МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК
И ПРОВЕРЕНО НА ОБЪЕМ
ЗАИМСТВОВАНИЯ
Заведующий кафедрой
д.п.н., профессор
_____И.Г. Захарова
_____29 июня 2018 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ВИЗУАЛИЗАЦИИ РЕШЕНИЯ ЧИСЛЕННЫХ ЗАДАЧ УРАВНЕНИЙ МАТЕМАТИЧЕСКОЙ ФИЗИКИ

02.04.03. Математическое обеспечение и администрирование информационных систем

Магистерская программа «Высокопроизводительные вычислительные системы»

Выполнил работу
Студент 2 курса
очной формы обучения
Метелёв Андрей Сергеевич

Научный руководитель
к.ф-м.н., доцент
Гаврилова Наталья Михайловна

Рецензент
к.ф-м. н, доцент
Ступников Андрей Анатольевич

Тюмень – 2018

Оглавление

Введение

Совершенствование вычислительной техники и широкое распространение персональных компьютеров открыло огромные перспективы для исследования процессов и явлений окружающего мира, включая сюда и человеческое общество. Моделирование является одним из способов познания мира.

Понятие моделирования достаточно сложное, оно включает в себя огромное разнообразие способов представления объектов и процессов: от создания натуральных моделей (уменьшенных или увеличенных копий реальных объектов) до вывода математических формул. Объект, который получается в результате моделирования, называется **моделью**. Это может быть математическая формула, графическое представление, а не обязательно реальный объект. Чаще всего в моделях воссоздаются какие-нибудь важные для данного исследования элементы, а остальными пренебрегают.

Компьютерное моделирование – это в определенной степени, то же самое моделирование, но реализуемое с помощью компьютерной техники. Компьютеры позволяют обработать большой объем информации, а также быстро выполнить необходимые численные операции. Кроме того, современные ЭВМ позволяют наглядно показать графическую модель того или иного объекта.

В рамках данной выпускной квалификационной работы рассматривается численное моделирование решений краевой задачи уравнения в частных производных параболического типа, классическим примером которой является краевая задача для уравнения теплопроводности. Существуют различные численные методы решения краевых задач для уравнения теплопроводности, в частности, метод переменных направлений.

Цель выпускной квалификационной работы - разработка приложения для визуализации численного решения краевой задачи уравнения теплопроводности.

Для достижения данной цели требуется выполнить следующие задачи:

- Изучить конечно-разностные методы для решения уравнений параболического типа с двумя пространственными переменными.
- Изучить алгоритмы визуализации распределения температуры.
- Сравнить параллельную и последовательную реализацию метода переменных направлений.
- Реализовать сохранение анимации распределения температуры в файл.

Глава 1. Постановка задачи и описание численного решения задачи теплопроводности

1.1. Постановка задачи

Студентам, при выполнении лабораторных работ на тему: “Численные методы решения краевых задач для уравнений в частных производных” необходимо не просто решить поставленную задачу, но и, для лучшего понимания темы, посмотреть визуализированное решение на разных временных слоях, задать свои входные данные. В связи с этим, есть необходимость в приложении, в котором можно задавать различные входные данные, просматривать численное и визуализированное решения.

Необходимо визуализировать численное решение двумерного уравнения теплопроводности. Данную задачу можно разбить на две подзадачи: реализация решения краевой задачи для двумерного уравнения теплопроводности и визуализация данного решения.

1.2. Краевая задача для двумерного уравнения теплопроводности

Рассмотрим двумерное уравнение параболического типа в прямоугольнике со сторонами l_1 и l_2 и граничными условиями I - го рода. Для пространственно-временной области:

G_T=G×[0,T], t ∈[0,T], G=G+Γ,G=$l_1 \times l_2$

Рассмотрим следующую задачу:

$\frac{\partial u}{\partial t}=\alpha(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}) + f(x,y,t), x \in (0,l_1), y \in (0,l_2), t > 0;$

$u(x,0,t) = \varphi_1(x,t), x \in [0,l_1], y = 0, t > 0,$

$u(x,l_2,t) = \varphi_2(x,t), x \in [0,l_1], y = l_2, t > 0,$

$u(0,y,t) = \varphi_3(y,t), x = 0, y \in [0,l_2], t > 0,$

$u(l_1,y,t) = \varphi_4(y,t), x = l_1, y \in [0,l_2], t > 0,$

$u(x,y,0) = \psi(x,y), x \in [0,l_1], y \in [0,l_2], t = 0.$

Введем пространственно-временную сетку с шагами $h1,h2,\tau$ соответственно по переменным x, y, t:

$\omega_{h_1h_2}^{\tau}=\{x_i = ih_1, i = \overline{0,I};\ y_j = jh_2, j = \overline{0,J};\ t^k = k\tau, k = 0,1,2,\dots\}$

и на этой сетке будем аппроксимировать данную дифференциальную задачу методом конечных разностей [2].

1.3. Метод конечных разностей

Основные определения, связанные с методом конечных разностей, рассмотрим на примере конечно-разностного решения первой начально-краевой задачи (с граничными условиями первого рода) для уравнения теплопроводности (1.1)-(1.4).

$$\frac{\partial u}{\partial t}=\alpha^2\frac{\partial^2 u}{\partial x^2}, 0 < x < l, t > 0 \qquad (1.1)$$

$$u(0,t) = \varphi_0(t), x = 0,\ t > 0, \qquad (1.2)$$

$$u(l,t) = \varphi_1(t), x = l,\ t > 0, \qquad (1.3)$$

$$u(x,0) = \psi(x),\ 0 \le x \le l, t = 0, \qquad (1.4)$$

Нанесем на пространственно-временную область $0 \le x \le l, 0 \le t \le T$ конечно-разностную сетку $\omega_{h\tau}$

$$\omega_{h\tau} = \{x_j = jh, j = \overline{0,N};\ t^k = k\tau, k = \overline{0,K}\} \qquad (1.5)$$

с пространственным шагом $h = l/N$ и шагом по времени $\tau = T/K$ (рис. 1.1)

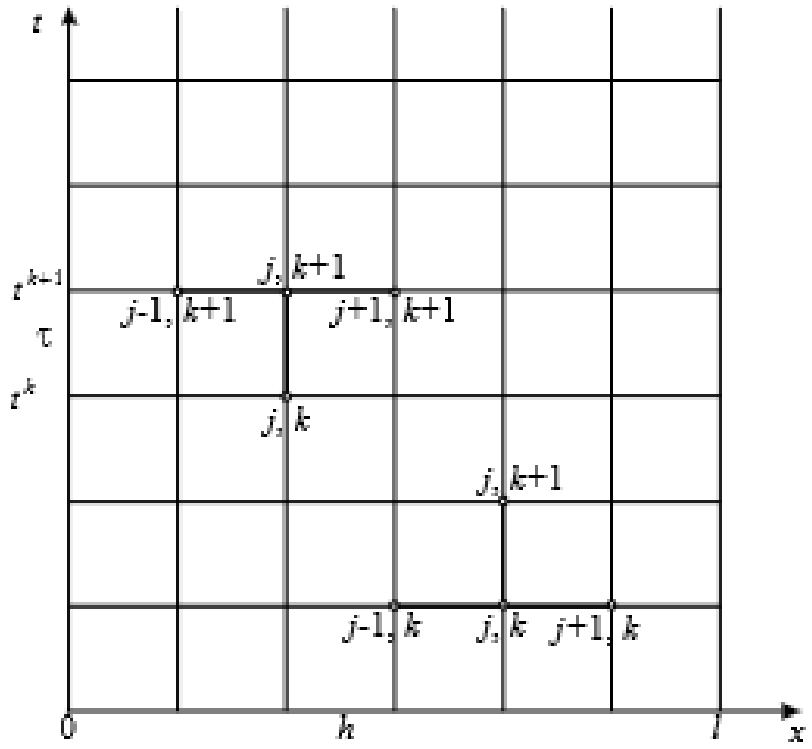


Рис. 1.1. Конечно - разностная сетка

Введем два временных слоя: нижний t^k=kτ , на котором распределение искомой функции $u(x_j, t^k)$, $j = \overline{0, N}$ известно (при k=0 распределение определяется начальным условием $u(x_j, t^0) = \psi(x_j)$) и верхний временной слой t^{k+1}=(k+1)τ , на котором распределение искомой функции $u(x_j, t^{k+1})$, $j = \overline{0, N}$ подлежит определению.

Сеточной функцией задачи (1.1)-(1.4) (обозначение u_j^k) назовем однозначное отображение целых аргументов j, k в значения функции $u_j^k = u(x_j, t^k)$

На введенной сетке (1.5) введем сеточные функции u_j^k, u_j^{k+1} , первая из которых известна, вторая – подлежит определению. Для ее определения в задаче (1.1)-(1.4) заменим (аппроксимируем) дифференциальные операторы отношением конечных, получим

$$\left.\frac{\partial u}{\partial t}\right|_j^k = \frac{u_j^{k+1} - u_j^k}{\tau} + O(\tau) \qquad (1.6)$$

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_j^k = \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + O(h^2) \qquad (1.7)$$

Подставляя (1.6), (1.7) в задачу (1.1)-(1.4), получим явную конечно-разностную схему для этой задачи в форме

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a^2 \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + O(\tau + h^2),\ j = \overline{1, N-1}\ , k = \overline{0, K-1}\ ,$$

$$u_0^k = \varphi_0(t^k),\ u_N^k = \varphi_1(t^k)\ , k = \overline{0, K};\ u_j^0 = \psi(x_j),\ j = \overline{0, N}, \qquad (1.8)$$

где для каждого j-го уравнения все значения сеточной функции известны, за исключением одного $-u_j^{k+1}$, которое может быть определено явно из соотношений (1.8).

В соотношения (1.8) краевые условия (*j =0; j= N*) входят при значениях *j=1* и *j=N-1*, а начальное условие – при *k=0*.

Если в (1.7) дифференциальный оператор по пространственной переменной аппроксимировать отношением конечных разностей на верхнем временном слое

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_j^{k+1} = \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{h^2} + O(h^2) \qquad (1.9)$$

то после подстановки (1.6), (1.9) в задачу (1.1)-(1.4), получим неявную конечноразностную схему для этой задачи

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a^2 \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{h^2} + O(\tau + h^2),\ j = \overline{1, N-1}\ , k = \overline{0, K-1}\ ,$$

$$u_0^{k+1} = \varphi_0(t^{k+1}),\ u_N^{k+1} = \varphi_1(t^{k+1})\ , k = \overline{0, K-1};\ u_j^0 = \psi(x_j),\ j = \overline{0, N} \qquad (1.10)$$

Теперь сеточную функцию u_j^{k+1} на верхнем временном слое можно получить из решения СЛАУ (1.10) с трехдиагональной матрицей. Эта СЛАУ в форме, пригодной для использования метода прогонки, имеет вид

$$a_1 = 0;\ c_{N-1} = 0; \begin{cases} b_1 u_1^{k+1} + c_1 u_2^{k+1} = d_1, j = 1, \\ a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j, j = \overline{2, N-2}, \\ a_{N-1} u_{N-2}^{k+1} + b_{N-1} u_{N-1}^{k+1} = d_{N-1}, j = N-1, \end{cases}$$

где $a_j = \sigma, j = \overline{2, N-1}$; $b_j = -(1+2\sigma), j = \overline{1, N-1}$; $c_j = \sigma, j = \overline{1, N-2}$;

$d_j = -u_j^k, j = \overline{2, N-2}$; $d_1 = -(u_1^k + \sigma\varphi_0(t^{k+1}))$;

$d_{N-1} = -(u_{N-1}^k + \sigma\varphi_1(t^{k+1}))$; $\sigma = \frac{a^2 r}{h^2}$.

1.4. Метод переменных направлений

В схеме метода переменных направлений (МПН), как и во всех методах расщепления, шаг по времени τ разбивается на число независимых пространственных переменных (в двумерном случае - на два). На каждом дробном временном слое один из пространственных дифференциальных операторов аппроксимируется неявно (по соответствующему координатному направлению осуществляются скалярные прогонки), а остальные явно. На следующем дробном шаге следующий по порядку дифференциальный оператор аппроксимируется неявно, а остальные – явно и т.д.

В двумерном случае схема метода переменных направлений для нашей задачи имеет вид:

Подсхема 1

$$\frac{u_{ij}^{k+1/2} - u_{ij}^k}{\tau/2} = \frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2} - 2u_{ij}^{k+1/2} + u_{i-1j}^{k+1/2}\right) + \frac{a}{h_2^2}\left(u_{ij+1}^k - 2u_{ij}^k + u_{ij-1}^k\right) + f_{ij}^{k+1/2}$$

Подсхема 2

$$\frac{u_{ij}^{k+1} - u_{ij}^{k+\frac{1}{2}}}{\tau/2} = \frac{a}{h_1^2}\left(u_{i+1j}^{k+1/2} - 2u_{ij}^{k+\frac{1}{2}} + u_{i-1j}^{k+\frac{1}{2}}\right) + \frac{a}{h_2^2}\left(u_{ij+1}^{k+1} - 2u_{ij}^{k+1} + u_{ij-1}^{k+1}\right) + f_{ij}^{k+\frac{1}{2}}$$

$$i = 1..I-1;\ j = 1..J-1;\ k = 0..N-1$$

Разностная постановка краевых условий

$$u_{i0}^k = \varphi_1(x_i, t^k); i = 0..I; j = 0; k = 0..N$$

$$u_{iJ}^{k} = \varphi_2(x_i, t^k)\ ; i = 0..I; j = J; k = 0..N$$

$$u_{0j}^{k} = \varphi_3\left(y_j, t^k\right); j = 0..J; i = 0; k = 0..N$$

$$u_{Ij}^{k} = \varphi_4(y_j, t^k); j = 0..J; i = I; k = 0..N$$

Начального условия

$$u_{ij}^{0} = \psi\left(x_i, y_j\right); i = 0..I; j = 0..J; k = 0$$

В подсхеме 1 на первом дробном шаге τ/2 оператор $a\frac{\partial^2}{\partial x^2}$ аппроксимируется неявно, а оператор $a\frac{\partial^2}{\partial y^2}$ явно (в результате весь конечно-разностный оператор по переменной *y* переходит в правые части, поскольку u_{ij}^{k} известно). С помощью скалярных прогонок в количестве, равном числу *J-1*, в направлении переменной *x* получаем распределение сеточной функции $u_{ij}^{k+1/2}$, $i = \overline{1,\ldots,I-1}$; $j = \overline{1,\ldots,J-1}$ на первом временном полуслое $t^{k+1/2} = t^k + \tau/2$.

В подсхеме 2 оператор $a\frac{\partial^2}{\partial y^2}$ аппроксимируется неявно на верхнем временном слое t^{k+1}=*(k+1)* τ , а оператор $a\frac{\partial^2}{\partial x^2}$ − явно в момент времени $t^{k+1/2} = t^k + \tau/2$ (конечно-разностный аналог этого оператора переходит в правые части). С помощью скалярных прогонок в направлении переменной y в количестве, равном числу *I−1* получаем распределение сеточной функции u_{ij}^{k+1}, $i = \overline{1,\ldots,I-1}$; $j = \overline{1,\ldots,J-1}$ на втором полуслое $t^{k+1} = t^{k+1/2} + \tau/2$. Шаблон схемы МПН представлен на рис.1.2.

Можно показать, что в двумерном случае схема МПН абсолютна устойчива. К достоинствам метода переменных направлений можно отнести высокую точность, поскольку метод имеет второй порядок точности по времени. К недостаткам можно отнести условную устойчивость при числе пространственных переменных больше двух. Кроме этого, МПН условно

устойчив в задачах со смешанными производными уже в двумерном случае [2].

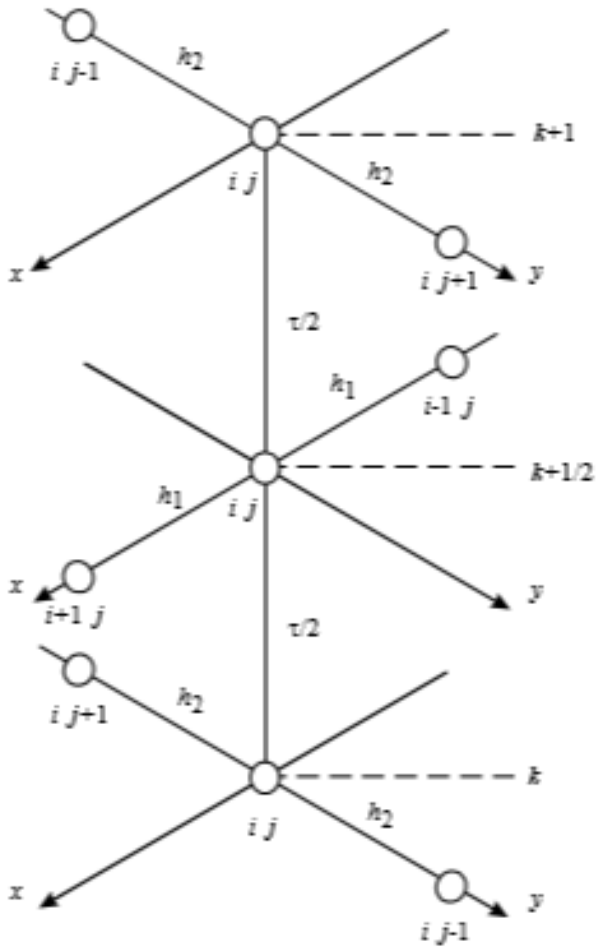


Рис. 1.2. Шаблон схемы метода переменных направлений

Уравнение Подсхемы 1 можно переписать в виде:

$$A_i u_{i-1j}^{k+1/2} + B_i u_{ij}^{k+1/2} + C_i u_{i+1j}^{k+1/2} = F_{ij}^{k}$$

$$i = 1..I-1,\ \ j = 0..J, k = 0..N-1$$

$$A_i = -\frac{\tau a}{2h^2{}_1}; \qquad B_i = 1 + \frac{\tau a}{h^2{}_1}; \qquad C_i = -\frac{\tau a}{2h^2{}_1}$$

$$F_{ij}^{k} = A_j u_{ij-1}^{k} + B_j u_{ij}^{k} + C_j u_{ij+1}^{k} + \frac{\tau}{2} f_{ij}^{k+1/2}$$

СЛАУ имеет трехдиагональный вид и может быть решена методом прогонки по оси x.

Уравнение Подсхемы 2 можно переписать в виде:

$$A_j u_{ij-1}^{k+1} + B_j u_{ij}^{k+1} + C_j u_{ij+1}^{k+1} = F_{ij}^{k+1/2}$$

$$j = 1..J-1, \;\; i = 0..I, k = 0..N-1$$

$$A_j = \frac{\tau a}{2h^2{}_2}; \qquad B_j = 1 - \frac{\tau a}{h^2{}_2}; \qquad C_j = \frac{\tau a}{2h^2{}_2}$$

$$F_{ij}^{k+1/2} = A_i u_{i-1j}^{k+1/2} + B_i u_{ij}^{k+1/2} + C_i u_{i+1j}^{k+1/2} + \frac{\tau}{2} f_{ij}^{k+1/2}$$

СЛАУ имеет трехдиагональный вид и может быть решена методом прогонки по оси y.

Краевые условия можно переписать в виде:

$$u_{0j} = \varphi_3(y_j), \qquad j = 0..J$$

$$u_{Ij} = \varphi_4(y_j), \qquad j = 0..J$$

$$u_{i0} = \varphi_1(x_i), \qquad i = 0..I$$

$$u_{iJ} = \varphi_2(x_i), \qquad i = 0..I$$

Глава 2. Описание программной реализации

2.1. Средства разработки

Программа была реализована на языке C#. В выпускной квалификационной работе использован именно он, так как у него есть ряд преимуществ:

- простая языковая база, из которой вынесены в библиотеки многие существенные возможности, вроде математических функций или функций управления файлами;
- ориентация на процедурное программирование, обеспечивающее удобство применения структурного стиля программирования;
- система типов, предохраняющих от бессмысленных операций;
- непосредственный доступ к памяти компьютера через использование указателей;
- минимальное число ключевых слов;
- передача параметров в функцию по значению, а не по ссылке (при этом передача по ссылке эмулируется с помощью указателей);
- указатели на функции и статические переменные;
- области действия имён;
- структуры и объединения — определяемые пользователем собирательные типы данных, которыми можно манипулировать как одним целым [6].

Для визуализации решений использовалась графическая библиотека OpenGL.

OpenGL (Open Graphics Library — открытая графическая библиотека, графический API) — спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

В выпускной квалификационной работе использована именно эта библиотека, так как у неё есть ряд преимуществ:

- Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов.
- Реализация библиотеки призвана эффективно использовать возможности оборудования.
- Скрывает сложности адаптации различных 3D-ускорителей, предоставляя разработчику единый API.
- Скрывает различия в возможностях аппаратных платформ, требуя реализации недостающей функциональности с помощью программной эмуляции.
- Стандартизирует доступ к графической аппаратуре путём смещения ответственности за создание аппаратного драйвера на производителя графического устройства [9].

Для реализации некоторых поставленных задач (сохранение в файл, работа с пикселями изображения) потребовалось прибегнуть к использованию GDI+.

GDI+ - это набор программных средств, которые используются в .NET. GDI+ позволяют создателям приложений выводить данные на экран или на принтер без необходимости обеспечивать работу с определенными типами устройств отображения. Для отображения информации программисту достаточно вызывать нужные методы классов GDI+. При этом автоматически учитываются типы определенных устройств и выполняются вызовы соответствующих драйверов [14].

2.2. Считывание введенных выражений и контроль ввода входных данных

Для вычисления значений выражений, заданных пользователем в различных полях, был создан класс **Calculator**. Значения берутся из полей в виде строк, поэтому невозможно без дополнительных действий рассчитать значение выражения.

Метод Parse класса Calculator получает на вход математическое выражение в виде строки, обрабатывает ее, вычисляет значение и возвращает численный результат. Класс Calculator умеет работать с целыми и вещественными числами, с экспонентой, с числом π, с операциями +, -, *, /, ^, а также их приоритетом, со скобками, с математическими функциями sin, cos и не чувствителен к пробелам.

Самым высоким приоритетом разбора является число (целое, вещественное, экспонента, число π и математические функции sin, cos), потом идут скобки, степень, умножение и деление, сложение и вычитание. Эти приоритеты соответствуют методам number, ckobki, pow, factor,expr.

Класс реализован так, что сначала вызывается метод, с наименьшим приоритетом разбора математического выражения. В начале каждого метода (кроме самого приоритетного) вызывается более приоритетный. Тем самым, вызовы методов доходят до самого приоритетного, и происходит исполнение каждого из них в порядке от самого приоритетного к менее приоритетному. После получения значения и “обратного хода” к самому низко приоритетному методу, будут выполнены математические операции +, - и еще раз вызван метод factor. Данные действия будут продолжаться, пока рассматриваемая строка не закончится.

В этом классе реализованы следующие методы:

- public double Parse(string s) – Этот метод является единственно доступным для вызова из других классов. Чтобы его вызвать, в классе MainForm создается экземпляр класса Calculator, вызывается метод

Parse и передается строка s, которая является математическим выражением. В данном методе, для более удобного разбора выражения, удаляются все пробелы между символами, а полученная строка передается в метод expr. Также, со строкой передается переменная buf, которая соответствуют текущему номеру символа, который подвергается анализу.

- double expr(string s, ref int buf) – В этом методе сначала проверяется, относится ли символ из строки s под номером buf к тому, у которого приоритет выше, чем у операций + и -. Для этого, в начале метода вызывается метод factor. А по его завершению будет проверка на операции + и -.

- double factor(string s, ref int buf) – В этом методе сначала проверяется, относится ли символ из строки s под номером buf к тому, у которого приоритет выше, чем у операций * и /. Для этого, в начале метода вызывается метод pow. А по его завершению будет проверка на операции * и /.

- double pow(string s, ref int buf) – В этом методе сначала проверяется, относится ли символ из строки s под номером buf к тому, у которого приоритет выше, чем у операций ^ (возведение в степень). Для этого, в начале метода вызывается метод ckobki. А по его завершению будет проверка на операцию ^.

- double ckobki(string s, ref int buf) – В этом методе проверяется, является ли текущий символ открывающейся скобкой. Если является, то выражение в скобках обрабатывается отдельно (Вызывается метод expr). В противном случае, вызывается метод number с наивысшим приоритетом.

- double number(string s, ref int buf) - В этом методе проверяется, каким числом является текущий символ. Если это экспонента или число π, то сразу же возвращается результат. Целые и вещественные числа состоят как правило из более чем одного символа, поэтому происходит сбор этого числа по одному символу, пока оно не закончится. Если же это математическая функция sin или cos, то вызывается метод ckobki, в котором вычисляется значение в скобках, а потом метод number возвращается sin или cos полученного числа.

2.3. Численное решение уравнения теплопроводности

В классе **MainForm** были реализованы методы, подготавливающие данные для последующего их использования при расчете численного решения уравнения теплопроводности:

- double ALLfi_psi_Fij_solution(string function, double X, double Y, double T) – В этом методе рассчитывается значение указанной функции в заданной точке. В зависимости от нужной функции, из соответствующего поля производится считывание значения, которое передается в виде строки в метод Parse класса Calculator. На выходе возвращается численное значение указанной функции в заданной точке.

- void Prepare(string num) - В этом методе, в зависимости от выбранного варианта краевой задачи, заполняются поля условий краевой задачи. Предусмотрено два варианта краевых задач. При выборе третьего номера, пользователю предоставляются пустые поля условий краевой задачи для самостоятельного заполнения.

- void Init() - В этом методе рассчитывается пространственно-временная сетка по переменным x,y,t, а также значения на границах.

- void Calculate – В этом методе происходит вызов методов ReadValue(), Init(), MPN(), в которых считываются введенные пользователем значения, инициализируются переменными и решается задача методом переменных направлений. Далее, если пользователь ввел точное решение, то рассчитывается точное решение, нормы матриц в методах FindUT() и CalcNorma() соответственно. Появляется вкладка “Погрешность”. Если точное решение отсутствуют, то оно и нормы матриц не рассчитываются, а вышеупомянутая вкладка исчезает.

Также, в этом классе были реализованы методы вывода решения в таблицу.

- void Compute () – В этом методе полученное приближенное решение выводится в соответствующее поле, а также вызывается метод, визуализирующий данное решение.

- void exact () – В этом методе полученное точное решение выводится в соответствующее поле, а также вызывается метод, визуализирующий данное решение.

- void delta () – В этом методе рассчитывается разница между точным и приближенным решением. Полученное значение выводится в соответствующее поле.

Расчет приближенного, точного (в случае его наличия) численного решения и расчет невязок (в случае наличия точного решения) происходит в классе Computing с помощью следующих методов:

- void MPN() – В этом методе производится поиск приближенного решения задачи методом переменных направлений.

- void FindUT() – В этом методе производится поиск точного решения.

- void CalcNorma() – В этом методе производится расчет норм матриц. В частности, рассчитываются нормы $||\Delta u||_1$, $||\Delta u||_2$, $||\Delta u||_\infty$ по полученным точному и приближенному решениям, где $\Delta u = u^T - u$ (разница между точным и приближенным решением) далее будет обозначаться как Δu. Полученные нормы выводятся в соответствующие поля.

2.4. Визуализация численного решения уравнения теплопроводности

2.4.1. Графическое 3D представление

Для отображения визуализированного решения в виде 3D графика был создан класс Draw3DGraphics и реализованы следующие методы:

- void InitOpenGL() – В этом методе производится инициализация графической библиотеки OpenGL.

- void Draw(double[] X,double[] Y, double[,,] U) - В этом методе, сначала, настраивается положение камеры для лучшего обзора, визуализируется полученное решение в узлах сетки. Узлы между собой соединены линиями в виде квадратов. Цвет внутри квадрата равномерно переходит один в другой. Окраска графика меняется от синего цвета к красному. Синему цвету на графике соответствует наименьшее значение температуры, а красному - наибольшее. Производится отрисовка координатных осей и подпись значений на этих осях.

Для настройки отображения графика реализован класс **Form_View**: пользователь имеет возможность вручную указать значения перемещения и поворота графика вокруг осей.

В этом классе реализованы следующие методы:

- private void txtbox_movegraf_TextChanged(object sender, EventArgs e) – В этом методе, при изменении значений в textbox, которые отвечают за перемещение графика вдоль осей x,y,u рисуется график с заданными значениями. Значения в соответствующих trackBar тоже изменяются. При указании в textbox значений, которые больше максимального или меньше минимального значения в trackBar, ползунок устанавливается на максимум или минимум соответственно. Если введены некорректные значения в textbox, то он подсвечивается красным цветом и график не рисуется. При изменении значений на корректные, подсветка пропадает и рисуется график.

- private void trackBarMove_Scroll(object sender, EventArgs e) – В этом методе, при изменении положений ползунков, отвечающих за перемещение графика по осям x,y,u, их значения записываются в соответствующие textbox. Визуализируется решение краевой задачи.

- private void trackBarRotate_Scroll(object sender, EventArgs e) – В этом методе, при изменении положений ползунков, отвечающих за поворот графика вокруг осей x,y,u на определенный угол, их значения выводятся в соответствующие label. При корректных значениях в textbox, отвечающих за перемещение графика, визуализируется решение краевой задачи. В противном случае, вывод графика не происходит.

Класс **MyNewColor** является вспомогательным и предназначен для вычисления цвета, в который будет покрашена точка в методе Draw() класса Draw3DGraphics при покраске графика. Выбор цвета происходит из спектра цветов радуги: чем больше значение, тем он ближе к красному.

В классе MyNewColor реализованы следующие методы:

- public MyNewColor(double MIN,double MAX) – конструктор класса, в который передаются максимальное и минимальное значение графика функции.
- Color FindColor(int k) – определяется к какому цвету стоит отнести данную точку.
- public Color GetNewColor(double u) – все поле значений функции разделяется на равные части. Определяется, в какой промежуток попадает переданное в функцию значение. Оно сопоставляется с цветовым промежутком и на основании этого точке присваивается цвет путем передачи в метод FindColor(int k).

2.4.2. Графическое 2D представление. Алгоритм билинейной интерполяции

Численное решение представляет собой матрицу значений u_{ij}^{k} в узлах сетки $i = 0..I;\ j = 0..J;$ на различных временных слоях $k = 0..N$. Чтобы представить решение в промежуточных точках расчетной сетки, требуется использовать алгоритм билинейной интерполяции.

Интерполяция - это способ вычислить промежуточное значение функции по нескольким уже известным ее значениям.

Двойная линейная интерполяция (билинейная интерполяция) - линейная интерполяция функции двух переменных, то есть интерполяция по четырем точкам, при известных значениях функции в этих точках.

На рис. 2.1. в четырёх красных точках с координатами x_1y_1, x_1y_2, x_2y_1, x_2y_2 заданы значения исходной функции. Требуется получить приближённое (интерполированное) значение исходной функции в зелёной точке с координатами x,y должно быть интерполировано.

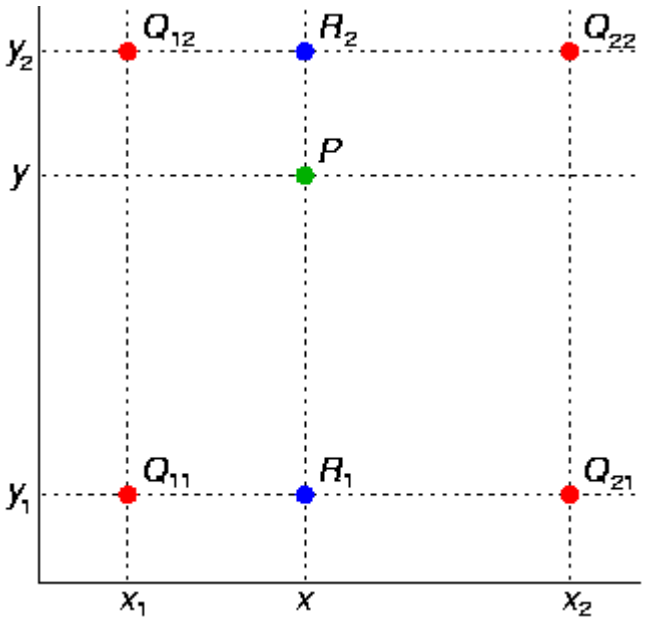


Рис. 2.1. Постановка задачи

Допустим, что необходимо интерполировать значение функции $f(x,y)$ в точке $P=(x,y)$. Значения функции в окружающих точку P точках $Q_{11}=(x_1,y_1)$, $Q_{12}=(x_1,y_2)$, $Q_{21}=(x_2,y_1)$, $Q_{22}=(x_2,y_2)$ известны (рис. 1).

Первым шагом линейно интерполируется значение вспомогательных точек R_1 и R_2 вдоль оси абсцисс, где $R_1=(x,y_1)$, $R_2=(x,y_2)$.

$$f(R_1)=\frac{x_2-x}{x_2-x_1}f(Q_{11})+\frac{x-x_1}{x_2-x_1}f(Q_{21})$$

$$f(R_2)=\frac{x_2-x}{x_2-x_1}f(Q_{12})+\frac{x-x_1}{x_2-x_1}f(Q_{22})$$

Теперь проводится линейная интерполяция между вспомогательными точками R_1 и R_2.

$$f(P)=\frac{y_2-y}{y_2-y_1}f(R_1)+\frac{y-y_1}{y_2-y_1}f(R_2)$$

Это и есть интерполируемое значение функции $f(x,y)$.

Реализация отображения цветного графика

Подготовка области отображения.

Перед началом отрисовки, требуется произвести подготовку отображаемой области и OpenGL: очистить область отображения, буфер цвета, текущую матрицу. Затем, среди всех значений температуры на текущем временном слое, ищется минимальное и максимальное значения для определения границ цветов, где минимальное соответствует синему, а максимальное – красному цвету.

```
// очистка окна
Gl.glClearColor(255, 255, 255, 1);
// очищаем буфер цвета
Gl.glClear(Gl.GL_COLOR_BUFFER_BIT);
// очищаем текущую матрицу
Gl.glLoadIdentity();

double minValue = U[0, 0, 0];
double maxValue = U[0, 0, 0];

for (int i = 0; i <= N; i++)
{
    for (int j = 0; j <= N1; j++)
    {
        for (int jj = 0; jj <= M; jj++)
        {
            if (U[i, j, jj] > maxValue) maxValue = U[i, j, jj];
            if (U[i, j, jj] < minValue) minValue = U[i, j, jj];
        }
    }
}
```

Расчет значений точек внутри расчетной сетки методом билинейной интерполяции.

Решение двумерного уравнения теплопроводности в пластине представляет собой сетку, где для значений x_i и y_j соответствует значение $u(x,y,t)$ (Рис.2.2.). Для цветового представления решения, требуется каждый квадрат из сетки (границы квадратов обозначены черным цветом) разбить на определенное количество точек (обозначены зеленым цветом), и в каждой из них вычислить значение функции с помощью билинейной интерполяции. Обход всех точек осуществляется с помощью четырех циклов: два цикла по узлам сетки и столько же по разбитым точкам (Приложение 1).

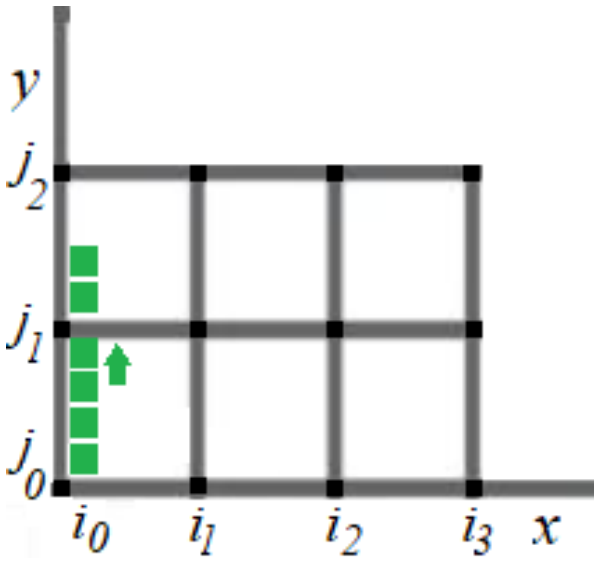


Рис.2.2. Сетка решения уравнения теплопроводности

Способ реализации цветовой схемы.

Теперь, когда значения в точках определены, нужно их покрасить в цвета соответствующие их значениям температуры. Для этого, интервал значений температуры делится на равные части (количество цветов), каждой из которых ставится в соответствие цвет от синего до красного (Приложение 2). Чтобы пользователю было понятно соответствие значений цвета и температуры, справа от графика располагается цветовая шкала. Для ее построения, требуется отобразить все интервалы цветов и соответствующие им значения.

```
//Отображение шкалы цветов
 Gl.glBegin(Gl.GL_QUAD_STRIP);
 Gl.glPointSize(10);
 int kk = 0;
 foreach (var v in mnc.ColorScale)
 {
     Gl.glColor4ub(v.Key.R, v.Key.G, v.Key.B, v.Key.Alpha);
     Gl.glVertex2d(scaleX, scaleY + kk);
     Gl.glVertex2d(scaleX + 1, scaleY + kk);
     Gl.glVertex2d(scaleX, scaleY + (kk + 1));
     Gl.glVertex2d(scaleX + 1, scaleY + (kk + 1));
     kk++;
 }
Gl.glEnd();
```

Построение изолиний

Обычно, в литературе, под изолиниями имеется в виду линии одинаковой температуры. В рамках данной работы, имеется диапазон значений, внутри которого температура принимается как одинаковая.

Для получения графика изолиний, надо по пиксельно пройти предыдущий график и выяснить, какие пиксели надо окрашивать черным, а какие – белым. Черные пиксели будут соответствовать изолиниям.

Для реализации данной подзадачи требуется

- Представить изображение на компоненте Bitmap.
- Отдельно сохранить пиксели и соответствующие им цвета в словарь.
- Отметить пиксели, через которые проходит изолиния.
- Отобразить отмеченные пиксели на форме, чтобы получился график изолиний на указанном временном слое.

Для того, чтобы скопировать изображение, нарисованное с помощью OpenGL из компонента SimpleOpenGLControl на компонент Bitmap, потребуется прибегнуть к помощи следующих функций: bitmap.LockBits, Gl.glReadBuffer, Gl.glReadPixels, bitmap.UnlockBits.

bitmap.LockBits(Rectangle r , ImageLockMode ilm, PixelFormat pf) - Производит блокировку битов растрового изображения. Этот метод нужен, чтобы существующий компонент Bitmap можно было отредактировать программно. Атрибут Rectangle определяет прямоугольную область компонента для блокировки битов. ImageLockMode предстваляет собой уровень доступа (чтение/запись) для компонента Bitmap, PixelFormat определяет формат данных для компонента Bitmap.

Gl.glReadBuffer(GLenum mode); - позволяет выбрать источник цветного буфера для последующего использования в Gl.glReadPixels. Атрибут mode задает цветной буфер.

Gl.glReadPixels(GLint x, GLint y, GLsizei width, GLsizei height, GLenum format, GLenum type, GLvoid *pixels) - Считывает пиксельные данные из прямоугольника буфера кадра, где x, y - левый нижний угол буффера, width и height - ширина и высота прямоугольника соответственно, pixels – массив, в которой метод сохраняет данные пикселей, аргумент format задает характер элементов пиксельных данных, которые должны быть считаны (величина индекса или величины R, G, B или A), а type задает тип данных для каждого элемента

bitmap.UnlockBits(BitmapData bd) - Производит разблокирование битов компонента Bitmap. BitmapData указывает атрибуты Bitmap, такие как размер, формат пикселей, начальный адрес пиксельных данных в памяти и длина каждой строки сканирования (шаг).

Для сохранения пикселей и соответствующих им цветов в словарь, требуется последовательно, в цикле, пройти по всем пикселям компонента Bitmap. Если цвет текущего пикселя не черный или белый, то данный пиксель относится к цветному графику, значит, координаты и значение цвета этого пикселя требуется записать в словарь. Получение цвета пикселя осуществляется с помощью метода Color Bitmap.GetPixel (Int x, Int y), где x и y – это координаты пикселя, а color – его цвет.

Для обнаружения пикселей, через которые проходит изолиния, был реализован следующий алгоритм: последовательно проходятся все пиксели из словаря и они окрашиваются черным цветом только при смене цвета, в противном случае – пиксель окрашивается белым цветом. На рис.2.3. различные цвета обозначаются цифрами 1 и 2. В цикле проходятся все значения, и если цвет текущей точки отличается от точек слева или сверху,

то данную точку окрашиваем черным. Если у текущей точки нет соседа слева или сверху, то она сравнивается только с существующей точкой. В результате обхода всех пикселей, также получается словарь, но в отличии от предыдущего, этот отображает только изолинии.

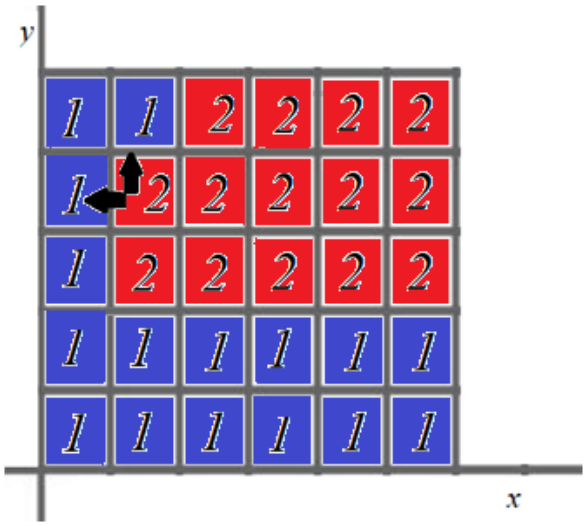


Рис. 2.3. Сетка решения уравнения теплопроводности для изображения методов обхода точек.

Для отображения пикселей и их значений на форме, требуется создать объект Bitmap на форме, и, в цикле по всем значениям словаря с изолиниями, окрасить пиксели компонента Bitmap с помощью метода bitmap.SetPixel(x,y,color), где x и y – значения пикселя, а color - его цвет.

Подпись значения температуры на изолинии

Для завершения представления графика изолиний, требуется подписать текущую температуру рядом с каждой изолинией. Для этого надо в существующем словаре, состоящем из всех точек всех изолиний, определить принадлежность каждой точки к определенной изолинии и потом подписать температуру в начальной точке каждой изолинии.

Чтобы найти все изолинии, необходимо реализовать следующий рекурсивный алгоритм: Взять первую точку из словаря и проверить ее соседей со всех сторон: сверху, снизу, слева, справа и по 4 диагоналям. Если

у точки есть, хотя бы один из соседей, то записываем эту точку в массив, и потом используем уже ее, для поиска соседей. Алгоритм заканчивается, когда у точки не оказывается соседей, или соседом является уже записанная точка. Далее, просто подписываем температуру на начальной точке средствами OpenGL.

Подпись изолиний

Для того, чтобы подписать значения на всех изолиниях необходимо циклом пройти график изолиний и разделить по массивам каждую отдельную изолинию. Имея массив всех изолиний, поставим значения изолиний рядом с первым пикселем изолинии. Обход значений происходит снизу вниз и слева направо.

Рекурсивный алгоритм определения изолинии

Имея словарь Dictionary<Point, Color> всех точек изображения и их цветов, можно разделить все имеющиеся изолинии следующим способом: заводим словарь из точек и цветов (Dictionary<Point, Color>), куда будем записывать точки текущей изолинии. Чтобы представить все изолинии в удобном виде, будем каждый словарь записывать в список, который будет содержать все изолинии (List<Dictionary<Point, Color>>). Потом обходим все точки словаря в цикле и проверяем, является ли текущая точка изолинией (окрашена ли черным цветом) и относится ли данная точка к какой-либо изолинии (содержится в List<Dictionary<Point, Color>>). Если точка черная и ее в массиве еще не было, то вызываем метод, который рекурсивно проверяет все соседние точки: сверху, снизу, слева, справа, по всем диагоналям. Если хотя бы одна из этих точек имеет черный цвет и не была записана в словарь Dictionary<Point, Color>, то мы записываем ее в словарь и рекурсивно вызываем этот же метод, но теперь уже в качестве параметра будет передана найденная точка и у нее будут проверяться все соседи. В случае, когда у точки отсутствует сосед (выход за границы изображения), или все соседи

точки уже были записаны, алгоритм заканчивается и возвращается в цикл обхода по всем точкам изображения. Аналогичным образом проходятся все пиксели и находятся оставшиеся изолинии.

Наглядно данный рекурсивный алгоритм можно представить в виде блок-схемы на рис. 2.4.:

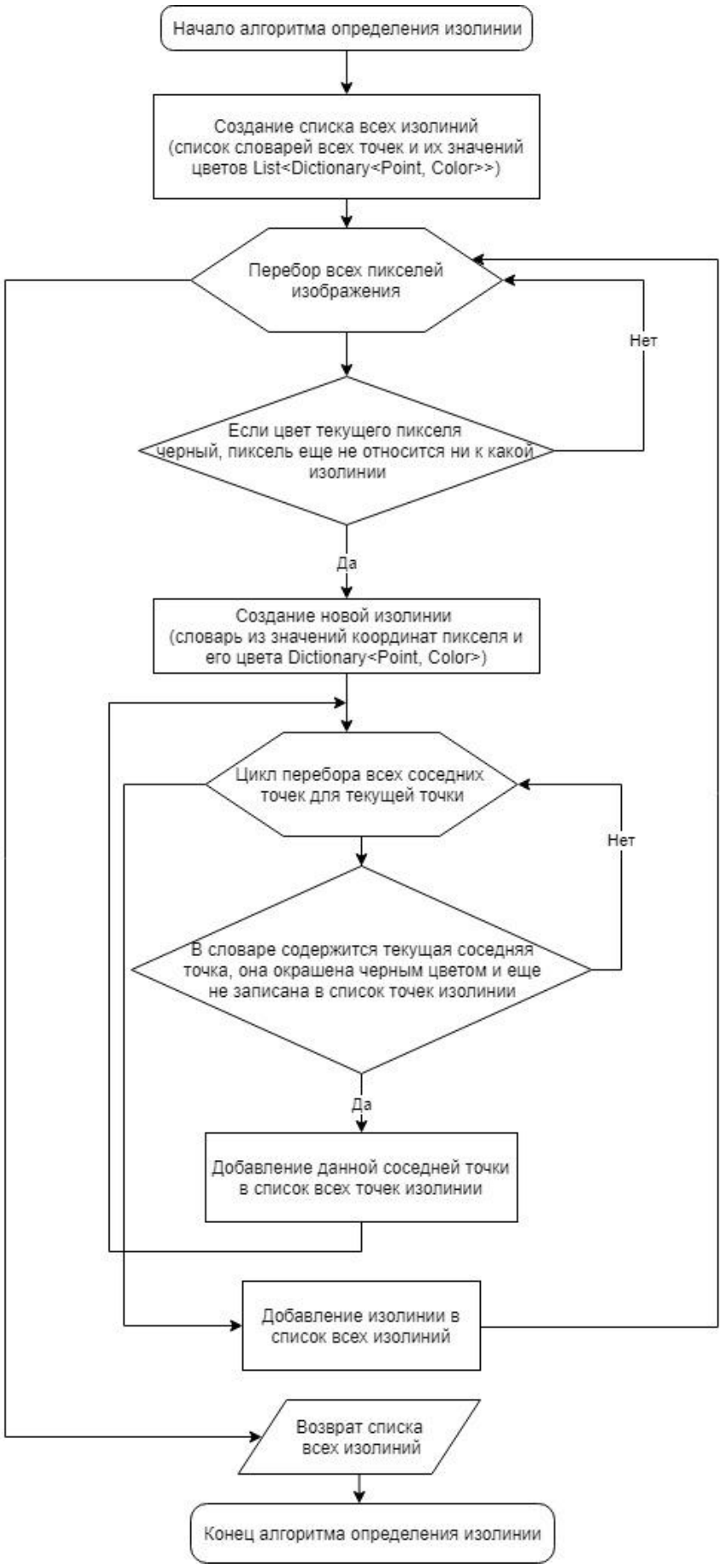


Рис. 2.4. Алгоритм определения изолиний

Для отображений значений на изолиниях, берется первая точка каждой изолинии и рядом с ней подписывается соответствующее значение. Значение определяется следующим образом: имея все значения температуры на всех временных слоях рассчитывается минимальное и максимальное значения. Затем, этот промежуток равномерно делится на 7 цветов (в соответствии с цветовой схемой), пограничное значение при переходе одного цвета в другой и отображается рядом с первым пикселем изолинии.

Масштабирование графиков

При изменении интервала по осям *x* или *y*, к графикам автоматически будет применяться масштабирование т.е. место, занимаемое графиками, останется прежним, размер графика будет автоматически уменьшен (увеличен) при увеличении (уменьшении) значений интервалов по осям *x* или *y*. Реализовано это с помощью введения множителя, который уменьшает расстояние между точками, если интервал по какой-либо из осей увеличился, или увеличивает расстояние, если заданный интервал уменьшился.

```
multX = otstup / b;
multY = otstup / b1;
Gl.glVertex2d(otstupX + x * multX, otstupY + y * multY);
```

где b и b1 – границы интервалов x и y соответственно

2.5. Оптимизация приложений .NET с использованием параллельного программирования

Долгие годы вычислительные мощности компьютерных систем увеличивались экспоненциально. Скорость процессоров росла с каждой новой моделью, и программы, прежде создававшиеся в расчете на высокопроизводительные и дорогостоящие рабочие станции, стали переноситься на ноутбуки и планшетные устройства. Эта эра закончилась несколько лет тому назад, и быстродействие современных процессоров увеличивается уже не экспоненциально; зато экспоненциально стало увеличиваться их количество. Создавать программы, использующие

преимущества многопроцессорных архитектур было непростым делом, когда такие системы были редкими и дорогостоящими, но оно не стало простым делом и сейчас, когда смартфоны снабжаются двух и четырехъядерными процессорами.

Перспективы и преимущества

Одной из захватывающих перспектив, с точки зрения параллельного программирования, является все возрастающее разнообразие многопроцессорных систем. Средние современные рабочие станции или мощные ноутбуки часто снабжаются мощными графическими процессорами (GPU), обладающими возможностью обслуживать сотни потоков выполнения.

Рост производительности, получаемый за счет применения приемов параллельного программирования, достаточно велик, чтобы от него легко было отказаться. Приложения, в основном занимающиеся операциями ввода/вывода, могут получить огромные преимущества за счет переноса этих операций в отдельные потоки, и выполнения их асинхронно и параллельно, благодаря чему обеспечивается более высокая отзывчивость и масштабируемость.

Преимущественно вычислительные приложения, реализующие параллельные алгоритмы, способны увеличивать свою производительность на порядок, за счет использования всех доступных процессоров, или на два порядка, за счет использования ядер графического процессора.

Новейшие параллельные фреймворки, включая библиотеки Task Parallel Library (.NET 4.0) которые будут использоваться, стремятся уменьшить сложность разработки параллельных приложений и дать возможность извлечь максимальную выгоду в виде высокой производительности [9].

С помощью различных библиотек можно реализовать параллелизм потоков, задач и данных. В рамках данной работы используется только параллелизм данных, т.к. отсутствует множество однотипных задач, которое требовалось бы решать и которые можно было бы решать параллельно. А те задачи, которые необходимо решить, возможно реализовать только последовательно.

Параллелизм данных

Основной целью парадигмы параллелизма данных является полное устранение задач из поля зрения и их замена высокоуровневой абстракцией - параллельными циклами. Иными словами, распараллеливается не реализация алгоритма, а данные, которыми она оперирует. Библиотека Task Parallel Library предлагает несколько вариантов поддержки параллелизма данных.

Циклы for и foreach часто являются отличными кандидатами для распараллеливания. Библиотека Task Parallel Library предоставляет поддержку распараллеливания циклов посредством явных методов, очень близких своим языковым эквивалентам. Речь идет о методах Parallel.For() и Parallel.ForEach(), максимально близко имитирующих поведение циклов for и foreach. При замене языковой инструкции цикла for вызовом метода Parallel.For(), автоматически обеспечивается параллельное выполнение итераций цикла. Кроме того, метод Parallel.For - это не простой цикл, генерирующий задачи в каждой итерации или для каждого фрагмента данных определенного размера. Вместо этого Parallel.For не спеша приспосабливается к темпу выполнения отдельных итераций, учитывая количество задач, выполняющихся в каждый момент, и исключает вероятность дробления диапазона на слишком мелкие фрагменты, производя его деление динамически [10].

2.5.1. Распараллеливание циклов метода переменных направлений с помощью метода Parallel.For

Части кода в методе переменных направлений, которые поддаются распараллеливанию – это прогонки по осям *x* и *y*. В МПН только эти части кода можно распараллелить, т.к. только эти циклы являются независимыми частями метода переменных направлений. Распараллеливание происходит с помощью Parallel.For(int fromInclusive, int toExclusive, Action<int, ParallelLoopState> body). Он обеспечивает возможность параллельного выполнения итераций цикла, где fromInclusive – начальное, а toExclusive – конечное значение итерационной переменной, body – делегат функции, который вызывается один раз за итерацию.

Ниже представлен пример кода распараллеливания метода трехдиагональной прогонки по осям *x* и *y*:

```
// прогонка по X
Parallel.For(1, N1, j =>
                {
                    double[,] A = new double[N + 1, N + 1];
                    double[] B = new double[N + 1];
                    for (int i1 = 0; i1 <= N; i1++)
                    {
                        B[i1] = r1 * U[i1, j + 1, k] + 2 * (1 - r1) * U[i1,
j, k] + r1 * U[i1, j - 1, k] + tau * ALLfi_psi_Fij_solution("Fij", X[i1],
Y[j], T[k]);
                        for (int j1 = 0; j1 <= N; j1++)
                        {
                            if (i1 == j1) A[i1, j1] = 2 * (1 + r);
                            else if ((j1 == i1 - 1) || (j1 == i1 + 1)) A[i1,
j1] = -r;
                            else A[i1, j1] = 0;
                        }
                    }
                    A[0, 0] = 1;
                    A[0, 1] = 0;
                    A[N, N - 1] = 0;
                    A[N, N] = 1;
                    B[0] = ALLfi_psi_Fij_solution("fi3", double.NaN, Y[j],
T[k]);
                    B[N] = ALLfi_psi_Fij_solution("fi4", double.NaN, Y[j],
T[k]);
                    double[] un;
                    Matrix.SLE_Thomas(A, B, out un);

                    for (int i = 0; i <= N; i++) U1[i, j, k] = un[i];
```

```
                });
// прогонка по Y
Parallel.For(1, N, i =>
                {
                    double[,] A1 = new double[N1 + 1, N1 + 1];
                    double[] B1 = new double[N1 + 1];
                    for (int i1 = 0; i1 <= N1; i1++)
                    {
                        B1[i1] = r * U1[i + 1, i1, k] + 2 * (1 - r) * U1[i,
i1, k] + r * U1[i - 1, i1, k] + tau * ALLfi_psi_Fij_solution("Fij", X[i],
Y[i1], T[k]);//??
                        for (int j1 = 0; j1 <= N1; j1++)
                        {
                            if (i1 == j1) A1[i1, j1] = 2 * (1 + r1);
                            else if ((j1 == i1 - 1) || (j1 == i1 + 1)) A1[i1,
j1] = -r1;
                            else A1[i1, j1] = 0;
                        }
                    }
                    A1[0, 0] = 1;
                    A1[0, 1] = 0;
                    A1[N1, N1 - 1] = 0;
                    A1[N1, N1] = 1;
                    B1[0] = ALLfi_psi_Fij_solution("fi1", X[i], double.NaN,
T[k]);
                    B1[N1] = ALLfi_psi_Fij_solution("fi2", X[i], double.NaN,
T[k]);

                    double[] un1;
                    Matrix.SLE_Thomas(A1, B1, out un1);

                    for (int j = 0; j <= N1; j++) U[i, j, k + 1] = un1[j];
                });
```

Чтобы замерить время, затраченное на обычное и параллельное исполнение метода переменных направлений, нам потребуется получить текущее время с помощью свойства DateTime.Now перед выполнением метода и после, а затем, вычесть из большего меньшее. Свойство DateTime.Now возвращает объект, который содержит текущую дату и время на компьютере, с которого запускается приложение.

Результаты замера времени записываются в файл. Для работы с файлом был использован класс StreamWriter пространства System.IO. При создании класса, используется конструктор StreamWriter(string writePath, bool rewrite), где writePath – путь к файлу, который будет открыт для записи, rewrite – флаг, указывающий, требуется ли новые данные добавлять в конец файла,

либо очищать его содержимое и потом осуществлять запись. Сама же запись происходит с помощью метода sw.WriteLine(string text), где sw - объект класса StreamWriter, а text – строка текста, которая будет записана в файл.

Пример файла с результатами представлен на рис. 2.5.

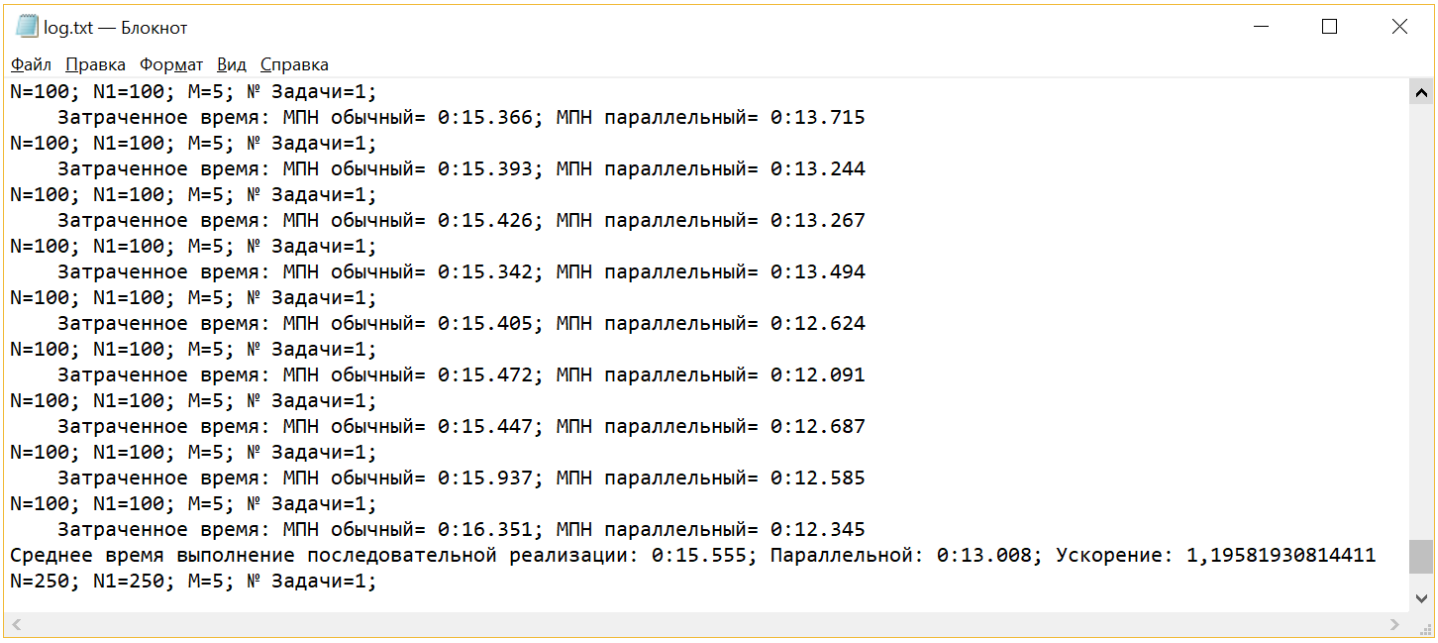


Рис. 2.5. Файл, содержащий параметры задачи и время выполнения метода переменных направлений.

Измерения проводились на ноутбуке с процессором Intel® Core™ i7-3632QM, с тактовой частотой 2,20 GHz, 4 ядрами и 8 потоками и 6 ГБ оперативной памяти. Каждое измерение было повторено 50 раз. В таблице представлено среднее арифметическое время выполнения. На его основе высчитано ускорение параллельного метода S=Тпосл/Тпар, где значение S<1 значит, что параллельная реализация выполняется дольше последовательной и не эффективна. При значении S>1, параллельная реализация дает выигрыш во времени в S раз.

Результаты измерений времени выполнения последовательного и параллельного выполнения метода переменных направлений на различных расчетных сетках для варианта краевой задачи №1 представлены в таблице 2.1.

Таблица 2.1

Сравнение времени выполнения последовательного и параллельного метода переменных направлений для варианта задачи №1

N (ось x)	N1 (ось y)	Время (Мин:Сек.МС) Последовательный	Время (Мин:Сек.МС) Параллельный	Ускорение
10	10	0:0.088	0:0.123	0,716
100	100	0:15.555	0:13.008	1,195
250	250	5:4.398	3:48.999	1,329

Таким образом, исходя из значений в таблице, можно сделать вывод, что параллельный метод изначально дает проигрыш во времени. С увеличением количества узлов по осям *x,y* (N,N1) до 100 и более, параллельная реализация дает преимущество во времени выполнения, ускорение больше 1.

Это обусловлено тем, что Parallel.For, в отличии от обычного for, - это не только цикл, генерирующий задачи в каждой итерации или для каждого фрагмента данных определенного размера, а метод, приспосабливающийся к темпу выполнения отдельных итераций, учитывающий количество выполняющихся в каждый момент задач, исключающий вероятность дробления диапазона на слишком мелкие фрагменты, производящий его деление динамически. Все эти подзадачи по оценке ситуации занимают время, которое в случае недостаточно большого количества слоев по осям *x,y* оказывается неэффективным, но с увеличением их количества параллельная реализация метода переменных направлений дает выигрыш во времени.

Однако, из-за недостаточного количества оперативной памяти не удалось замерить время на сетках 500x500 и 1000x1000. При замере на сетке 500x500 в методе с использованием Parallel.For объем занимаемой

приложением памяти достигает максимально возможного и приложение вылетает с соответствующем исключением - OutOfMemoryException. На рис. 2.6. приведен скриншот диспетчера задач, где занятая оперативная память близка к максимальной.

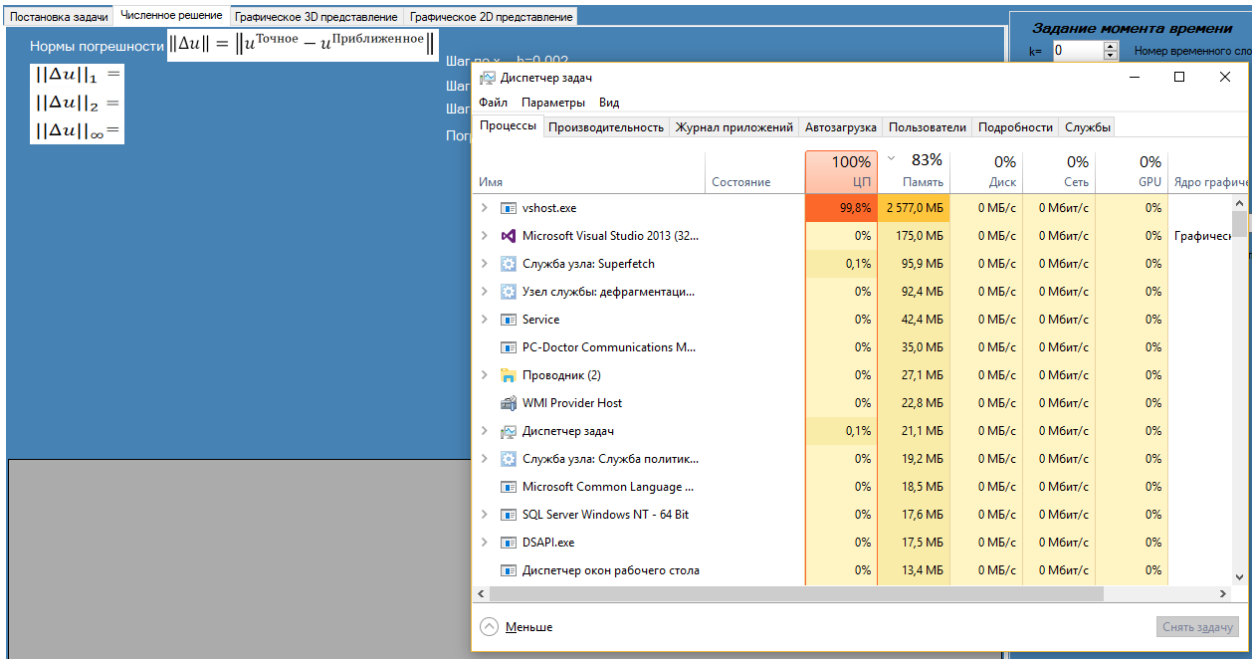


Рис. 2.6. Окно диспетчера задач при сетке 500x500

Данная проблема обусловлена тем, что в цикле Parallel.For создается множество массивов левых частей, которые являются трехдиагональными матрицами (размерность 500x500), массивы правых частей (размерность 500), хранятся массивы с результатами вычислений. А при выполнении обычного цикла for, каждый массив в единственном экземпляре.

2.6. Сохранение анимационного изображения в gif файл

Для сохранения анимационного изображения в gif файл, требуется объединить статические изображения цветного графика и графика изолиний в одно изображение. Аналогично требуется сделать на всех временных слоях. Затем записать полученные изображения в список из компонентов Bitmap (List<Bitmap>). После сохранения изображений в список на всех временных слоях, требуется записать данный список в файл.

На рис.2.7. представлена схема записи списка изображений gif файл:

Рис.2.7. Схема записи в gif файл

Объединение двух графиков в один

Необходимо чтобы на каждом временном слое оба графика были представлены компонентами bitmap. В результате изображения изолиний, цветной график уже был переведен из компонента SimpleOpenGLControl в график в Bitmap. График изолиний изначально создан в компоненте Bitmap. Теперь, для их объединения в один, надо создать новый компонент bitmap, записав график изолиний “как есть”, а второй график переписать со смещением в правую сторону, чтобы были видны оба графика на изображении. Коэффициентом смещения выступает ширина компонента Bitmap для изолиний. Точки цветного графика на новом Bitmap определяются как коэффициент смещения + координаты точки.

Запись списка изображений в файл

Для того чтобы сохранять в файл анимированное изображение, требуется подключить библиотеку Bumpkit. Это библиотека .NET, которая расширяет возможности библиотеки GDI System.Drawing. Используется для генерации изображений «на лету» в веб-приложениях на базе .NET.

В библиотека содержит следующие расширения:

- масштабирование изображения

- поворот изображения
- легкий и быстрый доступ к пикселям
- генерация анимированного GIF изображение (использует собственную кодировку gif)
- другие расширения [12].

В процессе реализации записи списка изображений в gif файл, были использованы следующие методы:

- MemoryStream() - создает поток памяти для чтения и записи данных.
- GifEncoder(MemoryStream stream, int repeatCount) – объединяет несколько изображений в один анимированный файл, где stream – поток памяти, в который идет запись, repeatCount – количество повторения анимации.
- encoder.AddFrame(Image image, TimeSpan delay) – добавляет в память изображение , где image – изображение, delay – задержка (мс) между сменой кадров.
- FileStream(String path, FileMode filemode, FileSystemRights filesystemrights)) - Предоставляет поток в файле, поддерживая синхронные и асинхронные операции чтения и записи, где path – путь к файлу, filemode – режим открытия файла (создать новый файл или открыть существующий), filesystemrights – список доступных прав (используется FileAccess.Write, т.е. открывается файл для записи)
- stream.WriteTo(fileStream) – записывает из потока памяти MemoryStream в файловый поток FileStream.

На рис 2.8. представлен сохраненный gif файл.

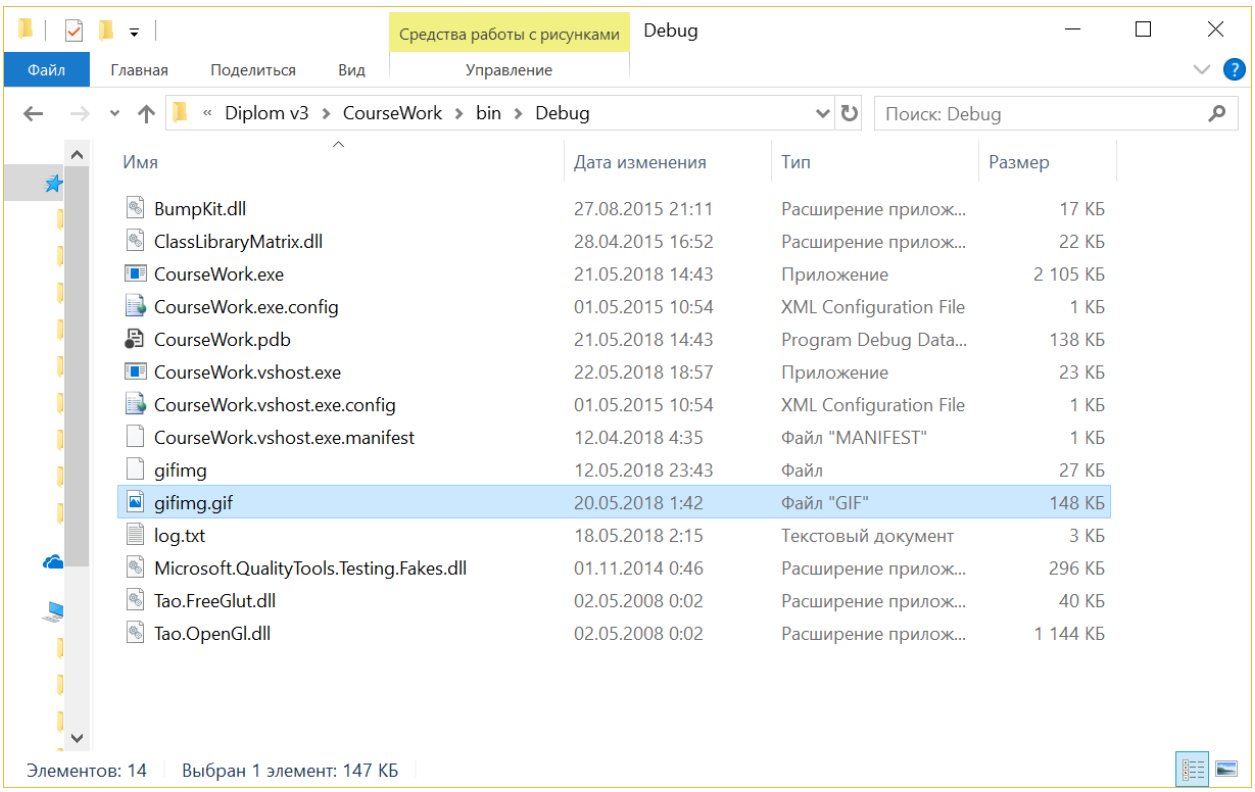


Рис. 2.8. Сохраненный gif файл

Глава 3. Описание интерфейса приложения

Форма приложения после его запуска отображена на рис.3.1. Интерфейс приложения разделен на две части. Левая часть содержит меню из четырех переключаемых вкладок, правая часть – статичная, и не изменяется при смене вкладок в левой части.

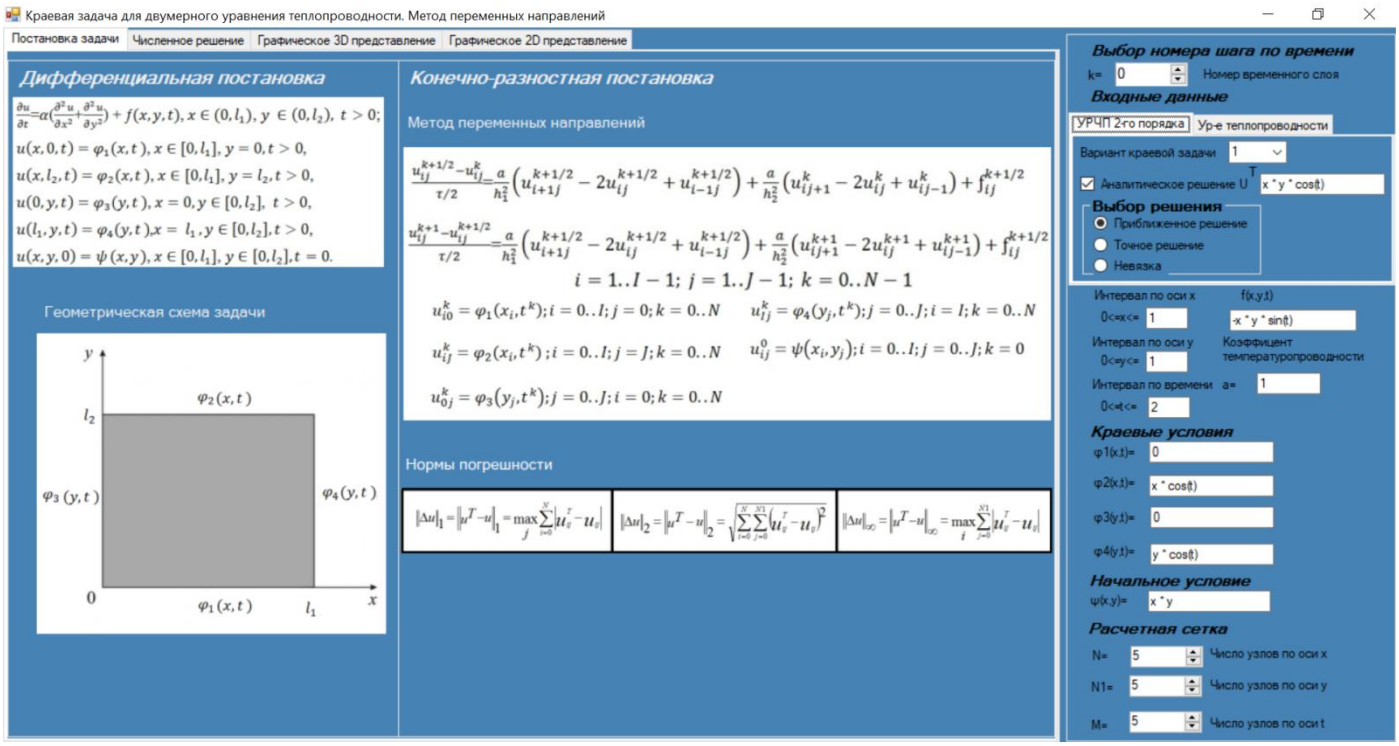


Рис. 3.1. Главное окно приложения

На правой панели, для начала работы, требуется выбрать тип решаемой задачи: дифференциальное уравнение в частных производных 2-го порядка параболического типа или уравнение теплопроводности. Выбор происходит переключением соответствующих вкладок.

При выборе “УРЧП 2-го порядка”, пользователю предлагается выбрать 1 из 5 заранее подготовленных вариантов краевой задачи, или, выбрать последний вариант (все поля станут пустыми) и задать все входные данные вручную. Подготовленные варианты имеют аналитическое решение. Также, можно выбрать между отображением приближенного, точного решений или невязки (рис 3.2.)

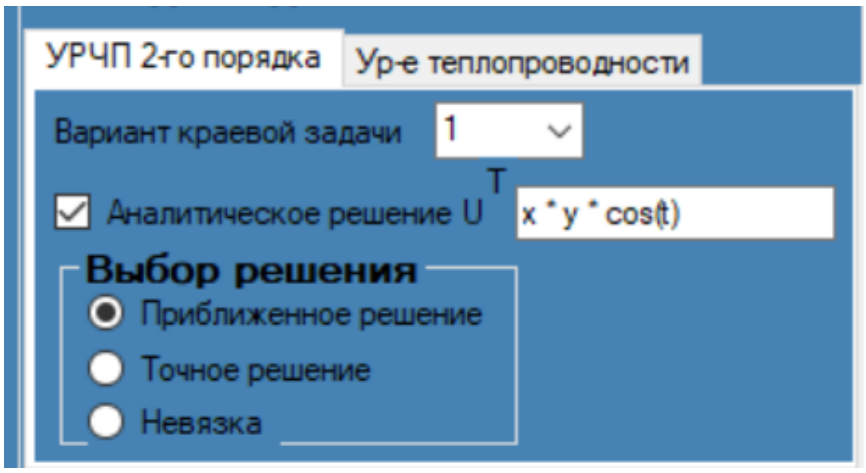


Рис. 3.2. Выбор “УРЧП 2-го порядка” в качестве типа решаемого уравнения

При выборе “Ур-е теплопроводности”, пользователю предлагается либо выбрать подготовленный вариант краевой задачи, либо задать все входные данные самостоятельно (выбор последнего варианта). В случае с уравнением теплопроводности, аналитическое решение отсутствует. Соответственно, отсутствует поле для его задания и меню переключения между приближенным, точным решениями и невязкой. Имеется возможность задать точку, в которой источник будет воздействовать на пластину (рис.3.3).

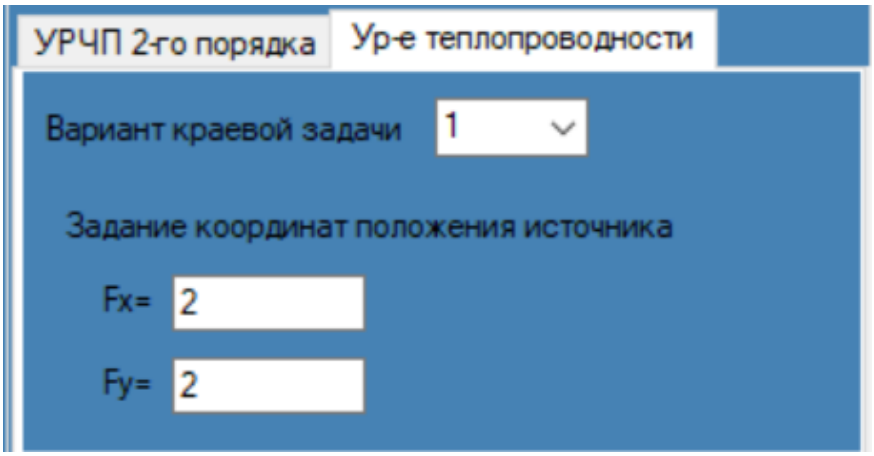


Рис. 3.3. Выбор “Ур-е теплопроводности” в качестве типа решаемого уравнения

На правой панели пользователь имеет возможность задавать различные входные данные: интервал по осям *x*,*y*,*t*, коэффициент

теплопроводности *a*, источник тепла - правую часть двумерного уравнения теплопроводности, краевые и начальное условия, параметры расчетной сетки (количество узлов расчетной сетки по осям *x*,*y*,*t*). При выборе “УРЧП 2-го порядка”, данные во всех полях являются безразмерными (рис.3.4)

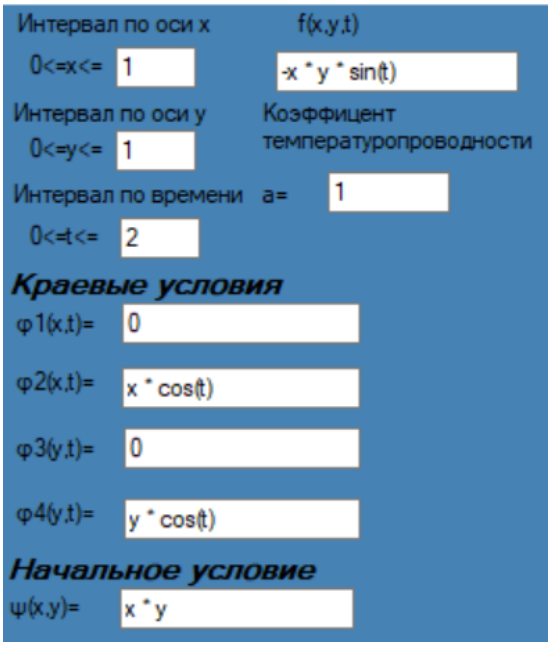


Рис. 3.4. Отсутствие подписи размерности данных

При переключении на вкладку “Ур-е теплопроводности”, рядом с полями отображаются их единицы измерения (рис. 3.5).

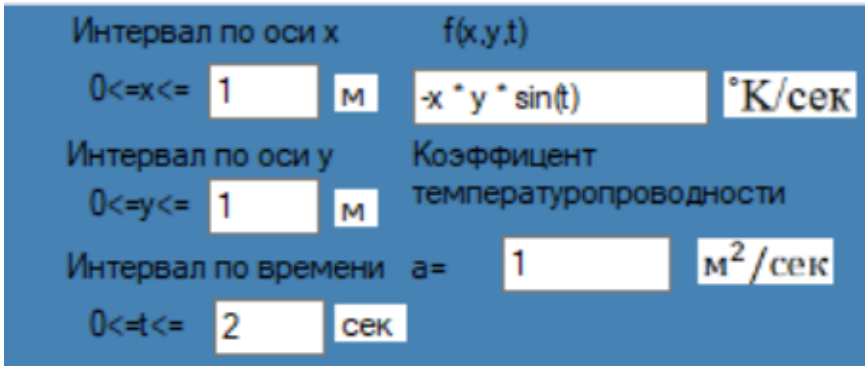


Рис. 3.5. Подпись размерности данных рядом с полями

Решение уравнения можно выводить на различных временных слоях. Для этого достаточно задать нужный временной слой k в пункте “Задание момента времени”.

Далее, рассмотрим содержимое каждой вкладки: “Постановка задачи”, “Численное решение”, “Графическое 3D представление”, “Графическое 2D представление”.

3.1. Вкладка “Постановка задачи”

Вкладка “Постановка задачи” содержит краткую теорию о краевой задаче для двумерного уравнения теплопроводности и о методе её решения. Окно вкладки “Постановка задачи” разделена на две части. Слева – дифференциальная постановка и геометрическая схема задачи, справа - конечно-разностная постановка задачи с использованием метода переменных направлений и формулы норм погрешности решения (рис. 3.6.).

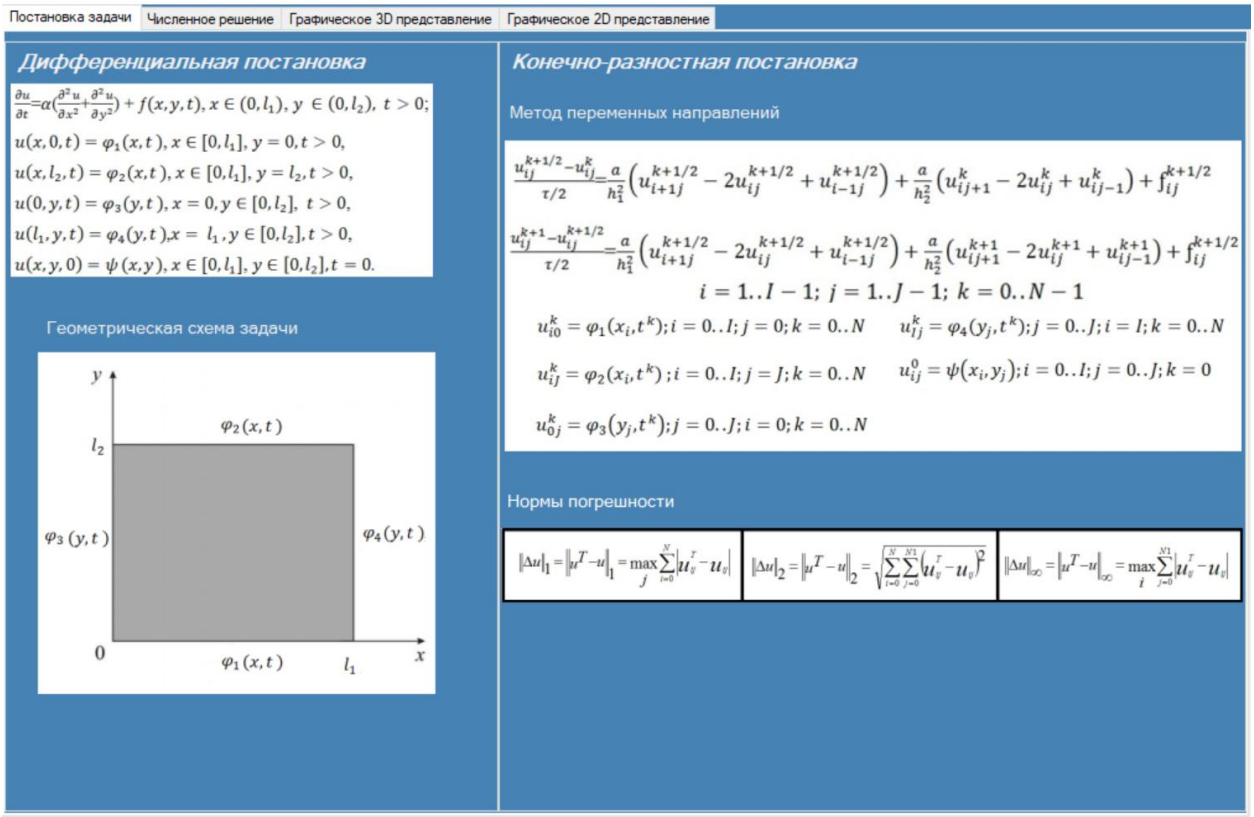


Рис.3.6. Вкладка “Постановка задачи”

Геометрическая постановка задачи наглядно на графике показывает интервалы по осям *x* и *y*, краевые условия задачи. Это поможет пользователю корректно задать краевые условия задачи (рис. 3.7).

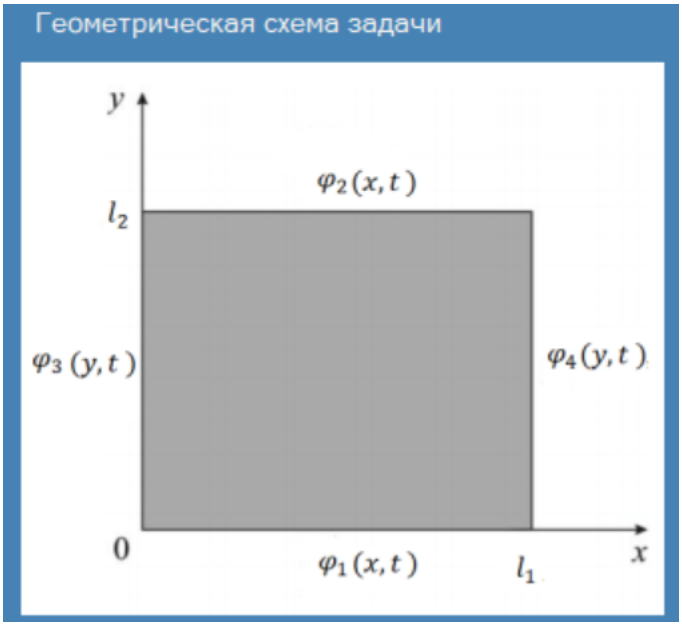


Рис.3.7. Геометрическая постановка задачи

Изображение с нормами погрешностей показывает, по каким формулам рассчитываются нормы во вкладке “Численное решение” (рис. 3.8).

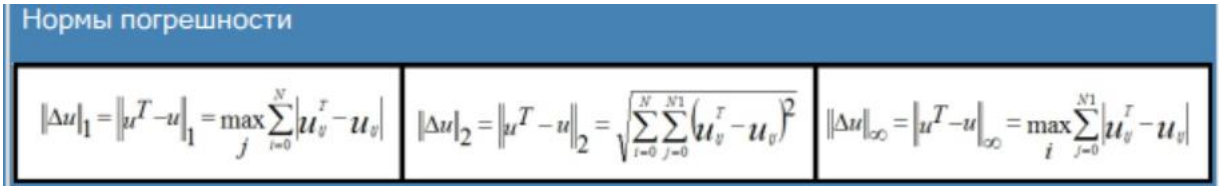
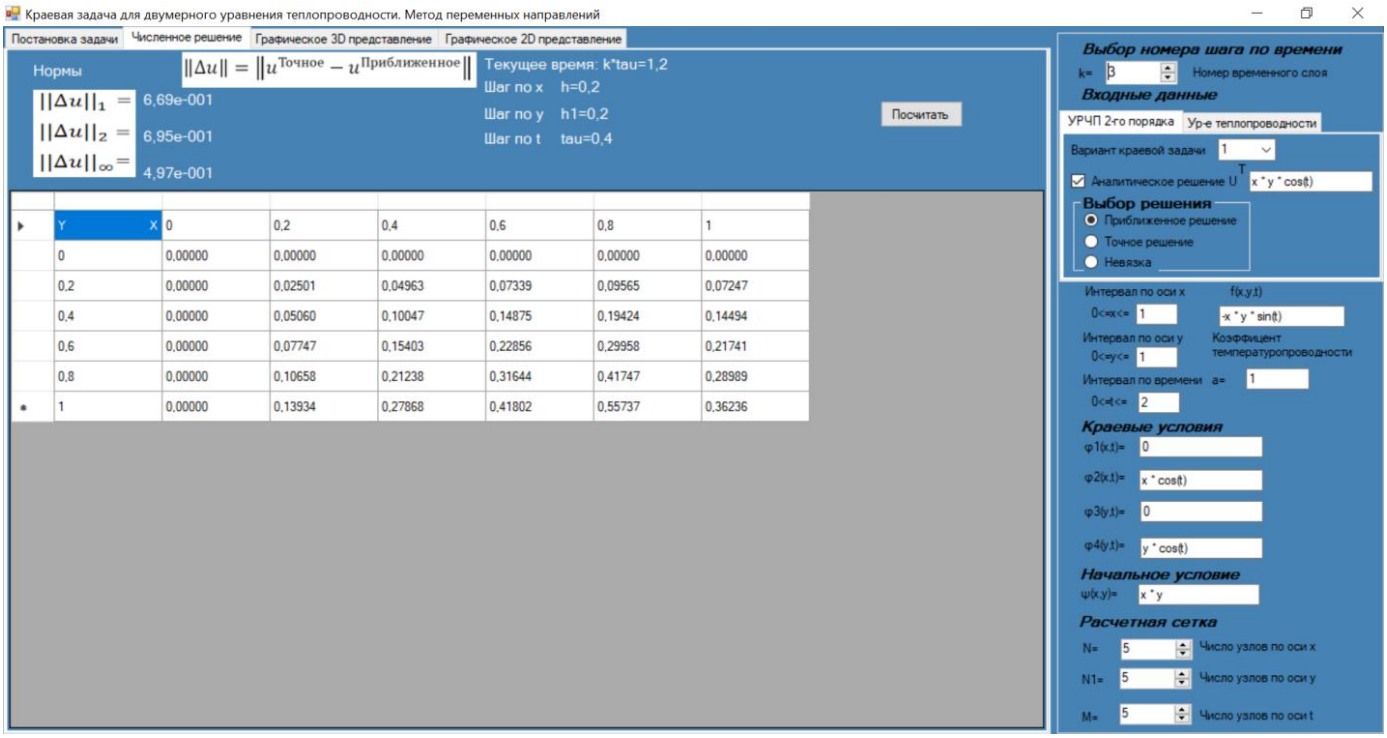


Рис. 3.8. Нормы погрешности

3.2. Вкладка “Численное решение”

Численное решение краевой задачи для двумерного уравнения теплопроводности в виде таблицы значений по осям *x*, *y* и значений температуры было реализовано в ходе выполнения выпускной квалификационной работы бакалавра.

На рис.3.9. представлено содержимое вкладки “Численное решение”.



Y \ X	0	0,2	0,4	0,6	0,8	1
0	0,00000	0,00000	0,00000	0,00000	0,00000	0,00000
0,2	0,00000	0,02501	0,04963	0,07339	0,09565	0,07247
0,4	0,00000	0,05060	0,10047	0,14875	0,19424	0,14494
0,6	0,00000	0,07747	0,15403	0,22856	0,29958	0,21741
0,8	0,00000	0,10658	0,21238	0,31644	0,41747	0,28989
1	0,00000	0,13934	0,27868	0,41802	0,55737	0,36236

Рис. 3.9. Вкладка “Численное решение”.

При нажатии на кнопку “Посчитать” в данной вкладке, на основании входных данных в правой части приложения, отображается численное решение краевой задачи для двумерного уравнения теплопроводности в виде таблицы значений температуры по осям *x*, *y*. Для просмотра численного решения на любом доступном временном слое требуется на правой панели в пункте “Задание момента времени” указать номер требуемого временного слоя k.

Рассчитаны шаги расчетной сетки по осям *x*,*y*,*t*, текущее время (*k***tau*), нормы погрешности (в случае наличия точного решения). Представлена погрешность аппроксимации метода переменных направлений.

3.3. Вкладка “Графическое 3D представление”

Графическое 3D представление решения краевой задачи для двумерного уравнения теплопроводности было реализовано в ходе выполнения выпускной квалификационной работы бакалавра. График изображается на основе полученного численного решения. При изменении

любых входных данных, решение пересчитывается и график обновляется (рис. 3.10.)

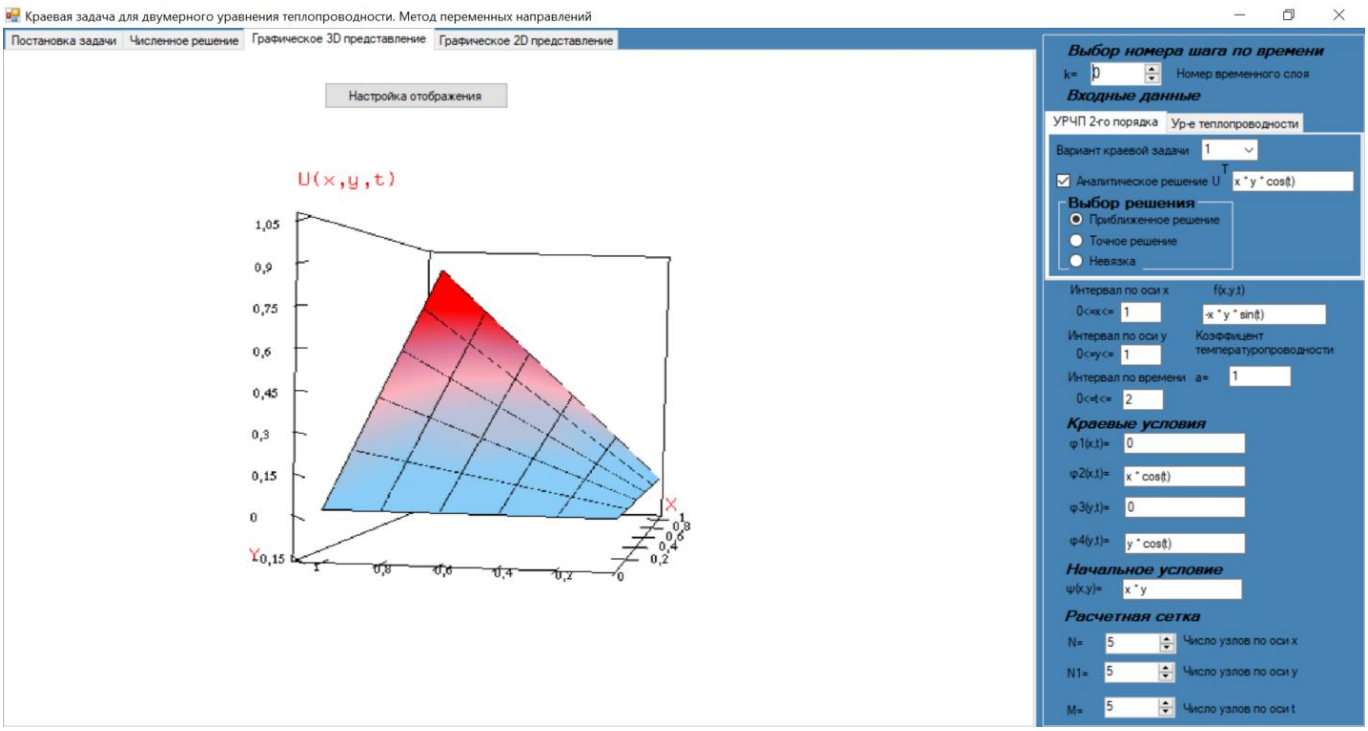


Рис.3.10. Трехмерный график решения уравнения теплопроводности

Данный график был получен с помощью средств OpenGL. Чтобы отобразить точки решения на графике, достаточно вызовов 2х функций: функция задания цвета и функция задания координат точки. Точки между собой соединены линиями в виде сетки. Для плавного перехода цвета от одного к другому, точки были заданы группой по 4 штуки. У каждой точки свой цвет, и OpenGL сам сделает переход от одного цвета к другому.

При нажатии на кнопку “Настройка отображения”, открывается окно, в котором можно переместить или повернуть график вокруг любой из 3х осей (рис. 3.11).

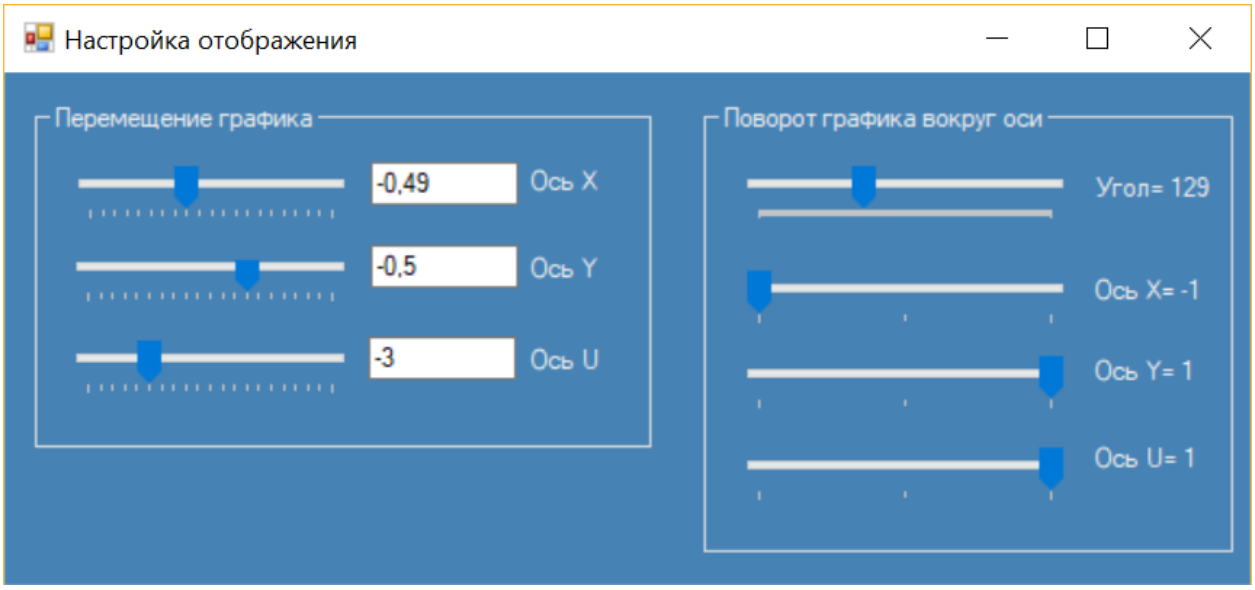


Рис. 3.11. Настройка отображения графика

Функции перемещения и поворота графика были реализованы с помощью OpenGL. Достаточно указать параметры в функции перемещения (координаты по осям *x*,*y*,*z*) и параметры в функции поворота (угол и значения поворота по *x*,*y*,*z*), привязать изменение значений в trackbar к вызову соответствующих функций.

3.4. Вкладка “Графическое 2D представление”

Для расширения функционала приложения и улучшения представления пользователя о течении физического процесса распространения температуры в пластине было решено реализовать анимацию двух графиков: график распределения температуры и график изолиний.

Поскольку уже был опыт работы с OpenGL при реализации трехмерной графики, данная библиотека была выбрана и для работы с двумерной графикой. Однако для работы с изолиниями и сохранением в gif файл был использован GDI+. Таким образом, было совмещено использование OpenGL и GDI+: из каждой библиотеки были выбраны наиболее подходящие под поставленные задачи методы.

На вкладке “Графическое 2D представление” представлена визуализация численного решения краевой задачи уравнения теплопроводности (рис. 3.12.).

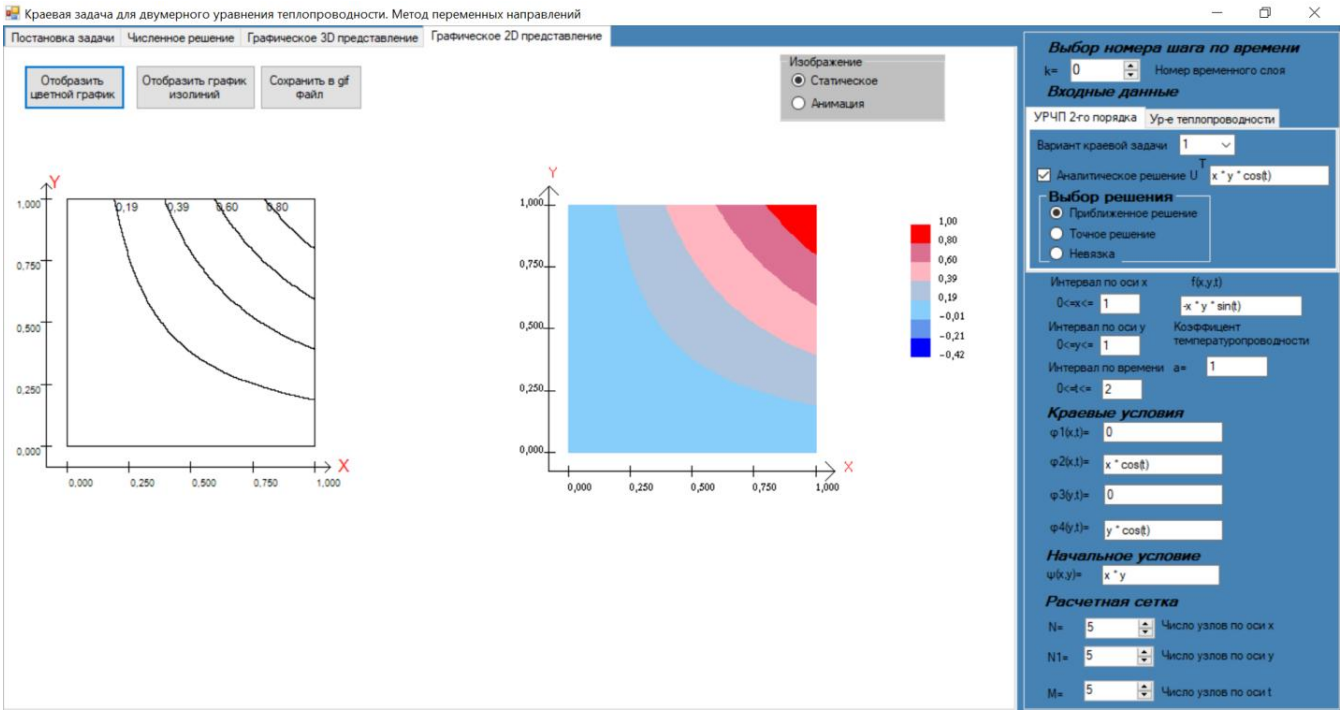


Рис. 3.12. Визуализация численного решения краевой задачи уравнения

Изначально, после задания всех входных данных, графики не отображаются на форме. Для их визуализации требуется нажать соответствующие кнопки на форме: кнопку “Отобразить цветной график” для представления графического решения краевой задачи двумерного уравнения теплопроводности в виде графика в цвете, кнопку “Отобразить график изолиний” для визуализации графика изолиний.

При нажатии на кнопку “Сохранить в gif файл”, в директории с .exe файлом приложения создастся анимационное изображение в формате gif. Файл позволять наблюдать процесс изменения температуры в пластине с заданными входными данными. Данный файл представляет собой оба графика, для которых каждую секунду будет меняться временной слой (от первого до последнего) и, соответственно, последовательно отобразятся оба графика на всех временных слоях.

Решение краевой задачи двумерного уравнения теплопроводности можно представить на различных временных слоях посредством их переключения на панели в правой части. При наличии точного решения, для каждого из графиков доступен выбор отображаемого решения: приближенное или точное. Кроме этого, доступен выбор вида отображения графиков на форме: статическое или анимация. (Рис.3.13.)

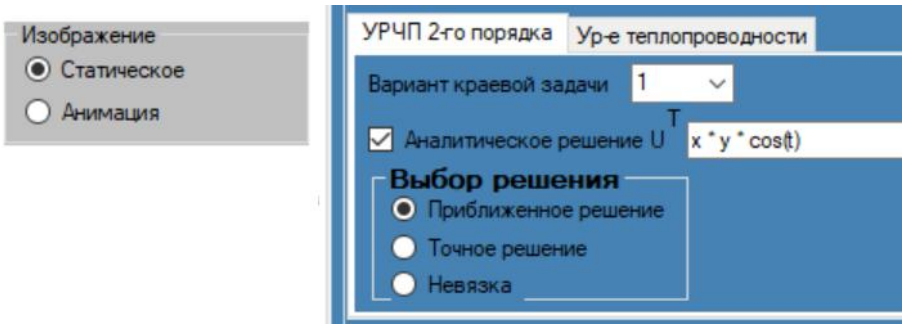


Рис. 3.13. Функционал приложения.

При выборе статического отображения, нужно самому вручную переключать номер временного слоя. При выборе анимации временные слои сами будут переключаться от начального до конечного момента времени.

Один из них графиков представляет собой набор изолиний. Значения температуры изолиний указаны рядом с ними. (Рис. 3.14)

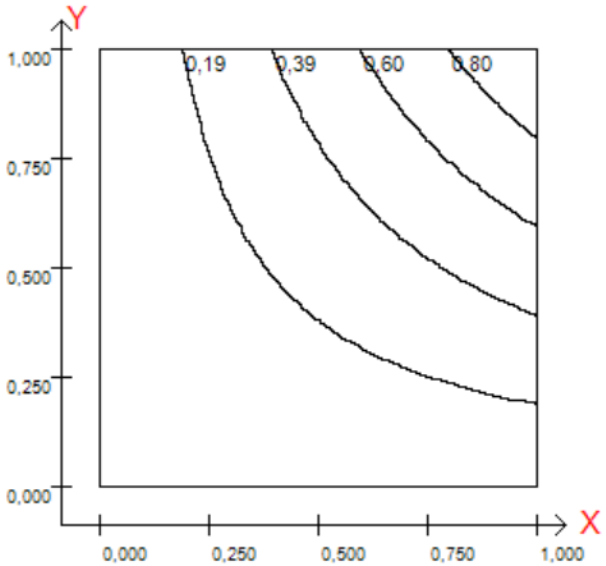


Рис. 3.14. График изолиний

Другой график в цвете представляет распределение температуры. Интервал значений температуры можно определить по цветовой шкале, которая находится справа от графика (Рис. 3.15).

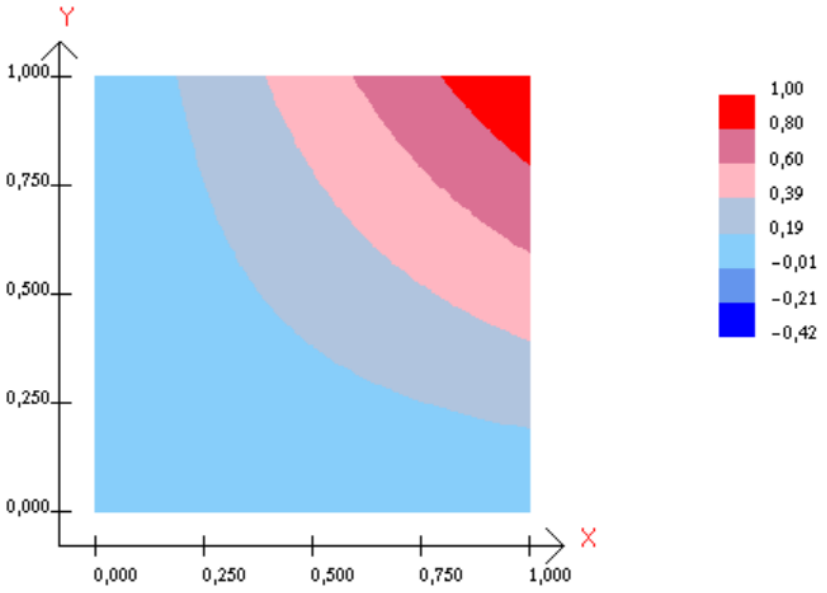


Рис. 3.15. График распределения температуры в цвете

При выборе вкладки “Ур-е теплопроводности”, пользователь может задать положение источника в определенной точке. К примеру, график решения уравнения теплопроводности без источника на последнем временном слое представлен на рис. 3.16

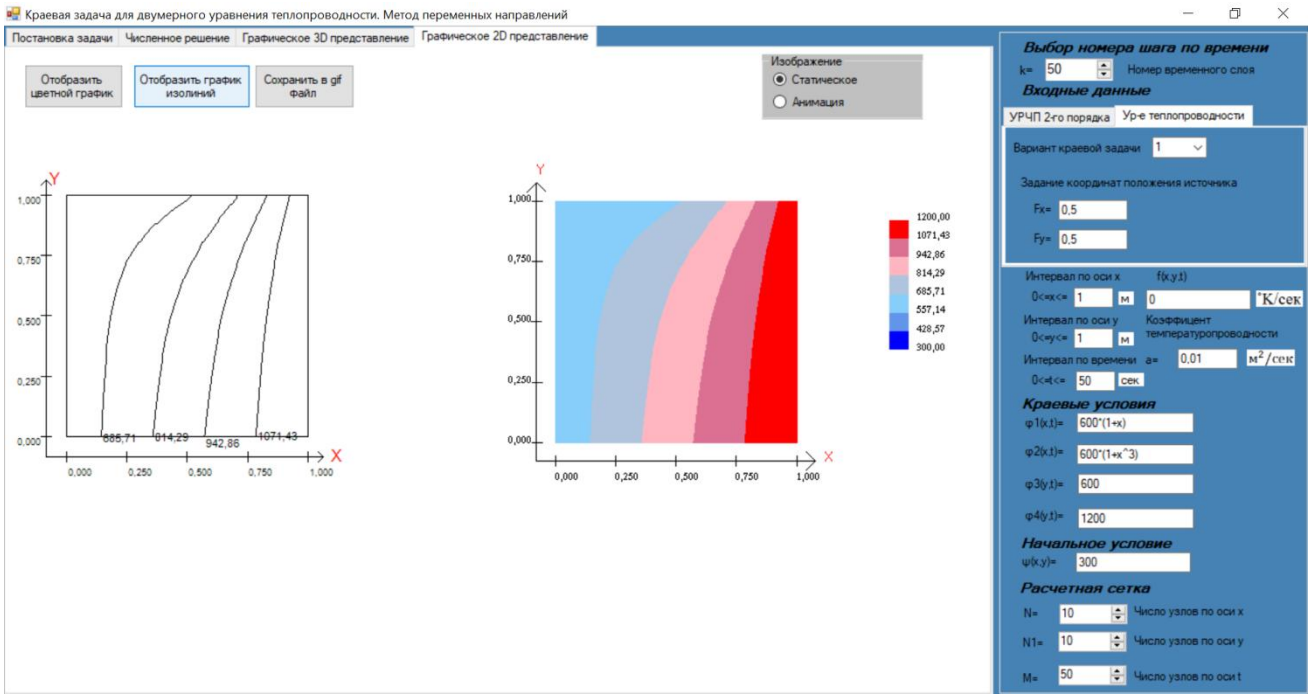


Рис. 3.16. График решения уравнения теплопроводности без источника

При задании источника, и указания его положения в середине пластины, график имеет следующий вид (рис. 3.17)

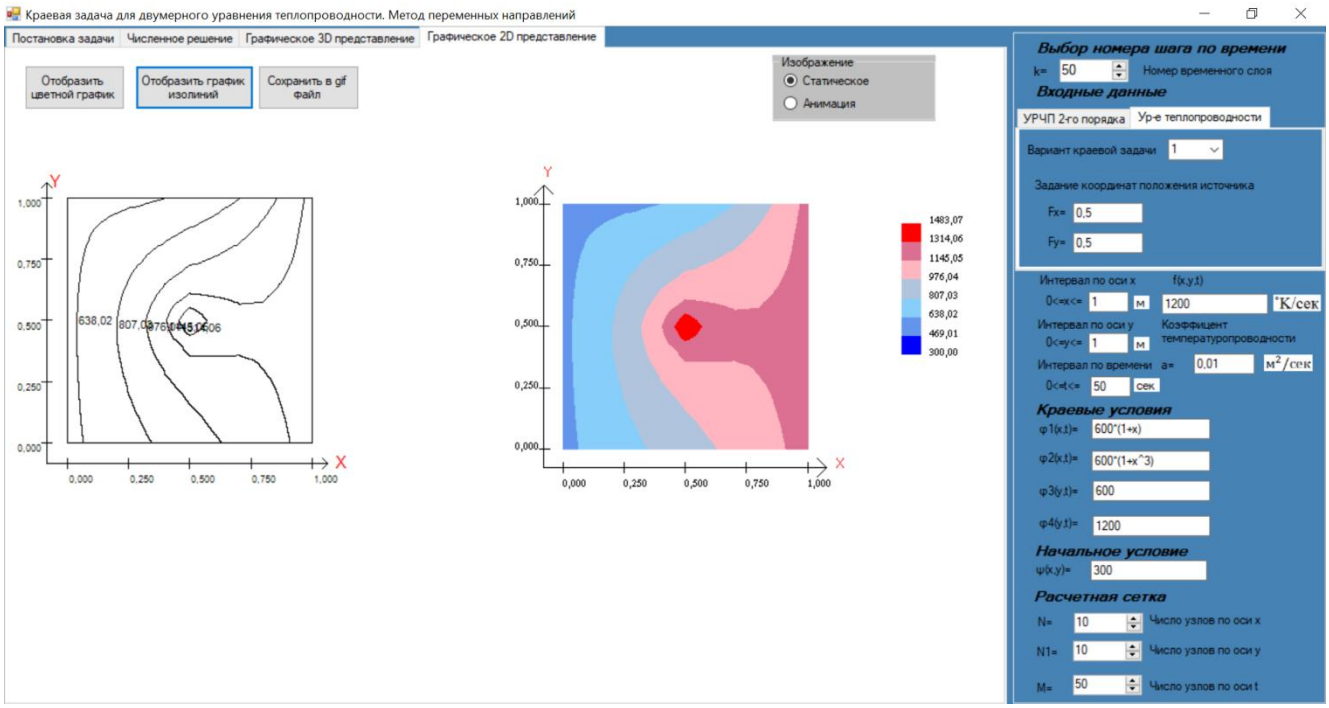


Рис.3.17. График решения уравнения теплопроводности с источником в середине пластины

Заключение

В результате выполнения выпускной квалификационной работы было разработано приложение для визуализации численного решения краевой задачи уравнения теплопроводности с двумя пространственными переменными.

Решены следующие задачи:

- Изучены конечно-разностные методы для решения уравнений параболического типа с двумя пространственными переменными.
- Реализован метод переменных направлений с использованием трехдиагональной прогонки по оси X и оси Y.
- Изучены возможности графической библиотеки OpenGL, которые использовались для визуализации результатов численного решения.
- Реализовано параллельное выполнение циклов с помощью метода Parallel.For в методе переменных направлений.

Разработанная программа обладает следующим функционалом:

- Приложение позволяет задавать различные начальные и краевые условия, параметры расчетной сетки.
- В приложении предусмотрено как численное, так и графическое отображение полученного приближенного решения.
- Приложение позволяет сохранять анимацию распределения температуры в gif файл.

Список литературы

1. Кузнецов, Г.В. / Разностные методы решения задач теплопроводности: учебное пособие / Г.В. Кузнецов, М.А. Шеремет. – Томск: Изд-во ТПУ, 2007. – 172 с.
2. Березин И.С., Жидков Н.П. Методы вычислений. Т.1, 2. М., Физматгиз, 1962– 464 с.
3. Солодов А.П. Численные методы теплопроводности. Электронный курс. [On-line]:http://docplayer.ru/34392474-9-chislennye-metody-eploprovodnosti.html (дата обращения: 05.03.2018)
4. Пирумов У.Г. Численные методы: Учеб. пособие для студ. втузов. – 3-е изд., испр. – М.: Дрофа, 2004. –224с.
5. Калиткин Н.Н. Численные методы. М., Наука, 1978 – 512 с.
6. Троелсон Э. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е изд. – М.: И.Д. «Вильямс», 2013. – 1312 с.
7. Запечников С. В. Визуализация данных и процессов с использованием кроссплатформенного программного интерфейса OpenGL. Изд.: Российский государственный гуманитарный университет (Москва). № 11 (133), 2014 г., С.: 184-214.
8. Васильев С.А. OpenGL. Компьютерная графика: учеб. Пособие. – Тамбов: 2012. – 81 с.
9. Оптимизация приложений .NET Framework. Параллелизм данных [On-line]: https://professorweb.ru/my/csharp/optimization/level4/4_4.php (дата обращения: 15.05.2018)
10. Оптимизация приложений .NET Framework. Параллельное программирование. [On-line]: https://professorweb.ru/my/csharp/optimization/level4/4_1.php (дата обращения: 16.05.2018)
11. Верма Р. Д. Введение в OpenGL. М., Горячая Линия – Телеком, 2004. – 304с.

12. Шилдт Г. C# 3.0. Полное руководство.: Пер. С англ. – М.:ООО «И.Д.Вильямс», 2010. – 992 с.

13. GDI+. Лекция из курса «Введение в программирование на C# 2.0». [On-line]: http://citforum.ru/programming/csharp/gdi_plus/ (дата обращения: 01.06.2018)

14. Вержбицкий В.М. Численные методы (линейная алгебра и нелинейные уравнения). Учеб. Пособие для вузов. М., Высшая Школа, 2002 – 840 с.

15. Крайнов А.Ю., Миньков Л.Л. Численные методы решения задач тепло и массопереноса : учеб. пособие.– Томск : STT, 2016. – 92 с.

16. Самарский, А.А. Теория разностных схем / А.А. Самарский – М. : Наука, 1977. – 656 с.

17. Шрайнер Д., Ву М., Нейдер Дж., Девис Т.OpenGL. Руководство по программированию. Спб.: Питер, 2006. – 624с.

Приложение 1

Визуализация распределения температуры внутри пластины в виде 2D графика

```
            Gl.glBegin(Gl.GL_POINTS);
            double x = 0; double y = 0;
            int countX = 0;
            int countY = 0;

            for (int i = 0; i < N; i++)
            {
                double onepartX = Math.Abs(X[i + 1] - X[i]) / partsX;

                while (true)
                {
                    x = X[i] + onepartX * countX;
                    if (x > X[i + 1]) { countX = 0; break; }
                    for (int j = 0; j < N1; j++)
                    {
                        double onepartY = Math.Abs(Y[j + 1] - Y[j]) /
partsY;
                        double x1 = X[i];
                        double y1 = Y[j];
                        double y2 = Y[j + 1];
                        double x2 = X[i + 1];
                        double q11 = U[i, j, tCur];
                        double q12 = U[i, j + 1, tCur];
                        double q21 = U[i + 1, j, tCur];
                        double q22 = U[i + 1, j + 1, tCur];
                        while (true)
                        {
                            y = Y[j] + onepartY * countY;
                            if (y > Y[j + 1]) { countY = 0; break; }
                            double bi =
Computing.BilinearInterpolation(x1, x2, y1, y2, q11, q12, q21, q22, x,
y);
                            var clr = mnc.GetNewColor_V2(bi);
                            Gl.glColor4ub(clr.R, clr.G, clr.B,
clr.Alpha);
                            Gl.glVertex2d(5 + x * multX, 15 + y *
multY);
                            countY++;
                        }
                    }
                    countX++;
                }
            }
            Gl.glEnd();
```

Приложение 2

Визуализация распределения температуры внутри пластины в виде 3D графика

```
            double minValue = U[0, 0, 0];
            double maxValue = U[0, 0, 0];

            for (int i = 0; i <= N; i++)
            {
                for (int j = 0; j <= N1; j++)
                {
                    for (int jj = 0; jj <= M; jj++)
                    {
                        if (U[i, j, jj] > maxValue) maxValue = U[i, j,
jj];
                        if (U[i, j, jj] < minValue) minValue = U[i, j,
jj];
                    }
                }
            }

            MyNewColor mnc = new MyNewColor(minValue,maxValue);
            Color clr;
            foreach (var vv in sqrs)
            {
                Gl.glBegin(Gl.GL_QUADS);

                clr = mnc.GetNewColor(vv.leftbottom.U);
                Gl.glColor3ub(clr.R, clr.G, clr.B);
                Gl.glVertex3d(vv.leftbottom.X + multi, vv.leftbottom.Y
+ multi, vv.leftbottom.U + multi);

                clr = mnc.GetNewColor(vv.lefttop.U);
                Gl.glColor3ub(clr.R, clr.G, clr.B);
                Gl.glVertex3d(vv.lefttop.X + multi, vv.lefttop.Y +
multi, vv.lefttop.U + multi);

                clr = mnc.GetNewColor(vv.righttop.U);
                Gl.glColor3ub(clr.R, clr.G, clr.B);
                Gl.glVertex3d(vv.righttop.X + multi, vv.righttop.Y +
multi, vv.righttop.U + multi);

                clr = mnc.GetNewColor(vv.rightbottom.U);
                Gl.glColor3ub(clr.R, clr.G, clr.B);
                Gl.glVertex3d(vv.rightbottom.X + multi,
vv.rightbottom.Y + multi, vv.rightbottom.U + multi);
                Gl.glEnd();
            }
```