

Системы компьютерной обработки изображений - Лабораторная работа № 5

Информация

Системы компьютерной обработки изображений

Лабораторная работа № 5

«Формирование изображения методом трассировки путей»

Выполнили студенты: Билошицкий Михаил Владимирович, Р3416 Шпинёва Ульяна Сергеевна, Р3416 Хоробрых Даниил Евгеньевич, Р3416 Нестеренко Ксения Максимовна, Р3416

Группа № Р3416

Преподаватель: Жданов Дмитрий Дмитриевич

Ссылка на репозиторий: [GitHub](#)

Санкт-Петербург, 2025

1. Цель работы

Освоить методы синтеза изображений трёхмерных сцен с глобальным освещением методом трассировки путей (Path Tracing).

2. Задание

Построить изображение сцены Cornell Box с корректным глобальным освещением.

Требования: - Геометрия — треугольная сетка - Материалы — диффузное (Ламберт) и зеркальное отражение - Выборка по значимости для выбора типа отражения - Русская рулетка для ограничения глубины - NEE для прямого освещения - Точечная камера с антиалиасингом - Тонемаппинг и гамма-коррекция ($\gamma = 2.2$) - Сохранение в PPM

3. Описание алгоритма

3.1. Уравнение рендеринга

В основе path tracing лежит уравнение рендеринга Каджии:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega} f_r(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\omega_i \cdot n) d\omega_i$$

- L_o — исходящая яркость из точки x в направлении ω_o
- L_e — собственное излучение (для источников света)
- f_r — BRDF материала
- L_i — входящая яркость
- $(\omega_i \cdot n)$ — косинус угла падения

Интеграл вычисляется методом Монте-Карло — усредняем по случайным направлениям:

$$L_o \approx \frac{1}{N} \sum_{j=1}^N \frac{f_r \cdot L_i \cdot (\omega_j \cdot n)}{p(\omega_j)}$$

В коде это реализовано как цикл по сэмплам для каждого пикселя:

```
for _ in range(samples_per_pixel):
    origin, direction = get_ray(x, y, ...)
    sample_color = trace_path(origin, direction, ...)
    pixel_color = pixel_color + sample_color

image[y, x] = pixel_color / samples_per_pixel
```

3.2. Пересечение луча с треугольником

Для поиска пересечений используется алгоритм Мёллера-Трумбера.

Луч: $R(t) = O + tD$. Точка на треугольнике (V_0, V_1, V_2) в барицентрических координатах:

$$P = (1 - u - v)V_0 + uV_1 + vV_2$$

Приравнивая и решая систему:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix}$$

где $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$, $T = O - V_0$.

Пересечение валидно если $t > 0$, $u \geq 0$, $v \geq 0$, $u + v \leq 1$.

```
@njit
def ray_triangle_intersect(ray_origin, ray_dir, v0, v1, v2):
    EPSILON = 1e-7
    edge1 = v1 - v0
    edge2 = v2 - v0

    h = cross(ray_dir, edge2)           # D × E2
    a = dot(edge1, h)                  # (D × E2) · E1
    if abs(a) < EPSILON: return -1.0  # луч параллелен

    f = 1.0 / a
    s = ray_origin - v0               # T
    u = f * dot(s, h)                 # u
    if u < 0.0 or u > 1.0: return -1.0

    q = cross(s, edge1)               # T × E1
```

```

v = f * dot(ray_dir, q)           # v
if v < 0.0 or u + v > 1.0: return -1.0

t = f * dot(edge2, q)             # t
if t < EPSILON: return -1.0

return t

```

3.3. Материалы и BRDF

Диффузное отражение (Ламберт) — свет рассеивается равномерно во все стороны:

$$f_r^{diffuse} = \frac{\rho_d}{\pi}$$

где ρ_d — альбедо. Деление на π нормализует BRDF для сохранения энергии.

Зеркальное отражение — свет отражается строго по закону отражения:

$$\omega_r = \omega_i - 2(\omega_i \cdot n) \cdot n$$

```
@njit
def reflect(direction, normal):
    return direction - normal * (2.0 * dot(direction, normal))
```

Выбор типа отражения делается случайно, пропорционально весам материала (importance sampling):

```

diff_weight = max(mat_diffuse)
spec_weight = max(mat_specular)
p_diffuse = diff_weight / (diff_weight + spec_weight)

if random() < p_diffuse:
    # диффузное отражение
    new_dir = random_cosine_hemisphere(normal)
    throughput = throughput * mat_diffuse
else:
    # зеркальное отражение
    new_dir = reflect(current_dir, normal)
    throughput = throughput * mat_specular
```

3.4. Косинус-взвешенная выборка направлений

Для диффузных поверхностей эффективнее сэмплировать с учётом косинуса — направления ближе к нормали вносят больший вклад.

Плотность распределения:

$$p(\omega) = \frac{\cos \theta}{\pi}$$

Генерация направления (ξ_1, ξ_2 — случайные числа из $[0, 1]$):

$$\begin{aligned}\phi &= 2\pi\xi_1 \\ \cos \theta &= \sqrt{\xi_2} \\ \sin \theta &= \sqrt{1 - \xi_2}\end{aligned}$$

```
@njit
def random_cosine_hemisphere(normal):
    r1 = random()
    r2 = random()

    phi = 2.0 * np.pi * r1
    cos_theta = np.sqrt(r2)
    sin_theta = np.sqrt(1.0 - r2)

    # локальные координаты
    x = np.cos(phi) * sin_theta
    y = np.sin(phi) * sin_theta
    z = cos_theta

    # переводим в мировые через базис от нормали
    tangent = normalize(cross(up, normal))
    bitangent = cross(normal, tangent)

    return normalize(tangent * x + bitangent * y + normal * z)
```

При такой выборке $\cos \theta$ в числителе и PDF сокращаются — меньше дисперсия.

3.5. Русская рулетка

Чтобы не трассировать бесконечно, после определённой глубины путь обрывается случайно. Вероятность продолжения зависит от отражательной способности материала:

$$p_{continue} = \min(\max(\rho_d + \rho_s), 0.95)$$

Чтобы оценка осталась несмешённой, вес луча корректируется:

```
throughput = throughput / p_continue

if depth >= rr_depth:
    p_continue = min(max(total_refl[0], total_refl[1], total_refl[2]), 0.95)
    if random() > p_continue:
        break # обрываем путь
    throughput = throughput / p_continue # компенсируем
```

3.6. Next Event Estimation (NEE)

Для диффузных поверхностей явно сэмплируем источник света — это сильно уменьшает шум.

Выбираем случайную точку y на источнике. Вклад прямого освещения:

$$L_{direct} = L_e(y) \cdot \frac{\rho_d}{\pi} \cdot G(x, y) \cdot A_{light}$$

Геометрический фактор:

$$G(x, y) = \frac{(\omega \cdot n_x) \cdot (-\omega \cdot n_y)}{|x - y|^2}$$

где $\omega = \frac{y-x}{|y-x|}$ — направление к источнику.

```
# выбираем точку на источнике
light_point, light_normal, light_emission, pdf = sample_light_point(...)

to_light = light_point - hit_point
dist = length(to_light)
light_dir = to_light / dist

cos_theta = dot(normal, light_dir)           # угол на поверхности
cos_theta_light = dot(-light_dir, light_normal) # угол на источнике

if cos_theta > 0 and cos_theta_light > 0:
    # проверяем видимость (shadow ray)
    shadow_t, _, _, _ = intersect_scene(hit_point + offset, light_dir, ...)

    if shadow_t < 0 or shadow_t > dist - 0.001: # путь свободен
        geometry = cos_theta * cos_theta_light / (dist * dist)
        brdf = mat_diffuse / np.pi
        direct = light_emission * brdf * geometry * total_light_area
        color = color + throughput * direct
```

3.7. Постобработка

Рендерер выдаёт HDR-изображение (значения могут быть > 1). Преобразуем в LDR.

Тонемаппинг Рейнхарда:

$$L_{out} = \frac{L_{in}}{1 + L_{in}}$$

Гамма-коррекция (для корректного отображения на мониторе):

$$V_{out} = V_{in}^{1/\gamma}, \quad \gamma = 2.2$$

```

def tonemap_and_gamma(image, gamma=2.2, exposure=1.5):
    image = image * exposure
    image = image / (1.0 + image)      # Reinhard
    image = np.clip(image, 0, 1)
    image = np.power(image, 1.0 / gamma)
    return image

```

3.8. Основной цикл трассировки

Собираем всё вместе:

```

@njit
def trace_path(ray_origin, ray_dir, ..., max_depth, rr_depth):
    color = np.zeros(3)
    throughput = np.ones(3)

    for depth in range(max_depth):
        # 1. Пересечение со сценой
        t, hit_point, normal, mat_id = intersect_scene(...)
        if mat_id < 0: break

        # 2. Попали в источник – берём его цвет
        if any(emission[mat_id] > 0):
            color = color + throughput * emission[mat_id]
            break

        # 3. Русская рулетка
        if depth >= rr_depth:
            if random() > p_continue: break
            throughput = throughput / p_continue

        # 4. Выбор типа отражения
        if random() < p_diffuse:
            # NEE – прямой свет
            color = color + throughput * direct_lighting(...)
            # продолжаем в случайном направлении
            new_dir = random_cosine_hemisphere(normal)
            throughput = throughput * mat_diffuse
        else:
            new_dir = reflect(current_dir, normal)
            throughput = throughput * mat_specular

        current_origin = hit_point + offset
        current_dir = new_dir

    return color

```

4. Параметры

Настройки в main.py:

```
CONFIG = {
    'width': 600,
    'height': 600,
    'samples_per_pixel': 512,
    'max_depth': 6,
    'russian_roulette_depth': 2,

    'camera_position': [0, 2.5, -8],
    'camera_look_at': [0, 2.5, 0],
    'camera_fov': 45,

    'room_size': 5.0,
    'left_wall_color': [0.75, 0.25, 0.25],
    'right_wall_color': [0.25, 0.25, 0.75],
    'wall_color': [0.75, 0.75, 0.75],

    'light_intensity': 15.0,
    'light_size': 1.8,

    'box1_diffuse': [0.75, 0.75, 0.75],
    'box2_specular': [0.9, 0.9, 0.9],

    'gamma': 2.2,
    'exposure': 1.5,
}
```

5. Результат

Эффекты глобального освещения: - Color bleeding — красный и синий оттенки от стен на кубах и полу - Мягкие тени от площадного источника - Отражения в зеркальном кубе

6. Вывод

Реализован path tracer с NEE и русской рулеткой. Получено физически корректное изображение Cornell Box с глобальным освещением.

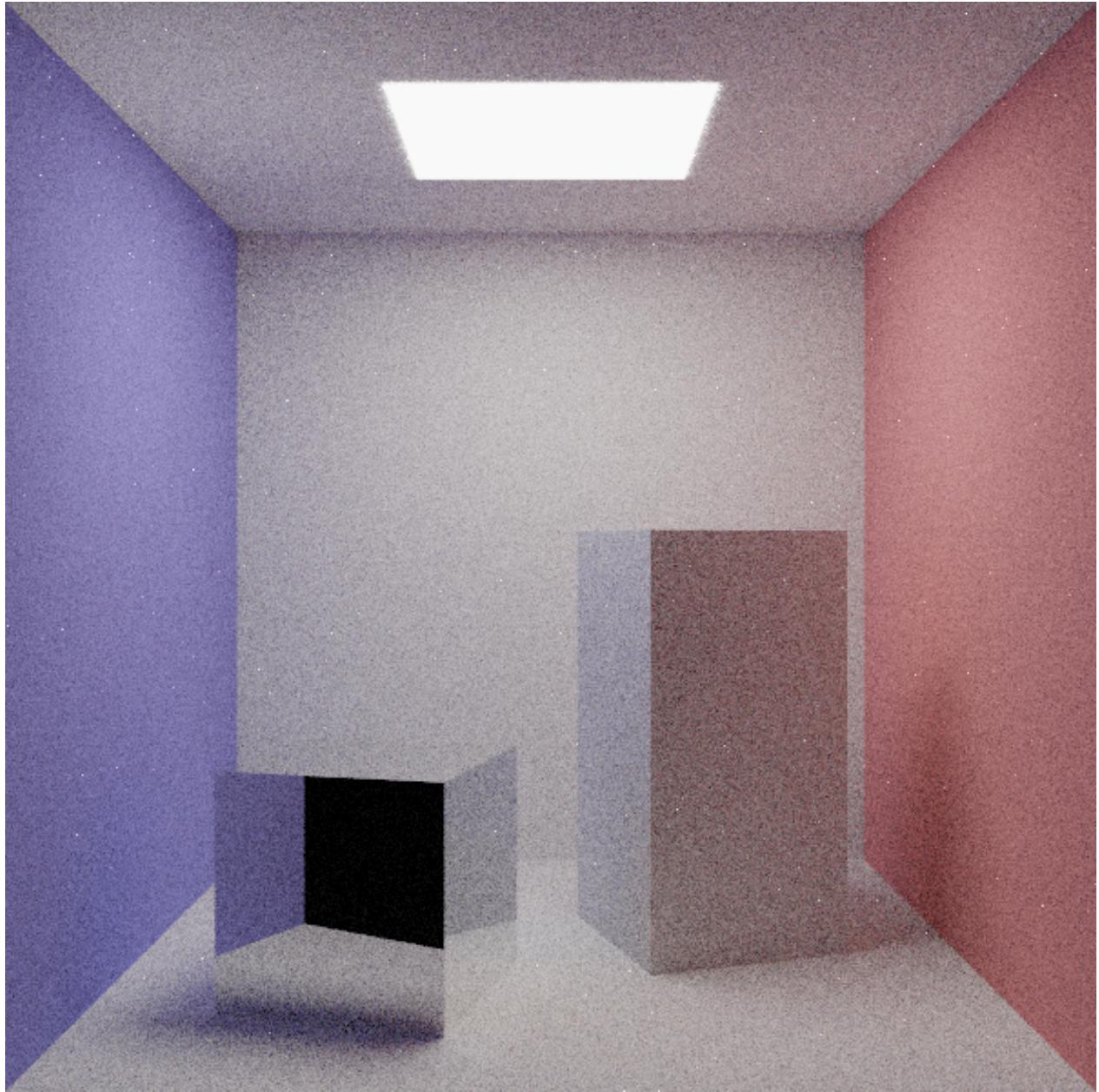


Figure 1: Результат