Dacă însă utilizăm culori închise de la Negru (0,0,0) la Roşu, respectiv la Albastru, atunci cu ochiul stâng (lentila Roşie) selectăm (vedem) imaginea roşie iar cu ochiul drept selectăm (vedem) imaginea albastră - menţine culoarea activă (vezi Figura 53). În aceaastă situaţie, cu ochiul sâng nu se vede imaginea albastră (este neagră prin lentila roşie) iar cu cel drept nu se vede cea roşie (este neagră prin lentila albastră - taie culoarea opusă). Din cauza aceasta, anagifele realizate cu culori închise sunt mai întunecate (vezi Figura 53) şi se văd corect sau cu ochelarii inversaţi sau trebuie ţinut cont de acest fapt la alegerea imaginilor sursă pentru ca acestea sa fie cele corecte. Deci, dacă în prima variantă imaginiea albastră este în/din stânga şi cea roşie în/din dreapta, în cea de-a doua variantă imaginile stau invers: imaginiea roşie este în/din stânga şi cea albastră în/din dreapta, sau la ochelari inversăm lentilele (culorile).

4 Reprezentarea si prelucrarea imaginilor digitale

4.1 Reprezentarea imaginilor prin funcţii şi cuvinte picturale

În cele ce urmează, sunt abordate în special probleme referitoare la imaginile de tip 3 şi 4 (definite în [[27]) şi anume câteva metode de reprezentare a imaginilor prin *linii şi curbe* respectiv printr-o *mulţime de puncte critice*, prin desene descrise de Π-cuvinte (şiruri de comenzi de deplasare a cursorului pe cele patru direcţii *r, u, l* şi *d*). De asemenea este prezentată şi o modalitate de conversie a unui desen dintr-o formă în alta. Pentru interpolare sunt utilizate curbele Bezier, datorită faptului că acestea au anumite proprietăţi care se potrivesc cu desenele descrise prin *cuvinte picturale* (Π-cuvinte).

O imagine poate fi modelată printr-o funcţie de două variabile definită în planul ecranului (pe matricea de puncte din fereastra de lucru, *ViewPort*). Valoarea funcţiei într-un punct va reprezenta nuanţa (culoarea) acelui pixel de

pe ecran. Aceasta înseamnă că funcţia ia valori din mulţimea culorilor posibile ale punctelor ecranului, deci o astfel de funcţie este nenegativă şi mărginită.

Dacă fereastra ecran este $[u_1,u_2]\times[v_1,v_2]$ (Figura 54), punctele P_{ij} din fereastră au coordonatele limitate astfel: $v_1 \le i \le v_2$ şi $u_1 \le j \le u_2$.

Fie S_{ij} pătratul definit astfel :

$$j-1 \le x \le j \text{ şi } i-1 \le y \le i .$$

În pătratul S_{ij} funcţia va lua o valoare constantă egală cu codul culorii punctului P_{ij}. O astfel de funcţie o vom numi *funcţie picturală*. Un caz particular important îl reprezintă funcţiile cu doar două valori 0 şi 1 pentru imagini alb-negru.

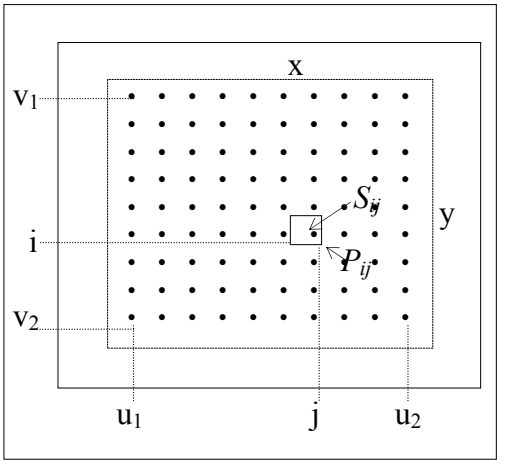

Figura 54 – *Fereastră ecran*

Funcţiile picturale sunt utilizate în numeroase probleme de prelucrare a imaginilor (codificare, aproximare, filtrare, restaurare, scalare, etc.) prin diverse operaţii aplicate acestora .

Descrierea unei imagini printr-o *funcţie picturală* este echivalentă cu descrierea acesteia prin *cuvinte picturale*.

Demonstraţia acestei afirmaţii o vom efectua prin precizarea modului de descriere a unei imagini formată din pătrate (S_{ij}), utilizând cuvinte de descriere prezentate în continuare.

Pentru imagini alb-negru este suficient un limbaj asemănător cu cel de descriere a desenelor discontinue ($\Pi_{\prec}^{*}$, unde $\Pi_{\prec}=\{r,u,l,d,\uparrow,\downarrow\}$), doar că primitivele de desenare nu vor mai fi segmente de lungime unitate ci pătrate de latură unitate. De exemplu, $\Pi_{\prec}$-cuvântul $r^5d^4l^4u^4\uparrow dr\downarrow r$ descrie imaginea din **Error! Reference source not found.**.

Pentru o funcţie picturală care ia valori din mulţimea {*alb,negru*} există şi putem construi un $\Pi_{\prec}$-cuvânt care să descrie aceeaşi imagine.

Desigur că există mai multe posibilităţi de definire a $\Pi_{\lessdot}$-cuvintelor echivalente cu o funcţie picturală. Teoretic este suficient să punem în evidenţă un singur $\Pi_{\lessdot}$-cuvânt pentru o funcţie picturală oarecare. Practic însă, ne interesează şi *complexitatea descrierii*, adică o descriere minimă a unei imagini.

În cele ce urmează vom prezenta o primă metodă de construcţie a unui cuvânt pictural pornind de la o funcţie picturală dată.

$\Pi_{\lessdot}$-cuvântul echivalent este $w=w_{v1}w_{v1}...w_i...w_{v2-1}w_{v2}$, unde prin w_i înţelegem subcuvântul de descriere al liniei i, iar funcţia *Redu* este definită astfel: $Redu(\alpha\uparrow\downarrow\beta)=\alpha\beta$; $\alpha, \beta \in \Pi_{\lessdot}^{*}$. $\Pi_{\lessdot}$-cuvântul w_i de descriere a liniei i ($v_1 \le i \le v_2$) este definit astfel :

$$w_i = \begin{cases} \alpha_{i,u1}\alpha_{i,u1+1}...\alpha_{i,j}...\alpha_{i,u2-1}\alpha_{i,u2} & \text{pentru } i \text{ impar,} \\ \beta_{i,u2}\beta_{i,u2-1}...\ \beta_{i,j}...\beta_{i,u1+1}\beta_{i+1,u1} & \text{pentru } i \text{ par.} \end{cases}$$

Subcuvintele de colorare (α_{ij}, β_{ij}) a pătratelor S_{ij} sunt definite după cum urmează:

$$\alpha_{ij} = \begin{cases} \uparrow r\downarrow & \text{pentru } j<u_2 \text{ şi } f(S_{ij}) = alb, \\ r & \text{pentru } j<u_2 \text{ şi } f(S_{ij}) = negru, \\ \uparrow d\downarrow & \text{pentru } j=u_2 \text{ şi } f(S_{ij}) = alb, \\ d & \text{pentru } j=u_2 \text{ şi } f(S_{ij}) = negru; \end{cases}$$

$$\beta_{ij} = \begin{cases} \uparrow l\downarrow & \text{pentru } j>1 \text{ şi } f(S_{ij}) = alb, \\ l & \text{pentru } j>1 \text{ şi } f(S_{ij}) = negru, \\ \uparrow d\downarrow & \text{pentru } j=1 \text{ şi } f(S_{ij}) = alb, \\ d & \text{pentru } j=1 \text{ şi } f(S_{ij}) = negru. \end{cases}$$

Se observă că $\Pi_{\lessdot}$-cuvântul de descriere, parcurge matricea de pătrate care reprezintă imaginea descrisă de funcţia f şi colorează acele pătrate pentru care valoarea funcţiei picturale este *negru*. Ordinea de parcurgere este cea din Figura 55.

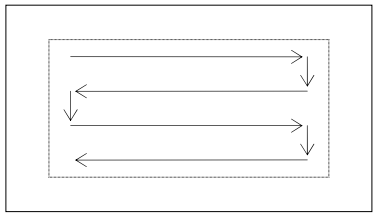

Figura 55 - Parcurgere

Invers, valoarea funcţiei picturale corespunzătoare pătratului S_{ij} este dată de poziţia creionului (↑sau↓) înainte de colorarea pătratului S_{ij} .

Pentru descrierea unei imagini color putem utiliza comenzi din mulţimea $\Pi \cup C$. Pentru a descrie o imagine formată din pătrate S_{ij} de tipul celor de mai sus (vezi Figura 54), vom preciza şi culoarea pătratului care se desenează (după deplasarea definită prin comanda $\tau \in \{r,u,l,d\}$). De exemplu Π_c-cuvântul $w = arbr^3 arbd^3 adbl^3 albu^3 fdrcr \in (\Pi \cup C)^*$ descrie imaginea din Figura 57 (unde C={*a,b,c,f*}, iar cu *f* am notat culoarea *fondului* care poate fi *alb* sau poate fi interpretat ca *fără culoare*).

Pentru funcţiile picturale de descriere a imaginilor color, Π_c-cuvântul echivalent se defineşte analog ca şi în cazul imaginilor discontinue, doar subcuvintele α_{ij} (respectiv β_{ij}) se definesc astfel :

$$\alpha_{ij} = \begin{cases} cr & \text{pentru } j<u_2 \text{ şi } f(S_{ij}) = c, \quad (c = \text{culoarea lui } S_{ij}) \\ \uparrow r & \text{pentru } j<u_2 \text{ şi } f(S_{ij}) = 0 \quad (\uparrow = \textit{fără culoare}) \\ cd & \text{pentru } j=u_2 \text{ şi } f(S_{ij}) = c, \\ \uparrow d & \text{pentru } j=u_2 \text{ şi } f(S_{ij}) = 0. \end{cases}$$

Subcuvintele β_{ij} se definesc analog (valorile fiind *cl,* ↑*l, cd* şi↑*d*).

În concluzie putem spune că mulţimea imaginilor descrise prin funcţii picturale este identică cu mulţimea imaginilor descrise prin Π_c-cuvinte, adică orice imagine descrisă printr-o funcţie picturală poate fi descrisă printr-un Π_c-cuvânt, şi invers.

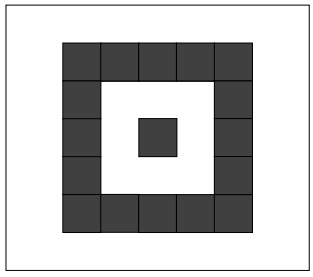

Figura 56 - $\Pi_{\prec}$-cuvânt,

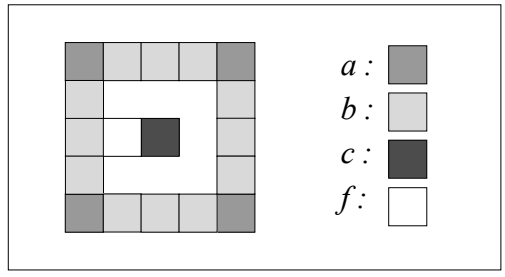

Figura 57 - Π_c-cuvânt.

A doua metodă de construcţie a $\Pi_{\prec}$-cuvintelor pornind de la o funcţie picturală este mai eficientă (în sensul complexităţii descrierii) pentru imaginile *rare* (care au puţine pătrate colorate în comparaţie cu fereastra ecran), adică raportul

$r = |\{f(S_{ij})>0 \ / \ v_1 \le i \le v_2 \ , \ u_1 \le j \le u_2\}| \ / \ (v_2-v_1)\cdot(u_2-u_1)$ este relativ mic.

Spre deosebire de prima metodă care traversează toată fereastra (Figura 55), metoda care urmează să fie prezentată va *căuta* doar acele pătrate care sunt colorate ($f(S_{ij})>0$).

Pentru imagini alb-negru vom construi $\Pi_{\prec}$-cuvântul astfel :

$w = w_{v1,u1}w_{v1,u1+1}...w_{v1,j}...w_{v1,u2}...w_{i,u1}w_{i,u1+1}...w_{i,j}...w_{i,u2}...w_{v2,u1}w_{v2,u1+1}...w_{v2,j}...w_{v2,u2}$,

unde:

$$w_{i,j} = \begin{cases} \lambda & \text{dacă } f(S_{ij}) = 0, \\ r^j d^i \downarrow u \uparrow u^i l^j & \text{dacă } f(S_{ij}) > 0 . \end{cases} \qquad (v_1 \le i \le v_2 \ , \ u_1 \le j \le u_2)$$

Pentru că acest cuvânt conţine multe deplasări inutile cu *creionul* ridicat, este necesară o reducere a acestui cuvânt. $\Pi_{\prec}$-cuvântul redus este $w_r = \text{ref}_{\prec}(w)$ care reprezintă $\Pi_{\prec}$-cuvântul independent de retrageri relativ la mulţimea retragerilor $R_{\prec}=\{ud, du, lr, rl, \uparrow\downarrow, \downarrow\uparrow\}$.

Pentru imagini color definirea Π_c-cuvântului este asemănătoare. Subcuvântul $w_{i,j}$ pentru cazul $f(S_{ij}) = c > 0$ este $w_{i,j} = r^j d^i c u \uparrow u^i l^j$, unde prin $\uparrow$ am notat culoarea *invizibilă* (prin care se permite deplasarea fără colorare).

Cuvintele picturale definite anterior sunt echivalente cu funcţiile picturale, dar nu sunt *minimale*. Desigur, ne-ar interesa un algoritm care să construiască un cuvânt de descriere cât mai scurt posibil, plecând de la o funcţie picturală *f*:

- Construieşte graful G=(V,A) astfel :
  $V=\{ (i,j) / f(S_{ij}) > 0 \}$, $A=V\times V$;
- Calculează distanţele directe ($\delta : A \to \mathbf{N}$) dintre vârfuri astfel :
  $\delta((i_1,j_1), (i_2,j_2))= |i_1 - i_2| + |j_1 - j_2|, \ \forall((i_1,j_1), (i_2,j_2)) \in A,$

- Determină cel mai scurt lanţ *hamiltonian*, $H = (v_{i1}, v_{i2}, ... , v_{in})$ (n=|V|);
- Construieşte cuvântul de descriere $w = Reduc(c_{i1} r\uparrow w_{i2}\, w_{i3} \ldots\ w_{in})$, unde

$$2 \le k \le n \quad w_{ik} = \begin{cases} \tau^{|j1-j2|-1}\, c\, \tau\uparrow & \text{dacă } i_1=i_2\ , \\ \tau^{|j1-j2|-1}\, d^{|i2-i1|-1}\, c\, d\uparrow & \text{dacă } i_1<i_2\ , \\ \tau^{|j1-j2|-1}\, u^{|i2-i1|-1}\, c\, u\uparrow & \text{dacă } i_1>i_2\ ; \quad \text{iar} \end{cases}$$

- c_{i1} este culoarea pătratului v_{i1},
- (i_1,j_1) şi (i_2,j_2) sunt coordonatele vârfurilor v_{ik-1} respectiv v_{ik-1} ,
- $\tau=r$ pentru $j_1<j_2$ sau $\tau=l$ pentru $j_1>j_2$,
- c reprezintă culoarea pătratului $S_{i2,j2}$ ($c=f(S_{i2,j2})>0$),
- $\uparrow$ *reprezintă* culoarea *invizibilă* , dacă $f(S_{i2,j2})=0$,
- $Reduc(\alpha\uparrow c\beta)=\alpha c\beta$ (nu se ridică creionul dacă urmează imediat o culoare).

Algoritmul prezentat determină un cuvânt *minimal* de descriere a unei imagini definită prin funcţia picturală *f*.

Demonstraţie. Se observă că se urmăreşte deplasarea cursorului începând din vârful v_{i1} prin pătratele $v_{i2}, \ldots , v_{ik} , \ldots , v_{in}$, pe care le colorează. În funcţie de poziţia relativă a două pătrate consecutive (v_{ik-1}, v_{ik}) din lanţul hamiltonian, cursorul se va deplasa pe unul din cele patru trasee din Figura 59.

Din cele $C^{\Delta u}_{\Delta u+\Delta v}$ trasee ($\Delta u = | j_1 - j_2|$, iar $\Delta v = |i_1 - i_2|$, vezi Figura 58), nu contează pe care îl vom parcurge pentru că în dreptungiul definit de cele două vârfuri diagonal opuse nu există nici un vârf pe care trebuie să-l colorăm (dacă ar exista, atunci lanţul determinat nu mai este minim) .

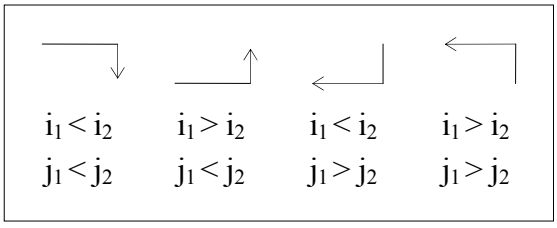

Figura 59 – Trasee posibile

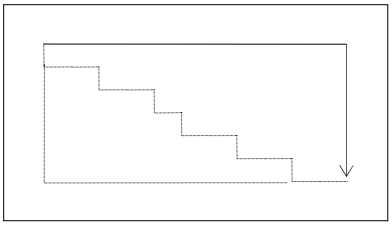

Figura 58– Traseul ales

În cele ce urmează vom studia diverse moduri de descriere a imaginilor de tip 3 (formate din linii şi curbe). Pentru că descrierea imaginilor formate din linii a fost prezentată în primul capitol (utilizând Π-limbaje), în continuarea acestui paragraf vor fi prezentate câteva tehnici de descriere a curbelor.

O imagine este de tip 3 dacă poate fi complet descrisă printr-un număr finit de linii şi curbe [[27]. O curbă poate fi aproximată utilizând o mulţime de puncte alese dintr-o mulţime de puncte din plan (de exemplu dintr-un sistem cartezian rectangular, ca în Figura 60 şi Figura 61).

Prin unirea punctelor apropiate de curbă se va obţine o aproximare poligonală formată din segmente de lungime unitate sau $\sqrt{2}$. Această aproximare se poate reprezenta prin şiruri de comenzi (pe cele 4 sau 8 direcţii).

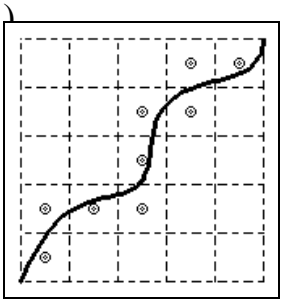

Figura 60 – Curba *4d*

De exemplu curba din Figura 60 se poate aproxima prin desenul descris de Π-cuvântul *urruurur* iar curba din şi Figura 61 prin *aaura* (am notat cu *a* deplasarea pe direcţia NE). Acest procedeu de trecere de la o imagine reprezentată prin curbe la o imagine reprezentată prin puncte se numeşte *determinarea punctelor critice*, iar procedeul invers se numeşte *interpolare*.

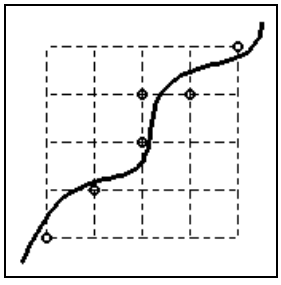

Figura 61 – Curba *8d*

Mulţimea punctelor critice determinate de curba $c \subset R^2$ este $M_{Pc}(c)=\{Apr(P)/P \in c\}$ ($Apr : R^2 \rightarrow Z^2$, returnează cel mai apropiat punct de coordonate întregi).

O imagine reprezentată prin puncte (de tip 4) poate fi îmbunătăţită prin interpolare, obţinând o imagine reprezentată prin curbe. Pentru a ilustra aceasta, am ales interpolarea *Bezier* (descrisă în [[27]). În figurile următoare (Figura 63 - şi Figura 62) se poate vedea o curbă şi punctele critice determinate, apoi descrierea corespunzătoare Π-cuvântului rezultat

(pentru patru respectiv opt direcţii), iar în final curba Bezier corespunzătoare.

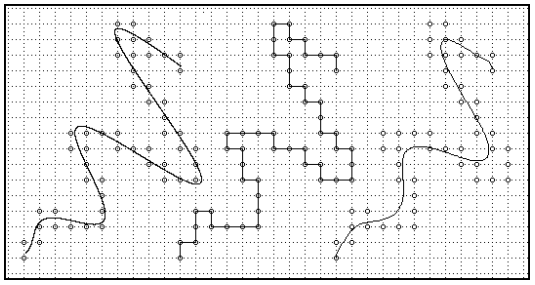

Figura 63 - Aproximare *4d*

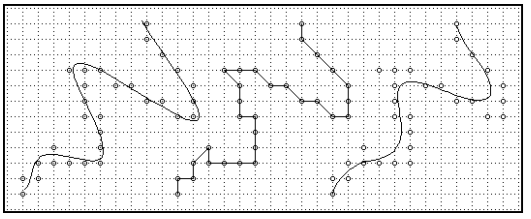

Figura 62- Aproximare *8d*

Modul de determinare al curbei Bezier (din Figura 64), plecând de la şirul de puncte $P_1,P_2,...,P_n$ este descris în cele ce urmează şi ilustrat în Figura 65, pentru n=4.

Din şirul punctelor date P_1, P_2, ... ,P_n se calculează, utilizând o relaţie de recurenţă, un alt şir de puncte $P_{1,2}$, $P_{2,2}$, ... ,$P_{n-1,2}$, apoi din punctele calculate vom determina punctele $P_{1,3}$, $P_{2,3}$, ... ,$P_{n-2,3}$ (utilizând aceeaşi formulă) şi aşa mai departe până când vom ajunge la un singur punct $P_{1,n}$. Acest punct obţinut pentru o valoare $\alpha \in [0,1]$ aparţine curbei Bezier.

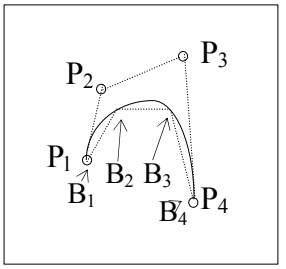

Figura 64 – Curba Bezier

Dând valori lui α din acest interval vom obţine aproximarea dorită (în Figura 65, $\alpha=1/4$).

Se observă că pentru $\alpha=0$ se obţine punctul P_1 iar pentru $\alpha=1$ se obţine punctul P_n.

Formula de calcul a şirului de puncte de pe nivelul *m*+1 în funcţie de punctele de pe nivelul *m* este următoarea:

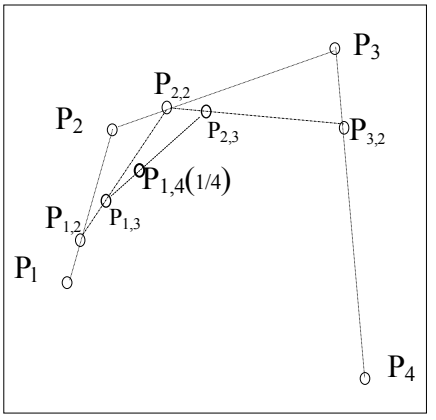

Figura 65 – Interpolarea Bezier

$$P_{i,m+1}(\alpha)=(1-\alpha)\cdot P_{i,m}(\alpha)+\alpha\cdot P_{i+1,m}(\alpha);\quad i=\overline{1,n-m};\ m=\overline{1,n-1};\ \alpha=\overline{0,1},\ 1/(s-1)$$

Se observă că formula determină punctele începând de pe nivelul doi, pentru că primul nivel conţine punctele date, adică $P_{i,1}(\alpha)=P_i$, $i=\overline{1,n}$.

Dacă dorim să determinăm *s* puncte

$$B_1=P_{1,n}(\alpha_1),\ B_2=P_{1,n}(\alpha_2),\ \ldots\ ,\ B_j=P_{1,n}(\alpha_j),\ \ldots\ ,\ B_s=P_{1,n}(\alpha_s)$$

corespunzătoare valorilor $\alpha_1=0$, $\alpha_2=1/(s-1)$, ... ,$\alpha_j=(j-1)\ /\ (s-1)$, ... ,$\alpha_s=1$, vom calcula coordonatele punctelor $P_{1,n}(\alpha_j)$ pentru fiecare valoare α_j. Pentru a obţine şi alte puncte în afara celor două capete (pentru că B_1 şi B_s coincid cu P_1 respectiv P_n , ca în Figura 64 pentru s=4), trebuie ca *s* să fie mai mare decât 2.

Un algoritm *Bezier* de determinare a celor *s* puncte (B_j, $j=\overline{1,s}$) de pe această curbă (fără a utiliza polinoame) poate fi următorul:

Pentru i:=1,n *execută* $P_{i,1}:=P_i$ *sf_pentru*;
Pentru j:=1,s *execută*
    $\alpha:=(j-1)/(s-1)$;
    Pentru m:=1,n–1 *execută*
        Pentru i:=1,n–m *execută*
            $P_{i,m+1} := (1-\alpha) \cdot P_{i,m} + \alpha \cdot P_{i+1,m}$
        Sfpentru
    Sfpentru;
    $B_j:= P_{1,n}$
Sfpentru.

Algoritmul prezentat ne dă ca rezultat un şir de puncte $B_1,B_2,...,B_s$. Teoretic, curba de interpolare Bezier (a punctelor $P_1,P_2,...,P_n$) este:

$$Bezier(P_1,P_2,...,P_n) = \{\ P_{1,n}(\alpha)\ /\ \alpha\in[0,1]\ \}.$$

Practic, reprezentatarea grafică a curbei este realizată prin desenarea celor *s*-1 segmente determinate de şirul punctelor $B_1, B_2, \ldots , B_s$, adică

$$Bezier_{Seg}(P_1,P_2,...,P_n) = \{\ \overline{B_jB_{j+1}}\ /\ j=\overline{1,s-1}\}$$ (vezi Figura 64).

Se observă în Figura 63 - şi în **Error! Reference source not found.** că dacă plecăm de la o curbă c şi determinăm mulţimea punctelor critice $M_{Pc}(c)$, apoi prin algoritmul *Bezier* calculăm punctele B_j, j=1,s , acestea se depărtează de punctele critice iniţiale, adică $M_{Pc}(c) \neq \{Apr(B_j) / j = 1,s\}$, iar curba $c' = Bezier_{Seg}(M_{Pc}(c)) \neq c$. Desigur că ne-ar interesa ca această curbă determinată prin interpolarea punctelor critice să fie cât mai apropiată de curba iniţială, adică :

$$\{Apr(P_{1,n}(\alpha)) / \alpha \in [0,1]\} = \{Apr(P) / P \in c\}.$$

Presupunem că avem o curbă c_0 , pentru care determinăm şirul punctelor s_0 de coordonate întregi cele mai apropiate. Acest şir poate fi descris printr-un Π-cuvânt w astfel încât mulţimea punctelor desenului descris de w să fie mulţimea punctelor critice corespunzătoare curbei c_0, adică:

$$V(dpic(w)) = \{Apr(P) / P \in c_0\}.$$

În continuare, dacă pentru acest şir de puncte $s_0=(Sh(a_1...a_i), i=\overline{1,n})$ determinat de Π-cuvântul $w=a_1...a_n$ (şir definit anterior prin funcţia *Sh*), determinăm curba c_1 prin interpolare Bezier, este puţin probabil (deşi ar fi de dorit acest lucru) să obţinem aceeaşi curbă iniţială c_0.

Aplicând iterativ acelaşi procedeu vom obţine un şir de curbe c_0, c_1, c_2, ... , c_p , ... Problema care în mod natural se va pune este următoarea : " Cum pot fi obţinute aceste transformări, astfel încât acest şir să fie finit? ".

Dacă reuşim să obţinem o curbă (prin interpolare) care să menţină punctele critice, atunci $s_1=s_0$, deci curba de aproximare nu se mai modifică ori de câte ori se aplică aceste transformări :

$$c_0 \longrightarrow s_0 \dashrightarrow c_1 \longrightarrow s_0 \dashrightarrow c_1 \ ...$$

Observaţie. Această proprietate a celor două transformări (de interpolare şi de determinare a punctelor critice) este importantă, pentru că fiecare transformare este, în acest caz, inversa celeilalte. De exemplu, după codificarea unei imagini (în scopul comprimării), decodificarea (adică transformarea inversă) se va efectua printr-o interpolare (care să conserve punctele critice, deci descrierea prin Π-cuvinte).

În acest scop, în continuare se va prezenta o transformare (prin interpolare Bezier) care conservă Π-cuvântul de descriere.

Pentru a obţine un şir de puncte cât mai apropiat de punctele critice, vom grupa punctele $P_1,P_2,...,P_n$ în subşiruri de o anumită lungime *q* (de exemplu în Figura 66 am grupat câte 5 respectiv câte 7), apoi vom aplica algoritmul *Bezier* pentru fiecare subşir.

Figura 66 - Grupuri de curbe

Curba rezultat fiind obţinută prin reuniunea curbelor descrise:

$$c = Bezier\ (P_1,...,P_q) \cup Bezier\ (P_q,...,P_{2q-1}) \cup ... \cup Bezier\ (P_{n-(n\ Mod\ q)+1},...,P_n)$$

O transformare de tipul 3.1.6, utilizând grupări de lungime q<7, conservă punctele critice ($Apr(c)=\{P_1,P_2,...,P_n\}$), deci şi Π-cuvântul de descriere (*w*), dacă acesta nu conţine *retrageri* ($w \in ref(\Pi^*)$).

Demonstraţia se poate face pentru fiecare *q* începând de la 2 până la 6.

Este mult mai eficientă gruparea punctelor iniţiale, $P_1,P_2,...,P_n$ în cât mai puţine subşiruri (deci cât mai puţine curbe Bezier) care să conserve punctele critice (fără a stabili un număr *q* de puncte din subşiruri). Aceasta înseamnă că se urmăreşte formarea de subşiruri de dimensiuni cât mai mari. La un subşir $P_i,P_{i+1},...,P_j$ care conservă punctele critice se va adăuga punctul P_{j+1} doar dacă şirul $P_i,P_{i+1},...,P_jP_{j+1}$ conservă punctele critice (situaţie în care se continuă cu punctul P_{j+2} şi aşa mai departe) altfel se începe o nouă grupare $P_j,P_{j+1},...$. În felul acesta se vor forma subşiruri de puncte de lungime cel puţin şase, aceasta depinzând de poziţia relativă a punctelor.

Algoritmul următor determină numărul minim (notat cu *k*) de *grupări* care conservă punctele critice:

Date $P_1, P_2, ..., P_n$; k:=0; i:=1;

Repetă j:=i+5; *Cât_timp* j>n *execută* j:=j−1;

Cât_timp (j<n) *şi* $Apr(Bezier(P_i, ..., P_j, P_{j+1})) = \{P_i, ..., P_j, P_{j+1}\}$ *execută* j:=j+1;

k:=k+1; $Gruparea_k := (P_i, ..., P_j)$; i:=j

Pânăcând i=n;

Rezultate $Gruparea_1, Gruparea_2, \ldots, Gruparea_k$.

Observaţie. Numărul minim de *grupări* care conservă punctele P_1, P_2, ... ,P_n este $NrGr(1,n)$, unde:

$$NrGr(i,j) = \begin{cases} 1 & \text{dacă } Apr(Bezier(P_i,...,P_j)) = \{P_i, ...,P_j\}, \text{ (a)} \\ \underset{i<l<j}{Min}\{NrGr(i,l)+NrGr(l,j)\} & \text{dacă } Apr(Bezier(P_i,...,P_j)) \neq \{P_i, ...,P_j\}. \text{ (b)} \end{cases}$$

Dacă *Lmin(i,j)* este indicele *l* pentru care se realizează minimul din formula precedentă cazul b) sau este egal cu 0 pentru cazul a), atunci şirul de puncte P_1, P_2, ... ,P_n este despărţit optim în două subşiruri, şi anume (P_1, P_2, ... ,$P_{Lmin(1,\ n)}$) şi ($P_{Lmin(1,\ n)}$, ... , P_n). În continuare, fiecare subşir P_i, ...,P_j se desparte în două subşiruri în punctul $P_{Lmin(i,\ j)}$ (dacă $Lmin(i,j)>0$), şi aşa mai departe până când toate subşirurile au indicele de separare *Lmin(i,j)* nul. Observăm că numărul subşirurilor astfel obţinute este minim.

Pentru ca acest algoritm să fie mai rapid (pentru a nu repeta anumite calcule) se recomandă depunerea valorilor deja calculate $NrGr(i, j)$ într-o matrice pătratică M de ordin *n*. Valoarea $NrGr(i, j)$ o vom depune în M_{ij}. Valorile acestei matrice le vom determmina succesiv pentru prima diagonală paralelă cu diagonala principală, apoi pentru a doua, şi aşa mai departe până se ajunge la M_{1n} care reprezintă valoarea căutată. Pe baza precizărilor anterioare, putem să evităm calculele pentru primele şase diagonale, acestea având toate valorile egale cu 1. Practic, calculul valorilor va începe cu diagonala a şaptea. Pentru a putea forma subşirurile optimale (nu ne interesează doar numărul de grupări minim dat de M_{1n}) vom memora în elementul $M_{j,i}$ indicele pentru care se realizează minimul funcţiei pentru subşirul punctelor de la *i* la *j* (putem acest lucru, deoarece elementele de sub diagonala principală nu sunt utilizate).

Vom construi un arbore binar care conţine în fiecare perechi de forma (*i,j*), unde *i* şi *j* reprezintă limitele unui subşir optim determinat (P_i, ... ,P_j). Acest arborele are rădăcina (1,*n*). Dacă pentru un nod (*i,j*) al arborelui avem $Apr(Bezier(P_i, \ldots, P_j)) = \{P_i, \ldots, P_j\}$ atunci acest nod este un nod terminal

(frunză), altfel va avea doi subarbori cu rădăcinile (i,M_{ji}) respectiv (M_{ji},j). Frontul acestui arbore ne va da subşirurile optimale.

Evident că algoritmii prezentaţi anterior (apelând metoda programării dinamice) nu dau neapărat aceeaşi soluţie, dar ambele sunt minimale. Dacă avem mai multe variante de despărţire a unui subşir de puncte, se preferă, separarea în punctul P_i, dacă punctele P_{i-1}, P_i şi P_{i+1} sunt coliniare (aceasta asigură continuitatea curbei Bezier, deoarece aceasta este tangentă la capetele liniei poligonale determinate de punctele de interpolare). În acest fel curba are mai puţine puncte de întoarcere, deci mai puţine *asperităţi*.

În continuare se va prezenta un model de descriere a unei imagini de tip 3 (după clasificarea dată în [27]) utilizând $\Pi_{\prec}$-cuvinte. Aceste cuvinte vor descrie un şir de puncte care determină o curbă prin interpolare Bezier. Aceasta înseamnă că o mulţime de $\Pi_{\prec}$-cuvinte descrie o imagine formată din curbe, adică de tip 3. Plecând de la un anumit punct de start $P_1(x_1, y_1)$ (care poate fi chiar originea), printr-un $\Pi_{\prec}$-cuvânt de descriere $w=a_1a_2...a_m$, vom construi şirul punctelor de interpolare $P_2(x_2, y_2)$, $P_3(x_3, y_3)$, ... , $P_n(x_n, y_n)$. Din şirul punctelor traversate prin comnezile $\Pi_{\prec}$-cuvântului w, se vor selecta în vederea interpolării doar punctele traversate cu creionul coborât (↓).

Fiecare $\Pi_{\prec}$-cuvânt descrie un şir de puncte aşa cum s-a prezentat mai sus, prin funcţiile definite anterior *Nr* şi *Pb*, iar prin interpolare Bezier a şirurilor construite se vor obţine curbele care compun imaginea.

De exemplu $\Pi_{\prec}$-cuvintele următoare

$w_1 = dr^2u^2\uparrow u^2\downarrow u\uparrow u^2r^2\downarrow r$,
$w_2 = \uparrow ld\downarrow d\uparrow d^2\downarrow d^3r^2u\uparrow u^2l$ şi
$w_3 = \uparrow d\downarrow l\uparrow l^2u\downarrow u\uparrow l\downarrow d$

descriu următoarele trei şiruri de puncte :

1. (0,0),(0,-1),(1,-1),(2,-1),(2,0),(2,1),(2,4),(5,6);
2. (5,6),(4,4),(4,1),(4,0),(4,-1),(5,-1),(6,-1),(6,0) şi
3. (5,2),(4,1),(2,3),(1,2).

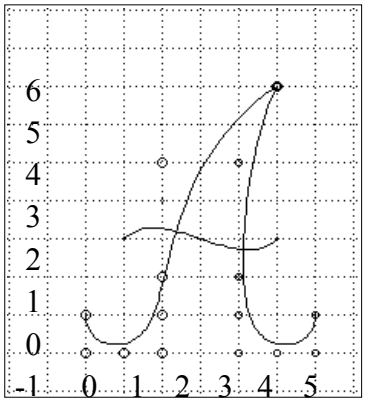

Figura 67 - Caracterul 𝒜

Prin interpolarea acestor puncte se obţin cele trei curbe reprezentând imaginea literei *A mare de mână*, din Figura 67. Asemănător se pot defini $\Pi_{<}$-cuvinte de descriere a unor simboluri matematice.

4.2 Reprezentarea imaginilor prin arbori

Arborii memorează informaţiile formelor prin noduri (stocând informaţii despre forme sau substructuri) şi prin arce (relaţii dintre *părinte* şi *succesori*). Arborii sunt eficienţi în reprezentarea structurală care implică o descompunere ierarhică pentru că descrierea devine compactă. O altă trăsătură importantă a structurilor arborescente este capacitatea de a construi şi de a descrie forme complexe, utilizând primitive cu mai multe puncte de concatenare (la structuri de tip şir am văzut că sunt doar două puncte *cap* şi *coadă*).

Dacă o formă complexă poate fi reprezentată natural prin arbori atunci este de asemenea natural să generarăm descrierea sa utilizând gramatici arborescente. Figura 68 arată cum poate fi utilizat un arbore pentru a descrie o formă grafică (o imagine) descompusă în *arii elementare disjuncte*. Imaginea este descompusă în patru *arii* (pătrate, considerate în ordinea a_{11} , a_{12} , a_{21} , a_{21}) iar dacă o *arie* conţine puncte de aceeaşi culoare atunci ea va fi reprezentată printr-un nod terminal (frunză), informaţia fiind culoarea respectivă, altfel (dacă *aria* conţine puncte de culori diferite) atunci ea va fi reprezentată printr-un nod intern şi se va descompune din nou în patru *arii elementare* (care vor fi subarbori) şi aşa mai departe până se ajunge la nivel de punct (pixel).

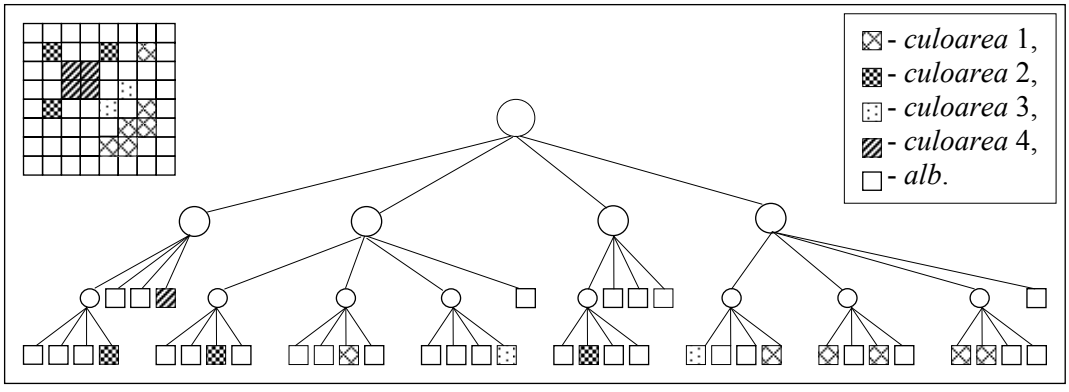

Figura 68 - Descompunerea în arii elementare

O modalitate de codificare a ierarhiei descrise de un arbore este prezentată în continuare şi ilustrată în Figura 69. Vârfurile sunt etichetate cu numere naturale şi se utilizează caracterul "**.**" (punct) pentru a descrie nodurile descendente. Arborele este descris prin simpla enumerare a mulţimii etichetelor nodurilor astfel :

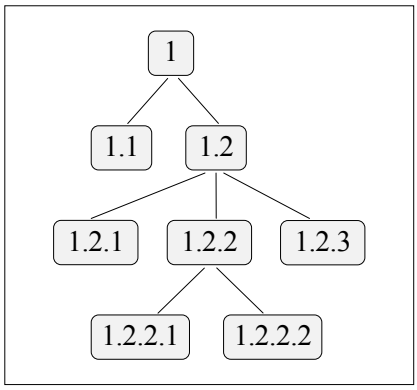

Figura 69 - Ierarhie

T = { 1, 1.1, 1.2, 1.2.1, 1.2.2, 1.2.3, 1.2.2.1, 1.2.2.2 }

Astfel de descrieri sunt adesea utilizate în compararea arborilor (calculul măsurii de similaritate, prin distanţe $d(T_1,T_2)$ T_1 şi T_2 fiind arborii care se compară).

Altă metodă de codificare a unei imagini reprezentată prin *Quad-Tree* (arbri care se ramifică în patru, prezentaţi mai sus) utilizează şiruri de caractere care conţin codul culorilor şi un alt caracter *de ramificare* (de exemplu '*'). De exemplu imaginea din Figura 70 se codifică prin şirul '***annanna*'.

a	*n*	*n*	*n*
n	*a*	*n*	*n*
n	*n*	*a*	*a*
n	*n*	*a*	*a*

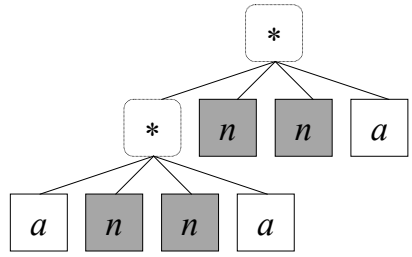

Figura 70 – Codificarea prin şiruri

Şirul *S* codificat poate fi returnat de funcţia *Qt* definită astfel:

Funcţia *Qt* (*l,t,n*) **Este**: { (*l,t*) = colţul *stânga-sus* al ferestrei }
Dacă *AccCul*(*l,t,n*) **Atunci** *Qt*:=*Culoare*(*l,t*) { *n* = lăţimea pătratului }

Altfel $n:=n/2$;
Qt:='*' + $Qt(l,t,n)$ + $Qt(l+n,t,n)$
+ $Qt(l,t+n,n)$ + $Qt(l+n,t+n,n)$

Sf_*Qt*.

Decodificarea şirului *S* poate fi realizată utilizând procedura *Pt* definită astfel:

Procedura *Pt* (*l,t,n*) **Este**:{ Reconstituie imaginea în (*l,t*) }

Cul:=Primul_caracter(*S*); *Şterge_Primul_Caracter*(*S*);
Dacă *Cul*∈*Mulţimii_Culorilor* **Atunci** *Desenează_Pătratul*(*l,t,n,Cul*)
Altfel $n:=n/2$; *Pt*(*l,t,n*); *Pt*(*l+n,t,n*);
Pt(*l,t+n,n*); *Pt*(*l+n,t+n,n*)

Sf_*Pt*.

Codificare a unei imagini prin arbri binari utilizează două caractere *de ramificare* '-' şi '|'. De exemplu imaginea din Figura 71 se codifică prin şirul: '-|-|*an*|*nan*|*na*' .

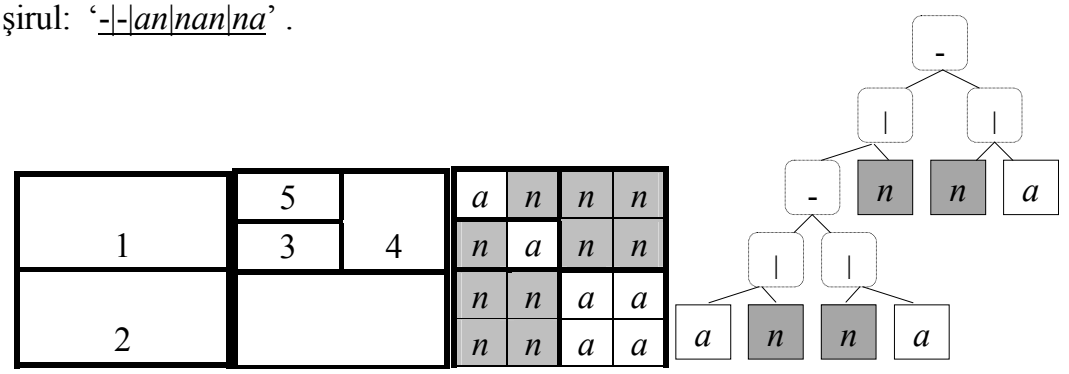

Figura 71 – Codificarea prin arbori binari

Şirul codificat poate fi returnat de funcţia *Bt* definită astfel:

Funcţia *Bt* (*l,t,m,n*) **Este**: { (*l,t*) = colţul *stânga-sus* al ferestrei }
Dacă *AccCul*(*l,t,m,n*) **Atunci** *Bt:=Culoare*(*l,t*) { *m,n* = dim. dreptunghiului}
Altfel Dacă *m=n* **Atunci** *m*:=*m*/2;
Bt:='-'+*Bt*(*l,t,m,n*)+*Bt*(*l,t+m,m,n*)
Altfel *n*:=*n*/2;

Bt:='|'+*Bt*(*l,t,m,n*)+*Bt*(*l*+*n,t,m,n*)

Sf_*Bt*.

Decodificarea poate fi realizată utilizând procedura *Pb* definită astfel:

Procedura *Pb* (*l,t,m,n*) **Este**: { Reconstituie imaginea în (*l,t*) }
Cul:=Primul_caracter(*S*); *Şterge_Primul_Caracter*(*S*);
Dacă *Cul*∈*Mulţimii_Culorilor* **Atunci**
Desenează_Dreptunghiul(*l,t,m,n,Cul*)
Altfel
Dacă *Cul*='-' **Atunci** *m*:=*m*/2; *Pb*(*l,t,m,n*); *Pb*(*l,t+m,m,n*);
{*Cul*='|'} **Altfel** *n*:=*n*/2; *Pb*(*l,t,m,n*); *Pb*(*l+n,t,m,n*);
Sf_*Pb*.

4.3 Îmbunătăţirea imaginilor.

Scopul acestor prelucrări îl constituie accentuarea sau punerea în evidenţă a unor caracteristici conţinute în imagine pentru a putea fi observate mai uşor (mai bine) la studiul acestora. Metodele utilizate în algoritmii de îmbunătăţire a imaginilor amplifică anumite caracteristici fără a mări cantitatea de informaţii conţinută în acestea. În cele ce urmează vor fi prezentate câteva tehnici grupate după algoritmii utilizaţi în următoarele două clase:

- *Operaţiuni punctuale* prin care se poate realiza creşterea contrastului, reducerea zgomotului, etc. şi
- *Operaţiuni spaţiale* care permit eliminarea zgomotului, filtrări, etc.

Pentru simplificarea prezentării, vom presupune că imaginile prelucrate prin aceste metode sunt de tip 2 (după clasificarea dată în [[27]), deci deci conţin diverse nuanţe de gri.