МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Отчёт по дисциплине «Алгоритмические основы компьютерной графики»

Лабораторная работа №2
«Алгоритм отсечения отрезка выпуклым телом. Алгоритм Кируса-Бека»
Вариант **№14**

Студент: _____ Салимли Айзек Мухтар Оглы

Преподаватель: _____ Курочкин Михаил Александрович

«____»_____ 20___ г.

Санкт-Петербург, 2025

Содержание

Введение

Алгоритм Кируса–Бека (Cyrus–Beck) — это один из классических алгоритмов отсечения отрезков, применяемых в компьютерной графике. Он основан на параметрическом представлении отрезка и векторной геометрии, что делает его особенно подходящим для работы с выпуклыми многоугольниками и многогранниками.

Данный алгоритм применяется в следующих областях:

- **Компьютерная графика:**
  - Отсечение лишних отрезков и линий за пределами видимой сцены (clipping);
  - Подготовка сцен к рендерингу, особенно в 2D и 3D-графике.
- **Геометрические алгоритмы:**
  - Проверка принадлежности точки/отрезка выпуклому многоугольнику;
  - Расчёт пересечений отрезков с границами объектов.
- **Системы автоматизированного проектирования (CAD):**
  - Удаление невидимых линий;
  - Обработка контуров и сложных геометрий.

На данный момент алгоритм Кируса–Бека используется в следующих случаях:

- При необходимости высокой точности и контроля над отсечением, особенно в научных и инженерных приложениях;
- Когда нужно работать с выпуклыми полигонами и важно строго учитывать направление нормалей;
- При разработке собственных графических движков или low-level алгоритмов, где важно понимать и контролировать всю геометрию вручную.

В данном отчете проведена реализация и сравнение алгоритма отсечения отрезка выпуклым телом (Кируса-Бека) с библиотечным алгоритмом Shapely.

Также в отчете теоретически сравнивается алгоритм Кируса-Бека с другими алгоритмами отсечения отрезка выпуклым телом, рассмотрены такие алгоритмы как:

- Алгоритм Коэна-Сазерленда
- Метод пересечения с полуплоскостями

1 Постановка задачи алгоритма Кируса-Бека

- Дано: **p0**, **p1** — отрезок, polygon = $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ — выпуклый многоугольник.
- Надо: Найти **q0**, **q1** — точки пересечения отрезка и многоугольника.
- Ограничения: polygon — выпуклый, отрезок **p0p1** пересекает polygon.

2 Математическое описание алгоритма Кируса-Бека

Пусть требуется отсечь **отрезок** с начальными и конечными точками **p0** и **p1** относительно выпуклого многоугольника polygon. Введём **параметрическое** уравнение для любой точки $\mathbf{p}(t)$ на прямой, содержащей этот отрезок:

$$\mathbf{p}(t) = \mathbf{p0} + t(\mathbf{p1} - \mathbf{p0}), \quad t \in [0, 1].$$

Здесь $t = 0$ соответствует точке **p0**, а $t = 1$ — точке **p1**. Если после выполнения алгоритма окажется, что допустимый диапазон t сузился до $[\, t_{\min}, t_{\max}\,] \subseteq [0, 1]$, то отсечённая часть отрезка будет задаваться:

$$\mathbf{q0} = \mathbf{p}(t_{\min}), \quad \mathbf{q1} = \mathbf{p}(t_{\max}).$$

Обозначим отсекающую область polygon — произвольный выпуклый многоугольник. Внутренняя нормаль **normal** в произвольной точке **cur**, лежащей на границе polygon, удовлетворяет условию

$$\mathbf{normal} \cdot (\mathbf{b} - \mathbf{cur}) \geq 0,$$

где **b** — любая другая точка на границе polygon. Это условие иллюстрирует рисунок 1, где обозначены внутренняя $\mathbf{normal}_B$ и внешняя нормаль $\mathbf{normal}_H$; указанное условие соответствует

$$\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right],$$

где угол θ — угол между внутренней нормалью $\mathbf{normal}_B$ и вектором $(\mathbf{b} - \mathbf{cur})$.

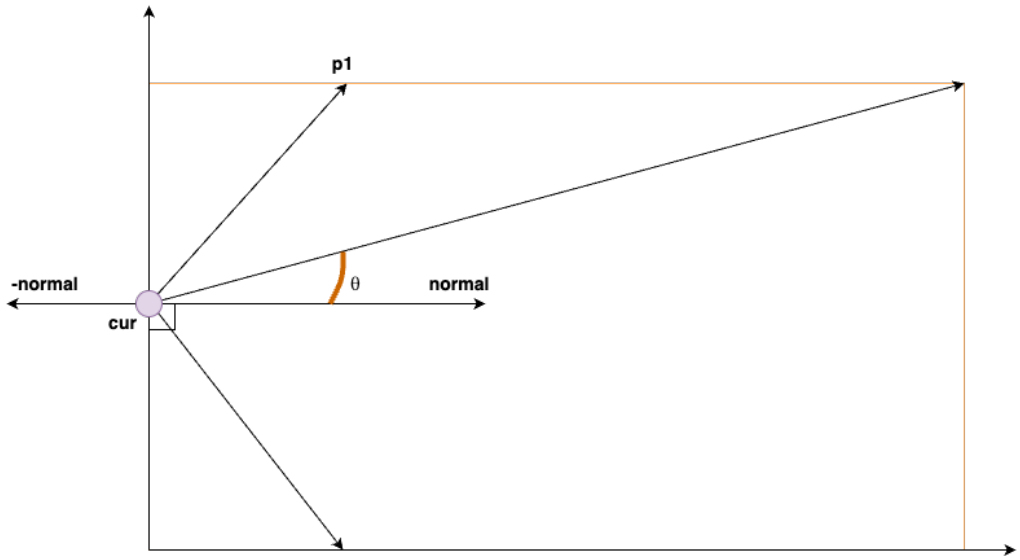

Рис. 1: Внутренняя и внешняя нормали

Если **cur** — граничная точка выпуклой области polygon, а **normal** — внутренняя нормаль к одной из ограничивающих эту область сторон, то для любой конкретной величины t, т. е. для любой точки отрезка **p0**, **p1**:

- из условия $\mathbf{normal} \cdot (\mathbf{p}(t) - \mathbf{cur}) < 0$ следует, что вектор $\mathbf{p}(t) - \mathbf{cur}$ направлен вовне области polygon;
- из условия $\mathbf{normal} \cdot (\mathbf{p}(t) - \mathbf{cur}) = 0$ следует, что $\mathbf{p}(t) - \mathbf{cur}$ лежит в плоскости, проходящей через **cur** и перпендикулярной нормали;
- из условия $\mathbf{normal} \cdot (\mathbf{p}(t) - \mathbf{cur}) > 0$ следует, что вектор $\mathbf{p}(t) - \mathbf{cur}$ направлен внутрь polygon, как показано на рис. 2.

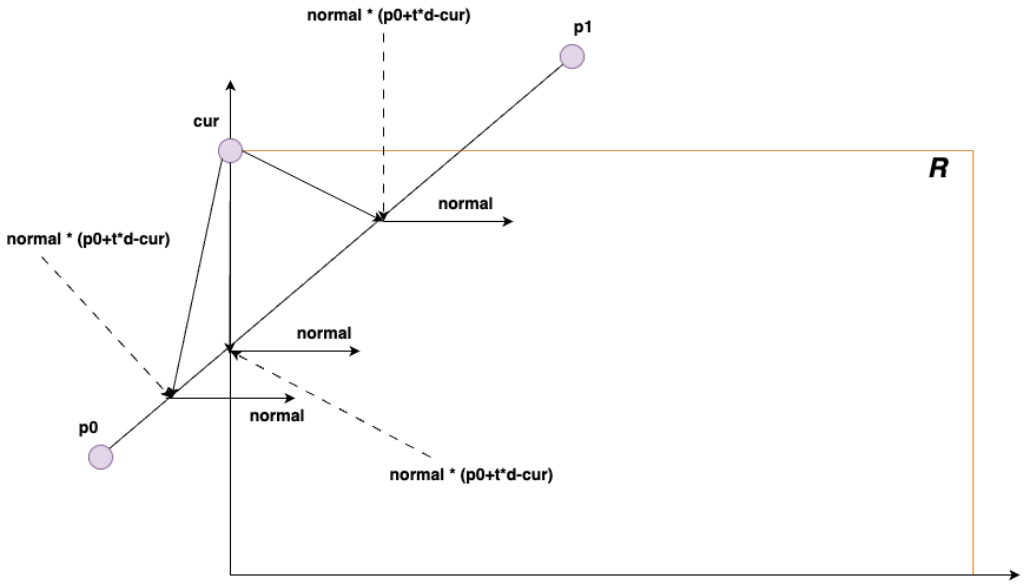

Рис. 2: Иллюстрация направлений нормалей

Из всех этих условий, взятых вместе, следует, что бесконечная прямая пересекает замкнутую выпуклую область ровно в двух точках. Далее, пусть эти две точки не принадлежат одной граничной стороне. Тогда уравнение

$$\mathbf{normal} \cdot (\mathbf{p}(t) - \mathbf{cur}) = 0$$

имеет только одно решение. Если точка **cur** лежит на той граничной стороне, для которой **normal** является внутренней нормалью, то точка на отрезке **p**, которая удовлетворяет последнему уравнению, будет точкой пересечения этого отрезка с указанной граничной стороной.

2.1 Расчёт параметра пересечения t

Подставим определение для $\mathbf{p}(t)$ в уравнение $\mathbf{normal} \cdot (\mathbf{p}(t) - \mathbf{cur}) = 0$, получим:

$$\mathbf{normal} \cdot (\mathbf{p0} + (\mathbf{p1} - \mathbf{p0})t - \mathbf{cur}) = 0 \iff \underbrace{\mathbf{normal} \cdot (\mathbf{p0} - \mathbf{cur})}_{=\text{numerator}} + \underbrace{\mathbf{normal} \cdot (\mathbf{p1} - \mathbf{p0})}_{=\text{denominator} \neq 0} t = 0$$

— условие пересечения отрезка с i-й границей области. Решая относительно t, получим:

$$t = -\frac{\text{numerator}}{\text{denominator}}, \quad i = 1, 2, \ldots, n$$

Здесь $\text{denominator} = 0$ может быть только в случае, если $\mathbf{p1} = \mathbf{p0}$, либо если отрезок параллелен грани многоугольника.

Классификация положения точки:

$$\text{numerator} \begin{cases} < 0, & \text{точка вне многоугольника;} \\ = 0, & \text{точка на границе многоугольника;} \\ > 0, & \text{точка внутри многоугольника.} \end{cases}$$

Если значение t лежит за пределами интервала $t \in [0; 1]$, то его можно проигнорировать. Хотя известно, что отрезок может пересечь выпуклый многоугольник не более чем в двух точках, уравнения могут дать большее число решений в интервале $t \in [0; 1]$. Эти решения следует разбить на две группы: нижнюю и верхнюю, в зависимости от того, к началу или к концу отрезка будет

ближе соответствующая точка. Нужно найти наибольшую из нижних и наименьшую из верхних точек.

Если denominator > 0, то найденное значение t рассматривается как возможный нижний предел. Если denominator < 0, то значение t рассматривается как возможный верхний предел.

3 Программная реализация алгоритма Кируса-Бека

Алгоритм был написан на языке Python 3.13.1 в интегрированной среде разработки VSCode. Для реализации графического отображения работы алгоритма использовалась библиотека matplotlib.

Инициализация параметрического представления и интервала t

```
d = p1 - p0
t_min, t_max = 0.0, 1.0
```

Пояснение: Отрезок задаётся через точку **p0** и вектор направления $\mathbf{d} = \mathbf{p1} - \mathbf{p0}$. Интервал параметра $t \in [0, 1]$ соответствует всему отрезку, где $t = 0$ даёт **p0**, а $t = 1$ даёт **p1**.

Цикл по сторонам выпуклого многоугольника

```
for i in range(n):
    cur = polygon[i]
    nxt = polygon[(i + 1) % n]
    edge_vec = nxt - cur
    normal = np.array([-edge_vec[1], edge_vec[0]], dtype=float)
```

Пояснение: Проходим по каждой стороне многоугольника. Для каждой стороны вычисляется вектор **edge_vec** и затем внутренняя нормаль **normal** $= (-\Delta y, \Delta x)$, направленная внутрь многоугольника (при условии, что вершины заданы в порядке обхода CCW).

Вычисление числителя и знаменателя для текущей стороны

```
numerator = np.dot(normal, (p0 - cur))
denominator = np.dot(normal, d)
```

Пояснение: Здесь вычисляются:

- numerator $= \mathbf{normal} \cdot (\mathbf{p0} - \mathbf{cur})$ — определяет положение точки **p0** относительно текущей границы.
- denominator $= \mathbf{normal} \cdot \mathbf{d}$ — показывает, как направлен отрезок относительно нормали.

Обработка параллельного случая

```
if abs(denominator) < 1e-12:
    if numerator < 0:
        return (False, None, None)
    else:
        continue
```

Пояснение: Если denominator почти равен нулю, отрезок параллелен текущей стороне. Если **p0** находится вне (то есть numerator < 0), то отрезок не может пересечь многоугольник и алгоритм возвращает "не пересекается". Если **p0** внутри, то эта сторона не изменяет интервал t и переходим к следующей.

Вычисление параметра t и обновление интервала $[t_{\min}, t_{\max}]$

```
t = - numerator / denominator
if denominator > 0:
    if t > t_max:
        return (False, None, None)
    if t > t_min:
        t_min = t
else:
    if t < t_min:
        return (False, None, None)
```

```
    if t < t_max:
        t_max = t

if t_min > t_max:
    return (False, None, None)
```

Здесь вычисляется значение $t = -\frac{\text{numerator}}{\text{denominator}}$, которое соответствует точке пересечения отрезка с данной гранью. В зависимости от знака denominator:

- Если denominator > 0 (входящая грань), обновляем $t_{\min}$ (если $t > t_{\min}$).
- Если denominator < 0 (выходящая грань), обновляем $t_{\max}$ (если $t < t_{\max}$).

Если после обновления $t_{\min} > t_{\max}$, значит пересечение отсутствует.

Вычисление итоговых точек отсечения

```
q0 = p0 + t_min * d
q1 = p0 + t_max * d
return (True, q0, q1)
```

Пояснение: Если после обработки всех сторон интервал $[t_{\min}, t_{\max}]$ остаётся допустимым, вычисляются итоговые точки пересечения:

$$\mathbf{q0} = \mathbf{p0} + t_{\min} \cdot \mathbf{d}, \quad \mathbf{q1} = \mathbf{p0} + t_{\max} \cdot \mathbf{d}.$$

Эти точки и задают фрагмент отрезка, лежащий внутри многоугольника.

3.1 Блок-схема

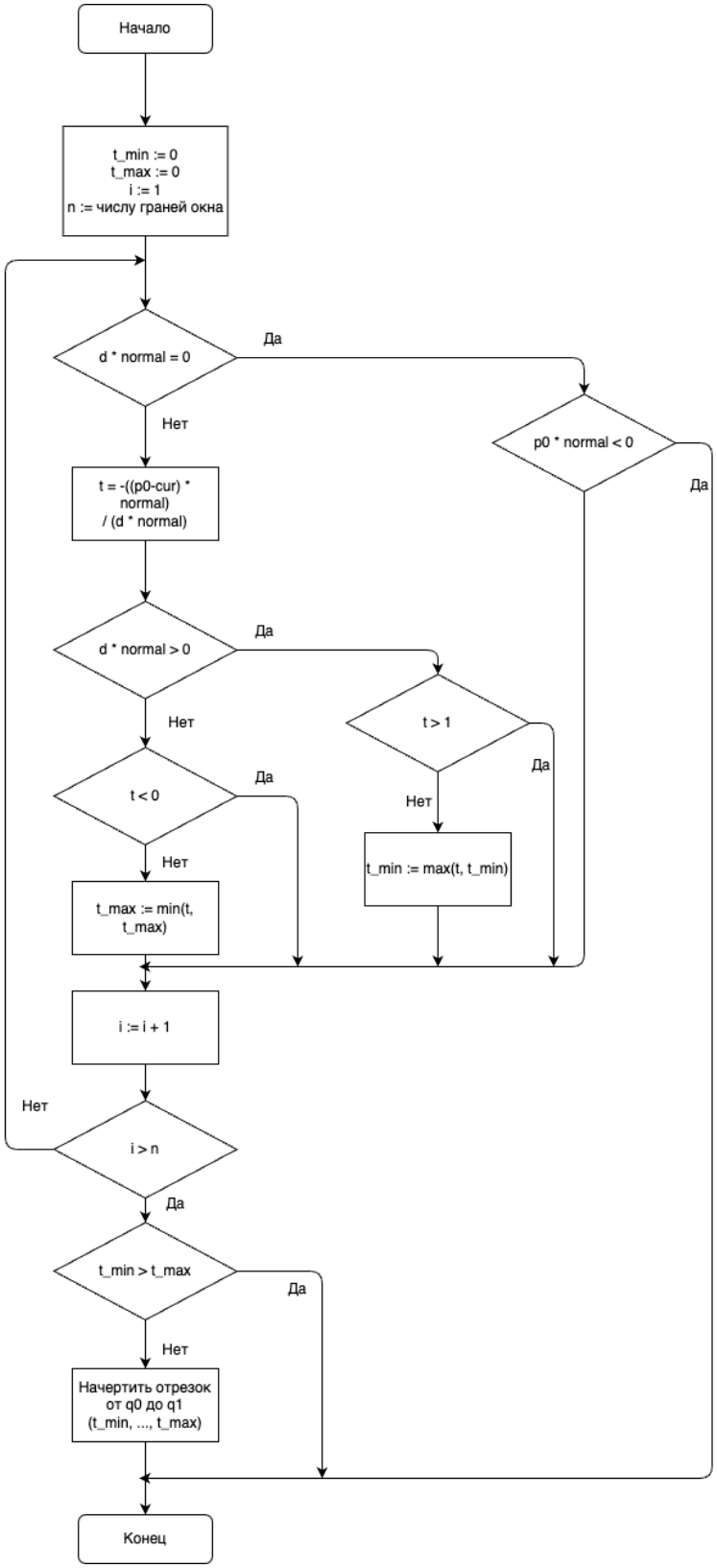

Рис. 3: Блок-схема алгоритма Кируса-Бека.

3.2 Результаты программной реализации

На рисунке 1 показан изначальный отрезок и выпуклый многоугольник, на рисунке 2 показано отсечение отрезка.

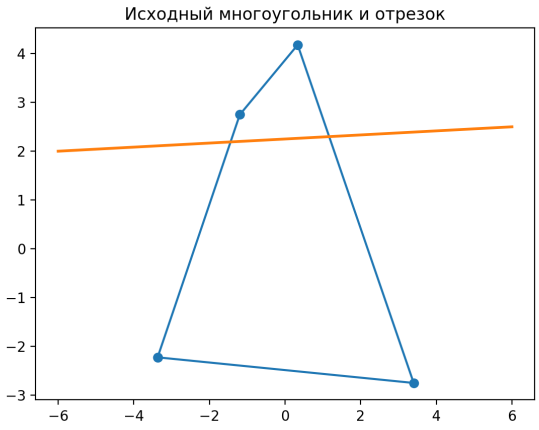

Рис. 4: Алгоритм Кируса-Бека. Начальная стадия.

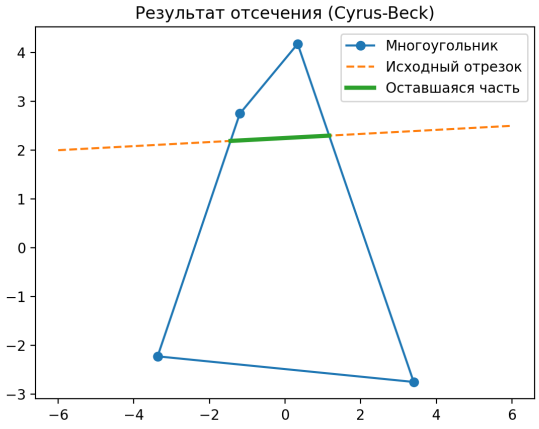

Рис. 5: Алгоритм Кируса-Бека. Полученный отрезок.

4 Другие алгоритмы отсечения отрезка выпуклым телом

4.1 Алгоритм Коэна-Сазерленда

Алгоритм Коэна-Сазерленда изначально был разработан для отсечения линий прямоугольным окном (axis-aligned rectangle). Он активно использовался в компьютерной графике (например, при рендеринге 2D-примитивов). Основная идея состоит в следующем:

1. Прямоугольник задаётся координатами $\text{xmin}, \text{ymin}, \text{xmax}, \text{ymax}$.
2. Каждой точке плоскости приписывается 4-битный «код региона» (Outcode), указывающий, где точка лежит относительно прямоугольника — сверху, снизу, слева, справа. Биты (слева-справа, снизу-сверху и т. д.). Например, используется нотация (TOP, BOTTOM, RIGHT, LEFT).
3. Для двух концов отрезка вычисляются эти кодовые слова.
4. **Логика отсечения:**
   - Если $(\text{код1} \,\&\, \text{код2}) \neq 0$, то это означает, что оба конца лежат в одной и той же «внешней» области относительно одной из сторон $\Rightarrow$ отрезок вне, его можно сразу отбросить.
   - Если $(\text{код1} \,\|\, \text{код2}) = 0$, то это значит, что оба конца внутри прямоугольника $\Rightarrow$ отрезок целиком внутри.
   - Иначе производится «усечение» одного конца, пересчитывается код, повторяются проверки:
     - Берётся один из концов, который находится снаружи (по коду).
     - Находится пересечение с соответствующей границей прямоугольника.
     - Заменяется этот конец на точку пересечения, снова вычисляется код.
     - Процесс повторяется, пока не станет ясно, что отрезок целиком отброшен или полностью лежит внутри.

Алгоритм легко реализуется, эффективно работает именно для прямоугольной области. Для произвольного выпуклого многоугольника он не применяется напрямую (без существенных модификаций).

Пример:

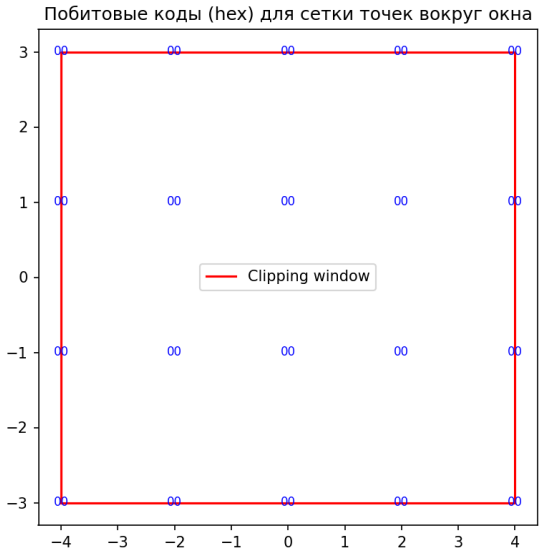

Рис. 6: Алгоритм Коэна-Сазерленда. «Карта» побитовых кодов (Outcode) в окрестности окна отсечения.

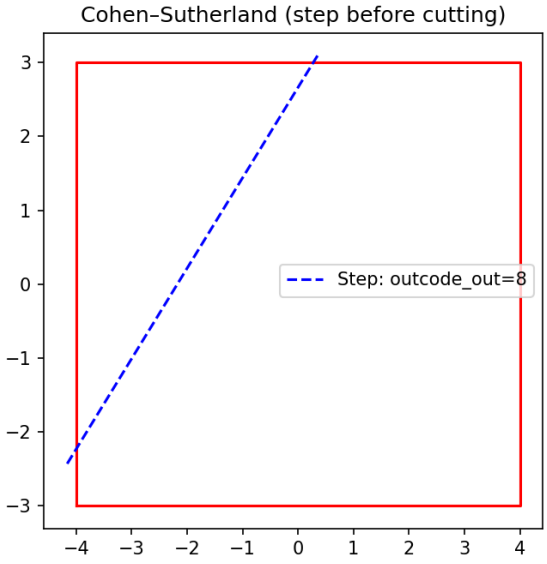

Рис. 7: Алгоритм Коэна-Сазерленда. Промежуточное состояние до очередного «обрезания».

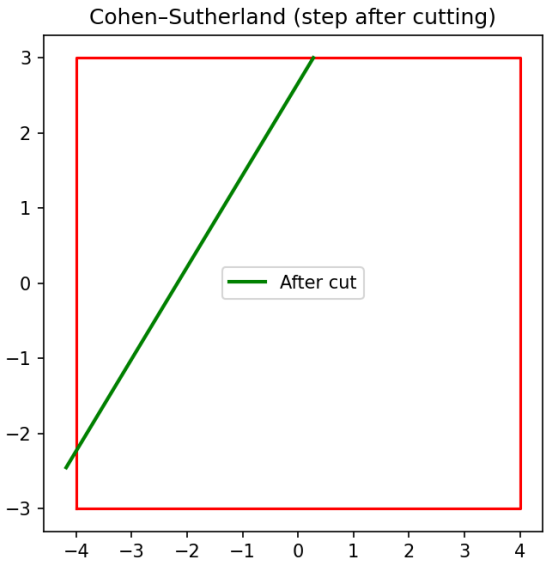

Рис. 8: Алгоритм Коэна-Сазерленда. То же самое состояние алгоритма, но после «обрезки» одного конца.

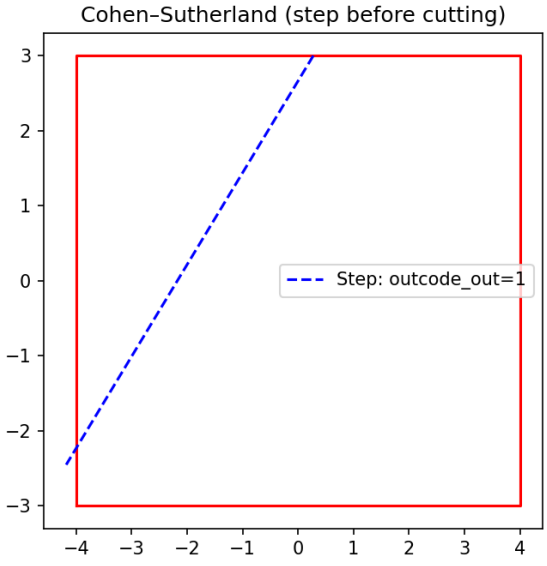

Рис. 9: Алгоритм Коэна-Сазерленда. Промежуточное состояние до очередного «обрезания».

Отсечение верхнего отрезка.

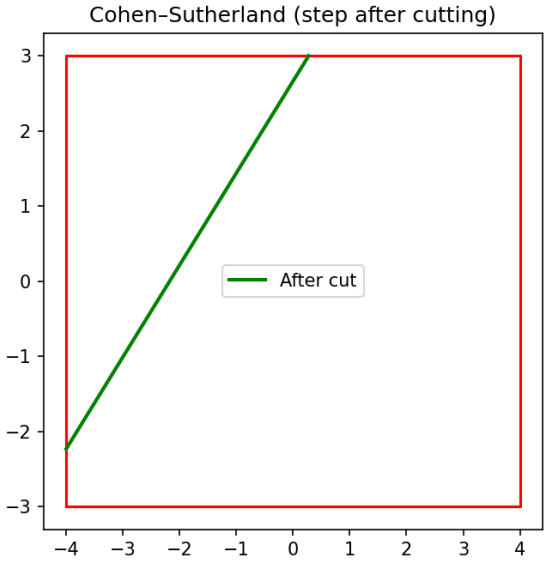

Рис. 10: Алгоритм Коэна-Сазерленда. Промежуточное состояние после «обрезания».

Конечный результат отсечения двух отрезков вне окна.

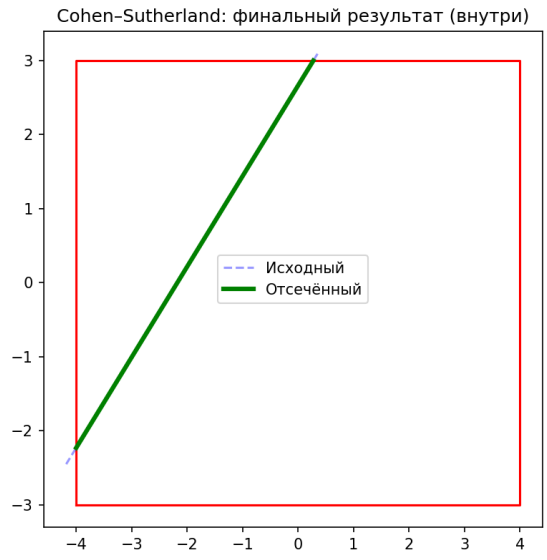

Рис. 11: Алгоритм Коэна-Сазерленда. Результат.

4.2 Метод пересечения с полуплоскостями

Любое выпуклое тело в 2D (выпуклый многоугольник) можно рассматривать как пересечение набора полуплоскостей (каждая сторона многоугольника задаёт одну полуплоскость). Соответственно, чтобы отсечь отрезок, мы можем:

1. Иметь список $\{H_1, H_2, \ldots, H_n\}$ полуплоскостей, каждая определяется линейным неравенством $a_i\,x + b_i\,y + c_i \geq 0$.
2. Начинать с «полного» отрезка [**p0**, **p1**].

3. Для каждой полуплоскости H_i **«обрезать»** текущую часть отрезка так, чтобы она оставалась только внутри H_i. Это эквивалентно вычислению пересечения отрезка с H_i.

4. Если на каком-то шаге кусок отрезка становится пустым, алгоритм завершает работу (всё снаружи).

5. В конце, если осталось что-то от отрезка, то это и есть итоговое пересечение с выпуклым многоугольником.

По сути, Кирус-Бек — это более структурированная версия «пересечения с полуплоскостями» с использованием параметра t. Но можно реализовать этот метод и «поэтапно», последовательно «отрезая» часть отрезка полуплоскостью.

Пример: Исходный отрезок до отсечения:

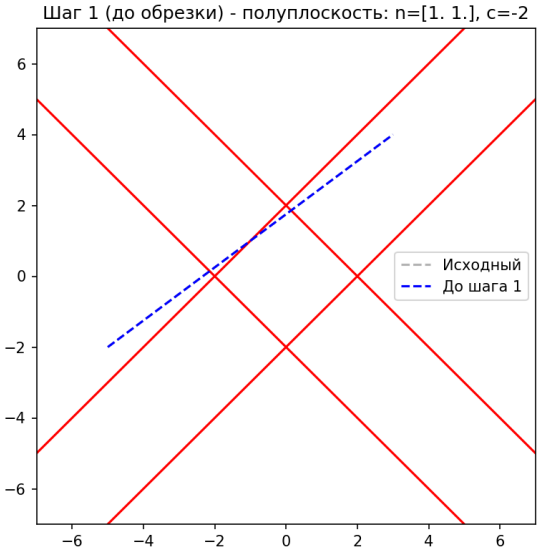

Рис. 12: Метод пересечения с полуплоскостями. До обрезки.

Отрезок до обрезки следующим шагом:

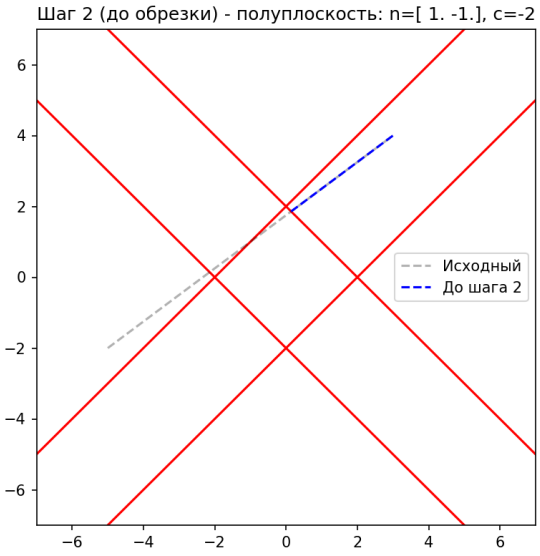

Рис. 13: Метод пересечения с полуплоскостями. До обрезки.

Результат обрезки отрезка:

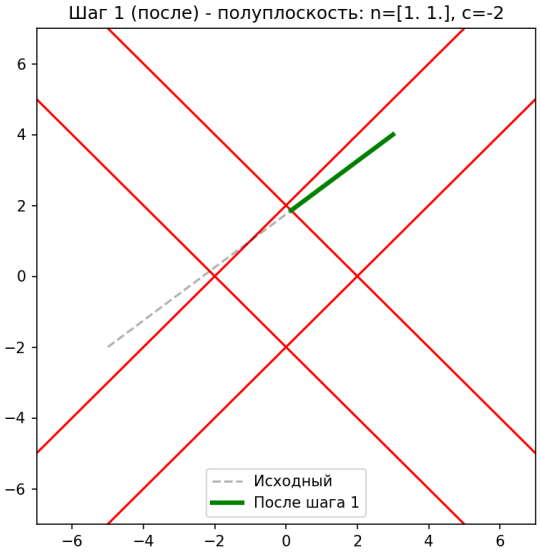

Рис. 14: Метод пересечения с полуплоскостями. После обрезки.

5 Сравнение алгоритмов

Коэн–Сазерленд отлично подходит для отсечения отрезков прямоугольным окном благодаря использованию битовых кодов, что обеспечивает очень быструю обработку (практически $O(1)$ для фиксированного окна). Однако алгоритм не универсален и требует значительных модификаций для произвольных выпуклых областей.

Кируса–Бека является универсальным алгоритмом, применимым для любого выпуклого многоугольника (и даже для 3D), но его сложность составляет $O(n)$, что может стать узким местом при большом числе сторон.

Метод пересечения с полуплоскостями по сути эквивалентен Кируса–Бека в плане функциональности и сложности ($O(n)$). Он концептуально проще, так как сводится к последовательному «подрезанию» отрезка по каждой полуплоскости, но на практике его реализация будет схожа по сложности и особенностям с алгоритмом Кируса–Бека.

Таким образом, выбор алгоритма зависит от конкретной задачи: для простых прямоугольных окон эффективен Коэн–Сазерленд, а для произвольных выпуклых областей лучше использовать Кируса–Бека или метод пересечения с полуплоскостями.

6 Сравнение собственной реализации с библиотечной

В библиотечной версии языка Python алгоритмом, аналогичным Кируса-Бека, является алгоритм Shapely. Было проведено 500 тестов. Где Cyrus-Beck — реализация собственного

Таблица 1: Сравнение производительности алгоритма Cyrus–Beck и библиотеки Shapely

Сценарий	Тестов	Вершин	Радиус	Cyrus–Beck (мкс)	Shapely (мкс)
1	500	10	10	15.48	41.70
2	500	30	10	28.63	40.30
3	500	50	10	46.48	88.04
4	500	100	15	30.62	35.06

алгоритма. Как было сказано во введении, наша реализация уступает библиотечной при малых n и незначительно уступает при больших n. Ниже представлена гистограмма сравнения.

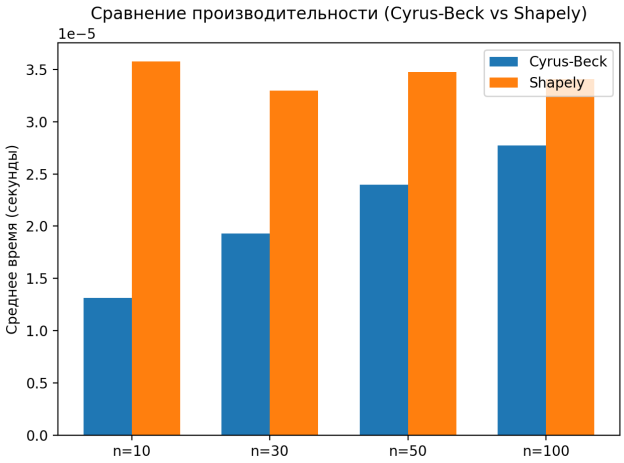

Рис. 15: Сравнение собственной реализации с Shapely.

Заключение

В данной лабораторной работе был изучен и реализован алгоритм отсечения отрезка выпуклым телом. Также были изучены и рассмотрены другие алгоритмы отсечения отрезка выпуклыми телами, такие как:

- Алгоритм Коэна-Сазерленда
- Метод пересечения с полуплоскостями

Математически был описан алгоритм Кируса-Бека, в последствии реализации было проведено сравнение с библиотечным аналогом алгоритма Shapely. Для реализации теста было создано 4 сценария по 500 тестов с разным числом вершин (n). В результате сравнения библиотечный аналог оказался быстрее нашего алгоритма, поскольку Shapely имеет реализацию на более низкоуровневом языке программирования, однако сложность обоих алгоритмов одинакова: $O(n)$.

Список литературы

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

Приложение А

```
import random
import time
import numpy as np
import matplotlib.pyplot as plt

try:
    from shapely.geometry import Polygon, LineString
    shapely_available = True
except ImportError:
    shapely_available = False

def convex_hull(points):
    points = sorted(points, key=lambda p: (p[0], p[1]))

    def cross(o, a, b):
        return (a[0] - o[0])*(b[1] - o[1]) - (a[1] - o[1])*(b[0] - o[0])

    lower = []
    for p in points:
        while len(lower) >= 2 and cross(lower[-2], lower[-1], p) <= 0:
            lower.pop()
        lower.append(p)

    upper = []
    for p in reversed(points):
        while len(upper) >= 2 and cross(upper[-2], upper[-1], p) <= 0:
            upper.pop()
        upper.append(p)

    lower.pop()
    upper.pop()
    hull = lower + upper
    return np.array(hull)

def generate_random_convex_polygon(num_points=7, radius=5):
    points = []
    for _ in range(num_points):
        r = radius * random.random()**0.5
        theta = 2 * np.pi * random.random()
        x = r * np.cos(theta)
        y = r * np.sin(theta)
        points.append((x, y))
    hull = convex_hull(points)
    return hull

def is_segment_outside_bounding(p0, p1, polygon):
    poly_x = polygon[:, 0]
    poly_y = polygon[:, 1]
    min_x, max_x = min(poly_x), max(poly_x)
    min_y, max_y = min(poly_y), max(poly_y)

    seg_x = [p0[0], p1[0]]
    seg_y = [p0[1], p1[1]]
    seg_min_x, seg_max_x = min(seg_x), max(seg_x)
    seg_min_y, seg_max_y = min(seg_y), max(seg_y)

    if seg_max_x < min_x or seg_min_x > max_x:
        return True
    if seg_max_y < min_y or seg_min_y > max_y:
```

```
        return True
    return False

def cyrus_beck_clip(p0, p1, polygon):
    d = p1 - p0
    t_min, t_max = 0.0, 1.0
    n = len(polygon)

    for i in range(n):
        cur = polygon[i]
        nxt = polygon[(i + 1) % n]
        edge_vec = nxt - cur
        normal = np.array([-edge_vec[1], edge_vec[0]], dtype=float)
        numerator   = np.dot(normal, (p0 - cur))
        denominator = np.dot(normal, d)

        if abs(denominator) < 1e-12:
            if numerator < 0:
                return (False, None, None)
            else:
                continue

        t = - numerator / denominator
        if denominator > 0:
            if t > t_max:
                return (False, None, None)
            if t > t_min:
                t_min = t
        else:
            if t < t_min:
                return (False, None, None)
            if t < t_max:
                t_max = t

        if t_min > t_max:
            return (False, None, None)

    if t_max < 0 or t_min > 1:
        return (False, None, None)

    q0 = p0 + t_min * d
    q1 = p0 + t_max * d
    return (True, q0, q1)

def clip_segment_with_cyrus_beck(p0, p1, polygon):
    if is_segment_outside_bounding(p0, p1, polygon):
        return (False, None, None)
    return cyrus_beck_clip(p0, p1, polygon)

def shapely_clip(p0, p1, polygon):
    poly  = Polygon(polygon)
    line  = LineString([p0, p1])
    inter = poly.intersection(line)
    if inter.is_empty:
        return (False, None, None)
    if inter.geom_type == 'LineString':
        coords = list(inter.coords)
        q0 = np.array(coords[0])
        q1 = np.array(coords[-1])
        return (True, q0, q1)
    if inter.geom_type == 'Point':
```

```
        q = np.array(inter.coords[0])
        return (True, q, q)
    if inter.geom_type == 'MultiPoint':
        coords = sorted([np.array(pt.coords[0]) for pt in inter.geoms], key=
            lambda x: (x[0], x[1]))
        return (True, coords[0], coords[-1])
    return (True, None, None)

def compare_performance(num_tests=1000, polygon_size=30, radius=10):
    polygon = generate_random_convex_polygon(num_points=polygon_size, radius
        =radius)

    cyrus_times = []
    shapely_times = []

    for _ in range(num_tests):
        p0 = np.array([random.uniform(-2*radius, 2*radius),
                       random.uniform(-2*radius, 2*radius)])
        p1 = np.array([random.uniform(-2*radius, 2*radius),
                       random.uniform(-2*radius, 2*radius)])

        start = time.perf_counter()
        res1 = clip_segment_with_cyrus_beck(p0, p1, polygon)
        end = time.perf_counter()
        cyrus_times.append(end - start)

        if shapely_available:
            start = time.perf_counter()
            res2 = shapely_clip(p0, p1, polygon)
            end = time.perf_counter()
            shapely_times.append(end - start)
        else:
            shapely_times.append(float('nan'))

    mean_cyrus   = np.mean(cyrus_times)
    mean_shapely = np.mean([t for t in shapely_times if not np.isnan(t)])
    return mean_cyrus, mean_shapely

def demo_comparison():
    scenarios = [
        (500,  10,  10),
        (500,  30,  10),
        (500,  50,  10),
        (500, 100, 15),
    ]

    cyrus_results = []
    shapely_results = []
    labels = []

if __name__ == "__main__":
    demo_comparison()
```