

## Code Samples: A Faster Averaging Filter in C# unsafe mode

Copyright © by V. Miszalok, last update: 22-11-2006

The following code is about 40% faster than the previous `average_fast` code. The trick is to avoid the `SetPixel(..)`-method of the `Bitmap`-class which is quite slow.

The price is a code compiled with the `unsafe` flag, which is obligatory when a code uses a pointer, the operating system warns any user at execution of this code.

The code of this project is nearly identical to the previous `fast_averaging`-code. There are just 8 new statements around and inside the "Vertical integration" part of the `private void fast_average_filter()-function`:

```

1  BitmapData b1Data = b1.LockBits( new Rectangle( 0,0,b1.Width,b1.Height ),
                                   ImageLockMode.WriteOnly, PixelFormat.Format32bppRgb );
2  unsafe
3  { UInt32* pblstart = (UInt32*)b1Data.Scan0, pbl;
    //Vertical integration
    .
    .
4      pbl = pblstart + Nh * b1.Width + x;
5      *pbl = (UInt32)Color.FromArgb( Rsum/MN, Gsum/MN, Bsum/MN ).ToArgb();
    .
    .
6      pbl = pblstart + y * b1.Width + x;
7      *pbl = (UInt32)Color.FromArgb( Rsum/MN, Gsum/MN, Bsum/MN ).ToArgb();
    }
    } //end of unsafe
8  b1.UnlockBits( b1Data );

```

line 1: Lock the output `Bitmap b1` in memory from being moved by the garbage collector.

line 2: Starting clause of the `unsafe`-code fragment.

line 3: `pblstart` points to the first pixel of `b1`.

line 4: `pbl` points to pixel`[x,Nh]`.

line 5: The averaged color is converted to an unsigned integer and written to pixel`[x,Nh]`.

line 6: `pbl` points to pixel`[x,y]`.

line 7: The averaged color is converted to an unsigned integer and written to pixel`[x,y]`.

line 8: Give back all rights to the garbage collector.

The algorithm behaves in the same manner as the previous did. It accepts any rectangular filter sizes  $M \neq N$  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 "faster\_average\_filter". Set the `unsafe`-flags as follows:

Main menu of Visual Studio → Project → faster\_average\_filter Properties → Build → check:

Allow unsafe code → Debug → check: Enable unmanaged code debugging. Go back to `Form1.cs`.

Delete the files `Form1.Designer.cs` and `Program.cs` from the project. Clear any prefabricated content from `Form1.cs` and replace it by the following code:

```

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 = "Faster Average Filter";
  SetStyle( ControlStyles.ResizeRedraw, true );
  try { b0 = new Bitmap( typeof( Form1 ), "faster_average_filter.Butterfly.jpg" );
    byte_arrays_and_b1_image();
    faster_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();
  faster_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 faster_average_filter()
{ Int32 x, x0, x1, x2, x3, y, y0, y1;
  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];
}

```

```

//Horizontal integration
for ( y=0; y < b0.Height; y++ )
{ for ( x=0; x < M; x++ ) //first column = column[Mh]
  { R[y,Mh] += R0[y,x];
    G[y,Mh] += G0[y,x];
    B[y,Mh] += B0[y,x];
  }
  for ( x0=0, x1=Mh, x2=Mh+1, x3=M; x3 < b0.Width; x0++, x1++, x2++, x3++ )
  { R[y,x2] = R[y,x1] - R0[y,x0] + R0[y,x3];
    G[y,x2] = G[y,x1] - G0[y,x0] + G0[y,x3];
    B[y,x2] = B[y,x1] - B0[y,x0] + B0[y,x3];
  }
}
BitmapData b1Data = b1.LockBits( new Rectangle( 0,0,b1.Width,b1.Height ),
                                ImageLockMode.WriteOnly, PixelFormat.Format32bppRgb );
unsafe
{ UInt32* pblstart = (UInt32*)b1Data.Scan0, pbl;
  //Vertical integration
  for ( x=Mh; x < b0.Width-Mh; x++ )
  { Rsum = Gsum = Bsum = 0;
    for ( y=0; y < N; y++ )
    { Rsum += R[y,x];
      Gsum += G[y,x];
      Bsum += B[y,x];
    }
    pbl = pblstart + Nh * b1.Width + x;
    *pbl = (UInt32)Color.FromArgb( Rsum/MN, Gsum/MN, Bsum/MN ).ToArgb();

    for ( y0=0, y=Nh+1, y1=N; y1 < b0.Height; y++, y0++, y1++ )
    { Rsum = Rsum - R[y0,x] + R[y1,x];
      Gsum = Gsum - G[y0,x] + G[y1,x];
      Bsum = Bsum - B[y0,x] + B[y1,x];
      pbl = pblstart + y * b1.Width + x;
      *pbl = (UInt32)Color.FromArgb( Rsum/MN, Gsum/MN, Bsum/MN ).ToArgb();
    }
  }
} //end of unsafe
b1.UnlockBits( b1Data );

t1 = DateTime.Now.Ticks;
s = "Faster 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  $1 \leq M < \text{image.Width}$ .
- 2) In line 10 of the code change `const Int32 N = 21` to any odd integer  $1 \leq N < \text{image.Height}$ .
- 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 `faster_average_filter` and delete `Form1.Designer.cs` and `Program.cs`.

- 1) In the Solution Explorer window right click on `faster_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 ), "faster_average_filter.Butterfly.jpg" );`  
Replace `Butterfly.jpg` by the name of your image.