Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

Лабораторная работа

по курсу «Моделирование»

«Построение статической модели»

Студент группы ИУ9-82Б Виленский С. Д.

Преподаватель Домрачева А. Б.

Москва, 2025 г.

ЦЕЛЬ

Формирование общих представлений об анализе и синтезе простых систем, изучение основных понятий в области аналитического моделирования, а также основных понятий теории погрешности.

ПОСТАНОВКА ЗАДАЧИ

Построить модель фрагмента поверхности по заданной матрице высот, обосновать выбор численного метода для приближенного вычисления высоты в заданной точке.

ОПИСАНИЕ МОДЕЛИ И ЕЕ ФОРМАЛИЗАЦИЯ

Для построения триангуляции наивным алгоритмом используется итеративный метод построения. В процессе построения триангуляции хранятся координаты построенных ребер, список ребер, составляющих периметр триангуляции, и расстояния от каждой незадействованной в триангуляции точки до периметра. На первом шаге выбирается случайная пара точек и строится между мини ребро, представляющее периметр на нулевой итерации. На каждое следующей итерации выбирается ближайшая к периметру точка и строятся ребра со всеми точками периметра, не пересекающие уже построенные ребра.

В качестве более эффективного метода построения триангуляции Делоне был выбран итеративный алгоритм “Удаляй и строй”. Для данного алгоритма используется структура данных “Узлы и треугольники”, а также хранится список ребер, составляющих периметр. На первой итерации выбирается случайная тройка точек, из которой строится первый треугольник. На каждой следующей итерации выбирается случайная незадействованная точка и в зависимости от того, находится она внутри одного из построенных треугольников или снаружи всей структуры построенной триангуляции, строятся новые треугольники. В случае нахождения в треугольнике удаляется внешний треугольник и строятся три новых, образованных после соединения новой точки и вершин удаленного треугольника. Далее производится проверка на выполнение условия Делоне со всеми соседними треугольниками и в случае невыполнения производится перестроение соответствующих треугольников. В случае, когда выбранная точка находится вне периметра триангуляции, строятся из данной точки треугольники, смежные со всеми доступными ребрами периметра, и для каждого из построенных треугольников аналогично проверяется условие Делоне и, в случае необходимости, производятся перестроения.

Для нахождения высоты произвольно заданной точки использовались поиск внешнего треугольника и метод пропорций. Поиск внешнего треугольника производился по проверке одинакового положения выбранной точки относительно векторов лежащих на ребрах рассматриваемого треугольника, направленных в одном направлении часовой стрелки. Алгоритм пропорции заключается в нахождении точки H, пересечения прямой, проходящей через некоторую из вершин и выбранную точку, и противолежащего ребра соответствующей некоторой из вершин. Далее вычисляются высоты в соответствии с вычисляемыми пропорциями, разделяемых отрезков соответствующими точками.

ВЫЧИСЛИТЕЛЬНЫЕ ЗАДАЧИ

Для реализации описанных моделей были поставлены следующие вычислительные задачи:

- нахождение расстояния между двумя точками — использовалась стандартная формула Пифагора;
- нахождение расстояния между точкой и отрезком — использовался алгоритм нахождения проекции точки на отрезок и проверка на его выход за границы отрезка, требовалось для наивного алгоритма;
- проверка, находится ли точка выше прямой — использовалась выведенное условие на основе канонического вида формул прямых, требовалась для проверки тройки точек на положительности;
- проверка на положительность тройки точек — использовалась проверка нахождения одной из точек выше прямой, проходящей через другие точки, и проверка положительности наклона построенной прямой, требовалась для проверки вхождения точки в треугольник;

- нахождение формулы описанной окружности около треугольника — использовалась форма записи окружности, проходящей через три точки в форме определителя матрицы размером 4 на 4, требовалось для проверки условия Делоне.

РЕЗУЛЬТАТЫ РАСЧЕТОВ

В результате работы программы были построены две триангуляции, изображенные на рисунках 1 и 2. Анализ построенных триангуляций представлен в таблице 1.

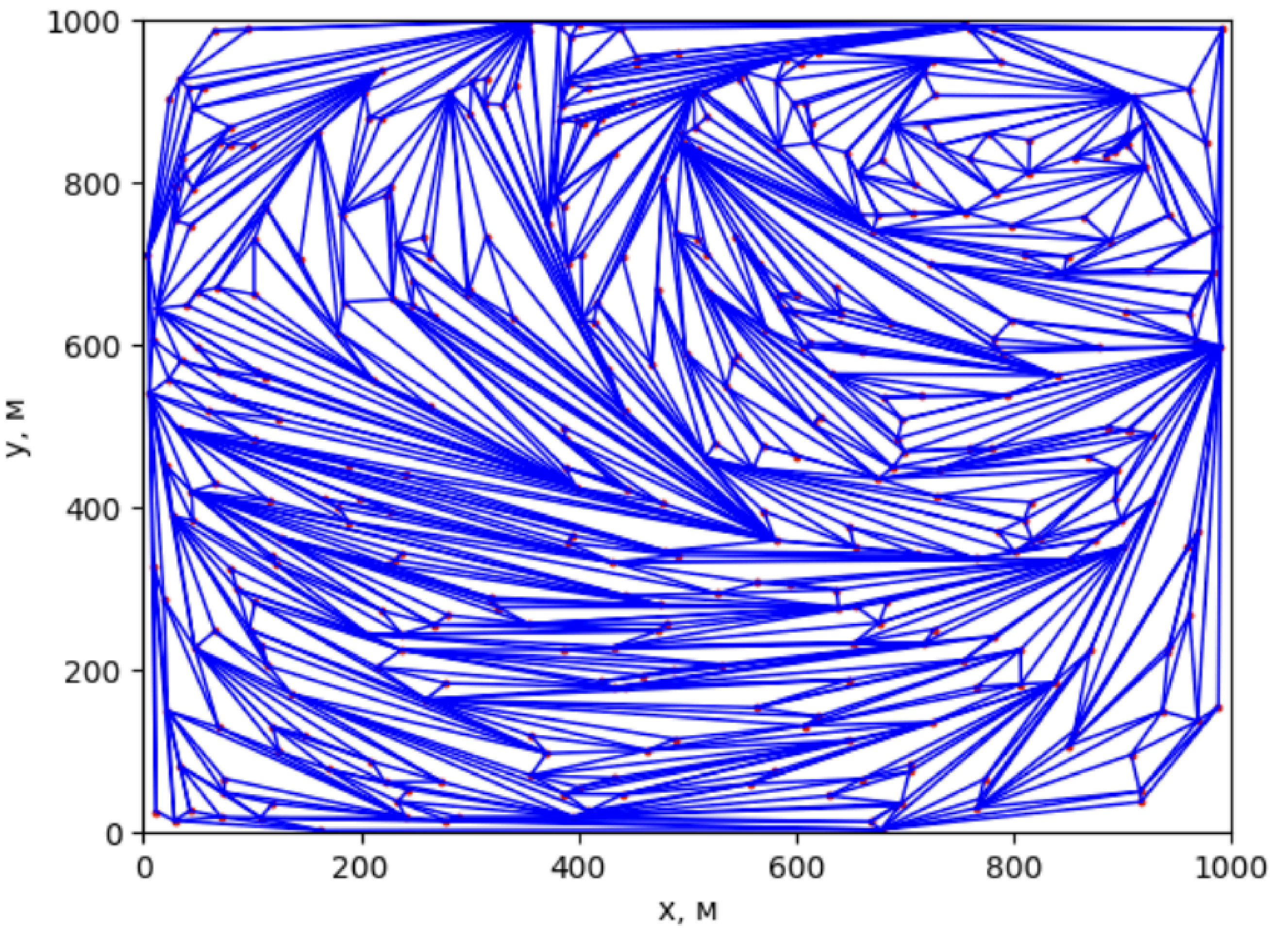

Рисунок 1 − Триангуляция, построенная наивным итеративным алгоритмом

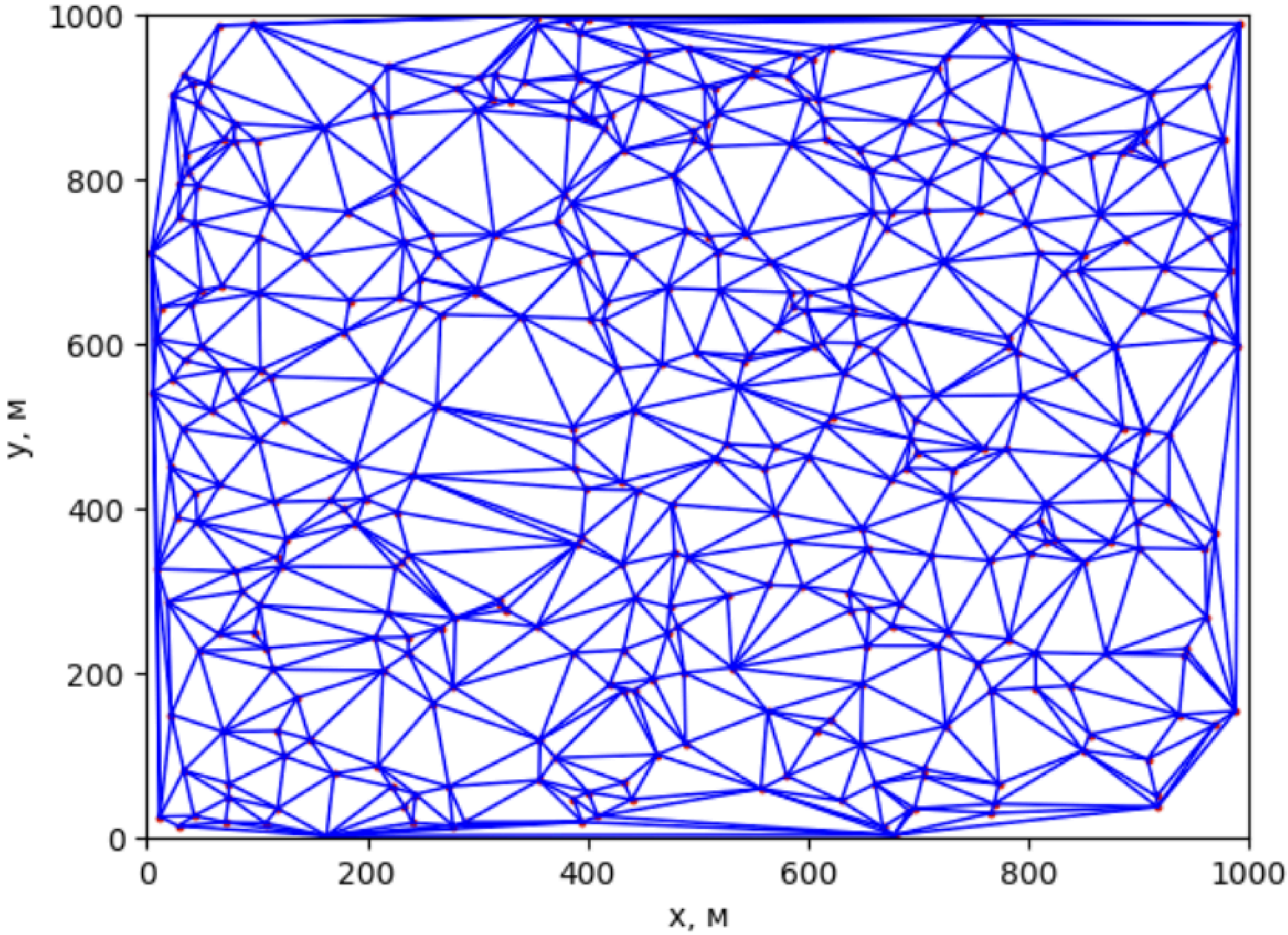

Рисунок 2 − Триангуляция, построенная итеративным алгоритмом “Удаляй и строй”

Таблица 1 − Сравнение выбранных моделей

Модель	Время работы, сек	Сумма длин ребер триангуляции, м	Количество треугольников	Асимптотика алгоритма
Наивный итеративный алгоритм	2.4	152056.3	783	O(N^2)
Итеративный алгоритм “Удаляй и строй”	1.7	73526.0	783	O(N^2 * logN)

ФРАГМЕНТЫ ИСХОДНОГО КОДА

Программные реализации моделей и алгоритмов представлены на листингах 1-4.

Листинг 1 – Реализации алгоритмов линейной алгебры

```
def is_point_upper_line(point, line):
    assert(line[1][0] != line[0][0])
    return (
        point[1] >=
        (point[0] - line[0][0]) / (line[1][0] - line[0][0])
        * (line[1][1] - line[0][1]) + line[0][1]
    )


def sqr_dist_between_points(point1, point2):
    return (point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2


def sqr_dist_from_point_to_line(point, line):
    (x0, y0), ((x1, y1), (x2, y2)) = point, sorted(line)

    line_dist = sqr_dist_between_points((x1, y1), (x2, y2))
    assert(line_dist != 0)

    # Укорочение линии для изключения неоднозначных ситуаций
    x1 -= (x1 - x2) * abs(x1 - x2) / line_dist * 1e-10
    y1 -= (y1 - y2) * abs(y1 - y2) / line_dist * 1e-10
    x2 -= (x2 - x1) * abs(x2 - x1) / line_dist * 1e-10
    y2 -= (y2 - y1) * abs(y2 - y1) / line_dist * 1e-10

    projection = (
         (y2 - y1) * ((y0 - y1) * (x2 - x1) - (x0 - x1) * (y2 - y1)) /
line_dist + x0,
```

```
        (x2 - x1) * ((x0 - x1) * (y2 - y1) - (y0 - y1) * (x2 - x1)) /
line_dist + y0
    )

    if projection[0] > x2:
        return sqr_dist_between_points((x0, y0), (x2, y2))
    if projection[0] < x1:
        return sqr_dist_between_points((x0, y0), (x1, y1))
    return sqr_dist_between_points((x0, y0), projection)

def is_positive_angle_between_three_points(point1, point2, point3):
    return is_point_upper_line(point3, (point1, point2)) != (point1[0]
> point2[0])
```

Листинг 2 – Программная реализация модели наивного итерационного алгоритма

```
while len(dists_points_to_outline) != 0:
      nearest_point = min(dists_points_to_outline, key=lambda point:
dists_points_to_outline[point][1])

    # Побдираем ближайшую грань из перимерта
    min_dist_edge = dists_points_to_outline[nearest_point][0]

    # Добавляем новые ребра в общий учет
    new_egde_first = (min_dist_edge[0], nearest_point)
    new_egde_second = (nearest_point, min_dist_edge[1])

    outline.remove(min_dist_edge)
    outline.add(new_egde_first)
    outline.add(new_egde_second)

    edges.add(new_egde_first)
    edges.add(new_egde_second)
```

```
    del dists_points_to_outline[nearest_point]
    for point in dists_points_to_outline:
        dists_points_to_outline[point] = min(
            dists_points_to_outline[point],
                  (new_egde_first,  sqr_dist_from_point_to_line(point,
new_egde_first)),
                 (new_egde_second,  sqr_dist_from_point_to_line(point,
new_egde_second)),
               key=lambda edge_and_dist: edge_and_dist[1] + (1e+10 if
edge_and_dist[0] not in outline else 0)
        )
```

Листинг 3 – Программная реализация модели итерационного алгоритма “Удаляй и строй”

```
while unused_points:
    new_point = unused_points.pop()

    for triangle in triangles:
        if (
                  is_positive_angle_between_three_points(triangle[0],
triangle[1], new_point) ==
                  is_positive_angle_between_three_points(triangle[1],
triangle[2], new_point) ==
                  is_positive_angle_between_three_points(triangle[2],
triangle[0], new_point)
        ):
            del triangles[triangle]

            new_triangles = [
                (triangle[0], triangle[1], new_point),
                (triangle[1], triangle[2], new_point),
                (triangle[2], triangle[0], new_point)
            ]
```

```
            for new_triangle in new_triangles:
                triangles[new_triangle] = get_coefs_of_extern_triangle_round(*new_triangle)

            for new_triangle in new_triangles:
                check_triangle_by_dilane(new_triangle)

            break
```

Листинг 4 – Программная реализация алгоритма вычисления высоты точки в треугольнике с известными высотами вершин

```
target_point = (450, 120)

for triangle in triangles:
    if (
            is_positive_angle_between_three_points(triangle[0], triangle[1], target_point) ==
            is_positive_angle_between_three_points(triangle[1], triangle[2], target_point) ==
            is_positive_angle_between_three_points(triangle[2], triangle[0], target_point)
    ):

        ((x11, y11), ((x12, y12), (x21, y21), (x22, y22))) = target_point, triangle

        y0 = (
            ((x22 * y21 - x21 * y22) * (y12 - y11) - (x12 * y11 - x11 * y12) * (y22 - y21))
            / ((x22 - x21) * (y12 - y11) - (x12 - x11) * (y22 - y21))
        )
        x0 = (y0 - y21) * (x22 - x21) / (y22 - y21) + x21
```

```
        h0 = (
            points_heights[(x21, y21)] * sqr_dist_between_points((x21, y21), (x0, y0)) ** .5 +
            points_heights[(x22, y22)] * sqr_dist_between_points((x22, y22), (x0, y0)) ** .5
        ) / sqr_dist_between_points((x21, y21), (x22, y22)) ** .5

        h = (
            points_heights[(x12, y12)] * sqr_dist_between_points((x12, y12), (x11, y11)) ** .5 +
            h0 * sqr_dist_between_points((x0, y0), (x11, y11)) ** .5
        ) / sqr_dist_between_points((x12, y12), (x0, y0)) ** .5

        print(h)
        break
else:
    print(None)
```

ВЫВОДЫ

В ходе выполнения лабораторной работы были сформированы общие представления об анализе и синтезе простых систем, изучены основные понятия в области аналитического моделирования, а также основных понятий теории погрешности. В результате выполнения данной работы было выявлено превосходство итеративного алгоритма “Удаляй и строй” над наивным алгоритмам по времени работы и эффективности построения Делоне, оцениваемой как сумма ребер построенной триангуляции.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Скворцов А.В., Костюк Ю.Л. Эффективные алгоритмы построения триангуляции Делоне // Геоинформатика. Теория и практика. Вып. 1. Томск: Изд-во Том. ун-та, 1998. С. 22–47.