# Лабораторная работа № 4 Работа с 2D графикой

**Цель лабораторной работы**

Получение навыков работы с итерационными процессами и закрепление навыков разработки методов и работы с 2D графикой на языке программирования C#.

**Постановка задачи**

**Работа с 2D графикой.**

Класс для работы с таймером Timer формирует в приложении повторяющиеся события. События повторяются с периодичностью, указанной в миллисекундах в свойстве Interval. Установка свойства Enabled в значение true запускает таймер.

Для форм в C# предусмотрен способ, позволяющий приложению при необходимости перерисовывать окно формы в любой момент времени. Когда вся клиентская область окна формы или часть этой области требует перерисовки, форме передается событие Paint. Все, что требуется от программиста, - это создать обработчик данного события, наполнив его необходимой функциональностью.

Для рисования линий и фигур, отображения текста, вывода изображений и т.д. нужно использовать объект Graphics. Этот объект предоставляет поверхность рисования и используется для создания графических изображения. Ниже представлены два этапа работы с графикой.

Создание или получение объекта Graphics.

Рисовать возможно на любом компоненте, но быстрее всего происходит отрисовка на **pictureBox-**е.

Добавим на форму pictureBox и установим свойство Anchor: Top, Left, Right, Bottom (рисунки 1 и 2).

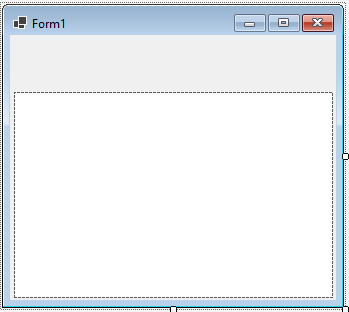


Рисунок 1 – Конструктор формы с PictureBox

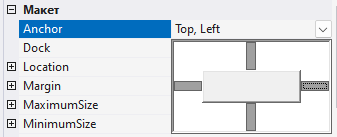


Рисунок 2 – Свойство Anchor у PictureBox

Рисование необходимо проводить в обработчике события Paint. Нарисуем круг радиуса 100 в середине полотна для рисования (рисунок 3).

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для рисования необходимо создать кисть Pen.

var pen = new Pen(Color.Red, 2);

int width = 100, height = 100;

var cx = (pictureBox.Width - width) / 2;

var cy = (pictureBox.Height - height) / 2;

e.Graphics.DrawEllipse(

pen, cx, cy, width, height);

}

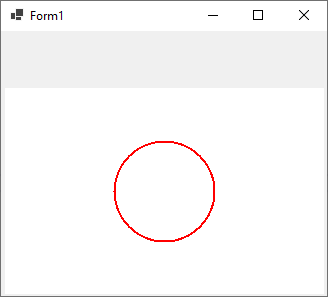


Рисунок 3 – Рисование круга в PictureBox

Нарисуем закрашенный прямоугольник шириной 100 и высотой 50 (рисунок 4). Здесь для рисования используется кисть и метод начинается не с Draw… а с Fill… (FillRectangle).

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для закрашенных фигур необходима кисть

var brush = new SolidBrush(Color.Red);

int width = 100, height = 50;

var cx = (pictureBox.Width - width) / 2;

var cy = (pictureBox.Height - height) / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

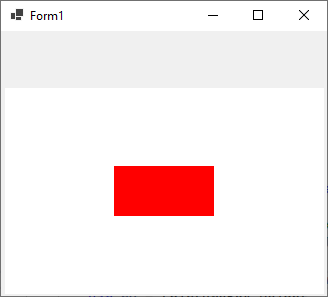


Рисунок 4 – Рисование закрашенного прямоугольника  
в PictureBox

Изменим код чтобы прямоугольник рисовался там, где была нажата ЛКМ (рисунок 5).

public partial class Form1 : Form

{

private int mouseX, mouseY; // Координаты мыши

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

// Для закрашенных фигур необходима кисть

var brush = new SolidBrush(Color.Red);

int width = 100, height = 50;

var cx = mouseX - width / 2;

var cy = mouseY - height / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

private void pictureBox\_MouseClick(object sender, MouseEventArgs e)

{

mouseX = e.X; // Считываем координаты мыши

mouseY = e.Y;

pictureBox.Refresh(); // Для вызова отрисовки

}

}

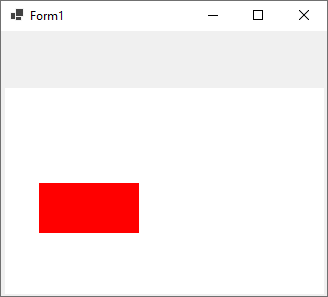


Рисунок 5 – Рисование закрашенного прямоугольника в PictureBox в точке нажатия левой кнопки мыши

Напишем код для рисования линий (рисунок 6):

public partial class Form1 : Form

{

private List<Point> points = new(); // Координаты точек

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

if (points.Count < 2) // Недостаточно точек

return;

var pen = new Pen(Color.Red, 2); e.Graphics.DrawLines(pen, points.ToArray());

}

private void pictureBox\_MouseClick(object sender, MouseEventArgs e)

{

points.Add(new(e.X, e.Y)); // Добавляем точку

pictureBox.Refresh();

}

}

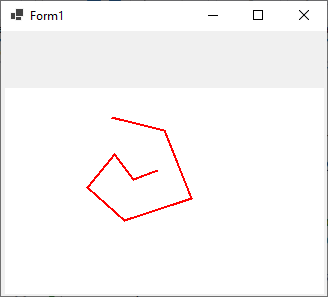


Рисунок 6 – Рисунок последовательности линий

Добавим возможность **выбора цвета** с помощью компонента ColorDialog:

private Color color = Color.Red; private void buttonColorDialog\_Click(object sender, EventArgs e)

{

var cd = new ColorDialog(); // Выбор цвета

if (cd.ShowDialog() == DialogResult.OK)

color = cd.Color;

}

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

if (points.Count < 2)

return;

var pen = new Pen(color, 2);

e.Graphics.DrawLines(pen, points.ToArray());

}

После нажатия на кнопку «Выбрать цвет» появляется диалоговое окно с палитрой (рисунок 7).

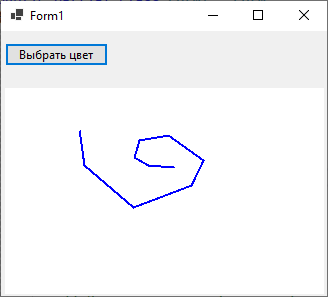
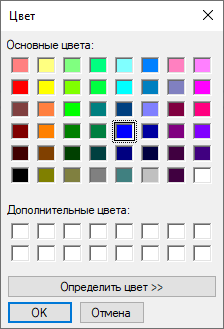
 

Рисунок 7 – Изменения цвета при рисовании

Имена большого количества методов, определенных в классе Graphics, начинаются с префикса Draw\* и Fill\*. Первые из них предназначены для рисования текста, линий и незакрашенных фигур (таких, как прямоугольные рамки), а вторые – для рисования закрашенных геометрических фигур.

Метод DrawLine рисует линию, соединяющую две точки с заданными координатами. У метода есть несколько перегруженных версий:

public void DrawLine(Pen, Point, Point);

public void DrawLine(Pen, PointF, PointF);

public void DrawLine(Pen, int, int, int, int);

public void DrawLine(Pen, float, float, float, float);

Первый параметр задает инструмент для рисования линии – перо. Перья создаются как объекты класса Pen, например:

Pen p = new Pen(Brushes.Black, 2);

Здесь создается черное перо толщиной 2 пикселя. При создании пера можно выбрать его цвет, толщину и тип линии, а также другие атрибуты.

Остальные параметры перегруженных методов DrawLine задают координаты соединяемых точек. Эти координаты могут быть заданы как объекты класса Point и PointF, а также в виде целых числе и числе с плавающей десятичной точкой.

В классах Point и PointF определены свойства X и Y, задающие, соответственно, координаты точки по горизонтальной и горизонтальной оси. При этом в классе Point эти свойства имеют целочисленные значения, а в классе PointF – значения с плавающей десятичной точкой.

Третий и четвертый варианты метода DrawLine позволяют задавать координаты соединяемых точек в виде двух пар чисел. Первая пара определяет координаты первой точки по горизонтальной и вертикальной оси, а вторая – координаты второй точки по этим же осям. Разница между третьим и четвертым методом заключается в использовании координат различных типов.

Метод DrawEllipse рисует эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров. При помощи метода DrawArc программа может нарисовать сегмент эллипса. Сегмент задается при помощи координат прямоугольной области, в которую вписан эллипс, а также двух углов, отсчитываемых в направлении против часовой стрелки. Первый угол Angle1 задает расположение одного конца сегмента, а второй Angle2 – расположение другого конца сегмента (рисунок 8).

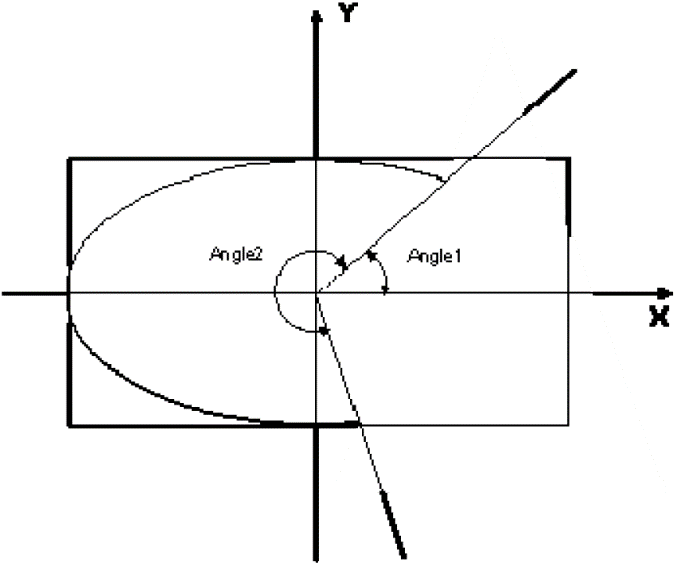


Рисунок 8 – Углы и прямоугольник, задающие сегмент эллипса

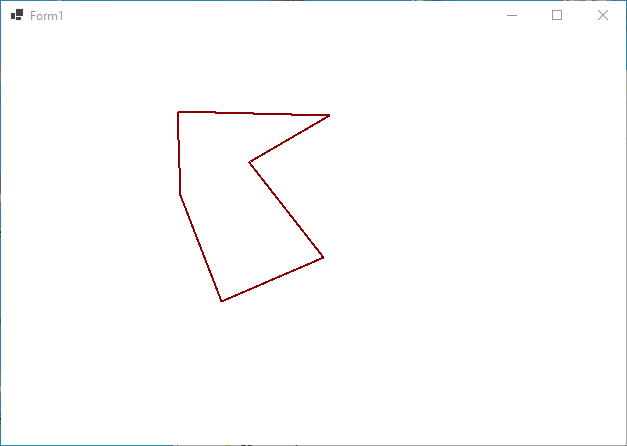


Рисунок 9 – Нарисованный многоугольник

**Создание анимации.**

Для создания простой анимации достаточно использовать таймер, при тике которого будут изменяться параметры изображения (например, координаты концов отрезка) и вызываться обработчик события Paint для рисования по новым параметрам. При таком подходе не надо заботиться об удалении старого изображения, ведь оно создается в окне заново.

Добавим таймер на форму, установим интервал 10 мс. И сразу включим **Enabled** = True. Определим основные переменные, такие как: скорость, размеры и т.д. При срабатывании таймера изменяем координаты прямоугольника, выполняем проверку на выход за границы области и рисуем.

public partial class Form1 : Form

{

private Color color = Color.Red;

private Point rectPoint = new Point(100, 100);

private int width = 100, height = 50;

private int speedX = 5, speedY = 5;

private void pictureBox\_Paint(object sender, PaintEventArgs e)

{

var brush = new SolidBrush(color);

var cx = rectPoint.X - width / 2;

var cy = rectPoint.Y - height / 2;

e.Graphics.FillRectangle(

brush, cx, cy, width, height);

}

private void timer\_Tick(object sender, EventArgs e)

{

rectPoint.X += speedX;

rectPoint.Y += speedY;

if (rectPoint.X <= 0 || rectPoint.X >= pictureBox.Width)

speedX \*= -1;

if (rectPoint.Y <= 0 || rectPoint.Y >= pictureBox.Height)

speedY \*= -1;

// Для вызова отрисовки

pictureBox.Refresh();

}

}

В результате на экране увидим двигающийся и отскакивающий от границ прямоугольник.

Движение по траектории реализуется аналогично. Для реализации движения по прямой нужно увеличивать переменные, являющиеся узловыми точками, на определенные константы: в приведенном выше примере это переменные x2 и y2. Для задания более сложной траектории можно использовать различные параметрические кривые.

В случае движения на плоскости обычно изменению подвергается один параметр. Рассмотрим пример реализации движения окружности по декартову листу. Декартов лист (рисунок 10) – это плоская кривая третьего порядка, удовлетворяющая уравнению в прямоугольной системе . Параметр 3*a* определяется как диагональ квадрата, сторона которого равна наибольшей хорде петли.

При переходе к параметрическому виду получаем:



где .

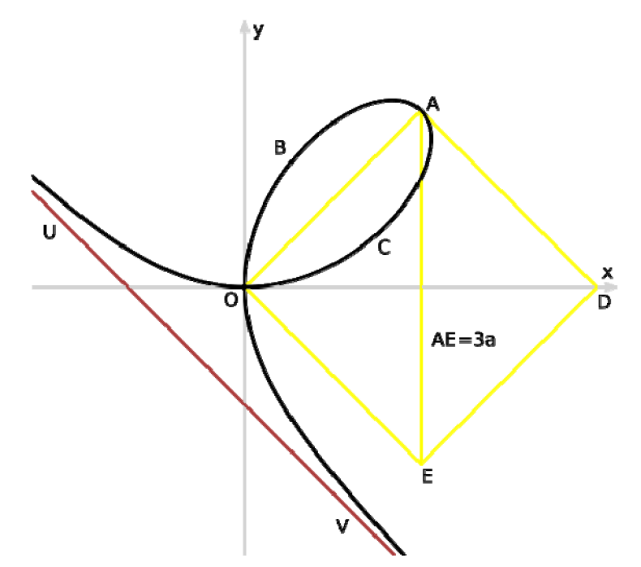


Рисунок 10 – Декартов лист

**Фракталы.**

Фрактал – геометрическая фигура, обладающая свойством самоподобия, согласно которому часть фигуры подобна ей целой. Выделяют следующие типы фракталов.

1) Геометрические фракталы – их получают с помощью ломаной (2D) или поверхности (3D), называемой генератором. За один шаг алгоритма каждый из отрезков, составляющих ломаную, заменяется на ломаную-генератор в соответствующем масштабе. В результате бесконечного повторения этой процедуры получается геометрический фрактал.

2) Алгебраические фракталы – строятся на основе алгебраических формул.

3) Стохастические фракталы – получаются, если в итерационном процессе хаотически менять какие-либо его параметры.

Рассмотрим алгебраические фракталы. Их получают при исследовании нелинейных динамических систем (отсюда другое название – динамические). Поведение такой системы можно описать комплексной нелинейной функцией – многочленом вида

.

В частном случае данная функция принимает вид

 или .

Возьмем какую-нибудь начальную точку  на комплексной плоскости. Теперь рассмотрим бесконечную последовательность чисел на комплексной плоскости, каждое следующее из которых получается из предыдущего:

.

В зависимости от начальной точки  такая последовательность может вести себя по-разному: стремиться к бесконечности при ; сходиться к какой-то конечной точке; циклически принимать ряд фиксированных значений; возможны и более сложные варианты.

1. Фрактал Мандельброта.

Рассмотрим последовательность комплексных чисел:

.

Множество точек , для которого эта последовательность не расходится, называется множеством *Мандельброта*. Для построения его графической интерпретации нужно определить исходные данные:

* Прямоугольное окно *w* x *h* (*w* – ширина окна, *h*- высота окна);
* Значение  – минимальный радиус расходимости множества Мандельброта.
* Максимальное число итераций .

Если точка  вышла за пределы круга радиуса при , то процесс вычисления останавливается.

Используя замену , разложим  на действительную и мнимую части:



Таким образом, чтобы построить фрактал, необходимо для каждой точки  запустить следующий итерационный процесс:



где и .

Составим матрицу M, элементы которой  равны номерам итераций, на которых процесс был остановлен. Далее матрицу можно вывести на экран как растровое изображение, предварительно сопоставив каждому числу из интервала  некоторый цвет. Пример визуализации множества Мандельброта приведен на рисунке 11.

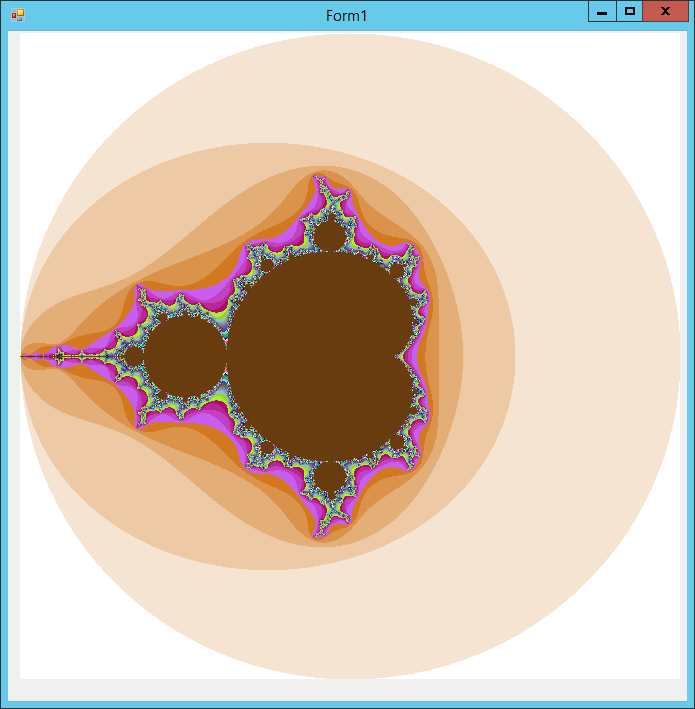


Рисунок 11 – Множество Мандельброта

2. Фрактал Жюлиа.

Рассмотрим ту же последовательность комплексных чисел, что и для множества Мандельброта



Множеством *Жюлиа* полинома  называется такое подмножество множества комплексных чисел, для каждой точки которого поведение функции под действием итераций является хаотичным, т.е. небольшие изменения в начальных условиях в некоторой небольшой окрестности начальной точки, значительно влияют на траекторию.

Исходные данные, этапы построения и условия остановки – те же, что и для множества Мандельброта, за исключением:

* значение  фиксируется, например: ;
* начальное значение  перебирается дискретно в области .

Рассматривая множество в общем виде  и изменяя  и , можно получать разнообразные фракталы.

На рисунке 12 представлен пример отображения фрактала Жюлиа при , .

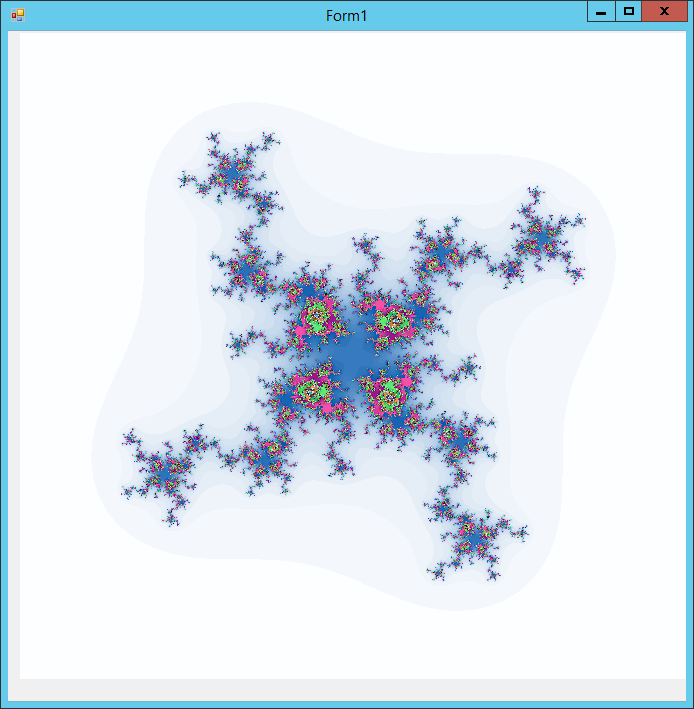


Рисунок 12 – Фрактал Жюлиа

3. Бассейны Ньютона.

Для построения, фракталов, называемых *бассейнами Ньютона*, используется метод приближенных итераций нахождения корней нелинейного уравнения алгоритмом Ньютона на комплексной плоскости.

Рассмотрим уравнение:

.

Оно имеет три корня. При выборе различных  процесс будет сходиться к различным корням (областям притяжения). Границы этих областей имеют фрактальную структуру.

Общая формула метода Ньютона имеет вид:

.

Подставив  в формулу метода, получим итерационную формулу для построения фрактала:

.

Итерационный процесс останавливается при:

.

Для построения графической интерпретации, так же, как и для фрактала Мандельброта, используется матрица, элементы которой равны номеру итерации, на которой остановился процесс (рисунок 13).

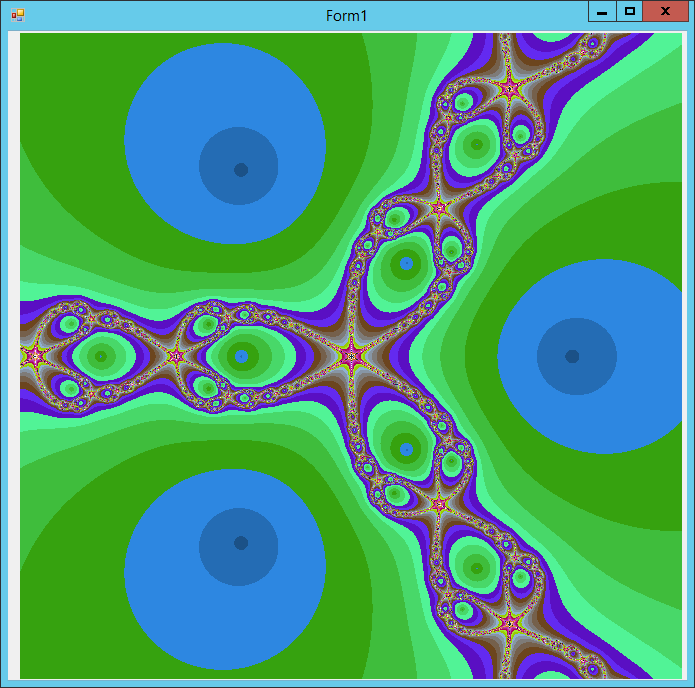


Рисунок 13 – Бассейн Ньютона

Псевдокод рисования фрактала:

Цикл i, j по пикселям в PictureBox:

x, y = Преобразование индексов в координаты на комплексной плоскости(i, j)

iter = Итерационный процесс до условия остановки(x, y)

r, g, b = Преобразование числа итераций в цвет(iter)

Раскраска пикселя(i, j, r, g, b)

Псевдокод метода:

Итерационный процесс до условия остановки(x, y)

iter = 0

Цикл пока (x\*x+y\*y<4 И iter < max\_iter)

x, y = комплексная\_функция(x, y)

iter = iter + 1

Вернуть iter

**Рисование в PictureBox по пикселям**

Для рисования фракталов необходимо закрашивать каждый пиксель по отдельности. Для этого необходимо добавить элемент **PictureBox** на форму (рисунок 14).

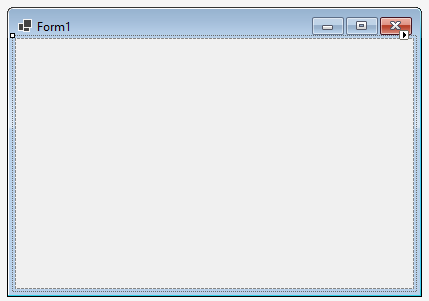


Рисунок 14 – Пустая форма с PictureBox для рисования

Далее в качестве примера рассмотрим градиентное закрашивание окна. Код программы выглядит следующим образом:

private void Form1\_Load(object sender, EventArgs e)

{

pictureBox.Image = new Bitmap(

pictureBox.Width, pictureBox.Height);

// Рисование происходит на объекте Bitmap

using var bitmap = (Bitmap)pictureBox.Image;

for (var x = 0; x < bitmap.Width; x++)

{

for (var y = 0; y < bitmap.Height; y++)

{

// Цвет меняется линейно слева направо и сверху вниз

// Red меняется по y, Blue – по x

var r = (int)(255.0 \* y / bitmap.Height);

var b = (int)(255.0 \* x / bitmap.Width);

bitmap.SetPixel(x, y, Color.FromArgb(r, 0, b));

}

}

}

Результат работы программы представлен на рисунке 15.

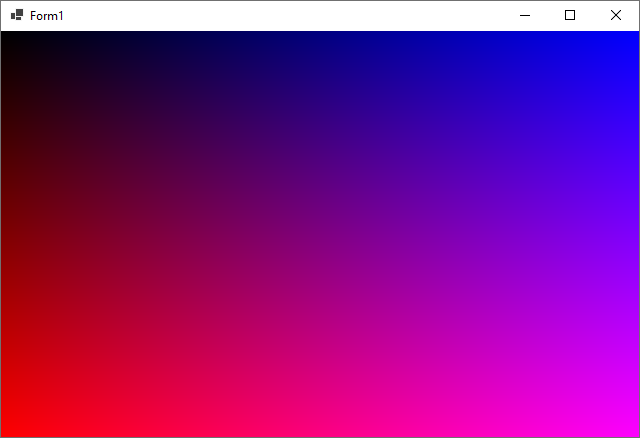


Рисунок 15 – Градиентное раскрашивание PictureBox

**Задание на лабораторную работу**

Построить алгебраический фрактал

1. Реализовать итерационный процесс построения алгебраического фрактала.

2. Реализовать следующий функционал программы:

* задание параметров фрактала;
* задание размеров комплексной области (начальный размер);
* увеличение выбранного участка фрактала. Участок определяется прямоугольной областью, выделяемой курсором мыши. Если соотношение сторон области не соответствует соотношению сторон окна рисования, то область дополняется (увеличивается) до соответствия;
* выбор палитры рисования.

3. Программа должна быть задокументирована с помощью XML комментариев.

4. **Дополнительно.** Вся вычислительная логика программы должна быть инкапсулирована в отдельном классе (или множестве классов). В классе формы должно быть только обращение к вычислительной логике для изменения состояния, пересчета и получения результатов.

5. Логика рисования должна быть в классе формы.

6. Рисование должно производиться на PictureBox.

**Варианты заданий на лабораторную работу**

1, 5, 9, 13, 17, 21, 25. Фрактал Мандельброта, , .

2, 6, 10, 14, 18, 22. Множество Жюлиа, , .

3, 7, 11, 15, 19, 23. Бассейн Ньютона, , .

4, 8, 12, 16, 20, 24. Бассейн Ньютона, , .

**Пример разработки приложения**

Построим фрактал «Бассейн Ньютона» для случая .

**Шаг 1: Подготовка проекта**

1. Создайте Windows Forms приложение
2. Добавьте на форму компонент PictureBox, который будет отображать фрактал
3. Установите его размеры и свойство Dock по необходимости

**Шаг 2: Реализация класса Complex**

Класс для работы с комплексными числами содержит:

* Основные свойства: Re (действительная часть), Im (мнимая часть), Magnitude (модуль)
* Арифметические операции: сложение, вычитание, умножение, деление
* Методы:
  + Pow - возведение в степень
  + GetConjugate - получение сопряженного числа
  + Преобразования между double и Complex
* Переопределенный ToString() для удобного вывода

public class Complex

{

    public double Re { get; set; } // Вещественная часть

    public double Im { get; set; } // Мнимая часть

    public double Magnitude => Math.Sqrt(Re \* Re + Im \* Im);

    public Complex(double re = 0, double im = 0)

    {

        Re = re;

        Im = im;

    }

    // Определяем оператор сложения для объектов типа Com-plex

    public static Complex operator +(Complex c1, Complex c2)

        => new(c1.Re + c2.Re, c1.Im + c2.Im);

    // Определяем оператор вычитания для объектов типа Complex

    public static Complex operator -(Complex c1, Complex c2)

        => new(c1.Re - c2.Re, c1.Im - c2.Im);

    // Определяем оператор сложения Complex и double

    public static Complex operator +(Complex c, double d)

        => new(c.Re + d, c.Im);

    // Операции по умолчанию не коммутативны

    public static Complex operator -(double d, Complex c)

        => new(d - c.Re, c.Im);

    public static Complex operator +(double d, Complex c)

        => c + d;

    public static Complex operator -(Complex c, double d)

        => new(c.Re - d, c.Im);

    public static Complex operator \*(Complex c, double d)

        => new(d \* c.Re, d \* c.Im);

    public static Complex operator \*(double d, Complex c)

        => c \* d;

    public static Complex operator /(Complex c, double d)

        => new(c.Re / d, c.Im / d);

    public static Complex operator \*(Complex c1, Complex c2)

        => new(c1.Re \* c2.Re - c1.Im \* c2.Im,

                        c1.Re \* c2.Im + c1.Im \* c2.Re);

    public static Complex operator /(Complex c1, Complex c2)

    {

        var conjugate = c2.GetConjugate();

        var c1New = c1 \* conjugate;

        var c2New = c2 \* conjugate;

        return c1New / c2New.Re;

    }

    // Сопряженное комплексное число

    public Complex GetConjugate()

        => new(Re, -Im);

    // Неявное преобразование вещественного числа в ком-плексное

    public static implicit operator Complex(double d)

      => new(d, 0);

    // Явное преобразование комплексного числа в веще-ственное с отбрасыванием мнимой части

    public static explicit operator double(Complex c)

      => c.Re;

    public Complex Pow(int degree)

    {

        var current = new Complex(Re, Im);

        for (int i = 1; i < degree; i++)

        {

            current \*= this;

        }

        return current;

    }

    public override string ToString() // Для красивого вывода на экран

    {

        if (Re == 0 && Im == 0)

            return "0";

        var sb = new StringBuilder();

        if (Re != 0) sb.Append(Re.ToString());

        if (Im < 0)

        {

            if (Im == -1) sb.Append($"-i");

            else sb.Append($"{Im}i");

        }

        else if (Im > 0)

        {

            if (Re != 0) sb.Append("+");

            if (Im == 1) sb.Append($"i");

            else sb.Append($"{Im}i");

        }

        return sb.ToString();

    }

}

**Шаг 3: Реализация класса Fractal**

Основной класс для расчета фрактала:

* Параметры конструктора:
  + centerX, centerY - центр области в комплексной плоскости
  + width - ширина области
  + rows, columns - размер сетки (соответствует размерам PictureBox)
* Метод Calculate() выполняет итерационный процесс:
  1. Для каждой точки комплексной плоскости (переводится из координат пикселей)
  2. Применяет метод Ньютона: zₙ₊₁ = zₙ - f(zₙ)/f'(zₙ)
  3. Подсчитывает количество итераций до достижения условия сходимости
* Функция и ее производная: f(z) = z³ - 1, f'(z) = 3z²

using System.Drawing;

using System.Numerics;

using System.Text;

namespace FractalLibrary;

public class Fractal

{

    public const double R\_MIN = 0.01;

    /// <summary>

    /// Фрактал

    /// </summary>

    /// <param name="centerX">Координата X центра комплексной области</param>

    /// <param name="centerY">Координата Y центра комплексной области</param>

    /// <param name="width">Ширина комплексной области</param>

    /// <param name="rows">Количество строк сетки</param>

    /// <param name="columns">Количество столбцов сетки</param>

    public Fractal(double centerX, double centerY, double width, int rows, int columns)

    {

        CenterX = centerX;

        CenterY = centerY;

        Width = width;

        Rows = rows;

        Columns = columns;

    }

    public int IterationsLimit { get; set; } = 100;

    // Координата X центра комплексной области

    public double CenterX { get; }

    // Координата Y центра комплексной области

    public double CenterY { get; }

    // Ширина комплексной области

    public double Width { get; }

    // Высота комплексной области, рассчитывается автома-тически пропорционально ширине

    public double Height => Width \* Rows / Columns;

    // Количество строк сетки

    public int Rows { get; }

    // Количество столбцов сетки

    public int Columns { get; }

    /// <summary>

    /// Рассчитать значения (количество итераций) в каждой ячейке

    /// </summary>

    /// <returns></returns>

    public int[,] Calculate()

    {

        var values = new int[Rows, Columns];

        for (int i = 0; i < Rows; i++)

        {

            for (int j = 0; j < Columns; j++)

            {

                var x = CenterX + Width \* j / (Columns - 1) - Width / 2;

                var y = CenterY + Height \* i / (Rows - 1) - Height / 2;

                var z = new Complex(x, y);

                var iterations = 0;

                while (z.Pow(3).Magnitude > R\_MIN && iter-ations <= IterationsLimit)

                {

                    z -= f(z) / df(z);

                    iterations++;

                }

                values[i, j] = iterations;

            }

        }

        return values;

    }

    private Complex f(Complex z) => z.Pow(3) - 1;

    private Complex df(Complex z) => 3 \* z.Pow(2);

}

**Шаг 4: Реализация визуализации**

Интерфейс IPainter и класс GrayScalePainter

* Преобразует количество итераций в цвета
* GrayScalePainter создает градиент от черного к белому
* Нормализует значения итераций относительно минимального и максимального значений

/// <summary>

/// Класс для перевода сеточных значений ячеек в цвета

/// </summary>

public interface IPainter

{

    public Color[,] Paint(int[,] gridValues);

}

/// <summary>

/// Класс для перевода сеточных значений ячеек в ЧБ цвета

/// </summary>

public class GrayScalePainter : IPainter

{

    public Color[,] Paint(int[,] gridValues)

    {

        var min = gridValues.Min();

        var max = gridValues.Max();

        var rows = gridValues.GetLength(0);

        var columns = gridValues.GetLength(1);

        var colors = new Color[rows, columns];

        for (int i = 0; i < rows; i++)

        {

            for (int j = 0; j < columns; j++)

            {

                var norm = (gridValues[i, j] - min) / (max - min);

                var colorValue = (int)(255.0 \* norm);

                colors[i, j] = Color.FromArgb(255, color-Value, colorValue, colorValue);

            }

        }

        return colors;

    }

}

Запишем класс-расширение для поиска минимума и максимума в массиве.

public static class ArrayExtensions

{

    public static T Min<T>(this T[,] values) where T : INumber<T>

    {

        var rows = values.GetLength(0);

        var columns = values.GetLength(1);

        var min = values[0, 0];

        for (int i = 0; i < rows; i++)

        {

            for (int j = 0; j < columns; j++)

            {

                if (values[i, j] < min)

                    min = values[i, j];

            }

        }

        return min;

    }

    public static T Max<T>(this T[,] values) where T : INumber<T>

    {

        var rows = values.GetLength(0);

        var columns = values.GetLength(1);

        var max = values[0, 0];

        for (int i = 0; i < rows; i++)

        {

            for (int j = 0; j < columns; j++)

            {

                if (values[i, j] > max)

                    max = values[i, j];

            }

        }

        return max;

    }

}

**Метод MainForm\_Load**

1. Создает объект Fractal с параметрами:
   * Центр в (0,0)
   * Ширина области = 4
   * Размеры сетки = размерам PictureBox
2. Вычисляет значения итераций
3. Преобразует итерации в цвета
4. Создает изображение и устанавливает его в PictureBox

Воспользуемся разработанными классами и запишем код для отображения фрактала на главной форме:

private void MainForm\_Load(object sender, EventArgs e)

{

    var fractal = new Fractal(0, 0, 4, pictureBox.Height, pictureBox.Width);

    var iterations = fractal.Calculate();

    var painter = new GrayScalePainter();

    var colors = painter.Paint(iterations);

    var bmp = new Bitmap(pictureBox.Width, picture-Box.Height);

    for (int x = 0; x < bmp.Width; x++)

    {

        for (int y = 0; y < bmp.Height; y++)

        {

            bmp.SetPixel(x, y, colors[y, x]);

        }

    }

    pictureBox.Image = bmp;

}

В результате получим изображение, показанное на рисунке 16.

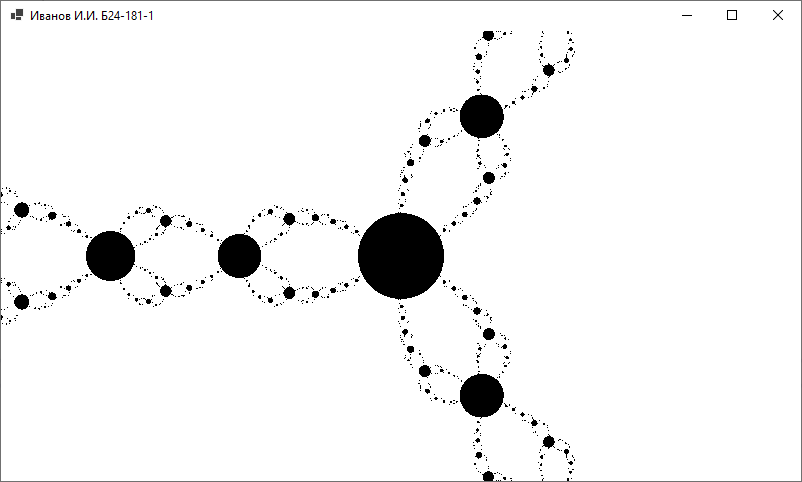


Рисунок 16 – Результат работы программы для построения фрактала Бассейн Ньютона

**Рекомендации по выполнению**

1. Разместите PictureBox на форме, установите ему подходящие размеры
2. Реализуйте классы в следующем порядке:
   * Complex
   * ArrayExtensions
   * IPainter и GrayScalePainter
   * Fractal
3. В обработчике Load формы:
   * Убедитесь, что PictureBox полностью инициализирован
   * Подберите параметры Fractal для лучшего отображения
4. Для ускорения работы можно:
   * Уменьшить размер PictureBox
   * Уменьшить IterationsLimit
   * Использовать параллельные вычисления

**Возможные улучшения**

1. Добавьте возможность масштабирования и перемещения по фракталу
2. Реализуйте цветную визуализацию (разные цвета для разных корней)
3. Добавьте элементы управления для настройки параметров:
   * Размер области
   * Лимит итераций
   * Точность сходимости (R\_MIN)

**Дополнительно.** Для построения цветного фрактала можно использовать следующий псевдокод:

//z^3-1

float2 Function (float2 z)

{

    return cpow(z, 3) - float2(1, 0); //cpow is an ex-ponential function for complex numbers

}

//3\*z^2

float2 Derivative (float2 z)

{

    return 3 \* cmul(z, z); //cmul is a function that handles multiplication of complex numbers

}

float2 roots[3] = //Roots (solutions) of the polynomial

{

    float2(1, 0),

    float2(-.5, sqrt(3)/2),

    float2(-.5, -sqrt(3)/2)

};

color colors[3] =  //Assign a color for each root

{

    red,

    green,

    blue

}

For each pixel (x, y) on the target, do:

{

    zx = scaled x coordinate of pixel (scaled to lie in the Mandelbrot X scale (-2.5, 1))

    zy = scaled y coordinate of pixel (scaled to lie in the Mandelbrot Y scale (-2, 1))

    float2 z = float2(zx, zy); //z is originally set to the pixel coordinates

    for (int iteration = 0;

         iteration < maxIteration;

         iteration++;)

    {

        z -= cdiv(Function(z), Derivative(z)); //cdiv is a function for dividing complex numbers

        float tolerance = 0.000001;

        for (int i = 0; i < roots.Length; i++)

        {

            float2 difference = z - roots[i];

            //If the current iteration is close enough to a root, color the pixel.

            if (abs(difference.x) < tolerance && abs (difference.y) < tolerance)

            {

                return colors[i]; //Return the color corresponding to the root

            }

        }

    }

    return black; //If no solution is found

}

**Контрольные вопросы**

1. Приведите особенности работы с двумерной графикой с использованием стандартных средств среды MS Visual Studio.
2. Какой метод называют рекурсивным?
3. В чем особенности фракталов?
4. Чем отличаются алгебраические фракталы от других фракталов?
5. Какие события используются при реализации увеличения выбранной области фрактала?
6. Приведите примеры применения фракталов на практике.

**Содержание пояснительной записки**

Пояснительная записка должна содержать:

1) формулировка задания;

2) алгоритм решения задачи;

3) краткое описание программы;

4) скриншоты работы программы

5) листинг программы.