УДК 515.142.33 Дата подачи статьи: 01.08.22, после доработки: 04.08.22
DOI: 10.15827/0236-235X.139.293-304 2022. Т. 35. № 3. С. 293–304

Использование частичной параллелизации для триангуляции двумерных областей

Т.Х. Бикбулатов [1], *студент, ch7cly@gmail.com*
Д.Н. Тумаков [1], *к.ф.-м.н., доцент, dtumakov@kpfu.ru*

[1] *Казанский (Приволжский) федеральный университет, Институт вычислительной математики и информационных технологий, г. Казань, 420008, Россия*

Триангуляция произвольной области является одной из фундаментальных задач вычислительной геометрии. Классические подходы к триангуляции, например, триангуляции Делоне, дают треугольники, которые имеют достаточно широкий диапазон изменения значений их углов. Однако для ряда задач необходимо, чтобы полученные треугольники обладали более конкретными свойствами.

В работе предложен алгоритм триангуляции областей сложной геометрии с учетом заранее заданных ограничений на минимальные значения углов и максимальные длины сторон полученных треугольников. Алгоритм состоит из трех основных этапов. Первый этап принимает на вход набор точек, задающих границу фигуры, и формирует из них начальное разбиение на подобласти, генерируя точки для дальнейшей триангуляции. Такая генерация для каждой подобласти проводится совершенно независимо, что позволяет достичь максимального ускорения при распараллеливании (нагружая все логические ядра процессора с технологией hyper-threading). Затем фигура триангулируется алгоритмом «разделяй и властвуй». Здесь наибольшая производительность достигается при числе потоков, равных числу физических ядер процессора. На последнем этапе произведено уточнение параметров треугольников на основе алгоритма Рупперта. В силу специфики алгоритма последовательный код является оптимальным на этом этапе.

Все этапы параллелизации реализованы с использованием технологии OpenMP на языке C++. Показаны численные результаты, характеризующие прирост производительности вычислений для различного числа потоков в зависимости от размерности задачи.

Ключевые слова: *триангуляция Делоне, алгоритм Рупперта, параллелизация, OpenMP.*

Известно, что триангуляция – это разбиение области на треугольные подобласти. Методы триангуляции широко используются в различных областях, таких как математическое моделирование, компьютерная графика, визуализация данных, распознавание образов [1], моделирование местности [1–3], при решении алгоритмических задач на основе графов [4, 5] и т.д. Сами задачи и области, для которых выполняется триангуляция, усложняются, поэтому быстрая и эффективная триангуляция остается актуальным направлением исследований.

Существуют последовательные эффективные реализации алгоритмов для триангуляции Делоне [6–8]. С развитием вычислительных возможностей компьютеров все больший интерес вызывают параллельные алгоритмы [9, 10], причем реализуемые подходы могут быть самыми разными. Наиболее часто используется алгоритм «разделяй и властвуй» [11]. В одних работах авторы отдают предпочтение так называемому алгоритму «продвигаемого фронта» [12], в других пытаются пойти дальше и представляют варианты формирования сеток с уникальными алгоритмами декомпозиции данных и последующей композицией полной триангуляции [13] или обработки независимых областей с параллельной реализацией [14].

Сегодня основная стратегия формирования сеток заключается в том, чтобы разбить интересующую нас область на как можно более маленькие части. Это нужно делать до тех пор, пока в оставшихся частях не останутся фрагменты, которые можно будет легко обработать. Такие части можно триангулировать независимо друг от друга. Но последующий шаг предусматривает обратный ход – композицию обработанных элементов до тех пор, пока все элементы исходной области не соединятся вместе. Конечное время такого алгоритма будет зависеть в основном от способа композиции элементов. Трудозатраты на композицию сильно влияют на конечную фактическую сложность алгоритма.

Но не только эффективные алгоритмы триангуляции ускоряют работу программы. Правильное представление области также имеет

большое значение, поскольку помогает подготовить данные так, чтобы при последующей обработке алгоритм проходил меньше шагов для достижения успеха. Существует несколько способов обработки. В данной работе один из ее этапов основан на методе «отрезания ушей» [14]. Более эффективной реализации также может способствовать та или иная генерация начальных сеток (или точек). В зависимости от расстановки точек может увеличиваться или уменьшаться фактическое время работы алгоритма уточнения триангуляции всей области. При правильном подходе может быть сэкономлено достаточно много времени.

При необходимости изменить триангуляцию, удалив из нее ненужные элементы, используются алгоритмы, способные анализировать общую топологию заданной сложной геометрии, удаляя ненужные элементы, но при этом не нарушая условие Делоне [15].

В настоящей работе рассмотрена задача триангуляции двумерной односвязной области сложной геометрии, включающей как выпуклые, так и невыпуклые многоугольники. Подробно описаны и проиллюстрированы рисунками алгоритмы этапов триангуляции. Выделены три этапа: генерация набора точек, триангуляция алгоритмом «разделяй и властвуй» и уточнение параметров треугольников методом, основанным на алгоритме Руппера [16].

Вычисления и сравнение алгоритмов проведены на 18-ядерном процессоре Intel Core i9 10980XE с 64 Гб оперативной памяти. Показано ускорение производительности параллельного кода по сравнению с последовательным для двух первых этапов.

Генерация набора точек

Сгенерировать набор точек для фигуры со сложной геометрией – нетривиальная задача, так как сначала нужно определиться с тем, как обрабатывать область. От способа генерации набора точек может зависеть сложность приведения вычислительной сетки к нужным параметрам. Поэтому необходимо сгенерировать набор точек наилучшим образом, соответствующим заданным параметрам триангуляции.

Разбиение фигуры на большие треугольники. Определить, попала ли точка в многоугольник, можно различными способами. Существует классический алгоритм установки этого факта путем проведения луча из рассматриваемой точки и последующего подсчета числа пересечений с границей многоугольника, но такой подход требует дополнительных уточнений. Наиболее эффективным является разбиение фигуры на треугольники методом «отрезания ушей» [14] с дальнейшим определением принадлежности точки одному из полученных треугольников.

Зададим фигуры набором точек, последовательно расположенных против хода часовой стрелки. Применим алгоритм «отрезания ушей» и сформируем треугольники, заполняющие фигуру. В последующем, перебирая элементы этого списка и проверяя принадлежность точки хотя бы одному из треугольников, можно установить принадлежность точки всей фигуре: $v_i \in \Delta_j \in D \Rightarrow v_i \in D$, где v_i – точка из множества всех генерируемых вершин; Δ_j – треугольник из множества, покрывающего всю область; D – обрабатываемая область. Пример такого разбиения приведен на рисунке 1.

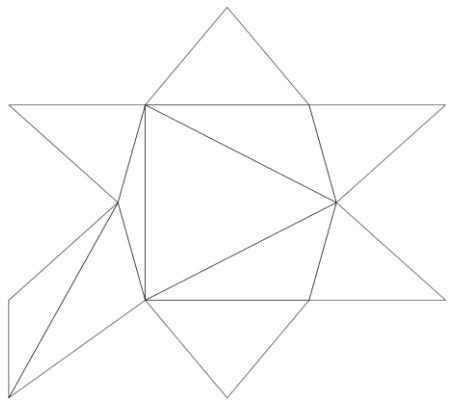


Рис. 1. Разделение области на треугольные подобласти

Fig. 1. Dividing an area into triangular subareas

Для проверки попадания точки в треугольник достаточно установить, что проверяемая точка и третья вершина лежат в одной полуплоскости относительно прямой, образованной двумя оставшимися вершинами треугольника, причем эту проверку необходимо выполнить для каждой стороны (прямой) треугольника.

Из-за дискретности графики в компьютерах все аналитические бесконечно тонкие линии перестают быть тонкими, и при изменении толщины линии изменяется точность таких проверок. Значит, от толщины линии зависит, насколько близко к стороне такого треугольника может лежать точка. С помощью списка ребер, ограничивающих обрабатываемую плоскость, при дальнейшей проверке на при-

надлежность им можно получать набор точек, попадающий на границу фигуры.

Как уже было сказано, исходный набор точек может влиять на последующую сложность генерации треугольной сетки, удовлетворяющей заданным параметрам. Рассмотрим два подхода к генерации набора точек: генерацию сетки из вершин со случайным распределением и генерацию сетки, вершины которой образуют правильные треугольники.

Генерация точек для сетки со случайным равномерным распределением. Для построения сетки используем генератор случайных чисел из библиотеки Standard template library (STL). В листинге (см. http://www.swsys.ru/uploaded/image/2022-3/2022-3-dop/22.jpg) приведена функция, генерирующая набор равномерно распределенных вершин. Для каждой нити в потоке определяется своя область для генерации точек, которая записывается в переменные `losStart` и `locEnd`. По ним с помощью генератора случайных чисел для каждой координаты вершины формируется число. Далее проверяется попадание точки в фигуру. Если вхождение подтверждается, то вершина сохраняется в общем контейнере `tFillingPoints` в критической секции, чтобы избежать конфликтов при одновременной записи информации в одну и ту же ячейку памяти. Также используется сквозной счетчик amount. Он нужен для того, чтобы плотность точек сохранялась примерно одинаковой для каждой нити. Таким способом пользователь заранее может задать для всей области сложной геометрии количество точек, необходимое для генерации.

На рисунке 2 показан график усредненного отношения времени (последовательного) выполнения одной нитью кода ко времени (параллельного) выполнения несколькими нитями, необходимого для генерации сетки из случайно равномерно распределенных вершин. Установлено, что при генерации случайной сетки максимальное ускорение достигается на 4 нитях. Вероятнее всего, это связано с тем, что дополнительное использование атомарных операций излишне замедляет весь процесс. Таким образом, дополнительные расходы, связанные с подготовкой данных для параллельных вычислений, не всегда могут быть компенсированы самими вычислениями.

Генерация точек для сетки из вершин, образующих правильные треугольники. Отличительной особенностью генерации такой сетки является выполнение нескольких дополнительных действий, связанных с триангуляцией границ области. В случае же с обычной сеткой из правильных треугольников оставляются только те точки, которые оказались внутри фигуры. Поэтому треугольники, изначально получившиеся на краях области, могут иметь слишком большие или малые углы.

Для формирования такой сетки сначала в функции `formNeighbourPoints()` найдем все соседние вершины сетки, то есть точки, расстояние до которых равно длине стороны треугольника. Эти данные нужны для дальнейшей обработки положения текущей вершины. Весь анализ проводит функция `formPointForSpecialHexMesh()`. Сначала каждая такая соседняя точка анализируется функцией `isInFigureWithBorders()`, отличие которой от `isInFigure()` состоит только в том, что при попадании точки на границу фигуры выдается ответ 2, а при попадании точки внутрь – 1 и 0 в случае, когда точка оказывается вне треугольника. Если же соседняя точка попала в фигуру, то в переменную dragp добавляются ее координаты, чтобы в дальнейшем использовать суперпозицию всех точек, попавших внутрь, для сдвига координат текущей точки.

Если все соседние точки попали внутрь фигуры, очевидно, что текущая точка также расположена внутри, и она заносится в список. В противном случае сначала строится прямая, проходящая через координаты текущей точки и точки-суперпозиции с координатами, представляющими собой арифметическое среднее

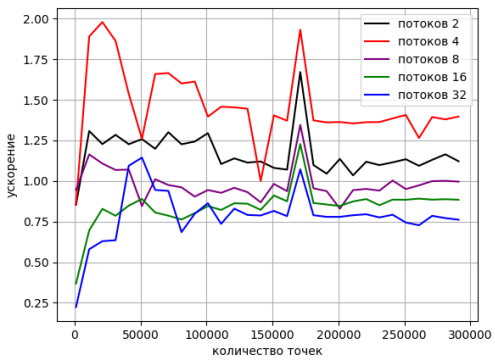


Рис. 2. График ускорения генерации набора точек из равномерно распределенных вершин

Fig. 2. An acceleration graph for generating a set of points from uniformly distributed vertices

координат соседних точек. Далее находится пересечение отрезков, образующих фигуру, с построенной прямой и определяется ближайшая точка пересечения к текущей точке. После этого координаты текущей точки изменяются на координаты ближайшей точки пересечения, тем самым текущая точка сдвигается на границу фигуры.

В случае, когда точка лежит внутри фигуры, выполняется следующее. Если расстояние между точкой пересечения и текущей точкой меньше четверти длины шага, то текущая точка отбрасывается. Если же эта длина больше половины длины шага, то точка сдвигается ближе к точке пересечения. Функция `formPointForSpecialHexMesh` приведена в листинге (см. http://www.swsys.ru/uploaded/image/2022-3/2022-3-dop/31.jpg). Она формирует набор точек, не имеет сквозного счетчика и атомарной секции, так как каждая нить обрабатывает свою область сетки.

На рисунке 3 приведен пример триангуляции фигуры с равномерной сеткой. На рисунке 4 показан график отношения времени исполнения кода на одной нити ко времени исполнения на нескольких нитях для генерации сетки из правильных треугольников. Видно, что при 32 нитях ускорение производительности достигает 10–12 раз для такого типа сетки. Еще большему ускорению мешает то, что при добавлении точки в критических областях все нити останавливают работу. Затраты также уходят на некоторые последовательные области и работу с памятью.

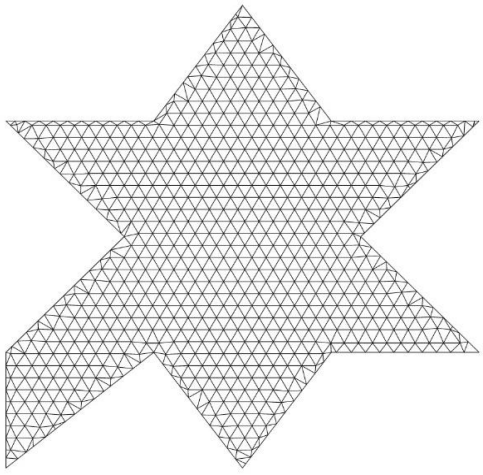


Рис. 3. Триангуляция Делоне по набору точек, составляющих правильные треугольники

Fig. 3. Delaunay triangulation on a set of points making up regular triangles

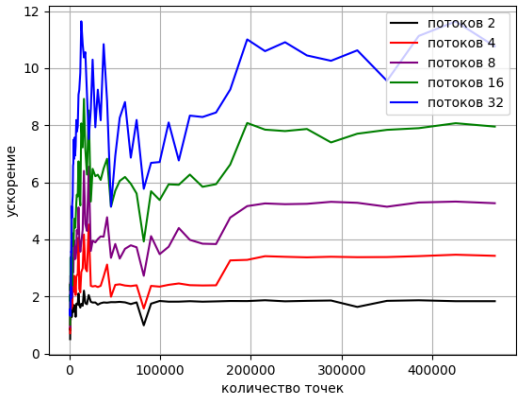


Рис. 4. График ускорения генерации набора точек из равномерно распределенных вершин

Fig. 4. An graph for accelerating the generation of a set of points from uniformly distributed vertices

Вычислительные эксперименты показали, что равномерная сетка требует меньшего числа итераций и, соответственно, времени, поэтому в дальнейшем для триангуляции будет использоваться такая сетка.

Триангуляции Делоне по набору точек

Перейдем к генерации по заданному набору точек сетки, удовлетворяющей условию Делоне. Рассмотрим два способа триангуляции: по всему набору точек и по отдельности на выпуклых подобластях.

Триангуляция по всему набору точек. Этот способ заключается в том, чтобы построить триангуляцию по заданному набору точек напрямую, а затем удалить лишние образовавшиеся ребра (ребра, изначально не принадлежащие заданной фигуре). Так как в местах, где образуются лишние ребра, мало вершин, небольшое количество ребер будет удалено.

Общий принцип алгоритма взят из [15] с использованием алгоритма сшивания подтриангуляций (триангуляций части исходной фигуры) «строй перестраивая». Рекурсивный подход разделен на две части – прямой и обратный, с использованием стека состояний вместо простого вызова рекурсивной функции. Таким способом алгоритм адаптирован к параллельной реализации.

Для более быстрого разделения областей на подобласти используем два массива, отсортированных по осям абсцисс и ординат. Чтобы поделить текущую подобласть пополам, достаточно разделить один из отсортированных массивов пополам, тогда в каждой из новых подоб-

ластей окажется примерно по половине вершин. Также для удобства отделим друг от друга операцию разделения области на подобласти и триангуляцию. Алгоритм приведен в листинге (см. http://www.swsys.ru/uploaded/image/2022-3/2022-3-dop/32.jpg).

В реализации функции триангуляции использованы стеки и очереди, которые помогают имитировать проход по дереву. Это необходимо для более эффективной работы с памятью в параллельных секциях программы.

Идея параллелизации довольно проста. Сначала необходимо разделить область на достаточное число подобластей, чтобы в дальнейшем работать уже с ними. Но не всегда в наборе точек хватает количества вершин, чтобы поровну поделить их между всеми нитями. Для этого используется переменная threads, в которую записывается оптимальное число нитей, вычисленное по количеству точек. Далее производится деление области на число подобластей, равное threads. Затем, раздавая каждой нити по своему набору вершин, запускается функция `foo`, описанная в следующем листинге:

```
void recurrentAlgorithm::foo(std::map<Rib,
SidePoints> & rc,
std::map<Point*, chain>& m, std::vector
<Point*>vx, std::vector<Point*>vy, bool ox)
{
  std::queue<mTh> climb;
  diggingDepth(climb, vx, vy, ox);
  totalmerging(climb, rc, m);
}
```

Так выполняется классический алгоритм триангуляции «разделяй и властвуй». Каждая нить занимается этим независимо, поэтому удается достичь максимального ускорения.

Как уже было замечено, сложной частью является композиция подтриангуляций. Чтобы решить эту задачу, нить получает две соседние подтриангуляции и реализует алгоритм «строй перестраивая», реализованный функцией mergeContainers. После того как все подтриангуляции были попарно сшиты, их остается в два раза меньше. Таким образом, на каждом шаге одна нить обрабатывает все большее число подтриангуляций. Это приводит к излишней нагрузке на одну нить и простаиванию остальных, что негативно влияет на общую скорость выполнения алгоритма.

На рисунке 5 показан результат работы алгоритма по набору точек для невыпуклой фигуры. Зоны, где не генерировались точки, перекрыты лишними ребрами, которые впоследствии следует удалить.

На рисунке 6 представлен график ускорения работы алгоритма в зависимости от количества потоков. Видно, что производительность увеличилась на 16 ядрах почти в 3,4 раза. 32 потока показали более низкую скорость, это объясняется тем, что на тестируемом процессоре всего 18 физических и 32 логических ядра. Таким образом, при 32 потоках используется технология hyper-threading. Однако специфика алгоритма не позволяет получить ускорение за счет использования логических ядер.

Способ первый. Удаление лишних ребер. Приведенный алгоритм триангуляции корректно работает как для выпуклых, так и для невыпуклых многоугольников. После работы алгоритма «разделяй и властвуй» всегда полу-

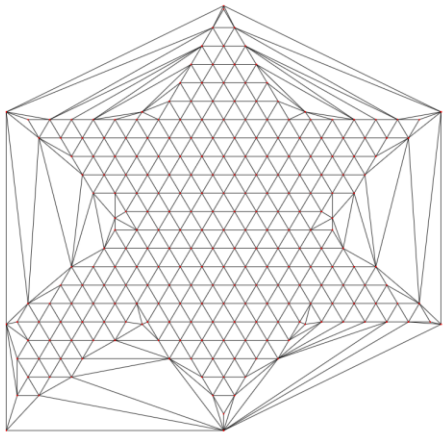


Рис. 5. Триангуляция по набору точек невыпуклой фигуры

Fig. 5. Triangulation on a set of non-convex figure points

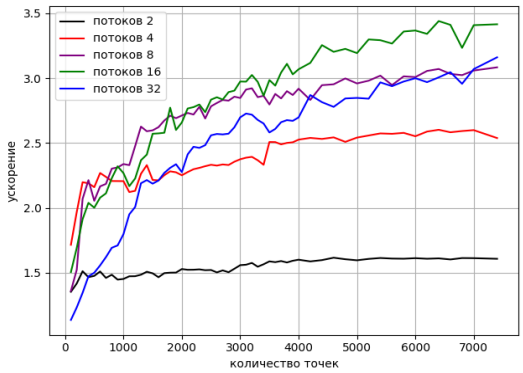


Рис. 6. График ускорения триангуляции области методом «разделяй и властвуй»

Fig. 6. A graph of the area triangulation acceleration using the divide-and-conquer method

чается выпуклая область. Из этого следует необходимость в некоторых случаях удалять лишние ребра. Продемонстрируем данную процедуру на наборах точек, которые не лежат на границе, но ее легко можно расширить на случай граничных точек.

Начнем с анализа недостающих ребер и их вставки в триангуляцию. Для этого найдем вершины ребер, которые необходимо вставить, – ребра, ограничивающие фигуру. На рисунке 7 такое ребро выделено фиолетовым цветом. Все треугольники, которые пересекаются выделенным ребром, запомним – на рисунке 7 они выделены красным цветом. Далее найдем все ребра красных треугольников, которые пересекают фиолетовое ребро, и удалим их. Затем отсортируем вершины красных треугольников, записав их в список так, чтобы они оказались по разные стороны от вставляемого ребра и образовали две фигуры.

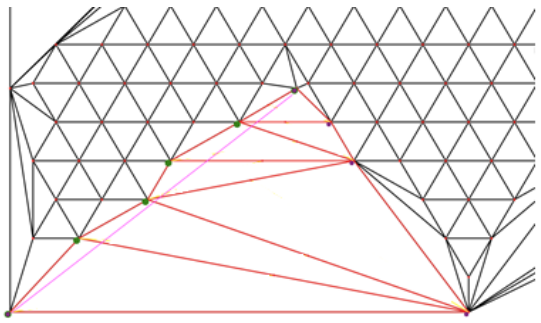


Рис. 7. Поиск ребер, мешающих вставке

Fig. 7. Searching edges that interfere with insertion

Фигуры со вставленным ребром не содержат триангуляции. Поэтому проведем триангуляцию этих двух фигур, находящихся по обе стороны от фиолетового ребра, простым алгоритмом [15]. Затем для образованных новых треугольников проверим выполнение условия Делоне и при необходимости произведем их перестроение [15].

Все вставленные ребра также запомним. После этого останется только определить лишние ребра на границе триангуляции и удалить их. Сам алгоритм приведен в листинге:

```
  void cutfigure:: cutUnneces-
saryEdge(){
     std::stack<Rib> del;

  for(auto a: _excess){
    auto it = _rc.find(a);
    if(it == _rc.end()){
      printf("Error no such Rib\n");
    }
    else{
      Rib temp;
      del.push(it->first);
      while (!del.empty()){
        temp = del.top();
        del.pop();
        auto tempit1 = _rc.find(temp);
        Point * p0 = tempit1->first.p[0];
        Point * p1 = tempit1->first.p[1];
        Point *mas[]={p1,p0};
        for(int i = 0; i<2;++i){
          temp = Rib(tempit1->first.p[i],
tempit1->second.notNULL());
          if(_figureRibs.find(temp)==
 figureRibs.end()){
            del.push(temp);
          }}
        _rc[Rib(tempit1->first.p[0],
tempit1->second.notNULL())].swap(
tempit1->first.p[1], nullptr);
        _rc[Rib(tempit1->first.p[1],
tempit1->second.notNULL())].swap(
tempit1->first.p[0], nullptr);
        _rc.erase(tempit1->first); }}}
}
```

Пример удаления ребер отражен на рисунке 8.

В функции `cutUnnecessaryEdge()` реализован алгоритм, который от внешнего лишнего ребра расходящейся волной удаляет все ребра, являющиеся ненужными. Такая рекурсивная волна идет до ребра, ограничивающего область. В конце необходимо перестроить по условию Делоне ребра, соседние с ограничивающими.

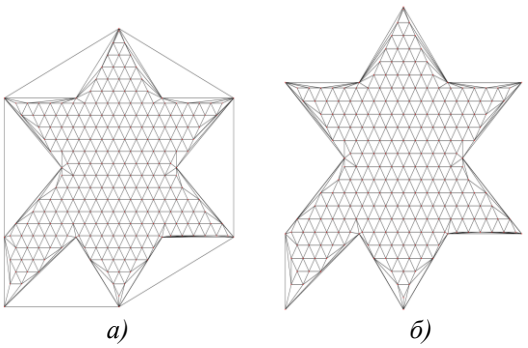


Рис. 8. Триангуляция а) после вставки всех требуемых ребер; б) после удаления всех лишних ребер

Fig. 8. Triangulation a) after inserting all required edges; б) after removing all extra edges

Способ второй. Этап 1. Деление фигуры на выпуклые части. Суть этого способа в том, чтобы изначально поделить область на выпуклые элементы и независимо распределить точки по ним, затем протриангулировать по отдельности и соединить. При большом количестве точек процесс должен выполниться быстрее, так как меньше работы с элементами три-

ангуляции, поскольку необходимо всего лишь запустить перестроение общих ребер по условию Делоне.

На вход алгоритму подается массив последовательно записанных вершин фигуры. Включим их в первый список, формально представляющий кольцо, где за последним элементом сразу идет первый (фигура является замкнутой). Этот список представляет собой текущую обрабатываемую фигуру. Для удобства формируем второй кольцевой список вершин, являющихся основанием для невыпуклых углов (больших 180°).

Пока во втором списке содержатся невыпуклые углы, будем отделять из первого списка вершины, образующие выпуклые подфигуры. При этом первый и второй списки будут постепенно сокращаться. Так, если во втором списке содержится лишь один невыпуклый угол, то сдвигаемся на $n/2$ вершин по первому списку, где n – количество вершин в первом списке. Затем, проходя в первом списке вершин в обе стороны от текущей вершины, по очереди пытаемся соединить вершины, чтобы получилась выпуклая подфигура. Эту подфигуру отсекаем, удалив лишние вершины из первого списка, и проверяем текущую фигуру на выпуклость.

Если во втором списке более одной вершины, находим в нем ближайшие вершины, являющиеся основаниями для невыпуклых углов. Соответственно, во втором списке это будут вершины с индексом, отличающимся на единицу. Соединяем эти вершины. Если хотя бы часть полученного ребра лежит вне фигуры, то ребро не принимаем и ищем другие вершины. Для этого в первом списке от непринятых вершин двигаемся поэлементно обратно к текущей вершине, пытаясь найти такую, соединение с которой даст выпуклую подфигуру. Если это удается сделать, также отсекаем ее, удаляя лишние вершины из первого списка. Если же не удается, переходим к следующей вершине во втором списке. Цикл продолжаем, пока не пройдем все невыпуклые углы.

Таким образом получим несколько выпуклых подфигур. Для них также произведем разрез на треугольники (пункт «Разбиение фигуры на треугольники») для локализации точек. Затем для каждой вершины определим ее подфигуру. Пример разбиения приведен на рисунке 9.

Алгоритм, используемый для триангуляции, такой же, как и при триангуляции по всему набору точек. С увеличением количества точек его эффективность увеличивается.

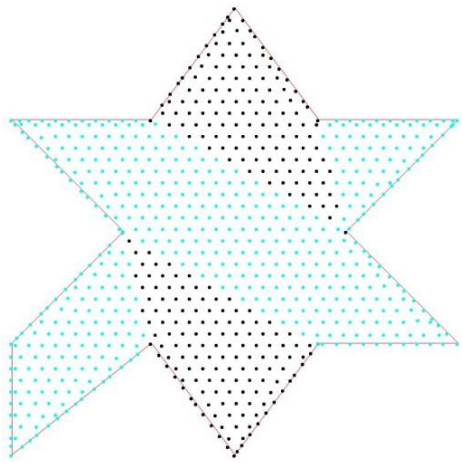
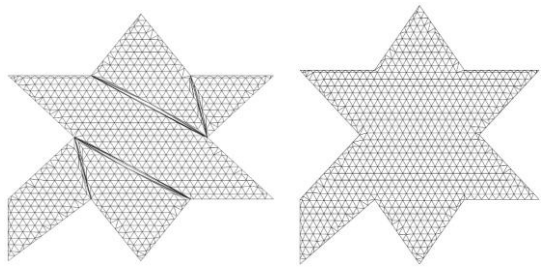


Рис. 9. Распределение точек на выпуклые подфигуры

Fig. 9. Distributing points to convex subfigures

Способ второй. Этап 2. Сшивание подобластей. Последний этап состоит в том, чтобы найти общие ребра у подфигур и объединить подтриангуляции в одну общую триангуляцию. Результат такого соединения можно увидеть на рисунке 10а. На следующем шаге отправим места сшивания на проверку и при необходимости перестроим общие ребра подфигур с учетом условия Делоне. Конечный результат этого алгоритма представлен на рисунке 10б.

В некоторых случаях такой подход может оказаться более эффективным, так как при триангуляции пропускаются самые трудозатрат-



а) *б)*

Рис. 10. Результат алгоритма сшивания подобластей: а) сшитые вместе подтриангуляции; б) перестроенная триангуляция, удовлетворяющая условию Делоне

Fig. 10. The result of the subdomain stitching algorithm is the following: a) the subtriangulations stitched together; б) a rearranged triangulation that satisfies the Delaunay condition

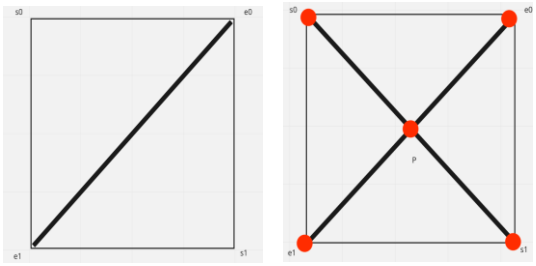
ные этапы, связанные с соединением самых больших частей через алгоритм «строй перестраивая».

Первый подход является более трудоемким, однако второй обладает неустойчивостью для областей сложной геометрии, делая разбиение на треугольники с параметрами, очень далекими от заданных. Поэтому, по мнению авторов, целесообразнее использовать первый подход без разбиения на подфигуры.

Приведение сетки к заданным параметрам

Длина каждого элемента должна получиться не больше заданной величины, а величина минимального угла – не меньше заданного значения. Для достижения нужного результата длину можно изменить за счет деления элементов пополам, а минимальный угол – за счет применения алгоритма Рупперта [16].

Последовательный алгоритм. Последовательный вариант представляет собой последовательный вызов нескольких функций. Функция `splitAllLongSegments()` делит пополам все элементы, длина которых превышает заданную, образуя 2 или 4 треугольника, как показано на рисунке 11. `splitEnroachedSegments()` делит пополам вторгшиеся ребра (те, внутри описанной окружности которых лежит любая вершина из триангуляции, кроме вершин самих ребер), также образуя 2 или 4 треугольника, а `fixingAngleByInsertAdditionalVertexes()` реализует алгоритм Рупперта.



а) *б)*

Рис. 11. Деление элементов:
а) ребро до вставки вершины в середину;
б) ребро после вставки вершины в середину

Fig. 11. Division of elements: a) an edge before inserting the vertex into the middle; б) an edge after inserting the vertex into the middle

Параллельный алгоритм. Параллельный вариант требует выполнения нескольких дополнительных этапов. Идея ускорения алгоритма заключается в том, чтобы сначала декомпозировать исходную триангуляцию, провести над ней необходимые операции, а затем объединить в единое целое.

Сначала нужно получить необходимое число подтриангуляций. Для этого разделим всю фигуру на подфигуры, а затем при необходимости будем делить эти подфигуры пополам. Пример для 32 зон (разделены красными отрезками) приведен на рисунке 12. Все ребра, которые касаются линий этих новых подфигур, зафиксируем, то есть запретим их изменять. Тогда получим 32 области, ограниченные собственными границами. Обратная компоновка будет осуществляться аналогично триангуляции по всему набору точек: попарно параллельно сшиваем подобласти. Функция, реализующая описанный алгоритм, представлена в листинге (см. http://www.swsys.ru/uploaded/image/2022-3/2022-3-dop/33.jpg).

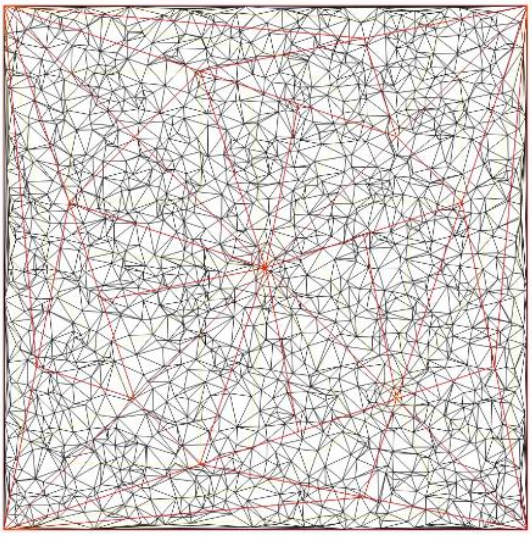
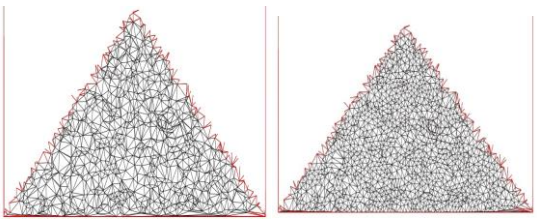


Рис. 12. Пример разделения триангуляции на подобласти

Fig. 12. An example of dividing a triangulation into subdomains

Также желательно ввести ограничение на количество допустимых вторгшихся ребер, чтобы избежать возможных зацикливаний программы. Тогда внутри можно будет делить пополам только одно вторгшееся ребро. Таким образом, допустим лишь предобработку подтриангуляции. В параллельную область вклю-

чим деление пополам длинных ребер и удаление вторгшихся ребер первого порядка. Пример такой предобработки можно увидеть на рисунке 13.



а) *б)*

Рис. 13. Подтриангуляция:
а) до предобработки; б) после предобработки

Fig. 13. Subtriangulation:
a) before preprocessing; б) after preprocessing

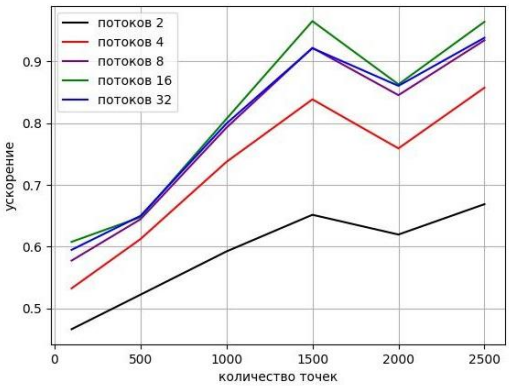
Оценить ускорение параллельного алгоритма, реализованное таким образом, достаточно сложно. Это связано с тем, что вставка точек является итеративным процессом. При этом изменение триангуляции происходит по формуле $x_{k+1} = x_k + v_k$, где x_k – триангуляция на k-м шаге, а v_k – вставляемая вершина. Соответственно, после добавления вершины в триангуляцию перестраиваем прилегающие треугольники по условию Делоне.

Рассмотрим триангуляцию с равномерно распределенными вершинами. Тогда при делении такой фигуры внутри каждой подобласти окажется примерно равное количество элементов. В этом случае при сохранении формы и увеличении плотности точек внутри элементов окажется больше, чем ограничивающих ребер, составляющих их список. Значит, предполагается, что количество элементов внутри фигуры будет расти быстрее, чем список ограничивающих ребер, и гораздо эффективнее произвести предобработку в параллельной области. Это может произойти или при большом количестве крупных элементов, или при множественном присутствии вторгшихся ребер.

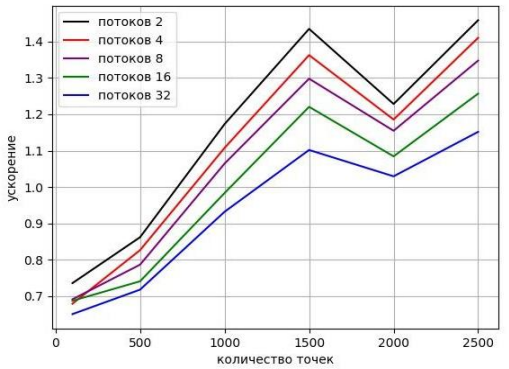
Программа может особенно заметно ускориться при разбиении длинных элементов, так как алгоритмом можно разбить почти все из них. В случае же со вторгшимися элементами можно только разделить их, потому что в противном случае программа может зациклиться из-за присутствия ребер из списков ограничивающих ребер, которые делить не разрешено. На практике ускорение может быть замечено только при больших размерностях задачи, так как накладные расходы на деление области и слияние подтриангуляций достаточно велики. Таким образом, планируется ускорять только два первых шага: удаление длинных элементов и удаление вторгшихся ребер. Оценим ускорение для этих шагов.

Чтобы оценить ускорение, сравним между собой тот блок, который производит подготовку к основному алгоритму Рупперта, то есть без функции `fixingAngleByInsertAdditionalVertexesScanningRib()`. На рисунке 14 отображены результаты сравнения производительности параллельного алгоритма.

Очевидно, что происходит замедление работы параллельного кода относительно последовательного. Снижение результатов ускорения на последних шагах можно объяснить тем, что с увеличением количества точек возрастает



а)



б)

Рис. 14. Ускорение приведения триангуляции (сторона фигуры – 1900):
а) с учетом объединения;
б) без учета объединения

Fig. 14. Acceleration of triangulation reduction (figure side – 1900):
a) the association is taken into account;
б) without taking the association into account

и их плотность, значит, меньший вклад дает параллелизованная функция `splitAllLongSegmentsP()`. При этом разделенные подобласти уточняются, не имея информации о соседних подобластях. В такой ситуации может произойти излишнее добавление вершин в одной подобласти, что в дальнейшем также потребует добавления вершин в соседних подобластях. Чем меньше дробится область, тем больше информации доступно в потоке, это приводит к снижению производительности.

Заключение

Предложен алгоритм, покрывающий сеткой односвязную область сложной геометрии. Рассмотрены обработка областей и генерация двух типов сеток: сеток со случайным набором точек, распределенных по равномерному закону, и сеток с вершинами, соответствующими правильным треугольникам. Для каждого типа сетки предложен свой параллельный алгоритм генерации вершин. Для сетки со случайным распределением вершин получено ускорение в 1,5 раза на 4 потоках, а для равномерной сетки – до 12 раз на 32 потоках.

После генерации точек предложены два алгоритма триангуляции Делоне по полученным наборам точек. Первый подход заключается в триангуляции сразу всей фигуры, второй – в триангуляции отдельных выпуклых частей исходной фигуры. Сделан вывод, что первый подход более трудозатратный, однако устойчив к сложности геометрии триангулируемого объекта. В обоих случаях триангуляция осуществлена алгоритмом «разделяй и властвуй». Наибольшая производительность достигнута при числе потоков, равных числу физических ядер процессора, ускорение составило 3,5 раза.

На последнем этапе проведено уточнение параметров треугольников на основе алгоритма Рупперта. Полученные результаты свидетельствуют о том, что последовательный код работает быстрее параллельного. Прежде всего это происходит из-за трудоемкого пересчета при сшивании подобластей.

Работа выполнена за счет средств Программы стратегического академического лидерства Казанского (Приволжского) федерального университета («ПРИОРИТЕТ-2030»).

Литература

1. Kugunavar S., Prabhakar C.J. Content-based medical image retrieval using Delaunay triangulation segmentation technique. J. of Information Technology Research, 2021, vol. 14, no. 2, pp. 48–66. DOI: 10.4018/JITR.2021040103.
2. Giniyatova D.K., Tumakov D.N., Markina A.G. Solving the problem of electromagnetic wave diffraction by a flat screen using CUDA. Lobachevskii J. of Mathematics, 2021, vol. 42, no. 6, pp. 1335–1344. DOI: 10.1134/S1995080221060081.
3. Tang L., Ren C., Liu Z., Li Q. A road map refinement method using Delaunay triangulation for big trace data. ISPRS Int. J. Geo-Inf., 2017, vol. 6, no. 2, art. 45. DOI: 10.3390/ijgi6020045.
4. Liu Z., Liu H., Lu Z., Zeng Q. A dynamic fusion pathfinding algorithm using Delaunay triangulation and improved A-star for mobile robots. IEEE Access, 2021, vol. 9, pp. 20602–20621. DOI: 10.1109/ACCESS.2021.3055231.
5. Agayan G.M., Balabaev N.K., Rodnikova M.N. Description of mixed networks of H-bonds in a water-ethylene glycol system by methods of graph theory and Delaunay simplices. Russian J. of Physical Chemistry A, 2021, vol. 95, no. 7, pp. 1283–1290. DOI: 10.1134/S0036024421070025.
6. Su P., Drysdale R.L.S. A comparison of sequential Delaunay triangulation algorithms. Computational Geometry, 1997, vol. 7, no. 5-6, pp. 361–385. DOI: 10.1016/S0925-7721(96)00025-9.
7. Shewchuk J. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. In: Computational Geometry Towards Geometric Engineering. Proc. WACG, 1996, vol. 1148, pp. 203–222. DOI: 10.1007/BFb0014497.
8. CGAL 5.4.1 – dD Convex Hulls and Delaunay Triangulations. URL: https://doc.cgal.org/latest/Convex_hull_d/index.html (дата обращения: 10.07.2022).
9. Batista V.H.F., Millman D.L., Pion S., Singler J. Parallel geometric algorithms for multi-core computers. Computational Geometry, vol. 43, no. 8, pp. 663–677. DOI: 10.1016/j.comgeo.2010.04.008.
10. Blelloch E.G., Miller L.G., Hardwick C.J., Talmor D. Design and implementation of a practical parallel Delaunay algorithm. Algorithmica, 1999, vol. 24, pp. 243–269. DOI: 10.1007/PL00008262.
11. Yang H., Liu L., Zhang C., Li R. et al. PatCC1: an efficient parallel triangulation algorithm for spherical and planar grids with commonality and parallel consistency. Geoscientific Model Development, 2019, vol. 12, pp. 3311–3328. DOI: 10.5194/gmd-12-3311-2019.

12. Nguyen C., Rhodes P.J. TIPP: parallel Delaunay triangulation for large-scale datasets. Proc. XXX Int. Conf. SSDM, 2018, no. 8, pp. 1–12. DOI: 10.1145/3221269.3223034.

13. Yu F., Zeng Y., Guan Z.Q., Lo S.H. A robust Delaunay-AFT based parallel method for the generation of large-scale fully constrained meshes. Computers & Structures, 2020, vol. 228, art. 106170. DOI: 10.1016/j.compstruc.2019.106170.

14. Mei G., Tipper J.C., Xu N. Ear-clipping based algorithms of generating high-quality polygon triangulation. In: Lecture Notes in Electrical Engineering, 2012, vol. 212, pp. 979–988. DOI: 10.1007/978-3-642-34531-9_105.

15. Скворцов А.В. Триангуляция Делоне и ее применение. Томск: Изд-во Том. ун-та, 2002. 128 с.

16. Ruppert J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. J. of Algorithms, 1995, vol. 18, no. 3, pp. 548–585. DOI: 10.1006/jagm.1995.1021.

Software & Systems
DOI: 10.15827/0236-235X.139.293-304

Received 01.08.22, Revised 04.08.22
2022, vol. 35, no. 3, pp. 293–304

Using partial parallelization to triangulate 2D domains

T.Kh. Bikbulatov [1], *Undergraduate Student, ch7cly@gmail.com*
D.N. Tumakov [1], *Ph.D. (Mathematics and Physics), Associate Professor, dtumakov@kpfu.ru*

[1] *Kazan Federal University, Institute of Computational Mathematics and Information Technologies, Kazan, 420008, Russian Federation*

Abstract. Delaunay triangulation of an arbitrary domain is one of the fundamental problems of computational geometry. Classical approaches to Delaunay triangulation produce triangles that have a wide range of angle values.

The paper proposes an algorithm for triangulation of complex geometry domains taking into account predetermined parameters: the minimum angle and the maximum side length of the obtained triangles. The algorithm consists of three main stages. The first stage takes a set of points that set the figure boundary as input, and forms an initial partition into subdomains from them generating points for further triangulation. Such generation for subdomains is completely independent; therefore, it is most effectively parallelized by the number of logical cores. The figure is then triangulated by the divide-and-conquer algorithm. Here, the highest performance is achieved with the number of threads equal to the number of physical processor cores. At the last stage, the parameters of triangles are refined by a method based on Ruppert's algorithm. Due to the specifics of the algorithm, the serial code is optimal at this stage.

All parallelization is implemented using OpenMP technology in C++. The paper shows numerical results representing the increase in computing performance for a different number of threads depending on the problem dimension.

Keywords: Delaunay triangulation, Ruppert's algorithm, parallelization, OpenMP.

Acknowledgements. *This paper has been supported by the Kazan Federal University Strategic Academic Leadership Program ("PRIORITY-2030").*

References

1. Kugunavar S., Prabhakar C.J. Content-based medical image retrieval using Delaunay triangulation segmentation technique. *J. of Information Technology Research*, 2021, vol. 14, no. 2, pp. 48–66. DOI: 10.4018/JITR.2021040103.

2. Giniyatova D.K., Tumakov D.N., Markina A.G. Solving the problem of electromagnetic wave diffraction by a flat screen using CUDA. *Lobachevskii J. of Mathematics*, 2021, vol. 42, no. 6, pp. 1335–1344. DOI: 10.1134/S1995080221060081.

3. Tang L., Ren C., Liu Z., Li Q. A road map refinement method using Delaunay triangulation for big trace data. *ISPRS Int. J. Geo-Inf.*, 2017, vol. 6, no. 2, art. 45. DOI: 10.3390/ijgi6020045.

4. Liu Z., Liu H., Lu Z., Zeng Q. A dynamic fusion pathfinding algorithm using Delaunay triangulation and improved A-star for mobile robots. *IEEE Access*, 2021, vol. 9, pp. 20602–20621. DOI: 10.1109/ACCESS.2021.3055231.

5. Agayan G.M., Balabaev N.K., Rodnikova M.N. Description of mixed networks of H-bonds in a water-ethylene glycol system by methods of graph theory and Delaunay simplices. *Russian J. of Physical Chemistry A*, 2021, vol. 95, no. 7, pp. 1283–1290. DOI: 10.1134/S0036024421070025.

6. Su P., Drysdale R.L.S. A comparison of sequential Delaunay triangulation algorithms. *Computational Geometry*, 1997, vol. 7, no. 5-6, pp. 361–385. DOI: 10.1016/S0925-7721(96)00025-9.

7. Shewchuk J. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. In: *Computational Geometry Towards Geometric Engineering. Proc. WACG*, 1996, vol. 1148, pp. 203–222. DOI: 10.1007/BFb0014497.

8. *CGAL 5.4.1 – dD Convex Hulls and Delaunay Triangulations*. Available at: https://doc.cgal.org/latest/Convex_hull_d/index.html (accessed July 10, 2022).

9. Batista V.H.F., Millman D.L., Pion S., Singler J. Parallel geometric algorithms for multi-core computers. *Computational Geometry*, vol. 43, no. 8, pp. 663–677. DOI: 10.1016/j.comgeo.2010.04.008.

10. Blelloch E.G., Miller L.G., Hardwick C.J., Talmor D. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica*, 1999, vol. 24, pp. 243–269. DOI: 10.1007/PL00008262.

11. Yang H., Liu L., Zhang C., Li R. et al. PatCC1: an efficient parallel triangulation algorithm for spherical and planar grids with commonality and parallel consistency. *Geoscientific Model Development*, 2019, vol. 12, pp. 3311–3328. DOI: 10.5194/gmd-12-3311-2019.

12. Nguyen C., Rhodes P.J. TIPP: parallel Delaunay triangulation for large-scale datasets. *Proc. XXX Int. Conf. SSDM*, 2018, no. 8, pp. 1–12. DOI: 10.1145/3221269.3223034.

13. Yu F., Zeng Y., Guan Z.Q., Lo S.H. A robust Delaunay-AFT based parallel method for the generation of large-scale fully constrained meshes. *Computers & Structures*, 2020, vol. 228, art. 106170. DOI: 10.1016/j.compstruc.2019.106170.

14. Mei G., Tipper J.C., Xu N. Ear-clipping based algorithms of generating high-quality polygon triangulation. In: *Lecture Notes in Electrical Engineering*, 2012, vol. 212, pp. 979–988. DOI: 10.1007/978-3-642-34531-9_105.

15. Skvortsov A.V. *Delaunay Triangulation and its Application*. Tomsk, 2002, 128 p. (in Russ.).

16. Ruppert J. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. of Algorithms*, 1995, vol. 18, no. 3, pp. 548–585. DOI: 10.1006/jagm.1995.1021.

Для цитирования

Бикбулатов Т.Х., Тумаков Д.Н. Использование частичной параллелизации для триангуляции двумерных областей // Программные продукты и системы. 2022. Т. 35. № 3. С. 293–304. DOI: 10.15827/0236-235X.139.293-304.

For citation

Bikbulatov T.Kh., Tumakov D.N. Using partial parallelization to triangulate 2D domains. *Software & Systems*, 2022, vol. 35, no. 3, pp. 293–304 (in Russ.). DOI: 10.15827/0236-235X.139.293-304.