Sadržaj

Uvod

Cilj ovog rada jeste da objasni osnovne pojmove vezane za video sekvence, te samo neke od brojnih i raznovrsnih algoritama koji se u praksi koriste za video interpolaciju. Objasnit ćemo šta je to tačno video sekvenca, optički tok, interpolacija, framerate, i još mnogo toga. Algoritmi video interpolacije su veoma specijalizovani, te imaju relativno usko podrčje primjene. Međutim, ipak su izuzetno korisni, te ćemo se sa konkretnim područjima primjene uskoro upoznati. Prije toga bi ipak bilo korisno napraviti kratak uvod u video sekvence, te u to kako ih računari vide.

1.1 Video sekvence

Ljudi svijet oko sebe vide kao sliku koja se kontinuirano mijenja. To jeste, svaki vremenski trenutak je različit od prošlog, koliko god da je blizu. Ljudi i objekti se kreću "glatko", bez potpuno naglih pokreta. Također, slike koje vidimo su prostorno kontinualne, prividno beskonačno detaljne. Računari, s druge strane, moraju slike i video fajlove čuvati u ograničenoj količini memorije, što znači da mora doći do kompromisa. Veća količina detalja zahtijeva više memorije za čuvanje. Svaka slika ima samo konačnu količinu detalja, što znači da se sastoji od konačnog broja piksela. Piksel (od eng. *picture element*) je najmanji djelić slike. Dva najčešća tipa piksela, pri čemu ćemo mi u ovom radu koristiti oba, su 1-komponentni i 3-komponentni (nekada se umjesto termina komponenta koristi termin kanal). Jednokomponentni piksel je djelić slike koji je jednobojan. 1-komponentni pikseli su gotovo isključivo monohromatski, tj. njihova boja može biti crna, bijela, ili neka nijansa sive. 3-komponentni pikseli, s druge strane, se sastoje od 3 jednobojne komponente, koje zovemo podpikseli ili subpikseli. Ako je piksel oblika kvadrata, onda su podpikseli oblika pravougaonika (koji su po pravilu izduženi vertikalno). Na svim modernim ekranima, boje podpiksela su crvena, zelena i

plava. Ovaj raspored ćemo ubuduće zvati RGB (od eng. *Red-Green-Blue*). Drugi tipovi podpiksela također postoje, koji se moraju konvertovati u RGB za prikaz na ekranima. Svaki ekran i svaka slika se sastoje od određenog broja piksela, koji su po pravilu raspoređeni u matricu. Ako na monitoru gledamo sliku koja ima više horizontalnih ili vertikalnih piksela od našeg monitora, onda ne možemo vidjeti sve detalje te slike. Koji detalji će biti izostavljeni zavisi od konkretnog algoritma korištenog za skaliranje slike, o kojima nećemo pričati u ovom radu.

Objasnili smo kako računari rješavaju problem prostorno kontinualnih slika, sada ćemo objasniti video sekvence. Isto kao što smo sliku rastavili u piksele, video ćemo rastaviti u *frejmove*. Frejmovi su pojedinačne sličice (ili okvirovi) od kojih se sastoji video sekvenca (sekvenca upravo zato što je to sekvenca frejmova). Svaki frejm provede određeno vrijeme na ekranu, nakon čega se mijenja narednim frejmom. U pravilu, to vrijeme je za svaki frejm isto (ako je to moguće), da bi video sekvenca koju gledamo izgledala što prirodnije. Broj frejmova koje vidimo svake sekunde (odnosno broj recipročan broju sekundi koji svaki frejm provede na ekranu) se zove *framerate*, a korištena jedinica je *frame per second*, skraćeno *fps*. Tako je maksimalni framerate većine ekrana 60 fps, framerate filmova je gotovo uvijek 24 fps, framerate igrica zavisi od mnogo faktora, ali je u većini slučajeva 30 ili 60 fps.

1.2 Interpolacija u video sekvenci

Sada kada znamo šta je framerate, možemo objasniti video interpolaciju. Naime, interpolacijom video sekvence se dobivaju novi frejmovi između postojećih. Taj novi frejm će biti veoma sličan svojim susjednim, jer u suštini predstavlja "među korak". Izlaz algoritma za video interpolaciju će biti video veoma sličan početnom, sa dodatnim frejmovima koji su generisani iz originalnih. U matematičkom smislu, interpolacija predstavlja izračunavanje novih vrijednosti na osnovu okolnih poznatih, što zvuči veoma slično video interpolaciji. Međutim, zbog svoje prirode, algoritmi interpolacije video sekvenci se potpuno razlikuju od onih korištenih u matematici.

1.3 Primjene video interpolacije

Kao što smo već rekli, izlaz algoritma za video interpolaciju jeste novi video koji je veoma sličan originalnom. Ako želimo, možemo koristiti taj video, koji će zbog svog višeg framerate-a izgledati prirodnije (pod uslovnom da je kvalitetno interpoliran) i glađe. Tako bi sportski sadržaj mogao biti sniman kamerom visoke rezolucije, ali relativno niskog fremerate-a, nakon čega bi frejmovi mogli biti interpolirani da bismo dobili video koji je i visoke kvalitete i visokog framerate-a. Također bismo mogli koristiti video interpolaciju za video pozive, gdje je mrežna propusnost često ograničavajući faktor. Možemo snimiti video niskog framerate-a, koji ne zahtijeva visoku mrežnu propusnost, te interpolirati dolazeći video. Naravno, u oba ova slučaja će dobiveni rezultat biti nižeg kvaliteta nego video sniman visokim framerate-om, ali uz kvalitetne algoritme razlika neće biti značajna.

Postoje već projekti koji koriste video interpolaciju za kreiranje *slow-motion* video sekvenci. Umjesto zadržavanje ukupne dužine video sekvencei povećavanja framerate-a, možemo povećati dužinu i zadržati framerate. Jedan primjer bi bio Butterflow.

Osnovne tehnike

Sada ćemo objasniti "trivijalne" algoritme interpolacije: duplikacija frejmova i linearna interpolacija. Linearna interploacija bi se čak mogla koristiti samostalno za kreiranje interpoliranih frejmova, međutim, rezultat bi bio izuzetno mutan. U narednim odjeljcima ćemo objasniti načine rada i područja primjene ovih algoritama.

2.1 Duplikacija frejmova

Kao što samo ime kaže, duplicirani frejm je identičan jednom od njegovih susjednih frejmova (prethodnom ili narednom). Duplicirani frejmovi se koriste u situacijama kada nema smisla koristiti standardne tehnike interpolacije. Naime, neki frejmovi se jednostavno potpuno razlikuju od svojih prethodnika. To se dešava pri promjeni scene ili bilo kakvoj nagloj promjeni slike. Frejm koji se značajno razlikuje od svog prethodnika nazivamo *keyframe*. Ako ćemo između svaka 2 susjedna frejma ubacivati određeni broj interpoliranih frejmova, onda moramo ubaciti nešto i prije keyframe-a. Jednostavno ćemo napraviti onoliko kopija posljednjeg frejma prethodne scene ili novog keyframe-a koliko nam treba interpoliranih frejmova. Ako pokušamo ipak interpolirati novi frejm između dva potpuno različita, dobit ćemo nešto što neće ličiti ni na jedan od dva frejma, te će biti veoma uočljivo gledaocima. Zbog toga je najbolje samo napraviti kopije. Alternativa tom pristupu bi bilo jednostavno preskočiti te frejmove i ostaviti ih kao susjedne. Nedostatak ovog pristupa je potencijalno stvaranje problema oko sinhronizacije ako uz video sekvencu imamo prateći audio zapis ili tekst prijevoda. Ta "rupa" u frejmovima bi značila da će audio zapis i prijevod kasniti za video zapisom. Tako da ćemo mi ipak koristiti duplikaciju prethodnog frejma.

2.2 Linearna interpolacija

Kao što ćemo vidjeti kasnije, tehnike interpolacije koje ćemo mi koristiti neće kreirati cjelokupan frejm. Postojat će velika područja frejma za koje algoritam jednostavno nije dao nikakve podatke. Da bismo izbjegli veoma očigledna "prazna" područja crnih piksela, moramo koristiti neku drugu tehniku koja će zagarantovano davati podatke o svakom pikselu. A najjednostavnija takva tehnika (osim duplikacije) jeste linearna interpolacija, koja radi na sljedećem principu:

Neka su A i B vektori kolone koji predstavljaju dva piksela (svaki član vektora po jednu od tri komponente), a C je piksel interpoliranog frejma (kojeg želimo generisati). Piksel A je piksel na nekoj poziciji prethodnog, a piksel B je piksel na toj istoj poziciji narednog frejma. Mi sada želimo da vidimo koji će se piksel nalaziti na toj poziciji u interpoliranom frejmu. Realan broj α ($0 < \alpha < 1$) predstavlja relativnu poziciju interpoliranog frejma između rethodnog i narednog. To jeste, ako između ta dva frejma generišemo n novih, a frejm koji želimo generisati je k-ti po redu, tada je:

$$\alpha = \frac{k}{n+1} \tag{2.1}$$

Sada je vrijednost piksela C interpoliranog frejma:

$$C = (1 - \alpha) * A + \alpha * B \tag{2.2}$$

Ako između svaka dva frejma jednostavno želimo generisati jedan novi, tada je $\alpha = \frac{1}{2}$, te će svaka komponenta generisanog piksela biti aritmetička sredina komponenti piksela A i B.

Kao što ćemo kasnije vidjeti, pri generisanju novog frejma držat ćemo "matricu posjećenosti", koja će za svaki piksel držati jednu binarnu vrijednost. Sve vrijedosti će na početku biti postavljene na 0 (false). Pri upisivanju novog piksela, odgovarajuću vrijednost matrice ćemo postaviti na 1 (true). Na kraju, piksele na pozicijama sa vrijednošću 0 ćemo generisati na drugi način, konkretno upravo linearnom interpolacijom.

Optički tok

3.1 Uvod

Sada ćemo se početi baviti složenijim tehnikama interpolacije koje mogu davati mnogo bolje rezultate od linearne interpolacije. U idealnom slučaju, rezultirajući frejmovi će izgledati potpuno prirodno. Postoji više klasa algoritama koji su u praksi korišteni, od kojih ćemo mi u ovom poglavlju objasniti tri: Tehnike zasnovane na faznoj korelaciji, upoređivanju blokova, i rješavanju diferencijalnih jednačina. Zajedničko svim ovim algoritmima jeste da njihov izlaz neće biti novi interpolirani frejm, već *polje optičkog toka*. Cilj ovog poglavlja jeste da odgovori na sljedeća pitanja:

1. Šta su vektori pomaka?
2. Šta je optički tok?
3. Kako možemo koristiti poznavanje optičkog toga za kreiranje interpoliranih frejmova?
4. Kojim metodama možemo izračunati optički tok?

Neka su zadana dva susjedna frejma, A i B, između kojih želimo interpolirati novi frejm. Intuitivno gledajući, neki objekti frejma A će se također nalaziti u frejmu B, samo na nekoj drugoj poziciji, blizu svoje originalne. Drugim riječima, postoje vektori pomaka koji odgovaraju tim objektima, koji imaju dvije komponente i dužkojih su objekti prividno pomjereni između frejmova A i B.

Određivanje vektora pomaka za objekte bi zahtjevalo traženje objekata u frejmovima, što je izuzetno zahtjevno samo po sebi. Umjesto toga, svi algoritmi koje ćemo objasniti će pridruživati pojedinačne vektore pomaka svakom pikselu jednog od frejmova (A ili B, neki algoritmi mogu odrediti

koji je od ta dva slučaja korisnije razmatrati). Algoritam će jednostavno dobiti dva frejma, i za svaki piksel dati jedan vektor, njegov vektor pomaka. Skup svih vektora pomaka piksela jednog frejma na drugi nazivamo poljem optičkog toka, pri čemu optički tok možemo definisati kao prividno kretanje objekata u video sekvenci.

Najveći dio posla pri generisanju interpoliranog frejma upravo jeste traženje kvalitetnog optičkog toka. Nakon toga, proces je relativno jednostavan:

1. Svaki piksel pomjeriti duž svog vektora pomaka pomnoženog sa α (realan broj dobiven na isti način kao α u dijelu u linearnoj interpolaciji) te ga spasiti na tu poziciju.

2. Polja za koja nije pronađen niti jedan piksel popuniti korištenjem linearne interpolacije.

3.1.1 Kernel konvolucija

Kernel konvolucija je tehnika često korištena za obradu slike, i ima veliki broj različitih primjena. Neke od njih uključuju detekciju ivica, izoštravanje, zamućivanje i druge razne filtere koji mogu biti primijenjeni na slike. Ulazi kernel konvolucije su slika i sam kernel, dok je izlaz obrađena slika. Kerneli funkcionišu na samo jednom dvodimenzionalnom signalu, tako da ćemo mi posmatrati samo intenzitete pojedinih piksela, ne njihove odvojene komponente (crvena, zelena i plava).

Kernel nije ništa drugo nego matrica realnih brojeva. Za naše potrebe, dimenzije matrice će morati biti neparni brojeva (vidjet ćemo zašto je lakše raditi sa takvim kernelima). Neka je zadana matrica intenziteta piksela A, kernel K i izlazna matrica B. Dimenzije izlazne matrice su iste kao dimenzije ulazne. Neka su dimenzije kernela (visina i širina) p i q. Svaki piksel izlazne matrice$B_{i,j}$ računamo narednom formulom:

$$B_{i,j} = \sum_{y=-p'}^{p'} \sum_{x=-q'}^{q'} A_{i+y,j+x} * K_{y+p'+1,x+q'+1}$$

pri čemu su:

$$p' = (p-1)/2$$
$$q' = (q-1)/2$$

Zamislimo da smo postavili kernel iznad piksela sa koordinatama i, j, tako da element u sredini kernela ima upravo koordinate i, j. Ovdje vidimo zašto smo postavili ograničenje da dimenzije kernela moraju biti neparni brojevi. Zatim ćemo pomnožiti svaki element kernela sa elementom matrice A koji se nalazi tačno ispod njega. Zbir svih tih proizvoda upisujemo u element $B_{i,j}$.

Korištenjem različitih kernela možemo dobiti različite efekte. Ispod slijede primjeri nekih od češće korištenih:

$$\text{Detekcija ivica: } \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\text{Zamućivanje: } \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

$$\text{Izoštravanje: } \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Neki od algoritama koje ćemo objasniti koriste kernel konvoluciju, u opisu algoritama ćemo navesti i konkretne korištene kernele.

3.2 Uparivanje blokova

Prva klasa algoritama koje ćemo proučavati kreću od iste osnovne ideje: Podijeliti prvi frejm na blokove, te naći vektor pomaka svakog bloka iz prvog frejma u drugi. Ovi algoritmi značajno pojednostave problem tako što pretpostave da će svi pikseli unutar jednog bloka imati iste vektore pomaka. Ovaj pristup može dovesti do grešaka, ali čini izračunavanje vektora pomaka jednostavnim i brzim.

Ako nam je cilj samo kreirati jedan novi frejm između svaka dva postojeća, svaki blok ćemo pomjeriti duž pola izračunatog vektora pomaka. Ako nam je cilj interpolirati dva frejma između dva postojeća, blok ćemo pomjeriti duž jedne trećine vektora pomaka za prvi, i dvije trećine za drugi interpolirani frejm, itd. Cilj slijedećih algoritama jeste uparivanje blokova prvog frejma sa blokom iste veličine u drugom frejmu. Međutim, postoji nekoliko pitanja na koja moramo odgovoriti prije nego što možemo primijeniti ove algoritme:

- Koju veličinu bloka ćemo koristiti?
- Koliki će biti prozor pretrage, odnosno koliko će se svaki blok moći maksimalno pomjeriti između prvog i drugog frejma?
- Koji je kriterij sličnosti dva bloka?
- Kako odrediti uspješnost uparivanja?

U praksi se koriste blokovi veličine 16x16 piksela, te prozor pretrage veličine 30x30 piksela. To znači da pretpostavljamo da se između dva susjedna frejma blokovi neće pomjeriti više od 7 piksela u bilo kojem od 4 kardinalna smjera. To nam daje 225 mogućih lokacija za svaki blok. Naravno, ne postoji definitivna, optimalna veličina bloka ili prozora pretrage za sve slučajeve. Manje blokove je brže uporediti, ali je njihov broj veći, te je veća vjerovatnoća da će dva bloka biti slučajno veoma slična. Veći prozor pretrage nam omogućava pronalaženje ispravnih vektora pomaka i u slučaju kada se desi pomak veći od 7 piksela, ali povećava vrijeme potrebno za izračunavanje te, slično kao u slučaju blokova, povećanjem prozora pretrage se povećava i vjerovatnoća uparivanja dva bloka koji su slični, ali zapravo ne predstavljaju isti blok. U svim slijedećim algoritmima ćemo koristiti blokove i prozore pretrage navedene veličine.

Sljedeće pitanje se tiče kriterija sličnosti dva bloka. Svaki blok je sastavljen od 256 piksela, koji se sastoje od 3 komponente: crvene, zelene, i plave. Svaka komponenta ima cjelobrojnu jačinu u rasponu od 0 do 255, uključivo. Za upoređivanje blokova nećemo koristiti sve 3 komponente piksela, nego ćemo izračunati njihov intenzitet, koji se dobija kao aritmetička sredina intenziteta crvene, zelene i plave komponente. Ovo se radi zato što jednostavno ne bismo imali koristi od odvojenog upoređivanja sve 3 komponente. Naravno, postoji vjerovatnoća da ćemo upariti dva piksela koji imaju slične intenzitete, a ne i iste boje (npr. potpuno crven i potpuno zelen piksel imaju iste intenzitete), ali računanjem vektora pomaka za sve 3 komponente zasebno bismo često dobili 3 različita vektora pomaka, što nam ne daju nikakvu korisnu informaciju. Zbog toga je bolje samo uzimati u obzir intenzitete piksela.

Korišteni kriteriji sličnosti blokova su veoma jednostavni. Jedan je *Mean Absolute Difference (MAD)*, odnosno *Srednja Apsolutna Razlika*. Ova mjera nije ništa drugo nego suma apsolutnih vrijednosti razlika intenziteta odgo-

varajućih piksela u blokovima, podijeljena sa veličinom bloka. Drugim riječima, zadana je formulom

$$MAD = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |A_{ij} - B_{ij}|$$

Pri čemu N predstavlja visinu i širinu bloka (u našem slučaju 16), dok A_{ij} i B_{ij} predstavljaju vrijednosti piksela na koordinatama (i, j) (sa početkom u gornjem lijevom uglu bloka) prvog, odnosno drugog razmatranog bloka.

Druga, veoma slična mjera jeste *Mean Squared Error (MSE)*, odnosno *Srednji Kvadrat Greške*. Umjesto uzimanja apsolutne vrijednosti razlika piksela, ova mjera kvadrira razlike piksela, čime se više kažnjavaju veće razlike. *MSE* je zadana formulom

$$MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (A_{ij} - B_{ij})^2$$

Mi ćemo ove funkcije ubuduće zvati zajedničkim imenom *funkcije cijene*. Cilj svih algoritama u ovom poglavlju jeste minimizacija funkcije cijene, bilo *MAD* ili *MSE*.

Još jedno veoma važno područje primjene algoritama uparivanja blokova (i zapravo područje gdje se najviše primjenjuju) jeste kompresija video sekvenci. O ovom poglavlju se obrađuju algoritmi koji su korišteni u standardima H.261, H.262 i H.263. U novijim standardima (pri čemu je najnoviji H.265, čija je prva verzija izašla 2013. godine) korišteni su napredniji algoritmi, koji u ovom radu neće biti obrađeni.

3.2.1 PSNR

Sada ćemo se osvrnuti na neke od osnovnih algoritama uparivanja blokova. Pretpostavit ćemo da su blokovi veličine 16x16 piksela, a prozori pretrage veličine 30x30 piksela, te da su svi pikseli monohromatski (crno-bijeli). Od svih slijedećih algoritama, samo prvi pronalazi optimalno uparivanje. Svi ostali ostali algoritmi su aproksimacije. Preciznije rečeno, postoji 225 mogućih lokacija gdje se počtni blok može nalaziti u prozoru pretrage. Definišimo matricu C_{15x15}, pri čemu svakom elementu matrice C odgovara vrijednost funkcije cijene (*MAD* ili *MSE*) koju dobijemo ako postavimo blok na to

mjesto u prozoru pretrage (drugim riječima, elementu *1,1* odgovara vrijednost funkcije cijene koju dobijemo ako blok postavimo u gornji lijevi ugao prozora pretrage, pomjeranjem bloka dobijamo druge vrijednosti matrice). Aproksimativni algoritmi pretpostavljaju da ova matrica ima jednu najmanju (optimalnu) vrijednost, i da vrijednosti elemenata matrice monotono rastu kako se udaljavamo od ovog elementa. Kontinualni analogon ove osobine bi bila funkcija 2 realne promjenljive koja ima samo jedan lokalni minimum, koji je ujedno i globalni minimum. Za funkciju koja ima ovu osobinu kažemo da je *unimodalna*. U slučaju kad ova osobina postoji, možemo pronaći minimalnu vrijednost matrice jednostavnim spuštanjem po gradijentu, odnosno, počevši od proizvoljnog početnog elementa, u svakom koraku pređemo na bilo koji od manjih elemenata dok ne dođemo do nekog koji je manji od svih svojih susjednih elemenata.

Naravno, ne postoji ništa da nam garantuje ovu osobinu, što znači da će aproksimativni algoritmi samo u rijetkim slučajevima pronaći optimalno rješenje.

3.2.2 Potpuna pretraga

Ovaj algoritam se zasniva na potpunom pretraživanju svih mogućih lokacija za blok, te pronalaženju lokacije koja daje najmanju vrijednost funkcije cijene. Postoji 225 mogućih lokacija za blok unutar prozora pretrage, a za izračunavanje vrijednosti funkcije cijene moramo uporediti $16*16 = 256$ piksela. To nam daje $225*256 = 57600$ upoređivanja piksela po bloku. HD video (dimenzija 1280x720 piksela) sadrži frejmove koji se sastoje od 3600 blokova, što nam ukupno daje preko 200 miliona upoređivanja piksela za svaki frejm video sekvence. Tako da nije teško vidjeti zašto se ovaj algoritam ne koristi u praksi.

3.2.3 Trostepena pretraga

Trostepena pretraga se oslanja na osobinu unimodalnosti matrice vrijednosti funkcije cijene. Ovaj algoritam je ovdje korišten isključivo kao uvod, jer ostvaruje veoma slabe rezultate u praksi. Algoritam je mnogo brži od potpune pretrage te koristi slične tehnike kao bolji algoritmi koji će biti obrađeni u narednim dijelovima.

Neka je zadan prozor pretrage veličine 30x30 piksela i blok veličine 16x16 piksela. Trostepena pretraga radi tako što u svakom koraku izračuna funkciju

cijene bloka na 9 lokacija u prozoru pretrage. Tih 9 lokacija su trenutni centar i sve kombinacije lokacija koje dobijemo ako blok pomjerimo za *-S, 0* ili S piksela po x i y osama (postoji 9 kombinacija, od kojih je jedna *(0, 0)*, što predstavlja naš centar). S predstavlja trenutnu *veličinu koraka*. Nakon izračunavanja funkcije cijene na svih 9 lokacija, izaberemo najbolju (tj. lokaciju koja daje najmanju vrijednost funkcije cijene), nju odredimo za novi centar, i smanjimo S na jednu polovinu trenutne vrijednosti. U prvom koraku, S će imati vrijednost 4, u drugom 2, a u trećem 1, dok će centar u prvom koraku predstavljati stvarni centar prozora pretrage. S obzirom da se na početku blok nalazi 7 piksela daleko od ivica prozora pretrage, a u 3 koraka (sa pomakom od 4, 2 i 1 pikselom) se moguće pomjeriti najviše tačno 7 piksela, blok može zauzeti bilo koju poziciju u prozoru pretrage.

U svakom od 3 koraka, imamo 9 različitih mogućih pozicija bloka, tako da izgleda da moramo izračunati funkciju cijene bloka 27 puta. Međutim, centar u drugom i trećem koraku je već izračunat u prethodnom koraku, tako da je zapravo potrebno izračunati samo 25 različitih funkcija pretrage, što je 9 puta manje od 225 kod potpune pretrage, što naravno predstavlja veoma značajno ubrzanje. Međutim, kao što je prije rečeno, ovaj algoritam generalno ne izračunava zadovoljavajuće vektore pomaka, tako da se u praksi ne koristi.

3.2.4 Četverostepena pretraga

Četverostepena pretraga radi na istom principu kao trostepena, samo sa drugačijim izborom veličine koraka. U prvom koraku S iznosi 2, odnosno lokacije koje pretražujemo će biti $(0,0)$, $(0,2)$, $(2,0)$, $(2,2)$, $(-2,0)$, $(0,-2)$, $(-2,-2)$, $(2,-2)$ i $(-2,2)$. U slučaju da lokacija $(0,0)$ daje najmanju vrijednost funkcije cijene, prelazimo na četvrti korak. Ovo vrijedi i u drugom i trećem koraku, za trenutni centar pretrage. Drugi i treći korak također imaju veličinu koraka $S = 2$, dok za četvrti korak vrijedi $S = 1$. Ukoliko prođemo kroz sva četiri koraka, vidimo da je maksimalni pomak jednak $2+2+2+1 = 7$.

Ovaj pristup zahtijeva od 17 do 27 upoređivanja blokova. Razlog ovome jeste što je veličina koraka u prva 3 koraka ista, što znači da će se značajan broj lokacija poklapati, te ih ne moramo računati ponovno. Iako je manje konsistentna od trostepene pretrage (koja uvijek zahtijeva 25 upoređivanja blokova), četverostepena pretraga će u većini slučajeva biti brža. Druga prednost je što četverostepena pretraga bolje detektuje male vektore pomaka (1 do 2 piksela u jednom smjeru), zato što je početni korak manji ($S = 2$ u

odnosu na $S = 4$), kao i zato što u slučaju da je najbolja lokacija u centru, automatski prelazimo u četvrti korak, koji može samo jos podesiti vektor pomaka za 1 piksel u bilo kojem smjeru.

3.2.5 Dijamantna pretraga

Dijamantna pretraga je slična četverostepenoj pretrazi, uz dvije modifikacije. Prva je da oblik koji prave lokacije koje pretražujemo nije više kvadrat čije su stranice paralelne sa koordinatnim osama, nego kvadrat rotiran za 45°(čiji oblik podsjeća na dijamant ili romb), a druga da ne postoji gornja granica na maksimalan broj koraka. Naravno, algoritam će uvijek terminirati jer postoji 225 mogućih lokacija, a funkcija cijene za svaku lokaciju se računa samo jednom, druga razlika samo znači da algoritam neće automatski preći na manju veličinu koraka nakon nekog broja koraka, nego se to radi samo ako, od 9 lokacija koje razmatramo u trenutnom koraku, najbolji rezultat daje trenutni centar. Do posljednjeg koraka, veličina koraka S iznosi 2, što u prvom koraku daje sljedeće lokacije koje trebamo pretražiti: $(0, 0)$, $(2, 0)$, $(0, 2)$, $(-2, 0)$, $(0, -2)$, $(1, 1)$, $(1, -1)$, $(-1, 1)$ i $(-1, -1)$. U posljednjem koraku, veličina koraka S iznosi 1, što znači da ćemo posmatrati samo trenutni centar i 4 lokacije direktno iznad, ispod, lijevo i desno od trenutnog centra (pomjerene po jedan piksel u kardinalnim smjerovima).

Garanciju da će algoritam eventualno preći na posljednji korak nam daje činjenica da ovaj algoritam (i oba prethodna) uvijek prelazi na lokaciju koja daje bolju vrijednost funkcije cijene (ili ostaje na trenutnoj lokaciji). S obzirom da postoji konačan (225) broj lokacija, a nikad ne prelazimo na lokaciju koja nam daje veću vrijednost funkcije cijene, eventualno ćemo doći do lokalnog minimuma (koji je u idealnom slučaju i globalni minimum).

3.2.6 Adaptive Rood Pattern Search - ARPS

Algoritam koji daje najbolje rezultate koji ćemo opisati u ovom poglavlju je ARPS. Za razliku od do sada opisanih algoritama, ARPS uzima u obzir ne samo vrijednosti piksela bloka i prozora pretrage, nego i prethodno izračunati vektor pretrage. Konkretno, da bismo koristili ARPS, potreban nam je jedan vektor pomaka za koji očekujemo da će vjerovatno biti sličan stvarnom vektoru pomaka trenutnog bloka. Mi ćemo koristiti vektor pomaka bloka koji se nalazi lijevo od trenutnog u tu svrhu. Ako se trenutni blok nalazi uz lijevu ivicu frejma (i ne postoji blok lijevo od njega), onda ćemo koristiti

vektor pomaka bloka iznad trenutnog. A u krajnjem slučaju, ako računamo vektor pomaka za prvi blok (u gornjem lijevom uglu frejma), možemo koristiti potpunu pretragu koja zagarantovano daje optimalno uparivanje (uticaj na performanse nije značajan jer potpunu pretragu vršimo samo jednom za jedan frejm).

Neka prethodni vektor pomaka iznosi (x, y). Uvest ćemo pomoćnu promjenljivu $h = max(|x|, |y|)$. U prvom koraku ćemo izračunati funkciju cijene na 6 lokacija: (0, 0), (h, 0), (0, h), (-h, 0), (0, -h), (x, y). Odnosno, pretražujemo lokaciju (0, 0) (jer su nula vektori pomaka česti), vektor pomaka od kojeg smo krenuli (x, y), te 4 lokacije koje su isto daleko od (0, 0) po tzv. *Manhattan udaljenosti*. Treba uzeti u obzir da je moguće da je $x = 0$ ili $y = 0$, u kojem slučaju je potrebno pretražiti samo 5 lokacija (2 se poklapaju). Ako je $x = y = 0$, onda je riječ o samo jednoj lokaciji, i možemo odmah preći na drugi korak algoritma.

Drugi korak ARPS algoritma je isti kao drugi korak dijamantne pretrage. Od pretraženih lokacija izaberemo najbolju, zatim izračunavamo 4 lokacije direktno iznad, ispod, lijevo i desno od trenutne najbolje, te prelazimo na najbolju od te 4. Algoritam se završava kada je trenutna lokacija bolje od 4 koje je okružuju. Kao i uvijek, možemo ubrzati algoritam tako što ne računamo opet funkciju cijene za lokacije koje smo već obradili.

3.3 Kros-korelacija

Svi do sada objašnjeni algoritmi se zasnivaju na direktnom upoređivanju piksela. Kros-korelacija, s druge strane, je metoda slična konvoluciji, te također radi, u osnovi, direktno upoređivanje piksela 2 bloka u cilju pronalaženja lokacije prvog (manjeg) bloka unutar drugog (većeg). Inače, treba napomenuti da se radi o diskretnoj kros-korelaciji, jer je skup piksela nad kojim se vrši ova operacija po svojoj prirodi diskretan. Kao što ćemo vidjeti, kros-korelacija u suštini radi potpunu pretragu, te implementirana direktno ima isti problem izuzetno visokog vremena izvršavanja. Međutim, kros-korelaciju je moguće izračunati korištenjem brze Fourierove transformacije (FFT), što će značajno ubrzati pretragu. Kros korelacije se najčešće označava simbolom *. U nastavku ćemo objasniti sam algoritam, a nakon toga će biti govora o njegovim performansama.

Jednodimenzionalna diskretna kros korelacija nizova f i g se računa for-

mulom:

$$(f * g)[n] := \sum_{m=-\infty}^{\infty} f^*[m]g[m+n]$$

3.4 Fazna korelacija

3.5 Diferencijalne metode

Uklanjanje grešaka

Direktnom primjenom algoritama uparivanja blokova, moguće je dobiti interpolirane frejmove. Međutim, ti frejmovi će biti veoma slabe kvalitete. U ovom poglavlju će biti razrađene tehnike pomoću kojih ćemo riješiti sljedeća 3 problema:

1. Postojat će velike "rupe" u interpoliranom frejmu, jer za značajan dio piksela neće biti pronađen niti jedan blok koji ih pokriva. U prvom dijelu će biti objašnjena tehnika koju ćemo koristiti za njihovo popravljanje.

2. Bez obzira na to koji algoritam uparivanja koristimo, za neke blokove će biti generisani neispravni vektori pomaka. U drugom dijelu će biti objašnjene neke tehnike za detekciju i korekciju neispravnih vektora pomaka.

3. U zavisnosti od toga koji algoritam uparivanja koristimo, postoji niža ili viša vjerovatnoća da će granice između blokova u interpoliranom frejmu biti veoma oštre i očigledne. Ova činjenica opet proizlazi iz nesavršenosti algoritama uparivanja. Da bismo učinili interpolirane frejmove realističnijim i ugodnijim za gledati, u trećem dijelu će biti razrađene neke tehnike pomoću kojih je moguće umanjiti ovaj efekat.

4.1 Popunjavanje rupa

4.2 Neispravni vektori pomaka

Korištenjem algoritama uparivanja blokova, dobili smo odgovarajuće vektore pomaka za svaki blok veličine 16x16 piksela. Međutim, postoji visoka vjerovatnoća da neki od tih vektora ne odgovaraju stvarnom pokretu u video

sekvenci. Prvo ćemo definisati šta zapravo znači neispravan vektor pomaka, kako ih detektovati, a zatim kako ih popraviti.

4.2.1 Detekcija neispravnih vektora pomaka

U ovom odjeljku ćemo objasniti generalnu metodu detekcije neispravnih vektora pomaka, pri čemu ćemo izračunavanje određenih konstanti ostaviti za poglavlje vezano za implementaciju (te konstante ćemo dobiti eksperimentalno).

Vektor pomaka za neki blok, bez obzira na korištenu metodu, je dobiven korištenjem *MAD* metode. Međutim, MAD nije efektivan u situacijama u kojima postoji veliki broj kandidata sa istom MAD mjerom. Da bismo preciznije izračunali ispravnost vektora pomaka, koristit ćemo još jednu mjeru, a to je *BAD* (*Boundary Absolute Difference*). Za blok B_1 nekog frejma, i njemu upareni blok B_2 iz narednog frejma, BAD računamo kao zbir apsolutnih vrijednosti razlika piksela na ivici bloka B_1 i njima najbližih piksela koji okružuju blok B_2. Za blok veličine 16x16 piksela, imat ćemo 64 ivična piksela. Radi konzistentnosti, možemo koristiti *SAD* umjesto MAD, odnosno *Sum Absolute Difference*, koji jednostavno predstavlja ukupni zbir apsolutnih vrijednosti razlika piksela, bez dijeljenja sa veličinom bloka. Nakon pronalaženja SAD i BAD mjera za odgovarajući blok, također trebamo odrediti konstante T_1, T_2, T_3 i T_4, pri čemu je $T_3 > T_1$ i $T_4 > T_2$. Ove konstante ćemo koristiti u algoritmu za određivanje kvalitete vektora pomaka, koji glasi:

- Ako je SAD $< T_1$ ILI BAD $< T_2$, vektor pomaka je ispravan
- Ako je SAD $< T_3$ I BAD $< T_4$, vektor pomaka je ispravan
- U suprotnom, vektor pomaka je neispravan

Postoji jošjedna metoda detekcije neispravnih vektora pomaka, koja se izvršava direktno na vektorima, umjesto na blokovima. Ovom metodom možemo otkriti vektore koji se značajno razlikuju od svojih susjednih vektora, što je najčešće indikator neispravnosti. Svaki vektor pomaka se sastoji od x-komponente i y-komponente, te ćemo definisati razliku dva vektora pomaka kao zbir apsolutnih vrijednosti razlika njihovih x i y komponenti, odnosno, razlika dva vektora pomaka mv_1 i mv_2 je $|mv_{1x} - mv_{2x}| + |mv_{1y} - mv_{2y}|$. Ako je za neki vektor pomaka zbir razlika sa njegovih 8 susjednih vektora veći od neke konstante, vektor označimo kao neispravan.

4.2.2 Ispravljanje neispravnih vektora pomaka

Neovisno od toga kako smo odredili neispravnost vektora pomaka, koristit ćemo istu metodu ispravljanja. S obzirom da očekujemo da će susjedni blokovi imati iste vektore pomaka, najlakši način da ispravimo neispravan vektor pomaka jeste da ga zamijenimo jednim od 8 susjednih vektora. Izbor ćemo izvršiti tako što ćemo, od 8 susjednih vektora, uzeti onaj sa minimalnim zbirom razlika od ostalih 7. Odnosno, uzet ćemo onaj vektor pomaka mv_k koji daje najmanji rezultat funkcije $\sum_{i=1}^{8}(|mv_{kx} - mv_{ix}| + |mv_{ky} = mv_{iy|})$, pri čemu su $mv_1, mv_2, \ldots, mv_8$ susjedni vektori pomaka našeg neispravnog vektora.

4.3 Izglađivanje granica između blokova

Implementacija interpolatora korištenjem OpenCV biblioteke i benchmark testovi

Zaključak