

# Presečišča Bezierovih krivulj

## Pristop s subdivizijskim algoritmom

BENJAMIN BENČINA

UNIVERZA V LJUBLJANI  
FAKULTETA ZA MATEMATIKO IN FIZIKO  
ODDELEK ZA MATEMATIKO

21. junij 2019

# Kazalo

1	Uvod	2
2	Bezierove krivulje	2
3	Ovojnice	2
3.1	Pravokotna ovojnica . . . . .	3
3.2	Konveksna ovojnica . . . . .	3
4	Presečišča mnogokotnikov	4
5	Subdivizija Bezierove krivulje	5
6	Presečišča dveh Bezierovih krivulj	6
7	Kdaj se robota srečata	6
8	Zaključek	6

# 1 Uvod

Denimo, da imamo dva robota, ki se gibljeta vsak po svoji Bezierovi krivulji. Radi bi izračunali, ali se robota kdaj srečata. Da rešimo problem, bomo očitno potrebovali presečišča teh krivulj, saj če se krivulji ne sekata, se tudi robotka gotovo nikoli ne srečata.

V tem delu bomo obravnavali subdivizijski algoritem za izračun presečišč dveh Bezierovih krivulj. Najprej si bomo ogledali ovojnico množice kontrolnih točk Bezierove krivulje. Natančneje, našli bomo pravokotnik, ki vsebuje vse kontrolne točke. V naslednjem koraku bomo implementirali algoritem, ki nam bo vrnil konveksno ovojnico množice kontrolnih točk, torej najmanjši konveksni mnogokotnik, ki vsebuje vse kontrolne točke. Nato se bomo vprašali, kdaj se dva mnogokotnika sekata. To bo naš kriterij sekanja, če se dve ovojnici ne sekata, se tudi krivulji ne bosta. Nato bomo definirala subdivizijo Bezierove krivulje in implementirali rekurzivni algoritem, ki bo poiskal vsa presečišča.

Poleg osnovne ideje v ozadju posameznega koraka bodo podani tudi sklici na kodne datoteke, ki bodo priložene.

## 2 Bezierove krivulje

Najprej ponovimo, kaj natančno je Bezierova krivulja.

DEFINICIJA 2.1: Bezierova krivulja  $B: [0, 1] \rightarrow \mathbb{R}^2$  določena z množico kontrolnih točk  $B = \{B_i\}_{i=0}^n$  je definirana z enačbo

$$B(t) = \sum_{i=0}^n b_{i,n}(t) B_i,$$

kjer so  $b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$  Bernsteinovi bazni polinomi.

Ta definicija je ekvivalentna definiciji z de Casteljaujevim algoritmom, ki je ne bomo posebej navajali, algoritem pa se nahaja v datoteki *deCasteljaui.m*.

## 3 Ovojnice

V tem poglavju si bomo natančneje ogledali dva tipa ovojnic končne množice kontrolnih točk. To sta pravokotnik, ki vsebuje vse kontrolne točke, in konveksna ovojnica. Kasneje si bomo s presečišči ovojnic pomagali pri iskanju presečišč krivulj. Potrebujemo naslednji izrek.

IZREK 3.1: Vsaka Bezierova krivulja je v celoti vsebovana v konveksni ovojnici svojih kontrolnih točk.

DOKAZ: Izrek sledi direktno iz konstrukcije Bezierove krivulje z deCasteljau-ovim algoritmom. Vsaka točka na Bezierovi krivulji je po deCasteljau-ovem algoritmu konveksna kombinacija kontrolnih točk in zato po definiciji vsebovana v konveksni ovojnici.  $\square$

POSLEDICA 3.2: Vsaka množica, ki vsebuje vse kontrolne točke Bezierove krivulje, vsebuje tudi celotno krivuljo.

Zadnja posledica nam pove, da za ovojnico lahko uporabimo poljuben lik, ki vsebuje vse kontrolne točke, torej lahko vzamemo računsko najlažjega. Osredotočili se bomo na pravokotnik.

### 3.1 Pravokotna ovojnica

Iščemo torej smislen pravokotnik, ki bo vseboval vse kontrolne točke Bezierove krivulje. Hkrati želimo, da je dovolj majhen, da ga lahko kasneje uporabimo za kriterij sekanja, ko bomo krivuljo smiselno manjšali.

Ideja je naslednja:

- Najdemo dve kontrolni točki  $P$  in  $Q$ , ki imata največjo medsebojno razdaljo.
- Vektor  $s$  od ene do druge točke predstavlja srednjico pravokotnika.
- Poiščemo tisto izmed preostalih kontrolnih točk, ki ima najdaljšo pravokotno razdaljo do premice, ki jo definirata vektor  $s$  in ena izmed točk  $P$  in  $Q$ .
- Vektor  $b$  naj bo vektor te pravokotne razdalje.
- Oglišča iskanega pravokotnika so potem  $P + b$ ,  $P - b$ ,  $Q + b$ ,  $Q - b$ .

Glavna lastnost tako definirane pravokotnika je, da je njegova velikost odvisna le od razdalj med kontrolnimi točkami. Če bomo krivuljo torej pravilno manjšali, bomo dobili manjši pravokotnik.

Implementiran algoritem se nahaja v datoteki *pravokotnik.m*.

### 3.2 Konveksna ovojnica

Vseeno pa bi za pravo mero sekanja radi uporabili najmanjšo množico, ki vsebuje vse kontrolne točke, torej njihovo konveksno ovojnico. Za končno množico točk  $B$  je to vedno konveksni mnogokotnik, čigar oglišča so urejena podmnožica  $K \subseteq B$ . Algoritmov za iskanje konveksne ovojnice množice točk je mnogo, vsak s svojimi prednostmi in slabostmi, osredotočili pa se bomo na *Grahamov algoritem* (angl. *Graham Scan*). Razloga za to sta dva. Prvič,

ideja algoritma je jedrnata, intuitivna in lahko razumljiva. Drugič, časovna zahtevnost algoritma je med bolj optimalnimi. S preprostim razmislekom lahko utemeljimo, da je hitrost iskanja konveksne ovojnice v optimalnem primeru navzdol omejena s sortiranjem množice točk  $B$ . Konveksne ovojnice torej ne moremo najti hitreje kot  $\mathcal{O}(n \log n)$ , kjer je  $n = |B|$ . Časovna zahtevnost Grahamovega algoritma za iskanje konveksne ovojnice pa je približno enaka  $\mathcal{O}(n \log n)$ .

Ideja algoritma je naslednja:

- Najdemo najbolj levo najnižjo točko  $P_0 \in B$ .
- Vse ostale točke razvrstimo po polarnem kotu vektorja  $P_i - P_0$ , kjer  $i = 1, \dots, n$ .
- Deklariramo prazen sklad, na katerega bomo nalagali trenutne kandidate za oglišča konveksne ogrinjače.
- Sprehodimo se čez urejen seznam točk in preverimo, da se z vsako novo točko premaknemo v nasprotni smeri urinega kazalca. Če to ni res, sklad praznimo dokler ni za trenutno točko to izpolnjeno.
- Ko pregled končamo, nam na skladu ostane urejen seznam oglišč konveksne ovojnice.

Vredno je omeniti, da je izbira spodnje leve točke popolnoma arbitrarna. Zagotavlja nam le, da bo polarni kot vsakega vektorja  $P_i - P_0$ , kjer  $i = 1, \dots, n$ , vedno med 0 in  $\pi$ , s čimer se izognemo problemom z negativnimi koti ter poskrbimo, da bo točka  $P_0$  vedno oglišče ovojnice. S podobnim argumentom bi lahko izbrali poljubno robno točko in primerno prilagodili sortiranje kotov.

Ideja algoritma je torej, da si izberemo začetno oglišče in nato v nasprotni smeri urinega kazalca tvorimo pozitivno orientiran konveksni mnogokotnik. Implementiran in dobro komentiran algoritem se nahaja v datoteki *graham-Scan.m*, funkcijo pa lahko dalje grafično preizkušate z datoteko *testGraham.m*.

Pošteno je še opomniti, da programska jezika MATLAB in Octave že imata implementirano funkcijo *convhull*, ki nam vrne indekse točk konveksne ovojnice množice točk  $B$ .

## 4 Presečišča mnogokotnikov

Sedaj, ko smo definirali ovojnice, se lahko vprašamo, kdaj se dve ovojnici sekata. Ker sta oba obravnava tipa v svojem bistvu mnogokotnika, nas bo

v tem poglavju zanimalo, natanko kdaj se dva mnogokotnika sekata, torej kdaj imata neprazen presek.

**IZREK 4.1:** Mnogokotnika  $A$  in  $B$  imata neprazen presek natanko tedaj, ko se seka najmanj en par njunih stranic ali pa eden vsebuje kakšno oglišče drugega.

**DOKAZ:** Če se seka en par stranic mnogokotnikov  $A$  in  $B$ , potem imata očitno neprazen presek. Zato privzemimo, da se poljubni dve stranici mnogokotnikov  $A$  in  $B$  ne sekata. Potem je bodisi en vsebovan v drugem bodisi sta si disjunktna. Če je en vsebovan v drugem, je presek kar manjši od obeh pravokotnikov. Če sta si disjunktna, je njun presek po definiciji prazen.

S tem je izrek dokazan.  $\square$

To idejo lahko sedaj implementiramo v preprost algoritem, ki vrne 1, če se podana mnogokotnika sekata, in 0, če se ne.

Najprej z vgrajeno funkcijo preverimo, če se katero izmed oglišč enega mnogokotnika nahaja v drugem, nato pa v skrajnem primeru preverimo še, če se sekata kateri njuni stranici. Algoritem se nahaja v datoteki *mnogokotnika\_sekata.m*. Uporablja pomožno funkcijo *preseki\_dalje*, ki se nahaja v istoimenski datoteki.

## 5 Subdivizija Bezierove krivulje

Recimo, da imamo Bezierovo krivuljo, podano z množico kontrolnih točk  $B$ , ki je parametrizirana na intervalu  $[0, 1]$ , radi pa bi dobili kontrolne točke dela krivulje, na katerem parameter teče po intervalu  $[0, c]$  za nek  $0 < c < 1$ . Tej podkrivulji se reče *subdivizija* Bezierove krivulje pri parametru  $c$ . Naredimo lahko še več, pri parametru  $c$  nas zanimajo kontrolne točke dveh podkrivulj: tiste, pri kateri parameter teče po intervalu  $[0, c]$ , in tiste, pri kateri parameter teče po intervalu  $[c, 1]$ . Na srečo nam kontrolne točke obeh podkrivulj da de Casteljaujev algoritem pri parametru  $c$ . Kontrolne točke prvega dela krivulje so prvi elementi vsakega stolpca v algoritemski shemi, kontrolne točke drugega pa zadnji elementi vsakega stolpca, kot ilustrirano na spodnji sliki.

Funkcija, ki sprejme kontrolne točke Bezierove krivulje in parameter  $c$  ter vrne kontrolne točke obeh subdivizijskih podkrivulj se nahaja v datoteki *sub\_demo.m*.

$$\begin{array}{rcl}
\beta_0 & = & \beta_0^{(0)} \\
& & \beta_0^{(1)} \\
\beta_1 & = & \beta_1^{(0)} \\
& & \vdots \\
& & \beta_0^{(n)} \\
\vdots & & \vdots \\
\beta_{n-1} & = & \beta_{n-1}^{(0)} \\
& & \beta_{n-1}^{(1)} \\
\beta_n & = & \beta_n^{(0)}
\end{array}$$

## 6 Presečišča dveh Bezierovih krivulj

Zlahka opazimo, da pri subdivizijskem parametru  $c = \frac{1}{2}$  krivuljo ravno razpolovimo. Subdivizija nam torej omogoča, da na krivulji počnemo bisekcijo, s čimer pa si bomo pomagali pri iskanju presečišč. Ideja subdivizijskega algoritma za iskanje presečišč je torej naslednja:

- Okoli vsake od krivulj naredimo ovojnico.
- Če se ovojnici ne sekata, opustimo iskanje.
- Če pa se ovojnici sekata, naredimo bisekcijski korak. Izračunamo subdivizije krivulj pri parametru  $c = \frac{1}{2}$  in rekurzivno ponovimo prvi korak na vseh možnih kombinacijah podkrivulj.
- Če sta obe ovojnici manjši od v naprej določene tolerance, dodamo središče ene od njiju v množico presečišč.

Rekurzivna algoritma za pravokotno in konveksno ovojnico sta implementirana zaporedoma v datotekah *pravokotni\_bezier.m* in *konveksni\_bezier.m*. Zraven sta dodani še datoteki *test\_prav.m* in *test\_konv.m*, kjer lahko s pomočjo grafičnega vmesnika testiramo algoritem za pravokotno oziroma konveksno ovojnico.

## 7 Kdaj se robota srečata

## 8 Zaključek

## Literatura

[1] kk