Code Samples: The Fastest Averaging Filter in C# unsafe mode

The following code is about 10 times faster than the previous `average_fast` code.
This acceleration is due to two differences.
1. The code avoids the `SetPixel(..)`-method of the `Bitmap`-class as **faster_average_filter.htm** does.
2. The code avoids the `[y,x]`-access to the pixels inside the two dimensional arrays `Byte[,] R0,G0,B0` and `Int32[,] R,G,B` and replaces it by an access via Byte-pointers `pR0,pG0,pB0`
and integer-pointers `pR,pG,pB`.

The price is a code compiled with the `unsafe` flag, which is obligatory when a code uses a pointer.
Therefore the operating system warns any user at execution of this code.
The code of this project is identical to the previous `faster_averaging`-code except
the complete `private void fastest_average_filter()`-function.
The algorithm behaves in the same manner as the previous did. It accepts any rectangular filter sizes `M≠N` or `M==N` where `M, N` must be positive odd integers `M < b0.Width, N < b0.Height`.
It works with any image format that can be read and written with the `GetPixel(..), SetPixel(..)` methods of the `Bitmap`-class.
Adjust the desired blur effect by choosing two `const`-values in lines 9 - 10.
Blur increases with the filter sizes `M` and `N`.
Create a new Windows-project "`fastest_average_filter`". Set the `unsafe`-flags as follows:
Main menu of Visual Studio → `Project` → `fastest_average_filter Properties` → `Build` → check: `Allow unsafe code` → `Debug` → check: `Enable unmanaged code debugging`. Go back to `Form1.cs`.

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;

public class Form1 : Form
{ static void Main() { Application.Run( new Form1() );  }
  Bitmap b0, b1;
  const Int32 M = 21, Mh = M/2;          //M must be an odd integer < b0.Width
  const Int32 N = 21, Nh = N/2, MN=M*N;  //N must be an odd integer < b0.Height
  Byte [,] R0, G0, B0;
  String s;

  public Form1()
  { MenuItem miRead = new MenuItem( "&Read", new EventHandler( MenuFileRead ) );
    MenuItem miExit = new MenuItem( "&Exit", new EventHandler( MenuFileExit ) );
    MenuItem miFile = new MenuItem( "&File", new MenuItem[] { miRead, miExit } );
    Menu = new System.Windows.Forms.MainMenu( new MenuItem[] { miFile } );
    Text = "Fastest Average Filter";
    SetStyle( ControlStyles.ResizeRedraw, true );
    try { b0 = new Bitmap( typeof( Form1 ), "fastest_average_filter.Butterfly.jpg" );
          byte_arrays_and_b1_image();
          fastest_average_filter();
        } catch {}
    Width  = 800;
    Height = 600;
  }

  void MenuFileRead( object obj, EventArgs ea )
  { OpenFileDialog dlg = new OpenFileDialog();
    if ( dlg.ShowDialog() != DialogResult.OK ) return;
    b0 = (Bitmap)Image.FromFile( dlg.FileName );
    byte_arrays_and_b1_image();
    fastest_average_filter();
    Invalidate();
  }

  void MenuFileExit( object obj, EventArgs ea )
  { Application.Exit(); }
```

```
protected override void OnPaint( PaintEventArgs e )
{ Graphics g = e.Graphics;
  g.Clear( BackColor );
  try
  { g.DrawImage( b0, 0, 0, b0.Width, b0.Height );
    g.DrawImage( b1, b0.Width+10, 0 );
    g.DrawString( s, new Font( "Arial", 16 ), new SolidBrush( Color.Red ),
                  0, ClientRectangle.Height-120 );
  } catch{}
}

private void byte_arrays_and_b1_image()
{ R0 = new Byte [b0.Height, b0.Width];
  G0 = new Byte [b0.Height, b0.Width];
  B0 = new Byte [b0.Height, b0.Width];
  if ( b1 != null ) b1.Dispose();
  b1 = new Bitmap( b0 );
  for ( int y=0; y < b0.Height; y++ )
    for ( int x=0; x < b0.Width; x++ )
    { Color c = b0.GetPixel( x, y );
      R0[y,x] = (Byte)c.R;
      G0[y,x] = (Byte)c.G;
      B0[y,x] = (Byte)c.B;
      b1.SetPixel( x, y, Color.Black );
    }
}

private void fastest_average_filter()
{ Int32 x, x0, x1, x2, x3, y, y0, y1;
  Int32 w = b0.Width, h = b0.Height;
  Int32 Rsum, Gsum, Bsum;
  Int64 t0, t1;
  Cursor.Current = Cursors.WaitCursor;
  t0 = DateTime.Now.Ticks;
  //Intermediary arrays
  Int32[,] R = new Int32[b0.Height, b0.Width];
  Int32[,] G = new Int32[b0.Height, b0.Width];
  Int32[,] B = new Int32[b0.Height, b0.Width];

  BitmapData b1Data = b1.LockBits( new Rectangle( 0,0,b1.Width,b1.Height ),
                                   ImageLockMode.WriteOnly, PixelFormat.Format32bppRgb );
  unsafe
  { UInt32* pb1start = (UInt32*)b1Data.Scan0, pb1;
    fixed ( Byte * pR0 = R0, pG0 = G0, pB0 = B0 )
    fixed ( Int32* pR  = R , pG  = G , pB  = B  )
    { Byte*  ppR0, ppG0, ppB0;
      Int32* ppR1, ppG1, ppB1;
      Int32* ppR2, ppG2, ppB2;
      Byte*  ppR3, ppG3, ppB3;
      //Horizontal integration
      for ( y=0; y < b0.Height; y++ )
      { ppR0 = pR0 +y*w   ; ppG0 = pG0 +y*w   ; ppB0 = pB0 +y*w   ;
        ppR1 = pR  +y*w+Mh; ppG1 = pG  +y*w+Mh; ppB1 = pB  +y*w+Mh;
        for ( x=0; x < M; x++ )
        { *ppR1 += *ppR0++;
          *ppG1 += *ppG0++;
          *ppB1 += *ppB0++;
        }
        ppR0 = pR0 +y*w; ppG0 = pG0 +y*w; ppB0 = pB0 +y*w;
        ppR2 = ppR1+1  ; ppG2 = ppG1+1  ; ppB2 = ppB1+1  ;
        ppR3 = ppR0+M  ; ppG3 = ppG0+M  ; ppB3 = ppB0+M  ;
        for ( x=Mh+1; x < w-Mh; x++ )
        { *ppR2++ = *ppR1++ - *ppR0++ + *ppR3++;
          *ppG2++ = *ppG1++ - *ppG0++ + *ppG3++;
          *ppB2++ = *ppB1++ - *ppB0++ + *ppB3++;
        }
      }
```

```
        //Vertical integration
        for ( x=Mh; x < b0.Width-Mh; x++ )
        { ppR1 = pR + x; ppG1 = pG + x; ppB1 = pB + x;
          Rsum = Gsum = Bsum = 0;
          for ( y=0; y < N; y++, ppR1+=w, ppG1+=w, ppB1+=w )
          { Rsum += *ppR1;
            Gsum += *ppG1;
            Bsum += *ppB1;
          }
          pb1 = pb1start + Nh * w + x;
          *pb1 = (UInt32)Color.FromArgb( Rsum/MN, Gsum/MN, Bsum/MN ).ToArgb();
          ppR1 = pR  +x  ; ppG1 = pG  +x  ; ppB1 = pB  +x  ;
          ppR2 = ppR1+N*w; ppG2 = ppG1+N*w; ppB2 = ppB1+N*w;
          for ( y=Nh+1; y < h-Nh; y++, ppR1+=w, ppR2+=w, ppG1+=w, ppG2+=w, ppB1+=w, ppB2+=w
)
          { Rsum = Rsum - *ppR1 + *ppR2;
            Gsum = Gsum - *ppG1 + *ppG2;
            Bsum = Bsum - *ppB1 + *ppB2;
            pb1 = pb1start + y * w + x;
            *pb1 = (UInt32)Color.FromArgb( Rsum/MN, Gsum/MN, Bsum/MN ).ToArgb();
          }
        }
      } //end of fixed
    } //end of unsafe
    b1.UnlockBits( b1Data );
    t1 = DateTime.Now.Ticks;
    s = "Fastest average filter\r\n" +
        "Image:  " + b0.Width.ToString() + " x " + b0.Height.ToString() + "\r\n" +
        "Filter: " + M.ToString() + " x " + N.ToString() + "\r\n" +
        "Filter Time: " + String.Format( "{0:F1}", (t1 - t0)/1000000f ) + " MegaTicks";
    Cursor.Current = Cursors.Arrow;
  }
}
```

Recommended experiments:
1) In line 9 of the code change `const Int32 M = 21` to any odd integer `1 ≤ M < image.Width`.
2) In line 10 of the code change `const Int32 N = 21` to any odd integer `1 ≤ N < image.Height`.
3) Load different images via the `File` menu.
4) Comment out the calls `border_painting();` in lines 24 and 37.

How to embed an arbitrary sample image into the code:
Copy all the code into an empty `Form1.cs` of a new `Windows Application C#`-project `fastest_average_filter` and delete `Form1.Designer.cs` and `Program.cs`.
1) In the `Solution Explorer` window right click on `fastest_average_filter`. A context menu opens.
2) Click `Add`. Click `Existing Item`. A file dialog box opens.
3) At the bottom of the box is a drop-down menu: `Files of type:`
4) Select `Image Files (*.gif;*.jpg;...)`
5) Choose an arbitrary image from your computer and leave the file dialog box with button `Add`.
6) The file name should now appear in the `Solution Explorer` tree. Right click this name.
7) A context menu opens. Click `Properties`.
   A `Properties`-window opens below the `Solution Explorer` window.
8) Click its first property: `Build Action` and set it to `Embedded Resource`.
9) Line 23 of the program below is:
`b0 = new Bitmap( typeof( Form1 ), "fastest_average_filter.Butterfly.jpg" );`
Replace `Butterfly.jpg` by the name of your image.