# Содержание

[Содержание 3](#_Toc129190139)

[Введение 4](#_Toc129190140)

[1. Анализ объекта 5](#_Toc129190141)

[2. Разработка алгоритма работы программы 6](#_Toc129190142)

[3. Описание основных компонентов программы и последовательности разработки 7](#_Toc129190143)

[4. Исходный код программы 8](#_Toc129190144)

[5. Проверка корректности работы программы в различных режимах 19](#_Toc129190145)

[Заключение 26](#_Toc129190146)

[Литература 27](#_Toc129190147)

# Введение

В настоящее время использование треугольных сеток широко распространено в различных областях науки и техники, таких как математическое моделирование, компьютерная графика, механика, геометрия и многие другие. Они позволяют представлять сложные объекты и поверхности в виде набора треугольников, что упрощает их анализ и обработку.

Цель данной курсовой работы заключается в разработке приложения для формирования треугольной сетки разбиения для заданной двумерной области. В рамках работы будет рассмотрен алгоритм построения такой сетки, а также реализовано приложение, которое будет позволять пользователю задавать геометрическую область и получать треугольную сетку ее разбиения.

Для достижения поставленной цели в работе будут использоваться следующие методы и подходы: изучение существующих алгоритмов построения треугольных сеток, выбор оптимального метода для данной задачи, программная реализация алгоритма и создание удобного пользовательского интерфейса для работы с приложением.

В итоге разработанное приложение будет представлять собой удобный инструмент для быстрого и эффективного построения треугольной сетки разбиения для заданной области, что может быть полезным во многих областях науки и техники.

,

# Анализ объекта

Анализ объекта – это процесс изучения особенностей и характеристик объекта, который является предметом исследования. В данном случае, объектом анализа является приложение для формирования треугольных сеток разбиения двумерной области.

Для начала, необходимо определить, что такое треугольная сетка. Треугольная сетка – это геометрическая сетка, состоящая из треугольных элементов, которая используется для аппроксимации геометрических фигур. Она широко применяется в различных областях, таких как компьютерная графика, численное моделирование, анализ данных и многих других.

Приложение для формирования треугольных сеток разбиения двумерной области является инструментом, который позволяет создавать такие сетки для заданной области. Оно может быть использовано для моделирования различных процессов, таких как распространение тепла, гидродинамика, механика твердых тел и т.д.

Одним из основных преимуществ треугольных сеток является их гибкость и адаптивность. Они могут быть созданы для любой формы области и могут быть адаптированы для различных задач. Кроме того, треугольные сетки обладают высокой точностью и устойчивостью, что делает их идеальным выбором для многих приложений.

Треугольные сетки - это геометрические структуры, состоящие из треугольных элементов, которые используются в компьютерной графике, вычислительной физике, инженерии и других областях.

Они используются для представления сложных форм и поверхностей в трехмерном пространстве, а также для численного решения дифференциальных уравнений. Идея использования треугольных сеток возникла еще в 19 веке, когда была разработана теория конечных элементов, которая позволяет решать сложные инженерные задачи.

Эффективное решение задач, связанных с проектированием технологии в процессах обработки металлов давлением (ОМД), требует проведения дорогостоящего экспериментального исследования, вместо которого часто прибегают к имитационному моделированию. В силу высокой достоверности результатов и возможности решения ряда сопутствующих задач наиболее эффективным средством моделирования формоизменения является метод конечных элементов (МКЭ). Широкое внедрение в практику анализа конечно-элементной технологии привело к появлению соответствующих пакетов прикладных программ; в задачах моделирования пластического формоизменения таковыми являются DEFORM, QForm, Simufact.forming и ряд других. Несмотря на наличие профессиональных инструментов, исследования в данной области по-прежнему актуальны в связи с высокой стоимостью и малой доступностью существующего программного обеспечения, закрытостью применяемого математического аппарата и невозможностью моделирования в ситуациях, требующих специального описания граничных условий.

Применение МКЭ требует построения «сетки» элементов – топологического множества точек (также используются термины «узел» и «вершина»), связанных между собой ребрами – отрезками прямых или кривых линий таким образом, что исходная дву- или трехмерная область разбивается на элементы заданной формы. В качестве элементов сетки чаще всего используются геометрические симплексы – треугольники в двумерном и тетраэдры в трехмерном случае. А сам процесс построения сетки принято называть дискретизацией, или триангуляцией. В настоящее время двумерная триангуляция без адаптации к конкретной предметной области считается решенной проблемой.

Актуальными остаются исследования, ориентированные на формирование сеток в специфических условиях. Одной из таких областей является моделирование пластического формоизменения, при котором возникают значительные градиенты скоростей деформации и температур, а деформируемая заготовка принимает сложную форму. Из всего многообразия методов для дискретизации сложных областей наиболее предпочтительными являются два класса – это методы на основе критерия Делоне и методы исчерпывания.

Методы на основе критерия Делоне основаны на формировании триангуляции по существующему множеству опорных точек, которые становятся узлами сетки. Триангуляция Делоне на плоскости – это множество непересекающихся треугольников, в котором ни одна вершина, не принадлежащая данному треугольнику, не попадает в описанную вокруг этого треугольника окружность. В двумерном случае этот подход является наиболее популярным, поскольку позволяет эффективно создавать высококачественные сетки. Однако его применение требует рационального решения сложной задачи первичного разбиения – построения множества опорных точек.

Сущность методов исчерпывания заключается в последовательном исключении элементов из заданной области до тех пор, пока вся область не будет исчерпана. В англоязычной литературе данный подход принято называть «advancing front» – продвигающийся фронт, что полностью соответствует идее метода. Исходными данными метода на любой итерации является «фронт» – граница еще не исчерпанной части области. На каждой итерации изымается один или более элемент, после чего «фронт» обновляется, и производится следующая итерация.

Идея методов исчерпывания была предложена и развита в конце 80-х годов прошлого века в работах Дж. Перейры, К. Моргана, С.Х. Ло, Р. Лонера и др. Подобные алгоритмы разрабатывались и отечественными учеными. Относительно недавно И.А. Щегловым были предложены алгоритмы «от угла» и «от ребра», позволяющие дискретизировать дву- и трехмерные области сложной формы. Данная работа посвящена адаптации и уточнению алгоритма «от угла» для построения триангуляции двумерной области в задачах моделирования пластического формоизменения, поскольку он в исходном варианте не позволяет генерировать качественную сетку для рассматриваемых областей.

Рассмотрим основные требования, предъявляемые к алгоритму триангуляции при решении задач моделирования пластической деформации:

* сетка должна обеспечивать перемещение элементов вдоль границы инструмента, в том числе на участках с изменением направления течения металла, где на каждом элементе границы должно находиться несколько узлов сетки;
* сетка должна строиться в сложных для затекания металла участках (например, в облойной канавке при горячей штамповке);
* сетка должна иметь возможность сгущения в областях с высокими градиентами скоростей деформации;
* форма элементов должна стремиться к правильной (равносторонний треугольник) для повышения стабильности расчетов;
* плотность сетки должна быть контролируемой, т.е. пользователь должен иметь возможность увеличить количество элементов, если это позволяют вычислительные возможности, но сетка всегда должна оставаться качественной.

Блок–схема предлагаемого алгоритма приведена на рис. 1. Рассмотрим основные его этапы. Исходной информацией для построения триангуляции является кусочно–линейная кривая, охватывающая заполняемую треугольниками область.

Для контроля плотности сетки необходимо определить желаемый размер элементов сетки, а точнее среднюю длину ребра (длину стороны треугольника); эту величину принято называть шагом триангуляции (блок 2). С точки зрения пользователя удобнее задавать не размеры элементов, а их количество. Использование итерационного метода с возможностью сгущения сетки не позволяет заранее спрогнозировать количество элементов. Можно рассчитать верхнее значение глобального шага триангуляции исходя из заданного количества элементов и площади деформируемой заготовки с учетом правильной формы элемента:

,

где h – шаг триангуляции; S3 – площадь сечения заготовки; Q – заданное количество элементов. Рассчитанная таким образом величина является ориентировочной и корректируется алгоритмом в зависимости от сложности участка области, в котором производится построение элемента, что ведет к увеличению количества элементов относительно указанного пользователем.

Diagram

Description automatically generated

Рис. 1. Блок-схема алгоритма двумерной триангуляции методом исчерпывания

Основным понятием алгоритмов исчерпывания является фронт, который представляется замкнутой последовательностью узлов. Первоначально фронт формируется на основе кусочно-линейной границы, являющейся сечением деформируемой заготовки, исходя из шага триангуляции и дополнительных условий (блок 3). В ходе триангуляции фронт постепенно сжимается вокруг незаполненной части области, при этом может происходить его разделение на два и большее количество фронтов. Узлы начального фронта в дальнейшем становятся узлами элементов и должны быть найдены так, чтобы максимально соответствовать решаемой задаче. При моделировании пластического формоизменения целесообразно исходить из условия, что на каждом отрезке кусочно-линейной границы должно находиться минимум 2 элемента сетки, кроме того, следует уменьшить размеры элементов в участках сложной формы, таких, как облой и радиусы закругления. С этой целью для каждого отрезка i границы рассчитывается делитель ni, определяющий шаг узлов mi=li/ni. При расчете делителя учитывается длина отрезка li , шаг триангуляции h и расстояние до ближайших линий границы в направлении, ортогональном каждому отрезку границы s. На рис. 2, а показано кусочно-линейное представление границы деформируемой заготовки, а на рис. 2, б – узлы исходного фронта для той же области.

|  |  |
| --- | --- |
| Diagram  Description automatically generated  а) | Diagram  Description automatically generated  б) |
| Рис. 2. Кусочно-линейная граница (а) и исходный фронт триангуляции (б) | |

Здесь для отрезка 1 нет дополнительных ограничений, и делитель определяется как округленное до целого числа отношение его длины к шагу триангуляции: . В случае коротких отрезков 2 и 3 с длиной менее шага триангуляции n 2,3< h выбирается минимальный делитель: n 2,3< 2. На отрезке 4 ограничивающим фактором является малое расстояние до противолежащего элемента, и делитель рассчитывается так, чтобы в данном участке размещались три слоя элементов: . Для исключения ситуации с резким изменением размеров соседних элементов на внешней границе области, ведущей к появлению очень узких треугольников (например, на стыке отрезков 1 и 2), производится сглаживание фронта, заключающееся во вставке дополнительных узлов между уже существующими: если расстояние от любого узла до двух соседних отличается более чем в 2 раза, то больший отрезок делится пополам путем вставки дополнительного узла.

Участок алгоритма с 4 по 16 блоки является его основным циклом. Построение элементов сетки начинается с выбора самого сложного участка среди всех фронтов (блок 4), для чего в каждом узле фронта Ni производится расчет сложности Ci по формуле

A picture containing text, clock, screenshot

Description automatically generated,

где lcp – среднее расстояние между узлами фронта; li-1, li – длины отрезков фронта, прилегающих к узлу Ni (рис. 3); Ka=2 – весовой коэффициент значимости угловой составляющей сложности; ai – угол между отрезками li-1, li . Данный выбор узлов производится только среди узлов с углом ai < p. Согласно этой формуле при близких значениях длин ребер большую сложность будут иметь узлы с малыми углами.

Diagram

Description automatically generated

Рис. 3. Участок фронта с узлом Ni

Следующим этапом является определение локального шага триангуляции в выбранном узле (блок 5), который рассчитывается по формуле:

,

где KL=1,35 – коэффициент максимального удлинения стороны элемента, позволяющий получать элементы с незначительным искажением; I – интенсивность сгущения сетки (0,2 ≤ I < 1); R – радиус области сгущения; Di – расстояние от узла до точки сгущения. Последний компонент рассматривается только при нахождении узла Ni в области сгущения Di <R. Использование локального шага триангуляции позволяет формировать сетки со сгущениями на сложных участках границы и в местах с высокими градиентами интенсивностей скоростей деформации, которые обладают достаточно высоким качеством.

Дальнейшая часть алгоритма представляет собой разбор возможных ситуаций, учитывающий значение угла в узле, длины прилегающих к нему ребер и положение близлежащих узлов фронта. Если количество узлов фронта всего 3 (рис. 4, а), то добавляется элемент сетки, а сам фронт удаляется (блоки 6-7).

Diagram, engineering drawing

Description automatically generated

Рис. 4. Формирование элементов в зависимости от геометрии фронта

Следующей возможной ситуацией является срезание фронта (блоки 8-9), которое производится при условии



или же

 и ,

где Kc=1,2 – коэффициент удлинения ребра при срезании (рис. 4,б). Необходимым условием, допускающим срезание фронта, также является отсутствие в треугольнике, образованном узлами N i-1, Ni , Ni+1, других узлов. В случае выполнения всех условий в триангуляцию добавляется элемент с узлами Ni-1, Ni , Ni+1, а узел Ni исключается из фронта.

Если срезание фронта невозможно, выполняется поиск ближайших узлов фронта, лежащих в секторе, образованном ребрами  и  (блок 10). При существовании некоторого узла Nj, расстояние до которого , среди четверки возможных элементов, включающих отрезок  (показаны штриховыми линиями на рис. 4,в), выполняется поиск элемента с наилучшим качеством по критерию отношения самой короткой стороны треугольника к самой длинной. Если такой треугольник является корректным (не имеет пересечений с линиями фронта), производится добавление выбранного элемента в триангуляцию, а текущий фронт разделяется на два, которые в дальнейшем рассматриваются самостоятельно (блок 12).

В случае, когда не произошло ни срезания, ни разделения фронта, внутрь области добавляется новый узел K (рис. 4, г), формируется элемент триангуляции с узлами, Ni, Ni+1, K, и узел K добавляется во фронт между Ni и Ni+1 (блок 13). При позиционировании узла используются следующие правила: угол  при  или  при ; длина ребра

.

Значение длины ребра уменьшается, когда расстояние от нового узла до противолежащего участка фронта оказывается меньше . В данной ситуации целесообразно использовать в качестве длины ребра половину этого расстояния.

Если ни один из четырех рассматриваемых на рис. 4 вариантов построения (блоки 6-13) не привел к образованию новых элементов, то локальный шаг триангуляции уменьшается на 20 % (блок 15), производится возврат к блоку 10, и анализ ситуации повторяется.

После заполнения элементами области триангуляции, выражающегося в сокращении всех фронтов, производится оптимизация полученной сетки двумя способами:

в случае вхождения узла сетки в 8 и более элементов производится локальное перестроение сетки путем разделении узла на 2, приводящее к уменьшению числа вхождений и увеличению минимальных значений углов в элементах (блок 17);

итерационное сглаживание сетки путем переноса каждого внутреннего узла сетки в точку центра масс системы треугольников, содержащих данный узел (блок 18).

# Разработка алгоритма работы программы

Алгоритм работы приложения является простым. Первым делом пользователь запускает приложение и вводит всю необходимую информацию любым удобным для него способом. У него на выбор есть варианты вручную добавить количество точек. На основе которых будет создана таблица для заполнения данных или же загрузить информацию из файлов.

После внесения всех данных пользователь нажимает на кнопку генерации, после чего приложение в новом окне сформирует треугольные сетки разбиения в пространстве построенном по всем введенным точкам ранее.

После построения треугольных сеток разбиение приложение позволяет пользователю сохранить как введенные данне (точки для построения пространства, шаг сетки) так и созранение полученного изображения в файл.

Теперь пользователь может либо продолжить работу с приложением внеся новые данные, либо сохранить полученный ответ в файл и выйти из приложения.

Касаемо внутренней части приложения - разработка алгоритма работы программы включает в себя следующие шаги:

1. Инициализация приложения

Приложение создает экземпляр класса TriangularMesh, который наследуется от класса JFrame и содержит в себе список точек pointList и значение шага step для разбиения области на треугольники. После инициализации данных полей происходит вызов метода init() для настройки параметров приложения и отображения окна.

1. Отображение области и границ

При отображении области сначала определяются минимальные и максимальные значения координат по осям X и Y из списка точек. Затем определяется количество строк и столбцов сетки для разбиения области на треугольники. Для отображения области в окне приложения вычисляется смещение xOffset и yOffset по осям X и Y.

После этого происходит отрисовка сетки с помощью циклов for. В циклах происходит проверка, что точка находится внутри области, и если это так, то отрисовываются соответствующие линии.

Также отрисовываются границы области путем соединения всех точек списка pointList.

1. Проверка, лежит ли точка внутри области

Для определения, лежит ли точка внутри области, используется метод isInsideArea, который проверяет, находится ли точка слева или справа от каждой из линий, заданных двумя точками из списка pointList. Если точка лежит слева от всех линий, то она находится внутри области и метод возвращает true, иначе - false.

В целом, алгоритм работы данной программы сводится к отрисовке сетки на основе списка точек, определенных пользователем, и проверке того, что точки находятся внутри области, заданной этими точками.

# Описание основных компонентов программы и последовательности разработки

Приложение написано на Java, поэтому основными компонентами являются поля и методы классов.

Программа реализует графический интерфейс пользователя для работы с множеством точек на плоскости, заданным пользователем или загруженным из файла. Возможно генерировать триангуляцию множества точек на основе заданного шага и сохранять её в файл, а также сохранять изображение триангуляции в формате png.

Основные компоненты программы:

* Класс PointForm наследуется от класса JFrame и отвечает за создание и отображение окна программы с элементами управления;
* JTable - таблица, в которой отображаются введенные пользователем точки;
* JTextField - текстовое поле, используется для ввода количества точек и шага;
* TriangularMesh - класс для генерации триангуляции множества точек на основе заданного шага.

Последовательность разработки:

1. Реализовать класс PointForm, создать окно программы, разместить на нём элементы управления.
2. Добавить обработчики событий на кнопки и текстовые поля:

* updateTableButton - обновление таблицы с точками;
* loadButton - загрузка множества точек из файла;
* saveButton - сохранение множества точек в файл;
* generateButton - генерация триангуляции на основе заданного шага;
* saveImageButton - сохранение изображения триангуляции в формате png.

1. Создать класс TriangularMesh, реализующий алгоритм триангуляции множества точек.
2. Реализовать методы чтения и записи множества точек в файл.
3. Провести тестирование и отладку программы.

# Исходный код программы

package org.hescha.triangle;  
  
import javax.imageio.ImageIO;  
import javax.swing.JButton;  
import javax.swing.JFileChooser;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JOptionPane;  
import javax.swing.JPanel;  
import javax.swing.JScrollPane;  
import javax.swing.JTable;  
import javax.swing.JTextField;  
import javax.swing.filechooser.FileNameExtensionFilter;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import java.io.FileWriter;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class PointForm extends JFrame {  
 private JTable pointTable;  
 private DefaultTableModel tableModel;  
 private JTextField stepField;  
 private JTextField pointField;  
 private TriangularMesh triangularMesh;  
  
 public PointForm() {  
 super("Point Form");  
 setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 setLayout(new BorderLayout());  
  
 JPanel northPanel = new JPanel(new FlowLayout());  
 JLabel pointLabel = new JLabel("Enter number of points:");  
 northPanel.add(pointLabel);  
 pointField = new JTextField(10);  
 northPanel.add(pointField);  
 JButton updateTableButton = new JButton("Update table");  
 northPanel.add(updateTableButton);  
 JLabel stepLabel = new JLabel("Enter step number");  
 northPanel.add(stepLabel);  
 stepField = new JTextField("10");  
 northPanel.add(stepField);  
 add(northPanel, BorderLayout.*NORTH*);  
  
 tableModel = new DefaultTableModel(new String[]{"X", "Y"}, 0);  
 pointTable = new JTable(tableModel);  
 JScrollPane scrollPane = new JScrollPane(pointTable);  
 add(scrollPane, BorderLayout.*CENTER*);  
  
  
 JPanel southPanel = new JPanel(new FlowLayout());  
 JButton loadButton = new JButton("Load data");  
 southPanel.add(loadButton);  
 JButton saveButton = new JButton("Save data");  
 southPanel.add(saveButton);  
 JButton generateButton = new JButton("Generate triangles");  
 southPanel.add(generateButton);  
 JButton saveImageButton = new JButton("Save image");  
 southPanel.add(saveImageButton);  
 add(southPanel, BorderLayout.*SOUTH*);  
  
 updateTableButton.addActionListener(e -> updateTableButtonAction(pointField));  
 loadButton.addActionListener(e -> loadButtonAction());  
 saveButton.addActionListener(e -> saveButtonAction());  
 generateButton.addActionListener(e -> generateButtonAction(stepField));  
 saveImageButton.addActionListener(e -> saveImageButtonAction());  
  
 setSize(550, 300);  
 setVisible(true);  
 }  
  
 private void saveImageButtonAction() {  
 try {  
 // create a file chooser to select the file to save  
 JFileChooser fileChooser = new JFileChooser();  
 fileChooser.setDialogTitle("Specify a file to save image");  
 fileChooser.setFileFilter(new FileNameExtensionFilter("\*.png", "png"));  
 int userSelection = fileChooser.showSaveDialog(null);  
  
 if (userSelection == JFileChooser.*APPROVE\_OPTION*) {  
 // get the selected file to save  
 File fileToSave = fileChooser.getSelectedFile();  
 JPanel panel = triangularMesh.getCanvasPanel();  
 // Get the preferred size of the panel  
 Dimension size = panel.getBounds().getSize();  
 // Create a BufferedImage and get its Graphics object  
 BufferedImage image = new BufferedImage(size.width, size.height, BufferedImage.*TYPE\_INT\_RGB*);  
 Graphics2D g2d = image.createGraphics();  
 // Draw the panel onto the BufferedImage  
 panel.printAll(g2d);  
 g2d.dispose();  
 // Save the BufferedImage to a file  
 ImageIO.*write*(image, "png", new File(fileToSave.getAbsolutePath() + ".png"));  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 JOptionPane.*showMessageDialog*(null, "Something goes wrong while saving file.");  
 }  
 }  
  
 private void generateButtonAction(JTextField stepField) {  
 List<Point> points = readPointsFromTable();  
 if (points == null) return;  
 try {  
 int step = Integer.*parseInt*(stepField.getText());  
 if (step < 1) throw new RuntimeException();  
 triangularMesh = new TriangularMesh(step, points);  
 } catch (RuntimeException ex) {  
 JOptionPane.*showMessageDialog*(PointForm.this, "Please enter valid positive number of step");  
 throw ex;  
 }  
 }  
  
 private void saveButtonAction() {  
 saveFile();  
 }  
  
 private void saveFile() {  
 try {  
 // create a file chooser to select the file to save  
 JFileChooser fileChooser = new JFileChooser();  
 fileChooser.setDialogTitle("Specify a file to save");  
 int userSelection = fileChooser.showSaveDialog(null);  
  
 if (userSelection == JFileChooser.*APPROVE\_OPTION*) {  
 // get the selected file to save  
 File fileToSave = fileChooser.getSelectedFile();  
  
 List<Point> points = readPointsFromTable();  
 int step = Integer.*parseInt*(stepField.getText());  
 int pointNumber = points.size();  
 System.*out*.println("Save as file: " + fileToSave.getAbsolutePath());  
 // create a file writer to write to the file  
 String separator = "\r\n";  
 FileWriter fw = new FileWriter(fileToSave);  
 fw.append(step + "").append(separator);  
 fw.append(pointNumber + "").append(separator);  
 // write the matrix data  
 for (Point point : points) {  
 fw.append(point.x + " " + point.y).append(separator);  
 }  
 fw.append(separator);  
 fw.flush();  
 fw.close();  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 JOptionPane.*showMessageDialog*(null, "Something goes wrong while saving file.");  
 }  
 }  
  
  
 //return step number  
 // reading  
 // step  
 // point number  
 // pointX pointY  
 public static int openFile(List<Point> points) {  
 Scanner scanner;  
 int step = -1;  
 // Creates an instance of JFileChooser to select a file.  
 JFileChooser fileChooser = new JFileChooser();  
 // The showDialog method returns the option selected by the user.  
 int ret = fileChooser.showDialog(null, "Open file");  
 // If the user approves the option, the selected file is processed.  
 if (ret == JFileChooser.*APPROVE\_OPTION*) {  
 // The selected file is stored in the file variable.  
 try {  
 File file = fileChooser.getSelectedFile();  
 scanner = new Scanner(file);  
 // A scanner is created to read the data from the file.  
 step = scanner.nextInt();  
 int pointsNumber = scanner.nextInt();  
  
 for (int i = 0; i < pointsNumber; i++) {  
 Point point = new Point(scanner.nextInt(), scanner.nextInt());  
 points.add(point);  
 }  
 } catch (Exception e) {  
 JOptionPane.*showMessageDialog*(null, "Please check that file exists and contains correct values.");  
 return -1;  
 }  
 }  
 return step;  
 }  
  
  
 private void loadButtonAction() {  
 List<Point> points = new ArrayList<>();  
 int step = *openFile*(points);  
 stepField.setText(step + "");  
 if (step > 0) {  
 tableModel.setRowCount(points.size());  
 pointField.setText(points.size() + "");  
 for (int i = 0; i < points.size(); i++) {  
 Point point = points.get(i);  
 tableModel.setValueAt(point.x, i, 0);  
 tableModel.setValueAt(point.y, i, 1);  
 }  
 }  
 }  
  
 private void updateTableButtonAction(JTextField pointField) {  
 try {  
 int numPoints = Integer.*parseInt*(pointField.getText());  
 if (numPoints > 2) {  
 tableModel.setRowCount(numPoints);  
 } else {  
 JOptionPane.*showMessageDialog*(PointForm.this, "Please enter a number more or equals 3");  
 }  
 } catch (NumberFormatException ex) {  
 JOptionPane.*showMessageDialog*(PointForm.this, "Please enter a valid integer for the number of points");  
 }  
 }  
  
 public List<Point> readPointsFromTable() {  
 List<Point> points = new ArrayList<>();  
 int x, y;  
 for (int i = 0; i < tableModel.getRowCount(); i++) {  
 String xCoord = tableModel.getValueAt(i, 0).toString();  
 String yCoord = tableModel.getValueAt(i, 1).toString();  
  
 try {  
 x = Integer.*parseInt*(xCoord);  
 y = Integer.*parseInt*(yCoord);  
 } catch (NumberFormatException ex) {  
 JOptionPane.*showMessageDialog*(PointForm.this, "Please enter valid integers for the coordinates on point " + i);  
 throw ex;  
 }  
 points.add(new Point(x, y));  
 }  
 return points;  
 }  
  
 public static void main(String[] args) {  
 new PointForm();  
 }  
}

package org.hescha.triangle;  
  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import java.awt.Color;  
import java.awt.Graphics;  
import java.awt.Point;  
import java.util.List;  
  
public class TriangularMesh extends JFrame {  
 private final int width = 800;  
 private final int height = 600;  
  
 private final List<Point> pointList;  
 private final int step;  
 private JPanel canvasPanel;  
  
 void init() {  
 setTitle("Triangular Mesh");  
 setSize(width, height);  
  
 JPanel panel = new JPanel();  
 panel.setLayout(null);  
  
 canvasPanel = new JPanel() {  
 @Override  
 public void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 drawTriangularMesh(g);  
 }  
 };  
 canvasPanel.setBounds(0, 0, width \* 3, height \* 3);  
 panel.add(canvasPanel);  
  
 add(panel);  
 setVisible(true);  
 }  
  
 public JPanel getCanvasPanel() {  
 return canvasPanel;  
 }  
  
 public TriangularMesh(int step, List<Point> pointList) {  
 this.pointList = pointList;  
 this.step = step;  
 init();  
 }  
  
 private void drawTriangularMesh(Graphics g) {  
 drawMesh(g);  
 drawBorders(g);  
 }  
  
 private void drawBorders(Graphics g) {  
 // Set line color and stroke  
 g.setColor(new Color(255, 128, 128)); // Light red  
  
 // Draw line through all points  
 pointList.add(pointList.get(0));  
 for (int i = 0; i < pointList.size() - 1; i++) {  
 Point p1 = pointList.get(i);  
 Point p2 = pointList.get(i + 1);  
 g.drawLine(p1.x, p1.y, p2.x, p2.y);  
 }  
 pointList.remove(pointList.size() - 1);  
 }  
  
 private void drawMesh(Graphics g) {  
 int minX = (int) pointList.stream().mapToDouble(Point::getX).min().orElse(Integer.*MAX\_VALUE*);  
 int maxX = (int) pointList.stream().mapToDouble(Point::getX).max().orElse(Integer.*MIN\_VALUE*);  
 int minY = (int) pointList.stream().mapToDouble(Point::getY).min().orElse(Integer.*MAX\_VALUE*);  
 int maxY = (int) pointList.stream().mapToDouble(Point::getY).max().orElse(Integer.*MIN\_VALUE*);  
  
 // Определение размеров сетки  
 int width = maxX - minX;  
 int height = maxY - minY;  
 int rows = height / step;  
 int cols = width / step;  
  
 // смещение для отображения области в окне  
 int xOffset = minX;  
 int yOffset = minY;  
  
 // Отрисовка сетки  
 g.setColor(Color.*BLACK*);  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 int x = j \* step + xOffset;  
 int y = i \* step + yOffset;  
 if (isInsideArea(x, y) && isInsideArea(x + step, y) && isInsideArea(x, y + step)) {  
 g.drawLine(x, y, x + step, y);  
 g.drawLine(x, y, x, y + step);  
 g.drawLine(x, y + step, x + step, y);  
 }  
 }  
 }  
 }  
  
 private boolean isInsideArea(int x, int y) {  
 // Вычислить уравнения линий между точками области  
 double[] eqs = new double[pointList.size()];  
 for (int i = 0; i < pointList.size(); i++) {  
 Point p1 = pointList.get(i);  
 Point p2 = pointList.get((i + 1) % pointList.size());  
 eqs[i] = (p2.getY() - p1.getY()) \* (x - p1.getX()) - (y - p1.getY()) \* (p2.getX() - p1.getX());  
 }  
 // Проверить, лежит ли точка слева или справа от каждой из линий  
 for (int i = 1; i < eqs.length; i++) {  
 if (eqs[i] > 0) {  
 return false;  
 }  
 }  
 // Вернуть true, если точка лежит слева от всех линий  
 return true;  
 }  
}

# Проверка корректности работы программы в различных режимах

Запустим приложение и посмотрим работает ли оно вообще.

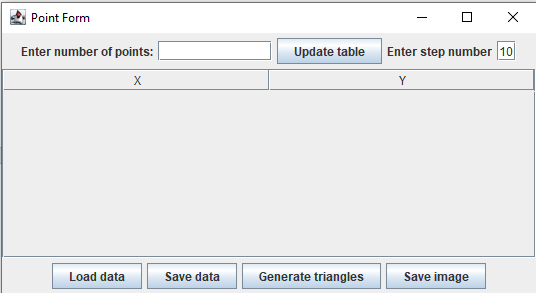


Рисунок 1 – Запуск приложения

Как видно из рисунка, приложение запускается и показывает поля для выбора количества точек, на основе которых будет создана областьдля заполнения. Попробуем создать таблицу с выбранным количеством строк и столбцов.

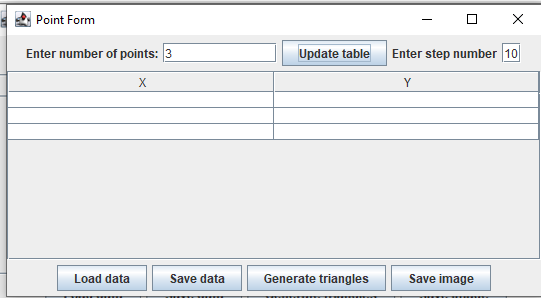


Рисунок 2 – создание таблицы заданной сложности

Таблица создается без проблем. Чтоб не заполнять все данные вручную, загрузим заранее заготовленный пример из файла.

Graphical user interface, text, application, email

Description automatically generated

Рисунок 3 – Чтение из файла

При нажатии на кнопку загрузить мы видим окно, в котором необходимо выбрать файл с данными для формирования треугольных сеток.

Graphical user interface, table

Description automatically generated

Рисунок 4 – Результат чтения из файла

После выбора файла в приложении отображается все его данные и мы можем сгенерировать изоюращение с заполненными треугольными сетками.

Background pattern

Description automatically generated with low confidence

Рисунок 5 – Результат выполнения

Красным цветом отображается область, которая была построена на основе всех введенных точек. Внутри этой области были сформированытреугольные сетки разбиения.

Данные из файла читаются в следующем порядке:

* Шаг сетки
* Количество точек
* Координата Х точки
* Координата Y точки

Graphical user interface, application, Word

Description automatically generated

Рисунок 6 – Спецификация загружаемого файла

После построение треугольной сетки, пользователь может сохранить изображение

Graphical user interface, application

Description automatically generated

Рисунок 7 – Сохранение результата

После нажатия на кнопку сохранения нам предложат выбрать название файла и метсо его сохранения.

A picture containing chart

Description automatically generated

Рисунок 8 Вид сохраненного файла

После всех этих манипуляций файл сохраняется в том же виде, в каком он был отображен на экране.

На случай незаполнения всех полей предусмотрен выброс ошибок.

Graphical user interface, text, application, email

Description automatically generated

Рисунок 9 - Вызов ошибки при попытке формирования сетки без заполненных полей

Так же в приложении имеется проверка на правильность ввода.

Graphical user interface, text, application, Word

Description automatically generated

Рисунок 10 – Вызов ошибки при некорректном вводе

# Заключение

В результате выполнения данной курсовой работы было разработано приложение для формирования треугольной сетки разбиения для заданной двумерной области на языке программирования Java с использованием Swing.

Также была реализована функциональность выбора шага сетки, вывода результата в файл, визуального отображения результата и сохранения изображения.

Приложение имеет удобный пользовательский интерфейс, позволяющий задавать параметры области и сетки, проверять введенные значения, сохранять и загружать данные в файлы. Визуализация результата позволяет пользователю быстро оценить качество сетки и внести необходимые изменения.

Разработанное приложение может быть использовано в различных областях науки и техники, где требуется построение треугольных сеток разбиения для заданной области.

В результате работы были получены следующие результаты:

* Разработано приложение для формирования треугольной сетки разбиения для заданной двумерной области
* Реализована функциональность выбора шага сетки, вывода результата в файл, визуального отображения результата и сохранения изображения.
* Приложение имеет удобный пользовательский интерфейс, позволяющий задавать параметры области и сетки, проверять введенные значения, сохранять и загружать данные в файлы.
* Разработанное приложение может быть использовано в различных областях науки и техники, где требуется построение треугольных сеток разбиения для заданной области.

Таким образом, выполнение данной курсовой работы позволило получить практические навыки разработки приложений на языке Java с использованием Swing, а также реализации алгоритма построения треугольной сетки разбиения для заданной области.

# Литература

1. Омельченко А.В. Конечно-элементный метод. Теория и практика применения в задачах гидравлики и гидропневмоприводов: учебное пособие / А.В. Омельченко. – М.: МГТУ им. Н.Э. Баумана, 2016. – 214 с.
2. Яромеева Н.Ф. Метод конечных элементов: учебное пособие / Н.Ф. Яромеева. – СПб.: БХВ-Петербург, 2016. – 256 с.
3. Bathe, K. Finite Element Procedures. Prentice-Hall, 1996. 1034p.
4. Hughes, T.J.R. The Finite Element Method: Linear Static and Dynamic Finite Element Analysis. Prentice-Hall, 1987. 853p.
5. Zienkiewicz, O.C. The Finite Element Method. McGraw-Hill, 1977. 768p.
6. Нечай И.Г. Конечноэлементный анализ в механике: учебное пособие / И.Г. Нечай, В.С. Кривошеин. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2006. – 415 с.
7. Беляев А.К. Конечно-элементный метод. Проектирование элементов машин: учебник для вузов / А.К. Беляев. – 2-е изд., перераб. и доп. – М.: Машиностроение, 2007. – 368 с.
8. Корн Г., Корн Т. Математический справочник для научных работников и инженеров. / Г. Корн, Т. Корн. – М.: Наука, 1973. – 832 с.
9. Львовский М. С. Лекции по методам оптимизации: учебное пособие. – М.: МЦНМО, 2014. – 257 с.
10. Сергеев В.С. Численные методы: учебник / В.С. Сергеев. – СПб.: Изд-во Политехнического университета, 2009. – 592 с.
11. Java [Электронный ресурс]. – Режим доступа: https://www.java.com/ru/. – Дата доступа: 03.08.2023.
12. Java Swing Tutorial [Электронный ресурс]. – Режим доступа: https://www.javatpoint.com/java-swing. – Дата доступа: 03.08.2023.
13. Show save file dialog using JFileChooser [Электронный ресурс]. – Режим доступа: https://www.codejava.net/java-se/swing/show-save-file-dialog-using-jfilechooser. – Дата доступа: 03.08.2023.