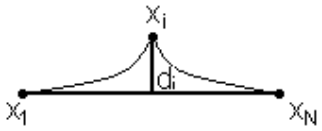
Aproximarea poligonală a frontierelor

Fie x_1 , x_2 , … , x_N pixelii unei frontiere, care poate fi o curba deschisa oarecare sau un contur inchis (conturul unei regiuni)



d_i – distanța de la punctul x_i la segmentul de dreaptă (x_1-x_N)

Definiţie: d_i este eroarea de aproximare a frontierei prin segmentul (x_1-x_N) în pixelul x_i

Definiţie: Eroarea maximă de aproximare a frontierei prin segmentul (x_1-x_N) este $E_{\max} = \max_{2 \le i \le N-1}(d_i)$

In continuare sunt prezentate două metode de poligonalizare:

a. Prin divizarea recursivă a frontierei până la nivel de segmente care se pot aproxima prin segmente de dreaptă
b. Prin unirea iterativă a punctelor frontierei

Poligonalizarea prin divizarea recursiva a frontierei

Algoritmul se aplica unei frontiere deschise. Dacă frontiera este un contur închis, punctele x_1 şi x_N se aleg astfel încât să fie situate la distanța maximă.

Se efectueaza poligonalizarea separat, pentru fiecare parte a conturului.

Dacă x_1 şi x_N sunt puncte extreme, se alege pixelul y, aflat la distanță maximă de segmentul (x_1-x_N). Frontiera se divide în (x_1-y) şi (y-x_N). Pentru fiecare parte se verifică dacă poate fi aproximată printr-un segment de dreaptă. Daca nu, procesul se repetă pentru fiecare parte până când E_{max} < *prag* pentru segmentul curent.

Algoritm:

Intrare: lista adreselor pixelilor de frontieră

Ieşire: lista vârfurilor liniei poligonale care aproximează frontiera

O implementare in C:

```
typedef struct PIXEL {int, x,y;} PIXEL;
typedef struct CEL { PIXEL p; struct CEL * urm} CEL;
typedef struct LISTV { CEL * start; CEL * end;} LISTV;

void Poligonalizare1(PIXEL * pixeli, LISTV * varfuri, int i1, int
i2, double prag)
{ // i1 si i2 sunt indicii in vectorul “pixeli” care delimiteaza
frontiera
        int i, j, imax;
        double d = 0, dmax;
        boolean gata = true;

        for (i=i1+1; i<=i2-1 ; i++)
        {       // calculeaza distanta de la pixelul i la
segmentul determinat de pixelii i1 si i2
                d = Distanta(pixeli, i1, i2, i);
                if (d > prag)
                {       gata = false;
```

```
                    break;
                }
            }
            if (gata)
                return;//toti pixelii frontierei sunt la o
distanta mai mica decat pragul
            dmax = d;
            imax = i;
            for (j=i+1; j<=i2-1; j++)
            {
                d = Distanta(pixeli, i1, i2, j);
                if (d > dmax)
                {   dmax = d;
                    imax = j;
                }
            }
            Poligonalizare1(pixeli, varfuri, i1, imax, prag);
            AdaugaVarf (varfuri, pixeli[imax]);
            Poligonalizare1(pixeli, varfuri, imax, i2, prag);
    }
```

Apel:

```
LISTV *Polig;
PIXEL *Frontiera;
int nfront; // numarul pixelilor de frontiera
double Prag;

AdaugaVarf (Polig, Frontiera[0]);
Poligonalizare1(Frontiera, Polig, 0, nfront-1, Prag);
AdaugaVarf (Polig, Frontiera[nfront-1]);
```

Dacă pragul este mic, poligonul va avea un număr mare de laturi.

Principalul avantaj al metodei: detectează punctele de inflexiune ale frontierei, care devin vârfuri ale liniei poligonale.

Poligonalizarea prin unirea iterativa a punctelor frontierei

Se pleacă din x_1 şi se avansează în lista de pixeli până într-un punct x_i, în care eroarea de aproximare a frontierei x_1-x_i printr-un segment de dreaptă depăşeşte pragul. Atunci se memorează x_{i-1} în lista vârfurilor liniei poligonale. Se repetă procedeul plecând din x_{i-1}.
Dacă frontiera este închisă, se alege x_1 ca punctul de inflexiune cel mai proeminent.

Algoritm:

Intrare: lista pixelilor de frontieră
Ieşire: lista vârfurilor liniei poligonale care aproximează frontiera

O implementare in C:

```
void Poligonalizare2(PIXEL * pixeli, LISTV * varfuri, int start,
int n, double prag)
{ // n este este indicele maxim in vectorul “pixeli”
            int i, end;
            double d, dmax;
            end=start+1;
      do{
            end++; dmax =0;
            for (i=start+1; i<end; i++)
```

```
            {  // calculeaza distanta de la pixelul i la segmentul
determinat de pixelii (start, end)
               d = Distanta(pixeli, start, end, i);
               if (d > dmax)
                    dmax = d;
            }
        } while (dmax <= prag && end != n);
        if (end < n)
        {  AdaugaVarf (varfuri, pixeli[end-1]);
           Poligonalizare2(pixeli, varfuri, end-1, n, prag);
        }
        else
        AdaugaVarf (varfuri, pixeli[n]); // memoreaza ultimul
punct de frontiera in lista de
                                                //varfuri
}
```

Apel:

```
AdaugaVarf (Polig, Frontiera[0]);
Poligonalizare2(Frontiera, Polig, 0, nfront-1, Prag);
```

Principalul dezavantaj al metodei: vârfurile poligonului nu coincid cu punctele de inflexiune ale frontierei. Problema se rezolvă combinând cele două metode.