Code Samples:
The Sigma Filter in C#

This code has been developed with Visual C#2.0.
The algorithm realizes the non-linear sigma filter which is an interesting variant of the linear average filter.
The trick is not to sum up all pixels inside the filter kernel but only the subset with grey values near the center grey value: Noise reduction without blurring.
The filter preserves the classical form with 4 nested `for`-loops and a final division by the cardinality of the subset.
The algorithm accepts any quadratic filter sizes `NxN` where `N` must be a positive odd integer `N < b0.Width, N < b0.Height`. It accepts any `0≤sigma&le255`. `sigma==0` does not change the image. `sigma==255` is a (slow) average filter.
The algorithm works with any image format that can be read and written with the `GetPixel(..), SetPixel(..)` methods of the `Bitmap`-class.
Time depends on ImageSize*FilterSize = `b0.Width*b0.Height*N*N`.

Create a new Windows-project "`sigma`". 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 N = 11, Nh = N/2;  //N must be an odd integer < b0.Width && < b0.Height
  const Int32 sigma = 100; //must be an integer 0<=sigma<=255
  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 = "Sigma Filter";
    SetStyle( ControlStyles.ResizeRedraw, true );
    try { b0 = new Bitmap( typeof( Form1 ), "sigma.Butterfly.jpg" );
          byte_arrays_and_b1_image();
          sigma_filter( sigma );
          //border_painting();
        } catch {}
    Width  = 800;
    Height = 600;
  }

  void MenuFileRead( object obj, EventArgs ea )
  { OpenFileDialog dlg = new OpenFileDialog();
    dlg.Filter = "bmp files (*.bmp)|*.bmp|All files (*.*)|*.*"  ;
    if ( dlg.ShowDialog() != DialogResult.OK ) return;
    b0 = (Bitmap)Image.FromFile( dlg.FileName );
    byte_arrays_and_b1_image();
    sigma_filter( sigma );
    //border_painting();
    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 sigma_filter( Int32 sigma )
  { Int32 x, xx, xxx, y, yy, yyy;
    Int32 r, g, b, sumR, sumG, sumB, countR, countG, countB;
    Int64 t0, t1;
    Color c;
    float Rf, Gf, Bf;
    Cursor.Current = Cursors.WaitCursor;
    t0 = DateTime.Now.Ticks;
    for ( y=Nh; y < b0.Height-Nh; y++ ) //==================
    { for ( x=Nh; x < b0.Width-Nh; x++ ) //===============
      { c = Color.FromArgb( R0[y,x], G0[y,x], B0[y,x] );
        sumR = sumG = sumB = countR = countG = countB = 0;
        for ( yy=-Nh; yy <= Nh; yy++ ) //=============
        { yyy = y + yy;
          for ( xx=-Nh; xx <= Nh; xx++ )//========
          { xxx = x + xx; r = R0[yyy,xxx]; g = G0[yyy,xxx]; b = B0[yyy,xxx];
            if ( Math.Abs( c.R - r ) <= sigma ) { sumR += r; countR++; }
            if ( Math.Abs( c.G - g ) <= sigma ) { sumG += g; countG++; }
            if ( Math.Abs( c.B - b ) <= sigma ) { sumB += b; countB++; }
          } //====== end for (int xx... ================
        } //======== end for (int yy... ==================
        Rf = sumR/countR; Gf = sumG/countG; Bf = sumB/countB;
        b1.SetPixel( x, y, Color.FromArgb( Convert.ToInt32(Rf),
                                           Convert.ToInt32(Gf),
                                           Convert.ToInt32(Bf) ) );
      } //============ end for (int x... =====================
    } //============== end for (int y... =======================
    t1 = DateTime.Now.Ticks;
    s = "Sigma filter\r\n" +
        "Image:  " + b0.Width.ToString() + " x " + b0.Height.ToString() + "\r\n" +
        "Filter: " + N.ToString() + " x " + N.ToString() + "\r\n" +
        "Sigma: " + sigma.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 N = 11;` to any odd integer `1 ≤ N < image.Width && 1 ≤ N < image.Height`. Try out extreme values `N = 1` and `N = 249`.

2) Load different images by using the `File` menu.

3) Try out different sigmas in line 10: `const Int32 sigma = 100;`