Code Samples:
The Fast Gauss Filter in C#

The program executes two separate algorithms:
1) A classical (simple) `NxN` convolution with its own quadratic filter kernel with circularily symmetric values of a Gaussian bell-shaped curve.
The bell-shaped curve automatically adjusts (at any kernel size) its central value to 1.0 and its corner values to 0.01.
2) A three time repetition of the fast averaging algorithm with 3 identical filter sizes `(N/2-1)x(N/2-1)`.
Until `N≤13` (Release Mode compilation) **1)** is faster than **2)** but with `N≥15` **2)** is faster than **1)**.
The algorithms accept any quadratic filter sizes `NxN`
where `N` must be a positive odd integer `7≤N<b0.Width, 7≤N<b0.Height`.
The algorithms works with any image format that can be read and
written with the `GetPixel(..), SetPixel(..)` methods of the `Bitmap`-class.
Create a new Windows-project "`gauss_fast`". 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, b2;
  const Int32 N = 21;  //N must be an odd integer N >= 9 but N < b0.Width && N < b0.Height
  const Int32 NN = N/2 - 1; //Size of the 3 consecutive average filters
  Byte [,] R0, G0, B0;
  String s1, s2;

  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 = "Fast and Simple Gauss Filters";
    SetStyle( ControlStyles.ResizeRedraw, true );
    try { b0 = new Bitmap( typeof( Form1 ), "gauss_fast.Butterfly.jpg" );
          byte_arrays_and_b1_image();
          convolution_with_a_gauss_filter( N );
          fast_quadratic_average_filter_x3( NN );
          //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();
    convolution_with_a_gauss_filter( N );
    fast_quadratic_average_filter_x3( NN );
    //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.DrawImage( b2, b0.Width+10, b0.Height+10 );
      g.DrawString( s1, new Font( "Arial", 16 ), new SolidBrush( Color.Red ),
                    b0.Width+10, 10 );
      g.DrawString( s2, new Font( "Arial", 16 ), new SolidBrush( Color.Red ),
                    b0.Width+10, b0.Height+20 );
    } 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();
  if ( b2 != null ) b2.Dispose();
  b1 = new Bitmap( b0 );
  b2 = 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 );
      b2.SetPixel( x, y, Color.Black );
    }
}

private void fast_quadratic_average_filter_x3( Int32 NN )
{ Int32 NNh = NN/2;
  Int32 x, x0, x1, x2, x3;
  Int32 y, y0, y1, y2, y3;
  Int64 t0, t1;
  float Rf, Gf, Bf;
  //Intermediary arrays
  Int32[,] R1 = new Int32[b0.Height, b0.Width];
  Int32[,] G1 = new Int32[b0.Height, b0.Width];
  Int32[,] B1 = new Int32[b0.Height, b0.Width];
  Int32[,] R2 = new Int32[b0.Height, b0.Width];
  Int32[,] G2 = new Int32[b0.Height, b0.Width];
  Int32[,] B2 = new Int32[b0.Height, b0.Width];
  Cursor.Current = Cursors.WaitCursor;
  t0 = DateTime.Now.Ticks;
  for ( int z=0; z < 3; z++ ) //z = counter for 3 passes
  { //Horizontal integration
    for ( y=0; y < b0.Height; y++ )
    { R1[y,NNh] = G1[y,NNh] = B1[y,NNh] = 0;
      for ( x=0; x < NN; x++ )
      { R1[y,NNh] += R0[y,x];
        G1[y,NNh] += G0[y,x];
        B1[y,NNh] += B0[y,x];
      }
      for ( x0=0, x1=NNh, x2=NNh+1, x3=NN; x3 < b0.Width; x0++, x1++, x2++, x3++ )
      { R1[y,x2] = R1[y,x1] - R0[y,x0] + R0[y,x3];
        G1[y,x2] = G1[y,x1] - G0[y,x0] + G0[y,x3];
        B1[y,x2] = B1[y,x1] - B0[y,x0] + B0[y,x3];
      }
    }
    //Vertical integration
    for ( x=NNh; x < b0.Width-NNh; x++ )
    { R2[NNh,x] = G2[NNh,x] = B2[NNh,x] = 0;
      for ( y=0; y < NN; y++ )
      { R2[NNh,x] += R1[y,x];
        G2[NNh,x] += G1[y,x];
        B2[NNh,x] += B1[y,x];
      }
      for ( y0=0, y1=NNh, y2=NNh+1, y3=NN; y3 < b0.Height; y0++, y1++, y2++, y3++ )
      { R2[y2,x] = R2[y1,x] - R1[y0,x] + R1[y3,x];
        G2[y2,x] = G2[y1,x] - G1[y0,x] + G1[y3,x];
        B2[y2,x] = B2[y1,x] - B1[y0,x] + B1[y3,x];
      }
    }
    float divisor = NN*NN;
    for ( y=NNh; y < b0.Height-NNh; y++ )
      for ( x=NNh; x < b0.Width-NNh; x++ )
      { Rf = R2[y,x]/divisor; Gf = G2[y,x]/divisor; Bf = B2[y,x]/divisor;
        Color c = Color.FromArgb( Convert.ToByte(Rf),
                                  Convert.ToByte(Gf),
                                  Convert.ToByte(Bf) );
        if ( z==2 ) b1.SetPixel( x, y, c );
        else { R0[y,x] = c.R; G0[y,x] = c.G; B0[y,x] = c.B; }
      }
  } //end of z-Loop
  t1 = DateTime.Now.Ticks;
  s1 = "3 consecutive fast average filters\r\n" +
       "Image:  " + b0.Width.ToString() + " x " + b0.Height.ToString() + "\r\n" +
       "Filters: " + NN.ToString() + " x " + NN.ToString() + "\r\n" +
       "Filter Time: " + String.Format( "{0:F1}", (t1 - t0)/1000000f ) + " MegaTicks";
  Cursor.Current = Cursors.Arrow;
}
```

```
  private void convolution_with_a_gauss_filter( Int32 N )
  { Int32 Nh = N/2;
    Int32 x, xx, xxx, y, yy, yyy;
    Int64 t0, t1;
    float[,] kernel = new float[N,N]; //quadratic Gauss kernel
    float sum_kernel = 0.0f;
    float sumR, sumG, sumB, weight, Rf, Gf, Bf;
    Cursor.Current = Cursors.WaitCursor;
    t0 = DateTime.Now.Ticks;
    //Construction of a suitable 2D-Gaussian bell-shape kernel:
    //The corner elements of the kernel are Nh*sqrt(2) away from its center
    //and therefore obtain the lowest values.
    //We adjust the parameter a of the e-function y = e -(a*x2) so,
    //that always (at any kernel size except 3x3) these corners obtain at least 1% weight.
    double a = 1.0f;
    if ( N > 3 ) a = - 2 * Nh * Nh / Math.Log( 0.01 );
    //fill the kernel with elements depending on their distance and on a.
    for ( y=0; y < N; y++ )
      for ( x=0; x < N; x++ )
      { double dist = Math.Sqrt( (x-Nh)*(x-Nh) + (y-Nh)*(y-Nh) );
        sum_kernel += kernel[y,x] = (float)( Math.Exp( - dist*dist / a ) );
      }
    //Convolution
    for ( y=Nh; y < b0.Height-Nh; y++ ) //==================
    { for ( x=Nh; x < b0.Width-Nh; x++ ) //===============
      { sumR = sumG = sumB = 0.0f;
        for ( yy=-Nh; yy <= Nh; yy++ ) //=============
        { yyy = y + yy;
          for ( xx=-Nh; xx <= Nh; xx++ )//========
          { weight = kernel[yy+Nh,xx+Nh];
            xxx = x + xx;
            sumR += weight * R0[yyy,xxx];
            sumG += weight * G0[yyy,xxx];
            sumB += weight * B0[yyy,xxx];
          } //====== end for (int xx... ================
        } //======== end for (int yy... ==================
        Rf = sumR/sum_kernel; Gf = sumG/sum_kernel; Bf = sumB/sum_kernel;
        b2.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;
    s2 = "Simple quadratic Gauss filter\r\n" +
         "Image:  " + b0.Width.ToString() + " x " + b0.Height.ToString() + "\r\n" +
         "Filter: " + N.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 N = 21` to any odd integer `9 ≤ N < image.Width && 9 ≤ N < image.Height`. Try out extreme values `N = 5` and `N = 211`.
2) Load different images via the `File` menu.
3) Increase or decrease `const Int32 NN = N/2 - 1;` in line 10.
4) Increase the no. of average passes `z<3` in line 93 to `z<5`.

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 `gauss_fast` and delete `Form1.Designer.cs` and `Program.cs`.
1) In the `Solution Explorer` window right click on `gauss_fast`. 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 22 of the program below is:
`b0 = new Bitmap( typeof( Form1 ), "gauss_fast.Butterfly.jpg" );`
Replace `Butterfly.jpg` by the name of your image.