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

Факультет: Информатики и систем управления

Кафедра: Теоретической информатики и компьютерных технологий

Домашнее задание №1

«Построение статической модели для визуализации поверхности рельефа»

по курсу: «МОДЕЛИРОВАНИЕ»

Выполнила: студентка группы ИУ9-81

Синявская А. А.

Проверила: Домрачева А.Б.

Москва, 2020

ЦЕЛЬ: выполнить триангуляцию Делоне для заданного набора точек с помощью алгоритма слияния «Разделяй и властвуй» и визуализировать поверхность рельефа с помощью построения статической модели на основе полученной триангуляции и сгенерированной матрицы высот.

ПОСТАНОВКА ЗАДАЧИ:

Дано: конечное множество несовпадающих точек на плоскости, количество которых ≥ 2.

Найти: соединения заданных точек непересекающимися отрезками так, чтобы была образована триангуляция Делоне, построить на её основе статическую модель поверхности.

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Триангуляция Делоне

Триангуляцией называется планарный граф, все внутренние области которого являются треугольниками. Если минимальный многоугольник, охватывающий все треугольники триангуляции, является выпуклым, то такая триангуляции называется выпуклой.

Некоторая триангуляция удовлетворяет условию Делоне, если внутрь окружности, описанной вокруг любого построенного треугольника, не попадает ни одна из заданных точек триангуляции. Триангуляция называется триангуляцией Делоне, если она является выпуклой и удовлетворяет условию Делоне [1].

Алгоритмы слияния работают по принципу разбиения исходного множества точек на несколько подмножеств, построение триангуляций на этих подмножествах, а затем слияние нескольких триангуляций в одно целое.

В алгоритме триангуляции «Разделяй и властвуй» множество точек разбивается на две как можно более равные части с помощью горизонтальных и вертикальных линий. Алгоритм рекурсивно применяется к полученным подмножествам точек, а затем производится объединение полученных подтриангуляций. Рекурсия прекращается при разбиении всего множества на

достаточно маленькие части, которые можно легко протриангулировать каким-нибудь другим простым способом.

Генерация матрицы высот

Для построения триангуляционной модели поверхности необходимо сгенерировать матрицу, содержащую z-координаты визуализируемого ландшафта. Стандартным способом генерации матрицы высот является применение функции шума с ограниченной полосой частот, к примеру, шума Перлина. Это градиентный шум, представляющий из себя набор векторов, рассчитанных для узловых точек. Значения в остальных точках рассчитываются как линейная интерполяция соседствующих с ними узловых точек. За счёт этого значения в близких точках получаются схожи и достигается плавность рельефа.

Построение статической модели

Статические модели описывают явления без развития. Для того, чтобы построить статическую модель, необходимо:

1. Обозначить вид представляемой информации, в данном случае графический.
2. Определить исследуемые и независимые показатели, в данном случае координаты точек.
3. Провести оптимизацию представления данных в зависимости от выбранного алгоритма.
4. Формализовать модель и описать решения необходимых задач при помощи численных методов.

Формализация описывается алгоритмом триангуляции.

ОПИСАНИЕ АЛГОРИТМА:

Триангуляция Делоне

Рассмотрим необходимые для реализации данного алгоритма структуры данных. Для каждой точки исходного множества $v_i \in V$ хранится упорядоченный список смежных ей точек $v_{i1}, \dots, v_{ik}$, где $(v_i, v_{ij}), j = 1, \dots, k$ — ребро триангуляции Делоне. Список двусвязный и циклический, на рисунке 1 можно увидеть пример связей в таком списке. $PRED(v_i, v_{ij})$ обозначает точку v_{ip}, которая появляется сразу же после точки v_i при обходе списка по часовой стрелке. Аналогично определяется структура $SUCC$ для обхода против часовой стрелке. На рисунке 1 $v_5 = PRED(v_1, v_6), v_5 = SUCC(v_1, v_4)$.

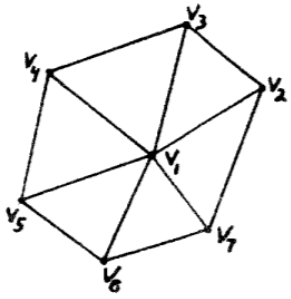
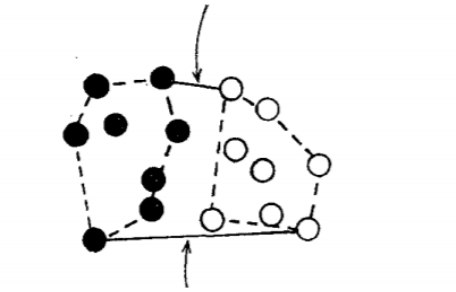


Рисунок 1 – Пример циклического двусвязного списка точек [2].

Если точка v_i находится на выпуклой оболочке $CH(V)$, то первая точка в ее списке смежности — это точка, обозначаемая $FIRST(v_i)$, появляющаяся после v_i при пересечении границы $CH(V)$, в направлении против часовой стрелки. $l(v_i, v_j)$ обозначает отрезок прямой, направленный из v_i в v_j.

Алгоритм начинается с переупорядочивания N точек таким образом, чтобы $v_1 < v_2 < \cdots < v_N$, где $(x_i, y_i) = v_i < v_j$ только в том случае, если $x_i \leq x_j$ и $y_i < y_j$. Затем множество точек V разделяется на два подмножества V_L и V_R, где $V_L = \{v_1, \dots, v_{\lfloor N/2 \rfloor}\}, V_R = \{v_{\lfloor N/2 \rfloor + 1}, \dots, v_N\}$. Начинается рекурсивный процесс построения триангуляций Делоне $DL(V_L)$ и $DL(V_R)$. Для объединения $DL(V_L)$ и $DL(V_R)$ используется выпуклая оболочка $CH(V_L \cup V_R)$, получаемая из объединения выпуклых оболочек $CH(V_L)$ и $CH(V_R)$. Формирование объединения $CH(V_L)$ и $CH(V_R)$ происходит путём нахождения общих верхних и нижних касательных (рисунок 2), которые будут частью финальной триангуляции Делоне.

общая верхняя касательная



общая нижняя касательная

Рисунок 2 – Объединение выпуклых оболочек двух подмножеств точек.

ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ

Для реализации поставленной задачи использован язык Python, для отображения полученной модели задействована библиотека matplotlib. В листингах 1 и 2 представлен программный код, отвечающий за выполнение триангуляции и генерацию шума Перлина для матрицы высот.

Листинг 1. Триангуляция Делоне

```
DIRECT = 1   # по часовой
ALIGNED = 0   # на одной линии
INDIRECT = -1   # против часовой

INSIDE = 1   # внутри окружности
CIRCLE = 0   # на окружности
OUTSIDE = -1   # вне окружности


def inv(point):
    x, y = point
    return (y, x)


def orientation(a, b, c):
    xa, ya = a
    xb, yb = b
    xc, yc = c
    # определитель матрицы:
    d = (xb - xa) * (yc - ya) - (yb - ya) * (xc - xa)
    if d > 0:
        return DIRECT
    elif d == 0:
        return ALIGNED
    return INDIRECT
```

```
def circle_position(a, b, c, d):
    ax, ay = a
    bx, by = b
    cx, cy = c
    dx, dy = d
    m = [[ax - dx, ay - dy, (ax - dx) ** 2 + (ay - dy) ** 2],
         [bx - dx, by - dy, (bx - dx) ** 2 + (by - dy) ** 2],
         [cx - dx, cy - dy, (cx - dx) ** 2 + (cy - dy) ** 2]]
    # определитель матрицы m :
    d = m[0][0] * m[1][1] * m[2][2] + m[0][1] * m[1][2] * m[2][0] + m[0][2] *
m[1][0] * m[2][1] - \
        m[0][0] * m[1][2] * m[2][1] - m[0][1] * m[1][0] * m[2][2] - m[0][2] *
m[1][1] * m[2][0]
    if d > 0:
        return INSIDE
    elif d == 0:
        return CIRCLE
    return OUTSIDE


def count_xy(points):
    n = len(points)
    sum_x, sum_y, sum_sqx, sum_sqy = 0, 0, 0, 0
    for (x, y) in points:
        sum_x += x
        sum_y += y
        sum_sqx += x ** 2
        sum_sqy += y ** 2
    var_x = (sum_sqx / n) - (sum_x / n) ** 2
    var_y = (sum_sqy / n) - (sum_y / n) ** 2
    return var_x, var_y


def median(points, key=None):
    n = len(points)
    points = sorted(points, key=key)
    return points[n // 2]


def median_recursion(points, start, end, key=None, k=7, max=100):
    if end - start <= max:
        return median(points[start:end], key)
    submedians = [median_recursion(points, start + (i * (end - start)) // k,
                                   start + ((i + 1) * (end - start)) // k,
                                   key, k, max)
                  for i in range(k)]
    return median(submedians, key)


def draw_median(l, key=None, k=7, max=100):
    return median_recursion(l, 0, len(l), key, k, max)


def delaunay_triangulation(points):
    succ = {}
    pred = {}
    first = {}

    def delete(a, b):
        sa = succ.pop((a, b))
        sb = succ.pop((b, a))
        pa = pred.pop((a, b))
        pb = pred.pop((b, a))
```

```
        succ[a, pa] = sa
        succ[b, pb] = sb
        pred[a, sa] = pa
        pred[b, sb] = pb

    def insert(a, b, sa, pb):
        pa = pred[a, sa]
        sb = succ[b, pb]
        succ[a, pa] = b
        succ[a, b] = sa
        pred[a, sa] = b
        pred[a, b] = pa
        pred[b, sb] = a
        pred[b, a] = pb
        succ[b, pb] = a
        succ[b, a] = sb

    def hull(x0, y0):
        x, y = x0, y0
        z0 = first[y]
        z1 = first[x]
        z2 = pred[x, z1]
        while True:
            if orientation(x, y, z0) == INDIRECT:
                y, z0 = z0, succ[z0, y]
            elif orientation(x, y, z2) == INDIRECT:
                x, z2 = z2, pred[z2, x]
            else:
                return (x, y)

    def merge(x, y):
        insert(x, y, first[x], pred[y, first[y]])
        first[x] = y

        while True:
            if orientation(x, y, pred[y, x]) == DIRECT:
                y1 = pred[y, x]
                y2 = pred[y, y1]
                while circle_position(x, y, y1, y2) == INSIDE:
                    delete(y, y1)
                    y1 = y2
                    y2 = pred[y, y1]
            else:
                y1 = None

            if orientation(x, y, succ[x, y]) == DIRECT:
                x1 = succ[x, y]
                x2 = succ[x, x1]
                while circle_position(x, y, x1, x2) == INSIDE:
                    delete(x, x1)
                    x1 = x2
                    x2 = succ[x, x1]
            else:
                x1 = None

            if x1 is None and y1 is None:
                break
            elif x1 is None:
                # Рисуем  (x, y1)
                insert(y1, x, y, y)
                y = y1
            elif y1 is None:
                # Рисуем (y, x1)
                insert(y, x1, x, x)
```

```
            x = x1
        elif circle_position(x, y, y1, x1) == INSIDE:
            insert(y, x1, x, x)
            x = x1
        else:
            insert(y1, x, y, y)
            y = y1

    first[y] = x

def triangulate(points):
    n = len(points)

    if n == 2:
        # если есть только две точки, рисуем отрезок [a, b]
        [a, b] = points
        succ[a, b] = pred[a, b] = b
        succ[b, a] = pred[b, a] = a
        first[a] = b
        first[b] = a

    elif n == 3:
        [a, b, c] = points

        if orientation(a, b, c) == DIRECT:
            succ[a, c] = succ[c, a] = pred[a, c] = pred[c, a] = b
            succ[a, b] = succ[b, a] = pred[a, b] = pred[b, a] = c
            succ[b, c] = succ[c, b] = pred[b, c] = pred[c, b] = a
            first[a] = b
            first[b] = c
            first[c] = a
        elif orientation(a, b, c) == INDIRECT:
            succ[a, b] = succ[b, a] = pred[a, b] = pred[b, a] = c
            succ[a, c] = succ[c, a] = pred[a, c] = pred[c, a] = b
            succ[b, c] = succ[c, b] = pred[b, c] = pred[c, b] = a
            first[a] = c
            first[b] = a
            first[c] = b
        else:
            [a, b, c] = sorted(points)
            succ[a, b] = pred[a, b] = succ[c, b] = pred[c, b] = b
            succ[b, a] = pred[b, a] = c
            succ[b, c] = pred[b, c] = a
            first[a] = b
            first[c] = b

    else:
        var_x, var_y = count_xy(points)
        if var_y < var_x:
            median = draw_median(points)
            right = [p for p in points if p >= median]
            left = [p for p in points if p < median]
            triangulate(left)
            triangulate(right)
            x, y = hull(max(left), min(right))
            merge(x, y)
        else:
            median = draw_median(points, key=inv)
            down = [p for p in points if inv(p) < inv(median)]
            up = [p for p in points if inv(p) >= inv(median)]
            triangulate(down)
            triangulate(up)
            x, y = hull(max(down, key=inv), min(up, key=inv))
            merge(x, y)
```

```
    triangulate(points)
    return succ
```

Листинг 2. Шум Перлина

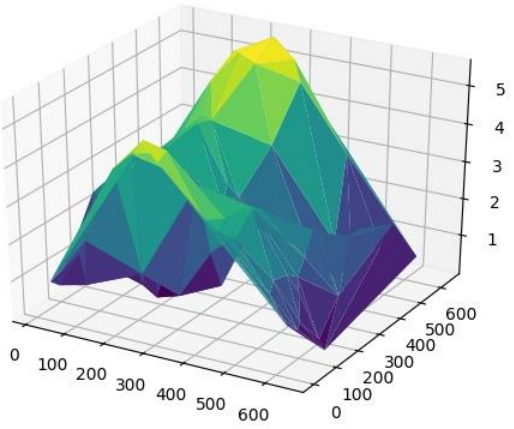
```
class PerlinGenerator(object):

    def __init__(self, octaves=2, tile=()):
        self.octaves = octaves
        self.dimension = 2
        self.tile = tile + (0,) * dimension
        self.scale = 2 * pow(dimension, -0.5)
        self.gradient = {}

    def find_gradient(self):

        if self.dimension == 1:
            return (random.uniform(-1, 1),)

        random_point = [random.gauss(0, 1) for _ in range(self.dimension)]
        scale = sum(n * n for n in random_point) ** -0.5
        return tuple(coord * scale for coord in random_point)

    def dot_noise(self, *point):
        tr_coords = []
        for x in point:
            min = math.floor(x)
            max = min_coord + 1
            tr_coords.append((min, max))

        dots = []
        for tr_point in product(*tr_coords):
            if grid_point not in self.gradient:
                self.gradient[tr_point] = self.find_gradient()
            gradient = self.gradient[tr_point]

            dot = 0
            for i in range(self.dimension):
                dot += gradient[i] * (point[i] - tr_point [i])
            dots.append(dot)


        dim =
        while len(dots) > 1:
            dim -= 1
            s = smoothstep(point[dim] - tr_coords[dim][0])

            future_dots = []
            while dots:
                next_dots.append(linear_interpolation(s, dots.pop(0),
dots.pop(0)))

            dots = future_dots

        return dots[0] * self.scale
    def __call__(self, *point):

        ret = 0
        for octave in range(self.octaves):
            octave2 = 1 << octave
```

```
            new_point = []
            for i, coord in enumerate(point):
                coord *= octave2
                if self.tile[i]:
                    coord %= self.tile[i] * octave2
                new_point.append(coord)
            ret += self.dot_noise(*new_point) / octave2

        ret /= 2 - pow(2,(1 - self.octaves))


        return ret
```

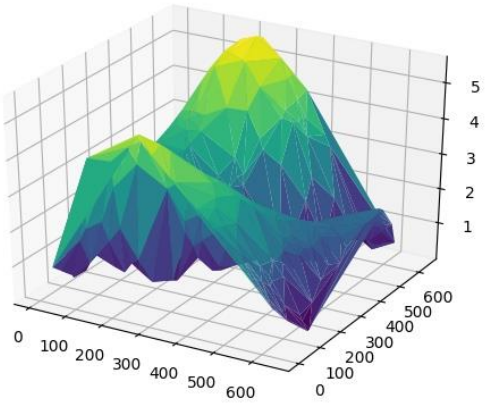
ТЕСТИРОВАНИЕ

Для каждого запуска программы производится триангуляция 100 точек, а затем последовательно идёт добавление точек до 300 и 600 в целях сгущения триангуляционной сетки и уточнения визуализируемой поверхности. На рисунке 3 представлены результаты работы программы.

N = 100



N = 300



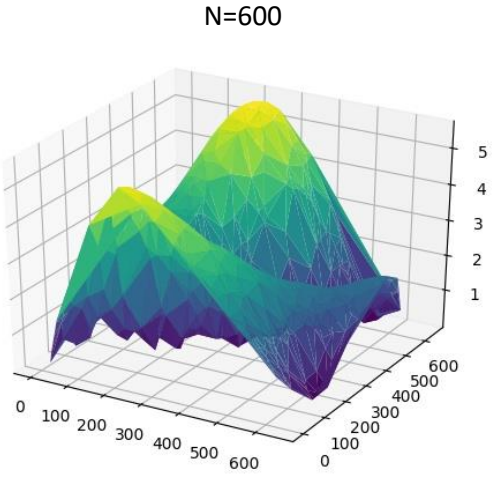


Рисунок 3 – Визуализированная модель поверхности рельефа.

ВЫВОД

В рамках данного домашнего задания изучен метод триангуляции Делоне «разделяй и властвуй», построена триангуляционная модель поверхности произвольного рельефа, матрица высот для которой генерируется путем использования шума Перлина. Сгущение триангуляционной сетки позволяет приблизить рельеф к более реалистичному виду, что является доказательством адекватности построенной модели.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Скворцов А.В., Триангуляция Делоне и ее применение // Томск: Изд-во Том. ун-та, 2002. - 128 с.
2. D. T. Lee 2 and B. J. Schachter, Two Algorithms for Constructing a Delaunay Triangulation // International Journal of Computer and Information Sciences, Vol. 9, No. 3, 1980