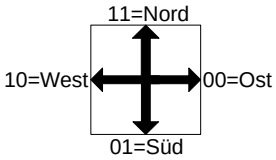
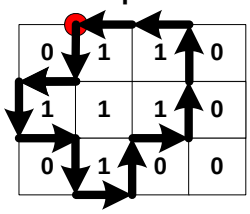
Computer Vision, Chapter 2: Chain Code

Der neue Kettencode mit 4 Richtungen = Chain Code oder Crack Code



Moderner Code für Gebietsbegrenzungen kodiert, ausgehend von einer 0-Zelle `(x0,y0)` die Begrenzung als Kette gerichteter Cracks `00`=Ost, `01`=Süd, `10`=West, `11`=Nord. Der Code beachtet die Berandungsrelation und liefert immer ein geschlossene Crackkette mit Endpunkt == Startpunkt == `(x0,y0)`.
Eigenschaft 1: Anzahl der Südcracks = Anzahl der Nordcracks
Eigenschaft 2: Anzahl der Ostcracks = Anzahl der Westcracks
Eigenschaft 3: Umfang des Gebiets = Anzahl der Cracks.
Konvention 1: Startpixel immer am ersten, obersten Übergang von Hintergrund auf Vordergrund.
Konvention 2: Startcrack ist immer Süd, d.h. Umlaufrichtung immer so, dass Vordergrund links und Hintergrund rechts liegt.

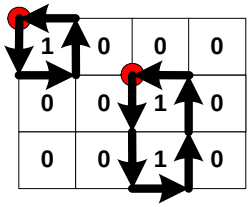
Beispiel 1:




```
          0110
Bild b = 1110
          0100
```

Startpunkt = Endpunkt = 0-Zelle `(1/0)`
Crackkette = süd, west, süd, ost, süd, ost, nord, ost, nord, nord, west, west = `swsosononnww`
Oft benutzt man englisch south, east, north, west mit den Abkürzungen s,e,n,w.
Dann heißt der komplette Chain Code des Gebiets: `(1/0)swsesenennww`
Perimeter = Anzahl der Cracks = 12, Fläche = Anzahl der eingeschlossenen Pixel = 6

Beispiel 2:




```
          1000
Bild b = 0010
          0010
```

Gebiet 1: `(0/0)senw` , Perimeter = 4, Fläche = 1
Gebiet 2: `(2/1)ssennw`, Perimeter = 6, Fläche = 2

Dieser Chain Code läßt sich zwanglos erweitern auf 3D-Raster. Er bekommt dann zwei zusätzliche Richtungen backwards = `b` und forwards = `f`.
`b` zeigt in die positive und `f` in die negativer Z-Richtung, also `b` in den Display und `f` aus dem Display.
Beispiel 1: `(0,0,0)esb` ist der Code für einen Weg, der bei einem Würfel entlang der C1H vorne oben, der C1V vorne rechts und der C1Z unten rechts von `(0,0,0)` nach `(1,1,1)` führt. Siehe:
../../../Samples/CV/StraightLine3D/straight_line3D.htm
../../../Samples/CV/StraightLine3D/straight_line3D.exe.
Die Gesetze des 2D-Chain-Codes gelten sinngemäß auch in 3D.
Deshalb beschränkt sich der folgende Text auf den 2D-Chain-Code.

Der Chain Code Algorithmus

ist die modernste Form der Gebietsbegrenzung. Wenn von einem Konturverfolger = contour finder = tracer = boundary tracing die Rede ist, dann ist heute meist der Chain Code Algorithmus gemeint.
Es gibt zwei Fassungen des Algorithmus:
1) mit `switch-case`: Länger aber leicht zu verstehen (wird im folgenden vorgestellt).
2) mit `Modulo 4` Operator: Sehr kompakt aber nicht so leicht zu verstehen:
../../Samples/CV/ChainCode/chain_code.htm.
Siehe auch die Vorlesung von Prof. Kovalevsky: **../../../Samples/CV/ChainCode/chaincode_kovalev_e.htm**
Erproben Sie auch die Samples:
../../../Samples/CV/StraightLine/straight_line.htm
../../../Samples/CV/StraightLine/straight_line.exe
../../../Samples/CV/StraightLine3D/straight_line3D.htm
../../../Samples/CV/StraightLine3D/straight_line3D.exe.

Aufgabenstellung: Gegeben sei ein Binärbild `b[ySize, xSize]` mit den Grauwerten `0`=Hintergrund und `1`=Vordergrund. In dem Bild soll es mindestens ein Vordergrundpixel mit dem Grauwert `1` geben, dessen obere linke Ecke als Startpunkt `c0[y0,x0]` und dessen linke Kante `c1v[y0,x0]` als vertikaler Startcrack in Richtung Süden dienen kann.
Durch den Start Richtung Süden ist festgelegt, dass der Vordergrund immer auf der linken Seite der gerichteten Cracks liegen muss.
Es muss weiterhin bekannt sein, ob im Vordergrund die 4er oder die 8er Nachbarschaft gilt.
Gesucht sei der vollständige Chain-Code der Berandung dieses Gebietes.

Der Algorithmus bei 4er Nachbarschaft

Taste zuerst nach dem Pixel links vor dem aktuellen Crack.
Ist es 0, dann biege nach links ab, fertig.
Ist es 1, dann taste nach dem Pixel rechts vor dem aktuellen Crack.
Ist es 0, gehe geradeaus, fertig.
Ist es 1, gehe nach rechts, fertig.
In Pseudocode:

```
    if ( Pixel links vorne == Hintergrund ) biege nach links ab;
else if ( Pixel rechts vorne == Hintergrund ) gehe geradeaus;
else biege nach rechts ab.
```

4er Fallunterscheidungen mit `switch ( last_crack )`, wenn der letzte Crack auf eine 0-Zelle`(x,y)` zeigt:


```
case 'e':     if (b[y-1,x  ] == 0) goto n;
         else if (b[y  ,x  ] == 0) goto e; else goto s;
```


```
case 's':     if (b[y  ,x  ] == 0) goto e;
         else if (b[y  ,x-1] == 0) goto s; else goto w;
```


```
case 'w':     if (b[y  ,x-1] == 0) goto s;
         else if (b[y-1,x-1] == 0) goto w; else goto n;
```


```
case 'n':     if (b[y-1,x-1] == 0) goto w;
         else if (b[y-1,x  ] == 0) goto n; else goto e;
```

Der 4er Algorithmus in C#, wenn Startpunkt `Point start` gegeben ist:

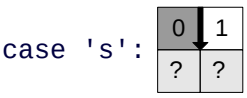
```
System.Text.StringBuilder cracks = new System.Text.StringBuilder();
cracks.Append( 's' );
Int32 x = start.X;
Int32 y = start.Y + 1;
Char last_crack = 's';
do
{ switch ( last_crack )
  { case 'e': if ( b[y-1,x  ] == 0 ) goto n; if ( b[y  ,x  ] == 0 ) goto e; goto s;
    case 's': if ( b[y  ,x  ] == 0 ) goto e; if ( b[y  ,x-1] == 0 ) goto s; goto w;
    case 'w': if ( b[y  ,x-1] == 0 ) goto s; if ( b[y-1,x-1] == 0 ) goto w; goto n;
    case 'n': if ( b[y-1,x-1] == 0 ) goto w; if ( b[y-1,x  ] == 0 ) goto n; goto e;
  }
  e: last_crack = 'e'; cracks.Append( 'e' ); x++; continue;
  s: last_crack = 's'; cracks.Append( 's' ); y++; continue;
  w: last_crack = 'w'; cracks.Append( 'w' ); x--; continue;
  n: last_crack = 'n'; cracks.Append( 'n' ); y--; continue;
} while ( x != start.X || y != start.Y );
```

Der Algorithmus bei 8er Nachbarschaft

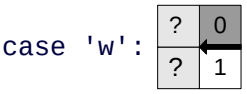
Taste zuerst nach dem Pixel rechts vor dem aktuellen Crack.
Ist es 1, dann biege nach rechts ab, fertig.
Ist es 0, dann taste nach dem Pixel links vor dem aktuellen Crack.
Ist es 1, gehe geradeaus, fertig.
Ist es 0, gehe nach links, fertig.
in Pseudocode:
    if (Pixel rechts vorne == Vordergrund) biege nach rechts ab;
else if (Pixel links vorne == Vordergrund) gehe geradeaus;
else biege nach links ab.

8er Fallunterscheidungen mit `switch ( last_crack )`,
wenn der letzte Crack auf eine 0-Zelle`(x,y)` zeigt:

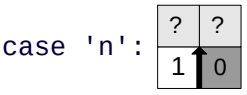

```
     if (b[y  ,x  ] == 1) goto s;
else if (b[y-1,x  ] == 1) goto e; else goto n;
```




```
     if (b[y  ,x-1] == 1) goto w;
else if (b[y  ,x  ] == 1) goto s; else goto e;
```




```
     if (b[y-1,x-1] == 1) goto n;
else if (b[y  ,x-1] == 1) goto w; else goto s;
```




```
     if (b[y-1,x  ] == 1) goto e;
else if (b[y-1,x-1] == 1) goto n; else goto w;
```

Der 8er Algorithmus in C#, wenn Startpunkt `Point start` gegeben ist:

```
System.Text.StringBuilder cracks = new System.Text.StringBuilder();
cracks.Append( 's' );
Int32 x = start.X;
Int32 y = start.Y + 1;
Char last_crack = 's';
do
{ switch ( last_crack )
  { case 'e': if ( b[y  ,x  ] == 1 ) goto s; if ( b[y-1,x  ] == 1 ) goto e; goto n;
    case 's': if ( b[y  ,x-1] == 1 ) goto w; if ( b[y  ,x  ] == 1 ) goto s; goto e;
    case 'w': if ( b[y-1,x-1] == 1 ) goto n; if ( b[y  ,x-1] == 1 ) goto w; goto s;
    case 'n': if ( b[y-1,x  ] == 1 ) goto e; if ( b[y-1,x-1] == 1 ) goto n; goto w;
  }
  e: last_crack = 'e'; cracks.Append( 'e' ); x++; continue;
  s: last_crack = 's'; cracks.Append( 's' ); y++; continue;
  w: last_crack = 'w'; cracks.Append( 'w' ); x--; continue;
  n: last_crack = 'n'; cracks.Append( 'n' ); y--; continue;
} while ( x != start.X || y != start.Y );
```

Der Algorithmus wurde auf der Basis eines Binärbildes formuliert.
Er funktioniert auch auf jedem Grauwertbild `b` mit einer Schwelle `threshold`, wenn alle Zugriffe auf die Pixel so geändert werden:
bei 4er Nachbarschaft: `if (b[...,...] == 0)` ersetzen durch: `if (b[...,...] < threshold)`
bei 8er Nachbarschaft: `if (b[...,...] == 1)` ersetzen durch: `if (b[...,...] >= threshold)`
Auf Farbbildern funktioniert er nur, wenn ein Farbkriterium formuliert ist, welches den Vordergrund vom Hintergrund eindeutig trennt. So ein Farbkriterium ist meist schwer zu finden.

Sie finden eine Bauanleitung zu diesem Thema unter **../../../C_CVCis/C2_ChainCode/CVCisChainCode_d.htm**,
oder Sie können eine lauffähiges EXE downloaden: **../../../C_CVCis/C2_ChainCode/CVCisChainCode.exe**.
Sie finden eine Vorlesung von Prof. Kovalevsky zum Chain Code unter: **../../../Samples/CV/ChainCode/chaincode_kovalev_e.htm**
Sie finden eine kompakte Formulierung des Kovalevsky-Algorithmus unter: **../../../Samples/CV/ChainCode/chain_code.htm**
und das passende Demo-Programm unter: **../../../Samples/CV/ChainCode/chain_code.exe**

Der Algorithmus am Bildrand

Die beiden Algorithmen scheitern beim Auftreffen des Chaincodes an einem der vier Bildränder,
weil sie nach nichtexistenten Pixeln außerhalb der Bildmatrix tasten.
Es sei `xSize == bmp.Width` = Spaltenzahl.
Es sei `ySize == bmp.Height` = Zeilenzahl.
Dann ist:
`linker Bildrand: x == 0; rechter Bildrand: x == xSize-1`
`oberer Bildrand: y == 0; unterer Bildrand: y == ySize-1`
Es gibt zwei Lösungen für das Bildrandproblem:
1. Man setzt willkürlich die Spalten `0` und `xSize-1`, sowie die Zeilen `0` und `ySize-1` auf Hintergrund und zerstört die dort vorhandene Bildinformation. Der Algorithmus kann jetzt den Bildrand nie erreichen.
2. Man kontrolliert die laufenden Indizes `x` und `y` und falls `x == 0` oder `x == xSize` oder falls `y == 0` oder `y == ySize` erzwingt man die am Bildrand notwendigen Richtungsänderungen ohne Pixeltests.

Konturverfolger bei 4er Nachbarschaft mit Bildrandbehandlung:

```
switch ( last_crack )
{ case 'e': if (x==xSize || b[y-1,x  ]==0) goto n; if (y==ySize || b[y  ,x  ]==0) goto e; goto s;
  case 's': if (y==ySize || b[y  ,x  ]==0) goto e; if (x==0     || b[y  ,x-1]==0) goto s; goto w;
  case 'w': if (x==0     || b[y  ,x-1]==0) goto s; if (y==0     || b[y-1,x-1]==0) goto w; goto n;
  case 'n': if (y==0     || b[y-1,x-1]==0) goto w; if (x==xSize || b[y-1,x  ]==0) goto n; goto e;
}
```

Konturverfolger bei 8er Nachbarschaft mit Bildrandbehandlung:

```
switch ( last_crack )
{ case 'e': if (x==xSize) goto n; if (y<ySize && b[y  ,x  ]==1) goto s; if (b[y-1,x  ]==1) goto e; goto n;
  case 's': if (y==ySize) goto e; if (x>0     && b[y  ,x-1]==1) goto w; if (b[y  ,x  ]==1) goto s; goto e;
  case 'w': if (x==0    ) goto s; if (y>0     && b[y-1,x-1]==1) goto n; if (b[y  ,x-1]==1) goto w; goto s;
  case 'n': if (y==0    ) goto w; if (x<xSize && b[y-1,x  ]==1) goto e; if (b[y-1,x-1]==1) goto n; goto w;
}
```

Startpunktfinder

Es ist leicht, in einem unbekannten Bild einen ersten Startpunkt für den ersten Crack-Code des ersten Gebiets zu finden. Man nimmt einfach die obere Ecke `c0(x0,y0)` des ersten auftauchenden vertikalen Cracks am Übergang von einer 0 zu einer 1.
Es sei `xSize == bmp.Width` = Spaltenzahl.
Es sei `ySize == bmp.Height` = Zeilenzahl.
Dann findet man den Startpunkt in einer Bildmatrix `b` der Größe `b[ySize,xSize]` so:

```
//linker Bildrand: x == 0; rechter Bildrand: x == xSize-1 (==bmp.Width -1)
//oberer Bildrand: y == 0; unterer Bildrand: y == ySize-1 (==bmp.Height-1)
for ( y0 = 0; y0 < ySize; y0++ )
{ if ( b[y0,0]==1 ) { x0 = 0; goto found; } /* if there is a "1" in column 0 */
  for ( x0 = 1; x0 < xSize; x0++ )
    if ( b[y0,x0-1]==0 && b[y0,x0]==1 ) goto found;
}
```

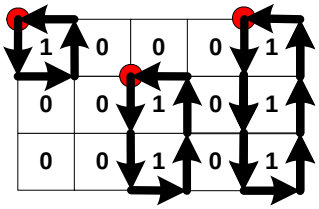
Schwerer ist das Auffinden neuer Startpunkte, nachdem ein, zwei oder mehrere Gebiete bereits durch Chain-Codes umrundet wurden. Hat man einen vertikalen Crack gefunden, dessen obere Ecke sich als Startpunkt eignet, dann muss man herausfinden, ob der Crack wirklich neu oder nicht vielmehr schon vorher einmal durchlaufen worden ist. Falls das der Fall ist, ist das zugehörige Gebiet bereits bekannt und umrundet. Man muss also alle im Laufe aller Konturverfolgungen gefundenen Süd-Cracks markieren, damit sie von weiterer Startpunktsuche ausgeschlossen werden.
Am einfachsten geschieht dies durch Anlegen der Matrix der vertikalen Cracks:
`Byte[,] c1v = new Byte[ySize,xSize+1]`, die auf `0` vorbesetzt sein werden muss und in die jeder gefundene vertikale Crack eingetragen wird, am besten in Form einer Nummer `j`,
welche die Zugehörigkeit des Cracks zu einem Code identifiziert. Sind mehr als 255 Chain-Codes zu erwarten, dann muß `j` und die Matrix `C1V` vom Typ `Int32` anstatt vom Typ `Byte` vereinbart werden.

```
/* Collecting all Chain-Codes ( 4-connectivity, no border pixels allowed ) */
Int32 x0, y0, x, y, i;
Byte[,] c1v = new Byte[ySize,xSize+1];
Byte j = 0; /* Change to type Int32 if j > 255 ! */
Char last_crack; for ( y0 = 0; y0 < ySize; y0++ )
  for ( x0 = 1; x0 < xsize; x0++ )
    if ( b[y0,x0-1]==0 && b[y0,x0]==1 && c1v[y0,x0]==0 )
    { x = x0; y = y0 + 1; i = 1; j++; c1v[y0,x0] = j;
      cracks.Remove( 0, cracks.Length ); cracks.Append( 's' ); last_crack = 's';
      do
      { switch ( last_crack )
        { case 'e': if ( b[y-1,x  ]==0 ) goto n; if ( b[y  ,x  ]==0 ) goto e; goto s;
          case 's': if ( b[y  ,x  ]==0 ) goto e; if ( b[y  ,x-1]==0 ) goto s; goto w;
          case 'w': if ( b[y  ,x-1]==0 ) goto s; if ( b[y-1,x-1]==0 ) goto w; goto n;
          case 'n': if ( b[y-1,x-1]==0 ) goto w; if ( b[y-1,x  ]==0 ) goto n; goto e;
        }
        e: last_crack = 'e'; cracks.Append( 'e' ); x++; continue;
        s: last_crack = 's'; cracks.Append( 's' ); c1v[y,x] = j; y++; continue;
        w: last_crack = 'w'; cracks.Append( 'w' ); x--; continue;
        n: last_crack = 'n'; cracks.Append( 'n' ); c1v[y,x] = j; y--; continue;
      } while ( x != x0 || y != y0 );
    }
```

Die Matrix der vertikalen Cracks und Label-Matrix

Der Algorithmus mit dem Startpunktfinder liefert eine `C1V`-Matrix mit nummerierten Gebietsbegrenzungen. Die Nummerierungen laufen von 1 bis Anzahl der Gebiete, geordnet nach der Reihenfolge, in der die Startpunkte gefunden wurden. Gebiet Nr. 1 hat den höchsten Startpunkt links, Startpunkt von Gebiet Nr. 2 liegt weiter rechts oder tiefer usw. Aus dieser `C1V`-Matrix lässt sich leicht eine `C2`-Matrix herstellen, die jedem Pixel eine Gebietsnummer gibt = Label-Matrix.

Beispiel:



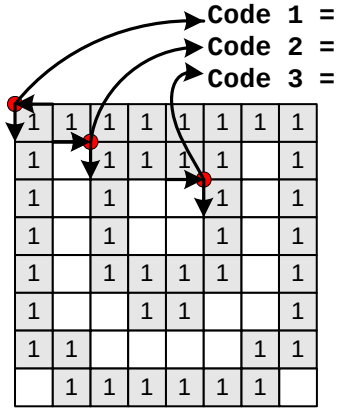

```
             10001         110022        10002
Bild b = 00101   C1V = 003322   C2 = 00302
             00101         003322        00302
```

Gebiet 1: `(0/0)senw` Perimeter = 4, Fläche = 1
Gebiet 2: `(4/0)sssennnw` Perimeter = 8, Fläche = 3
Gebiet 3: `(2/1)ssennw` Perimeter = 6, Fläche = 2

Innen- und Außenberandungen

Ein Gebiet hat immer genau eine äußere Berandung = Außenrand. Es kann eine, zwei oder mehrere innere Berandungen (Löcher) haben. Innerhalb dieser inneren Berandungen können weitere Gebiete liegen wie Inseln in einem See, die wiederum innere Berandungen haben usw.
Der Algorithmus ist zunächst blind für die Zuordnung von Berandungen zu Gebieten.
Er weiß z.B. nicht, ob und welche Codes innerhalb anderer Codes liegen.
Legt man jedoch den Startpunkt konsequent in die linke Ecke des linkesten Pixels in der obersten Zeile eines Gebiets und beachtet die Konvention, dass alle Chain-Codes immer mit einem Süd-Crack beginnen müssen und dass beim Umlauf das Gebiet immer links liegen der Begrenzung liegen muß, dann gilt:
a) Hat der letzte Crack eines Chain-Codes (letzter Crack vor der Wiedereinmündung in den Startpunkt) eine West-Richtung, dann codiert der Code eine Außenbegrenzung.
b) Hat der letzte Chain-Code eine Ost-Richtung, dann handelt es sich um eine Innenbegrenzung.
Diese Unterscheidung ist nicht erschöpfend, da noch unbekannt ist, welche Innenbegrenzung zu welcher Außenbegrenzung gehört.
Zu diesem Zweck muss man alle Chain-Codes durchlaufend nummerieren und man muss für jede Innenbegrenzung genau eine Außenbegrenzung finden, zu der sie gehört.
Gegeben sei:
1. ein Chain-Code einer Innenbegrenzung mit der CodeNr: `j`
2. beliebig viele Chain-Codes von Innen- und Außenbegrenzungen in beliebiger Reihenfolge mit den CodeNummern `k < j`.
Lösung:
Untersuche in der `C1V`-Matrix die Bildzeile `y0(j)`, in der der Start-Süd-Crack von `j` liegt, angefangen von `x0(j)+1` bis zum rechten Randspalte `xSize` in folgender Weise:
Bestimme alle diejenigen `k`, welche dieses Zeilenstück durchstoßen und speichere die `k` der Durchstoßstellen (egal ob nach Süden oder nach Norden) in einem Array. Sind alle Chain-Codes daraufhin untersucht, ob sie das Zeilenstück durchstoßen, dann muss man die Anzahl gleicher Marken `k` in der Zeile zählen.
Diejenigen `k`, die geradzahlig oft vorkommen sind uninteressant, mit denen hat `j` nichts zu tun.
Diejenigen `k`, die ungeradzahlig oft vorkommen, sind Hüllen von `j`, d.h. Code `j` liegt innerhalb von Code `k`.

Beispiel für eine C2-Matrix mit Löchern:



Code 1 hat einen Endcrack Richtung "w" und ist damit eine Außenrand.
Codes 2 und 3 haben einen Endcrack Richtung "e" und sind damit Löcher.

Rechts von (2/1) liegen zwei vertikale Cracks von 2 und einer von 1 (am äußersten rechten Rand).
Das bedeutet, dass 2 in 1 liegt.

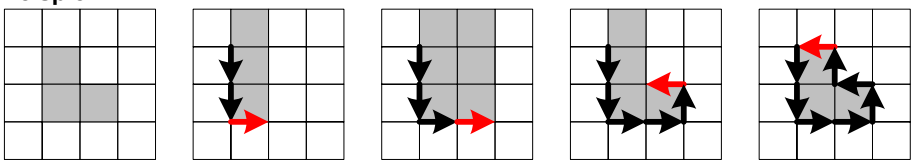
Rechts von (5/2) liegen ebenfalls zwei vertikale Cracks von 2 und einer von 1.
Das bedeutet, dass 3 mit 2 nichts zu tun hat, aber in 1 liegt.

Umfang, Fläche, Schwerpunkt, umschreibendes Rechteck

Umfang
Der Umfang eines Chain-Codes ist schlicht die Anzahl seiner Cracks.
Konsequenzen dieser Definition sind z.B.:
Der Umfang eines einzelnen Pixels ist 4. Sein "Radius" ist `1/2`.
Der Umfang eines Rasterkreises mit dem Radius `r` ist `8 * r`, also identisch mit dem Umfang des umschreibenden Quadrats und keineswegs `2*Math.Pi*r`.
Ein Gebiet hat immer genau einen äußeren Umfang, kann jedoch zusätzlich noch einen, zwei oder mehrere innere Umfänge besitzen (=Umfänge der Löcher).

Fläche

Die Fläche eines Chain-Codes ist gleich der Anzahl der von ihm eingeschlossenen Pixel.
Diese Anzahl ist negativ, wenn der Code eine Innenberandung (=Loch) codiert.
Die Formel zur Flächenberechnung ist die der Berechnung der Fläche eines Polygons:
`F` = Summe über alle `i` von `(x[i] - x[i+1]) * (y[i] + y[i+1]) / 2`.
Fasst man die beiden Enden eines jeden Cracks als zwei aufeinander folgende Ecken eines Polygons auf, so ergibt sich:
für West-Cracks: `x[i] - x[i+1] = 1 und y[i] = y[i+1] = y` = Zeilennummer → `F = F + y;`
für Ost-Cracks : `x[i] - x[i+1] = -1 und y[i] = y[i+1] = y` = Zeilennummer → `F = F - y;`
für Süd-Cracks : `x[i] - x[i+1] = 0` → `F = F;`
für Nord-Cracks: `x[i] - x[i+1] = 0` → `F = F;`
Zusammengefasst ergibt sich eine einfache Rechenvorschrift für die Fläche eines Crack-Codes:
F = Summe aller Zeilennummern aller Ost-Cracks minus Summe aller Zeilennummern aller West-Cracks.
Die Fläche eines Chain-Codes ergibt sich damit vorzeichenrichtig, d.h. positiv bei Außenbegrenzungen und negativ bei Innenbegrenzungen.
Die Fläche eines Gebiets ist die Summe der positiven Fläche der Außenkontur plus der negativen Flächen der Löcher.
Beispiel:



Original
area = 0

east: y = 3
area += y
area = 3

east: y = 3
area += y
area = 6

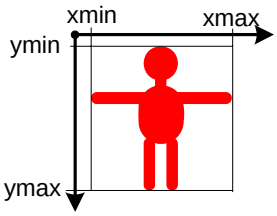
west: y = 2
area -= y
area = 4

west: y = 1
area -= y
area = 3

Schwerpunkt

Zur Berechnung des Schwerpunktes `S = (xs,ys)` eines Chain-Codes stelle man sich alle Pixel `c2[y,x]` im Inneren als auf ihren Pixelmittelpunkt `c0[y+0.5,x+0.5]` geschrumpfte Elemente der Masse 1 vor.
`xs` = Summe aller x-Koordinaten der geschrumpften Elemente dividiert durch ihre Anzahl.
`ys` = Summe aller y-Koordinaten der geschrumpften Elemente dividiert durch ihre Anzahl.
Anzahl der geschrumpften Elemente = Anzahl der Pixel = positive Fläche des Chain-Codes.
Vorsicht: Bei der Berechnung der Schwerpunkte von Gebieten mit Innenkonturen dürfen die geschrumpften Elemente der Löcher-Pixel nicht mitsummiert werden.

Umschreibendes Rechteck



Das umschreibende Rechteck eines Gebietes umschließt dessen äußere Begrenzung (= Außen-Chain-Code) mit 4 achsparallelen Geraden. Diese Geraden sind Verlängerungen der Cracks mit den niedersten und höchsten Spaltennummern (`xmin` und `xmax`) und den niedersten und höchsten Zeilennummern `(ymin,ymax)`. Ein umschreibendes Rechteck ist vollständig definiert durch Angabe seiner linken oberen Ecke `Pmin=(xmin,ymin)` und seiner rechten unteren Ecke `Pmax=(xmax,ymax)`.
Man ersetzt die realen (manchmal kompliziert begrenzten) Gebiete durch ihre umschreibenden Rechtecke, wenn schnelle (jedoch nicht unbedingt genaue) Antworten gefordert werden, wie z.B.:
Frage: Liegt ein Mauszeiger mit der Koordinate `x,y` innerhalb eines Gebietes ?
Antwort: Nein, wenn `x<xmin` oder `x>xmax` oder `y<ymin` oder `y>ymax`.
Frage: Überlappen sich zwei Gebiete G1 und G2 ?
Antwort: Nein, wenn ihre umschreibenden Rechtecke nicht überlappen.
Frage: Wo liegt der Mittelpunkt `(xm, ym)` eines Gebiets ?
Antwort: `xm = (xmax+xmin)/2, ym = (ymax+ymin)/2`

Die Berechnungen in Form einer Subroutine

```
void crack_calculations( int x0, int y0, Byte[] crack, int no )
{ /* x0, y0 = starting point; crack[0 .. no-1] contains e,s,w,n characters; */
  int x, y, xmin, ymin, xmax, ymax, i = 0, peri = 0; area = 0;
  float sx = 0, sy = 0;
  x = xmin = xmax = x0;
  y = ymin = ymax = y0;
  do
  { if ( i++ > no ) break;
    switch ( cracks[i-1] )
    { case 'e': x++; if (x > xmax) xmax=x; peri++; area += y; sx += y*(x+0.5); sy += y * y/2; break;
      case 's': y++; if (y > ymax) ymax=y; peri++; break;
      case 'w': x--; if (x < xmin) xmin=x; peri++; area -= y; sx -= y*(x+0.5); sy -= y * y/2; break;;
      case 'n': y--; if (y < ymin) ymin=y; peri++;;
    }
  } while ( x != x0 || y != y0 );
  sx /= area; sy /= area;
  /* here is a good place to print perimeter, area, sx, sy, xmin, ymin, xmax, ymax */
}
```