Sadržaj

Uvod

1.1 Općenito o video sekvencama

Radi jednostvnosti, mi ćemo imati relativno jednostavan pogled na video sekvence. Fokus rada je na direktnoj manipulaciji nad frejmovima (sličicama) i pikselima radi dobivanja krajnje sekvence, ne u algoritmima dekodiranja i enkodiranja video fajlova, koji mogu biti veoma složeni. Konkretno, mi ćemo na video sekvencu gledati kao običan niz frejmova. Svaki frejm je matrica piksela. Veličina frejma naravno zavisi od veličine konkretnog fajla na kojem radimo, ali za primjere ćemo koristiti frejmove dimenzija 1280x720 piksela (1280 piksela širine i 720 piksela visine). Video sekvenc ove dimenzije se često zovu HD (*High Definition*) video sekvence. Neke druge često korištene dimenzije uključuju 640x360, 1920x1080 (*Full HD*), 2560x1440 (*2K*), 3840x2160(*4K*), itd. HD veličina je također pogodno jer jedan frejm sadrži ~1 milion piksela (tačno 921600), što proračune čija preciznost nije od izrazite važnosti čini lakšim. Postoje posebni frejmovi u video sekvenci koji se zovu *keyframe*-ovi. Keyframe je prvi frejm nove scene, ili se previše značajno razlikuje od prethodnog frejma da bi bilo smisla tražiti sličnosti sa prethodnim frejmom. Ove frejmove je zbog toga potrebno posebno tretirati, što će biti objašnjeno u narednom poglavlju.

Još jedna važna osobina video sekvenci jeste *framerate*, broj frejmova u sekundi. Framerate video sekvence zavisi od primjene. Za filmove, standardni framerate iznosi 24 *fps* (*frames per second*, broj frejmova u sekundi). Video igre najčešće prikazuju 30 ili 60 fps. Većina monitora ne može prikazivati više od 60 fps. Iako predstavlja važnu osobinu video sekvenci, nama framerate nije od naročite važnosti. Ulaz i izlaz predstavljaju sekvence frejmova, koje mogu biti prikazane proizvoljno često. O tome koji framerate je pogodan za koju namjenu će biti više govora u dijelu o primjenama video interpolacije

Zavisno od toga u kojoj smo fazi obrade frejma, piksel predstavlja jednu od dvije različite stvari. Početni i krajnji video fajlovi imaju piksele sa 3 komponente: Crvena, zelena i plava. Svaka komponenta ima cjelobrojnu jačinu između 0 i 255, uključivo. Međutim, u fazi obrade, često je pogodnije gledati na svaki piksel kao samo jedan broj (u istom opsegu) koji predstavlja jačinu svjetlosti piksela (drugim riječima, frejm postaje crno-bijel). Intenzitet svjetlosti svakog piksela će biti obična aritmetička sredina crvene, zelene i plave komponente.

1.1.1 Video fajlovi

Iako fokus rada nije na video fajlovima, vrijedi spomenuti nekoliko detalja. Proces čitanja i pisanja video sekvenci u fajlove će obavljati OpenCV biblioteka (o kojoj će biti više riječi u posljednjem poglavlju). Čuvati video fajlove kao obične nizove frejmova bi bilo izuzetno neefikasno. Naime, video HD dimenzije dužine 2 sata (pri čemu svaki piksel zauzima 3 bajta i framerate iznosi 24 fps) bez ikakve kompresije bi zauzimao $2*3600*24*1280*720*3 = 477757440000$ bajta (~466 GB), gdje u stvarnosti ta veličina iznosi najčešće nekoliko gigabajta. Kompresija video fajlova se zasniva na 2 osnovna principa: *intraframe* kompresija i *interframe* kompresija. Intraframe kompresija se bavi smanjivanjem veličine svakog pojedinačnog frejma, što nema dodirnih tačaka sa tehnikama interpolacije frejmova. Interframe kompresija pokušava smanjiti veličinu krajnjeg fajla tako što traži sličnosti između uzastopnih frejmova, te umjesto spašavanja čitavih frejmova, spasi razlike između trenutnog i prošlog frejma (ili trenutnog i narednog). Poglavlje 3 je posvećeno upravo tehnikama koje traže sličnosti između frejmova. Osim video sadržaja, video fajlovi sadrže i druge komponente poput jedne ili više audio komponente, prijevoda, menija, itd. Ove komponente neće biti obrađivane u ovom radu, niti ih OpenCV biblioteka podržava.

1.2 Primjene video interpolacije

1.3 Osnovni koraci algoritma video interpolacije

U ovom dijelu će biti objašnjeni generalni koraci od kojih se sastoje algoritmi interpolacije frejmova, pri čemu su neki koraci zasebni algoritmi. Za većinu koraka će u ovom radu biti objašnjeno više algoritama.

1. Učitamo naredna dva frejma originalne video sekvence.

2. Odredimo da li je drugi frejm keyframe. Ako jeste, interpolirani frejm je jednak prvom frejmu.

3. U suprotnom, svaki od piksela prvog frejma povežemo sa jednim pikselom drugog frejma (ili suprotno, zavisno od algoritma), ili odredimo da ne postoji konekcija.

4. Pomak piksela iz jednog frejma u drugi, odnosno razlika koordinata piksela i njegovog povezanog piksela, nazivamo vektorom pomaka. Na piksele primijenimo njihove odgovarajuće vektore pomaka pomnožene sa faktorom pomaka, koji zavisi od toga gdje se interpolirani frejm nalazi između dva originalna frejma (npr. faktor pomaka bi bio 0.5 ako između svaka dva frejma) interpoliramo samo jedan. Piksele upišemo na pronađenu lokaciju.

5. Izvršimo eventualne popravke kao i popunjavanje rupa koje nastaju jer ne postoji vektor pomaka za svaki piksel, i neki pikseli će biti upisani na isto mjesto.

6. Novokreirani frejm spasimo.

Osnovne tehnike

2.1 Duplikacija frejmova

2.2 Linearna interpolacija

Algoritmi uparivanja blokova

3.1 Uvod

Prva klasa algoritama koje ćemo proučavati kreću od iste osnovne ideje: Podijeliti prvi frejm na blokove, za svaki blok pronaći vektor pomaka iz prvog u drugi frejm, te primijeniti jedan dio tog vektora pomaka na blok. 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 značajno 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 se prvo pikseli pretvore u crno-bijele, sa jednom komponentom koja predstavlja jačinu bijele boje 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 jačina odgovarajućih 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. U dijelu vezanom za osnovne algoritme i ARPS, obrađuju se 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.

3.1.1 PSNR

3.2 Osnovni algoritmi

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.

3.2.1 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.

3.2.2 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 i ne koristi.

3.2.3 Četverostepena pretraga

3.2.4 Dijamantna pretraga

3.3 Adaptive Rood Pattern Search - ARPS

3.4 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 formulom:

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

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 testiranjem).

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

4.2.2 Ispravljanje neispravnih vektora pomaka

4.3 Izglađivanje granica između blokova

Interpolacija frejmova korištenjem optičkog toka

Paralelno izvršavanje algoritama interpolacije

Implementacija interpolatora korištenjem OpenCV biblioteke i benchmark testovi

Zaključak