

Code Samples: Simple Component Labeling in C#

Copyright © by W. Kovalevski and V. Miszalok, last update: 11-03-2006

This code has been developed with Visual C#2.0.

Create a new Windows-project "simple_label". 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.Windows.Forms;

public class Form1 : Form
{ static void Main() { Application.Run( new Form1() ); }
  const Int32 xSize = 11;
  const Int32 ySize = 12;
  Byte[,] I      = new Byte[ySize,xSize]; //Original Image
  int [,] Lfirst = new int [ySize,xSize]; //Labels after first step
  int [,] Lmerge = new int [ySize,xSize]; //Labels after merging
  int [,] Lsorted = new int [ySize,xSize]; //Labels after sorting
  Brush[] brush = new Brush[10];
  Brush redbrush = new SolidBrush( Color.Red );
  Button[] button = new Button[ySize];
  CheckBox checkbox = new CheckBox();
  Int32 i, x, y, dx, dy, flag, label, no_of_merge_passes;

  public Form1()
  { BackColor = Color.White;
    Text      = "Simple Labeling";
    SetStyle( ControlStyles.ResizeRedraw, true );
    Width    = 800;
    Height   = 600;
    for ( i=0; i < 10; i++ )
      brush[i] = new SolidBrush( Color.FromArgb( i*25, i*25, i*25 ) );
    for ( y=0; y < 6; y++ )
    { button[y] = new Button();
      Controls.Add(button[y]);
      button[y].BackColor = Color.Gray;
      button[y].Click += new EventHandler( do_it );
    }
    button[0].Name = button[0].Text = "Homunculus";
    button[1].Name = button[1].Text = "Label First";
    button[2].Name = button[2].Text = "Label Merge";
    button[3].Name = button[3].Text = "Label Sorted";
    button[4].Name = button[4].Text = "Noise";
    button[5].Name = button[5].Text = "Clear";
    Controls.Add(checkbox);
    checkbox.Text = "8-Neighborhood";
  }

  protected override void OnPaint( PaintEventArgs e )
  { Graphics g = e.Graphics;
    Rectangle r = ClientRectangle;
    dx = r.Width / (xSize+2);
    dy = r.Height / ySize;
    for ( y=0; y < 6; y++ )
    { button[y].Top = y*dy+1;
      button[y].Left = xSize*dx+1;
      button[y].Width = 2*dx-2;
      button[y].Height = dy-2;
    }
    checkbox.Top = y*dy+1;
    checkbox.Left = xSize*dx+1;
  }
}
```

```

for ( y=0; y < ySize; y++ )
  for ( x=0; x < xSize; x++ )
    { g.FillRectangle( brush[I[y,x]], x*dx, y*dy, dx, dy );
      switch ( flag )
        { case 2: g.DrawString( Lfirst [y,x].ToString(), Font, redbrush, x*dx+dx/2, y*dy+dy/2); break;
          case 3: g.DrawString( Lmerge [y,x].ToString(), Font, redbrush, x*dx+dx/2, y*dy+dy/2); break;
          case 4: g.DrawString( Lsorted[y,x].ToString(), Font, redbrush, x*dx+dx/2, y*dy+dy/2); break;
        }
      }
String s = "          ";
if ( flag == 3 || flag == 4 ) s = "passes = " + no_of_merge_passes.ToString();
g.DrawString( s, Font, redbrush, checkbox.Left, checkbox.Top+checkbox.Height+2 );
}

protected void do_it( object sender, System.EventArgs e )
{ switch( ((Button)sender).Name )
  { case "Homunculus":
    I = new Byte[,] { {9,0,0,0,0,4,0,0,0,0,9},
                      {0,0,0,0,9,9,9,0,0,0,0},
                      {0,0,0,0,9,5,9,0,0,0,0},
                      {0,0,0,0,9,9,9,0,0,0,0},
                      {0,0,0,0,0,9,0,0,0,0,0},
                      {0,9,9,9,9,9,9,9,9,0},
                      {0,0,0,0,9,9,9,0,0,0,0},
                      {0,0,0,0,9,9,9,0,0,0,0},
                      {0,0,0,0,9,0,9,0,0,0,0},
                      {0,0,0,0,9,0,9,0,0,0,0},
                      {0,0,0,0,9,0,9,0,0,0,0},
                      {9,0,8,8,0,0,0,8,8,0,9} };

    flag = 1; Invalidate(); break;
  case "Label First":
    fill_Lfirst();
    flag = 2; Invalidate(); break;
  case "Label Merge":
    fill_Lfirst();
    fill_Lmerge();
    flag = 3; Invalidate(); break;
  case "Label Sorted":
    fill_Lfirst();
    fill_Lmerge();
    fill_Lsorted();
    flag = 4; Invalidate(); break;
  case "Noise":
    Random random = new Random();
    for ( y=0; y < ySize; y++ )
      for ( x=0; x < xSize; x++ )
        { Int32 noise = random.Next() % 3 - 1;
          noise += I[y,x];
          if ( noise < 0 ) I[y,x] = 0;
          else if ( noise > 9 ) I[y,x] = 9;
          else I[y,x] = (Byte)noise;
        }
    flag = 5; Invalidate(); break;
  case "Clear":
    for ( y=0; y < ySize; y++ )
      for ( x=0; x < xSize; x++ ) { Lfirst[y,x] = Lmerge[y,x] = Lsorted[y,x] = I[y,x]
= 0; }
    flag = 6; Invalidate(); break;
  }
}

private void fill_Lfirst()
{ Lfirst[0,0] = label = 1;
  for ( x=1; x < xSize; x++ )
    if ( I[0,x] == I[0,x-1] ) Lfirst[0,x] = Lfirst[0,x-1]; //label from left
    else Lfirst[0,x] = ++label;
  for ( y=1; y < ySize; y++ )
    { if ( I[y,0] == I[y-1,0] ) Lfirst[y,0] = Lfirst[y-1,0]; //label from upstairs
      else Lfirst[y,0] = ++label;
      for ( x=1; x < xSize; x++ )
        if ( I[y,x] == I[y-1,x] ) Lfirst[y,x] = Lfirst[y-1,x]; //label from upstairs
        else if ( I[y,x] == I[y,x-1] ) Lfirst[y,x] = Lfirst[y,x-1]; //label from left
        else Lfirst[y,x] = ++label;
    }
}
}

```

```

private void fill_Lmerge()
{ for ( y=0; y < ySize; y++ )
  for ( x=0; x < xSize; x++ )
    Lmerge[y,x] = Lfirst[y,x]; //just copy
no_of_merge_passes = 0;
do
{ i = 0; no_of_merge_passes++;
  for ( y=0; y < ySize; y++ )
    for ( x=0; x < xSize; x++ )
      { if ( x < xSize-1 && I[y,x] == I[y ,x+1] ) //merge with right neighbor ?
        if ( Lmerge[y,x] != Lmerge[y ,x+1] )
          { Lmerge[y,x] = Lmerge[y ,x+1] =
            Math.Min( Lmerge[y,x], Lmerge[y ,x+1] ); i++; }
        if ( y < ySize-1 && I[y,x] == I[y+1,x ] ) //merge with lower neighbor ?
          if ( Lmerge[y,x] != Lmerge[y+1,x ] )
            { Lmerge[y,x] = Lmerge[y+1,x ] =
              Math.Min( Lmerge[y,x], Lmerge[y+1,x ] ); i++; }
        if ( x > 0 && I[y,x] == I[y ,x-1] ) //merge with left neighbor ?
          if ( Lmerge[y,x] != Lmerge[y ,x-1] )
            { Lmerge[y,x] = Lmerge[y ,x-1] =
              Math.Min( Lmerge[y,x], Lmerge[y ,x-1] ); i++; }
        if ( y > 0 && I[y,x] == I[y-1,x ] ) //merge with upper neighbor ?
          if ( Lmerge[y,x] != Lmerge[y-1,x ] )
            { Lmerge[y,x] = Lmerge[y-1,x ] =
              Math.Min( Lmerge[y,x], Lmerge[y-1,x ] ); i++; }
        if ( checkbox.Checked ) //8-Neighborhood ?
        { if ( x < xSize-1 && y < ySize-1 && I[y,x] == I[y+1,x+1] ) // right lower neighbor ?
          if ( Lmerge[y,x] != Lmerge[y+1,x+1] )
            { Lmerge[y,x] = Lmerge[y+1,x+1] =
              Math.Min( Lmerge[y,x], Lmerge[y+1,x+1] ); i++; }
          if ( x > 0 && y < ySize-1 && I[y,x] == I[y+1,x-1] ) // left lower neighbor ?
            if ( Lmerge[y,x] != Lmerge[y+1,x-1] )
              { Lmerge[y,x] = Lmerge[y+1,x-1] =
                Math.Min( Lmerge[y,x], Lmerge[y+1,x-1] ); i++; }
          if ( x > 0 && y > 0 && I[y,x] == I[y-1,x-1] ) // left upper neighbor ?
            if ( Lmerge[y,x] != Lmerge[y-1,x-1] )
              { Lmerge[y,x] = Lmerge[y-1,x-1] =
                Math.Min( Lmerge[y,x], Lmerge[y-1,x-1] ); i++; }
          if ( x < xSize-1 && y > 0 && I[y,x] == I[y-1,x+1] ) // right upper neighbor ?
            if ( Lmerge[y,x] != Lmerge[y-1,x+1] )
              { Lmerge[y,x] = Lmerge[y-1,x+1] =
                Math.Min( Lmerge[y,x], Lmerge[y-1,x+1] ); i++; }
        }
      }
    } while ( i > 0 );
}

private void fill_Lsorted()
{ int[,] histogram = new int[label+1,2]; //histogram with 2 lines
  for ( y=0; y < ySize; y++ )
    for ( x=0; x < xSize; x++ )
      histogram[Lmerge[y,x],0]++; //first line filling
  int max, imax=0, newlabel=1;
  do //sorted new labels from 1 = biggest, 2 = second etc.
  { max = 0;
    for ( i=1; i <= label; i++ )
      { if ( histogram[i,1] > 0 ) continue; //already sorted
        if ( histogram[i,0] > max ) { max = histogram[i,0]; imax = i; }
      }
    if ( max > 0 ) histogram[imax,1] = newlabel++;
  } while ( max > 0 );
  for ( y=0; y < ySize; y++ ) //apply new labels
    for ( x=0; x < xSize; x++ )
      Lsorted[y,x] = histogram[Lmerge[y,x],1];
}
}

```

Recommended experiments:

- 1) Observe how the no. of labels shrinks from "Label First" to "Label Merge" to "Label Sorted".
- 2) Try out the "8-Neighborhood" checkbox, label again and observe the feet of the homunculus.
- 3) Observe how the no. of passes leaps (red string under the checkbox) between checked and unchecked. Explanation for the homunculus sample: In case of 8-neighborhood the merging chain of the black area passes below the legs from left to right.
- 4) Replace the "4"-pixel in the mid of the uppermost row of the homunculus matrix I by a "0"-pixel. The no. of passes doesn't leap any more between checked and unchecked. Explanation for the homunculus sample: The first labeling already connects left and right black areas. There is no need for any complicated passes below the legs.
- 5) Try out more changes of the homunculus matrix I , such as cutting the neck etc.
- 6) Clear and click several times on "Noise" to produce a random image and try it out without and with "8-Neighborhood".
- 7) Change the homunculus matrix I to a single diagonal pixel line and observe how it behaves without and with "8-Neighborhood".

FAQ: What is the difference between normal labeling and component labeling ?

A: 1) Normal labeling:

Pixels within a range of gray values obtain the same label number all over the image. (Very easy with a Look Up Table = palette).

Sample: Both feet of the homunculus obtain the same number.

Consequence: Different disconnected areas of the same range obtain the same number.

Drawback: It is not possible to count the number of disconnected areas.

A: 2) Component labeling:

Any connected area obtain its own unique arbitrary number.

Sample: The feet of the homunculus obtain different numbers.

FAQ: What is the basic idea of the program ?

A: The basic idea is region growing:

1. Step: From upper left to lower right: any new grey value generates a new component.
2. Step: Any labeled pixel seeks to expand itself and to absorb as many neighbors as possible. The process stops when no further absorptions are possible.
3. Step: Sort the components by size (label 1 = biggest component) and remove the number gaps.

FAQ: What are the drawbacks of this program ?

A: The 2. Step can be very time consuming: A complete run through the whole image just expands a component by a thin rim of thickness 1. In the worst case a complete run through the whole image just expands one component by one pixel.

FAQ: How to adapt the code to real images ?

A:

1. Copy Your whole image to a matrix I for faster pixel access.
2. Define a label matrix L (= unified L_{first} , L_{merge} and L_{sorted}).
3. Define a lower and an upper grey value $threshold[j]$ for any desired component j (RGB-thresholds in case of color images).
4. Replace all tests such as "if ($I[y,x] == I[y,x+1]$)" by


```
if (  $I[y, x] >= lower\_threshold[j]$  &&  $I[y, x] < upper\_threshold[j]$  &&
       $I[y,x+1] >= lower\_threshold[j]$  &&  $I[y,x+1] < upper\_threshold[j]$  )
```

FAQ: Alternative method of component labeling ?

A: Yes: crack code labeling (much faster).

- 1) Make up a C1V-Matrix $L[ySize, xSize+1]$.
- 2) Define Your desired threshold j .
- 3) Whenever You find a new starting point give a new number to this crack code and when there occurs a south- or north-crack, store the number into the L -matrix.
- 4) When You do not find any more start points goto 2).
- 5) When You do not want more thresholds, fill the horizontal empty gaps between identical numbers in all lines of L .