

СОДЕРЖАНИЕ

1. Математическая модель	3
1.1. Система уравнений Максвелла	3
1.2. Вывод ротор-роторного уравнения	4
1.3. Применение технологии выделения поля	5
1.4. Соотношения между компонентами электрического и магнитного полей	6
 2. Построение конечноэлементной сетки из прямоугольников с неравномерным шагом в горизонтально-слоистой среде с неоднородностями	 7
2.1. Условные обозначения. Входные данные	7
2.2. Выходные данные	8
2.3. Алгоритм построения сетки	11
2.4. Учет горизонтальных слоев	14
2.5. Руководство по программе	16
2.5.1. Используемые при разработке средства	16
2.5.2. Структура программы	17
2.5.3. Реализованные структуры	18
2.5.4. Описание пользовательского интерфейса	21
2.5.5. Взаимодействие с Plot из пакета ScottPlot	24
2.6. Примеры сеток	27
2.6.1. Количество объектов	27
2.6.2. Строгие и нестрогие сетки	30
2.6.3. Параллельные объекты	32

3. Процедура решения СЛАУ с использованием библиотеки	
Intel OneMKL	34
3.1. Введение в Intel oneAPI Math Kernel Library	34
3.2. Используемые при разработке средства	34
3.3. Разреженный формат хранения матрицы CSR3	34

1. Математическая модель

1.1. Система уравнений Максвелла

Уравнения Максвелла — система уравнений в дифференциальной или интегральной форме, описывающих электромагнитное поле и его связь с электрическими зарядами и токами в сплошных средах.

Уравнения Максвелла представляют собой в векторной записи систему из четырёх фундаментальных математических выражений.

Первое системное уравнение Максвелла выражает закон Фарадея, физический смысл которого в том, что под влиянием изменяющегося магнитного поля образуется поле электрической природы, способное индуцировать электрический ток в цепи:

$$\operatorname{rot} \vec{E} = -\frac{\partial \vec{B}}{\partial t}, \quad (1.1)$$

\vec{E} - напряжённость электрического поля (в СИ — В/м);

\vec{B} - магнитная индукция (в СИ — Тл = Вб/м² = кг · с⁻² · А⁻¹);

Второе уравнение отображает закон Гаусса, которое говорит о том, что поток поля магнитной природы сквозь какой-либо замкнутый контур всегда будет нулевым:

$$\operatorname{div} \vec{B} = 0, \quad (1.2)$$

Третье уравнение отражает теорему о циркуляции магнитного поля. Электрический ток и изменение электрической индукции порождают вихревое магнитное поле:

$$\operatorname{rot} \vec{H} = \vec{J}^{\text{ср}} + \sigma \vec{E} + \frac{\partial(\varepsilon \vec{E})}{\partial t}, \quad (1.3)$$

\vec{H} - напряжённость магнитного поля (в единицах СИ — А/м);

$\vec{J}^{\text{ср}}$ - вектор плотностей сторонних токов (в СИ — А/м²);

σ — удельная электрическая проводимость среды (в СИ — См);

$\varepsilon \mathbf{E}$ — электрическая индукция (в СИ — Кл/м²);

ε — диэлектрическая проницаемость среды (в СИ — Ф/м);

Четвертое уравнение — это закон Гаусса. Суть закона Гаусса в том, что электрические поля создаются зарядами и зависят от них, другими словами, чем больше зарядов присутствует в данной области, тем сильнее будет электрическое поле:

$$\operatorname{div} \varepsilon \vec{E} = \rho, \quad (1.4)$$

ρ — объёмная плотность стороннего электрического заряда (в СИ — Кл/м³)

1.2. Вывод ротор-роторного уравнения

Когда решается нелинейная по μ нестационарная задача, система уравнений (1.1)-(1.3) может быть преобразована к виду:

$$\operatorname{rot} \left(\frac{1}{\mu} \operatorname{rot} \vec{A} \right) + \sigma \frac{\partial \vec{A}}{\partial t} + \varepsilon \frac{\partial^2 \vec{A}}{\partial t^2} = \vec{J}^{\text{ст}}, \quad (1.5)$$

где \vec{A} — вектор-потенциал, через который определяются индукция \vec{B} магнитного поля и напряжённость \vec{E} электрического поля в виде:

$$\vec{B} = \operatorname{rot} \vec{A} = \mu \vec{H}, \quad (1.6)$$

$$\vec{E} = -\frac{\partial \vec{A}}{\partial t} = -i\omega \vec{A}, \quad (1.7)$$

А после подстановки (1.6) и (1.7) в уравнение (1.3) оно превращается в уравнение (1.5).

Когда в нестационарном электромагнитном поле токи смещения являются пренебрежимо малыми, т.е. можно считать, что $\frac{\partial(\varepsilon \vec{A})}{\partial t} = 0$, тогда уравнение (1.5) принимает вид:

$$\operatorname{rot} \left(\frac{1}{\mu} \operatorname{rot} \vec{A} \right) + \sigma \frac{\partial \vec{A}}{\partial t} = \vec{J}^{\text{ст}}. \quad (1.8)$$

1.3. Применение технологии выделения поля

На многих практических задачах, описываемых сложными математическими моделями, приближение получают с использованием большого количества вычислительных затрат. И для такой задачи можно заменить сложную модель на более простую, тем самым сократив вычислительные затраты и получив достаточно точное решение.

Рассмотрим краевую задачу (1.8) в области Ω , применив соотношение (1.7):

$$\frac{1}{\mu} \operatorname{rot} \operatorname{rot} \vec{A} + i\sigma\omega \vec{A} = \vec{J}^{\text{ст}}, \quad (1.9)$$

Будем считать, что область Ω полностью включается в область Ω' (или совпадает с ней), и в области Ω' определена функция $\vec{A}^{\rightarrow 1D}$, являющаяся решением краевой задачи:

$$\frac{1}{\mu} \operatorname{rot} \operatorname{rot} \vec{A}^{\rightarrow 1D} + i\sigma^{1D}\omega \vec{A}^{\rightarrow 1D} = \vec{J}^{\text{ст}}, \quad (1.10)$$

Будем считать, что функция $\vec{A}^{\rightarrow 1D}$ получена с гораздо меньшими вычислительными затратами численно, как решение задачи меньшей размерности, и при этом она достаточно хорошо приближает искомое решение \vec{A} задачи (1.9). Тогда решение \vec{A} исходной задачи (1.9) будем искать в виде суммы решений $\vec{A}^{\rightarrow 1D}$ «основной» задачи (1.10) и задачи на «добавочное» поле $\vec{A}^{\rightarrow a}$:

$$\begin{aligned} \frac{1}{\mu} \operatorname{rot} \operatorname{rot} \left(\vec{A}^{\rightarrow 1D} + \vec{A}^{\rightarrow a} \right) + i\sigma\omega \left(\vec{A}^{\rightarrow 1D} + \vec{A}^{\rightarrow a} \right) = \\ \vec{J}^{\text{ст}} - \vec{J}^{\text{ст}} + \frac{1}{\mu} \operatorname{rot} \operatorname{rot} \vec{A}^{\rightarrow 1D} + i\sigma^{1D}\omega \vec{A}^{\rightarrow 1D}, \end{aligned} \quad (1.11)$$

Упростим выражение (1.11):

$$\frac{1}{\mu} \operatorname{rot} \operatorname{rot} \vec{A}^{\rightarrow a} + i\omega\sigma \vec{A}^{\rightarrow a} = -i\omega(\sigma - \sigma^{1D}) \vec{A}^{\rightarrow 1D}. \quad (1.12)$$

1.4. Соотношения между компонентами электрического и магнитного полей

Компоненты поля в случае Е-поляризации:

$$\tilde{E}_y = \frac{E_y}{E_y^n}, \quad \tilde{H}_x = \frac{H_x}{H_x^n}, \quad \tilde{H}_z = \frac{H_z}{H_x^n},$$

Компоненты поля в случае Н-поляризации:

$$\tilde{H}_y = \frac{H_y}{H_y^n}, \quad \tilde{E}_x = \frac{E_x}{E_x^n},$$

Где $E_x^n, E_y^n, H_x^n, H_y^n$ - нормальные поля, рассчитанные по одномерным формулам на поверхности Земли для левой модели нормального разреза ($z = 0, x = -\infty$).

Кажущиеся сопротивления вычисляются по формуле:

$$\rho_k = \frac{|Z|^2}{\omega \mu}, \quad (1.13)$$

Компонента Z в случае Е-поляризации:

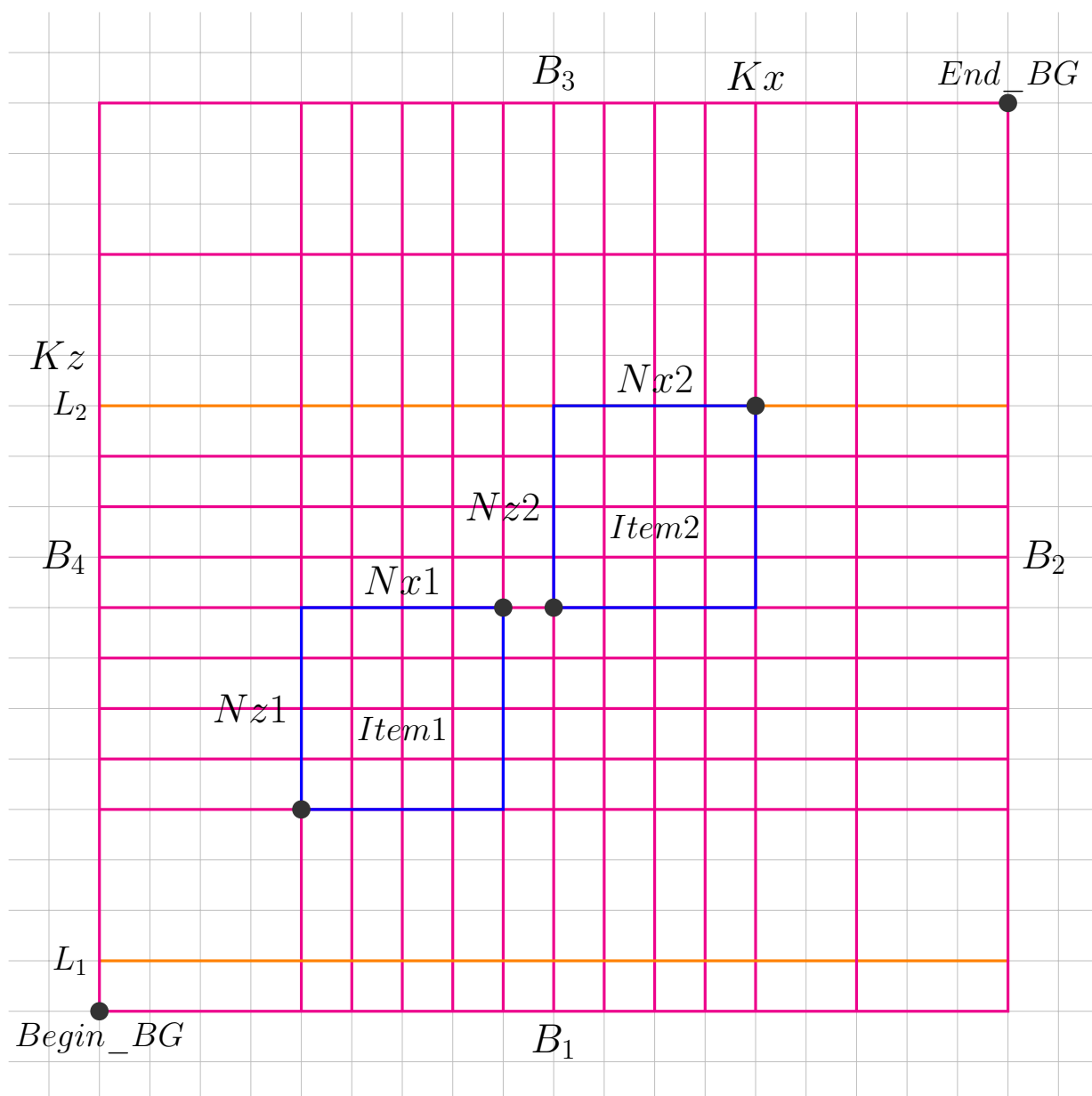
$$Z = -\frac{E_y}{H_x}, \quad (1.14)$$

Компонента Z в случае Н-поляризации:

$$Z = \frac{E_x}{H_y}. \quad (1.15)$$

2. Построение конечноэлементной сетки из прямоугольников с неравномерным шагом в горизонтально-слоистой среде с неоднородностями

2.1. Условные обозначения. Входные данные



Данные для сетки вводятся пользователем через оконный интерфейс.

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

Item1 и **Item2** - объекты (точки объекта).

N_x и N_z - количество разбиений по объекту.

K_x и K_z - коэффициент разрядки от объекта.

L_1, L_2 - горизонтальные слои.

$\text{SideBound}(B_1, B_2, B_3, B_4)$ - номера краевых на границах.

2.2. Выходные данные

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

Структура файла **nodes.txt**:

Первое число (**int CountNodes**) - количество узлов.

Далее **CountNodes** строк - узлы (**double X**, **double Z**)

Структура файла **edges.txt**:

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

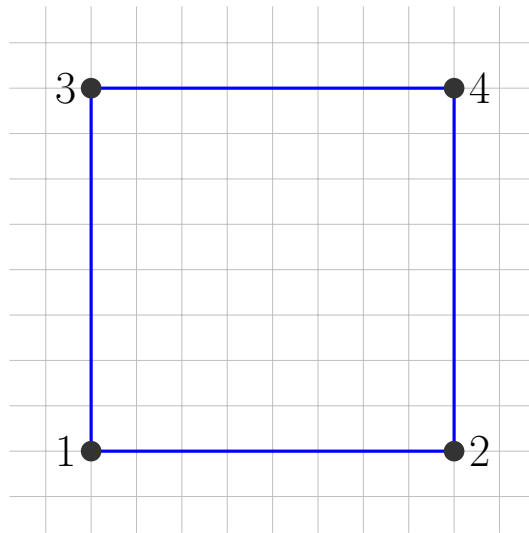
Далее **CountEdge** строк - координаты двух узлов ребра **double** (X_1, Z_1, X_2, Z_2).

Структура файла **elems.txt**:

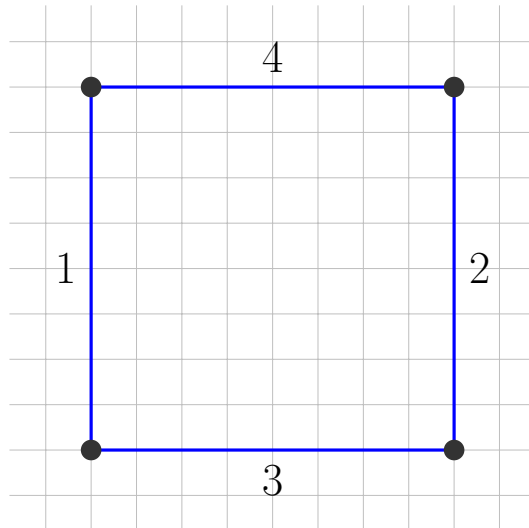
Первое число (**int CountElem**) - количество конечных элементов.

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.

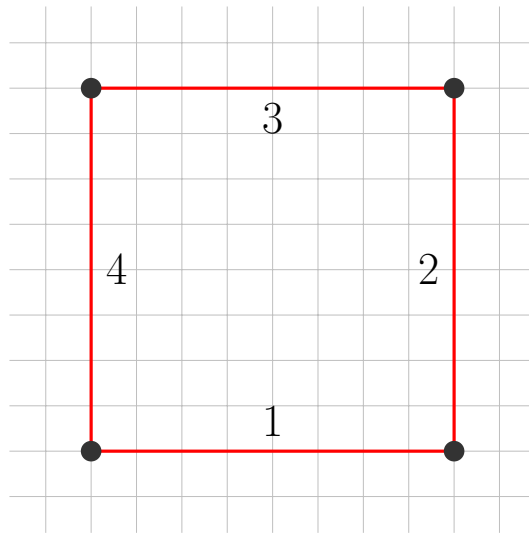
Далее **CountBound** строк - характеристика краевого условия. Характеристика краевого условия выглядит следующим образом:

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

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

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

Нумерация сторон сетки:



Структура файла **items.txt**:

Первое число (**int CountItems**) - количество объектов на сетке.

Далее **CountItems** строк - характеристика объекта. Характеристика объекта выглядит следующим образом:

Первые числа - координаты точек объекта **double** (X_1, Z_1, X_2, Z_2).

Далее два **int** числа - количество разбиений по объекту ось X и ось Z соответственно.

Далее **double** число - значение проводимости объекта.

Последнее **string** слово - имя объекта.

Структура файла **layers.txt**:

Первое число (**int CountLayers**) - количество слоев на сетке.

Далее **CountLayers** строк по два **double** числа - координата **Z** и значение проводимости слоя.

Структура файла **sigmas.txt**:

Первое число (**int CountMaterial**) - количество материалов.

Далее **CountMaterial** строк по два **double** числа - значения одномерной и двумерной проводимости.

Структура файла **interface.txt**:

Первая строка четыре **double** числа - координаты точек бака (X_1, Z_1, X_2, Z_2)

Вторая строка два **int** числа - количество узлов ось X и ось Z соответственно.

Третья строка два **double** числа - коэффициенты разрядки ось X и ось Z соответственно.

Четвертая строка два числа - **double** число минимальный шаг по объекту и **int** число минимальное количество шагов по объекту.

Пятая строка **bool** значение - строгая сетка (true), не строгая сетка соответственно (false).

Шестая строка четыре **int** числа - номера краевых на сторонах сетки нижняя, правая, верхняя, левая соответственно.

2.3. Алгоритм построения сетки

1. Равномерно разбиваем объекты:
2. Считаем равномерные шаги (h_x, h_z) по объектам:

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

3. Составляем список шагов (H_x, H_z) следующим образом:
 - (a) Добавляем значение шага h_x в список H_x (N_x) раз (h_z в список H_z (N_z) раз),
 - (b) Увеличиваем шаг $h_x * K_x$ ($h_z * K_z$) и добавляем в начало списка пока не пересечем **Begin_BG_x** (**Begin_BG_z**),
 - (c) Увеличиваем шаг $h_x * K_x$ ($h_z * K_z$) и добавляем в конец списка пока не пересечем **End_BG_x** (**End_BG_z**),
 - (d) В начало списка H_x (H_z) добавлем 0.
4. Генерируем узлы по составленным спискам шагов (H_x, 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 $\mathbf{Elem}(\mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3, \mathbf{N}_4)$,
 - iv. Прибавляем к \mathbf{j} единицу и повторяем шаг (b), пока $\mathbf{j} < \mathbf{CountX} - 1$,
 - v. Обновляем $\mathbf{j} = 0$. Прибавляем к \mathbf{i} единицу и возвращаемся на шаг (b), пока $\mathbf{i} < \mathbf{CountZ} - 1$.
- (b) Генерируем ребра и номера ребер для конечного элемента:
 - i. Берем начальные значения $\mathbf{i} = 0$, $\mathbf{j} = 0$,
 - 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:

Edge[left](Elem[n_elem].Node[1], Elem[n_elem].Node[3]),
Edge[right](Elem[n_elem].Node[2], Elem[n_elem].Node[4]),
Edge[bottom](Elem[n_elem].Node[1], Elem[n_elem].Node[2])
Edge[top](Elem[n_elem].Node[3], Elem[n_elem].Node[4]),

iv. Добавляем к элементу номера ребер:

Elem[n_elem](left, right, bottom, top),

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

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

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

6. Генерируем краевые условия:

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

(b) Генерируем краевые нижней границы:

i. Берем начальное значение **$i = 0$,**

ii. Подсчитываем индекс ребра:

$id = i$

iii. Генерируем краевое:

Bound(SideBound[0], 0, id),

iv. Прибавляем к i единицу и возвращаемся на шаг (ii), пока

$i < \text{CountX} - 1$.

(c) Генерируем краевые правой границы:

i. Берем начальное значение **$i = 1$,**

ii. Подсчитываем индекс ребра:

$id = i * \text{CountX} + i * (\text{CountX} - 1) - 1$

iii. Генерируем краевое:

Bound(SideBound[1], 1, id),

iv. Прибавляем к i единицу и возвращаемся на шаг (ii), пока

$i < \text{CountZ}$.

(d) Генерируем краевые верхней границы:

- i. Берем начальное значение $i = 0$,
 - ii. Подсчитываем индекс ребра:

$$id = CountX * (CountZ - 1) +$$

$$+ (CountX - 1) * (CountY - 1) + i$$
 - iii. Генерируем краевое:

$$Bound(SideBound[2], 2, id),$$
 - iv. Прибавляем к i единицу и возвращаемся на шаг (ii), пока
 $i < CountX - 1$.
- (e) Генерируем краевые левой границы:
- i. Берем начальное значение $i = 0$,
 - ii. Подсчитываем индекс ребра:

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

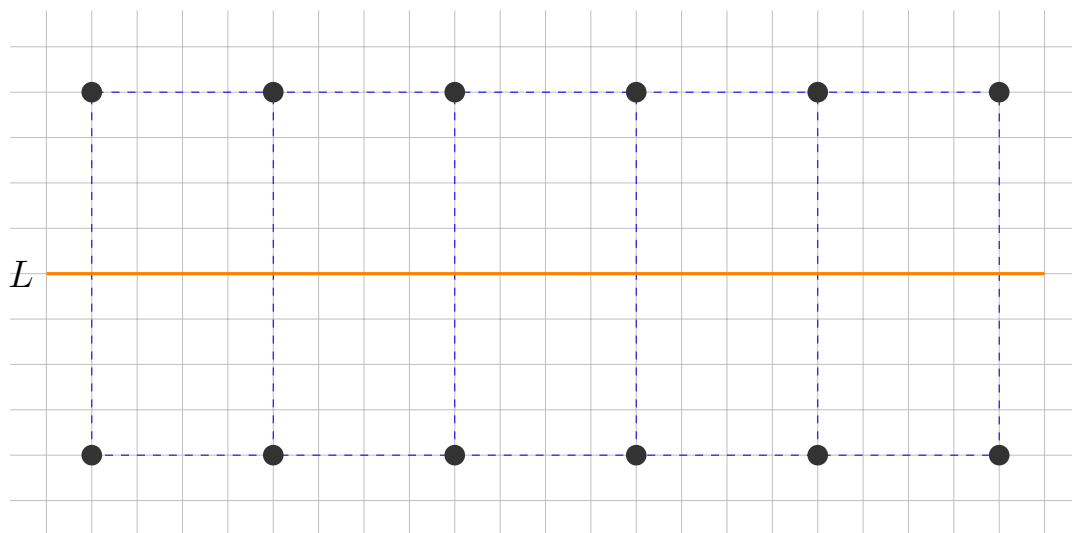
$$Bound(SideBound[3], 3, id),$$
 - iv. Прибавляем к i единицу и возвращаемся на шаг (ii), пока
 $i < CountZ - 1$.
- (f) Сортируем краевые в порядке убывания номера краевого.

2.4. Учет горизонтальных слоев

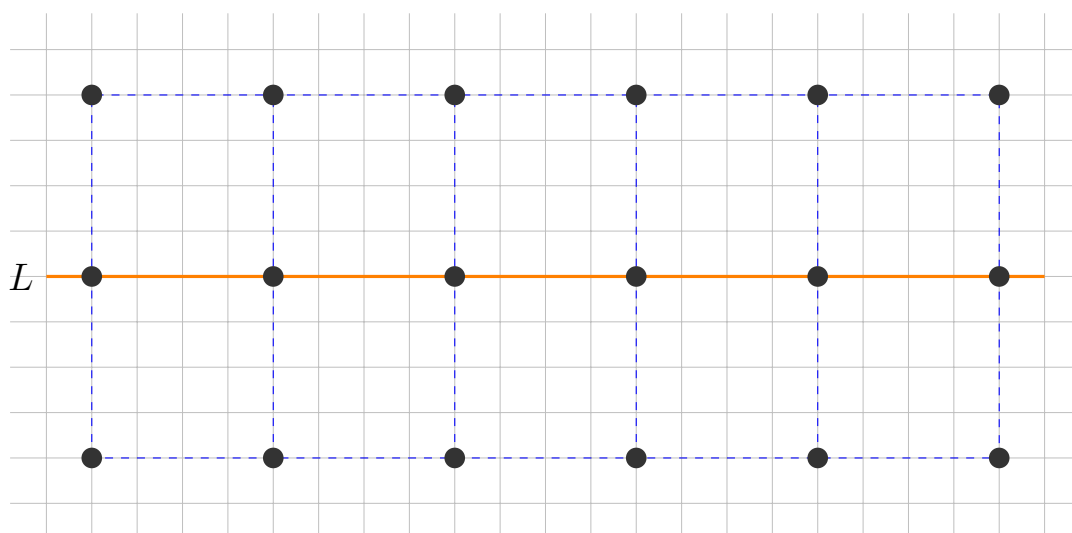
В сетке можно задавать горизонтальные слои (**L**).

Важно: слой должен проходить через узлы.

Неправильный учет слоя:



Правильный учет слоя:



Алгоритм учета горизонтальных слоев:

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

1. Берем начальные значения $\mathbf{Z} = \mathbf{Begin_BG}_z$,
2. Берем шаг из списка \mathbf{H}_z , прибавляем к \mathbf{Z} и получаем $\mathbf{Z_new}$,
3. Если $\mathbf{Z_new} = \mathbf{L}$, то на слое будут лежать узлы,
Если $\mathbf{Z_new} > \mathbf{L}$, добавим шаг, чтобы учесть слой,
4. Если добавленный шаг меньше минимального ($\mathbf{min_step}$), то значение шага прибавляем к предыдущему значению шага, а текущий шаг удаляем,

5. Если шаги в списке \mathbf{H}_z и непройденные слои остались, то возвращаемся к пункту (2).

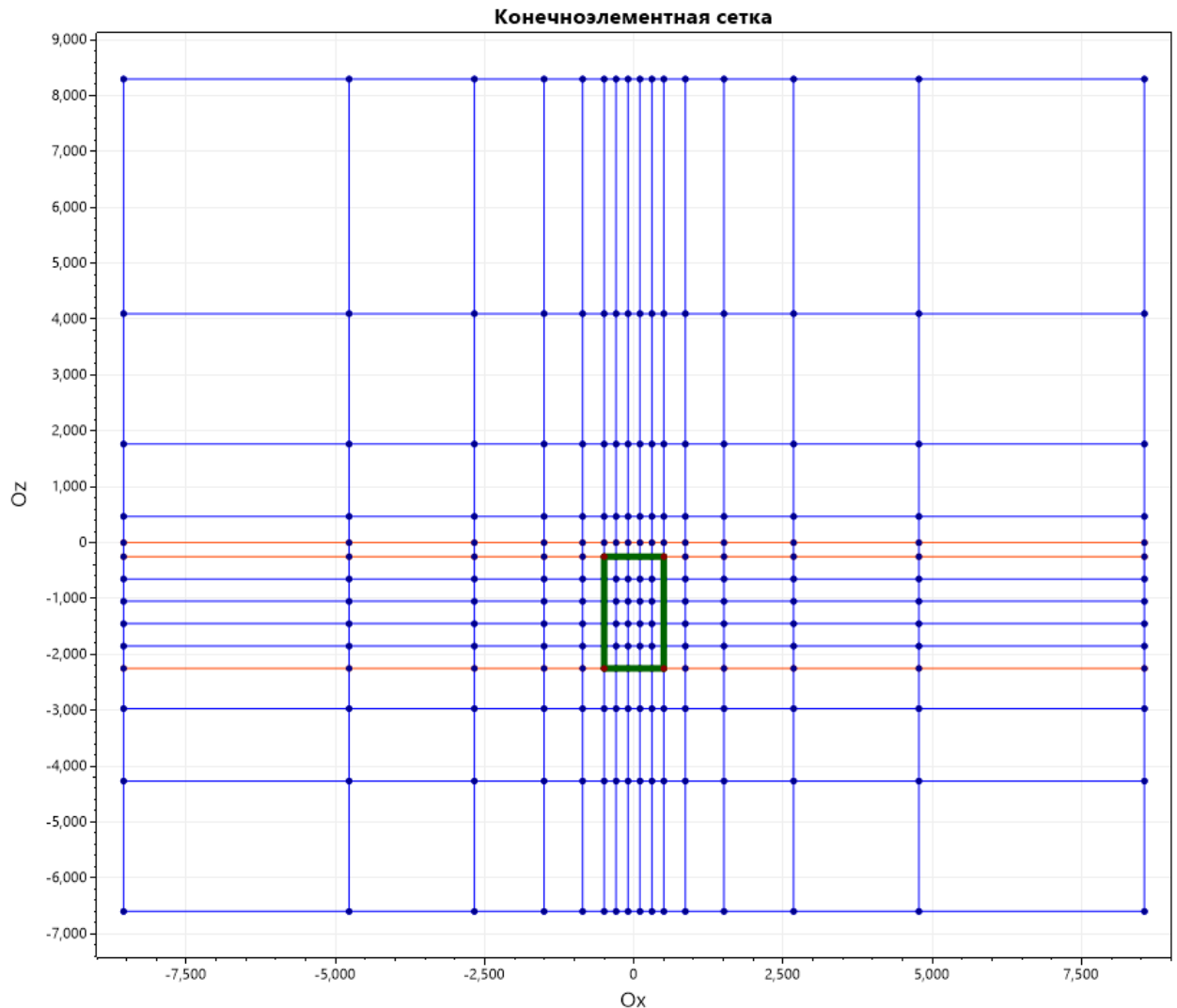


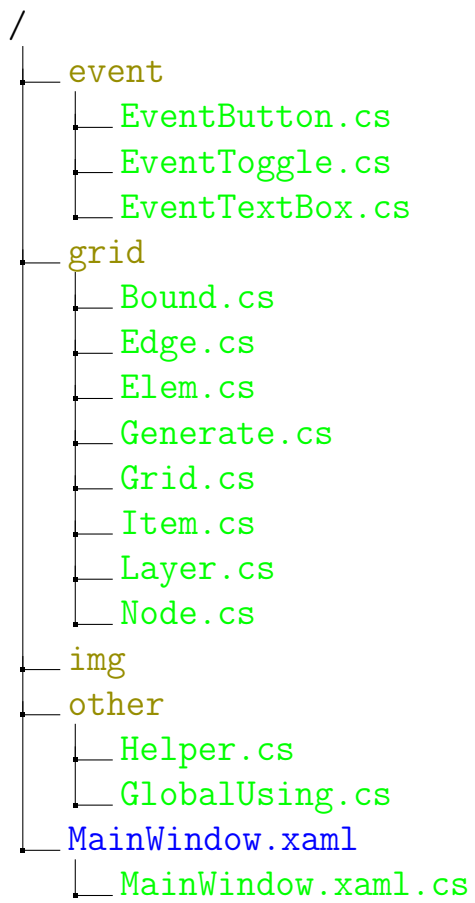
Рисунок 2.1 – Сетка в горизонтальной слоистой среде $L=(0,-250,-500)$

2.5. Руководство по программе

2.5.1. 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.

2.5.2. Структура программы



2.5.3. Реализованные структуры

Структура Item

```
1 public struct Item
2 {
3     public Vector<double> Begin { get; set; } // Нижняя-Левая точка объекта
4     public Vector<double> End { get; set; } // Верхняя-Правая точка объекта
5     public int Nx { get; set; } // Количество разбиений по Оси X
6     public int Nz { get; set; } // Количество разбиений по Оси Z
7     public string Name { get; set; } // Имя объекта
8     public double Sigma { get; set; } // Значение проводимости
9
10    //: Конструктор
11    public Item(Vector<double> begin, Vector<double> end, int nx, int nz,
12               double sigma, string name) { }
13 }
```

Структура Node

```
1 public struct Node
2 {
3     public double X { get; set; } // Координата X
4     public double Z { get; set; } // Координата Z
5
6     public Node(double _X, double _Z) {
7         (X, Z) = (_X, _Z);
8     }
9
10    public void Deconstruct(out double x,
11                           out double z) {
12        (x, z) = (X, Z);
13    }
14 }
```

Структура Edge

```
1 public struct Edge
2 {
3     public Node NodeBegin { get; set; } // Начальный узел ребра
4     public Node NodeEnd   { get; set; } // Конечный узел ребра
5
6     public Edge(Node _begin, Node _end) {
7         NodeBegin = _begin;
8         NodeEnd   = _end;
9     }
10
11     public void Deconstruct(out Node begin,
12                             out Node end) {
13         (begin, end) = (NodeBegin, NodeEnd);
14     }
15 }
```

Структура Elem

```
1 public struct Elem
2 {
3     public int[] Node;           // Номера узлов конечного элемента
4     public int[] Edge;          // Номера ребер конечного элемента
5     public int Material { get; set; } // Номер материала
6
7     public Elem(params int[] node) {
8         Node = node;
9     }
10
11     public void Deconstruct(out int[] nodes,
12                             out int[] edges) {
13         nodes = Node;
14         edges = Edge;
15     }
16 }
```

Структура Bound

```
1 public struct Bound
2 {
3     public int Edge    { get; set; } // Номер ребра
4     public int NumBound { get; set; } // Номер краевого
5     public int NumSide  { get; set; } // Номер стороны
6
7     public Bound(int num, int side, int edge) {
8         NumBound = num;
9         NumSide = side;
10        Edge = edge;
11    }
12
13    public void Deconstruct(out int num, out int side, out int edge) {
14        num = NumBound;
15        side = NumSide;
16        edge = Edge;
17    }
18 }
```

Структура Layer

```
1 public struct Layer
2 {
3     public double Z    { get; set; } // Координата слоя
4     public double Sigma { get; set; } // Значение проводимости
5
6     public Layer(double z, double sigma) {
7         this.Z = z;
8         this.Sigma = sigma;
9     }
10 }
```

2.5.4. Описание пользовательского интерфейса

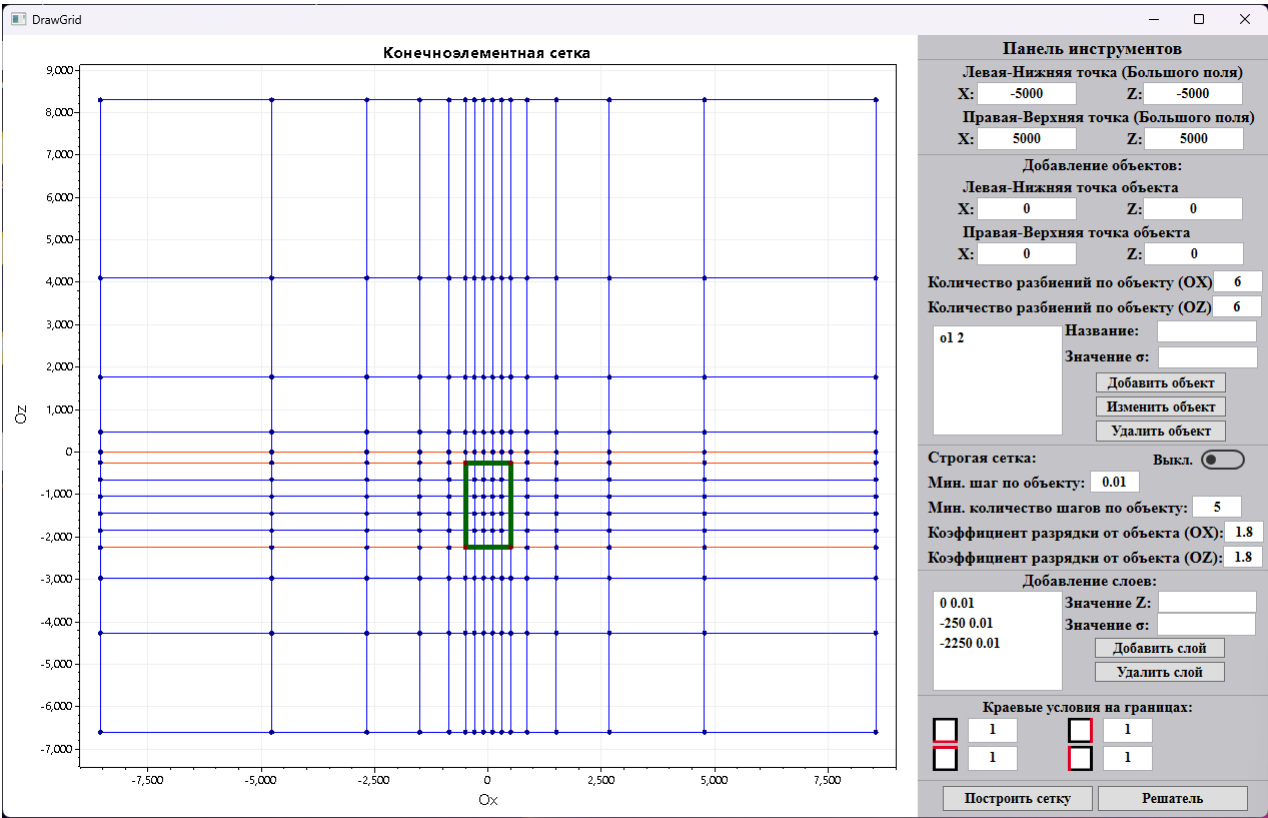


Рисунок 2.2 – Вид пользовательского интерфейса



Таблица 2.1 – Описание бокового меню

Пункт меню	Назначение
Пункты меню для задания координат большого поля	
Левая-Нижняя точка (Большого поля)	Задаёт точку Begin_BG
Правая-Верхняя точка (Большого поля)	Задаёт точку End_BG
Пункты меню для задания объекта	
Левая-Нижняя точка объекта	Задаёт точку Begin
Правая-Верхняя точка объекта	Задаёт точку End

Продолжение таблицы 2.1

Пункт меню	Назначение
Количество разбиений по объекту (Ox)	Задаёт параметр объекта Nx
Количество разбиений по объекту (Oz)	Задаёт параметр объекта Nz
Название	Задаёт параметр объекта Name
Значение σ	Задаёт параметр объекта Sigma
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	Задаёт параметр min_step
Минимальное количество шагов по объекту	Задаёт параметр count_step
Коэффициент разрядки от объекта (OX)	Задаёт параметр Kx
Коэффициент разрядки от объекта (OZ)	Задаёт параметр Kz
Пункты меню для задания слоя	
Значение Z	Задаёт значение слоя
Значение σ	Задаёт проводимость слоя
Кнопка «Добавить слой»	Добавляет значение слоя на сетку
Кнопка «Удалить слой»	Удаляет значение слоя из сетки

Продолжение таблицы 2.1

Пункт меню	Назначение
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

2.5.5. Взаимодействие с Plot из пакета ScottPlot

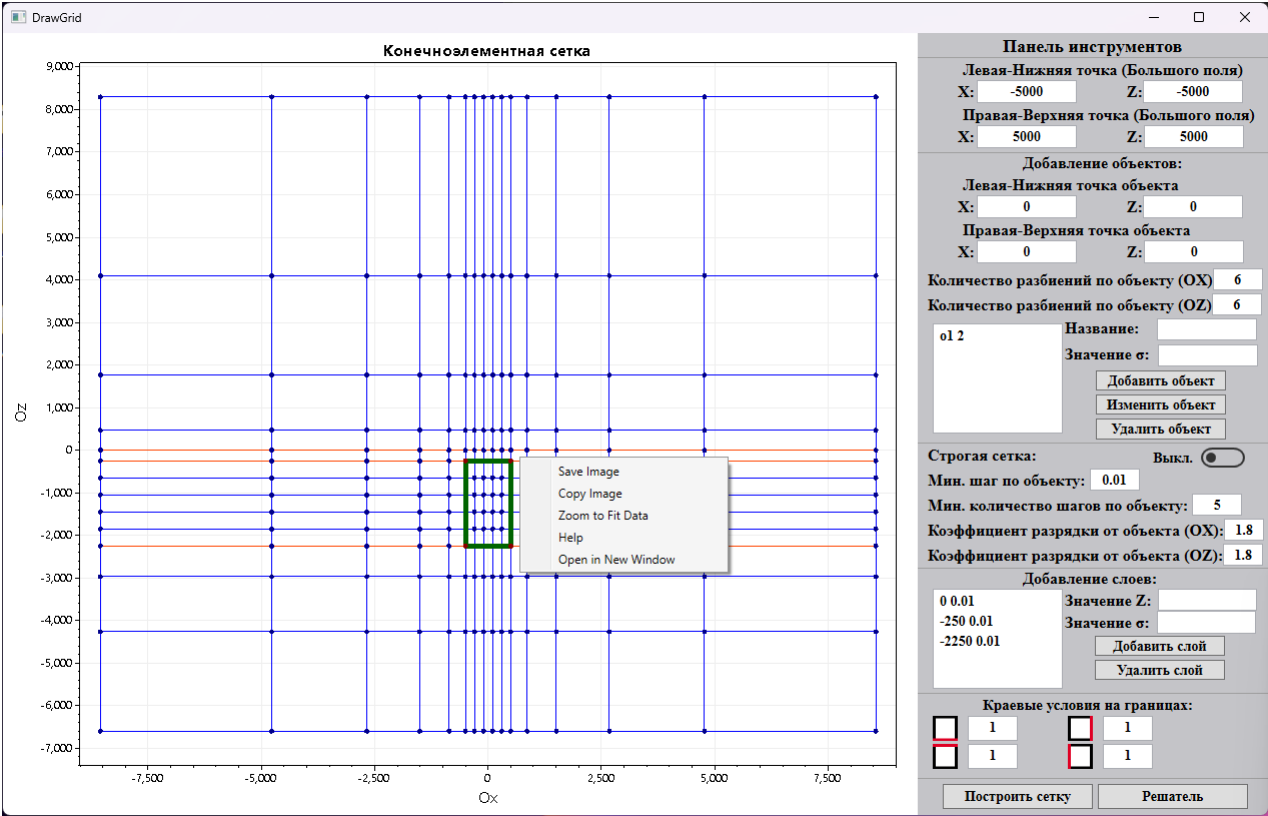


Таблица 2.2 – Описание инструментария пакета

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 2.2

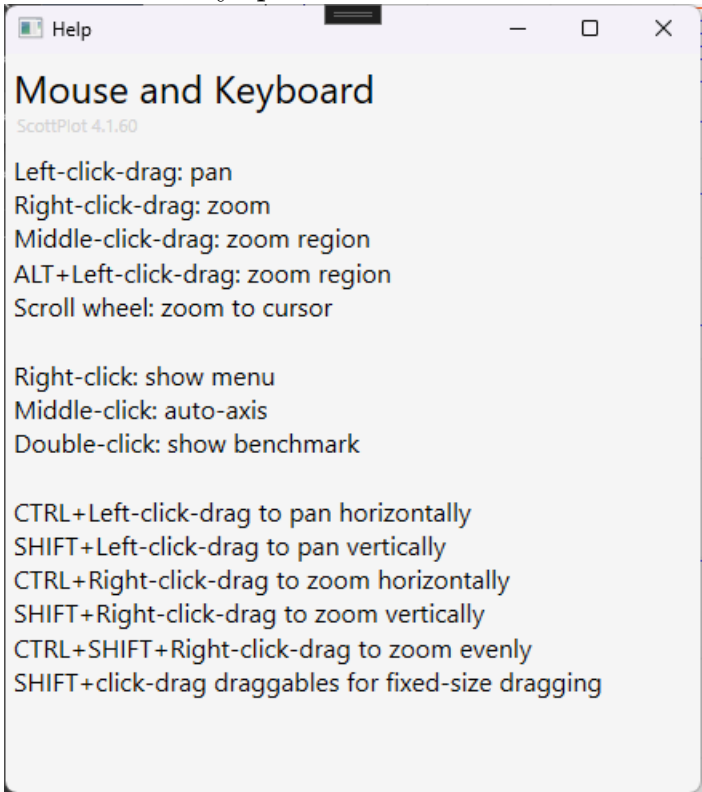
Пункт меню	Назначение
Help	<p>Открывает окошко с инструкцией для управления Plot</p>  <p>The screenshot shows a window titled 'Help' with the subtitle 'Mouse and Keyboard' and version 'ScottPlot 4.1.60'. It lists the following actions:</p> <ul style="list-style-type: none"> Left-click-drag: pan Right-click-drag: zoom Middle-click-drag: zoom region ALT+Left-click-drag: zoom region Scroll wheel: zoom to cursor Right-click: show menu Middle-click: auto-axis Double-click: show benchmark 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
Open in New Window	Открывает Plot в новом окошке

Таблица 2.3 – 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 2.3

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

2.6. Примеры сеток

2.6.1. Количество объектов

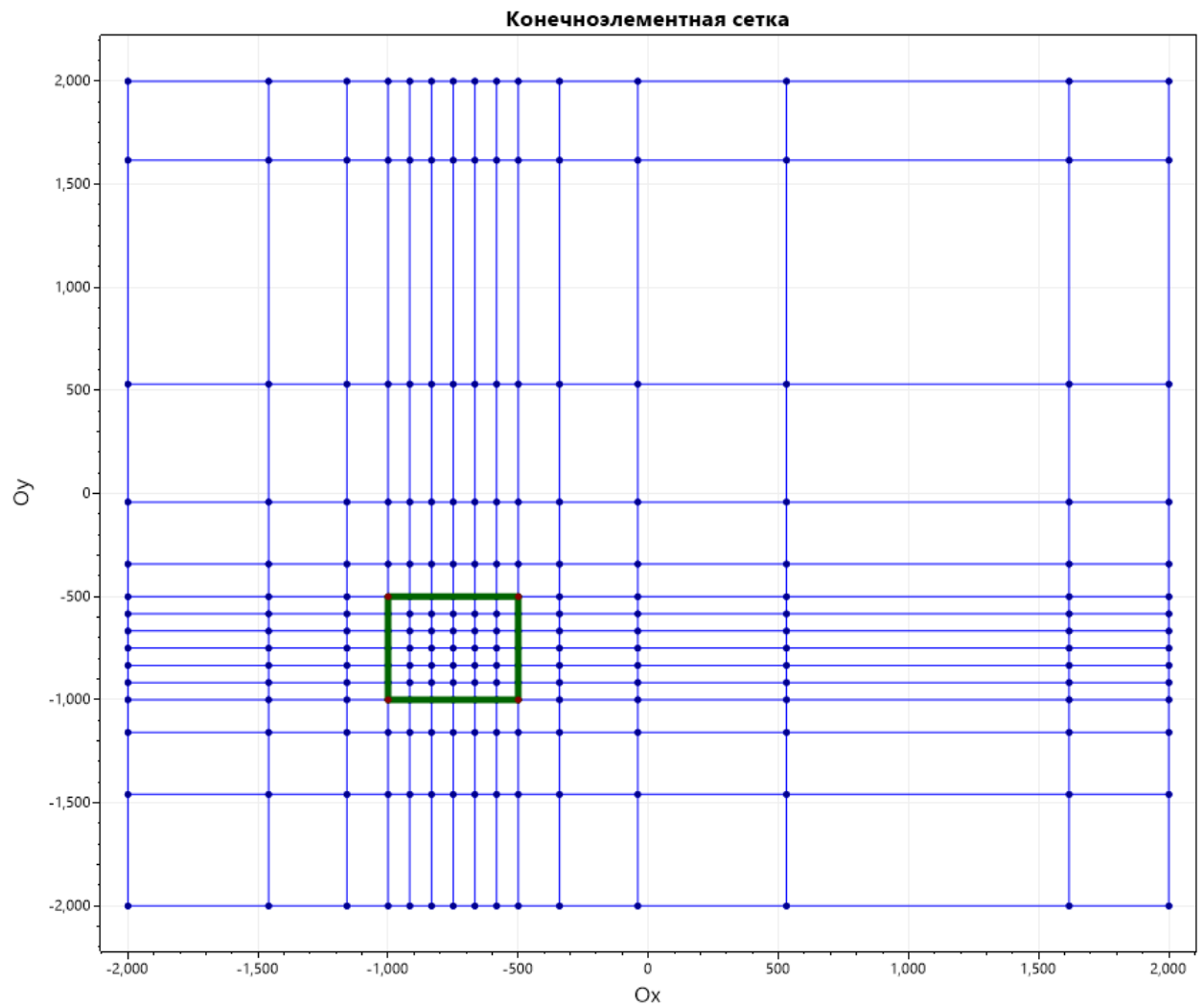


Рисунок 2.3 – Сетка с одним объектом

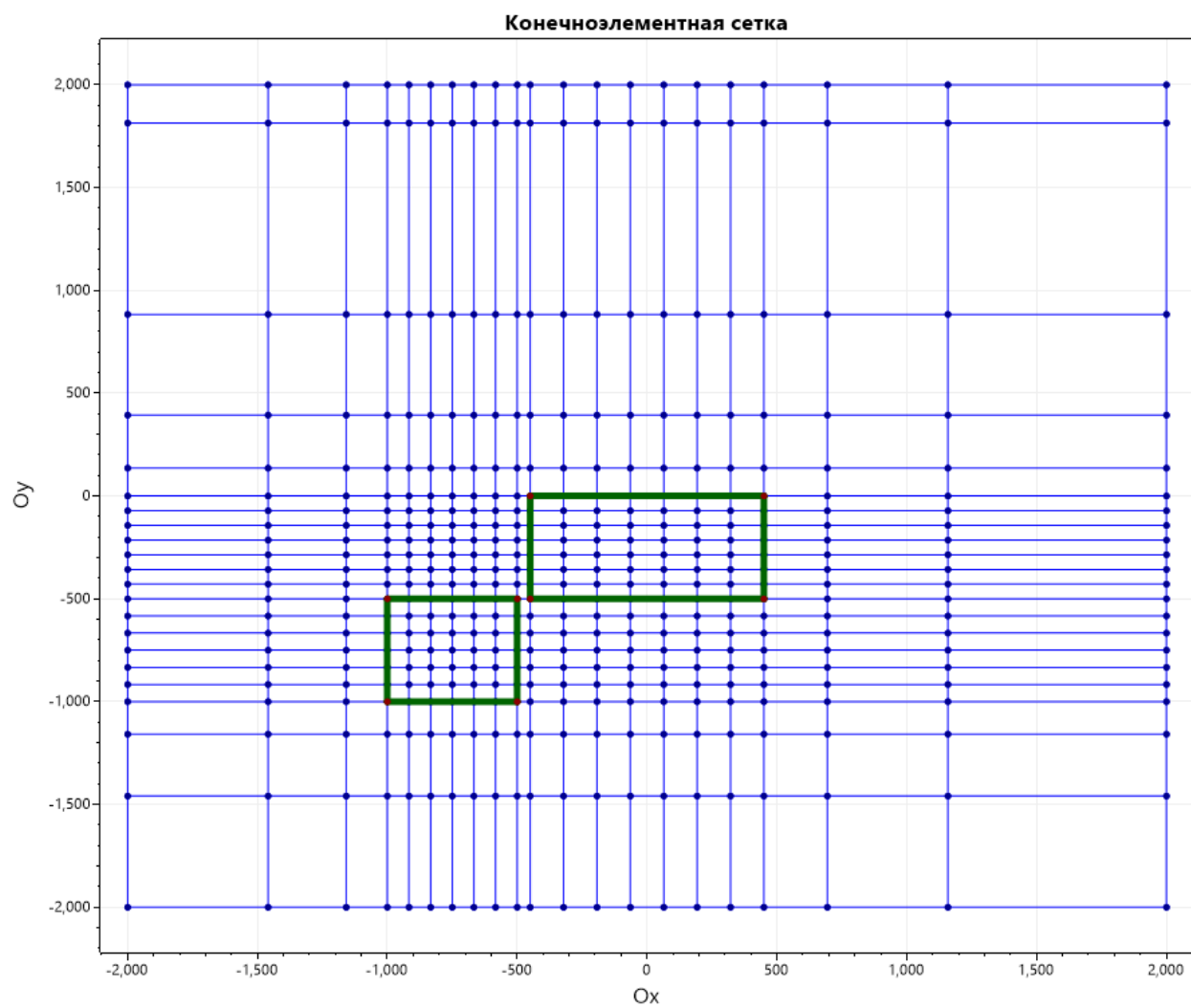


Рисунок 2.4 – Сетка с двумя объектами

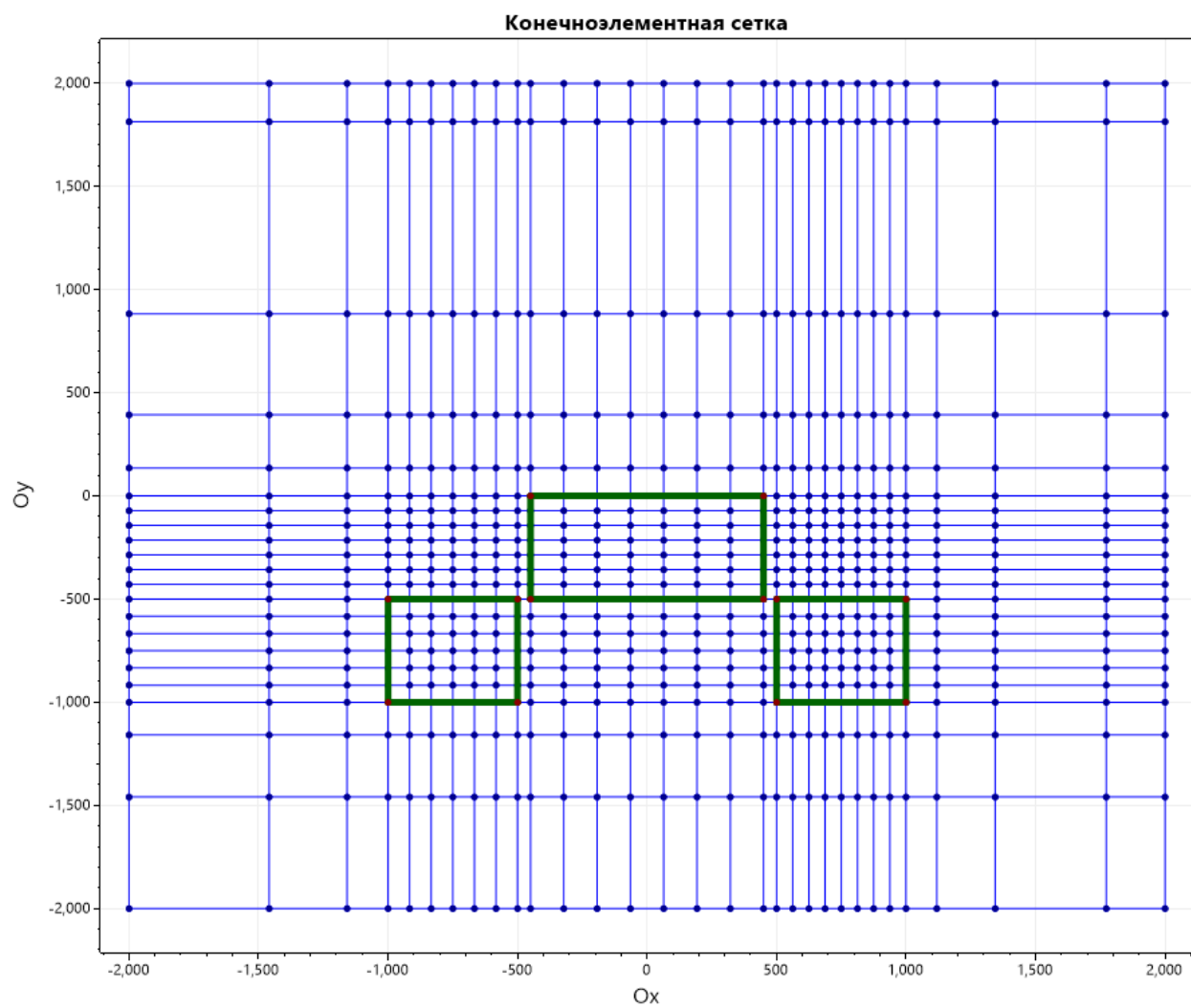


Рисунок 2.5 – Сетка с тремя объектами

2.6.2. Строгие и нестрогие сетки

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

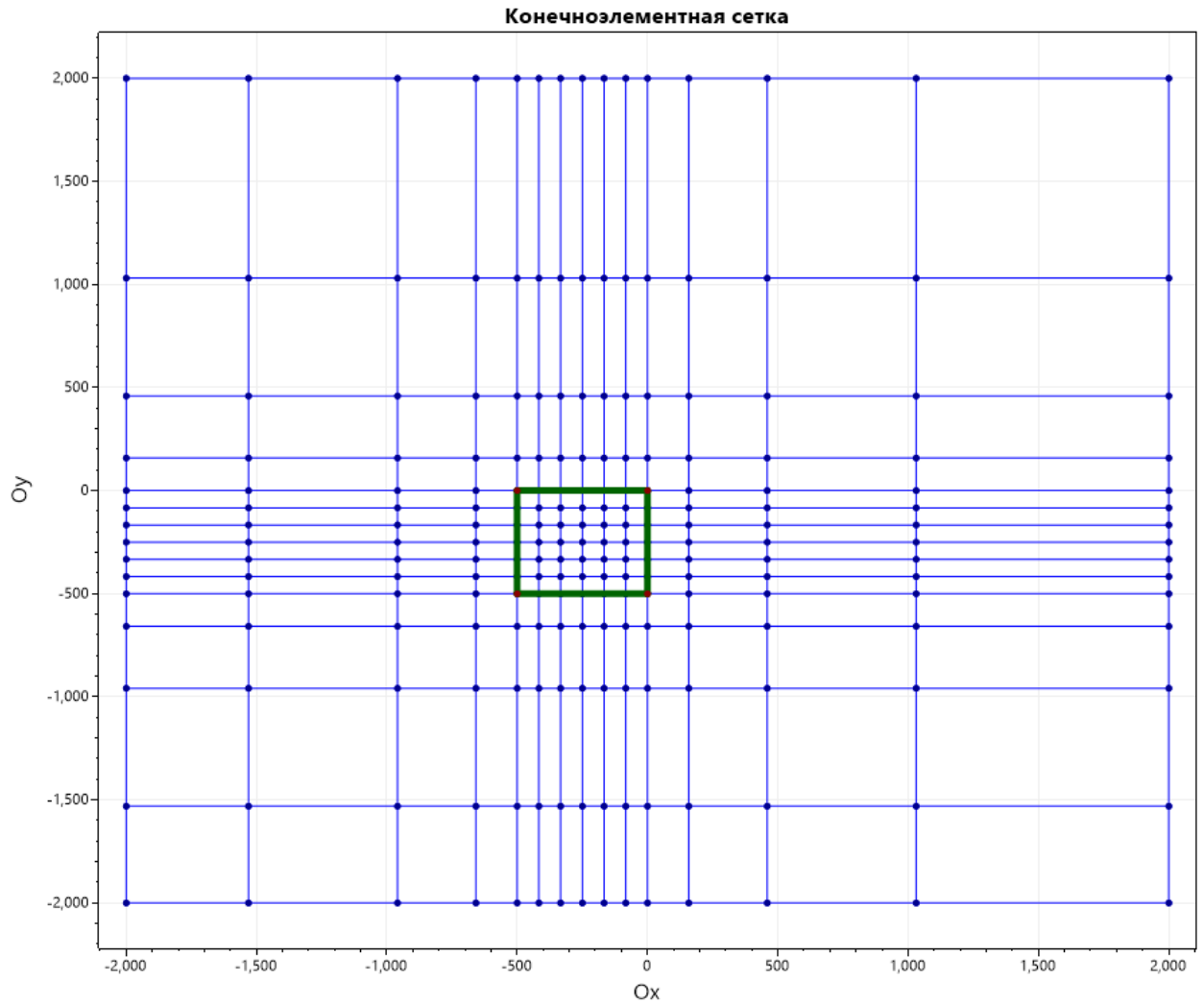


Рисунок 2.6 – Строгая сетка $\text{Begin_BG}=(-2000,2000)$, $\text{End_BG}=(2000, 2000)$

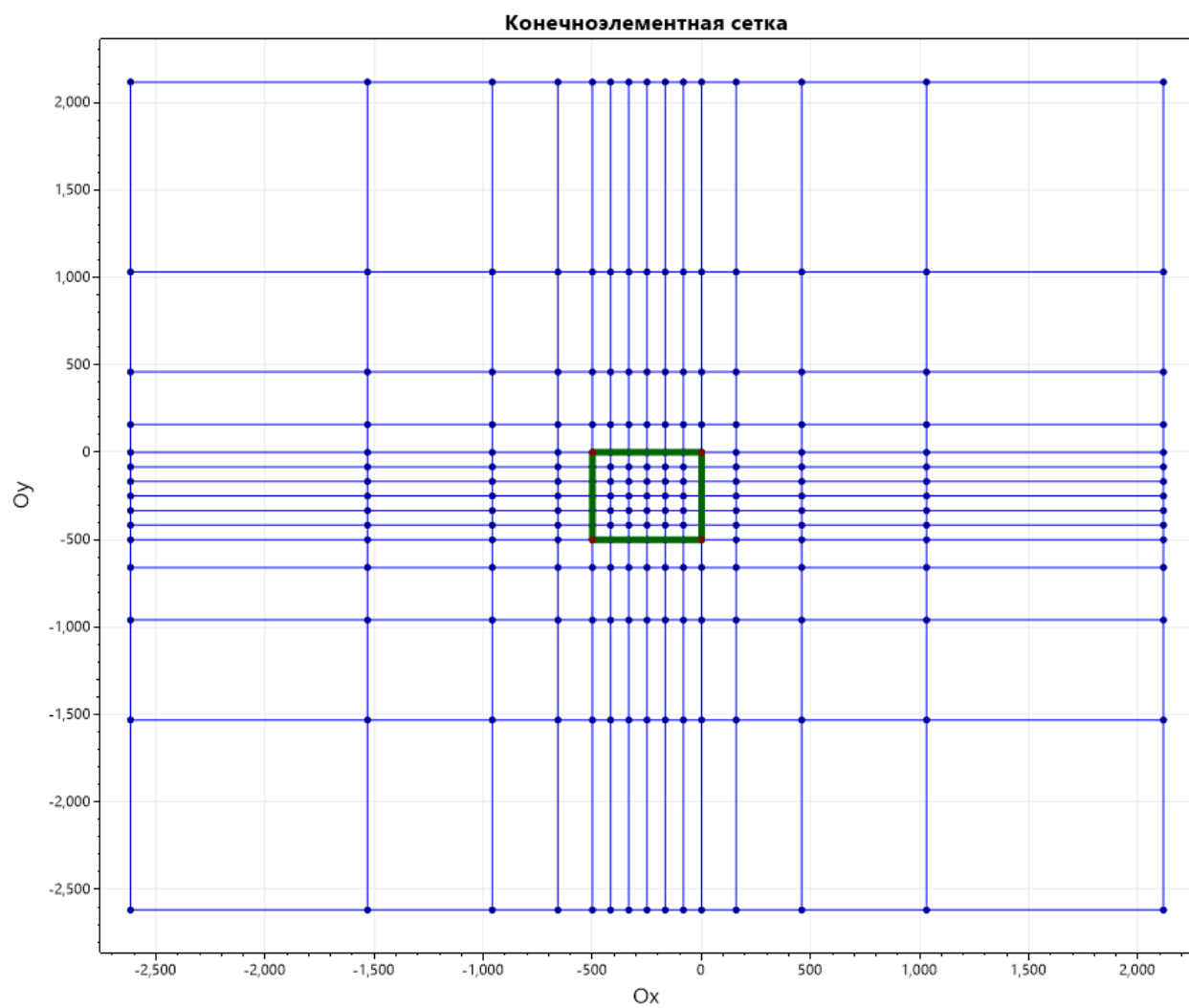


Рисунок 2.7 – Нестрогая сетка $\text{Begin_BG}=(-2000,-2000)$, $\text{End_BG}=(2000, 2000)$

2.6.3. Параллельные объекты

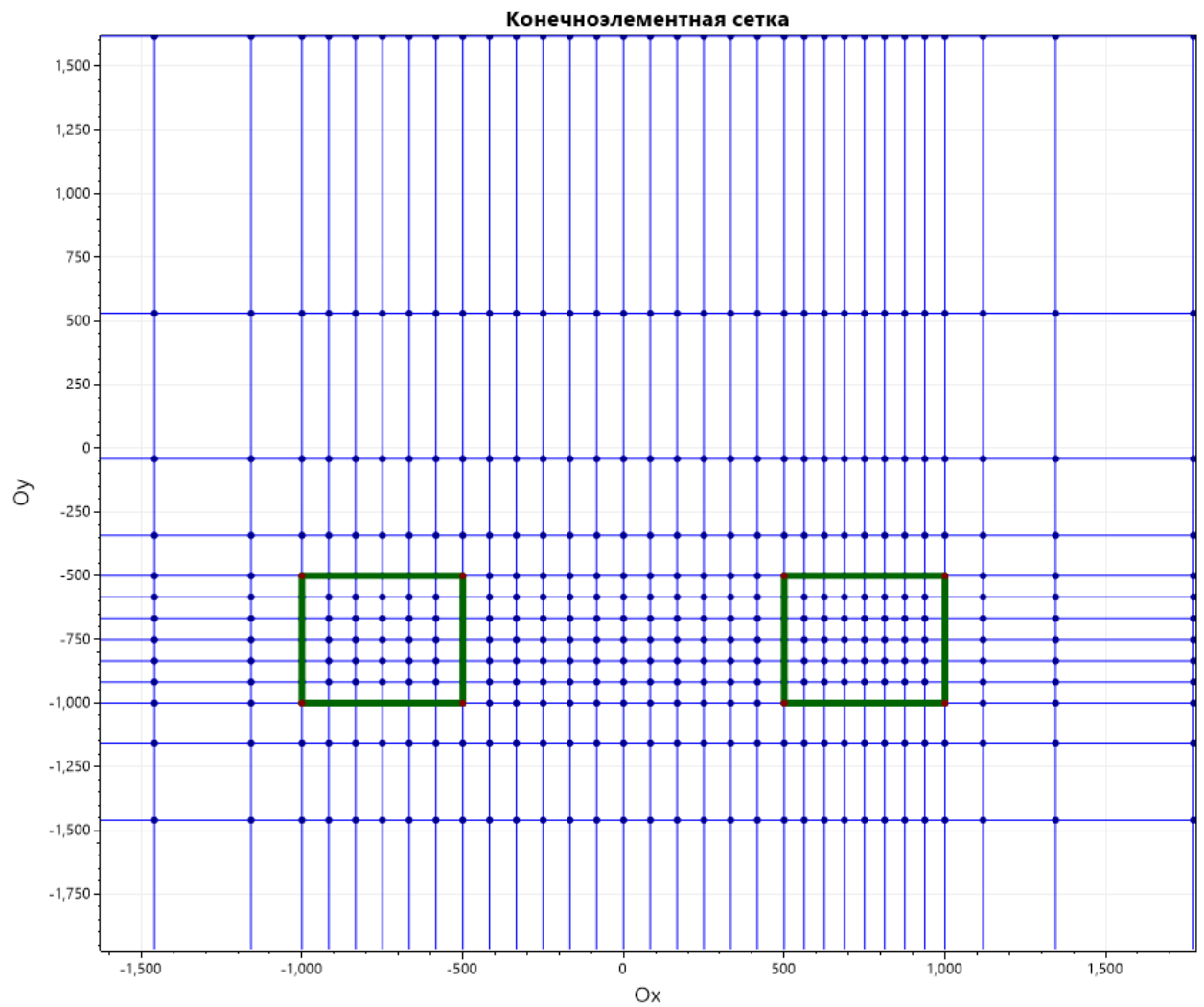


Рисунок 2.8 – Параллельные объекты с разным количеством разбиений

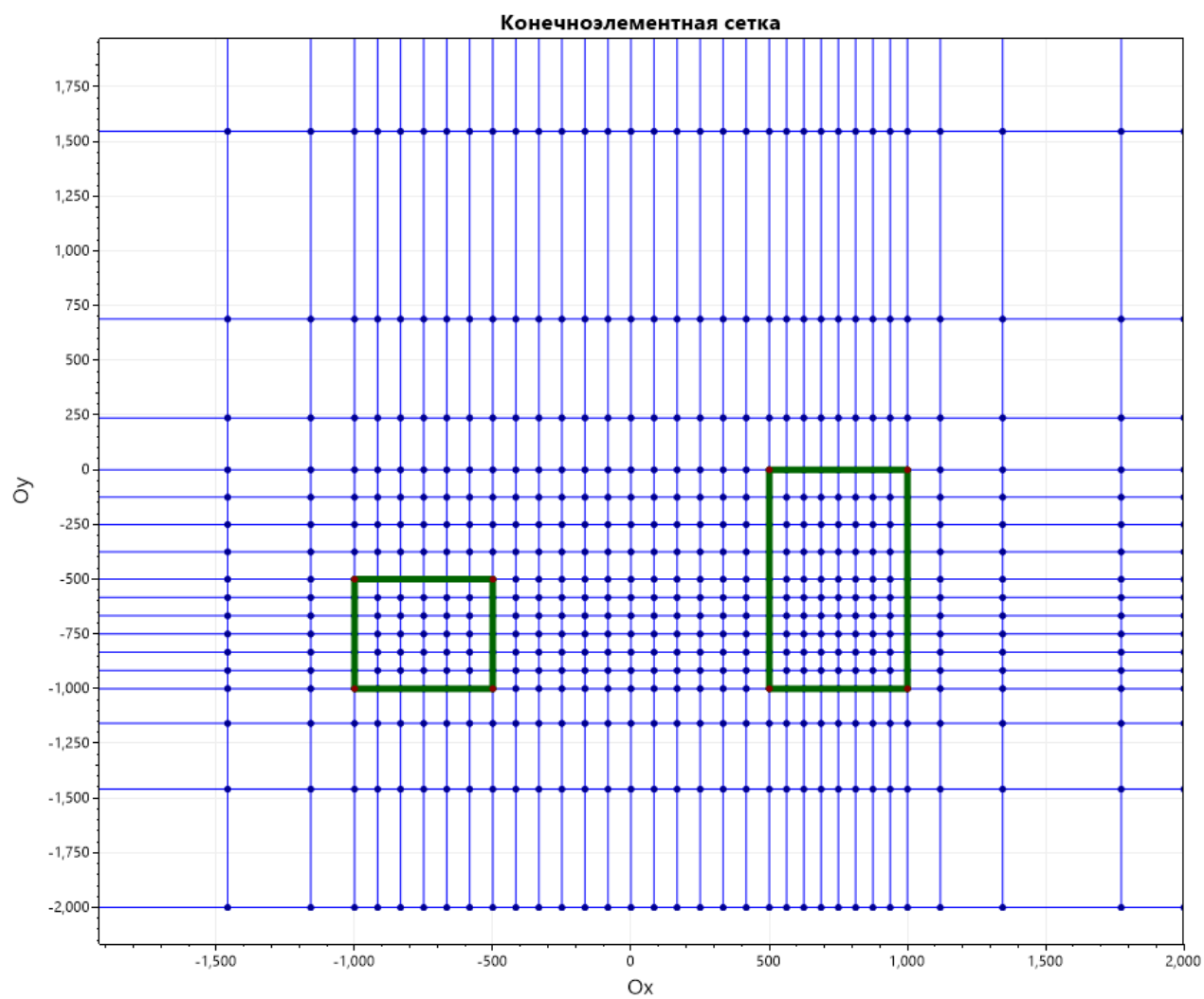


Рисунок 2.9 – Параллельные объекты с разной размерностью

3. Процедура решения СЛАУ с использованием библиотеки Intel OneMKL

3.1. Введение в Intel oneAPI Math Kernel Library

Intel oneAPI Math Kernel Library (oneMKL) - это вычислительная математическая библиотека, состоящая из высокооптимизированных многопоточных подпрограмм для приложений, требующих максимальной производительности. Библиотека предоставляет интерфейсы языков программирования Fortan и C.

Из математической библиотеки были использованы процедуры OneMKL PARDISO, который поддерживает прямой разреженный решатель, итеративный разреженный решатель и поддерживающие разреженные процедуры BLAS для решения разреженных систем уравнения.

3.2. Используемые при разработке средства

1. Visual Studio 2022 – основная среда разработки,
2. C, C++ – основные языки логики приложения,
3. Intel OneMKL - библиотека оптимизированных математических процедур для научных приложений.

3.3. Разреженный формат хранения матрицы CSR3

Формат CSR3 (с 3 массивами), принятый для прямых разреженных решателей, является разновидностью формата CSR (с 4 массивами).

Для симметричных матриц необходимо сохранить только верхнюю треугольную половину матрицы (верхний треугольный формат) или нижнюю треугольную половину матрицы (нижний треугольный формат).

Формат хранения разреженных матриц Intel one API MKL для прямых разреженных решателей определяется тремя массивами: a , ja и ia .

a - комплексный массив, содержащий ненулевые элементы разреженной матрицы.

ja - i -й элемент целочисленного массива ja - это номер столбца, который содержит i -й элемент в массиве a .

ia - j -й элемент целочисленного массива ia дает индекс элемента в массиве a , который является первым ненулевым элементом в строке j .

Длина массивов a и ja равна количеству ненулевых элементов в матрице.

$$B = \begin{bmatrix} 1 & -1 & * & -3 & * \\ -1 & 5 & * & * & * \\ * & * & 4 & 6 & 4 \\ -3 & * & 6 & 7 & * \\ * & * & 4 & * & -5 \end{bmatrix} \quad (3.1)$$

Матрица \mathbf{B} в разреженном строчно-столбцовом формате:

$$di = [1 \quad 5 \quad 4 \quad 7 \quad -5]$$

$$gg = [-1 \quad -3 \quad 6 \quad 4]$$

$$ig = [0 \quad 0 \quad 1 \quad 1 \quad 3 \quad 4]$$

$$jg = [0 \quad 0 \quad 2 \quad 2]$$

Матрица \mathbf{B} в формате CSR3:

$$a = [1 \quad -1 \quad -3 \quad 5 \quad 4 \quad 6 \quad 4 \quad 7 \quad -5]$$

ja = [0 1 3 1 2 3 4 3 4]

ia = [0 3 4 7 8 9]

Процедура преобразования матрицы в CSR формат

```
1 void FromRSFToCSR_Real_2_Sym(int nb, int* ig, int* jg, complex<double>* di,
2     complex<double>* gg, MKL_INT* ia, MKL_INT* ja, MKL_Complex16* a)
3 {
4     vector<MKL_INT> adr;
5
6     // подсчитываем число элементов в каждой строчке
7     adr.resize(nb, 0);
8
9     for (int i = 0; i < nb; i++) {
10         adr[i] += 1; // диагональ
11         // верхний треугольник
12         for (int j = ig[i]; j <= ig[i + 1] - 1; j++)
13             adr[jg[j]]++;
14     }
15
16     // ia
17     ia[0] = 0;
18     for (int i = 0; i < nb; i++)
19         ia[i + 1] = ia[i] + adr[i];
20
21     // ja, a
22     for (int i = 0; i < ig[nb] + nb; i++)
23         a[i] = MKL_Complex16{ 0, 0 };
24
25     for (int i = 0; i < nb; i++)
26         adr[i] = ia[i]; // в какую позицию заносить значение
27
28     // диагональ
29     for (int i = 0; i < nb; i++) {
30         ja[adr[i]] = i;
31         a[adr[i]] = MKL_Complex16{ di[i].real(), di[i].imag() };
32         adr[i]++;
33     }
34
35     // верхний треугольник
36     for (int i = 0; i < nb; i++) {
37         for (int j = ig[i]; j <= ig[i + 1] - 1; j++) {
38             int k = jg[j];
39             ja[adr[k]] = i;
40             a[adr[k]] = MKL_Complex16{ gg[j].real(), gg[j].imag() };
```

```

41         adr[k]++;
42     }
43 }
44 }

```

Процедура решения СЛАУ

```

1  int _tmain(int argc, _TCHAR* argv[])
2  {
3      // Путь к СЛАУ
4      string path = "slauBIN/";
5
6      logfile.open(path + "pardiso64.log");
7      if (!logfile) {
8          cerr << "Cannot open pardiso64.log" << endl;
9          return 1;
10     }
11
12     int ig_n_1 = 0;
13     int sz_ia = 0;
14     int sz_ja = 0;
15
16     MKL_Complex16* pr = NULL;
17     MKL_Complex16* x = NULL;
18
19     // Исходная матрица в разреженном строчно-столбцовом формате
20     int nb;
21     int* ig = NULL;
22     int* jg = NULL;
23     complex<double>* di = NULL;
24     complex<double>* gg = NULL;
25
26     // Матрица, сконвертированная в разреженный строчный формат
27     MKL_INT* ia = NULL;
28     MKL_INT* ja = NULL;
29     MKL_Complex16* a = NULL;
30
31     // Для PARDISO
32     clock_t begin = clock();
33     MKL_INT n = 0;
34     MKL_INT mtype = 6; // комплексная симметричная
35     MKL_INT nrhs = 1;
36     void* pt[64];
37     MKL_INT maxfct = 1;
38     MKL_INT      mnum = 1;
39     MKL_INT msglvl = 1;
40     MKL_INT phase = 13;
41     MKL_INT* perm = NULL;

```

```

42     MKL_INT iparm[64];
43     MKL_INT info = -100;
44
45     for (int i = 0; i < 64; i++)
46         pt[i] = 0;
47
48     iparm[0] = 0; //iparm(2) - iparm(64) заполняются значениями по умолчанию.
49     //iparm[0] = 1; // Нужно указать все значения в компонентах iparm(2) - iparm(64).
50     for (int i = 1; i < 64; i++)
51         iparm[i] = 0;
52
53     // Чтение размерности nb
54     Read_Long_From_Txt_File((path + "kuslau").c_str(), &nb);
55
56     // ig
57     ig = new int[nb + 1];
58     Read_Bin_File_Of_Long((path + "ig").c_str(), ig, nb + 1, 1);
59     ig_n_1 = ig[nb];
60
61     // jg
62     jg = new int[ig_n_1];
63     Read_Bin_File_Of_Long((path + "jg").c_str(), jg, ig_n_1, 1);
64
65     // di
66     di = new complex<double>[nb];
67     Read_Bin_File_Of_Complex((path + "di").c_str(), di, nb, 1);
68
69     // gg
70     gg = new complex<double>[ig_n_1];
71     Read_Bin_File_Of_Complex((path + "gg").c_str(), gg, ig_n_1, 1);
72
73     // pr
74     pr = new MKL_Complex16[nb];
75     Read_Bin_File_Of_Complex((path + "pr").c_str(), pr, nb, 1);
76
77     // Конвертирование в CSR
78     FromRSFTtoCSR_Real_1_Sym(nb, ig, &sz_ia, &sz_ja);
79     ia = new MKL_INT[sz_ia];
80     ja = new MKL_INT[sz_ja];
81     a = new MKL_Complex16[sz_ja];
82     FromRSFTtoCSR_Real_2_Sym(nb, ig, jg, di, gg, ia, ja, a);
83
84     for (int i = 0; i < sz_ia; i++)
85         ia[i]++;
86
87     for (int i = 0; i < sz_ja; i++)
88         ja[i]++;
89
90     if (ig) { delete[] ig; ig = NULL; }
91     if (jg) { delete[] jg; jg = NULL; }

```

```

92     if (di) { delete[] di; di = NULL; }
93     if (gg) { delete[] gg; gg = NULL; }
94
95     // Запуск PARDISO
96     n = nb;
97     x = new MKL_Complex16[nb];
98     perm = new MKL_INT[nb];
99     for (int i = 0; i < nb; i++) {
100         x[i] = MKL_Complex16{0, 0};
101         perm[i] = 0;
102     }
103
104     cout << "pardiso start.." << endl << flush;
105     PARDISO(pt, &maxfct, &mnum, &mtype, &phase, &n, a, ia, ja, perm, &nrhs, iparm,
106             &msglvl, pr, x, &info);
107
108     clock_t time = (clock() - begin) / CLOCKS_PER_SEC;
109
110     ofstream fout;
111     fout.open((path + "kit").c_str(), ios_base::app);
112     fout << "n=" << nb << " pardiso time=" << time << " s" << endl;
113     fout.close();
114
115     // Запись решения
116     Write_Txt_File_Of_Complex((path + "q.txt").c_str(), x, nb, 1);
117     Write_Txt_File_Of_Complex(string("slauTXT/q.txt").c_str(), x, nb, 1);
118
119     // Запись и вывод info
120     cout << "info=" << info << endl;
121     logfile << "info=" << info << endl;
122
123     if (a) { delete[] a; a = NULL; }
124     if (x) { delete[] x; x = NULL; }
125     if (pr) { delete[] pr; pr = NULL; }
126     if (ja) { delete[] ja; ja = NULL; }
127     if (ia) { delete[] ia; ia = NULL; }
128     if (perm) { delete[] perm; perm = NULL; }
129
130     logfile.close();
131     logfile.clear();
132
133     system("pause");
134     return 0;
135 }

```