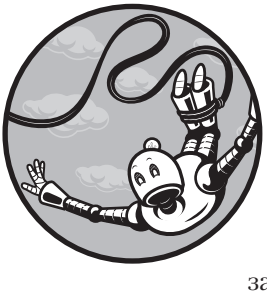
7

Геометрия



Людям присуще интуитивное восприятие геометрии. Каждый раз, когда вы толкаете диван по узкому коридору, рисуете картину в Pictionary или оцениваете, на каком расстоянии от вас находится другая машина на дороге, вы занимаетесь сложным геометрическим осмыслением, которое часто зависит от подсознательно усвоенных вами алгоритмов. К настоящему моменту вас уже не удивит тот факт, что геометрия естественно подходит для алгоритмических рассуждений.

В этой главе мы воспользуемся геометрическим алгоритмом для решения задачи почтмейстера. Начнем с формулировки задачи и посмотрим, как она решается с использованием диаграмм Вороного. Оставшаяся часть главы объясняет, как сгенерировать это решение алгоритмическим методом.

Задача почтмейстера

Предположим, в одном городе среди жилых домов размещаются четыре почтовых отделения (рис. 7.1).

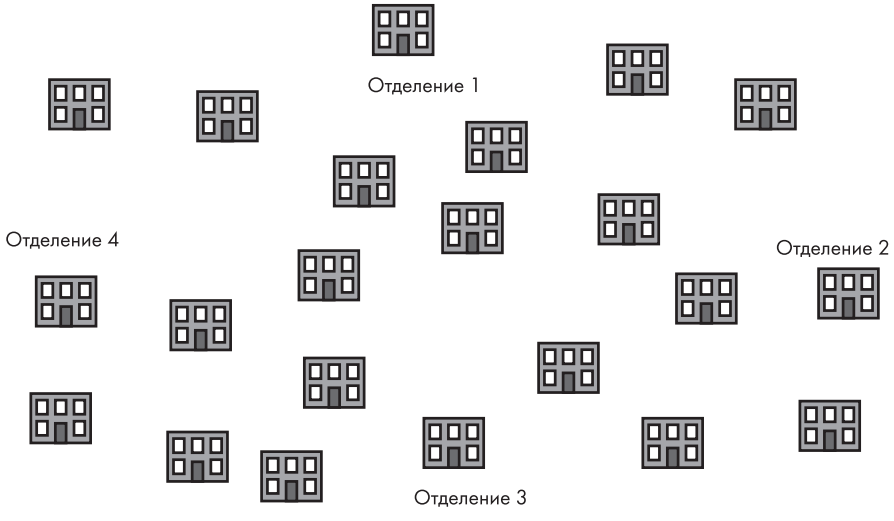


Рис. 7.1. Город с почтовыми отделениями

Система доставки почты между отделениями явно нуждается в оптимизации. Может оказаться, что почтовое отделение 4 должно доставлять почту в дом, который находится ближе к отделениям 2 и 3; в то же время почтовое отделение 2 должно доставлять почту в дом, который находится ближе к отделению 4 (рис. 7.2).

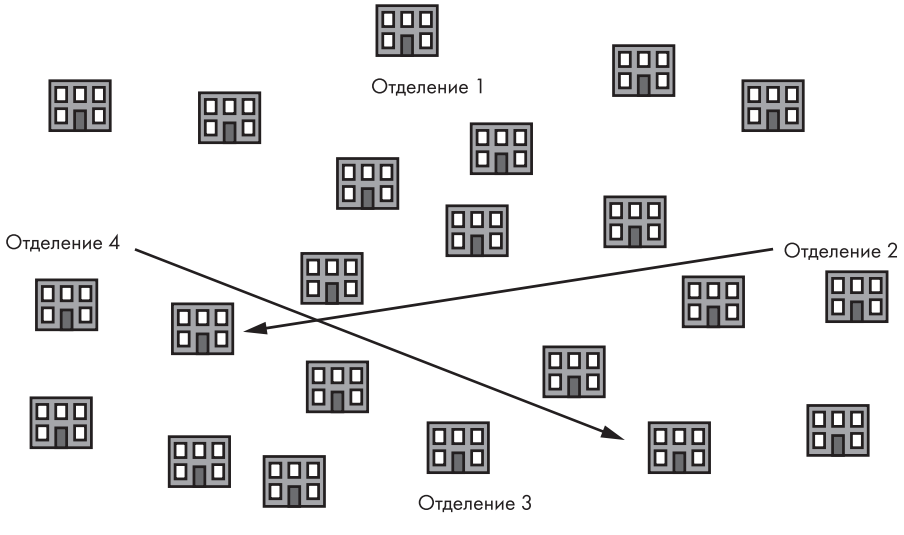


Рис. 7.2. Неэффективная доставка почты из почтовых отделений 2 и 4

Систему доставки нужно реорганизовать так, чтобы каждый дом получал почтовые отправления из наиболее подходящего почтового отделения. Таковым может быть то, в котором больше всего свободного персонала, или то, в котором имеется подходящий транспорт для доставки, или то, которое располагает необходимой информацией для нахождения всех адресов в окрестностях. Но, скорее всего, идеальным почтовым отделением будет попросту ближайшее. Возможно, вы заметили, что эта задача отчасти напоминает задачу коммивояжера — по крайней мере в том смысле, что объекты перемещаются и мы хотим сократить расстояние перемещения. Однако задача коммивояжера является задачей оптимизации порядка обхода заданного маршрута одним коммивояжером, а в новой задаче оптимизируется назначение маршрутов для нескольких почтальонов. На самом деле эту задачу и задачу коммивояжера можно решать последовательно, чтобы достичь максимальной эффективности: после того как вы решите, какое почтовое отделение должно осуществлять доставку в те или иные дома, отдельные почтальоны могут воспользоваться задачей коммивояжера для определения порядка обхода этих домов.

Простейший подход к решению данной задачи, которую можно назвать *задачей почтмейстера*, основан на поочередном рассмотрении каждого дома, вычислении расстояния между домом и каждым из четырех почтовых отделений и назначении ближайшего отделения для доставки почты в дом.

У такого подхода есть ряд недостатков. Во-первых, он не предоставляет простой возможности назначения при появлении новых домов; каждый вновь построенный дом должен пройти через тот же трудоемкий процесс сравнения со всеми существующими почтовыми отделениями. Во-вторых, вычисления на уровне отдельных домов не позволяют собрать информацию о регионе в целом. Например, может оказаться, что весь район находится в шаге от одного почтового отделения, но отдален на много миль от других почтовых отделений. Было бы лучше сделать вывод, что весь район должен обслуживаться одним и тем же ближайшим почтовым отделением. К сожалению, наш метод требует повторения вычислений для каждого дома в районе только для того, чтобы каждый раз получать один и тот же результат. Вычисляя расстояния для каждого дома по отдельности, мы повторяем работу, которую не пришлось бы повторять, если бы мы могли сделать какие-то обобщающие выводы о целых регионах. И эта работа будет только накапливаться. В мегаполисах с десятками миллионов жителей, множеством почтовых отделений и быстрыми темпами строительства такой подход будет излишне медленным и затратным по вычислительным ресурсам.

Другое, более элегантное решение рассматривает всю карту в целом и разбивает ее на регионы, каждый из которых представляет зону обслуживания одного почтового

отделения. В нашем гипотетическом городе для этого достаточно провести всего две линии (рис. 7.3).

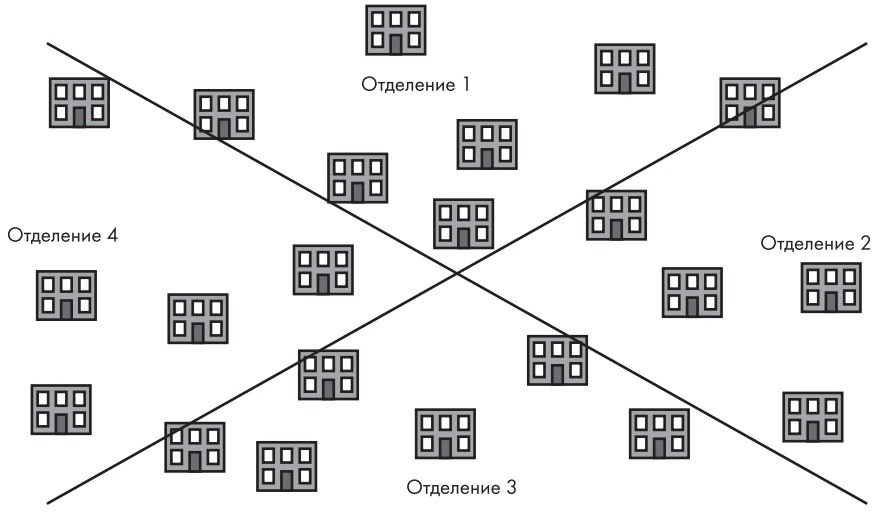


Рис. 7.3. Диаграмма Вороного, разбивающая город на регионы оптимальной доставки

Регионы обозначают ближайшие области, в которых для каждого отдельного дома ближайшим почтовым отделением является то, которое находится в данном регионе. Теперь, когда вся карта была разбита на регионы, можно легко закрепить любое вновь построенное здание за ближайшим почтовым отделением. Для этого достаточно проверить, какому региону оно принадлежит.

Диаграмма, которая разбивает карту на регионы, приближенные к одному почтовому отделению, называется *диаграммой Вороного*. У этих диаграмм долгая история, которая начинается еще со времен Рене Декарта. Они использовались для анализа размещения водных колонок в Лондоне и сбора доказательств относительно распространения холеры. В наши дни эти диаграммы продолжают использоваться в физике и математике для представления кристаллических структур. В данной главе будет описан алгоритм генерирования диаграммы Вороного для произвольного набора точек, позволяющий решить задачу почтмейстера.

Треугольники: краткий курс

Сделаем шаг назад и начнем с простейших элементов исследуемых алгоритмов. В геометрии простейшим элементом анализа является точка. Мы будем представ-

лять точки в виде списков с двумя элементами: координатой x и координатой y, как в следующем примере:

```
point = [0.2,0.8]
```

На следующем уровне сложности точки соединяются, образуя треугольники. Треугольник представляется списком из трех точек:

```
triangle = [[0.2,0.8],[0.5,0.2],[0.8,0.7]]
```

Определим также вспомогательную функцию, которая преобразует набор из трех разрозненных точек в треугольник. Эта маленькая функция просто включает три точки в список и возвращает результат:

```
def points_to_triangle(point1,point2,point3):
    triangle = [list(point1),list(point2),list(point3)]
    return(triangle)
```

Будет полезно наглядно представить треугольники, с которыми мы работаем. Создадим простую функцию, которая получает произвольный треугольник и рисует его. Для начала воспользуемся функцией `genlines()`, определенной в главе 6. Напомню, что функция получает набор точек и преобразует их в линии. Эта функция тоже очень проста, она всего лишь присоединяет точки к списку `lines`:

```
def genlines(listpoints,itinerary):
    lines = []
    for j in range(len(itinerary)-1):
        lines.append([listpoints[itinerary[j]],listpoints[itinerary[j+1]]])
    return(lines)
```

Далее напишем простую функцию графического вывода. Она получает переданный треугольник, разбивает его на значения x и y, вызывает `genlines()` для создания набора отрезков на базе этих значений, рисует точки и линии и, наконец, сохраняет изображение в файле `.png`. Для рисования используется модуль `pylab`, а для создания коллекции отрезков — код модуля `matplotlib`. Эта функция приведена в листинге 7.1.

Листинг 7.1. Функция рисования треугольников

```
import pylab as pl
from matplotlib import collections as mc
def plot_triangle_simple(triangle,thename):
    fig, ax = pl.subplots()

    xs = [triangle[0][0],triangle[1][0],triangle[2][0]]
```

```
    ys = [triangle[0][1],triangle[1][1],triangle[2][1]]

    itin=[0,1,2,0]

    thelines = genlines(triangle,itin)

    lc = mc.LineCollection(genlines(triangle,itin), linewidths=2)

    ax.add_collection(lc)

    ax.margins(0.1)
    pl.scatter(xs, ys)
    pl.savefig(str(thename) + '.png')
    pl.close()
```

Теперь можно выбрать три точки, преобразовать их в треугольник и нарисовать этот треугольник всего одной строкой:

```
plot_triangle_simple(points_to_triangle((0.2,0.8),(0.5,0.2),(0.8,0.7)),'tri')
```

Результат показан на рис. 7.4.

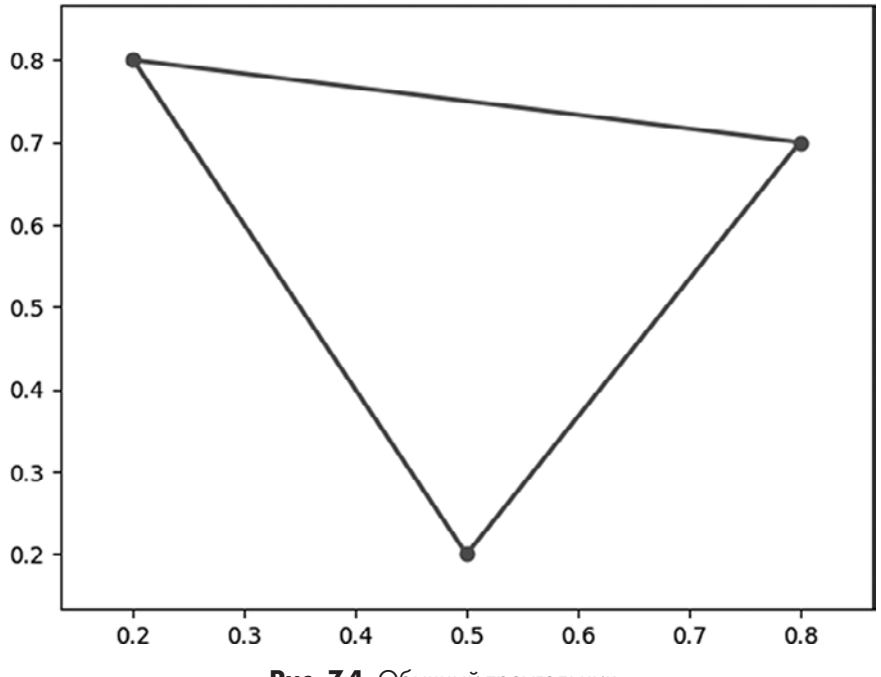


Рис. 7.4. Обычный треугольник

Будет полезна также функция, которая позволяет вычислить расстояние между любыми двумя точками по теореме Пифагора:

```
def get_distance(point1,point2):
    distance = math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)
    return(distance)
```

Остается напомнить некоторые распространенные термины из области геометрии:

- *равносторонний* — термин для описания фигуры, у которой все стороны имеют равную длину;
- *перпендикулярный* — термин для описания двух линий, расположенных под углом 90 градусов;
- *вершина* — точка, в которой встречаются две стороны геометрической фигуры.

Продвинутая теория треугольников

Ученый и философ Готфрид Вильгельм Лейбниц считал, что наш мир является лучшим из всех возможных миров, поскольку имеет «максимально простые законы, из которых вытекает наибольшее богатство явлений». Лейбниц считал, что научные законы могут быть сведены к нескольким простым правилам, но эти правила ведут к нескончаемому разнообразию и красоте мира, которые мы наблюдаем. Возможно, это и не относится к Вселенной, но безусловно истинно для треугольника. Начиная с чего-то настолько концептуально простого, как треугольник (фигура с тремя сторонами), мы входим в мир, чрезвычайно богатый явлениями.

Поиск центра описанной окружности

Чтобы вы начали понимать все богатство мира треугольников, рассмотрим простой алгоритм, который можно опробовать с любым треугольником.

1. Найти среднюю точку каждой стороны треугольника.
2. Провести линию от каждой вершины к середине стороны, противоположной от вершины.

Выполнив этот алгоритм, вы получите результат, сходный с тем, что показан на рис. 7.5.

Интересно, что все проведенные вами линии сходятся в одной точке, которая становится своего рода центром треугольника. Все три линии сходятся в одной точке

независимо от того, с какого треугольника вы начинаете. Эта точка иногда называется *центроидом* треугольника, и она всегда находится внутри, в месте, которое можно назвать центром треугольника.

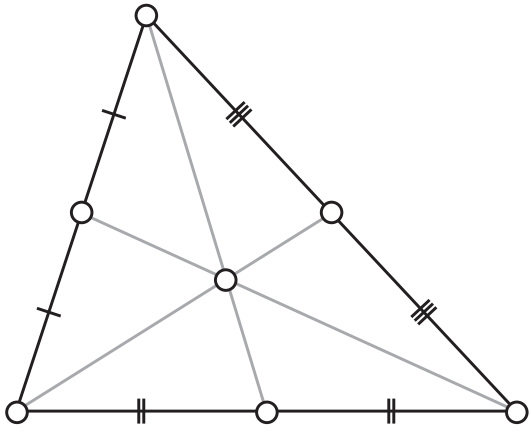


Рис. 7.5. Центроид треугольника (источник: Wikimedia Commons)

Некоторые фигуры — например, окружности — всегда имеют одну точку, которую можно однозначно назвать центром фигуры. Но с треугольниками дело обстоит иначе: центроид — всего лишь одна центроподобная точка, но есть и другие, которые тоже можно считать центрами. Рассмотрим новый алгоритм для произвольного треугольника.

1. Найти среднюю точку каждой стороны треугольника.
2. Провести линию, перпендикулярную к каждой стороне, через среднюю точку этой стороны.

В данном случае линии обычно не проходят через вершины, как это было при нахождении центроида. Сравните рис. 7.5 с рис. 7.6.

Обратите внимание: все линии снова сходятся в одной точке, и она не является центроидом, но часто находится внутри треугольника. У этой точки есть еще одно интересное свойство: она является центром уникальной окружности, которая проходит через все три вершины треугольника. Здесь также проявляется богатство явлений применительно к треугольникам: у каждого треугольника существует одна уникальная окружность, проходящая через все три его вершины. Эта окружность называется окружностью, *описанной вокруг треугольника*. Только что описанный алгоритм находит центр этой окружности.

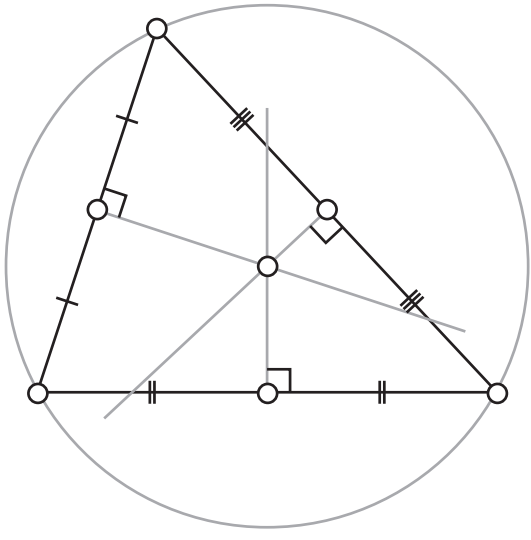


Рис. 7.6. Центр описанной окружности (источник: Wikimedia Commons)

Центр описанной окружности, как и центроид, является точкой, которую можно назвать центром треугольника, но это не единственные кандидаты — в энциклопедии по адресу https://faculty.evansville.edu/ck6/encyclopedia/ETC.html приведен список 40 000 (на данный момент) точек, которые по тем или иным причинам могут называться центрами треугольника. Как сказано в самой энциклопедии, определение центра треугольника «удовлетворяется бесконечным множеством объектов, из которых когда-либо будет опубликовано только конечное подмножество». Интересно, что, начав с трех простых точек и трех прямых отрезков, мы получаем потенциально бесконечную энциклопедию уникальных центров — Лейбниц был бы в восторге.

Мы можем написать функцию, которая находит центр описанной окружности и ее радиус для любого заданного треугольника. Эта функция использует преобразование в комплексные числа. На входе она получает треугольник, а на выходе возвращает координаты центра и радиус:

```
def triangle_to_circumcenter(triangle):
    x,y,z = complex(triangle[0][0],triangle[0][1]),
            complex(triangle[1][0],triangle[1][1]), \
    complex(triangle[2][0],triangle[2][1])
    w = z - x
    w /= y - x
    c = (x-y) * (w-abs(w)**2)/2j/w.imag - x
    radius = abs(c + x)
    return((0 - c.real,0 - c.imag),radius)
```

Подробности того, как эта функция вычисляет центр и радиус, весьма сложны. Я не стану на них задерживаться — при желании вы можете проанализировать код самостоятельно.

Расширение графического вывода

Теперь, когда вы научились находить центр описанной окружности и ее радиус для любого треугольника, улучшим функцию `plot_triangle()`, чтобы она могла вывести всю информацию. Новая функция приведена в листинге 7.2.

Листинг 7.2. Улучшенная функция plot_triangle(), которая рисует описанную окружность и ее центр

```
def plot_triangle(triangles,centers,radii,thename):
    fig, ax = pl.subplots()
    ax.set_xlim([0,1])
    ax.set_ylim([0,1])
    for i in range(0,len(triangles)):
        triangle = triangles[i]
        center = centers[i]
        radius = radii[i]
        itin = [0,1,2,0]
        thelines = genlines(triangle,itin)
        xs = [triangle[0][0],triangle[1][0],triangle[2][0]]
        ys = [triangle[0][1],triangle[1][1],triangle[2][1]]

        lc = mc.LineCollection(genlines(triangle,itin), linewidths = 2)

        ax.add_collection(lc)
        ax.margins(0.1)
        pl.scatter(xs, ys)
        pl.scatter(center[0],center[1])

        circle = pl.Circle(center, radius, color = 'b', fill = False)

        ax.add_artist(circle)
    pl.savefig(str(thename) + '.png')
    pl.close()
```

Начнем с добавления двух новых аргументов: переменной `centers`, которая содержит список соответствующих центров описанных окружностей всех треугольников, и переменной `radii`, которая содержит список радиусов описанных окружностей всех треугольников. Обратите внимание: передаваемые аргументы состоят из списков, так как функция предназначена для рисования нескольких треугольников вместо одного. Для рисования кругов используется

функциональность модуля `pylab`. Позднее мы будем работать с несколькими треугольниками одновременно, и тогда нам пригодится функция, которая может рисовать сразу несколько треугольников вместо одного. Мы включим в функцию цикл, который перебирает все треугольники и центры и последовательно рисует их.

Вызовем эту функцию со списком треугольников, который определим:

```
triangle1 = points_to_triangle((0.1,0.1),(0.3,0.6),(0.5,0.2))
center1,radius1 = triangle_to_circumcenter(triangle1)
triangle2 = points_to_triangle((0.8,0.1),(0.7,0.5),(0.8,0.9))
center2,radius2 = triangle_to_circumcenter(triangle2)
plot_triangle([triangle1,triangle2],[center1,center2],[radius1,radius2],'two')
```

Вывод показан на рис. 7.7.

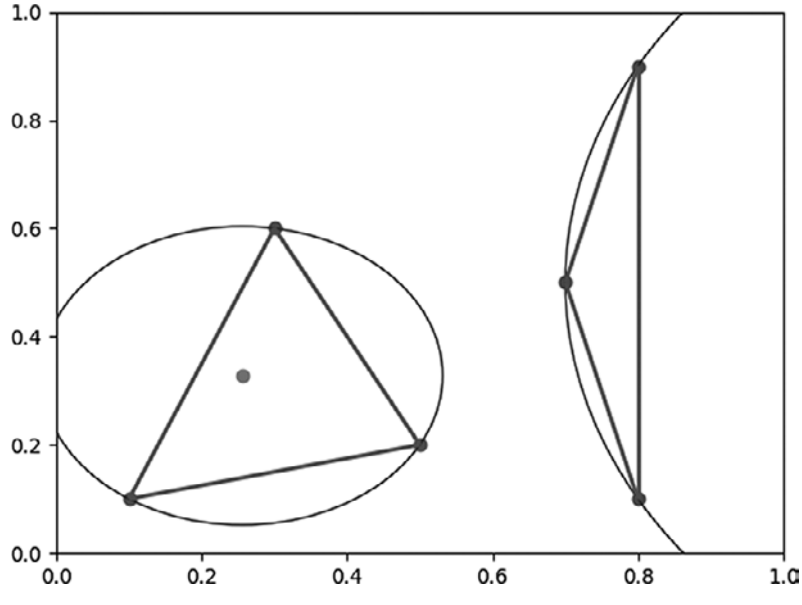


Рис. 7.7. Два треугольника с описанными окружностями и центрами описанных окружностей

Обратите внимание: первый треугольник почти равносторонний. Его описанная окружность мала, а ее центр лежит внутри. Второй треугольник — узкий и вытянутый. Его описанная окружность велика, а ее центр находится далеко за границами данного изображения. Каждый треугольник имеет уникальную описанную

окружность, и для разных треугольников создаются разные описанные окружности. Вы можете самостоятельно исследовать разные треугольники и описанные окружности, которые им соответствуют. Позднее различия между описанными окружностями треугольников начнут играть важную роль.

Триангуляция Делоне

Вы готовы к первому серьезному алгоритму данной главы. Он получает множество точек на входе и возвращает множество треугольников на выходе. В этом контексте преобразование множества точек во множество треугольников называется *триангуляцией*.

Функция `points_to_triangle()`, определенная ближе к началу главы, является простейшим алгоритмом триангуляции из всех возможных. Тем не менее этот алгоритм весьма ограничен, поскольку работает только в том случае, если ему передаются ровно три входные точки. Если вы хотите выполнить триангуляцию более чем для трех точек, то способов триангуляции неизбежно будет несколько. Например, рассмотрим два разных способа триангуляции для тех же семи точек, изображенных на рис. 7.8.

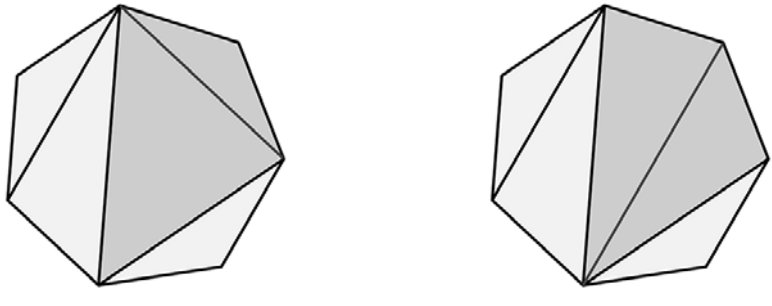


Рис. 7.8. Два разных способа триангуляции семи точек (Wikimedia Commons)

Более того, для этого правильного семиугольника существуют 42 возможных способа триангуляции (рис. 7.9).

Если у вас более семи точек и они не образуют правильную фигуру, то количество возможных триангуляций может достичь невероятных высот. Триангуляцию можно выполнить вручную, соединяя точки на бумаге с карандашом. Но, конечно, то же самое можно быстрее и лучше сделать с помощью алгоритма.