|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования 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 и системы управления

КАФЕДРА Теоретическая информатика и компьютерные технологии

**Отчет по лабораторной работе № 2**

***по курсу «Моделирование»***

**«Триангуляция Делоне»**

Студент ИУ9-81Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  Костриця М.И.

(Группа) (Подпись, дата) (Фамилия И.О.)

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Домрачева А. Б. (Подпись, дата) (Фамилия И.О.)

*Москва, 2023 г.*

# Цель работы

Изучить различные алгоритмы триангуляции Делоне и реализовать итеративный алгоритм “удаляй и строй” на произвольном наборе точек.

# Теоретическая часть

Триангуляцией называется планарный граф, все внутренние области которого являются треугольниками.

Выпуклой триангуляцией называется такая триангуляция, для которой минимальный многоугольник, охватывающий все треугольники, будет выпуклым. Триангуляция, не являющаяся выпуклой, называется невыпуклой.

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.

**2.1. Итеративный алгоритм «Удаляй и строй»**

Алгоритм «удаляй и строй»основывается на последовательном добавлении точек в частично построенную триангуляцию. Пусть имеется триангуляция Делоне из n-1 точки, тогда при добавлении очередной n-й точки надо выполнить следующие шаги:

1. Локализовать точку, то есть найти построенный ранее треугольник, в который попадает точка. Если точка попадает не внутрь триангуляции, то найти ближайший к ней треугольник.

2. Делаем один из следующих шагов в зависимости от положения точки.

(1) Если точка попала на ранее вставленную (𝜖-окрестность), то она, как правило, отбрасывается.

(2) Если точка попала на некоторое ребро, то оно разбивается на два новых. Оба смежных с ребром треугольника также делятся на два меньших.

(3) Если точка попала строго внутрь какого-нибудь треугольника, то он делится на три новых.

(4) Если точка попала вне триангуляции, то строится один или более новых.

3. После добавления новой точки условие Делоне может быть нарушено, поэтому надо проверить все вновь построенные треугольники и соседние с ними, а также выполнить после этого необходимые перестроения.

1. **Практическая реализация**
2. **import** numpy **as** np
3. **from** math **import** sqrt
4. **import** matplotlib.pyplot **as** plt
5. **import** matplotlib.tri
6. **import matplotlib.collections**

9. **class** Delaunay:
10. **def** \_\_init\_\_(self, center=(0, 0), radius=9999):
11. **center = np.asarray(center)**
12. self.coords = [center + radius \* np.array((-1, -1)),
13. center + radius \* np.array((+1, -1)),
14. center + radius \* np.array((+1, +1)),
15. center + radius \* np.array((-1, +1))]
17. self.triangles = {}
18. self.circles = {}
20. t1 = (0, 1, 3)
21. **t2 = (2, 3, 1)**
22. self.triangles[t1] = [t2, None, None]
23. self.triangles[t2] = [t1, None, None]
25. **for** t **in** self.triangles:
26. **self.circles[t] = self.circum\_center(t)**
28. **def** circum\_center(self, tri):
29. pts = np.asarray([self.coords[v] **for** v **in** tri])
30. pts2 = np.dot(pts, pts.T)
31. **a = np.bmat([[2 \* pts2, [[1],**
32. [1],
33. [1]]],
34. [[[1, 1, 1, 0]]]])
36. **b = np.hstack((np.sum(pts \* pts, axis=1), [1]))**
37. x = np.linalg.solve(a, b)
38. bary\_coords = x[:-1]
39. center = np.dot(bary\_coords, pts)
41. **radius = np.sum(np.square(pts[0] - center))**
42. **return** (center, radius)
44. **def** inCircleFast(self, tri, p):
45. center, radius = self.circles[tri]
46. **return np.sum(np.square(center - p)) <= radius**
48. **def** addPoint(self, p):
49. p = np.asarray(p)
50. idx = len(self.coords)
51. **print("coords[", idx,"] ->",p)**
52. self.coords.append(p)
54. bad\_triangles = []
55. **for** temp **in** self.triangles:
56. **if self.inCircleFast(temp, p):**
57. bad\_triangles.append(temp)
59. boundary = []
60. temp = bad\_triangles[0]
61. **edge = 0**
62. **while** True:
63. **print**('-'\*10)
64. **print**(f"temp: {temp}")
65. tri\_op = self.triangles[temp][edge]
66. **print(f'tri\_op: {tri\_op}')**
67. **if** tri\_op **not** **in** bad\_triangles:
68. **print**("not in bad")
69. boundary.append((temp[(edge + 1) % 3], temp[(edge - 1) % 3], tri\_op))
70. edge = (edge + 1) % 3
71. **print(f"edge: {edge}")**
72. **print**(f"boundary: {boundary}")
73. **if** boundary[0][0] == boundary[-1][1]:
74. **break**
75. **else**:
76. **print("in bad")**
77. edge = (self.triangles[tri\_op].index(temp) + 1) % 3
78. temp = tri\_op
79. **print**(f"edge: {edge}")
80. **print**(f"temp: {temp}")
82. **print**('-' \* 10)
83. **for** temp **in** bad\_triangles:
84. **del** self.triangles[temp]
85. **del** self.circles[temp]
87. new\_triangles = []
88. **for** (e0, e1, tri\_op) **in** boundary:
89. temp = (idx, e0, e1)
90. self.circles[temp] = self.circum\_center(temp)
91. **self.triangles[temp] = [tri\_op, None, None]**
93. **if** tri\_op:
94. **for** i, neigh **in** enumerate(self.triangles[tri\_op]):
95. **if** neigh:
96. **if e1 in neigh and e0 in neigh:**
97. self.triangles[tri\_op][i] = temp
99. new\_triangles.append(temp)
101. **n = len(new\_triangles)**
102. **for** i, temp **in** enumerate(new\_triangles):
103. self.triangles[temp][1] = new\_triangles[(i + 1) % n]
104. self.triangles[temp][2] = new\_triangles[(i - 1) % n]
106. **def exportTriangles(self):**
107. **return** [(a - 4, b - 4, c - 4)
108. **for** (a, b, c) **in** self.triangles **if** a > 3 **and** b > 3 **and** c > 3]
110. **def** exportCircles(self):
111. **return [(self.circles[(a, b, c)][0], sqrt(self.circles[(a, b, c)][1]))**
112. **for** (a, b, c) **in** self.triangles **if** a > 3 **and** b > 3 **and** c > 3]

115. **def** plot\_one(num\_seeds, square\_size, addCircles=False):
116. **seeds = square\_size \* np.random.random((num\_seeds, 2))**
117. **print**("seeds:**\n**", seeds)
118. **print**("BBox Min:", np.amin(seeds, axis=0),
119. "Bbox Max: ", np.amax(seeds, axis=0))
121. **center = np.mean(seeds, axis=0)**
122. dt = Delaunay(center, 50 \* square\_size)
124. **for** s **in** seeds:
125. dt.addPoint(s)
127. **print**(len(dt.exportTriangles()), "Delaunay triangles")
129. fig, ax = plt.subplots()
130. ax.margins(0.1)
131. **ax.set\_aspect('equal')**
132. plt.axis([-1, square\_size + 1, -1, square\_size + 1])
134. cx, cy = zip(\*seeds)
135. dt\_tris = dt.exportTriangles()
136. **ax.triplot(matplotlib.tri.Triangulation(cx, cy, dt\_tris))**
138. **if** addCircles:
139. **for** c, r **in** dt.exportCircles():
140. ax.add\_artist(plt.Circle(c, r, color='k', fill=False, ls='dotted'))
142. plt.show()

145. **def** step\_by\_step\_plot(num\_seeds, square\_size, addCircles=False):
146. **seeds = square\_size \* np.random.random((num\_seeds, 2))**
147. **print**("seeds:**\n**", seeds)
148. **print**("BBox Min:", np.amin(seeds, axis=0),
149. "Bbox Max: ", np.amax(seeds, axis=0))
151. **center = np.mean(seeds, axis=0)**
153. dt2 = Delaunay(center, 50 \* square\_size)
154. **for** i, s **in** enumerate(seeds):
155. **print**("Inserting seed", i, s)
156. **dt2.addPoint(s)**
157. **if** i > 1:
158. fig, ax = plt.subplots()
159. ax.margins(0.1)
160. ax.set\_aspect('equal')
161. **plt.axis([-1, square\_size + 1, -1, square\_size + 1])**
162. **for** i, v **in** enumerate(seeds):
163. plt.annotate(i, xy=v)
164. **for** t **in** dt2.exportTriangles():
165. polygon = [seeds[i] **for** i **in** t]
166. **plt.fill(\*zip(\*polygon), fill=False, color="b")**
167. **if** addCircles:
168. **for** c, r **in** dt2.exportCircles():
169. ax.add\_artist(plt.Circle(c, r, color='k', fill=False, ls='dotted'))
170. plt.show()

173. **if** \_\_name\_\_ == '\_\_main\_\_':
174. num\_seeds = 50
175. square\_size = 1000
177. plot\_one(
178. num\_seeds=num\_seeds,
179. square\_size=square\_size,
180. addCircles=False
181. **)**
183. '''step\_by\_step\_plot(
184. num\_seeds=num\_seeds,
185. square\_size=square\_size,
186. **addCircles=False**
187. )'''

190. **Результаты**

Результаты работы программы – построение триангуляции Делоне представлены на рисунке 1.

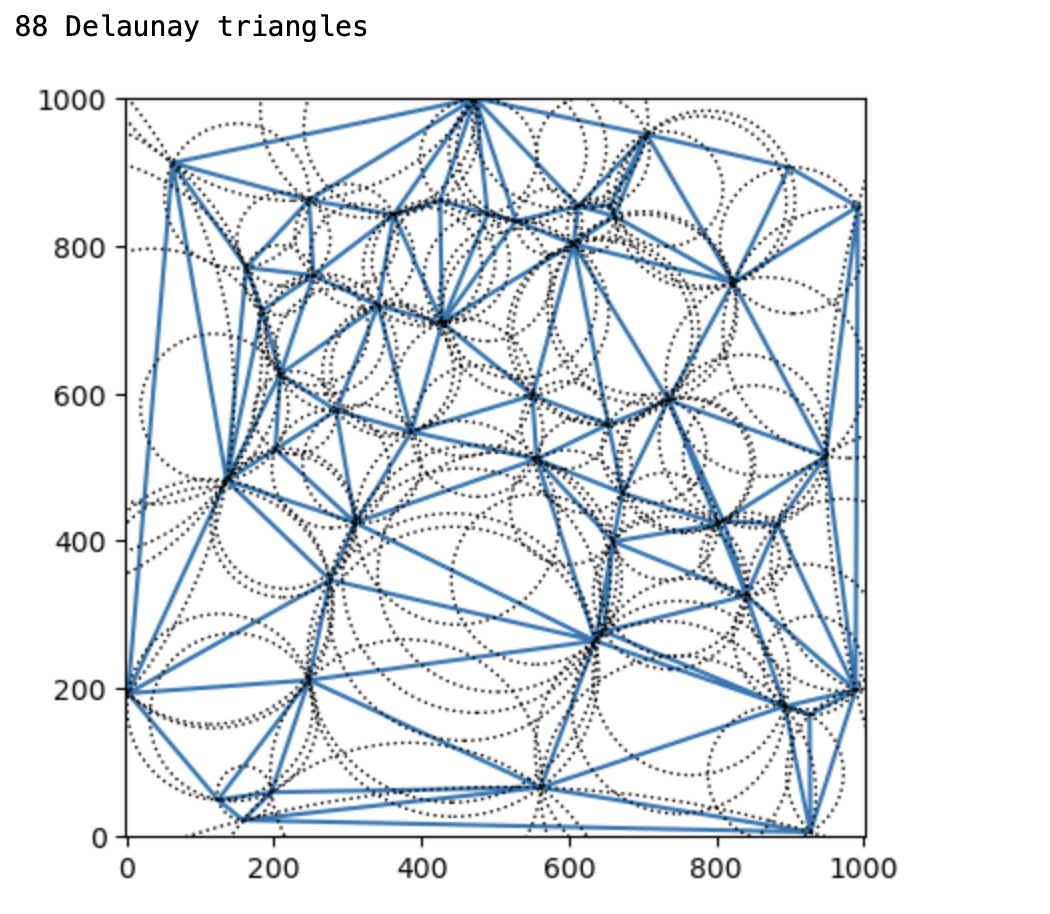
****

Рисунок 1 — триангуляция Делоне

1. **Вывод**

В ходе выполнения лабораторной работы были изучены алгоритмы построения триангуляции Делоне и реализован итеративный алгоритм “Удаляй и строй” на Python, а также проверка условия Делоне.

Он проигрывает по быстродействию у многих существующих алгоритмов, так как имеет необходимость перестраивать множество треугольников.