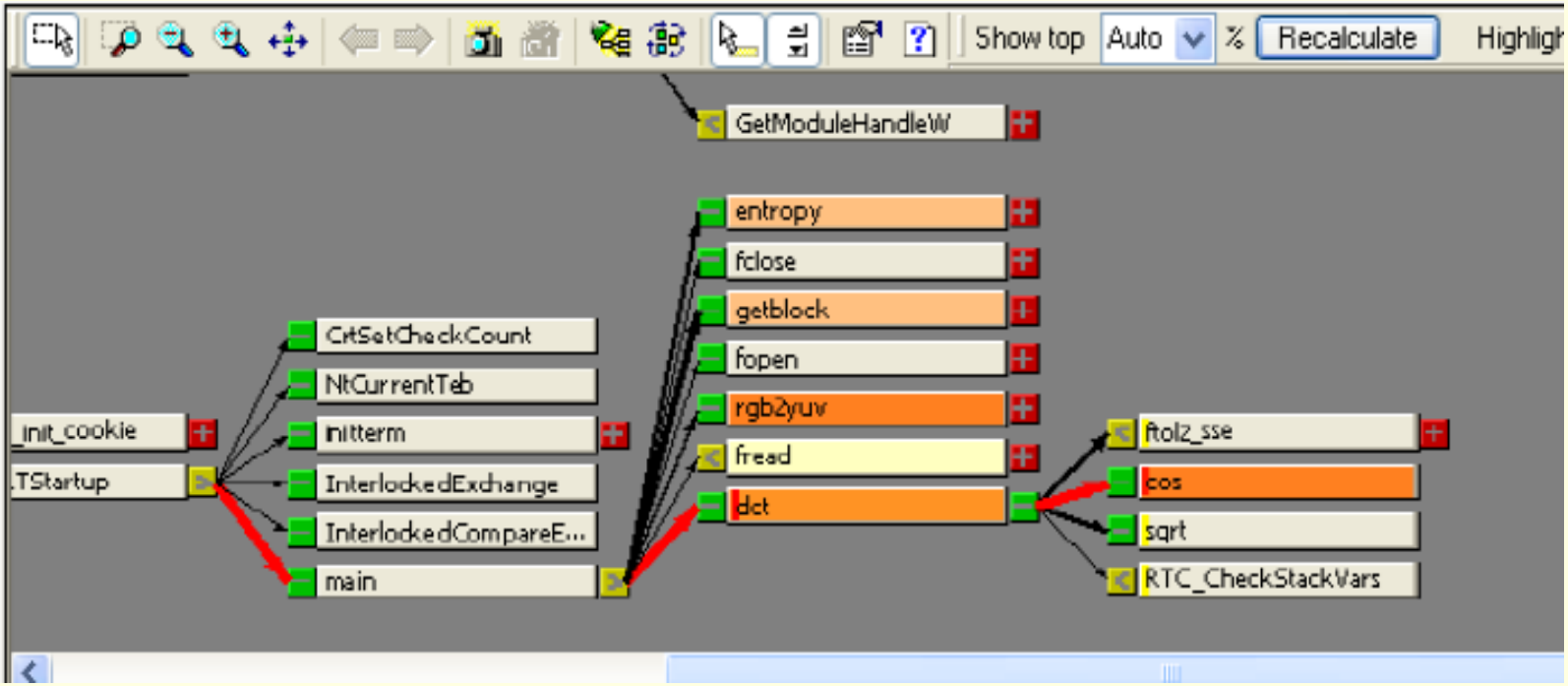
Količine podataka

- Jednostavan izračun daje nam okvirne količine nekih tipičnih formata koje ste upoznali

Rezolucija slike	**Bita/pikselu**	**Veličina**
640x480	8	307kB
1280x1024	24	3,9MB

Video rezolucija	**Slika/s**	**Veličina/s**
640x480 (8bpp)	10	3 MB/s
1280x1024 (24bpp)	30	120 MB/s

I sada počinje ozbiljna analiza..



Kako to implementirati u SW ili HW

- Najzahtjevnije operacije pri obradi slike/videa
  - Procjena/predviđanje pokreta !!!!!....!!!
  - DCT
  - RGB-YUV transformacija
  - Entropijsko kodiranje

Konverzija prostora boja

- Kao što ste naučili RGB način zapisa podataka za sliku nije dobar za kompresiju podataka već se za to koristi YUV (YCrCb) prostor
- Prije pokretanja kompresije slika u RBG zapisu konvertira se u YUV
- Inverzna transformacija obavlja se nakon dekompresije a prije prikaza na zaslonu

Konverzija prostora boja

- RGB-YCbCr konverzija vrlo se lako može obaviti jednostavnom matričnom operacijom:

$$Y = 0.299\ R + 0.587\ G + 0.114\ B$$

$$Cb = -\ 0.1687\ R - 0.3313\ G + 0.5\ B + 128$$

$$Cr = 0.5\ R - 0.4187\ G - 0.0813\ B + 128$$

- Vidimo da za izračun svakog piksela treba 9 množenja i 8 zbrajanja (+dohvat,spremanje)

Konverzija prostora boja

- Iako matematički trivijalna, konverzija boja jedan je od računalno zahtjevnijih zadataka pri kompresiji
- Ako izračunamo koliko je potrebno da se obradi jedna slika 1280x1024:
  - 11,8M množenja
  - 10,5M zbrajanja
  - +dohvat, spremanje

Konverzija prostora boja

- Verzija 2: osnovna aproximacija

Y = 0.299 R + 0.587 G

Cb = - 0.3313 G + 0.5 B + 128

Cr = 0.5 R - 0.4187 G + 128

- Verzija 3: poboljšana aproximacija

```
data[i++] = (byte)(R/4 + G/2);
data[i++] = (byte)(-(G/4) + B/2 + 128);
data[i++] = (byte)(R/2 - (G /2) + 128);
```

Analiza

- Za jednu vrlo zahtjevnu operaciju uspjeli smo značajno smanjiti procesorske zahtjeve
- No to smo postigli uz **degradaciju** kvalitete izračuna podataka
- Da li je kvaliteta zadovoljavajuća ili ne vrlo je teško empirijski definirati te je zato potrebno napraviti mnogo testova i subjektivnih provjera

Načini optimizacije algoritama

- Neke osnovne grupe optimizacija:
  - Smanjenje preciznosti izračuna
  - Razvoj ekvivalentnih matematičkih algoritama sa manjom kompleksnosti
  - Promjena programske razvojne okoline
  - Promjena programske izvedbene okoline
  - Promjena arhitekture sustava za izvođenje

JPEG

- RGB2YCbCr:
  - Definitivno velika količina podataka
  - Osnovna metoda: 9*, 8+ po pikselu
  - (1280x1240) ≈ 10^7 množenja, 10^7 zbrajanja
- Vidjeli smo jednu metodu za smanjenje kompleksnosti ove konverzije

JPEG

- 2D-DCT, 2D-IDCT

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) cos(((2x+1)u\pi)/2N) cos(((2y+1)v\pi)/2N)$$

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u)\alpha(v) C(u,v) cos(((2x+1)u\pi)/2N) cos(((2y+1)v\pi)/2N)$$

- pokušajmo proučiti koliko je to operacija po 2D-DCT elementu...

JPEG - DCT

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) cos(((2x+1)u\pi)/2N) cos(((2y+1)v\pi)/2N) \qquad \alpha(u) = \begin{cases} \sqrt{(1/N)} & \text{for } u = 0 \\ \sqrt{(2/N)} & \text{for } u = 1, 2, ..., N-1 \end{cases}$$

- Za svaki element:
  - α()α() ΣΣ f*cos()*cos()
  - Teoretski 66 množenja, 63 zbrajanja, 128 izračuna cos() funkcije (koja samo u parametrima ima 1 dijeljenje, 3 množenja, 1 zbrajanje)
  - Pokušajte ovo izračunati u bilo kojem programskom jeziku.....

JPEG - DCT

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) cos(((2x+1)u\pi)/2N) cos(((2y+1)v\pi)/2N) \qquad \alpha(u) = \begin{cases} \sqrt{(1/N)} & \text{for } u = 0 \\ \sqrt{(2/N)} & \text{for } u = 1, 2, ..., N-1 \end{cases}$$

- Primjer približne kompleksnosti računanja po teorijskoj formuli...
- Očito je da ovo ovako nije iskoristivo ni za kakve primjene....

JPEG - DCT

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) cos(((2x+1)u\pi)/2N) cos(((2y+1)v\pi)/2N) \qquad \alpha(u) = \begin{cases} \sqrt{(1/N)} & \text{for } u = 0 \\ \sqrt{(2/N)} & \text{for } u = 1, 2, ..., N-1 \end{cases}$$

- Postoji mnogo različitih matematičkih i računalnih metoda kako efikasno izračunati ovakvu funkciju
- Pogledati ćemo neke najvažnjije
- Odvojivost (separability), lookup tablice,...

JPEG – 2D-DCT odvojivost

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) cos(((2x+1)u\pi)/2N) cos(((2y+1)v\pi)/2N) \qquad \alpha(u) = \begin{cases} \sqrt{(1/N)} & \text{for } u = 0 \\ \sqrt{(2/N)} & \text{for } u = 1, 2, ..., N-1 \end{cases}$$

- Jedna od najvažnijih karakteristika 2D DCT je odvojivost (engl. separability).
- Koristeći tu karakteristiku, rezultat transformacije moguće je izračunati preko 1D transformacije nad svim redovima nakon čega slijedi 1D transformacija nad svim stupcima.
- Kao rezultat ovoga, u literaturi se mogu pronaći dva pristupa rješavanju 2-D DCT: "potpunim" 2D postupkom ili korištenjem dva 1D postupka
- 2D pristup: nešto efikasniji, znatno kompleksniji

1D DCT

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) cos(((2x+1)u\pi)/2N)$$

- Ako pogledamo kompleksnost vidimo da je približna kompleksnost ove formule za svaki element:
  - α() Σ f*cos()
  - Teoretski 9 množenja, 7 zbrajanja, 8 izračuna cos() funkcije (koja u parametrima ima 1 dijeljenje, 3 množenja, 1 zbrajanje)

2D DCT preko 1D DCT

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) cos(((2x+1)u\pi)/2N)$$

- Priličan problem je cos() funkcija
- S obzirom na diskretizirane vrijednosti parametara cos() faktori se NE računaju već se koriste unaprijed izračunate vrijednosti pohranjene u tablicu (tzv.lookup tablica) te se kao takvi oni obično i kombiniraju sa α() parametrom.
- Samo ovim postupkom ZNAČAJNO smo smanjili kompleksnost računanja
- No i sa tako izvedenim cos() kompleksnost je velika

2D DCT

- 2D DCT za blok 8x8:
  - 4096 množenja, 4032 zbrajanja
- Samo za jednu sliku 1280x1024
  - 83886080 množenja, 82575360 zbrajanja

2D DCT preko 1D DCT

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) cos(((2x+1)u\pi)/2N)$$

Koristeći svojstvo odvojivosti

- 1D DCT za 8 podataka
  - 64 množenja, 56 zbrajanja
- 2D DCT preko 1D DCT
  - 1024 množenja, 896 zbrajanja
- Samo za jednu sliku 1280x1024
  - 20971520 množenja, 18350080 zbrajanja

2D DCT

- No i ovo što smo do sada predložili nije dovoljno da u stvarnom svijetu izvedete neki algoritam za kompresiju uz razumnu potrošnju resursa (procesora, vremena, energije,...)

- Potreban je novi pristup računanju....
- BRZI ALGORITMI

Simetrije, tablice,..., i još ...

- Kad razmatramo kako implementirati neki algoritam moramo pokušati proučiti problem sa mnogih strana
- U slučaju DCT analizom formula za 1D DCT mogu se uočiti neke simetrije članova koji se računaju te uz malo trigonometrije reducirati broj izračuna cos() funkcije
- Isto tako mogu se uočiti neki parovi elemenata koji se zbrajaju i oduzimaju

...

- Na taj način mogu se identificirati elementi:
  - C_k
  - S_k
  - s_{jk} i d_{jk}

$$C_k = \cos(k\pi/16)$$
$$S_k = \sin(k\pi/16)$$

$$C_1 = S_7 = 0.9808$$
$$C_2 = S_6 = 0.9239$$
$$C_3 = S_5 = 0.8315$$
$$C_4 = S_4 = 0.7071$$
$$C_5 = S_3 = 0.5556$$
$$C_6 = S_2 = 0.3827$$
$$C_7 = S_1 = 0.1951$$

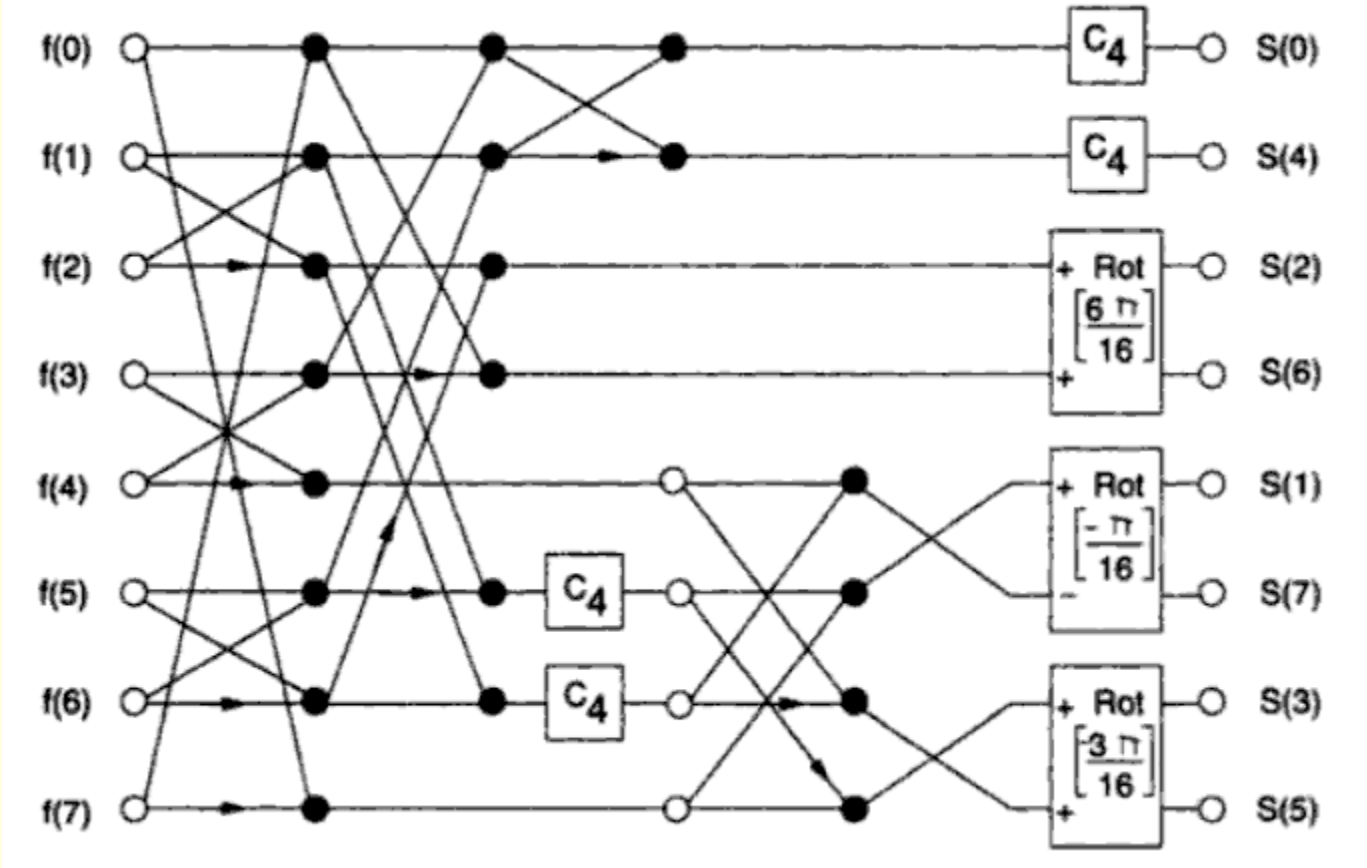
$$s_{jk} = s(j) + s(k)$$
$$s_{07} = s(0) + s(7)$$
$$s_{16} = s(1) + s(6)$$
$$s_{25} = s(2) + s(5)$$
$$s_{34} = s(3) + s(4)$$
$$s_{0734} = s_{07} + s_{34}$$
$$s_{1625} = s_{16} + s_{25}$$

$$d_{jk} = s(j) - s(k)$$
$$d_{07} = s(0) - s(7)$$
$$d_{16} = s(1) - s(6)$$
$$d_{25} = s(2) - s(5)$$
$$d_{34} = s(3) - s(4)$$
$$d_{0734} = s_{07} - s_{34}$$
$$d_{1625} = s_{16} - s_{25}$$

Ligtenberg, Vetterli

- Na temelju prethodnih analiza Ligtenberg i Vetterli su izveli formule za 1D DCT:
  - 2S(0)=C4((s07+s12)+(s34+s56))
  - 2S(1)=C1d07 + C3d16 + C5d25 + C7d34
  - ...
- Također se može vidjeti da ako dva podatka x i y želimo rotirati za kut kπ/16 tada nove vrijednosti iznose:
  - X=Ck*x+Sk*y
  - Y= -Sk*x+Ck*y

Ligtenberg, Vetterli



Ligtenberg, Vetterli

- Kao rezultat ovog razmatranja Ligtenberg i Veterli su izveli algoritam koji zahtjeva samo 13 operacija množenja i 29 operacija zbrajanja za računanje 1-D DCT.

- Već ovaj algoritam pokazuje drastično smanjenje broja matematičkih operacija u usporedbi s konzervativnim načinom računanja.

Brzi DFT....

- Bitno drugačiji pristup računanju DCT predložili su neki autori koji su se intenzivno bavili i proučavanjem brzih algoritama za računanje diskretne Fourierove transformacije (DFT). Tako je u svom radu Haralick pokazao da se DCT s N točaka može izračunati koristeći dvije brze Fourierove transformacije (FFT) s N točaka koristeći se simetrijom ulaza.

- Kroz brojne članke (npr. Tseng i Miller) pokazalo se kako se DCT može efikasnije izračunati na način da se, uz simetrično raspoređene ulaze, koriste samo realni dijelovi prvih N koeficijenata iz DFT s 2N točaka.

Skalirana DCT

- Ovi radovi pored toga uvode dodatnu prednost:
  - Naime kod konzervativnog pristupa ili kod primjera Lightenbergovog i Veterlijevog algoritma, kvantizacija koeficijenata koja slijedi transformaciju uvodi dodatne matematičke operacije u postupak.
  - Algoritam Tsenga i Millera naziva se tzv. Skaliranim algoritmom jer za dobivanje DCT koeficijenata rezultati DFT se moraju pomnožiti s određenim konstantama (skalirati).
  - Ako nakon postupka transformacije odmah slijedi kvantizacija tada se kvantizacijski koraci mogu prije pomnožiti s konstantama skaliranja.Na taj način na izlazu će se bez dodatnih operacija dobiti kvantizirani DCT koeficijenti. Koristeći se ovim načinom razmišljanja, DCT s 8 točaka koja se koristi u JPEG normi može biti zamijenjena DFT-om s 16 točaka i operacijom skaliranja.

Brzi algoritmi – što se koristi

- Tipično se za izračun 1D DCT preko 1D FFT koristi teorija:
  - AAN algoritam za skalirani 1D DCT (8 točaka):
    - 5 množenja, 29 zbrajanja, 16 dvojnih komplementa
  - Kovač, Ranganathan algoritam za skalirani 1D DCT (8 točaka):
    - 5 množenja, 29 zbrajanja, 12 dvojnih komplementa

KR algoritam

Korak 1:

b0 = a0 + a7; b1 = a1 + a6; b2 = a3 - a4; b3 = a1 - a6;

b4 = a2 + a5; b5 = a3 + a4; b6 = a2 - a5; b7 = a0 - a7;

Korak 2:

c0 = b0 + b5; c1 = b1 - b4; c2 = b2 + b6; c3 = b1 + b4;

c4 = b0 - b5; c5 = b3 + b7; c6 = b3 + b6; c7 = b7;

Korak 3:

d0 = c0 + c3; d1 = c0 - c3; d2 = c2; d3 = c1 + c4;

d4 = c2 - c5; d5 = c4; d6 = c5; d7 = c6; d8 = c7;

Korak 4:

e0 = d0; e1 = d1; e2 = m3 * d2; e3 = m1 * d7;

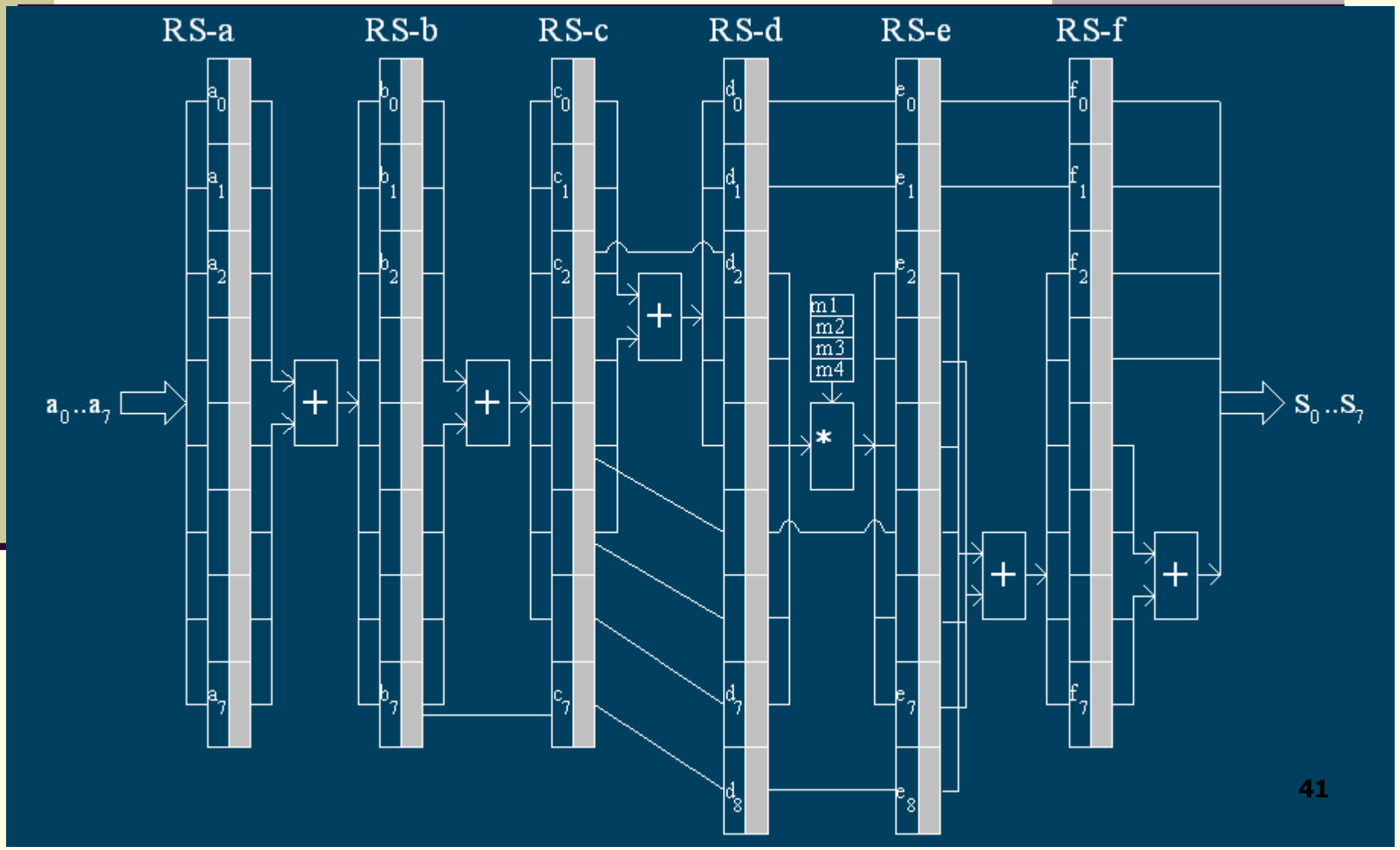
e4 = m4 * d6 e5 = d5; e6 = m1 * d3; e7 = m2 * d4; e8 = d8;

..... I tako dalje

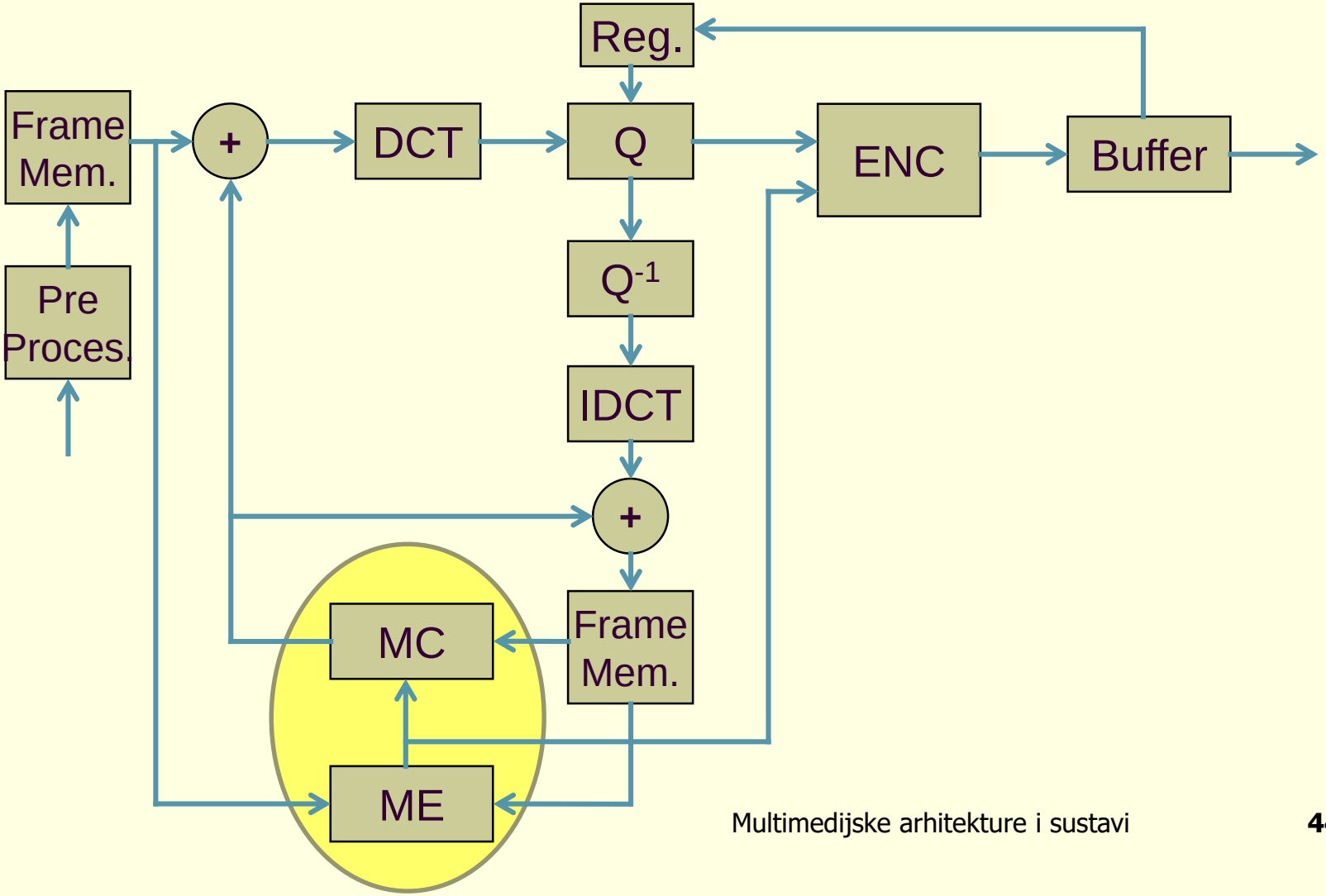
Uštede ....

- Korištenjem ovih brzih algoritama za jednu sliku 1280x1024 potrebno je približno:
  - 1638400 množenja
  - 9502720 zbrajanja
  - 3932160 dvojnih komplementa
- Najvažnije je da je broj operacija množenja (SPORO u procesoru) smanjen cca. 15 puta

I što nam to još omogućuje... (o tome će riječi biti kasnije)



MPEG koder


Reg.
Frame Mem.
+
DCT
Q
ENC
Buffer
Pre Proces.
Q^{-1}
IDCT
+
MC
Frame Mem.
ME

Intra i inter-blokovska kompresija

- **Unutar-blokovska** kompresija
  - Uklanjanje informacija visokih frekvencija koje ljudsko oko ne može prepoznati
  - Isto kao kod kompresije slika (JPEG) što je prije objašnjeno
- **Među-blokovska** kompresija
  - Smanjivanje vremenske redundancije
- Obje se metode zasnivaju na karakteristikama ljudskog vizualnog sustava (HVS)

Međublokovska kompresija

- Često se pojedini dijelovi slike unutar niza vrlo malo mijenjaju ili su čak nepromijenjeni
- Pozadina slike često se ne mijenja

Procjena i kompenzacija pokreta

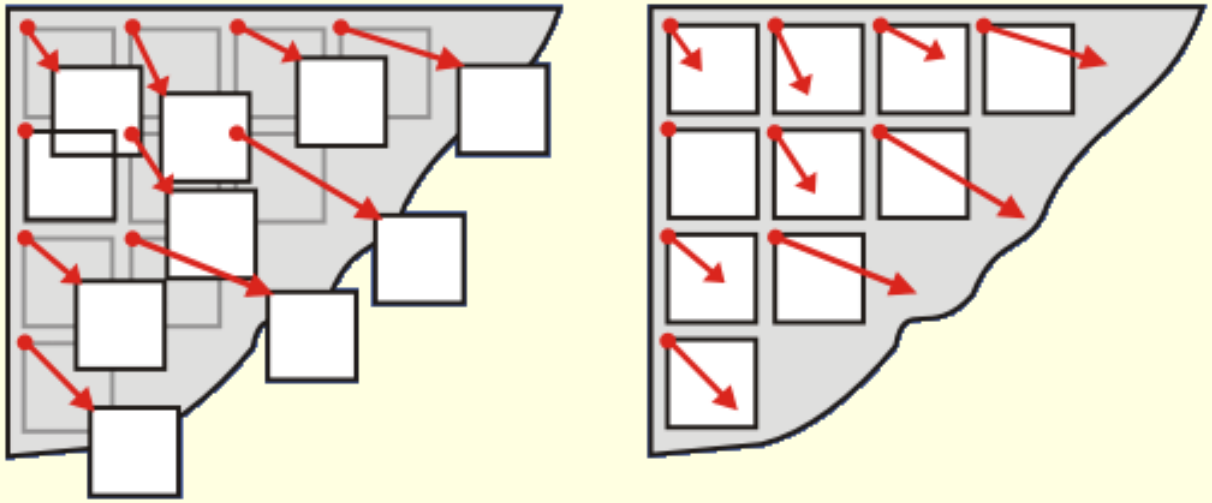
- **Procjena pokreta (motion estimation, ME):**
  - Izdvajanje dijelova slike (segmentacija pokreta)
  - Opisivanje pokreta (procjena)
  - Izvodi se u koderu
- **Kompenzacija pokreta (motion compensation, MC):**
  - Korištenje rezultata procjene pokreta
  - Izvodi se u dekoderu

Procjena i kompenzacija pokreta

- Kako bi omogućili jednostavniju implementaciju nameću se potrebna pojednostavljenja:
  - Segmentacija na pravokutne dijelove unaprijed znanih dimenzija (blok)
  - Svaki je pokret translacijski (vektor pomaka)

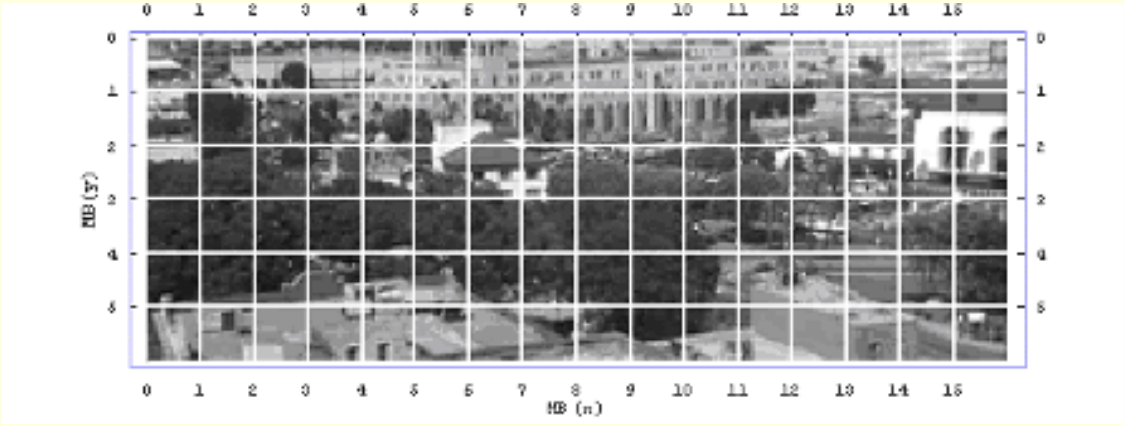
Procjena i kompenzacija pokreta

- Algoritmi procjene pokreta za svaki blok računaju pripadni **vektor pomaka**
- Vektor pokazuje izvorište bloka u prethodnoj slici prije pomaka na poziciju u trenutnoj slici



Slika kao pojedinačni objekt pri kompresiji

- Radi efikasnije obrade za procjenu pokreta dovoljna je **komponenta Y**



Mjera poremećaja

- Mjera sličnosti dvaju blokova jest **mjera poremećaja** slike od jednog trenutka do drugog
- Procjena pokreta svodi se na **optimizaciju** mjere poremećaja kao kriterijske funkcije
- Algoritmi u definiranom **području pretraživanja** nastoje pronaći minimum kriterijske funkcije
  - Promjena slike na tom je mjestu najmanja
  - proglašavamo da se blok pomakao duž vektora

Mjera poremećaja – primjer

- Neke moguće mjere poremećaja dvaju blokova:
  - Srednja kvadratna pogreška (MSE) – računski zahtjevna

$$MSE(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left[Y(x+i,y+j,t_1)-Y(x+d_x+i,y+d_y+j,t_0)\right]^2$$

  - Srednji apsolutni poremećaj (MAD) – široko prihvaćena

$$MAD(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left|Y(x+i,y+j,t_1)-Y(x+d_x+i,y+d_y+j,t_0)\right|$$

  - Broj podudarajućih slikovnih elemenata (MPC)

$$MPC(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}T\left[Y(x+i,y+j,t_1),Y(x+d_x+i,y+d_y+j,t_0)\right]$$

$$T(\alpha,\beta)=\begin{cases}1, & \text{za } |\alpha-\beta|\le T_0\\ 0, & \text{inace}\end{cases}$$

Algoritmi

- Ovisno o načinu pretraživanja razlikujemo:
  - **Algoritam potpunog pretraživanja**
    - pretražuje sve točke unutar područja pretraživanja
    - računski vrlo zahtjevan
    - Daje najbolji mogući rezultat
- **Nepotpuni (brzi, napredni) algoritmi**

Algoritam potpunog pretraživanja

- Veoma zahtjevan algoritam za računalne resurse:
  - Izračun jedne vrijednosti poremećaja (MAD):
    - Dohvat podataka (vrlo zahtjevno)
    - Za blok 16x16: 256 oduzimanja + 1 pomak (dijeljenje sa 256)
- Ako područje pomaka iznosi +-8 u obje dimenzije:
  - (2*8+1)*(2*8+1)*(256+dohvat)=73984 + dohvati
- Pronalaženje minimuma
  - Za jednu sliku 1280x1024
  - 5120*73984=378.798.080 + dohvati
- Za 30 slika u sekundi: $1,1*10^{10}$ zbrajanja/s

Algoritam potpunog pretraživanja

- Vidljivo je da algoritam potpunog pretraživanja daje optimalni rezultat ali je nažalost u praksi teško primjenjiv
- Koriste ga uglavnom samo HW koderi
- Upravo zato potreba za brzim algoritmima
  - Prednosti: manje potrebnih operacija
  - Nedostaci: veće razlike koje se trebaju kodirati a time je i izlazna količina podataka veća
  - Kvaliteta nekih algoritama zadovoljavajuća te su blizupotpunom pretraživanju
  - Problemi sa tipovima pokreta (objašnjeno kasnije)

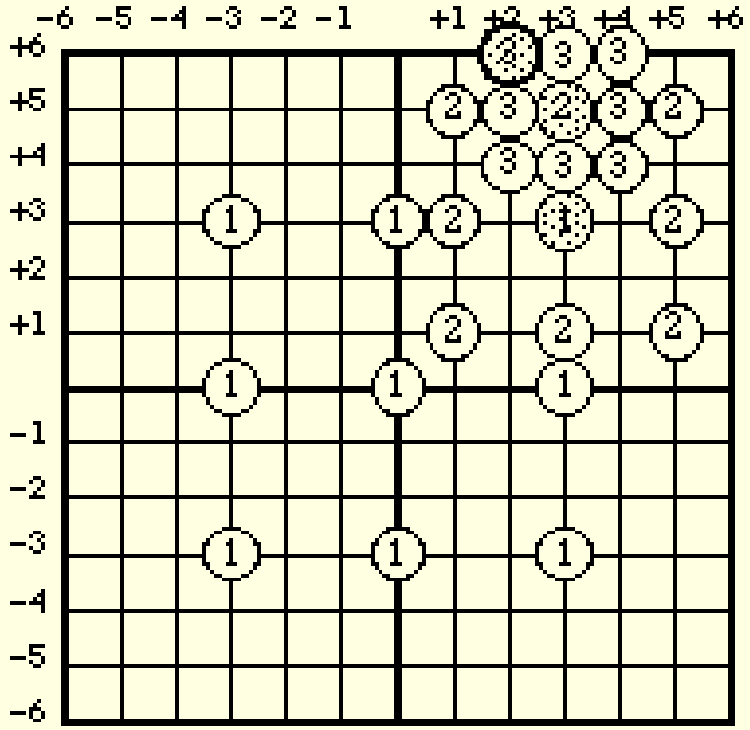
Brzi algoritmi

- **Brzi, napredni algoritmi pretraživanja** usmjeravaju pretraživanje ovisno o vrijednosti poremećaja slike

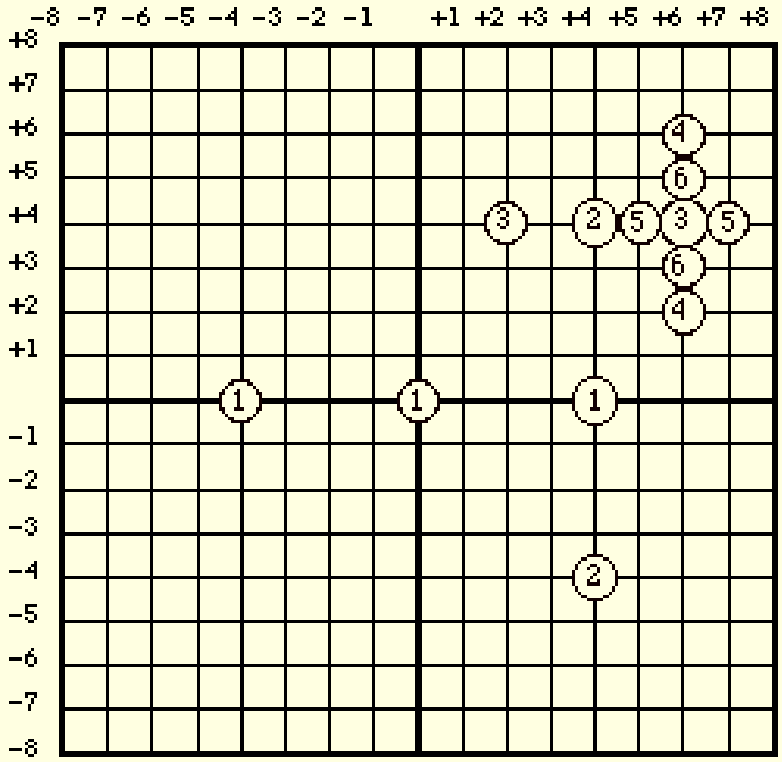
Neki primjeri:

- Logaritamsko pretraživanje (LOG)
- Pretraživanje u tri koraka (3SS)
- Ortogonalno pretraživanje (ORT)
- Algoritam gradijentnog spusta temeljen na blokovima (BBGDS)

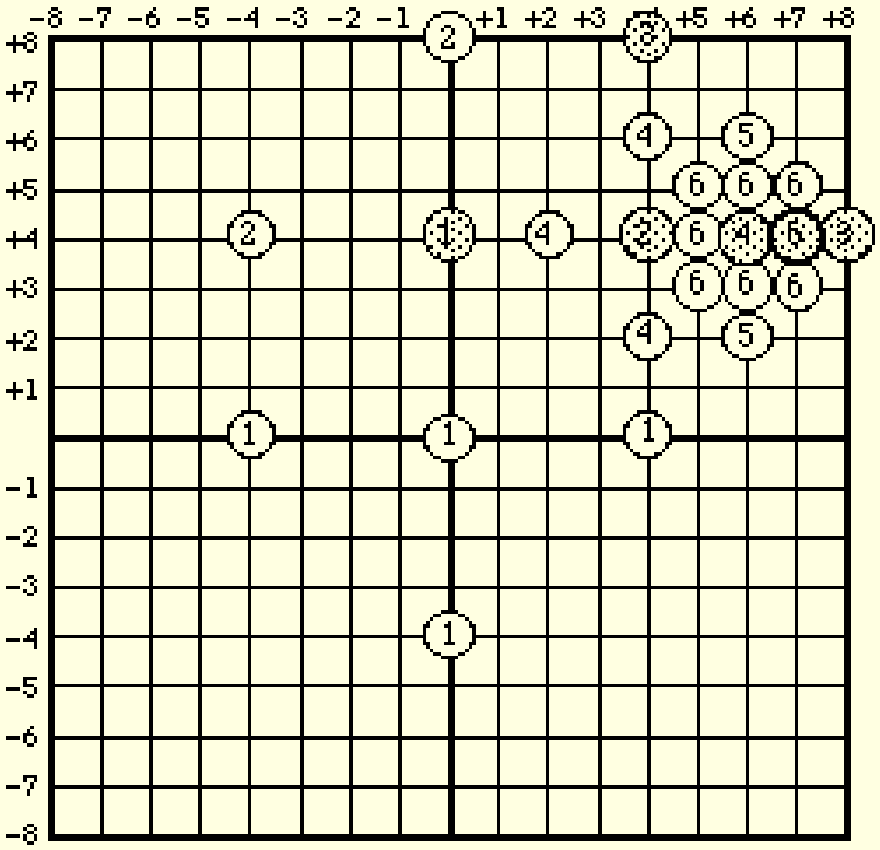
Algoritmi – primjer (3SS)



Algoritmi – primjer (ORT)

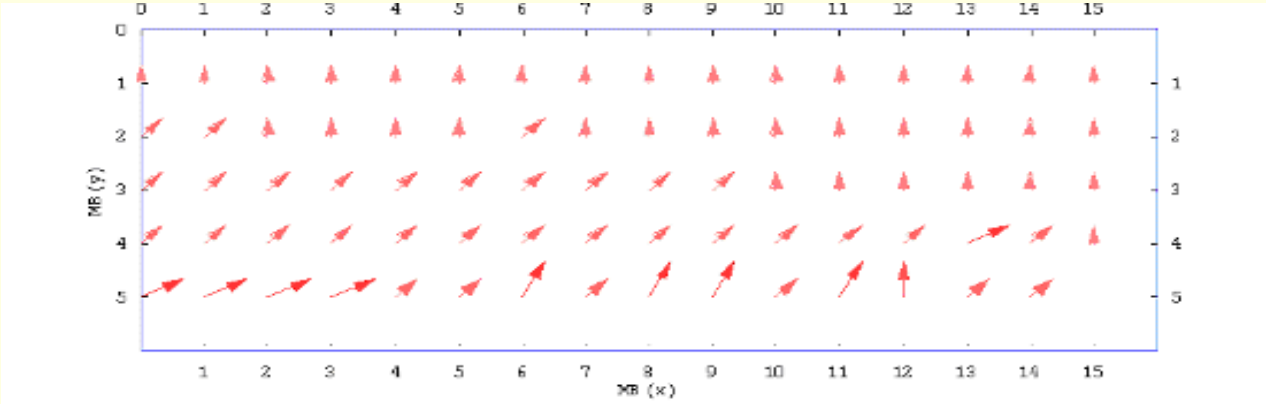


Algoritmi – primjer (LOG)



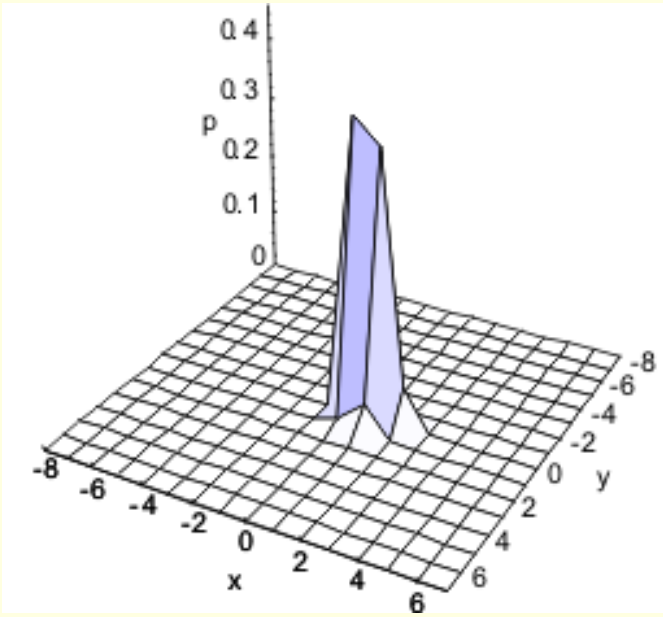
Raspodjela vektora pomaka

- Prikaz upućuje na ravnomjerno raspoređenepokrete između dvije slike sekvence
- Vektori su većinom kratki – centrirana raspodjela vektora - tipična za sekvence iz stvarnog svijeta



Raspodjela vektora (2)

- Prebrojavanje vektora za sve moguće parove komponenti (x,y) tijekom cijele sekvence daje relativne frekvencije vektora
- Učinkovitost algoritma moguće je povećati ako se u obzir uzme uočena centrirana raspodjela vjerojatnosti vektora
- središtu područja pretraživanja treba posvetiti više pažnje



Primjeri usporedbe algoritama

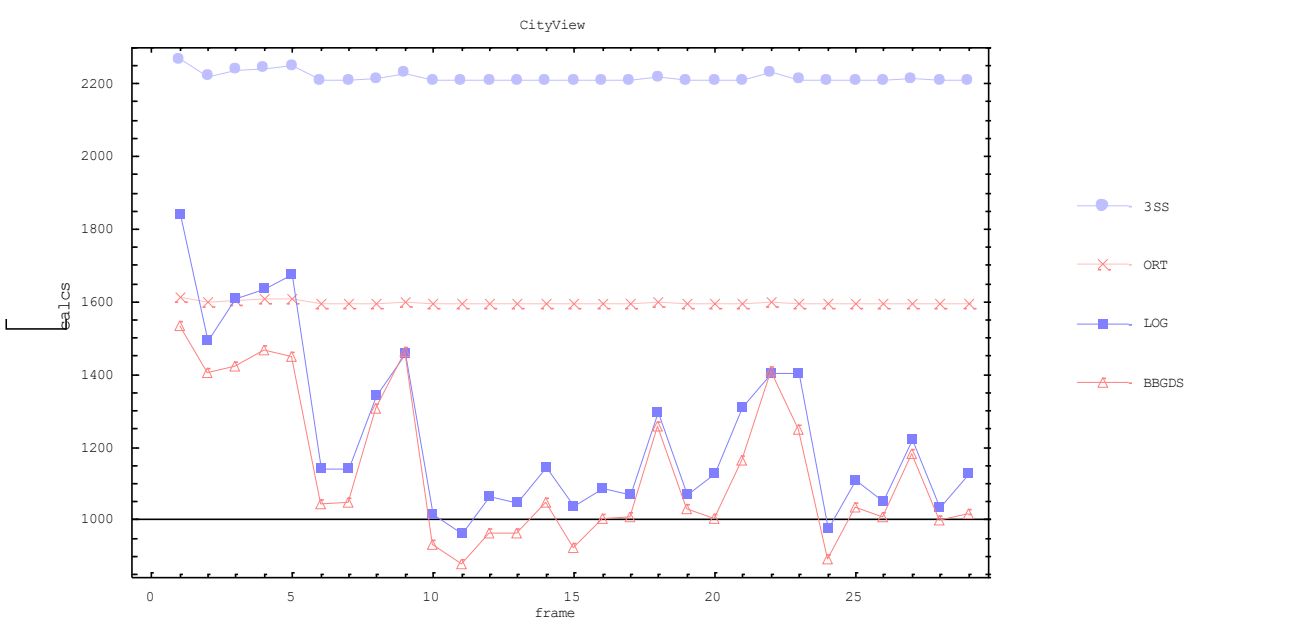
- Parametri koji određuju učinkovitost algoritma:
  - Ukupna **mjera poremećaja** nakon procjene pokreta treba biti što manja (bolja kompresija)
  - Ukupni broj **ispitnih točaka** također treba biti što manji (veća brzina)
- Potreban je kompromis **kompresija ↔ brzina**
- Primjer sekvenci koje sadrže različite vrste pokreta:
  - *"City View"* - ujednačeno raspoređen i malen pokret
  - *"Car Jump"* - lociran i velik pokret
  - *"Troops"* - ujednačeno raspoređen i velik pokret

Implementacija – primjer

- Norma za kompresiju **ne specificira** postupke procjene pokreta
- Kvaliteta aplikacije znatno ovisi o načinu procjene pokreta

Rezultati (1)

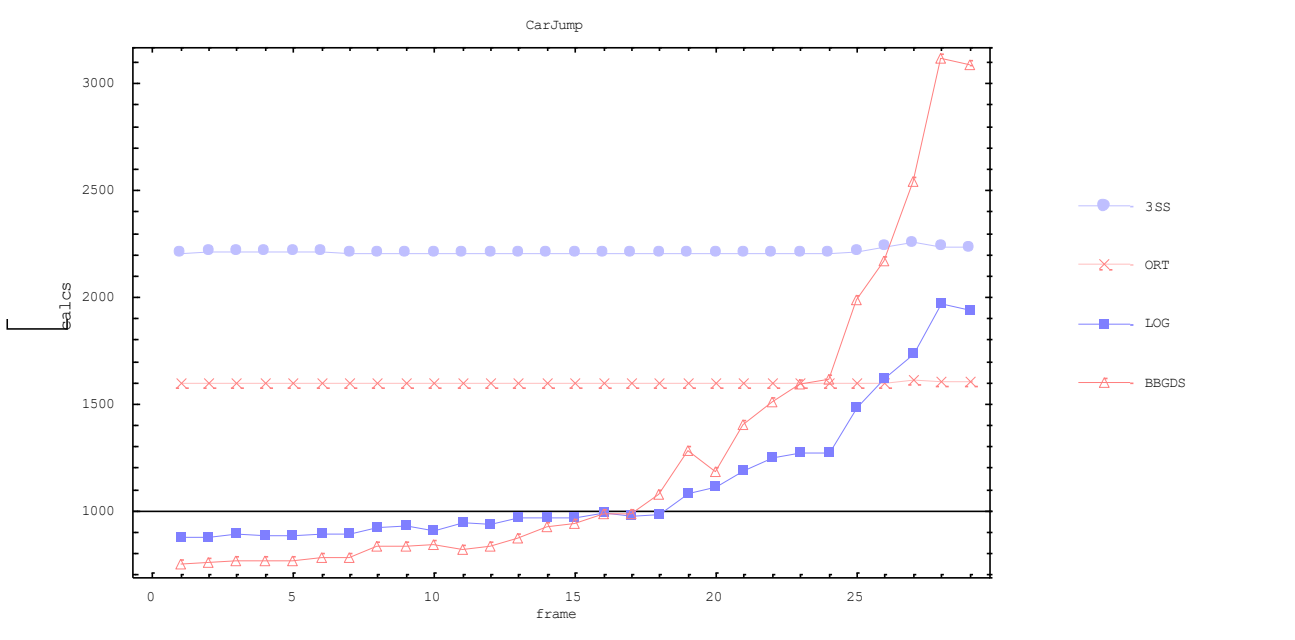
- “City View” sekvenca – broj ispitnih točaka:



- Centrirana raspodjela: BBGDS najbolji
- ORT i 3SS konstantan i relativno velik broj ispitnih točaka

Rezultati (2)

- “Car Jump” sekvenca – broj ispitnih točaka:



- Lokalizirana rastuća promjena: BBGDS i LOG daju slabije rezultate kako poremećaj raste
- ORT bolji od 3SS, konstantni unatoč poremećaju

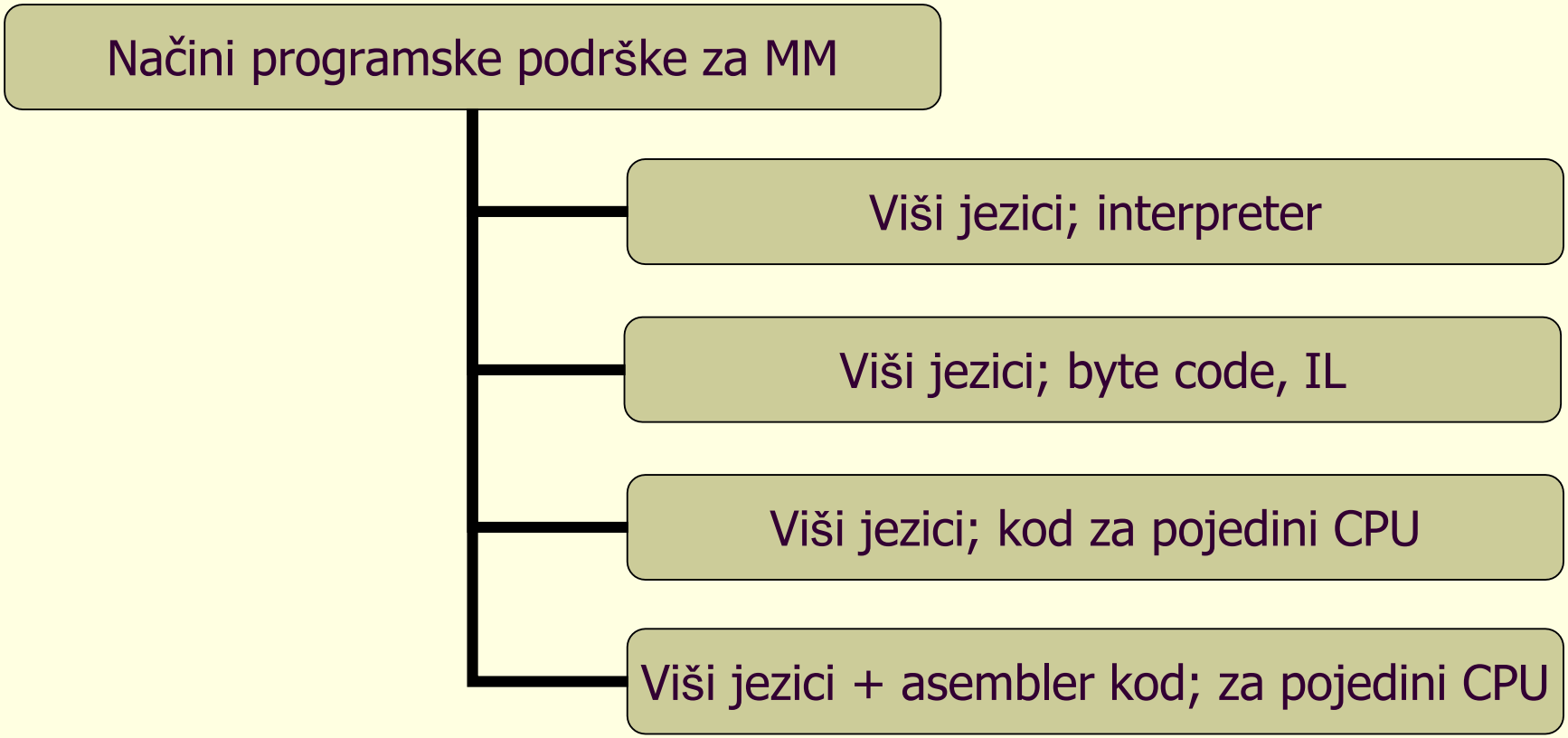
Rezultati (3)

- Učinkovitost pojedinog algoritma ovisi o vrsti sadržanog pokreta
- Učinkovit postupak će na temelju vrste pokreta heuristički odabrati najprikladniji algoritam: **adaptivni algoritmi**

Optimizacije

- Neke osnovne grupe optimizacija:
  - Smanjenje preciznosti izračuna
  - Razvoj ekvivalentnih matematičkih algoritama sa manjom kompleksnosti
  - Promjena programske razvojne okoline
  - Promjena programske izvedbene okoline
  - Promjena arhitekture sustava za izvođenje

Osnovni načini SW podrške za MM



Osnovni načini HW podrške za MM