SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1915

POSTUPCI ODREĐIVANJA KOSTURA MODELA NA OSNOVI POLIGONALNOG MODELA

Robert Mrkonjić

Zagreb, lipanj 2011.

SADRŽAJ:

1. UVOD

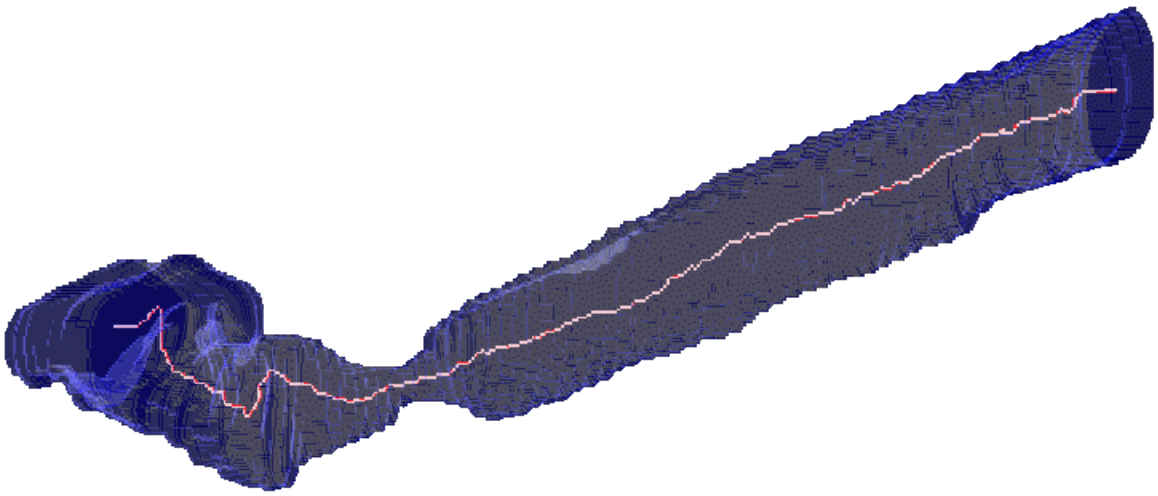
Postupak izvlačenja kostura iz poligonalnog modela je tehnika koja je razvijena tek nedavno te proizlazi iz vječne čovjekove želje da otkrije nepoznato. U ovom radu će se čitatelj upoznati s osnovnim tehnikama izvlačenja kostura.

Na početku rada se definira pojam kostura u skeletonizaciji i njegova osnovna svojstva. Nakon toga slijedi poglavlje gdje će se čitatelj upoznati s osnovnim klasama skeletonizacijskih algoritama gdje se uz svaku klasu daje po nekoliko primjera algoritama za dotičnu klasu. Nakon toga slijedi poglavlje gdje će se pokazati kako implementirati algoritam stanjivanja (eng. *thinning*). Ukratko će se pokazati glavni implementacijski detalji te prednosti i nedostaci pojedinih implementiranih algoritama. Na kraju rada će se pokazati rad implementiranog algoritma nad konkretnim primjerom. Čitatelj će vidjeti kako se ulazni primjer postepeno "ljušti" do kostura.

Nakon čitanja ovog rada čitatelj će posjedovati znanje o osnovnim algoritmima skeletonizacije i znanje da implementira algoritam po volji iz ovog rada.

2. DEFINICIJA KOSTURA U SKELETONIZACIJI

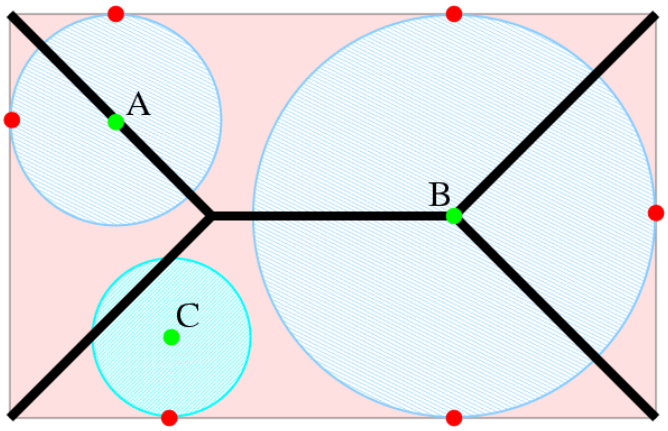
Cilj skeletonizacijskih algoritama je izvlačenje pojednostavljenog prikaza modela koji sa svojim značajkama predstavlja opći oblik objekta nad kojim se obavlja skeletonizacija, tj. određuje kostur modela.



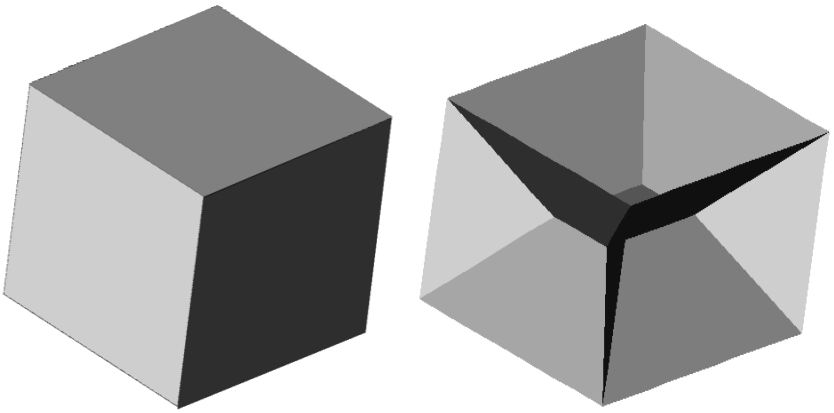
Slika 2.1. prikaz skeletonizacije na 3D cjevastom objektu

Pojam kostura, u kontekstu skeletonizacije, je prvi put predstavljen od Harry-ja Bluma kao rezultat transformacije medijalne osi (*MAT*) ili transformacije simetrične osi (*SAT*). MAT određuje najbližu graničnu točku ili više njih za svaku točku koja se nalazi unutar objekta. Unutrašnja točka pripada kosturu ako ima barem dvije granične točke koje su blizu nje. Možemo pojednostaviti ovu definiciju ako ju usporedimo s požarom na livadi. Ako zapalimo granicu livade, požar se širi dok se plameni ne sastanu i međusobno ugase što će na kraju ocrtati kostur livade.

Formalna definicija kostura kaže da je kostur mjesto središta svih maksimalnih upisanih hiper-sfera (npr. diskovi i lopte u 2D i 3D) gdje je upisana hiper-sfera maksimalna ako nije pokrivena nekom drugom hiper-sferom.

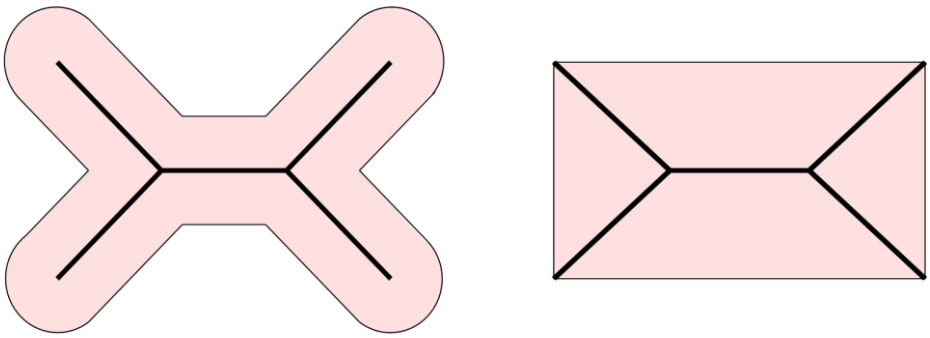


Slika 2.2. kostur pravokutnika, točke A i B pripadaju kosturu dok C ne pripada



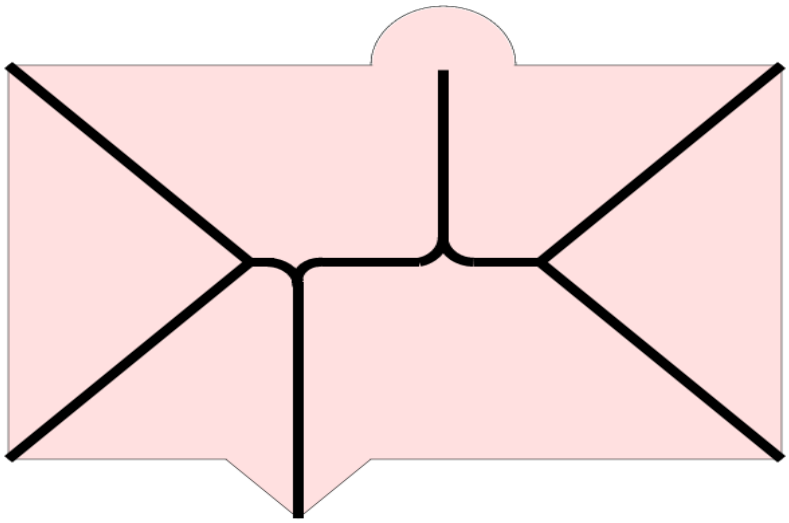
Slika 2.3. kostur kocke

Kako vidimo na slikama 2.2. i 2.3., definicija ne ovisi o broju dimenzija te se može primijeniti na 2D i 3D modele. Zbog sam definicije, ponekad se dobije isti kostur za različite modele.



Slika 2.4. prikaz istog kostura različitih objekata

Upitna je i stabilnost kostura jer se ponekad za modele sa čudnim i neravnomjernim oblikom dobije iskrivljeni kostur.



Slika 2.4. prikaz iskrivljenog kostura

3. SKELETONIZACIJSKI ALGORITMI

Kako samo ime kaže, skeletonizacijski algoritmi su oni algoritmi koji računaju kostur nekog objekta. U današnje vrijeme postoji mnogo algoritama koji obavljaju skeletonizaciju, ali ćemo se zadržati na najosnovnijim algoritmima koji se primjenjuju već duže vrijeme.

Algoritmi se klasificiraju u tri osnovne klase

1. Metode temeljene na udaljenosti (eng. *distance-based methods*)
2. Metode temeljene na poligonima (eng. *polygon-based methods*)
3. Topološko stanjivanje (eng. *topological thinning*)

Iako kažemo da izvlačimo kostur, algoritmi rade aproksimaciju stvarnog kostura, pri čemu je važno da algoritmi ispune slijedeće uvjete:

1. mora se očuvati topologija objekta
2. kostur mora biti unutar objekta i invarijantan na važnije geometrijske tranformacije (npr. translacija, rotacija, skaliranje)

Tablica 3.1. prikaz klasa metoda i uvjeta koje zadovoljavaju

KLASA	UVJET 1.	UVJET 2.
Metode temeljene na udaljenosti (Distance-based methods)	Ne	Da
Metode temeljene na poligonima (Polygon-based methods)	Da	Da
Topološko stajivanje (Topological thinning)	Da	Ne

U nastavku slijedi obrada predstavnika za svaku klasu.

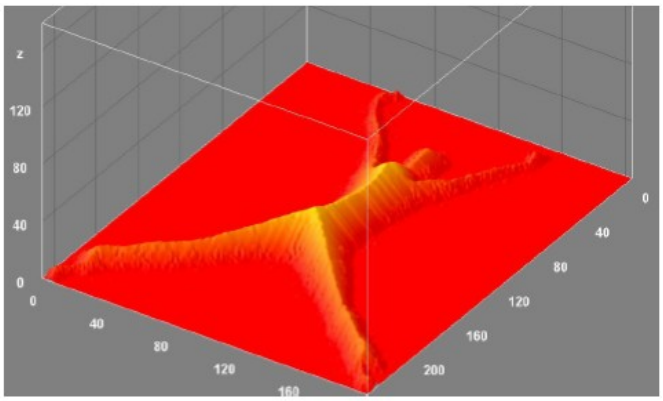
3.1. ALGORITAM TRANSFORMACIJE UDALJENOSTI

Ovaj algoritam je predstavnik metoda temeljenih na udaljenosti. Sastoji se od tri koraka:

1. Izvorna binarna slika se prebacuje u oblik gdje su vidljivi značajni (eng. *feature*) i neznačajni (eng. *non-feature*) elementi, pri čemu se značajnim elementom smatra element koji se nalazi na granici slike (element je dio granice slike).
2. Generira se mapa udaljenosti gdje svaki element označava udaljenost do najbližeg značajnog elementa.
3. Nalaze se lokalni ekstremi koji predstavljaju kostur slike.

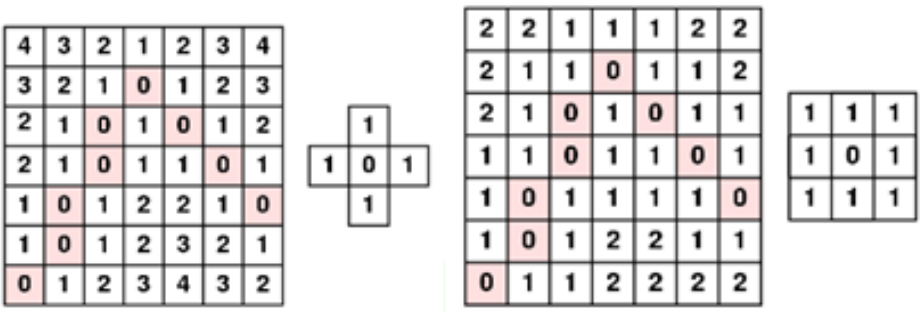
Slika 3.1.1. primjer algoritma transformacije udaljenosti, izvorna slika lijevo, mapa udaljenosti desno

Za početak ćemo vidjeti jednostavan primjer rada algoritma. Rubovi mape udaljenosti su tamni (vidi sliku 3.1.1. desno) jer su to značajni elementi, dok su elementi sve svjetliji što su dalji od ruba slike. Elementi koji su najviše udaljeni od ruba daju naznaku kostura slike. Mapu udaljenosti možemo promatrati tako da elementi svoju udaljenost gledaju kao visinu. Kako se udaljenosti prikazuju u intervalu od 0 (tamnije) na više (svjetlije), svaki element dobro određuje svoju visinu te se na ovaj način bolje vidi kostur slike koji će se još posebnim algoritmom izvaditi.



Slika 3.1.2. primjer algoritma transformacije gdje element određuje svoju visinu

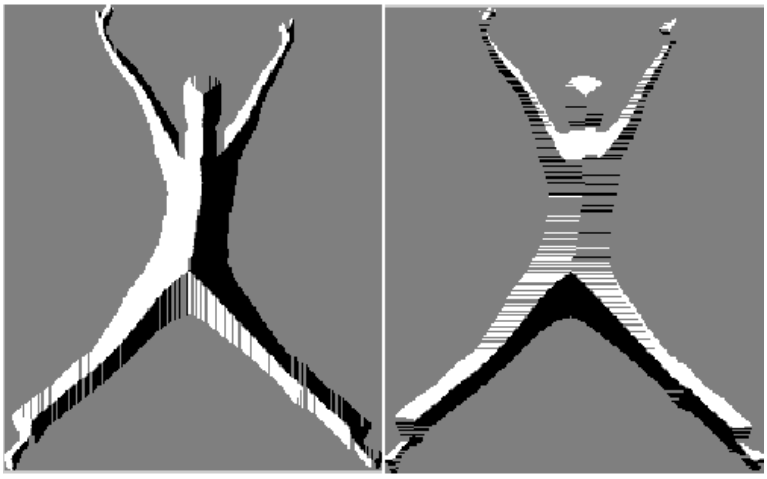
Dok je određivanje značajnih elemenata intuitivno i samo po sebi lagano za provesti, određivanje udaljenosti pojedinih piksela se može napraviti na više načina. Mogu se koristiti bilo kakve metrike, ali su uobičajene u praksi Euklidova i Manhattan metrika zbog svoje lake implementacije. Odabirom pravilne metrike možemo odrediti svojstva algoritma, npr. ako nam je važna brzina, koristiti ćemo Manhattan metriku, dok ćemo koristiti Euklidovu ili šahovsku metriku za povećanje preciznosti algoritma (vidi sliku 3.1.3.).



Slika 3.1.3. mape udaljenosti, Manhattan udaljenost lijevo, šahovska udaljenost desno

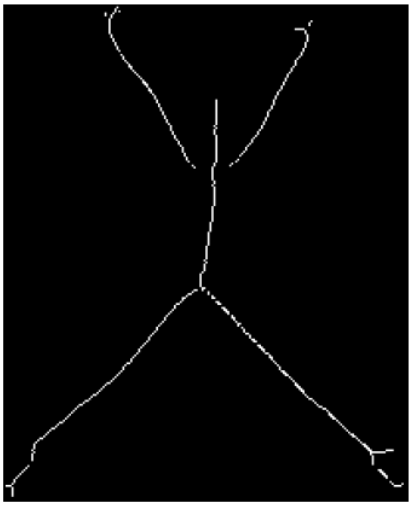
Nakon određivanja mape udaljenosti, slijedi potraga za lokalnim ekstremima ili "grebenima" (eng. *ridge*). Postoje mnogi postupci određivanja tih grebena. Česta je upotreba diferencijalnog računa, ali kako su takvi postupci teški za implementirati i zahtijevaju dodatno znanje, u ovom radu ću se okrenuti jednostavnoj izvedbi baziranoj na analizi gradijenta ([2] *Chang S., Extracting Skeletons from Distance Maps*). Algoritam analizira retke i stupce mape udaljenosti u potrazi za specifičnim uzorcima. Prvo se gledaju redci s lijeva na desno na način da se trenutačni piksel uspoređuje s njemu desnim i ako on ima

veću vrijednost pohranjenu, algoritam vraća "+", ako je vrijednost susjednog piksela jednaka vraća se "0" , i ako je vrijednost manja vraća se "-". Često se "+" vizualiziraju s bijelom, "0" sa sivom i "-" sa crnom bojom. Nakon obrade redaka, rezultat se sprema i počinje obrada stupaca odozgo prema dolje.



Slika 3.1.4. primjer rada algoritma za određivanja grebena, slika nakon obrade redaka lijevo, slika nakon obrade stupaca

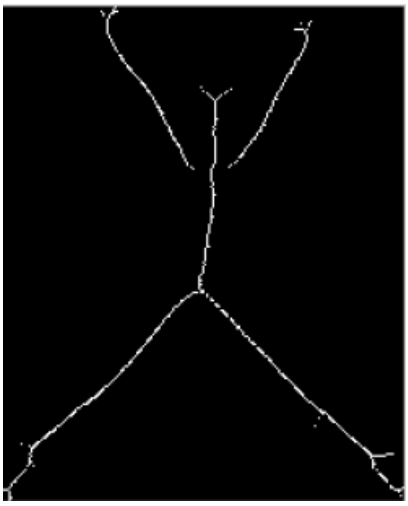
Nakon prolaska po retcima i stupcima, kreće se u potragu za pokazateljima grebena. Očiti uzorak grebena je "+-" (granica bijele i crne boje) koji se pojavljuje samo na grebenima. Moguće je da na grebenima dvije susjedne točke imaju istu udaljenost do ruba, pa će se uključiti uzorak "+0-" kao jaki pokazatelj grebena.



Slika 3.1.5. izvučeni kostur nakon određivanja grebena

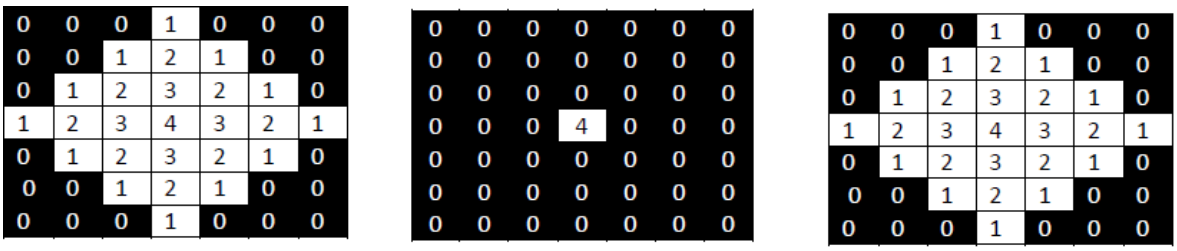
Izvučeni kostur dobro opisuje oblik slike, ali se također vide rupe koje ne postoje u kosturu. Kako bi se poboljšao ovaj kostur, moguće je proširiti skup

uzoraka s "+0" i "-0" kao slabe pokazatelje grebena. Uzorak "+0" može biti početak serije "0"-a koji ako završavaju na "0-" ukazuju na greben. Poboljšanje je naočitije na području glave zbog već prije navedene serije proširenih uzoraka.



Slika 3.1.6. izvučeni kostur nakon proširenja skupa uzoraka

Dobro svojstvo ovog algoritma je da čuva informaciju o širini pomoću koje je moguće rekonstruirati izvornu sliku iz kostura slike ako nam je poznata metrika koja je korištena. Ako smo koristili Manhattan metriku, crtamo piramidalne oblike s radijusom koji se nalazi u točkama kostura mape udaljenosti. To se najbolje vidi ako uzmemo kao primjer samu piramidu, čija je sredina udaljena za četiri od rubova. Nakon obrade algoritmom dobijemo kostur od samo jednog piksela sa sačuvanom širinom. Kako bi se rekonstruirala slika, crtamo piramidu oko piksela sa radijusom četiri.



Slika 3.1.7. primjer rekonstrukcije piramide

U slučaju da smo koristili neku drugu metriku, trebao bi se koristiti neki drugi oblik pogodan za rekonstrukciju, npr. za Euklidovu metriku pogodne su kružnice.

Slika 3.1.8. primjer rekonstrucije slike 3.1.1., izvorna slika lijevo, rekonstruirana slika u sredini, pogreške rekonstrukcije desno

3.2. ALGORITAM RAČUNANJA VORONOI DIJAGRAMA

Ovaj algoritam je predstavnik klase metoda temeljenih na poligonima. Prije nego što se objasni veza Voronoi dijagrama i ekstrakcije kostura, definirati ćemo radi boljeg razumijevanja Voronoi dijagram. Nakon toga, moći će se prikazati ekstrakcija kostura iz Voronoi dijagrama.

Voronoi dijagram je geometrijski koncept koji se temelji na blizini diskretnih skupova podataka. Uobičajeno se kaže da je Voronoi dijagram unija rubova između Voronoi regija definiranih kao skup točaka koje su bliže jednom elementu prostora (npr. u slici 3.2.1. element prostora je točka) nego nekom drugom. Moguće je i formalno definirati na sljedeći način:

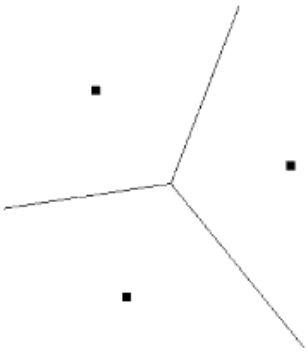
Voronoi regija jednog elementa e iz skupa elemenata E, oznake VR(e), je skup točaka koje su bliže elementu e nego nekom drugom elementu iz E

$$VR(e) = \{\, p \in \mathbb{R}^n \mid d(p,e) \leq\ d(p,e^{'}), \forall e^{'} \neq\ e, e^{'} \in E \,\}$$

gdje je d uobičajena Euklidova udaljenost između dva elementa. Voronoi dijagram iz E, oznake VD(E), definira se kao unija granica Voronoi regija

$$VD(E) =\ \cup_i \delta VR(e_i)$$

gdje je simbol δ granica elementa.



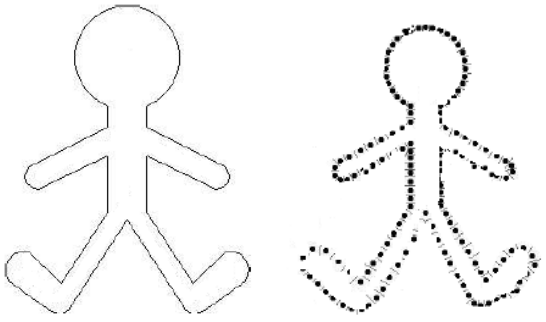
Slika 3.2.1. Jednostavan Voronoi dijagram triju točaka

Ukoliko promotrimo primjer 3.2.1., vidimo tri Voronoi regije čije su granice ravne linije. Voronoi regije se nalaze oko elemenata prostora koji su u ovom slučaju točke. Moguće je uzeti i druge oblike kao elemente prostora, npr. ravnu liniju koja može predstavljati jedan element ili skup konačnih, različitih i povezanih elemenata beskonačno blizu jedan drugom. To je važno svojstvo Voronoi dijagrama koje omogućuje grupiranje više točaka u valjani element. Ovo svojstvo dolazi do izražaju kod izvlačenja kostura.

Postoje brojne metode za izvlačenje Voronoi dijagrama pri čemu je, u zadnje vrijeme, najpopularnija inkrementalna metoda. Metoda počinje s jednostavnim dijagramom s tri elementa. Naknadno se dodaju elementi i pri svakom dodavanju se postojeći dijagram modificira.

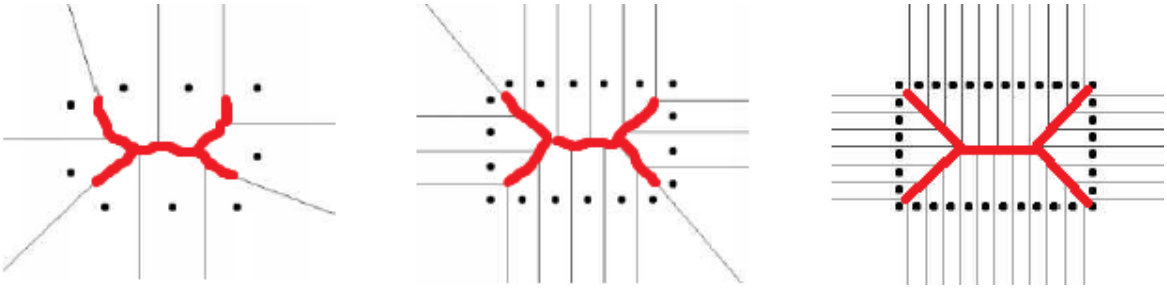
Izvlačenje kostura pomoću ove metode temelji se na sljedećim koracima:

1. konverzija rubova modela u elemente prostora (u našem slučaju točke)
2. računanje Voronoi dijagrama
3. izvlačenje kostura iz Voronoi dijagrama



Slika 3.2.2. primjer izvlačenje elemenata prostora iz modela

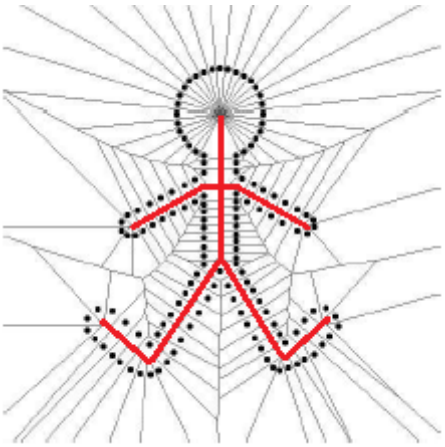
Potrebno je rubove modela pretvoriti u što veći broj točaka jer porastom broja točaka i samim time porastom gustoće točaka dobivamo precizniji kostur modela.



Slika 3.2.3. manje precizan kostur lijevo, srednje precizan u sredini i jako precizan kostur desno istog modela

Nakon što se odrede točke, provodi se računanje Voronoi dijagrama upotrebom nekih od metoda za izračun. Izračunom Voronoi dijagrama preostaje još izvlačenje kostura iz dijagrama. U današnje vrijeme postoje mnoge metode gdje svaka metoda ima svoje pozitivne i negativne značajke. Na programeru je da odluči koju da koristi.

Ovaj postupak, za razliku od algoritma transformacije udaljenosti, daje jako dobre rezultate s malo šuma, ali je zato jako skup i samim time spor proces, posebno za velike i složene modele.



Slika 3.2.4. kostur slike 3.2.2.

3.3. ALGORITAM STANJIVANJA

Ovaj algoritam je najbolji primjer H. Blum-ove analogije paljenja livade. Kako gorenjem livade od ruba prema unutrašnjosti ostaje samo kostur, tako se primjenom algoritma stanjivanja (eng. *thinning*) odstranjuju suvišni pikseli od ruba objekta sve dok na kraju ne ostane samo kostur objekta.

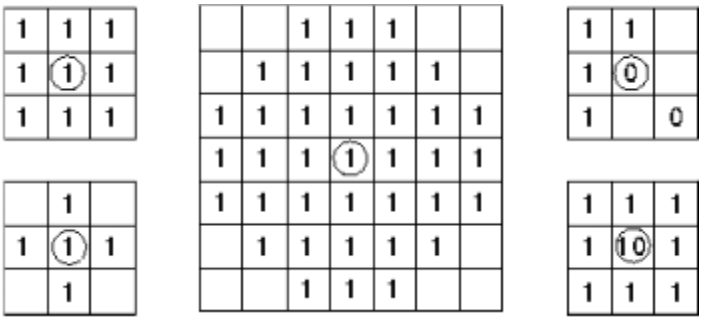
Za stanjivanje su nam potrebna dva skupa podataka:

1. ulazna slika u pogodnom obliku
2. element za strukturiranje (eng. *structuring element*)

Pogodan oblik za sliku je u binarnom ili crnom-bijelom obliku (eng. *greyscale*) zbog lakšeg obavljanja operacija nad pojedinim pikselima. Često se koristi Sobel operator za dobivanje slike u pogodnom obliku, kod koje se naknadno vrijednosti piksela skaliraju na neku nižu vrijednost (eng. *thresholding*).

Slika 3.3.1. primjer dobivanja slike u pogodnom obliku pomoću Sobel operatora, izvorna slika gore lijevo, Sobel operatorom dobivena slika gore desno, skalirana slika dolje

Element za strukturiranje se sastoji od uzorka određen koordinatama diskretnih točaka u odnosu na neko podrijetlo (eng. *origin*). Uobičajeno se koristi kartezijev sustav za prikaz koordinata pa se zato element može prikazati kao mala slika u kvadratnom polju. Podrijetlo se često smješta u centar slike.



Slika 3.3.2. primjer raznih elemenata za strukturiranje

Kao što se vidi na slici 3.2.2., moguće je da točke unutar elementa imaju neku vrijednost. Često se za postupak stanjivanja uzimaju elementi kod kojih "1" predstavljaju piksele u prvom planu, "0" predstavljaju piksele u pozadini i praznine predstavljaju piksele koje nas ne zanimaju tj. nad njima se ne provodi nikakva operacija.

Postupak stanjivanja se svodi na translatiranje podrijetla elementa za strukturiranje na svaki piksel slike i uspoređivanje uzorka s pikselima slike koji se nalaze ispod uzorka. Ako pikseli uzorka točno odgovaraju pikselima slike ispod uzorka, onda se piksel slike koji se nalazi ispod podrijetla briše tj. postavlja se u pozadinu, u suprotnom se ne radi ništa s njim. Kako vidimo, izbor elementa za strukturiranje je jako važan jer određuje u kojem kontekstu se stanjivanje koristi.

Algoritmi stanjivanja se mogu podijeliti na dvije klase:

1. iterativni algoritmi
2. ne-iterativni algoritmi

pri čemu su algoritmi druge klase brži, ali ne daju uvijek najbolje rezultate.

U nastavku će se obraditi poznati predstavnici iterativnih algoritama zbog svoje jednostavnosti i rasprostranjenosti.

3.3.1. ITERATIVNI ALGORITMI STANJIVANJA

Ovi algoritmi stanjivanja koriste postupak opisan neposredno iznad u više iteracije sve dok se ne izvuče kostur objekta. Koriste se različiti uzorci za elemente strukturiranja prolazeći pritom po slici slijeva na desno i odozgo prema dolje brišući na putu vanjske slojeve piksela. Zbog pouzdanosti i učinkovitosti ovih algoritama, jako su popularni u primjeni, posebno Stentiford i Zhang-Suen algoritmi koji koriste brojeve povezanosti (eng. *connectivity numbers*) za označavanje i brisanje piksela.

Broj povezanosti mjeri koliko različitih objekata je povezano s nekim promatranim pikselom. Broj povezanosti se može izraziti sljedećom formulom:

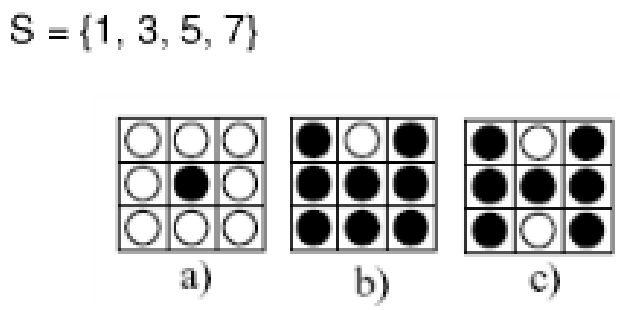
$$C_n = \sum_{k \in S} N_k - (N_k \cdot N_{k+1} \cdot N_{k+2})$$

gdje je N_k boja od osam susjednih piksela koji se analiziraju, pri čemu je N_0 središnji piksel, N_1 vrijednost boje piksela desno od središnjeg piksela i ostali N_k koji predstavljaju boje piksela poredanih u smjeru suprotno od smjera kazaljke na satu oko središnjeg piksela. Na slici 3.3.1.1. su prikazani nekoliko uzoraka čiji su brojevi povezanosti:

$$C_n = N_1 - (N_1 \cdot N_2 \cdot N_3) + N_3 - (N_3 \cdot N_4 \cdot N_5) + N_5 - (N_5 \cdot N_6 \cdot N_7) +$$

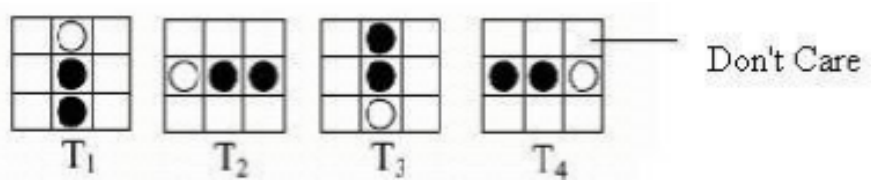
$$N_7 - (N_7 \cdot N_8 \cdot N_1)$$

a) 0 - (0 · 0 · 0) + 0 - (0 · 0 · 0) + 0 - (0 · 0 · 0) + 0 - (0 · 0 · 0) = 0
b) 1 - (1 · 1 · 1) + 1 - (1 · 1 · 1) + 1 - (1 · 1 · 0) + 0 - (0 · 1 · 1) = 1
c) 1 - (1 · 1 · 0) + 0 - (0 · 1 · 1) + 1 - (1 · 1 · 0) + 0 - (0 · 1 · 1) = 2



Slika 3.3.1.1. primjer za brojeve povezanosti

Stentifordov algoritam stanjivanja koristi skup od četiri elementa strukturiranja koje možemo vidjeti na sljedećoj slici.

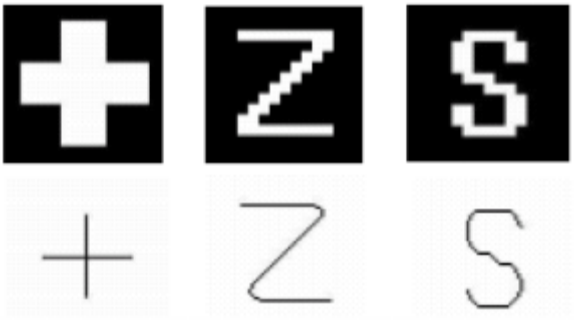


Slika 3.3.1.2. elementi strukturiranja za Stentiford algoritam

Algoritam se može iskazati sljedećim koracima:

1. pronađi skup piksela na slici koji odgovaraju elementu strukturiranja T1
2. ako središnji element (podrijetlo) nije krajnja točka (eng. *endpoint)* i ima broj povezanosti jednak jedan, označi ga za brisanje
   -> krajnja točka je piksel koji je povezan sa samo još jednim drugim pikselom, npr. kraj kostura
3. ponavljaj korak 1 i 2 sve dok se ne prođe cijela slika (slijeva na desno, odozgo prema dolje)
4. ponavljaj korake 1-3 za preostale elemente strukturiranja T2, T3 i T4
5. postavi piksele označene za brisanje u bijelo

Prolaskom s T1 po slici brišemo piksele pri vrhu slike, s T2 brišemo piksele na lijevoj strani slike, s T3 brišemo piksele pri dnu slike i s T4 brišemo piksele na desnoj strani slike. Na taj način nam na kraju preostanu samo pikseli koji tvore kostur objekta.



Slika 3.3.1.3 primjer Stentiford algoritma

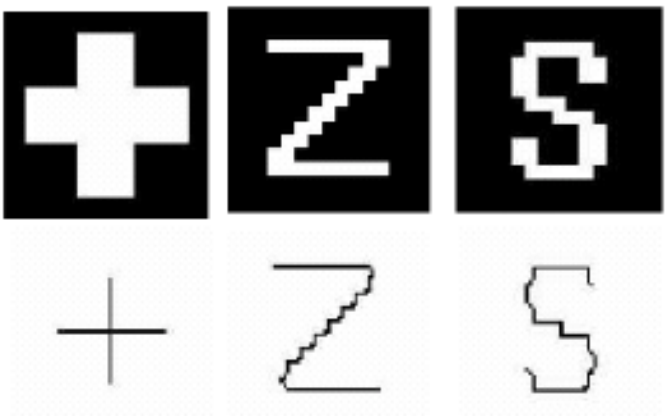
Drugi primjer ove klase, algoritam od Zhang-Suena, je brz i jednostavan za implementirati te se zasniva na paralelnom pristupu, što znači da nova dobivena vrijednost jedino ovisi o vrijednosti iz prijašnje iteracije. Ovaj algoritam ima dvije poditeracije gdje se u prvoj piksel I(i, j) označava za brisanje ako su zadovoljeni sljedeći uvjeti:

1. broj povezanosti piksela I iznosi jedan
2. broj crnih susjeda (onih u prvom planu) je između dva i šest
3. barem jedan piksel I(i, j+1), I(i-1, j) ili I(i, j-1) je bijeli (u pozadini)
4. barem jedan piksel I(i-1, j), I(i+1, j) ili I(i, j-1) je bijeli (u pozadini)

U drugoj poditeraciji pikseli se označavaju za brisanje ako zadovoljavaju sljedeće uvjete:

1. broj povezanosti piksela I iznosi jedan
2. broj crnih susjeda (onih u prvom planu) je između dva i šest
3. barem jedan piksel I(i-1, j), I(i, j+1) ili I(i+1, j) je bijeli (u pozadini)
4. barem jedan piksel I(i, j+1), I(i+1, j) ili I(i, j-1) je bijeli (u pozadini)

Na kraju se brišu pikseli koji su zadovoljili ove uvjete. Algoritam se zaustavlja, ako na kraju bilo koje poditeracije ne bude piksela za brisanje.



Slika 3.3.1.4. primjer Zhang-Shuen algoritma

4. IMPLEMENTACIJA

U ovom poglavlju će se obraditi implementacija algoritma stanjivanja. Pokazat će se glavni dijelovi programa te njihova uloga u programu. Objašnjenje će biti nadopunjeno raznim isječcima koda i slikama rada algoritma.

4.1. UVOD U IMPLEMENTACIJU

Implementirani su više različitih algoritama stanjivanja (eng. *thinning*), pri čemu se razlikuju po elementima za strukturiranje. Kako već objašnjeno, odabirom elemenata za strukturiranje određujemo preciznost i brzinu rada algoritma. Većina implementiranih algoritama daje slične ili potpuno iste rezultate, osim modificiranog algoritma kod kojeg su dodani elementi za strukturiranje po vlastitom odabiru autora završnog rada koji daje malo promijenjene rezultate u odnosu na druge algoritme.

Algoritmi su implementirani u programskom okruženju Visual C++ 2010 Express Edition uz dodatak openCV knjižice. OpenCV je vrlo moćan za projekte bazirani na računalnom vidu i grafici jer sadrži gotove funkcije koje pomažu tome. OpenCV je u ovom slučaju bio jako koristan za dohvat piksela i uspoređivanje određenih piksela s predloškom te za transformaciju učitane slike u pogodan oblik za provođenje algoritma. OpenCV je također riješio problem prikaza slika jer sadrži ugrađene funkcije za prikaz slika prema volji programera.

4.2. PRIKAZ GLAVNIH IMPLEMENTACIJSKIH DIJELOVA

U ovom odjeljku će se prikazati glavni dijelovi koda zaslužni za učitavanje, prikaz i filtriranje slike te za dohvaćanje, manipuliranje i postavljanje određenih piksela slike. Kod je napravljen da bude generički, pri čemu su neki dijelovi nužni i statični, a drugi dijelovi, poput postavljanje elemenata za strukturiranje, dinamički i potiču korisnika da za sebe smisli elemente za strukturiranje koji stvaraju što bolje rezultate.

U početku je učitavanje slike bio problem koji se brzo riješio primjenom openCV knjižice. OpenCV sadrži funkciju koja učitava sliku iz memorije pomoću putanje te odmah vrši transformaciju ako je potrebno. Moguće je sliku učitati u RGB formatu, u formatu sivih razina (eng. *grayscale*) ili sliku ostaviti kakva je tako

da se navede određeni drugi parametar u pozivu funkcije. U ovom slučaju sliku je potrebno transformirati u formatu sivih razina jer je u takvom formatu lakše raditi operacije nad pikselima te nam je u konačnici važan samo oblik slike za ekstrakciju njezinog kostura. To se postiže postavljanjem drugog parametra pozivajuće funkcije u nulu.

```
//ucitavanje i stvaranje slika
IplImage *ulaznaSlika = cvLoadImage(slika, 0);      //grayscale slika
if(!ulaznaSlika){
    printf("Slika se nije mogla učitati!\n");
    scanf("%d", &stani);
    return 1;
}
```

Isječak koda 4.2.1. primjer učitavanja slike u formatu sivih razina

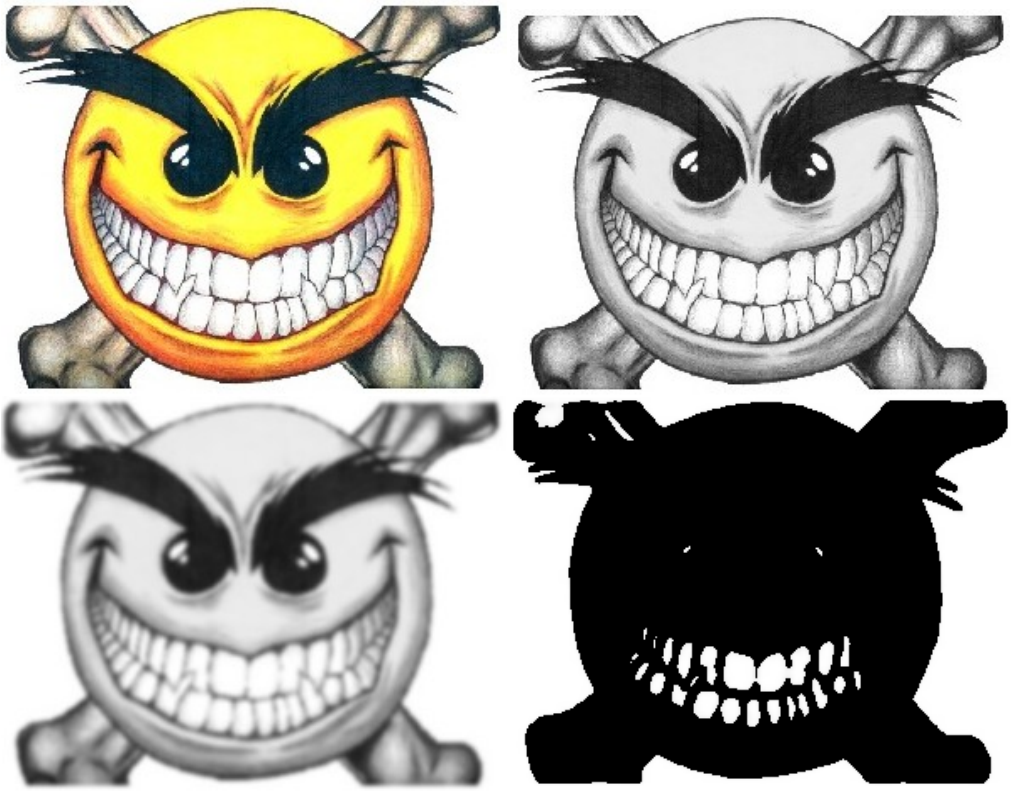
Nakon učitavanja slike i transformacije u format sivih razina, potrebno je sliku pripremiti u oblik koji je pogodan za provođenje algoritma. Kako nam nisu važni detalji slike i njezine boje već samo izgled, možemo od slike napraviti njezinu "sjenu" koja prikazuje samo njezin oblik.

```
//primjena gaussovog filtra
cvSmooth(ulaznaSlika, ulaznaSlika, CV_GAUSSIAN, 7, 7);

//prilagodba slike algoritmu
for (int i=0; i<slikaX; i++){
    for (int j=0; j<slikaY; j++){
        uchar pix = data[i*step + j];
        int piksel = (int)pix;
        if(piksel < 239){
            data[i*step + j] = 0;
        }
    }
}
```

Isječak koda 4.2.2. primjer pretvorbe slike u pogodan oblik

Prvo će se preko slike proći s Gaussovim filtrom koji će ukloniti sa slike nečistoće koje kasnije mogu prouzročiti krive kosture (stvaraju se zrna na slici koja iskrive kostur). Nakon toga se prolazi s drugim filtrom koji će svaki piksel koji nije bijel postaviti u crno tako da se dobije na kraju oblik slike u crnom. Rezultati pojedinih koraka mogu se vidjeti u donjem primjeru.



Slika 4.2.1. primjer transformacije slike u pogodan oblik, izvorna slika gore lijevo, slika u sivim razinama gore desno, slika nakon Gaussovog filtra dolje lijevo, krajna slika nakon drugog filtra dolje desno

Kako je učitavanje slike predstavljao problem na početku, tako je i prikazivanje slike koji se također riješio uključivanjem openCV knjižice u rad. U openCV-u postoje ugrađene funkcije za prikaz slika u prozoru koji imaju svojstva koje programer određuje vrlo lako definiranjem određenih parametara. Konkretno, prvo se stvori prozor s određenim parametrima koji se potom spoji sa slikom za prikaz.

```
cvNamedWindow("Ulaz", CV_WINDOW_AUTOSIZE);
cvShowImage("Ulaz", ulaznaSlika);
```

Isječak koda 4.2.3. primjer prikaza slike u prozoru

Nakon što je slika pripremljena u potreban oblik, počinje se provoditi algoritam nad slikom. Kako je već objašnjeno ranije, algoritam stanjivanja prolazi preko svakog piksela i gleda susjede od trenutnog piksela koji se nalaze oko njega raspoređeni u matrici tri puta tri.

```
uchar pix = data[(i-1)*step + (j-1)];
pikseli[0] = (int)pix;
pix = data[(i-1)*step + (j)];
pikseli[1] = (int)pix;
pix = data[(i-1)*step + (j+1)];
pikseli[2] = (int)pix;
pix = data[(i)*step + (j+1)];
pikseli[3] = (int)pix;
pix = data[(i+1)*step + (j+1)];
pikseli[4] = (int)pix;
pix = data[(i+1)*step + (j)];
pikseli[5] = (int)pix;
pix = data[(i+1)*step + (j-1)];
pikseli[6] = (int)pix;
pix = data[(i)*step + (j-1)];
pikseli[7] = (int)pix;
pix = data[i * step + j];
pikseli[8] = (int)pix;
```

Isječak koda 4.2.4. primjer koda za dohvat susjeda

Dohvaćeni susjedni pikseli se pohranjuju u obično polje kao cijeli brojevi. Kako bi nam kasnije bilo lakše manipulirati dohvaćenim pikselima, oni se skaliraju na 0 ako je piksel bijele boje ili na 1 ako je crne boje. To će nam poslužiti u sljedećem koraku u kojem se gledaju koliko je od dohvaćenih susjeda vrijednosti 1. Također se gleda koliki je broj povezanosti nad tim pikselima.

```
for(int k=0; k<8; k++){                    //broj susjeda
    if(pikseli[k] == 1){
        susjedi += 1;
    }
}

for(int k=0; k<8; k++){                    //broj povezanosti
    int x = pikseli[k] - pikseli[(k+1) % 8];
    if(x == 1)
    {
        prijelazi += 1;
    }
}
```

Isječak koda 4.2.5. kod za dohvaćanje broja crnih susjeda i broja povezanosti

Umjesto da se broj povezanosti određuje prema formuli definiranoj u odlomku za iterativne algoritme stanjivanja, lakši način je određivanje broja prijelaza $1 \rightarrow 0$ ili

$0 \rightarrow 1$ između prvog piksela i zadnjeg piksela tako da se ide u krug počevši u prvom i završivši u zadnjem pikselu. Dobije se isti rezultat za oba načina.

Nakon određivanja broja crnih susjeda i broja povezanosti, završili smo sa statičnim dijelom programa i počinjemo s promjenjivim dijelom koji se sastoji od definiranja elemenata za strukturiranje te provjeru tih elemenata s dohvaćenim pikselima. Umjesto da se svaki definirani element za strukturiranje uspoređuje piksel po piksel s dohvaćenim pikselima, prave se uvjeti koji odgovaraju elementima za strukturiranje i ako se zadovolje uvjeti onda se središnji piksel obilježi za brisanje.

```
//uvjeti
bool uvjet1 = (pikseli[5] == 1) && (pikseli[8] == 1) && (pikseli[1] == 0);
bool uvjet2 = (pikseli[3] == 1) && (pikseli[8] == 1) && (pikseli[7] == 0);
bool uvjet3 = (pikseli[1] == 1) && (pikseli[8] == 1) && (pikseli[5] == 0);
bool uvjet4 = (pikseli[3] == 0) && (pikseli[8] == 1) && (pikseli[7] == 1);
```

```
int provjeriUvjete(int trenutniPix, int susjedi, int prijelazi, bool uvjet1, bool uvjet2, bool uvjet3, bool uvjet4){
    if((susjedi > 1) && (prijelazi == 1) && ((uvjet1 == true) || (uvjet2 == true) || (uvjet3 == true) || (uvjet4 == true)))
    {
        return true;
    }else
        return false;
}
```

Isječak koda 4.2.6. primjer koda za stvaranje uvjeta i funkcije koja provjerava uvjete (Stentiford algoritam)

Uz provjeru elemenata za strukturiranje , dodatno se gleda je li broj crnih susjeda između 2 i 6 čime se osigurava da središnji piksel nije osamljeni element (broj crnih susjeda je 0), vrh linije (broj crnih susjeda je 1), element unutar doline piksela (broj crnih susjeda je 7) ili da piksel nije dio granice slike (broj crnih susjeda je 8). Također se gleda je li broj povezanosti veći od 1 jer tada se središnji piksel nalazi na mostu između dva ili više dijelova.

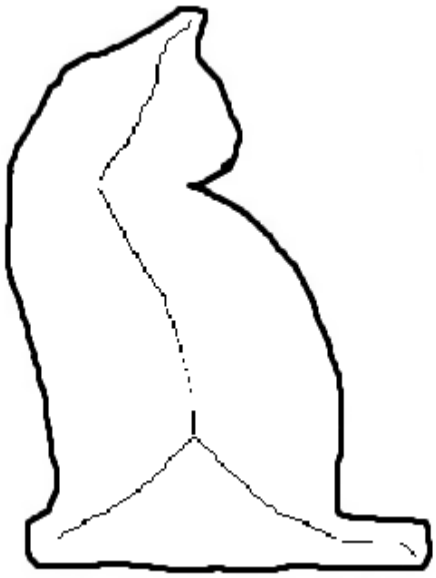
Kada se prođe po svim pikselima i provjere uvjeti nad tim pikselima, pikseli koji su označeni za brisanje se obrišu tako da se njihova vrijednosti postavi u bijelu boju.

4.3. PREDNOSTI I NEDOSTACI IMPLEMENTIRANIH ALGORITAMA

U ovom odjeljku će se analizirati pojedini implementirani algoritmi te će se usporediti prednosti i nedostaci tih algoritama. Implementirana su tri algoritma: Stentifordov, Zhang-Shuenov i modificirani Stentifordov algoritam.

Kako je već opisano ranije, pojedini algoritmi se razlikuju po elementima za strukturiranje. Za Stentifordov i Zhang-Shuenov algoritam opis se može vidjeti u odlomku o iterativnim algoritmima stanjivanja. Modificirani algoritam stanjivanja se razlikuje po tome što su dodana četiri dodatna elementa za strukturiranje kako bi se pokrio sprecifični slučaj kojeg ne pokrivaju ostali algoritmi.

Glavna prednost Stentifordovog algoritma leži u njegovoj brzini koja proizlazi iz broja prolaza po pikselima koji je u ovom slučaju jednak jedan i po broju elemenata za strukturiranje koji se koriste za stanjivanje. Glavni nedostatak je loša preciznost izvučenog kostura koja proizlazi iz manjka elemenata za strukturiranje. Loša preciznost je vidljiva po tome što algoritam nekad obriše i piksele koji pripadaju kosturu pa se s time narušava povezanost kostura.

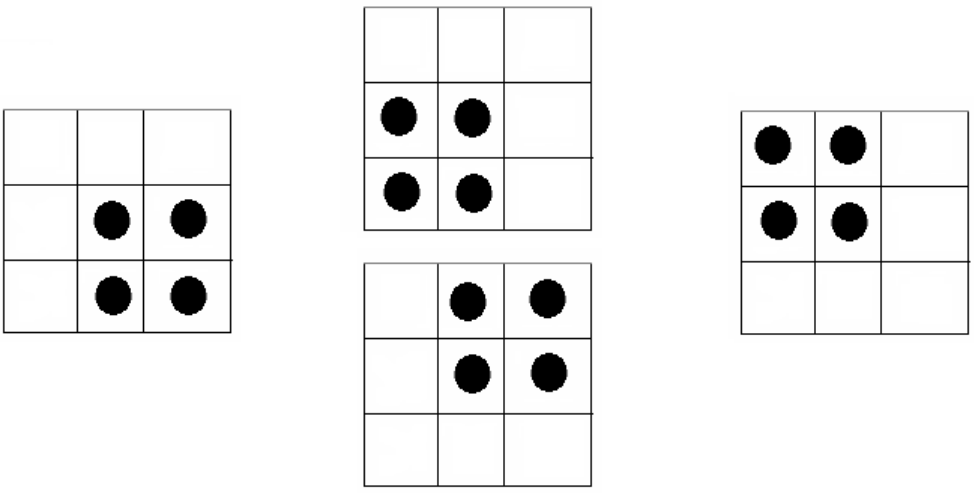


Slika 4.3.1. primjer narušavanja povezanosti kod Stentifordovog algoritma za sliku "maca.jpg"

U Zhang-Shuenovom algoritmu se glavna iteracija iz Stentifordovog algoritma dijeli na dvije poditeracije pri čemu se u prvoj iteraciji brišu pikseli na jugu, istoku i sjevero-zapadu slike, a u drugoj se brišu pikseli na sjeveru, zapadu i jugo-istoku slike. Na taj način postiže se veća preciznost kostura, a i s time se očuva povezanost kostura. Nedostatak je manja brzina obrade slike koja nastaje zbog udvostručenja posla, ali nije gotova vidljiva golom oku zbog razvoja tehnologije u današnje vrijeme.

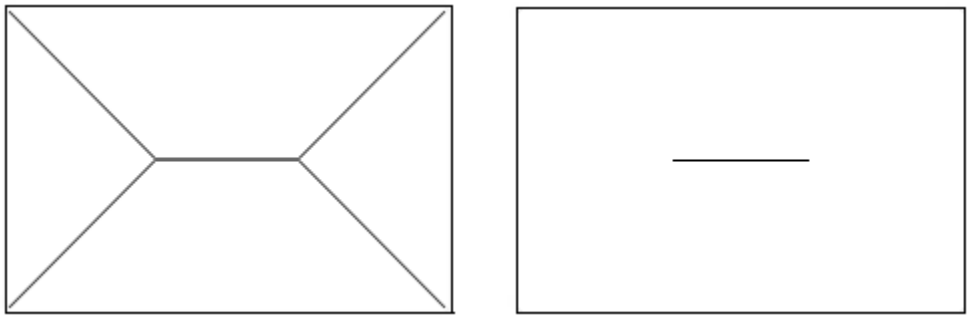
Slika 4.3.2. primjer kostura kod Zhang-Shuenovog algoritma za sliku "maca.jpg"

Modificirani Stentifordov algoritam je demonstracija promjene elemenata za strukturiranje. Zbog nemogućnosti Stentifordovog i Zhang-Shuenovog algoritma da dobiju pravilan kostur za pravokutnik, autor ovog rada je dodao još četiri nova elemenata za strukturiranje kako bi riješio problem.

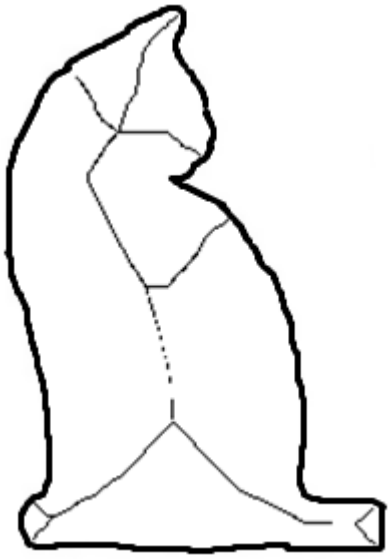


Slika 4.3.4. dodatni elementi za strukturiranje

Dodatni elementi ne brišu piksele ako se nalaze na nekim od vrhova slike. Na ovaj način se dobije pravilan kostur za pravokutnik, ali se dobiju čudni rezultati za malo složenije slike. Zbog toga nije u potpunosti ispravan, ali zato služi kao pokazatelj utjecaja elemenata za strukturiranje na algoritam.



Slika 4.3.5. primjer kostura za sliku "pravokutnik.jpg", modificirani Stentifordov algoritam lijevo, Stentifordov algoritam desno



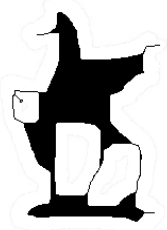
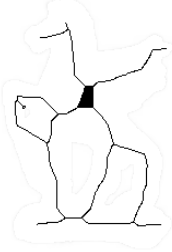
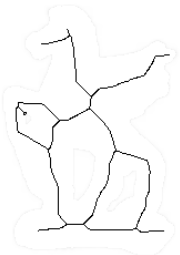
Slika 4.3.6. primjer kostura kod modificiranog Stentifordovog algoritma za sliku "maca.jpg"

4.4. RAD ALGORITMA ZA KONKRETAN PRIMJER

Algoritam stanjivanja je iteracijski algoritam gdje se pri svakoj iteraciji skidaju suvišni pikseli sve dok se ne dođe do kostura. U ovom odjeljku će se prikazati rad algoritma po iteracijama za sliku "konj.jpg". Broj iteracija koji je potreban da se dođe do kostura ovisi o implementaciji algoritma i veličini slike nad kojom se obavlja algoritam.

Slika 4.4.1. ulazna slika "konj.jpg"

Tablica 4.4.1. primjer rada algoritma po iteracijama za sliku "konj.jpg"

Broj iteracija	Slika
5	
10	
30	
35	

4.5. VRIJEME IZVRŠAVANJA IMPLEMENTIRANIH ALGORITAMA

Kako već ranije objašnjeno, Zhang-Shuenov algoritam je, u odnosu na Stentifordov algoritam, sporiji zbog dodatne iteracije kojom se ukupni posao udvostruči.

```
THINNING ALGORITAM ZA SKELETONIZACIJU POLIGONALNOG
Upute: Na pocetku unesite ime slike nad kojom zeli
e daljnu iteraciju algoritma nad modelom stisnite
Ako zelite zavrsiti, zatvorite glavni prozor!
Ucitajte sliku:
majmun.jpg
Algoritam je trajao 0.138 sekundi u 1. iteraciji
Algoritam je trajao 0.062 sekundi u 2. iteraciji
Algoritam je trajao 0.079 sekundi u 3. iteraciji
Algoritam je trajao 0.074 sekundi u 4. iteraciji
Algoritam je trajao 0.074 sekundi u 5. iteraciji
Algoritam je trajao 0.076 sekundi u 6. iteraciji
Algoritam je trajao 0.075 sekundi u 7. iteraciji
Algoritam je trajao 0.071 sekundi u 8. iteraciji
Algoritam je trajao 0.077 sekundi u 9. iteraciji
Algoritam je trajao 0.075 sekundi u 10. iteraciji
Algoritam je trajao 0.076 sekundi u 11. iteraciji
Algoritam je trajao 0.074 sekundi u 12. iteraciji
Algoritam je trajao 0.072 sekundi u 13. iteraciji
Algoritam je trajao 0.075 sekundi u 14. iteraciji
Algoritam je trajao 0.078 sekundi u 15. iteraciji
Algoritam je trajao 0.074 sekundi u 16. iteraciji
Algoritam je trajao 0.073 sekundi u 17. iteraciji
Algoritam je trajao 0.08 sekundi u 18. iteraciji
Algoritam je trajao 0.076 sekundi u 19. iteraciji
```

```
THINNING ALGORITAM ZA SKELETONIZACIJU POLIGONALNOG
Upute: Na pocetku unesite ime slike nad kojom zeli
e daljnu iteraciju algoritma nad modelom stisnite
Ako zelite zavrsiti, zatvorite glavni prozor!
Ucitajte sliku:
majmun.jpg
Algoritam je trajao 0.102 sekundi u 1. iteraciji
Algoritam je trajao 0.047 sekundi u 2. iteraciji
Algoritam je trajao 0.048 sekundi u 3. iteraciji
Algoritam je trajao 0.046 sekundi u 4. iteraciji
Algoritam je trajao 0.046 sekundi u 5. iteraciji
Algoritam je trajao 0.047 sekundi u 6. iteraciji
Algoritam je trajao 0.048 sekundi u 7. iteraciji
Algoritam je trajao 0.043 sekundi u 8. iteraciji
Algoritam je trajao 0.045 sekundi u 9. iteraciji
Algoritam je trajao 0.045 sekundi u 10. iteraciji
Algoritam je trajao 0.049 sekundi u 11. iteraciji
Algoritam je trajao 0.046 sekundi u 12. iteraciji
Algoritam je trajao 0.047 sekundi u 13. iteraciji
Algoritam je trajao 0.043 sekundi u 14. iteraciji
Algoritam je trajao 0.047 sekundi u 15. iteraciji
Algoritam je trajao 0.048 sekundi u 16. iteraciji
Algoritam je trajao 0.045 sekundi u 17. iteraciji
Algoritam je trajao 0.048 sekundi u 18. iteraciji
Algoritam je trajao 0.047 sekundi u 19. iteraciji
```

```
THINNING ALGORITAM ZA SKELETONIZACIJU POLIGONALNOG
Upute: Na pocetku unesite ime slike nad kojom zelit
e daljnu iteraciju algoritma nad modelom stisnite
Ako zelite zavrsiti, zatvorite konzolu!
Ucitajte sliku:
majmun.jpg
Algoritam je trajao 0.104 sekundi u 1. iteraciji
Algoritam je trajao 0.033 sekundi u 2. iteraciji
Algoritam je trajao 0.047 sekundi u 3. iteraciji
Algoritam je trajao 0.047 sekundi u 4. iteraciji
Algoritam je trajao 0.054 sekundi u 5. iteraciji
Algoritam je trajao 0.043 sekundi u 6. iteraciji
Algoritam je trajao 0.045 sekundi u 7. iteraciji
Algoritam je trajao 0.039 sekundi u 8. iteraciji
Algoritam je trajao 0.05 sekundi u 9. iteraciji
Algoritam je trajao 0.043 sekundi u 10. iteraciji
Algoritam je trajao 0.046 sekundi u 11. iteraciji
Algoritam je trajao 0.048 sekundi u 12. iteraciji
Algoritam je trajao 0.047 sekundi u 13. iteraciji
Algoritam je trajao 0.047 sekundi u 14. iteraciji
Algoritam je trajao 0.046 sekundi u 15. iteraciji
Algoritam je trajao 0.048 sekundi u 16. iteraciji
Algoritam je trajao 0.044 sekundi u 17. iteraciji
Algoritam je trajao 0.047 sekundi u 18. iteraciji
Algoritam je trajao 0.05 sekundi u 19. iteraciji
```

Slika 4.5.1. ispis vremena izvršavanja za Zhang-Shuenov (gore desno), Stentifordov (dolje lijevo) i modificirani algoritam (dolje desno) za ulaznu sliku "majmun.jpg" (gore lijevo)

Algoritmi su implementirani na način da se provjere, koje određuju hoće li se neki piksel izbrisati, izvršavaju samo za piksele modela (crne piksele). Na taj način se vrijeme izvršavanja dodatno skraćuje. Iz ovog indirektno slijedi da se vrijeme jedne iteracije, za svaki implementirani algoritam, povećava veličinom slike tj. brojem crnih piksela. Ako se usporede vremena izvršavanja iz primjera iz slike 4.5.1. i primjera iz slike 4.5.2., može se lako uočiti dokaz za ovu tvrdnju.

```
THINNING ALGORITAM ZA SKELETONIZACIJU POLIGONALNOG
Upute: Na pocetku unesite ime slike nad kojom zeli
e daljnu iteraciju algoritma nad modelom stisnite
Ako zelite zavrsiti, zatvorite glavni prozor!
Ucitajte sliku:
covjek2.jpg
Algoritam je trajao 0.079 sekundi u 1. iteraciji
Algoritam je trajao 0.047 sekundi u 2. iteraciji
Algoritam je trajao 0.046 sekundi u 3. iteraciji
Algoritam je trajao 0.047 sekundi u 4. iteraciji
Algoritam je trajao 0.047 sekundi u 5. iteraciji
Algoritam je trajao 0.047 sekundi u 6. iteraciji
Algoritam je trajao 0.063 sekundi u 7. iteraciji
Algoritam je trajao 0.047 sekundi u 8. iteraciji
Algoritam je trajao 0.047 sekundi u 9. iteraciji
Algoritam je trajao 0.047 sekundi u 10. iteraciji
Algoritam je trajao 0.047 sekundi u 11. iteraciji
Algoritam je trajao 0.047 sekundi u 12. iteraciji
Algoritam je trajao 0.047 sekundi u 13. iteraciji
Algoritam je trajao 0.047 sekundi u 14. iteraciji
Algoritam je trajao 0.046 sekundi u 15. iteraciji
Algoritam je trajao 0.062 sekundi u 16. iteraciji
Algoritam je trajao 0.047 sekundi u 17. iteraciji
Algoritam je trajao 0.047 sekundi u 18. iteraciji
Algoritam je trajao 0.046 sekundi u 19. iteraciji
```

```
THINNING ALGORITAM ZA SKELETONIZACIJU POLIGONALNOG
Upute: Na pocetku unesite ime slike nad kojom zeli
e daljnu iteraciju algoritma nad modelom stisnite
Ako zelite zavrsiti, zatvorite glavni prozor!
Ucitajte sliku:
covjek2.jpg
Algoritam je trajao 0.078 sekundi u 1. iteraciji
Algoritam je trajao 0.047 sekundi u 2. iteraciji
Algoritam je trajao 0.032 sekundi u 3. iteraciji
Algoritam je trajao 0.031 sekundi u 4. iteraciji
Algoritam je trajao 0.031 sekundi u 5. iteraciji
Algoritam je trajao 0.031 sekundi u 6. iteraciji
Algoritam je trajao 0.032 sekundi u 7. iteraciji
Algoritam je trajao 0.031 sekundi u 8. iteraciji
Algoritam je trajao 0.031 sekundi u 9. iteraciji
Algoritam je trajao 0.047 sekundi u 10. iteraciji
Algoritam je trajao 0.031 sekundi u 11. iteraciji
Algoritam je trajao 0.031 sekundi u 12. iteraciji
Algoritam je trajao 0.031 sekundi u 13. iteraciji
Algoritam je trajao 0.032 sekundi u 14. iteraciji
Algoritam je trajao 0.047 sekundi u 15. iteraciji
Algoritam je trajao 0.031 sekundi u 16. iteraciji
Algoritam je trajao 0.032 sekundi u 17. iteraciji
Algoritam je trajao 0.031 sekundi u 18. iteraciji
Algoritam je trajao 0.046 sekundi u 19. iteraciji
```

```
THINNING ALGORITAM ZA SKELETONIZACIJU POLIGONALNOG
Upute: Na pocetku unesite ime slike nad kojom zeli
e daljnu iteraciju algoritma nad modelom stisnite
Ako zelite zavrsiti, zatvorite konzolu!
Ucitajte sliku:
covjek2.jpg
Algoritam je trajao 0.063 sekundi u 1. iteraciji
Algoritam je trajao 0.031 sekundi u 2. iteraciji
Algoritam je trajao 0.047 sekundi u 3. iteraciji
Algoritam je trajao 0.032 sekundi u 4. iteraciji
Algoritam je trajao 0.031 sekundi u 5. iteraciji
Algoritam je trajao 0.031 sekundi u 6. iteraciji
Algoritam je trajao 0.046 sekundi u 7. iteraciji
Algoritam je trajao 0.031 sekundi u 8. iteraciji
Algoritam je trajao 0.032 sekundi u 9. iteraciji
Algoritam je trajao 0.047 sekundi u 10. iteraciji
Algoritam je trajao 0.047 sekundi u 11. iteraciji
Algoritam je trajao 0.031 sekundi u 12. iteraciji
Algoritam je trajao 0.031 sekundi u 13. iteraciji
Algoritam je trajao 0.031 sekundi u 14. iteraciji
Algoritam je trajao 0.047 sekundi u 15. iteraciji
Algoritam je trajao 0.031 sekundi u 16. iteraciji
Algoritam je trajao 0.047 sekundi u 17. iteraciji
Algoritam je trajao 0.047 sekundi u 18. iteraciji
Algoritam je trajao 0.031 sekundi u 19. iteraciji
```

Slika 4.5.2. ispis vremena izvršavanja za Zhang-Shuenov (gore desno), Stentifordov (dolje lijevo) i modificirani algoritam (dolje desno) za ulaznu sliku "covjek2.jpg" (gore lijevo)

Mjerenja su izvršena na računalu koji sadrži Intel Pentium Dual Core procesor (svaka jezgra takta frekvencije 2 GHz-a) i veličine radne memorije od 4 GB-a. Važno je znati konfiguraciju računala na kojem se vrši algoritam jer se algoritam na starijim računalima izvršava sporije u odnosu na novija računala.

5. ZAKLJUČAK

Pisanje ovog rada je bilo vrlo zanimljivo te sam mogao izvući nekoliko zanimljivih zaključaka. Najvažniji zaključak je da su skeletonizacijski algoritmi u svom usponu i da će biti od iznimne važnosti u budućnosti. To nam govori područje primjene koje se za sada uglavnom svodi na medicinu gdje se upotrebljava za procjenu laringotrahijalne stenoze, procjenu infrarenijalne aneurizme aorte i za dobivanje bolje slike debelog crijeva.

Bilo je zanimljivo promatrati različite rezultate koji nastaju uslijed promjene elemenata za strukturiranje. Pritom je teško naći optimalnu kombinaciju.

U ovom radu smo se uglavnom usredotočili na 2D model bez prelaska na 3D. Iako su ovi algoritmi uglavnom namijenjeni radu s pikselima, moguće ih je uz primjenu više matematike prilagoditi radu s vokselima. To nas dovodi do zaključka da je princip skeletonizacije isti (posebno kod algoritma stanjivanja), samo je različita implementacija.

U ovom radu se koristila openCV knjižica koja je vrlo intuitivna za koristiti te je puno pomogla u izradi ovog rada.

POTPIS: _____

(ROBERT MRKONJIĆ)

LITERATURA

[1] Palagyi K., *A 3-subiteration 3D thinning algorithm for extracting medial surfaces*, University of Szeged, 2002.

[2] Chang S., *Extracting Skeletons from Distance Maps*, Penn State University, 2007.

[3] Skeletonization using distance transform, http://www.cvmt.dk/education/teaching/f10/MED8/CV/Stud/838.pdf, 6.3.2011.

[4] Teleu A., *A Robust Level-Set Algorithm for Centerline Extraction*, University of Tehnology Den Dolech, Eindhoven, 2003.

[5] Skeletonization, http://www.inf.u-szeged.hu/~palagyi/skel/skel.html, 6.3.2011.

[6] Skeletonization/Medial Axis Transform, http://homepages.inf.ed.ac.uk/rbf/HIPR2/skeleton.htm, 6.3.2011.

[7] Fabbri R., *On Voronoi Diagrams and Medial Axes*, University of Sao Paulo, Brazil

[8] Amato N.M., *On computing Voronoi diagrams by divide-prune-and-conquer,* Texas A&M University

[9] Voronoi Diagram, http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/CG-Applets/VoroDiagram/vorocli.htm, 20.3.2011.

[10] Thinning, http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm, 10.4.2011.

[11] Structuring Elements, http://homepages.inf.ed.ac.uk/rbf/HIPR2/strctel.htm, 10.4.2011.

[12] Hit-and-Miss Transform, http://homepages.inf.ed.ac.uk/rbf/HIPR2/hitmiss.htm, 10.4.2011.

[13] Thinning Algorithm, http://www.docstoc.com/docs/61481377/Thinning-Algorithm, 10.4.2011.

SAŽETAK

POSTUPCI ODREĐIVANJA KOSTURA MODELA NA OSNOVI POLIGONALNOG MODELA

U ovom radu čitatelju se želi dati uvid u osnovne postupke za izvlačenje kostura iz pologonalnih modela.

Kostur u grafici je najlakše objasniti pomoći H. Blumove analogije paljenja livade u kojoj, nakon paljenja livade, mjesta sjecišta plamena čine kostur. Postupci za izvlačenje kostura se mogu klasificirati u tri klase: postupci bazirani na udaljenosti, postupci bazirani na poligonima i topološko stanjivanje. U postupcima baziranim na udaljenostima rade se mape udaljenosti pomoću kojih se izvlače kosturi s posebnim algoritmima, dok se to u postupcima baziranim na poligonima radi pomoću Voronoi dijagrama. Najrasprostranjeniji su postupci stanjivanja kod kojih se korak po korak brišu pikseli oko modela dok na kraju ne ostane kostur.

Četvrto poglavlje opisuje implementaciju Stentifordovog i Zhang-Sheunovog algoritma stanjivanja. Nakon implementacije, opisani su prednosti i nedostaci pojedinih algoritama te rad algoritma za konkretan primjer.

Algoritam je implementiran u Visual C++-u uz dodatak openCV knjižice.

Ključne riječi: kostur, postupci bazirani na udaljenosti, postupci bazirani na poligonima, topološko stanjivanje, Stentifordov algoritam, Zhang-Shuenov algoritam, openCV

SUMMARY

PROCEDURES FOR DETERMINING SKELETON OF MODEL BASED ON POLYGON BASED MODEL

In this paper, to the reader is given insight into the basic procedures of extracting skeletons from polygonal models

Skeleton in the graphic can be easily explained using H. Blum analogy burning fields in which, after burning the meadow, the intersection of the flame makes the skeleton. Methods for extracting skeletons can be classified into three classes: distance-based methods, polygon-based methods and topological thinning. Distance-based procedure use distance maps for extracting skeletons with special procedures, while in the polygon-based procedure Voronoi diagrams are used instead. The most popular procedure is thinning in which pixels are deleted step-by-step around the model until only the skeleton remains.

The fourth chapter describes the implementation of Stentiford and Zhang-Shuen thinning algorithm. After implementation, the advantages and disadvantages of different algorithms and the work of the algorithm for an concrete example are described.

The algorithm is implemented in Visual C++ with the addition of OpenCV library.

Keywords: skeleton, distance-based methods, polygon-based methods, topological thinning, Stentiford algorithm, Zhang-Shuen algorithm, openCV