Практическое занятие №5

Тема: Работа с двухмерной графикой в WPF

Общие сведения о WPF

Windows Presentation Foundation (WPF) – система для построения клиентских Windows приложений для технологии Microsoft.NET с визуально привлекательными возможностями взаимодействия с пользователем. С помощью WPF можно создавать широкий спектр как автономных, так и размещенных в браузере приложений.

В основе WPF лежит векторная система визуализации, не зависящая от разрешения и созданная с расчетом на возможности современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML, элементы управления, привязку данных, макеты, двухмерную и трехмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление.

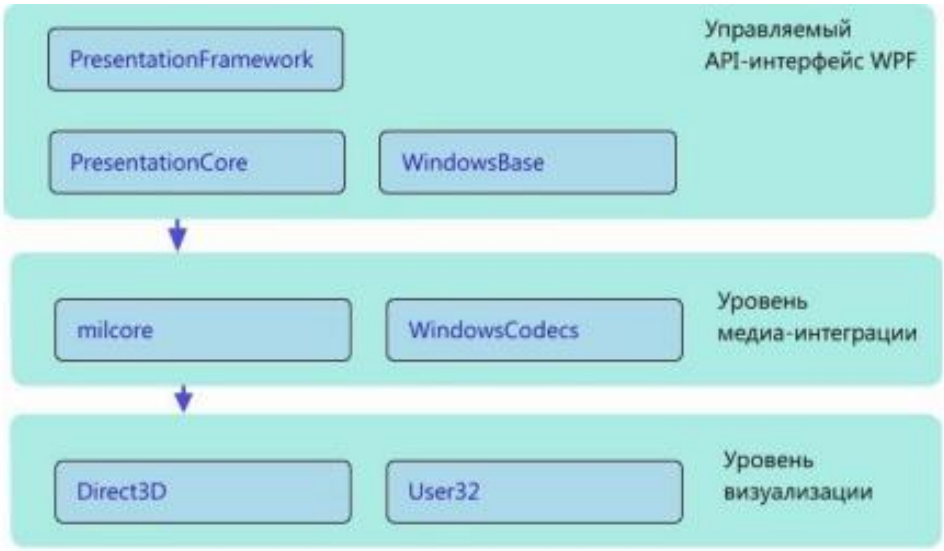
Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI/GDI+ . Производительность WPF выше, чем у GDI+ за счёт использования аппаратного ускорения графики через DirectX.

WPF обеспечивает интерфейс пользователя высокого уровня и предоставляет следующие возможности:

- веб-подобную модель компоновки, которая обеспечивает размещение и упорядочивание элементов управления по их содержимому;
- многофункциональную модель рисования на базе графических примитивов (базовых форм, текстовых блоков, графических ингредиентов);
- модель с форматированным текстом, которая обеспечивает отображение форматированного стилизованного текста в любой части пользовательского интерфейса, комбинирования текста со списками, рисунками и другими интерфейсными элементами;
- задание анимации с помощью декларативных дескрипторов;

- поддержка аудиовизуальной среды для проигрывания любых аудио- и видеофайлов;
- стили и шаблоны, которые позволяют стандартизировать форматирование и управление визуализацией элементов управления, а также повторно использовать эти решения в различных местах проекта;
- команды, которые позволяют определять их в одном месте и многократно связывать с различными элементами управления в приложении;
- декларативный пользовательский интерфейс, который позволяет описывать содержимое окон или страниц с помощью языка XAML.

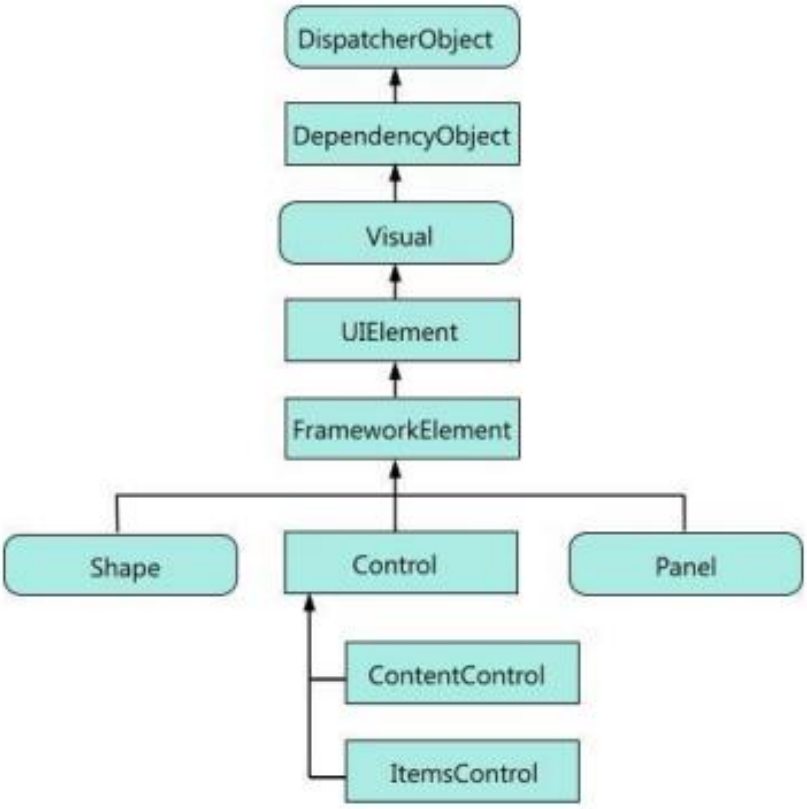
Основные компоненты WPF.



Компонент PresentationFramework содержит типы WPF верхнего уровня, включая представление окна, панелей и других элементов управления. PresentationCore содержит базовые типы, такие как UIElement и Visual, от которых порождаются все формы и элементы управления. WindowsBase включает различные типы, которые могут использоваться за пределами WPF, в частности, компоненты DispatchObject и DependencyObject. Компонент milcore является ядром визуализации WPF. WindowsCodecs представляет собой низкоуровневый API-интерфейс для поддержки создания изображений. Direct 3D также является низкоуровневым API-интерфейсом, через который

осуществляется визуализация всей графики в WPF. User32 используется для определения, какая программа получает тот или иной участок экрана.

Архитектура WPF определяет основные пространства имен для иерархии классов. Базовый набор элементов управления WPF определяет ключевые иерархии классов системы.



Фундаментальные классы WPF, абстрактные классы изображены овалами, а конкретные классы – прямоугольниками.

Двумерная графика в WPF

Вся двумерная графика WPF реализуется классами, производными от абстрактных классов Drawing и Shape. Классы, наследующие Drawing, являются более простыми конструкциями, чем классы, произведенные от Shape, и поэтому требуют меньших системных ресурсов.

Абстрактный класс Drawing находится в пространстве имен System.Windows.Media.Drawing и содержит общее описание того, что должно

быть нарисовано: фигура, точечный рисунок, строка текста или видео. Производные от него классы описывают конкретные типы содержимого:

1. GeometryDrawing - готовит внутри себя данные фигуры для рисования
2. ImageDrawing - готовит внутри себя данные изображения (точечный рисунок)
3. GlyphRunDrawing - готовит внутри себя текстовые данные
4. VideoDrawing - готовит внутри себя аудио- или видеофайл
5. DrawingGroup - послойно накапливает коллекцию данных предыдущих четырех объектов Drawing как один составной объект рисования для последующего вывода через класс отображения (DrawingBrush, DrawingImage или Visual)

Все эти производные классы являются запечатанными (sealed) и не могут продолжать цепочку наследования.

Нужно понимать, что перечисленные объекты содержат только описание того, что нужно нарисовать. Но чтобы воспроизвести содержимое любого из них, нужно передать этот объект в конструктор экземпляра класса-контейнера DrawingImage, а сам контейнер присоединить к свойству Source объекта-представления Image.

Применение объекта GeometryDrawing для построения графиков

Объект GeometryDrawing позволяет создать фигуру с заливкой и контуром путем совместного использования объектов Geometry, Pen и Brush, адресуемых его одноименными свойствами-ссылками, где

- объект Geometry описывает структуру самой геометрии фигуры
- объект Pen описывает контур фигуры
- объект Brush описывает заливку фигуры

Свойство-ссылка типа Geometry адресует экземпляр одного из классов, производных от абстрактного класса Geometry. Таковыми являются следующие запечатанные (sealed) классы:

Класс GeometryGroup может накапливать описание геометрических примитивов своих родственников за счет свойства-коллекции Children типа GeometryCollection, а затем передавать их как единый составной объект свойству-ссылке Geometry объекта GeometryDrawing. В дополнение к такому описанию геометрии объект GeometryDrawing добавляет свое описание свойств Pen и Brush. Затем можно передать все это содержимое в коллекцию DrawingGroup как очередной готовый слой для последующего отображения. Такой механизм определяет векторную (в отличие от точечной) графику, которую можно произвольно масштабировать без потери качества рисунка на устройстве отображения с любым разрешением.

Продемонстрируем применение описанного на примере построения графиков.

Построим два графика: sin() и cos() для одного периода.

Создайте новое решение WpfGraphics с новым проектом WpfApp1 типа WPF Application

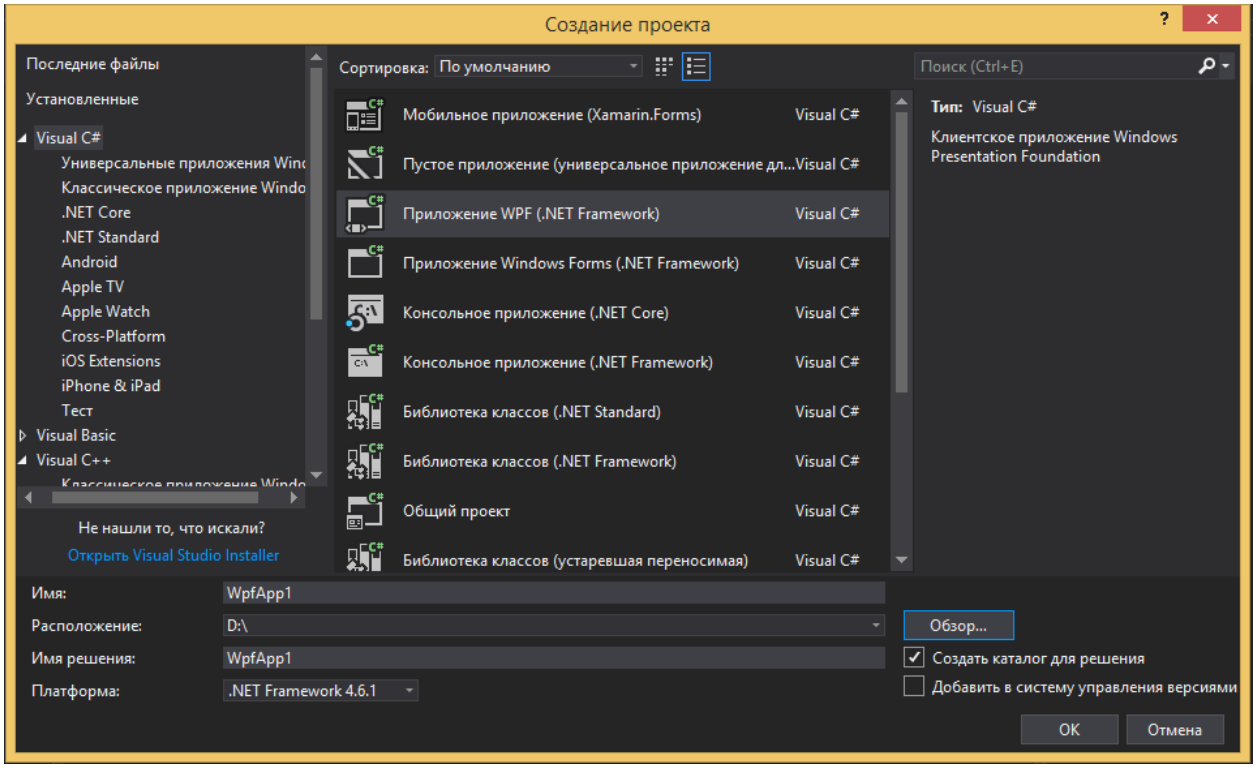


Рисунок 1 – Внешний вид окна создания проекта WPF Application

Заполните декларативную часть Window1.xaml проекта следующей разметкой:

```
<Window x:Class="WpfApp1.Window1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentat
ion"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="300" Width="300">
    <Grid Margin="20">
        <Image Name="image1" />
    </Grid>
</Window>
```

Реализуйте бизнес-логику приложения в файле Window1.xaml.cs

Инициализируем окно.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using System.Globalization;

namespace WpfApp1
{
    public partial class Window1 : Window
    {
        //
        // Поля
        //
        const int countDot = 30;// Количество отрезков
        // Список для хранения данных
        List<double[]> dataList = new List<double[]>();
        // Или можно DoubleCollection data = new DoubleCollection();
        // Контейнер слоев рисунков
        DrawingGroup drawingGroup = new DrawingGroup();

        public Window1()
        {
            InitializeComponent();

            DataFill();// Заполнение списка данными
            Execute(); // Заполнение слоев

            // Отображение на экране
            image1.Source = new DrawingImage(drawingGroup);
        }
```

Расчет функций.

```
// Генерация точек графиков
        void DataFill()
        {
            double[] sin = new double[countDot + 1];
            double[] cos = new double[countDot + 1];

            for (int i = 0; i < sin.Length; i++)
            {
                double angle = Math.PI * 2 / countDot * i;
                sin[i] = Math.Sin(angle);
                cos[i] = Math.Cos(angle);
            }

            dataList.Add(sin);
            dataList.Add(cos);
        }
```

Готовим фон и создаем сетку для графика.

```
// Фон
private void BackgroundFun()
{
    // Создаем объект для описания геометрической фигуры
    GeometryDrawing geometryDrawing = new GeometryDrawing();

    // Описываем и сохраняем геометрию квадрата
    RectangleGeometry rectGeometry = new RectangleGeometry();
    rectGeometry.Rect = new Rect(0, 0, 1, 1);
    geometryDrawing.Geometry = rectGeometry;

    // Настраиваем перо и кисть
    geometryDrawing.Pen = new Pen(Brushes.Red, 0.005);// Перо рамки
    geometryDrawing.Brush = Brushes.Beige;// Кисть закраски

    // Добавляем готовый слой в контейнер отображения
    drawingGroup.Children.Add(geometryDrawing);
}
```

```
// Горизонтальная сетка
        private void GridFun()
        {
            // Создаем коллекцию для описания геометрических фигур
            GeometryGroup geometryGroup = new GeometryGroup();

            // Создаем и добавляем в коллекцию десять параллельных линий
            for (int i = 1; i < 10; i++)
            {
                LineGeometry line = new LineGeometry(new Point(1.0, i * 0.1),
                    new Point(-0.1, i * 0.1));
                geometryGroup.Children.Add(line);
            }

            // Сохраняем описание геометрии
            GeometryDrawing geometryDrawing = new GeometryDrawing();
            geometryDrawing.Geometry = geometryGroup;

            // Настраиваем перо
            geometryDrawing.Pen = new Pen(Brushes.Gray, 0.003);
            double[] dashes = { 1, 1, 1, 1, 1 };// Образец штриха
            geometryDrawing.Pen.DashStyle = new DashStyle(dashes, -.1);

            // Настраиваем кисть
            geometryDrawing.Brush = Brushes.Beige;

            // Добавляем готовый слой в контейнер отображения
            drawingGroup.Children.Add(geometryDrawing);
        }
```

Рисуем графики функций.

```
// Строим синус линией
        private void SinFun()
        {
            // Строим описание синусоиды
            GeometryGroup geometryGroup = new GeometryGroup();
            for (int i = 0; i < dataList[0].Length - 1; i++)
            {
                LineGeometry line = new LineGeometry(
                    new Point((double)i / (double)countDot,
                        .5 - (dataList[0][i] / 2.0)),
                    new Point((double)(i + 1) / (double)countDot,
                        .5 - (dataList[0][i + 1] / 2.0)));
                geometryGroup.Children.Add(line);
            }

            // Сохраняем описание геометрии
            GeometryDrawing geometryDrawing = new GeometryDrawing();
            geometryDrawing.Geometry = geometryGroup;

            // Настраиваем перо
            geometryDrawing.Pen = new Pen(Brushes.Blue, 0.005);

            // Добавляем готовый слой в контейнер отображения
            drawingGroup.Children.Add(geometryDrawing);
        }



// Строим косинус точками
        private void CosFun()
        {
            // Строим описание косинусоиды
            GeometryGroup geometryGroup = new GeometryGroup();
            for (int i = 0; i < dataList[1].Length; i++)
            {
                EllipseGeometry ellips = new EllipseGeometry(
                    new Point((double)i / (double)countDot,
                    .5 - (dataList[1][i] / 2.0)), 0.01, 0.01);
                geometryGroup.Children.Add(ellips);
            }

            // Сохраняем описание геометрии
            GeometryDrawing geometryDrawing = new GeometryDrawing();
            geometryDrawing.Geometry = geometryGroup;

            // Настраиваем перо
            geometryDrawing.Pen = new Pen(Brushes.Green, 0.005);

            // Добавляем готовый слой в контейнер отображения
            drawingGroup.Children.Add(geometryDrawing);
        }
```

Выводим надписи на графиках.

```
// Надписи
        private void MarkerFun()
        {
            GeometryGroup geometryGroup = new GeometryGroup();
            for (int i = 0; i <= 10; i++)
            {
                FormattedText formattedText = new FormattedText(
                String.Format("{0,7:F}", 1 - i * 0.2),
                CultureInfo.InvariantCulture,
                FlowDirection.LeftToRight,
                new Typeface("Verdana"),
                0.05,
                Brushes.Black);

                formattedText.SetFontWeight(FontWeights.Bold);

                Geometry geometry = formattedText.BuildGeometry(new Point(-0.2, i * 0.1 - 0.03));
                geometryGroup.Children.Add(geometry);
            }

            GeometryDrawing geometryDrawing = new GeometryDrawing();
            geometryDrawing.Geometry = geometryGroup;

            geometryDrawing.Brush = Brushes.LightGray;
            geometryDrawing.Pen = new Pen(Brushes.Gray, 0.003);

            drawingGroup.Children.Add(geometryDrawing);
        }
```

Объединим перечисленные функции

```
// Послойное формирование рисунка в Z-последовательности
        voidExecute()
{
            // Фон
            // Мелкая сетка
            // Строим синус линией
            // Строим косинус точками
            // Надписи
}
```

и запустим приложение - результат будет таким

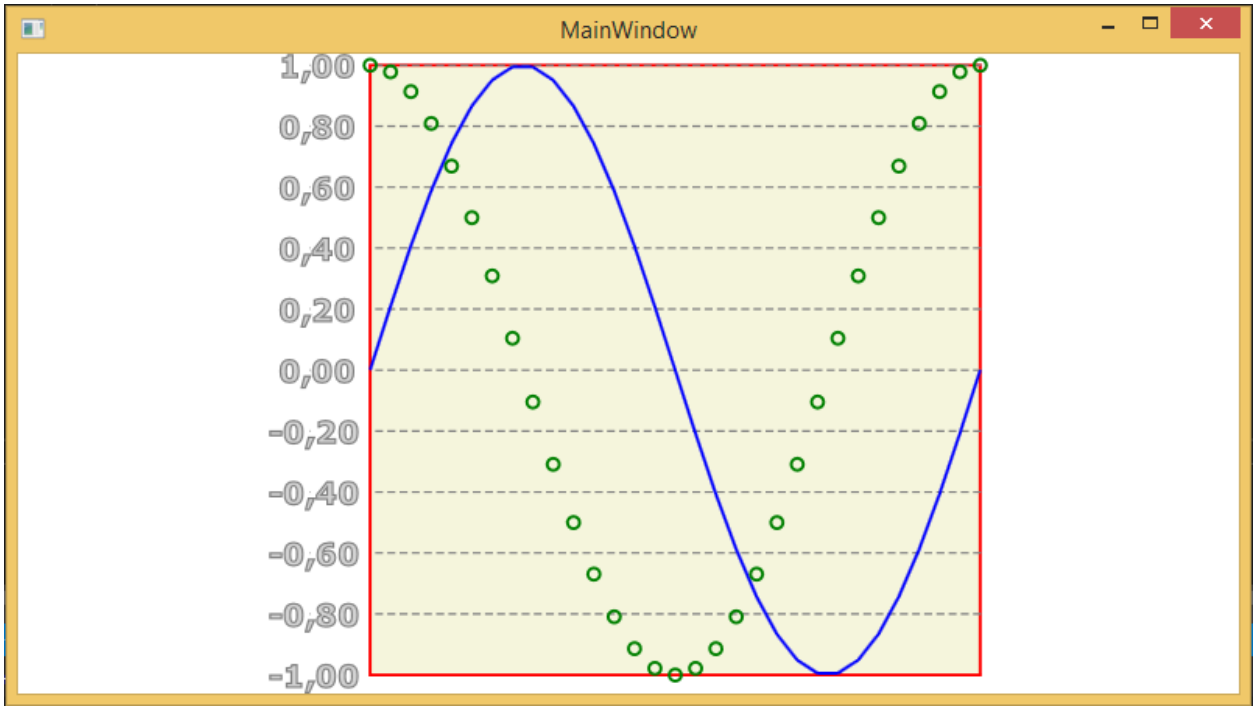


Рисунок 2 – Внешний вид окна с построенным графиком функции

Индивидуальные задания

Согласно своему варианту (таблица 5.1) создать программу для построения графика функции.

Таблица 5.1 – Индивидуальное задание

Вариант	Функция, для построения графика
1	$sin^3(2x)$
2	$\sqrt[2]{x^3}sin(x^3)$
3	$\sqrt[3]{x^3}\sin(x)$
4	$5x^2sin^3(x^3)+2x^3$
5	$x^3cos^2(x^5+2x)$
6	$3x^3sin^2(x^5)$
7	$(\sin(4x)+1)^2$
8	$sin^3(x^2)$
9	$cos^3(x^3+2)^2$

10	$sin^3(2x)$
11	$x^2cos(x)$
12	$(x-1)^3+\cos(2x^3)$
13	$\cos(x^3+2x^2-x)^2$
14	$xsin(x^2+2x)$
15	$(\cos(2x)+1)^2$
16	$sin(x^3+1)$
17	$3(sinx+x^2)$
18	$sinx^2+cos(x^2+2)$
19	$x^2-cos(x)$
20	$x^2+sin(5x)$
21	x^3+sinx^2
22	$2\sin(x-e^{-x})$
23	$sinx^2+cosx$
24	$sin^2(x)+cos^4x$
25	$sin^3(x^4)$
26	$\sin(x^3+x^2)$
27	$x^xcos(x)$
28	$x^4sin(4x)$
29	$sin^2(x^3)$
30	$(x+1)^2cos(x^3)$