Računarski fakultet, Univerzitet UNION

Diplomski rad:

Morfing slike i algoritam Bajera i Nilija (Image morphing and the Beier-Neely algorithm)

Kandidat:

Emilija Dotlić, RN 103/2020

Mentor:

Dragan Mašulović

Beograd, 2023.

Predgovor

Diplomski rad koji imate pred sobom predstavlja rezultat mojeg interesovanja i strasti prema računarstvu i grafici. Od malena sam uživala u tome da budem kreativna i nešto stvaram, pa je upravo taj kreativni aspekt ono što me je prvo privuklo kod računarske grafike. Morfing slike i algoritam Bajera i Nilija je tema koja mi je od početka delovala interesantno, jer je to strana grafike sa kojom nisam imala dodirnih tačaka do sad. Pored toga, od početka mi je delovala grafički primamljivo zbog mogućnosti dobijanja finog prelaza između dve slike. S obzirom da ova tema obuhvata spoj programiranja, kreativnosti i matematike, odmah sam na nju gledala kao na jedan zanimljivi izazov, ali iako sam sebi dala vremena da razmotrim druge ideje, uvek sam se vraćala na ovu temu, jer mi je bila izuzetno zanimljiva i u potpunosti nova za mene, te sam htela da se oprobam u toj tehnici računarske grafike.

Pre nego što pređemo na stručni deo rada, želela bih da iskoristim ovu priliku da izrazim svoju iskrenu zahvalnost svojoj porodici, prijateljima i profesorima koji su bili uz mene tokom celog studiranja. Najpre, hvala mom mentoru, profesoru Draganu Mašuloviću, na podršci, stručnim savetima i vođenju tokom procesa izrade rada.

Još jednom, hvala mojoj porodici, majci Aniti, ocu Đurici i bratu Mladenu, što su uvek verovali u mene i bili mi neizmerna podrška.

Emilija Dotlić

Sadržaj

Uvod

U savremenoj računarskoj grafici, morfing slike i Bajer-Nilijev algoritam predstavljaju bitne koncepte koji su oblikovali način na koji percepiramo, stvaramo i manipulišemo vizuelnim sadržajem. Ova interdisciplinarna oblast nam omogućava kreiranje spektakularnih vizuelnih efekata, kontrolu oblika, i rešavanje različitih izazova u domenima animacije i dizajna.

Morfing slike predstavlja tehniku koja omogućava postepenu i fluidnu transformaciju između dve ili više slika. Ova tehnika je široko prisutna u filmovima, video igrama, medicinskim simulacijama i mnogim drugim aplikacijama. Bajer-Nilijev algoritam je matematički pristup za kontrolisanu transformaciju slika ili oblika, a često se koristi u grafičkom dizajnu kako bi se postigli glatki oblici i precizne tranzicije između različitih vizuelnih elemenata.

Kroz ovaj diplomski rad objasniću ove tehnike, sa ciljem da pružim dublje razumevanje principa morfinga slike i Bajer-Nilijevog algoritma.

Morfing slike je značajna tehnika u računarskoj grafici u kojoj se susreću matematika, programiranje i umetnička kreativnost. Kroz ovaj rad dotaći ću se svih ovih delova, kao i mogućnosti unapređivanja i primene u oblasti računarske grafike. Algoritam Bajera i Nilija, sa svojim matematičkim osnovama i preciznošću, dodatno unapređuje ovu oblast, omogućavajući dizajnerima i programerima da postignu izvanredne efekte u svojim radovima.

Prilikom izrade diplomskog rada implementirala sam algoritam po smernicama iz originalnog dokumenta Bajera i Nilija na ovu temu. Algoritam je implementiran u programskom jeziku C, uz pomoćne biblioteke OpenGL i RAFGL. Iako sam na internetu uglavnom nailazila na implementacije u programskom jeziku C++, odlučila sam da implementiram algoritam u C programskom jeziku, iako u njemu ne postoje pomoćne strukture kao što su na primer vektori, zato što sam želela da objedinim neka od znanja koje sam stekla u prethodne četiri godine na fakultetu.

1. Glava: Uvod u osnovne pojmove, preduslove izvršavanja algoritma i ciljeve izrade rad

U okviru ovog dela, pojasniću osnovne pojmove, preduslove algoritma i ciljeve izrade rada, koji su potrebni za bolje razumevanje tehnike i algoritma opisanih u radu.

1.1. Spajanje slika (Image blending)

Spajanje slika je tehnika koja se često koristi u računarskoj grafici i obradi slika kako bi se spajale dve ili više slika interpolacijom boja piksela između tih slika. Svaka sekvenca (promenjeni frame) animacije postignute ovom tehnikom predstavlja ponderisanu sumu izvorne i ciljane slike bez poravnanja karakteristika (image warpinga). To najlakše možemo opisati kao dodavanje jedne slike preko druge, uz napomenu da originalna slika bledi vremenom, a ciljna se pojavljuje. Ova tehnika sama po sebi daje vizuelno loše rezultate i daje izgled dvostruke slike sa najčešće neporavnatim karakteristikama. Zbog toga se pre početka blendiranja slike često poravnavaju po karakteristikama, uz pomoć odgovarajućih deformacija slike. Pored toga često se za poboljšanje ovog algoritma koriste i različite tehnike za prilagođavanje providnosti (alfa kanala) pojedinih piksela, gradijentne maske, ili matematičke funkcije koje određuju kako će se pikseli iz različitih slika kombinovati. Nadogradnjom image blendinga svim ovim tehnikama može se postići poprilično glatka i prirodna tranzicija, ali algoritam sam po sebi bez napomenutih nadogradnji ne daje previše zadivljujuće rezultate.

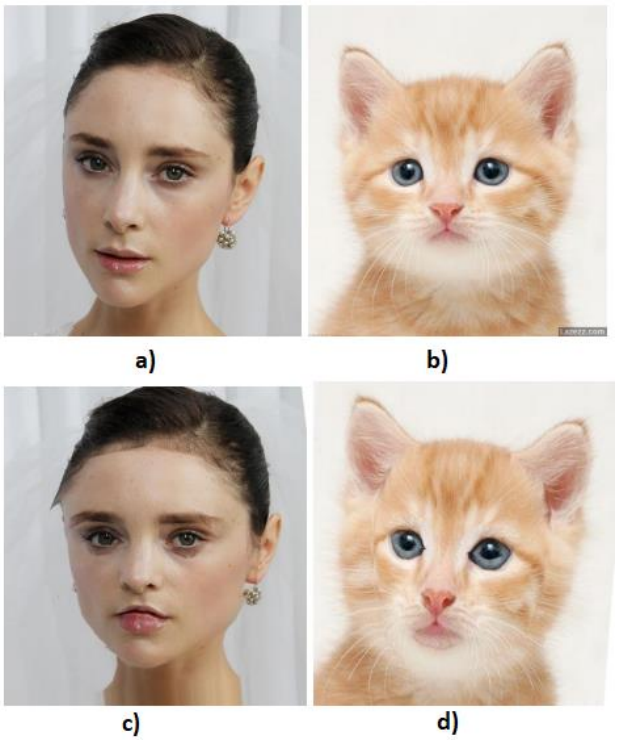
a) b) c)

Slika 1. Primer mešanja (spajanja) slika
a) Izvorna slika (od koje krećemo)
b) Sekvenca generisana primenom spajanja slika (Predstavlja srednju sekvencu algoritma, gde obe slike imaju podjednak uticaj, tj. na pola puta smo što se tiče mešanja od prve ka drugoj, tj. krajnjoj slici)
c) Krajnja slika
Izvor: [15]

1.2. Deformacija slike (Image warping)

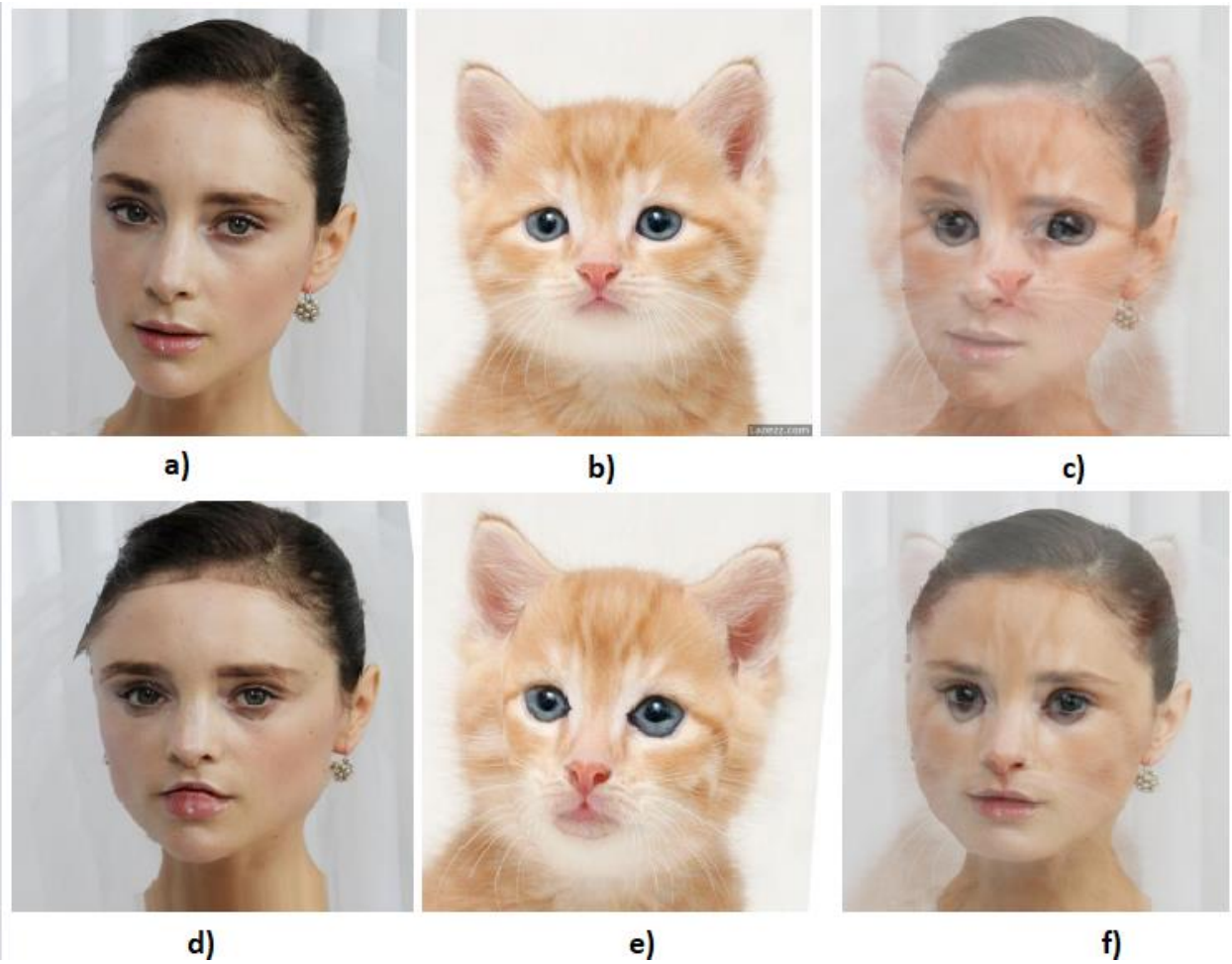
Deformacija slike je proces transformacije slike ili njenih delova kako bi se postigla promena u geometriji, obliku ili perspektivi. Tokom ovog procesa dolazi do transformacije koordinata, tj. menja se prostorni izgled slike, ali ne i boja. Uz pomoć odgovarajuće preslikavajuće funkcije, svaki piksel iz izvorne slike će se preslikati na odgovarajuće mesto u ciljnoj slici. Postoje dva načina preslikavanja piksela, a to su forward mapping i reverse mapping.

- Forward mapping prolazi kroz izvornu sliku piksel po piksel, kopirajući svaki piksel na odgovarajuće mesto u ciljnoj slici. Može se javiti problem da neki pikseli na ciljnoj slici nemaju nijedan piksel iz izvorne slike preslikan na njih, te da ostaju prazni, tj. nemamo informaciju koje boje piksel treba da bude na toj poziciji.
- Reverse mapping je način preslikavanja piksela prilikom kojeg algoritam prolazi kroz odredišnu (ciljnu) sliku piksel po piksel, uzorkujući odgovarajuću tačku iz izvorne slike. S obzirom da može doći do situacije da se piksel koji bi trebalo da bude mapiran sa izvorne slike nalazi izvan granica izvorne slike, kako ne bismo to mesto ostavljali prazno možemo primeniti bilinearnu interpolaciju u odnosu na okolne piksele. Na ovaj način bismo umesto neke nasumične vrednosti koju je raster izvorne slike imao pri inicijalizaciji, za taj piksel stavili srednju vrednost piksela njegove okoline.



Slika 2. Primer deformacije slike (Image warping)
a) i b) Originalne slike
c) i d) Deformisane slike po zajedničkim karakteristikama
Modifikovano iz [3]

Deformacija slike omogućava promenu izgleda slike, koja se često primenjuje radi prilagođavanja ili poboljšanja drugih vizuelnih efekata. Samim tim neretko se koristi pre blendiranja, odnosno spajanja više slika u jednu. Ovo značajno poboljšava rezultate spajanja slika, jer na ovaj način pre samog spajanja dobijamo slike koje su poravnate po karakteristikama, pa samim tim i mešanje tih slika ima više smisla jer su karakteristike na obe slike na istim pozicijama. Primer koji lepo opisuje ovaj proces se nalazi na sledećoj slici.



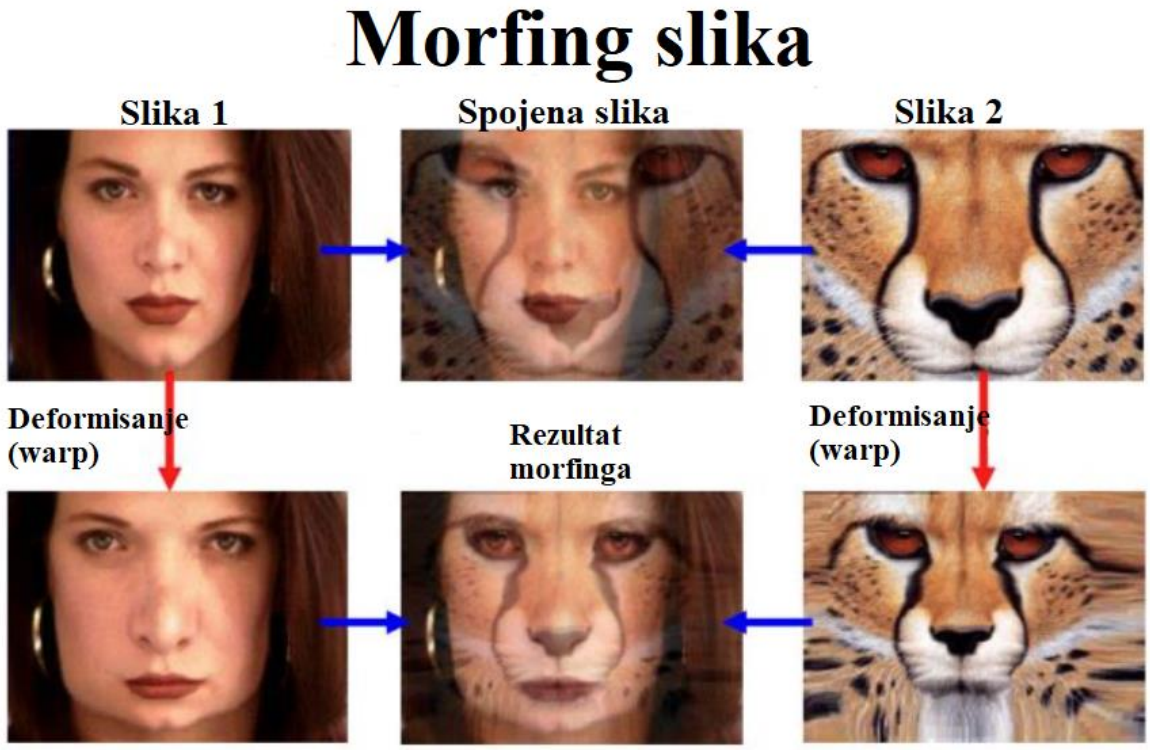
Slika 3. Primer mešanja (spajanja) slika sa i bez deformacije pre početka mešanja
a) Izvorna slika (od koje krećemo)
b) Ciljna slika (do koje želimo da stignemo)
c) Spojena originalna izvorna i krajnja slika sa podjednakim uticajem na rezultujuću sliku, ali bez deformisanja originalnih slika (blending slika bez prethodnog poravnanja po karakteristikama slike)
d) Deformisana izvorna slika
e) Deformisana ciljna slika
f) Spojena originalna izvorna i krajnja slika sa podjednakim uticajem na rezultujuću sliku, ali sa deformisanjem originalnih slika (blending slika sa prethodnim poravnanjem po karakteristikama slike)
Izvor: [3]

1.3.Morfing slike (Image morphing)

Morfing slike se može definisati kao tehnika obrade uz pomoć koje se postepeno, putem glatkog prelaza, jedna slika pretvara u drugu. Ova tehnika objedinjuje prethodno dva objašnjena procesa, tj. obuhvata i spajanje i deformaciju slike.

Glavna razlika koja odvaja proces morfinga slike od spajanja slike sa prvobitnom deformacijom po karakteristikama je u tome što se u morfingu slike deformacija slike radi u svakoj iteraciji i pomeraj karakteristika zavisi od jedinice vremena u kojem se algoritam nalazi. Nasuprot tome imamo spajanje slike sa deformacijom poravnanja po karakteristikama slike koji se vrši samo pre početka algoritma blendiranja slika. Dakle, proces blendinga kreće sa već unapred deformisanim slikama po karakteristikama, a zatim samo vrši obično spajanje te dve unapred deformisane slike. Sa druge strane, morfing slike kreće od originalnih, nedeformisanih slika i kako vreme protiče, tj. kako algoritam napreduje, tako uporedo u svakoj iteraciji vrši i deformisanje i spajanje slika. Stepen prelaza, odnosno deformisanja i blendovanja slike zavisi od jedinice vremena u kojoj se algoritam nalazi.

Poenta ovog algoritma je da između dve ulazne slike napravi fin i postepen prelaz iz jedne sliku u drugu.



Slika 4. Šema kako se kombinuju spajanje slika i deformacija slike i kao rezultat daju morfingovanu sliku
Modifikovano iz [15]

Kratak pregled ovog algoritma je prikazan na šemi iznad, a sada ćemo malo bliže pojasniti kako zapravo ovaj algoritam funkcioniše.

Kako prolazi vreme, tako tokom ovog procesa prva slika postepeno bledi i izobličava se (prema drugoj slici), čime sve više dobija karakteristike druge slike; istovremeno druga slika počinje da se postepeno pojavljuje i od izmenjene slike počinje da se izobličava ka svom krajnjem izgledu. Samim tim, na početku ovog procesa sekvence su mnogo sličnije prvoj slici. Srednja slika sekvence je prosečna vrednost prve slike izobličene na pola puta prema drugoj i druge slike izobličene na pola puta prema prvoj slici, dok su sekvence koje dobijamo na kraju procesa mnogo sličnije drugoj slici.

a) b) c) d) e)

Slika 5. Primer morfinga slike
a) Izvorna slika (od koje želimo morfing)
b), c), d) sekvence generisane primenom morfinga slike u različitim trenucima algoritma (te je slika pod b) mnogo sličnija izvornoj slici, jer je u prvoj polovini izvršavanja algoritma, slika pod c) sekvenca sa polovine puta izvršavanja programa, gde je podjednak uticaj obe slika, a slika pod d) je sekvenca koja je mnogo sličnija krajnjoj slici, jer je u drugoj polovini izvršavanja algoritma)
e) Krajnja slika (željeni rezultat morfinga od izvorne slike)
Izvor:[16]

Algoritmi za morfing slike mogu se podeliti u tri kategorije: mrežna deformacija, deformacija zasnovana na karakteristikama slike i interpolacija tanke ploče. U sledećim odeljcima ću kratko pojasniti svaku kategoriju, a u daljem tekstu ću se više baviti deformacijom zasnovanom na karakteristikama slike, s obzirom da u tu kategoriju spada tema ovog rada.

1.3.1. Mrežna deformacija (Mesh Warping)

Mrežna deformacija je pristup morfingu koji se fokusira na promenu oblika slike putem deformacije "mrežom" tačaka ili čvorova. Osnovna ideja je da se slika podeli na mrežu (grid) ili skup tačaka, gde će svaka tačka predstavljati određeni region slike. Korisnik ili algoritam postavlja "bitne tačke" na početnoj i ciljnoj slici, a zatim se mreža između ovih tačaka transformiše tako da se postigne postepena promena oblika.

Mrežna deformacija omogućava preciznu kontrolu nad oblikom, ali može zahtevati veći napor pri postavljanju kontrolnih tačaka.

1.3.2. Deformacija zasnovana na karakteristikama slike (Feature-Based Warping)

Ova kategorija algoritama morfinga slike se fokusira na manipulaciju karakterističnih tačaka ili obeležja slike kako bi se postigao smislen morfološki prelaz. Osnovna ideja je da se prate određene karakteristike na početnoj i ciljnoj slici, kao što su uglovi, ivice, tačke interesovanja i drugi vizuelni elementi.
Kada se karakteristike identifikuju, algoritam prati njihovu transformaciju tokom morfinga.
Transformacija karakteristika se može postići različitim metodama, uključujući rotaciju, translaciju i skaliranje.
Deformacija zasnovana na karakteristikama slike omogućava preciznu kontrolu nad promenama u određenim delovima slike.
Pored toga, ova metoda je relativno jednostavna što se tiče označavanja karakteristika slike, a samim tim i definisanja toka deformacije slika, što značajno olakšava animatorima korišćenje.

1.3.3. Interpolacija tanke ploče (Thin-Plate Spline Interpolation)

Interpolacija tanke ploče je matematički model koji se koristi za morfing slike i koristi posebnu vrstu funkcije, nazvanu "tanka ploča", koja se koristi za interpolaciju pozicija tačaka između početnih i ciljnih kontrolnih tačaka.
Ova metoda je pogodna za morfing slika sa mnogo tačaka i obimnim promenama u obliku. Interpolacija tanke ploče često se koristi u kombinaciji sa mrežnom deformacijom ili karakterističnim tačkama kako bi se postigla bolja kontrola nad morfološkom tranzicijom.

Od ove tri podvrste morfinga slike mnogi smatraju da je baš deformacija zasnovana na karakteristikama slike najpraktičnija metoda. Na tu temu Zhu Feng je rekao [3]:

"Due to the acceptable computational complexity, proper degree of implementation difficulty, and proven quality of visual effect, we feel that the feature-based approach is most practical." [1]

Algoritam opisan u ovom radu, Bajer-Nilijev algoritam, pripada kategoriji morfinga slike zasnovanih na karakteristikama, tako da ćemo se tom kategorijom najviše i baviti.

[1] Feng Zhu, Image Morphing with the Beier-Neely Method, B.Sc., Northwestern Polytehnical University, 2013

1.4. Algoritam Bajera i Nilija

Sada kada smo malo bliže pojasnili pojam morfinga slike, možemo preći i na jedan od najpoznatijih primera ove tehnike, a to je morfing zasnovan na karakteristikama slike kojeg su razvili Bajer i Nili (Beier i Neely) 1992. godine za potrebe spota pesme Majkla Džeksona pod nazivom "Black or White". Baš slike iz ovog spota ću i ja koristiti kao primer primene ovog morfinga uz pomoć algoritma Bajera i Nilija.

Osnovna ideja ove metode je da uz pomoć skupa parova usmerenih segmenata linija omogući fini prelaz iz izvorne u ciljnu sliku. Dakle, bilo kakva značajna karakteristika sa jedne slike treba biti označena usmerenom linijom i da ima svog para na ciljanoj slici. Samo preslikavanje pikseka sa izvorne na ciljnu sliku se vrši uz pomoć tih prethodno definisanih parova usmerenih linija. Ključna stvar kod ovog algoritma je funkcija preslikavanja, tj. odgovarajuće deformacije u odnosu na karakteristike slika koje su obeležene usmerenim linijama. Deformacija piksela je zapravo ponderisana suma preslikavanja svakog para usmerenih linija, pri čemu su težine povezane sa dužinom linije i najkraćom udaljenošću između piksela i linija. Svi ti faktori se lako mogu parametrizovati u algoritmu, te će o tim parametrima biti priče u kasnijim delovima rada. Rezultat ovog procesa je fina transformacija od izvorne ka ciljanoj slici uz adekvatnu deformaciju karakteristika. Ukoliko u ovaj algoritam dodamo i vremensku jedinicu po kojoj bi se vršila deformacija i mešanje deformisanih sekvenci, onda bismo kao rezultat dobili animirani prelaz između dve slike.

1.5. Anotacija korišćena u radu

U okviru ovog rada piksele, tj. tačke sa slike ću obeležavati ***VELIKIM BOLDOVANIM KURZIVNIM SLOVIMA***, i svaka takva tačka je određena svojim koordinatama (x,y). Pored toga linije ću definisati preko 2 tačke, tj. kao par piksela (***PQ***), gde je usmerenost linije određena od prvog navedenog piksela ka drugom. Skalari su u ovom dokumentu zapisani u obliku ***malih boldovanih kurzivnih slova***.

Sve oznake u tekstu koje pored sebe imaju oznaku **'** (apostrof), odnose se na vrednosti definisane u odnosu na izvornu sliku. Promenljive bez te oznake se odnose na vrednosti definisane u odnosu na ciljnu sliku (ovaj način obeležavanja je smisleniji jer se u okviru algortima koristi reverse mapping).

Kada se u tekstu naiđe na reč linija, misli se na usmerenu liniju koja definiše karakteristiku slike. Takođe, svaka linija ima svog parnjaka, tj. postoji njoj kompatibilna linija sa druge slike.

Oznake u kodu se mogu razlikovati od ovde navedenih anotacija, zbog nemogućnosti ili nepraktičnosti primene naziva definisanih na ovaj način. Promenljive iz koda koje su nejasno definisane, tj. koje se ne podudaraju sa nazivom objašnjenom u samom algoritmu će biti objašnjene u tekstu.

S obzirom da se u kodu pored klasičnih promenljivih programskog jezika C nalaze i dodatno kreirane strukture, za reprezentaciju tački i linija, evo načina na koji su definisane:

```
typedef struct Vec2D{
    float x;
    float y;
}Vec2D;

typedef struct Line{
    Vec2D start;
    Vec2D end;
    Vec2D vector;
    struct Line* next;
}Line;
```

Slika 6. Struktura za dvodimenzioni vektor (najčešće posmatran kao tačka slike) i za liniju(koja je definisana uz pomoć dve tačke, a to su start i end tačka linije)

1.6. Preduslovi primene algoritma

Preduslovi primena algoritma su da ulazne slike budu istih dimenzija, s obzirom da algoritam sam po sebi ne pokriva slučaj promene veličine prozora u okviru kojeg se dešava morfing slike.

Takođe, veoma bitan preduslav je da na obe ulazne slike bude postavljen jednak broj linija, kako bi svaka slika sa izvorne slike imala svog para na ciljnoj slici, i obrnuto. Ukoliko se broj razlikuje, algoritam neće znati sa koje pozicije na koju treba da mapira piksele iz njegove okoline, te će se javljati problem i prilikom samog pokretanja koda.

Sledeća stavka ne predstavlja tehnički problem, za razliku od prethodne dve stavke, jer je program sa ovim "problemom" moguće pokrenuti, ali se neće dobijati lepi rezultati. Ova stavka je zapravo smisleno postavljanje linije na obe od slika. Dakle, svaka linija koja je postavljena na nekoj od karakteristika jedne slike treba da bude postavljena na istoj karakteristici druge slike, kako bi deformacija između te dve karakteristike imala smisla ili bar bila u skladu sa animatorovim željama.
Mi možemo izvršiti morfing slika i između slike čovekovog lica i slike kuće, ali sama deformacijama po karakteristikama malo tu gubi smisao, iako i u takvoj situaciji animator može napraviti da to ima smisla, ako na primer poveže dva prozora sa očima, usta sa vratima, a kosu sa krovom. Samim tim vidimo da nam ovaj algoritam pruža dosta slobode prilikom samog određivanja toka animiranja procesa prelaska iz jedne slike u drugu, ali bi nevešti animatori mogli zbog loše postavljenih linija da dobiju rezultat kakav nisu prvobitno želeli.

Sličan problem predstavlja i pomešan redosled linija, jer ukoliko je na jednoj slici obeleženo levo oko, pa desno oko, pa nos, pa usta, a na drugoj slučajno obrnutim redosledom: desno oko, levo oko, usta, nos, dobiće se prelaz između različitih karakteristika slike, iako su iste karakteristike obeležene na obe slike. U navedenom primeru će se levo oko sa izvorne slike deformisati ka desnom oku izvorne slike i obrnuto. Ista stvar bi se desila i za nos i za usta, te bi rezultat bio čudan i verovatno ne onakav kakav je animator želeo.

Kako bi se postigli bolji rezultati u prelazu iz jedne slike u drugu, savetuje se da se sve značajne tačke, tj. karakteristike označe, kako bi se pikseli baš blizu tih piksela preslikali tačno na mesto gde želimo. Ako bismo koristili malo linija, tačnije ukoliko bismo preskočili da linijom obeležimo neku bitnu karakteristiku slike, veoma verovatno bi se ona pod uticajem okolnih linija deformisala

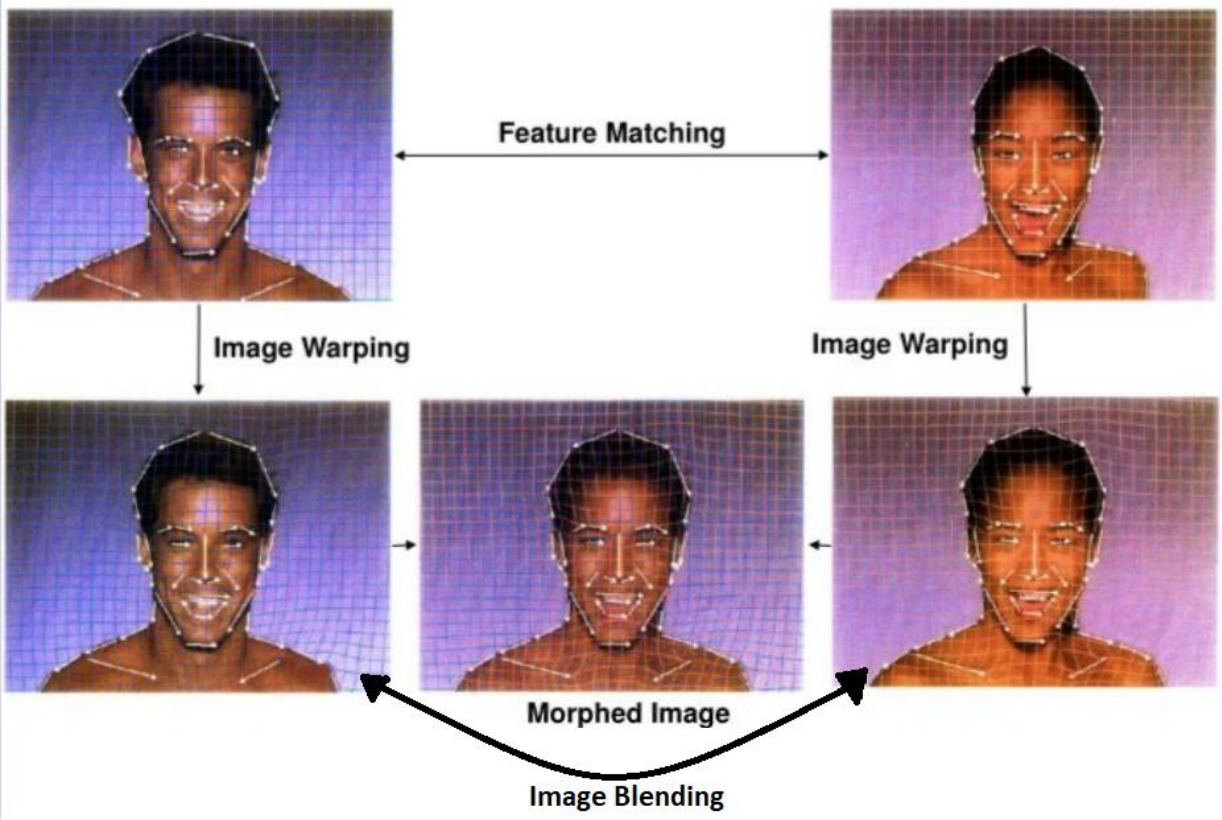
na način na koji mi nismo želeli. Samim tim, savet je da se sve značajne tačke na obe slike obeleže što je moguće preciznije.

1.7. Cilj implementacije algoritma

Cilj implementacije algoritma je da u svakoj iteraciji deformiše obe slike po karakteristikama, a zatim da te dve deformisane sekvence spoji u jednu rezultujuću sekvencu uz pomoć blendinga. Zbog toga je bitno da su karakteristike dobro obeležene linijama, jer ukoliko pomešamo koja linija je sa kojom par ili ukoliko ih pogrešno usmerimo možemo dobijati jako čudna rešenja. Samim tim bitna faza da bismo dobili željeni rezultat algoritma je postavljanje odgovarajućih usmerenih parova linija na obe slike.

Rezultat koji želimo da dobijemo je taj da od sekvenci generisanih kroz iteracije napravimo animaciju, a ako u algoritam ubacimo faktor vremena i po njemu radimo deformisanje i blendovanje slika, dobićemo upravo to.

O samom načinu implementacije ovog algoritma će biti više priče u nastavku dokumenta, ali osnovnu ideju možemo da vidimo na sledećoj ilustraciji.



Slika 7. Prikazuje šemu prolaska kroz algoritam od gornje dve originalne slike, pa do donje centralne slike koja predstavlja rezultat morfinga slike
Modifikovano iz [10]

2. Glava: Algoritam Bajera i Nilija

U okviru ove glave opisan je algoritam Bajera i Nilija koji pripada kategoriji deformacija zasnovanih na karakteristikama slike (Feature-Based Warping).

2.1. Opis tehnike

Tehnika deformacije kod Bajer Nilijevog algoritma zasniva se na karakteristikama koje su označene linijama na izvornim i ciljnim slikama. Svaka usmerena linija ima svoje polje dejstva kojim utiče na svoju okolinu, pri čemu jačina tog polja opada kako se udaljavamo od te linije. Ovakva vrsta morfinga slika nam omogućava veći stepen kontrole nad morfološkom transformacijom.

Najčešće kada se ovaj algoritam koristi radi se sa više parova usmerenih linija, pa je tada rezultat pomeranja piksela zapravo ponderisana suma preslikavanja svakog para usmerenih linija, pri čemu su težine povezane sa dužinom linije i najkraćom udaljenošću između tačke i linije.

Svaki međurezultat ove tehnike dobijamo uz pomoć funkcije deformacije primenjene na izvornu i ciljnu sliku u određenom trenutku. Kako vreme odmiče tako transformacijom od linija sa izvorne slike sve više težimo ka liniji koji je njen odgovarajući par sa ciljne slike vršeći interpolaciju tih linija. Odnosno, kroz iteracije, sve više i više modifikujemo međulinije od onih sa izvorne slike ka onima sa ciljne, kako bismo dobili smernice po kojima treba da vršimo deformisanje piksela u datoj iteraciji.

Da bismo bolje razumeli kako ovaj algoritam funkcioniše proći ćemo prvo kroz deformaciju sa jednim parom linija, jer nam je ona osnova deformacije sa više parova linija. U suštini mi ćemo u algoritmu sa više linija, za svaku liniju računati kakav bi ona uticaj imala za prosleđeni piksel, a onda gledati kolika je njena težina, odnosno koliko ona zapravo utiče na taj piksel u odnosu na ostale linije sa te slike. Samim tim vidimo da se deformacija za više linija nadovezuje na algoritam deformacije sa jednim parom linija, te je bitno da prvo razumemo kako on funkcioniše.

2.2. Deformacija sa jednim parom linija karakteristika slika

Kada radimo sa samo jednim parom linija, transformacija svih piksela zavisi samo od te jedne karakteristike koja je označena linijom. Dakle ona ima 100% uticaja na sve piksele sa slike koju deformišemo.

Ono što mi želimo da uradimo u okviru ovog algoritam je da prolazimo kroz svaki piksel ciljne slike i da za njegovu poziciju tražimo odakle treba da uzorkujemo piksel koji ćemo dodeliti našem pikselu ciljne slike. Dakle, mi ovde radimo ono što smo ranije definisali kao reverse mapping, kako nam se ne bi javljao problem da neki pikseli na ciljnoj slici nemaju nijedan piksel iz izvorne slike preslikan na njih. Samim tim ovde radimo uzorkovanja sa deformisane pozicije u odnosu na piksel iz ciljne slike. Ukoliko nije moguće da uzorkujemo tačno taj piksel sa dobijene pozicije, tj. ukoliko te nove koordinate koje smo dobili ispadaju iz okvira slike, potrebno je da odradimo bilinearnu interpolaciju po okolnim pikselima, kako nam taj piksel iz ciljne slike ne bi ostao

prazan. Ukoliko je pozicija sa koje treba da uzorkujemo unutar izvorne slike, piksel sa te pozicije ćemo prekopirati na poziciju piksela za koji smo prolazili u ciljnoj slici.

2.2.1. Matematička strana algoritma

Ukoliko sa ***X*** označimo poziciju na ciljnoj slici, a sa ***X'*** odgovarajuću poziciju na izvornoj slici, dok liniju sa ciljne slike predstavimo preko tačaka ***P*** i ***Q***, a njoj odgovarajuću liniju sa izvorne slike sa ***P'Q'***, onda funkciju preslikavanja možemo opisati sledećom jednačinom:

$$X' = P' + u \cdot (Q' - P') + \frac{v \cdot Perpendicular(Q' - P')}{\|Q' - P'\|}$$

[1]

gde su ***u*** i ***v*** definisani na sledeći način:

$$u = \frac{(X - P) \cdot (Q - P)}{\|Q - P\|^2}$$

$$v = \frac{(X - P) \cdot Perpendicular(Q - P)}{\|Q - P\|}$$

[1]

U ovim formulama **Perpendicular()** predstavlja funkciju koja vraća vektor koji je normalan i iste dužine kao vektor koji joj je prosleđen. Nije bitno da li se koristi levi ili desni normalni vektor, samo je da bitno da se dosledno koristi onaj za koji se opredelimo.

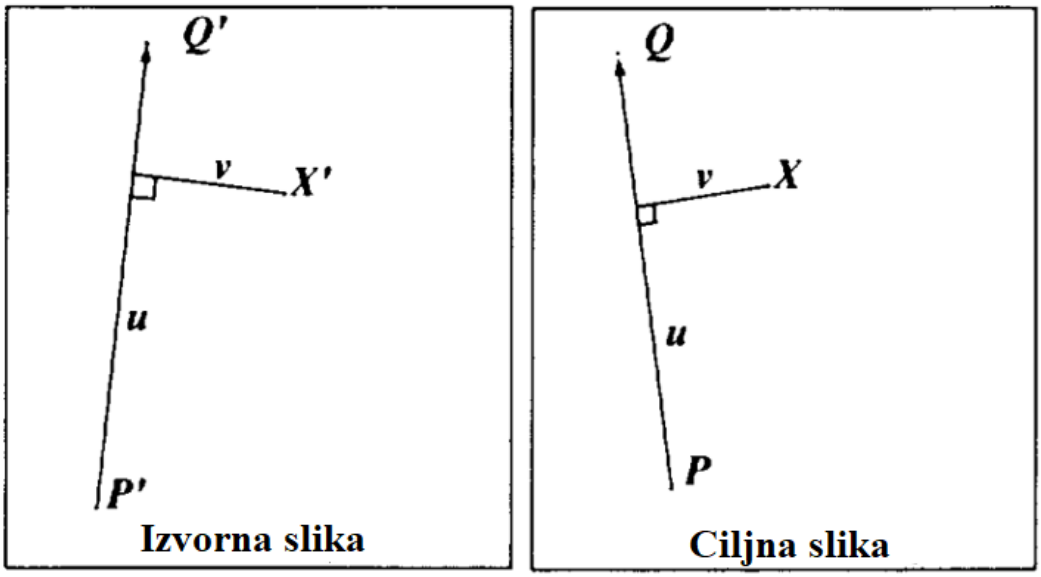
Vrednosti ***u*** i ***v*** su vektori, gde ***u*** predstavlja poziciju duž linije i varira od 0 do 1 kako piksel putuje od tačke ***P*** do tačke ***Q***. Vrednost ***v*** predstavlja normalan vektor na liniju koji predstavlja udaljenost piksela od linije izraženu u pikselima.

Algoritam funkcioniše tako što za svaki piksel iz ciljne slike prolazimo (u datoj iteraciji je to piksel sa pozicijom ***X***) i za njega računamo ***u***, ***v*** i ***X'***. Nakon što to sračunamo uzorkujemo sa originalne slike piksel na poziciji ***X'*** i vrednost tog piksela dodeljujemo ciljnoj slici za poziciju piksela **X**.

Dakle, *ciljnaSlika(**X**) = izvornaSlika(**X'**).*

Ovo nije moguće odraditi ako je pozicija piksela ***X'*** sa koje treba da uzorkujemo vrednost iz izvorne slike van opsega te slike. Tačnije, nije moguće da čitamo sa izvorne slike piksel X' sa kordinatama (x,y), ako je njegova koordinata x van opsega [0, (širina izvorne slike – 1)] ili

koordinata y van opsega [0, (visina izvorne slike - 1)]. U ovoj situaciji potrebno je odraditi bilinearnu interpolaciju u odnosu na okolne piksele kako krajnji piksel ne bi ostao prazan.



Slika 8. Primer jednog para linija
Modifikovano iz [1]

2.2.2. Implementacija morfinga piksela slike za jedan par linija

Po prethodno definisanim koracima izrade algoritma sam implementirala funkciju koja za prosleđeni par linija vrši računanje pozicije ***X'*** (u kodu ***X_source***). Ovaj deo algoritma sam odvojila u posebnu pomoćnu funkciju kako bih mogla da vršim računanje ***X'*** za bilo koju prosleđenu poziciju piksela ***X***(u kodu ***X_dest***) i za bilo koji par linija (u kodu **sourceLine** i **destLine**). Samim tim ova funkcija je lepa demonstracija kako algoritam radi sa jednim parom linija, a na dalje je lako mogu koristiti i u primeni sa više parova linija, pozivajući za svaki piksel i za svaki par linija ovu funkciju.

U okviru implementacije ove funkcije koristila sam dosta pomoćnih funkcija za osnovne računske operacije nad vektorima.

Pored toga, trudila sam se da delove formula raščlanim, kako bi bilo lakše uočljivo koja zagrada se na šta odnosi i kako bi se implementacija koda mogla lakše uporediti sa formulom koja je u njoj primenjena.

Kod sa slike ispod je prosta implementacija formule objašnjene u prethodnom odeljku. Jedina razlika koja može zbuniti čitaoca je to što umesto da kao povratnu vrednost funkcije navedem samo ***X_source*** promenljivu, ja ovde navodim neku strukturu podatka. Za implementaciju sa jednim parom linija je i samo vraćanje ***X_source*** promenljive bilo dovoljno, ali s obzirom da sam ovaj deo koda integrisala u implementaciju deformacije sa više usmerenih linija, gde mi je

potrebno da znam i vrednosti vektora ***u*** i ***v***, uvela sam i ovu jednostavnu strukturu. Dakle, struktura mi služi samo kako bih mogla da vratim sve tri vrednosti promenljivih.

```
TransformResult transformPoint(Vec2D X_dest, Line sourceLine, Line destLine) {

    Vec2D P_dest = destLine.start;
    Vec2D Q_dest = destLine.end;

    Vec2D P_source = sourceLine.start;
    Vec2D Q_source = sourceLine.end;

    Vec2D PX_dest = subtractVecs(X_dest, P_dest);
    Vec2D PQ_dest = subtractVecs(Q_dest, P_dest);


    float intensityVecPQ_dest = intensityVec(subtractVecs(Q_dest, P_dest));

    Vec2D above_u = multiplyVecs(PX_dest, PQ_dest);
    float below_u = pow(intensityVecPQ_dest, 2);


    Vec2D u = divideVecByScalar(above_u, below_u);

    Vec2D above_v= multiplyVecs(PX_dest, perpendicular(PQ_dest));
    float below_v = intensityVecPQ_dest;

    Vec2D v = divideVecByScalar(above_v, below_v);

    Vec2D PQ_source = subtractVecs(Q_source, P_source);
    Vec2D firstPart_X_source = addVecs(P_source, multiplyVecs(u,PQ_source));
    Vec2D secondPart_X_source = divideVecByScalar(multiplyVecs(v, perpendicular(PQ_source)), intensityVec(PQ_source));

    Vec2D X_source = addVecs(firstPart_X_source, secondPart_X_source);

    TransformResult result;
    result.u = u;
    result.v = v;
    result.X_source = X_source;

    return result;
}
```

*Slika 3. Implementacija funkcije **transformPoint()** koja transformiše koordinate piksela na osnovu jednog para linija (jedne linije sa izvorne slike i jedne linije sa krajnje slike)*

2.3. Deformacija sa više parova linija karakteristika slika

Uglavnom nam deformacije sa jednim parom linija ne mogu dati zadivljujuće rezultate, te se zbog toga obično koristi više parova linija. Dakle, svaka karakteristika slike označena linijom ima neki uticaj na piksele u okolini u zavisnosti od njene težine.

Težina zavisi od najkraćeg rastojanja piksela do linije. Samim tim, težina bi trebala biti najveća kada se piksel nalazi tačno na toj liniji, a sve slabija i slabija kako je piksel dalji od nje. Pored toga, težinu kojom neka linija ima uticaj na neki piksel možemo još dodatno parametrizovati promenljivima (u daljem tekstu te promenljive su označene sa ***a***, ***b*** i ***p***).

2.3.1. Matematička strana algoritma

Jednačina za računanje težine koja je korišćena u algoritmu je sledeća:

$$weight = \left(\frac{length^{p}}{(a + dist)} \right)^{b}$$

Slika 9. Formula računanja težine neke linije u odnosu na prosleđeni piksel
Izvor: [1]

gde ***length*** predstavlja dužinu linije, ***dist*** je udaljenost piksela od linije, dok su vrednosti ***a***, ***b*** i ***p*** konstante koje se mogu parametrizovati i uz pomoć kojih se menja relativni efekat linija.

Promenljiva ***dist*** u ovoj formuli može biti predstavljena na različite načine u zavisnosti od vrednosti vektora ***u***. Ukoliko je vrednost vektora ***u*** između 0 i 1, onda je distanca jednaka apsolutnoj vrednosti vektora ***v***. U slučaju da je vrednost vektora ***u*** manja od 0, distanca je jedanaka udaljenosti piksela do tačke ***P***, dok ukoliko je vrednost vektora ***u*** veća od 1, onda distance dobijamo kao udaljenost piksela do tačke ***Q***.

Vrednost promenljive ***a*** predstavlja jačinu kojom neka linija utiče na piksele. Ukoliko je vrednost za komponentu ***a*** jedva veća od 0, onda znamo da ukoliko radimo sa pikselom koji se nalazi tačno na liniji, tj. ukoliko je njegova udaljenost od linije jednaka nula, onda će i jačina te linije biti "beskonačna". Dakle sa ovom vrednošću za ***a*** znamo da će pikseli sa te linije da se preslikaju tačno na liniju koja je njen par. U slučaju kada je ***a*** postvaljeno na 0, može doći do nedefinisanog rezultata, prilikom preseka dve linije, s obzirom da će u toj situaciji obe linije imati beskonačne težine u tački preseka.

Vrednost promenljive ***b*** određuje kako relativna jačina različitih linija opada sa udaljenošću, tj. ukoliko je vrednost velika, tada će svaki piksel biti pogođen samo jednom linijom koja mu je najbliža, dok ukoliko je vrednost za ***b*** nula, onda će svaki piksel biti pogođen svim linijama podjednako. Vrednost komponente ***b*** je najkorisnija u rasponu od 0.5 do 2.

Vrednosti promenljive ***p*** definiše da li će i u kojoj meri dužina linije uticati na njenu težinu. Vrednost ovog parametra je uglavnom u rasponu od 0 do 1, gde ako je nula, onda sve linije imaju istu težinu, dok ako je jedan, onda duže linije imaju veću relativnu težinu od kraćih linija.

Nakon što smo prošli kako dobijamo težinu kojom neka linija ima uticaj na okolne piksele, i nakon što smo već u prethodnom poglavlju prošli kroz to kako izgleda i radi funkcija **transformPoint(),** možemo pogledati implementaciju samog algoritma za rad sa više parova linija.

2.3.2. Implementacija algoritma

Cilj implementacije algoritma je da se za svaku poziciju piksela ciljne slike nađe odgovarajuće mesto sa kojeg treba da se pročita piksel iz izvorne slike, a zatim da se taj piksel i dodeli našem pikselu sa ciljne slike.

Samim tim moramo prolaziti kroz sve piksele ciljne slike i za svaku x i y poziciju na slici pozivati izvršavanje funkcije Bajer Nilijevog algoritma. Piksel sa pozicijom (x,y) nam zapravo predstavlja ono što je u algoritmu definisano sa ***X***.

```
for(y = 0; y < h; y++)
{
    yn = 1.0f*y/h;
    for(x = 0; x < w; x++)
    {
        xn = 1.0f*x/w;

        Vec2D point = {x,y};

        Vec2D newPoint_source = beierNeelyTransform(point, sourceLines, interLines, a, b, p);
        if(newPoint_source.x<0 || newPoint_source.x>w-1 || newPoint_source.y<0 || newPoint_source.y>h-1){
            sampled_source = rafgl_bilinear_sample(&image1, xn, yn);

        }else{
            sampled_source = pixel_at_m(image1, (int)newPoint_source.x, (int)newPoint_source.y);
        }
        pixel_at_m(source, x, y) = sampled_source;

        Vec2D newPoint_dest = beierNeelyTransform(point, destLines, interLines, a, b, p);
        if(newPoint_dest.x<0 || newPoint_dest.x>w-1 || newPoint_dest.y<0 || newPoint_dest.y>h-1){
            sampled_dest = rafgl_bilinear_sample(&image2, xn, yn);

        }else{
            sampled_dest = pixel_at_m(image2, (int)newPoint_dest.x, (int)newPoint_dest.y);
        }
        pixel_at_m(dest, x, y) = sampled_dest;
    }
}
```

*Slika 10. Prolazak kroz piksele ciljne slike i poziv funkcije koja izvršava algoritam za prosleđene vrednosti: **X**(u kodu **point**), niz izvornih linija(u kodu **sourceLines**), niz interpolisanih linija(u kodu **interLines**, koji predstavlja interpolaciju izvornih i ciljnih linija u odnosu na jedinicu vremena) i parametara algoritma a, b i p. Ista funkcija se za deformaciju ciljne slike poziva sa nizom linija sa te slike (u kodu **destLines**)*

Pre prelaska na samu implementaciju algoritma, pojasnila bih prvo na koji način dobijamo niz interpolisanih linija koji se prosleđuje samom algoritmu. Prilikom svakog ulaska u update funkciju, tj. prilikom svake iteracije potrebno je da generišemo novi niz međulinija. Ovaj niz zapravo predstavlja interpolaciju linija sa izvorne i sa ciljne slike u određenoj jedinici vremena. Sam poziv pomoćne funkcije koja nam kreira ovaj niz je prikazan na sledećoj slici.

```
head_inter = line_interpolation(head_source, head_dest,  t);
```

Slika 11. Poziv funkcije koja kreira niz interpolisanih linija

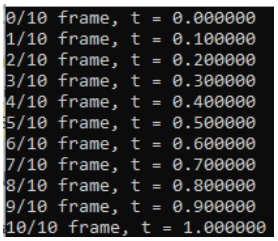
Niz interpolisanih linija dobijam preko goredefinisane pomoćne funkcije, koja od parametara prima glavu liste linija sa izvorne slike, glavu liste linija sa ciljne slike i parameter ***t***, koji predstavlja jedinicu vremena. Parametar ***t*** sam računala u odnosu na to koliko je ulazaka u update funkciju prošlo podeljeno sa nekom konstantom koja predstavlja koliko ulazaka želimo da do imamo pre nego što dođemo do totalnog prelaza u ciljnu sliku.

```
float t = (float)c / frames;
```

*Slika 12. Računanje promenljive **t***

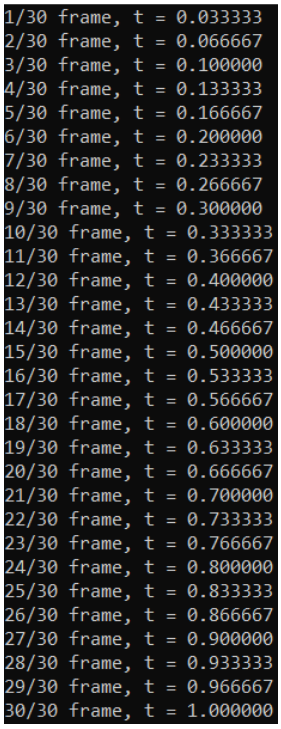
U ovoj računici ***c*** je zapravo brojač koliko frejmova je prošlo od kad smo pokrenuli program, tj. ***c*** se povećava svakim ulaskom u funkciju update. Promenljiva ***frames*** je konstanta uz pomoć koje definišem koliko brzo želim da se povećava parametar ***t***. Na primer, ukoliko parametar ***frames***

postavim na 10, to znači da će se vrednost parametra ***t*** povećavati prolaskom svakih 10 frame-ova. Kada ***t*** dođe do vrednosti 1, tada se završava i proces morfinga slike, tj. kada je ***t*** = 1, tada kao rezultat dobijamo sliku koja je identična ciljnoj slici.



*Slika 12. Demonstracija rasta vrednosti promenljive **t** kroz vreme, za **frames** = 10*

Da sam za vrednost promenljive ***frames*** stavila 30, promenljiva ***t*** bi sporije rasla, pa bi i sam proces animacije bio duži i sporiji jer će se algoritam izvršavati u više iteracija.



*Slika 13. Demonstracija rasta vrednosti promenljive **t** kroz vreme, za **frames** = 30*

Sada kada smo definisali šta nam predstavljaju i kako se dobijaju promenljive koje prosleđujemo funkciji za interpolaciju linija možemo i da pogledamo kako ona izgleda i radi.

```
// interpolacija linija
Line* line_interpolation(Line* head_source, Line* head_dest, float t) {

    Line* curr_source = head_source;
    Line* curr_dest = head_dest;

    freeListOfLines(interLines);
    interLines = NULL;

    while (curr_source != NULL) {
        Vec2D start, end;
        start.x = (1 - t) * curr_source->start.x + t * curr_dest->start.x;
        start.y = (1 - t) * curr_source->start.y + t * curr_dest->start.y;
        end.x = (1 - t) * curr_source->end.x + t * curr_dest->end.x;
        end.y = (1 - t) * curr_source->end.y + t * curr_dest->end.y;

        interLines = addLine(interLines, start, end);
        curr_source = curr_source->next;
        curr_dest = curr_dest->next;
    }
    head_inter = interLines;
    return head_inter;
}
```

Slika 14. Funkcija za interpolaciju linija kroz vreme

Cilj ove funkcije je da napravi niz interpolisanih linija u odnosu na prosleđenu jedinicu vremena ***t***, a to postiže na sledeći način. U okviru ove funkcije prolazim kroz nizove linija sa izvorne i ciljne slike i kreiram dve nove tačke, koje bi određivale tu novu interpoliranu liniju. Vrednosti **x** i **y** koordinata za ove dve tačke dobijam tako što mešam uticaj izvorne i krajnje linije u odnosu na to koliko je jedinica vremena proslo. U početku animacije interpolisane linije će biti približnije vrednostima izvorne slike, na sredini procesa te linije će predstavljati srednju vrednost za nizove linija sa izvorne i krajnje slike, dok će pri kraju procesa animacije linije biti mnogo sličnije vrednostima linija sa ciljne slike.

Niz interpolisanih linija ćemo prosleđivati funkciji za algoritam Bajera i Nilija za izvornu sliku kao ciljnu destinaciju za datu iteraciju, tj. to će nam služiti kao parnjak naše izvorne linije za datu iteraciju. Dakle, u toj situaciji, algoritam će težiti pomeraju od izvorne linije ka toj interpolisanoj liniji.

Nasuprot tome, kada algoritam Bajera i Nilija pozivamo za ciljnu sliku, ovaj niz interpolisanih linija će nam predstavljati parnjak u odnosu na našu liniju sa ciljne slike za datu iteraciju. Dakle, u toj situaciji, algoritam će težiti pomeraju od linije sa ciljne slike ka toj interpolisanoj liniji.

Dakle, iako znamo da su linije sa izvorne i ciljne slike pravi parnjaci, mi radi animacije hoćemo da kažemo da linije sa izvorne slike imaju svoj parnjak u nizu interpolisanih linija (koji je samo interpolacija od linije sa izvorne slike, ka liniji sa ciljne slike u nekoj prosleđenoj jedinici vremena). Analogno je i za linije sa ciljne slike, koje teže svojim parnjacima iz niza interpolisanih linija.

Sada kada smo upoznati sa tim koje linije su parovi u datoj iteraciji, možemo pogledati kako bi izgledao pseudokod samog algoritma koji vrši deformaciju za više parova linija.

Prolazak petljama kroz svaki piksel ciljne slike

DSUM = {0,0}

weightsum = 0

Prolazak petljom kroz sve linije sa ciljne slike (***PiQi***)

računanje ***u*** i ***v*** vrednosti na osnovu linije ***PiQi***

računanje ***X'i*** na osnovu linije ***Pi'Qi'*** i vrednosti ***u*** i ***v***

računanje pomeraja ***Di = X'i - X'***

dist = najkraća distanca piksela do linije ***PiQi***

weight = (length^p/(a+dist))^b

DSUM += Di * weight

weightsum += weight

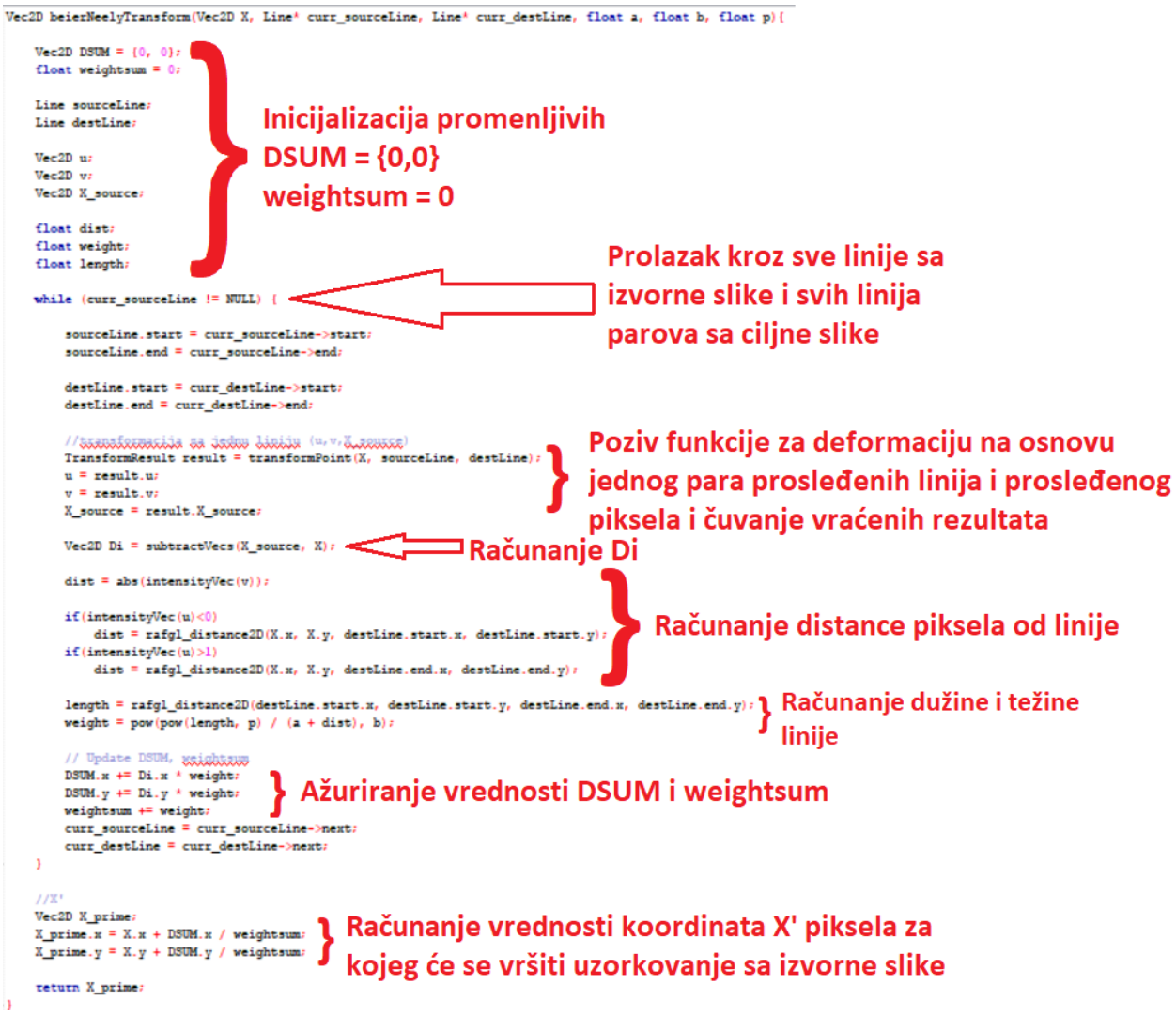
X' = X + DSUM / weightsum

destinationImage(***X***) = sourceImage(***X'***)

Slika 15. Pregled pseudokoda algoritma Bajera i Nilija
Modifikovano iz [1]

Nakon što smo približili ideju algoritma i nakon što smo istakli sve potrebne informacije o podacima koje šaljemo prilikom poziva funkcije koja izvršava srž algoritam Bajera i Nilija, red je da pogledamo kako sama ta funkcija izgleda u okviru implementacije.

S obzirom da u okviru update funkcije implementacije petljama prolazim kroz svaki piksel ciljne slike, funkciju **beierNeelyTransform()** pozivam za svaki taj piksel. To znači da je vrednost promenljive ***X***, zapravo trenutni piksel do koga je program stigao i za koga poziva izvršavanje ovog algoritma. Dakle, početna linija pseudokoda ne pripada samoj funkciji, jer je funkcija pozvana već u okviru petlje koja prolazi kroz sve piksele. Isto tako poslednju liniju pseudokoda izvršavam van ove funkcije.



Slika 16. Funkcija koja izvršava srž algoritma Bajera i Nilija

2.4. Animacija algoritma morfinga slike

Animacija ovog algoritma se pre svega postiže uz pomoć jedinice vremena ***t***, koju sam u prošlom odeljku opisala.

Interpolacija linija koju vršimo korak po korak u odnosu na vrednost promenljive ***t*** nam doprinosi tome da ne vršimo nagli prelaz od početnih linija na krajnje linije u jednom koraku, nego to radimo tako što ćemo se u svakoj iteraciji pomerati malo po malo od linija sa izvorne sliku u linije sa ciljne slike.

Pored toga deo koji doprinosi animaciji samog algoritma je deo u kojem vršim mešanje deformisanih slika u nekom trenutku ***t*** tako što blendujem te dve slike na osnovu parametra ***t***.

Samim tim je u početku više prikazan uticaj prve slike koji se vremenom gubi, dok se, suprotno tome, uticaj druge slike vremenom povećava.

```
for(y = 0; y < h; y++)
{
    for(x = 0; x < w; x++)
    {

        sampled_source = pixel_at_m(source,x,y);
        sampled_dest = pixel_at_m(dest,x,y);

        result.r = (1-t)*sampled_source.r + t*sampled_dest.r;
        result.g = (1-t)*sampled_source.g + t*sampled_dest.g;
        result.b = (1-t)*sampled_source.b + t*sampled_dest.b;

        pixel_at_m(raster, x, y) = result;

    }
}
```

*Slika 17. Blending deformisanih slika u nekom trenutku **t** (prilikom ovog procesa za rezultujući piksel rastera, odnosno našeg rezultata algoritma, dobijamo uticaj obe deformisane slike srazmeran vremenskoj jedinici **t**)*

Napomena: Zbog boljeg pregleda šta se dešava u ovom delu koda, na slici je prikazana verzija u kojoj se deo o animaciji nalazi u zasebnim petljama, iako je u mojoj implementaciji to deo prethodnog prolaska kroz sve piksele.

3. Glava: Vodič kroz implementaciju

Implementaciju ovog algoritma možete naći na github nalogu, na linku: https://github.com/EmilijaDotlic00/Image-Morphing

Nakon što se pokrene kod, u prozoru programa se prikazuje animacija morfinga slika, ali klikom na odgovarajuće tipke sa tastature pokazuju se različite slike vezane za algoritam.

- Klikom na tipku "1" prikazuje sa izvorna slika
- Klikom na tipku "2" prikazuje se cijna slika
- Klikom na tipku "3" prikazuje se izvorna slika sa isctranim linijama koje sam koristila za označavanje karakteristikama i po kojima sam vršila deformisanje slika
- Klikom na tipku "4" prikazuje se ciljna slika sa isctranim linijama koje sam koristila za označavanje karakteristikama i po kojima sam vršila deformisanje slika
- Klikom na tipku "5" animacija se ponovno pokreće, na način da se promenljiva ***t*** koja određuje vremensku jedinicu resetuje na vrednost 0

Ukoliko nije kliknuta nijedna od gore navedenih tipki, animacija teče svojim tokom. Kada se animacija završi dobija se prikaz ciljne slike, sve dok se ne klikne neka od prethodno navedenih tipki.

3.1. Resursi korišteni prilikom implementacije algoritma

Na primeru sa kojim sam radila, prvobitno sam za obe slike našla odgovarajuće parove linija koje definišu karakteristike obe slike (konkretno na priloženom primeru u ovom radu koristila sam 42 para linija), po kojima se vrši deformacija slika.

Slika 18. Prikazuje izvornu i ciljnu sliku nad kojim sam vršila morfing slike, tj. ove slike sam koristila kao primer u sopstvenoj implementaciji.
Izvor: [4]

Slika 19. Prikazuje izvornu i ciljnu sliku sa iscrtanim linijama, koje predstavljaju karakteristike obe slike

3.2. Kratak pregled sadržaja implementacije

3.2.1. Definisanje struktura i kreiranje globalnih promenljivih

U okviru ovog dela koda definisala sam strukture **Vec2D** (koji predstavlja dvodimenzioni vektor, najčešće korišćen za predstavljanje tačaka), **Line** (koja predstavlja usmerenu liniju, definisanu sa početnom i krajnjom tačkom) i **TransformResult** (koja predstavlja samo povratnu vrednost za funkciju koja vrši deformaciju za jedan par piksela, ova struktura se sastoji od promenljivih ***u***, ***v*** i ***X'*** koje se u okviru ove funkcije računaju).

Pored toga, ovde se nalazi i deklarisanje globalnih promenljivih kao što su dinamički nizovi linija sa izvorne slike, interpolisanih linija i linija sa ciljne slike, ali i definisanje svih potrebnih rastera i definisanje teksture.

3.2.2. Pomoćne funkcije

U ovom delu nalazi se implementacija svih pomoćnih funkcija potrebnih za izvršavanje progama ovog koda. Pomoćne funkcije koje se koriste su:

- **Line* addLine(Line* head, Vec2D start, Vec2D end)** – služi za dodavanje elemenata u dinamičku listu linija. Dodaje liniju sa vrednostima tačaka definisanih sa **start** i **end** u dinamičku listu čija je glava prosleđena. Povratna vrednost je glava liste, tj. pokazivač na početak liste u koju je dodat element.
- **void freeListOfLines(Line* head)** – služi za oslobađanje memorije rezervisane za prosleđenu listu.
- **Line* makeSourceList()** – pomoćna funkcija u okviru koje se definišu linije sa izvorne slike koje je potrebno dodati u niz linija sa izvorne slike. Dodavanje vršim pozivom funkcije addLine. Povratna vrednost je glava na kreiran niz linija sa izvorne slike.
- **Line* makeDestList()** – funkcija analogna makeSourceList() funkciji, samo što su u pitanju linije sa ciljne slike, a ne sa izvorne.

- **Line* line_interpolation(Line* head_source, Line* head_dest, float t)** – funkcija prethodno opisana u okviru ovog rada, koja vrši interpolaciju od linija sa izvorne slike ka linijama sa ciljne slike u odnosu na vremenski faktor ***t***.
- **Vec2D perpendicular(Vec2D vec)** – za prosleđeni vektor vraća njemu normalan vektor
- **Vec2D addVecs(Vec2D vector1, Vec2D vector2)** – sabiranje prosleđenih vektora i vraćanje njihove sume.
- **Vec2D subtractVecs(Vec2D vector1, Vec2D vector2)** – oduzimanje drugog vektora od prvog i vraćanje njihove razlike.
- **Vec2D multiplyVecs(Vec2D vector1, Vec2D vector2)** – množenje prosleđenih vektora i vraćanje njihovog proizvoda.
- **Vec2D divideVecByScalar(Vec2D vector, float scalar)** – deljenje prosleđenog vektora skalarom i vraćanje njihovog količnika.
- **float intensityVec(Vec2D vector)** – za računanje intenziteta prosleđenog vektora.
- **TransformResult transformPoint(Vec2D X_dest, Line sourceLine, Line destLine)** – funkcija prethodno opisana u okviru ovog rada, koja vrši deformisanje prosleđene pozicije piksela u odnosu na prosleđen par linija.
- **Vec2D beierNeelyTransform(Vec2D X, Line* curr_sourceLine, Line* curr_destLine, float a, float b, float p)** – funkcija prethodno opisana u okviru ovog rada, koja vrši deformaciju pozicije prosleđenog piksela u odnosu na sve linije i kao povratnu vrednost vraća poziciju sa koje treba da se uzorkuje prosleđeni piksel.

3.2.3. Inicijalizacija (init)

U okviru ovog dela alocira se prostor za rastere i teksturu koje ćemo koristiti, ali vrši se i učitavanje slika nad kojim ćemo raditi morfing. Takođe, u ovom delu pozivam funkcije za kreiranje niza linija sa izvorne i ciljne slike.

Pored toga preko rastera u koji sam prekopirala originalne slike iscrtavam i linije karakteristika, kako bih mogla da prikažem na osnovu čega se vrši deformacija slika.

Funkcija za inicijalizaciju se poziva samo jednom i to na početku programa.

3.2.4. Ažuriranje stanja rastera (update)

U update funkciji ćemo menjati kako će se stvari dešavati u našoj aplikaciji. Ova funkcija se poziva sve dok traje program naizmenično sa funkcijom render. U okviru update funkcije vršimo promene stanja koje zatim preko render funkcije prikazujemo na ekran.

U okviru ove funkcije obrađujem unos sa tastature, na način da preko flagova koje su globalne promenljive pamtim koji od rastera treba da prikažem u datom trenutku. Ove flagove ću kasnije, u render funkciji, koristiti za prikaz željenog rastera. Izuzetak predstavlja klik tipke „5“, koji ne utiče na prikaz rastera, već samo resetuje animaciju algoritma.

```
if(game_data->keys_down[RAFGL_KEY_1])
    original=1;
else
    original=0;
if(game_data->keys_down[RAFGL_KEY_2])
    destination=1;
else
    destination=0;
if(game_data->keys_down[RAFGL_KEY_3])
    original_with_lines=1;
else
    original_with_lines=0;
if(game_data->keys_down[RAFGL_KEY_4])
    destination_with_lines=1;
else
    destination_with_lines=0;
if(game_data->keys_down[RAFGL_KEY_5])
    c=0;
```

Slika 20. Obrada unosa sa tastature

Takođe, u ovoj funkciji vršim inicijalizaciju parametara koje koristim u algoritmu, kao što su ***a***, ***b***, ***p*** i ***t.*** Nakon toga, prolazim kroz sve piksele ciljne slike i za svaku poziciju pozivam izvršavanje funkcije **beierNeelyTransform()** i za izvorne i interpolisane linije, i za ciljne i interpolisane linije. Ukoliko kao rezultat te funkcije dobijem poziciju piksela koji se nalazi unutar slike - vršim uzorkovanje, dok ukoliko se poziciija nalazi van okvira slike - vršim bilinearnu interpolaciju u odnosu na okolne piksele. Na ovaj način kada se pronađe odgovarajuća vrednost za trenutni piksel izvorne i ciljne slike, nama samo preostaje da odradimo spajanje piksela sa te pozicije (uz pomoć image blendinga) izvorne i ciljne slike. Tako da je samo još potrebno izvršiti interpolaciju između vrednosti piksela sa izvorne i ciljne slike u odnosu na trenutak ***t***. Nakon toga mi smo zapravo dobili rezultujući piksel naše sekvence. Kada prođemo kroz sve piksele, mi smo zapravo dobili konačnu sekvencu.

3.2.5. Renderovanja promenjenog rastera (render)

Render funkcija nam služi za iscrtavanje stanja definisanih u okviru update funkcije. Samim tim, ova funkcija se poziva nakon izvršavanja update funkcije.

U okviru ove funkcije, na osnovu globalnih flagova proveravamo koji raster je potrebno prekopirati u teksturu koja će se na kraju i prikazati na ekranu.

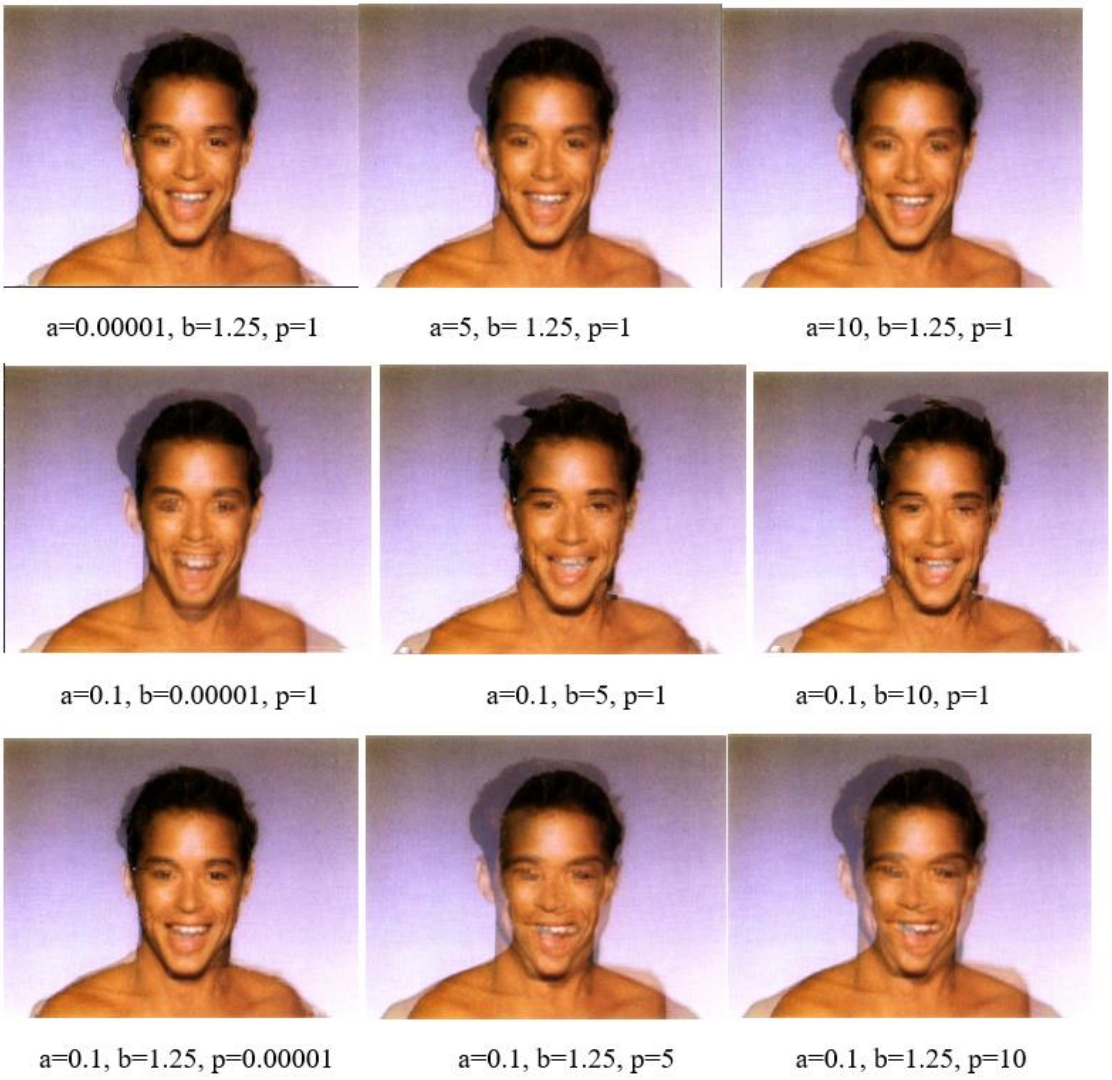
3.2.6. Oslobađanje memorije (cleanup)

U okviru ovog dela vršim oslobađanje memorije za sve promenljive za koje sam u inicijalizaciji zauzela memoriju. Ovaj deo se poziva samo jednom i izvršava na kraju programa.

4. Glava: Analiziranje parametrizacije algoritma

S obzirom da nije moguće iz prve pogoditi najbolje parametre za primenu algoritma, prošla sam kroz fazu isprobavanja istih, dok nisam pronašla one koji su mi delovali najbolje. Prilažem sledeće slike da bih vam približila kako izgleda promena samo po jednom parametru i na koji način ona utiče na izgled sekvence algoritma.

Sekvence iz animacije koje vam prilažem su sve iz istog trenutka procesa primene algoritma, tačnije prikazana je središnja sekvenca iz animacije sa datim parametrima. Dakle, parametar ***t*** je na svim dole navedenim slikama imao vrednost 0.5.



*Slika 21. Ovaj prikaz za cilj ima da pokaže kako menjanjem samo jednog parametra možemo dobiti potpuno različite rezultate, te je po redovima prikazano menjanje samo jedne komponente U prvom redu menjala sam vrednosti komponente **a**, u drugom redu komponente **b**, a u trećem redu komponente **p***

Nadam se da vam je ovaj prikaz bolje pojasnio kakvi se sve artefakti mogu dobiti na sekvencama animacije samo uz pomoć jedne pogrešno setovane vrednosti.

Naravno, razlika u promeni ovih parametara je mnogo uočljivija kada se gleda ceo proces morfinga slike, tako da ukoliko to želite da vidite, savetujem vam da u implementaciji samo promenite ove parametre, te da gledate kakve sve različite deformacije dobijate tokom samog procesa. Za potrebe prikazivanja primera u radu odlučila sam da to uradim samo za središnju sekvencu. Takođe, pravila sam velike razlike u menjanju parametara, kako bi promene bile lakše uočljive.

Promena vrednosti parametra ***frames*** utiče na brzinu rasta promenljive ***t*** između 0 i 1 što određuje brzinu izvršavanja algoritma. U suštini vrednost koja je navedena za promenljivu ***frames*** predstavlja broj sekvenci koje će se generisati prilikom animacije programa. Dakle, što je manja vrednost parametra ***frames***, to će prelaz biti nagliji, a sama animacija sačinjena od manje sekvenci, dok će sa većim vrednostima za parametar ***frames*** prelaz biti sve blaži, jer će se i sama animacija sastojati od više sekvenci.

5. Glava: Primena

Kada se koristi na pravi način morfing slike može stvoriti iluziju da se fotografije ili računarom generisane slike mogu transformisati na fluidan, a često i nadrealističan način. Samim tim ova tehnika je pogodna za korišćenje u različitim filmovima, animacijama, ali i za potrebe različitih modelovanja i renderovanja slike koje se transformišu tokom vremena.

Ukoliko se primena vrši nad slikama koje imaju značajne karakteristike pozadine potrebno je i njih označiti karakteristikama, kako ne bi dolazilo do nekontrolisanog izobličavanja.

Kako bi se postigla primena ove tehnike na snimcima, umesto samo između dve statične slike, potrebno je označiti karakteristike svih ključinih kadrova sa oba snimka. Na ovaj način su napravljene i sekvence u spotu Majkla Džeksona “Black or White”, kada je i prvi put u okviru nekog spota primenjena morfološka transformacija. Takođe, mnogi Diznijevi crtaći su napravljeni uz pomoć morfološke transformacije radi ubrzanja proizvodnje.

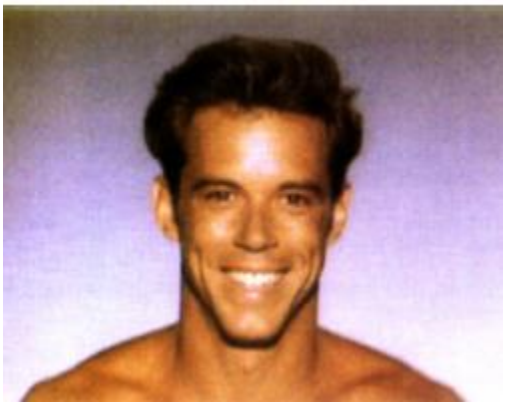
Primenu ovog algoritma pronalazimo u različitim animacijama i vizuelnim efektima kako u reklamama, slikama, filmovima, tako i u video igrama, na primer za transformaciju jenog karaktera u drugi. Takođe česta je primena prilikom tranzicija između scena.

Pored ovih, naizgled očekivanim primenama, ova tehnika se često koristi i u drugačijim oblastima. Na primer u medicinskim istraživanjima, za vizualizaciju promena u anatomiji ili obliku organa tokom vremena. Ovo je korisno u praćenju razvoja ili promena u ljudskom telu. Takođe, u forenzici i medicinskoj rekonstrukciji, morfing slike se koristi za rekonstrukciju lica osoba na osnovu starih fotografija ili skica kako bi se dobio što tačniji prikaz izgleda.

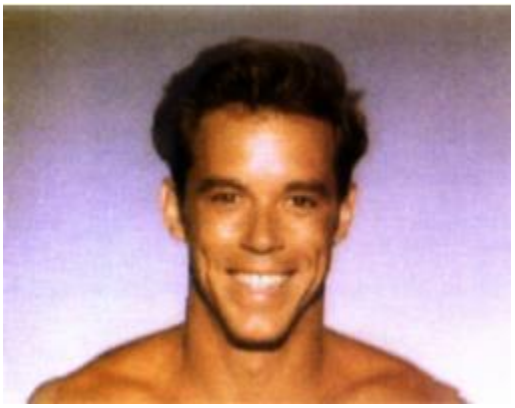
6. Glava: Dobijeni rezultati

Na narednih 11 fotografija se nalaze sekvence animacije poređene redom kako protiče vreme. Vrednost promenljive ***t*** definiše koji je trenutak animacije prikazan na slici. Na sekvenci sa vrednosti ***t*** = 0 nalazi se originalna slika, dok se na sekvenci sa vrednošću ***t*** = 1 nalazi krajnja slika. Sve vrednosti promenljive ***t*** koje se nalaze između 0 i 1 su kombinacija originalne i krajnje slike u nekom trenutku. Animacija prikazana na primerima je odrađena sa vrednostima parametara koji su mi se pokazali najbolje, a to su:

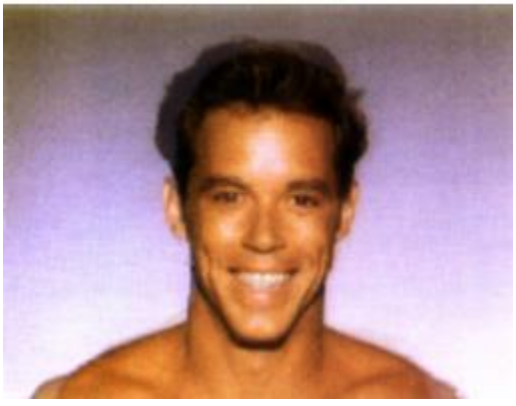
- ***a*** = 0.1
- ***b*** = 1.25
- ***p*** = 1



a) *t* = 0.0



b) *t* = 0.1



c) *t* = 0.2

d) *t* = 0.3

e) ***t*** **= 0.4**

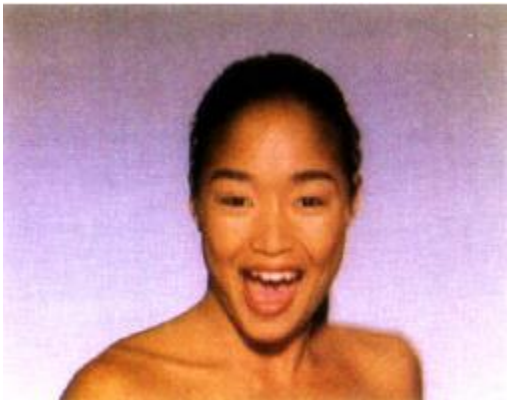
f) ***t*** **= 0.5**

g) ***t*** **= 0.6**

h) ***t*** **= 0.7**

i) ***t*** **= 0.8**

j) ***t*** **= 0.9**



k) *t* = 1.0

Slika 22. Predstavlja sve sekvence iz animacije dobijene primenom morfinga slika sa algoritmom Bajera i Nilija, gde su sekvence poređane redom od sekvence pod a) do sekvence pod k), dakle vremenom smo prolazili između ovih sekvenci
Sekvenca pod a) je zapravo izvorna slika od koje krećemo, dok je sekvenca pod k) ciljna slika do koje želimo da stignemo primenom ovog algoritma.

Zaključna razmatranja

U ovom radu proučavala sam morfing slike i algoritam Bajera i Nilija, koji za osnovnu ideju i cilj imaju specificiranje deformacija koja izobličava prvu sliku u drugu, dok njena inverzija izobličava drugu sliku u prvu, a zatim da se od tih deformisanih slike kreira postepeni prelaz. Ovu tehniku sam implementirala u svom softeru u okviru programskog jezika C, za koji nisam našla pređašnje implementacije.

Smatram da ovakav pristup daje animatoru visok nivo kontrole nad vizualnim efektom, zbog mogućnosti jednostavnog definisanja karakteristika slike po kojim bi se vršila morfološka transformacija. Takođe, smatram da bi se razvijanjem korisničkog interfejsa animatorima značajno olakšao proces obeležavanja karakteristika linijama, jer bi direktno klikom miša mogli da ih iscrtavaju. Verujem da bi ovo doprinelo lakšem korišćenju programa, te da bi i početnici mogli sa lakoćom da ga koriste. Samim tim ovaj dodatak smatram značajnim za dalji razvoj.

Pored toga, ovaj program bi se mogao unaprediti automatizacijom obeležavanja karakteristika uz pomoć mašinskog učenja. To bismo mogli postići dodavanjem algoritma koji bi pronalazio sličnosti između slika i obeležavao ih linijama karakteristika. Kako bi ovaj dodatak bolje radio treba odabrati izvorne i ciljne slike s sličnom kompozicijom, uključujući boju pozadine, proporcije objekta i vrstu objekta koji se pojavljuje na slici.

Nadam se da sam vam približila pristup morfingu slike i Bajer-Nilijevom algoritmu, te da sam vam pružila informacije potrebne za njihovo dublje razumevanje.

Literatura

[1] Feature-Based Image Metamorphosis, Thaddeus Beier, Shawn Neely, CA, 1992

[2] Computer Graphics Image warping and morphing. 紀明德 mtchi@cs.nccu.edu.tw Department of Computer Science, National Chengchi University

[3] Feng Zhu, Image Morphing with the Beier-Neely Method, B.Sc., Northwestern Polytehnical University, 2013

[4] Image Morphing, Computer Science University of Toronto

[5] CS 195-G: Face Morphing, Evan Wallace

[6] Programming Assignment 1: Morphing, Computer Graphics 2, 15-463, Carnegie Mellon University, 2001.

[7] Feature-Based Volume Metamorphosis, Apostolos Lerios, Chase D. Garfinkle, and Marc Levoy, Proceedings of SIGGRAPH '95 (Los Angeles, CA, August 6-11, 1995). In Computer Graphics Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH, pp. 449-456.

[8] CSC5280 Project 1: Image Morphing, Wei Zhang, Dept. of Information Engineering, The Chinese University of Hong Kong

[9] Morphing, Swarup Deepika Jagmohan, 2002.

[10] CSC2530 - Visual Modeling Project #2 - View Morphing Implementation Details and Results, George ElKoura, Sam Hasinoff, Computer Science University of Toronto, 2001

[11] Motion and Feature-Based Video Metamorphosis, Robert Szewczyk, Andras Ferencz, Henry Andrews, Brian C. Smith, Seattle, 1997

[12] Morphing, COSC450 Assignment 2, University Otago New Zealand, 2016.

[13] ELEC 539: Image Morphing, Roger Claypoole, Srikrishna Bhashyam, Kevin Kelly, Rice University, Houston, 1997.

[14] Computational Photography: Face Morphing and Modelling a Photo Collection, Aneesh Akella, University of California

[15] CS559: Image warping, Li Zhang, DigiVFX

[16] EE368 Project, A study on face morphing algorithms, Stanford

Biografija

Emilija Dotlić, rođena 24. avgusta 2000. u Pančevu, odrasla u Kovinu. Završila je osnovnu školu “Jovan Jovanović Zmaj” u Kovinu, kao i srednju školu “Gimnazija i ekonomska škola Branko Radičević”, gde je bila opšti smer gimnazije. Nakon toga upisuje Računarski fakultet, gde tokom četvrte godine studija postaje i asistent na predmetima Računarska grafika i Uvod u bioinformatiku.