ПРЕДИСЛОВИЕ НАУЧНОГО РЕДАКТОРА ПЯТИТОМНОГО ЦИКЛА МОНОГРАФИЙ

Данный том пятитомного цикла монографий для специалиста в области прочностных расчетов стоит несколько особняком. Исследователь или инженер-прочнист обычно не задумывается над алгоритмами, а тем более над сложнейшим математическим аппаратом, на основе которого разработаны алгоритмы построения (генерации) сеток для численного решения задач прочности. Вероятно, это связано с тем, что если для расчетов используется универсальная или нишевая CAE (Computer Aided Engineering — система инженерного анлиза), то она содержит промышленный генератор сеток. Если решение частной сложной исследовательской задачи требует разработки самостоятельной программы, то используется уже имеющийся генератор сеток, так как для решения частной задачи практически всегда этого достаточно. При этом следует понимать, что такая программа обычно работоспособна только в руках исследователя.

Необходимость более глубокого погружения в проблему возникает при разработке полнофункциональной CAE, например, для прочностного инженерного анализа, способной решать сложные задачи при конечных деформациях и их перераспределении. Такие задачи описывают механические процессы, с конечными деформациями при нагружении, в которых происходит изменение границ и граничных условий. К таким процессам относятся, например: образование в процессе нагружения концентраторов напряжений; изменение свойств материала части тела; принудительное, с точки зрения механики деформируемого твердого тела, наращивание (изменение) массы тела. Такие задачи решаются на основе теории многократного наложения больших деформаций (т. I–III).

На необходимость создания и развития данной теории [1)] неоднократно указывал академик Л.И. Седов, отмечая более трети века назад ее практическую значимость в будущем. Заметим, что в наше время с развитием аддитивных технологий, созданием новых материалов, с усилившейся потребностью более точно оценивать закритические сценарии нагружения и ресурс изделия промышленная реализация подобных задач стала практической необходимостью.

[1)] *Левин В. А.* Многократное наложение больших деформаций в упругих и вязкоупругих телах / С предисл. акад. Л. И. Седова. — М.: Наука, Физматлит, 1999. — 223 с.

В этом случае от генератора сеток требуется очень высокая функциональность. Это не только базовая функциональность — возможность построения структурированных и некоторых типов неструктурированных сеток, но и, в частности, автоматическая адаптация сетки к изменяющейся геометрии (топологии) расчетной области и к полям напряжений/деформаций в процессе решения, включая анизотропные сетки, позволяющие учитывать наведенную анизотропию в окрестностях концентраторов напряжений. Это важно, например, в динамических задачах, в которых нужно обеспечивать высокую точность решения на фронте волны.

Для реализации указанной функциональности необходимы, в частности: решение проблемы перестроения сеток вблизи криволинейных границ тела; управление анизотропной адаптацией сеток посредством их локальных модификаций; восстановление геометрии тела по деформированной сетке; автоматическое перестроение сеток для улучшения сходимости итерационных алгоритмов решения существенно нелинейных задач при больших деформациях и искажениях исходной геометрии тела; автоматическое перестроение (улучшение свойств) конечноэлементной сетки на основе исходной, содержащей сильно вытянутые элементы и/или элементы с «излишне острыми» углами. Описанию решения подобных проблем и посвящен данный том.

Большинство исследований в данной области в силу специализации, возникшей в современной науке, к сожалению, заканчиваются на этапе исследовательского кода и редко получают промышленную реализацию. Авторы тома смогли довести их до отчуждаемого программного продукта, доказавшего свою полезность в исследовательских задачах и способного дополнять функциональность промышленных генераторов сеток [1]. Именно на доведение научного результата до практического использования всегда нацеливал своих учеников и сотрудников академик Г.И. Марчук.

Следует отметить, что за четыре года, прошедших от согласования структуры пятитомного цикла до его выхода, в технологиях решения задач прочностного инженерного анализа наметилась серьезная тенденция использования методов изогеометрического анализа [29, 33, 44, 58, 84]. Суть подхода заключается в использовании в качестве базисных функций линейного пространства, в рамках которого отыскивается решение исходной краевой задачи в частных производных, например, методом взвешенных невязок Галёркина, неоднородных рациональных B-сплайнов (NURBS), задающих одновременно геометрию рассматриваемого тела (CAD-модель; на практике это совместное использование CAD и CAE). В этом случае отпадает необходимость введения дополнительного базиса, состоящего из функций форм, заданных на конечноэлементной дискретизации исходной модели

[1]) Такая интеграция выполнена в CAE FIDESYS (www.cae-fidesys.com).

и, как следствие, не требуется построение, зачастую нетривиальное, неструктурированных сеток.

Стоит отметить, что на текущий момент разработаны эффективные методики геометрического описания границ трехмерных тел (криволинейных поверхностей) с использованием NURBS, и, как следствие, изогеометрический анализ успешно используется, например, при решении задач для оболочечных конструкций [1]. Однако использование данного подхода в промышленных задачах прочностного инженерного анализа в общем случае невозможно из-за нерешенной проблемы построения базиса из трехмерных NURBS для областей произвольной формы [32]. Когда (если) эта проблема будет преодолена, вероятно, значительная часть тривиальных стандартных прочностных расчетов будет проводиться с использованием изогеометрического анализа. Сложные задачи, связанные с анализом поведения дефекта, развития зон предразрушения, изменения границ и граничных условий, массы тела будут по-прежнему требовать использования генераторов сеток. И, вероятно, в промышленных пакетах будет реализован симбиоз этих или подобных подходов. Можно предположить возникновение расчетного блока с условным названием «генератор дискретизации», сочетающего эти подходы [2]).

Методически похожий процесс наблюдается при введении в промышленную расчетную практику метода спектральных элементов вместо метода конечных элементов (см. т. I, II) [3]).

В контексте рассматриваемых современных методов численной дискретизации геометрии тела или группы тел следует упомянуть метод внешних аппроксимаций [4]). Данный метод является своего рода логичным обобщением метода конечных элементов, расширяя и дополняя основные его идеи, включая слабую формулировку и поиск решения в подпространствах Соболева. Не вдаваясь в детали, отметим, что в рамках метода внешних аппроксимаций вводится понятие обобщенного конечного элемента, являющегося в задачах прочности сборок и сложных составных конструкций фактически их составной частью — по сути, каждая деталь сборки может описываться одним обобщенным

[1]) *Cottrell J. A., Hughes T. J. R., Bazilevs Y.* Isogeometric Analysis: Toward Integration of CAD and FEA. — N.Y.: Wiley & Sons, 2009.

[2]) Поэтому научный редактор данного цикла монографий предложил подготовить вторую часть т. IV (как отдельный том) по использованию изогеометрического анализа при численном решении задач механики деформируемого твердого тела.

[3]) Описанное сочетание методов численной дискретизации реализовано в промышленном пакете прочностного инженерного анализа CAE FIDESYS (www.cae-fidesys.com).

[4]) *Апанович В. Н.* Метод внешних конечноэлементных аппроксимаций. — Минск: Вышэйш. шк., 1991. — 170 с.; *Aubin J.-P.* Approximation of elliptic boundary value problems. — N.Y.: Wiley-Interscience, 1972. См. также [2].

конечным элементом произвольной геометрической формы. Другим отличием от классического конечного элемента с изопараметрической дискретизацией геометрии и полей неизвестных функций внутри элемента является возможность задания произвольного набора базисных функций, никак не связанных с геометрией обобщенного элемента; единственное требование к выбору базиса — полнота в соответствующем подпространстве Соболева. Нерешенной проблемой с точки зрения массового промышленного использования является недостаточно точное описание сложных границ. Отметим, что с практической точки зрения для пользователя метод внешних аппроксимаций подобен изогеометрическому анализу, позволяя проводить расчет напрямую на исходной геометрии тела без каких-либо ее модификаций и дополнительных действий по подготовке модели к расчету.

Учитывая всё сказанное, редактор цикла ожидает создание в течение ближайшего десятилетия в промышленных пакетах прочностного инженерного анализа неких комбинаций «генераторов дискретизации» и решателей, объединяющих имеющиеся и возникающие методы решения задач прочности.

В заключение хотелось бы отметить глубокое погружение авторов тома в проблематику задач прочности при конечных деформациях и их перераспределении, практическое участие в их решении [1]), внимательную проработку примеров, приведенных в книге и предложенных редактором цикла [2]).

Данный том будет интересен как пользователям, так и разработчикам пакетов для прочностного инженерного анализа, а также полезен студентам старших курсов и аспирантам профильных специальностей.

Москва, МГУ им. М. В. Ломоносова Профессор *В. А. Левин*
11 июня 2016 г.

[1]) В частности, автоматическая адаптация сетки в задачах о распространении упругой волны при конечных деформациях по повреждаемому материалу (т. I) выполнена А.А. Даниловым.

[2]) Именно поэтому автор предисловия счел для себя правильным не входить в состав авторского коллектива, а ограничиться подбором примеров и обсуждением структуры тома.

АВТОРСКОЕ ПРЕДИСЛОВИЕ

Мы начали работать над проблемой построения квазиоптимальных сеток для численного решения уравнений в частных производных более 15 лет назад. Изначально нам было очевидно, что такие сетки должны быть неструктурированными и могут содержать треугольники или тетраэдры, сильно вытянутые в каком-нибудь направлении. Анализ свойств численных решений на таких сетках потребовал разработки новых алгоритмов и специального программного обеспечения. Целью книги является детальное обсуждение разнообразного практического инструментария, востребованного в инженерных приложениях и необходимого для эффективного автоматизированного построения неструктурированных адаптивных расчетных сеток, как регулярных, так и анизотропных.

Одно из наших практических наблюдений состоит в том, что быстрое и надежное построение расчетных сеток требует комбинирования трех различных методов. Метод продвигаемого фронта достаточно быстро строит сетку в 95–99 % вычислительной области. Метод триангуаляции Делоне требует бóльших вычислительных затрат на построение каждого элемента сетки, но позволяет достроить сетку в оставшихся проблемных подобластях. Наконец, форма элементов сетки в проблемных подобластях улучшается с использованием метода перестроения анизотропных сеток, имеющего самую высокую вычислительную сложность. Алгоритмы, составляющие основу нашей технологии надежного построения сетки, представлены в главах 3 и 5. Оптимизация вычислительных затрат в динамических задачах предполагает, что начальная сетка меняется во времени в зависимости от динамики решения. В главе 4 представлены методы построения адативных динамических сеток посредством их многоуровневого иерархического измельчения и огрубления. В остальных главах и приложении мы обсуждаем различные методы контроля локального размера и формы элементов сетки, использующие скалярные и тензорные метрики, а также различные оценки численной ошибки.

Книга предназначена для разработчиков систем инженерного анализа, инженеров и математиков-вычислителей, деятельность которых связана с построением расчетных сеток.

Наши коллеги оказали нам значительную помощь при написании книги. А. Агузал сделал определяющий вклад в теоретический анализ свойств численных решений на анизотропных сетках. Разработанные совместно с ним математические методы легли в основу ряда алгоритмов, реализованных в нашем открытом программном обеспечении,

пакетах Ani2D и Ani3D. Мы благодарны А.В. Вершинину и А.В. Плёнкину, внесшим значительный вклад в разработку метода улучшения заданной поверхностной сетки. Мы благодарим В.А. Гаранжу, Р.В. Гаримеллу, В.М. Головизнина, Н. Деби, В.Г. Дядечко, П.Л. Джорджа, И.В. Капырина, Т.К. Козубскую, В.Д. Лисейкина, Д.Р. Мадиссона, Ю.М. Нечепуренко, К.Д. Никитина, И.Б. Петрова, А.В. Плёнкина, И.А. Сазонова, П.Э. Фаррелла, Ф. Хеша и А.Ю. Чернышенко за обсуждения алгоритмов построения расчетных сеток и замечания, позволившие нам улучшить книгу. За неоценимый вклад в тестирование нашего программного обеспечения мы признательны студентам МФТИ и МГУ, а также многим пользователям во всем мире, приславшим свои замечания.

Работа над книгой заняла несколько лет и много личного времени. Мы глубоко признательны нашим семьям за поддержку в это время.

Москва, Россия и Лос-Аламос, США — *Ю.В. Василевский*
3 декабря 2014 г. — *А.А. Данилов*
К.Н. Липников
В.Н. Чугунов

Глава 1

ВВЕДЕНИЕ

Настоящая книга — четвертый том цикла «Нелинейная вычислительная механика прочности». Основной целью этого цикла является изложение различных математических моделей, численных методов и алгоритмов, составляющих основу систем инженерного анализа (СИА) и позволяющих описывать поведение деформируемого твердого тела при конечных деформациях, когда в процессе его нагружения, дискретно или непрерывно, изменяются границы и граничные условия. Важным приложением СИА являются задачи прочности, решение которых связано с численным моделированием развития дефекта в твердом теле с учетом возникновения в вершине дефекта зон предразрушения и фазовых переходов, происходящих под действием механических напряжений. Численное решение подобного рода задач требует богатого инструментария для построения расчетных сеток. Описанию различных технологий построения сеток и посвящен четвертый том цикла.

Основная цель книги заключается в изложении ряда современных алгоритмов построения расчетных сеток для инженерных приложений, в частности, связанных с задачами прочности. Мы ограничимся рассмотрением только неструктурированных сеток, которые могут быть построены практически в любой расчетной области, заданной как на плоскости, так и в пространстве. Технологии построения структурированных сеток (т. е. сеток с ijk-структурой связности между узлами), основанные на построении отображения некоторой канонической области в заданную расчетную область, являются менее автоматизированными, чем технологии построения неструктурированных сеток. Методам и алгоритмам построения структурированных сеток посвящен ряд монографий и научных сборников, изданных как в нашей стране, так и за рубежом [1, 10, 14–16, 22, 56, 59, 61, 68, 69]; к этим источникам мы отсылаем заинтересованного читателя.

Эта книга представляет собой обобщение многолетнего опыта авторов в деле практического построения неструктурированных расчетных сеток. Основной целью книги является обсуждение разнообразного инструментария, используемого при построении таких сеток. Область применения излагаемого инструментария нам видится прежде всего в разработке и внедрении новых сеточных генераторов в инженерных приложениях. Целевую аудиторию книги составляют разработчики СИА, инженеры и математики-вычислители, деятельность которых связана с построением расчетных сеток, т. е. те, кто непосредственно программирует или использует сеточные генераторы.

Книга ни в коей мере не претендует на исчерпывающий обзор и теоретический анализ всех методов построения неструктурированных сеток и используемых для этого структур данных. Изучить эти вопросы читатель может по монографиям [8, 11, 18, 36, 50, 52, 54] и трудам специализированных отечественных (NUMGRID-2004, 2006, 2008, 2010, 2012) и международных (International Meshing RoundTable, 1992–2014) конференций (см., например, [76, 77]). Отдельного упоминания заслуживает еще не опубликованная монография В.А. Гаранжи «Численная геометрия и построение расчетных сеток», содержащая богатый теоретический материал по рассматриваемой тематике. Каркас нашей книги сформирован набором алгоритмов, реализованных нами в пакетах программ ani2D и ani3D, находящихся в свободном доступе в сети Интернет: www.sf.net/projects/ani2d; www.sf.net/projects/ani3d. Содержание представляет собой обсуждение представляемых алгоритмов, включающее практические результаты их использования.

Настоящая книга обладает рядом особенностей, отличающих ее от уже изданных монографий. Во-первых, она представляет *все этапы* технологической цепочки построения расчетной сетки, начиная от задания области и кончая процедурами улучшения и адаптации сетки. В известных авторам монографиях [18, 36, 50, 54] рассматриваются один или несколько этапов технологической цепочки. Во-вторых, мы рассматриваем *автоматизированные технологии* надежной генерации сеток, обеспечивающие безотказную работу алгоритмов. Данное качество чрезвычайно важно при построении неструктурированных трехмерных сеток в сложных областях. Надежность построения трехмерной сетки достигается комбинацией нескольких методов, поскольку ни один индивидуальный метод сам по себе не может обеспечить построение сетки приемлемого качества. В-третьих, мы даем *детальное описание* используемых алгоритмов, что позволяет понять особенности технической реализации алгоритмов в наших программах. Алгоритмический взгляд на методы построения расчетных сеток характерен также для книги [18]. Перечисленные выше особенности книги и программные реализации изложенных алгоритмов делают ее удобным пособием не только для пользователей нашего программного обеспечения, но и для разработчиков новых библиотек генерации сеток.

Большинство геометрических моделей, рассматриваемых в книге, являются характерными примерами в задачах прочности: двумерное или трехмерное тело с отверстиями или трещинами. При наложении значительной механической нагрузки на такие тела очень важно расчитывать внутренние напряжения и предсказывать зоны предразрушения, что, по-видимому, невозможно без использования адаптивных сеток. Например, в случае распространения трещины необходимо динамически перестраивать сетку, обеспечивая ее сгущение к текущему положению края трещины. Для этого нужно: во-первых, построить начальную сетку, адаптивно сгущающуюся к краю трещины в начальный момент

времени; во-вторых, перестраивать сетку на каждом временно́м шаге, сгущая ее к новому расчетному положению края трещины и огрубляя ее в других зонах. В случае расчета зон предразрушения в телах с полостями нужно адаптировать сетку к вершинам дефектов и отслеживать границы зон предразрушения и, возможно, перестраивать сетку в зависимости от реализации фазовых переходов. Мы не представляем результаты расчетов задач деформации твердого тела, но представляем сетки, которые могут быть использованы для решения таких задач, а также возможные технологии их построения.

Перечислим кратко основные достоинства разработанных нами технологий построения и адаптации неструктурированных расчетных сеток. Понятия, которые мы используем во введении, будут строго определены в главе 2 книги.

В книге предложен набор алгоритмов и программный инструментарий для надежного построения двумерных и трехмерных сеток с регулярными ячейками в областях сложной формы, которые могут быть заданы различными способами. Например, трехмерная область может быть задана набором операций объединения, вычитания и пересечения геометрических примитивов, таких, как эллипсоид, параллелепипед, конус и т. п. Граница области также может быть задана как набор параметризованных кусков поверхностей, границы которых также параметризованы. Наиболее общий подход задания области — использование системы автоматизации проектных работ (САПР) [1)], являющейся составной частью многих СИА. Взаимодействие сеточного генератора с САПР, хранящей внутреннее параметрическое представление границы области, обеспечивается специальным программным интерфейсом.

Построенные неструктурированные сетки могут быть измельчены с помощью методов локального измельчения сетки. При этом выбранные пользователем треугольные или тетраэдральные ячейки разбиваются на две подъячейки, так же как и некоторые соседние ячейки, этим гарантируется конформность результирующей сетки. Будучи примененной несколько раз, эта процедура обеспечивает многоуровневое локальное измельчение, в ходе которого качество новых ячеек может ухудшиться лишь незначительно. Более того, измельченные сетки допускают многоуровневое локальное огрубление согласно правилам, установленным пользователем, благодаря чему обеспечивается построение динамически адаптируемых сеток, отслеживающих движущиеся особенности сеточного решения.

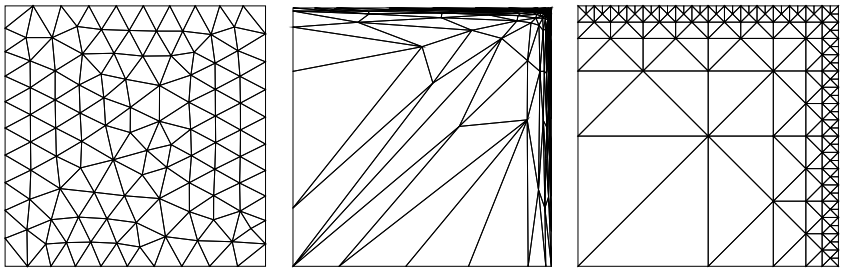
Помимо иерархического измельчения и огрубления, неструктурированные сетки можно перестраивать полностью через последовательность локальных модификаций сетки. Такой подход обладает наибольшими возможностями по сравнению с иерархическим измельчением

[1)] В англоязычной литературе CAD (Computer Aided Design).

сетки, поскольку позволяет строить анизотропные сетки. Анизотропная адаптация уменьшает или увеличивает размеры ячеек в выделенных направлениях, что является разумным для решений с анизотропными особенностями. Набор локальных операций весьма ограничен. В двумерном случае достаточно реализовать всего пять базовых операций: вставка узла на ребро, замена ребра, сдвиг узла, удаление ребра и удаление узла. В трехмерном случае число базовых операций — семь: вставка узла на ребро, прямая и обратная замены грани на ребро, замена ребра на полигональную грань, сдвиг узла, удаление ребра и удаление узла.

Те же самые базовые операции применяются для исправления и распутывания сеток. Такие сетки возникают, например, при решении задач деформации с использованием динамических сеток, которые движутся вместе с деформируемым телом.

Примеры неадаптированной неструктурированной сетки, анизотропной адаптированной сетки и иерархической локально измельченной сетки приведены на рис. 1.1. Выбор между локально измельченными и полностью неструктурированными сетками зависит от задачи. В ряде задач упругости и гидродинамики, особенно в задачах с сильными особенностями, анизотропная неструктурированная сетка позволяет достичь приемлемой точности сеточного решения на значительно меньшем количестве ячеек, чем регулярная сетка. С другой стороны, построение анизотропной сетки требует бóльших вычислительных затрат.



а *б* *в*

Рис. 1.1. Примеры расчетных сеток: *а* — неадаптированная неструктурированная, *б* — анизотропно адаптированная, *в* — иерархическая локально измельченная

Автоматизированные алгоритмы адаптивного перестроения расчетных сеток требуют управления свойствами этих сеток. Управлять локальным иерархическим перестроением сетки можно с помощью индикаторов апостериорной ошибки, примеры которых мы приводим. Для построения неструктурированных сеток нужно определять непрерывные функции, характеризующие локальные свойства ячеек, например, их желаемый размер в каждой точке области. Апостериорные оценки ошибки лежат в основе построения таких функций. Функции, управляющие изотропным измельчением или огрублением сетки, являются скалярными, а функции, управляющие анизотропной адаптацией, являются тензорными, поскольку должны задавать как размер ячейки, так и направление ее вытянутости. Правильное задание управляющей функции является важным условием эффективности расчетной сетки. Мы предлагаем новые методы построения этих функций и показываем их эффективность как теоретически [24, 26], так и практически.

Наконец, еще одно достоинство нашего подхода заключается в том, что перечисленные выше технологии построения расчетных сеток реализованы в свободно распространяемых пакетах программ, поэтому могут быть протестированы любым пользователем.

Перейдем к краткому описанию содержания книги. Глава 2 является вводной для читателя, не обладающего специальными знаниями. В § 2.1 мы даем основные определения неструктурированных сеток, рассматриваемых в книге, а также вводим единые обозначения. Общее название сеток, порождаемых нашей технологической цепочкой, — конформные симплициальные сетки. *Симплекс* — это треугольная ячейка для двумерных сеток или тетраэдральная ячейка для трехмерных сеток. В § 2.2 мы обсуждаем основные свойства симплициальных сеток, которые могут быть использованы в приложениях. В § 2.3 рассматриваются структуры данных и быстрые алгоритмы, лежащие в основе технологии генерации неструктурированных сеток.

Глава 3 посвящена построению неструктурированных симплициальных сеток в произвольных областях с кусочно-гладкой границей. В § 3.1 рассматриваются несколько способов *задания* границы области. Совокупность способов охватывает наиболее широкое множество областей, встречающихся в приложениях. В § 3.2 и 3.3 обсуждаются два основных метода построения двумерных сеток — метод триангуляции Делоне и метод продвигаемого фронта. В § 3.4 метод продвигаемого фронта переносится на построение поверхностных триангуляций, что является начальным этапом в генерации тетраэдральной сетки. Если граница трехмерной области уже задана поверхностной триангуляцией, то этот этап технологической цепочки может быть опущен. В случае низкого качества заданной поверхностной сетки может возникнуть необходимость в методах ее улучшения, рассматриваемых в § 3.5. Вопросам автоматической генерации тетраэдральных сеток с заданным следом на границе посвящены § 3.6 и 3.7.

Глава 4 освещает технологию многоуровневого иерархического локального измельчения и огрубления симплициальных сеток, сохраняющую регулярность ячеек-симплексов. В этой главе, как и во всех последующих, предполагается, что начальная сетка в области построена. В § 4.1 приводятся основные принципы иерархического измельчения сетки. В § 4.2 и 4.3 детально описываются алгоритмы метода бисекций для треугольных и тетраэдральных сеток. В § 4.4 рассматривается технология многоуровневого огрубления локально измельченной сетки,

которая открывает путь к построению последовательности сеток с динамической адаптацией, описанному в § 4.5.

Глава 5 представляет наиболее общую технологию адаптивного перестроения симплициальных сеток посредством их локальных модификаций. Гибкость и общность технологии обусловлена тем, что, во-первых, свойства адаптивной перестроенной сетки практически не зависят от свойств начальной сетки и, во-вторых, подобный подход допускает построение анизотропных адаптивных сеток. В § 5.1 представлены основные принципы организации алгоритмов перестроения сетки на основе ее локальных модификаций. В § 5.2 и 5.3 детально описаны все типы локальных операций с двумерными и трехмерными сетками соответственно. Последовательность локальных модификаций тетраэдральной сетки может быть вычислительно затратной, поэтому в § 5.4 мы описываем параллельную версию рассматриваемой технологии. Помимо адаптивности, набор локальных сеточных модификаций предоставляет также возможность исправления и распутывания сеток, чему посвящен § 5.5.

Несмотря на свой небольшой объем, глава 6 дает читателю важную практическую информацию об управлении свойствами неструктурированных сеток. В § 6.1 и 6.2 последовательно рассматриваются способы управления свойствами регулярных и анизотропных симплициальных сеток. Эти параграфы описывают построение необходимых скалярных и тензорных функций соответственно.

Адаптация расчетных сеток является важным звеном технологической цепочки построения сеток с оптимальным расположением узлов и реберных связей между ними, которые минимизируют ошибку сеточного решения. Некоторые важные практические вопросы сеточной адаптации рассмотрены в приложении. В § П.1 рассматривается перестроение сеток вблизи криволинейных границ. Неточная аппроксимация криволинейных границ может сильно ограничить точность сеточных решений. В § П.2 описываются методы апостериорной оценки ошибки для управления свойствами иерархических локально измельченных адаптивных сеток. В § П.3 приводятся различные подходы к управлению анизотропной адаптацией сеток посредством их локальных модификаций.

Глава 2

ОСНОВНЫЕ ПОНЯТИЯ

В данной главе будут введены основные понятия, необходимые для построения треугольных и тетраэдральных сеток. Мы также напомним основные результаты теории графов, которые будут использоваться в последующих главах. Наконец, мы опишем основные структуры данных, необходимые для реализации представленных в книге алгоритмов построения локально измельченных и неструктурированных сеток.

§ 2.1. Треугольные и тетраэдральные сетки

Треугольник на плоскости — простейший многоугольник, имеющий три вершины v_1, v_2, v_3 и три ребра e_{12}, e_{23}, e_{13}. *Ребро* e_{ij} — отрезок, соединяющий пару вершин v_i и v_j, где $1 \leqslant i < j \leqslant 3$. Если все три вершины треугольника лежат на одной прямой, то он называется *вырожденным*.

Пусть на плоскости задана декартова система координат (x, y), в которой любая точка v_i представима вектором $\mathbf{v}_i$ с двумя компонентами x_i и y_i. Направленное ребро e_{ij}, соединяющее две вершины v_i и v_j, представимо вектором $\mathbf{e}_{ij} = \mathbf{v}_j - \mathbf{v}_i$. (Здесь и далее мы будем использовать обычный шрифт для точек или ребер как таковых и жирный шрифт для точек или ребер как векторов.)

Треугольником Δ называется выпуклая оболочка трех вершин $\mathbf{v}_1$, $\mathbf{v}_2$ и $\mathbf{v}_3$, т. е. множество точек

$$\Delta = \left\{ \mathbf{x} : \ \mathbf{x} = \sum_{i=1}^{3} \alpha_i \mathbf{v}_i, \quad \alpha_i \geqslant 0, \quad \sum_{i=1}^{3} \alpha_i = 1 \right\}.$$

Алгебраической площадью треугольника Δ называется величина

$$S_\Delta = \frac{1}{2} \det\{\mathbf{e}_{12},\mathbf{e}_{13}\} \equiv \frac{1}{2} \left((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1) \right), \tag{2.1.1}$$

где $\det\{\mathbf{e}_{12},\mathbf{e}_{13}\}$ обозначает детерминант матрицы с двумя столбцами $\mathbf{e}_{12}$ и $\mathbf{e}_{13}$. Ввиду свойств детерминанта алгебраическая площадь меняет знак при перестановке местами любых двух вершин в треугольнике Δ. *Площадь* треугольника есть абсолютное значение алгебраической площади.

Каноническим треугольником назовем треугольник, в котором два ребра единичной длины совпадают с ортами выбранной декартовой

системы координат. *Равносторонним* треугольником является треугольник с равными длинами ребер. Диаметром треугольника Δ, как и любого многоугольника, является величина

$$\operatorname{diam}(\Delta) = \max_{\mathbf{x}\in\Delta,\mathbf{y}\in\Delta} |\mathbf{x}-\mathbf{y}|.$$

В силу выпуклости треугольника его диаметр совпадает с длиной наибольшего ребра. Примеры канонического и равностороннего треугольников показаны на рис. 2.1.

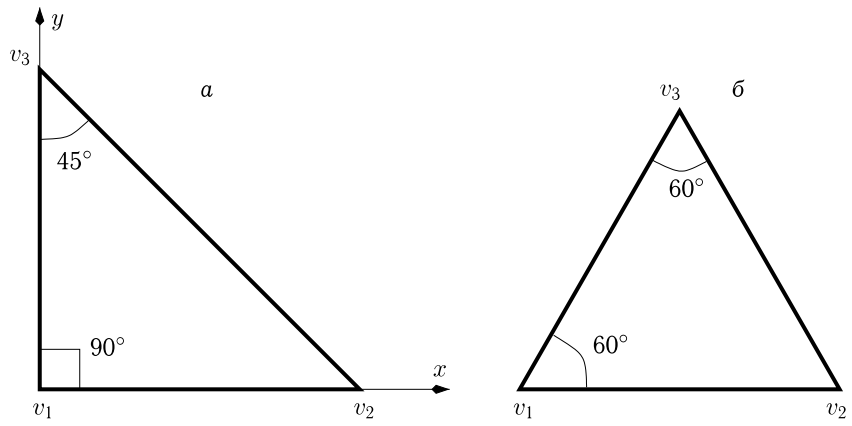


Рис. 2.1. Канонический (*а*) и равносторонний (*б*) треугольники

Площадь канонического треугольника равна $1/2$, а площадь равностороннего треугольника диаметра 1 равна

$$S_* = \frac{\sqrt{3}}{4}. \tag{2.1.2}$$

Треугольник называется *регулярным*, если отношение его площади и площади равностороннего треугольника того же диаметра является величиной порядка 1. Известны несколько эквивалентных мер регулярности треугольника. Пусть $|e|$ обозначает длину ребра e треугольника Δ, а P_Δ — его периметр: $P_\Delta = |e_{12}| + |e_{13}| + |e_{23}|$. В данной книге мы рассматриваем следующую меру регулярности, или качества формы, треугольника:

$$Q_\Delta = 12\sqrt{3}\frac{|S_\Delta|}{P_\Delta^2}. \tag{2.1.3}$$

Данная формула основана на следующем свойстве: равносторонний треугольник имеет наибольшую площадь среди всех треугольников с фиксированным периметром. Из этого определения следует, что Q_Δ является положительной величиной, не превышающей 1. Более того, $Q_\Delta = 1$ только для равностороннего треугольника.

Связь между качеством формы треугольника и параметрами его геометрической формы иллюстрируется в табл. 2.1 на двух примерах. Данные в первых двух строках соответствуют вытягиванию равностороннего треугольника в остроугольный равнобедренный треугольник. Коэффициент растяжения s треугольника равен 1 для равностороннего треугольника и растет линейно с наибольшей высотой треугольника. Отметим, что качество формы вытянутого треугольника падает примерно обратно пропорционально коэффициенту растяжения s.

Данные в последних двух строках соответствуют равнобедреннему треугольнику с изменяющимся углом при равных сторонах. Угол меньше 90° соответствует вытягиванию треугольника, а угол больше 90° — его сплющиванию. Отметим особо, что качество формы прямоугольного треугольника достаточно высоко и равно примерно 0,9.

Таблица 2.1
Изменения качества формы при вытягивании (вторая строка) и сплющивании (четвертая строка) равнобедренного треугольника

s	1	2	4	8	16	32	64	128	256
Q_Δ	1,0	0,849	0,563	0,325	0,174	0,090	0,046	0,023	0,012
α	10°	30°	50°	70°	90°	110°	130°	150°	170°
Q_Δ	0,382	0,820	0,983	0,986	0,892	0,738	0,548	0,336	0,113

На практике мы рекомендуем строить сетки с элементами, для которых $Q_\Delta \geqslant 0{,}7$. Для приведенных выше примеров это означает, что $s \in (1,\ 2{,}9)$ и $\alpha \in (23°{,}114°)$.

Треугольной сеткой, или *триангуляцией* Ω_h полигональной (многоугольной) области Ω, называется конечный набор треугольников Δ_i, $i = 1,\ \ldots,\ N$, такой, что:

— область Ω полностью покрыта этими треугольниками;

— граница области $\partial\Omega$ полностью покрыта ребрами треугольников;

— внутренности треугольников не пересекаются.

Данное определение сетки является довольно общим и включает как конформные, так и неконформные триангуляции (триангуляция называется *конформной*, если пересечение любых двух треугольников является либо их общей вершиной, либо их общим ребром, либо пустым множеством).

Триангуляция называется *равномерной*, если она состоит из равносторонних треугольников, и *квазиравномерной* с шагом h, если все ее треугольники близки (по мере Q_Δ) к равностороннему треугольнику диаметра h. Отметим, что равномерные сетки не существуют в трех измерениях.

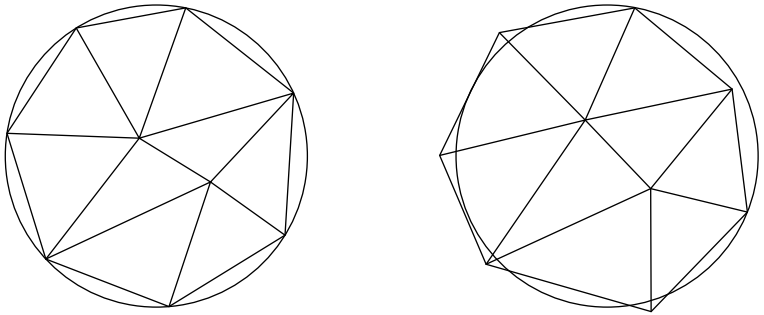
Триангуляция является *регулярной*, если она состоит из регулярных треугольников. Для треугольных регулярных сеток справедливы следующие неравенства:

$$c \leqslant \frac{\rho_\Delta}{\operatorname{diam}(\Delta)} < 1, \tag{2.1.4}$$

где ρ_Δ — радиус вписанной окружности, а константа c не зависит от треугольника сетки. В отличие от квазиравномерной сетки, регулярная сетка может содержать треугольники, сильно различающиеся в диаметре, хотя диаметры любых двух соседних треугольников отличаются не сильно, т. е. близки (по мере Q_Δ). Например, для регулярной сетки, показанной на рис. 1.1, *в*, имеем $c = (\sqrt{2}-1)/2$. Мерой регулярности сетки Ω_h, или ее качеством $Q(\Omega_h)$, может служить наименьшее из качеств формы Q_{Δ_i} треугольников Δ_i, составляющих сетку:

$$Q(\Omega_h) = \min_{1 \leqslant i \leqslant N} Q_{\Delta_i}.$$

Понятие триангуляции области Ω с кусочно-гладкой криволинейной границей является естественным обобщением понятия триангуляции полигональной области. Основное отличие состоит в том, что граница области приближается ребрами сетки, и допускаются как неполное покрытие приграничной части области, так и выход сетки за границы области, см. рис. 2.2.



а *б*

Рис. 2.2. Граничные узлы триангуляции лежат на криволинейной границе (*а*) и вблизи этой границы (*б*)

Во многих практических приложениях граничные узлы триангуляции лежат на криволинейной границе (см. рис. 2.2). Этим обеспечивается приближение границы области ребрами триангуляции со вторым порядком точности по отношению к длине ребра, т. е. растояниие от граничного ребра сетки e до криволинейной границы является величиной порядка $|e|^2$.

Понятие триангуляции можно обобщить для области, заданной на двумерной поверхности в трехмерном пространстве, а также

для замкнутой двумерной поверхности, являющейся границей трехмерного тела. В этом случае узлы сетки должны иметь три координаты (x, y, z), ребро треугольника является криволинейным отрезком, лежащим на двумерной поверхности, а абсолютное значение алгебраической площади (2.1.1) является лишь приближением интегральной площади треугольника.

Тетраэдральные сетки в пространстве определяются по аналогии с треугольными сетками на плоскости. *Тетраэдр* в трехмерном пространстве — это выпуклая оболочка четырех вершин v_1, v_2, v_3 и v_4. Если все вершины тетраэдра лежат в одной плоскости, то он называется *вырожденным*. *Ребром* тетраэдра назовем отрезок e_{ij}, соединяющий пару вершин v_i и v_j, а *гранью* тетраэдра назовем треугольник f_{ijk} с вершинами v_i, v_j и v_k.

Зададим в пространстве некоторую декартову систему координат (x, y, z), в которой любая точка v_i представляется вектором $\mathbf{v}_i$ с тремя компонентами x_i, y_i, z_i. Направленное ребро, соединяющее две вершины v_i и v_j, представимо вектором $\mathbf{e}_{ij} = \mathbf{v}_j - \mathbf{v}_i$. Ориентация грани с вершинами v_i, v_j и v_k задается вектором нормали $\mathbf{n}_{ijk} = \mathbf{e}_{ij} \times \mathbf{e}_{ik}$. Вектор нормали может быть направлен как внутрь, так и наружу тетраэдра.

Определение тетраэдра можно формализировать следующим образом. Тетраэдром с вершинами $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$ и $\mathbf{v}_4$ называется множество точек

$$\Delta = \left\{ \mathbf{x}\colon\ \mathbf{x} = \sum_{i=1}^{4} \alpha_i \mathbf{v}_i, \quad \alpha_i \geqslant 0, \quad \sum_{i=1}^{4} \alpha_i = 1 \right\}.$$

Алгебраическим, или ориентированным, *объемом* тетраэдра Δ называется величина

$$V_\Delta = \frac{1}{6} \det\{\mathbf{e}_{12},\mathbf{e}_{13},\mathbf{e}_{14}\}, \tag{2.1.5}$$

где $\det\{\mathbf{e}_{12},\mathbf{e}_{13},\mathbf{e}_{14}\}$ обозначает детерминант матрицы с тремя столбцами $\mathbf{e}_{12}$, $\mathbf{e}_{13}$ и $\mathbf{e}_{14}$. Алгебраический объем меняет знак при перестановке местами любых двух вершин в тетраэдре Δ. *Объем* тетраэдра есть абсолютное значение алгебраического объема.

Каноническим тетраэдром назовем тетраэдр, в котором три ребра совпадают с ортами выбранной декартовой системы координат. *Равносторонним*, *или правильным*, тетраэдром является тетраэдр с равными длинами ребер. Диаметром тетраэдра Δ называется длина его наибольшего ребра. Примеры канонического и правильного тетраэдров показаны на рис. 2.3.

Объем канонического тетраэдра равен $1/6$, а объем правильного тетраэдра диаметра 1 равен

$$V_* = \frac{1}{6\sqrt{2}}. \tag{2.1.6}$$

Тетраэдр называется *регулярным*, если отношение его объема и объема правильного тетраэдра того же диаметра является величиной

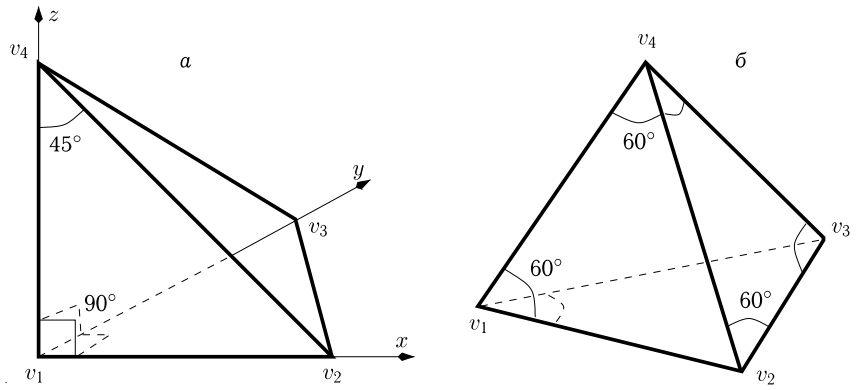


Рис. 2.3. Канонический и правильный тетраэдры

порядка 1. Известны несколько эквивалентных мер регулярности тетраэдра. Пусть P_Δ — сумма длин ребер тетраэдра Δ, т. е. $P_\Delta = |e_{12}| + |e_{13}| + |e_{14}| + |e_{23}| + |e_{24}| + |e_{34}|$. В данной книге мы рассматриваем следующую меру регулярности, или качества формы, тетраэдра:

$$Q_\Delta = 6^4\sqrt{2}\,\frac{|V_\Delta|}{P_\Delta^3}. \tag{2.1.7}$$

Из определения Q_Δ следует, что качество формы тетраэдра — положительная величина, не превышающая 1. Можно показать, что $Q_\Delta = 1$ только для правильного тетраэдра.

В дальнейшем (гл. 2–5) иногда вместо «качество формы» мы будем писать просто «качество».

Связь между значением качества тетраэдра и его формой на двух конкретных примерах показывает табл. 2.2. Данные в первых двух строках соответствуют вытягиванию правильного тетраэдра в направлении одной из его высот. Коэффициент растяжения s равен 1 для правильного тетраэдра. Сравнение с аналогичной таблицей для треугольников показывает значительно более быстрое уменьшение качества с увеличением s. При бóльших значениях s удвоение s приводит

Таблица 2.2
Изменения качества при модификации правильного (вторая строка) и канонического (пятая строка) тетраэдров

s	1	2	4	8	16	32	64
Q_Δ	1,0	0,785	0,398	0,148	0,046	0,013	0,003
α	61°	70°	90°	110°	130°	150°	170°
β	84,2°	71°	54,7°	41,4°	29,1°	17,4°	5,78°
Q_Δ	0,468	0,999	0,804	0,575	0,386	0,223	0,073

к уменьшению качества тетраэдра в 3–4 раза. Поэтому на практике мы рекомендуем строить сетки с качеством $Q(\Omega_h) \geqslant 0{,}2$.

Данные в последних трех строках таблицы соответствуют тетраэдру, который получается из канонического тетраэдра путем изменения трех равных двугранных углов при вершине v_1 (см. рис. 2.3). Эти углы могут меняться в ограниченном диапазоне $\alpha \in (60°,\ 180°)$, где крайние значения соответствуют вырожденным тетраэдрам. В предпоследней строке таблицы мы приводим значения β трех других равных двугранных углов. Качество прямоугольного тетраэдра равно примерно 0,8. Аналогично табл. 2.1, при стремлении α к 180° качество тетраэдра также уменьшается. Однако ограниченность двухгранных углов в тетраэдре не означает его высокое качество: в предельном случае, когда $\alpha \to 60°$ и $\beta \to 90°$, качество Q_Δ стремится к нулю. В этом проявляется существенное отличие качества тетраэдра от качества треугольника, где ограниченность углов означает его высокое качество. В тетраэдре необходима ограниченность как двугранных, так и плоских углов.

Полиэдральная (*многогранная*) область — это область с кусочно-гладкой границей, представимой конечным набором плоских граней. Тетраэдральной сеткой, или *тетраэдризацией* Ω_h полиэдральной области Ω называется конечный набор тетраэдров Δ_i, $i = 1, \ldots, N$, такой, что:

— область Ω полностью покрыта этими тетраэдрами;
— граница области полностью покрыта гранями тетраэдров;
— внутренности тетраэдров не пересекаются.

Это определение сетки включает как конформные, так и неконформные сетки. Тетраэдризация называется *конформной*, если пересечение любых двух тетраэдров является либо их общей вершиной, либо их общим ребром, либо их общей гранью, либо пустым множеством. Тетраэдризация называется *квазиравномерной* с шагом h, если все ее тетраэдры близки по мере Q_Δ к правильному тетраэдру диаметра h. Отметим, что пространство нельзя покрыть правильными тетраэдрами, т. е. равномерных тетраэдральных сеток не существует.

Тетраэдризация является *регулярной*, если она состоит из регулярных тетраэдров. Для регулярных сеток справедливы неравенства (2.1.4). Мерой регулярности $Q(\Omega_h)$ тетраэдризации Ω_h может служить наименьшее из качеств Q_{Δ_i} тетраэдров Δ_i, составляющих сетку:

$$Q(\Omega_h) = \min_{1 \leqslant i \leqslant N} Q_{\Delta_i}. \tag{2.1.8}$$

Регулярные триангуляции и тетраэдризации широко используются в инженерных вычислениях, поскольку дискретизации на регулярных сетках обладают рядом преимуществ, таких, как устойчивость дискретизации, приемлемая обусловленность порождаемых матриц, существование эффективных алгоритмов поиска.

Понятие тетраэдризации области с криволинейной кусочно-гладкой границей является естественным обобщением понятия тетраэдризации

полиэдральной области, при котором граница области приближается граничными гранями сетки. При этом допускаются как неполное покрытие приграничной части области, так и выход сетки за границы области. На практике, как правило, граничные узлы тетраэдризации лежат на границе области, что позволяет приближать эту границу со вторым порядком точности по отношению к диаметру грани.

В конце параграфа мы представим систему обозначений, используемую в дальнейшем для различных элементов сетки. Термин «симплекс» используется для общего названия треугольника и тетраэдра. Символ Ω используется для обозначения расчетной области, а Ω_h — для ее триангуляции или тетраэдризации. Суперэлемент $\sigma(v)$ обозначает множество симплексов с общей вершиной v. Символ $\sigma(\Delta)$ обозначает суперэлемент, или множество симплексов, пересекающихся с симплексом Δ. Таким образом,

$$\sigma(\Delta) = \bigcup_{v_i \in \Delta} \sigma(v_i).$$

Аналогично, суперэлемент $\sigma(e)$ обозначает множество симплексов с общим ребром e. Если $v \in e \in \Delta$, то

$$\sigma(v) \subset \sigma(e) \subset \sigma(\Delta).$$

Пусть d обозначает размерность пространства: $d = 2$ для плоскости и $d = 3$ для пространства. Сеточный элемент (симплекс) с вершинами v_{k_i}, $i = 1, \ldots, d+1$, обозначается как $\Delta(v_{k_1}, \ldots, v_{k_{d+1}})$, или сокращенно $\Delta_{k_1 \ldots k_{d+1}}$. Аналогично, мы будем использовать $f(v_{k_1}, v_{k_2}, v_{k_3})$, или $f_{k_1 k_2 k_3}$ для обозначения треугольной грани в пространстве, и $e(v_{k_1}, v_{k_2})$, или $e_{k_1 k_2}$ для обозначения ребра. Для каждого типа сеточного элемента, в зависимости от контекста, мы будем использовать обозначения, приведенные в табл. 2.3:

Таблица 2.3

Таблица обозначений для элементов сетки

Элемент сетки	как объект	как вектор
вершина	$v \quad v_i$	$\mathbf{v} \quad \mathbf{v}_i$
ребро	$e \quad e_{ij}$	$\mathbf{e} \quad \mathbf{e}_{ij} \equiv \mathbf{v}_j - \mathbf{v}_i$
грань	$f \quad f_{ijk}$	$\mathbf{n}_f \quad \mathbf{n}_{ijk} \equiv \mathbf{e}_{ij} \times \mathbf{e}_{ik}$

§ 2.2. Свойства сеток и элементы теории графов

Взаимосвязь между графами и сетками позволяет использовать некоторые результаты теории графов при исследовании свойств сеток и осуществлении операций над сетками. Приближенные соотношения между количествами узлов, ребер и элементов сетки, приведенные

ниже, полезны для оценки компьютерной памяти, необходимой для хранения сетки.

Наиболее известным результатом теории графов, применимым к анализу сеток, является формула Эйлера, связывающая количества вершин, ребер и граней у любого выпуклого многогранника. Пусть N_v, N_e, N_f обозначают соответственно количества вершин, ребер и граней некоторого многогранника. Отметим, что грани многогранника могут быть произвольными многоугольниками. Тогда

$$N_f = N_e - N_v + 2. \tag{2.2.1}$$

Например, для тетраэдра $N_v = N_f = 4$, $N_e = 6$, а для куба $N_v = 8$, $N_f = 6$, $N_e = 12$.

Опираясь на формулу Эйлера (2.2.1), мы выведем соотношение между количествами узлов, ребер и треугольников в произвольной двумерной конформной триангуляции. Для этого заметим, что любую конформную триангуляцию некоторой односвязной области можно представить как распластанную на плоскости поверхность некоторого многогранника, в которой удалена одна из граней (см. рис. 2.4). Показанный многогранник является объединением тетраэдра Δ_{4567} и треугольной призмы. При таком распластывании предполагается, что удаленная грань соответствует дополнению рассматриваемой триангуляции до плоскости или бесконечной ячейке.

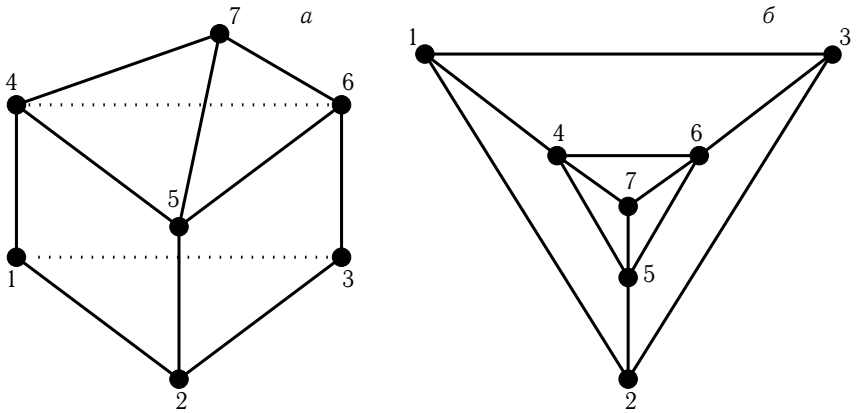


Рис. 2.4. Многогранник (*а*) и его распластанная поверхность, нижняя грань удалена (*б*)

Поэтому для конформной триангуляции односвязной области верно соотношение

$$N_f = N_e - N_v + 1.$$

В случае неодносвязной области конформная триангуляция будет содержать N_h нетриангулированных многоугольников, каждый из которых соответствует региону, вырезанному из области. Такие регионы

задаются, например, препятствиями в задачах обтекания или пустотами в образце в задачах механики твердого тела. Каждый из таких многоугольников вносит вклад в число граней в формуле Эйлера, однако не должен содержать треугольники рассматриваемой триангуляции. Таким образом, модификация формулы Эйлера (2.2.1) для случая конформной триангуляции неодносвязной области имеет вид

$$N_f + N_h = N_e - N_v + 1. \tag{2.2.2}$$

В качестве иллюстрации мы будем использовать треугольную сетку, показанную на рис. 2.5.

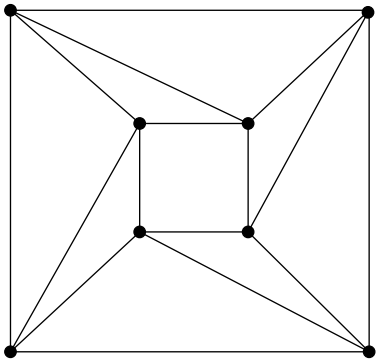


Рис. 2.5. Треугольная сетка в области с квадратным отверстием: $N_v = 8$, $N_e = 16$, $N_f = 8$, $N_h = 1$, $N_{e_i} = 8$ и $N_{e_b} = 8$

Формулу (2.2.2) можно переписать, используя число граничных ребер вместо числа всех ребер триангуляции. Граничное ребро e_b принадлежит лишь одному треугольнику, в отличие от внутреннего ребра e_i, принадлежащего двум треугольникам. Поэтому между числами граничных ребер N_{e_b}, внутренних ребер N_{e_i} и числом треугольников N_f в конформной триангуляции существует следующая взаимосвязь:

$$2N_{e_i} + N_{e_b} = 3N_f. \tag{2.2.3}$$

Используя формулы (2.2.2) и (2.2.3), мы получаем соотношение для числа N_f:

$$N_f = 2N_v - N_{e_b} + 2N_h - 2. \tag{2.2.4}$$

Поскольку $N_{e_b} \geqslant 3$ и $N_v \geqslant 3$, из (2.2.4) мы получаем неулучшаемую оценку количества треугольников при $N_h = 0$:

$$N_f \leqslant 2N_v - 5. \tag{2.2.5}$$

Поскольку в сетке с $N_h > 0$ количество треугольников меньше, чем в сетке с триангулированными пустотами, оценка (2.2.5) верна и для общего случая $N_h \geqslant 0$. При построении динамических сеток мы можем

выделить необходимую для работы память, используя только один параметр — максимально допустимое число узлов триангуляции.

Из формул (2.2.2) и (2.2.5) мы получаем неулучшаемую оценку количества ребер

$$N_e \leqslant 3N_v - 6. \tag{2.2.6}$$

В подавляющем числе практических случаев количество узлов в конформной триангуляции значительно превосходит количество граничных ребер N_{e_b} и количество вырезанных компонент N_h, поэтому на практике имеют место приближенные соотношения

$$N_f \approx 2N_v \quad \text{и} \quad N_e \approx 3N_v,$$

выражающие асимптотическую зависимость между количествами треугольников, ребер и узлов конформной триангуляции.

Отметим, что формула (2.2.4) выведена в предположении, что Ω_h — триангуляция связной области, заданной на плоскости. Для триангуляций замкнутых поверхностей, заданных в пространстве, таких, как границы трехмерных тел, эта формула может оказаться неверной. В этом случае нужно использовать топологические характеристики поверхности, основанные на классификации замкнутых двумерных поверхностей в виде конечной суммы сфер и торов. Например, если на связной замкнутой поверхности можно провести N_g непересекающихся замкнутых кривых, не разрезающих поверхность на несвязные компоненты, то такая поверхность гомеоморфна сфере с N_g ручками, и для ее триангуляции верно [3, 17]

$$N_f = 2N_v + 4N_g - 4.$$

Рассмотрим теперь тетраэдральные сетки и воспользуемся формулой Эйлера (2.2.1) для вывода соотношения между количеством тетраэдров N_t, ребер N_e и узлов N_v в тетраэдризации связной области, удовлетворяющей следующему ограничению: для каждого узла v множество тетраэдров, содержащих v (инцидентных к v), формирует многогранник, гомеоморфный сфере. Напомним, что каждый такой многогранник называется суперэлементом и обозначается $\sigma(v)$, или сокращенно σ. Введенное ограничение не обременительно на практике.

Определим характеристическую функцию δ_b для границы сеточной области. Пусть $\delta_b(v) = 1$, если узел v лежит на границе Ω_h, и $\delta_b(v) = 0$ для внутреннего узла области. Введем количества граней $N_f(\sigma)$, ребер $N_e(\sigma)$, узлов $N_v(\sigma)$ на поверхности многогранника σ, количество тетраэдров $N_t(\sigma)$ в σ, количество ребер $N_e(v)$, инцидентных к v, а также количество граней $N_b(\sigma)$ в σ, лежащих на границе Ω_h и содержащих v. Тогда:

$$N_v(\sigma) = N_e(v) + \delta_b(v), \quad N_f(\sigma) = N_t(\sigma) + N_b(\sigma),$$

а формула (2.2.1) записывается для σ следующим образом:

$$N_t(\sigma) + N_b(\sigma) + N_e(v) + \delta_b(v) = N_e(\sigma) + 2. \tag{2.2.7}$$

Используя формулу (2.2.3), на поверхности σ имеем тождество

$$2N_e(\sigma) = 3(N_t(\sigma) + N_b(\sigma)),$$

что с учетом (2.2.7) дает

$$N_e(v) = \frac{1}{2}\,N_t(\sigma) + \frac{1}{2}\,N_b(\sigma) - \delta_b(v) + 2.$$

Просуммировав по всем узлам сетки Ω_h и учтя, что

$$\sum_v N_e(v) = 2N_e, \quad \sum_v N_t(\sigma) = 4N_t,$$

мы получаем

$$N_e = N_t + \frac{1}{4}\sum_v N_b(\sigma(v)) - \frac{1}{2}\sum_v \delta_b(v) + N_v.$$

Пусть N_{f_b} и N_{v_b} — число граничных граней и число граничных узлов тетраэдризации Ω_h соответственно. Поскольку $\sum\limits_v N_b(\sigma(v)) = 3N_{f_b}$ и $\sum\limits_v \delta_b(v) = N_{v_b}$, то

$$N_e = N_t + N_v + \frac{3}{4}\,N_{f_b} - \frac{1}{2}\,N_{v_b}.$$

Поскольку $N_v \geqslant 4$, $N_{f_b} \geqslant N_{v_b}$, а максимально возможное количество ребер N_e ограничено пессимистичной оценкой

$$N_e \leqslant \frac{1}{2}\,N_v(N_v - 1), \tag{2.2.8}$$

то

$$N_t \leqslant \frac{1}{2}\,N_v(N_v - 3) - \frac{3}{4}\,N_{f_b} + \frac{1}{2}\,N_{v_b} \leqslant \frac{1}{2}\,N_v(N_v - 3). \tag{2.2.9}$$

Однако, наложив дополнительные ограничения, можно получить и более оптимистичную оценку. Если для каждого узла v его степень (количество инцидентных, или сходящихся в нем, ребер) ограничена некоторой величиной α, т. е. $N_e(v) \leqslant \alpha$, то

$$N_e \leqslant \frac{\alpha}{2}\,N_v$$

и

$$N_t \leqslant \left(\frac{\alpha}{2} - 1\right) N_v.$$

Таким образом, в тетраэдризациях с равномерно ограниченной степенью узлов количество тетраэдров оценивается сверху через линейную функцию от количества узлов. Отметим, что существуют тетраэдризации, в которых $N_t \simeq N_v^2$, однако методы, рассматриваемые в этой книге, порождают сетки с равномерно ограниченной степенью узлов. Здесь $a \simeq b$ означает, что отношение a/b — величина порядка 1. В неструктурированных тетраэдризациях с большим числом тетраэдров и узлов, где подавляющее число вершин — внутренние, верно приблизительное соотношение

$$N_t \approx 5{,}5N_v.$$

Формула Эйлера допускает обобщение на многомерный случай следующим образом. Рассмотрим d-мерный выпуклый многогранник (политоп), состоящий из набора k-мерных граней, $k = 0, \ldots, d-1$. Под k-мерной гранью здесь понимается выпуклое открытое множество, лежащее в некоторой k-мерной гиперплоскости. Вершина многогранника является 0-мерной гранью, ребро — 1-мерной гранью, а сам многогранник — d-мерной гранью. Для удобства введем (-1)-мерную грань. Пусть n_k обозначает количество k-мерных граней многогранника, причем любому многограннику соответствует ровно одна (-1)-мерная грань, а d-мерная грань совпадает с самим многогранником. Таким образом, по определению, $n_{-1} = 1$ и $n_d = 1$. Обобщенная формула Эйлера для d-мерных политопов выглядит следующим образом:

$$\sum_{k=-1}^{d} (-1)^k n_k = 0.$$

Связь между d-мерными и трехмерными определениями такова:

$$n_2 = N_f, \quad n_1 = N_e, \quad n_0 = N_v.$$

Теория графов применима к анализу свойств треугольных сеток. Одним из следствий теории плоских графов является утверждение о возможности распутывания любой триангуляции, представляющей собой «запутанную» сеть треугольников, граничащих попарно через свои целые ребра, в конформную триангуляцию. Поясним это утверждение, а также объясним термин «распутывание триангуляции».

Графом $G = (V, E)$ называется множество вершин V и множество ребер E, не обязательно прямых, причем каждое ребро из E соединяет пару вершин из V. *Планарным* графом называется ненаправленный граф, который может быть уложен на плоскости без самопересечений. Планарный граф, нарисованный на плоскости, называется *плоским* графом. Гранью плоского графа называется ограниченная односвязная область, граница которой задана ребрами графа в виде простого цикла v_1v_2, v_2v_3, …, v_kv_1, где все вершины входят два раза. Если все грани плоского графа являются треугольниками (т. е. $k = 3$), то такой граф называется *триангулированным*. Триангуляцией графа называется процедура дополнения множества ребер E до получения триангулированного графа.

Согласно теореме Радо [80], любой многоугольник с границей, имеющей вид простой замкнутой ломаной с конечным числом звеньев,

допускает конформную триангуляцию без добавления новых точек. В терминах теории графов это утверждение выглядит следующим образом.

Теорема 2.2.1. *Любой плоский граф с прямолинейными ребрами можно триангулировать.*

Данное утверждение, как и теорема Радо, доказывает существование конформной триангуляции для многоугольных областей.

Построение или перестроение триангуляции может привести к неконформной сетке. Под простым распутыванием сетки будем понимать перемещение ее узлов, сохраняющее реберные связи между узлами, с целью получения конформной триангуляции. Теорема Фари [47] гарантирует существование такого перемещения узлов для сетки, заданной планарным триангулированным графом.

Теорема 2.2.2. *Любой планарный граф имеет плоское представление, в котором все ребра представлены в виде отрезков прямых.*

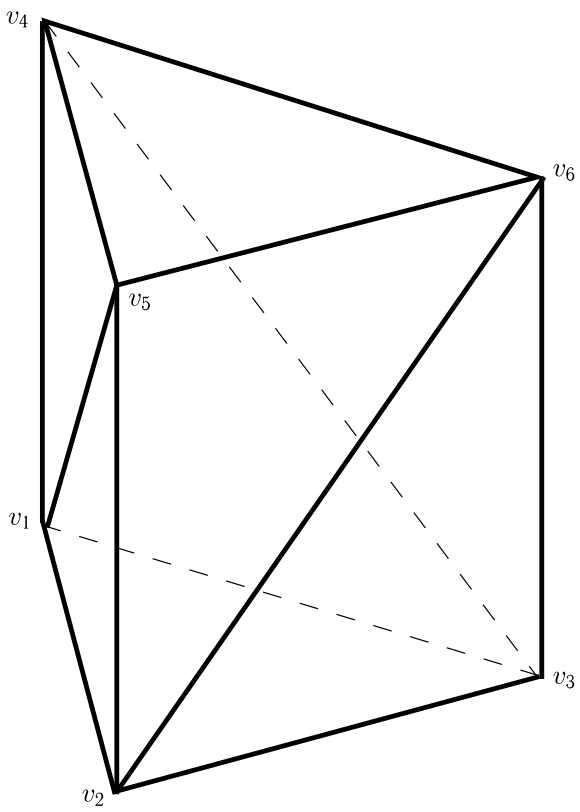


Рис. 2.6. Призма Шонхардта с треугольным основанием $v_1\,v_2\,v_3$

Существуют критерии планарности графов [63], поэтому для треугольных сеток, удовлетворяющих таким критериям, существует процедура простого распутывания сетки в конформную триангуляцию.

К сожалению, вышеприведенный анализ не может быть перенесен на случай конформных тетраэдризаций. Например, в области с заданной поверхностной сеткой, изображенной на рис. 2.6, нельзя построить конформную тетраэдризацию без добавления дополнительных внутренних узлов. В гл. 6 мы рассмотрим более сложные алгоритмы распутывания сетки, которые меняют ее топологию и поэтому применимы для распутывания как триангуляций, так и тетраэдризаций.

Другим важным понятием теории планарных графов является понятие *дуального* графа. Для планарного графа G дуальным является граф G_{dual} со следующими свойствами:

— каждая вершина G_{dual} асоциирована с гранью G;

— каждое ребро G_{dual} асоциировано с ребром G;

— если ребро G разделяет две грани f_i и f_j, то соответствующее ребро дуального графа G_{dual} соединяет вершины G_{dual}, асоциированные с f_i и f_j.

Понятие дуального графа используется как в некоторых методах построения треугольных сеток и сеток Вороного, так и в методах дискретизации дифференциальных уравнений [45, 74].

§ 2.3. Структуры данных и быстрые алгоритмы

Для хранения сетки и выполнения каких-либо операций над ней необходимо использовать различные структуры данных. Существует большое количество разнообразных структур данных для хранения сетки, ориентированных на разные цели [18, 50, 54]. Перечисление и сравнительный анализ этих структур данных выходит за рамки данной книги, и мы ограничимся лишь теми структурами, которые часто используются при работе с триангуляциями и легко переносятся на тетраэдризации.

Минимальным и простейшим представлением треугольной сетки является список координат пронумерованных узлов (x_i, y_i) и список пронумерованных треугольников, каждый из которых задан индексами трех узлов (i_1, i_2, i_3). При этом координаты хранятся в вещественном двумерном массиве $Vrt(2, N_v)$, а треугольники — в целочисленном двумерном массиве $Tri(3, N_f)$.

В конечноэлементных методах могут использоваться также двумерные целочисленные массивы ребер $Edge(2, N_e)$ и граничных ребер $Bnd(2, N_{e_b})$, хранящие индексы узлов. Эти структуры являются вспомогательными, так как они могут быть созданы автоматически из массива Tri.

x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_{N_v}	y_{N_v}

Рис. 2.7. Структурированный список $Vrt(2, N_v)$ для хранения координат узлов

Все перечисленные массивы представляют собой *структурированные списки*, или таблицы с заданной длиной строки (см. рис. 2.7). К структурированным спискам относятся также данные о номерах треугольников, смежных с каждым треугольником. Построение такого списка обсуждается в гл. 4.

Алгоритм перебора элементов структурированного списка очень прост. Он состоит из двух вложенных циклов по строкам и столбцам соответствующего двумерного массива, как показано в алгоритме 1 для структурированного списка Tri.

Алгоритм 1. Перебор элементов структурированного списка Tri

```
1: for i = 1, N_f do
2:     for j = 1, 3 do
3:         return Tri(j, i) – номер j-й вершины треугольника i
4:     end for
5: end for
```

В дальнейшем мы будем использовать обозначение XY для массива, где сеточный объект $x \in X$ содержит ссылки на ближайшие сеточные объекты $y \in Y$, такие, что $x \cap y \neq \varnothing$. Например, TV — другое обозначение для двумерного массива $Tri(3, N_f)$, где T обозначает треугольник, а V — узел сетки.

В ряде приложений нужны данные, которые разумно хранить в виде неструктурированного списка. Например, данные о номерах треугольников, содержащих узел сетки (сходящихся в общей вершине), характеризуются непостоянностью числа треугольников, приписываемых каждому узлу. Разумеется, такие данные можно также хранить в виде двумерного массива $VT(K_{\max}, N_v)$, где $K_{\max}$ — максимально возможное число треугольников, сходящихся в общей вершине. Однако подобная стратегия может значительно увеличить объем оперативной памяти компьютера, резервируемой под данные, поскольку $K_{\max}$ может значительно превосходить среднее по сетке значение, и бóльшая часть массива VT будет заполнена нулями.

Неструктурированный список представляется двумя линейными целочисленными массивами. В рассматриваемом случае первый линейный массив $iVT(3N_f)$ содержит только ненулевые значения из массива VT, упорядоченные по строкам. Второй линейный массив $nVT(N_v)$ содержит индексы элементов массива iVT, являющихся последними в каждой строке массива VT (см. рис. 2.8). Данный формат аналогичен разреженному строчному формату, широко используемому при хранении разреженных матриц.

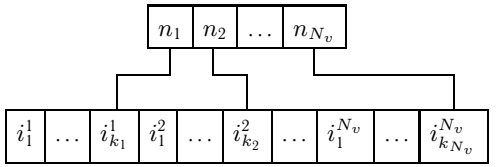


Рис. 2.8. Неструктурированный список для хранения треугольников, содержащих общую вершину. Массив $nVT(N_v)$ наверху, массив $iVT(3N_f)$ внизу

Алгоритм 2. Перебор элементов неструктурированного списка iVT, nVT

```
1: Пусть n_1 = 1
2: for i = 1, N_v do
3:     n_2 = nVT(i)
4:     for n = n_1, n_2 do
5:         k = n − n_1 + 1
6:         return iVT(n) — номер k-го треугольника, содержащего
                узел i
7:     end for
8:     n_1 = n_2 + 1
9: end for
```

Количество ненулевых элементов в каждой строке массива VT вычисляется следующим образом:

$$\begin{aligned} &nVT(i) - nVT(i-1), && i > 1;\\ &nVT(1), && i = 1. \end{aligned}$$

Алгоритм перебора элементов неструктурированного списка состоит из двух вложенных циклов, представленных в алгоритме 2.

Отметим, что количество элементов в массиве iVT равно количеству элементов в массиве TV, и это не случайно. Неструктурированный список iVT, nVT является обратным к структурированному списку TV и поэтому должен содержать то же самое число элементов. Точно так же структурированный список TV может рассматриваться как обратный к неструктурированному списку iVT, nVT.

Имеются простые алгоритмы с линейной сложностью для создания обратных списков. Например, построение списка iVT, nVT описано в алгоритме 3.

Алгоритм 3. Построение неструктурированного списка iVT, nVT

```
Инициализировать nVT(1) = ... = nVT(N_v) = 0
for n = 1,N_f do
    for i = 1,3 do
        i_1 = TV(i,n)
        nVT(i_1) = nVT(i_1) + 1
    end for
end for
for n = 2,N_v do
    nVT(n) = nVT(n) + nVT(n − 1)
end for
сохранить nLast = nVT(N_v)
for n = 1,N_f do
    for i = 1,3 do
        i_1 = TV(i,n)  и  k = nVT(i_1)
        iVT(k) = n
        nVT(i_1) = k − 1
    end for
end for
for n = 1,N_v − 1 do
    nVT(n) = nVT(n + 1)
end for
восстановить nVT(N_v) = nLast
```

Процесс построения или перестроения сетки сопряжен с ее постоянными модификациями, а следовательно, и с постоянными изменениями структур данных. Одной из важных проблем, возникающих при реализации сеточных алгоритмов, является проблема быстрого поиска внутри динамической структуры данных.

Вначале рассмотрим проблему поддержания *упорядоченного списка* элементов некоторого одномерного динамического массива вещественных чисел $Q(N)$. Предположим, что этот массив упорядочен по возрастанию элементов:

$$Q(i) \leqslant Q(i+1). \tag{2.3.1}$$

Напомним классический алгоритм 4 поиска интервала, содержащего значение x, в упорядоченном массиве чисел $Q(N)$. Этот алгоритм основан на методе бисекции (или деления пополам, или дихотомии). Арифметическая сложность метода бисекции пропорциональна $\log_2 N$.

Алгоритм 4. Поиск интервала в упорядоченном списке методом бисекции

```
l = 1, r = N
while r − l ≠ 1 do
    i = [(l + r)/2]
    if x ⩽ Q(i) then r = i
    else l = i
    end if
end while
if Q(l) ⩽ x ⩽ Q(r) then found=.TRUE.
else found=.FALSE.
end if
```

В задачах, где размер массива Q и значения его элементов постоянно меняются, поддержание его физической упорядоченности неэффективно, поэтому предположение (2.3.1) уже неверно. Например, вставка нового элемента во вторую позицию требует сдвига $(N-1)$ элементов. Для задания порядка используется связный список, основанный на двумерном массиве $Lst(2, N)$ таком, что

$$Q(Lst(1, i)) \leqslant Q(i) \leqslant Q(Lst(2, i)), \qquad 1 < i < N,$$

где $Q(Lst(1, i))$ и $Q(Lst(2, i))$ — ближайшие (по значению) к $Q(i)$ элементы вещественного массива Q. Массив Q, для которого известен связный список Lst, будем называть *неявно упорядоченным списком*. Начало $Q(i_b)$ и конец $Q(i_e)$ неявно упорядоченного списка определяются равенствами $Lst(1, i_b) = 0$ и $Lst(2, i_e) = 0$. Пример, приведенный на рис. 2.9, показывает, что начало и конец списка могут располагаться в соседних ячейках массива Q.

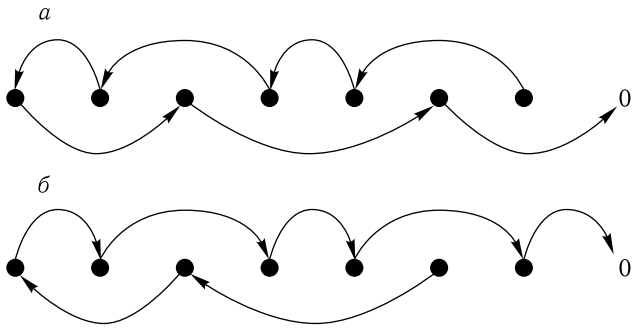


Рис. 2.9. Неявно упорядоченный список: *а* — ссылки $Lst(1, i)$, *б* — ссылки $Lst(2, i)$

Изменения в массиве чисел Q предполагают добавление или удаление элемента и соответственно увеличение или уменьшение N на единицу, а также изменение значения какого-то элемента без изменения N. Эти операции легко реализуются с помощью связного списка Lst, если известно, в какую позицию нужно поместить новый элемент или переместить измененный элемент. Например, процедура удаления элемента i из связного списка представлена в алгоритме 5.

Алгоритм 5. Удаление элемента i из связного списка Lst

1: $p = Lst(1, i)$, $n = Lst(2, i)$
2: **if** $p \neq 0$ **then** $Lst(2, p) = n$
3: **end if**
4: **if** $n \neq 0$ **then** $Lst(1, n) = p$
5: **end if**
6: $N = N - 1$

При удалении элемента неявно упорядоченного списка соответствующая ячейка памяти в массиве Q остается на месте. Поскольку связный список Lst больше не ссылается на эту ячейку памяти, она становится пропуском в массиве Q. Чтобы избежать неконтролируемого увеличения числа пропусков, необходимо постоянно заполнять их при добавлении новых элементов в список Q, т. е. требуется хранить список пропусков в дополнительном линейном массиве.

Процедура добавления элемента i в связный список также проста и представлена в алгоритме 6. Процедура переноса элемента связного списка из одной позиции в другую получается последовательным применением алгоритмов 5 и 6.

Алгоритм 6. Добавление элемента i после p-го элемента связного списка Lst

1: $Lst(2, i) = Lst(2, p)$, $n = Lst(2, i)$
2: $Lst(1, i) = p$
3: $Lst(2, p) = i$
4: **if** $n \neq 0$ **then** $Lst(1, n) = i$
5: **end if**
6: $N = N + 1$

Быстрый поиск позиции, куда нужно вставить или переместить элемент неявно упорядоченного списка, требует дополнительной структуры. Действительно, последовательный перебор элементов массива Q с использованием указателей $Lst(2, i)$ на следующий элемент, вплоть до нахождения нужной позиции, требует $O(N)$ операций присваивания и сравнения, что является недопустимо дорогой процедурой. Метод бисекции не применим к массиву Q, поскольку массив не упорядочен явно. Метод бисекции также не применим к связному списку Lst, поскольку середина списка неизвестна.

Наш алгоритм быстрого поиска основан на введении дополнительных указателей на элементы массива $Lst(2, N)$. Эти указатели хранятся в одномерном массиве $Ptr(M)$, где M — число указателей. Указатели $Ptr(i)$ разбивают массив Q на блоки с длинами, примерно равными $\log_2 N$, так что

$$Q(Ptr(i)) \leqslant Q(Ptr(j)), \quad 1 \leqslant i < j \leqslant M.$$

Для удобства предположим, что $Ptr(1) = i_b$ и $Ptr(M) = i_e$. Таким образом, число указателей M примерно равно $N/\log_2 N$. Массив Ptr задает некоторый порядок в неявно упорядоченном списке Q. Массив элементов $Q(Ptr(i))$ является явно упорядоченным подмножеством массива Q, поэтому к нему применим метод бисекции. Список указателей Ptr задает *блочное упорядочение* для Q. Массив Q, для которого известны Ptr и Lst, будем называть *блочно упорядоченным* списком.

Быстрый поиск интервала, содержащего значение x в блочно упорядоченном списке вещественных чисел Q, осуществляется в два этапа, показанных в алгоритме 7. Шаг 1 этого алгоритма представляет собой метод бисекции для массива указателей Ptr, и его арифметическая цена равна $O(\log_2 M) = O(\log_2 N)$. После этого последовательно перебираются все элементы i-го блока до нахождения искомого интервала. Следовательно, арифметическая цена второго этапа также равна $O(\log_2 N)$.

Алгоритм 7. Поиск интервала в блочно упорядоченном списке

1: Применить алгоритм 4 для поиска i-го блока, содержащего x, т. е. $Q(Ptr(i)) \leqslant x \leqslant Q(Ptr(i + 1))$
2: $i_1 = Ptr(i)$, $i_2 = Lst(2, i_1)$
3: **while** $x > Q(i_1)$ и $i_2 \neq 0$ **do**
4: $\quad i_1 = i_2$, $i_2 = Lst(2, i_1)$
5: **end while**
6: **return** i_1, i_2

Изменения в вещественном массиве Q требуют периодического обновления массива указателей Ptr для поддержания фиксированной длины блоков. Эти указатели сдвигаются на предшествующие или последующие элементы связного списка Lst. Изменение i-го указателя может привести к необходимости изменения $(i - 1)$-го и $(i + 1)$-го указателей. В наихудшем случае общее количество операций присваивания на этом шаге может достигать M. Значительное сокращение вызовов процедуры обновления указателей достигается введением плавающей длины блоков. До тех пор, пока все длины блоков лежат в промежутке $[0{,}75 \log_2 N, 1{,}25 \log_2 N]$, процедура обновления указателей $Ptr(i)$ не применяется. Эффективность предложеного метода

поддержания динамического блочно упорядоченного списка вещественных чисел обсуждается в гл. 5 при построении адаптивных сеток.

Многие операции с неструктурированными сетками основаны на быстрой локализации различных элементов сетки (ребер, граней, ячеек) в заданной окрестности. Наиболее удобной вспомогательной структурой данных для быстрого поиска элементов триангуляции (или тетраэдризации) является структура четверичного (или восьмеричного) дерева. Сеточные аналоги четверичных и восьмеричных деревьев изображены на рис. 2.10. Рассмотрим четверичное дерево более подробно.

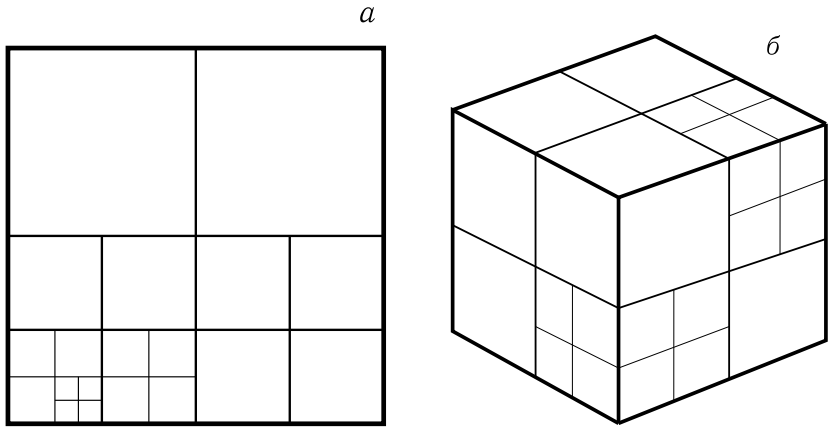


Рис. 2.10. Сеточные аналоги четверичного (*а*) и восьмеричного (*б*) деревьев

Четверичное дерево устроено как обычное бинарное дерево; отличие в том, что у каждой вершины не пара потомков, а четыре. Каждая вершина такого дерева отвечает за некоторую часть единичного квадрата $[0,\,1]\times[0,\,1]$. При этом корень дерева отвечает за весь квадрат, а его потомки отвечают каждый за свою четверть родительского квадрата. При переходе от родителя к потомкам квадратная ячейка родителя делится на 4 равные части и распределяется между потомками. Уровнем вершины назовем количество ребер в пути от этой вершины до корня дерева. Таким образом, уровень корня равен 0, а уровень его прямых потомков равен 1. В каждой вершине будем хранить список приписанных к ней элементов триангуляции, про которую предполагается, что она принадлежит $[0,\,1]\times[0,\,1]$. Для выбора вершины, к которой должен быть приписан элемент, приме́ним алгоритм 8.

Алгоритм 8. Выбор вершины четверичного дерева

1: Найти середину x и диаметр d сеточного элемента
2: Найти такое наименьшее число n, что $d \geqslant 2^{-n}$
3: Найти вершину четверичного дерева на уровне n, отвечающую за квадрат, содержащий x

Рассмотрим преимущества подобной структуры при поиске элемента треугольной сетки, пересекающегося с некоторым заданным кругом B, лежащим внутри единичного квадрата. Для каждого треугольника Δ_i определим минимальный круг B_i, его содержащий. Тогда если Δ_i пересекает B, то и B_i пересекает B. Очевидно, что B и B_i пересекаются тогда и только тогда, когда расстояние между их центрами ρ_i меньше суммы их радиусов R и R_i. Заметим, что $R_i \leqslant 2^{-n_i}$, где n_i — уровень вершины четверичного дерева, которой приписан треугольник Δ_i. Структура четверичного дерева позволяет очень быстро найти все треугольники, для которых $\rho_i \leqslant R + 2^{-n_i}$, поскольку затраты на поиск одного элемента сетки пропорциональны $\log_4 N_f$ действий. Только для таких треугольников есть смысл проверять их пересечение с кругом B. Если рассматриваемая триангуляция не принадлежит единичному квадрату, то она может быть отображена переносом и масштабированием в единичный квадрат до осуществления каких-либо действий.

Алгоритмы быстрой локализации элементов сетки на основе четверичных и восьмеричных деревьев эффективны для регулярных триангуляций и тетраэдризаций. Эффективность использования этих структур данных для операций с адаптивными сетками обсуждается в гл. 3.

Глава 3

ПОСТРОЕНИЕ НЕСТРУКТУРИРОВАННЫХ СЕТОК В ПРОИЗВОЛЬНЫХ ОБЛАСТЯХ

В данной главе мы опишем два метода построения неструктурированных симплициальных сеток: метод продвигаемого фронта и метод триангуляции Делоне. При построении тетраэдральных сеток каждый метод имеет свои преимущества и недостатки, поэтому наиболее эффективная стратегия построения таких сеток состоит в комбинировании обоих методов. Особое внимание также будет уделено методам улучшения заданной поверхностной сетки, поскольку ее качество сильно влияет на работу методов построения пространственной сетки.

§ 3.1. Способы задания вычислительной области

Для автоматического построения симплициальной (треугольной или тетраэдральной) сетки необходима информация о геометрической модели вычислительной области. В настоящее время применяются несколько подходов для представления геометрических моделей в памяти компьютера. Наиболее универсальный подход используется в системах автоматизации проектных работ (САПР). Геометрическая модель внутри САПР может быть представлена разными способами. Модель может быть составлена из простых фигур — геометрических примитивов, таких, как круг, квадрат, шар, куб, цилиндр. Модель может быть также представлена в виде многоугольника или многогранника, в этом случае задается только граница объекта в виде ломаной на плоскости или набора треугольников в пространстве. Разбиение границы на отрезки или треугольники называют *дискретизацией границы* области. Вместо отрезков могут использоваться кривые линии, а вместо плоских треугольников — криволинейные поверхности. Для хранения геометрической информации о криволинейных составляющих объекта, как правило, используются параметризации

$$\gamma : t \mapsto (x, y, z), \quad \pi : (p, s) \mapsto (x, y, z).$$

Широко распространенным способом задания параметризации в САПР является использование неоднородных рациональных B-сплайнов (NURBS — non-uniform rational B-spline). Возможна комбинация трех подходов: часть модели может быть параметризована с помощью

NURBS, другая часть — задана дискретизацией своей границы, а третья часть определяется примитивами. Посредством операций объединения, пересечения, дополнения и вычитания получается конечная геометрическая модель.

Классический подход к построению симплициальных сеток сводится к первоначальному построению дискретной границы области с последующим построением симплициальной сетки внутри области на основе имеющейся граничной дискретизации. Таким образом, задача построения симплициальных сеток сводится к двум отдельным задачам: построению дискретной границы для заданной геометрической модели и построению симплициальной сетки внутри области с заданной дискретной границей. Задача построения граничной дискретизации может решаться по-разному в зависимости от того, в каком виде задана геометрическая модель. Вообще говоря, на втором этапе информация о геометрической модели может быть недоступна.

При решении второй задачи важным требованием является сохранение следа симплициальной сетки на границе области. Невыполнение этого условия может привести к невозможности проведения расчетов на полученной сетке либо потребовать от пользователя дополнительной работы по обработке полученной сетки. Например, если построенная сетка в дальнейшем будет объединена с другой сеткой, то сохранение следа сетки на их общей границе позволит получить конформную общую сетку. При задании граничных условий в модельной задаче сохранение следа сетки позволит избежать дополнительной переинтерполяции граничных данных.

В некоторых случаях для пользователя может быть допустимо незначительное отклонение от заданного следа на границе. След построенной сетки на границе может быть более мелким, чем заданный изначально. При этом измельчение допускается только с помощью добавления новых узлов на границе и разбиения элементов на более мелкие. В этом случае при необходимости пользователь может восстановить конформность на стыке с соседней сеткой с помощью соответствующего разбиения граничных элементов второй сетки.

С точки зрения автоматизации процесса построения сеток универсальным подходом является точное сохранение следа сетки на границе области. При этом качество полученной симплициальной сетки может быть ограничено качеством граничной дискретизации. Так, в трехмерном случае вытянутые треугольники на границе области значительно ограничивают качество прилегающих к ним тетраэдров. *Для получения тетраэдральных сеток высокого качества необходима поверхностная триангуляция высокого качества.*

В двумерном случае граница области представляет из себя набор из нескольких линий, в общем случае кривых. Дискретизация кривой линии, параметризованной функцией $\gamma : t \mapsto (x, y)$, $t \in [t_0, t_1]$, строится с помощью стандартных методов. Шаг дискретизации вдоль кривой

либо выбирается пользователем, либо вычисляется автоматически, например, на основе оценки локальной кривизны.

В трехмерном случае построение хорошей поверхностной триангуляции является более сложной задачей. В большинстве случаев вся поверхность объекта может быть разбита на несколько частей, каждая из которых является либо частью плоскости, либо частью параметризованной поверхности $\pi : (p, s) \mapsto (x, y, z)$. Границами этих частей в общем случае являются кривые линии в трехмерном пространстве, имеющие собственную параметризацию $\gamma : t \mapsto (x, y, z)$. На первом этапе строится дискретизация кривых границ (линий) в трехмерном пространстве. Далее для каждого куска поверхности строится триангуляция со следом на границе, совпадающим с построенной дискретизацией. Сохранением следа поверхностных сеток разных кусков поверхности на общем криволинейном ребре гарантируется конформность общей поверхностной сетки. Способ построения поверхностной триангуляции с помощью метода продвигаемого фронта будет подробно рассмотрен в § 3.4.

В некоторых случаях геометрическая модель области может быть уже задана в виде поверхностной триангуляции. Качество элементов в поверхностной триангуляции может быть очень низким, что особенно характерно для сеток, полученных с помощью экспорта модели из САПР. Дальнейшее построение тетраэдральной сетки с плохой поверхностной триангуляцией приведет к неизбежному появлению плохих тетраэдров в конечной сетке. В § 3.5 будет рассмотрен подход, позволяющий перестраивать поверхностные сетки с сохранением геометрических особенностей области. Этот же подход может быть использован для получения более мелких или более грубых поверхностных сеток на основе имеющихся.

В качестве альтернативного способа задания геометрической модели может использоваться индикаторная, или характеристическая, функция

$$\chi : (x, y, z) \mapsto \{0, 1\}.$$

Точка находится внутри модели, если $\chi(x, y, z) = 1$. Имеются методики [64] получения многогранной границы области, заданной с помощью индикаторной функции. Как правило, для улучшения качества полученной дискретной границы необходима ее дополнительная обработка.

Выпуклая область может быть также задана облаком точек в пространстве. Часто в этом случае пользователя интересует симплициальная сетка с узлами в заданных точках. Если начальные данные модельной задачи известны и заданы в конкретных точках, то сетка с узлами в этих точках позволит избежать дополнительной переинтерполяции начальных данных. Как правило, в этом случае используют алгоритмы построения триангуляции Делоне, рассматриваемые в следующем параграфе.

§ 3.2. Построение триангуляции Делоне

Рассмотрим задачу построения триангуляции на заданном наборе узлов $\mathcal{V} = \{v_1, \ldots, v_n\}$. Будем говорить, что триангуляция удовлетворят *условию Делоне*, если для любого треугольника внутри описанной вокруг него окружности отсутствуют вершины других треугольников. *Триангуляцией Делоне* назовем выпуклую триангуляцию, удовлетворяющую условию Делоне. Триангуляция Делоне является уникальной, если никакие четыре узла из $\mathcal{V}$ не лежат на одной окружности.

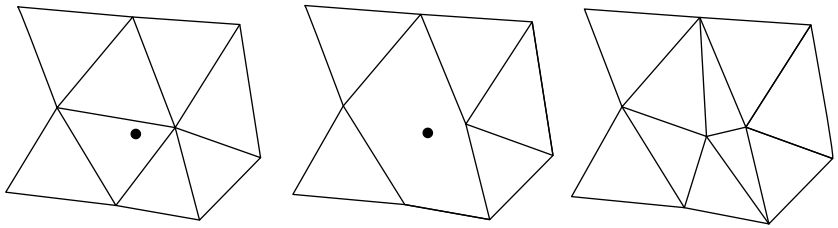
Известны несколько методов построения триангуляции Делоне. Помимо прямых методов построения триангуляции Делоне, ее можно также получить из любой другой триангуляции, последовательно перестраивая смежные пары треугольников, не удовлетворяющих условию Делоне, в пары треугольников, удовлетворяющих этому условию.

В этом параграфе будет кратко рассмотрен простой итерационный алгоритм построения триангуляции Делоне. Задачу триангуляции можно сформулировать следующим образом: пусть в области имеется частично построенная триангуляция, к которой добавляется новый узел; требуется достроить триангуляцию во всей области.

В упомянутом алгоритме сначала определяется положение нового узла относительно имеющейся триангуляции, а затем в зависимости от результата выполняются те или иные действия. Если новый узел попадает на некоторое ребро, то оно разбивается на два ребра и смежные треугольники разбиваются на два меньших. Если новый узел попадает внутрь какого-либо треугольника, то последний разбивается на три меньших треугольника. Если узел не попадает внутрь триангуляции, то находятся граничные ребра текущей триангуляции, с которыми он может образовать новые треугольники. Предложенный алгоритм можно упростить (см. алгоритм 9), если заранее добавить несколько вспомогательных узлов и построить начальную триангуляцию Делоне, целиком покрывающую набор узлов $\mathcal{V}$. В таком случае каждый новый узел будет лежать внутри триангуляции Делоне.

Во всех случаях новые треугольники и соседние с ними проходят проверку на выполнение условия Делоне. При нарушении этого условия производится локальное перестроение триангуляции, которое заключается в следующем. При добавлении нового узла находятся и удаляются все треугольники, для которых нарушается условие Делоне, и внутри образовавшегося многоугольника строятся новые треугольники с участием нового узла (см. рис. 3.1). Полученная триангуляция будет удовлетворять условию Делоне [18].

При построении триангуляции Делоне для заданной двумерной области необходимо определить набор узлов $\mathcal{V}$, лежащих внутри и на границе этой области; для этого набора и строится триангуляция Делоне. При этом в качестве итерационного алгоритма для добавления нового узла внутри области может быть использован следующий простой алгоритм: выберем самое длинное ребро в текущей триангуляции



а *б* *в*

Рис. 3.1. Построение триангуляции Делоне итерационным алгоритмом: *а* — добавление нового узла, *б* — удаление треугольников, *в* — построение новых треугольников

Алгоритм 9. Построение триангуляции Делоне

1: Добавить три вспомогательных узла, образующих объемлющий треугольник множества $\mathcal{V}$
2: **for all** узлов $v \in \mathcal{V}$ **do**
3: Добавить узел v в триангуляцию Ω_h
4: Найти множество треугольников Σ, для которых нарушается условие Делоне, и удалить их из Ω_h
5: Построить новые треугольники, соединяя v с ребрами границы $\partial\Sigma$
6: **end for**
7: Удалить вспомогательные узлы и соответствующие им треугольники

и поставим новый узел в его середину. Для контроля за равномерностью и плотностью расположения узлов в области потребуются более сложные методы.

Предложенный алгоритм построения триангуляции Делоне строит триангуляцию для выпуклой оболочки множества узлов. Если искомая область невыпукла, то необходимо либо использовать методы построения триангуляций Делоне с ограничениями [18], либо искусственно доразбивать триангуляцию, добавляя новые узлы на границу области, а потом удалять треугольники, лежащие вне искомой области. При использовании второго подхода след сетки для выпуклых областей сохраняется, а для невыпуклых областей он может быть измельчен.

Условие Делоне естественно обобщается на тетраэдризации в трехмерном пространстве. *Тетраэдризацией Делоне* для конечного множества узлов $\mathcal{V} = \{v_1, \ldots, v_n\}$ называется такое конформное разбиение выпуклой оболочки множества $\mathcal{V}$ на тетраэдры, что для любого тетраэдра внутри описанной вокруг него сферы отсутствуют другие узлы из $\mathcal{V}$. Простой итерационный алгоритм построения триангуляции Делоне может быть использован и при построении тетраэдризаций Делоне в трехмерном пространстве [54]. В отличие от двумерных треугольных сеток, в трехмерном случае существуют такие многогранные

области, для которых невозможно построить конформную тетраэдральную сетку с заданным следом на границе без добавления новых узлов. Пример такой области приведен в § 2.2 на рис. 2.6. Известны несколько методов для сохранения границы невыпуклой области в трехмерном пространстве: локальные модификации сетки [34], измельчение сетки [42] и построение тетраэдризации Делоне с ограничениями [81].

Отличительной особенностью тетраэдризации Делоне является наличие так называемых *сливеров* — сильно вырожденных тетраэдров, у которых вершины лежат практически в одной плоскости и практически на одной окружности. Для таких тетраэдров может выполняться условие Делоне, но при этом сами тетраэдры будут иметь очень плохое качество. Для улучшения качества сетки используются локальные передвижения узлов.

На практике при реализации алгоритмов построения триангуляции Делоне необходимо особое внимание уделять структурам данных и алгоритмам быстрого поиска треугольников или тетраэдров вблизи заданной точки. Хороший обзор структур данных и алгоритмов представлен в книге [18] и в § 2.3. Отдельного внимания заслуживает также точность вычислений при проверке выполнения условия Делоне. Некоторые из этих вопросов будут рассмотрены в следующем параграфе.

§ 3.3. Построение триангуляции методом продвигаемого фронта

В этом параграфе будет предложен алгоритм, строящий конформные треугольные сетки для двумерных областей, заданных дискретизациями своих границ. При построении сетки алгоритм сохраняет заданный след на границе. Предложенный алгоритм применим как к простым многоугольникам, так и к произвольным многокомпонентным многосвязным многоугольным областям, границы которых могут не быть одномерными многообразиями. Вопрос построения дискретной границы для произвольной области с криволинейными границами здесь обсуждаться не будет, однако мы вернемся к нему в § 3.4.

Ниже мы рассмотрим классический алгоритм продвигаемого фронта, исследуем влияние на него вычислительных погрешностей, а также конечность и время работы алгоритма. Нам понадобятся следующие понятия и обозначения.

Положительной полуплоскостью относительно направленного ребра $\mathbf{e}_{12} = \mathbf{v}_2 - \mathbf{v}_1$ будем называть множество точек $\mathbf{v}$, для которых алгебраическая площадь положительна [см. формулу (3.3.1)]. Используя тензорное произведение векторов, удобно записать алгебраическую площадь в виде векторного произведения:

$$S_{\Delta(v_1,v_2,v)} \equiv \frac{1}{2}\,(\mathbf{v}_1 - \mathbf{v}) \times (\mathbf{v}_2 - \mathbf{v}) > 0.$$

Аналогично, *отрицательной полуплоскостью* будем называть множество точек, для которых $S_{\Delta(v_1,v_2,v)} < 0$.

Ломаной на плоскости назовем конечный набор отрезков, последовательно соединенных своими концами. При этом не исключается расположение двух последовательных отрезков ломаной на одной прямой. Если первая и последняя точки ломаной совпадают, то такую ломаную будем называть *замкнутой*. Ломаная, отрезки которой не пересекаются друг с другом, будем называть ломаной *без самопересечений*.

Многоугольником на плоскости назовем ограниченную открытую часть плоскости, граница которой состоит из одной или нескольких непересекающихся ломаных без самопересечений. *Простым многоугольником* назовем многоугольник, ограниченный одной замкнутой ломаной. Вершинами многоугольника будем называть вершины ломаных, а его сторонами — отрезки ломаных. На рис. 3.2 иллюстрируются несколько видов многоугольников.

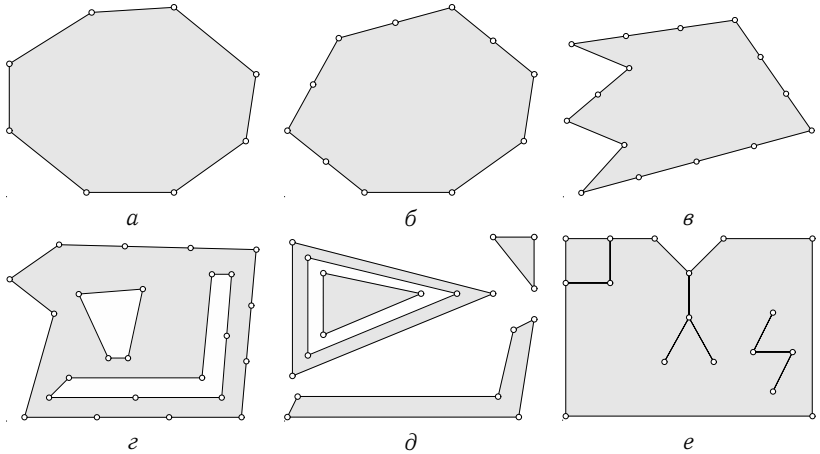


Рис. 3.2. Виды многоугольников: *а* — строго выпуклый, *б* — нестрого выпуклый, *в* — невыпуклый, *г* — многосвязный, *д* — многокомпонентный, *е* — с внутренними разрезами

Стороны многоугольника можно подразделить на два типа: внешние и внутренние. Пусть v_1v_2 — сторона многоугольника $\mathcal{P}$, v — середина v_1v_2 и $\omega_\varepsilon(v)$ — ε-окрестность точки v. Рассмотрим $\mathcal{P}_\varepsilon = \mathcal{P} \cap \omega_\varepsilon(v)$. Если при любом $\varepsilon > 0$ множество $\mathcal{P}_\varepsilon$ лежит по обе стороны от отрезка v_1v_2, то будем называть v_1v_2 *внутренней* стороной, или *разрезом*. Если при некотором $\varepsilon > 0$ часть многоугольника $\mathcal{P}_\varepsilon$ лежит только по одну сторону от отрезка v_1v_2, то v_1v_2 будем называть *внешней* стороной многоугольника $\mathcal{P}$. В последнем случае на ребре e_{12} можно ввести направление так, чтобы часть многоугольника $\mathcal{P}_\varepsilon$ оказалась в положительной полуплоскости относительно v_1v_2. Будем называть это направление направлением обхода многоугольника *против часовой стрелки*. На рис. 3.3 показаны направления обхода внешних сторон нескольких многоугольников. Заметим, что совокупность всех внешних сторон образует одну или несколько замкнутых ломаных.

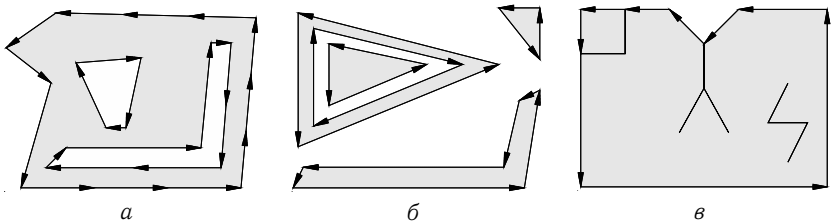


Рис. 3.3. Обход многоугольника против часовой стрелки: *а* — многосвязный многоугольник, *б* — многокомпонентный многоугольник, *в* — многоугольник с внутренними разрезами

Фронтом на плоскости будем называть набор направленных ребер без самопересечений. Каждому многоугольнику $\mathcal{P}$ можно поставить в соответствие фронт $\mathcal{F}(\mathcal{P})$. Для этого введем на внешних сторонах многоугольника направление обхода против часовой стрелки, а каждой внутренней стороне v_1v_2 поставим в соответствие пару направленных ребер e_{12} и e_{21}. Совокупность всех этих направленных ребер и образует фронт $\mathcal{F}(\mathcal{P})$. Будем называть фронт $\mathcal{F}$ *замкнутым*, если существует такой многоугольник $\mathcal{P}$, что $\mathcal{F} = \mathcal{F}(\mathcal{P})$.

3.3.1. Алгоритм продвигаемого фронта

В этом пункте мы рассмотрим алгоритм продвигаемого фронта для построения в заданной многоугольной области $\mathcal{P}$ конформной треугольной сетки, согласованной с границей $\mathcal{P}$.

Текущим фронтом $\mathcal{F}^k$ назовем некоторый замкнутый фронт — границу многоугольной области $\mathcal{P}^k$, в которой нужно построить треугольную сетку. Символом $\mathcal{T}^k$ будем обозначать множество построенных треугольников. В начальный момент времени мы положим $\mathcal{P}^0 = \mathcal{P}$, $\mathcal{F}^0 = \mathcal{F}(\mathcal{P}^0)$ и $\mathcal{T}^0 = \varnothing$. Далее, на k-м шаге будем строить новый треугольник $\Delta_{123} \subset \mathcal{P}^k$, добавлять его в треугольную сетку и вычитать из области, в которой сетка еще не построена:

$$\mathcal{T}^{k+1} = \mathcal{T}^k \cup \{\Delta_{123}\}, \quad \mathcal{P}^{k+1} = \mathcal{P}^k \setminus \Delta_{123}.$$

Когда $\mathcal{F}^{k+1} = \mathcal{F}(\mathcal{P}^{k+1})$ станет пустым множеством, или, что то же самое, $\mathcal{P}^{k+1} = \varnothing$, можно утверждать, что множество треугольников $\mathcal{T} = \mathcal{T}^{k+1}$ полностью покрывает $\mathcal{P}$.

Напомним, что согласно нашим обозначениям Δ_{123} обозначает треугольник с вершинами v_1, v_2 и v_3, а e_{12} — ребро с вершинами v_1 и v_2. Треугольник Δ_{123} будем строить так, что одна из его сторон будет одним из отрезков текущего фронта $\mathcal{F}^k$. Тогда после вычитания из многоугольной области нового треугольника текущий фронт изменится

незначительно: как минимум один отрезок из фронта будет удален и не более чем два новых будут добавлены.

Рассмотрим некоторое направленное ребро e_{12} текущего фронта. Для того, чтобы Δ_{123} лежал внутри $\mathcal{P}^k$, достаточно выполнения двух условий:

1) алгебраическая площадь Δ_{123} положительна, т. е. $S_{\Delta_{123}} > 0$;
2) треугольник Δ_{123} не пересекает фронт $\mathcal{F}^k$.

При этом допускается, чтобы вершина v_3 была вершиной фронта, а стороны треугольника совпадали с отрезками $\mathcal{F}^k$.

Будем называть треугольник Δ_{123}, удовлетворяющий этим условиям, *подходящим*. Такой выбор Δ_{123} гарантирует, что после добавления его в сетку она сохранит свойство конформности. При этом будет обеспечена согласованность конечной сетки $\mathcal{T}$ с начальной областью $\mathcal{P}$.

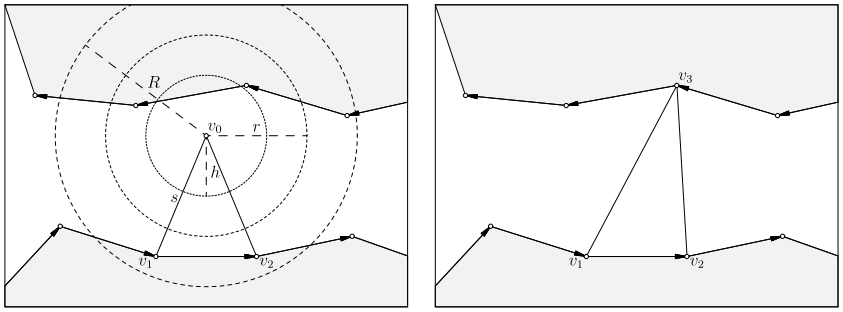
Для определенности мы будем выбирать на каждом шаге направленное ребро e_{12} с наименьшей длиной и строить на нем новый треугольник Δ_{123}. При построении Δ_{123} выбор третьей вершины v_3 не является однозначным. Мы можем руководствоваться как желаемым размером треугольника в этой области, так и какими-то другими эвристическими критериями. Вершину v_3 мы будем выбирать только из некоторого конечного множества Σ вершин-кандидатов. В это множество включим точку v_0, равноудаленную от точек v_1 и v_2 на некоторое расстояние s, а также добавим все вершины фронта, лежащие в положительной полуплоскости относительно направленного ребра e_{12}.

Так как множество Σ конечно, то мы можем просто перебирать все треугольники Δ_{123}, где $v_3 \in \Sigma$, до тех пор, пока очередной из них не окажется подходящим (см. рис. 3.4, *б*). В п. 3.3.3 будет доказано, что хотя бы один подходящий треугольник всегда найдется.

Сложность такого алгоритма очень большая, и на практике он не будет эффективным. Поэтому мы рассмотрим некоторую выпуклую открытую окрестность ω, содержащую точки v_1, v_2 и v_0 (см. рис. 3.4). В качестве области ω можно выбрать круг радиуса $R > s$ с центром в v_0. Выберем из текущего фронта только те отрезки, которые имеют хотя бы одну общую точку с $\overline{\omega}$. Получившийся фронт может оказаться незамкнутым. Назовем его *локальным фронтом* и обозначим $\mathcal{F}_\omega$. Любая подобласть ω пересекается с текущим фронтом тогда и только тогда, когда она пересекается с локальным фронтом.

Построим множество $\Sigma_\omega = \Sigma \cap \omega$. В силу выпуклости ω любой треугольник Δ_{123}, где $v_3 \in \Sigma_\omega$, будет лежать в ω. Для проверки пересечения этого треугольника с текущим фронтом достаточно проверить его пересечение с локальным фронтом. Локальный перебор будет быстрее, но уже не будет гарантировать существование подходящего треугольника. Поэтому надо предусмотреть возможность вернуться к полному перебору в случае, когда локальный перебор закончится неудачно.

Для доказательства конечности числа операций при работе алгоритма нужно как-то оценить максимальное количество треугольников,



а *б*

Рис. 3.4. Продвижение фронта: *а* — окрестности вершины-кандидата, соответствующие разным значениям R; *б* — добавление подходящего треугольника в сетку

которое мы с его помощью построим. Для этого нужно ввести какие-то ограничения на сами треугольники, на ребра и на вершины сетки. С практической точки зрения наиболее удобно ограничение снизу на минимальное попарное расстояние между узлами сетки. Для поддержания этого ограничения мы будем исключать из Σ точку v_0, если она лежит близко к вершинам фронта или к уже построенной сетке. Для этого мы введем два параметра: h — ограничение на минимальное расстояние до текущего фронта; r — ограничение на расстояние до вершин фронта (см. рис. 3.4, *а*). Если текущий фронт пересекает h-окрестность точки v_0 либо вершины фронта лежат в r-окрестности точки v_0, то исключим точку v_0 из множества вершин-кандидатов. Отметим, что на рис. 3.4 оба условия нарушаются, поэтому третья вершина v_3 является вершиной фронта. Более подробно о работе с этими параметрами будет рассказано в п. 3.3.3.

С учетом всех предыдущих замечаний составим алгоритм 10. Будем использовать следующие дополнительные обозначения. На k-м шаге алгоритма обозначим множество вершин фронта $\mathcal{F}^k$ через $\{p_i^k\}$, а множество узлов построенной сетки $\mathcal{T}^k$ через $\{v_i^k\}$. Для некоторого $R > 0$ и точки v обозначим через $\omega_R(v)$ открытую выпуклую R-окрестность точки v.

Далее мы изучим устойчивость этого алгоритма к вычислительным погрешностям, его конечность и сложность.

3.3.2. Влияние вычислительных погрешностей

При реализации алгоритма 10 на ЭВМ особое внимание стоит уделить вычислительным погрешностям, возникающим при работе с вещественными числами. Использование специализированных библиотек для вычислений с произвольной точностью может решить проблему за счет увеличения времени работы программы. В этом пункте мы проанализируем влияние вычислительных погрешностей на работу

Алгоритм 10. Метод продвигаемого фронта в двумерном пространстве

1: Положить $\mathcal{T}^0 = \varnothing$ и $\mathcal{F}^0 = \mathcal{F}(\mathcal{P})$
2: **for** $k = 0, 1, \ldots$ **do**
3: Выбрать $e_{12} \in \mathcal{F}^k$ с минимальной длиной
4: Определить желаемую длину s стороны треугольника
5: Построить вершину v_0: $|e_{10}| = |e_{20}| = s$, $S_{\Delta_{120}} > 0$
6: Выбрать некоторые $R > s$, $h > 0$ и $r > 0$
7: Построить локальный фронт $\mathcal{F}_R^k$: $\mathcal{F}_R^k \supseteq \mathcal{F}^k \cap \omega_R(v_0)$
8: Определить множество вершин-кандидатов $\Sigma_R^k = \{p_i^k\} \cap \cap\, \omega_R(v_0)$
9: Если $\{p_i^k\} \cap \omega_r(v_0) = \varnothing$ и $\mathcal{F}^k \cap \omega_h(v_0) = \varnothing$, то добавить v_0 к Σ_R^k
10: **for all** $v_3 \in \Sigma_R^k$ **do**
11: **if** Δ_{123} не пересекает $\mathcal{F}_R^k$ **then**
12: Добавить треугольник Δ_{123} в сетку: $\mathcal{T}^{k+1} = \mathcal{T}^k \cup \cup\, \{\Delta_{123}\}$
13: Обновить фронт: $\mathcal{P}^{k+1} = \mathcal{P}^k \setminus \Delta_{123}$, $\mathcal{F}^{k+1} = \mathcal{F}(\mathcal{P}^{k+1})$
14: Если $\mathcal{F}^{k+1} = \varnothing$, то перейти на шаг 20
15: Перейти на шаг 19
16: **end if**
17: **end for**
18: Положить $R = \infty$, перейти к 7
19: **end for**
20: Положить $\mathcal{T} = \mathcal{T}^{k+1}$

алгоритма, а также предложим несколько идей, которые позволят свести это влияние к минимуму.

Одной из основных операций, чувствительных к вычислительным погрешностям, является проверка пересечения треугольника с отрезком. Неправильное определение пересечения из-за вычислительных погрешностей может привести к неправильной работе всего алгоритма. Отметим, что возможны только две ситуации получения неправильного результата:

1) непересекающиеся треугольник и отрезок неверно восприняты как пересекающиеся;

2) пересекающиеся треугольник и отрезок неверно восприняты как непересекающиеся.

С точки зрения алгоритма 10 ошибки первого типа некритичны: ложное пересечение треугольника и отрезка означает, что новый треугольник не пройдет проверку пересечения с текущим фронтом, и соответствующая вершина-кандидат будет забракована. Ошибки же второго типа могут исказить проверку пересечения треугольника с текущим фронтом, что может привести к самопересекающемуся фронту или к неконформной сетке. Таким образом, ошибки первого типа лишь увеличивают перебор вершин-кандидатов, незначительно замедляя работу

программы, а ошибки второго типа могут привести к неправильному результату работы программы.

Мы будем использовать алгоритм проверки пересечения пары треугольник–отрезок, исключающий возникновение ошибок второго типа, но допускающий возникновение ошибок первого типа. В общем случае два плоских выпуклых объекта не пересекаются тогда и только тогда, когда существует разделяющая их прямая. В условиях алгоритма 10 также допускаются общие вершины и общие ребра: например, отрезок и треугольник могут иметь общую вершину либо отрезок может быть стороной треугольника. В этих случаях конформность сетки нарушаться не будет, поэтому их нужно рассматривать как непересекающиеся.

Рассмотрим треугольник Δ_{123} и ребро e_{45} (см. рис. 3.5). Будем предполагать, что

$$S_{\Delta_{123}} > 0,$$

так как только такие треугольники будут тестироваться в алгоритме 10. Сначала определим, сколько общих вершин у треугольника и отрезка.

Рассмотрим случай, когда общих вершин нет. Треугольник и отрезок не пересекаются тогда и только тогда, когда либо треугольник лежит целиком в одной из полуплоскостей относительно прямой e_{45}, либо e_{45} лежит по другую сторону от треугольника относительно одной из его сторон (см. рис. 3.5). Таким образом, если пересечений нет, то выполняется хотя бы одно из следующих условий:

1) $S_{\Delta_{451}} > 0$, $S_{\Delta_{452}} > 0$, $S_{\Delta_{453}} > 0$, либо
   $S_{\Delta_{451}} < 0$, $S_{\Delta_{452}} < 0$, $S_{\Delta_{453}} < 0$;
2) $S_{\Delta_{124}} < 0$, $S_{\Delta_{125}} < 0$;
3) $S_{\Delta_{234}} < 0$, $S_{\Delta_{235}} < 0$;
4) $S_{\Delta_{314}} < 0$, $S_{\Delta_{315}} < 0$.

Верно и обратное: если выполняется одно из этих условий, то пересечений нет. Эти условия можно записать в более удобной форме:

1) $\operatorname{sign}(S_{\Delta_{451}}) + \operatorname{sign}(S_{\Delta_{452}}) + \operatorname{sign}(S_{\Delta_{453}}) = \pm 3$;
2) $\operatorname{sign}(S_{\Delta_{124}}) + \operatorname{sign}(S_{\Delta_{125}}) = -2$;
3) $\operatorname{sign}(S_{\Delta_{234}}) + \operatorname{sign}(S_{\Delta_{235}}) = -2$;
4) $\operatorname{sign}(S_{\Delta_{314}}) + \operatorname{sign}(S_{\Delta_{315}}) = -2$.

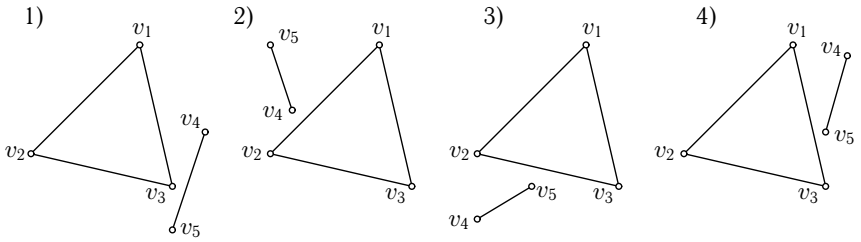


Рис. 3.5. Иллюстрация условий 1), 2), 3) и 4), когда треугольник и ребро не пересекаются

Перейдем к случаю, когда у треугольника и отрезка есть одна общая вершина. В этом случае треугольник и отрезок не пересекаются в нашем смысле тогда и только тогда, когда e_{45} лежит в отрицательной полуплоскости относительно одной из сторон треугольника, при этом их общая вершина лежит на прямой, а вторая попадает в отрицательную полуплоскость. Эти условия можно записать в следующем виде:

1) $\operatorname{sign}(S_{\Delta_{124}}) + \operatorname{sign}(S_{\Delta_{125}}) = -1$;
2) $\operatorname{sign}(S_{\Delta_{234}}) + \operatorname{sign}(S_{\Delta_{235}}) = -1$;
3) $\operatorname{sign}(S_{\Delta_{314}}) + \operatorname{sign}(S_{\Delta_{315}}) = -1$.

Если выполняется хотя бы одно из этих условий, то пересечения нет; и наоборот, если нет пересечения, то выполняется хотя бы одно из условий.

В последнем случае, когда обе вершины отрезка совпадают с двумя вершинами треугольника, ребро e_{45} является стороной треугольника, и по нашей договоренности треугольник и отрезок считаются непересекающимися.

Предположим, что у нас есть функция $d(v_1, v_2, v_3)$, неточно вычисляющая знак выражения $S_{\Delta_{123}}$. Но при этом если $d(v_1, v_2, v_3) \neq 0$, то $d(v_1, v_2, v_3) = \operatorname{sign}(S_{\Delta_{123}})$, а в спорных ситуациях $d(v_1, v_2, v_3) = 0$. Тогда проверку на пересечение можно выполнять с помощью алгоритма 11.

Алгоритм 11. Проверка пересечения треугольника с отрезком

1: Найти общие вершины у треугольника Δ_{123} и ребра e_{45}
2: **if** общих вершин нет **then**
3: Если $d(v_4, v_5, v_1) + d(v_4, v_5, v_2) + d(v_4, v_5, v_3) = \pm 3$, то пересечения нет
4: Если $d(v_1, v_2, v_4) + d(v_1, v_2, v_5) = -2$, то пересечения нет
5: Если $d(v_2, v_3, v_4) + d(v_2, v_3, v_5) = -2$, то пересечения нет
6: Если $d(v_3, v_1, v_4) + d(v_3, v_1, v_5) = -2$, то пересечения нет
7: **end if**
8: **if** одна общая вершина **then**
9: Если $d(v_1, v_2, v_4) + d(v_1, v_2, v_5) = -1$, то пересечения нет
10: Если $d(v_2, v_3, v_4) + d(v_2, v_3, v_5) = -1$, то пересечения нет
11: Если $d(v_3, v_1, v_4) + d(v_3, v_1, v_5) = -1$, то пересечения нет
12: **end if**
13: **if** две общих вершины **then**
14: Пересечения нет
15: **end if**
16: В остальных случаях считаем, что треугольник и отрезок пересекаются

Отметим, что допустимая неточность вычисления $d(v_1, v_2, v_3)$ может привести только к ошибкам первого типа, а ошибки второго типа алгоритм 11 исключает.

Операция определения знака алгебраической площади $S_{\Delta_{123}}$ является основной операцией, используемой как в алгоритме 10, так и в алгоритме 11. Предложим способ неточного определения знака $d(v_1, v_2, v_3)$ со следующим свойством: если $d(v_1, v_2, v_3) \neq 0$, то $d(v_1, v_2, v_3) = \operatorname{sign}(S_{\Delta_{123}})$.

Вычисление $S_{\Delta_{123}}$ сводится к нахождению детерминанта матрицы 2×2:

$$2S_{\Delta_{123}} = \mathbf{e}_{31} \times \mathbf{e}_{32} = \begin{vmatrix} x_1 - x_3 & y_1 - y_3 \\ x_2 - x_3 & y_2 - y_3 \end{vmatrix} \equiv \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc = D. \quad (3.3.1)$$

При вычислении элементов матрицы мы уже вносим некоторую относительную погрешность. В дополнение к этому в выражениях ad и bc опять вносится относительная погрешность, и в конце вносится еще одна погрешность в D. Суммарную абсолютную погрешность можно оценить сверху как $\varepsilon(|a| + |b| + |c| + |d| + 2|ad| + 2|bc|)$, где ε зависит от машинной точности. С учетом этой допустимой погрешности запишем алгоритм для определения $\operatorname{sign}(S_{\Delta_{123}})$.

Алгоритм 12. Вычисление $d(v_1, v_2, v_3)$

1: Вычислить $a = x_1 - x_3$, $b = y_1 - y_3$, $c = x_2 - x_3$, $d = y_2 - y_3$
2: Оценить погрешность $r = \varepsilon(|a| + |b| + |c| + |d| + 2|ad| + 2|bc|)$
3: Вычислить детерминант матрицы $D = ad - bc$
4: Если $D > r$, то **return** 1
5: Если $D < -r$, то **return** -1, иначе **return** 0

В алгоритме 12 величина D вычисляется с абсолютной погрешностью, не превосходящей r. Поэтому если $d(v_1, v_2, v_3) \neq 0$, то можно быть уверенным, что знак D вычислен правильно и $d(v_1, v_2, v_3) = \operatorname{sign}(S_{\Delta_{123}})$.

На практике если для хранения координат в памяти ЭВМ используется тип вещественных чисел с одинарной точностью, а вычисления величин a, b, c, d, D в алгоритме 12 производятся с двойной точностью, то машинной точности хватит для точного вычисления как всех промежуточных величин, так и окончательной величины D. В этом случае можно положить $\varepsilon = 0$, и алгоритм 12 будет вычислять $\operatorname{sign}(S_{\Delta_{123}})$ точно.

3.3.3. Конечность работы алгоритма 10

Покажем конечность числа операций в алгоритме 10. В предложенном алгоритме имеются два явных вложенных цикла и один неявный за счет перезапуска перебора на шаге 18. Цикл перебора на шаге 10 всегда конечен в силу конечности множества Σ_R^k. Необходимо удостовериться, что после перезапуска перебора подходящий треугольник всегда будет найден, а также показать, что внешний цикл на шаге 2 будет конечным.

Докажем, что при поиске вершины-кандидата v_3 на шаге 10 всегда найдется хотя бы один подходящий треугольник после перезапуска перебора на шаге 18. Из теоремы Радо [80] следует, что любой многоугольник можно разбить диагоналями на треугольники. Из существования триангуляции без дополнительных точек следует, что при каждой стороне многоугольника можно построить хотя бы один треугольник. Сформулируем это утверждение.

Лемма 3.3.1. *Пусть $\mathcal{P}$ — многоугольник с вершинами $v_1, v_2, \ldots, v_n$ и v_1v_2 — сторона многоугольника $\mathcal{P}$. Тогда существует такая третья вершина многоугольника $\widetilde{v}_3$, что треугольник $\Delta(v_1, v_2, \widetilde{v}_3)$ целиком лежит внутри $\mathcal{P}$.*

Если алгоритм 10 доходит до шага 18, то перебор на шаге 10 становится полным. Из леммы 3.3.1, с учетом предыдущих замечаний о влиянии вычислительных погрешностей и возможности точного вычисления в алгоритме 12, следует, что после перезапуска перебора подходящий треугольник всегда будет найден.

Покажем, что внешний цикл на шаге 2 будет конечным, т. е. количество построенных треугольников конечно. Для этого мы оценим количество треугольников через количество узлов сетки, а количество узлов ограничим сверху за счет ограниченности многоугольника $\mathcal{P}$ и ограничения снизу на расстояние между узлами v_i^k.

Пусть d_k — минимальное из попарных расстояний между точками фронта p_i^k, ρ_k — минимальное из попарных расстояний между узлами из v_i^k, $\rho_0 = \infty$. При добавлении нового треугольника Δ_{123} в $\mathcal{T}^k$ возможны две ситуации: $v_3 \in \{p_i^k\}$ и $v_3 = v_0$. В первом случае минимальное расстояние между вершинами сетки не уменьшится больше, чем до минимального расстояния между вершинами фронта, тогда минимальное расстояние между вершинами фронта не уменьшится: $\rho_{k+1} \geqslant \min(\rho_k, d_k)$, $d_{k+1} \geqslant d_k$. Рассмотрим случай $v_3 = v_0$. Так как Δ_{120} не пересекает $\mathcal{F}^k$, то расстояние от v_0 до множества $\{v_i^k\}$ не меньше расстояния от v_0 до $\mathcal{F}^k$, которое по построению не меньше h, выбранного на шаге 6. Расстояние от v_0 до $\{p_i^k\}$ не меньше r. Соответственно, $\rho_{k+1} \geqslant \min(\rho_k, h)$, $d_{k+1} \geqslant \min(d_k, r)$. Предложим некоторый эвристический способ выбора параметров s, r и h. Обозначим через $l = |v_1v_2|$ длину отрезка v_1v_2, а через h_c высоту Δ_{120}, опущенную из вершины v_0 (см. рис. 3.6).

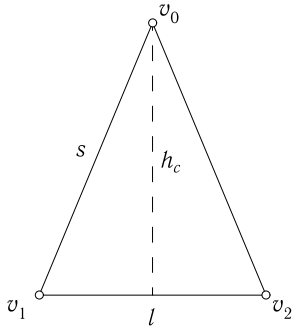


Рис. 3.6. Треугольник с диаметром s, основанием l и высотой h_c

Предположим, что у нас есть некоторая скалярная функция $s(x, y)$: $\mathcal{P} \to \mathbb{R}^+$, и мы хотим, чтобы размер элементов в сетке менялся в соответствии с этой функцией. В этом случае будем говорить, что размер треугольников

задается *пользовательской функцией* $s(x, y)$. Потребуем, чтобы функция $s(x, y)$ была отделена от нуля на $\mathcal{P}$: $s(x, y) > s_{\min} > 0$. Возьмем

$$s = \max\left\{\frac{5}{9}\,l,\ s(x, y)\right\}.$$

Такой выбор гарантирует существование точки v_0 и ограничивает высоту треугольника снизу: $h_c \geqslant \dfrac{\sqrt{19}}{18}\,l$, $h_c \geqslant \dfrac{\sqrt{19}}{10}\,s$.

Если соответствующая функция $s(x, y)$ не задана, то будем определять желаемую длину треугольника на основе длины его основания и некоторого параметра γ, отвечающего за скорость увеличения размера треугольников:

$$s = \gamma l, \qquad \gamma \geqslant 1.$$

В этом случае будем говорить, что шаг сетки выбирается *автоматически*. Так как $\gamma > \dfrac{1}{2}$, то точка v_0 всегда существует, а $h_c = \sqrt{\gamma^2 - \dfrac{1}{4}}\,l \geqslant \dfrac{\sqrt{3}}{2}\,l$.

Возьмем параметр $h = \dfrac{1}{2}\,h_c$, а параметр r вычислим по формуле

$$r = \frac{\beta}{2}\left(s + s_0 + \sqrt{(s - s_0)^2 + \alpha^2}\right), \tag{3.3.2}$$

где $\beta > 0$, $s_0 \geqslant 0$ и α — некоторые параметры. Отметим, что при $s > s_0$ величина $r \geqslant \beta s$, а при $s < s_0$ величина $r \geqslant \beta s_0$. Параметр α можно подобрать так, чтобы при $s = s_0$ выполнялось $r = s_0$. Ограничение $r \geqslant \beta s$ позволяет на практике избегать появления резких перепадов размеров соседних отрезков в новом фронте, а ограничение $r \geqslant \beta s_0$ позволяет ограничить снизу величину r.

При реализации алгоритма на ЭВМ параметры, введенные выше, можно выбирать одним из двух способов в зависимости от того, задается ли размер треугольников пользователем или используется автоматический выбор размера. Рассмотрим оба случая.

Пусть размер треугольников задается пользователем. Положим

$$\beta = \frac{1}{2}, \quad s_0 = 0, \quad \alpha = 0.$$

Тогда $r = \dfrac{1}{2}\,s$. По нашему предположению $s > s_{\min}$, следовательно, $r > \dfrac{1}{2}\,s_{\min}$. Тогда $d_k \geqslant \min\left\{d_0, \dfrac{1}{2}\,s_{\min}\right\}$ для любого k. По построению $h \geqslant \dfrac{\sqrt{19}}{20}\,s$ и $s > s_{\min}$, поэтому для любого k

$$\rho_k \geqslant \min\left\{d_0, \frac{\sqrt{19}}{20}\,s_{\min}\right\}.$$

Пусть теперь размер треугольников выбирается автоматически. Возьмем

$$\beta = \frac{1}{2}, \quad s_0 = d_0, \quad \alpha = 2s_0(1-\beta)/\beta.$$

Тогда $r \geqslant \frac{1}{2} d_0$, т. е. $d_k > \frac{1}{2} d_0$ для любого k. По построению $h \geqslant \frac{\sqrt{3}}{4} l$, а $l \geqslant d_k$, поэтому $h \geqslant \frac{\sqrt{3}}{8} d_0$. Следовательно, для любого k

$$\rho_k \geqslant \frac{\sqrt{3}}{8} d_0.$$

Мы только что показали, что существует такое $\delta > 0$, что для любого k верно $\rho_k > \delta$. С помощью этого условия мы можем оценить максимальное количество узлов сетки. Множество узлов сетки $\{v_i^k\}$ лежит в замыкании $\mathcal{P}$, и попарное расстояние между ними больше δ. Пусть N_v^k — количество узлов во множестве $\{v_i^k\}$. Построим в каждом узле круг радиусом $\frac{\delta}{2}$. Эти круги не пересекаются и покрывают площадь, равную $N_v^k \pi \frac{\delta^2}{4}$. Расширим исходный многоугольник $\mathcal{P}$ на $\delta/2$, $\mathcal{P}_\delta = \left\{ v_3 \colon \operatorname{dist}(v_3, \mathcal{P}) < \frac{\delta}{2} \right\}$ (см. рис. 3.7). Тогда $\mathcal{P}_\delta$ полностью покроет круги.

Оценим площадь $\mathcal{P}_\delta$. У каждой стороны многоугольника длиной l появляется прямоугольник, который дает прирост площади не больше, чем $\frac{\delta}{2} l$. Таким образом, прирост площади за счет прямоугольников составит $\frac{\delta}{2} p(\mathcal{P})$, где $p(\mathcal{P})$ — периметр многоугольника $\mathcal{P}$. Круговые секторы у вершин с внутренним углом α дают прирост площади не больше, чем $(\pi - \alpha)\frac{\delta^2}{8}$, причем эта оценка верна и для внутренних углов, бóльших π; в этом случае прирост становится отрицательным, но он полностью покрывается наложением двух прямоугольников, которые мы учли ранее. Из формулы на сумму углов многоугольника можно оценить вклад всех круговых секторов как $\pi \frac{\delta^2}{4} K(\mathcal{P})$, где $K(\mathcal{P})$ — количество компонент связности в $\mathcal{P}$. Имеем $S(\mathcal{P}_\delta) \leqslant S(\mathcal{P}) + \frac{\delta}{2} p(\mathcal{P}) + \pi \frac{\delta^2}{4} K(\mathcal{P})$. Отсюда можно оценить количество вершин:

$$N_v^k \leqslant \frac{4S(\mathcal{P})}{\pi\delta^2} + \frac{2p(\mathcal{P})}{\pi\delta} + K(\mathcal{P}).$$

Заметим, что аналогичная оценка верна и для количества вершин во фронте $\mathcal{F}^k$:

$$N_p^k \leqslant \frac{4S(\mathcal{P})}{\pi\delta^2} + \frac{2p(\mathcal{P})}{\pi\delta} + K(\mathcal{P}). \tag{3.3.3}$$

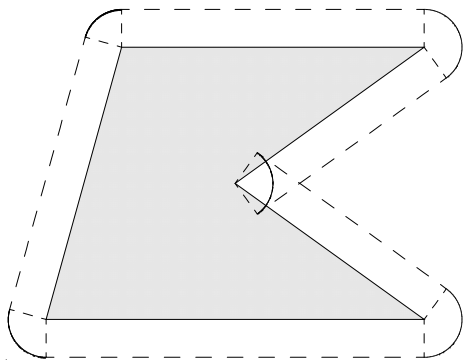


Рис. 3.7. Расширение многоугольника на $\frac{\delta}{2}$

Из (2.2.5) следует, что количество треугольников в $\mathcal{T}^k$ ограничено:

$$N_f^k \leqslant \frac{8S(\mathcal{P})}{\pi\delta^2} + \frac{4p(\mathcal{P})}{\pi\delta} + 2K(\mathcal{P}) - 5. \tag{3.3.4}$$

Эта оценка дает ограничение на количество итераций во внешнем цикле алгоритма 10, что в свою очередь доказывает конечность числа операций алгоритма.

3.3.4. Скорость работы алгоритма продвигаемого фронта

Проведем краткий анализ скорости работы алгоритма 10. Формула (3.3.4) оценивает сверху количество итераций внешнего цикла на шаге 2. Обозначим через $N_e(\mathcal{F}^k)$ и $N_e(\mathcal{F}_R^k)$ количества отрезков в $\mathcal{F}^k$ и $\mathcal{F}_R^k$ соответственно. Выпишем все нетривиальные операции, которые используются в алгоритме 10.

1. Выбор отрезка с минимальной длиной из $\mathcal{F}^k$ (строка 3).
2. Построение локального фронта $\mathcal{F}_R^k$ (строка 7).
3. Построение списка кандидатов Σ_R^k (строка 8).
4. Проверка пересечения треугольника с локальным фронтом (строка 11).
5. Обновление фронта (строка 13).

Операция 4 выполняется за $N_e(\mathcal{F}_R^k)$ операций проверки пересечения треугольника с отрезком (см. алгоритм 11). Операция 3 может быть выполнена за $2N_e(\mathcal{F}_R^k)$ операций проверки принадлежности вершин из $\mathcal{F}_R^k$ окрестности $\omega_R(v_0)$.

Сложность операций 1, 2 и 5 зависит от используемых структур данных. Для хранения фронта введем упорядоченный список с минимальным по длине элементом в корне. Для быстрого поиска локального фронта мы применим четверичное дерево поиска. В этом случае операция 1 становится тривиальной, сложность операции 2 будет в среднем пропорциональна сумме $\log N_e(\mathcal{F}^k) + N_e(\mathcal{F}_R^k)$. Операция 5 состоит из трех операций добавления или удаления отрезка из фронта, сложность

каждой из них в среднем пропорциональна $\log N_e(\mathcal{F}^k)$. Более подробно устройство дерева поиска обсуждается в § 2.3.

Подытожим имеющиеся результаты по вычислительной сложности алгоритма 10. Пусть на входе задан многоугольник $\mathcal{P}$, полностью покрываемый квадратом со стороной H. Если локальный шаг сетки задается пользовательской функцией желаемого размера треугольников $s(x, y)$, то предположим опять, что она отделена от нуля: $s(x, y) \geqslant s_{\min} > 0$.

В п. 3.3.3 было показано, что существует параметр δ, с помощью которого по формуле (3.3.4) можно оценить максимальное количество треугольников в области. Оценим сверху вычислительную сложность одной итерации внешнего цикла в алгоритме 10. Нас будет интересовать зависимость сложности алгоритма от шага сетки, и соответственно от δ.

Предположим, что подходящий треугольник был найден в локальном переборе. На практике R выбирается в пределах от s до $2s$, и длины отрезков фронта в этой окрестности сопоставимы с s. Средняя оценка количества отрезков в локальном фронте $N_e(\mathcal{F}_R^k)$ получается не зависящей от δ. Однако в худшем случае она может составить величину, пропорциональную $N_e(\mathcal{F}^k)$, которая, в свою очередь, в худшем случае достигает $\dfrac{4}{\pi\delta^2}\,S(\mathcal{P}) + \dfrac{2}{\pi\delta}\,p(\mathcal{P}) + K(\mathcal{P})$. Поиск локального фронта в среднем пропорционален величине $N_e(\mathcal{F}_R^k) + \log_2 \dfrac{H}{\delta}$.

Далее, для каждой вершины локального фронта требуется проверить пересечение треугольника-кандидата с локальным фронтом. Сложность этой операции в среднем пропорциональна $(N_e(\mathcal{F}_R^k))^2$, что является величиной, не зависящей от δ. Но в худшем случае сложность операции может достигать величины, пропорциональной $(N_e(\mathcal{F}^k))^2$.

Полный перебор, когда алгоритм 10 переходит на шаг 18, соответствует худшей оценке для локального перебора.

Общая сложность алгоритма в среднем пропорциональна

$$W_{\text{avg}} = \Big(\frac{4S(\mathcal{P})}{\pi\delta^2} + \frac{2p\,(\mathcal{P})}{\pi\delta} + K(\mathcal{P})\Big)\log_2 \frac{H}{\delta}.$$

В редком наихудшем случае она пропорциональна

$$W_{\text{bad}} = \Big(\frac{4S(\mathcal{P})}{\pi\delta^2} + \frac{2p\,(\mathcal{P})}{\pi\delta} + K(\mathcal{P})\Big)^3.$$

В заключение мы приведем несколько областей, для которых сложность алгоритма значительно хуже средней оценки. Рассмотрим область в виде расчески, представленную на рис. 3.8. Будем увеличивать в ней количество зубьев N так, что при этом площадь будет оставаться примерно одной и той же, а периметр будет пропорционален N. Параметр δ будет уменьшаться обратно пропорционально N. Количество отрезков в начальном фронте будет $2N + 1$. На каждом шаге локальная

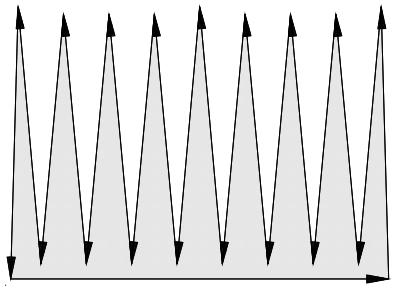


Рис. 3.8. Пример наихудшей конфигурации фронта

окрестность будет почти целиком накрывать всю область, поэтому сложность одной итерации внешнего цикла будет пропорциональна N^2. Всего будет построено $(2N - 1)$ треугольников. Общая сложность работы составит величину, пропорциональную N^3, что значительно хуже оценки в среднем порядка $N \log N$.

3.3.5. Результаты экспериментов

В этом пункте мы проиллюстрируем поведение алгоритма 10 на нескольких примерах. Сперва мы экспериментально оценим среднюю сложность работы алгоритма, а затем приведем результаты использования разных методов задания желаемого шага сетки.

Для экспериментального измерения скорости работы алгоритма продвигаемого фронта мы возьмем единичный квадрат и будем строить в нем неструктурированные квазиравномерные сетки с шагом h. Уменьшая шаг сетки h, будем следить за количеством треугольников в сетке N_f и временем построения сетки t. Результаты экспериментов представлены в табл. 3.1.

Таблица 3.1

Скорость работы алгоритма продвигаемого фронта

h	N_f	t, с	N_f/t, с$^{-1}$	$t/(N_f \log N_f)$, мкс
0,01	19304	0,29	66566	1,52
0,005	128334	2,08	61699	1,38
0,0025	512956	8,96	57250	1,33
0,00125	2019486	37,67	53610	1,28

Из результатов расчетов видно, что время работы пропорционально величине $N_f \log N_f$. Отметим, что коэффициент пропорциональности близок к 1. В таблицу также включен столбец со скоростью построения сетки (треугольники в секунду).

Теперь мы проанализируем, как выбор локального шага сетки, или параметра s, при построении нового треугольника в алгоритме 10 влияет на сетку и ее качество.

Рассмотрим единичный квадрат с вырезанным в центре кругом радиуса 0,1. Предлагаемые в этом пункте алгоритмы позволяют задавать желаемый размер элементов сетки с помощью скалярной функции $s(x, y)$. С помощью этой же функции можно построить дискретную границу области, заданной аналитически. Подробнее об этом будет рассказано в следующем параграфе.

При построении сеток использовались две скалярные функции: $s_1(x, y)$ и $s_2(x, y)$, отвечающие за желаемый размер треугольников. Первая функция уменьшалась вблизи двух полуокружностей, напоминающих по форме график функции $\sin(2\pi x)$. Вторая функция была тождественно равна константе на всей области. Полученные с их помощью сетки представлены на рис. 3.9, *а* и *б* соответственно.

Далее мы протестируем автоматический выбор размера сетки при заданной дискретной границе области. В качестве дискретной границы возьмем границу, полученную в предыдущем эксперименте с постоянной функцией $s_2(x, y)$. Протестируем работу алгоритма с автоматическим увеличением шага при разных значениях $\gamma = 1$; 1,05; 1,25; 20. Полученные сетки представлены на рис. 3.9, *в–е*. Выбор большего значения γ приводит к быстрому разрежению сетки внутрь области. Поэтому при очень большом значении γ мы получили нерегулярную сетку с минимальным количеством узлов внутри области (см. рис. 3.9, *е*).

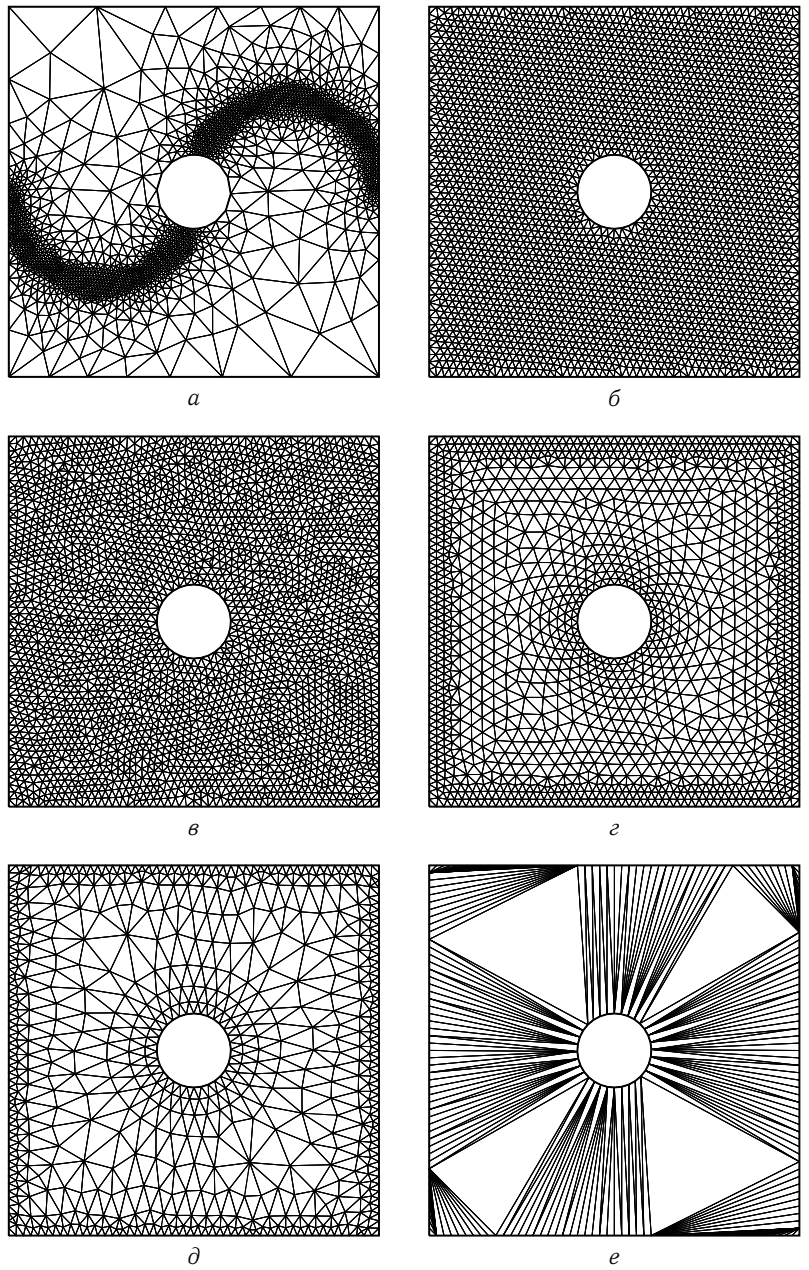
В табл. 3.2 собрана информация о количестве треугольников в построенных сетках. Напомним, что качество равнобедренного прямоугольного треугольника [см. формулу (2.1.3)] равно примерно 0,89, а качество равностороннего треугольника равно 1. Качество сетки $Q(\Omega_h)$ равно качеству наихудшего треугольника в ней. Отметим, что во всех рассмотренных примерах, кроме последнего, $Q(\Omega_h) > 0{,}6$, что характерно для регулярных сеток высокого качества.

Таблица 3.2

Количество треугольников и наихудшее качество треугольников при разных способах выбора шага сетки

	$s_1(x, y)$	$s_2(x, y)$	$\gamma = 1$	$\gamma = 1{,}05$	$\gamma = 1{,}25$	$\gamma = 20$
N_f	3636	6212	5386	2596	1194	232
$Q(\Omega^h)$	0,732	0,829	0,633	0,877	0,778	$0{,}641 \cdot 10^{-2}$

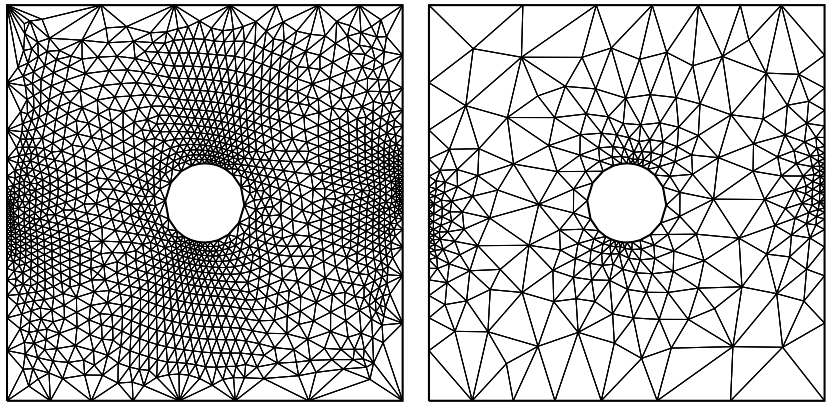
В дополнение к рассмотренным сеткам с равномерным следом на границе, проверим алгоритм 10 для неравномерной граничной дискретизации. В качестве такой дискретизации выберем след триангуляции, полученной с помощью функции $s_1(x, y)$ (см. рис. 3.9, *а*). Граничная дискретизация имеет непостоянный, но плавно меняющийся шаг. Результаты автоматического выбора шага с параметрами $\gamma = 1{,}05$



а *б* *в* *г* *д* *е*

Рис. 3.9. Разные способы выбора шага сетки: *а* — нетривиальная функция $s_1(x, y)$; *б* — константная функция $s_2(x, y)$; *в–е* — автоматический выбор с $\gamma = 1$; 1,05; 1,25 и 20

и $\gamma = 1{,}25$ показаны на рис. 3.10. Количество треугольников в получившихся сетках составило 2904 и 550 соответственно. Наихудшее качество треугольников было $Q(\Omega_h) = 0{,}334$ и $Q(\Omega_h) = 0{,}696$ соответственно.



а *б*

Рис. 3.10. Автоматический выбор шага сетки при неравномерной дискретизации границы: $\gamma = 1{,}05$ (*а*), $\gamma = 1{,}25$ (*б*)

Используя геометрические критерии для выбора функции $s(x, y)$, мы можем строить сетки, адаптированные к особенностям вычислительной модели. Уменьшая значение s в некоторой подобласти, например, в зоне предразрушения, мы будем измельчать там сетку. Более сложный подход к построению функции $s(x, y)$ основан на оценке ошибки конечноэлементого решения дифференциальной задачи — см. методы, описанные в приложении. Например, $s(x, y)$ можно согласовывать с некоторой нормой ошибки.

§ 3.4. Построение поверхностной триангуляции методом продвигаемого фронта

Технология построения треугольных *поверхностных* сеток, т. е. сеток на границе трехмерных областей, необходима для дальнейшего построения неструктурированных тетраэдральных сеток. Поверхностные треугольные сетки могут также использоваться при решении квазидвумерных задач на криволинейных поверхностях. Примерами таких задач могут служить задачи деформации тонкостенных конструкций или решение уравнений мелкой воды на поверхности сферы или геоида.

В этом параграфе мы покажем, что двумерный алгоритм продвигаемого фронта может быть расширен на случай криволинейных поверхностей, а также обсудим способы построения дискретных криволинейных границ и взаимодействия с САПР для получения необходимой информации о границе. Мы исследуем конечность и сложность представленных алгоритмов и приведем несколько примеров их работы.

Поверхностью назовем двумерное многообразие в трехмерном пространстве. *Простой* поверхностью назовем такую поверхность, которая гомеоморфна единичному квадрату.

Кривой на поверхности или в пространстве назовем одномерное многообразие на поверхности или в пространстве соответственно. *Простой* кривой назовем кривую, которая гомеоморфна единичному отрезку.

По возможности мы будем использовать обозначения из § 3.3. Площадь треугольника Δ_{123} на поверхности вычисляется так:

$$S_{\Delta_{123}} = \frac{1}{2}\,(\mathbf{e}_{31} \times \mathbf{e}_{32}) \cdot \mathbf{n}_3,$$

где $\mathbf{n}_3$ — внешняя единичная нормаль к поверхности в точке v_3, а $\mathbf{a} \cdot \mathbf{b}$ — скалярное произведение двух пространственных векторов. Отметим, что для плоскости новое обозначение переходит в определение из § 2.1. Отметим также, что такая величина соответствует значению $S_{\Delta(v_1', v_2', v_3)}$ в касательной к v_3 плоскости, где v_1' и v_2' — ортогональные проекции на плоскость точек v_1 и v_2 соответственно.

3.4.1. Представление поверхности

Рассмотрим сначала простую поверхность. Согласно ее определению, должна существовать такая параметризующая функция, которая переводит точку из единичного квадрата в параметрическом пространстве в точку на поверхности. Построение такого гомеоморфизма для произвольной поверхности является трудной задачей. На практике проще построить гомеоморфизм из некоторой ограниченной области $\omega \subset \mathbb{R}^2$, гомеоморфной единичному квадрату.

Разбиение границы ω при гомеоморфизме порождает разбиение границы простой поверхности, которое можно рассматривать как дискретную границу поверхности. Используя эту границу как фронт, можно применить алгоритм продвигаемого фронта, локально рассматривая поверхность как плоскость.

Сложная поверхность, как правило, может быть разрезана на несколько простых поверхностей. Зафиксировав дискретизацию на разрезах, можно для каждой простой поверхности построить согласованную с этой дискретизацией триангуляцию, тогда общая треугольная сетка для всей сложной поверхности будет конформной.

Составим алгоритм действий на основе описанной выше идеи. Вся поверхность разбивается на несколько простых поверхностей π_i, будем называть их *криволинейными гранями*. Каждая криволинейная грань параметризуется гладкой вектор-функцией

$$\pi_i = \left\{(x, y, z)\colon\ (x, y, z) = \mathbf{F}_i(p, s), \quad (p, s) \in \omega_i \subset \mathbb{R}^2\right\}.$$

Предположим, что вектор-функции $\mathbf{F}_i$ имеют непрерывные первые производные: $\mathbf{F}_i \in (C^1(\omega_i))^3$.

Границу криволинейной грани разобъем на несколько простых кривых γ_k, которые будем называть *криволинейными ребрами*. Введем параметризацию на криволинейных ребрах:

$$\gamma_k = \left\{(x, y, z)\colon\ (x, y, z) = \mathbf{G}_k(t), \quad t \in [t_k^0, t_k^1]\right\}.$$

Нам также понадобятся отображения p_{ik},s_{ik} из параметрического пространства кривой в параметрическое пространство поверхности:

$$\mathbf{G}_k(t) = \mathbf{F}_i(p_{ik}(t), s_{ik}(t)), \quad t \in [t_k^0, t_k^1].$$

Отметим, что для произвольных $\mathbf{F}_i$ и $\mathbf{G}_k$ вычисление этих отображений может оказаться сложной задачей. Поэтому на практике такой подход применим для достаточно простых областей.

При реализации этого метода на ЭВМ можно использовать другой подход. Рассмотрим параметризацию криволинейных ребер в параметрическом пространстве поверхности отдельно для каждой криволинейной грани. Пусть грань π_i параметризована функцией $\mathbf{F}_i(p, s)$, тогда ребро γ_k можно рассматривать как кривую в параметрическом пространстве (p, s). Например, если мы используем отображение $p \to (p, s_{ik}(p))$, то криволинейное ребро будет параметризовано следующим образом:

$$\gamma_k = \{(x, y, z)\colon\ (x, y, z) = \mathbf{F}_i(p, s_{ik}(p)), \quad p \in [p_i^0, p_i^1]\}.$$

Такие параметризации кривой γ_k могут быть разными для разных криволинейных граней, содержащих γ_k, но должны обеспечивать математически эквивалентные представления кривой. На практике можно допускать незначительное рассогласование, в пределах допустимой погрешности, представлений одной и той же кривой различными параметризациями, что упрощает построение параметризаций и расширяет класс областей, для которых такой подход применим.

Для построения поверхностных сеток на криволинейных гранях сначала строится дискретизация криволинейных ребер. Рассмотрим параметризованную простую кривую:

$$(x, y, z) = \mathbf{F}(p, s(p)) = \mathbf{G}(p), \quad p \in [p_0, p_1].$$

Будем строить на ней точки:

$$\mathbf{v}_i = \mathbf{G}(t_i), \quad i = 0, 1, \ \ldots, n.$$

Значения параметров $t_i \in [p_0, p_1]$ выбираются с помощью метода бисекции в соответствии с желаемым расстоянием между точками $\mathbf{v}_i$.

После построения дискретизации границы криволинейной грани мы можем применить аналог алгоритма продвигаемого фронта для построения треугольной сетки. Так как соседние грани имеют одну и ту же дискретизацию их общего ребра, а построенные на них треугольные

сетки согласованы с этой дискретизацией, то составленная из них общая сетка будет конформной.

Выпишем основные этапы построения поверхностной сетки. Предположим, что поверхность разбита на несколько непересекающихся простых поверхностей — криволинейных граней. Криволинейная граница каждой грани разбита на несколько простых кривых — криволинейных ребер. Будем рассматривать только конформные разбиения, т. е. такие, в которых у двух соседних граней общие криволинейные ребра.

На первом этапе строится дискретизация для всех криволинейных ребер. Для этого на кривой линии расставляются точки, и кривая аппроксимируется ломаной. Длина отрезков регулируется желаемым размером элементов сетки, при этом положение точек вычисляется с помощью метода бисекции.

На втором этапе для каждой криволинейной грани составляется дискретная граница из имеющихся ребер. Для точек на ребрах восстанавливаются значения параметров (p, s) в параметрическом пространстве грани. Наконец, с помощью алгоритма продвигаемого фронта строится поверхностная триангуляция, согласованная с дискретной границей. Формализуем эти идеи в алгоритме 13.

Алгоритм 13. Построение поверхностной сетки

```
1: for all криволинейных ребер do
2:     Выбрать одну из параметризаций простой кривой
3:     Построить дискретизацию кривой с помощью метода бисекций
4: end for
5: for all криволинейных граней do
6:     Составить дискретизацию границы из дискретизаций ребер
7:     Вычислить параметризацию узлов дискретной границы в параметрическом пространстве грани
8:     Применить метод продвигаемого фронта для построения сетки
9: end for
10: Объединить все построенные сетки в одну общую сетку
```

Согласованностью поверхностных сеток граней с дискретизациями криволинейных ребер гарантируется конформность общей сетки.

3.4.2. Взаимодействие с геометрическим ядром САПР

Информация о границе области может быть получена с помощью геометрического ядра САПР [4]. Большинство из существующих САПР предлагают интерфейс для взаимодействия со своим внутренним геометрическим ядром, позволяя получать информацию о топологии и геометрии области. Каждая САПР использует свой собственный интерфейс для этого взаимодействия, и общих стандартов в этой области пока нет. Некоторые САПР с открытым исходным кодом хорошо

документированы. Одним из примеров такой САПР может служить открытая система Open CASCADE Technology [19]. Однако в большинстве случаев закрытость исходного кода и отсутствие документации по интерфейсам в свободном доступе усложняет разработку интерфейсов с САПР.

Перспективным направлением видится использование промежуточных библиотек, предоставляющих общий унифицированный интерфейс для разработчика и поддерживающих взаимодействие с разными САПР. Во время разработки коммерческого пакета CUBIT Tool Suite, включающего в себя, в частности, собственное геометрическое ядро ACIS, разработчики стали добавлять новые интерфейсы к другим САПР. Для этого была создана специальная прослойка, предоставляющая общий интерфейс для взаимодействия с САПР. Позднее эта часть кода была вынесена из коммерческого проекта в виде отдельной открытой библиотеки Common Geometry Module (CGM) [20, 82]. На тот момент были разработаны интерфейсы к геометрическим ядрам систем ACIS и Pro/ENGINEER. Открытость исходного кода позволила использовать и совершенствовать эту библиотеку. В настоящее время осуществляется развитие проекта CGM под новым именем CGMA [21]. В этой версии добавлена возможность взаимодействия с геометрическим ядром Open CASCADE.

CGMA предлагает универсальный интерфейс для общения с геометрическими ядрами САПР. При этом вся информация делится на две части.

- Топологическая информация о модели: составные части модели и их топологические отношения между собой.
- Геометрическая информация: координаты, размеры, параметры и параметризующие функции для кривых и поверхностей.

В большинстве ядер САПР используется граничное представление моделей (B-Rep или BREP — сокращение от boundary representation). Такой метод представления используется и в библиотеке CGMA. Модель состоит из частей, образующих древовидную иерархию. Рассмотрим эти части, переходя от простых к более сложным.

1. Точка, для которой заданы ее координаты в пространстве (x, y, z).
2. Ребро — часть гладкой параметризованной кривой, ограниченная точками. Для ребра определена некоторая параметризация $t \leftrightarrow (x, y, z)$, а также ограничивающие ее точки и значения параметра t в этих точках.
3. Петля — связный и замкнутый набор ребер. Она не несет никакой геометрической информации и является чисто топологическим объектом.
4. Грань — часть гладкой параметризованной поверхности, ограниченная петлей. Ребра петли должны лежать на поверхности грани. Грань задана параметризацией поверхности $(p, s) \leftrightarrow (x, y, z)$.

5. Оболочка — связный и замкнутый набор граней. Так же, как и петля, является чисто топологическим объектом.
6. Тело — часть пространства, ограниченная оболочкой.
7. Набор тел, представляющих модель в целом.

Как отмечалось ранее, для построения трехмерной треугольной поверхностной сетки на каждой грани строится своя триангуляция, причем триангуляции граней конформно соединяются на общих ребрах. Для этого сначала строится дискретизация ребер. Ребра аппроксимируются ломаными с шагом в пространстве, заданным пользователем. После этого для каждой грани строится триангуляция с помощью алгоритма продвигаемого фронта [54]. В качестве начального фронта выступает дискретизация ребер.

Особое внимание стоит уделить криволинейным поверхностям. Геометрические ядра САПР часто создают грани с периодической параметризацией, а также с параметризацией, имеющей особые точки. Например, Open CASCADE задает боковую поверхность цилиндра одной гранью с периодической параметризацией, а боковую поверхность конуса гранью с особой точкой в вершине конуса. Более того, поверхность шара задается одной гранью с периодической параметризацией и двумя особыми точками в полюсах; сама грань при этом ограничивается лишь одним ребром, соединяющим два полюса. Алгоритм продвигаемого фронта должен быть соответствующим образом модифицирован, чтобы допускать такие особенности. Примеры работы алгоритма с геометрическими моделями САПР будут приведены в п. 3.4.4.

3.4.3. Алгоритм продвигаемого фронта

Алгоритм 10 продвигаемого фронта расширяется на случай криволинейных поверхностей. В этом пункте мы лишь отметим основные особенности алгоритма продвигаемого фронта для таких поверхностей.

В п. 3.3.2 был предложен алгоритм 11 для проверки пересечения треугольника с отрезком. В этом алгоритме использовалась только функция определения знака выражения $S_{\Delta_{123}}$. Покажем, как получить знак выражения $S_{\Delta_{123}}$ на поверхности. Построим точку $\mathbf{v} = \mathbf{v}_3 + \mathbf{n}_3$ так, что $\mathbf{n}_3 = \mathbf{v} - \mathbf{v}_3$ будет внешней нормалью к поверхности в точке v_3. По нашему определению

$$2S_{\Delta_{123}} = (\mathbf{e}_{31} \times \mathbf{e}_{32}) \cdot \mathbf{n}_3 = \begin{vmatrix} x_{v_1} - x_{v_3} & y_{v_1} - y_{v_3} & z_{v_1} - z_{v_3} \\ x_{v_2} - x_{v_3} & y_{v_2} - y_{v_3} & z_{v_2} - z_{v_3} \\ x_v - x_{v_3} & y_v - y_{v_3} & z_v - z_{v_3} \end{vmatrix} \equiv \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}.$$

Знак детерминанта $D = aei - ceg + bfg - afh + cdh - bdi$ матрицы 3×3 вычисляется по алгоритму, аналогичному алгоритму 12. Абсолютную погрешность можно оценить величиной $r = \varepsilon(|a| + |b| + |c| + |d| + |e| + |f| + |g| + |h| + |i| + |aei| + |ceg| + |bfg| + |afh| + |cdh| + |bdi|)$. Отметим, что если для координат узлов вещественных чисел используется одинарная точность, а для вычисления детерминанта

и промежуточных значений — двойная точность, то для получения точного ответа может не хватить запаса точности. Для точных вычислений необходима тройная или четверная точность. Некоторые современные архитектуры ЭВМ позволяют проводить вычисления с четверной точностью, и некоторые компиляторы могут использовать программную реализацию четверной точности на обычных процессорах.

Алгоритм 11 с учетом предыдущего замечания о вычислении $S_{\Delta_{123}}$ можно применять для проверки пересечения треугольника и отрезка на поверхности. На самом деле мы будем проверять пересечение проекции треугольника и проекции отрезка на касательную плоскость, поэтому этот метод применим только в локальных окрестностях, в которых поверхность мало отличается от касательной плоскости.

Для вычисления значения $S_{\Delta_{123}}$ нужно уметь строить нормаль к поверхности. Пусть поверхность параметризована следующим образом: $(p, s) \mapsto (x, y, z)$, и в некоторой окрестности точки v_3 существуют и непрерывны первые производные компонент x, y, z по параметрам p, s. Тогда направление нормали можно вычислить по следующей формуле:

$$\mathbf{n}_3 = k \left(\frac{D(y, z)}{D(p, s)}, \frac{D(z, x)}{D(p, s)}, \frac{D(x, y)}{D(p, s)} \right)^{\mathrm{T}}, \tag{3.4.1}$$

где

$$\frac{D(y, z)}{D(p, s)} = \begin{vmatrix} \dfrac{\partial y}{\partial p} & \dfrac{\partial y}{\partial s} \\ \dfrac{\partial z}{\partial p} & \dfrac{\partial z}{\partial s} \end{vmatrix},$$

$\dfrac{D(z, x)}{D(p, s)}$, $\dfrac{D(x, y)}{D(p, s)}$ определяются аналогично, k — нормирующий множитель.

В формуле (3.4.1) все три компоненты могут одновременно обратиться в 0. В этом случае для определения нормали можно выбрать производные вдоль других направлений в параметрическом пространстве. Условно запишем это так:

$$\mathbf{n}_3 = k \left(\frac{D(y, z)}{D(p', s')}, \frac{D(z, x)}{D(p', s')}, \frac{D(x, y)}{D(p', s')} \right)^{\mathrm{T}},$$

где, например, $p' = p - s$ и $s' = p + s$. В случае, если точка v_3 является особой точкой параметризации, можно отступить на небольшое расстояние от точки v_3 и вычислить нормаль в соседней точке.

Еще одной важной операцией, которая легко выполняется на плоскости, является построение на заданном ребре e_{12} равнобедренного треугольника $\Delta(v_1, v_2, v_3)$ с боковыми сторонами длины l. Для нахождения положения вершины v_3 будем искать такую точку в параметрическом пространстве (p_{v_3}, s_{v_3}), что $|v_1v_3| = |v_2v_3| = l$ в трехмерном

пространстве. Пусть вершина v_1 параметризована точкой (p_{v_1}, s_{v_1}). Будем искать параметризацию точки v_3 в виде

$$(p_{v_3}, s_{v_3}) = (p_{v_1} + \rho\cos\varphi, s_{v_1} + \rho\sin\varphi), \qquad \rho > 0, \quad \varphi \in [0, 2\pi].$$

При фиксированном φ можно подобрать ρ так, что $|v_1v_3| = l$ с некоторой точностью. Метод бисекции позволит ускорить поиск такой точки. В зависимости от результата сравнения стороны $|v_2v_3|$ с желаемой длиной l мы будем увеличивать или уменьшать φ до тех пор, пока не добьемся выполнения равенства $|v_2v_3| = l$ с некоторой точностью. Метод бисекции позволит ускорить поиск приемлемого направления φ.

Использование двух вложенных методов бисекции приводит к большому количеству вычислений, необходимых для построения равнобедренного треугольника. Поэтому на практике метод продвигаемого фронта на поверхности работает медленнее аналогичного метода на плоскости. Далее проведем краткий анализ алгоритма продвигаемого фронта для построения поверхностных сеток.

При анализе алгоритма продвигаемого фронта нужно уделить внимание двум моментам: существованию подходящего треугольника на каждом шаге и конечности количества шагов. К сожалению, вывод о существовании подходящего треугольника, полученный в п. 3.3.3, не переносится на случай поверхностного многоугольника. Более того, при достаточно большой кривизне поверхности и больших длинах отрезков во фронте, подходящего треугольника на заданном ребре может не существовать.

На практике при условии, что кривизна поверхности ограничена, а длины отрезков фронта достаточно малы, подходящий треугольник обычно находится. Более того, он находится среди кандидатов в некоторой небольшой окрестности рассматриваемого ребра.

Если кривизна поверхности ограничена и локальная окрестность достаточно мала, то проекция поверхности на касательную плоскость будет взаимно однозначным отображением. В этом случае для поиска пересечений треугольника с локальным фронтом можно использовать аналог алгоритма 11.

При использовании алгоритма 10 мы можем поддерживать нижнюю оценку минимального попарного расстояния между вершинами поверхностной сетки. Рассуждения из п. 3.3.3 применимы в этом случае с точностью до кривизны поверхности и эйлеровой характеристики поверхности, которая для простых поверхностей равна эйлеровой характеристике плоскости [13].

Анализ сложности алгоритма продвигаемого фронта из п. 3.3.4 применим и в поверхностном случае. Хотя конечность алгоритма в общем случае не доказана, на практике, при работе с поверхностями ограниченной кривизны и с достаточно мелким шагом сетки, сложность алгоритма в среднем пропорциональна $N_f \log N_f$, где N_f — общее количество треугольников.

3.4.4. Результаты экспериментов

В этом пункте мы проведем два численных эксперимента. В первом эксперименте мы оценим сложность алгоритма продвигаемого фронта для криволинейных поверхностей, заданных аналитически. Во втором эксперименте мы покажем возможность использования интерфейса с геометрическим ядром САПР при построении поверхностной сетки.

Измерим экспериментально скорость работы алгоритма продвигаемого фронта на поверхности. В качестве поверхности возьмем верхнюю полусферу единичной сферы. Зададим параметризацию полусферы аналитически с помощью отображения $(p, s) \mapsto (p, s, \sqrt{1 - p^2 - s^2})$, где $p^2 + s^2 \leqslant 1$.

Будем строить квазиравномерную сетку с шагом h. Уменьшая шаг сетки, будем следить за количеством треугольников в сетке N_f и временем построения сетки t. Результаты экспериментов представлены в табл. 3.3.

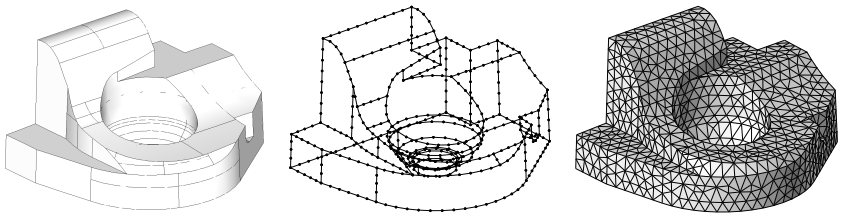
Таблица 3.3
Скорость работы алгоритма продвигаемого фронта на поверхности

h	N_f	t, с	N_f/t, с^{-1}	$t/(N_f \log N_f)$, мкс
0,1	1298	0,49	2649	121
0,05	5460	2,18	2505	107
0,025	22152	9,36	2367	97
0,0125	89730	40,34	2224	91

Из результатов расчетов видно, что время работы ограничено величиной $N_f \log N_f$. Коэффициент пропорциональности (последний столбец в таблице) достаточно большой. В таблицу также включен столбец со скоростью построения сетки (треугольники в секунду).

Продемонстрируем совместную работу генератора поверхностных треугольных сеток и геометрического ядра САПР — OpenCASCADE. В качестве примера рассмотрим модель `29_misc1` с сайта OpenCASCADE [19]. Геометрическая модель состоит из 63 вершин, 96 криволинейных ребер и 36 криволинейных граней (см. рис. 3.11, *а*).

Сначала строится дискретизация всех криволинейных ребер геометрической модели. Построенная дискретизация изображена на рис. 3.11, *б*. Она состоит из 461 узла и 494 отрезков. Далее, для каждой из 36 криволинейных граней запускается алгоритм продвигаемого фронта на поверхности. В зависимости от размера криволинейной грани количество построенных треугольников колеблется от 3 до 641. Итоговая поверхностная триангуляция содержит 2594 треугольника и 1299 узлов. Полученная поверхностная триангуляция изображена на рис. 3.11, *в*.



а *б* *в*

Рис. 3.11. Модель, заданная в САПР: *а* — BREP-модель; *б* — дискретизация криволинейных ребер; *в* — поверхностная квазиравномерная треугольная сетка

Отметим, что скорость работы алгоритма, использующего интерфейс с ядром САПР, сильно зависит от скорости вычисления параметризующих функций внутри САПР. В рассмотренном примере грань с наибольшим количеством треугольников (641) была частью плоскости, и триангуляция для нее была построена за 2,11 с, в то время как для другой криволинейной грани было потрачено 4,54 с на треугольную сетку, состоящую всего из 25 треугольников. Общее время построения поверхностной сетки составило 14 с.

§ 3.5. Метод улучшения заданной поверхностной сетки

Поверхностные сетки, полученные экспортом из многих систем САПР, обладают тем свойством, что количество треугольников на плоских участках минимально, а сами треугольники могут иметь очень вытянутую форму, что сказывается на регулярности триангуляции начального фронта, на качестве результирующей тетраэдральной сетки и даже на возможности ее построить. *Для построения тетраэдральных сеток высокого качества необходима регулярность начального фронта.*

Предлагаемая технология улучшения поверхностной сетки базируется на выделении почти плоских связных подобластей в исходной поверхностной сетке и использовании алгоритма продвигаемого фронта для покрытия их регулярной сеткой.

Основной идеей предлагаемого метода является выделение плоских (или почти плоских) кусков поверхности, называемых далее *полигонами*, и переразбиение их на новые треугольники регулярной формы. При этом возможно незначительное отклонение от исходной дискретной поверхности [7].

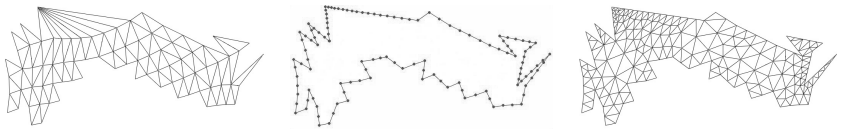
Входными данными для этого метода являются конформная триангуляция поверхности и возможная маркировка треугольников, обеспечивающая разделение поверхности на полигоны. Каждый полигон будет обрабатываться отдельно, причем геометрические границы полигонов сохранятся в исходном виде. Для пользователя доступны некоторые параметры, позволяющие управлять результатом переразбиения.

Среди них скорость огрубления треугольников при построении новой поверхностной сетки с помощью алгоритма продвигаемого фронта. Есть также параметр, отвечающий за допустимую степень отклонения полигонов от плоскости.

Алгоритм построения регулярной триангуляции заданной дискретной поверхности представляет собой итерационную процедуру (см. алгоритм 14). Сначала поверхность разбивается на полигоны, для которых задача сводится к двумерной, а потом из полученных триангуляций полигонов собирается триангуляция всей поверхности. Основные операции с полигоном продемонстрированы на рис. 3.12.

Алгоритм 14. Перестроение поверхностной сетки

1: **while** исходная триангуляция не исчерпана **do**
2: Выделить связную подобласть (полигон), треугольники которой лежат в одной плоскости в пределах заданной точности
3: С помощью поворота плоскости полигона на плоскость Oxy и обнуления z-координат отобразить полигон на плоскость Oxy
4: Разбить границы полигона и применить метод продвигаемого фронта для построения двумерной регулярной триангуляции области с заданным следом на границе
5: Спроецировать (по оси Oz) каждый узел новой триангуляции на повернутый полигон. Поскольку последний слабо отклоняется от плоскости Oxy, качество спроецированных треугольников практически не ухудшается. Поворотом плоскости Oxy на плоскость полигона завершить обратную замену координат
6: Исключить из исходной триангуляции выделенный полигон
7: **end while**



а *б* *в*

Рис. 3.12. Выделенный полигон: *а* — исходная сетка в полигоне; *б* — новое разбиение границы полигона; *в* — новая сетка в полигоне

Рассмотрим процедуру выделения полигональной подобласти более подробно. Полигональная подобласть — связное множество треугольников, принадлежащих с заданной точностью одной плоскости. Точность определяется максимально допустимым расстоянием h_0 от треугольника до плоскости и максимальным отклонением ε между нормалью к треугольнику и нормалью к заданной плоскости.

Формирование полигона начинается с выбора начального треугольника. Для определенности выбирается еще не рассмотренный

треугольник с наибольшей площадью. Его плоскость считается плоскостью будущего полигона, его маркер задает маркер полигона, а граница назначается текущей границей полигона. Будем называть эту плоскость *опорной плоскостью*. Далее работает итерационный алгоритм построения полигона.

Выбирается одно из граничных ребер полигона и проверяются все треугольники исходной сетки, которые граничат с полигоном через это ребро. Треугольник добавляется в полигон, если его маркер совпадает с маркером полигона, расстояние от опорной плоскости до любой из вершин треугольника меньше h_0 и синус угла между нормалью опорной плоскости и нормалью к треугольнику-кандидату меньше ε. Если треугольник удовлетворяет всем этим условиям, то он добавляется в полигон и строится новая граница; в противном случае граничное ребро помечается как проверенное и рассматривается следующее. Через одно ребро может быть добавлен только один треугольник. Если нашлось несколько подходящих треугольников-кандидатов, то ни один из них не добавляется, а ребро помечается как проверенное. Такая ситуация возможна, если ребро принадлежит одновременно нескольким поверхностным триангуляциям (см. рис. 3.13). После того, как все граничные ребра оказываются помеченными, полигон считается построенным.

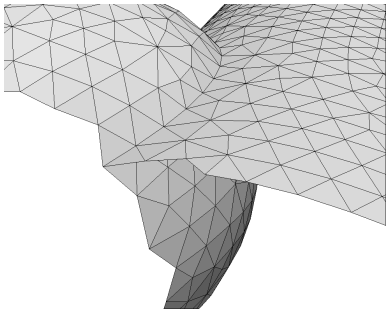


Рис. 3.13. Пересечение двух поверхностных триангуляций

Дискретизация границы полигона, задаваемая исходной триангуляцией, может оказаться сильно неравномерной, поскольку исходная триангуляция может быть нерегулярной (в смысле формы треугольников). Неоднородность дискретизации границы неизбежно приведет к появлению треугольников с очень острыми углами вблизи границы. Поэтому при новом разбиении границы надо избавиться от резких перепадов шага разбиения.

Одним из вариантов решения может быть использование постоянного равномерного шага для дискретизации границы полигона. Этот подход удобен для построения квазиравномерной поверхностной сетки на основе заданной поверхностной триангуляции. Однако в этом случае шаг дискретизации должен быть не меньше, чем минимальная длина

граничного ребра каждого полигона. Пример улучшения поверхностной сетки показан на рис. 3.14.

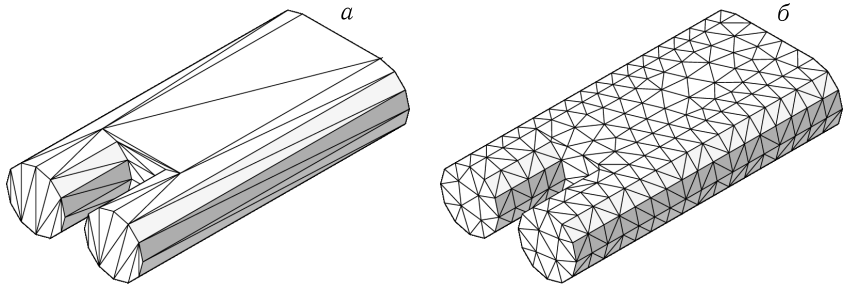


Рис. 3.14. Поверхностная сетка, экспортированная из САПР (*а*), и улучшенная поверхностная сетка (*б*)

Можно использовать другой подход, в котором шаг дискретизации границы полигона согласован с локальной кривизной поверхности. Вычисление локальной кривизны поверхности, заданной своей триангуляцией, является отдельной сложной задачей. В этом пункте будет предложен способ, не вычисляющий явным образом локальную кривизну поверхности.

Предполагая, что локальный размер треугольников исходной сетки согласован с кривизной поверхности, для построения регулярной триангуляции будем использовать минимальные размеры существующих треугольников. Определим для каждого узла v исходной триангуляции параметр шага h_v как минимальную высоту из всех треугольников, сходящихся в узле. Параметр шага задает размер регулярной сетки в вершинах выделенного полигона. Для каждого граничного ребра полигона определим непрерывную функцию параметра шага как линейную интерполяцию заданных выше величин h_{v_1} и h_{v_2} в вершинах ребра. При этом расположение узлов новой дискретизации вдоль ребра $e(v_1, v_2)$ задается формулой

$$\mathbf{r}_i = \mathbf{v}_1 + \frac{\mathbf{v}_2 - \mathbf{v}_1}{|v_1 v_2|}\left(h'_{v_1} + d\frac{i}{2}\right) i, \quad i = 1, \ \ldots, [Q] - 1,$$

где

$$d = \frac{h'_{v_2} - h'_{v_1}}{[Q]}, \quad h'_{v_1} = h_{v_1}\frac{Q}{[Q]}, \quad h'_{v_2} = h_{v_2}\frac{Q}{[Q]}, \quad Q = \frac{2|v_1 v_2|}{h_{v_1} + h_{v_2}},$$

$[Q]$ — целая часть Q.

Отметим, что после поворота полигона и обнуления z-координаты его вершин (шаг 3 алгоритма 14) возможно самопересечение границы плоского полигона, который необходимо триангулировать. Подобная ситуация не позволяет использовать алгоритм продвигаемого фронта. В этом случае производится возврат алгоритма, и полигон строится заново, причем при добавлении очередного треугольника в полигон дополнительно проверяется возможность самопересечения границы.

При построении новой сетки может быть использован равномерный шаг сетки, если граница полигона была разбита с постоянным шагом. В случае неравномерного шага на границе удобен вариант алгоритма продвигаемого фронта с автоматическим выбором шага и огрублением при продвижении внутрь области. На практике шаг сетки внутри полигона лучше ограничивать некоторым максимальным значением. В случае, если имеется возможность вычисления локальной кривизны поверхности, заданной своей триангуляцией, локальный шаг сетки может быть согласован с локальной кривизной поверхности.

Поскольку расположение узлов на ребре полигона зависит только от величин h_{v_1} и h_{v_2} в вершинах ребра, а сетка внутри каждого полигона имеет заданный след на границе, результирующая поверхностная сетка является конформной.

Улучшение поверхностной сетки — важнейший этап в технологической цепочке построения тетраэдральных сеток высокого качества. Качество объемной сетки сильно зависит от качества поверхностной сетки. Другим примером применения предложенного здесь метода может служить уменьшение количества треугольников в слишком мелких поверхностных сетках. Такие сетки могут быть получены, например, в результате трехмерного сканирования. Они могут насчитывать сотни тысяч треугольников и очень неудобны для построения тетраэдральных сеток. Метод улучшения поверхности может быть использован для уменьшения общего количества треугольников с сохранением основных геометрических особенностей модели.

§ 3.6. Построение тетраэдральной сетки методом продвигаемого фронта

В этом параграфе будет изучено расширение алгоритма продвигаемого фронта для трехмерного пространства. Будут обсуждены вопросы его надежности с точки зрения неточных вычислений и с точки зрения возможности продвижения фронта, конечность алгоритма и сложность его работы.

Введем понятия и обозначения, аналогичные используемым в § 3.3. Напомним, что $f(v_1, v_2, v_3)$, или f_{123} обозначает треугольник с вершинами v_1, v_2 и v_3. Аналогично, $\Delta(v_1, v_2, v_3, v_4)$, или Δ_{1234} обозначает тетраэдр с вершинами v_1, v_2, v_3 и v_4.

Ориентированным треугольником в пространстве назовем треугольник с некоторым зафиксированным, с точностью до четной перестановки, порядком обхода его вершин.

Положительным полупространством относительно ориентированного треугольника f_{123} является множество точек v, для которых

ориентированный объем $V_{\Delta(v_1, v_2, v_3, v)} > 0$. Аналогично, *отрицательным полупространством* является множество точек, для которых $V_{\Delta(v_1, v_2, v_3, v)} < 0$.

Триангуляцией в пространстве назовем конечный набор треугольников, конформно связанных через общие ребра. При этом два соседних треугольника могут лежать в одной плоскости. Если каждое ребро триангуляции принадлежит ровно двум треугольникам, то такую триангуляцию будем называть *замкнутой*. Триангуляцию, треугольники которой (как открытые множества) не пересекаются друг с другом, будем называть триангуляцией *без самопересечений*.

Многогранником в пространстве назовем ограниченную часть пространства, граница которой состоит из одной или нескольких непересекающихся триангуляций без самопересечений. *Простым многогранником* назовем многогранник, ограниченный одной замкнутой триангуляцией. *Вершинами* многогранника будем называть узлы его триангуляции, а *гранями* многогранника — треугольники триангуляции. Отметим, что гранями полиэдральных областей, рассматриваемых в § 2.2, могут быть любые плоские многоугольники.

Аналогично § 3.3, грани многогранника делятся на внешние и внутренние. При этом на внешних гранях можно ввести ориентацию так, что многогранник будет лежать в положительном полупространстве относительно каждого ориентированного треугольника. Такую ориентацию будем называть *положительной*.

Фронтом в пространстве является набор ориентированных треугольников без самопересечений. Каждому многограннику $\mathcal{P}$ можно поставить в соответствие фронт $\mathcal{F}(\mathcal{P})$. Для этого введем на внешних гранях $\mathcal{P}$ положительную ориентацию, а каждой внутренней грани f_{123} поставим в соответствие пару противоположно ориентированных треугольников f_{123} и f_{321}. Совокупность всех этих ориентированных треугольников образует фронт $\mathcal{F}(\mathcal{P})$. Будем называть фронт $\mathcal{F}$ *замкнутым*, если существует такой многогранник $\mathcal{P}$, что $\mathcal{F} = \mathcal{F}(\mathcal{P})$.

Тетраэдральной сеткой Ω_h для многогранной области $\mathcal{P}$ назовем разбиение этой области на непересекающиеся тетраэдры. Напомним, что сетка является *конформной*, если любые ее два элемента либо не имеют общих точек, либо имеют одну общую вершину, либо имеют одно общее целое ребро, либо одну общую целую грань. Будем говорить, что конформная сетка *согласована с границей* $\mathcal{P}$, если каждая грань $\mathcal{P}$ является гранью какого-то тетраэдра в сетке. В частности, у внутренних граней многогранника будут ровно два соседних тетраэдра, а у внешних — ровно один. Множество тетраэдров сетки будем обозначать $\mathcal{T}$.

3.6.1. Алгоритм продвигаемого фронта

Алгоритм 10 продвигаемого фронта обобщается на случай трехмерного пространства. В этом пункте мы отметим основные особенности трехмерного алгоритма продвигаемого фронта.

Отличительной особенностью трехмерного пространства является существование таких фронтов, для которых не существует подходящих тетраэдров. По этой причине в алгоритм добавляется возможность помечать грани, для которых не нашлось подходящих тетраэдров, и пропускать их при дальнейшем переборе.

В конце работы алгоритма мы получим множество тетраэдров $\mathcal{T}$ сетки Ω_h^{mesh} для многогранной подобласти $\mathcal{P}_{\text{mesh}} \subseteq \mathcal{P}$ и фронт для неразбитой многогранной подобласти $\mathcal{P}_{\text{out}} = \mathcal{P} \setminus \mathcal{P}_{\text{mesh}}$. Отметим, что если алгоритм продвигаемого фронта завершится неудачей, то сетка будет построена только в части области. В этом случае $\mathcal{P}_{\text{out}} \neq \varnothing$, и для завершения построения сетки необходимо применить второй алгоритм.

Общая последовательность действий представлена в алгоритме 15. Разберем в деталях основные действия, которые выполняются при продвижении фронта.

Алгоритм 15. Метод продвигаемого фронта в трехмерном пространстве

1: Положить $\mathcal{T}^0 = \varnothing$, $\mathcal{F}^0 = \mathcal{F}(\mathcal{P})$
2: **for** $k = 0, 1, \ldots$ **do**
3: Если непомеченных граней нет, то перейти к 22
4: Выбрать непомеченную грань $f_{123} \in \mathcal{F}^k$ с минимальной площадью
5: Определить желаемую длину s ребра тетраэдра
6: Построить вершину v_0 с учетом s и $V_{\Delta_{1230}} > 0$
7: Выбрать некоторое $R_0 > s$, $h > 0$ и $r > 0$
8: **for** $R \in \{R_0, \infty\}$ **do**
9: Построить локальный фронт $\mathcal{F}_R^k$: $\mathcal{F}_R^k \supseteq \mathcal{F}^k \cap \omega_R(v_0)$
10: Определить множество вершин-кандидатов $\Sigma_R^k = \{p_i^k\} \cap \cap\, \omega_R(v_0)$
11: Если $\{p_i^k\} \cap \omega_r(v_0) = \varnothing$ и $\mathcal{F}^k \cap \omega_h(v_0) = \varnothing$, то добавить v_0 к Σ_R^k
12: **for all** $v_4 \in \Sigma_R^k$ **do**
13: **if** Δ_{1234} не пересекает $\mathcal{F}_R^k$ **then**
14: Добавить тетраэдр Δ_{1234} в сетку, $\mathcal{T}^{k+1} = \mathcal{T}^k \cup \cup \{\Delta_{1234}\}$
15: Обновить фронт: $\mathcal{P}^{k+1} = \mathcal{P}^k \setminus \Delta_{1234}$ и $\mathcal{F}^{k+1} = \mathcal{F}(\mathcal{P}^{k+1})$
16: Перейти к 21
17: **end if**
18: **end for**
19: **end for**
20: Пометить f_{123} и перейти к 4
21: **end for**
22: Положить $\mathcal{T} = \mathcal{T}^k$, $\mathcal{P}_{\text{out}} = \mathcal{P}^k$ и $\mathcal{F}_{\text{out}} = \mathcal{F}^k$

На шаге 4 из фронта выбирается непомеченная грань f_{123} с наименьшей площадью и строится вершина v_0 в соответствии с выбранным параметром l, зависящим от s (см. шаг 5). При реализации алгоритма на ЭВМ предлагается использовать следующий эвристический метод выбора l и положения вершины v_0. Пусть a, b, c — длины сторон треугольника f_{123}. Сначала строится центр масс v_m этого треугольника, затем от v_m по нормали к f_{123} на расстоянии $\sqrt{\frac{2}{3}}\,l$ фиксируется вершина v_0 (см. рис. 3.15). Параметр l выбирается на основании функции желаемого размера элементов, предоставленной пользователем, либо вычисляется автоматически по формуле $l = \gamma\sqrt{\frac{a^2+b^2+c^2}{3}}$, где $\gamma \geqslant 1$ — некоторый параметр, отвечающий за скорость автоматического огрубления сетки.

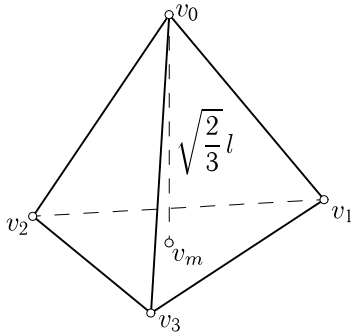


Рис. 3.15. Тетраэдр с основанием f_{123} и вершиной v_0

Оценим длину ребер e_{10}, e_{20} и e_{30}. При $\gamma \geqslant 1$ высота тетраэдра $l_0 \geqslant \frac{1}{3}\sqrt{2a^2+2b^2+2c^2}$. Рассмотрим ребро e_{10}: $|e_{10}|^2 = |e_{1m}|^2 + |e_{m0}|^2$. Длина e_{1m} выражается из формулы для медианы треугольника: $|e_{1m}| = \frac{1}{3}\sqrt{2b^2+2c^2-a^2}$. Таким образом, получаем

$$|e_{10}|^2 \geqslant \frac{2b^2+2c^2-a^2}{9} + \frac{2a^2+2b^2+2c^2}{9} = \frac{a^2+4b^2+4c^2}{9} \geqslant \min\{a^2, b^2, c^2\}.$$

Если стороны треугольника Δ_{123} были не меньше d, то и $|e_{10}|$ будет не меньше d. Аналогичные оценки верны для e_{20} и e_{30}.

Для тетраэдра-кандидата с вершиной v_4 мы проверяем его пересечение с локальным фронтом. Если все кандидаты оказались неподходящими, то проводится полный перебор всего фронта. Если и в этом случае подходящего тетраэдра найдено не было, то грань f_{123} помечается как уже рассмотренная, и перебор начинается сначала для следующей грани.

В отличие от алгоритма 10, участок алгоритма 15 с номерами шагов 4–20 может выполниться несколько раз для построения одного тетраэдра. При каждом выполнении этого участка соответствующая грань либо удаляется из фронта, либо помечается. В конце работы алгоритма все такие грани станут гранями конечной сетки Ω_h. Поэтому количество выполнений этого участка алгоритма ограничено сверху не количеством тетраэдров в конечной сетке, а количеством граней.

Рассмотрим более детально остальные отличия алгоритма продвигаемого фронта в трехмерном пространстве от его аналога в двумерном пространстве из § 3.3.

Построим алгоритм пересечения тетраэдра с треугольником, аналогичный алгоритму 11. Как и в п. 3.3.2, будем проводить проверки на основе знака определителя матрицы 3×3.

В общем случае два выпуклых объекта не пересекаются тогда и только тогда, когда существует плоскость, разделяющая их. В условиях алгоритма 15 тетраэдр и треугольник могут также иметь одну общую вершину или одно общее ребро, или треугольник может быть одной из граней тетраэдра.

Выделим из задачи проверки пересечения отдельную подзадачу проверки пересечения треугольника с отрезком. В п. 3.4.3 мы уже рассматривали аналог этой задачи для поверхности. Теперь рассмотрим общий случай: будем искать такую плоскость, которая разделяет треугольник и отрезок с учетом предыдущего замечания об общих вершинах. Перебор всех возможных плоскостей представлен в алгоритме 16.

Проверка пересечения треугольника с отрезком используется в алгоритме 17 проверки пересечения тетраэдра с треугольником. Сначала мы проверяем, имеются ли пересечения у границы тетраэдра с границей треугольника, а потом проверяем особые случаи, такие, как треугольник, целиком лежащий внутри тетраэдра. Этот порядок проверок на практике наиболее эффективен, так как в большинстве случаев хотя бы одна из граней тетраэдра пересекает ребро треугольника, реже треугольник пересекает хотя бы одно из ребер тетраэдра, и еще реже треугольник лежит внутри тетраэдра.

Предложенная комбинация алгоритмов 16 и 17 для проверки пересечения использует функцию $d(v_1, v_2, v_3, v_4)$, вычисляющую знак выражения $V_{\Delta_{1234}}$, аналогичную функции $d(v_1, v_2, v_3)$, определенной в алгоритме 12. Пусть про функцию $d(v_1, v_2, v_3, v_4)$ известно, что при $d(v_1, v_2, v_3, v_4) \neq 0$ выполняется $d(v_1, v_2, v_3, v_4) = \operatorname{sign}(V_{\Delta_{1234}})$. Тогда предложенные алгоритмы исключают возникновение ошибок второго типа (см. § 3.3), т. е. фактически пересекающиеся тетраэдр и треугольник не будут ошибочно восприняты как непересекающиеся. Этим гарантируются замкнутость и отсутствие самопересечений при продвижении фронта.

Алгоритм 16. Проверка пересечения треугольника с отрезком в трехмерном пространстве

1: Найти общие вершины у треугольника Δ_{123} и отрезка e_{45}
2: **if** общих вершин нет **then**
3: Определить положение вершин v_4 и v_5 относительно плоскости Δ_{123}
4: **if** v_4 и v_5 лежат в одном полупространстве **then**
5: Пересечения нет
6: **else if** v_4 и v_5 лежат в разных полупространствах **then**
7: Вычислить $k_1 = d(v_4, v_5, v_2, v_3)$, $k_2 = d(v_4, v_5, v_3, v_1)$, $k_3 = d(v_4, v_5, v_1, v_2)$
8: Если среди k_1, k_2, k_3 имеются и 1 и -1, то пересечения нет
9: **else if** v_4 и v_5 лежат в плоскости Δ_{123} **then**
10: Проверить, что одна из прямых, заданных отрезками v_4v_5, v_1v_2, v_2v_3 или v_3v_1, разделяет треугольник и отрезок в разные полуплоскости
11: Если такая прямая нашлась, то пересечения нет
12: **end if**
13: **end if**
14: **if** одна общая вершина **then**
15: Если вторая вершина отрезка не лежит в плоскости треугольника, то пересечения нет
16: Иначе проверить, что одна из прямых, заданных отрезками v_1v_2, v_2v_3, v_3v_1, разделяет треугольник и отрезок в разные полуплоскости
17: Если такая прямая нашлась, то пересечения нет
18: **end if**
19: **if** две общих вершины **then**
20: Пересечения нет
21: **end if**
22: В остальных случаях считаем, что треугольник и отрезок пересекаются

Проведем анализ алгоритма 15 и покажем конечность числа операций. В предложенном алгоритме присутствуют три явных вложенных цикла и один неявный за счет перезапуска перебора на шаге 20. Цикл перебора на шаге 12 всегда конечен в силу конечности множества Σ_R^k. Цикл на шаге 8 делает не больше двух итераций.

В трехмерном случае возможны такие конфигурации фронта, для которых не существует подходящего тетраэдра с четвертой вершиной из множества вершин-кандидатов фронта. Примеры такой конфигурации фронта, для которой дальнейшее продвижение невозможно, представлены на рис. 3.16: любой тетраэдр с вершинами из множества $\{v_1, v_2, v_3, v_4, v_5, v_6\}$ пересекает одно из ребер $\{e_{26}, e_{34}, e_{15}\}$. Именно по этой причине на шаге 20 алгоритма 15 грани, не имеющие

Алгоритм 17. Проверка пересечения тетраэдра с треугольником

1: Для каждой грани тетраэдра Δ_{1234} и ребра треугольника f_{567} проверить их пересечение
2: Для f_{567} и каждого ребра Δ_{1234} проверить их пересечение
3: Найти общие вершины у тетраэдра Δ_{1234} и треугольника f_{567}
4: **if** общих вершин нет **then**
5: Если три точки v_5, v_6, v_7 лежат внутри Δ_{1234}, то есть пересечение
6: **end if**
7: **if** одна общая вершина **then**
8: Если две оставшиеся вершины f_{567} лежат внутри Δ_{1234}, то есть пересечение
9: **end if**
10: **if** две общих вершины **then**
11: Если оставшаяся вершина f_{567} лежит внутри Δ_{1234}, то есть пересечение
12: **end if**
13: В остальных случаях считаем, что пересечения нет

подходящих тетраэдров, помечаются и пропускаются при дальнейшем построении сетки. Каждая грань помечается не более одного раза, а количество граней в $\mathcal{F}^k$ конечно, поэтому неявный цикл на шаге 20 конечен.

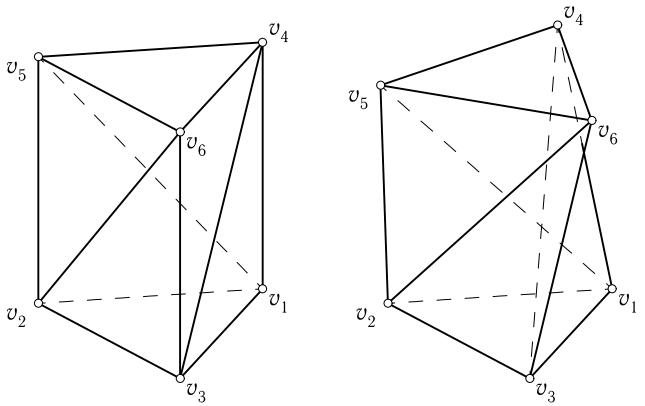


Рис. 3.16. Примеры конфигурации фронта, для которых дальнейшее продвижение невозможно: призмы Шонхардта

Покажем, что при определенном выборе параметров в алгоритме 15 можно ограничить максимально возможное количество построенных тетраэдров. Рассуждения из § 3.3.3 полностью переносятся на трехмерный случай. Мы можем ограничить снизу расстояние между узлами сетки: $\rho_k \geqslant \delta > 0$ при любом k. Верна и аналогичная (3.3.3) оценка

максимального количества узлов, которое можно поместить в области $\mathcal{P}$, имеющей объем $V(\mathcal{P})$, площадь поверхности $S(\mathcal{P})$, сумму длин ребер $p(\mathcal{P})$ и состоящей из $K(\mathcal{P})$ компонент связности:

$$N_v^k \leqslant \frac{6V(\mathcal{P})}{\pi\delta^3} + \frac{3S(\mathcal{P})}{\pi\delta^2} + \frac{3P(\mathcal{P})}{2\delta} + K(\mathcal{P}). \tag{3.6.1}$$

Из (2.2.9) следует, что количество тетраэдров и, следовательно, число итераций внешнего цикла в алгоритме 15 ограничены.

Анализ сложности алгоритма продвигаемого фронта из п. 3.3.4 применим и в трехмерном случае. Оценка сверху количества операций в худшем случае составит величину, пропорциональную N_f^3, где N_f — количество граней в конечной сетке, которое ограничено согласно (2.2.9) и оценке количества узлов (3.6.1). Оценка в среднем количества операций составляет величину, пропорциональную $N_f \log N_f$. Эта оценка средней скорости работы будет экспериментально подтверждена в п. 3.7.6.

Мы показали, что алгоритм 15 совершает конечное число операций. Однако он не гарантирует построение конформной тетраэдральной сетки для всей области $\mathcal{P}$. На практике алгоритм продвигаемого фронта разбивает более 90 % объема области, а $\mathcal{P}_{\text{out}}$ — неразбитая часть области — состоит из некоторого количества изолированных лакун — многогранников с малым количеством граней. Для тетраэдризации области $\mathcal{P}_{\text{out}}$ применяется второй метод, о котором речь пойдет в следующем параграфе.

§ 3.7. Надежный алгоритм построения тетраэдральной сетки

В этом параграфе мы изложим метод построения тетраэдральной сетки, согласованной с заданным фронтом, на основе тетраэдризации Делоне. Общая идея этого метода была предложена Джорджем с соавторами в [55]. Мы рассмотрим упрощенную версию этого метода, применимую для разбиения лакун, оставленных алгоритмом продвигаемого фронта [12]. Этот метод пригоден для произвольных замкнутых фронтов, однако не позволяет контролировать ни размер, ни качество построенных тетраэдров. Для улучшения качества сетки требуется третий метод.

Построение сетки делится на три этапа. На первом этапе строится сетка для выпуклой оболочки множества точек заданного фронта. На втором этапе сетка измельчается, и восстанавливается геометрия области. На третьем этапе восстанавливается согласованность сетки с заданным фронтом.

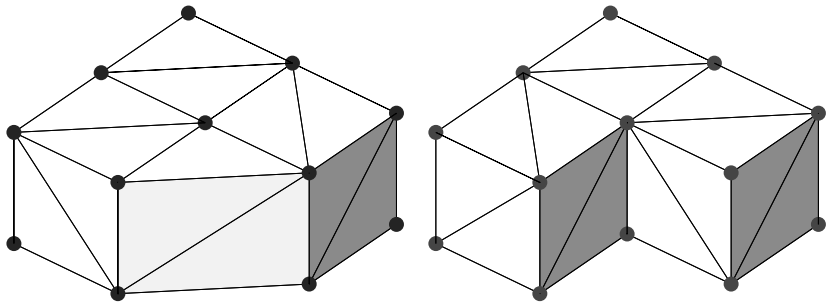
3.7.1. Тетраэдризация Делоне

Напомним, что *тетраэдризацией Делоне* для конечного множества точек $\mathcal{V} = \{v_1, \ldots, v_n\}$ называется такое конформное разбиение выпуклой оболочки множества точек на тетраэдры, что для любого тетраэдра внутри описанной вокруг него сферы нет других точек из $\mathcal{V}$.

Известно много методов построения тетраэдризации Делоне; мы не будем останавливаться на деталях, отсылая читателя к [18]. Отметим лишь, что самый простой итерационный метод по сложности в худшем случае пропорционален n^2, а в среднем пропорционален $n^{3/2}$.

3.7.2. Восстановление геометрии области

На этом этапе у нас имеются тетраэдральная сетка Ω_h со множеством тетраэдров $\mathcal{T}$ и множеством узлов $\mathcal{V}$ и заданный фронт $\mathcal{F}$ с вершинами $\mathcal{V}$. Пример сетки и фронта показан на рис. 3.17. Отметим, что они представляют различные области.



а *б*

Рис. 3.17. Две сетки на одном наборе вершин: *а* — тетраэдральная сетка для выпуклой оболочки точек $\mathcal{V}$, *б* — фронт $\mathcal{F}$ для невыпуклой лакуны

Будем измельчать сетку Ω_h, добавляя новые узлы на ребрах и гранях так, чтобы ребра и грани фронта $\mathcal{F}$ были представлены в новой сетке — либо целиком, либо в виде своих разбиений. Детали приведены в алгоритме 18, использующем рекурсивные процедуры.

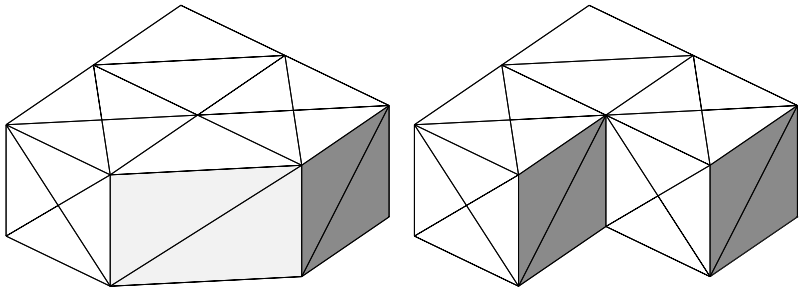
Первым проходом проверяются пересечения ребер фронта $\mathcal{F}$ с гранями $\mathcal{T}$. Для каждого ребра из $\mathcal{F}$ пересекающие его тетраэдры из $\mathcal{T}$ разбиваются таким образом, чтобы ребро было представлено в сетке — либо целиком, либо в виде разбиения. Второй проход аналогичен, проверяются пересечения ребер тетраэдров из $\mathcal{T}$ с гранями $\mathcal{F}$. Для каждой грани из $\mathcal{F}$ пересекающие ее тетраэдры из $\mathcal{T}$ разбиваются таким образом, чтобы грань была представлена в сетке — либо целиком, либо в виде разбиения.

Каждый раз при разбиении тетраэдров из множества $\mathcal{T}$ мы создаем точку пересечения, лежащую на $\mathcal{F}$. При этом разбиваются только элементы из $\mathcal{T}$. В конце $\mathcal{F}$ останется без изменений, а новый набор тетраэдров $\mathcal{T}$ своими гранями будет полностью покрывать грани $\mathcal{F}$.

Удаляя тетраэдры из $\mathcal{T}$, лежащие за границами многогранника $\mathcal{P}$, мы восстановим геометрию области. На рис. 3.18 показана измельченная сетка с набором тетраэдров $\mathcal{T}$ до и после удаления тетраэдров.

Алгоритм 18. Восстановление геометрии области

```
 1: procedure ОБРАБОТКА_РЕБРА(v1, v2)
 2:     Найти тетраэдр с вершиной v1, пересекающий ребро e(v1, v2)
 3:     if тетраэдр найден then
 4:         Построить точку пересечения v на грани тетраэдра
 5:         Разбить соответствующие тетраэдры в T
 6:         ОБРАБОТКА_РЕБРА(v, v2)
 7:     end if
 8: end procedure
 9: procedure ОБРАБОТКА_ГРАНЕЙ(v1, v2)
10:     for all граней f ∈ F do
11:         if грань f пересекает ребро e(v1, v2) then
12:             Построить точку пересечения v
13:             Разбить соответствующие тетраэдры в T
14:             ОБРАБОТКА_ГРАНЕЙ(v1, v)
15:             ОБРАБОТКА_ГРАНЕЙ(v, v2)
16:             Выйти из процедуры
17:         end if
18:     end for
19: end procedure
20: for all ребер e(v1, v2) из F do
21:     ОБРАБОТКА_РЕБРА(v1, v2)
22: end for
23: for all ребер e(v1, v2) из T do
24:     ОБРАБОТКА_ГРАНЕЙ(v1, v2)
25: end for
26: Удалить тетраэдры, лежащие за пределами F
```



а *б*

Рис. 3.18. Восстановление геометрии области: *а* — измельченная сетка, *б* — сетка с правильной геометрией области

Полученная сетка Ω_h будет конформной сеткой для многогранной области $\mathcal{P}$, однако она не будет согласована с ее границей $\mathcal{F}$. На границе Ω_h будут лежать новые точки пересечения, которые мы только что построили (ср. рис. 3.17, *б* и рис. 3.18, *б*). Алгоритм удаления этих точек с границы будет представлен в следующем пункте.

3.7.3. Восстановление следа сетки на границе

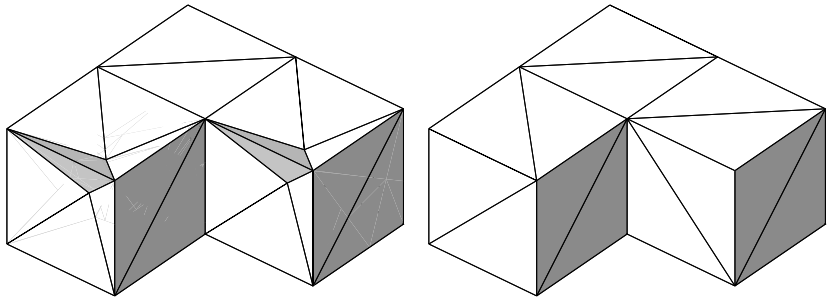
На завершающем этапе с границы сетки удаляются лишние узлы. Это достигается за счет их сдвига внутрь области и заполнения образовавшихся «вмятин» конформными тетраэдральными сетками.

Все лишние точки делятся на точки, лежащие на ребрах $\mathcal{F}$, и на точки, лежащие на гранях $\mathcal{F}$. Избавление от точек в обоих случаях происходит одним и тем же способом. Рассмотрим для определенности второй случай.

Пусть v — узел на грани из $\mathcal{F}$, который надо убрать с поверхности. Предположим для простоты изложения, что фронт $\mathcal{F}$ является простым. С незначительными дополнениями предлагаемый алгоритм применим и в общем случае.

Рассмотрим множество $\sigma(v)$ тетраэдров из $\mathcal{T}$ с общей вершиной v. Граница $\omega_v = \partial\sigma(v)$ этого множества состоит из треугольников двух типов: лежащих на границе $\mathcal{P}$ и лежащих внутри $\mathcal{P}$. При сдвиге вершины v внутрь $\mathcal{P}$ на границе области образуются «вмятины». Каждая из «вмятин» является пирамидой с вершиной v и многоугольными основанием. Она может быть конформно разбита на тетраэдры, для этого многоугольные основания разбиваются диагоналями на треугольники.

Можно показать, что внутри $\sigma(v)$ всегда есть непустое открытое множество точек, куда может быть сдвинут узел v с сохранением невырожденности тетраэдров сетки $\mathcal{T}$. На рис. 3.19 представлена сетка Ω_h с четырьмя видимыми «вмятинами» и с заделанными «вмятинами».



а *б*

Рис. 3.19. Восстановление следа сетки на границе: *а* — сетка с образованными «вмятинами», *б* — сетка с правильным следом на границе

Первый случай, когда узел лежит на ребре $\mathcal{F}$, рассматривается аналогично; единственное отличие — вместо пирамиды «вмятина» будет объединением двух конусов с вершиной v и многоугольными основаниями.

Полный алгоритм действий на этом этапе представлен в алгоритме 19. Сначала рассматриваются узлы на ребрах $\mathcal{F}$, а потом на гранях $\mathcal{F}$.

Алгоритм 19. Восстановление следа сетки на границе

1: **for all** узлов v на ребрах $\mathcal{F}$ и гранях $\mathcal{F}$ **do**
2: Построить множество соседних тетраэдров $\sigma(v)$
3: Из $\omega_v = \partial\sigma(v)$ выделить грани, лежащие на $\mathcal{F}$; пусть $\omega'_v = \omega_v \cap \mathcal{F}$
4: Переразбить ω'_v на треугольники, исключив узел v
5: Добавить во множество $\mathcal{T}$ тетраэдры, образованные треугольниками из ω'_v и узлом v
6: Сдвинуть v внутрь $\sigma(v)$, восстановив невырожденность этих тетраэдров
7: **end for**

Предложенный метод гарантирует построение сетки, согласованной с заданным замкнутым фронтом, при условии, что все вычисления проводятся точно. Качество построенных тетраэдров никак не ограничивается снизу. На практике из-за накопления вычислительных погрешностей метод может породить вырожденные или вывернутые тетраэдры. По этой причине в качестве завершающего этапа необходимо применение алгоритмов, позволяющих значительно улучшить качество сетки и избавиться от вырожденных элементов (см., например, табл. 3.5).

3.7.4. Конечность работы алгоритма

Проведем краткий анализ конечности числа операций в предложенном методе, не вдаваясь в оценки вычислительной сложности.

На первом этапе мы строим тетраэдризацию Делоне Ω_h. Количество узлов в ней совпадает с количеством вершин во фронте $\mathcal{F}$. При наличии оценки количества узлов в сетке Ω_h с помощью (2.2.8)–(2.2.9) мы получаем оценки количеств ребер, граней и тетраэдров.

На втором этапе при восстановлении геометрии в алгоритме 18 используются три основные операции. В строке 21 для каждого ребра из $\mathcal{F}$ новых узлов в Ω_h добавляется не более, чем количество тетраэдров в $\mathcal{T}$ на момент перед выполнением строки 21. Количество ребер в $\mathcal{F}$ конечно, и, следовательно, ограничено количество новых узлов.

В строке 24 для каждого ребра тетраэдров из $\mathcal{T}$ новых узлов в Ω_h добавляется не более, чем количество граней в $\mathcal{F}$. Отметим, что новые ребра при добавлении в Ω_h будут лежать на $\mathcal{F}$, и поэтому сами не будут порождать новых вершин.

Оставшиеся операции: удаление внешних тетраэдров в строке 24 алгоритма 18 и сдвиг лишних точек с границы области в алгоритме 19, не добавляют новых вершин в Ω_h и, значит, будут работать конечное время.

На практике предложенный алгоритм на основе тетраэдризации Делоне используется только для локализованных лакун с небольшим количеством вершин и граней. Количество построенных с его помощью тетраэдров невелико, и время его работы дает незначительный вклад в общее время работы всей цепочки построения тетраэдральной сетки. В п. 3.7.6 будут проведены эксперименты для оценки распределения времени работы между алгоритмом продвигаемого фронта и предложенным алгоритмом.

3.7.5. Улучшение качества полученной сетки

Для построения тетраэдральной сетки применяется комбинация трех алгоритмов. Первый алгоритм продвигаемого фронта используется для построения большей части сетки с возможностью контролирования шага сетки и качества тетраэдров. Второй алгоритм, на основе тетраэдризации Делоне, нужен для разбиения оставшейся части области. Третий, заключительный алгоритм применяется для улучшения качества сетки.

Отметим, что для достижения хорошего качества сетки с использованием только первых двух алгоритмов, как правило, необходима хорошая поверхностная сетка — начальный фронт. Регулярный начальный фронт, подходящим образом подобранная функция, отвечающая за желаемый размер тетраэдров, или правильно подобранный параметр автоматического огрубления являются ключевыми факторами для получения качественной сетки. Тем не менее в рамках полностью автоматической и надежной технологии построения неструктурированных тетраэдральных сеток необходимо дополнительное улучшение качества сетки с помощью третьего алгоритма.

В этом разделе мы не будем обсуждать какой-то конкретный алгоритм для улучшения качества тетраэдральных сеток. Отметим лишь основные идеи, которые могут быть использованы.

Самые простые методы используют сдвиг узлов сетки без изменения ее топологической структуры. Более сложные алгоритмы используют локальные модификации топологии сетки, такие, как разбиение ребра новым узлом и стягивание ребра в один узел, в дополнение к сдвигу узлов. Эти методы, как правило, позволяют значительно улучшить качество сетки. Они будут описаны в гл. 5. Их использование рекомендуется как завершающий шаг в цепочке построения тетраэдральной сетки. В п. 3.7.6 будут приведены эксперименты по измерению качества сетки на разных этапах ее построения.

3.7.6. Результаты экспериментов

В этом пункте будет представлено несколько примеров работы рассмотренных выше алгоритмов. Нас будет в первую очередь интересовать качество сетки $Q(\Omega_h)$ и распределение качества тетраэдров по всей сетке. Качество сетки вычисляется по формулам (2.1.7) и (2.1.8).

Для лучшего отображения результатов мы будем использовать логарифмическую шкалу.

Эксперименты проводились на разных ЭВМ при различных условиях. Однако тестовые запуски в пределах одного эксперимента проводились в одинаковых условиях.

Сначала мы измерим экспериментально скорость работы алгоритма продвигаемого фронта. В качестве области возьмем единичный куб. На поверхности куба с помощью алгоритма продвигаемого фронта, описанного в п. 3.4.3, построим квазиравномерную сетку с шагом h. Полученная поверхностная триангуляция используется как начальный фронт для трехмерного алгоритма продвигаемого фронта при построении квазиравномерной сетки с шагом h.

Уменьшая шаг сетки h, мы будем следить за количеством тетраэдров в сетке N_t и временем ее построения t. Результаты экспериментов представлены в табл. 3.4.

Таблица 3.4
Скорость работы алгоритма продвигаемого фронта в трехмерном пространстве

h	N_t	t, с	N_t/t, с^{-1}	$t/(N_t \log N_t)$, мкс
0,1	3544	0,48	7383	16,6
0,05	27988	4,81	5819	16,8
0,025	204734	43,35	4723	17,3
0,0125	1613980	406,75	3968	17,6

Из результатов расчетов видно, что время работы пропорционально величине $N_t \log N_t$, с относительно небольшим коэффициентом. В таблицу также включен столбец со скоростью построения сетки (тетраэдры в секунду).

Теперь мы рассмотрим пример последовательного применения алгоритма продвигаемого фронта и устойчивого алгоритма на основе тетраэдризации Делоне. Первый алгоритм используется для построения большей части сетки, второй — для разбиения оставшихся лакун.

Рассмотрим начальную поверхностную триангуляцию трехмерной отсканированной модели дракона, изображенной на рис. 3.20, *а*. Она состоит из 54 296 узлов и 108 582 треугольников. Все узлы сетки лежат в плоских срезах, параллельных плоскости xy. В каждом срезе вершины соединены отрезками, образуя контур сечения модели плоскостью. Контуры соседних срезов соединены треугольниками и образуют конформную поверхностную треугольную сетку. Отметим, что эта начальная сетка содержит достаточно большое количество треугольников с плохим качеством.

а *б*

Рис. 3.20. Работа алгоритма продвигаемого фронта на модели дракона: *а* — начальный фронт, *б* — конечный фронт

На первом этапе мы применим метод продвигаемого фронта с автоматическим увеличением шага сетки при продвижении внутрь области. Параметр γ, отвечающий за скорость увеличения шага сетки, выбран равным 1,3. Алгоритм продвигаемого фронта построил сетку в части области, занимающей 99,67 % объема всей области. Всего был построен 250 461 тетраэдр, а в конечном фронте осталось 1718 треугольников, которые изображены на рис. 3.20, *б*. Распределение качества тетраэдров полученной сетки представлено в первой строке табл. 3.5. В столбцах таблицы показано количество тетраэдров, качество которых превышает соответствующее значение и которые не вошли в предыдущие столбцы. Наихудшее качество сетки составляет $Q(\Omega_h) = 3{,}698 \cdot 10^{-7}$, что обусловлено плохим качеством треугольников в начальном фронте.

Полученный фронт автоматически разбивается на несвязные лакуны, и для каждой из них запускается устойчивый метод. Лакуны имеют очень плохую форму, и поэтому устойчивый метод порождает очень плохие элементы. Распределение качества тетраэдров в конечной сетке приведено во второй строке табл. 3.5. Наихудшее качество сетки теперь составляет $Q(\Omega_h) = 1{,}232 \cdot 10^{-14}$, т. е. наихудший тетраэдр —

Таблица 3.5

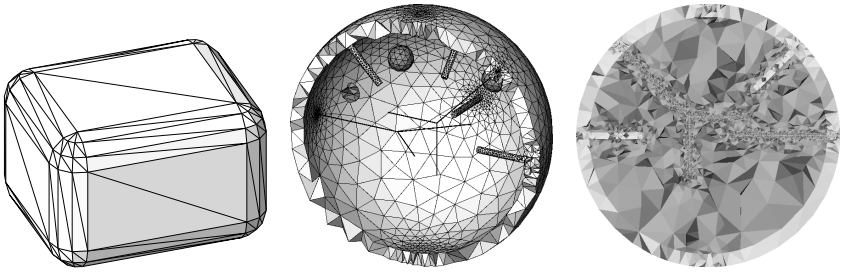
Распределение качества тетраэдров на разных этапах построения сетки для модели дракона: Ф — алгоритм продвигаемого фронта; Д — устойчивый алгоритм на основе тетраэдризации Делоне; ПС — улучшение сетки путем ее перестроения с сохранением заданного следа на границе

	$Q(\Omega^h)$	N_t	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}
Ф	$3{,}7 \cdot 10^{-7}$	250 461	248 359	1923	166	11	1	0	1
Ф+Д	$1{,}2 \cdot 10^{-14}$	254 764	249 674	3428	1020	340	138	71	93
Ф+Д+ПС	$1{,}9 \cdot 10^{-7}$	267 475	258 449	8269	718	30	6	0	3

это сливер. Итоговая тетраэдральная сетка содержит 71 241 узел, 108 582 граничные грани, полностью совпадающие с заданным начальным фронтом, и 254 764 тетраэдра.

Для улучшения качества полученной сетки были использованы методы перестроения сетки (ПС), описанные в гл. 5, при этом граничные грани сетки были зафиксированы. Распределение качества тетраэдров в новой сетке показано в третьей строке табл. 3.5. Тетраэдры с плохим качеством, добавленные в сетку на втором этапе, были удалены. Качество тетраэдральной сетки в этом примере ограничивается плохим качеством начальной поверхностной триангуляции. В новой сетке 75 665 узлов, 108 582 граничные грани и 267 475 тетраэдров. Отметим, что при предварительном улучшении качества начальной поверхностной сетки можно получить тетраэдральную сетку с лучшим качеством [40].

Аналогичные эксперименты были проведены для большого количества разных областей, приведем здесь лишь несколько показательных примеров (см. рис. 3.21). Первый пример — модель `rbox1`, полученная экспортом из САПР триангуляция поверхности для параллелепипеда со скругленными краями. Поверхностная сетка состоит из 104 узлов и 204 треугольников. Качество треугольников в ней очень плохое, имеются резкие перепады размеров треугольников. Построенная



а *б* *в*

Рис. 3.21. Примеры использованных областей: *а* — модель `rbox1`, полученная экспортом из САПР; *б* — сечение начального фронта модели `Lymph`; *в* — сечение построенной сетки для модели `Lymph`

для модели `rbox1` тетраэдральная сетка содежит 138 узлов и 436 тетраэдров, минимальное качество тетраэдров составляет $Q(\Omega_h) = 4{,}430 \cdot 10^{-13}$. Второй пример — модель `drag3`, рассмотренная в этом разделе модель дракона. Третий пример — модель `Lymph`, схематичная модель лимфоузла, созданная в системе САПР Open CASCADE. Модель лимфоузла состоит из нескольких компонентов. Внешняя часть состоит из краевого синуса — толстой сферической оболочки, шести цилиндрических трабекулярных синусов и двух Y-образных кондуитов. Диаметр кондуитов в 1000 раз меньше диаметра лимфоузла. В модель включено 4 сферических фолликула. Остальная часть внутри сферической оболочки лимфоузла представляет корковую и паракортикальную области. Геометрическая модель `Lymph` состоит из 50 узлов, 62 криволинейных ребер и 30 криволинейных граней. Построенная треугольная поверхностная сетка состояла из 24 720 вершин и 95 732 треугольников, а объемная тетраэдральная — из 103 891 вершины и 619 691 тетраэдра. Наихудшее качество тетраэдров составило $Q(\Omega_h) = 3{,}45 \cdot 10^{-12}$.

В табл. 3.6 для каждой модели приведено распределение объема области, для которой построены сетки с помощью метода продвигаемого фронта и устойчивого метода. Для каждого метода приведены также количество построенных тетраэдров и время, которое потребовалось для его работы.

Таблица 3.6

Распределение объема области, количества тетраэдров и времени работы между алгоритмом продвигаемого фронта и устойчивым алгоритмом на основе тетраэдризации Делоне

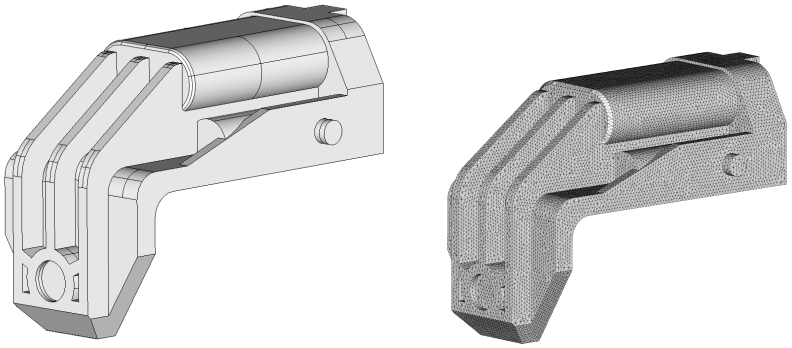
Модель	Продвигаемый фронт			Устойчивый метод		
	%	N_t	t, с	%	N_t	t, с
`rbox1`	97,25	265	0,18	2,75	171	0,03
`drag3`	99,67	250 461	101,70	0,33	4303	0,76
`Lymph`	99,96	617 691	168,93	0,04	2000	0,34

Отметим, что устойчивый метод используется лишь в незначительной части всей области и поэтому занимает малую часть времени по сравнению с алгоритмом продвигаемого фронта.

Построенные для моделей `rbox1` и `Lymph` тетраэдральные сетки имеют плохое качество; это ожидаемый результат, так как представленные в этой главе алгоритмы нацелены в первую очередь на построение топологически корректных сеток. Качество сетки может быть улучшено с помощью методов перестроения сеток, описанных в гл. 5: например, качество новой сетки для модели `rbox1` увеличивается до $Q(\Omega_h) = 7{,}818 \cdot 10^{-8}$, а для модели `Lymph` —

до $Q(\Omega_h) = 7{,}78 \cdot 10^{-2}$. При предварительном улучшении поверхностной сетки (см. § 3.6) и заключительном сглаживании пространственной сетки (см. гл. 5) качество конечной сетки для модели `rbox1` увеличивается до $Q(\Omega_h) = 2{,}18 \cdot 10^{-1}$.

В заключение мы продемонстрируем совместную работу методов построения поверхностных треугольных и объемных тетраэдральных сеток. В качестве примера рассмотрим модель `62_shaver1` с сайта САПР Open CASCADE [19]. Геометрическая модель состоит из 266 узлов, 403 криволинейных ребер и 153 криволинейных граней (см. рис. 3.22, *а*). С помощью метода продвигаемого фронта на поверхности, описанного в § 3.4, была построена квазиравномерная треугольная сетка с 33 176 узлами и 66 348 треугольниками (см. рис. 3.22, *б*).



а *б*

Рис. 3.22. Модель, заданная в САПР: *а* — BREP-модель, *б* — поверхностная квазиравномерная треугольная сетка

Для построения тетраэдральной сетки использовалась комбинация трех методов: алгоритма продвигаемого фронта, устойчивого алгоритма на основе тетраэдризации Делоне и метода улучшения качества сетки, описанного в гл. 5. Первым методом была построена сетка с 158 548 тетраэдрами для части области, занимающей 99,81 % от общего объема модели. Распределение качества тетраэдров полученной сетки представлено в первой строке табл. 3.7. Во фронте осталось 838 треугольных граней; они были переданы на вход второму методу, которым построено еще 1280 тетраэдров. Детальное распределение качества тетраэдров в итоговой сетке приведено во второй строке табл. 3.7. Минимальное качество полученной сетки составило $Q(\Omega_h) = 1{,}117 \cdot 10^{-6}$.

После исправления сетки с помощью третьего метода качество сетки значительно улучшилось, и минимальное качество одного тетраэдра составило $Q(\Omega_h) = 1{,}102 \cdot 10^{-2}$. Итоговая сетка содержит 48 999 узлов, 66 348 граничных граней, полностью совпадающих с гранями, построенными поверхностным генератором сеток, и 176 764 тетраэдра.

Таблица 3.7

Распределение качества тетраэдров на разных этапах построения сетки для модели `62_shaver1`: Ф — алгоритм продвигаемого фронта; Д — устойчивый алгоритм на основе тетраэдризации Делоне; ПС — улучшение сетки путем ее перестроения с сохранением заданного следа на границе

	$Q(\Omega^h)$	N_t	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
Ф	$7{,}03 \cdot 10^{-4}$	158 548	157 091	1430	26	1	0	0
Ф+Д	$1{,}12 \cdot 10^{-6}$	159 828	157 693	1821	244	36	28	6
Ф+Д+ПС	$1{,}10 \cdot 10^{-2}$	176 764	177 690	74	0	0	0	0

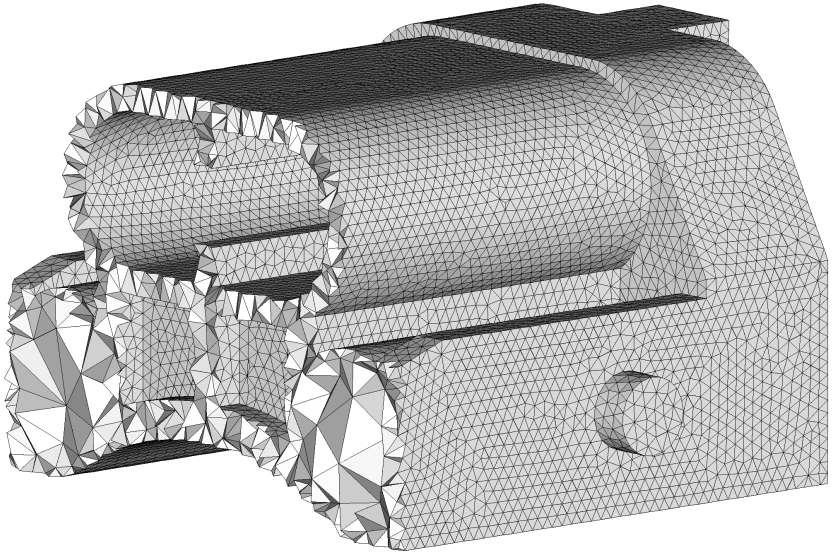


Рис. 3.23. Разрез тетраэдральной сетки для области, заданной в САПР

Распределение качества тетраэдров в новой сетке представлено в третьей строке табл. 3.7. На рис. 3.23 показан срез тетраэдральной сетки.

Общее время работы составило 5 мин 53 с. Распределение времени по этапам следующее: построение поверхностной сетки — 3 мин 23 с, построение объемной сетки — 1 мин 37 с, улучшение сетки — 8 с. Оставшееся время было потрачено на проверку корректности полученной сетки и разные вспомогательные операции.

Глава 4

МНОГОУРОВНЕВОЕ ИЕРАРХИЧЕСКОЕ ИЗМЕЛЬЧЕНИЕ И ОГРУБЛЕНИЕ СЕТКИ

В данной главе будут описаны методы построения динамических двумерных и трехмерных сеток, основанные на многоуровневом иерархическом измельчении и огрублении треугольников и тетраэдров. Представляемый набор алгоритмов дает возможность эффективного многократного локального перестроения сетки при моделировании различных динамических процессов.

§ 4.1. Принципы многоуровневого построения сеток

На практике часто встречаются задачи, связанные с изучением нестационарных процессов с движущимися особенностями. Такими процессами могут быть движение фронта зоны загрязнения, распространение трещины, формирование зоны предразрушения. Для успешного решения таких задач требуются сетки, которые можно было бы быстро перестраивать, чтобы отражать происходящие изменения.

Один из способов состоит в построении в рассматриваемой области последовательности симплициальных (треугольных или тетраэдральных) сеток Ω_h^n, $n = 0, 1, \ldots$, удовлетворяющих следующим свойствам:

— любая сетка Ω_h^n является конформной;

— начальная сетка Ω_h^0 может быть произвольной;

— каждая сетка при $n \geqslant 1$ получается из предыдущей с помощью операций многоуровневого измельчения или огрубления некоторых сеточных элементов. Эти операции порождают последовательность иерархических сеток. Совокупность треугольников или тетраэдров, подвергаемых измельчению или огрублению, определяется особенностями задачи, желаниями пользователя и требованием сохранения конформности получающихся сеток.

Такие сетки называются *иерархическими*. Описанию и детальному анализу алгоритмов модификации иерархических триангуляций и тетраэдризаций будут посвящены § 4.2, 4.3 и 4.4. В § 4.5 рассматривается алгоритм построения сеток, отслеживающих особенности динамических процессов.

Важно отметить, что операции измельчения и огрубления иерархических сеток подчиняются следующей совокупности принципов.

1) Операция измельчения сетки применима к *произвольной* начальной конформной триангуляции или тетраэдризации области.

2) Операция огрубления сетки применима только к тем элементам сетки, которые были ранее измельчены. Таким образом, элемент сетки Ω_h^n, $n > 0$, совпадающий с некоторым элементом Ω_h^0, не может быть огрублен.

3) Описываемые алгоритмы позволяют совершать локальные измельчения и огрубления, что дает возможность строить сетки, сгущающиеся в подобласти.

4) Получаемые в результате сетки обладают тем свойством, что минимальный угол любого треугольника не может быть меньше половины минимального из углов треугольников начальной сетки. Аналогично, минимальный двугранный угол любого тетраэдра не может быть меньше половины минимального из двугранных углов тетраэдров начальной сетки. То же самое верно для плоских углов тетраэдров.

5) Операции измельчения и огрубления выполняются достаточно быстро; число совершаемых действий растет линейно с увеличением количества треугольников или тетраэдров в сетке, к которой применяются рассматриваемые операции.

Принцип 4) особенно важен для численного решения уравнений в частных производных методом конечных элементов.

§ 4.2. Метод бисекций для измельчения триангуляций

Измельчение триангуляции представляет собой процесс последовательного многократного применения операции разбиения определенного треугольника на два треугольника. Такую операцию разбиения будем рассматривать как базисную и называть *бисекцией* треугольника.

Бисекция одного треугольника часто вызывает нарушение конформности сетки на соседнем элементе. Из-за этого увеличивается множество сеточных элементов, которые необходимо измельчить для построения сетки. Как видно из рис. 4.1, при бисекции треугольника Δ_{123} у треугольника Δ_{134} нарушается конформность на ребре e_{13}. Сохранить конформность сетки помогает разбиение треугольника Δ_{134}.

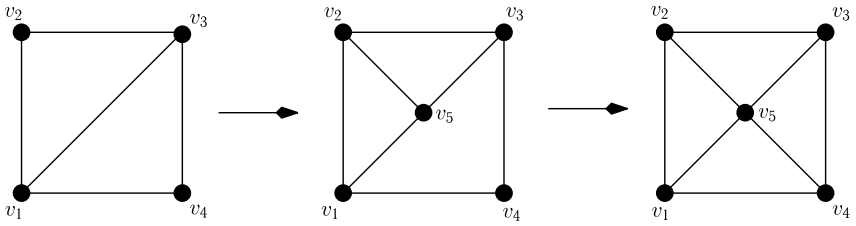


Рис. 4.1. Восстановление конформности сетки

Прежде чем описывать сам алгоритм измельчения треугольной сетки, остановимся подробно на бисекции конкретного сеточного

элемента. Применяются различные методы разбиения треугольника. Мы сконцентрируем внимание на двух из них.

Первый метод был предложен Риварой в [78, 79] и известен как *метод разбиения по наибольшему ребру*. Метод заключается в том, что если треугольник Δ входит во множество сеточных элементов, которые требуется измельчить, или на одном из его ребер не выполняется свойство конформности сетки после бисекции других треугольников, то сначала данный сеточный элемент разбивается отрезком, соединяющим середину наибольшего ребра с противоположной вершиной. Если после совершения этой бисекции свойство конформности сетки все еще нарушается хотя бы на одном ребре исходного треугольника, то середина этого ребра соединяется с серединой наибольшего ребра треугольника Δ.

Принцип работы метода разбиения по наибольшему ребру проиллюстрирован на рис. 4.2. В результате бисекции треугольника Δ_{123} у треугольника Δ_{134} на ребре e_{13} нарушается свойство конформности. Для сохранения конформности треугольник Δ_{134} должен быть измельчен. Сначала его разбиение проводится ребром e_{36}. Однако это не восстанавливает свойство конформности сетки. Лишь последующее разбиение треугольника Δ_{136} ребром e_{56} приводит к желаемому результату.

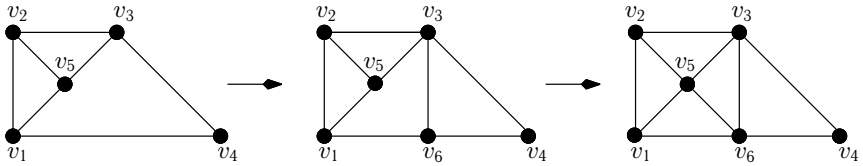


Рис. 4.2. Алгоритм Ривары бисекции треугольника

Другой способ, который был описан в статье Бэнша [31], получил название *метода разбиения по помеченному ребру*. На наш взгляд, он более удобен и прост в реализации при разработке сложного алгоритма построения динамических сеток. Уделим этому методу особое внимание. Каждому треугольнику сетки поставим в соответствие одно из его ребер, которое будем называть *помеченным*. Выбор такого ребра для треугольников начальной сетки осуществляется произвольным образом, независимо от того, какое ребро соседнего треугольника оказалось помеченным. Часто берут самое длинное ребро в треугольнике.

Метод разбиения Бэнша не только разбивает треугольник на два, но и помечает ребра двух новых треугольников. Согласно этому методу, каждый треугольник Δ можно разбить лишь ребром, соединяющим середину помеченного ребра с противоположной вершиной. Для двух новых треугольников помеченными будут ребра, совпадающие с ребрами исходного треугольника Δ, которые не изменились при его бисекции. Принцип работы алгоритма Бэнша представлен на рис. 4.3, где пунктирными линиями в треугольниках выделены помеченные ребра.

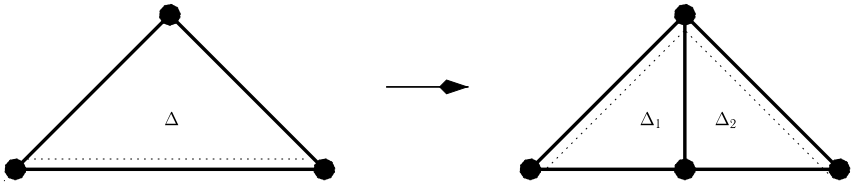


Рис. 4.3. Алгоритм Бэнша бисекции треугольника

На первый взгляд может показаться, что два предложенных алгоритма совершают одно и тоже разбиение. Для опровержения этого предположения приведем пример начальной сетки, состоящей из одного треугольника, и результаты работы двух описанных методов. При этом мы выполним бисекцию каждого из имеющихся треугольников дважды: сначала методом разбиения по наибольшему ребру, затем по помеченному ребру. Так как начальная сетка состоит лишь из единственного треугольника, в процессе бисекций конформность сетки не будет нарушаться. Рис. 4.4 демонстрирует работу алгоритма Ривары, а рис. 4.5 — алгоритма Бэнша. При этом в исходном треугольнике помеченным выбирается ребро, имеющее максимальную длину. Как видим, результирующие сетки оказались различными.

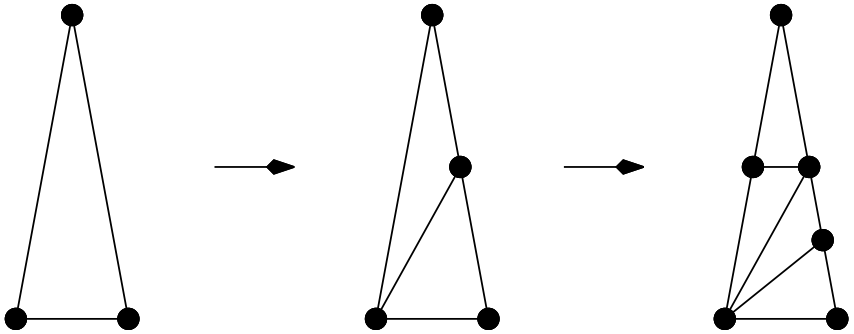


Рис. 4.4. Два шага бисекций треугольников методом Ривары

В дальнейшем, говоря о бисекции определенного треугольника Δ, будем считать, что она производится по методу Бэнша. Введем следующее удобное обозначение для операции бисекции треугольника:

$$\mathcal{B}(\Delta) = (\Delta_1, \Delta_2). \tag{4.2.1}$$

Пусть $\Omega_h(\mathcal{V}, \mathcal{T})$ обозначает треугольную сетку, где $\mathcal{V}$ — совокупность узлов сетки, каждый из которых определяется двумя координатами, а $\mathcal{T}$ — множество треугольников. Пусть $\mathcal{V}(\Delta)$ — множество глобальных индексов вершин треугольника $\Delta(v_1, v_2, v_3)$, а $r(\Delta)$ — локальный номер помеченного ребра в треугольнике Δ:

$$\mathcal{V} = \{v_1, v_2, v_3\}, \qquad 1 \leqslant r(\Delta) \leqslant 3.$$

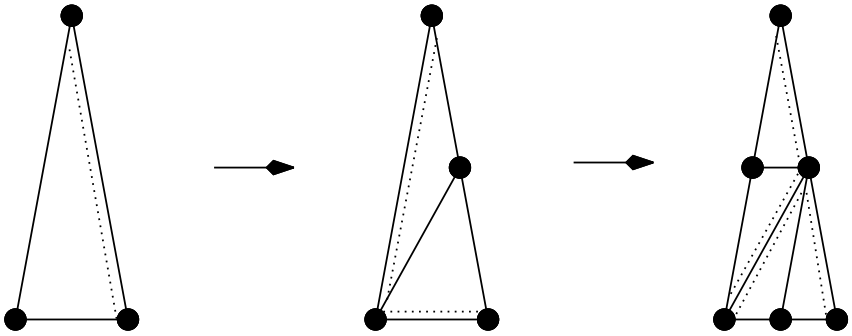


Рис. 4.5. Два шага бисекций треугольников методом Бэнша

Для простоты мы предположим, что локальный номер вершины, противоположной помеченному ребру, также равен $r(\Delta)$. Определим четверку чисел

$$D(\Delta) = (\mathcal{V}(\Delta), r(\Delta)). \tag{4.2.2}$$

Эффективная реализация алгоритма бисекции триангуляции требует для всех сеточных элементов информации о соседних треугольниках. Для каждого треугольника Δ определим множество

$$U(\Delta) = (u_1(\Delta), u_2(\Delta), u_3(\Delta)),$$

где подмножество $u_i(\Delta)$ содержит номера треугольников, которые граничат с Δ по i-му ребру (см. рис. 4.6). Для конформной сетки в $u_i(\Delta)$ содержится только один треугольник. Если i-е ребро лежит на границе области, то положим $u_i(\Delta)$ равным метке соответствующей части границы со знаком минус. Пусть $U(\Omega_h)$ обозначает совокупность $U(\Delta)$ для всех Δ, принадлежащих сетке Ω_h. Множество $U(\Omega_h)$ для конформной сетки лучше всего представимо в виде описанного в гл. 2 структурированного списка $TT(3,N_f)$, где N_f — число треугольников в текущей сетке.

При разбиении одного треугольника меняются данные только о его ближайших соседях. Поэтому стоимость поддержания структурированного списка $U(\Omega_h)$ фактически не влияет на трудоемкость всего процесса измельчения сетки. Однако перед выполнением процедуры бисекции триангуляции нужно определить $U(\Delta)$ для каждого треугольника. Сложность оптимального алгоритма вычисления $U(\Omega_h)$ линейна по отношению к числу треугольников в Ω_h. При построении $U(\Omega_h)$ для каждого узла сетки v будем использовать суперэлемент $\sigma(v)$ как совокупность треугольников, имеющих в качестве одной из своих вершин узел v. Вычисление $U(\Omega_h)$ представлено в алгоритме 20.

Для проверки конформности сетки или ее нарушения на каком-то сеточном элементе достаточно для каждого треугольника определить бинарный массив $C(\Delta)$ из трех чисел $(c_1(\Delta), c_2(\Delta), c_3(\Delta))$. Пусть

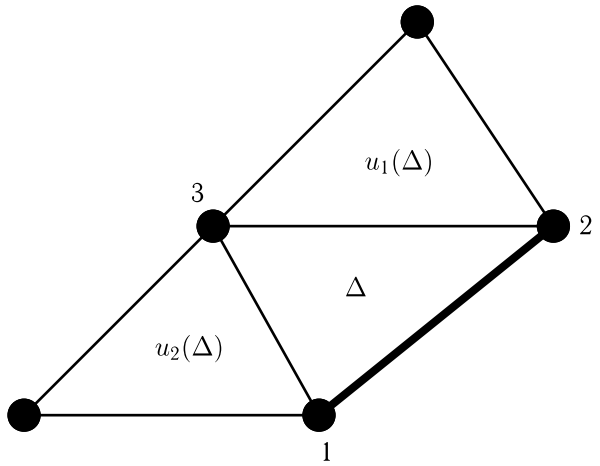


Рис. 4.6. Множество $U(\Delta)$ для граничного треугольника

Алгоритм 20. Построение структурированного списка $U(\Omega_h)$ для триангуляции

```
 1: Для каждого узла v положить σ(v) = ∅
 2: loop по всем треугольникам Δ ∈ Ω_h
 3:     Для всех v ∈ V(Δ) добавить Δ во множество σ(v)
 4: end loop
 5: loop по всем треугольникам Δ ∈ Ω_h
 6:     loop по всем вершинам v_j ∈ V(Δ)
 7:         Определить v_k,v_m ∈ V(Δ), отличные от v_j, v_k ≠ v_m
 8:         Найти множество треугольников L = σ(v_k) ⋂ σ(v_m)
 9:         if L содержит треугольник Δ_1 ≠ Δ then
10:             Положить u_j(Δ) = Δ_1
11:         else
12:             Определить u_j(Δ) по информации о метке граничного
                ребра
13:         end if
14:     end loop
15: end loop
```

значение $c_i(\Delta) = 1$ сигнализирует о проблемах с конформностью на ребре, противоположном вершине i.

Пусть $\mathcal{M}$ — некоторое заданное подмножество $\mathcal{T}$, а выражение

$$\widetilde{\Omega}_h(\widetilde{\mathcal{V}}, \widetilde{\mathcal{T}}) = \texttt{Метод_бисекции}\,(\Omega_h(\mathcal{V}, \mathcal{T}),\mathcal{M}) \tag{4.2.3}$$

будет означать, что сетка $\widetilde{\Omega}_h$ получена из триангуляции Ω_h путем обязательной бисекции всех треугольников из множества $\mathcal{M}$.

При измельчении сетки множества узлов и треугольников постоянно меняются. Процедуру измельчения можно записать в виде алгоритма 21.

Алгоритм 21. Метод бисекции триангуляции $\Omega_h(\mathcal{V}, \mathcal{T})$

1: Выполнить алгоритм 20 для начального определения $U(\Omega_h)$
2: Положить $\widetilde{\mathcal{V}} = \mathcal{V}$, $\widetilde{\mathcal{T}} = \mathcal{T}$
3: **while** множество $\mathcal{M}$ не пусто **do**
4: **loop** по всем треугольникам $\Delta \in \mathcal{M}$
5: Совершить бисекцию $\mathcal{B}(\Delta) = (\Delta_1, \Delta_2)$
6: Добавить Δ_1 и Δ_2 во множество $\widetilde{\mathcal{T}}$
7: Удалить Δ из множеств $\mathcal{M}$ и $\widetilde{\mathcal{T}}$
8: **end loop**
9: **loop** по всем треугольникам $\Delta \in \widetilde{\mathcal{T}}$
10: **if** массив $C(\Delta)$ содержит 1 **then**
11: Добавить Δ во множество $\mathcal{M}$
12: **end if**
13: **end loop**
14: **end while**

В этом алгоритме операция бисекции $\mathcal{B}(\Delta) = (\Delta_1, \Delta_2)$ представляет собой подпрограмму, которая по заданной четверке чисел $D(\Delta)$ (4.2.2) строит $D(\Delta_1)$ и $D(\Delta_2)$ и одновременно обновляет вспомогательную информацию о сетке (см. алгоритм 22). Отметим, что во время работы этого алгоритма конформность сетки может нарушаться и некоторые треугольники могут граничить (через ребро) с двумя треугольниками, обозначенными как Δ_3 и Δ_4.

Проиллюстрируем работу данного алгоритма на примере рис. 4.1, опуская стадию определения соседей для каждого сеточного элемента. Пусть исходная сетка Ω_h состоит из двух треугольников Δ_{123} и Δ_{134}, имеющих общее помеченное ребро e_{13}. Множество $\mathcal{M}$ содержит лишь треугольник Δ_{123}. Представленный алгоритм сначала определяет для Δ_{123} соседей Δ_3 и Δ_4 через ребро e_{13}. В этом случае Δ_3 и Δ_4 совпадают — это треугольник Δ_{134}. Далее совершается бисекция сеточного элемента Δ_{123}, которая приводит к появлению двух новых треугольников Δ_{125} и Δ_{235}. Так как Δ_3 и Δ_4 совпадают, то у треугольника Δ_{134} на ребре e_{13} не было нарушения конформности сетки. Поэтому после разбиения Δ_{123} добавляем точку v_5 в список узлов сетки и налагаем на ребро e_{13} треугольника Δ_{134} условие нарушения конформности сетки. Для треугольников Δ_{125} и Δ_{235} определяем соседей, а у Δ_{134} меняем информацию о соседях через ребро e_{13}. Важно заметить, что в данный момент этот треугольник имеет сразу двух соседей. Далее добавляем Δ_{125} и Δ_{235} во множество треугольников сетки, а сеточный элемент Δ_{123} удаляем из сетки и из множества $\mathcal{M}$. В результате $\mathcal{M}$ становится пустым. После этого, проходя по всем элементам сетки

Алгоритм 22. Бисекция треугольника $\mathcal{B}(\Delta) = (\Delta_1, \Delta_2)$ и модификация вспомогательной информации о сетке

1: По $U(\Delta)$ найти Δ_3 и Δ_4, соседствующие с Δ по ребру $r(\Delta)$
2: **if** $\Delta_3 = \Delta_4$ **then**
3: Задать v как середину стороны $r(\Delta)$
4: **else**
5: Определить v как общую точку Δ_3 и Δ_4, лежащую на ребре $r(\Delta)$
6: **end if**
7: Определить $D(\Delta_1)$ и $D(\Delta_2)$ по $D(\Delta)$
8: Определить $U(\Delta_1)$ и $U(\Delta_2)$ и откорректировать $U(\widetilde{\Omega}_h)$ у соседей Δ
9: **if** $\Delta_3 = \Delta_4$ **then**
10: Элементу массива $C(\Delta_3)$, соответствующему ребру, общему с Δ, приписать 1. Добавить v в $\widetilde{\mathcal{V}}$
11: **end if**
12: Положить $c_{r(\Delta_1)}(\Delta_1) = 1$ и $c_{r(\Delta_2)}(\Delta_2) = 1$, если $c_j(\Delta) = 1$ на соответствующем ребре Δ

с целью обнаружения треугольников, на ребрах которых нарушается свойство конформности, в $\mathcal{M}$ записываем треугольник Δ_{134}.

Так как множество помеченых треугольников опять непусто, то алгоритм снова начинает совершать бисекцию треугольников из $\mathcal{M}$, а именно Δ_{134}. Теперь роль Δ_3 и Δ_4 играют сеточные элементы Δ_{125} и Δ_{235}. Поскольку они различны, то после бисекции Δ_{134} нам не нужно добавлять точку v_5 в список узлов сетки (она там уже есть), и нет необходимости налагать условие нарушения конформности через ребро e_{13}. После бисекции Δ_{134} удаляем его из множества $\mathcal{M}$, которое становится пустым. Получаем требуемую конформную сетку.

Сделаем несколько важных замечаний. Как видно из алгоритма 22, бисекция одного треугольника может повлечь за собой нарушение конформности сетки лишь на одном сеточном элементе. Поэтому если множество $\mathcal{M}$ содержит небольшое число элементов по сравнению с количеством всех треугольников в сетке, к которой применяется процедура бисекции, то сетка будет измельчена локально. Операция нахождения v — общей точки Δ_3 и Δ_4, лежащей на помеченном ребре Δ — может быть осуществлена с учетом информации о соседях Δ_3 и Δ_4. Если разбиваемое ребро принадлежало границе, то при бисекции новые треугольники на ребрах, находящихся на границе, должны сохранить значение $u_i(\Delta)$, ранее присвоенное разбиваемому ребру. Это позволит сохранить информацию о различных граничных метках.

Процесс *многоуровневого измельчения триангуляции* заключается в последовательном применении алгоритма бисекции. Этот процесс

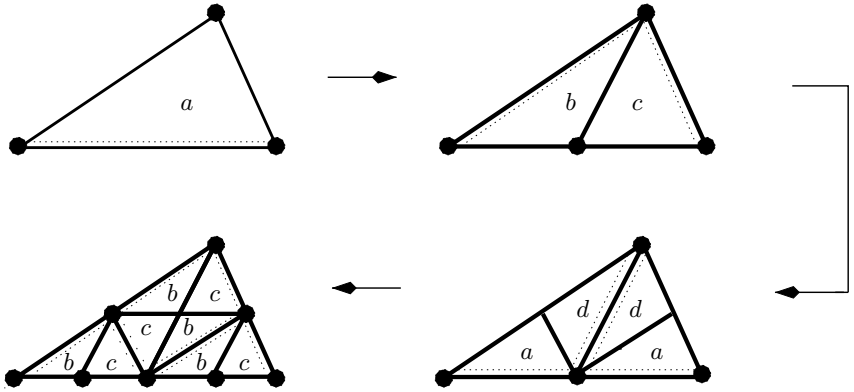


Рис. 4.7. Подобные треугольники, возникающие при одном, двух и трех уровнях равномерного измельчения треугольника

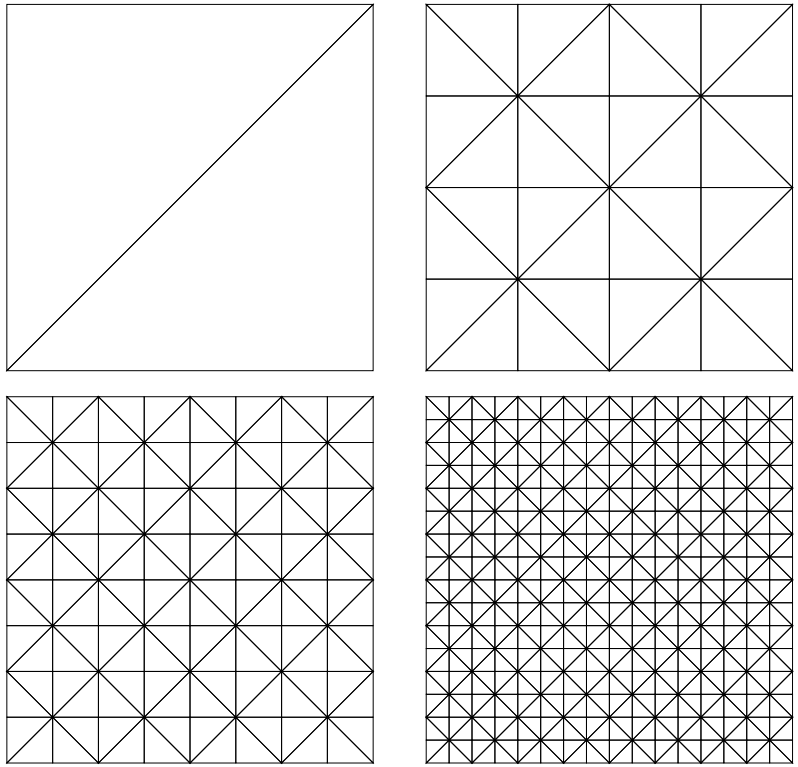


Рис. 4.8. Несколько уровней равномерного разбиения сетки

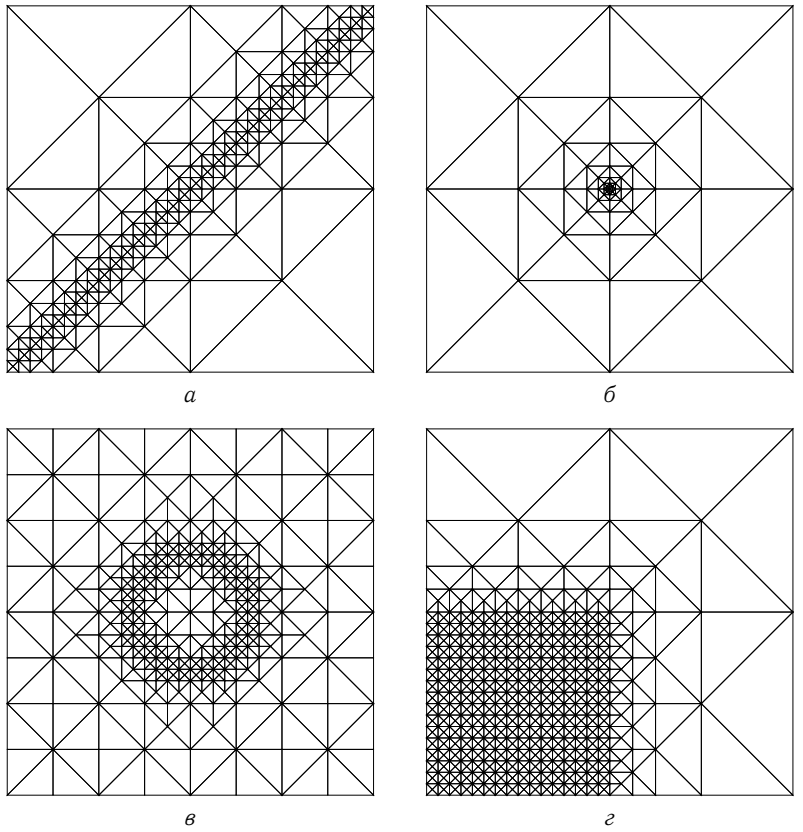


Рис. 4.9. Различные адаптивные измельчения сетки

предполагает наличие процедуры определения множества помеченных треугольников.

Предложеный Бэншем выбор помеченных сторон обеспечивает регулярность треугольников в измельченной сетке. Это вытекает из следующей теоремы [31].

Теорема 4.2.1. *При многоуровневом измельчении триангуляции минимальный угол любого сеточного элемента при любом числе разбиений сетки не меньше половины минимального из углов треугольников начальной сетки.*

Доказательство. Согласно описанию метода разбиения треугольника по помеченному ребру, линия бисекции каждый раз выходит из вершины, появившейся при предыдущем измельчении, и соединяет ее с серединой противоположной стороны. Поэтому если мы рассмотрим несколько уровней измельчения произвольного треугольника, то получим множество треугольников, каждый из которых подобен либо

самому́ исходному треугольнику, либо треугольникам, полученным из рассматриваемого после одного или двух уровней измельчения (см. рис. 4.7, одинаковыми буквами помечены подобные треугольники). Минимальный угол при одном или двух измельчениях не может уменьшиться более, чем в два раза, причем максимальное уменьшение достигается при разбиении равностороннего треугольника. Следовательно, и при большем числе измельчений уменьшения угла треугольника более, чем в два раза, произойти не может. □

Из теоремы 4.2.1 следует, что если минимальный из углов треугольников начальной сетки равен α, то в результирующей сетке любой угол ограничен сверху величиной $\pi - \alpha$.

В заключение мы приведем два примера работы алгоритма. На рис. 4.8 представлены начальная сетка и результирующие триангуляции после 4, 6 и 8 уровней равномерного разбиения; рис. 4.9 иллюстрирует различные виды адаптивных измельчений: к диагонали (*а*), к точке (*б*), к окружности (*в*) и к подобласти (*г*).

§ 4.3. Метод бисекций для измельчения тетраэдризаций

В основе метода бисекций для измельчения тетраэдризаций лежат те же подходы, которые были описаны при исследовании процедуры измельчения треугольной сетки. Основной базисной операцией становится *бисекция* тетраэдра, разбивающая его на два тетраэдра, как показано на рис. 4.10. Однако третье измерение вносит свои особенности в процесс построения сетки.

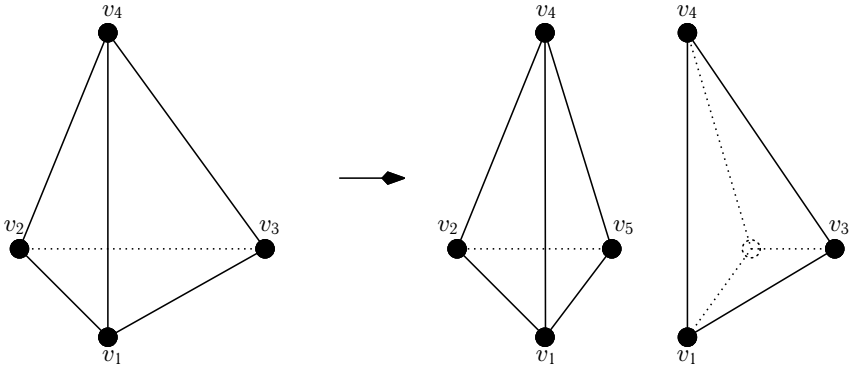


Рис. 4.10. Бисекция тетраэдра

Как и в двумерном случае, сначала подробно опишем метод бисекции отдельного тетраэдра. В трехмерном случае гораздо больше возможностей для задания правил бисекции тетраэдра Δ и определения способов дальнейшего измельчения сеточных элементов, являющихся

результатом бисекции Δ. Любой допустимый набор правил должен гарантировать, что минимальные двугранный и плоский углы в результирующей сетке, при ее многократном измельчении, уменьшаются не более чем в два раза по сравнению с соответствующими минимальными двугранным и плоским углами начальной сетки. Мы остановимся на подходе, который был предложен Арнольдом и соавторами в [28]. Представляемый способ является обобщением метода Бэнша бисекции тетраэдров [31].

Введем следующие обозначения. Каждому тетраэдру Δ поставим в соответствие одно из его ребер, которое будем называть *разбиваемым ребром* $r(\Delta)$, а две грани, содержащие это ребро, — *разбиваемыми гранями*. В каждой из двух граней, не являющихся разбиваемыми, выделим одно из ребер, называя его *помеченным*, и обозначим эти ребра через $l_1(\Delta)$ и $l_2(\Delta)$. Разбиваемое и помеченные ребра будем называть *особыми*. В отличие от двумерного случая, когда помеченное ребро треугольника являлось разбиваемым, здесь разбиваемое и помеченные ребра различны. Тетраэдр Δ будет характеризоваться *бинарным флажком* $s(\Delta) \in \{0, 1\}$, связанным со взаимным расположением особых ребер. После бисекции этот флажок определяет вид дальнейшего разбиения. Величины $s(\Delta)$ должны быть определены для всех элементов начальной сетки. При этом $s(\Delta)$ можно задать произвольным образом с учетом лишь одного ограничения: если особые ребра не лежат в одной плоскости, то $s(\Delta) = 0$. Ниже, при формулировании метода бисекции, мы укажем, как на основании особых ребер тетраэдра Δ определить разбиваемые и помеченные ребра и бинарные флажки ячеек, на которые разбивается тетраэдр Δ.

Метод бисекции тетраэдра Δ заключается в следующем. Сначала Δ разбивается на два тетраэдра Δ_1 и Δ_2 плоскостью, которая проходит через середину разбиваемого ребра и две вершины тетраэдра Δ, не принадлежащие этому ребру (см. рис. 4.10). Заметим, что каждый из тетраэдров Δ_1 и Δ_2 целиком содержит одну из неразбиваемых граней Δ. Помеченное ребро этой грани определим как разбиваемое ребро для соответствующего нового тетраэдра. В результатате для Δ_1 и Δ_2 становится известно, какие из граней являются разбиваемыми.

Осталось определить помеченные ребра для неразбиваемых граней тетраэдров Δ_1 и Δ_2. Пусть v — середина разбиваемого ребра тетраэдра Δ, которая стала общей для Δ_1 и Δ_2. Рассмотрим любую из неразбиваемых граней сеточного элемента Δ_1 или Δ_2. Если эта грань является частью грани тетраэдра Δ, то пометим в ней ребро, противолежащее v. В случае, когда рассматривается общая грань для Δ_1 или Δ_2, она наделяется одинаковым помеченным ребром по следующему правилу. Если $s(\Delta) = 1$, то помеченным будет ребро, которое связывает вершину уже определенного разбиваемого ребра Δ_1 или Δ_2 и точку v, в противном случае помеченным будет ребро, противолежащее v. Для двух новых тетраэдров $s(\Delta_1) = s(\Delta_2) = 1$ лишь в том случае, если разбиваемое и помеченные ребра сеточного элемента Δ

лежали в одной плоскости и выполнялось условие $s(\Delta) = 0$. Таким образом полностью описан способ бисекции конкретного тетраэдра. Эту операцию, по аналогии с двумерным случаем, тоже будем обозначать формулой $\mathcal{B}(\Delta) = (\Delta_1, \Delta_2)$.

Для простоты работы с тетраэдральной сеткой удобно ввести соответствие между вершинами, ребрами и гранями конкретного сеточного элемента. Для тетраэдра, заданного четырьмя узлами сетки v_1, v_2, v_3 и v_4, мы упорядочим шесть ребер следующим образом:

$$e(v_1, v_2),\ e(v_1, v_3),\ e(v_1, v_4),\ e(v_2, v_3),\ e(v_2, v_4),\ e(v_3, v_4),$$

а четыре грани так:

$$f(v_1, v_2, v_3),\ f(v_1, v_2, v_4),\ f(v_1, v_3, v_4),\ f(v_2, v_3, v_4).$$

Теперь, говоря о каком-то локальном номере ребра или грани тетраэдра, можно однозначно установить их вершины.

Пусть $\Omega_h(\mathcal{V}, \mathcal{T})$ обозначает тетраэдральную сетку Ω_h, где $\mathcal{V}$ — совокупность узлов сетки, каждый из которых определяется тремя координатами, а $\mathcal{T}$ — множество тетраэдров. Всякому тетраэдру $\Delta \in \mathcal{T}$ приписывается набор данных из восьми чисел

$$D(\Delta) = (\mathcal{V}(\Delta), r(\Delta), l_1(\Delta), l_2(\Delta), s(\Delta)), \tag{4.3.1}$$

где $\mathcal{V}(\Delta) = (v_1, v_2, v_3, v_4)$ — вектор, состоящий из глобальных номеров четырех узлов сетки, являющихся вершинами Δ, $r(\Delta)$ — локальный номер разбиваемого ребра, $l_1(\Delta)$, $l_2(\Delta)$ — локальные номера помеченных ребер и $s(\Delta)$ — бинарный флажок взаимного расположения особых ребер. Отметим, что

$$1 \leqslant r(\Delta),\ \ l_1(\Delta),\ \ l_2(\Delta) \leqslant 6.$$

Процесс бисекции можно ускорить, если информация о соседних тетраэдрах доступна для всякого сеточного элемента. Каждому тетраэдру Δ поставим в соответствие множество $U(\Delta) = (u_1(\Delta), u_2(\Delta), u_3(\Delta), u_4(\Delta))$, где подмножество $u_i(\Delta)$ содержит номера тетраэдров, которые граничат с Δ по грани, противоположной вершине i. Для конформной сетки в $u_i(\Delta)$ содержится номер только одного тетраэдра. Для граничных граней в качестве $u_i(\Delta)$ можно взять метку границы со знаком минус, по аналогии с тем, как это было предложено в двумерном случае. Пусть $U(\Omega_h)$ обозначает совокупность $U(\Delta)$ для всех Δ, принадлежащих сетке Ω_h. При построении $U(\Omega_h)$ для каждого узла v сетки Ω_h будем использовать суперэлемент $\sigma(v)$ как совокупность тетраэдров, имеющих в качестве одной из своих вершин точку v. Процедура определения $U(\Omega_h)$ может быть реализована со сложностью, линейно зависящей от числа тетраэдров в Ω_h (см. алгоритм 23).

Как и при бисекции треугольной сетки, в трехмерном случае необходима операция проверки конформности сетки. Нарушение конформности легче отслеживать на ребрах, так как измельчение одного

Алгоритм 23. Построение структурированного списка $U(\Omega_h)$ для тетраэдризации

```
 1: Для каждого узла v положить σ(v) = ∅
 2: loop по всем тетраэдрам Δ ∈ Ω_h
 3:     Для каждой v ∈ V(Δ) добавить Δ во множество σ(v)
 4: end loop
 5: loop по всем тетраэдрам Δ ∈ Ω_h
 6:     loop по всем вершинам v_j ∈ V(Δ)
 7:         Определить различные вершины v_k, v_m, v_n ∈ V(Δ), отличные от v_j
 8:         Найти пересечение L = σ(v_k) ⋂ σ(v_m) ⋂ σ(v_n)
 9:         if Δ_1 ∈ L и  Δ_1 ≠ Δ then
10:             Положить u_j(Δ) = Δ_1
11:         end if
12:         if L = {Δ} then
13:             Определить u_j(Δ) по информации о границе
14:         end if
15:     end loop
16: end loop
```

тетраэдра часто влечет за собой нарушение конформности сетки сразу на нескольких сеточных элементах. Для каждого тетраэдра введем целочисленный массив $C(\Delta)$ из шести чисел $(c_i(\Delta), i = 1, ..., 6)$, соответствующий ребрам данного сеточного элемента. Если $c_i(\Delta) \neq 0$, то это означает нарушение конформности на ребре i, т. е. появление в середине ребра тетраэдра Δ нового узла сетки v_0. Это ребро может принадлежать сразу целой совокупности сеточных элементов $\mathcal{W}$. Для сохранения конформности все тетраэдры из $\mathcal{W}$ должны быть измельчены бисекциями, разбивающими рассматриваемое ребро. Поэтому всем сеточным элементам из $\mathcal{W}$ необходима информация об узле v_0. Условие того, что бисекция некоторого тетраэдра и появление нового узла v_0 приводят к нарушению конформности на ребре j тетраэдра Δ, будем задавать в виде $c_j(\Delta) = v_0$. В начальный момент для каждого сеточного элемента все $c_i(\Delta)$ нулевые. Для каждого элемента Δ определим глубину, или уровень измельчения $z(\Delta)$. Как было отмечено в § 4.1, имеется последовательность сеток, каждая из которых получается из предыдущей измельчением или огрублением. Будем считать, что всякий элемент начальной сетки имеет $z(\Delta) = 0$. При разбиении тетраэдра с уровнем z_1 получаем два сеточных элемента с уровнем $z_1 + 1$.

Обозначим процедуру измельчения тетраэдризации Ω_h следующим образом:

$$\widetilde{\Omega}_h(\widetilde{\mathcal{V}}, \widetilde{\mathcal{T}}) = \texttt{Метод_бисекции}(\Omega_h(\mathcal{V}, \mathcal{T}), \mathcal{M}), \tag{4.3.2}$$

где $\mathcal{M}$ — некоторое заданное подмножество $\mathcal{T}$. Равенство (4.3.2) означает, что сетка $\widetilde{\Omega}_h$ получена из тетраэдризации Ω_h путем обязательной бисекции всех сеточных элементов из множества $\mathcal{M}$. Алгоритм 24 измельчения тетраэдральной сетки похож на свой двумерный аналог (алгоритм 21). Как и при бисекции треугольников, совокупность тетраэдров сетки постоянно меняется, а множество узлов увеличивается.

Алгоритм 24. Метод бисекции тетраэдризации $\Omega_h(\mathcal{V}, \mathcal{T})$

Выполнить алгоритм 23 начального определения $U(\Omega_h)$
Положить $\widetilde{\mathcal{V}} = \mathcal{V}$ и $\widetilde{\mathcal{T}} = \mathcal{T}$
while множество $\mathcal{M}$ не пусто **do**
    Найти наименьшее значение $z_{\min}$ из $z(\Delta)$ по всем $\Delta \in \mathcal{M}$
    Сформировать $\widetilde{\mathcal{M}} = \{\Delta \in \mathcal{M} \mid z(\Delta) = z_{\min}\}$
    loop по всем тетраэдрам $\Delta \in \widetilde{\mathcal{M}}$
        Совершить бисекцию $\mathcal{B}(\Delta) = (\Delta_1, \Delta_2)$
        Добавить Δ_1 и Δ_2 во множество $\widetilde{\mathcal{T}}$
        Удалить Δ из множеств $\mathcal{M}$ и $\widetilde{\mathcal{T}}$
    end loop
    loop по всем тетраэдрам $\Delta \in \widetilde{\mathcal{T}}$
        if массив $C(\Delta)$ содержит ненулевое значение **then**
            Добавить Δ во множество $\mathcal{M}$
        end if
    end loop
end while

В этом алгоритме операция бисекции $\mathcal{B}(\Delta) = (\Delta_1, \Delta_2)$ представляет собой подпрограмму, которая в трехмерном случае гораздо сложнее в силу того, что одно ребро может принадлежать сразу целому множеству сеточных элементов (см. алгоритм 25).

Измельчение одного тетраэдра может повлечь за собой нарушение конформности сетки лишь на небольшом числе соседних сеточных элементов. Если множество $\mathcal{M}$ мало́ по сравнению с количеством всех элементов в сетке, то сетка будет измельчена локально.

Алгоритм бисекции тетраэдральной сетки строится с учетом того, чтобы процесс сохранения конформности не приводил к бесконечному циклу, при котором восстановление конформности на одном тетраэдре ведет к нарушению ее на других сеточных элементах. Корректность этого утверждения обосновывается в [28]. Процесс бисекции будет успешен лишь при корректном определении разбиваемых и помеченных ребер для всех элементов начальной сетки. В отличие от начальных условий для измельчения триангуляции, здесь необходима согласованность особых ребер. Поэтому мы подробно остановимся на способе задания разбиваемых и помеченных ребер для произвольной начальной тетраэдризации.

Алгоритм 25. Бисекция тетраэдра $\mathcal{B}(\Delta) = (\Delta_1, \Delta_2)$ и модификация вспомогательной информации о сетке

Определить ребро $r(\Delta)$ из $D(\Delta)$
if $c_{r(\Delta)}(\Delta) = 0$ **then**
    Поставить новый узел v на середину ребра $r(\Delta)$ тетраэдра Δ
else
    $v = c_{r(\Delta)}(\Delta)$
end if
Определить $D(\Delta_1)$ и $D(\Delta_2)$ по $D(\Delta)$
Определить $U(\Delta_1)$ и $U(\Delta_2)$ и изменить $U(\widetilde{\Omega}_h)$ у соседей Δ
if $c_{r(\Delta)}(\Delta) = 0$ **then**
    Найти вершины v_{k_1} и v_{k_2} ребра $r(\Delta)$
    loop по всем тетраэдрам $\widetilde{\Delta} \in \sigma(v_{k_1}) \bigcap \sigma(v_{k_2})$ кроме Δ
        Положить $c_j(\widetilde{\Delta}) = v_0$ для ребра j с вершинами v_{k_1} и v_{k_2}
    end loop
    Добавить v_0 во множество $\widetilde{\mathcal{V}}$
end if
Положить $c_j(\Delta_1) = c_i(\Delta)$, если $c_i(\Delta) \neq 0$ на соответствующем неразбиваемом ребре i тетраэдра Δ
Положить $c_j(\Delta_2) = c_i(\Delta)$, если $c_i(\Delta) \neq 0$ на соответствующем неразбиваемом ребре i тетраэдра Δ

Для начальной сетки мы определим список ее ребер, каждое из которых задается глобальными номерами своих вершин. Упорядочим ребра по возрастанию длины и припишем каждому ребру порядковый номер в новом списке, называемый в дальнейшем его *весом*. Далее, для каждого тетраэдра выберем ребро с наибольшим весом и пометим его как разбиваемое. Затем для каждой неразбиваемой грани находим ребро этой грани, имеющее наибольший вес, и считаем его помеченным. Благодаря этому алгоритму любая грань, принадлежащая двум тетраэдрам, для каждого из них будет иметь одинаковое особое ребро, так как у всякой грани три ребра имеют различные веса. В начальной сетке значения бинарных флажков $s(\Delta)$ можно выбрать, например, нулевыми.

Аналогично алгоритму многоуровневой бисекции триангуляции, можно говорить и о *многоуровневой бисекции тетраэдризации*. При многократном измельчении сетки алгоритм бисекции с введением особых ребер обеспечивает ограниченность снизу двугранного и плоского углов сеточных элементов половинами минимальных двугранных и плоских углов тетраэдров начальной сетки. Это достигается за счет деления граней на разбиваемые и неразбиваемые и специального выбора помеченных ребер, определяющих дальнейший ход бисекции, как обосновано в [28]. Параллелизация алгоритмов многоуровневой бисекции обсуждается в [37, 38].

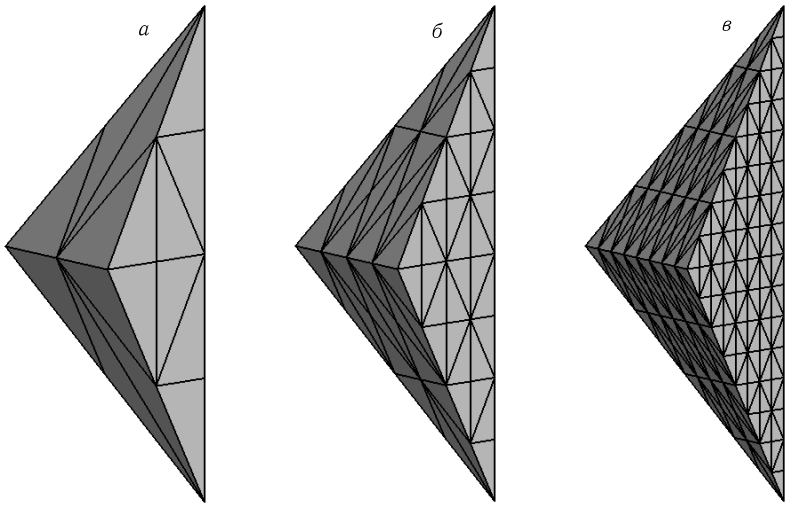


Рис. 4.11. Несколько уровней равномерного измельчения сетки

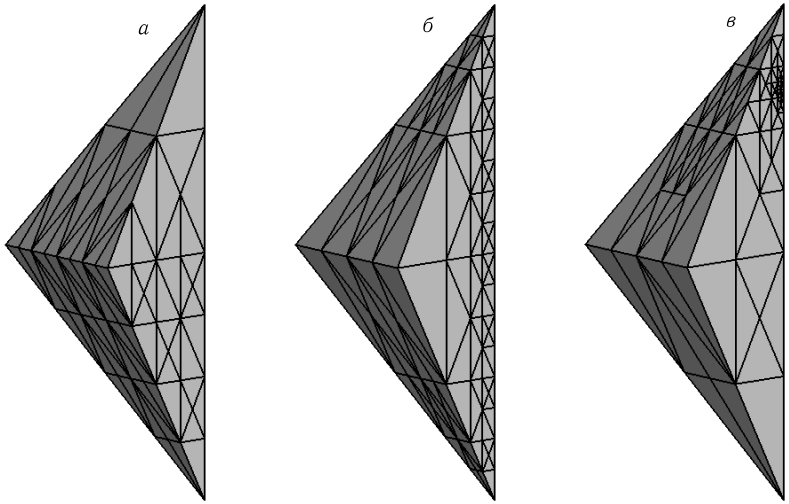


Рис. 4.12. Различные локальные измельчения сетки

В заключение продемонстрируем два примера работы алгоритма бисекции в трехмерном случае. На рис. 4.11 приведены результирующие тетраэдризации после трех (*а*), шести (*б*) и девяти (*в*) уровней равномерного разбиения начальной сетки, состоящей из двух канонических тетраэдров. Рис. 4.12 иллюстрирует различные виды локальных сгущений: к подобласти (*а*), к прямой (*б*) и к точке (*в*).

§ 4.4. Алгоритм многоуровневого огрубления

При изучении динамических процессов важным фактором является быстрое перестроение сетки. Обладая только алгоритмами иерархического измельчения, пришлось бы много раз строить малоразличающиеся сетки, которые учитывают небольшие изменения в динамическом процессе и связаны друг с другом только посредством самой грубой сетки. Подключение алгоритмов огрубления сеток обеспечивает тесную связь между малоразличающимися сетками. В данном параграфе будут описаны алгоритмы огрубления как двумерных, так и трехмерных сеток, однако все встречающиеся проблемы и методы их решения будут проиллюстрированы на примере треугольных сеток.

Алгоритм огрубления сеток, полученных бисекцией треугольников методом Ривары, можно найти в [79]. Здесь мы представим описание огрублений для триангуляций и тетраэдризаций, полученных измельчениями, изложенными в § 4.2 и 4.3.

Цель огрубления заключается в укрупнении некоторых выделенных сеточных элементов путем объединения двух рядом стоящих. Заметим, что объединение можно осуществить по-разному. На рис. 4.13 приведено объединение двух пар треугольников, стоящих на одной горизонтали. Рис. 4.14 иллюстрирует другой вид огрубления. В результате получаем две различные сетки.

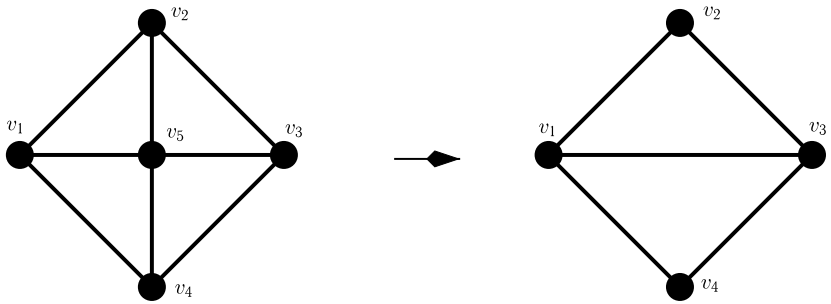


Рис. 4.13. Объединение треугольников по горизонтали

Процесс огрубления будет применяться только к сеткам, полученным в результате метода бисекции. Поэтому основная цель объединения сеточных элементов — восстановление сеток, к которым применялось измельчение. Это можно осуществить, сохраняя историю сеточных преобразований.

При иерархическом измельчении сетки ее элементы подвергаются различному числу операций бисекции. Поэтому простое объединение произвольного треугольника с любым из своих соседей может привести к появлению сеточных элементов, отличных от треугольников, как показано на рис. 4.15.

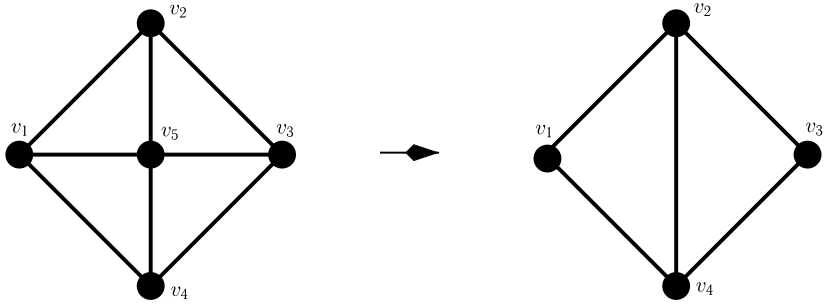


Рис. 4.14. Объединение треугольников по вертикали

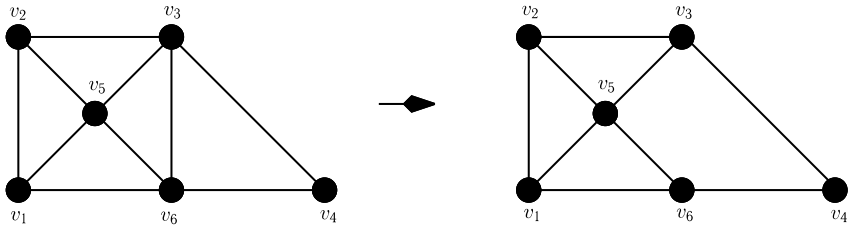


Рис. 4.15. Объединение треугольника с произвольным соседом, ведущее к появлению четырехугольника

Решить эту и другие проблемы поможет описываемый ниже алгоритм огрубления сетки. Его основная цель состоит в том, что, начиная работу с конформной сеткой, являющейся результатом процедуры бисекции, алгоритм огрубления укрупняет заданные сеточные элементы и некоторые другие для сохранения конформности и выдает сетку в таком виде, что к ней снова можно применять либо измельчение, либо огрубление.

В алгоритме бисекции предполагалось, что конформность нарушается на ребре сеточного элемента и может быть восстановлена дополнительным разбиением соседних элементов. Аналогично, при огрублении сетки конформность также может нарушаться на ребре из-за наличия узла v в его середине, как показано на рис. 4.16, но восстанавливаться

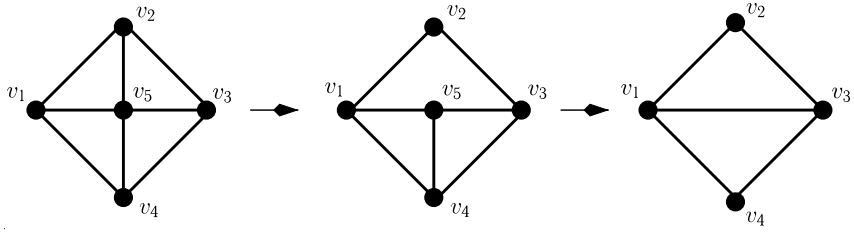


Рис. 4.16. Восстановление конформности при огрублении сетки

она будет за счет огрубления сеточных элементов, вершиной которых является v, и последующим удалением самого узла v. Поэтому будем говорить, что конформность нарушается в узле.

Перейдем к формализации алгоритма огрубления. Пусть Ω_h обозначает треугольную или тетраэдральную сетку. Напомним, что каждый элемент $\Delta \in \Omega_h$ имеет уровень измельчения $z(\Delta)$. Объединить два элемента в один можно лишь в том случае, если они имеют одинаковый уровень, скажем, z_1, тогда результирующий укрупненный сеточный элемент будет иметь уровень $z_1 - 1$. Согласно основным принципам построения сеток, укрупнять элементы начальной сетки нельзя. Это можно сформулировать как запрет на укрупнение элементов с нулевым уровнем: уровень любого треугольника или тетраэдра обязан быть неотрицательным.

Кроме уровня измельчения, каждому элементу Δ поставим в соответствие массив $h_\Delta(z)$, который будет содержать историю модификации данного сеточного элемента. В двумерном случае для каждого уровня z_0 единственный элемент $h_\Delta(z_0)$ определяется следующим образом. Если при бисекции треугольника Δ образовались сеточные элементы Δ_1 и Δ_2, то $h_{\Delta_1}(z_0)$ содержит локальный номер вершины треугольника Δ_1, противоположной ребру, по которому Δ_1 граничит с Δ_2. Как видно из описания бисекции произвольного треугольника, $h_{\Delta_1}(z_0)$ никогда не может совпасть с вершиной, противоположной разбиваемой стороне.

В трехмерном случае $h_\Delta(z_0) = (h^{(1)}_\Delta(z_0), h^{(2)}_\Delta(z_0))$ — это пара локальных вершин сеточного элемента Δ. Когда тетраэдр Δ разбивается на Δ_1 и Δ_2, общая грань двух новых сеточных элементов проходит через середину v разбиваемого ребра Δ. Условимся считать, что $h^{(1)}_{\Delta_1}(z_0)$ является локальным номером вершины Δ_1, соответствующей точке v, а $h^{(2)}_{\Delta_1}(z_0)$ — локальным номером вершины Δ_1, противоположной грани, по которой Δ_1 граничит с Δ_2.

Сначала рассмотрим процесс объединения двух соседних треугольников Δ_1 и Δ_2 в единый сеточный элемент Δ. При этом нам снова понадобится структурированный список $U(\Omega_h)$ соседей треугольников, определенный в § 4.2.

Два треугольника Δ_1 и Δ_2 могут быть объединены в один при выполнении трех условий:

1) $z(\Delta_1) = z(\Delta_2) = z$;
2) если $k = h_{\Delta_1}(z)$, то $u_k(\Delta_1) = \Delta_2$;
3) если $m = h_{\Delta_2}(z)$, то $u_m(\Delta_2) = \Delta_1$.

В этом случае глобальный номер сеточной вершины, соответствущий локальному номеру вершины треугольника Δ_1, противоположной помеченному ребру, совпадает с эквивалентно определенным номером для такой же вершины треугольника Δ_2. При огрублении эта общая вершина v будет удаляться. Так как Δ_1 и Δ_2 имели общее ребро, то у них были две общие сеточные вершины. После удаления узла v другая

общая вершина Δ_1 и Δ_2 и две остальные их вершины формируют новый сеточный элемент Δ. Чтобы корректно определить операцию огрубления, осталось показать, какое ребро в Δ будет помеченным. В качестве $r(\Delta)$ следует брать ребро, противоположное неудаленной общей вершине Δ_1 и Δ_2. Рис. 4.17 иллюстрирует процесс огрубления. Сравнение с рис. 4.3 показывает, что этот процесс является обратным к методу Бэнша. Для простоты операцию огрубления будем обозначать так:

$$\Delta = \mathcal{G}(\Delta_1, \Delta_2). \tag{4.4.1}$$

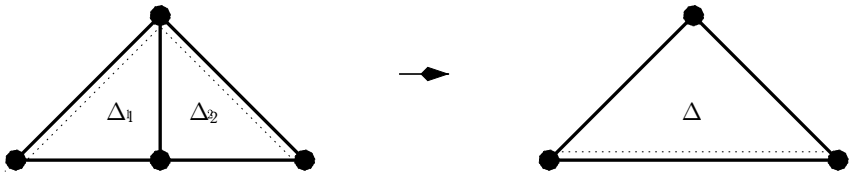


Рис. 4.17. Метод огрубления треугольников

По аналогии с процессом бисекции при огрублении нам понадобится проверка конформности сетки. Эту роль снова возложим на бинарный массив $C(\Delta)$ из трех чисел $(c_1(\Delta), c_2(\Delta), c_3(\Delta))$. Значение $c_i(\Delta) = 1$ сигнализирует о проблемах с конформностью в вершине i, т. е. о наличии треугольника $\widetilde{\Delta}$, у которого вершина, соответствующая i-й вершине треугольника Δ, лежит в середине ребра. В начальный момент для каждого сеточного элемента все $c_i(\Delta)$ нулевые.

Процедуру огрубления триангуляции Ω_h обозначим так:

$$\widetilde{\Omega}_h(\widetilde{\mathcal{V}}, \widetilde{\mathcal{T}}) = \texttt{Метод_огрубления}\,(\Omega_h(\mathcal{V}, \mathcal{T}), \mathcal{M})\,, \tag{4.4.2}$$

где $\mathcal{M}$ — некоторое заданное подмножество $\mathcal{T}$. Равенство (4.4.2) означает, что сетка $\widetilde{\Omega}_h$ получена из сетки Ω_h обязательным огрублением всех элементов из множества $\mathcal{M}$. Алгоритм огрубления применим только к сетке, в которой найдутся элементы, отличающиеся от треугольников начальной сетки, т. е. полученные в результате метода бисекции. Таким образом, в момент начала работы алгоритма мы можем предположить, что для каждого сеточного элемента Δ набор его соседей $U(\Delta)$ уже определен.

Процесс огрубления сетки можно записать в виде алгоритма 26. В данном алгоритме операцию удаления вершины из множества узлов сетки лучше осуществить следующим образом. Введем некоторый вспомогательный бинарный массив меток узлов сетки. В этом массиве число элементов равно количеству узлов в начальной триангуляции Ω_h, задаваемой для огрубления. В начальный момент все элементы этого массива равны 1. Удаление узла из сетки означает обнуление соответствующей компоненты массива меток. Такой подход дает возможность использовать данные об удаленном узле v при дальнейшем огрублении

сеточных элементов, для которых v еще является вершиной. По окончании работы алгоритма 26 массив меток дает информацию о вершинах результирующей сетки. В качестве базовой операции алгоритм 26 использует процедуру огрубления $\Delta = \mathcal{G}(\Delta_1, \Delta_2)$ треугольников Δ_1 и Δ_2, которая детально описана в алгоритме 27.

Алгоритм 26. Метод огрубления триангуляции $\Omega_h(\mathcal{V}, \mathcal{T})$

```
1: Положить 𝒱̃ = 𝒱 и 𝒯̃ = 𝒯
2: loop пока множество ℳ не пусто
3:     Удалить Δ из множества ℳ, если z(Δ) = 0
4:     Найти z_m — наибольшее значение из z(Δ) по всем Δ ∈ ℳ
5:     Сформировать подмножество ℳ̃ = {Δ ∈ ℳ | z(Δ) = z_m}
6:     loop по всем треугольникам Δ1 ∈ ℳ̃
7:         Найти Δ2, который можно объединить с Δ1
8:         Определить узел сетки v, которому отвечает вершина k
           треугольника Δ1, противоположная помеченной стороне
9:         if c_k(Δ1) = 0 then
10:            Используя множество U(Ω̃_h), наложить условие нару-
               шения конформности на все Δ̃ ∈ σ(v)
11:            Удалить v из множества 𝒱̃
12:        end if
13:        if Δ2 найден then
14:            Совершить огрубление Δ = 𝒢(Δ1, Δ2)
15:            Добавить Δ во множество 𝒯̃ и удалить Δ1 и Δ2 из ℳ̃
               и 𝒯̃
16:        end if
17:    end loop
18:    loop по всем треугольникам Δ ∈ 𝒯̃:
19:        if массив c(Δ) содержит 1 then
20:            Добавить Δ во множество ℳ
21:        end if
22:    end loop
23: end loop
```

Продемонстрируем работу алгоритма 26 на примере рис. 4.18. Пусть дана сетка Ω_h, состоящая из пяти треугольников, а множество $\mathcal{M}$ содержит лишь треугольник Δ_{346}. При этом предположим, что уровень измельчения у всех сеточных элементов равен 2, кроме Δ_{346}, у которого он равен 1. Построенное множество $\widetilde{\mathcal{M}}$ совпадает с $\mathcal{M}$. Начинаем работу с сеточным элементом Δ_{346}. Сначала ищем треугольник, с которым его можно было бы объединить. Треугольник Δ_{356} не подходит на эту роль, так как имеет иной уровень измельчения: $z(\Delta_{356}) \neq z(\Delta_{346})$. Поэтому мы можем лишь наложить условие нарушения конформности на все сеточные элементы, содержащие точку v_6.

Алгоритм 27. Огрубление треугольников $\Delta = \mathcal{G}(\Delta_1, \Delta_2)$ и модификация вспомогательной информации о сетке

1: Используя $D(\Delta_1)$, определить вершину v, соответствующую ребру $r(\Delta_1)$
2: Найти общую вершину v_1 для Δ_1 и Δ_2, отличную от v
3: Определить $\mathcal{V}(\Delta)$ по $\mathcal{V}(\Delta_1)$ и $\mathcal{V}(\Delta_2)$
4: Положить $r(\Delta)$ равным локальному номеру вершины Δ, соответствующему v_1
5: Определить $U(\Delta)$ и откорректировать $U(\widetilde{\Omega}_h)$ у соседей Δ_1 и Δ_2

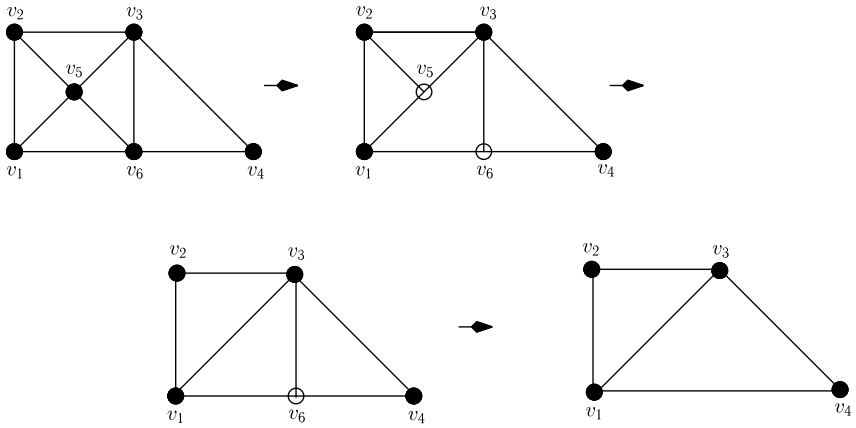


Рис. 4.18. Иллюстрация алгоритма огрубления

Таких треугольников два: Δ_{156} и Δ_{356}. Саму точку v_6 удаляем из списка вершин сетки. Треугольники Δ_{156} и Δ_{356} имеют одинаковый уровень измельчения. Их можно объединить в один треугольник Δ_{136} и удалить Δ_{156} и Δ_{356} из множества $\mathcal{M}$, одновременно объявив вершину v_5 удаленной. При этой операции огрубления конформность сетки начинает нарушаться в точке v_5, поэтому во множество $\mathcal{M}$ добавляются треугольники Δ_{125} и Δ_{235}. Их объединение в сеточный элемент Δ_{123} ведет к сохранению конформности сетки. Все треугольники имеют теперь одинаковый уровень измельчения, но множество $\mathcal{M}$ все еще содержит треугольник Δ_{346}. Теперь его можно объединить с сеточным элементом Δ_{136} и получить треугольник Δ_{134}. В результате имеем огрубленную конформную сетку.

Теперь мы опишем процесс объединения двух соседних тетраэдров Δ_1 и Δ_2 в единый сеточный элемент Δ. Снова используем структурированный список $U(\Omega_h)$ соседей, описанный в § 4.3, и вектор $C(\Delta)$ из четырех чисел $c_1(\Delta)$, $c_2(\Delta)$, $c_3(\Delta)$ и $c_4(\Delta)$, показывающий нарушение конформности в вершине i.

Тетраэдры Δ_1 и Δ_2 можно объединить в один, если выполняются четыре условия:

1) $z(\Delta_1) = z(\Delta_2) = z$;

2) если $k = h^{(2)}_{\Delta_1}(z)$, то $u_k(\Delta_1) = \Delta_2$;

3) если $m = h^{(2)}_{\Delta_2}(z)$, то $u_m(\Delta_2) = \Delta_1$;

4) вершина Δ_1 с номером $h^{(1)}_{\Delta_1}(z)$ совпадает с вершиной Δ_2 с номером $h^{(1)}_{\Delta_2}(z)$.

В отличие от двумерного случая, при огрублении тетраэдра Δ_1 недостаточно информации о том, с каким сеточным элементом Δ_2 его можно объединить. Необходимо знать еще номер вершины v, которая будет удаляться при образовании нового тетраэдра, см. условие 4. Его невозможно определить из $D(\Delta_1)$ и $D(\Delta_2)$. Поэтому при бисекции этот номер сохраняется как значение элементов $h^{(1)}_{\Delta_1}(z)$ и $h^{(1)}_{\Delta_2}(z)$. При огрублении общая вершина v удаляется из сетки. Отрезок, соединяющий $h^{(2)}_{\Delta_1}(z)$ с $h^{(2)}_{\Delta_2}(z)$ и проходящий через $h^{(1)}_{\Delta_1}(z)$, становится в тетраэдре Δ разбиваемым ребром. Помеченными ребрами Δ станут те, которые были разбиваемыми у Δ_1 и Δ_2. Если у Δ_1 и у Δ_2 разбиваемое и помеченные ребра лежат в разных плоскостях, а особые ребра Δ — в одной, то определяем $s(\Delta) = 1$; в противном случае $s(\Delta) = 0$.

Операцию объединения двух тетраэдров будем снова обозначать как (4.4.1), а процедуру огрубления тетраэдризации Ω_h через (4.4.2). Алгоритм огрубления применяется только к тетраэдризации, в которой найдутся элементы, отличающиеся от тетраэдров начальной сетки, т. е. полученные в результате бисекции. В начальный момент работы алгоритма мы можем предположить, что для каждого сеточного элемента Δ список его ближайший соседей $U(\Delta)$ уже определен и для каждого узла известно множество $\sigma(v)$.

Процесс огрубления сетки может быть представлен модифицированным алгоритмом 26. Он использует другую базовую процедуру $\Delta = \mathcal{G}(\Delta_1, \Delta_2)$ объединения тетраэдров Δ_1 и Δ_2 и модификации вспомогательной информации (см. алгоритм 28). Другое отличие модифицированного алгоритма 26 в том, что в трехмерном случае необходимо для каждой вершины v постоянно поддерживать и корректировать информацию о множестве $\sigma(v)$.

Описанные алгоритмы огрубления имеют одну общую особенность. Если некоторый сеточный элемент Δ надо огрубить, а кандидатура для объединения с Δ отсутствует в силу того, что соседи имеют больший уровень измельчения, то алгоритм налагает на соседей требование быть огрубленными и, дождавшись нужного кандидата для Δ, совершает его укрупнение.

Так как огрубление одного сеточного элемента часто приводит к объединению в пары небольшого количества сеточных элементов, то представленные алгоритмы приводят к локальному изменению сетки.

Алгоритм 28. Огрубление тетраэдров $\Delta = \mathcal{G}(\Delta_1, \Delta_2)$ и модификация вспомогательной информации о сетке

1: Найти вершину сетки v, соответствующую $h^{(1)}_{\Delta_1}(z)$
2: Определить $\mathcal{V}(\Delta)$ по $\mathcal{V}(\Delta_1)$ и $\mathcal{V}(\Delta_2)$ с учетом v
3: Присвоить номер $r(\Delta)$ ребру, соединяющему узлы, соответствующие $h^{(2)}_{\Delta_1}(z)$ и $h^{(2)}_{\Delta_2}(z)$
4: Определить $l_1(\Delta)$, $l_2(\Delta)$ и $s(\Delta)$ по $D(\Delta_1)$ и $D(\Delta_2)$
5: Определить $U(\Delta)$ и откорректировать $U(\widetilde{\Omega}_h)$ у соседей Δ_1 и Δ_2
6: Удалить Δ_1 и Δ_2 из множеств $\sigma(\widehat{v})$, где $\widehat{v} \in \mathcal{V}(\Delta_1) \bigcup \mathcal{V}(\Delta_2)$
7: Добавить Δ во множества $\sigma(\widehat{v})$, где $\widehat{v} \in \mathcal{V}(\Delta_1) \bigcup \mathcal{V}(\Delta_2)$ и $\widehat{v} \neq v$

§ 4.5. Алгоритмы построения динамических сеток

Описанные алгоритмы многоуровневого иерархического измельчения и огрубления треугольных и тетраэдральных сеток являются удобными инструментами для построения динамических сеток. Они позволяют быстро перестраивать сетку при моделировании нестационарных процессов, адаптируя ее под особенности меняющегося решения.

Алгоритм 29. Построение динамической сетки

1: Пусть задана квазиравномерная сетка $\Omega^0_h(\mathcal{V}^0,\mathcal{T}^0)$ с шагом d_0
2: Определить помеченные ребра в треугольниках или бинарные флажки, разбиваемые и помеченные ребра в тетраэдрах
3: **loop** по $n = 1, ..., K$
4: Для каждого элемента вычислить функцию $s(\mathbf{x}_\Delta,\ t^n)$
5: Положить $\widetilde{\mathcal{V}} = \mathcal{V}^{n-1}$ и $\widetilde{\mathcal{T}} = \mathcal{T}^{n-1}$
6: Сформировать множество $\mathcal{M} \subset \widetilde{\mathcal{T}}$ из элементов Δ, удовлетворяющих условию $\mathrm{diam}(\Delta) > s(\mathbf{x}_\Delta, t^n)$
7: **if** множество $\mathcal{M}$ не пусто **then**
8: $\Omega^n_h(\mathcal{V}^n, \mathcal{T}^n) =$ `Метод_бисекции` $\left(\Omega^n_h(\widetilde{\mathcal{V}}, \widetilde{\mathcal{T}}), \mathcal{M}\right)$
9: Положить $\widetilde{\mathcal{V}} = \mathcal{V}^n$ и $\widetilde{\mathcal{T}} = \mathcal{T}^n$. Перейти на шаг 6
10: **end if**
11: Выдать $\widetilde{\mathcal{V}}$ и $\widetilde{\mathcal{T}}$
12: Сформировать множество $\mathcal{M} \subset \widetilde{\mathcal{T}}$ из элементов Δ, удовлетворяющих условию $\mathrm{diam}(\Delta) < s(\mathbf{x}_\Delta, t^n)$
13: **if** множество $\mathcal{M}$ не пусто **then**
14: $\Omega^n_h(\mathcal{V}^n, \mathcal{T}^n) =$ `Метод_огрубления` $\left(\Omega^n_h(\widetilde{\mathcal{V}}, \widetilde{\mathcal{T}}), \mathcal{M}\right)$
15: Положить $\widetilde{\mathcal{V}} = \mathcal{V}^n$ и $\widetilde{\mathcal{T}} = \mathcal{T}^n$. Перейти на шаг 12
16: **end if**
17: **end loop**

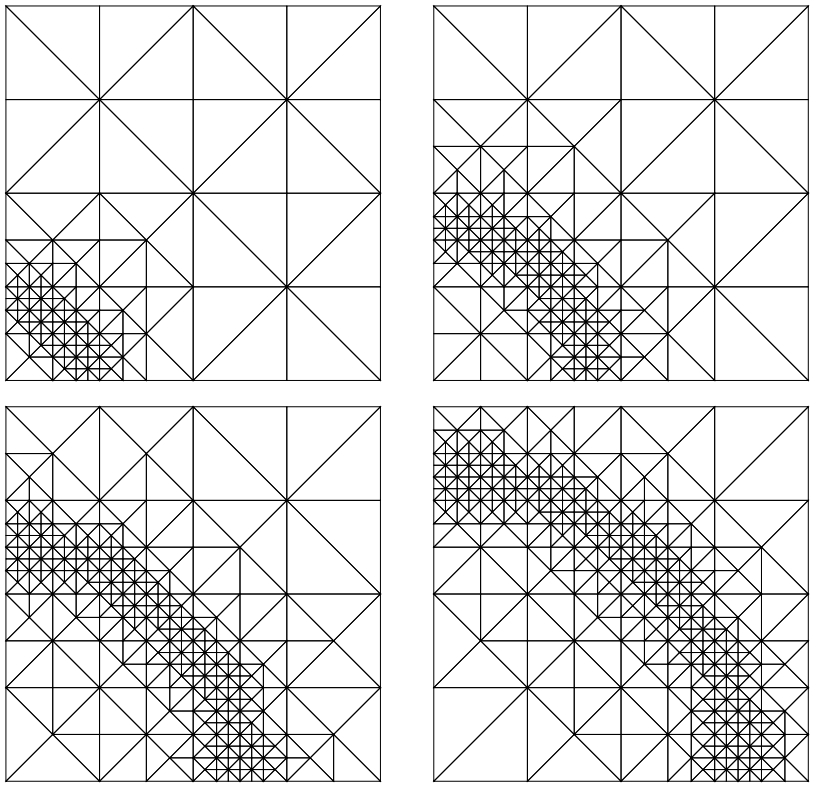


Рис. 4.19. Движение кругового фронта с центром в левом нижнем углу

Рассмотрим построение в области Ω набора сеток Ω^n_h, в которых локальный размер сеточного элемента определяется функцией шага сетки $s(\mathbf{x}, t^n)$. Функция $s(\mathbf{x}, t)$ может быть кусочно-постоянной, например, для отслеживания движения ударной волны в жидкости или распространения трещины в твердом теле. В общем случае $s(\mathbf{x}, t)$ определяется через апостериорные оценки ошибки.

Пусть Ω^0_h — заданная начальная квазиравномерная триангуляция или тетраэдризация области Ω с шагом d_0. Сам процесс создания динамических сеток можно схематично представить в виде алгоритма 29. Принадлежность элемента сетки Δ множеству $\mathcal{M}$ зависит от значения $s(\mathbf{x}_\Delta, t^n)$ в его геометрическом центре $\mathbf{x}_\Delta$. Процесс построения динамической сетки представляет собой многократное применение процедур измельчения и огрубления. Суть процесса состоит в том, что, построив сначала сетку, сгущающуюся к некоторой подобласти, далее можно

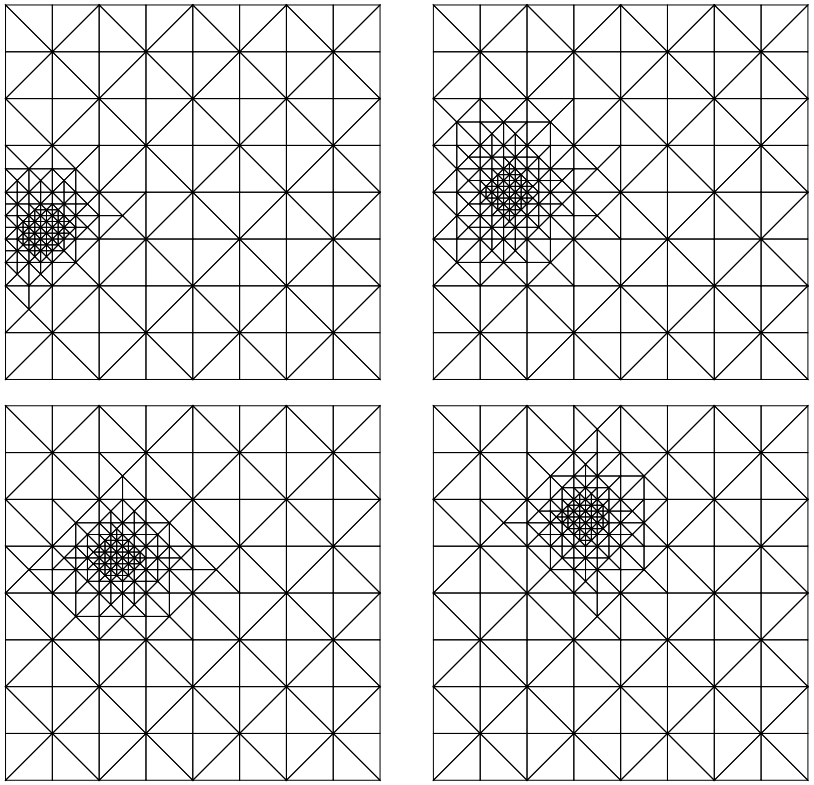


Рис. 4.20. Движение точечной особенности

ее огрубить в области сгущения, а потом снова измельчить в другой подобласти, не перестраивая всю сетку целиком.

Действуя подобным образом, можно смоделировать как движение целого фронта (рис. 4.19), так и перемещение некоторой локальной особенности (рис. 4.20). Использование иерархически вложенных сеток обеспечивает эффективную переинтерполяцию сеточных функций при конечноэлементном решении нестационарных уравнений в частных производных на динамических сетках.

Глава 5

ПЕРЕСТРОЕНИЕ СИМПЛИЦИАЛЬНЫХ СЕТОК ПОСРЕДСТВОМ ЛОКАЛЬНЫХ МОДИФИКАЦИЙ

В данной главе будут описаны методы перестроения треугольных и тетраэдральных сеток. Алгоритмы, описанные в предыдущих главах, ориентированы в основном на построение регулярных сеток. В ряде прикладных задач возникает необходимость строить сетки со специальными свойствами, например, анизотропные сетки, в которых симплексы сильно вытянуты вдоль границы области, фронта ударной волны, или сетки, сгущающиеся к зоне разрушения. Представленный набор алгоритмов дает возможность строить такие сетки путем локальной модификации элементов исходной сетки. В дополнение к этому, технология локальных модификаций может быть использована для распутывания сетки, построения сетки с заданными свойствами и адаптации сетки к заданному сеточному решению.

§ 5.1. Принципы организации алгоритмов

Пусть $\Omega_h(\mathcal{V}, \mathcal{T})$ обозначает расчетную сетку со множеством узлов $\mathcal{V}$ и множеством элементов (симплексов) $\mathcal{T}$. Пусть $N(\Omega_h)$ — число симплексов Δ в этой сетке. Напомним, что мы используем термин «элемент» или «симплекс» как общее обозначение для треугольника и тетраэдра.

Первый важный принцип организации алгоритмов перестроения сетки Ω_h — *локальность* базовых операций. Каждая модификация сетки затрагивает только несколько симплексов, открывая возможность для использования эффективных методов параллелизации. Сетка может быть модифицирована независимо и одновременно в нескольких местах.

Поскольку вычислительная сложность перестроения сетки пропорциональна числу локальных модификаций, распределение работы между несколькими процессорами значительно упрощается. Например, синхронизация процессоров возможна после каждой локальной модификации, т. е. практически сразу же после получения процессором этой команды.

Локальность — это также ключ к разработке надежных и эффективных алгоритмов. Во-первых, можно проанализировать несколько вариантов изменения топологии сетки и выбрать наилучший из них. Во-вторых, наш опыт показывает, что последовательное изменение

сетки в пространственно не связанных местах требует в конечном счете меньшего числа локальных модификаций, чем последовательное изменение сетки вокруг соседних элементов.

Второй важный принцип организации алгоритмов перестроения сетки — *монотонное увеличение качества сетки*. Напомним, что понятия качества сетки $Q(\Omega_h)$ и качества симплекса $Q(\Delta)$ были введены в § 2.1 для регулярных сеток. В § 6.2 эти понятия будут обобщены на случай анизотропных сеток. Поскольку в этой главе точная формула для $Q(\Delta)$ не используется, мы просто предположим, что процедура вычисления качества симплекса известна, причем

$$0 < Q(\Delta) \leqslant 1.$$

Качество симплекса близко к 1, если его форма и размер в окрестности конкретной точки $\mathbf{x} \in \Omega$ вычислительной области близки к параметрам, заданным пользователем. Например, для регулярной сетки естественно потребовать, что $Q(\Delta) = 1$ для равностороннего (правильного) симплекса диаметра $h(\mathbf{x}_\Delta)$, где функция $h(\mathbf{x})$ задается пользователям, а $\mathbf{x}_\Delta$ обозначает геометрический центр симплекса. Качество сетки вычисляется так же, как и раньше:

$$Q(\Omega_h) = \min_{\Delta \in \Omega_h} Q(\Delta).$$

Таким образом, качество сетки совпадает с качеством наихудшего симплекса в сетке.

Аналогичным образом качество произвольного множества симплексов совпадает с качеством наихудшего симплекса в этом множестве. Например, для суперэлемента $\sigma(v)$, образованного симплексами с общей вершиной v, мы имеем

$$Q(\sigma(v)) = \min_{\Delta \in \sigma(v)} Q(\Delta).$$

Напомним, что согласно системе обозначений, введенной в § 2.1, обычный шрифт (v) используется для обозначения вершины как сеточного объекта, а жирный шрифт ($\mathbf{v}$) — для вектора пространственных координат вершины v.

Обозначим через $Q^{(i)}(\Omega_h)$ качество сетки после i локальных модификаций. Принцип монотонного увеличения качества сетки гласит:

$$Q^{(i)}(\Omega_h) \leqslant Q^{(j)}(\Omega_h), \qquad \text{если} \qquad i < j.$$

Отметим, что последняя формула не гарантирует, что качество сетки будет стремиться к 1 с ростом числа локальных модификаций, поскольку увеличение качества сетки возможно только за счет увеличения качества наихудшего элемента сетки. К сожалению, возможны случаи, когда локальные алгоритмы модификации сетки вокруг наихудшего симплекса не могут повысить его качество. В этих случаях локальные алгоритмы применяются ко второму наихудшему симплексу сетки, затем к третьему, и т. д. (см. § 5.2). Данный подход требует упорядочить симплексы по возрастанию их качества с использованием методов, описанных в § 2.3.

§ 5.2. Перестроение триангуляций

Минимальный набор структур данных, определяющих треугольную сетку, включает три структурированных списка для треугольников, граничных ребер и координат узлов. Перестроение сетки потребует несколько вспомогательных структур данных, которые будут описаны ниже.

Надежное перестроение треугольной сетки требует использования как алгоритмов, меняющих топологию сетки, так и алгоритмов, меняющих форму треугольников. Рассмотрим треугольник $\Delta(v_1, v_2, v_3)$ (или Δ_{123} для простоты обозначений) с вершинами v_1, v_2 и v_3 и пять базовых локальных алгоритмов для улучшения его качества. Мы опишем базовые алгоритмы для треугольника Δ, расположенного строго внутри области. Обобщение этих алгоритмов на приграничный треугольник обсуждается в комментариях к алгоритмам.

Пусть $\sigma(\Delta_{123})$ обозначает множество треугольников, включающее Δ_{123} и тех его соседей, которые имеют хотя бы одну общую точку с Δ_{123}:

$$\sigma(\Delta_{123}) = \bigcup_{i=1}^{3} \sigma(v_i).$$

Базовые операции изменяют сетку внутри суперэлемента $\sigma(\Delta_{123})$. Сетка на границе $\sigma(\Delta_{123})$ и в оставшейся вычислительной области не меняется. На основе этого свойства можно построить надежный параллельный алгоритм перестроения треугольной сетки.

Первый базовый алгоритм ставит точку на середину ребра треугольника Δ, разбивая Δ и соседний треугольник пополам. Рис. 5.1 иллюстрирует работу этого алгоритма для внутреннего ребра сетки e_{12}, общего для треугольников Δ и Δ'. Алгоритм 30 описывает первый базовый алгоритм для треугольника Δ.

Алгоритм 30 легко обобщается на случай треугольника Δ с граничными ребрами. Если e — граничное ребро, то соседний треугольник Δ' не существует и $Q_0 = Q(\Delta)$. Разбиение граничного ребра может также потребовать обновления списка граничных ребер.

Возможно, что алгоритм 30 ни разу не выполнит шаг 5. В этом случае мы переходим к следующей базовой операции для улучшения качества Δ.

Модель расчетной области может быть задана с использованием криволинейных границ. В этом случае в базовый алгоритм 30 должна быть включена дополнительная проверка правильности сетки. Допустим, что ребро e_{12} с вершинами $\mathbf{v}_1$ и $\mathbf{v}_2$ лежит на некоторой криволинейной границе (внутренней или внешней), и предположим,

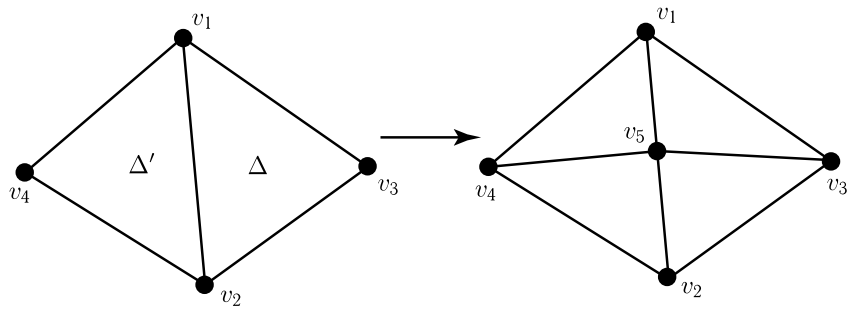


Рис. 5.1. Локальная модификация топологии сетки после добавления узла v_5

Алгоритм 30. Постановка узла на ребро треугольника

1: **loop** по всем ребрам e треугольника Δ
2: Найти соседний треугольник Δ' с вершинами v_1, v_2 и v_4, как показано на рис. 5.1. Определить $Q_0 = \min\{Q(\Delta),\ Q(\Delta')\}$
3: Поставить пробный узел v_5 на середину ребра e и вычислить качества треугольников Δ_{315}, Δ_{235}, Δ_{145} и Δ_{425}. Определить $Q_1 = \min\{Q(\Delta_{315}), Q(\Delta_{235}), Q(\Delta_{145}), Q(\Delta_{425})\}$
4: **if** $Q_1 > Q_0$ **then**
5: Добавить узел v_5 в сетку и заменить треугольники Δ и Δ' треугольниками Δ_{315}, Δ_{235}, Δ_{145} и Δ_{425}. Закончить алгоритм
6: **end if**
7: **end loop**

что каждая точка этого ребра отображается в единственную точку криволинейной границы. На практике для каждой криволинейной границы удобно использовать однопараметрическую функцию $\mathbf{F}(t)$ такую, что

$$\mathbf{v}_1 = \mathbf{F}(t_1), \qquad \mathbf{v}_2 = \mathbf{F}(t_2),$$

а промежуточные значения t определяют точку на криволинейном ребре. Параметры t_1 и t_2 можно хранить в дополнительном структурированном массиве граничных ребер. Координаты средней точки ребра e_{12} и точки, спроецированной на криволинейную границу, вычисляются следующим образом:

$$\widehat{\mathbf{v}}_5 = (\mathbf{v}_1 + \mathbf{v}_2)/2, \qquad \mathbf{v}_5 = \mathbf{F}((t_1 + t_2)/2).$$

Для линейной функции $\mathbf{F}(t)$ точки $\widehat{\mathbf{v}}_5$ и $\mathbf{v}_5$ совпадают. Для гладкой функции $\mathbf{F}(t)$ вторая формула дает узел, расположенный недалеко от $\widehat{\mathbf{v}}_5$. В противном случае могут быть использованы различные методы проецирования средней точки $\widehat{\mathbf{v}}_5$ на заданную кривую.

После того как узел $\mathbf{v}_5$ определен, необходимо проверить правильность сетки. Как показывает рис. 5.2, проекция узла $\widehat{v}_5$ на внешнюю

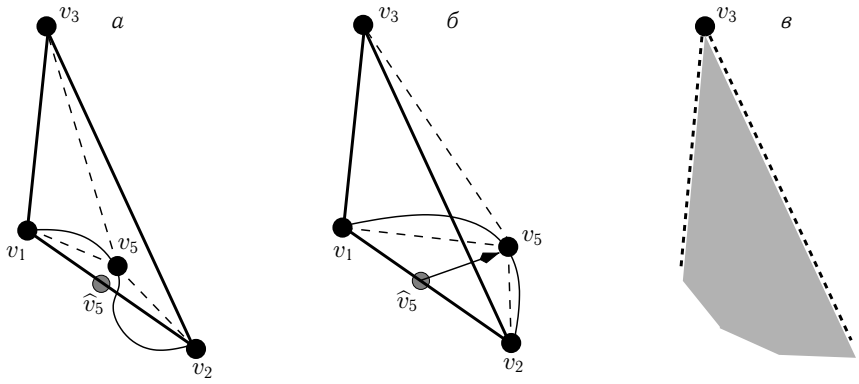


Рис. 5.2. Два случая проекции узла $\widehat{\mathbf{v}}_5$ на внешнюю криволинейную границу (*а, б*) и допустимое множество положений узла $\mathbf{v}_5$ (*в*, закрашенная область)

криволинейную границу может приводить как к правильной сетке (*а*), так и к вывернутой сетке (*б*).

Пусть $S_{\Delta_{315}}$ обозначает алгебраическую площадь треугольника Δ_{315}:

$$S_{\Delta_{315}} = \frac{1}{2}\det\{\mathbf{v}_1 - \mathbf{v}_3,\, \mathbf{v}_5 - \mathbf{v}_3\}, \tag{5.2.1}$$

где $\det(\cdot)$ обозначает детерминант матрицы 2×2, составленной из векторов $\mathbf{v}_1 - \mathbf{v}_3$ и $\mathbf{v}_5 - \mathbf{v}_3$. Геометрическая интерпретация этого детерминанта через векторное произведение

$$\det\{\mathbf{v}_1 - \mathbf{v}_3, \mathbf{v}_5 - \mathbf{v}_3\} = (\mathbf{v}_1 - \mathbf{v}_3) \times (\mathbf{v}_5 - \mathbf{v}_3)$$

показывает, что алгебраическая площадь отличается от обычной площади треугольника только знаком. Оба определения площади совпадают, когда вершины треугольника упорядочены против часовой стрелки (если смотреть на треугольник сверху).

Рассмотрим вспомогательные треугольники, построенные с использованием $\widehat{\mathbf{v}}_5$ вместо $\mathbf{v}_5$. Сетка остается корректной, если алгебраические площади построенных треугольников (Δ_{315} и Δ_{235}) имеют тот же знак, что и алгебраические площади соответствующих вспомогательных треугольников.

Более детальный анализ показывает, почему метод алгебраических площадей приводит к правильной проверке сетки. Рассмотрим вначале случай, когда соседний треугольник Δ' не существует. Прямая, задаваемая ребром e_{23}, разбивает плоскость на две полуплоскости. Пусть π_1 — полуплоскость, содержащая вершину $\mathbf{v}_1$. Эта полуплоскость определяется как множество точек

$$\pi_1 = \left\{\mathbf{x} \in \mathbb{R}^2 \colon\ \det\{\mathbf{x} - \mathbf{v}_2,\ \mathbf{x} - \mathbf{v}_3\} > 0\right\}.$$

Теперь рассмотрим разбиение плоскости на две полуплоскости прямой, задаваемой ребром e_{31}. Пусть π_2 — полуплоскость, содержащая вершину $\mathbf{v}_2$:

$$\pi_2 = \left\{\mathbf{x} \in \mathbb{R}^2 \colon \ \det\{\mathbf{x} - \mathbf{v}_3,\ \mathbf{x} - \mathbf{v}_1\} > 0\right\}.$$

Построим множество $\mathcal{D}$ как пересечение этих полуплоскостей:

$$\mathcal{D} = \pi_1 \cap \pi_2.$$

Множество $\mathcal{D}$ показано на рис. 5.2, *в*. Отметим, что это множество не ограничено.

Лемма 5.2.1. *В условиях точной арифметики проекция* $\mathbf{v}_5 = \mathbf{F}((t_1 + t_2)/2)$ *приводит к правильной сетке, если* $\mathbf{v}_5 \in \mathcal{D}$.

Доказательство. Рассмотрим рис. 5.2 и прямую, задаваемую ребром e_{23}. Эта прямая разбивает плоскость на две полуплоскости. Если узлы v_5 и v_1 находятся в разных полуплоскостях, то треугольник Δ_{153} будет пересекать ребро e_{32}. Таким образом, первое достаточное условие заключается в том, что узел v_5 должен находиться в полуплоскости π_1.

Рассмотрим прямую, задаваемую ребром e_{31}, и треугольник Δ_{325}. Аналогичные рассуждения приводят к заключению, что узел v_5 должен находиться в полуплоскости π_2. Таким образом, если узел v_5 принадлежит множеству $\mathcal{D}$, то внутренности никаких двух треугольников не пересекаются. Следовательно, сетка корректна. □

Для определения правильного знака алгебраической площади мы применим алгоритмы, описанные в § 3.3. Эти алгоритмы используют функцию $d(v_1, v_2, v_3)$, которая возвращает либо правильный знак алгебраической площади $S_{\Delta_{123}}$, либо ноль, если ошибки округления могут привести к неправильному результату. Например, непересекающиеся треугольник и отрезок могут быть неверно восприняты как пересекающиеся. Ошибки такого типа допустимы при перестроении сетки, поскольку они лишь сужают множество $\mathcal{D}$. Поэтому в наихудшем случае эта базовая операция не будет выполняться. Применение функции $d(v_1, v_2, v_3)$ рекомендуется и для других базовых операций, использующих алгебраические площади треугольников.

Лемма 5.2.1 дает достаточное условие для положения узла v_5. Это же условие становится необходимым, когда оба соседних треугольника существуют. В противном случае анализ необходимого условия становится нетривиальным.

Лемма 5.2.2. *Пусть* $\widehat{\Delta}_{315}$ *и* $\widehat{\Delta}_{235}$ *— вспомогательные треугольники, построенные с использованием* $\widehat{\mathbf{v}}_5$ *вместо* $\mathbf{v}_5$. *Тогда сетка остается корректной, если алгебраические площади треугольников* Δ_{315} *и* Δ_{235} *имеют тот же знак, что и алгебраические площади соответствующих вспомогательных треугольников.*

Доказательство. Согласно определению полуплоскости π_1, алгебраическая площадь $S_{\Delta_{235}}$ не меняет знак, когда узел $\mathbf{v}_5$ лежит в этой полуплоскости. Аналогично, алгебраическая площадь $S_{\Delta_{315}}$ не меняет

знак, когда узел $\mathbf{v}_5$ лежит в полуплоскости π_2. Таким образом, сохранение знака алгебраических площадей равносильно выполнению обоих условий, т. е. узел $\mathbf{v}_5$ принадлежит множеству $\mathcal{D}$. Согласно лемме 5.2.1 заключаем, что сетка корректна. □

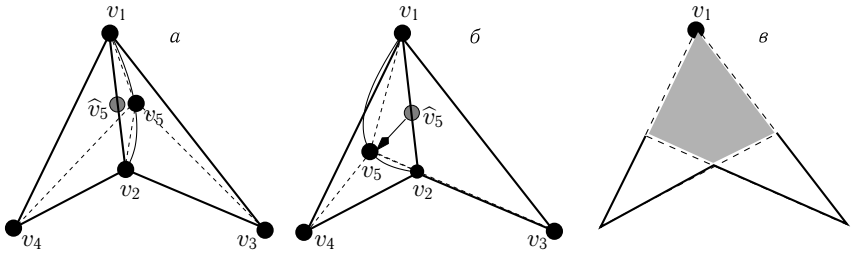


Рис. 5.3. Два случая проекции узла $\widehat{\mathbf{v}}_5$ на внутреннюю криволинейную границу (*а, б*) и допустимое множество положений узла $\mathbf{v}_5$ (*в*)

Рассмотрим случай, когда ребро e_{12} является частью внутренней криволинейной границы (см. рис. 5.3). Определим две дополнительные полуплоскости, связанные с ребрами e_{14} и e_{42}:

$$\pi_3 = \left\{\mathbf{x} \in \mathbb{R}^2 \colon \ \det\{\mathbf{x} - \mathbf{v}_1,\ \mathbf{x} - \mathbf{v}_4\} > 0\right\}$$

и

$$\pi_4 = \left\{\mathbf{x} \in \mathbb{R}^2 \colon \ \det\{\mathbf{x} - \mathbf{v}_4,\ \mathbf{x} - \mathbf{v}_2\} > 0\right\}.$$

Построим множество $\mathcal{D}$ как пересечение четырех полуплоскостей:

$$\mathcal{D} = \pi_1 \cap \pi_2 \cap \pi_3 \cap \pi_4.$$

Множество $\mathcal{D}$ показано на рис. 5.3, *в*. Отметим, что это множество ограничено, в отличие от предыдущего случая. Как показано на рисунке, если объединение треугольников Δ и Δ' не выпукло, то множество $\mathcal{D}$ включает только часть этих треугольников. В противном случае множество $\mathcal{D}$ совпадает с внутренностью выпуклого четырехугольника $\Delta \cup \Delta'$.

Лемма 5.2.1 остается справедливой, но только с новым определением множества $\mathcal{D}$. Лемма 5.2.2 будет также верна, если сравнивать знаки алгебраических площадей четырех вспомогательных треугольников с общей вершиной $\widehat{v}_5$ и четырех треугольников с общей вершиной v_5.

Второй базовый алгоритм применяется к паре треугольников, имеющих общее ребро. Если объединение этих треугольников образует строго выпуклый четырехугольник, то существует второе разбиение этого четырехугольника на два треугольника. Алгоритм 31 определяет, какое из разбиений улучшает качество сетки.

Алгоритм 31 легко обобщается на случай треугольника Δ с ребрами, лежащими на внутренних границах. Поскольку такие ребра не могут быть удалены из сетки, они не рассматриваются в алгоритме.

Алгоритм 31. Замена ребра

1: **loop** по всем ребрам e треугольника Δ
2: Найти соседний треугольник Δ' с вершинами v_2, v_3, и v_4, как показано на рис. 5.4. Если треугольники Δ и Δ' не образуют строго выпуклый четырехугольник, то перейти к следующему ребру
3: Определить $Q_0 = \min\{Q(\Delta),\ Q(\Delta')\}$
4: Вычислить качества треугольников Δ_{124} и Δ_{134}. Определить $Q_1 = \min\{Q(\Delta_{124}), Q(\Delta_{134})\}$
5: **if** $Q_1 > Q_0$ **then**
6: Заменить треугольники Δ и Δ' треугольниками Δ_{124} и Δ_{134}. Закончить алгоритм
7: **end if**
8: **end loop**

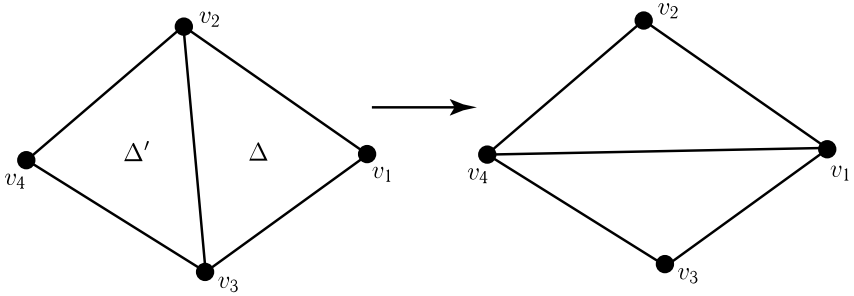


Рис. 5.4. Локальная модификация топологии сетки после замены ребра e_{23} на ребро e_{14}

Операция замены ребра иногда называется *флипом* и имеет другое интересное применение. Флип может быть использован для построения триангуляции Делоне на основе уже существующей сетки. Для этого мы проверяем условие Делоне для пары соседних треугольников Δ и Δ', и если условие не выполняется, то заменяем их на треугольники Δ_{124} и Δ_{134} (рис. 5.4). Теоретическое обоснование такого подхода базируется на следующей теореме [18].

Теорема 5.2.1. *Триангуляцию Делоне можно получить из любой конформной триангуляции, последовательно применяя алгоритм замены ребра к паре треугольников, не удовлетворяющих условию Делоне.*

Третий базовый алгоритм меняет положение вершины сетки. Пусть v_1 — одна из вершин треугольника Δ. Рассмотрим суперэлемент $\sigma(v_1)$, образованный треугольниками с общей вершиной v_1, как показано на рис. 5.5, *а*. На рис. 5.5, *б*, узел v_1 ставится в центр масс суперэлемента $\sigma(v_1)$. Данный подход используется в методах улучшения формы треугольников. В общем случае оптимальное положение узла v_1

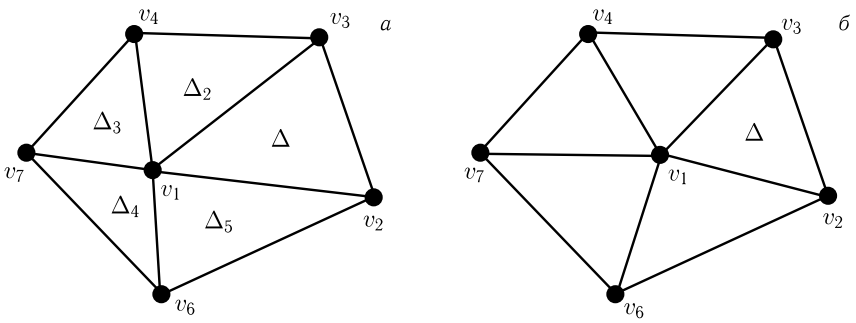


Рис. 5.5. Локальная модификация сетки путем перемещения узла v_1

должно увеличивать качество наихудшего треугольника в суперэлементе $\sigma(v_1)$. Как правило, качество $Q(\sigma(v_1))$ есть нелинейная функция координат вершины v_1. В этом случае поиск оптимального положения v_1 потребует применения достаточно сложных вычислительных методов.

Поскольку в процессе построения оптимальной сетки положение соседних узлов может быть изменено, необходимость в применении точных методов оптимизации отпадает. Алгоритм 32 решает задачу об оптимальном положении вершины приближенно. Пусть $\mathbf{v}_1 = (x_1,\ y_1)$ и, для простоты, $Q_\sigma = Q(\sigma(v_1))$. Обозначим приближенный градиент функции качества Q_σ через $\nabla^h Q_\sigma$. Компоненты $\nabla^h Q_\sigma$ вычисляются на основе конечных разностей:

$$\frac{\partial Q_\sigma}{\partial x} \approx \frac{Q_\sigma(x_1 + \Delta x,\ y_1)}{\Delta x}, \qquad \frac{\partial Q_\sigma}{\partial y} \approx \frac{Q_\sigma(x_1,\ y_1 + \Delta y)}{\Delta y}. \tag{5.2.2}$$

Разумным значением для приращений Δx и Δy является корень квадратный из машинной точности: $\Delta x = \Delta y = \sqrt{\epsilon}$. Для надежности алгоритма необходимо проверить, что вычисленное приращение значительно меньше диаметра суперэлемента, например, меньше 1 % от длины минимального ребра:

$$\Delta x = \Delta y = \min\{\sqrt{\epsilon},\ \min_{v_i, v_j \in \sigma(v_1)} |v_i - v_j|/100\}.$$

Узел v_1 сдвигается в направлении приближенного градиента до тех пор, пока не будет достигнут максимум Q_σ:

$$\mathbf{v}_1 :\ = \mathbf{v}_1 + \alpha \nabla^h Q_\sigma, \qquad \alpha \geqslant 0. \tag{5.2.3}$$

Для поиска этого максимума могут быть использованы различные методы, например, метод бисекций или различные модификации метода Левенберга–Маркуарда (Levenberg–Marquardt) [65]. Отметим, что узел v_1 не может выйти за пределы суперэлемента, поскольку качество Q_σ стремится к нулю, когда v_1 приближается к границе.

Тем не менее перемещение узла даже внутри суперэлемента может привести к запутыванию сетки.

Рассмотрим случай, когда v_1 — внутренний узел сетки с начальным положением $\widehat{\mathbf{v}}_1$ и конечным положением $\mathbf{v}_1$. Пусть $\widehat{\Delta}_i$ — треугольники с общей вершиной $\widehat{\mathbf{v}}_1$, Δ_i — треугольники с общей вершиной $\mathbf{v}_1$, а e_i — граничное ребро суперэлемента $\sigma(v_1)$, принадлежащее треугольнику Δ_i. Это ребро определяет прямую, которая разбивает плоскость на две полуплоскости. Пусть π_i — полуплоскость, которая содержит точку $\widehat{\mathbf{v}}_1$. Определим множество $\mathcal{D}$ следующим образом:

$$\mathcal{D} = \pi_1 \cap \pi_2 \cap \ldots \cap \pi_n,$$

где n — число треугольников в суперэлементе. Если точка $\mathbf{v}_1 \notin \overline{\mathcal{D}}$, то внутренность одного из треугольников Δ_i будет пересекаться с внутренностью треугольника, расположенного по другую сторону от ребра e_i. Отметим, что допустимое множество $\mathcal{D}$ совпадает с суперэлементом $\sigma(v_1)$, когда он выпукл. В противном случае $\mathcal{D}$ является подмножеством $\sigma(v_1)$, аналогично конфигурации, показанной на рис. 5.3.

Алгоритм 32. Сдвиг узла

1: **loop** по всем узлам v_1 треугольника Δ
2: Найти треугольники Δ_1, Δ_2, …, Δ_n, которые образуют суперэлемент $\sigma(v_1)$. Пусть $\Delta = \Delta_1$, как показано на рис. 5.5, где $n = 5$
3: Вычислить приближенный градиент $\nabla^h Q_\sigma$ по формулам (5.2.2)
4: Вычислить максимально возможное смещение узла v_1 в направлении $\nabla^h Q_\sigma$, не нарушающее топологию суперэлемента. Пусть $\alpha = \alpha_{\max}$ для этого экстремального положения
5: Найти значение α в полуоткрытом интервале $[0, \alpha_{\max})$, которое максимизирует качество Q_σ
6: Если $\alpha > 0$, то запомнить новое положение узла v_1 в сетке и закончить алгоритм
7: **end loop**

Отметим, что возможны такие конфигурации суперэлемента $\sigma(v_1)$, когда движение вдоль приближенного градиента не увеличивает качество Q_σ. В этом случае $\alpha = 0$, и алгоритм переходит к следующей вершине треугольника Δ. Если $\alpha = 0$ для всех вершин Δ, то мы применяем следующую базовую операцию к этому треугольнику.

Максимально возможное смещение вершины v_1 в направлении $\nabla^h Q_\sigma$, не нарушающее топологию суперэлемента $\sigma(v_1)$ (см. шаг 4 алгоритма 32), требует найти пересечение прямой, задаваемой этим направлением, с множеством $\mathcal{D}$. Другой метод основан на сравнении знаков алгебраических площадей треугольников Δ_i и $\widehat{\Delta}_i$. Они имеют одинаковые знаки, если вершина $\mathbf{v}_1$ находится внутри $\mathcal{D}$.

Алгоритм 32 легко обобщается на случай треугольника с вершинами, лежащими на внутренних и на внешних границах. Такие узлы сетки могут двигаться только по граничным ребрам. Движение по криволинейным ребрам должно сопровождаться проверкой правильности сетки. Как и раньше, для этого достаточно контролировать сохранение знака алгебраической площади треугольников Δ_1, …, Δ_n.

Четвертый базовый алгоритм удаляет ребро из сетки. Алгоритм 33 можно интерпретировать следующим образом. Начнем смещать вершины ребра навстречу друг другу, возможно, с разными скоростями. Когда вершины сливаются в одну, топология сетки меняется. Как показано на рис. 5.6, две пары ребер сливаются, каждая в одно ребро, и два треугольника исчезают из сетки.

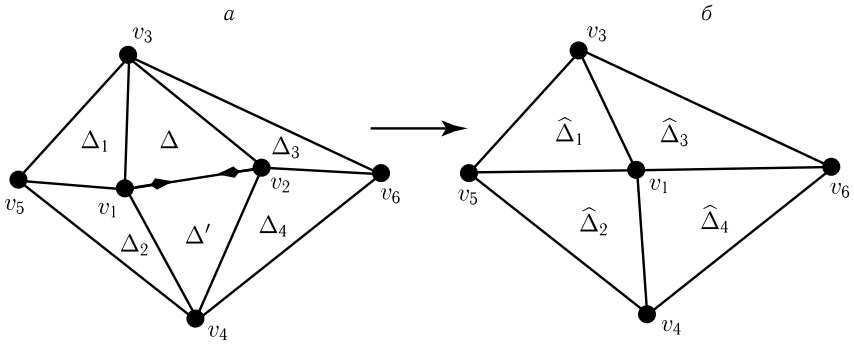


Рис. 5.6. Локальная модификация сетки путем удаления ребра e_{12}

Алгоритм удаления ребра является самым сложным из рассмотренных базовых алгоритмов, поскольку он изменяет большее число треугольников. Тем не менее существует простой метод проверки правильности модифицированной сетки. Неудивительно, что он опять основан на проверке алгебраических площадей модифицированных треугольников. Модифицированная сетка остается правильной, если алгебраические площади треугольников сохраняют знак. Рассмотрим, например, треугольник Δ_{263} (рис. 5.6, *а*), который переходит в треугольник Δ_{163} (рис. 5.6, *б*). Алгебраические площади этих треугольников положительны.

Обобщение алгоритма 33 на случай треугольника с граничным ребром e_{12} требует дополнительного анализа. Удаление этого ребра из сетки может привести к существенному изменению границы. Даже незначительное локальное изменение границы может накапливаться от операции к операции. Это следует иметь в виду при разработке численных методов, в которых сохранение площади вычислительной области или ее подобласти особенно важно.

Удаления граничного ребра сетки также разумно избегать, когда удаляется единственное ребро, представляющее важную часть

Алгоритм 33. Удаление ребра

1: **loop** по всем ребрам e треугольника Δ
2: Найти соседний треугольник Δ' с вершинами v_1, v_2 и v_4, как показано на рис. 5.6
3: Найти треугольники Δ_1, Δ_2, …, Δ_n, которые: (а) отличны от треугольников Δ, Δ'; (б) входят в суперэлемент $\sigma(v_1)$ или $\sigma(v_2)$
4: Определить новое общее (виртуальное) положение $\widehat{v}_1$ вершин v_1 и v_2: $\widehat{\mathbf{v}}_1 = \frac{1}{2}(\mathbf{v}_1 + \mathbf{v}_2)$
5: Определить новые (виртуальные) треугольники $\widehat{\Delta}_1$, $\widehat{\Delta}_2$, …, $\widehat{\Delta}_n$, сдвигая вершины v_1 и v_2 исходных треугольников в $\widehat{v}_1$
6: Если алгебраические площади треугольников Δ_i и $\widehat{\Delta}_i$ имеют разные знаки хотя бы для одного i, то удалить все виртуальные объекты и перейти к следующему ребру
7: Определить качества $Q_0 = \min\{Q(\Delta_1), Q(\Delta_2), \ldots, Q(\Delta_n)\}$ и $Q_1 = \min\{Q(\widehat{\Delta}_1), Q(\widehat{\Delta}_2), \ldots, Q(\widehat{\Delta}_n)\}$
8: **if** $Q_1 > Q_0$ **then**
9: Удалить вершину v_2 и треугольники Δ, Δ' из сетки; заменить вершину v_1 на $\widehat{v}_1$ и треугольники Δ_i на $\widehat{\Delta}_i$. Закончить алгоритм
10: **end if**
11: **end loop**

границы. Такие ситуации часто возникают при использовании грубых сеточных аппроксимаций заданной границы и при построении сеток, сильно вытянутых в одном направлении.

Четвертый шаг алгоритма 33 требует выбрать общую вершину $\widehat{v}_1$ для v_1 и v_2. С одной стороны, естественно рассмотреть задачу об оптимальном положении вершины $\widehat{v}_1$. С другой стороны, поскольку аналогичная задача рассматривается в базовом алгоритме сдвига вершины, усложнение алгоритма 33 не требуется. Оптимальное положение вершины $\widehat{v}_1$ может быть достигнуто путем последовательного применения двух базовых алгоритмов. Поэтому, мы предлагаем рассматривать только одно возможное положение этой вершины: $\widehat{\mathbf{v}}_1 = \frac{1}{2}(\mathbf{v}_1 + \mathbf{v}_2)$.

Пятый базовый алгоритм удаляет из сетки узел вместе с содержащими его треугольниками. Пустота, созданная в сетке, заполняется меньшим числом треугольников, для построения которых используются только существующие узлы сетки. На рис. 5.7 показано, что удаление внутренней вершины v_1 приводит к уменьшению числа треугольников ровно на два.

Алгоритм 34 представляет собой частный случай алгоритма 33, где ребро e_{15} удаляется путем смещения узла v_1 в узел v_5. Такая

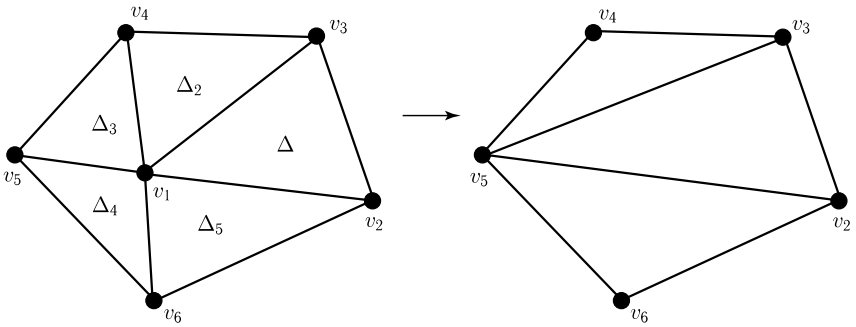


Рис. 5.7. Первая локальная модификация сетки путем удаления узла v_1

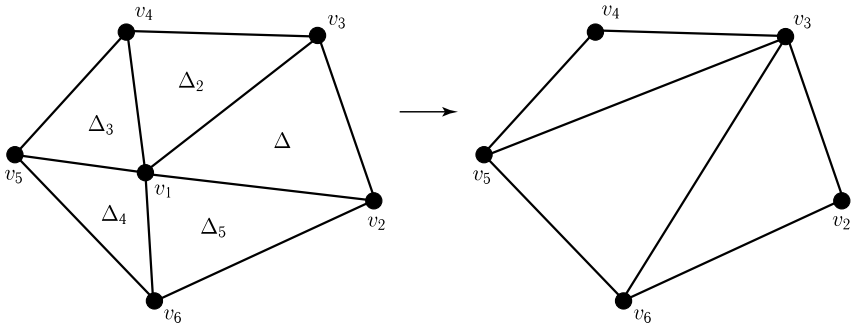


Рис. 5.8. Вторая локальная модификация сетки путем удаления узла v_1

точка зрения излагается в [35] и позволяет существенно упростить программную реализацию этого алгоритма. Мы различаем алгоритмы 34 и 33, поскольку новая триангуляция суперэлемента $\sigma(v_1)$ на шаге 4 алгоритма 34 не единственна. Альтернативная новая триангуляция показана на рис. 5.8 и соответствует смещению узла v_1 в узел v_3. Заметим, что данная базовая операция работает быстрее операции удаления ребра.

Остановимся более подробно на сохранении топологии границ между различными материалами. Пусть на рис. 5.7 треугольники Δ, Δ_2 и Δ_3 принадлежат одному материалу, а треугольники Δ_4 и Δ_5 — другому. Это означает, что ребра e_{23}, e_{34} и e_{45} должны оставаться границей первого материала после операции удаления узла v_1, т. е. этот узел может быть смещен только в узел v_5 или в узел v_2.

Комбинация базовых алгоритмов отрывает широкие возможности для перестроения расчетных сеток с использованием разнообразных критериев, описанных в гл. 6. Рассмотрим, какие структуры данных требуются для эффективной реализации базовых алгоритмов.

Во-первых, необходимо поддерживать список треугольников, упорядоченных по возрастанию их качества. Базовые алгоритмы меняют позиции треугольников в этом списке, удаляют треугольники из списка

Алгоритм 34. Удаление узла

1: **loop** по всем вершинам v треугольника Δ
2: Пусть треугольники $\Delta_1, \Delta_2, \ldots, \Delta_{n-1}$ образуют суперэлемент $\sigma(v)$ с вершинами $v_2, v_3, \ldots, v_n$, как показано на рис. 5.7, где $\Delta = \Delta_1$ и $n = 6$
3: **loop** $i = 2, \ldots, n$
4: Разбить суперэлемент $\sigma(v)$ на виртуальные треугольники $\Delta'_1, \ldots, \Delta'_{n-3}$, соединяя граничную вершину v_i с остальными граничными вершинами, которые не связаны с v_i ребрами
5: Если знаки алгебраических площадей треугольников Δ_k и Δ'_l с одним и тем же граничным ребром суперэлемента различны хотя бы для одного k, то перейти к следующей граничной вершине
6: Если ребра виртуальных треугольников не аппроксимируют исходные границы, то перейти к следующей вершине
7: Вычислить локальные качества $Q_0 = \min\{Q(\Delta_1), \ldots, Q(\Delta_{n-1})\}$ и $Q_1 = \min\{Q(\Delta'_1), \ldots, Q(\Delta'_{n-3})\}$
8: **if** $Q_1 > Q_0$ **then**
9: Удалить вершину v из сетки. Заменить треугольники $\Delta_1, \Delta_2, \ldots, \Delta_{n-1}$ треугольниками $\Delta'_1, \Delta'_2, \ldots, \Delta'_{n-3}$. Закончить алгоритм
10: **end if**
11: **end loop**
12: **end loop**

и добавляют новые треугольники. Эти элементарные операции требуют алгоритмов быстрого поиска внутри динамического списка, описанных в § 2.3. Для списка, состоящего из N_f треугольников, каждая элементарная операция требует $O(\log_2 N_f)$ арифметических действий. Качество сетки — монотонная функция от числа успешно реализованных базовых алгоритмов. К сожалению, качество конкретно взятого треугольника может сильно колебаться. Это является основной причиной отсутствия более эффективных методов поиска внутри динамического списка.

Во-вторых, для каждого треугольника Δ необходимо быстро находить треугольники в суперэлементе $\sigma(\Delta)$. Для этого применяются несколько оптимальных методов с арифметической сложностью $O(1)$. Самый простой метод состоит в построении структурированного списка $U(\Omega_h)$ ближайших соседей для каждого треугольника сетки и его модификации после каждого базового алгоритма. Начальное построение такого списка требует $O(N_f)$ арифметических действий. Детальное описание алгоритма заполнения списка $U(\Omega_h)$ представлено в § 4.2. Каждая базовая операция изменяет всего несколько треугольников, поэтому обновление этого списка имеет оптимальную сложность. Поиск

треугольников, входящих в суперэлемент $\sigma(\Delta)$, начинается с треугольника Δ. После этого в уже существующий непустой список треугольников мы добавляем их ближайших соседей из списка $U(\Omega_h)$, а затем соседей соседей, которые имеют общую вершину с Δ. Данный процесс повторяется до тех пор, пока такие соседи существуют. Несложно заметить, что данный метод имеет оптимальный порядок вычислительной сложности, если количество треугольников в суперэлементе $\sigma(\Delta)$ ограничено.

В-третьих, базовые алгоритмы меняют число основных сеточных объектов: узлов, треугольников и граничных ребер. Эти объекты представлены структурированными списками (см. § 2.3). Добавление нового объекта в структурированный список равносильно добавлению новой строки в двумерный массив. Удаление объекта предполагает сдвиг на одну строку всех объектов, расположенных ниже удаленного объекта. Это очень трудоемкая операция, и ее следует избегать. Вместо этого мы будем хранить дополнительный список удаленных объектов, т. е. номера соответствующих строк в структурированном массиве. При добавлении нового объекта мы сперва проверяем наличие свободных мест в структурированном списке, а затем заполняем их. При отсутствии свободных мест новый объект добавляется в конец списка. После перестроения сетки свободные места в структурированном списке заполняются, например, путем перенумерации всех сеточных объектов.

Основные этапы перестроения сетки представлены в алгоритме 35. Обратим внимание на шаг 10 этого алгоритма, который выполняется, когда ни один из базовых алгоритмов не смог увеличить качество треугольника Δ. Сдвиг указателя на одну позицию означает, что данный треугольник временно выпадает из анализа (отложен на хранение в корзину). За это время треугольники в его окрестности могут измениться. Более того, их изменение может затронуть и сам треугольник Δ. Если этого не произойдет, то алгоритм вернется к треугольнику Δ, когда указатель будет установлен опять на наихудший треугольник в сетке, т. е. $k = 1$.

Алгоритм 35 требует выбрать четыре параметра Q_0, N_1, N_2 и N_3. По определению, желаемое конечное качество сетки $Q_0 \leqslant 1$. В гл. 6 мы рассмотрим несколько примеров с различными определениями качества $Q(\Delta)$ и сформулируем критерии выбора Q_0. Максимальное допустимое число базовых операций N_1 контролирует вычислительную сложность алгоритма и играет важную роль в разработке параллельных методов перестроения сетки (см. § 5.4). Для последовательного алгоритма достаточно выбрать число N_1, которое в несколько раз больше ожидаемого числа треугольников в конечной сетке. Максимальный допустимый размер N_2 корзины отложенных на хранение треугольников зависит от числа треугольников в начальной и конечной сетках. Рекомендуемое значение N_2 равняется приблизительно 5–10 % от среднего числа треугольников в сетке. Максимальное допустимое

Алгоритм 35. Перестроение сетки

1: Вычислить качества треугольников и создать из них блочно упорядоченный список. Инициализировать вспомогательные структуры. Выбрать желаемое качество Q_0 для конечной сетки, максимально допустимое число базовых операций N_1, максимально допустимый размер корзины N_2, максимально допустимое число корзин N_3, установить указатель $k = 1$ и счетчик $m = 1$
2: **loop** от 1 до N_1
3: Взять k-й наихудший треугольник Δ из списка
4: Если $k = 1$ и $Q(\Delta) \geqslant Q_0$, то завершить алгоритм
5: Вычислить суперэлемент $\sigma(\Delta)$
6: Применить к Δ базовые алгоритмы (в произвольном порядке) до первого успешного алгоритма
7: **if** успешный алгоритм найден **then**
8: Изменить сетку и обновить список качеств треугольников и другие вспомогательные списки
9: **else**
10: Передвинуть указатель $k := k + 1$. Если $k > N_2$ или k больше числа треугольников в сетке, положить $k = 1$ и $m := m + 1$
11: Если $m > N_3$, перейти на шаг 14
12: **end if**
13: **end loop**
14: Обновить структурированные списки вершин, треугольников и граничных ребер

число корзин N_3 позволяет алгоритму завершить быстро работу, когда дальнейшее увеличение качества сетки невозможно. Например, геометрические особенности расчетной области (острые углы, узкие участки и т. п.) могут налагать ограничения на качество треугольников. При этом корзины отложенных треугольников будут содержать одни и те же треугольники. Мы предполагаем, что число особенных треугольников мало́, и используем постоянное значение N_3. Отметим, что $N_2 N_3$ должно быть больше среднего числа треугольников в сетке.

На первый взгляд кажется странным, что алгоритм 35 не требует, в качестве входного параметра, желаемого количества треугольников в конечной сетке. На самом деле мы предполагаем, что эта информация содержится в определении качества сетки и качества треугольника, что налагает определенные требования на формирование $Q(\Delta)$. Другими словами, $Q(\Omega_h) = 1$ только тогда, когда сетка Ω_h содержит желаемое число треугольников.

Алгоритм 35 примени́м для построения расчетных сеток на основании очень грубой начальной сетки. Например, квазиравномерная

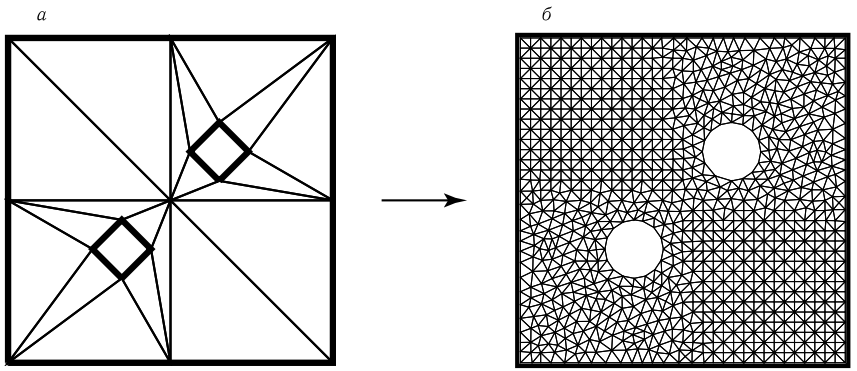


Рис. 5.9. Построение квазиравномерной сетки на основании грубой начальной сетки

сетка, показанная на рис. 5.9, *б*, была построена из сетки на рис. 5.9, *а*, содержащей всего лишь 20 треугольников. Начальная сетка — это минимальное представление модели квадрата с двумя круглыми отверстиями. Отметим очень грубую начальную аппроксимацию отверстий. Каждое из них представлено четырьмя криволинейными ребрами.

Параметр Q_0 был выбран таким образом, чтобы качество равнобедренного прямоугольного треугольника было меньше Q_0. В ряде методов конечных элементов треугольники с углами, не превышающими 90°, позволяют строить численные схемы с дополнительными свойствами, такими, как дискретный принцип максимума.

Отметим, что надежное перестроение расчетных сеток требует всех пяти базовых алгоритмов. Мы проиллюстрируем это утверждение на примере модели, показанной на рис. 5.9, *б*. Начальная квазиравномерная сетка содержит 1827 треугольников. Попробуем перестроить ее в расчетную регулярную сетку с тем же количеством треугольников, которая сгущается к двум отверстиям. Качество треугольника $Q(\Delta)$ определяется таким образом, что его диаметр увеличивается прямо пропорционально минимальному расстоянию от центров отверстий. Казалось бы, что этого можно достигнуть, поставив несколько дополнительных узлов на границы отверстий и сдвинув остальные вершины сетки, применяя базовый алгоритм 32. Тем не менее после 13 000 итераций алгоритма 35 начальная сетка практически не изменилась. Конечная сетка показана на рис. 5.10, *а*. Если мы добавим базовые алгоритмы 30, 33 и 34, которые добавляют узел и удаляют ребро или узел, то получим сетку, очень похожую на сетку, показанную на рис. 5.10, *б*. Для построения этой сетки алгоритму 35 потребовалось 47 287 итераций. Как мы и предполагали, 97 % итераций пришлось на сдвиг узлов сетки. Качество этой сетки $Q(\Omega_h) = 0{,}73$, а среднее качество треугольников составляет 0,88.

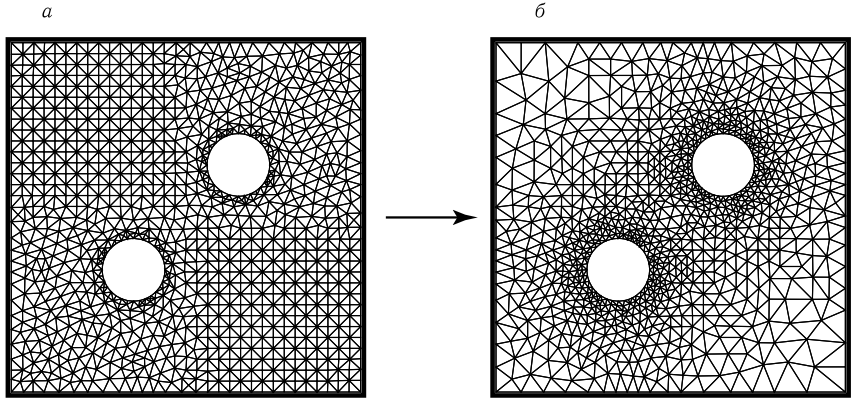


Рис. 5.10. Перестроение квазиравномерной сетки в регулярную сетку, сгущающуюся к отверстиям

Добавление последнего базового алгоритма приводит к сетке, показанной на рис. 5.10, *б*. Детальный анализ алгоритма 35 показывает, что число итераций уменьшилось примерно вдвое: до 22 682, качество сетки выросло до $Q(\Omega_h) = 0{,}78$, а среднее качество треугольников повысилось незначительно: до 0,89.

Таким образом, минимальная возможность изменять топологию сетки позволяет перестраивать ее за разумное число операций. Расширение множества топологических операций приводит к более быстрой сходимости алгоритма. Отметим также, что полная перестройка сетки требует нескольких итераций на один элемент сетки (12 в данном примере).

Оптимальный порядок базовых алгоритмов на шаге 6 алгоритма 35 зависит не только от свойств начальной и конечной сеток, но и от динамики процесса. Например, перестроение анизотропной сетки с треугольниками, вытянутыми в горизонтальном направлении, в анизотропную сетку с треугольниками, вытянутыми в вертикальном направлении, может происходить через промежуточную квазиравномерную сетку. В начале процесса перестроения сетки первый базовый алгоритм будет выполняться значительно чаще других алгоритмов. Затем начнут доминировать четвертый и пятый базовые алгоритмы. Поэтому оптимальный порядок базовых алгоритмов практически невозможно определить заранее.

Вершины квадрата на рис. 5.10 являются особыми точками модели. Узлы сетки в вершинах квадрата обычно заданы раз и навсегда, и базовые алгоритмы должны сохранять их положение. Одно из возможных решений — создать дополнительный список специальных свойств узлов сетки и незначительно модифицировать базовые алгоритмы. Например, алгоритм 32 не должен двигать узел сетки, помеченный как фиксированный.

Определение специальных свойств узлов, в дополнение к уже заданным, может быть автоматизировано, если доступна дополнительная информация о модели. Например, если граница области представлена несколькими кривыми, то разумно зафиксировать узлы сетки, соответствующие начальным и конечным точкам этих кривых. Если граница задана как набор ребер, то общую вершину двух ребер, образующих острый угол, можно пометить как фиксированную. Могут быть введены также специальные свойства для граничных ребер и треугольников. Дополнительная информация о сеточных объектах позволяет эффективно управлять процессом перестроения сетки. Мы вернемся к этому вопросу в § 5.4.

Оценка вычислительной сложности алгоритма 35 складывается из оценок сложности исполнения базовых операций (W_1), поддержки различных структур данных (W_2) и поиска в динамических неструктурированных списках (W_3). Вычислительная сложность одной базовой операции и связанного с ней обновления структур данных не зависит от размера сетки. К сожалению, быстрый поиск в неструктурированных списках зависит от размера сетки. Поэтому на очень мелких сетках следует ожидать, что $W_3 \gg W_1 + W_2$.

Рассмотрим последовательность сеток с растущим числом треугольников. Табл. 5.1 показывает относительную стоимость одной итерации алгоритма 35 при построении квазиравномерных сеток Ω_h, начиная с очень грубой сетки, показаной на рис. 5.9, *а*, а также при перестроении этих сеток в регулярные сетки $\widetilde{\Omega}_h$, сгущающиеся к отверстиям и аналогичные сетке, показаной на рис. 5.10. Напомним, что динамический список качеств треугольников разбивается на блоки длины $O(\log_2 N_f)$, и что увеличение числа таких блоков приводит к росту времени одной операции с этим списком. Данные, представленные в таблице, показывают, что W_3 начинает заметно доминировать, когда число треугольников превышает 10^5.

Таблица 5.1
Относительная средняя стоимость одной итерации алгоритма 35

Квазиравномерные сетки		Регулярные сетки	
$N_f(\Omega_h)$	время	$N_f(\widetilde{\Omega}_h)$	время
930	1,00	986	1,00
11 916	0,88	9934	1,00
116 726	0,80	108 357	1,11
466 742	1,84	438 003	3,53
904 210	3,36	1 066 444	9,05

§ 5.3. Перестроение тетраэдризаций

Минимальный набор структур данных, определяющих тетраэдральную сетку, включает три структурированных списка для тетраэдров, граничных граней и координат вершин. Построение дополнительных структур данных на основе этой минимальной информации обсуждается ниже.

Надежное перестроение тетраэдральных сеток — значительно более сложная задача, чем перестроение треугольных сеток. И причина заключается не только в дополнительной размерности. Множество фундаментальных результатов для триангуляций не распространяется на тетраэдризации. Например, широко известно, что правильные треугольники одинакого размера покрывают плоскость без пересечений и пустот. Аналогичное утверждение для правильных тетраэдров неверно, хотя существуют тетраэдры одинаковой формы, для которых утверждение верно. Не существует также простой локальной модификации тетраэдральной сетки, для которой был бы верен аналог теоремы 5.2.1 о построении триангуляции Делоне. Даже тетраэдризация Делоне может содержать сливеры (т. е. тетраэдры с двугранными углами, близкими к 180°), которые приводят к различным проблемам при численном решении уравнений в частных производных.

Большинство существующих подходов к улучшению свойств тетраэдральной сетки используют комбинацию различных методов, таких, как сглаживание сетки, изменение ее топологии и ее измельчение. Эти методы приводят к разнообразным локальным модификациям сетки, которые мы будем называть *базовыми операциями*.

В § 5.2 мы показали, что расширение множества базовых операций положительно влияет на сходимость метода перестроения сетки. Для тетраэдральных сеток можно выделить семь базовых алгоритмов, которые меняют топологию сетки и форму тетраэдров. Рассмотрим тетраэдр $\Delta(v_1, v_2, v_3, v_4)$ (или Δ_{1234} для простоты обозначений) с вершинами v_1, v_2, v_3 и v_4, качество которого требуется увеличить. Мы представим структуру базовых алгоритмов для тетраэдра, расположенного внутри области. Расширения алгоритмов на приграничный тетраэдр будут обсуждаться в комментариях к алгоритмам.

Все базовые операции изменяют сетку внутри суперэлемента $\sigma(\Delta_{1234})$, образованного тетраэдрами, которые имеют хотя бы одну общую точку с Δ_{1234}:

$$\sigma(\Delta_{1234}) = \bigcup_{i=1}^{4} \sigma(v_i).$$

Сетка на границе $\sigma(\Delta_{1234})$ не меняется. Как было отмечено в § 5.1, это свойство — ключ к разработке параллельных методов перестроения сетки.

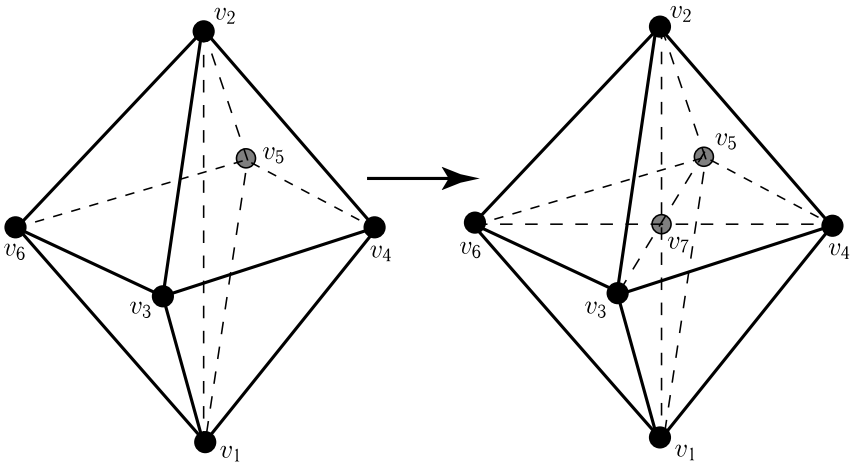


Рис. 5.11. Постановка узла v_7 на ребро e_{12}

Первый базовый алгоритм ставит узел на середину ребра тетраэдра Δ и разбивает прилежащие к ребру тетраэдры пополам. Алгоритм аналогичен алгоритму 30 для треугольника. Поскольку двугранный угол тетраэдра меньше 180°, число тетраэдров с общим ребром больше либо равно трем. Рис. 5.11 иллюстрирует работу этого алгоритма для внутреннего ребра сетки, которое является общим для четырех тетраэдров Δ_{1234}, Δ_{1245}, Δ_{1256} и Δ_{1263}.

Алгоритм 36. Постановка узла на ребро тетраэдра

1: **loop** по всем ребрам e тетраэдра Δ
2: Найти тетраэдры $\Delta_1, \ldots, \Delta_n$ с общим ребром e, как показано на рис. 5.1, и положить $\Delta_1 \equiv \Delta$. Определить $Q_0 = \min\{Q(\Delta_1), \ldots, Q(\Delta_n)\}$
3: Поставить пробный узел v_7 на середину ребра e и разбить каждый тетраэдр Δ_i на два виртуальных тетраэдра Δ_i^a и Δ_i^b. Определить $Q_1 = \min\{Q(\Delta_1^a), Q(\Delta_1^b), \ldots, Q(\Delta_n^a), Q(\Delta_n^b)\}$
4: **if** $Q_1 > Q_0$ **then**
5: Добавить узел v_7 в сетку и заменить каждый из тетраэдров Δ_i двумя тетраэдрами Δ_i^a и Δ_i^b. Закончить алгоритм
6: **end if**
7: **end loop**

Алгоритм легко обобщается на случай тетраэдра с одним или несколькими ребрами, лежащими на границе области. Дополнительный анализ требуется только для ребер, лежащих на криволинейных границах. Мы будем называть такие ребра *криволинейными ребрами*. Отметим, что дополнительная информация о криволинейных границах

может быть доступна, если модель расчетной области задана с использованием САПР. Библиотеки САПР позволяют эффективно и надежно работать с криволинейными границами, и по возможности их следует использовать.

Предположим, что каждая точка криволинейного ребра отображается в единственную точку криволинейной границы, а функция $F(\mathbf{x})$ описывает это отображение. Координаты средней точки ребра e_{12} и точки, спроецированной на криволинейную границу, вычисляются следующим образом:

$$\widehat{\mathbf{v}}_7 = (\mathbf{v}_1 + \mathbf{v}_2)/2, \qquad \mathbf{v}_7 = F(\widehat{\mathbf{v}}_7).$$

Как и в случае треугольных сеток, проекция узла может привести к запутыванию сетки. Для проверки правильности сетки мы воспользуемся аналогом леммы 5.2.1. Рассмотрим суперэлемент $\sigma(e_{12})$, образованный тетраэдрами с общим ребром e_{12}, и выделим те грани, которые лежат на поверхности этого суперэлемента и не содержат вершину $\widehat{\mathbf{v}}_7$. Каждая из этих граней задает плоскость, которая разбивает пространство на два полупространства. Пусть π_i — полупространства, содержащие вершину $\widehat{\mathbf{v}}_7$. Определим множество $\mathcal{D}$ как пересечение полупространств π_i.

Лемма 5.3.1. *Проекция $\mathbf{v}_7 = F(\widehat{\mathbf{v}}_7)$ приводит к правильной сетке, если $\mathbf{v}_7 \in \mathcal{D}$.*

Доказательство этой леммы повторяет рассуждения леммы 5.2.1 и поэтому не приводится. Важное следствие доказательства касается знака алгебраического объема тетраэдра с вершиной $\mathbf{v}_7$. Алгебраический объем тетраэдра Δ_{1347} вычисляется по формуле

$$V_{\Delta_{1347}} = \frac{1}{6} \det\{\mathbf{v}_3 - \mathbf{v}_1, \mathbf{v}_4 - \mathbf{v}_1, \mathbf{v}_7 - \mathbf{v}_1\}, \tag{5.3.1}$$

где det — детерминант матрицы 3×3 со столбцами $\mathbf{v}_3 - \mathbf{v}_1$, $\mathbf{v}_4 - \mathbf{v}_1$ и $\mathbf{v}_7 - \mathbf{v}_1$. Рассмотрим вспомогательные (виртуальные) тетраэдры, построенные с использованием $\widehat{\mathbf{v}}_7$ вместо $\mathbf{v}_7$. Следуя обозначениям, введенным в алгоритме 36, мы обозначим эти тетраэдры $\widehat{\Delta}_i^a$ и $\widehat{\Delta}_i^b$. Верен следующий аналог леммы 5.2.2.

Лемма 5.3.2. *Пусть $\widehat{\Delta}_i^a$ и $\widehat{\Delta}_i^b$ — вспомогательные тетраэдры, построенные с использованием $\widehat{\mathbf{v}}_7$ вместо $\mathbf{v}_7$. Сетка остается корректной, если алгебраические объемы тетраэдров $\widehat{\Delta}_i^a$ и Δ_i^a (аналогично для тетраэдров $\widehat{\Delta}_i^b$ и Δ_i^b) имеют один и тот же знак.*

Приведем другую интерпретацию формулы (5.3.1):

$$V_{\Delta_{1347}} = \frac{1}{6}\,((\mathbf{v}_3 - \mathbf{v}_1) \times (\mathbf{v}_4 - \mathbf{v}_1)) \cdot (\mathbf{v}_7 - \mathbf{v}_1) = \mathbf{a} \cdot (\mathbf{v}_7 - \mathbf{v}_1).$$

Вычисление многих алгебраических объемов происходит для двух положений вершины тетраэдра. Например, для вершины $\mathbf{v}_7$ эффективная реализация этих вычислений основана на выделении вектора $\mathbf{a}$, который не зависит от положения этой вершины и может быть вычислен только один раз.

При вычислениях на ЭВМ определение знака алгебраического объема зависит от ошибок округления. В гл. 3 мы предложили два метода решения этой проблемы: (а) вычисление детерминанта (5.3.1) с дополнительной точностью и (б) введение функции $d(v_1, v_2, v_3, v_4)$, которая возвращает ноль, когда знак детерминанта не может быть определен точно. Мы рекомендуем использовать один из этих методов всякий раз, когда требуется найти знак алгебраического объема. Отметим, что нулевое значение функции $d(v_1, v_2, v_3, v_4)$ приводит к ошибкам первого рода, которые не критичны для перестроения сетки и лишь ограничивают число допустимых базовых операций.

Второй базовый алгоритм применяется к паре тетраэдров, имеющих общую грань. Если объединение этих тетраэдров образует выпуклый многогранник, в котором никакие четыре точки не лежат в одной плоскости, то существует другое разбиение этого многогогранника на три тетраэдра (см. рис. 5.12). Алгоритм 37 определяет, какое из разбиений улучшает качество сетки. В отличие от алгоритма двумерного флипа, число элементов в сетке увеличивается на 1.

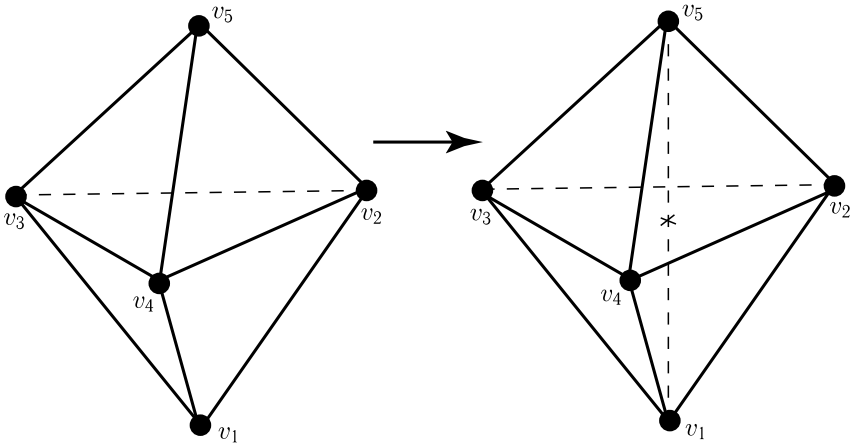


Рис. 5.12. Замена грани f_{234} на ребро e_{15}

Алгоритм 37 легко обобщается на случай тетраэдра Δ с гранями, лежащими на внутренних границах. Поскольку такие грани не могут быть удалены из сетки, алгоритм не должен их рассматривать.

Третий базовый алгоритм применяется к тройке тетраэдров, имеющих общее ребро. Этот алгоритм является обратным ко второму базовому алгоритму, т. е. последовательное применение этих алгоритмов возвращает сетку в исходное положение. Если объединение трех тетраэдров образует выпуклый многогранник, то существует другое разбиение этого многогранника на два тетраэдра (см. рис. 5.12).

Отметим, что этот алгоритм также является аналогом двумерного алгоритма 31, но уменьшает число элементов в сетке на один. Таким

Алгоритм 37. Замена грани на ребро

loop по всем граням f тетраэдра Δ
  Найти соседний тетраэдр Δ' с гранью f и вершинами v_2, v_3, v_4, v_5, как показано на рис. 5.12. Если тетраэдры Δ и Δ' не образуют строго выпуклый многогранник, то перейти к следующей грани
  Определить $Q_0 = \min\{Q(\Delta), Q(\Delta')\}$
  Вычислить качества тетраэдров Δ_{1425}, Δ_{1235} и Δ_{1345}. Определить $Q_1 = \min\{Q(\Delta_{1425}), Q(\Delta_{1235}), Q(\Delta_{1345})\}$
  if $Q_1 > Q_0$ **then**
    Заменить тетраэдры Δ и Δ' тетраэдрами Δ_{1425}, Δ_{1235} и Δ_{1345}. Закончить алгоритм
  end if
end loop

образом, в пространстве существуют две операции *флипа*. Мы не приводим формальную структуру третьего базового алгоритма ввиду ее очевидности. В дальнейшем, ссылаясь на алгоритм 37, мы будем подразумевать обе операции флипа.

Четвертый базовый алгоритм является обобщением третьего базового алгоритма на случай большего числа тетраэдров с общим ребром. Для иллюстрации мы воспользуемся конфигурацией, показанной на рис. 5.13. Ребро e_{12} заменяется триангулированной полигональной гранью P с вершинами v_3, v_4, v_5, v_6 и v_7. Отметим, что вершины этого многоугольника обычно не лежат в одной плоскости, и о его выпуклости можно говорить лишь в смысле выпуклости его проекции на плоскость, наименее отклоняющуюся от его вершин.

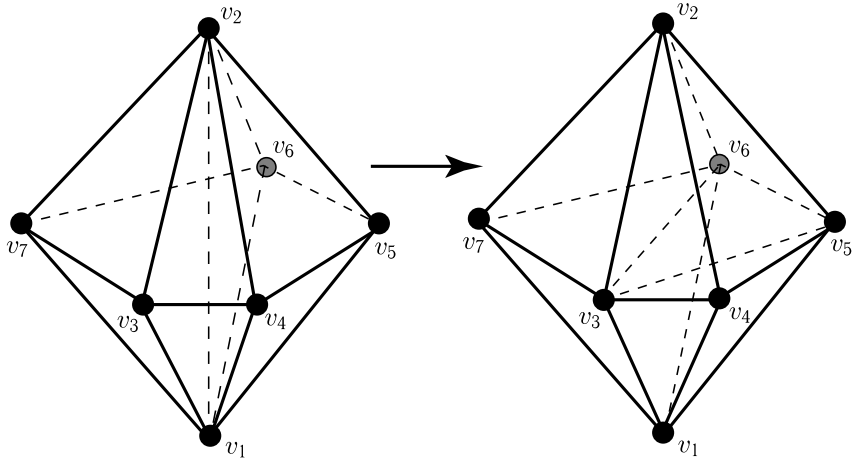


Рис. 5.13. Замена ребра e_{12} на триангулированную полигональную грань f_{34567}

Алгоритм 38. Замена ребра на полигональную грань

loop по всем ребрам e тетраэдра Δ
  Найти тетраэдры $\Delta_1, \ldots, \Delta_{n-2}$ с общим ребром e, как показано на рис. 5.13, где $e \equiv e_{12}$; положить $\Delta_1 \equiv \Delta$. Определить $Q_0 = \min\{Q(\Delta_1), \ldots, Q(\Delta_{n-2})\}$
  loop $i = 3, \ldots, n$
    Разбить полигон P с вершинами $v_3, v_4, \ldots, v_n$ ($n = 7$ на рис. 5.13) на треугольники $f_3, f_4, \ldots, f_{n-2}$, соединяя вершину v_i с остальными вершинами, не связанными с v_i ребром
    Построить тетраэдры Δ_k^a и Δ_k^b с общим основанием f_k и различными вершинами v_1 и v_2, где $k = 3, \ldots, n-2$
    Определить $Q_1 = \min\{Q(\Delta_3^a), Q(\Delta_3^b), \ldots, Q(\Delta_{n-2}^a), Q(\Delta_{n-2}^b)\}$
    if $Q_1 > Q_0$ **then**
      Заменить тетраэдры $\Delta_1, \ldots, \Delta_{n-2}$ тетраэдрами Δ_k^a и Δ_k^b, где $k = 3, \ldots, n-2$. Закончить алгоритм
    end if
  end loop
end loop

Алгоритм 38 рассматривает только $(n-2)$ разбиений многоугольника P на треугольники f_k, хотя число возможных разбиений может достигать $(n-2)!$ для выпуклого многоугольника. Такое большое число возможных разбиений делает непрактичным их полный перебор для $n > 8$. В представленных ниже численных экспериментах алгоритм 38 выполняется всегда, но число рассматриваемых разбиений ограничено тремястами.

Среди специалистов не существует однозначного мнения о необходимости четвертого базового алгоритма для полноты множества топологических операций с сеткой. Мы отметим работы [49, 60, 72], где на основе анализа большого множества сеток было показано, что добавление операций с многоугольными гранями позволяет значительно улучшить качество перестраиваемых сеток. Эксперимент в конце этого параграфа подтверждает результаты этих работ, хотя дополнительное увеличение качества сетки незначительно.

Пятый базовый алгоритм меняет положение узла сетки. Пусть v_1 — одна из вершин тетраэдра Δ. Рассмотрим суперэлемент $\sigma(v_1)$, образованный тетраэдрами с общей вершиной v_1. На рис. 5.14 узел v_1 сдвигается в геометрический центр суперэлемента $\sigma(v_1)$. Данный подход используется в методах улучшения формы тетраэдров. В отличие от своего двумерного аналога, он часто приводит к тетраэдрам плохой формы.

В общем случае оптимальное положение вершины v_1 должно увеличивать качество наихудшего тетраэдра в суперэлементе $\sigma(v_1)$. Качество

$Q(\sigma(v_1))$ — нелинейный функционал координат вершины v_1; поэтому, как и в случае треугольных сеток, поиск оптимального положения этой вершины требует применения методов минимизации (или максимизации) невыпуклых функционалов.

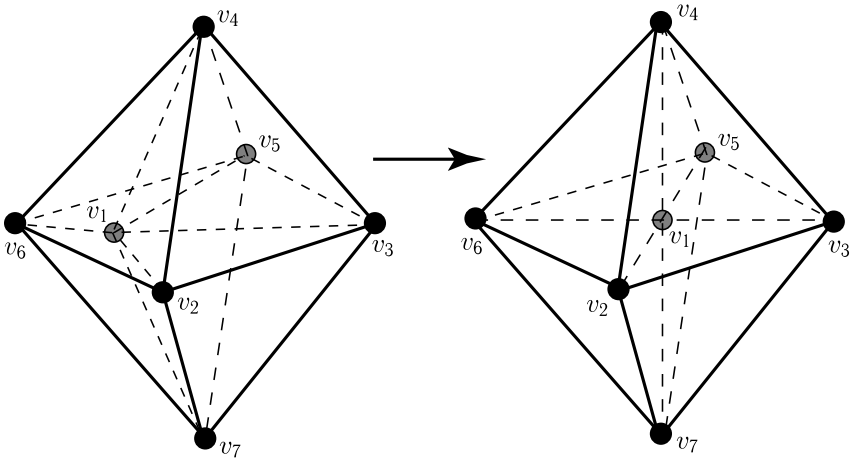


Рис. 5.14. Локальная модификация сетки путем перемещения узла v_1

Пусть $\mathbf{v}_1 = (x_1, y_1, z_1)$ и, для простоты, $Q_\sigma = Q(\sigma(v_1))$. Приближенный градиент Q_σ вычисляется с использованием конечно-разностной дискретизации:

$$\nabla^h Q_\sigma = \Big(\frac{Q_\sigma(x_1 + \Delta x, y_1, z_1)}{\Delta x}, \frac{Q_\sigma(x_1, y_1 + \Delta y, z_1)}{\Delta y}, \frac{Q_\sigma(x_1, y_1, z_1 + \Delta z)}{\Delta z} \Big)^{\mathrm{T}}.$$

Разумным значением приращений Δx, Δy и Δz является корень квадратный из машинной точности ϵ. Для надежности алгоритма необходимо проверить, что вычисленное приращение значительно меньше диаметра суперэлемента, например, меньше 1 % от длины минимального ребра:

$$\Delta x = \Delta y = \Delta z = \min\{\sqrt{\epsilon}, \min_{v_i, v_j \in \sigma(v_1)} |v_i - v_j|/100\}.$$

Узел v_1 сдвигается вдоль приближенного градиента до тех пор, пора максимум Q_σ не будет достигнут:

$$\mathbf{v}_1 := \mathbf{v}_1 + \alpha \nabla^h Q_\sigma, \qquad \alpha \geqslant 0.$$

Отметим, что простая замена Q_σ на $1 - Q_\sigma$ позволяет применить стандартные методы минимизации фукционалов без их модификации.

Поскольку Q_σ не является квадратичным функционалом, метод градиентного спуска (в нашем случае — метод градиентного подъема) обычно не приводит к вычислению, даже приближенному, локального максимума функционала на суперэлементе $\sigma(v_1)$. Мы не рекомендуем

циклическое повторение описанной выше процедуры, как это делается в стандартных методах минимизации (или максимизации) функционалов. Перестроение сетки может многократно изменять любой из тетраэдров, поэтому существует большая вероятность того, что конфигурация суперэлемента $\sigma(v_1)$ быстро изменится.

Для нахождения параметра α используется алгоритм 39, который может использовать метод бисекции или различные варианты метода Левенберга–Маркуарда.

Алгоритм 39. Сдвиг узла

1: **loop** по всем вершинам v_1 тетраэдра Δ
2: Найти тетраэдры $\Delta_1, \Delta_2, \ldots, \Delta_n$, которые образуют суперэлемент $\sigma(v_1)$; положить $\Delta = \Delta_1$, как показано на рис. 5.14, где $n = 8$
3: Вычислить приближенный градиент $\nabla^h Q_\sigma$
4: Вычислить максимально возможное смещение вершины v_1 в направлении $\nabla^h Q_\sigma$, не нарушающее топологию суперэлемента. Пусть $\alpha = \alpha_{\max}$ для этого экстремального положения
5: Найти значение α в полуоткрытом интервале $[0, \alpha_{\max})$, которое максимизирует Q_σ
6: Если $\alpha > 0$, то запомнить новое положение вершины v_1 в сетке и закончить алгоритм
7: **end loop**

Отметим, что узел v_1 не может выйти за пределы суперэлемента, поскольку качество Q_σ стремится к нулю, когда v_1 приближается к его границе. Тем не менее перемещение узла даже внутри суперэлемента может привести к запутыванию сетки, когда этот суперэлемент не выпукл. Анализ достаточных условий для правильности сетки аналогичен анализу, разработанному для треугольных сеток. Для внутреннего узла сетки v_1 допустимое множество $\mathcal{D}$ его положений определяется как пересечение полупространств $\pi_1, \ldots, \pi_n$. Полупространство задается одной из граничных граней суперэлемента $\sigma(v_1)$ и содержит исходный узел v_1. Отметим, что множество $\mathcal{D}$ совпадает с суперэлементом $\sigma(v_1)$, только когда он выпукл.

Другой метод проверки правильности сетки состоит в вычислении алгебраических объемов тетраэдров $\Delta_1, \ldots, \Delta_n$ для каждого нового положения их вершины v_1. Сетка остается правильной, если эти объемы сохраняют знак при смещении вершины v_1.

Алгоритм 39 легко обобщается на случай тетраэдра с вершинами, лежащими на внутренних и на внешних границах. Такие вершины могут двигаться только по граничным граням. Движение по криволинейным граням должно сопровождаться проверкой правильности сетки. Как и раньше, для этого достаточно контролировать сохранение знака алгебраических объемов тетраэдров $\Delta_1, \ldots, \Delta_n$. Аналогично,

движение узла, лежащего на ребре, разделяющем граничные грани, или ребре модели САПР, должно происходить только вдоль этого ребра.

Шестой базовый алгоритм удаляет ребро из сетки. Алгоритм можно интерпретировать следующим образом. Начинаем смещать вершины ребра навстречу друг другу, возможно, с разными скоростями. Когда вершины сливаются в одну, топология сетки меняется. Как показано на рис. 5.15, при слиянии вершин v_1 и v_8 каждая из четырех пар ребер сливается в одно ребро, а четыре тетраэдра с общим ребром e_{18} исчезают из сетки. В алгоритме 40 мы используем обозначения, показанные на рис. 5.15.

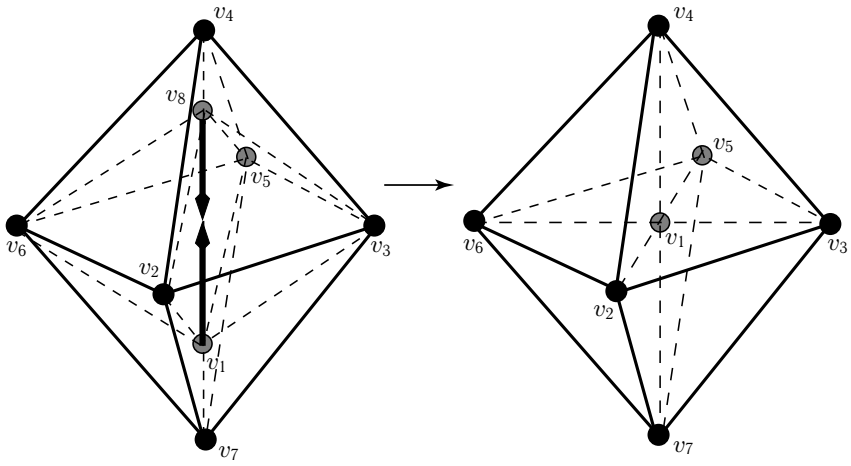


Рис. 5.15. Локальная модификация сетки путем слияния вершин v_1 и v_8

Алгоритм 40 легко обобщается на случай граничного ребра, лежащего внутри плоского участка границы модели. Ребро с одной из вершин, лежащей на ребре модели, удаляется из сетки путем смещения противоположной вершины в данную вершину. Удаление же ребра сетки, лежащего полностью на ребре модели, т. е. на пересечении двух плоских участков границы, требует проверки того, что это не единственное ребро, представляющее важную часть геометрической модели.

Шаг 4 алгоритма 40 требует выбрать общую вершину $\widehat{v}_1$ для вершин v_1 и v_8. С одной стороны, естественно рассмотреть задачу об оптимальном положении вершины $\widehat{v}_1$. С другой стороны, поскольку аналогичная оптимизационная задача рассматривается в базовом алгоритме сдвига вершины, усложнение алгоритма 40 не требуется. Оптимальное положение вершины $\widehat{v}_1$ может быть достигнуто путем последовательного применения двух базовых алгоритмов. Поэтому мы

Алгоритм 40. Удаление ребра

1: **loop** по всем ребрам e тетраэдра Δ
2: Найти тетраэдры $\Delta_1, \Delta_2, \ldots, \Delta_n$ которые образуют суперэлемент $\sigma(e)$, и положить $\Delta = \Delta_1$, как показано на рис. 5.15, где $n = 4$
3: Найти тетраэдры $\Delta'_1, \Delta'_2, \ldots, \Delta'_m$, которые: (а) отличны от тетраэдров в суперэлементе $\sigma(e)$, (б) входят в суперэлемент $\sigma(v_1)$ или $\sigma(v_8)$
4: Определить новое общее (виртуальное) положение $\widehat{v}_1$ вершин v_1 и v_8: $\widehat{\mathbf{v}}_1 = \frac{1}{2}(\mathbf{v}_1 + \mathbf{v}_8)$
5: Определить новые (виртуальные) тетраэдры $\widehat{\Delta}'_1, \widehat{\Delta}'_2, \ldots, \widehat{\Delta}'_m$, сдвигая вершины v_1 и v_8 исходных тетраэдров в $\widehat{v}_1$
6: Если алгебраические объемы тетраэдров Δ'_i и $\widehat{\Delta}'_i$ имеют разные знаки хотя бы для одного i, то удалить все виртуальные объекты и перейти к следующему ребру
7: Определить качества $Q_0 = \min\{Q(\Delta_1), Q(\Delta_2), \ldots, Q(\Delta_n)\}$ и $Q_1 = \min\{Q(\widehat{\Delta}'_1), Q(\widehat{\Delta}'_2), \ldots, Q(\widehat{\Delta}'_m)\}$
8: **if** $Q_1 > Q_0$ **then**
9: Удалить вершину v_8 и тетраэдры $\Delta_1, \ldots, \Delta_n$ из сетки; заменить вершину v_1 на $\widehat{v}_1$ и тетраэдры Δ'_i на $\widehat{\Delta}'_i$. Закончить алгоритм
10: **end if**
11: **end loop**

предлагаем рассматривать только одно возможное положение новой вершины: $\widehat{\mathbf{v}}_1 = \frac{1}{2}(\mathbf{v}_1 + \mathbf{v}_8)$. Исключение составляют случаи, когда одна из вершин ребра e_{18} лежит на границе области. В этом случае вершина, лежащая внутри области, сдвигается в граничную вершину.

Обобщение алгоритма 40 для криволинейных границ требует проецирования вершины $\widehat{\mathbf{v}}_1$, полученной на шаге 4 алгоритма, на криволинейную границу. Как и любое другое перемещение вершины, проекция должна попадать в допустимое множество сдвигов, при которых сетка не запутывается. Достаточным условием сохранения правильности сетки является проверка знаков алгебраических объемов на шаге 6 алгоритма. Оценка локальной кривизны границы также может быть использована для определения возможности удаления криволинейного ребра.

Седьмой базовый алгоритм удаляет из сетки узел вместе со всеми содержащими его тетраэдрами. Пустота, созданная в сетке, заполняется меньшим числом тетраэдров. Для построения этих тетраэдров в алгоритме 41 используются только существующие узлы сетки.

Отметим, что проверка знаков алгебраических объемов достаточна для узла v, лежащего внутри области Ω_h и вне внутренних границ

Алгоритм 41. Удаление узла

loop по всем вершинам v тетраэдра Δ
  Найти тетраэдры $\Delta_1, \Delta_2, \ldots, \Delta_n$, которые образуют суперэлемент $\sigma(v)$, и положить $\Delta = \Delta_1$. Подсчитать число вершин M_v на границе суперэлемента
  loop $i = 1, \ldots, M_v$
    Разбить суперэлемент $\sigma(v)$ на виртуальные тетраэдры $\Delta'_1, \ldots, \Delta'_m$, соединяя граничную вершину v_i с остальными граничными вершинами, не соединенными с v_i ребрами
    Если знаки алгебраических объемов тетраэдров Δ_k и Δ'_l с общей граничной гранью суперэлемента $\sigma(v)$ различны хотя бы для одного k, то перейти к следующей граничной вершине
    Если грани виртуальных тетраэдров не аппроксимируют внутренние и граничные поверхности, перейти к следующей вершине
    Определить локальные качества $Q_0 = \min\{Q(\Delta_1), \ldots, Q(\Delta_n)\}$ и $Q_1 = \min\{Q(\Delta'_1), \ldots, Q(\Delta'_m)\}$
    if $Q_1 > Q_0$ **then**
      Удалить узел v из сетки, заменить тетраэдры $\Delta_1, \ldots, \Delta_n$ тетраэдрами $\Delta'_1, \ldots, \Delta'_m$. Закончить алгоритм
    end if
  end loop
end loop

(например, границ материалов). В этом случае суперэлемент $\sigma(v)$ является многогранником.

В противном случае требуется несколько дополнительных проверок аппроксимации внешних и внутренних границ гранями виртуальных тетраэдров. Для плоских границ достаточно проверить сохранение всех этих границ с машинной точностью. Для криволинейных границ необходимо удостовериться, что топология новых дискретных границ соответствует топологии исходных границ. Например, предположим, что грани f_{124}, f_{145}, f_{157} и f_{172} на рис. 5.16, *а*, аппроксимируют часть криволинейной границы геометрической модели. Топология этой границы сохраняется для одной из двух пар новых граней: f_{245}, f_{257} или f_{247}, f_{457}. Как и в двух измерениях, анализ топологии границы становится проще, если рассматривать базовую операцию удаления узла как частный случай более общей операции удаления ребра.

Описанные выше семь базовых алгоритмов наиболее часто используются в методах улучшения качества сетки [86]. Мы также отметим несколько дополнительных операций с сеткой, которые встречаются в литературе. Для улучшения формы тетраэдров авторы работы [60] предлагают добавлять в сетку узлы Штейнера, используя достаточно

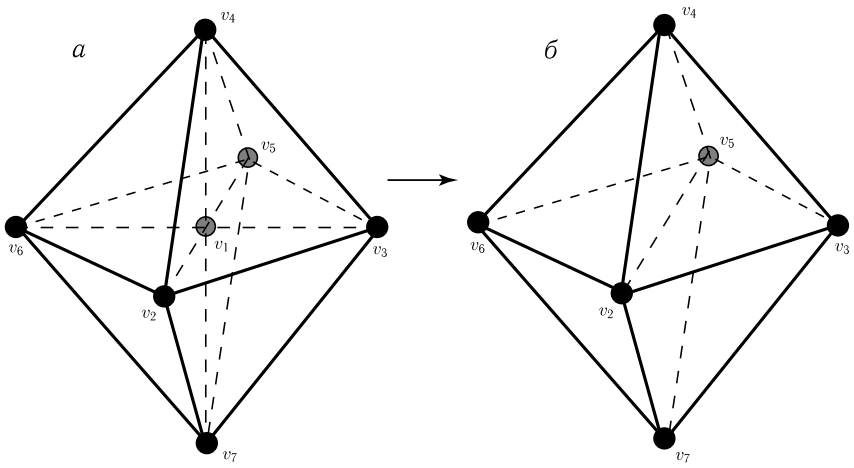


Рис. 5.16. Локальная модификация сетки путем удаления узла v_1

сложный алгоритм, обобщение которого на произвольное качество $Q(\Delta)$ не очевидно. Операция, обратная к базовому алгоритму 40, рассматривается в [49]. Можно показать, что она сводится к комбинации алгоритмов флипа, если ни один из них не уменьшает качество сетки. Операция перестроения триангуляции полигональной грани f_{34567} (см. рис. 5.13) рассматривается в [72] и является комбинацией базового алгоритма 38 и обратного к нему, если ни одна из этих операций не уменьшает качество сетки.

Комбинация базовых алгоритмов открывает широкие возможности для построения или перестроения расчетных сеток. Рассмотрим, какие структуры данных требуются для эффективной реализации базовых алгоритмов. Отметим, что эти структуры будут аналогичны структурам данных, использующимся для перестроения треугольных сеток.

Во-первых, как и в случае треугольных сеток, необходимо поддерживать список тетраэдров, упорядоченных по возрастанию их качества. Как и в двумерном случае (см. § 5.2), мы будем использовать блочно структурированный список и алгоритмы его модификации, описанные в § 2.3.

Во-вторых, для каждой вершины v тетраэдра Δ необходимо быстро находить тетраэдры в суперэлементе $\sigma(\Delta)$. Для этого применяются несколько оптимальных методов с арифметической сложностью $O(1)$. Самый простой метод состоит в построении структурированного списка ближайших соседей для каждого тетраэдра сетки $U(\Omega_h)$ и его модификации после каждого базового алгоритма. Число таких соседей не больше четырех. Начальное построение такого упорядоченного списка требует $O(N_t)$ арифметических действий, где N_t — число тетраэдров в сетке, и подробно обсуждается в § 4.3. Каждая базовая операция изменяет всего несколько тетраэдров, поэтому обновление этого списка

имеет оптимальную сложность. Построение суперэлемента $\sigma(\Delta)$ основано на списке $U(\Omega_h)$ и начинается с Δ. После этого мы добавляем в список его ближайших соседей, а затем соседей соседей, которые имеют общую вершину с Δ. Данный процесс повторяется до тех пор, пока такие соседи существуют.

В-третьих, базовые алгоритмы меняют число основных сеточных объектов: узлов, тетраэдров и граничных граней. Эти объекты представлены структурированными списками (см. § 2.3). Добавление нового объекта в структурированный список равносильно добавлению новой строки в двумерный массив. Удаление объекта предполагает сдвиг на одну строку всех объектов, расположенных ниже удаленного объекта, что является дорогостоящей операцией. Вместо этого мы будем хранить дополнительные списки удаленных узлов, граничных граней и тетраэдров, т. е. номера соответствующих строк в структурированных массивах. При добавлении нового объекта мы сначала проверяем наличие свободных мест в структурированном списке и заполняем их. После перестроения сетки свободные места в структурированном списке заполняются, например, путем перемещения сеточных объектов из конца списка на свободные места.

Перестроение сетки осуществляется алгоритмом 35, описанным в § 5.2. Поскольку адаптация этого алгоритма к тетраэдральным сеткам состоит в замене треугольников на тетраэдры и ребер на грани, мы не будем повторять структуру этого алгоритма. В дальнейшем мы будем ссылаться на алгоритм 35 как на алгоритм перестроения тетраэдральных сеток.

Алгоритм 35 применим для построения расчетных сеток на основе очень грубой начальной сетки. Например, сетка, показанная на рис. 5.17, *а*, содержит 24 узла и 36 тетраэдров. Отметим, что начальная сетка содержит лишь на четыре узла больше, чем требуется для минимального представления модели параллелепипеда с отверстием в форме правильной шестиугольной призмы.

Параметр $Q_0 = 0{,}81$ был выбран таким образом, чтобы качество канонического тетраэдра было меньше Q_0. В ряде конечноэлементных методов тетраэдры с двугранными углами, не превышающими 90°, позволяют построить численные схемы с дополнительными свойствами, такими, как дискретный принцип максимума.

Отметим, что алгоритм 35 не гарантирует построение сеток с высоким качеством $Q(\Omega_h)$, поскольку $Q(\Omega_h)$ всегда равно качеству наихудшего тетраэдра. Геометрия модели, в частности, наличие острых двугранных углов, налагает ограничение на максимальное качество приграничных тетраэдров и, следовательно, на качество сетки, которое может быть достигнуто. Тем не менее среднее качество тетраэдров может быть достаточно высоким. Отметим также, что приведенный набор базовых алгоритмов не гарантирует, что любая исходная сетка может быть перестроена в любую другую сетку более высокого качества

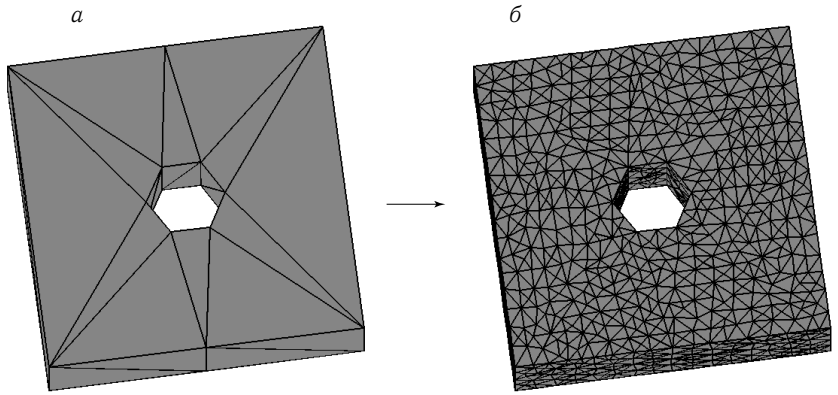


Рис. 5.17. Построение квазиравномерной сетки на основе грубой начальной сетки

путем локальных изменений топологии сетки, постоянно увеличивающих ее качество. Несмотря на недостаток теоретических результатов, на практике алгоритм 35 позволяет успешно строить сетки, близкие к оптимальным, для эффективного приближенного решения дифференциальных уравнений в частных производных.

Возвращаясь к сетке, показанной на рис. 5.17, *б*, отметим, что ее качество $Q(\Omega_h) = 0{,}45$ при среднем качестве тетраэдров 0,67. Хотя $Q(\Omega_h)$ значительно меньше Q_0, тем не менее максимальный двугранный угол равен 138°, а минимальный 22°, что является приемлемым результатом для генераторов тетраэдральных сеток. Вторая строка в табл. 5.2 показывает распределение тетраэдров по качеству в этой казиравномерной сетке. Отметим, что только 28 тетраэдров имеют качество в диапазоне (0,4, 0,5].

Таблица 5.2

Распределение тетраэдров по качеству

Качество / Сетка	0,4	0,5	0,6	0,7	0,8	0,9	1,0
квазиравномерная	0	28	7002	8285	5998	2761	450
регулярная-1	0	0	3416	9455	6854	3244	506
регулярная-2	0	5	8389	17 316	13 027	6018	1086

Теперь мы перестроим квазиравномерную сетку в регулярную сетку с примерно таким же числом тетраэдров, которая сгущается к отверстию. Качество тетраэдра определяется таким образом, что его размер увеличивается прямо пропорционально квадрату расстояния до центральной оси отверстия. След сетки на поверхности области показан на рис. 5.18, *а*. Качество перестроенной сетки $Q(\Omega_h) = 0{,}53$ при среднем

качестве тетраэдров 0,70. Максимальный двугранный угол равен 135°, а минимальный 24°. Таким образом, перестроение сетки позволило не только изменить пространственное распределение тетраэдров, но и сохранить важные свойства сетки. Это потверждает и табл. 5.2, в третьей строке которой представлено распределение тетраэдров по качеству.

Чтобы показать надежность алгоритма перестроения сетки, мы перестроим квазиравномерную сетку в сетку, которая не только сгущается к отверстию, но и содержит в два раза больше тетраэдров. След сетки на поверхности области показан на рис. 5.18, *б*. Качество перестроенной сетки $Q(\Omega_h) = 0{,}49$ при среднем качестве тетраэдров 0,70. Максимальный двугранный угол равен 136°, а минимальный 23°. Последняя строка в табл. 5.2 показывает примерно двукратное увеличение числа тетраэдров в каждом диапазоне качества. Таким образом, обе регулярные сетки имеют примерно одинаковые свойства.

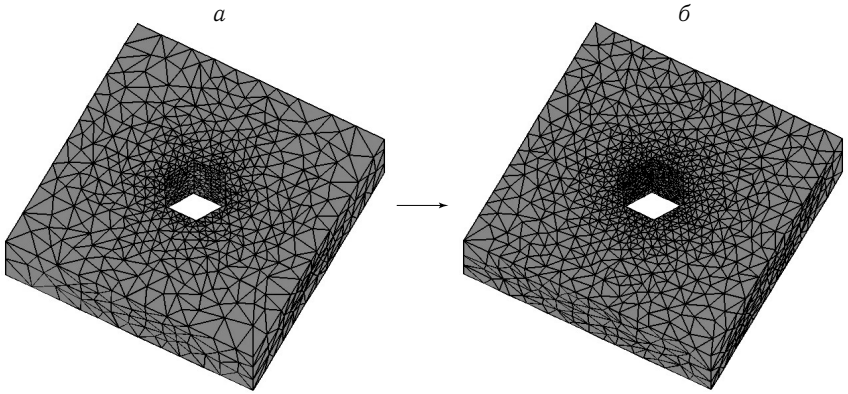


Рис. 5.18. Построение регулярных сеток на основе квазиравномерной сетки

Отметим, что гладкость поверхностной сетки меньше, чем у сеток, построенных методом продвинутого фронта. Гладкость сетки может приводить к суперсходимости конечноэлементных решений, если дифференциальное решение достаточно гладкое. В ряде задач упругости, особенно в задачах с трещинами, основная ошибка сосредоточена вблизи трещин, где решение недостаточно гладкое, поэтому эффект суперсходимости отсутствует. В таких приложениях с негладким решением визуальная гладкость сетки не приводит к значительному улучшению точности конечноэлементных решений.

В заключение параграфа мы приведем один полезный практический совет для перестроения сеток в областях с криволинейными границами. Если аналитическое представление этих границ неизвестно, то мы рекомендуем зафиксировать раз и навсегда узлы сетки на этих границах. При этом перестроенная сетка будет аппроксимировать криволинейные границы с тем же порядком, что и исходная сетка.

Более того, компьютерная реализация базовых алгоритмов значительно упрощается. Более строгий подход состоит в локальной реконструкции криволинейной границы и рассматривается в приложении.

§ 5.4. Параллелизация трехмерного алгоритма

Параллелизация алгоритма 35 основана на его локальности. Сетку можно перестраивать одновременно в нескольких местах при условии, что любая пара перестраиваемых суперэлементов $\sigma(\Delta)$ и $\sigma(\Delta')$ не имеет общих тетраэдров. Как правило, перестроение треугольных сеток не приводит к чрезмерной вычислительной работе и может быть выполнено на однопроцессорном компьютере. Поэтому мы сосредоточимся на перестроении тетраэдральных сеток.

Простейшая модель параллельного алгоритма состоит из главного процессора (мастер), который распределяет работу среди других процессоров (подчиненные), посылая им суперэлементы $\sigma(\Delta)$ и включая результаты их работы в сетку. Ввиду большого количества структур данных, которые связаны с суперэлементом $\sigma(\Delta)$ и необходимы для его перестроения, время на посылку и получение пакетов может превосходить время перестройки одного суперэлемента. Поэтому такая модель рассчитана на малое число процессоров.

Пусть P — число процессоров в параллельном компьютере. В основе предлагаемой модели параллельного алгоритма лежит разбиение сетки на связные множества $\Omega_h^{(i)}$, $i = 1, \ldots, P$, с примерно равным числом тетраэдров. Важное требование к множеству $\Omega_h^{(i)}$ состоит в минимизации его границы, а точнее, числа тетраэдров, у которых одна из вершин лежит на границе $\Omega_h^{(i)}$. Исключение составляют те границы множества, которые являются также границами исходной сетки. Применение базовых операций к тетраэдру Δ, лежащему в приграничном слое множества $\Omega_h^{(i)}$, требует обмена данными с соседним множествами. Поэтому уменьшение количества таких тетраэдров, называемых *интерфейсными*, позволит построить более эффективный параллельный алгоритм.

Известно несколько методов разбиения сетки Ω_h на P приблизительно равных частей. Наиболее популярные методы включают различные варианты метода бисекции. Спектральный метод бисекции [48, 75] основан на спектральных свойствах графа сетки, таких, как собственный вектор, соответствующий второму собственному значению сеточного оператора Лапласа. Необходимость многократного решения задачи на собственные значения в процессе перестроения сетки делает этот метод слишком дорогостоящим для нашего параллельного алгоритма. Метод инерциальной бисекции [85] значительно дешевле по вычислительной работе, но порождает большее число интерфейсных тетраэдров. Тем не менее, ввиду возможности сильного увеличения или уменьшения числа тетраэдров на любом процессоре в ходе

генерации сетки, оптимизация числа интерфейсных тетраэдров может быть принесена в жертву минимизации вычислительной работы.

Метод инерциальной бисекции использует только координаты узлов сетки для вычисления тензора инерции с последующим разбиением сетки вдоль его главных осей. Вычислительная сложность этого метода пропорциональна числу тетраэдров $N_t(\Omega_h)$.

Рассмотрим тело, состоящее из единичных точечных масс, расположенных в геометрических центрах тетраэдров сетки Ω_h. Тензор инерции этого тела определяется так:

$$\mathbb{T}(\Omega_h) = \sum_{i=1}^{N_t(\Omega_h)} \left(\|\mathbf{p}_i\|^2 \mathbb{I}_3 - \mathbf{p}_i \, \mathbf{p}_i^t \right), \qquad \mathbf{p}_i = \mathbf{x}_{\Delta_i} - \overline{\mathbf{x}}_{\Omega_h}, \tag{5.4.1}$$

где $\mathbf{x}_{\Delta_i} \in \mathbb{R}^3$ обозначает геометрический центр тетраэдра Δ_i, а $\overline{\mathbf{x}}_{\Omega_h}$ — центр сетки:

$$\overline{\mathbf{x}}_{\Omega_h} = \frac{1}{N_t(\Omega_h)} \sum_{i=1}^{N_t(\Omega_h)} \mathbf{x}_{\Delta_i}.$$

Каждое из слагаемых в сумме (5.4.1) является положительно полуопределенной матрицей 3×3. Поэтому тензор $\mathbb{T}(\Omega_h)$ положительно определен для системы из четырех или более точек, не лежащих в одной плоскости. В силу малости порядка $\mathbb{T}(\Omega_h)$ вычисление собственных векторов тензора инерции является дешевой операцией.

Используя тензор инерции, мы можем разбить сетку на подсетки с примерно равным числом элементов плоскостями, ортогональными одному из собственных векторов этого тензора (см. рис. 5.19). Чтобы избежать создания слишком тонких подобластей, мы рекомендуем применять небольшое число разбиений.

Рассмотрим упрощенную реализацию метода инерциальной бисекции. Эта реализация эффективна, когда число процессоров небольшое, но каждый процессор имеет достаточно оперативной памяти, чтобы содержать минимальную информацию обо всей сетке. Простые вычисления показывают, что для хранения тетраэдральной сетки, содержащей 10^6 элементов, необходимо около 34 МБ оперативной памяти, что в сотни раз меньше объемов памяти в современных компьютерах. Небольшое число процессоров позволяет разбивать сетку только вдоль одного направления.

После разбиения сетки на P примерно равных подсеток алгоритм перестроения сетки применяется независимо к каждой подсетке. Для сохранения конформности глобальной сетки налагаются определенные ограничения на операции с тетраэдрами на границах между подсетками: допустимые операции не должны изменять сетку на этих границах.

Арифметическая сложность декомпозиции сетки (шаг 3 в алгоритме 42) и сборки конформной сетки (шаг 5) пропорциональна $N_t(\Omega_h)$. Численные эксперименты подтверждают, что эти шаги не занимают

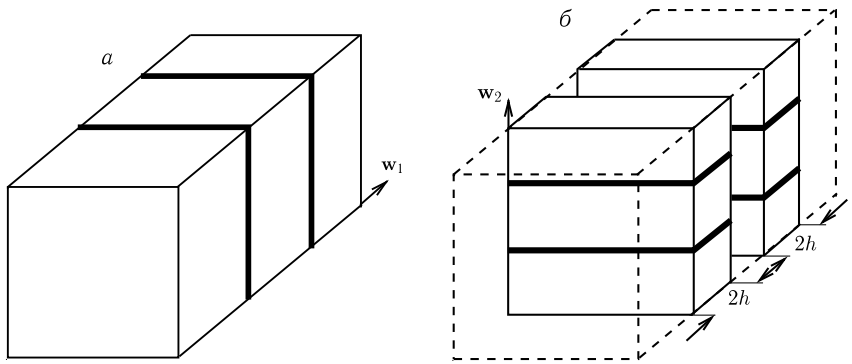


Рис. 5.19. Разбиение вычислительной области на подобласти плоскостями, ортогональными первому (*а*) и второму (*б*) собственным векторам тензора инерции

Алгоритм 42. Параллельное перестроение тетраэдральной сетки

1: Вычислить тензор инерции $\mathbb{T}(\Omega_h)$ на процессоре с рангом 0
2: **loop** $k = 1, 2, 3$:
3: Процессор с рангом 0 выделяет тетраэдры, чье качество меньше Q_0, а также всех их соседей. Множество выделенных тетраэдров разбивается на P примерно равных частей методом инерциальной бисекции для k-й главной оси тензора $\mathbb{T}(\Omega_h)$ и распределяется по всем процессорам
4: Процессор с рангом m, $m = 0, \ldots, P-1$, перестраивает свою подсетку таким образом, что границы с соседними подсетками не изменяются
5: Процессор с рангом 0 собирает подсетки с других процессоров и строит новую конформную глобальную сетку Ω_h. Если $Q(\Omega_h) \geqslant Q_0$, то алгоритм прекращает работу
6: **end loop**
7: Процессор с рангом 0 выделяет тетраэдры, чьи вершины принадлежали одной из границ между подсетками на каждом из трех предыдущих шагов, перестраивает их и переходит к шагу 1

много времени. Таким образом, перестроение подсетки остается самой трудоемкой операцией.

Для реализации алгоритма 42 требуется несколько дополнительных структур данных. Сборка глобальной сетки из подсеток значительно упрощается, если узлы сетки на границах между подсетками сохраняют информацию об исходной глобальной нумерации. Сравнение глобальных индексов приводит к быстрому и надежному нахождению пар одинаковых сеточных узлов.

Отметим, что шаг 4 алгоритма 42 может приводить к низкому качеству приграничных тетраэдров и, как следствие, к низкому качеству

подсеток и глобальной сетки. Поэтому перемена направлений разрезания сетки на подсетки очень важна для сходимости алгоритма. На рис. 5.19, *б* показан случай, когда все приграничные тетраэдры в квазиравномерной сетке с шагом h имеют низкое качество. Эти тетраэдры и их соседи, схематически представленные параллелепипедами толщины $2h$, разбиваются на подобласти плоскостями, ортогональными второму собственному вектору тензора инерции.

Собственные векторы тензора инерции представляют собой три ортогональных направления в пространстве. Однако даже после смены трех направлений глобальная сетка может содержать тетраэдры, чьи вершины оставались все время на границах между подсетками. В результате только часть базовых алгоритмов была применена к этим тетраэдрам. Чтобы улучшить их качество, необходим последний шаг в алгоритме 42. Число таких тетраэдров пропорционально числу пересечений различных троек параллельных плоскостей, равному $(P-1)^3$. Поскольку число процессоров мало́, последний шаг алгоритма реализуется на одном процессоре.

Другое интересное замечание состоит в том, что динамика модификаций сетки может быть совершенно различна в параллельном и последовательном алгоритмах перестроения сетки, поскольку параллельный алгоритм изменяет глобальную сетку в P различных местах одновременно [66, 67]. Рассмотрим модельный пример, показывающий, что это может сильно влиять на работу алгоритма. Численные эксперименты [66] проводились на параллельном компьютере COMPAQ Tru64 Cluster с процессорами ev6 с частотой 667 МГц.

Рассмотрим уравнение Пуассона в области Ω с одним входящим углом, $\Omega = (0,\,1)^3 \setminus [0,\,0{,}5]^3$, и сингулярной правой частью:

$$\begin{aligned} -\Delta u &= |x - x_0|^{-1} \quad \text{в} \quad \Omega, \\ u &= 0 \quad \text{на} \quad \partial\Omega, \end{aligned} \tag{5.4.2}$$

где $x_0 = (0{,}5,\,0{,}5,\,0{,}5)$. Свойства решения задачи (5.4.2) исследованы в [27]. Решение обладает слабыми анизотропными реберными особенностями и сильной изотропной особенностью в вершине входящего угла. Для решения уравнения мы применим метод конечных элементов с кусочно-линейными базисными функциями. Чтобы оценить ошибку дискретизации, мы заменим неизвестное точное решение u дискретным решением $\overline{u}$, вычисленным на очень мелкой адаптивной сетке, содержащей примерно $1{,}5 \cdot 10^6$ тетраэдров. Для построения адаптивной сетки задачу (5.4.2) требуется решить несколько раз на последовательности сеток с приблизительно равным числом ячеек. Мы используем 20 итераций представленного в приложении адаптивного алгоритма 45, на каждой итерации которого используется алгоритм 42.

На рис. 5.20 представлены результаты перестроения сетки для различных значений P. Графики на рис. 5.20, *а* показывают максимальную норму ошибки между $\overline{u}$ и конечноэлементным решением u^h

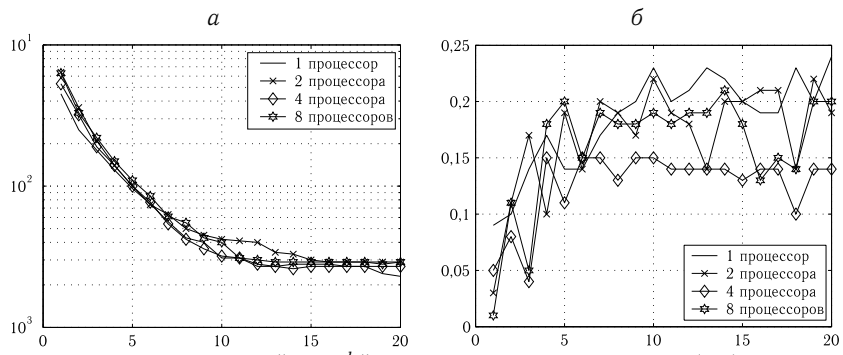


Рис. 5.20. Ошибка $\|\overline{u} - u^h\|_{L^\infty(\Omega)}$ (*а*) и качество сетки $Q(\Omega_h)$ (*б*)

как функцию номера L адаптивной итерации. Эта ошибка примерно одна и та же для всех параллельных расчетов. Графики на рис. 5.20, *б* показывают, что качество сетки после пяти адаптивных итераций больше 0,1 для всех значений P, что указывает на надежность алгоритма 42.

В следующих двух таблицах показана арифметическая сложность параллельного перестроения сетки. Арифметическая сложность измеряется числом реализованных базовых операций $\#mod$. В табл. 5.3 показаны время и число выполненных базовых операций для последней, двадцатой, итерации. Число локальных базовых операций пропорционально $N(\Omega_h)$, а время исполнения каждой операции примерно пропорционально $N(\Omega_h)^{1/2}$. По-видимому, эта зависимость обусловлена неоптимальной реализацией алгоритмов работы с упорядоченным списком качеств элементов.

Таблица 5.3

Время перестроения сетки, $P = 1$, $L = 20$

$N_t(\Omega_h)$	9735	19 359	28 151	36 134	52 079	100 075	160 944
$\#mod$	731	1155	3747	3097	6075	11 147	18 904
время ЦПУ	1,2	1,9	4,7	4,6	11,2	25,3	58,6
$\frac{\text{время ЦПУ}}{\#mod} \cdot 10^3$	1,6	1,6	1,3	1,5	1,8	2,3	3,1

В табл. 5.4 показаны число реализованных базовых операций в параллельных расчетах и полное время перестроения сетки. Уместно отметить, что только после 10-й адаптивной итерации, когда сетка практически установилась, время перестроения сетки стало пропорционально числу реализованных базовых операций. На неустановившихся сетках число базовых операций может не коррелировать со временем перестроения сетки. Это объясняется различными пространственными распределениями локальных модификаций сетки.

Таблица 5.4
Число базовых операций и время перестроения сетки с $N_t(\Omega_h) \sim 160\,000$

L	$P=1$		$P=2$		$P=4$		$P=6$		$P=8$	
	$\#mod$	с	$\#mod$	с	$\#mod$	с	$\#mod$	с	$\#mod$	с
1	30 403	27,2	16 204	15,2	8752	7,9	6448	7,2	5484	6,5
2	30 850	39,6	18 273	27,1	12 340	14,5	8231	11,6	6937	10,8
4	28 027	48,7	22 585	38,4	9804	16,6	6570	12,8	5294	11,8
6	33 018	74,6	19 332	38,8	7643	15,0	4841	11,2	3061	9,4
10	32 986	87,9	13 776	30,2	4996	11,7	2602	8,7	1445	7,2
14	23 878	65,9	10 336	21,5	3741	9,8	1885	7,6	911	6,3
18	25 056	75,8	9802	20,1	2842	8,6	1536	6,3	434	5,7
20	18 904	58,6	7902	15,6	3265	9,7	1384	6,0	785	6,1

Другое интересное наблюдение заключается в том, что число реализованных базовых операций на одном процессоре не всегда обратно пропорционально числу процессоров. На первых адаптивных итерациях суммарное число базовых операций $\#mod \cdot P$ растет с ростом P. Однако, когда сетка установилась, оно уменьшается, что приводит к суперлинейному ускорению времени построения сетки (табл. 5.5). Подчеркнем, что это новое и интересное свойство сеточного генератора.

Таблица 5.5
Ускорение при параллельном перестроении сетки с $N_t(\Omega_h) \sim 160\,000$

P \ L	1	2	3	4	5	6	7	8	9	10	11	12
2	1,8	1,5	1,3	1,3	1,4	1,9	2,2	2,6	2,8	2,9	3,1	3,4
3	2,4	1,9	2,0	2,1	2,9	3,4	4,4	5,3	4,9	5,4	5,1	6,1
4	3,4	2,7	2,3	2,9	3,7	5,0	5,4	6,1	6,5	7,5	7,0	7,5
6	3,8	3,4	3,3	3,8	4,6	6,7	7,5	9,3	10	10,1	9,7	10,1
8	4,2	3,7	3,5	4,1	5,4	7,9	9,4	10,5	11,8	12,2	12,5	12,8

Суперлинейное ускорение обусловлено различными порядками исполнения локальных модификаций сетки. Оказывается, что если сетка слишком далека от установившейся в ходе адаптации, то наиболее эффективная последовательность ее локальных модификаций заключается в выборе тетраэдра с наименьшим качеством. Поэтому суммарное число базовых операций меньше при $P = 1$. Если сетка примерно адаптирована под решение, то более эффективна та последовательность

ее локальных модификаций, которая состоит в выборе наихудших тетраэдров в подсетках. Будучи разделенными в пространстве, такие локальные модификации улучшают качество у большего числа тетраэдров, что приводит к более быстрому уменьшению $\#mod$ на 6 и 8 процессорах.

§ 5.5. Исправление и распутывание сеток

За последние полвека было предложено множество методов построения треугольных сеток высокого качества. К сожалению, аналогичное утверждение неверно для тетраэдральных сеток, поскольку каждый из методов построения тетраэдральных сеток имеет свои достоинства и недостатки. Так, и методом продвигаемого фронта, и методом тетраэдризации Делоне можно построить сильно сплющенные тетраэдры, в которых все четыре вершины лежат практически в одной плоскости (сливеры). Для удаления таких тетраэдров и исправления сетки используются различные методы исправления или улучшения качества сетки. В данном параграфе для исправления сетки мы применим методы перестроения сеток, отличительной особенностью которых является изменение топологии сетки.

Для сложных геометрических моделей генераторы сеток могут производить топологически правильные, но физически некорректные сетки. Например, использование простых методов для закрытия лакун, порождаемых в методе продвигаемого фронта, завершает процесс построения сетки, но может привести к запутанной сетке. В частности, поэтому в § 3.7 мы используем достаточно сложный метод для закрытия лакун. Другой источник запутанных сеток — лагранжевы методы, в которых сетка движется вместе с течением, запутываясь при этом. Методы перестроения сеток помогут исправить сетку в обоих случаях.

Напомним, что качество правильной сетки определяется как положительная функция: $0 < Q(\Omega_h) \leqslant 1$. Чтобы применить базовые алгоритмы перестроения сетки для запутанной сетки, мы обобщим понятие качества симплекса $Q(\Delta)$ следующим образом. Предположим, что запутанная сетка топологически правильная, что верно в подавляющем большинстве случаев. Тогда проверка запутанности сетки сводится к проверке знака алгебраических площадей (в двух измерениях) или алгебраических объемов (в трех измерениях) у пар соседних симплексов. Рассмотрим более подробно треугольные сетки, хотя все сказанное будет справедливо и для тетраэдральных сеток.

В запутанной треугольной сетке всегда найдется пара ячеек Δ_1 и Δ_2 с общим ребром и с одинаковыми знаками алгебраических площадей [см. (5.2.1)], как показано на рис. 5.21, *б*:

$$S_{\Delta_{241}} S_{\Delta_{243}} \geqslant 0. \tag{5.5.1}$$

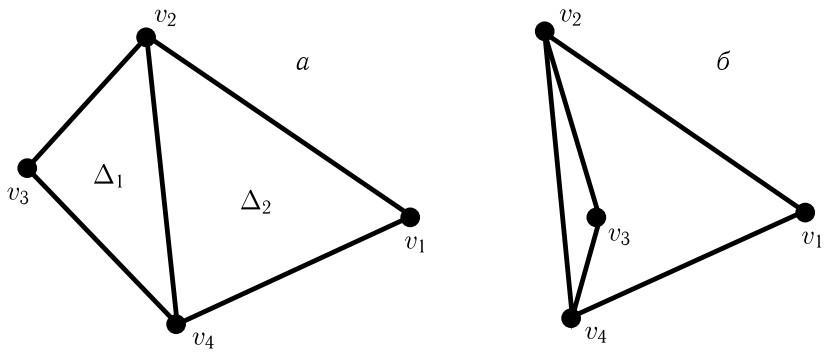


Рис. 5.21. Пары распутанных ($а$) и запутанных ($б$) треугольников

Пометим оба треугольника Δ_1 и Δ_2 как запутанные и припишем им отрицательные качества:

$$\widetilde{Q}(\Delta_1) = -Q(\Delta_1), \qquad \widetilde{Q}(\Delta_2) = -Q(\Delta_2).$$

Качества остальных треугольников не меняются, т. е. $\widetilde{Q}(\Delta) = Q(\Delta)$. Новое качество треугольника $\widetilde{Q}(\Delta)$ меняется непрерывно от -1 до 1 и равняется 0, когда один из запутанных треугольников вырождается в отрезок.

Непрерывность модифицированного качества $\widetilde{Q}(\Delta)$ необходима для успешной реализации базовых алгоритмов. Распутывание сетки происходит за счет роста качества от отрицательного значения, через ноль, к положительному (реальному) качеству. Модифицированное качество не требует изменения базовых алгоритмов, за исключением расширения алгоритма вычисления качества. В дополнение к существующему алгоритму требуется проверить знак в неравенствах (5.5.1) и изменить знак качества треугольника, если это необходимо.

Рассмотрим два примера. Начальная запутанная сетка, показанная на рис. 5.22, $а$, содержит 158 треугольников. Сетка была искусственно запутана путем случайного смещения узлов. Отметим, что некоторые из узлов были смещены за границу области. Качество запутанной сетки $Q(\Omega_h) = -0{,}93$ при среднем качестве треугольников 0,24.

Для исправления этой запутанной сетки мы применим алгоритм 35 с модификациями, описанными в § 5.2. Для сохранения границы области мы зафиксируем узлы сетки на этой границе. Конечная квазиравномерная сетка Ω_h, показанная на рис. 5.22, $б$, содержит 145 треугольников. Ее качество $Q(\Omega_h) = 0{,}53$ при среднем качестве треугольников 0,84.

Второй пример иллюстрирует применение базовых алгоритмов перестроения сетки для улучшения качества тетраэдральной сетки. В данном примере качество тетраэдра определяется как качество его формы, т. е. $Q(\Delta) = 1$ только на правильном тетраэдре любого размера. Детально описание различных качеств приведено в гл. 6.

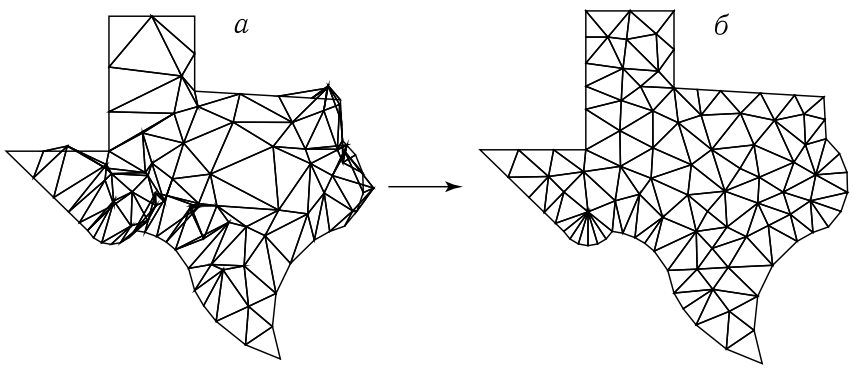


Рис. 5.22. Распутывание треугольной сетки путем ее перестроения

Геометрическая модель шестерни, показанная на рис. 5.23, $а$, была создана с использованием САПР. Она содержит 422 узла, 636 криволинейных ребер и 217 криволинейных поверхностей. Для этой модели были построены две тетраэдральные сетки с различным числом элементов путем комбинации метода продвигаемого фронта с методом тетраэдризации Делоне, как описано в гл. 3. Для построения грубой сетки поверхность шестерни была разбита на 7084 треугольника с 3530 узлами, как показано на рис. 5.23, $б$. Для построения мелкой сетки поверхность модели была разбита на 38 834 треугольника с 19 405 узлами.

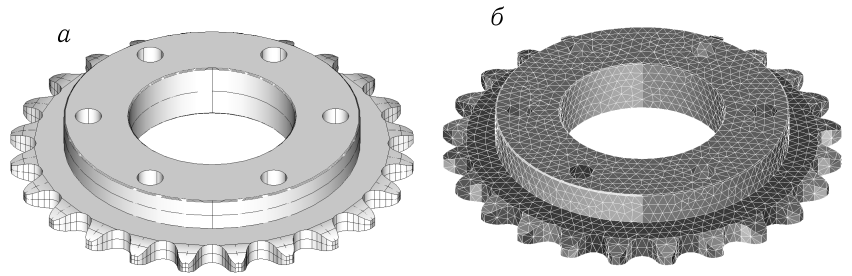


Рис. 5.23. Модель шестерни ($а$) и поверхностная сетка с 7084 треугольниками

Напомним, что надежный алгоритм построения трехмерной сетки (см. § 3.7) применяет последовательно методы продвигаемого фронта (Ф) и тетраэдризации Делоне (Д). Метод продвигаемого фронта недостаточно надежен, и в нашем примере он смог построить сетку только на участке с объемом 99,96 % от объема области. Такое же поведение метода продвигаемого фронта наблюдалось в других примерах, рассмотренных в гл. 3. После завершения работы метода во фронте осталось 32 треугольника в грубой сетке и 352 треугольника в мелкой сетке. Качество недостроенных сеток получилось довольно низким

(см. табл. 5.6 и 5.7). Сетки были достроены методом тетраэдризации Делоне. После этого качество сеток упало еще сильнее — до $6{,}1 \cdot 10^{-4}$ в мелкой сетке и до $1{,}6 \cdot 10^{-5}$ в грубой сетке. Детальное распределение тетраэдров по качеству представлено в табл. 5.6 и 5.7.

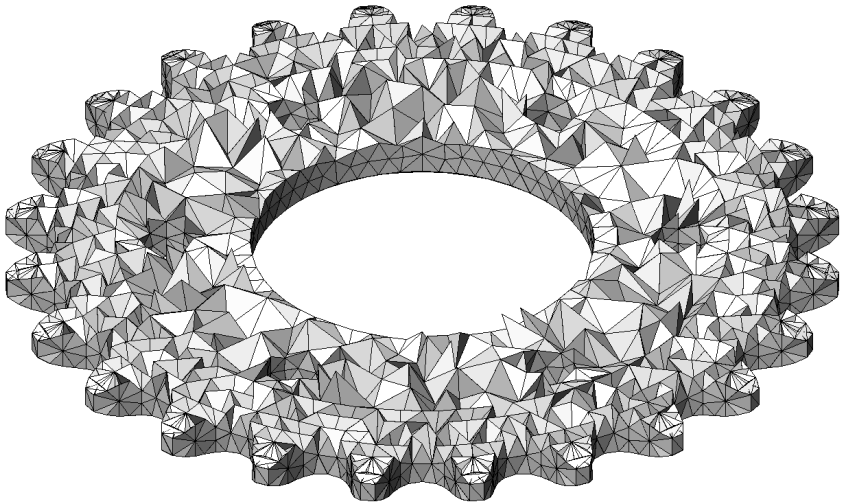


Рис. 5.24. Срез пространственной сетки для шестерни. Сетка содержит 6870 узлов, 7084 граничных треугольника и 22 456 тетраэдров

Таблица 5.6

Распределение тетраэдров по качеству в грубой сетке

Метод	$Q(\Omega_h)$	N_t	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
Ф	$1{,}06 \cdot 10^{-2}$	13 655	13 375	280	0	0	0
Ф+Д	$6{,}12 \cdot 10^{-4}$	13 687	13 390	290	5	2	0
Ф+Д+ПС	$1{,}44 \cdot 10^{-1}$	25 485	25 485	0	0	0	0

Алгоритмы перестроения сеток (ПС) значительно улучшили их качество — до 0,14 в грубой сетке и до 0,2 в мелкой сетке. Тетраэдры низкого качества, появившиеся после тетраэдризации Делоне, были успешно перестроены. Часть пространственной сетки показана на рис. 5.24. Построение мелкой сетки потребовало 9 мин 33 с. Большинство ресурсов было затрачено на формирование начального фронта:

Таблица 5.7

Распределение тетраэдров по качеству в мелкой сетке

Метод	$Q(\Omega_h)$	N_t	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
Ф	$1{,}01 \cdot 10^{-3}$	140 442	139 684	750	8	0	0
Ф+Д	$1{,}60 \cdot 10^{-5}$	140 723	139 817	854	37	5	10
Ф+Д+ПС	$2{,}00 \cdot 10^{-1}$	156 538	156 538	0	0	0	0

8 мин 40 с. Построение пространственной сетки заняло 41 с, а ее перестроение — всего лишь 12 с.

Другие примеры исправления тетраэдральных сеток представлены в § 3.7.

Глава 6

УПРАВЛЕНИЕ СВОЙСТВАМИ СЕТОК

В данной главе мы рассмотрим различные методы построения сеток с заданными свойствами, включая контроль локального размера симплексов (треугольников или тетраэдров), вытягивание симплексов в заданном направлении и контроль числа симплексов в сетке. Контроль осуществляется расщеплением качества симплекса на множители, отвечающие за его размер и форму.

§ 6.1. Управление свойствами регулярных сеток

В регулярной сетке симплексы могут значительно различаться по размеру, оставаясь близкими по форме к правильным симплексам. Более точно, каждый симплекс Δ в регулярной сетке удовлетворяет следующему условию:

$$\frac{R_\Delta}{r_\Delta} \leqslant C,$$

где R_Δ и r_Δ обозначают радиусы описанной и вписанной сфер соответственно, а константа C не зависит от симплекса. При $C \sim 1$ регулярная сетка не содержит сливеров и анизотропных симплексов.

Регулярные сетки представляют важный класс сеток для приложений. Во-первых, они могут сгущаться к особенностям модели или к особенностям решения по закону, заданному пользователем. Во-вторых, подавляющее большинство теоретических результатов по устойчивости численных методов и оценкам точности численного решения доказаны для регулярных сеток.

Для построения регулярных сеток необходимо задать функцию шага сетки $h(\mathbf{x})$, определяющую желаемый диаметр симплекса в точке $\mathbf{x}$. В данном параграфе мы предположим, что функция $h(\mathbf{x})$ известна, и рассмотрим три метода построения регулярных сеток.

Рассмотрим единичный квадрат $[0, 1]^2$ и построим там регулярную сетку, которая сгущается к точке $(0, 0)$. Определим функцию $h(\mathbf{x})$ в виде

$$h(\mathbf{x}) = \left(10^{-4} + 10^{-2}\,\|\mathbf{x}\|^2\right)^{1/2}, \qquad (6.1.1)$$

где $\|\mathbf{x}\|$ обозначает евклидову норму вектора $\mathbf{x}$, $\|\mathbf{x}\|^2 = x^2 + y^2$. Неструктурированная сетка, построенная методом продвигаемого фронта с шагом сетки $h(\mathbf{x})$, показана на рис. 6.1, *а*. Она содержит

999 треугольников и 540 узлов. Сетка, построенная методом многоуровнего иерархического измельчения, показана на рис. 6.1, *б*. Она содержит 908 треугольников и 494 узла.

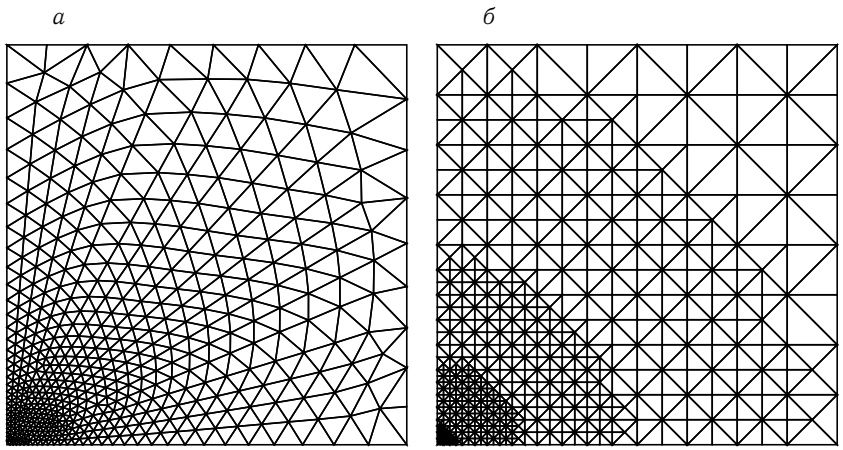


Рис. 6.1. Сетки, построенные методом продвигаемого фронта (*а*) и методом многоуровнего иерархического измельчения (*б*)

У обеих сеток имеются преимущества и недостатки. В неструктурированной сетке размер треугольников меняется гладко, без больших скачков. При численном решении дифференциальных уравнений в частных производных гладкость сетки может привести к более точному дискретному решению. С другой стороны, реализация ряда численных методов проще на иерархических или структурированных сетках. Иерархичность сетки позволяет использовать более эффективные методы решения возникающих алгебраических задач и интерполяции сеточных решений между двумя сетками в нестационарных задачах.

Третий метод построения регулярных сеток использует алгоритмы перестроения сеток, описанные в гл. 5. Эти алгоритмы требуют ввести качество симплекса $Q(\Delta)$. Большинство существующих подходов определяют качество симплекса как произведение двух множителей:

$$Q(\Delta) = Q_{\text{size}}(\Delta)\, Q_{\text{shape}}(\Delta),$$

каждый из которых меняется от 0 до 1. Первый множитель отвечает за размер симплекса, второй — за его форму. Пусть p_d и V_d обозначают соответственно сумму длин ребер и объем правильного симплекса с единичным ребром, где $d = 2$ для треугольника и $d = 3$ для тетраэдра. Несложные вычисления дают:

$$p_d = \frac{d(d+1)}{2}, \qquad V_d = \frac{1}{d!}\sqrt{\frac{d+1}{2^d}}.$$

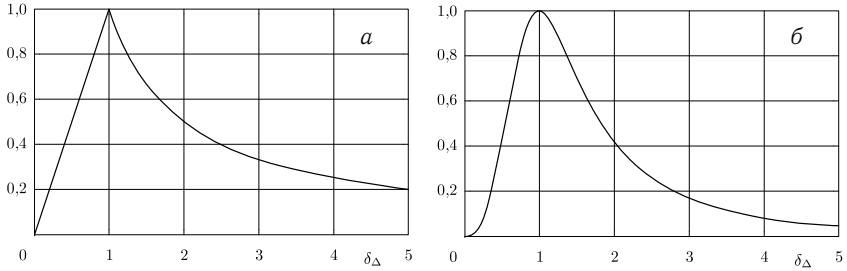


Рис. 6.2. Качество размера $Q_{\text{size}}(\Delta)$, вычисленное по формулам (6.1.2) (*а*) и (6.1.3) (*б*) как функция от δ_Δ

Множитель $Q_{\text{size}}(\Delta)$ строится обычно на основе функции шага сетки $h(\mathbf{x})$, например:

$$Q_{\text{size}}(\Delta) = \min\left\{\delta_\Delta,\, \frac{1}{\delta_\Delta}\right\}, \quad \delta_\Delta = \frac{p_\Delta}{p_d h(\mathbf{x}_\Delta)}, \tag{6.1.2}$$

где $\mathbf{x}_\Delta$ — центр масс симплекса Δ, p_Δ — сумма длин его ребер:

$$p_\Delta = \sum_{e\in\Delta} l_e.$$

Таким образом, независимо от значения δ_Δ качество $Q_{\text{size}}(\Delta) \leqslant 1$.

Такое определение $Q_{\text{size}}(\Delta)$ приводит к разрывности производных $Q_{\text{size}}(\Delta)$ по отношению к δ_Δ и координатам вершин симплекса. В базовых алгоритмах оптимизации положения узла сетки дифференцируемость качества симплекса является не обязательным, но желательным свойством. Поэтому в приведенных ниже примерах мы определим качество симплекса, используя следующую диффренцируемую функцию F:

$$Q_{\text{size}}(\Delta) = F(\delta_\Delta) = \left[\min\left\{\delta_\Delta,\, \frac{1}{\delta_\Delta}\right\}\left(2 - \min\left\{\delta_\Delta,\, \frac{1}{\delta_\Delta}\right\}\right)\right]^3. \tag{6.1.3}$$

Данное качество возрастает при $0 < \delta_\Delta \leqslant 1$, убывает при $\delta_\Delta > 1$ и имеет единственный максимум ($Q_{\text{size}}(\Delta) = 1$) в точке $\delta_\Delta = 1$ (см. рис. 6.2).

Множитель $Q_{\text{shape}}(\Delta)$, отвечающий за форму симплекса, также может быть определен неединственным способом. В приведенных ниже примерах мы используем следующее качество:

$$Q_{\text{shape}}(\Delta) = \frac{p_d^d}{V_d}\,\frac{V_\Delta}{p_\Delta^d}. \tag{6.1.4}$$

В двумерном пространстве другая распространенная формула для качества формы использует вместо p_Δ^2 сумму квадратов длин ребер. Отметим, что $Q_{\text{shape}}(\Delta)$ — безразмерная величина с максимальным значением 1, которое достигается только на правильном симплексе. В общем случае любая комбинация безразмерных величин, которая достигает единственного максимума на правильном симплексе, может быть использована как определение качества формы. Для более

детального анализа различных качеств симплекса мы отсылаем читателя к работе [62].

Положим

$$Q(\Delta) = \frac{p_d^d}{V_d}\,\frac{V_\Delta}{p_\Delta^d}\,F(\delta_\Delta), \quad \delta_\Delta = \frac{p_\Delta}{p_d h(\mathbf{x}_\Delta)}. \tag{6.1.5}$$

Таким образом, максимальное значение качества $Q(\Delta)$ достигается на правильном симплексе с длиной ребра $l_e = h(\mathbf{x}_\Delta)$. По сравнению с формулами из § 2.1 формула (6.1.5) содержит дополнительный множитель $F(\delta_\Delta)$.

В табл. 6.1 приведены значения качества формы (6.1.4) и качества размера (6.1.3) для равнобедренного треугольника с углами α, $\beta = 180° - 2\alpha$ и основанием $h(\mathbf{x}_\Delta) = 1$. Отметим, что качество формы $Q_{\text{shape}}(\Delta) = 0{,}8$ может означать как тупоугольный, так и остроугольный равнобедренные треугольники. Этого недостаточно при построении хороших сеток для конечноэлементных расчетов. Качество размера имеет более сильно выделенный максимум $\alpha = 60°$, так что в результате качество $Q(\Delta) \geqslant 0{,}8$ достигается только на остроугольном треугольнике.

Баланс качеств формы и размера может регулироваться многими способами, например,

$$Q(\Delta) = \big(Q_{\text{shape}}(\Delta)\big)^n \big(Q_{\text{size}}(\Delta)\big)^m,$$

где n и m — некоторые положительные параметры. Таким образом, при анализе свойств построенных сеток необходимо использовать аналог табл. 6.1 для каждого нового определения качества треугольника или тетраэдра.

Таблица 6.1
Качество равнобедренного треугольника, вычисленное по формуле (6.1.5)

α	5°	15°	25°	35°	45°	55°	65°	75°	85°
β	170°	150°	130°	110°	90°	70°	50°	30°	10°
$Q_{\text{shape}}(\Delta)$	0,11	0,34	0,55	0,74	0,89	0,99	0,98	0,82	0,38
$Q_{\text{size}}(\Delta)$	0,70	0,72	0,76	0,81	0,89	0,98	0,96	0,62	0,08
$Q(\Delta)$	0,08	0,24	0,41	0,60	0,79	0,96	0,95	0,51	0,03

На практике построение идеальной сетки с качеством $Q(\Omega_h) = 1$ нереально. Во-первых, не существует двумерной сгущающейся сетки из равносторонних треугольников. Во-вторых, тетраэдральной сеткой из правильных тетраэдров нельзя покрыть пространство. В-третьих, двугранные и плоские углы в геометрической модели ограничивают максимально возможное качество сетки. Эти ограничения следует учитывать при выборе качества Q_0 в алгоритме 35 перестроения сетки.

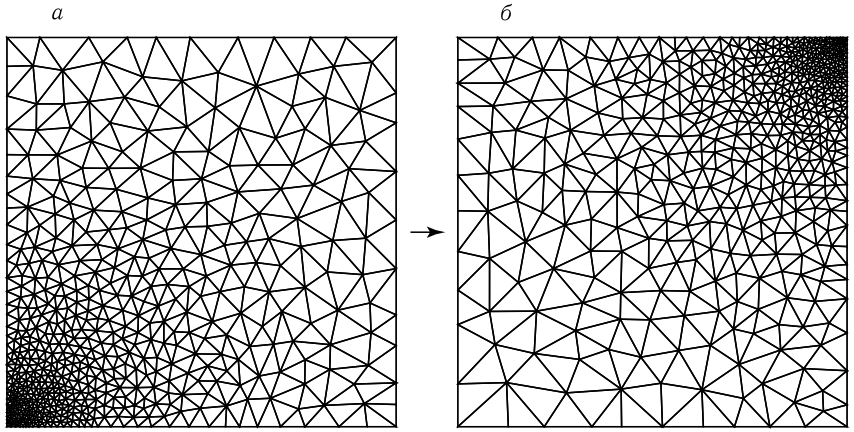


Рис. 6.3. Сетки, порожденные алгоритмом перестроения сетки

Вернемся к функции (6.1.1). Результат работы алгоритмов перестроения сетки показан на рис. 6.3, *а*. Сетка содержит 1004 треугольника и 545 узлов. Несмотря на визуальное ощущение негладкости сетки, ее качество достаточно высокое: $Q(\Omega_h) = 0{,}797$. Гибкость этого подхода заключается в возможности перестроить любую сетку в сетку с новыми свойствами заменой функции шага сетки $h(\mathbf{x})$. Например, выбор

$$h(\mathbf{x}) = \left(10^{-4} + 10^{-2}\,\|\mathbf{x} - \mathbf{e}\|^2\right)^{1/2}, \qquad \mathbf{e} = (1,\,1)^{\mathrm{T}},$$

приводит к сетке, показанной на рис. 6.3, *б*. Эта сетка была получена путем перестроения сетки на рис. 6.3, *а*.

§ 6.2. Управление свойствами анизотропных сеток

Регулярные сетки высокого качества, рассмотренные в § 6.1, состоят обычно из треугольников, близких по форме к правильному треугольнику. Размер этих треугольников контролируется одним параметром — локальным шагом сетки. Построение анизотропных сеток, где ориентация треугольников играет важную роль, требует введения дополнительных параметров, которые неизбежно приводят к понятию тензорной метрики.

В евклидовой метрике расстояние между двумя точками $\mathbf{x}_1$ и $\mathbf{x}_2$ на плоскости измеряют по теореме Пифагора:

$$\rho(\mathbf{x}_1,\,\mathbf{x}_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Обобщение понятия расстояния требует введения матрицы $\mathbb{M}$ — симметричной положительно определенной матрицы 2×2. Расстояние

между двумя точками $\mathbf{x}_1$ и $\mathbf{x}_2$ в постоянной метрике $\mathbb{M}$ вычисляется следующим образом:

$$\rho_M(\mathbf{x}_1,\,\mathbf{x}_2) = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^{\mathrm{T}}\,\mathbb{M}\,(\mathbf{x}_1 - \mathbf{x}_2)}. \tag{6.2.1}$$

Расстояние между двумя точками в пространстве вычисляется по аналогичной формуле, только в этом случае $\mathbb{M}$ — симметричная положительно определенная матрица 3×3.

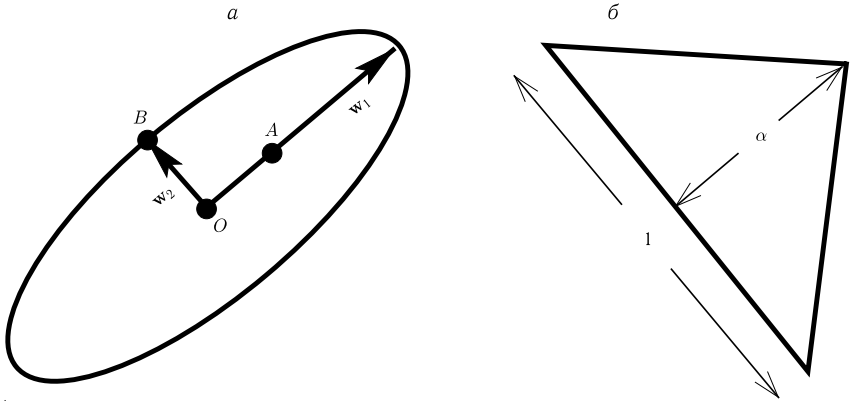


Рис. 6.4: *а* — собственные векторы 2×2 матрицы $\mathbb{M}$, *б* — высота треугольника равна $\alpha = \sqrt{3\lambda_2/(4\lambda_1)}$

Собственные векторы $\mathbf{w}_1$ и $\mathbf{w}_2$ конкретной матрицы

$$\mathbb{M} = \begin{pmatrix} 5{,}89 & 2{,}51 \\ 2{,}51 & 5{,}01 \end{pmatrix}$$

показаны на рис. 6.4, *а* как полуоси эллипса с центром в точке O. Они соответствуют собственным значениям $\lambda_1 = 8{,}0$ и $\lambda_2 = 2{,}9$. В евклидовой метрике точки A и B равноудалены от центра O. В метрике $\mathbb{M}$ расстояние до точки A в $\sqrt{\lambda_1/\lambda_2}$ раз больше, чем до точки B. Равносторонний в метрике $\mathbb{M}$ треугольник показан на рис. 6.4, *б*. Высота этого треугольника в $\sqrt{\lambda_2/\lambda_1}$ раз меньше высоты треугольника, равностороннего в евклидовой метрике.

Для сильно анизотропной метрики, у которой $\lambda_1 \gg \lambda_2$, показанный на рис. 6.4, *б* треугольник, равносторонний в метрике $\mathbb{M}$, будет выглядеть, как сильно сплющенный в евклидовой метрике. Другая особенность анизотропной метрики заключается в том, что при повороте равностороннего треугольника в метрике $\mathbb{M}$ его форма будет сильно меняться. Например, при повороте на 90°, вместо сплющенного треугольника мы получим сильно вытянутый треугольник в форме иглы.

Отношение собственных значений λ_1/λ_2 называется *числом обусловленности* матрицы $\mathbb{M}$, или анизотропией метрики. Последний

пример показывает, что увеличение анизотропии метрики в 100 раз приводит к вытягиванию (или сплющиванию) треугольника в 10 раз. В трехмерном пространстве анизотропия метрики определяется как отношение максимального собственного значения матрицы $\mathbb{M}$ к минимальному.

Дальнейшее обобщение понятия расстояния предполагает, что элементы матрицы $\mathbb{M}$ являются функциями от пространственной координаты $\mathbf{x}$. Отметим, что $\mathbb{M}(\mathbf{x})$ остается симметричной и положительно определенной матрицей в каждой точке $\mathbf{x}$. Пусть $\boldsymbol{\gamma}(t)$ — простейшая параметризация ребра $e = [\mathbf{x}_1, \mathbf{x}_2]$:

$$\boldsymbol{\gamma}(t) = \mathbf{x}_1 + t(\mathbf{x}_2 - \mathbf{x}_1), \qquad 0 \leqslant t \leqslant 1,$$

для которой $\boldsymbol{\gamma}'_t(t) = \mathbf{x}_2 - \mathbf{x}_1$. Длина ребра e вычисляется следующим образом:

$$l_{e,\mathbb{M}} = \int\limits_0^1 \sqrt{\boldsymbol{\gamma}_t'^{\mathrm{T}}\, \mathbb{M}(\boldsymbol{\gamma}(t))\, \boldsymbol{\gamma}'_t}\, dt = \int\limits_0^1 \sqrt{(\mathbf{x}_2 - \mathbf{x}_1)^{\mathrm{T}}\, \mathbb{M}(\boldsymbol{\gamma}(t))\, (\mathbf{x}_2 - \mathbf{x}_1)}\, dt. \quad (6.2.2)$$

В случае постоянной метрики формулы (6.2.2) и (6.2.1) совпадают. Отметим также, что в случае переменной метрики прямая не всегда является кратчайшим расстоянием между двумя точками.

При перестроении сеток нет необходимости вычислять точную длину ребра в метрике $\mathbb{M}$ по формуле (6.2.2). Достаточно применить квадратуру второго порядка с центральной точкой:

$$l_{e,\mathbb{M}} \approx \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^{\mathrm{T}}\, \mathbb{M}(\mathbf{x}_{12})\, (\mathbf{x}_1 - \mathbf{x}_2)}, \qquad \mathbf{x}_{12} = \frac{1}{2}\,(\mathbf{x}_1 + \mathbf{x}_2).$$

На практике при перестроении сетки метрика вычисляется или в каждом симплексе, или в каждом узле сетки. В первом случае метрика кусочно-постоянна и разрывна между симплексами, т. е. длина ребра сетки зависит от симплекса, на котором она вычисляется. Во втором случае метрика считается линейной в каждом симплексе и непрерывной во всей области, поэтому длина ребра одинакова для всех симплексов, содержащих это ребро. Поэтому непрерывная метрика приводит к более быстрому перестроению сетки. Все примеры перестроения сеток, приведенные в этой книге, используют непрерывную метрику.

В евклидовой метрике объем симплекса вычисляется по формуле

$$V_\Delta = \int\limits_\Delta dV.$$

В метрическом пространстве единица длины в направлении собственного вектора умножается на корень квадратный из соответствующего собственного значения. Поскольку детерминант матрицы $\mathbb{M}$ — произведение ее собственых значений, имеем следующую формулу для объема:

$$V_{\Delta,\mathbb{M}} = \int\limits_\Delta \sqrt{\det \mathbb{M}(\mathbf{x})}\, dV. \quad (6.2.3)$$

В случае постоянной метрики мы получаем соотношение

$$V_{\Delta,\mathbb{M}} = V_\Delta \sqrt{\det \mathbb{M}}.$$

Используя формулы для длины ребра и объема симплекса в метрике $\mathbb{M}$, мы определим новое качество симплекса как произведение качеств размера и формы:

$$Q(\Delta) = Q_{\mathrm{size},\mathbb{M}}(\Delta)\, Q_{\mathrm{shape},\mathbb{M}}(\Delta), \quad (6.2.4)$$

где

$$Q_{\mathrm{size},\mathbb{M}}(\Delta) = F(\delta_{\Delta,\mathbb{M}}), \quad Q_{\mathrm{shape},\mathbb{M}}(\Delta) = \frac{p_d^d}{V_d}\, \frac{V_{\Delta,\mathbb{M}}}{p_{\Delta,\mathbb{M}}^d}. \quad (6.2.5)$$

Здесь функция F определяется формулой (6.1.3), где

$$\delta_{\Delta,\mathbb{M}} = \frac{p_{\Delta,\mathbb{M}}}{p_d\, h_{\mathbb{M}}(\mathbf{x}_\Delta)}, \qquad p_{\Delta,\mathbb{M}} = \sum_{e \in \Delta} l_{e,\mathbb{M}},$$

а функция $h_{\mathbb{M}}(\mathbf{x}_\Delta)$ задает шаг регулярной сетки в пространстве с метрикой $\mathbb{M}$. Новое качество симплекса построено таким образом, что оно совпадает с качеством (6.1.5) для изотропной метрики:

$$\mathbb{M}(\mathbf{x}) = h(\mathbf{x})^{-2}\, I_d, \qquad h_{\mathbb{M}}(\mathbf{x}_\Delta) = 1,$$

где I_d обозначает единичную матрицу порядка d. Действительно, используя формулы (6.2.3) и (6.2.2), мы получим $V_{\Delta,\mathbb{M}} = V_\Delta h(\mathbf{x}_\Delta)^{-d}$ и $p_{\Delta,\mathbb{M}} = h(\mathbf{x}_\Delta)^{-1} p_\Delta$, что преобразует формулы (6.2.4) и (6.2.5) к формуле (6.1.5).

Качество (6.2.4) достигает максимума, когда $Q_{\mathrm{size},\mathbb{M}}(\Delta) = Q_{\mathrm{shape},\mathbb{M}}(\Delta) = 1$. Первый множитель равен 1, если все ребра Δ имеют одинаковую длину $h_{\mathbb{M}}(\mathbf{x}_\Delta)$ в метрике $\mathbb{M}$. Второй множитель равен 1, когда отношение $V_{\Delta,\mathbb{M}}/p_{\Delta,\mathbb{M}}^d$ достигает максимального значения, т. е. также на правильном симплексе, рассматриваемом в метрике $\mathbb{M}$.

Таким образом, в метрическом пространстве нет необходимости введения отдельной функции $h(\mathbf{x})$, описывающей размер симплекса, и параметров, определяющих его ориентацию. Метрика $\mathbb{M}$ контролирует как ориентацию симплекса, так и его размер.

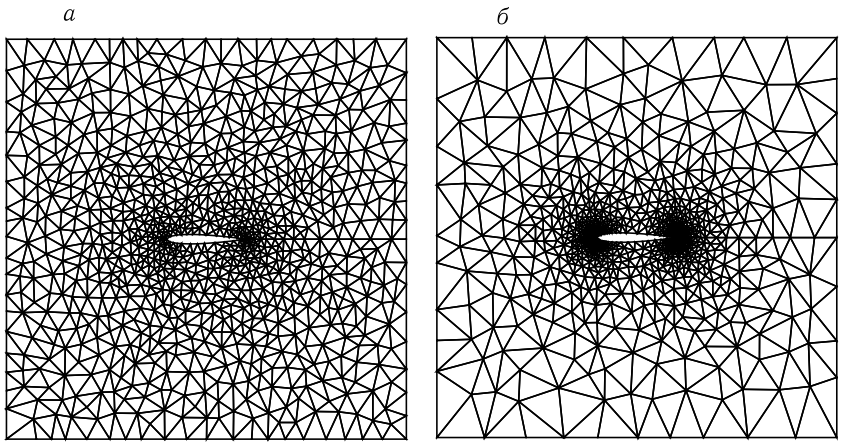


Рис. 6.5. Изотропные сетки с $a = 0{,}5$ (*а*) и $a = 1$ (*б*)

Рассмотрим пример контроля свойств регулярных сеток. Используя изотропную метрику $\mathbb{M} = h(\mathbf{x})^{-2}\, I_2$, где

$$h(\mathbf{x}) = \max\{|(x-0{,}4)^2 + (y-0{,}5)^2 + 10^{-4}|^{a/2}, \\ |(x-0{,}6)^2 + (y-0{,}5)^2 + 10^{-4}|^{a/2}\},$$

и изменяя значение параметра a, мы будем изменять степень сгущения изотропной сетки к двум точкам (0,4, 0,5) и (0,6, 0,5) на границе препятствия в форме крыла (см. рис. 6.5). Обе сетки на этом рисунке содержат около 1800 треугольников каждая и имеют примерно одинаковое качество. Отметим, что сетка на рис. 6.5, *б* была построена из сетки на рис. 6.5, *а* с помощью методов перестроения треугольных сеток, описанных в гл. 5.

Основной недостаток введенной выше метрики — отсутствие контроля числа симплексов в построенной сетке. Для возвращения этого контроля необходимо установить соответствие между желаемым числом симплексов $N_\star$ и параметром $h_{\mathbb{M}}(\mathbf{x}_\Delta)$. В идеальной сетке, состоящей из правильных симплексов одинакового диаметра в метрике $\mathbb{M}$, этот параметр не зависит от симплекса Δ, поэтому

$$h_{\mathbb{M}}(\mathbf{x}_\Delta) = h_\star, \tag{6.2.6}$$

где величину $h_\star$ нужно оценить. Площадь (объем) расчетной области в двух (трех) измерениях вычисляется по формуле

$$|\Omega|_{\mathbb{M}} = N_\star\, V_d\, h_\star^d.$$

Таким образом, для оценки параметра $h_\star$ достаточно вычислить площадь (объем) области Ω в заданной метрике $\mathbb{M}$:

$$h_\star = \left(\frac{1}{N_\star\, V_d} \int\limits_\Omega \sqrt{\det \mathbb{M}(\mathbf{x})}\, dV\right)^{1/d}.$$

Используя некоторую триангуляцию (тетраэдризацию) Ω_h^0 расчетной области и применяя квадратуру с центральной точкой, мы получим простую оценку этого параметра:

$$h_\star \approx \left(\frac{1}{N_\star\, V_d} \sum_{i=1}^{N(\Omega_h^0)} V_{\Delta_i} \sqrt{\det \mathbb{M}(\mathbf{x}_{\Delta_i})}\right)^{1/d}. \tag{6.2.7}$$

Таким образом, для построения сетки с $N_\star$ симплексами мы сперва вычисляем параметр $h_\star$ по формуле (6.2.7), используя некоторую начальную триангуляцию (тетраэдризацию) области, а затем качество симплекса по формулам (6.2.4)–(6.2.6). В случае грубой начальной оценки параметра $h_\star$ число симплексов в построенной сетке $N(\Omega_h)$ может значительно отличаться от $N_\star$. В этом случае построенную сетку можно использовать для более точной оценки параметра $h_\star$ и повторить процесс перестроения сетки.

Как мы отмечали выше, в подавляющем большинстве задач построить сетку с максимальным качеством $Q(\Omega_h) = 1$ нереально. Поэтому желаемое число симплексов $N_\star$ почти всегда будет достигнуто лишь приближенно. Отклонение $N_\star$ от $N(\Omega_h)$ зависит от качества $Q(\Omega_h)$ построенной сетки. Чем ниже среднее качество элементов сетки, тем больше отклонение $N(\Omega_h)$ от желаемого значения.

Рассмотрим пример контроля свойств анизотропных сеток. Пусть

$$\mathbb{M} = R_{\pi/4} \begin{pmatrix} 10^a & 0 \\ 0 & 1 \end{pmatrix} R_{\pi/4}^{\mathrm{T}},$$

где $R_{\pi/4}$ — матрица поворота на угол 45°. Таким образом, с увеличением a метрика становится сильнее в направлении $(-1, 1)$. Сетка, квазиравномерная в такой метрике, должна вытягиваться в перпендикулярном направлении $(1, 1)$.

Пусть $N_\star = 2000$. Рассмотрим два значения $a = 1$ и $a = 2$. В обоих случаях мы оценим $h_\star$, используя сетку, показанную на рис. 6.3, *а*. Результат перестроения сетки показан на рис. 6.6. Сетка на рис. 6.6, *а* содержит 1850 треугольников, а ее качество $Q(\Omega_h) = 0{,}574$ при среднем качестве элементов 0,870. Только 9 треугольников имеют качество ниже 0,8. Отметим, что, несмотря на относительно высокое среднее качество, отклонение $N(\Omega_h)$ от $N_\star$ составляет 7,5 %.

Сетка на рис. 6.6, *б* содержит 1867 треугольников, а ее качество $Q(\Omega_h) = 0{,}208$ при среднем качестве элементов 0,864. Несколько треугольников вблизи левого верхнего и правого нижнего углов не могут

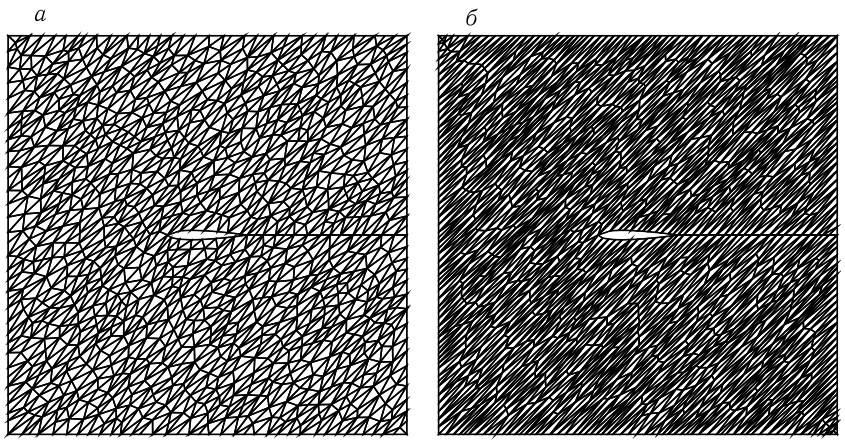


Рис. 6.6. Анизотропные сетки с $a = 1$ ($а$) и $a = 2$ ($б$)

быть сильно вытянуты в направлении $(1, 1)$, что приводит к уменьшению их качества и, следовательно, качества сетки. Тем не менее, ввиду высокого среднего качества треугольников, отклонение $N(\Omega_h)$ от $N_\star$ составляет лишь 6,7 %.

Наш опыт показывает, что построение сильно анизотропных тетраэдральных сеток — значительно более сложная проблема, чем построение анизотропных треугольных сеток. На практике алгоритмы, описанные в гл. 5, позволяют строить треугольные сетки, в которых элементы могут быть легко вытянуты в 10^3–10^4 раз в каком-либо направлении. Для тетраэдральных сеток степень вытягивания элементов ограничена обычно значениями 10–100.

Приложение

НЕКОТОРЫЕ ПРОБЛЕМЫ СЕТОЧНОЙ АДАПТАЦИИ

Под *сеточной адаптацией* мы понимаем процесс построения или перестроения расчетной сетки, согласованной с геометрическими особенностями расчетной области и/или с особенностями сеточного решения. Процесс сеточной адаптации, как правило, направлен на приближенную минимизацию некоторого функционала при заданных ограничениях. В приложении мы рассмотрим несколько характерных примеров таких функционалов и ограничений при перестроении некоторой начальной симплициальной сетки.

§ П.1. Адаптация к внешним и внутренним границам

Распространенным примером сеточной адаптации является адаптация начальной сетки к криволинейной границе, которая может быть как внешней, так и внутренней. Основной целью этой адаптации является улучшение приближения исходной границы граничными ребрами или гранями адаптивной сетки. В качестве минимизируемого функционала можно выбрать, например, ошибку аппроксимации криволинейной границы кусочно-линейной границей в максимальной норме, а в качестве ограничения — максимально допустимое количество сеточных узлов. Такая постановка соответствует проблеме построения адекватной дискретной модели в классе симплициальных сеток для заданной области с криволинейной границей.

Очевидно, что уменьшение ошибки аппроксимации криволинейной границы кусочно-линейной границей подразумевает измельчение заданной сетки в окрестности границы, т. е. улучшение разрешения исходной дискретной модели. В зависимости от метода измельчения сетки необходимо реализовать или сдвиг граничных узлов вдоль границы, или постановку новых узлов на границу. Обе операции предполагают наличие данных о криволинейной границе.

Ниже мы рассмотрим несколько способов использования данных о криволинейной границе для адаптации сетки в ее окрестности. Для простоты изложения ограничимся случаем, когда все граничные узлы исходной сетки лежат на границе области.

П.1.1. Адаптация триангуляции к криволинейной параметризованной границе

В двумерном случае криволинейная граница может быть параметризована набором функций от одной действительной переменной, называемой параметром t. При этом граница покрывается конечным набором кривых, каждая из которых описывается непрерывными функциями координат точек:

$$\Gamma = \bigcup_{i=1}^{m} \Gamma_i,$$

где

$$\Gamma_i = \{(x, y) \mid x = \chi_i(t),\ y = \gamma_i(t),\ t \in [t_i^b, t_i^e]\}.$$

Здесь t_i^b, t_i^e — концы кривой Γ_i в параметрическом пространстве. Введенные обозначения показаны на рис. П.1.

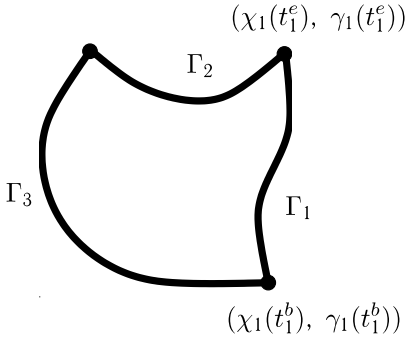


Рис. П.1. Пример области с тремя криволинейными границами

Поскольку все граничные узлы исходной сетки лежат на границе Γ, то для любой точки $v \in \Gamma_i$ с координатами (x_v, y_v), лежащей между двумя ближайшими граничными узлами v_1 и v_2 с параметрами t_1, t_2 соответственно, существует параметр $t_v \in [t_1, t_2]$ такой, что $x_v = \chi_i(t_v)$, $y_v = \gamma_i(t_v)$.

Таким образом, сдвиг любой точки v с параметром t_v вдоль сегмента границы Γ_i сводится к последовательному увеличению или уменьшению параметра t_v. Вставка нового узла v с координатами (x_v, y_v) и параметром t_v в середину криволинейного отрезка, параметризованного функциями χ, γ и соединяющего две вершины v_1, v_2 с параметрами t_1, t_2, зависит от определения середины криволинейного отрезка. Если середина является образом середины отрезка в параметрическом пространстве, то

$$t_v = (t_1 + t_2)\ /2. \qquad \text{(П.1.1)}$$

Если середина криволинейного отрезка — это точка, равноудаленная от вершин v_1 и v_2, то t_v является решением нелинейного уравнения

$$(x_1 - \chi(t_v))^2 + (y_1 - \gamma(t_v))^2 = (x_2 - \chi(t_v))^2 + (y_2 - \gamma(t_v))^2. \qquad \text{(П.1.2)}$$

Корень t_v, принадлежащий интервалу (t_1,t_2), можно найти методом бисекции или методом Ньютона. Если середина криволинейного отрезка — точка, пути от которой вдоль отрезка до вершин v_1 и v_2 равны, то t_v является решением более сложного нелинейного уравнения

$$\int_{t_1}^{t_v} \sqrt{\chi'^2(t) + \gamma'^2(t)}\, dt = \int_{t_2}^{t_v} \sqrt{\chi'^2(t) + \gamma'^2(t)}\, dt. \qquad \text{(П.1.3)}$$

В табл. П.1 приведены значения параметра t_v для середины криволинейного отрезка (части параболы), определенного следующим образом:

$$\chi(t) = t, \quad \gamma(t) = t^2, \quad 0 \leqslant t \leqslant 1.$$

Таблица П.1
Параметр середины отрезка, вычисленный по формулам (П.1.1)–(П.1.3).

Формула	(П.1.1)	(П.1.2)	(П.1.3)
t_v	0,5	0,618	0,611

Хотя с математической точки зрения определение середины криволинейного отрезка по формуле (П.1.3) является единственно верным, наш практический опыт показывает, что использование более простых определений (П.1.1) и (П.1.2) не приводит к серьезным искажениям качества сетки при ее адаптации к криволинейной границе, поскольку в методах перестроения сетки (гл. 5) положение середины отрезка корректируется базовой операцией сдвига узла. Отметим также, что если длина сеточного ребра стремится к нулю, то решения, найденные по формулам (П.1.2) и (П.1.3), сходятся друг к другу со вторым порядком.

П.1.2. Адаптация тетраэдризации к гладкой параметризованной поверхности

Для трехмерных областей доступность параметризации внешних или внутренних границ зависит от конкретного приложения. В ряде случаев такие параметризации известны явно, как, например, при задании границы конечным набором параметризованных поверхностей. В других случаях эти параметризации известны неявно, как, например, при взаимодействии с ядром САПР.

При этом, независимо от способа задания, каждый кусок поверхности Γ_i параметризован тремя непрерывными функциями от двух действительных переменных, называемых параметрами p и s, которые

заданы в некоторых областях ω_i двумерного параметрического пространства. Пусть

$$\Gamma = \bigcup_{i=1}^{m} \Gamma_i,$$

где

$$\Gamma_i = \{(x, y, z) : x = \chi_i(p, s),\ y = \gamma_i(p, s),\ z = \zeta_i(p, s),\ (p, s) \in \omega_i\}.$$

В силу непрерывности параметризующих функций для любой точки $v \in \Gamma_i$ с координатами (x_v, y_v, z_v) существует точка в параметрической области $(p_v, s_v) \in \omega_i$ такая, что $x_v = \chi_i(p_v, s_v)$, $y_v = \gamma_i(p_v, s_v)$ и $z_v = \zeta_i(p_v, s_v)$.

Одна из базовых операций при перестроении сетки — это сдвиг узла вдоль криволинейной поверхности. Сдвиг точки v с параметрами (p_v, s_v) вдоль куска параметризованной поверхности Γ_i сводится к изменению этих параметров. Вставка новой вершины v с координатами (x_v, y_v, z_v) и параметрическими координатами (p_v, s_v) на поверхность, параметризованную функциями χ, γ, ζ, посередине между двумя вершинами v_1, v_2 с параметрами (p_1, s_1), (p_2, s_2) может быть реализована несколькими способами. Во-первых, можно построить образ середины отрезка в параметрическом пространстве:

$$p_v = (p_1 + p_2)/2, \quad s_v = (s_1 + s_2)/2. \tag{П.1.4}$$

Во-вторых, можно построить точку, равноудаленную от вершин v_1 и v_2, прообраз которой в параметрическом пространстве лежит на отрезке, соединяющем (p_1, s_1) и (p_2, s_2), т. е.

$$p_v = p_1\theta + p_2(1 - \theta), \quad s_v = s_1\theta + s_2(1 - \theta),$$

где $\theta \in (0, 1)$ является решением нелинейного уравнения

$$\begin{aligned}(x_1 - \chi(p_v(\theta), s_v(\theta)))^2 + (y_1 - \gamma(p_v(\theta), s_v(\theta)))^2 + \\ + (z_1 - \zeta(p_v(\theta), s_v(\theta)))^2 = (x_2 - \chi(p_v(\theta), s_v(\theta)))^2 + \\ + (y_2 - \gamma(p_v(\theta), s_v(\theta)))^2 + (z_2 - \zeta(p_v(\theta), s_v(\theta)))^2.\end{aligned} \tag{П.1.5}$$

Это уравнение можно решить методом бисекции в интервале $(0, 1)$ или методом Ньютона. Отметим, что метод бисекции опирается исключительно на процедуру оценки функции одного переменного в некоторых точках, поэтому способ вычисления функций $\chi(p_v(\theta), s_v(\theta))$, $\gamma(p_v(\theta), s_v(\theta))$ и $\zeta(p_v(\theta), s_v(\theta))$ для этого подхода не важен.

Другим способом поиска пары (p_v, s_v) является составление и решение системы двух нелинейных уравнений, выражающих равноудаленность v от вершин v_1 и v_2 и минимальность расстояния от $\mathbf{v}$ до $(\mathbf{v}_1 + \mathbf{v}_2)/2$. Помимо возможной неединственности решения такой системы, поиск решения может быть затруднительным (см. также п. 3.4.3).

П.1.3. Адаптация сеток к границам с неизвестной параметризацией

В тех случаях, когда параметризация границы области недоступна, построение сетки, адаптированной к границе, затруднено. При этом, как правило, граница области задана с помощью поверхностной триангуляции. Рассмотрим способы адаптации тетраэдральной сетки к границе такого типа.

Наиболее простой подход заключается в постановке новых узлов на текущую (в процессе адаптации) границу сеточной области. Как показывает простой двумерный пример, приведенный на рис. П.2, удаление граничных узлов, возможное при адаптации, может привести к существенной ошибке в аппроксимации исходной границы граничными ребрами сетки даже при том, что шаг адаптированной сетки меньше шага начальной сетки.

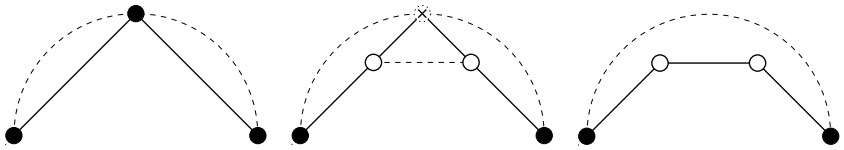


Рис. П.2. Пример адаптации сетки, ухудшающей аппроксимацию границы

Простейшее решение этой проблемы состоит в сохранении начальной поверхностной триангуляции за счет ее измельчения или сдвига узла только по исходной кусочно-линейной поверхости [43]. Такой подход, «замораживающий» исходную границу, естественно, сохраняет ошибку аппроксимации криволинейной границы исходным множеством граничных граней. При численном решении уравнений математической физики недостаточно высокое разрешение поверхности границы может привести к существенной ошибке и даже свести на нет усилия, затраченные на построение адаптивной сетки внутри области.

Другое решение проблемы — использовать результат адаптивного расчета в качестве обратной связи с САПР для построения новой поверхностной триангуляции. Подобный подход требует прямого вмешательства пользователя и может быть слишком сложным для некоторых приложений.

Третье решение проблемы основано на предположении, что если неизвестная поверхность границы достаточно гладкая (или кусочно-гладкая), то ее триангуляция неявно несет дополнительную информацию об этой поверхности. Ниже мы рассмотрим метод построения новой дискретной поверхности, для которой ошибка приближения гладкой границы существенно меньше, чем для кусочно-линейной триангуляции.

Точность аппроксимации поверхности можно существенно повысить за счет восстановления поверхности более высокого порядка на основе заданной кусочно-линейной поверхности. Используются несколько

методов локального восстановления такой поверхности, см. работы [53, 70, 71, 73] и ссылки в них. Мы опишем метод [43], который использует дискретную дифференциальную геометрию для вычисления кусочно-квадратичной непрерывной функции, приближающей восстанавливаемую поверхность. Непрерывность этой функции является отличительной чертой данного метода. Гессиан (матрица вторых производных) этой функции вычисляется на основе слабой формулировки, по аналогии с методом конечных элементов.

Рассмотрим гладкий кусок поверхности Γ с границей Θ. Пусть Γ_h — некоторая кусочно-линейная аппроксимация Γ с дискретной границей Θ_h. Предположим, что узлы Γ_h и Θ_h лежат на Γ и Θ соответственно, хотя данное предположение не является необходимым на практике. Кусочно-квадратичная экстраполяция $\widetilde{\Gamma}_h$ триангуляции Γ_h определяется как непрерывная поверхность, состоящая из локальных квадратичных экстраполяций $\widetilde{f}_t \in \widetilde{\Gamma}_h$ над треугольниками $f_t \in \Gamma_h$.

Пусть локальная экстраполяция $\widetilde{f}_t$ описывается квадратичной функцией $\varphi_{2,\,t}$. Ниже мы будем опускать индекс t, если это не приводит к путанице. Опишем функцию φ_2 в удобной для нас локальной системе координат $\boldsymbol{\eta} = (\eta_1, \eta_2)$, связанной с плоскостью треугольника f_t, используя ее гессиан $\mathbb{H}^{\varphi_2} = \{\mathbb{H}^{\varphi_2}_{km}\}^2_{k,\,m=1}$, $\mathbb{H}^{\varphi_2}_{km} \equiv \dfrac{\partial^2 \varphi_2}{\partial \eta_k \partial \eta_m}$:

$$\varphi_2(\boldsymbol{\eta}) = -\frac{1}{2} \sum_{i=1}^{3} (\mathbb{H}^{\varphi_2}(\boldsymbol{\eta} - \boldsymbol{\eta}_i), \boldsymbol{\eta} - \boldsymbol{\eta}_i)\, \lambda_i(\boldsymbol{\eta}).$$

Здесь $\boldsymbol{\eta}_i$ — вектор координат вершины треугольника v_i, $\lambda_i(\boldsymbol{\eta})$ — значение в $\boldsymbol{\eta}$ линейной функции, равной 1 в v_i и 0 в остальных вершинах треугольника f_t. Данная формула задания квадратичной функции является следствием многоточечной формулы Тейлора [39] для представления функции ошибки линейного интерполирования дважды дифференцируемой функции на треугольнике f_t.

Построение гессиана $\mathbb{H}^{\varphi_2}$ состоит из двух шагов. Сначала мы вычислим гессиан в узлах сетки Γ_h, а затем продолжим его внутрь треугольников $f_t \in \Gamma_h$ таким образом, что восстановленные на f_t функции $\varphi_{2,\,t}(\boldsymbol{\eta})$ формируют непрерывную поверхность $\widetilde{\Gamma}_h$.

Ш а г 1. Для каждого внутреннего узла v_i сетки Γ_h рассмотрим суперэлемент $\sigma(v_i)$ и определим плоскость, аппроксимирующую в смысле минимальных квадратов узлы этого суперэлемента и приближающую касательную к Γ в v_i плоскость. Пусть (ξ_1, ξ_2) обозначает локальную систему координат для этой плоскости. Определим суперэлемент $\widehat{\sigma}_i$ как ортогональную проекцию треугольников из σ_i на (ξ_1, ξ_2)-плоскость. На рис. П.3 изображены суперэлемент σ_i, его проекция $\widehat{\sigma}_i$, а также заштрихованы некоторый треугольник из σ_i и его проекция на плоскость (ξ_1, ξ_2). Используя замену переменных, определим непрерывную функцию $\varphi^i(\xi_1, \xi_2)$, локально представляющую поверхность Γ,

и непрерывную кусочно-линейную функцию $\varphi^i_h(\xi_1, \xi_2)$, локально представляющую Γ_h. Предположим, что обе функции однозначны над $\widehat{\sigma}_i$. Наконец, обозначим гессиан φ_i через $\mathbb{H}^{\varphi}$, а дискретный гессиан φ^h_i через $\mathbb{H}^h$.

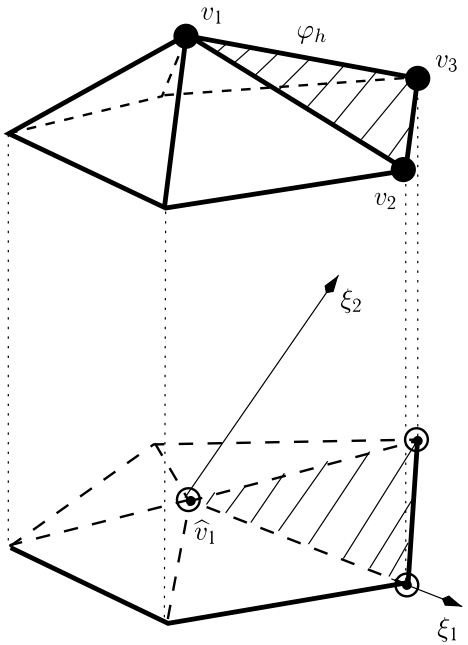


Рис. П.3. Локальная система координат для суперэлемента $\sigma(v_1)$

Собственные значения и векторы гессиана $\mathbb{H}^{\varphi}(v_i)$ связаны с главными кривизнами и главными направлениями поверхности Γ в точке v_i, поэтому его проекция $(\mathbb{H}^{\varphi}(v_i)\mathbf{e}, \mathbf{e})$ на любой единичный вектор $\mathbf{e}$ в касательной к Γ плоскости есть нормальная кривизна в направлении $\mathbf{e}$. Если компоненты дискретного гессиана $\mathbb{H}^h_{km}(v_i)$ в узле v_i приближают компоненты дифференциального гессиана $\mathbb{H}^{\varphi}_{km}(v_i)$, то величина $(\mathbb{H}^h\mathbf{e}, \mathbf{e})$ приближает нормальную кривизну поверхности Γ в направлении $\mathbf{e}$. В [51] предложен метод расчета кривизн дискретной поверхности Γ_h, сходящихся к кривизнам гладкой поверхности Γ в максимальной норме при стремлении h к нулю. Мы используем другой подход к оценке нормальных кривизн, простой и хорошо работающий на практике, несмотря на отсутствие теоретического обоснования. В этом подходе компоненты дискретного гессиана $\mathbb{H}^h$ в узле v_i определяются через слабую формулировку:

$$\int\limits_{\widehat{\sigma}_i} \mathbb{H}^h_{km}(v_i)\psi_h \, dS = -\int\limits_{\widehat{\sigma}_i} \frac{\partial \varphi^i_h}{\partial \xi_k} \frac{\partial \psi_h}{\partial \xi_m} \, dS, \qquad k,\, m = 1,\, 2, \qquad \text{(П.1.6)}$$

которая должна выполняться для любых непрерывных кусочно-линейных функций ψ_h, зануляющихся на $\partial\widehat{\sigma}_i$.

Ш а г 2. Продолжение гессиана внутрь треугольников основано на величинах α_{ij}, представляющих проекции гессиана на ребра $\mathbf{e}_{ij}$ треугольника f_t. Здесь и далее мы предполагаем, что вектор $\mathbf{e}_{ij}$ начинается в вершине $\mathbf{v}_i$ и заканчивается в $\mathbf{v}_j = \mathbf{v}_{i+1}$, где $\mathbf{v}_4 \equiv \mathbf{v}_1$. В локальной системе координат векторы $\mathbf{e}_{ij}$ задаются двумя координатами: $\mathbf{e}_{ij} = (e_1^{ij},\ e_2^{ij})$. Тогда по определению α_{ij} имеем:

$$\left(\begin{pmatrix} \mathbb{H}_{11}^{\varphi_2} & \mathbb{H}_{12}^{\varphi_2} \\ \mathbb{H}_{12}^{\varphi_2} & \mathbb{H}_{22}^{\varphi_2} \end{pmatrix} \begin{pmatrix} e_1^{ij} \\ e_2^{ij} \end{pmatrix}, \begin{pmatrix} e_1^{ij} \\ e_2^{ij} \end{pmatrix} \right) = \alpha_{ij},$$

что порождает систему из трех линейных уравнений с тремя неизвестными матричными элементами:

$$e_1^{ij} e_1^{ij} \mathbb{H}_{11}^{\varphi_2} + e_2^{ij} e_2^{ij} \mathbb{H}_{22}^{\varphi_2} + 2\, e_1^{ij} e_2^{ij} \mathbb{H}_{12}^{\varphi_2} = \alpha_{ij}, \qquad i = 1,\, 2,\, 3. \qquad \text{(П.1.7)}$$

Следующий результат показывает, что матрица системы невырождена, т. е. решение (П.1.7) существует и единственно.

Лемма П.1.1. *Матрица B коэффициентов системы* (П.1.7) *невырождена.*

Д о к а з а т е л ь с т в о. Прямые вычисления детерминанта матрицы B с учетом $\mathbf{e}_{12} + \mathbf{e}_{23} + \mathbf{e}_{31} = 0$ дают

$$|\det B| = 2\left|e_1^{12} e_2^{23} - e_1^{23} e_2^{13}\right|^3 = 16|f_t|^3 > 0,$$

где $|f_t|$ — площадь треугольника f_t. □

Поскольку функция $\varphi_{2,\,t}$ квадратична на ребре треугольника, она однозначно определяется значениями в вершинах ребра и соответствующей величиной α_{ij}. Поэтому для построения непрерывной поверхности $\widetilde{\Gamma}_h$ мы полагаем, что она содержит узлы исходной сетки Γ_h и что величина α_{ij} на ребре e_{ij} вычисляется одинаково для всех треугольников, содержащих это ребро. Определим α_{ij} как среднее значение двух узловых аппроксимаций гессиана:

$$\alpha_{ij} = \frac{1}{2}\left((\mathbb{H}^h(v_i)\,\mathbf{e}_{ij},\ \mathbf{e}_{ij}) + (\mathbb{H}^h(v_j)\,\mathbf{e}_{ij},\ \mathbf{e}_{ij})\right). \qquad \text{(П.1.8)}$$

У этого определения имеются два исключения. Если $v_i \in \Theta_h$ и $v_j \notin \Theta_h$, то $\alpha_{ij} = (\mathbb{H}^h(v_j)\mathbf{e}_{ij},\ \mathbf{e}_{ij})$. Если $v_i \in \Theta_h$ и $v_j \in \Theta_h$, то $\alpha_{ij} = 0$. Это означает, что узловая аппроксимация гессиана не восстанавливается на границах гладких кусков. Это также означает, что следы Γ_h и $\widetilde{\Gamma}_h$ на Θ_h совпадают.

Если поверхность Γ представима гладкой функцией, то предложенная кусочно-квадратичная экстраполяция $\widetilde{\Gamma}_h$ обеспечивает лучшую по сравнению с Γ_h ошибку аппроксимации Γ [43]. Другими словами,

$$\|\varphi - \varphi_{2,\,t}\|_{L^\infty(\widehat{f}_t)} \ll \|\varphi - \varphi_h\|_{L^\infty(\widehat{f}_t)}.$$

В качестве иллюстрации рассмотрим сферу Γ радиуса 0,18, гладкую поверхность без границы, $\Theta = \varnothing$. Рассмотрим последовательность квазиравномерных поверхностных триангуляций Γ_h с шагами $h = 0{,}1;\ 0{,}05;\ 0{,}025;\ 0{,}0125$, изображенных на рис. П.4.

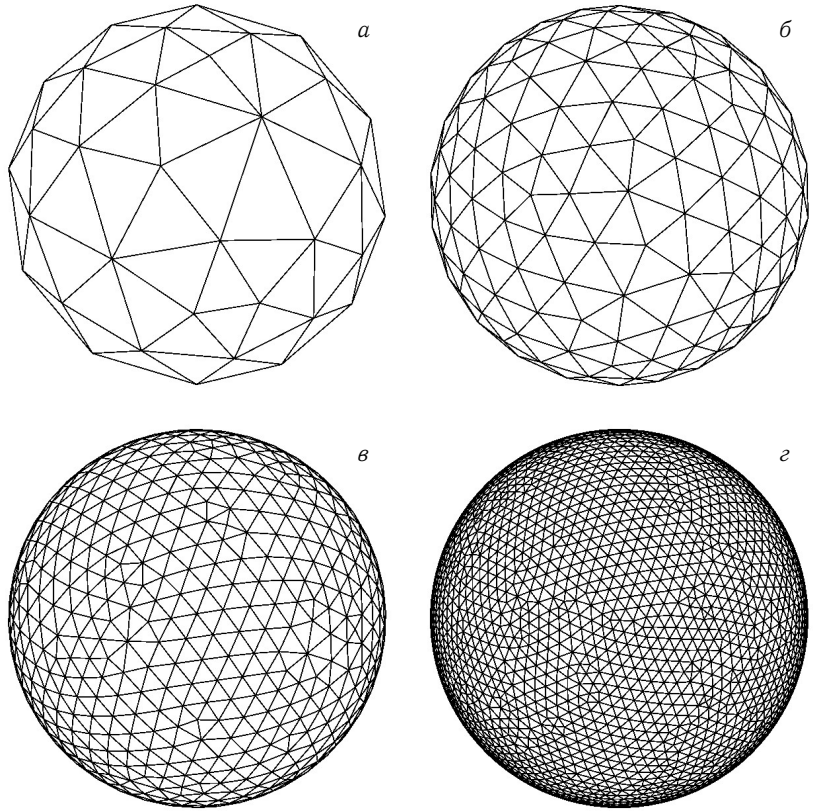


Рис. П.4. Квазиравномерные триангуляции сферы радиуса 0,18 с шагами $h =$ $= 0{,}1$ (*а*); 0,05 (*б*); 0,025 (*в*) и 0,0125 (*г*)

В табл. П.2 представлена максимальная норма ошибок приближения поверхности Γ кусочно-линейными поверхностями Γ_h и восстановленными кусочно-квадратичными поверхностями $\widetilde{\Gamma}_h$. Данные таблицы подтверждают, что кусочно-квадратичное восполнение позволяет существенно уменьшить ошибку представления гладкой поверхности, заданной поверхностной триангуляцией.

Отметим, что компоненты дискретного гессиана, восстановленные по формуле (П.1.6), могут не сходиться в максимальной норме к соответствующим компонентам гессиана $\mathbb{H}^\varphi$ при $h \to 0$. Это объясняет, почему мы не наблюдаем кубического уменьшения ошибки в правом

Таблица П.2
Ошибки представления сферы кусочно-линейными (Γ_h) и восстановленными кусочно-квадратичными ($\widetilde{\Gamma}_h$) поверхностями

h	$\|\Gamma - \Gamma_h\|_{L\infty}$	$\|\Gamma - \widetilde{\Gamma}_h\|_{L\infty}$
0,1	$1{,}3 \cdot 10^{-2}$	$1{,}4 \cdot 10^{-3}$
0,05	$3{,}5 \cdot 10^{-3}$	$2{,}3 \cdot 10^{-4}$
0,025	$8{,}9 \cdot 10^{-4}$	$5{,}7 \cdot 10^{-5}$
0,0125	$2{,}4 \cdot 10^{-4}$	$3{,}4 \cdot 10^{-5}$

столбце. Тем не менее использование величин (П.1.6) дает существенный выигрыш при использовании кусочно-квадратичного восполнения в практических расчетах. Использование более сложного метода [51] оценки кривизн дискретной поверхности Γ_h может еще больше уменьшить ошибку представления поверхности Γ восстановленной кусочно-квадратичной поверхностью $\widetilde{\Gamma}_h$.

§ П.2. Адаптация к решению посредством локального иерархического измельчения

Основополагающим принципом построения адаптивных расчетных сеток является принцип равнораспределения ошибки численного решения $u - u_h$ по ячейкам сетки Ω_h. Согласно этому принципу, сетка обеспечивает минимальную норму ошибки, т. е. минимизирует функционал ошибки, среди всех сеток с заданным количеством ячеек, если на всех ее ячейках норма ошибки одинакова. При этом конкретный вид нормы не имеет значения. Применение принципа равнораспределения ошибки базируется на монотонной зависимости нормы ошибки на каждой ячейке сетки от ее размера и на возможности модифицировать сетку с неравнораспределенной ошибкой таким образом, что норма ошибки уменьшится.

Во многих приложениях, однако, принцип равнораспределения ошибки не может быть реализован, поскольку ошибка численного решения не может быть вычислена из-за недоступности точного решения. В этих случаях применяется принцип равнораспределения *апостериорной оценки ошибки* η_Δ, которую можно вычислить без точного решения. Нормы ошибок численного решения на сетке с равнораспределенной ошибкой и на сетке с равнораспределенной апостериорной оценкой ошибки будут сопоставимы в том случае, если оценка локальной ошибки хорошо приближает саму локальную ошибку:

$$c\eta_\Delta \leqslant \|u - u_h\|_{*,\Delta} \leqslant C\eta_\Delta,$$

где c, C — константы порядка единицы, а $\|\cdot\|_{*,\Delta}$ — некоторая норма ошибки на ячейке Δ. Различные подходы к вычислению апостериорных оценок ошибки будут рассмотрены в конце параграфа.

Адаптивное построение расчетной сетки посредством локального иерархического измельчения может быть реализовано на основе доступной апостериорной оценки ошибки η_Δ и заданного порога δ для максимальной ошибки. Примером такой реализации служит алгоритм 43.

Отметим следующие особенности алгоритма. Данный алгоритм адаптации не обеспечивает строгого равнораспределения ошибки, а лишь не измельчает сетку там, где локальная оценка ниже заданного порога. Начальная сетка должна быть достаточно груба, чтобы на всех ее ячейках выполнялось $\eta_\Delta > \delta$. В противном случае на тех ячейках, где $\eta_\Delta \leqslant \delta$, принцип равнораспределения оценки ошибки будет нарушен.

Алгоритм 43. Адаптивное построение сетки путем локального иерархического измельчения

1: Построить начальную сетку Ω_h. Положить $\eta_{\max} = \infty$
2: **while** пока $\eta_{\max} > \delta$ **do**
3: Найти численное решение u_h на сетке Ω_h. Положить $\mathcal{M} = \varnothing$
4: **loop** по всем ячейкам $\Delta \in \Omega_h$
5: Вычислить локальную апостериорную оценку ошибки η_Δ
6: Добавить ячейку в множество $\mathcal{M}$, если $\eta_\Delta > \delta$
7: **end loop**
8: Вычислить $\eta_{\max} = \max\limits_{\Delta \subset \Omega_h} \eta_\Delta$
9: Построить новую сетку Ω_h путем бисекции ячеек из множества $\mathcal{M}$ (см. алгоритмы 21 и 24)
10: **end while**

В случае области с криволинейной границей иерархическое измельчение в окрестности границы, строго говоря, не имеет смысла, поскольку новые узлы могут не лежать на границе области. В таких случаях вновь появившиеся граничные узлы должны быть спроецированы на границу (см. § П.1) и иерархическое вложение сеток будет нарушено. Тем не менее, с топологической точки зрения, иерархическая вложенность сеток сохраняется и метод многоуровневого измельчения сетки может быть модифицирован для этого случая.

Если в дополнение к процедуре локального измельчения сетки имеется процедура локального огрубления сетки, то адаптивный алгоритм можно усилить. Для этого нужно потребовать, чтобы начальная сетка допускала многоуровневое огрубление (см. § 4.4). Алгоритм 44 порождает сетку, в которой оценка ошибки на всех ячейках лежит в интервале $[\kappa\delta, \delta]$, $0 < \kappa < 1$. Параметр κ выбирается таким образом,

чтобы локальное измельчение (огрубление) ячейки уменьшало (увеличивало) ошибку не более, чем в κ^{-1} раз.

Алгоритм 44. Адаптивное построение сетки путем локального иерархического измельчения и огрубления

1: Построить начальную сетку Ω_h, допускающую многоуровневое огрубление. Положить $\eta_{\max} = \infty$, $\eta_{\min} = 0$
2: **while** $\eta_{\max} > \delta$ и $\eta_{\min} < \kappa\delta$ **do**
3: Найти численное решение u_h на сетке Ω_h. Положить $\mathcal{M}_1 = \varnothing$, $\mathcal{M}_2 = \varnothing$
4: **loop** по всем ячейкам $\Delta \in \Omega_h$
5: Вычислить локальную апостериорную оценку ошибки η_Δ
6: Добавить ячейку во множество $\mathcal{M}_1$, если $\eta_\Delta > \delta$
7: Добавить ячейку во множество $\mathcal{M}_2$, если $\eta_\Delta < \kappa\delta$
8: **end loop**
9: Вычислить $\eta_{\max} = \max\limits_{\Delta\subset\Omega_h} \eta_\Delta$ и $\eta_{\min} = \min\limits_{\Delta\subset\Omega_h} \eta_\Delta$
10: Построить новую сетку Ω_h путем огрубления ячеек из множества $\mathcal{M}_2$ и измельчения ячеек из множества $\mathcal{M}_1$
11: **end while**

Приведенные выше адаптивные алгоритмы применимы главным образом для приближенного решения стационарных уравнений математической физики. Для нестационарных задач нужно учитывать следующие обстоятельства.

1) Помимо адаптивного контроля над пространственным шагом сетки, необходимо адаптивное управление временны́м шагом. Дискретизация по времени должна обеспечивать ошибку, сопоставимую с ошибкой дискретизации по пространству.

2) Адаптивное перестроение сетки на основе апостериорной оценки ошибки можно делать не на каждом шаге по времени, а один раз за несколько временны́х шагов [46].

3) Огрубление ячеек является неотъемлемой частью адаптации сетки. В противном случае рост количества ячеек сетки с течением времени может замедлить или даже заблокировать расчет.

В некоторых приложениях иерархическая адаптация сетки может сопровождаться сдвигом узлов для достижения лучшей аппроксимации разрывов решения, таких, как ударные волны и скачки уплотнения в газовой динамике, разрывы материала и появление трещин в механике твердого тела, а также геометрические особенности модели.

В качестве примера локального иерархического измельчения рассмотрим приближенное решение уравнения диффузии

$$-\operatorname{div} \rho\nabla u = f \qquad \text{(П.2.1)}$$

с однородным условием Дирихле методом конечных элементов на адаптивной треугольной сетке [57]. Уравнение рассматривается в единичном квадрате, разбитом на три подобласти Ω_1, Ω_2 и Ω_3, причем Ω_2 — внутреннее кольцо (см. рис. П.5). Коэффициент диффузии $\rho(x) = \rho_i$ в подобласти Ω_i определяется как $\rho_1 = \rho_3 = 1$, $\rho_2 = 10\,000$, а правая часть $f(x) = 1$. Механическая аналогия рассматриваемой краевой задачи — деформация под нагрузкой композитного материала с фиксированными границами.

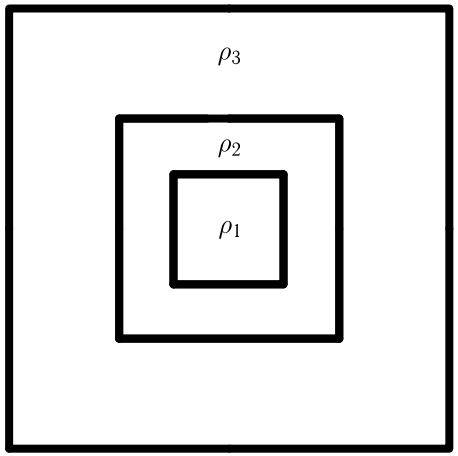


Рис. П.5. Разбиение на три подобласти

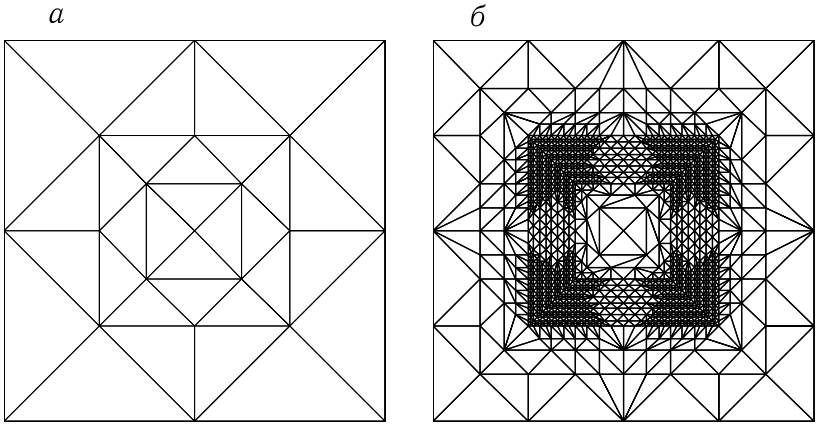


Рис. П.6. Начальная сетка (*а*) и измельченная иерархическая сетка (*б*)

Начальная сетка представлена на рис. П.6, *а*. Для построения адаптивной сетки, изображенной на рис. П.6, *б*, применялся алгоритм 43, в котором в качестве величины η_Δ использовалась апостериорная

оценка ошибки по невязке, изложенная ниже, а локальное измельчение обеспечивалось методом красно-зеленого разбиения [30], альтернативным методу бисекции. Ошибка пропорциональна коэффициенту диффузии, что приводит к сильному измельчению сетки в подобласти Ω_2.

Опишем кратко несколько подходов к формированию апостериорных оценок ошибки для кусочно-линейного конечноэлементного решения u_h двумерного эллиптического уравнения на регулярных триангуляциях. Для простоты мы рассмотрим уравнение Пуассона с краевым условием Дирихле:

$$\begin{cases} -\Delta u &= f & \text{в } \Omega, \\ u &= 0 & \text{на } \partial\Omega. \end{cases} \qquad \text{(П.2.2)}$$

Более подробно о методах апостериорной оценки ошибки и их применении для более широкого класса задач можно прочитать в [83].

П.2.1. Апостериорная оценка ошибки по невязке

Апостериорная оценка ошибки по невязке основана на тождестве

$$\int\limits_\Omega \nabla(u-u_h)\cdot\nabla v\,dV = \int\limits_\Omega fv\,dV - \int\limits_\Omega \nabla u_h\cdot\nabla v\,dV,$$

справедливом для обобщенного решения $u\in H_0^1(\Omega)$ задачи (П.2.2) и любой функции $v\in H_0^1(\Omega)$, где $H_0^1(\Omega)$ — пространство Соболева функций с нулевым следом на $\partial\Omega$ и интегрируемым квадратом обобщенной производной [39]. Поскольку

$$\|\nabla v\| = \sup_{w\in H_0^1(\Omega),\|\nabla w\|=1} \int\limits_\Omega \nabla v\cdot\nabla w\,dV,$$

градиентная ошибка $\|\nabla(u-u_h)\|$ может быть оценена через норму невязки конечноэлементного решения в сопряженном пространстве с нормой

$$\sup_{w\in H_0^1(\Omega),\|\nabla w\|=1} \Big\{\int\limits_\Omega fw\,dV - \int\limits_\Omega \nabla u_h\cdot\nabla w\,dV\Big\}.$$

Интегрируя по частям и используя факт, что решение линейно на каждом треугольнике, можно показать, что

$$\int\limits_\Omega fw\,dV - \int\limits_\Omega \nabla u_h\cdot\nabla w\,dV = \sum_{\Delta\in\Omega_h}\int\limits_\Delta fw\,dV - \sum_{e\in\Omega_h,\,e\notin\partial\Omega}\int\limits_e [\mathbf{n}_e\cdot\nabla u_h]_e w\,dS,$$

где $[\mathbf{n}_e\cdot\nabla u_h]_e$ обозначает скачок нормальной составляющей потока функции u_h через внутреннее ребро сетки e, разделяющее два треугольника Δ_+, Δ_-:

$$[\mathbf{n}_e\cdot\nabla u_h]_e = \mathbf{n}_e\cdot\nabla u_h|_{\Delta_+} - \mathbf{n}_e\cdot\nabla u_h|_{\Delta_-}.$$

С учетом этих соображений локальная апостериорная оценка ошибки по невязке может быть записана в виде

$$\eta_{R,\Delta} = \left\{(\operatorname{diam}(\Delta))^2\|f\|^2_{L^2(\Delta)} + \frac{1}{2}\sum_{e\in\Delta,\,e\notin\partial\Omega}|e|\,\|[\mathbf{n}_e\cdot\nabla u_h]_e\|^2_{L^2(e)}\right\}^{1/2}. \qquad \text{(П.2.3)}$$

Множитель $\frac{1}{2}$ во втором слагаемом учитывает, что при суммировании величин $\eta^2_{R,\Delta}$ по всем треугольникам интеграл по каждому внутреннему ребру входит дважды. На практике вычисление L^2-нормы правой части на каждом треугольнике осуществляется с помощью квадратурных формул. Более того, возможна замена интегрируемой функции f на ее среднее значение

$$f_\Delta = \frac{1}{|\Delta|}\int\limits_\Delta f\,dV,$$

где $|\Delta|$ обозначает площадь треугольника Δ.

П.2.2. Апостериорная оценка ошибки, основанная на решении локальных подзадач

Главным принципом всех апостериорных оценок ошибки, основанных на решении локальных подзадач, является формирование большого числа задач с малым числом неизвестных. При решении вспомогательных задач необходимо использовать конечноэлементные пространства более высокого порядка, чем при решении исходной задачи (П.2.2).

Для каждого треугольника Δ сформируем подобласть ω_Δ, содержащую Δ и треугольники, соседствующие с Δ по ребру. В каждой подобласти ω_Δ рассмотрим приближенное решение U_h локальной задачи

$$\begin{cases} -\Delta U &= f & \text{в } \omega_\Delta, \\ U &= u_h & \text{на } \partial\omega_\Delta \end{cases}$$

методом конечных элементов, в котором на каждом треугольнике $\Delta(v_1, v_2, v_3)$ пространство многочленов первого порядка обогащается функцией-пузырем $b_\Delta = \lambda_{\Delta,1}\lambda_{\Delta,2}\lambda_{\Delta,3}$. Здесь и далее $\lambda_{\Delta,i}$ обозначает линейную конечноэлементную базисную функцию на Δ, т. е. $\lambda_{\Delta,i}(v_j) = \delta_{ij}$. Отметим, что кусочно-кубическая функция U_h кусочно-линейна на $\partial\omega_\Delta$.

Апостериорная оценка ошибки, основанная на решении локальных подзадач, вычисляется по формуле

$$\eta_{L,\Delta} = \|\nabla(U_h - u_h)\|_{L^2(\omega_\Delta)}. \qquad \text{(П.2.4)}$$

Возможны и другие варианты формирования локальных подзадач. Например, вместо суперэлемента ω_Δ можно рассмотреть суперэлемент $\sigma(v)$, образованный треугольниками с общей вершиной v. Помимо локальной задачи Дирихле, можно также рассмотреть локальную задачу Неймана на внутреннем треугольнике и смешанную краевую задачу на приграничном треугольнике:

$$\begin{cases} -\Delta u &= f & \text{в } \Delta, \\ u &= u_h & \text{на } \partial\Delta \cap \partial\Omega, \\ \dfrac{\partial u}{\partial n} &= -\dfrac{1}{2}\,[\mathbf{n}_e \cdot \nabla u_h]_e & \text{на } \partial\Delta \setminus \partial\Omega. \end{cases}$$

Данная задача решается методом конечных элементов с обогащенным пространством, описанным выше.

П.2.3. Иерархическая апостериорная оценка ошибки

Иерархическая апостериорная оценка ошибки [41] для решения задачи (П.2.2) основана на обогащении исходного пространства кусочно-линейных конечных элементов. Пусть λ_i обозначает кусочно-линейную базисную функцию, связанную с узлом v_i. Для каждого ребра e_{ij} определим функцию-пузырь $b_{ij} = 4\lambda_i\lambda_j$. Обогащенное пространство строится путем добавления к исходному базису λ_i реберных функций-пузырей b_{ij}.

Пусть U_h обозначает конечноэлементное решение задачи (П.2.2) в обогащенном пространстве. Основное предположение метода заключается в том, что разница между кусочно-линейным решением u_h и кусочно-квадратичным решением U_h является хорошим приближением к ошибке $u - u_h$:

$$\|\nabla(u_h - U_h)\|_{L^2(\Omega)} \approx \|\nabla(u_h - u)\|_{L^2(\Omega)}. \qquad (\text{П.2.5})$$

Для того, чтобы оценить $\|\nabla(u_h - U_h)\|_{L^2(\Omega)}$, разложим u_h и U_h по базисным функциям:

$$u_h = \sum_{v_i} u_{L,\,i}\lambda_i, \quad U_h = \sum_{v_i} U_{L,\,i}\lambda_i + \sum_{e_{ij}} U_{Q,\,ij} b_{ij}.$$

Коэффициенты этого разложения удовлетворяют системам линейных уравнений

$$A_{LL}u_L = F_L \qquad (\text{П.2.6})$$

и

$$\begin{pmatrix} A_{LL} & A_{LQ} \\ A_{QL} & A_{QQ} \end{pmatrix} \begin{pmatrix} U_L \\ U_Q \end{pmatrix} = \begin{pmatrix} F_L \\ F_Q \end{pmatrix}.$$

Если разложить функцию $u_h - U_h$ по базисным функциям:

$$u_h - U_h = \sum_{v_i} D_{L,\,i}\lambda_i + \sum_{e_{ij}} D_{Q,\,ij} b_{ij},$$

то коэффициенты этого разложения удовлетворяют системе линейных уравнений

$$\begin{pmatrix} A_{LL} & A_{LQ} \\ A_{QL} & A_{QQ} \end{pmatrix} \begin{pmatrix} D_L \\ D_Q \end{pmatrix} = \begin{pmatrix} F_L - A_{LL}u_L \\ F_Q - A_{QL}u_L \end{pmatrix}, \qquad (\text{П.2.7})$$

в которой векторы F_L, F_Q и u_L известны. Однако решение системы (П.2.7) является слишком затратной процедурой, требующей большей арифметической работы, чем решение исходной системы (П.2.6). Для построения легко вычисляемой апостериорной оценки заменим систему (П.2.7) на систему упрощенного вида

$$\begin{pmatrix} A_{LL} & O \\ O & A_{QQ} \end{pmatrix} \begin{pmatrix} \widetilde{D}_L \\ \widetilde{D}_Q \end{pmatrix} = \begin{pmatrix} F_L - A_{LL}u_L \\ F_Q - A_{QL}u_L \end{pmatrix},$$

из которой следует, что $\widetilde{D}_L = 0$, а вектор $\widetilde{D}_Q$ удовлетворяет системе

$$A_{QQ}\,\widetilde{D}_Q = F_Q - A_{QL}u_L. \qquad (\text{П.2.8})$$

Матрица A_{QQ} хорошо обусловлена на сетках с регулярными ячейками, поэтому систему (П.2.8) можно эффективно решить, например, методом сопряженных градиентов. Можно также показать [41], что

$$\left\| \sum_{e_{ij}} \widetilde{D}_{Q,\,ij} \nabla b_{ij} \right\|_{L^2(\Delta)} \simeq \|\nabla(u_h - U_h)\|_{L^2(\Delta)}. \qquad (\text{П.2.9})$$

Поэтому величина

$$\eta_{H,\,\Delta} = \frac{1}{2} \left\| \sum_{e_{ij} \in \Delta} \widetilde{D}_{Q,\,ij} \nabla b_{ij} \right\|_{L^2(\Delta)}$$

является легко вычисляемой иерархической апостериорной оценкой локальной градиентной ошибки.

П.2.4. Апостериорная оценка ошибки по осредненному градиенту

Апостериорная оценка ошибки по осредненному градиенту основана на легко вычисляемом приближении Gu_h функции ∇u, для которого

$$\|\nabla u - Gu_h\| \leqslant \beta \|\nabla(u - u_h)\|, \quad 0 \leqslant \beta < 1.$$

Неравенство треугольника приводит к двусторонней оценке ошибки:

$$\frac{1}{1+\beta}\,\|Gu_h - \nabla u_h\| \leqslant \|\nabla(u - u_h)\| \leqslant \frac{1}{1-\beta}\,\|Gu_h - \nabla u_h\|.$$

Приближение Gu_h определяется как дискретная L^2-проекция кусочно-постоянной вектор-функции ∇u_h на пространство непрерывных кусочно-линейных вектор-функций. Дискретность проекции

выражается в специальном выборе скалярного произведения на основе замены интеграла по треугольнику квадратурной формулой прямоугольников:

$$\int\limits_{\Delta(v_1,v_2,v_3)} \varphi\, dV \approx \frac{|\Delta|}{3}\,(\varphi(v_1)+\varphi(v_2)+\varphi(v_3)).$$

Такая замена приводит к локальному вычислению функции Gu_h в любом узле сетки v:

$$Gu_h(v) = \sum_{\Delta\in\sigma(v)} \frac{|\Delta|}{|\sigma(v)|}\,\nabla u_{h|\Delta}.$$

Таким образом, апостериорная оценка ошибки по осредненному градиенту определяется так:

$$\eta_{Z,\Delta} = \|Gu_h - \nabla u_h\|_{L^2(\Delta)}.$$

В определенном смысле все рассмотренные апостериорные оценки ошибки эквивалентны, поскольку обеспечивают верхнюю и нижнюю оценки ошибки конечноэлементного решения [83].

§ П.3. Адаптация к сеточному решению посредством локальных модификаций

Как было показано в § 6.2, управление свойствами сетки с помощью тензорной метрики является наиболее гибким средством контроля за свойствами сеточных ячеек. Основная причина, выделяющая управление метрикой из других способов контроля, заключается в возможности построения анизотропных сеток с сильно вытянутыми ячейками. Методы продвигаемого фронта и триангуляции Делоне становятся значительно менее надежными при построении анизотропных сеток. В § 6.2 рассматривается метод построения анизотропных сеток, использующий в качестве контроля тензорную метрику $\mathbb{M}(\mathbf{x})$. Метод сводится к построению сетки, квазиравномерной в заданной метрике $\mathbb{M}(\mathbf{x})$, посредством последовательности локальных модификаций текущей сетки. Мы будем называть такую сетку $\mathbb{M}$-*квазиравномерной сеткой*.

При разработке методов построения треугольных или тетраэдральных сеток, которые адаптированы к сеточному решению, основной проблемой является построение тензорной метрики на основе текущего сеточного решения. Выбор метрики сопряжен с приближенной минимизацией некоторой нормы ошибки сеточного решения на множестве конформных сеток Ω_h с ограниченным числом ячеек $N(\Omega_h)$:

$$\min_{N(\Omega_h)\leqslant N_\star} \|u-u_h\|_*.$$

Предположим, что метод построения тензорной метрики задан, тогда алгоритм построения адаптивной сетки с заданным количеством ячеек выглядит очень просто.

Алгоритм 45. Построение адаптивной сетки с $N_\star$ элементами

1: Построить начальную сетку Ω_h, найти сеточное решение u_h и вычислить тензорную метрику $\mathbb{M}$
2: **while** сетка Ω_h не является $\mathbb{M}$-квазиравномерной **do**
3: Построить $\mathbb{M}$-квазиравномерную сетку Ω_h с $N_\star$ элементами
4: Вычислить сеточное решение u_h
5: Вычислить новую тензорную метрику $\mathbb{M}$ по u_h
6: **end while**

Ниже мы рассмотрим несколько методов построения тензорной метрики на основе сеточного решения и проиллюстрируем результирующие адаптивные алгоритмы численными результатами.

П.3.1. Тензорная метрика на основе восстановления гессиана

Первый способ построения тензорной метрики использует методы восстановления гессиана сеточной функции u_h. Напомним, что гессианом $\mathbb{H}$ скалярной дважды дифференцируемой функции u называется матрица вторых частных производных этой функции с элементами $\mathbb{H}_{km}(u) = \dfrac{\partial^2 u}{\partial x_k \partial x_m}$, где $x_1 = x$, $x_2 = y$, $x_3 = z$. Поскольку в большинстве приложений сеточные решения не являются дважды дифференцируемыми в классическом смысле функциями, классические частные производные заменяются на обобщенные. Мы приведем два способа вычисления непрерывного кусочно-линейного сеточного гессиана $\mathbb{H}_h$ от непрерывной кусочно-линейной функции u_h. Для этого достаточно определить значения компонент $\mathbb{H}_h$ в узлах сетки.

Пусть d обозначает размерность пространства, $d = 2, 3$. Наиболее распространенный способ вычисления значений гессиана сеточной функции u_h в узлах сетки, основанный на идее конечноэлементной дискретизации эллиптических уравнений второго порядка, был кратко описан в § П.1. Рассмотрим суперэлемент $\sigma(v_i)$ как объединение всех симплексов, содержащих v_i. Компоненты сеточного гессиана $\mathbb{H}_{h,\,km}(v_i)$, $k, m = 1, \ldots, d$, восстанавливаются во внутреннем узле v_i сетки Ω_h следующим образом:

$$\int\limits_{\sigma(v_i)} \mathbb{H}_{h,\,km}(v_i)\,\psi_h\,dx = -\int\limits_{\sigma(v_i)} \frac{\partial u_h}{\partial x_k}\,\frac{\partial \psi_h}{\partial x_m}\,dx \qquad (\text{П.3.1})$$

для любой непрерывной кусочно-линейной конечноэлементной функции ψ_h, зануляющейся на границе суперэлемента $\sigma(v_i)$. В граничном

узле сетки гессиан $\mathbb{H}_h(v)$ определяется как выпуклая линейная комбинация значений гессиана в ближайших внутренних узлах:

$$\mathbb{H}_h(v) = \sum_{v_i \in \sigma(v), v_i \neq v} \alpha_i \, \mathbb{H}_h(v_i), \qquad \sum_{v_i \in \sigma(v), v_i \neq v} \alpha_i = 1.$$

В выборе весов мы следуем работе [23], где вес определяется как мера пересечения двух суперэлементов:

$$\alpha_i = |\sigma(v) \cap \sigma(v_i)| \left(\sum_{v_i \in \sigma(v), v_i \neq v} |\sigma(v) \cap \sigma(v_i)| \right)^{-1}.$$

Сеточный гессиан $\mathbb{H}_h(v_i)$ является симметричной матрицей, которая может быть неопределенной. Поэтому для построения метрики (положительно определенной матрицы) используется спектральный модуль $|\mathbb{H}_h(v_i)|$ матрицы $\mathbb{H}_h(v_i)$. Рассмотрим спектральное разложение этой матрицы:

$$\mathbb{H}_h(v_i) = \mathbb{W} \, \Lambda \, \mathbb{W}^{\mathrm{T}},$$

где $\mathbb{W}$ — ортонормальная матрица собственных векторов $\mathbb{H}_h(v_i)$, Λ — диагональная матрица собственных значений, упорядоченных по неубыванию их абсолютного значения:

$$\Lambda = \operatorname{diag}\{\lambda_i\}, \qquad |\lambda_1| \leqslant \ldots \leqslant |\lambda_d|.$$

Определим следующую тензорную метрику:

$$\mathbb{M}_h(v_i) = |\mathbb{H}_h(v_i)| = \mathbb{W} \, |\Lambda| \, \mathbb{W}^{\mathrm{T}}, \tag{П.3.2}$$

где $|\Lambda|$ — диагональная матрица абсолютных величин собственных значений.

В случае вырожденного гессиана $\mathbb{H}_h$ метрика формируется по формуле (П.3.2) с использованием матрицы возмущенных собственных значений

$$|\Lambda| := \operatorname{diag}\left\{\max\{|\lambda_1|; \varepsilon\}, \ \ldots, \ \max\{|\lambda_3|; \varepsilon\}\right\}.$$

Основным преимуществом вышеизложенного метода построения тензорной метрики по сеточной функции u_h является его независимость от задачи, решением которой является эта функция. Недостатком такого подхода является возможность большой ошибки при оценке сеточного гессиана с помощью формулы (П.3.1), что ярко проявляется в окрестности особенностей решения. Этим вызвана невозможность провести теоретический анализ адаптивного алгоритма, хотя алгоритм хорошо зарекомендовал себя на практике [5, 9, 66].

Математическим обоснованием адаптивного построения $\mathbb{M}_h$-квазиравномерной сетки с метрикой (П.3.1)–(П.3.2) является следующий результат для задачи оптимального интерполирования [5, 23]. Для непрерывной функции u определим оператор интерполирования

$I_{\Omega_h}(u)$, который использует значение функции в узлах сетки и возвращает непрерывную кусочно-линейную функцию.

Теорема П.3.1. *Пусть $u \in C^2(\overline{\Omega})$, $\det \mathbb{H} \neq 0$, Ω_h — $|\mathbb{H}|$-квазиравномерная сетка с $N_\star$ элементами. Далее, пусть $\Delta \in \Omega_h$ — элемент, на котором достигается максимум ошибки кусочно-линейного интерполирования, и $\mathbb{H}_\star = \mathbb{H}(x_\Delta)$, где $x_\Delta = \arg \max\limits_{x \in \Delta} |\det \mathbb{H}(x)|$. Наконец, пусть $\mathbb{H}_h$ — согласованная аппроксимация гессиана $\mathbb{H}$ на элементе Δ, для которой*

$$\|\mathbb{H} - \mathbb{H}_h\|_{L^\infty(\Delta)} < \frac{q_\star}{2} \, |\lambda_1(\mathbb{H}_\star)|, \qquad q_\star > 0,$$

где $\lambda_1(\mathbb{H}_\star)$ обозначает ближайшее к нулю собственное значение $\mathbb{H}_\star$. Тогда верна оценка ошибки

$$\|u - I_{\Omega_h}(u)\|_{L^\infty(\Omega)} \leqslant C(q_\star, \, |\mathbb{H}|, \, \Omega) N_\star^{-2/d}. \tag{П.3.3}$$

Отметим, что требование согласованной аппроксимации всегда выполняется для фиксированной функции u при достаточно больших значениях $N_\star$.

Можно также показать [5, 23], что для функции $u \in C^2(\overline{\Omega})$ с невырожденным гессианом при достаточно больших $N_\star$ верна оценка

$$C N_\star^{-2/d} \leqslant \min_{\widetilde{\Omega}_h : N(\widetilde{\Omega}_h) \leqslant N_\star} \|u - I_{\widetilde{\Omega}_h}(u)\|_{L^\infty(\Omega)}.$$

Таким образом, при разумных ограничениях $|\mathbb{H}|$-квазиравномерная сетка с $N_\star$ элементами обеспечивает асимптотически оптимальную скорость убывания интерполяционной ошибки в максимальной норме, т. е. является квазиоптимальной. Оценки ошибки в максимальной норме можно обобщить [6] на L^p-норму [1)], $p > 0$. При этом метрику $\mathbb{M} = |\mathbb{H}|$ необходимо заменить следующей метрикой:

$$\mathbb{M}_p = (\det |\mathbb{H}|)^{-1/(2p+d)} |\mathbb{H}|. \tag{П.3.4}$$

Пусть $\widetilde{\Omega}_h$ — $\mathbb{M}_p$-квазиравномерная сетка с $N_\star$ симплексами. Тогда асимптотическая скорость убывания ошибки интерполирования в L^p-норме будет той же, что и для максимальной нормы:

$$\|u - I_{\widetilde{\Omega}_h}(u)\|_{L^p(\Omega)} \leqslant \widehat{C} \, N_\star^{-2/d},$$

с константой $\widehat{C}$, зависящей от $q_\star$, $|\mathbb{H}|$, Ω, но не зависящей от p и $N_\star$.

Вышеизложенный анализ имеет непосредственное подтверждение численным экспериментом. Рассмотрим задачу оптимизации ошибки кусочно-линейной интерполяции функции

$$u(x, y) = yx^2 + y^3 + \operatorname{th}(6(\sin(5y) - 2x)) \tag{П.3.5}$$

[1)] При $0 < p < 1$ норма становится квазинормой.

на сетках с заданным числом ячеек. Для минимизации ошибки в L^∞- и L^1-нормах воспользуемся алгоритмом 45, в котором метрика восстанавливается по формулам (П.3.1), (П.3.2) и (П.3.4).

Таблица П.3 показывает, что измеренные нормы ошибок обратно пропорциональны числу треугольников $N_\star$ в сетке. На рис. П.7 показаны изолинии функции u и адаптивные сетки при $p = 1$ и $p = \infty$. Как видно из рисунка, большее значение p приводит к более интенсивному измельчению сетки вокруг особенности функции.

Таблица П.3
Ошибки интерполяции функции (П.3.5) на квазиоптимальных сетках в нормах L^1 и L^∞

$N_\star$	$\|u - u_h\|_{L^1(\Omega)}$	$\|u - u_h\|_{L^\infty(\Omega)}$
1000	$2{,}4 \cdot 10^{-2}$	$4{,}4 \cdot 10^{-2}$
2000	$1{,}2 \cdot 10^{-2}$	$2{,}2 \cdot 10^{-2}$
4000	$5{,}6 \cdot 10^{-3}$	$9{,}0 \cdot 10^{-3}$
8000	$2{,}8 \cdot 10^{-3}$	$5{,}0 \cdot 10^{-3}$

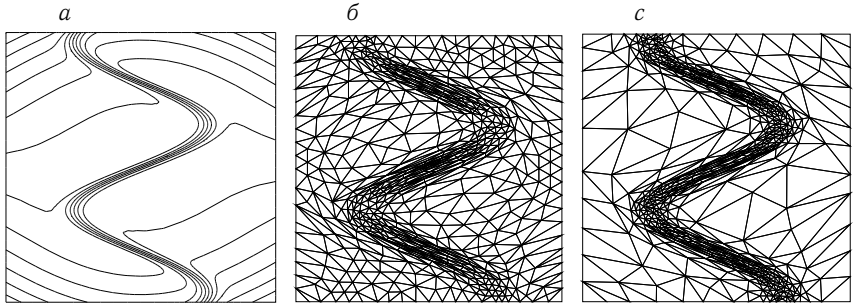


Рис. П.7. Изолинии интерполируемой функции (*а*) и квазиоптимальные сетки с приблизительно 1000 треугольников, минимизирующие ошибку интерполяции в нормах L^1 (*б*) и L^∞ (*в*)

Теоретический анализ асимптотических свойств $|\mathbb{H}_h|$-квазиравномерных сеток представлен только для задачи кусочно-линейного интерполирования. Отметим, что в алгоритме 45 можно использовать любое сеточное решение u_h на сетке Ω_h, например, конечноэлементное решение краевой задачи. Несмотря на отсутствие обоснования как сходимости адаптивных итераций, так и асимптотических свойств построенной таким образом адаптивной сетки, численные эксперименты показывают применимость такого расширения даже для задач с особенностями.

Рассмотрим, к примеру, классическую краевую задачу о трещине с точным решением $u(r, \theta) = r^{1/4} \sin(\theta/4)$ в полярных координатах

(r, θ) (см. рис. П.8). Задача определена в единичном круге Ω с центром в начале координат, без разреза S, задаваемого условиями $x > 0$ и $y = 0$:

$$\begin{cases} \Delta u = 0 & \text{в } \Omega \setminus S, \\ u = \sin\dfrac{\theta}{4} & \text{на } \partial\Omega \setminus S, \\ u = 0 & \text{на } S^+, \\ \dfrac{\partial u}{\partial n} = 0 & \text{на } S^-, \end{cases} \qquad \text{(П.3.6)}$$

где S^+ и S^- обозначают стороны разреза S, обращенные к полуплоскостям $x > 0$ и $y < 0$ соответственно.

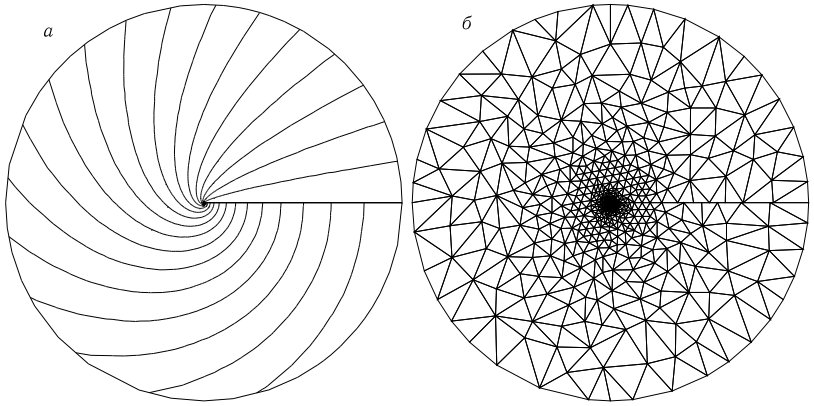


Рис. П.8. Изолинии решения задачи о трещине (*а*) и адаптивная сетка (*б*)

Будем искать решение этой задачи в пространстве непрерывных кусочно-линейных функций, заданных на конформных триангуляциях, а сеточный гессиан восстанавливать по формулам (П.3.1) и (П.3.2), не обращая внимания на особенность в начале полярной системы координат. Как видно из табл. П.4, на построенных адаптивных сетках (см. рис. П.8, *б*) максимальная норма ошибки конечноэлементного решения демонстрирует почти оптимальную скорость сходимости: $\|u - u_h\|_{L^\infty(\Omega)} = O(N_\star^{-0{,}9})$. Причина небольшого отклонения скорости сходимости от единицы заключается в том, что метод конечных элементов минимизирует энергетическую норму ошибки $\|\nabla(u - u_h)\|_{L^2(\Omega)}$.

Таблица П.4
Максимальная норма ошибки конечноэлементного решения для задачи о трещине

$N_\star$	1000	4000	16 000	64 000
$\|u - u_h\|_{L^\infty(\Omega)}$	0,056	0,016	0,005	0,0014

Поскольку максимальная норма ошибки не подчинена энергетической норме, она может падать с меньшей асимптотической скоростью.

Отметим, что построенные адаптивные сетки обеспечивают значительно меньшую ошибку, чем обычные квазиравномерные триангуляции: норма ошибки на сетке с 64 000 треугольников равна 0,16, что на два порядка превосходит величину 0,0014 из табл. П.4.

Алгоритм 45 и метод восстановления метрики (П.3.1), (П.3.2) успешно применяются для адаптивного решения краевых задач с несамосопряженными операторами и анизотропными особенностями решения, такими, как пограничные слои [5, 9, 43, 66]. Более того, квадратичная экстраполяция кусочно-линейных аппроксимаций криволинейных границ (см. § П.1) позволяет эффективно адаптировать сетку около криволинейной границы. В качестве иллюстрации рассмотрим уравнение конвекции–диффузии

$$\begin{cases} -0{,}01\Delta u + \mathbf{b}\cdot\nabla u = 0 & \text{в } \Omega, \\ u = g & \text{на } \Gamma_{\text{in}}, \\ \dfrac{\partial u}{\partial n} = 0 & \text{на } \Gamma_{\text{out}}, \\ u = 0 & \text{на } \partial\Omega \setminus (\Gamma_{\text{in}} \cup \Gamma_{\text{out}}). \end{cases} \qquad (\text{П.3.7})$$

Здесь $\mathbf{b} = (1, 0, 0)^{\mathrm{T}}$ — постоянное поле скоростей, $\Omega = (0, 1)^3 \setminus B_{0,5}(0{,}18)$ — кубическая расчетная область с удаленным шаром $B_{0,5}(r)$ радиуса $r = 0{,}18$ с центром в точке $(0{,}5, 0{,}5, 0{,}5)$. Границами вычислительной области являются поверхность шара, $\Gamma = \partial B_{0,5}(0{,}18)$, и поверхность куба. На поверхности куба мы выделим плоскости $\Gamma_{\text{in}} = \{\mathbf{x} \in \partial\Omega\colon x = 0\}$ и $\Gamma_{\text{out}} = \{\mathbf{x} \in \partial\Omega\colon x = 1\}$. Наконец, $g(y, z) = 16y(1 - y)z(1 - z)$ обозначает стандартный профиль течения Пуазейля.

Решение задачи (П.3.7) имеет пограничный слой вдоль подветренной части сферической границы Γ и очень гладко в теневой зоне за препятствием. Поскольку точное решение неизвестно, в экспериментах оно заменено на кусочно-линейное конечноэлементное решение u_*, вычисленное на очень мелкой адаптивной (квазиоптимальной) сетке, содержащей более $1{,}28 \cdot 10^6$ тетраэдров (см. рис. П.9). Для генерации этой адаптивной сетки использовалось аналитическое представление $\partial\Omega$.

Первый эксперимент (график на рис. П.10, *а*) подтверждает асимптотический результат (П.3.3) с u_* вместо u. Ошибка в максимальной норме почти совпадает с аналитической кривой $60\,N_\star^{-2/3}$.

Во втором эксперименте (график на рис. П.10, *б*) граница Γ аппроксимируется квазиравномерной триангуляцией Γ_h. Мы представляем максимальную норму ошибки адаптивного решения как функцию от $N_\star$ для трех различных значений h. На графике видно насыщение этой

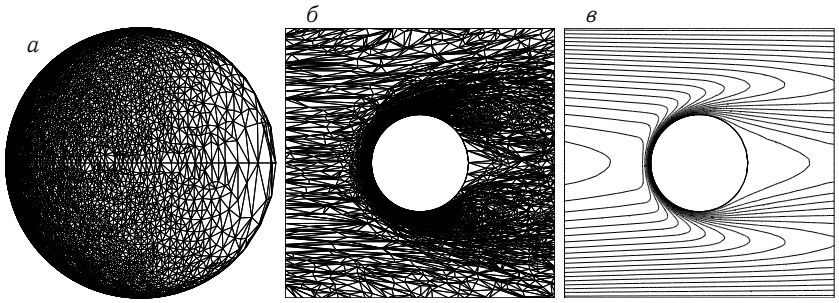


Рис. П.9. След адаптивной сетки на препятствии (*а*), разрез сетки (*б*) и изолинии решения u_* (*в*) в плоскости xy, проходящей через центр препятствия

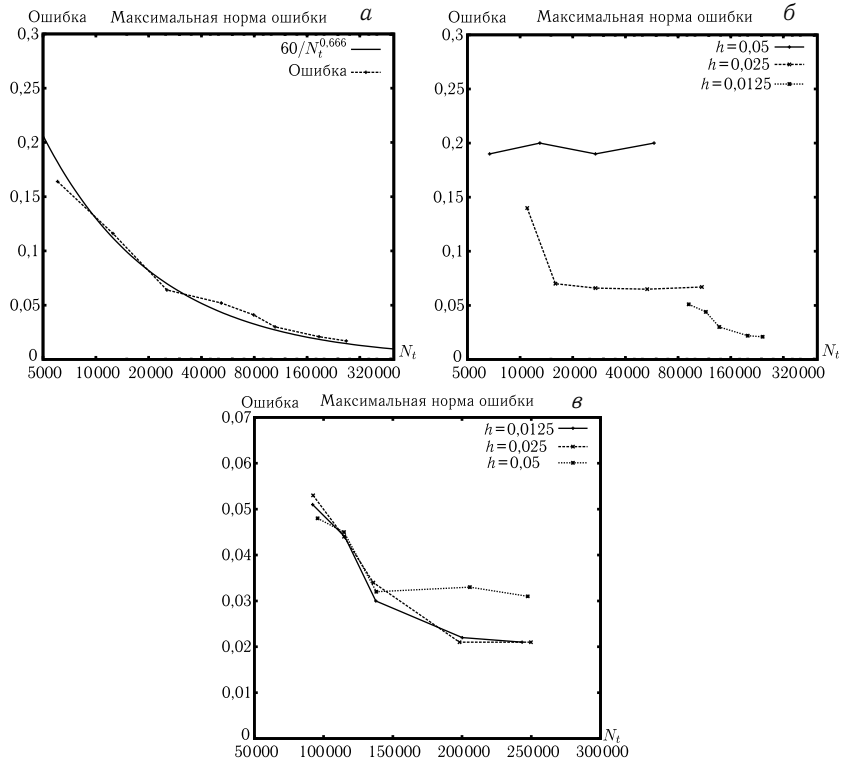


Рис. П.10. Сходимость сеточного решения: в области с аналитическим представлением сферы Γ (*а*), в областях с тремя дискретными моделями $\Gamma_{0,05}$, $\Gamma_{0,025}$ и $\Gamma_{0,0125}$ для сферы Γ (*б*; обозначены линиями 0,05, 0,025, 0,0125), в областях с дискретными моделями $\Gamma_{0,0125}$, $\widetilde{\Gamma}_{0,025}$ и $\widetilde{\Gamma}_{0,05}$ для сферы Γ (*в*; обозначены линиями 0,0125, 0,025, 0,05)

ошибки из-за неточного разрешения криволинейной границы. Отметим, что насыщаемая ошибка ε_h почти обратно пропоциональна h^2: $\varepsilon_{0,05} = 0{,}2$, $\varepsilon_{0,025} = 0{,}067$ и $\varepsilon_{0,0125} = 0{,}021$. Вероятно, это связано с аппроксимацией второго порядка кусочно-линейным многообразием Γ_h для гладкой границы Γ.

В третьем эксперименте (график на рис. П.10, *в*) изучается влияние кусочно-квадратичного восполнения $\widetilde{\Gamma}_h$ для Γ_h на точность дискретного решения. Сравниваются ошибки насыщения для трех фиксированных поверхностных сеток с одинаковым количеством узлов: первая сетка — квазиравномерная триангуляция $\Gamma_{0,0125}$, вторая и третья сетки получены из сетки $\Gamma_{0,0125}$ проецированием ее узлов на востановленные кусочно-квадратичные поверхности $\widetilde{\Gamma}_{0,025}$ и $\widetilde{\Gamma}_{0,05}$ соответственно, т. е. они используют информацию о границе с разрешением $h = 0{,}025$ и $h = 0{,}05$. Для удобства будем обозначать вторую и третью сетки также $\widetilde{\Gamma}_{0,025}$, $\widetilde{\Gamma}_{0,05}$. Графики зависимости ошибки адаптивного решения в областях с фиксированными границами $\Gamma_{0,0125}$ и $\widetilde{\Gamma}_{0,025}$ фактически совпадают, что приводит к примерному равенству ошибок насыщения: $\widetilde{\varepsilon}_{0,025} \approx \varepsilon_{0,0125} \approx 0{,}021$. Отметим, что ошибка насыщения $\varepsilon_{0,025}$ в области с фиксированной границей $\Gamma_{0,025}$ с разрешением $h = 0{,}025$ превышает $\widetilde{\varepsilon}_{0,025}$ более чем в три раза. Более грубое кусочно-линейное представление границы с разрешением $h = 0{,}05$ демонстрирует еще большее различие в ошибках насыщения: $\varepsilon_{0,05} = 0{,}2$ для области с границей $\Gamma_{0,05}$ и $\widetilde{\varepsilon}_{0,05} = 0{,}03$ для области с границей $\widetilde{\Gamma}_{0,05}$.

Таким образом, кусочно-квадратичное восполнение позволяет существенно уменьшить ошибку насыщения сеточной адаптации в областях с криволинейными границами, которые представлены поверхностными триангуляциями.

П.3.2. Тензорная метрика на основе реберных оценок ошибки

Недостатками адаптивной процедуры на основе метрики, полученной восстановлением гессиана сеточной функции, являются большие ошибки восстановления гессиана в окрестности особенностей решения, отсутствие контроля ошибки интерполяции или дискретизации, а также то, что она порождает сетки, минимизирующие L^p-норму ошибки, $0 < p \leqslant \infty$, в то время как конечноэлементные решения, как правило, минимизируют энергетическую норму ошибки. Минимизация энергетической нормы ошибки представляет больший интерес для задач упругости, так как она приводит к более точному вычислению напряжений в зонах предразрушений. Энергетическая норма связана с L^2-нормой *градиентной* ошибки, поэтому для оптимизации ошибки конечноэлементной дискретизации нужно уметь строить метрику, порождающую сетки, минимизирующие L^2-норму градиентной ошибки.

Предлагаемый метод построения тензорной метрики основан на использовании реберных апостериорных оценок энергетической ошибки

конечноэлементного решения. Его основные преимущества заключаются в применимости к более широкому классу функций, включая функции с особенностями, в возможности апостериорно оценить энергетическую норму ошибки и в минимизации нормы ошибки, естественной для метода конечных элементов. Следовательно, адаптивная процедура с использованием такой метрики обеспечивает оптимальную скорость сходимости сеточного решения к дифференциальному и контроль ошибки дискретизации $u_h - u$ [25].

В § П.2 мы рассмотрели иерархический метод оценки апостериорной ошибки, базирующийся на предположении (П.2.5) о том, что разница между кусочно-линейным конечноэлементным решением u_h и кусочно-квадратичным конечноэлементным решением U_h дает хорошее приближение к ошибке $u_h - u$. Для эффективного приближенного вычисления $u_h - U_h$ на каждом треугольнике Δ бралась линейная комбинация (П.2.9) функций-пузырей b_{ij}, связанных с ребрами e_{ij} треугольника Δ. Коэффициенты этой комбинации удовлетворяют системе (П.2.8):

$$u_h - u \approx \varepsilon_h \equiv \sum_{e_{ij} \in \Delta} \widetilde{D}_{Q,\,ij}\, b_{ij}. \tag{П.3.8}$$

Зафиксируем треугольник Δ и введем вектор $\mathbf{d} \in \mathbb{R}^3$ с коэффициентами $\widetilde{D}_{Q,\,ij}$, соответствующими ребрам этого треугольника. Градиентная L^2-норма апостериорно оцененной ошибки ε_h может быть записана следующим образом:

$$\|\nabla \varepsilon_h\|^2_{L^2(\Delta)} = \left\| \sum_{k=1}^{3} d_k \nabla b_k \right\|^2_{L^2(\Delta)} = |\Delta| (\mathbb{B}\,\mathbf{d},\ \mathbf{d}), \tag{П.3.9}$$

где суммирование идет по ребрам e_{12}, e_{23}, e_{31}, а $\mathbb{B}$ обозначает симметричную положительно определенную матрицу 3×3 с элементами

$$\mathbb{B}_{k,\,m} = \frac{1}{|\Delta|} \int\limits_{\Delta} \nabla b_k \cdot \nabla b_m \, dV.$$

К сожалению, норма ошибки на треугольнике — число, которое не дает достаточной информации для определения тензорной метрики $\mathbb{M}$. Для того чтобы определить тензорную метрику, разобьем ошибку на три реберных составляющих $\alpha_k \geqslant 0$ таких, что

$$\|\nabla \varepsilon_h\|^2_{L^2(\Delta)} = |\Delta| \sum_{k=1}^{3} \alpha_k, \quad \alpha_k = |d_k|\,(\mathbb{B}\,\mathbf{d},\ \mathbf{d}) \left(\sum_{k=1}^{3} |d_k| \right)^{-1}. \tag{П.3.10}$$

Расщепление нормы ошибки на три реберные величины α_k имеет простое обоснование. Такой выбор дополнительно равнораспределяет максимальную норму ошибки дискретизации на ребрах треугольника, а следовательно, и на всех ребрах сетки [24, 26].

Три числа α_k позволяют определить три элемента метрического тензора $\mathbb{M}_\Delta$ на треугольнике Δ. Для этого введем квадратичную функцию $v_2 = -\frac{1}{2}\sum_{k=1}^{3}\alpha_k b_k$ и обозначим ее гессиан через $\mathbb{H}_2$. Если $\det\mathbb{H}_2 \neq 0$, положим

$$\mathbb{M}_\Delta = (\det|\mathbb{H}_2|)^{-1/4}|\mathbb{H}_2|, \qquad (\text{П.3.11})$$

где $|\mathbb{H}_2|$ — спектральный модуль матрицы $\mathbb{H}_2$. В противном случае незначительно увеличим наибольшую из трех реберных ошибок, так чтобы гессиан модифицированной квадратичной функции v_2 стал невырожденной матрицей. На практике увеличение на 1 % оказывается достаточным.

Построенная метрика $\mathbb{M}_\Delta$ связывает L^2-норму градиентной ошибки (П.3.10) с геометрическими свойствами треугольника Δ. В [24, 26] доказаны следующие неравенства:

$$\frac{2}{5}\,V^2_{\mathbb{M}_\Delta,\Delta} \leqslant \|\nabla\varepsilon_h\|^2_{L^2(\Delta)} \leqslant V_{\mathbb{M}_\Delta,\Delta}\,|p_{\mathbb{M}_\Delta,\Delta}|^2. \qquad (\text{П.3.12})$$

Принцип равнораспределения ошибки, лежащий в основе адаптивного построения сеток, приводит к равнораспределению следующих геометрических свойств:

$$V_{\mathbb{M}_\Delta,\Delta} \sim |p_{\mathbb{M}_\Delta,\Delta}|^2 \sim \frac{1}{N(\Omega_h)}\,|\Omega_h|_{\mathbb{M}}. \qquad (\text{П.3.13})$$

Таким образом, нам необходимо строить сетки с одинаковыми площадями треугольников и одинаковыми периметрами треугольников, измеряемыми в тензорной метрике $\mathbb{M}$, составленной из кусочно-постоянных метрик $\mathbb{M}_\Delta$. Согласно (П.3.13), $\mathbb{M}$-квазиравномерная сетка Ω_h, содержащая $N_\star$ треугольников, обеспечивает асимптотически оптимальную скорость убывания оценки градиентной ошибки

$$\|\nabla\varepsilon_h\|_{L^2(\Omega)} \sim N_\star^{-1/2}.$$

В случае тетраэдральных сеток вывод метрики повторяет выкладки (П.3.9)–(П.3.11) с поправкой на то, что у тетраэдра шесть ребер, причем масштабирование в (П.3.11) изменено:

$$\mathbb{M}_\Delta = (\det|\mathbb{H}_2|)^{-1/5}|\mathbb{H}_2|. \qquad (\text{П.3.14})$$

Как и для треугольных сеток, $\mathbb{M}$-квазиравномерные сетки, содержащие $N_\star$ тетраэдров, обеспечивают асимптотически оптимальную скорость убывания оценки градиентной ошибки [24, 26]:

$$\|\nabla\varepsilon_h\|_{L^2(\Omega)} \sim N_\star^{-1/3}.$$

В вычислительной практике кусочно-постоянная тензорная метрика заменяется непрерывной метрикой, что обеспечивает более быструю сходимость алгоритма 45. Непрерывная тензорная метрика формируется на основе значений в узлах сетки и линейной интерполяции

внутрь каждого элемента. Значение в узле v берется равным значению метрики на симплексе из множества $\sigma(v)$, имеющей наибольший детерминант.

Таблица П.5

Энергетические нормы ошибки адаптивного конечноэлементного решения $\|\nabla(u_h - u)\|_{L^2(\Omega)}$ задачи о трещине, для двух методов построения тензорной метрики

Метод восстановления метрики \ $N_\star$	1000	4000	16 000	64 000
(П.3.1), (П.3.2)	0,11	0,053	0,026	0,013
(П.2.8), (П.3.9)–(П.3.11)	0,11	0,053	0,027	0,015

Таблица П.6

Энергетические нормы ошибки и оценки ошибки адаптивного конечноэлементного решения для задачи о трещине

ошибка \ $N_\star$	1000	4000	16 000	64 000
$\|\nabla(u_h - u)\|_{L^2}$	0,11	0,053	0,027	0,015
$\|\nabla\varepsilon_h\|_{L^2}$	0,10	0,051	0,026	0,013

Как показывает эксперимент по построению адаптивной сетки в задаче о трещине (П.3.6), оба метода построения метрики (использующий восстановление сеточного гессиана и использующий апостериорную иерархическую оценку ошибки) обеспечивают асимптотически оптимальную ($N_\star^{-1/2}$) скорость падения энергетической нормы ошибки. В табл. П.5 приведены энергетические нормы конечноэлементной ошибки для двух методов построения метрики: а) по формулам (П.3.1), (П.3.2) и б) по формулам (П.2.8), (П.3.9)–(П.3.11). Данные таблицы подверждают двукратное падение энергетической нормы конечноэлементной ошибки при четырехкратном увеличении количества сеточных элементов, что подтверждает асимптотику $N_\star^{-1/2}$.

Дополнительным преимуществом построения метрики по формулам (П.2.8), (П.3.9)–(П.3.11) является контроль нормы ошибки за счет известной апостериорной оценки (П.3.8). В табл. П.6 мы сравниваем энергетические нормы точной ошибки $u_h - u$ и апостериорно оцененной ошибки ε_h. Табличные значения демонстрируют близость норм обеих ошибок. Контроль ошибки конечноэлементного решения в инженерных расчетах является важным свойством этой вычислительной технологии.

Предметный указатель

Адаптация сетки иерархическая 187
— — к границе 177
— — модификациями 137, 194
Алгоритм перестроения сетки 136
— — —: вычислительная сложность 139
— — —: параллельная версия 155
— — —: суперлинейное ускорение 160
— построения тетраэдризации 76
— — триангуляции 49
Апостериорная оценка ошибки иерархическая 192
— — — по невязке 190
— — — — осредненному градиенту 193
— — — через локальные подзадачи 191
— — — — реберные ошибки 203

Базовые алгоритмы модификации тетраэдризации 140
— — — триангуляции 123
— — — у криволинейной границы 123, 131, 141
Библиотека CGM 66
— CUBIT 66
— Open CASCADE 66
Быстрый поиск: блочно упорядоченный список 37, 139
— —: метод бисекции 35
— —: четверичное дерево 38, 57

Граница, заданная с помощью САПР 40
— параметризованная 42, 64, 68, 178
Граф дуальный 31
— планарный 29
— плоский 29
— триангулированный 29

Дерево поиска восьмеричное 38
— — четверичное 38

Метод градиентного спуска 129, 146
— инерциальной бисекции 156
— продвигаемого фронта 47, 67, 75
— разбиения по наибольшему ребру 96
— — — помеченному ребру 96
Метрика: длина ребра сетки 172
—: объем симплекса 172
—: расстояние между точками 170, 171
Многогранник — изолированная лакуна 82
— простой 76
Многоугольник невыпуклый 46
— нестрого выпуклый 46
— строго выпуклый 46

Площадь треугольника 17, 167
Построение метрики по реберным оценкам ошибки 203
— — — сеточному гессиану 195

Реберные ошибки 202

САПР 13
Сетка динамическая 118
— иерархическая 94
—: иерархическое измельчение 95, 104
—: — огрубление 114, 117
—: качество 18, 122
— квазиоптимальная 197
—: локальная модификация 121
—: распутывание 161
— регулярная 20, 166

Сетка симплициальная 15
Сеточный гессиан 184, 195
Симплекс 15, 24
Список блочно упорядоченный 37
— неструктурированный 33
— неявно упорядоченный 35
— структурированный 32
— упорядоченный 34
Суперэлемент 24, 123, 140
—: качество 122

Тензор инерции 156
Тетраэдр: алгебраический объем 21, 142
— вырожденный 21
—: качество 22, 168, 169
—: объем 21
— правильный 21, 140
— регулярный 22
—: сливер 45, 161
Тетраэдризация Делоне 44, 83
— квазиравномерная 23
Тетраэдризация конформная 23
— регулярная 23
Треугольник: алгебраическая площадь 17, 125
— канонический 18
—: качество 18, 168, 169
— ориентированный 75
— равносторонний 18, 171
Триангуляция $\mathbb{M}$-квазиравномерная 173, 194
— Делоне 43
— квазиравномерная 19
— конформная 19
— поверхностная 20, 41, 62, 181, 202
— регулярная 20

Улучшение поверхностной сетки 71

Формула Эйлера 25
Фронт двумерный 47
— трехмерный 76

Список литературы

1. *Азаренок Б. Н.* Вариационные методы построения структурированных сеток и их приложения к газовой динамике: Дис... докт. физ.-мат. наук. — М.: ВЦ РАН, 2009.
2. *Апанович В. Н.*, *Долгова Т. А.* Практическая сходимость метода внешних конечноэлементных аппроксимаций при решении трехмерных задач теории упругости. — Деп. в ВИНИТИ 17.03.93, № 645-В.
3. *Болтянский В. Г.*, *Ефремович В. А.* Наглядная топология. — М.: Наука, 1983.
4. *Боровиков С. Н.*, *Иванов И. Э.*, *Крюков И. А.* Построение тетраэдризации Делоне с ограничениями для тел с криволинейными границами // Ж. вычисл. матем. и матем. физ. 2005. Т. 45. С. 1407–1423.
5. *Василевский Ю. В.*, *Липников К. Н.* Адаптивный алгоритм построения квазиоптимальных сеток // Ж. вычисл. матем. и матем. физ. 1999. Т. 39, № 9. С. 1532–1551.
6. *Василевский Ю. В.*, *Агузал А.* Объединенный асимптотический анализ интерполяционных ошибок на оптимальных сетках // ДАН. 2005. Т. 405, № 3. С. 1–4.
7. *Василевский Ю. В.*, *Вершинин А. В.*, *Данилов А. А.*, *Плёнкин А. В.* Технология построения тетраэдральных сеток для областей, заданных в САПР // Матричные методы и технологии решения больших задач / Под ред. Е. Е. Тыртышникова. — М.: ИВМ РАН, 2005. — С. 21–32.
8. *Василевский Ю. В.* Параллельные технологии решения краевых задач: Дис... докт. физ.-мат. наук. — М.: ИВМ РАН, 2006.
9. *Василевский Ю. В.*, *Липников К. Н.* Использование методики восстановления гессиана при построении адаптивных сеток // Вопр. атом. науки и техн.: сер. Матем. моделирование физ. процессов. 2006. Вып. 3. С. 37–53.
10. *Гаранжа В. А.* Дискретные кривизны, квазиизометрические отображения и квазиоптимальные расчетные сетки: Дис... докт. физ.-мат. наук. — М.: ВЦ РАН, 2011.
11. *Данилов А. А.* Технология построения неструктурированных сеток и монотонная дискретизация уравнения диффузии: Дис... канд. физ.-мат. наук. — М.: ИВМ РАН, 2010.
12. *Данилов А. А.* Построение тетраэдральных сеток для областей с заданными поверхностными триангуляциями // Численные методы, параллельные вычисления и информационные технологии. — М.: МГУ, 2008. — С. 119–130.
13. *Долбилин Н.* Три теоремы о выпуклых многогранниках // Квант. 2001. № 5. С. 7–12.
14. *Иваненко С. А.* Адаптивно-гармонические сетки. — М.: ВЦ РАН, 1997.

15. *Лисейкин В. Д.*, *Лиханова Ю. В.*, *Шокин Ю. И.* Разностные сетки и координатные преобразования для численного решения сингулярно возмущенных задач. — Новосибирск: Наука, 2007.
16. *Лисейкин В. Д.*, *Шокин Ю. И.*, *Васева И. А.*, *Лиханова Ю. В.* Технология построения разностных сеток. — Новосибирск: Наука, 2009.
17. *Мищенко А. С.*, *Фоменко А. Т.* Курс дифференциальной геометрии и топологии. — М.: Факториал Пресс, 2000.
18. *Скворцов А. В.* Триангуляция Делоне и её применение. — Томск: Изд-во Том. ун-та, 2002.
19. Электронный ресурс: Open CASCADE Technology: http://www.opencascade.org/
20. Электронный ресурс: The Common Geometry Module: http://cubit.sandia.gov/cgm.html
21. Электронный ресурс: The Common Geometry Module, Argonne (CGMA): http: //trac.mcs.anl.gov/projects/ITAPS/wiki/CGM
22. Advances in grid generation / Ed. O. Ushakova. — N.Y.: Nova Science Publishers, 2005.
23. *Agouzal A.*, *Lipnikov K.*, *Vassilevski Y.* Adaptive generation of quasioptimal tetrahedral meshes // East-West J. Numer. Math. 1999. V. 7. P. 223–244.
24. *Agouzal A.*, *Lipnikov K.*, *Vassilevski Y.* Hessian-free metric-based mesh adaptation via geometry of interpolation error // Ж. вычисл. матем. и матем. физ. 2010. Т. 50, № 1. С. 131–145.
25. *Agouzal A.*, *Lipnikov K.*, *Vassilevski Y.* On optimal convergence rate of finite element solutions of boundary value problems on adaptive anisotropic meshes // Math. Comput. Simul. 2011. V. 81. P. 1949–1961.
26. *Agouzal A.*, *Vassilevski Yu.* Minimization of gradient errors of piecewise linear interpolation on simplicial meshes // Comput. Meth. Appl. Mech. a. Eng. 2010. V. 199. P. 2195–2203.
27. *Apel T.* Anisotropic finite elements: Local estimates and applications. — Stuttgart: Teubner, 1999.
28. *Arnold D.*, *Mukherjee A. Pouly L.* Locally adapted tetrahedral meshes using bisection // SIAM J. Sci. Comput. 2000. V. 22, № 2. P. 431–448.
29. *Auricchio F.*, *Beirão da Veiga F.*, *Hughes T. J. R. et al.* Isogeometric collocation for elastostatics and explicit dynamics // Comput. Meth. Appl. Mech. a. Eng. 2012, 1 Dec. V. 249–252. P. 2–14.
30. *Bank R. E.* PLTMG: a software package for solving elliptic partial differential equations, users' guide 6.0. — Philadelphia: SIAM, 1990.
31. *Bänsch E.* Local mesh refinement in 2 and 3 dimensions // IMPACT of Comput. in Sci. a. Eng. 1991. V. 3. P. 181–191.
32. *Bazilevs Y.*, *Calo V. M.*, *Cottrell J. A. et al.* Isogeometric analysis using T-splines // Comput. Meth. Appl. Mech. a. Eng. 2010, 1 Jan. V. 199, is. 5–8. P. 229–263.
33. *Borden M. J.*, *Verhoosel C. V.*, *Scott M. A. et al.* A phase-field description of dynamic brittle fracture // Comput. Meth. Appl. Mech. a. Eng. 2012, 1 Apr. V. 217–220. P. 77–95.
34. *Borouchaki H.*, *Hecht F.*, *Saltel E.*, *George P.-L.* Reasonably efficient Delaunay based mesh generator in 3 dimensions // Proc. 4th Int. Meshing Roundtable. — Albuquerque, 1995. — P. 3–14.

35. *Buscaglia G. C.*, *Dari E. A.* Anisotropic mesh optimization and its application in adaptivity // Int. J. Numer. Meth. Engng. 1997. V. 40. P. 4119–4136.
36. *Carey G. F.* Computational grids: generation, adaptation, and solution strategies. — Washington: Taylor & Francis, 1997.
37. *Chugunov V.*, *Vassilevski Yu.* Parallel multilevel data structures for a non-conforming finite element problem on unstructured meshes // Russ. J. Numer. Anal. Math. Modelling. 2003. V. 18, № 1. P. 1–11.
38. *Chugunov V.*, *Svyatski D.*, *Tyrtyshnikov E.*, *Vassilevski Yu.* Parallel iterative multilevel solution of mixed finite element systems for scalar equations // Concur. a. Comput.: Pract. a. Exper. 2006. V. 18, № 5. P. 501–518.
39. *Ciarlet P.* The finite element method for elliptic problems. — Amsterdam: North-Holland, 1978.
40. *Danilov A.* Unstructured tetrahedral mesh generation technology // Ж. вычисл. матем. и матем. физ. 2010. Т. 50, № 1. С. 146–163.
41. *Deuflhard P.*, *Leinen P.*, *Yserentant H.* Concepts of an adaptive hierarchical finite element code // IMPACT. 1989. V. 1. P. 3–35.
42. *Du Q.*, *Wang D.* Recent progress in robust and quality Delaunay mesh generation // J. Comput. a. Appl. Math. 2006. V. 195. P. 8–23.
43. *Dyadechko V.*, *Lipnikov K.*, *Vassilevski Yu.* Hessian based anisotropic mesh adaptation in domains with discrete boundaries // Russ. J. Numer. Anal. Math. Modelling. 2005. V. 20, № 4. P. 391–402.
44. *Elguedj T.*, *Hughes T. J. R.* Isogeometric analysis of nearly incompressible large strain plasticity // Comput. Meth. Appl. Mech. a. Eng. 2014, 1 Jan. V. 268. P. 388–416.
45. *Eymard R.*, *Gallouët T. R.*, *Herbin R.* The finite volume method // Handbook of Numerical Analysis. V. 7/ Ed. P. G. Ciarlet and J. L. Lions. — North Holland, 2000. — P. 713–1020.
46. *Farrell P. E.*, *Maddison J. R.* Conservative interpolation between volume meshes by local Galerkin projection // Comput. Meth. Appl. Mech. a. Eng. 2011. V. 200. P. 89–100.
47. *Fary I.* On straight line representations of planar graphs // Acta Sci. Math. 1948. V. 11. P. 229–233.
48. *Foster I.* Designing and building parallel programs. — N.Y.: Addison-Wesley, 1995.
49. *Freitag L. A.*, *Ollivier-Gooch C.* Tetrahedral mesh improvement using swapping and smoothing // Int. J. Numer. Meth. Eng. 1997. V. 40. P. 3979–4002.
50. *Frey P. J.*, *George P. L.* Mesh generation: application to finite elements. — Oxford: Hermes Science Publishing, 2000.
51. *Garanzha V. L.* Discrete extrinsic curvatures and approximation of surfaces by polar polyhedra // Ж. вычисл. матем. и матем. физ. 2010. Т. 50, № 1. С. 71–98.
52. *Garimella R. V.* Anisotropic Tetrahedral Mesh Generation. — Rensselaer Polytechnic Institute, Report 1998-31, PhD Thesis. — 1998.
53. *Garimella R. V.*, *Swartz B. K.* Curvature estimation for unstructured triangulations of surfaces. — Los Alamos Report LA-UR-03-8240; http: // math.lanl.gov/Research/Publications/Docs/garimella-2003-curvature.pdf.
54. *George P. L.*, *Borouchaki H.* Delaunay triangulation and meshing: application to finite elements. — Paris: Editions Hermes, 1998.

55. *George P. L.*, *Borouchaki H.*, *Saltel E.* 'Ultimate' robustness in meshing an arbitrary polyhedron // Int. J. Numer. Meth. Eng. 2003. V. 58. P. 1061–1089.
56. Handbook of grid generation/ Ed. J. Tompson, B. Soni, N. Weatherill. — Boca Raton, FL: CRC Press, 1999.
57. *Hoppe R.*, *Iliash Yu.*, *Kuznetsov Yu. et al.* Analysis and parallel implementation of adaptive mortar element methods // East-West J. Numer. Math. 1998. V. 6. P. 223–248.
58. *Hughes T. J. R.*, *Cottrell J. A.*, *Bazilevs Y.* Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement // Comput. Meth. Appl. Mech. a. Eng. (Elsevier). 2005, 1 oct. V. 194. P. 4135–4195.
59. *Ivanenko S. A.* Selected chapters on grid generation and applications. — Moscow: Dorodnicyn Computing Center of the Russian Academy of Sciences, 2004.
60. *Klingner B. M.*, *Shewchuk J. R.* Agressive tetrahedral mesh improvement // Proc. 16th Int. Meshing Roundtable/ Ed. M. Brewer, D. Marcum. — Berlin–Heidelberg: Springer-Verlag, 2007. — P. 3–23.
61. *Knupp P.*, *Steinberg S.* Fundamentals of grid generation. — Boca Raton: CRC Press, 1994.
62. *Knupp P.* Algebraic mesh quality metrics // SIAM J. Sci. Comput. 2001. V. 23. P. 193–218.
63. *Kuratowski K.* Sur le probleme de courbe gauches en topologie // Fund. Math. 1930. V. 15. P. 271–283.
64. *Lachaud J.-O.* Topologically defined iso-surfaces // Proc. 6th DGCI. — Berlin: Springer-Verlag, 1996. — P. 245–256.
65. *Levenberg K.* A Method for the Solution of Certain Non-Linear Problems in Least Squares // Quart. Appl. Math. 1944. V. 2. P. 164–168.
66. *Lipnikov K.*, *Vassilevski Yu.* Parallel adaptive solution of 3D boundary value problems by Hessian recovery // Comput. Meth. Appl. Mech. a. Eng. 2003. V. 192, № 11–12. P. 1495–1513.
67. *Lipnikov K.*, *Vassilevski Yu.* On control of adaptation in parallel mesh generation // Eng. with Comput. 2004. V. 20, № 3. P. 193–201.
68. *Liseikin V.* Grid generation methods. — Berlin: Springer-Verlag, 1999.
69. *Liseikin V.* A computational differential geometry approach to grid generation. — Berlin: Springer-Verlag, 2006.
70. *McIvor A. M.*, *Valkenburg R. J.* A comparison of local surface geometry estimation methods // Machine Vision a. Appl. 1997. V. 10. P. 17–26.
71. *Meyer M.*, *Lee H.*, *Barr A. H.*, *Desbrun M.* Generalized barycentric coordinates on irregular polygons // J. Graph. Tools. 2002. V. 7, № 1. P. 13–22.
72. *Misztal M. K.*, *Baerentzen J. A.*, *Anton F.*, *Erleben K.* Tetrahedral mesh improvement using multi-face retriangulation // Proc. 18th Int. Meshing Roundtable/ Ed. B. W. Clark. — Berlin–Heidelberg: Springer-Verlag, 2009. — P. 539–555.
73. *Mokhtarian F.*, *Khalili N.*, *Yuen P.* Curvature computation of free-form 3-D meshes at multiple scales // Comput. Vision a. Image Understanding. 2001. V. 83. P. 118–139.
74. *Nicolaides R. A.* Direct discretization of planar div-curl problems // SIAM J. Numer. Anal. 1992. V. 29. P. 32–56.

75. *Pothen A.*, *Simon H.*, *Liou K.* Partitioning sparse matrices with eigenvectors of graphs // SIAM J. Math. Anal. Appl. 1990. V. 11. P. 430–452.
76. Proc. 16th Int. Meshing Roundtable/ Ed. M. Brewer, D. Marcum. — Berlin–Heidelberg: Springer-Verlag, 2007.
77. Proc. 18th Int. Meshing Roundtable/ Ed. B. W. Clark. — Berlin–Heidelberg: Springer-Verlag, 2009.
78. *Rivara M.* Mesh refinement processes based on the generalized bisection of simplexes // SIAM J. Numer. Anal. 1984. V. 21. P. 604–613.
79. *Rivara M.* Selective refinement/derefinement algorithms for sequences of nested triangulations // Int. J. Numer. Meth. Eng. 1989. V. 28. P. 2889–2906.
80. *Rado T.* Über den Begriff der Riemannschen Flächen // Acta Scient. Mathem. (Szeged). 1925. V. 2. P. 101–121.
81. *Shewchuk J. R.* Constrained Delaunay tetrahedralizations and provably good boundary recovery // Proc. 11th Int. Meshing Roundtable. — Ithaca, 2002. — P. 193–204.
82. *Tautges T. J.* The Common Geometry Module (CGM): A Generic, Extensible Geometry Interface // Eng. with Comput. 2001. V. 17. P. 299–314.
83. *Verfürth R.* A review of a posteriori error estimation and adaptive mesh-refinement techniques. — Stuttgart: Wiley–Teubner, 1996.
84. *Verhoosel C. V.*, *Scott M. A.*, *Borden M. J. et al.* Discretization of Higher Order Gradient Damage Models Using Isogeometric Finite Elements // Damage Mechanics of Cementitious Materials and Structures. — N.Y.: Wiley, 2012. – P. 89–120.
85. *Williams R.* Performance of dynamic load balancing algorithms for unstructured mesh calculations // Concurrency: Practice and experience. 1992. V. 3. P. 457–481.
86. *Zavattieri P.*, *Dari E.*, *Buscaglia G.* Optimization strategies in unstructured mesh generation // Int. J. Numer. Meth. Eng. 1996. V. 39. P. 2055–2071.

ОГЛАВЛЕНИЕ

Предисловие научного редактора пятитомного цикла монографий (*В. А. Левин*) 5
Авторское предисловие 9

Глава 1. **Введение** 11

Глава 2. **Основные понятия** 17
§ 2.1. Треугольные и тетраэдральные сетки 17
§ 2.2. Свойства сеток и элементы теории графов 24
§ 2.3. Структуры данных и быстрые алгоритмы 31

Глава 3. **Построение неструктурированных сеток в произвольных областях** 40
§ 3.1. Способы задания вычислительной области 40
§ 3.2. Построение триангуляции Делоне 43
§ 3.3. Построение триангуляции методом продвигаемого фронта 45
3.3.1. Алгоритм продвигаемого фронта 47
3.3.2. Влияние вычислительных погрешностей 49
3.3.3. Конечность работы алгоритма 10 53
3.3.4. Скорость работы алгоритма продвигаемого фронта 57
3.3.5. Результаты экспериментов 59
§ 3.4. Построение поверхностной триангуляции методом продвигаемого фронта 62
3.4.1. Представление поверхности 63
3.4.2. Взаимодействие с геометрическим ядром САПР 65
3.4.3. Алгоритм продвигаемого фронта 67
3.4.4. Результаты экспериментов 70

§ 3.5. Метод улучшения заданной поверхностной сетки 71
§ 3.6. Построение тетраэдральной сетки методом продвигаемого фронта 75
3.6.1. Алгоритм продвигаемого фронта 76
§ 3.7. Надежный алгоритм построения тетраэдральной сетки. 82
3.7.1. Тетраэдризация Делоне. 82
3.7.2. Восстановление геометрии области 83
3.7.3. Восстановление следа сетки на границе 85
3.7.4. Конечность работы алгоритма 86
3.7.5. Улучшение качества полученной сетки 87
3.7.6. Результаты экспериментов 87

Глава 4. **Многоуровневое иерархическое измельчение и огрубление сетки** 94
§ 4.1. Принципы многоуровневого построения сеток. 94
§ 4.2. Метод бисекций для измельчения триангуляций 95
§ 4.3. Метод бисекций для измельчения тетраэдризаций. 104
§ 4.4. Алгоритм многоуровневого огрубления 111
§ 4.5. Алгоритмы построения динамических сеток 118

Глава 5. **Перестроение симплициальных сеток посредством локальных модификаций** 121
§ 5.1. Принципы организации алгоритмов. 121
§ 5.2. Перестроение триангуляций. 123
§ 5.3. Перестроение тетраэдризаций 140
§ 5.4. Параллелизация трехмерного алгоритма 155
§ 5.5. Исправление и распутывание сеток. 161

Глава 6. **Управление свойствами сеток** 166
§ 6.1. Управление свойствами регулярных сеток 166
§ 6.2. Управление свойствами анизотропных сеток. 170

Приложение. **Некоторые проблемы сеточной адаптации**. 177
§ П.1. Адаптация к внешним и внутренним границам 177
П.1.1. Адаптация триангуляции к криволинейной параметризованной границе 178
П.1.2. Адаптация тетраэдризации к гладкой параметризованной поверхности 179
П.1.3. Адаптация сеток к границам с неизвестной параметризацией . . 181

§ П.2. Адаптация к решению посредством локального иерархического измельчения. 186
П.2.1. Апостериорная оценка ошибки по невязке 190
П.2.2. Апостериорная оценка ошибки, основанная на решении локальных подзадач 191
П.2.3. Иерархическая апостериорная оценка ошибки. 192
П.2.4. Апостериорная оценка ошибки по осредненному градиенту. . . . 193
§ П.3. Адаптация к сеточному решению посредством локальных модификаций 194
П.3.1. Тензорная метрика на основе восстановления гессиана. 195
П.3.2. Тензорная метрика на основе реберных оценок ошибки 202

Предметный указатель 206

Список литературы 208

CAE Fidesys

ПАКЕТ ПРИКЛАДНЫХ ПРОГРАММ
ДЛЯ ПРОЧНОСТНОГО ИНЖЕНЕРНОГО АНАЛИЗА

Триальная версия (на 1 месяц) **продукта и руководство пользователя с пошаговыми примерами доступны по адресу**

http://www.cae-fidesys.com

Тел.: +7 (495) 930-87-53, e-mail: contact@cae-fidesys.com

УВАЖАЕМЫЕ КОЛЛЕГИ!

Мы предлагаем Вам сотрудничество в рамках ряда направлений:

1. Если у Вас есть промышленный модуль для пакета инженерного анализа, то мы предлагаем его интеграцию с CAE Fidesys для их совместного продвижения на рынок.

2. Если у Вас есть научный программный продукт (для любого типа инженерного анализа) или Вы работаете над его созданием, то мы готовы оказать содействие в его коммерциализации путем интеграции с пакетом CAE Fidesys.

3. Если Вы сотрудничаете с промышленными предприятиями, то мы предлагаем вместе с Вами сделать на базе пакета CAE Fidesys новый программный продукт — специализированное отраслевое решение — и передать его предприятию (отрасли). Научно-методическое руководство остается за Вами и/или представителями предприятия (отрасли).

4. Если Вы развиваете новое направление в инженерном анализе, то мы готовы рассмотреть алгоритмизацию и программную реализацию и при необходимости выступить в качестве «индустриального партнера».

Нам интересно и промышленное использование современных численных методов, например разрывного метода Галёркина, изогеометрического анализа.

5. Если Вам интересно внести инновационную составляющую в учебный процесс в Вашем вузе, то мы предлагаем создать (развернуть) «Виртуальную лабораторию по курсам прочностного инженерного анализа» как частное облако Вашего вуза. Это позволит преподавателям дистанционно проводить занятия, а студентам использовать при обучении планшеты или даже смартфоны. В этом случае мы можем предложить совместную разработку лабораторных работ.

Указанное частное облако может быть дополнено возможностью использовать пакет CAE Fidesys в работе инжинирингового центра Вашего вуза.

Кроме того, мы готовы при приобретении вузом одной коммерческой лицензии поставить до 5 учебных лицензий в промышленной комплектации. При этом мы предлагаем создание совместных с Вашим вузом центров обучения для промышленных пользователей CAE Fidesys.