Министерство образования и науки Российской Федерации ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Отчёт по дисциплине
«Алгоритмические основы компьютерной графики»

# Лабораторная работа №2 «Алгоритм построения теней»

Студент: _____ Богдан А. В.
группы 5130201/20102
Преподаватель: _____ Курочкин М. А.

«_____»_____ 2025г.

Санкт-Петербург, 2025

# Содержание

## Введение

Современная компьютерная графика активно развивается, предлагая всё более реалистичные методы визуализации. Одним из ключевых аспектов, влияющих на восприятие сцены, является корректное отображение теней. Тени не только добавляют глубину и реализм в трёхмерные сцены, но и помогают пользователю лучше ориентироваться в пространстве, определяя взаимное расположение объектов и источников света.

# 1 Постановка задачи

В рамках данной лабораторной работе требуется реализовать алгоритм построения теней. Для этого необходимо выполнить следующее;

1. Изучить особенности алгоритмов построения теней.
2. Реализовать алгоритм построения теней.
3. Представить результаты реализации алгоритма построения теней.

# 2 Алгоритм построения теней

**Дано:** 3D-сцена:

- Параллелепипед P, заданный координатами вершин $\{\mathrm{v}_i \in R^3\}_{i=1}^8$.
- Плоскость $H$, заданная точкой $p_0 \in R^3$ и нормалью $n_H \in R^3$, $||n_H|| = 1$.
- Источник света L, находящийся на бесконечности положительной части оси Z.
- Наблюдатель $O$, заданный позицией $o \in R^3$ и ориентацией (широта $\phi$ и долгота $\theta$).

**Требуется:** Построить проекционную тень, отбрасываемую параллелепипедом на плоскость (рис. 2).

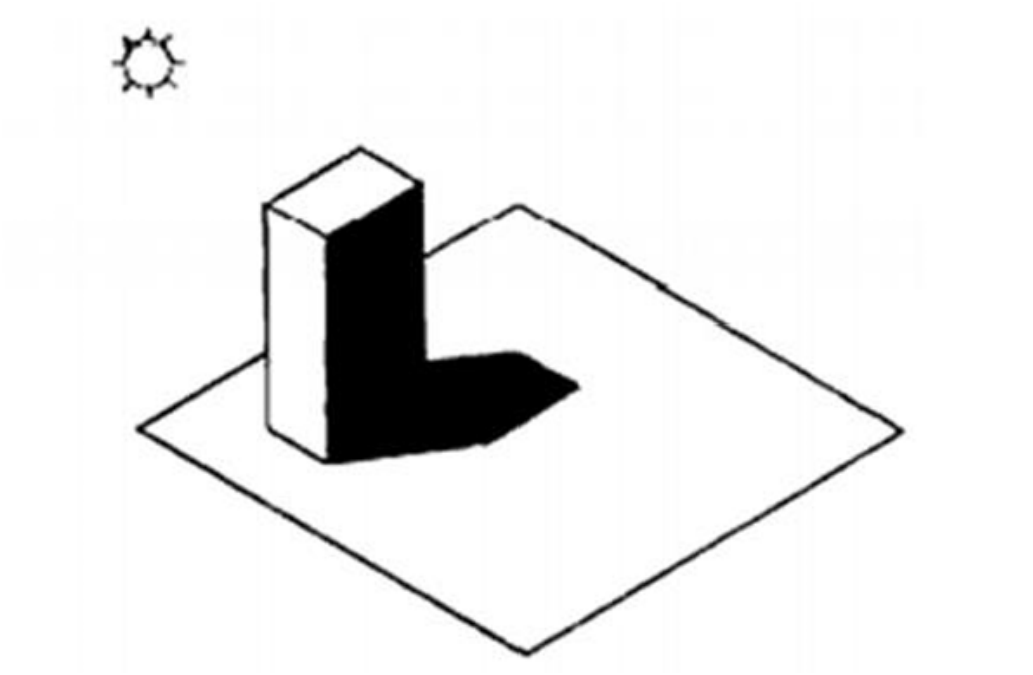

Рис. 1: Тень, отбрасываемая параллелепипедом на плоскость

## 2.1 Шаги алгоритма

1. **1. Определить нелицевые грани параллелепипеда:**
   - Для грани $F_j$, заданной вершинами $\{\mathrm{v}_a, \mathrm{v}_b, \mathrm{v}_c\}$, нормаль вычисляется следующим образом:
   $$n_j = \frac{(\mathrm{v}_b - \mathrm{v}_a) \times (\mathrm{v}_c - \mathrm{v}_a)}{||(\mathrm{v}_b - \mathrm{v}_a) \times (\mathrm{v}_c - \mathrm{v}_a)||}$$
   - Грань $Fj$ параллелепипеда $P$ считается нелицевой, если произведение её нормали $n_j$ (нормаль можно представить как вектор $(n_1(F_j), n_2(F_j), n_3(F_j))$) и вектора источника света $d_L$ положительна:
   $$n_j * d_L > 0$$
   - Но так как источник света расположен на бесконечности на положительной части оси Z, то грань будет считаться нелицевой в случае, если:
   $$n_3(F_j) < 0$$
2. Проекция нелицевых граней на плоскость H.

- Для каждой нелицевой грани $F_j$ выполняется параллельная проекция вершин $\{\mathrm{v}_k\}_{k=1}^4$ на Н вдоль $d_L$.
- Для вершины $\{\mathrm{v}_k\} = (x_K, y_k, z_k)$:

$$\mathrm{v}'_k = \mathrm{v}_k + t * d_L, t = \frac{n_H * (p_0 - \mathrm{v}_k)}{n_H * d_L}$$

- Для каждой нелицевой грани $F_j$, из проецированных вершин в порядке, соответствующем $F_j$ формируется теневой многоугольник, после чего он добавляется в структуру данных.

3. Построить вид сцены из заданной точки наблюдения.

- Применим матрицу сцены к каждой точке v (при этом координаты точек переводятся в однородные координаты посредством добавлением скалярного множителя $w = 1$), преобразующую мировые координаты в систему координат камеры:
  - Матрица трансляции:

$$T(o) = \begin{pmatrix} 1 & 0 & 0 & -o_x \\ 0 & 1 & 0 & -o_y \\ 0 & 0 & 1 & -o_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - Матрица вращения вокруг оси Y на угол $\theta$:

$$R_y(\theta) = \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - Матрица вращения вокруг оси X на угол $\phi$:

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - Итоговая матрица:

$$M_{view} = R_x(\phi) * R_y(\theta) * T(o)$$

- Далее применяется ортографическая проекция:

$$M_{ortho} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Последним шагом является визуализация сцены.

# 3 Результаты

Для реализации алгоритма трассировки лучей был использован Python 3.8 и библиотеки numpy, pyplot. На рис. 2-4 представлены результаты работы алгоритма построения теней для параллелепипеда и плоскости с трех разных ракурсов со следующими параметрами:

Вектор направления луча света: (0, 0, -1).

Координаты вершин параллелепипеда: [ [4.96, 2.2, 6.6], [6.7, 2.2, 5.6], [[7.2, 3.92, 6.46], [5.46, 3.92, 7.46], [5.83, 1.2, 8.1], [7.56, 1.2, 7.1], [8.06, 2.93, 7.96], [6.33, 2.93, 8.96] ]

Точка плоскости:[0, 0, 0]

Нормаль плоскости:[0, 0, 1]

На рисунке 2 представлена визуализация сцены с широтой и долготой наблюдателя аналогичной источнику света. Источник света расположен позади наблюдателя.

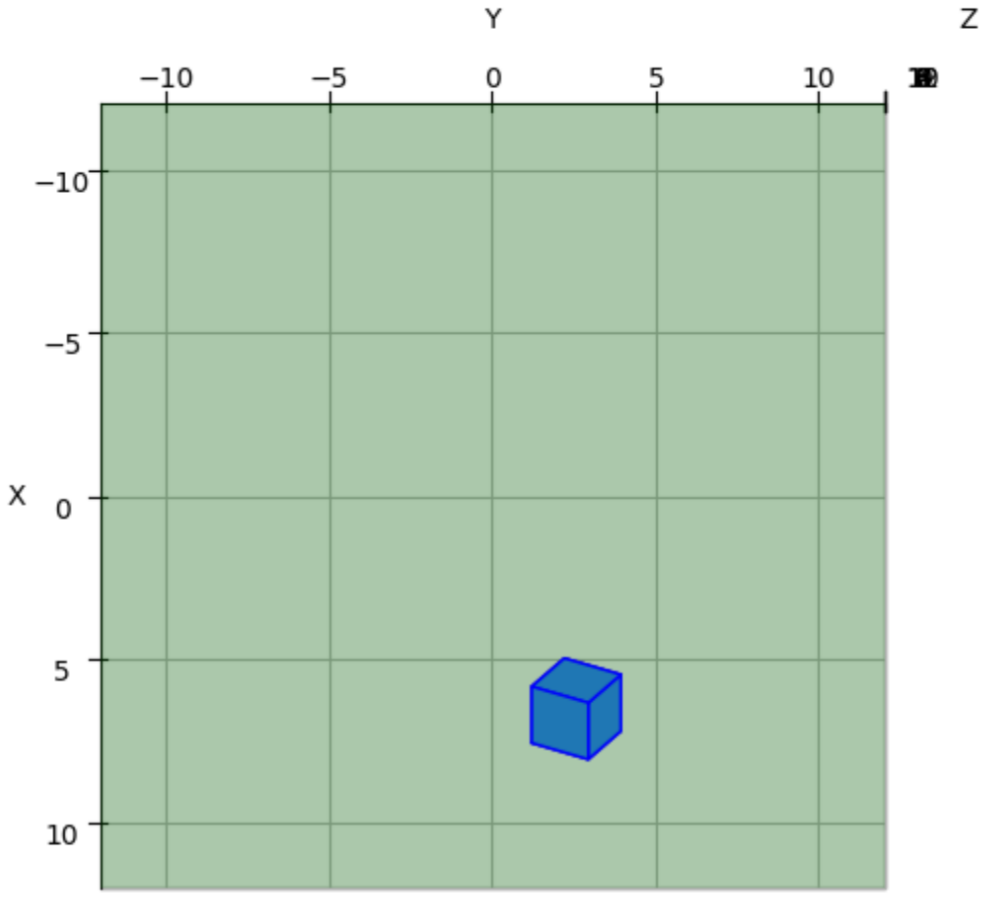


Рис. 2: Положение наблюдателя (0, 0, 10), долгота $0^o$, широта $90^o$

Как можно заметить, тени не видно, так как она полностью закрыта объектом.
На рисунке 3 представлена визуализация сцены после перемещения наблюдателя в позицию (-5, 0, 8.66) изменения долготы наблюдателя до 180, широты до 60.

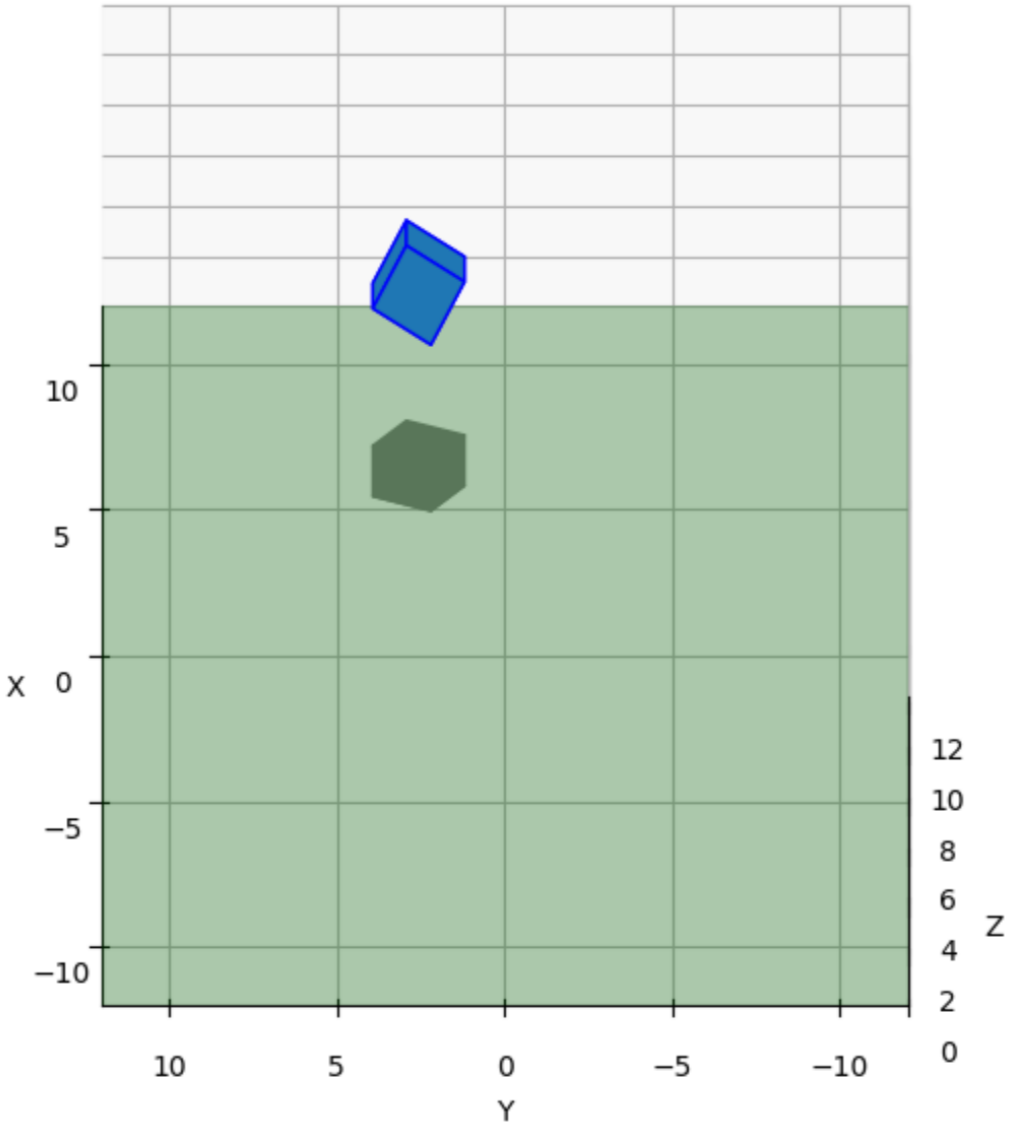


Рис. 3: Положение наблюдателя (0, 0, 10), долгота $180^o$, широта $60^o$

На рисунке 4 представлена визуализация сцены после изменения долготы наблюдателя до 0 и широты до -90.

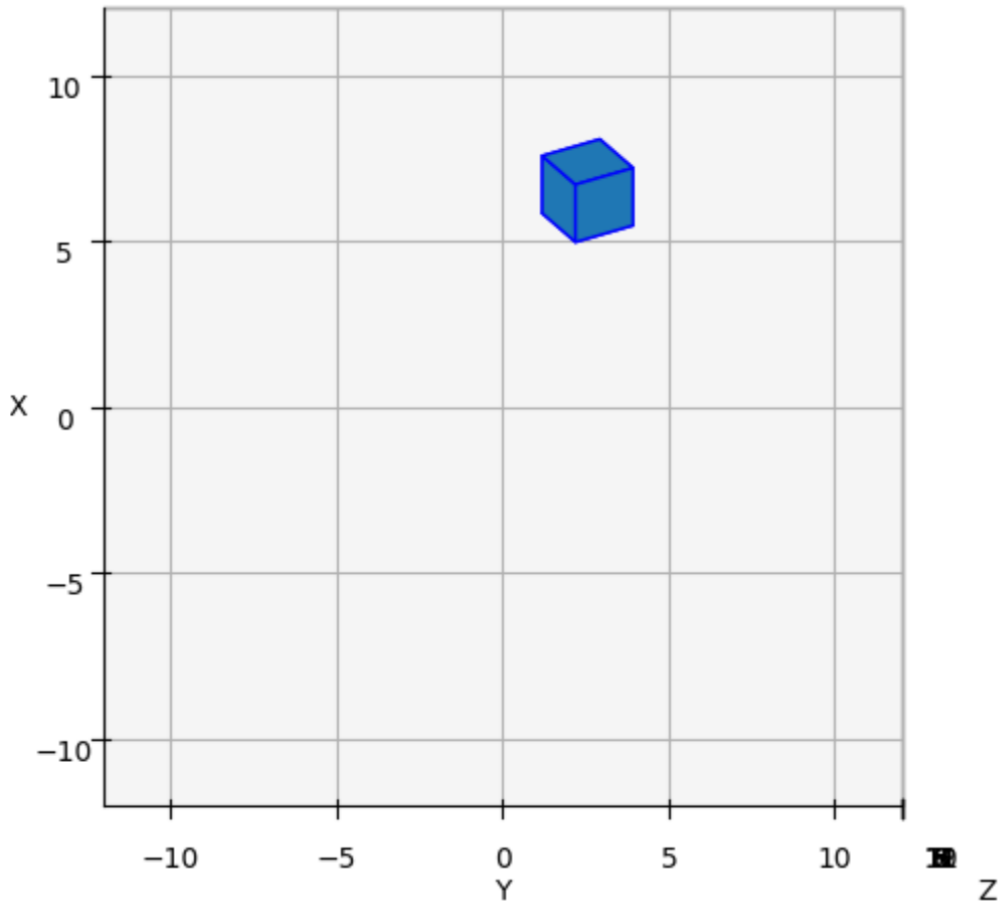


Рис. 4: Положение наблюдателя (0, 0, 5), долгота $0^o$, широта $-90^o$

В этом положении наблюдатель смотрит прямо на источник света, тень расположена позади наблюдателя.

# 4 Заключение

В данной работе были изучены особенности алгоритмов построения теней. В частности, были изучены особенности алгоритма трассировки лучей, а также сделана его реализация на языке Python в среде разработки Visual Studio Code. В качестве результатов алгоритма были представлены 3 изображения сцены с различными параметрами положения наблюдателя, а также направления его взгляда.

# Список литературы

[1] Препарата Ф., Шеймос М., "Вычислительная геометрия: введение"

# Приложение 1. Код программы

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

def rotate_points(points, angle_x, angle_y, angle_z):
    # Матрицы поворота
    rx = np.array([
        [1, 0, 0],
        [0, math.cos(angle_x), -math.sin(angle_x)],
        [0, math.sin(angle_x), math.cos(angle_x)]
    ])

    ry = np.array([
        [math.cos(angle_y), 0, math.sin(angle_y)],
        [0, 1, 0],
        [-math.sin(angle_y), 0, math.cos(angle_y)]
    ])

    rz = np.array([
        [math.cos(angle_z), -math.sin(angle_z), 0],
        [math.sin(angle_z), math.cos(angle_z), 0],
        [0, 0, 1]
    ])

    # Комбинированный поворот
    rotation_matrix = rz @ ry @ rx

    # Применение поворота ко всем точкам
    return [rotation_matrix @ point for point in points]


class ShadowProjection:
    def __init__(self, box_vertices, plane_point, plane_normal):
        self.box_vertices = np.array(box_vertices)
        self.plane_point = np.array(plane_point)
        self.plane_normal = np.array(plane_normal)

        # box_vertices = [
        #     [1, 1, 1], [3, 1, 1], [3, 3, 1], [1, 3, 1],
        #     [1, 1, 3], [3, 1, 3], [3, 3, 3], [1, 3, 3]
        # ]

        # Определение граней параллелепипеда (индексы вершин)
        self.faces = [
            [3, 2, 1, 0],    # нижняя грань
            [4, 5, 6, 7],    # верхняя грань
            [0, 3, 7, 4],    # левая грань
            [1, 2, 6, 5],    # правая грань
            [0, 1, 5, 4],    # передняя грань
            [2, 3, 7, 6]     # задняя грань
        ]

    def get_light_direction(self, latitude, longitude):
        # Преобразование широты/долготы в вектор направления
        lat = np.radians(latitude)
        lon = np.radians(longitude)
```

```
59          return np.array([
60              np.cos(lat) * np.cos(lon),
61              np.cos(lat) * np.sin(lon),
62              np.sin(lat)
63          ])
64
65      def calculate_face_normal(self, face):
66          # Вычисление нормали грани через векторное произведение
67          v1 = self.box_vertices[face[1]] - self.box_vertices[face[0]]
68          v2 = self.box_vertices[face[2]] - self.box_vertices[face[0]]
69          normal = np.cross(v1, v2)
70          return normal / np.linalg.norm(normal)
71
72      def project_shadow(self, light_dir):
73          # 1. Найти нелицевые грани
74          back_faces = []
75          for i, face in enumerate(self.faces):
76              normal = self.calculate_face_normal(face)
77              if np.dot(normal, light_dir) <= 0:
78                  back_faces.append(face)
79
80          # 2. Проекция нелицевых граней на плоскость
81          shadow_polygons = []
82          for face in back_faces:
83              projected = []
84              for v_idx in face:
85                  vertex = self.box_vertices[v_idx]
86                  # Параллельная проекция на плоскость
87                  t = np.dot(self.plane_normal, self.plane_point - vertex)
                        / np.dot(self.plane_normal, light_dir)
88                  shadow_point = vertex + t * light_dir
89                  projected.append(shadow_point)
90              shadow_polygons.append(projected)
91
92          return shadow_polygons
93
94      def visualize(self, light_dir, observer_pos):
95          fig = plt.figure(figsize=(10, 8))
96          ax = fig.add_subplot(111, projection='3d')
97
98          # Создаем поверхность (плоскость)
99          x = np.linspace(-12, 12, 20)
100         y = np.linspace(-12, 12, 20)
101         X, Y = np.meshgrid(x, y)
102
103         # Уравнение плоскости: n  (r - r0) = 0 => n_x(x-x0) + n_y(y-y0) +
              n_z(z-z0) = 0
104         # Решаем относительно Z: z = (n_x(x0-x) + n_y(y0-y))/n_z + z0
105         # Решаем относительно Y: y = (n_x(x0-x) + n_z(z0-z))/n_y + y0
106         Z = (self.plane_normal[0]*(self.plane_point[0]-X) +
107              self.plane_normal[1]*(self.plane_point[1]-Y)) / self.plane_
                 normal[2] + self.plane_point[2]
108
109         # # Отрисовка плоскости
110         ax.plot_surface(X, Y, Z, alpha=0.3, color='green')
111
112         # Отрисовка параллелепипеда
113         ax.add_collection3d(Poly3DCollection(
114             [self.box_vertices[face] for face in self.faces],
115             alpha=1, linewidths=1, edgecolor='b'
```

```
        ))

        # # Отрисовка теней
        shadows = self.project_shadow(light_dir)
        ax.add_collection3d(Poly3DCollection(
            shadows, alpha=1, color='gray'
        ))

        # Настройка камеры
        ax.view_init(elev=observer_pos[0], azim=observer_pos[1])
        ax.set_proj_type('ortho')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')
        ax.set_xlim3d(-12, 12)
        ax.set_ylim3d(-12, 12)
        ax.set_zlim3d(0, 12)
        plt.show()

# Пример использования
# box_vertices = [
#     [1, 6, 6], [3, 6, 6], [3, 8, 6], [1, 8, 6],
#     [1, 6, 8], [3, 6, 8], [3, 8, 8], [1, 8, 8]
# ]

# # Пример использования
# box_vertices = [
#     [1, 1, 1], [3, 1, 1.5], [3.5, 3, 1.5], [1.5, 3, 1],
#     [1, 1.5, 3], [3, 1, 3.5], [3.5, 3, 4], [1.5, 3.5, 3]
# ]

base_box = [
    [1, 6, 6], [3, 6, 6], [3, 8, 6], [1, 8, 6],
    [1, 6, 8], [3, 6, 8], [3, 8, 8], [1, 8, 8]
]

box_vertices = rotate_points(base_box,
                             math.radians(30),
                             math.radians(30),
                             math.radians(0)
                            )

sp = ShadowProjection(
    box_vertices=box_vertices,
    plane_point=[0, 0, 0],
    plane_normal=[0, 0, 1]  # Плоскость Y=0
)

light_dir = sp.get_light_direction(90, 0)

sp.visualize(light_dir, observer_pos=(90, 0))
```