МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(МОСКОВСКИЙ ПОЛИТЕХ)

Информатика и вычислительная техника

Образовательная программа (профиль)

«Интеграция и программирование в САПР»

Кафедра «СМАРТ технологии»

ОТЧЕТ

по дисциплине:

Проектная деятельность

на тему:

Решение вывода изолиний для визуальной оценки.

Преподаватель: / Толстиков А.В., к.т.н. /

подпись ФИО, уч. звание и степень

Студент: / Пересторонин А.М. 201-324 /

подпись ФИО, группа

Москва, 2022 г.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Изоли́ния  или линия уровня (функции) или горизонталь — условное обозначение на карте, чертеже, схеме или графике, представляющее собой линию, в каждой точке которой измеряемая величина сохраняет одинаковое значение. Изолинии — способ представления скалярной функции от двух переменных на плоскости. Это плоское сечение трехмерного графика функции  , параллельное плоскости ,. В картографии горизонталь соединяет точки равной высоты над данным уровнем, например над средним уровнем моря. Контурная карта — это карта, проиллюстрированная контурными линиями, например, топографическая карта, которая, таким образом показывает долины и холмы, а также уклон и степень сглаженности склонов. Контурный интервал контурной карты — это разность высот между последовательными линиями контура. Градиент функции всегда перпендикулярен контурным линиям. Когда линии расположены близко друг к другу, величина градиента велика, изменение крутое. Набор уровней является обобщением линии контура для функций любого числа переменных. На карте, описывающей пересечение реальной или гипотетической поверхности с одной или несколькими горизонтальными плоскостями, горизонтали могут быть изогнутыми, прямыми или смешанными. Конфигурация этих горизонталей позволяет читателям карт оценить относительный градиент параметра и оценивать этот параметр в определенных местах. Горизонтали могут быть либо прослежены на видимой трехмерной модели поверхности, например, когда фотограмметрист, просматривая стереомодель, вычерчивает линии уровня (горизонтали), либо могут быть интерполированы из расчетных высот поверхности, например, когда компьютерная программа интерполирует горизонтали по сети высотных точек. В последнем случае метод интерполяции влияет на точность отдельных горизонталей и на их отображении наклонов, ям и пиков.

JIT-компиляция, динамическая компиляция — технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы. Таким образом достигается высокая скорость выполнения по сравнению с интерпретируемым байт-кодом (сравнимая с компилируемыми языками) за счёт увеличения потребления памяти (для хранения результатов компиляции) и затрат времени на компиляцию. Технология JIT базируется на двух более ранних идеях, касающихся среды выполнения: компиляции байт-кода и динамической компиляции.

Так как JIT-компиляция является, по сути, одной из форм динамической компиляции, она позволяет применять такие технологии, как адаптивная оптимизация и динамическая рекомпиляция. Благодаря этому JIT-компиляция может показывать лучшие результаты в плане производительности, чем статическая компиляция. Интерпретация и JIT-компиляция особенно хорошо подходят для динамических языков программирования, при этом среда исполнения справляется с поздним связыванием типов и гарантирует безопасность исполнения.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Создадим форму с названием Isoline и применим к ней свойства:

* BackColor: white
* Locked: True
* MaximizeBox: False
* MinimizeBox: False

Добавим на создание форму элементы управления, для удобства разделим их на типовые компоненты:

1. Button:
   1. ButtonApplication
   2. ButtonMinus
   3. ButtonPlus
2. TextBox:
   1. TextBoxFormula
   2. ContourIines
3. ComboBox:
   1. ComboBoxDesignation
   2. ComboBoxlines
4. Chart:
   1. ChartGraphic

Перечислим за что отвечают каждый элемент:

* ButtonApplication – Применения условий введённой формулы.
* ButtonMinus – Уменьшения графика в Form.
* ButtonPlus – Увеличения графика в Form.
* TextBoxFormula – Поле ввода формулы.
* ContourIines – Количество изолиний на графике.
* ChartGraphic – Поле вывода графика функции.

Для общего понимания надо упоминать что C# является объектное ориентированным языком и для достижения максимальной продуктивности кода он разбит на отдельные методы.

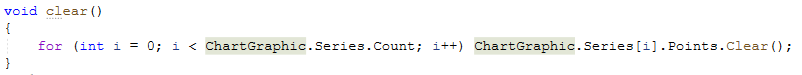
Создадим группу математических методов в связи с тем, что из вызова происходит весьма часто на протяжении всей программы, так что исходя из принципов ООП такой код необходимо выносить в отдельные методы. Что бы не засорять общий код визуально. Они были вынесены как директивы начального класса.

Структура директивы выглядит следующем образом: sin,cos,l1,l2,l3,m1,m2,m3,n1,n2,n3,clear,drawxyz

Здесь надо остановится на двух методах: clear,drawxyz.

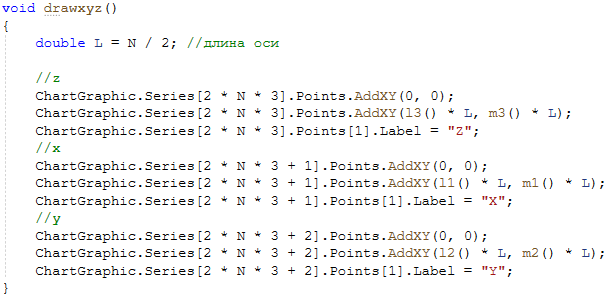
Метод под названием clear отчищает точки сетки функции и так как обрисовка осуществляется с помощи сплайнов, то и при удалении сточек сетки обрисовка не будет производится. Это представлена на листинге 1.

Листинг 1 – Реализация события «Очистить»



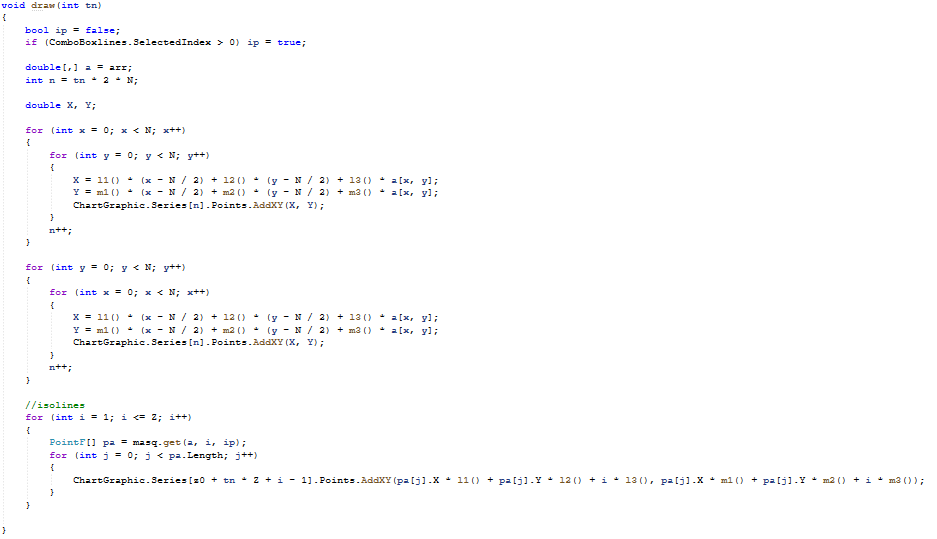
Метод drawxyz производит обрисовку координатной сетки функции по осям X, Y, Z используя частично методы представление выше. Это представлена на листинге 2.

Листинг 2 – Реализация события вывода осей координат.



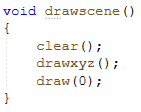
Следующая функция носит название draw. Предназначение этого метода в выводе графика функции вместе с изолинией. Первая группа for обрисовывает линии графика функции по оси X. Вторая группа for рисует лини графика по оси Y. Третья группа циклов выводит изолинии на график. Параметр Z является количество изолиний на графии. Вышеописанные циклы представлены на листинге 3.

Листинг 3 – Реализация события вывода графика.



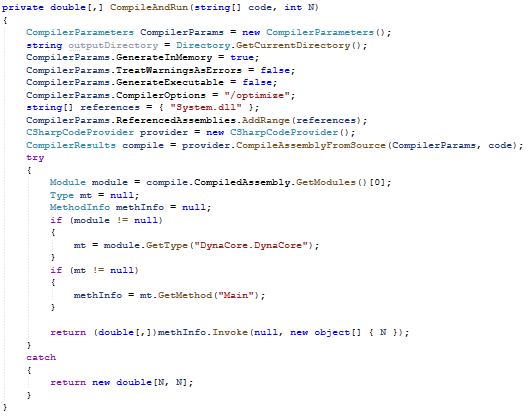
Вышеописанные методы объединены в последовательность команд для вывода нового графика функции. Для этого объявлен метод с названием drawscene где вызывается методы clear, drawxyz, draw. Это представлена на листинге 4.

Листинг 4 – Реализация события вывода графика.



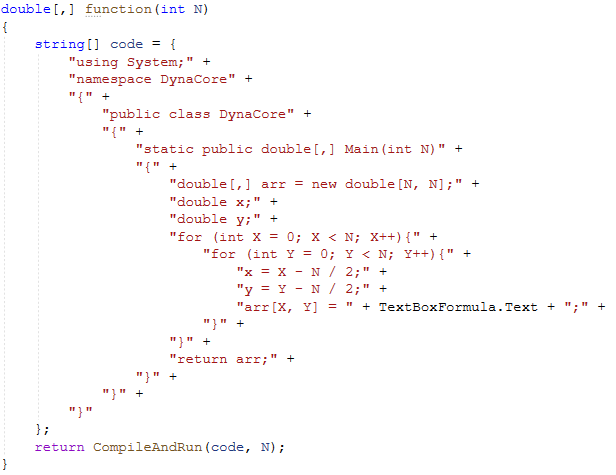
Далее идет сложный набор методов об единёных в дириктивы, так как фактически они выполняют одну функцию компиляция кода на лету. Первая функция представляет компиляция кода с добавлением библиотек из стандартных и другие административные действия проверок работе с api операционной системы. Этот административный код представлен в методе CompileAndRun и показана на листинге 5.

Листинг 5 – Реализация динамическая JIT компиляция кода.



Для работы этого метода необходимо передать параметр code, тем самым далее образуется метод function использующая ранее описанный метод. Это представлена на листинге 6.

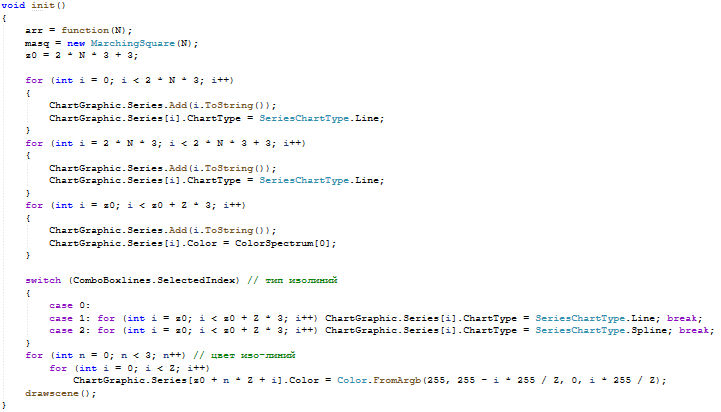
Листинг 6 – Реализация вызов функции CompileAndRun.



Следующая функция является административным методом, который создает экземпляры и переопределяет типы экземпляров.

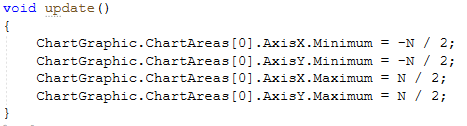
Первые три for добавляют экземпляры графика. По сути эти циклы добавляют просто экземпляры с одновременным переопределениям, этот этап можно было бы заменить сразу обелением экземпляров, однако язык c# со стандартными библиотеками графики не позволяют это сделать. На профессиональном языке такой процесс можно назвать костыль. После идет конструкция switch которая выбирает тип линий изолинии. Далее градируется цвет изолиний в следующей вложенной конструкции for. Выше описанные цикля представлены на листинге 7.

Листинг 7 – Реализация административного метода.



Метод update так же как и предыдущий метод является административным. Он задает минимальные и максимальные значения графика. Если этого метода не будет, то график автоматически будет масштабироваться так как formc# посчитает необходимым. Для более прогнозированного поведения графика функции и нужна этот метод. Это представлена на листинге 8.

Листинг 8 – Реализация выключения авто масштабирования.

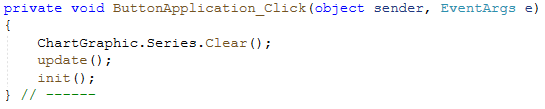


Теперь обратимся к элементу управления формы и коду вызова последовательности ранее описанных методов. Для удобства описания ход описания будет в порядке, приведённом выше.

ButtonApplication

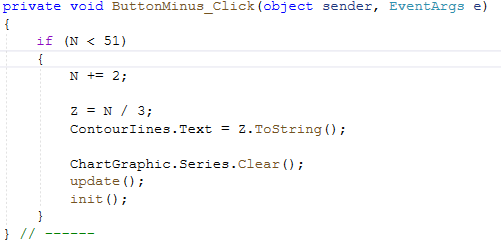
Этот метод вызывает перерисовку графика, а так как при перерисовке применяется перерастёт графика включая компиляцию JIT программы, тем самым и происходит изменения функции графика. Это представлена на листинге 9.

Листинг 9 – Реализация перерисовки графика.



ButtonMinus

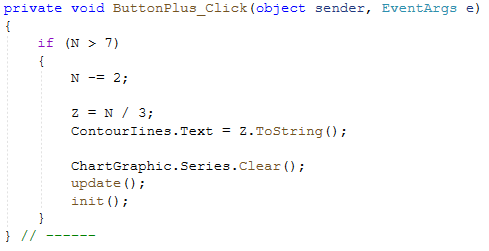
Этот метод изменяет значения масштабирования, количество изолиний и применяет все тот же знакомый набор команд, приведенный выше. Это представлена на листинге 10.

Листинг 10 – Реализация уменьшения графика.

ButtonPlus

Аналогично предыдущему, но в обратную сторону производится уменьшения графика. Это представлена на листинге 11.

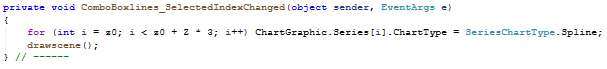
Листинг 11 – Реализация увеличения графика.



ComboBoxlines

Этот элемент управления изменяет тип изолинии, тем самым в нутре он вызывает функцию изменения типа вывода изолиний. Это представлена на листинге 12.

Листинг 12 – Реализация увеличения графика.



Пример работы программы представлен на рисунке 1.

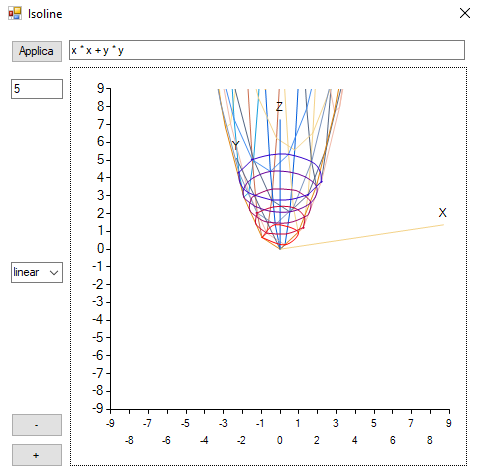


Рисунок 1 – Пример работы программы

РЕЗУЛЬТАТЫ

В результате работы сделана программа по расчету и визуализации изолиний на графике, динамическая компиляция графика и вывод его в form приложении конечного продукта.

ПРИЛОЖЕНИЕ 1. Код файла Form1.cs

|  |
| --- |
| using Microsoft.CSharp;  using System;  using System.CodeDom.Compiler;  using System.Drawing;  using System.IO;  using System.Reflection;  using System.Windows.Forms;  using System.Windows.Forms.DataVisualization.Charting;  namespace Isoline  {  public partial class Isoline : Form  {  static int N = 19; //must be odd  double alpha = 0, beta = 0, gamma = 0; //alpha, beta, gamma  bool onmove = false;  Point startpos;  delegate double func(double x, double y);  int Z = 5;  double[,] arr;  MarchingSquare masq;  int z0;  public Isoline()  {  InitializeComponent();  ContourIines.Text = Z.ToString();  ChartGraphic.ChartAreas[0].AxisX.Interval = 1;  ChartGraphic.ChartAreas[0].AxisY.Interval = 1;  ChartGraphic.ChartAreas[0].AxisX.MajorGrid.LineWidth = 0;  ChartGraphic.ChartAreas[0].AxisY.MajorGrid.LineWidth = 0;  update();  init();  ComboBoxlines.SelectedIndex = 1;  ChartGraphic.Legends.Clear();    }  void update()  {  ChartGraphic.ChartAreas[0].AxisX.Minimum = -N / 2;  ChartGraphic.ChartAreas[0].AxisY.Minimum = -N / 2;  ChartGraphic.ChartAreas[0].AxisX.Maximum = N / 2;  ChartGraphic.ChartAreas[0].AxisY.Maximum = N / 2;  }  void init() // +++++++  {  arr = function(N);  masq = new MarchingSquare(N);  z0 = 2 \* N \* 3 + 3;  for (int i = 0; i < 2 \* N \* 3; i++)  {  ChartGraphic.Series.Add(i.ToString());  ChartGraphic.Series[i].ChartType = SeriesChartType.Line;  }  for (int i = 2 \* N \* 3; i < 2 \* N \* 3 + 3; i++)  {  ChartGraphic.Series.Add(i.ToString());  ChartGraphic.Series[i].ChartType = SeriesChartType.Line;  }  for (int i = z0; i < z0 + Z \* 3; i++)  {  ChartGraphic.Series.Add(i.ToString());  ChartGraphic.Series[i].Color = Color.LightBlue;  }  switch (ComboBoxlines.SelectedIndex) // тип изолиний  {  case 0:  case 1: for (int i = z0; i < z0 + Z \* 3; i++) ChartGraphic.Series[i].ChartType = SeriesChartType.Line; break;  case 2: for (int i = z0; i < z0 + Z \* 3; i++) ChartGraphic.Series[i].ChartType = SeriesChartType.Spline; break;  }  for (int n = 0; n < 3; n++) // цвет изо-линий  for (int i = 0; i < Z; i++)  ChartGraphic.Series[z0 + n \* Z + i].Color = Color.FromArgb(255, 255 - i \* 255 / Z, 0, i \* 255 / Z);  drawscene();  }  #region function, CompileAndRun  double[,] function(int N)  {  string[] code = {  "using System;" +  "namespace DynaCore" +  "{" +  "public class DynaCore" +  "{" +  "static public double[,] Main(int N)" +  "{" +  "double[,] arr = new double[N, N];" +  "double x;" +  "double y;" +  "for (int X = 0; X < N; X++){" +  "for (int Y = 0; Y < N; Y++){" +  "x = X - N / 2;" +  "y = Y - N / 2;" +  "arr[X, Y] = " + TextBoxFormula.Text + ";" +  "}" +  "}" +  "return arr;" +  "}" +  "}" +  "}"  };  return CompileAndRun(code, N);  }  private double[,] CompileAndRun(string[] code, int N)  {  CompilerParameters CompilerParams = new CompilerParameters();  string outputDirectory = Directory.GetCurrentDirectory();  CompilerParams.GenerateInMemory = true;  CompilerParams.TreatWarningsAsErrors = false;  CompilerParams.GenerateExecutable = false;  CompilerParams.CompilerOptions = "/optimize";  string[] references = { "System.dll" };  CompilerParams.ReferencedAssemblies.AddRange(references);  CSharpCodeProvider provider = new CSharpCodeProvider();  CompilerResults compile = provider.CompileAssemblyFromSource(CompilerParams, code);  try  {  Module module = compile.CompiledAssembly.GetModules()[0];  Type mt = null;  MethodInfo methInfo = null;  if (module != null)  {  mt = module.GetType("DynaCore.DynaCore");  }  if (mt != null)  {  methInfo = mt.GetMethod("Main");  }  return (double[,])methInfo.Invoke(null, new object[] { N });  }  catch  {  return new double[N, N];  }  } // +++++++  #endregion // +++++++  void drawscene()  {  clear();  drawxyz();  draw(0);  } // +++++++  void draw(int tn)  {  bool ip = false;  if (ComboBoxlines.SelectedIndex > 0) ip = true;  double[,] a = arr;  int n = tn \* 2 \* N;  double X, Y;  for (int x = 0; x < N; x++)  {  for (int y = 0; y < N; y++)  {  X = l1() \* (x - N / 2) + l2() \* (y - N / 2) + l3() \* a[x, y];  Y = m1() \* (x - N / 2) + m2() \* (y - N / 2) + m3() \* a[x, y];  ChartGraphic.Series[n].Points.AddXY(X, Y);  }  n++;  }  for (int y = 0; y < N; y++)  {  for (int x = 0; x < N; x++)  {  X = l1() \* (x - N / 2) + l2() \* (y - N / 2) + l3() \* a[x, y];  Y = m1() \* (x - N / 2) + m2() \* (y - N / 2) + m3() \* a[x, y];  ChartGraphic.Series[n].Points.AddXY(X, Y);  }  n++;  }  //isolines  for (int i = 1; i <= Z; i++)  {  PointF[] pa = masq.get(a, i, ip);  for (int j = 0; j < pa.Length; j++)  {  ChartGraphic.Series[z0 + tn \* Z + i - 1].Points.AddXY(pa[j].X \* l1() + pa[j].Y \* l2() + i \* l3(), pa[j].X \* m1() + pa[j].Y \* m2() + i \* m3());  }  }  } // +++++++  #region sin,cos,l1,l2,l3,m1,m2,m3,n1,n2,n3,clear,drawxyz  double sin(double x)  {  return Math.Sin(x \* Math.PI / 180);  }  double cos(double x)  {  return Math.Cos(x \* Math.PI / 180);  }  double l1()  {  return cos(alpha) \* cos(gamma) - cos(beta) \* sin(alpha) \* sin(gamma);  }  double m1()  {  return sin(alpha) \* cos(gamma) + cos(beta) \* cos(alpha) \* sin(gamma);  }  double l2()  {  return -cos(alpha) \* sin(gamma) + cos(beta) \* sin(alpha) \* cos(gamma);  }  double m2()  {  return -sin(alpha) \* sin(gamma) + cos(beta) \* cos(alpha) \* cos(gamma);  }  double l3()  {  return sin(beta) \* sin(alpha);  }  double m3()  {  return -sin(beta) \* cos(alpha);  }  void clear()  {  for (int i = 0; i < ChartGraphic.Series.Count; i++) ChartGraphic.Series[i].Points.Clear();  }  void drawxyz()  {  double L = N / 2; //длина оси  //z  ChartGraphic.Series[2 \* N \* 3].Points.AddXY(0, 0);  ChartGraphic.Series[2 \* N \* 3].Points.AddXY(l3() \* L, m3() \* L);  ChartGraphic.Series[2 \* N \* 3].Points[1].Label = "Z";  //x  ChartGraphic.Series[2 \* N \* 3 + 1].Points.AddXY(0, 0);  ChartGraphic.Series[2 \* N \* 3 + 1].Points.AddXY(l1() \* L, m1() \* L);  ChartGraphic.Series[2 \* N \* 3 + 1].Points[1].Label = "X";  //y  ChartGraphic.Series[2 \* N \* 3 + 2].Points.AddXY(0, 0);  ChartGraphic.Series[2 \* N \* 3 + 2].Points.AddXY(l2() \* L, m2() \* L);  ChartGraphic.Series[2 \* N \* 3 + 2].Points[1].Label = "Y";  }  #endregion // +++++++  #region buttons,mouse move,controls  private void ContourIines\_KeyDown(object sender, KeyEventArgs e)  {  if (e.KeyCode == Keys.Enter)  {  int tZ = Z;  if (!int.TryParse(ContourIines.Text, out tZ))  {  ContourIines.Text = Z.ToString();  return;  }  if (tZ < 1 || tZ > 19)  {  ContourIines.Text = Z.ToString();  return;  }  Z = tZ;  ChartGraphic.Series.Clear();  init();  }  } // ------  private void ButtonColour\_Click(object sender, EventArgs e)  {  }  private void ButtonMinus\_Click(object sender, EventArgs e)  {  if (N < 51)  {  N += 2;  Z = N / 3;  ContourIines.Text = Z.ToString();  ChartGraphic.Series.Clear();  update();  init();  }  } // ------  private void ButtonPlus\_Click(object sender, EventArgs e)  {  if (N > 7)  {  N -= 2;  Z = N / 3;  ContourIines.Text = Z.ToString();  ChartGraphic.Series.Clear();  update();  init();  }  } // ------  private void ButtonApplication\_Click(object sender, EventArgs e)  {  ChartGraphic.Series.Clear();  update();  init();  } // ------  private void ChartGraphic\_MouseDown(object sender, MouseEventArgs e)  {  if (e.Button == MouseButtons.Left)  {  onmove = true;  startpos = e.Location;  }  } // ------  private void ChartGraphic\_MouseMove(object sender, MouseEventArgs e)  {  if (onmove)  {  if ((startpos.Y - e.Y) < 0) beta--;  if ((startpos.Y - e.Y) > 0) beta++;  if ((startpos.X - e.X) < 0) gamma--;  if ((startpos.X - e.X) > 0) gamma++;  if (beta > 359) beta = 0;  if (gamma > 359) gamma = 0;  if (beta < 0) beta = 359;  if (gamma < 0) gamma = 359;  drawscene();  }  } // ------  private void ChartGraphic\_MouseUp(object sender, MouseEventArgs e)  {  if (e.Button == MouseButtons.Left) onmove = false;  } // ------  private void ComboBoxDesignation\_SelectedIndexChanged(object sender, EventArgs e)  {  }  private void ComboBoxlines\_SelectedIndexChanged(object sender, EventArgs e)  {  for (int i = z0; i < z0 + Z \* 3; i++) ChartGraphic.Series[i].ChartType = SeriesChartType.Spline;  drawscene();  } // ------  #endregion  }  } |

ПРИЛОЖЕНИЕ 2. Код файла MarchingSquare.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Isoline  {  internal class MarchingSquare  {  bool border = false;  int Z;  int N;  double[,] fields;  int startx = 0, starty = 0;  List<PointF> permission;  enum dir  {  None,  Up,  Left,  Down,  Right  }  dir prevStep;  dir nextStep;  public MarchingSquare(int n)  {  N = n;  }  void findstartpos()  {  for (int y = 0; y < N; y++)  for (int x = 0; x < N; x++)  if (fields[x, y] < Z)  {  startx = x;  starty = y;  return;  }  }  bool check(int x, int y)  {  if (x == N - 1 || y == N - 1 || x == 0 || y == 0) border = true;  if (x < 0 || y < 0 || x >= N || y >= N) return false;  if (fields[x, y] < Z) return true;  return false;  }  void step(int x, int y)  {  bool ul = check(x - 1, y - 1);  bool ur = check(x, y - 1);  bool dl = check(x - 1, y);  bool dr = check(x, y);  prevStep = nextStep;  int state = 0;  if (ul) state |= 1;  if (ur) state |= 2;  if (dl) state |= 4;  if (dr) state |= 8;  switch (state)  {  case 1: nextStep = dir.Down; break;  case 2: nextStep = dir.Right; break;  case 3: nextStep = dir.Right; break;  case 4: nextStep = dir.Left; break;  case 5: nextStep = dir.Down; break;  case 6:  if (prevStep == dir.Down)  {  nextStep = dir.Left;  }  else  {  nextStep = dir.Right;  }  break;  case 7: nextStep = dir.Right; break;  case 8: nextStep = dir.Up; break;  case 9:  if (prevStep == dir.Right)  {  nextStep = dir.Down;  }  else  {  nextStep = dir.Up;  }  break;  case 10: nextStep = dir.Up; break;  case 11: nextStep = dir.Up; break;  case 12: nextStep = dir.Left; break;  case 13: nextStep = dir.Down; break;  case 14: nextStep = dir.Left; break;  default:  nextStep = dir.None;  break;  }  }  public PointF[] get(double[,] a, int z, bool ip)  {  border = false;  fields = a;  Z = z;  startx = starty = 0;  nextStep = prevStep = dir.Left;  permission = new List<PointF>();  bool first = true;  PointF dp1 = new PointF();  double x0 = -N / 2;  double y0 = -N / 2;  findstartpos();  int x = startx;  int y = starty;  do  {  step(x, y);  if (x > 0 && x < N && y > 0 && y < N)  {  int dx = 0, dy = 0;  switch (prevStep)  {  case dir.Down: dy = 1; break;  case dir.Left: dx = 1; break;  case dir.Up: dy = -1; break;  case dir.Right: dx = -1; break;  default: break;  }  double X = x0 + x;  double Y = y0 + y;  if (first)  {  double tx = X, ty = Y;  if (ip) ty = y0 + y + (Z - a[x, y - 1]) / (a[x, y] - a[x, y - 1]) - 1;  dp1 = new PointF((float)tx, (float)ty);  first = false;  }  if (ip) //ip - interpolation  {  if (dx != 0 && prevStep == nextStep) Y = y0 + y + (Z - a[x, y - 1]) / (a[x, y] - a[x, y - 1]) - 1;  if (dy != 0 && prevStep == nextStep) X = x0 + x + (Z - a[x - 1, y]) / (a[x, y] - a[x - 1, y]) - 1;  if (nextStep == dir.Down && prevStep == dir.Left) Y = y0 + y + (Z - a[x, y - 1]) / (a[x, y] - a[x, y - 1]) - 1;  if (nextStep == dir.Down && prevStep == dir.Right) X = x0 + x + (Z - a[x - 1, y]) / (a[x, y] - a[x - 1, y]) - 1;  if (nextStep == dir.Left && prevStep == dir.Down) X = x0 + x + (Z - a[x - 1, y]) / (a[x, y] - a[x - 1, y]) - 1;  if (nextStep == dir.Left && prevStep == dir.Up) X = x0 + x + (Z - a[x - 1, y]) / (a[x, y] - a[x - 1, y]) - 1;  if (nextStep == dir.Up && prevStep == dir.Right) X = x0 + x + (Z - a[x - 1, y]) / (a[x, y] - a[x - 1, y]) - 1;  if (nextStep == dir.Up && prevStep == dir.Left) X = x0 + x + (Z - a[x - 1, y]) / (a[x, y] - a[x - 1, y]) - 1;  if (nextStep == dir.Right && prevStep == dir.Up) Y = y0 + y + (Z - a[x, y - 1]) / (a[x, y] - a[x, y - 1]) - 1;  if (nextStep == dir.Right && prevStep == dir.Down) X = x0 + x + (Z - a[x - 1, y]) / (a[x, y] - a[x - 1, y]) - 1;  if (!(nextStep == dir.Down && prevStep == dir.Right) && !(nextStep == dir.Left && prevStep == dir.Up))  permission.Add(new PointF((float)X, (float)Y));  }  else permission.Add(new PointF((float)X, (float)Y));  }  switch (nextStep)  {  case dir.Down: y--; break;  case dir.Left: x--; break;  case dir.Up: y++; break;  case dir.Right: x++; break;  default: break;  }  } while (x != startx || y != starty);  if (!border)  {  if (!first) permission.Add(dp1);  if (permission.Capacity > 0) permission.Add(permission[0]);  }  return permission.ToArray();  }  }  } |