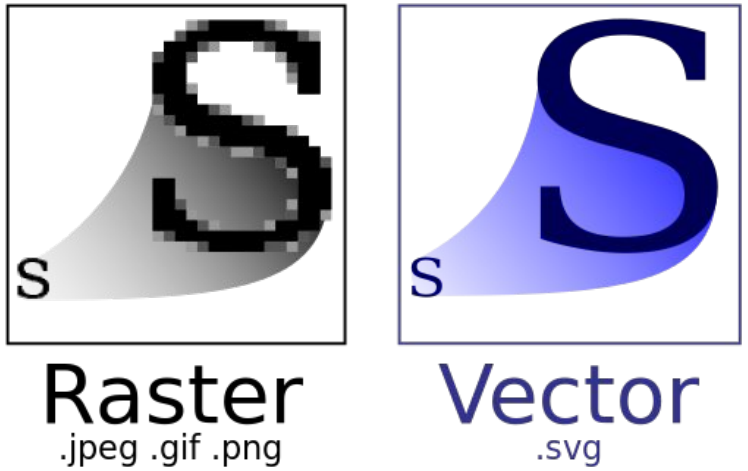
# Pretvaranje rasterskih slika u vektorske

**Projekat iz Soft Computing-a**

Vladimir Makarić

# Motivacija za vektorsku grafiku




Vektorska grafika je pogodna kao medij za prenos vizuelnih podataka iz više razloga:

- Malo zauzeće memorije kada je u pitanju grafički sadržaj poput dijagrama, tehničkih crteža, grafikona, GUI elemenata,...
- Nezavisnost kvaliteta prikaza od rezolucije uređaja. Posledica čuvanja grafičkog sadržaja pomoću matematičkih primitiva je primena invertibilnih transformacija nad slikom bez gubitka informacija ili degradacije sadržaja (rotacija, skaliranje, smicanje,..).

Ovi razlozi imaju još veću težinu u skorije vreme zbog velikog diverziteta uređaja (pogotovo mobilnih) za konzumaciju vizuelnih medija.

# Motivacija za pretvaranje rasterske u vektorsku

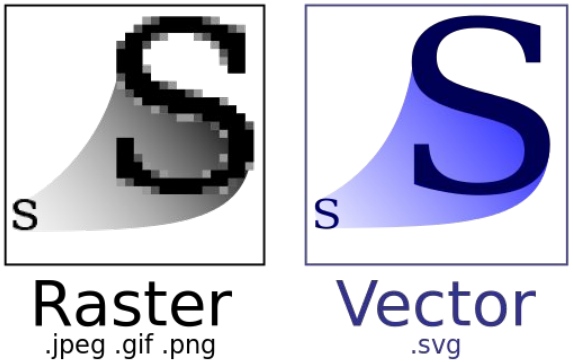
Postoji ogromna količina kvalitetnog grafičkog sadržaja koji je sačuvan u rasterskom obliku, čiji originalni oblik nije dostupan:

- Ilustracije u knjigama
- Tehnički crteži
- Dijagrami
- Karte
- …

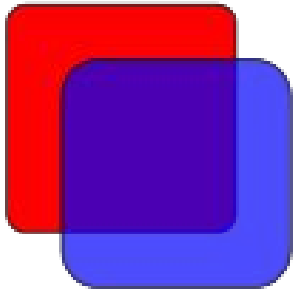
Korisno bi bilo kada bi se geometrijski oblici sa tih rasterskih slika mogli prepoznati, jer bi tada te slike mogle da se transformisu bez gubitaka i time bi se omogućilo njihovo štampanje ili prikaz na medijima visoke rezolucije.

# SVG - Scalable Vector Graphics

- Format za čuvanje vektorske grafike razvijen od stane W3C
- Baziran na XML-u
- Podržavaju ga moderni Web pretraživači (Chrome, Firefox, Opera..)




Primer:

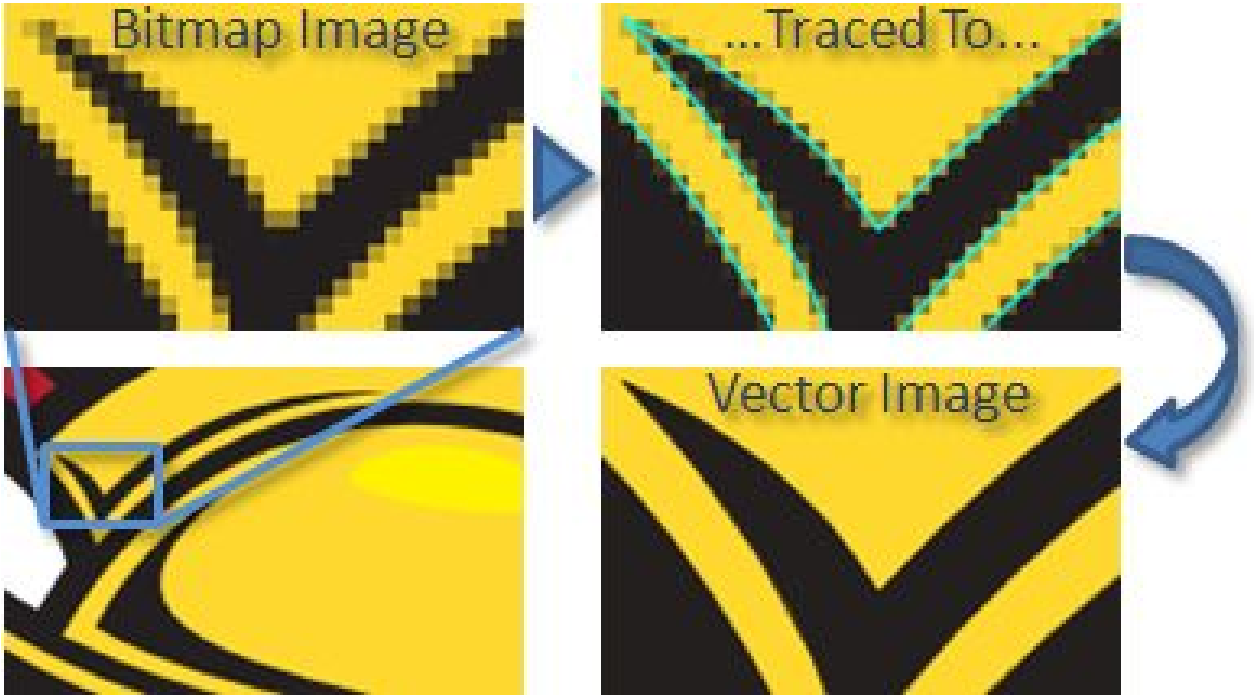


```
<svg xmlns="http://www.w3.org/2000/svg"
 width="467" height="462">
  <rect x="80" y="60" width="250" height="250" rx="20"
     style="fill:#ff0000; stroke:#000000;stroke-width:2px;" />

  <rect x="140" y="120" width="250" height="250" rx="40"
     style="fill:#0000ff; stroke:#000000; stroke-width:2px;
     fill-opacity:0.7;" />
</svg>
```

# Vektorizacija

Postupak pretvaranja rasterskih slika u vektorske se zove Image Tracing ili vektorizacija.



# Postojeća rešenja

Popularan algoritam za vektorizaciju je Potrace (http://potrace.sourceforge.net/potrace.pdf).

Zadatak svakog algoritma za vektorizaciju je da napravi dobar trade-off u prepoznavanju ivica i glatkih krivih (uglavnom se koriste Bezierove krive). Ako se prepozna previše ivica, izlazni oblik će izgledati kao poligon i neće biti gladak. Sa druge strane ako se sve aproksimira sa glatkim krivama, izlazni oblik će izgledati previše zaobljeno:

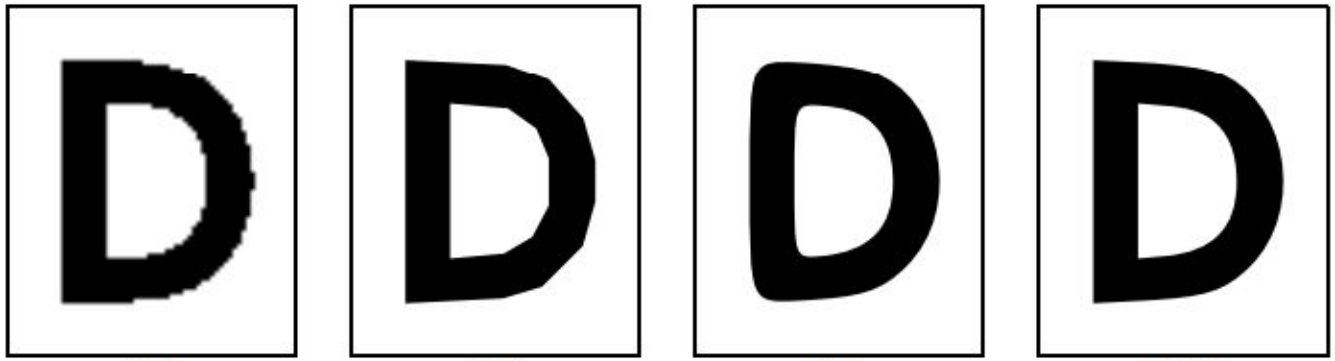


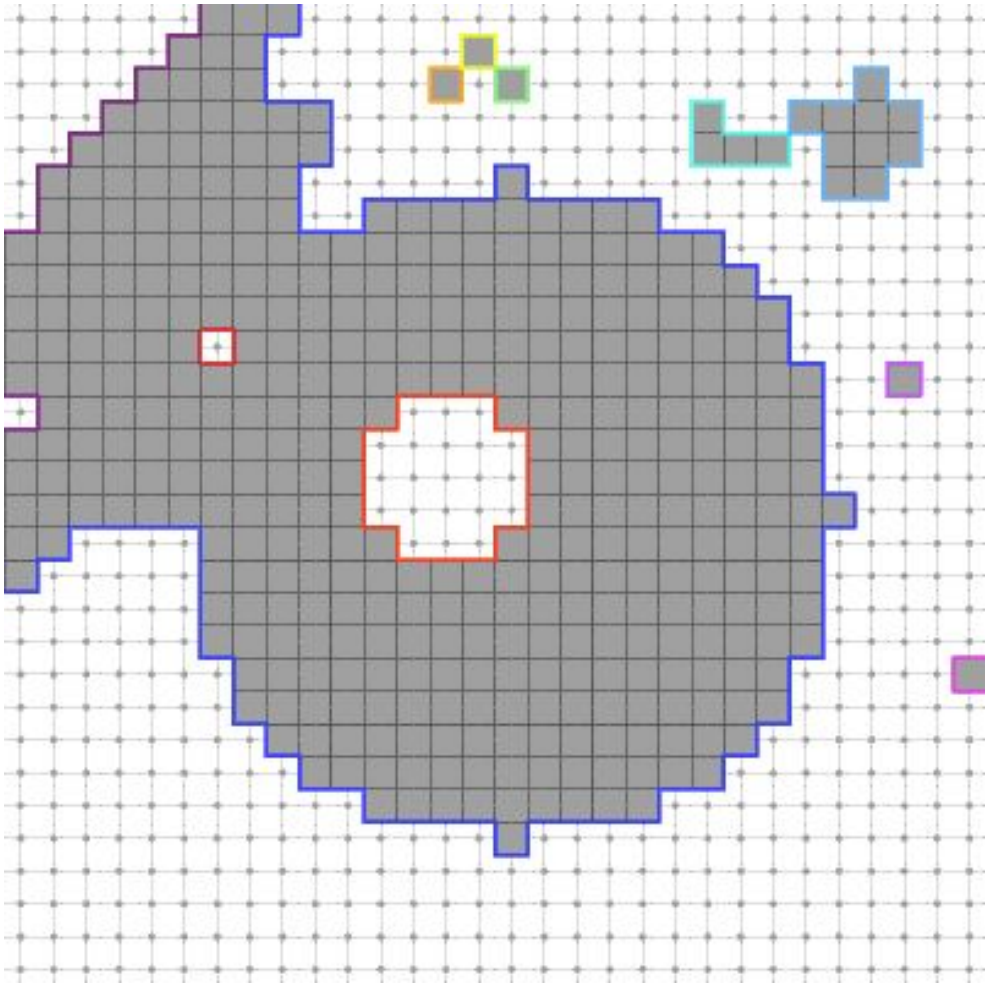

Figure 1: Corner detection. (a) the original bitmap; (b) too many corners; (c) too few corners; (d) good corner detection.

# Sažetak projekta po delovima

1. Ekstrakcija konture sa binarne slike (crno beli pikseli)
2. Računanje grafa povezanosti tačaka konture za aproksimaciju konture poligonom
3. Računanje optimanog poligona nad tačkama konture
4. Modifikacija optimalnog poligona
5. Nalaženje ćoškova poligona
6. Fitovanje Bezierovih krivih između ivica
7. Korekcija fitovanih krivih

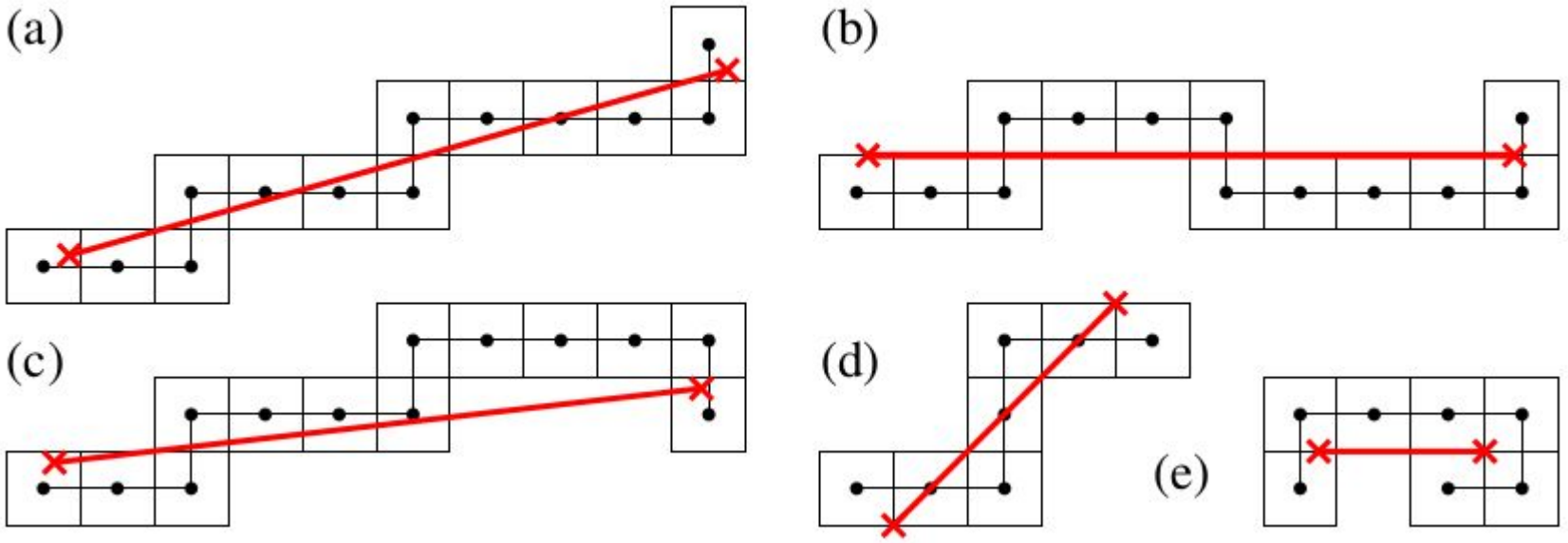
# Ekstrakcija konture sa binarne slike

Kontura je graf nad matricom koja cini sliku, gde su povezani susedni pikseli ivice objekta, primeri kontura:



# Računanje grafa povezanosti konture

Na poliliniji $p_1$, $p_2$,.., $p_n$, tacke $a=p_1$ i $b=p_n$ su povezane ako postoji duž koja počinje u kvadratnoj okolini polovine piksela a i završava se u kvadratnoj okolini polovine piksela tačke b, stim da mora da prolazi kroz kvadratne okoline polovine piksela svih tačaka između. Dodatan uslov je da polilinija ne sadrži sva četiri smera veza između $p_i$ i $p_{i+1}$ , primer kršenja ovog uslova je slika pod (e):



# Računanje grafa povezanosti konture

Ekvivalentan uslov koji je lakše proveriti je da za svaku trojku čvorova poligona važi da kroz dve eksterne tačke može da se povuče duž tako da ona dodiruje kvadratnu razdaljinu veličine 1 od srednje tačke.

# Računanje optimalnog poligona

Svakoj od potencijalnih ivica se dodeli težina koja je jednaka dužini ivice puta standardnoj devijaciji tačaka izmedju od ivice:

$$P_{i,j} = |v_j - v_i| \cdot \sqrt{\frac{1}{j \ominus i + 1} \sum_{k=i}^{j} dist(v_k, \overline{v_i v_j})^2},$$

Ovakva formula je izabrana jer je moguće "keširati" parcijalne sume kvadrata x i y koordinata vektora i njihovog proizvoda i u konstantnom vremenu izračunati težinu za proizvoljne dve tačke poligona.

$$P_{i,j} = \sqrt{cx^2 + 2bxy + ay^2},$$

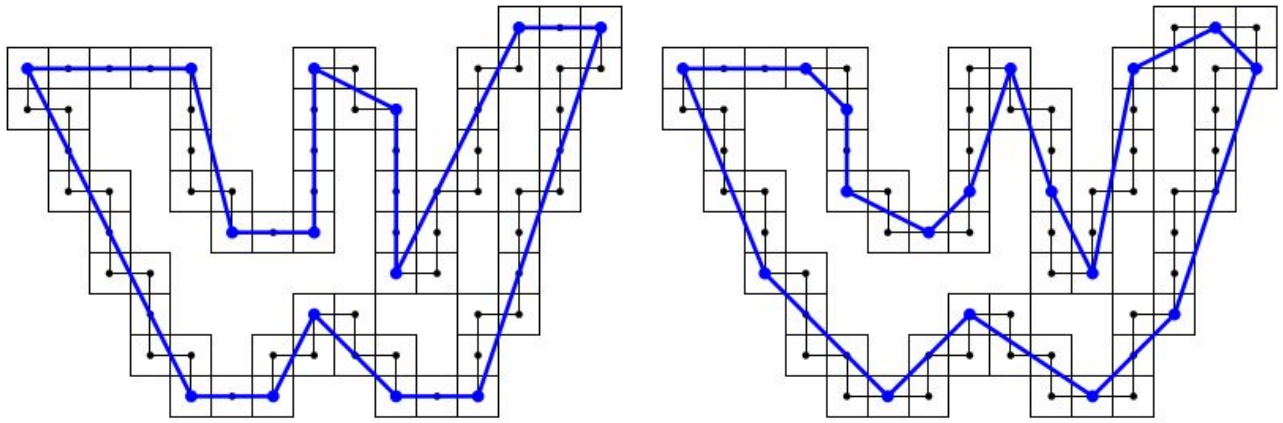
$$E(x_k^2) = \frac{1}{j \ominus i+1} \sum_{k=i}^{j} x_k^2$$

$$\begin{aligned} a &= E(x_k^2) - 2\bar{x}E(x_k) + \bar{x}^2, \\ b &= E(x_k y_k) - \bar{x}E(x_k) - \bar{y}E(y_k) + \overline{xy}, \\ c &= E(y_k^2) - 2\bar{y}E(y_k) + \bar{y}^2. \end{aligned}$$

# Računanje optimalnog poligona

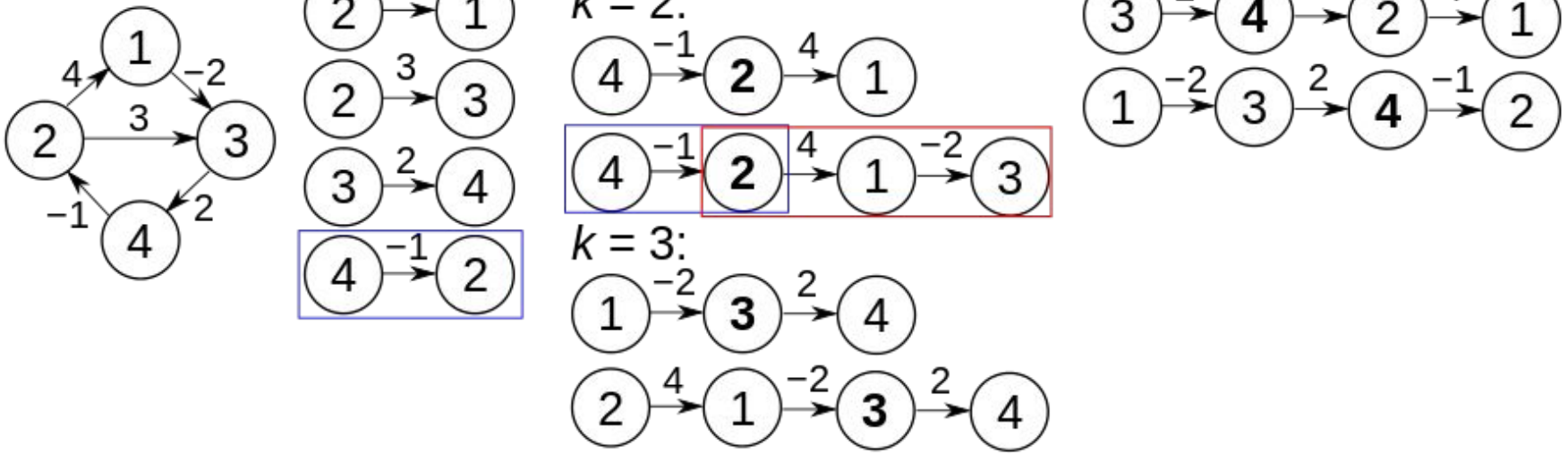
Kada se formira graf sa težinama, onda se nad tim grafom primeni Floyd-Warshal algoritam za nalaženje najkraćih putanja između svih parova ivica (all pair shortest path problem) sa modifikacijom inicijalnih uslova tako da pronađe najkraće cikluse, sa leksikografskom metrikom (l, w), gde je l dužina putanje a w težina ivice. Rezultat je da se nađe najkraći ciklus po broju ivica čiji zbir težina je najmanji.

Primer rezultata, optimalan poligon je levi:



# Foyd-Warshall

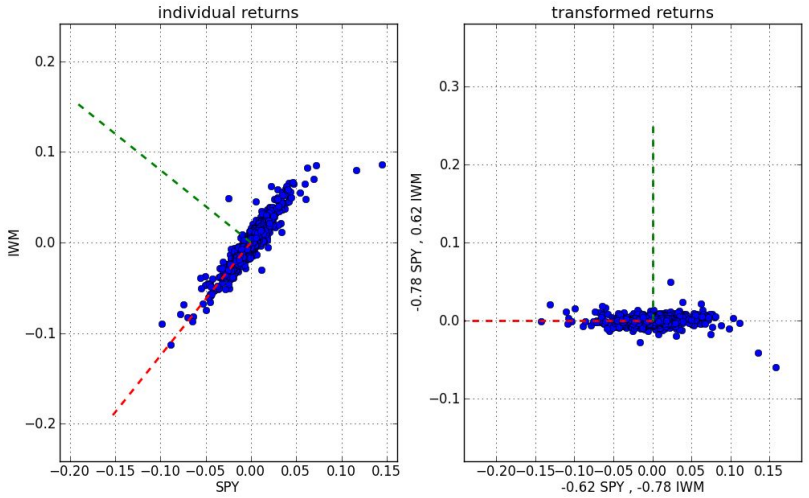
Algoritam na bazi dinamičkog programiranja. Potproblemi su naći najkraće putanje čiji među čvorovi sadrže čvorove od 1..k. Algoritam se sastoji od k iteracija, u k-toj iteraciji se pokuša poboljšati trenutna dužina putanje od i do j tako što se pokušava proći kroz čvor k, spajajući najkraće putanje od i do k i k do j (za koje se zna da su najbolje od svih koje uzimaju međučvorove iz skupa 1..k-1).



# Modifikacija optimalnog poligona

Poligon je optimalan među onim čija temena su u tačkama konture, u ivicama piksela. Taj uslov se olabavljuje tako što se dozvoljava da se temena nađu u kvadratnoj okolinu od pola piksela oko svojih trenutnih pozivija.

Za svake dve povezane tačke na poligonu se nalazi optimalna prava koja minimizuje varijansu do tačaka na originalnoj polilinji. Iskorišćena je PCA (principal component analysis) metoda, koja se bazira na dijagonalizaciji kovarijansne matrice x i y koordinata tačaka:



# Modifikacija optimalnog poligona

Nove tačke poligona se nalaze tako što se stare tačke pomeraju u svojoj 0.5 piksel kvadratnoj okolini tako da se minimizuje kvadrat odstojanja od susednih pravih izračunatih pomoću PCA (kvadratna optimizacija sa ograničenjima).

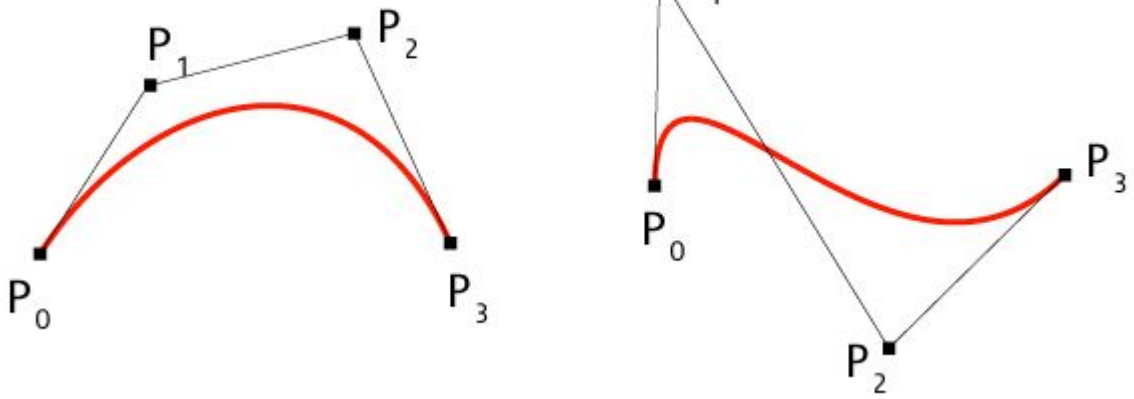
# Nalaženje ćoškova poligona

Teme poligona b je ćošak ako je vrednost skalarnog proizvoda normalizovanih vektora do njegovog prethodnika i sledbenika pozitivna ili, u slučaju da je negativna, da je veća od neke negativne konstante, npr -0.2. Drugim rečima da je manji ugao između ta dva vektora manji od 180 - arccos(0.2) ~= 100 stepeni.

Sledeći korak je fitovanje Bezierovih krivih između ćoškova.

# Fitovanje Bezierovih krivih

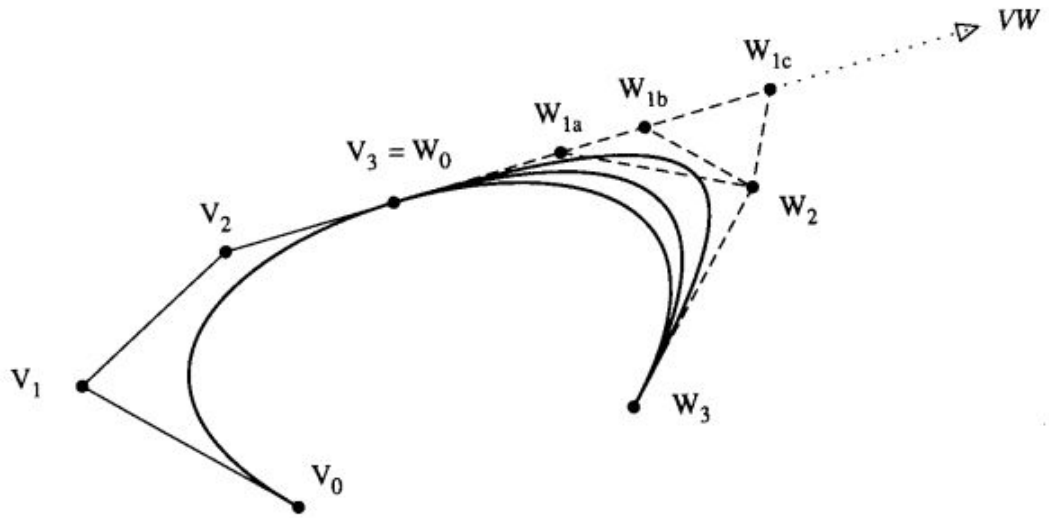
Za rad su korišćene Bezierove krive trećeg stepena, koje su standardne za opis krivih oblika u softveru za vektorsku ilustraciju, kao i u formatima za vektorsku grafiku.




$$\mathbf{B}(t) = (1-t)^3\mathbf{P_0} + 3(1-t)^2 t\mathbf{P_1} + 3(1-t)t^2\mathbf{P_2} + t^3\mathbf{P_3} \ , 0 \leq t \leq 1.$$

# Fitovanje Bezierovih krivih

Glatki delovi poligona (između ćoškova) se aproksimiraju sa nizom povezanih bezierovih krivih sa ciljem da se minimizuje rastojanje od krivih do tačaka poligona i da u tački gde se povezuju dve krive ne postoji geometrijski diskontinuitet, to jest da se prvi izvodi po parametru t poklapaju po pravcu, a intenzitet može biti različiti.



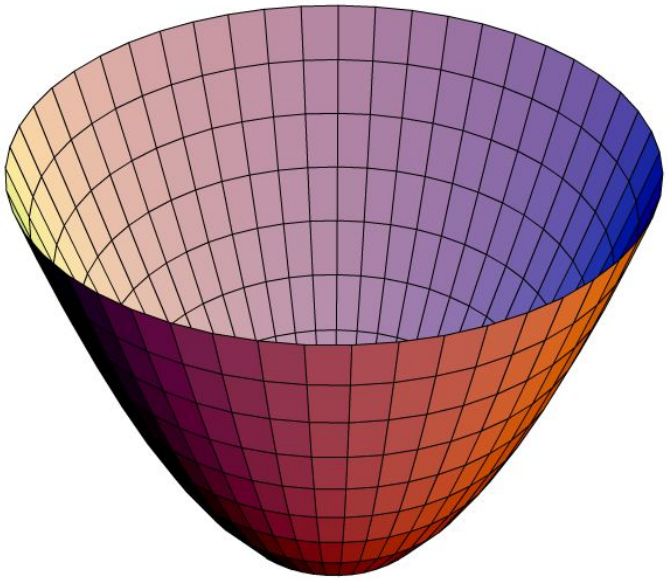

# Fitovanje Bezierovih krivih

Fitovanje Bezierove krive kroz poliliniju je optimizacioni problem, treba minimizovati kvadratno rastojanje tačaka polilinije od krive.

- Problem je što je je Bezierova kriva definisana u X-Y-t prostoru gde su X i Y funkcije od t, a tačke polilinije su X,Y parovi bez asociranog t-a. Potrebno je parametrizovati tačke polilinije da bi mogle da se porede sa Q(t), gde je Q vektorska funkcija realno promenjive koja predstavlja Bezierovu površ. Koristi se chord length parametrization, gde se svakoj tačci daje parametar dužine od početne do nje kroz ukupna dužina polilinije.
- Ulaz u algoritam tačke koje čine poliliniju, prva i poslednja će činiti prvu i poslednju tačku krive. Potrebno je naći dve kontrolne tačke. Da bi olakšali problem i omogućuli fitovanje niza krivih, kao ulaz algoritma se prosleđuju tangente krive u prvoj i poslednjoj tačci polilinije, kod povezanih krivih tangente moraju biti u istom pravcu, suprotnom smeru.

# Fitovanje Bezierovih krivih

Zbog toga što su date tangente, sada je problem sveden na skaliranje dve kontrolne tačke po tangentama. Rešenje je potpuno određeno sa dva realna broja, pošto je funkcija razdaljine uvek pozitivna ona predstavlja paraboloid sa minimumom, čiji minimum je u nuli parcijalnih izvoda po obe promenjive:



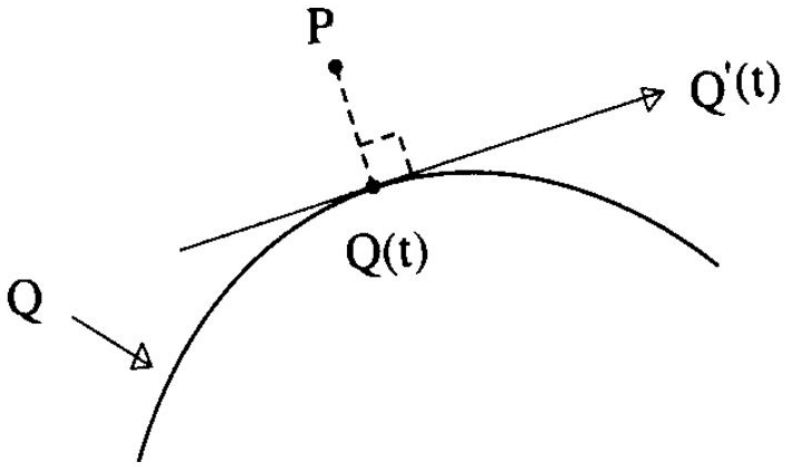
$$\frac{\partial S}{\partial \alpha_1} = 0 \qquad S = \sum_{i=1}^{n} [d_i - Q(u_i)]^2$$

$$\frac{\partial S}{\partial \alpha_2} = 0. \qquad = \sum_{i=1}^{n} [d_i - Q(u_i)] \cdot [d_i - Q(u_i)],$$

*$d_i$ je pozicija tačke i na poliniji čiji aproksimirani parametar je $u_i$

# Korekcija fitovane krive

Moguće je da fit nije smanjio kvadrat greške na zadovoljavajući nivo, često je potrebno reparametrizovati tačke polilinije i pokušati ponovo. Ponovni pokušaj zavisi od praga kvadratnog odstupanja krive od tačaka polinije. Ukoliko je greška veća od praga onda se reparametrizacija radi traženjem pravog parametra na trenutno nađenoj krivoj, numeričkim (Njutnov metod) rešavanjem jednačine:




$$[Q(t) - P]\, Q'(t) = 0$$

Ako je greška prevelika (mnogo veća od praga), onda je potrebno podeliti poliniju na dva dela i rekurzivno ih fitovati (sa tangentama u istom pravcu).

# Reference

Za prvi deo rada (Nalaženje optimalnog poligona) su korišćeni delovi rada:

Potrace: a polygon-based tracing algorithm, Peter Selinger 2003, (http://potrace.sourceforge.net/potrace.pdf)

Za drugi deo (Fitovanje Bezierovih krivih) korišćen je metod fitovanja iz knjige

Graphics Gems 1, Andrew S. Glassner 1993.