Министерство науки и образования Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра теоретических основ информатики

УДК 681.03

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, д.т.н., проф.
_____Ю.Л. Костюк
«___»_____2004 г.

Кравченко Дмитрий Геннадьевич

ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА СТРУКТУРНЫХ ЛИНИЙ ДЛЯ ПОСТРОЕНИЯ ЦИФРОВОЙ МОДЕЛИ РЕЛЬЕФА МЕСТНОСТИ

Дипломная работа

Научный руководитель,
к.т.н., доцент _____ А.Л. Фукс

Исполнитель,
студент гр. 1491 _____ Д.Г. Кравченко

Электронная версия дипломной работы помещена
в электронную библиотеку. Файл
Администратор

Томск – 2004

РЕФЕРАТ

Дипломная работа: 38 с., 19 рис., 6 источников, 2 приложения.

ГЕОИНФОРМАЦИОННАЯ СИСТЕМА, ЦИФРОВАЯ МОДЕЛЬ РЕЛЬЕФА, ТРИАНГУЛЯЦИЯ ДЕЛОНЕ, ТРИАНГУЛЯЦИЯ С ОГРАНИЧЕНИЯМИ, СТРУКТУРНЫЕ ЛИНИИ РЕЛЬЕФА, ПРЕДОБРАБОТКА ИСХОДНЫХ ДАННЫХ

(1) Объект исследования – цифровая модель рельефа местности.

(2) Цель исследования – разработка программного обеспечения, обеспечивающего предварительную обработку исходных данных для построения высококачественной модели рельефа.

(3) Метод исследования – экспериментальный на ЭВМ.

(4) Полученные результаты – разработано программное обеспечение, позволяющее устранять основные недостатки структурных линий рельефа, полученных на основе векторизации картографических материалов.

Содержание

Введение

В настоящее время в регионах Российской Федерации много усилий прилагается к созданию специальных информационных систем, которые призваны обеспечить поддержку управления территориями. Учитывая значительную роль в таких задачах так называемых пространственных факторов, ставка делается на использование геоинформационных систем (ГИС). Одной из таких систем является «ГИС – Природные ресурсы Республики Алтай», которая разрабатывается в НПО «Сибгеоинформатика» с целью информационного и аналитического обеспечения решения задач по управлению природными ресурсами региона.

С учетом того, что территория республики является горной, особую важность приобретает обработка данных о рельефе. Многие задачи природопользования требуют обязательного учета свойств рельефа. При работе с отдельными объектами требуются графические данные крупных масштабов – от 1: 25 000 и крупнее. Передача данных о рельефе для таких масштабов является сложной задачей, так как данная поверхность, как правило, осложнена многочисленными структурными нарушениями, которые для таких масштабов существенны и должны учитываться.

В настоящее время основным методом получения информации о рельефе региона является оцифровка существующих картографических материалов. Рельеф на картах представлен наборами высотных отметок и структурных линий. Высотные отметки обычно определяют характерные точки рельефа, например, локальные экстремумы. Структурные линии задают дополнительные ограничения на форму рельефа. Это могут быть, прежде всего, горизонтали (изолинии, изогипсы) и области резкого изменения наклона поверхности (границы оврагов, береговые линии, линии насыпей, обрывов и т.п.).

Процесс оцифровки достаточно трудоемкий и монотонный, выполняется многими операторами, поэтому неизбежно возникает много ошибок, как атрибутных, так и графических. В подавляющем большинстве это различные пики, петли на линиях, осцилляции или излишняя сглаженность, приводящая к неоправданному росту объемов данных. Применение известных методов аппроксимации для устранения данных ошибок может привести к недопустимым пересечениям изолиний разных уровней и другим топологическим нарушениям.

Качество цифровой модели рельефа (ЦМР) зависит, прежде всего, от качества исходных данных. Поэтому необходимым этапом в процессе построения ЦМР является предварительная обработка исходных данных о рельефе. Решению данной задачи и посвящена настоящая дипломная работа.

1. Исходные данные и основные методы построения цифровой модели рельефа

1.1. Данные о рельефе в задачах природопользования

Рельеф является важнейшей характеристикой любой территории и поэтому данные о рельефе необходимы при решении разнообразных задач. В природопользовании такими задачами являются:

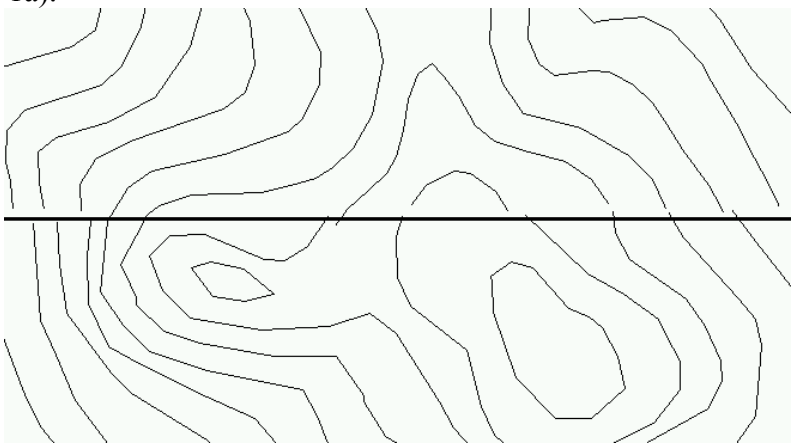
1. Расчет ширины водоохранных зон (ширина зоны зависит от свойств рельефа вдоль береговой линии). В этих зонах устанавливается особый режим хозяйствования, т.е. они являются определенными ограничениями при недропользовании.
2. Точное определение по картографическим материалам площадей земельных участков, предоставляемых недропользователям.
3. Комплексная оценка участка с месторождением полезных ископаемых для расчета платежей за недропользование.
4. Оценка воздействия на окружающую среду при разработке месторождений.
5. Поиск наилучших вариантов трасс при проектировании линейных объектов (дорога, линии электропередач, трубопровод) для обустройства месторождений и разведочных площадей.
6. Оценка объемов вскрышных работ при проектировании карьеров для добычи полезных ископаемых.
7. Выбор промплощадок при создании горнодобывающих предприятий.

Как правило, результатом решения вышеназванных задач являются различные зоны (полигоны) или линейные объекты с соответствующими атрибутивными характеристиками. Эти новые объекты должны быть совмещены с исходными картами для дальнейшего использования и визуального анализа. При этом границы новых объектов должны хорошо коррелировать с исходной моделью рельефа, прежде всего с изолиниями (изогипсами), а также с теми объектами, которые на исходных материалах связаны с рельефом – линии водотоков, контура растительного покрова, дороги и т. п. Таким образом, очень важно иметь высококачественную цифровую модель местности, чтобы по ней правильно решать различные расчетные и аналитические задачи, в том числе с использованием данных о рельефе.

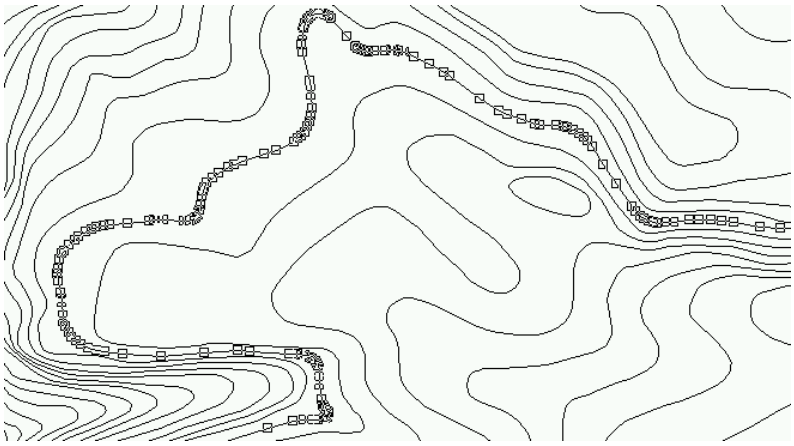
В большинстве случаев исходными данными для построения ЦМР являются наборы структурных линий и высотных отметок. Высотные отметки обычно определяют характерные точки рельефа, например, локальные экстремумы. Структурные линии задают дополнительные ограничения на форму рельефа. Это могут быть, прежде всего, горизонтали (изолинии, изогипсы) и области резкого изменения наклона поверхности (границы оврагов, береговые линии, линии насыпей, обрывов и т.п.). Объем информации об изолиниях обычно существенно превышает объем информации о высотных отметках и линиях резкого изменения наклона поверхности. Поэтому далее, говоря о структурных линиях, будем иметь в виду, прежде всего, изолинии.

Как правило, структурные линии получаются в результате ручной или полуавтоматической оцифровки бумажных карт. Процесс оцифровки достаточно трудоемкий и монотонный, выполняется многими операторами, поэтому неизбежно возникает много ошибок. Например, объем полученных таким путем данных в масштабе 1:100 000 на Республику Алтай составляет более 115 Мб, а количество ошибок операторов измеряется тысячами.

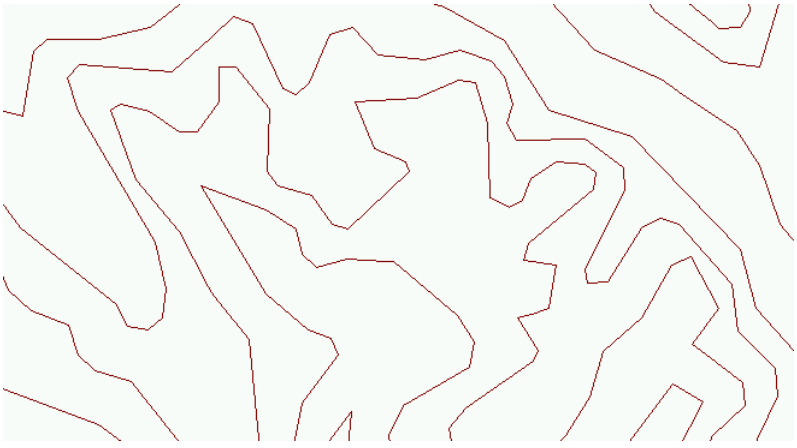
Кроме ошибок в атрибутных данных, велико количество графических ошибок. В подавляющем большинстве это различные пики, петли на изолиниях, двойные (дублирующие) изолинии. Существует и ряд технологических недостатков. Как правило, оцифровка ведется «полистно», причем соседние листы могут обрабатываться разными операторами, сшивка листов производится формально, и в результате на стыке листов возникают скачки, изломы (рис. 1а).



а)



б)



в)

Рис. 1. Технологические недостатки оцифровки изолиний: а) нестыковка изолиний на стыке смежных листов, порождающая скачки и изломы; б) излишнее количество узловых точек; в) недостаточное количество узловых точек, приводящее к осцилляциям изолиний

При использовании автоматических режимов векторизации (что далеко не всегда возможно) изолинии часто оказываются излишне сглаженными, что приводит к неоправданному росту объемов материала (рис. 1б). При использовании ручных режимов оцифровки изолинии получаются слишком «ломаными», сильно осциллируют (рис. 1в). Хотя используемые современные средства векторизации формально позволяют избежать всех этих недостатков, реально выдержать технологический оптимум на практике оказывается не всегда возможным.

Многие векторизаторы, в частности, EasyTrace предоставляют функцию сглаживания изолиний, но эти методы не учитывают влияния остальных изолиний и других данных о модели рельефа, в результате чего возможны недопустимые пересечения изолиний разных уровней и другие топологические нарушения. Поэтому для повышения качества ЦМР исходные горизонтали (возможно, и другие структурные линии) нужно заменить аппроксимирующими линиями, которые должны быть визуально гладкими, содержать минимальное число узловых точек и при этом сохранять топологическую корректность набора линий.

1.2. Основные методы построения цифровой модели рельефа

При работе с рельефом важнейшая задача заключается в построении его цифровой модели, т.е. цифрового представления с помощью прямоугольной или треугольной сетки, в узлах которой заданы высоты. Цифровая модель позволяет получить производные данные для обработки и анализа поверхности.

Для построения цифровой модели на основе прямоугольной сетки необходимо построить триангуляцию по проекциям исходных точек в плоскости *XOY*, а затем с помощью методов локальной аппроксимации и интерполяции рассчитать высоты в узлах сетки по значениям высот в нескольких исходных соседних точках. Основным преимуществом такого способа построения ЦМР является простая и удобная структура прямоугольной сетки, которая позволяет упростить алгоритмы анализа и обработки поверхности.

Однако у прямоугольной сетки есть 2 существенных недостатка. Координаты исходных высотных отметок и узловых точек линий резкого изменения наклона поверхности, как правило, удается определить с высокой точностью, поэтому данные точки желательно сохранить в результирующей модели рельефа, но в получаемой цифровой модели они утрачиваются. Кроме того, при использовании прямоугольной сетки практически невозможно учитывать ограничения, налагаемые структурными линиями. Оба этих недостатка отсутствуют в ЦМР на основе треугольной сетки.

В ЦМР на основе треугольной сетки исходные нерегулярные наборы высотных отметок и линий структурируются на основе триангуляции с ограничениями. Для этого по проекциям высотных отметок и узловых точек линий строится триангуляция Делоне [3], которая далее перестраивается таким образом, чтобы проекции всех отрезков исходных структурных линий совпадали с ребрами треугольников [5].

Получающаяся в результате сетка с заданными в ее узлах высотами в большинстве известных геоинформационных систем используется для анализа и обработки данных о рельефе. Однако, в [2] показано, что подобная модель при наличии изолиний в наборе структурных линий является недопустимо грубым приближением реального рельефа местности. Это связано с тем, что в кусочно-линейной поверхности, полученной с помощью триангуляции с ограничениями, содержатся недопустимые горизонтальные участки на уровнях изолиний. В [2] и [6] эту проблему предлагается решать с помощью дополнительной триангуляции с сильными ограничениями, в процессе которой горизонтальные участки перестраиваются путем расчета дополнительных вершин ЦМР.

1.3. Предварительная обработка структурных линий рельефа

Как было указано выше, целью обработки является замена исходных структурных линий аппроксимирующими линиями, которые должны быть визуально гладкими, содержать минимальное число узловых точек и при этом сохранять топологическую корректность набора линий.

В [6] рассматривается решение похожей задачи геометрической аппроксимации расчетных горизонталей (сечений) однозначной поверхности. В соответствии с ним, на треугольной сетке для каждого сечения выделяется область его допустимого расположения – коридор, границы которого (ломаные) рассчитываются с учетом погрешности определения координат вершин триангуляции. Затем внутри коридора строится по возможности плавно изменяющаяся линия, которая и считается искомым сечением. При расчете границ коридора производится выделение последовательности ребер треугольников, через которые должно проходить данное сечение, при этом границы коридоров также пересекают эти ребра. Однако данный метод имеет один недостаток: если сечение проходит точно через вершину ребра, то и границы коридора будут прижиматься к данной вершине, и коридор будет вырождаться.

В настоящей работе решается задача обработки не расчетных, а исходных изолиний, поэтому аппроксимация линий должна проводиться до начала построения ЦМР. Тем не менее, очевидно, что для возможности управления формой и расположением кривой и для исключения пересечения изолиний разных уровней, можно использовать набор непересекающихся коридоров для всех изолиний. Однако для построения коридоров необходимо структурировать исходные данные так, чтобы можно было легко выделить области, непосредственно прилегающие к каждой линии. Для этого по исходному набору структурных линий и высотных отметок нужно построить триангуляцию с ограничениями. Каждый отрезок линии станет при этом ребром триангуляции, поэтому можно элементарно выделять треугольники, примыкающие к линии слева и справа по ходу движения.

После построения триангуляции все отрезки структурных линий совпадают с ребрами, а, значит, будут проходить через вершины треугольников, поэтому коридоры, используемые в [6], в нашем случае будут вырождаться. В [1] предлагается модификация метода, предложенного в [6], для аппроксимации исходных структурных линий до начала построения ЦМР. В настоящей дипломной работе был реализован именно этот метод.

Отметим, что полученная триангуляция с ограничениями используется только как вспомогательная структура для выделения коридоров и не рассматривается как ЦМР. После расчета коридоров все исходные структурные линии будут заменены расчетными коридорными, поэтому можно использовать любой алгоритм триангуляции с ограничениями – как с добавлением дополнительных точек, так и без него [5].

2. Структуризация исходных данных на основе триангуляции

2.1. Построение триангуляции Делоне

Начальный этап при выделении коридоров – это построение триангуляции Делоне на основе исходных наборов высотных отметок и узлов структурных линий.

Триангуляция – это одна из форм представления поверхности по нерегулярно заданной системе отсчётов в виде совокупности смежных треугольников.

Задача построения триангуляции состоит в следующем. Дано n точек на плоскости. Требуется соединить их непересекающимися отрезками так, чтобы образовалось планарное разбиение плоскости со следующими свойствами:

- Триангулярность. Разбиение должно состоять из m фигур, из которых m–1 треугольники и еще одна – внешняя бесконечная фигура.
- Выпуклость. Нельзя построить ни одного нового отрезка между данными точками без пересечения с уже существующими.

Среди всевозможных видов триангуляции часто пользуются триангуляцией Делоне, которая дополнительно также удовлетворяет условию Делоне (рис. 2): внутрь окружности, описанной вокруг любого ее треугольника, не попадает никакая из вершин других треугольников.

В этом случае треугольники становятся настолько «равноугольными», насколько возможно. Гарантируется также, что любая точка на плоскости наиболее близка к вершинам треугольника, внутрь которого она попала, чем к другим точкам. Триангуляция Делоне часто используется на практике для построения кусочно-линейной модели поверхности.

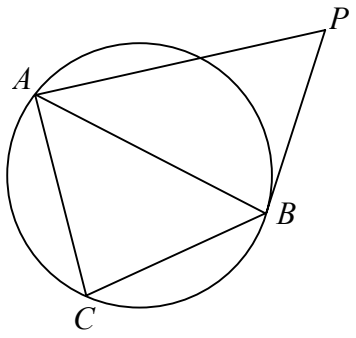


Рис. 2. Проверка условия Делоне для треугольников *ABC* и *APB*

Большинство алгоритмов построения триангуляции Делоне можно условно разделить на две основные группы: итеративные алгоритмы и алгоритмы, построенные с использованием принципа «разделяй и властвуй» (алгоритмы слияния) [5].

Алгоритмы слияния предполагают разбиение исходного множества точек на несколько подмножеств, построение триангуляций на этих подмножествах, а затем объединение (слияние) полученных триангуляций в единое целое.

Итеративные алгоритмы являются логически самыми простыми алгоритмами построения триангуляции Делоне: по 3-4 точкам строится начальная триангуляция, а затем все исходные точки по одной добавляются в текущую систему треугольников. Для каждой добавляемой точки необходимо провести две основные операции: локализация (определение треугольника, внутрь которого попадает точка) и перестроение системы треугольников таким образом, чтобы условие Делоне не нарушалось ни на одном из ребер триангуляции.

Трудоемкость итеративного алгоритма триангуляции Делоне в наихудшем составляет $O(n^2)$. Для любой известной реализации алгоритма всегда можно подобрать такое расположение n точек, при котором трудоемкость построения триангуляции будет близка

к наихудшей. В то же время для любого набора точек можно найти такой порядок их выбора, при котором трудоемкость триангуляции будет линейной. В силу единственности триангуляции Делоне конечный вид системы треугольников не зависит от порядка задания точек, поэтому задача определения нужного порядка имеет существенное значение для повышения быстродействия алгоритма. Эта задача была подробна исследована в работе [6], в которой был предложен итеративный алгоритм триангуляции Делоне, имеющий линейную трудоемкость, если размещение исходных точек в области определения удовлетворяет ограничениям, реально существующим в большинстве практических задач. В данном алгоритме для ускорения построения триангуляции производится предварительная обработка набора исходных точек, в результате которой строится 2D-дерево точек. Во время триангуляции данная структура трансформируется в структуру связей «точка – треугольник», которая существенно упрощает как последующую триангуляцию с ограничениями, так и процесс выделения коридоров. В настоящей работе для построения триангуляции Делоне использовался именно этот алгоритм.

Идея итеративного алгоритма состоит в том, что на каждом шаге мы имеем триангуляцию, построенную на некотором подмножестве исходного множества точек (частичную триангуляцию), в которую добавляем по одной точке из оставшихся, изменяя, возможно, в соответствии с условием Делоне связи между вершинами треугольников.

Для каждой добавляемой точки A необходимо провести две основные операции: локализацию и перестроение системы треугольников. Если A окажется за границей триангуляции, то необходимо достраивать новые треугольники с общей вершиной A так, чтобы граница снова стала выпуклым многоугольником. Для исключения таких вариантов к исходному набору добавляются 4 вершины прямоугольника, объемлющего исходные точки, а начальная триангуляция строится именно по этим вершинам. Тогда точка A никогда не окажется за границей триангуляции и обязательно попадет внутрь некоторого треугольника.

Таким образом, начальный шаг алгоритма – это начальная триангуляция по вершинам объемлющего прямоугольника. Следующий шаг – предобработка точек для определения нужного порядка добавления точек в триангуляцию.

Наконец, после предобработки применялся обычный итеративный алгоритм, добавляющий точки в триангуляцию в порядке, определенном деревом точек, сформированным на предыдущем шаге.

Локализация точки T предполагает последовательную проверку смежных треугольников, расположенных вдоль отрезка ST, где S – точка принадлежит исходному треугольнику, а T – искомому. При реализации алгоритма применялся следующий метод проверки попадания T в очередной треугольник, основанный на том, что порядок следования вершин в любом треугольнике соответствует обходу треугольника против часовой стрелки. Для любого треугольника ABC последовательно вычисляются векторные произведения ребер треугольника и отрезков, соединяющих T с вершинами треугольника $\overrightarrow{AT} \times \overrightarrow{AB}$, $\overrightarrow{BT} \times \overrightarrow{BC}$, $\overrightarrow{CT} \times \overrightarrow{CA}$, точнее, только их Z-координаты, т.к. все векторы-сомножители лежат в плоскости XOY. Если очередная вычисленная величина окажется положительной, то T находится вне ABC, проверка прекращается, а поиск продолжается в треугольнике, смежном с ABC по соответствующему ребру, например, если Z-координата $\overrightarrow{AT} \times \overrightarrow{AB}$ $z_{ATB} > 0$, то поиск необходимо продолжать в треугольнике, смежном с ABC по ребру AB. В противном случае проверяется векторное произведение для следующего ребра треугольника.

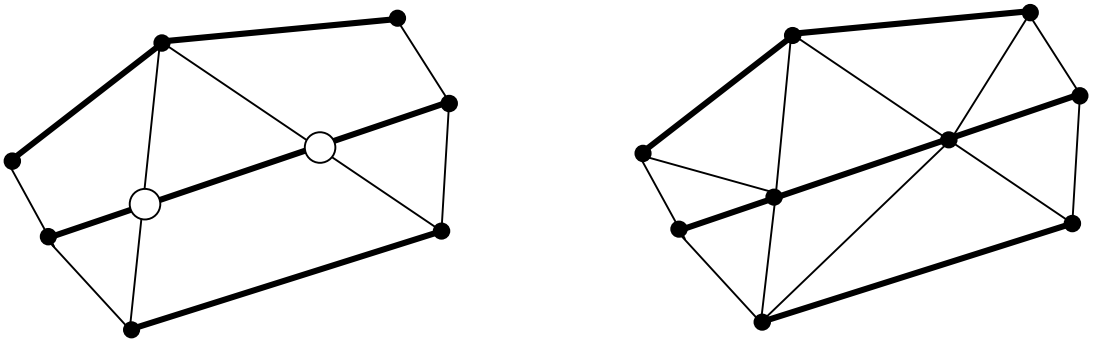
2.2. Построение триангуляции с ограничениями

Триангуляцию Делоне необходимо перестроить так, чтобы проекции всех отрезков исходных структурных линий совпадали с ребрами треугольников триангуляции.

Для корректной работы алгоритма перестроения необходимо, чтобы исходные данные были топологически корректными. Предполагается, что рельеф – это однозначная поверхность, поэтому попадания высотных отметок на изолинии, пересечения изолиний разных уровней, а также самопересечения структурных линий недопустимы. Топологические нарушения легко обнаружить, однако корректно устранить их возможно только в интерактивном режиме. Перед началом работы алгоритма предобработки изолиний предполагается, что такая работа уже проведена.

Кроме того, если структурные линии соприкасаются, пересекаются или высотная отметка расположена на линии, то в исходном наборе точек для алгоритма триангуляции Делоне обязательно окажутся совпадающие точки, при этом только одна из них может быть включена в триангуляцию. Поэтому в наборе классов, реализующих алгоритм сглаживания изолиний, был предусмотрен ряд процедур предварительной обработки линий, исключающий случаи пересечения отрезков линий или попадания точек внутрь отрезка. При этом для каждой вершины триангуляции предусмотрено хранение и ведение списка совпадающих с ней вершин.

Известно несколько подходов к вставке отрезков структурных линий в триангуляцию [5]. В настоящей работе использовался метод вставки «Строй, разбивая». Его суть состоит в том, что для каждого вставляемого отрезка линии ищутся точки пересечения этого отрезка с ребрами триангуляции, и эти точки вставляются как в триангуляцию с перестроением имеющихся треугольников, так и в структурную линию, разбивая вставляемый отрезок на несколько более мелких (рис. 3). При этом триангуляция остается триангуляцией Делоне, а поэтому не содержит слишком вытянутых треугольников, площадь которых близка к нулю.



(а) (б)

Рис. 3. Вставка структурных отрезков в триангуляцию Делоне методом «Строй, разбивая»: (а) – начальная триангуляция, (б) – разбиение отрезка на части

Выбор данного метода обусловлен тем, что исходные изолинии нужны только для выделения коридоров и будут далее заменены новыми линиями с новыми узловыми точками, поэтому добавление новых узлов в исходные линии вполне допустимо.

3. Выделение коридора для структурной линии на треугольной сетке

3.1. Расчет границ коридора

После построения треугольной сетки для каждой структурной линии необходимо рассчитать область допустимого расположения линии – коридор, границами которого являются две непересекающиеся ломаные, а затем внутри коридора строится по возможности плавно изменяющаяся линия. Границы коридора задают область, за пределы которой кривая не должна выходить при использовании различных методов интерполяции. В настоящей работе использовался метод выделения коридора, предложенный в [1].

Согласно этому методу для каждого типа структурных линий определяются два значения абсолютной погрешности: ε_{XY} – точность задания координат узлов в плоскости XOY и ε_Z - точность расчета высот в узлах. Пусть AB – отрезок структурной линии L, а M – вершина, соединенная с A ребром триангуляции, причем отрезок AM не является отрезком структурной линии (рис. 4).

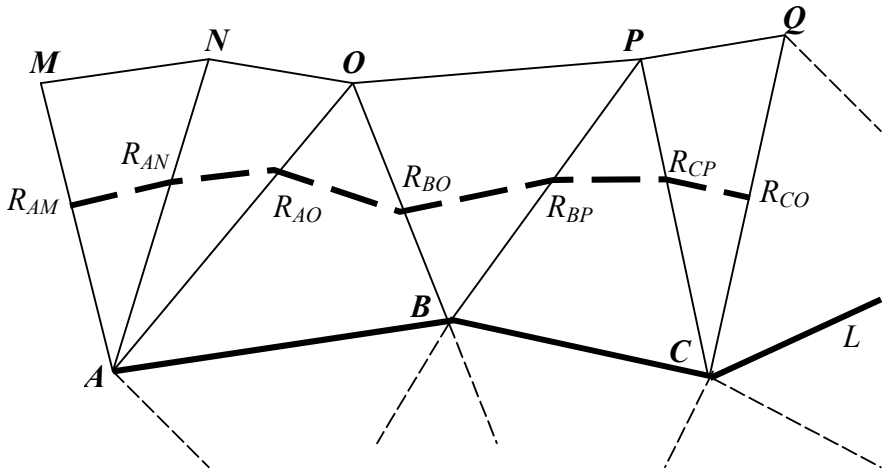


Рис. 4. Выделение коридора для изолинии по треугольной сетке

Считается, что точка R на ребре AM принадлежит коридору линии L, если она удовлетворяет условиям:

$$\begin{aligned} &\sqrt{(x_R - x_A)^2 + (y_R - y_A)^2} \le \varepsilon_{XY}, \\ &\left|z_R - z_A\right| \le \varepsilon_Z, \\ &\sqrt{(x_R - x_A)^2 + (y_R - y_A)^2} \le \frac{1}{2}\sqrt{(x_M - x_A)^2 + (y_M - y_A)^2}, \end{aligned} \qquad (1)$$

т.е. расстояние между R и A как в плоскости XOY, так и по высоте не должно превышать заданной величины погрешности, и точка R должна находиться ближе к A, чем к M, чтобы исключить возможность пересечения границ коридоров. Левая граница коридора линии L получается при последовательном просмотре всех ее вершин и смежных с ними ребер, расположенных слева от L (AM, AN, AO, BO и т.д.) и определении на этих ребрах узловых точек левой границы коридора – R_{AM}, R_{AN}, R_{AO}, R_{BO} и т.д. (рис. 4). Аналогично рассчитываются узлы правой границы коридора L, только рассматриваются ребра, примыкающие к вершинам L справа.

Отдельно стоит сказать о том, как выделяются начальные и конечные узловые точки для каждой из границ коридора. Для корректной работы расчета коридорной линии не-

обходимо, чтобы любой треугольник, через который проходит левая или правая граница коридора, пересекался этими границами строго один раз. Для этого достаточно, чтобы угол *LAR*, где *L* и *R* – начальные вершины границ коридора, *A* – начальная вершина исходной линии, не превышал 180º (рис. 5).

Подобрать соответствующие вершины на каждой из границ достаточно просто. Например, для начальных вершин надо просматривать ребра триангуляции от первого отрезка *AB* структурной линии в 2 направлениях – по и против часовой стрелки вокруг первой вершины *A* линии (рис. 5), вычисляя на них точки, удовлетворяющие соотношениям (1).

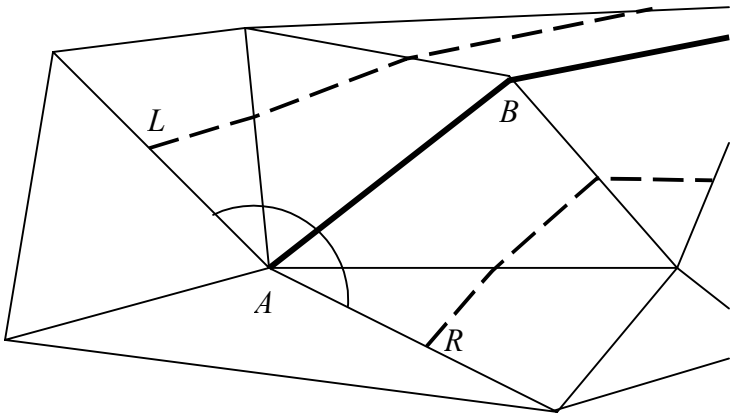


Рис. 5. Определение начальных точек границ коридора

Допустим, *L* и *R* – последние выделенные точки на каждом из 2 направлений. Они и берутся в качестве начальных вершин на каждой из границ. Аналогично находятся последние вершины границ коридора.

На рис. 6 приведен пример рассчитанных коридоров для изолиний.

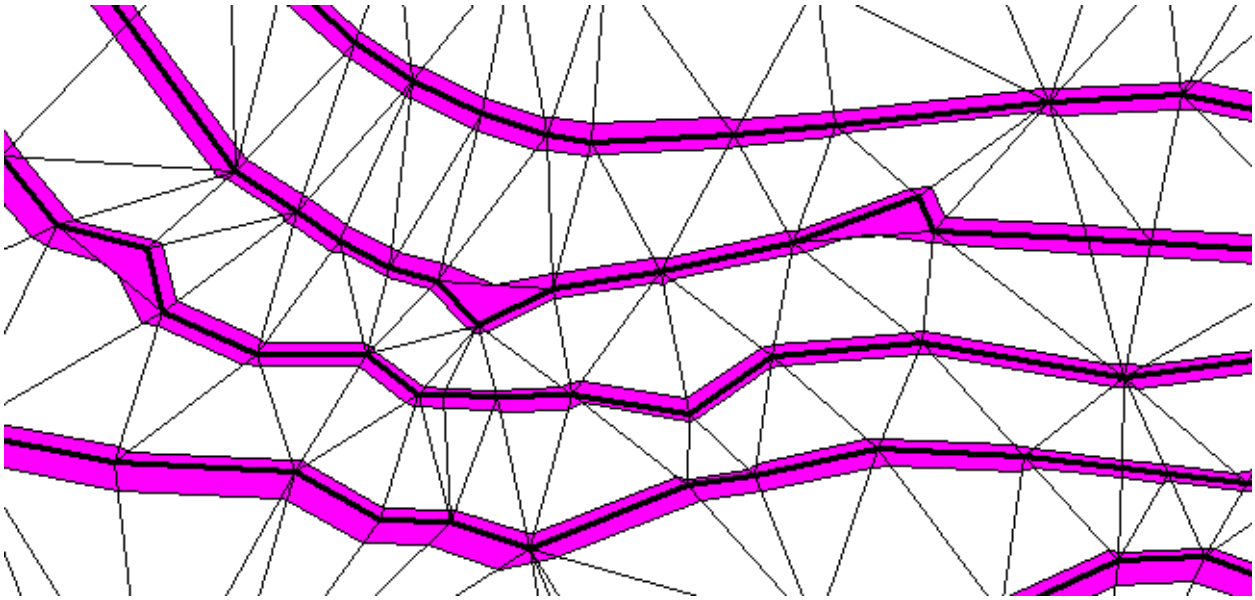


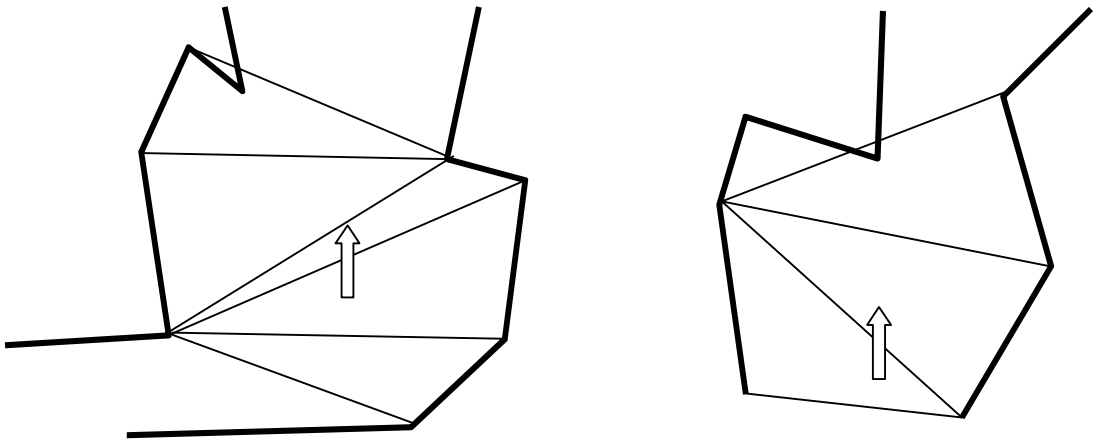
Рис. 6. Примеры коридоров для изолиний на треугольной сетке

3.2. Модификация границ коридора

Расчет линии, целиком лежащей внутри коридора, требует проверок взаимного расположения отрезков линии и соответствующих участков двух границ коридора. Однако, в отличие от задачи расчета сечений однозначной поверхности, в нашем случае границы коридора вычисляются независимо, и число их узлов может различаться. Поэтому, проверка попадания линии в такой коридор существенно усложняется. Самый простой способ решения данной проблемы – построение триангуляции Делоне по точкам двух границ внутри коридора. Каждый треугольник в данной триангуляции соответствует одному участку линии, поперечные ребра (соединяющие вершины разных границ) являются границами отдельными участков расчетной линии. Триангуляция приведет к модификации границ коридора: число вершин в каждой из них будет равно числу поперечных ребер, причем каждая вершина входит в границу столько раз, скольким поперечным ребрам она инцидентна.

В [1] предложен алгоритм модификации границ коридора. Его идея состоит в последовательном прохождении вдоль обеих границ коридора. На каждом шаге анализируются 2 текущие точки с каждой границы (l_i и r_j) и 2 следующие за ними (l_{i+1} и r_{j+1}). Для треугольника $l_i r_j r_{j+1}$ и точки l_{i+1} проверяется условие Делоне. При его выполнении в новые формируемые границы добавляется пара вершин l_i и r_{j+1}, в противном случае вершины l_{i+1} и r_j. Добавляемые вершины становятся текущими. Фактически внутри коридора строится триангуляция Делоне, а каждая добавляемая к новым границам пара вершин представляет собой ребро этой триангуляции. Причем триангуляция строится таким образом, что для любого ее треугольника часть вершин принадлежит одной границе, а часть другой, т.е. отсутствуют треугольники, все 3 вершины у которых принадлежат только одной из границ. Каждая вершина начальной границы, инцидентная s внутренним ребрам триангуляции, превращается в s последовательных и совпадающих вершин модифицированной границы. Поэтому, если в исходной правой границе было n точек, а в левой – m, то в обеих новых границах будет по $n+m-1$ точек.

Однако данный алгоритм работает только для таких коридоров, ширина которых достаточно мала по сравнению с длинами ребер триангуляции. При достаточной ширине коридоров возможна ситуация, когда в каком-либо коридоре в принципе невозможно построить такую триангуляцию, так как ребра триангуляции будут пересекать границы коридора (рис. 7).



(а) (б)

Рис. 7. Случаи пересечения ребрами триангуляции Делоне границ коридора (стрелка указывает направление прохождения вдоль границ коридора)

В настоящей работе предлагаются 3 новых способа модификации границ коридора. Первый способ самый простой. Его суть состоит в построении триангуляции Делоне по набору узловых точек границ для каждого коридора без перестроения отрезков границ и последующем исключении из триангуляции тех треугольников, все 3 вершины которых принадлежат только одной из сторон коридора (рис. 8). К сожалению, данный способ требует существенных дополнительных затрат времени и памяти.

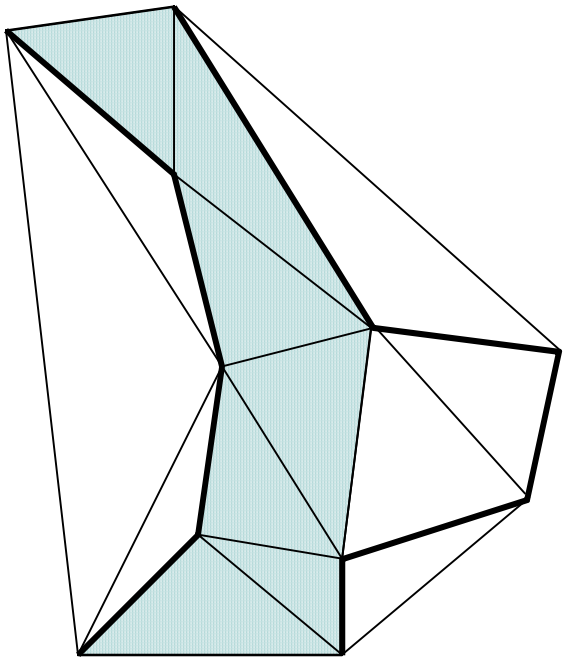


Рис. 8. Модификация границ коридора с помощью построения триангуляции Делоне из точек обрабатываемой линии. Выделен участок, соответствующий модифицированному коридору

Два других способа рассмотрены подробно в следующих разделах.

3.3. Алгоритм модификации границ коридора, основанный на последовательном прохождении коридора

Второй способ является модификацией алгоритма, приведенного в [1]. Пусть R_i и L_j – последняя пара вершин, включенных в новые границы коридора. Модификация заключаются в том, что на каждом шаге происходит проверка факта пересечения отрезком R_iL_j границ коридора. Если пересечение имеет место (рис. 9а), осуществляется 2 действия. Во-первых, откат назад – удаление последних добавленных отрезков триангуляции до того состояния, из которого можно перестроить триангуляцию без пересечения границ (рис. 9б). Во-вторых, добавление пар вершин к новым границам так, чтобы соответствующие отрезки не пересекали границ коридора (рис. 9в).

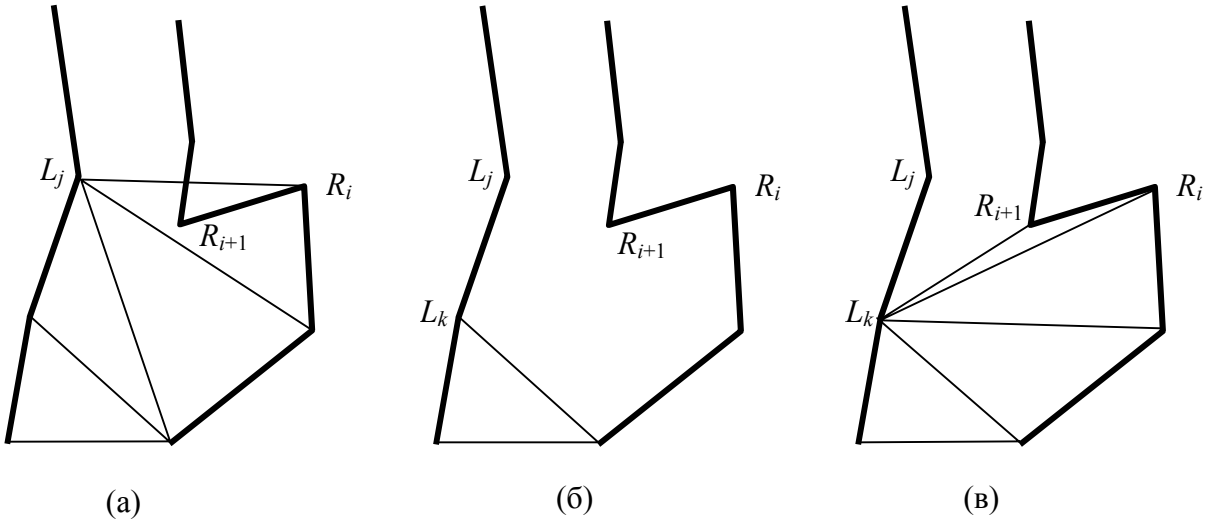


Рис. 9. Исправление пересечения ребрами триангуляции границ коридора: (а) – исходное состояние, (б) – откат назад – удаление последних добавленных отрезков триангуляции, (в) – добавление пар вершин к новым границам

Кроме того, на каждом шаге контролируется, чтобы добавляемый отрезок (R_iL_{j+1} или $R_{i+1}L_j$) не пересекал границ коридора. Для простоты проверка условия Делоне на каждом шаге заменена проверкой углов, образуемых R_iL_j и следующими отрезками границ. Если $\angle R_{i+1}R_iL_j < \angle R_iL_jL_{j+1}$, то к новым границам добавляются вершины R_{i+1} и L_j, иначе R_i и L_{j+1}.

Таким образом, вместо проверки условия Делоне на каждой итерации выполняются следующие шаги:

1) Проверка факта пересечения отрезком R_iL_j границ коридора. Введем 2 функции: *точка_правее* (C, A, B), которая возвращает **истина**, если точка C расположена правее луча AB, и **ложь** в противном случае, и *величина_угла* (A, B, C), возвращающая значение угла ABC в градусах (строго говоря, для разных границ коридора должны использоваться разные функции *величина_угла*). Тогда необходимое условие пересечения правой границы коридора можно сформулировать так:

   точка_правее(L_j, R_i, R_{i+1}) and *величина_угла*(R_{i+1}, R_i, R_{i-1}) < 180

   Если оно истинно (рис. 9а), то на левой границе находится такая точка L_k, что *точка_правее*(L_k, R_i, R_{i+1}) возвращает значение **истина**, затем из новых границ удаляются последние добавленные пары вершин r_v и l_w до тех пор, пока не выполнится $l_w = L_k$ (рис. 9б), после чего L_k последовательно соединяется отрезками триангуляции со всеми вершинами на правой стороне от последней добавленной до R_{i+1} включительно (рис. 9в). После этого происходит переход на начало следующей итерации цикла.

   Для левой границы коридора условие пересечения можно сформулировать аналогично, и корректировка пересечения производится по аналогии, если поменять местами правую и левую границы.

2) Проверка факта пересечения правой и левой границ коридора отрезками $R_{i+1}L_j$ и R_iL_{j+1}. Условие пересечения какой-либо из границ выглядит так: *точка_правее*(L_{j+1}, R_i, R_{i+1}) or not *точка_правее*(R_{i+1}, L_j, L_{j+1}). Если оно ложно, происходит переход к следующей проверке. Иначе определяется, какой именно отрезок пересекает границы – $R_{i+1}L_j$ или R_iL_{j+1}, и в новые границы

добавляется отрезок, не пересекающий границ коридора. Если истинна первая часть условия, то вычисляется значение угла $R_{i+1}R_iL_j$. Если оно меньше 180º, это свидетельствует о том, что отрезок R_iL_{j+1} пересекает границы коридора, поэтому в новые границы добавляются вершины R_{i+1} и L_j, иначе в новые границы добавляются R_i и L_{j+1}. Если истинна вторая часть условия, то пересечение аналогично анализируется по значению угла $R_iL_jL_{j+1}$. После этого происходит переход на начало следующей итерации цикла.

3) Сравнение значений углов $R_{i+1}R_iL_j$ и $R_iL_jL_{j+1}$. Если $\angle R_{i+1}R_iL_j < \angle R_iL_jL_{j+1}$, то к новым границам добавляются вершины R_{i+1} и L_j, иначе R_i и L_{j+1}.

Заметим, что на 1 и 2 шаге возможны ситуации, когда дальнейшее построение триангуляции в коридоре невозможно без пересечения границ коридора, поэтому необходима модификация формы этих границ (рис. 10–12). Рассмотрим эти ситуации подробнее.

На первом шаге на левой границе находится такая вершина L_k, что *точка_правее*(L_k, R_i, R_{i+1}) возвращает значение **истина**. Однако, такой точки может не существовать, либо на левой границе есть такая вершина L_p, $p > k$, для которой уже выполнялись такие же действия на 1 шаге, как и для вершины R_i в данной ситуации (рис. 10а). В этом случае вершина R_i удаляется из правой границы коридора, а из новых границ удаляются все последние добавленные пары вершин r_v и l_w до тех пор, пока не выполнится $l_w = L_k$ (рис. 10б), после чего происходит переход на начало следующей итерации цикла.

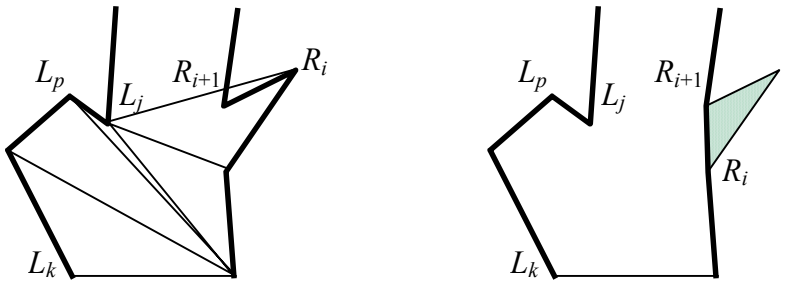
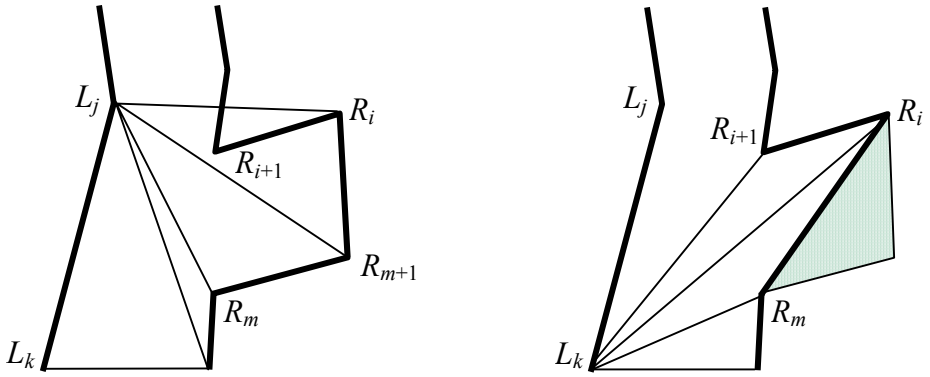


Рис. 10. Модификация формы границ коридора: (а) – исходное состояние, (б) – модифицированное. Выделен отсеченный участок старого коридора

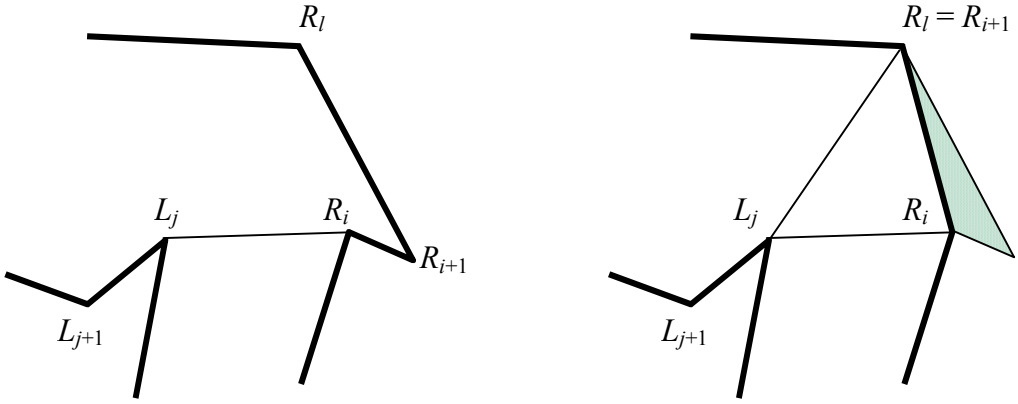
Далее на этом же шаге L_k последовательно соединяется отрезками триангуляции со всеми вершинами на правой стороне от последней добавленной до R_{i+1} включительно. Однако при этом отрезок R_mL_k (R_m – очередная добавляемая вершина на правой границе) может пересекать правую границу (рис. 11а), поэтому перед соединением необходимо постоянно проверять значение функции *точка_правее*(R_m, L_k, R_{m+1}). Если функция возвращает значение **ложь**, то необходимо исключить вершину R_{m+1} из правой границы коридора (рис. 11б).



(а) (б)

Рис. 11. Модификация формы границ коридора: (а) – исходное состояние, (б) – модифицированное. Выделен отсеченный участок старого коридора

На втором шаге возможен случай, когда одновременно истинны оба условия *точка_правее*(L_{j+1}, R_i, R_{i+1}) и not *точка_правее*(R_{i+1}, L_j, L_{j+1}). Это означает, что и $R_{i+1}L_j$ и R_iL_{j+1} будут пересекать границы коридора (рис. 12а). В этой ситуации надо выбрать из еще не просмотренных вершин на одной из границ такую точку P, для которой одно из условий стало бы ложным и при этом отрезок, соединяющий P с текущей вершиной, расположенной на противоположной границе, не пересекал границ коридора. Пусть такая точка P нашлась на правой границе (в случае с левой границей последовательность действий аналогичная), и $P = R_l$. Тогда нужно удалить из правой границы вершины с номерами от i+1 до l–1 включительно и добавить R_l и L_j к новым формируемым границам (рис.12б), после чего перейти к началу следующей итерации.



(а) (б)

Рис. 12. Модификация формы границ коридора: (а) – исходное состояние, (б) – модифицированное. Выделен отсеченный участок старого коридора

Таким образом, данный алгоритм модифицирует границы коридора необходимым образом путем добавления кратных вершин, при этом он устойчив к случаям, когда форма коридора не позволяет соединить вершины отрезками так, чтобы они не пересекали границы коридора. Трудоемкость данного алгоритма оценить достаточно трудно, однако очевидно, что общее количество итераций даже в наилучшем случае не меньше, чем у немодифицированного алгоритма – n+m–1. Кроме того, количество элементарных операций

на каждом шаге имеет порядок выше $O(1)$. Отсюда вытекает существенный недостаток данного алгоритма – достаточно высокая трудоемкость для большинства случаев. Поэтому также был рассмотрен и реализован третий способ модификации границ коридора, который описывается в следующем разделе.

3.4. Использование триангуляции с сильными ограничениями

Этот способ использует предобработку построенной треугольной сетки до выделения границ коридоров, для того, чтобы форма получаемых коридоров позволяла применить алгоритм, предложенный в [1]. Эта предобработка заключается в устранении ребер триангуляции, которые соединяют точки, принадлежащие одной структурной линии, но при этом не являются отрезками этой линии, например, ребра L_1L_4 и L_2L_4 на рис. 13. Наличие таких ребер может привести к такому изменению формы границ коридора, при котором проверка условия Делоне в простейшем алгоритме, приведенном в п. 3.2 не будет давать корректные результаты.

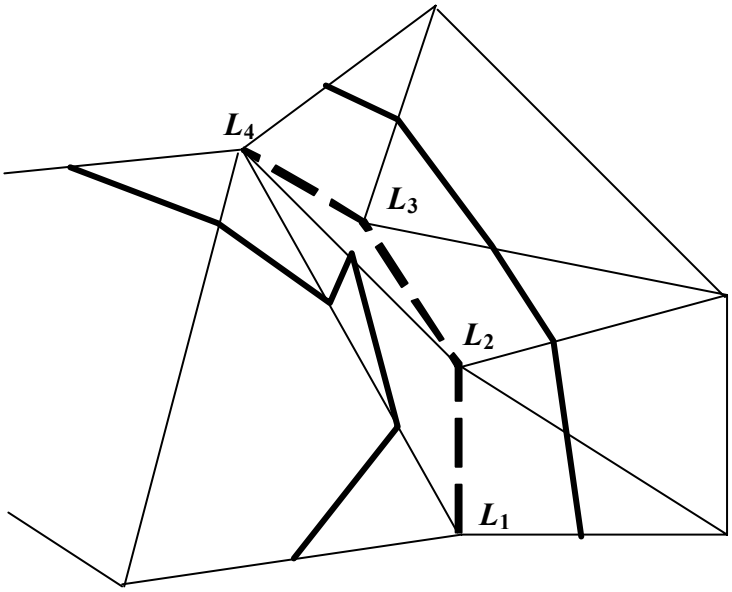


Рис. 13. Изменение формы коридора вследствие наличия ребер триангуляции, которые соединяют точки, принадлежащие одной структурной линии, но при этом не являются отрезками этой линии

В [2] такие ребра называются «недопустимыми», а триангуляция с ограничениями, в которой отсутствуют недопустимые ребра, называется «триангуляцией с сильными ограничениями». Содержательно, учет сильных ограничений означает устранение горизонтальных треугольников, которые появляются при построении обычной триангуляции с ограничениями (в [2] она называется «триангуляция со слабыми ограничениями») на наборе структурных линий, содержащем изолинии (горизонтали). Триангуляция с сильными ограничениями является более точной ЦМР, чем триангуляция со слабыми ограничениями, т.к. более полно учитывает ограничения, задаваемые горизонталями. Идея алгоритма устранения недопустимых ребер состоит в следующем. Находится цепочка недопустимых ребер (т.е. последовательность содержащих их треугольников) и разбивается на 2 более коротких путем добавления новой вершины в середину одного из ребер. При добавлении производится перестроение триангуляции как в итеративном алгоритме триангуляции Делоне, но перестраиваются только недопустимые ребра. Ребро, в которое добавляется вершина выбирается на основе анализа разности высот в вершинах начального и конечного треугольников цепочки. Более подробно этот алгоритм изложен в [2].

Коридоры, выделенные на треугольной сетке, учитывающей сильные ограничения, лучше подходят для обработки алгоритмом модификации коридора, предложенном в [1]. На рис. 14 приведен участок коридора с рис. 13, построенный после удаления из триангуляции недопустимых ребер, а также фрагмент построенной в этом коридоре триангуляции Делоне.

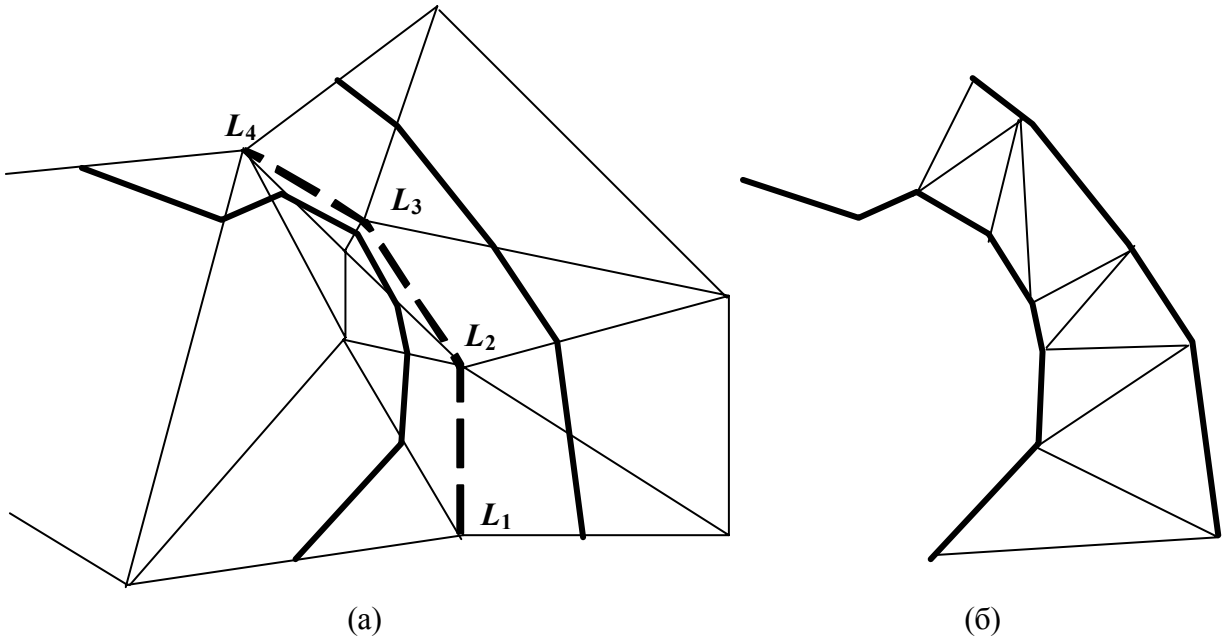


Рис. 14. Расчет коридора после триангуляции с сильными ограничениями: (а) – участок коридора, построенного после удаления недопустимых ребер, (б) – фрагмент построенной в этом участке коридора триангуляции Делоне

4. Построение расчетной линии внутри коридора

4.1. Нахождение ломаной минимальной длины в коридоре

Следующий шаг в предобработке структурных линий – это построение в каждом коридоре ломаной минимальной длины с заданными начальной и конечной точками. Эта ломаная по форме подобна нити с двумя закрепленными концами, растянутой внутри коридора. Если основной целью предобработки является лишь уменьшение числа осцилляций и генерализация структурной линии, а особой гладкости не требуется, то данная линия может использоваться в качестве расчетной структурной линии. Важно отметить, что число узловых точек минимальной ломаной существенно ниже, чем у исходной структурной линии.

В [6] предлагаются 2 алгоритма построения ломаной минимальной длины. В данной работе использовался алгоритм, основанный на отслеживании сектора видимости. Его суть состоит в том, при последовательном прохождении вдоль границ коридора из последней добавленной точки минимальной ломаной постоянно отслеживается сектор видимости остальных точек коридора. Лучи сектора определяются 2 вершинами – по одной на левой и на правой границах. При переходе к следующим парам точек этот сектор может только уменьшаться, поэтому на каком-то шаге отрезок, образованный очередной парой точек оказывается не виден из последней добавленной вершины минимальной ломаной, т.е. лежит вне сектора видимости. Это означает, что этот отрезок полностью закрывает либо левая, либо правая граница коридора. Если отрезок закрывается правой границей, то к ломаной минимальной добавляется точка, определяющая правый луч сектора видимости, иначе – точка, определяющая левый луч.

Отметим, что если с добавляемой вершиной совпадает еще несколько кратных вершин, в ломаную они не включаются.

Этот алгоритм был выбран для реализации потому, что в нем вершины ломаной выделяются последовательно, и поэтому нет необходимости в преобразовании массивов точек, задающих границы, в списки и обратно. На рис. 15 изображен фрагмент ломаной минимальной длины, построенной в одном из коридоров.

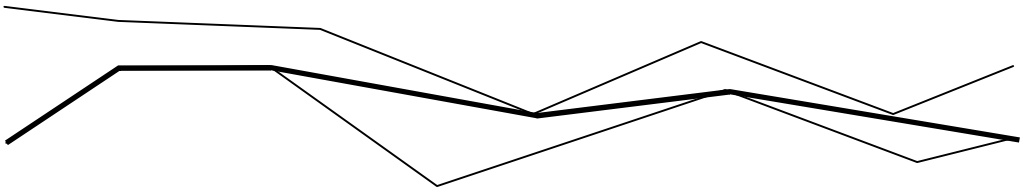


Рис. 15. Ломаная минимальной длины в коридоре

Недостатком этой ломаной является то, что на участках выпуклости (вогнутости) ломаная минимальной длины прижимается к одной из границ коридора. Для устранения данного недостатка использовалась предложенная в [6] последовательность действий:

1. Построение ломаной минимальной длины в исходном коридоре.
2. Модификация коридора, при которой все вершины границ, совпадающие с узлами ломаной на участках выпуклости, сдвигаются к центру коридора, т.е. переносятся в середины отрезков, соединяющих их с вершинами противоположной границы (рис. 16). В нашем случае, если с узлом ломаной совпадает несколько кратных вершин, то либо каждая из этих вершин должна заменяться серединой соответствующего отрезка, либо все они должны заменяться се-

рединами 2 отрезков, соответствующих первой и последней из этих вершин. В данной работе был использован только первый вариант.

3. Построение ломаной минимальной длины в модифицированном коридоре.

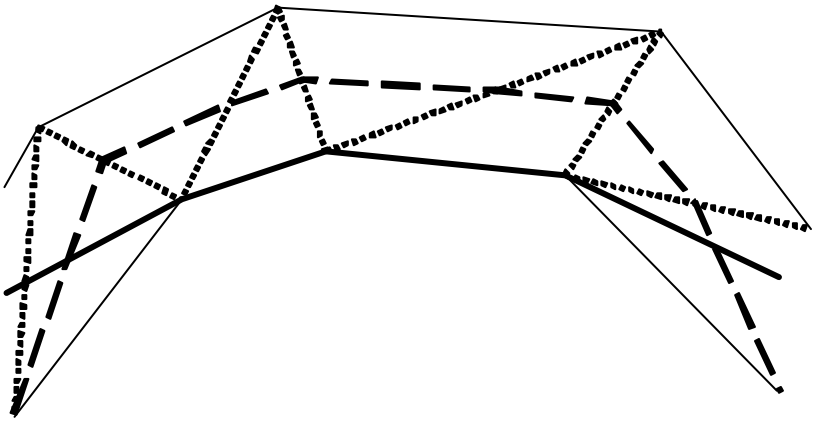
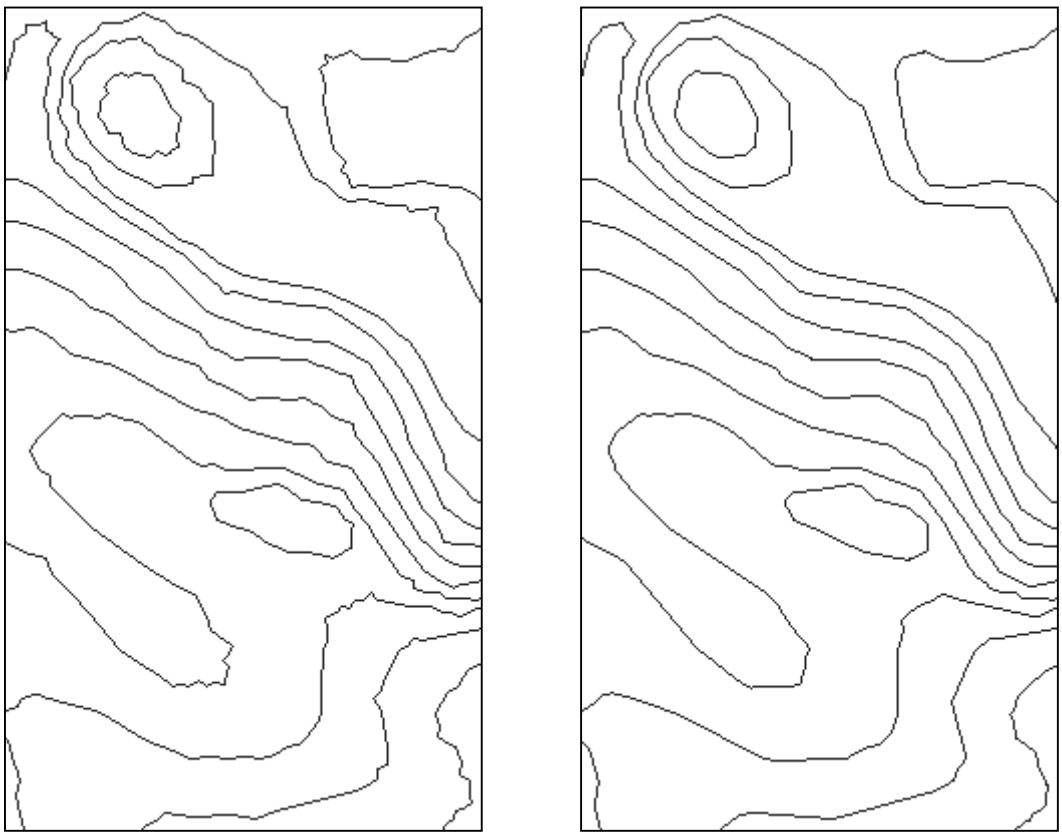


Рис. 16. Модификация коридора

В результате строится линия, узлы которой располагаются, в основном, в центре коридора, число осцилляций увеличивается незначительно, а порядок трудоемкости не увеличивается.

Если не требуется гладкости расчетных линий, эта линия может использоваться в качестве выходной для алгоритма предобработки изолиний. На рис. 17 приведены для сравнения исходные и преобразованные изолинии одного участка местности, где преобразованные изолинии являются ломаными минимальной длины в модифицированных коридорах и содержат примерно в 2 раза меньше узловых точек, чем исходные.



(а) (б)

Рис. 17. Наборы изолиний для участка рельефа: (а) – исходные изолинии, (б) – расчетные изолинии – ломаные минимальной длины в модифицированных коридорах

4.2. Замена отрезков ломаной кубическими кривыми Безье

Так как ломаная минимальной длины очень далека от гладкой, то в ряде случаев ее необходимо интерполировать гладкой кривой. В большинстве практических случаев достаточно, чтобы интерполирующая кривая имела непрерывный наклон (непрерывный единичный вектор касательной). Для исключения возможности пересечения нескольких кривых необходимо, чтобы каждая линия целиком лежала в своем коридоре. Одним из наиболее простых способов такого сглаживания является замена отрезков ломаной минимальной длины кубическими кривыми Безье [4]

$$P(t) = \sum_{i=0}^{3} P_i J_{3,j}(t),\ 0 \le t \le 1,$$

где $J_{3,j}(t) = C_3^i t^i (1-t)^{3-i}$ – полиномы Бернштейна, а P_i – вершины характеристического многоугольника.

Пусть M_i, M_{i+1} – две соседние вершины ломаной минимальной длины. Тогда кривую Безье на i-м отрезке определят четыре вершины многоугольника, причем $P_0 = M_i$, $P_3 = M_{i+1}$, а дополнительные точки P_1 и P_2 рассчитываются через наклоны и модули векторов производных в точках M_i и M_{i+1}. Это связано со свойством кривой Безье: векторы производной в конечных точках равны $3 \cdot \overrightarrow{P_0P_1}$ и $3 \cdot \overrightarrow{P_2P_3}$. Если на ($i$–1)-м участке кривую Безье определяют точки Q_0, Q_1, Q_2 и Q_3, то, очевидно, $Q_3 = P_0 = M_i$, а для обеспечения непрерывности наклона в M_i точки Q_2, M_i и P_1 должны лежать на одной прямой.

В настоящей работе для вычисления наклонов кривой Безье в каждой вершине минимальной ломаной использовался метод расчета наклонов для сплайна с равными углами между производной и 2 прилегающими отрезками ломаной. Согласно этому методу на *i*-м участке кривой наклон $\overrightarrow{V_{M_i}}$ в точке M_i рассчитывается по следующей формуле:

$$\overrightarrow{V_{M_i}} = \frac{\overrightarrow{M_i} - \overrightarrow{M_{i-1}}}{l_{i-1}} + \frac{\overrightarrow{M_{i+1}} - \overrightarrow{M_i}}{l_i},$$

где l_k – длина *k*-го участка ломаной минимальной длины.

4.3. Вписывание кривой Безье в коридор

После определения наклонов в узловых точках ломаной необходимо на каждом участке расположить на прямых, задаваемых этими наклонами две промежуточные точки характеристического многоугольника, причем сделать это нужно таким образом, чтобы кривая Безье целиком лежала внутри соответствующего участка коридора. В настоящей работе для этого применяется способ, предложенный в [1], который состоит в следующем. Векторы производных в конечных точках каждого отрезка ломаной нормируются на длину этого отрезка. Для этого на каждом *i*-м участке P_1 и P_2 задаются таким образом, чтобы длины отрезков P_0P_1 и P_2P_3 составляли 1/3 длины отрезка M_iM_{i+1}. После этого, получаемая кривая Безье заменяется ломаной с небольшим количеством узлов и сравнивается с границами коридора. При этом вычисляется коэффициент *r*, показывающий во сколько раз расстояние от заменяющей ломаной до хорды M_iM_{i+1} больше расстояния от границы участка коридора до M_iM_{i+1} в том месте, где ломаная больше всего выходит за границы. Если описанная ломаная полностью умещается в коридоре, то $r = 1$, иначе $r < 1$. Далее используется известное свойство кубической кривой: при уменьшении модулей конечных производных в *r* раз расстояние от любой точки кривой $P(t)$ до хорды M_iM_{i+1} уменьшается также в *r* раз. Если $r < 1$, то модули производных в P_0 и P_3 умножаются на *r*, и рассчитываются новые координаты P_1 и P_2. После этого кривая Безье снова заменяется ломаной и сравнивается с границами коридора, и находится новое значение *r*. Этот итеративный процесс продолжается до тех пор, пока кривая не будет вписана в коридор. Как показывают практические проверки, достаточно выполнить 1-2 итерации. На рис. 18 приведен пример кривых Безье, вписанных в коридоры.

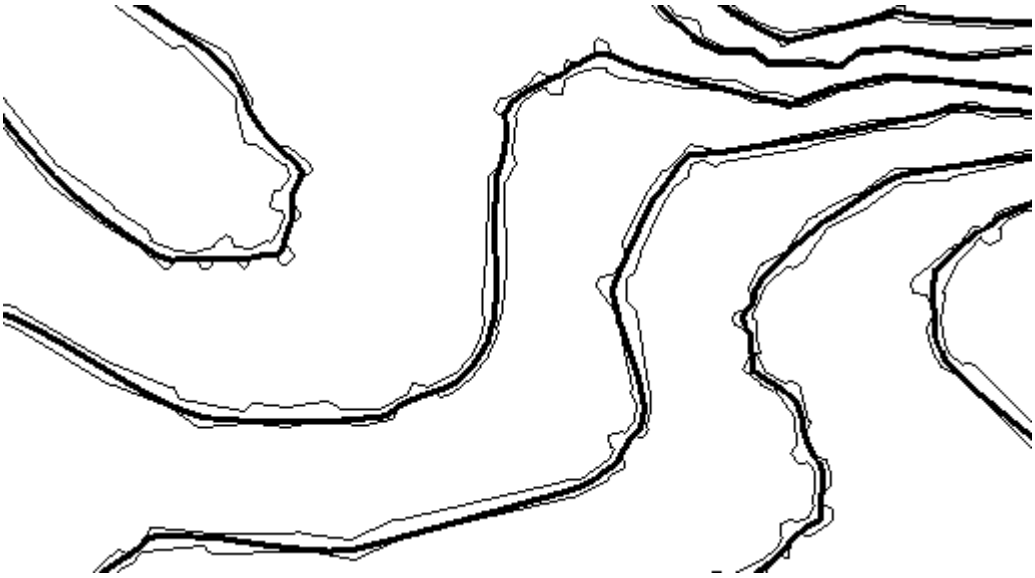


Рис. 18. Кривые Безье, вписанные в коридоры

Таким образом, для каждой структурной линии строится ломаная минимальной длины и набор промежуточных точек характеристических многоугольников Безье, каждому отрезку ломаной соответствует пара промежуточных точек. На основе этих данных может быть построен набор кривых Безье, интерполирующих исходные структурные линии и удовлетворяющих требованиям о гладкости и уменьшении осцилляций. Этот набор может использоваться для визуализации, однако его трудно использовать для каких-либо дополнительных расчетов. Поэтому гладкую кривую необходимо аппроксимировать ломаной линией.

Обычно в подобных задачах аппроксимации кривых имеются 2 параметра-ограничения: предельное число вершин N ломаной и точность аппроксимации ε. Причем один из этих параметров фиксируется, а второй в процессе аппроксимации пытаются минимизировать. Если ломаная будет использоваться для визуализации, то параметр ε фиксируется и подбирается таким образом, чтобы ломаная являлась визуально гладкой. Однако в нашем случае расчетные структурные линии будут использоваться для построения ЦМР. Увеличение числа узлов не оказывает существенного влияния на качество ЦМР, но повышает временную и емкостную сложность задачи ее построения. Поэтому при аппроксимации нужно ограничивать число узлов.

В [1] предлагается алгоритм интерполяции кривой с n узлами ломаной с предельным числом вершин $N > n$, которая будет визуально настолько близка к гладкой линии, насколько это позволяют значение N и точность интерполяции ε. Используя этот алгоритм всегда можно по набору кривых Безье рассчитать аппроксимирующие ломаные с заданным предельным числом вершин.

На рис. 19 приведены для сравнения исходные изолинии одного участка местности и построенные по ним гладкие коридорные изолинии.

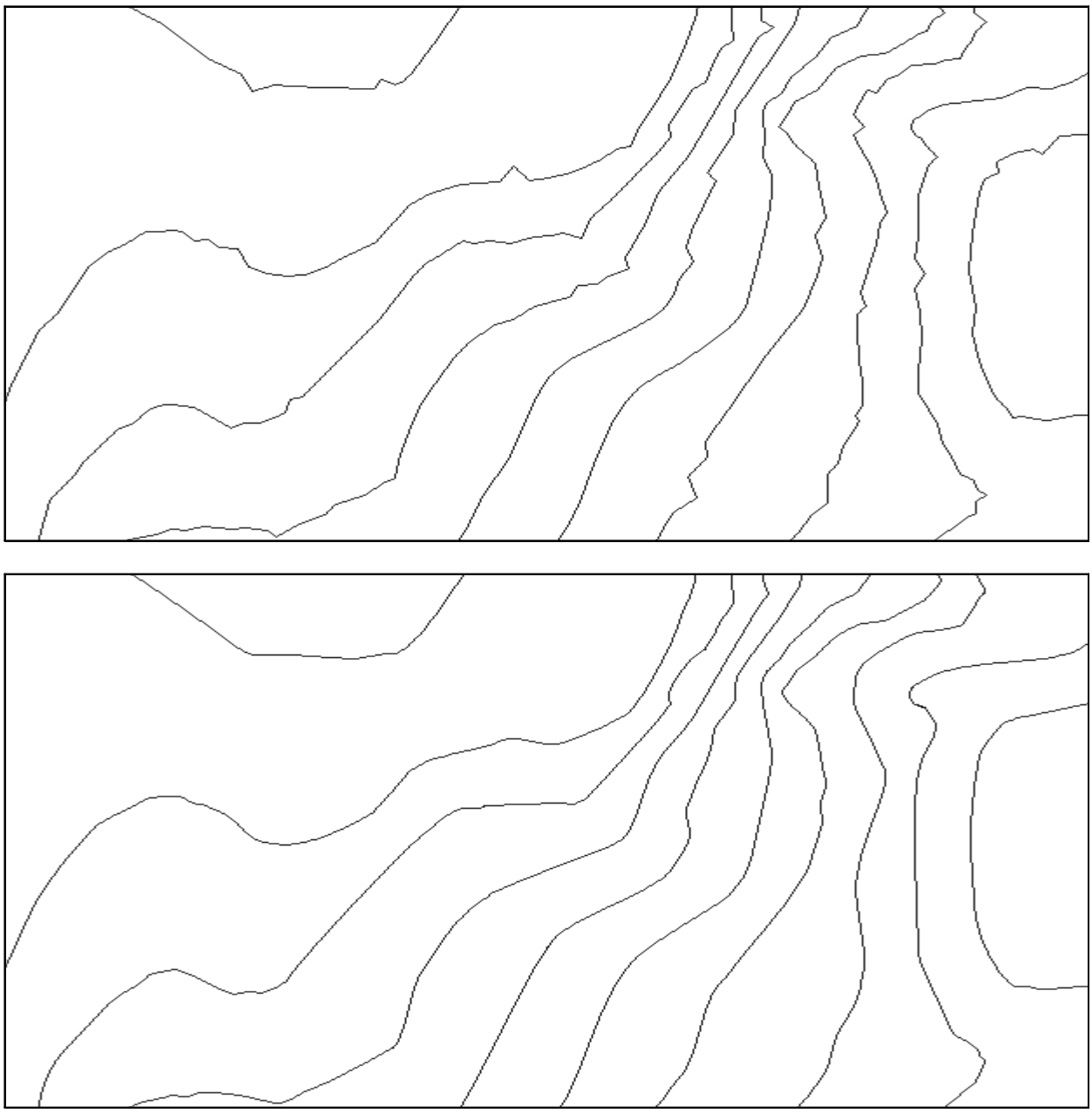


Рис. 19. Исходные и преобразованные изолинии для одного участка рельефа

Заключение

Настоящая дипломная работа посвящена созданию программного обеспечения для обработки информации о рельефе местности. Основная цель работы – обеспечение автоматизированной предобработки векторизованных структурных линий для получения высококачественной цифровой модели рельефа. Основные результаты дипломной работы:

1. Реализован алгоритм триангуляции Делоне с ограничениями, позволяющий структурировать исходные данные.
2. Исследован алгоритм расчета коридоров для структурных линий рельефа, предложено и реализовано 2 модификации данного алгоритма. Первая основана на формировании новых границ при последовательном прохождении вдоль границ коридора, вторая – на предварительном построении триангуляции с сильными ограничениями.
3. Реализован алгоритм расчета ломаной минимальной длины в коридоре, основанный на анализе текущего сектора видимости внутри коридора.
4. Реализован алгоритм сглаживания ломаной минимальной длины кривой Безье, целиком лежащей внутри коридора.

Список использованных источников

1. Костюк Ю.Л., Фукс А.Л. Предварительная обработка исходных данных для построения цифровой модели рельефа местности // Вестник ТГУ, 2003. №280. С. 281-285.
2. Костюк Ю.Л., Фукс А.Л. Построение цифровой модели рельефа местности на основе структурных линий и высотных отметок // Вестник ТГУ, 2003. №280. С. 286-289.
3. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. Пер. с англ. – М.: Мир, 1989. – 478 с.
4. Роджерс Д., Адамс Дж. Математические основы машинной графики: Пер. с англ. – М.: Мир, 2001. – 604 с.
5. Скворцов А.В. Триангуляция Делоне и ее применение. – Томск: Изд-во Том. ун-та, 2002. – 128 с.
6. Фукс А.Л. Разработка и исследование алгоритмов интерполяции однозначных поверхностей и их использование при построении цифровых моделей рельефа: Диссер. на соискание уч. степ. к.т.н. Томск, ТГУ, 2001.

Приложение А. Руководство программиста

Набор классов, осуществляющий предобработку исходных данных и построение ЦМР

Данный набор классов реализован в виде заголовочных файлов (*.h) и файлов с исходными текстами (*.cpp), написанными на Visual C++ с использованием STL – стандартной библиотеки шаблонов C++. Для разработки использовалась среда Microsoft Visual Studio 6.0. Для подключения классов к существующему проекту необходимо включить в свой код заголовочный файл demlinear.h.

Базовыми классами для работы с исходными данными и цифровой моделью рельефа являются классы CDEMData и CDEMLinear.

Класс CDEMData объединяет основные наборы, представляющие объекты обработки программ построения ЦМР: набор точек, набор наборов линий всех используемых типов, набор ограничений. CDEMData содержит также наборы информационных строк для входных и выходных файлов и фабрику объектов для работы с файлами заданных типов.

Данный класс объявлен в заголовочном файле demdata.h и реализован в файлах demdata.cpp, demtopo.cpp и demshapedata.cpp. CDEMData содержит следующие члены, объявленные с уровнем доступа protected:

- CPointSet m_Points – набор точек
- CLineMap m_Lines – набор наборов линий
- CRestrictionMap m_Restrictions – набор ограничений
- CFileInfoSet m_InputFiles – набор информационных строк входных файлов
- CFileInfoSet m_OutputFiles – набор информационных строк выходных файлов
- CDEMLoader *m_DEMLoader – фабрика объектов CxxxLoader
- CRegion m_Region – область определения
- double m_Zmin, m_Zmax – диапазон значений Z
- double m_XYTolerance – точность задания X и Y
- double m_ZTolerance – точность задания Z
- int m_InputPoints – число исходных (введенных) точек
- int m_BorderLength – заданное количество граничных точек, которые нужно добавить к имеющимся (4, 8 или 16)
- int m_BorderCreated – рассчитанное количество дополнительных граничных точек (4, 8 или 16)
- vector<int> m_PntOrder – рабочий массив – порядок выбора точек

Методы класса CDEMData, кроме стандартных конструктора и деструктора, по предназначению можно разделить на следующие группы:

- выделение памяти для наборов и инициализация
- задание и получение допусков для значений координат
- расчет области определения, создание новых точек и линий
- добавление объектов к наборам

- доступ к наборам по ссылке, получение и установка значений ограничений
- очистка наборов
- ввод и вывод информации
- проверка топологической корректности данных
- проверки взаимного расположения точек и линий

Методы для выделения памяти под наборы и инициализации:

– int AllocPoints(int inputpnt, int bordpnt = 0) – выделение памяти под набор точек

– int AllocInput(int filnum) – выделение памяти под набор информационных строк входных файлов

– int AllocOutput(int filnum) – выделение памяти под набор информационных строк выходных файлов

– void InitRestrictions() – инициализация ограничений

Методы для задания и получение допусков для значений координат:

– void SetXYTolerance(double xytol) и double GetXYTolerance() – установка и получение значения m_XYTolerance

– void SetZTolerance(double ztol) и double GetZTolerance() – установка и получение значения m_ZTolerance

Методы для расчета области определения и создания новых точек и линий:

– int CreateRegionBorder() – создание границы

– C3DPoint* CreatePoint(double x, double y, double z, unsigned char type = 0, unsigned char fnum = 0, char *attr = NULL) – создание точки

– CLine* CreateLine(int size, unsigned char type = 0, unsigned char fnum = 0, char *attr = NULL) – создание структурной линии

Методы для добавления объектов к наборам:

– int Append(C3DPoint *pnt) – добавление точки

– int Append(CLine *lin) – добавление линии

– int AppendInput(CFileInfo *fin) – добавление информационной строки входного файла

– int AppendOutput(CFileInfo *fin) – добавление информационной строки выходного файла

Методы для доступа к наборам, получения и установки значений ограничений:

– CLineSet &LinesByType(unsigned char type) – доступ к набору линий различных типов

– CPointSet &Points() – доступ к набору точек

– __declspec(property(get=GetRestr, put=PutRestr)) double Restrict[] – свойство для получения и установки значения ограничения

– CFileInfo *GetInput(int nNumber) – доступ к информационной строке входного файла по номеру в наборе

Методы очистки наборов:

– void ClearLines() – полная очистка набора линий
– void ClearLines(unsigned char type) – очистка набора линий заданного типа
– void ClearRestrictions() – очистка набора ограничений
– void ClearInput() – очистка набора описаний входных файлов
– void ClearInput(int nIndex) – удаление описания входного файла по заданному номеру
– void ClearOutput() – очистка набора описаний выходных файлов
– void ClearLinesPoints() – очистка наборов линий и точек

Методы для ввода и вывода информации о ЦМР:

– int EvaluateInput() – проверка входных файлов, возвращает общее число точек, содержащихся в файлах; используется перед выделением памяти под набор точек
– int LoadData() – чтение информации из файлов входного набора, возвращает число обработанных файлов
– int SaveData() – запись информации в файлы выходного набора, возвращает число обработанных файлов
– int PreLoadShape(CString FileName) – формирует описание shape-файла, расположенного по адресу, указанному в параметре, и добавляет его к набору описаний входных файлов

Методы для проверки топологической корректности данных:

– int InitialLineTest() – проверка замкнутости, невырожденности, правильности задания номеров и совпадения узлов линий
– int JoinCoincidedPoints() – попарная проверка и соединение совпадающих (в пределах допуска) точек без проверки совпадения высот
– int DeleteEmptySegments() – удаление вырожденных отрезков линий, заданных списками в векторе m_PntOrder
– int CheckPointsOnSegments() – проверка попадания точек на отрезки линий (в пределах допуска) с требуемой корректировкой линий, заданных списками в m_PntOrder
– int CheckSegmentsIntersections() – проверка пересечения отрезков линий с добавлением новых точек в отрезки и требуемой корректировкой линий, заданных списками m_PntOrder
– int LinesToLists(), int ListsToLines(), void ClearLists() – вспомогательные методы, предназначенные для преобразования линий в списки вершин в m_PntOrder, и обратно, а также для очистки m_PntOrder

Методы проверки взаимного расположения точек и линий:

– double SegmLen_2(double xa, double ya, double xb, double yb) – квадрат длины отрезка ab
– int PointsCoincide(C3DPoint *a, C3DPoint *b) – проверка совпадения двух точек (в пределах допуска по XY)
– int PointOnSegm(C3DPoint *p, C3DPoint *a, C3DPoint *b) – проверка варианта принадлежности точки p отрезку ab (в пределах допуска по XY)

– int LineSectSegm(C3DPoint *a, C3DPoint *b, C3DPoint *c, C3DPoint *d) – проверка пересечения прямой ab и отрезка cd

– int InterSection(C3DPoint *a, C3DPoint *b, C3DPoint *c, C3DPoint *d, double &xx, double &yy, double &zab, double &zcd) – определение варианта и расчет точки пересечения отрезков ab и cd

Класс CDEMLinear является порожденным от CDEMData и объединяет основные наборы из CDEMData (точек, линий, ограничений) и массив треугольников, а также методы предобработки исходных данных, построения и обработки кусочно-линейной ЦМР (заданной набором пространственных треугольников). Данный класс объявлен в заголовочном файле demlinear.h и реализован в файлах demlinear.cpp, preproc.cpp, triangul.cpp и isolines.cpp.

Кроме членов, унаследованных от CDEMData класс CDEMLinear содержит:

– vector<CTrian> m_Triangles – массив треугольников триангуляции

– CDEMLinear *m_Corridors – объединяет точки и линии, задающие границы коридоров, а также ломаные минимальной длины

– vector <BezierLine *> BezierLines – хранит промежуточные точки характеристических четырехугольников для построения кривых Безье

– vector <TrianDesc *> TrianNALines – хранит информацию о видах ребер в треугольниках: недопустимые, отрезки изолиний и все остальные. Используется для учета сильных ограничений

– int m_Preprocessed – счетчик числа предобработанных точек (без учета дополнительных граничных точек) перед построением триангуляции Делоне

– int m_Triangulated – счетчик числа обработанных точек при построении ЦМР

– int m_TrianVertices – хранит число вершин триангуляции

– double DeltaZ – величина отклонения во внутренних точках треугольников при построении ЦМР, учитывающей сильные ограничения

– char CorridorMethod – задает используемый метод модификации границ коридора (0 – для алгоритма, основанного на последовательном прохождении вдоль границ; 1 – для использования триангуляции с сильными ограничениями)

Основные методы класса CDEMLinear выполняют задачу предобработки исходных наборов структурных линий, а также связанные с ней задачи: построение триангуляции Делоне и кусочно-линейной ЦМР, учет слабых и сильных ограничений.

Для построения триангуляции Делоне по набору исходных точек используются следующие методы:

– int DelaunayTriangulation() – основная функция, выполняющая построение триангуляции по набору точек

– int AddDelaunayVertex(int newpnt, int begtri, vector<int> &TriQue) – вспомогательная функция, вызываемая из основной, производит добавление новой вершины к триангуляции Делоне

Также перед вызовом основной функции для построения триангуляции должны вызываться методы:

– int PointPreprocessing() – предварительная обработка набора точек для определения порядка выбора точек при последующей триангуляции (порядок выбора сохраняется в векторе m_PntOrder)

– int AllocTriangles() – выделение памяти для набора треугольников

– int InitialTriangulation() – построение начальной триангуляции по дополнительным 4, 8 или 16 граничным точкам

Перестроение триангуляции Делоне для учета ограничений, задаваемых структурными линиями, осуществляет функция int DelaunayTriangulationWithRestrictions().

Метод int AdjustDEM() используется для преобразования ЦМР с целью устранения недопустимых ребер (учета сильных ограничений). В нем используются также вызовы следующих вспомогательных процедур:

– boolean SearchNeighborhood (int nStartTrn) – проводит поиск негоризонтального треугольника в окрестности треугольника с номером nStartTrn и определение знака приращения по высоте во внутренних точках nStartTrn. Возвращает TRUE, если знак приращения положительный, и FALSE, если отрицательный

– int ProcessChain(int nStartTrn, vector<SearchTrian *> &Chain) – рекурсивная процедура, обрабатывающая цепочку недопустимых ребер. Вызывает FindChain для поиска цепочки.

– int FindChain (int nStartTrn, vector<SearchTrian *> &SearchStack) – проводит поиск цепочки треугольников с недопустимыми ребрами, начинающейся с nStartTrn и сохраняет ее в массиве SearchStack

– int AddDelaunayVertexWithStrongRestrictions(int newpnt, int begtri, vector<int> &TriQue) – производит добавление новой вершины к триангуляции Делоне, при этом перестраивает только недопустимые ребра

Следующие методы осуществляют предобработку исходных изолиний с целью уменьшения числа точек и осцилляций, и сглаживания:

– int SetCorridorBuildMethod(char BuildMethod) – устанавливает значение переменной CorridorMethod

– int BuildCorridors() – строит коридоры для расчетных линий

– int AdjustCorridors() – модифицирует коридоры таким образом, чтобы было взаимно однозначное соответствие между левой и правой границами коридоров

– int BuildNewLines() – строит в каждом коридоре ломаную минимальной длины, затем на основе этой ломаной – кривую Безье, целиком помещающуюся в коридоре

– int CreatePolyLineWithMinLength (CLine *pRBorder, CLine *pLBorder, CPointSet *Points, int nStartPoint, int nEndPoint, CLine *pLine, vector<int> *pCorIndex) – выполняет построение ломаной минимальной длине в заданном коридоре и с заданными начальной и конечной точками

– double TestCorridorCurve(Point2D *pPoly, CLine *pRBorder, CLine *pLBorder, int nStart, int nEnd) – проверяет, вписывается ли кубическая кривая Безье в заданный участок коридора, если нет, то возвращаемое значение – коэффициент уменьшения модулей производных в конечных точках кривой

Также для предобработки используются следующие вспомогательные функции:

– double SectMultCoeff(double aX, double aY, double bX, double bY, double cX, double cY, double dX, double dY) – рассчитывает коэффициент сжатия кривой Безье путем анализа взаимного расположения отрезков ab (от хорды до кривой) и cd (отрезок границы коридора). Возвращает минимально необходимый для данной пары коэффициент (<= 1)

- boolean DelaunayCondition(C3DPoint *p, C3DPoint *a, C3DPoint *b, C3DPoint *c) – проверка условия Делоне для треугольников PAB и ABC
- boolean PointOnTheLeft(CPointSet *Points, int P, int A, int B) и boolean PointOnTheLeft(C3DPoint *P, C3DPoint *A, C3DPoint *B) – возвращают TRUE, если точка P лежит слева от прямой AB, иначе FALSE
- boolean PointOnTheRight(C3DPoint *p, C3DPoint *a, C3DPoint *b) – возвращает TRUE, если точка P лежит справа от прямой AB, иначе FALSE
- boolean EqualPoints(C3DPoint *a, C3DPoint *b) – проверяет 2 точки на совпадение в пределах допуска
- double CalcLBAngle(C3DPoint *p, C3DPoint *a, C3DPoint *b) – вычисляет величину угла PAB в радианах
- double CalcRBAngle(C3DPoint *p, C3DPoint *a, C3DPoint *b) – вычисляет величину угла BAP в радианах
- double ScaleProd(C3DPoint *a, C3DPoint *b, C3DPoint *c) – вычисляет скалярное произведение векторов BA и BC
- double VectorProd(C3DPoint *a, C3DPoint *b, C3DPoint *c) – вычисляет Z-координату векторного произведения векторов BA и BC

Остальные методы класса CDEMLinear предназначены для создания и добавления треугольников к имеющемуся набору, а также свойства и функции для доступа ко всему набору треугольников по ссылке и доступа к вершинам отдельных треугольников и треугольникам, смежным с ними, причем доступ возможен как по имени (A, B, C для вершин, AB, BC, CA – для треугольников, смежных по соответствующим ребрам), так и по номеру (нумерация вершин идет против часовой стрелки и начинается с вершины A, имеющей номер 0).

Класс CRegion предназначен для хранения прямоугольных областей на плоскости XOY. Он объявлен в файле region.h и реализован в файлах region.cpp и region_iso.cpp.

Для организации наборов объектов (точек, линий, описаний файлов) и проведения операций с ними используется класс-шаблон CSet. Набор хранимых объектов объявлен как vector<T*> m_Set и имеет уровень доступа private. В качестве элементов набора используются указатели на объекты. В файле demdata.h объявлено несколько типов наборов:

```
typedef CSet<C3DPoint> CPointSet;

typedef CSet<CLine> CLineSet;

typedef CSet<CFileInfo> CFileInfoSet;

typedef map<unsigned char, CLineSet> CLineMap;

typedef map<int, double> CRestrictionMap;
```

Все остальные классы и структуры, кроме загрузчиков, используются в классах CDEMData и CDEMLinear как составные элементы наборов. Такими элементами могут быть точки, линии, треугольники, а также описания входных или выходных файлов (информационные строки).

Класс C3DPoint – базовый класс для точек поверхности. Он объявлен в заголовочном файле point.h и реализован в файле point.cpp. Класс содержит координаты X,Y,Z, тип точки Type, указатель на строку атрибутов AttrStr, номер файла в списке ввода/вывода FileNum и 2 значения, определяющие связи данной точки с другими точками и треуголь-

никами ЦМР: CoinVert – номер вершины триангуляции (в наборе точек), с которой совпадает данная точка, и NearNeib – номер точки, ближайшей к данной и включаемой в триангуляцию раньше нее (до включения точки в триангуляцию), или номер одного из треугольников, вершиной которого является данная точка или совпадающая с ней (после включения и при любом перестроении триангуляции).

Класс CLine – базовый класс для линий поверхности. Он объявлен в заголовочном файле line.h и реализован в файлах line.cpp и shapeline.cpp. Класс CLine содержит указатель на связанный с линией набор точек и массив порядковых номеров вершин линии в этом наборе, а также тип линии Type, номер файла в списке ввода/вывода FileNum и указатель на строку атрибутов линии AttrStr (может быть NULL).

Класс CTrian представляет треугольники. Он объявлен в заголовочном файле trian.h и реализован в файле trian.cpp. Любой треугольник содержит поля:

- vA, vB и vC – порядковые номера точек в наборе точек, с которым связан набор треугольников, содержащий данный треугольник (обход вершин производится против часовой стрелки);
- sAB, sBC, sCA – номера в наборе треугольников (от 0) треугольников, смежных с данным по ребрам (vA,vB), (vB,vC) и (vC,vA), соответственно, или -1, если ребро является граничным в триангуляции;
- Flag – флаг состояния треугольника (T_DELETED, если треугольник считается удаленным).

CFileInfo – базовый класс информационных строк файлов. Он объявлен в заголовочном файле fileinfo.h и реализован в файле fileinfo.cpp. Содержит имя файла FileName, область Region для выбора элементов из CDEMData при выводе и тип файла FileType.

Типы файлов объявлены в заголовочном файле fileinfo.h. В настоящей работе была реализована поддержка только одного типа – shape-файлы ArcInfo. Информационные строки для shape-файлов представлены классом CShapeInfo. Этот класс порожден от CFileInfo, объявлен в файле fileinfo.h и реализован в файле fileinfo.cpp. Кроме базовой информации, содержит также:

- тип shape-файла nShapeType (точки, линии и т.д)
- заголовок связанного dbf-файла DBFHeader
- указатель на массив описаний полей dbf-файла pFields, объявлен как DBFieldDesc *pFields
- количество полей в dbf-файле nNumFields
- номер атрибута высоты nHeightNum в массиве по адресу pFields

Для загрузки данных для построения ЦМР из входных файлов и сохранения в выходных файлах используются загрузчики. Для поддержки загрузки и сохранения данных предусмотрено 3 класса: CLoader, CShapeLoader и CDEMLoader, объявленных в файле loader.h и реализованных в файле loader.cpp.

Например, при загрузке данных в методе LoadData класса CDEMData в цикле перебираются все информационные строки входных файлов и для каждого вида файлов создается загрузчик соответствующего типа, который и загружает данные в методе Load.

Класс CLoader является абстрактным базовым классом для загрузчиков всех типов. Содержит указатели на связанный с ним объект CDEMData – m_DEMData и на информа-

ционную строку – m_FileInfo, а также номер файла в наборе ввода/вывода m_FileNum. Не имеет реализации, а только декларирует виртуальные методы Load и Save.

Класс CShapeLoader порожден от класса CLoader и представляет загрузчик для shape-файлов ArcInfo. Его методы Load и Save реализованы в файле shape.cpp.

Класс CDEMLoader используется при создании и освобождении объектов-загрузчиков для всех используемых типов файлов.

На различных этапах обработки данных о рельефе используются также следующие структуры:

- Point2D – задает точку или вектор на плоскости *XOY*
- BezierPoints – задает пару промежуточных точек для характеристического четырехугольника кубической кривой Безье. Тип BezierLine объединяет структуры BezierPoints в массив для всех отрезков ломаной
- CrossPointDesc – описывает точку пересечения структурной линии и ребра триангуляции. Используется при включении в триангуляцию отрезков структурных линий
- DBFieldDesc – описывает одно поле dbf-файла
- IsoPointDesc – для точки, являющейся узлом структурной линии, содержит номер этой линии в общем наборе линий и индекс данной точки в этой линии. Если точка не является узлом структурной линии, то значение члена структуры nLineNum равно –1. Структура используется для выделения недопустимых ребер в ЦМР
- SearchTrian – описывает треугольник в цепочке треугольников с недопустимыми ребрами. Используется при поиске и обработке данных цепочек
- TrianDesc – для каждого треугольника описывает тип его ребер (недопустимые, принадлежащие изолинии и т.д.), а также содержит количество недопустимых ребер и флаг, указывающий, был ли данный треугольник пройден при поиске по структуре смежных треугольников. Используется на этапе преобразования ЦМР для учета сильных ограничений

Приложение Б. Руководство пользователя

Алгоритмы предварительной обработки структурных линий реализованы на Visual C++ в рамках подсистемы построения цифровой модели рельефа. Базовыми классами для работы с исходными данными и цифровой моделью рельефа являются классы CDEMData и CDEMLinear.

Набор классов реализован в виде заголовочных файлов (*.h) и файлов с исходными текстами (*.cpp). Для подключения классов к существующему проекту необходимо включить в свой код заголовочный файл demlinear.h.

На вход алгоритма предобработки подается набор высотных отметок и структурных линий, считанных в память из заданных входных файлов, представляющих темы ArcView в shape-формате (shape-файлов), значения высот берутся из связанной с каждой темой атрибутной таблицы в формате dBase.

Пример работы с функциями классов CDEMData и CDEMLinear для предобработки изолиний без проверок значений, возвращаемых функциями

```
#include “demlinear.h”

CDEMLinear Dem;

// Функции-члены CDEMData для получения и проверки исходных данных ЦМР:
// задание входных файлов линий и точек
CShapeInfo *pInputShape;
Dem.PreLoadShape(“points.shp”);
pInputShape = Dem.GetInput(0);
strcpy(pShape->HeightName, “HEIGHT”);
Dem.PreLoadShape(“isolines.shp”);
pInputShape = Dem.GetInput(1);
strcpy(pShape->HeightName, “LEVEL”);
// вычисление используемого числа точек в объекте Dem и резервирование памяти под их
// набор для эффективного использования динамических массивов
int n = Dem.EvaluateInput();          // расчет общего числа точек
Dem.AllocPoints(n);                   // резервирование памяти для n*1.2 точек
Dem.LoadData();                       // ввод информации из входных файлов

// Несколько необходимых действий, которые нужно выполнять
// после ввода всех исходных точек и линий:
```

```
// установка точности задания координат точек
Dem.SetXYTolerance(0.3);
Dem.SetZTolerance(0.1);
// расчет дополнительных граничных точек
Dem.CreateRegionBorder();
// проверка правильности задания и необходимая корректировка исходных линий
Dem.InitialLineTest();

// Функции-члены CDEMLinear, необходимые для построения триангуляции Делоне:
// предварительная обработка набора точек для определения порядка выбора точек при
// последующей триангуляции
Dem.PointPreprocessing();
// выделение памяти для массива треугольников
Dem.AllocTriangles();
// построение начальной триангуляции по дополнительным 4, 8 или 16 граничным точкам
Dem.InitialTriangulation();
// построение триангуляции Делоне набора точек в Dem
Dem.DelaunayTriangulation();

// Функции-члены CDEMLinear, необходимые для предобработки изолиний:
// построение триангуляции Делоне с ограничениями
Dem.DelaunayTriangulationWithRestrictions();
// выделение коридоров на треугольной сетке
Dem.SetCorridorBuildMethod(0);
Dem.BuildCorridors();
// расчет новых линий в коридорах
Dem.BuildNewLines();

// Вывод информации в выходной набор shape-файлов
// задание выходного файла для расчетных линий
CShapeInfo *pOutputShape = new CShapeInfo("newlines.shp");
pOutputShape->nShapeType = LINES;
Dem.AppendOutput(pOutputShape);

// сохранение полученных данных в выходной shape-файл
Dem.SaveData();
```