Univerzitetna založba
Univerze v Mariboru

UVOD V RAČUNALNIŠKO VEČPREDSTAVNOST IN OBDELAVO MULTIMEDIJSKIH PODATKOV

Štefan Kohek, Borut Žalik

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Uvod v računalniško večpredstavnost in obdelavo multimedijskih podatkov

Štefan Kohek, Borut Žalik

Maribor, 2025

Naslov
Title
Uvod v računalniško večpredstavnost in obdelavo multimedijskih podatkov
Introduction to Computer Multimedia and Multimedia Data Processing

Avtorja
Authors
Štefan Kohek
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Borut Žalik
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Recenzija
Review
Janez Brest
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Matija Marolt
(Univerza v Ljubljani, Fakulteta za računalništvo in informatiko)

Jezikovni pregled
Language editing
AMIDAS, d. o. o. (Špela Vidmar)

Tehnični uredniki
Technical editors
Štefan Kohek
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)
Borut Žalik
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)
Jan Perša
(Univerza v Mariboru, Univerzitetna založba)

Grafične priloge
Graphics material
Viri so lastni, razen če ni navedeno drugače.
Kohek, Žalik (avtorja), 2025

Oblikovanje ovitka
Cover design
Monika Ferk Ovčjak
(Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko)

Založnik
Published by
Univerza v Mariboru
Univerzitetna založba
Slomškov trg 15, 2000 Maribor, Slovenija
https://press.um.si, zalozba@um.si

CIP - Kataložni zapis o publikaciji
Univerzitetna knjižnica Maribor

004.032.6(0.034.2)

KOHEK, Štefan, 1988-
Uvod v računalniško večpredstavnost in obdelavo multimedijskih podatkov [Elektronski vir] / Štefan Kohek, Borut Žalik. - 1. izd. - E-knjiga. - Maribor : Univerza v Mariboru, Univerzitetna založba, 2025

Način dostopa (URL): https://press.um.si/index.php/ump/catalog/book/1051

ISBN 978-961-299-041-1 (PDF)
doi: 10.18690/um.feri.8.2025
COBISS.SI-ID 246759427

Izdajatelj
Issued by
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko
Koroška cesta 46, 2000 Maribor, Slovenija
https://feri.um.si, feri@um.si

Izdaja
Edition
Prva izdaja

Vrsta publikacije
Publication type
E-knjiga

Dostopno na
Available at
https://press.um.si/index.php/ump/catalog/book/1051

Izdano
Published at
Maribor, september 2025

ISBN
978-961-299-041-1

DOI
https://doi.org/10.18690/um.feri.8.2025

Cena
Price
Brezplačno

Odgovorna oseba založnika
For publisher
Prof. dr. Zdravko Kačič,
rektor Univerze v Mariboru

Citiranje
Attribution
Kohek, Š. in Žalik, B. (2025). *Uvod v računalniško večpredstavnost in obdelavo multimedijskih podatkov.* Univerza v Mariboru, Univerzitetna založba. doi: 10.18690/um.feri.8.2025

Kazalo

Slike

Tabele

Seznam najpogostejših kratic

AAC	angl. Advanced Audio Coding
AAC-ELD	angl. Enhanced Low Delay Advanced Audio Coding
AAC-LC	angl. Low Complexity Advanced Audio Coding
AAC-LD	angl. Low Delay Advanced Audio Coding
AAC-SSR	angl. Scalable Sampling Rate Advanced Audio Coding
ADC	angl. Analog–Digital Converter
AES	angl. Advanced Encryption Standard
AIFF	angl. Audio Interchange File Format
AM	angl. Amplitude Modulation
AMR	angl. Adaptive Multi–Rate
AMR-WB	angl. Adaptive Multi–Rate Wideband
ANS	angl. Asymmetric Numeral Systems
ANSI	angl. American National Standards Institute
ADPCM	angl. Adaptive Differential Pulse Code Modulation
APNG	angl. Animated Portable Network Graphics
ASCII	angl. American Standard Code for Information Interchange
ASF	angl. Advanced Systems Format
ASP	angl. Advanced Simple Profile
ATSC	angl. Advanced Television Systems Committee
AV1	angl. AOMedia Video 1
AVC	angl. Advanced Video Coding
AVIF	angl. AV1 Image File Format
AVI	angl. Audio Video Interleave
AVS	angl. Avdio Video Standard
B	angl. Bidirectional
BMP	angl. Bitmap
BPC	angl. Bits Per Channel
Bpf	angl. Bytes per frame
bpf	angl. bits per frame

bpp	angl. bits per pixel
Bpp	angl. Bytes per pixel
CABAC	angl. Context–Adaptive Binary Arithmetic Coding
CAD	angl. Computer–Aided Design
CAF	angl. Core Audio Format
CAVLC	angl. Context–Adaptive Variable Length Coding
CBR	angl. Constant Bit Rate
CTB	angl. Coding Tree Block
CCITT	fra. Comite Consultatif Internationale de Telegraphique et Telephonique
CELT	angl. Constrained Energy Lapped Transform
CIE	fra. Commission Internationale de l'Éclairage
CIF	angl. Common Intermediate Format
CJK	angl. Chinese, Japanese, and Korean
CMY	angl. Cyan–Magenta–Yellow
CMYK	angl. Cyan–Magenta–Yellow–Key
CPMV	angl. Control Point Motion Vector
CRC	angl. Cyclic Redundancy Check
CRT	angl. Cathode–Ray Tube
CSS	angl. Cascading Style Sheets
DAC	angl. Digital–Analog Converter
DPCM	angl. Differencial Pulse Code Modulation
DCT	angl. Discrete Cosine Transform
DST	angl. Discrete Sine Transform
DES	angl. Data Encryption Standard
DIB	angl. Device Independent Bitmap
DM	angl. Delta Modulation
DMVR	angl. Decoder–side Motion Vector Refinement
DNG	angl. Digital Negative
DOS	angl. Disk Operating System
DP	angl. DisplayPort
DPI	angl. Dots Per Inch
DRM	angl. Digital Rights Management
DTMB	angl. Digital Terrestrial Multimedia Broadcast
DV	angl. Digital Video
DVB	angl. Digital Video Broadcasting
DVB-C	angl. DVB–Cable
DVB-H	angl. DVB–Handheld
DVB-S	angl. DVB–Satellite
DVB-T	angl. DVB–Terrestrial

DVI	angl. Digital Visual Interface
DVI-A	angl. DVI–Analog
DVI-D	angl. DVI–Digital
DVI-I	angl. DVI–Integrated
DWT	angl. Discrete Wavelet Transform
ECMA	angl. European Computer Manufacturers Association
EMF	angl. Enhanced Metafiles
EOB	angl. End Of Block
EOF	angl. End Of File
EPS	angl. Encapsulated PostScript
ESC	angl. Epson Standard Code
Exif	angl. Exchangeable image file format
EVC	angl. Essential Video Coding
FIFO	angl. First–In, First–Out
FLAC	angl. Free Lossless Audio Compression
FM	angl. Frequency Modulation
fps	angl. frames per second
GDI	angl. Graphics Device Interface
GIF	angl. Graphics Interchange Format
GOB	angl. Group Of Blocks
GOP	angl. Group Of Pictures
GSM-FR	angl. Global System for Mobile Communications – Full Rate
GSM-HR	angl. Global System for Mobile Communications – Half Rate
HD	angl. High–Definition
HDTV	angl. High–Definition Television
HDMI	angl. High–Definition Multimedia Interface
HE-AAC	angl. High–Efficiency Advanced Audio Coding
HEIF	angl. High Efficiency Image File format
HEVC	angl. High Efficiency Video Coding
HSL	angl. Hue, Saturation, Lightness
HSV	angl. Hue, Saturation, Value
HTML	angl. Hypertext Markup Language
I	angl. Intra–coded
ICC	angl. International Color Consortium
IEC	angl. International Electrotechnical Commission
IFF	angl. Interchange File Format
IGMP	angl. Internet Group Management Protocol
IFD	angl. Image File Directory
IFH	angl. Image File Header
IMDCT	angl. Inverse Modified Discrete Cosine Transform

IPTC	angl. International Press Telecommunications Council
IPTV	angl. Internet Protocol TV
ISDB	angl. Integrated Services Digital Broadcasting
ISDN	angl. Integrated Services Digital Network
ISO	angl. International Organization for Standardization
ISP	angl. Internet Service Provider
ITU	angl. International Telecommunication Union
ITU-T	angl. International Telecommunication Union Telecommunication standardization sector
JBIG	angl. Joint Bi–level Image experts Group
JFIF	angl. JPEG File Interchange Format
JPEG	angl. Joint Photographic Experts Group
JUS	jugoslovanski standard
JVT	angl. Joint Video Team
LCD	angl. Liquid–Crystal Display
LPC	angl. Linear Predictive Coding
LPCM	angl. Linear Pulse Code Modulation
LSB	angl. Least Significant Bit
LUT	angl. Look–Up Table
LZMA	angl. Lempel–Ziv–Markov chain Algorithm
LPC	angl. Linear Predictive Coding
MBpf	angl. MBytes per frame
MDCT	angl. Modified Discrete Cosine Transform
MELP	angl. Mixed Excitation Linear Prediction
MJPEG	angl. Motion JPEG
MLP	angl. Meridian Lossless Packing
MNG	angl. Multiple–image Network Graphics
MNR	angl. Mask–to–Noise ratio
MPEG	angl. Moving Picture Experts Group
MSB	angl. Most Significant Bit
MSE	angl. Root Mean Square Error
MUSICAM	angl. Masking pattern adapted Universal Subband Integrated Coding And Multiplexing
MXF	angl. Material Exchange Format
NTSC	angl. National Television System Committee
OBMC	angl. Overlapped Block Motion Compensation
OCR	angl. Optical Character Recognition
ODF	angl. Open Document Format
ODIF	angl. Office Document Interchange Format
OOXML	angl. Office Open Extensible Markup Language

OTF	angl. OpenType font
P	angl. Predicted
PAM	angl. Pulse Amplitude Modulation
PCM	angl. Pulse Code Modulation
PDC	angl. Pel Difference Classification
PDF	angl. Portable Document Format
PDL	angl. Page Description Language
PEAQ	angl. Perceptual Evaluation of Audio Quality
PAL	angl. Phase–Alternating Line
PNG	angl. Portable Network Graphics
PPI	angl. Pixels Per Inch
PPM	angl. Pixels Per Meter
PPM	angl. Pulse–Position Modulation
PS	angl. PostScript
PSNR	angl. Peak Signal–to–Noise Ratio
PWM	angl. Pulse–Width Modulation
QCIF	angl. Quarter CIF
QT	angl. Quick Time
RGB	angl. Red, Green, Blue
RGBA	angl. Red, Green, Blue, Alpha
RIFF	angl. Resource Interchange File Format
RLE	angl. Run–Length Encoding
RMSE	angl. Root Mean Square Error
ROI	angl. Region Of Interests
RSA	Rivest–Shamir–Adleman
RTCP	angl. Real–Time Transport Control Protocol
RTP	angl. Real–Time Transport Protocol
RTSP	angl. Real–Time Streaming Protocol
SD	angl. Standard–Definition
SDTV	angl. Standard–Definition TV
SECAM	fra. SÉquentiel de Couleur À Mémoire
SGML	angl. Standard Generalized Markup Language
SIL	angl. Sound Intensity Level
IAMF	angl. Immersive Audio Model and Formats
SMIL	angl. Synchronized Multimedia Integration Language
SMPTE	angl. Society of Motion Picture and Television Engineers
SNR	angl. Signal–to–Noise Ratio
SMR	angl. Signal–to–Mask Ratio
SQNR	angl. Signal–to–Quantisation Noise Ratio
SLS	angl. Scalable to Lossless

SI	angl. Switching I
SP	angl. Switching P
SPI	angl. Samples Per Inch
SPL	angl. Sound Pressure Level
SSIM	angl. Structural Similarity Index Measure
STEP	angl. STandard for the Exchange of Product model data
SVC	angl. Scalable Video Coding
SVG	angl. Scalable Vector Graphics
TCP	angl. Transmission Control Protocol
TIFF	angl. Tagged Image File Format
TTF	angl. TrueType Font
UCS	angl. Universal coded Character Set
UDP	angl. User Datagram Protocol
UHD	angl. Ultra High–Definition
UTF	angl. Unicode Transformation Format
VarDCT	angl. Variable Discrete Cosine Transform
VBR	angl. Variable Bit Rate
VGA	angl. Video Graphics Array
VHS	angl. Video Home System
VLC	angl. Variable Length Coding
VOP	angl. Video Object Plane
VVC	angl. Versatile Video Coding
W3C	angl. World Wide Web Consortium
WAV	angl. Waveform Audio File Format
WMA	angl. Windows Media Audio
WMV	angl. Windows Media Video
WMF	angl. Windows MetaFile
WWW	angl. World Wide Web
WYSIWYG	angl. What You See Is What You Get
XML	angl. Extensible Markup Language
XMP	angl. Extensible Metadata Platform

Predgovor

Učbenik, ki je pred vami, je namenjen študentom prve stopnje univerzitetnega in visokošolskega študijskega programa Računalništvo in informacijske tehnologije na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru (UM FERI) ter zajema vsebino predmetov Multimedia in Računalniška večpredstavnost. Morda bo posegel po njem tudi kdo, ki ga zanimajo ključni koncepti in algoritmi obdelave multimedijskih podatkov. Izbrana snov bo vsakemu bralcu zagotovo v pomoč, saj se z multimedijskimi podatki srečujemo vsakodnevno.

Tuja literatura z obravnavanega področja je izjemno bogata in področje opisuje tako iz uporabniškega kot iz inženirskega vidika; razumevanje slednjega pa zahteva precejšnje predznanje. Poleg tega se področje intenzivno razvija. Učbenik zato ponuja postopen in sistematičen pregled temeljnih konceptov obdelave multimedijskih podatkov, pri čemer vključuje tudi najnovejša spoznanja. Poseben poudarek je na predstavitvi osnovnih tipov multimedijskih podatkov, izbranih algoritmov za njihovo obdelavo in najpogostejših multimedijskih formatov. Obširen seznam literature bo študentom zagotovo v pomoč pri samostojnem študiju.

Struktura učbenika sledi osnovnim tipom multimedijskih podatkov, in sicer tekstu, slikam, digitalnemu avdiu in videu. Vsako poglavje se zaključi z naborom vprašanj in praktičnih nalog, kar bo študentom omogočalo dobro pripravo na izpit ter poglobljeno razumevanje obravnavane snovi.

Avtorja

Poglavje 1

Uvod

Multimedija (angl. multimedia) je izraz, ki je nastal v šestdesetih letih v ameriški zabavni industriji. Uvedel ga je Bob Goldstein za promocijo predstave *LightWorks at L'Oursin*, v kateri je hkrati predvajal glasbo in filme ter jih pospremil še s svetlobnimi učinki [14]. Beseda je skovanka latinskih besed **multi** (pomeni *več*) in **media** (pomeni *sredino, center*, nekaj, kar je *vmes*), v današnjem času pa jo razumemo kot *nosilec informacije* (na primer radijski medij).

Človeštvo je skozi svoj razvoj uporabljalo različne nosilce informacij od kamna, glinenih ploščic, papirusa, pergamenta, papirja. Seveda se v tem učbeniku s temi nosilci podatkov ne bomo ukvarjali, a tudi ne z različnimi računalniškimi nosilci informacij, kot so luknjaste kartice, teleprinterski trakovi, magnetni, optični in polprevodniški pomnilni mediji. V tem učbeniku se bomo seznanili, kako **interpretirati** oziroma **predstaviti** binarne podatkovne tokove, shranjene na računalniških pomnilnih medijih ali pridobljene po komunikacijskih kanalih, da jih bodo uporabniki zaznali kot črke, slike, filme, glasbo ali kar njihovo kombinacijo. Prav zato besedo multimedija slovenimo kot **večpredstavnost**.

Multimedijo oziroma večpredstavnost lahko obravnavamo iz dveh vidikov [15]:

- **Uporabniški/aplikacijski vidik.** Od svojega nastanka je multimedija nepogrešljiva na področjih zabavne industrije, kulture, informiranja, oglaševanja, izobraževanja, inženirstva, medicine in drugje. Za podporo tako imenovani kreativni industriji so razvili namensko strojno in programsko opremo, kot so na primer grafične in zvočne kartice ter aplikacije za obdelavo teksta, slik, zvoka in videa. Najbolj znan primer integracije namenske strojne in programske opreme so sistemi za obogateno, navidezno, razširjeno in mešano resničnost.

Nove rešitve na področju multimedije pa imajo tudi širši družbeni vpliv, saj se na primer za podporo kreativni industriji pojavljajo novi študijski programi in novi poklici, kot je študijski program medijskih komunikacij.

- **Znanstveno-inženirski vidik.** Multimedia vključuje široko množico znanj, ki omogočajo zajem, shranjevanje, prenos, obdelavo in predstavitev velikih količin digitalnih podatkov. Posledično je multimedija izjemno živahno interdisciplinarno raziskovalno področje, ki jo podpirajo pomembne znanstvene revije, na primer Multimedia Systems (Springer Nature), Multimedia Tools and Applications (Springer Nature), Journal of Multimedia (Academy Publisher), IEEE Transactions on Multimedia in interesna združenja (na primer ACM SIGMM).

V tem učbeniku se bomo posvetili inženirskemu vidiku večpredstavnosti; natančneje, načinom predstavitve zaporedij digitalnih podatkov, iz katerih lahko tvorimo znake, rastrske in vektorske slike, digitalne zvoke in digitalne videe na način, primeren za človeka. Čeprav gre v vseh primerih za zaporedje bitov, bomo govorili o **multimedijskih podatkih**. Za pravilno interpretacijo zaporedij bitov morajo biti ti organizirani na povsem določen način, čemur pravimo **podatkovni format** ali krajše kar **format**. Po tem, ko podatke interpretiramo/dekodiramo, nad njimi pogosto izvajamo različne **operacije**; mnoge izmed njih ste spoznali že pri drugih predmetih.

Multimedijske tipe podatkov delimo v dve skupini [15]; na **statične** in na **dinamične**. Pri slednjih je čas ključen dejavnik za njihovo ustrezno predstavitev. Glasba, ki se ne odigra v predpisanem ritmu, bo za uporabnika moteča. Tudi video ali animacija, ki ne uspe predvajati zaporedja slik dovolj hitro, bo neprimerna. V takšnih primerih pravimo, da moramo predstavitev multimedijskih podatkov opraviti v **realnem času**. Zaradi velikih količin podatkov, ki jih moramo dostaviti in interpretirati na računalniškem sistemu, ki ga ima na voljo uporabnik, moramo občutno zmanjšati količino podatkov. Pri statičnih multimedijskih tipih čas za dostavo in predstavitev podatkov ni tako kritičen. Četudi bi se na primer rastrska slika rekonstruirala nekaj sekund, bi bila slika za uporabnika povsem uporabna. Dinamični multimedijski tipi so glasba, zvok, animacija in video, statični pa tekst in slike.

Poglavje 2

Tekst

Tekst je v računalništvu najstarejši medijski tip, sestavljen iz znakov/simbolov in končne abecede. Z njim posredujemo informacije v jedrnati in nedvoumni obliki. Lahko predstavlja zapis naravnega jezika, pa tudi zaporedje kod (na primer kode DNK), program v programskem jeziku, šifrirano sporočilo in podobno. Znaki pa niso samo črke, ampak tudi številke, ločila, kontrolni znaki, včasih celo ideogrami, hieroglifi in podobno.

Tekst je **linearen** medij, saj ga obdelujemo/beremo znak za znakom od začetka proti koncu. S pojavom svetovnega spleta se je pojavil tako imenovan **hipertekst** (angl. hypertext), ki pretvori tekst tudi v nelinearen medij, saj s povezavami na posamezne dele teksta lahko po njem tudi *skačemo*.

2.1 Predstavitve teksta

Tekst zapišemo z zaporedjem binarnih kod, pri čemer vsaka koda pomeni natanko en znak. Za preslikavo znakov v kode uporabimo enega izmed **naborov znakov** (angl. character encoding/set); nekaj jih bomo opisali v nadaljevanju.

2.1.1 ASCII in razširjen ASCII

Najpogostejši standardiziran način za kodiranje znakov v elektronskih komunikacijah in v računalništvu je ASCII (angl. American Standard Code for Information Interchange). Prva izdaja standarda datira v leto 1963, trenutno veljavna inačica pa je v uporabi od leta 1986. Združenje IANA (angl. Internet Assigned Numbers Authority) priporoča kratico US-ASCII.

Koda ASCII ima zanimivo zgodovino in sega še v čase pred intenzivno uporabo računalnikov. Ljudje so si vedno želeli izmenjevati sporočila na

daljavo in to jim je omogočal izum elektrike. Za prenos sporočil so najprej izumili telegraf, ki je po prevodniku pošiljal krajše (pikice) ali daljše (črtice) impulze; to je Morsejevo abecedo. Sprejemanje in oddajanje sporočila so lahko opravili le izurjeni posamezniki – telegrafisti. Zato so inženirji dobili idejo, da bi impulze in presledke med njimi združili v skupine, ki bi s kodirnim sistemom krmilili selenoidne tuljavice, te pa bi imele na svojih kotvah vgravirane znake. Ob aktiviranju kotve je vgraviran znak pustil na papirju odtis tako kot pri klasičnem pisalnem stroju. Za to, da bi zapisali vse črke angleške abecede, številke, interpunkcijo in nekaj komand za nadzor naprave (na primer premik v novo vrstico, odmakni od roba za nekaj znakov), je bilo dovolj sedem impulzov v kodi znaka. Napravi, ki je tiskala na daljavo, so rekli teleprinter. V Združenih državah Amerike, takrat vodilni tehnološki sili, je začelo izdelovati teleprinterje več podjetij, žal vsako s svojim kodiranjem. Veliko teleprinterjev je seveda kupila tudi državna administracija, a različni kodirni sistemi so učinkovitost komunikacije zelo ovirali. Zato je takratni ameriški predsednik Johnson izdal ukaz, da morajo vsi teleprinterji, ki jih ima ameriška administracija, podpirati izbran način kodiranja. Ta način so poimenovali kodiranje ASCII.

Kot smo že omenili, je bila koda ASCII 7-bitna. Izmed vseh 128 možnih znakov je 95 vidnih (angl. printable), preostalih 33 pa je nevidnih in so bili namenjeni nadzoru teleprinterja (slika 2.1). ASCII je bila vedno načrtovana kot ena izmed mnogih nacionalnih variant. Tako so druga mednarodna standardizacijska telesa ratificirala standard ISO 646 [16] v letu 1973, ki je zelo podoben standardu ASCII, a je vključeval različne znake neangleških abeced (na primer, ö, õ, ç). V takratni skupni državi je na primer organizacija za standarde JUS definirala kodne nabore YUSCII, in sicer:

- **JUS I.B1.002** [17] (ISO-IR-141) [18], ki je določal znake latinske pisave slovenskega in takratnega srbohrvaškega jezika,
- **JUS I.B1.003** [19] (ISO-IR-146) [20], ki je bil namenjen srbski cirilici, in
- **JUS I.B1.004** [21] (ISO-IR-147) [22], ki je definiral kodiranje za makedonsko cirilico.

Šumniki so tako nadomestili nekatera ločila (ˆ → Č, ˜ → č, [→ Š, { → š, @ → Ž, ` → ž), kar je nekoliko otežilo življenje takratnih programerjev. A kodni nabor se je takrat uveljavil in podobno rešitev uporabljamo na tipkovnicah še danes.

Koda ASCII je kmalu prešla iz sveta teleprinterjev v svet računalnikov, kjer so impulze in presledke nadomestile logične enice in ničle (oziroma

$b_7 b_6 b_5$ → / Bits					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓	b_3 ↓	b_2 ↓	b_1 ↓	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Slika 2.1: Kodni nabor znakov US-ASCII iz leta 1972, kjer so za vse kombinacije bitov od *b1* do *b7* določeni vidni in kontrolni znaki (vir: [1] v javni domeni)

.

ustrezni napetostni nivoji). Ker pa je bila koda ASCII 7-bitna, računalniška beseda pa je že bila uveljavljena kot osembitna, so najpomembnejši bit uporabili za kontrolo prenosa podatkov, torej je osmi bit postal paritetni bit. Kmalu pa je komunikacija znotraj računalniškega stroja postala zanesljivejša, zato paritetni bit ni bil več smiseln; dejansko je samo zmanjševal učinkovitost komunikacije. Zato so v sedemdesetih letih prejšnjega stoletja standardizirali 8-bitno besedo ASCII. Obstoječ standard ASCII s 128 znaki so z dodatkom osmega bita zelo preprosto razširili in zagotovili kompatibilnost z obstoječim standardom ASCII. Takšnemu kodiranju pravimo **razširjen ASCII** (angl. Extended ASCII). Za slovenski jezik se je na operacijskih sistemih DOS uporabljala kodna tabela cp852 oziroma DOS-Latin-2. Na operacijskih sistemih Windows so razvili drugačne kodne tabele, med katerimi so najbolj znane Windows-125x oziroma cp125x. Pri tem x označuje specifično varianto. Za slovenski jezik se uporablja kodna tabela Windows-1250 (za srednje- in vzhodnoevropske jezike).

2.1.2 ISO/IEC 8859

Na pojav novih kodnih tabel oziroma razširitev ASCII se je odzvala tudi mednarodna organizacija za standarde. Ta je skupaj z IEC (angl. International Electrotechnical Commision) izdala standard ISO/IEC 8859 [23], ki ima skoraj enake preslikave, kot so pri kodnih tabelah Windows. Standard ISO/IEC 8859 ima 15 variant (varianta 12 je opuščena), med katerimi so najbolj znane:

- ISO/IEC 8859-1: zahodnoevropski jeziki;
- ISO/IEC 8859-2: srednje- in vzhodnoevropski jeziki, med njimi tudi šest šumnikov slovenskega jezika Č, č, Š, š, Ž, ž, ki so dobili naslednje šestnajstiške kode: C8, E8, A9, B9, AE in BE;
- ISO/IEC 8859-3: turški, malteški in esperanto;
- ISO/IEC 8859-4: estonski, latvijski, litovski, grenlandski;
- ISO/IEC 8859-5: slovanski jeziki, ki uporabljajo cirilico;
- ISO/IEC 8859-6: arabščina;
- ISO/IEC 8859-7: grščina;
- ISO/IEC 8859-8: hebrejščina.

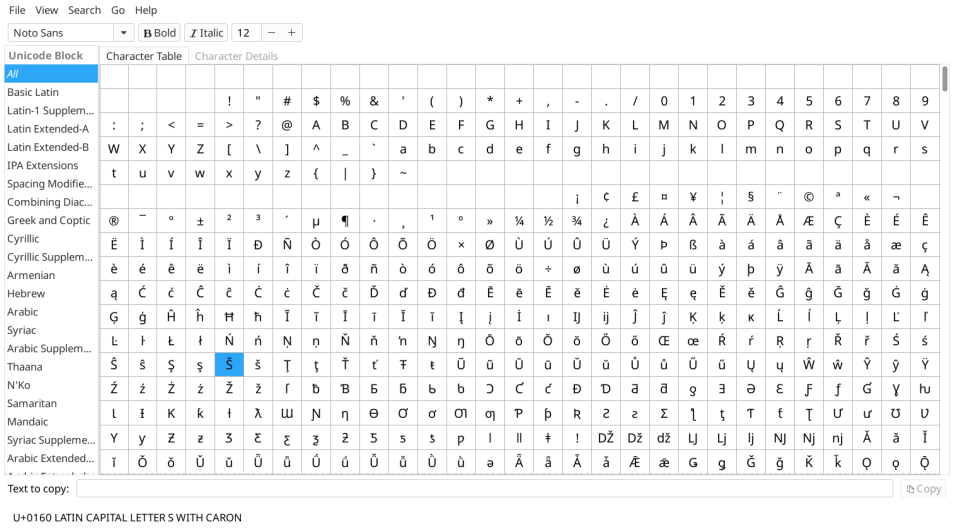
Žal pa tudi ISO/IEC 8859 še zdaleč ni rešil vseh težav, saj mnoge pisave vsebujejo neprimerno več znakov, kot jih zmore predstaviti ta standard.

2.1.3 Standard Unicode

V osemdesetih in devetdesetih letih prejšnjega stoletja so prva globalna računalniška podjetja želela svojim uporabnikom omogočiti pisanje v maternem jeziku, zato so predlagala razvoj ustreznega standarda. Sprva je bilo predvidenih 16 bitov na znak oziroma 65.536 znakov, kar pa še vedno ni zadostovalo za vse pisave. Končno rešitev je prinesel standard Unicode oziroma (ISO/IEC 10646 [24]).

Standard ISO/IEC 10646 vzdržuje konzorcij Unicode [25], ki določa kodni nabor znakov UCS (angl. Universal Coded Character Set) in način zapisa kod znakov oziroma kodiranja znakov. Specifikacije iz standarda Unicode so usklajene z mednarodnim standardom ISO/IEC 10646. Ta način kodiranja znakov je danes večinoma izpodrinil predhodne načine.

UCS ima podporo za 1,1 milijona znakov, pri čemer je z različico 15.1 definiranih 149.813 znakov. Kodna tabela UCS je razdeljena na ravnine, ki



Slika 2.2: Posnetek aplikacije *gucharmap* za pregledovanje kodnega nabora znakov iz standarda Unicode, kjer je na levi strani seznam blokov kodne tabele Unicode, na desni pa so znaki prvega dela kodne tabele.

obsegajo po 65.536 znakov. Prva ravnina je osnovna in določa najpogosteje uporabljane znake (angl. Basic Multilingual Plane – BMP). Vključuje tudi pogostejše pismenke CJK (angl. Chinese, Japanese, Korean). Druga ravnina je namenjena zgodovinskim znakom in simbolom z različnih področij (angl. Supplementary Multilingual Plane – SMP), v tretji ravnini so manj pogoste pismenke CJK (angl. Supplementary Ideographic Plane – SIP), preostale ravnine pa obsegajo manj pogoste znake ali so rezervirane za zasebno rabo znotraj posameznih organizacij.

Kodo znakov UCS praviloma definiramo v šestnajstiškem zapisu. Znak Š ima kodo UCS U+0160, ena izmed pogostejših pismenk CJK pa kodo U+8449, torej je še vedno v prvi ravnini. Redkeje uporabljena pismenka CJK ima kodo U+20021, pri čemer prva številka pove, da gre za tretjo ravnino kodne tabele UCS.

Za ohranitev združljivosti s starejšo programsko opremo je prvih 256 znakov iz UCS enakih kot pri ISO/IEC-8859-1. Kot je prikazano na sliki 2.2, je kodni nabor razdeljen v več blokov. Bralec lahko preveri ujemanje prvih znakov s kodnim naborom US-ASCII. Šumniki za slovenski jezik so znotraj prvega razširjenega bloka latinice (angl. Latin Extended-A).

Za kodiranje znakov UCS se je uveljavilo več formatov. Neposreden zapis znakov je možen s kodiranjem UTF-32 (angl. Unicode Transformation Format-32-bit), ki za zapis uporablja 32 bitov, kar zadostuje za celoten nabor

znakov UCS. Glede na to, da je ta format prostorsko potraten, se v praksi redkeje uporablja. Boljša rešitev je format UTF-16, ki za zapis uporablja 16 bitov za najpogosteje uporabljene znake, torej iz prve ravnine, in 32 bitov za manj pogoste znake. V preteklosti se je uporabljal tudi format UCS-2, ki pa ima fiksno 16-bitno dolžino in zato ne omogoča zapisa vseh znakov UCS. Treba se je zavedati razlike med UCS-2 in UTF-16 iz vidika vzdrževanja starejše programske kode, saj se je UCS-2 pogosto uporabljal v starejših različicah nekaterih programskih jezikov (na primer Java) in sistemskih klicih starejših operacijskih sistemov, na primer Microsoft Windows 95.

2.1.3.1 UTF-8

Najpogosteje uporabljeno kodiranje znakov za zapis spletnih strani je od leta 2009 UTF-8 (angl. Unicode Transformation Format-8-bit), saj je najučinkovitejši način zapisa najpogosteje uporabljanih znakov. Predstavi lahko 1.112.064 znakov, zapisanih z enim, dvema, tremi ali štirimi zlogi. Tabela 2.1 določa položaj in število kontrolnih bitov pri kodiranju UTF-8, na podlagi katere lahko opravimo kodiranje znakov.

Tabela 2.1: Predloga za kodiranje UTF-8

št. zlogov	št. bitov za kodo	začetna koda	končna koda	zlog 1	zlog 2	zlog 3	zlog 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Prvih 128 znakov je kodiranih v US-ASCII in potrebuje en zlog. Naslednjih 1920 znakov zahteva dva zloga in vključuje znake preostalih latinskih abeced (tudi šumnike), grških znakov, cirilice, hebrejščine, arabščine in še nekaterih drugih pisav, kar vidimo na sliki 2.2. S tremi zlogi predstavimo naslednjih 65.536 znakov in z njimi pokrijemo večino znakov CJK. Preostali znaki, ki jih kodiramo s štirimi zlogi, pa vključujejo manj pogoste znake CJK in njihove izpeljanke, znake starinskih pisav ter matematične in piktogramske simbole.

Najpomembnejša lastnost UTF-8 je popolna združljivost z ASCII. Prvih 128 znakov ima namreč enake kode kot ASCII. Kot vidimo v tabeli 2.1, je najbolj pomemben bit (angl. most significant bit – MSB) do vrednosti 127 enak 0, torej ne začne nobenega drugega zloga. Dvo-, tro- in štirizložno kodiranje je označeno z unarno oziroma vejično kodo [26].

Programska oprema, ki uporablja UTF-8, lahko samodejno zazna razširjen ASCII in ga pretvori v UTF-8. Pri uporabi UTF-8 v programski kodi se

moramo zavedati največje omejitve, da zgolj iz števila znakov ne moremo vnaprej izračunati zahtevanega števila zlogov, ki ga bo zahtevala datoteka s tekstom, saj je število zlogov na znak spremenljivo. Oglejmo si primer kodiranja znaka € v UTF-8:

1. Koda Unicode za € je U+20AC.
2. Glede na tabelo 2.1 ugotovimo, da je koda U+20AC med U+0800 in U+FFFF, to pomeni, da bomo potrebovali za njeno kodiranje tri zloge.
3. Če U+20AC zapišemo v binarni obliki, dobimo 00100000 10101100.
4. Ker bo koda dolga 3 zloge, bo število 3 zapisano z unarno kodo kot 1110. V prvi zlog vpišemo še prve štiri bite naše kode, torej je prvi zlog 11100010.
5. Preostala dva zloga se začneta z MSB 10, preostalih 12 bitov pa razporedimo v zloge po vrsti. Končna koda je torej: 1110**0010** 10**000010** 10**101100**, kar zapišemo v šestnajstiškem zapisu kot 0xE2, 0x82 in 0xAC.

Na podoben način izvedemo dekodiranje. Dekodirajmo štiri zloge: 0xF0, 0x9F, 0x98 in 0xBC:

1. Kodo zapišemo z biti: 11110000 10011111 10011000 10111100.
2. Iz prvih bitov ugotovimo, da je znak zapisan s štirimi zlogi.
3. Glede na tabelo 2.1 izluščimo dejanske bite: 11110**000** 10**011111** 10**011000** 10**111100**, torej dobimo: 00001 11110110 00111100.
4. Končni rezultat je torej 0x0001F63C oziroma U+1F63C.

2.2 Operacije nad tekstom

Nad tekstom lahko izvajamo najrazličnejše operacije. Veliko izmed elementarnih operacij nad tekstovnimi nizi že poznate, zato jih bomo samo omenili. Najbolj elementarne operacije so operacije urejanja (angl. editing), ki jih delimo na operacije nad znaki (vrinjanje, brisanje, prepisovanje) in operacije nad nizi (iskanje niza v tekstu, zamenjave delov besedila, izrezovanje in kopiranje). Najbolj znani algoritmi za iskanje v zaporedju so Knut-Morris-Prat, Rabin-Karp, Horspool in Sunday [26].

2.2.1 Jezikovnoodvisne operacije

Naprednejše operacije temeljijo na poznavanju jezika (naravnega ali programskega). Najbolj znani so črkovalniki, ki samodejno zaznajo napake pri pisanju teksta [27]. Sledijo popravljalniki slovnice [28] in samodejni prevajalniki iz enega v drug jezik [29], iskalniki sopomenk in nadpomenk ter samodejno deljenje besed [30]. Za študente računalništva so zanimiva predvsem orodja za samodejno preimenovanje spremenljivk in barvanje delov teksta glede na pomen v programski kodi (angl. syntax highlighting), kar lahko realiziramo s klasičnimi algoritmi iz teorije prevajalnikov [31] ali s strojnim učenjem [32].

2.2.2 Šifriranje

Za šifriranje teksta je bilo razvitih veliko pristopov. Prvi šifrirnik je tako imenovana Cezarjeva šifra, kjer se posamezna črka nadomesti z drugo črko, ki je določeno število mest naprej v abecedi. Glede na to, da je pri tem postopku možno hitro uganiti število mest, se v današnjem času uporabljajo naprednejši šifrirniki, kot so na primer DES (angl. Data Encryption Standard), AES (angl. Advanced Encryption Standard), RSA (Rivest–Shamir–Adleman), Blowfish in Serpent [33].

2.2.3 Stiskanje

Pri multimedijskih podatkih pričakujemo veliko podatkov, zato je stiskanje ključna naloga. Tekst je morda izjema, saj je prostorsko najmanj zahteven. Stiskamo ga samo brezizgubno. Najbolj znani osnovni algoritmi stiskanja podatkov so Huffmanovo kodiranje, LZ77 in aritmetično kodiranje, ki jih obravnavajo že drugi študijski predmeti [26]. V zadnjih letih pa se je uveljavil postopek stiskanja ANS [34, 35] (angl. Asymmetric numeral systems), ki je po stopnji stiskanja primerljiv z aritmetičnim kodirnikom, vendar računsko manj zahteven. Pri tekstu je smiselno tudi stiskanje s slovarjem; najbolj znani algoritmi so LZ77, LZ78, LZW, Deflate in LZMA (Deflate si bomo ogledali v razdelku 2.2.3.1). Za posamezne predstavitve teksta obstajajo tudi domensko odvisni algoritmi za stiskanje, na primer za XML [36, 37] ali HTML [38].

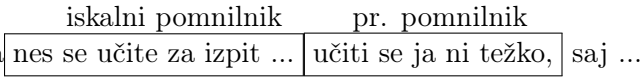
Stiskanje teksta je še danes izziv. Hutterjeva [39] nagrada se podeljuje avtorjem algoritmov stiskanja, ki jim uspe 1 GB veliko tekstovno datoteko z besedilom iz Wikipedije stisniti na manjšo velikost kot predhodni algoritmi. Seveda se pri določitvi stopnje stiskanja upošteva kombinacija programa in stisnjene datoteke, sicer bi lahko 1 GB datoteko premaknili v slovar programa za stiskanje. Z datotečnim formatom ZIP lahko 1 GB veliko

tekstovno datoteko stisnemo na 300 MB. Trenutni rekorder z dne 2. 2. 2024 je Kaido Orav, ki mu je datoteko uspelo stisniti na 112 MB [39]. Zadnji rekorderji pri stiskanju temeljijo na kombinacij tehnik strojnega učenja, obdelave naravnega jezika in uporabe slovarja.

2.2.3.1 Algoritem stiskanja Deflate

Eden izmed najpopularnejših algoritmov za stiskanje podatkov je PkZIP, ki ga je razvil P. W. Katz [40]. Katz je ustanovil podjetje PKWARE, Inc., in za program PkZIP dobil patent [41][1]. Patent pa je bilo možno zaobiti, kar sta izkoristila Gailly in Adler [42] ter razvila Deflate, ki sta ga ponudila v knjižnicah ZLIB [43, 42] in GZIP [44]. Knjižnici nista bili zaščiteni z nobenim patentom, prav tako avtorja nista zahtevala nobenega licenciranja. Deflate, neobremenjen s patenti in licencami, je tako postal del protokola HTTP in datotečnih formatov PDF, PNG in MNG [7].

Deflate kombinira algoritem LZ77 s Huffmanovim kodiranjem. LZ77 pregleduje niz podatkov z **drsečim oknom** (angl. sliding window), ki je razdeljeno v **iskalni** (angl. search buffer) in **primerjalni pomnilnik** (angl. look-ahead buffer), kot vidimo v primeru na sliki 2.3.



Slika 2.3: Ideja LZ77

LZ77 poišče najdaljše ujemanje med začetkom niza v primerjalnem pomnilniku z nizom v iskalnem pomnilniku. Če ujemanje najde, pošlje na izhod žeton v obliki trojčka *[odmik, dolžina, naslednji znak]*. V zgornjem primeru bi poslali žeton *[7, 4, i]*. Tretjo komponento trojčka potrebujemo v primeru, ko ujemanja nismo našli, s čimer zagotovimo premik drsečega okna. Takrat pa, žal, pošljemo na izhod zelo drag žeton *[0, 0, naslednji znak]*, ki slabi učinkovitost stiskanja, zato so se mu mnogi poskušali izogniti [7]. Tudi Deflate uspešno zaobide ta primer; s Huffmanovimi kodami hkrati kodira znake in dolžine ujemanja, posebej pa dolžine. Za to seveda potrebuje dve Huffmanovi tabeli:

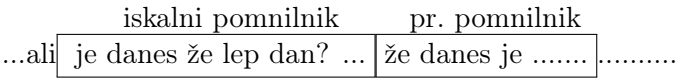
- prva je za znake in dolžine – znaki so vedno predstavljeni s kodami ASCII, torej z osmimi biti, osem bitov pa je rezerviranih tudi za dolžine;

[1] Patent je leta 2010 že potekel.

- druga je za odmike – zaloga vrednosti razdalj je odvisna od velikosti iskalnega pomnilnika, ki je v primeru Deflate 32 kB (2^{15}).

Deflate najprej tvori par simbol-razdalja in mu določi Huffmanovo kodo, ki jo zapiše v izhodni bitni niz. Tej kodi običajno, takrat ko smo našli ujemanje, sledi še Huffmanova koda odmika. V primeru neujemanja iz prve Huffmanove kode vemo, da kodirnik razdalje ni zapisal.

Naslednja izboljšava algoritma Deflate je preverjanje, ali ni bolje kratkih zaporedij kodirati tako, da zakodiramo samo prvi znak in ne ujemanja. Poglejmo si primer na sliki 2.4.



Slika 2.4: Preskok kratkega ujemanja

Najdaljše ujemanje je na položaju 11 v dolžini treh znakov. Preden kodirnik uporabi to ujemanje, preveri, ali ne bi bilo bolje shraniti prvega znaka (to je ž) in poiskati ujemanje, ki sledi. V našem primeru bi se to dejansko zgodilo, saj je naslednje ujemanje e␣danes␣ bistveno daljše od prvega že␣. Deflate uporabo te izboljšave nadzira z uporabniško nastavljivim parametrom, ki določa tri načine delovanja algoritma:

- način *normal*: kadar je prvo ujemanje daljše od vnaprej nastavljenega parametra, drugega ujemanja ne opravimo;
- način *high-compression*: iskanje drugega najdaljšega ujemanja opravimo vedno;
- način *fast*: iskanje drugega ujemanja ne opravimo nikoli.

Deflate podatke razdeli v bloke in vsakega od njih stisne neodvisno. Bloki so lahko različne dolžine, dolžino pa določi kodirnik glede na razpoložljivo količino pomnilnika. Dekodirnik mora biti zmožen dekodirati bloke ne glede na njihovo velikost. Vsak blok je lahko zapisan v treh načinih:

1. Način **brez stiskanja.** Ta način uporabimo pri delih datotek, ki so nestisljiva, kot na primer povsem naključni podatki ali podatki, ki so že bili stisnjeni. Blok brez stiskanja začne z oznako načina, ki mu sledi dolžina bloka, zapisana s 16 biti.
2. Način **s predefiniranimi kodirnimi tabelami.** Deflate v tem primeru uporablja dve vnaprej pripravljeni Huffmanovi tabeli, ki smo ju omenili že prej. Blok se začne s kodo načina, zaključi pa s Huffmanovo kodo EOB (angl. end of block).

3. Način **s prilagojenimi kodirnimi tabelami.** Vnaprej pripravljene Huffmanove kodirne tabele seveda niso optimalne. Deflate omogoča, da kodirnik za posamezen blok sestavi optimalni kodirni tabeli, ki pa ju seveda mora vpisati v glavo bloka. Če sta ti dve kodirni tabeli kratki, se to izplača. Tudi ta blok se zaključi s kodo EOB.

Novejša nadgradnja Deflate je Brotli [45], ki postopoma nadomešča Deflate/GZIP tudi pri protokolu HTTP [46].

2.2.4 Oblikovanje teksta

Za tiskanje oziroma izris teksta na prikazovalniku moramo le-tega predhodno oblikovati. Najprej moramo določiti videz črk in postavitev elementov besedila. Oblikovanje je lahko **interaktivno**, torej po paradigmi, kar vidiš, to tudi dobiš (angl. What You See Is What You Get – WYSIWYG), ali **neinteraktivno**. V slednjem primeru mora uporabnik tekst urediti, pognati **oblikovalnik** in šele nato lahko tekst prikaže na izhodni napravi. Najbolj znan primer neinteraktivnega oblikovalnika je TEX.

Tipografija je tehnika določanja videza besedila, da je berljivo, razumljivo in vizualno privlačno. S tipografijo določamo obliko znakov oziroma tipografsko družino (angl. typeface, font family), velikost pisave in vse drugo, kar vpliva na videz besedila. **Tipografska družina** oziroma družina pisav ali črkopis (angl. font family, typeface) določa obliko in značilnosti znakov/črk. Tipografska družina za vsak znak določa obliko – **glif** (angl. glyph). Kombinacija izbrane tipografske družine, velikosti, debeline, naklona in širine določa tipografsko obliko – **pisavo** (angl. font).

2.2.4.1 Predstavitev pisav

Obliko posameznih črk lahko predstavimo na dva načina. Prvi in najbolj preprost način je, da posamezne znake predstavimo z rastrsko sliko, torej z 2D-matriko vrednosti, kot bi jih videli na prikazovalniku. Glavna težava pri takšnem načinu predstavitve je v izgubi jasnosti oblike pri povečevanju ali spremembi nagiba znakov. Zaradi tega se pogosteje uporablja drugi način, to je **parametrični opis** obrisov posameznih znakov, in ki je sestavljen iz točk, daljic in krivulj. Na takšen način lahko posamezne znake poljubno povečujemo in nagibamo brez izgube jasnosti. Parametrični opis pa je pred izrisom treba pretvoriti v rastrsko sliko; pravimo, da opravimo **rasterizacijo** za konkretno izhodno napravo. Več o rasterizaciji bomo zapisali v naslednjem poglavju.

Za prikaz teksta na prikazovalniku moramo najprej naložiti datoteke z opisom oblike posameznih črk, torej datoteke, ki vsebujejo glife za podprte

znake. Pri tem je treba preveriti ali pisave podpirajo želen nabor znakov (Unicode ali ASCII). V primeru kodnega nabora ASCII vseh možnih oblik ni veliko, če želimo s pisavo pokriti celotni kodni nabor Unicode, pa imamo večji izziv, saj je vseh možnih znakov več tisoč. Zaradi tega so nekatere pisave, kot je na primer *Noto*, razdeljene v pakete za posamezen nabor znakov, na primer za nabor znakov CJK [47]. Pri uporabi je treba upoštevati še licenčne pogoje, saj nekatere pisave, ki so priložene operacijskemu sistemu, zahtevajo licenčni operacijski sistem [48].

Najpogostejši datotečni formati za pisave uporabljajo vektorski zapis obrisov pisav in so [49]:

- Adobe Type 1 (1982), ki za opis oblik uporablja zapis PostScript in Bézierove krivulje tretjega reda (angl. PostScript fonts). Datotečni format je standardiziran (ISO/IEC 9541 [50]).

- TrueType (Apple, 1991) [51] je prvi široko razširjen način parametričnega zapisa pisav. Oblike pisav so predstavljene z lomljenkami in Bézierovimi krivuljami drugega reda, ki so hitrejše za rasterizacijo kot Bézierove krivulje tretjega reda. Te pisave so na voljo v obliki datotek TTF in so še danes zelo razširjene na svetovnem spletu in v programski opremi (na primer pisarniški paketi in operacijski sistemi).

- OpenType (1996) je standardiziran (ISO/IEC 14496-22:2019 [52]) in zasnovan na osnovi TrueType in PostScript. Postopoma nadomešča starejše formate. Datoteke imajo končnico OTF (včasih tudi TTF). Datoteke z zbirkami več pisav imajo končnico OTC ali TTC.

- WOFF (angl. Web Open Font Format [53]) je specifikacija konzorcija W3C, namenjena prenosu pisav po spletu ter vključuje pisave TrueType, OpenType in metapodatke v obliki XML.

V datotekah s pisavami so zraven obrisa tudi dodatne informacije, na primer kako naj se pisava prikaže v različnih velikostih, da dosežemo dobro čitljivost (angl. hinting). Zelo pogost parameter je še **zarezovanje** (angl. kerning), s katerim določamo razmik med črkami, na primer v parih črk AV in ff praviloma želimo manjši razmik.

Tudi za zapis rastrskih pisav obstaja več formatov, med katerimi so najbolj znani:

- Bitmap Distribution Format (BDF, Adobe) [49],

- Portable Compiled Format (PCF), ki je izboljšan glede na BDF in ga najdemo v okenskem sistemu Xorg [49],

- PC Screen Font (PSF), ki se uporablja v konzoli operacijskega sistema Linux [49] in
- OpenType, ki se je najbolj uveljavil.

2.2.4.2 Izris teksta

Da lahko oblikovan tekst prikažemo na prikazovalniku, moramo najprej zagnati oblikovalnik teksta. Če uporabljamo urejevalnik WYSIWYG, se oblikovalnik izvede samodejno, v nasprotnem primeru ga moramo izvesti ročno, kjer je najbolj znano orodje TEX. Ob zagonu oblikovalnika se sama vsebina in informacije o videzu pošljejo v **razporejevalnik teksta** (angl. text layout engine). V razporejevalniku teksta se pridobi seznam potrebnih simbolov (**glifov**), na podlagi katerega se nato naložijo datoteke z zahtevanimi pisavami. V naslednjem koraku se znaki zapišejo z zaporedjem simbolov oziroma glifov. Na podlagi informacij o tipografiji, kot sta na primer velikost pisave in zarezovanje, se glifi razporedijo znotraj dokumenta. Primer namenske knjižnice za razporejanje teksta je Pango [54].

2.2.5 Shranjevanje teksta

Tekst najlažje shranimo v datoteko tako, da zapišemo zaporedje kod za posamezne znake, na primer uporabimo kodiranje UTF-8. V primeru oblikovanja teksta moramo zraven zapisati dodatne informacije, kot sta postavitev znakov in oblika znakov. Pri tem je treba upoštevati več vidikov, na primer ali želimo tekst zgolj prikazati, ali ga želimo tudi urejati, ali ga želimo urejati na način WYSIWYG, ali želimo tekst strukturirati v poglavja in podpoglavja ipd.

Oblikovanje teksta najlažje izvajamo z aplikacijami po paradigmi WYSIWYG. Najbolj znane aplikacije so Microsoft Word, WordPerfect, StarOffice, OpenOffice/LibreOffice [55]. Te aplikacije so v preteklosti uporabljale namenske datotečne formate, kot sta DOC in RTF, ter so bile vezane na specifično različico aplikacije.

2.2.5.1 Označen tekst

Formati za oblikovanja teksta imajo preveč podrobnosti, da bi jih uporabnik lahko urejal ročno. Tekst ima tako vsebino in obliko. Vsebino predstavlja zaporedje znakov, obliko pa določimo z vstavljanjem kontrolnih informacij v samo besedilo. V tem primeru govorimo o **označenem tekstu** (angl. markup text ali markup language). Zelo znan je TEX. ISO je sprejel standard SGML (angl. Standardized General Markup Language, ISO 8879:1986 [56]),

na podlagi katerega sta bila razvita XML in HTML. [57]. Za dokumentiranje programov se med programerji uveljavljata tudi formata Markdown [58] (datoteke MD) in AsciiDoc [59, 60] (datoteke ADOC).

2.2.5.2 Strukturiran tekst

Besedilo je zelo pogosto organizirano v poglavja in podpoglavja, deli besedila so podobno oblikovani, vsebuje lahko enačbe, slike in tabele. Neposredna nadgradnja označenega teksta je strukturiran zapis teksta. LaTeX na primer nadgrajuje TeX z ukazi za strukturiranje besedila v poglavja in podpoglavja, nadzira številčenje slik, enačb, tabel, referenc in poskrbi za tvorbo kazal. Za shranjevanja dokumentov je bil razvit standard ODA (angl. Office Document Architecture ali Open Document Architecture v novejšem standardu CCITT T.411-T.424, ISO 8613 [61]) in spremljajoč standard za dejansko kodiranje ODIF (angl. Office Document Interchange Format). Čeprav je ta standard manj znan, predstavlja temelje bolj razširjenih standardov za shranjevanje dokumentov, kot sta OOXML – Office Open XML (ECMA-376 [62], ISO/IEC 29500-1:2012 [63]) in ODF – Open Document Format for Office Applications (ISO/IEC 26300 [64]). Oba standarda namreč uporabljata označen tekst na podlagi XML, organizacijo besedila v poglavja in ločeno definirane sloge.

2.2.5.3 Jeziki za opis strani

Datotečni formati za shranjevanje teksta so postali zelo obsežni. Standard ECMA-376 je zapisan na več tisoč straneh. Če želimo tekst shraniti zgolj za prikaz na drugi napravi, lahko to storimo preprosteje. Izhod oblikovalnika teksta je lahko ali rastrska slika ali pa natančna navodila za konstrukcijo bitne slike v jeziku za opis strani (angl. Page Description Language, PDL). Zapis v jeziku za zapis strani je na višjem nivoju kot zgolj rastrska slika strani, a na nižjem nivoju od prej omenjenih formatov za shranjevanje teksta. Poznamo več jezikov za opis strani:

- ESC/P (angl. Epson Standard Code for Printers) [65] je bil namenjen matričnim tiskalnikom v novejših razširitvah pa tudi drugim tipom tiskalnikov.
- PCL (angl. Printer Command Language) je razvil Hewlett-Packard [66] za matrične, brizgalne in laserske tiskalnike.
- PostScript [67] je razvil Adobe. PostScript je dejansko programski jezik, ki se izvaja znotraj tolmača in omogoča simulacijo Turingovega stroja [68]. Datoteke PostScript imajo končnico .PS in .EPS za posamezne skice.

- Tudi PDF (angl. Portable Document Format) je razvil Adobe in je mednarodni standard (ISO 32000 [69]). Ukazi za izris so podmnožica ukazov PostScript, saj je nadzor toka izvajanja kode odstranjen. Glede na PostScript pa uvaja nove funkcionalnosti, na primer podporo za prosojnost in dodatne medijske tipe (na primer 3D-modeli) ter hiperpovezave. Za arhiviranja se priporoča poenostavitev PDF/A (ISO 19005 [70]), saj hrani vse potrebne informacije za prikaz (na primer pisave) v eni datoteki.

- Med jezike za opis strani uvrstimo tudi DVI (angl. Device Independent) [71], ki je izhod iz programa TEX. DVI se ponavadi takoj pretvori v datoteke PostScript ali PDF, čeprav obstajajo tudi vizualizatorji zanj (na primer YAP, ki je del distribucij LATEX).

2.2.6 Zajem teksta

Zajem teksta najpogosteje poteka z vhodnimi napravami (na primer tipkovnica, prikazovalnik na dotik) in urejevalnikov besedila. Alternativni vir teksta je digitalizacija obstoječega teksta. Pri tem je ključno **optično prepoznavanje znakov** (angl. Optical Character Recognition – OCR), ki rastrsko sliko pretvori v besedilo [72]. Preprostejši programi OCR opravijo razpoznavo na nivoju znakov, naprednejši pa vključujejo tudi jezikovnoodvisne informacije za samodejno popravljanje napak pri postopku razpoznavanja. Te so posledica šuma, ki nastane zaradi neoptimalne poravnave besedila ali zahtevne tipografije. Naslednji vir digitalizacije je zajem besedila rokopisov (angl. handwriting recognition). Danes rokopis pogosto zajemamo iz prikazovalnikov na dotik ali iz namenskega pisala [72]. Zelo pomemben vir je tudi pretvorba govora v besedilo (angl. speech recognition) [73].

2.2.7 Steganografija

Steganografija/prikrivanje (angl. steganography) je tehnika **skrivanja informacije** v vidnem sporočilu. Legenda pravi, da je leta 440 pr. n. št. Histiaeus pobril glavo svojemu najbolj zvestemu sužnju in nanjo vtetoviral sporočilo. Ko so lasje zrasli, so zakrili sporočilo in suženj je lahko s sporočilom odšel [74]. Med ameriško državljansko vojno so intenzivno uporabljali nevidno črnilo. Med drugo svetovno vojno so skrivali sporočila v mikropike. Sporočilo so pomanjšali na velikost pike in ga vstavili v tekst. Tekst, zapisan po vnaprej dogovorjenem načinu, lahko prekriva osnovno sporočilo, kot na primer:

»Velika ekipa čaka pred računalnikom. Ena dama skriva temni avtomobil. Ven naj odnese stvari takoj!«

skriva sporočilo »večpredstavnost«. Ga najdemo?

Seveda je steganografija še kako zanimiva tudi v digitalni dobi. S steganografijo je močno povezano tudi digitalno žigosanje (angl. watermarking). Pri žigosanju nas zanimata dve informaciji: kdo je lastnik dokumenta, četudi je bil ta delno spremenjen, in ali je dokument verodostojen, to je, da ni bil spremenjen.

Tekst, kot eden izmed multimedijskih tipov, je za prikrivanje stegoinformacije še najmanj primeren, saj je zelo definiran in jedrnat medij. Kljub temu poznamo kar nekaj možnosti [75, 76]. Metode delimo glede na to, ali manipuliramo z videzom teksta ali dejansko z vsebino. Pri tem lahko tvorimo nov tekst ali spremenimo obstoječega. Preprostejše metode [75] si na kratko oglejmo v nadaljevanju, pregled naprednejših pa najdemo v [76].

2.2.7.1 Pomik vrstic in besed

Skrito sporočilo lahko v sliko besedila vstavimo tako, da posamezne vrstice (stegovrstice) besedila pomaknemo v navpični smeri za majhno vrednost δ glede na neko referenčno nevidno črto [77]. Premik stegočrte lahko določimo na dva načina, s čimer definiramo tudi postopek dekodiranja:

1. Vse črne piksle na strani preštejemo po rastrskih vrsticah in dobimo histogram/profil črnih pikslov. Vrstice s tekstom so tako jasno ločene, saj je število pikslov precej večje od praznega mesta med vrsticami tudi ob upoštevanju šuma. Za vsako takšno gručo izračunamo centroid, ki predstavlja našo navidezno črto, glede na katero opravimo navpični premik vrstic za δ; recimo za $-\delta$ za bit 0 in za $+\delta$ za bit 1. Zaznavo vpisanega bita opravimo na dva načina. V prvem primeru imamo na voljo originalni profil (izvorno besedilo, v katero nismo vnesli prikrite informacije). V tem primeru je zaznava premika navidezne črte zanesljiva tudi v primeru dodatnega šuma. V drugem primeru pa originalnega profila nimamo. Zato najprej izračunamo vse razdalje med navideznimi črtami centroidov in izračunamo povprečno razdaljo, ki nam potem služi za oceno premika posameznega centroida.

2. Izhajamo iz predpostavke, da je razdalja med osnovnimi črtami (angl. baseline), na katerih ležijo črte, konstantna. Če za δ premaknemo samo vsako drugo osnovno črto, dobimo na ta način referenčno razdaljo med črtami, s katero lahko premik osnovne stegočrte določimo zelo zanesljivo.

Podobno lahko premikamo posamezne besede za majhno vrednost δ [77]. Bite skrite informacije zaznamo v treh korakih. Najprej določimo škatlo

vsake besede, ki jo dobimo iz profila pikslov v posamezni vrstici. Vsaki škatli določimo sredino. Določene škatle nato enakomerno razmaknemo in ugotovimo, za koliko so se premaknili centri posameznih škatel, iz česar potem izluščimo vrednost bitov.

2.2.7.2 Metode s stegopresledki

Tudi razmik med besedami oziroma stegopresledke (angl. white steg) lahko uporabimo za skrivanje informacije. Zapisane bite pa odkrijemo zelo preprosto, če le sumimo, da sporočilo vsebuje tudi skrito informacijo. Obstaja več idej [78]:

- Za vsakim terminalnim simbolom (pika v stavku, podpičje v programski kodi) vrinemo enega ali dva presledka, ki predstavljata bit 0 oziroma bit 1.
- Na koncu vsake vrstice dodamo en ali dva presledka.
- Na koncu vsake besede vrinemo en ali dva presledka.

2.2.7.3 Sintaktične metode

Tudi pri sintaktičnih metodah obstaja več možnosti [78]:

- Za vstavljanje skrite informacije lahko uporabimo ločila. Na primer, v angleškem jeziku sta obe frazi »bread, butter, and milk« in »bread, butter and milk« pravilni glede zapisa vejice. Če jo postavimo, pomeni prikrit bit 1, sicer pa bit 0. V slovenskih besedilih je zaradi strogih pravil o postavljanju ločil ta tehnika manj uporabna.
- Tudi nadzorovana uporaba kratic (na primer: »to je« ali »tj.«, ali »tiskalnik IBM« ali »IBM-tiskalnik«) nam omogoča vnos bitov skrite informacije.
- Bit informacije lahko zakrijemo tudi v oblikovanje stavkov. Na primer: »Še pred zoro bom oddal nalogo.« ali »Nalogo bom oddal še pred zoro.«, kjer lahko ugotavljamo položaj glagola v stavku.

2.2.7.4 Semantične metode

Bit skrite informacije vnesemo z uporabo sopomenk [78], na primer: ideja/zamisel, predor/tunel, hlebec/štruca, sadež/plod, kolo/bicikel. Za kodiranje/dekodiranje pripravimo preslikovalno tabelo. Na podlagi tega lahko v poved »Skozi tunel se peljem s kolesom po štruco kruha.« skrijemo informacijo 010.

2.2.7.5 Kodiranje značilnosti

V črkah pisave je možno izrabiti množico značilnosti (angl. feature coding) za zapis stegobita. Na primer, pikice nad črkama i in j so lahko malenkost dvignjene ali premaknjene, dolžino črtic pri črkah f in t lahko spreminjamo, razmik višine trebuščka pri črkah b in d je lahko različen.

2.2.8 Urejanje teksta

Obstaja veliko znanih algoritmov urejanja od hitrega urejanja in urejanja s kopico do urejanja z zlivanjem, številnega urejanja in korenskega urejanja [26]. Obstajajo pa namenske rešitve, ki so učinkovite, a iz različnih razlogov manj znane. ABC sort (Allen Beechick Character) je takšen primer [79], za katerega je bil leta 1993 podeljen ameriški patent [80].

Za urejanje teksta oziroma besedilnih nizov je koristno, da je algoritem urejanja stabilen, saj na takšen način lahko izvedemo zgolj delno urejanje po nekaj prvih znakih in pri tem v primeru enakih podnizov teksta ohranimo vrstni red pred urejanjem. Pri urejanju nizov moramo še določiti, ali želimo urejanje po abecednem vrstnem redu, torej da niz »a12« postavimo pred »a2« in ali želimo velike črke postaviti pred manjše. V zadnjem primeru govorimo o **ASCIIbetičnem** vrstnem redu urejanja. Če želimo niz »a2« postaviti pred »a12«, govorimo o **naravnem vrstnem redu urejanja**. V tem primeru moramo v tekstu razpoznati števila in jih obravnavati ločeno. Takšen način najlažje implementiramo z uporabo enega izmed primerjalnih urejanj (angl. comparison sort), kot je na primer hitro urejanje.

2.2.8.1 Algoritem TimSort

Pri urejanju podatkov pogosto naletimo na že delno urejena zaporedja, upoštevati pa moramo tudi značilnosti strojne opreme, kot je predpomnilnik na centralni procesni enoti. Eden izmed algoritmov, ki se zelo dobro izkaže na realnih podatkih, je TimSort. TimSort je izdelal Tim Peters leta 2002 [81, 82, 83]. Čeprav je v literaturi manj znan, je bil vključen v programski jezik Python od različice 2.3 do 3.10, uporabljajo pa ga tudi Java od različice 7 naprej, platforma Android, GNU Octave in Google Chrome.

TimSort je stabilen hibridni algoritem, ki združuje urejanje z zlivanjem in urejanje z vstavljanjem, kot je to predlagal že McIlroy [84]. Urejanje z vstavljanjem je namreč zelo učinkovito za polja, ki so dovolj majhna, to je, katerih velikost je manjša od *size* (*size* določimo eksperimentalno; izkaže se, da je ugoden $size \in [32, 64]$). Urejanje TimSort pa nadgradi to opisano idejo, in sicer zakaj bi urejali nekaj, kar je morebiti že urejeno. Zato vhodnega zaporedja ne razdeli na vnaprej določene dele, temveč v vhodnem

zaporedju poišče vse naraščajoče in padajoče dele. Podobno idejo najdemo tudi pri drugih algoritmih urejanja, kot je na primer algoritem Podgorelca in Klajnška [85].

Čeprav obstaja več različic algoritma TimSort, lahko osnovno idejo algoritma povzamemo v naslednjih korakih:

1. Celotno zaporedje razdelimo na urejena krajša podzaporedja, ki so ali naraščajoča ali padajoča ali neurejena.
2. Padajoča podzaporedja obrnemo, da postanejo naraščajoča.
3. Neurejena podzaporedja uredimo z algoritmom urejanja z vstavljanjem.
4. Zlijemo približno enako velika podzaporedja. Za uravnoteženje velikosti podzaporedij za zlivanje uporabimo sklad reda 4.

Poglejmo si delovanje algoritma na primeru polja R:

```
i: 0,  1,  2,  3,  4,  5,  6,  7,  8,  9
R: 1,  7,  6,  5,  2,  3,  4,  10, 8,  9
```

Zaporedje razdelimo na krajša urejena podzaporedja. Naj bo najkrajši urejen del dolžine 2. V primeru dolžine 3 in več bi kratke neurejene dele uredili z urejanjem z vstavljanjem. Dobimo:

```
i :  0,  1
R1: (1,  7)          (i)

i :  2,  3,  4
R2: (6,  5,  2)      (d)

i :  5,  6,  7,
R3: (3,  4,  10)     (d)

i :  8,  9
R4:  8,  9           (i)
```

kjer (i) označuje naraščajoče, (d) pa padajoče zaporedje. Ko padajoča zaporedja obrnemo, dobimo:

```
i :  0,  1
R1: (1,  7)

i :  2,  3,  4
```

```
R2: (2,  5,  6)

i :  5,  6,  7,
R3: (3,  4,  10)

i :  8,  9
R4:  8,  9
```

Dobljena podzaporedja zatem začnemo zlivati. Urejanje z zlivanjem je najučinkovitejše, če sta dela, ki ju zlivamo, čim bolj enakih dolžin. Algoritem TimSort za to uporabi **sklada reda 4** [83]. Sklad z redom $k \geq 2$ pri operaciji vključuje k vnosov z vrha sklada. Strategija zlivanja s skladom stopnje $k = 4$ pomeni, da bodo v zlivanju sodelovala štiri urejena podzaporedja. Predpostavimo, da so na vrhu sklada delno urejena podzaporedja W, Z, Y, X (X je na vrhu). Potem zlivamo po naslednjih pravilih [83]:

- če $|X| \geq |Z|$, potem Zlij(Y, Z),
- če $|X| \geq |Y|$, potem Zlij(X, Y),
- če $|X| + |Y| \geq |Z|$, potem Zlij(X, Y),
- če $|Y| + |Z| \geq |W|$, potem Zlij(X, Y).

TimSort zliva tako, da na sklad vpisuje indekse delno urejenih delov polja, ki smo jih dobili v prvem delu algoritma. Zatem ob vsakem vstavljanju podzaporedij na sklad preveri, katera podzaporedja lahko zlije. Če nadaljujemo z našim primerom, najprej na vrh sklada vstavimo prvi dve podzaporedji. Tako dobimo:

```
X  |X|=|R2|=3     (2,  5,  6)
Y  |Y|=|R1|=2     (1,  7)
```

Ker je izpolnjen drugi pogoj, zlijemo X in Y. Dobimo:

```
 X  |X|=5     (1,  2,  5,  6,  7)
```

Ker nobeden izmed zgornjih pogojev ni izpolnjen, nadaljujemo z vstavljanjem preostalih vhodnih podzaporedij na vrh sklada. Dobimo:

```
 X  |X|=|R3|=3     (3,  4, 10)
 Y  |Y|=5          (1,  2,  5,  6,  7)
```

Ker nobeden izmed zgornjih pogojev ni izpolnjen, nadaljujemo z vstavljanjem naslednjega vhodnega podzaporedja na vrh sklada. Dobimo:

```
X  |X|=|R4|=2     (8,  9)
Y  |Y|=|R3|=3     (3,  4, 10)
Z  |Z|=5          (1,  2,  5,  6,  7)
```

Ker je izpolnjen tretji pogoj, zlijemo X in Y. Dobimo:

```
X  |X|=5      (3,  4,  8,  9, 10)
Y  |Y|=5      (1,  2,  5,  6,  7)
```

Ker je izpolnjen prvi pogoj oziroma ker vhodnih podzaporedij ni več, zlijemo X in Y. Dobimo končno urejeno zaporedje:

```
X  |X|=10     (1,  2,  3,  4,  5,  6,  7,  8,  9, 10)
```

2.3 Seznam nalog

1. Zakodirajte znak Ū (koda UCS U+016A) z UTF-8 in UTF-32. Preverite z uporabo urejevalnika teksta, ali imate pravilno rešitev.

2. V urejevalniku teksta shranite daljšo besedilo s šumniki v formatih UTF-32, UTF-16 in UTF-8. Preverite razliko v velikosti datotek. V katerih primerih je možno iz števila znakov oceniti velikost datotek?

3. Z aplikacijo za pregledovanje kodnega nabora znakov Unicode poiščite prvo mesto neujemanja s kodno tabelo US-ASCII.

4. Na svojem operacijskem sistemu preverite videz znaka s kodo UCS U+1F63C. Najpogostejši način pretvorbe kode v znak v pisarniških paketih je uporaba kombinacije tipk ALT in X nad izbrano kodo.

5. V nastavitvah operacijskega sistema zamenjajte tipkovnico na angleški jezik in preverite, v katere znake se preslikajo šumniki na slovenski tipkovnici. Ugotovite še, kje na slovenski tipkovnici so preslikani znaki.

6. V pisarniškem paketu zapišite besedilo in ga shranite v datotečni format DOCX ali ODT. Shranjeni datoteki spremenite končnico na ZIP in preverite vsebino arhiva. Ali lahko v arhivu najdete zapisan tekst?

7. Niz *Dobro jutro.* skrijte v vašo programsko kodo z metodo s stegopresledki z vsemi tremi metodami (glej podpoglavje 2.2.7.2). Koliko vrstic programske kode potrebujete v posameznem primeru, če je niz zapisan z znaki ASCII? Implementirajte tudi dešifrirnik.

8. Izdelajte slovar sopomenk/sinonimov (pomagate si lahko s slovarjem sopomenk sodobne slovenščine [86]), nato pa v dokumentu *Navodila za pisanje zaključnih del na študijskih programih prve in druge stopnje UM FERI* skrijte izbrano sporočilo. Napišite tudi dešifrirnik.

9. Uredite zaporedje [4, 5, 7, 2, 13, 23, 1, 35, 33, 24, 12, 8, 9, 6, 3] z algoritmom TimSort. Najkrajši urejen del naj bo dolžine 3.

10. Implementirajte program urejanja TimSort. Preverite najboljši in najslabši primer ter implementacijo primerjajte z algoritmom hitrega urejanja in urejanja s kopico.

Poglavje 3

Slike

Vid je človekov najpomembnejši sprejemnik zunanjih dražljajev, ki možgane oskrbi z informacijami o dogajanju v okolju. Verodostojen prikaz vidne informacije je ključen v računalniški večpredstavnosti. Osnovni gradnik vidne predstavitve podatkov so mirujoče slike. Te delimo glede na način predstavitve v dve skupini:

1. **Rastrska slika** (tudi **rastrska grafika**) je računalniška predstavitev vidne informacije nekega časovnega trenutka (praviloma) realne scene. Za zajem rastrske slike uporabimo digitalni fotoaparat ali jo pridobimo iz prebirnih naprav (angl. scanners).

2. Risbam, ki jih narišemo z risalnimi programi (na primer AutoCAD, CorelDraw, InkScape), pravimo **vektorska grafika**. Vektorska grafika temelji na funkcijskem opisu uporabljenih gradnikov (črte, stožernice, Bézierove krivulje, Hermitove krivulje, krivulje B-zlepkov, krivulje NURBS in druge) [87]. Če zanemarimo končno aritmetiko digitalnega računskega stroja, je vektorska grafika analogna. Ker tudi vektorsko grafiko danes praviloma prikažemo na rastrskih izhodnih napravah, moramo njihovo interno funkcijsko predstavitev pred prikazom pretvoriti v rastrsko obliko/sliko z **rasterizacijo**. Obratnemu postopku (ko poskušamo iz rastrske slike pridobiti funkcijski opis nekaterih delov slike) pa pravimo **vektorizacija**. Postopki vektorizacije so težavnejši in manj dorečeni kot postopki rasterizacije.

Glede na vsebino slike delimo na fotografije, skice, pobarvanke in (inženirske) risbe. Fotografije imajo bogat nabor barv in barvnih prelivov, zato jih najlažje predstavimo z enakomerno mrežo enako velikih celic. Skice/pobarvanke/risbe pa so slike, ki jih ustvarja človek. Njihova skupna značilnost je majhno število barv, med barvami so ostri prehodi, ki

so pogosto poudarjeni z odebeljenim robom. Zaradi tega jih je smiselno predstaviti z vektorsko grafiko. Seveda pa lahko vektorsko in rastrsko grafiko med seboj tudi kombiniramo.

3.1 Zaznava in predstavitev barv

Pri računalniški predstavitvi slik moramo upoštevati značilnosti človeškega vida. Naše oko reagira le na dražljaje elektromagnetnega valovanja med 380 nm in 750 nm (vijolična (380–450 nm), modra (450–495 nm), zelena (495–570 nm), rumena (570–590 nm), oranžna (590–620 nm), rdeča (620–750 nm)). Ker je barvni spekter zvezen, je barv v vidnem delu spektra neskončno. Ocene, koliko barv zmore ločiti povprečno človeško oko, se močno razlikujejo, in sicer med 10^5 [88] do 10^7 barv [89]. Kakšno barvo vidimo, je odločitev možganov, na katero vplivajo poleg valovne dolžine svetlobnega žarka, ki zadene mrežnico očesa, tudi barva ozadja, jakost svetlobe in poznavanja okolja. Kot vemo, so na mrežnici očesa paličice in čepki [90]. Ko jih zadene energija elektromagnetnega valovanja vidne valovne dolžine, pride do kemične reakcije, ki se kot živčni signal prenese v možgane. Čepki zaznavajo barve, paličice pa jakost svetlobe. Paličic je v očesu okrog 92 milijonov, čepkov pa reda 5 milijonov [90].

Imamo tri vrste čepkov z največjo občutljivostjo v področju elektromagnetnega valovanja, ki ga zaznavamo kot modro, zeleno in rdečo barvo. Razmerje med jakostjo stimulacije teh treh vrst čepkov določa zaznavo poljubne barve. Po vzoru značilnosti človeškega vida deluje **barvni model RGB** (angl. red, green, blue), ki barvo na sliki predstavi s kombinacijo treh vrednosti oziroma **kanalov** rdeče, zelene in modre barve. Barve v posameznih kanalih predstavimo s številskimi vrednostmi in jih lahko zapišemo normirano z decimalnimi števili, torej med 0 in 1, kjer 0 predstavlja črno barvo, 1 pa največjo možno intenziteto posamezne barve. Če imajo vse tri komponente vrednost 1, dobimo belo barvo. Vrednosti barv pa lahko zapišemo tudi s celimi števili, na primer zelo pogosto z enim zlogom, torej med 0 in 255.

Ne glede na način predstavitev barv njihovo natančnost omejuje število bitov na posamezno vrednost. Pri barvnem modelu RGB zelo pogosto uporabimo 8 bitov na barvni kanal. To pomeni, da lahko predstavimo $2^{(3\times 8)}$ barv. Na prvi pogled je to visoka številka, vendar moramo upoštevati, da človeški vid ne zaznava barv linearno glede na intenziteto svetlobe. Vid je bolj občutljiv na spremembe temnejših odtenkov kot svetlejših. Zaradi tega se lahko zgodi, da pri prikazu prelivov z omejenim številom barv na temnejših področjih namesto gladkega prehoda med barvami vidimo večja območja z

enako barvo, torej **barvne proge** (angl. colour banding), med katerimi je jasno razvidna meja. Zaradi značilnosti človeškega vida na prehodu sorodnih pasov postane rob opaznejši, kot je v resnici, čemur pravimo tudi **Machov pojav** (angl. Mach effect) [91]. Da to težavo omilimo, manjše vrednosti predstavimo bolj natančno kot večje. To dosežemo s **korekcijo gama** (angl. gamma correction) [92] tako, da normirane vhodne vrednosti barv c_{vhod} nad vsakim barvnim kanalom preslikamo:

$$c_{\text{izhod}} = c_{\text{vhod}}^{\gamma}, \tag{3.1}$$

kjer sta c_{izhod} in c_{vhod} vhodni in izhodni barvi normirani med 0 in 1. Ko sliko zajamemo in preden barve v sliki shranimo v datoteko z omejenim številom bitov, jih preslikamo z $\gamma < 1$. Preden sliko iz datoteke prikažemo, barve pretvorimo z $\gamma > 1$.

Poglejmo, kako deluje korekcija gama pri $\gamma = 1/2{,}2$. Imejmo normalizirane vrednosti barv 0,1 in 0,2. Po transformaciji dobimo 0,351 in 0,481, kar nam da večjo razliko 0,13 in s tem tudi večjo natančnost. Če imamo večje vrednosti z enako razliko, na primer 0,8 in 0,9, dobimo 0,904 in 0,953, torej občutno manjšo razliko 0,049.

Prikazovalniki in kamere/fotoaparati delujejo po podobnem principu kot človeški vid. Zaradi tega je bilo treba že pri analognih prikazovalnikih uskladiti barve med različnimi napravami. Za analogne prikazovalnike se je zelo pogosto uporabljala korekcija gama z $\gamma = 2{,}2$ [93].

Pri obravnavi barv na slikah se moramo torej zavedati, da se preslikava med intenziteto svetlobe in dejanskimi vrednostmi barvnega modela RGB izvaja na več mestih, za kar se zelo pogosto uporablja korekcija gama. Preslikava se najprej izvede ob zajemu slike v fotoaparatu in nato pri shranjevanju slike. Ko sliko odpremo, jo moramo preslikati za prikaz na prikazovalniku.

Barvni model je matematična predstavitev barv s številskimi vrednostmi, ki pa še ne določa natančnega odtenka oziroma videza posameznih barv. Barve, zajete s fotoaparatom in potem prikazane na različnih računalniških prikazovalnikih ali natisnjene na tiskalnikih, pogosto niso videti enako, saj naprave različno prikažejo barve na podlagi vrednosti barvnega modela RGB in imajo tudi različen **barvni razpon** (angl. gamut). Ko govorimo o barvnem modelu RGB, pogosto mislimo na standardiziran barvni prostor **sRGB** (IEC 61966-2-1:1999), ki določa videz barv na izhodnih napravah, torej določa preslikavo številčne vrednosti v končno barvo. Večina naprav privzeto uporablja barvni prostor sRGB. Barvni prostor sRGB s preslikavo aproksimira vrednost γ iz enačbe 3.1 na približno 2,2.

Slabost barvnega prostora sRGB je, da barvni razpon ni dovolj širok za predstavitev vseh možnih barv, kar potrebujemo v profesionalnih aplikacijah.

Zaradi tega barvno predstavitev RGB pogosto pretvorimo v drug barvni model (CMYK, CMY, CIE, HSL, HSV, YUV, YCbCr) ali prostor. Podrobnosti barvnih modelov in prostorov sicer spadajo na področje računalniške grafike [94] ter interakcije med človekom in računalnikom, zaradi tega jih v tem učbeniku ne obravnavamo. V okviru obdelave podatkov večpredstavnosti je smiselno omeniti poleg barvnega modela RGB še barvna modela YUV in YCrCr, pri katerih Y predstavlja **svetlost** (angl. luminance), preostali komponenti pa določata **barvitost** (angl. chrominance). Njun pomen bomo spoznali v nadaljevanju.

Veliko novejših naprav omogoča večji barvni razpon od sRGB, zato potrebujemo ustrezno neposredno pretvorbo med barvnimi prostori naprav, tako da lahko uskladimo barve med vhodnimi in izhodnimi napravami (na primer s fotoaparata na tiskalnik). Informacije za pretvorbo med barvnimi prostori lahko shranimo v obliki **barvnih profilov** ICC (angl. International Color Consortium profile). Profili ICC določajo barvni prostor naprave (vhodne ali izhodne) na podlagi **povezovalnega barvnega prostora** CIEXYZ ali CIELAB, ki sta prilagojena človeškemu vidu. Profili ICC so standardizirani (ISO 15076-1 [95]) in pogosto shranjeni v datotekah ICC/CIM ali v slikovnih datotekah. Profili ICC pa s preslikavo med barvnimi prostori zajemajo tudi korekcijo gama.

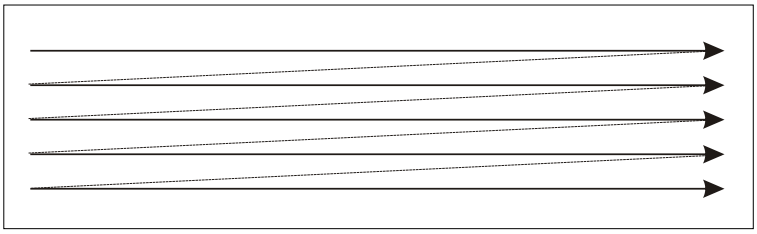
3.2 Rastrske slike

V računalniku lahko predstavimo le delček barvne informacije realnega sveta, kar dosežemo s **kvantizacijo** in z **vzorčenjem**. Oba pojma sta sicer povezana s teorijo signalov, kjer vzorčenje pove, s kakšno hitrostjo odčitavamo vrednost analognega signala, kvantizacija pa, s koliko biti odčitano vrednost tudi predstavimo. Pri računalniški predstavitvi slik dobimo končno mnogo diskretnih delčkov slike – **pikslov** (angl. PIcture X ELement). Piksli so najpogosteje kvadratne/pravokotne oblike (možne so tudi druge oblike, na primer ovalni piksli, ki so bili karakteristični pri katodnih ceveh, ali piksli v obliki pravilnih šestkotnikov) ter urejeni v matriko, ki ji pravimo **raster**. Številu pikslov v rastru pravimo **ločljivost** (angl. resolution). Če se od rastra dovolj odmaknemo, barve posameznega piksla ne zaznamo več, raster pa dojemamo kot enovito sliko. Število bitov, ki jih potrebujemo za opis barve posameznega piksla, je različno glede na uporabo; od enega bita za črno-belo sliko do danes običajnih 24 bitov pa vse do 48 bitov. Če je uporabljen še kanal alfa, zanj običajno porabimo dodatnih 8 bitov.

Rastrske slike najpogosteje zajemamo z digitalnimi fotoaparati, ki vsebujejo slikovne senzorje CCD (angl. Charge–Coupled Device) ali CMOS (angl.

Complementary metal–oxide–semiconductor) [96, 97]. Ti senzorji dajejo napetostni odziv glede na prevladujočo dolžino elektromagnetnega valovanja. Napetosti na senzorjih zajamemo v delčku časa, napetosti kvantiziramo, dobljene vrednosti pa ustrezajo barvi posameznih pikslov. Naslednji vir rastrskih slik so prebirne/skenirne naprave, ki slikovne senzorje pomikajo po dokumentu. Zadnji večji vir, ki pa ga je vedno manj, je digitalizacija analognega videosignala [2].

Rastrske slike prikazujemo na rastrskih izhodnih napravah, izmed katerih so danes najpogostejši **prikazovalniki** (angl. displays), ki so se, hkrati s celotnim računalniškim sistemom, intenzivno razvijali [2]. V preteklosti so se najpogosteje uporabljali prikazovalniki s katodno cevjo (angl. Cathode–ray tube, CRT), te pa so danes povsem nadomestile druge tehnologije: prikazovalniki LCD (angl. Liquid–Crystal Display), OLED (angl. Organic Light–Emitting Diode), AMOLED (angl. Active–Matrix Organic Light-Emitting Diode) in prikazovalniki s kvantnimi pikami QD/QD–LED (angl. Quantum Dot). Za podporo interaktivnosti je barve pikslov treba osveževati v za človeka realnem času. Vrstnemu redu osveževanju pikslov pravimo **rastrsko prebiranje** (angl. scanning) in ga shematsko vidimo na sliki 3.1. Čeprav moderni prikazovalniki, v nasprotju s prikazovalniki CRT, omogočajo neposreden dostop do pikslov, rastrsko prebiranje še danes uporabljamo pri shranjevanju slik v datoteke in pri prenosu po omrežju. Zgodovinske razloge za takšen način obiskovanja pikslov na sliki bomo razložili v nadaljevanju.



Slika 3.1: Rastrsko prebiranje (po vzoru glede na [2])

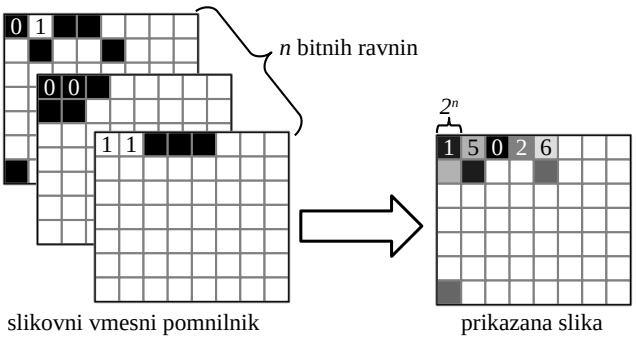
Pomen ločljivosti pri rastrskih slikah in vhodno-izhodnih napravah se nekoliko razlikuje. Pri rastrskih slikah praviloma pomeni število pikslov v smereh x in y. Na primer, ločljivost 1280×768 določa število pikslov v širini in višini slike. Pri določanju natančnosti vhodno-izhodne naprave pa ločljivost podajamo s številom pikslov, točk ali vzorcev na dolžinsko enoto (na primer cm ali m). **Število pikslov na dolžinsko enoto** lahko uporabljamo pri prikazovalnikih ali skenerjih (pri slednjih pogosto podamo tudi število vzorcev (angl. samples) na dolžinsko enoto). Število pikslov na enoto lahko ob skeniranju shranimo zraven rastrske slike, kar uporabimo pri

tisku, da dobimo natisnjeno sliko enake velikosti, kot smo jo zajeli ne glede na ločljivost naprav. **Število točk na dolžinsko enoto** pa najdemo pri tiskalnikih, saj pri tisku tiskamo pike in ne pikslov.

Zaradi zgodovinskih razlogov so se pri merjenju ločljivosti uveljavile imperialne merske enote oziroma običajne enote Združenih držav Amerike (angl. United States customary units), torej število **pikslov na palec** (angl. pixels per inch, PPI), **točk na palec** (angl. dots per inch, DPI) in **vzorcev na palec** (angl. samples per inch, SPI), ki jih naprava lahko prikaže, natisne ali zajame. Seveda so v uporabi tudi enote mednarodnega sistema SI, kot so število **točk na centimeter** (angl. dots per centimeter, dpcm), število **pikslov na meter** (angl. pixels per meter, PPM) in razmik med točkami, a v praksi praviloma srečamo le imperialne enote.

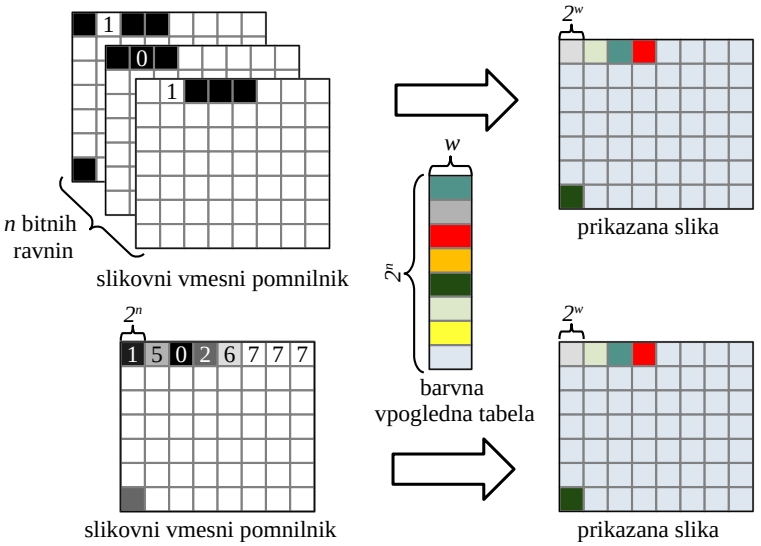
3.2.1 Predstavitev rastrskih slik

Podatke o barvah pikslov hranimo v **slikovnem pomnilniku** (angl. video memory, framebuffer). Vrednosti pikslov v slikovnem pomnilniku najpogosteje zapišemo v vrstnem redu rastrskega prebiranja. Drugi način predstavitve uporablja slikovni pomnilnik, ki je logično razdeljen po teži bitov v **bitne ravnine** (angl. bit planes). Slika 3.2 shematično prikazuje slikovni pomnilnik, ki omogoča predstavitev 2^n barvnih odtenkov.



Slika 3.2: Slikovni vmesni pomnilnik z n bitnimi ravninami

Pogosto se izkaže, da vseh barvnih odtenkov ne potrebujemo, zato lahko zmanjšamo število bitov na piksel. V tem primeru uporabimo **barvno paleto** oziroma **barvno vpogledno tabelo** (angl. look-up table, LUT), shematično predstavljeno na sliki 3.3. Vrednost iz slikovnega pomnilnika definira indeks v barvni vpogledni tabeli, ki ima 2^n vhodov. Širina barvne vpogledne tabele je w bitov ($w > n$). Na ta način lahko na sliki predstavimo 2^w odtenkov, vendar iz nabora 2^n vnosov.



Slika 3.3: Izbira w barvnih odtenkov iz barvne vpogledne tabele, pri čemer je lahko slikovni pomnilnik predstavljen z n bitnimi ravninami ali zaporedno z n-bitno globino.

Vrednosti pikslov oziroma barve pri barvnih slikah razdelimo na posamezne barvne komponente. Najpogosteje je število bitnih ravnin oziroma bitov za posamezno komponento 8 (angl. bits per channel, BPC), pri čemer uporabljamo barvni model **RGB** (angl. red, green, blue), ki barvo piksla predstavi s kombinacijo treh kanalov, torej rdeče, zelene in modre barve. V tem primeru lahko s kanali RGB prikažemo 2^{24} različnih barv oziroma uporabimo 24 **bitov na piksel** (angl. bits per pixel, bpp), čemur pravimo **popoln prikaz barv** (angl. true colours). 256 odtenkov ene barve je lahko premalo, zato včasih popoln prikaz barv razširimo s slikovnimi tabelami za posamezno barvo. Pri kakovostnejših slikah danes uporabljamo več bpp; od 32 pa vse do 48 bitov. V preteklosti se je pogosto uporabljala tudi 16-bitna globina (angl. high colours), ki omogoča 65.536 barv, pri čemer uporabimo praviloma 5 ali 6 bitov na barvni kanal RGB.

Naslednji pogost barvni model je še **RGBA**, kjer dodamo kanal alfa oziroma A (angl. alpha) za prosojnost (32-bitna globina pri 8 bitih na kanal). Kanal alfa nadzira prekrivanje dveh rastrskih slik oziroma prosojnost pikslov rastrske slike, ki prekrivajo piksle druge rastrske slike. Vrednosti kanala alfa preslikamo v interval $[0, 1]$ in s tem določimo delež vidnosti piksla p_A slike A, ki prekriva piksel p_B slike B. Recimo, če v barvnem modelu RGBA vrednosti vseh kanalov preslikamo v interval $[0, 1]$, bi zapis $(0, 0{,}5, 0, 0{,}3)$

po preslikavi pomenil, da ima odtenek zelene barve 70-odstotno prosojnost. Najpreprostejši način za določitev nove barve c, torej slike C, dobimo z naslednjo linearno enačbo:

$$c(p_C) = \alpha(p_A)c(p_A) + (1 - \alpha\,(p_A))\,c(p_B) \tag{3.2}$$

Če je $\alpha(p_C) = 1$, je piksel p_A neprozoren in vidimo njegovo barvo $c(p_A)$. Če je $\alpha(p_C) = 0$, je p_A popolnoma prozoren, zato vidimo barvo $c(p_B)$ piksla p_B. Če je $\alpha = 0{,}5$, dobimo barvo, ki je na aritmetični sredini med barvama $c(p_A)$ in $c(p_B)$. Poleg preproste linearne odvisnosti iz enačbe 3.2 so za doseganje različnih učinkov predlagali tudi druge odvisnosti [98, 99].

3.2.2 Shranjevanje rastrskih slik

Rastrske slike zasedejo veliko prostora. Slika v popolnem prikazu barv velikosti 1024×1024 zasede 3.145.728 zlogov. Že od leta 2012 lahko digitalni fotoaparati v mobilnih telefonih zajamejo več kot 40×10^6 pikslov [100], kar zahteva 120 M zlogov veliko datoteko, če vsako barvno komponento zapišemo s tremi zlogi. Zaradi tega je stiskanje slik bistvenega pomena. V grobem ga delimo na:

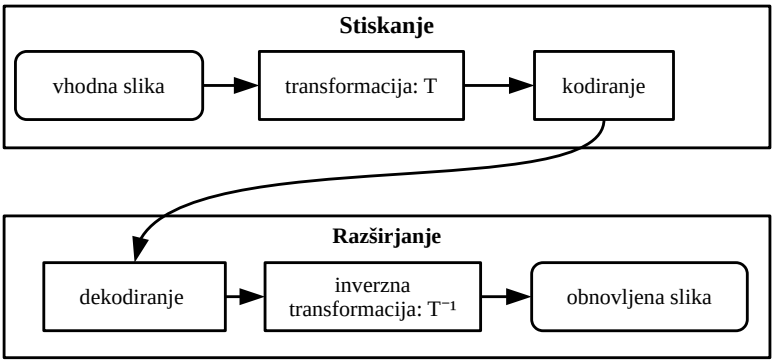
- **Brezizgubno stiskanje** (angl. lossless), kjer izguba kakršnihkoli informacij ni dovoljena. Brezizgubno stiskanje je bistveno tam, kjer rastrske slike hranijo rezultate meritev, na primer na področju medicine in daljinskega zaznavanja.

- **Izgubno stiskanje** (angl. lossy) zavrže del informacije o barvah v sliki, pri čemer želimo, da je izguba čim manj opazna. Z izgubnim stiskanjem dosežemo bistveno višjo stopnjo stiskanja kot pri brezizgubnem, pri čemer pa se moramo zavedati, da po razširjanju stisnjene slike dobimo le **aproksimacijo** originalne slike.

- **Skoraj brezizgubno** (angl. near losless) bi lahko prištevali k izgubnemu stiskanju, pri čemer pa obstaja pomembna razlika. Pri skoraj brezizgubnem stiskanju lahko vnaprej določimo zgornjo mejo izgub posameznih pikslov, česar pri izgubnem stiskanju ne moremo nadzirati.

Podobno kot pri tekstu moramo tudi pri stiskanju slik odpraviti redundanco v podatkih. Pri tem pa se srečujemo z različnimi tipi redundance [101, 102]:

- **Prostorska redundanca** je med piksli in deli slike, saj so sosednji piksli pogosto podobnih barv ali v ponavljajočih se vzorcih (na primer okna ali valovi).



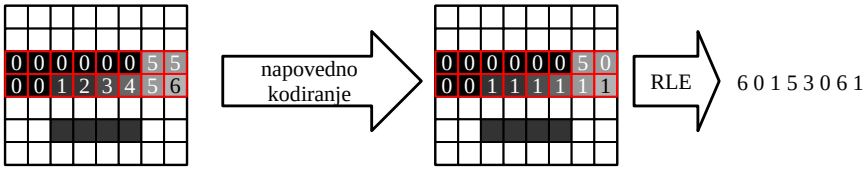
Slika 3.4: Tipičen cevovod brezizgubnega stiskanja slik

- **Spektralna redundanca** je korelacija med vrednostmi v barvnih kanalih, na primer v primeru svetlejših delov slike pričakujemo višje vrednosti vseh treh barvnih kanalov RGB.
- **Statistična redundanca** nam omogoča, da pogostejše vrednosti pikslov zapišemo z manjšim številom bitov.
- **Psihovizualna redundanca** (angl. psychovisual) je temelj izgubnega stiskanja. Izgubo informacij na bolj dinamičnih področjih namreč težje razpoznamo kot na bolj homogenih delih slike. Na podlagi števila čepkov in paličic v očesu lahko sklepamo, da slabše razpoznamo spremembo barv kot spremembe svetlosti barve. To nam omogoča odstraniti več informacij, kot jih lahko povrnemo v upanju, da izgube uporabnik sploh ne bo zaznal.

3.2.2.1 Brezizgubno stiskanje

Brezizgubno stiskanje (angl. lossless compression) najpogosteje izvedemo v dveh korakih, kot je prikazano na sliki 3.4. Najprej sliko transformiramo v vmesno obliko, s čimer poskusimo zmanjšati informacijsko entropijo podatkov. Pri tem najpogosteje odpravljamo prostorsko redundanco z na primer izračunom razlike med sosednjimi piksli in/ali brezizgubno transformacijo v frekvenčni prostor. V naslednjem koraku odpravimo statistično redundanco s statističnimi kodirniki (na primer Huffmanovo ali aritmetično kodiranje), s slovarjem (na primer LZW [26]) ali s kakšnim drugim pristopom (na primer RLE). Razširjanje stisnjene slike (dekompresija) je popolnoma obratno.

Glavni namen transformacije je odprava redundance v sliki. Najpogosteje odpravljamo redundanco med sosednjimi piksli tako, da na primer napovemo



Slika 3.5: Primer napovednega stiskanja, kjer stiskamo označen del sivinske slike, vrednosti pikslov so zapisane znotraj vsakega piksla in za napoved uporabimo vrednost levega soseda.

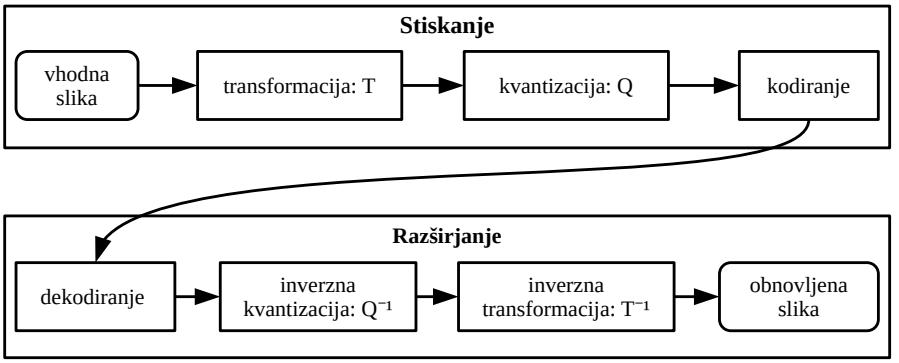
spodnjo vrstico slike glede na predhodno vrstico. V tem primeru govorimo o napovednem kodiranju (angl. predictive coding), saj predvidevamo vrednost piksla na podlagi sosednjih pikslov. Poglejmo si preprost primer na sliki 3.5. Originalno vrednost piksla p zapišemo kot $s(p)$. Nato napovemo vrednost vsakega piksla p na podlagi njegovega levega soseda. Za prvi piksel predpostavimo, da je vrednost njegovega levega soseda 0. Napoved piksla p zapišimo kot $\hat{s}(p)$ (angl. prediction), nato pa izračunamo napako napovedi:

$$e(p) = s(p) - \hat{s}(p). \tag{3.3}$$

Pri tem si želimo, da bi bila večina napak blizu vrednosti 0 ali da so si med seboj podobne. Kot lahko opazimo, smo že z zelo preprosto napovedjo zmanjšali število različnih vrednosti, kar lahko učinkoviteje stisnemo že z RLE (angl. Run–Length Encoding). Pri brezizgubnem stiskanju slik lahko uporabimo tudi druge transformacije ali zaporedje več transformacij. Na primer namesto obiska pikslov v prebirnem vrstnem redu lahko to storimo z eno izmed krivulj polnjenja prostora, z Burrows-Wheelerjevo transformacijo lahko preuredimo vrednosti pikslov tako, da zgručimo piksle z enakimi ali podobnimi vrednostmi, za tem pa s transformacijo premik naprej ali s transformacijo inverznih frekvenc zmanjšamo njihovo informacijsko entropijo [26].

3.2.2.2 Izgubno stiskanje

Človeška zaznava informacije na slikah se spreminja s časom opazovanja, razdaljo gledanja in psihofizičnimi lastnostmi človeka. Prav zaradi tega lahko v mnogih primerih s slike odstranimo del informacije, ne da bi občutno vplivali na njen videz. Seveda se postavi vprašanje, kako ugotoviti, kateri del informacijske vsebine je za človeški vidni sistem nezaznaven. Za to nalogo uporabimo transformacije, ki opravijo analizo frekvenčne vsebine slike. Najpogostejši transformaciji, uporabljeni pri izgubnem stiskanju slik, sta **Fourierjeva** in **valčna transformacija** ter njune izpeljanke, na primer



Slika 3.6: Tipičen cevovod izgubnega stiskanja slik

diskretna kosinusna transformacija (angl. Discrete Cosine Transform, DCT) [103]. V naslednjem koraku s **kvantizacijo** (angl. quantisation) odstranimo višje harmonske frekvence glede na uporabniško podan kvantizacijski parameter. Višje harmonske frekvence namreč ustrezajo tistim detajlom v sliki, ki jih človeški vidni sistem pogosto sploh ne zazna. Opisan koncept vidimo na sliki 3.6. Razširjanje stisnjene slike je obraten postopek, pri čemer pa vrednosti vseh pikslov obnovljene slike niso enake pikslom vhodne slike. Obstaja veliko izgubnih metod stiskanja rastrskih slik [7], nekaj izmed njih si bomo ogledali v nadaljevanju.

3.2.3 Formati za shranjevanje rastrskih slik

Za zapis rastrskih slik je bilo razvitih ogromno formatov, nekateri izmed njih so mednarodni standardi (JPEG, JPEG-LS, JPEG2000, JBIG, HEIF, JPEG XL), drugi so široko razširjene specifikacije, tretji pa so zelo redki in se uporabljajo v zelo omejenih industrijskih okoljih. Poleg informacij o barvah pikslov formati hranijo tudi metapodatke, kot na primer: ločljivost, barvna globina, barvni model (na primer 32 bitov in RGBA), barvni profil ICC in dodatne informacije (na primer uporabljen fotoaparat, avtor, ločljivost v številu točk na palec) ter način zapisa pikslov in barv (na primer uporabljen algoritem stiskanja). Najpogostejši formati so:

- za brezizgubno stiskanje: BMP, PNG, GIF, JPEG-LS,
- za izgubno stiskanje: JPEG, JPEG2000, WebP, AVIF, HEIF in
- za podporo urejanju: PSD (Adobe Photoshop Document), CPT (Corel Photo-Paint Document), PSP (PaintShop Pro Image File), XCF

(GIMP); slednji omogočajo hrambo dodatnih informacij, na primer slojev in zgodovine operacij nad slikami. Teh formatov ne bomo obravnavali.

3.2.3.1 BMP

Format BMP[1] (angl. bitmap image file) je razvil Microsoft v osemdesetih letih za operacijski sistem Microsoft Windows. Namenjen je brezizgubnemu shranjevanju slik, ni patentiran ter je v osnovni različici enostaven za branje in kompatibilen z orodji v operacijskih sistemih Microsoft Windows. Glavna težava formata je v njegovi prostorski potratnosti, saj ne uporablja stiskanja ali pa je to zelo preprosto. S časom je nastalo več različic formata, kar je treba upoštevati pri branju datotek BMP [104]. Način zapisa je vključen tudi v druge formate; na primer format WMF uporablja strukturo datoteke BMP različice 5 [105]. Datoteka BMP je sestavljena iz treh delov [106, 105, 104]:

- Glava BMP hrani metapodatke, ki vključujejo ločljivost slike, način stiskanja, način zapisa pikslov in v novejših različicah tudi opis predstavitve barv.
- Barvna vpogledna tabela ima toliko vnosov, kot je barv v sliki. Če imamo 24-bitno sliko, se barvna vpogledna tabela ne uporablja.
- Zaporedje bitov opisuje barve pikslov v sliki v vrstnem redu prebiranja (angl. scan-line order). Pri zapisu pikslov moramo upoštevati, da je dolžina vrstice v zlogih vedno večkratnik štirih zlogov (32 bitov). Če to glede na širino slike in število zlogov na piksel ne velja, moramo na koncu vsake vrstice izvesti zapolnjevanje z zlogi (angl. padding).

BMP zapisuje barve v barvnem modelu RGBA, bodisi kot indekse barv v barvni paleti bodisi neposredno. Zapis pikslov je lahko:

- 1-bitni (dvobarvna slika, indeksni način zapisa barv),
- 4-bitni (16-barvni, indeksni način zapisa barv),
- 8-bitni (256-barvni, indeksni način zapisa barv),
- 16-bitni (65.536 barv),
- 24-bitni ($2^{3\times8}$ barv), v tem primeru zloge R, G, B prepletamo,
- 32-bitni (2^{32} barv).

[1] Poznamo ga tudi kot format DIB (angl. device independent bitmap)

BMP je zelo prilagodljiv glede zapisa barv. Za vse štiri kanale barvnega modela RGBA lahko z **bitno masko** določimo, kateri biti piksla pripadajo kateremu kanalu RGBA. Za predstavitev kanalov RGB s po osmimi biti, torej RGB24, bi zapisali takšno masko:

- R: 11111111 00000000 00000000,
- G: 00000000 11111111 00000000,
- B: 00000000 00000000 11111111.

V tem primeru bi torej prvih osem bitov dodelili rdeči, naslednjih osem bitov zeleni, osem najmanj pomembnih bitov pa modri barvi.

Z bitno masko lahko določimo tudi na primer 16-bitni zapis pikslov, kjer vsakemu barvnemu kanalu RGBA dodelimo po štiri bite. Če prosojnosti ne potrebujemo, lahko za barvna kanala rdeče in modre barve uporabimo po 5 bitov, za zeleno barvo pa 6 bitov. Zelo pogosto se uporablja 32-bitni zapis, kjer za vsak barvni kanal RGBA uporabimo po 8 bitov.

BMP uporablja stiskanje RLE za 4- in 8-bitne slike. Kljub prostorski potratnosti je BMP v okolju Windows izjemno pogost in priljubljen industrijski standard. BMP je z novejšimi različicami dobil podporo tudi za barvne profile ICC [104], vendar je zapis barv še vedno v barvnem modelu RGB oziroma sRGB.

3.2.3.2 GIF

Format GIF (angl. Graphics Interchange Format) je razvilo podjetje CompuServe v osemdesetih letih prejšnjega stoletja za prikaz rastrskih slik na takrat razvijajočem se internetu [107, 108, 109]. Datotečni format podpira prosojnost, prepletanje in hrambo več slik v eni datoteki, kar omogoča izvedbo preprostih in kratkih animacij. Obstajata dve različici:

- GIF87a (1987) uporablja prepletanje, 256-barvno paleto in omogoča shranjevanje več sličic v eni datoteki.
- GIF89a (1989) poleg lastnosti predhodnika vključuje še prosojnost (zgolj binarno), določanje hitrosti prikazovanja zaporedja sličic in podporo za metapodatke.

Datoteka GIF je [109] sestavljena iz glave datoteke in seznama več slik:

- Na začetku datoteke so prvi zlogi »GIF87a« ali »GIF89a«, zatem so podane splošne informacije (velikost celotne slike oziroma navideznega platna in informacije za prosojnost) ter globalna barvna paleta.

- Nato sledi seznam slik. Vsaka slika vsebuje parametre za njen prikaz znotraj navideznega platna (dimenzija slike in njen položaj znotraj tega platna), lokalno barvno paleto, nato pa je zapisano telo slike oziroma zaporedje pikslov.

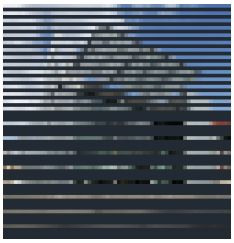
Slika, zapisana v GIF, lahko vsebuje do 256 barv iz **lokalne** ali **globalne barvne palete**. V barvni paleti so barve predstavljene z barvnim modelom RGB, kjer lahko vsaka komponenta zasede vrednosti med 0 in 255. Omejena velikost barvne palete omejuje GIF za kakovostno shranjevanje slik ali za njihovo uporabo v namiznem založništvu, a je večinoma povsem dovolj za prikaz slik na prikazovalniku (slika 3.7b). Pri tem je nujno, da za barvno

(a) (b) (c) (d)

Slika 3.7: Primerjava med (a) izvorno sliko (RGB24) in sliko, pri kateri uporabimo barvno paleto z (b) 256 naključnimi barvami, (c) 256 najbolj primernimi barvami ter (d) 256 najbolj primernimi barvami in uporabo stresanja (avtor izvorne slike: Miran Kambič).

paleto izberemo najbolj zastopane barve (slika 3.7). Da bi povečali iluzijo večjega števila barv, pogosto uporabljamo tudi tehniko **stresanja** (angl. dithering), kar pa ni lastnost GIF sama po sebi. Primer stresanja je prikazan na sliki 3.7d, pri čemer je najbolj opazna razlika v prelivu modre barve. Kot smo omenili, lahko GIF v eno datoteko shrani več slik za izvedbo preproste animacije, pri čemer se vsaka slika prikaže znotraj navideznega platna in uporablja svojo (lokalno) paleto.

Prepletanje (angl. interlacing, interleaving) daje občutek hitrejšega prikaza slike na prikazovalniku, kot se je ta dejansko zgodil. Ne smemo pozabiti, da je bil GIF zasnovan v zgodnjih časih interneta, ko so bile hitrosti prenosa podatkov zelo nizke. Primer prepletanja vidimo na sliki 3.8, kjer bi lahko moteče črne trakove prekrili z vrednostmi iz predhodnih vrstic. Prepletanje ni nič drugega kot preskakovanje vrstic iz prebirnega vrstnega reda in je bilo, kot bomo spoznali, najprej uporabljeno pri analogni televiziji. GIF uporablja naslednjo shemo prepletanja:



Slika 3.8: Primer prepletanja slike 3.7a iz datotečnega formata GIF, pri čemer lahko takoj razpoznamo objekt na sliki pri manj kot polovici prenesenih podatkov.

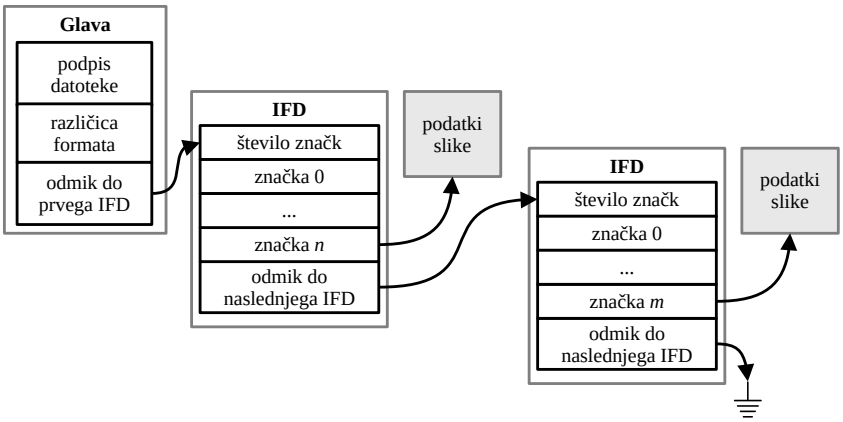
- najprej prikažemo piksle vsake osme vrstice, pri čemer začnemo s prvo, sledi deveta, sedemnajsta in tako naprej,
- sledijo piksli vmesnih vrstic 5, 13, 21 ...,
- zatem pošljemo preostale lihe vrstice (3, 7, 11, 15 ...),
- v zadnjem koraku pa prenesemo in prikažemo še preostale sode vrstice (2, 4, 6 ...).

GIF za zmanjševanje količine podatkov uporablja algoritem LZW. A ravno uporaba algoritma LZW se je izkazala kot ključna slabost formata GIF, saj ga je avtor Terry Welch zaščitil s patentom [110]. Zaradi tega se je spletna skupnost odločila razviti drug format, ki ne bi bil podvržen ne patentni zaščiti ne licenciranju; razvili so format PNG, ki ga bomo spoznali nekoliko pozneje.

3.2.3.3 TIFF

TIFF (angl. Tagged Image File Format) [111, 112, 113] je bil razvit že leta 1987. Je izjemno priljubljen in široko podprt format za shranjevanje rastrskih slik še posebej v programih namiznega založništva. TIFF je razvilo podjetje Aldus (program se je takrat imenoval PageMaker), ki ga je pozneje kupilo podjetje Adobe. Z verzijo TIFF 6 iz leta 1992 se je format stabiliziral. TIFF podpira:

- črno-bele slike,
- sivinske slike,
- slike s paleto (od eno- do osembitne palete),
- RGB,



Slika 3.9: Primer datotečne strukture formata TIFF z dvema vključenima slikama – IFD (angl. Image File Directory)

- YCbCr,
- CMYK,
- CIE.

Globina posameznega kanala je omejena na 16 bitov, kar omogoča shranjevanje izjemno kakovostnih slik. TIFF podpira stiskanje z LZW, PackBits in CCITT Fax group 3 & 4 ter tudi izgubno stiskanje. Velikost slike je omejena z velikostjo datoteke, ki je 4 G zlogov (z uporabo stiskanja lahko opišemo sliko do velikosti $2^{32} \times 2^{32}$ pikslov).

TIFF, kot že nakazuje ime, temelji na **značkah** (angl. tags), s katerimi opiše lastnosti rastrske slike, kar lahko vidimo na sliki 3.9. Datoteke TIFF se začnejo z glavo – IFH (angl. Image File Header). Format najprej določi tip datoteke in vrstni red zlogov v podatkih (angl. endianness). V glavi je nato zapisan odmik do podatkov o prvi sliki – IFD (angl. Image File Directory). V IFD je podano zaporedje značk, med katerimi ena izmed njih kaže na podatke pripadajoče slike. Pomembna značilnost TIFF je tudi, da vsak IFD kaže na naslednji IFD. Na takšen način format podpira praktično neomejeno število slik, saj je podatkovna struktura podobna seznamu.

TIFF določa osnovni nabor značk (angl. baseline TIFF), ki vsebujejo osnovne informacije, kot sta barvna globina in način stiskanja. V razširitvah oziroma z dodatnimi značkami je možno uporabiti tudi stiskanje z JPEG (izgubno stiskanje), z Deflate [114] in z LZW. Za shranjevanje slik daljinskega zaznavanja (na primer digitalni model terena in satelitski podatki) najdemo

v TIFF nove razširitve z novimi značkami GeoTIFF [115]. Z njimi podamo informacije o umestitvi slik v prostor, na primer položaj in kartografska projekcija. Sliko lahko tudi razdelimo na trakove (angl. strips), s čimer dosežemo hitrejši dostop do posameznih delov slike. Prav zaradi množice značk pa je implementacija popolnega bralnika TIFF zahtevna.

Na podlagi specifikacije TIFF je izšlo še več razširitev, med katerimi so nekatere tudi standardizirane, na primer TIFF/EP (ISO 12234-2:2001 [116]) in TIFF/IT (ISO 12639:2004 [117]). V primeru aplikacij, ki zahtevajo še večje slike, kot jih omogoča TIFF, na primer daljinskega zaznavanja in medicinske slike, so razvili specifikacijo BigTIFF, ki uporablja 64-bitne vrednosti za navezovanje na podatke znotraj slike [118]. Glede na to, da je TIFF osnova nekaterih novejših datotečnih formatov, lahko TIFF imenujemo **vsebnik** (angl. container).

3.2.3.3.1 EXIF

Ponavadi želimo v sliko poleg pikslov shraniti tudi dodatne informacije, kot sta na primer avtorstvo slike in uporabljena naprava za zajem rastrske slike. Pri tem je pomembno, da te podatke shranimo na standardiziran način. EXIF (angl. Exchangeable image file format) [119] je specifikacija, namenjena zapisu metapodatkov različnih datotečnih formatov. Z razvojem je začela JEIDA (angl. Japan Electronic Industries Development Association), zdaj pa ga vzdržujeta organizaciji JEITA (angl. Japan Electronics and Information Technology Industries Association) in CIPA (angl. Camera & Imaging Products Association). Primeri najpogosteje uporabljanih metapodatkov so: nastavitve fotoaparata, datum, lokacija zajema, avtor, opis in predogledna sličica. Format zapisa metapodatkov je podoben kot pri TIFF, torej temelji na značkah. Specifikacija poleg opisa metapodatkov predpisuje tudi način vključitve metapodatkov v nekatere obstoječe slikovne datotečne formate. Pri datotečnem formatu TIFF se uporablja nov IFD, ki vsebuje seznam značk EXIF. Podobno velja za druge datotečne formate, ki jih bomo obravnavali v nadaljevanju: PNG, JPEG, WEBP in tudi WAV.

3.2.3.3.2 XMP

Za podoben namen so pod okriljem ISO razvili standard XMP (angl. Extensible Metadata Platform), ki metapodatke hrani v zapisu XML (standarda: ISO 16684-1:2019 [120] in ISO 16684-2:2014) [121]. Standard je nadgradnja starejšega IPTC (angl. Information Interchange Model). XMP predpisuje podatkovni model, značke, način serializacije podatkov in tudi način shranjevanja podatkov v obstoječe datotečne formate (na primer PNG, TIFF, GIF, PDF, JPEG, WebP, JPEG XL). V primeru TIFF se podatki XMP

zapišejo znotraj značke 700. XMP podpira tudi druge multimedijske tipe podatkov, kot na primer MP3.

3.2.3.4 PNG

Kot smo že omenili, je bila glavna motivacija za razvoj format PNG (angl. Portable Network Graphics) [122, 123, 124] patentna zaščita LZW, ki ga je uporabljal GIF. Zasnovala in razvila ga je leta 1995 spletna skupina navdušencev, ki jo je vodil Thomas Boutell. Pravo popularnost je doživel že naslednje leto, ko je W3C začel načrtno spodbujati njegovo uporabo. Od leta 2004 je mednarodni standard (ISO/IEC 15948 [125]). Danes je med najbolj popularnimi brezizgubnimi formati za stiskanje rastrskih slik, čeprav njegova učinkovitost zmanjševanja količine informacije ni več primerljiva s sodobnejšimi formati. PNG omogoča shranjevanje:

- črno-belih slik,
- sivinskih slik z 2^8 ali 2^{16} sivinskimi odtenki,
- slik, ki uporabljajo barvno paleto (od enobitne do osembitne),
- RGB z $2^{3\times8}$ ali $2^{3\times16}$ barvami.

PNG podpira prosojnost za sivinske in barvne slike. Omogoča samo barvno kodiranje RGB in vključuje možnost korekcije gama (angl. gamma correction), ki nadzira svetlost slike [126]. Namreč, slike, ki smo jih ustvarili na računalnikih z operacijskim sistemom macOS, so na računalnikih z operacijskim sistemom Windows ali Linux pretemne in obratno. S tem podatkom lahko prilagodimo prikaz slike glede na uporabljen operacijski sistem. PNG podpira tudi barvne profile ICC za enak videz na različnih napravah, informacije o fizičnih dimenzijah slike, podane v PPM in metapodatkovni zapis EXIF.

Glede na to, da so PNG načrtovali kot zamenjavo za GIF, vključuje tudi prepletanje, ki pa je bistveno izboljšano. Prepletanje **Adam7** deluje z bloki, kar daje boljšo vizualno zaznavo slike. Sliko najprej razdelimo na bloke velikosti 8×8 pikslov in piksle posameznih blokov prenesemo po naslednjem vrstnem redu:

- V prvem koraku prenesemo prvi piksel bloka.
- V drugem koraku sledi piksel v prvi vrstici in sredinskem stolpcu bloka.
- V tretjem koraku prenesemo piksla, ki sta v sredinski vrstici ter v prvem in sredinskem stolpcu bloka.

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

Slika 3.10: Vrstni red zapisanih pikslov bloka 8×8 pri prepletanju Adam7

- Če je blok večji od 2×2, razdelimo blok na štiri četrtine in rekurzivno ponovimo korake 2, 3 in 4 na vseh četrtinah.

Vrstni red pikslov v datoteki znotraj enega bloka vidimo na sliki 3.10, ko prepletanje Adam7 izvede sedem korakov. V primeru več blokov najprej prenesemo piksle iz posameznih iteracij iz vseh blokov. Tako sliko prikažemo najprej v osmini originalne ločljivosti, zatem v četrtini, polovici in nato celo sliko.

Naslednja razlika med konceptoma PNG in GIF je ta, da PNG uporablja napovedi, vrednosti napovedi odšteje od dejanskih vrednosti pikslov, napake pa zatem stisne z algoritmom Deflate.

Datoteke PNG so strukturirane tako, da so prvi zlogi enaki 0x89, 0x50, 0x4E, 0x47 0x0D, 0x0A, 0x1A in 0x0A, pri čemer drugi, tretji in četrti zlog predstavljajo ime PNG, zapisano v kodnem naboru ASCII. Sledijo **skupki** (angl. chunk), ki predstavljajo skupine tipov informacij. Vsak skupek hrani dolžino, tip skupka, podatke skupka (na primer piksle) in zgoščevalno vsoto podatkov (angl. cyclic redundancy check, CRC). Prvi skupek je praviloma IHDR, ki nosi osnovne informacije o sliki (širina, višina, bitna globina, način prepletanja). Nato sledi skupek IDAT, katerega podatki so stisnjeni z algoritmom Deflate. V datoteki PNG so lahko vključeni še preostali skupki, ki jih lahko pregledovalnik slik upošteva ali ne. Nekateri pogostejši so:

- PLTE – barvna paleta,
- cHRM – kalibracija barv,
- gAMA – razmerje med svetlostjo pikslov na sliki in svetlostjo na prikazovalniku,
- sBIT – število najpomembnejših bitov, ki nosijo informacije, kar se uporablja v primeru shranjevanja podatkov z več biti, kot je potrebno,

- tRNS – prosojnost,
- tIME – čas in datum zadnje spremembe slike,
- pHYs – dejanske fizične dimenzije pikslov, ki so za razliko od sorodnih tehnologij podane v številu pikslov na meter,
- tEXt – dodatne tekstovne informacije, kodirane z Latin-1 oziroma ISO/IEC-8859-1 [127],
- zTXt – dodatne tekstovne informacije, ki so stisnjene z Deflate,
- bKGD – barva ozadja,
- iTXt – dodatne tekstovne informacije, kodirane z UTF-8, pri čemer so lahko podatki tudi stisnjeni z Deflate,
- iCCP ali sRGB – za bolj natančno kalibracijo barv, ki preglasita cHRM in gAMA.

Metapodatki EXIF se v datoteke PNG zapišejo v skupek »eXIf«, v primeru metapodatkov XMP pa v skupek iTXt s ključno besedo »XML:com.adobe.xmp«.

3.2.3.4.1 Napovedno kodiranje

Pri napovednem kodiranju vrednosti pikslov obdelamo s petimi napovedmi (specifikacija PNG uporablja besedo *filter*). Filtre uporabimo nad posame-

Tabela 3.1: Oznake filtrov pri PNG

tip	0	1	2	3	4
ime	None	Sub	Up	Average	Paeth

znimi vrsticami. Vsaka vrstica uporablja enega izmed filtrov iz tabele 3.1. Stiskanje izvedemo na nivoju posameznih zlogov (ne pikslov) in je zato neodvisno od barvne globine, načina kodiranja barve piksla in uporabe kanala alfa. Filtri so naslednji:

1. **Filter None:** V tem primeru vrstice ne filtriramo, torej:

$$\mathrm{None}(x) = \mathrm{Raw}(x), \tag{3.4}$$

pri čemer x označuje položaj zloga v vrstici, $\mathrm{Raw}(x)$ pa je vrednost zloga na položaju x.

2. **Filter Sub:** Zlog, ki ga kodiramo, dobimo kot razliko dveh enakoležnih zlogov zaporednih pikslov:

$$Sub(x) = (\text{Raw}(x) - \text{Raw}(x - Bpp)) \bmod 256, \tag{3.5}$$

kjer Bpp določa število zlogov na piksel (angl. bytes per pixel). Če je vrednost manjša kot 1, jo zaokrožimo na 1. Na primer, če je bitna globina 16 in imamo barvno sliko v RGB, je $Bpp = 6$. Za črno-belo sliko z bitno globino 1 je $Bpp = 1$.

3. **Filter Up:** V tem primeru izračunamo razliko med zlogi obravnavanega piksla in istoležnimi zlogi piksla nad obravnavanim pikslom:

$$Up(x) = (\text{Raw}(x) - \text{Prior}(x)) \bmod 256, \tag{3.6}$$

kjer je $\text{Prior}(x)$ zlog na položaju x v predhodni skanirni vrstici. Ko je $x = 0$, so vse vrednosti $\text{Prior}(x) = 0$.

4. **Filter Average:** Pri tem filtru uporabljamo zloge iz dveh pikslov (iz levega in zgornjega):

$$\text{Average}(x) = \text{Raw}(x) - \left\lfloor \frac{\text{Raw}(x - Bpp) + \text{Prior}(x)}{2} \right\rfloor \bmod 256. \tag{3.7}$$

5. **Filter Paeth:** (filter je predlagal A. W. Paeth). Tokrat določimo napoved s tremi vrednostmi enakoležečih zlogov iz treh pikslov: levega, piksla nad trenutno kodiranim pikslom in zgoraj levo od trenutno kodiranega piksla (enačba 3.8).

$$\text{Paeth}(x) = (\text{Raw}(x) - \text{FPaeth}(x)) \bmod 256. \tag{3.8}$$

Funkcijo FPaeth določimo s psevdokodom 1. Filter FPeath lahko zapišemo tudi z enačbo 3.9 [128]. Pri tem so $a = \text{Raw}(x - Bpp)$, $b = \text{Prior}(x)$ in $c = \text{Prior}(x - Bpp)$.

$$\text{FPaeth}(a, b, c) = \begin{cases} a : \text{if } |b - c| \leq |a - c| \wedge |b - c| \leq |a + b - 2c| \\ b : \text{if } |a - c| \leq |a + b - 2c| \\ c : \text{otherwise} \end{cases} \tag{3.9}$$

Zloge, dobljene iz filtrov, nato stisnemo z algoritmom Deflate.

Psevdokod 1: Izračun napovedi Paeth

```
1:  function FPaeth(x)
2:
3:      a = Raw(x − Bpp)                          ▷ vrednost levo
4:      b = Prior(x)                              ▷ vrednost zgoraj
5:      c = Prior(x − Bpp)                        ▷ vrednost zgoraj levo
6:      p = a + b − c;                            ▷ začetna napoved
7:      pa = abs(p − a);                          ▷ razdalja do a
8:      pb = abs(p − b);
9:      pc = abs(p − c);
10:     if (pa ≤ pb) AND (pa ≤ pc) then return a;
11:     end if
12:     if pb ≤ pc then return b;
13:     end if
14:     return c;
15: end function
```

3.2.3.5 MNG in APNG

Pomembna omejitev datotečnega formata PNG glede na GIF je nezmožnost shranjevanja animacij. Leta 1999 so zato predlagali specifikacijo za razširitev PNG, imenovano MNG (angl. Multiple-Image Network Graphics) [129]. MNG vključuje tako podporo za animacije kot izgubno stiskanje. Zaradi zahtevnosti pa ni uspel in se opušča.

Uporabnejši je format APNG (angl. Animated Portable Network Graphics), ki so ga razvili leta 2004. APNG prinaša zgolj manjše nadgradnje formata PNG za podporo animacijam. APNG je podprt tudi v večini novejših spletnih brskalnikov in je vključen v uradno različico specifikacije za PNG [124].

Pomembna lastnost APNG je združljivost s PNG. Prva sličica animacije je shranjena v formatu PNG, kar omogoča pregledovanje datotek APNG s starejšo programsko opremo, ki prikaže zgolj prvo sličico. Parametri animacije in dodatne sličice pa so shranjeni v dodatnih neobveznih skupkih iste datoteke PNG. Sličice animacije so zapisane zgolj z razlikami glede na predhodne sličice, s čimer zmanjšamo velikost datoteke.

3.2.3.6 JPEG in JFIF

Z razvojem digitalnih fotoaparatov se je v 70. letih prejšnjega stoletja pojavilo veliko proizvajalcev teh naprav, ki pa so slike shranjevali na različne načine in v lastnih, med seboj nekompatibilnih formatih. Zato so pri CCITT (franc.

Comite Consultatif Internationale de Telegraphique et Telephonique) in ISO (angl. International Standard Organization) ustanovili skupno delovno skupino *Joint Photographic Experts Group* ali krajše *JPEG*, ki bi izdelala standard za učinkovito izgubno stiskanje barvnih slik z zveznimi barvnimi toni (to je za fotografije). Leta 1991 so predstavili predlog standarda, ki je dobil oznako ISO/IEC 10918-1 [130] oziroma CCITT Rec. T.81 [131]. Standard so vodilni proizvajalci tako računalniške opreme kot digitalnih fotoaparatov sprejeli s širokimi rokami, tako da je JPEG še danes uspešen in široko uporabljan standard za stiskanje digitalnih fotografij. Načrtovalci so definirali postopke in izhodni format (angl. JPEG File Interchange Format, JFIF), način implementacije pa so prepustili razvijalcem. Standard omogoča štiri načine stiskanja:

- **Zaporedno stiskanje** (angl. baseline, sequential). Je najpogostejši način, kjer vsako barvno komponento stisnemo v enem prehodu slike v prebirnem vrstnem redu.

- **Napredujoče stiskanje** (angl. progressive). V tem načinu stisnemo sliko v več prehodih tako, da najprej vidimo grobo, neostro sliko, nato pa se njena kakovost z vsakim prehodom izboljšuje. Uporabnik tako hitro dobi osnovne informacije o sliki. Pristop je uporaben pri zelo velikih slikah ali nizkih hitrostih prenosa podatkov.

- **Hierarhično stiskanje** (angl. hierarchial). Omogoča shranjevanje slike v več ločljivostih tako, da je skupna količina podatkov manjša od ločenega shranjevanja posameznih slik.

- **Stiskanje brez izgub** (angl. lossles). Temelji na napovednem modelu. Način pa se je izkazal za neučinkovitega in v praksi ni v uporabi.

V nadaljevanju si bomo ogledali samo najpogosteje uporabljeno zaporedno stiskanje, katerega koraki so naslednji:

1. Pretvorba barvnih prostorov (iz RGB v YCbCr). Pri sivinskih slikah ta korak preskočimo.

2. Komponentama barvitosti (*Cb, Cr*) zmanjšamo ločljivost. Pri sivinskih slikah tudi ta korak preskočimo.

3. Vsem pikslom odštejemo vrednost 128, tako da dobimo vrednosti na intervalu $[-128, 127]$.

4. Vrednosti združimo v skupine/bloke velikosti 8×8.

5. Vsak blok transformiramo v frekvenčni prostor z **diskretno kosinusno transformacijo** (angl. discrete cosine transform, **DCT**), s katero izračunamo 64 frekvenčnih komponent tega dela slike. Dobljene frekvenčne koeficiente prav tako shranimo v matriko 8×8. Komponenta v zgornjem levem kotu je komponenta **DC**, druge pa **AC**. Komponenta DC predstavlja povprečno barvo piksla v bloku, komponente AC pa višje harmonske frekvence, pri čemer so koeficienti, ki predstavljajo najvišje frekvence, v spodnjem desnem delu matrike.

6. Koeficiente DCT kvantiziramo z deljenjem in zaokrožanjem navzdol. Močnejšo kvantizacijo uporabimo za višjeharmonske koeficiente. Na ta način je predvsem v desnem spodnjem delu matrike 8×8 veliko koeficientov z vrednostjo 0. Kvantizacija vnaša izgubo in z njo nadziramo razmerje kakovosti slike v primerjavi z velikostjo datoteke.

7. Vsako komponento DC zapišemo z razliko glede na koeficient DC iz predhodnega bloka (napoved).

8. Koeficiente prebiramo od levega zgornjega dela matrike proti desnemu spodnjemu delu v tako imenovanem zaporedju cik-cak. Nato ločeno stisnemo koeficiente AC in DC. V prvem koraku uporabimo RLE, nato pa Huffmanovo ali aritmetično kodiranje.

Celoten postopek JPEG bomo podrobneje spoznali na drugi študijski stopnji. JPEG ima poleg veliko dobrih lastnosti tudi nekaj slabih:

- Izračun diskretne kosinusne transformacije ima visoko časovno zahtevnost, zaradi česar smo prisiljeni omejiti velikost bloka. Pri večjih stopnjah stiskanja zato pride do zrnatosti slike, saj vsak blok opišemo le z njegovo povprečno barvo (ostane samo koeficient DC, vsi koeficienti AC so kvantizirani in postavljeni na 0). Žal je človeško oko na takšno zrnatost zelo občutljivo, kar nam daje iluzijo svetlih robov na mejah področij. Pojav poznamo kot **Machov pojav** (angl. Mach effect) in ga vidimo na sliki 3.11.

- Omogoča samo 3 barvne kanale, ne podpira prosojnosti.

- Prvotni JPEG podpira samo barvne slike z bitno globino 8 in s tem postaja preveč omejujoč za moderne fotoaparate.

- Brezizgubni način je zelo neučinkovit.

- JPEG ni primeren za stiskanje slik z ostrimi barvnimi prehodi, saj vnaša moteč šum.

(a) (b) (c)

Slika 3.11: Stiskanje (a) izvorne slike 3.7a z velikim razmerjem stiskanja povzroči (b) zrnatost z izrazitim Machovim pojavom. (c) Pri najnižji kakovosti vidimo zgolj še bloke 8×8.

Slika 3.12: JPEG je neprimeren za stiskanje teksta.

- JPEG ni primeren za stiskanje inženirskih risb in teksta (slika 3.12).

V okviru standarda ISO/IEC 10918 [130] oziroma T.81 [131] je bil sprva predlagan datotečni format JIF (angl. JPEG Interchange Format), nato pa je izšla izboljšava v obliki standarda ISO/IEC 10918-5 [132] z imenom JFIF (angl. JPEG File Interchange Format), ki se uporablja še danes. Datotečna struktura JIF temelji na podlagi segmentov in njihovih označevalcev (angl. marker). Oznaka označevalcev je dolga 4 zloge, sledi dolžina podatkov v segmentu, nato pa podatki segmenta, na primer stisnjeni podatki slike ali metapodatki. Datoteka JFIF se začne z zlogi 0xFFD8 (označevalec SOI – angl. Start Of Image), ki označujejo tip datoteke, nato sledi segment APP0, ki vsebuje osnovne metapodatke in tudi predogledno sličico. Stisnjeni piksli so v segmentu SOS (angl. Start Of Scan), v segmentu DHT (angl. Define Huffman Table) se nahaja Huffmanova tabela, označevalci SOF določajo enega izmed prej omenjenih štirih načinov kodiranja slike in tudi način stiskanja entropije. V zadnjem času se vedno bolj uveljavljajo datoteke JFIF, strukturirane na podlagi specifikacije EXIF. Glavna razlika je v tem, da se

namesto segmenta APP0 uporablja segment APP1, ki vsebuje metapodatke Exif, torej IFD iz datotečnega formata TIFF.

Skupina JPEG je aktivna še danes. Predlagala je različne razširitve in izboljšave prvotnega standarda [133], kot so JPEG XT, JPEG XR, JPEG XS in JPEG XL, ki jih obravnavamo v nadaljevanju.

3.2.3.7 JBIG

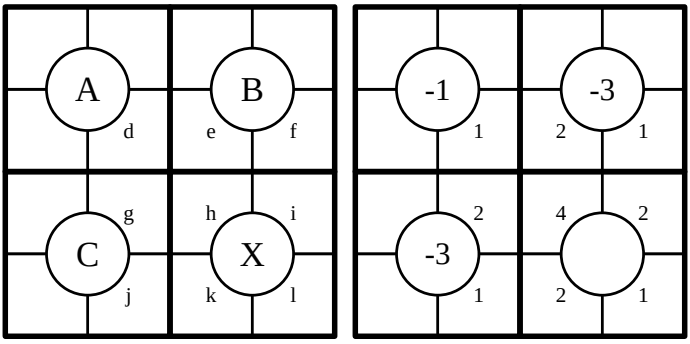
JBIG (angl. Joint Bi-level Image Experts Group) je mednarodni standard iz leta 1993 (ISO/IEC 11544 [134], ITU-T T.82 [135]), ki je namenjen napredujočemu kodiranju črno-belih slik [136, 7]. Najpogostejši vir črno-belih slik so skenirani dokumenti in podatki iz faksimilnih naprav (angl. facsimile). JBIG lahko uporabimo tudi pri brezizgubnem stiskanju sivinskih slik, pri čemer pa stiskamo vrednosti bitov po posameznih bitnih ravninah. Čeprav je standard JBIG manj znan, je pri stiskanju črno-belih slik izjemno učinkovit. Omogoča namreč vsaj dvakrat boljše razmerje stiskanja od gzip (Deflate) in od tri- do štirikrat boljše razmerje stiskanja od GIF glede na trditve avtorjev knjižnice JBIG-KIT [137].

Preprostejši algoritmi za stiskanje črno-belih slik stiskajo skanirne vrstice z RLE. JBIG pa pri kodiranju trenutnega piksla napove njegovo barvo na podlagi konteksta že zakodiranih pikslov. Verjetnost pojavitve črnega (oziroma belega) piksla potem uporabi v aritmetičnem kodirniku s prilagajanjem. Le-ta, odvisno od implementacije, uporablja od 1024 do 4096 statističnih modelov s prilagajanjem. Hkrati pa kodiranje deluje tako, da omogoča napredujoče (angl. progressive) stiskanje.

Kodiranje pri JBIG deluje tako, da najprej hierarhično izvede **podvzorčenje** (angl. downsampling), združi namreč piksle v višji ločljivosti v piksle v nižji ločljivosti. Nato napredujoče zakodira slike tako, da za vsako sliko v višji ločljivosti izračuna razlike glede na sliko v nižji ločljivosti. Pri kodiranju razlik uporablja aritmetični kodirnik s prilagajanjem (angl. adaptive arithmetic encoder), ki stanje kodirnika prilagaja glede na okoliške piksle. Postopek dekodiranja je obraten. Dekodirnik najprej prejme slike v nižji ločljivosti, zato lahko že takoj prikaže aproksimacijo originalne slike. Dekodirnik nato postopoma prejema razlike do slike v višji ločljivosti in jih uporabi za izboljšanje slike v višji ločljivosti.

Napredujoče stiskanje temelji na ponavljajočem se *podvzorčenju* na polovično ločljivost. Na vsakem nivoju ločljivosti združimo štiri sosednje piksle v skupnega predstavnika v nižji ločljivosti. Pri tem moramo paziti, da se pri slikah v nižji ločljivosti ohranijo ključne podrobnosti, na primer tanke črte in vzorci raztresanja. V primeru najbolj osnovnega podvzorčenja, torej na podlagi štetja črnih sovpadajočih pikslov v višji ločljivosti, bi zelo

hitro izgubili videz originalne slike. Zaradi tega JBIG podvzorči na podlagi oddaljenejših pikslov oziroma širše povezave/konteksta, kot ga vidimo na sliki 3.13a. Za izračun piksla X slike v polovični ločljivosti upošteva vredno-



(a) (b)

Slika 3.13: (a) Piksli v višji (male črke) in nižji ločljivosti (velike črke), potrebni za določitev vrednosti piksla X, (b) uteži teh pikslov glede na standard ITU-T T.82.

sti predhodnih pikslov iz skanirne pretvorbe A, B in C ter vrednosti pikslov od d do l slike v višji (dvojni) ločljivosti. Za določitev vrednosti piksla X specifikacija JBIG določa vpogledno tabelo, ki na podlagi 12 pikslov A, B, C in od d do l določa novo vrednost piksla X. Zato ima tabela 2^{12} oziroma 4096 vnosov. Pri tem je posebna pozornost namenjena množici izjem in pravil, s katerimi ohranja robove (132 izjem), vodoravne in navpične robove (420 izjem), ponavljajoče se vzorce (10 izjem) in vzorce raztresanja (angl. dithering) (12 izjem). Preostale vnose vpogledne tabele JBIG izračunamo tako, da pikslom najprej priredimo uteži, kot vidimo na sliki 3.13b, in potem izračunamo njihovo uteženo vsoto S z enačbo 3.10.

$$S = 4h + 2(e + g + i + k) + (d + f + j + l) - 3(B + C) - A. \qquad (3.10)$$

Hiter izračun nam pokaže, da je vrednost enačbe 3.10 0, če so vsi piksli beli (imajo vrednost 0), oziroma 9, če so vsi piksli črni (imajo vrednost 1). Če je $S > 4{,}5$, bo piksel X 1 (črn), sicer bo 0 (bel).

V letih 2000 in 2001 sta ISO in ITU objavila izboljšan standard JBIG2 (ISO/IEC 14492 [138] in ITU-T T.88 [139]), ki prinaša naslednje nadgradnje:

- Nove metode za stiskanje teksta in *poltoniranih* (angl. halftone) slik ali delov slik. V primeru stiskanja slike besedila se posamezne črke v sliki pojavljajo zelo pogosto. Zaradi tega JBIG2 uvaja slovar, v katerega vstavi samo eno sliko črke in jo potem uporabi večkrat.

- Izgubno in brezizgubno stiskanje: V primeru brezizgubnega stiskanja se ob uporabi slovarja zakodirajo tudi napake oziroma popravki napovedi glede na originalno sliko. V primeru izgubnega stiskanja kodiranje napak izpustimo.
- Možnost izbire med adaptivnim aritmetičnim kodirnikom MQ ali Huffmanovim kodirnikom, saj je patent kodirnika MQ [140] potekel šele pred kratkim.
- Dva načina napredujočega stiskanja:
  - Klasičen način, kjer se slika izboljšuje iz nižje proti višji kakovosti.
  - *Kontekstno-odvisno* napredujoče kodiranje, kjer se najprej prenesejo pomembnejši deli slike, na primer najprej tekst in potem sivine.
- Podpora za stiskanje dokumentov z več stranmi.
- Boljša razširjenost, saj je možno v datoteke PDF vključiti podatke JBIG2.
- Od tri- do štirikrat boljše razmerje stiskanja od JBIG in neprimerljivo boljši rezultati od na primer GIF in PNG [141].

Uporaba slovarja v JBIG2 je bila zelo uspešna za zmanjševanje velikosti datotek. Zato to idejo uporablja tudi odprta specifikacija za stiskanje dokumentov DjVu [142].

3.2.3.8 JPEG-LS

JPEG-LS (angl. Joint Photographic Experts Group – LosslesS) je mednarodni standard ISO/IEC 14495-1:1999 [143] za brezizgubno ali skoraj brezizgubno stiskanje slik z zveznimi barvnimi toni (fotografije) [144, 145, 146, 147]. JPEG-LS kot priporočilo uvaja tudi specifikacija ITU-T T.87 [148], nadgradnje in izboljšave pa najdemo pod oznako ISO/IEC 14495-2:2003 [149] in ITU-T T.870 [150]. Najpogostejša končnica datotek formata JPEG-LS je JLS.

Kodiranje deluje v dveh načinih: **navadni način** (angl. regular mode) in **način s ponavljanjem** (angl. run mode). Navadni način temelji na napovedovanju vrednosti trenutno kodiranega piksla na podlagi okolice že zakodiranih pikslov. Nato napoved odštejemo od dejanske vrednosti kodiranega piksla, s čimer dobimo zaporedje napak. Napake so praviloma majhne celoštevilske vrednosti. Zaradi tega za zapis napak uporablja Golombovo kodiranje [26]. Za dodatno zmanjšanje napak pri napovedi JPEG-LS

uvaja še kontekstno modeliranje. Poleg korekcije napovedi s kontekstnim modeliranjem določamo tudi Golombov parameter. V primeru skoraj brezizgubnega stiskanja napake zaokrožimo na ustrezno natančnost oziroma jih kvantiziramo na takšen način, da so odstopanja posameznih pikslov glede na originalno sliko znotraj vnaprej določene meje. Drug način, torej način s ponavljanjem, kodirnik uporabi v primeru zaporedja enakih pikslov. Za kodiranje tega zaporedja uporablja prilagojeno kodiranje RLE. Standard JPEG-LS pa kljub svoji učinkovitosti ni prepričal uporabnikov.

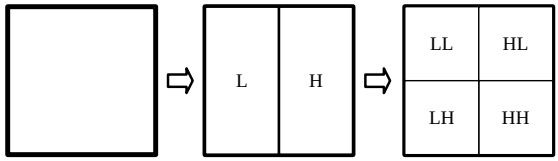
3.2.3.9 JPEG2000

Delovna skupina JPEG je po objavi standarda JPEG nadaljevala delo in v letu 2000 objavila nov standard JPEG2000 ISO/IEC 15444-1 [151] (standardna oznaka datoteke .jp2) in pozneje njegovo razširitev ISO/IEC 15444-2 [152]) [153]. JPEG2000 vključuje veliko drugačnih rešitev glede na JPEG. Tudi jedro JPEG, diskretno kosinusno transformacijo, so zamenjali z **diskretno valčno transformacijo** (angl. discrete wavelet transform – DWT).

JPEG2000, podobno kot JPEG, v prvem koraku pretvori barvni format RGB v YCbCr ali v modificiran YUV za brezizgubno stiskanje, nato pa vsako barvno komponento obdeluje posebej kot sivinsko sliko. JPEG2000 sliko deli v **ploščice** (angl. tiles). Ploščice so mnogo večje kot bloki pri JPEG. Slike, ki niso res zelo velike, so predstavljene z eno samo ploščico. S tem se rešimo pojava zrnatosti in Machovega pojava. Vsaka ploščica je obdelana povsem neodvisno od drugih ploščic, s čimer zagotovimo računsko učinkovitost in naključen dostop v dele slike.

JPEG2000 temelji na **diskretni valčni transformaciji** [154, 155], ki izvede konvolucijo z izvornim signalom (ploščico), tako da razdeli njegovo frekvenčno vsebino v nizek (L) in visok (H) pas (angl. band). JPEG2000 uporablja družino valčkov **Daubechies** za izgubno stiskanje in celoštevilske valčke **Le Gall** za brezizgubno stiskanje. Postopek določitve valčkov bomo spoznali na drugi stopnji.

V primeru slik opravimo delitev na nizek in visok pas v dveh korakih (slika 3.14). V prvem prehodu razdelimo vrstice slike v nizek (L) in visok pas



Slika 3.14: Delitev slike v frekvenčne pasove

(H) polovične ločljivosti, nato pa stolpce teh dveh pasov ponovno transformiramo tako, da dobimo štiri področja: področje LL hrani nizke frekvence, preostala področja pa višjeharmonske značilnosti, ki jih lahko v nadaljnjem postopku stiskanja tudi kvantiziramo in s tem nadzorujemo izgube. Postopek transformacije nato hierarhično nadaljujemo nad področjem LL, kot vidimo na sliki 3.15.

(a) (b) (c)

Slika 3.15: Valčna transformacija sivinske slike 3.7a od nivoja 1 do nivoja 3

JPEG2000 za kodiranje kvantiziranih koeficientov uporablja algoritem EBCOT (angl. Embedded Block Coding with Optimised Truncation) [156]. EBCOT je razdeljen v dva dela; v prvem pripravimo podatke za kodiranje z aritmetičnim kodirnikom v drugem koraku, kjer JPEG2000 uporablja kodirnik MQ (angl. MQ-Coder) [7].

(a) (b) (c)

Slika 3.16: Slika 3.7a, stisnjena z (a) JPEG pri 2,90 bita na piksel (bpp), (b) JPEG pri 0,54 bpp in (c) JPEG2000 pri 0,35 bpp.

Povzemimo še ključne lastnosti JPEG2000:

- Zelo dobra učinkovitost. Slika 3.16 kaže primerjavo učinkovitosti med

JPEG in JPEG2000. Na sliki 3.17 vidimo rezultat rekonstrukcije stisnjene slike JPEG2000 pri različnih bpp, pri čemer pri nizkih bpp ne opazimo težav z bloki.

(a) (b) (c) (d)

Slika 3.17: Slika 3.7a, stisnjena z JPEG2000 pri (a) 1,89 bita na piksel (bpp), (b) 0,51 ppb, (c) 0,17 bpp in (d) 0,03 bpp.

- Izgubno in brezizgubno stiskanje uporabljata enak cevovod s to izjemo, da pri brezizgubnem stiskanju ne uporabimo kvantizacije.
- Napredujoče stiskanje dosežemo s primerno organizacijo podatkov prenosa višjeharmonskih koeficientov po valčni transformaciji.
- Standard omogoča stiskanje delov slik v boljši kakovosti (angl. region of interests – **ROI**). JPEG2000 omogoča tudi naključni dostop do ROI.
- Kodiranje podpira preverjanje pravilnosti prenešenih podatkov v kanalih s šumom.
- Omogoča globino barvnih kanalov do 32 bitov.
- Podpira transparentne slike.

Kljub vsemu pa v praksi JPEG2000 ni zelo uspešen, saj ga industrija ni podprla. Za klasične fotografije in rokovanje z njimi je namreč JPEG zaenkrat še uporaben. Široko uporabo JPEG2000 od vsega začetka omejujejo tudi patentne pravice in licenčnine razvijalcev. Z vsem bogatim naborom možnosti in lastnosti pa je tudi zelo zahteven za implementacijo.

3.2.3.10 WebP

WebP je datotečni format za izgubno in brezizgubno stiskanje barvnih rastrskih slik z zveznimi barvnimi toni, ki ga je razvilo podjetje Google [157]. Format podpirajo v njihovem brskalniku, implementiran pa je tudi v drugih

novejših spletnih brskalnikih. Cilj formata je biti učinkovitejši od JPEG ter manj zapleten in obremenjen z licenčninami kot JPEG2000. WebP omogoča:

- izgubno stiskanje, ki uporablja postopke stiskanja **ključnih okvirjev** oziroma izvornih slik videa pri videoformatu **VP8**, ki ga bomo spoznali pozneje,
- ločen postopek za brezizgubno stiskanje,
- kanal alfa z bitno globino 8,
- animacije,
- razdelitev slike na **segmente**, pri čemer lahko za vsak segment uporabimo drugačno stopnjo kvantizacije.

Datotečni format uporablja podatkovni tok videoformata VP8 znotraj vsebnika **RIFF** (angl. Resource Interchange File Format), ki je osnova datotečnega formata WAV in ga bomo spoznali v naslednjem poglavju.

Večina korakov, ki jih uporablja izgubno stiskanje WebP, je prevzeto iz JPEG. Najprej opravi pretvorbo iz barvnega modela RGB v YUV in izvede podvzorčenje kanalov UV za polovico v obeh smereh. Nato razdeli sliko v bloke. Ključna izboljšava v primerjavi z JPEG je napovedovanje vrednosti blokov. WebP uporablja štiri osnovne možnosti:

- **vodoravna napoved** (angl. horizontal prediction, H_PRED): vsak stolpec bloka za napoved je kopija najbolj levega stolpca bloka;
- **navpična napoved** (angl. vertical prediction, V_PRED): vsaka vrstica napovednega bloka je kopija najbolj zgornje vrstice bloka;
- **napoved DC** (angl. DC prediction, DC_PRED): vrednosti v bloku za napoved dobimo kot povprečje vrednosti v zgornji vrstici in levem stolpcu;
- **napoved gibanja** (angl. TrueMotion prediction, TM_PRED) [158]: Napovedni blok B velikosti $n \times n$ napovemo na podlagi sosednje zgornje vrstice A, sosednjega levega stolpca L in vrednosti sosednjega levega zgornjega piksla C od bloka B, torej levo od A_0 nad L_0. Vrednost na položaju $B_{i,j}$, kjer sta i in $j \geq 0$, dobimo z enačbo 3.11.

$$B_{i,j} = L_i + A_j - C \tag{3.11}$$

Za kanal Y je možnih še šest dodatnih napovedi, podobnih V_PRED in H_PRED pri različnih orientacijah.

WebP napovedane vrednosti nato odšteje od enakoležečih dejanskih vrednosti. Razlike so praviloma majhne po absolutni vrednosti (izberemo tisto napoved, kjer je absolutna vsota razlik najmanjša), zato je lahko stiskanje uspešnejše. Koraki, ki sledijo za tem, so podobni korakom pri JPEG: DCT, kvantizacija koeficientov in tvorba zaporedja cik-cak. WebP v zadnjem koraku uporabi binarno aritmetično kodiranje s prilagajanjem.

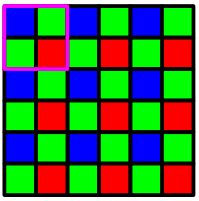
Ginesun [159] je s sodelavcema opravil primerjavo WebP z družino algoritmov JPEG (JPEG, JPEG2000 in JPEG XR). Primerjali so podobnost stisnjenih slik originalu pri različnih stopnjah stiskanja. Pri izjemno velikih stopnjah stiskanja daje WebP primerljive rezultate kot JPEG2000. Pri generiranih slikah z nezveznimi barvnimi toni je WebP slabši od JPEG2000, a boljši od JPEG in JPEG XR. WebP je bil v primerjavi z JPEG počasnejši pri kodiranju in hitrejši pri dekodiranju.

Brezizgubno stiskanje WebP deluje po popolnoma drugačnem postopku. Med kanali vhodne slike izvede *dekorelacijo* tako, da kanal R izrazi s kanalom G, kanal B pa določi s kanaloma G in R. Nato uporabi 13 napovednih modelov za odpravo prostorske korelacije na nivoju blokov. WebP predpisuje samodejno uporabo barvne palete pri manj kot 256 uporabljenih barvah (indeksni način zapisa barv). Naslednja zanimiva tehnika je uporaba **medpomnilnika barv** (angl. color caching), ki deluje kot drseče okno z lokalno barvno paleto (zadnjih 32 uporabljenih barv) in omogoča dostop do nazadnje uporabljenih barv z manj biti. Ponavljajoče dele slike kodira z LZ77, pri čemer pomembnejše bite odmikov in dolžin zakodira s Huffmanovim kodirnikom, manj pomembne bite pa zapiše brez stiskanja.

3.2.3.11 Shranjevanje surovih slik – RAW

Pri shranjevanju slik v na primer formatu PNG ponavadi izpustimo originalne podatke iz senzorja (digitalizacija). Intenziteta svetlobe na barvni kanal je namreč pogosto predstavljena z 12 ali 14 biti, barvni model pa velikokrat ni RGB, ampak BGGR, kot je prikazano na sliki 3.18.

Za shranjevanje originalnih informacij z digitalnega fotoaparata ter dodatnih informacij o fotoaparatu in okoliščinah zajete slike je možno uporabiti enega izmed namenskih formatov za predstavitev surovih (angl. raw) slik. Pri tem so proizvajalci digitalnih fotoaparatov v preteklosti razvili lastne formate, na primer: erf (EPSON), .crw, .cr2, .cr3 (Canon) in .nef/.nrw (Nikon), ki so temeljili na vsebniku TIFF. Zaradi tega je pogosto prihajalo do težav z združljivostjo z različno programsko opremo. Poleg tega so bili podatki pogosto shranjeni brez stiskanja.



Slika 3.18: Postavitev posameznih barvnih komponent znotraj enega piksla (filter **Bayer**)

Med bolj razširjenimi formati je treba omeniti DNG (angl. digital negative) in starejši standard TIFF/EP (angl. Tag Image File Format/Electronic Photography) – ISO 12234-2 [116]. DNG je odprta specifikacija formata podjetja Adobe [160] in kot osnovo uporablja format oziroma vsebnik (angl. container) TIFF. Za stiskanje podatkov lahko uporablja algoritem Deflate ali pa JPEG oziroma JPEG XL, ki ga obravnavamo v nadaljevanju. Glede na to, da podatki RAW praviloma zahtevajo veliko prostora in potrebujejo zahtevnejše algoritme za prikaz na prikazovalniku, DNG omogoča hrambo predoglednih sličic za hiter prikaz v brskalniku datotek. V tem primeru lahko uporablja izgubno stiskanje JPEG. Podobno kot predhodni formati DNG vključuje podporo za metapodatkovne zapise EXIF, XMP in IPTC ter barvne profile ICC.

3.2.3.12 JPEG XT

JPEG XT (angl. eXtension) je neposredna nadgradnja JPEG, standardizirana kot ISO/IEC 18477 [161]. JPEG XT omogoča bitno globino do 16 bitov na kanal tudi za izgubno stiskanje, izboljšuje brezizgubno stiskanje ter dodaja kanal alfa. Nadgradnje so izvedene tako, da ohranijo popolno *združljivost* s starejšimi bralniki JPEG [162]. Združljivost za nazaj pri večjih bitnih globinah dosega tako, da najbolj pomembne bite kodira s klasičnim JPEG, manj pomembne pa zakodira ločeno. Tudi za brezizgubno stiskanje uporablja DCT in kvantizacijo iz JPEG, dobljene vrednosti rekonstruiranih pikslov pa uporabi kot *napoved*. Razlike med napovedanimi in dejanskimi vrednostmi nato zapiše v razširitvi osnovnega JPEG.

3.2.3.13 JPEG XR

V letih 2005/2006 je podjetje Microsoft na podlagi kodirnika PTC (angl. Progressive Transform Codec) [163] razvilo nov format za shranjevanje slik HD Photo. Slednjega so leta 2009 standardizirali (ISO/IEC 29199-2/ITU-T

T.832 [164, 165]) pod imenom JPEG XR (angl. eXtended Range). Standard se osredotoča na še boljše stiskanje slik z zveznimi barvnimi toni (fotografije). Po zmogljivosti je primerljiv z JPEG2000, glede na hitrost pa z JPEG. Format povečuje bitno globino kanala na 16 in 32 bitov. Poleg barvnega modela RGB omogoča tudi model CMYK, slike deli v ploščice z naključnim dostopom, izboljšuje napredujoč način prenosa ter se pri tem prilagaja širini komunikacijskega kanala (*skalabilnost*), podpira prosojnost in omogoča predstavitev vrednosti pikslov s plavajočo vejico (na primer slike daljinskega zaznavanja).

Cevovod stiskanja je podoben JPEG, vendar prinaša nekaj novih konceptov. V enotnem cevovodu podpira tako izgubno kot brezizgubno stiskanje. DCT nadgrajuje z novo **delno hierarhično** transformacijo, ki odpravljajo vidnost blokov pri visokih razmerjih stiskanja, ki je značilno za JPEG. Transformacijo najprej izvede nad bloki velikosti 4×4 pikslov, nato pa še nad koeficienti DC 4×4 transformiranih blokov (koeficient DC ima podobno vlogo kot pri DCT). Za shranjevanje uporablja vsebnik TIFF. Najpogostejša končnica datotek je JXR, v preteklosti pa tudi HDP in WDP.

3.2.3.14 JPEG XS

JPEG XS (angl. eXtra Small/Speed) je standard ISO/IEC 21122 [166], ki je namenjen stiskanju slik z zelo majhno **zakasnitvijo** (angl. latency) za kodiranje in dekodiranje s pomočjo vzporedne obdelave na napravah, kot so FPGA (angl. A field-programmable gate array), ASIC (angl. application-specific integrated circuit) in grafične procesne enote. Namenjen je za aplikacije, ki so zaradi zahtev po sprotnosti morale uporabljati nestisnjene slike (shranjevanje slik v realnem času, naglavni prikazovalniki za navidezno resničnost, avtomobilska industrija s samovozečimi vozili in videopovezave). Zaradi tega je ključna zahteva standarda zagotoviti **vizualno brezizgubno stiskanje**, kot je specificirano s standardom ISO/IEC 29170-2 [167], pri čemer dosega razmerje stiskanja do 1 : 10 [168]. Podpira 12–bitne barvne kanale, vsebuje pa tudi namenske postopke za stiskanje surovih slik s fotoaparatov (filter Bayer).

JPEG XS je po delovanju podoben JPEG2000, saj uporablja diskretno valčno transformacijo, vendar s prilagoditvami za nižjo zakasnitev in računsko zahtevnost. JPEG XS ima zakasnitev največ 32 vrstic slike, kar doseže z uporabo delne diskretne valčne transformacije, tako da izvede manj delitev v navpični smeri. Posebno pozornost pri razvoju so namenili podpori za natančen nadzor bitne hitrosti.

3.2.3.15 JPEG XL

JPEG XL je standardizirana (ISO/IEC 18181-1 [169]) nadgradnja JPEG, katere cilj je bil, da ga tudi v celoti nadomesti. Omogoča tako izgubno kot brezizgubno stiskanje. S stopnjo stiskanja občutno presega JPEG in PNG [170]. Podpira različne barvne modele (RGB, CMYK), več barvnih kanalov, prosojnost, animacije, do 32 bitov na barvni kanal, ploščice, 360° stopinjske slike, napredujoče stiskanje in celo možnost zapisa vrednosti pikslov z decimalnimi števili. Koraki izgubnega stiskanja so naslednji [170]:

1. Pretvorba v barvni prostor YCbCr.
2. Zaznava značilnosti na sliki (na primer šum, krivulje, ponavljajoči vzorci), ki jih stisne ločeno.
3. Izvedba **VarDCT**, torej DCT z bloki spremenljive velikosti od 2×2 do 256×256.
4. Adaptivna kvantizacija koeficientov DCT.
5. Uporaba napovednega modela za zapis koeficientov DCT, kar je bistvena novost.
6. Kodiranje entropije s kombinacijo L777, ANS (angl. Asymmetric Numeral System) [34] ali Huffmanovega kodirnika.

Zaporedje korakov pri brezizgubnem stiskanju je podobno. Glavna razlika je v zamenjavi VarDCT z drugimi transformacijami, kot je Haarova transformacija, in izločitvi napredujočega stiskanja.

Avtorji JPEG XL so posebno pozornost namenili združljivosti z originalnim JPEG. JPEG XL omogoča brezizgubno pretvorbo iz JPEG, saj lahko deluje tudi s koeficienti DCT 8×8 iz JPEG. Uporaba prej omenjenih dodatnih tehnik stiskanja omogoča od 16 do 22 % manjšo velikost dobljenih datotek [170].

3.2.3.16 JPEG Pleno in AI

Med nadgradnjami JPEG je treba omeniti še JPEG Pleno, ki predstavlja ogrodje za predstavitev novih vizualnih modalnosti, kot so 3D-teksture, oblaki točk in holografske slike. Plod skupine JPEG Pleno je več standardov. Objavljeni standardi od ISO/IEC 21794-1 do ISO/IEC 21794-3 [171, 172, 173] predpisujejo postopke shranjevanja vrednosti **svetlobnega polja** fotoaparata (angl. light field/plenoptic camera). Standard ISO/IEC

21794-5 [174] predpisuje postopke kodiranja holografskih slik. Prihajajoči standard ISO/IEC DIS 21794-6 [175] je predviden za oblake točk.

JPEG AI [176] je bodoči standard ISO/IEC DIS 6048-1 [177] za kodiranje slik na podlagi metod strojnega učenja. Pri tem se ne omejuje zgolj na slike, temveč tudi na druge modalnosti, kot so oblaki točk. Skupina JPEG je izdala še več standardov, ki pa presegajo ta učbenik. Eden izmed primerov je ISO/IEC 19566-6 [178], ki predpisuje način hrambe 360-stopinjskih slik znotraj obstoječih standardov JPEG.

3.2.3.17 HEIF

Z razvojem WebP se je dokazal smisel uporabe postopkov iz kodiranja videa, ki jih bomo spoznali v naslednjih poglavjih, tudi za slike. Tudi format HEIF (angl. High Efficiency Image File Format), ki je standardiziran vsebnik ISO/IEC 23008-12 [179], hrani slike z uporabo postopkov stiskanja videa; najpogosteje HEVC/H.265 (angl. High Efficiency Video Coding). V tem primeru govorimo o datotekah HEIC ali HEIF. Dosežene stopnje stiskanja so primerljive z JPEG XL. Vsebnik HEIF omogoča uporabo tudi starejšega postopka stiskanja videa AVC/H.264 (angl. Advanced Video Coding), pri čemer dobljene datoteke označujemo z AVCI. Podobno kot pri večini novejših datotečnih formatov patenti upočasnjujejo razširjenost novejših postopkov stiskanja.

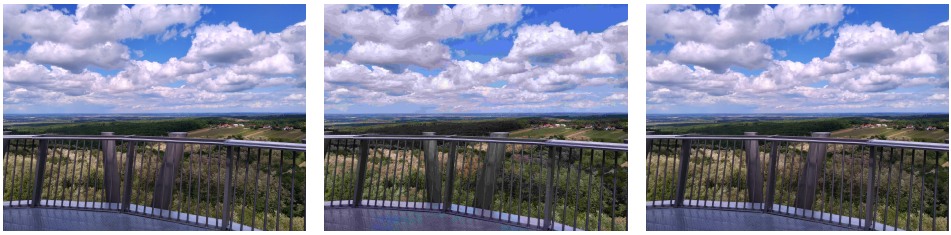
3.2.3.18 AVIF

V letu 2019 je organizacija Alliance for Open Media izdala odprto in patentovprosto specifikacijo formata AVIF (angl. AV1 Image File Format). Kot lahko razberemo iz imena, gre za uporabo postopka stiskanja videa AV1 znotraj vsebnika HEIF. Zaradi odprtosti specifikacije je ta format zelo dobro podprt v spletnih brskalnikih in drugih orodjih za obdelavo slik.

AVIF omogoča različne barvne prostore, profile ICC, od 8- do 12-bitne barvne kanale, prosojnost, animacije. Slike lahko razdeli v ploščice, stiskanje pa je lahko tako izgubno kot brezizgubno, ki je primerljivo z JPEG XL in HEIF. Sam postopek stiskanja uporablja napovedi, različno velike bloke za DCT in možnost uporabe **diskretne sinusne transformacije** (DST, angl. Discrete sine transform). Morebitne artifakte na robovih blokov zmanjšuje s filtriranjem. Za stiskanje entropije uporablja aritmetični kodirnik [180]. Med zanimivostmi datotečnega formata AVIF je odstranjevanje šuma za ločeno stiskanje preostanka slike in potem vračanje šuma v sliko oziroma **sinteza šuma** (angl. film grain synthesis).

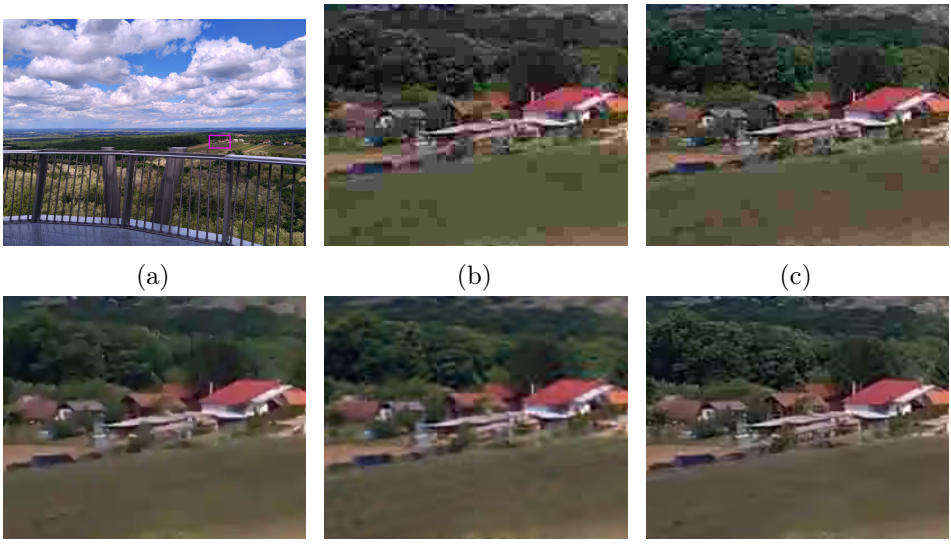
Napredek pri razvoju modernejših datotečnih formatov glede na JPEG

je zelo opazen. Na sliki 3.19 vidimo fotografijo, ki je močno stisnjena z JPEG in z AVIF. Opazimo, da izgub na sliki, stisnjeni z AVIF, ni več opaziti. Napake pri stiskanju opazimo šele ob pogledu od blizu na sliki 3.20, kjer vidimo, da so novejši formati WebP, JPEG XL in AVIF občutno boljši.



(a) (b) (c)

Slika 3.19: Slika ločljivosti 4000 × 3000 (a) neposredno s fotoaparata, (b) stisnjena z JPEG na 0,13 bpp in (c) stisnjena z AVIF na 0,13 bpp (vir slike: lastni).



(a) (b) (c)

(d) (e) (f)

Slika 3.20: (a) Izbran del slike ločljivosti 4000 × 3000, ki je stisnjena na 0,13 bpp z (b) JPEG in Huffmanovim kodirnikom, (c) JPEG in aritmetičnim kodirnikom, (d) WebP, (e) JPEG XL in (f) AVIF (vir slike: lastni).

3.2.4 Izbrani algoritmi nad rastrskimi slikami

V preteklosti je bila razvita nepregledna množica algoritmov za obdelavo rastrskih slik. Mnogo teh algoritmov ste spoznali pri drugih predmetih. V nadaljevanju si bomo ogledali še nekatere.

3.2.4.1 Izračun podobnosti med slikami

Ključno vprašanje pri izgubnem stiskanju je, ali bo človeško oko izgube zaznalo. Ocena *po občutku* je zelo odvisna od posameznika. Zato želimo metriko, ki bi izgube ocenila kvantitativno. Naloga je več kot zahtevna, saj karakteristik človeške vidne zaznave ne znamo formalno opisati. Za primerjavo izvorne slike X in stisnjene slike Y zato najpogosteje uporabljamo dve metriki.

3.2.4.1.1 Metrika PSNR

Metrika PSNR (angl. peak signal-to-noise ratio) prihaja iz sveta telekomunikacij in določa razmerje med maksimalno močjo signala v primerjavi s signalom, podvrženem šumu in slabljenju v komunikacijskem signalu. PSNR merimo v dB in jo nad slikama oziroma zaporedjema vrednosti pikslov X in Y izračunamo z enačbo 3.12.

$$PSNR(X,Y) = 20\log_{10}\frac{MAX}{RMSE(X,Y)}, \tag{3.12}$$

kjer je MAX maksimalna jakost signala in je najpogosteje enaka $2^{bpp} - 1$, $RMSE$ pa je kvadratni koren povprečne vrednosti napake, določen z enačbo 3.13.

$$RMSE(X,Y) = \sqrt{MSE(X,Y)} \tag{3.13}$$

Povprečno vrednost napake med slikama X in Y dobimo iz enačbe 3.14.

$$MSE(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(X_i - Y_i)^2, \tag{3.14}$$

kjer je n število pikslov na sliki.

Pri barvnih slikah uporabimo barvno globino komponente svetilnosti. Če sta sliki enaki, je $RMSE = MSE = 0$, $PSNR$ pa je ∞, ali bolje, nedefiniran. Majhne vrednosti $PSNR$ nam torej povedo, da se sliki X in Y razlikujeta, nič pa ne izvemo, ali bo to razliko zaznalo tudi človeško oko. Izkustvene

vrednosti $PSNR \in [20, 40]$ dajo sliko, ki jo človek oceni kot primerno, to je takšno z dovolj malo izgubami.

3.2.4.1.2 Metrika SSIM

Ker metrika PSNR ni prilagojena vidni zaznavi, jo moramo uporabljati zelo previdno. Njena uporaba je dejansko upravičena samo v primeru, ko primerjamo rezultate iste metode pri različnih stopnjah stiskanja [181]. **Metriko indeksa strukturne podobnosti** (angl. Structural Similarity Index Metric – SSIM) pa so razvili za to, da bi dobili oceno vizualne podobnosti dveh slik, ki bi bila skladna s človeško oceno [182]. V splošnem SSIM izračunamo za poljubno okno slike. Naj bo $X \subseteq A$ okno slike A in $Y \subseteq B$ istoležno okno slike B. Velikosti obeh oken sta $m \times n$, pri čemer sta m in n najpogosteje enaka 8 ali 11 [182]. Metriko SSIM izračunamo z enačbo 3.15.

$$SSIM(X, Y) = \frac{(2\mu(X)\mu(Y) + c_1)(2\sigma(X, Y) + c_2)}{(\mu(X)^2 + \mu(Y)^2 + c_1)(\sigma(X)^2 + \sigma(Y)^2 + c_2)}, \tag{3.15}$$

kjer so:

- $\mu(X)$: povprečje vrednosti v oknu X,
- $\mu(Y)$: povprečje vrednosti v oknu Y,
- $\sigma(X)$: varianca vrednosti v oknu X,
- $\sigma(Y)$: varianca vrednosti v oknu Y,
- $\sigma(X, Y)$: kovarianca X in Y,
- c_1 in c_2: spremenljivki (izračunamo ju kot $c_1 = (k_1 MAX)^2$ in $c_2 = (k_2 MAX)^2$), s katerima stabiliziramo deljenje pri majhnem imenovalcu, določeni kot:
  - MAX: največja možna vrednost v oknu (tipično $2^{bpp} - 1$);
  - $k_1 = 0{,}01$ in $k_2 = 0{,}03$.

Metrika SSIM daje najboljše rezultate, če jo uporabimo nad komponento svetilnosti barvnega modela YCbCr. Metrika vrne vrednosti iz intervala $[-1, 1]$, kjer vrednost 1 dosežemo pri identičnih vrednostih v oknih.

Za primerjavo izgub v slikah so predlagali še množico drugih metrik podobnosti, na primer MS-SSIM (angl. Multi-Scale SSIM) [183], VIF (angl. visual information fidelity) [184], NIQE (angl. natural image quality evaluator) [185] in mnoge druge [186, 187].

3.2.4.2 Kvantizacija barv

Čeprav je za človekove vidne zaznave zelo ugodno prikazati veliko barv, pa je v nekaterih aplikacijah treba število barv zmanjšati, na primer:

- Izhodna naprava deluje samo z omejenim številom barv.
- Število barv želimo zmanjšati za to, da bi zmanjšali velikost informacije, ki opisuje sliko. Da lahko to uspešno storimo, vidimo na sliki 3.21.

(a) (b) (c) (d)

Slika 3.21: Primer zmanjševanja barv: (a) original z 2^{24} barvami, (b) 256 barv, (c) 16 barv in (d) 4 barve (avtor izvorne slike: Miran Kambič)

Najenostavnejši način za pretvorbo 24-bitne slike v 8-bitno je, da za rdečo in zeleno barvo uporabimo po 3 bite, preostala 2 bita pa za modro barvo (modro barvo namreč najslabše zaznavamo). Tako 256 različnih barvnih odtenkov posamezne barvne komponente kvantiziramo v kombinacijo osmih rdečih in zelenih ter štirih modrih barv. Metoda pa žal za vse slike ne daje dobrih rezultatov. Če imamo sliko z veliko odtenki ene barve, bo rezultat zelo slab. Zato je veliko bolje za vsako sliko izbrati najprimernejši nabor barv in jih zapisati v, kot že vemo, barvno vpogledno tabelo (angl. look-up table – LUT). Problem kvantizacije barv potem razpade v dva podproblema:

- izbrati najustreznejše barve za dano sliko in
- barvi, ki ni bila izbrana, najti najbolj podobno izbrano barvo.

Obe nalogi rešujemo povezano. Omenimo najpogostejše pristope:

1. **Algoritem gručenja** (angl. clustering algorithm) **k voditeljev** (angl. k-means) daje dobre rezultate. Vsako izmed barv predstavimo kot točko v 3D-prostoru RGB, določimo želeno število gruč (na primer 256), nato pa v iterativnem postopku poiščemo centre gruč, ki s tem določajo tudi vrednost barv v LUT. S tem rešimo tudi drugi podproblem, saj so vse barve, ki niso centri gruč, najbližje samo enemu centru/barvi. V [188] najdemo množico izboljšav te osnovne metode.

2. **Algoritem popularnosti** (angl. popularity algorithm) razdeli prostor barv v množico enako velikih celic, na primer kock velikosti $64 \times 64 \times 64$ [189]. S tem barvni prostor iz 16.777.216 diskretnih vrednosti razdelimo na še vedno veliko 262.144 regij. V vpogledno tabelo vključimo 256 najbolj zasedenih celic. Celicam/barvam, ki niso bile izbrane, dodelimo najbližjo celico glede na prostorsko razdaljo.

3. **Algoritem rezanja po mediani** (angl. median cut algorithm) rekurzivno deli prostor barv glede na porazdelitev vhodne množice barv [190]. Algoritem poteka v naslednjih korakih:

   (a) Poiščemo oklepajočo škatlo točk, ki predstavljajo barve (običajno delamo v 3D-barvnem prostoru RGB).

   (b) Uredimo točke vzdolž najdaljše osi oklepajoče škatle in izračunamo mediano.

   (c) Razdelimo škatlo v dve podškatli glede na mediano.

   (d) Proces deljenja ponavljamo vedno v največji škatli, dokler ne razdelimo barvnega prostora na želeno število regij.

   Reprezentativne barve dobimo s povprečenjem barv v vsaki škatli.

4. **Algoritem z osmiškim drevesom** (angl. octree algorithm) [191] vsako barvo iz slike shrani v osmiško drevo globine 8, ki ima natanko $8^8 = 16.777.216$ listov, kar ustreza številu vseh možnih barv barvnega modela RGB z 8-bitno barvno globino. V nekaterih listih bo več pikslov, mnogo listov bo tudi praznih. Drevo zdaj obrežemo (angl. tree pruning) tako, da odstranjujemo vozlišča, katerih potomci so prazni. Če ostane več kot K vozlišč (K je običajno 256), jih združujemo. Pri tem lahko uporabimo različne kriterije. Na primer:

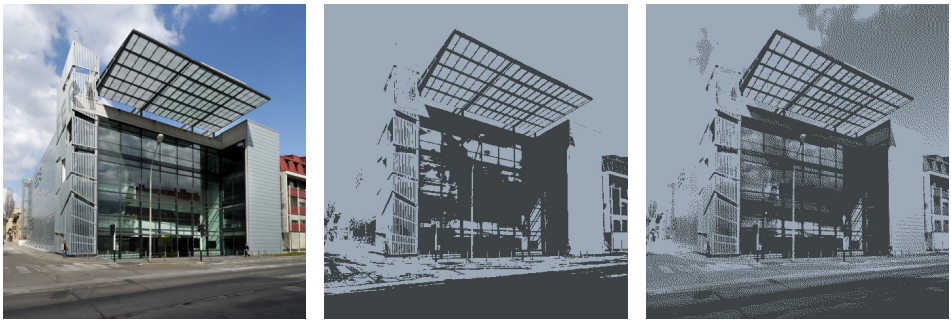
   - združujemo celice, ki hranijo najmanj barv,
   - združimo celice, ki izhajajo iz nadrejenega vozlišča in imajo največ barv (tako bomo sicer večja področja zapolnili z eno barvo, ohranili pa bomo ostrino detajlov).

Novejši postopki temeljijo na optimizacijskih algoritmih, pregled najdemo v [192].

3.2.4.3 Stresanje

Slabost barvne kvantizacije so opazno slabša kakovost in vidne barvne proge (angl. color banding). Stresanje je izjemno popularna tehnika, ki zmanjšuje

vizualno zaznavo napake po kvantizaciji. Uporabna je tako pri slikah kot pri zvoku in videu. Tehnike se spomnimo iz starih časopisov z majhnim številom barv; v začetku sta bili na voljo samo dve barvi (črna in bela). V takšni tehniki je prikaz slik z več odtenki na prvi pogled nemogoč. A kot vidimo na sliki 3.22, lahko pričaramo odtenke barv s konstrukcijo vzorcev pikslov.



(a) (b) (c)

Slika 3.22: (a) Originalna slika 3.7a, (b) barvna kvantizacija na dve barvi in (c) uporaba stresanja med barvno kvantizacijo

Najbolj pogosto uporabljan algoritem sta razvila Floyd in Steinberg [193]. Temeljna ideja algoritma je, da **porazdeli** napako kvantizacije barv na okoliške piksle. Algoritem izvajamo nad posameznimi piksli po prebirnem vrstnem redu v naslednjih korakih:

1. Za piksel iz izvorne slike določimo najbližjo vrednost v barvni paleti.
2. Izračunamo napako med kvantizirano vrednostjo in vrednostjo izvornega piksla (če imamo barvno sliko, izračunamo napako za vsako barvno komponento posebej).
3. Dobljeno napako porazdelimo (prištejemo) na sosednje piksle, ki še nisi bili obdelani (piksle desno in spodaj glede na obravnavan piksel x) s predpisanimi utežmi (slika 3.23).

   Korigirane vrednosti pikslov uporabimo v naslednjih korakih kvantizacije.

Oglejmo si primer. 256 vrednosti sivin enakomerno kvantizirajmo na 8 barv, torej v barvni paleti dobimo 8 vnosov, kot vidimo v tabeli 3.2.
Zaporedje pikslov vhodne slike vidimo na levi strani slike 3.24, rezultat stresanja nad drugim pikslom pa na desni strani. Za trenutni piksel z

$$\begin{array}{ccc} & x & \frac{7}{16} \\ \frac{3}{16} & \frac{5}{16} & \frac{1}{16} \end{array}$$

Slika 3.23: Floyd-Steinbergove uteži porazdelitve vrednosti napake okrog piksla x

Tabela 3.2: Barvna vpogledna tabela (primer)

vrednosti	16	48	80	112	144	176	208	240

vrednostjo 20 poiščemo najbližjo vrednost v preslikovalni tabeli. Ta vrednost je 16. Izračunamo napako med kvantizirano in originalno vrednostjo: $\epsilon = 20 - 16 = 4$. S kvantizacijo smo v tem primeru sliko potemnili, zato moramo svetlost dodati oziroma razpršiti na okoliške piksle. Tako ϵ uporabimo za korekcijo ostalih pikslov s podanimi utežmi. Desni piksel korigiramo na naslednji način: $32 + \epsilon \times \frac{7}{16} = 33{,}75 = 34$. Preostale vrednosti izračunamo na enak način, le da uporabimo druge uteži. V naslednji iteraciji algoritma bi se pomaknili na tretji piksel. Tokrat bi tretji piksel kvantizirali na vrednost 48, kar bi predstavljalo napako -14, ki bi jo spet morali razpršiti na sosednje piksle. Na takšen način bi postopek ponavljali do zadnjega piksla.

$$\begin{array}{ccc} 0 & \underline{20} & 32 \\ 10 & 10 & 10 \end{array} \qquad\qquad \begin{array}{ccc} 0 & \underline{16} & 34 \\ 11 & 11 & 10 \end{array}$$

Slika 3.24: Vrednosti pikslov pred razpršitvijo (levo) in po (desno) razpršitvi napake drugega piksla s Floyd-Steinbergovim algoritmom

Poleg Floyd-Steinbergovega algoritma poznamo še množico drugih algoritmov [3, 194, 195]. Metoda od avtorjev **Jarvis, Judice in Ninke** [3] je zelo podobna predhodni metodi, a razpršitev napake opravi nad širšo okolico, kot vidimo na sliki 3.25.

Da področje še zdaleč ni pozabljeno, dokazuje množica patentov [196, 197, 198].

3.2.4.4 Steganografija v rastrskih slikah

Rastrske slike so zaradi informacijske redundantnosti zelo primeren nosilec prikritega sporočila. Razvitih je bilo zelo veliko metod, ki jih delimo v tri

$$\begin{array}{ccccc} & & x & \frac{7}{48} & \frac{5}{48} \\ \frac{3}{48} & \frac{5}{48} & \frac{7}{48} & \frac{5}{48} & \frac{3}{48} \\ \frac{1}{48} & \frac{3}{48} & \frac{5}{48} & \frac{3}{48} & \frac{1}{48} \end{array}$$

Slika 3.25: Uteži porazdelitve vrednosti napake okrog piksla x pri metodi Jarvis, Judice in Ninke [3]

skupine: naivne metode, metode, ki delujejo v **prostorski domeni** (angl. spatial domain techniques), in tiste, ki delujejo v **prostoru transformacij** oziroma v **frekvenčnem prostoru** (angl. transformation/frequency domain techniques). Slednje so namenjene slikam, ki so stisnjene z uporabo transformacij JPEG in JPEG2000. Za njihovo razumevanje je potrebno zelo dobro poznavanje teh transformacij, zato si teh tehnik ne bomo ogledali. Raje se bomo osredotočili na tehnike, ki delujejo v prostorski/barvni domeni. Obširen pregled dosedanjih rešitev najdemo v [199].

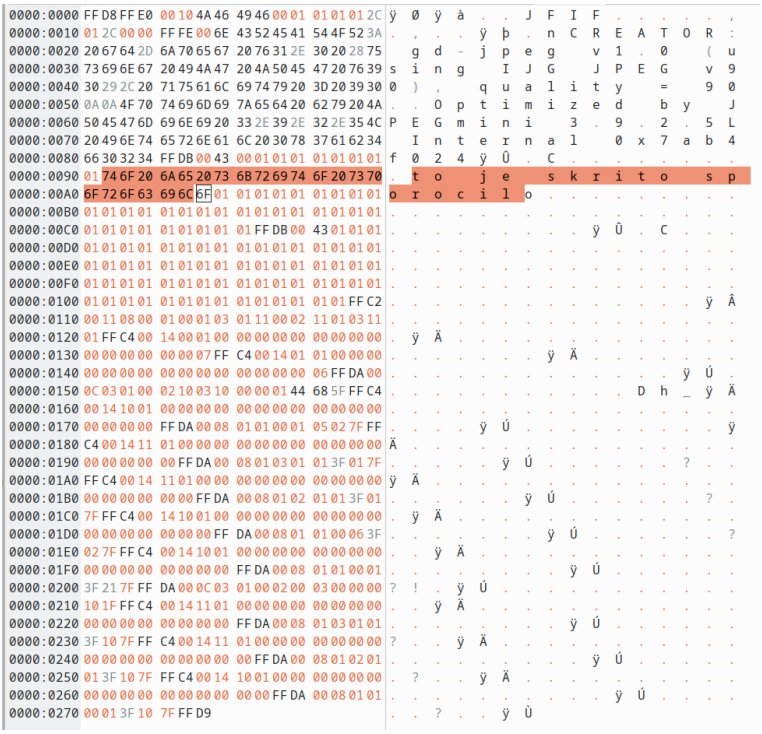
3.2.4.4.1 Naivne metode

Najpreprostejšo metodo lahko preprosto realiziramo kar z operacijskim sistemom Windows z naslednjim ukazom:

c:>copy testImage.jpg/b+Sporocilo.txt/b Stego.jpg .

Ukaz združi datoteko s sliko *testImage.jpg* in sporočilom, zapisanim v datoteki *Sporocilo.txt*, kar tvori enotno datoteko *Stego.jpg*. Ukaz *copy* prepiše oznako $< EOF >$ na koncu datoteke .jpeg in ji doda besedilo iz navedene datoteke. Sliko bomo še vedno lahko pogledali v vsakem programu za prikaz slik. Če pa bomo sliko včitali z urejevalnikom ASCII (na primer Notepad), bomo na dnu lahko prebrali sporočilo. S tem pristopom je slika povsem nespremenjena, uporabnik pa ne dobi nobene informacije, da je aplikacija prebrala samo del datoteke. Ključna slabost je seveda v spremembi velikosti datoteke, kar pazljiv uporabnik hitro ugotovi (a pazljivih uporabnikov je zelo malo).

Naslednja naivna metoda je vpis prikritega sporočila v glavo slikovne datoteke, ki je pogosto nekoliko redundantna. Primer na sliki 3.26 kaže vpisano sporočilo v glavo datoteke JFIF. Podobno lahko naredimo s slikami PNG, kjer ustvarimo nov poljuben skupek ali podatke zapišemo v skupek iTXt. Alternativna možnost je vpis skritega sporočila v EXIF ali XMP.



Slika 3.26: Vpis prikritega sporočila v glavo datoteke JFIF

3.2.4.4.2 Metoda LSB

Najenostavnejša steganografska metoda je metoda LSB [200]. Za zapis prikritega sporočila uporabi najmanj pomemben bit v vsakem barvnem kanalu. Pri 24-bitnih slikah tako vsak piksel prikrije 3 bite sporočila. Vizualne spremembe slike so očem praktično nevidne. Pri krajših prikritih sporočilih piksle izbiramo po vnaprej določenem mehanizmu, druge pa uporabljajo tiste piksle, kjer je morebitna sprememba vrednosti LSB najmanj opazna (na primer na robovih, v teksturi). Da bi povečali kapaciteto, je s pazljivo izbiro pikslov možno povečati število bitov za zapis prikritega sporočila do 4. V literaturi najdemo množico izboljšav metode LSB [199].

3.2.4.4.3 Metoda PVD

Metoda LSB je preprosta in zelo fleksibilna, njega glavna slabost pa je, da če povečamo število najmanj pomembnih bitov piksla za zapis prikritega sporočila, hitro opazno poslabšamo stegosliko. Da bi to pomanjkljivost zmanjšali, so predlagali metodo PVD (angl. pixel value difference) [201].

Odločitev, koliko bitov LSB lahko uporabimo za zapis prikritega sporočila, je odvisna od razlike dveh sosednjih pikslov. Piksle izvorne slike uredimo v neskanirnem vrstnem redu, na primer izmenjajoče levo-desno prebiranje. Sosede pa obravnavamo kot pare pikslov. Večja kot je razlika med sosednjima piksloma, več bitov lahko uporabimo. Tudi metoda PVD je doživela množico izboljšav in nadgradenj [199].

3.2.4.4.4 Metoda GLM

Metoda GLM (angl. grey level modification) razširja metodo LSB tako, da najprej z vnaprej določeno funkcijo izbere piksle v sliki, v katere bo zapisano sporočilo [202]. Nato preveri vrednosti v pikslih in lihe vrednosti inkrementira, da postanejo sode. Vpis bitov prikritega sporočila je sedaj preprost. Če vpisujemo bit 0, vrednosti piksla ne spremenimo, če pa vpisujemo lih bit, pa vrednost v bitu dekrementiramo. Razširitev na barvne slike je trivialna [203].

3.2.4.4.5 Napovedna metoda

Neposredno spreminjanje pikslov ima dve slabosti: očitna sprememba slike in relativno majhna kapaciteta za vnos prikritega sporočila. To težavo nekoliko omilijo napovedne metode (angl. pixel value prediction) [4]. Napoved barve naslednjega piksla je pogosto uporabljena pri stiskanju rastrskih slik brez izgub (JPEG-LS, LOCO-I, FELICS, JBIG) in v nekaterih primerih tudi pri izgubnem stiskanju. Za modeliranje napovedi poznamo različne sheme. Napovedna metoda za vpis prikritega sporočila je neodvisna od napovednega modela, saj uporablja samo rezultat le-tega; to je, napako, e, med napovedano in dejansko vrednostjo piksla. Vpis prikritih pikslov opravimo na način, kot je opisan v originalnem članku [4]:

- Če je $|e| \mod 2^h = \xi$ oziroma če so LSB biti prikritega sporočila enaki $|e|$, potem e ne spremenimo, pri čemer:
  - h določa število bitov/piksel za vpis prikritega sporočila,
  - ξ pa je vrednost prikritega sporočila, ki ga želimo vpisati v piksel.
- V nasprotnem primeru napako e spremenimo. Najprej izračunamo novo napako $\hat{e}$ z enačbo 3.16, s čimer bite LSB napake postavimo na prikrito sporočilo.

$$\hat{e} = |e| + (\xi - |e|) \mod 2^h. \tag{3.16}$$

Če $|e+\hat{e}| < |e-\hat{e}|$ potem je vrednost napake $e = -\hat{e}$, sicer pa ostane $\hat{e}$.

Primer povzamemo po [4]. Na sliki 3.27a je zaporedje petih vrednosti pikslov, napako med dejansko in napovedano vrednostjo pa vidimo na sliki 3.27b. Biti, ki jih želimo vpisati, so na sliki 3.27c. Najprej predpostavimo $h = 1$ bit/piksel. V prvem primeru je $e = 0$, vrednost, ki jo vstavljamo, pa je $\xi = 1$. Ker $|e| \mod 2 = 0 \neq \xi$, izračunamo enačbo 3.16. Dobimo $\hat{e} = 0 + (1 - 0 \mod 2) = 1$. $|e + \hat{e}| = 1$, kar ni manj od $|e - \hat{e}| = 1$, zato je $\hat{e} = 1$.

V drugem primeru je $e = 2$ in $|e| \mod 2 = 0$. Ker je $\xi = 1$, nadaljujemo z izračunom enačbe 3.16. Dobimo $\hat{e} = 2+(1-2) \mod 2 = 3$. Ker $|2+3| = 5$ ni manjše od $|2 - 3| = 1$, je modificirana vrednost napake $\hat{e} = 3$. Preostale vrednosti izračunamo na enak način.

```
155 157 157 158 159          0 2 2 1 2
a)                           b)

1  1  0  0  1                1 3 2 0 3
c)                           d)

01 11 10 00 01               1 3 2 0 1
e)                           f)
```

Slika 3.27: Vpis prikritega sporočila z napovedno metodo [4]

Oglejmo si še primer, ko $h = 2$. Bite prikritega sporočila, ki jih želimo vpisati, vidimo na sliki 3.27e. V prvem primeru je $e = 0$ in $|0| \mod 2^2 = 0 \neq \xi = (01)_2$. Izračunamo $\hat{e} = 0 + (1 - 0) \mod 4 = 1$. $|e - \hat{e}| = 1 \leq |e + \hat{e}| = 1$, zato ostane $\hat{e} = 1$. V drugem primeru je $e = 2$, vrednost, ki jo vstavljamo, pa $\xi = 11_2 = 3$. Ker $|2| \mod 4 = 2 \neq \xi$, po enačbi 3.16 dobimo $\hat{e} = 3$. $|e - \hat{e}| = |2 - 3| = 1 \leq |e + \hat{e}| = |2 + 3| = 5$, ostane $\hat{e} = 3$. Za vajo lahko izračunamo še preostale vrednosti $\hat{e}$ in jih preverimo na sliki 3.27f. Ta slika dejansko kaže vso moč metode. Čeprav smo povečali število prikritih bitov za enkrat glede na prejšnji primer, se korigirane vrednosti napak niso bistveno spremenile (primerjajmo sliki 3.27d in f). Reference na druge metode, temelječe na napovedi, najdemo v [203]. Pri uporabi teh metod pa moramo paziti na fenomen **akumulacije napak**, saj so vrednosti pikslov dejansko relativne glede na predhodne vrednosti. Kakršnakoli sprememba piksla vpliva na vrednosti vseh nadaljnjih pikslov.

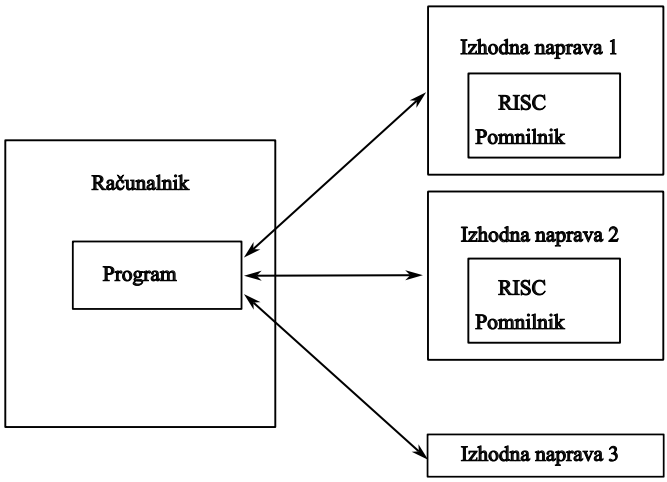
3.3 Vektorske slike

V nasprotju z rastrskimi slikami, kjer je gradnik slike le eden, to je točka oziroma piksel, so gradniki vektorskih slik krivulje prvega (daljice) ali drugega reda (**stožnice** – krožnice, elipse, redkeje parabole in hiperbole, torej vse, kar lahko predstavimo s presekom dvojnega stožca in ravnine). Namesto krivulj višjih redov v računalništvu (oziroma v računalniški grafiki) praviloma uporabljamo **zlepke** (izjema so le fraktalne krivulje). Zlepki (angl. spline) so pravzaprav tudi krivulje višjih redov (najpogosteje so kubične), ki pa jih znamo med seboj zvezno povezovati. Razvitih je bilo veliko metod (Hermitova, Bézierova, B-zlepki, NURBS), ki spadajo na področje računalniške grafike.

Bistvo gradnikov vektorskih slik je v tem, da je njihova predstavitev analogna in s tem neodvisna od ločljivosti izhodne naprave. Za prikaz na prikazovalniku zato potrebujemo pretvorbo med funkcijskim opisom gradnikov in rastrskimi napravami – **rasterizacijo**. Slika 3.28 demonstrira tipičen scenarij. Na računalnik so povezane različne izhodne naprave, ki se operacijskemu sistemu predstavijo z uporabo svojega krmilnega programa/gonilnika. Recimo, *izhodna naprava 1* je brizgalni tiskalnik z ločljivostjo 118,11 piksla/cm, *izhodna naprava 2* je laserski tiskalnik z ločljivostjo 472,44 piksla/cm, *izhodna naprava 3* pa je prikazovalnik z ločljivostjo 1920×1080 pikslov. Na vseh napravah bi seveda radi prikazali vektorsko sliko na najboljši način, ki ga izhodna naprava zmore. Če brizgalni tiskalnik uporablja papir velikosti A4, je število pikslov, ki jih lahko naslovi, 2480×3507 pikslov, laserski tiskalnik lahko nariše 9921×14031 pikslov, prikazovalnik pa seveda ostane pri svoji nazivni ločljivosti. Pri večjem papirju je število pikslov lahko mnogo večje. Danes so tiskalniki zelo pogosto opremljeni s procesorji, ki opravijo rasterizacijo in s tem razbremenijo računalnik, da bi za vsako stran dokumenta opravljal to računsko dokaj naporno nalogo. Namesto tega sprejmejo parametričen opis vsakega znaka za celotno stran besedila, znake rasterizirajo in jih nato prikažejo.

Nad gradniki, ki so opisani s funkcijami, lahko opravljamo najrazličnejše matematične transformacije; najpogostejše so pomik (angl. translation), vrtenje (angl. rotation) in povečevanje/pomanjševanje (angl. scaling, zoom). Tudi pretvorbo iz lokalnega koordinatnega sistema posameznega gradnika v prostor rastrske naprave opravimo s transformacijami. Objekte, ki jih zgradimo z vektorskimi gradniki, lahko opišemo tudi s parametri, s čimer močno povečamo njihovo ponovno uporabljivost. Prav tako omogočimo nelinearno spreminjanje oblik glede na, na primer, povečevanje. Tipična uporaba je pri načrtovanju in predstavitvi pisav.

Uporabnost vektorske predstavitve so hitro spoznali v inženirstvu. Vsi



Slika 3.28: Računalniška konfiguracija z več izhodnimi napravami

inženirski risalni paketi (AutoCAD, QCAD, DraftSight, NanoCAD in mnogi drugi) so vektorski. Nekoliko manj inženirsko orientirani vektorski programski paketi, ki jih bolj poznamo, so CorelDraw, Inkscape, Gravit, Vectr, LibreOffice Draw, sK1 in mnogi drugi.

3.3.1 Formati vektorskih slik

Tudi za vektorske slike obstaja več kot 100 formatov [204]. Predvsem v okolju za **računalniško podprto načrtovanje** (angl. Computer-aided design – **CAD**) je še kako pomembno, da bi bili formati izmenljivi. Namreč, proizvajalci omogočajo različne gradnike, tudi interni opis gradnikov je lahko drugačen (na primer parametrična ali eksplicitna oblika). Formati CAD poleg geometrijskih informacij vsebujejo še množico drugih podatkov, potrebnih za izdelavo izdelka (na primer tolerance, material, kakovost obdelave, termične lastnosti ipd.), katerih opis tudi ni poenoten. Prav zato je razvoj nevtralnega formata, ki bi ga znali prebrati in pravilno interpretirati programski paketi različnih proizvajalcev, še kako pomemben. Prva takšna specifikacija je bil IGES (angl. Initial Graphics Exchange Specification), ki ga je v sedemdesetih letih sprejel ANSI oziroma NBSIR (angl. National Bureau of Standards). ISO je leta 1994 sprejel STEP (angl. STandard for the Exchange of Product model data) (ISO 10303 [205]). S pojavom STEP je razvoj IGES zamrl. STEP se še zmeraj razvija in dopolnjuje. Z več kot 200 razširitvami je celo največji standard pod okriljem ISO.

V osemdesetih in devetdesetih letih so tudi računalniška podjetja začela razvijati splošno-namenske vektorske formate. Tako je Microsoft v devetdese-

tih razvil vektorski format WMF (angl. Windows Metafile) [105], ki dejansko predpisuje uporabo ukazov iz aplikacijskega programskega vmesnika GDI, ki se uporablja za izris znotraj operacijskega sistema Windows. Z razvojem 32-bitnih operacijskih sistemov je sledila nadgradnja na EMF, s pojavom aplikacijskega programskega vmesnika GDI+ pa EMF+.

V osemdesetih letih je Adobe razvil PostScript za opis strani, zato je bil razvoj formata za hrambo vektorskih slik EPS (angl. Encapsulated PostScript) logična izbira. Kot že ime pove, gre za zapis PostScript, ki je omejen na eno sliko znotraj vnaprej definirane obrobe. Na podlagi EPS je Adobe pozneje razvil datotečni format AI za programsko opremo Adobe Illustrator Artwork in datotečni format PDF. Bistvo pri tem formatu je dobra podpora za vektorske gradnike in rastrske slike z različnimi načini stiskanja, kot so RLE, LZW, JPEG in JPEG2000.

Z razširjenostjo svetovnega spleta se je leta 1999 pojavil format SVG, ki si ga bomo na kratko ogledali v nadaljevanju. Po tem letu so se pojavili tudi datotečni formati za pisarniške aplikacije, ki omogočajo risanje z vektorskimi gradniki. Format ODG (angl. OpenDocument Graphics, ISO/IEC 26300 [64]) je nastal leta 2006 in se danes uporablja v LibreOffice Draw.

3.3.1.1 Format SVG

SVG (angl. Scalable Vector Graphics) je odprt standard na podlagi značk XML, ki ga je razvil W3C (angl. World Wide Web Consortium). Čeprav je bil glavni cilj podpora vektorski grafiki na spletu (interpretirajo ga vsi popularnejši brskalniki), je postal tudi neformalni nevtralni izmenjevalni format različnih vektorskih risarskih programov (CorelDraw in Inkscape imata zanj zelo dobro podporo), saj je platformsko in programsko neodvisen.

Preprost primer, ki nariše krožnico s polmerom 50, središčem kot zelen krogec s polmerom 5 in vodoravno črto, ki gre skozi središče krožnice, vidimo v izpisu 2.

Psevdokod 2: Izris s SVG

```
1: <svg width="140" height="170" xmlns="http://www.w3.org/2000/svg">
2: <title>TEST</title>;
3: <desc>Primer SVG</desc>;
4: <circle cx="70" cy="95" r="50" style="stroke: black; fill: none"/>
5: <circle cx="70" cy="95" r="5" stroke="black" stroke-width="1" fill="green"/>
6: <line x1="20" y1="95" x2="120" y2="95" style="stroke:rgb(255,0,0);stroke-
   width:2"/>
7: </svg>
```

Datoteke SVG je možno stisniti z algoritmom Deflate. V tem primeru datoteke dobijo končnico SVGZ. SVG vključuje naslednje gradnike: daljice,

lomljenke, mnogokotnike, krožnice, elipse, Bézierove krivulje drugega in tretjega reda. Gradniki imajo lahko različne karakteristike, vsi pa si delijo globalne transformacijske matrike. Oblikovanje gradnikov opravimo z uporabo CSS (angl. Cascading Style Sheets). Gradnike lahko združujemo in jih organiziramo hierarhično [206]. SVG omogoča polnjenje gradnikov (z eno barvo ali gradientno), prozornost, filtre, obrezovanje, barvna modela RGB in HSL. Poleg vektorskih gradnikov SVG podpira tudi rastrske slike (vsaj JPEG in PNG) in tekst. Tako tekst kot rastrske slike imata lastnosti ostalih gradnikov; lahko ju transformiramo z geometrijskimi transformacijami.

SVG omogoča značke SMIL (angl. Synchronized Multimedia Integration Language), s katerimi podpira animacije. Najprej izberemo gradnik (ali skupino gradnikov), nato pa določimo, katere njegove atribute lastnosti CSS bomo spreminjali. Določimo še vmesne vrednosti in časovne parametre. Obširen opis SVG najdemo v [207].

3.3.2 Algoritmi nad vektorskimi slikami

Algoritmov nad vektorskimi slikami je res ogromno in veliko izmed njih smo spoznali pri predmetih računalniška grafika in računalniška geometrija. V nadaljevanju si bomo ogledali le nekaj algoritmov za poenostavljanje lomljenk in rasterizacijo.

3.3.2.1 Poenostavljanje lomljenke

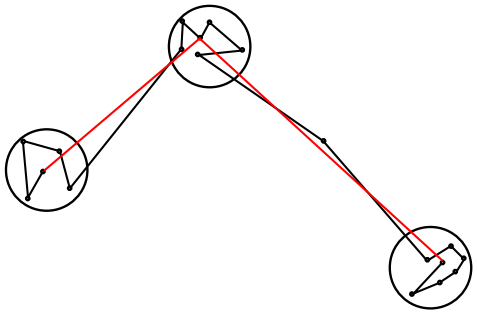
Pri pretvorbi rastrske slike v vektorsko dobimo praviloma množico zelo kratkih odsekov, ki jih je smiselno povezati v daljše odseke, a na način, da čim manj vplivamo na videz lomljenke. Takšna pretvorba je smiselna tudi v aplikacijah, kjer imamo opravka z veliko količino vektorskih podatkov (na primer geografski informacijski sistemi) in kjer pri oddaljenem pogledu nima smisla izrisovati vseh podrobnosti lomljenk. Obstajata dve skupini algoritmov za poenostavljanje [208]:

- Poenostavljeno lomljenko določajo samo točke, ki so podmnožica prvotnih, prva in zadnja točka sta vedno del poenostavljene lomljenke;
- Točke, ki predstavljajo poenostavljeno lomljenko, so lahko drugačne, saj lahko tistim na vhodu dodamo nove.

3.3.2.1.1 Odstranjevanje bližnjih oglišč

Odstranjevanje bližnjih oglišč je najpreprostejša naivna metoda. Oglišča, ki so dovolj blizu, združimo v eno oglišče. Metoda oglišč, ki bi sicer lahko bila

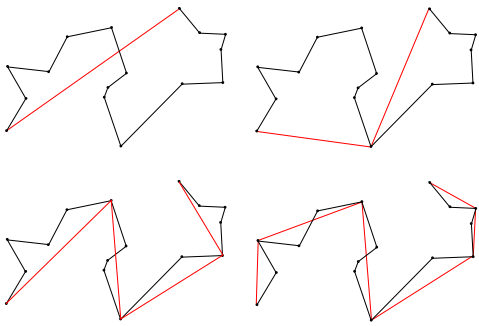
na eni nosilki, a so dlje od določene razdalje, ne odstrani. Idejo prikazuje slika.



Slika 3.29: Poenostavljanje lomljenke z naivno metodo

3.3.2.1.2 Algoritem Douglas-Peucker

Algoritem Douglas-Peucker je najbolj razširjen algoritem [209]. Algoritem najprej poveže prvo in zadnjo točko vhodne lomljenke, nato pa preveri vse preostale točke, če so bliže tej daljici od vnaprej podane razdalje d. Če takšne točke obstajajo, med njimi izberemo tisto, ki je od naše daljice najbolj oddaljena. To oglišče postane novo oglišče poenostavljene lomljenke. Algoritem deluje rekurzivno nad novima odsekoma, dokler ne zadostimo vhodnemu kriteriju dovolj majhne razdalje d (glej sliko 3.30).



Slika 3.30: Poenostavljanje lomljenke z metodo Douglas Peucker

3.3.2.2 Algoritmi rasterizacije

Ena izmed najstarejših, a še vedno aktualnih operacij nad vektorskimi gradniki je pretvorba v dan raster. Postopku pravimo **rasterizacija** in se ga danes niti ne zavedamo, saj je zelo pogosto pohitren s strojno opremo.

Najpreprostejši geometrijski element je 2D točka, ki jo v raster vstavimo zelo preprosto. Njene koordinate (praviloma zapisane s števili s plavajočo vejico) moramo le transformirati v piksel rastra.

3.3.2.2.1 Rasterizacija daljice

Nekoliko zapletenejša je že pretvorba daljice v njeno rastrsko predstavitev. Prvi algoritem, ki pa je še danes izjemno priljubljen in zaradi svoje preprostosti pogosto implementiran v strojni opremi, je razvil **Bresenham** [210], katerega opis povzemamo po [2]. Izhajajmo iz enačbe za naklon premice:

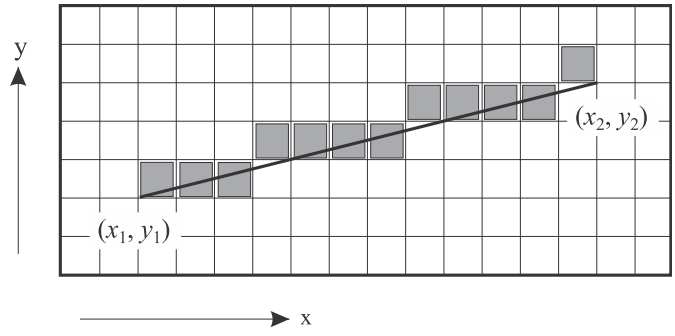
$$k = \frac{y_2 - y_1}{x_2 - x_1} = \frac{dy}{dx}, \tag{3.17}$$

kjer x_1 in y_1 predstavljata koordinate prve točke premice in x_2 in y_2 druge točke. Enačbo 3.17 zapišimo v iterativni obliki:

$$y_{i+1} = y_i + k(x_{i+1} - x_i). \tag{3.18}$$

Predpostavimo, da rišemo daljico z naklonom $0 \leq k < 1$ (slika 3.31); x torej povečujemo od x_1 do x_2 s korakom ena. Enačbo 3.18 lahko zato poenostavimo:

$$y_{i+1} = y_i + k. \tag{3.19}$$



Slika 3.31: Rasterizacija daljice

Vrednost y med dvema zaporednima korakoma v smeri x zato povečamo za naklon k. V vsakem koraku moramo prižgati piksel. Pri tem pa naredimo napako e, ki je enaka oddaljenosti piksla od idealne daljice. V vsakem koraku

torej izbiramo med pikslom, ki je nad ali pod premico. Izberemo tistega, ki ima manjšo kumulativno napako, kar nadziramo z:

- če je $e \leq \frac{1}{2}$, bo y-koordinata piksla enaka predhodnemu pikslu,
- če je $e > \frac{1}{2}$, bomo y-koordinato povečali za ena glede na predhodno vrednost, nova vrednost napake pa bo enaka prejšnji vrednosti napake, zmanjšani za ena.

Ekvivalentno lahko enačbo 3.18 prilagodimo za premice s $k > 1$. V tem primeru korakoma povečujemo koordinato y in enačbo 3.18 spremenimo v:

$$x_{i+1} = x_i + k^{-1}. \tag{3.20}$$

Če zagotovimo $x_1 \leq x_2$ in $y_1 \leq y_2$, lahko zapišemo algoritem za izris daljice, ki ga opisuje psevdokod 3.

Opisan algoritem lahko pohitrimo in poenostavimo njegovo implementacijo v strojni opremi z uporabo zgolj celoštevilskih aritmetičnih operacij:

1. Da bi testirali zgolj predznak napake, moramo začetno vrednost napake premakniti za $\frac{1}{2}$. Za daljice z naklonom $0 \leq k \leq 1$ bo začetna vrednost:

$$e = k - \tfrac{1}{2} = \frac{dy}{dx} - \tfrac{1}{2}. \tag{3.21}$$

2. Enačbo 3.21 pomnožimo z $2dx$ in dobimo:

$$2\,dx\;e = 2\,dy - dx. \tag{3.22}$$

   V nadaljevanju uporabljamo preoblikovano napako, torej: $err = 2\,dx\;e$.

3. Pri določitvi prižganega piksla napaki e prištejemo k, zato bomo korigirali napako err kot:

$$2\,dx\;e = 2\,dx(e + \tfrac{dy}{dx}) = 2\,dx\;e + 2dy = err + 2\,dy. \tag{3.23}$$

Psevdokod 3: Bresenhamov algoritem rasterizacije daljice

```
1: function LineBresenham(x1, y1, x2, y2, c)
2:                          ▷ (x1,y1), (x2,y2) – koordinati točk daljice, c – barva
3:     dx = x2 − x1;
4:     dy = y2 − y1;
5:     x = x1;
6:     y = y1;
7:     if (dx ≥ dy) then                    ▷ naklon daljice je manjši ali enak 1
8:         e = k = dy/dx;                   ▷ inicializiramo napako in naklon
9:         for (i = 1; i ≤ dx; i = i + 1) do
10:            PutPixel(x, y, c);
11:            if (e > 1/2) then
12:                e = e − 1;                       ▷ nova vrednost napake
13:                y = y + 1;                       ▷ premik v naslednjo vrstico
14:            end if
15:            e = e + k;
16:            x = x + 1;
17:        end for
18:    else                                 ▷ naklon daljice je večji od 1
19:        e = k = dx/dy;
20:        for (i = 1; i ≤ dy; i = i + 1) do
21:            PutPixel(x, y, c);
22:            if (e < −1/2) then
23:                e = e + 1;                       ▷ nova vrednost napake
24:                x = x + 1;                       ▷ premik v naslednji stolpec
25:            end if
26:            e = e − k;
27:            y = y + 1;
28:        end for
29:    end if
30: end function
```

4. Poglejmo še, za koliko moramo povečati err v primeru, ko bomo prižgali piksel, katerega koordinata y je za ena večja od predhodnega piksla:

$$err = 2\,dx(e-1) = 2\,dx\;e - 2\,dx = err - 2\,dx. \tag{3.24}$$

Podobno določimo izraz za korekcijo napake tudi pri daljicah s $k > 1$. Celoštevilski algoritem je zapisan v psevdokodi 4. Če algoritem posplošimo in dodamo manjše izboljšave, dobimo psevdokod 5, kot ga je predlagal Bresenham.

Psevdokod 4: Celoštevilski Bresenhamov algoritem rasterizacije daljice

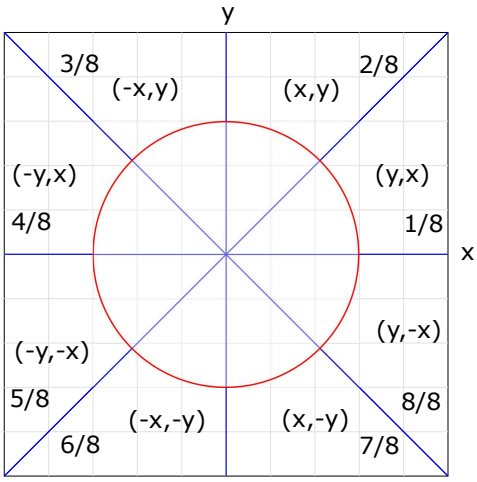
```
1: function LineBresenhamInt(x1, y1, x2, y2, c)
2:                          ▷ (x1,y1), (x2,y2) – koordinati točk daljice, c – barva
3:                                  ▷ Veljati mora: x1 ≤ x2 in y1 ≤ y2
4:     dx = x2 − x1;
5:     ax = dx << 1;
6:     dy = y2 − y1;
7:     ay = dy << 1;
8:     x = x1;   y = y1;
9:     if (ax ≥ ay) then                  ▷ naklon daljice je manjši ali enak 1
10:        err = ay − dx;                              ▷ začetna vrednost napake
11:        for (i = 1; i ≤ dx; i = i + 1) do
12:            PutPixel(x, y, c);
13:            if (err ≥ 0) then
14:                err = err − ax;                     ▷ nova vrednost napake
15:                y = y + 1;                  ▷ premik v naslednjo vrstico
16:            end if
17:            err = err + ay;
18:            x = x + 1;
19:        end for
20:    else                                   ▷ naklon daljice je večji od 1
21:        err = ax − dy;
22:        for (i = 1; i ≤ dy; i = i + 1) do
23:            PutPixel(x, y, c);
24:            if (err > 0) then
25:                err = err − ay;                     ▷ nova vrednost napake
26:                x = x + 1;                  ▷ premik v naslednji stolpec
27:            end if
28:            err = err − ax;
29:            y = y + 1;
30:        end for
31:    end if
32: end function
```

Psevdokod 5: Končni celoštevilski Bresenhamov algoritem rasterizacije daljice

```
1: function BRESENHAM(x1, y1, x2, y2, c)
2:                          ▷ (x1,y1), (x2,y2) – koordinati točk daljice, c – barva
3:                                  ▷ Veljati mora: x1 ≤ x2 in y1 ≤ y2
4:     dx = x2 − x1;   ax = ABS(dx) << 1;
5:     dy = y2 − y1;   ay = ABS(dy) << 1;
6:     sx = SGN(dx);   sy = SGN(dy);
7:     x = x1;   y = y1;
8:     if (ax ≥ ay) then                  ▷ naklon daljice je manjši ali enak 1
9:         err = ay − (ax >> 1);                    ▷ začetna vrednost napake
10:        for (i = 1; i ≤ ABS(dx); i = i + 1) do
11:            PutPixel(x, y, c);
12:            if (err ≥ 0) then
13:                err = err − ax;                  ▷ nova vrednost napake
14:                y = y + sy;                  ▷ premik v naslednjo vrstico
15:            end if
16:            err = err + ay;
17:            x = x + sx;
18:        end for
19:    else                                   ▷ naklon daljice je večji od 1
20:        err = ax − (ay >> 1);
21:        for (i = 1; i ≤ ABS(dy); i = i + 1) do
22:            PutPixel(x, y, c);
23:            if (err > 0) then
24:                err = err − ay;                  ▷ nova vrednost napake
25:                x = x + sx;                  ▷ premik v naslednji stolpec
26:            end if
27:            err = err + ax;
28:            y = y + sy;
29:        end for
30:    end if
31: end function
```

3.3.2.2.2 Rasterizacija krožnice

Bresenham je predlagal tudi algoritem za rasterizacijo krožnice [211]. Opis povzemamo po [212]. Krožnica je sestavljena iz osmih simetričnih delov/osmin, zato je dovolj, če izračunamo piksle samo enega kvadranta, preostale piksle pa dobimo s preslikavo, kot vidimo na sliki 3.32.



Slika 3.32: Razdelitev krožnice na simetrične osmine

Izračun opravimo v drugi osmini. Predpostavimo, da je središče krožnice v izhodišču in naj bo njen polmer r. Postopek rasterizacije začnemo v točki $(0, y)$ oziroma $(0, r)$. V vsakem koraku se bo koordinata x povečala za 1, koordinata y pa bo ostala enaka koordinati y predhodnega piksla ali pa se bo zmanjšala; odločitev o tem nam pove najkrajša razdalja od dejanske krožnice in središč pikslov kandidatov. Predpostavimo, da se trenutno nahajamo v pikslu $P = (x_k, y_k)$, odločamo pa se med piksloma $N = (x_{k+1}, y_k)$ in $S = (x_{k+1}, y_{k-1})$, ki ju oba prebada naša krožnica (slika 3.33). Dobro znana enačba krožnice s središčem v (h, k) je

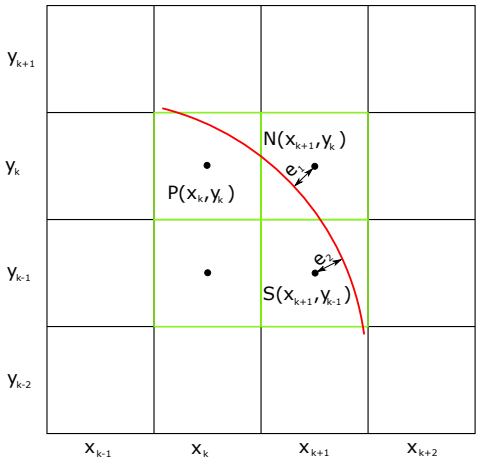
$$(x - h)^2 + (y - k)^2 = r^2. \tag{3.25}$$

Da si olajšamo izračun, postavimo krožnico v izhodišče, tako da dobimo enačbo krožnice, identično Pitagorovemu izreku:

$$x^2 + y^2 = r^2. \tag{3.26}$$

Izračunamo vrednosti napak e_1 in e_2 kot najkrajši razdalji med središčema pikslov N in S ter dejansko krožnico kot:

$$e_1 = x_{k+1}^2 + y_k^2 - r^2, \quad e_2 = x_{k+1}^2 + y_{k-1}^2 - r^2. \tag{3.27}$$



Slika 3.33: Iz piksla P se lahko premaknemo v piksel N ali piksel S.

Ker je N izven krožnice, bo $e_1 > 0$ in obratno, $e_2 < 0$, saj je S vedno znotraj krožnice. Odločitveno spremenljivko označimo z D_k in jo izračunamo kot

$$D_k = e_1 + e_2. \tag{3.28}$$

Če je $D_k < 0$, potem je $|e_2| > |e_1|$, kar pomeni, da je točka N bliže krožnici kot točka S. Zato bomo za naslednji piksel izbrali piksel N. Podobno, če je $D_k > 0$, potem je S bliže in postane naš naslednji piksel. Ko se premaknemo v naslednji piksel, moramo korigirati vrednost napake D_k kot sledi:

$$D_k = x_{k+1}^2 + y_k^2 - r^2 + x_{k+1}^2 + y_{k-1}^2 - r^2. \tag{3.29}$$

x_{k+1} pomeni, da se premaknemo za eno mesto v desno, y_{k-1} pa, da gremo v piksel nižje, zato lahko enačbo 3.29 zapišemo tudi drugače:

$$\begin{aligned} D_k &= (x_k+1)^2 + y_k^2 - r^2 + (x_k+1)^2 + (y_k-1)^2 - r^2 = \\ &= 2(x_k+1)^2 + y_k^2 + (y_k-1)^2 - 2r^2. \end{aligned} \tag{3.30}$$

Določimo še D_{k+1} s tem, da vsak k zamenjamo s $k+1$.

$$D_{k+1} = 2(x_{k+1}+1)^2 + y_{k+1}^2 + (y_{k+1}-1)^2 - 2r^2. \tag{3.31}$$

Ponovno zamenjamo x_{k+1} z $x_k + 1$, y_{k+1} pa tokrat ne moremo zamenjati, ker ne vemo, ali smo se premaknili v piksel N ali S. Tako dobimo:

$$\begin{aligned} D_{k+1} &= 2(x_k+1+1)^2 + y_{k+1}^2 + (y_{k+1}-1)^2 - 2r^2 = \\ &= 2(x_k+2)^2 + y_{k+1}^2 + (y_{k+1}-1)^2 - 2r^2 \end{aligned} \tag{3.32}$$

Odločitveno spremenljivko za naslednji piksel določimo kot

$$\begin{aligned} D_{k+1} - D_k &= 2(x_k+2)^2 + y_{k+1}^2 + (y_{k+1}-1)^2 - 2r^2 - \\ &\quad -(2(x_k+1)^2 + y_k^2 + (y_k-1)^2 - 2r^2) = \\ &\quad 4x_k + 2y_{k+1}^2 - 2y_{k+1} - 2y_k^2 + 2y_k + 6 \end{aligned} \tag{3.33}$$

Končno dobimo:

$$D_{k+1} = D_k + 4x_k + 2y_{k+1}^2 - 2y_{k+1} - 2y_k^2 + 2y_k + 6. \tag{3.34}$$

Ko je $D_k < 0$, kot že vemo, izberemo točko $N = (x_{k+1}, y_k)$, zato v enačbi 3.34 zamenjamo y_{k+1} z y_k in dobimo:

$$\begin{aligned} D_{k+1} &= D_k + 4x_k + 2y_k^2 - 2y_k - 2y_k^2 + 2y_k + 6 = & (3.35) \\ &= D_k + 4x_k + 6. & (3.36) \end{aligned}$$

Podobno, ko je $D_k > 0$, izberemo točko $S = (x_{k+1}, y_{k-1})$. Naslednja koordinata y bo torej y_{k-1}, zato v enačbi 3.34 zamenjamo y_{k+1} z y_{k-1} in dobimo:

$$D_{k+1} = D_k + 4x_k + 2y_{k-1}^2 - 2y_{k-1} - 2y_k^2 + 2y_k + 6. \tag{3.37}$$

Zamenjamo y_{k-1} z $y_k - 1$ in dobimo enačbo 3.38 kot sledi:

$$\begin{aligned} D_{k+1} &= D_k + 4x_k + 2(y_k-1)^2 - 2(y_k-1) - 2y_k^2 + 2y_k + 6 = \\ &= D_k + 4x_k + 2y_k^2 - 4y_k + 2 - 2y_k + 2 - 2y_k^2 + 2y_k + 6 = \\ &= D_k + 4x_k - 4y_k + 10 = \\ &= D_k + 4(x_k - y_k) + 10. \end{aligned} \tag{3.38}$$

Nazadnje še določimo inicializacijsko vrednost odločitvene spremenljivke v začetni točki $(0, y = r)$, kar vstavimo v enačbo 3.29 kot sledi:

$$\begin{aligned} D_k &= 2(x_k+1)^2 + y_k^2 + (y_k-1)^2 - 2r^2 \\ D_0 &= 2(0+1)^2 + r^2 + (r-1)^2 - 2r^2 = \\ &= 2 + r^2 + r^2 - 2r + 1 - 2r^2 = \\ &= 3 - 2r. \end{aligned} \tag{3.39}$$

Bresenhamov algoritem za rasterizacijo krožnice s polmerom r in središčem v (cx, yc) podaja psevdokod 6.

Poleg Bresenhamovega algoritma obstaja še veliko drugih rasterizacijskih algoritmov za daljice [213, 214], krožnice [215, 216], stožnice [217], krivulje [218, 219, 220]. Za mnoge izmed rasterizacijskih algoritmov so pridobljeni tudi patenti [221, 222]. Rasterizacija je pomembna tudi v 3D-prostoru vokslov [223, 224, 225, 226].

Psevdokod 6: Bresenhamov algoritem rasterizacije krožnice

```
1: function BresenhamCircle(xc, yc, r, c)
2:                          ▷ (xc, yc) – središče krožnice, r – polmer, c – barva
3:     x = 0;   y = r;
4:     D = 3 − 2 ∗ r;
5:     while (x ≤ y) do
6:         PutPixel(y + xc, x + yc, c);                          ▷ prva osmina
7:         PutPixel(x + xc, y + yc, c);                          ▷ druga osmina
8:         PutPixel(−x + xc, y + yc, c);                         ▷ tretja osmina
9:         PutPixel(−y + xc, x + yc, c);                         ▷ četrta osmina
10:        PutPixel(−y + xc, −x + yc, c);                        ▷ peta osmina
11:        PutPixel(−x + xc, −y + yc, c);                        ▷ šesta osmina
12:        PutPixel(x + xc, −y + yc, c);                         ▷ sedma osmina
13:        PutPixel(y + xc, −x + yc, c);                         ▷ osma osmina
14:        if (D < 0) then
15:            D = D + 4 ∗ x + 6;                                ▷ ažuriramo napako
16:            x = x + 1;                  ▷ premik v naslednji piksel - piksel N
17:        else
18:            D = D + 4 ∗ (x − y) + 10;                         ▷ ažuriramo napako
19:            x = x + 1;
20:            y = y − 1;                          ▷ naslednji piksel je piksel S
21:        end if
22:    end while
23: end function
```

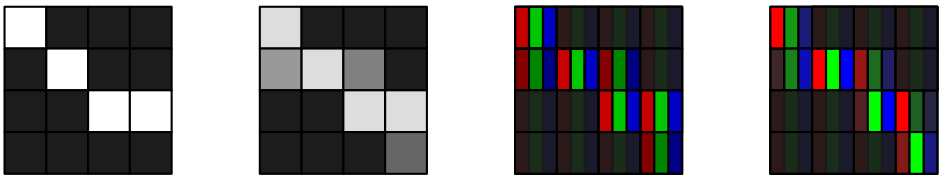
3.3.2.3 Mehčanje robov

Pri prej omenjenih postopkih dobimo nezvezne žagaste (angl. jagged) oziroma nazobčane oblike. To je opazno predvsem pri nizki ločljivosti rastra. Ta pojav imenujemo **alias** (angl. aliasing) in je še posebej opazen pri tekstu na prikazovalnikih nižje ločljivosti. Rešitev je v mehčanju robov (angl. antialiasing), torej rasterizacijo spremenimo tako, da nekatere piksle *prižgemo* le delno. Obstajajo namenske metode za mehčanje robov daljic [227] in krožnic [228]. Ta problematika je v literaturi dobro zastopana [229] in spada na področje računalniške grafike, zato je podrobneje ne obravnavamo.

3.3.2.3.1 Mehčanje robov pod nivojem pikslov

V preteklosti je bila problematika aliasa pri prikazu besedila še posebej moteča, saj so bili takrat prikazovalniki zelo nizke ločljivosti, na primer 800×600 pikslov. Postopki mehčanja robov so prinesli zadovoljivo rešitev, vendar je bila nazobčanost robov zaradi izjemno nizke ločljivosti prikazovalnikov še vedno moteča. Glede na to, da je večinoma tekst statičen in črne barve, je možno mehčanje robov še izboljšati z delnim prižiganjem pikslov. Piksli na prikazovalniku so namreč sestavljeni iz kombinacije področij rdeče, zelene in modre barve. Navidezno ločljivost prikazanega teksta lahko povečamo z ločenim prižiganjem posameznih področij piksla na prikazovalniku. V tem primeru govorimo o mehčanju robov pod nivojem pikslov.

Primer rasterizacije daljice z mehčanjem robov pod nivojem pikslov je prikazan na sliki 3.34. Opazimo, da z mehčanjem robov navidezno povečamo



(a) (b) (c) (d)

Slika 3.34: Rasterizacija premice (a) brez mehčanja robov, (b) z mehčanjem robov, (c) z mehčanjem robov, kot je vidno na prikazovalniku z matriko RGB, in (d) z mehčanjem robov pod nivojem pikslov na prikazovalniku z matriko RGB.

prostorsko ločljivost, kar daje občutek ostrejše slike. Opisana tehnika se je še do pred kratkim zelo pogosto uporabljala pri rasterizaciji vektorskih pisav. S pojavom prikazovalnikov z višjo ločljivostjo pa ni več potrebna in se postopoma opušča, saj ima tudi več slabosti. Rasterizirana slika je

primerna za prikaz zgolj na točno določenih prikazovalnikih. V primeru prikaza na drugih napravah takoj opazimo vizualne motnje, na primer v primeru tiskanja posnetka prikazovalnika črnega besedila dobimo moteče barvne lise na robovih pisav. V primeru animacij gibanja teksta bi bilo še huje, saj bi na robovih znakov videli migetanje barv.

3.4 Seznam nalog

- Izračunajte končno barvo piksla, če na belo ozadje rišemo z delno prosojno barvo RGBA = (0,5, 0,25, 0,4, 0,25).
- Izračunajte ločljivost prikazovalnika 4K z diagonalo 101,6 cm.
- Izračunajte širino prikazovalnika 4K v centimetrih z diagonalo 101,6 cm.
- Izračunajte količino pomnilnika, ki ga zasedejo piksli slike z ločljivostjo 256×256 v formatu BMP, če imamo barvni model RGBA z osmimi biti na kanal.
- Sliko z zapisom R5G6B5 želimo zapisati v datoteko BMP. Sestavite bitno masko za kanale RGB.
- Napišite program, ki iz slike BMP prebere bitno masko za kanale RGB in izpiše, koliko bitov se uporablja za posamezen kanal.
- Na primeru proučite razliko med podvzorčenjem v JBIG in stresanjem.
- Naštejte nove tehnike oziroma izboljšave, ki jih je prinesel JPEG XL glede na JPEG.
- Napišite program, ki bo iz datoteke PNG razbral besedilo iz skupka tEXt.
- Napišite program, ki za poljubno veliko sliko izpiše zaporedje pikslov v vrstnem redu prepletanja Adam7.
- Slike iz lastne galerije stisnite izgubno pri različnih stopnjah stiskanja in izračunajte podobnost stisnjenih slik z metrikama PSNR in SSIM.
- Izvedite rasterizacijo krožnice in izpišite prižgane piksle za krožnico s polmerom 5 pikslov.
- Napišite program, ki bo v sliko v formatu BMP vpisal prikrito sporočilo v formatu ASCII.

- V poljubno sliko vpišite prikrito sporočilo z metodo LSB in izračunajte podobnost nove slike z metodama PSNR in SSIM.
- Napišite program za poenostavljanje lomljenk z algoritmom Douglas-Peucker.
- Napišite Bresenhamov program za rasterizacijo daljic.
- Z uporabo formata SVG napišite program, ki bo izrisal preprost model avtomobila in animiral njegovo vožnjo.

Poglavje 4

Digitalni avdio

Sluh je drugi najpomembnejši receptor zunanjih dražljajev [230]. Valovanju ustrezne valovne dolžine, ki potuje skozi medij (zrak, voda, material) in zadene naš slušni sistem, pravimo **zvok**. Leta 1877 je Thomas Edison izumil in patentiral [231] fonograf (angl. phonograph), mehansko napravo, ki je zmogla zajeti/posneti zvok in ga tudi ponovno predvajati, njegovemu izumu pa je 10 let pozneje sledil izum gramofona [232]. V dvajsetih letih prejšnjega stoletja so mehanske fonografe in gramofone začele nadomeščati elektronske naprave, ki so danes povsem samoumevne: mikrofoni, zvočniki in elektronski ojačevalniki zvoka. Prvi zapis zvoka na magnetni medij je uspel nemški industriji (podjetjema AEG in BASF) tik pred začetkom druge svetovne vojne. Sledili sta vinilska gramofonska plošča (1948) in kaseta (1963), nato pa se je sredi osemdesetih let začela era digitalnega avdia z zapisom na CD, z MP3-predvajalniki in s pretočnim avdiom. Zvoku, ki ga posnamemo, shranimo, obdelamo, prenesemo in nato predvajamo z elektronskimi napravami, pravimo **avdio** (angl. audio).

Najpomembnejša parametra za opis zvoka sta **amplituda** in **frekvenca**; fizikalni veličini, ki ju lahko merimo. Amplitudo lahko povežemo z **glasnostjo**, frekvenco pa z **višino zvoka**, veličini **psihoakustike** oziroma **psihofizike**. Povezavo med fizikalnimi in psihoakustičnimi značilnostmi zvoka bomo zelo neformalno spoznali v nadaljevanju.

Frekvenčni pas valovanja, ki ga slišimo, je teoretično med 20 Hz in 20 kHz; frekvence nad 20 kHz imenujemo **ultrazvok**, tiste pod 20 Hz pa **infrazvok**. Spomnimo, da je valovna dolžina valovanja λ določena kot:

$$\lambda = \frac{v}{f}, \tag{4.1}$$

pri čemer je f frekvenca, v pa hitrost razširjanja valovanja (kot se spomnimo

iz fizike, je $v \approx 343{,}8$ m/s za zrak pri 20 °C na nadmorski višini 0 m). Valovna dolžina valovanja, ki ga slišimo, je torej med 17 m in 1,7 cm. Pri tem si zvok pogosto predstavljamo kot čisti sinusni signal. V terminologiji psihoakustike takšnemu signalu pravimo **ton**. Toni so v realnem svetu redki, pogosteje je zvok sestavljen iz valovanj različnih frekvenc, to je iz osnovne frekvence in višjeharmonskih frekvenc. Takrat govorimo o **zvenu**. V veliko primerih je zvok sestavljen iz množice različnih frekvenc s težko določljivimi oblikami valovanj, kar imenujemo **šum**. Aperiodičnemu šumu s kratkim časovnim valovanjem pa pravimo **pok**.

Zvočno valovanje ustvarja **zvočni tlak** oziroma pritisk na bobnič v našem ušesu. Najmanjši zvočni tlak, ki ga zaznamo, je približno 20 µPa, zvočni tlak, ki presega 20 Pa, pa je lahko že usoden za slušni sistem. Na primer, pogovor na razdalji 1 m povzroči zvočni tlak od 2 mPa do 20 mPa, avtomobil na oddaljenosti 10 m od 0,02 do 0,2 Pa, reaktivni motor na razdalji med 100 in 30 m pa od 6 do 630 Pa [12]. Zvočni tlak nam pove jakost zvoka v neki točki prostora, a z njim ne moremo neposredno določiti moči vira zvoka oziroma **zvočne moči**, saj je zvočni tlak odvisen od oddaljenosti, velikosti prostora, akustičnih značilnosti prostora, temperature, zračnega tlaka in še česa. Kot zanimivost podajmo nekaj vrednosti iz [12]: raketa Saturn V je oddajala 40 MW zvočne moči, letalski reaktivni motor ima zvočno moč 10 kW, motorna žaga 0,1 W, pogovor 10^{-5} W, sodoben hladilnik pa 10^{-7} W. Pri merjenju zvoka se pogosto uporablja tudi **jakost zvoka** (angl. sound intensity), ki določa moč zvočnega valovanja na določeno površino; merimo jo z W/m^2. Maksimalna jakost, ki jo naše uho še zmore, je 1 W/m^2, prag slišnosti pa je 1 pW/m^2.

Naš slušni sistem očitno zmore zaznati zvok v izjemno širokem razponu zvočnega pritiska, zvočne moči oziroma jakosti zvoka. To je možno le, ker je karakteristika našega slušnega sistema logaritemska [233]. Nivo **glasnosti zvoka** lahko izračunamo tako glede na zvočno moč, jakost zvoka ali glede na zvočni tlak. Slednjega je še najlažje določiti, saj ga dokaj preprosto izmerimo z mikrofonom. Tako imenovano **raven zvočnega tlaka** (angl. sound pressure level – SPL) določa enačba 4.2, kjer je p dejanski zvočni tlak, p_0 pa je referenčna vrednost zvočnega tlaka pri pragu slišnosti zvoka s frekvenco 1 kHz. Referenčna vrednost je določena kot 20 µPa.

$$L_p = 2 \log_{10} \frac{p}{p_0} [\mathrm{B}] = 20 \log_{10} \frac{p}{p_0} [\mathrm{dB}] \tag{4.2}$$

Enota merjenja glasnosti zvoka je bell B (angl. Bell), pogosteje pa uporabljamo **decibel** dB, zato govorimo kar o **decibelski glasnosti zvoka**.

Kot smo že omenili, naš slušni sistem zmore zaznati ekstremen razpon glasnosti zvoka med 0 dB in 120 dB (to je zvočni tlak v razmerju $1 : 10^6$).

V tabeli 4.1 podajamo primere glasnosti zvoka glede na vir [12]. Opomnimo, da je uho zmožno le kratkotrajnejše izpostavljenosti visoki glasnosti. Pri izpostavljenosti v osmih urah se lahko naš zvočni sistem trajno poškoduje že pri zvočnem tlaku 0,5 Pa, torej pri 88 dB [12].

Tabela 4.1: Primeri glasnosti zvoka v dB na podlagi zvočnega tlaka pri podani oddaljenosti [12]

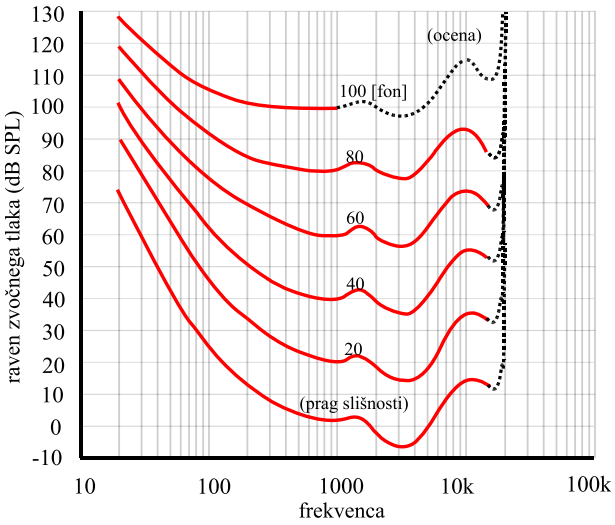
opis	zvočni tlak [Pa]	glasnost [dB]
motor reaktivnega letala (30 m)	630	150
motor reaktivnega letala (100 m)	6–200	110–140
cesta (10 m)	0,2–0,6	80–90
avtomobil (10 m)	0,02–0,2	60–80
pogovor (1 m)	0,002–0,02	40–60
tiha soba	0,0002–0,0006	20–30

4.1 Zaznava zvoka

Na zaznavo glasnosti močno vpliva tudi frekvenca zvoka. Valovanje zvočnega tlaka povzroči nihanje bobniča, ki se prenese do ušesnega polža. Živčne celice ušesnega polža (**dlačnice**) pretvorijo tresljaje v električne impulze, ki se nato prenesejo v možgane. Že zaradi oblike ušesnega polža in strukture dlačnic najbolje slišimo v območju 1 kHz, nižje in višje frekvence pa slišimo slabše. S starostjo se slušna meja oži, saj postanejo dlačnice manj prožne, posledično slabše slišimo višje frekvence [234].

Pionirja določanja odvisnosti dojemanja glasnosti od frekvence sta Fletcher in Munson [235]. Narisala sta **krivulje enake glasnosti** [235], ki določajo nivo zvočnega tlaka pri različnih frekvencah, da ga dojemamo kot enako glasnega pri 1 kHz. Tem krivuljam pravimo **Fletcher-Munsonove krivulje** in so bile osnove za standard ISO 226 [236] (glej sliko 4.1). Vidimo, da se **prag slišnosti** močno spreminja s frekvenco. Na primer, zvok s frekvenco 20 Hz mora biti kar 70 dB močnejši kot zvok s frekvenco 1 kHz. Naš slušni organ je najbolj občutljiv v frekvenčnem pasu med 1 in 5 kHz, kjer lahko slišimo tone celo pod 0 dB.

Na podlagi krivulj enake glasnosti lahko izračunamo **zaznano glasnost** G (angl. perceived loudness) z upoštevanjem vrednosti minimalnega slišnega praga glede na frekvenco (enačba 4.3). G merimo v **fonih** (angl. Phon),



Slika 4.1: Krivulje enake glasnosti zvoka po standardu ISO 226:2003; prirejeno [5] (javna domena)

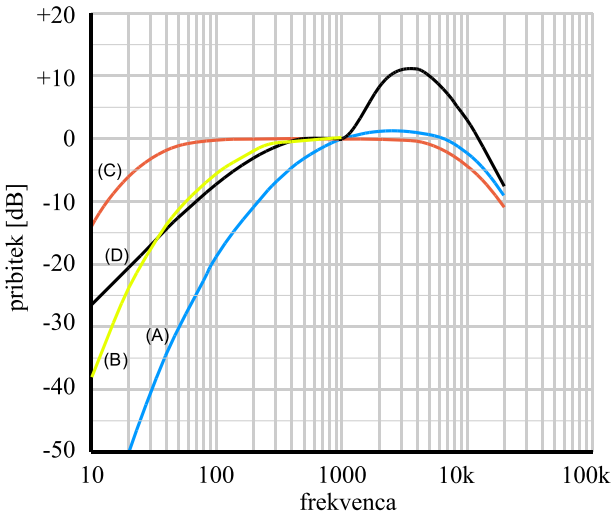
zato včasih govorimo tudi o **fonski glasnosti**.

$$G = 20 \log_{10} \frac{p}{p_0(f)} [\text{fon}]. \tag{4.3}$$

Fon ni fizikalna veličina (uporablja se v psihoakustiki), saj temelji na eksperimentalnih podatkih človeškega občutka, *da nekaj sliši*. Je pa fon definiran s standardi ANSI (ANSI S3.4-2007 [237], ANSI S1.1-2013 [238], ANSI S3.6-2018 [239]).

Kot alternativo uporabljamo tudi **obteževanje A** (angl. A-weighting) po standardu IEC 61672 [240]. Obteževanje A poenostavlja krivulje glasnosti iz standarda ISO 226 [236] tako, da glasnost zvoka z določeno frekvenco korigira s prištevanjem **pribitka** (angl. gain) glasnosti v dB. Pribitek glede na frekvenco je prikazan na sliki 4.2. V primeru zvoka s frekvenco 200 Hz moramo za zapis glasnosti z obteževanjem A vrednost glasnosti znižati za približno 10 dB. Na grafu vidimo, da obstajajo še obteževanja B, C, D, ki pa se redkeje uporabljajo.

S podobnimi izzivi kot pri glasnosti se srečujemo tudi pri zaznavanju frekvenc zvoka, na primer višine tona. Tudi to zaznavamo logaritmično. Pri nižjih frekvencah ločimo tone, ki se razlikujejo le za nekaj Hz, pri višjih pa je za to potrebno nekaj 100 Hz. Tako interval med 100 Hz in 200 Hz zaznamo kot oktavo, prav tako pa slišimo kot oktavo tudi frekvenci 1 kHz in 2 kHz. Zaznava višine tona je odvisna tudi od glasnosti in morebitnih višjeharmonskih frekvenc. Večina poslušalcev bo glasen ton z 200 Hz zaznala



Slika 4.2: Pribitek glasnosti z obteževanji A, B, C in D glede na frekvenco zvoka; prirejeno [6] (javna domena).

kot nižji ton v primerjavi s tišjim tonom enake frekvence. Pri najbolj občutljivih frekvencah v okolici treh kHz pa uho ne zaznava spremembe tona ob spremembi jakosti. Če je zvok sestavljen iz več frekvenc, pogosto pride do interferenc. Ko sta dve frekvenci tonov zelo blizu, slišimo razliko tonov, na primer, tona s frekvenco 32 Hz in 48 Hz slišimo kot ton frekvence 16 Hz [241].

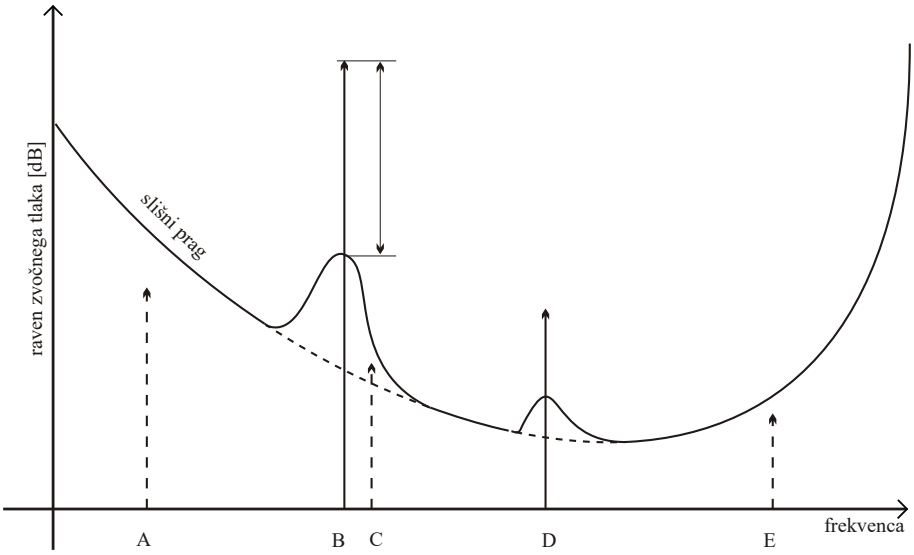
Pojavu, ko en ton onemogoča zaznavo drugega tona, pravimo **slušno prekrivanje** oziroma **maskiranje**. Slušno prekrivanje nastane zaradi omejitev našega slušnega sistema. Prekrivanje je lahko **frekvenčno** ali **časovno**.

- Frekvenčno (tudi istočasno) prekrivanje nastane takrat, ko ob istem času predvajamo dva zvoka podobne frekvence. Na primer, imamo dva tona podobne višine, zato glasnejši ton vpliva na dojemanje tišjega. Pri večji razliki v glasnosti lahko glasnejši ton popolnoma prekrije tišji ton. Stopnja prekrivanja je odvisna od **kritičnega pasu** (angl. critical bands). Kritični pas je frekvenčni pas, v katerem en zvok moti zaznavanje drugega zvoka. Eksperimenti, ki jih je opravil Fletcher [242], so pokazali, da so ti kritični pasovi ožji pri nizkih frekvencah in širši pri višjih. Tako je 3/4 vseh kritičnih pasov pod 5 kHz. Izkustvena formula za določitev širine kritičnega pasu podaja enačba 4.4 [243]

$$\text{širina kritičnega pasu}(f) = 24{,}7(4{,}37\, f + 1)[\text{Hz}], \tag{4.4}$$

  kjer je f središčna frekvenca kritičnega pasu. Prekrivanje lahko pred-

stavimo kot premik minimalnega slišnega praga, kot to vidimo na sliki 4.3. Toni oziroma zvočni signali so predstavljeni s puščicami, njihove višine pa ustrezajo amplitudam puščic. Krivulja ponazarja slišni prag oziroma maskirno krivuljo. Zvok A je pod slišnim pragom te frekvence, zato ga ne slišimo. Zvok B je precej nad slišnim pragom, zato povzroči učinek prekrivanja in nekoliko dvigne krivuljo slišnega praga v neposredni okolici signala. Zaradi tega je tudi zvok C prekrit in ga ne slišimo.



Slika 4.3: Učinek frekvenčnega prekrivanja

- Časovno maskiranje nastane takrat, ko zvok povzroči neslišnost drugega zvoka, ki je tik pred prvim zvokom ali po njem. Pri tem poznamo maskiranje tako naprej kot tudi nazaj [244]. To pomeni, da lahko glasen zvok prekrije drug zvok, ki se je predvajal tik pred izvornim zvokom (**maskiranje nazaj**) ali tik po njem (**maskiranje naprej**). Pri tem je maskiranje nazaj močnejše kot maskiranje naprej. Maskiranje nazaj lahko prekrije glasnejši zvok kot maskiranje naprej, in to zvok, ki se je predvajal več kot 50 ms pred njim. Maskiranje naprej izzveni veliko prej [244].

4.2 Digitalna predstavitev avdia

Kot smo že omenili, je prvo napravo za shranjevanje in predvajanje zvoka izumil Edison 1877. Njegov fonograf je bil mehanski. Po letu 1920 pa je

prišlo do razcveta elektronskih naprav. Iznajdba **mikrofona** je omogočila pretvorbo zvočnega tlaka v analogni električni signal, ki ga je bilo možno shraniti na magnetni medij. Namreč, električni tok, ki ga pošljemo skozi tuljavico, povzroči magnetni pretok. Če je tuljavica navita na visokopermeabilnem magnetnem jedru, jedro pa ima zračno režo, lahko usmerjamo magnetne domene magnetnega materiala, nanesenega na nemagnetni nosilec. Če magnetni material premikamo pod tuljavico, dobimo predstavitev zvočnega pritiska v obliki usmerjenih magnetnih domen. Predvajanje izkorišča princip inducirane napetosti, ki se pojavi kot posledica premikanja orientiranih magnetnih domen v bližini tuljavice. Inducirana napetost požene električni tok, ki steče skozi navitje **zvočnika**. Navitje je nameščeno na membrani zvočnika, ki se zaradi spremembe toka skozi tuljavico v magnetnem polju trajnega magneta premika skladno z jakostjo električnega toka, s čimer povzroči zračni tlak, ki ga slišimo.

Analogni nosilci pa imajo pomembno slabost. Pri prepisovanju oziroma kopiranju orientacij magnetnih domen iz nosilca na nosilec pride do delne izgube, s čimer se poveča šum. Orientacija magnetnih domen se s časom delno spremeni, s čimer se kakovost posnetka, žal, slabša. Te slabosti je odpravila uvedba digitalnega zapisa avdia. Tega dobimo z **vzorčenjem** in **kvantizacijo** analognega signala. To nalogo opravijo **analogno-digitalni pretvorniki** (angl. analog-digital converter, ADC). Pri predvajanju zaporedje vzorcev pretvorimo v analogen električni signal z **digitalno-analognim pretvornikom** (angl. digital-analog converter, DAC), ki ga ojačimo in pošljemo zvočniku.

Signale, ki jih vzorčimo z več kot **Nyquistovo frekvenco** [245, 7] (dvakratno frekvenco najvišje frekvence v signalu), lahko uspešno rekonstruiramo. Tipične frekvence vzorčenja avdia so med 8 kHz in 48 kHz. S frekvenco vzorčenja 8 kHz lahko vzorčimo signale s frekvencami do 4 kHz, kar praviloma zadošča za vzorčenje človeškega glasu. Frekvenca vzorčenja 48 kHz pa več kot zadovoljivo pokrije celoten slišni spekter. V praksi se za digitalni avdio zelo pogosto uporablja nižja frekvenca vzorčenja, in sicer 44,1 kHz. Zvoku, vzorčenim s to frekvenco, pravimo, da ima CD-kakovost.

Amplitudo zvoka najpogosteje predstavimo s šestnajstimi biti, za današnje kakovostnejše zvočne kartice pa uporabljamo tudi 32 bitov. Na podlagi stopnje kvantizacije vzorcev lahko izračunamo kvantizacijsko napako kot **razmerje med signalom in šumom** (angl. signal-to-noise ratio, SNR):

$$SNR = 20 \log_{10} \frac{A_{\text{signal}}}{A_{\text{šum}}} [\text{dB}], \tag{4.5}$$

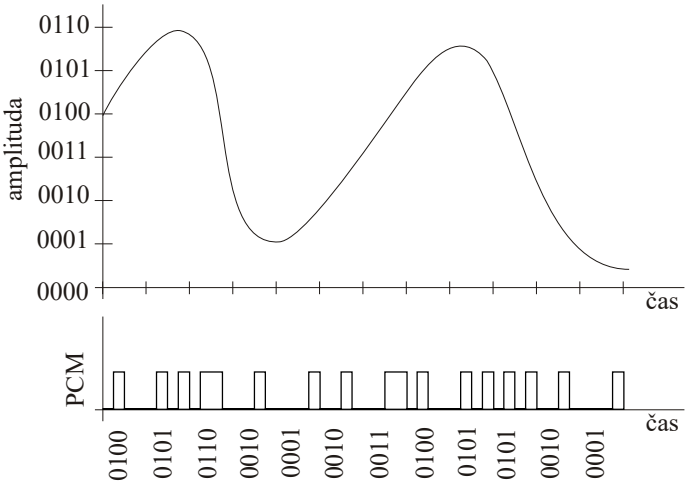
kjer A_{signal} predstavlja največjo možno amplitudo, ki jo lahko predstavimo z n biti, torej (2^{n-1}) v primeru predznačnih vrednosti, $A_{\text{šum}}$ pa postavimo

na 1/2. Kvantizacijska napaka pri 8-bitnih vzorcih bi bila, na primer, 48,16 dB. Enačbo 4.5 lahko aproksimiramo z **razmerjem med signalom in kvantizacijskim šumom** (angl. signal-to-quantisation noise ratio – SQNR):

$$SQNR = 20 \log_{10} 2^n \approx 6{,}021\ n[\text{dB}]. \tag{4.6}$$

4.2.1 Pretvorba zvoka v diskretne vzorce

Najpogosteje zvok vzorčimo v enakomernih intervalih, čemur pravimo **pulzno-kodna modulacija** (angl. pulse code modulation – PCM) (glej sliko 4.4).



Slika 4.4: Pulzno-kodna modulacija

Najpreprostejša izvedba PMC je **linearna pulzno-kodna modulacija** (angl. Linear pulse-code modulation – LPCM), ki velikost amplitude vzorči linearno. LPCM zaradi svoje preprosti uporabljamo zelo pogosto. Najdemo jo v datotečnem formatu **WAV**, pri avdio CD-jih in DVD-jih ter pri prenosu avdia z uporabo priključka HDMI. Žal pa količina podatkov, dobljena z LPCM, ni zanemarljivo majhna. Na primer, na avdio CD zapisujemo z vzorčevalno frekvenco 44,1 kHz, velikost vzorca je 16 bitov in imamo dva zvočna kanala, kar zahteva hitrost prenosa 1.411.200 bitov/s. Zato je smiselno avdio **stiskati**. Algoritmi stiskanja avdia **brez izgub** uporabljajo klasične pristope k stiskanju (napoved in kodiranje dobljenih napak), prav nasprotno pa se tehnike odstranjevanja manj pomembnih informacij pri stiskanju **z izgubami** bistveno razlikujejo od tistih pri stiskanju slik. Slušna zaznava je namreč zelo drugačna kot vidna. Metode stiskanja avdia upoštevajo logaritmično karakteristiko zaznave tako amplitud kot frekvenc in podrobneje

vzorčijo šibkejše jakosti in nižje frekvence. Prvi korak k uspešnejšemu zmanjševanju količine podatkov je zato nelinearna pretvorba vrednosti, zajetih z AD-pretvornikom. Na primer, enačba 4.7 pretvori 16-bitne vzorce v 15-bitne na način, da majhne vrednosti prizadene manj:

$$y = 32767(2^{\frac{x}{65536}} - 1). \tag{4.7}$$

Inverzna enačba je naslednja:

$$x = 65536 \log_2(1 + \frac{y}{32767}) \tag{4.8}$$

S pretvorbo 16-bitnega vzorca v 15-bitnega, kot je predlagano z enačbama 4.7 in 4.8, seveda ne dosežemo velikega razmerja stiskanja. Mnogo boljše rezultate dobimo s **transformacijama μ** in **A**, ki sta bili razviti za telefonijo **ISDN** (angl. Integrated Services Digital Network) v okviru mednarodnega standarda **G.711** [246]. Najdemo ju tudi v nekaterih datotečnih formatih, na primer WAV ali Au.

Transformacija μ (enačba 4.9) prejme na vhodu 14 bitov in jih preslika v 8 bitov:

$$y = sgn(x)\frac{\ln(1 + \mu|x|)}{\ln(1 + \mu)}, \quad \text{kjer} \;\; sgn(x) = \begin{cases} +1 & x > 0, \\ 0 & x = 0, \\ -1 & x < 0 \end{cases} \tag{4.9}$$

Na vhodu imamo torej 112.000 bitov/s, na izhodu pa dobimo 64.000 bitov/s, kar nam da stopnjo stiskanja 1,75. Vhodne vzorce iz intervala $[-8.192, 8.161]$ najprej preslikamo v interval $[-1, 1]$, dobljene vrednosti pa nato pomnožimo z 256, tako da so transformirane vrednosti iz intervala $[-128, 127]$.

Inverzna preslikava je definirana z enačbo:

$$x = sgn(y)\frac{(1 + \mu)^{|y|} - 1}{\mu}. \tag{4.10}$$

Vrednost y je seveda odvisna od parametra $\mu > 0$. Večje vrednosti močneje kvantizirajo večje amplitude. Standard G.711 za govor predlaga $\mu = 255$. Poglejmo, kako deluje transformacija μ. Imejmo vrednosti 0,17 in 0,18, μ pa naj bo 255. Po transformaciji dobimo 0,683857774 in 0,693939502, kar nam da razliko 0,010081728. Če imamo večje normalizirane vrednosti, a z enako razliko, recimo 0,96 in 0,97, dobimo 0,992667638 in 0,993801261 z občutno manjšo razliko 0,001133623.

Transformacija A (angl. A-law) na vhodu prejme 13 bitov in jih preslika v 8 bitov. Definiramo jo z enačbo:

$$y = \begin{cases} sgn(x)\frac{A|x|}{1+\ln(A)} & |x| < \frac{1}{A} \\ sgn(x)\frac{1+\ln(A|x|)}{1+\ln(A)} & \frac{1}{A} \leq |x| \leq 1 \end{cases} \tag{4.11}$$

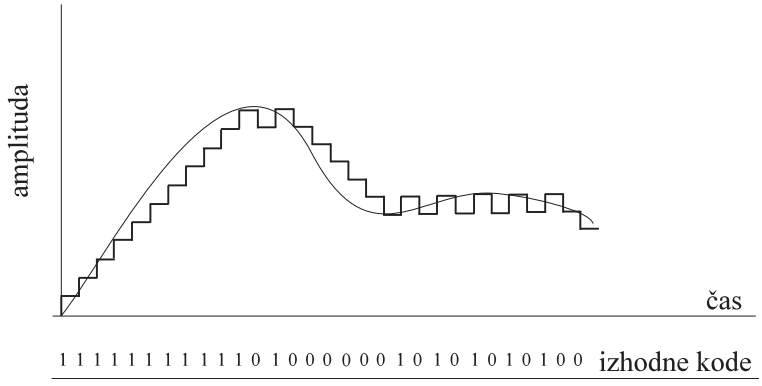
G.711 predlaga $A = 87{,}6$. Inverzno transformacijo A podaja enačba 4.12:

$$x = \begin{cases} sgn(y)\frac{|y|(1+\ln(A))}{A} & |y| < \frac{1}{1+\ln(A)} \\ sgn(y)\frac{\exp(-1+|y|(1+\ln(A)))}{A} & \frac{1}{1+\ln(A)} \leq |y| < 1 \leq 1 \end{cases} \tag{4.12}$$

Transformacijo A uporabljamo v Evropi, transformacijo μ pa v Združenih državah in na Japonskem.

Ko govorimo o količini potrebne informacije za zapis avdia, je zagotovo smiselna uporaba napovedi, torej izračunati napako napovedi (to je razliko med napovedano in dejansko vrednostjo) ter shraniti napako. Že najbolj preprosta napoved, to je predhodni vzorec, daje uporabne rezultate. To idejo uporablja **DPCM** (angl. Differencial Pulse Code Modulation). Slabost DPCM je, da v primeru napake pri prenosu podatkov pravilnih vrednosti ne moremo več rekonstruirati.

Modulacija delta, tudi **modulacija Δ** (angl. Delta Modulation, MD), je inačica modulacije DPCM, ki je uporabna pri zelo velikih frekvencah vzorčenja. Za vsak vzorec uporabi le en bit. Idejo vidimo na sliki 4.5. Zaporedje enic označuje povečevanje, zaporedje ničel pa zmanjševanje amplitude signala. Če je frekvenca vzorčenja premajhna, izhodni signal ne more dobro slediti spremembam vhodnega signala, posledica česar je preveliko popačenje pri rekonstrukciji. Da bi na primer dosegli kakovost 16-bitnega kodiranja PCM s SNR 96 dB, mora biti frekvenca vzorčenja DM reda 200 MHz.



Slika 4.5: Modulacija delta

To slabost rešuje **prilagodljiva DPCM** (angl. Adaptive Differential Pulse Code Modulation ali Adaptive Delta Pulse Code Modulation – ADPCM). Na primeru slike 4.5 bi ADPCM na desni strani pri bolj položnem signalu zmanjšala velikost koraka. Dovolj hitro sledenje signalu označujejo izmenjajoče se zaporedje logičnih enic in ničel, zato se lahko

velikost stopnice v tem primeru zmanjša. Posledica je izboljšanje razmerja signal/šum brez povečanja frekvence vzorčenja. Zaradi dobrih lastnosti je ADPCM podprt v datotečnih formatih WAV in AU.

Obstajajo še druge variante pulzne modulacije, na primer **pulzno-amplitudna modulacija** (angl. pulse amplitude modulation – **PAM**), **pulzno-položajna modulacija** (angl. pulse position modulation – **PPM**) in **pulzno-širinska modulacija** (angl. pulse width modulation – **PWM**), a se pri digitalnem avdiu uporabljajo zelo redko.

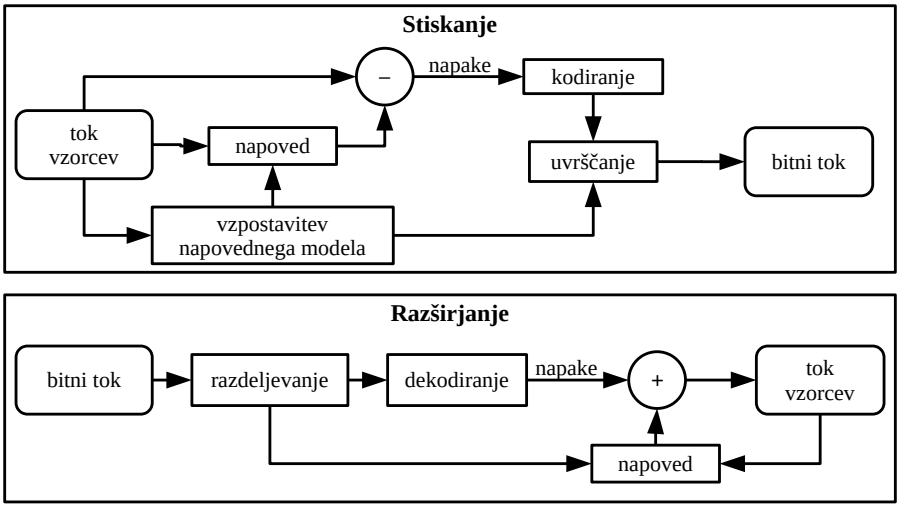
4.3 Stiskanje avdia

Količino podatkov, ki jo zasede avdio, lahko zmanjšamo z znižanjem frekvence vzorčenja, bitne globine, s prej omenjenimi transformacijami (na primer transformacija delta) ali kombinacijo naštetega. Slabost teh postopkov pa je opazno poslabšanje kakovosti, zato jih je treba uporabljati zelo previdno. Za stiskanje avdia so zato razvili učinkovitejše, avdiu prirejene postopke, ki jih bomo spoznali v nadaljevanju. Podobno kot pri slikah ločimo med izgubnim, brezizgubnim in skoraj brezizgubnim stiskanjem avdia.

4.3.1 Brezizgubno stiskanje digitalnega avdia

Pri brezizgubnem stiskanju avdia poskušamo zmanjšati **informacijsko entropijo** z zmanjšanjem korelacije, pri čemer razlikujemo med korelacijo med zaporednimi vzorci in korelacijo med kanali. V obeh primerih uporabljamo **napovedi**, razliko med dejansko in napovedano vrednostjo vzorca pa stisnemo brezizgubno. Slika 4.6 prikazuje splošen postopek brezizgubnega stiskanja avdia. Glede na vhodni tok vrednosti vzorcev vzpostavimo napovedni model, ki napove vrednosti vzorcev na podlagi predhodnih vzorcev ali drugih kanalov. Na podlagi napovedi nato izračunamo **napake**, ki jih zakodiramo s klasičnimi algoritmi stiskanja. Pri tem lahko uporabimo na primer Huffmanov kodirnik ali kodiranje Golomb-Rice [26]. V izhodni bitni tok pošljemo tudi koeficiente napovednega modela. Avdio praviloma želimo stiskati sproti, torej že med zajemom zvoka, zato s postopkom **uvrščanja** koeficiente napovednega modela shranimo blizu pripadajočim napakam. Napovedni model se v mnogih primerih neprestano prilagaja trenutnim značilnostim avdia. Pri nekaterih metodah opravimo tudi napovedi med kanali tako, da vrednosti v enem kanalu napovemo na podlagi drugega kanala. Tako nam uspe avdio stisniti brez izgube informacij do, odvisno od podatkov, tretjine velikosti originalnega zapisa avdia.

Razširjanje je obraten postopek stiskanja. Vhodni bitni tok **razdelimo** na zaporedje napak napovedi in parametrov napovednega modela. Nato



Slika 4.6: Poenostavljen splošen postopek brezizgubnega sprotnega stiskanja avdia

vrednosti vzorca napovemo na podlagi parametrov napovednega modela in predhodnih vzorcev, iz bitnega niza dekodiramo napake in jih prištejemo k napovedim. Rezultat pošljemo na izhod kot zaporedje vzorcev. Razširjanje je praviloma hitrejše od stiskanja, saj parametre napovednega modela dobimo iz bitnega niza, kodirnik pa jih določi z računskimi postopki.

Najbolj znani postopki brezizgubnega stiskanja avdia so SHORTEN, MLP, FLAC, Monkey's Audio, WavPack, ALAC, MPEG-4 ALS, MPEG-4 SLS, OptimFROG DualStream, DTS-HD Master Audio (DTS-HD MA; DTS++), WavPack. Delovanje najpogostejših postopkov stiskanja si bomo pogledali v nadaljevanju.

4.3.2 Izgubno stiskanje avdia

Algoritmi za brezizgubno stiskanje avdia ohranijo vse frekvence zvoka, a veliko jih zaradi razlogov, ki smo jih opisali na začetku tega poglavja, ne slišimo. Poleg tega je v avdiu tudi šum, ki je predstavljen s povsem naključnimi vrednostmi vzorcev, stiskanje takšnih zaporedij pa ni učinkovito. Kot smo omenili, v najboljšem primeru dosegamo razmerje stiskanja 1 : 3. Če bi imeli na primer 5-minutni stereozvok vzorčen s 44,1 kHz in velikostjo vzorca 16 bitov, bi velikost stisnjene datoteke zmanjšali s prvotnih 53 MB na 18 MB.

Če želimo občutneje zmanjšati velikost avdiodatotek, moramo uporabiti izgubno stikanje avdia. Postopki za izgubno stiskanje avdia temeljijo na

podobnih konceptih, kot smo spoznali pri izgubnem stiskanju slik, na primer pri JPEG, pri čemer pa uporabimo vedenje o naših slušnih zaznavah. Frekvenčni pas avdia zato najprej razdelimo v **frekvenčne podpasove** (angl. subband), podatke, ki se nahajajo v njih, pa pretvorimo v frekvenčni prostor. Zatem močneje kvantiziramo tiste frekvence, ki jih slabše slišimo. Naprednejši postopki stiskanja upoštevajo tudi časovno in frekvenčno maskiranje na podlagi psihoakustičnega modela. Tako dosegajo razmerje stiskanja tudi do 1 : 20 brez opazne izgube kakovosti. Avdio iz prejšnjega primera bi tako brez težav stisnili na slabih 3 MB.

Če s postopkom izgubnega stiskanja ne zaznamo izgub, govorimo o **prosojnosti stiskanja** (angl. transparency) oziroma **stiskanju brez zaznavne izgube**. Seveda je doseganje prosojnosti stiskanja odvisno od parametrov, s katerimi določamo stopnjo kvantizacije. **Prag prosojnosti** (angl. transparency threshold) določa parametre, s katerimi dosežemo prosojnost stiskanja.

Najbolj znani postopki izgubnega stiskanja avdia so AC3, WMA, AAC, Vorbis, Opus, in MPEG-1 Audio.

4.4 Formati za shranjevanje avdia

Tudi za zapis avdia obstaja veliko datotečnih formatov, ki so sestavljeni iz dveh delov: iz **glave**, ki vsebuje **metapodatke** (na primer avtor skladbe, naslov posnetka, število kanalov, frekvenca vzorčenja), in **telesa**, ki vsebuje avdiopodatke. Večina datotečnih formatov omogoča različne načine zapisa avdia in vključitev drugih tipov multimedijskih podatkov, kar je podprto z **vsebniki/vsebovalniki** (angl. containers). Sam način kodiranja torej ni nujno določen z vsebnikom. Na primer, kodiranje FLAC seveda uporablja datotečni format FLAC, a tudi vsebniki Ogg, Matroska in MP4 lahko zapišejo podatke s kodiranjem FLAC. Sam postopek kodiranja in dekodiranja opravlja namenska programska ali strojna oprema za kodiranje in dekodiranje multimedijskih vsebin (avdia ali videa), torej kodirnik-dekodirnik (angl. COder-DECoder), ki mu pravimo krajše **kodek** (angl. codec). Kodek je torej implementacija postopka stiskanja/razširjanja, ki se uporablja pri branju vsebin iz vsebnikov (to je iz datotek). Najbolj znani vsebniki za avdio in druge multimedijske tipe podatkov (na primer video) so: OGG, 3GP, AVI, Matroska (MKV), WebM, MPEG-TS (prenos po omrežju), MP4, RM in QuickTime.

4.4.1 WAV

Z razvojem datotečnega formata WAV (angl. Waveform Audio File Format) sta začeli podjetji Microsoft in IBM (1991) [247]. Z njim najpogosteje

zapišemo nestisnjene avdie, pridobljene z modulacijo LPCM. Podpira tudi stisnjene oblike zapisa digitalnega avdia (na primer transformacija μ), različne frekvence vzorčenja in kvantizacijo z različnim številom bitov. V primeru uporabe LPCM, frekvence vzorčenja 44,1 kHz in 16-bitne globine z dvema kanaloma (**stereo**) je možen neposreden zapis na avdio CD, pri čemer je treba izpustiti metapodatke. Velikost datotek WAV je omejena na 4 GB zaradi uporabe 32 bitnih vrednosti v glavi datoteke (ta omejitev je pogosta tudi pri drugih formatih). Zaradi tega je izšla nadgradnja Wave 64 (W64), ki pa se ni uveljavila, saj je pri večjih datotekah vsekakor smiselno uporabiti naprednejše postopke stiskanja.

Datotečna struktura WAV temelji na formatu **RIFF** (angl. Resource Interchange File Format), ki smo ga že omenjali pri formatu WebP. Datoteka RIFF je sestavljena iz **segmentov** oziroma **skupkov** (angl. chunks). Vsak segment je sestavljen iz treh elementov:

- *ID* – štirje zlogi, ki s kodiranjem ASCII določajo tip segmenta,
- *size* – štirje zlogi za velikost podatkov v številu zlogov in
- *data* – podatki.

Datoteka RIFF se začne s segmentom tipa RIFF, pri katerem se element s podatki začne s štirimi zlogi z oznako formata podatkov (angl. FormType), na primer »WAVE« pri datotečnem formatu WAV. Struktura glavnega segmenta temelji na tipu segmenta *LIST*, zato v nadaljevanju vsebuje gnezdene segmente (angl. subchunks), katerih tip je odvisen od formata podatkov. Pri datotekah WAV element s podatki glavnega segmenta poleg oznake »WAVE« vsebuje vsaj dva segmenta, in sicer »fmt␣« in »data«. Segment »fmt␣« (angl. format) vsebuje podatke o načinu zapisa avdia, kot sledi:

- format kodiranja, na primer kodiranje LPCM s celimi števili, kodiranje LPCM z decimalnimi števili ali uporaba stiskanja (na primer transformacija μ, ADPCM in celo MPEG-1 Audio);
- število kanalov;
- frekvenca vzorčenja (na primer 8 kHz, 44,1 kHz);
- bitna globina, najbolj pogosto: 8, 16 in 32;
- bitna hitrost v številu zlogov na sekundo;
- velikost blokov, kar se uporablja v primeru stiskanja in združevanja vzorcev v bloke (na primer ADPCM).

Sledi segment »data«, ki je pri LPCM zapisan z zaporedjem prepletenih vrednosti iz posameznih kanalov.

Datoteke WAV lahko vsebujejo dodatne informacije, kot je na primer datum snemanja. Takrat segment RIFF vsebuje poleg segmentov »fmt␣« in »data« še segment »LIST«, pri katerem se element s podatki začne s štirimi zlogi »INFO« in nato sledijo segmenti s specifičnimi informacijami, na primer segmenta *»ICOP«* (angl. Copyright) in *»ICRD«* (angl. Creation date).

Podobno kot datoteka WAV so strukturirane datoteke **AIFF** (angl. Audio Interchange File Format) [248], ki se že od leta 1988 uporabljajo predvsem na računalnikih podjetja Apple. Glavne razlike glede na WAV so v tem, da AIFF temelji na **IFF** (angl. Interchange File Format), ki pa je zelo podoben RIFF. RIFF dejansko temelji na IFF. AIFF podpira samo LPCM, dodatni načini stiskanja pa so podprti v datotečnem formatu **AIFF-C** [249], ki je razširitev AIFF. Namesto AIFF-C pa se uveljavlja novejši format podjetja Apple, to je **CAF** (angl. Core Audio Format).

4.4.2 SHORTEN

SHORTEN je brezizguben format za stiskanje avdia v CD-kakovosti (44,1 kHz, 16 bit, stereo, PCM). Razvil ga je Robinson [250] in je prosto dostopen. Zvočne datoteke, kodirane s SHORTEN, imajo končnico SHN. Osnovna ideja algoritma je zelo podobna tisti s slike 4.6. SHORTEN vrednosti vzorcev napove na podlagi predhodnih vzorcev. Pri tem vzorce najprej razbije na bloke, napovedi (**napovedni model**) pa prilagaja posameznim blokom. Kodirni proces deluje v treh korakih: tvorjenje blokov, modeliranje napovedi in kodiranje ostankov.

1. **Tvorjenje blokov.** Pri **tvorbi blokov** zaporedje vzorcev razbijemo na ravno prav velike bloke, pri čemer je njihova velikost odvisna od vsebine avdia. Prekratki bloki niso smiselni, ker za opis njihovega napovednega modela porabimo preveč prostora, pri predolgih blokih pa običajno ne moremo določiti parametrov napovednega modela, ki bi dobro napovedovali veliko večino vzorcev. Privzeta dolžina bloka je pri SHORTEN 256 vzorcev. Vsak kanal obravnavamo posebej. Čeprav je pri večkanalnem avdiu (na primer stereo) korelacija občutna, pa avtor predpostavlja, da je korelacija znotraj kanala večja.

2. **Modeliranje napovedi.** SHORTEN napove vrednosti vzorcev in kodira, kot običajno, zgolj napako e med dejansko vrednostjo s in napovedano vrednostjo vzorca $\hat{s}$ (enačba 4.13)

$$e(t) = s(t) - \hat{s}(t) \tag{4.13}$$

SHORTEN ima dve vrsti napovedi: **standardno linearno napoved** in **omejeno napoved**, kjer izberemo enega izmed štirih vnaprej pripravljenih polinomov.

- Standardna linearna napoved uporablja enačbo 4.14.

$$\hat{s}(t) = \sum_{i=1}^{n} a_i s(t-i) \tag{4.14}$$

  Ključna naloga je določiti tako red polinoma n kot koeficiente a_i. Te določimo z inkrementalnim algoritmom **Levinson-Durbin** [250]. Izbiro koeficientov opravi tako, da za vsako iteracijo izračuna pričakovano število bitov za kodiranje celotnega bloka, torej ostankov in koeficientov a_i, nato pa postopek ponavljamo, dokler se skupno število bitov zmanjšuje.

- Omejena napoved preizkuša polinome, podane z enačbo 4.15, pri čemer uporablja nekaj predhodnih vrednosti vzorcev.

$$\begin{aligned} \hat{s_0}(t) &= 0 \\ \hat{s_1}(t) &= s(t-1) \\ \hat{s_2}(t) &= 2s(t-1) - s(t-2) \\ \hat{s_3}(t) &= 3s(t-1) - 3s(t-2) + s(t-3), \end{aligned} \tag{4.15}$$

  ter z njimi določa napake $e_i(t)$ z enačbami 4.16 [250]:

$$\begin{aligned} e_0(t) &= s(t) \\ e_1(t) &= e_0(t) - e_0(t-1) \\ e_2(t) &= e_1(t) - e_1(t-1) \\ e_3(t) &= e_2(t) - e_2(t-1), \end{aligned} \tag{4.16}$$

  nato pa izbere tisto funkcijo, katere kodiranje napak da najmanjše število bitov. Avtor je z eksperimenti pokazal, da metoda pri uporabi napovedi prvega reda uspe stiskati na 45 % velikosti izvornih podatkov. Z uporabo funkcij višjih redov se uspešnost še nekoliko poveča, in sicer se datoteka v povprečju zmanjša na 42 % [250]. Seveda pa je hitrost stiskanja v prvem primeru veliko višja.

3. SHORTEN zapiše ostanke z Golomb-Riceovo kodo, razloženo v mnogih virih [251, 7, 26].

4.4.3 FLAC

FLAC (angl. Free Lossless Audio Compression codec) je odprtokodni projekt za brezizgubno predstavitev avdia, pri katerem sodeluje množica razvijalcev. Zaradi tega je podprt na večini današnjih operacijskih sistemov, vključno z Windows, Mac OS in Linux, pa tudi v strojni opremi. Podpira zelo široko območje vzorčenja (od 1 Hz do 1.048.570 Hz), stiska lahko do 8 kanalov, bitna globina vzorcev je od 4 do 32 bitov. Osnovna ideja stiskanja temelji na kodeku SHORTEN, pri čemer pa odpravlja tudi korelacijo med kanali [7]. Tudi FLAC je, tako kot večina avdiokodirnikov, asimetričen, saj je postopek razširjanja (predvajanja) hitrejši od stiskanja.

Tudi FLAC razdeli podatke v bloke in vsak blok kodira neodvisno. Najprej poskuša zmanjšati korelacijo med kanali, pri čemer izbira med štirimi načini:

- **Način neodvisno.** Vsak kanal kodiramo ločeno.
- **Način sredina.** Izračunamo aritmetično sredino levega in desnega vzorca kot $mid = (left + right)/2$ ter njuno razliko $side = left - right$.
- **Način leva stran.** Shranimo vrednost levega vzorca ($left$) in razliko med levim in desnim vzorcem ($left - right$).
- **Način desna stran.** Podobno kot v prejšnjem primeru, le da tokrat kodiramo vrednost desnega vzorca ($right$) in razliko z levim, to je $right - left$.

Način zmanjševanja korelacije med kanali lahko poda uporabnik, lahko pa kodirnik preizkusi vse načine in za vsak blok izbere najboljšega.

Zatem FLAC zmanjša korelacijo med vzorci kanala. Tudi v tem primeru uporablja štiri načine:

1. **Način nespremenjeno** (angl. verbatim), ki vedno napoveduje vrednost 0, razlika je zato kar originalna vrednost vzorca, zato vzorce brez dodatnega kodiranja zapišemo na izhod.
2. **Način konstantno.** Ta način je primeren za kodiranje tako imenovane **digitalne tišine** (angl. digital silence). Digitalna tišina je predstavljena z vzorci, katerih vrednosti so v bližnji okolici polovice vrednosti, ki jih zmore poslati na izhod analogno-digitalni pretvornik.
3. **Način omejena linearna napoved** (angl. fixed linear predictor). Napovedne funkcije so isti polinomi kot pri metodi SHORTEN, le da je vključena tudi funkcija četrtega reda. FLAC na začetku vsakega bloka s tremi biti zakodira red uporabljenega polinoma.

4. **Način linearna napoved** (angl. linear predictive coding – LPC). Tudi v tem primeru lahko določimo koeficiente z algoritmom Levison-Durbin, pri čemer je najvišji red uporabljenega polinoma 32.

Tudi FLAC dobljene napake med dejansko in napovedano vrednostjo zakodira z Golomb-Riceovimi kodami.

Kot smo že omenili, FLAC omogoča tudi linearno napovedovalno funkcijo reda 4. Poglejmo idejo določitve [7]. Začnemo s štirimi točkami $P_4 = (t-4, s_4)$, $P_3 = (t-3, s_3)$, $P_2 = (t-2, s_2)$ in $P_1 = (t-1, s_1)$ in skonstruirajmo Lagrangeov polinom. Dobimo polinom $Q_3(t) = \sum_{i=0}^{3} P_{i+1} L_i^3(t)$, kjer je Lagrangeov interpolacijski polinom določen z enačbo 4.17

$$L_i^3 = \frac{\prod_{j \neq i}^{3} (t - t_j)}{\prod_{j \neq i}^{3} (t_i - t_j)} \quad 0 \leq i \leq 3, \tag{4.17}$$

od koder dobimo

$$\begin{aligned} Q(t) &= -\frac{1}{6}(t-1)(t-2)(t-3)P_4 + \frac{1}{2}t(t-2)(t-3)P_3 - \\ &\quad -\frac{1}{2}t(t-1)(t-3)P_2 + \frac{1}{6}t(t-1)(t-2)P_1. \end{aligned} \tag{4.18}$$

Enostavno lahko preverimo, da velja $Q(0) = P_4$, $Q(1) = P_3$, $Q(2) = P_2$ in $Q(3) = P_1$. Če vstavimo $t = 4$, dobimo $Q(4) = 4P_1 - 6P_2 + 4P_3 - P_4$, od koder dobimo:

$$\hat{s}(t) = 4s(t-1) - 6s(t-2) + 4s(t-3) - s(t-4). \tag{4.19}$$

Napaka e_4 je potem podana z enačbo:

$$\begin{aligned} e_4(t) &= s(t) - \hat{s}(t) \\ &= s(t) - 4s(t-1) - 6s(t-2) + 4s(t-3) - s(t-4). \end{aligned} \tag{4.20}$$

Zapis FLAC lahko shranimo v datotečni format FLAC ali v katerega izmed vsebnikov, na primer Ogg ali Matroska (MKV). FLAC predpisuje način hrambe toka podatkov FLAC, ki ga lahko shranimo neposredno v datoteko, kot je opisano v specifikaciji RFC 9639 [252]. Datoteka se začne z zlogi ASCII »fLaC«, nato sledi blok z metapodatki toka podatkov (na primer vzorčevalna frekvenca, bitna globina, število kanalov). Nato lahko sledijo dodatni metapodatki, na primer komentarji in slike. Podobno kot pri drugih datotečnih formatih vsak blok na začetku vsebuje med drugim oznako bloka in velikost v zlogih. Med dodatnimi bloki, ki jih je smiselno omeniti,

je **preiskovalna tabela** (angl. seek table), ki vsebuje sinhronizacijske kode, s katerimi se je možno premakniti na poljubno mesto v stisnjenem nizu, torej omogoča hitro **preiskovanje vzorcev** (angl. seeking). FLAC namreč vzorce stiska z različnimi stopnjami stiskanja, kar predvajalniku glasbe oteži preskakovanje glede na čas. Sinhronizacijske kode so podane v obliki kazalcev s preslikavo iz zaporedne številke vzorca na odmik v datoteki. Več kot je kazalcev, hitrejše preskakovanje dobimo.

Zatem sledijo **okvirji** (angl. frame), ki vsebujejo bloke vzorcev. Vsak okvir vsebuje toliko blokov oziroma **podokvirjev** (angl. subframe), kot je kanalov. Vsak okvir ima glavo, ki določa velikost blokov in ponovno osnovne parametre (na primer frekvenca vzorčenja, bitna globina, število kanalov, način odstranjevanja korelacije med kanali). Ta redundanca je potrebna v primeru uporabe FLAC za pretočno glasbo, pri čemer se lahko med predvajanjem spreminja na primer vzorčevalna frekvenca. Nato sledijo podokvirji oziroma bloki, po en na kanal. Če imamo dva kanala in je definirana velikost bloka 4.608 vzorcev, potem je dejanska velikost okvirja 9.216 vzorcev. Na začetku bloka je zapisan način kodiranja bloka (na primer konstantna napoved ali LPC), nato sledijo napake napovedi. Na koncu okvirja je še koda CRC, s katero lahko zaznamo napako med prenosom toka podatkov.

4.4.4 WavPack

WavPack [253] je odprtokoden algoritem, ki je namenjen brezizgubnemu stiskanju, omogoča pa tudi skoraj brezizgubno in tako imenovano hibridno stiskanje. WavPack je razvil David Bryant, ki je opis podal v [7]. Vzorci so lahko veliki do 32 bitov, podpira tako celoštevilčni zapis kot zapis s plavajočo vejico. Tudi WavPack razdeli vhod v bloke, ki so lahko mono ali stereo. WavPack podpira vse možne hitrosti vzorčenja.

WavPack deluje v treh korakih: odstranitev korelacije med kanali, odstranitev korelacije med vzorci na podlagi večkratne uporabe napovedi in nato kodiranje entropije. WavPack podpira tudi skoraj brezizgubni način, ki pa je zgolj na nivoju kvantizacije vzorcev v časovni domeni. Hibridni način kombinira izgubni in skoraj brezizgubni način v povezano celoto. V tem primeru tvori dve datoteki. Prva datoteka hrani zapis z izgubami, druga pa *popravke*, ki, če jo kombiniramo s prvo datoteko, v celoti rekonstruira podatke in zagotavlja brezizgubno predvajanje.

Popularna metoda za brezizgubno stiskanje avdia je še Monkey's Audio, ki pa že uporablja tehnike strojnega učenja. Opis najdemo v [7].

4.4.5 Družina avdiostandardov MPEG

V osemdesetih in devetdesetih letih se je začela digitalizacija tudi na področju videa in avdia. Pomemben cilj je bil podpreti shranjevanje videa in avdia celotnega filma na podatkovne nosilce, ki so bili takrat na voljo. Za shranjevanje avdia je hitro postalo jasno, da brezizgubno stiskanje zaradi časovne potratnosti ne pride v poštev in je nujno treba uporabiti izgubno stiskanje z izločitvijo tistih informacij, katerih izgube ne zaznamo. MUSICAM (angl. Masking pattern adapted Universal Subband Integrated Coding And Multiplexing) [254] je bil eden izmed prvih postopkov za izgubno stiskanje avdia na podlagi dognanj iz psiohoakustike. Raziskovalci so že leta 1988 demonstrirali oddaljen prenos avdia, stisnjenega z MUSICAM, z uporabo različnih nosilcev. Predvidena bitna hitrost pri frekvenci vzorčenja 48 kHz je bila od 64 kbit/s do 192 kbit/s na kanal, kar je občutno znižanje prostorske zahtevnosti, saj dosega stopnjo stiskanja od 4 do 12 v primerjavi s kodiranjem LPCM pri 16-bitni globini vzorcev.

Leta 1988 je nastala skupina MPEG (angl. Moving Picture Experts Group), katere cilj je bil standardizacija predstavitve in stiskanja digitalnega avdia in videa. Skupina MPEG-Audio je že naslednje leto testirala takrat 14 obstoječih algoritmov za stiskanje avdia [7]. Na podlagi dognanj iz predhodnih algoritmov so v letih 1991 in 1993 najprej pripravili osnutek standarda MPEG-1 Audio (ISO/IEC 11172-3 [255]), ki vključuje tudi dobro znan format MP3. Skupina MPEG je nato nadaljevala z razvojem in v letu 1997 izdala standard ISO/IEC 13818-7 [256] za format AAC (angl. MPEG-2 Advanced Audio Coding). V letu 2009 je nato izdala nadgradnje AAC v okviru MPEG-4. Standardi v okviru skupine MPEG so patentirani, a njihova patentna zaščita je že potekla.

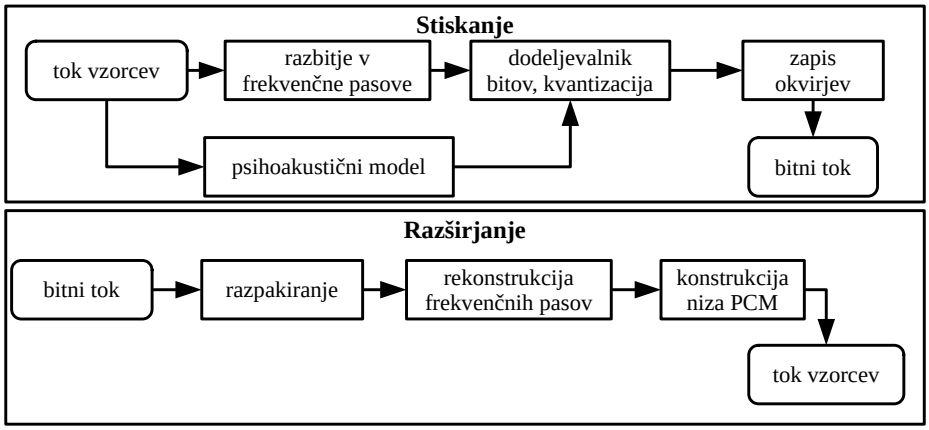
4.4.6 MPEG-1 Audio

Standard MPEG-1 Audio opisuje tri različne načine stiskanja [8, 7], ki jih imenujemo **nivoji** (angl. layers) ter označujemo kot **nivo I** (**MP1**), **nivo II** (**MP2**) in **nivo III** (**MP3**). Nivoji so kompatibilni navzgor; nivo III, na primer, lahko dekodira nivo I in nivo II. Najzahtevnejši za kodiranje je nivo III, ki dosega zelo dobro kakovost stiskanja. Na srečo je dekodirnik relativno preprost, zato je nivo III postal izjemno popularen, bolj znan pod imenom MP3. MP3 je eden največjih uspehov konzorcija MPEG.

Ključni cilj MPEG-1 Audio je bil opremiti digitalni video še z zvočno informacijo, zato je bila dovoljena količina stisnjenih podatkov določena vnaprej. Uporabnik torej vnaprej določi stopnjo stiskanja v obliki bitne hitrosti, ki je lahko od 32 do 448 kbit/s v primeru MP1 in do 320 kbit/s

v primeru MP3. Posledično so faktorji stiskanja zelo visoki (tudi do 24). Predstavitev podpira en ali dva kanala (stereo), ki sta lahko dekorelirana ali kodirana neodvisno drug od drugega (na primer za večjezično podporo). Podprte vzorčevalne frekvence so 32, 44,1 ali 48 kHz. Z MPEG-1 smo dobili kodiran zapis avdia, katera kakovost ni bila bistveno slabša od kakovosti glasbene plošče CD ob bistveno manjši porabi pomnilniškega prostora.

Osnovna ideja stiskanja MPEG-1 Audio nivoja I in II je zelo podobna postopku MUSICAM, pri čemer je nivo I poenostavitev nivoja II, s čimer omogoča nižjo zakasnitev kot nivo II. Zvočni posnetek, shranjen v določenem frekvenčnem pasu, na primer med 0 in 48 kHz, najprej razdelimo v bloke oziroma **okvirje** (angl. frame). Nato izvedemo psihoakustično analizo in vsak okvir transformiramo tako, da ga razbijemo na več frekvenčnih podpasov. Vsak podpas okvirja posebej zapišemo v izhodno datoteko, pri čemer zanj določimo stopnjo kvantizacije oziroma število bitov za zapis **signala** (v terminologiji MPEG so signali števila, ki smo jih dobili po predhodni transformaciji). Stopnjo kvantizacije določimo na podlagi psihoakustične analize in **algoritma dodeljevanja bitov** (angl. bit allocation). Na primer, če je avdio v določenem frekvenčnem podpasu pod slušnim pragom zaznavanja, mu dodelimo manj bitov. MPEG nivoja III nadgradi idejo z dodatnimi koraki stiskanja. Posebna pozornost MPEG je namenjena videu, zato mora biti dekodirnik avdia zelo hiter in enostaven. Dekodirnik tako ne uporablja nobenega psihoakustičnega modela in ne algoritma dodeljevanja bitov. Splošno shemo kodirnika in dekodirnika MPEG-1 avdio nivojev I in II kaže slika 4.7.



Slika 4.7: (a) Kodirnik MPEG-1 avdio in (b) dekodirnik (nivoja I in II)

Poglejmo si postopek stiskanja podrobneje na primeru nivojev I in II. V

prvem koraku vhodni signal razbijemo na okvirje, ki so dolgi 384 vzorcev pri nivoju I in 1152 vzorcev pri nivojih II in III, da lahko kodirnik še učinkoviteje zakodira vzorce. V naslednjem koraku signal razbijemo na 32 frekvenčnih podpasov, pri čemer so ti enako široki in pokrivajo celotni vhodni frekvenčni spekter. To razbitje izvedemo na podlagi banke filtrov. **Banka filtrov** je skupek filtrov, ki razbijejo vhodni signal na več ločenih podpasov, ki skupaj pokrivajo celotno območje slišnih frekvenc. Pri MPEG nivojih I in II v banki filtrov uporabljajo **večfazni filter** (angl. polyphase filter), to je nabor 32 filtrov z enako pasovno širino. V primeru nivoja I dobimo 12 vzorcev znotraj vsakega izmed 32 frekvenčnih podpasov, v primeru nivojev II in III pa je vzorcev v frekvenčnem podpasu 36.

Naslednji korak kodirnika je analiza avdia na podlagi psihoakustičnega modela. Standard predlaga 2 psihoakustična modela, pri čemer se preprostejši uporablja za višje bitne hitrosti, kompleksnejši pa za nižje. Osnovna ideja pa je, da za vsak frekvenčni podpas izračunamo **razmerje med signalom in pragom maskiranja** – SMR (angl. Signal-to-Mask Ratio), pri čemer upoštevamo slušno prekrivanje na podlagi frekvenčnega in časovnega maskiranja. SMR potem uporabimo pri določanju, ali je signal glasnejši od praga maskiranja, na podlagi česar izvedemo ustrezno kvantizacijo.

4.4.6.1 Algoritem dodeljevanja bitov

Jedro stiskanja avdia MPEG je kvantizacija, ki bi jo bilo pravilneje imenovati **dodeljevanje bitov** (angl. bit allocation). Dovoljena bitna hitrost je znana vnaprej, zato kodirnik vsak trenutek ve, koliko bitov lahko dodeli kvantiziranim signalom. To nalogo opravlja tako imenovani algoritem dodeljevanja bitov (angl. bit allocation algorithm). Glavna naloga algoritma je določitev števila bitov za zapis signala za vsak frekvenčni podpas posebej. Tako lahko zagotovimo, da je zvočni signal prenašan s konstantno hitrostjo (na primer 128 kbps), biti pa so dodeljeni tistim frekvencam, ki so ključne v signalu.

Algoritem dodeljevanja bitov deluje tako, da najprej za okvir izračuna dodeljeno število bitov [8]:

$$bitov\ /\ okvir = \frac{bps}{okvirjev/s}, \tag{4.21}$$

kjer je:

$$okvirjev/\ s = \frac{vzorcev\ na\ sekundo}{vzorcev\ na\ okvir}. \tag{4.22}$$

Algoritem dodeljevanja bitov zmanjša dodeljeno število bitov za količino bitov, ki so namenjeni opisu okvirja. V primeru skladbe s frekvenco vzorčenja 48 kHz in bitno hitrostjo 320 kbit/s bi za posamezen okvir nivoja I imeli na

voljo 2560 bitov. Ko upoštevamo še velikost glave okvirja, nam preostane okoli 2000 bitov za zapis signala.

Število dodeljenih bitov za vsak frekvenčni podpas nato postavimo na vrednost 0, velikost **bazena bitov** (angl. bit pool), ki so še na voljo za dodeljevanje bitov, pa na velikost okvirja brez glave. Nato tistim frekvenčnim podpasovom, ki najbolj potrebujejo višjo natančnost zapisa, postopoma dodeljujemo bite iz bazena bitov. Bit najprej dodamo tistemu frekvenčnemu podpasu, ki ima najnižje **razmerje med slušnim pragom in šumom** – *MNR* (angl. Mask-to-Noise ratio). MNR za vsak podpas izračunamo kot:

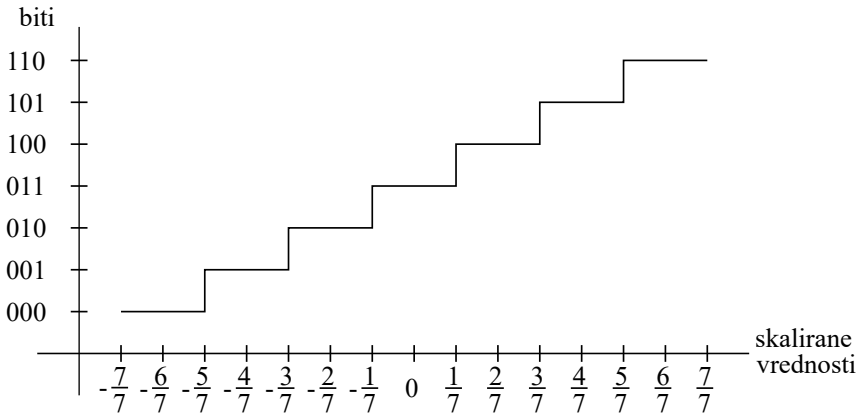
$$MNR = SQNR - SMR. \tag{4.23}$$

Standard MPEG-1 [257] vsebuje tabele z ocenami *SQNR* glede na število dodeljenih bitov. **Šum** pri *MNR* in *SQNR* pa pomeni napako zaradi kvantizacije s trenutno dodeljenim številom bitov. Torej šum zaradi kvantizacije oziroma premajhnega števila bitov z dodeljevanjem bitov znižujemo pod slušni prag. Celoten postopek ponavljamo, dokler še imamo bite na razpolago. Dodeljevanje bitov je smiselno ponavljati tako dolgo, dokler še slišimo šum oziroma izgube, torej dokler še obstaja frekvenčni podpas, ki ima $MNR < 0$. Če je $MNR \geq 0$ pri vseh frekvenčnih podpasovih, bi pri idealnem psihoakustičnem modelu dosegli transparentnost stiskanja.

Poglejmo si primer. Frekvenčnemu podpasu smo že dodelili določeno število bitov in tako dobimo $MNR > 0$, $SQNR > 0$ in $SMR > 0$. To pomeni, da imamo vrednost signala nad slišnim pragom, šum pa je pod slišnim pragom, zato frekvenčnemu podpasu ni treba dodeliti nobenega bita. Če je $MNR < 0$, $SQNR > 0$ in $SMR > 0$, je signal nad slišnim pragom, šum pa je tudi nad slišnim pragom, zato moramo temu frekvenčnemu podpasu dodeliti vsaj en bit. Če je $MNR > 0$, $SQNR > 0$ in $SMR < 0$, sta tako signal kot šum pod pragom slišnosti, zato temu frekvenčnemu podpasu ni treba dodeliti nobenega bita. Povzemimo: z dodelitvijo bitov zmanjšujemo količino šuma in s tem večamo *SQNR* in *MNR*.

Delitev avdia na okvirje in posamezne frekvenčne podpasove prinaša še eno prednost. Natančnost zapisa vrednosti signalov v posameznih frekvenčnih podpasovih izboljšamo tako, da jih skaliramo, torej raztegnemo oziroma skrčimo. Če se v nekaterih frekvenčnih podpasovih nahajajo zgolj tišji signali, jih normaliziramo na večje območje, ki ga lahko z dodeljenim številom bitov predstavimo. Tako povečamo natančnost predstavitve signalov. Skaliramo tako, da najprej poiščemo največjo vrednost v frekvenčnem podpasu. To vrednost nato skaliramo čim bliže vrednosti 1 (a ne preko 1). S skalirno vrednostjo nato množimo še preostalih 11 oziroma 35 vrednosti v frekvenčnih podpasovih.

Število kvantizacijskih nivojev l in število bitov po kvantizaciji q za vsakega izmed 32 podpasov določa relacija $l = 2^q$. Slika 4.8 kaže kvantizacijo, ko je $q = 3$, vrednosti signalov pa so že skalirane na interval $[-1, 1]$. Vrednosti $[-\frac{3}{7}, -\frac{1}{7}]$ bodo na primer kvantizirane v bite 010. Ko bomo vrednosti rekonstruirali, bomo 010 interpretirali kot središče intervala, to je $-\frac{2}{7}$ v našem primeru.



Slika 4.8: Kvantizacija pri MPEG avdio (po vzoru [7, 8])

4.4.6.2 Zapis izhodnega toka MPEG-Audio I in II

Signali so v posameznih frekvenčnih podpasovih zbrani v okvirje (angl. frames), ki jih skaliramo in kvantiziramo glede na psihoakustični model in algoritem za dodelitev bitov. Kvantizirane vrednosti hkrati s podatki o kvantizaciji za vsak okvir in vsak frekvenčni podpas v okvirju zapišemo v izhodni stisnjen niz.

MPEG-1 na izhod zapiše zgolj zaporedje okvirjev. V nivoju I je vsak okvir sestavljen iz opisa okvirja, ki mu sledi 12 kvantiziranih vrednosti na podpas, skupaj 384 oziroma 32×12 vrednosti [8]. Znotraj frekvenčnega podpasu zapišemo vrednosti za vsak kanal posebej. Opis okvirja začnemo z 32 sinhronizacijskimi biti za glavo, ki najprej vsebuje 12 enic in nato 20 bitov za naslednje kodirne parametre:

- nivo MPEG-Audio (nivo I, II, ali III);
- uporaba odprave napak s CRC;
- hitrost prenosa podatkov (najnižja je 32 kb/s, najvišja 448 kb/s);
- hitrost vzorčenja (44,1 kHz, 48 kHz ali 32 kHz);

- uporaba zapolnjevanja bitov (angl. padding), ki lahko doda dodatne bite v bitni niz po določenem številu okvirjev;
- določitev stereonačina: stereo, **povezan stereo** (angl. joint stereo), dva ločena kanala, en kanal;
- izvor stisnjenega niza (originalni stisnjen niz ali kopija) in ali gre za avtorsko zaščiten avdio;
- način **poudarjanja** (angl. emphasis), torej ali je bil vhodni signal ob kodiranju spremenjen za zmanjšanje motenj. V tem primeru mora dekodirnik odstraniti poudarjanje.

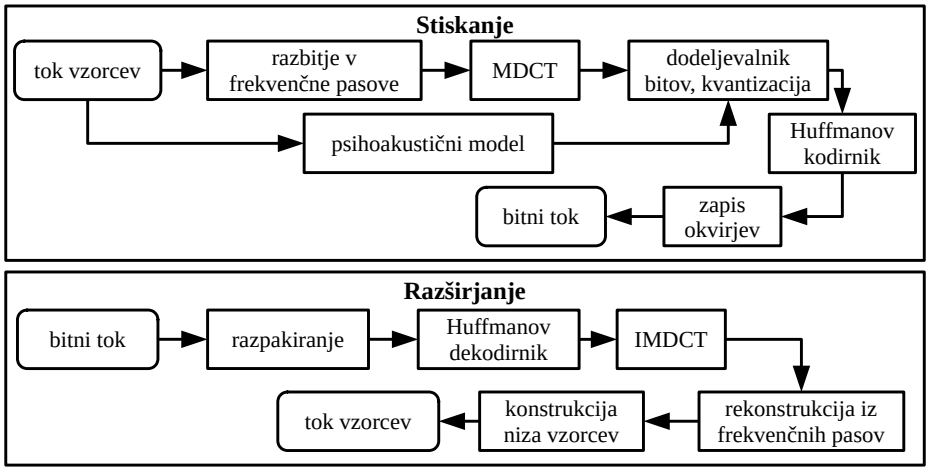
Če uporabljamo odpravo napak, sledi 16 bitov za kodo CRC. Sledi opis vzorcev frekvenčnih podpasov. Število bitov za zapis vrednosti na frekvenčni podpas zapišemo v okvir s štirimi biti. Biti 0000 določajo, da so vrednosti v frekvenčnem podpasu zapisane z enim bitom, vrednost 1110 pa, da so zapisane s 15 biti. Zaporedje bitov 1111 ni uporabljeno, da ne bi povzročili konflikta s sinhronizacijsko značko. Kadar kvantizator vrednosti v nekaterih frekvenčnih podpasovih postavi na 0, to označi v glavi okvirja. Štiribitno kodo potem ignoriramo. Nato sledi 32 6-bitnih vrednosti, ki določajo skalirne faktorje. Standard določa tabelo skalirnih faktorjev, zato zapiše indeks iz tabele. Na koncu sledi 384 oziroma 32×12 vrednosti signalov.

MPEG-1 nivo II zapiše podatke na podoben način z nekaj izboljšavami. Največja razlika nastane zaradi 36 vzorcev na frekvenčni podpas. Nivoja II in III imata namreč po 36 vrednosti na podpas, kar nam da 1152 vrednosti. Zaradi daljših frekvenčnih podpasov lahko kodirnik zapiše ločene skalirne faktorje za vsako tretjino frekvenčnih podpasov ali pa za vse tri dele uporabi en skalirni faktor. Naslednja izboljšava je, da za zapis stopnje kvantizacije pri določenih frekvenčnih podpasovih uporablja manj kot 4 bite.

4.4.6.3 Kodiranje nivoja III

Kodiranje nivoja III, torej MP3, se močno razlikuje od predhodnikov in posledično dosega tudi boljšo stopnjo stiskanja [258]. Razlika se začne že povsem na začetku, saj za večfaznim filtrom nad posameznimi okvirji takoj sledi **modificirana diskretna kosinusna transformacija** – **MDCT** (angl. modified discrete cosine transform), ki se izvede nad posameznimi frekvenčnimi podpasovi, kot je prikazano na sliki 4.9. To pomeni, da se kodiranje po transformaciji izvaja v frekvenčnem prostoru, torej podobno kot pri JPEG in DCT. Modificirana DCT se uporablja namesto klasične DCT za odpravo nenadnih skokov med okvirji, ki so značilni za DCT pri JPEG, saj upošteva vzorce iz sosednjih okvirjev. MDCT pri transformaciji

upošteva sosednje vzorce z **okensko funkcijo** (angl. window), ki bolj oddaljenim vzorcem iz sosednjih okvirjev dodeli manjšo vlogo pri transformaciji. Zaradi MDCT je tudi dekodirnik nivoja III zahtevnejši, saj mora realizirati inverzno MDCT (IMDCT). Kombinacijo uporabe večfaznih filtrov in MDCT imenujemo **hibridna banka filtrov** (angl. hybrid filter bank).



Slika 4.9: (a) Kodirnik MPEG-1 avdio in (b) dekodirnik (nivo III)

Uporaba MDCT pa prinaša eno težavo. Z MDCT nad celotnim frekvenčnim podpasom dosežemo boljšo frekvenčno ločljivost, vendar je časovna ločljivost slabša. Pride do tako imenovanega principa **negotovosti** (angl. uncertainty principle), kjer lahko ločljivost ene domene povečamo zgolj na račun druge. Časovna in frekvenčna domena sta namreč komplementarni. Krajši in glasnejši zvok (impulz) tako vpliva na celotno območje, ki ga transformiramo z MDCT. Nato pa, ko v naslednjem koraku izvedemo kvantizacijo koeficientov MDCT, dejansko povzročimo napake, razpršene v času kot slišne motnje. Tipičen moteč učinek je **vnaprejšnji odmev** (angl. pre-echo) prej omenjenega glasnejšega zvoka, ki ga zaradi našega časovnega prekrivanja prej opazimo kot poznejši odmev. Zaradi te težave lahko frekvenčni podpas v MP3 razdelimo na tri krajše bloke v času in MDCT izvedemo za vsak blok posebej. Dolgi bloki (okvirji) so primernejši za avdiosignale s stacionarno karakteristiko, krajši pa za tiste z izraženim spreminjanjem.

Nad posameznimi frekvenčnimi podpasovi okvirja oziroma posameznimi bloki (če so krajši) izvedemo MDCT, ki kot rezultat vrne polovico izhodov, kot je vhodov. Po tem koraku dobimo 576 ($32 \times 3 \times 12/2$) vrednosti oziroma koeficientov na okvir, ki ustrezajo 1152 avdiovzorcem okvirja. Da preprečimo nenadne skoke med mejami blokov, uporabimo pri izračunu MDCT zadnjo

polovico vzorcev iz trenutnega bloka in prvo polovico naslednjega. Tako dosežemo 50 % prekrivanja med bloki MDCT, kar velja tako za daljše kot krajše bloke. Vse to pa pri dekodiranju pomeni, da okvirji vsebujejo informacije o vzorcih iz naslednjega okvirja. Vsak okvir vsebuje torej 1152 koeficientov MDCT.

Nato sledi kvantizacija oziroma dodelitev bitov na podoben način kot pri nivoju II. Kvantizirane kode okvirja nazadnje stisnemo s Huffmanovim kodirnikom. Koeficiente nato uredimo in umestimo v tri regije. Vsako regijo zakodiramo z različnim naborom Huffmanovih kod, saj je pričakovana statistična porazdelitev različna v vsaki regiji. Vrednosti za visoke frekvence so pričakovano majhne in imajo daljša zaporedja ničel, vrednosti za nižje frekvence pa so večje. Huffmanove tabele predpisuje standard.

Nivo III prinaša še možnost odstranitve korelacije med povezanima kanaloma (angl. joint stereo), kjer namesto obeh kanalov shranimo njuno **vsoto** (angl. middle) in **razliko** (angl. side) [258]. Kodirnik poleg kodiranja s **konstantno hitrostjo** (angl. Constant bit rate – CBR) omogoča tudi **kodiranje z variabilno dolžino** (angl. Variable Bit Rate – VBR). To pomeni, da lahko preprostejše dele signala kodiramo z manj biti, prihranek pa uporabimo pri zahtevnejšem delu zvoka tako, da v povprečju ohranimo predpisano bitno hitrost.

Format zakodiranih okvirjev je podoben kot pri nivoju II [8], saj je glava okvirja enaka. Za glavo sledijo podatki za vsak kanal. Večja razlika nastane pri zapisu posameznih signalov. Vsak kanal je sestavljen iz dveh zrn (angl. granules), ki ustreza 576 koeficientom MDCT in se začne z dodatnimi informacijami: tip okna MDCT (dolgo ali kratko), uporabljene Huffmanove tabele za dele okvirja in opisi skalirnih faktorjev. Znotraj zrna sledijo skalirni faktorji in tok bitov iz Huffmanovega kodirnika.

Razvoj MPEG Audio se je nadaljeval tudi po letu 1993. V letu 1995 je izšel standard ISO/IEC 13818-3 [259] (MPEG-2 Part 3), ki predhodni standard nadgrajuje z naslednjimi novostmi:

- Dodaja podporo za število kanalov do 5.1, torej **prostorski avdio** (angl. surround) s petimi osnovnimi zvočniki in enim za nižje frekvence, pri čemer ohranja kompatibilnost s starejšim standardom (MP3).
- Dovoljuje uporabo dodatnih polovičnih frekvenc vzorčenja (16, 22,05 in 24 kHz) in več bitnih hitrosti (tudi 8 kbit/s).

4.4.6.4 ID3

Datoteke MP1, MP2, MP3 so shranjene kot zaporedje okvirjev, torej ne definirajo metapodatkov. V ta namen je nastala neuradna specifikacija

ID3 [260], ki omogoča vključitev metapodatkov tudi v datoteko MP3. Prva različica **ID3v1** je dovoljevala naslednje informacije: naslov, avtor, ime albuma, leto, komentar, žanr. Zapis je bil shranjen na koncu datoteke kot skupek 128 zlogov, pri čemer so se prvi trije zlogi začeli s črkami »TAG«. Nekateri starejši predvajalniki MP3 so te dodatne zloge interpretirali kot okvir in s tem predvajali šum. Nova specifikacija **ID3v2** je podprla dinamično velike zapise, kodiranje Unicode in podporo za slike (na primer za platnico albuma skladbe). Za razliko od predhodne specifikacije ID3v2 hrani metapodatke na začetku datoteke. Specifikacija posebno pozornost namenja zapisu podatkov tako, da predvajalnik ne zamenja značke za okvir z vzorci s tem, da se izogne zapisu zaporedja enic, značilnih za glavo okvirja MPEG Audio.

4.4.7 Stiskanje govora

Stiskanje govora izvedemo z namenskimi postopki, saj je pri govoru zelo malo višjih frekvenc. Želimo ohraniti prepoznavnost in razumljivost govora, hkrati pa čim bolj zmanjšati bitno hitrost za prenos. To je še posebej pomembno pri tehnologijah z manjšo bitno hitrostjo (na primer Bluetooth z majhno porabo energije, pri starejših telefonskih, radijskih omrežjih in mobilnih omrežjih). Ključna zahteva pri stiskanju govora je čim manjša **zakasnitev** (angl. latency) kodeka. Postopki, ki temeljijo na MDCT, imajo pogosto daljšo zakasnitev. MDCT uporablja daljše bloke in pri izračunu upošteva tudi sosednja bloka. V primeru blokov dolžine 1152 vzorcev in pri vzorčevalni frekvenci 32 kHz je zakasnitev reda 50 ms, kar pri pogovoru pomeni 100 ms dodatno zakasnjen odgovor.

Pri kodiranju govora se je na začetku uveljavilo kodiranje z redkejšim vzorčenjem, kvantizacijo in transformacijo. Standard **ITU-T G.711** [246, 7] iz leta 1972 filtrira vhodni signal na frekvenčni pas med 300 Hz in 3400 Hz, avdio vzorči z 8 kHz in vzorce kvantizira s 13 biti, ki jih nato transformira s transformacijo μ ali A. Predvidena bitna hitrost je 64 kbps, kar je bilo primerno za telefonska omrežja v preteklosti. Standard **ITU-T G.722** [261] pa že omogoča boljšo kakovost, saj vhodni signal filtrira v frekvenčni pas do frekvenc 7 kHz, vzorči s 16 kHz in kvantizira na 14 bitov. Stiskanje z ADPCM izvaja v dveh ločenih frekvenčnih podpasovih: za nižji podpas (kjer je večina pomembnih frekvenc) nameni več bitov kot za višji podpas, čemur pravimo **podpasovna ADPCM** (angl. Subband ADPCM) [262]. Pri novejših izvedbah, na primer G.722.1 [263], pa se že uvaja DCT.

Pri stiskanju govora obstaja tudi druga pomembna skupina algoritmov, ki stiska na podlagi napovedi in **sinteze govora** (**vokoder**) [7]. Takšno stiskanje uporablja model tvorbe govora, s katerim avdio govora najprej

analizira in nato opiše s parametri. Dekodirnik na podlagi parametrov rekonstruira govor (avdio). Takšna predstavitev govora zahteva bistveno nižje bitne hitrosti. Standard za telefonijo GSM ETS 300 961 (GSM 06.10) [264], znan pod imenom GSM-FR (angl. Full Rate) deluje na osnovi LPC in zahteva zgolj 13 kbps. Standard GSM-HR (angl. Half rate) oziroma ETSI EN 300 969 (GSM 06.20) [265] pa potrebuje le še slabo polovico. Uporabimo lahko tudi eno izmed variant kodiranja **AMR** (angl. Adaptive Multi-Rate), ki se prilagaja hitrosti prenosa. Najbolj znana varianta je **AMR-WB** (angl. Adaptive Multi-Rate Wideband) oziroma standard G.722.2 [266]. Med temi postopki je še smiselno omeniti tudi **MELP** (angl. Mixed Excitation Linear Prediction) [267], ki uporablja napovedi, potrebuje pa zgolj 300 bps (uporablja se za satelitsko komunikacijo in v vojaške namene).

4.4.8 MPEG-2 – AAC

AAC (angl. Advanced Audio Coding) so načrtovali kot zamenjavo za MP3. Leta 1997 je pod okvirjem **MPEG-2** izšel kot standard ISO/IEC 13818-7 [256], dve leti pozneje pa je izšla nadgradnja **MPEG-4** ISO/IEC 14496-3 [268]. AAC podpira frekvence vzorčenja od 8 do 96 kHz, do 48 kanalov, uvaja različne metode odpravljanja redundance med kanali ter vključuje dodatne metode stiskanja (na primer napovedi). Format AAC se najpogosteje uporablja v datotečnih formatih 3GP in MP4 (M4A). AAC je tudi znan po zaščiti za preprečevanje **nepooblaščenega kopiranja** za zaščito avtorskih pravic (angl. digital rights management – DRM). DRM bitne datoteke zašifrira, ključ pa zapiše v varno shrambo, do katere lahko dostopa zgolj določena naprava ali uporabnik.

AAC zaradi omejitev večfaznih filtrov uporablja zgolj MDCT za delitev na frekvenčne pasove. Podobno kot pri MP3 tvori različno velike bloke oziroma okna za MDCT od 256 do 2048 vzorcev. Na podlagi psihoakustičnega modela uvaja dodatne tehnike zmanjševanja redundance in odprave napak. V primeru daljših blokov omogoča zmanjševanje redundance z napovedmi, nato izvede kvantizacijo in vzorce zakodira s Huffmanovim kodirnikom z različnimi Huffmanovimi tabelami [269]. Obstajajo tri izvedbe kodiranja AAC, ki jih, v nasprotju z nivoju pri MPEG-1 Audio, imenujemo profili [7]:

- **AAC Main** predstavlja osnovno varianto kodirnika in uporablja večino algoritmov iz standarda.
- **AAC LC** (angl. low complexity) je poenostavljena varianta AAC in ne uporablja napovedi.
- **AAC SSR** (angl. scalable sampling rate) je namenjen prenosu s prilagajanjem bitne hitrosti.

Najbolj znane nadgradnje AAC so:

- **AAC LD** (angl. low delay) je standardizirana [268] varianta AAC za telefonijo. Omogoča kodiranje s kratko zakasnitvijo (angl. latency) in nizkimi bitnimi hitrostmi [270]. Ključ za krajšo zakasnitev so krajši bloki in ožje okno znotraj MDCT z upoštevanjem manjšega števila vzorcev iz sosednjih okvirjev.
- **AAC ELD** (angl. enhanced low delay) je nadgradnja AAC LD z večjo učinkovitostjo pri kodiranju višjih frekvenc.
- **HE-AAC** (angl. high–efficiency AAC) oziroma AAC+ je nadgradnja AAC-LC za pretočno predvajane glasbe [270], ki ponuja dinamično povečevanje bitne hitrosti za boljšo rekonstrukcijo višjih frekvenc pri nižji bitni hitrosti.

V okviru skupine standardov MPEG-4 je izšel tudi standard za brezizgubno stiskanje **MPEG-4 SLS** (angl. Scalable to Lossless) ISO/IEC 14496-3:2019. MPEG-4 SLS omogoča brezizgubno stiskanje s kombinacijo izgubnega stiskanja AAC in shranjevanja razlike do originala.

4.4.9 Vorbis

Vorbis je odprta specifikacija [271] za izgubno kodiranje avdia. Po zmogljivosti je primerljiv z AAC [272], vendar ni patentiran. Tudi tukaj so jedro kodirnika MDCT nad okvirji, uporaba psihoakustičnega modela, kvantizacija in Huffmanov kodirnik. Ključna razlika glede na AAC pa je delitev vhodnega signala v **aproksimacijo signala** (angl. floor) in **preostanek** (angl. residue), ki ju zakodira ločeno.

4.4.9.1 Ogg

Specifikacija Vorbis predpisuje postopek kodiranja, ne pa tudi načina zapisa zaporedja okvirjev in odprave napak. Zato se praviloma uporablja v kombinaciji z odprto specifikacijo Ogg [273], ki se razvija v isti skupini kot Vorbis. Ogg namreč dopolnjuje Vorbis z določitvijo načina hrambe okvirjev v datoteki in za **pretočno predvajanje** (angl. streaming). Ogg poleg Vorbis omogoča uporabo tudi drugih formatov kodiranja, na primer Opus in FLAC.

Bistvena značilnost Ogg je zapis avdia z zaporedjem časovno umeščenih **strani** (OggS), ki so velike nekaj kB in vsebujejo okvirje, na primer iz kodirnika Vorbis, in metapodatke. Vsaka stran se začne z zlogi ASCII »OggS«, nato sledijo osnovni parametri, na primer različica specifikacije, časovna umestitev, zaporedna številka, koda CRC in število vključenih

blokov. Na koncu je **tabela segmentov** (angl. segment table), ki vsebuje dolžine posameznih segmentov na koncu strani OggS. Dolžine segmentov zapišemo z osmimi biti, zato so lahko daljši paketi oziroma okvirji kodirnika sestavljeni iz več segmentov. Preiskovanje vzorcev oziroma navigacija po času znotraj datoteke Ogg je realizirana z **bisekcijo**, ki upošteva opise strani, da hitro najde ustrezno iskano stran.

4.4.10 WMA

WMA (angl. Windows Media Audio) je serija kodekov podjetja Microsoft [274]:

- *Windows Media Audio 9* omogoča vzorčevalne frekvence do 48 kHz, 16-bitno globino, stereo kanala in spremenljivo bitno hitrost.
- *Windows Media Audio 10 Professional* je izboljšana različica, saj omogoča vzorčenje z do 96 kHz, 24-bitno globino in do 8 kanalov (7 kanalov za prostorski avdio in ločen kanal za nižje frekvence).
- *Windows Media Audio 9 Lossless* je brezizgubna različica WMA, ki dosega razmerje stiskanja od $1:2$ do $1:3$.
- *Windows Media Audio 9 Voice* je različica WMA za telefonijo, torej z nizko zakasnitvijo in bitno hitrostjo.

WMA nima javno objavljenih specifikacij. Zaradi tega so nastala razna odprtokodna orodja, ki so s povratnim inženiringom podprla branje datotek WMA. Tako lahko v izvorni kodi dekodirnikov FFmpeg [275] opazimo, da WMA uporablja osnovne koncepte, ki smo jih že spoznali, torej MDCT, skalirne faktorje za frekvenčne pasove, odpravo redundance med kanali in Huffmanove tabele.

4.4.11 Dolby Digital

Podjetje Dolby Laboratories se ukvarja s profesionalnim avdiom. V preteklosti je predstavilo več formatov za kodiranje avdia. Najbolj znan format je **Dolby Digital** oziroma **AC3** [7] iz leta 1991 in je standardni format za DVD, BluRay (640 Kbps) in HDTV. Podpira 6 kanalov pri postavitvi 5.1.

Kodirnik AC-3 deluje z bloki dolžine 512 ali 256 vzorcev, nad katerimi izvede MDCT [7]. Kriterij izbire velikost blokov je podoben kot pri MP3. V naslednjem koraku odpravi redundanco med kanali. Vrednosti koeficientov predstavi z decimalnimi števili, kjer ločeno kodira **števko** (angl. mantissa) in potenco osnove 10 (angl. exponent) koeficientov, tako da kvantizacijo

z dodeljevanjem bitov izvaja nad števkami koeficientov. Stopnjo kvantizacije določa na podlagi psihoakustičnega modela. Kot zanimivost je treba omeniti, da tako kodirnik kot dekodirnik izvajata dodeljevanje bitov na podlagi poenostavljenega psihoakustičnega modela, ki se pri AC-3 imenuje hibridno prilagodljivo dodeljevanje bitov za nazaj/naprej (angl. hybrid backward/forward adaptive bit allocation) [276].

V letu 2005 je izšla nadgradnja **E-AC-3** (angl. enhanced AC-3) oziroma **Dolby Digital Plus** v obliki standarda ATSC A/52 z naknadnimi dopolnitvami [277], ki omogoča več kanalov in izboljšano stopnjo stiskanja ob hkratni ohranitvi združljivosti z AC-3.

Tudi **Dolby TrueHD** je del podjetja Dolby Technologies in je namenjen za brezizgubno stiskanje na podlagi formata **MLP** (angl. Meridian Lossless Packing), ki ga je sprva razvilo podjetje Meridian Audio, podjetje Dolby Laboratories pa je nato leta 1998 prevzelo nadzor nad licenciranjem [278, 279, 7].

Neposredna nadgradnja kodirnika AC-3 je **AC-4**, ki je podan v standardu ETSI TS 103 190-1 [280]. Omogoča še večjo fleksibilnost glede predstavitve kanalov in izboljšuje stiskanje, saj že bitna hitrost od 64 kbps do 96 kbps omogoča zadovoljivo stereokakovost [281].

4.4.12 Opus

Prva specifikacija Vorbis je bila javno objavljena že leta 2000 [271], zato je bilo po nekaj letih treba razviti njegovega naslednika. Pod okriljem organizacije Xiph.Org je v letu 2012 nastala specifikacija kodiranja avdia Opus, ki prinaša popolnoma nov postopek kodiranja. Opus je napredoval do predloga standarda RFC 6716 [282]. Bistvo kodeka Opus je visoka prilagodljivost pri kodiranju tako govora kot glasbe. Razvijalci Opusa so ugotovili, da so postopki na podlagi napovedi boljši za govor, za glasbo pa MDCT. Opus kombinira tehnike stiskanja s področja glasbe in govora, pri tem pa dosega dobro stiskanje pri manjših zakasnitvah, tako da po zmogljivosti presega AAC [283]. Opus deluje v treh načinih:

- Za kodiranje govora in ozkih frekvenčnih območij uporablja kodirnik SILK, ki uporablja napovedi (LPC).
- Za kodiranje glasbe je na voljo kodirnik CELT (angl. Constrained Energy Lapped Transform), ki uporablja MDCT in napovedi v frekvenčnem prostoru.
- Hibridni način uporablja CELT za visoke frekvence in SILK za nizke frekvence.

Tudi Opus deluje na nivoju okvirjev, zato lahko vsak okvir predstavi na zanj najprimernejši način. Dosega bitne hitrosti od 6 kbps do 510 kbps, pri čemer je zakasnitev kodirnika v območju med 5 ms in 65,2 ms. Opus predpisuje format zapisa okvirjev, ki se lahko zapišejo znotraj vsebnikov Matroska, WebM, MPEG-TS, MP4 in Ogg (praviloma s končnico .opus).

4.4.13 IAMF

IAMF (angl. Immersive Audio Model and Formats), znan tudi kot **Eclipsa Audio**, je odprta specifikacija za prilagodljivo shranjevanje avdia z večjim številom kanalov [284]. Avtor formata je skupina **AOM** (angl. Alliance for Open Media). IAMF določa način hrambe podatkov in ne načina stiskanja. Načrtovan je za scenarije, ko želimo shraniti prostorski avdio z več kanali z različnimi postavitvami mikrofonov/zvočnikov, hkrati pa lahko v datoteko IAMF shranimo zapis zgolj za slušalke. Tako lahko isto datoteko predvajamo na različnih napravah. Vključuje tudi ločene kanale za različne jezike.

IAMF je namenjen tudi za naprednejše aplikacije, kot je navidezna resničnost. IAMF omogoča zapis avdia neodvisno od postavitve zvočnikov tako, da opiše smer in jakost zvoka v celotnem prostoru z **ambisonično predstavitvijo** (angl. ambisonics). Za predvajanje takšnega prostorskega zvoka z na primer klasičnimi stereoslušalkami potrebujemo algoritme **upodabljanja za dve ušesi** (angl. binaural rendering), ki iz ambisonične predstavitve tvori LPCM za dva kanala slušalk. V primeru navidezne resničnosti upodabljalnik upošteva orientacijo slušalk v prostoru. Starejša alternativa IAMF je standard MPEG-H 3D Audio (ISO/IEC 23008-3 [285]).

4.5 Merjenje podobnosti med avdioposnetki

Ko govorimo o izgubnem stiskanju, hitro pridemo do vprašanja, kako meriti podobnost med zvočnimi posnetki. RMSE za avdio ni primeren zaradi podobnih razlogov kot pri slikah, torej zaradi načina zaznavanja zvoka pri človeku. Pri slikah se je SSIM izkazal za primerno rešitev. Pri merjenju zvoka pa moramo uporabiti znanja iz psihoakustike. Pri avdiu se je v preteklosti uveljavila metoda **PEAQ** (angl. Perceptual Evaluation of Audio Quality), ki je opisana v obliki priporočila BS.1387 [286]. Implementacija metode presega obseg tega učbenika, saj temelji na zahtevnih postopkih frekvenčne analize in psihoakustičnega modela. PEAQ kot izhod vrne podobnost v obliki vrednosti od 0 do -4, kjer velja:

- 0: neopazna razlika;
- -1: opazno, vendar ne moteče;

- -2: rahlo moteče;
- -3: moteče;
- -4: zelo moteče.

Z razvojem metod strojnega učenja pa so se začele pojavljati metode, ki model za oceno podobnosti izpeljejo iz realnih meritev [287].

4.6 Seznam nalog

- V aplikaciji za obdelavo avdia (na primer Audacity) ustvarite signal, kjer imajo vsi vzorci večjo in enako neničelno vrednost. Signal predvajajte in razmislite, kaj se dogaja z zvočnikom v tem primeru.
- V aplikaciji za obdelavo avdia (na primer Audacity) ustvarite ton s frekvenco 100 Hz in ga predvajajte. Nato ustvarite ton s frekvenco 200 Hz in ocenite, kakšna je razlika med njima. Nato ponovite enak postopek s frekvencama 1000 Hz in 1100 Hz. Kje prej opazite razliko 100 Hz?
- Napišite program, ki bo na podlagi krivulj enake glasnosti z obteževanjem A omogočil izpis glasnosti avdia na podani frekvenci in zvočnem tlaku.
- V aplikaciji za obdelavo avdia (na primer Audacity) ustvarite avdio z glasnejšim pokom. Nato 10 ms pred njim in 10 ms za njim ustvarite tišji pok. Preverite, v katerem primeru bolje slišite tišji pok.
- Izračunajte, koliko zlogov zahteva avdio LPCM, ki ga zapišemo v dveh kanalih s frekvenco vzorčenja 48 kHz in 16-bitno globino vzorcev.
- Napišite program, ki bo z naključnim poskušanjem sestavil polinom za prileganje krajšemu zaporedju vzorcev. Nato napišite program, ki bo avdiosignal predstavil s polinomom in na izhod zapisal razlike glede na vhodni signal. Preverite, s kakšno stopnjo stiskanja lahko stisnete razlike. Primerjajte uspešnost Huffmanovega kodirnika in kodirnika Golomb-Rice.
- Napišite program, ki bo iz datoteke WAV, kodirane z LPCM, izpisal amplitude vzorcev.
- Napišite program, ki bo iz datoteke FLAC izpisal napake napovedi. Pomagajte si z uradno specifikacijo FLAC.

- Napišite program, ki bo simuliral obnašanje dodeljevalnika bitov iz formata MP1 na podlagi naključnih vrednosti SMR in SQNR.
- Napišite bralnik datotek Ogg, ki izpiše metapodatke strani. Preverite, ali se prvi dve strani razlikujeta od preostalih.
- Napišite program, ki bo v datotekah MP3 preveril, ali imajo vsi okvirji enake osnovne parametre, kot je na primer frekvenca vzorčenja. V nekaterih okvirjih jo spremenite in preverite obnašanje predvajalnika.
- Poiščite orodje, ki z metodo PEAQ vrne podobnost med avdioposnetki. Nato primerjajte kodirnike, omenjene v učbeniku, pri različnih bitnih hitrostih. Narišite grafikone podobnosti glede na bitno hitrost za vsakega izmed njih.

Poglavje 5

Video

Video je zaporedje slik (pri videu jih bomo imenovali **okvirji** – angl. frames), ki jih prikažemo dovolj hitro. Pionirji videa so eksperimentalno ugotovili, da je za iluzijo gibanja najmanjša hitrost prikazovanja 15 sličic na sekundo (angl. frames per second – **fps**), zato so bili prvi posnetki opravljeni s hitrostjo 16 fps. A za sunkovite gibe je bila ta hitrost premajhna, zato so hitrost predvajanja povišali na 24 fps, hitrost, ki je ostala v uporabi še do danes.

Začetki snemanja in predvajanja videa segajo v 19. stoletje. Takrat je Eadweard Muybridge s skoraj hkratno prožitvijo 24 fotoaparatov ustvaril zaporedje slik, ki ga je bilo možno predvajati kot video [288]. Takšne fotografije je pozneje namestil na prozoren steklen disk, iz katerega je med vrtenjem na platno projiciral gibajoče slike. To idejo so pozneje uporabili pri prvih filmih, ki so jih posneli na celuloidni trak. Ti filmi so bili nemi, z razvojem elektronike pa so na filmski trak dodali avdio v obliki optičnega zapisa zvočnega valovanja (angl. waveform), torej v analogni obliki. Trak, navit v kolute, so fizično prinesli na mesto predvajanja. Rokovanje s trakom je zahtevalo posebno pozornost, saj je bil trak zelo dolg. Če je višina sličice 7,49 mm, potrebujemo pri 24 fps za enourni zapis 647 m filmskega traka.

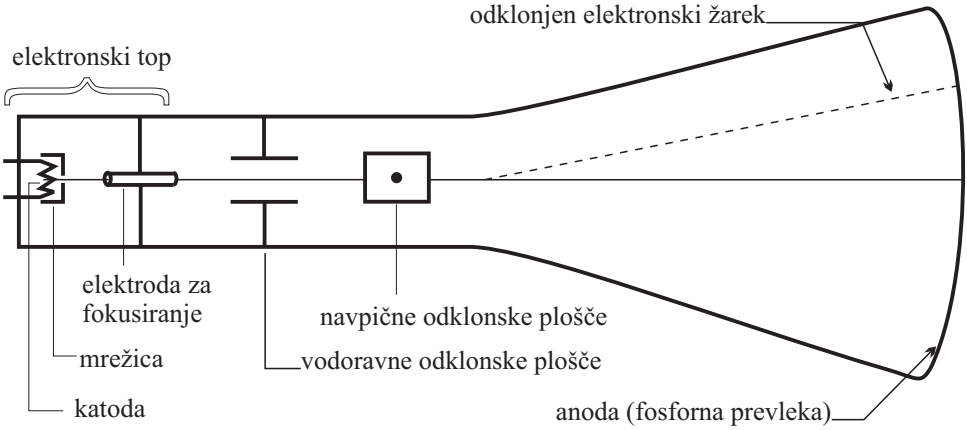
5.1 Analogna televizija

Ideja o prenosu slike na daljavo je bila ključna za množico inovacij ob koncu devetnajstega in na začetku dvajsetega stoletja, ki so privedle do razvoja analogne televizije in njene poznejše digitalizacije. Televizija je bila za tiste čase revolucionaren izdelek, ki je rešil naslednje težave:

- zajem slike s kamero,
- predstavitev slike z napetostnim signalom, torej analognim signalom,

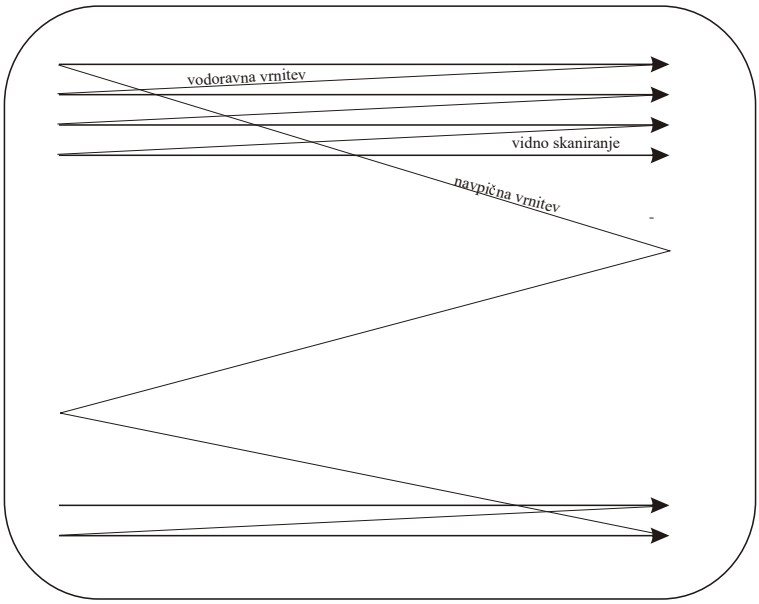
ki ga lahko moduliramo na visokofrekvenčni nosilni signal, tega pa v eter oddajamo z oddajnikom,

- sprejem in demoduliranje signala iz visokofrekvenčnega nosilca,
- prikaz slike iz videosignala na ustrezno napravo ter
- sinhronizacijo zapisa zvočne in vidne informacije.



Slika 5.1: Shema klasične katodne cevi (vir: [2])

Da bomo razumeli razlike med analognim in digitalnim videom, si najprej poglejmo delovanje prikazovalnika s katodno cevjo, ključne naprave prvih televizorjev. Princip delovanja je naslednji [2]: ko se **katoda** (slika 5.1) segreje, začne oddajati elektrone. **Anoda** se nahaja neposredno na prikazovalni površini in je pozitivno naelektrena, zato privlači elektrone. Da bi elektroni dobili dovolj energije, je anoda na zelo visokem napetostnem potencialu glede na katodo. Elektrone, ki izhajajo iz katode, usmerjamo (pravimo tudi, da jih fokusiramo) s primerno oblikovanimi elektrodami v ozek curek, ki ima dovolj veliko kinetično energijo, da vzbudi atome fosforja, s katerim je prevlečena prikazovalna površina. Ko se atomi fosforja vračajo v osnovno energetsko stanje, oddajajo fotone v vidnem delu spektra, kar vidimo kot sliko. Neposredno za katodo je nameščena **krmilna mrežica**, ki nadzoruje količino elektronov, ki zadanejo anodo. Katodo, mrežico in elektrodo za fokusiranje imenujemo **elektronski top** (angl. electron gun). Žarek elektronov običajno odklanjamo elektromagnetno (s tuljavicami, nameščenimi na vratu katodne cevi) ali, redkeje, z vodoravnimi in navpičnimi odklonskimi ploščami, ki so vgrajene v katodno cev.



Slika 5.2: Rastrsko prebiranje analognega videosignala (vir: [2])

Curek elektronov premikamo po vzoru rastrskega prebiranja (slika 5.2) tako, da ga usmerjamo od leve proti desni od zgoraj navzdol. Po koncu zadnje vrstice žarek elektronov vrnemo na začetek prve vrstice. Žarek je po vsaki vrstici oziroma po koncu vseh vrstic zatemnjen (z negativno napetostjo na mrežici). Proces premikanja žarka in njegovega zatemnjevanja nadzorujeta dva oscilatorja žagastega signala. Oscilator z višjo frekvenco skrbi za vodoravno odklanjanje (tako imenovana vodoravna odklonska frekvenca), drugi, z nižjo frekvenco, pa za navpično odklanjanje (navpična odklonska frekvenca). Ti dve frekvenci tako določata ločljivost slike. Krmilna mrežica, ki stoji takoj za katodo, skrbi za to, da elektrone, ki izhajajo iz katode, spustimo v preostali del katodne cevi samo takrat, ko želimo na nekem mestu vzbuditi fosfor. Z napetostjo na krmilni mrežici tako nadziramo jakost svetlobe na točno določenem delu slike. Proces rastrskega prebiranja mora biti povsem skladen z oddajnikom (oziroma z zajemanjem slike s kamero); pravimo, da mora biti rastrski proces **sinhroniziran**.

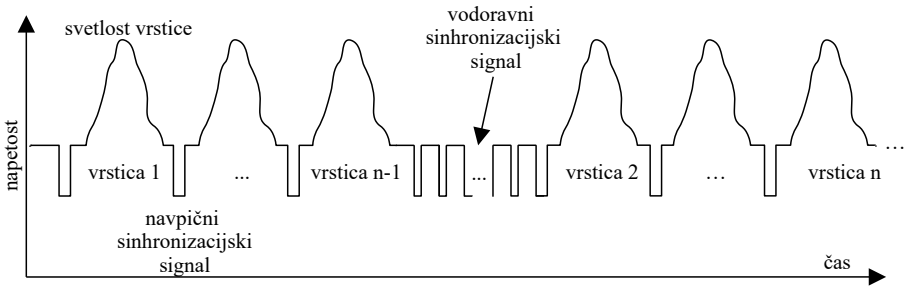
Analogna televizijska kamera s katodno cevjo (angl. video camera tube) se od prikazovalnika s katodno cevjo razlikuje v tem, da ima namesto anode s fosforno prevleko ploščo s fotoupori in lečo za fokusiranje svetlobe na to ploščo. Fotouporom se upornost spreminja glede na jakost svetlobe, močnejša svetloba povzroči zmanjšanje upornosti, s čimer dobimo različne napetostne nivoje glede na osvetlitev posameznih fotouporov. Te napetostne

nivoje nato moduliramo in z uporabo oddajnika pošljemo televizijskim sprejemnikom.

5.1.1 Analogni videosignal

V preteklosti so se v različnih delih sveta uveljavili različni analogni videostandardi [289]:

- **NTSC** (angl. National Television System Committee): Severna in del Južne Amerike ter Japonska,
- **SECAM** (fra. Séquentiel de couleur à mémoire): Francija, Rusija, del Zahodne Afrike in
- **PAL** (angl. Phase Alternating Line): večina Evrope, Afrika, Avstralija, del Južne Amerike.



Slika 5.3: Analogni videosignal

Ti standardi med seboj niso združljivi. Razlikujejo se po hitrosti osveževanja sličic (okvirjev), številu vrstic ter načinu kodiranja slike in avdia. V Evropi, kjer je frekvenca električnega omrežja 50 Hz, sistem PAL uporablja hitrost osveževanja 25 sličic na sekundo (fps) in sliko s 625 vidnimi vrsticami. Podobno se je NTSC prilagodil frekvenci omrežne napetosti 60 Hz tako, da je hitrost osveževanja 30 fps, slika pa ima 480 vidnih vrstic. Osnovno strukturo analognega videosignala prikazuje slika 5.3. V vsaki vrstici amplituda signala določa svetlost, med vrsticami in sličicami pa so vstavljeni sinhronizacijski impulzi, ki omogočajo pravilno sinhronizacijo prikazovalnika. Vsi trije standardi uporabljajo **prepletanje** (angl. interlacing), kjer se slika na zaslonu prikazuje z dveh **polj** (angl. fields), ki imata polovično navpično ločljivost. V prvem polju se osvežijo lihe vrstice, v drugem pa sode, kar navidezno poveča frekvenco osveževanja. V primeru PAL tako dobimo 50 polj na sekundo.

Sprva je bila na voljo le črno-bela televizija. Pozneje so razvili tudi barvno televizijo. Način razvoja pa demonstrira izjemen primer nadgradnje prenosa informacij s popolno združljivosti za nazaj. Barvno televizijo so namreč uvedli tako, da so uporabniki starejših črno-belih televizorjev lahko še naprej spremljali televizijski program v sivinah, čeprav je signal vseboval tudi vso potrebno informacijo za prikaz barv. Uporabniki novejših barvnih televizijskih sprejemnikov pa so lahko spremljali tako barvni kot črno-beli videosignal (večina filmov je bila takrat posneta še v črno-beli tehniki). Nadgradnjo črno-belega signala so namreč izvedli s kodiranjem barvnih informacij v višje frekvence videosignala. Na primer, pri standardu NTSC so namesto enostavnega signala, kjer amplituda določa svetlost, uporabili signal višje frekvence. Pri tem **faza** (začetna točka signala) določa barvni **odtenek** (angl. hue), jakost nihanja pa **barvitost** (angl. saturation). Signal višje frekvence je bil načrtovan tako, da je povprečna napetost oziroma amplituda še vedno predstavljala svetlost slike, kar je ohranilo združljivost s črno-belimi televizorji.

Pri analogni televiziji govorimo o **sestavljenem** videosignalu (angl. composite video), saj so vse informacije o sliki kodirane hkrati. V računalništvu pa se je uveljavil **razčlenjen videosignal** (angl. component video), ki uporablja ločene signale za rdeči, zeleni in modri barvni kanal. Najbolj znan primer je priključek **VGA** (angl. Video Graphics Array). Kombinacijo sestavljenega in razčlenjenega videosignala predstavlja priključek **S-video**, ki svetlost (luminanco) in barvitost (krominanco) prenaša v ločenih nosilcih.

5.2 Digitalna predstavitev videa

Digitalni video dobimo ali z vzorčenjem analognega videa ali, danes že praviloma, neposredno iz digitalne kamere. Digitalni video ima naslednje glavne prednosti pred analognim:

- lahko ga prenašamo po poljubnih digitalnih prenosnih kanalih ali zapišemo na poljuben računalniški pomnilni medij,
- lahko ga kopiramo brez izgube,
- ponuja večjo fleksibilnost predstavitve videa (na primer različne ločljivosti),
- lahko ga urejamo s programskimi orodji neposredno na računalniku.

Z razvojem digitalnega videa se je spremenila tudi hitrost prikazovanja okvirjev in se prilagodila posameznim aplikacijam (na primer, od nekaj

fps za nadzorne kamere pa do specialnih kamer, ki zajamejo tudi nekaj milijonov slik na sekundo). Za običajno uporabo je najpogostejša hitrost okvirjev 24, 25, 30 ali 60 v sekundi. Podobno smo dobili večjo fleksibilnost glede ločljivosti videa. Pri digitalnem videu se za določanje ločljivosti videa najpogosteje uporabljajo naslednje oznake:

- 480i (SD – angl. Standard-definition): ločljivost 640×480 iz standarda NTSC, pri čemer je v vsakem okvirju zapisana polovica vrstic (prepleten način);
- 576i (tudi SD): ločljivost 720×576 iz standarda PAL (prepleten način);
- 720p (HD – angl. High Definition): ločljivost 1280×720 v **napredujočem** (angl. progressive) načinu brez prepletanja;
- 1080p (Full HD): ločljivost 1920×1080 brez prepletanja;
- 1080i (Full HD): ločljivost 1920×1080 s prepletanjem;
- UltraWide HD: ločljivost 2560×1080;
- 2160p oziroma 4K (Ultra HD): ločljivost 3840×2160;
- 4320p oziroma 8K (Ultra HD): ločljivost 7680×4320.

Poleg prednosti pa pri digitalnem videu kaj hitro naletimo tudi na množico težav ter znanstvenih in inženirskih izzivov:

- **Prostorska zahtevnost**. Količina podatkov, ki jih moramo shraniti pri digitalni predstavitvi videa, je spoštovanja vredna. En okvir pri 720p ima 921.600 pikslov, kar nam pri 24-bitni globini da 22.118.400 bpf (angl. bits per frame) ali 2,7 MBpf (angl. Mega bytes per frame). Pri 2160p dobimo že 24,8832 MBpf. Pri frekvenci osveževanja 30 Hz potrebujemo za eno sekundo videa 720p 83 MB, za eno uro pa 300 GB. Takšna količina podatkov je izziv še danes tako pri predvajanju v realnem času kot hrambi. Zato je ključ digitalnega videa uspešno stiskanje podatkov, ki ga bomo bolj podrobno obravnavali v nadaljevanju. Pri stiskanju videa imamo dejansko opravka z vsaj tremi multimedijskimi tipi: poleg zaporedja slik še tekst in zvok. Za vsakega izmed njih potrebujemo svojo metodo stiskanja, a video predstavlja največji izziv. Za prenos ene ure stereoavdia z vzorčevalno frekvenco 44,1 kHz in 16-bitno globino bi potrebovali *le* 635,04 MB prostora.
- **Kodiranje in dekodiranje v realnem času.** Pri snemanju z digitalno kamero želimo video kodirati sproti, kar zahteva dovolj

veliko računsko moč ali strojno podprte rešitve. Najbolj zahteven korak kodiranja videa pa je stiskanje. Če je postopek preveč zahteven, sprotno kodiranje v realnem času ni možno. Enako velja za dekodiranje, pri čemer je večina formatov, podobno kot pri avdiu, **asimetričnih**, kjer je dekodiranje manj zahtevno.

- **Prenos videa**. Stisnjenje podatke videa in avdia ob zajemu predstavimo z ločenimi multimedijskimi podatkovnimi tokovi, ki jih nato **uvrstimo** (angl. sequencing) v zaporedje bitov, te pa razdelimo v pakete, primerne za prenos oziroma predvajanje. Vsakega izmed prisotnih podatkovnih tokov mora kodirnik enolično označiti, zato vsak paket opremimo z glavo s potrebnimi metapodatki. Postopek uvrščanja poskrbi, da so elementarni podatkovni tokovi pravočasno dostavljeni dekodirniku.

- **Napake** (angl. errors). Pri prenosu podatkov in tudi shranjevanju lahko prihaja do napak v obliki izgube podatkov ali napačnih podatkov. Veliko napak je možno odpraviti pri protokolih prenosa na nižjih nivojih (na primer prenos videa z uporabo protokola TCP), v vseh primerih pa ne (na primer prenos videa z uporabo protokola UDP ali začasno sesutje oddajnika). Sprejemnik mora biti sposoben takšne napake zaznati in omiliti na način, da je degradacija signala čim manj moteča in da ne vpliva na nadaljnje dekodiranje (angl. error recovery).

- **Zakasnitve.** Prenos digitalnega videa mora potekati na takšen način, da lahko sprejemnik čim prej dekodira prejete podatke, torej da ima nizko zakasnitvev (angl. latency). Pri nekaterih aplikacijah, kot so na primer videoklici, je to še posebej pomembno, zato mora biti način kodiranja temu posebej prilagojen.

- **Pasovna širina** (angl. bandwidth). Pasovna širina za prenos informacij je omejena. Oddajni sistem mora zagotoviti, da je ob prenosu bitnega niza ne prekorači. Ta izziv rešuje **nadzornik hitrosti** (angl. rate control) in tako poveča **prožnost** sistema (angl. skalability). Prožnost je lahko prostorska (manjša ločljivost videa), časovna (manj fps) ali kvalitativna (višja stopnja izgubnega stiskanja).

- **Preskakovanje** (angl. seeking). Pri prenosu digitalnega videa v živo preskakovanje seveda ni možno. Pri predvajanju vnaprej pripravljenih vsebin pa ga potrebujemo. Način kodiranja videa mora biti takšen, da omogoča preskakovanje videa. Uporabljamo lahko podobne koncepte kot pri avdiu (na primer **iskalna tabela** – angl. seek table in/ali ustrezni metapodatki v okvirjih in paketih).

- **Urejanje.** Če želimo video urejati, potrebujemo ustrezno predstavitev videa, da ga lahko preskakujemo in nato urejamo posamezne okvirje tako, da ni treba takoj ponovno kodirati celotnega videa. Zavedati se moramo, da uporaba že izgubnega stiskanja ob nekajkratnem nalaganju in shranjevanju lahko opazno okvari videz izvornega videa/slik (angl. generation loss [290]).

- **Dolgoročna dostopnost.** Ob dolgoročnem shranjevanju (arhiviranju) videa je pomembno ohraniti dostopnost ob poznejši uporabi. Postopki kodiranja videa, ki so opisani v specifikacijah, ali odprtokodni postopki omogočajo dolgoročnejšo dostopnost.

- **Avtorska zaščita in verodostojnost.** Že s pojavom analognega videa so nastali postopki za preprečitev nepooblaščenega kopiranja, na primer iz kaset VHS. Tudi za kodiranje digitalnega videa so, podobno kot pri digitalnem avdiu, zasnovali postopke za zaščito avtorskih pravic (angl. digital rights management – DRM).

- **Strojna in sistemska neodvisnost** (angl. multiplatform). Bitni niz bo dekodiran na različnih napravah z različnimi arhitekturami in operacijskimi sistemi. Manj zmogljive naprave morajo zato dobiti video v slabši ločljivosti, na primer v nižji ločljivosti (**prostorska prožnost**).

Ko danes omenimo video, pomislimo na digitalno televizijo ali na spletni/pretočni video. Video prikažemo na specializiranih napravah (seveda so to televizorji) ali na računalniških prikazovalnikih. Še nedavno se je za prenos videosignala na računalnikih uporabljala analogna predstavitev videa s priključkom **VGA** (angl. Video Graphics Array), saj so bili računalniški prikazovalniki analogni; uporabljali so katodno cev. Šele s pojavom digitalnih prikazovalnikov (na primer prikazovalniki LCD – angl. liquid-crystal display) je postal priključek VGA odveč. Prva alternativa VGA za prenos digitalnega videosignala je bil **DVI** (angl. Digital Visual Interface) oziroma DVI-D (angl. digital) ali DVI-I (angl. integrated), ki hkrati omogoča digitalni in analogni videosignal. Danes sta najpogostejša standarda **HDMI** (angl. High-Definition Multimedia Interface) in **DP** (angl. DisplayPort), ki omogočata bistveno večjo hitrost prenosa podatkov (reda nekaj deset Gb/s).

Kot zanimivost omenimo, da je tudi za prenos signala digitalne televizije v uporabi več standardov [291]:

- **ATSC** (angl. Advanced Television Systems Committee): Severna Amerika,

- **DVB** (angl. Digital Video Broadcasting): Evropa, Avstralija, del Azije in Afrike,
- **ISDB** (angl. Integrated Services Digital Broadcasting): Japonska, Južna Amerika in del Afrike,
- **DTMB** (angl. Digital Terrestrial Multimedia Broadcast): Kitajska.

DVB uporabljamo v Evropi in je razdeljen na naslednje specifikacije:

- **DVB-T** (angl. Terrestrial): prenos s pomočjo zemeljskih oddajnikov,
- **DVB-H** (angl. Handheld): prenos za mobilne naprave,
- **DVB-C** (angl. Cable): prenos s pomočjo kabelskega omrežja,
- **DVB-S** (angl. Satellite): prenos s pomočjo satelitov.

Pri analogni televiziji je znotraj ozkega frekvenčnega prostora umeščen en kanal. Bistvena razlika glede na analogno televizijo je, da pri digitalni televiziji lahko v isti frekvenčni pas umestimo več kanalov, ki jih moramo zato **multipleksirati** (angl. multiplex), s čimer povečamo izrabo frekvenčnega prostora.

Pri digitalni televiziji ne smemo pozabiti na televizijo z uporabo internetnega protokola (angl. Internet Protocol TV – **IPTV**). IPTV deluje tako, da ponudnik IPTV oziroma internetnih storitev (angl. internet service provider – ISP) po svojem zaprtem omrežju prenaša digitalni video do uporabnikovega sprejemnika z uporabo IP-protokola. Pri IPTV se televizijski kanali (ki vključujejo video, zvok in dodatne metapodatke) multipleksirajo in odpošljejo kot internetni paketi. Pogosto se za prenos uporablja transportni tok MPEG, katerega specifikacije so opisane v priporočilu H.222 [292] in ga bomo povzeli v nadaljevanju. Za nadziranje prenosa najdemo tudi druge protokole, na primer **RTP** (angl. Real-time Transport Protocol), **RTSP** (angl. Real-Time Streaming Protocol) in **RTCP** (angl. RTP Control Protocol) [233].

Eden glavnih izzivov pri IPTV je pasovna širina na strani uporabnika, ki je bila še nedavno precej manjša, kot je zahtevala tradicionalna analogna in digitalna televizija. Zato sprejemnik IPTV, v nasprotju s sistemom, kot je DVB, ne prejema vseh kanalov hkrati. Uporabnik v omrežju ponudnika pošlje zahtevo za prenos želenega televizijskega kanala. Ponudniki televizije seveda ne pošiljajo ločenega podatkovnega toka vsakemu uporabniku posebej, saj bi to preobremenilo omrežje, ampak uporablja razpošiljanje (angl. multicast) v okviru protokola **IGMP** (angl. Internet Group Management Protocol). Tako ponudnik znotraj zaprtega omrežja pošlje videopakete hkrati več uporabnikom. Posamezen uporabnik pa se prijavi na določeno

skupino v omrežju, ki ji pripada želeni program. Opisano seveda velja za prejemanje televizijskih programov v živo. Neposredno pošiljanje videopaketov posameznemu uporabniku (angl. unicast) pa se najpogosteje uporablja pri videu na zahtevo (angl. video on demand – **VOD**).

Na koncu ne smemo pozabiti na **pretočno** (angl. streaming) oziroma internetno televizijo/video, kjer se videopaketi prenašajo po internetu. Tudi tukaj je ključni izziv omejena in spremenljiva pasovna širina, ki jo rešujeta stiskanje videa in prožnost videa, kar bomo podrobneje spoznali v nadaljevanju.

5.3 Koncepti stiskanja digitalnega videa

Kot smo že omenili, se moramo tako pri prenosu kot shranjevanju digitalnega videa spopasti z veliko količino podatkov, ki jo je treba za nemoteno delovanje aplikacij močno zmanjšati. Principi stiskanja digitalnega videa, ki jih bomo spoznali v tem podpoglavju, so sicer preprosti, a dejanske implementacijske podrobnosti so zelo zapletene in presegajo namen tega učbenika. Bralec, ki ga zanimajo podrobnosti, bo našel dodatno gradivo na mnogih spletnih straneh, v člankih in knjigah [293, 294, 7].

5.3.1 Podvzorčenje okvirjev in prepletanje

Pri **podvzorčenju okvirjev** (angl. frames subsampling) kodirnik shrani samo vsak drug okvir, dekodirnik pa vsak prebran okvir podvoji. Pri tej tehniki pa žal zaznamo manj gladko gibanje, kar vsaj delno rešujemo s tehnikami **interpolacije gibanja** (angl. motion interpolation). Podvzorčenje okvirjev se redko uporablja, saj z njim zmanjšamo količino podatkov zgolj za faktor 2. Se pa uporablja interpolacija gibanja, in sicer za doseganje občutka gladkejšega gibanja pri obstoječih posnetkih, pri katerih bi želeli dvigniti hitrost prikazovanja okvirjev iz na primer 25 fps na 50 fps.

Prepletanje (angl. interlacing) je zelo pogost koncept, ki izhaja iz klasične televizije. V lihih okvirjih hranimo samo lihe vrstice, v sodih pa sode. Tudi v tem primeru zmanjšamo količino potrebnih podatkov zgolj za faktor 2, a tehnika je vseeno zelo pogosta.

5.3.2 Podvzorčenje barv

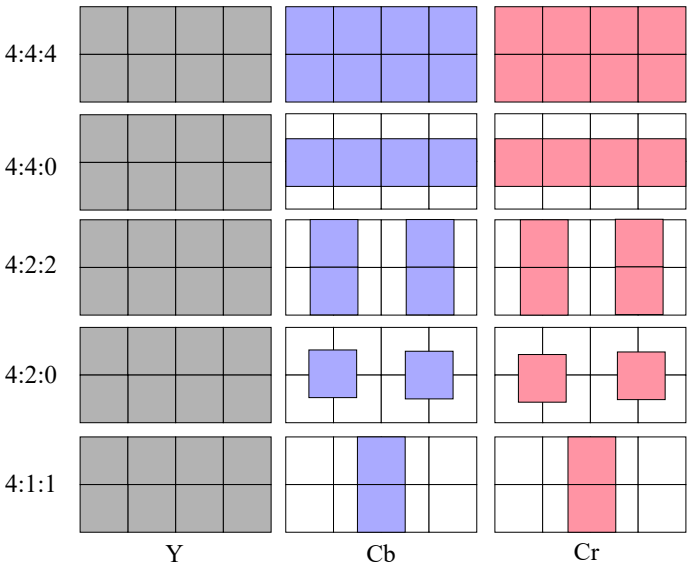
Kot vemo iz stiskanja slik, je človeški vidni sistem veliko bolj občutljiv za jakost svetlobe kot za razločevanje barv. Zato okvir, ki je predstavljen z barvnim modelom RGB, pretvorimo v model YCbCr (na primer po priporočilu ITU-R BT.601 [295]). Komponenta **svetlosti** Y (angl. luminance or luma)

prinaša večino informacije slike. Kontrast je namreč tisti, s katerim določamo obliko objektov na sceni. **Barvitost** (angl. chrominance) pa je določena z dvema komponentama Cr in Cb. Barvitost je seveda potrebna za popolno dojemanje slike, a ima manjši vpliv na zaznavo vsebine. Pri digitalnem videu je zato smiselno izvesti **podvzorčenje komponent YCbCr** (angl. colour subsampling), kjer močneje podvzorčimo kanale barvitosti, s čimer lahko s sprejemljivimi izgubami za človeško oko zmanjšamo prostorsko zahtevnost enega okvirja. Način podvzorčenja opisujemo z notacijo $y : c_h : c_v$. Prva številka nam pove velikost vzorca v vodoravni smeri (praviloma je $y = 4$) in s tem število komponent svetlosti Y. Drugi dve števili se nanašata na barvitost in sta določeni relativno glede na prvo številko. c_h določa vodoravno povzorčenje barvnih komponent kot število sprememb barvitosti glede na y, c_v pa navpično podvzorčenje kot število sprememb barvitosti iz prve v drugo vrstico. Pri digitalnem videu najdemo naslednje možnosti:

- **Način 4:4:4.** V tem primeru barvni komponenti nista podvzorčeni in vsak piksel je sestavljen iz treh komponent. Primer vidimo na sliki 5.4.
- **Način 4:4:0.** Barvni komponenti imata polovično ločljivost v navpični smeri (zahtevata polovico prostora kot komponenta Y).
- **Način 4:2:2.** Komponenti Cr in Cb imata polovično ločljivost v vodoravni smeri (zahtevata polovico prostora kot komponenta Y).
- **Način 4:2:0.** Barvni komponenti imata polovično ločljivost v obeh smereh oziroma prinašata samo 1/4 razpoložljive informacije.
- **Način 4:1:1.** Komponenti Cr in Cb imata le četrtino ločljivosti v vodoravni smeri (zahtevata četrtino prostora glede na komponento Y).

Pri opisovanju podvzorčenja barv moramo biti pozorni tudi na terminologijo. Pri kodiranju videa se za piksle pogosto uporablja beseda **pel** (angl. picture element). Pomen za piksel/pel pa je odvisen od konteksta. Pel lahko označuje piksel barvnega kanala slike ali piksel, ki ga vidimo na zaslonu. Pri podvzorčenju barv imamo manjšo ločljivost za barvni komponenti, zato posamezne vrednosti iz kanalov YCbCr imenujemo tudi **vzorci** (angl. samples).

Vrednosti kanalov C_bC_r so glede na priporočilo H.262 za kodiranje videa [13] namenjene kvantizirani predstavitvi barv pri digitalnem videu. Izračunamo jih na podlagi vrednosti $Y'P'_BP'_R$, ki so namenjene zvezni predstavitvi barv (za analogni signal). Oznaka $'$ pomeni, da smo vrednosti normalizirali, in sicer $Y' \in [0, 1]$ in $C'_B, C'_R \in [-0{,}5, 0{,}5]$ [13]. Tudi barvne komponente R, G in B so normalizirane v območje med 0 in 1.



Slika 5.4: Podvzorčenje barv

Glede na priporočilo za HDTV ITU-R BT.709 [296], na katerega se navezuje H.262 [13], izračunamo vrednosti $Y'P'_BP'_R$ kot:

$$\begin{aligned} Y' &= 0{,}7154G' + 0{,}0721B' + 0{,}2125R', \\ P'_B &= -0{,}386G' + 0{,}500B' - 0{,}115R', \\ P'_R &= -0{,}454G' - 0{,}046B' + 0{,}500R'. \end{aligned} \tag{5.1}$$

Obstajajo pa tudi druge preslikave, na primer priporočilo za digitalno televizijo ITU-R BT.601 [295]. Kam preslikamo intervale ob kvantizaciji, je odvisno od uporabe. Glede na priporočilo H.262 [13] za 8-bitne vrednosti Y' preslikamo v interval $[16, 235]$, kjer je 16 črna in 235 bela, P'_B in P'_R pa v interval $[16, 240]$:

$$\begin{aligned} Y &= 219\,Y' + 16, \\ C_b &= 224\,P'_B + 128, \\ C_r &= 224\,P'_R + 128. \end{aligned} \tag{5.2}$$

5.3.3 Odprava prostorske in časovne korelacije

Vse do zdaj naštete ideje omogočajo zgolj omejeno stopnjo stiskanja, kar glede na prostorsko zahtevnost videa ni dovolj. Kot vemo iz stiskanja slik, obstaja prostorska korelacija med piksli posameznega okvirja. S postopki izgubnega

stiskanja, najpogosteje na osnovi DCT, dosežemo razmerje stiskanja okoli 1 : 20. Govorimo tudi o stiskanju znotraj okvirja (angl. **intra**-frame).

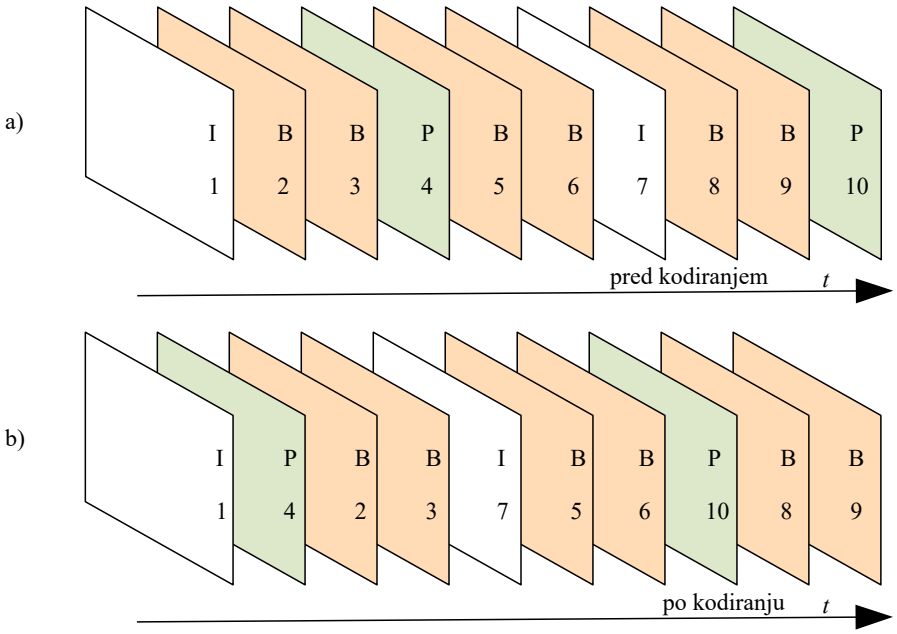
Pri videu pa obstaja tudi časovna korelacija med istoležnimi piksli zaporednih okvirjev. V tem primeru govorimo o stiskanju **med okvirji** (angl. **inter**-frame), ki ga realiziramo s **shranjevanjem razlik**. Ker se piksli v zaporednih okvirjih praviloma zelo malo razlikujejo, so razlike majhne in jih lahko učinkoviteje zakodiramo (na primer z aritmetičnim kodiranjem ali kodiranjem Golomb-Rice).

5.3.4 Hkratna odprava prostorske in časovne korelacije

Pri stiskanju videa pa je seveda smiselno odstraniti tako prostorsko kot časovno korelacijo, torej tisto **znotraj okvirja** (angl. **intra**-frame) in **med okvirji** (angl. **inter**-frame). Klasičen pristop k stiskanju videa odpravlja oba tipa redundance na naslednji način: prvi okvir stisnemo z eno izmed metod za stiskanje rastrskih slik (zelo pogosto JPEG z izgubami). Zatem za nekaj naslednjih okvirjev poiščemo razlike med njimi in **referenčnim okvirjem** ter zakodiramo samo razlike. Če je pri nekem okvirju razlik preveč, potem okvir ponovno zakodiramo kot samostojno sliko. Takšen okvir imenujemo **vstopni** ali kodiran znotraj (angl. intra frame). Okvirji, ki jih kodiramo glede na razliko z njihovim predhodnikom, pa so **vmesni** okvirji (angl. inter frame).

Kodiranje videa je skoraj vedno izgubno. Izgube se akumulirajo, zato prihaja do razširjanja napak. Če je bil na primer vstopni okvir kodiran z izgubami, bodo izgubo podedovali tudi vsi vmesni okvirji. Prav zaradi tega videa ne moremo zakodirati tako, da bi bil vstopni okvir samo eden, to je prvi okvir. Vstopni okvirji so pogosto razporejeni enakomerno v zaporedju okvirjev. Vse **vstopne okvirje** označimo z I (angl. intra), **vmesne** napovedane okvirje pa s P (angl. predictive). Koncept zatem lahko posplošimo in predpostavimo, da lahko nekatere okvirje kodiramo tako s predhodnim kot naslednjim okvirjem. Takšnim **obojestransko napovedanim** okvirjem pravimo okvirji B (angl. bidirectional). Da bi lahko okvir B dekodirali, moramo poznati tako predhodni kot naslednji okvir I ali P. Zato moramo ob kodiranju spremeniti vrstni red zapisanih okvirjev. Na sliki 5.5a vidimo primer vrstnega reda okvirjev pred kodiranjem in po njem (slika 5.5b). Za dekodiranje na primer okvirja 3 moramo predhodno dekodirati okvirja 1 in 4. Žal pa s tem pristopom povečamo zakasnitev. Zaporedje od okvirja I do zadnjega pred okvirjem I imenujemo **skupina slik** (angl. group of pictures – GOP).

Opisan koncept skoraj vedno vključuje naslednji dve nadgradnji, ki ju opisujemo po vzoru iz [7].



Slika 5.5: Vrstni red okvirjev (a) pred kodiranjem in (b) po njem, torej pred predvajanjem

5.3.5 Shranjevanje razlik blokov

Pri videu okvir skoraj vedno razdelimo v bloke, nato pa v najbolj preprosti varianti primerjamo istoležne bloke referenčnega in kodiranega okvirja, čemur pravimo **kodiranje razlik blokov** (angl. block differencing). Če se vsaj en piksel v kodiranem bloku Y razlikuje za več kot predpisan prag glede na istoležni blok X v referenčnem okvirju, zapišemo razlike celega bloka. V tem primeru najprej podamo koordinate bloka, nato pa sledijo razlike.

Pri hkratni uporabi podvzorčenja in razdelitve okvirjev na bloke moramo paziti na ustrezno terminologijo. Pri kodiranju znotraj okvirja se zelo pogosto uporablja transformacija DCT nad bloki velikosti 8×8 pikslov, ki jim pravimo tudi **transformirani bloki** (angl. transform block). Za shranjevanje razlik pa so se bolje izkazali večji bloki oziroma **makrobloki** (angl. macroblock). V priporočilu H.262 [13] je makroblok velikosti 16×16 pikslov kanala svetlosti; makroblok predstavlja torej 4 bloke. Novejši postopki stiskanja (na primer H.265 [297]) pa že uvajajo dinamično velike bloke. Zaradi tega bomo v nadaljevanju pri opisu specifičnih postopkov stiskanja uporabljali oznako makroblok, pri splošnih konceptih pa bomo ostali pri oznaki blok.

5.3.6 Kompenzacija gibanja

V zaporednih okvirjih najdemo zelo pogosto dele scene, ki se niso spremenili, morda so se le nekoliko premaknili (tipičen primer je premik kamere ali večjih objektov). Kodirnik s **kompenzacijo gibanja** (angl. motion compensation) najprej razdeli kodiran okvir v bloke Y in nato za vsak Y poišče najbolj podoben blok X v referenčnem (predhodnem) okvirju. Kodirnik zapiše razliko položajev blokov X in Y kot (Δ_x, Δ_y), kar imenujemo **vektor gibanja** (angl. motion vector). V praksi poleg vektorjev gibanja včasih shranjujemo še razlike med bloki X in Y.

Kompenzacija gibanja je najbolj učinkovita pri premikanju delov scene vzporedno z ravnino kamere, težje pa jo uporabimo v drugih primerih (sprememba osvetljenosti, deformacije, približevanje in tudi obračanje objektov). Za upoštevanje teh primerov obstajajo naprednejše rešitve [298], ki pa presegajo namen tega učbenika. A kljub vsemu je kompenzacija gibanja eden najpomembnejših konceptov pri stiskanju digitalnega videa, zato jo bomo v nadaljevanju opisali podrobneje.

Ko načrtujemo postopek stiskanja na podlagi kompenzacije gibanja, moramo upoštevati več vidikov, ki vplivajo na hitrost kodiranja in stopnjo stiskanja.

5.3.6.1 Segmentacija okvirjev

Premikajoči se deli scene lahko imajo poljubno obliko, a se vsaj pri starejših pristopih zadovoljimo s preprostejšo rešitvijo. Okvir razdelimo oziroma segmentiramo (angl. frame segmentation) v enako velike neprekrivajoče se bloke. Pri izbiri velikosti blokov moramo biti pazljivi. Pri premajhnih blokih porabimo preveč prostora za hrambo vektorjev premika, pri prevelikih blokih pa redkeje naletimo na popolno ujemanje z referenčnim blokom, posledično razlike med blokoma zahtevajo preveč prostora. Dejansko bi bilo najbolje za statične dele scene uporabiti večje bloke, za dinamične pa manjše. Kot dober kompromis se je pri starejših kodirnikih uveljavila velikost 8×8 ali 16×16 pikslov, novejši pa že omogočajo dinamično velikost.

5.3.6.2 Izbira ujemajočega bloka

Po segmentaciji moramo za vsak blok najti **ujemajoči se blok** v referenčnem okvirju (angl. block matching), pri tem pa potrebujemo ustrezen kriterij za primerjavo. **Mera ujemanja** (angl. distortion measure) mora biti dovolj zanesljiva in hitro izračunljiva. Seveda lahko uporabljamo mere podobnosti med slikami, ki smo jih spoznali v podpoglavju 3.2.4.1 (na primer

MSE, RMSE, PSNR, SSIM), pri čemer moramo upoštevati, da želimo najti ujemanje bloka čim hitreje.

Obstaja tudi **klasifikator razlike pel** (angl. pel difference classification – PDC), ki šteje zgolj razlike med vzorci blokov X in Y, ki so pod uporabniško določenim pragom. Nekoliko zahtevnejša je **integralna projekcija** (angl. integral projection), ki je definirana kot:

$$IP = \sum_{i=1}^{b} \left| \sum_{j=1}^{b} X_{i,j} - \sum_{j=1}^{b} Y_{i,j} \right| + \sum_{j=1}^{b} \left| \sum_{i=1}^{b} X_{i,j} - \sum_{i=1}^{b} Y_{i,j} \right|, \tag{5.3}$$

kjer je b velikost bloka. Integralna projekcija izračuna razlike med seštetimi elementi vrstic in stolpcev blokov X in Y. Na takšen način izračun pohitrimo, saj seštete elemente blokov iz referenčnega okvirja lahko ponovno uporabimo pri več blokih v kodiranem okvirju.

5.3.6.3 Kodiranje vektorjev premika in razlik med bloki

Pri manjših blokih lahko vektorji premika zavzamejo veliko prostora, zato jim moramo kodirati čim bolj učinkovito. A tokrat mora biti stiskanje brezizgubno. Pri tem lahko predpostavimo, da imajo sosednji bloki najpogosteje zelo podobne vektorje premika in so potemtakem ti korelirani. To velja tudi za vektorje premikov istoležnih blokov med sosednjimi okvirji. Zaradi tega je smiselno vektorje premika med okvirji napovedovati iz istoležnega bloka predhodnega okvirja in/ali predhodnega bloka v trenutnem okvirju. Napovedovanje izvedemo na podoben način, kot smo spoznali pri napovednem kodiranju slik.

Tudi pri kodiranju razlik med bloki upoštevamo pričakovano porazdelitev podatkov. Pričakujemo lahko veliko majhnih, pri hitro se premikajočih objektih bodo spremembe večje.

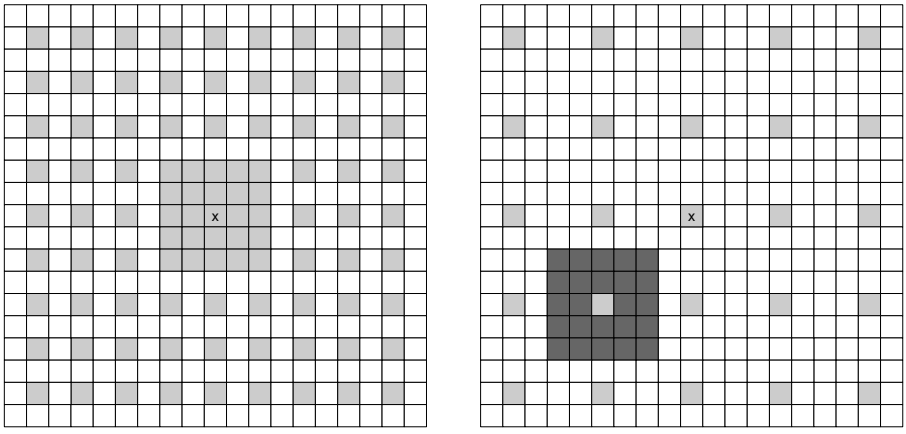
5.3.6.4 Iskanje blokov

Postopek iskanja referenčnih blokov (angl. reference block search) je lahko časovno zelo zahteven, saj moramo za kodiranje vsakega bloka poiskati najbolj primeren blok v referenčnem okvirju. Blokov znotraj okvirjev pa je zelo veliko, kar upočasni postopek določanja ujemanja pri polnem iskanju (angl. Exhaustive Search). Smiselna rešitev je iskanje bloka samo znotraj manjšega **iskalnega področja**, ki ga določimo z maksimalnim odmikom v smeri x in y, glede na vrednosti parametrov d_x in d_y. Iskalno področje obsega $(2d_x + 1)(2d_y + 1)$ pikslov in s tem blokov, ki jih moramo primerjati za kodiranje vsakega izmed blokov. Če je X kvadrat velikosti b, potem bi

pri polnem iskanju morali računati razlike med $(2d_x + 1)(2d_y + 1)b^2$ pari pikslov za vsak kodiran blok, kar lahko zahteva preveč časa, še posebej pri aplikacijah sprotnega kodiranja. Da bi zmanjšali prostor iskanja in s tem pohitrili iskanje vektorja premika, so raziskovalci predlagali različne rešitve. Te metode morebiti ne najdejo najboljšega ujemanja, a omogočajo občuten prihranek časa. Pri veliko aplikacijah je čas pomembnejši od morebiti nekoliko večje napake. Opis najpomembnejših metod je prirejen po [7]:

1. Pri **iskanju na podlagi podpisa** (angl. signature) vse bloke predstavimo s podpisom, ki je lahko zgolj eno ali več števil (na primer pri integralni projekciji imamo vsote po stolpcih in vrsticah). Nato iščemo bloke s podobnim podpisom, kot ga ima kodiran blok. Na takšen način se izognemo primerjavi posameznih pikslov blokov. Za bolj natančno iskanje pa lahko bloke z najpodobnejšimi podpisi primerjamo tudi po pikslih.

2. Največje ujemanje lahko pričakujemo pri manjših premikih, zato je smiselno bolj natančno preiskovati zgolj bližnjo okolico. Oddaljene bloke preiskujemo manj natančno; pravimo, da **redčimo razdalje** (angl. distance-diluted search). Primer pri velikosti iskanja $d_x{=}d_y{=}9$ vidimo na sliki 5.6a. Z naivno metodo bi morali pregledati $(2d_x + 1) \times (2d_y + 1) = 361$ blokov. Če z razdaljo 3 povečamo korak na 2, pridemo na 97 primerjav. To metodo seveda lahko kombiniramo z iskanjem na podlagi podpisa.



a) b)

Slika 5.6: Iskanje bloka (a) z metodo redčenja razdalje in (b) lokalno iskanje, kjer × na sredini slike označuje položaj kodiranega bloka.

3. Pri **lokalnem iskanju** (angl. locality-based search) najprej z grobim iskanjem preiščemo širšo okolico in tam, kjer najdemo najboljše ujemanje, nadaljujemo s podrobnejšim preiskovanjem. Na sliki 5.6b pri velikosti iskanja 9 pregledamo zgolj 25 blokov, nato pa okrog najbolj podobnega bloka preiščemo bližnjo okolico 24 blokov. Skupaj torej 49 blokov.

4. Lokalno iskanje lahko nadgradimo z učinkovitejšimi vzorci iskanja, ki se dobro obnesejo v različnih aplikacijah. Zelo znano je **dvodimenzionalno logaritmično iskanje** (angl. two-dimensional logarithmic search), kjer algoritem deluje v naslednjih korakih:

   (a) Naj bo začetna točka iskanja (x_1, y_1) na začetku enaka položaju bloka X v trenutnem okvirju in naj d označuje velikost iskanja.

   (b) Najprej izračunamo korak $s = d/2$

   (c) Primerjamo blok X z bloki na položajih (x_i, y_i), $(x_i, y_i + s)$, $(x_i, y_i - s)$, $(x_i + s, y_i)$ in $(x_i - s, y_i)$, kjer i označuje iteracijo tega algoritma.

   (d) Izberemo blok z najboljšim ujemanjem in položaj zapišemo kot (x_{i+1}, y_{i+1}).

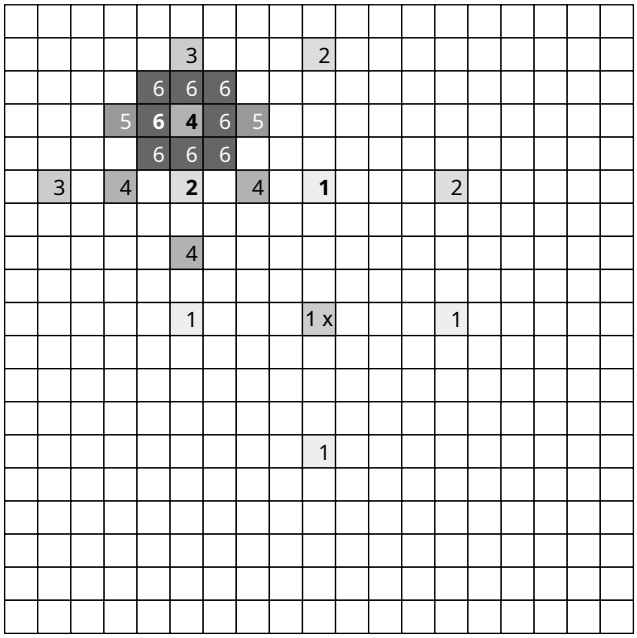
   (e) Če je $(x_{i+1}, y_{i+1}) = (x_i, y_i)$, s razpolovimo, sicer pa s ne spreminjamo.

   (f) Če je $d > 1$, postopek ponovimo s tretjim korakom. V naslednji iteraciji iskanje premaknemo na položaj (x_{i+1}, y_{i+1}).

   (g) Na koncu, ko je $s = 1$, preiščemo še vseh devet blokov (tistega na položaju (x_{i+1}, y_{i+1}) in osem neposrednih sosedov), izberemo najmanjšega in zaključimo iskanje.

   Primer iskanja, ko je $d = 8$, vidimo na sliki 5.7a. Številka v bloku označuje zaporedno številko preiskovanja, torej i.

5. V knjigi [7] omenjajo še druge metode, med katerimi je večina zelo podobnih prej omenjeni metodi, na primer **trikoračni algoritem** (angl. three-step search), ki se izvede v treh korakih, kar je pomembno pri sprotnem stiskanju, **ortogonalni algoritem** (angl. orthogonal search), ki izmenično išče v vodoravni in navpični smeri, ter **križno iskanje** (angl. cross search), ki išče v diagonalnih smereh. Iskanje blokov je še vedno zanimiv raziskovalni izziv, saj so se v zadnjih dveh desetletjih pojavile še druge metode [299, 300].



Slika 5.7: Dvodimenzionalno logaritmično iskanje

Vektorji gibanja so osnova večine postopkov stiskanja. Na sliki 5.8 vidimo primer vektorjev gibanja okvirja P, ki smo ga dobili iz orodja ffmpeg. Iz vektorjev gibanja na sliki lahko takoj ugotovimo, da sovpadajo z gibanjem objektov na sceni.

5.4 Shranjevanje videa

Video lahko prenašamo po omrežju ali pa ga shranimo v datoteke. V vsakem primeru uporabljamo podobne koncepte kot pri avdiu. Video praviloma stiskamo s **kodeki** in ga v **vsebovalnik** zapišemo v obliki zaporedja okvirjev ali **paketov** (angl. packet). Vsak paket vsebuje dovolj podatkov za uspešno dekodiranje; hrani torej skupino sličic GOP. Ker imamo poleg zaporedja slik še druge multimedijske podatke, na primer avdio, lahko okvirje/paketke zapišemo s prepletanjem. Tako lahko v zelo kratkem času dobimo vse pripadajoče multimedijske tokove. Način kodiranja videa (torej kodek) ni določen z izbiro vsebnika. Vsebovalniki vsebujejo pakete, ki jih kodek dekodira. Tako vsebovalnik kot način stiskanja pa sta odvisna od zahtev uporabe (na primer interaktivne aplikacije, pretočno kodiranje, arhiviranje). Najpogostejši datotečni formati za video so: AVI, ASF, QT, MP3, MPG,

(a) (b)

Slika 5.8: Okvir (a) I in (b) okvir P z vektorji gibanja iz animiranega filma Coffe Run (licenca: CC-BY-4.0 (CC), Blender Foundation, studio.blender.org)

MKV, MJ2, MXF, FLV, WEBM, MP4, WMV.

5.4.1 AVI

AVI (angl. Audio Video Interleave) je zelo znan datotečni format za video. Specifikacijo najdemo v [301], opise pa v člankih, na primer [302]. Datotečni format AVI temelji na strukturi RIFF, ki smo jo že spoznali. Datoteka AVI se torej začne z zlogi ASCII »RIFF«, nato sledi velikost datoteke in oznaka segmenta »AVI␣«. Segment tipa »AVI␣« vsebuje vsaj segmenta »hdrl« in »movi«.

Segment »hdrl« najprej vsebuje segment »avih« (angl. AVI Header), ki opisuje format kodiranja videa. Najpomembnejši podatki, ki jih segment vsebuje so:

- število podatkovnih tokov (tipična datoteka ima dva tokova, enega za video in enega za avdio),
- število vseh sličic,
- fps,
- največje število zlogov na sekundo,
- dimenzija videa,

- ali se uporablja indeks, ki omogoča naključen dostop do okvirjev videa,
- ali je video prepleten,
- ali je video avtorsko zaščiten,
- ali je predvajanje nujno z uporabo indeksa, torej če okvirji niso shranjeni kronološko.

Nato znotraj segmenta »hdrl« sledijo metapodatki za vsakega izmed multimedijskih **tokov** (angl. stream) v obliki seznama z oznako »strh« (angl. stream header). Vsak opis toka podatkov vsebuje:

- tip toka (na primer avdio, tekst, video),
- predlagan kodek za dekodiranje (na primer MJPG, DVSD, DVHD),
- ali se sme tok privzeto predvajati,
- prioriteta toka (kateri tok je treba predvajati z višjo prioriteto),
- časovni zamik toka, na primer če avdio in video nista sinhronizirana,
- dolžina toka,
- kakovost toka (močneje stiskamo, slabša je kakovost, kar je uporabno na šibkejših napravah – prožnost),
- področje znotraj navideznega platna, kamor rišemo video (podobno, kot pri datotečnem formatu GIF).

Za segmentom »strh« sledi opis formata toka podatkov v obliki segmenta »strf« (angl. stream format), katerega predstavitev je odvisna od tipa podatkov. V primeru videa opišemo kodiranje sličic, na primer dimenzija, bitna globina, način prostorskega stiskanja (PNG, JPG). Pri avdiu pa uporabimo opis, kot smo ga spoznali pri datotekah WAV (na primer način stiskanja in število kanalov). Na koncu lahko segment »strl« vsebuje še ime toka (»strn«) in druge dodatne podatke, ki so odvisni od samega tipa toka (»strd«).

Za segmentom »hdrl« znotraj segmenta »AVI␣« sledi segment »movi«, ki vsebuje dejanske kodirane podatke v obliki skupkov (angl. chunk). Vsak skupek je zapisan tako, da se začne s štirimi zlogi, kjer prva dva zloga določata številko multimedijskega toka, druga dva pa določata tip multimedijskega podatka (na primer nestisnjen okvir, stisnjen okvir, avdiopaket vzorcev). Pri kodiranju brez časovnega stiskanja skupki vsebujejo slike (okvirje I), v primeru uporabe naprednejših postopkov stiskanja (kodekov) pa skupki

vsebujejo podatke (pakete) neposredno za kodek, iz katerih tvori zaporedje slik.

Format AVI lahko znotraj segmenta »movi« vsebuje tudi zaporedje segmentov »rec␣«, kjer vsak vsebuje določeno število skupkov. Segment »rec␣« se uporablja pri shranjevanju videa na optične nosilce, kjer je nujno skupke zapisati na prepleten način, saj je takrat preskakovanje po datoteki zelo potratno. AVI podpira tudi izbirni segment »idx1«, ki, kot že ime pove, vsebuje sinhronizacijske kode, podane s kazalci na skupke v datoteki. Več kot je kazalcev, hitrejše preskakovanje je možno. AVI pa seveda omogoča tudi druge segmente, na primer »INFO«, ki smo ga spoznali pri datotekah WAV.

Tudi AVI je, podobno kot WAV, omejen z velikostjo datotek. Zaradi tega je izšla razširitev z imenom OpenDML AVI ali AVI 2.0.

5.5 Formati za kodiranje videa

Razvoj kodirnikov za stiskanje digitalnega videa je še vedno izjemno silovit. V začetku so bili gonilci razvoja telekomunikacijska podjetja in podjetja iz sveta zabavne elektronike. Ta so tudi izdelala prve standarde v okviru svojih standardizacijskih teles, na primer ITU in IEC (angl. International Electrotechnical Committee). Ti standardi so tudi najbolj znani, najbolj dokumentirani in so bili osnova mnogih raziskav. Implementacije teh standardov so bile praviloma patentno zaščitene, kar je po eni strani zagotavljalo konkurenčno prednost podjetjem na trgu, po drugi strani pa je bil prodor na trg z nestandardnimi rešitvami praktično nemogoč. Na ta način se je razvoj tudi dejansko upočasnjeval. Ključen preboj pri razvoju postopkov za stiskanje digitalnega videa se je zgodil z razvojem interneta in premikom prenosa digitalnega videa na ta medij. Internet je s svojim delovanjem omogočal nastanek rešitev, ki jih niso razvili proizvajalci zabavne elektronike ali telekomunikacijska podjetja. Računalniška podjetja so s svojimi rešitvami učinkovito konkurirala na novo nastalem trgu. V zadnjem času so se vodilna računalniška podjetja odločila, da bodo svojo priložnost iskala v storitvah in ne v kodirnikih, zato so novonastale rešitve odprtokodne in proste patentov, po kakovosti pa že močno presegajo uveljavljeno pot prejšnjega časa – standardizacijo.

V nadaljevanju bomo kronološko našteli najpomembnejše rešitve za stiskanje digitalnega videa, ki jih povzemamo po [303, 304, 7, 11, 233, 305] in posameznih specifikacijah. Nekatere pomembnejše pregledamo podrobneje.

5.5.1 H.120

H.120 je prvi standard [306] za kodiranje digitalnega videa, ki ga je CCITT (franc: Comité Consultatif International Téléphonique et Tél'egraphique)[1] sprva izdal leta 1984, nato pa je bil dopolnjen leta 1988. Kot vhod je bil predviden s PCM vzorčen analogni video NTSC. Ciljna uporaba je bila v videokonferencah. Ne glede na častitljivo starost so že takrat predlagali preprosto varianto napovednega kodiranja znotraj okvirjev (intra-frame), ki se pogosto omenja kot diferenčna pulzno-kodna modulacija (DPCM) in smo jo spoznali pri digitalnem zvoku. V standardu iz leta 1988 je opisana tudi kompenzacija gibanja (inter-frame) ter kodiranje napak in vektorjev gibanja s **kodami s spremenljivo dolžino** (angl. variable length coding – VLC). Za avdio so predpisali kodiranje po specifikaciji G.722 [261].

Standard je danes zastarel in ni več v uporabi. Standard ni bil uspešen niti ob izdaji, saj stopnja stiskanja ni bila dovolj visoka.

5.5.2 M-JPEG

M-JPEG (tudi MJPEG – angl. Motion JPEG) izhaja neposredno iz standarda JPEG, ki pa, v nasprotju z JPEG, ni standardiziran, prav tako ni predpisan zajem zvoka. M-JPEG deluje tako, da vsak okvir stisne neodvisno od drugih okvirjev s postopkom JPEG. Na ta način dosega razmerje stiskanja reda $1 : 20$. Ker ne izkorišča časovne/medokvirne redundance in ne vključuje kompenzacije gibanja, je občutno manj učinkovit od drugih postopkov, ga je pa veliko lažje urejati in predvajati vzvratno kot formate kodiranja, ki odstranjujejo tudi časovno redundanco. Zaradi obstoječe podpore za JPEG se je pogosto uporabljal pri digitalnih kamerah in orodjih za urejanje videa. Možno ga je shraniti tudi v vsebnike, na primer AVI, ali prenašati po omrežju (protokol za prenos avdia in videa RTP – angl. Real-time Transport Protocol) [307]. Podobno deluje novejši Motion JPEG2000, ki za stiskanje slik uporablja JPEG 2000 in je tudi standardiziran [308].

5.5.3 H.261

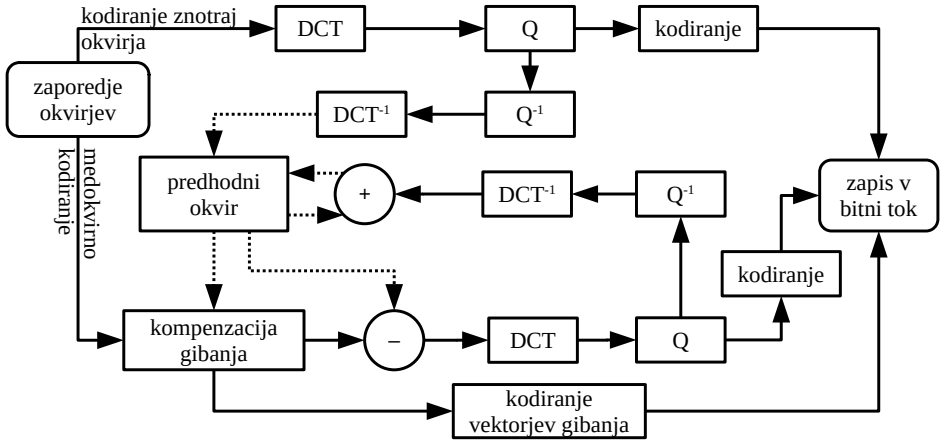
Pomemben preboj pri kodiranju videa je prinesel standard ITU **H.261** v letih 1988 [309] in 1990 [9]. Namenjen je bil izključno digitalnemu videu, in sicer v okolju telefonije ISDN (angl. Integrated Services Digital Network), ki je delovala s hitrostjo prenosa $p \times 64$ kb/s, $0 < p \leq 32$. Omogočal je prenos videa CIF (angl. Common intermediate format, Common Interchange Format) z ločljivostjo okvirja 352×288, s hitrostjo 29,97 fps in barvami,

[1] CCITT se je pozneje preimenoval v ITU (angl. International Telegraph Union)

zakodiranimi z YCbCr. Ločljivost je blizu slike v sistemu PAL, hitrost osveževanja slike pa je enaka kot pri NTSC. Takšen izbor je bil naraven kompromis za videokonference med državami z različnimi TV-standardi. H.261 je omogočal tudi ločljivost QCIF (angl. Quarter CIF) s polovično ločljivostjo po obeh oseh glede na CIF, ni pa podpiral prepletanja videa in je bil brez zvoka; predpostavljal je namreč, da bo zvok prenešen z uporabo ločenega kanala.

H.261 uporablja podvzorčenje barvnih komponent 4:2:0, velikost bloka 16×16 pikslov za kompenzacijo gibanja, DCT nad bloki velikosti 8×8 pikslov, kvantizacijo, cik-cak zaporedje in Huffmanovo kodiranje, torej koncepte, ki so večinoma v uporabi še danes. Kodirnik H.261 omogoča kodiranje okvirjev I in P, ki so kodirani relativno glede na predhodni okvir. Kodiranje poteka na naslednji način (slika 5.9):

- Kodirnik začne s kodiranjem okvirja I tako, da najprej izvede DCT in kvantizacijo (Q).
- Rezultat kvantizacije pošlje v kodirnik entropije z vnaprej podanimi kodirnimi tabelami in nato zapiše zakodirane podatke v izhodni tok.
- Kodirnik zakodiran okvir dekodira na enak način, kot bi ga dekodirnik, in sliko zapiše v medpomnilnik ter ga uporabi pri kodiranju okvirja P.



Slika 5.9: Diagram kodiranja videa H.261 s kodiranjem med okvirji in znotraj njih (prirejeno po [9] in [10]), pri čemer Q označuje kvantizacijo, polne črte zaporedje korakov, kar vključuje tudi tok podatkov, pikčaste črte pa zgolj tok podatkov.

H.261 izvede kodiranje okvirjev P na naslednji način:

- S kompenzacijo gibanja na nivoju blokov izračuna razliko glede na predhodni okvir.
- Vektorje gibanja pošlje v kodirnik entropije in nato v izhodni tok podatkov.
- Razlike med trenutnim in predhodnim okvirjem transformira z DCT in nato kvantizira.
- Transformirane razlike pošlje v kodirnik entropije in nato v izhodni tok podatkov.
- Stisnjen okvir rekonstruira na enak način, kot bi ga dekodirnik (inverzna kvantizacija, inverzna DCT, prištevanje k referenčnemu predhodnemu okvirju na sliki 5.9) in ga nato shrani v medpomnilnik za uporabo pri naslednjem okvirju P.

H.261 opisuje tudi bitni tok podatkov, ki je organiziran hierarhično na naslednji način. Tok podatkov sestavlja zaporedje okvirjev, pri čemer se okvir začne z 20 sinhronizacijskimi biti, nato sledi časovna umestitev okvirja, ločljivost okvirja (CIF ali QCIF), dodatne informacije in nato zaporedje **skupin blokov** (angl. group of blocks – **GOB**). Število skupin blokov je 12 pri CIF (6 vrstic in 2 stolpca znotraj okvirja) ali tri pri QCIF (tri vrstice znotraj okvirja). V GOB je zapisanih 16 sinhronizacijskih bitov, nato sledijo še nekateri drugi parametri (na primer parametri kvantizacije), nato pa zaporedje 33 blokov (postavitev blokov znotraj GOB je 11 stolpcev in 3 vrstice). Znotraj vsakega bloka je zapisan način kodiranja (intra/inter, torej kodiranje med okvirji ali znotraj okvirjev), vektor gibanja in 6 blokov (štirje za kanal svetlosti in dva za barvitost zaradi podvzorčenja 4:2:0). V vsakem bloku so zapisani koeficienti DCT velikosti 8×8 pikslov.

Opisana predstavitev določa način kodiranja med okvirji ali znotraj njih na nivoju blokov oziroma makroblokov. To pomeni, da lahko določene okvirje P zakodiramo na takšen način, da nekaj makroblokov kodiramo kot pri okvirjih I, nekaj pa kot pri okvirjih P. Izbira pa je seveda odvisna od kodirnika.

5.5.4 DV

Standard **DV** (angl. Digital Video) je razvil IEC [310]. Namenjen je bil videokameram, ki so shranjevale digitalni video na magnetni trak. DV vsak okvir kodira neodvisno z DCT [311], torej ne vključuje kompenzacije gibanja, in tudi zvoka ne stiska. DV kodira s konstantno pasovno širino 25 Mb/s ter omogoča bloke DCT velikosti 8×8 in 4×8 za prepleten video [311].

Pri slednjem izvede DCT ločeno nad sodimi in lihimi polji. Torej, pri prepletenem videu nad vsakim blokom 8×8 izvede ločeno DCT 4×8 nad lihimi in potem še nad sodimi vrsticami. Opisana rešitev je pomembna predvsem pri hitrem gibanju. Sony in Panasonic sta standard DV prilagodila in patentno zaščitila (DVCAM, DVCPRO).

5.5.5 Družina videostandardov MPEG

MPEG-1 je prvi standard [312] iz družine standardov **MPEG** za kodiranje videa in avdia (objavljen je bil leta 1993). Pri projektu MPEG (angl. Moving Pictures Experts Group) je sodelovalo več kot 100 strokovnjakov pod okriljem ISO in IEC [7]. Kako potreben je bil ta standard, kaže dejstvo, da so ga začeli uporabljati različni proizvajalci zabavne elektronike, še preden je bil formalno sprejet kot mednarodni standard. MPEG-1 so sledili še drugi standardi iz družine MPEG. Nadgradnja MPEG-1 je standard MPEG-2. MPEG-3 je bil načrtovan za HDTV, a se je kmalu izkazal za nepotrebnega in se je zato zlil z MPEG-2. Sledili so jim MPEG-4, MPEG-H in MPEG-I.

Standardi družine MPEG so, podobno kot preostali standardi ISO/IEC, sestavljeni iz **normativnega** in **informativnega** dela. Normativni del vključuje dejanske specifikacije standarda in je namenjen razvijalcem, informativni del pa opisuje koncepte in podaja razloge za posamezne izbire. Del normativnega dela so na primer kodirne tabele, del informativnega dela pa algoritmi za kompenzacijo gibanja blokov in mere ujemanja blokov. MPEG ne predpisuje nobenega algoritma in razvijalci lahko implementirajo povsem svoje metode.

MPEG uporablja tudi svoj besedni zaklad. **Videosekvenca** (angl. video sequence) je sestavljena iz **slik** (angl. pictures) oziroma **okvirjev** (angl. frame), izraz, ki se uporablja tudi pri MPEG-1 Avdio (v nadaljevanju bomo enakovredno uporabljali izraza okvir in slika). Okvir je sestavljen iz treh komponent: **svetlosti**/**luminance** Y (angl. luminance) in **barvitosti**/**krominance** Cb, Cr (angl. chrominance). Vsaka komponenta je predstavljena z 2D-poljem **vzorcev** (angl. samples). **Pel** (angl. pel) je razumljen kot vrednosti treh barvnih komponent Y, Cb in Cr.

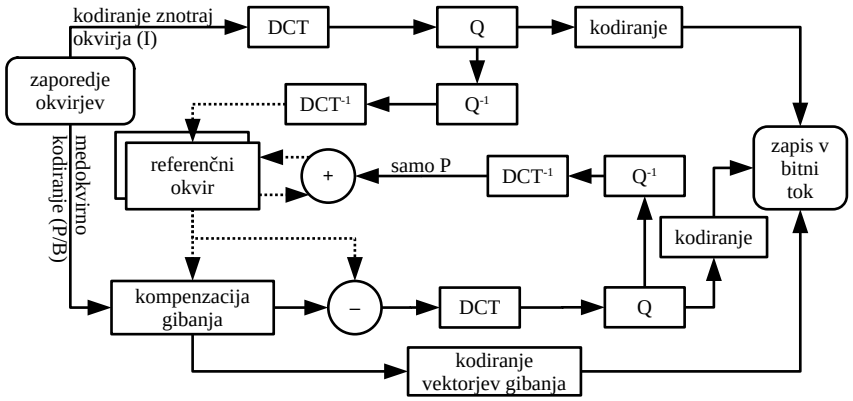
5.5.6 MPEG-1

MPEG-1 definira kodirni sistem za sinhronizirane avdio- in videopodatke, ki ga je industrija zabavne elektronike takoj široko sprejela in podprla, tako da ga še danes najdemo v mnogih napravah in aplikacijah. Namenjen je digitalnemu videu brez prepletanja v ločljivosti 352×288 (omogoča tudi druge ločljivosti vse do 4k [11]) s 24/30 fps (angl. frames per seconds). Za

video s takšno ločljivostjo pri 24 fps in barvno globino 24 bpp potrebujemo 59.719.680 b/s. Video spremljata še dva zvočna kanala, ki ju vzorčimo s 44,1 kHz in kvantiziramo s 16 biti, kar nam dodatno da 1.411.200 b/s, kot smo spoznali v prejšnjem poglavju. Skupaj dobimo 61,1 Mb/s. MPEG mora to količino podatkov stisniti za predvideno hitrost prenosa 1,5 Mb/s, to je toliko, kot zahtevata samo oba zvočna kanala. MPEG-1 za prenos videopodatkov namenja približno 1,2 Mb/s in okoli 250 kb/s za zvok. Da bi to dosegli, mora biti razmerje stiskanja preko 1 : 50 za video in dobrih 1 : 5,6 za avdio. Drug pomemben faktor je hitrost dekodiranja, saj moramo tako video- kot avdiopodatke dekodirati in predvajati v realnem času. Del standarda MPEG-1 definira, kako avdio- in videopodatkovna tokova sestavimo v multipleksirane podatke. V nadaljevanju tega podpoglavja se z zvokom ne bomo ukvarjali; osredotočili se bomo izključno na video, kot je opisano v standardu [312].

5.5.6.1 Kodiranje videa MPEG-1

Kodiranje videa MPEG-1 nadgrajuje standard H.261 z okvirji B, zaradi tega je kodirnik nekoliko zahtevnejši. Postopek stiskanja povzemamo na primeru kodiranja okvirjev I, P in B po [313, 11, 7, 294], poenostavljen blokovni diagram videokodirnika MPEG-1 pa vidimo na sliki 5.10.



Slika 5.10: Diagram kodiranja videa MPEG-1 na primeru kodiranja okvirjev I, P in B (prirejeno po [10]), pri čemer Q označuje kvantizacijo, polne črte zaporedje korakov, kar vključuje tudi tok podatkov, pikčaste črte pa zgolj tok podatkov.

Vhodne okvirje najprej razporedimo glede na tip okvirja (slika 5.5), te pa nato razdelimo v makrobloke velikosti 16×16 pikslov. Vsak makroblok razdelimo na 4 bloke 8×8 za svetlost in na dva bloka 8×8 za barvitost

(zaradi podvzorčenja barv). Okvirje I zakodiramo z JPEG. Vsak blok transformiramo z DCT, kvantiziramo, koeficiente uredimo v zaporedje cik-cak ter zakodiramo z RLE in s Huffmanovimi kodami, ki so predpisane vnaprej. Zakodiran okvir dekodiramo tako, da izvedemo inverzno kvantizacijo in inverzno DCT ter ga shranimo v vmesni medpomnilnik kot referenčni okvir (za poznejše kodiranje okvirjev P in B).

Okvirje P določimo na podlagi predhodnega okvirja I ali P, pri čemer pa ni nujno, da so vsi makrobloki kodirani medokvirno. Nekateri so lahko predstavljeni samostojno kot makrobloki v okvirjih I. Če ima okvir vsaj en makroblok, ki je določen glede na predhodne slike, potem okvir obravnavamo kot okvir P. Tip kodiranja zapišemo v glavi vsakega makrobloka. Pri makroblokih P najprej opravimo kompenzacijo gibanja na podlagi predhodnega referenčnega okvirja in za vsak makroblok določimo vektor premika. Pomembna nadgradnja MPEG-1 v primerjavi s H.261 je natančnejša predstavitev vektorjev gibanja na **polovico piksla** (angl. half-pel). Izračunane vektorje premika uporabimo pri izračunu razlik glede na referenčno sliko. Nato nad makrobloki razlik opravimo podoben postopek kot pri kodiranju okvirja I (DCT nad razlikami, kvantizacija, kodiranje in zapis v izhodni bitni tok).

Ob zapisu v izhodni bitni tok kodirnik preveri, ali lahko količino podatkov še dodatno zmanjša. Če so vsi koeficienti bloka 0, pošljemo na izhod samo kodo EOB (angl. end of block). V najboljšem primeru je cel makroblok enak makrobloku referenčne slike in v tem primeru makrobloka ne kodiramo. Če makrobloka nista identična, potem zelo pogosto postaneta identična z uporabo vektorja premika. V tem primeru takšen makroblok zakodiramo samo z vektorjem premika. Količina podatkov tako raste z razliko med trenutnim in referenčnim okvirjem. Če kodirnik oceni, da postane kodiranje okvirjev P neučinkovito, okvir zakodira kot okvir I.

Tudi pri okvirjih P izvedemo dekodiranje za poznejše kodiranje okvirjev P in B. Vsak blok v makrobloku dekodiramo z inverzno kvantizacijo in inverzno DCT. Dobljene vrednosti pikslov medokvirno kodiranih makroblokov prištejemo k vrednostim referenčnega makrobloka z upoštevanjem vektorjev premika. Nato sliko shranimo kot referenčni okvir za kodiranje okvirjev P in B.

Kodiranje okvirjev B je nekoliko zahtevnejše, saj za njih potrebujemo predhodno in naslednjo sliko. Zaradi tega mora kodirnik hraniti dva referenčna okvirja (slika 5.5). Podobno kot pri okvirjih P so lahko tudi v okvirjih B nekateri makrobloki kodirani neodvisno kot v I-okvirjih. Če je vsaj en makroblok kodiran z uporabo predhodnika in naslednika, potem je celoten okvir obravnavan kot okvir B. Pri kodiranju makroblokov okvirja B

imamo naslednje možnosti:

- Makroblok kodiramo brez napovedi, zato v njega ne zapišemo vektorjev premika (makroblok obravnavamo kot makroblok I).
- Makroblok kodiramo relativno glede na predhodni referenčni okvir (torej kot makroblok okvirja P), zato v makroblok shranimo **vektor gibanja naprej** (angl. forward-motion vector).
- Makroblok kodiramo relativno glede na naslednji referenčni okvir (torej od bodoče slike, ki smo jo že predhodno zakodirali), zato v makroblok shranimo **vektor gibanja nazaj** (angl. backward-motion vector).
- Makroblok kodiramo relativno glede na predhodni in naslednji referenčni okvir s povprečenjem vrednosti iz obeh okvirjev. Takrat v makroblok shranimo oba vektorja premika.

Pri okvirjih B v MPEG-1 imamo še eno posebnost, in sicer da nikoli ne postanejo referenčni okvirji. Ta omejitev je v novejših standardih odpravljena. MPEG-1 ima tudi **okvirje D**, ki so namenjeni hitremu predogledu. Vsebujejo zgolj komponente DC transformacije DCT (povprečna barva bloka). Okvirji D pa se v novejših kodirnikih ne uporabljajo več, saj so redundantni glede na okvirje I.

Seveda je dejanska implementacija MPEG-1 zahtevnejša, kot smo idejno povzeli. Tipičen primer je na primer kvantizacija koeficientov DCT, ki jo nad makrobloki okvirjev I opravimo drugače kot nad makrobloki okvirjev P in B. Levi zgornji koeficienti v blokih DCT makroblokov okvirjev I predstavljajo nižje harmonske frekvence, zato jih kvantiziramo šibkeje kot višje harmonske frekvence. Makrobloki okvirjev P in B predstavljajo razlike glede na referenčne okvirje, s čimer smo že odstranili korelacijo in odpravili povezavo s psihofizičnimi lastnostmi našega vida. Zaradi tega vse frekvenčne koeficiente kvantiziramo enako. Z enačbo 5.4 kvantiziramo bloke DCT makroblokov okvirjev I, bloke makroblokov okvirjev P in B pa z enačbo 5.5 [7].

$$QDCT[i,j] = round\left(\frac{8DCT[i]}{sQ_I[i,j]}\right) \tag{5.4}$$

$$QDCT[i,j] = round\left(\frac{8DCT[i]}{sQ_{PB}[i,j]}\right) \tag{5.5}$$

Kvantizacija je odvisna od parametra s, ki ga kodirnik med kodiranjem spreminja in prilagaja sceni ter predvsem zahtevani hitrosti prenosa. Standardni privzeti kvantizacijski tabeli Q_I in Q_{PB} vidimo v tabeli 5.1.

Tabela 5.1: Kvantizacijski tabeli za bloke okvirjev I (levo) ter P in B (desno) [7]

8	16	19	22	26	27	29	34	16	16	16	16	16	16	16	16
16	16	22	24	27	29	34	37	16	16	16	16	16	16	16	16
19	22	26	27	29	34	34	38	16	16	16	16	16	16	16	16
22	22	26	27	29	34	37	40	16	16	16	16	16	16	16	16
22	26	27	29	32	35	40	48	16	16	16	16	16	16	16	16
26	27	29	32	35	40	48	58	16	16	16	16	16	16	16	16
26	27	29	34	38	46	56	69	16	16	16	16	16	16	16	16
27	29	35	38	46	56	69	83	16	16	16	16	16	16	16	16

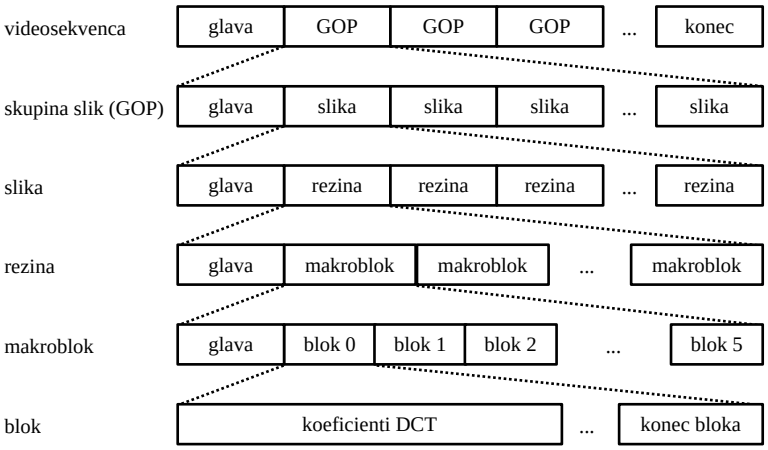
5.5.6.2 Bitni tok podatkov

Bitni tok MPEG-1 je sestavljen iz treh glavnih delov, ki jih imenujemo **plasti** (angl. layers): za predstavitev videa, avdia in sistemskih podatkov. V sistemski plasti so shranjeni paketi podatkov za avdioplast (okvirji iz MPEG-Audio) in videoplast (**videosekvenca**) [11]. Tok podatkov za video in avdio je torej multipleksiran v okviru paketov. Opis sistemske plasti je v prvem delu standarda, torej v ISO/IEC 11172-1 [314].

Na sliki 5.11 vidimo vse podatkovne plasti videokodirnika MPEG-1. Okvirji I, P in B so urejeni v skupine **GOP** znotraj videosekvence. Vsak GOP je sestavljen iz ene slike I ter več slik P in B (število slik B je lahko tudi 0). GOP je osnovna enota, pri kateri okvir I uporabljamo kot vstopno točko, s čimer do neke mere omogočamo naključni dostop. Osnovni gradnik slike je makroblok, slike pa razdelimo v **rezine** makroblokov (angl. slice). Rezina je zaporedje makroblokov v prebirnem vrstnem redu, ki imajo približno enako svetlost (komponento Y). Število makroblokov v rezini ni določeno vnaprej, ampak jo določi kodirnik glede na pričakovano pogostost napak pri prenosu. Makroblok je sestavljen iz 16×16 vzorcev komponente Y (ki je nadalje razdeljen v 4 bloke po 8×8) in dva bloka komponent barvitosti, ki sta velikosti 8×8. Vrsto makrobloka zapišemo v glavi makrobloka. Če gre za makroblok, ki je zakodiran na podlagi napovedi, glava hrani tudi en ali dva vektorja premika (okvirji B). Nato sledijo stisnjeni podatki o koeficientih za vseh 6 blokov. Za okvirje I zapišemo vse makrobloke, pri okvirjih B in P pa lahko nekatere preskočimo.

5.5.7 MPEG-2 ali H.262

MPEG-2 ali **H.222/H.262** je neposredni naslednik MPEG-1, ki sta ga skupaj izdelala ISO in ITU ter izdala leta 1996. Standard H.262 [13] oziroma



Slika 5.11: Podatkovne plasti videa pri MPEG-1 po vzoru iz [11]

ISO/IEC 13818-2 [315] določa kodiranja videa, ISO/IEC 13818-3 [259] prinaša izboljšano kodiranje zvoka MPEG-Audio, ISO/IEC 13818-7 [256] pa nov način kodiranja zvoka AAC (angl. Advanced Audio Coding). Tako kot MPEG-1 tudi MPEG-2 ne standardizira samega procesa kodiranja, ampak samo sintakso izhodnega binarnega niza. Standard H.222 [292], znan tudi kot ISO/IEC 13818-1 [316], določa format bitnega toka oziroma sistemsko plast. Ta standard opisuje tudi **transportni tok** (angl. transport stream), ki omogoča hkraten prenos in multipleksiranje več podatkovnih tokov videa in zvoka, vključno s televizijskimi programi za digitalno televizijo (kot so DVB, ATSC in IPTV).

MPEG-1 je namenjen napredujočemu kodiranju digitalnega videa s konstantno hitrostjo prenosa 1,5 Mb/s, kot so v tistih časih omogočali bralniki CD-ROM-ov. Cilj MPEG-2 pa je bil podpreti višjo kakovost tako digitalnega videa kot digitalnega zvoka za digitalno televizijo. MPEG-2 je bil zato načrtovan za višje hitrosti prenosa podatkov, in sicer od 4 do 15 Mb/s [187], kar omogoča tudi višje ločljivosti (od SD, Full HD ter celo Ultra HD) [11]. Za digitalno televizijo MPEG-2 predvideva stiskanje videa, ki je praktično enako kot pri MPEG-1, a prilagojeno prepletanju za predstavitev digitaliziranih vsebin iz analognih standardov PAL/NTSC/SECAM in podpori za prožnost. Vse naprave MPEG-2 lahko predvajajo tudi MPEG-1. Kodiranje avdia je tudi razširjeno glede na MPEG-1, ki je omogočal samo dva kanala, MPEG-2 pa ponuja tudi prostorski zvok s šestimi kanali (angl. 5.1 surround sound). Zaradi vsega tega je MPEG-2 najpogostejši format digitalne televizije.

5.5.7.1 Prožnost

Bitni niz imenujemo **prožen**, če lahko del bitnega niza odstranimo tako, da dobimo podniz, ki ga ciljni dekodirnik še vedno razpozna kot veljavnega in ga je zmožen dekodirati. Seveda je dobljena kakovost takšnega videa nižja od kakovosti, pridobljene iz izvornega niza. Takšnemu bitnemu nizu pravimo tudi **večnivojski** (angl. multi-layered), bitni niz, ki te značilnosti nima, pa je **enonivojski** (angl. single-layered). Prožen video se lahko na ta način prilagaja razpoložljivi hitrosti prenosa, računalniški moči na strani dekodirnika in ločljivosti prikazovalnika.

MPEG-2 uvaja koncept **prožnega kodiranja** (angl. scalable coding) oziroma **nivojskega kodiranja** (angl. layered coding), pri katerem najprej zakodiramo video v nižji kakovosti, ki mu nato sledi informacija v novem nivoju, kako kakovost slike izboljšati. Osnovni nivo ima višjo prioriteto prenosa. Če so nastale napake ali je informacijski kanal zelo obremenjen, dodatnih informacij ne pošljemo. MPEG-2 podpira naslednje načine prožnega kodiranja [11]:

- **Prožnost SNR** (angl. signal-to-noise ratio scalability) zakodira video najprej v nižji kakovosti, torej izvede močnejšo kvantizacijo koeficientov DCT, s čimer zmanjša količino podatkov in poveča SNR (od tod ime SNR). Nato izračuna razlike med prvotnim in zakodiranim zaporedjem slik in jih zakodira z DCT, kjer uporabi nižjo stopnjo kvantizacije, torej zniža SNR. Takšen video pošlje v ločenem višjem nivoju.
- **Prostorska prožnost** (angl. spatial scalability) omogoča kodiranje videa najprej v nižji ločljivosti, naslednji nivo pa vsebuje informacije o sliki višje ločljivosti. MPEG-2 video višje ločljivosti zakodira z razlikami glede na interpolirano sliko v nižji ločljivosti. Na takšen način lahko v osnovnem nivoju pošljemo video v ločljivosti SD, v višjem pa video v ločljivosti HD.
- **Časovna prožnost** (angl. temporal scalability) v osnovni nivo zapiše video z nižjim fps, v višji nivo pa doda vmesne okvirje, ki so kodirani s kompenzacijo gibanja glede na osnovni nivo.
- **Hibridna prožnost** (angl. hybrid scalability) omogoča kombinacijo dveh od prvih treh prožnosti. Prožnost v MPEG-2 je možna v treh nivojih: osnovni nivo in dva dodatna višja nivoja.

5.5.7.2 Profili in nivoji

MPEG-2 je bil načrtovan za različne aplikacije. Zaradi tega so pri MPEG-2 uvedli koncept **profilov** (angl. profile), ki določajo nabore (podmnožico)

funkcionalnosti formata za določene aplikacije. MPEG-2 za kodiranje videa predpisuje naslednje profile:

1. **SP** (angl. Simple Profile) je namenjen za aplikacije z majhno zakasnitvijo (na primer za videokonference) in ne vključuje okvirjev B [11].

2. **MP** (angl. Main Profile) je najpogostejši profil, ki ga uporabljamo pri na primer DVD-ju, videu na zahtevo in digitalni televiziji, torej tam, kjer potrebujemo dobro kakovost neprožnega videa. Profil MP temelji na kodiranju slik I, P in B, ki jih poveže, tako kot MPEG-1, v GOP.

3. **SNR** (angl. SNR Scalable Profile) je namenjen za aplikacije, ki potrebujejo video enake ločljivosti, a v dveh kakovostih (prožnost SNR), in poteka v dveh nivojih.

4. **Spt** (angl. Spatially Scalable Profile) podpira tudi prostorsko prožnost, torej različne ločljivosti videa, in seveda hibridno prožnost.

5. **HP** (angl. High Profile) je nadgradnja profila Spt, ki omogoča tudi kakovostnejšo kodiranje barvitosti v ločljivosti 4:2:2.

MPEG-2 omogoča zelo širok nabor bitnih hitrosti in ločljivosti, kar zahteva zapletenejšo implementacijo kodekov. Zaradi tega določa znotraj vsakega profila še do štiri možne **nivoje** (angl. layer), ki omejujejo nabor parametrov kodiranja (na primer ločljivost in bitna hitrost), kot vidimo v tabeli 5.2. Parametri kodiranja so za posamezne profile in nivoje predpisani v standardu [13].

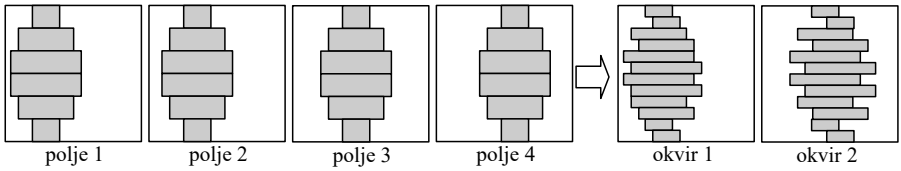
Tabela 5.2: Bitne hitrosti pri profilih MPEG-2

Nivo	ločljivost	fps	bitna hitrost (Mbps)				
			SP	MP	SNR	Spt	HP
High	1920 × 1152	60		80			100
High-1440	1440 × 1152	60		60		60	80
Main	720 × 576	30	15	15	15		20
Low	352 × 288	30		4	4		

V MPEG-2 so naknadno dodali tudi profil **MV** (angl. Multiview Profile), ki je namenjen stiskanju scene, posnete iz vsaj dveh kamer (stereopogled). Video iz leve kamere zakodiramo v osnovni nivo, video iz desne pa v dodaten **nivo izboljšanja** (angl. enhancement layer). Na ta način starejši predvajalniki brez težav predvajajo video iz le osnovnega nivoja.

5.5.7.3 Kodiranje prepletenega videa

MPEG-2 je namenjen predvsem digitalni televiziji, kjer je veliko videovsebin predstavljenih s prepletanjem. Okvir prepletenega videa je sestavljen iz dveh polj (angl. field), in sicer **lihega** oziroma **zgornjega polja** (angl. top field) in **sodega** oziroma **spodnjega polja** (angl. bottom field). Spodnje polje hrani stanje scene v trenutku za zgornjim poljem. Vsak okvir prepletenega videa je sestavljen iz zgornjega polja, ki ga prikažemo v lihih vrsticah, in spodnjega polja, prikazanega s sodimi vrsticami (slika 5.12). Pri hitrem gibanju zato lahko zaznamo motnje zaradi prikaza stanja v dveh trenutkih v enem okvirju. Če bi takšne okvirje kodirali z MPEG-1, bi videz slike zelo popačili zaradi močne kvantizacije koeficientov DCT višjih frekvenc, ki nastanejo prav zaradi hitrega gibanja pri prepletenem videu.



Slika 5.12: Zgornja in spodnja polja ter združena okvirja

Zato MPEG-2 nadgrajuje posamezne korake kodiranja videa iz MPEG-1 za podporo prepletenemu videu, ki ga lahko kodira na dva načina. Posamezna polja prepletenega videa lahko zapiše kot ločene **slike polj** (angl. field pictures) in potem nad posameznimi slikami uporabi enak postopek kodiranja kot pri MPEG-1. Drugi način je, da celotne okvirje prepletenega videa zakodira kot slike okvirjev (angl. frame pictures) [11]. Posamezni koraki kodiranja se na podlagi te odločitve ustrezno prilagodijo.
MPEG-2 omogoča pet različnih napovedi slik prepletenega videa [11, 233]:

1. **Napoved okvirjev.** Napoved je enaka kot pri MPEG-1; tvorimo makrobloke velikosti 16×16 ter za okvirje P in B opravimo kompenzacijo gibanja.

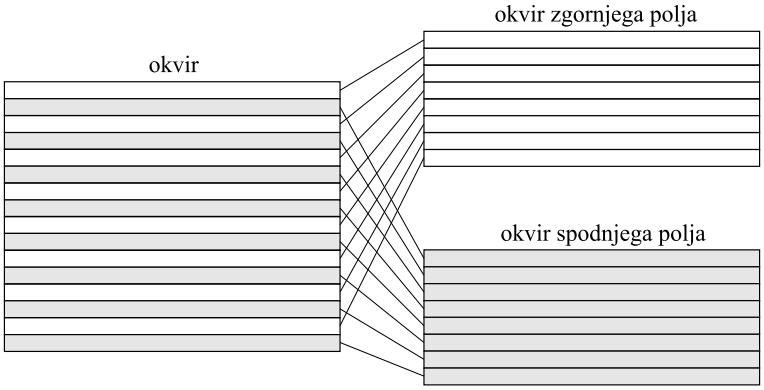
2. **Napoved polj iz slik polj.** Tudi v tem primeru uporabimo makroblok velikosti 16×16, ampak nad polji, ki so shranjena v slikah. To pomeni, da en makroblok 16×16 zajema 16×32 pikslov okvirja.

3. **Napoved polj iz slik okvirjev** je nadgradnja predhodnega načina. Zgornje in spodnje polje znotraj makrobloka napoveduje ločeno. Makroblok 16×16 iz ciljnega okvirja polja razdelimo na dva dela velikosti 16×8; vsak del prihaja iz ustreznega polja, kot kaže slika 5.13. Zatem

izvedemo napoved gibanja ločeno za prvi in drugi del. Pri tem načinu tako za makrobloke P hranimo dva vektorja gibanja, za makrobloke B pa štiri.

4. **Kompenzacija gibanja** 16×8 **nad slikami polj.** Vsak makroblok velikosti 16×16 iz slike polj razdelimo v zgodnji in spodnji del, nato pa izvedemo napoved gibanja za vsako polovico. Za vsak začetni makroblok 16×16 slike polj tudi tokrat dobimo dva vektorja gibanja za slike P in štiri za slike B.

5. **Dvojna napoved (angl. dual-prime) za slike P.** Najprej za vsako polje (zgornje ali spodnje) poiščemo vektor premika v predhodnem polju enake vrste. Ta vektor gibanja uporabimo za določitev izračunanega vektorja gibanja (angl. calculated motion vector) za napoved iz polja druge vrste (na primer iz zgornjega polja v spodnjega), nato pa na podlagi obeh vektorjev za vsak makroblok izvedemo dve napovedi in zakodiramo povprečno napako obeh napovedi.



Slika 5.13: Okvir iz zgornjega in spodnjega polja

Izbran način kodiranja vpliva tudi na kodiranje kvantiziranih koeficientov DCT. Prepletanje spremeni frekvence v navpični smeri, zato MPEG-2 takrat uporablja drugačni vrstni red kodiranja koeficientov po DCT in kvantizaciji kot MPEG-1. Primerjavo vidimo v tabeli 5.3.

5.5.8 H.263

ITU je leta 1996 izdal **H.263** [317]. Namenjen je okoljem z nizko hitrostjo prenosa, kot sta videotelefonija in videokonference. Zelo pogosto se je uporabljal tudi na spletu (na primer Adobe Flash, Youtube, Google Video).

Tabela 5.3: Vrstni red kodiranja koeficientov bloka DCT 8×8 za (levo) napredujoč in (desno) prepleten video pri MPEG-2 [13]

0	1	5	6	14	15	27	28	0	4	6	20	22	36	38	52
2	4	7	13	16	26	29	42	1	5	7	21	23	37	39	53
3	8	12	17	25	30	41	43	2	8	19	24	34	40	50	54
9	11	18	24	31	40	44	53	3	9	18	25	35	41	51	55
10	19	23	32	39	45	52	54	10	17	26	30	42	46	56	60
20	22	33	38	46	51	55	60	11	16	27	31	43	47	57	61
21	34	37	47	50	56	59	61	12	15	28	32	44	48	58	62
35	36	48	49	57	58	62	63	13	14	29	33	45	49	59	63

Podpira različne ločljivosti od 128×96 do 1408×1152. V osnovni varianti temelji na kodiranju **H.261** [9], saj ne uporablja okvirjev B. Kot novost pa uvaja napoved vektorjev gibanja glede na sosednje makrobloke [233]. Za napoved vsake izmed koordinat vektorja gibanja izbere srednjo vrednost iz levega, zgornjega in zgornjega desnega sosednjega makrobloka. Vektorje gibanja določa na pol piksla natančno.

H.263 je v osnovni varianti dokaj preprost, ima pa veliko nadgradenj in **dopolnitev** (angl. annex). Nekatere pomembnejše so [233]: daljši vektorji gibanja, uporaba aritmetičnega kodirnika in okvirji B. Pomembna izboljšava pa je naprednejši način napovednega kodiranja (angl. advanced prediction mode), ki vključuje dve novosti: prva je uporaba manjših makroblokov 8×8 za kompenzacijo gibanja, ki jim pravimo **bloki napovedi** (angl. prediction block), druga pa je način kodiranja s **prekrivanjem blokov pri kompenzaciji gibanja** (angl. overlapped block motion compensation – **OBMC**), s katero zmanjšamo vidnost robov med makrobloki pri nižji bitni hitrosti. Deluje tako, da piksle znotraj bloka napove na podlagi vektorja gibanja trenutnega bloka in dveh sosednjih blokov glede na oddaljenost od meje bloka. V letu 1998 je ITU izdal novo različico standarda, ki prinaša še več dopolnitev. Najpomembnejše so [233, 318]:

- Izboljšano kodiranje makroblokov okvirjev I, ki odpravlja redundanco med bloki z napovedovanjem koeficientov DCT iz sosednjih blokov.
- Večje območje vektorjev gibanja.
- Podpora za rezine namesto GOB.
- Prožnost (časovna, SNR, prostorska).
- Uporaba filtra za odpravo vidnih robov med bloki, ki je integriran znotraj **zanke kodirnika** (angl. coding loop). Zanka kodirnika je

prikazana na sliki 5.9 in predstavlja zaporedje korakov: kodiranje okvirja, dekodiranje in zapisa v referenčni okvir za druge okvirje P/B. Pri H.263 filtriramo dekodirano sliko tik pred zapisom v referenčni okvir in jo nato uporabimo za druge okvirje.

V letih 2000 in 2005 je ITU objavil drugo nadgradnjo standarda, ki prinaša še več dopolnitev [233]. H.263 je skupaj z nekaterimi nadgradnjami predstavljal osnovo za razvoj novejših formatov.

5.5.9 MPEG-4 in MPEG-4 part 2

Standard MPEG-4 je bil zasnovan zelo ambiciozno, saj je vključeval množico novih konceptov, kot so interaktivna grafika, kodiranje objektov, kodiranje vstopnih okvirjev z valčno transformacijo, modeliranje obraza in govora ter 3D-grafiko. A večino teh konceptov ni našlo komercialne uporabe, zato so se pozneje osredotočili predvsem na kodiranje digitalnega videa in avdia.

H.263 je predstavljal osnovo za razvoj prvega videoformata iz skupine standardov MPEG-4, ki je znan pod imenom MPEG-4 part 2 in je opisan v sklopu standarda ISO/IEC 14496-2 [319]. Predvajalniki MPEG-4 part 2 lahko predvajajo osnovni videoformat H.263. Seveda pa MPEG-4 part 2 prinaša kar nekaj nadgradenj [320, 11, 233]:

- Kompenzacija gibanja je možna z natančnostjo 1/4 piksla (angl. quarter-pixel – **qpel**).
- **Globalna kompenzacija gibanja** (angl. global motion compensation) izvaja kompenzacijo gibanja nad celotnim okvirjem, kar omogoča zahtevnejšo predstavitev gibanja (ne zgolj s premikanjem, ampak tudi z na primer vrtenjem in skaliranjem). S takšno rešitvijo lahko pri premiku kamere dosežemo krajše vektorje gibanja v posameznih makroblokih, kar omogoča učinkovitejše kodiranje.
- Fleksibilnejša prožnost z več nivoji prožnosti.
- **Kodiranje videa na nivoju objektov** (angl. object-based coding), kar omogoča ločeno kodiranje posameznih objektov v različnih **ravninah** (angl. video object plane – **VOP**). Kodiranje na nivoju objektov omogoča ločen prenos okvirjev oziroma ravnin ozadja in objektov na sceni z različnimi stopnjami kvantizacije. Kodiranje na nivoju objektov prinaša nove izzive, kot je na primer predstavitev obrobe objektov v kombinaciji z makrobloki.
- Ponovno je dodana podpora za okvirje B, ki se v okviru VOP imenujejo ravnine B (B-VOP).

MPEG-4 Part 2 prinaša veliko novih funkcionalnosti, zato uvaja tudi več profilov, podobno kot MPEG-2, da podpre tako šibke naprave kot vsebine z ločljivostjo 4k. Najbolj znana profila sta SP (angl. Simple profile), ki je osnovna verzija H.263, in ASP (angl. Advanced Simple Profile), ki omogoča prepleten video in izboljšuje učinkovitost kodiranja (na primer okvirji B, kompenzacija gibanja na četrtino piksla natančno). Večina aplikacij uporablja profil ASP, zato se velikokrat MPEG-4 Part 2 imenuje tudi MPEG-4 ASP.

5.5.10 MPEG-4 AVC/H.264

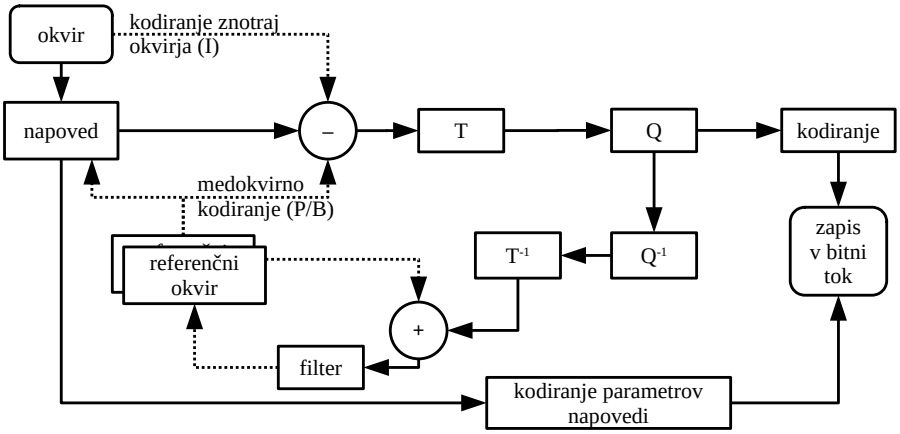
MPEG-4 part 10 oziroma **AVC** (angl. Advanced Video Coding) iz leta 2003 je drugi videoformat v okviru MPEG-4 [321]. Pri razvoju je sodelovala skupina JVT (angl. Joint Video Team), ki je sestavljena iz članov ISO/IEC JTC1/SC29/WG11(MPEG) in ITU-T SG16 Q.6 (VCEG). Zaradi tega je ta standard izdan tudi v obliki priporočila H.264 [322]. Glavni cilj formata H.264 je bil izboljšati stiskanje, kar so uspešno dosegli, saj dosega do 30 % boljše stiskanje kot MPEG-4 ASP in do 50 % boljše stiskanje kot MPEG-2 [233]. Tako je H.264 danes še vedno eden najbolj široko sprejetih in uporabljanih standardov za digitalni video. Standard opisuje samo stiskanje videa ter je izjemno zahteven in obsežen, saj je v zadnji dopolnitvi priporočila ITU H.264 iz leta 2024 opisan na več kot 800 straneh. Krajši opis povzemamo po [294, 320, 7, 323, 324, 233].

Videokodirnik H.264 vsebuje vse glavne dele kot dosedanji kodirniki: napoved, transformacija, kvantizacija in kodiranje entropije, kar vidimo na sliki 5.14. H.264 pa posamezne gradnike izboljšuje in prinaša nekatere novosti v osnovni koncept kodiranja. Bistvene novosti so:

- Vsak okvir je sestavljen iz makroblokov. Ker vsak makroblok napovemo in obdelamo posamezno, pride do neskladja na mejah med makrobloki. H.264 uporablja **filter** za odpravo vidnih robov med bloki (angl. deblocking filter), ki je nova komponenta in je integrirana **znotraj zanke kodirnika** (angl. in-loop filter), torej tik pred zapisom v referenčni okvir (kar vidimo na sliki 5.14, torej enako kot pri razširitvi H.263).

- Za odpravljanje medokvirne redundance omogoča določanje parametrov na nivoju rezin, ki smo jih spoznali pri MPEG-1; torej rezine tipa I, P in B. Makrobloki v rezinah tipa P so tako lahko kodirani na podlagi enega izmed predhodno kodiranih okvirjev.

- Makrobloki v rezinah tipa B niso več napovedani zgolj medokvirno



Slika 5.14: Poenostavljen diagram kodirnika H.264 s kodiranjem med okvirji in znotraj njih, pri čemer Q označuje kvantizacijo, T transformacijo, polne črte zaporedje korakov, kar vključuje tudi tok podatkov, pikčaste črte pa zgolj tok podatkov.

iz obeh smeri v času (angl. bidirectional), kot na primer makrobloki okvirjev B v MPEG-1, ampak iz poljubnih smeri v času (angl. bi-predictive). Glavni pogoj je zgolj, da je referenčne okvirje kodirnik predhodno že zakodiral [320].

- Rezine tipa P se lahko sklicujejo na okvirje poljubnega tipa, tudi na okvirje tipa B, kar je velika nadgradnja glede na predhodne pristope, kjer se okvirji tipa P niso smeli sklicevati na okvirje tipa B.

- Pri določanju vpliva referenčnih okvirjev na napoved makroblokov uvaja uteži in odmik [320], kar je koristno na primer pri spremembi osvetlitve v opazovani sceni, ki povzroči zamik vrednosti pikslov.

- H.264 prinaša napovedno kodiranje makroblokov **intra**, torej znotraj okvirja oziroma rezine (angl. intra prediction), ki ima podoben koncept delovanja kot pri formatu kodiranja slik WebP. Na sliki 5.14 vidimo, da kodiranje znotraj okvirja napove makrobloke na podlagi sosednjih makroblokov v trenutnem okvirju oziroma rezini. Razlike zakodiramo na podoben način kot pri kodiranju okvirjev P in B. Rezine tipa I so kodirane na podlagi podatkov znotraj iste rezine. H.264 predpisuje 4 načine napovedi za bloke svetlosti 16×16, 9 za bloke svetlosti 4×4 in 8×8 ter 4 za bloke barvitosti [324]. Osnovna ideja napovedi je kopiranje vrednosti iz referenčnih makroblokov v različnih smereh.

Kodirnik preizkusi vse možnosti in izbere najboljšo napoved oziroma najboljši način.

- **Kontekstno-adaptivno kodiranje z variabilno dolžino** (angl. Context-Adaptive Variable Length Coding – CAVLC) nadgrajuje kodiranje z variabilno dolžino na podlagi vnaprej pripravljene kodirne tabele (na primer Huffmanove tabele). CAVLC izbere kodirno tabelo izmed nabora vnaprej predpisanih kodirnih tabel na podlagi predhodno kodiranih/dekodiranih blokov, torej konteksta [233].

- Pristopi na podlagi kodirnih tabel delujejo slabše v primeru kodiranja posameznih vrednosti z zelo veliko frekvenco, saj minimalno število bitov na znak ne more biti manjše od enega bita. To težavo rešuje aritmetični kodirnik. H.264 uvaja **kontekstno-adaptivni binarni aritmetični kodirnik** (angl. Context-Adaptive Binary Arithmetic Coding – CABAC), ki uporablja podobno idejo kot CAVLC, torej na podlagi konteksta prilagaja stanje aritmetičnega kodirnika.

- Čeprav tudi H.264 deli okvirje v makrobloke velikosti 16×16, jih za namen kompenzacije gibanja oziroma **napovedovanja inter** lahko deli še dalje na bloke oziroma **particije** (angl. partition) velikosti 8×16, 16×8, 8×8, 8×4, 4×8 in 4×4. V tem primeru vsak blok uporablja svoj vektor gibanja, ki pa ga kodirnik napove glede na vektorje gibanja iz okoliških blokov. Stopnjo delitve določa kodirnik.

- Kot smo videli na shemi kodirnika, transformiramo (in inverzno transformiramo) bloke z razlikami. H.264 nadomešča DCT s spremenjeno celoštevilsko transformacijo, s čimer se izognemo težavam zaradi izgub pri delu z decimalnimi števili in jo hkrati pohitrimo. Podrobnosti o transformacijah, ki jih uporablja H.264, najdemo v [7].

- H.264 uvaja nova tipa okvirjev oziroma rezin za **preklapljanje med bitnimi tokovi** SP (angl. switching P) in SI (angl. switching I), ki ju ne najdemo ne v MPEG-1 in ne v MPEG-2. Uporabimo ju za sinhronizacijo, ki dekodirniku omogoča predstavitev videovsebin, ki uporabljajo različne hitrosti prenosa – tako imenovano **spajanje bitnih nizov** (angl. bit stream switching, splicing), za okrevanje po napakah, za naključni dostop in za hitrejše predvajanje tako naprej kot nazaj [323]. Okvirji SP uporabljajo napoved gibanja, s katerim izkoriščajo redundanco v zaporedju okvirjev, podobno kot okvirji P, a s to razliko, da lahko rekonstruirajo enak okvir, četudi je napovedan iz različnih referenčnih okvirjev. Ker okvirji SP uporabljajo napoved inter, potrebujejo občutno manj podatkov kot okvirji I, da bi dosegli

podobno kakovost. Okvirji SI imajo enako nalogo kot okvirji SP, a uporabljajo le napoved intra.

- H.264 omogoča **prožno videokodiranje** (angl. Scalable Video Coding – **SVC**). Idejo prožnosti so vključevali že predhodniki, na primer MPEG-2 in H.263. Žal pa je pri njih prožnost močno zapletla tako kodirnik kot dekodirnik ter privedla do občutno manjše učinkovitosti kodiranja. Dober prožni kodirnik mora generirati bitni niz, ki je občutno manjši kot neodvisno stiskanje posameznih kakovostnih nivojev, ter s tem omogočiti hkratno oddajanje več kakovostnih nivojev. H.264 SVC glede na MPEG-2 nadgrajuje z večjo učinkovitostjo in fleksibilnostjo. Večjo učinkovitost omogoča na primer z napovedovanjem nivojev iz drugih nivojev (angl. inter-layer prediction) [324].

H.264 podpira več profilov, ki so namenjeni specifičnim videoaplikacijam. Najpomembnejši so [233]:

- **Osnovni profil** (angl. base-line profile) je namenjen za podporo videotelefoniji, videokonferencam in videu, ki ga prenašamo brezžično. Uporablja rezine tipa I in P, kompenzacijo gibanja med okvirji, filter za odpravo vidnih robov znotraj zanke kodirnika, napovedovanje znotraj okvirja, za končno kodiranje pa CAVLC.

- **Razširjen profil** (angl. extened profile) je namenjen pretočnemu videu (angl. streaming video) in kot edini vključuje nova tipa rezin SI in SP.

- **Glavni profil** (angl. main profile) je načrtovan za digitalno televizijo in za shranjevanje videa. Omogoča postopke za predstavitev prepletajočega se videa, rezine B, uteži, makrobloke pa kodira s CABAC in ne več s CAVLC.

- **Visoki profil** (angl. high profile) omogoča transformacije tudi nad bloki 8×8 in ne zgolj 4×4 ter napovedi intra na nivoju blokov 8×8.

5.5.11 H.265, HEVC, MPEG-H

HEVC (angl. High Efficiency Video Coding) je naslednik H.264, objavljen leta 2013 v okviru **MPEG-H** kot standard ISO/IEC 23008-2 [325] oziroma priporočilo **H.265** [297]. Motivacija za razvoj HEVC je bila izboljšati stopnjo stiskanja glede na predhodnika H.264 zaradi večje razširjenosti videovsebin v visokih ločljivostih, na primer 8K UHD (7680×4320). HEVC dosega 50 %

boljše stiskanje podatkov pri primerljivi kakovosti videa oziroma občutno izboljša kakovost videa pri enaki hitrosti prenosa [233].

Osnovni koncept kodiranja je podoben H.264. Bistvena novost je ukinitev makroblokov, ki jih nadomeščajo bloki **CTB** (angl. coding tree block). CTB so dimenzij do 64×64 pikslov in se dinamično delijo na manjše bloke s štiriškim drevesom. Nivo delitve štiriškega drevesa določa velikost blokov znotraj CTB, nad katerimi izvajamo napovedi ali transformacije, in je spremenljiv glede na značilnosti na sliki. Manjši bloki so smiselni za kompleksnejše dele na sceni, večji bloki pa za enostavnejše. Dimenzije blokov, nad katerimi izvajamo posamezne operacije (na primer transformacija in napoved oziroma kompenzacija gibanja) so lahko dimenzij od 4×4 pa vse do 32×32 pikslov [233]. Preostale novosti izboljšujejo posamezne korake kodiranja. Najpomembnejše so:

- razširjeno kodiranje intra s 35 napovedmi,
- dodana je **diskretna sinusna transformacija** (angl. discrete sine transform – DST),
- CAVLC se ukinja, saj ga v vseh primerih nadomešča CABAC.

5.5.12 H.266, MPEG-I Part 3, VVC

VVC (angl. Versatile Video Coding) je nadgradnja HEVC iz leta 2020. Izdan je tako v obliki priporočila H.266 [326] kot del standarda ISO/IEC 23090-3 [327] družine MPEG-I. H.266 nadgrajuje korake iz H.265, tako da zmanjša količino potrebnega pomnilnika glede na H.265 za približno 40 % [328]. Najpomembnejše nadgradnje so [233, 328]:

- Dimenzije blokov, nad katerimi izvaja posamezne korake stiskanja, lahko dosegajo 128×128 pikslov.
- Medokvirno kodiranje pri kompenzaciji gibanja upošteva poleg premika tudi druge afine geometrijske transformacije (na primer rotacije in skaliranje) [328], kar izvede na podlagi dveh ali treh **vektorjev gibanja kontrolnih točk** (angl. control point motion vector – **CPMV**) v napovedanem bloku. Rotacije je možno opisati že z dvema vektorjema, zahtevnejše transformacije pa s tremi.
- Za še natančnejšo kompenzacijo gibanja je dodan nov korak za **korigiranje vektorjev gibanja z dekodirnikom** (angl. decoder-side motion vector refinement – **DMVR**). Korekcija uporablja optični tok (angl. optical flow) in s pomočjo vektorjev gibanja iz okoliških okvirjev korigira dekodirane vektorje gibanja.

- Dodana je podpora tudi za nekatere novejše aplikacije, na primer 360° video.

5.5.13 Drugi videoformati

V preteklosti je bilo razvitih še več videoformatov in kodekov.

5.5.13.1 RealVideo in DivX

RealNetworks je eno prvih uspešnih računalniških podjetij, ki je prodajalo pretočni video in avdio na internetu. Razvili so serijo lastnih kodekov **RealVideo** za različne platforme (Windows, Mac, Linux, Solaris in nekatere mobilne platforme). Prva verzija iz leta 1997 je temeljila na H.263, poznejše pa na osnovi H.264.

DivX je serija kodekov v lasti podjetja DivX ter vključuje tri kodeke: DivX (MPEG-4 part 2), DivX Plus HD (MPEG-4 AVC) in DivX HEVC Ultra HD (MPEG-H HEVC). DivX se je začel ukvarjati tudi z razvojem odprtokodnega kodeka na osnovi MPEG-4, katerega razvoj so pozneje ustavili. Razvoj kodeka je nadaljevala druga skupina pod imenom **XviD** [329]. MPEG-4 ASP je vključen tudi v drugih kodekih, na primer QuickTime-6 in Nero Digital [233].

5.5.13.2 On2 VP3 in Theora

On2 Technologies je podjetje, ki je izdelalo niz kodirnikov *TrueMotion*. Zelo znan je TrueMotion VP3, saj so kodirnik izdali v odprtokodni obliki in se odpovedali patentnim zahtevkom. Na podlagi izvorne kode je v letu 2004 nastala specifikacija odprtokodnega videoformata **Theora** [330]. Osnovni koncept kodirnika je zelo podoben MPEG-1, saj vključuje okvirje I in P, makrobloke, kompenzacijo gibanja, DCT in Huffmanov kodirnik. Ne omogoča pa okvirjev B in prožnosti. Theora vključuje tudi filter za odpravo vidnih robov med bloki. V preteklosti smo kodirnik pogosto uporabljali v vsebniku Ogg v kombinaciji z avdioformatom Vorbis.

5.5.13.3 WMV 9 in VC-1

WMV 9 (angl. Windows Media Video) je format podjetja Microsoft, pozneje pa ga je Microsoft odprl in leta 2006 standardiziral v okviru SMPTE (angl. Society of Motion Picture and Television Engineers) kot SMPTE 421 [331] z imenom **VC-1**. VC-1 temelji na klasičnih pristopih od kompenzacije gibanja do DCT in Huffmanovih tabel in je primerljiv s H.264 [305]. VC-1 je poleg

H.262 in H.264 eden izmed privzetih formatov kodiranja za predvajalnike Blu-ray.

5.5.13.4 AVS

AVS (angl. Avdio Video Standard) je projekt kitajske vlade, da bi razvili svoj videostandard. Poleg raziskovalnih ustanov so se projektu pridružila visokotehnološka kitajska podjetja kot Huawei, TCL, Skyworth, Hisense in druga ter začeli s projektom leta 2002. AVS je del oddajnega sistema digitalne TV na Kitajskem. Izvorni standard ima oznako AVS1 in je standardiziran pod oznako IEEE 1857.1. AVS1 uporablja podobne koncepte kot H.264 [233]. AVS2 je bil razvit pozneje (objavljen leta 2013 kot IEEE 1857.4) in je namenjen za 4k UHD. AVS2 temelji na blokih sprejemljive velikosti, katerih velikost se lahko spreminja od 8×8 do 64×64 pikslov. Podatkovna struktura, ki nadzira sistem neenako velikih makroblokov, je štiriško drevo [332], torej zelo podobno H.265. AVS3 je nadgradnja AVS2, namenjena za 8K UHD. Izdana je pod oznako IEEE 1857.10-2021. AVS3 uporablja nekatere koncepte iz H.266, kot je podpora za afine transformacije pri kompenzaciji gibanja [333].

5.5.13.5 VP8, VP9 in AV1

Podjetja On2 Technologies je razvilo TrueMotion **VP8**, ki je po delovanju primerljiv s H.264, saj omogoča napovedi znotraj okvirjev in filter znotraj zanke kodirnika za odpravo mej med bloki. On2 Technologies je nato prevzelo podjetje Google, ki je v letu 2010 izdalo VP8 kot odprtokodno specifikacijo [305]. Format je delno zaščiten s patenti v lasti podjetja Google, ki pa dovoljuje prosto uporabo. VP8 je postal jedro formata WebM podjetja Google. Kodiranja intra iz VP8 pa uporablja format WebP.

Google je leta 2013 na podlagi formata VP8 izdal odprtokodno nadgradnjo **VP9** [305], ki omogoča različno velike bloke (od 4×4 do 64×64 pikslov), razširja kodiranje intra z več napovedmi, omogoča DCT nad bloki velikosti 32×32 pikslov, razširja prožnost, izboljšuje aritmetični kodirnik in omogoča brezizgubno stiskanje. VP9 je bil namenjen predvsem za Googlovo videoplatformo YouTube. Zato je VP9, v nasprotju s HEVC, zelo dobro podprt tudi v spletnih brskalnikih in mobilnih platformah. V kombinaciji s kodiranjem avdia Opus se zelo pogosto uporablja za kodiranje vsebin v datotekah WebM [233].

AV1 je odprtokodna nadgradnja VP9 iz leta 2018, namenjena prenosu visoko kakovostnega videa po internetu. AV1 je bil razvit pod okriljem združenja **AOMedia** (angl. Alliance for Open Media), ki je zelo širok indu-

strijski konzorcij (Google, Mozilla, Cisco, Amazon, Intel, Microsoft, Netflix), ustanovljen leta 2015 za razvoj odprtih standardov kodiranja multimedijskih vsebin.

AV1 glede na VP9 omogoča še večjo fleksibilnost pri velikosti blokov (velikosti do 128×128 pikslov) z uporabo štiriških dreves, 56 načinov napovedovanja pri kodiranju znotraj okvirjev, kompenzacijo gibanja s podporo za afine transformacije in kombiniranje različnih transformacij (DCT, DST) nad bloki velikosti do 64×64 [305]. Kodiranje znotraj okvirjev je uporabljeno pri formatu kodiranja slik AVIF.

5.5.13.6 MPEG EVC

Problematike licenciranja in patentov so se začeli zavedati tudi pri združenju MPEG. Tako so izdali format MPEG-5 **EVC** (angl. Essential Video Coding) in ga standardizirali pod oznako ISO/IEC 23094-1 [334]. EVC v osnovnem profilu (angl. baseline) uporablja postopke, za katere lahko pričakujemo uporabo brez plačila morebitnih licenčnin. To vključuje tehnike izpred dvajsetih let, za katere lahko pričakujemo potek patentnih pravic, in postopke, ki so javno objavljeni s klavzulo o odpovedi licenčninam [305].

EVC se zgleduje po HEVC in ima v osnovnem profilu velikosti blokov do 64×64 pisklov, uporablja napovedi pri kodiranju znotraj okvirjev, kompenzacijo gibanja, variabilno velike bloke DCT in aritmetični kodirnik, pri katerem je patent že potekel [305]. Glavni profil pa vsebuje postopke, ki izboljšujejo učinkovitost stiskanja in so pod patentno zaščito, kar vključuje na primer dinamično velike bloke, podporo za afine transformacije pri kompenzaciji gibanja in korigiranje vektorjev gibanja na strani dekodirnika (ideja iz H.266).

5.6 Seznam nalog

- Razmislite, zakaj pri analognem signalu določamo zgolj navpično ločljivost in ne vodoravne.
- Predpostavimo, da za kodiranje videa uporabljamo kodirnik MPEG-2, torej kodiramo z okvirji IPB. Napišite program, ki za vse okvirje na izhod izpiše njihovo številko in tip glede na uporabniško podano frekvenco okvirjev I in P. Nato naj aplikacija po vzoru iz slike 5.5 preuredi vrstni red okvirjev, primeren za dekodiranje, ter izpiše to zaporedje okvirjev.
- Napišite program, ki bo, glede na priporočilo H.262 [13] oziroma enačbi 5.2 in 5.3, sliko iz barvnega modela RGB pretvoril v barvni model

YCbCr. Program naj vrednosti pikslov zaokroži na cela števila. Prav tako implementirajte pretvorbo iz modela YCbCr v model RGB. Z metrikama SSIM in PSNR analizirajte podobnost med izvorno in pretvorjeno sliko RGB. Ocenite, kakšno izgubo povzroči pretvorba barvnih modelov. Uporabite slike z različnimi motivi in različno globino barvnih kanalov.

- Izračunajte, koliko zlogov zahteva enourni prepleten video 8K pri 30 fps s podvzorčenjem barv 4:2:0, pri čemer za zapis vsake komponente YCbCr uporabimo 8 bitov.

- Napišite bralnik datotek AVI, ki bo izpisal število in imena multimedijskih podatkovnih tokov, ter za vsakega izmed njih tudi njihov način kodiranja.

- Implementirajte PSNR in integralno projekcijo kot meri ujemanja blokov med dvema slikama. Izmerite njune čase izvajanja glede na velikost blokov. Analizirajte, kako se spremeni čas izvajanja integralne projekcije, če na prvi sliki predpostavite vnaprej izračunane podpise (tj. vsote vrstic in stolpcev).

- Implementirajte logaritemsko iskanje ujemanja blokov. Kot mero ujemanja uporabite PSNR. Nad dvema zaporednima okvirjema videa poženite algoritem in izmerite čas izvajanja glede na ločljivost slike in glede na velikost blokov.

- Poiščite prepleten video s hitrim gibanjem in ga predvajajte v predvajalniku, ki naj ne odpravlja vidnega prepletanja (angl. deinterlacing). Okvir z vidnim hitrim gibanjem shranite v sliko. Uporabite takšen okvir, kjer so opazne motnje zaradi prepletanja. Nato sliko stisnite na dva različna načina:

  1. Sliko stisnite s formatom JPEG tako močno, da bodo vidne motnje zaradi stiskanja.
  2. Prvotno sliko shranite v dve ločeni sliki. V prvi sliki naj bodo lihe vrstice prvotne slike, v drugi pa sode. Obe sliki stisnite ločeno s formatom JPEG (uporabite enako stopnjo stiskanja kot prej), zatem pa obe sliki sestavite v skupno sliko.

  Preverite, ali je pri drugem načinu stiskanja res manj vidnih motenj kot pri prvem.

- Iz videa s hitrim gibanjem shranite dva zaporedna okvirja. V urejevalniku slik na manjši površini slike ročno preverite, kakšen delež blokov

dimenzij 16×16 pikslov lahko predstavite zgolj z bloki iz predhodne slike. Nato ročno preverite, koliko blokov bi bilo smiselno rotirati in skalirati.

Literatura

[1] United States. Department of Defense, *Military Standard: Common Long Haul and Tactical Communication System Technical Standards*. Institute for Simulation and Training collection, Department of Defense, 1972.

[2] B. Žalik, M. Zadravec in D. Podgorelec, *Računalniške periferne naprave in uporabniški vmesniki: učbenik*. Fakulteta za elektrotehniko, računalništvo in informatiko, 2002.

[3] J. Jarvis, C. Judice in W. Ninke, A survey of techniques for the display of continuous tone pictures on bilevel displays, *Computer graphics and image processing*, 5(1), 13–40, 1976.

[4] Y.-H. Yu, C.-C. Changa in Y.-C. Hub, Hiding secret data in images via predictive coding, *Pattern recognition*, 38, 691–705, 2005.

[5] Lindosland, Equal-loudness contours created by lindosland using OpenOffice Draw by reference to ISO 226:2003 standard. https://upload.wikimedia.org/wikipedia/commons/4/47/Lindos1.svg in https://commons.wikimedia.org/wiki/File:Lindos1.svg, 2005, zadnji dostop feb. 2025.

[6] Lindosland, A, C and D weighting curves, created by lindosland using OpenOffice Draw. https://upload.wikimedia.org/wikipedia/commons/3/39/Acoustic_weighting_curves_%281%29.svg in https://commons.wikimedia.org/wiki/File:Acoustic_weighting_curves_(1).svg, 2005, zadnji dostop feb. 2025.

[7] D. Salomon in G. Motta, *Handbook of data compression*. Springer, 5. izd., 2010.

[8] S. Shlien, Guide to MPEG-1 audio standard, *IEEE Transactions on Broadcasting*, 40(4), 206–218, 1994.

[9] H.261 : Video codec for audiovisual services at p x 64 kbit/s, Recommendation H.261 (12/90). 1990. International Telecommunication Union (ITU).

[10] T. Ebrahimi in M. Kunt, Visual data compression for multimedia applications, *Proceedings of the IEEE*, 86(6), 1109–1125, 1998.

[11] A. C. Bovik, *Handbook of image and video processing*. Academic press, 2010.

[12] M. Kutz, *Handbook of Measurement in Science and Engineering, Volume 1*. Wiley, 2015.

[13] H.262 : Information technology – Generic coding of moving pictures and associated audio information: Video, Recommendation H.262 (07/95). 1995. International Telecommunication Union (ITU).

[14] C. A. Peláez, A. Solano, T. Granollers in C. Collazos, Methodologies and Trends in Multimedia Systems: A Systematic Literature Review, *Social Computing and Social Media. Design, Human Behavior and Analytics* (G. Meiselwitz, ur.), (Cham), 109–127, Springer International Publishing, 2019.

[15] N. Chapman in J. Chapman, *Digital multimedia*. Wiley, Chichester, 2. izd., 2004.

[16] Information technology – 7-bit coded character set for information processing interchange, ISO 646. 1973. International Organization for Standardization.

[17] Obrada podataka – Skup znakova za razmenu podataka kodiranih sa 7 bitova za hrvatskosrpsko i slovenačko latinično pismo, JUS I.B1.002. 1982. Jugoslovanski zavod za standardizacijo.

[18] Serbocroatian and Slovenian Latin Alphabet, ISO-IR-141. 1988. International Organization for Standardization.

[19] Obrada podataka – Skup znakova za razmenu podataka kodiranih sa 7 bitova za srpskohrvatsko ćirilično pismo, JUS I.B1.003. 1982. Jugoslovanski zavod za standardizacijo.

[20] Serbocroatian Cyrillic Alphabet, ISO-IR-146. 1988. International Organization for Standardization.

[21] Obrada podataka – Skup znakova za razmenu podataka kodiranih sa 7 bitova za makedonsko ćirilično pismo, JUS I.B1.004. 1982. Jugoslovanski zavod za standardizacijo.

[22] Macedonian Cyrillic Alphabet, ISO-IR-147. 1988. International Organization for Standardization.

[23] Information technology – 8-bit single-byte coded graphic character sets, ISO/IEC 8859. 1999. International Organization for Standardization and International Electrotechnical Commission.

[24] Information technology – Universal Coded Character Set (UCS), ISO/IEC 10646. 2017. International Organization for Standardization and International Electrotechnical Commission.

[25] Unicode Consortium, About the Unicode Consortium. https://home.unicode.org/about-unicode/, zadnji dostop jul. 2024.

[26] B. Žalik, *Aplikacije računalniških algoritmov*. Univerza v Mariboru, Univerzitetna založba; Fakulteta za elektrotehniko, računalništvo in informatiko, 1 izd., 2023.

[27] V. J. Hodge in J. Austin, A comparison of standard spell checking algorithms and a novel binary neural approach, *IEEE transactions on knowledge and data engineering*, 15(5), 1073–1081, 2003.

[28] M. Miłkowski, Developing an open-source, rule-based proofreading tool, *Software: Practice and Experience*, 40(7), 543–566, 2010.

[29] T. Poibeau, *Machine translation*. MIT Press, 2017.

[30] F. M. Liang, *Word Hy-phen-a-tion by Com-put-er*. Citeseer, 1983.

[31] V. A. Alfred, S. L. Monica in D. U. Jeffrey, *Compilers - Principles, Techniques & Tools - Second Edition*. Addison-Wesley, Reading, 2006.

[32] M. E. Palma, P. Salza in H. C. Gall, On-the-fly syntax highlighting using neural networks, *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 269–280, 2022.

[33] M. Bertaccini, *Cryptography Algorithms: A guide to algorithms in blockchain, quantum cryptography, zero-knowledge protocols, and homomorphic encryption*. Packt Publishing, 2022.

[34] J. Duda, Asymmetric numeral systems, 2009. arXiv, https://arxiv.org/abs/0902.0271.

[35] J. Duda, Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding, 2014. arXiv, https://arxiv.org/abs/1311.2540.

[36] P. Skibiński, S. Grabowski in J. Swacha, Effective asymmetric XML compression, *Software: Practice and Experience*, 38(10), 1027–1047, 2008.

[37] S. Sakr, XML compression techniques: A survey and comparison, *Journal of Computer and System Sciences*, 75(5), 303–322, 2009.

[38] P. Skibiński, Improving HTML Compression, *Informatica*, 33, 363–373, 2009.

[39] M. Hutter, 500'000? Prize for Compressing Human Knowledge. http://prize.hutter1.net/, zadnji dostop jun. 2024.

[40] L. P. Deutsch, DEFLATE Compressed Data Format Specification version 1.3. http://www.ietf.org/rfc/rfc1951, zadnji dostop jul. 2024.

[41] Phillip W. Katz, String searcher, and compressor using same, US5051745A, 1990.

[42] G. Roelofs, J.-L. Gailly in M. Adler, zlib. http://www.zlib.net/, 2019, zadnji dostop jul. 2024.

[43] P. Deutsch in J.-L. Gailly, Zlib compressed data format specification version 3.3, teh. por., Network Working Group, https://www.rfc-editor.org/rfc/rfc1950, 1996, zadnji dostop dec. 2024.

[44] P. Deutsch, GZIP file format specification version 4.3, teh. por., Network Working Group, https://www.rfc-editor.org/rfc/rfc1952.html, 1996, zadnji dostop dec. 2024.

[45] J. Alakuijala, A. Farruggia, P. Ferragina, E. Kliuchnikov, R. Obryk, Z. Szabadka in L. Vandevenne, Brotli: A general-purpose data compressor, *ACM Transactions on Information Systems (TOIS)*, 37(1), 1–30, 2018.

[46] J. Alakuijala in Z. Szabadka, RFC 7932: Brotli Compressed Data Format. https://www.rfc-editor.org/info/rfc7932, 2016, zadnji dostop avg. 2024.

[47] Noto, Distribution site for Noto fonts. https://github.com/notofonts/notofonts.github.io/tree/main/fonts, zadnji dostop jun. 2024.

[48] Microsoft, Font redistribution FAQ (Frequently Asked Questions) for Windows. hhttps://learn.microsoft.com/en-us/typography/fonts/font-faq, 2024, zadnji dostop jul. 2024.

[49] Y. Haralambous, *Fonts & encodings*. O'Reilly Media, Inc., 2007.

[50] Information technology – Font information interchange, ISO/IEC 9541. 1991. International Organization for Standardization and International Electrotechnical Commission.

[51] Apple, TrueType Reference Manual. https://developer.apple.com/fonts/TrueType-Reference-Manual/, 2019, zadnji dostop jul. 2024.

[52] Information technology – Coding of audio-visual objects – Part 22: Open Font Format, ISO/IEC 14496-22. 2015. International Organization for Standardization and International Electrotechnical Commission.

[53] W3C, WOFF File Format 2.0, W3C Recommendation, 08 August 2024. https://www.w3.org/TR/WOFF2/, 2024, zadnji dostop avg. 2024.

[54] O. Taylor, Pango, an open-source Unicode text layout engine, *Proceedings of 25th Internationalization and Unicode Conference*, 2004.

[55] C. Rapp, T. Heilmann in O. Kruse, Beyond MS Word: Alternatives and Developments, *Digital Writing Technologies in Higher Education: Theory, Research, and Practice*, 33–47, Springer, 2023.

[56] Information processing – Text and office systems – Standard Generalized Markup Language (SGML), ISO 8879. 1986. International Organization for Standardization.

[57] S. J. DeRose, *The SGML FAQ book: understanding the foundation of HTML and XML*, 7. Springer Science & Business Media, 1997.

[58] CommonMark, A strongly defined, highly compatible specification of Markdown. https://github.com/commonmark/commonmark-spec in https://commonmark.org/, zadnji dostop jul. 2024.

[59] J. J. White, Using Markup Languages for Accessible Scientific, Technical, and Scholarly Document Creation., *Journal of Science Education for Students with Disabilities*, 25(1), 2022.

[60] D. Allen in S. White, AsciiDoc Language Documentation. https://docs.asciidoctor.org/asciidoc/latest/, zadnji dostop jul. 2024.

[61] Information processing – Text and office systems – Office Document Architecture (ODA) and interchange format, ISO 8613. 1989. International Organization for Standardization.

[62] Office Open XML File Formats, ECMA-376. 2016. Ecma International.

[63] Information technology – Document description and processing languages – Office Open XML File Formats, ISO/IEC 29500-1. 2008. International Organization for Standardization and International Electrotechnical Commission.

[64] Information technology – Open Document Format for Office Applications (OpenDocument) v1.0, ISO/IEC 26300. 2006. International Organization for Standardization and International Electrotechnical Commission.

[65] SEIKO EPSON Corporation, EPSON ESC/P, Reference Manual. https://files.support.epson.com/pdf/general/escp2ref.pdf, 1990, zadnji dostop jul. 2024.

[66] Hewlett-Packard, PCL 5, Printer Language, Technical, Quick Reference Guide. http://www.hp.com/ctg/Manual/bpl13205.pdf, zadnji dostop jul. 2024.

[67] Adobe Systems Inc, CORPORATE, *PostScript language reference manual*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[68] R. I. Soare, *Turing computability: Theory and applications*. Springer Berlin, Heidelberg, 2016.

[69] Document management – Portable document format – Part 1: PDF 1.7, ISO 19005-1. 2008. International Organization for Standardization.

[70] Document management – Electronic document file format for long-term preservation – Part 1: Use of PDF 1.4 (PDF/A-1), ISO 19005-1. 2005. International Organization for Standardization.

[71] The TUG DVI Driver Standards Committee, The DVI Driver Standard, Level 0. http://mirrors.ctan.org/dviware/driv-standard/level-0/dvistd0.pdf, 2004, zadnji dostop jul. 2024.

[72] F. Sabry, *Optical Character Recognition: Fundamentals and Applications*. Artificial Intelligence, One Billion Knowledgeable, 2023.

[73] D. Yu in L. Deng, *Automatic speech recognition*. Springer, 2016.

[74] Y. Bassil, Steganography & the art of deception: a comprehensive survey, *Int. Journal on Latest Trends Computing*, 4(3), 128–138, 2013.

[75] M. Agarwal, Text Steganographic Approaches: A Comparison, *International Journal of Network Security & Its Applications*, 5(1), 91–106, 2013.

[76] M. A. Majeed, R. Sulaiman, Z. Shukur in M. K. Hasan, A review on text steganography techniques, *Mathematics*, 9(21), 2021.

[77] S. H. Low, N. F. Maxemchuk, J. T. Brassil in L. O. Gorman, Document marking and identification using both line and word shifting, *Proceedings of the* 14^{th} *Annual Joint Conf. of the IEEE Computer and Communication Societies*, 853–860, 1995.

[78] W. Bender, D. Gruhl, N. Morimoto in A. Lu, Techniques for data hiding, *IBM Systems Journal*, 35, 313–336, 1996.

[79] A. Beechick, When Choosing a Sorting Algorithm, Do You Want Fast, Faster, or Fastest?. http://www.aislebyaisle.com/cb/access/sorting/beechicksort.htm, zadnji dostop jul. 2024.

[80] Allen Beechick, Apparatus and method for sorting a list, US5218700A, 1993.

[81] T. Peters, Timsort description. http://svn.python.org/projects/python/trunk/Objects/listsort.txt, zadnji dostop 2024.

[82] N. Auger, N. C in C. Pivoteau, Merge Strategies: from Merge Sort to TimSort, *HAL*, https://hal-upec-upem.archives-ouvertes.fr/hal-01212839v2, 2015.

[83] N. Auger, V. Jugé, C. Nicaud in C. Pivoteau, On the Worst-Case Complexity of TimSort, *26th Annual European Symposium on Algorithms (ESA 2018)* (Y. Azar, H. Bast in G. Herman, ur.), 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (Dagstuhl, Germany), 4:1–4:13, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

[84] P. McIlroy, Optimistic sorting and information theoretic complexity, *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, 467–474, 1993.

[85] D. Podgorelec in G. Klajnšek, Acceleration of sweep-line technique by employing smart quicksort, *Information sciences*, 169(3–4), 383–408, 2005.

[86] Center za jezikovne vire in tehnologije, Slovar sopomenk sodobne slovenščine. https://viri.cjvt.si/sopomenke/slv/, zadnji dostop dec. 2024.

[87] N. Guid, *Računalniška grafika*. Fakulteta za elektrotehniko, računalništvo in informatiko, 2001.

[88] D. J. Calkins, *Mapping color perception to a physiological substrate*. The MIT Press, 1993.

[89] B. D. Judd in G. Wyszecki, *Color in Business, Science and Industry*. Wiley Series in Pure and Applied Optics, 3. izd., 1975.

[90] C. A. Curcio, K. R. Sloan, R. E. Kalina in A. E. Hendrickson, Human photoreceptor topography, *Journal of comparative neurology*, 292(4), 497–523, 1990.

[91] A. Fiorentini, Mach Band Phenomena, *Visual Psychophysics* (D. Jameson in L. M. Hurvich, ur.), 188–201, Berlin, Heidelberg: Springer Berlin Heidelberg, 1972.

[92] W3C, PNG (Portable Network Graphics) Specification Version 1.0: 13. Appendix: Gamma Tutorial. https://www.w3.org/TR/PNG-GammaAppendix.html, zadnji dostop jul. 2025.

[93] M. Stokes, M. Anderson, S. Chandrasekar in R. Motta, A Standard Default Color Space for the Internet - sRGB. https://www.w3.org/Graphics/Color/sRGB.html, zadnji dostop jul. 2025.

[94] P. Shirley, M. Ashikhmin in S. Marschner, *Fundamentals of computer graphics*. AK Peters/CRC Press, 2009.

[95] Image technology colour management – Architecture, profile format and data structure – Part 1: Based on ICC.1:2010, ISO 15076-1. 2010. International Organization for Standardization.

[96] R. Lorencon, *Elektronski elementi in vezja*. Studio Maya, 1996.

[97] D. Durini, *High performance silicon imaging: fundamentals and applications of CMOS and CCD sensors*. Woodhead Publishing, 2019.

[98] T. Porter in T. Duff, Compositing Digital Images, *Computer Graphics*, 18(3), 253–259, 1984.

[99] A. R. Smith, Alpha and the history of Digital Compositing – Microsoft Technical Memo 7, 1995.

[100] GSMArena, Flashback: the Nokia 808 PureView was from the future. https://www.gsmarena.com/flashback_nokia_808_pureview-news-40064.php, 2019, zadnji dostop nov. 2024.

[101] T. Acharya in P. Tsai, *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*. Wiley, 2005.

[102] K. Plataniotis in A. Venetsanopoulos, *Color Image Processing and Applications*. Digital Signal Processing, Springer Berlin Heidelberg, 2013.

[103] V. Britanak, P. C. Yip in K. R. Rao, *Discrete cosine and sine transforms: general properties, fast algorithms and integer approximations*. Elsevier, 2010.

[104] Microsoft Corporation, BITMAPV5HEADER structure (wingdi.h). https://learn.microsoft.com/en-us/windows/win32/api/wingdi/ns-wingdi-bitmapv5header, 2024, zadnji dostop jul. 2024.

[105] Microsoft Corporation, [MS-WMF] - v20240423, Windows Metafile Format. https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-wmf/4813e7fd-52d0-4f42-965f-228c8b7488d2?redirectedfrom=MSDN, 2024, zadnji dostop jul. 2024.

[106] The BMP file format. https://www.prepressure.com/library/file-formats/bmp, zadnji dostop 2019.

[107] The GIF file format. https://www.prepressure.com/library/file-formats/gif, zadnji dostop jun. 2024.

[108] GIF File Format Summary. https://www.fileformat.info/format/gif/egff.htm, zadnji dostop jun. 2024.

[109] CompuServe Incorporated, Columbus, Ohio, GRAPHICS INTERCHANGE FORMAT(sm) - Version 89a. https://www.w3.org/Graphics/GIF/spec-gif89a.txt, 1990, zadnji dostop jul. 2024.

[110] Terry A. Welch, High speed data compression and decompression apparatus and method , US4558302A, 1985.

[111] TIFF File Format Summary. https://www.fileformat.info/format/tiff/egff.htm, zadnji dostop jul. 2024.

[112] The TIFF file format. https://www.prepressure.com/library/file-formats/tiff, zadnji dostop jul. 2024.

[113] Sustainability of Digital Formats: Planning for Library of Congress Collections, TIFF, Revision 6.0. https://www.loc.gov/preservation/digital/formats/fdd/fdd000022.shtml, zadnji dostop jul. 2024.

[114] Adobe Photoshop® TIFF Technical Notes. https://web.archive.org/web/20180810205806/https://www.adobe.io/content/udp/en/open/standards/TIFF/_jcr_content/contentbody/download_1370394226/file.res/TIFFphotoshop.pdf, 2002, zadnji dostop jul. 2024.

[115] Sustainability of Digital Formats: Planning for Library of Congress Collections, GeoTIFF, Revision 1.0. https://www.loc.gov/preservation/digital/formats/fdd/fdd000279.shtml, zadnji dostop jul. 2024.

[116] Electronic still picture imaging – Removable memory – Part 2: TIFF/EP image data format, ISO 12234-2:2001. 2001. International Organization for Standardization.

[117] Graphic technology – Prepress digital data exchange – Tag image file format for image technology (TIFF/IT), ISO 12639:2004. 2004. International Organization for Standardization.

[118] Sustainability of Digital Formats: Planning for Library of Congress Collections, BigTIFF. https://www.loc.gov/preservation/digital/formats/fdd/fdd000328.shtml, zadnji dostop jul. 2024.

[119] Camera & Imaging Products Association, Exchangeable image file format for digital still cameras: Exif Version 3.0. https://www.cipa.jp/std/documents/download_e.html?DC-008-Translation-2023-E, 2023, zadnji dostop jul. 2024.

[120] Graphic technology – Extensible metadata platform (XMP) – Part 1: Data model, serialization and core properties, ISO 16684-1:2019. 2019. International Organization for Standardization.

[121] Graphic technology – Extensible metadata platform (XMP) – Part 2: Description of XMP schemas using RELAX NG, ISO 16684-2:2014. 2019. International Organization for Standardization.

[122] The PNG file format. https://www.prepressure.com/library/file-formats/png, zadnji dostop 2024.

[123] PNG File Format Summary. https://www.fileformat.info/format/png/egff.htm, zadnji dostop 2019.

[124] W3C, Portable Network Graphics (PNG) Specification (Third Edition). https://www.w3.org/TR/png/, 2025, zadnji dostop jul. 2025.

[125] Information technology – Computer graphics and image processing – Portable Network Graphics (PNG): Functional specification, ISO/IEC 15948. 2004. International Organization for Standardization and International Electrotechnical Commission.

[126] LibPNG, Chapter 10. Gamma Correction and Precision Color. http://www.libpng.org/pub/png/book/chapter10.html, zadnji dostop jul. 2024.

[127] Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1, ISO/IEC 8859-1. 1998. International Organization for Standardization and International Electrotechnical Commission.

[128] B. Žalik, D. Podgorelec, I. Kolingerová, D. Strnad in Š. Kohek, A case study on entropy-aware block-based linear transforms for lossless image compression, *Scientific Reports*, 14(1), 2024.

[129] MNG (Multiple-image Network Graphics) Format Version 1.0. http://www.libpng.org/pub/mng/spec/, zadnji dostop jul. 2024.

[130] Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines, ISO/IEC 10918-1. 1994. International Organization for Standardization and International Electrotechnical Commission.

[131] Information technology – Digital compression and coding of continuous-tone still images – Requirements and guidelines, CCITT Rec. T.81. 1992. International Telecommunication Union (ITU).

[132] Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF), ISO/IEC 10918-5. 2013. International Organization for Standardization and International Electrotechnical Commission.

[133] Overview of JPEG 1. https://jpeg.org/jpeg/index.html, zadnji dostop 2019.

[134] Information technology – Coded representation of picture and audio information – Progressive bi-level image compression, ISO/IEC 11544. 1993. International Organization for Standardization and International Electrotechnical Commission.

[135] Information technology – Coded representation of picture and audio information – Progressive bi-level image compression, ITU-T T.82. 1993. International Telecommunication Union (ITU).

[136] Overview of JBIG. https://jpeg.org/jbig/, zadnji dostop avg. 2024.

[137] M. Kuhn, JBIG-KIT. https://www.cl.cam.ac.uk/~mgk25/jbigkit/, zadnji dostop avg. 2024.

[138] Information technology – Lossy/lossless coding of bi-level images, ISO/IEC 14492. 2001. International Organization for Standardization and International Electrotechnical Commission.

[139] Information technology – Lossy/lossless coding of bi-level images, ITU-T T.88. 2000. International Telecommunication Union (ITU).

[140] R. Miyamoto, Arithmetic coding/decoding apparatus of MQ-Coder system and renormalization method, US6765515B2, 2004.

[141] N. Saraswat in H. Ghosh, A study on size optimization of scanned textual documents, *Image and Video Technology: 7th Pacific-Rim Symposium, PSIVT 2015, Auckland, New Zealand, November 25-27, 2015, Revised Selected Papers 7*, 75–86, Springer, 2016.

[142] A. Mikheev, L. Vincent, M. Hawrylycz in L. Bottou, Electronic document publishing using DjVu, *Document Analysis Systems V: 5th International Workshop, DAS 2002 Princeton, NJ, USA, August 19–21, 2002 Proceedings 5*, 480–490, Springer, 2002.

[143] Information technology – Lossless and near-lossless compression of continuous-tone still images – Part 1: Baseline, ISO/IEC 14495-1:1999. 1999. International Organization for Standardization and International Electrotechnical Commission.

[144] JPEG, Overview of JPEG LS. https://jpeg.org/jpegls/, zadnji dostop avg. 2024.

[145] M. Weinberger, G. Seroussi in G. Sapiro, The LOCO-I Lossless Image Compression Algorithm: principles and Standardization into JPEG-LS, teh. por., HP Computer Systems laboratory, 1998.

[146] M. J. Weinberger, G. Seroussi in G. Sapiro, The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization Into JPEG-LS, *IEEE Transactions on image processing*, 9(8), 1309–1324, 2000.

[147] D. Špelič, *Postopek brezizgubnega stiskanja razčlenjenih vokselskih podatkov: doktorska disertacija.* Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, 2011.

[148] Information technology – Lossless and near-lossless compression of continuous-tone still images – Baseline, ITU-T T.87. 1998. International Telecommunication Union (ITU).

[149] Information technology – Lossless and near-lossless compression of continuous-tone still images – Extensions, ISO/IEC 14495-2:2003. 2003. International Organization for Standardization and International Electrotechnical Commission.

[150] Information technology – Lossless and near-lossless compression of continuous-tone still images – Extensions, ITU-T T.870. 2003. International Telecommunication Union (ITU).

[151] Information technology – JPEG 2000 image coding system – Core coding system, ISO/IEC 15444-1. 2000. International Organization for Standardization and International Electrotechnical Commission.

[152] Information technology – JPEG 2000 image coding system – Extensions, ISO/IEC 15444-2. 2004. International Organization for Standardization and International Electrotechnical Commission.

[153] D. S. Taubman in M. W. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*. Springer Science+Business Media, 2013.

[154] C. K. Chui, *An introduction to wavelets*. Academic press, 1992.

[155] T. Acharya in P.-S. Tsai, *JPEG2000 standard for image compression: concepts, algorithms and VLSI architectures*. John Wiley & Sons, 2004.

[156] G. Impoco, JPEG2000 - A Short Tutorial. http://www.impoco.it/files/impoco_2004_tutorial_JPEG2000.pdf, zadnji dostop 2019.

[157] An image format for the Web. https://developers.google.com/speed/webp/, zadnji dostop 2019.

[158] J. Bankoski, P. Wilkins in Y. Xu, Technical overview of VP8, an open source video codec for the web, *2011 IEEE International Conference on Multimedia and Expo*, 1–6, IEEE, 2011.

[159] G. Ginesun, M. Pintus in D. D. Giusto, Objective assessment of the WebP image coding algorithm, *Signal Processing:Image Communication*, 27, 867–874, 2012.

[160] Adobe, Digital Negative (DNG) Specification, Version 1.7.1.0, september 2023. https://helpx.adobe.com/content/dam/help/en/photoshop/pdf/DNG_Spec_1_7_1_0.pdf, 2023, zadnji dostop avg. 2024.

[161] Information technology – Scalable compression and coding of continuous-tone still images – Part 1: Core coding system specification, ISO/IEC 18477-1. 2015. International Organization for Standardization and International Electrotechnical Commission.

[162] A. Artusi, R. K. Mantiuk, T. Richter, P. Korshunov, P. Hanhart, T. Ebrahimi in M. Agostinelli, JPEG XT: A compression standard for HDR and WCG images [standards in a nutshell], *IEEE Signal Processing Magazine*, 33(2), 118–124, 2016.

[163] H. S. Malvar, Fast Progressive Image Coding Without Wavelets, *Data Compression Conference*, Institute of Electrical and Electronics Engineers, Inc., 2000.

[164] Information technology – JPEG XR image coding system – Part 2: Image coding specification, ISO/IEC 29199-2. 2012. International Organization for Standardization and International Electrotechnical Commission.

[165] T.832 : Information technology – JPEG XR image coding system – Image coding specification, ITU-T T.832. 2009. International Telecommunication Union (ITU).

[166] Information technology – JPEG XS low-latency lightweight image coding system – Part 1: Core coding system, ISO/IEC 21122-1. 2019. International Organization for Standardization and International Electrotechnical Commission.

[167] Information technology – Advanced image coding and evaluation – Part 2: Evaluation procedure for nearly lossless coding, ISO/IEC 29170-2. 2015. International Organization for Standardization and International Electrotechnical Commission.

[168] A. Descampe, T. Richter, T. Ebrahimi, S. Foessel, J. Keinert, T. Bruylants, P. Pellegrin, C. Buysschaert in G. Rouvroy, JPEG XS–A new standard for visually lossless low-latency lightweight image coding, *Proceedings of the IEEE*, 109(9), 1559–1577, 2021.

[169] Information technology – JPEG XL image coding system – Part 1: Core coding system, ISO/IEC 18181-1. 2022. International Organization for Standardization and International Electrotechnical Commission.

[170] J. Alakuijala, J. Sneyers, L. Versari in J. Wassenberg, JPEG White Paper: JPEG XL Image Coding System, *ISO/IEC JTC 1/SC 29/WG1 N100400*, 2023.

[171] ISO/IEC 21794-1:2020 Information technology – Plenoptic image coding system (JPEG Pleno) – Part 1: Framework, ISO/IEC 21794-1. 2020. International Organization for Standardization and International Electrotechnical Commission.

[172] Information technology – Plenoptic image coding system (JPEG Pleno) – Part 2: Light field coding, ISO/IEC 21794-2. 2021. International Organization for Standardization and International Electrotechnical Commission.

[173] ISO/IEC 21794-3:2021 Information technology – Plenoptic image coding system (JPEG Pleno) – Part 3: Conformance testing, ISO/IEC 21794-3. 2021. International Organization for Standardization and International Electrotechnical Commission.

[174] Information technology – Plenoptic image coding system (JPEG Pleno) – Part 5: Holography, ISO/IEC 21794-5. 2024. International Organization for Standardization and International Electrotechnical Commission.

[175] Information technology – Plenoptic image coding system (JPEG Pleno) – Part 6: Learning-based point cloud coding, ISO/IEC DIS 21794-6. 2024. International Organization for Standardization and International Electrotechnical Commission.

[176] J. Ascenso in E. Upenik, White Paper on JPEG AI Scope and Framework v1. 0, *ISO/IEC JTC 1/SC 29/WG1 N90049*, 2021.

[177] Information technology – JPEG AI learning-based image coding system – Part 1: Core coding system, ISO/IEC DIS 6048-1. 2024. International Organization for Standardization and International Electrotechnical Commission.

[178] Information technologies – JPEG systems – Part 6: JPEG 360, ISO/IEC 19566-6. 2019. International Organization for Standardization and International Electrotechnical Commission.

[179] Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 12: Image File Format, ISO/IEC 23008-12. 2017. International Organization for Standardization and International Electrotechnical Commission.

[180] J. Han, B. Li, D. Mukherjee, C.-H. Chiang, A. Grange, C. Chen, H. Su, S. Parker, S. Deng, U. Joshi *et al.*, A technical overview of AV1, *Proceedings of the IEEE*, 109(9), 1435–1462, 2021.

[181] Q. Huynh-Thu in M. Ghanbari, Scope of validity of PSNR in image/video quality assessment, *Electronics Letters*, 44(13), 800–801, 2019.

[182] W. Zhou, A. C. Bovik, H. R. Sheikh in E. P. Simoncelli, Image Qualifty Assessment: From Error Visibility to Structural Similarity, *IEEE Transactions on Image Processing*, 13(4), 600–612, 2004.

[183] Z. Wang, E. P. Simoncelli in A. C. Bovik, Multiscale structural similarity for image quality assessment, *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers*, 2, 1398–1402, IEEE, 2003.

[184] H. R. Sheikh in A. C. Bovik, Image Information and Visual Quality, *IEEE Transactions on Image Processing*, 15(2), 430–444, 2006.

[185] A. Mittal, R. Soundararajan in A. C. Bovik, Making a Completely Blind Image Quality Analyzer, *IEEE Signal Processing Letters*, 22(3), 209–212, 2013.

[186] Y. Niu, Y. Zhong, W. Guo, Y. Shi in P. Chen, 2D and 3D Image Quality Assessment: A Survey of Metrics and Challenges, *IEEE Access*, 7, 782–801, 2019.

[187] A. C. Bovik in Z. Wang, *Modern Image Quality Assessment*. Morgan and Claypool, 2006.

[188] M. E. Celebi, Improving the performance of k-means for color quantization, *Image and Vision Computing*, 29(4), 260–271, 2011.

[189] D. Clark, The popularity algorithm, *DR. Dobb's Journal*, 20(7), 121, 1995.

[190] P. Heckbert, Color Image Quantization for Frame Buffer Display, *Computer Graphics*, 16(3), 297–303, 1982.

[191] M. Gervautz in W. Purgathofer, A simple method for color quantization: Octree quantization, *New Trends in Computer Graphics: Proceedings of CG International'88*, 219–231, Springer, 1988.

[192] C. Ozturk, E. Hancer in D. Karaboga, Color Image Quantization: A Short Review and an Application with Artificial Bee Colony Algorithm, *Informatica*, 25(3), 485–503, 2014.

[193] R. W. Floyd in L. Steinberg, An Adaptive Algorithm for Spatial Grey Scale, *International Symposium Digest of Technical Papers, Society for Information Displays*, 36–37, 1975.

[194] L. Akarun, Y. Yardimci in A. E. Cetin, Adaptive Methods for Dithering Color Images, *IEEE Transactions on image processing*, 6(7), 950–955, 1997.

[195] P. Stucki, Image Processing for Document Reproduction, *Advances in Digital Image Processing: Theory, Application, Implementation* (P. Stucki, ur.), 177–218, Springer, 1979.

[196] X. Luo in F. Lin, Method and system for image dithering, US9251732B2, 2016.

[197] Y.-P. Chang, K. Dai in J.-J. Huang, LCD device with image dithering function and related method of image dithering, US9697780B2, 2017.

[198] S. P. Aramendia, A. M. Barambio, L. Garcia in M. I. B. Bayona, Image processing using content–based weighted dithering, US9906686B2, 2018.

[199] M. Hussain, A. W. A. Wahab, Y. I. B. Idris, A. T. S. Ho in K.-H. Jung, Image steganography in spatial domain: A survey, *Signal Processing: Image Communication*, 65, 46–66, 2018.

[200] C.-K. Chan in L.-M. Cheng, Hiding data in images by simple LSB substitution, *Pattern recognition*, 37, 469–474, 2004.

[201] D.-C. Wu in W.-H. Tsai, A steganographic method for images by pixel-value differencing, *Pattern Recognition Letters*, 24, 1613–1626, 2003.

[202] V. M. Potdar in E. Chang, Grey Level Modification Steganography for Secret Communication, 2^{nd} *IEEE International Conference on Industrial Informatics INDIN2004*, 223–228, 2004.

[203] K. Muhammad, J. Ahmad, H. Farman, Z. Jan, M. Sajjad in S. Baik, A secure method for color image steganography using gray-level modification and multi-level encryption, *KSII Transactions on Internet and Information Systems*, 9, 1938–1962, 2015.

[204] Vector Image Files. https://fileinfo.com/filetypes/vector_image, zadnji dostop jun. 2024.

[205] Industrial automation systems and integration – Product data representation and exchange – Part 1: Overview and fundamental principles, ISO 10303-1. 1994. International Organization for Standardization.

[206] A. Quint, Scalable vector graphics, *IEEE Multimedia*, 10(3), 99–102, 2003.

[207] J. D. Eisenberg in A. Bellamy-Royds, *SVG Essentials*. O'Reilly Media, 2. izd., 2015.

[208] M. A. Abam, M. de Berg, P. Hachenberger in A. Zarie, Streaming algorithms for line simplification, *Discrete & Computational Geometry*, 43(3), 497–515, 2009.

[209] D. Douglas in T. Peuker, Algorithms for the reduction of the number of points required to represent a digitised line or its caricature, *The Canadian Cartographer*, 10, 112–122, 1973.

[210] J. E. Bresenham, Algorithm for computer control of a digital plotter, *IBM System Journal*, 4(1), 25–30, 1965.

[211] J. E. Bresenham, A linear algorithm for incremental digital display of digital arcs, *Communications of ACM*, 20(2), 100–106, 1977.

[212] Bresenham's Circle Drawing Algorithm. https://getsetcg.blogspot.com/2018/10/bresenhams-circle-drawing-algorithm.html, zadnji dostop 2019.

[213] G. Bao in J. G. Rokne, Quadruple-Step Line Generation, *Computers & Graphics*, 13(4), 461–469, 1989.

[214] G. W. Gill, N-step incremental Straight-line algorithms, *IEEE Computer Graphics and Applications*, 14(3), 66–72, 1994.

[215] B. K. P. Horn, Circle generation for display devices, *Computer Graphics and image processing*, 5, 280–288, 1976.

[216] J. R. van Aken, An Efficient Ellipse Drawing Algorithm, *IEEE Computer graphics & Applications*, 4(9), 24–35, 2019.

[217] M. L. V. Pitteway, Algorithm for Drawing Ellipses or Hyperbolae with a Digital Plotte, *Computer Journal*, 10(3), 282–289, 1967.

[218] H. D. Hobby, Rasterization of Nonparametric Curves, *ACM Transactions on Graphics*, 9(3), 262–277, 1990.

[219] Y. K. Liu, P. J. Wang, D. D. Zhao, D. Špelič, D. Mongus in B. Žalik, Pixel-level algorithms for drawing curves, *Theory and practice of computer graphics 2011 : Eurographics UK chapter proceedings* (I. Grimstead, ur.), 33–40, 2011.

[220] F. Hussain in M. L. V. Pitteway, Rasterizing the outlines of fonts, *Electronic publishing*, 6(3), 171–181, 1993.

[221] H. P. Moreton in F. C. Crow, Line rasterization techniques, US-8482567B1, 2013.

[222] M. H. Anderson, 2D/3D line rendering using 3D rasterization algorithms. US 7,362,325 B2, 2008.

[223] Y. P. Kuzmin, Ray Traversing of Spatial Structures, *Computer Graphics Forum*, 13(4), 223–227, 1994.

[224] J. C. Cleary in G. Wyvill, Analysis of an Algorithm for Fast Ray Tracing using Uniform Space Subdivision, *The Visual Computer*, 4(2), 65–83, 1988.

[225] B. Žalik, G. Clapworthy in Č. Oblonšek, An Efficient Code-Based Voxel Traversing Algorithm, *Computer Graphics Forum*, 16(2), 119–128, 1997.

[226] Y. K. Liu, B. Žalik in H. Yang, An Integer One-Pass Algorithm for Voxel Traversal, *Computer Graphics Forum*, 23(2), 167–172, 2004.

[227] X. Wu, An efficient antialiasing technique, *ACM Siggraph Computer Graphics*, 25(4), 143–152, 1991.

[228] X. Wu, Fast anti-aliased circle generation, *Graphics Gems II*, 446–450, Elsevier, 1991.

[229] L. Yang, S. Liu in M. Salvi, A survey of temporal antialiasing techniques, *Computer graphics forum*, 39(2), 607–621, 2020.

[230] C. Stephanidis in G. Salvendy, *Human-Computer Interaction: Foundations and Advances*. CRC Press, 2024.

[231] T. A. Edison, Improvement in phonograph or speaking machines, US200521A, 1877.

[232] E. Berliner, Gramophone, US372786A, 1887.

[233] Z. Li, M. Drew in J. Liu, *Fundamentals of Multimedia*. Texts in Computer Science, Springer International Publishing, 2021.

[234] D. A. Bies in C. H. Hansen, *Engineering noise control: theory and practice*. CRC press, 2003.

[235] H. Fletcher in W. A. Munson, Loudness, its definition, measurement and calculation, *Bell System Technical Journal*, 12(4), 377–430, 1933.

[236] Acoustics – Normal equal-loudness-level contours, ISO 226. 2003. International Organization for Standardization.

[237] Procedure for the Computation of Loudness of Steady Sounds (includes loudness program), ANSI/ASA S3.4-2007 (R2017). 2017. American National Standards Institute, Acoustical Society of America.

[238] Acoustical Terminology, ANSI/ASA S1.1-2013. 2013. American National Standards Institute, Acoustical Society of America.

[239] Specification for Audiometers, ANSI/ASA S3.6-2018. 2018. American National Standards Institute, Acoustical Society of America.

[240] Electroacoustics – Sound level meters – Part 1: Specifications, IEC 61672-1:2013. 2013. International Electrotechnical Commission.

[241] P. Buser in M. Imbert, *Audition*. MIT Press, 1992.

[242] H. Fletcher, Auditory patterns, *Reviews of modern physics*, 12(1), 47, 1940.

[243] D. Howard in J. Angus, *Acoustics and psychoacoustics*. Routledge, 2013.

[244] L. Elliott, Backward and forward masking, *Audiology*, 10(2), 65–76, 1971.

[245] S. Tomažič in S. Leonardis, *Diskretni signali in sistemi*. Založba FE, 1. izd., 2017.

[246] Pulse code modulation (PCM) of voice frequencies, G.711. 1988. International Telecommunication Union.

[247] IBM Corporation, Microsoft Corporation, Multimedia Programming Interface and Data Specifications 1.0. https://www.mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/riffmci.pdf, 1991, zadnji dostop feb. 2025.

[248] Apple Computer, Inc, Audio Interchange File Format: AIFF, A Standard for Sampled Sound Files, Version 1.3, 1991.

[249] Apple Computer, Inc, Audio Interchange File Format: AIFF-C, 1991.

[250] T. Robinson, SHORTEN: Simple lossless and near-lossless waveform compression, Teh. Por. CUED/F-INFENG/TR.156, Cambridge University, Engineering Department, 1994.

[251] S. W. Golomb, Run-length encodings, *IEEE Transactions on Information Theory*, IT–12(3), 399–401, 1966.

[252] M. van Beurden in A. Weaver, RFC 9639, Free Lossless Audio Codec (FLAC). https://www.rfc-editor.org/rfc/rfc9639.html, 2024, zadnji dostop feb. 2025.

[253] WavPack – Hybrid Lossless Audio Compression. http://www.wavpack.com/, zadnji dostop 2019.

[254] Y. F. Dehery, M. Lever in P. Urcun, A MUSICAM source codec for digital audio broadcasting and storage, *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, 3605–3606, IEEE Computer Society, 1991.

[255] Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio, ISO/IEC 11172-3:1993. 1993. International Organization for Standardization.

[256] Information technology – Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC), ISO/IEC 13818-7. 1997. International Organization for Standardization.

[257] P. Noll, MPEG Digital Audio Coding Standards, *The Digital Signal Processing Handbook* (V. Madisetti in D. B. Williams, ur.), IEEE Press/CRC Press, 1997.

[258] D. Pan, A tutorial on MPEG/audio compression, *IEEE Multimedia*, 2(2), 60–74, 1995.

[259] Information technology – Generic coding of moving pictures and associated audio information – Part 3: Audio, ISO/IEC 13818-3:1995. 1995. International Organization for Standardization.

[260] M. Nilsson, ID3 tag version 2.4.0 – Main Structure. https://id3.org/id3v2.4.0-structure, 2000, zadnji dostop feb. 2025.

[261] G.722 : 7 kHz audio-coding within 64 kbit/s, G.722. 1988. International Telecommunication Union.

[262] Z. Mezgec, *Indirektne segmentacijske metode za adaptivno robustno kontrolo prenosa podatkov po govornem kanalu sistema GSM: doktorska disertacija*. Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, 2009.

[263] G.722.1 : Low-complexity coding at 24 and 32 kbit/s for hands-free operation in systems with low frame loss, G.722.1. 1999. International Telecommunication Union.

[264] Digital cellular telecommunications system (Phase 2+); Full rate speech; Transcoding (GSM 06.10 version 5.1.1), ETS 300 961. 1998. European Telecommunications Standards Institute.

[265] Digital cellular telecommunications system (Phase 2+); Half rate speech; Half rate speech transcoding (GSM 06.20 version 5.1.1), ETS 300 969. 1998. European Telecommunications Standards Institute.

[266] G.722.2 : Wideband coding of speech at around 16 kbit/s using Adaptive Multi-Rate Wideband (AMR-WB), G.722.2. 2002. International Telecommunication Union.

[267] A. McCree, A scalable phonetic vocoder framework using joint predictive vector quantization of melp parameters, *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, IEEE, 2006.

[268] Information technology – Coding of audio-visual objects – Part 3: Audio, ISO/IEC 14496-3. 1999. International Organization for Standardization.

[269] S. Han in T. Fingscheidt, Robust MPEG-4 high-efficiency AAC with fixed-and variable-length soft-decision decoding, *Audio Engineering Society Convention 139*, Audio Engineering Society, 2015.

[270] M. Schnell, M. Schmidt, M. Jander, T. Albert, R. Geiger, V. Ruoppila, P. Ekstrand in G. Bernhard, MPEG-4 Enhanced Low Delay AAC-a new standard for high quality communication, *Audio Engineering Society Convention 125*, Audio Engineering Society, 2008.

[271] Xiph.Org Foundation, Vorbis I specification. https://xiph.org/vorbis/doc/Vorbis_I_spec.html, 2020, zadnji dostop feb. 2025.

[272] I. Siegert, A. F. Lotz, L. L. Duong in A. Wendemuth, Measuring the impact of audio compression on the spectral quality of speech data, *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2016*, 229–236, 2016.

[273] Xiph.Org Foundation, Ogg Documentation. https://xiph.org/vorbis/doc/Vorbis_I_spec.html, 2010, zadnji dostop feb. 2025.

[274] Microsoft, About the Windows Media Codecs. https://learn.microsoft.com/en-us/windows/win32/medfound/about-the-windows-media-codecs, 2021, zadnji dostop feb. 2025.

[275] FFmpeg, libavcodec/wma.c. https://ffmpeg.org/doxygen/0.6/wma_8c-source.html, 2007, zadnji dostop feb. 2025.

[276] C. C. Todd, G. A. Davidson, M. F. Davis, L. D. Fielder, B. Link in S. Vernon, AC-3: Flexible perceptual coding for audio transmission and storage, *Audio Eng. Soc. 96th Convention, 1994*, 1994.

[277] ATSC Standard: Digital Audio Compression (AC-3, E-AC-3), A/52:2012. 2012. Advanced Television Systems Committee.

[278] M. A. Gerzon, P. G. Craven, J. R. Stuart, M. J. Law in R. J. Wilson, The MLP lossless compression system, *Audio Engineering Society Conference: 17th International Conference: High-Quality Audio Coding*, Audio Engineering Society, 1999.

[279] J. R. Stuart, P. G. Craven, M. A. Gerzon, M. J. Law in R. J. Wilson, MLP lossless compression, *AES 9th Regional Convention (Tokyo, Japan)*, 1999.

[280] Digital Audio Compression (AC-4) Standard; Part 1: Channel based coding, ETSI TS 103 190-1 V1.3.1 (2018-02). 2018. European Telecommunications Standards Institute.

[281] Dolby, Dolby AC-4: Audio Delivery for Next-Generation Entertainment Services. https://web.archive.org/web/20151204095813/http://www.dolby.com/in/en/technologies/ac-4/Next-Generation-Entertainment-Services.pdf, 2015, zadnji dostop feb. 2025.

[282] J. M. Valin, K. Vos in T. B. Terriberry, Definition of the Opus Audio Codec. RFC 6716, https://www.rfc-editor.org/info/rfc6716, 2012.

[283] J. M. Valin, G. Maxwell, T. B. Terriberry in K. Vos, The opus codec, *135th AES International Convention. New York, USA*, 2013.

[284] Alliance for Open Media, Immersive Audio Model and Formats. https://aomediacodec.github.io/iamf/v1.1.0.html, 2024, zadnji dostop feb. 2025.

[285] Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 3: 3D audio, ISO/IEC 23008-3. 2015. International Organization for Standardization.

[286] BS.1387 : Method for objective measurements of perceived audio quality, Recommendation BS.1387. 2001. International Telecommunication Union.

[287] D. Mumtaz, V. Jakhetiya, K. Nathwani, B. N. Subudhi in S. C. Guntuku, Nonintrusive perceptual audio quality assessment for user-generated content using deep learning, *IEEE Transactions on Industrial Informatics*, 18(11), 7780–7789, 2021.

[288] R. Hanson, *Mass Communication: Living in a Media World*. SAGE Publications, 2018.

[289] J. Arnold, M. Frater in M. Pickering, *Digital television: technology and standards*. John Wiley & Sons, 2007.

[290] L. Li, Z. Yang, Y. Zhai, J. Yang in R. Wang, Improving multi-generation robustness of learned image compression, *2023 IEEE International Conference on Multimedia and Expo (ICME)*, 2525–2530, IEEE, 2023.

[291] M. El-Hajjar in L. Hanzo, A survey of digital television broadcast transmission techniques, *IEEE Communications surveys & tutorials*, 15(4), 1924–1949, 2013.

[292] H.222 : H.222.0 : Information technology – Generic coding of moving pictures and associated audio information: Systems, Recommendation H.222 (07/95). 1995. International Telecommunication Union (ITU).

[293] B. G. Haskell, A. Puri in A. N. Netravali, *Digital video: An introduction to MPEG–2*. Kluwer academic publishers, 2002.

[294] I. E. G. Richardson, *H.264 and MPEG-4 video compression*. John Wiley & Sons, 2003.

[295] BT.601 : Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios, ITU-R BT.601. 2011. International Telecommunication Union (ITU).

[296] BT.709 : Parameter values for the HDTV standards for production and international programme exchange, ITU-R BT.709-6. 2015. International Telecommunication Union (ITU).

[297] H.265 : High efficiency video coding, Recommendation H.265 (04/13). 2013. International Telecommunication Union (ITU).

[298] B. Li, J. Han, Y. Xu in K. Rose, Optical flow based co-located reference frame for video compression, *IEEE Transactions on Image Processing*, 29, 8303–8315, 2020.

[299] C. Je in H.-M. Park, Optimized hierarchical block matching for fast and accurate image registration, *Signal Processing: Image Communication*, 28(7), 779–791, 2013.

[300] A. K. Mishra in N. Kohli, Analysis of block matching algorithms for motion estimation in video data, *Machine Learning and Information Processing: Proceedings of ICMLIP 2020*, 331–341, Springer, 2021.

[301] Microsoft Corporation, AVI File Format. https://learn.microsoft.com/en-us/windows/win32/directshow/avi-riff-file-reference, 2023, zadnji dostop mar. 2025.

[302] T. Gloe, A. Fischer in M. Kirchner, Forensic analysis of video file formats, *Digital Investigation*, 11, 68–76, 2014.

[303] M. Jacobs in J. Probell, A Brief History of Video Coding. https://pdfs.semanticscholar.org/6aa0/2f12913215e76cd7e830e371109b24a3daa6.pdf, zadnji dostop 2019.

[304] A. Davis, An Overview of Video Compression Algorithms. https://www.edn.com/design/communications-design/4018012/An-Overview-of-Video-Compression-Algorithms, zadnji dostop 2019.

[305] I. E. Richardson, *Coding video: A practical guide to HEVC and beyond*. John Wiley & Sons, 2024.

[306] H.120 : Codecs for videoconferencing using primary digital group transmission, Recommendation H.120 (03/93). 1993. International Telecommunication Union (ITU).

[307] P. Stewart, S. McCanne, B. Fenner, L. Berc in R. Frederick, RTP Payload Format for JPEG-compressed Video. RFC 2435, 1998.

[308] Information technology – JPEG 2000 image coding system: Motion JPEG 2000, ISO/IEC 15444-3. 2007. International Organization for Standardization and International Electrotechnical Commission.

[309] H.261 : Video codec for audiovisual services at p x 384 kbit/s, Recommendation H.261 (11/88). 1988. International Telecommunication Union (ITU).

[310] Recording – Helical-scan digital video cassette recording system using 6,35 mm magnetic tape for consumer use (525-60, 625-50, 1125-60 and 1250-50 systems) – Part 1: General specifications, IEC 61834-1. 1998. International Electrotechnical Commission.

[311] H. Kato, Y. Takishima in Y. Nakajima, A fast DV to MPEG-4 transcoder integrated with resolution conversion and quantization, *IEEE transactions on circuits and systems for video technology*, 17(1), 111–119, 2006.

[312] Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 2: Video, ISO/IEC 11172-2:1993. 1993. International Organization for Standardization.

[313] C. W. Brown in B. J. Shepherd, *Graphics File Formats: reference and guide*. Manning Publications Co., 1995.

[314] Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 1: Systems, ISO/IEC 11172-1:1993. 1993. International Organization for Standardization.

[315] Information technology – Generic coding of moving pictures and associated audio information – Part 2: Video, ISO/IEC 13818-2:1996. 1996. International Organization for Standardization.

[316] Information technology – Generic coding of moving pictures and associated audio information – Part 1: Systems, ISO/IEC 13818-1:1996. 1996. International Organization for Standardization.

[317] H.263 : Video coding for low bit rate communication, Recommendation H.263 (03/96). 1996. International Telecommunication Union (ITU).

[318] H.263 : Video coding for low bit rate communication, Recommendation H.263 (02/98). 1998. International Telecommunication Union (ITU).

[319] Information technology – Coding of audio-visual objects – Part 2: Visual, ISO/IEC 14496-2. 1999. International Organization for Standardization and International Electrotechnical Commission.

[320] C. Wootton, *A Practical Guide to Video and Audio Compression: From Sprockets and Rasters to Macro Blocks*. Taylor & Francis, 2005.

[321] Information technology – Coding of audio-visual objects – Part 10: Advanced video coding, ISO/IEC 14496-10. 2003. International Organization for Standardization and International Electrotechnical Commission.

[322] H.264 : Advanced video coding for generic audiovisual services, Recommendation H.264 (05/03). 2003. International Telecommunication Union (ITU).

[323] N. Özbek in T. Tunali, A Survey on the H.264/AVC Standard, *Turk Journal of Electrical Engineering*, 13(3), 287–302, 2005.

[324] I. Richardson, *The H.264 Advanced Video Compression Standard: Second edition*. Wiley, 2010.

[325] Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 2: High efficiency video coding, ISO/IEC 23008-2. 2013. International Organization for Standardization and International Electrotechnical Commission.

[326] H.266: Versatile video coding, Recommendation H.266 (09/20). 2020. International Telecommunication Union (ITU).

[327] Information technology – Coded representation of immersive media – Part 3: Versatile video coding, ISO/IEC 23090-3. 2021. International Organization for Standardization.

[328] M. Lee, H. Song, J. Park, B. Jeon, J. Kang, J.-G. Kim, Y.-L. Lee, J.-W. Kang in D. Sim, Overview of versatile video coding (H. 266/VVC) and its coding performance analysis, *IEIE Transactions on Smart Processing & Computing*, 12(2), 122–154, 2023.

[329] H. S. Spilker in S. Hoier, Technologies of piracy? Exploring the interplay between commercialism and idealism in the development of MP3 and DivX, *International Journal of Communication*, 7, 20, 2013.

[330] X. Foundation, Theora Specification. http://www.theora.org/doc/Theora.pdf, 2017, zadnji dostop apr. 2025.

[331] Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process, SMPTE 421M-2006. 2006. Society of Motion Picture and Television Engineers (SMPTE).

[332] S. Ma, T. Huang in W. Gao, The second generation IEEE 1857 video coding standard, *2015 IEEE China Summit and International Conference on Signal and Information Processing (ChinaSIP)*, 171–175, IEEE, 2015.

[333] S. Ma, L. Zhang, S. Wang, C. Jia, S. Wang, T. Huang, F. Wu in W. Gao, Evolution of AVS video coding standards: twenty years of innovation and development, *Science China Information Sciences*, 65(9), 192101, 2022.

[334] Information technology – General video coding – Part 1: Essential video coding, ISO/IEC 23094-1. 2020. International Organization for Standardization.

Stvarno kazalo

DOI
https://doi.org/10.18690/um.feri.8.2025

ISBN
978-961-299-041-1

Ključne besede:
multimedija,
računalniška večpredstavnost,
datotečni formati,
avdio,
video,
slika,
tekst

Uvod v računalniško večpredstavnost in obdelavo multimedijskih podatkov

Štefan Kohek, Borut Žalik
Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
stefan.kohek@um.si, borut.zalik@um.si

Učbenik opisuje osnovne vidike multimedije, multimedijskih tipov, njihovih formatov in operacij nad njimi. Ponuja postopen in sistematičen pregled temeljnih konceptov obdelave multimedijskih podatkov. Poseben poudarek je na predstavitvi osnovnih tipov multimedijskih podatkov, izbranih algoritmov za njihovo obdelavo in najpogostejših multimedijskih formatov. Struktura učbenika sledi osnovnim tipom multimedijskih podatkov, in sicer tekstu, slikam, digitalnemu avdiu in videu. Vsako poglavje se zaključi z naborom vprašanj in praktičnih nalog za poglobljeno razumevanje obravnavane snovi.

Univerzitetna založba Univerze v Mariboru

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko