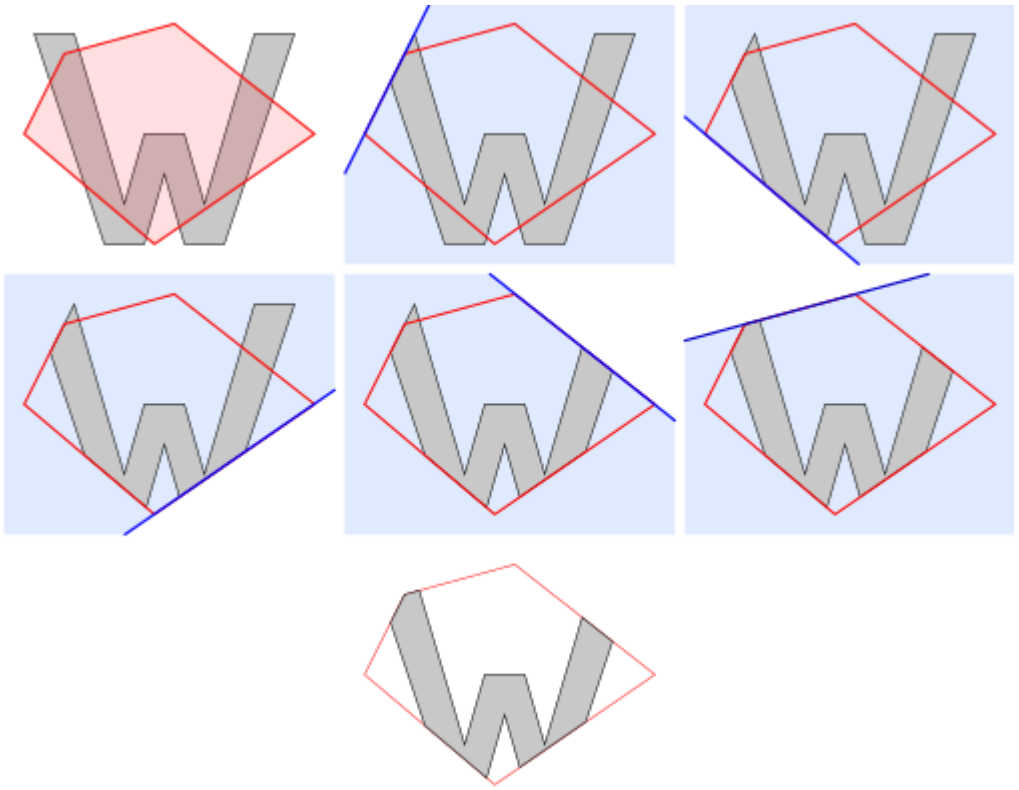
Geometrijski algoritmi @ MATF

Saterlend-Hodžmanov algoritam

Strahinja Milojević



Opis problema

Saterlend-Hodžmanov algoritam se koristi za klipovanje, odnosno "seckanje" poligona. Kao ulaz se zadaju dva poligona – jedan predstavlja glavni poligon, odnosno poligon koji sečemo, dok je drugi klip poligon, odnosno poligon po kom sečemo. Preduslov algoritma je da je klip poligon konveksan. Kao rezultat se dobija onaj deo glavnog poligona koji se nalazi i u klip poligonu. Možemo zamisliti klip poligon kao nekakav okvir, odnosno vidno polje koje se pomera, i u datom trenutku izračunavamo koji deo glavnog poligona će biti vidljjv kroz taj okvir. Algoritam ima primene u kompjuterskoj grafici i simulaciji fizike. Koristi se u algoritmima za renderovanje, kako bi znali koji deo "sveta", odnosno nekog 2D/3D modela će biti vidljiv na ekranu, što drastično smanjuje količinu podataka koje treba obraditi. Što se tiče fizičkih simulacija, primenjuje se kod detekcija kolizije.

Ulaz: *DCEL struktura sa 2 poligona, glavnim i klip poligonom.*

Izlaz: *Poligon koji predstavlja deo glavnog poligona koji se nalazi i u klip poligonu.*

Naivno rešenje problema

Naivno rešenje ovog problema je relativno jednostavno – uporedimo svaku stranicu glavnog poligona sa svakom stranicom klip poligona i pronađemo sve njihove preseke. Oni će se sigurno naći u rezultujućem poligonu, pa te tačke odmah dodamo. Sada prođemo kroz sve tačke početnog poligona i proverimo da li se nalaze unutar klip poligona. U rezultujući poligon dodamo samo one tačke koje su i u klip poligonu. Analogno radimo za sve tačke klip poligona – ako se tekuća tačka nalazi i u glavnom poligonu, dodaćemo je u rezultujući poligon, inače je odbacujemo.

Za ispitivanje da li je data tačka unutar poligona, može se iskoristiti algoritam koji broji preseke desno usmerenog zraka sa centrom u datoj tački sa stranicama poligona.

Saterlend-Hodžmanov algoritam

Pogledajmo najpre pseudokod Saterlend-Hodžmanovog algoritma.

```
List outputList = subjectPolygon;
for (Edge clipEdge in clipPolygon) do
   List inputList = outputList;
   outputList.clear();
   Point S = inputList.last;
   for (Point E in inputList) do
      if (E inside clipEdge) then
         if (S not inside clipEdge) then
            outputList.add(ComputeIntersection(S,E,clipEdge));
         end if
         outputList.add(E);
      else if (S inside clipEdge) then
         outputList.add(ComputeIntersection(S,E,clipEdge));
      end if
      S = E;
   done
done
```

Ideja algoritma je da se prođe kroz stranice klip poligona, a zatim kroz sve tačke tekućeg poligona koji se ažurira. U svakom koraku imamo početnu i završnu tačku (S i E) koje čine stranicu tekućeg poligona.

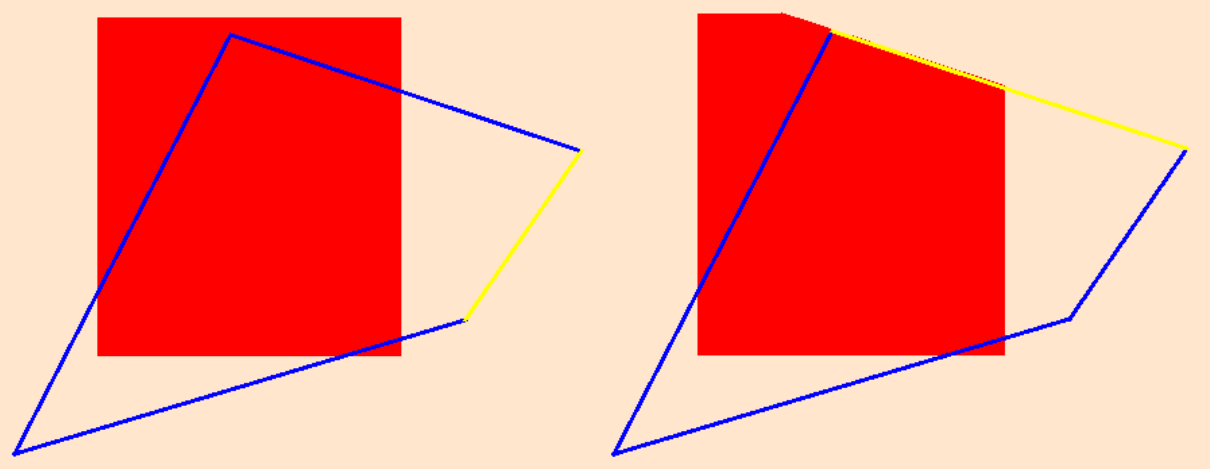
Najpre će S biti poslednja tačka u tekućem poligonu, a E tačka tekućeg poligona u tekućoj iteraciji. Za svaku sledeću stranicu, brišemo sadržaj tekućeg poligona i dodajemo ažurirane tačke. Ispitujemo u kakvom su one položaju u odnosu na tekuću stranicu klip poligona. Razmatramo nekoliko slučajeva:

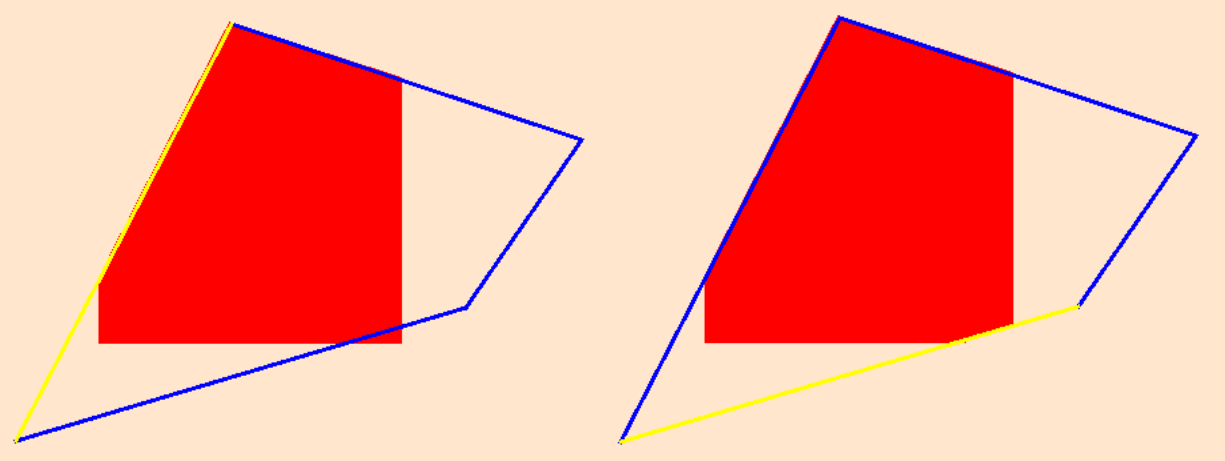
- Ako je tačka E ispred tekuće stranice klip poligona (što proveravamo orijentacijom 3 tačke – početka i kraja tekuće stranice klip poligona, kao i tačke E), a S iza stranice, onda je potrebno izračunati presečnu tačku duži SE i stranice poligona, i zatim je dodati u tekući poligon, zajedno sa tačkom E.
- Ako su i S i E ispred tekuće stranice klip poligona, onda u tekući poligon dodajemo samo E.
- Ako je E iza tekuće stranice klip poligona, a S ispred, onda treba izračunati presečnu tačku SE i tekuće stranice klip poligona i dodati je u tekući poligon.
- Ako su i S i E iza tekuće stranice klip poligona, nijednu od njih ne dodajemo.

Na kraju S postaje E i nastavljamo sa sledećom tačkom tekućeg poligona, dok god ih ima. Kada završimo sa svim tačkama, prelazimo na sledeću stranicu klip poligona i ponavljamo postupak sa ažuriranim rezultujućim poligonom.

Vizuelizacija algoritma

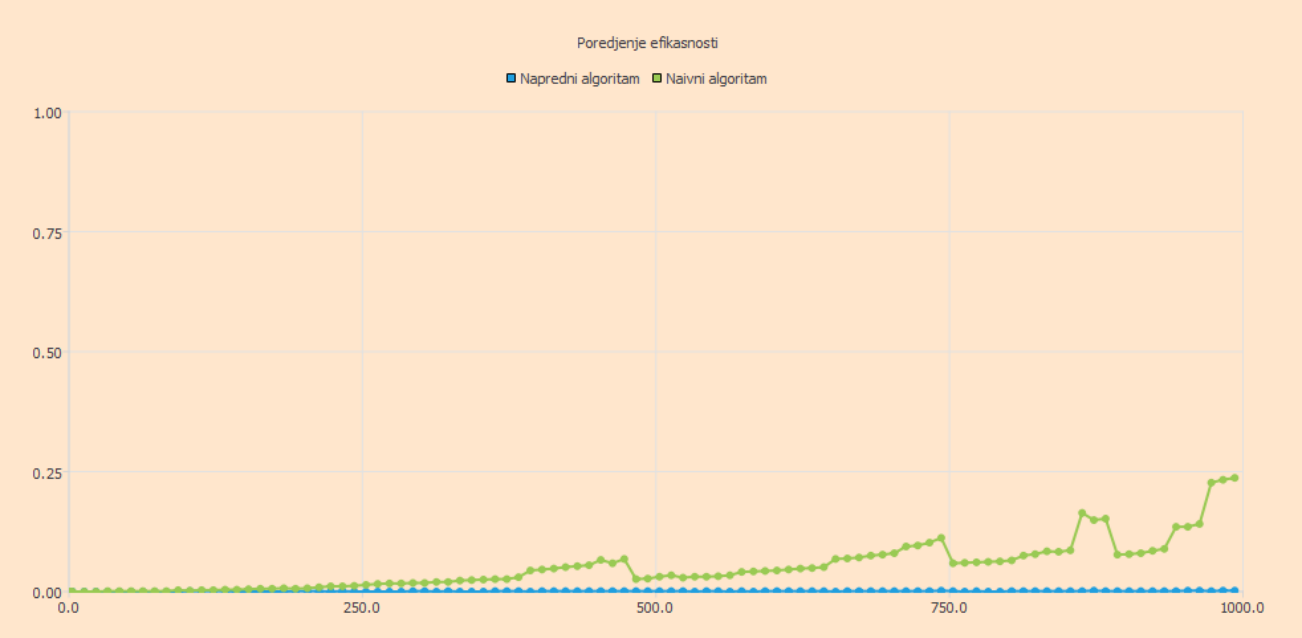
Na slikama ispod nalazi se vizualizacija algoritma. Crvenom bojom prikazan je glavni poligon - onaj koji sečemo, a plavom klip poligon – onaj kojim sečemo. Žutom bojom je označena stranica klip poligona u odnosu na koju algoritam radi u datom trenutku.





Poredjenje efikasnosti naivnog i naprednog algoritma

Sledeći grafik ilustruje koliko je optimalni algoritam efikasniji od naivnog. Na x osi se nalazi dimenzija poligona, odnosno broj stranica, dok se na y osi nalazi vreme izvršavanja algoritma.



Testiranje ispravnosti algoritma

U nastavku su opisani neki od testova.

Naziv testa	Opis testa	Ulaz	Očekivani izlaz
WrongInput	Preduslov algoritma je da klip poligon bude konveksan. U suprotnom, algoritam se neće izvršiti.	DCEL struktura u kojoj klip poligon nije konveksan	Prazan poligon
CompareResult3	U fajlu test3.off se nalazi slučaj kada je neophodno dodati i tačku klip poligona.	DCEL struktura u kojoj se teme klip poligona nalazi u unutrašnjosti glavnog poligona.	Tačke rezultujućeg poligona optimalnog i naivnog algoritma se poklapaju.
CompareResult1	U fajlu test1.off se nalazi slučaj kada postoji stranica klip poligona koja seče bar 2 stranice glavnog poligona, kao i stranica glavnog poligona koja seče bar 2 stranice klip poligona.	DCEL struktura u kojoj postoji stranica klip poligona koja seče bar 2 stranice glavnog poligona, kao i stranica glavnog poligona koja seče bar 2 stranice klip poligona.	Tačke rezultujućeg poligona optimalnog i naivnog algoritma se poklapaju.
CompareResult4	U fajlu test4.off se nalazi slučaj kada je glavni poligon u potpunosti unutar klip poligona.	DCEL struktura u kojoj je glavni poligon u potpunosti unutar klip poligona.	Tačke rezultujućeg poligona optimalnog i naivnog algoritma se poklapaju.
CompareResult5	U fajlu test5.off se nalazi slučaj kada je klip poligon u potpunosti unutar glavnog poligona.	DCEL struktura u kojoj je klip poligon u potpunosti unutar glavnog poligona.	Tačke rezultujućeg poligona optimalnog i naivnog algoritma se poklapaju.
CompareResultRandom	Zadati su nasumični poligonu.	Nasumični poligoni	Tačke rezultujućeg poligona optimalnog i naivnog algoritma se poklapaju.