Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

**УТВЕРЖДАЮ**
Заведующий кафедрой
_____ / В.В. Шайдуров

«24» июня 2020 г.

# БАКАЛАВРСКАЯ РАБОТА

**Направление** 02.03.01 Математика и компьютерные науки

## ПРИМЕНЕНИЕ ТРИАНГУЛЯЦИОННЫХ НЕРЕГУЛЯРНЫХ СЕТОК ДЛЯ СОЗДАНИЯ МОДЕЛЕЙ РЕЛЬЕФА

Научный руководитель
кандидат физико-математических наук, доцент _____ 23.06.20 / Е.В. Кучунова

Выпускник _____ 23.06.20 / Ю.В. Лобко

Красноярск 2020

# РЕФЕРАТ

Бакалаврская работа по теме «Применение триангуляционных нерегулярных сеток для создания моделей рельефа» содержит 30 страниц текста, 6 приложений, 7 использованных источников.

ЦИФРОВАЯ МОДЕЛЬ РЕЛЬЕФА, МОДЕЛЬ TIN, РАСТРОВОЕ ПРЕДСТАВЛЕНИЕ МОДЕЛИ, ТРИАНГУЛЯЦИЯ ДЕЛОНЕ, ПОЛЬСКИЙ ФОРМАТ.

Цель работы – реализовать вычислительный алгоритм с использованием нерегулярной триангуляционной сетки для создания цифровой модели местности по векторным картам в польском формате. Исходными данными являются текстовые файлы, содержащие информацию об объектах местности, в том числе наборы изолиний. Каждая из изолиний представляет собой набор координат с одинаковой высотой. Строится цифровая модель с помощью алгоритма триангуляции Делоне и волнового итерационного алгоритма.

Для изучения были выбраны растровая модель представления поверхности и модель TIN (Triangulated Irregular Network), как наиболее эксплуатируемые. Обе модели оказались по-своему интересными и на их основе были разработаны программы для практических примеров – построения горного хребта Борус, Кутурчин, Ергаки. Для реализации описанных примеров написаны программы на языке C++ с использованием графической библиотеки OpenGL.

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ ..... 3
1 Обзор цифровых моделей рельефа ..... 5
1.1 Растровое представление поверхности ..... 5
1.2 Определение модели TIN ..... 7
2 Получение данных об исходном наборе изолиний ..... 10
3 Исследование алгоритмов триангуляции Делоне ..... 12
3.1 Задача построения триангуляции Делоне ..... 12
3.2 Проверка условия Делоне через уравнение описанной окружности ..... 13
3.3 Обзор основных алгоритмов построения триангуляции Делоне ..... 14
4 Алгоритмы построения поверхности рельефа местности ..... 15
4.1 Постановка задачи ..... 15
4.2 Вычислительный алгоритм №1 ..... 16
4.3 Вычислительный алгоритм №2 ..... 19
5 Этапы построения поверхности ..... 24
6 Результаты тестовых экспериментов ..... 25
6.1 Построение поверхности рельефа для тестового набора изолиний ..... 25
6.2 Построение поверхности рельефа горного хребта Борус ..... 27
6.3 Построение поверхности рельефа горного хребта Кутурчин ..... 29
6.4 Построение поверхности рельефа горного хребта Ергаки ..... 29
ЗАКЛЮЧЕНИЕ ..... 31
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ ..... 32
ПРИЛОЖЕНИЯ ..... 33

# ВВЕДЕНИЕ

С появлением первых компьютеров многократно возросли возможности хранения информации и практически сразу же были предприняты попытки для реализации карт на компьютерах с целью связать каждый объект карты с его описанием, внесенным в базу данных. Основными координатами для таких ГИС – моделей, кроме привычной широты и долготы служили также данные о высоте (глубине). Исследуемая поверхность задается набором точек большого объема и в связи с доступностью быстрой компьютерной обработки громадных массивов данных, становится реально выполнимой задача создания максимально точной цифровой модели рельефа (ЦМР) [6].

Цифровую модель очень удобно использовать, поскольку современные технологии трехмерной визуализации дают возможность даже профессионально неподготовленным людям «полетать» над местностью и «взглянуть» на рельеф местности из любой точки пространства и под любым углом. Например, для того чтобы получить информацию о угле наклона, высоте, поперечном сечении или просто проложить удобный маршрут для передвижения. То есть ЦМР дает возможность получить достаточно полную картину представления о рельефе.

В начале 2000 годов появился новый формат представления геоданных, который имеет простую и хорошо документированную структуру. Данный формат получил название «польский формат» или формат MP, так как изначально был разработан польскими программистами. Польский формат – это формат для векторных карт, предназначенный для хранения данных об объектах местности в текстовом формате и применяемый в качестве промежуточного формата при конвертировании или редактировании электронных карт.

Данные, представленные в польском формате, легко поддаются редактированию и сохраняют совместимость с программами, работающими со стандартной структурой польского формата. То есть такой текстовый формат легко расширять, дополнять своими объектами или полями объектов, что делает

его удобным при разработке картографического ПО со специфическими требованиями.

Цель работы: реализовать вычислительный алгоритм с использованием нерегулярной триангуляционной сетки для создания цифровой модели местности по векторным картам в польском формате. Исходными данными являются текстовые файлы, содержащие информацию об объектах местности, в том числе наборы изолиний. Каждая из изолиний представляет собой набор координат с одинаковой высотой.

Цифровая модель рельефа строится с применением треугольной или прямоугольной сетки по проекциям исходных точек на плоскости *XOY*, в узлах которой содержатся значения высот. Благодаря треугольной сетке, в получаемой цифровой модели сохраняются исходные точки и учитываются ограничения, налагаемые структурными линиями поверхности, чего не происходит при использовании прямоугольной.

Актуальность работы обусловлена развитием геоинформационных технологий и увеличением потребностей в географических исследованиях, с использованием данных рельефа в цифровой форме для решения различных задач. Также необходимо повышать качество и эффективность методов создания ЦМР для обеспечения достоверности готовых моделей.

Для достижения поставленной цели в работе требовалось решить следующие задачи:

1. разработать вычислительный алгоритм построения поверхности рельефа;
2. реализовать алгоритм на языке C++ с использованием графической библиотеки OpenGL;
3. провести тестовые эксперименты с применением реализованного алгоритма.

# 1 Обзор цифровых моделей рельефа

ЦМР (Цифровая Модель Рельефа) – цифровое представление трехмерных пространственных объектов в виде трехмерных данных с координатами $X, Y, Z,$ расположенными:

- вдоль изолиний (векторная модель);
- в узлах любой регулярной сетки с образованием матрицы высот (растровая модель);
- в узлах нерегулярной треугольной сетки (TIN-модель).

Данные виды моделей ЦМР приведены на рисунке 1.

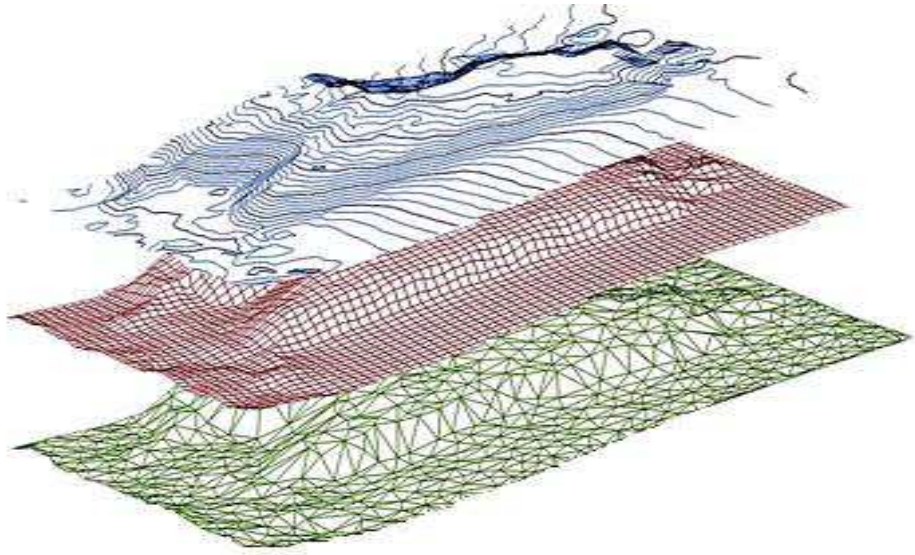

Рисунок 1 – Виды ЦМР: векторная модель, растровая модель и модель TIN

Растровая модель и модель TIN считаются наиболее распространёнными способами построения ЦМР. Далее более подробно рассмотрим каждое из представлений ЦМР.

## 1.1 Растровое представление поверхности

Растровое представление – это графический формат представления поверхности в виде матрицы равномерно распределенных точек, характеризующихся своим параметром $Z$ (высота местности, количество особей и т.д.), то есть растровая модель использует регулярную сетку, состоящую из смежных не пересекающихся граней правильного n-угольника. Любой элемент растра имеет только одно значение плотности или цвета.

На рисунке 2 представлены виды регулярных сеток.

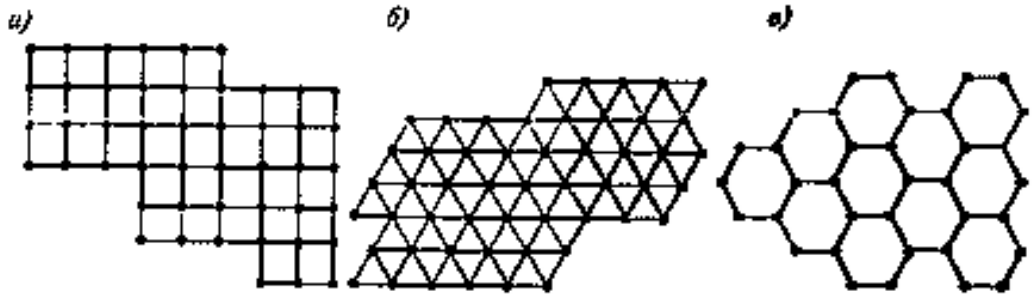


Рисунок 2 – Виды регулярных сеток: а – прямоугольная, б – треугольная, в – шестиугольная

Такие растровые изображения как изображение двумерной карты, сканированное изображение карты, либо космический или аэрофотоснимок натягиваются поверх модели рельефа методом текстурирования (рисунок 3).

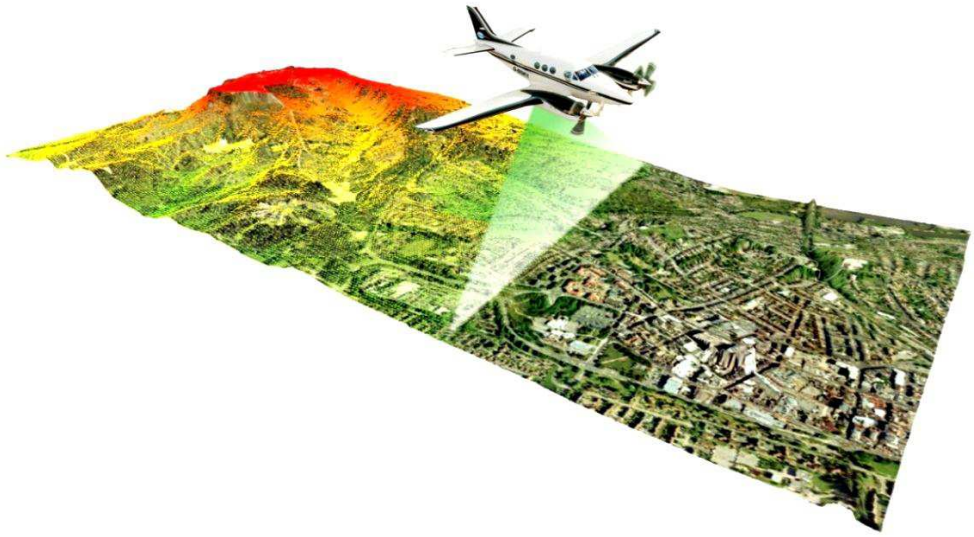

Рисунок 3 – Растровая модель рельефа

Различают "решеточную" и "ячеистую" модели (рисунок 4) в зависимости от способа вычисления высот между точками поверхности в пространстве.

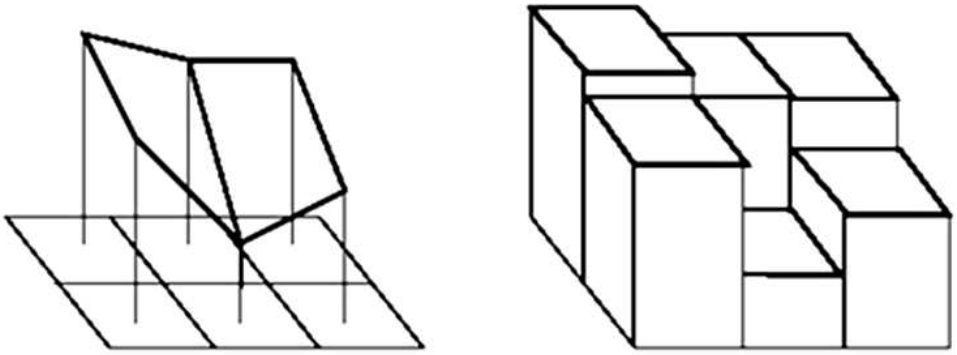

Рисунок 4 – Виды растровых моделей: "решеточная" и "ячеистая"

Модель слева «решеточная» представлена в виде значений, интерполируемых по значениям *Z* в нескольких соседних точках. Модель справа «ячеистая» и для нахождения высот используется несколько центральных точек ячеек со значениями *Z*, расположенных через равные горизонтальные интервалы.

Главным достоинством растрового представления поверхности является высокое быстродействие и простота обработки растровых данных. Также благодаря регулярной структуре удается избежать резких граней и выступов, то есть осуществляется «сглаживание» моделируемой поверхности.

Однако регулярная структура является причиной и для недостатков, например она не приспособлена к изменениям сложного рельефа, точные местоположения точек вершин, дна теряются. Также растровые изображения занимают, как правило, большие объемы памяти из-за большого количества разбиений. Растровые объекты сложно масштабировать: при приближении объекта становятся видны отдельные пиксели, пропадает гладкость, изображение становится зернистым.

### 1.2 Определение модели TIN

Модель TIN (Triangulation Irregular Network – триангуляционная нерегулярная сеть) географических объектов – модель поверхности в виде сети смежных не пересекающихся треугольных граней, определенная по узлам и ребрам, которые покрывают поверхность [7].

Данная модель широко известна и применима, так как передает рельеф местности наиболее точно. Точность обусловлена плотным «прилеганием» треугольников к моделируемой поверхности. Таким образом, построение ЦМР с использованием модели TIN сводится к созданию оптимальной нерегулярной сети треугольников, где треугольники «стремятся» к равносторонним, и любая точка двумерного пространства имеет только одну высотную координату.

На рисунке 5 приведен пример построения модели TIN.

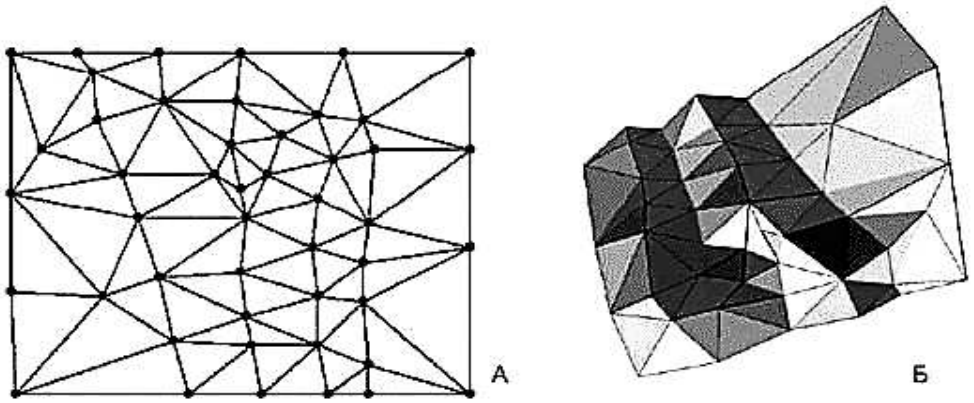


Рисунок 5 – TIN представление: А – план на плоскости (нерегулярная триангуляционная сеть); Б – 3D отображение

Модель TIN обладает некоторыми преимуществами в сравнении с растровыми моделями. Во-первых, расположение точек адаптировано под местность: на равнинных участках точки расположены реже, а на гористых – чаще. Выборочные точки соединяются отрезками, образующими треугольники, внутри которых поверхность представляется плоскостью. Структуры данных в TIN-моделях более компактны и экономичны: TIN-модели из сотен точек может соответствовать растровая DEM из десятков тысяч точек [2]. Несмотря на все достоинства модели, создание модели TIN требует решения ряда сложных задач, например: как располагать выборочные точки, как объединять точки в треугольники или как моделировать поверхность внутри треугольника.

На рисунке 6 продемонстрирован пример построенной модели TIN.

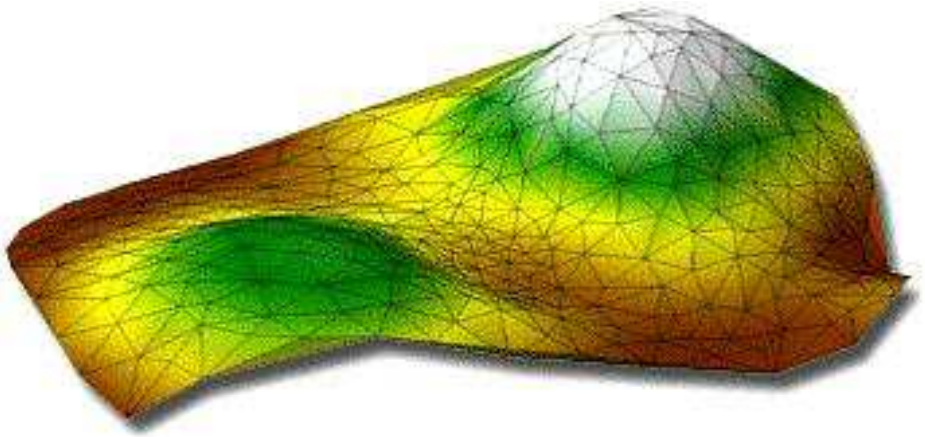

Рисунок 6 – Модель TIN

Если обобщать выше представленную информацию, то для растрового представления поверхности свойственен эффект размывания из-за чего с крупными масштабами применение ограничено. Модель TIN же дает более точное представление крупного масштаба сильнопересеченной местности, ведь точки располагают в узлах нерегулярной сетки так, чтобы лучшим образом обогнуть поверхность. Таким образом, если объемы занимаемой памяти и точность построения не главное, а особо важно время и общее представление о местности, то лучше использовать растровую модель, так как она проще и быстрее для реализации.

# 2 Получение данных об исходном наборе изолиний

В текстовом файле с расширением .mp хранятся данные об исходном наборе изолиний. Такой текстовый файл содержит информацию не только об изолиниях, но и обо всех объектах местности (реки, озера, ручьи, тропы, леса, здания или сооружения), которые задаются при помощи полилиний и полигонов, содержащих географические координаты (широту и долготу) множества точек.

Изолиния (линия одного уровня) – это условное обозначение на карте или чертеже в виде линии, в каждой точке которой значение исследуемой величины неизменно.

Полилиния – это незамкнутый набор соединенных между собой отрезков или дуг, воспринимаемый как единый объект.

Полигон – замкнутая полилиния.

На рисунке 7 приведен пример хранения изолинии в текстовом файле с расширением .mp.

```
[POLYLINE]
Type=0x20            - объект изолиния
Label=340            - уровень(высота) изолинии
Data=(52.84780,91.41503),(52.84752,91.41494),(52.84765,91.41484),(52.84778,91.41
487),(52.84786,91.41494)      - физические координаты (широта и долгота) набора точек
[END]
```

Рисунок 7 – Пример хранения изолиний в текстовом файле

Таким образом, из текстового файла считывается информация только об объекте изолиния и в ходе работы программы географические координаты (широта и долгота) переводятся в единую систему – метры (исходя из того, что высота в файле задана в метрах) с помощью простых математических выкладок.

Для того чтобы узнать сколько, километров заключено в одном градусе широты, нужно разделить длину меридиана, которая составляет 40000 км, на 360

$$\frac{40000\ \text{км}}{360} = 111{,}111\ \text{км} = 111 * 10^3\text{м}. \quad (1)$$

Аналогичным образом получаем, что на экваторе один градус долготы будет тоже равен 111,111 км, но при приближении к полюсам, длина градуса долготы будет уменьшаться, пока не станет равной нулю. Следовательно, перевод долготы в километры будет зависеть от широты местности. Тогда рассчитать ее можно будет умножив длину экватора на косинус угла, равного соответствующему градусу широты

$$111{,}111 \text{ км} * \cos(\chi) = 111 * 10^3 * \cos(\chi) \text{ м}. \quad (2)$$

Таким образом, в результате вычислений получим, что один градус широты равен $111 * 10^3$м, а один градус долготы $111 * 10^3$м *$\cos(\chi)$ м, где $\chi$ – соответствующий градус широты.

# 3 Исследование алгоритмов триангуляции Делоне

Наилучшей триангуляцией для моделирования рельефа является триангуляция Делоне, так как этот метод позволяет добиться наиболее точного соответствия полученной модели исходному множеству вершин по сравнению с другими методами. Также данный метод удовлетворяет основным свойствам ЦМР описанными О. Р. Мусиным и А. В. Скворцовым:

- триангуляция близка к равноугольной триангуляции;
- выполняется свойство максимальности минимального угла;
- выполняется свойство минимальности площади образуемой поверхности.

## 3.1 Задача построения триангуляции Делоне

Пусть на плоскости задано множество точек $\{S_i=(x_i,y_i)\}$ используя которое необходимо получить триангуляцию Делоне.

Триангуляцией будем называть планарный граф, все внутренние области которого являются треугольниками.

Триангуляцией Делоне будем называть такую триангуляцию для заданного множества точек $S_i$ на плоскости, при которой для любого треугольника все точки из данного множества $S_i$ за исключением точек, являющихся его вершинами, будут лежать вне окружности, описанной вокруг этого треугольника.

Другими словами, точки нужно соединить в треугольники так, чтобы выполнялось условие Делоне.

Условие Делоне: внутрь окружности, описанной вокруг любого построенного треугольника, не должна, попадать никакая из точек триангуляции. На рисунке 8 представлены примеры, где условие Делоне выполняется и нет.

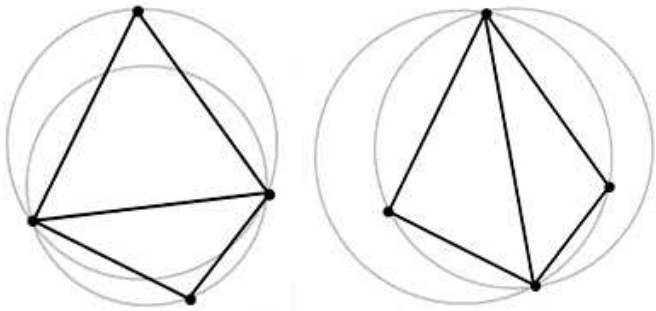

Рисунок 8 – Условие Делоне выполняется (слева) и условие Делоне не выполняется (справа)

На рисунке 9 приведен пример треугольной сетки, удовлетворяющей условию Делоне.

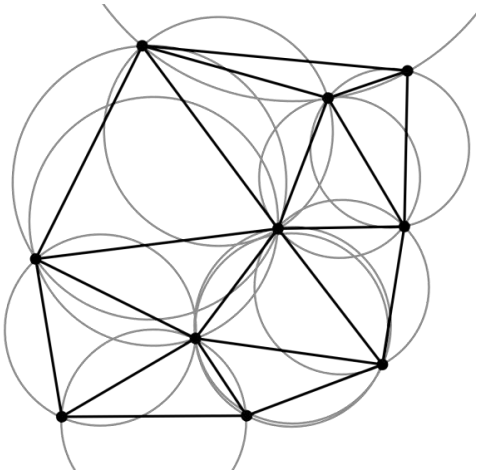

Рисунок 9 – Треугольная сетка, удовлетворяющая условию Делоне

### 3.2 Проверка условия Делоне через уравнение описанной окружности

Проверка условия Делоне осуществляется через уравнение описанной окружности. Уравнение окружности, проходящей через три точки ($x_1$,$y_1$), ($x_2$,$y_2$), ($x_3$,$y_3$) имеет вид

$$(x^2 + y^2) * a - x * b + y * c - d = 0, \tag{3}$$

где

$$a = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}, \tag{4}$$

$$b = \begin{bmatrix} x_1^2 + y_1^2 & y_1 & 1 \\ x_2^2 + y_2^2 & y_2 & 1 \\ x_3^2 + y_3^2 & y_3 & 1 \end{bmatrix}, \quad (5)$$

$$c = \begin{bmatrix} x_1^2 + y_1^2 & x_1 & 1 \\ x_2^2 + y_2^2 & x_2 & 1 \\ x_3^2 + y_3^2 & x_3 & 1 \end{bmatrix}, \quad (6)$$

$$d = \begin{bmatrix} x_1^2 + y_1^2 & x_1 & y_1 \\ x_2^2 + y_2^2 & x_2 & y_2 \\ x_3^2 + y_3^2 & x_3 & y_3 \end{bmatrix}. \quad (7)$$

Тогда из этого следует, что условие Делоне для любого заданного треугольника (($x_1$,$y_1$), ($x_2$,$y_2$), ($x_3$,$y_3$)) будет выполняться тогда и только тогда, когда для любого узла ($x_0$,$y_0$) из триангуляции не являющегося вершинами рассматриваемого треугольника будет выполняться неравенство

$$\left((x_0^2 + y_0^2) * a - x_0 * b + y_0 * c - d\right) * sgn\ a \geq 0. \quad (8)$$

Другими словами, когда любой узел ($x_0$,$y_0$) из триангуляции не попадает внутрь окружности, описанной вокруг рассматриваемого треугольника (($x_1$,$y_1$), ($x_2$,$y_2$), ($x_3$,$y_3$)).

### 3.3 Обзор основных алгоритмов построения триангуляции Делоне

На сегодняшний день известно немалое количество различных алгоритмов построения триангуляции Делоне, которые подразделяют на следующие группы:

- итеративные алгоритмы;
- алгоритмы прямого построения;
- алгоритмы слияния;
- двухпроходные алгоритмы;
- пузырьковая упаковка (Bubble packing).

# 4 Алгоритмы построения поверхности рельефа местности

Из всех известных методов, применяемых для построения цифровых моделей рельефа особенно актуальной проблемой, является построение триангуляции, так как многие существующие алгоритмы недоработаны, то есть по-прежнему неустойчивы и имеют неудовлетворительное время работы.

Применяя разные способы триангуляции, множество точек на плоскости будет триангулироваться по-разному, вследствие чего будут получаться различные немного схожие поверхности. Для изучения были выбраны два различных алгоритма триангуляции применяемых для разных видов ЦМР, а именно модели TIN и растровой модели представления поверхности. Для представления модели TIN выбран алгоритм «прямого построения триангуляции» Делоне, для растровой модели волновой итерационный алгоритм. Здесь уточним, что растровая модель, использующая регулярную триангуляционную сетку, заинтересовала меня в ходе работы и была реализована дополнительно с целью возможности наглядно сравнить с моделью TIN использующей нерегулярную триангуляционную сетку.

## 4.1 Постановка задачи

Дана некоторая исходная прямоугольная область

$$\Omega = [x_{min}, x_{max}] \times [y_{min}, y_{max}], \tag{9}$$

на которой задан набор изолиний $L = \{l_0, l_1, \dots, l_{m-1}\}$. Каждая из изолиний представляется парой $l_k =< h_k, V_k >$, где $h_k$ – уровень изолинии или высота набора вершин $V_k = \{(x_i^k, y_i^k), i = 0, \dots, n_k\}$.

По данным изолиниям необходимо построить ЦМР, произведя триангуляцию области $\Omega$, используя вычислительные алгоритмы №1(метод Делоне) и №2(волновой итерационный алгоритм), которые будут представлены далее.

### 4.2 Вычислительный алгоритм №1

Для построения модели TIN было решено использовать метод триангуляции Делоне из-за выполнения ранее указанных свойств ЦМР. Для реализации метода был выбран один из алгоритмов, который с точки зрения представленной классификации в главе 3.3, является одним из разновидностей алгоритма «прямого построения триангуляции».

При изучении метода выбор пал именно на этот алгоритм так как с его применением строятся сразу нужные треугольники, которые заведомо будут удовлетворять условию Делоне в итоговой триангуляции, и не будут перестраиваться, как в иных алгоритмах. За счет чего, данный алгоритм становится немного проще в реализации. Алгоритм «прямого построения триангуляции» Делоне описан Скворцовым А.В. в книге «Триангуляция Делоне и её применение» [4].

Вычислительный алгоритм №1 заключается:

1. в получении данных об исходном наборе изолиний;
2. в реализации алгоритма «прямого построения триангуляции» для построения сетки Делоне по заданному набору точек, где входные данные: множество точек, а выходные данные: нерегулярная треугольная сетка Делоне с высотными отметками в узлах;
3. в графическом представление построенной нерегулярной триангуляционной сетки с использованием библиотеки OpenGL на языке C++.

Опишем основные шаги алгоритма «прямого построения триангуляции» Делоне.

На первом шаге алгоритма находится начальное ребро, которое заведомо войдет в триангуляцию, как одно из ребер многоугольника выпуклой оболочки исходного множества точек. Первая точка ($A$) искомого ребра выбирается как точка с наименьшей координатой по $x$, а среди точек с одинаковой координатой

$x$ – как точка с наименьшей координатой по $y$ относительно начала координат. Вторая точка ($B$) начального ребра – это точка с наименьшим угловым коэффициентом отрезка [$AB$].

На рисунке 10 приведен пример выбора первого ребра.

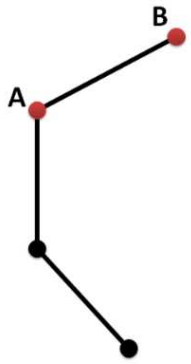


Рисунок 10 – Выбор первого ребра [$AB$]

В ходе работы алгоритма постоянно модифицируется список активных ребер *edges*, где «активными» ребрами называются ребра, у которых с одной стороны уже есть треугольник, а с другой еще надо пристроить (на рисунке 11 «активные» ребра выделены сплошным черным цветом). В этот список одни ребра добавляются, другие удаляются. Изначально список состоит из единственного ребра [$AB$].

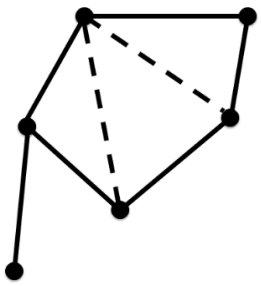

Рисунок 11 – Виды ребер

Дальнейший алгоритм состоит в повторении следующих шагов:

- осуществляется выбор очередного ребра из списка *edges*. Если это первое ребро, то предусмотрено его удаление из списка;

- поиск для выбранного ребра вершины – узла, который вместе с концами выбранного ребра в триангуляции Делоне будет образовывать вершины одного треугольника. То есть мы выбираем среди всех точек множества такую точку $P_i$ , что угол $AP_iB$ будет максимальным или радиус окружности, проходящей через точки $AP_iB$, будет наименьшим. Найденная вершина соединяется отрезками с концами отрезка и образует треугольник $AP_iB$. На рисунке 12 приведен пример поиска вершины от выбранного ребра [$AB$].

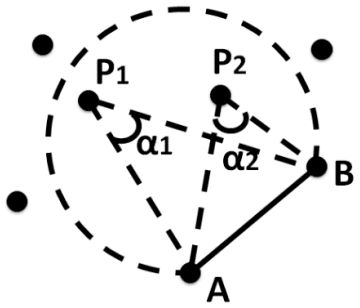


Рисунок 12 – Процесс поиска вершины

- новые рёбра $AP_i$ и $BP_i$ построенного треугольника добавляются в список *edges,* если их там еще нет. Если есть – то удаляем, это означает, что обработка ребра была завершена.

Алгоритм продолжает работу, пока список активных ребер не станет пустым.

Если точек $P_i$ с наименьшим радиусом окружности оказывается несколько (рисунке 13(а)), надо не выбирать какую-то одну из этих точек, а сразу добавить все получающиеся треугольники таким образом, чтобы не было самопересечений и конечно, модифицировать список активных ребер. На рисунке 13(б) показаны добавляемые треугольники – $AC_1B$, $C_1C_2B$, $C_2C_3B$ и новые активные ребра $AC_1$, $C_1C_2$, $C_2C_3$, $C_2C_3$, $C_3B$.

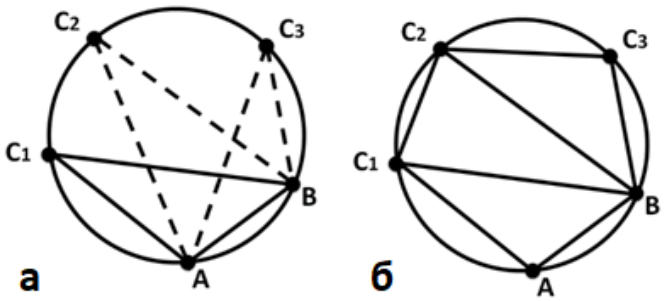


Рисунок 13 – Несколько точек с наименьшим радиусом окружности (а). Добавляемые треугольники, после модифицирования списка активных ребер (б).

### 4.3 Вычислительный алгоритм №2

Для построения растровой цифровой модели используется равномерная прямоугольная сетка и вычислительный алгоритм №2, позволяющий создавать достоверные ЦМР по высотам для любого типа рельефа.

В области $\overline{\Omega}$ строится равномерная прямоугольная сетка $\Omega_h$, которая является удобной и простой структурой для обработки поверхности:

$$\Omega_h = \{(x_i, y_j);\ x_i = x_{min} + i * h_x, y_j = y_{min} + j * h_y; i = 0, \dots, N; j = 0, \dots, M\}, \quad (10)$$

где $N$, $M$ – количество узлов сетки вдоль оси $OX$ и $OY$, а $h_x, h_y$ – шаг сетки по оси $OX$ и $OY$ соответственно

$$h_x = \frac{x_{max} - x_{min}}{N}, \quad (11)$$

$$h_y = \frac{y_{max} - y_{min}}{M}. \quad (12)$$

При создании регулярной сетки высот желательно учитывать плотность сетки (шаг сетки) поскольку, чем меньше выбранный шаг, тем точнее ЦМР, но тем больше количество узлов сетки, следовательно, больше времени потребуется на вычисления и больше места для хранения данных.

В каждой точке $\Omega_h$ необходимо вычислять значения высоты $z_{ij}$ с применением волнового итерационного алгоритма описанного в статье Кошель С.М. «Моделирование рельефа по изолиниям» [3].

Алгоритм №2 базируется на скором вычислении расстояний до двух ближайших изолиний разного уровня, и последующей линейной интерполяции, как и в пакетах GRASS и ILWIS, что дает возможность корректно определять высоты в узлах на участках, ограниченных изолинией только одного уровня, опираясь на значения на соседних участках. Для качественной работы программы необходимо безошибочно подготовить исходные данные, так как алгоритм опирается на топологическую структуру изолиний. Таким образом, для корректной работы алгоритма оцифрованные изолинии должны быть замкнутыми или начинаться и заканчиваться вне области моделирования, то есть не должны иметь разрывов. В рассматриваемых векторных картах в польском формате перечисленные условия учтены еще при составлении.

На рисунке 14 приведен пример одного из возможных наборов изолиний.

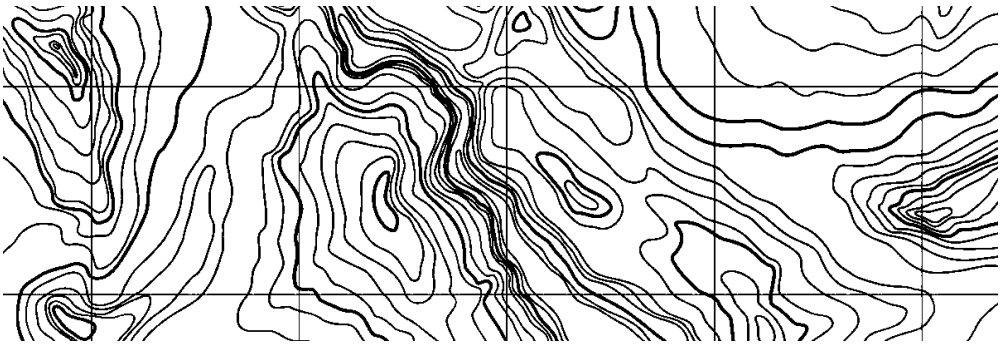

Рисунок 14 – Пример исходного набора изолиний.

Вычислительный алгоритм №2 заключается:

1. в получении данных об исходном наборе изолиний;
2. в реализации волнового итерационного алгоритма для вычисления высот в узлах регулярной триангуляционной сетки по набору точек изолиний, где входные данные: набор изолиний и количество разбиений сетки, а выходные данные: регулярная треугольная сетка с образованной матрицей высот;

3. в графическом представлении построенной регулярной триангуляционной сетки с использованием библиотеки OpenGL на языке C++.

Волновой итерационный алгоритм состоит из трех основных этапов.

На первом этапе (рисунок 15) находятся все пересечения изолиний со всеми ребрами сетки, в том числе с диагональными. Эти ребра помечаются, а соответствующему узлу сетки присваивается соответствующие высоты и не более двух ближайших расстояний до точки пересечения с изолинией разных уровней. В ходе присвоения расстояний может возникнуть три случая:

1. если узлу уже приписано некоторое расстояние до изолинии того же уровня, то выбирается меньшее из расстояний и запоминается уровень изолинии;

2. если узлу приписано только одно расстояние, причем до изолинии другого уровня, то запоминаются оба расстояния и соответствующие уровни изолиний;

3. если узлу приписаны два расстояния до изолиний разного уровня, причем новое расстояние вычислено до изолинии третьего уровня, то запоминаются только два значения с наименьшими расстояниями.

После окончания работы первого этапа алгоритма мы имеем некоторые помеченные ребра сетки и соответствующие узлы, которым приписаны расстояния не более чем до двух изолиний разного уровня (на рисунке 15 такие узлы обозначены черными точками).

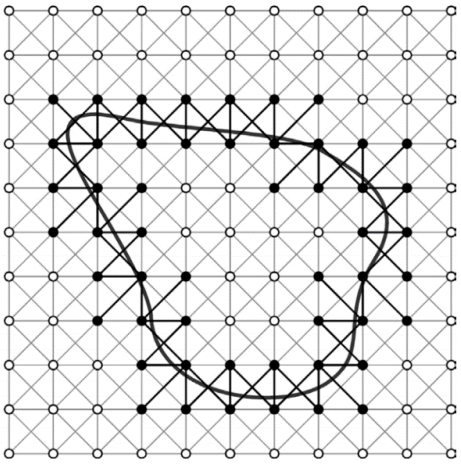

Рисунок 15 – Первый этап работы алгоритма

На втором этапе алгоритма находятся расстояния до двух ближайших изолиний в тех узлах сетки, которым еще не приписаны значения на первом этапе (на рисунке 15 эти узлы обозначены белыми точками) с помощью волнового итерационного алгоритма. Таким образом, последовательно перебираются узлы сетки, и из узлов, которым уже приписаны расстояния, выпускается "волна" в восьми направлениях (рисунке 16) вдоль ребер сетки, если ребро не было помечено, иначе движение в данном направлении запрещается.

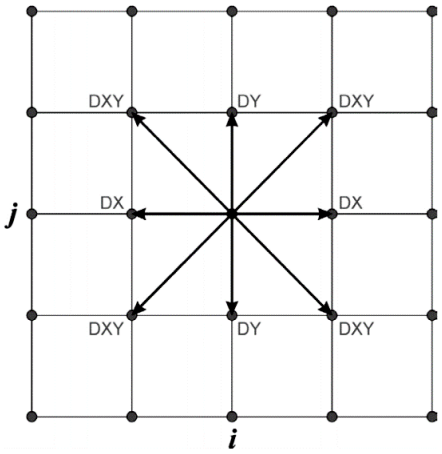


Рисунок 16 – Второй этап работы алгоритма

Далее к расстояниям, которые присвоены в текущем узлу например с индексами (*i*,*j*), прибавляется длина ребра сетки вдоль которого двигаемся (на рисунке 16 длины ребер сетки обозначены через *DX*, *DY*, *DXY*). Новые полученные расстояния и соответствующие им уровни изолинии сравниваются

с теми, которые приписаны узлу, в который мы попадаем в результате движения. В новом узле замена значений расстояний происходит также, как и на первом этапе, где рассматривались три возможных случая. Итерации выполняются до тех пор, пока в узлах происходят изменения с приписанными значениями расстояний. Количество итераций при этом всегда будет конечно.

На третьем этапе вычисляются высоты во всех узлах сетки. То есть после того, как цикл итераций будет завершен, всем узлам сетки будут приписаны расстояния до ближайших изолиний и значения уровней этих изолиний. Причем для некоторых узлов ближайших расстояний до изолиний будет два, а для некоторых – только одно. Дело в том, что изолинии разбивают ЦМР на участки, в границу которых могут входить либо изолинии двух различных уровней, либо только одного, например локальные холмы или впадины. Для узлов с двумя расстояниями значение высоты рассчитывается по формуле

$$h = \frac{h_1 d_2 + h_2 d_1}{d_1 + d_2}, \tag{13}$$

где $d_1$, $d_2$ – расстояния до ближайших изолиний, $h_1$, $h_2$ – значения соответствующих уровней изолиний.

Узлам с одним расстоянием присваивается значение высоты, равное уровню соответствующей изолинии. Вычисления на этом заканчиваются.

# 5 Этапы построения поверхности

Опишем общие этапы построения ЦМР:

1. считываются изолинии из текстового файла;
2. исходное множество точек наносится на плоскость *XOY* (без учета координаты *z* (высоты)). Высоты каждой точки хранятся отдельно;
3. плоскость *XOY* окрашивается цветами с учетом значения координаты *z* в каждой точке на плоскости;
4. строится триангуляционная сетка на плоскости *XOY* с использованием триангуляционного алгоритма;
5. строится ЦМР в пространстве *XYZ*, при помощи соединения точек на плоскости *XOY* с известными значениями высот.

На рисунке 17 приведены наглядные этапы построения искусственной поверхности.

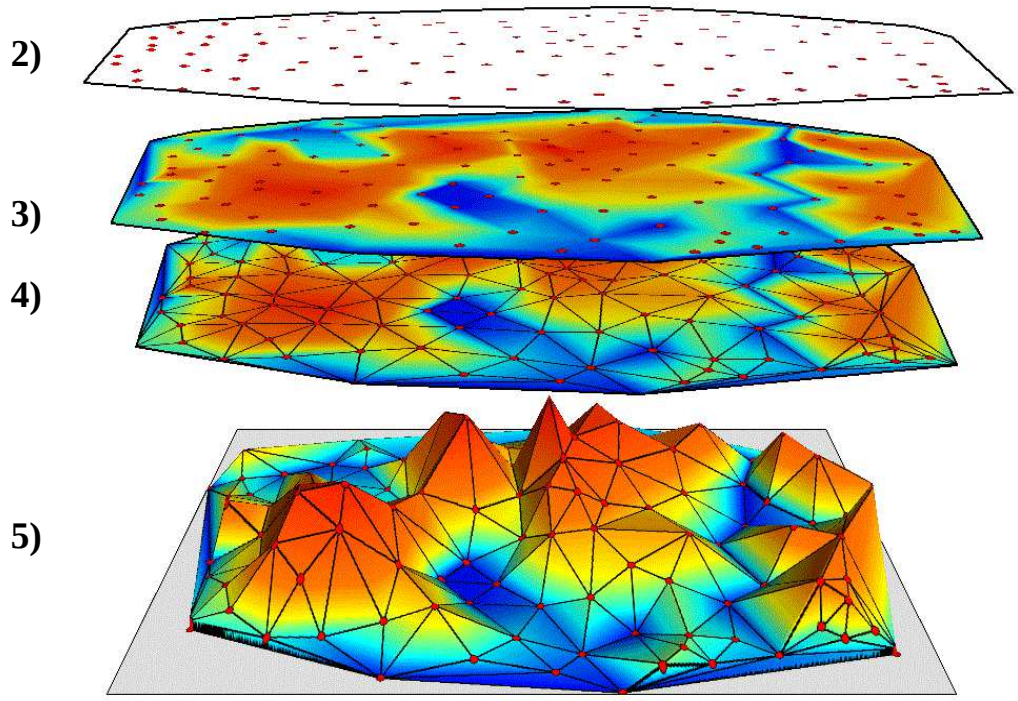


Рисунок 17 – Искусственная поверхность

В приложении А на рисунке А.1 приведены наглядные этапы построения реальной поверхности горного хребта Борус.

# 6 Результаты тестовых экспериментов

Вычислительные алгоритмы №1 и №2 реализованы в виде прикладных программ на языке C++ [5] с использованием графической библиотеки OpenGL [1]. В качестве входных данных для программ выступают текстовые файлы, содержащие информацию об объектах местности, в том числе наборы изолиний. Каждая из изолиний представляет собой набор координат с одинаковой высотой. В программах осуществляется чтение входных текстовых файлов, из которых выделяется информация только об изолиниях, вся остальная информация отбрасывается. В результате работы программ на экран выводится изображение поверхности, построенное с применением графических функций библиотеки OpenGL. Программы поддерживают возможность осуществить повороты представленной поверхности для удобства восприятия пользователем. При выводе изображения используется палитра цветов, каждый цвет которой обозначает разные уровни высоты. Таким образом, мы получаем цветное изображение поверхности, иллюстрирующее рельеф местности.

## 6.1 Построение поверхности рельефа для тестового набора изолиний

Для тестирования работы алгоритма был сгенерирован набор изолиний (рисунок 19), представляющий некую искусственную поверхность. Этот набор содержит 12 изолиний (109 точек), разделяющихся на три группы, каждая их, которых представляет собой некое подобие горы. Перепад высот в изолиниях составляет от 0 до 140 метров.

На рисунке 18 представлен вывод программы об успешном чтении данных из входного текстового файла, содержащего вышеуказанный набор изолиний.

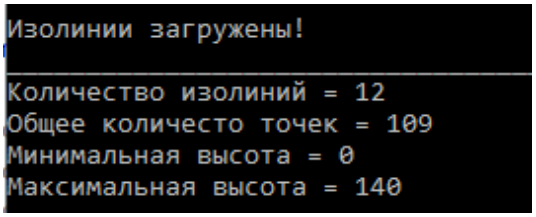


Рисунок 18 – Вывод программы об успешном чтении данных из текстового файла

Текстовый файл, содержащий указанный набор изолиний (рисунок 19), приведен в приложении Б.

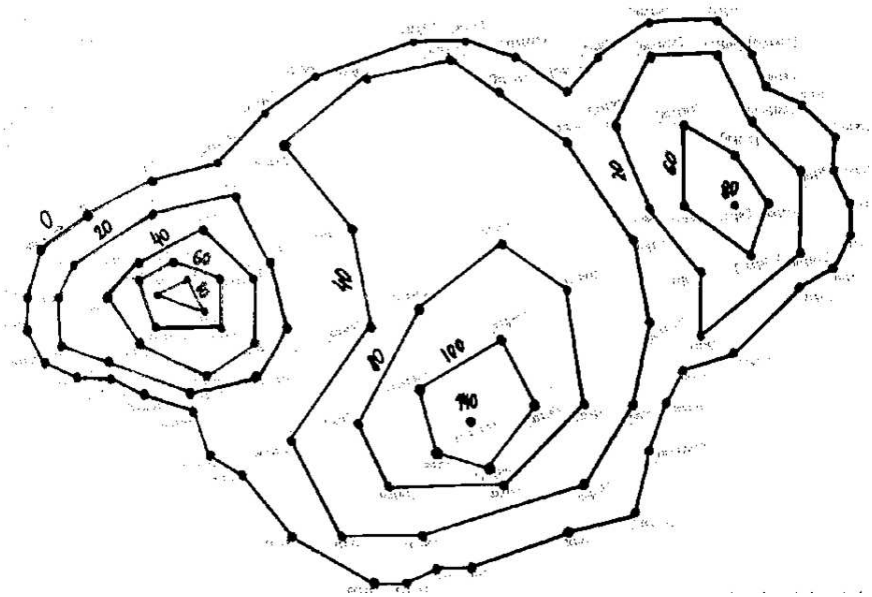


Рисунок 19 – Исходный набор изолиний искусственной поверхности

Результаты работы программы, реализующей вычислительный алгоритм №1 для набора изолиний (рисунок 19), представлены на рисунке 20 с четырех ракурсов (поворот осуществлялся против часовой стрелки).

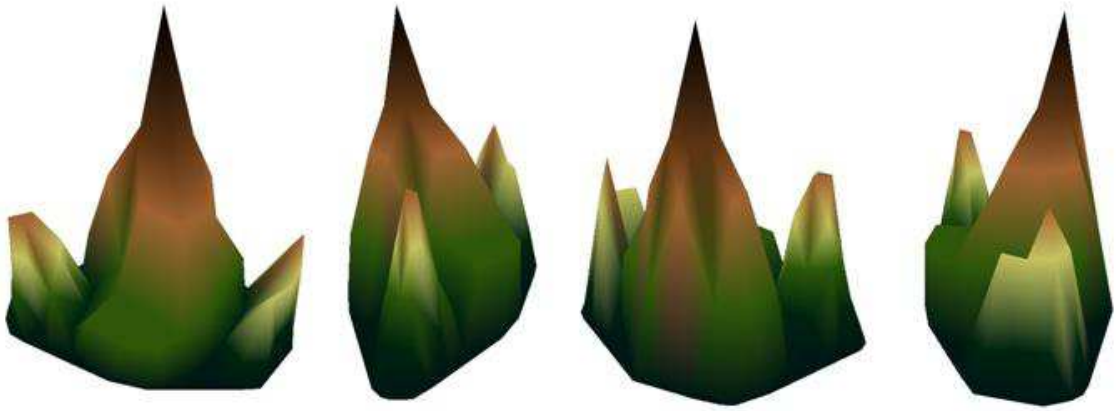

Рисунок 20 – Поверхности, построенные для сгенерированного набора изолиний

Результаты работы программы, строящую регулярную сетку, приведены на рисунке 21. Вычисления производились для прямоугольной сетки с числом разбиений $N$=6;20;60;120 по каждой размерности.

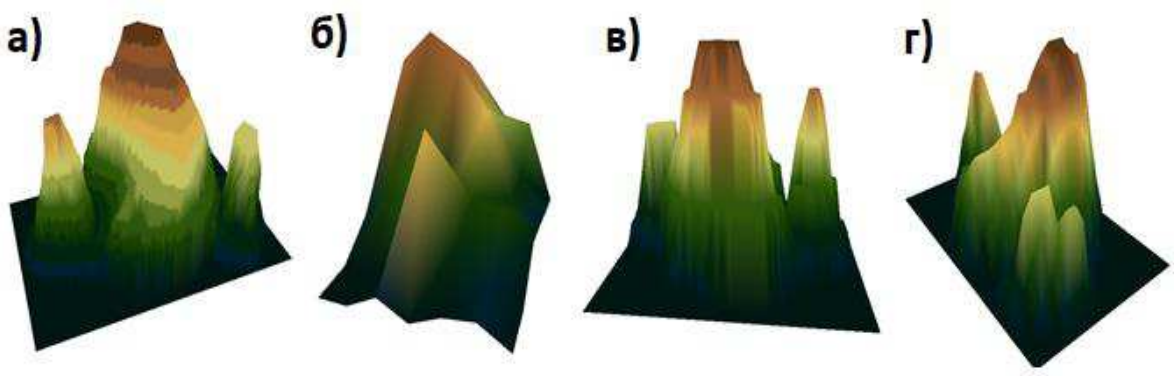


Рисунок 21 – Результаты работы программы для регулярной сетки с числом разбиений а) *N*=120; б) *N*=6; в) *N*=60; г) *N*=20

### 6.2 Построение поверхности рельефа горного хребта Борус

Алгоритмы применялись и тестировались также для реальной местности, с целью создать объемную поверхность уже существующего объекта по имеющемуся набору изолиний. Для рассмотрения была выбрана векторная карта горного хребта Борус, расположенного на юге Красноярского края, в системе Западного Саяна. Карта содержит более 500 изолиний, которые в совокупности имеют около $3*10^5$ точек. Протяженность составляет от 91.20º до 91.40º восточной долготы и от 52.42º до 52.54º северной широты, а перепад высот от 320 до 2260 метров над уровнем моря. На рисунке 22 представлены растровые карты горного хребта Борус, загруженные из сети интернет (справа скриншот карты из Google Earth), для возможного сравнения их с результатами работы программ, представленных на рисунке 24,25.


Рисунок 22 – Растровые карты горного хребта Борус

На рисунке 23 продемонстрирован вывод программы об успешном чтении данных из входного текстового файла, содержащего набор изолиний о горном хребте Борус.

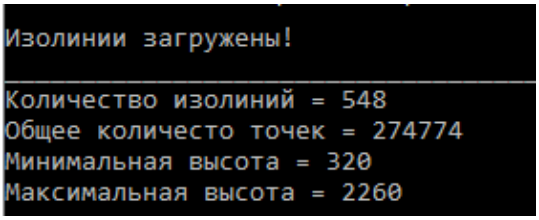


Рисунок 23 – Вывод программы об успешном чтении данных из текстового файла

Результаты работы программы, реализующей вычислительный алгоритм №1, представлены на рисунке 24 с четырех ракурсов (поворот поверхности осуществлялся против часовой стрелки).

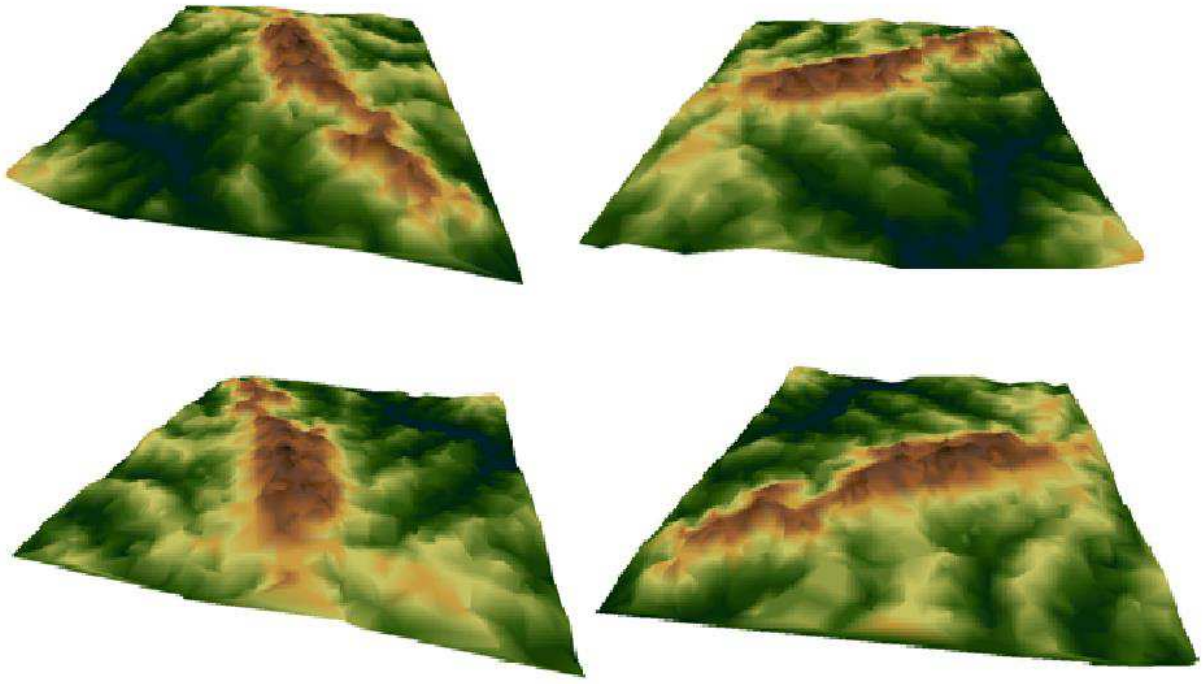

Рисунок 24 – Вычислительный алгоритм №1. Нерегулярная треугольная сетка

С применение вычислительного алгоритма №2 построена регулярная треугольная сетка. Вычисления производились для прямоугольной сетки с числом разбиений $N$=20;80;180;400 по каждой размерности. Полученные цифровые модели представлены с четырех разных ракурсов на рисунке 25 (поворот поверхности осуществлялся против часовой стрелки).

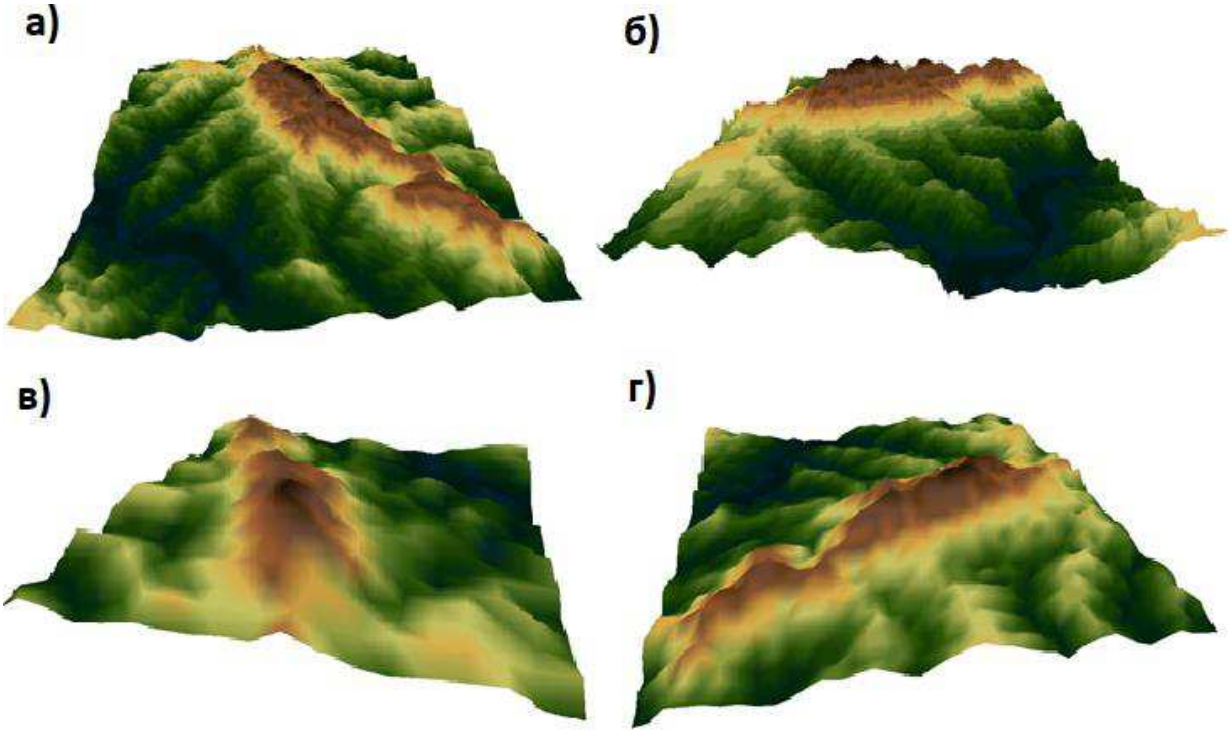


Рисунок 25 – Результаты работы программы для регулярной сетки с числом разбиений а) $N$=180; б) $N$=400; в) $N$=20; г) $N$=80

### 6.3 Построение поверхности рельефа горного хребта Кутурчин

Кутурчинское Белогорье (Кутурчин) – горный хребет протяженностью около 80 км, расположенный в Красноярском крае, в пределах Восточного Саяна, состоящий из нескольких хребтов, вершин и горных плато.

Протяженность выбранной карты составляет от 93.61º до 94.35º восточной долготы и от 54.69º до 55.05º северной широты. Карта содержит 3070 изолиний, которые в совокупности содержат около $23*10^4$ вершин. Максимальная высота хребта – 1876 м. Полученные цифровые модели приведены в приложении В.

### 6.4 Построение поверхности рельефа горного хребта Ергаки

Горный хребет Ергаки расположен на юге Красноярского края, входит в состав Западного Саяна. Протяжённость с севера на юг составляет 75 км, а по долготе – около 120 км.

Протяженность выбранной карты составляет от 92.87º до 93.70º восточной долготы и от 52.44º до 53.01º северной широты. Карта содержит 3108 изолиний, которые в совокупности содержат около $16*10^5$ вершин. Высота от 1100 м до

1800 м. Главные пики Ергак достигают 2000-2200 м. Наивысшая вершина – пик Звёздный, высота 2265 м. Полученные цифровые модели представлены в приложении Г.

Тексты программ, реализующие рассмотренные вычислительные алгоритмы приведены в приложениях. В приложении Д представлен текст программы, реализующей вычислительный алгоритм №1, а в приложении Е – текст программы вычислительного алгоритма №2.

# ЗАКЛЮЧЕНИЕ

В представленной работе были решены все поставленные задачи.

1. Реализован вычислительный алгоритм построения поверхности рельефа с использованием триангуляции Делоне.

2. Реализован волновой итерационный алгоритм построения поверхности рельефа.

3. Вычислительные алгоритмы реализованы в виде программного комплекса на языке C++ с использованием графической библиотеки OpenGL.

4. Проведены тестовые эксперименты с применением реализованных алгоритмов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Баранов С. Н. Основы компьютерной графики [Текст] : учебное пособие / С. Н. Баранов, С. Г. Толкач ; Сиб. федер. ун-т, Ин-т математики и фундамент. информатики. - Красноярск: СФУ, 2018. – 85 с.
2. Варфоломеев И.В. Алгоритмы и структуры данных геоинформационных систем: Методические указания для студентов специальности 071903 – «Геоинформационные системы» / Варфоломеев И.В., Ермакова И.Г., Савельев А.С. – Красноярск: КГТУ, 2003. – 34 с.
3. Кошель С.М. Моделирование рельефа по изолиниям. Географический факультет МГУ им. Ломоносова, кафедра картографии и геоинформатики [Электронный ресурс]: – Режим доступа https://www.researchgate.net/publication/294870416_Modelirovanie_relefa_po_izoliniam
4. Объектно-ориентированное программирование на C++: учебник / И.В. Баранова, С.Н. Баранов, И.В. Баженова [и др.]. – Красноярск: Сиб. федер. ун-т, 2019. – 288с.
5. Скворцов А.В. Триангуляция Делоне и её применение. Томск. 2002. – 128 с.
6. Хромых В.В. Цифровые модели рельефа: Учебное пособие / Хромых В.В., Хромых О.В. – Томск: Изд-во «ТМЛ-Пресс», 2007. – 178 с.
7. Шнитко С.Г. Конспект лекций по предмету «ГИС в геодезии» [Электронный ресурс] – Режим доступа: http://konferenciya.seluk.ru/lekcii/1099429-1-konspekt-lekciy-obschie-svedeniya-geoinformacionnih-sistemah-ponyatie-geograficheskoy-informacionnoy-sistemi-poyavlenie.php

# ПРИЛОЖЕНИЯ

## Приложение А. Этапы построения поверхности горного хребта Борус

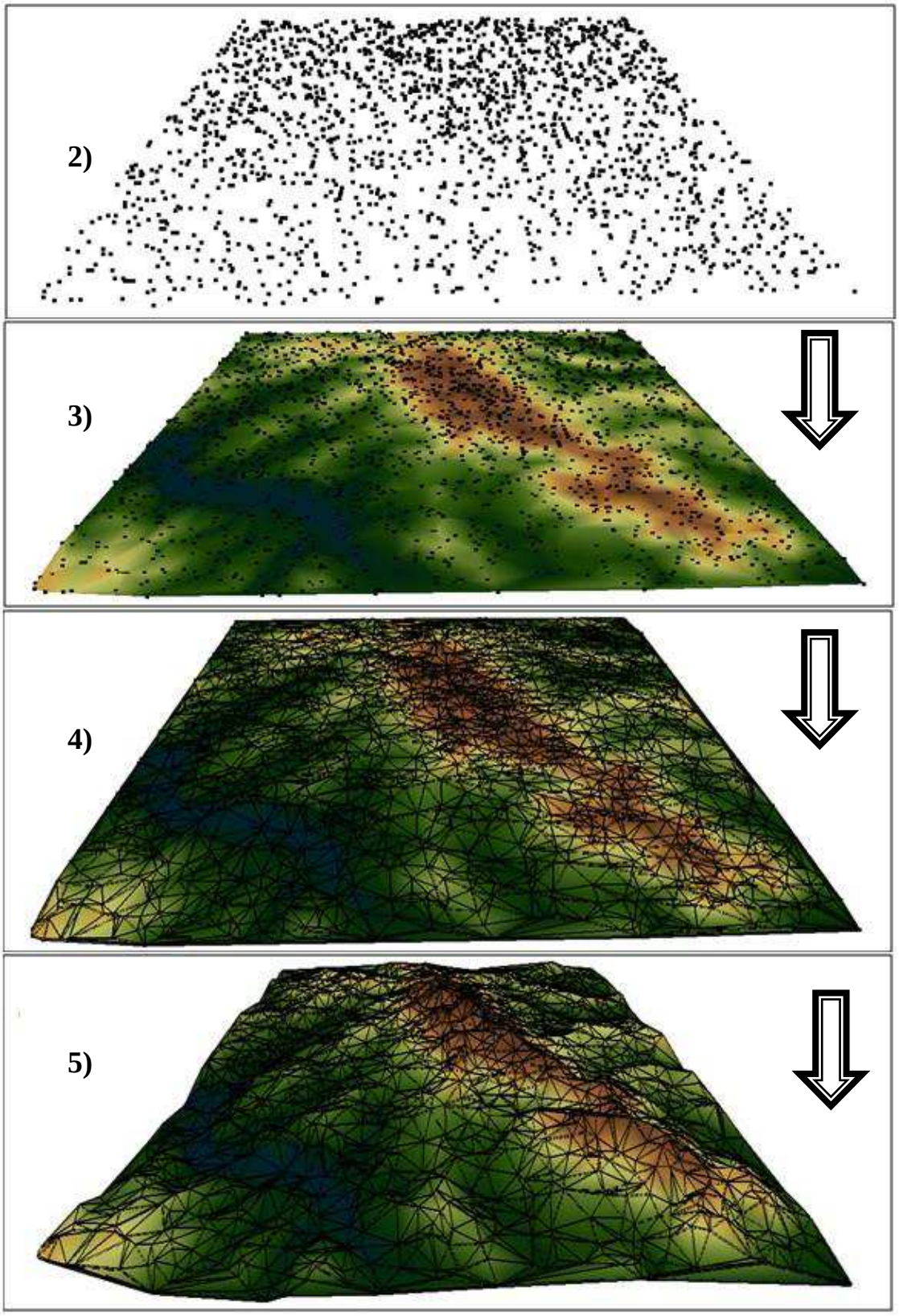


Рисунок А.1 – Этапы построения поверхности горного хребта Борус

## Приложение Б. Набор сгенерированных изолиний

```
[POLYLINE]
Type=0x20
Label=0
Data=(10,55),(17.5,60),(27.5,65),(37.5,67.5),(45,75),(52.
5,80),(67.5,85),(75,85),(82.5,82.5),(90,77.5),(95,82.5),(
102.5,87.5),(112.5,87.5),(117.5,82.5),(120,77.5),(125,75)
,(130,70),(130,65),(132.5,60),(132.5,55),(130,50),(125,47
.5),(115,37.5),(107.5,35),(105,30),(102.5,22.5),(100,12.5
),(90,10),(75,5),(70,5),(65,2.5),(60,2.5),(47.5,10),(40,2
0),(35,22.5),(32.5,30),(25,32.5),(20,35),(15,35),(10,37.5
),(7.5,42.5),(7.5,47.5)
[END]
[POLYLINE]
Type=0x20
Label=20
Data=(27.5,60),(40,62.5),(45,52.5),(47.5,42.5),(42.5,35),
(32.5,32.5),(20,37.5),(15,40),(15,47.5),(15,52.5)
[END]
[POLYLINE]
Type=0x20
Label=20
Data=(97.5,72.5),(102.5,82.5),(112.5,82.5),(117.5,72.5),(
125,65),(125,52.5),(110,40),(110,50),(107.5,60)
[END]
[POLYLINE]
Type=0x20
Label=40
Data=(20,52.5),(35,57.5),(42.5,50),(42.5,40),(35,37.5),(2
5,40),(20,47.5)
[END]
[POLYLINE]
Type=0x20
Label=40
Data=(57.5,57.5),(47.5,70),(60,80),(72.5,82.5),(80,77.5),
(90,70),(100,55),(107.5,42.5),(100,30),(92.5,17.5),(67.5,
10),(55,10),(47.5,25),(60,42.5)
[END]
[POLYLINE]
Type=0x20
Label=60
Data=(25,50),(30,52.5),(40,50),(40,42.5),(27.5,42.5)
[END]
[POLYLINE]
```

```
Type=0x20
Label=60
Data=(107.5,72.5),(115,67.5),(120,60),(117.5,52.5),(107.5
,60)
[END]
[POLYLINE]
Type=0x20
Label=80
Data=(27.5,47.5),(32.5,50),(35,45)
[END]
[POLYLINE]
Type=0x20
Label=80
Data=(115,60)
[END]
[POLYLINE]
Type=0x20
Label=80
Data=(67.5,45),(80,55),(90,47.5),(92.5,30),(80,17.5),(62.
5,17.5),(57.5,27.5)
[END]
[POLYLINE]
Type=0x20
Label=100
Data=(67.5,32.5),(77.5,40),(85,30),(77.5,20),(70,22.5)
[END]
[POLYLINE]
Type=0x20
Label=140
Data=(75,27.5)
[END]
```

## Приложение В. Полученные цифровые модели горного хребта Кутурчин

Полученная ЦМР с применением вычислительного алгоритма №1 с четырех ракурсов (поворот поверхности осуществлялся по часовой стрелке) приведена на рисунке В.1.

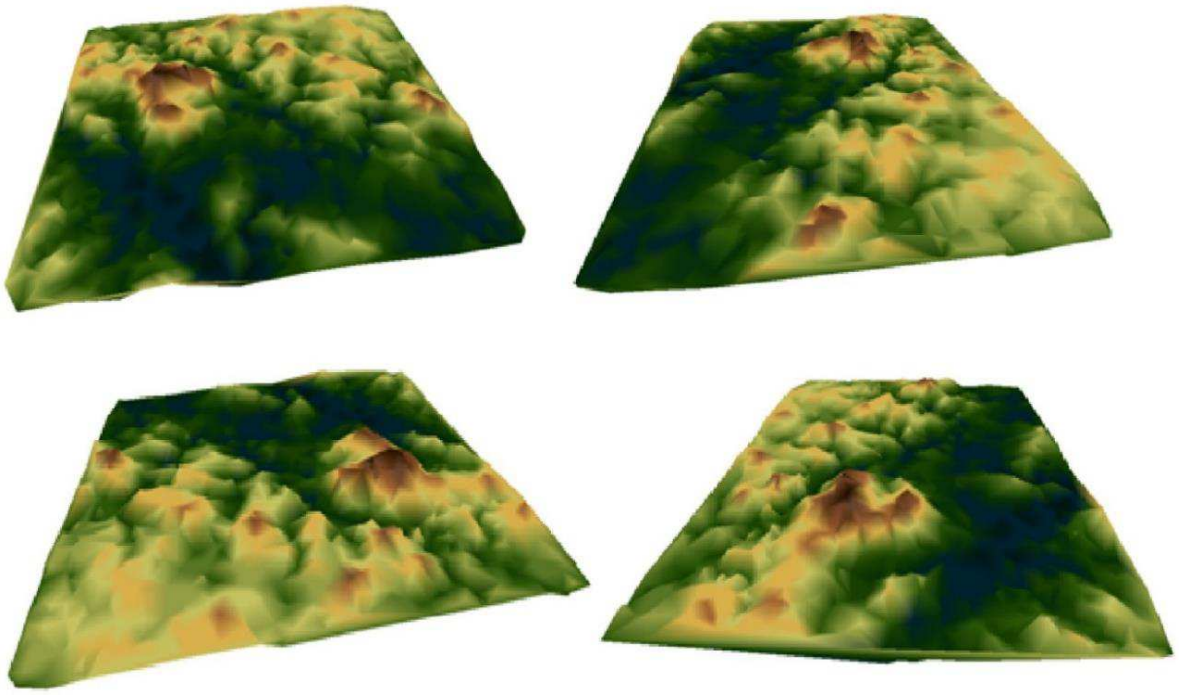

Рисунок В.1 – Вычислительный алгоритм №1

Полученная ЦМР с применением вычислительного алгоритма №2 с четырех ракурсов (поворот поверхности осуществлялся по часовой стрелке) для прямоугольной сетки с числом разбиений $N$=30;80;160;400 по каждой размерности приведена на рисунке В.2.

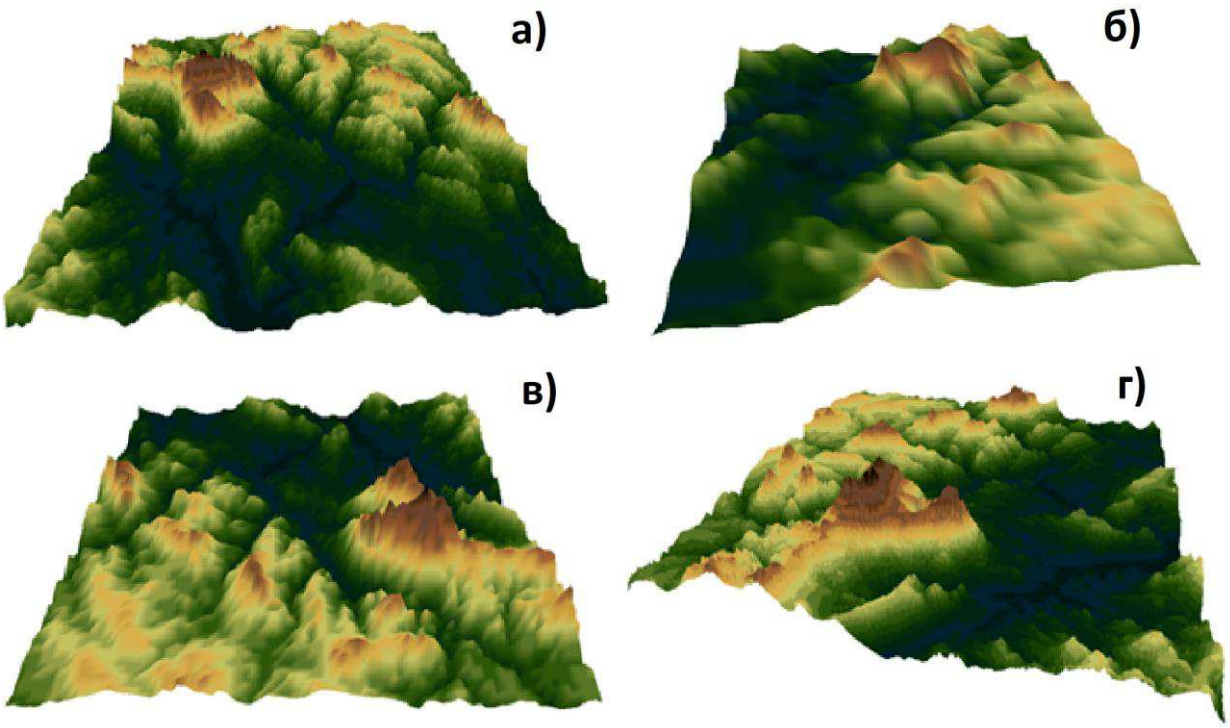


Рисунок В.2 – Результаты работы программы для регулярной сетки с числом разбиений а) N=160; б) N=30; в) N=400; г) N=80

## Приложение Г. Полученные цифровые модели горного хребта Ергаки

Полученная ЦМР с применением вычислительного алгоритма №1 с четырех ракурсов (поворот поверхности осуществлялся против часовой стрелки) представлена на рисунке Г.1.

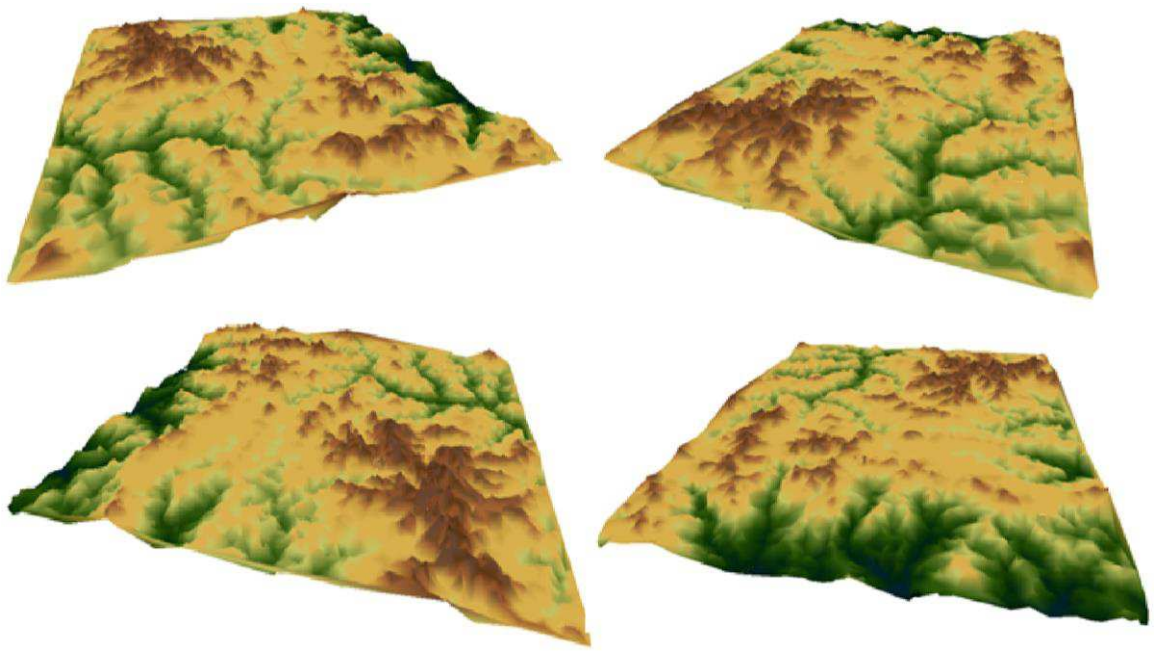

Рисунок Г.1 – Вычислительный алгоритм №1

Полученная ЦМР с применением вычислительного алгоритма №2 с четырех ракурсов (поворот поверхности осуществлялся по часовой стрелке) для прямоугольной сетки с числом разбиений $N$=30;60;150;400 по каждой размерности представлена на рисунке Г.2.

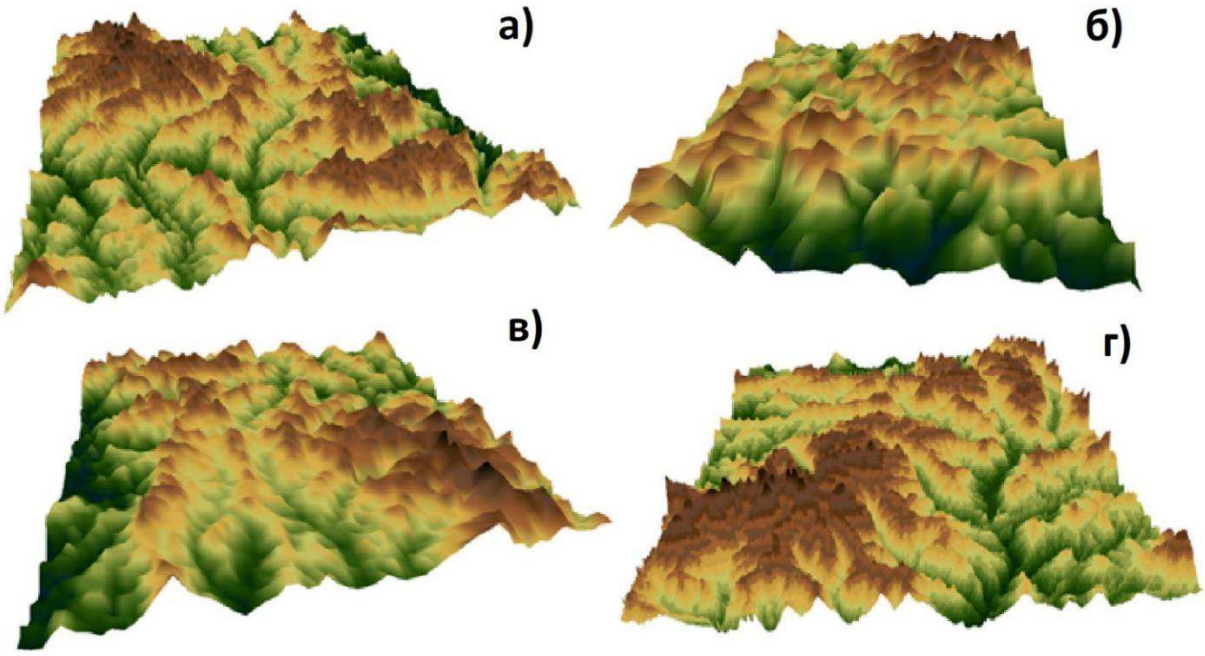


Рисунок Г.2 – Результаты работы программы для регулярной сетки с числом разбиений а) N=150; б) N=30; в) N=60; г) N=400

**Приложение Д. Текст программы вычислительного алгоритма №1**

```
#include <iostream>
#include <math.h>
#include <windows.h>
#include <vector>
#include <fstream>
#include <string>
#include <GL/gl.h>
#include <GL/glu.h>
#include "glaux.h"
using namespace std;
#include "Delaunay.h"
#pragma comment (lib,"opengl32.lib")
#pragma comment (lib,"glu32.lib")
#pragma comment (lib, "glaux.lib")
#pragma comment(lib, "legacy_stdio_definitions.lib")
// Описание типов данных:
struct Point //запись точек
{
    double x, y, z;
    Point(double x_ = 0, double y_ = 0, double z_ = 0)
    {
        x = x_;
        y = y_;
        z = z_;
    }
};
typedef vector<Point> Vector;
Vector V;
pvector p, tr;
void Load(string filename, Vector& V);//Загрузка
float anglez = 45, anglex = 0, rz = 200;
float ex = 0, ey = 10, ez = 0;
float ax = 0, ay = 0, az = 0;
int  zmax;
const int CountColors = 20;
double RGB[CountColors][3] =
{
    {0,0,0},
    {0.15,0.07,0},
    {0.27,0.17,0.06},
    {0.44,0.26,0.17},
    {0.40,0.27,0.19},
    {0.37,0.21,0.08},
```

```
    {0.56,0.33,0.17},
    {0.45,0.29,0.17},
      {0.63,0.39,0.2},
    {0.79,0.62,0.25},
      {0.85,0.69,0.3},
      {0.77,0.76,0.38},
      {0.58,0.62,0.27},
    {0.35,0.46,0.15},
    {0.18,0.34,0.04},
    {0.1,0.25,0.01},
      {0.03,0.18,0.04},
    {0.0,0.14,0.06},
    {0,0.13,0.18},
      {0,0.09,0.08}
};
void CALLBACK PressKeyA(void) // шаг по кругу
{
    anglez += 0.1;
}
void CALLBACK PressKeyD(void) // шаг по кругу
{
    anglez -= 0.1;
}

void CALLBACK PressKeyW(void) // шаг вперед
{
    rz -= 1;
}
void CALLBACK PressKeyS(void)// шаг назад
{
    rz += 1;
}
void CALLBACK PressKeyQ(void) // шаг вверх
{
    ey += 10;
}
void CALLBACK PressKeyE(void) // шаг вниз
{
    ey -= 10;
}
void CALLBACK resize(int width, int height)
{
    glViewport(0,  0,  width,  height);//область  вывода
OpenGL-графики, совпадает с размером окна
```

```
    glMatrixMode(GL_PROJECTION);//включение режима работы
с матрицей проекций
    glLoadIdentity();//загрузка единичной матрицы,
другими словами сброс текущей матрицы
    gluPerspective(1500, double(width) / height, 1,
1000);//Устанавливаем тип проекции. Задание перспективной
проекции
    glMatrixMode(GL_MODELVIEW);//включение режима работы
с модельно-видовой матрицей
}
void system(int n) // Функия - система координат
{
    glLineWidth(2);
    //ось OX - красный
    glBegin(GL_LINES);
    glColor3d(1, 0, 0);
    glVertex3d(-n, 0, 0);
    glVertex3d(n, 0, 0);
    glEnd();
    //стрелки на OX
    glBegin(GL_LINE_STRIP);
    glVertex3d(n - 2, 2, 0);
    glVertex3d(n, 0, 0);
    glVertex3d(n - 2, -2, 0);
    glEnd();
    //ось OY - синий
    glBegin(GL_LINES);
    glColor3d(0, 0, 1);
    glVertex3d(0, -n, 0);
    glVertex3d(0, n, 0);
    glEnd();
    //стрелки на OY
    glBegin(GL_LINE_STRIP);
    glVertex3d(-2, n - 2, 0);
    glVertex3d(0, n, 0);
    glVertex3d(2, n - 2, 0);
    glEnd();
    //ось OZ - зеленый
    glBegin(GL_LINES);
    glColor3d(0, 1, 0);
    glVertex3d(0, 0, n);
    glVertex3d(0, 0, -n);
    glEnd();
    //стрелки на OZ
    glBegin(GL_LINE_STRIP);
```

```
    glVertex3d(0, -2, n - 2);
    glVertex3d(0, 0, n);
    glVertex3d(0, 2, n - 2);
    glEnd();
}

void DrawTriangle(GLdouble x1, GLdouble y1, GLdouble z1,
    GLdouble x2, GLdouble y2, GLdouble z2,
    GLdouble x3, GLdouble y3, GLdouble z3,
    GLdouble r1, GLdouble g1, GLdouble b1,
    GLdouble r2, GLdouble g2, GLdouble b2,
    GLdouble r3, GLdouble g3, GLdouble b3)
{ // передаются координаты и цвета трех точек треугольника
    glBegin(GL_TRIANGLES);
    glColor3d(r1, g1, b1);  glVertex3d(x1, y1, z1);
    glColor3d(r2, g2, b2);  glVertex3d(x2, y2, z2);
    glColor3d(r3, g3, b3);  glVertex3d(x3, y3, z3);
    glEnd();
}
void TRIANGLES_Kontur(float x1, float y1, float z1, float
x2, float y2, float z2, float x3, float y3, float z3, float
r, float b, float g) //контур треугольника
{
    glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);//режим
растеризации(заливка, контур или точки вершин)- контур
    glLineWidth(4);
    glBegin(GL_TRIANGLES);
    glColor3d(r, g, b);
    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y2, z2);
    glVertex3f(x3, y3, z3);
    glEnd();
    glPolygonMode(GL_FRONT_AND_BACK,
GL_FILL);//восстановление режима по умолчанию - заливка
}
void san()
{
    float H;
    int k1, k2, k3;
    float r1, r2, r3, g1, g2, g3, b1, b2, b3;
    float  min_z,  max_z,  min_x,  max_x,  min_y,  max_y;
//нахожу максимальные и минимальные значения x,y,z
    min_z = max_z = V[0].z;
    min_x = max_x = V[0].x;
    min_y = max_y = V[0].y;
```

```
    for (int i = 0;i < V.size();i++)
    {
        if (V[i].z < min_z)
            min_z = V[i].z;
        if (V[i].z > max_z)
            max_z = V[i].z;
        if (V[i].x < min_x)
            min_x = V[i].x;
        if (V[i].x > max_x)
            max_x = V[i].x;
        if (V[i].y < min_y)
            min_y = V[i].y;
        if (V[i].y > max_y)
            max_y = V[i].y;
    }
    H = (max_z - min_z) / CountColors;// разбиваю высоты
на части по цветаи
    Point A, B, C;
    for (int i = 0;i < tr.size(); i++)
    {
        for (int j = 0; j < p.size(); j++)
        {
            if (tr[i].x == p[j].x && tr[i].y == p[j].y) {
                tr[i].z = p[j].z;
                break;
            }
        }
    }
    for (int i = 0;i < tr.size(); i += 3)
    {
        A.x = tr[i].x; A.y = tr[i].y; A.z = tr[i].z;
        B.x = tr[i + 1].x; B.y = tr[i + 1].y; B.z = tr[i
+ 1].z;
        C.x = tr[i + 2].x; C.y = tr[i + 2].y; C.z = tr[i
+ 2].z;

        k1 = (max_z - A.z) / H;
        if (k1 == CountColors) k1 = CountColors - 1;
        r1 = RGB[k1][0];g1 = RGB[k1][1];b1 = RGB[k1][2];

        k2 = (max_z - B.z) / H;
        if (k2 == CountColors) k2 = CountColors - 1;
        r2 = RGB[k2][0];g2 = RGB[k2][1];b2 = RGB[k2][2];

        k3 = (max_z - C.z) / H;
```

```
            if (k3 == CountColors) k3 = CountColors - 1;
            r3 = RGB[k3][0];g3 = RGB[k3][1];b3 = RGB[k3][2];
            //рисую треугольник
            DrawTriangle(
                A.x, A.y, A.z,
                B.x, B.y, B.z,
                C.x, C.y, C.z,
                r1, g1, b1,
                r2, g2, b2,
                r3, g3, b3);
        }
    }
    void scene()
    {
        glRotated(-90, 1, 0, 0);
        glRotatef(0.45f, 0.0f, 0.0f, 1.0f); //поворачиваем по
    школе z на 45 градусов,
        glPushMatrix();//сохранили текущую систему координат

        san();
        glPopMatrix();
    }
    void CALLBACK display(void)
    {
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    //очищаем буфер цвета и буфер глубины
        glLoadIdentity();
        ex = rz * sin(anglez);
        ez = rz * cos(anglez);
        gluLookAt(ex, ey, ez, ax, ay, az, 0, 1, 0);
        scene();
        auxSwapBuffers();//эта команда выводит содержимого
    буфера на экран
    }
    void RunOpenGL()
    {
        auxKeyFunc(AUX_w, PressKeyW);
        auxKeyFunc(AUX_s, PressKeyS);
        auxKeyFunc(AUX_a, PressKeyA);
        auxKeyFunc(AUX_d, PressKeyD);
        auxKeyFunc(AUX_q, PressKeyQ);
        auxKeyFunc(AUX_e, PressKeyE);
        auxInitPosition(0, 0, 1000, 1000);// верхний левый
    угол (0,0), ширина и высота - 1000
```

```
    auxInitDisplayMode(AUX_RGB|AUX_DEPTH|AUX_DOUBLE);//
устанавливаем параметры контекста OpenGL
    auxInitWindow(L"Glaux Template");// создаем окно на
экране
    // наше окно будет получать сообщения от клавиатуры,
мыши, таймера или любые другие
    // когда никаких сообщений нет, будет вызываться
функция display. Так мы получаем анимацию
    auxIdleFunc(display);
    auxReshapeFunc(resize);
    glClearColor(1, 1, 1, 0); //цвет фона
    glEnable(GL_DEPTH_TEST);//включить тест глубины
}
void Load(string filename, Vector& V)//Загрузка множества
точек с файла
{
    fstream Loading(filename.c_str());
    string str, str2;
    char symbol;
    double h, x, y, z;
    while (!Loading.eof())
    {
        do
        {
            getline(Loading, str, '\n');
        } while (str != "[POLYLINE]");
        getline(Loading, str, '\n'); // Type
        getline(Loading, str, '='); // Label
        Loading >> h;
        getline(Loading, str, '='); // Data0
        Loading.get();
        while (1)
        {
            Loading >> y >> symbol >> x;
            do
            {
                Loading >> symbol;
            } while (!isdigit(Loading.peek()) && symbol
!= '[');
            V.push_back(Point(x * 67000, y * 111000,
h));// произвела перевод широты и долготы в метры
            //V.push_back(Point(x, y, h));
            //V.push_back(Point(floor(x*10)/100, floor(y
* 10) / 100, h/10));
            if (symbol == '[' || symbol == 'E') break;
```

```
        }
        getline(Loading, str, '\n');
        Loading.get();
    }
    //нахожу максимальные и минимальные значения x,y,z
    float min_z, max_z, min_x, max_x, min_y, max_y;
    min_z = max_z = V[0].z;
    min_x = max_x = V[0].x;
    min_y = max_y = V[0].y;
    for (int i = 0;i < V.size();i++)
    {
        if (V[i].z < min_z)
            min_z = V[i].z;
        if (V[i].z > max_z)
            max_z = V[i].z;
        if (V[i].x < min_x)
            min_x = V[i].x;
        if (V[i].x > max_x)
            max_x = V[i].x;
        if (V[i].y < min_y)
            min_y = V[i].y;
        if (V[i].y > max_y)
            max_y = V[i].y;
    }
    float DX = (max_x + min_x) / 2; //масштабирование
    float DY = (max_y + min_y) / 2;
    float DZ = (max_z + min_z) / 2;
    float MAXXX = max((max_z - min_z) / 2, max((max_x -
min_x) / 2, (max_y - min_y) / 2));
    for (int i = 0;i < V.size();i++)
    {
        V[i].x -= DX;
        V[i].x = (V[i].x / MAXXX);
        V[i].x *= 100; // размерность от -100 до 100
        V[i].y -= DY;
        V[i].y = (V[i].y / MAXXX);
        V[i].y *= 100;
        V[i].z -= DZ;
        V[i].z = (V[i].z / MAXXX);
        V[i].z *= 100;
    }
}
void main()//тело программы
{
    Load("borus_full.txt", V); //Загрузка изолиний
```

```
    for (int i = 0; i < V.size();i ++)
    {
        p.add(V[i].x, V[i].y, V[i].z);
    }
    Delaunay(p, tr);
    RunOpenGL();
    auxMainLoop(display);//Запускаем основной цикл обработки событий.
}
```

## Приложение Е. Текст программы вычислительного алгоритма №2

```
#include <iostream>
#include <list>
#include "point.h"
#include <math.h>
#include <windows.h>
#include <vector>
#include <fstream>
#include <string>
#include <GL/gl.h>
#include <GL/glu.h>
#include "glaux.h"
using namespace std;
#pragma comment (lib,"opengl32.lib")
#pragma comment (lib,"glu32.lib")
#pragma comment (lib, "glaux.lib")
#pragma comment(lib, "legacy_stdio_definitions.lib")
//Описание типов данных:
int N = 150, M = N; // кол-во разбиений по Ox и по Oy
int CountT = 2 * N * M;
double x_min(0), x_max(0), y_min(0), y_max(0), z_min(0),
z_max(0);
double hx, hy;
const int CountColors = 20;
double RGB[CountColors][3] =
{
   {0,0,0},
   {0.15,0.07,0},
   {0.27,0.17,0.06},
   {0.44,0.26,0.17},
   {0.40,0.27,0.19},
   {0.37,0.21,0.08},
   {0.56,0.33,0.17},
   {0.45,0.29,0.17},
   {0.63,0.39,0.2},
   {0.79,0.62,0.25},
   {0.85,0.69,0.3},
   {0.77,0.76,0.38},
   {0.58,0.62,0.27},
   {0.35,0.46,0.15},
   {0.18,0.34,0.04},
   {0.1,0.25,0.01},
   {0.03,0.18,0.04},
   {0.0,0.14,0.06},
```

```
   {0,0.13,0.18},
   {0,0.09,0.08}
};
float anglez = 45, anglex = 0, rz = 200;
float ex = 0, ey = 10, ez = 0;
float ax = 0, ay = 0, az = 0;
struct Vershina //запись вершин сетки
{
   float R1, R2, H1, H2, H;
   bool b1, b2;
};
typedef vector<vector<Vershina>> VectorNodes;
typedef vector < vector<bool>> VectorRebro;
typedef list<pair<list<CPoint>, double> > IsoLines;
VectorNodes Nodes;
VectorRebro RebroV, RebroG, RebroN, RebroS;
IsoLines lines;
void Load(string filename, IsoLines& lines);//Загрузка
void Search_min_max(const IsoLines& lines);
void statistics(const IsoLines& points);
//стандартные функции для графики (отрисовки поверхности)
void CALLBACK PressKeyA(void) // шаг по кругу
{
   anglez += 0.1;
}
void CALLBACK PressKeyD(void) // шаг по кругу
{
   anglez -= 0.1;
}
void CALLBACK PressKeyW(void) // шаг вперед
{
   rz -= 1;
}
void CALLBACK PressKeyS(void)// шаг назад
{
   rz += 1;
}
void CALLBACK PressKeyQ(void) // шаг вверх
{
   ey += 10;
}
void CALLBACK PressKeyE(void) // шаг вниз
{
   ey -= 10;
}
```

```
void CALLBACK resize(int width, int height)
{
   glViewport(0,  0,  width,  height);//область  вывода
OpenGL-графики, совпадает с размером окна
   glMatrixMode(GL_PROJECTION);//включение режима работы с
матрицей проекций
   glLoadIdentity();//загрузка единичной матрицы, другими
словами сброс текущей матрицы
   gluPerspective(1500,  double(width)  /  height,  1,
1000);//Устанавливаем тип проекции. Задание перспективной
проекции
   glMatrixMode(GL_MODELVIEW);//включение режима работы с
модельно-видовой матрицей
}
void system(int n) //Функия - система координат
{
   glLineWidth(2);
   //ось OX - красный
   glBegin(GL_LINES);
   glColor3d(1, 0, 0);
   glVertex3d(-n, 0, 0);
   glVertex3d(n, 0, 0);
   glEnd();
   //стрелки на OX
   glBegin(GL_LINE_STRIP);
   glVertex3d(n - 2, 2, 0);
   glVertex3d(n, 0, 0);
   glVertex3d(n - 2, -2, 0);
   glEnd();
   //ось OY - синий
   glBegin(GL_LINES);
   glColor3d(0, 0, 1);
   glVertex3d(0, -n, 0);
   glVertex3d(0, n, 0);
   glEnd();
   //стрелки на OY
   glBegin(GL_LINE_STRIP);
   glVertex3d(-2, n - 2, 0);
   glVertex3d(0, n, 0);
   glVertex3d(2, n - 2, 0);
   glEnd();
   //ось OZ - зеленый
   glBegin(GL_LINES);
   glColor3d(0, 1, 0);
   glVertex3d(0, 0, n);
```

```
   glVertex3d(0, 0, -n);
   glEnd();
   //стрелки на OZ
   glBegin(GL_LINE_STRIP);
   glVertex3d(0, -2, n - 2);
   glVertex3d(0, 0, n);
   glVertex3d(0, 2, n - 2);
   glEnd();
}
void Load(string filename, IsoLines& lines)//Загрузка
{
   fstream Loading(filename.c_str());
   string str, str2;
   char symbol;
   double h, x, y, z;
   while (!Loading.eof())
   {
      do
      {
         getline(Loading, str, '\n');
         if (Loading.eof()) return;
      }
while (str != "[POLYLINE]");
      getline(Loading, str, '='); // Слово Type
      getline(Loading, str, '\n'); // Значение
      if (!(str == "0x20" || str == "0x21" || str ==
"0x22"))
      {
         continue;
      }
      getline(Loading, str, '='); // Label
      if (str != "Label") continue;
      Loading >> h;
      getline(Loading, str, '='); // Data0
      Loading.get();
      lines.push_back(make_pair(list<CPoint>(), h));
      while (1)
      {
         Loading >> y >> symbol >> x;
         do
         {
            Loading >> symbol;
            if (symbol == 'D')
               getline(Loading, str, '=');
```

```
         } while (!isdigit(Loading.peek()) && symbol != '[');
         lines.back().first.push_back(CPoint(x * 67000, y * 111000));
         if (symbol == '[' || symbol == 'E') break;
      }
      getline(Loading, str, '\n');
      Loading.get();
   }
}
void Search_min_max(const IsoLines& lines)
{
   IsoLines::const_iterator iter_lines = lines.begin();
   list<CPoint>::const_iterator p = iter_lines->first.begin();
   x_min = x_max = p->x;
   y_min = y_max = p->y;
   z_min = z_max = iter_lines->second;
   double h;
   for (iter_lines = lines.begin(); iter_lines != lines.end(); ++iter_lines)
   {
      // Цикл по точкам изолиней
      h = iter_lines->second;
      if (h < z_min)
         z_min = h;
      if (h > z_max)
         z_max = h;
      for (p = iter_lines->first.begin(); p != iter_lines->first.end(); ++p)
      {
         if (p->x < x_min) x_min = p->x;
         if (p->x > x_max) x_max = p->x;
         if (p->y < y_min) y_min = p->y;
         if (p->y > y_max) y_max = p->y;
      }
   }
   hx = (x_max - x_min) / N;
   hy = (y_max - y_min) / M;
}

void statistics(const IsoLines& points)
{
   cout << "Количество изолиний = " << points.size() << endl;
```

```
   IsoLines::const_iterator iter_lines;
   long long Count(0);
   for (iter_lines = points.begin(); iter_lines != points.end(); ++iter_lines)
   {
      Count += iter_lines->first.size();
   }
   cout << "Общее количесто точек = " << Count << endl;
   cout << "Минимальная высота = " << z_min << endl;
   cout << "Максимальная высота = " << z_max << endl;
}
bool Peresechenie(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4, double &r1, double &r2)
{
   double v1, v2, v3, v4;
   v1 = (x4 - x3) * (y1 - y3) - (y4 - y3) * (x1 - x3);
   v2 = (x4 - x3) * (y2 - y3) - (y4 - y3) * (x2 - x3);
   v3 = (x2 - x1) * (y3 - y1) - (y2 - y1) * (x3 - x1);
   v4 = (x2 - x1) * (y4 - y1) - (y2 - y1) * (x4 - x1);
   double Px, Py;
   Px = x1 + (x2 - x1) * fabs(v1)/fabs(v2 - v1);
   Py = y1 + (y2 - y1) * fabs(v1)/fabs(v2 - v1);
   if ( (v1*v2<=0) && (v3*v4<=0))
   {
      r1 = sqrt((Px - x3) * (Px - x3) + (Py - y3) * (Py - y3));
      r2 = sqrt((Px - x4) * (Px - x4) + (Py - y4) * (Py - y4));
      return true;
   }
   else
   {
      return false;
   }
}
bool SetVershina(int i, int j, double distance, double h, VectorRebro& Rebro)
{
   Rebro[i][j] = true;
   if (!Nodes[i][j].b1)// 1 случай
   {
      Nodes[i][j].b1 = true;
      Nodes[i][j].R1 = distance;
      Nodes[i][j].H1 = h;
```

```
      return true;
   }
   else if (!Nodes[i][j].b2) // 2 случай
   {
      if (Nodes[i][j].H1 != h)
      {
         Nodes[i][j].b2 = true;
         Nodes[i][j].R2 = distance;
         Nodes[i][j].H2 = h;
         return true;
      }
      else
      {
         Nodes[i][j].b1 = true;
         Nodes[i][j].R1 = min(distance, Nodes[i][j].R1);
         Nodes[i][j].H1 = h;
         return true;
      }
   }
   else
      if  (!(distance  >  Nodes[i][j].R1  &&  distance  >
Nodes[i][j].R2)) // 3 случай
   {
      if ((Nodes[i][j].H1 != h) && (Nodes[i][j].H2 != h))
// если все 3 узла разных уровней
      {
         if (Nodes[i][j].R1 < Nodes[i][j].R2) // ??????
         {
            Nodes[i][j].R2 = distance;
            Nodes[i][j].H2 = h;
            return true;
         }
         else
         {
            Nodes[i][j].R1 = distance;
            Nodes[i][j].H1 = h;
            return true;
         }
      }
      else // если уровень новго узла равен уровню какого-
то друго узла из 2
      {
         if (Nodes[i][j].H1 == h)  // если уровень нового
узла равен уровню 1 узла
         {
```

```
            Nodes[i][j].R1        =        min(distance,
Nodes[i][j].R1);
            Nodes[i][j].H1 = h;
            return true;
         }
         if (Nodes[i][j].H2 == h)// если уровень нового
узла равен уровню 2 узла
         {
            Nodes[i][j].R2        =        min(distance,
Nodes[i][j].R2);
            Nodes[i][j].H2 = h;
            return true;
         }
      }
   }
   return false;
}
void FirstStage(IsoLines& lines, VectorRebro& Rebro, int
ykaz)
{
   bool f;
   int i, j, i0, j0, i1, j1;
   double distance1, distance2;
   double h, x_1, x_2, y_1, y_2, x3, x4, y3, y4;
   IsoLines::iterator iter_lines;
   for   (iter_lines   =   lines.begin();   iter_lines   !=
lines.end(); ++iter_lines)
   {
      h = iter_lines->second;
      // Цикл по точкам изолиней
      list<CPoint>::iterator     p     =     iter_lines-
>first.begin()++;
      list<CPoint>::reverse_iterator p_end;
      list<CPoint>::iterator q = ++p;
      for   (p   =   iter_lines->first.begin();   p   !=   --
iter_lines->first.end(); )
      {
         x_1 = min(p->x, q->x);
         x_2 = max(p->x, q->x);
         y_1 = min(p->y, q->y);
         y_2 = max(p->y, q->y);
         i0 = (x_1 - x_min) / hx;
         i1 = (x_2 - x_min) / hx;
         j0 = (y_1 - y_min) / hy;
         j1 = (y_2 - y_min) / hy;
```

```
         if (ykaz == 1)
         {
            for (i = i0; i <= i1 && i <= N; i++)
               for (j = j0; j <= j1 && j < M; j++)
            {
               x3 = x4 = x_min + i * hx;
               y3 = y_min + j * hy;
               y4 = y_min + (j + 1) * hy;

               f = Peresechenie(p->x, p->y, q->x, q->y,
x3, y3, x4, y4, distance1, distance2);
               if (f)
               {
                  Rebro[i][j] = true;
                  SetVershina(i, j, distance1, h, Rebro);
                  SetVershina(i,   j+1,   distance2,   h,
Rebro);
               }
            }
         }
         else
            if (ykaz == 2)
            {
               for (i = i0; i <= i1 && i< N; i++)
                  for (j = j0; j <= j1 && j<= M; j++)
                  {
                     x3 = x_min +  i* hx;
                     x4 = x_min + (i + 1) * hx;
                     y3 = y4 = y_min + j * hy;

                     f  =  Peresechenie(p->x,p->y,q->x,q-
>y,x3,y3,x4,y4,distance1,distance2);
                     if (f)
                     {
                        Rebro[i][j] = true;
                        SetVershina(i,  j,  distance1,  h,
Rebro);
                        SetVershina(i+1, j, distance2, h,
Rebro);

                     }
                  }
            }
            else
               if (ykaz == 3)
```

```
                {
                   for (i = i0; i <= i1 && i < N; i++)
                      for (j = j0; j <= j1 && j < M; j++)
                      {
                         x3 = x_min + i * hx;
                         x4 = x_min + (i + 1) * hx;
                         y3 = y_min + j * hy;
                         y4 = y_min + (j + 1) * hy;

                         f   =   Peresechenie(p->x,p->y,q-
>x,q->y,x3,y3,x4,y4,distance1,distance2);
                         if (f)
                         {
                            Rebro[i][j] = true;
                            SetVershina(i, j, distance1, h,
Rebro);
                            SetVershina(i+1,          j+1,
distance2, h, Rebro);
                         }
                      }
                }
                else
                   if (ykaz == 4)
                   {
                      for (i = i0; i <= i1 && i < N; i++)
                         for (j = j0; j <= j1 && j < M; j++)
                         {
                            x3 = x_min + i * hx;
                            x4 = x_min + (i + 1) * hx;
                            y3 = y_min + (j+1) * hy;
                            y4 = y_min + j * hy;

                            f  =  Peresechenie(p->x,p->y,q-
>x,q->y,x3,y3,x4,y4,distance1,distance2);
                            if (f)
                            {
                               Rebro[i][j] = true
                               SetVershina(i,          j+1,
distance1, h, Rebro);
                               SetVershina(i+1,          j,
distance2, h, Rebro);
                            }
                         }
                   }
          p = q;          // Передвижение итераторов
```

```
            if (q != iter_lines->first.end())
               ++q;
         }
         p = iter_lines->first.begin();
         p_end = iter_lines->first.rbegin();
         x_1 = min(p->x, p_end->x);
         x_2 = max(p->x, p_end->x);
         y_1 = min(p->y, p_end->y);
         y_2 = max(p->y, p_end->y);
         i0 = (x_1 - x_min) / hx;
         i1 = (x_2 - x_min) / hx;
         j0 = (y_1 - y_min) / hy;
         j1 = (y_2 - y_min) / hy;
         if (ykaz == 1)
         {
            for (i = i0; i <= i1 && i <= N; i++)
               for (j = j0; j <= j1 && j < M; j++)
            {
               x3 = x4 = x_min + i * hx;
               y3 = y_min + j * hy;
               y4 = y_min + (j + 1) * hy;
               f   =   Peresechenie(p->x,p->y,p_end->x,p_end-
>y,x3,y3,x4,y4, distance1, distance2);
               if (f)
               {
                  Rebro[i][j] = true;
                  SetVershina(i, j, distance1, h, Rebro);
                  SetVershina(i, j + 1, distance2, h, Rebro);
               }
            }
         }
         else
            if (ykaz == 2)
            {
               for (i = i0; i <= i1 && i < N; i++)
                  for (j = j0; j <= j1 && j <= M; j++)
                  {
                     x3 = x_min + i * hx;
                     x4 = x_min + (i + 1) * hx;
                     y3 = y4 = y_min + j * hy;
                     f = Peresechenie(p->x, p->y, p_end->x,
p_end->y, x3, y3, x4, y4, distance1, distance2);
                     if (f)
                     {
                        Rebro[i][j] = true;
```

```
                        SetVershina(i,   j,   distance1,   h,
Rebro);
                        SetVershina(i+1,  j,  distance2,  h,
Rebro);
                     }
                  }
            }
            else
               if (ykaz == 3)
               {
                  for (i = i0; i <= i1 && i < N; i++)
                     for (j = j0; j <= j1 && j < M; j++)
                     {
                        x3 = x_min + i * hx;
                        x4 = x_min + (i + 1) * hx;
                        y3 = y_min + j * hy;
                        y4 = y_min + (j + 1) * hy;

                        f = Peresechenie(p->x, p->y, p_end-
>x, p_end->y, x3, y3, x4, y4, distance1, distance2);
                        if (f)
                        {
                           Rebro[i][j] = true;
                           SetVershina(i,  j,  distance1,  h,
Rebro);
                           SetVershina(i+1,  j+1,  distance2,
h, Rebro);
                        }
                     }
               }
               else
                  if (ykaz == 4)
                  {
                     for (i = i0; i <= i1 && i < N; i++)
                        for (j = j0; j <= j1 && j < M; j++)
                        {
                           x3 = x_min + i * hx;
                           x4 = x_min + (i + 1) * hx;
                           y3 = y_min + (j + 1) * hy;
                           y4 = y_min + j * hy;
                           f   =   Peresechenie(p->x,   p->y,
p_end->x, p_end->y, x3, y3, x4, y4, distance1, distance2);
                           if (f)
                           {
                              Rebro[i][j] = true;
```

```
                                SetVershina(i, j+1, distance1,
h, Rebro);
                                SetVershina(i+1, j, distance2,
h, Rebro);
                            }
                        }
                }
    }
}
void SecondStage(IsoLines& lines, VectorRebro& RebroV,
VectorRebro& RebroG, VectorRebro& RebroN, VectorRebro&
RebroS, bool& otmetka)
{
    int i, j, jj, ii;
    double NewDistance0, NewDistance, NewH;
    otmetka = false; // предполагаем, что больше не осталось
непомеченных ребер т.е все ребра имеют пересечения
    // проходим по всем узлам
    for (i = 0; i <= N; i++)
        for (j = 0; j <= M; j++)
        {
            if (Nodes[i][j].b1 == true || Nodes[i][j].b2 ==
true) // если узлу уже приписаны расстояния (хотя бы 1)
            {
                if (Nodes[i][j].b1 && Nodes[i][j].b2)
                {
                    if (Nodes[i][j].R1 < Nodes[i][j].R2)
                    {
                        NewDistance = Nodes[i][j].R1;
                        NewH = Nodes[i][j].H1;
                    }
                    else
                    {
                        NewDistance = Nodes[i][j].R2;
                        NewH = Nodes[i][j].H2;
                    }
                }
                else if (Nodes[i][j].b1)
                {
                    NewDistance = Nodes[i][j].R1;
                    NewH = Nodes[i][j].H1;
                }
                else
                {
                    NewDistance = Nodes[i][j].R2;
```

```
                NewH = Nodes[i][j].H2;
            }
          //Выпускаем волну в трех направлениях
            if (i > 0 && j > 0 && !RebroN[i - 1][j - 1])
// Ю-С (влево вниз)
            {
                ii = i - 1;
                jj = j -1;
                NewDistance0 = NewDistance;
                while (RebroN[ii][jj] == false && ii > 0 &&
jj > 0 )// пока ребро не было отмечено
                {
                    NewDistance0 += sqrt(hy*hy + hx*hx);
                    otmetka    +=    SetVershina(ii,    jj,
NewDistance0, NewH, RebroN);
                    ii--;
                    jj--;
                }
            }
            if (i > 0 && !RebroG[i - 1][j]) //влево
            {
                ii = i - 1;
                NewDistance0 = NewDistance;
                while (ii > 0 && RebroG[ii][j] == false  )
// пока ребро не было отмечено
                {
                    NewDistance0 += hx;
                    otmetka    +=    SetVershina(ii,    j,
NewDistance0, NewH, RebroG);
                    ii--;
                }
            }
            if (i > 0 && j < M && !RebroS[i - 1][j + 1])
// С-Ю (влево вверх)
            {
                ii = i - 1;
                jj = j + 1;
                NewDistance0 = NewDistance;
                while (ii > 0 && jj < M && RebroS[ii][jj]
== false  ) // пока ребро не было отмечено
                {
                    NewDistance0 += sqrt(hy * hy + hx * hx);
                    otmetka    +=    SetVershina(ii,    jj,
NewDistance0, NewH, RebroS);
                    ii--;
```

```
                jj++;
            }
        }
        if (j < M && !RebroV[i][j + 1])// вверх
        {
            jj = j + 1;
            NewDistance0 = NewDistance;
            while (jj < M && RebroV[i][jj] == false )
// пока ребро не было отмечено
            {
                NewDistance0 += hy;
                otmetka    +=    SetVershina(i,    jj,
NewDistance0, NewH, RebroV);
                jj++;
            }
        }
        if (i < N && j < M && !RebroN[i+1][j+1])// Ю-
С (вправо вверх)
        {
            ii = i + 1;
            jj = j + 1;
            NewDistance0 = NewDistance;
            while (ii < N && jj < M && RebroN[ii][jj]
== false) // пока ребро не было отмечено
            {
                NewDistance0 += sqrt(hy * hy + hx * hx);
                otmetka    +=    SetVershina(ii,    jj,
NewDistance0, NewH, RebroN);
                ii++;
                jj++;
            }
        }
        if (i < N && !RebroG[i+1][j])// вправо
        {
            ii = i + 1;
            NewDistance0 = NewDistance;
            while (ii < N && RebroG[ii][j] == false) //
пока ребро не было отмечено
            {
                NewDistance0 += hx;
                otmetka    +=    SetVershina(ii,    j,
NewDistance0, NewH, RebroG);
                ii++;
            }
        }
```

```
            if (i < N && j>0 && !RebroS[i + 1][j - 1])//
С-Ю (вправо вниз)
            {
               ii = i + 1;
               jj = j - 1;
               NewDistance0 = NewDistance;
               while (ii < N && jj > 0 && RebroS[ii][jj]
== false ) // пока ребро не было отмечено
               {
                  NewDistance0 += sqrt(hy * hy + hx * hx);
                  otmetka    +=    SetVershina(ii,    jj,
NewDistance0, NewH, RebroS);
                  ii++;
                  jj--;
               }
            }
            if (j > 0 && !RebroV[i][j - 1])// вниз
            {
               jj = j - 1;
               NewDistance0 = NewDistance;
               while (jj > 0 && RebroV[i][jj] == false )
// пока ребро не было отмечено
               {
                  NewDistance0 += hy;
                  otmetka     =     SetVershina(i,     jj,
NewDistance0, NewH, RebroV);
                  jj--;
               }
            }
         }
      }
}
void DrawTriangle(GLdouble x1, GLdouble y1, GLdouble z1,
GLdouble  x2,  GLdouble  y2,  GLdouble  z2,  GLdouble  x3,
GLdouble  y3,  GLdouble  z3,  GLdouble  r1,  GLdouble  g1,
GLdouble  b1,  GLdouble  r2,  GLdouble  g2,  GLdouble  b2,
GLdouble r3, GLdouble g3, GLdouble b3)
{ // передаются координаты и цвета трех точек треугольника
   glBegin(GL_TRIANGLES);
   glColor3d(r1, g1, b1);  glVertex3d(x1, y1, z1);
   glColor3d(r2, g2, b2);  glVertex3d(x2, y2, z2);
   glColor3d(r3, g3, b3);  glVertex3d(x3, y3, z3);
   glEnd();
}
```

```
void TRIANGLES_Kontur(float x1, float y1, float z1, float
x2, float y2, float z2, float x3, float y3, float z3, float
r, float b, float g)
{
   glPolygonMode(GL_FRONT_AND_BACK,GL_LINE);//режим
растеризации(заливка, контур или точки вершин)- контур
   glLineWidth(4);
   glBegin(GL_TRIANGLES);
   glColor3d(r, g, b);
   glVertex3f(x1, y1, z1);
   glVertex3f(x2, y2, z2);
   glVertex3f(x3, y3, z3);
   glEnd();
   glPolygonMode(GL_FRONT_AND_BACK,
GL_FILL);//восстановление режима по умолчанию - заливка
}
void drawtriangle()
{
   //Цикл по треугольникам
   float H;
   int k1,k2,k3;
   float r1, r2, r3, g1, g2, g3, b1, b2, b3;
   H = (z_max - z_min) / CountColors;
   //масштабирование
   float DX = (x_max + x_min) / 2;
   float DY = (y_max + y_min) / 2;
   float DZ = (z_max + z_min) / 2;
   float MAXXX = max((z_max - z_min) / 2, max((x_max -
x_min) / 2, (y_max - y_min) / 2));
   float K = hy / hx;
   for (int i = 1; i < N-1; i++)
      for (int j = 1; j < M-1; j++)
      {
         // нижний треугольник
         k1 = (z_max - Nodes[i][j].H) / H;
         if (k1 == CountColors) k1 = CountColors - 1;
         r1 = RGB[k1][0]; g1 = RGB[k1][1]; b1 = RGB[k1][2];
         k2 = (z_max - Nodes[i + 1][j].H) / H;
         if (k2 == CountColors) k2 = CountColors - 1;
         r2 = RGB[k2][0]; g2 = RGB[k2][1]; b2 = RGB[k2][2];
         k3 = (z_max - Nodes[i + 1][j + 1].H) / H;
         if (k3 == CountColors) k3 = CountColors - 1;
         r3 = RGB[k3][0]; g3 = RGB[k3][1]; b3 = RGB[k3][2];
         //рисую треугольник
         DrawTriangle(
```

```
            (x_min+i*hx-DX)/MAXXX*100*K, (y_min + j * hy-
DY) / MAXXX * 100, (Nodes[i][j].H-DZ) / MAXXX * 100*2,
            (x_min + (i+1) * hx - DX) / MAXXX * 100 * K,
(y_min + j * hy-DY) / MAXXX * 100,  (Nodes[i+1][j].H - DZ)
/ MAXXX * 100 * 2,
            (x_min + (i + 1) * hx - DX) / MAXXX * 100 * K,
(y_min + (j+1) * hy-DY) / MAXXX * 100, (Nodes[i+1][j+1].H
- DZ) / MAXXX * 100 * 2,
               r1, g1, b1,
               r2, g2, b2,
               r3, g3, b3);
            // верхний треугольник
            k1 = (z_max - Nodes[i][j].H) / H;
            if (k1 == CountColors) k1 = CountColors - 1;
            r1 = RGB[k1][0]; g1 = RGB[k1][1]; b1 = RGB[k1][2];
            k2 = (z_max - Nodes[i][j + 1].H) / H;
            if (k2 == CountColors) k2 = CountColors - 1;
            r2 = RGB[k2][0]; g2 = RGB[k2][1]; b2 = RGB[k2][2];
            k3 = (z_max - Nodes[i + 1][j + 1].H) / H;
            if (k3 == CountColors) k3 = CountColors - 1;
            r3 = RGB[k3][0]; g3 = RGB[k3][1]; b3 = RGB[k3][2];
            //рисую треугольник
            DrawTriangle(
               (x_min + i * hx - DX) / MAXXX * 100 * K, (y_min
+ j * hy - DY) / MAXXX * 100, (Nodes[i][j].H - DZ) / MAXXX
* 100 * 2,
               (x_min + i * hx - DX) / MAXXX * 100 * K, (y_min
+ (j+1) * hy - DY) / MAXXX * 100, (Nodes[i][j+1].H - DZ) /
MAXXX * 100 * 2,
               (x_min + (i + 1) * hx - DX) / MAXXX * 100 * K,
(y_min + (j + 1) * hy - DY) / MAXXX * 100, (Nodes[i + 1][j
+ 1].H - DZ) / MAXXX * 100 * 2,
               r1, g1, b1,
               r2, g2, b2,
               r3, g3, b3);
      }
}
void scene()
{
   glRotated(-90, 1, 0, 0);
   glRotatef(0.45f, 0.0f, 0.0f, 1.0f); //поворачиваем по
школе z на 45 градусов
   glPushMatrix();//сохранили текущую систему координат
   drawtriangle();
   glPopMatrix();
```

```
}
void CALLBACK display(void)
{
   glClear(GL_COLOR_BUFFER_BIT   |   GL_DEPTH_BUFFER_BIT);
//очищаем буфер цвета и буфер глубины
   glLoadIdentity();
   ex = rz * sin(anglez);
   ez = rz * cos(anglez);
   gluLookAt(ex, ey, ez, ax, ay, az, 0, 1, 0);
   scene();
   auxSwapBuffers();//эта  команда  выводит  содержимого
буфера на экран
}
void RunOpenGL()
{
   auxKeyFunc(AUX_w, PressKeyW);
   auxKeyFunc(AUX_s, PressKeyS);
   auxKeyFunc(AUX_a, PressKeyA);
   auxKeyFunc(AUX_d, PressKeyD);
   auxKeyFunc(AUX_q, PressKeyQ);
   auxKeyFunc(AUX_e, PressKeyE);
   auxInitPosition(0, 0, 1000, 1000);// верхний левый угол
(0,0), ширина и высота - 1000
   auxInitDisplayMode(AUX_RGB | AUX_DEPTH | AUX_DOUBLE);//
устанавливаем параметры контекста OpenGL
   auxInitWindow(L"Glaux  Template");//  создаем  окно  на
экране
   // наше окно будет получать сообщения от клавиатуры,
мыши, таймера или любые другие
   // когда никаких сообщений нет, будет вызываться функция
display. Так мы получаем анимацию
   auxIdleFunc(display);
   //регистрируем  resize
   auxReshapeFunc(resize);
   glClearColor(1, 1, 1, 0); //цвет фона
   glEnable(GL_DEPTH_TEST);//включить тест глубины
}
//тело программы
void main()
{
   int ykaz; Search_min_max(lines); statistics(lines); int
i, j;
   setlocale(LC_ALL, "RUS");
   Load("borus_full.txt",    lines);//Загрузка    реальной
поверхности
```

```
   cout << "Изолинии загружены!\n";
   cout << "____________________ЭТАП №1____________________"
<< endl;
   Nodes.resize(N + 1);
   RebroV.resize(N + 1);
   RebroG.resize(N + 1);
   RebroN.resize(N + 1);
   RebroS.resize(N + 1);
   for (i = 0; i <= N; i++)
   {
      Nodes[i].resize(M + 1);
      RebroV[i].resize(M + 1);
      RebroG[i].resize(M + 1);
      RebroN[i].resize(M + 1);
      RebroS[i].resize(M + 1);
      for (j = 0; j <= M; j++)
      {
         Nodes[i][j].b1 = Nodes[i][j].b2 = false;
         Nodes[i][j].R1 = Nodes[i][j].R2 = 0;
         RebroV[i][j] = false;
         RebroG[i][j] = false;
         RebroN[i][j] = false;
         RebroS[i][j] = false;
      }
   }
   cout << "___________ПЕРЕСЕЧЕНИЯ С ВЕРТИКАЛЬНЫМИ РЕБРАМИ
СЕТКИ___________ " << endl;
   FirstStage(lines, RebroV,1);
   cout  <<  "___________ПЕРЕСЕЧЕНИЯ  С  ГОРИЗОНТАЛЬНЫМИ
РЕБРАМИ СЕТКИ___________ " << endl;
   FirstStage(lines, RebroG,2);
   cout  <<  "___________ПЕРЕСЕЧЕНИЯ  С  ДИАГОНАЛЬНЫМИ  Ю-С
РЕБРАМИ СЕТКИ___________ " << endl;
   FirstStage(lines, RebroN,3);
   cout  <<  "___________ПЕРЕСЕЧЕНИЯ  С  ДИАГОНАЛЬНЫМИ  С-Ю
РЕБРАМИ СЕТКИ___________ " << endl;
   FirstStage(lines, RebroS,4);
   cout << "____________________ЭТАП №2____________________"
<< endl;
   bool otmetka = true;
   while (otmetka)
      {
         SecondStage(lines,   RebroV,   RebroG,   RebroN,
RebroS,otmetka);
      }
```

```
   //вычисляем высоту в узлах с двумя значениями
   for (i = 0; i <= N; i++)
      for (j = 0; j <= M; j++)
      {
         if (Nodes[i][j].b1 && Nodes[i][j].b2)
         {
            Nodes[i][j].H        =        (Nodes[i][j].H1*
Nodes[i][j].R2         +          Nodes[i][j].H2          *
Nodes[i][j].R1)/(Nodes[i][j].R1 + Nodes[i][j].R2);
         }
         else if (Nodes[i][j].b1)
            {
               Nodes[i][j].H = Nodes[i][j].H1;
            }
            else
            {
               Nodes[i][j].H = Nodes[i][j].H2;
            }
      }
   RunOpenGL();
   auxMainLoop(display);//Запускаем     основной     цикл
обработки событий.
   system("pause");
}
```

Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

**УТВЕРЖДАЮ**
Заведующий кафедрой
_____ / В.В. Шайдуров

«___» _____2020 г.

# БАКАЛАВРСКАЯ РАБОТА

**Направление** 02.03.01 «Математика и компьютерные науки»

## ПРИМЕНЕНИЕ ТРИАНГУЛЯЦИОННЫХ НЕРЕГУЛЯРНЫХ СЕТОК ДЛЯ СОЗДАНИЯ МОДЕЛЕЙ РЕЛЬЕФА

Научный руководитель
кандидат физико-математических наук,
доцент _____ / Е.В. Кучунова
23.06.2020

Выпускник _____ 23.06.20 / Ю.В. Лобко

Красноярск 2020