

## Code Samples: Kovalevsky's Crack Code Algorithm in C

Copyright © by V. Kovalevsky, last update: 02-03-2006

Das Projekt soll die neue Funktion "TraceUni" testen. Diese verfolgt die Begrenzungen in einem 2D-Binärbild und erzeugt den Crackcode. Das Besondere an dieser Funktion, dass sie die singulären 0-Zellen verschieden bearbeitet, je nach dem, ob sie dem Objekt (Vordergrund) oder dem Hintergrund zugewiesen wurden.

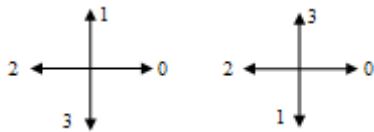
Es gibt drei Möglichkeiten, diese Zuweisung zu realisieren:

1. Die globale Variable "AllPoints" gleich 0 setzen. Dann gehören alle singulären 0-Zellen (auch "Punkte" genannt) dem Hintergrund.
2. Die globale Variable "AllPoints" gleich 2 setzen. Dann gehören alle singulären 0-Zellen (auch "Punkte" genannt) dem Objekt.
3. Die globale Variable "AllPoints" gleich 1 setzen. Dann wird der Benutzer an jedem singulären Punkt gefragt, ob dieser Punkt dem Objekt gehören soll. Falls ja, dann wird im Bild, wo jedem Pixel ein Byte zugewiesen ist, das Bit 0 gesetzt. Sonst wird nur das Bit 7 benutzt, um zu definieren, welche Pixel dem Objekt (Byte=128) und welche dem Hintergrund (Byte=0) gehört.

Das Bild wird in diesem Projekt als ein globales Array vom Typ "unsigned char" definiert und mit Konstante 0 und 1 initialisiert. In "main" werden die "1" in das Bit 7 verschoben, d.h. durch 128 ersetzt.

Die Funktion "Search" durchsucht das Bild Zeile für Zeile. Sobald sie einen Übergang vom Hintergrund- zum Objektpixel findet, prüft sie ob der dazwischen liegende senkrechte Crack bereits (von "TraceUni") besucht wurde. Wenn nicht, wird an diesem Crack die Funktion "TraceUni" gestartet.

Diese verfolgt die Begrenzung in solch einer Richtung, dass das Objekt auf der negativen Seite der Verfolgungsbewegung liegt. Hier werden anstatt "links" und "rechts" die Begriffe "Negative" und "Positive" benutzt. Dies hat den Vorteil, dass die Funktion "TraceUni" und die zu ihr gehörigen Konstantfelder "Negative[4]", "Positive[4]" und "step[4]" sowohl im mathematischen Koordinatensystem als auch in dem System der Computergrafik ohne jegliche Änderung anwendbar sind. "



"Positive" bedeutet einfach "liegt an der gleichen Seite, wie die positive Y-Achse relativ zu der positiven X-Achse". Die Begriffe "links" und "rechts" sollen dagegen beim Übergang zum anderen System vertauscht werden. Bei der Version mit "positiv" und "negativ" müssen nur die codierten Richtungen mit dem Koordinatensystem übereinstimmen: Richtung 1 entspricht immer der positiven Y-Achse.

Mathematik      Computergrafik

Das Objekt soll auf der negativen Seite der Verfolgungsrichtung liegen, damit die einfachste Berechnung des Flächeninhalts (siehe die entsprechende einzige Zeile in "TraceUni") positive Werte für Objekte und negative Werte für Löcher bringt. Dadurch kann der Flächeninhalt eines Objekts mit Löchern besonders einfach berechnet werden:  $F = F(\text{Ob}) + ?F(\text{Loch})$ .

Das Neue in dieser Funktion ist die Benutzung der Variablen

```
int ObjectPoint=AllPoints==2 || AllPoints==1 && image[P.X+NX*P.Y] & 1;
```

Diese wird nur für singuläre Punkte ausgewertet. Ein singulärer Punkt gehört zum Objekt, (ObjectPoint=TRUE), wenn entweder alle solche Punkte zum Objekt gehören (AllPoints==2) oder die Entscheidung wurde dem Benutzer überlassen (AllPoints==1) und er hat diesen Punkt mit "1" markiert (image[P.X+NX\*P.Y] & 1>0).

Die Logik des doppelten "if"-Ausdrucks ist nicht wesentlich komplizierter geworden:

```
if      ( PosFor && ( NegFor || ObjectPoint)) direction=(direction+1) % 4; //pos.
turn
else if (!NegFor && (!PosFor || !ObjectPoint)) direction=(direction+3) % 4;
//negative turn
```

Hier bedeuten die logischen Variablen folgendes:

PosFor - das Pixel auf der positiven Seite der alten Richtung gehört zum Objekt;

NegFor - das Pixel auf der negativen Seite der alten Richtung gehört zum Objekt;

Der Ausdruck (direction %4) bedeutet den Rest der Division durch 4. Das ist die schnellste und einfachste Methode, eine zyklische Änderung der Richtung zu realisieren. Dies bedeutet, nach der Richtung 3 kommt die Richtung 0.

Diese relativ einfache Lösung habe ich mit Hilfe des von mir vor vielen Jahren erfundenen Diagramms für die Minimierung logischer Funktionen gefunden. Hier ist das Beispiel für die logische Funktion.

PosTurn(ObjectPoint, NegFor, PosFor) von drei logischen Argumenten.

0	0	1	0
0	0	1	1

Jedes Kästchen der Tabelle entspricht einer Konjunktion (UND) von drei logischen Variablen. So z. B. das untere rechte Kästchen entspricht dem logischen Ausdruck `ObjectPoint && PosFor && !NegFor`. Jedem der gegebenen logischen Argumenten "ObjectPoint", "NegFor" und "PosFor" entspricht eine Symmetrieachse: Eine lange Achse, wie "PosFor", ist die Achse der ganzen Tabelle, eine kürzere Achse ist die Achse einer Hälfte der Tabelle; eine noch kürzere (in diesem Beispiel sind keine noch kürzeren) wäre es die Achse eines Viertel usw.

Wenn die gesuchte logische Funktion, z.B. PosTurn (positive Wendung) in zwei Kästchen den Wert 1 hat und diese Kästchen liegen symmetrisch relativ zu einer der Achsen, dann heißt dies, dass die zwei Ausdrücke, die diesen Kästchen entsprechen, können vereinigt werden und dabei wird die Variable, die der Symmetrieachse entspricht, weggelassen. So z.B. sind die zwei Kästchen rechts in der unteren Reihe symmetrisch relativ zu der Achse "NegFor". Daraus folgt:

```
ObjectPoint && PosFor && NegFor. || ObjectPoint && PosFor && !NegFor ==
ObjectPoint && PosFor.
```

Aus zwei Ausdrücken mit jeweils drei Termen ist ein Ausdruck mit nur zwei Termen geworden; also aus sechs Termen nur zwei.

Ähnlich, ergeben die zwei Kästchen in der dritten Spalte:

```
!ObjectPoint && PosFor && NegFor. || ObjectPoint && PosFor && NegFor == PosFor &&
NegFor.
```

Als Ergebnis kann die Funktion PosTurn so dargestellt werden:

```
PosTurn=ObjectPoint && PosFor || PosFor && NegFor.
```

Jetzt kann man "PosFor" ausklammern:

```
PosTurn=PosFor && (ObjectPoint || NegFor).
```

Genau dieser Ausdruck wird durch das erste "if" realisiert:

```
if (PosFor && (NegFor || ObjectPoint) )
direction=(direction+1) % 4; // rest of division=pos. turn
```

Ähnlich wurde der Ausdruck für das zweite "if" gefunden. Wenn keiner dieser Ausdrücke wahr ist, wird keine Änderung der Richtung vorgenommen; die Variable "direction" bleibt unverändert.

This code has been developed with Visual C++.NET 2003.

Create a new Visual C++ Win32 Console Project project "crackcode". Click Project from the main menu. A drop down menu opens. Click Properties. A dialog box "crackcode Property Pages" appears. Choose C/C++ → Precompiled Headers on the left side. Go to line Create/Use Precompiled Header. Switch to Not Using Precompiled Headers.

Delete the files `stdafx.cpp` and `stdafx.h` from the project. Clear any prefabricated content from `crackcode.cpp` and replace it by the following code:

```
// File TestTrace.cpp. Zum Testen des universellen Algorithms mit "AllPoints"=0,1,2..
// Mathematische Standardkoordinaten im Bild und in "TraceUni".
// Das Objekt liegt auf der negativen Seite, damit die Fläche positiv wird.
// Markiert werden nur die +Y-Cracks.
// Bit_Belegung: 1=Point; 2=X-Crack; 4=Y-Crack; 128=Pixel.
```

```
#include "stdio.h"
#include "conio.h"
typedef struct {int X,Y;} iPOINT;
typedef struct {int X,Y,last,Area;} LOOP;
unsigned char CC[1000];
LOOP Object[100];
int iCC=0, iOb=0;
iPOINT Negative[4]= {{0,-1}, { 0,0}, {-1, 0}, {-1,-1}};
iPOINT Positive[4]= {{0, 0}, {-1,0}, {-1,-1}, { 0,-1}};
iPOINT step [4]= {{1, 0}, { 0,1}, {-1, 0}, { 0,-1}};
iPOINT VoxSingP={3,3};
int const foreground=128, background=0, NX=6, NY=NX, val=128;
int const Bit0=1, Bit2=4, Bit7=foreground;
int AllPoints;
```

```

// Achtung! Damit die Funktion TraceUni einfach bleibt, soll man die äusseren
// Zeilen und Spalten mit Nullen füllen. Die "1" werden sofort durch 128 ersetzt,
// damit Bits für Markierungen frei bleiben.
// Bitbelegung: Bit0=0-Zelle, Bit1=waagerechter Crack; Bit2=senkrechter Crack;
//                Bit7=Pixel.
unsigned char image[NX*NY]={0,0,0,0,0,0,
                           0,1,1,1,1,0,
                           0,1,0,1,1,0,
                           0,1,1,0,1,0,
                           0,1,1,1,1,0,
                           0,0,0,0,0,0};

int SingularPoints()
{ int cnt=0, deb=1, act, left, bot, botleft;
  for (int y=0; y < NY; y++) //=====
  { botleft=left=0;
    for (int x=0; x < NX; x++) //=====
    { act=image[x+NX*y] & Bit7; bot=image[x+NX*(y-1)] & Bit7;
      int singular=act && !left && botleft && !bot ||
                !act && left && !botleft && bot;
      if (deb && x==3 && y==3) printf("x=%d y=%d left=%d act=%d botleft=%d bot=%d
sing=%d\n",
                                   x, y, left, act, botleft, bot, singular);
      if (y > 1 && y < NY-1 && x > 1 && x < NX-1 && singular)
      { printf("Soll der singulaere Punkt (%d,%d) zum Objekt gehoeren? ",x,y);
        int repl=getche(); printf("\n");
        if (repl=='j')
        { image[x+NX*y] |=Bit0; cnt++;
        }
      }
      left=act; botleft=bot;
    } //===== end for (x... =====
  } //===== end for (y... =====
  printf("SingularPoints: %d singulaere Punkte dem Objekt zugewiesen\n",cnt);
  return cnt;
} //***** end SingularPoints *****

void PrintCC()
{ printf("\n The CrackCode contains %d objects\n",iOb);
  const int CrackInLine=20;
  for (int i=0; i<= Object[i].last; ib++)
  { printf("%2d; ",CC[ib]); cnt++;
    if (cnt % CrackInLine == CrackInLine-1) printf("\n");
  }
  printf("\n");
}

int TraceUni(int x, int y)
{ //Universell, positive Fläche, 0-Zellen durch "AllPoints" oder Markierung
// "x, y" sind Standardkoordinaten des Startpixels. P in Standardkoordinaten.
iPOINT Neg, P, Pos; // P in top. Koordinaten.
int Area=0, DEB=0, deb=0, i=0, direction;

Object[iOb].X=x; Object[iOb].Y=y; Object[iOb].Area=0; // Object ist global
P.X=x; P.Y=y; direction=1;
//CC[iCC++]=direction;
if (DEB) printf("TraceUni startet bei (%d,%d) in Richtung %d\n",x,y,direction);
do
{ Neg.X=P.X+Negative[direction].X; // Neg is the "negative" pixel
  Neg.Y=P.Y+Negative[direction].Y;
  Pos.X=P.X+Positive[direction].X; // Pos is the "positive" pixel
  Pos.Y=P.Y+Positive[direction].Y;
  int PosFor=(image[Pos.X+NX*Pos.Y] & Bit7)==foreground;
  int NegFor=(image[Neg.X+NX*Neg.Y] & Bit7)==foreground;
  int ObjectPoint=AllPoints==2 || AllPoints==1 && image[P.X+NX*P.Y] & 1;

  if (PosFor && (NegFor || ObjectPoint)) direction=(direction+1)%4; //pos. turn
  else if (!NegFor && (!PosFor || !ObjectPoint)) direction=(direction+3)%4; //neg. turn

  if (DEB) printf("x=%d y=%d dir=%d Neg=(%d, %d) Pos=(%d, %d)\n",
                 P.X,P.Y,direction,Neg.X,Neg.Y,Pos.X,Pos.Y);
}

```

```

    if (direction==1)
    { image[P.X+NX*P.Y] |=4; // Markierung
      if (deb) printf("***** Markierung bei (%d,%d)\n",P.X,P.Y);
    }

    CC[iCC++]=direction; // Eintrag in den CrackCode

    Object[iOb].Area+=P.Y*step[direction].X; // Berechnung des Flächeninhalts

    P.X=P.X+step[direction].X; //a move in the new direction
    P.Y=P.Y+step[direction].Y; //a move in the new direction
    i++;
    if (i > 30) return -1;
  } while( P.X!=x || P.Y!=y);
  Object[iOb++].last=iCC-1;
  return Object[iOb-1].Area;
} // end TraceUni

int Search(int Area[])
{ int cnt=0, deb=0, lab, val, valOld, x0, y0;
  for (y0=0; y0 < NY; y0++) //===== Zeilen =====
  { valOld=0;
    for (x0=0; x0 < NX; x0++) //===== Spalten =====
    { val=image[x0+NX*y0] & Bit7; // Bit7=128 ist global
      lab=image[x0+NX*y0] & Bit2; // Bit2=4 ist global
      if (deb) printf("Search: Pix=(%d,%d) valOld=%3d val=%3d lab=%3d\n",
        x0,y0,valOld,val,lab);
      if (val && !valOld && !lab)
      { Area[cnt++]=TraceUni(x0,y0);
      }
      valOld=val;
    } //===== end for (x... =====
  } //===== end for (y... =====
  return cnt;
} //***** end Search *****

void main()// Trace und TraceNew funktionieren bei y0=1,2,3
{ int Area[20], cnt=0, deb=0, i;
  for (i=0; i < NX*NY; i++) if (image[i]==1) image[i]=foreground;
  int Error=0;
  do
  { printf("Waehle die Objektzugehoerigkeit der singulaeren 0-Zellen: 0, 1 oder 2: ");
    AllPoints=getche()-48; printf("\n");
    if (AllPoints2) printf("Falscher Wert=%d\n",AllPoints);
  } while(AllPoints2);
  printf("AllPoints=%d\n",AllPoints);

  if (AllPoints==1) SingularPoints();

  cnt=Search(Area); // Sucht Stratpunkte und startet TraceUni

  printf("\n");
  for (i=0; i < cnt; i++)
    printf("Das %dte Objekt hat Flaeche=%d\n",i,Area[i]);
  PrintCC();
}

```

Click Debug from the main menu. A drop down menu opens. Click Start Without debugging Ctrl+F5.

### Recommended experiments:

- 1) Try out the program with "Objektzugehoerigkeit" 1. and try out the answers "j" and "n" when asked: "Soll der singulaere Punkt (3,3) zum Objekt gehoeren?".
- 2) Create a new and bigger sample matrix image[NX\*NY] and adjust the global values of NX and NY accordingly.