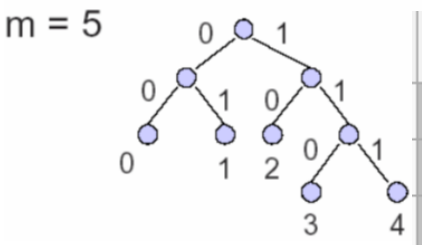
Golomb - Rice kód

- GR kódy sú optimálne (resp. blízko optimality) pre obojstranné geometrické rozdelenia
- $N = qm + r$, pričom $0 \leq r < m$
  - q je kvocient a r je zvyšok
- kód predstavuje q zakódované unárne (počtom jednotkových bitov) + „0“ + r zakódované ako skrátený binárny kód, nazývaný aj Rice kód.
  - Skrátený binárny kód sa používa pre rovnomerné rozdelenia pravdepodobnosti pre veľkosť abecedy m, pričom m nemusí byť mocninou o základe 2
  - ak $m = 2^r$, potom je výsledok identický s binárnym kódom
  - inak $m = 2^r + b$
    - prvým $2^r - b$ symbolom priradí kódové slová o dĺžke r
    - ostatným $2b$ symbolom priradí posledných $2b$ kódových slov o dĺžke $r + 1$
  - na obrázku sú príklady pre m=5
    - ako vyzerá strom – ukazuje, že výsledný kód je **prefixový**
    - ako vyzerá binárna reprezentácia - šedé hodnoty sú vynechané bity a kombinácie), na reprezentáciu 5 možných zvyškov sa použijú biele bity



Kódové slovo		
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Príklad: Golombov kód pre m=5 a náhodne zvolené čísla N

N	q	r	Rice(r)	Golombov kód
2	0	2	10	010
6	1	1	01	1001
9	1	4	111	10111
10	2	0	00	11000
27	5	2	10	11111010

Príklad:
Pre GR kód s m=5 dekódujte z bitov
10000011011011001
=5+0,0+1,5+3,10+1
=5, 1, 8, 11

Run-Length kódovanie (Run Length Encoding – RLE)

- Existuje veľa verzií
- Použitie: kódovanie bitových rovín, čiernobiele obrázky, ...

Základná verzia:

- Na začiatku postupnosti sa predpokladajú nuly a kóduje sa ich počet, následne sa kóduje počet jednotiek, následne núl, atď.
- Počty sa kódujú fixným počtom bitov

Príklad:

Vstupná postupnosť 30 bitov: 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1

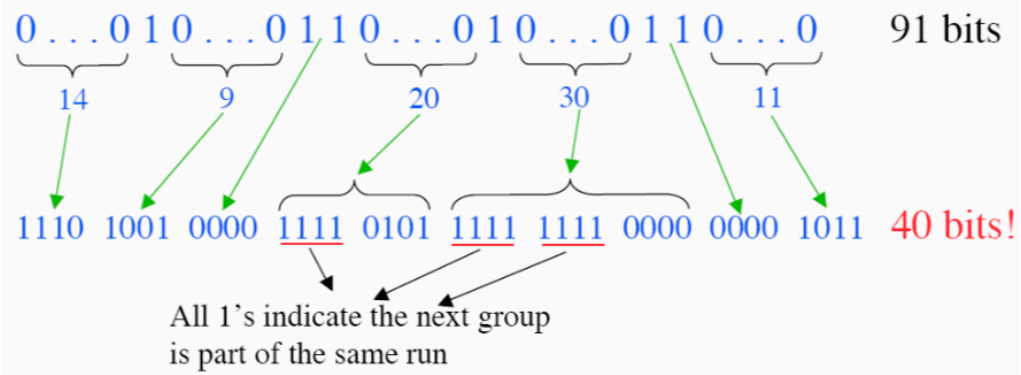
- Použijeme 3 bity na kódovanie počtov
  - Počty: 3, 5, 1, 2, 6, 5, 1, 7
  - bity: 011 101 001 010 110 101 001 111
  - Výsledný kód má 24 bitov.
- Použijeme 4 bity na kódovanie počtov
  - Počty sa nezmenia
  - bity: 0011 0101 0001 0010 0110 0101 0001 0111
  - Výsledný kód má 32 bitov
- Použijeme 2 bity na kódovanie počtov
  - Počty: 3, 3, 0, 2, 1, 2, 3, 0, 3, 3, 0, 2, 1, 3, 0, 3, 0, 1
  - bity: 11 11 00 10 01 10 11 00 11 11 00 10 01 11 00 11 00 01
  - Výsledný kód má 36 bitov

Run-Length kódovanie (Run Length Encoding – RLE) 2

- Ako zabezpečíme optimálnu kódovú dĺžku?
- Prečo nepoužiť variabilnú dĺžku?
  - Huffmanov kód
  - Exp. Golombov / Golombov kód
    - zvážime použitie najmä vtedy, ak je potencionálne nekonečná množina symbolov
    - Golombov → vysvetlený na ďalšej strane

RLE Verzia 2

- predpokladáme, že jednotky sa vyskytujú zriedkavo a osamotene
- kóduje iba behy núl, behy jednotiek nekóduje
- ak sa náhodou vyskytuje viac jednotiek za sebou, oddeľujú sa oznámením, že medzi nimi je 0 núl
- Príklad (fixná dĺžka behov, kóduje sa pomocou r=4 bitov) – z 91 bitov sa dostávame na 40:



Poznámka ku kódovaniu behov núl pri fixnej dĺžke r=4bity:

- 0-14: 0000 - 1110
- 15-29: 1111 0000 – 1111 1110
- ...

Poznámky:

- namiesto fixného kódovania dĺžky behov núl môžeme použiť napr. GR kód (beh núl môže byť veeeľmi dlhý ...)
- pre RLE2 sa dá jednoducho vypočítať aj optimálne m na kódovanie behov núl, ak má byť na ich kódovanie použitý GR kód (viď ďalšia strana)

Aké m v GR -kóde je optimálne pre RLE verzie 2

- Nech pravdepodobnosť symbolu 0 je p a pravdepodobnosť symbolu 1 je $1-p$
- Potom optimálne $m = ceil(-1/\log_2 p)$
- Napr. ak $p = \frac{127}{128}$, potom $m = ceil\left(-\frac{1}{\log_2\left(\frac{127}{128}\right)}\right) = ceil(88.38) = 89$

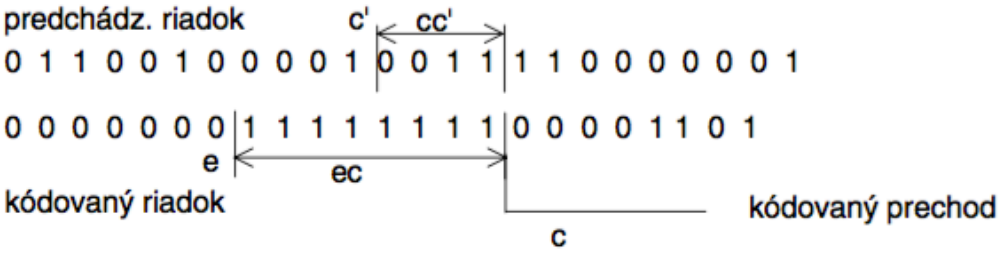
Obraz

- 2D verzie základných kódovacích postupov
- existujú aj intuitívne aj sofistikované verzie ...

2D verzia RLE – kódovanie bitových rovín

- Používa sa najmä RAC (relative address coding) kódovanie
- Kódujeme napr. po riadkoch
- Môžeme si vybrať čo kódujeme pre každý prechod (príklad: prechod c):
  - Vzdialenosť od posledného prechodu v tom istom riadku (príklad: prechod e, 0->1)
  - Vzdialenosť od rovnakého prechodu (príklad: prechod c', 1->0) v predchádzajúcom riadku smerom vľavo
  - vzdialenosť od rovnakého prechodu v predchádzajúcom riadku smerom vpravo
- Potom potrebujeme
  - Vybrať tú vzdialenosť, ktoré je najkratšia (kódovateľná najmenším počtom bitov)
  - vybrať prefix tejto vzdialenosti, aby bolo jasné o ktorú s uvedených vzdialeností kódujeme

Príklad [Gonzales, Woods, str. 453]:



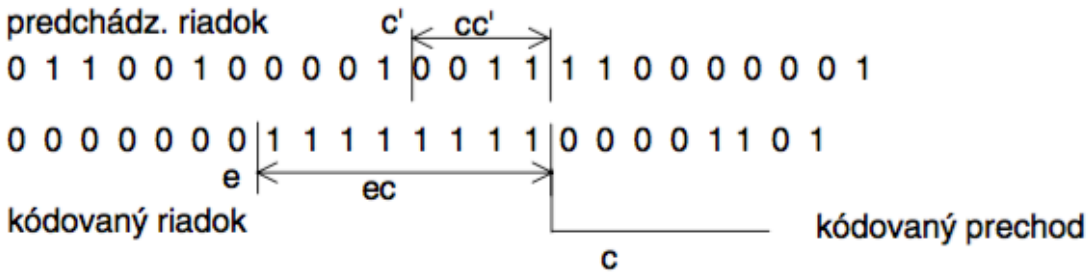
možná vzdialenosť	vzdialenosť	kód
cc'	0	0
ec alebo cc' (vľavo)	1	100
cc' (vpravo)	1	101
ec	d (d>1)	111h(d)
cc' (c' vľavo)	d (d>1)	1100h(d)
cc' (c' vpravo)	d (d>1)	1101h(d)

Výsledký kód predstavuje

- prefix možností pre rôzne kombinácie vľavo/vpravo, aktuálny/minulý riadok, d=0, d=1, d>1
- kódovanie vzdialenosti d pomocou vhodného kódu s variabilnou dĺžkou – h(d)

1 – 4	0 xx
5 – 20	10 xxxx
21 – 84	110 xxxxxx
85 – 340	1110 xxxxxxxx
341 – 364	11110 xxxxxxxxx
1365 – 5460	111110 xxxxxxxxxx

Príklad – postup:



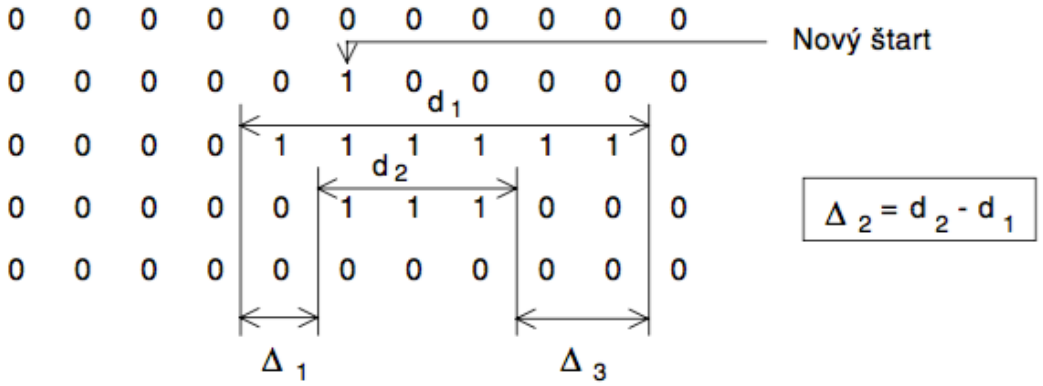
1 – 4	0 xx
5 – 20	10 xxxx
21 – 84	110 xxxxxx
85 – 340	1110 xxxxxxxx
341 – 364	11110 xxxxxxxxx
1365 – 5460	111110 xxxxxxxxxx

možná vzdialenosť	vzdialenosť	kód
cc'	0	0
ec alebo cc' (vľavo)	1	100
cc' (vpravo)	1	101
ec	d (d>1)	111h(d)
cc' (c' vľavo)	d (d>1)	1100h(d)
cc' (c' vpravo)	d (d>1)	1101h(d)

Výsledný kód (kódujeme kompletne oba riadky):

- horný riadok: ec d=1, ec d=2, ec d=2, ec d=2, ec d=4, ec d=1, ec d=2, ec d=4, ec d=6, ec d=1
  - 100, 111+001, 111+001,…, 111+10 0001, …
- dolný riadok (už môžem použiť aj predchádzajúci riadok):
  - rozhodovanie medzi [ec d=7]= 111+10 0010 (9bitov) a [cc'(c' vľavo) d=6] = 1100+01 0010 (10 bitov), vyhrava [ec d=7]
    - pri cc' hľadám zmenu 0->1, prvá takáto zmena je hneď po pdvom bite. Čo je o 6 pozícií vľavo, ako naša aktuálna pozícia
  - rozhodovanie medzi [ec d=8]=111+10 0011 (9bitov) a [cc'(c' vľavo) d=4] = 1100+0 11 (7 bitov) (rátam/scanujem od predchádzajúcej zmeny) … vyhráva [cc'(c' vľavo) d=4].
    - pri cc' hľadám zmenu 1->0, prvá takáto zmena po poslednom prechode je c'na obrázku, čo je o 4 vľavo ako aktuálna pozícia,

Kódovanie bitových rovín - kódovanie kontúr objektov: PDQ, DDC



Po štarte kódujeme každý riadok od „štartu“ objektu (jeho „najsevernejší bod“).

1) Najprv kódujeme prvý riadok: kóduje sa pozícia štartu (x,y) a dĺžka prvého riadku d1
2) Potom kódujeme riadky pod
   a. Metóda PDQ (predikčná diferenčná kvantizácia) kóduje pre každý riadok Δ_1, Δ_2, vo vzorcoch d_2 predstavuje dĺžku aktuálneho a d_1 dĺžku predošlého riadku
   b. Metóda DDQ (dvojnásobné delta kódovanie) kóduje pre každý riadok Δ_1, Δ_3
3) Na kódovanie uvedených hodnôt za použije niektorý z kódov s variabilnou dĺžkou slova kódujúci aj znamienko.

Príklad na obrázku:

PDQ	DDQ
1) Kóduj (x,y)=(6,2), d=1 2) Kóduj $\Delta_1 = -1, \Delta_2 = 5$ 3) Kóduj $\Delta_1 = 1, \Delta_2 = -3$	4) Kóduj (x,y)=(6,2), d=1 5) Kóduj $\Delta_1 = -1, \Delta_3 = 4$ 6) Kóduj $\Delta_1 = 1, \Delta_3 = -2$

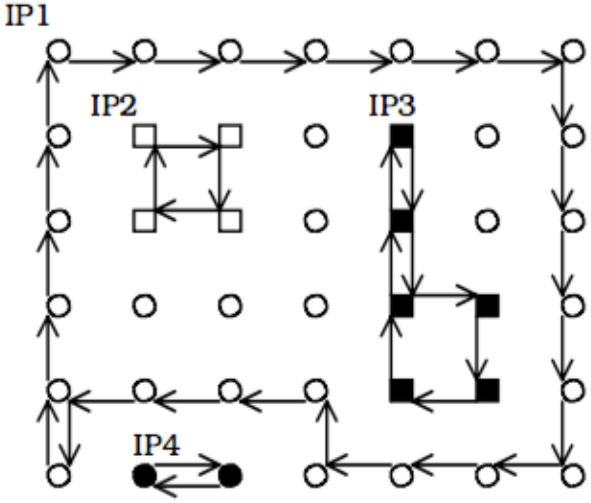
Viacúrovňové kódovanie

Podobný spôsob ako je PDQ a DDC kódovanie, je možný realizovať pre kódovanie obrazu, ktorý obsahuje viac kvantizačných úrovní [29]. Principiálne ide o opis uzavretých geometrických útvarov s rovnakou kvantizačnou úrovňou.

Pre realizáciu takéhoto kódovania je potrebné kódovať:

1. úroveň jasu (prípadne poradie farby),
2. štartovací bod príslušného objektu,
3. tvar objektu - uzavretú cestu od štartovacieho bodu opisujúcu obrys objektu a vracajúcu sa späť k štartu.

Postup je už len logickým riešením daného problému. Je potrebné obrysy označovať tak, aby v každom uzavretom reťazci existovala buď len príslušná kvantizačná úroveň alebo ďalší uzavretý objekt, s tou istou podmienkou. Ukážeme si to na príklade so štyrmi úrovňami:



Na záver potrebujeme zakódovať podľa tabuľky znaky napr. v poradí : poradie obrysu (IPx), úroveň, riadková súradnica IPx, stĺpcová súradnica IPx, smer prvého posunu, smer druhého posunu, ...

Kód je samozrejme najúčinnejší pri malom počte kvantizačných úrovní a veľkých jednoúrovňových objektoch.

Viacúrovňové kódovanie – tabuľky

poradie obrysu	kódové slovo
1	00
2	01
3	10
4	11

riadky a stĺpce	kódové slovo
1	000
2	001
:	
8	111

úroveň	kódové slovo
○	00
□	01
■	10
●	11

smer popisu	kódové slovo
↑	00
→	01
↓	10
←	11

Príklad kódovania:

00 –poradie obrysu IP1 (na spodku), 00-úroveň jasu „○“, 000-x, 000-y, 01-posun1, 01-posun2, 01, 01, ...,00, 00 (dorazili sme do počiatočných súradníc, tu je koniec) , 01 – poradie obrysu IP2 (nad IP1), ...

Kódovanie tvarov (videoobjektov) v MPEG4 – reťazový kód

Pri tejto metóde sú hranice objektu reprezentované uzavretou linkou. Kódovanie spočíva v zaznamenávaní smeru pohybu tejto linky. Kódovanie je ukončené, keď je dosiahnutý východzí bod.

Zakódované dáta určujú štartovací bod s prvým smerovým kódom a nasledujúce diferenčne zakódované smery. Ak VOP obsahuje viacero uzavretých liniek, tak reťazové kódy nasledujú za informáciou o počte oblastí.

Keďže reťazový kód je cyklický, smer môžeme diferenčne kódovať do hodnôt od –3 do 4:

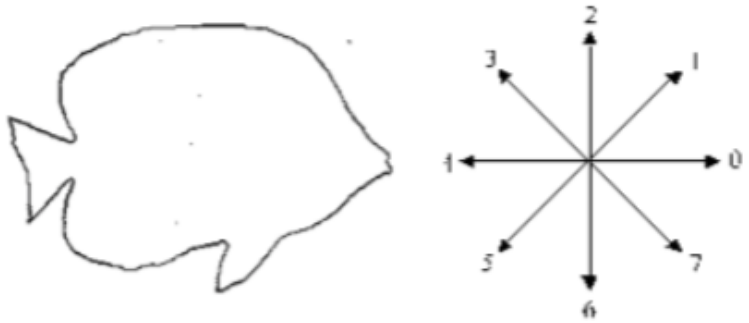
$$d = \begin{cases} cn - cn\text{-}1 + 8, & \text{ak } cn - cn\text{-}1 > -3 \\ cn - cn\text{-}1 - 8, & \text{ak } cn - cn\text{-}1 < 4 \\ cn - cn\text{-}1, & \text{inak} \end{cases} \quad (4.1)$$

kde d je diferenčný reťazový kód, cn súčasný a cn-1 predchádzajúci smer. Na kódovanie d sa používa Huffmanov kód.

Poznámka: Pozor, v texte vyššie sú operátory < > naopak!

Huffmanov kód pre diferenčný reťazový kód

d	kód
0	1
1	00
-1	011
2	0100
-2	01011
3	010100
-3	0101011
4	0101010



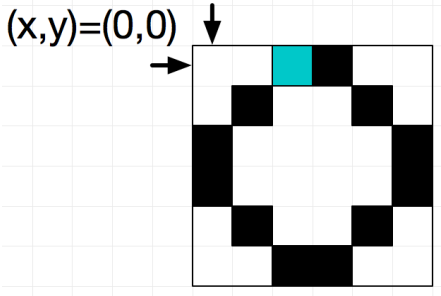
Obr. 6.7 Kódovaný objekt a znázornenie ôsmych kódovacích smerov [64]

Na strane prijímača je cn dekódovaná podľa vzorca

$$cn = (cn\text{-}1 + d + 8) \bmod 8$$

Príklad:

Objekt	Kódovanie hranice , začíname v bode (x, y) = (2, 0)



Začíname doprava (výsledok má 26 bitov)

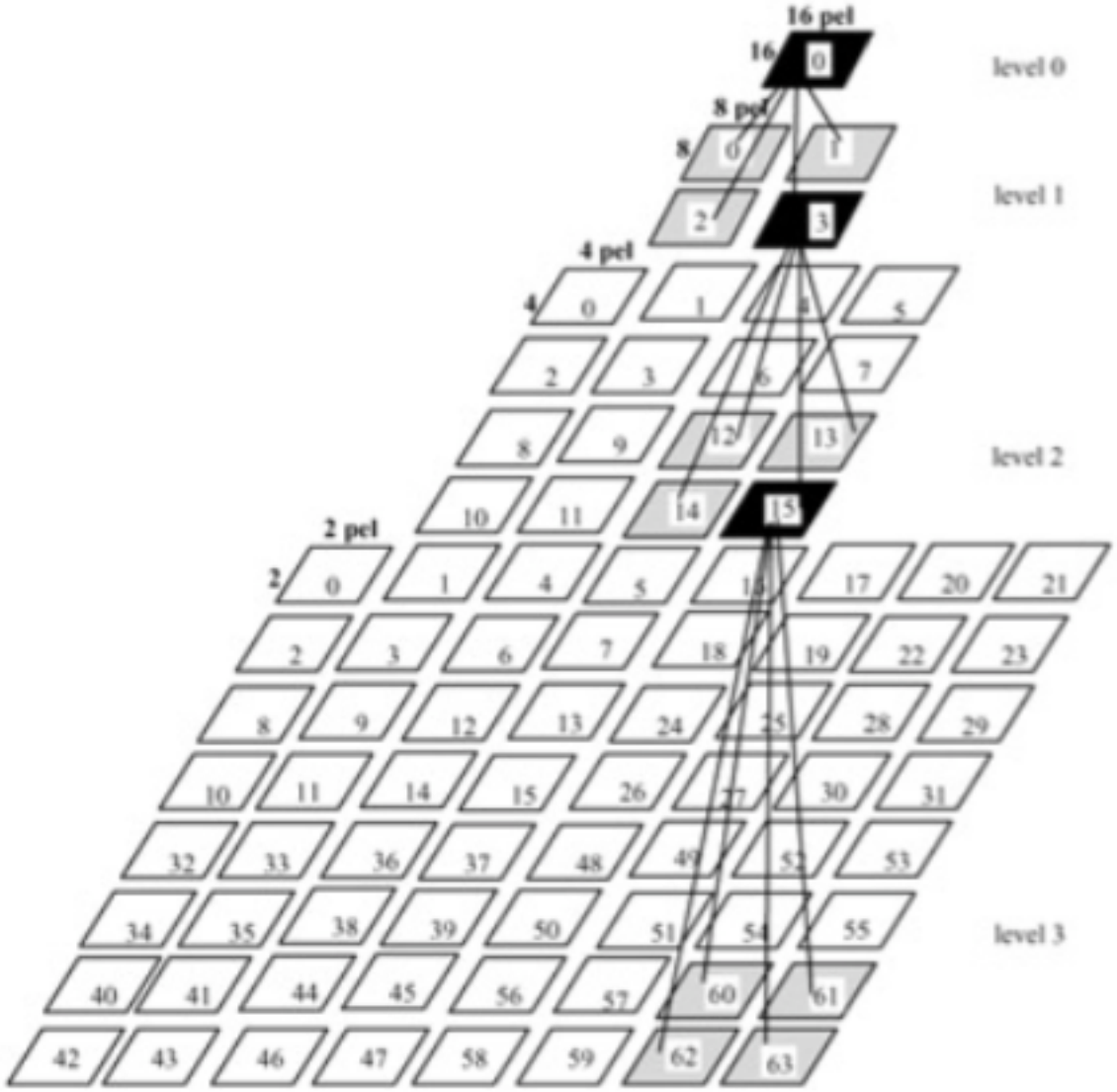
N	0	1	2	3	4	5	6	7	8	9	10	11
c_n	0	7	7	6	5	5	4	3	3	2	1	1
c_n-c_n-1	0	7	0	-1	-1	0	-1	-1	0	-1	-1	0
d_n [-3,4]	0	-1	0	-1	-1	0	-1	-1	0	-1	-1	0
kód	1	011	1	011	011	1	011	011	1	011	011	1

Začíname doľava (výsledok má 25 bitov)

N	0	1	2	3	4	5	6	7	8	9	10	11
c_n	5	5	6	7	7	0	1	1	2	3	3	4
c_n-c_n-1	5	0	1	1	0	-7	1	0	1	1	0	1
d_n [-3,4]	-3	0	1	1	0	1	1	0	1	1	0	1
kód	0101011	1	00	00	1	00	00	1	00	00	1	00

Kódovanie tvarov (videoobjektov) v MPEG4 – quad tree kódovanie

Vstup – tzv. BAB (Binárny Alfa Blok) bloky – veľkosti 16x16 (binárne hodnoty: 0=čierna, 255=biela)



BAB blok vyjadruje hodnotami:

- kde objekt je
- kde objekt nie je

Každý BAB blok je vyjadrený hierarchicky:

- V úrovni 3 (L3) je rozdelený na 64 blokov, rozmeru 2x2
- V úrovni 2 (L2) je rozdelený na 16 blokov zo 4x4 pixelmi
- V úrovni 1 (L1) je rozdelený na 4 bloky zo 8x8 pixelmi
- V úrovni 0 (L0) je rozdelený na 16 blokov zo 16x16

Každý blok na každej úrovni je kódovaný pomocou čísla označovaného ako „index“.

Quad tree úroveň 3

b[0] b[1] b[2] b[3]	**Vypočítanie indexu pre každý zo 64 blokov:** IndexL3=27b[0] + 9b[1] + 3b[2] + 1b[3] pričom • b[i]=2 ak pixel=255, b[i]=0 ak pixelu=0 • dostávame takto 16 rôznych hodnôt intexu (min=0, max=80)
horný_ index[0] horný_ index[0] ľavý_ index[0] index[0] index[1] ľavý_ index[0] index[2] index[3]	**Poprehadzovanie indexov (zvýšenie medziblokovej korelácie) v rámci štvorice indexov na základe situácie nad a vľavo od tejto štvorice:** Ak horný_index[0] < horný_index[1], vymeň index[0] s index[1] a index[2] s index[3] okrem sub-blokov 0,1,4,5,16,17,20 a 21 Ak ľavý_index[0] < ľavý_index[1], vymeň index[0] s index[2] a index[1] s index[3] okrem sub-blokov 0,2,8,10,32,34,40 a 42 Ak horný_index[0] + horný_index[1] < ľavý_index[0] + ľavý_index[1], vymeň index[1] s index[2] okrem sub-blokov 0,1,2,4,5,8,10,16,17,20,21,32,34,40 a 42

Quad tree úroveň 2

Analogický obrázok ako pri L3.	**Vypočítanie indexu pre každý zo 16 blokov úrovne 2 zo 4 indexov úrovne 3** Index_L2=27f(indexL3[0]) + 9f(indexL3[1]) + 3f(indexL3[2]) + 1f(indexL3[3]) pričom • f(x)=0 ak x=0 • f(x)=2 ak x=80 • f(x)=1 ináč
Analogický obrázok ako pri L3.	**Poprehadzovanie indexov (zvýšenie medziblokovej korelácie) v rámci štvorice indexov na základe situácie nad a vľavo od tejto štvorice** • Prehodenie je tym istým spôsobom ako pri leveli 3, t.j. vynechávajú sa prvy riadok a stĺpec pri niektorýchvýmenách.

Quad tree úroveň 1

Analogický obrázok ako pri L3.	**Vypočítanie indexu pre každý zo 4 blokov úrovne 1 zo 4 indexov úrovne 2 ako na druhej úrovni**
	Poprehadzovanie indexov nie je.

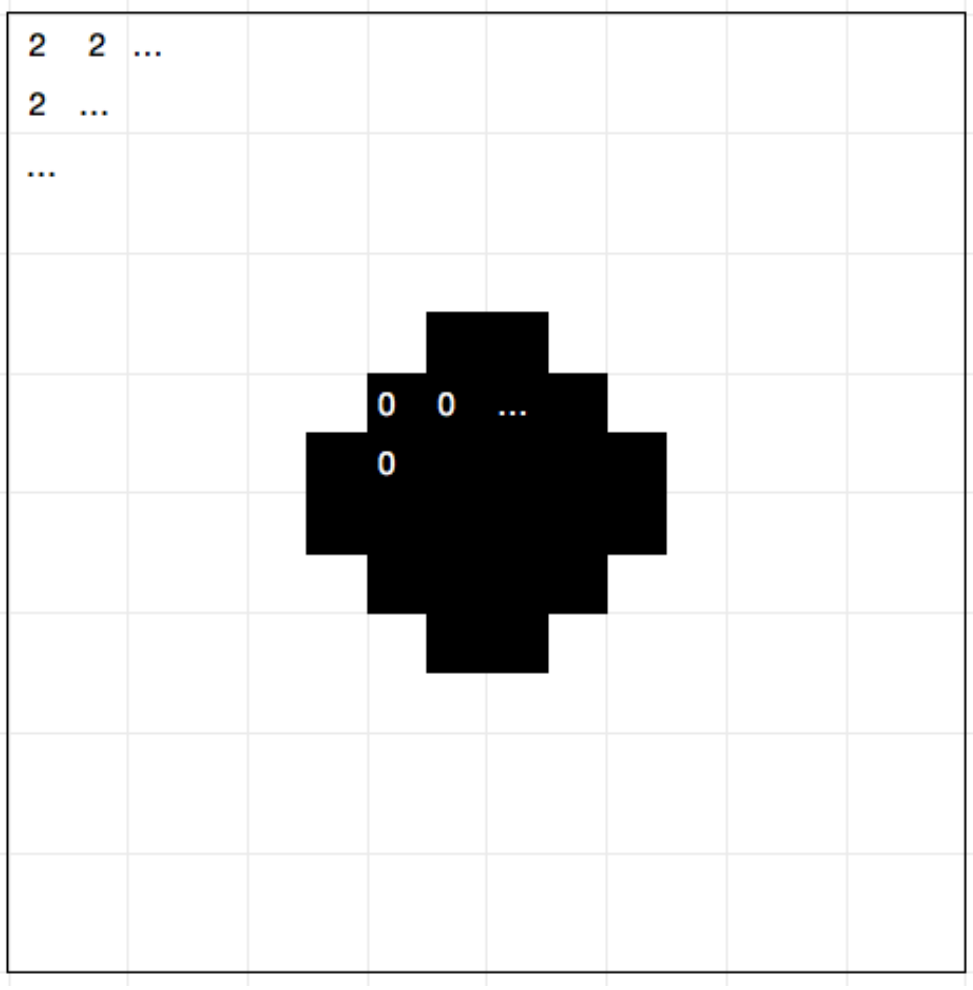
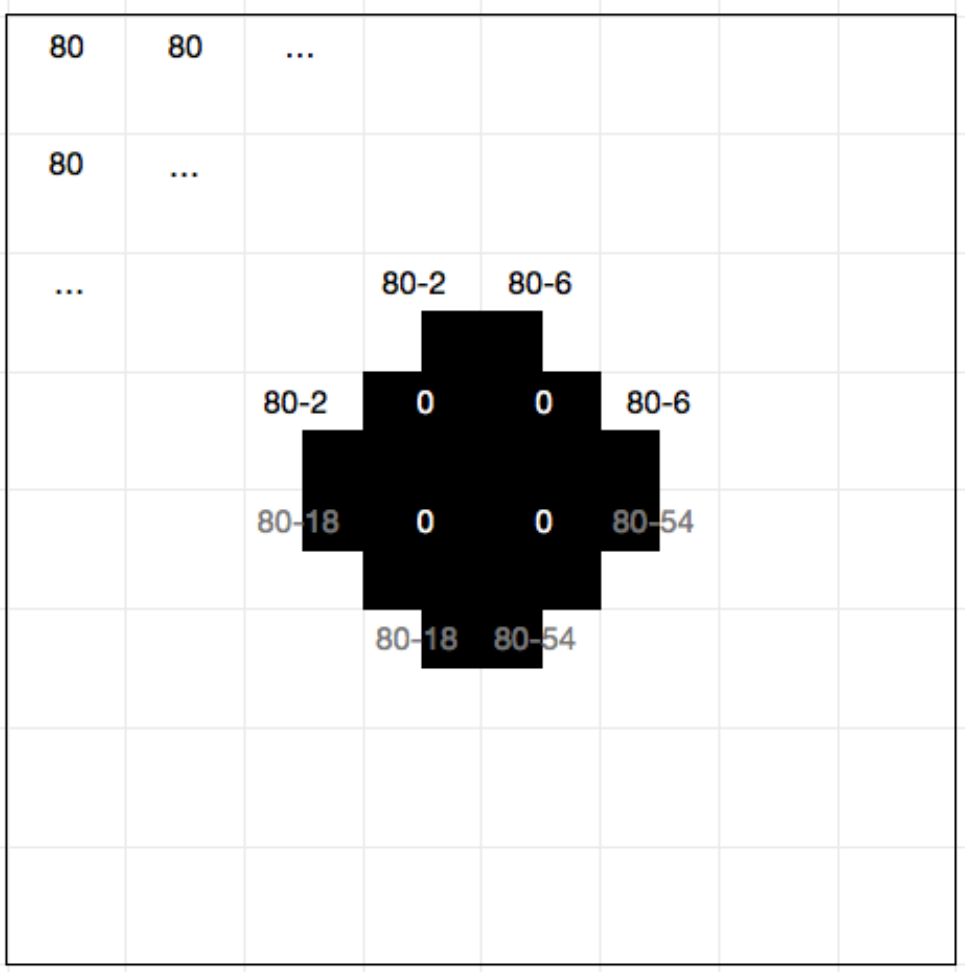
Quad tree úroveň 0

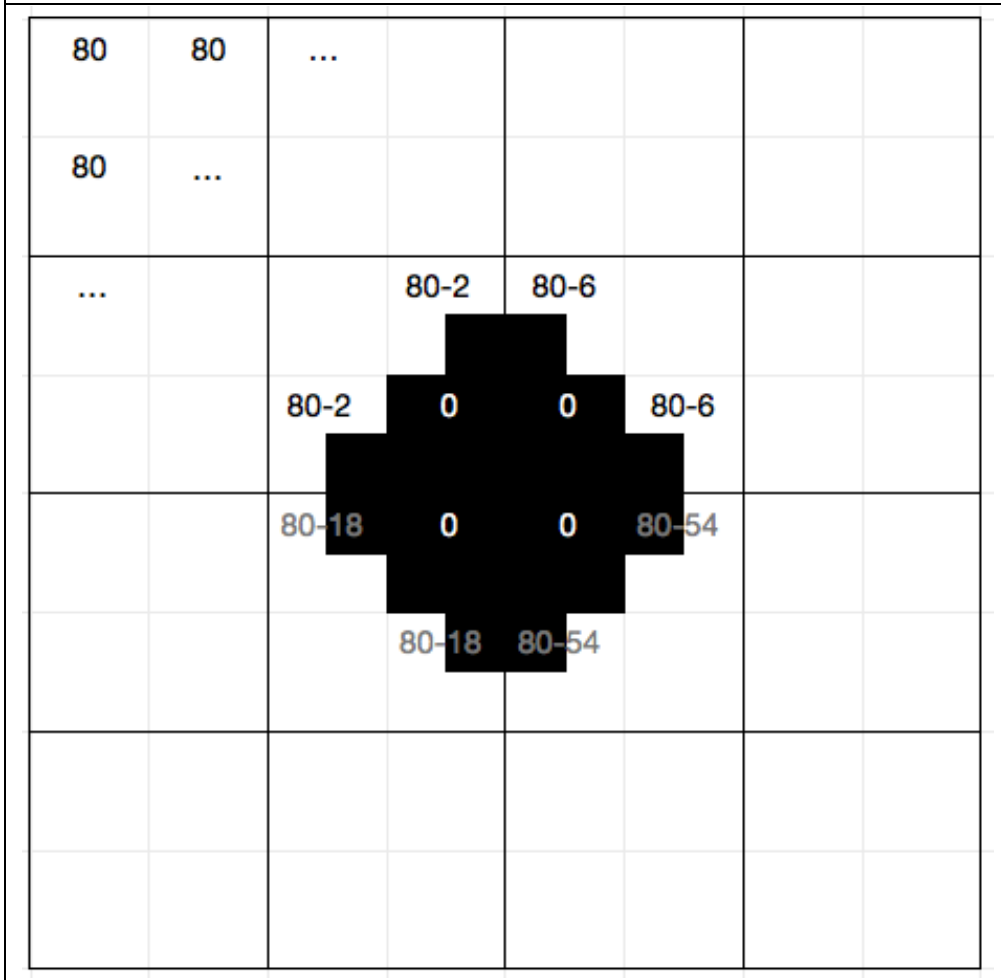
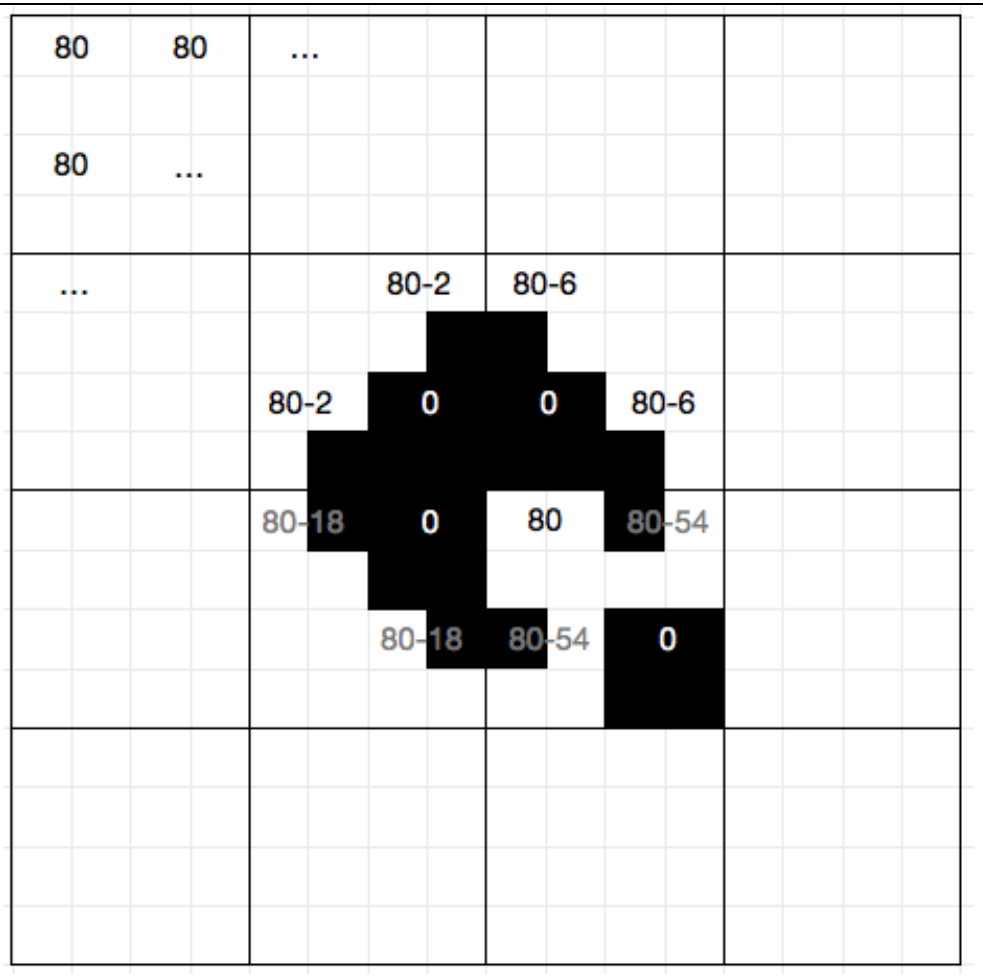
Analogický obrázok ako pri L3.	**Vypočítanie 1 indexu zo 4 indexov úrovne 2 ako na 1 úrovni**

Kódovanie:

- Máme 85 indexov, ktoré určujú ako čo bolo poprehadzované, tie si musíme pamätať
- Indexy kódujeme pomocou Huffmanovho kódovania(HK) (2 druhy)
- Postupujeme po úrovniach v poradí: 0, 1, 2, 3
- V rámci úrovne postupujeme podľa poradia na obrázku stromu.
- Hodnoty:
    - v úrovni 3 je 64 indexov, každý s 16 možnými hodnotami
        - jeden Huffmanov kód (preddefinovaná tabuľka)
    - v ostatných úrovniach majú indexy 80 možných hodnôt
        - druhý Huffmanov kód (preddefinovaná tabuľka)

Príklad

BAB blok 16x16 s naznačenými b hodnotami v leveli 3	Odpovedajúce indexy L3 • 27x2+9x2+3x2+2=80 • týchto 64 indexov sa kóduje pomocou HK
	

Indexy – nepovymienané v L3	Indexy – povymienané v L3 • v rámci belasého kvadrantu došlo k indexov vo vertikálnom aj horizontálnom smere (prvé 2 podmienky)
	

Následne sa pokračuje počítaním indexov pre L2 na základe povymieňaných L3 indexov