

Course 2D_WPF: 2D-Computer Graphics with C# + WPF

Chapter C2: The Complete Code of the Draw Project

Copyright © by V. Miszalok, last update: 26-02-2008

- ↓ [Preliminaries](#)
- ↓ [draw1.cs](#)

Preliminaries

Guidance for **Visual Studio 2008**:

1) Main Menu after start of VS 2008: Tools → Options →
check lower left checkbox: Show all Settings → Projects and Solutions →
Visual Studio projects location: → C:\temp

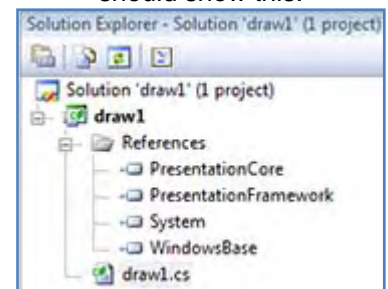
2) Main Menu after start of VS 2008: File → New Project... →
Visual Studio installed templates: Empty project
Name: draw1 → Location: C:\temp → Create directory for solution: switch off → OK.

3) In the window titled:

Solution Explorer -Solution 'draw1' (1 project) we have to
add 4 References and an draw1.cs file:

3.1 **Right-click** the branch References. A drop-down menu appears.
Click Add Reference... An Add Reference- window appears.
Scroll down until You detect the Component Names Presentation Core
and Presentation Framework and select them by Strg+click.
Continue scrolling and Strg+click two more Component Names: System
and WindowsBase. Quit the Add Reference- window with the button OK.
3.2 **Right-click** the branch **draw1**. A drop-down menu appears.
Click Add and select New Item... An Add New Item - draw1-
window appears. Select the template Code File and give it the Name:
draw1.cs. Quit the Add New Item - draw1- window with the button
Add.

The Solution Explorer
should show this:



4) Main menu of Visual Studio 2008 → Project → draw1 Properties... → Application →
Output type: Change from Console Application to Windows Application.

5) Main menu of Visual Studio 2008 → Tools → Options... → An Options-window appears.
Double-click the branch Text Editor. **Double-click** C#. **Double-click** Formatting.
Click General. Uncheck all three check boxes.
Click Indentation. Uncheck all four check boxes.
Click New Lines. Uncheck all thirteen check boxes.
Click Spacing. Uncheck all twenty three check boxes.
Click IntelliSense. Uncheck all six check boxes.
Quit the Options- window with the OK-button.

draw1.cs

Copy this code into the empty draw1.cs-file:

```
using System;
using System.Windows;
using System.Windows.Media;
using System.Windows.Controls;
using System.Windows.Shapes;
using System.Windows.Input;
using System.Windows.Threading;

public class window1 : Window
{ [STAThread] static void Main() { new Application().Run( new window1() ); }
  Canvas canvas = new Canvas();
  TextBox textBox = new TextBox();
  Polyline p = new Polyline();
  Point p0 = new Point();
  Point p1 = new Point();
  bool pressed = false;
  Point mid_of_polygon = new Point();
  Button button = new Button();
  int tickcount = 0;
  DispatcherTimer timer = new DispatcherTimer();
  public window1() //constructor
  { this.Width = this.Height = 500;
    this.Title = "draw1";
    Content = canvas;
    canvas.Children.Add( p );
    canvas.Children.Add( textBox );
    p.Stroke = Brushes.Black;
    p.StrokeThickness = 2;
    p.Points = new PointCollection();
    textBox.Text = "Press the left mouse button and move!";
    textBox.FontFamily = new FontFamily( "Courier New" );
    textBox.FontSize = 12;
    canvas.Background = new LinearGradientBrush( Colors.Red, Colors.Blue, 90 );
    button.Click += buttonOnClick;
    button.Content = "Click here to start the animation !";
    timer.Interval = TimeSpan.FromMilliseconds( 1 );
    timer.Tick += TimerOnTick;
  }
  protected override void OnMouseDown( MouseButtonEventArgs args )
  { canvas.Children.Clear(); //erase everything from the canvas
    canvas.Children.Add( p ); //except the polygon
    canvas.Children.Add( textBox ); //and the textbox
    p.Points.Clear(); //erase everything from the polygon
    p0 = args.GetPosition( canvas ); //get mouse position
    p.Points.Add( p0 ); //store mouse position
    textBox.Text = p0.X.ToString() + '/' + p0.Y.ToString();
    vertexCircle( p0 ); //mark this vertex
    pressed = true;
  }
  protected override void OnMouseMove( MouseEventArgs args )
  { if ( !pressed ) return;
    p1 = args.GetPosition( canvas ); //get mouse position
    double dx = p1.X - p0.X;
    double dy = p1.Y - p0.Y;
    if ( dx*dx + dy*dy < 400 ) return;
    p.Points.Add( p1 ); //store mouse position
    textBox.Text = p0.X.ToString() + '/' + p0.Y.ToString();
    vertexCircle( p1 ); //mark this vertex
    p0 = p1; //old end is new start
  }
}
```

```

protected override void OnMouseUp( MouseButtonEventArgs args )
{ p.Points.Add ( p.Points[0] ); //closed polygon
    mid_of_polygon          = new Point( 0, 0 );
    Point mid_of_minmax     = new Point( 0, 0 );
    Rectangle minmax_rectangle = new Rectangle();
    Rectangle mid_of_minmax_rectangle = new Rectangle();
    Ellipse mid_of_polygon_circle = new Ellipse ();
    Double perimeter = 0, area = 0;
    Double xmin, xmax, ymin, ymax;
    xmin = xmax = p0.X = p.Points[0].X;
    ymin = ymax = p0.Y = p.Points[0].Y;
    for ( int i=1; i < p.Points.Count; i++ )
    { p1 = p.Points[i];
        Double dx = p1.X - p0.X;
        Double dy = p1.Y - p0.Y;
        Double my = (p0.Y + p1.Y) / 2.0;
        perimeter += Math.Sqrt( dx*dx + dy*dy ); //Pythagoras
        area      += dx * my; //Trapezoid formula
        if ( p1.X < xmin ) xmin = p1.X; //shift the left wall to the left
        if ( p1.X > xmax ) xmax = p1.X; //shift the right wall to the right
        if ( p1.Y < ymin ) ymin = p1.Y; //shift the upper wall upward
        if ( p1.Y > ymax ) ymax = p1.Y; //shift the lower wall downward
        mid_of_polygon.X += p1.X; //sum up all x-coordinates
        mid_of_polygon.Y += p1.Y; //sum up all x-coordinates
        p0 = p1; //set the new start to the former end
    }
    mid_of_minmax.X = ( xmax + xmin ) / 2; //mid between left and right border
    mid_of_minmax.Y = ( ymax + ymin ) / 2; //mid between upper and lower border
    mid_of_polygon.X /= p.Points.Count-1; //mean of all x-coordinates
    mid_of_polygon.Y /= p.Points.Count-1; //mean of all y-coordinates
    mid_of_minmax_rectangle.Width = mid_of_minmax_rectangle.Height = 5;
    mid_of_polygon_circle.Width = mid_of_polygon_circle.Height = 5;
    minmax_rectangle.Width = xmax - xmin + 2;
    minmax_rectangle.Height = ymax - ymin + 2;
    Canvas.SetLeft( minmax_rectangle, xmin );
    Canvas.SetTop ( minmax_rectangle, ymin );
    Canvas.SetLeft( mid_of_minmax_rectangle, mid_of_minmax.X - 2 );
    Canvas.SetTop ( mid_of_minmax_rectangle, mid_of_minmax.Y - 2 );
    Canvas.SetLeft( mid_of_polygon_circle, mid_of_polygon.X - 2 );
    Canvas.SetTop ( mid_of_polygon_circle, mid_of_polygon.Y - 2 );
    canvas.Children.Add( minmax_rectangle );
    canvas.Children.Add( mid_of_minmax_rectangle );
    canvas.Children.Add( mid_of_polygon_circle );
    minmax_rectangle.Stroke = mid_of_minmax_rectangle.Stroke =
        mid_of_polygon_circle.Stroke = Brushes.Black;
    textBox.Text = String.Format( "Vertices = {0}\n", p.Points.Count-1 );
    textBox.Text += String.Format( "Perimeter = {0,2:F1}\n", perimeter );
    textBox.Text += String.Format( "Area = {0,2:F1}" , area );
    pressed = false;
    canvas.Children.Add( button );
    OnRenderSizeChanged( null );
}

void vertexCircle( Point p ) //vertex marker function
{ Ellipse elli = new Ellipse();
  elli.Width = elli.Height = 5; //diameter
  elli.Stroke = Brushes.Black;
  canvas.Children.Add( elli ); //add it to the canvas
  Canvas.SetLeft( elli, p.X - 2 ); //x-position on the canvas
  Canvas.SetTop ( elli, p.Y - 2 ); //y-position on the canvas
}

protected override void OnRenderSizeChanged( SizeChangedEventArgs sizeInfo )
{ Canvas.SetLeft( button, 0 );
  Canvas.SetTop ( button, canvas.ActualHeight-20 );
  button.Width = canvas.ActualWidth;
}

```

```
void buttonOnClick( Object sender, RoutedEventArgs rea )
{ canvas.Children.Clear();
  canvas.Children.Add( p );
  timer.Start();
}
void TimerOnTick( Object sender, EventArgs args )
{ if ( tickcount < 360 ) tickcount++;
  else { timer.Stop(); tickcount = 0; return; }
  double arcus = 2*Math.PI / 360;
  double cosinus = Math.Cos( arcus );
  double sinus = Math.Sin( arcus );
  double zoom = 1.0;
  if ( tickcount < 90 || tickcount > 270 ) zoom = 0.99;
  else zoom = 1.01;
  for ( int i=0; i < p.Points.Count; i++ )
  { Point pp = p.Points[i];
    double x = zoom * ( pp.X - mid_of_polygon.X );
    double y = zoom * ( pp.Y - mid_of_polygon.Y );
    pp.X = x*cosinus - y*sinus + mid_of_polygon.X;
    pp.Y = x*sinus + y*cosinus + mid_of_polygon.Y;
    p.Points[i] = pp;
  }
}
```