1 Suono

Forma di energia, sulla terra vibrazione = suono.
Velocità 0° C: $340.27\,[m/s]$, 20° C: $343\,[m/s]$ (mach 1).
Nell'acqua, il suono viaggia 4.3 volte più veloce che nell'aria. Nel vuoto non c'è suono. Oggetti a velocità superiore a mach 1 viaggiano a velocità supersonica.

1.1 Energia Sonora

Joule [J], il mezzo di propag. (aria, acqua, ...) funge da accumulatore di energia, sia potenziale che cinetica.

$$W = W_{\text{potential}} + W_{\text{kinetic}} = \int_V \frac{p^2}{2\rho_0 c^2}\,\mathrm{d}V + \int_V \frac{\rho v^2}{2}\,\mathrm{d}V$$

dove:

V	è il volume d'interesse
p	è la pressione sonora
v	è la velocità delle particelle
ρ_0	è la densità del mezzo in assenza di suono
ρ	è la densità locale del mezzo
c	è la velocità del suono

1.2 Pressione Sonora

Detta anche pressione acustica, è rappr. dalla variazione di pressione esercitata dal suono rispetto alla pressione atmosferica ambientale. Unità di misura: Pascal [Pa]

$$p_{\text{total}} = p_{\text{stat}} + p$$

p	è la pressione sonora
p_{total}	è la pressione totale
p_{stat}	è la pressione ambientale

1.3 Pressione Sonora Effettiva (RMS)

Valore medio di pressione istantanea calcolata su un intervallo di tempo definito. La media RMS viene anche detta media quadratica.

1.4 Livello di pressione sonora (SPL)

Valore relativo calcolato come il logaritmo del rapporto fra una data pressione sonora e una pressione sonora di riferimento. Nell'aria si usa 20 [μPa], 1 [μPa] per altri mezzi. Si tratta del limite di capacità d'udito dell'essere umano.

1.5 dB SPL

Il livello di pressione sonora si misura in dB SPL:

$$L_p = 20\log_{10}\left(\frac{p_{\text{rms}}}{p_{\text{ref}}}\right)\text{dB}$$

Una pressione RMS di 1 [Pa] corrisponde ad un livello di 94 [dB SPL].

1.6 Potenza sonora (L_w)

Energia Acustica emessa da una sorgente. Valore assoluto. Indipendente dalla distanza e dalle dim. della stanza. È la potenza totale emessa dalla sorgente in tutte le direzioni. Misurato in dB SWL. Viene utilizzato p.es per misurare la quantità di rumore prodotto dagli utensili di lavoro.

1.7 Psicoacustica

Scienza che studia la percezione del suono e la risposta psicologica prodotta. Usata nella localizzazione delle sorgenti sonore, riconoscimento dei timbri e negli effetti di mascheramento. Possiamo ingannare l'orecchio approfittando degli effetti psicoacustici. E.g: stereofonia / compressione psicoacustica dell'MP3.

1.8 Suoni

Suoni periodici vs aperiodici.
Ampiezza determina l'intensità del suono.
Lunghezza d'onda e **frequenza**: per i suoni periodici determinano l'altezza (pitch) del suono.
La frequenza è proporzionale alla lunghezza d'onda. Per una maggiore lunghezza d'onda si avrà una minore frequenza.

1.8.1 Periodo e Frequenza

La lunghezza d'onda di un suono ne determina il periodo: è il tempo necessario per completare un ciclo di vibrazione. Più il periodo è lungo, più la frequenza (e di conseguenza la nota) sarà bassa.
Periodo T misurato in secondi.

$$F = \frac{1}{T}\ [\text{Hz}]$$

L'uomo sente da 20 Hz a 20 kHz, con percezione migliore fra i 3 e 4 kHz.

1.8.2 A-weighting (dBA)

Per adattare le misure di livello sonoro alla differente percezione che l'essere umano ha delle frequenze vengono usate delle pesature. La più comune è l' A-weighting utilizzata nel dBA.

1.8.3 Fase

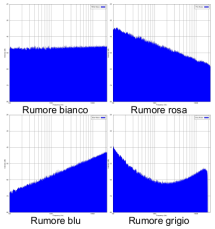
Varia a dipendenza dell'istante in cui l'onda comincia la propria oscillazione. Solitamente inaudibile, ha però un forte effetto quando vengono sommate.

1.8.4 Timbro

Anche chiamato colore del suono. Determina il carattere del suono. Strumenti diversi producono suoni di timbro diverso. Timbro più semplice: sintetizzatore di sinusoide, timbro più complesso: rumore bianco.
La percezione del timbro è determinata da due caratteristiche fisiche: lo spettro e l'inviluppo.

1.8.5 Spettro

Ogni suono può essere modellato come combinazione di onde sonore sinusoidali a differenti frequenze. Se un suono contiene freq. che sono multipli di una freq. fondamentale, l'unione di questi suoni viene percepita come un unico suono la cui frequenza è quella fondamentale e il cui carattere è det. dalle freq. e ampiezze coinvolte.
Per un tono puro (una sinusoide), il grafico è formato da una semplice linea verticale, per un suono musicale da una serie di linee in corrispondenza alle freq. fondamentali ed alle armoniche. Per un rumore lo spettro è costituito da una banda di frequenza.



1.8.6 Inviluppo

Andamento dell'ampiezza nel tempo. Ogni timbro ha un suo particolare inviluppo.
L'inviluppo di un suono è solitamente composto da 4 fasi, dal momento in cui viene originato, al momento in cui si estingue: attack, decay, sustain, release.

1.8.7 Spettrogramma

Lo spettrogramma è la rappresentazione grafica dell'intensità di un suono in funzione sia del tempo, che della frequenza. È lo strumento migliore per osservare il timbro di un suono. Ne esistono due varianti: bidimensionale e tridimensionale.

1.8.8 Direzione e distanza dal suono

Il sistema uditivo degli esseri umani è capace di distinguere la direzione di provenienza e di stimare la distanza della sorgente di un suono. Tendenzialmente la direzione viene identificata soprattutto sfruttando la differenza tra l'istante di arrivo del suono ad un orecchio, rispetto all'altro. Mentre per la stima della distanza, vengono soprattutto sfruttati i riverberi (prodotti dall'ambiente), in particolare l'informazione di predelay (il tempo che intercorre fra l'arrivo del suono diretto e le prime riflessioni).
Head related transfer function (HRTF) Quando il suono si scontra con l'ascoltatore, le caratteristiche fisiche dei ricettori (forma della testa, orecchie, etc.) modificano il suono e quindi la percezione ottenuta.

1.8.9 Stereofonia

L'ascolto stereofonico è un inganno psicoacustico in cui vengono utilizzate due sorgenti sonore (correttamente posizionate), con una tecnica di riproduzione in cui i medesimi contenuti sonori vengono riprodotti con differenti ampiezze. Quest'inganno permette di posizionare sorgenti sonore virtuali nello spazio compreso fra i due altoparlanti.

2 Audio Analogico vs Audio Digitale

Analog to Digital Converter (ADC) Un sistema che converte un segnale analogico in un segnale digitale.
Digital to Analog Converter (DAC) Un sistema che converte un segnale digitale in un segnale analogico.

2.1 dBV e dBu

Unità di misura usate per intensità di voltaggio. Sono valori specifici a valori di riferimento.
dbV: voltaggio RMS relativo a 1 [V]
dbu: voltaggio RMS relativo a $\sqrt{0.6}V \approx 0.7746V \approx -2.218dBV$

2.2 Range dinamico

Rapporto fra ampiezza e onda sinusoidale più forte possibile (non distorta) e rumore di fondo (rumore RMS) del sistema in uso.
Spesso confuso con SNR, che è la differenza fra livello medio del segnale e livello medio del rumore di fondo. Il range dinamico dell'orecchio umano è attorno ai 140 dB. Dai 120 dB a 140 c'è la soglia del dolore.

2.3 Livelli ed Headroom

Livelli di voltaggio negli standards sono detti livelli nominali. L'headroom è la differenza fra il livello nominale e il livello massimo, oltre il quale il segnale viene compromesso dalla distorsione.

2.4 VU meters

Usati per rappresentare livelli di intensità sonora in apparecchi come mixer. Unità di misura: Volume Unit (VU). In sit. normali, 0VU permette un headroom di almeno 18dB senza distorsione significativi. Solitamente negli apparecchi professionali 0 VU = +4 dBu, mentre in quelli consumer 0 VU = -10 dBV

2.5 Range dinamico di segnali digitali

Dettato dall'errore di quantizzazione.

bits	8	12	16	18	20	24
dB	48	72	96	108	120	144

Qualità CD: 96 dB. Esistono algoritmi di noise shaping (dithering) capaci di spostare la perc. del range din. a 120 dB. Per il processing si usano 32 bits, float.

2.6 Pulse Code Modulation (PMC)

La tecnica di conversione più utilizzata è quella di PCM: ad essere campionata a intervalli regolari è l'ampiezza del segnale analogico. Ogni campione viene quantizzato al valore più vicino all'interno del range dei possibili valori digitali.
Esistono quindi due caratteristiche principali di un segnale audio: bit depth e sample rate.

2.6.1 Bit Depth

Quantità di bits utilizzati, influenza la qualità sonora in termini di dinamica massima acquisibile.

2.6.2 Sample Rate

Frequenza di campionatura. Con 44.1 kHz è possibile campionare freq. fino ad un massimo di 22050 Hz.

2.6.3 Nyquist-Shannon

$$F_s = 2 \cdot F_{max}$$

2.7 Decibels Full-Scale (dbFS)

Per misurare l'ampiezza dei segnali digitali viene utilizzata l'unità di misura dBFS. È un'unità di misura che mette in relazione l'ampiezza del segnale con l'ampiezza del picco massimo permesso da quel sistema (detto il full-scale). Oltre alla soglia del full-scale interviene il clipping, una forma di distorsione molto aggressiva. Al full-scale è assegnato il valore di 0 dBFS, quindi tutti i valori più piccoli del massimo sono negativi.

2.8 Aliasing

Capita quando si campionano frequenze oltre alla frequenza di Nyquist. Le freq. oltre alla freq. massima vengono riflesse all'interno dello spettro generando freq. inesistenti.

2.9 Jittering

Il sampling jitter è una lieve variazione nella precisione dei clock dei convertitori ADC o DAC. Questa lieve alterazione può modificare lo spettro delle fasi del segnale rovinandolo.

2.10 Total Harmonic Distortion (THD)

Valore RMS degli armonici prodotti dai convertitori relativi al livello RMS di un segnale di input sinusoidale vicino a full-scale.

2.11 Formati audio per files

- Formati non compressi (Dati PCM/RAW, .wav, .aiff, .au)
- Formati con compressione lossless (Simil-zip, .flac, .ape, .wv)
- Formati con compressione lossy (.mp3, .aac, .opus)

PCM, 16 bits, 44.1 kHz, stereo: $16 \cdot 44100 \cdot 2 = 1411200$ bps
MP3 256, comprime al massimo 256 kbps

3 Tecniche base di audio processing

3.1 Nozioni

Risposta impulsiva La risposta impulsiva di un sistema è una funzione (del tempo) che coincide con l'output del sistema se sollecitato con un impulso.
Risposta in frequenza La risposta in frequenza è una misura quantitativa dello spettro di output di un sistema, utilizzata per caratterizzare la dinamica del sistema.
Risposta in fase In maniera simile alla risposta in frequenza è possibile rappresentare la risposta in fase: mette in relazione le differenze di fase fra il segnale di input e quello di output di un sistema.

3.2 Trasformata di Fourier

Scompone una funzione del tempo (quindi un segnale) nelle frequenze che la compongono. Risulta una funzione di numeri complessi, il cui valore assoluto rappresenta l'ampiezza di ogni frequenza del segnale, mentre l'argomento complesso è la differenza di fase dalla sinusoide di quella determinata frequenza.
Permette quindi di passare dal dominio del tempo al dominio delle frequenze. La sua inversa si chiama sintesi di Fourier.

3.3 Filtri: Sistemi LTI (Linear Time Invariant System)

Time-invariant Sistema che dato un input in diversi istanti di tempo, restituisce lo stesso output.

Nella realtà, molti sistemi possono però essere modellati per semplicità come sistemi LTI, trascurando le eventuali non-linearità. I sistemi LTI sono importanti perché possono essere risolti con metodologie di signal processing. E' possibile realizzare filtri con l'FFT, alternative più efficaci sono i filtri FIR e IIR.

3.3.1 Filtro FIR

Un filtro con finite impulse response (FIR) è un filtro la cui risposta impulsiva (o risposta a qualsiasi input di lunghezza finita) è di durata finita, perché l'output va automaticamente a 0 in un tempo finito. La risposta impulsiva di un FIR di ordine N dura esattamente $N + 1$ campioni.

3.3.2 Filtri IIR

Filtri con risposta impulsiva infinita. A differenza di FIR hanno un feedback, per questo motivo la risposta in freq. dei filtri IIR è migliore di quella dei filtri FIR dello stesso ordine. IIR sono più efficienti e possono essere facilmente parametrizzati in tempo reale. La risposta in fase non è lineare. Se non vengono disegnati correttamente possono diventare instabili a causa dell'anello di feedback.

4 Nozioni base sull'immagine

4.1 Immagine

Una rappresentazione della percezione visiva umana di un soggetto o di un ambiente. Può essere bidimensionale (foto) o tridimensionale (ologramma). Le immagini catturano la luce e possono essere generate con dispositivi ottici o dagli umani tramite l'occhio.

4.2 Luce

Radiazione elettromagnetica (EMR) in una certa porzione dello spettro elettromagnetico.
EMR Classificata per lunghezza d'onda in radio, micro onde, infrarosso, luce visibile, ultravioletto, raggi X e raggi gamma .
Lunghezza d'onda 400 - 700 nm (fra 430 e i 750 THz).
Energia radiante [J]
Flusso radiante [W]
Radianza Watt allo steradiante per metro quadrato.
Luce visibile Viene emessa e assorbita in piccoli pacchetti chiamati fotoni, che esibiscono proprietà sia delle onde che delle particelle, questa caratteristica è la dualità onda-particella.
Ottica Studio della luce e della sua interazione con la materia (fenomini ottici come arcobaleno ecc.)

4.2.1 Grandezze per misurare la luce

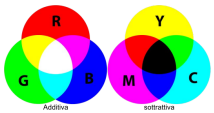
Radiometriche Relative alla radiazione elettromagnetica, la misurazione è compito della radiometria. Sono: Energia radiante [J], Flusso radiante [W] e Radianza [W allo steradiante per m^2].
Fotometriche Quantificano l'emissione luminosa partendo dalle grandezze radiometriche ma mediante pesatura con la curva di risposta spettrale dell'occhio umano. Sono: Flusso luminoso [Lumen], Intensità luminosa [Canela], Illuminamento [Lux].

4.3 Colori

I colori vengono generati dagli oggetti in funzione della loro capacità di riflettere le varie lunghezze d'onda presenti nella luce visibile. L'occhio umano può distinguere fino a 10M di colori.

4.3.1 Modelli

Adattiva somma dei colori derivati dai primari → bianco (somma colori modello RGB).
Sottrattiva somma dei primari (partendo dai secondari) → nero (somma colori modello CMY).



4.3.2 Parametri

Visione Binoculare Anche detta stereoscopica o stereopsi, è una caratteristica del sistema visivo. Avviene in un campo ristretto rispetto alla normale visione bioculare (pesci); in questa entrambi gli occhi partecipano alla visione dello stesso soggetto o della stessa area di campo. La parte del singolo campo visivo monoculare dell'occhio destro, va a sommarsi alla stessa parte visibile dell'occhio sinistro, formando appunto una immagine binoculare. L'acutezza visiva è maggiore rispetto alla visione monoculare. Il campo visivo è di circa 95° orizzontali e 80° verticali.
Disparità Binoculare Gli occhi umani sono separati orizzontalmente da una distanza interpupillare di circa 50–75 mm quindi ogni occhio ha una visione leggermente diversa del mondo circostante. La disparità binoculare è la differenza di posizione fra i punti di proiezione, espressa in gradi (angolo visivo).
Hue Gradazione o varietà di colore
Colorfulness, Saturation e Chroma (da non confondere con Video Chroma) Rappresentano l'intensità cromatica
Brightness e Lightness Rappresentano la percezione di intensità luminosa.
Contrast rappresenta la differenza in intensità luminosa o colore dell'immagine.

4.3.3 Istogramma dei colori

Rappresentazione della distribuzione dei colori. Se l'immagine è monocromatica → istogramma di intensità.

4.4 Parallasse

Differenza della posizione apparente di un oggetto visto da due differenti punti di vista.

5 Immagine digitale

5.1 Vettoriale vs Raster

Vettoriali indipendenti dalla risoluzione, sono però più laboriose da produrre.
Raster Non possono essere adattate alle risoluzioni senza perdita di qualità.

5.2 Sensori CCD o CMOS

Charge-Coupled Devices (CCD) e Complementary Metal-Oxide-Semiconductor (CMOS). Un CCD ha un unico amplificatore per ogni pixel. La maggior parte delle camere digitali ha sensori CMOS perché offrono prestazioni migliori, sono più veloci e consumano meno. I CCD vengono ancora utilizzati per fotocamere digitali a basso costo.

5.3 Campionatura e Quantizzazione

Strumenti come i frame grabber permettono di discretizzare le immagini. L'immagine deve essere discretizzata sia spazialmente (sampling) sia in termini di intensità di luce (quantization).
I fattori che influenzano la digitalizzazione sono:

- Sampling rate: determina la risoluzione spaziale
- Livello di quantizzazione: determina la tonalità di grigio o livelli di colore che vengono acquisiti.

5.4 Risoluzione Spaziale

Misura del più piccolo dettaglio discernibile in un'immagine digitale. Per misurarla si usano i DPI (dots per inches). Affermare che un'immagine ha una risoluzione di MxN pixels non ha un effettivo significato se la risoluzione non è messa in relazione ad un'unità di misura spaziale.

5.5 Risoluzione di intensità

Più piccolo dettaglio discernibile nel livello di intensità di grigio o colore. Per le immagini grayscale vengono d'abitudine usati 8 bits (256 valori), mentre per le immagini a colori si usano 16 bits. L'utilizzo di 32 bits è raro.

5.6 Colori nelle Schede Video

Le schede video attuali dedicano 24 bits per pixel, mostrando quindi $2^{24} = 16M$ di colori. 24 bit (16M di colori) è chiamato True Color.

5.7 Aliasing

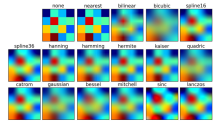
L'introduzione di artefatti durante la ricostruzione di un segnale che, prima di venir campionato, conteneva oggetti più piccoli della metà del sampling rate (Nyquist). Nelle immagini la freq. è in relaz. alla dimensione strutturale. Parti piccole hanno freq. alta. Nelle immagini l'aliasing è spaziale.

5.8 Dither

Forma di rumore introdotta intenzionalmente per rendere casuale l'errore di quantizzazione.

5.9 Interpolazione

L'interpolazione è un'operazione spesso utilizzata per le immagini per eseguire zoom, rotazioni, e altre correzioni geometriche. Viene eseguita utilizzando informazioni conosciute in determinate posizioni per stimare i valori di posizioni sconosciute.



5.10 Relazioni fra pixels

Vicinanza ogni pixel ha 2 vicini verticali, 2 vicini orizzontali e 4 vicini diagonali.
Adiacenza due pixels sono adiacenti se sono vicini e rispettano determinati vincoli di prossimità di grigio o colore.

5.11 Distanza fra pixel

Euclidian Distance, City-Block distance, Chess-Board distance.

5.12 Formati file per immagini digitali

Raster: .gif, .jpg, .tiff, .png, .bmp

5.12.1 GIF

Crea una tabella fino a 256 colori, da un pool di 16M. Se l'immagine ha meno di 256 colori l'algoritmo è capace di riprodurre l'immagine in maniera esatta. Per i casi in cui l'immagine contiene più di 256 colori, alcuni algoritm. usano il colore più vicino mentre altri l'"error diffusion" per minimizzare l'errore. GIF sostituisce eventuali patterns (ampie zone di colore uniforme) con delle abbreviazioni (come bianco · 20). Di conseguenza GIF è lossless solo per immagini con 256 colori o meno. Per immagini True Color può perdere il 99.998% dei colori.

5.12.2 JPG

Formato ottimizzato per la fotografia ed immagini, contenenti colori simili in zone contigue. Lavora scartando informazione che l'occhio umano non dovrebbe notare. Salva l'informazione come colore a 24 bit. Compressione lossy, livello di compr. adattabile.

5.12.3 TIFF, PNG, BMP

TIFF è un formato flessibile che può essere sia lossless che lossy. Viene principalmente usato senza nessun tipo di compressione. I dettagli dell'algoritmo utilizzato vengono salvati sul file.
PNG è lossless. Riduce la taglia del file cercando patterns che può sfruttare per il salvataggio dell'informazione.
BMP formato proprietario non compresso inventato da Microsoft.

5.13 Header e Container

Nelle immagini i file sono composti da una prima parte di meta informazione, contenente le caratteristiche dell'informazione salvata, seguita dagli effettivi dati dell'immagine.

5.14 Operazioni Array vs Matriciali

Le immagini raster possono essere viste come matrici. Su di esse è possibile eseguire operazioni di tipo array, sia matriciali. Ad es: il prodotto di due matrici può essere inteso come prodotto array (ogni elemento delle due matrici viene moltiplicato), oppure come l'effettivo prodotto matriciale.
Di norma, nel digital signal processing si intendono le operazioni come operazioni di tipo array.

5.15 Operazioni lineari vs. Non lineari

Per un filtro lineare (ad esempio la media) si ha che:

$$F_m(A + \lambda B) = F_m(A) + \lambda F_m(B)$$

Questa proprietà non viene invece soddisfatta dai filtri non lineari, come ad esempio la mediana.

5.16 Linear Image Processing

Le op. di linear image processing sono basate, come per il signal processing, su due op. fondamentali: la convoluzione e la trasformata di Fourier.
Nell'image processing la conv. è l'op. più importante perché le immagini hanno l'informazione codificata nel dominio spaziale anziché quello delle frequenze.
Filtri lineari possono ad es. venir utilizzati per: accentuare gli spigoli degli oggetti, ridurre il rumore di fondo. correggere l'illuminazione, ecc... Queste operazioni vengono eseguite per convoluzione tra l'immagine originale e un kernel di filtro, producendo l'immagine filtrata.

5.17 Analisi di Fourier

Meno utile per le immagini. Converte l'informazione chiara del dom. spaziale in una forma poco intellegibile nel dominio delle frequenze.

6 Tecniche base di image processing

6.1 Digital Image Processing

6.1.1 Equalizzazione dell'istogramma

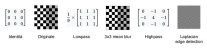
È un metodo dell'image processing per l'adeguamento del contrasto. L'obiettivo è di ridistribuire le intensità nell'istogramma, fornendo, come conseguenza, maggiore contrasto a zone che hanno basso contrasto locale.
Normalizzazione dell'immagine È un processo simile all'equalizzazione dell'istogramma, in cui vengono modificate le intensità dei pixels (estendendone il range) per ottenere una migliore occupazione del range dinamico disponibile.

6.1.2 Blur

Filtri digitali tramite convoluzione o DFT Per blur e sharpening si utilizzano tecniche di convoluzione o di DFT. Il filtraggio può essere eseguito nel dominio spaziale tramite convoluzione, applicando dei kernels, o nel dominio delle frequenze, tramite la trasformata di Fourier (DFT).
Gaussian blur Una sfocatura gaussiana si ottiene applicando all'immagine una funzione gaussiana tramite convoluzione, facendo ciò si riduce il rumore ma anche la presenza di dettagli nelle immagini.
Box blur Un box blur è un filtro passa basso nel dominio spaziale, in cui ogni pixel è la media dei suoi vicini nell'immagine di partenza. Sono specificati tramite una matrice 3X3.
Sharpening Un'immagine è a fuoco se ha buona risoluzione e un buon contrasto sui contorni (edges). Lo sharpening permetti di migliorare/peggiorare questo contrasto.
Convoluzione Nel trattamento d'immagine, la convoluzione è il processo di aggiungere ogni elemento ai propri vicini, moltiplicato per i valori specificati in un kernel di dimensione 3x3.

6.1.3 Kernel

I kernel usati nella convoluzione vengono anche detti matrici di convoluzione o maschere. Possono essere utilizzati per implementare una moltitudine di filtri, incluse alcune soluzioni per la detezione dei contorni.



Normalizzazione del kernel La normalizzazione dei valori di un kernel è definita come la divisione di ogni elemento del kernel per la somma di tutti gli elementi del kernel, in modo che la somma di tutti i valori all'interno del kernel dia 1. Questa soluzione garantisce che che i pixels siano in media luminosi nell'immagine risultante quanto lo sono nell'immagine originale.

6.2 Detezione dei contorni

Il riconoscimento dei contorni (Edge detection) ha lo scopo di marcare i punti in cui l'intensità luminosa cambia bruscamente. L'obbiettivo di questa tecnica è quella di rilevare i contorni quindi qualsiasi dettaglio non utile alla definizione di una forma o caratteristiche strutturali viene scartato.

6.3 DFT 2D

F è il risultato (lo spettro) della trasformata di Fourier dell'immagine f. L'operazione inversa permette di ricostruire l'immagine a partire dallo spettro. Il fatto che la modellazione eseguita dalla DFT sfrutti segnali periodici può presentare delle complicazioni nel caso delle immagini, visto che non presentano necessariamente periodicità. In maniera simile alla convoluzione, La DFT 2D viene utilizzata per applicare varie tipologie di filtri.

6.4 Denoise

Immagini acquisite con camere digitali o convertite da film fotografico convenzionale, contengono rumore introdotto da una moltitudine di fattori per es. quelli dati dai circuiti elettrici. Molto frequentemente il processing di queste immagini richiede la rimozione (almeno parziale) di questo rumore, questo processo si chiama denoise.

6.5 Rumore salt-and-pepper

Il rumore di tipo salt-and-pepper, anche conosciuto come rumore impulsivo, causa disturbi improvvisi e netti all'interno dell'immagine. Si presenta di norma come occorrenze sparse di pixels neri o bianchi. Per risolverlo → filtro mediano.

6.6 Rumore gaussiano

Il rumore gaussiano ha una distribuzione statistica di tipo normale.

6.7 Restoration: Wiener filter

I filtri più tradizionali eseguono la separazione tra segnale e rumore a condizione che questi occupino diverse bande di frequenza. Il filtro di Wiener supera questa limitazione con un approccio statistico. Si assume di avere conoscenza delle varie caratteristiche e si cerca il filtro LTI il cui risultato sia "il più vicina possibile" al segnale originale. Questo filtro è spesso utilizzato nel processo di deconvoluzione.

6.8 Segmentazione

La segmentazione è il processo in cui si suddivide un'immagine digitale in più segmenti (sets di pixel). L'obiettivo è quello di semplificare/modificare la rappresentazione dell'immagine in qualcosa che sia più facile e sensato da analizzare.

6.9 Thresholding

Il metodo più semplice per eseguire una segmentazione è chiamato metodo di sogliatura (Thresholding). Nella sua versione più essenziale, un'immagine in scale di grigi viene convertita in un'immagine con pixels solo bianchi o neri. Il valore usato come discriminante è la soglia (Threshold).

6.9.1 Basato sull'istogramma

I metodi di thresholding che sfruttano l'istogramma, sono normalmente efficaci rispetto agli altri metodi, perché necessitano di un solo passaggio di processing sui pixels. Picchi e valli dell'istogramma vengono utilizzati per identificare i clusters all'interno dell'immagine.

6.10 K-means

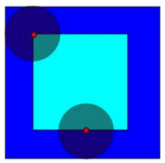
Abbiamo appena fatto Data Science quindi sappiamo tutti come funziona. Si usa per partizionare l'immagine in clusters.

6.10.1 Morphological matematica nell'Image Processing

La morfologia matematica è una teoria e tecnica per l'analisi e il processing di strutture geometriche fondata sulla teoria degli insiemi, dei reticoli, sulla topologia e sulle funzioni random. I quattro operatori principali sono: erosioni, dilatazione, apertura e chiusura.

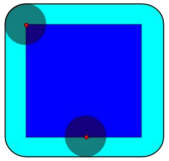
6.10.2 Erosione

In uno spazio euclideo E, L'erosione dell'immagine A con lo structuring element B è la combinazione di tutti i punti che può raggiungere il centro di B quando si muove all'interno di A.



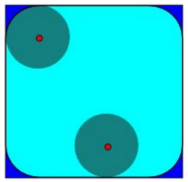
6.10.3 Dilatazione

La dilatazione dell'immagine A con lo structuring element B è la combinazione di tutti i punti che può raggiungere B quando il centro di B si muove all'interno di A.



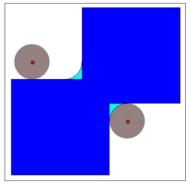
6.10.4 Apertura

L'apertura è la traslazione dello structuring element B all'interno di A.



6.10.5 Chiusura

La chiusura è il complemento della traslazione dello structuring element B all'esterno dall'immagine A.



7 Introduzione GPU e CUDA

7.1 Legge di Moore

"Il numero di transistor nei circuiti integrati raddoppia circa ogni 18 mesi"

7.2 Ostacoli tecnici

Vi sono principalmente 3 ostacoli quando si vuole progettare un hardware efficiente:

- Memory wall: crescente divario tra la velocità di processore e memoria.
- ILP wall: (Instruction Level Parallelism) aumentando il numero di core dei processori cresce la difficoltà nel programmarli.
- Power wall: aumentando la potenza aumenta anche il consumo di energia e di conseguenza la produzione di calore.

7.3 Accelerazione hardware

Designare l'HW per risolvere problemi specifici. L'esempio più comune è la GPU. La tecnica più utilizzata per sfruttare questi accelleratori è l'*heterogeneous computing*.
Heterogeneous computing Gestire i compiti da eseguire e attribuendoli all' HW corretto (CPU/GPU).

7.4 GPU

Componenti HW che eseguono al meglio operazioni simili tra loro (immagine mentale → un "esercito di soldati" che eseguono la stessa operazione). Le caratteristiche tipiche di una GPU sono:

- Le GPU sono processori specializzati con memoria dedicata e con un design multi-core ottimizzato per SIMD(Single Instruction Multiple Data) e FPU(Floating Point Unit).
- Le GPU dedicano proporzionalmente più transistor alle unità aritmetiche/ logiche e meno alla cache e al controllo di flusso in confronto a una CPU (no branch prediction/speculative execution, etc.). Le GPU hanno in genere anche una memoria DRAM più veloce.
- Le GPU hanno un'architettura di throughput parallela per ottimizzare l'esecuzione di molti thread simultanei lentamente, piuttosto che eseguirne uno solo molto rapidamente.

7.5 Tensor Core

Teconologia rivoluzionaria che ottimizza le tecniche di deep learning.

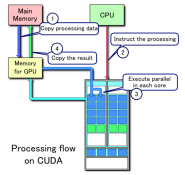
7.6 Programmare la GPU

Nozioni:
Host La CPU e la sua memoria (memoria host)
Device La GPU e la sua memoria (memoria device)

7.7 CUDA

Compute Unified Device Architecture è una piattaforma di calcolo parallelo e che adotta un modello di programmazione inventato da NVIDIA il quale consente di sfruttare la potenza delle GPU Nvidia. Sfrutta l'architettura SIMT (Single Instruction Multiple Threads).

7.7.1 Modello di programmazione CUDA



7.7.2 Memoria Unificata

La memoria unificata è un sistema di gestione della memoria che semplifica lo sviluppo della GPU fornendo un unico spazio di memoria direttamente accessibile da tutte le GPU e CPU nel sistema. La migrazione delle pagine tra memoria host e memoria device serve a ridurre la latenza della memoria.

7.7.3 Kernel

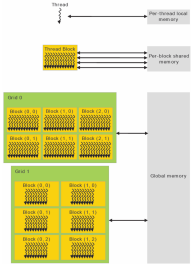
Possiamo definire come kernel una funzione che esegue sulla GPU richiamata dall'host. Quidni l'esecuzione su GPU avviene tramite i kernel che sono programmati sfruttando CUDA. Ogni kernel deve eseguire il minor lavoro possibile.

7.7.4 Esecuzione Del Kernel

Quando un programma CUDA sulla CPU richiama un kernel, i blocchi della griglia vengono distribuiti a tutti i core della GPU disponibili. Ogni blocco di thread esegue parallelamente su un multiprocessore e più blocchi di thread possono essere eseguiti in modo concorrente sullo stesso multiprocessore. Quando i blocchi di thread terminano, vengono avviati nuovi blocchi sul multiprocessore vuoto. Il gruppo di thread che esegue un kernel è organizzato come griglia di blocchi di thread. Queste griglie hanno:

- Numero limitato di thread per blocco.
- Consentono di eseguire la singola chiamata di un kernel con più thread
- I blocchi sono identificabili tramite id.
- I blocchi possono essere array mono o bidimensionali.

Il multiprocessore crea, gestisce, schedula ed esegue i thread in gruppi da 32 chiamati warps. Ogni singolo thread che costituisce un warp parte dallo stesso punto nel programma. Tuttavia, i thread hanno il proprio address counter e il proprio stato dei registri, quindi eseguono in modo indipendente. I thread sono raggruppati in warps sempre allo stesso modo. Ogni warp contiene i thread in ordine partendo dal thread con id 0. I blocchi rimangono in esecuzione finche ogni thread non ha finito il suo lavoro. Un singolo warp esegue in modo efficiente solo se tutti i thread che lo compongono eseguono le stesse istruzioni nello stesso momento, se uno o più thread cambiano "il ramo di esecuzione" in seguito ad una condizione (if) i thread vengono separati e disabilitati per poi essere eseguiti in modo seriale. Se durante l'esecuzione di un warp è richiesto a più thread di eseguire un'operazione atomica sulla stessa area di memoria questa operazione verrà eseguita in modo seriale in ordine casuale.



8 Programmare le GPU con CUDA

CUDA fornisce, attraverso API runtime e driver, diverse aggiunte agli ambienti C/C++, ma ha alcune limitazioni sul device code comparate allo standard completo. Per esempio non supporta i tipi di informazioni runtime (RTTI)

8.1 CUDA Kernel

Il qualificatore global dichiara una funzione che deve essere un kernel eseguibile con i device CUDA. Questa funzione può essere chiamata solo dall'host. Tutti i kernels devono essere dichiarati con il tipo di ritorno void.

8.2 Qualificatori per il Memory Management

__device__ Eseguibile e richiamabile solo da Device. Risiede nella global memory sapce, è lifetime of an application ed è accessibile da tutti i threads all'interno della griglia e dall'host attraverso la runtime library

__global__ Eseguibile sul device, richiamabile solo dall'host

__costant__ Risiede nella (global) costant memory space, è lifetime of an App. ed è accessibile da tutti i threads all'interno della griglia e dall'host

__host__ Eseguibile e richiamabile solo da host

__shared__ Risiede nella shared memory space del thread block. Ha lifetime sul blocco. Accessibile da tutti i threads attraverso i blocchi

8.3 Chiamare il kernel

```
<<<                    >>>
```

Func Nome funzione da usare

Dg Dimensione del grid geometry (numero di gruppi) -> datatype: dim3

Db Dimensione dei block geometry (numero di threads in un gruppo) dim3

Ns (opzionale) numero di bytes nella shared memory allocati dinamicamente. size_t

dim3 datatype per Dg e Db. Inizializzazione: dim3 grid(512, 512, 1); grid(x,y,z) -> z opzionale = 1

Dg.x*Dg.y Numero di blocchi caricati

Db.x*Db.y*Db.z Numero di threads per blocco

8.4 Trovare gli indici

Calcoli per identificare gli indici di spostamento all'interno della matrice che lavora la GPU

- int x = blockIdx.x * blockDim.x + threadIdx.x;
- int y = blockIdx.y * blockDim.y + threadIdx.y;

8.5 Variabili Built-in

grimDim dim3 - dimensione della griglia

blockIdx uint3 - indice del blocco all'interno della griglia

blockDim dim3 - dimensione del blocco

threadIdx uint3 - indice del thread all'interno del blocco

8.6 Register Memory

La forma più veloce di memoria sulla GPU, risiede sul multi-core, è unicamente accessibile dai threads ed è lifetime of the thread.

8.7 Shared Memory

Può essere veloce come un registro quando non ci sono conflitti sui banchi di memoria o quando si legge dallo stesso indirizzo. Accessibile da qualsiasi thread del blocco da cui è stato creato. È lifetime of the block.

8.8 Global Memory

Potenzialmente 150x più lenta di un registro o della Shared memory. Bisogna fare attenzione alla scrittura e lettura quando non sono coerenti tra di loro. Accessibile da entrambi i lati (host o device). Lifetime of the application.

8.9 Local Memory

È potenzialmente un collo di bottiglia, risiede nella global memory, può essere 150x più lento di un registro o di una shared memory, è accessibile solo via thread ed è lifetime of the thead. Usato automaticamente da CUDA per: register overflow, array addressed.. e letture nella memoria globale

8.10 Costant Memory

Cached access to global memory, di sola lettura, limitato a 64kb per applicazione, condiviso da tutti i threads e dall'istruction cache. La cache è 8kb

8.11 Texture Memory

Cached access to global memory, di sola lettura, L1 shared by multiple processors (a seconda della generazione) e L2 su tutti i threads. Ottimizzata per accessi 2D. L1 cache è da 6 a 8 kb (dipende dal processore)

8.12 Syncthreads

- I thread all'interno di un blocco possono cooperare condividendo i dati attraverso una memoria condivisa e sincronizzano la loro esecuzione per coordinare gli accessi alla memoria
- I punti di sincronizzazione sono specificati nel kernel attraverso la chiamata __syncthreads() (eseguito attraverso una barriera in cui tutti i thread nel blocco devono attendere prima di poter procedere)
- Per una cooperazione efficiente, la memoria condivisa dovrebbe essere una low-latency memory vicino a ciascun core del processore __syncthreads() dovrebbe essere leggero

8.13 Processing Flow

1. Copia input data dalla GPU memory alla GPU memory
2. Carica il programma GPU, lo esegue e fa caching data on chip per la performance
3. copia i risultati dalla memoria della GPU alla memoria della CPU

9 Video Processing e Computer Vision

9.1 Fenomeno Beta-Phi

Il cervello ricostruisce un'esperienza di movimento apparente se vede una sequenza di immagini quasi identiche a 10/12 FPS.

9.2 Formati

Proiettori

- Per i film amatoriali: 8mm, super 8 e 9.5 mm
- Per il cinema: 35mm, 70mm

Salvataggio Betamax, cassette VHS, DVD e dischi Blu-ray ad alta definizione.

9.3 Tecnologie per il display

- CRT (Cathode Ray Tube): tubo sottovuoto contenente uno o più emettitori di fasci di elettroni accelerati o deflessi su uno schermo fluorescente.
- DLP (Digital Light Processing): tecnologia per video proiettori che sfrutta micro-specchi digitali.
- PDP (Plasma Display Panel): pannello piatto che utilizza piccole celle contenenti gas ionizzato caricato elettricamente.
- LCD (Liquid Crystal Display): pannello piatto che utilizza cristalli liquidi modulati dalla luce generata dietro al pannello.
- OLED (Organic Light-Emitting Diode): tecnologia che sfrutta la luce emessa da LED, in combinazione con un elemento elettroluminescente composto da uno strato di materiale organico capace di produrre differenti tipi di luce in risposta a una sollecitazione elettrica.

9.4 Risoluzione

Rappresentata dalla quantità di pixel orizzontali. **Nel cinema** 2K (2048×1080 quindi 2.2 megapixels) o 4K (4096×2160 quindi 8.8 megapixels).
Nelle televisioni e nei monitor SD (Standard Definition): 576i, sistema PAL/SECAM Europeo, 480i, sistema NTSC Americano, HD (High Definition): 720p - 1280×720p progressive scan quindi 921'600 pixels, 1080i - 1920×1080i interlacciato quindi 1'036'800 pixels, 1080p - 1920×1080p quindi 2'073'600 pixels, UHD (Ultra High Definition): 4K - 3840×2160 progressive scan, True 4K - 4096×2160, 8K - 7680×4320, True 8K - 8192×4320.

9.5 Aspect ratio

Descrive la proporzione fra la larghezza e l'altezza di un'immagine.

- 4:3 è il formato utilizzato dall'invenzione della cinematografia, i vecchi film a 35 mm.
- 16:9 è il formato standard internazionale dell'HDTV e del DVD.
- 21:9 è un formato cinematografico recente utilizzato oggi negli ultra-wide monitors.

9.6 Framerate

Frequenza con cui le immagini vengono mostrate in un filmato.

9.7 Video processing

come l'image processing si utilizzano tecniche di filtraggio dei segnali applicandole a streams di tipo video. Le tecniche di video processing vengono impiegate per: modifica dell'aspect ratio, zoom digitale, aggiustamento di brightness, contrast, hue, saturation, sharpness, gamma, conversione del frame rate, aggiustamento del colore, riduzione del rumore, tecniche di upscaling.
Le tecniche attuate a livello di singola immagine vengono dette tecniche di *intra-frame processing*. Mentre quelle che sfruttano l'informazione temporale esistente fra più immagini in sequenza vengono dette tecniche di *inter-frame processing*.

9.7.1 Formati di codifica e compressione video

Modo di rappresentare i dati per il salvataggio e la trasmissione. Sono definiti da una specifica e il programma che li codifica prende il nome di codec (MPEG-2 Part 2, MPEG-4 Part 2, H.264 (MPEG-4 Part 10), HEVC e RealVideo RV40).

9.7.2 Formati container per il multimedia

Contengono il video combinato con uno stream audio (AVI, MP4, FLV, RealMedia, Matroska, o QuickTime).

9.7.3 Compressione video

Esistono sia compressori lossy che lossless. Alcuni formati video come il Dirac e l'H264 permettono entrambe le tipologie.

9.7.4 Formati intra-frame

La compressione viene applicata esclusivamente alle singole immagini senza approfittare della correlazione tra immagini successive. Questa soluzione è meno computationally intensive delle soluzioni che applicano interframe prediction.

9.7.5 Inter-frame prediction

I video contengono molta informazione ridondante spaziale, ma soprattutto temporale. Algoritmi di compressione come H.264 sfruttano questa ridondanza con tecniche di inter-frame prediction che gli permettono di raggiungere rapporti di compressione di 1:50.

9.7.6 Motion JPEG

Formato di compressione video intraframe in cui ogni frame è compresso come immagine JPEG ha un'efficienza limitata a 1:20.

9.7.7 Computer Vision

Tecnologie e metodi per acquisire, modificare ed analizzare immagini con lo scopo di estrapolare informazioni per poter prendere decisioni in maniera automatica.

9.7.8 Struttura di un sistema di computer vision

1. Acquisizione dell'immagine
2. Pre-processing: rimozione del rumore, l'adeguamento del contrasto, ecc.
3. Estrazione delle feature: estrazione di caratteristiche (linee, contorni).
4. Detection/Segmentation: decidere quali punti o regioni dell'immagine sono rilevanti.
5. Processing di tipo high-level: classificare gli oggetti, determinarne posizione, dimensione ...

9.7.9 Background substraction

Estrae gli elementi in primo piano all'interno di un video confrontando il frame corrente con un frame di riferimento (background). Tecniche di background subtraction:

- Frame differencing (sottrazione tra pixel dell'immagine attuale e quella di background).
- Mean filter
- Gaussian average.

9.7.10 Video motion Traking

Tecnica di video tracking che associa gli oggetti in frames video consecutivi. I componenti principali sono: rappresentazione e localizzazione del target, filtraggio e associazione dei dati.

9.7.11 Mean shift

Tecnica di analisi non parametrica per identificare i maxima (detti modes) di una funzione di densità.

10 Audio e Video Streaming

10.1 Broadcasting

Distribuzione contenuti audio/video tramite un mezzo di comunicazione (onde radio) con modello 1-molti.

- **Radio broadcasting**: trasm. audio tramite tecnologia radio (oggi anche via cavo/Internet).
  - **AM** (Amplitude Modulation): modula ampiezza in base all'intensità del segnale. Produce rumore nella ricezione (RFI Radio Frequency Interferation).
  - **FM** (Frequency Modulation): mod. freq. nella banda VHF(Very High Frequency) con range 30-300MHz, lung. d'onda 1-10m
- **Television broadcasting**: trasm. anche foto in movimento. Usavano onde-radio, ora dig. terrestre, via satellite, overIP
  - **via cavo**: Trasm. digitale con coxial-cable (anche per telefono, radioFM, internet. C'è conduttore interno circondato da uno strato d'isolazione. Coassile perché conduttore e schermatura condividono stesso asse geometrico)
  - **via satellite**: satell. che orbita attorno la terra. Un'antenna (oggi piccola perché il segnale è digitale) riceve segnale
  - **digitale terrestre**: per radio si usa DAB(Digital Audio Broadcasting). Segnale analogico digitale tramite compress. con formati MP2. Per televisione sfrutta onde radio per trasm. segnale digitale. Formati: ATSC,DVB,ISDB,DTMB
  - **via internet**: VoIP (Voice), RoIP(radio), IPTV(Internet Prot. Television). Si usa la fibra ottica(piccoli tubi di vetro che trasm. fascio luminoso= informazione (26000 + veloce del twi-pair))

via cavo: Trasm. digitale con coxial-cable (anche per telefono, radioFM, internet. C'è conduttore interno circondato da uno strato d'isolazione. Coassile perché conduttore e schermatura condividono stesso asse geometrico) via satellite: satell. che orbita attorno la terra. Un'antenna (oggi piccola perché il segnale è digitale) riceve segnale digitale terrestre: per radio si usa DAB(Digital Audio Broadcasting). Segnale analogico digitale tramite compress. con formati MP2. Per televisione sfrutta onde radio per trasm. segnale digitale. Formati: ATSC,DVB,ISDB,DTMB via internet: VoIP (Voice), RoIP(radio), IPTV(Internet Prot. Television). Si usa la fibra ottica(piccoli tubi di vetro che trasm. fascio luminoso= informazione (26000 + veloce del twi-pair))

10.2 Streaming

- Contenuti a/v tramite internet
- Non necessario scaricare tutto il contenuto prima di poterlo sentire/vedere
- Riproduzione inizia "istantaneamente" (YouTube,Spotify...)

Tre categorie di streaming:

- **Live media**: accedere alla programmazione dal vivo con o senza interattività (TV). Streaming real-time. Hard da realizzare: necessita videocamere, microfoni,encoder real-time per digitalizzazione, strumento di publishing e una CDN per distribuirli.
- **Time-shifted media**: acc. alla programmazione passata
- **Video on-demand(VOD)**: acc. ad un catalogo di contenuti disponibili per lo streaming (Netflix)

Problemi: Packet-loss, Packet-delay, Network jitter (ritardi di ricezione per congestione di code nel router). Poca larghezza di banda = poca qualità di ricezione e dati non arrivano in tempo.
Unicast vs Multicast: (1) invia una copia separata dallo stream a ogni user. Scalabilità scarsa se + utenti ricevono gli stessi contenuti. (2) riduce carico di lavoro su server. Unica stream dalla sorgente agli utenti. Disponibilità non sempre garantita. Routers e FW possono ostacolare passaggio
CDN = Content Delivery Network, stream dalla sorgente ad un'infrastruttura centrale cloud che distribuisce dei contenuti agli utenti.
LAN vs WAN: overIP, (1) si usa Dante o Ravenna (tecnologia str. audio su LAN basata su RTP, supporta unicast/multicast, usa SDP (Session Description Prot. che ha sintassi testuale, utile per gestire la conn. e stream), sync con PTPv2, QoS con DiffServ(> scalabilità di IP, assicura QoS in app end-to-end, CDN-client). (2) + complesso perché ci sono routers, fw, ISP(no multicast) che riducono la conn. HTTP unico canale di comunicazione.
Protocolli e formati: (Network) IP, (Transport) TCP, UDP, (Application) HTTP, FTP, RTP, RTMPT(http tunnel), (Dynamic Adaptive Streaming(su HTTP))HLS(Apple), MPEG-DASH, Smooth Streaming(Microsoft)

- **RTP**: Real-Time Transport Protocol, a/v in real-time overIP. UDP con RTCP (per sync delle streams). supporta multicast.
- **MPEG-DASH**: tecnica bitrate adattiva. Non è proprietario, std internazionale. I contenuti sono divisi in segmenti (es. 10 sec) messi a disposizione in bitrates differenti. In base alla condizione della rete, il ricevente è responsabile della scelta del formato adeguato da scaricare e riprodurre (è anche responsabile di pacc. Mancanti o out-of-order). Download con HTTP. A differenza di HLS e Smooth non ha vincoli di codec (prog. o dispositivo che cod/decod digitalmente un segnale perché possa essere salvato su un supporto di memor.)

11 Gstreamer

- Framework open source per la creazione di applicazioni multimediali
- Piattaforma altamente modulare sviluppata in C
- Multi-platform: Linux-kernel based, Android, MacOS, iOS, Windows
- Esistono dei bindings per vari linguaggi: ad esempio Python, C++, Perl, Ruby, C#, ...
- Utilizzabile anche da linea di comando
- Utilizza un'architettura a plug-ins
- Molti elementi quali codecs, container formats, input e output drivers ed effetti sono forniti al momento dell'installazione
- Possibilità di sviluppare ulteriori plug-ins utilizzando il linguaggio C

Elemento è il componente base di una pipeline, ha delle proprietà alle queali si possono assegnare dei valori. 3 elementi di base: source, filter e sink
Pipelines insiemi di elementi collegati sequenzialmente.
Pipeline di base -> almeno una source, almeno un sink e uno o più elementi filter descritta da una serie di caratteri separati da !

11.1 Elementi (audio)

Generazione segnali audio di test:
audiotestsrc -> generatore di suoni (utilizzato per testare delle pipelines audio). Il tipo di suono è scelto utilizzando la proprietà wave
Audio adapters:
audioconvert -> converte dei dati audio raw in vari formati
audioresample -> esegue una conversione di sample rate
Ascolto di audio:
autoaudiosink -> seleziona automaticamente la device di riproduzione dell'audio

11.2 Elementi (video)

Generazione video di test:
videotestsrc -> generatore di pattern video (utilizzato per testare delle pipelines video). Il tipo di pattern è scelto utilizzando la proprietà pattern
Video adapters:
videoconvert -> converte da un color space ad un altro (ad esempio da RGB a YUV)
videorate -> esegue una conversione di sample rate (eliminando o duplicando dei frames)
videoscale -> ridimensiona il video
Visualizzazione dei media: **autovideosink** -> seleziona automaticamente la device di riproduzione del video

11.3 Elementi utili

Bins: elementi in grado di costruire dinamicamente una pipeline in base alle informazioni estratte dal media
playbin -> gestisce tutti gli aspetti di riproduzione di un media (dalla lettura, alla decodifica fino alla riproduzione)
uridecodebin -> decodifica i dati ricevuti da un URI fino ad ottenere dei dati raw
decodebin -> decodifica i dati in entrata fino ad ottenere dei dati raw

11.4 File input / output

filesrc -> legge i dati presenti in un file
filesink -> scrive i dati in input su di un file

11.5 Gst tools

gst-inspect-1.0 -> Mostra una lista di tutti gli elementi disponibili
gst-inspect-1.0 [PLUGIN | ELEMENT] -> Mostra le info disponibili per il plugin/elemento selezionato.
gst-launch-1.0 PIPELINE_DESCRIPTION -> Tool per creare ed eseguire una pipeline Utilizzando l'opzione -v si attiva il supporto verbose

11.6 Debug

GST_DEBUG -> Variabile d'ambiente utilizzata per specificare il livello di debug desiderato
Level Name Description
0 none No debug information is output.
1 ERROR Logs all fatal errors.
2 WARNING Logs all warnings.
4 INFO Logs all informational messages.
5 DEBUG Logs all debug messages.
6 LOG Logs all log messages.
GST_DEBUG_DUMP_DOT_DIR -> Variabile d'ambiente utilizzata per specificare la path dove salvare il grafico della pipeline

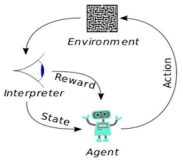
12 Machine Learning

Campo dell'informatica con l'obiettivo di sviluppare funzionalità per rendere i computers capaci di imparare senza che li si programmi in maniera esplicita. La realizzazione di buoni algoritmi di machine learning è però un compito complesso perché riconoscere i patterns può essere difficile e spesso i dati disponibili non sono sufficienti. Tipi di machine learning:

- Supervised learning
- Reinforcement learning
- Unsupervised learning

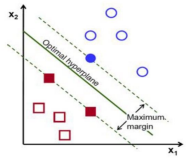
12.1 Supervised learning

Nel supervised learning all'algoritmo vengono forniti sia esempi di input, che i rispettivi dati di output, da un "educatore" dell'algoritmo. L'algoritmo cerca di scoprire la regola generale che associa gli inputs agli outputs.



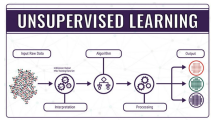
12.2 Reinforcement learning

Nelle forme più estreme, come ad esempio nel reinforcement learning, come unica forma di feedback l'output viene approvato o respinto. Il reinforcement learning viene ad esempio sfruttato nei sistemi di guida automatica.



12.3 Unsupervised learning

Nel unsupervised learning, nessuna informazione viene fornita all'algoritmo, che è costretto a scoprire in maniera autonoma la struttura dei dati in input. È la forma più complessa di machine learning.



12.4 Reti artificiali

Una delle famiglie più utilizzate di algoritmi di machine learning è quella delle reti neurali. I calcoli vengono strutturati in termini di gruppi di neuroni artificiali interconnessi, che processano collettivamente l'informazione. Le neural networks utilizzate oggi sono strumenti non lineari per la modellazione di tipo statistico di dati con una certa complessità.
Vantaggi: generalmente più efficaci e semplici da allenare degli altri algoritmi di machine learning. Offrono più flessibilità di parametrizzazione.
Svantaggi: la loro natura di tipo black-box può risultare svantaggiosa in casi in cui è necessario il controllo del comportamento dell'algoritmo (predictability). Per di più le reti neurali artificiali sono facilmente soggette ad overfitting. Con pochi dati non identificano la caratteristica generica desiderata ma il caso specifico. Per renderle capaci di generalizzare è necessario allenarle con data sets molto ampi e ben costruiti.

12.5 Deep Learning

La crescita delle capacità di calcolo, la progressiva riduzione del costo dell'hardware e l'evoluzione delle GPUs degli ultimi anni, hanno contribuito allo sviluppo del concetto di deep learning. Vengono utilizzate reti neurali composte da layers multipli. Ogni layer impara a trasformare i dati di input in una rappresentazione ogni volta leggermente più astratta e composta. Nelle soluzioni end-to-end l'intero processo, dai sensori agli attuatori, viene eseguito da un'unica layered o recurrent neural network senza modularizzazione. Vantaggio: capacità di continuare a migliorare le performances all'aumentare dei dati di apprendimento disponibili.
Per quanto concerne l'ambito del multimedia, oggi esistono applicazioni di successo soprattutto nella computer vision e nel riconoscimento vocale. Ad esempio, il deep learning viene utilizzato con successo nella classificazione di immagini.
Esempi:

- Amazon Rekognition: aggiunge funzionalità di analisi d'immagine alle applicazioni.
- Amazon Transcribe: analizza file audio e li traduce in file di testo.
- Caffe: framework di deep learning veloce, modulare e facile da utilizzare.

13 Code

13.1 Lego Identifier

```
using namespace std;
using namespace cv;
int main(int argc, char **argv) {
  Mat image,greyImage,threShold,erodeImage,dilateImage;
  vector <vector<cv::Point>> contours;
  vector<double> aree;
  vector<cv::Point> label;
  cv::Moments M;
  int x, y;
  vector<String>colori;
image = imread("lego.png", CV_LOAD_IMAGE_COLOR); // Read
 ↪  the file
  if (!image.data) // Check for invalid input{
    std::cout << "Could not open or find the image" <<
     ↪  std::endl;
    return -1;}
// 1. convert in grayscale and compute a threshold
  cv::cvtColor(image, greyImage, CV_BGR2GRAY);
  cv::threshold(greyImage, threShold, 200, 255,
     ↪  cv::THRESH_BINARY_INV);
// 2. filter the noise by using one or more morphological
 ↪  filters
  Mat kernel=cv::getStructuringElement(cv::MORPH_RECT,
    ↪  cv::Size(5, 5));
  cv:erode(threShold, erodeImage, kernel);
  cv::dilate(erodeImage, dilateImage, kernel);
// 3. find out how to use the findContours algorithm and
 ↪  use it to segment the lego blocks
  cv::findContours(dilateImage, contours,
    ↪  cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);
  Mat contoursImage = imread("lego.png",
    ↪  CV_LOAD_IMAGE_COLOR);
  for (int i = 0; i < contours.size(); i++) {
    cv::drawContours(contoursImage, contours, i,
      ↪  cv::Scalar());
    label.push_back(contours.at(i).at(0));}
// 4. find a way to understand size and color of each lego
 ↪  block
  for (int i = 0; i < contours.size(); i++) {
    aree.push_back (cv::contourArea(contours[i]));}
  //centro delle figure
  for (int i = 0; i < aree.size(); i++) {
    M = cv::moments(contours.at(i));
    x = int(M.m10 / M.m00);
    y = int(M.m01 / M.m00);
    Point punto{ x, y };
    Scalar scalar{ image.at<cv::Vec3b>(punto) };
    int blu = int(scalar[0]);
    int green = int(scalar[1]);
    int red = int(scalar[2]);
    //adesso classifico i rettangoli per colore

    if (red == 0 && green == 0 && blu == 0) {
      colori.push_back("Nero");}
    }}
  for (int i = 0; i < aree.size(); i++) {
    if (aree.at(i) > 12000 && aree.at(i) < 16000) {
      std::stringstream testoLabel;
      testoLabel << "2x2; " << colori.at(i);
```

```
      cv::putText(image, testoLabel.str(),
        label.at(i),
        cv::FONT_HERSHEY_DUPLEX,
        1.0,
        CV_RGB(0, 0, 200), //font color
        2);}
//Same as above
// 5. put a text label near each block with its properties
  ↪  (size & color)
  cv::imwrite("immagine_finale.jpg", image);
  cv::imshow("image", image);
  cv::waitKey(0);
  return 0;}
```

13.2 CUDA

```
__global__ static void readChannel(
  unsigned char * channelData,
  unsigned char *source,
  int imageW,
  int imageH,
  int channelToExtract,
  int numChannels)
{
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int i = (y * (imageW * numChannels)) + (x * numChannels)
   ↪  + channelToExtract;
  int i_mono = (y * (imageW) + x);
  channelData[i_mono] = source[i];
}

__global__ static void writeChannel(
  unsigned char* destination,
  unsigned char* channelData,
  int imageW,
  int imageH,
  int channelToMerge,
  int numChannels)
{
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int i = (y * (imageW * numChannels)) + (x * numChannels)
   ↪  + channelToMerge;
  int i_mono = (y * (imageW) + x);
  destination[i] = (channelData[i_mono]);
}
__global__ void compute_average(
  unsigned char* h_Dst,
  unsigned char* h_Src,
  int imageW,
  int imageH)
{
  const unsigned int numElements = ((2 * KERNEL_RADIUS) +
   ↪  1) * ((2 * KERNEL_RADIUS) + 1);
  // Calculate index
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  unsigned int sum = 0;
  for (int kY = -KERNEL_RADIUS; kY <= KERNEL_RADIUS; kY++)
   ↪  {
    const int curY = y + kY;
    if (curY < 0 || curY > imageH) {
      continue;
    }
    for (int kX = -KERNEL_RADIUS; kX <= KERNEL_RADIUS;
     ↪  kX++) {
      const int curX = x + kX;
      if (curX < 0 || curX > imageW) {
        continue;
      }
      const int curPosition = (curY * imageW + curX);
      if (curPosition >= 0 && curPosition < (imageW *
       ↪  imageH)) {
        sum += h_Src[curPosition];
      }
    }
  }
  h_Dst[y * imageW + x] = (unsigned char)(sum /
   ↪  numElements);
}

void average_gpu(
  unsigned char* inputImage,
  unsigned char* outputImage,
  int imageW,
  int imageH,
  int numChannels
) {
  int size = imageW * imageH * numChannels *
   ↪  sizeof(unsigned char);
  unsigned char* d_inputImage;
  unsigned char* d_outputImage;
  cudaMalloc((void **)&d_inputImage, size);
  cudaMalloc((void **)&d_outputImage, size);
  unsigned char* d_in_channel;
  unsigned char* d_out_channel;
  cudaMalloc((void **)&d_in_channel, size / numChannels);
  cudaMalloc((void **)&d_out_channel, size / numChannels);
  int curChannel;
  dim3 dimBlock(16, 16);
  dim3 dimGrid(imageW / dimBlock.x, imageH / dimBlock.y);
  cudaMemcpy(d_inputImage, inputImage, size,
   ↪  cudaMemcpyHostToDevice);
  for (curChannel = 0; curChannel < numChannels;
   ↪  curChannel++) {
    readChannel<<<dimGrid, dimBlock>>>(d_in_channel,
     ↪  d_inputImage, imageW, imageH, curChannel,
     ↪  numChannels);
    compute_average<<<dimGrid, dimBlock>>>(d_out_channel,
     ↪  d_in_channel, imageW, imageH);
    writeChannel<<<dimGrid, dimBlock>>>(d_outputImage,
     ↪  d_out_channel, imageW, imageH, curChannel,
     ↪  numChannels);
  }
  cudaMemcpy(outputImage, d_outputImage, size,
   ↪  cudaMemcpyDeviceToHost);
  cudaFree(d_in_channel);
  cudaFree(d_out_channel);
}
```

13.3 Video Processing (Tracker)

```
using namespace std;
using namespace cv;

int main(int argc, char **argv) {
  unsigned long frameNumber = 0;
  Rect trackWindowLeft = Rect(60, 170, 100, 100);
  float xLeft = 70;
  int cntLeft = 0;
  int cntAutoLeft = 0;
  Rect trackWindowRight = Rect(810, 130, 100, 100);
  float xRight = 830;
  int cntRight = 0;
  int cntAutoRight = 0;
  VideoCapture cap("./../videoauto.mp4");
  if (!cap.isOpened())
    return -1;
  namedWindow("video", 1);
  Mat frame, grayScaleFrame, fgMaskMOG2, filterFrame,
   ↪  thresholdFrame;
  Rect trackWindow, trackWindow1;
  RotatedRect trackBoxPrec, trackBoxPrec1;
  trackWindow = Rect(880, 180, 20, 20);
  trackWindow1 = Rect(74, 210, 30, 30);
  Ptr<BackgroundSubtractor> pMOG2;
  pMOG2 = createBackgroundSubtractorMOG2();
  for (;;) {
    frameNumber++;
    cap >> frame;
    if (frame.rows == 0 && frame.cols == 0) {
      cout << "last frame!" << endl;
      break;
    }
    // 1. convert in grayscale
    cvtColor(frame, grayScaleFrame, CV_BGR2GRAY);
    // 2. use the background subtraction MOG2 algorithm to
     ↪  obatin the fore ground image
    pMOG2->apply(grayScaleFrame, fgMaskMOG2);
    // 3. filter the noise by using one or more
     ↪  morphological filters
    Mat kernel = cv::getStructuringElement(cv::MORPH_RECT,
     ↪  cv::Size(2, 2));
    cv:erode(fgMaskMOG2, filterFrame, kernel);
    cv::dilate(filterFrame, filterFrame, kernel);
    cv::GaussianBlur(filterFrame, filterFrame, Size(), 2);
    // 4. apply a threshold to better define the fore
     ↪  ground
    cv::threshold(filterFrame, thresholdFrame, 50, 255,
     ↪  cv::THRESH_BINARY);
    // 5. if needed, use one ore more morphological
     ↪  filters to improve the fore ground definition
    // 6. find a way to track the cars using the camshift
     ↪  algorithm (hint: implement an object to handle the
     ↪  tracker)
    Rect copyTrackWindowLeft = trackWindowLeft;
    Rect copyTrackWindowRight = trackWindowRight;
    RotatedRect TrackBoxLeft = CamShift(thresholdFrame,
     ↪  copyTrackWindowLeft,
      TermCriteria(TermCriteria::EPS |
       ↪  TermCriteria::COUNT, 10, 1));
    RotatedRect TrackBoxRight = CamShift(thresholdFrame,
     ↪  copyTrackWindowRight,
      TermCriteria(TermCriteria::EPS |
       ↪  TermCriteria::COUNT, 10, 1));
    // 7. extract the requested information (number of
     ↪  vehicles, number of vehicles for each direction,
     ↪  timestamp, color)
    // 8. save ONE Nj for the vehicle with the requested
     ↪  information written on
    if (TrackBoxLeft.size.width > 0 && xLeft <=
     ↪  TrackBoxLeft.center.x) {
      cntLeft = 0;
      trackWindowLeft = TrackBoxLeft.boundingRect();
      xLeft = TrackBoxLeft.center.x;
      if (TrackBoxLeft.center.x > 300) {
        // Obtaining color to parse
        Mat frameAuto = frame(trackWindowLeft);
        Mat filteredAuto = filterFrame(trackWindowLeft);
        Scalar color = mean(frameAuto, filteredAuto);
        // Construction of a color class
        Color autoColor{
         ↪  int(color[0]),int(color[1]),int(color[2])};
        // Cloning frame to save image
        Mat clone = frame.clone();
        // Regnognization of a color and construction of
         ↪  image's label
        ostringstream str;
        str << "Frame: " << frameNumber << " , Color: " <<
         ↪  autoColor.label() << " from left to right";
        putText(clone, str.str(), Point(10, 30),
         ↪  FONT_ITALIC, 0.4, cvScalar(0, 0, 255));
        string name = "./fromLeft" +
         ↪  to_string(cntAutoLeft) + ".jpg";
        imwrite(name, clone);
        // Setting starter parameters
        trackWindowLeft = Rect(60, 170, 100, 100);
        xLeft = 70;
        // Incrementation of the auto detected variable
        cntAutoLeft++;
      }
    }
    if (cntLeft > 3) {
      trackWindowLeft = Rect(60, 170, 100, 100);
      xLeft = 70;
    }
    if (TrackBoxRight.size.width > 0 && xRight >=
     ↪  TrackBoxRight.center.x) {
      cntRight = 0;
      trackWindowRight = TrackBoxRight.boundingRect();
      xRight = TrackBoxRight.center.x;
      if (TrackBoxRight.center.x < 600) {
        // Obtaining color to parse
        Mat frameAuto = frame(trackWindowLeft);
        Mat filteredAuto = filterFrame(trackWindowLeft);
        Scalar color = mean(frameAuto, filteredAuto);
        // Construction of a color class
        Color autoColor{
         ↪  int(color[0]),int(color[1]),int(color[2]) };
        // Cloning frame to save image
        Mat clone = frame.clone();
        // Regnognization of a color and construction of
         ↪  image's label
        ostringstream str;
        str << "Frame: " << frameNumber << " , Color: " <<
         ↪  autoColor.label() << " from right to left";
        putText(clone, str.str(), Point(10, 30),
         ↪  FONT_ITALIC, 0.4, cvScalar(0, 0, 255));
        string name = "./fromRight" +
         ↪  to_string(cntAutoRight) + ".jpg";
        imwrite(name, clone);
        // Setting starter parameters
        trackWindowRight = Rect(810, 130, 100, 100);
        xRight = 830;
        // Incrementation of the auto detected variable
        cntAutoRight++;
      }
    }
    if (cntRight > 3) {
      trackWindowRight = Rect(810, 120, 100, 100);
      xRight = 830;
    }
    rectangle(frame, trackWindowLeft, Scalar(0, 0, 255),
     ↪  3, LINE_AA);
    rectangle(frame, trackWindowRight, Scalar(0, 255, 0),
     ↪  3, LINE_AA);
    cntLeft++;
    cntRight++;
    imshow("Tracking", frame);
    if (waitKey(40) >= 0) break;
  }
  return 0;
}
```

13.4 Audio (Dial Tones)

```
      class Tone {
public:
  Tone(int num, float min, float max, std::string val)
   ↪  :numero{ num }, minValue{ min }, maxValue{ max },
   ↪  valore{ val } {}
  Tone() {}
  bool lookup(float buffer[]) {
    if (buffer[(int)(minValue / (8000 / 1024.0))] > -30 &&
     ↪  buffer[(int)(maxValue / (8000 / 1024.0))] > -30)
      return true;
    return false;}
  int getNumero() {
    return numero;}
  char decode(int repetitions) {
    return valore.at(repetitions - 1);}
private:
  int numero; float minValue; float maxValue; std::string
   ↪  valore;};

int main(int argc, char **argv){
  int i = 0;
  if (argc != 2) {
    std::cerr << "Missing input audio file to analyze." <<
     ↪  std::endl;
    return 1;}
  File input =
   ↪  File::getCurrentWorkingDirectory().getChildFile(argv[1]);
  if (input.exists() == false) {
    std::cerr << "File doesn't exists." << std::endl;}
  AudioFormatManager fmgr;
  fmgr.registerBasicFormats();
  ScopedPointer<AudioFormatReaderSource> source = new
   ↪  AudioFormatReaderSource(fmgr.createReaderFor(input),
   ↪  true);
  unsigned int numChannels =
   ↪  source->getAudioFormatReader()->numChannels;
  double sampleRate =
   ↪  source->getAudioFormatReader()->sampleRate;
  std::cout << "Channels=" << numChannels << "
   ↪  sampleRate=" << (int)sampleRate << std::endl;
  AudioBuffer<float> myBuffer(numChannels, sampleRate /
   ↪  10);
  AudioSourceChannelInfo info(myBuffer);
  source->prepareToPlay(512, sampleRate);
  const int FFTsize = 1024;// perché voglio campionare
   ↪  circa 4 Hz ed devo usare il doppio
  float buffer[2 * FFTsize];
  CircularBuffer<float> circular{ FFTsize, 128 };
  Tone t0{ 0,941,1336 ," " };
  Tone t1{ 1,697,1209, " " };
  Tone t2{ 2, 697,1336,"ABC" };
  //tutti gli altri toni associati agli hz della tabella
  Tone vet[10] = {t0,t1,t2,t3,t4,t5,t6,t7,t8,t9};
  int num = -1;
  Tone ultimoTono{};
  int contTemp = 0; int numeroRep = 0;  std::string
   ↪  messaggio = "";
  while (source->getNextReadPosition() <
   ↪  source->getTotalLength()) {
    // Read next audio block
    source->getNextAudioBlock(info);
    const int numSamples = info.buffer->getNumSamples();
    const int numChannels = info.buffer->getNumChannels();
    const float **data =
     ↪  info.buffer->getArrayOfReadPointers();
    bool finito = false;
    // loop through each channel
    for (int channel = 0; channel < numChannels;
     ↪  channel++) {
      // channelData contains now 'numSamples' of input
       ↪  data
      const float *channelData = data[channel];
    //TODO implement you algorithm! The following example
     ↪  dumps to stdout the audio contents
      std::cout << "Channel=" << channel << " numSamples="
       ↪  << numSamples << std::endl;
      for (int sampleIndex = 0; sampleIndex < numSamples;
       ↪  sampleIndex++) {
        if (circular.process(channelData[sampleIndex])) {
          circular.copyData(buffer);
          MyFilter<float> filtro{ 690, 1650, sampleRate };
          dsp::FFT tras{ 10 };
          filtro.process(buffer, 2 * FFTsize);

           ↪  tras.performFrequencyOnlyForwardTransform(buffer);
          //il buffer ora contiene le frequenze e la fase
          buffer[0] = 20.0f * log10f(buffer[0] /
           ↪  (float)FFTsize / 2.0f);
          for (int i = 1; i < FFTsize / 2; i++) {
            buffer[i] = 20.0f * log10f((2.0f * buffer[i])
             ↪  / (float)FFTsize / 2.0f);}
          bool found = false;
          std::cout << "Numero:" << std::endl;
          for (int i = 0; i < 10; i++) {
            if (vet[i].lookup(buffer)) {
              std::cout << vet[i].getNumero() <<
               ↪  std::endl;
              if (num == -1) {
                numeroRep = 0;
                num = vet[i].getNumero();}
              if (contTemp > 3) {
                numeroRep++;}
              ultimoTono = vet[i];
              found = true;
              contTemp = 0;}}
          if(found==false&& contTemp>10 && num!=-1){
            contTemp=0;

             ↪  messaggio.push_back(ultimoTono.decode(numeroRep)
            num = -1;}
          if (found == false) {
            contTemp++;}
          std::cout << std::endl;}}}
    std::cout << std::endl;}
std::cout << "messaggio trovato: " << messaggio <<
 ↪  std::endl;
  return 0;}
```

13.5 Gstreamer

```
Generare segnale audio:audiotestsrc wave=0 freq=440 !
 ↪  audioconvert ! autoaudiosink
Generare video con pattern diversi: videotestsrc pattern=0
 ↪  ! videoconvert ! autovideosink
Lettura file e riproduzione Audio/video con bin: playbin
 ↪  uri=file:/tmp/BigBuckBunny_320x180.mp4
Solo video con filesrc: filesrc
 ↪  location=/tmp/BigBuckBunny_320x180.mp4 ! decodebin !
 ↪  videoconvert ! autovideosink
Audio/video con filesrc: filesrc
 ↪  location=/tmp/BigBuckBunny_320x180.mp4 ! decodebin
 ↪  name=dec dec.src_0 ! videoconvert ! autovideosink
 ↪  dec.src_1 ! audioconvert ! autoaudiosink
Codifica del video in formato H264
Solo video: filesrc location=/tmp/BigBuckBunny_320x180.mp4
 ↪  ! decodebin ! videoconvert ! x264enc ! avdec_h264 !
 ↪  videoconvert ! autovideosink
Audio/video: filesrc
 ↪  location=/tmp/BigBuckBunny_320x180.mp4 ! decodebin
 ↪  name=dec dec.src_0 ! videoconvert ! x264enc !
 ↪  avdec_h264 ! videoconvert ! autovideosink dec.src_1 !
 ↪  audioconvert ! autoaudiosink
Streaming di video:
Sender: location=/tmp/BigBuckBunny_320x180.mp4 ! decodebin
 ↪  ! videoconvert ! x264enc ! rtph264pay ! udpsink
 ↪  host=127.0.0.1 port=5000
Receiver: udpsrc port=5000 ! application/x-rtp,
 ↪  media=video, encoding-name=H264, clock-rate=90000 !
 ↪  rtph264depay ! avdec_h264 ! videoconvert !
 ↪  autovideosink
```