



МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»



**НГТУ
НЭТИ** | **Факультет прикладной
математики и информатики**

Кафедра прикладной математики
Курсовой проект
по дисциплине «Метод конечных элементов»

**ПОСТРОЕНИЕ КОНЕЧНОЭЛЕМЕНТНОЙ СЕТКИ ИЗ
ПРЯМОУГОЛЬНИКОВ С НЕРАВНОМЕРНЫМ ШАГОМ В
ГОРИЗОНТАЛЬНО-СЛОИСТОЙ СРЕДЕ С
НЕОДНОРОДНОСТЯМИ**

Группа ПМ-92 АРТЮХОВ РОМАН

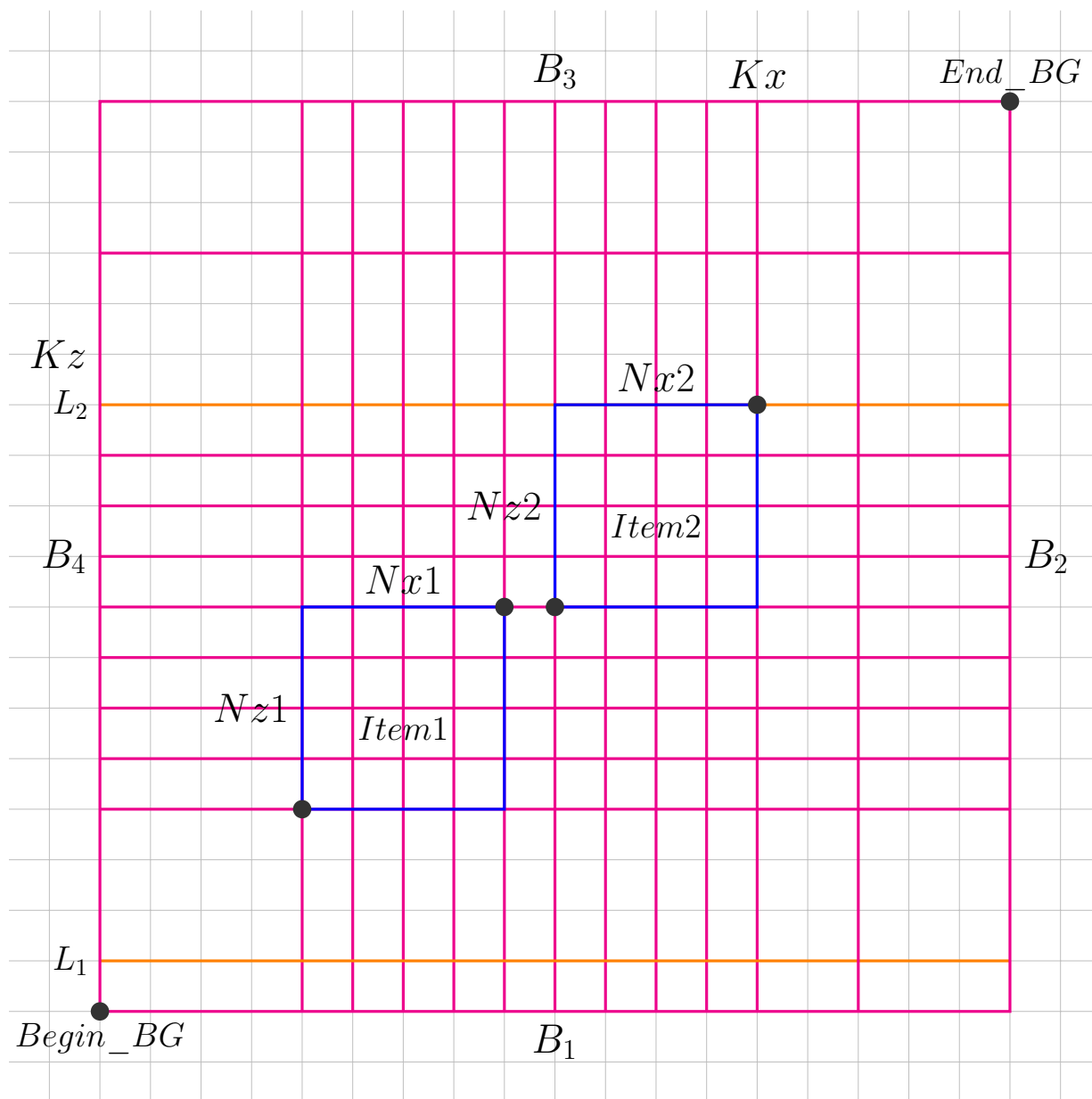
Преподаватели ЗАДОРОВ А. Г.
ПАТРУШЕВ И. И.

Новосибирск, 2022

СОДЕРЖАНИЕ

1. Условные обозначения. Входные данные	3
2. Выходные данные	4
3. Алгоритм построения сетки	6
4. Учет горизонтальных слоев	10
5. Руководство по программе	12
5.1. Используемые при разработке средства	12
5.2. Структура программы	12
5.3. Реализованные структуры	13
5.4. Описание пользовательского интерфейса	15
5.5. Взаимодействие с Plot из пакета ScottPlot	18
6. Тестирование	21
6.1. Количество объектов	21
6.2. Характеристика сетки «Строгость»	24

1. Условные обозначения. Входные данные



Данные для сетки вводятся пользователем через оконный интерфейс.

Begin_BG и **End_BG** - точки начала и конца большого поля.

Item1 и **Item2** - объекты.

Nx и **Nz** - количество разбиений по объекту.

Kx и **Kz** - коэффициент разрядки от объекта.

L1, **L2** - горизонтальные слои.

SideBound(B1, B2, B3, B4) - номера краевых на границах.

2. Выходные данные

На выходе мы получаем файлы **nodes.txt**, **elems.txt**, **edges.txt**, **bounds.txt**, а также с помощью пакета *ScottPlot.WPF* можно отобразить сетку в окне и сохранить в виде картинки формата (.png).

Структура файла **nodes.txt**:

Первое число (**int CountX**) - количество узлов по Оси X.

Второе число (**int CountZ**) - количество узлов по Оси Z.

Далее (**CountX * CountZ**) строк - узлы (double X, double Z)

Структура файла **edges.txt**:

Первое число (**int CountEdge**) - количество ребер.

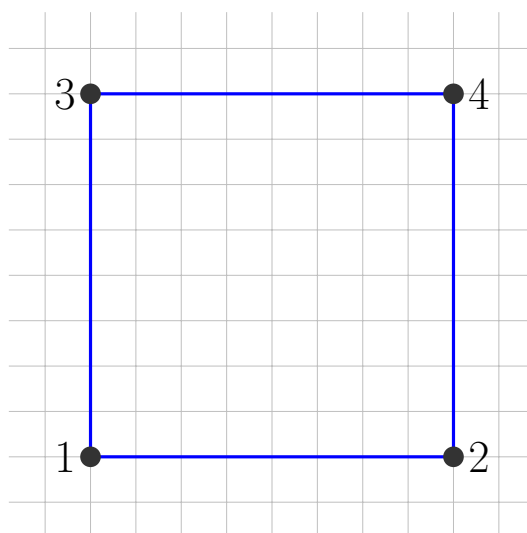
Далее **CountEdge** строк - координаты двух узлов ребра **double** (X_1, Z_1, X_2, Z_2).

Структура файла **elems.txt**:

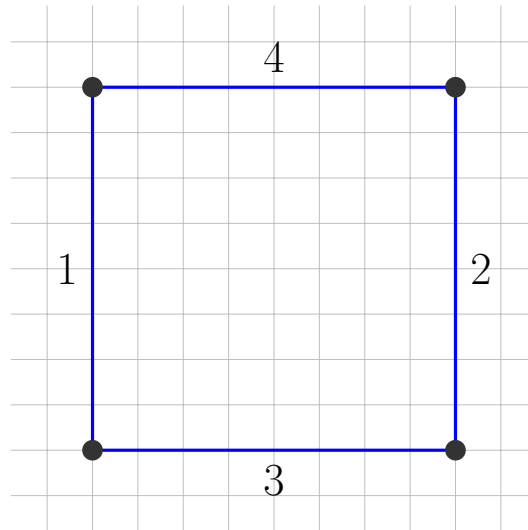
Первое число (**int CountElem**) - количество конечных элементов.

Далее **CountElem** строк - нумерация конечного элемента по узлам и ребрам.

Нумерация конечного элемента по узлам выглядит следующим образом:



Нумерация конечного элемента по ребрам выглядит следующим образом:



Структура файла **bounds.txt**:

Первое число (**int CountBound**) - количество ребер на которых задано краевое условие.

Далее **CountBound** строк - характеристика краевого условия.

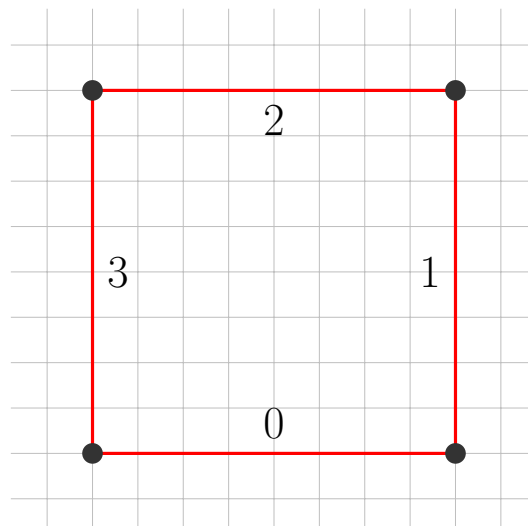
Характеристика краевого условия выглядит следующим образом:

Первое число (**int NumberBound**) - номер краевого условия.

Второе число (**int NumberSide**) - номер стороны сетки.

Третье число (**int NumberEdge**) - номер ребра.

Нумерация сторон сетки:



3. Алгоритм построения сетки

1. Равномерно разбиваем объекты:
2. Считаем равномерные шаги ($\mathbf{h}_x, \mathbf{h}_z$) по объекту:

$$h_x = \frac{Item.End_x - Item.Begin_x}{Nx}, \quad h_z = \frac{Item.End_z - Item.Begin_z}{Nz}$$

3. Составляем список шагов ($\mathbf{H}_x, \mathbf{H}_z$) следующим образом:
 - (a) Добавляем значение шага \mathbf{h}_x в список \mathbf{H}_x (\mathbf{N}_x) раз (\mathbf{h}_z в список \mathbf{H}_z (\mathbf{N}_z) раз),
 - (b) Увеличиваем шаг $\mathbf{h}_x * \mathbf{K}_x$ ($\mathbf{h}_z * \mathbf{K}_z$) и добавляем в начало списка пока не пересечем $\mathbf{Begin_BG}_x$ ($\mathbf{Begin_BG}_z$),
 - (c) Увеличиваем шаг $\mathbf{h}_x * \mathbf{K}_x$ ($\mathbf{h}_z * \mathbf{K}_z$) и добавляем в конец списка пока не пересечем $\mathbf{End_BG}_x$ ($\mathbf{End_BG}_z$),
 - (d) В начало списка \mathbf{H}_x (\mathbf{H}_z) добавлем 0.
4. Генерируем узлы по составленным спискам шагов ($\mathbf{H}_x, \mathbf{H}_z$):
 - (a) Берем начальные значения $\mathbf{X} = \mathbf{Begin_BG}_x$, $\mathbf{Z} = \mathbf{Begin_BG}_z$,
 - (b) Берем шаг из списка \mathbf{H}_z , прибавляем к \mathbf{Z} и получаем $\mathbf{Z_new}$,
 - (c) Берем шаг из списка \mathbf{H}_x , прибавляем к \mathbf{X} и получаем $\mathbf{X_new}$,
 - (d) Создаем узел $\mathbf{Node}(\mathbf{X_new}, \mathbf{Z_new})$,
 - (e) Если шаги в списке \mathbf{H}_x остались, то возвращаемся к пункту (c),
 - (f) Обновляем $\mathbf{X} = \mathbf{Begin_BG}_x$ и возвращаемся на пункт (b), пока шаги в списке \mathbf{H}_z не закончатся.

5. Генерируем нумерацию конечных элементов:

- (a) Генерируем номера узлов:
 - i. Берем начальные значения $\mathbf{i} = 0, \mathbf{j} = 0$,
 - ii. Генерируем номера конечного элемента
$$\mathbf{N}_1 = \mathbf{i} * \mathbf{CountX} + \mathbf{j},$$
$$\mathbf{N}_2 = \mathbf{i} * \mathbf{CountX} + \mathbf{j} + 1,$$

$$\mathbf{N}_3 = (\mathbf{i} + 1) * \mathbf{CountX} + \mathbf{j},$$

$$\mathbf{N}_4 = (\mathbf{i} + 1) * \mathbf{CountX} + \mathbf{j} + 1,$$

iii. Создаем конечный элемент **Elem**(**N**₁, **N**₂, **N**₃, **N**₄),

iv. Прибавляем к **j** единицу и повторяем шаг (b), пока

$$\mathbf{j} < \mathbf{CountX} - 1,$$

v. Обновляем **j** = 0. Прибавляем к **i** единицу и возвращаемся на шаг (b), пока **i** < **CountZ** - 1.

(b) Генерируем ребра и номера ребер для конечного элемента:

i. Берем начальные значения **i** = 0, **j** = 0,

ii. Генерируем пять индексов:

$$\mathbf{left} = \mathbf{i} * ((\mathbf{CountX} - 1) + \mathbf{CountX}) + (\mathbf{CountX} - 1) + \mathbf{j},$$

$$\mathbf{right} = \mathbf{i} * ((\mathbf{CountX} - 1) + \mathbf{CountX}) + (\mathbf{CountX} - 1) + \mathbf{j} + 1,$$

$$\mathbf{bottom} = \mathbf{i} * ((\mathbf{CountX} - 1) + \mathbf{CountX}) + \mathbf{j},$$

$$\mathbf{top} = (\mathbf{i} + 1) * ((\mathbf{CountX} - 1) + \mathbf{CountX}) + \mathbf{j},$$

$$\mathbf{n_elem} = \mathbf{i} * (\mathbf{CountX} - 1) + \mathbf{j},$$

iii. Создаем четыре ребра:

$$\mathbf{Edge}[\mathbf{left}](\mathbf{Elem}[\mathbf{n_elem}].\mathbf{Node}[\mathbf{1}], \mathbf{Elem}[\mathbf{n_elem}].\mathbf{Node}[\mathbf{3}]),$$

$$\mathbf{Edge}[\mathbf{right}](\mathbf{Elem}[\mathbf{n_elem}].\mathbf{Node}[\mathbf{2}], \mathbf{Elem}[\mathbf{n_elem}].\mathbf{Node}[\mathbf{4}]),$$

$$\mathbf{Edge}[\mathbf{bottom}](\mathbf{Elem}[\mathbf{n_elem}].\mathbf{Node}[\mathbf{1}], \mathbf{Elem}[\mathbf{n_elem}].\mathbf{Node}[\mathbf{2}]),$$

$$\mathbf{Edge}[\mathbf{top}](\mathbf{Elem}[\mathbf{n_elem}].\mathbf{Node}[\mathbf{3}], \mathbf{Elem}[\mathbf{n_elem}].\mathbf{Node}[\mathbf{4}]),$$

iv. Добавляем к элементу номера ребер:

$$\mathbf{Elem}[\mathbf{n_elem}](\mathbf{left}, \mathbf{right}, \mathbf{bottom}, \mathbf{top}),$$

v. Прибавляем к **j** единицу и возвращаемся на шаг (ii), пока

$$\mathbf{j} < \mathbf{CountX} - 1,$$

vi. Обновляем **j** = 0. Прибавляем к **i** единицу и возвращаемся на шаг (ii), пока **i** < **CountZ** - 1.

6. Генерируем краевые условия:

(a) Берем из входных данных номера краевых условий **SideBound**,

(b) Генерируем краевые нижней границы:

- i. Берем начальное значение $\mathbf{i} = \mathbf{0}$,
 - ii. Подсчитываем индекс ребра:

$$\mathbf{id} = \mathbf{i}$$
 - iii. Генерируем краевое:

$$\mathbf{Bound}(\mathbf{SideBound}[\mathbf{0}], \mathbf{0}, \mathbf{id}),$$
 - iv. Прибавляем к \mathbf{i} единицу и возвращаемся на шаг (ii), пока

$$\mathbf{i} < \mathbf{CountX} - \mathbf{1}.$$
- (с) Генерируем краевые правой границы:
- i. Берем начальное значение $\mathbf{i} = \mathbf{1}$,
 - ii. Подсчитываем индекс ребра:

$$\mathbf{id} = \mathbf{i} * \mathbf{CountX} + \mathbf{i} * (\mathbf{CountX} - \mathbf{1}) - \mathbf{1}$$
 - iii. Генерируем краевое:

$$\mathbf{Bound}(\mathbf{SideBound}[\mathbf{1}], \mathbf{1}, \mathbf{id}),$$
 - iv. Прибавляем к \mathbf{i} единицу и возвращаемся на шаг (ii), пока

$$\mathbf{i} < \mathbf{CountZ}.$$
- (d) Генерируем краевые верхней границы:
- i. Берем начальное значение $\mathbf{i} = \mathbf{0}$,
 - ii. Подсчитываем индекс ребра:

$$\mathbf{id} = \mathbf{CountX} * (\mathbf{CountZ} - \mathbf{1}) +$$

$$+ (\mathbf{CountX} - \mathbf{1}) * (\mathbf{CountY} - \mathbf{1}) + \mathbf{i}$$
 - iii. Генерируем краевое:

$$\mathbf{Bound}(\mathbf{SideBound}[\mathbf{2}], \mathbf{2}, \mathbf{id}),$$
 - iv. Прибавляем к \mathbf{i} единицу и возвращаемся на шаг (ii), пока

$$\mathbf{i} < \mathbf{CountX} - \mathbf{1}.$$
- (е) Генерируем краевые левой границы:
- i. Берем начальное значение $\mathbf{i} = \mathbf{0}$,
 - ii. Подсчитываем индекс ребра:

$$\mathbf{id} = (\mathbf{i} + \mathbf{1}) * (\mathbf{CountX} - \mathbf{1}) + \mathbf{i} * \mathbf{CountX}$$
 - iii. Генерируем краевое:

Bound(SideBound[3], 3, id),

iv. Прибавляем к i единицу и возвращаемся на шаг (ii), пока
 $i < \text{CountZ} - 1$.

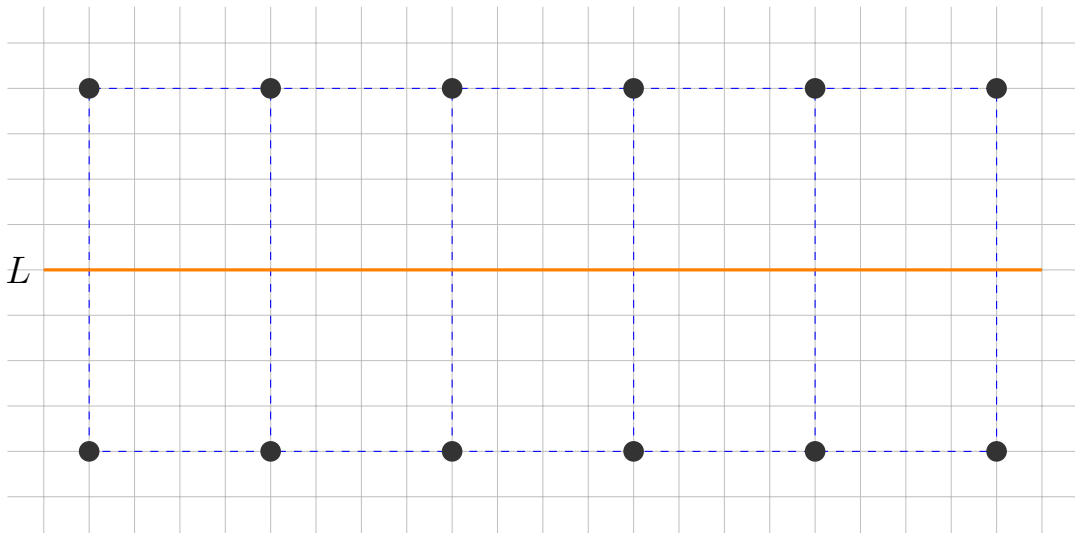
(f) Сортируем краевые в порядке убывания номера краевого.

4. Учет горизонтальных слоев

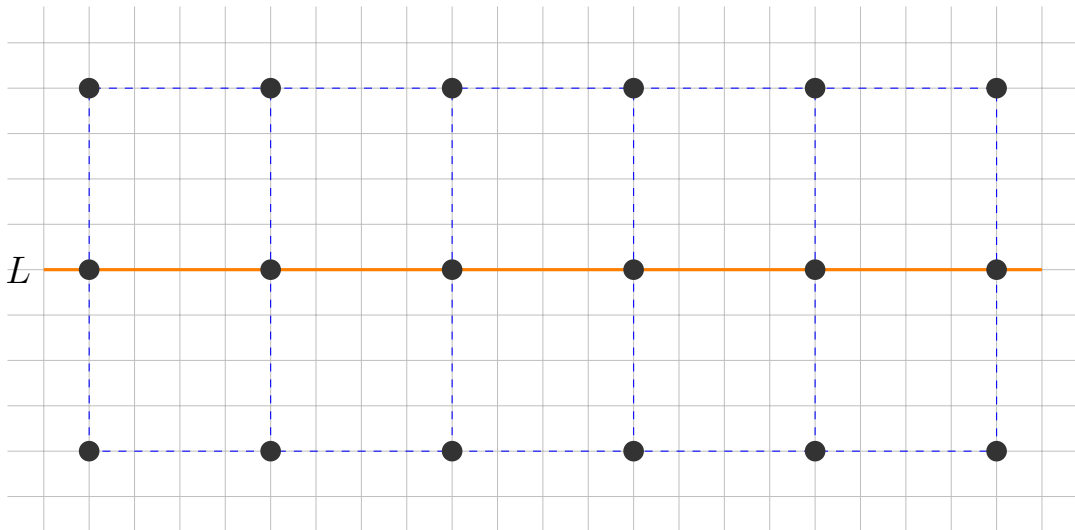
В сетке можно задавать горизонтальные слои (\mathbf{L}).

Важно: слой должен проходить через узлы.

Неправильный учет слоя:



Правильный учет слоя:



Алгоритм учета горизонтальных слоев:

Учитывать слои будем после этапа построения списков шагов ($\mathbf{H}_x, \mathbf{H}_y$).

1. Берем начальные значения $\mathbf{Y} = \mathbf{Begin_BG}_y$,
2. Берем шаг из списка \mathbf{H}_y , прибавляем к \mathbf{Y} и получаем $\mathbf{Y_new}$,

3. Если $Y_{\text{new}} = L$, то на слое будут лежать узлы,
Если $Y_{\text{new}} > L$, добавим шаг, чтобы учесть слой,
4. Если добавленный шаг меньше минимального (min_step), то значение шага прибавляем к предыдущему значению шага, а текущий шаг удаляем,
5. Если шаги в списке H_y и непройденные слои остались, то возвращаемся к пункту (2).

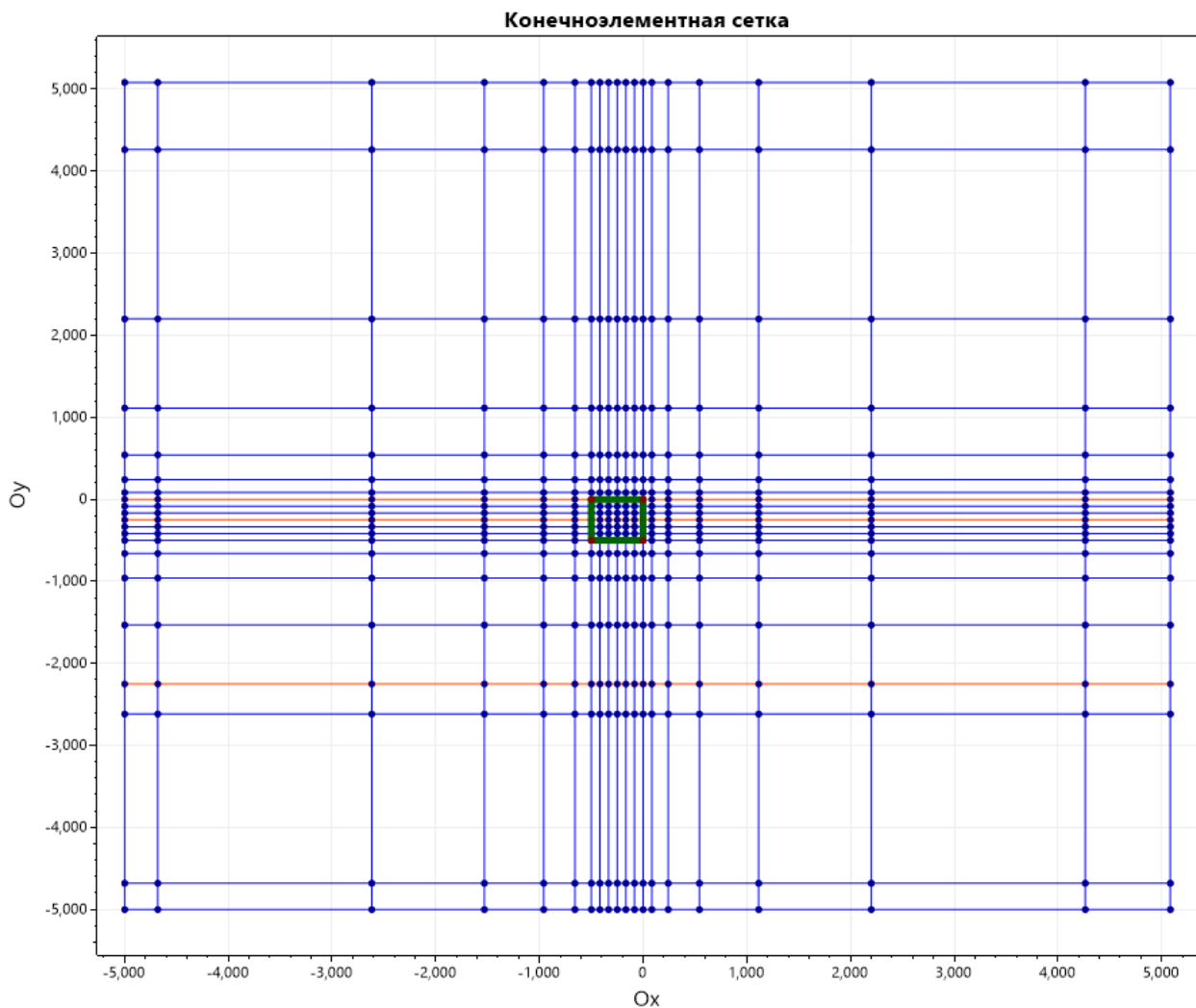


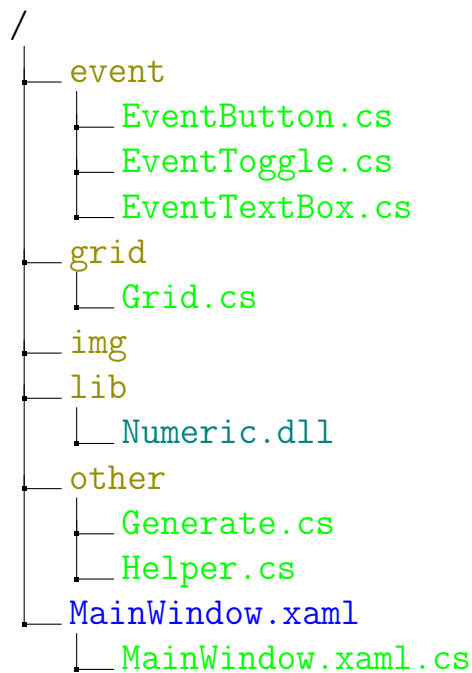
Рисунок 4.1 – Сетка в горизонтальной слоистой среде $L=(0,-250,-500)$

5. Руководство по программе

5.1. Используемые при разработке средства

1. Visual Studio 2022 – основная среда разработки,
2. WPF – построение графического интерфейса,
3. C# – основной язык логики приложения,
4. ScottPlot - библиотека построения графиков,
5. WPFToggleSwitch - пакет для Toggle Button.

5.2. Структура программы



5.3. Реализованные структуры

Структура Item

```
1 public struct Item
2 {
3     ///  
4     public Vector<double> Begin { get; set; }
5     public Vector<double> End   { get; set; }
6     public int           Nx    { get; set; }
7     public int           Ny    { get; set; }
8     public string        Name  { get; set; }
9
10    ///  
11    public Item(Vector<double> begin, Vector<double> end,
12                int nx, int ny, string name = "None") { }
13 }
```

Структура Node<Type>

```
1 public struct Node<T> where T : System.Numerics.INumber<T>
2 {
3     ///  
4     public T X { get; set; } ///  
5     public T Y { get; set; } ///  
6
7     ///  
8     public Node(T _X, T _Y) { }
9
10    ///  
11    public void Deconstruct(out T x, out T y) { }
12
13    ///  
14    public override string ToString() { }
15 }
```

Структура Edge<Type>

```
1 public struct Edge<T> where T : System.Numerics.INumber<T>
2 {
3     //: Fields and properties
4     public Node<T> NodeBegin { get; set; } /// The begin node of the edge
5     public Node<T> NodeEnd { get; set; } /// The end node of the edge
6
7     //: Constructor
8     public Edge(Node<T> _begin, Node<T> _end) { }
9
10    //: Deconstructor
11    public void Deconstruct(out Node<T> begin, out Node<T> end) { }
12
13    //: String view structure
14    public override string ToString() { }";
15 }
```

Структура Elem

```
1 public struct Elem
2 {
3     //: Fields and properties
4     public int[] Node; /// Numbers node final element
5     public int[] Edge; /// Numbers edge final element
6
7     //: Constructor
8     public Elem(params int[] node) { }
9
10    //: Deconstructor
11    public void Deconstruct(out int[] nodes, out int[] edges) { }
12
13    //: String view structure
14    public override string ToString() { }
15 }
```

Структура Bound

```
1 public struct Bound
2 {
3     ///  
4     public int Edge      { get; set; }    ///  
5     public int NumBound { get; set; }    ///  
6     public int NumSide  { get; set; }    ///  
7  
8     ///  
9     public Bound(int num, int side, int edge) { }  
10  
11    ///  
12    public void Deconstruct(out int num, out int side, out int edge) { }  
13  
14    ///  
15    public override string ToString() { }  
16 }
```

5.4. Описание пользовательского интерфейса

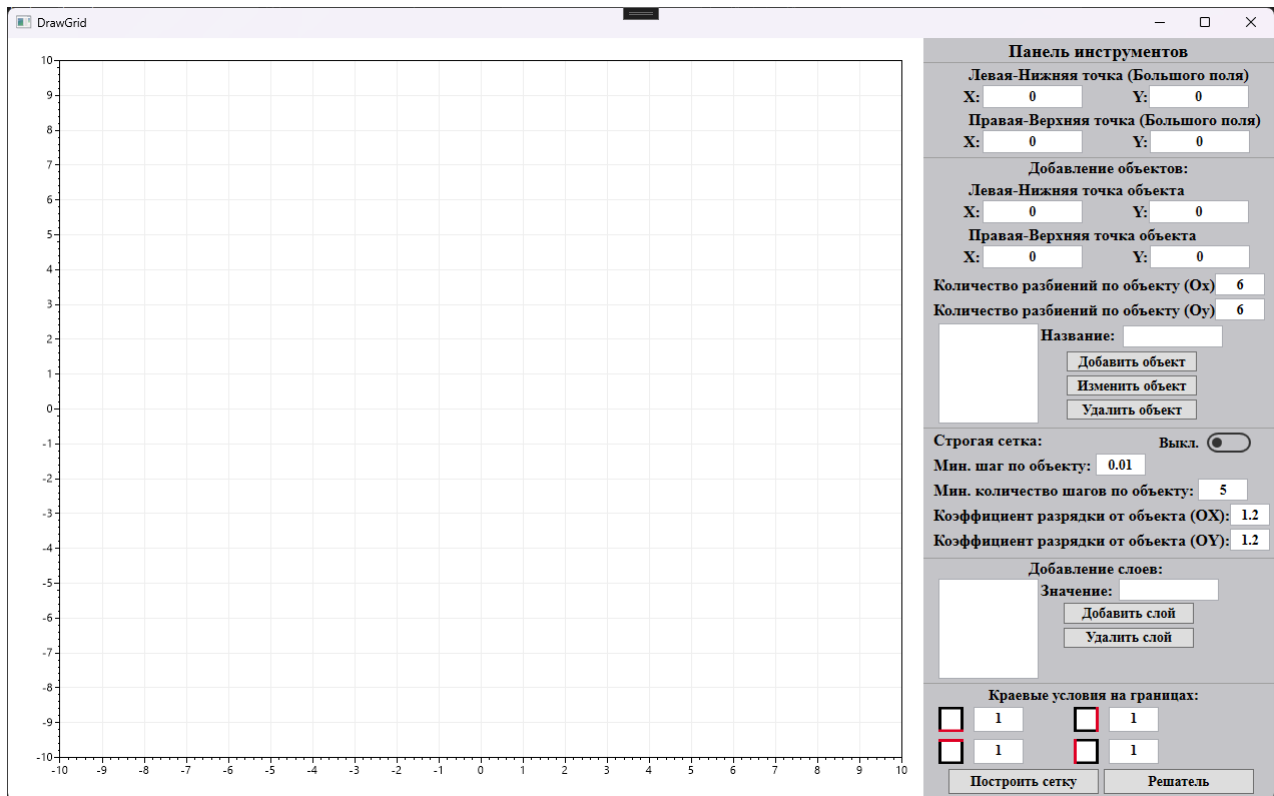






Рисунок 5.1 – Вид пользовательского интерфейса

Таблица 5.1 – Описание бокового меню

Пункт меню	Назначение
Пункты меню для задания координат большого поля	
Левая-Нижняя точка (Большого поля)	Задает точку Begin_BG
Правая-Верхняя точка (Большого поля)	Задает точку End_BG
Пункты меню для задания объекта	
Левая-Нижняя точка объекта	Задает точку Begin
Правая-Верхняя точка объекта	Задает точку End
Количество разбиений по объекту (Ox)	Задает параметр объекта Nx
Количество разбиений по объекту (Oy)	Задает параметр объекта Ny
Название	Задает параметр объекта Name
Кнопка «Добавить объект»	Добавляет новый объект в список объектов
Кнопка «Изменить объект»	Изменяет выбранный в ListBox объект
Кнопка «Удалить объект»	Удаляет выбранный в ListBox объект
Пункты меню для задания общих параметров	
Строгая сетка	ВКЛ./ВЫКЛ. строгость сетки
Минимальный шаг по объекту	Задает параметр min_step
Минимальное количество шагов по объекту	Задает параметр count_step

Продолжение таблицы 5.1

Пункт меню	Назначение
Коэффициент разрядки от объекта (OX)	Задаёт параметр K_x
Коэффициент разрядки от объекта (OY)	Задаёт параметр K_y
Пункты меню для задания слоя	
Значение	Задаёт значение слоя
Кнопка «Добавить слой»	Добавляет значение слоя на сетку
Кнопка «Удалить слой»	Удаляет значение слоя из сетки
Пункты меню для задания краевых условий	
	Задаёт краевое условие на <u>нижней</u> стороне сетки
	Задаёт краевое условие на <u>правой</u> стороне сетки
	Задаёт краевое условие на <u>верхней</u> стороне сетки
	Задаёт краевое условие на <u>левой</u> стороне сетки
Кнопки построения сетки и запуска решателя	
Кнопка «Построить сетку»	Строит (перестраивает) сетку
Кнопка «Решатель»	Открывает окошко решателя

5.5. Взаимодействие с Plot из пакета ScottPlot

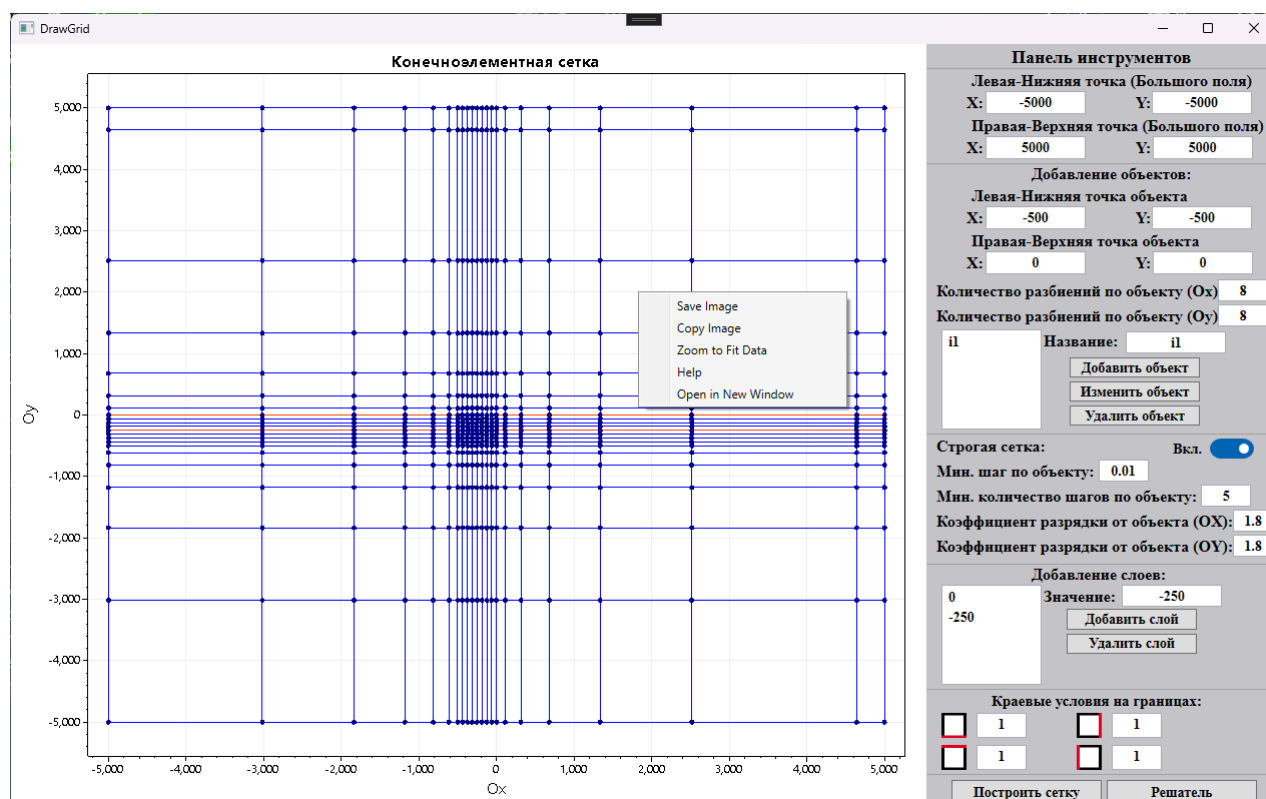


Таблица 5.2 – Описание инструментария пакета

Пункт меню	Назначение
Save Image	Открывает диалоговое окно для сохранения Plot
Copy Image	Сохраняет картинку в буфер
Zoom to Fit Data	Увеличивает масштаб, чтобы соответствовать данным

Продолжение таблицы 5.2

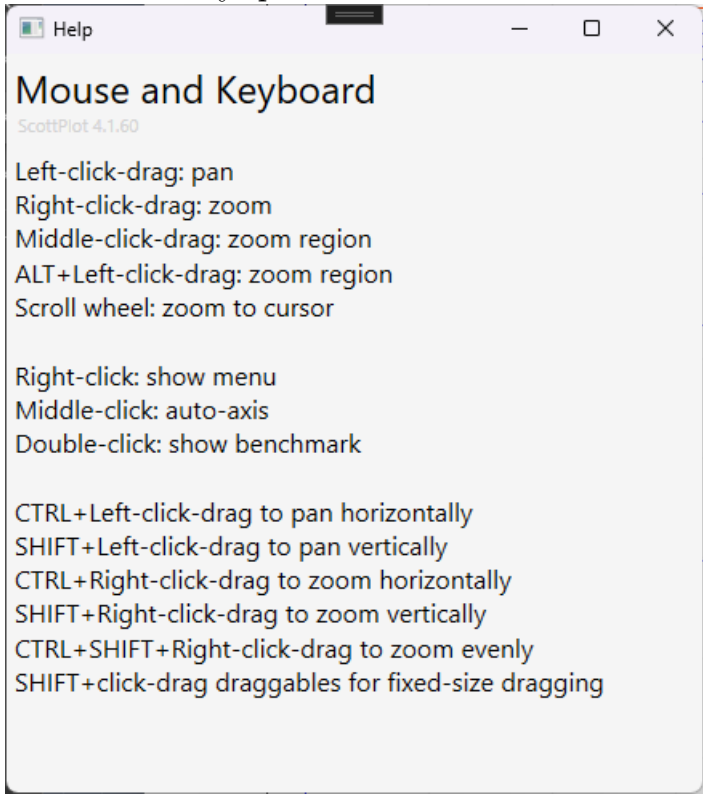
Пункт меню	Назначение
Help	<p>Открывает окошко с инструкцией для управления Plot</p>  <p>Left-click-drag: pan Right-click-drag: zoom Middle-click-drag: zoom region ALT+Left-click-drag: zoom region Scroll wheel: zoom to cursor</p> <p>Right-click: show menu Middle-click: auto-axis Double-click: show benchmark</p> <p>CTRL+Left-click-drag to pan horizontally SHIFT+Left-click-drag to pan vertically CTRL+Right-click-drag to zoom horizontally SHIFT+Right-click-drag to zoom vertically CTRL+SHIFT+Right-click-drag to zoom evenly SHIFT+click-drag draggables for fixed-size dragging</p>
Open in New Window	Открывает Plot в новом окошке

Таблица 5.3 – Описание окошка с инструкцией

Действие	Назначение
ЛКМ + перетаскивание	Панорамирование (движение по сетке)
ПКМ + перетаскивание	Увеличение масштаба
СКМ + перетаскивание	Выделение области масштабирования
ALT + ЛКМ + перетаскивание	Выделение области масштабирования
Колесо прокрутки	Масштабирование к курсору
Щелчок ПКМ	Открывает окошко помощи

Продолжение таблицы 5.3

Действие	Назначение
Щелчок СКМ	Увеличивает масштаб, чтобы соответствовать данным
Двойной щелчок ЛКМ	Показывает benchmark
CTRL + ЛКМ + перетаскивание	Горизонтальное перемещение
SHIFT + ЛКМ + перетаскивание	Вертикальное перемещение
CTRL + ПКМ + перетаскивание	Горизонтальное увеличение
SHIFT + ПКМ + перетаскивание	Вертикальное увеличение
CTRL + SHIFT + ПКМ + перетаскивание	Равномерное увеличение

6. Тестирование

6.1. Количество объектов

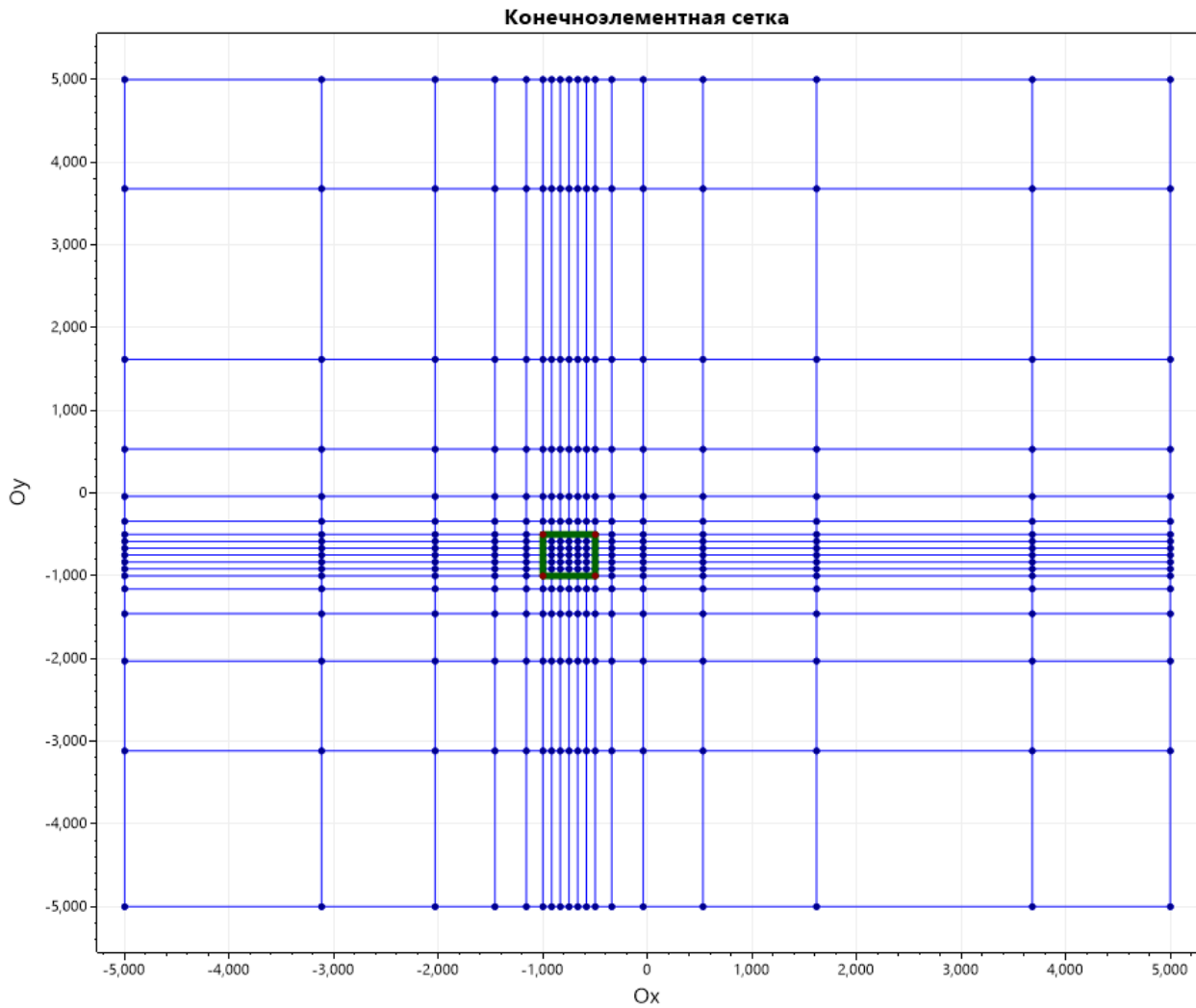


Рисунок 6.1 – Сетка с одним объектом

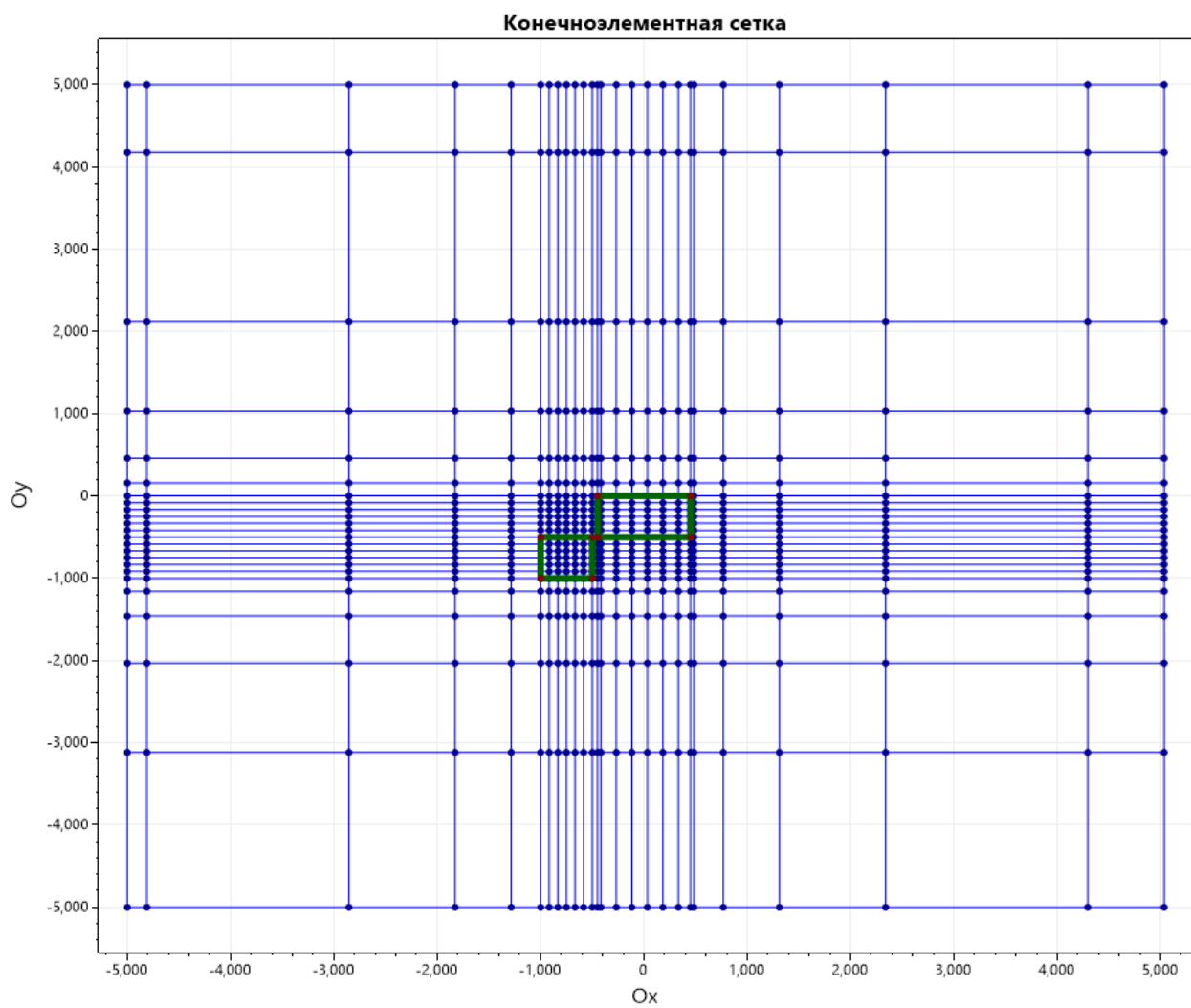


Рисунок 6.2 – Сетка с двумя объектами

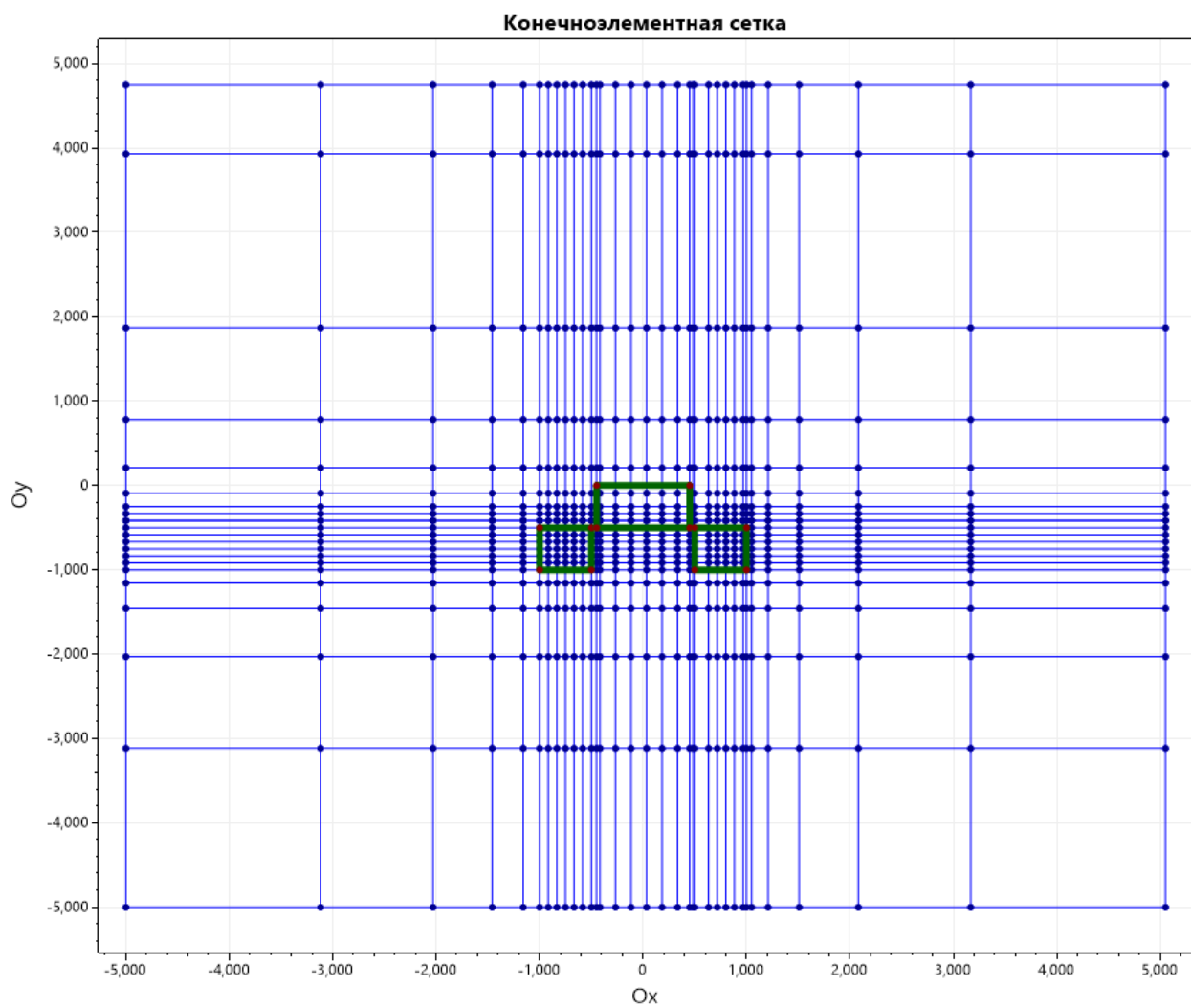


Рисунок 6.3 – Сетка с тремя объектами

6.2. Характеристика сетки «Строгость»

Будем говорить сетка строгая, если ее границы совпадают с точками (Begin_BG, End_BG).

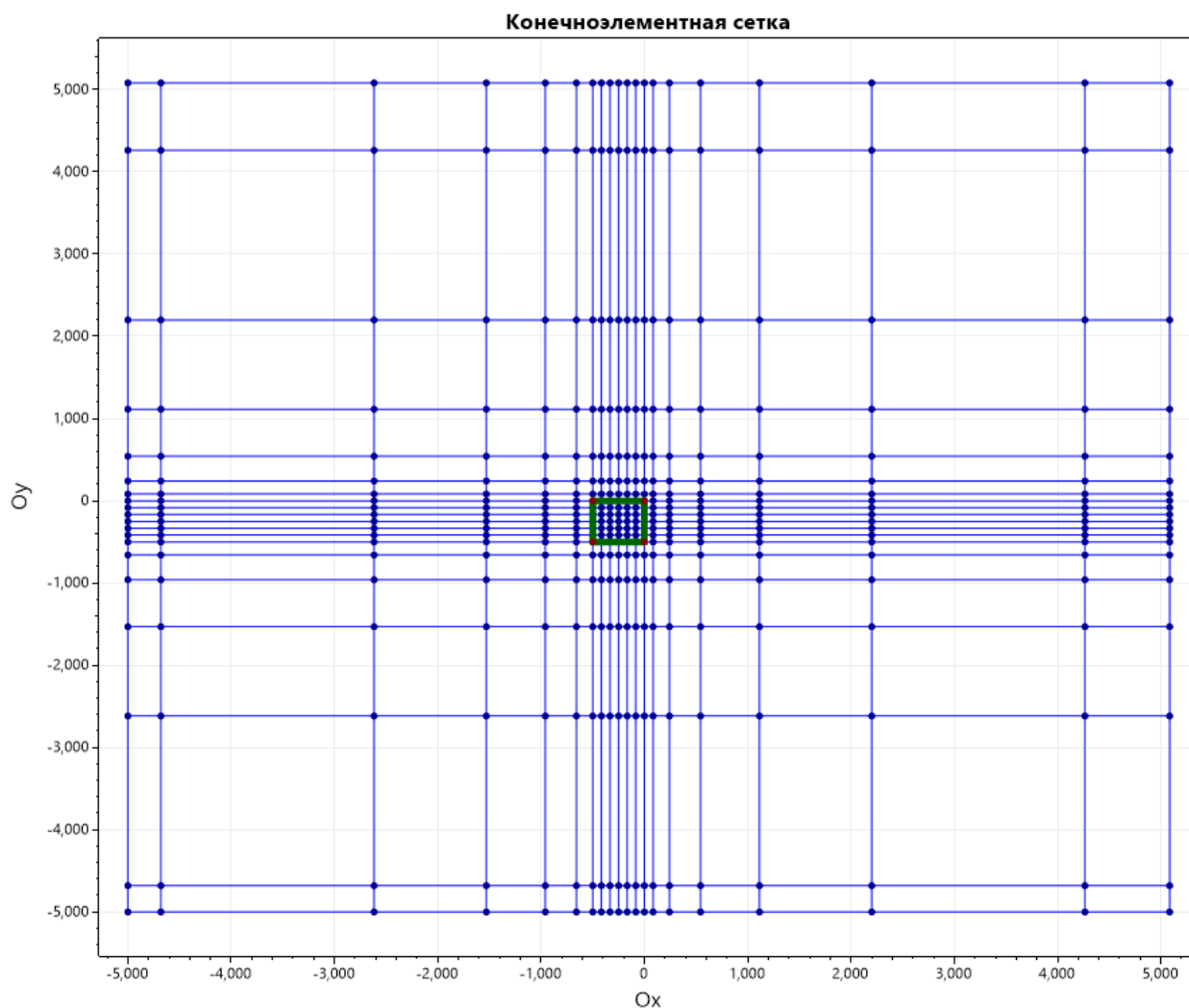


Рисунок 6.4 – Строгая сетка Begin_BG=(-5000,-5000), End_BG=(5000, 5000)

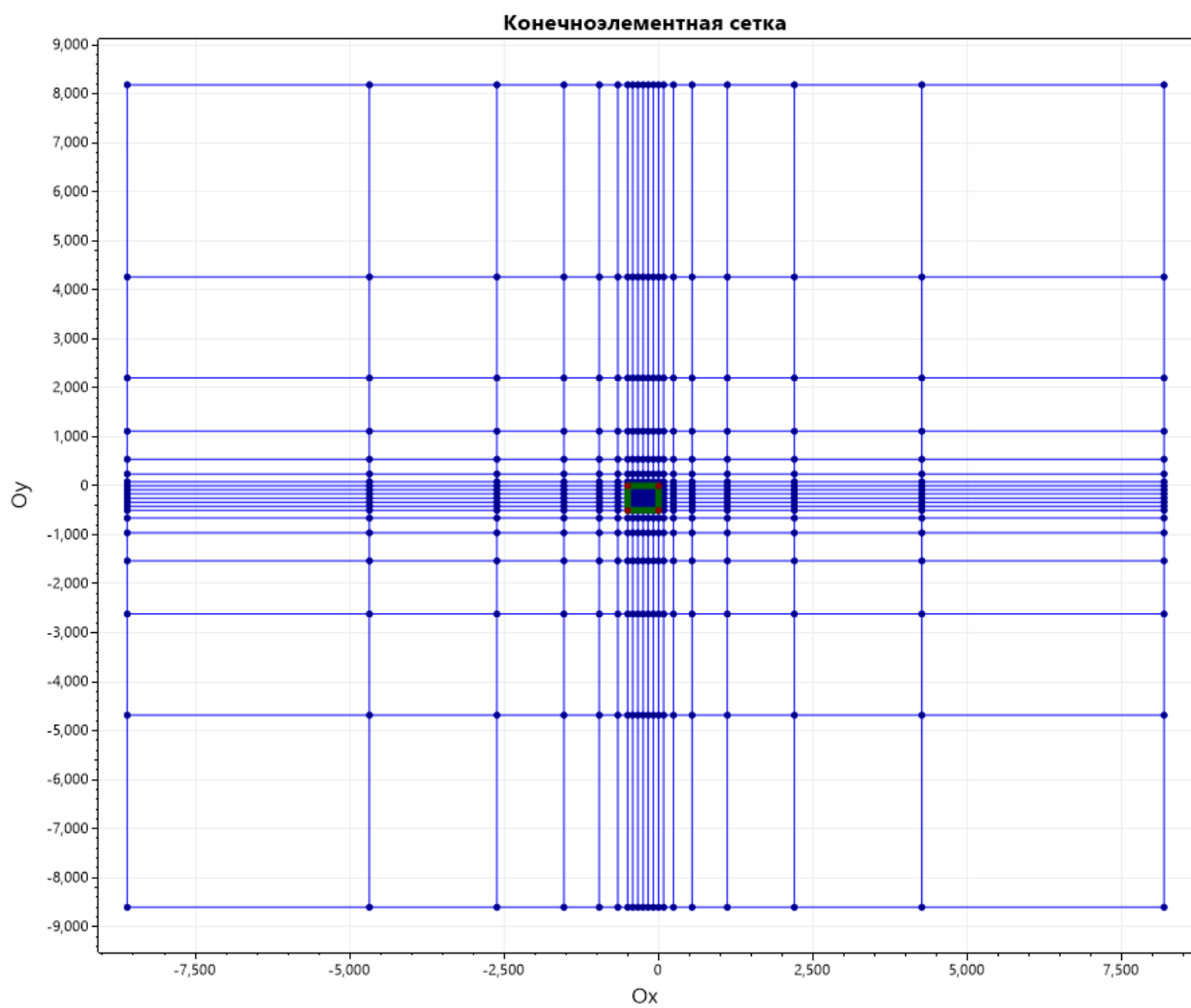


Рисунок 6.5 – Нестрогая сетка $\text{Begin_BG}=(-5000,-5000)$, $\text{End_BG}=(5000, 5000)$