Ugotavljanje vsebnosti točk nad posplošenimi mnogokotniki

Matej Gomboši
Laboratorij za geometrijsko modeliranje in algoritme multimedijev
Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru, Smetanova 17, 2000 Maribor
matej.gombosi@uni-mb.si

Izvleček

V članku je opisan razširjen algoritem za določanje vsebnosti točk nad posplošenimi mnogokotniki, ki poleg daljic vsebujejo tudi krožne loke. Algoritem uporablja klasično metodo sekanja žarka. Razlika je v tem, da moramo testirati dve vrsti objektov. Nalogo opravimo z enostavnimi in učinkovitimi testi, ki nam hitro odgovorijo na vprašanje. Z uporabo ustreznih podatkovnih struktur nalogo rešimo zanesljivo in enostavno. Kljub razširitvi deluje algoritem še vedno v linearni časovni zahtevnosti.

Abstract

Containment Test for Generalized Polygons

The paper describes an extended algorithm for solving the point-in-polygon problem. The polygon in this case consists of straight edges and also of circular arcs. This represents a generalization of Reuleaux polygon. The algorithm uses the classical ray intersection method. The difference is that we have two types of geometric objects to test for intersections. Processing is done with simple and efficient tests, which quickly answer our question. Using the appropriate data structure, this task can be done safely and easily. Despite the extension of the classical ray intersection method, the algorithm still runs in linear time complexity.

1 Uvod

Vsebnostni test je eden osnovnih in pogosto uporabljanih algoritmov v računalniški geometriji in njenih aplikacijah [9]. To še posebej velja za računalniško grafiko, sisteme CAD/CAM in GIS aplikacije.

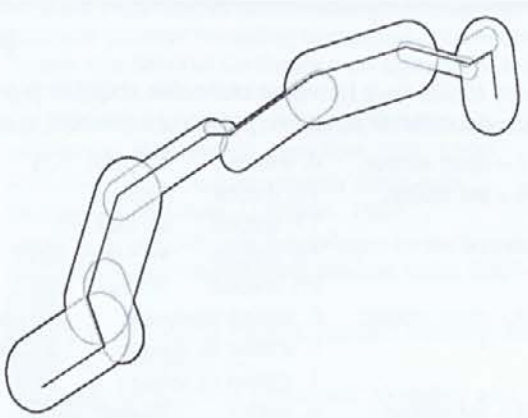
Algoritem je odvisen od tipa mnogokotnikov, s katerimi delamo:

1. mnogokotniki z ravnimi robovi,
2. mnogokotniki, ki vsebujejo tudi krožne loke.

Reševanje problemov prvega tipa je lažje in hitrejše. Večinoma se tudi srečujemo s takšnimi mnogokotniki. Posledica tega je, da so bile dosedanje raziskave usmerjene pretežno v to smer. Tako obstaja precej dobrih algoritmov, ki rešujejo ta problem: metoda sekanja žarka [4], [6], kodirani koordinatni sistem [2], metoda trikotnikov [3], Swathova metoda [10], algoritem na osnovi uniformne delitve ravnine [14].

Vsi ti algoritmi sprejmejo le mnogokotnike z ravnimi robovi. Mi pa se želimo osrediniti na posplošene mnogokotnike, ki vsebujejo tudi krožne loke. Rešitev za ta tip mnogokotnikov še ni bila podana, zato smo se odločili, da skonstruiramo svoj algoritem. Za osnovo smo vzeli metodo sekanja žarka. Dejstvo, da operiramo s krožnimi loki, nekoliko spremeni in oteži osnovni vsebnostni test. Naša želja je bila skonstruirati hiter in zanesljiv algoritem, ki bi poleg daljic učinkovito obravnaval tudi krožne loke.

Glavno motivacijo so nam predstavljali problemi iz geodezije, kjer se večkrat srečujejo s krožnimi loki. Realen primer na sliki 1 ponazarja naš problem. To je problem onesnaženja okolice cest z izpušnimi plini. Predstavljajmo si cesto, ki je ponazorjena s povezanimi daljicami. Vedeli bi radi, kako se izpušni plini širijo v



Slika 1: **Problem širjenja izpušnih plinov ponazorjen s pomočjo posplošenega mnogokotnika**

okolico ceste in kateri objekti in zgradbe so znotraj onesnaženega področja. Za predstavitev tega problema potrebujemo krožne loke, saj se plini širijo v vse smeri enako. Na sliki 1 vidimo primer ceste in širjenja plinov. Povezane daljice v sredini predstavljajo cesto. Področje širjenja izpušnih plinov je predstavljeno kot mnogokotnik s krožnimi loki. Imamo majhne mnogokotnike za posamezne odseke ceste in en skupni mnogokotnik za celo področje, ki smo ga dobili z združevanjem manjših. To nam je omogočil algoritem za tvorbo očrtij [13].

Drugo poglavje podaja nekaj osnovnih definicij, uporabljenih v tem članku. Poglavje tri predstavlja glavno idejo algoritma. Četrto poglavje obravnava robne primere. Analizo algoritma podaja peto poglavje. V šestem poglavju so podani zaključki in ugotovitve. Sedmo poglavje prikazuje relevantno literaturo.

2 Definicije

Posplošimo definicijo enostavnega mnogokotnika [7]: Imamo n točk $p_0, p_1, \ldots, p_{n-1}$ v ravnini. Pari točk p_0p_1, $p_1p_2, \ldots, p_{n-1}p_0$ so povezani z daljicami ali krožnimi loki. Določajo enostaven posplošen mnogokotnik P_g, če velja:

- sosednje daljice ali krožni loki se dotikajo v eni sami skupni točki p_i, $0 \leq i < n$,
- nesosednje daljice nimajo skupnih točk.

Daljice (označene z e_i na sliki 2) in krožni loki (označeni z a_i) s središči c_i predstavljajo robove posplošenega mnogokotnika, točke v_i pa robne točke. Zaporedje robov, ki deli ravnino v omejen in neomejen del, se imenuje zanka [7]. Vsak posplošen mnogokotnik ima natanko eno zanko. Prstan predstavlja luknjo znotraj mnogokotnika. Luknje so lahko tudi vgnezdene in tvorijo hierarhijo. Zanka je protiurno usmerjena, luknja pa sourno.

Poglejmo malo bliže strukturo krožnega loka a_i na sliki 3. Polmer je označen z r_i. Končni točki sta v_i in v_{i+1}. Ti dve točki določata tetivo. Ker je mnogokotnik orientiran, nam ta tetiva predstavlja vektor (v_iv_{i+1}) in nam pove tudi, na kateri strani tega vektorja leži mnogokotnik. V primeru slike 3 leži mnogokotnik na desni.

3 Algoritem

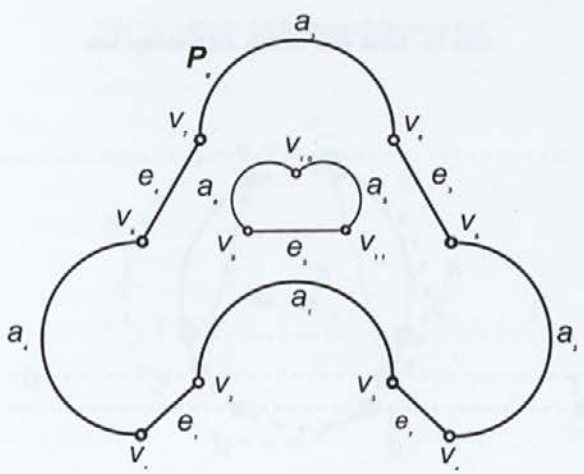
Za ugotavljanje, ali je neka točka znotraj ali zunaj mnogokotnika, se uporablja metoda sekanja žarka. Iz točke, ki jo testiram, pošljemo žarek in preverimo kolikokrat seka mnogokotnik. Za določitev vsebnosti uporabimo liho-sodo pravilo. To pravi, da če imamo liho število presečišč, je točka znotraj mnogokotnika, če jih je pa sodo število, je točka zunaj mnogokotnika. Za lažje in hitrejše računanje je naš žarek vodoraven. Tako dobimo vodoraven poltrak. Usmerjenost levo ali desno ni pomembna. Na sliki 4 vidimo primer žarka p iz točke t, ki je zunaj mnogokotnika P.

Žarek seka eno daljico in dva krožna loka. Krožni lok a_1 je presekan na dveh mestih. To nam da štiri presečišča in liho-sodo pravilo pravi, da je v tem primeru točka zunaj mnogokotnika.

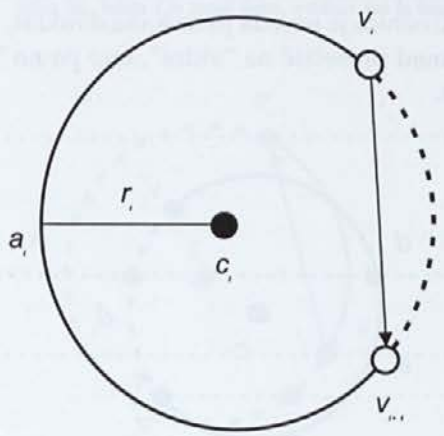
Algoritem deluje v dveh korakih:

1. iskanje presečišč med ravnimi robovi mnogokotnika in poltrakom,
2. določanje presečišč med krožnimi loki in poltrakom.

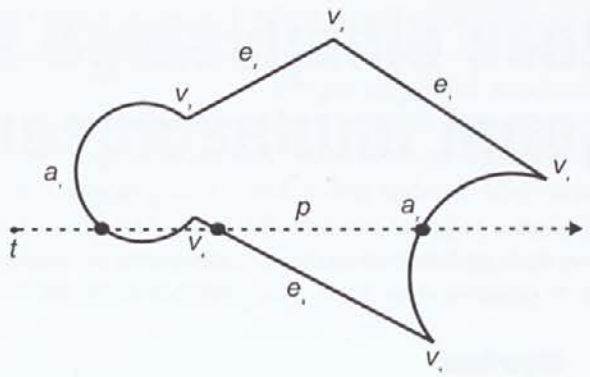
Prvi del algoritma je lažji, saj je test presečišča med vodoravnim poltrakom in daljico dobro poznan in ni



Slika 2: **Posplošen mnogokotnik P_g**



Slika 3: **Definicija krožnega loka**



Slika 4: **Metoda sekanja vodoravnega žarka iz točke *t***

zahteven. Dejanskega presečišča nam ni potrebno računati, ker nas zanima samo obstoj le-tega, ne pa njegove koordinate.

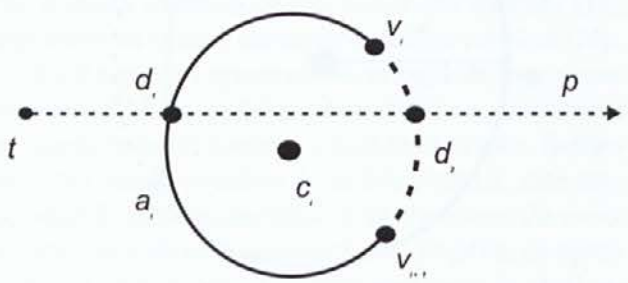
Drugi del algoritma je za nas zanimiv. Tu je bilo treba sestaviti ustrezne teste, ki bi hitro in enostavno določili obstoj presečišča med poltrakom in krožnim lokom. Slika 5 prikazuje krožni lok in poltrak, ki ga seka v točkah d_i.

Ugotoviti moramo, katera presečišča štejejo (d_1) in katera ne (d_2). Pomembno je, kako je krožni lok predstavljen. Potrebovali bomo vse podatke, omenjene v definiciji.

V splošnem imamo dve možni situaciji pri krožnih lokih:

- točka je zunaj krožnice,
- točka je znotraj krožnice.

V prvem primeru se lahko pojavita dve možnosti. Poltrak lahko seka ali pa ne seka tetive. Slika 5 kaže primer, ko poltrak seka tetivo. V tem primeru takoj brez nadaljnjih testov vemo, da imamo samo eno presečišče. Krožnica je seveda presekana dvakrat, vendar je eno izmed presečišč na "vidni", eno pa na "nevidni" strani.



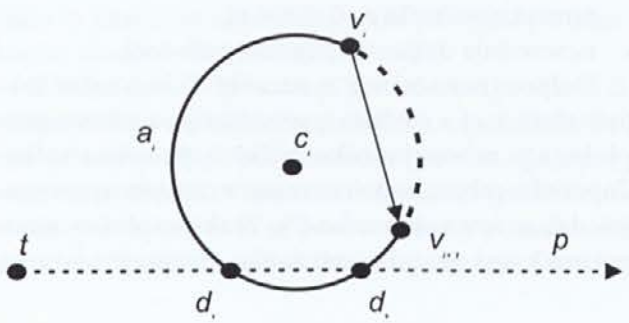
Slika 5: **Krožni lok *a*, je presekan v dveh mestih**

Tetiva je presekana, ko je žarek p znotraj vodoravnega pasu, omejenega s točkama v_i in v_{i+1} ($t.y < v_i.y$ in $t.y > v_{i+1}.y$ ali obratno) in je točka t na desni strani vektorja v_iv_{i+1} $(v_iv_{i+1} \times v_it > 0)$. Za ugotavljanje, na kateri strani vektorja leži določena točka, vedno uporabljamo vektorski produkt, ker predstavlja hitrejši test kot dejansko računanje presečišča.

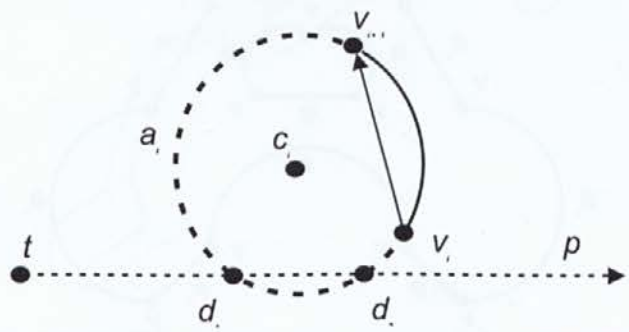
V primeru, da poltrak ne seka tetive, nam to sploh ne vpliva na rezultat, zato nam tega primera ni potrebno obravnavati. Poglejmo, zakaj.

Poltrak lahko še vedno seka preostali del krožnega loka ali pa se ga dotika. Treba bi bilo preveriti razdaljo od središča. Če je ta večja od polmera, nimamo presečišča. Če je ta manjša kot polmer, potem imamo dve presečišči. Če sta ti dve presečišči na "vidni" strani (slika 6a), obe upoštevamo, če sta na "nevidni" strani (slika 6b), pa nobenega. V obeh primerih pa to ne spremeni rezultata. Nas zanima samo sodost oz. lihost števila presečišč. Če k rezultatu prištejemo 0 ali 2, bo ostal enak. V primeru, da je razdalja enaka polmeru, imamo dotikališče, česar pa nam ni treba upoštevati, saj dotikanje ne vpliva na rezultat.

Vidimo, da smo se elegantno izognili odvečnemu testiranju in s tem pospešili algoritem.



Slika 6a: **Žarek seka »vidni« del krožnega loka**



Slika 6b: **Žarek seka »nevidni« del krožnega loka**

Podobne situacije imamo tudi v primeru, da je točka t znotraj krožnice. Točka je lahko znotraj vodoravnega pasu, ki ga določata končni točki ali izven. V primeru, da je znotraj, je število presečišč odvisno od tega, na kateri strani se nahaja "vidni" del kroga. Če je levo od tetive, nimamo presečišč (slika 7a). To pa zato, ker je potem na desni strani, torej tisti strani, v katero gre poltrak, "nevidni" del kroga. Torej se v tem primeru presečišče d_1 ne upošteva.

V nasprotnem primeru, ko je "vidni" del na desni strani, imamo seveda eno presečišče. Situacija je popolnoma enaka, le da se presečišče d_1 zdaj nahaja v "vidnem" delu kroga, ker sta "vidni" in "nevidni" del zamenjana (slika 7b).

Če pa je točka zunaj tega pasu, so ugotovitve ravno nasprotne. Če je "vidni" del na levi, imamo eno presečišče (slika 8a), v nasprotnem primeru pa ni presečišč (slika 8b).

S tem smo opisali vse možne situacije, ki lahko nastopijo pri testiranju krožnega loka. Dejanskega testiranja je zelo malo. Povzemimo zdaj vse možnosti na enem mestu (tabela 1):

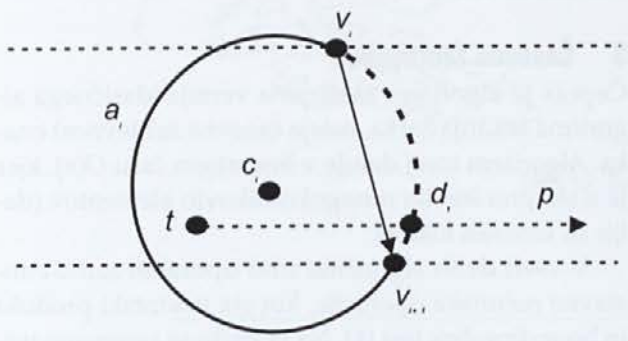
Situacija	Št. presečišč
Točka je zunaj krožnice Poltrak seka tetivo	+1
Točka je znotraj krožnice Točka je znotraj pasu tetive "Vidni" del na desni strani	+1
Točka je zunaj pasu tetive "Vidni" del na levi strani	+1

Tabela 1: **Situacije pri testiranju**

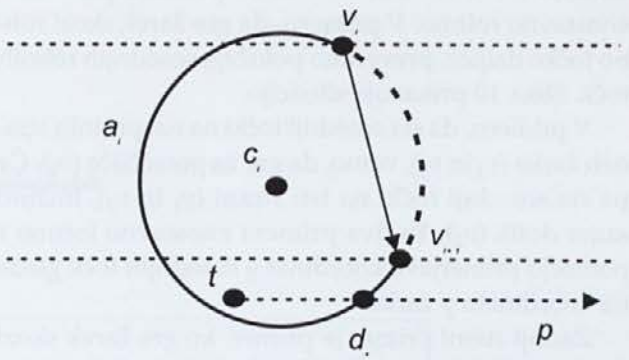
V primeru, da mnogokotnik vsebuje luknje, nam to samih pravil za testiranje ne spremeni. Vse luknje se hkrati z zanko upoštevajo pri testiranju. Tako avtomatično dobimo pravilno število presečišč.

4 Robni primeri

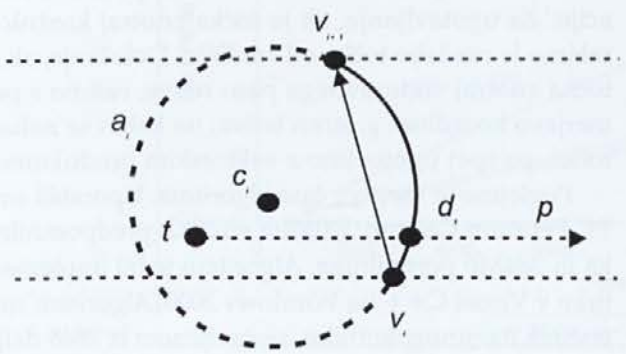
Običajen robni primer se zgodi, ko je testna točka t na mnogokotniku. To je lahko daljica ali pa krožni lok. Te situacije enostavno odkrijemo in nam ne spremenijo poteka algoritma. Če je točka na daljici, odkrijemo to s pomočjo vektorskega produkta $(v_1t_1 \times v_1v_2 = 0)$. Če je na krožnem loku, odkrijemo s pomočjo primerjave



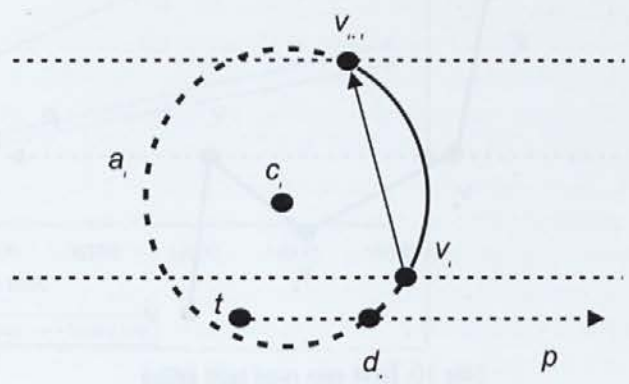
Slika 7a: **»Vidni« del krožnega loka je na levi**



Slika 8a: **Točka *t* je zunaj pasu, »vidni« del je levo**

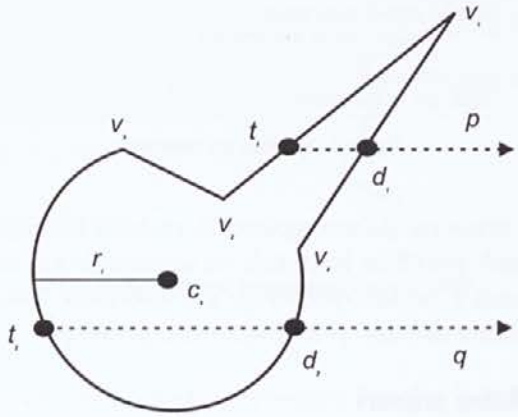


Slika 7b: **»Vidni« del krožnega loka je na desni**



Slika 8b: **Točka *t* je zunaj pasu, »vidni« del je desno**

razdalje do središča krožnega loka $d(c_1,t_2) = r_1$. Slika 9 prikazuje ta dva primera. Testna točka (t_1) je na daljici, točka (t_2) pa na krožnem loku.



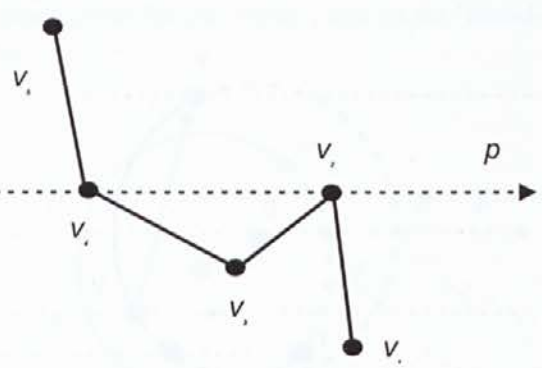
Slika 9: **Testni točki na robu mnogokotnika**

Odvisno od potrebe aplikacije se točka v takšnem primeru določi kot zunanja ali notranja.

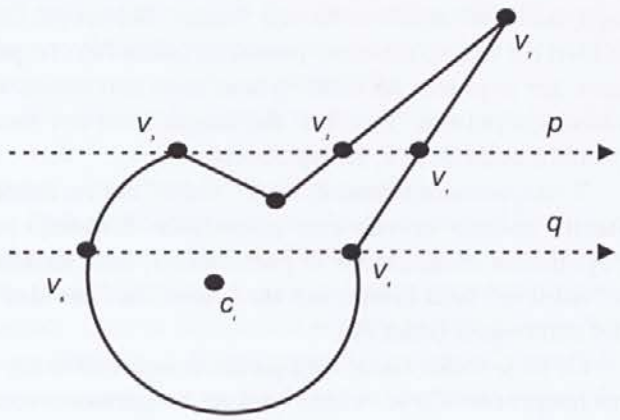
Bolj zanimivi so primeri, ko gre žarek skozi robno točko mnogokotnika. To so lahko robne točke daljic ali končne točke krožnih lokov. Te primere prav tako enostavno rešimo. V primeru, da gre žarek skozi robno točko daljice, preverimo položaja sosednjih robnih točk. Slika 10 prikazuje situacijo.

V primeru, da sta sosednji točki na nasprotnih straneh žarka (v_3 in v_5), vemo, da gre za presečišče (v_4). Če pa sta sosednji točki na isti strani (v_1 in v_3), imamo samo dotik (v_2). Ta dva primera enostavno ločimo s pomočjo primerjave koordinat y sosednjih točk glede na koordinato y žarka.

Zadnji robni primer je primer, ko gre žarek skozi končno točko krožnega loka. Slika 11 kaže dva primera. Žarek p gre skozi točko v_3, žarek q pa skozi točko v_4. V prvem primeru imamo dotik, v drugem pa presečišče. Poglejmo, zakaj.



Slika 10: **Žarek seka robni točki daljice**



Slika 11: **Žarek seka končne točke krožnega loka**

V primeru točke v_3 vidimo, da se oba sosednja dela mnogokotnika (rob v_3v_6 in krožno lok a_1) nadaljujeta na isti strani žarka. V primeru točke v_4 pa se sosednja dela (rob v_4v_1 in krožni lok a_1) nadaljujeta na različnih straneh žarka.

5 Časovna zahtevnost

Čeprav je algoritem razširjena verzija klasičnega algoritma sekanja žarka, ostaja časovna zahtevnost enaka. Algoritem torej deluje v linearnem času $O(n)$, kjer je n skupno število mnogokotnikovih elementov (daljic in krožnih lokov).

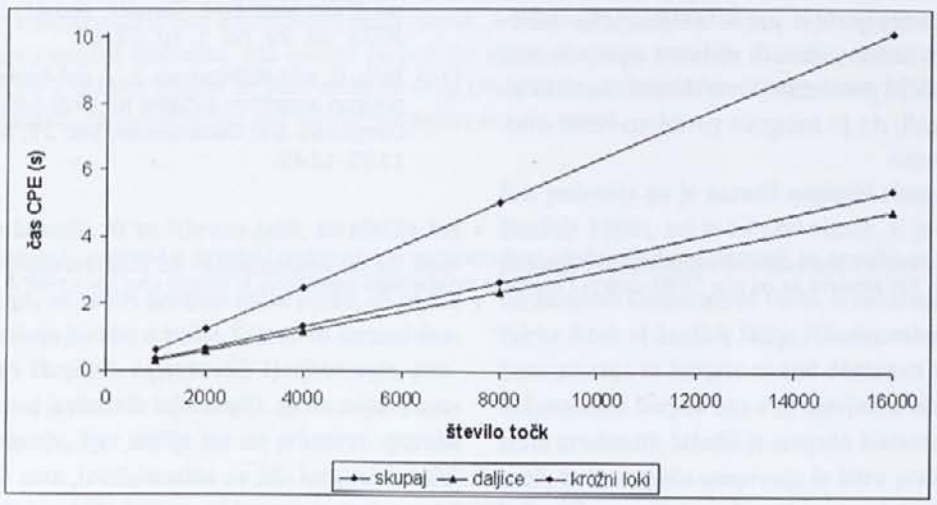
V vseh delih algoritma smo uporabili samo enostavne računske operacije, kot sta vektorski produkt in bounding-box test [1]. Na ta način se izognemo težjim računskim operacijam, saj ti dve zahtevata samo celoštevilsko množenje in seštevanje.

Pri testiranju krožnih lokov imamo podobno situacijo. Za ugotavljanje, ali je točka znotraj krožnice, rabimo le razdaljo točke od središča. Določanje, ali je točka znotraj vodoravnega pasu tetive, rešimo s primerjavo koordinat y, stran tetive, na kateri se nahaja točka, pa spet ugotovimo z vektorskim produktom.

Poglejmo še meritve časa algoritma. Uporabili smo PC Pentium Celeron 300Mhz s 128Kb predpomnilnika in 384Mb pomnilnika. Algoritem je bil implementiran v Visual C++ na Windows 2000. Algoritem smo testirali na mnogokotniku, sestavljenem iz 2848 daljic in 3334 krožnih lokov (slika 12). V tabeli 2 so podani

časi algoritma za različna števila testnih točk. Skupni čas algoritma je razdeljen tudi na čas za testiranje daljic in čas za testiranje krožnih lokov. Slika 13 kaže grafično predstavitev meritev časa.

Slika 12: **Testni mnogokotnik**



Slika 13: **Graf porabljenega časa CPE**

Št. točk	Čas daljic	Čas kr. lokov	Skupni čas
1000	0,30	0,33	0,63
2000	0,59	0,66	1,25
4000	1,18	1,32	2,50
8000	2,40	2,66	5,06
16000	4,75	5,35	10,1

Tabela 2: **Meritve časa CPE [s]**

Ugotovili smo tudi, da je razmerje med časom, porabljenim za testiranje daljic in krožnih lokov 1,1. To pomeni, da zahtevajo krožni loki okoli 10 % več procesorskega časa. Iz teh rezultatov lahko sklepamo, da so testi krožnih lokov učinkoviti, saj se skoraj enakovredno kosajo s testiranjem daljic.

6 Sklep

Predstavili smo razširjen algoritem za določanje vsebnosti točk v posplošenih mnogokotnikih, ki vsebujejo tudi krožne loke. Takšni primeri se najdejo predvsem v GIS in CAD/CAM sistemih, kjer obstajajo zelo veliki posplošeni mnogokotniki. Algoritem je sestavljen tako, da z uporabo preprostih nezahtevnih računskih operacij hitro in učinkovito reši problem.

Algoritem testira vse daljice in krožne loke na presečišče z vodoravnim žarkom. Posebej za krožne loke smo skonstruirali teste za določitev presečišča. Ti se razlikujejo od testov za daljice, vendar so še vedno dovolj hitri.

Z analizo algoritma ugotovimo, da kljub razširitvi še vedno deluje v linearni časovni zahtevnosti $O(n)$. To prikazujejo tudi meritve porabljenega časa.

Posplošeni mnogokotniki predstavljajo težjo nalogo z vsebnostnim testom. Zaradi njihove uporabnosti, predvsem v področju geodezije, smo skonstruirali ta algoritem in pokazali, da je mogoče problem rešiti enostavno in učinkovito.

7 Literatura

[1] Berg, M., M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*, Springer-Verlag, Berlin, 1997.

[2] Chen, M., Townsend, P., Efficient and consistent algorithms for determining the containment of points in polygons and polyhedra, *Proceedings of* Eurographics'87, 423–437, Elsevier Science, Amsterdam, 1987.

[3] Feito, F., J.C. Torres, A. Urena, Orientation, simplicity, and inclusion test for planar polygons, *Computers & Graphics*, Vol. 19, No. 4, 1995, 595–600.

[4] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F., *Computers Graphics-Principles and Practice, 2nd ed.*, Addison-Wesley, Reading, MA, 1990.

[5] Huang, C.-W., T.-Y. Shin, On the complexity of Point-in-Polygon Algorithms, *Computers & Geosciences*, Vol. 23, No. 1, 1997, 109–118.

[6] Manber, U., *Introduction to algorithms - a creative approach*, Addison-Wesley, Reading, MA, 1990.

[7] Mortenson, M. E., *Geometric Modeling*, John Wiley & Sohns, 1985.

[8] O'Rourke, J: *Computational Geometry in C*, Cambridge University Press, 1993.

[9] Preparata, F. P. ,M. I. Shamos, *Computational Geometry An Introduction*, Springer-Verlag, 1985.

[10] Salomon, K.B., An efficient point-in-polygon algorithm, *Computers & Geosciences*, Vol. 4, No. 2, 173–175, 1978.

[11] Taylor, G., Point in polygon test. *Survey review*, Vol. 32, 1994, 479–484.

[12] Žalik, B., Gomboši, M., Podgorelec, D., A Quick Intersection Algorithm for Arbitrary Polygons, *Proceedings of the Spring Conference on Computer Graphics SCCG 1998*, Budmerice, Slovakia, April 23–25, 1998.

[13] Žalik, Borut, Zadravec, Mirko, Clapworthy, Gordon J. Construction of a non-symmetric geometric buffer from a set of line segments. *Comput. geosci.*. ŠPrint ed.Ć, 2003, vol. 29, no. 1, str. 53–63.

[14] Žalik, B. and Kolingerova, I., A cell-based point-in-polygon algorithm suitable for large sets of points, *Computers and Geosciences*, Vol. 27, No. 10, 2001, 1135–1145.

Mag. Matej Gomboši je asistent na Fakulteti za elektrotehniko, računalništvo in informatiko Univerze v Mariboru. Diplomiral je leta 1999 in magistriral leta 2002. Kot asistent se od leta 1999 ukvarja z algoritmi računalniške geometrije in njihovo uporabo v realnih problemih.