

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО
Факультет Программной Инженерии и Компьютерной Техники

Системы компьютерной обработки изображений

Лабораторная работа № 1

«Вычисление освещенности в точках поверхности»

Выполнил студент
Нодири Хисравхон

Группа № Р3431
Преподаватель: Жданов Дмитрий Дмитриевич

г. Санкт-Петербург 2025

Цель

Изучить, как вычисляется освещенность в точках точки на плоскости треугольника при освещении его точечным источником света с заданной диаграммой излучения.

Используемые формулы

1. «Цветная» интенсивность источника под углом к оси источника света:

$I(RGB, \vec{s}) = I_0(RGB) \cos \theta$, где $I_0(RGB)$ – «цветная» интенсивность источника света в направлении его оси \vec{O} , $\|\vec{O}\| = 1$, θ - угол между направлением распространения света и осью источника света, $\cos \theta$ – диаграмма излучения.

2. «Цветная» освещенность точки:

$E(RGB, \vec{P}_T) = \frac{I(RGB, \vec{s}) \cos \alpha}{R^2}$, где $I(RGB, \vec{s})$ – «цветная» интенсивность света, α - угол между направлением света и нормалью к освещаемой поверхности, R - расстояние от источника света до рассматриваемой точки.

3. Перевод локальных координат точки в плоскости в глобальные

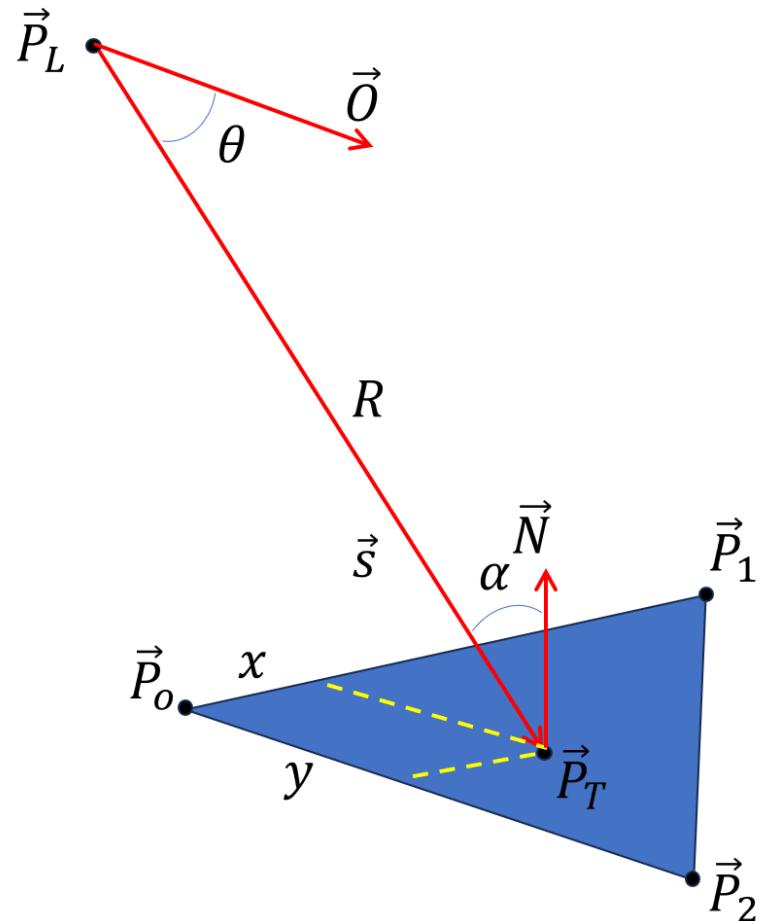
$\vec{P}_T = \vec{P}_0 + \left(\frac{\vec{P}_1 - \vec{P}_0}{\|\vec{P}_1 - \vec{P}_0\|} \cdot x + \frac{\vec{P}_2 - \vec{P}_0}{\|\vec{P}_2 - \vec{P}_0\|} \cdot y \right)$, где x и y смещения по ребрам треугольника.

4. Вычисление вектора нормали плоскости треугольника через 3 точки:

$$\vec{N} = \frac{(\vec{P}_2 - \vec{P}_0) \times (\vec{P}_1 - \vec{P}_0)}{\|(\vec{P}_2 - \vec{P}_0) \times (\vec{P}_1 - \vec{P}_0)\|}$$

5. Вектор от точки плоскости до источника света:

$$\vec{s} = \vec{P}_T - \vec{P}_L, R^2 = \|\vec{s}\|^2, \cos \alpha = \frac{\vec{s} \cdot \vec{N}}{\|\vec{s}\|}, \cos \theta = \frac{\vec{s} \cdot \vec{O}}{\|\vec{s}\|}$$



Шаги алгоритма

1. Вектор от точки до источника света
2. Вычисление углов
3. Интенсивность с учетом диаграммы направленности
4. Освещенность точки

Программа для расчета освещенности

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
import sys
import io

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

def calculate_color_intensity(I0_RGB, direction_vector, cos_theta):
    return I0_RGB * cos_theta

def calculate_point_illumination(I_RGB_s, R):
    return I_RGB_s / (R ** 2)

def calculate_triangle_normal(P0, P1, P2):
    edge1 = P1 - P0
    edge2 = P2 - P0
    normal = np.cross(edge1, edge2)
    return normal / np.linalg.norm(normal)

def convert_local_to_global(P0, P1, P2, x, y):
    edge1 = P1 - P0
    edge2 = P2 - P0
    P_T = P0 + (edge1 / np.linalg.norm(edge1)) * x + (edge2 /
    np.linalg.norm(edge2)) * y
    return P_T

def calculate_illumination_at_point(I0_RGB, direction_O, P_L, P_T,
normal_N):
    s_vector = P_L - P_T
    R = np.linalg.norm(s_vector)

    if R < 1e-10:
        return np.array([0.0, 0.0, 0.0])

    s_normalized = s_vector / R
    cos_alpha = np.dot(s_normalized, normal_N)
    cos_theta = np.dot(s_normalized, -direction_O)

    if cos_alpha <= 0 or cos_theta <= 0:
        return np.array([0.0, 0.0, 0.0])

    I_RGB_s = calculate_color_intensity(I0_RGB, s_normalized, cos_theta)
    E_RGB = calculate_point_illumination(I_RGB_s * cos_alpha, R)

    return E_RGB

def generate_grid_points(num_x=5, num_y=5, max_x=1.0, max_y=1.0):
```

```

x_values = np.linspace(0, max_x, num_x)
y_values = np.linspace(0, max_y, num_y)
return x_values, y_values

def calculate_illumination_grid(I0_RGB, direction_O, P_L, P0, P1, P2,
x_values, y_values):
    normal_N = calculate_triangle_normal(P0, P1, P2)

    results = []
    for y in y_values:
        row = []
        for x in x_values:
            P_T = convert_local_to_global(P0, P1, P2, x, y)
            E_RGB = calculate_illumination_at_point(I0_RGB, direction_O,
P_L, P_T, normal_N)
            row.append(E_RGB)
        results.append(row)

    return np.array(results), normal_N

def visualize_example_points(I0_RGB, direction_O, P_L, P0, P1, P2,
normal_N):
    example_points = [
        (0.0, 0.0, "P0 (вершина)", 'red'),
        (1.5, 1.25, "центр", 'orange'),
        (3.0, 0.0, "P0-P1", 'magenta')
    ]

    fig = plt.figure(figsize=(18, 6))
    fig.canvas.manager.set_window_title('Примеры расчетов освещенности')

    for idx, (x_local, y_local, description, color) in
enumerate(example_points):
        ax = fig.add_subplot(1, 3, idx + 1, projection='3d')

        triangle = np.array([P0, P1, P2])
        poly = Poly3DCollection([triangle], alpha=0.3,
facecolor='lightblue', edgecolor='black', linewidths=2)
        ax.add_collection3d(poly)

        ax.scatter(*P0, color='blue', s=80, marker='o', alpha=0.5)
        ax.scatter(*P1, color='blue', s=80, marker='o', alpha=0.5)
        ax.scatter(*P2, color='blue', s=80, marker='o', alpha=0.5)

        P_T = convert_local_to_global(P0, P1, P2, x_local, y_local)
        ax.scatter(*P_T, color=color, s=200, marker='o', label=f'Pt
({description})', edgecolors='black', linewidths=2)
        ax.scatter(*P_L, color='yellow', s=250, marker='*', label='Pl
(light)', edgecolors='orange', linewidths=2)

        s_vector = P_L - P_T
        ax.quiver(P_T[0], P_T[1], P_T[2], s_vector[0], s_vector[1],
s_vector[2],

```

```

            color='purple', arrow_length_ratio=0.1, linewidth=2,
label='s (to light)')
    ax.quiver(P_T[0], P_T[1], P_T[2], normal_N[0], normal_N[1],
normal_N[2],
            color='red', arrow_length_ratio=0.3, linewidth=2,
label='N (normal)')
    ax.quiver(P_L[0], P_L[1], P_L[2], direction_O[0] * 0.8,
direction_O[1] * 0.8, direction_O[2] * 0.8,
            color='orange', arrow_length_ratio=0.3, linewidth=2,
label='O (dir)', alpha=0.7)

    R = np.linalg.norm(s_vector)
    s_normalized = s_vector / R
    cos_alpha = np.dot(s_normalized, normal_N)
    cos_theta = np.dot(s_normalized, -direction_O)
    I_RGB_s = I0_RGB * cos_theta
    E_RGB = I_RGB_s * cos_alpha / (R ** 2)

    ax.set_xlabel('X', fontsize=10)
    ax.set_ylabel('Y', fontsize=10)
    ax.set_zlabel('Z', fontsize=10)
    ax.set_title(f'{description}\nR={R:.2f}, cos(θ)={cos_theta:.3f},\ncos(α)={cos_alpha:.3f}\nE(R,G,B)=({E_RGB[0]:.2f}, {E_RGB[1]:.2f},\n{E_RGB[2]:.2f})',
                 fontsize=11, fontweight='bold')
    ax.legend(loc='upper left', fontsize=8)

    all_points = np.vstack([P0, P1, P2, P_L])
    max_range = np.array([all_points[:, 0].max() - all_points[:, 0].min(),
                         all_points[:, 1].max() - all_points[:, 1].min(),
                         all_points[:, 2].max() - all_points[:, 2].min()]).max() / 2.0
    mid_x = (all_points[:, 0].max() + all_points[:, 0].min()) * 0.5
    mid_y = (all_points[:, 1].max() + all_points[:, 1].min()) * 0.5
    mid_z = (all_points[:, 2].max() + all_points[:, 2].min()) * 0.5
    ax.set_xlim(mid_x - max_range, mid_x + max_range)
    ax.set_ylim(mid_y - max_range, mid_y + max_range)
    ax.set_zlim(mid_z - max_range, mid_z + max_range)

plt.tight_layout()

def visualize_illumination_heatmap(I0_RGB, direction_O, P_L, P0, P1, P2,
normal_N, results, x_values, y_values):
    fig = plt.figure(figsize=(15, 5))
    fig.canvas.manager.set_window_title('Тепловая карта освещенности')

    channels = ['Red', 'Green', 'Blue']
    channel_idx = [0, 1, 2]

    for idx, (channel, ch_idx) in enumerate(zip(channels, channel_idx)):
        ax = fig.add_subplot(1, 3, idx + 1)

```

```

    illumination_data = results[:, :, ch_idx]

    im = ax.imshow(illumination_data, cmap='hot',
interpolation='bilinear', origin='lower',
                           extent=[x_values[0], x_values[-1], y_values[0],
y_values[-1]], aspect='auto')

    contours = ax.contour(x_values, y_values, illumination_data,
levels=8, colors='white', alpha=0.4, linewidths=0.5)
    ax.clabel(contours, inline=True, fontsize=8, fmt='%.2f')

    ax.set_xlabel('x (локальные координаты)', fontsize=10)
    ax.set_ylabel('y (локальные координаты)', fontsize=10)
    ax.set_title(f'Освещенность - {channel} канал\nE_{channel}(x,y)',
fontsize=12, fontweight='bold')

    cbar = plt.colorbar(im, ax=ax)
    cbar.set_label(f'E_{channel}', rotation=270, labelpad=15)

example_points = [(0.0, 0.0), (1.5, 1.25), (3.0, 0.0)]
for x, y in example_points:
    if x <= x_values[-1] and y <= y_values[-1]:
        ax.plot(x, y, 'o', color='cyan', markersize=8,
markeredgecolor='white', markeredgewidth=2)

plt.tight_layout()

def visualize_scene(P_L, direction_O, P0, P1, P2, normal_N):
    fig = plt.figure(figsize=(12, 9))
    fig.canvas.manager.set_window_title('Освещение треугольника точечным
источником света')
    ax = fig.add_subplot(111, projection='3d')

    triangle = np.array([P0, P1, P2])
    poly = Poly3DCollection([triangle], alpha=0.6, facecolor='blue',
edgecolor='black', linewidths=2)
    ax.add_collection3d(poly)

    ax.scatter(*P0, color='blue', s=100, marker='o', label='P0')
    ax.scatter(*P1, color='blue', s=100, marker='o', label='P1')
    ax.scatter(*P2, color='blue', s=100, marker='o', label='P2')

    P_T_center = (P0 + P1 + P2) / 3
    ax.scatter(*P_T_center, color='red', s=100, marker='o', label='P_t
(center)')
    ax.scatter(*P_L, color='yellow', s=200, marker='*', label='P_l
(light)', edgecolors='orange', linewidths=2)

    ax.quiver(P_T_center[0], P_T_center[1], P_T_center[2], normal_N[0],
normal_N[1], normal_N[2],
                           color='red', arrow_length_ratio=0.3, linewidth=2, label='N
(normal)')

```

```

    ax.quiver(P_L[0], P_L[1], P_L[2], direction_O[0], direction_O[1],
direction_O[2],
              color='orange', arrow_length_ratio=0.3, linewidth=2,
label='O (direction)')

    s_vector = P_L - P_T_center
    ax.quiver(P_T_center[0], P_T_center[1], P_T_center[2], s_vector[0],
s_vector[1], s_vector[2],
              color='purple', arrow_length_ratio=0.1, linewidth=1.5,
label='s (to light)', alpha=0.7)

    edge1_norm = (P1 - P0) / np.linalg.norm(P1 - P0) * 0.5
    edge2_norm = (P2 - P0) / np.linalg.norm(P2 - P0) * 0.5
    ax.quiver(P0[0], P0[1], P0[2], edge1_norm[0], edge1_norm[1],
edge1_norm[2],
              color='green', arrow_length_ratio=0.3, linewidth=1.5,
label='x-axis', linestyle='--')
    ax.quiver(P0[0], P0[1], P0[2], edge2_norm[0], edge2_norm[1],
edge2_norm[2],
              color='cyan', arrow_length_ratio=0.3, linewidth=1.5,
label='y-axis', linestyle='--')

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('Освещение треугольника точечным источником света')
    ax.legend(loc='upper left', fontsize=8)

    all_points = np.vstack([P0, P1, P2, P_L])
    max_range = np.array([all_points[:, 0].max() - all_points[:, 0].min(),
all_points[:, 1].max() - all_points[:, 1].min(),
all_points[:, 2].max() - all_points[:, 2].min()]).max() / 2.0
    mid_x = (all_points[:, 0].max() + all_points[:, 0].min()) * 0.5
    mid_y = (all_points[:, 1].max() + all_points[:, 1].min()) * 0.5
    mid_z = (all_points[:, 2].max() + all_points[:, 2].min()) * 0.5
    ax.set_xlim(mid_x - max_range, mid_x + max_range)
    ax.set_ylim(mid_y - max_range, mid_y + max_range)
    ax.set_zlim(mid_z - max_range, mid_z + max_range)

    plt.tight_layout()

def print_illumination_table(x_values, y_values, results):
    print("\n" + "*100)
    print("ТАБЛИЦА ОСВЕЩЕННОСТИ E(RGB, P_T) для точек на треугольнике")
    print("*100)

    y_x_label = 'y \\ x'
    header = f"{y_x_label:>10}"
    for x in x_values:
        header += f" | {f'x={x:.2f}':>20}"
    print(header)

```

```

print("-" * 100)

for i, y in enumerate(y_values):
    row = f'{y:.2f}'[:>10]
    for j, x in enumerate(x_values):
        E_RGB = results[i, j]
        rgb_str = f'({E_RGB[0]:.4f}, {E_RGB[1]:.4f}, {E_RGB[2]:.4f})'
        row += f' | {rgb_str:>20}'
    print(row)

print("*100)

def main():
    I0_RGB = np.array([100.0, 80.0, 60.0])

    P0 = np.array([0.0, 0.0, 0.0])
    P1 = np.array([4.0, 0.0, 0.0])
    P2 = np.array([2.0, 3.0, 0.0])

    P_L = np.array([2.0, 2.0, 5.0])

    triangle_center = (P0 + P1 + P2) / 3
    direction_O = triangle_center - P_L
    direction_O = direction_O / np.linalg.norm(direction_O)

    print("*100)
    print("ВХОДНЫЕ ДАННЫЕ")
    print("*100)
    print(f'I0(RGB) = {I0_RGB}')
    print(f'O (направление) = {direction_O}')
    print(f'P1 (источник света) = {P_L}')
    print(f'P0 = {P0}')
    print(f'P1 = {P1}')
    print(f'P2 = {P2}')
    print("*100)

    x_values, y_values = generate_grid_points(num_x=5, num_y=5,
max_x=3.0, max_y=2.5)

    results, normal_N = calculate_illumination_grid(I0_RGB, direction_O,
P_L, P0, P1, P2, x_values, y_values)

    print(f'\nВектор нормали N = {normal_N}')
    print(f'||N|| = {np.linalg.norm(normal_N)}')

    print_illumination_table(x_values, y_values, results)

print("\n" + "*100)
print("ПРИМЕРЫ РАСЧЕТОВ ДЛЯ НЕСКОЛЬКИХ ТОЧЕК")
print("*100)

example_points = [
    (0.0, 0.0, "P0 (вершина)",


```

```

(1.5, 1.25, "центр треугольника"),
(3.0, 0.0, "вдоль стороны P0-P1")
]

for x_local, y_local, description in example_points:
    print("\n" + "-"*100)
    print(f"ТОЧКА: {description} | Локальные координаты: x = {x_local}, y = {y_local}")
    print("-"*100)

    P_T = convert_local_to_global(P0, P1, P2, x_local, y_local)
    print(f"Pt (глобальные координаты) = {P_T}")

    s_vector = P_L - P_T
    R = np.linalg.norm(s_vector)

    if R < 1e-10:
        print("Точка совпадает с источником света!")
        continue

    s_normalized = s_vector / R
    cos_alpha = np.dot(s_normalized, normal_N)
    cos_theta = np.dot(s_normalized, -direction_O)

    print(f"\nШаг 1: Вектор от точки до источника света")
    print(f"    s = Pt - PL = {s_vector}")
    print(f"    R = ||s|| = √(s02 + s12 + s22) = √({s_vector[0]:.2f}2 +
{s_vector[1]:.2f}2 + {s_vector[2]:.2f}2) = {R:.4f}")

    print(f"\nШаг 2: Вычисление углов")
    print(f"    cos(θ) = (s · (-O)) / ||s|| = {cos_theta:.4f}")
    print(f"        где θ - угол между вектором s и осью источника О")
    print(f"    cos(α) = (s · N) / ||s|| = {cos_alpha:.4f}")
    print(f"        где α - угол между вектором s и нормалью N к
поверхности")

    if cos_alpha <= 0:
        print(f"    E(RGB, Pt) = (0.0, 0.0, 0.0)")
        continue

    if cos_theta <= 0:
        print(f"    E(RGB, Pt) = (0.0, 0.0, 0.0)")
        continue

    I_RGB_s = I0_RGB * cos_theta
    print(f"\nШаг 3: Интенсивность с учетом диаграммы
направленности")
    print(f"    I(RGB, s) = I0(RGB) × cos(θ)")
    print(f"    I(RGB, s) = {I0_RGB} × {cos_theta:.4f}")
    print(f"    I(RGB, s) = {I_RGB_s}")

    E_RGB = I_RGB_s * cos_alpha / (R ** 2)
    print(f"\nШаг 4: Освещенность точки")

```

```

        print(f" E(RGB, P_t ) = I(RGB, s) × cos(α) / R²")
        print(f" E(RGB, P_t ) = {I_RGB_s} × {cos_alpha:.4f} / {R:.4f}²")
        print(f" E(RGB, P_t ) = {I_RGB_s} × {cos_alpha:.4f} / {R**2:.4f}")
        print(f" E(RGB, P_t ) = ({E_RGB[0]:.6f}, {E_RGB[1]:.6f},
{E_RGB[2]:.6f}))")

    print("\n" + "="*100)

    visualize_scene(P_L, direction_O, P0, P1, P2, normal_N)
    visualize_example_points(I0_RGB, direction_O, P_L, P0, P1, P2,
normal_N)

    x_dense = np.linspace(0, 3.0, 30)
    y_dense = np.linspace(0, 2.5, 25)
    results_dense, _ = calculate_illumination_grid(I0_RGB, direction_O,
P_L, P0, P1, P2, x_dense, y_dense)
    visualize_illumination_heatmap(I0_RGB, direction_O, P_L, P0, P1, P2,
normal_N, results_dense, x_dense, y_dense)

plt.show()

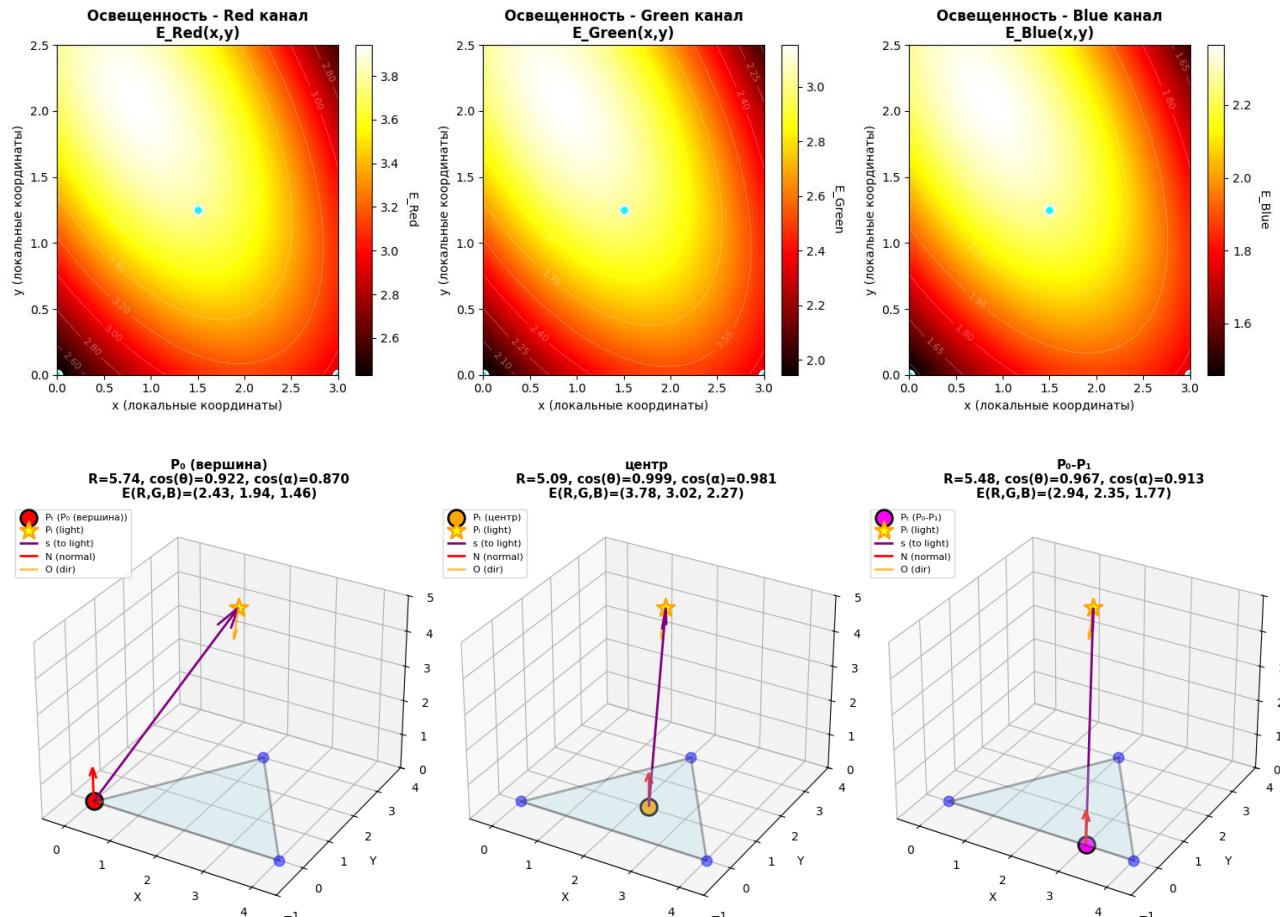
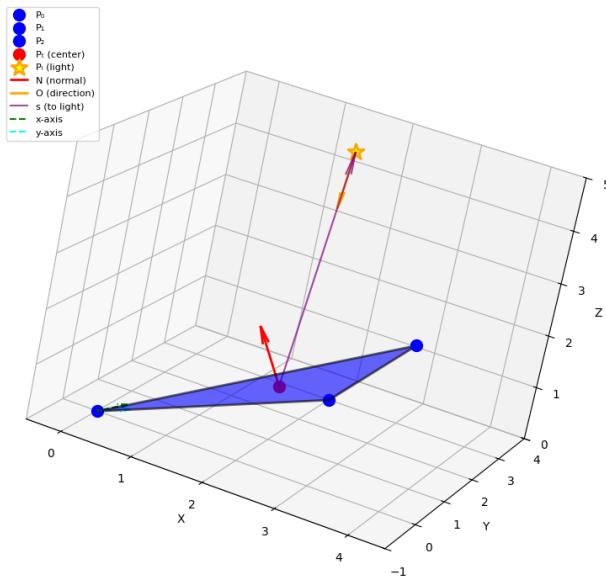
if __name__ == "__main__":
    main()

```

Пример входных данных и результаты

1. $I_0(RGB) = (100, 80, 60)$
2. $\vec{O} = (0, -0.19611, -0.98058)$
3. $\vec{P}_L = (2, 2, 5)$
4. $\vec{P}_0 = (0, 0, 0)$
5. $\vec{P}_1 = (4, 0, 0)$
6. $\vec{P}_2 = (2, 3, 0)$
7. $x_1 = 0.00, y_1 = 0.00$
 $x_2 = 0.75, y_2 = 0.62$
 $x_3 = 1.5, y_3 = 1.25$
 $x_4 = 2.25, y_4 = 1.88$
 $x_5 = 3.00, y_5 = 2.50$

Освещение треугольника точечным источником света



Вычисленные значения освещенности $E(RGB, \vec{P}_T)$ для точек, заданных локальными координатами:

x y	0.00	0.75	1.5	2.25	3.00
0.00	(2.43, 1.94, 1.47)	(2.83, 2.26, 1.70)	(3.09, 2.47, 1.85)	(3.13, 2.50, 1.88)	(2.94, 2.35, 1.76)
0.62	(2.89, 2.31, 1.73)	(3.31, 2.64, 1.98)	(3.50, 2.80, 2.10)	(3.42, 2.73, 2.05)	(3.08, 2.46, 1.85)
1.25	(3.33, 2.66, 2.00)	(3.69, 2.95, 2.21)	(3.77, 3.02, 2.26)	(3.54, 2.83, 2.12)	(3.07, 2.45, 1.84)
1.88	(3.65, 2.92, 2.19)	(3.91, 3.13, 2.34)	(3.84, 3.07, 2.30)	(3.45, 2.76, 2.07)	(2.89, 2.31, 1.73)
2.50	(3.79, 3.03, 2.27)	(3.90, 3.12, 2.34)	(3.67, 2.93, 2.20)	(3.18, 2.55, 1.91)	(2.59, 2.07, 1.55)

Вывод

В результате выполнения работы был изучен и реализован алгоритм вычисления освещенности точки на плоскости от источника света. Рассмотрены основные этапы вычислений, включая преобразование координат, определение нормали к плоскости, вычисление вектора до источника света и углов между векторами.