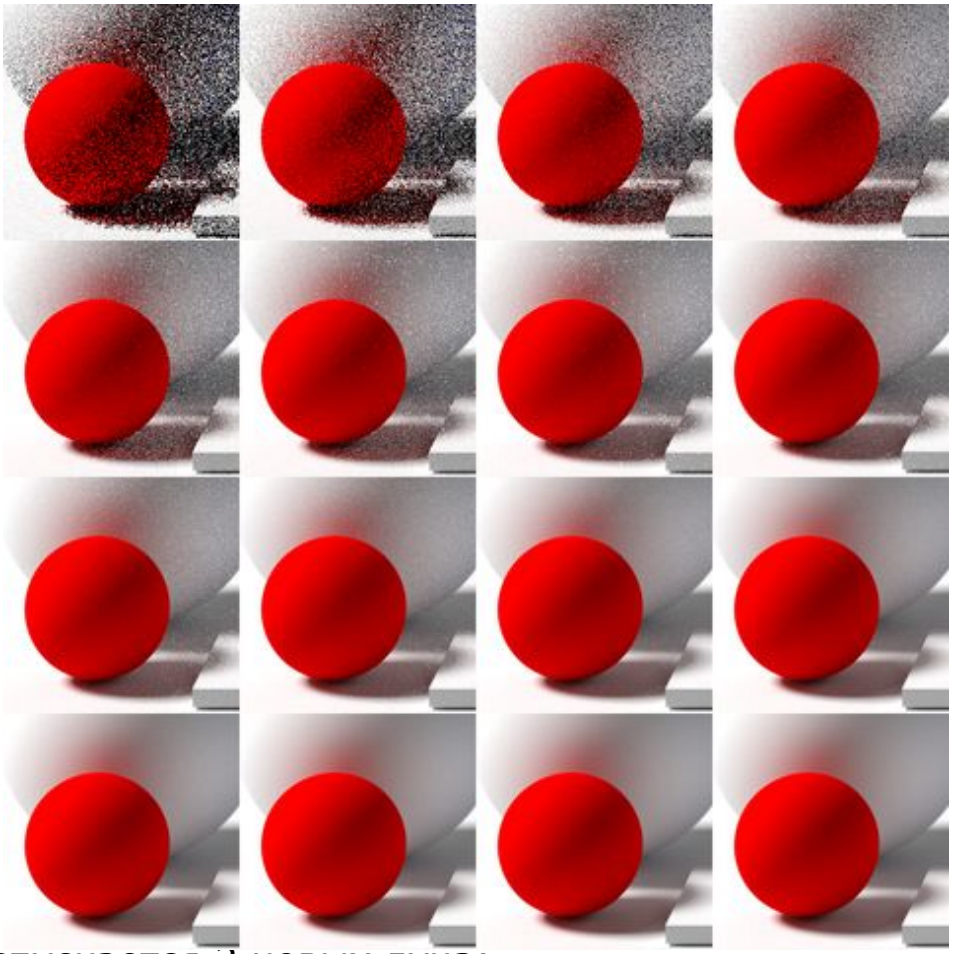


трассировка только первичных лучей
(пришедших напрямую от источника)



трассировка и вторичных лучей
(отраженных и преломленных)

Трассировка путей
(path tracing)



При попадании луча на поверхность испускается 2 новых луча:

1) напрямую к источнику света
2) в случайном направлении (не факт, что достигнет источника света)

Трассировка путей
(BDPT и MLT)

Bidirectional Path Tracing (испускает лучи одновременно от источника и из камеры
Metropolis Light Transport (учитывает значимость луча)

Излучательность
(radiosity)

Ограничения:

1. весь свет – диффузионный
2. система – замкнута (суммарная энергия – константа)

Алгоритм:

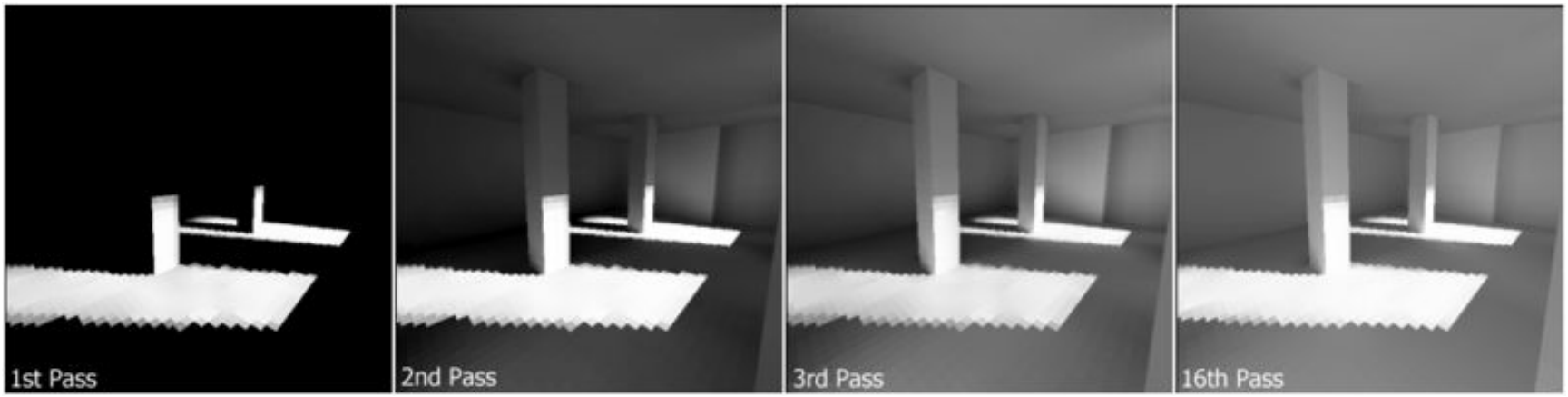
1. все поверхности сцены делятся на патчи (фрагменты, элементарные единицы)
2. для каждого патча итерационно вычисляется доля излученной и поглощенной энергии

$$\Delta \mathbf{B}_i = \rho_i \, \mathbf{B}_1 \, \mathbf{F}_{1i} \, \mathbf{A}_1 / \mathbf{A}_i$$

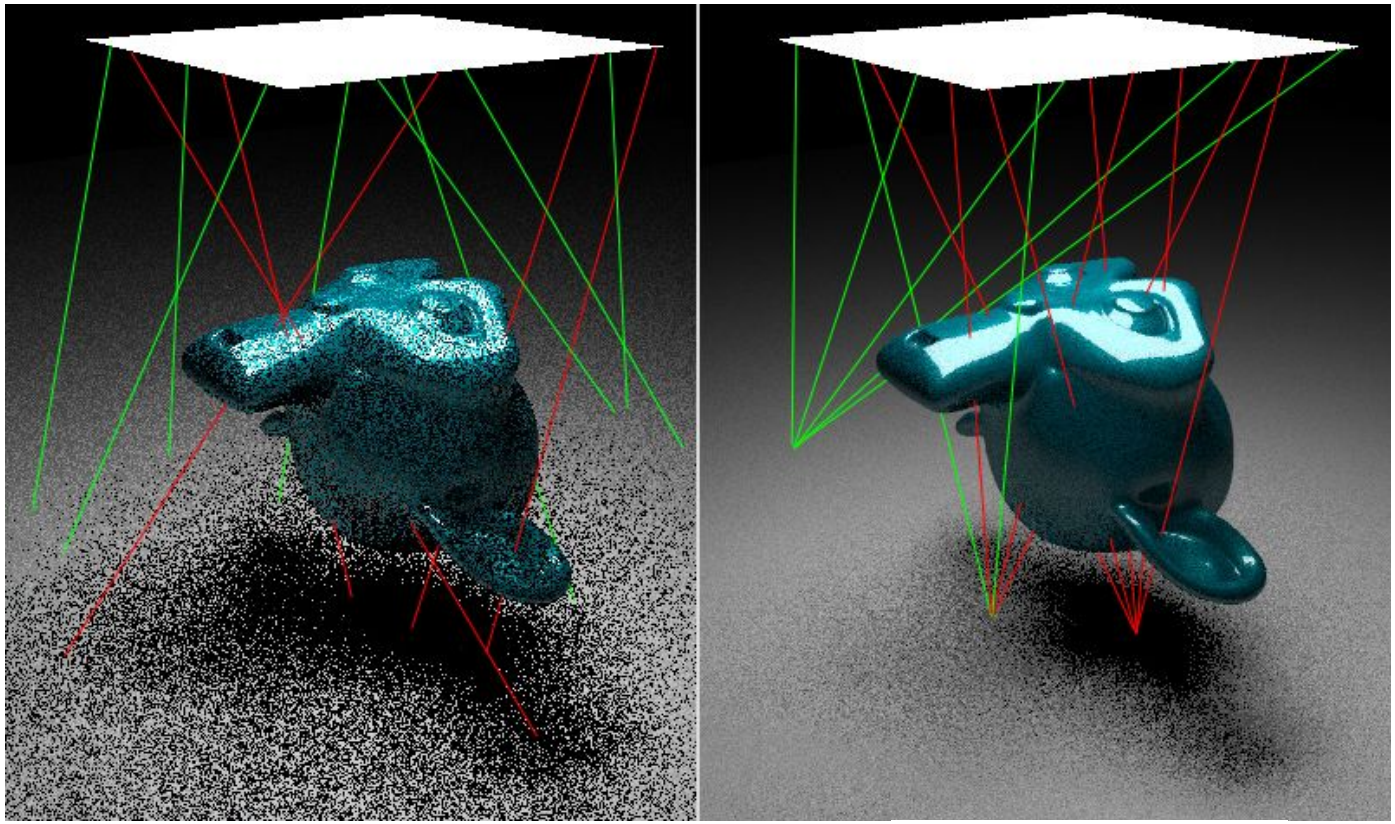
где $\Delta \mathbf{B}_i$ *- вклад в radiosity i-го патча от первого патча,*
ρ_i *- коэффициент, характеризующий свойства диффузного отражения поверхности, которой принадлежит i-й патч;*
$\mathbf{B}_1$ *- radiosity 1- го патча;*
$\mathbf{F}_{1j}$ *- форм-факторы первого патча в количестве n;*
$\boldsymbol{n}$ *- количество всех патчей трехмерной сцены;*
$\mathbf{A}_1$ *- площадь первого патча;*
$\mathbf{A}_i$ *- площади остальных патчей.*

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_i \cos\theta_j}{\pi r^2} HID_{ij} \, dA_j \, dA_i$$

$\boldsymbol{F_{ij}}$ *- форм-фактор i-го патча;*
$\boldsymbol{A_i}$ *,* $\boldsymbol{A_j}$ *- площади i- го и j-го патчей;*
$\boldsymbol{HID_{ij}}$ *- функция видимости патчей;*
$\boldsymbol{r}$ *- расстояние между патчами*
θi, θj *- углы между нормалями патчей и линией, их соединяющей*



Окружающие помехи
(ambient occlusion)

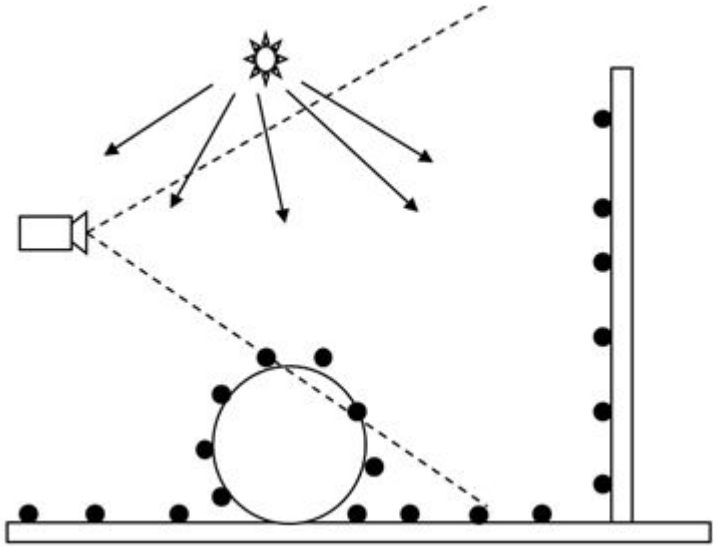


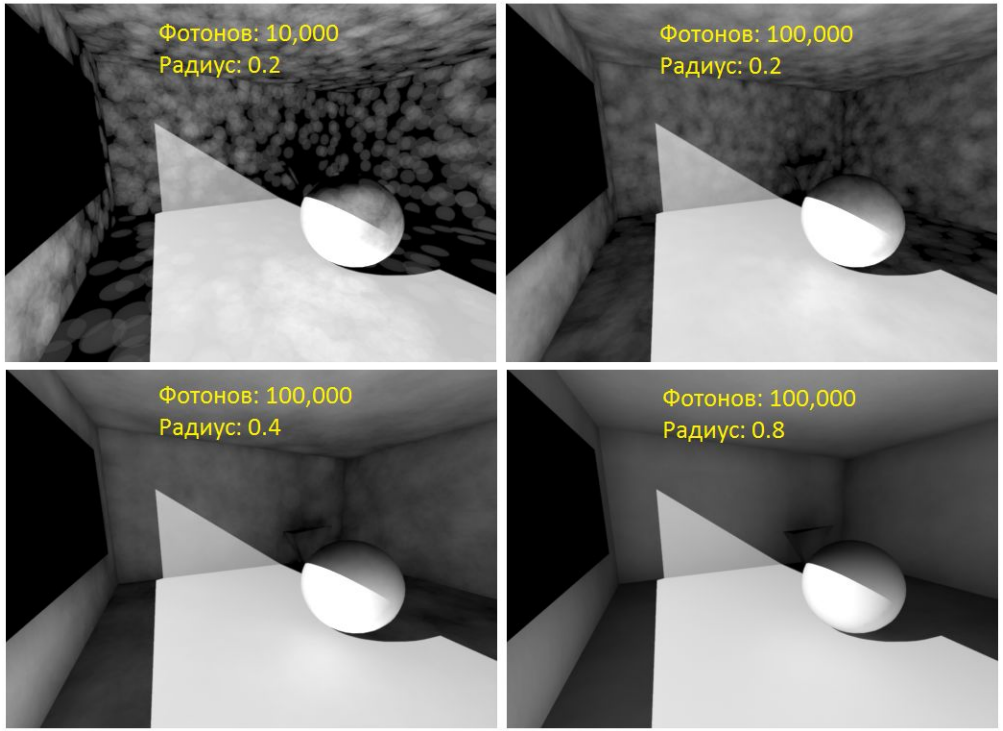
Лучи пускаются во всех направлениях по сфере: $A_p = \frac{1}{\pi} \int_{\Omega} V_{p,\omega} (N \cdot \omega) \, d\omega$

1) Лучи, достигнувшие фона («неба»), увеличивают яркость на поверхности
2) Лучи, пересекающие другие объекты, не добавляют яркости

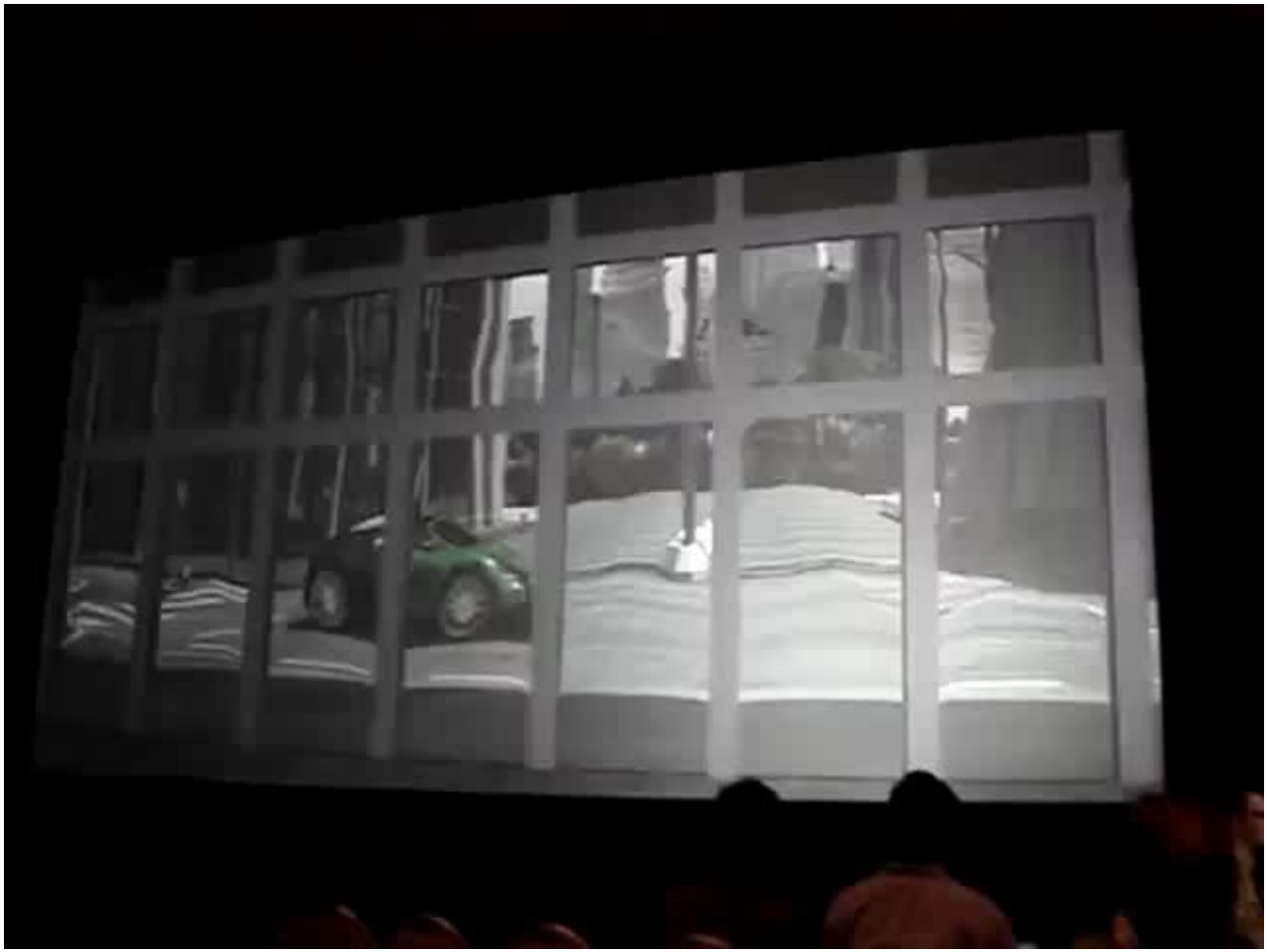
Фотонные карты
(Photon Mapping)

1. Испускаются лучи (фотоны) от источника
   - при столкновении с поверхностями, фотоны *отдают* часть своей энергии и *отражаются* в некотором направлении
   - информация об энергии *сохраняется* в *фотонной карте*
2. Испускаются лучи из камеры
   - при попадании луча на поверхность интенсивность рассчитывается через ближайших значений в фотонной карте





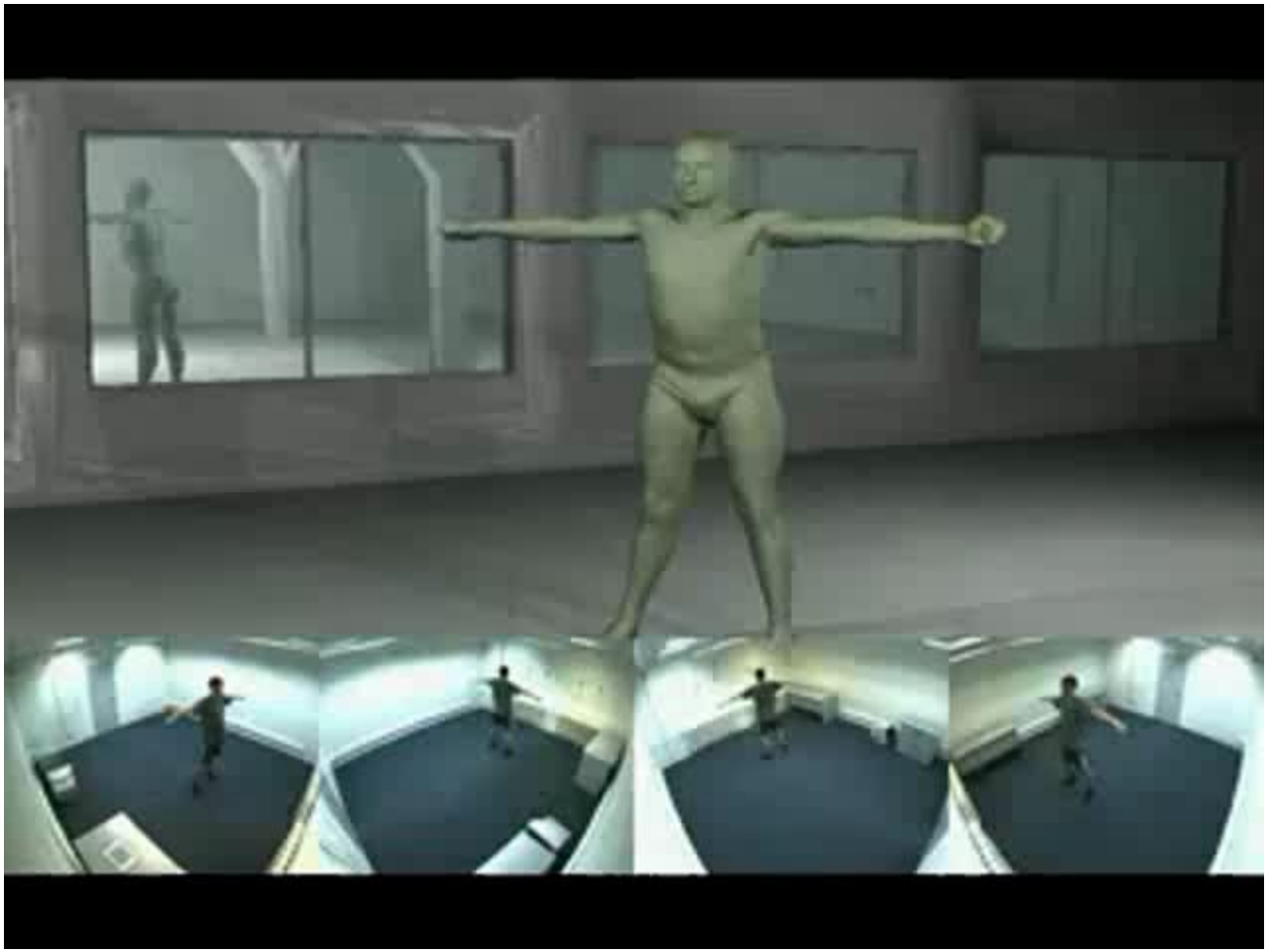
Примеры в режиме реального времени (от NVidia)



только один автомобиль (Bugatti)

Демонстрация в 2009 г. от Nvidia в режиме real-time на совокупности нескольких четырёхпроцессорных профессиональных карт

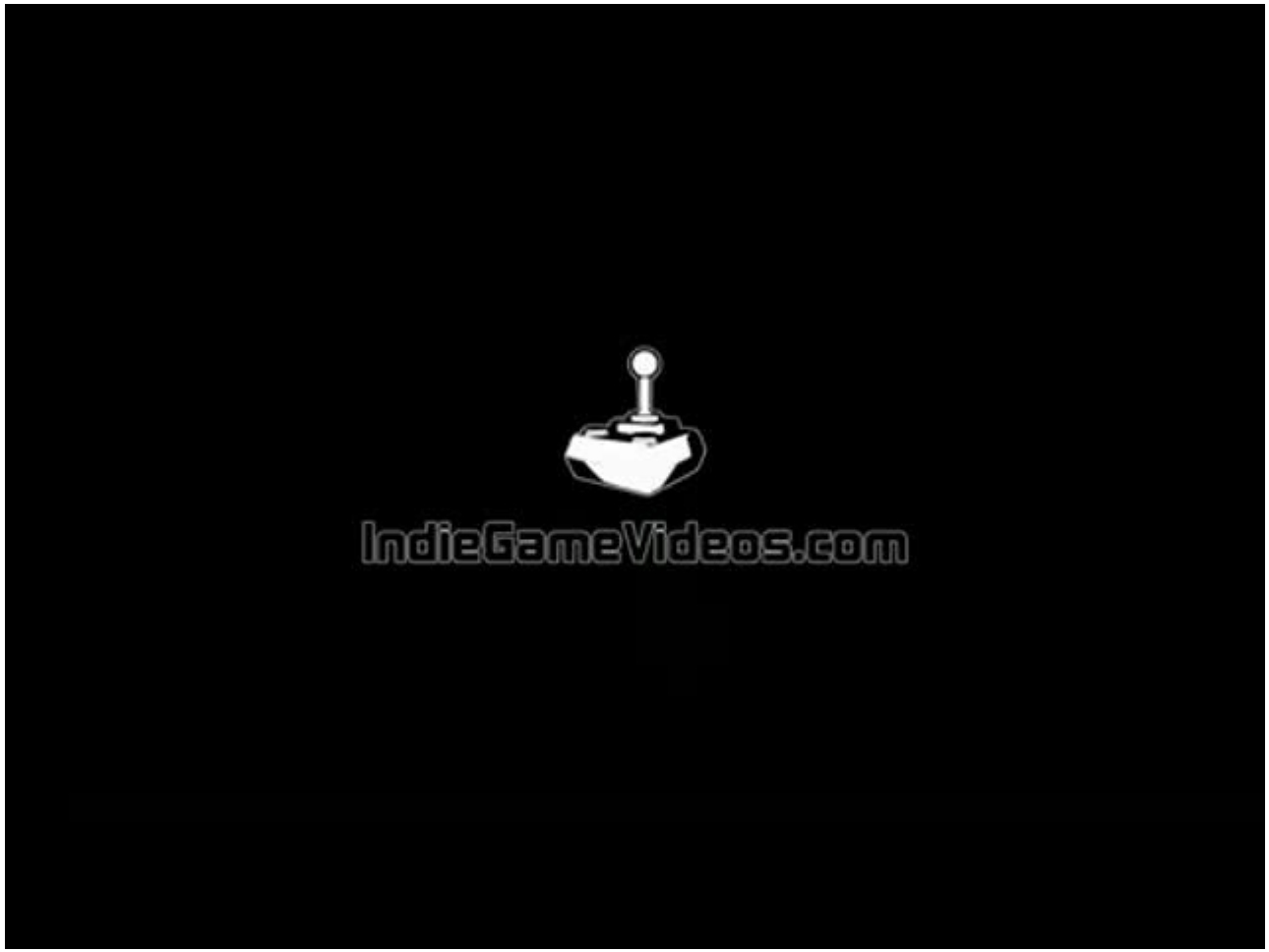
Примеры в режиме реального времени (от Intel)



рассеивание в приповерхностном слое кожи
+
неровность колжи
+
волоски

Демонстрация в 2010 г. от Intel в режиме real-time
на совокупности систем большой мощности

Примеры в режиме реального времени
(AntiPlanet2 - мир сфер)



Динамические:

✔освещение
✔тени
✔прозрачность

2x ядерный процессор + GeForce GTX + Nvidia CUDA