Методы расчета интеграла освещенности на основе трассировки лучей

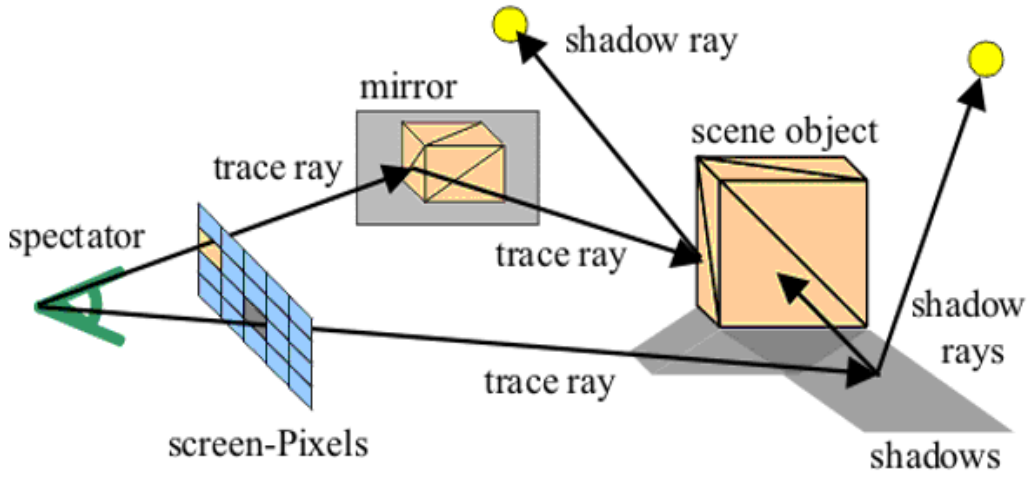
Фролов В.А., Игнатенко А.В.

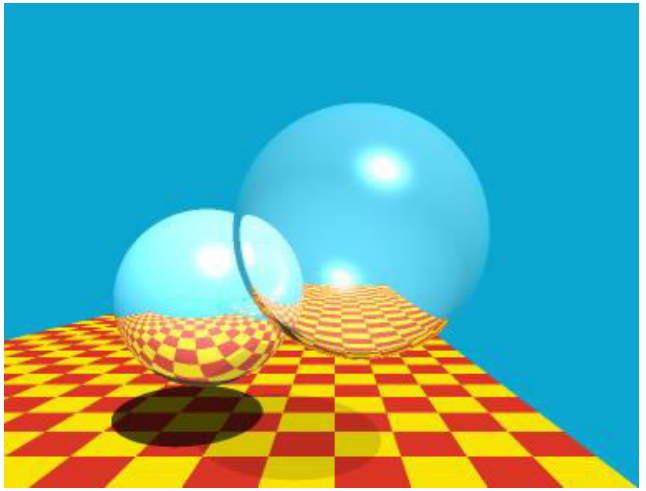
План лекции

- Трассировка лучей
  - Структуры пространственного разбиения
- Интеграл освещенности
  - Метод Монте-Карло, biased/unbiased методы
- Стохастическая трассировка лучей
  - **PT**, LT, BDPT, MLT, ERPT
  - Стратегии сэмплирования – стохастическая и детерминированная
  - MIS
- Фотонные карты
  - **PPM**
- Оптимизации
  - Кэш освещенности (Irradiance cache)
  - Параллелизм

Трассировка лучей

- “Whitted-style Ray Tracing”

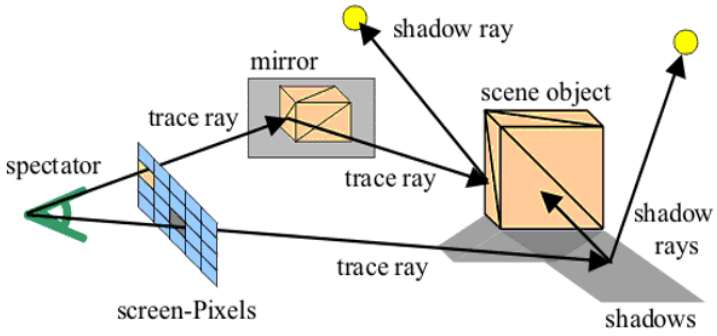




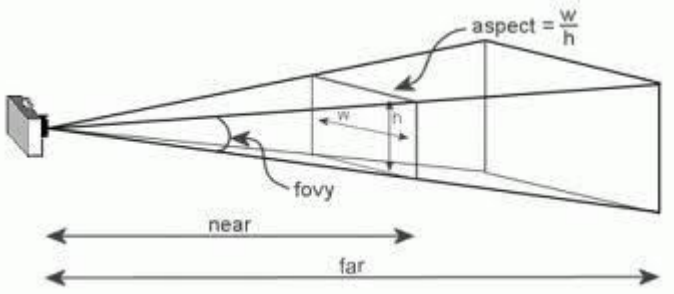
Трассировка лучей

- Генерация лучей

```
float3 EyeRayDir(float x, float y, float w, float h)
{
  float fov = 3.141592654f/(2.0f);
  float3 ray_dir;

  ray_dir.x = x+0.5f - (w/2.0f);
  ray_dir.y = y+0.5f - (h/2.0f);
  ray_dir.z = -(w)/tan(fov/2.0f);

  return normalize(ray_dir);
}
```




```
float3 EyeRayDir2(float x, float y, float w, float h)
{
  float4 pos( 2.0f * (x + 0.5f) / w - 1.0f,
              -2.0f * (y + 0.5f) / h + 1.0f, 0.0f, 1.0f );

  pos = g_mViewProjInv*pos; // (mView*mProj)^-1*pos
  pos /= pos.w;

  pos.y *= (-1.0f);

  return normalize(pos.xyz);
}
```



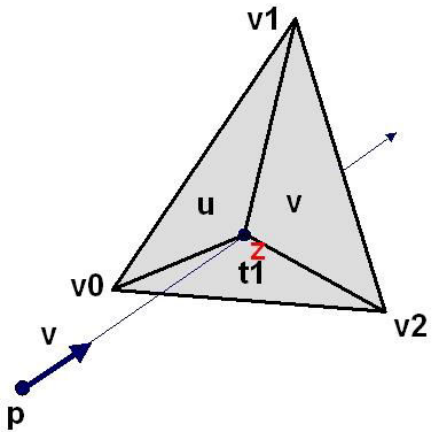
Трассировка лучей

- Пересечение лучей и примитивов

u := u/S

v := v/S

t1 := t1/S (t1 = 1 – u – v)



$$z(u,v) = (1-u-v)*v1+u*v2+v*v0$$

$$z(t) = p+t*d$$

$$p+t*d = (1-u-v)*v1+u*v2+v*v0$$

Трассировка лучей

- Пересечение лучей и примитивов

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{dot(P, E1)} * \begin{bmatrix} dot(Q, E2) \\ dot(P, T) \\ dot(Q, D) \end{bmatrix}$$

$$E1 = v1 - v0$$

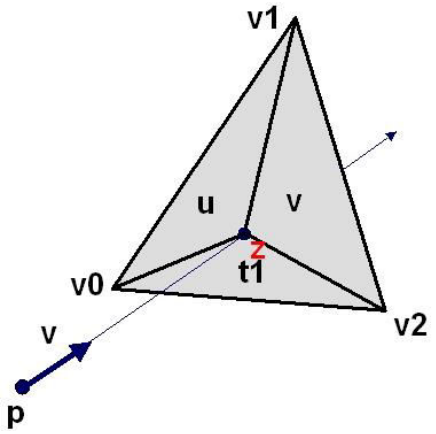
$$E2 = v2 - v0$$

$$T = p - v0$$

$$P = cross(D, E2)$$

$$Q = cross(T, E1)$$

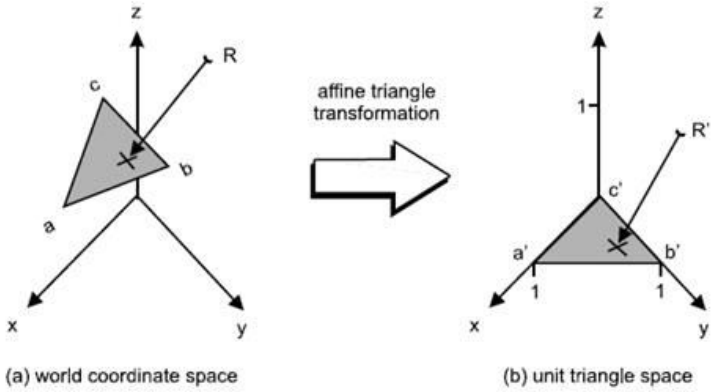
$$D = v$$



Трассировка лучей

- Пересечение лучей и примитивов
  - “unit test”

```
float3 o = mul3x4(m, ray_origin);
float3 d = mul3x3(m, ray_direction);

float  t = -o.z/d.z;
float  u = o.x + t*d.x;
float  v = o.y + t*d.y;
```



(a) world coordinate space (b) unit triangle space

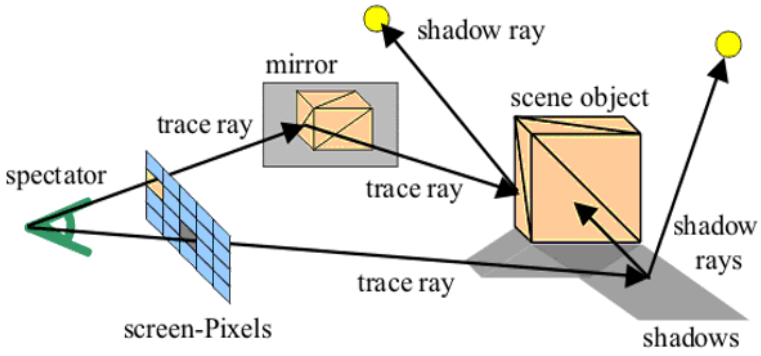
$$T_\Delta^{-1}(X) = \begin{pmatrix} A_x - C_x & B_x - C_x & N_x - C_x \\ A_y - C_y & B_y - C_y & N_y - C_y \\ A_z - C_z & B_z - C_z & N_z - C_z \end{pmatrix} \cdot X + \begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix}$$

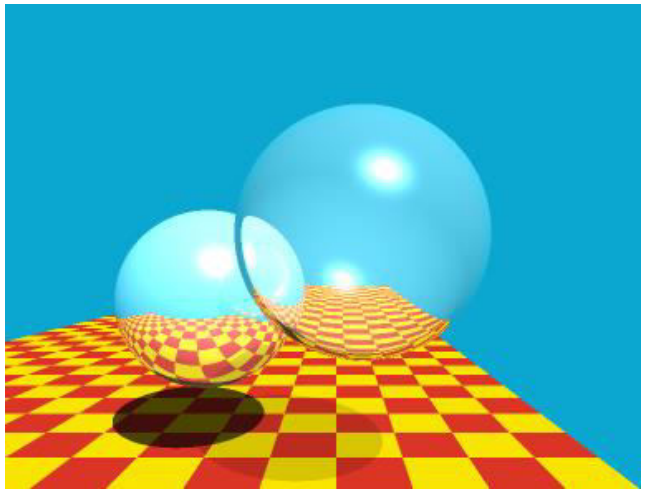
$$T_\Delta^{-1}\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = A \quad T_\Delta^{-1}\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = B$$

$$T_\Delta^{-1}\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = C \quad T_\Delta^{-1}\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = N$$

Трассировка лучей

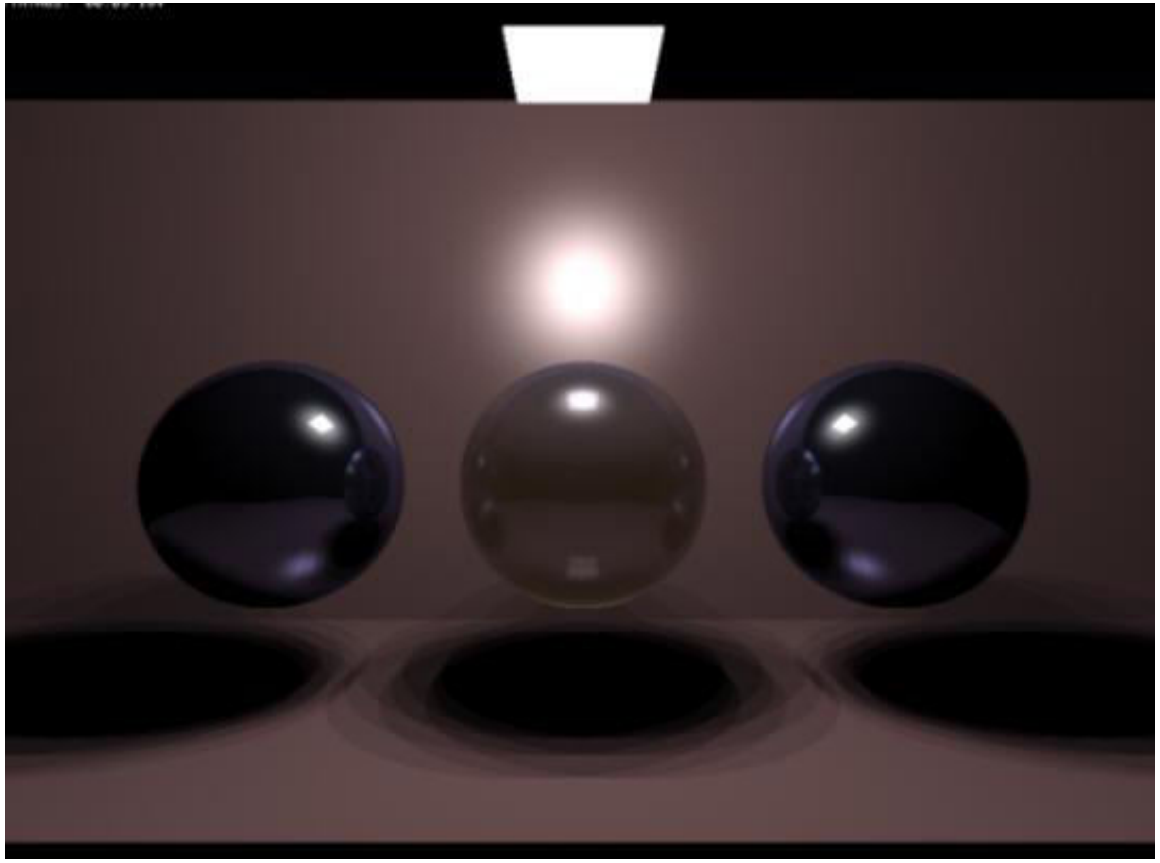
- Затенение (shading)
- color += I/(R*R);
- ?
- Отражения/преломлени

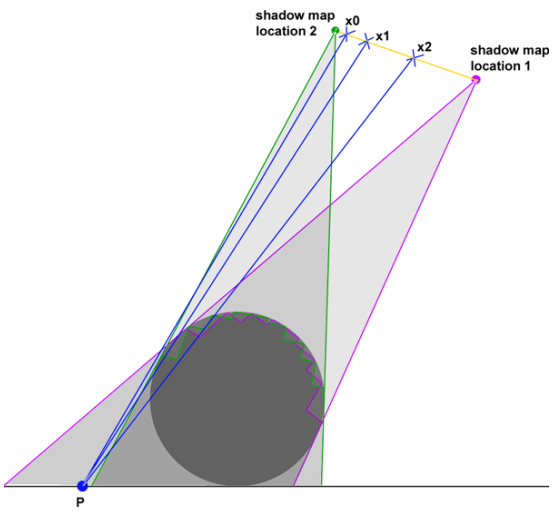


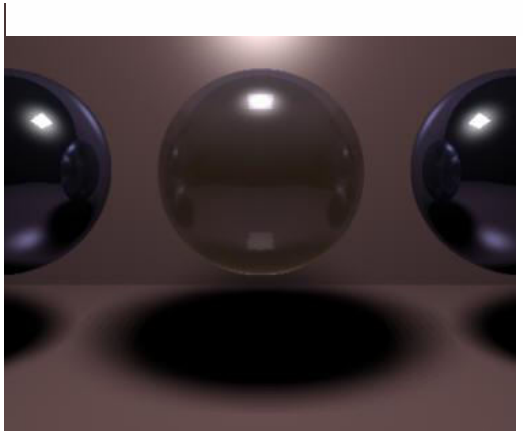


Трассировка лучей

- Мягкие тени

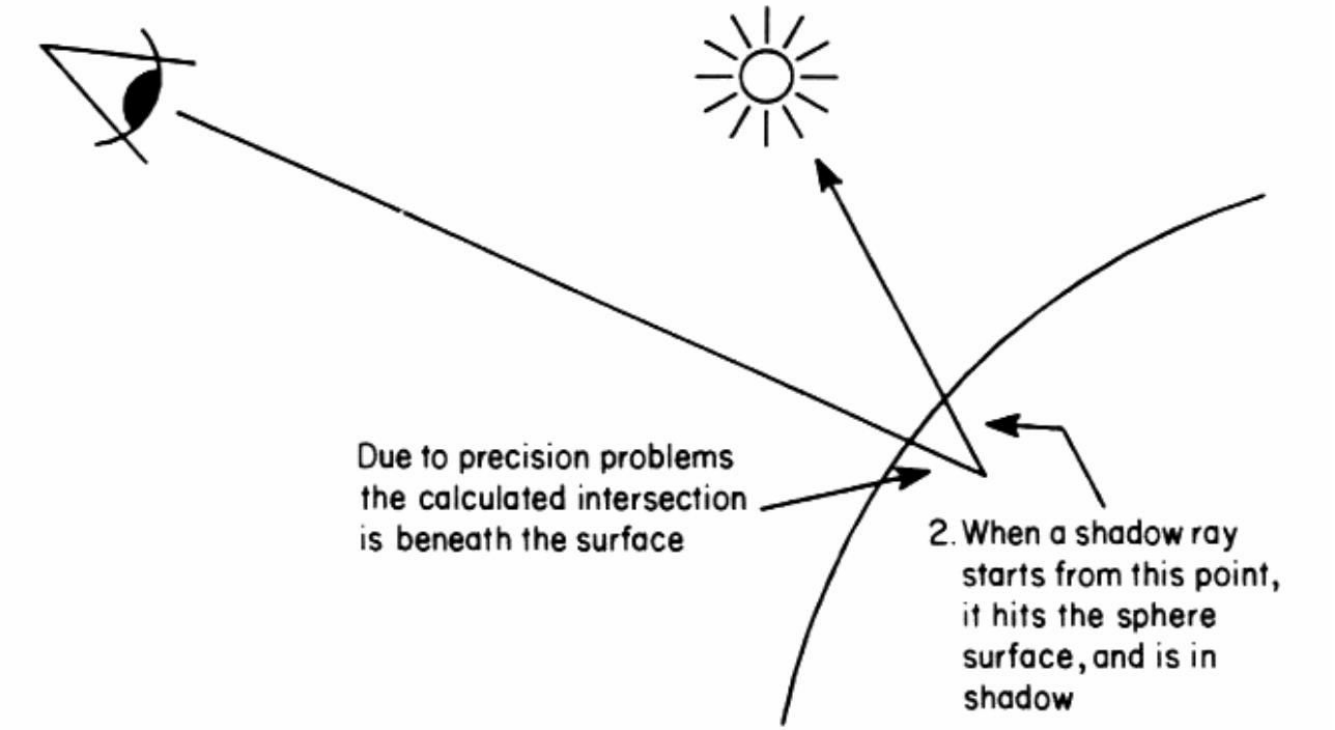






Трассировка лучей

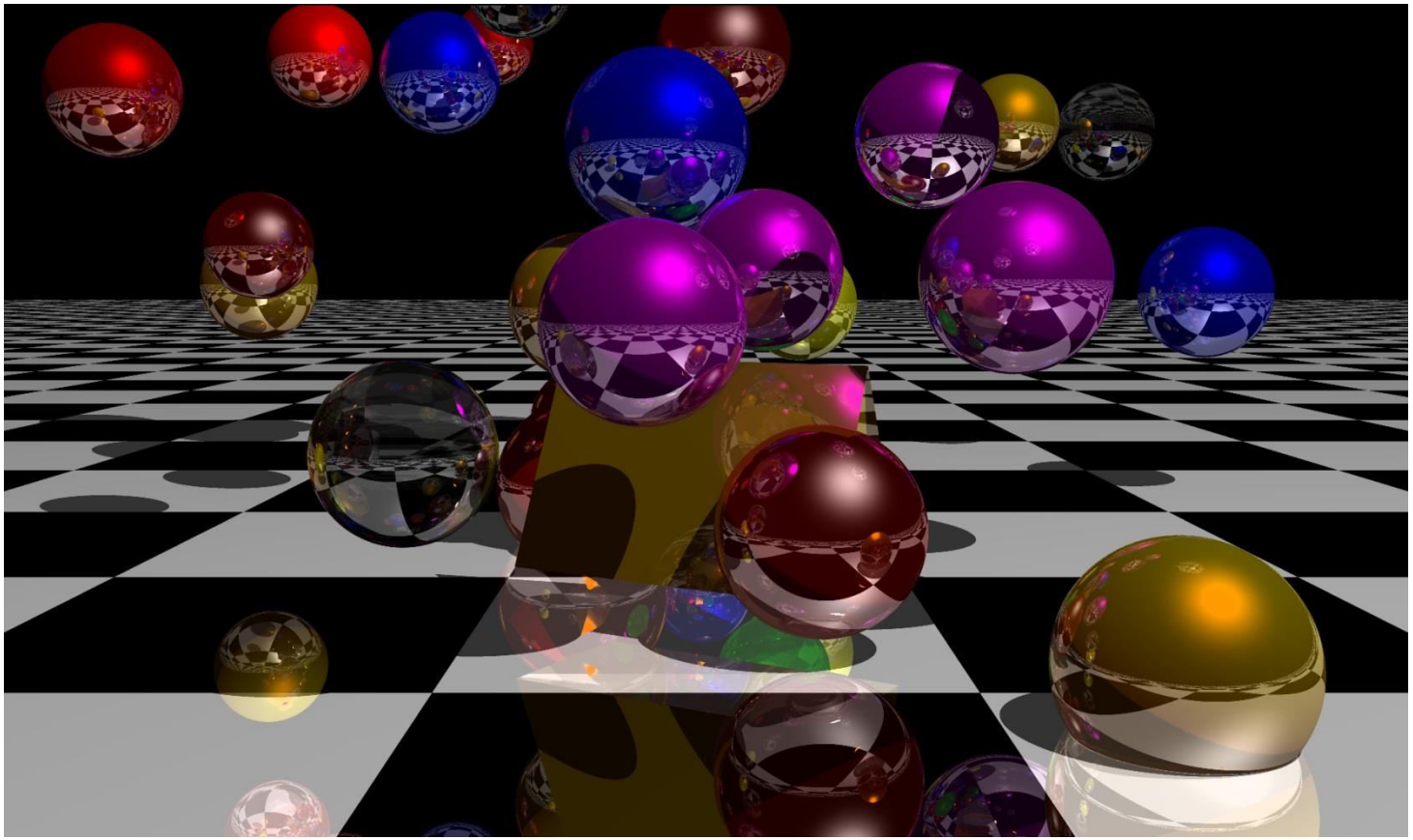
- Проблема точности



Problem in surface intersection.

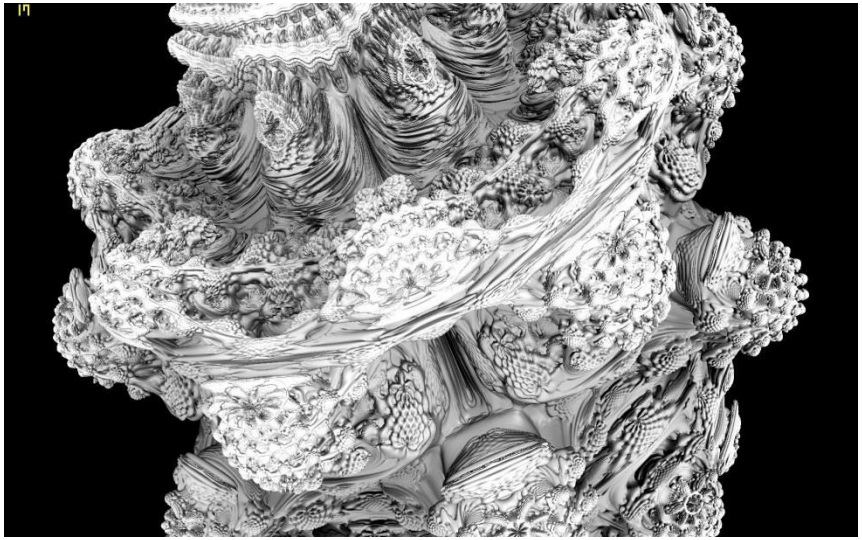
Трассировка лучей

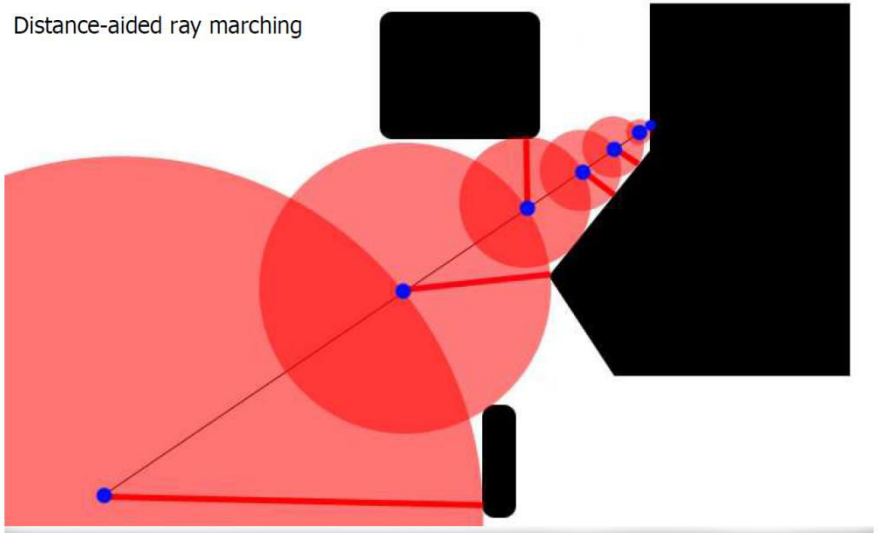
- “Whitted Ray Tracing”



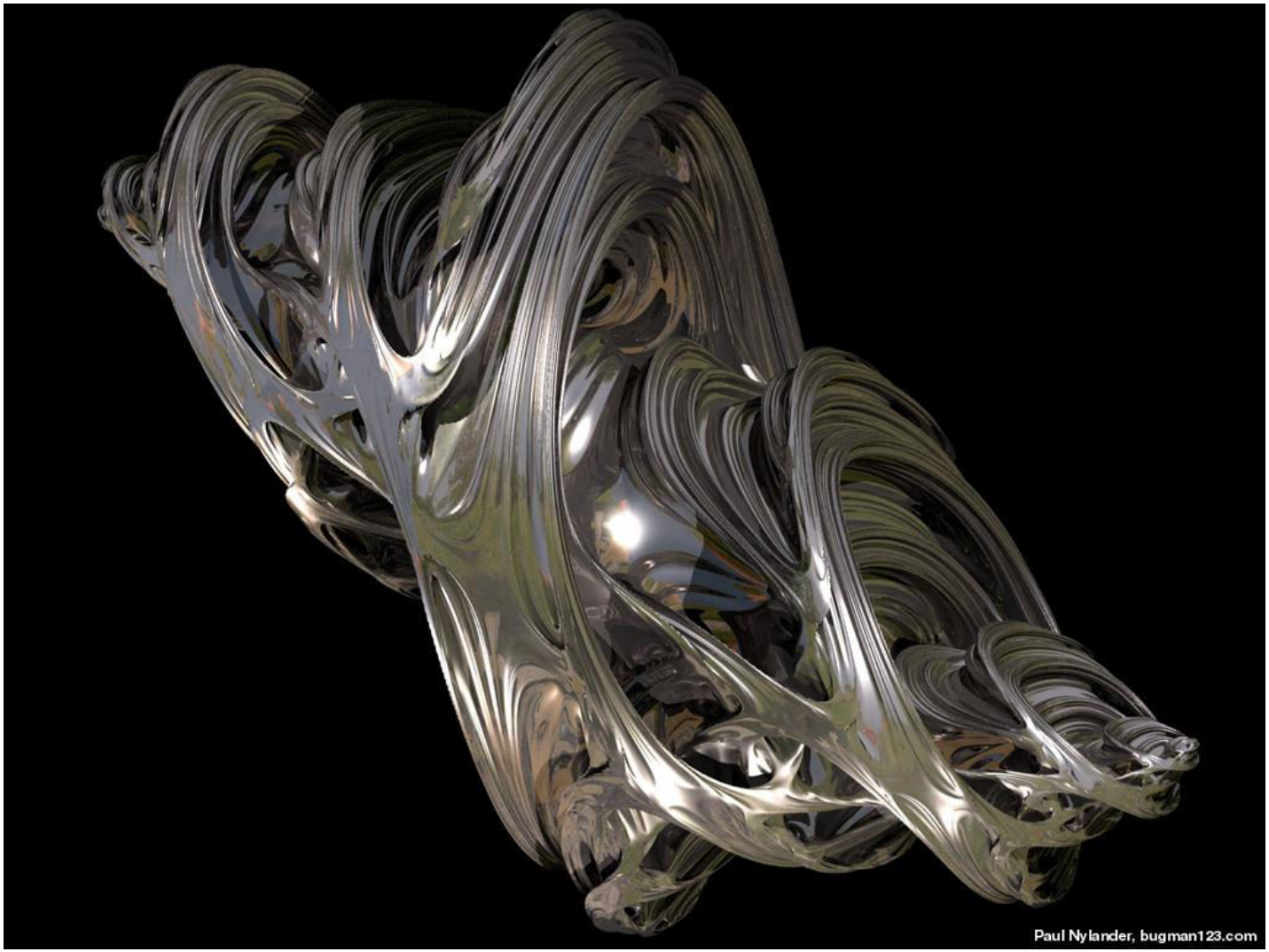
Трассировка лучей

- Функции расстояния
- D = f(x,y,z)
- Distance-Aided Ray Marching

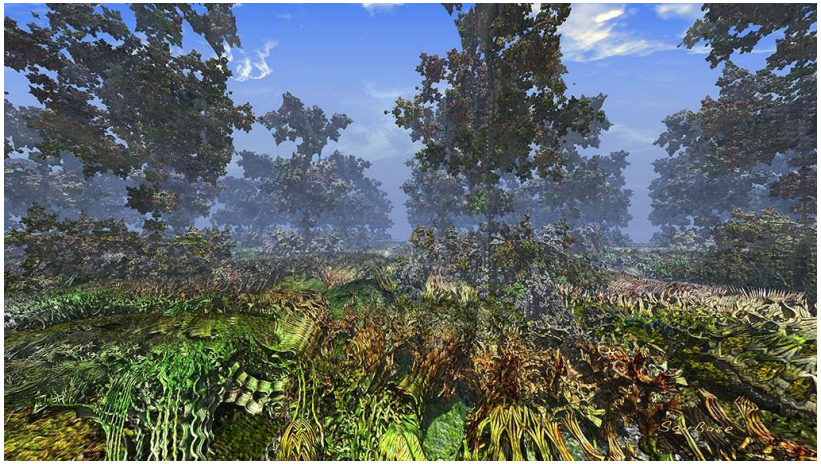


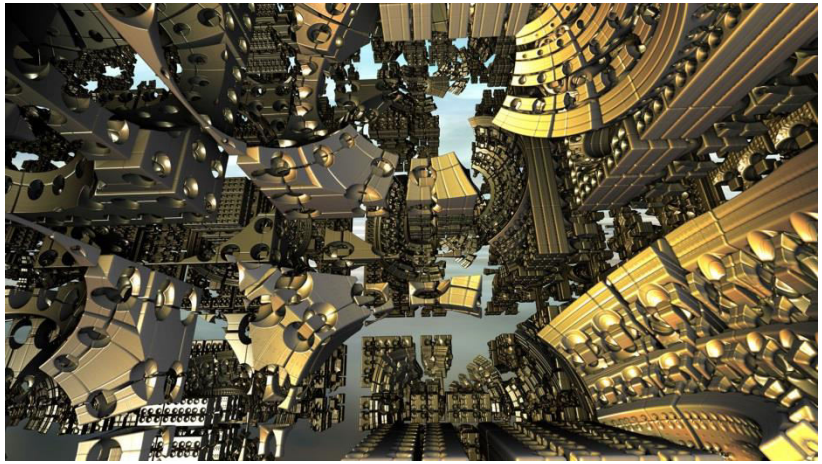


Трассировка лучей



Трассировка лучей

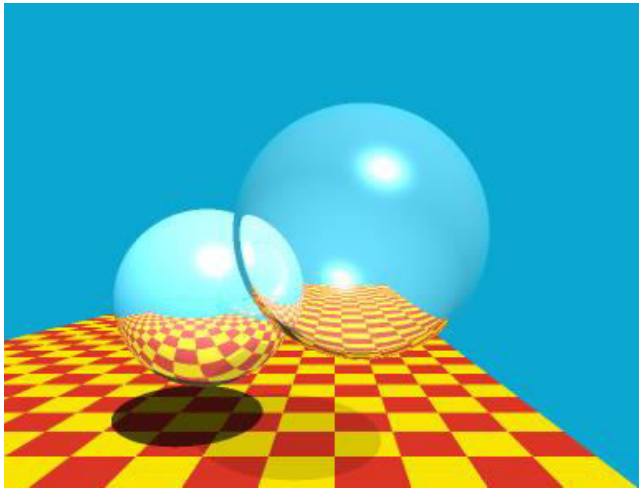


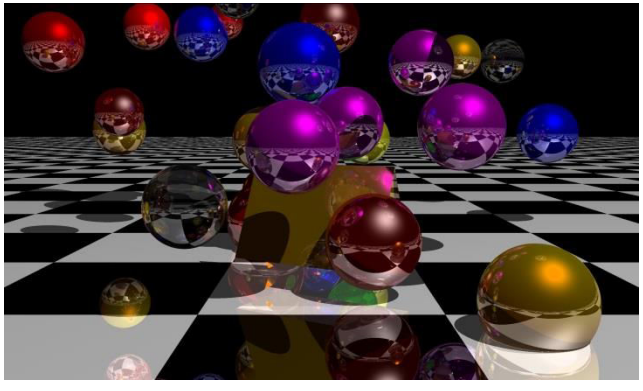


http://www.gamedev.ru/code/forum/?id=146616

Трассировка лучей

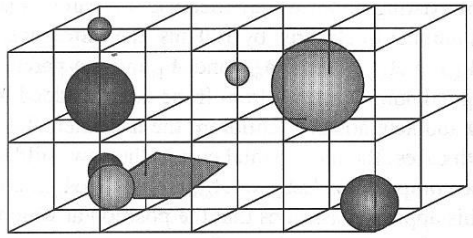
- Проблема (много треугольников)
  - Поиск пересечений

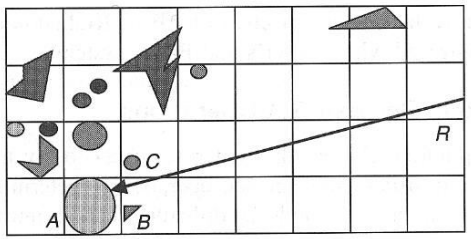




Трассировка лучей

- Ускоряющие структуры
  - Регулярные и иерархические сетки
  - Октодеревья
  - kd-деревья, bsp-деревья
  - BVH-деревья, BIH деревья





- Резюме по ускоряющим структурам
  - Нерегулярные разбиения лучше регулярных
  - Деревья должны быть разбалансированы
  - Наиболее практичной структурой считается BVH

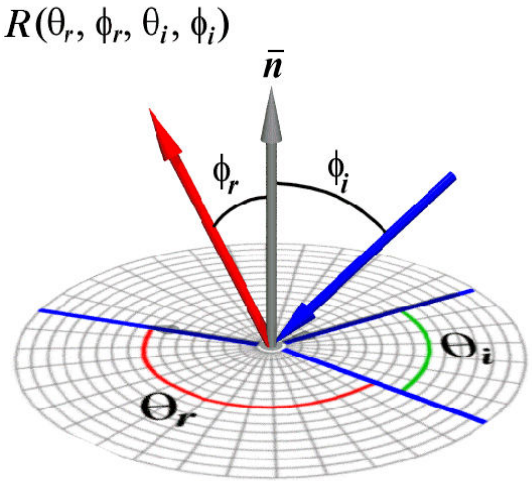
Трассировка лучей

Интеграл освещенности

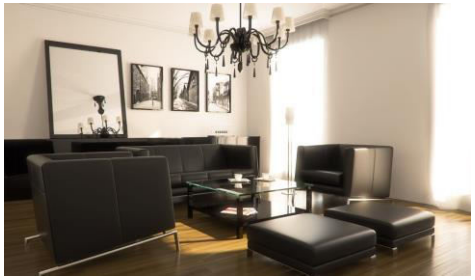
$$I(\varphi_r, \theta_r) = \int_{\varphi_i} \int_{\theta_i} L(\varphi_i, \theta_i) R(\varphi_i, \theta_i, \varphi_r, \theta_r) \cos(n, l_{\varphi_i, \theta_i}) d\varphi_i d\theta_i$$

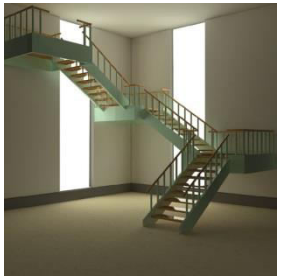
$n - нормаль$

$l_{\varphi_i, \theta_i} - вектор$



Глобальное освещение



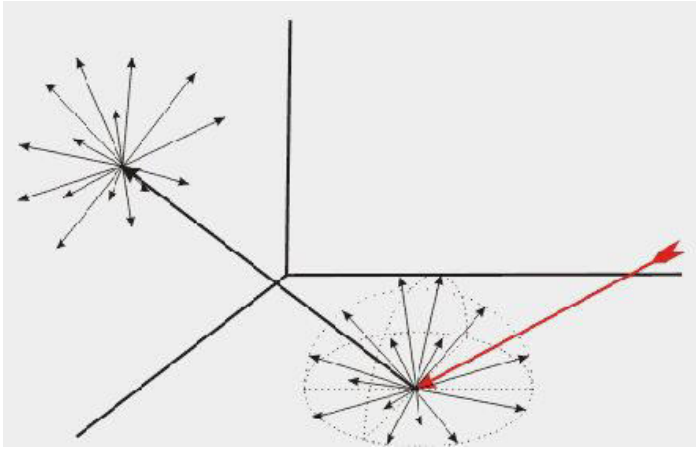


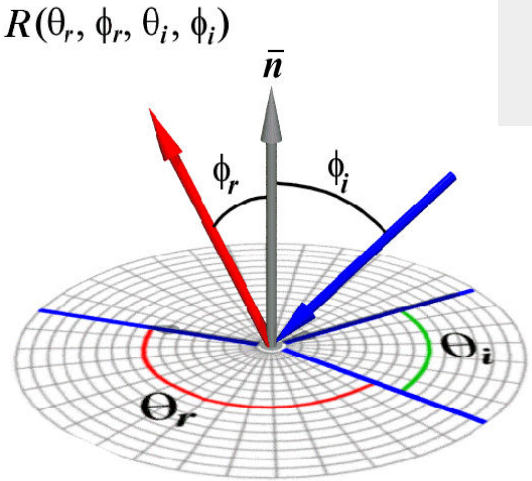
Интеграл освещенности

$$I(\varphi_r,\theta_r)=\int_{\varphi_i}\int_{\theta_i} L(\varphi_i,\theta_i)R(\varphi_i,\theta_i,\varphi_r,\theta_r)\cos(n,l_{\varphi_i,\theta_i})d\varphi_i d\theta_i$$

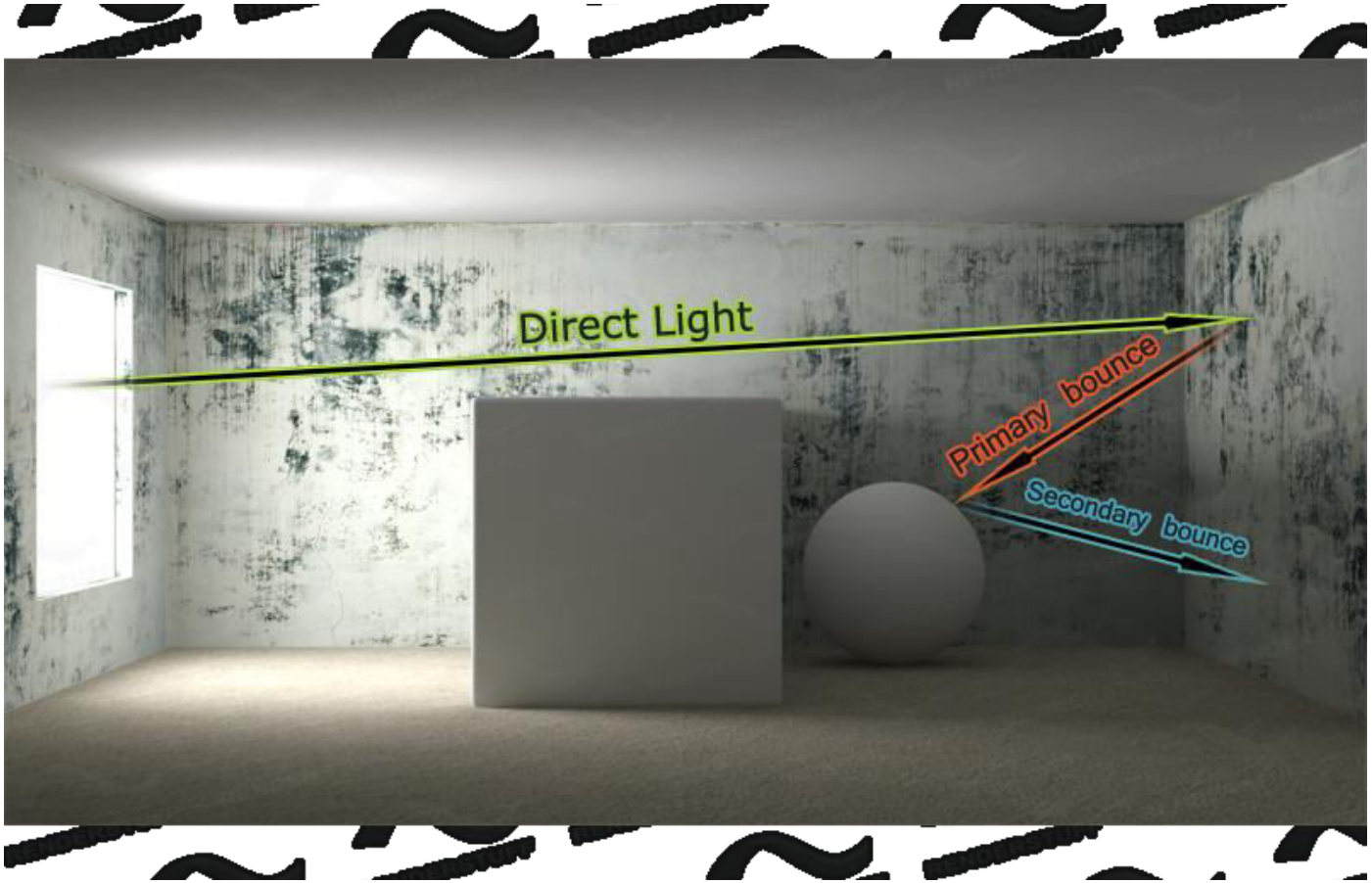
$n-$ *нормаль*

$l_{\varphi_i,\theta_i}-$ *вектор*





Глобальное освещение



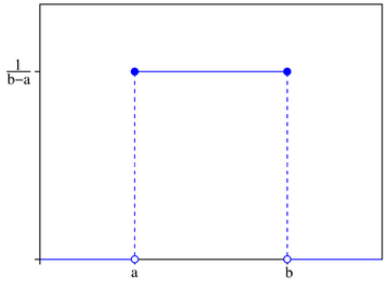
$$I(\varphi_r, \theta_r) = \int_{\varphi_i} \int_{\theta_i} L(\varphi_i, \theta_i) R(\varphi_i, \theta_i, \varphi_r, \theta_r) \cos(n, l_{\varphi_i, \theta_i}) d\varphi_i d\theta_i$$

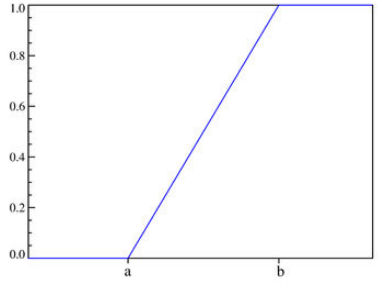
Метод Монте-Карло

- u – случ. Величина, равн. Распред. на [a,b]
- Тогда f(u) – тоже случ. величина.

$$Ef(u) = \int_a^b f(x)\varphi(x)dx$$

$$\varphi(x) = \frac{1}{b-a}$$





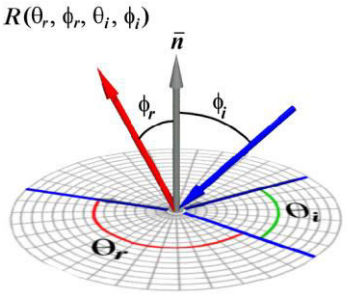
$$\int_a^b f(x)dx = (b-a)Ef(u) \approx \frac{b-a}{N}\sum f(u_i)$$

Метод Монте-Карло

$$I(\varphi_r, \theta_r) = \int_{\varphi_i} \int_{\theta_i} L(\varphi_i, \theta_i) R(\varphi_i, \theta_i, \varphi_r, \theta_r) \cos(n, l_{\varphi_i, \theta_i}) d\varphi_i d\theta_i$$

$$\int_a^b f(x)dx = (b-a)Ef(u) \approx \frac{b-a}{N} \sum f(u_i)$$

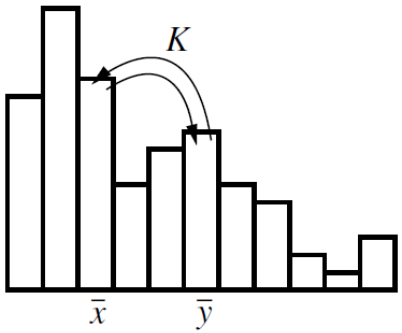
$$I(\varphi_r, \theta_r) \approx \frac{\sum_i L(\varphi_i, \theta_i) R(\varphi_i, \theta_i, \varphi_r, \theta_r) \cos(n, l_{\varphi_i, \theta_i})}{N}$$

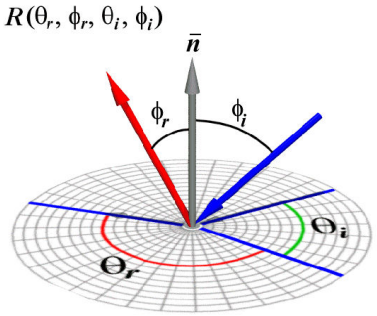


Метод Монте-Карло

- Смещенная оценка (biased)
- Несмещенная оценка (unbiased)
- Выборка по значимости (Importance sampling)

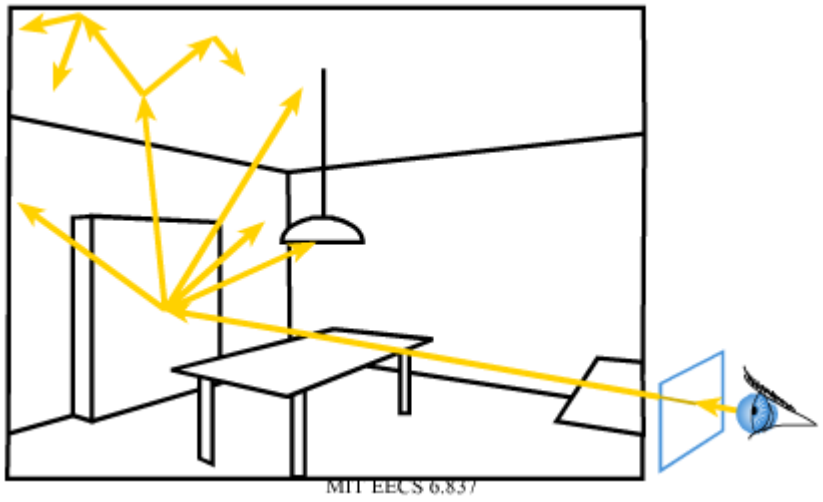
$$I(\varphi_r,\theta_r) \approx \frac{\sum_i L(\varphi_i,\theta_i)R(\varphi_i,\theta_i,\varphi_r,\theta_r)\cos(n,l_{\varphi_i,\theta_i})}{N} \approx \frac{\sum_i f(u_i)}{M}$$

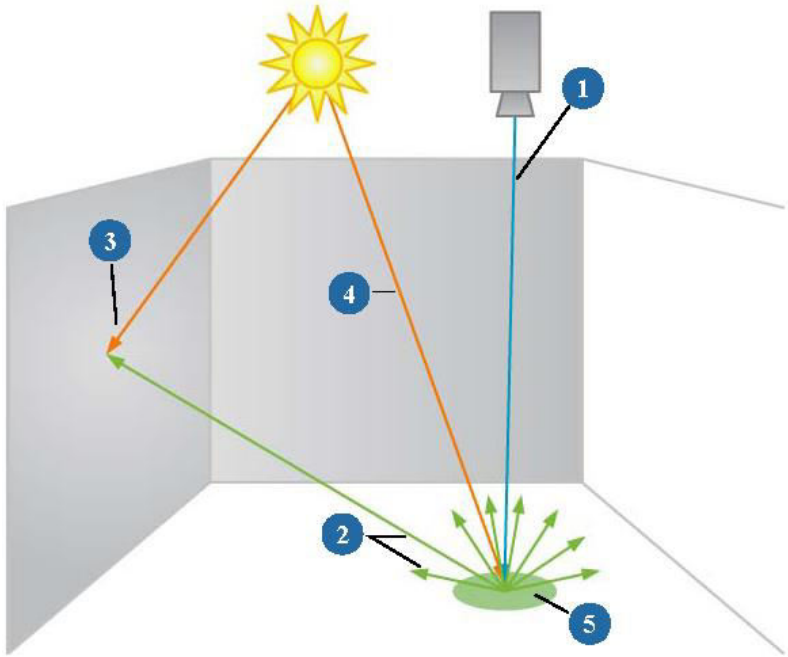




Монте-Карло Трассировка

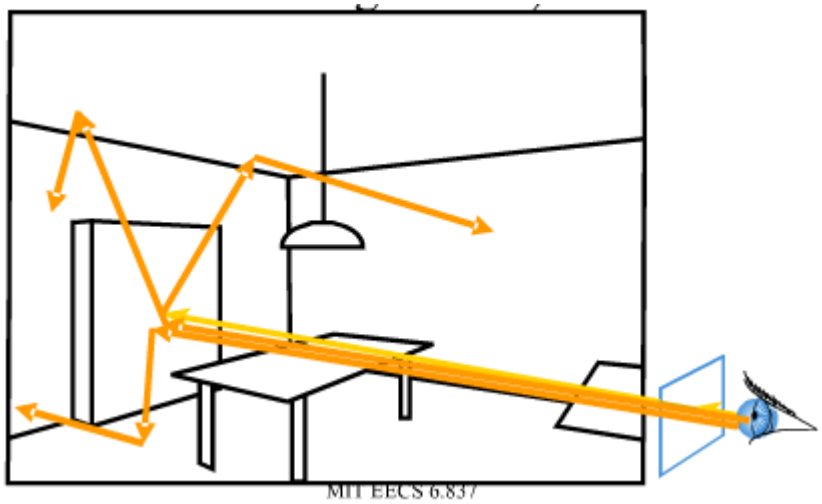
$$I(\varphi_r, \theta_r) \approx \frac{\sum_{i,r} L(\varphi_i, \theta_i) R(\varphi_i, \theta_i, \varphi_r, \theta_r) \cos(n, l_{\varphi_i, \theta_i})}{N}$$

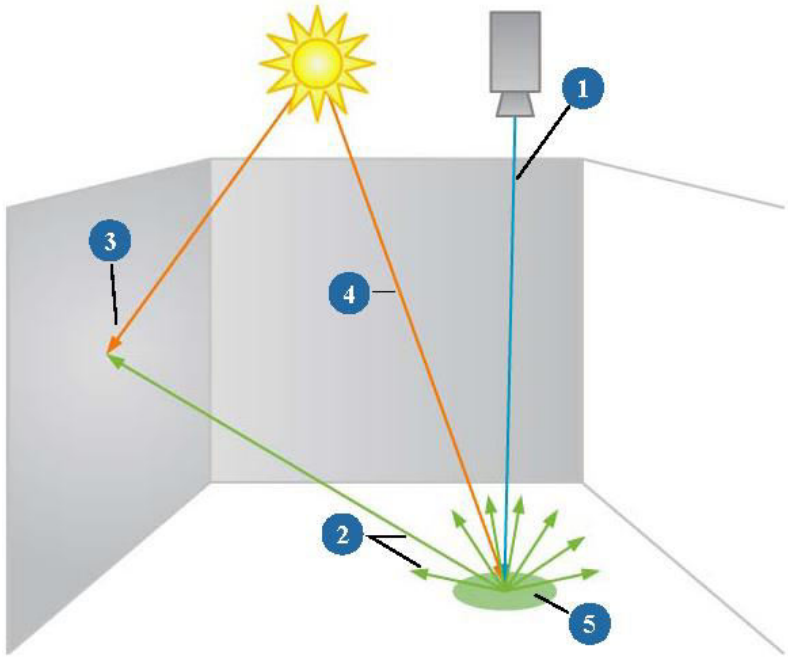




Трассировка путей

$$I(\varphi_r,\theta_r) \approx \frac{\sum_i L(\varphi_i,\theta_i) R(\varphi_i,\theta_i,\varphi_r,\theta_r) \cos(n, l_{\varphi_i,\theta_i})}{N}$$





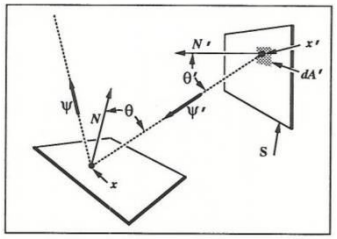
Исходные параметры: ray - Луч, depth - глубина рекурсии

Результат: Монте-Карло выборка значения освещенности

Function *TracePath(ray: Ray; depth : Integer) : Integer* **is**

```
if depth == MAX_DEPTH then
    return Black;
end
hit ← RaySceneIntersection(ray)
if not hit.exists then
    return Black;
end
m ← hit.material
if m.IsLight then
    return m.emittance;
end
newRay.O ← hit.pos
newRay.D ← RandomUnitVectorInHemisphereOf(hit.norm)
cosTheta ← dot(newRay.D, hit.norm)
BRDF ← m.reflectance * cosTheta
return BRDF * TracePath(newRay, depth + 1) ;
```

end



Алгоритм 2: Простейший вид Монте-Карло трассировки путей. I

Генерация случайного луча

```
float3 MapSamplesToSphere(float r1, float r2) // [0..1]
{
  float phi = r1*3.141592654f*2;              //[0..2*PI]
  float h   = r2*2 - 1;                       //[-1..1]

  float sin_phi, cos_phi;
  sincosf(phi, &sin_phi, &cos_phi);

  float x = sin_phi*sqrt(1-h*h);
  float y = cos_phi*sqrt(1-h*h);
  float z = h;

  return float3(x,y,z);
}
```

http://www.raytracegroundup.com/

Что вычислять и когда остановиться?

- Нужно накапливать:
  - Сумму цветов
  - Сумму квадратов цветов
- Ошибка оценивается из
  - Стандартного отклонения

$$C_0 = P_0$$

$$C_1 = \frac{1}{2}C_0 + \frac{1}{2}P_1$$

$$C_2 = \frac{2}{3}C_1 + \frac{1}{3}P_2$$

$$C_3 = \frac{3}{4}C_2 + \frac{1}{4}P_3$$

$$\ldots$$

$$C_n = \frac{n}{n+1}C_{n-1} + \frac{1}{n+1}P_n$$

$$D = \frac{\sum_{i=1}^{n} X_i^2 - \frac{(\sum_{i=1}^{n} X_i)^2}{n}}{n-1}$$

$$\sigma = \sqrt{\frac{D}{n}}$$

Выборка по значимости

Исходные параметры: ray - Луч, depth - глубина рекурсии

Результат: Монте-Карло выборка значения освещенности

Function *TracePath(ray: Ray; depth : Integer) : Integer* **is**

```
if depth == MAX_DEPTH then
    return Black;
end
hit ← RaySceneIntersection(ray)
if not hit.exists then
    return Black;
end
m ← hit.material
if m.IsLight then
    return m.emittance;
end
newRay.O ← hit.pos
newRay.D ← RandomUnitVectorInHemisphereOf(hit.norm)
cosTheta ← dot(newRay.D, hit.norm)
BRDF ← m.reflectance * cosTheta
return BRDF * TracePath(newRay, depth + 1) ;
```

end

Алгоритм 2: Простейший вид Монте-Карло трассировки путей. I

Исходные параметры: ray - Луч, depth - глубина рекурсии

Результат: Монте-Карло выборка значения освещенности

Function *TracePath(ray: Ray; depth : Integer) : Integer* **is**

```
if depth == MAX_DEPTH then
    return Black;
end
hit ← RaySceneIntersection(ray)
if not hit.exists then
    return Black;
end
m ← hit.material
if m.IsLight then
    return m.emittance;
end
newRay.O ← hit.pos
newRay.D ← RandomCosineVectorOf(hit.norm)
BRDF ← m.reflectance
return BRDF * TracePath(newRay, depth + 1) ;
```

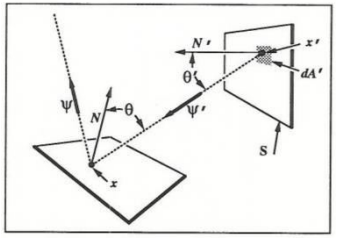
end

Алгоритм 3: Трассировка путей с учетом косинусоидального распределения отраженных лучей.

Генерация случайного луча

```
float3 MapSampleToCosineDistribution(float r1, float r2)
{
  float e = 1.0;
  float sin_phi, cos_phi;

  sincosf(2*r1*3.141592654f, &sin_phi, &cos_phi);

  float cos_theta = powf(1.0f-r2,1.0f/(e+1.0f));
  float sin_theta = sqrtf(1.0f-cos_theta*cos_theta);

  float x = sin_theta*cos_phi;
  float y = sin_theta*sin_phi;
  float z = cos_theta;

  return float3(x,y,z);
}
```

http://www.raytracegroundup.com/



Генерация случайного луча

- Не диффузная BRDF

```
float3 MapSampleToCosineDistribution(float r1, float r2,
                                     float3 direction, float3 hit_norm, float power) {
  float e = power;
  float sin_phi, cos_phi;

  sincosf(2*r1*3.141592654f, &sin_phi, &cos_phi);

  float cos_theta = powf(1.0f-r2,1.0f/(e+1.0f));
  float sin_theta = sqrtf(1.0f-cos_theta*cos_theta);

  float3 deviation(sin_theta*cos_phi, sin_theta*sin_phi, cos_theta);

  float3 ny = direction;
  float3 nx = normalize(cross(ny, float3(1.04,2.93f,-0.6234f)));
  float3 nz = normalize(cross(nx, ny));
  swap(ny,nz);

  float3 res = nx*deviation.x + ny*deviation.y + nz*deviation.z;

  float invSign = dot(direction, hit_norm) > 0.0f ? 1.0f : -1.0f;

  if(invSign*dot(res, hit_norm) < 0.0f)
    res = -nx*deviation.x + ny*deviation.y - nz*deviation.z;

  return res;
}
```

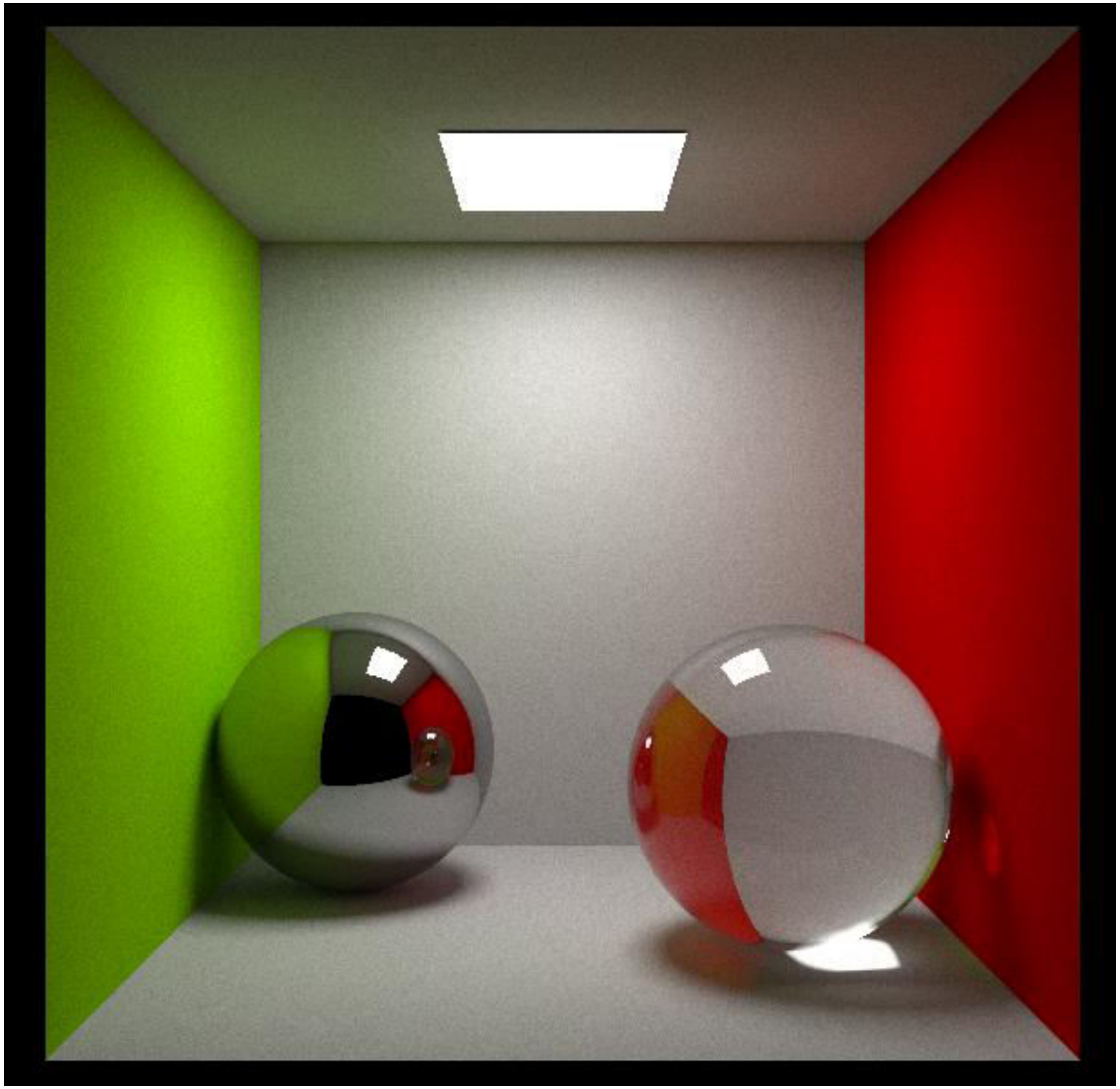
http://www.raytracegroundup.com/

Генерация случайного луча

- Правильная ортогонализация Грамма-Шмидта

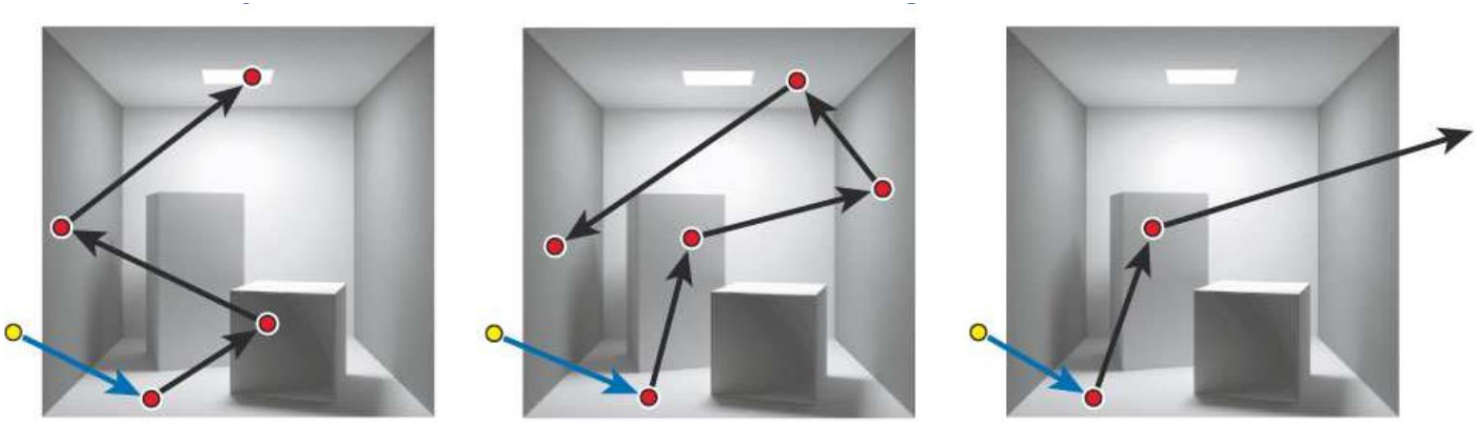
```
float3 GetPerpendicular(float3 a_vec) const
{
  float3 leastPerpendicular;
  float  bestProjection;
  float  xProjection = fabs(a_vec.x);
  float  yProjection = fabs(a_vec.y);
  float  zProjection = fabs(a_vec.z);

  if((xProjection <= yProjection + 1e-5) && (xProjection <= zProjection + 1e-5))
  {
    leastPerpendicular = float3(1, 0, 0);
    bestProjection = xProjection;
  }
  else if((yProjection < xProjection + 1e-5) && (yProjection <= zProjection + 1e-5))
  {
    leastPerpendicular = float3(0, 1, 0);
    bestProjection = yProjection;
  }
  else
  {
    leastPerpendicular = float3(0, 0, 1);
    bestProjection = zProjection;
  }

  return normalize(cross(a_vec, leastPerpendicular));
}
```

Пример работы алгоритма



Трассировка путей

- Иллюстрация



Генерация случайного луча

- Не только диффузная BRDF

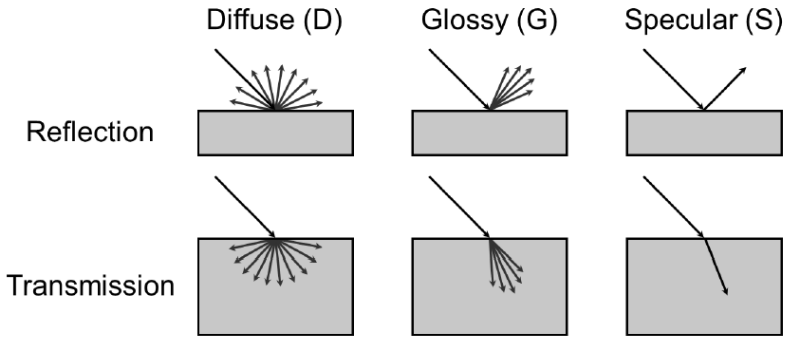
$$\Sigma = kd + ks + kt$$

$$\xi = rnd(0, 1)$$

$$\xi \in [0, \frac{kd}{\Sigma}] \Rightarrow diffuse$$

$$\xi \in [\frac{kd}{\Sigma}, \frac{kd + ks}{\Sigma}] \Rightarrow specular$$

$$\xi \in [\frac{kd + ks}{\Sigma}, 1] \Rightarrow transmittion$$



При этом:

Уже не нужно умножать цвет на коэффициенты kd, ks , kt!

Генерация случайного луча

- Добавляем цвет

$$Pd = max(kd.r, kd.g, kd.b)$$

$$Ps = max(ks.r, ks.g, ks.b)$$

$$Pt = max(kt.r, kt.g, kt.b)$$

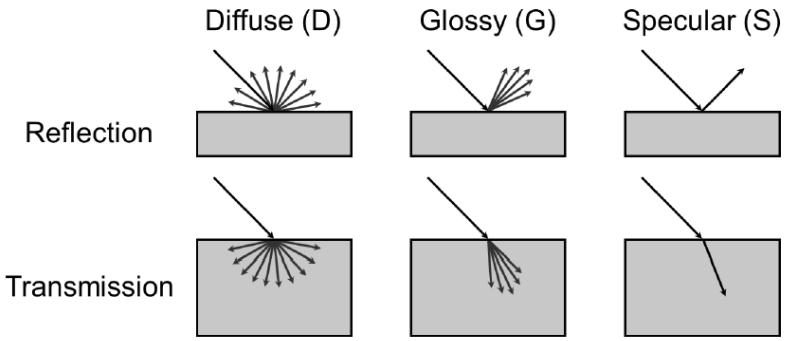
$$\Sigma = Pd + Ps + Pt$$

$$\xi = rnd(0, 1)$$

$$\xi \in [0, \frac{Pd}{\Sigma}] \Rightarrow diffuse$$

$$\xi \in [\frac{Pd}{\Sigma}, \frac{Pd + Ps}{\Sigma}] \Rightarrow specular$$

$$\xi \in [\frac{Pd + Ps}{\Sigma}, 1] \Rightarrow transmittion$$



$$color.r = color.r/(\frac{Ps}{\Sigma})$$

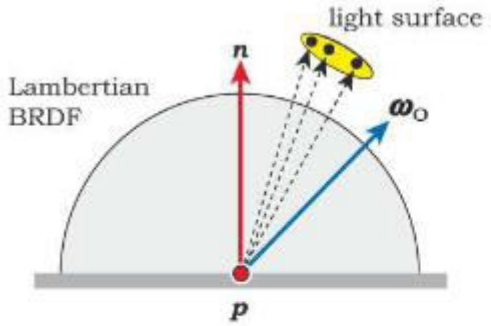
$$color.g = color.g/(\frac{Ps}{\Sigma})$$

$$color.b = color.b/(\frac{Ps}{\Sigma})$$

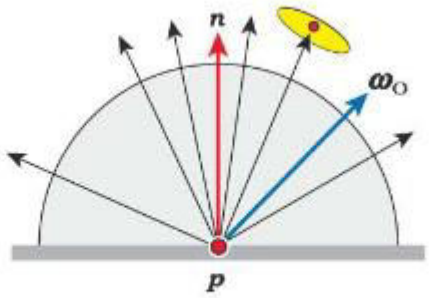
Трассировка путей

- Добавляем теневые лучи
- Выборка по значимости
- Затухание!

float attenuation = 1/(R*R);



appropriate sampling



incident radiance is
small for many samples

diffuse surface

Исходные параметры: ray - Луч, depth - глубина рекурсии

Результат: Монте-Карло выборка значения освещенности

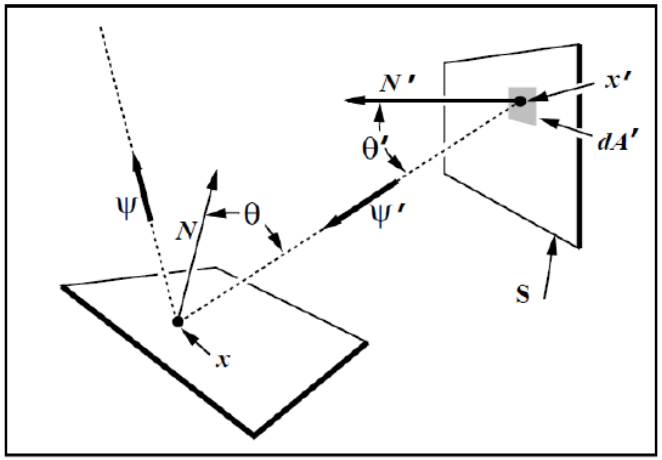
Function *TracePath(ray: Ray; depth : Integer) : Integer* **is**

```
if depth == MAX_DEPTH then
    return Black;
end
hit ← RaySceneIntersection(ray)
if not hit.exists then
    return Black;
end
m ← hit.material
if m.IsLight then
    return Black;
end
lpos ← LightSample(light)
shadow ← Visibility(hit.pos, lpos)
R ← dist(hit.pos, lpos)
sdir ← normalize(lpos − hit.pos)
cosTheta1 ← −dot(sdir, light.norm)
cosTheta2 ← dot(sdir, hit.norm)
implicitP ← light.area ∗ cosTheta1 ∗ cosTheta2/(π ∗ R²)
newRay.O ← hit.pos
newRay.D ← RandomCosineVectorOf(hit.norm)
BRDF ← m.reflectance
explicitColor ← shadow ∗ BRDF ∗ light.emittance ∗ implicitP
return explicitColor + BRDF * TracePath(newRay, depth + 1) ;
```

end

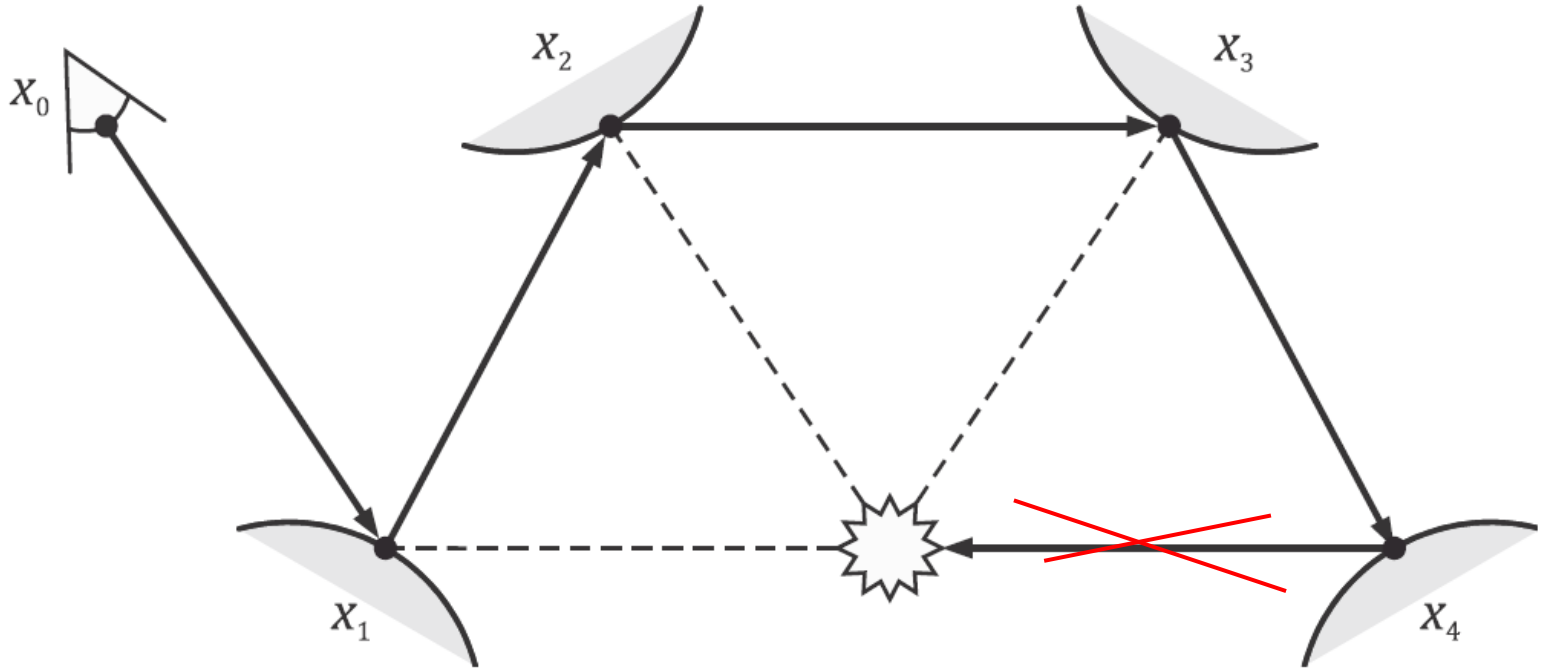
Алгоритм 4: Трассировка путей с теневыми лучами.

$$I(x,\psi) = \int_{\psi'} R(x,\psi')L(x,\psi')\frac{dA'cos\theta cos\theta'}{||x-x'||}$$

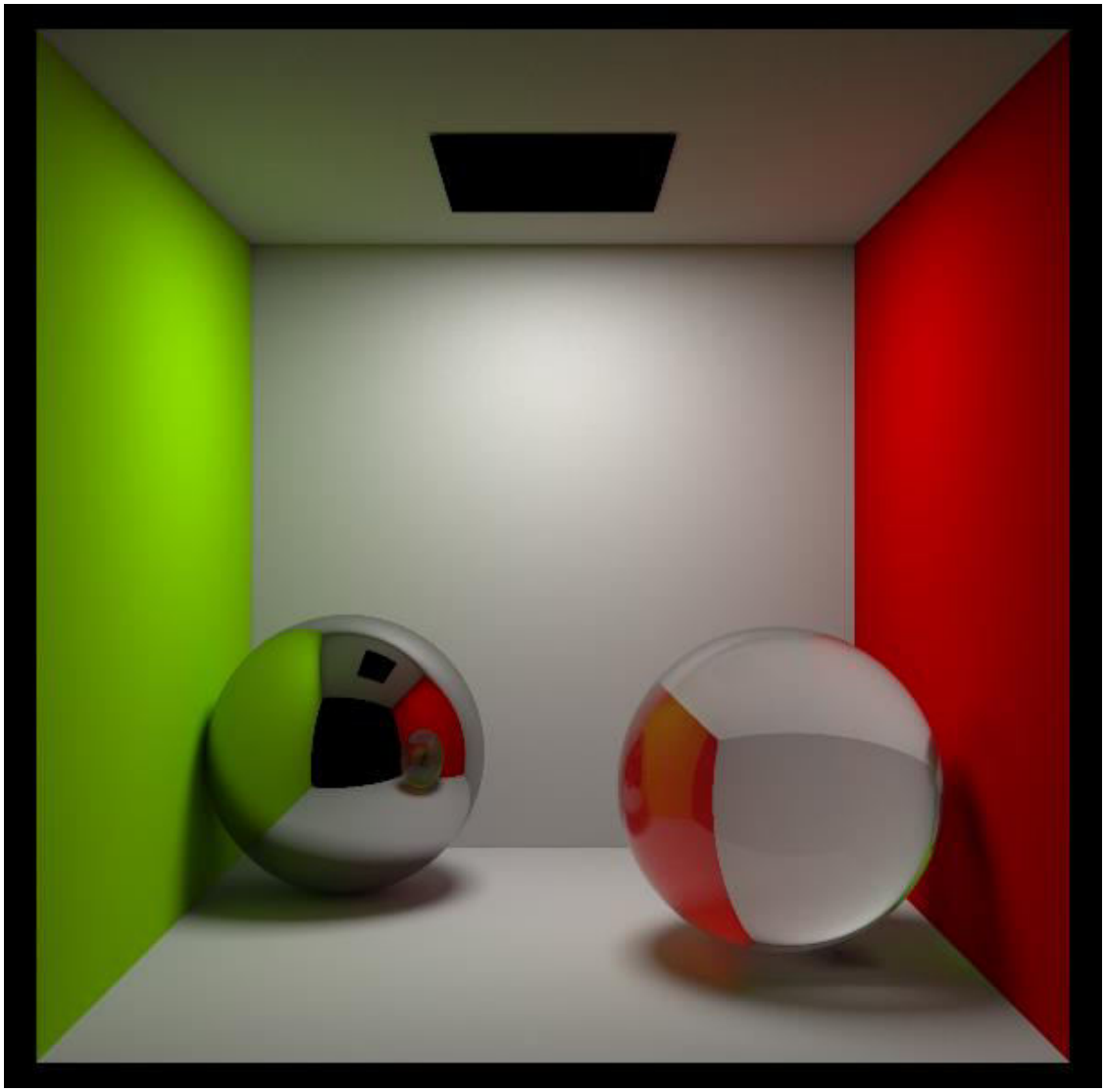


Корректность

- Почему схема работает?
- Только 1 стратегия, нельзя ловить источник!

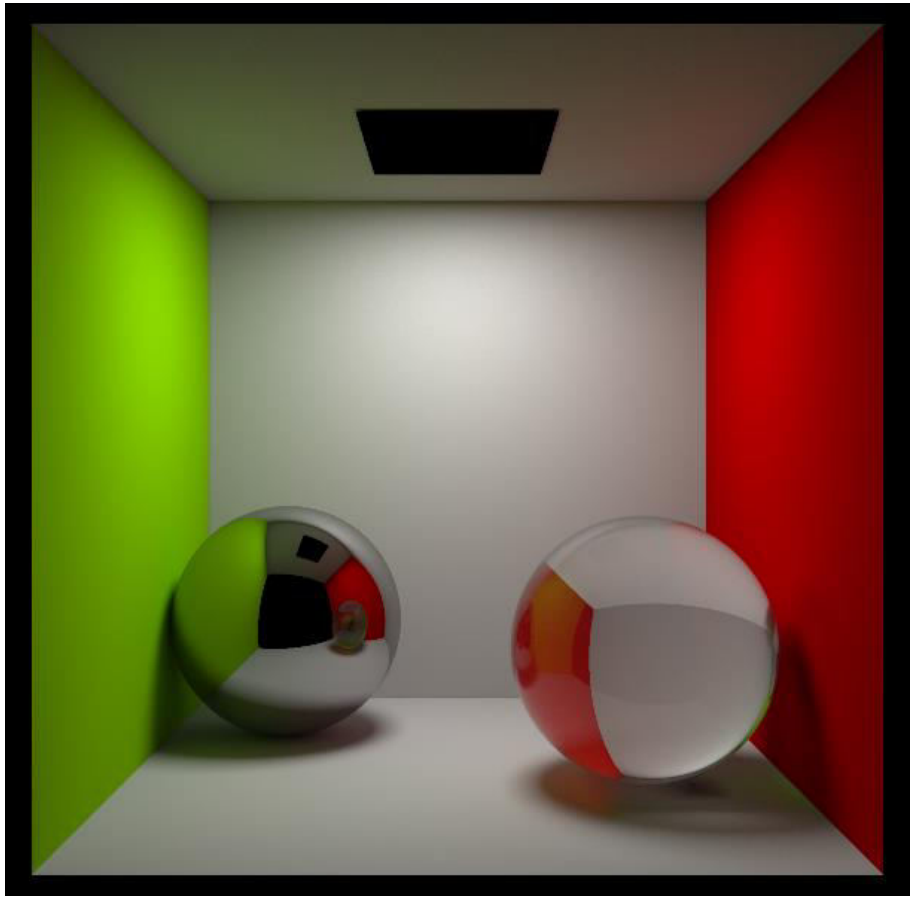


Пример работы алгоритма



Пример работы алгоритма

- Куда делись каустики?



Что дают теневые лучи?

- Сравнение

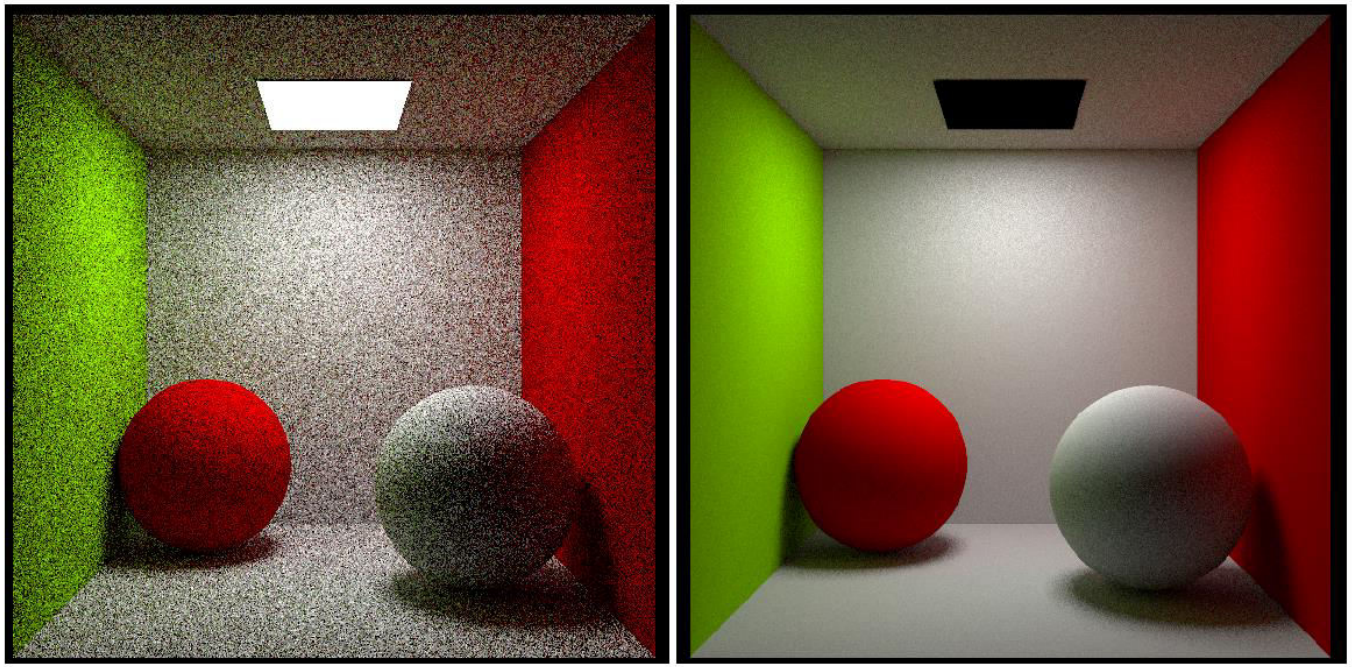
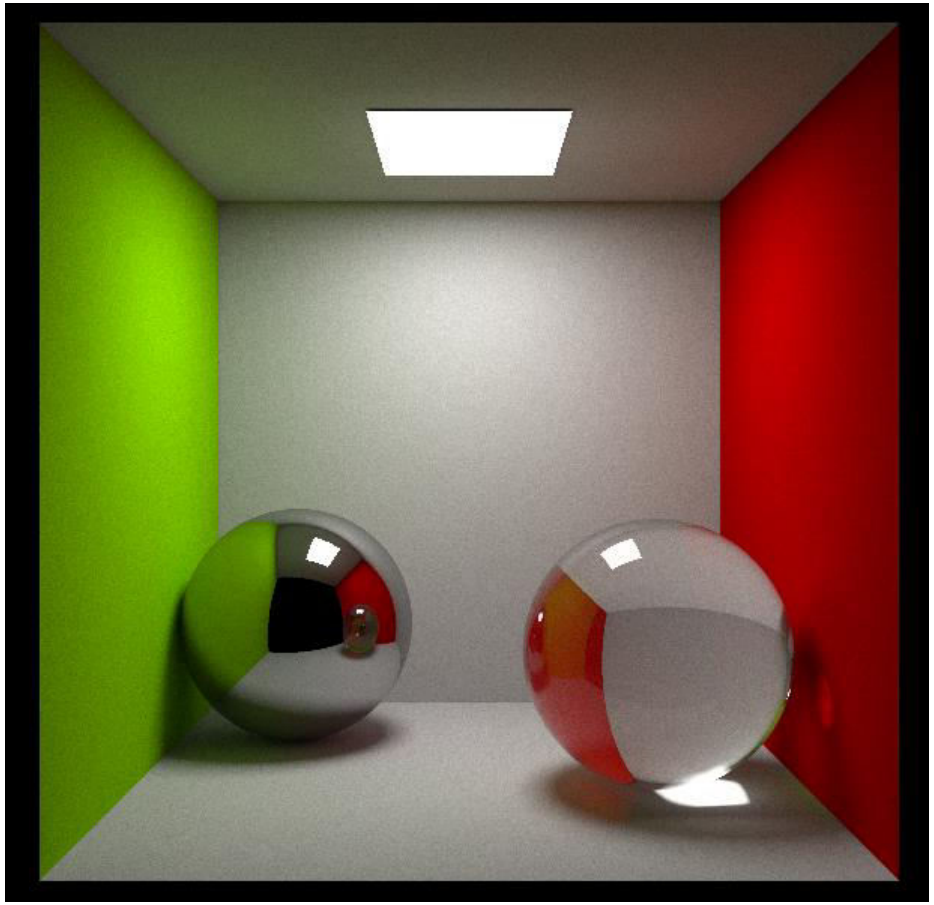


Рис. 1.35. Сравнение простого алгоритма трассировки путей (64 путя на пиксел, слева на изображении) и алгоритма трассировки путей использующего теневые лучи (32 путя на пиксел, справа на изображении). Оба изображения получены за примерно одинаковое время.

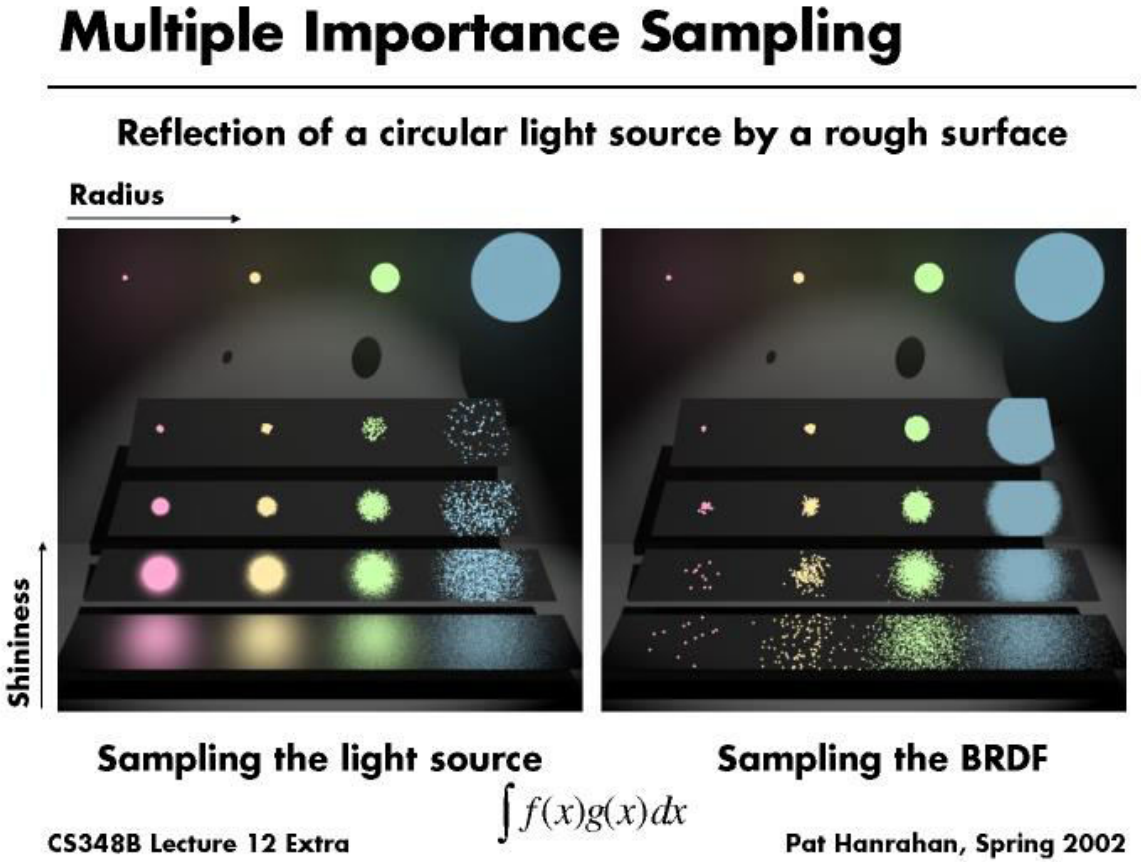
Пример работы алгоритма

- Как вернуть каустики?



Две стратегии сэмплирования

- Явная
- Неявная



$$p_i = cos(\theta)/\pi$$

$$p_e = 1/A$$

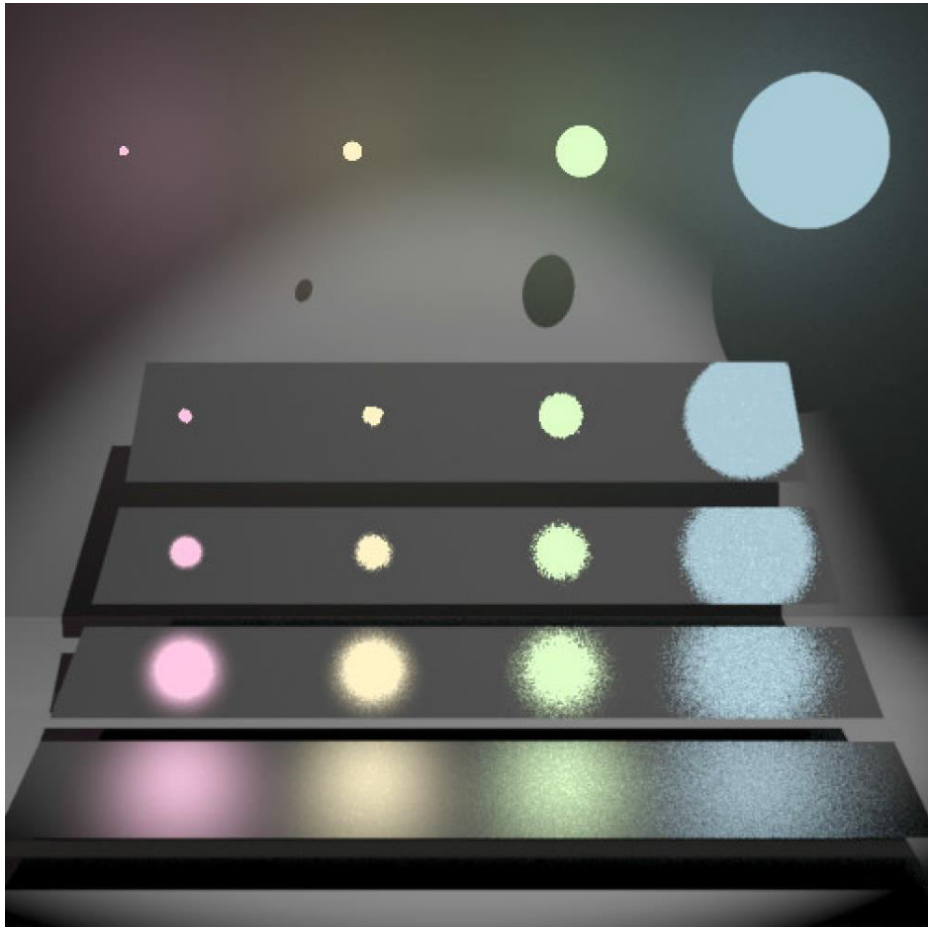
$$w_e = \frac{p_e^\beta}{p_i^\beta + p_e^\beta}$$

$$w_i = \frac{p_i^\beta}{p_i^\beta + p_e^\beta}$$

$$\beta = 2$$

Пример работы алгоритма

- Многократная выборка по значимости (MIS)



$$p_i = cos(\theta)/\pi$$

$$p_e = 1/A$$

$$w_e = \frac{p_e^\beta}{p_i^\beta + p_e^\beta}$$

$$w_i = \frac{p_i^\beta}{p_i^\beta + p_e^\beta}$$

$$\beta = 2$$

Пример работы алгоритма

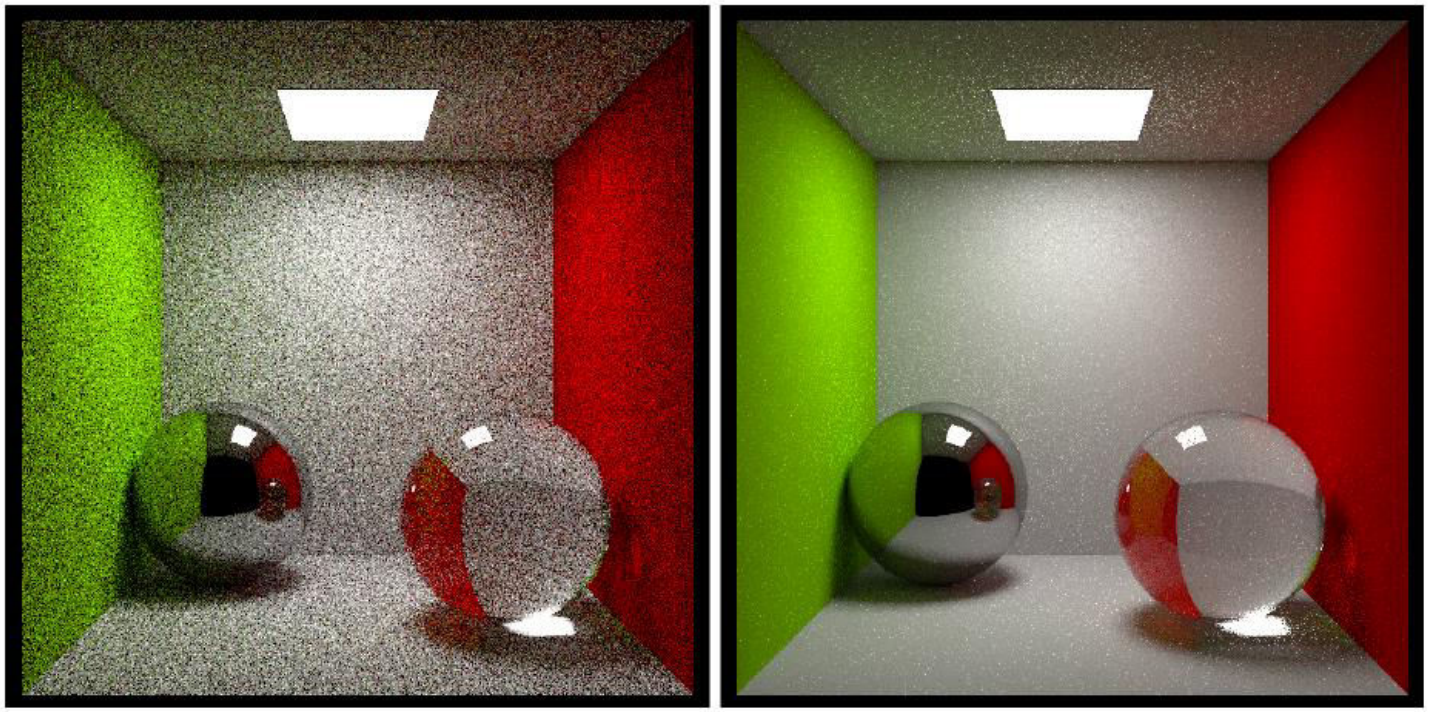
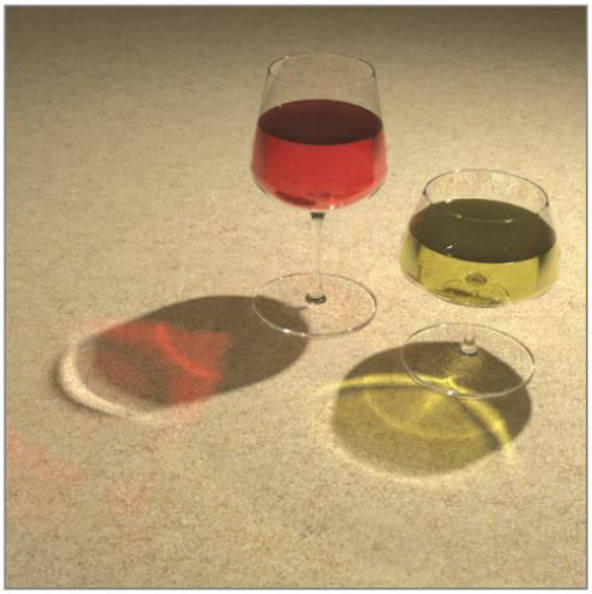


Рис. 1.36. Сравнение простого алгоритма трассировки путей (64 путя на пиксел, слева на изображении) и алгоритма трассировки путей использующего многократную выборку по значимости (32 путя на пиксел, справа на изображении). Обе картинки получены за примерно одинаковое время.

Простой пример реализации http://ray-tracing.ru/articles229.html
http://code.google.com/p/ada-ray-tracer/

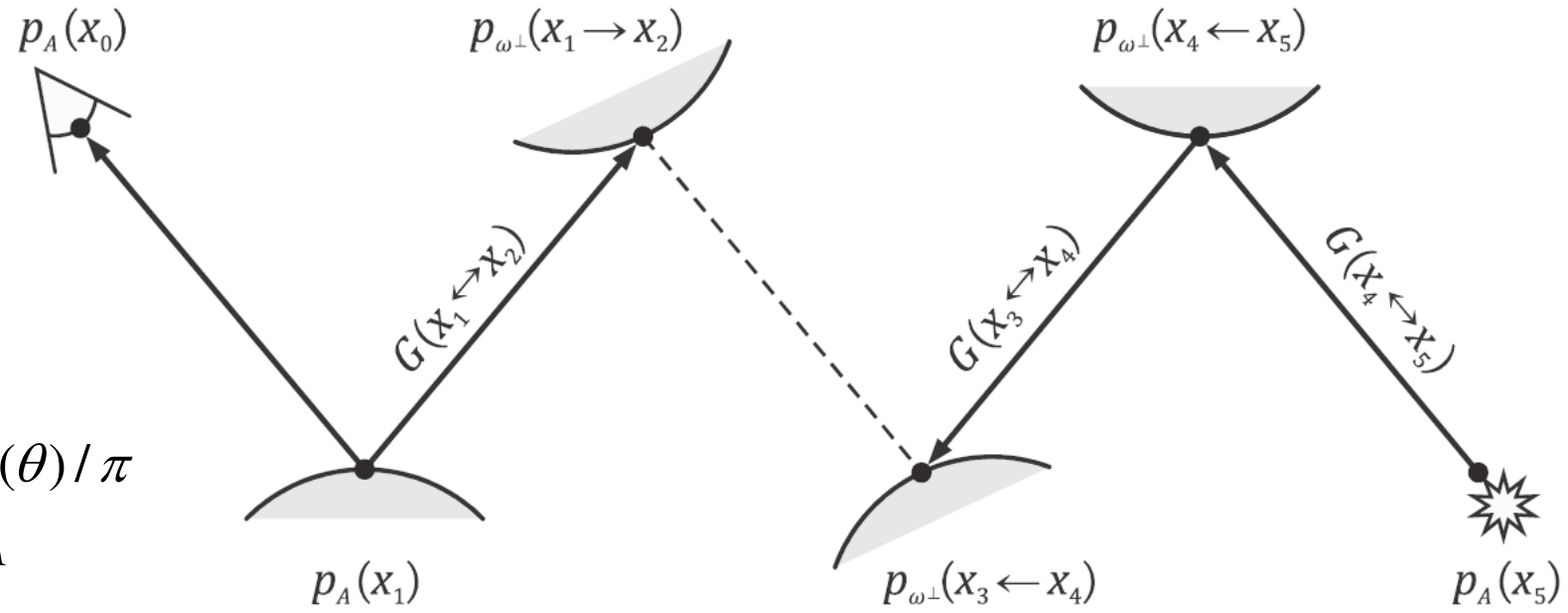
Ligh Tracing

- Прямая трассировка - LT (Light Tracing)



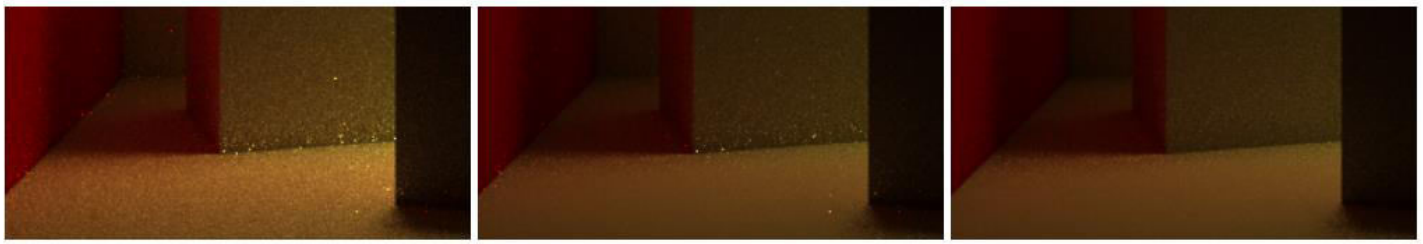
Трассировка путей

- Двунаправленная (BDPT)



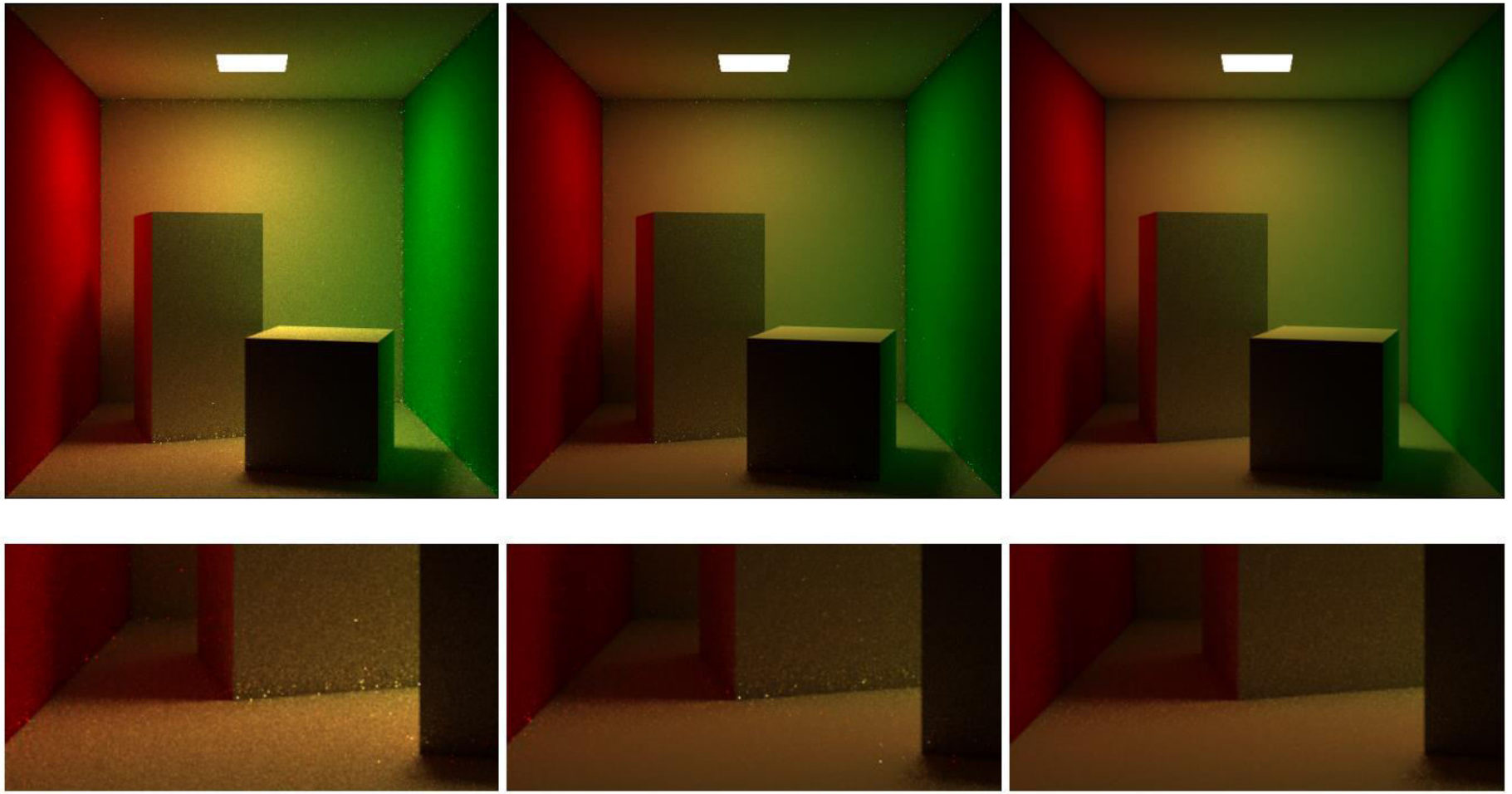
$$p_e = \cos(\theta)/\pi$$

$$p_i = 1/A$$



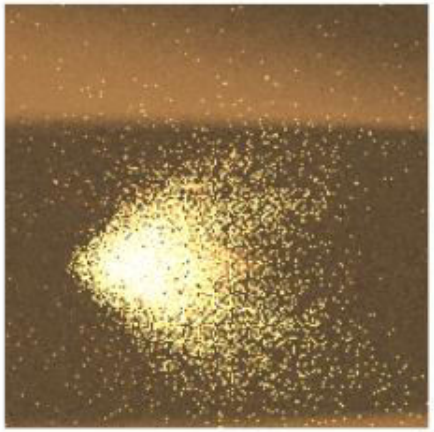
Трассировка путей

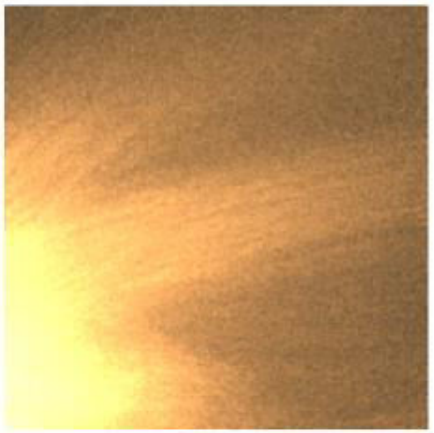
- Двунаправленная (BDPT)

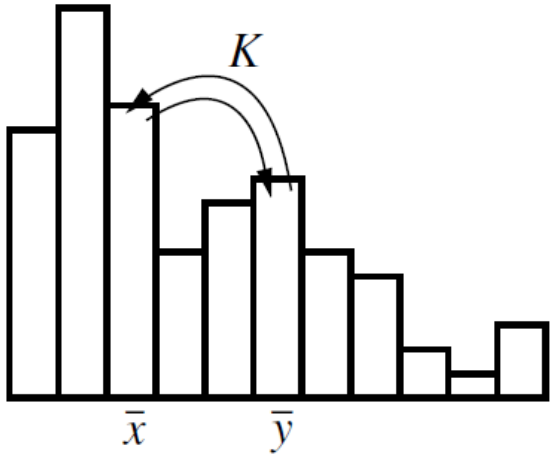


MLT, ERPT

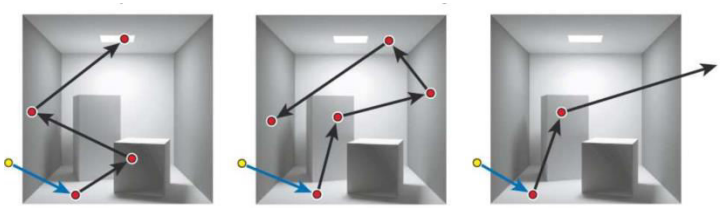
- MLT (Metropolis Light Transport)
- ERPT (Energy Redistribution Path Tracing)







$$K(\bar{y}|\bar{x}) = K(\bar{x}|\bar{y}).$$



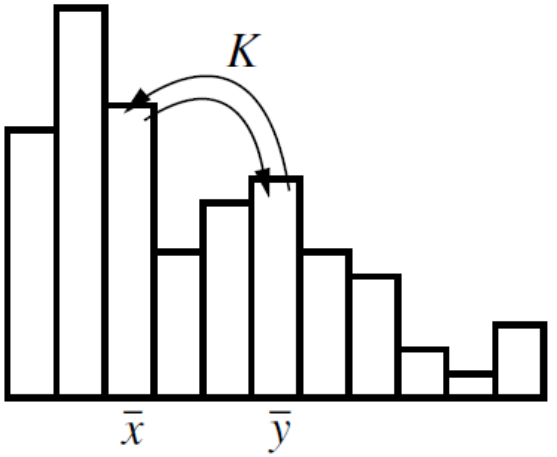
MLT, ERPT

- MLT (Metropolis Light Transport)
- ERPT (Energy Redistribution Path Tracing)

$$T(\bar{y} \mid \bar{x}) = p(\bar{y} \xrightarrow{next} \bar{x})$$

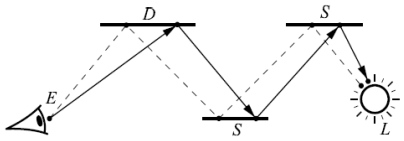
$$a(\bar{y} \mid \bar{x}) = \min\{1, \frac{f(\bar{y})T(\bar{x} \mid \bar{y})}{f(\bar{x})T(\bar{y} \mid \bar{x})}\}$$

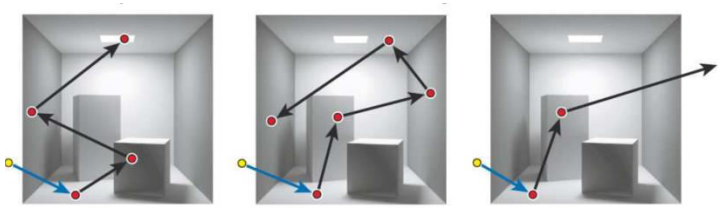
$$1 - a(\bar{y} \mid \bar{x})$$



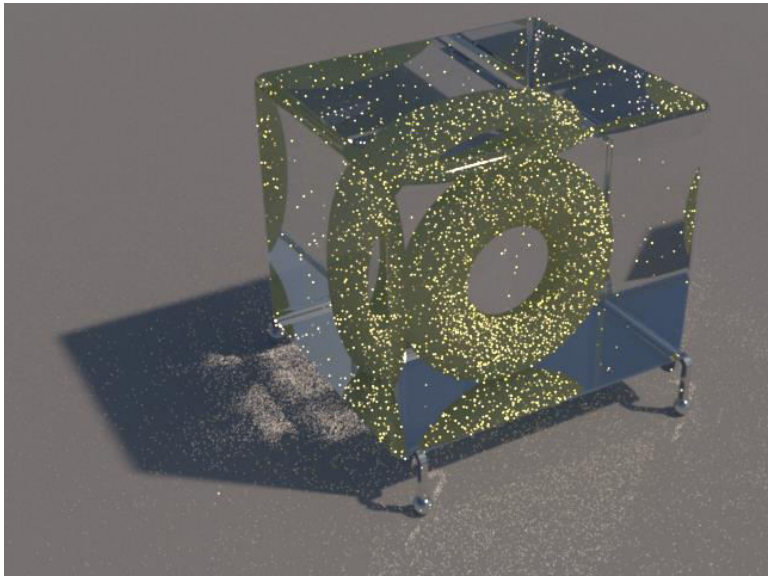
$\bar{x} \leftarrow$ InitialPath()
image $\leftarrow$ { array of zeros }
for $i \leftarrow 1$ **to** N
  $\bar{y} \leftarrow$ Mutate($\bar{x}$)
  $a \leftarrow$ AcceptProb($\bar{y}|\bar{x}$)
  if Random() $< a$
    then $\bar{x} \leftarrow \bar{y}$
  RecordSample(*image*, $\bar{x}$)
return *image*

$$K(\bar{y}|\bar{x}) = K(\bar{x}|\bar{y}).$$

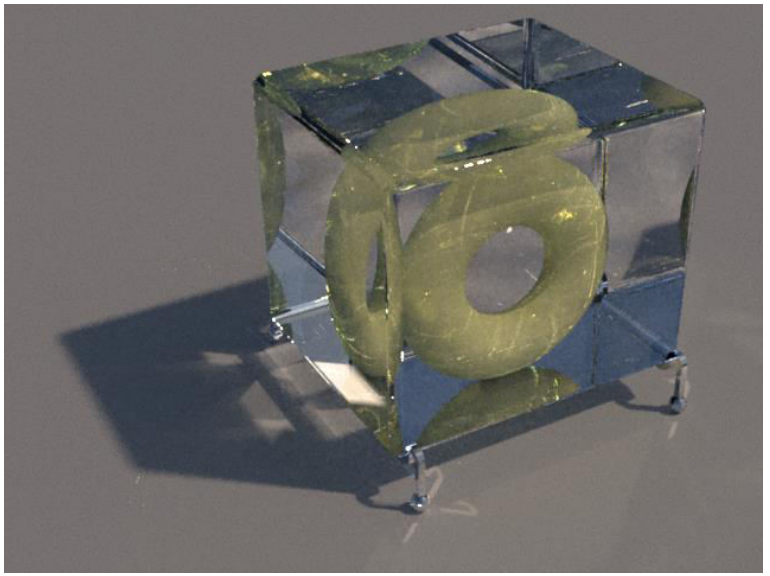




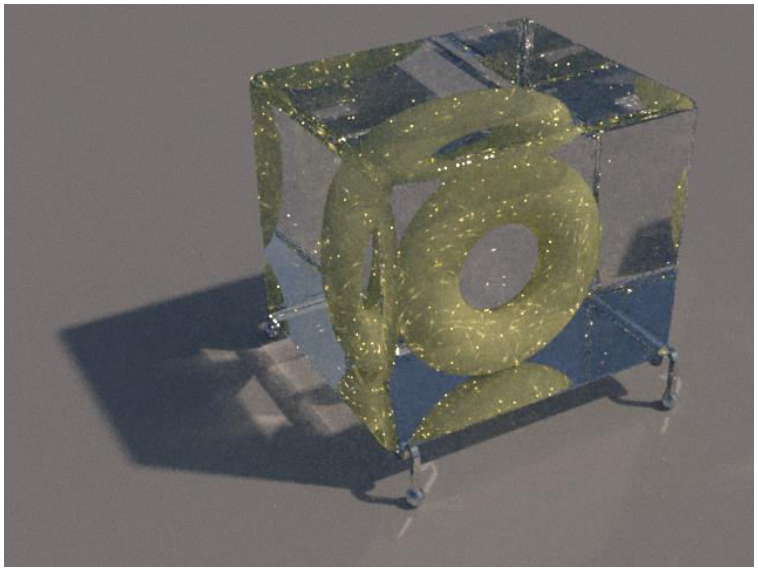
BDPT



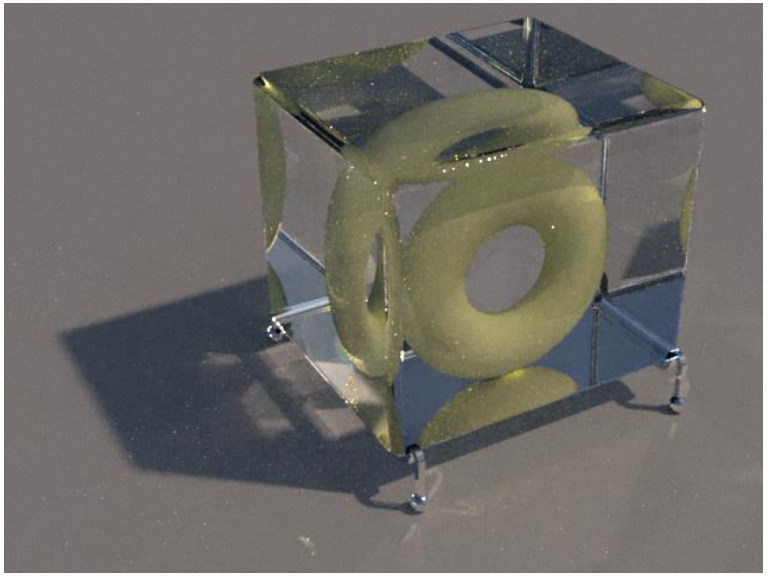
MLT1



ERPT



MLT2

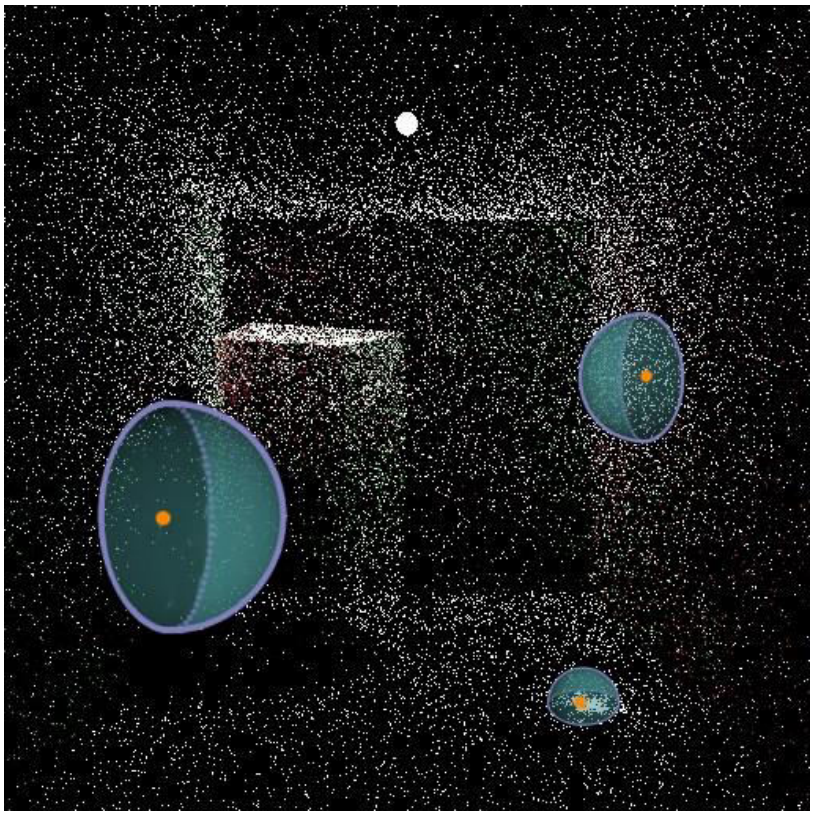


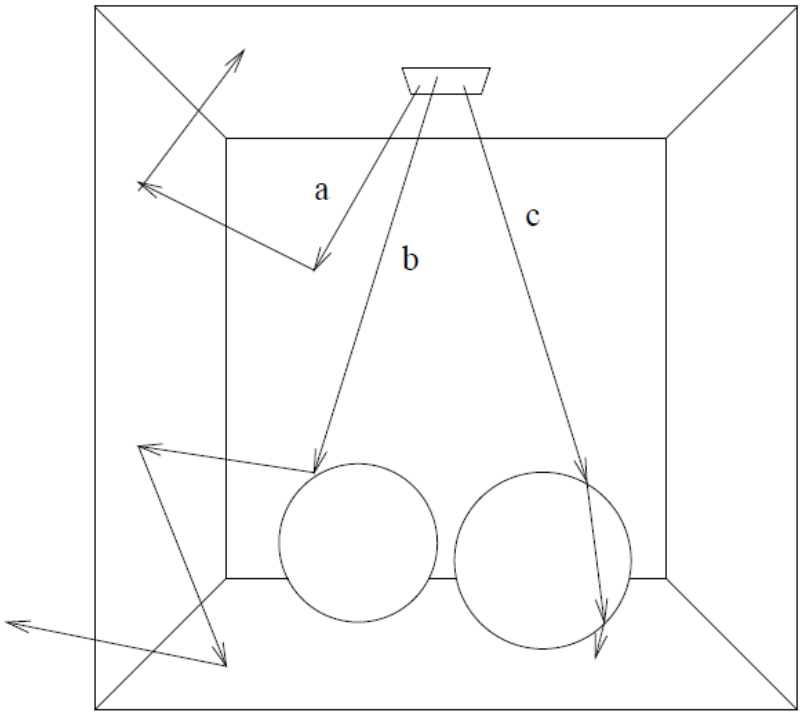
Время рендеринга (~ 20 минут на CPU)

Монте-Карло Трассировка

Фотонные карты

- Трассировка + сбор освещенности





Фотонные карты

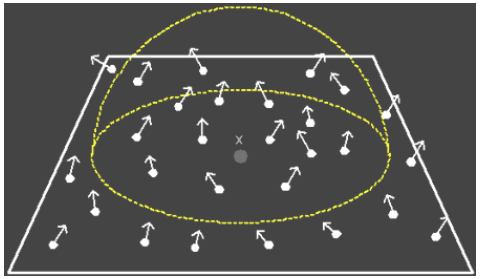
- Дифференциальный поток можно аппроксимировать суммой энергий фотонов в окрестности

$$I(\varphi_r,\theta_r)=\int_{\varphi_i}\int_{\theta_i} L(\varphi_i,\theta_i)R(\varphi_i,\theta_i,\varphi_r,\theta_r)\cos(n,l_{\varphi_i,\theta_i})d\varphi_i d\theta_i$$

$$L(\varphi_i,\theta_i)=\frac{d^2\Phi}{\cos(n,l_{\varphi_i,\theta_i})d\varphi_i d\theta_i dA}$$

$$I(\varphi_r,\theta_r)=\int_{\varphi_i}\int_{\theta_i} \frac{d^2\Phi}{\cos(n,l_{\varphi_i,\theta_i})d\varphi_i d\theta_i dA} R(\varphi_i,\theta_i,\varphi_r,\theta_r)\cos(n,l_{\varphi_i,\theta_i})d\varphi_i d\theta_i$$

$$I(\varphi_r,\theta_r)=\int_{\varphi_i}\int_{\theta_i} R(\varphi_i,\theta_i,\varphi_r,\theta_r)\frac{d^2\Phi}{dA}$$

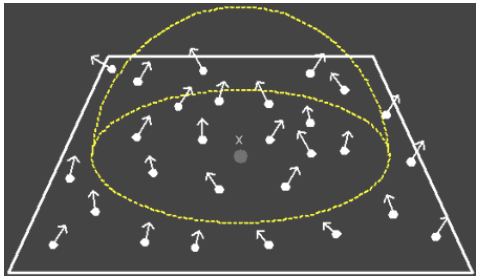


Фотонные карты

- В результате получаем сумму K ближайших фотонов

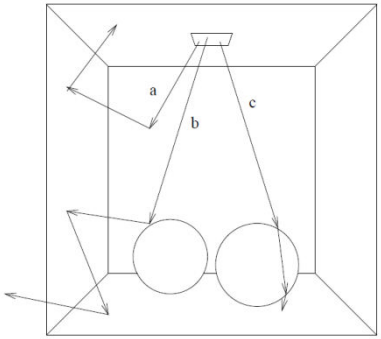
$$I(\varphi_r, \theta_r) = \int_{\varphi_i} \int_{\theta_i} R(\varphi_i, \theta_i, \varphi_r, \theta_r) \frac{d^2\Phi}{dA}$$

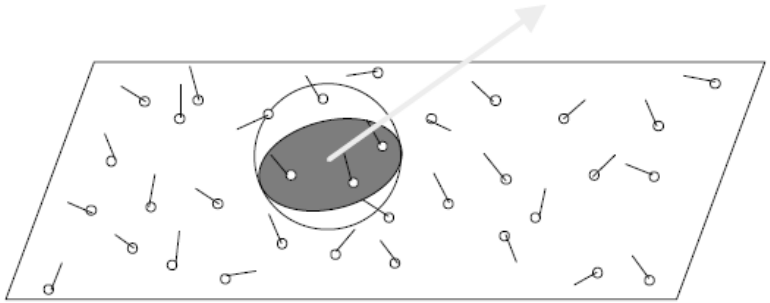
$$I(\varphi_r, \theta_r) \approx \frac{1}{\pi r^2} \sum_{1}^{K} R(\varphi_i, \theta_i, \varphi_r, \theta_r) \Delta\Phi(\varphi_i, \theta_i)$$



Фотонные карты

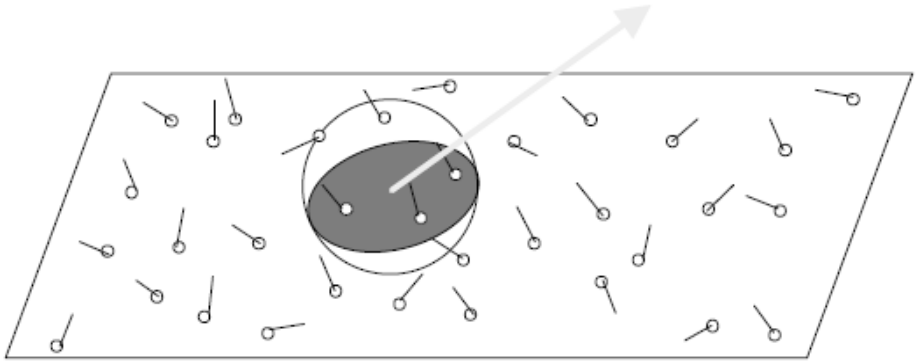
- Трассировка фотонов
- Построение ускоряющих структур
- Обратная трассировка и сбор освещенности
  - Возможна Монте-Карло трассировка (FG)

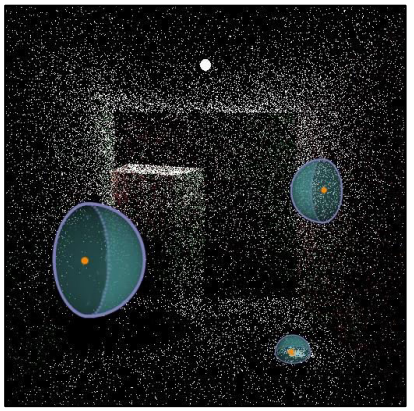




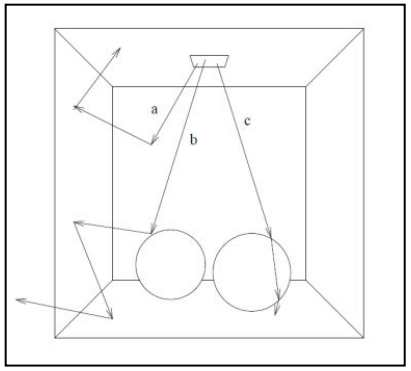
Фотонные карты

- Отличия от Light Tracing и BDPT
  - Фотоны переносят порции энергии
  - Вклад фотона учитывается на площади не 0-го размера!
    - Но трассируются фотоны как обычные лучи
  - Русская рулетка



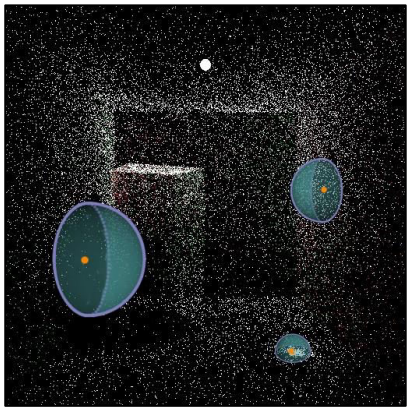


Фотонные карты



- Русская рулетка
  - Не модифицируем энергию фотона при отражении!

$$\begin{aligned} \xi \in [0, d] &\longrightarrow \text{diffuse reflection} \\ \xi \in]d, s+d] &\longrightarrow \text{specular reflection} \\ \xi \in]s+d, 1] &\longrightarrow \text{absorption} \end{aligned}$$



- Сохранение
  - Только на диффузных поверхностях
  - Каждый фотон может быть сохранен много раз

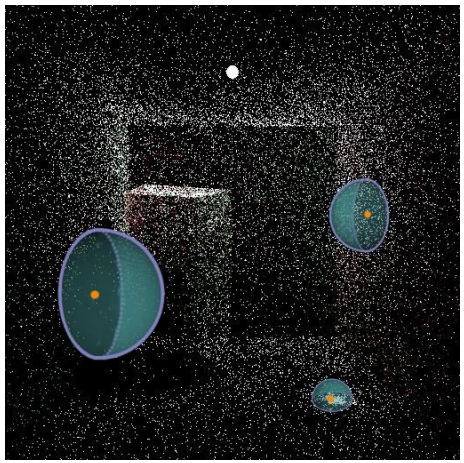
Фотонные карты

- Два метода сбора освещенности
  - k-ближайших
  - Фиксированный радиус
- Два метода визуализации
  - Прямая визуализация
  - Финальный сбор
- Три фотонные карты
  - Диффузная
  - Каустическая
  - Объемная (для объемных эффектов)

Фотонные карты

- К-ближайших

```
kdtree *balance( points ) {
  Find the cube surrounding the points
  Select dimension dim in which the cube is larges
  Find median of the points in dim
  s1 = all points below median
  s2 = all points above median
  node = median
  node.left = balance( s1 )
  node.right = balance( s2 )
  return node
}
```



given the photon map, a position x and a max search distance d^2
this recursive function returns a heap h with the nearest photons.
Call with `locate_photons(1)` *to initiate search at the root of the kd-tree*

```
locate_photons( p ) {
  if ( 2p+1 < number of photons ) {
    examine child nodes
                Compute distance to plane (just a subtract)
    δ = signed distance to splitting plane of node n
    if (δ < 0) {
                We are left of the plane - search left subtree first
      locate_photons( 2p )
      if ( δ² < d² )
        locate_photons( 2p+1 )        check right subtree
    } else {
                We are right of the plane - search right subtree first
      locate_photons( 2p+1 )
      if ( δ² < d² )
        locate_photons( 2p )          check left subtree
    }
  }
              Compute true squared distance to photon
  δ² = squared distance from photon p to x
  if ( δ² < d² ) {              Check if the photon is close enough?
    insert photon into max heap h
              Adjust maximum distance to prune the search
    d² = squared distance to photon in root node of h
  }
}
```

Фотонные карты

- Структуры данных

```
struct Photon // 20 байт
{
  float3 pos;
  char   color[4];
  float  phi, theta;
  short  flag;
}
```

```
uint encodeNormal(float3 n)
{
  short x = (short)(n.x*32767.0f);
  short y = (short)(n.y*32767.0f);

  ushort sign = (n.z >= 0) ? 0 : 1;

  int sx = (int(x & 0xfffe) | sign);
  int sy = (int(y & 0xfffe) << 16 );

  return (sx | sy);
}
```

```
struct Photon // 24 байта
{
  float3  pos;
  uint    compressedNorm;
  half4   color; // или uchar4
}
```

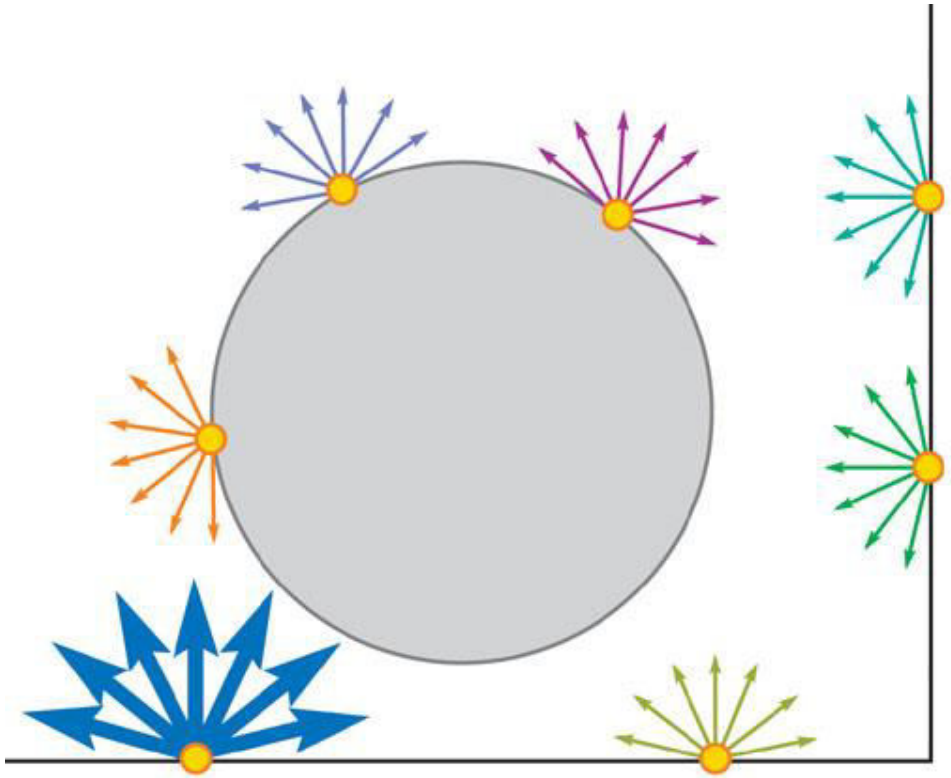
```
float3 decodeNormal(uint a_data)
{
  const float divInv = 1.0f/32767.0f

  short2 a_enc;
  a_enc.x = short(a_data & 0x0000FFFF);
  a_enc.y = short(int(a_data & 0xFFFF0000) >> 16);

  float sign = (a_enc.x & 0x0001) ? -1.0f : 1.0f;
  float x = short(a_enc.x & 0xfffe)*divInv;
  float y = short(a_enc.y & 0xfffe)*divInv;
  float z = sign*sqrtf(fmaxf(1.0f-x*x-y*y, 0.0f));
  return float3(x,y,z);
}
```

Фотонные карты

- Проблемы
  - Память (Сжатие, ppm)
    - Но сжатие может снизить скорость сбора!
  - Скорость (упростить алгоритм сбора)
    - Но это снизит точность!
  - Темные углы (Convex Hull, FG)
    - Снижает скорость
  - Бесполезные фотоны
    - Пометка видимых поверхностей
    - Частицы видимости (память)

Фотонные карты

- Финальный сбор



Set of Rays in Conventional Final Gathering

Фотонные карты

- Бесполезные фотоны
  - Карты проекций
  - Определение видимых поверхностей
    - Простая пометка
    - Частицы видимости

Фотонные карты

- Прогрессивные фотонные карты (PPM)

PPM Radius Update Rule

$$\frac{r_{i+1}^2}{r_i^2} = \frac{N_i + \alpha M_i}{N_i + M_i}$$

Local Statistics

Our Radius Sequence

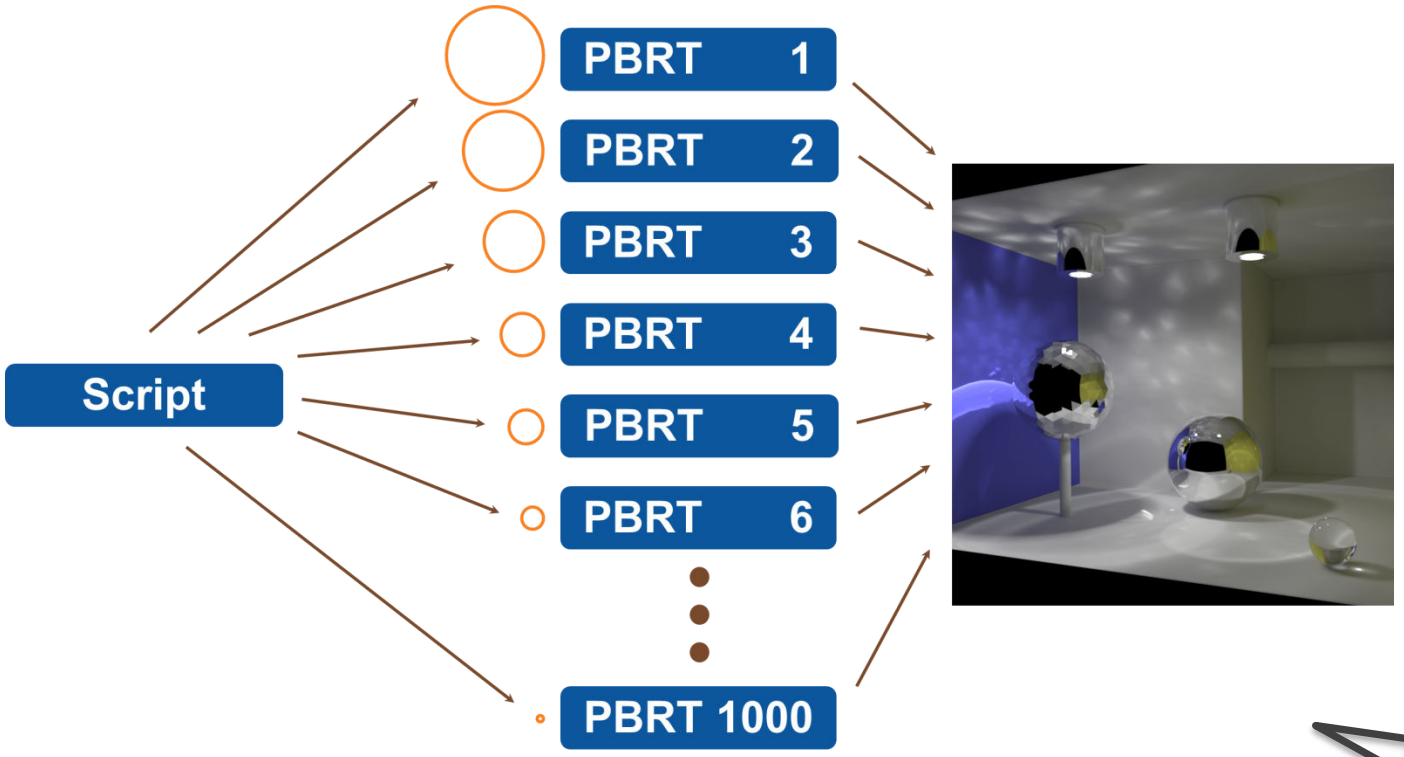
$$\frac{r_{i+1}^2}{r_i^2} = \frac{i + \alpha}{i + 1}$$

No Local Statistics!

http://users-cs.au.dk/toshiya/starpm2012/

Фотонные карты

- Прогрессивные фотонные карты (PPM)



$$\frac{r_{i+1}^2}{r_i^2} = \frac{i+\alpha}{i+1}$$

http://users-cs.au.dk/toshiya/starpm2012/

Фотонные карты

- Прогрессивные фотонные карты (PPM)

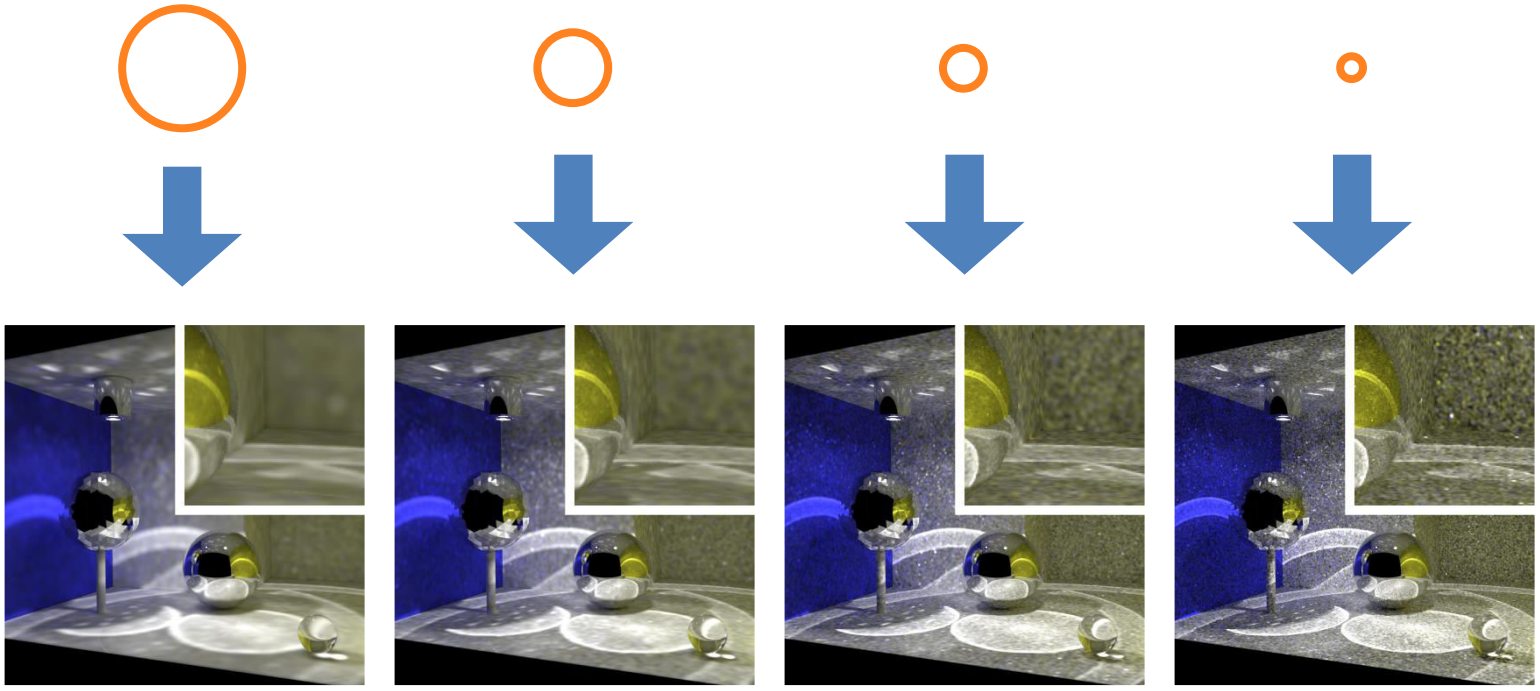
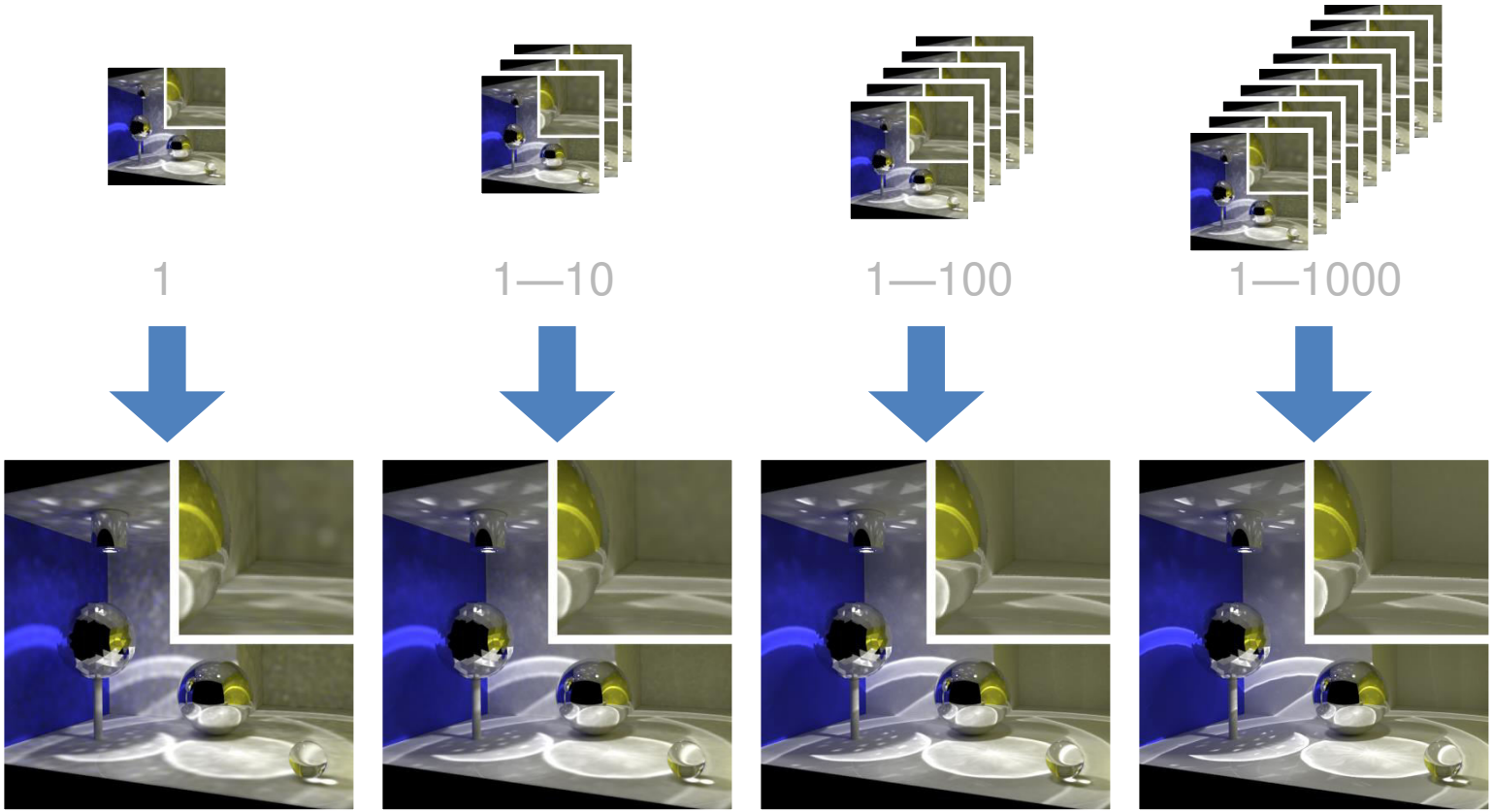


Image 1 Image 10 Image 100 Image 1000

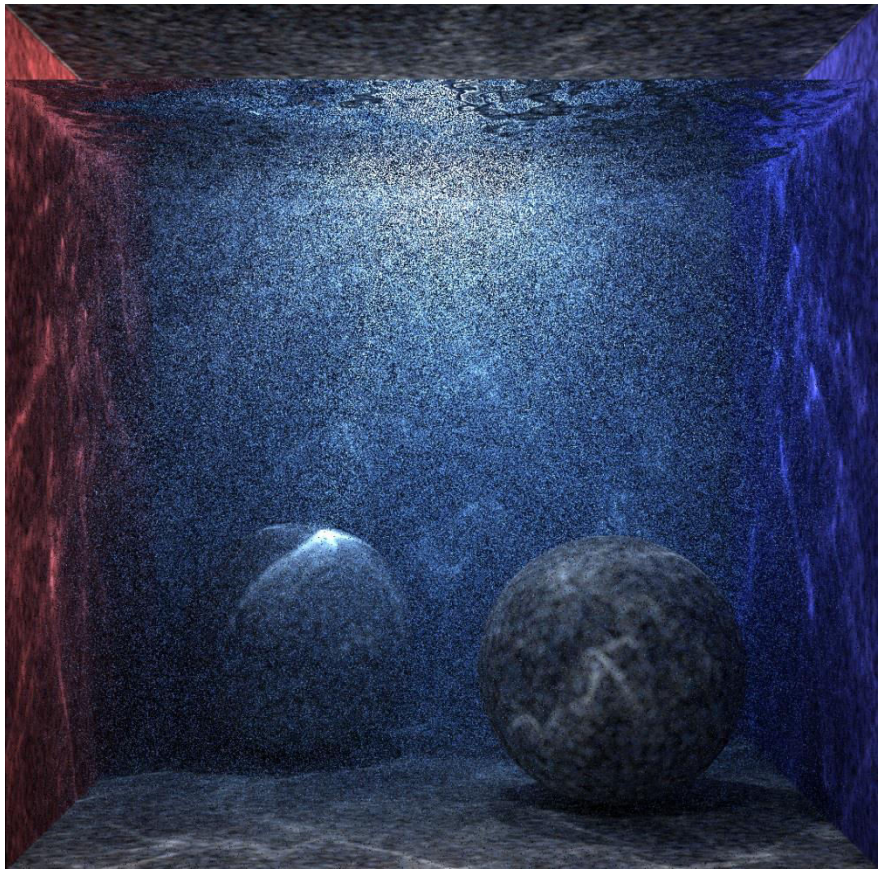
Фотонные карты

- Прогрессивные фотонные карты (PPM)



Фотонные карты

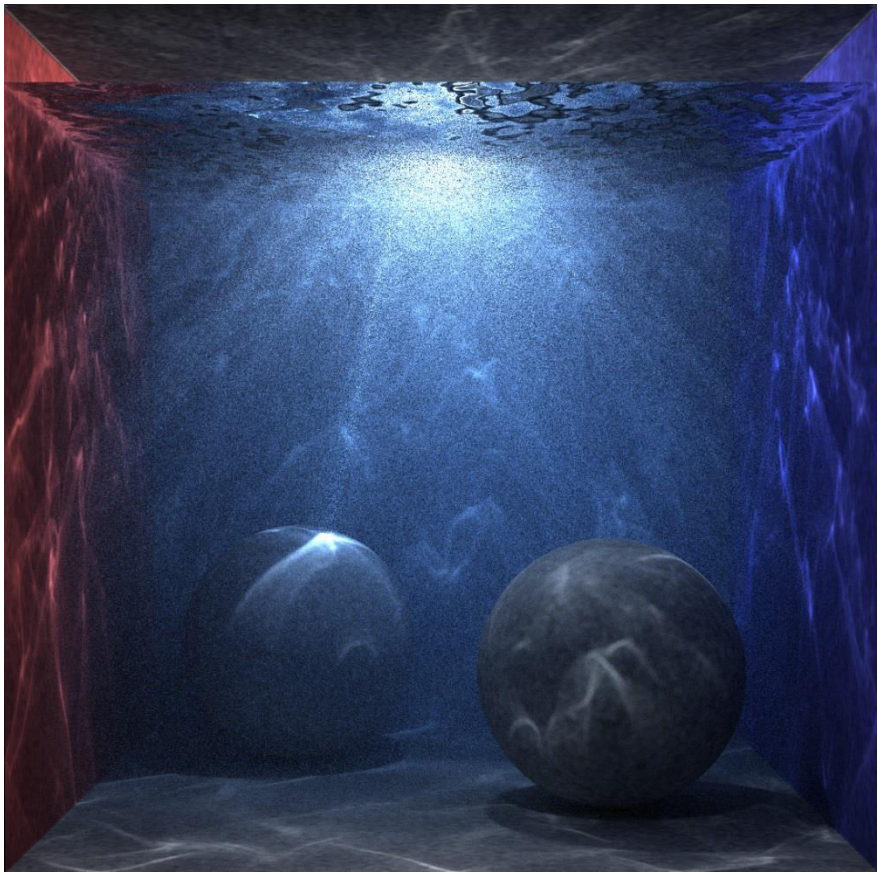
- 2 миллиона фотонов



$$\frac{r_{i+1}^2}{r_i^2} = \frac{i+\alpha}{i+1}$$

Фотонные карты

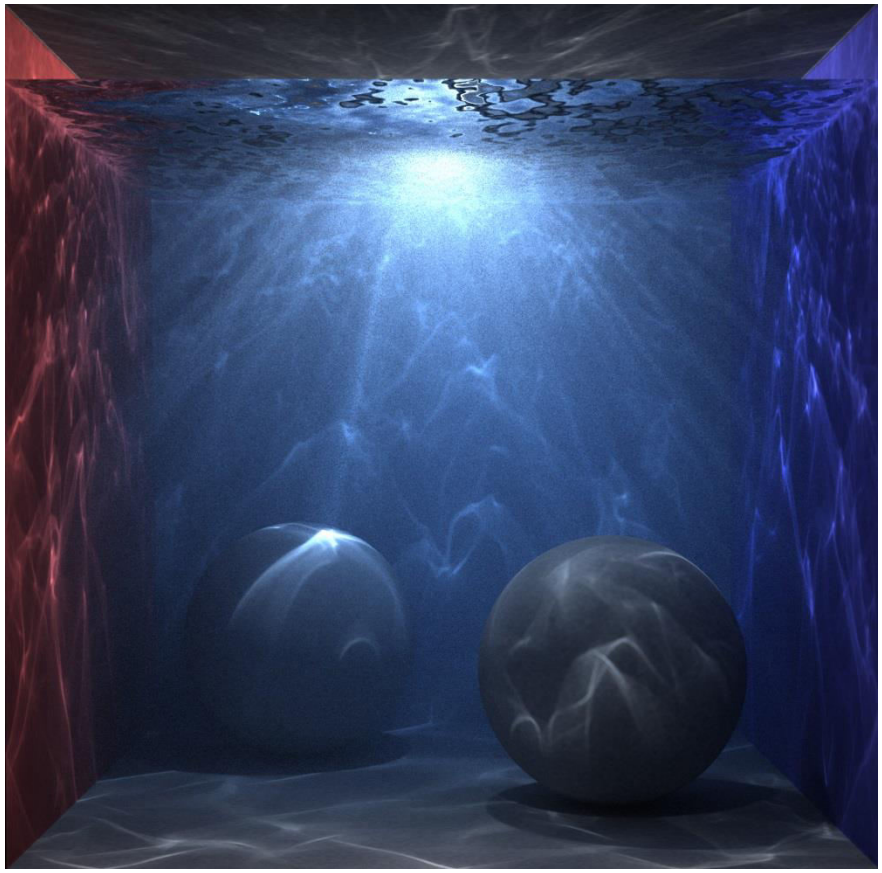
- 20 миллиона фотонов (10 итераций)



$$\frac{r_{i+1}^2}{r_i^2} = \frac{i+\alpha}{i+1}$$

Фотонные карты

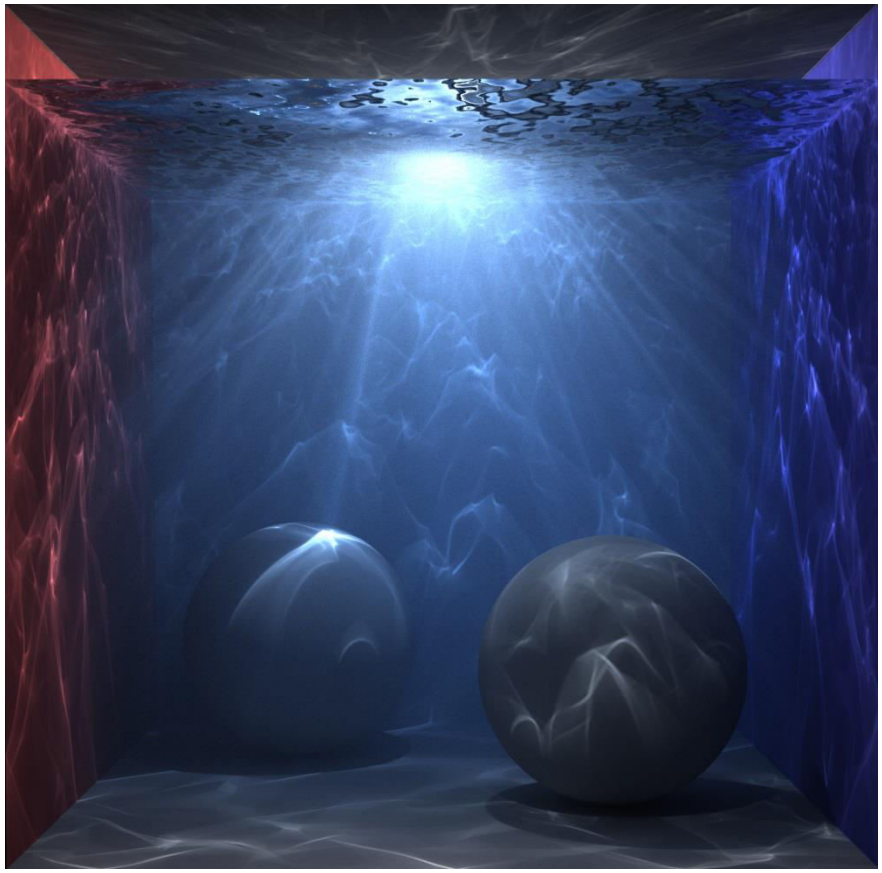
- 200 миллионов фотонов (100 итераций)



$$\frac{r_{i+1}^2}{r_i^2} = \frac{i+\alpha}{i+1}$$

Фотонные карты

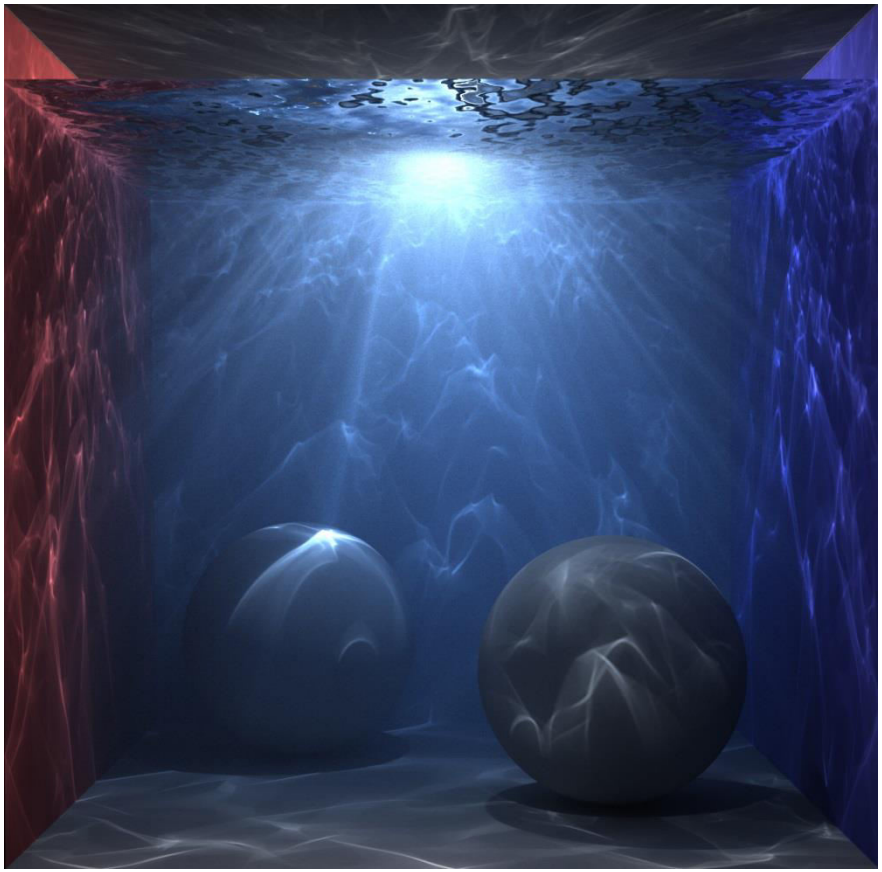
- 2 миллиарда фотонов (1000 итераций)

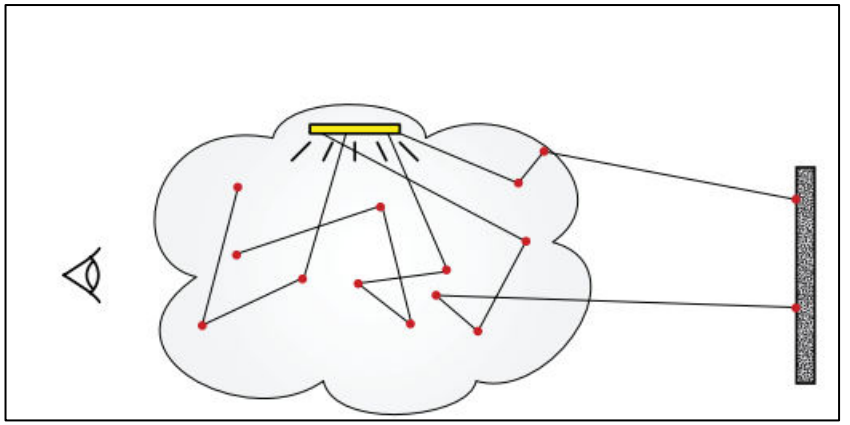


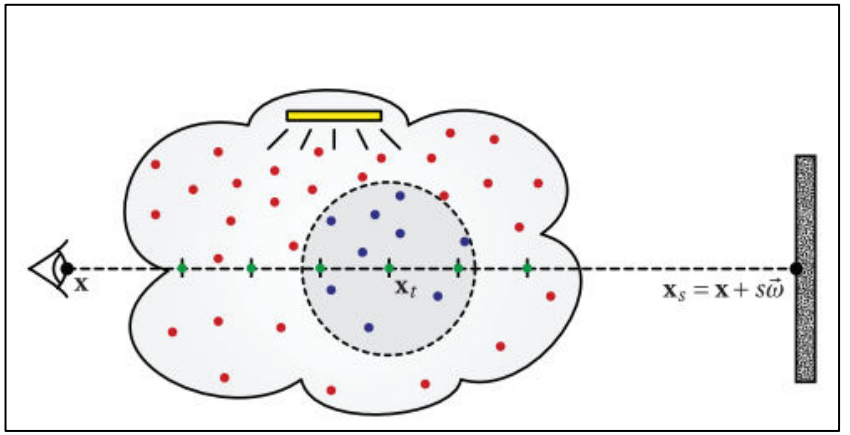
$$\frac{r_{i+1}^2}{r_i^2} = \frac{i+\alpha}{i+1}$$

Фотонные карты

- 2 миллиарда фотонов (1000 итераций)



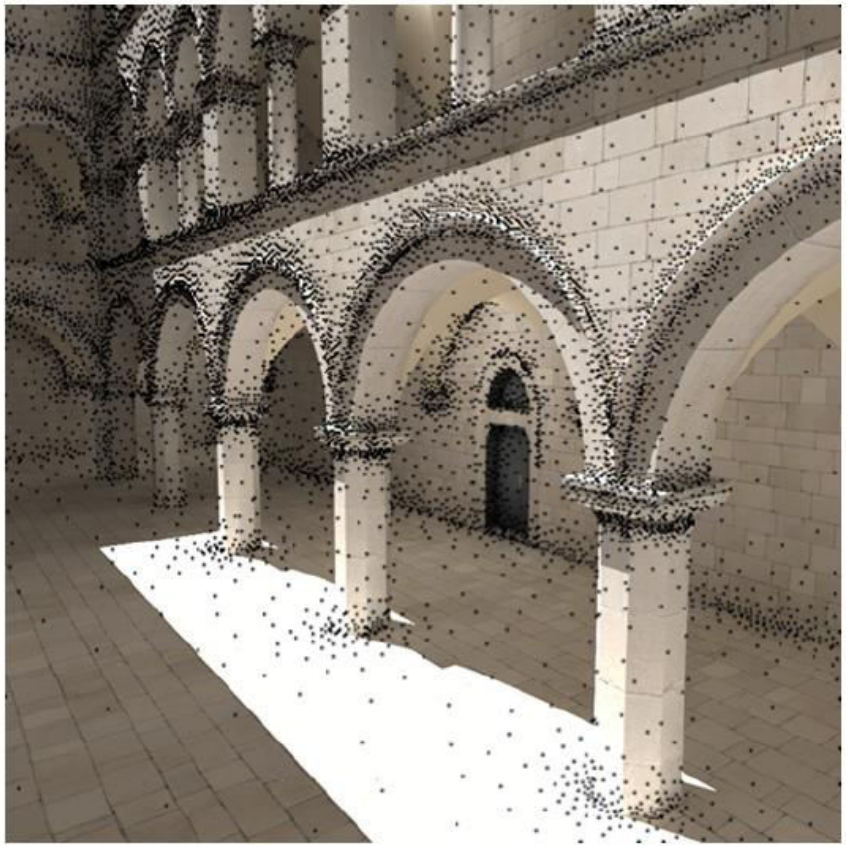




Вопросы

- В чем отличие фотонных карт от BDPT?
  - Русская рулетка, порции энергии, площадь
- В чем особенность русской рулетки?
  - Стохастическое отражение
- Способы сбора освещенности?
  - К-ближайших, фикс. Радиус
- Какие типы фотонных карт используются?
  - Диффузная, каустическая, объемная

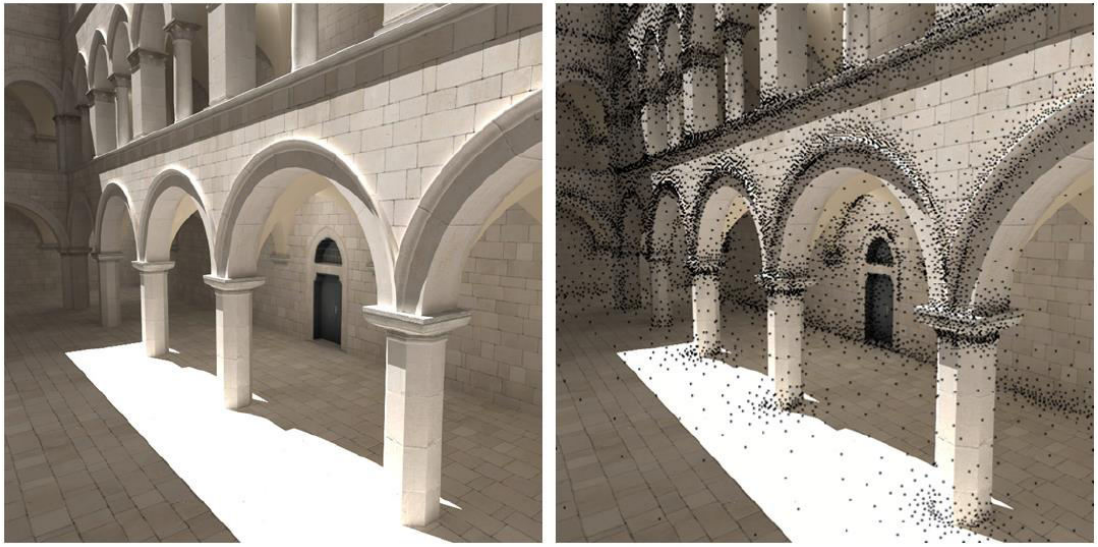
Кэш освещенности



Кэш освещенности

Algorithm 5 Lazy irradiance evaluation used in irradiance caching.

```
function IrradianceCaching(p, n)
    if one or more irradiance values can be used for interpolation at p then
        return irradiance interpolated from the cached values.
    else
        Compute new irradiance value and gradients by hemisphere sampling.
        Store the value with gradients as a new record in the cache.
        return the new irradiance value.
    end if
end function
```

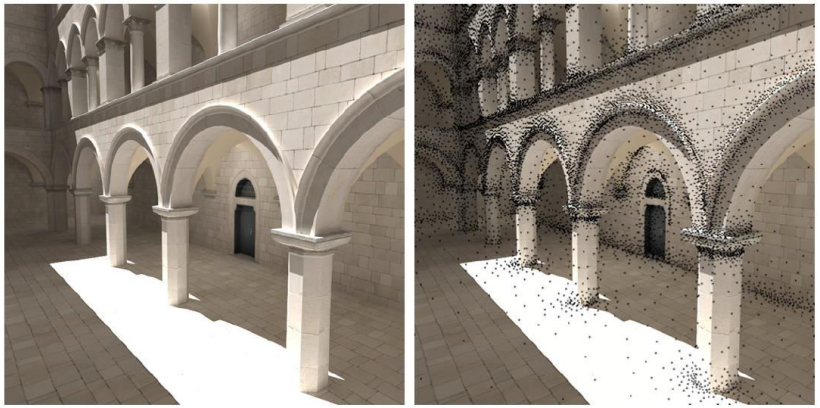


Кэш освещенности

- Тонкости
  - Оценка точности
  - Вычисление радиуса валидности

Algorithm 5 Lazy irradiance evaluation used in irradiance caching.

```
function IrradianceCaching(p, n)
    if one or more irradiance values can be used for interpolation at p then
        return irradiance interpolated from the cached values.
    else
        Compute new irradiance value and gradients by hemisphere sampling.
        Store the value with gradients as a new record in the cache.
        return the new irradiance value.
    end if
end function
```



Кэш освещенности

Кэш освещенности + FG

Кэш освещенности + FG

Параллельность

- Как распараллеливать рассмотренные методы?

Резюме (ИМХО):

Оценка в формате: функциональность/ скорость / простота

- Наиболее практичные подходы:
  - PT (без теневых лучей) 10/2/10
  - PT + LT 8/8/8
  - Просто PPM 10/5/5
  - PT + PPM для каустиков 10/8/5
  - (PT + PM + FG + IC) + PPM для каустиков 10/10/2
    - Для FG необходимо делать Irradiance Maps
    - Или Recursive IC