

UNIVERZA V LJUBLJANI
FAKULTETA ZA MATEMATIKO IN FIZIKO
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Računalništvo in matematika – 2. stopnja

Kevin Štampar

**ORODJE ZA GRAFIČNI PRIKAZ LASTNOSTI
BÉZIERJEVIH IN PH KRIVULJ**

Magistrsko delo

Mentor: prof. dr. Emil Žagar

Ljubljana, 2024

Zahvala

Zahvaljujem se mentorju prof. dr. Emilu Žagarju za vso pomoč in usmeritve tekom magistrskega dela, za spodbudne besede ter potrpežljivost pri delu z mano, še posebej pa za zelo sproščen odnos.

Za vso podporo in zaupanje se zahvaljujem tudi partnerici, družini in prijateljem. Posebej bi izpostavil prijatelja Kristjana Bolčiča, ki mi je skozi študentsko življenje veliko pomagal tudi z deljenjem zapiskov in bil na voljo, ko se mi je s snovjo (oziroma v življenju) kje zataknilo.

Kazalo

1	Uvod	1
2	Bézierjeve krivulje	3
2.1	Bernsteinovi bazni polinomi	3
2.2	Večdimenzionalne oznake	5
2.3	Bézierjeve krivulje	6
2.4	De Casteljaujev algoritem	7
2.5	Subdivizija	10
2.6	Ekstrapolacija	12
2.7	Višanje stopnje	13
2.8	Odvodi Bézierjeve krivulje	13
3	Racionalne Bézierjeve krivulje	15
3.1	De Casteljaujev algoritem za racionalne Bézierjeve krivulje	17
3.2	Farinove točke	17
4	Zlepki Bézierjevih krivulj	20
4.1	Geometrijska zveznost	21
4.2	Konstrukcija zlepkov Bézierjevih krivulj	23
4.2.1	Enostranska konstrukcija	23
4.2.2	Simetrična konstrukcija	24
4.2.3	Posebni tipi parametrizacij (α -parametrizacije)	26
5	PH krivulje	29
5.1	Konstantna parametrična hitrost	29
5.2	Polinomska parametrična hitrost	29
5.3	Kontrolne točke Bézierjevih PH krivulj	31
5.4	Parametrična hitrost Bézierjeve PH krivulje	33
5.5	Enakomerna parametrizacija	34
5.6	Tangenta, normala in ukrivljenost	34
5.7	Racionalni odmiki krivulje	35
5.8	Reprezentacija s kompleksnimi števili	37
6	Orodje za grafični prikaz lastnosti Bézierjevih in PH krivulj - <i>Bezeg</i>	38
6.1	Uporabniški vmesnik	38
6.1.1	Orodja	38
6.1.2	Ukazi na grafu	39
6.1.3	Ukazi in transformacije izbrane krivulje	39
6.2	Implementacija	42
6.2.1	Vmesniki	43
6.2.2	Implementacije točk	43
6.2.3	Implementacije krivulj	44
6.2.4	Povezava implementacij krivulj s knjižnico JXG	49
6.2.5	Grafi	50

7 Zaključek	52
Literatura	53

Program dela

V okviru magistrskega dela opišite Bézierjeve krivulje in krivulje s pitagorejskim hodografom (PH krivulje). Predstavitev njihove osnovne lastnosti in izdelajte spletno orodje za njihovo vizualizacijo.

Osnovna literatura

1. R. Farouki, *Pythagorean-hodograph curves: algebra and geometry inseparable*, 1st, Springer Publishing Company, Incorporated, 2007.

Podpis mentorja:

Orodje za grafični prikaz lastnosti Bézierjevih in PH krivulj

POVZETEK

V magistrskem delu obravnavamo Bézierjeve krivulje, racionalne Bézierjeve krivulje, zlepke Bézierjevih krivulj ter PH krivulje. Skozi delo predstavimo njihove lastnosti, ki so ključne za rabo v sistemih za računalniško podprto načrtovanje (ang. computer-aided design, okrajš. CAD) in sistemih za računalniško podprto geometrijsko oblikovanje (ang. computer-aided geometric design, okrajš. CAGD) ter algoritme, ki njihovo rabo v takšnih sistemih omogočajo. Predstavimo tudi orodje za grafični prikaz lastnosti prej naštetih krivulj - Bezug. Ogledamo si njegove funkcionalnosti in pokažemo osnovne ideje za tem, kako je orodje implementirano in kako so v orodju implementirane prej naštete krivulje.

Tool for graphical representation of properties of Bézier and PH curves

ABSTRACT

An abstract of the work is written here. This includes a short description of the content and not the structure of your work.

Math. Subj. Class. (2020): 74B05, 65N99

Ključne besede: integracija, kompleks

Keywords: integration, complex

1 Uvod

Za namene računalniško podprtga geometrijskega oblikovanja (ang. computer-aided geometric design, okrajš. CAGD) v avtoindustriji, sta v zgodnjih šestdesetih letih prejšnjega stoletja francoska inženirja Paul de Casteljau in Pierre Bézier vzporedno uvedla Bézierjeve krivulje. To so parametrično podane krivulje, ki temeljijo na kontrolnih točkah, preko katerih uporabnik CAGD sistema s krivuljo upravlja. Zaradi enostavnega upravljanja z njimi, se je njihova raba razširila tudi v sisteme za računalniško podprto geometrijsko oblikovanje izven avtoindustrije, ter sisteme za računalniško podprto načrtovanje (ang. computer-aided design, okrajš. CAD). Danes se o Bézierjevih krivuljah (in njihovih variacijah) učimo tudi na faksu, kjer se je pojavila želja po orodju, s katerim bi lahko predavatelji grafično prikazovali njihove lastnosti. Cilj magistrskega dela je krivulje opisati, ter takšno orodje tudi izdelati. V času pisanja je orodje že izdelano, zato je delo bogato s slikovnim gradivom, ki je bilo ustvarjeno z njim.

Delo bomo začeli s poglavjem o Bézierjevih krivuljah, kjer bomo najprej spoznali Bernsteinove bazne polinome. Predstavili bomo nekaj njihovih ključnih lastnosti, nato pa bomo z njihovo pomočjo Bézierjeve krivulje tudi vpeljali. Pokazali bomo nekaj osnovnih lastnosti krivulj, nadaljevali pa bomo z De Casteljaujevim algoritmom, ki je ključen za stabilno računanje točk Bézierjeve krivulje. Podrobnejše si bomo ogledali lastnosti subdivizije, ekstrapolacije ter višanja stopnje Bézierjeve krivulje, zaključili pa bomo z odvodi Bézierjeve krivulje, ki bodo pomembni pri tvorjenju zlepkov Bézierjevih krivulj v šestem poglavju.

V tretjem poglavju se bomo posvečali racionalnim Bézierjevim krivuljam. Ogledali si bomo delovanje uteži racionalne Bézierjeve krivulje in število dodatnih prostih parametrov, ki jih dobimo v primerjavi s polinomskimi Bézierjevimi krivuljami. Pokazali bomo racionalni De Casteljaujev algoritem, ki je razširitev De Casteljaujevega algoritma iz drugega poglavja. Nato pa bomo predstavili še Farinove točke, ki uporabniku CAGD sistema nudijo naravno kontrolo nad utežmi racionalne Bézierjeve krivulje.

Četrto poglavje je namenjeno zlepkom Bézierjevih krivulj. Najprej bomo zlepke krivulj definirali, nato pa izpeljali pogoje gladkosti za zlepke Bézierjevih krivulj. Predstavili bomo koncept geometrijske zveznosti in izpeljali nekaj pogojev za različne stopnje geometrijske zveznosti. Proti koncu poglavja bomo podali nekaj algoritmov za konstrukcijo zlepkov Bézierjevih krivulj, zaključili pa bomo s posebnimi tipi parametrizacij.

V naslednjem poglavju predstavimo krivulje s pitagorejskim hodografom (PH krivulje). Najprej pokažemo, da nelinearne polinomske krivulje ne premorejo konstantne parametrične hitrosti. Kot alternativo predstavimo PH krivulje, ki imajo polinomsko parametrično hitrost. S pomočjo Bernsteinovih polinomov nato izrazimo kontrolne točke Bézierjevih PH krivulj stopnje 3 in 5. Izpeljemo parametrično hitrost Bézierjevih PH krivulj, ki jo nato uporabimo pri iskanju približka za enakomerno parametrizacijo. Pokažemo tudi, da so funkcije tangente, normale in ukrivljenosti PH krivulje racionalne funkcije in kot posledico izrazimo odmik krivulje Bézierjeve PH krivulje kot racionalno Bézierjevo krivuljo. V zaključku poglavja izrazimo še kontrolne točke kubične in kvintične Bézierjeve PH krivulje s kompleksnimi števili.

V šestem poglavju se posvetimo spletnemu orodju (Bezeg) za grafični prikaz la-

stnosti krivulj, ki je nastalo v okviru tega dela. V prvem delu poglavja predstavimo uporabniški vmesnik orodja in njegove funkcionalnosti. Drugi del pa je namenjen predstavitvi implementacije orodja, kjer predstavimo arhitekturo aplikacije in opisemo osnovne ideje za njo, ter podamo nekaj primerov kode iz implementacije.

2 Bézierjeve krivulje

2.1 Bernsteinovi bazni polinomi

V tem podrazdelku bomo predstavili Bernsteinove bazne polinome in nekaj njihovih lastnosti, ki bodo ključne pri vpeljavi Bézierjevih krivulj. Začnimo z njihovo definicijo.

Definicija 2.1. Za nenegativna cela števila n je i -ti *Bernsteinov bazni polinom* podan s predpisom

$$B_i^n(t) := \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, 1, \dots, n.$$

Opomba 2.2. Kjer je potrebno, lahko definicijo razširimo tudi na indekse $i > n$, oziroma $i < 0$. Iz definicije potem sledi $B_i^n(t) = 0$.

Pri določeni stopnji n je Bernsteinovih baznih polinomov torej $n+1$. Brez dokaza povejmo, da so linearne neodvisni in zato tvorijo bazo prostora polinomov stopnje manjše ali enake n (\mathbb{P}_n). Takšni bazi pravimo *Bernsteinova baza*, polinomu, izraženemu v njej, pa pravimo *Bernsteinov polinom*.

Primer 2.3. Za primer si bomo ogledali Bernsteinove bazne polinome stopenj $n = 0, 1, 2, 3$. Pri stopnji 0 imamo konstantni polinom s funkcijskim predpisom $B_0^0(t) = 1$. Za stopnjo 1 dobimo dva polinoma s funkcijskima predpisoma $B_0^1(t) = 1 - t$ in $B_1^1(t) = t$. Funkcijski predpisi kvadratnih in kubičnih Bernsteinovih baznih polinomov pa so:

$$\begin{aligned} B_0^2(t) &= (1-t)^2, & B_1^2(t) &= 2t(1-t), & B_2^2(t) &= t^2, \\ B_0^3(t) &= (1-t)^3, & B_1^3(t) &= 3t(1-t)^2, & B_2^3(t) &= 3t^2(1-t), & B_3^3(t) &= t^3. \end{aligned}$$

Grafe polinomov iz primera si lahko ogledamo na sliki 1. \diamond

Brez dokaza v naslednjem izreku naštejmo nekaj osnovnih lastnosti Bernsteinovih baznih polinomov.

Izrek 2.4. Za Bernsteinove bazne polinome B_i^n veljajo naslednje lastnosti.

1. So nenegativni na intervalu $[0, 1]$.

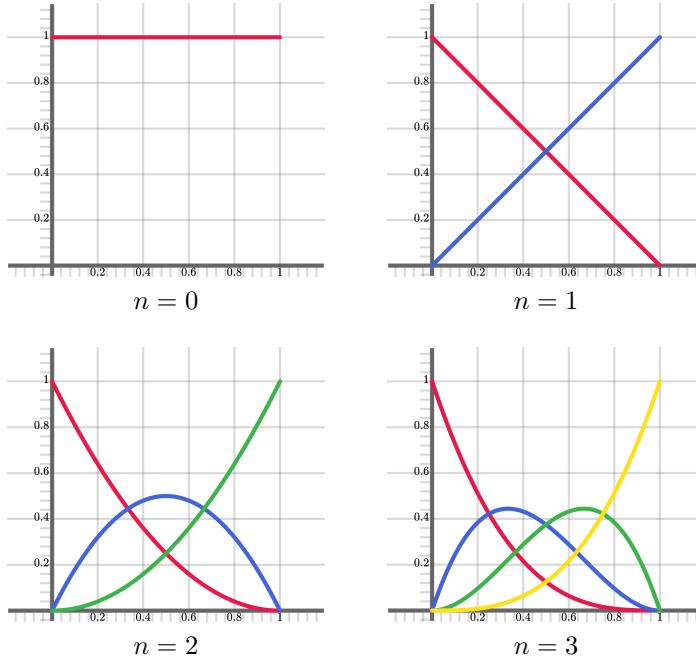
2. $B_i^n(0) = \delta_{i,0}$ in $B_i^n(1) = \delta_{i,n}$, kjer je $\delta_{i,j} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$

3. So simetrični, tj. $B_i^n(1-t) = B_{n-i}^n(t)$, $t \in \mathbb{R}$.

4. Tvorijo razčlenitev enote, tj. $\sum_{i=0}^n B_i^n(t) = 1$, $t \in \mathbb{R}$.

Prve tri lastnosti iz izreka lahko opazimo na že prej omenjeni sliki 1. Četrto lastnost pa je moč opaziti na sliki 2, kjer so prikazani naloženi ploščinski grafikoni Bernsteinovih baznih polinomov. Količina barve pri določenem parametru $t \in [0, 1]$ pove, koliko pripadajoč polinom B_i^n prispeva k razčlenitvi enote.

S sledečim izrekom podamo rekurzivno zvezo za računanje vrednosti Bernsteinovih baznih polinomov.



Slika 1: Bernsteinovi bazni polinomi stopnje $n = 0, 1, 2, 3$.

Izrek 2.5. Za Bernsteinove bazne polinome stopnje $n \geq 1$ velja rekurzivna zveza

$$B_i^n(t) = (1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t), \quad t \in \mathbb{R}.$$

Izrek je enostavno dokazati s pomočjo indukcije, zato bomo dokaz izpustili. V kasnejših razdelkih bomo potrebovali tudi odvode in integrale Bernsteinovih baznih polinomov. Ker Bernsteinovi bazni polinomi stopnje n tvorijo bazo prostora \mathbb{P}_n , lahko njihove odvode izrazimo v Bernsteinovi bazi dimenzijske $n-1$. Z nekaj računanja pridemo do sledečega rezultata.

Izrek 2.6. Za odvode Bernsteinovih baznih polinomov velja zveza

$$B_i^{n'} = n(B_{i-1}^{n-1} - B_i^{n-1}).$$

S pomočjo prejšnjega izreka in indukcije pa pridemo do izreka za integrale Bernsteinovih baznih polinomov.

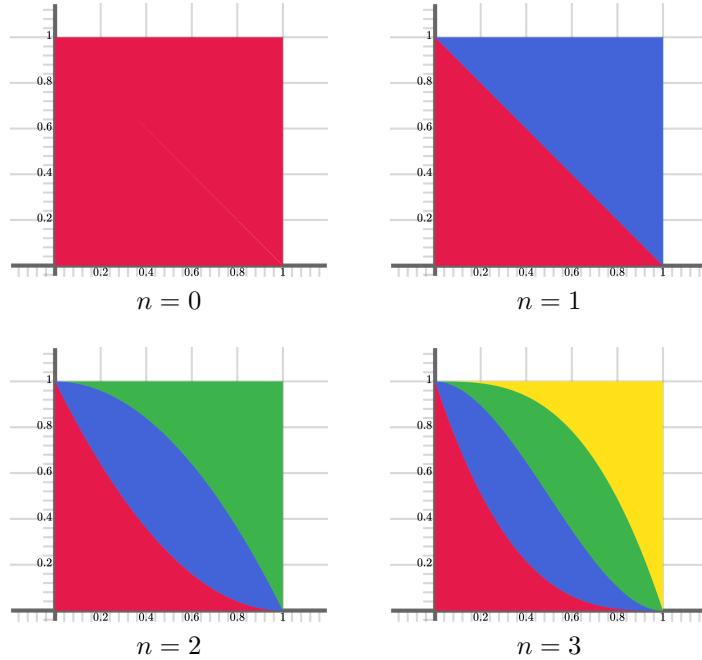
Izrek 2.7. Nedoločeni integrali Bernsteinovih baznih polinomov so

$$\int B_i^n(t) dt = \frac{1}{n+1} \sum_{k=1}^{n-i+1} B_{i+k}^{n+1}(t) + C, \quad C \in \mathbb{R}.$$

Za konec podrazdelka izpeljimo še formulo za zmnožek dveh Bernsteinovih polinomov.

Izrek 2.8. Naj bosta f in g Bernsteinova polinoma definirana kot $f = \sum_{i=0}^m \alpha_i B_i^m$ in $g = \sum_{i=0}^n \beta_i B_i^n$. Potem za njun zmnožek velja

$$fg = \sum_{i=0}^{m+n} \left(\sum_{j=\max(0, i-n)}^{\min(m, i)} \frac{\binom{m}{j} \binom{n}{i-j}}{\binom{m+n}{i}} \alpha_i \beta_{i-j} \right) B_i^{m+n}.$$



Slika 2: Naloženi ploščinski grafikoni Bernsteinovih baznih polinomov.

Dokaz. Naj bosta f in g Bernsteinova polinoma iz predpostavk izreka. Polinoma zmnožimo in dobimo

$$fg = \sum_{i=0}^m \alpha_i B_i^m \sum_{j=0}^n \beta_j B_j^n = \sum_{i=0}^{m+n} \sum_{l=0}^i \alpha_l B_l^m \beta_{i-l} B_{i-l}^n.$$

V zadnji izraz vstavimo funkcjske predpise Bernsteinovih baznih polinomov, ga poenostavimo in dobimo

$$\sum_{i=0}^{m+n} \sum_{l=0}^i \alpha_l \beta_{i-l} \binom{m}{l} \binom{n}{i-l} t^i (1-t)^{m+n-i}.$$

Slednje lahko z izpostavitvijo binoma $\binom{m+n}{i}$ predstavimo v Bernsteinovi bazi kot

$$\sum_{i=0}^{m+n} \left(\sum_{l=0}^i \alpha_l \beta_{i-l} \frac{\binom{m}{l} \binom{n}{i-l}}{\binom{m+n}{i}} \right) B_i^{m+n}.$$

V primerih, ko je $l > m$ ali $i - l > n$, imamo v števcu ulomka 0, kar privede do zapisa iz izreka. \square

2.2 Večdimenzionalne oznake

Da bodo zapisi v sledečih razdelkih bolj pregledni, bomo uvedli večdimenzionalne oznake. Točke v večdimenzionalnem prostoru bomo označili z odebeljenimi črkami, na primer $\mathbf{x} = (x_0, x_1, \dots, x_n) \in \mathbb{R}^{n+1}$. Podobno bomo odebelili črke funkcij, ki v večdimenzionalni prostor slikajo, na primer $\mathbf{f} : \mathbb{R} \rightarrow \mathbb{R}^{n+1}$.

2.3 Bézierjeve krivulje

Če v Bernsteinovem polinomu skalarne koeficiente zamenjamo s točkami, dobimo predpis parametrizacije Bézierjeve krivulje.

Definicija 2.9. Bézierjeva krivulja $\mathbf{B} : [0, 1] \rightarrow \mathbb{R}^d$ stopnje $n \in \mathbb{N}$ je polinomska krivulja podana s točkami $\mathbf{p}_i \in \mathbb{R}^d$, $i = 0, 1, \dots, n$, in parametrizacijo

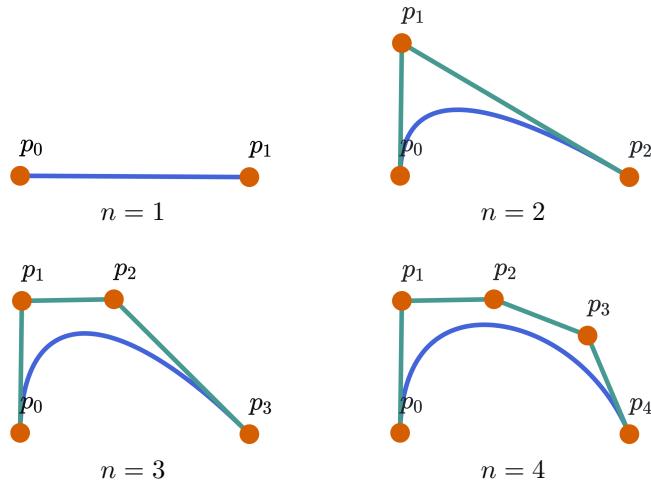
$$\mathbf{B}(t) = \sum_{i=0}^n \mathbf{p}_i B_i^n(t).$$

Točkam \mathbf{p}_i pravimo *kontrolne točke*. Če zaporedne kontrolne točke povežemo, dobimo *kontrolni poligon*.

Opomba 2.10. Kjer je potrebno, lahko definicijo razširimo tudi na stopnjo $n = 0$. Iz zgornje parametrizacije potem sledi $\mathbf{B}(t) = \mathbf{p}_0$.

Opomba 2.11. Pri slikovnem gradivu iz dela se bomo omejili na prostor dimenzije $d = 2$, torej na Bézierjeve krivulje v ravnini.

Na sliki 3 si lahko ogledamo primere Bézierjevih krivulj stopenj $n = 1, 2, 3, 4$ s pripadajočimi kontrolnimi poligoni.



Slika 3: Bézierjeve krivulje s pripadajočimi kontrolnimi poligoni za stopnje $n = 1, 2, 3, 4$.

Zapišimo sedaj nekaj osnovnih lastnosti Bézierjevih krivulj.

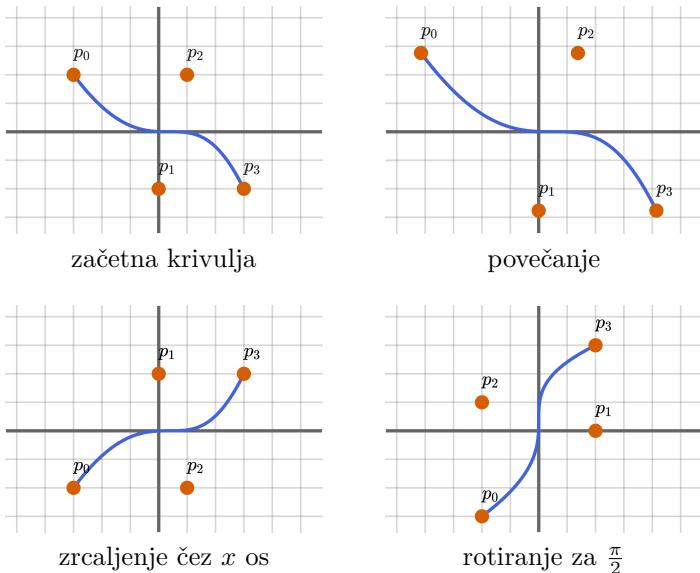
Izrek 2.12. Bézierjeva krivulja \mathbf{B} s kontrolnimi točkami \mathbf{p}_i , $i = 0, 1, \dots, n$, ima sledeče lastnosti.

1. Interpolira končni točki, tj. velja $\mathbf{B}(0) = \mathbf{p}_0$ in $\mathbf{B}(1) = \mathbf{p}_n$.
2. Je afino invariantna, tj. za poljubno afino transformacijo A velja

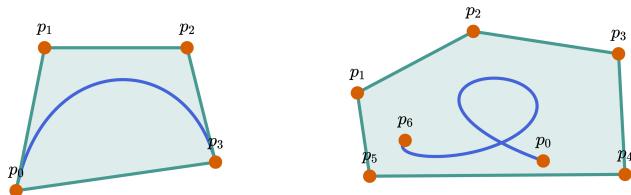
$$A \left(\sum_{i=0}^n \mathbf{p}_i B_i^n(t) \right) = \sum_{i=0}^n A(\mathbf{p}_i) B_i^n(t).$$

3. Leži znotraj konveksne ovojnice svojih kontrolnih točk.

Dokazi lastnosti so enostavni, zato jih izpustimo. Preden si lastnosti ogledamo na slikah povejmo, zakaj so pomembne za CAGD sisteme. Interpolacija končnih točk uporabniku omogoča kontrolo nad tem, kje se bo krivulja začela in kje zaključila. Zaradi afine invariance lahko uporabnikove transformacije krivulje v ozadju CAGD sistema prevedemo v transformacije kontrolnih točk. Tretja lastnost pa uporabniku s kontrolnimi točkami omogoča upravljanje krivulje, kjer je krivulja zmerom v bližini svojih kontrolnih točk. Lastnosti si sedaj oglejmo na slikah. Interpolacijo končnih točk je bilo moč videti že na sliki 3. Posledice afine invariance si lahko ogledamo na sliki 4. Na sliki 5 pa si lahko ogledamo konveksni ovojnici kontrolnih točk dveh Bézierjevih krivulj. Vidimo, da krivulji ležita znotraj njih.



Slika 4: Afine transformacije Bézierjeve krivulje.



Slika 5: Konveksni ovojnici kontrolnih točk Bézierjevih krivulj.

2.4 De Casteljaujev algoritem

Stabilnost metod je v CAD in CAGD sistemih bistvene narave. Direktno računanje vrednosti Bernsteinovih polinomov preko izrazov iz definicije 2.1 pa ni stabilno [7]. Da lahko točke Bézierjevih krivulj računamo stabilno, potrebujemo sledeč izrek.

Izrek 2.13. Označimo z $\mathbf{B}_{[\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n]}$ Bézierjevo krivuljo s kontrolnimi točkami $\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n$. Potem za poljubno realno število t in naravno število n velja rekurzivna zveza

$$\mathbf{B}(t)_{[\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n]} = (1-t)\mathbf{B}(t)_{[\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}]} + t\mathbf{B}(t)_{[\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n]}.$$

Izrek s pomočjo indukcije tudi dokažimo.

Dokaz. Za $n = 1$ zveza drži, saj je

$$\mathbf{B}(t)_{[\mathbf{p}_0, \mathbf{p}_1]} = (1-t)\mathbf{p}_2 + t\mathbf{p}_1 = (1-t)\mathbf{B}(t)_{[\mathbf{p}_0]} + t\mathbf{B}(t)_{[\mathbf{p}_1]}.$$

Indukcijski korak pa dokažemo tako, da v desni del rekurzivne zvezne iz izreka vstavimo parametrizaciji Bézierjevih krivulj in dobimo

$$(1-t)\mathbf{B}(t)_{[\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}]} + t\mathbf{B}(t)_{[\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n]} = (1-t) \sum_{i=0}^{n-1} \mathbf{p}_i B_i^{n-1}(t) + t \sum_{i=0}^{n-1} \mathbf{p}_{i+1} B_i^{n-1}(t).$$

Nato zamaknemo indeks desne vsote in skupne točke postavimo pod eno vsoto. Od tod sledi

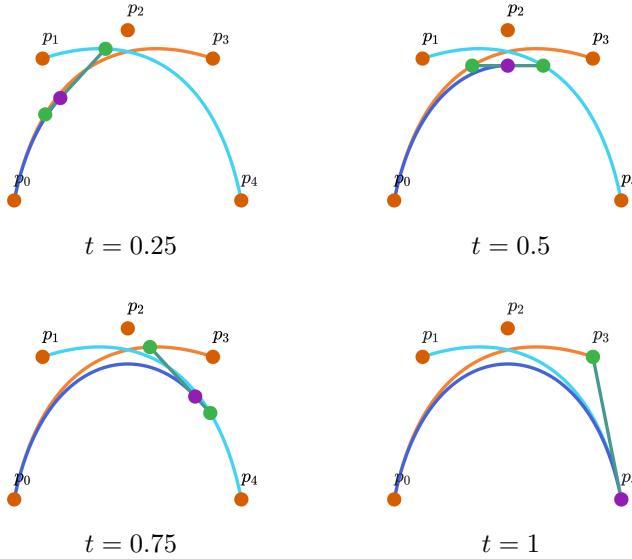
$$\begin{aligned} & (1-t) \sum_{i=0}^{n-1} \mathbf{p}_i B_i^{n-1}(t) + t \sum_{i=1}^n \mathbf{p}_i B_{i-1}^{n-1}(t) \\ &= \mathbf{p}_0(1-t)B_0^{n-1}(t) + \sum_{i=1}^{n-1} \left((1-t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \right) \mathbf{p}_i + \mathbf{p}_n B_{n-1}^{n-1}(t). \end{aligned}$$

Uporabimo še rekurzivno zvezo Bernsteinovih baznih polinomov iz izreka 2.5, da dobimo

$$\mathbf{p}_0 B_0^n(t) + \sum_{i=1}^{n-1} \mathbf{p}_i B_i^n(t) + \mathbf{p}_n B_n^n(t) = \sum_{i=0}^n \mathbf{p}_i B_i^n(t).$$

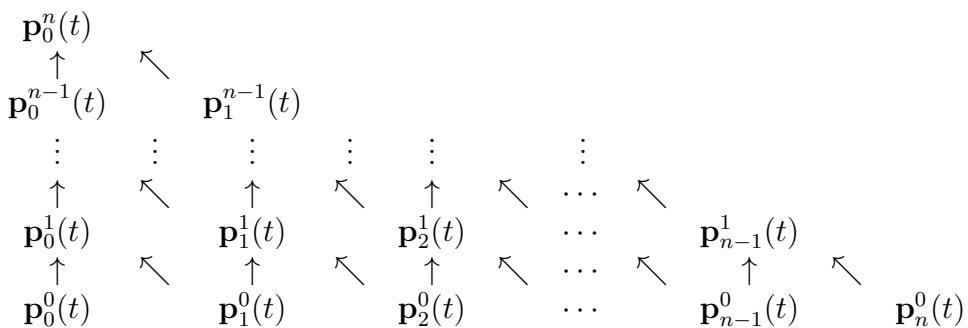
□

Na sliki 6 lahko vidimo, kako lahko točke Bézierjeve krivulje $\mathbf{B}_{[\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]}$ računamo s pomočjo točk Bézierjevih krivulj $\mathbf{B}_{[\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]}$ in $\mathbf{B}_{[\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4]}$.



Slika 6: Izračun točke Bézierjeve krivulje.

Označimo sedaj $\mathbf{p}_i^r(t) = \mathbf{B}(t)_{[\mathbf{p}_i, \mathbf{p}_{i+1}, \dots, \mathbf{p}_{i+r}]}$. Velja torej $\mathbf{p}_i^0(t) = \mathbf{p}_i$ in $\mathbf{p}_0^n(t) = \mathbf{B}(t)_{[\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n]}$. Iz izreka 2.13 sledi, da lahko točke Bézierjeve krivulje $\mathbf{B}(t)_{[\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n]}$ računamo s pomočjo *de Casteljaujeve sheme*, ki jo lahko vidimo na sliki 2.4. V shemi diagonalne puščice ponazarjajo množenje točke z vrednostjo t , vertikalne pa z vrednostjo $1 - t$. V vrhu puščic dobljene vrednosti seštejemo.



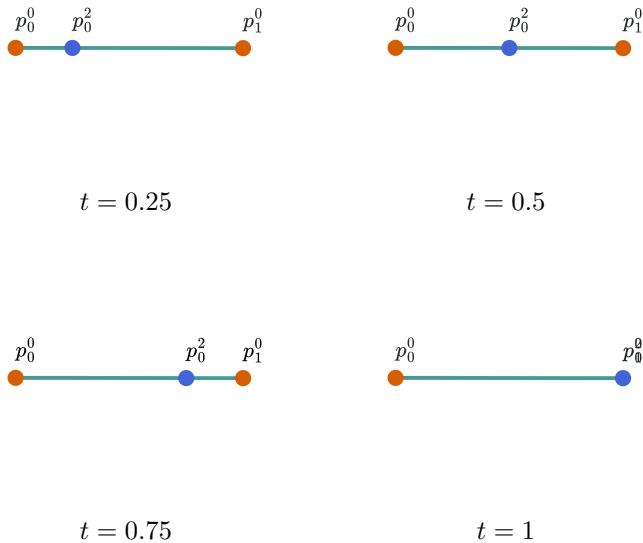
Slika 7: De Casteljaujeva shema.

Izračun točke Bézierjeve krivulje pri poljubnem parametru t lahko sedaj podamo v obliki *de Casteljaujevega algoritma 1*.

Algoritem 1 De Casteljaujev algoritem

```
p ← p0, p1, ..., pn
 $t \leftarrow t$ 
for  $i = 0, 1, \dots, n$  do
     $\mathbf{p}_i^0(t) = \mathbf{p}_i$ 
end for
for  $r = 1, 2, \dots, n$  do
    for  $i = 0, 1, \dots, n - r$  do
         $\mathbf{p}_i^r(t) = (1 - t)\mathbf{p}_i^{r-1}(t) + t\mathbf{p}_{i+1}^{r-1}(t)$ 
    end for
end for
return  $\mathbf{p}_0^n(t)$ 
```

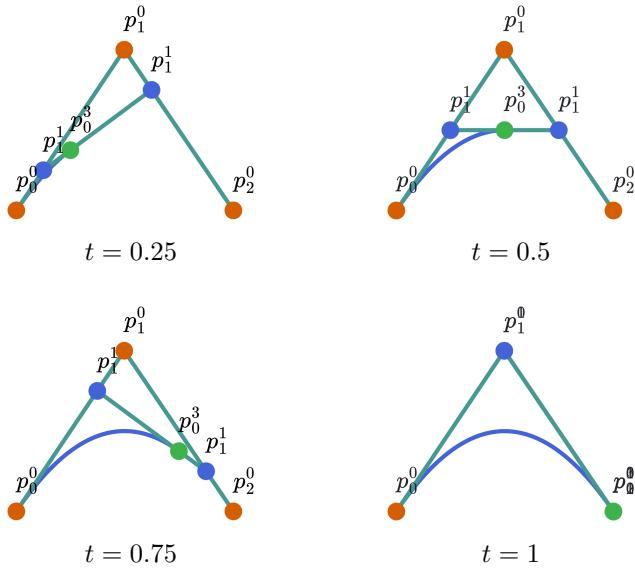
De Casteljaujev algoritem ima tudi geometrijski pomen. Pri stopnji $n = 1$ se prevede na interpolacijo dveh točk, kar lahko vidimo na sliki 8. Pri višjih stopnjah n pa algoritem predstavlja zaporedno interpolacijo točk, saj v njem za vsak nivo $r = 1, 2, \dots, n$ interpoliramo sosednje točke prejšnjega nivoja. Slednje lahko vidimo na slikah 9 in 10.



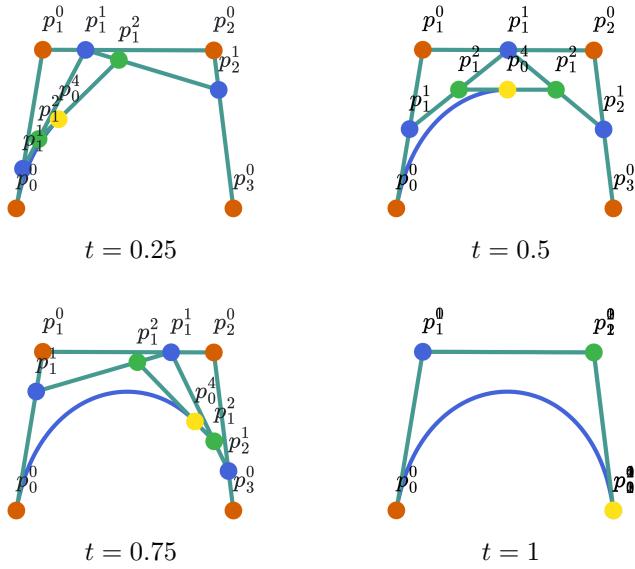
Slika 8: De Casteljaujev algoritem za $n = 1$.

2.5 Subdivizija

V CAD in CAGD sistemih se mnogokrat zgodi, da uporabnik želi ohraniti le en del Bézierjeve krivulje. Naj bo to tisti del krivulje, ki ga dobimo tako, da za prvotno krivuljo parameter t omejimo na interval $[0, t_0]$, za neko pozitivno realno število $t_0 < 1$. Ta del krivulje označimo z B_{t_0-} . Izkaže se, da lahko krivuljo B_{t_0-} podamo kot Bézierjevo krivuljo s kontrolnimi točkami $\mathbf{p}_0^i(t_0)$ za $i = 0, 1, \dots, n$, kjer točke $\mathbf{p}_0^i(t_0)$



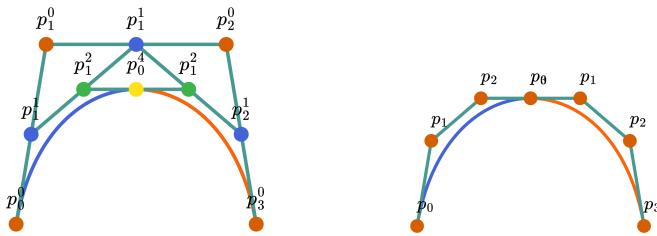
Slika 9: De Casteljaujev algoritem za $n = 2$.



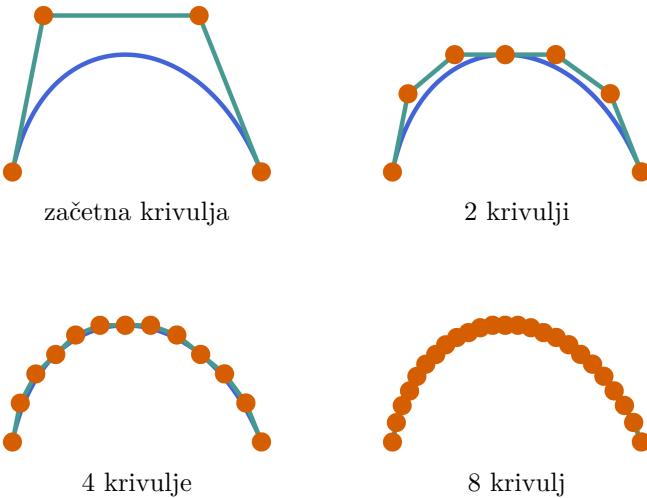
Slika 10: De Casteljaujev algoritem za $n = 3$.

dobimo iz de Casteljaujeve sheme pri $t = t_0$. Podobno se izkaže tudi to, da lahko preostali del krivulje, B_{t_0+} , podamo kot Bézierjevo krivuljo s kontrolnimi točkami $\mathbf{p}_i^i(t_0)$, $i = 0, 1, \dots, n$. Procesu deljenja krivulje na dva dela pravimo *subdivizija*. Radovedni bralci lahko dokaz trditev najdejo v delu [6]. Primer si lahko ogledamo na sliki 11.

Če izberemo sedaj $t_0 = \frac{1}{2}$ in krivuljo subdiviziramo, dobimo dve krivulji. Če subdivizijo nato na dobljenih krivuljah ponavljamo, dobimo po k korakih 2^k krivulj. Na sliki 12 si lahko ogledamo postopek za prve tri korake. Opazimo, da so kontrolni poligoni dobljenih krivulj vse bližje krivulji.



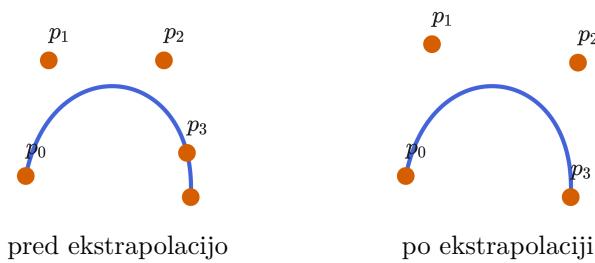
Slika 11: De Casteljaujeva shema pri $t = t_0$ razdeli krivuljo na dva dela, modrega, B_{t_0-} , in oranžnega, B_{t_0+} .



Slika 12: Ponavljanje subdivizije.

2.6 Ekstrapolacija

Ker so Bernsteinovi bazni polinomi definirani za vsa realna števila t , lahko Bézierjeve krivulje rišemo tudi izven domene $[0, 1]$. Recimo, da želimo neko Bézierjevo krivuljo risati na domeni $[0, t_0]$, kjer je $t_0 > 1$. Bézierjevo krivuljo na domeni $[0, t_0]$ lahko predstavimo kot Bézierjevo krivuljo na domeni $[0, 1]$. Pri tem kontrolne točke krivulje preberemo iz de Casteljaujeve sheme, kakor smo to storili za točke krivulje B_{t_0-} v prejšnjem razdelku, krivulja B_{t_0+} pa predstavlja ekstrapoliran del krivulje. Na sliki 13 lahko na prvem grafu vidimo risanje krivulje izven intervala $[0, 1]$, na drugem pa ekstrapolirano krivuljo.



Slika 13: Ekstrapolacija Bézierjeve krivulje.

2.7 Višanje stopnje

Nekateri algoritmi v CAD in CAGD sistemih za vhod potrebujejo dve Bézierjevi krivulji iste stopnje Recimo, da imamo Bézierjevo krivuljo \mathbf{B} stopnje n , ki jo želimo predstaviti kot Bézierjevo krivuljo stopnje $n+1$. Upoštevajoč $1-t+t=1$, lahko parametrizacijo krivulje \mathbf{B} zapišemo tudi kot

$$\mathbf{B}(t) = (1-t)\mathbf{B}(t) + t\mathbf{B}(t) = \sum_{i=0}^n \mathbf{p}_i(1-t)B_i^n(t) + \sum_{i=0}^n \mathbf{p}_i t B_i^n(t). \quad (2.1)$$

Če sedaj zvezni za Bernsteinove bazne polinome

$$\begin{aligned} (1-t)B_i^n(t) &= \frac{n+1-i}{n+1} \binom{n+1}{i} t^i (1-t)^{n+1-i} &= \frac{n+1-i}{n+1} B_i^{n+1}(t), \\ t B_i^n(t) &= \frac{i+1}{n+1} \frac{(n+1)!}{(n-i)!(i+1)!} t^{i+1} (1-t)^{n+1-i-1} &= \frac{i+1}{n+1} B_{i+1}^{n+1}(t), \end{aligned}$$

vstavimo v izraz (2.1), dobimo

$$\begin{aligned} \mathbf{B}(t) &= \sum_{i=0}^n \mathbf{p}_i \frac{n+1-i}{n+1} B_i^{n+1}(t) + \sum_{i=0}^n \mathbf{p}_i \frac{i+1}{n+1} B_{i+1}^{n+1}(t) \\ &= \sum_{i=0}^n \mathbf{p}_i \frac{n+1-i}{n+1} B_i^{n+1}(t) + \sum_{i=1}^{n+1} \mathbf{p}_{i-1} \frac{i}{n+1} B_i^{n+1}(t). \end{aligned}$$

Od tod sledi, da lahko krivuljo \mathbf{B} predstavimo kot Bézierjevo krivuljo stopnje $n+1$ s kontrolnimi točkami

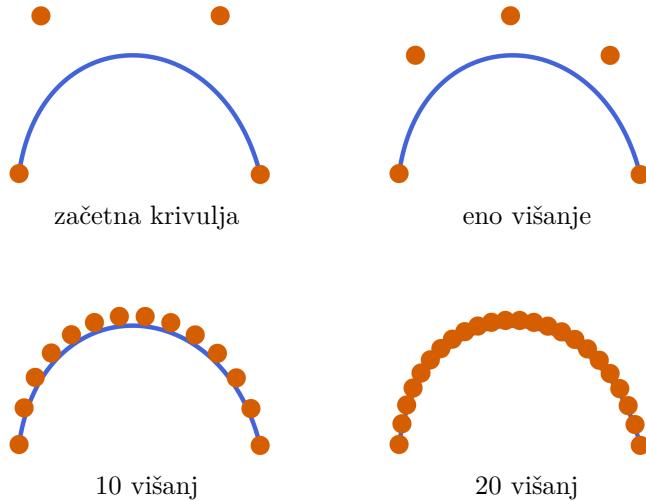
$$\mathbf{c}_0 = \mathbf{p}_0, \quad \mathbf{c}_i = \mathbf{p}_i \frac{n+1-i}{n+1} + \mathbf{p}_{i-1} \frac{i}{n+1}, \quad \mathbf{c}_{n+1} = \mathbf{p}_n.$$

Oglejmo si kako višanje stopnje poteka na neki dani krivulji. Na sliki 14 je na prvem grafu narisana Bézierjeva krivulja stopnje 3. Na drugem grafu, smo stopnjo krivulje zvišali za 1, na tretjem in četrtem grafu pa smo naredili 10 oziroma 20 višanj stopnje začetne krivulje. Krivulja je na vseh grafih enaka, imamo le več kontrolnih točk. Opaziti je tudi možno, da so kontrolne točke z vsakim višanjem stopnje bližje začetni krivulji, kontrolni poligon pa se vedno bolj prilega začetni krivulji.

2.8 Odvodi Bézierjeve krivulje

V kasnejših razdelkih bomo potrebovali odvode Bézierjevih krivulj, zato jih na tem mestu izpeljimo. Z upoštevanjem izreka 2.6 o odvodu Bernsteinovih baznih polinomov dobimo

$$\begin{aligned} \mathbf{B}'(t) &= \left(\sum_{i=0}^n \mathbf{p}_i B_i^n(t) \right)' = n \sum_{i=0}^n \mathbf{p}_i (B_{i-1}^{n-1}(t) - B_i^{n-1}(t)) \\ &= n \left(\sum_{i=1}^n \mathbf{p}_i B_{i-1}^{n-1}(t) - \sum_{i=0}^{n-1} \mathbf{p}_i B_i^{n-1}(t) \right) = n \sum_{i=0}^{n-1} (\mathbf{p}_{i+1} - \mathbf{p}_i) B_i^{n-1}(t). \end{aligned}$$



Slika 14: Višanje stopnje Bézierjeve krivulje.

Za krajše zapise višjih odvodov Bézierjeve krivulje bomo uvedli operator *prema differenci*, ki ga označimo z Δ . Operator deluje na točki \mathbf{p}_i , definiramo pa ga rekurzivno kot

$$\Delta^0 \mathbf{p}_i = \mathbf{p}_i, \quad \Delta^1 \mathbf{p}_i = \Delta \mathbf{p}_i = \mathbf{p}_{i+1} - \mathbf{p}_i, \quad \Delta^k \mathbf{p}_i = \Delta^{k-1} \mathbf{p}_{i+1} - \Delta^{k-1} \mathbf{p}_i.$$

Za naravno število k lahko v zaključeni obliki podamo ekvivalenten zapis

$$\Delta^k \mathbf{p}_i = \sum_{j=0}^k \binom{k}{j} (-1)^{k-j} \mathbf{p}_{i+j}.$$

Opomba 2.14. Iz definicije je očitno, da je operator Δ^k definiran le na kontrolnih točkah z indeksi manjšimi ali enakimi $n - k$.

Odvode Bézierjeve krivulje lahko sedaj predstavimo s pomočjo preme difference.

Izrek 2.15. *Naj bo \mathbf{B} Bézierjeva krivulja s kontrolnimi točkami \mathbf{p}_i , $i = 0, 1, \dots, n$. Za njene odvode velja*

$$\mathbf{B}^{(k)}(t) = \frac{n!}{(n-k)!} \sum_{i=0}^{n-k} \Delta^k \mathbf{p}_i B_i^{n-k}(t), \quad k = 0, 1, \dots, n.$$

Izrek je enostavno dokazati s pomočjo indukcije, zato bomo dokaz izpustili.

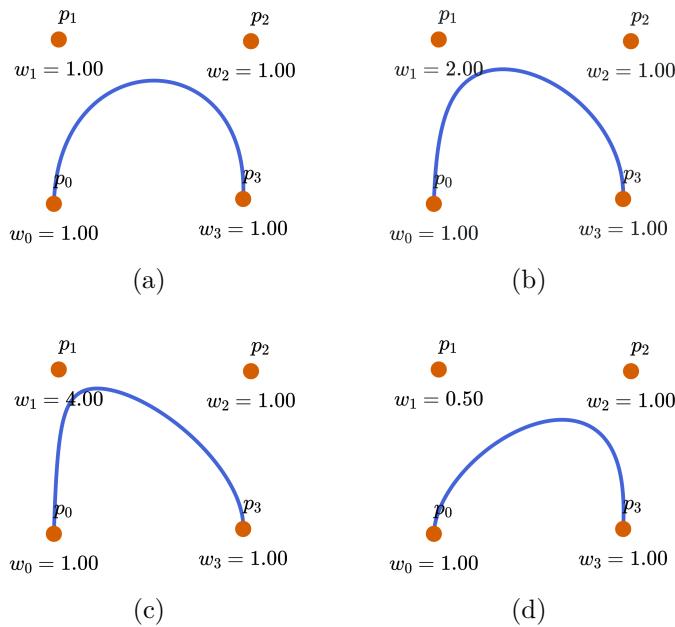
3 Racionalne Bézierjeve krivulje

Pogosto s polinomskimi Bézierjevimi krivuljami ne moremo dovolj dobro (ali eksaktно) predstaviti določenih praktično uporabnih krivulj. Med njimi so tudi takšne, ki so za CAGD sisteme zelo pomembne, na primer krožni loki. Za opis takšnih krivulj lahko posežemo po racionalnih Bézierjevih krivuljah. Racionalno Bézierjevo krivuljo stopnje n v \mathbb{R}^d dobimo tako, da Bézierjevo krivuljo stopnje n v \mathbb{R}^{d+1} projiciramo na hiperravnino $w = 1$. Točke iz \mathbb{R}^{d+1} pri tem definiramo kot (w, x_1, \dots, x_n) , projekcijo pa s predpisom $(w, \mathbf{x}) \rightarrow (1, \frac{\mathbf{x}}{w})$. Takšna izpeljava privede do naslednje definicije.

Definicija 3.1. Racionalna Bézierjeva krivulja stopnje $n \in \mathbb{N}$ je racionalna krivulja podana s kontrolnimi točkami $\mathbf{p}_i \in \mathbb{R}^d$ in utežmi $w_i \in \mathbb{R}$, $i = 0, 1, \dots, n$, ter parametrizacijo $\mathbf{R} : [0, 1] \rightarrow \mathbb{R}^d$ določeno s predpisom

$$\mathbf{R}(t) = \frac{\sum_{i=0}^n w_i \mathbf{p}_i B_i^n(t)}{\sum_{i=0}^n w_i B_i^n(t)}.$$

Uteži so prosti parametri, ki jih lahko uporabimo pri oblikovanju. Kadar so vse uteži enake, je racionalna Bézierjeva krivulja polinomska Bézierjeva krivulja z istimi kontrolnimi točkami. Da bi se izognili težavam pri deljenju z 0, ponavadi privzamemo, da so vse uteži pozitivne. Vpliv uteži si poglejmo na sliki 15. Utež spremojamo le pri kontrolni točki \mathbf{p}_1 , vse ostale uteži puščamo enake 1. Na grafu (a) je utež nastavljena na število 1, krivulja na sliki je zato polinomska Bézierjeva krivulja. Na grafih (b) in (c) lahko vidimo, da se z višanjem uteži, krivulja bliža točki \mathbf{p}_1 . Na grafu (c) pa lahko vidimo, da se z nižanjem uteži, krivulja od točke \mathbf{p}_1 oddaljuje.



Slika 15: Vpliv uteži na racionalno Bézierjevo krivuljo.

Iz zapisa parametrizacije v definiciji 3.1 lahko hitro vidimo, da množenje vseh uteži s poljubnim neničelnim številom krivulje ne spremeni. Tako lahko brez izgube

splošnosti poljubno utež fiksiramo na 1. Naslednji izrek pa pove, da lahko to storimo za dve uteži.

Izrek 3.2. *Racionalno Bézierjevo krivuljo s pozitivnimi utežmi w_i in parametrizacijo \mathbf{R} lahko reparametriziramo v $\tilde{\mathbf{R}}$ s pozitivnimi utežmi \tilde{w}_i tako, da velja $\tilde{w}_0 = \tilde{w}_n = 1$.*

Dokaz izreka bo konstrukcijske narave. Našli bomo torej uteži \tilde{w}_i , katere lahko zamenjamo z utežmi w_i tako, da ohranimo isto krivuljo.

Dokaz. Naj bo \mathbf{R} racionalna Bézierjeva funkcija s poljubnimi pozitivnimi utežmi. Uporabimo reparametrizacijsko funkcijo $\varphi(t) : [0, 1] \rightarrow [0, 1]$ s predpisom $\varphi(t) = \frac{t}{\rho(1-t)+t}$, kjer je ρ pozitivno realno število. Če reparametrizacijsko funkcijo vstavimo v i -ti Bernsteinov bazni polinom dobimo

$$\begin{aligned} B_i^n(\varphi(t)) &= \binom{n}{i} \left(\frac{t}{\rho(1-t)+t} \right)^i \left(1 - \frac{t}{\rho(1-t)+t} \right)^{n-i} \\ &= \binom{n}{i} \left(\frac{t}{\rho(1-t)+t} \right)^i \left(\frac{\rho(1-t)}{\rho(1-t)+t} \right)^{n-i} \\ &= \binom{n}{i} \frac{\rho^{n-i} t^i (1-t)^{n-i}}{(\rho(1-t)+t)^n} = \frac{\rho^{n-i}}{(\rho(1-t)+t)^n} B_i^n(t). \end{aligned}$$

Reparametrizirane Bernsteinove bazne polinome sedaj vstavimo v parametrizacijo racionalne Bézierjeve krivulje, da dobimo

$$\tilde{\mathbf{R}}(t) = \mathbf{R}(\varphi(t)) = \frac{\sum_{i=0}^n \rho^{n-i} w_i \mathbf{p}_i B_i^n(t)}{\sum_{i=0}^n \rho^{n-i} w_i B_i^n(t)}.$$

Nove uteži izrazimo s starimi $\hat{w}_i = \rho^{n-i} w_i$. Želimo, da bi veljalo $\hat{w}_0 = \hat{w}_n$, zato nastavimo $\rho = \sqrt[n]{\frac{w_n}{w_0}}$. Ker velja $\hat{w}_n = w_n$ lahko uteži \hat{w}_i delimo z utežjo w_n , da dobimo željene uteži

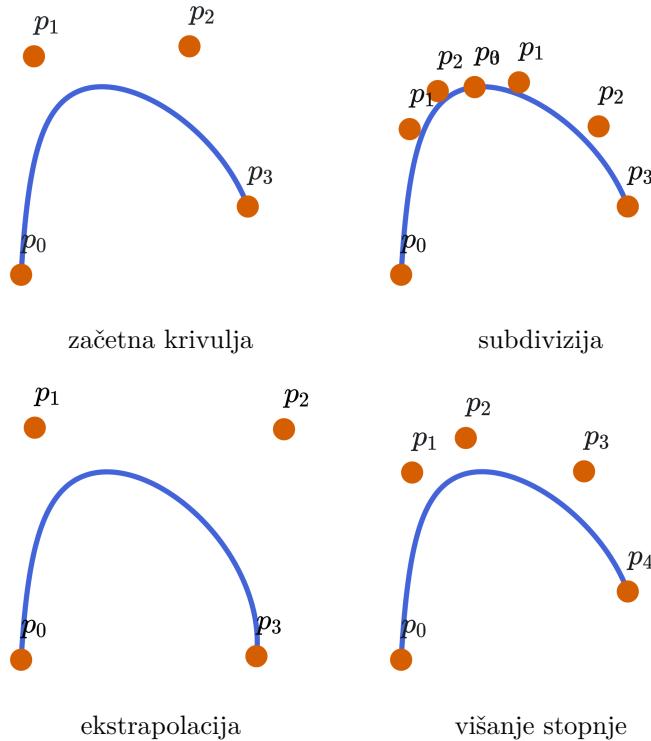
$$\tilde{w}_i = \frac{1}{w_n} \hat{w}_i = \frac{w_i}{\sqrt[n]{w_n^i w_0^{n-i}}}.$$

□

Posledica 3.3. *Z uvedbo racionalnih Bézierjevih krivulj smo dobili le $n - 1$ dodatnih prostih parametrov, glede na polinomske Bézierjeve krivulje z istim številom kontrolnih točk.*

Opomba 3.4. Če velja $w_0 = w_n = 1$ pravimo, da je racionalna Bézierjeva krivulja predstavljena v *standardni obliki*.

Brez dokaza povejmo, da lastnosti Bézierjevih krivulj, ki smo jih podali v izreku 2.12, veljajo tudi za racionalne Bézierjeve krivulje s pozitivnimi utežmi. Subdiviziranje, ekstrapoliranje in višanje stopnje polinomske Bézierjeve krivulje lahko na racionalne Bézierjeve krivulje stopnje d razširimo tako, da krivuljo zapišemo kot polinomsko Bézierjevo krivuljo stopnje $d+1$ s kontrolnimi točkami $(w_i, w_i \mathbf{p}_i) \in \mathbb{R}^{d+1}$. Polinomsko krivuljo nato subdiviziramo/ekstrapoliramo/ji zvišamo stopnjo in iz dobljenih kontrolnih točk $(\tilde{w}_i, \tilde{w}_i \tilde{\mathbf{p}}_i) \in \mathbb{R}^{d+1}$ preberemo nove kontrolne točke $\tilde{\mathbf{p}}_i \in \mathbb{R}^d$ in uteži $\tilde{w}_i \in \mathbb{R}$. Pri takšnem procesu samo ekstrapolacija ne ohranja pozitivnosti uteži. Primere si lahko ogledamo na sliki 16.



Slika 16: Subdivizija, ekstrapolacija ter višanje stopnje racionalne Bézierjeve krivulje z utežmi $w_1 = 2$, $w_0 = w_2 = w_3 = 1$.

3.1 De Casteljaujev algoritem za racionalne Bézierjeve krivulje

Točke racionalnih Bézierjevih krivulj dimenzije d bi lahko računali s pomočjo de Casteljaujevega algoritma za polinomske Bézierjeve krivulje stopnje $d + 1$ na podoben način, kot smo v prejšnjem razdelku razširili subdivizijo, ekstrapolacijo in višanje stopnje krivulje. Takšno računanje točk se izkaže za nestabilno [4], zato na tem mestu podamo algoritom 2, ki je razširitev de Casteljaujevega algoritma za racionalne Bézierjeve krivulje in točke racionalne Bézierjeve krivulje računa stabilno. Brez dokaza z naslednjim izrekom podajmo pravilnost algoritma 2.

Izrek 3.5. *Naj bo $t \in \mathbb{R}$ in $\mathbf{p}_i^r(t)$ vmesne točke iz algoritma 2. Potem je*

$$\mathbf{p}_i^r(t) = \frac{\sum_{j=0}^r w_{i+j} \mathbf{p}_{i+j} B_j^r(t)}{\sum_{i=0}^n w_{i+j} B_j^r(t)}.$$

3.2 Farinove točke

Uporabniku CAGD sistema želimo nuditi čim bolj naraven način kontroliranja uteži racionalne Bézierjeve krivulje. V ta namen lahko uteži predstavimo s *Farinovimi točkami*. Farinove točke ležijo na daljicah kontrolnega poligona, kjer i -ta Farinova točka \mathbf{f}_i deli i -to daljico kontrolnega poligona v razmerju $w_{i+1} : w_i$. Slednje lahko zapišemo kot

$$\frac{\|\mathbf{f}_i - \mathbf{p}_i\|}{\|\mathbf{f}_i - \mathbf{p}_{i+1}\|} = \frac{w_{i+1}}{w_i}.$$

Algoritem 2 Racionalni de Casteljaujev algoritem

```

 $\mathbf{p} \leftarrow \mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_n$ 
 $w \leftarrow w_0, w_1, \dots, w_n$ 
for  $i = 0, 1, \dots, n$  do
     $\mathbf{p}_i^0(t) = \mathbf{p}_i$ 
     $w_i^0(t) = w_i$ 
end for
for  $r = 1, 2, \dots, n$  do
    for  $i = 0, 1, \dots, n - r$  do
         $w_i^r(t) = (1 - t)w_i^{r-1}(t) + tw_{i+1}^{r-1}(t)$ 
         $\mathbf{p}_i^r(t) = (1 - t)\frac{w_i^{r-1}(t)}{w_i^r(t)}\mathbf{p}_i^{r-1}(t) + t\frac{w_{i+1}^{r-1}(t)}{w_i^r(t)}\mathbf{p}_{i+1}^{r-1}(t)$ 
    end for
end for
return  $\mathbf{p}_0^n(t)$ 

```

S kontrolnimi točkami in utežmi lahko Farinove točke izrazimo kot

$$\mathbf{f}_i := \frac{w_i}{w_i + w_{i+1}}\mathbf{p}_i + \frac{w_{i+1}}{w_i + w_{i+1}}\mathbf{p}_{i+1}.$$

Da bi uporabnik lahko s Farinovimi točkami kontroliral uteži, želimo s Farinovimi točkami izraziti uteži. Brez izgube splošnosti lahko za prvo utež izberemo $w_0 = 1$. Ostale točke lahko nato rekurzivno izračunamo s pomočjo izraza

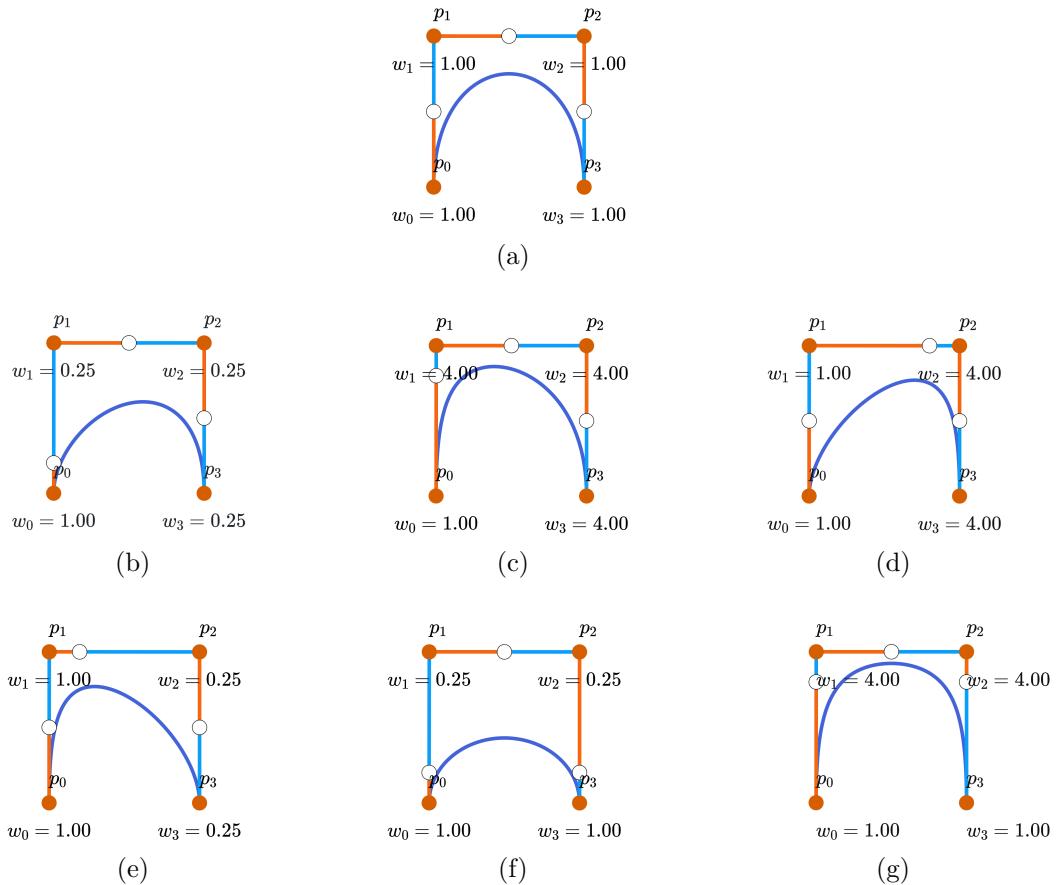
$$w_{i+1} = w_i \frac{\|\mathbf{f}_i - \mathbf{p}_i\|}{\|\mathbf{f}_i - \mathbf{p}_{i+1}\|}.$$

Po želji lahko uteži tudi standardiziramo s pomočjo formule

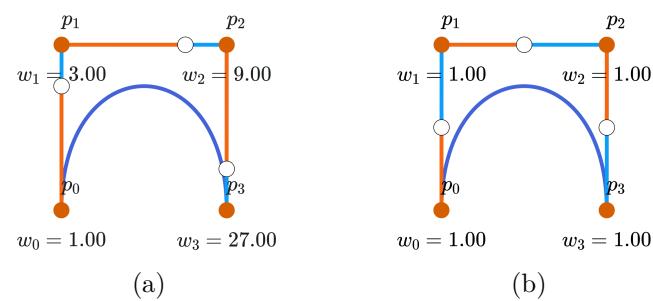
$$\tilde{w}_i = \frac{w_i}{\sqrt[n]{w_n^i}},$$

ki sledi iz dokaza izreka 3.2.

Na sliki 17 si lahko ogledamo delovanje Farinovih točk. Uteži na sliki niso standardizirane zato, da lahko bolje vidimo razmerja med njimi. Na grafu (a) Farinove točke ležijo na sredini pripadajočih daljic. Posledično so vsa razmerja $\|\mathbf{p}_i - \mathbf{f}_i\| : \|\mathbf{p}_{i+1} - \mathbf{f}_i\|$ enaka in vse uteži enake 1. Na grafu (b) lahko vidimo, da se s premikom prve Farinove točke bližje k točki \mathbf{p}_0 , k njej približa tudi krivulja. Utež w_1 , ki predstavlja razmerje $\|\mathbf{p}_0 - \mathbf{f}_0\| : \|\mathbf{p}_1 - \mathbf{f}_0\|$, je zato manjša kot 1. Na grafu (c) lahko vidimo premik Farinove točke bližje k točki \mathbf{p}_1 . Krivulja se točki približa, utež w_1 pa se poveča na več kot 1. Ker na grafih (b) in (c) druga in tretja Farinova točka ležita na sredini ustreznih daljic, so uteži w_1, w_2 in w_3 enake. Na grafih (d) in (e) lahko vidimo, kako se krivulja in uteži obnašajo ob premiku druge Farinove točke. Na grafih (f) ter (g) pa lahko vidimo obnašanje krivulje, ko premaknemo prvo ter tretjo Farinovo točko hkrati više oziroma nižje. Poglejmo še, kaj se zgodi, če vse tri točke premaknemo pomaknemo tako, da so razmerja na vseh daljicah enaka. Na levem grafu slike 18 lahko vidimo, da krivulja izgleda kakor začetna. Če uteži standardiziramo, kar smo storili na desnem grafu, vidimo, da je temu res tako. Da bi preprečili takšno izbiro Farinovih točk, lahko po vsakem premiku Farinove točke uteži standardiziramo, ter ponovno izračunamo Farinove točke.



Slika 17: Kontroliranje krivulje s Farinovimi točkami.



Slika 18: Krivulja z enakimi premiki vseh Farinovih točk, pred in po standardizaciji.

4 Zlepki Bézierjevih krivulj

Če si ponovno ogledamo de Casteljaujev algoritmom 1, lahko hitro opazimo, da je časovna kompleksnost algoritma $O(n^2)$. Računanje točk kompleksnejše Bézierjeve krivulje, z veliko kontrolnimi točkami, je zato zamudno. Da bi ohranili uporabniku naravno kontrolo krivulj ter hiter izračun točk, posežemo po zlepkih Bézierjevih krivulj.

Definicija 4.1. Zlepek krivulj $\mathbf{S} : [a, b] \rightarrow \mathbb{R}^d$ nad zaporedjem stičnih točk

$$a = u_0 < u_1 < \dots < u_{m-1} < u_m = b$$

je parametrično podana polinomska krivulja stopnje $n \in \mathbb{N}$, katere komponente so odsekoma polinomske funkcije $\mathbf{S}|_{[u_{l-1}, u_l]} \in \mathbb{P}_n^d$, $l = 1, 2, \dots, m$.

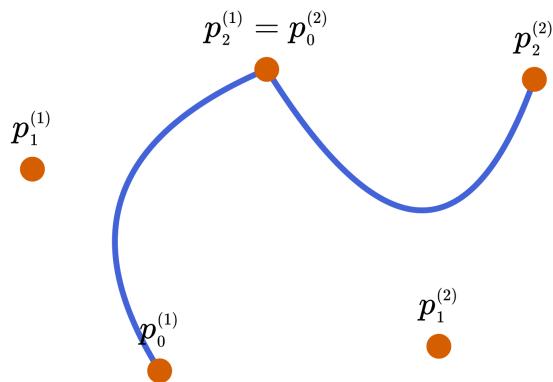
Želimo, da bi zlepk tvorili gladko neprekinjeno krivuljo, saj so takšne krivulje v CAGD sistemih najbolj uporabne. Krivulja je po definiciji na posameznih odsekih polinomska in zato tudi gladka, problem je le v stičnih točkah. Naj bo

$$\mathbf{S}(u) = \begin{cases} \mathbf{S}_1(u) = \sum_{i=0}^n \mathbf{p}_i^{(1)} B_i^n \left(\frac{u - u_0}{\Delta u_0} \right), & u \in [u_0, u_1], \\ \mathbf{S}_2(u) = \sum_{i=0}^n \mathbf{p}_i^{(2)} B_i^n \left(\frac{u - u_1}{\Delta u_1} \right), & u \in [u_1, u_2], \end{cases} \quad (4.1)$$

zlepek dveh Bézierjevih krivulj. Da bo zlepek zvezen, mora v stični točki u_1 veljati $\mathbf{S}_1(u_1) = \mathbf{S}_2(u_1)$ oziroma

$$\mathbf{p}_n^{(1)} = \mathbf{p}_0^{(2)}. \quad (4.2)$$

Takšen zlepek, si lahko ogledamo na sliki 19.



Slika 19: C^0 zlepek dveh Bézierjevih krivulj.

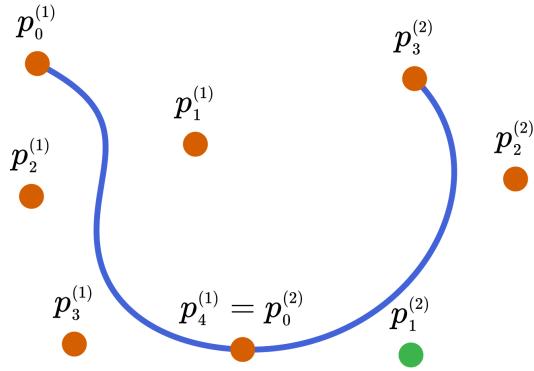
Da bo zlepek vsaj zvezno odvedljiv, mora veljati (4.2), morata pa v stični točki sovpadati tudi odvoda $\mathbf{S}'_1(u_1) = \mathbf{S}'_2(u_1)$. Iz izreka 2.15 sledi, da mora zato veljati

$$\frac{\mathbf{p}_n^{(1)} - \mathbf{p}_{n-1}^{(1)}}{\Delta u_0} = \frac{\mathbf{p}_1^{(2)} - \mathbf{p}_0^{(2)}}{\Delta u_1}. \quad (4.3)$$

Upoštevajoč (4.2) lahko enačbo zapišemo tudi kot

$$\mathbf{p}_n^{(1)} = \mathbf{p}_0^{(2)} = \frac{\Delta u_0 \mathbf{p}_1^{(2)}}{\Delta u_0 + \Delta u_1} + \frac{\Delta u_1 \mathbf{p}_{n-1}^{(1)}}{\Delta u_0 + \Delta u_1},$$

od koder je moč prebrati geometrijski pomen pogoja. Skupna točka $\mathbf{p}_n^{(1)} = \mathbf{p}_0^{(2)}$ mora namreč deliti daljico $\mathbf{p}_{n-1}^{(1)}\mathbf{p}_1^{(2)}$ v razmerju $\frac{\Delta u_0}{\Delta u_1}$. Primer takšnega zlepka si lahko ogledamo na sliki 20. Podobno lahko iz izreka 2.15 preberemo pogoje gladkosti za

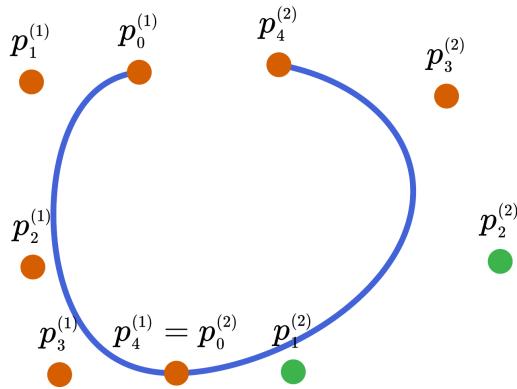


Slika 20: C^1 zlepek dveh Bézierjevih krivulj, kjer je $\Delta u_0 = \Delta u_1$.

$C^r([a, b])$,

$$\frac{1}{(\Delta u_0)^k} \Delta^k \mathbf{p}_{n-k}^{(1)} = \frac{1}{(\Delta u_1)^k} \Delta^k \mathbf{p}_0^{(2)}, \quad k = 0, \dots, r.$$

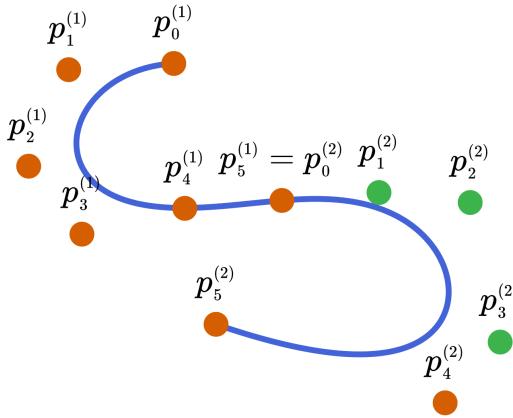
Primer C^2 oziroma C^3 zlepka si lahko ogledamo na sliki 21 oziroma sliki 22.



Slika 21: C^2 zlepek dveh Bézierjevih krivulj, kjer je $\Delta u_0 = \Delta u_1$.

4.1 Geometrijska zveznost

Da bi zlepki bili gladki, ne potrebujemo zvezne odvedljivosti v analitičnem smislu, dovolj je zvezna odvedljivost v geometrijskem smislu.



Slika 22: C^3 zlepek dveh Bézierjevih krivulj, kjer je $\Delta u_0 = \Delta u_1$.

Definicija 4.2. Zlepek krivulj

$$\mathbf{S}(u) = \begin{cases} \mathbf{S}_1(u), & u \in [u_0, u_1), \\ \mathbf{S}_2(u), & u \in [u_1, u_2], \end{cases}$$

je k -krat geometrijsko zvezno odvedljiv (G^k), če v okolici stične točke u_1 obstaja takšna regularna reparametrizacijska funkcija φ , da je $\varphi(u_1) = u_1$ in je zlepek krivulj

$$\mathbf{S}(u) = \begin{cases} \mathbf{S}_1(u), & u \in [u_0, u_1), \\ \mathbf{S}_2(\varphi(u)), & u \in [u_1, u_2], \end{cases}$$

k -krat zvezno odvedljiv.

Izpeljimo sedaj pogoje gladkosti za G^k , $k = 0, 1, 2$. Za G^0 mora veljati

$$\mathbf{S}_1(u_1) = \mathbf{S}_2(\varphi(u_1)) = \mathbf{S}_2(u_1), \quad (4.4)$$

kar je enak pogoj kakor za C^0 . Za G^1 mora veljati (4.4) in

$$\mathbf{S}'_1(u_1) = \mathbf{S}'_2(\varphi(u_1))\varphi'(u_1) = \mathbf{S}'_2(u_1)\varphi'(u_1). \quad (4.5)$$

Za G^2 pa mora veljati (4.4) in (4.5) ter

$$\mathbf{S}''_1(u_1) = \mathbf{S}''_2(\varphi(u_1))(\varphi'(u_1))^2 + \mathbf{S}'_2(\varphi(u_1))\varphi''(u_1) = \mathbf{S}''_2(u_1)(\varphi'(u_1))^2 + \mathbf{S}'_2(u_1)\varphi''(u_1).$$

Če sedaj označimo odvode reparametrizacijske funkcije v točki u_1 z $\beta_i = \varphi^{(i)}(u_1)$, lahko pogoje za G^2 zveznost izrazimo s povezovalno matriko

$$\begin{bmatrix} \mathbf{S}'_1(u_1) \\ \mathbf{S}''_1(u_1) \end{bmatrix} = \begin{bmatrix} \beta_1 & 0 \\ \beta_2 & \beta_1^2 \end{bmatrix} \begin{bmatrix} \mathbf{S}'_2(u_1) \\ \mathbf{S}''_2(u_1) \end{bmatrix}.$$

Lahko pa tudi najprej izberemo pozitiven skalar β_1 in poljuben skalar β_2 , ter konstruiramo reparametrizacijsko funkcijo, ki ustreza pogojem $\varphi' > 0$, $\varphi'(u_1) = \beta_1$ in $\varphi'(u_1) = \beta_2$. Za funkcijo φ lahko vzamemo kar Taylorjev polinom

$$\varphi(u) = u_1 + \beta_1(u - u_1) + \frac{1}{2}\beta_2(u - u_1)^2.$$

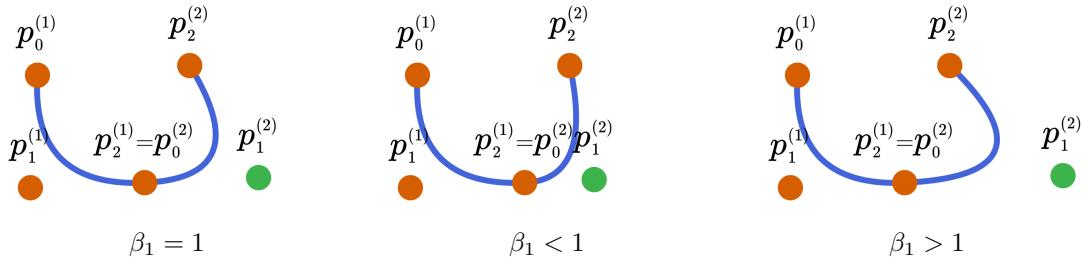
Oglejmo si sedaj, kako pogoji za geometrijsko zveznost vplivajo na zlepke Bézierjevih krivulj. Da je zlepek (4.1) G^1 zvezen, mora veljati $\mathbf{S}'_1(u_1) = \beta_1 \mathbf{S}'_2(u_1)$, iz česar sledi

$$\frac{\mathbf{p}_n^{(1)} - \mathbf{p}_{n-1}^{(1)}}{\Delta u_0} = \beta_1 \frac{\mathbf{p}_1^{(2)} - \mathbf{p}_0^{(2)}}{\Delta u_1}.$$

Če označimo sedaj $\alpha_1 = \frac{\beta_1 \Delta u_0}{\Delta u_1}$ lahko pogoj zapišemo kot

$$\mathbf{p}_n^{(1)} - \mathbf{p}_{n-1}^{(1)} = \alpha_1 (\mathbf{p}_1^{(2)} - \mathbf{p}_0^{(2)}). \quad (4.6)$$

Glede na C^1 smo torej pridobili prosti parameter, ki ga lahko v CAGD sistemih uporabimo za dodatno kontrolo nad krivuljo. Primer G^1 zlepka in vpliv izbire β_1 si lahko ogledamo na sliki 23.



Slika 23: Vpliv izbire β_1 pri G^1 zlepku.

4.2 Konstrukcija zlepkov Bézierjevih krivulj

4.2.1 Enostranska konstrukcija

Pogoje gladkosti iz prejšnjega razdelka bi lahko izpeljali tudi drugače. Da bi zlepek (4.1) bil C^∞ , bi morali krivulji \mathbf{S}_1 in \mathbf{S}_2 biti del iste Bézierjeve krivulje. Krivuljo \mathbf{S}_2 lahko zato dobimo z ekstrapolacijo krivulje \mathbf{S}_1 . Da je zlepek C^r je dovolj, da krivulja \mathbf{S}_2 ustrezata ekstrapolirani krivulji v prvih $r+1$ kontrolnih točkah, saj le te vplivajo na prvih r odvodov krivulje \mathbf{S}_2 v stični točki u_1 . Zlepek gladkosti r lahko torej konstruiramo tako, da začnemo z Bézierjevo krivuljo. Krivuljo nato ekstrapoliramo in ekstrapoliranemu delu fiksiramo prvih $r+1$ točk, ostale pa izberemo poljubno. Postopek je podrobnejše zapisan v algoritmu 3, kjer funkcija decasteljau_shema predstavlja funkcijo, ki vrača De Casteljaujevo shemo 2.4.

Algoritem 3 Enostranska konstrukcija C^r zlepka stopnje n

Vhod:

$\mathbf{p} \leftarrow \mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_m$

$u \leftarrow u_0, u_1, \dots, u_n$

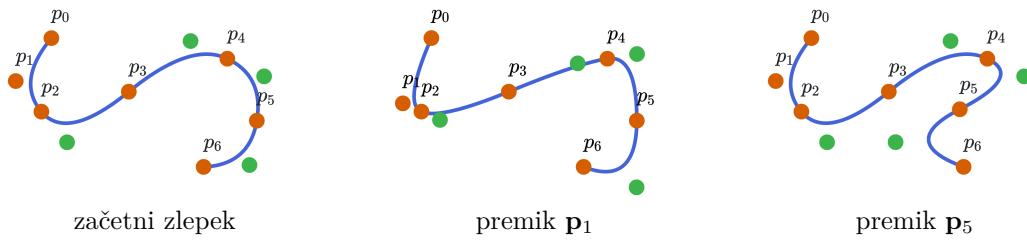
```

st_krivulj =  $\frac{m-n}{n-r}$ 
for  $i = 0, 1, \dots, n$  do
     $\mathbf{p}_i^{(1)} = \mathbf{p}_i$ 
end for
for  $j = 2, 3, \dots, st\_krivulj$  do
     $t_0 = \frac{\Delta u_j + \Delta u_{j+1}}{\Delta u_j}$ 
     $\mathbf{e} = decasteljau\_shema(\mathbf{p}^{(j-1)}, t_0)$ 
     $\{\mathbf{p}_i^{(j)}\}_{i=0}^r = \{\mathbf{e}_j^{(n)}\}_{j=n}^{n-r}$ 
    for  $k = 0, 1, \dots, (n-r)$  do
         $\mathbf{p}_{n-r+k}^{(j)} = \mathbf{p}_{j(n-r)+k}$ 
    end for
end for

return  $\{\mathbf{p}^{(i)}\}_{i=1}^{st\_krivulj}$ 

```

Zlepki, konstruirani s takšnim algoritmom, niso uporabniku prijazni, saj kontrolne točke različno vplivajo na obliko krivulje. Za primer si na sliki 24 oglejmo obnašanje tako konstruiranega kvadratnega C^1 zlepka.



Slika 24: Obnašanje enostranskega kvadratnega C^1 zlepka.

4.2.2 Simetrična konstrukcija

Uporabniku mnogo bolj prijazni so algoritmi simetričnih konstrukcij. Brez izpeljave podajmo algoritma 4 in 5 za simetrični konstrukciji kvadratnega C^1 in kubičnega C^2 zlepka. Obnašanje tako konstruiranih zlepkov si lahko ogledamo na slikah 25 in 26.

Algoritem 4 Simetrična konstrukcija kvadratnega C^1 zlepka

Vhod:

$\mathbf{p} \leftarrow \mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{m+1}$

$u \leftarrow u_0, u_1, \dots, u_m$

```

 $\mathbf{p}_0^{(1)} = \mathbf{p}_0$ 
 $\mathbf{p}_1^{(1)} = \mathbf{p}_1$ 
for  $l = 1, 2, \dots, m-1$  do
     $\mathbf{p}_1^{(l)} = \mathbf{p}_l$ 
     $\mathbf{p}_2^{(l)} = \mathbf{p}_0^{(l+1)} = \frac{\Delta u_l}{\Delta u_{l-1} + \Delta u_l} \mathbf{p}_l + \frac{\Delta u_{l-1}}{\Delta u_{l-1} + \Delta u_l} \mathbf{p}_{l+1}$ 
end for
 $\mathbf{p}_0^{(m)} = \mathbf{p}_m$ 
 $\mathbf{p}_1^{(m)} = \mathbf{p}_{m+1}$ 

return  $\{\mathbf{p}^{(l)}\}_{l=1}^m$ 

```

Algoritem 5 Simetrična konstrukcija kubičnega C^2 zlepka

Vhod:

$\mathbf{p} \leftarrow \mathbf{p}_{-1}, \mathbf{p}_0, \dots, \mathbf{p}_m, \mathbf{p}_{m+1}$

$u \leftarrow u_0, u_1, \dots, u_m$

```

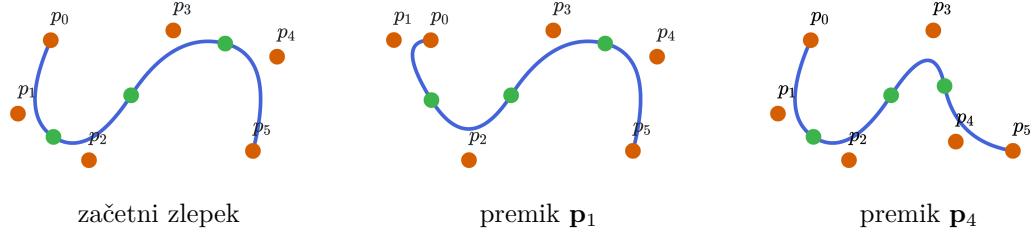
 $\mathbf{p}_0^{(1)} = \mathbf{p}_{-1}$ 
 $\mathbf{p}_1^{(1)} = \mathbf{p}_0$ 
for  $l = 1, 2, \dots, m-2$  do
     $\mathbf{p}_1^{(l+1)} = \frac{\Delta u_l + \Delta u_{l+1}}{\Delta u_{l-1} + \Delta u_l + \Delta u_{l+1}} \mathbf{p}_l + \frac{\Delta u_{l-1}}{\Delta u_{l-1} + \Delta u_l + \Delta u_{l+1}} \mathbf{p}_{l+1}$ 
     $\mathbf{p}_2^{(l+1)} = \frac{\Delta u_{l+1}}{\Delta u_{l-1} + \Delta u_l + \Delta u_{l+1}} \mathbf{p}_l + \frac{\Delta u_{l-1} + \Delta u_l}{\Delta u_{l-1} + \Delta u_l + \Delta u_{l+1}} \mathbf{p}_{l+1}$ 
end for
 $\mathbf{p}_1^{(m)} = \frac{\Delta u_{m-1}}{\Delta u_{m-2} + \Delta u_{m-1}} \mathbf{p}_{m-1} + \frac{\Delta u_{m-2}}{\Delta u_{m-2} + \Delta u_{m-1}} \mathbf{p}_m$ 
for  $l = 1, 2, \dots, m-1$  do
     $\mathbf{p}_3^{(l)} = \mathbf{p}_0^{(l+1)} = \frac{\Delta u_l}{\Delta u_{l-1} + \Delta u_l} \mathbf{p}_l + \frac{\Delta u_{l-1}}{\Delta u_{l-1} + \Delta u_l} \mathbf{p}_{l+1}$ 
end for
 $\mathbf{p}_2^{(m)} = \mathbf{p}_m$ 
 $\mathbf{p}_3^{(m)} = \mathbf{p}_{m+1}$ 

return  $\{\mathbf{p}^{(l)}\}_{l=1}^m$ 

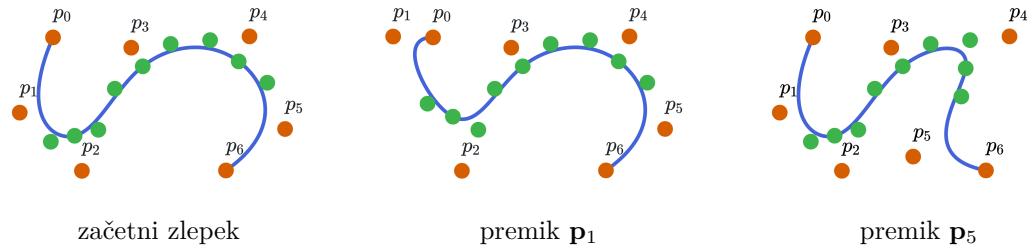
```

Tako konstruirani zlepki ustrezajo lastnostim Bézierjevih krivulj iz izreka 2.12. Primere afinih transformacij simetrično konstruiranega C^2 zlepka si lahko ogledamo na sliki 27. Na sliki 28 pa si lahko ogledamo konveksni ovojnici dveh takšnih zlepkov.

Algoritem za simetrično konstrukcijo kvadratnega C^1 zlepka lahko adaptiramo tako, da upoštevamo pogoj (4.6) za G^1 zveznost. Dobimo algoritem 6 za simetrično konstrukcijo kvadratnega C^1 zlepka, kjer števila β_l omejimo na interval $(0, 1)$. Obnašanje zlepka ob spremenjanju β_1 si lahko ogledamo na sliki 29.



Slika 25: Obnašanje simetrično konstruiranega kvadratnega C^1 zlepka.



Slika 26: Obnašanje simetrično konstruiranega kubičnega C^2 zlepka.

Algoritem 6 Simetrična konstrukcija kvadratnega G^1 zlepka

Vhod:

$$\begin{aligned} \mathbf{p} &\leftarrow \mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_m \\ \beta &\leftarrow \beta_1, \beta_2, \dots, \beta_{m-1} \end{aligned}$$

```

 $\mathbf{p}_0^{(1)} = \mathbf{p}_0$ 
 $\mathbf{p}_1^{(1)} = \mathbf{p}_1$ 
for  $l = 1, 2, \dots, m-1$  do
     $\mathbf{p}_1^{(l)} = \mathbf{p}_l$ 
     $\mathbf{p}_2^{(l)} = \mathbf{p}_0^{(l+1)} = (1 - \beta_l)\mathbf{p}_l + \beta_l\mathbf{p}_{l+1}$ 
end for
 $\mathbf{p}_0^{(m)} = \mathbf{p}_m$ 
 $\mathbf{p}_1^{(m)} = \mathbf{p}_{m+1}$ 

return  $\{\mathbf{p}^{(l)}\}_{l=1}^m$ 

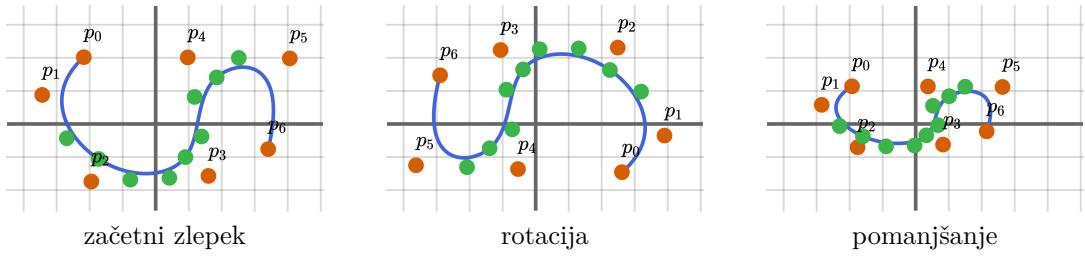
```

4.2.3 Posebni tipi parametrizacij (α -parametrizacije)

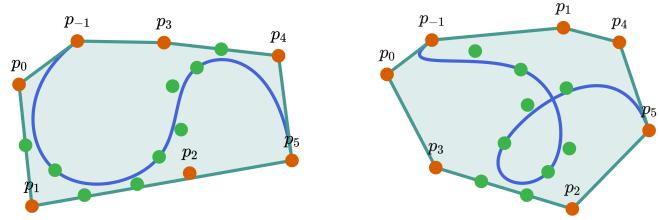
V praksi stičnih točk u_i dostikrat ne dobimo podanih, ampak jih izračunamo sami. Smiselno jih je postaviti tako, da s parametrom t po krivulji potujemo čim bolj enakomerno. Želimo torej, da za točke u_i velja

$$\frac{\Delta u_{i+1}}{\Delta u_i} = \frac{dolzina_krivulje(\mathbf{s}^{(i+1)})}{dolzina_krivulje(\mathbf{s}^{(i)})}.$$

Pri simetrično konstruiranih kubičnih C^2 zlepkih iz prejšnjega razdelka lahko dolžine krivulj $\mathbf{s}^{(i)}$ aproksimiramo z dolžino premih diferenc Δp_i . Takšnem generiranjtu točk



Slika 27: Afini transformaciji simetrično konstruiranih kubičnih C^2 zlepkov.



Slika 28: Konveksni ovojnici simetrično konstruiranih kubičnih C^2 zlepkov.

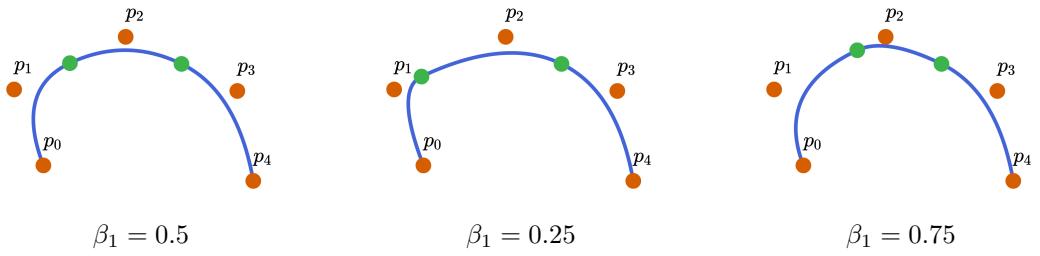
pravimo α *parametrizacije*. Točke u_i torej izberemo tako, da velja:

$$\Delta u_i = \|\Delta p_i\|^\alpha, \quad \alpha \in [0, 1], \quad i = 0, 1, \dots, m.$$

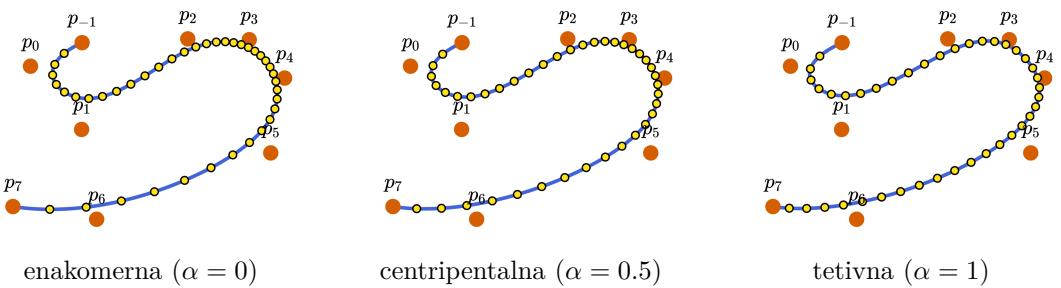
Če želimo upoštevati točki p_{-1} in p_{m+1} , ju lahko upoštevamo pri izračunu u_1 in u_m na naslednji način

$$u_1 = u_0 + \|p_1 - p_{-1}\|^\alpha, \quad u_m = u_{m-1} + \|p_{m+1} - p_{m-1}\|^\alpha, \quad \alpha \in [0, 1].$$

Vpliv izbire α si lahko ogledamo na sliki 30. Izbira vpliva tako na obliko, kot tudi na enakomernost porazdeljenosti točk na krivulji, izračunanih pri ekvidistantnih parametrih $\{t_i\}_{i=0}^N$.



Slika 29: Vpliv izbire β_1 pri simetričnem kvadratnem G^1 zlepku.



Slika 30: Vpliv izbire α pri izračunu točk u_i za simetrični kubični C^2 zlepek. Rumene točke predstavljajo točke na krivulji pri ekvidistantnih $\{t_i\}_{i=0}^N$.

5 PH krivulje

5.1 Konstantna parametrična hitrost

Definirajmo za začetek hodograf ter parametrično hitrost krivulje.

Definicija 5.1. Hodograf krivulje podane s parametrizacijo $\mathbf{r} = (x, y)$ je vektorsko polje $\mathbf{r}' = (x', y')$.

Definicija 5.2. Parametrična hitrost krivulje \mathbf{r} s parametrizacijo $\mathbf{r} = (x, y)$ je definirana s funkcijskim predpisom

$$\sigma(t) = \|\mathbf{r}'(t)\| = \sqrt{x'^2(t) + y'^2(t)}. \quad (5.1)$$

Idealno bi bilo, da je parametrična hitrost krivulje $\sigma \equiv 1$. Krivulje s konstantno parametrično hitrostjo $\sigma \equiv C$ lahko reparametriziramo s reparametrizacijsko funkcijo $\varphi(t) = \frac{t}{C}$, da dobimo krivuljo s parametrično hitrostjo 1. Za poljubno premico s parametrizacijo

$$\mathbf{r}(t) = \mathbf{p}_0 + t\mathbf{s}, \quad \mathbf{s} = (s_x, s_y),$$

lahko torej uporabimo reparametrizacijsko funkcijo s predpisom $\varphi(t) = \frac{t}{\sqrt{s_x^2 + s_y^2}}$. Polinomske krivulje višjih stopenj pa konstantne hitrosti ne morejo imeti. Naj bo vsaj ena izmed komponent parametrizacije $\mathbf{r} = (x, y)$ polinom stopnje $n > 1$. Ker sta vodilna koeficiente polinomov x'^2 in y'^2 pozitivna, iz izraza (5.1) hitro sledi, da polinom $\sigma^2 \in \mathbb{P}_{2n-2}$ ni konstanten.

5.2 Polinomska parametrična hitrost

V prejšnjem podrazdelku smo pokazali, da polinomske krivulje stopnje > 1 ne morejo imeti konstantne parametrične hitrosti. Če krivulje nekoliko omejimo, lahko dosežemo vsaj to, da bo funkcija hitrosti polinom. Iz enačbe (5.1) sledi, da bo to res v primerih, ko je polinom $x'^2 + y'^2 = z^2$.

Definicija 5.3. Polinomi a, b, c tvorijo pitagorejsko trojico, če zadoščajo enačbi

$$a^2 + b^2 = c^2.$$

Opomba 5.4. Vrstni red navajanja polinomov iz definicije 5.3 je pomemben. Če polinomi a, b, c tvorijo pitagorejsko trojico, jo tvorijo tudi polinomi b, a, c , polinomi a, c, b pa je ne.

Definicija 5.5. Naj bo $\mathbf{r} : \mathbb{R} \rightarrow \mathbb{R}^2$ parametrično podana krivulja s parametrizacijo $\mathbf{r} = (x, y)$. Krivulja \mathbf{r} je krivulja s pitagorejskim hodografom (PH krivulja), če obstaja tak polinom z , da polinomi x', y', z tvorijo pitagorejsko trojico.

Pri konstrukciji Bézierjevih PH krivulj si bomo pomagali z naslednjim izrekom.

Izrek 5.6. Polinomi a, b, c tvorijo pitagorejsko trojico natanko tedaj, ko obstajata tuja si polinoma u in v ter nek polinom w , da velja:

$$a = (u^2 - v^2)w, \quad b = 2uvw, \quad c = (u^2 + v^2)w.$$

Dokaz.

(\Leftarrow) Dokaz je enostaven, saj lahko definicije polinomov a, b in c iz izreka vstavimo v enačbo iz definicije 5.3 in jo preverimo.

(\Rightarrow) Najprej definirajmo polinom $w \equiv \gcd(a, b, c)$. Pri tako definiranem polinomu w so polinomi

$$\tilde{a} = \frac{a}{w}, \quad \tilde{b} = \frac{b}{w}, \quad \tilde{c} = \frac{c}{w},$$

tuji in zadoščajo enačbi $\tilde{a}^2 + \tilde{b}^2 = \tilde{c}^2$. Enačbo zapišimo nekoliko drugače

$$\tilde{b}^2 = \tilde{c}^2 - \tilde{a}^2 = (\tilde{c} + \tilde{a})(\tilde{c} - \tilde{a}).$$

Polinoma $\tilde{c} + \tilde{a}$ in $\tilde{c} - \tilde{a}$ ne moreta imeti skupnih ničel, saj bi to impliciralo skupne ničle polinomov $\tilde{a}, \tilde{b}, \tilde{c}$, kar bi bilo v nasprotju z njihovo definicijo. Ker skupnih ničel nimata, mora biti vsaka ničla polinoma \tilde{b} tudi ničla sode stopnje enega izmed polinomov $\tilde{c} + \tilde{a}$ ali $\tilde{c} - \tilde{a}$. Potem takem lahko definiramo tuja si polinoma u in v , da velja $\tilde{c} + \tilde{a} = 2u^2$ in $\tilde{c} - \tilde{a} = 2v^2$. Iz enačb potem hitro sledi

$$\tilde{a} = u^2 - v^2, \quad \tilde{b} = 2uv, \quad \tilde{c} = u^2 + v^2.$$

Enačbe pomnožimo s polinomom w in dobimo željeno obliko. \square

Iz konstrukcije dokaza lahko hitro vidimo, da sta za pitagorejske trojice katerih največji skupni delitelj je konstanta, konstantna tudi polinoma w in $\gcd(u, v)$. Ta-kšnim trojicam pravimo *primitivne pitagorejske trojice*. Parametrizacijo ravninske PH krivulje $\mathbf{r} = (x, y)$ lahko konstruiramo tako, da vstavimo tuja si polinoma u, v in polinom w v izraza

$$x' = (u^2 - v^2)w, \quad y' = 2uvw \tag{5.2}$$

in integriramo. Pri tem nekateri izbori polinomov u, v, w porodijo izrojene krivulje, ki bi jih radi izločili. Izrojene krivulje dobimo v naslednjih izborih.

1. Če je polinom $w \equiv 0$ ali polinoma $u = v \equiv 0$, sta polinoma $x' = y' \equiv 0$. Polinoma x in y sta zato konstantna in definirata točko, ne pa krivulje.

2. Če so polinomi u, v, w konstantni in neničelnii, potem sta konstantna tudi polinoma x' in y' . V tem primeru definirata parametrizacijo premice s konstantno hitrostjo.

3. Če sta polinoma u in v konstantna ter vsaj eden neničelen, polinom w pa ni konstanten, potem za polinoma x', y' velja $x' = Cy'$, $C \in \mathbb{R}$. Tudi tokrat polinoma definirata premico, a hitrost ni enakomerna.

4. Podobno kot v točki 3. dobimo tudi v primeru, ko je polinom w neničelen in je eden izmed polinomov u in v ničelen.

Neizrojene PH krivulje dobimo torej pri izborih neničelnih polinomov u, v in w , kjer vsaj eden izmed polinomov u in v ni konstanten. S številom λ označimo stopnjo polinoma w , s številom μ pa $\max(\deg(u), \deg(v))$. PH krivulja, pridobljena z integriranjem polinomov x' in y' iz (5.2), je stopnje $n = \lambda + 2\mu + 1$. Prostih parametrov pa je manj. Vsak izmed polinomov u in v je namreč definiran z največ $\mu + 1$ parametri. Brez izgube splošnosti lahko vodilni koeficient polinoma w fiksiramo na 1. Polinom w je zato definiran z λ parametri. Integracijska konstanta pri integriranju hodografa nam poda še 2 prosta parametra. Skupno je prostih parametrov torej $\lambda + 2(\mu + 1) + 2 = \lambda + 2\mu + 4 = n + 3$.

5.3 Kontrolne točke Bézierjevih PH krivulj

V tem podrazdelku bomo konstruirali *Bézierjeve PH krivulje*, to so Bézierjeve krivulje, ki so tudi PH krivulje. Krivulje bomo konstruirali s pomočjo izreka (5.2), kjer za polinomom w vzamemo $w \equiv 1$, da bodo krivulje zagotovo regularne. Polinoma x', y' sta torej definirana kot

$$x' = u^2 - v^2, \quad y' = 2uv. \quad (5.3)$$

Za parametrično hitrost krivulje s parametrizacijo $\mathbf{r} = (x, y)$ potem velja

$$\|\mathbf{r}'\| = \sqrt{x'^2 + y'^2} = u^2 + v^2.$$

Od tod pa sledi $\|\mathbf{r}'(t)\| \neq 0$, $t \in \mathbb{R}$, saj polinoma u in v nimata skupnih ničel. Krivulje so torej regularne in lihe stopnje $n = 2\mu + 1$. Najbolj osnovne netrivialne PH krivulje dobimo tako, da za polinoma u in v izberemo Bernsteinova polinoma

$$u = u_0 B_0^1 + u_1 B_1^1, \quad v = v_0 B_0^1 + v_1 B_1^1. \quad (5.4)$$

Zanju mora veljati $u_0 v_1 - u_1 v_0 \neq 0$, da sta si polinoma u in v tuja, in $(u_1 - u_0)^2 + (v_1 - v_0)^2 \neq 0$, da vsaj eden od njiju ni konstanten. Ko polinoma vstavimo v izraza (5.3) dobimo

$$\begin{aligned} x' &= (u_0^2 - v_0^2)B_0^2 + (u_0 u_1 - v_0 v_1)B_1^2 + (u_1^2 - v_1^2)B_2^2, \\ y' &= 2u_0 v_0 B_0^2 + (u_0 v_1 + u_1 v_0)B_1^2 + 2u_1 v_1 B_2^2. \end{aligned}$$

Polinoma x', y' integriramo s pomočjo izreka 2.7 in upoštevamo, da Bernsteinovi polinomi tvorijo razčlenitev enote. Dobimo

$$\begin{aligned} x &= x_0(B_0^3 + B_1^3 + B_2^3 + B_3^3) \\ &\quad + \frac{1}{3}(u_0^2 - v_0^2)(B_1^3 + B_2^3 + B_3^3) \\ &\quad + \frac{1}{3}(u_0 u_1 - v_0 v_1)(B_2^3 + B_3^3) \\ &\quad + \frac{1}{3}(u_1^2 - v_1^2)B_3^3, \end{aligned} \quad \begin{aligned} y &= y_0(B_0^3 + B_1^3 + B_2^3 + B_3^3) \\ &\quad + \frac{1}{3}2u_0 v_0(B_1^3 + B_2^3 + B_3^3) \\ &\quad + \frac{1}{3}(u_0 v_1 + u_1 v_0)(B_2^3 + B_3^3) \\ &\quad + \frac{1}{3}2u_1 v_1 B_3^3, \end{aligned}$$

kar ustrezza Bézierjevi krivulji s kontrolnimi točkami

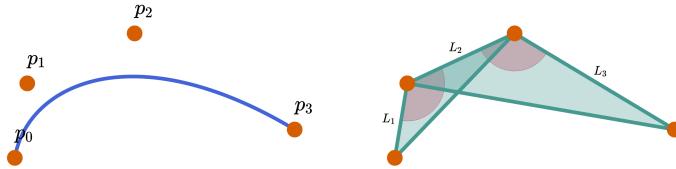
$$\begin{aligned} \mathbf{p}_1 &= \mathbf{p}_0 + \frac{1}{3}(u_0^2 - v_0^2, 2u_0 v_0), \\ \mathbf{p}_2 &= \mathbf{p}_1 + \frac{1}{3}(u_0 u_1 - v_0 v_1, u_0 v_1 + u_1 v_0), \\ \mathbf{p}_3 &= \mathbf{p}_2 + \frac{1}{3}(u_1^2 - v_1^2, 2u_1 v_1). \end{aligned} \quad (5.5)$$

Kjer smo kontrolno točko $\mathbf{p}_0 = (x_0, y_0)$ izbrali poljubno, saj sta x_0 in y_0 integracijski konstanti. Kontrolni poligoni dobljeni z izrazi (5.5) imajo tudi geometrijsko interpretacijo. Brez dokaza podajmo izrek, ki jo definira.

Izrek 5.7. Naj bo \mathbf{B} kubična Bézierjeva krivulja s kontrolnimi točkami \mathbf{p}_i , $i = 0, 1, 2, 3$. Z $L_i = \|\Delta p_i\|$, $i = 0, 1, 2$, označimo dolžine stranic kontrolnega poligona, s θ_1 in θ_2 pa kota kontrolnega poligona pri točkah \mathbf{p}_1 in \mathbf{p}_2 . Krivulja \mathbf{B} je PH krivulja natanko tedaj, ko je

$$L_1 = \sqrt{L_2 L_0} \quad \text{in} \quad \theta_1 = \theta_2.$$

Pogoji iz izreka 5.7 so ekvivalentni temu, da morata biti trikotnika $\triangle \mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ in $\triangle \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ podobna, kar lahko vidimo na sliki 31.



Slika 31: Geometrija kontrolnega poligona kubične Bézierjeve PH krivulje.

V prejšnjem razdelku smo povedali, da ima PH krivulja n -te stopnje $n+3$ prostih parametrov. Za pravkar definirane kubične Bézierjeve PH krivulje to pomeni, da imajo le 6 prostih parametrov. Če želimo podobno raven kontrole, kakor pri navadnih kubičnih Bézierjevih krivuljah, moramo poseči po Bézierjevih PH krivuljah stopnje 5. Izpeljemo jih podobno, le da tokrat za polinoma u in v izberemo Bernsteinova polinoma polinoma stopnje 2

$$u = u_0 B_0^2 + u_1 B_1^2 + u_2 B_2^2, \quad v = v_0 B_0^2 + v_1 B_1^2 + v_2 B_2^2. \quad (5.6)$$

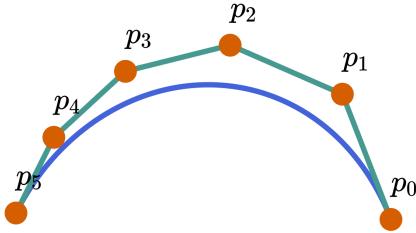
Da sta si polinoma u in v tuja, mora tokrat veljati

$$(u_2 v_0 - u_0 v_2)^2 \neq 4(u_0 v_1 - u_1 v_0)(u_1 v_2 - u_2 v_1).$$

Polinoma vstavimo v izraza (5.3) in integriramo ter z nekaj računanja dobimo Bézierjevo krivuljo s kontrolnimi točkami

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{p}_0 + \frac{1}{5}(u_0^2 - v_0^2, 2u_0 v_0), \\ \mathbf{p}_2 &= \mathbf{p}_1 + \frac{1}{5}(u_0 u_1 - v_0 v_1, u_0 v_1 + u_1 v_0), \\ \mathbf{p}_3 &= \mathbf{p}_2 + \frac{2}{5}(u_1^2 - v_1^2, 2u_1 v_1) + \frac{1}{5}(u_0 u_2 - v_0 v_2, u_0 v_2 + u_2 v_0), \\ \mathbf{p}_4 &= \mathbf{p}_3 + \frac{1}{5}(u_1 u_2 - v_1 v_2, u_1 v_2 + u_2 v_1), \\ \mathbf{p}_5 &= \mathbf{p}_4 + \frac{1}{5}(u_2^2 - v_2^2, 2u_2 v_2). \end{aligned} \quad (5.7)$$

Primer takšne krivulje si lahko ogledamo na sliki 32.



Slika 32: Bézierjeva PH krivulja stopnje 5.

5.4 Parametrična hitrost Bézierjeve PH krivulje

Parametrična hitrost Bézierjeve PH krivulje, podane s parametrizacijo $\mathbf{r} = (x, y)$, je dana s polinomom

$$\sigma = \|\mathbf{r}'\| = \sqrt{x'^2 + y'^2} = u^2 + v^2.$$

Če je krivulja stopnje n , potem sta polinoma u in v stopnje $m = \frac{1}{2}(n-1)$. Zapišimo polinome u, v in σ v Bernsteinovi bazi

$$u = \sum_{j=0}^m u_j B_j^m, \quad v = \sum_{j=0}^m v_j B_j^m, \quad \sigma = \sum_{j=0}^{n-1} \sigma_j B_j^{n-1}.$$

S pomočjo pravila za množenje Bernsteinovih polinomov iz izreka 2.8, lahko koeficiente σ_j izrazimo s koeficienti polinomov u in v kot

$$\sigma_j = \sum_{k=\max(0, j-m)}^{\min(m, j)} \frac{\binom{m}{k} \binom{m}{j-k}}{\binom{n-1}{j}} (u_k u_{j-k} + v_k v_{j-k}).$$

Tako smo parametrično hitrost Bézierjeve PH krivulje izrazili v Bernsteinovi bazi. Parametrično hitrost lahko uporabimo za izračun dolžine krivulje. Dolžina odseka krivulje pri parametru $t \in [0, t_0]$, $t_0 \in [0, 1]$ je podana z določenim integralom

$$s(t_0) = \int_0^{t_0} \sigma(t) dt = \int_0^{t_0} \sum_{j=0}^{n-1} \sigma_j B_j^{n-1}(t) dt.$$

Integral izračunamo s pomočjo integracijskega pravila iz izreka 2.7. Dobimo dolžino odseka Bézierjeve PH krivulje pri parametru $t \in [0, t_0]$ izraženo v Bernsteinovi bazi

$$s(t_0) = \sum_{k=0}^n s_k B_k^n(t_0), \quad s_0 = 0, \quad s_k = \frac{1}{n} \sum_{j=0}^{k-1} \sigma_j.$$

Iz zapisa hitro sledi, da je celotna dolžina Bézierjeve PH krivulje enaka

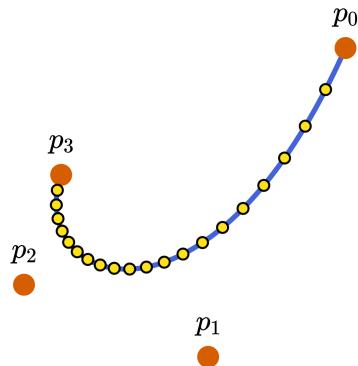
$$s(1) = \frac{\sigma_0 + \sigma_1 + \cdots + \sigma_{n-1}}{n}.$$

Dolžino poljubnega odseka krivulje na intervalu $[a, b]$, $0 \leq a < b \leq 1$, pa lahko izračunamo s pomočjo izraza

$$\int_a^b \sigma(t) dt = \int_0^b \sigma(t) dt - \int_0^a \sigma(t) dt = s(b) - s(a).$$

5.5 Enakomerna parametrizacija

V razdelku 5.1 smo pokazali, da polinomske krivulje stopnje višje od 1 ne morejo imeti konstantne parametrične hitrosti. V tem razdelku bomo reševali problem, ki je s konstantno parametrično hitrostjo tesno povezan. Želimo najti takšne parametre $\{t_i\}_{i=0}^N$, $N \in \mathbb{N}$, da bodo odseki krivulje med njimi enako dolgi. Za krivulje s konstantno hitrostjo je problem enostavno rešiti, saj pri ekvidistantno izbranih parametrih $\{t_i\}_{i=0}^N$, $\Delta t_i = \frac{1}{N+1}$, zanje velja $s(t_{i+1}) - s(t_i) = \frac{C}{N+1}$. Bézierjeve PH krivulje pa konstantne hitrosti nimajo, zato ekvidistantna izbira parametrov $\{t_i\}_{i=0}^N$ problema ne reši. Slednje lahko vidimo na sliki 33. Čeprav parametrična hitrost



Slika 33: Dvajset točk na Bézierjevi PH krivulji pri ekvidistantnih t_i .

Bézierjeve PH krivulje ni konstantna, lahko njen enostaven izračun izkoristimo, da najdemo parametre $\{t_i\}_{i=0}^N$ za katere velja $s(t_k) = k\Delta s$, kjer je $\Delta s = \frac{s(1)}{N}$. Ker je parametrična hitrost σ pozitivna za vsa realna števila $t \in [0, 1]$, je funkcija dolžine s strogo naraščajoča. Parametri t_i so zato enolično določeni in ležijo med t_{i-1} in 1. Za iskanje parametrov t_i lahko uporabimo Newton-Raphsonovo iteracijo. Za začetni približek vzamemo $t_k^{(0)} = t_{k-1} + \frac{\Delta s}{\sigma(t_{k-1})}$ popravljamo pa ga z iteriranjem

$$t_k^{(r)} = t_k^{(r-1)} - \frac{s(t_k^{(r-1)})}{\sigma(t_k^{(r-1)})}, \quad r = 1, 2, \dots$$

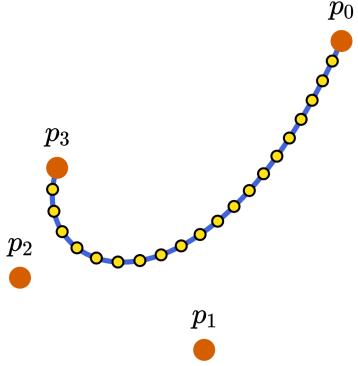
Znano je, da takšna iteracija pri začetnih približkih, ki so dovolj blizu parametru t_k , konvergira kvadratično. Za večino primerov zato izračun približka t_k , do natančnosti reda 10^{-16} , potrebuje le tri do štiri korake iteracije. V praksi so takšni približki ponavadi zadovoljivi. Točke na PH krivulji, pridobljene s takšnimi približki, si lahko ogledamo na sliki 34.

5.6 Tangenta, normala in ukrivljenost

V razdelku 5.4 smo pokazali, da je parametrična hitrost Bézierjeve PH krivulje polinom. Od tod sledi, da so tangenta, normala in ukrivljenost PH krivulje racionalne funkcije.

Izrek 5.8. *Tangenta, normala in ukrivljenost Bézierjeve PH krivulje, dobljene z integriranjem izraza (5.3), so enake*

$$\mathbf{t} = \frac{(u^2 - v^2, 2uv)}{\sigma}, \quad \mathbf{n} = \frac{(2uv, v^2 - u^2)}{\sigma}, \quad \kappa = 2 \frac{uv' - u'v}{\sigma^2},$$



Slika 34: Dvajset točk na Bézierjevi PH krivulji, pridobljenih s parametri, za katere velja $s(t_{i+1}) - s(t_i) = \frac{s(1)}{N}$.

kjer je $\sigma = u^2 + v^2$ parametrična hitrost krivulje.

Dokaz. Dokaza za tangento in normalo sta enostavna, saj izraza (5.3) le vstavimo v izraza iz izreka in upoštevamo, da je polinom $\sigma = u^2 + v^2$. Dokažimo še izraz za ukrivljenost κ . V enačbo za ukrivljenost $\kappa = \frac{x'y'' - y'x''}{(\sigma^2)^{3/2}}$ vstavimo polinoma (5.3), ter njuna druga odvoda $x'' = 2(uu' - vv')$ in $y'' = 2(u'v + uv')$, dobimo

$$\begin{aligned} \kappa &= 2 \frac{(u^2 - v^2)(u'v + uv') - 2uv(uu' - vv')}{(\sigma^2)^{3/2}} \\ &= 2 \frac{u^2vv' + u^3v' - v^3u' - v^2uv' - 2u^2vu' + 2uv^2v'}{\sigma^3} \\ &= 2 \frac{(u^2 - v^2 + 2v^2)uv' - (-u^2 + v^2 + 2u^2)u'v}{\sigma^3} \\ &= 2 \frac{(u^2 + v^2)uv' - (u^2 + v^2)u'v}{\sigma^3} \\ &= 2 \frac{(u^2 + v^2)(uv' - u'v)}{\sigma^3} = 2 \frac{uv' - u'v}{\sigma^2}. \end{aligned}$$

□

5.7 Racionalni odmiki krivulje

Odmik krivulje na razdalji d od \mathbf{r} je krivulja, podana s parametrizacijo

$$\mathbf{r}_d = \mathbf{r} + d\mathbf{n}.$$

Takšne krivulje v splošnem niso racionalne, saj v imenovalcu funkcije za smer enotske normale \mathbf{n} nastopa koren. V prejšnjem razdelku smo pokazali, da je smer normale \mathbf{n} za PH krivulje racionalna funkcija, iz česar sledi, da je tudi odmik krivulja PH krivulje racionalna funkcija. Še več, odmik Bézierjeve PH krivulje \mathbf{r} lahko predstavimo kot racionalno Bézierjevo krivuljo s kontrolnimi točkami izraženimi s kontrolnimi točkami krivulje \mathbf{r} . Zapišimo kontrolne točke Bézierjeve PH krivulje \mathbf{r} v homogenih koordinatah

$$\mathbf{P}_k = (W_k, X_k, Y_k) = (1, x_k, y_k), \quad k = 0, 1, \dots, n.$$

Preme diference takšnih točk so

$$\Delta \mathbf{P}_k = \mathbf{P}_{k+1} - \mathbf{P}_k = (0, \Delta x_k, \Delta y_k), \quad k = 0, 1, \dots, n-1.$$

Označimo še pravokotno smer $\Delta \mathbf{P}_k^\perp = (0, \Delta y_k, -\Delta x_k)$. Parametrizacijo odmika krivulje lahko izrazimo kot

$$\mathbf{r}_d = \left(\frac{\widetilde{X}}{\widetilde{W}}, \frac{\widetilde{Y}}{\widetilde{W}} \right),$$

kjer so polinomi \widetilde{W} , \widetilde{X} in \widetilde{Y} stopnje $2n-1$, njihovi koeficienti v Bernsteinovi bazi pa definirajo kontrolne točke racionalne Bézierjeve krivulje s homogenimi koordinatami

$$\mathbf{O}_k = (\widetilde{W}_k, \widetilde{X}_k, \widetilde{Y}_k), \quad k = 0, \dots, 2n-1.$$

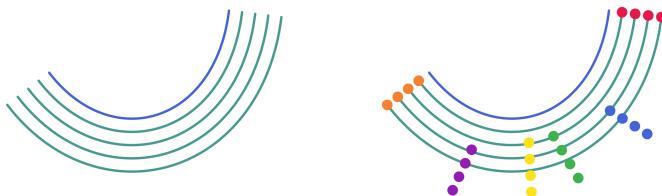
Homogene koordinate kontrolnih točk odmika krivulje lahko v zaključeni obliki izrazimo s kontrolnimi točkami začetne Bézierjeve PH krivulje kot [5]:

$$\mathbf{O}_k = \sum_{j=\max(0, k-n)}^{\min(n-1, k)} \frac{\binom{n-1}{j} \binom{n}{k-j}}{\binom{2n-1}{k}} (\sigma_j \mathbf{P}_{k-j} + dn \Delta \mathbf{P}_j^\perp), \quad k = 0, \dots, 2n-1.$$

Za primer si oglejmo kontrolne točke odmik krivulje kubične PH krivulje

$$\begin{aligned} \mathbf{O}_0 &= \sigma_0 \mathbf{P}_0 + 3d \Delta \mathbf{P}_0^\perp, \\ \mathbf{O}_1 &= \frac{1}{5} [2\sigma_1 \mathbf{P}_0 + 3\sigma_0 \mathbf{P}_1 + 3d(3\Delta \mathbf{P}_0^\perp + 2\Delta \mathbf{P}_1^\perp)], \\ \mathbf{O}_2 &= \frac{1}{10} [\sigma_2 \mathbf{P}_0 + 6\sigma_1 \mathbf{P}_1 + 3\sigma_0 \mathbf{P}_2 + 3d(3\Delta \mathbf{P}_0^\perp + 6\Delta \mathbf{P}_1^\perp + \Delta \mathbf{P}_2^\perp)], \\ \mathbf{O}_3 &= \frac{1}{10} [3\sigma_2 \mathbf{P}_1 + 6\sigma_1 \mathbf{P}_2 + 3\sigma_0 \mathbf{P}_3 + 3d(\Delta \mathbf{P}_0^\perp + 6\Delta \mathbf{P}_1^\perp + 3\Delta \mathbf{P}_2^\perp)], \\ \mathbf{O}_4 &= \frac{1}{5} [3\sigma_2 \mathbf{P}_2 + 2\sigma_1 \mathbf{P}_3 + 3d(2\Delta \mathbf{P}_1^\perp + 3\Delta \mathbf{P}_2^\perp)], \\ \mathbf{O}_5 &= \sigma_2 \mathbf{P}_3 + 3d \Delta \mathbf{P}_2^\perp. \end{aligned}$$

Na sliki 35 lahko vidimo, da se kontrolne točke odmika krivulje v odvisnosti od razdalje premikajo po premicah.



Slika 35: Odmiki krivulje, levo, skupaj z njihovimi kontrolnimi točkami, desno.

5.8 Reprezentacija s kompleksnimi števili

Bézierjeve PH krivulje stopnje 3 in 5 smo v prejšnjih podrazdelkih izpeljali tako, da smo izbrali Bernsteinove polinome definirane z izrazi (5.4),(5.6) in jih integrirali. Če predstavimo koeficiente polinomov s kompleksnimi števili $\mathbf{w}_k = u_k + iv_k \in \mathbb{C}$ in enako storimo s kontrolno točko $\mathbf{p}_0 = x_0 + iy_0$, lahko preostale kontrolne točke kubične Bézierjeve PH krivulje izrazimo kot kompleksna števila z naslednjimi izrazi

$$\mathbf{p}_1 = \mathbf{p}_0 + \frac{1}{3}\mathbf{w}_0^2, \quad \mathbf{p}_2 = \mathbf{p}_1 + \frac{1}{3}\mathbf{w}_0\mathbf{w}_1, \quad \mathbf{p}_3 = \mathbf{p}_2 + \frac{1}{3}\mathbf{w}_1^2.$$

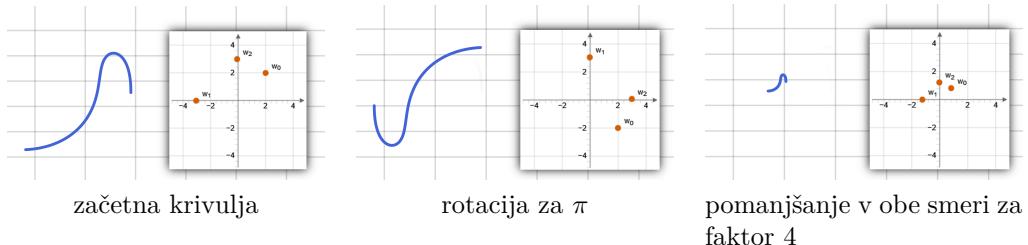
Podobno za Bézierjevo PH krivuljo stopnje 5 dobimo

$$\begin{aligned} \mathbf{p}_1 &= \mathbf{p}_0 + \frac{1}{5}\mathbf{w}_0^2, & \mathbf{p}_2 &= \mathbf{p}_1 + \frac{1}{5}\mathbf{w}_0\mathbf{w}_1, & \mathbf{p}_3 &= \mathbf{p}_2 + \frac{1}{5}\frac{2\mathbf{w}_1^2 + \mathbf{w}_0\mathbf{w}_2}{3}, \\ \mathbf{p}_4 &= \mathbf{p}_3 + \frac{1}{5}\mathbf{w}_1\mathbf{w}_2, & \mathbf{p}_5 &= \mathbf{p}_4 + \frac{1}{5}\mathbf{w}_2^2. \end{aligned}$$

Da so izrazi ekvivalentni izrazom (5.5) oziroma izrazom (5.7), je enostavno preveriti z nekaj računanja. Reprezentacija s kompleksnimi števili omogoča enostavno rotiranje in skaliranje krivulje. Zapišimo število \mathbf{w}_k v polarnem zapisu, tj. $\mathbf{w}_k = r_k e^{i\varphi_k}$. Potem lahko rotacijo števila \mathbf{w}_k okoli središča koordinatnega sistema za kot θ predstavimo z množenjem števila s kompleksnim številom $\mathbf{z} = e^{i\theta}$, tj. $\mathbf{z}\mathbf{w}_k = r_k e^{i(\varphi_k + \theta)}$. Kontrolne točke pri rotiranih točkah $\tilde{\mathbf{w}}_i = \mathbf{z}\mathbf{w}_k$ potem izgledajo takole

$$\tilde{\mathbf{p}}_1 = \mathbf{p}_0 + \frac{1}{3}\mathbf{z}^2\mathbf{w}_0^2, \quad \tilde{\mathbf{p}}_2 = \tilde{\mathbf{p}}_1 + \frac{1}{3}\mathbf{z}^2\mathbf{w}_0\mathbf{w}_1, \quad \tilde{\mathbf{p}}_3 = \tilde{\mathbf{p}}_2 + \frac{1}{3}\mathbf{z}^2\mathbf{w}_1^2.$$

Če fiksiramo točko $\mathbf{p}_0 = (0, 0)$, iz izrazov hitro sledi $\tilde{\mathbf{p}}_i = \mathbf{z}^2\mathbf{p}_i$. Od tod pa sledi, da rotiranje števil \mathbf{w}_k okoli središča za kot θ rotira krivuljo okoli točke \mathbf{p}_0 za kot 2θ . Podobno dobimo, če za število \mathbf{z} vzamemo skalar. V tem primeru skaliranje števil \mathbf{w}_k za skalar d , skalira krivuljo za d^2 . Primera rotacije in skaliranja krivulje s pripadajočimi števili \mathbf{w}_k si lahko ogledamo na sliki 36. Več o kompleksni reprezentaciji Bézierjevih PH krivulj si bralec lahko prebere v delu [6].



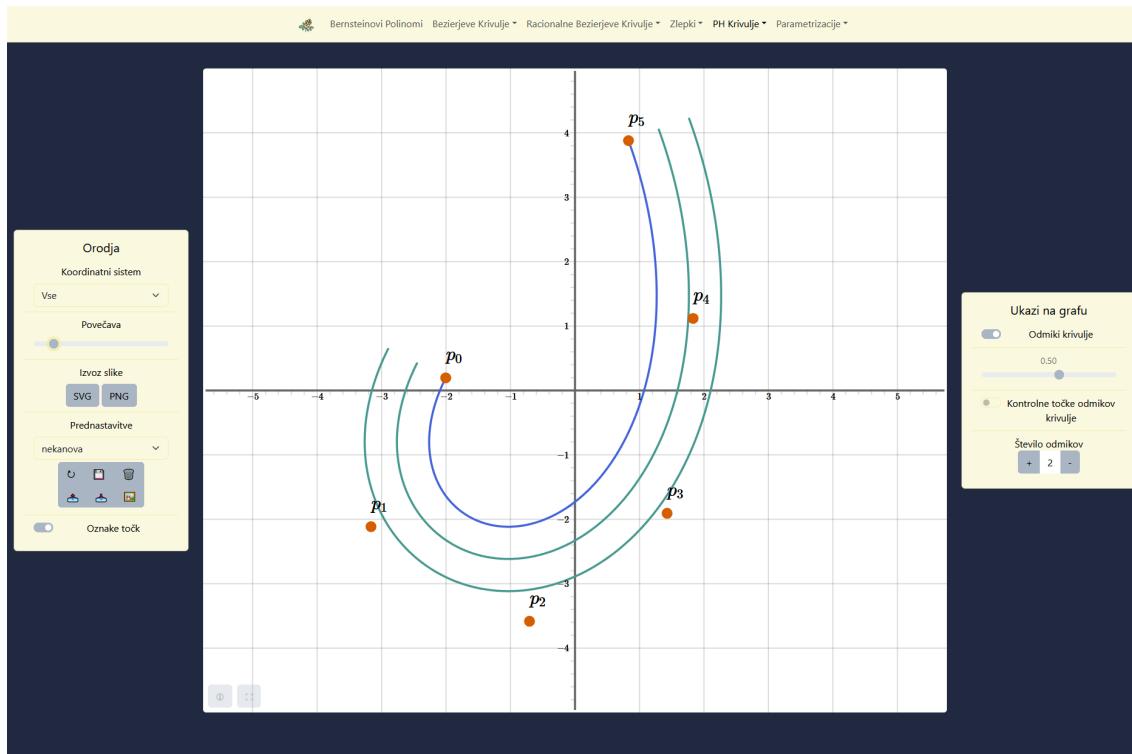
Slika 36: Rotacija in skaliranje Bezierjeve PH krivulje s pripadajočimi števili \mathbf{w}_k .

6 Orodje za grafični prikaz lastnosti Bézierjevih in PH krivulj - *Bezeg*

Vse krivulje, ki so bile v magistrskem delu predstavljene, so tudi implementirane v spletnem orodju Bezeg, ki je bilo izdelano v okviru dela. Orodje v celoti živi v uporabnikovem brskalniku in zato zalednih sistemov ne potrebuje. Objavljeno je preko storitve GitHub Pages [1], ki omogoča zastonjsko objavo statične spletne strani. Dostopno pa je na spletni strani [2]. Spletno orodje pod krovom za izdelavo uporabniških vmesnikov uporablja knjižnico React [10], za risanje interaktivnih grafov knjižnico JSXGraph (okrajš. JXG) [9], za oblikovanje spletne strani pa uporablja ogrodje Bootstrap [3] v paru s knjižnico react-bootstrap [11]. Napisano pa je v jeziku TypeScript [12]. Vse izbrane tehnologije so odprto kodne in namenjene prosti rabi. Podrobnejše jih ne bomo opisovali, več o njih si lahko bralec prebere na pripadajočih spletnih straneh.

6.1 Uporabniški vmesnik

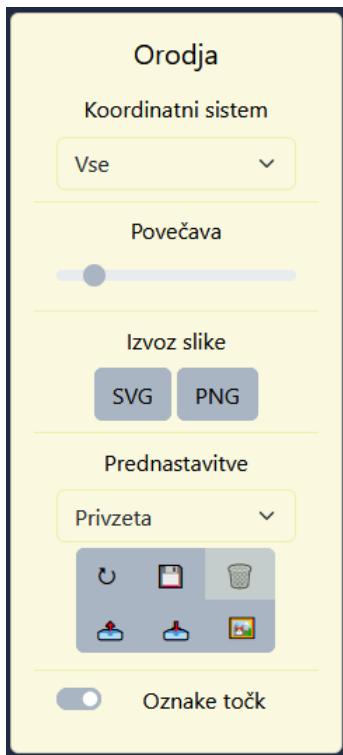
Osnovni izgled uporabniškega vmesnika za izbrani graf na spletni strani Bezga si lahko ogledamo na sliki 37. Na vrhu se nahaja navigacijska vrstica s povezavami do različnih grafov. Na sredini se nahaja instanca grafa knjižnice JXG. Levo od grafa se nahaja kartica s splošnimi orodji za vse grafe, desno od njega pa se nahaja kartica z grafu specifičnimi ukazi. V naslednjih podrazdelkih bomo podrobnejše predstavili funkcionalnosti Bezga.



Slika 37: Osnovni izgled uporabniškega vmesnika.

6.1.1 Orodja

Kartico z orodji si oglejmo na sliki 38 in predstavimo čemu posamezno orodje služi.



Slika 38: Kartica z orodji.

- *Koordinatni sistem*

Orodje za koordinatni sistem premore le izbirnik, s katerim uporabnik izbere kateri elementi koordinatnega sistema bodo prikazani na grafu.

- *Povečava*

Orodje za povečavo uporabniku nudi drsnik, s katerim lahko poveča velikost elementov na grafu.

- *Izvoz slike*

Uporabnik lahko s pritiskom na gumb SVG oziroma PNG izvozi sliko v izbranem formatu. Pri izvozu v format SVG, zaradi pomanjkljivosti knjižnice JXG, izgubimo oznake točk.

- *Prednastavitev*

Orodje za prednastavitev uporabniku omogoča shranjevanje in izbiranje prikazov. Posamezno prednastavitev lahko uporabnik izbere z izbirnikom. Ko uporabnik pripravi željeni prikaz, ga lahko s poljubnim imenom shrani kot prednastavitev. Prednastavitev se shranjujejo lokalno v uporabnikovem brskalniku. Če uporabnik želi svoje prednastavitev prenesti na drug računalnik, lahko to storí tako, da prednastavitev izvozi in jih na drugem računalniku

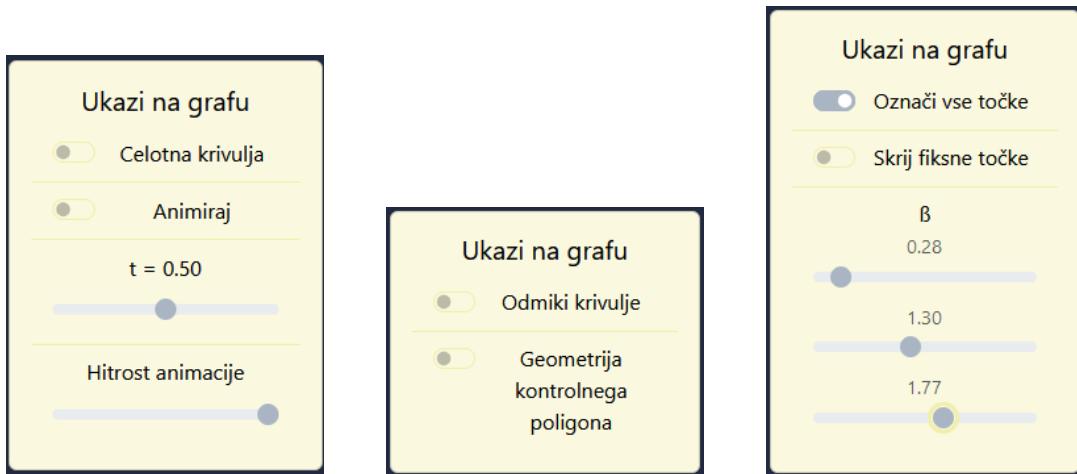
uvazi. Prednastavitev se izvozijo kot JSON datoteka. Uporabnik ima tudi možnost generirati slike svojih prednastavitev. Ob zagonu generacije slik orodje začne preklapljati med prednastavtvami ter ustvarjati slike. Slike nato v PNG formatu pod imeni prednastavitev zapakira v zip datoteko, ki jo uporabnik prenese.

- *Oznake točk*

Preklopnik za prikazovanje oznak točk na grafih z Bézierjevimi krivuljami.

6.1.2 Ukazi na grafu

Kartica z ukazi na grafu je namenjena ukazom posameznega grafa. Za primere si oglejmo kartice na sliki 39.

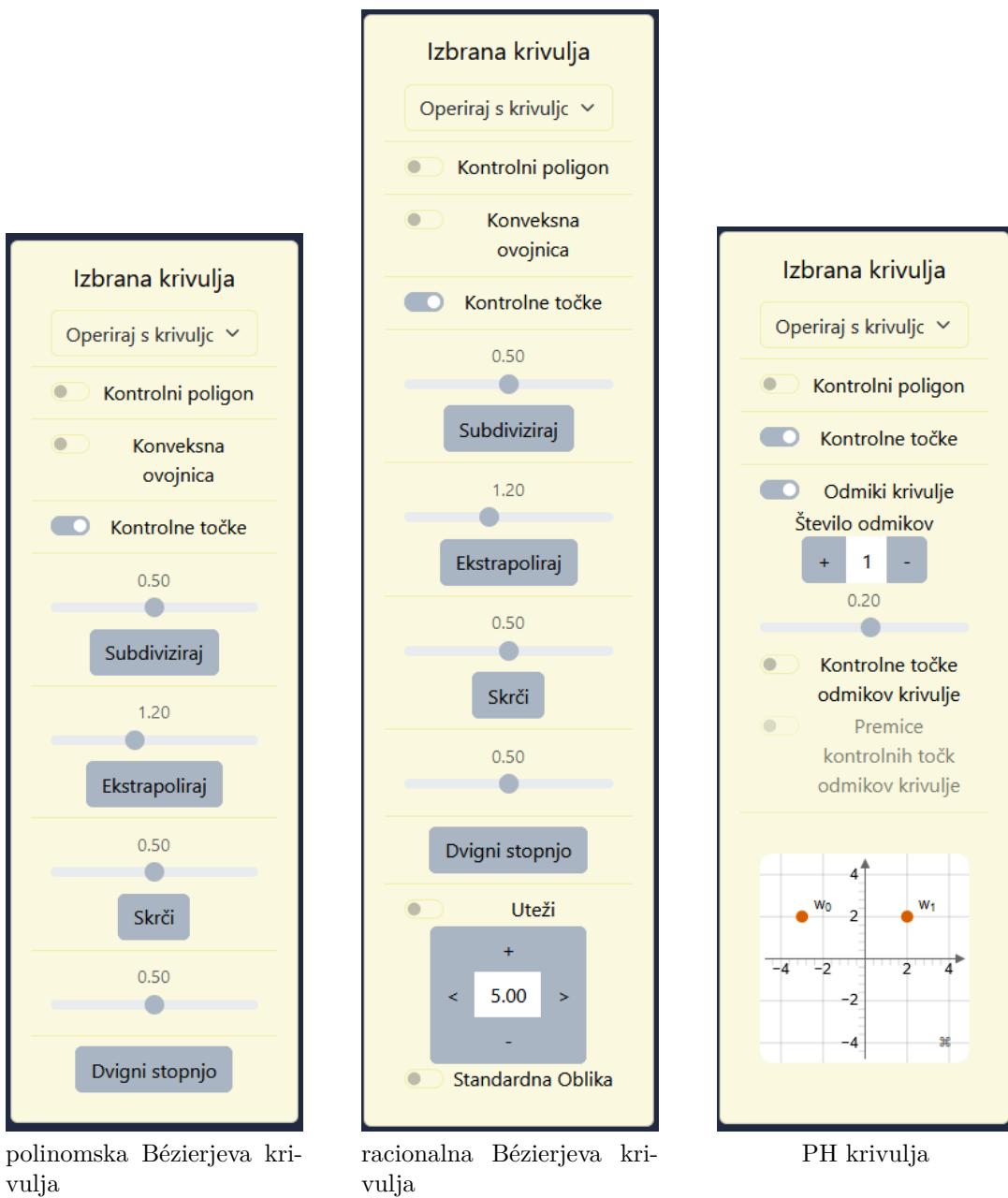


Slika 39: Kartice ukazov na grafu za različne grafe.

6.1.3 Ukazi in transformacije izbrane krivulje

Uporabnik lahko krivuljo izbere tako, da nanjo klikne. Ko je krivulja izbrana, se okoli nje nariše črtkan pravokotnik, kartico z ukazi na grafu pa zamenja kartica z ukazi izbrane krivulje. Primere takšnih kartic si lahko ogledamo na sliki 40. Naštejmo še, kakšne ukaze premorejo različne krivulje.

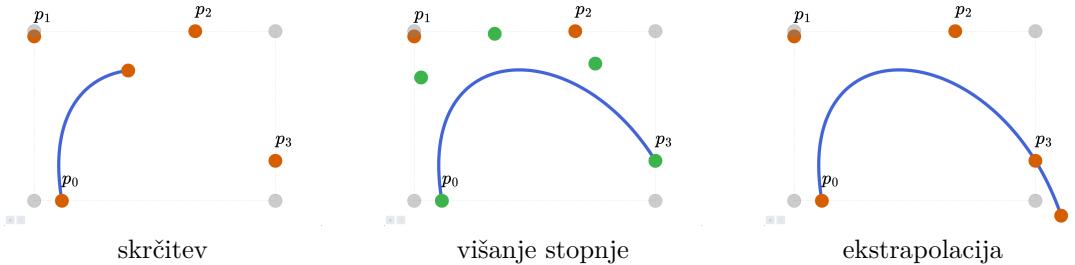
- Vse krivulje omogočajo preklapljanje prikazovanja kontrolnega poligona, konveksne ovojnice kontrolnih točk ter kontrolnih točk samih.
- Bézierjeva krivulja premore poleg osnovih ukazov še ukaze za subdivizijo, ekstrapolacijo, skrčitev, višanje stopnje ter prikazovanje De Casteljaujeve sheme. Ob premiku miške na posamezni ukaz se na grafu predstavi nekakšen predogled ukaza. Primer si lahko ogledamo na sliki 41.
- Racionalna Bézierjeva krivulja ima poleg funkcionalnosti Bézierjeve krivulje dodane še ukaze za prikaz in sprememjanje uteži, ter preklopnik za prikaz standardne oblike uteži.



Slika 40: Kartice z ukazi za različne izbrane krivulje.

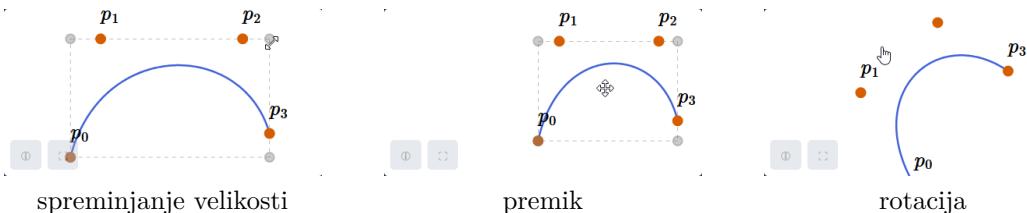
- PH Bézierjeve krivulje nimajo ukazov, ki so jih imele Bézierjeve krivulje, saj se pri nekaterih ukazih izgubi PH lastnost. Premorejo pa naslednje ukaze: prikazovanje odmikov krivulje in spremenjanje njihovih razdalij do krivulje, prikaz kontrolnih točk odmikov krivulje ter prikazovanje premic na katerih ležijo, dodajanje in odstranjevanje odmika krivulje, in koordinatni sistem z interaktivnimi točkami, ki predstavljajo točke w_k iz razdelka 5.8.
- C^n zlepki premorejo le ukaze osnovne krivulje, G^1 zlepki pa imajo poleg teh še drsnike za spremenjanje parametra β_1 .

Uporabnik lahko izbrano krivuljo transformira tako, da upravlja s črtkanim pravokotnikom.



Slika 41: Predogledi rezultatov različnih ukazov za krivuljo.

tnikom. Klik na oglišče pravokotnika zažene spremenjanje velikosti krivulje, klik ob stranici pravokotnika zažene rotiranje krivulje, klik znotraj pravokotnika pa zažene premikanje krivulje. Primere si lahko ogledamo na sliki 42.



Slika 42: Transformacije izbrane krivulje.

6.2 Implementacija

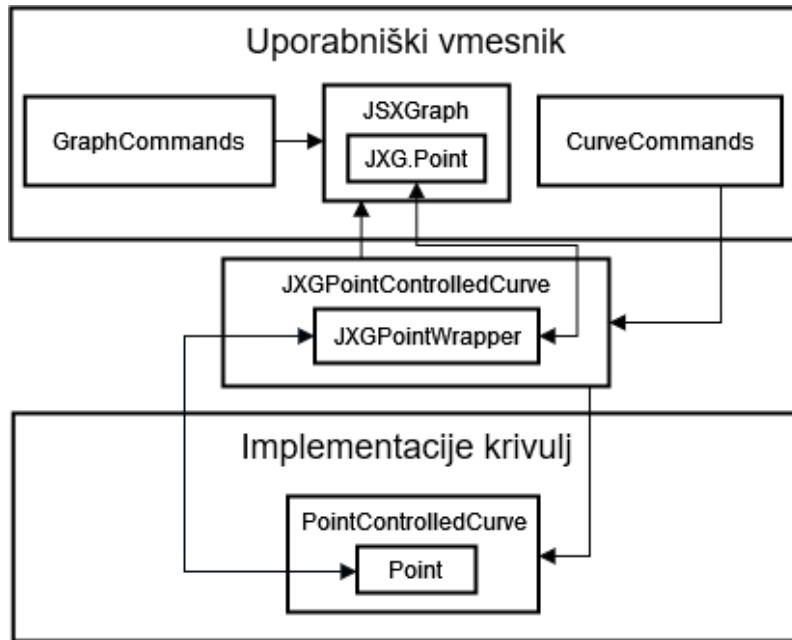
Arhitekturo aplikacije si lahko ogledamo na sliki 43. Uporabniški vmesnik upravlja s krivuljami preko razreda JXGPointControlledCurve. Razred JXGPointControlledCurve točke iz knjižnice JXG ovije v ovoj JXGPointWrapper, ki nudi podporo za rabo točk v implementiranih krivuljah Bezga. Ovoj JXGPointWrapper je dvosmeren - spremenjanje točke v implementaciji krivulj spremeni točko iz knjižnice JXG in obratno. Tako dosežemo, da uporabnikovo premikanje točk iz knjižnice JXG spreminja obliko krivulje, razne transformacije krivulje pa transformirajo tudi točke iz knjižnice JXG. V naslednjih razdelkih bomo podrobneje predstavili posamezne člene Bezga.

Opomba 6.1. Implementacija krivulj v Bezgu je napisana v programskem jeziku TypeScript. Za voljo kompaknosti, bodo primeri kode v magistrskem delu pisani v mešanici psevdokode in TypeScripta, vmesniki in razredi v primerih, pa bodo poenostavljeni do njihovih najpomembnejših funkcionalnosti. Dejansko implementacijo si lahko bralec ogleda na repozitoriju Bezga [8].

6.2.1 Vmesniki

Vse krivulje, predstavljene v delu, temeljijo na kontrolnih točkah. Točke v Bezgu predstavljamo z vmesnikom

```
1 interface Point {
```



Slika 43: Arhitektura spletnega orodja Bezug.

```

2 X(): number;
3 Y(): number;
4 setX(x: number | (() => number)): void;
5 setY(y: number | (() => number)): void;
6 .

```

Polja X in Y, ki predstavljata koordinati točke, sta definirana kot funkciji in ne kot fiksna števila, saj s tem omogočimo večjo fleksibilnost točk. Vmesnik za upravljanje s krivuljo, ki temelji na kontrolnih točkah je:

```

1 interface PointControlledCurve {
2     eval(t: number): Point;
3     transform(A: number[][][], b: number[], center: number[]): void;
4     getPoints(): Point[];
5     setPoints(points: Point[]): void;
6 }.

```

6.2.2 Implementacije točk

V projektu se nahajata dve implementaciji vmesnika Point. Za osnovno delo s krivuljami je na voljo razred PointImpl, ki predstavlja osnovno implementacijo metoda vmesnika. Za primer pokažimo konstruktor razreda ter implementacijo metode X v tem razredu:

```

1 class PointImpl implements Point {
2     constructor(x: number | (() => number), y: number | (() =>
3         number)) {
4         this.x = x;
5         this.y = y;
6     }
7     ...
8     X(): number {

```

```

8         if (typeof this.x == 'number') {
9             return this.x;
10        }
11        return this.x();
12    }
13    ...
14}.

```

Takšna implementacija omogoča, da definiramo točke, ki so odvisne od drugih točk. Na primer:

```

1 a = new PointImpl(-1,1);
2 b = new PointImpl(() => a.X() + 2, () => a.Y());

```

Kot koordinati točke b smo podali funkciji, ki v izračunu evalvirata koordinati točke a. Premik točke a (z metodo setX oziroma setY), zato premakne tudi točko b.

Druga implementacija vmesnika Point, razred JXGPointWrapper, predstavlja ovoj za točke iz knjižnice JXG. Implementacije metod vmesnika zato delo le prenesejo na metode ovite točke. Ovoj nam omogoča, da točke iz knjižnice JXG uporabljamo pri konstrukciji svojih krivulj.

6.2.3 Implementacije krivulj

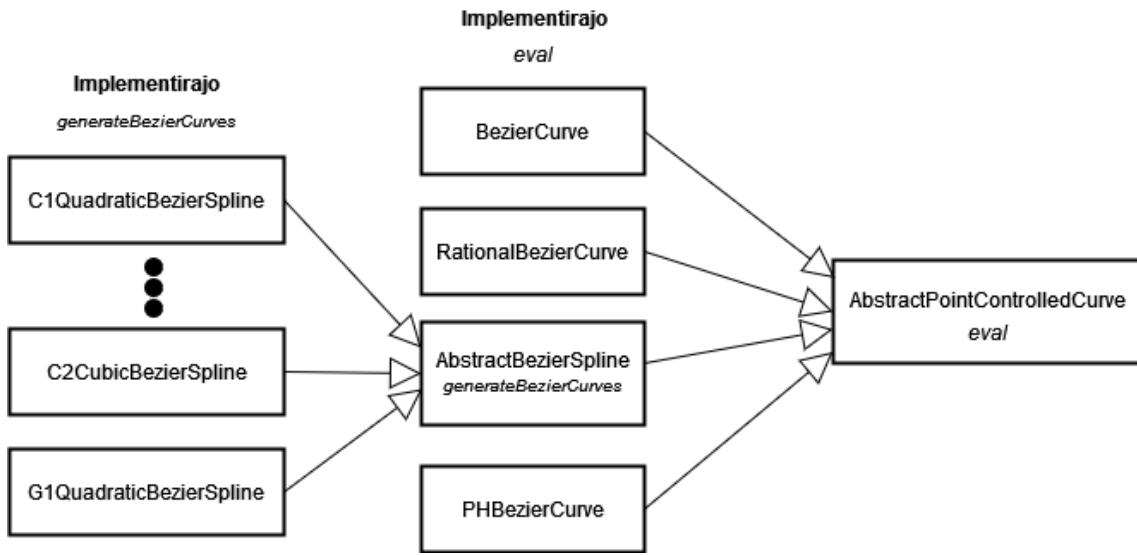
V naslednjih nekaj razdelkih, bosta večkrat omenjena koncepta abstraktnega razreda in abstraktne metode, zato ju na tem mestu na kratko opišemo. Abstrakten razred je tak razred, ki definira (ponavadi tudi uporablja) vsaj eno abstraktno metodo. Abstraktna metoda pa je neke vrste vmesnik za metodo, ki jo podrazredi abstraktnega razreda implementirajo.

Osnovne implementacije vmesnika PointControlledCurve so združene v abstraktnem razredu AbstractPointControlledCurve, ki implementira vse metode, razen metode eval. Metodo eval implementacije razreda AbstractPointControlledCurve implementirajo poljubno. Razredno hierarhijo krivulj v Bezgu si lahko ogledamo na sliki 46. Za primer pokažimo kako je implementirana metoda transform v razredu AbstractPointControlledCurve. Ker so vse krivulje iz prejšnjih razdelkov, razen PH krivulj, afino invariantne, jih transformiramo tako, da transformiramo njihove kontrolne točke:

```

1 abstract class AbstractPointControlledCurve implements
2     PointControlledCurve {
3     points: Point[];
4     ...
5     transform(A: number[][], b: number[], center: number[]): void {
6         this.points.forEach(point => {
7             x = point.X();
8             y = point.Y();
9             x = x - center[0];
10            y = y - center[1];
11            newX = A[0][0] * x + A[0][1] * y;
12            newY = A[1][0] * x + A[1][1] * y;
13            newX = newX + center[0] + b[0];
14            newY = newY + center[1] + b[1];
15            point.setX(newX);
16            point.setY(newY);
17        });
18    }
19}

```



Slika 44: Razredna hierarhija krivulj.

```

17     ...
18 }.

```

Polinomska in racionalna Bézierjeva krivulja

Razreda, ki predstavlja polinomsko ter racionalno Bézierjevo krivuljo, poleg metod vmesnika PointControlledCurve implementirata tudi metode elevate, extrapolate in subdivide. Metode vračajo nove krivulje in prvotne krivulje ne spreminjajo. Implementirane so po korakih iz pripadajočih razdelkov. Za primer si oglejmo implementaciji metod eval in elevate razreda PolynomialBezierCurve:

```

1  class PolynomialBezierCurve extends AbstractPointControlledCurve {
2      ...
3      eval(t: number): Point {
4          points = copy(this.getPoints());
5          n = points.length - 1;
6          for (r = 1; r <= n; r++) {
7              for (i = 0; i <= n-r; i++) {
8                  points[i] = (1-t)*points[i] + t*points[i+1];
9              }
10         }
11         return points[0];
12     }
13
14     elevate(): PolynomialBezierCurve {
15         points = this.getPoints();
16         n = points.length;
17         newPoints = [];
18         newPoints.push(points[0]);
19         for (let i=1; i<n; i++) {
20             newPoints.push(newPoint);
21         }
22         newPoints.push(this.points[n-1]);

```

```

23     return new PolynomialBezierCurve(newPoints);
24 }
25 ...
26 }.
```

Razred, ki predstavlja racionalno Bézierjevo krivuljo, ima poleg polja za kontrolne točke tudi polje za uteži. Za primer si oglejmo, kako implementira metodi eval in extrapolate:

```

1 class RationalBezierCurve extends AbstractPointControlledCurve
2     implements BezierCurve {
3     weights: number[];
4     ...
5     eval(t: number): Point {
6         points = copy(this.getPoints());
7         w = copy(this.weights);
8         n = points.length - 1;
9         for (r = 1; r <= n; r++) {
10             for (i = 0; i <= n-r; i++) {
11                 w = (1 - t) * w1 + t * w2;
12                 points[i] = (1-t)*w[i]/w*points[i] + t*w[i+1]/w*points[i]
13                     +1];
14                 w[i] = w;
15             }
16         }
17         return points[0];
18     }
19     extrapolate(t: number): this {
20         weights = this.weights;
21         // Transformiramo koordinate v homogene
22         points = this.getPoints().map(
23             (point, i) => [point.X()*weights[i], point.Y()*weights[i],
24                 weights[i]]
25         );
26         // Naredimo deCasteljaujev algoritmom na homogenih tockah
27         decasteljauScheme = deCasteljau(t, points);
28         pointsToProject = decasteljauScheme.map((row, i) => row[0]);
29         // Izbrane tocke projiciramo nazaj
30         newPoints = pointsToProject.map(p=>new PointImpl(p[0]/p[2], p[1]/
31             p[2]));
32         newWeights = pointsToProject.map(p => p[2]);
33         return new RationalBezierCurve(newPoints, newWeights);
34     }
35     ...
36 }.
```

Zlepki Bézierjevih krivulj

Osnovno implementacijo zlepka Bézierjevih krivulj ponazarjam z abstraktnim razredom AbstractBezierSpline. Razred ima polje, ki vsebuje seznam Bézierjevih krivulj iz katerih je sestavljen, ter abstraktno metodo generateBezierCurves, ki te krivulje generira. Posamezna implementacija razreda metodo generateBezierCurves implementira, da generira ustrezne krivulje. Za primer si oglejmo implementacijo metode eval v razredu AbstractBezierSpline, kjer metoda getU vrača normalizirane

stične točke zlepka:

```
1 abstract class AbstractBezierSpline extends
2     AbstractPointControlledCurve {
3         bezierCurves: PolynomialBezierCurve[];
4         ...
5         constructor(points: Array<Point>) {
6             super(points);
7             this.generateBezierCurves();
8         }
9         eval(t: number): Point {
10            n = this.bezierCurves.length;
11            u = this.getU();
12            for (i = 0; i < n; i++) {
13                if (t < u[i+1]) {
14                    t = t - u[i];
15                    t = t / (u[i+1] - u[i]);
16                    return this.bezierCurves[i].eval(t);
17                }
18            }
19            return this.bezierCurves[this.bezierCurves.length - 1].eval(1);
20        }
21    }.
```

Kot primer implementacije razreda AbstractBezierSpline si oglejmo implementacijo enostransko konstruiranega C^r zlepka n -te, ki je predstavljen z razredom GenericBezierSpline. Razred ima poleg kontrolnih točk tudi polji za stopnjo krivulje in stopnjo zveznosti. Oglejmo si, kako je implementiran konstruktor razreda, ter metoda generateBezierCurves, ki krivulje generira po algoritmu 3:

```
1 class GenericBezierSpline extends AbstractBezierSpline {
2     degree: number;
3     continuity: number;
4
5     constructor(points: Array<Point>, degree: number, continuity:
6                 number) {
7         super(points);
8         this.degree = degree;
9         this.continuity = continuity;
10        this.generateBezierCurves();
11    }
12    ...
13    generateBezierCurves() {
14        this.bezierCurves = [];
15        step = this.degree - this.continuity;
16        bezierCurvePoints = this.points.slice(0, this.degree+1);
17        this.bezierCurves.push(new PolynomialBezierCurve(
18            bezierCurvePoints));
19
20        for (let i = this.degree; i < this.points.length-1; i = i+step) {
21            bezierCurvePoints = [this.points[i]];
22
23            l = this.bezierCurves.length;
24            previousBezierCurve = this.bezierCurves[l - 1];
25            for (let j = 1; j <= this.continuity; j++) {
26                nonFreePoint = new PointImpl(() => {
```

```

25         u = this.getU();
26         deltaU1 = u[1] - u[1-1];
27         deltaU2 = u[1+1] - u[1];
28         nextT = (deltaU2+deltaU1) / deltaU1;
29         decasteljau = previousBezierCurve.decasteljauScheme(
30             nextT);
31         return decasteljau[this.degree][this.degree - j].X();
32     }, () => {...});
33     bezierCurvePoints.push(nonFreePoint);
34 }
35
36     bezierCurvePoints.push(...this.points.slice(i+1, i+step+1));
37     this.bezierCurves.push(new PolynomialBezierCurve(
38         bezierCurvePoints));
39 }
40 ...
41 .

```

Podobno so implementirani tudi ostali zlepki iz magistrskega dela, kjer je metoda generateBezierCurves za posamezen zlepek implementirana po pripadajočem algoritmu iz razdelka 4.2. Implementaciji G^1 in G^2 zlepkov imata dodatno polje, ki drži števila β_i .

Bézierjeve PH krivulje

Za delo z Bézierjevimi PH krivuljami je na voljo razred PhBezierCurve. Razred prejme kontrolno točko p_0 in točke w_k , predstavljene v razdelku 5.8. Na podlagi števila vnešenih točk w_k se razred nato odloči, katere izraze iz razdelka 5.8 uporabi pri generaciji točk polinomske Bézierjeve krivulje. Ker Bézierjeve PH krivulje niso afino invariantne, metodo transform povozimo, da v primeru klica vrne napako. Rotacija, enakomerno skaliranje krivulje v obe smeri in premik krivulje obdržijo PH lastnost, zato so v razredu implementirane metode, ki te funkcionalnosti podpirajo. Za primer si oglejmo konstruktor razreda, povoženo funkcijo transform in funkciji moveFor ter scale:

```

1 export class PhBezierCurve extends AbstractPointControlledCurve {
2     private underlyingBezierCurve: BezierCurve;
3     private underlyingCurveControlPoints: Point[];
4     private w: Point[];
5
6     constructor(point: Point, w: Point[]) {
7         super([point]);
8         this.w = w;
9         degree = 2 * w.length - 1;
10        this.underlyingCurveControlPoints = this.
11            generatePointsForDegree(degree);
12        this.underlyingBezierCurve = new PolynomialBezierCurve(this.
13            underlyingCurveControlPoints);
14    }
15    ...
16
17    override transform(A: number[][] , b: number[], center: number[]): void {
18        throw "PH curves don't allow general affine transformations.";
19    }

```

```

17 });
18
19 moveFor(x: number, y: number) {
20     point = this.getPoints()[0];
21     point.setX(point.X() + x);
22     point.setY(point.Y() + y);
23 }
24
25 scale(scale: number) {
26     scale = Math.sqrt(scale);
27     this.w.forEach(point => {
28         point.setX(scale * point.X());
29         point.setY(scale * point.Y());
30     });
31 }
32 ...
33 }.

```

6.2.4 Povezava implementacij krivulj s knjižnico JXG

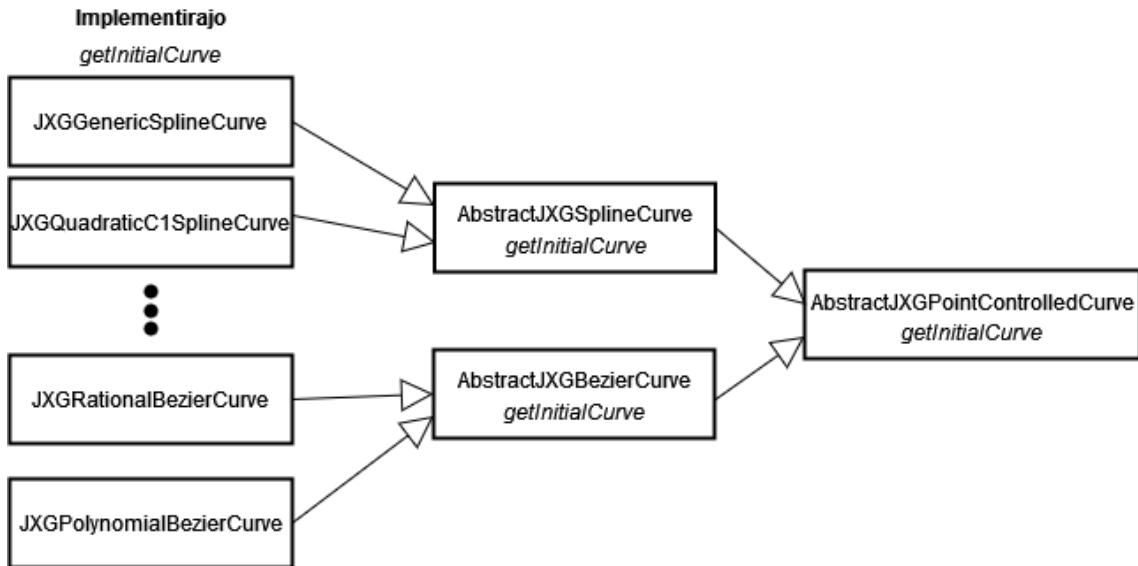
V Bezgu interaktivne Bézierjeve krivulje rišemo s pomočjo knjižnice JXG. V razdelku 6.2.1 smo povedali, da imamo na voljo razred, ki ovije točke iz knjižnice JXG in omogoča njihovo pri konstrukciji naših krivulj. Oglejmo si, kako lahko krivulje iz prejšnjega razdelka s knjižnico JXG narišemo in naredimo interaktivne.

```

1 // Ustvarimo interaktivne tocke knjiznice JXG
2 jxgPoints = [[-1, 0],[0, 1],[1, 1],[2,0]].map(p=> board.create(
3     'point', [p[0], p[1]]));
4 // Tocke ovijemo
5 wrappedPoints = jxgPoints.map(p=> new JXGPointWrapper(p));
6 // Ustvarimo Bezgovo Bezierjevo krivuljo
7 curve = new PolynomialBezierCurve(wrappedPoints);
8 // Ustvarimo se JXG krivuljo
9 board.create('curve',
10     [(t: number) => curve.eval(t).X(),
11      (t: number) => curve.eval(t).Y(),
12      0,1]
12 );

```

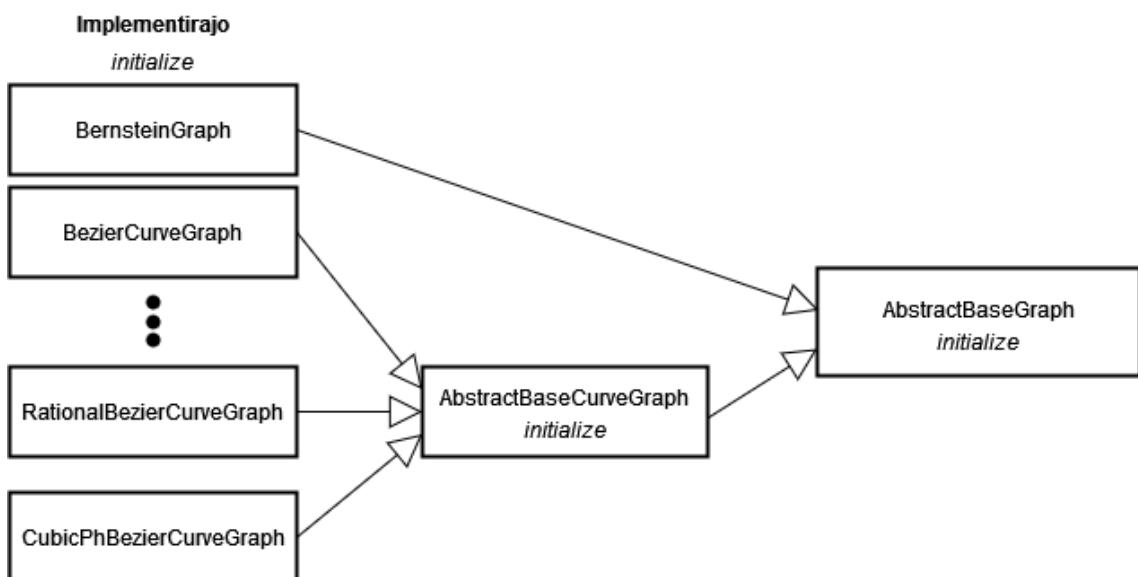
Na grafu se po izvedenih korakih izriše Bézierjeva krivulja z izbranimi kontrolnimi točkami. Premik kontrolnih točk spreminja obliko krivulje. Za zgornje in ostalo delo potrebno za izris grafa, v Bezgu poskrbi abstrakten razred AbstractJXGPointControlledCurve z abstraktno metodo getInitialCurve. Metodo getInitialCurve implementacije razreda implementirajo, da izrišejo svojo krivuljo. Razredno hierarhijo si lahko ogledamo na sliki 45. Poleg risanja krivulje na graf, razred implementira tudi druge metode za delo s krivuljo npr. metodo za prikazovanje kontrolnega poligona, metodo za prikazovanje konveksne ovojnice, metodo za prikazovanje in skrivanje kontrolnih točk ipd.. Poleg metod namenjenih uporabniku, ima razred JXGPointControlledCurve tudi metode za procesiranje potez uporabnikove miške processMouseDown, processMouseMove, processMouseUp, ki skrbijo za to, da lahko uporabnik z rabo miške upravlja z izbrano krivuljo.



Slika 45: Razredna hierarhija JXG krivulj.

6.2.5 Grafi

Za inicializacijo grafa knjižnice JXG in funkcionalnosti naštete v razdelku 6.1 skrbi abstrakten razred `AbstractBaseGraph`, ki ima abstraktno metodo `initialize`. Posamezna implementacija razreda `AbstractBaseGraph` implementira metodo `initialize`, da na grafu nariše željeno. Za funkcionalnost izbere krivulje poskrbi abstrakten razred `AbstractBaseCurveGraph`, ki metode `initialize` ne implementira. Metodo implementirajo posamezni razredi, da narišejo potrebne krivulje. Razredno arhitekturo grafov si lahko ogledamo na sliki 46. Kot primer implementacije metode `initialize` si



Slika 46: Razredna hierarhija grafov.

oglejmo kako graf racionalne Béziereve krivulje nariše svojo krivuljo:

```

1   class RationalCurveGraph extends AbstractBaseCurveGraph {
2   ...
3       initialize(): void {
4           this.curve = new JXGRationalBezierCurve
5               ([[1,2],[2,2],[3,2]],[1,1,1],this.board);
6       }
7   }.
```

Posamezni grafi lahko ukaze na kartico z ukazi na grafu dodajo tako, da povozijo metodo getGraphCommands. Za primer si oglejmo, kako graf za višanje stopnje krivulje ustvari gumb, ki ob kliku zviša stopnjo krivulje:

```

1 class ElevationGraph extends AbstractBaseCurveGraph {
2 ...
3     override getGraphCommands(): JSX.Element[] {
4         return [<Button onClick={() => this.elevate()}>Dvigni
5             stopnjo</Button>];
6     }
7 }.
```

Ukaze za kartico izbrane krivulje poda izbrana krivulja sama. Vse implementacije razreda JXGPointControlledCurve imajo metodo getCurveCommands, ki vrača seznam ukazov, s katerimi jo upravljamo. Na primer razred JXGPHBezierCurve na naslednji način poda ukaze za upravljanje z odmiki krivulje:

```

1 class JXGPHBezierCurve extends JXGPointControlledCurve {
2 ...
3     getCurveCommands(): JSX.Element[] {
4         return [<OffsetCurveSettings></OffsetCurveSettings>];
5     }
6 ...
7 }.
```

7 Zaključek

Večino dela smo posvečali predstavitvi Bezierjevih krivulj (in njihovih variacij) ter PH krivulj. Opisali smo njihove glavne lastnosti in podali nekaj algoritmov, ki omogočajo njihovo rabo v CAD in CAGD sistemih. Zaradi obširnosti snovi, smo marsikateri dokaz izpustili. Manjkajoče dokaze, in več o predstavljenih krivuljah, lahko bralec najde v delih [6] in [4].

V koncu dela smo predstavili spletno orodje Bezeg, ki je bilo v okviru dela tudi izdelano. Orodje uporabniku omogoča grafično prikazovanje lastnosti Bezierjevih in PH krivulj. Grafe v spletnem orodju lahko uporabnik (do neke mere) prilagodi svojim željam, prilagoditve pa shrani kot prednastavitev, ki jih po želji tudi prenaša med računalniki. Orodje ima veliko prostora za izboljšave in razširitve. Da bi Bezeg študentom bolje služil kot učno orodje, bi lahko poleg grafov bila razložena tudi snov, ki je na grafu predstavljena. Tako bi se lahko študent o krivuljah učil na enem mestu brez, da bi moral podporno snov iskati drugje. Da bi predstavitve grafov bile bolj prilagodljive, bi lahko eden izmed grafov predstavljal t. i. peskovnik, kjer bi lahko uporabnik vstavljal poljubne krivulje. V primeru širše rabe orodja, bi bilo smiselno dodati zaledni sistem za delo z uporabniki in njihovimi prednastavitevami. Poleg izboljšav samega orodja, bi lahko razširili tudi obseg snovi, ki je v orodu implementirana. Snov bi bilo možno razširiti tako na snov o interpolacijskih zlepkih, B-zlepkih, racionalnih PH krivuljah ipd., kot tudi na snov o Bezierjevih ploskvah, saj knjižnica JSXGraph podpira tridimenzionalne grafe.

Literatura

- [1] *About GitHub Pages*, <https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages>, Dostopno 9.9.2024.
- [2] *Bezeg - Orodje za grafični prikaz lastnosti Bézierjevih in PH krivulj*, <https://efodx.github.io/bezeg/>, Dostopno 12.9.2024.
- [3] *Bootstrap · The most popular HTML, CSS, and JS library in the world*. <https://getbootstrap.com/>, Dostopno 12.9.2024.
- [4] G. Farin, *Curves and surfaces for cagd: a practical guide*, 5th, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.
- [5] R. T. Farouki in T. Sakkalis, *Pythagorean hodographs*, IBM Journal of Research and Development **34**(5) (1990) 736–752, DOI: 10.1147/rd.345.0736.
- [6] R. Farouki, *Pythagorean-hodograph curves: algebra and geometry inseparable*, 1st, Springer Publishing Company, Incorporated, 2007.
- [7] R. Farouki in T. Goodman, *On the optimal stability of the bernstein basis*, Math. Comput. **65** (1996) 1553–1566, DOI: 10.1090/S0025-5718-96-00759-4.
- [8] *GitHub repozitorij Bezga*, <https://github.com/efodx/bezeg>, Dostopno 12.9.2024.
- [9] *JSXGraph - Dynamic Mathematics with JavaScript*, <https://jsxgraph.uni-bayreuth.de/wp/index.html>, Dostopno 12.9.2024.
- [10] *React - The library for web and native user interfaces*, <https://react.dev/>, Dostopno 12.9.2024.
- [11] *React Bootstrap - The most popular front-end framework, rebuilt for React*, <https://react-bootstrap.netlify.app/>, Dostopno 12.9.2024.
- [12] *TypeScript: JavaScript with syntax for type*, <https://www.typescriptlang.org/>, Dostopno 12.9.2024.