Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gregor Poročnik

Implementacija izrisa Bézierovih krivulj in B-zlepkov v HTML5

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Matija Marolt

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1]Več o licenci na http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00299/2012
Datum: 13.04.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GREGOR POROČNIK**

Naslov: **IMPLEMENTACIJA IZRISA BEZIEROVIH KRIVULJ IN B-ZLEPKOV V HTML5**

DRAWING BEZIER CURVES AND B-SPLINES IN HTML5

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

V diplomski nalogi izdelajte aplikacijo za izris parametričnih krivulj v brskalnikih s podporo HTML5. Osredotočite se na Bezierove krivulje in B-zlepke in opišite njihovo matematično ozadje ter postopke za učinkovit izris. Aplikacija naj podpira postavljanje, premikanje in brisanje kontrolnih točk, izris krivulj poljubne stopnje in izris mešalnih funkcij.

Mentor:

doc. dr. Matija Marolt

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Gregor Poročnik, z vpisno številko **63090330**, sem avtor diplomskega dela z naslovom:

Implementacija izrisa Bézierovih krivulj in B-zlepkov v HTML5
Bézier and B-splines implementation in HTML5

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matije Marolta,
- so elektronska oblika diplomskega dela, naslov, povzetek ter ključne besede identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 24. septembra 2012 Podpis avtorja:

Rad bi se zahvalil vsem, ki so me na kakršen koli način spodbujali in podpirali tekom študija. Predvsem bi se zahvalil družini za podporo in potrpežljivost v težkih in stresnih trenutkih, ki jih v času študija ni bilo malo. Poleg tega bi se zahvalil svojim kolegom za vzajemno pomoč na laboratorijskih vajah, ko je bilo to potrebno, ter za vesele trenutke ob študiju.

Za pomoč pri pisanju diplomskega dela se moram zahvaliti mentorju doc. dr. Matiji Maroltu, ki me je podprl s svojim mentorstvom, mi ob tem nudil tehnične nasvete pri implementaciji dela, svetoval pri vsebini, poleg tega pa mi omogočil tudi nekaj dodatne literature.

Kazalo

Seznam uporabljenih kratic in simbolov

Izraz	Slovenski prevod	Opis
API (Application Programming Interface)	Vmesnik uporabniškega programa	Vmesnik, ki nudi podporo pri razvoju programske opreme.
OS (Operating System)	Operacijski sistem	Programska oprema, nujna za delovanje računalnika, ki deluje kot vmesnik med strojno in programsko opremo.
JS (JavaScript)	Javaskripta	Skriptni jezik, ki omogoča razvoj dinamičnih vsebin na spletu.
Applet	Applet	Vtičnik na brskalniku, ki omogoča poganjanje Java razredov na odjemalcu.
Canvas	Canvas	HTML5 element, ki omogoča izrisovanje likov na spletno stran.
CAD (Computer Aided Design)	Računalniško podprto risanje	Programska oprema, ki s pomočjo vgrajenih algoritmov omogoča izris ter načrtovanje produktov.
CAM (Computer Aided Manufacturing)	Računalniško podprta proizvodnja	Programska oprema, ki omogoča proizvodnjo izdelkov v industriji s pomočjo CAD.
HTML5 (Hyper Text Markup Language)	Jezik za označevanje besedila	Spletni označevalni jezik, s pomočjo katerega so sestavljene spletne strani.
MS (Microsoft)	Microsoft	Znano podjetje, ki izdeluje programsko opremo.
OSX	OSX	Operacijski sistem podjetja Apple.
2D	2D	2-dimenzionalen (x-y)

Povzetek

Vsebina diplomskega dela se navezuje na eno izmed vej računalniške znanosti, računalniško grafiko. Glavna tema dela so krivulje (*ang. curves*), njihov izris ter implementacija v programskem jeziku. Praktični del je implementiran v programskem jeziku Javascript s podporo HTML5 canvas elementa za grafično izrisovanje. V delu so opisane tipične metode izrisa krivulj, teoremi krivulj z matematični ter praktični primeri izrisa. Opisane so inženirske metode, ki prikazujejo kako od velikega projekta oziroma težave preko razbitij priti do izzivov manjših razsežnosti in postopoma rešiti bolj obvladljive enote. Tematika se predvsem osredotoči na Bézierove krivulje ter B-zlepke, za katere je v praktičnem delu ločena implementacija, ki poučno prikazuje tematiko. Vsebina dela je povezana z aksiomi, ki poleg primerov teoretično podprejo dano tematiko. Poleg tega je omenjena praktična uporaba krivulj s podrobnim opisom določenih korakov ter metod.

Ključne besede: krivulja, računalniška grafika, Pierre Bézier, Paul de Casteljau, B-zlepki

Abstract

The content of the thesis is related to one of the branches of computer science, computer graphics. The main subject of my thesis are curves, their plotting and implementation in programming language. The practical part is implemented in Javascript programming language with support of HTML5 canvas element for graphics rendering. This paper describes typical methods of drawing curves as well as its theorems with mathematical and practical examples. Engineering methods which are described show how one can solve a big project or a problem by breaking it into challenges of smaller dimensions and gradually solving these more manageable units. The topic mainly focuses on Bezier curve and B-splines which, in practical terms, have separate implementations which shows the topic from an educational point of view. The content is related to axioms that, along with examples, support the theoretical issues. Also, the practical application of said curve is mentioned with a detailed description of specific steps and methods.

Keywords: curve, computer graphics, Bézier Pierre, Paul de Casteljau, B-splines

Poglavje 1

Uvod

1.1 Opredelitev problema

Cilj dela je izdelati aplikacijo za demonstracijo Bézierovih krivulj ter B-zlepkov. Program je namenjen ljudem, ki potrebujejo grafično predstavitev krivulj za izobraževalne namene. Ker program omogoča izris krivulj, odvisnih od vnesenih parametrov, se uporabnik lahko sam odloča in preizkuša odziv glede na vnesene nastavitve. Takšen način učenja je bliže uporabniku, saj se vsak primer izriše v trenutku, kljub zelo dolgi računski operaciji. Funkcionalnosti programa omogočajo izpis izračunanih koordinat za določene parametre in se lahko uporabi kot testni primer. Program omogoča poleg tega tudi izris bolj ali manj zapletenih likov, kot so srce, trikotnik, krog in podobnih, tako da je primeren tudi za osnovnošolce, ki spoznavajo geometrijske like. Tovrstne aplikacije trenutno ni mogoče najti na spletu, saj za zagon aplikacije uporabnik ne potrebuje nikakršnih vtičnikov, temveč le novejši brskalnik s podporo HTML5. Poleg tega naloga vsebuje unikatne rešitve, saj trenutno takšnih primerov na takšni platformi ni mogoče najti. To je bil tudi razlog za izgradnjo nove tovrstne aplikacije.

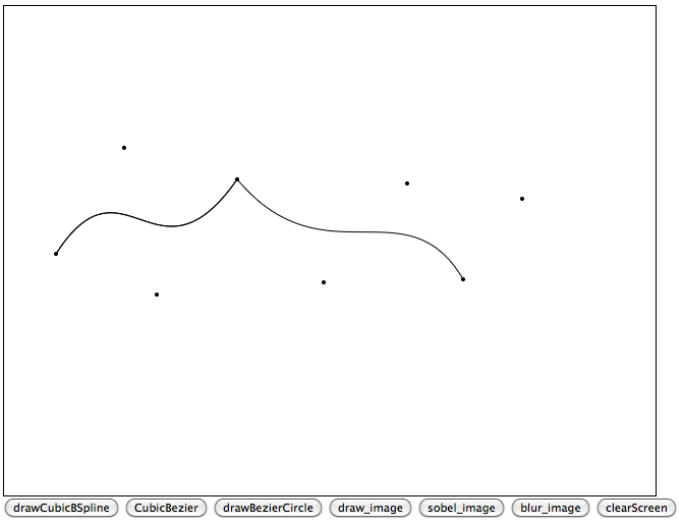
1.2 Primeri tovrstnih aplikacij na spletu

Med tovrstnimi aplikacijami na spletu ni bilo mogoče najti takšne, ki bi omogočala funkcionalnosti, opisane v tem delu. Obstajajo primeri, ki omogočajo izris Bézierovih krivulj tretje stopnje brez funkcionalnosti B-zlepkov ali pa se pojavljajo programi z zelo okrnjenimi funkcionalnostmi obeh tipov krivulj [5]. Avtorji se navadno poslužujejo takšnih rešitev, ki omogočajo trivialno implementacijo preko vgrajenih funkcionalnosti. Takšen izris je enostavnejši za implementacijo in za to avtorji ne vložijo veliko truda. Za primer: krivulje tretje stopnje je mogoče izrisati že z obstoječo Javascript funkcijo, ki je vgrajena v Javascript API.

```
1  //define variables
2  var x, y, a, b, c, d, e, f, ctx;
3  //get context from canvas
4  ctx = canvas.getContext("2d");
5  // start drawing
6  ctx.beginPath();
7  //move pointer to position
8  ctx.moveTo(x, y);
9  //draw bezier
10 ctx.bezierCurveTo(a, b, c, d, e, f);
11 ctx.stroke();
```

Primer 1.1: Integrirana funkcija za izris kubične Bézierove krivulje

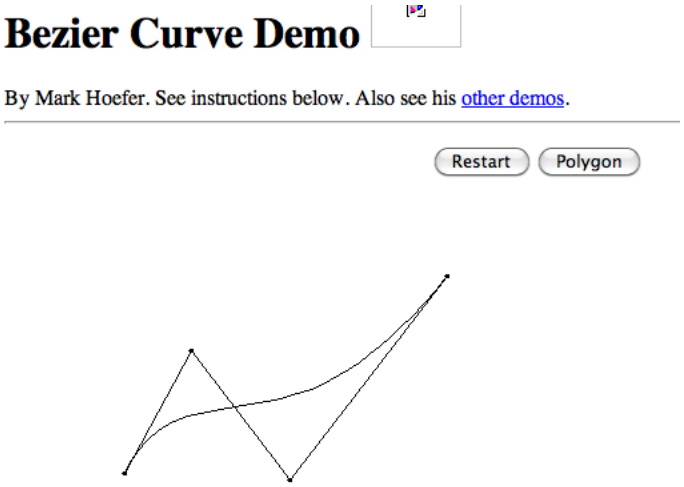
Pri tem je x in y koordinata, pri kateri se prične izris, parametri od a do f pa so tri kontrolne točke, preko katerih se krivulja izračuna. Takšen primer izrisa uporablja primer aplikacije na spletu z vira [3].



Slika 1.1: Primer programa s spleta, ki uporablja Javascript HTML5 [3].

Na sliki 1.1 je program, ki omogoča izris Bézierovih krivulj tretje stopnje z vgrajeno funkcijo bezierCurveTo(). Ta postopek je nekoliko izboljšan tako, da se ob večjem številu točk pričnejo izrisovati zlepki, kakor je prikazano.

Obstajajo tudi primeri iz spleta (slika 1.2), ki omogočajo nekaj več funkcionalnosti. Takšni primeri so implementirani v programskem jeziku Java, za delovanje pa potrebujemo na računalniku nameščen določen programski paket. Težava pri takšnih aplikacijah je, da je onemogočen dostop uporabniku, ki si lasti napravo z vgrajenim brskalnikom, ne omogoča pa poganjanja javanskih Appletov.



Slika 1.2: Primer programa ki teče na programskem jeziku Java kot Applet [4].

Takšen javanski program ima nekaj prednosti pred rešitvijo v Javascript jeziku, saj je programska koda skrita uporabniku, razvijalcu pa omogoča svobodno odločitev pri izdaji programske kode. Gledano z drugega stališča je program namenjen izobraževanju, zato je koda ključnega pomena ob interpretiranju določene rešitve. Preko programske kode je uporabniku omogočen vpogled skozi izvajanje programa in je zato ključen dejavnik izobraževanja.

1.3 Uvod v krivulje

Krivulje in z njimi povezani programi za izris, na primer CAD, so zelo pomembni v današnji industriji. Avtomobilska industrija je zelo odvisna zaradi aerodinamike, vizualnih lastnosti ter ostalih lastnosti produktov, zato je že leta 1962 francoski inženir Pierre Bézier pričel graditi osnovne CAD/CAM sisteme za prikaz krivulj. Zaposlen je bil kot inženir v avtomobilskem pod-

jetju Renault, ki izdeluje avtomobile še danes. Zaradi patenta, ki ga je registriral, imajo krivulje njegovo ime, kljub temu da ta čast pripada njegovem kolegu Paulu de Casteljauju. Nekaj let pred tem (1959) je francoski matematik Paul de Casteljau razvil enega prvih algoritmov za grafično predstavitev krivulj, za katerega se je izkazalo, da je zelo učinkovit. Primera algoritma in izrisa sledita kasneje v delu. Poleg Pierra je bil inženir De Casteljau prav tako matematik in tudi fizik, ki je bil sicer zaposlen kot inženir v francoskem podjetju Citroen.

Za predstavitev krivulj potrebujemo vsaj dvodimenzionalen (x-y) prostor, v naprednejših okoljih pa se lahko za izris uporabijo tudi večdimenzionalni prostori. Nekatere izmed enostavnejših primerov krivulj najdemo že v vsakem programu, ki omogoča risanje. Preko OS Windows lahko kreiramo enostavnejšo krivuljo že s samim programom risar. Prav tako lahko izrisujemo enake krivulje v programskem paketu MS Office, ki z nekoliko naprednejšimi funkcionalnostmi skrbi za obogateno uporabniško izkušnjo. Pri tem gre le za orodja, namenjena pisarniški uporabi, pri grafičnem oblikovanju ter modeliranju pa razvijalci uporabljajo orodja, kot so na primer Photoshop, podjetja Adobe. Zmotno je mišljenje, da se pri tem stvari končajo, saj je potrebno vzeti v zakup, da so se konkretne spremembe dogajale tudi v mehaniki in strojništvu nasploh. V strojništvu so se nekoč uporabljale Bézierove krivulje ter B-zlepki, v današnji dobi pa naj bi jih nadomestile alternative.

1.4 Matematično stališče

Iz matematičnega stališča poznamo več vrst krivulj, od najbolj trivialne, katero lahko predstavimo z ravno črto, do kompleksnejših, kot so zlepki krivulj različnih stopenj. Krivulje praktično ni težko definirati, saj jo lahko izrišemo s pomočjo ostrega svinčnika, s katerim drsimo po listu papirja. Krivulja je tako neskončno velika množica točk, ob pogoju, da ima vsaka točka le dve sosednji točki - Točko ki je predhodna na izrisu, in točko ki jo nasledi. Ob tem veljajo posebnosti za točke, ki imajo le eno sosednjo točko. To sta točki,

ki sta zadnji v redu in zaključujeta krivuljo na vsaki strani. Teh točk sicer ni v primerih, kjer je krivulja povezana sama nase (je v zanki) in tvori krog. V matematiki obstajajo trije načini definicije krivulj:

Implicitna prezentacija krivulj definira točke na krivulji, preko katere lahko preverimo, ali določena točka leži na krivulji. Torej je krivulja definirana za točke, pri katerih je izpolnjen naslednji pogoj.

$$f(x, y) = 0 \tag{1.1}$$

Eksplicitna oblika zapisa krivulje omogoča enostaven izračun točke na koordinati y preko vhodnega parametra točke x. V delu se izkaže kot počasen zapis za implementacijo izrisa.

$$y = f(x) \tag{1.2}$$

Parametrična oblika nudi izris preko prostega parametra t, ki se uporablja kasneje v implementaciji. Spremenljivka t sicer običajno teče od 0 do 1, lahko pa je poljubnih vrednosti. V praksi s svinčnikom in papirjem bi to spremenljivko lahko vzeli kot čas, ki teče od pričetka risanja do konca. Tako lahko za vsak trenutek med izrisom povemo, kje je bila konica svinčnika ob danem trenutku.

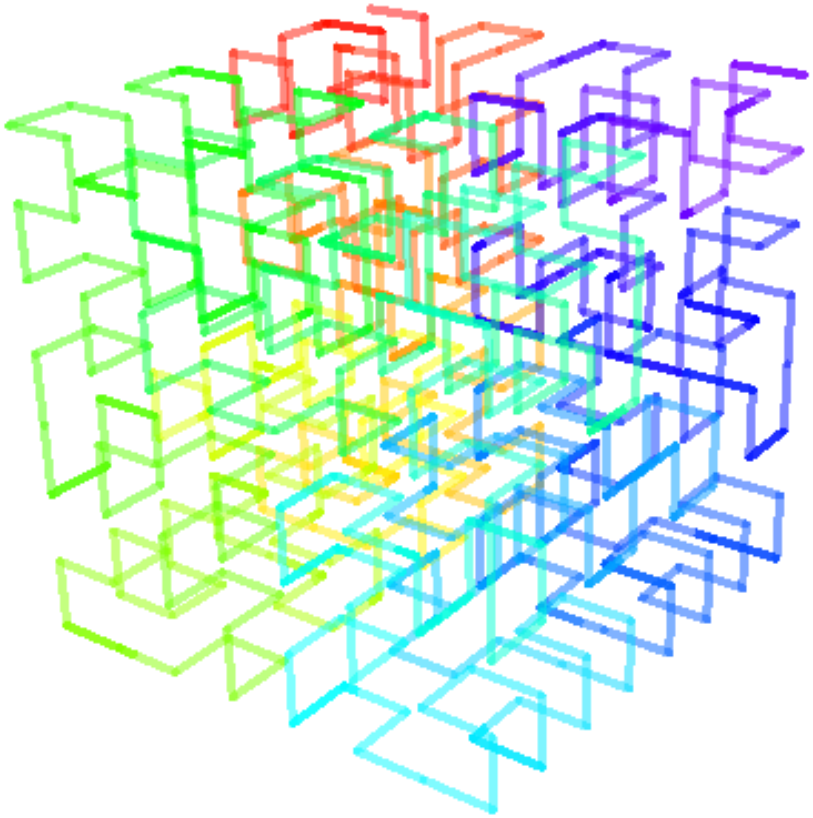
$$x, y = f(t) \tag{1.3}$$

Poglavje 2

Krivulje

2.1 Krivulje

Glavni del dela je namenjen opisu določenih orodij, metod, specifikacij, načinov uporabe, oziroma na splošno tematiki, kot je že opisana v uvodu. Poglavitni so opisi korakov ter določenih individualnih rešitev, ki so skozi proces razvoja tekli od težave do končne rešitve. Te rešitve so v delu predstavljene kot krivulje B-zlepkov, odvisne od parametrov vektorja vozlov, stopnje in kontrolnih točk do izrisa Bézierovih krivulj, katere določa predvsem stopnja in kakovost izrisa, ki je določena z drsnikom. V delu je predstavljenih le nekaj primerov tipov krivulj, poleg teh pa obstaja še veliko drugih, z drugačnimi lastnostmi in pripadajočimi izrisi, kot je na primer Hilbertova krivulja v tridimenzionalnem prostoru, prikazana na sliki 2.1. Poleg implementacije, izvedene v delu, obstaja še veliko možnih rešitev, predstavljena rešitev pa je intelektualno delo s strani avtorja.



Slika 2.1: Primer Hilbertove krivulje v tro-dimenzionalnem prostoru (wikipedia).

2.2 Polinomi

Polinomi so sestavni deli te naloge, saj se preko njih izračunajo vrednosti na krivuljah. Na obliko funkcij predvsem vpliva stopnja polinoma, ki mora imeti nenegativne cele eksponente. Pri tem ločimo polinome glede na njihovo stopnjo, in sicer:

Izrek 2.1 *Polinom ničte stopnje ali konstantni polinom*

$$p(x) = a_0, \text{ če velja: } a_0 \neq 0 \tag{2.1}$$

Izrek 2.2 *Polinom prve stopnje ali linearni polinom*

$$p(x) = a_1x + a_0, \text{ če velja: } a_1 \neq 0 \tag{2.2}$$

Izrek 2.3 *Polinom druge stopnje ali kvadratni polinom.*

$$p(x) = a_2x + a_1x + a_0, \text{ če velja: } a_2 \neq 0 \tag{2.3}$$

Izrek 2.4 *Polinom tretje stopnje ali kubični polinom.*

$$p(x) = a_3x + a_2x + a_1x + a_0, \text{ če velja: } a_3 \neq 0 \tag{2.4}$$

Pri tem je graf oziroma krivulja nepretrgoma zvezna v kolikor ne gre za racionalna števila z neznanim imenovalcem. Prejšnji izreki so izpeljani iz splošnega zapisa polinomov, ki je definiran v izreku 2.5. V delu so uporabljene tudi nekatere računske operacije nad polinomi, ki so predstavljene v sledečih primerih.

Izrek 2.5 *Polinom tretje stopnje ali kubični polinom.*

$$\begin{aligned} p_n(x) &= a_nx + a_{n-1}x + ... + a_1x + a_0 \\ &\textit{ali krajše} \\ p_n(x) &= \sum_{k=0}^{n} a_kx^k \end{aligned} \tag{2.5}$$

Za opisane polinome različnih stopenj se v implementaciji uporablja tudi nekaj osnovnih računskih operacij. Zapišimo.

Izrek 2.6 *Množenje polinoma s konstanto*

$$c * p(x) = \sum_{k=0}^{n} c * a_kx^k \tag{2.6}$$

Izrek 2.7 *Seštevek dveh polinomov*

$$\sum_{k=0}^{n} a_kx^k + \sum_{k=0}^{m} b_kx^k = \sum_{k=0}^{max(n,m)} (a_k + b_k)x^k \tag{2.7}$$

Izrek 2.8 *Odštevanje dveh polinomov*

$$\sum_{k=0}^{n} a_k x^k - \sum_{k=0}^{m} b_k x^k = \sum_{k=0}^{max(n,m)} (a_k - b_k) x^k \tag{2.8}$$

Za enakost polinomov velja, da sta polinoma enaka takrat, kadar se ujemata v stopnji in vseh koeficientih.

2.3 Opisi orodij, načini uporabe, zahtevnost in specifikacije le-teh

Pri izbiri samih orodij, ki jih uporabljamo vsakodnevno ali le občasno, je predvsem pomembno, da se uporabnik pouči o uporabi in namembnosti orodja. Poleg tega se večinoma odločimo tudi med drugimi alternativami, vzroki za to pa so nezadovoljstvo s funkcionalnostmi, slaba preglednost, nezanesljivost programov ali pa so za uporabnika odločilne platformne omejitve njegove naprave.

Pri implementaciji krivulj v jeziku Javascript se soočamo predvsem z odvisnostjo od brskalnika ter orodja za tekstovni vnos programske kode. Brskalniki v tem času niso več tako okrnjeni, kakor so bili nekdaj, zato omogočajo mnogo funkcionalnosti preko dveh možnih virov dostopa. Za uporabnika je pomembnejši grafični vmesnik, interakcija z vmesnikom ter preprostost uporabe programa. S stališča spletnih razvijalcev pa je pomembna funkcionalnost z vidika hitrosti razvoja samega izvajanja in podpora razvojne ekipe podjetja, ki izdeluje brskalnik ter njihove različice, od katerih je po večini lahko odvisna sama rešitev. To trenutno v tej tematiki ne predstavlja pomembnega izziva, saj je za rešitev, katero predstavlja naloga, pomemben vsaj eden izmed novejših brskalnikov, ki omogočajo HTML pete generacije. V tem primeru je bil na razvoju uporabljen program Google Chrome, ki je v razvojnem okolju tekel na OSX. Program je bil primeren predvsem zaradi integrirane konzole, preko katere je možno razhroščevanje, za trenutke,

pri katerih pa koda steče izven meja nadzora, pa je prišlo v pomoč ločeno procesno izvajanje programa.

V nadaljevanju je omenjena tudi kompatibilnost, ki v tem času povzroča težave mnogim razvijalcem. Kot alternativno orodje za razvoj programske kode je možno posegati po mnogi programski opremi: tisti, ki je priložena operacijskim sistemom, ali pa tisti, ki je na voljo na spletu. Za razvoj programske kode je priporočljivo uporabljati program razširjene beležnice Notepad++. Žal je program na voljo le za platformo podjetja Microsoft, zato v tem delu ni bil uporabljen, kot alternativa pa je služil program TextMate, ki deluje na platformi OSX.

Na splošno se omenjena orodja uporabljajo v različne namene. Velika večina uporabnikov izkorišča brskalnike v namen pregledovanja spletnih strani, igranja iger preko vtičnikov ali brskanju po multimedijskih vsebinah, kot so video in glasba, ki jih je na spletu iz ure v uro več. Prav tako velja mnogo načinov uporabe orodij tekstovnih vsebin, kot je TextMate, čeprav takšna orodja po večini omogočajo le vpis podatkov iz periferije v tekstovno datoteko.

2.4 Opis krivulj

Iz geometrije poznamo več tipov krivulj, ki se razlikujejo glede na stopnjo polinoma, preko katerega so definirane, kontrolnih točk v ravnini ter načina izrisa na ravnino. Razlike lahko opazimo pri nekaterih lastnostih, kot so večkratne vrednosti ter lokalnost. Lokalnost je odvisna od tipa in stopnje krivulje, oziroma načina premika točk izrisa ob premiku kontrolnih točk, ki jih definira uporabnik. Pri tem gre za lokalno ter globalno kontrolo kontrolnih točk nad krivuljo. Krivuljo lahko razdelimo na več delov, pri čemer lahko enako obliko predstavlja več manjših krivulj z drugačnimi parametri. Pri pogledu na obliko krivulje lahko govorimo tudi o variaciji krivulje, na katero vplivajo večkratni vozli ali oddaljenost položaja posameznih kontrolnih točk.

2.5 Polinomska aproksimacija

O polinomski aproksimaciji govorimo, ko izrisujemo določeno krivuljo, ki je matematično določena s polinomom določene stopnje. Pri tem se zgodi, da določenih vrednosti ne moramo izračunati zaradi določenih razlogov, zato pričnemo vrednosti aproksimirati. Za primer krivulje prve stopnje govorimo o linearni aproksimaciji, pri kateri aproksimiramo vrednosti med dvema točkama. Z matematičnega vidika rečemo, da gre za problem pri zapisu določenega števila, zato zapišemo približek, ki je takšen kot je predstavljeno v enačbi 2.9.

$$e_i = f_i - g(x_i) \tag{2.9}$$

Pri tem predstavlja e_i dano napako, f_i pravo vrednost, g pa je funkcija preko katere se izračuna aproksimirana vrednost. Iz programerskega stališča navadno pride do aproksimacije zaradi optimizacijskih rešitev ob implementaciji. Eden izmed primerov je izris ravnih črt med točkami, preko katerih bi se sicer morale izračunati nove dejanske vrednosti, ki niso aproksimirane.

Poglavje 3

Implementacija

3.1 Programski jezik Javascript

Programski jezik Javascript (JS) spada med skriptne jezike in nima veliko skupnega s programskim jezikom Java, ki se sicer imenuje podobno. To, da je jezik skriptni, pomeni, da se koda interpretira sproti z izvajanjem programa. Skriptni jeziki so podmnožica programskih jezikov, ker se z njimi programira določene funkcionalnosti, pri tem pa se ločijo po načinu delovanja.

Jezik Javascript se sicer uporablja v različne namene, tudi za programiranje večjih ali manjših skript na spletnih straneh, ki se izvajajo na strani odjemalca, s čimer pa spletni programerji želijo med drugim tudi razbremeniti procesorsko moč na spletnih strežnikih. Jezik se uporablja tudi pri programiranju namiznih aplikacij ter manjših skript, kot so vtičniki. V jeziku je omogočena uporaba objektov, s tem, da so definirani malo drugače, kakor pri drugih programskih jezikih. Jezik omogoča kreiranje objektov in metod preko funkcij, ki lahko sicer stojijo samostojno v programski kodi.

V programskem jeziku je potrebno omeniti posebnost pri računanju s števili in napakami pri zapisu števil. Za primer pri računanju deljenja z 0 dobimo rezultat, vendar je ta znakovnega tipa z vrednostjo *Infinity*. Prav tako je možen rezultat *-Infinity* pri negativnem deljenju zato je potrebna prilagoditev kode. Naslednja posebnost je računanje decimalnih števil, pri

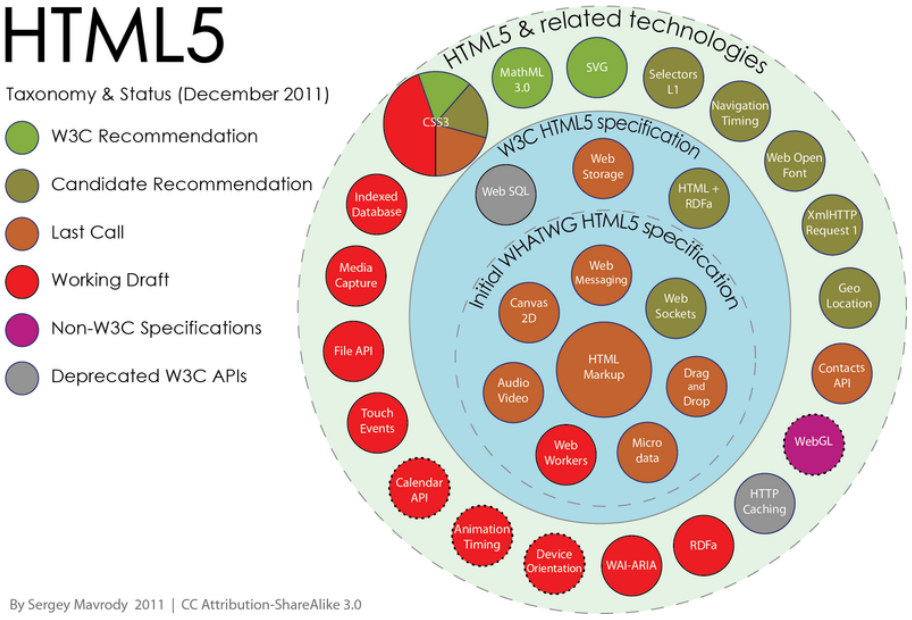
čemer nam pri seštevanju treh enostavnih števil rezultat ne vrne pravilne vrednosti. Pri tem gre za napako pri zaokroževanju števila. Ker ima računalnik za zapis števila določeno število bitov in te lahko uporabi le do določene natančnosti, se število zapiše do določene natančnosti. To napako se lahko delno odpravi z vgrajeno metodo toFixed(), ki poreže majhen del napake, ki se je zgodil med računanjem. Ta metoda je uporabna sicer samo do določene mere, ker se lahko pri napačni uporabi pričenjajo spreminjati dejanske vrednosti rezultatov. Računska napaka se prav tako pojavi pri velikih številskih vrednostih. Za število, ki konvergira proti meji možnega zapisa na strojno opremo, se pričenjajo vrednosti zaokroževati na večjo mejo, za primer, če dovolj velikemu številu prištejemo 1, nam rezultat vrne še vedno enako število. V določenih primerih je takšno obnašanje uporabno, predvsem pri skriptah na spletnih straneh, ki omogočajo razpolaganje z množico vrednosti, ki jih vnaša uporabnik, kljub temu pa se ob nepravilnih vrednostih programi še ne podrejo. To je zato prepuščeno samemu programerju skripte, da lovi takšne napake, obvestila pa pošilja na višji nivo maske.

Resnejši jeziki ob takšnem početju tvorijo izjeme in na drugačen način opozarjajo na svoje težave pri računanju. To je vsekakor boljša izbira, da jezik programerju ne omogoča toliko svobode, saj je zato programska koda najverjetneje hitrejša, poleg tega pa tudi kvalitetnejša.

3.2 HTML5

HTML 5. generacije je označevalni jezik, ki se uporablja v sodobnih različicah spletnih strani. Z novo različico prinaša mnogo dodatnih značk, v tej nalogi je uporabljan canvas, ki obogatijo spletno stran z novimi možnostmi. Poleg tega omogoča tudi dodatne možnosti parametrov obstoječih elementov. Takšen primer je možno najti v nalogi v znački input, ki dodatno omogoča številski tip vnosa. Poleg teh novosti omogoča tudi "drag and drop" funkcionalnost, za katero so v preteklosti poskrbele dodatne knjižnice. Novost pri pregledu multimedijskih vsebin sta elementa video in audio, katera omogočita

integracijo video in glasbenega predvajalnika v vsebino spletne strani.



Slika 3.1: Tehnologije, povezane z HTML5 (wikipedia).

Program ki je napisan v spletnem jeziku HTML pete generacije, je preizkušen na spletnem brskalniku Google Chrome, izvorna koda pa je pregledana s strani pregledovalnika spletnih vsebin organizacije W3C.

3.3 Uporaba programa

Implementiran program ima to lastnost, da deluje v interakciji s človekom, kar pomeni, da se odziva na uporabnikove vnose preko tipkovnice, miške ali katerega drugega uporabniškega vmesnika. Zaradi načina delovanja se pri izrisu takšnega tipa krivulj uporabi prosti vnos konkretnih kontrolnih točk, ki so osnova pri izračunu, s katerim je definiran algoritem. Vnos je predviden preko strojne opreme, natančneje računalniške miške, pri kateri uporabimo levi gumb za dodajanje novih kontrolnih točk. Ob dvakratnem pritisku gumba oziroma dvojnem kliku, kjer miškin kazalec kaže na obstoječo

točko, to izbrišemo. Ta točka se izbriše iz podatkovne strukture, izris pa se prilagodi ostalim točkam, v kolikor so izpolnjeni vsi pogoji za izris izbranega tipa krivulje. Tej lastnosti bi v tem primeru lahko rekli funkcionalnost.

Osnovna naloga programa je, da izrisuje tipe krivulj glede na podane točke in dodatne parametre. Brez te dodatne funkcionalnosti bi uporabnik sicer moral v primeru spremembe osvežiti stran in ponovno opravljati postopek dodajanja točk v koordinatni sistem. Premik točk po dveh dimenzijah koordinatnega sistema je dodatna funkcionalnost, pri kateri lahko s premikom ene ali mnogih kontrolnih točk spremenimo lego in s tem tudi obliko samega geometrijskega lika.

V programu se specifično glede na tip krivulje doda tudi drsnik, ki določa kot med vektorji. S tem drsnikom se praktično nastavlja kakovost izrisa, poleg tega pa je omogočen tudi ročen vpis vrednosti v vnosno polje. Specifično je dodano tudi vpisno polje vektorja vozlov, pri katerem uporabnik vpiše vektor v obliki, kot je preddefinirana ob zagonu programa.

3.4 Bézierove krivulje

Bézierove krivulje so definirane z množico kontrolnih točk kot ostale krivulje, s tem, da si lastijo to lastnost, da se pričnejo v prvi kontrolni točki P_0 in končajo v zadnji kontrolni točki P_N. Pri tem ostale kontrolne točke definirajo obliko in s tem tudi variacijo krivulje. Krivulja je tako zvezna, ob tem pa omogoča lomljenje z uporabo definiranih večkratnih vrednosti kontrolnih točk. S premikanjem množice kontrolnih točk, ki ležijo na koordinatnem sistemu, sicer ne moremo spreminjati lokalnosti krivulje, če govorimo v območju vrednosti, določenem s stopnjo polinoma. Ob matematični predstavitvi krivulje, ki je definirana z rekurzivno enačbo, lahko izračunamo položaj poljubne točke na krivulji.

3.5 Predstavitev algoritma Bézierovih krivulj

Pri definiranju algoritma si navadno pomagamo z realnostnim modelom, ki ga definiramo ob tem, ko določen problem razstavimo na podprobleme in pridemo do obvladljivih vzorcev. Za izris Bézierove krivulje velja, da je glavna naloga oziroma rezultat izrisana celotna krivulja. To težavo lahko razbijemo tako, da pridemo do problema izračuna neke posamezne točke ki leži na definicijskem območju vrednosti. Ob konkretnem številu točk pridemo do rešitve, preko katere bi s pomočjo interpolacije lahko izrisali dovolj natančno krivuljo, ki bi predstavljala rezultat izrisa. Izračun določene točke je v nalogi implementiran s pomočjo deljenja z naslednjim rekurzivnim algoritmom, ki omogoča izračun točk na krivulji poljubne stopnje.

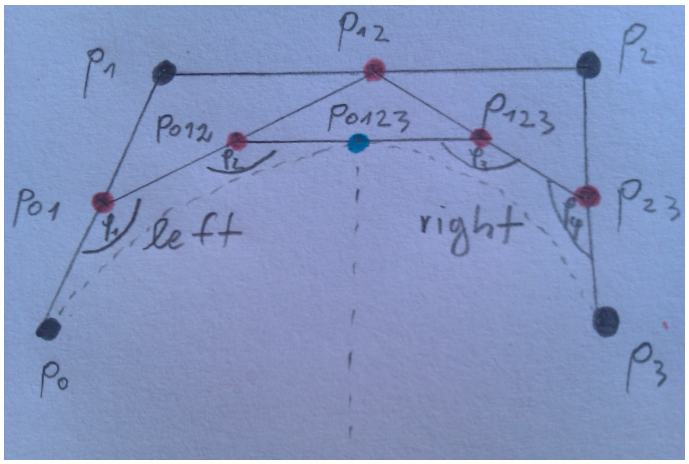
```
1  //temp
2  var c = new Array();
3  //out
4  var d = new Array();
5
6  d.push(array[0]);
7  for(var i = 0; i < array.length-1; i++)
8    c.push(middlePoint(array[i],array[i+1]));
9
10 if(c.length > 1)
11   d = d.concat(middlePoints(c));
12 else
13   d.push(c[0]);
14
15 d.push(array[array.length-1]);
16
17 return d;
18
19 function middlePoint(a, b) {
20     return new Tocka(Math.min(a.x,b.x)+Math.abs(a.x-b.x)/2,
          Math.min(a.y,b.y)+Math.abs(a.y-b.y)/2);
21 }
```

Primer 3.1: Izsek algoritma za izračun točke

3.6 Postopek implementacije algoritma Bézierovih krivulj

Za implementacijo Bézierovih krivulj sicer obstaja več različnih rešitev, ki se razlikujejo po dovršenosti. V nalogi je implementiran primer izrisa Bézierovih krivulj z de Casteljau algoritmom, ki se imenuje po francoskem inženirju, ki je iznašel način izrisa s pomočjo deljenja. Za začetni primer je razvijalcu lažje narediti nekaj primerov na papir, recimo vsaj nekatere scenarije, ki so trivialni, najpogostejši ali pa robni primeri. Za vsak takšen primer je v navadi, da se skupaj z razvito kodo napišejo tudi enotski testi, ki preverjajo delovanje samih metod, modulov ali celotne aplikacije.

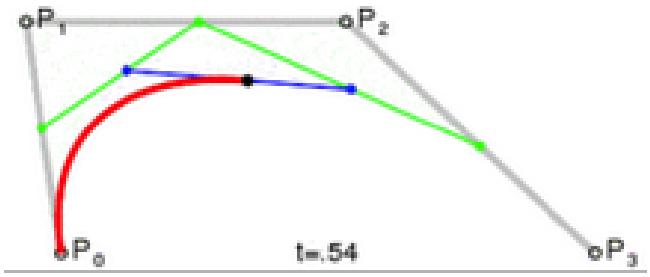
V tem primeru lahko rezultat do določenega merila primerjamo ali logično predvidevamo zaradi lastnosti prikaza rezultata v računalniški grafiki. Pri računalniški grafiki gre predvsem za računsko oziroma matematično predstavitev števil, ki lahko tvorijo različne dimenzije. Dimenzije so najpogosteje predstavljene kot vektorji, matrike, koti ali točke v dve in več dimenzionalnih koordinatnih sistemih.



Slika 3.2: Prikaz izračuna točke na krivulji s pomočjo deljenja na papirju.

Za ta uporabljen primer testnega izrisa ter predstavitve na papir po de Casteljau algoritmu je izvedljiv ročni izris z računanjem ali s pomočjo geo-

metrijskih orodij, kot so šestilo in ravnilo. Primer je izrisan v sliki 3.2. V področju implementacije izrisa se uporabi dvodimenzionalen koordinatni sistem s točkami, pri čemer se zaradi specifike algoritma uporablja razpolavljanje razdalj med točkami v koordinatnem sistemu. Poleg tega je uporabljen matematičen izračun kotov med vektorji, zaradi katerega dobi program lastnost optimiziranega programa. Za Bézierov način bi bili ti izračuni osnova za izgradnjo oziroma izris enostavnejše ali kompleksnejše krivulje.



Slika 3.3: Prikaz izračuna točke na krivulji s pomočjo deljenja.

Za začetek in boljšo predstavitev je na sliki 3.3 prikazan primer deljenja razdalj med posameznimi kontrolnimi točkami. Grafični primer na sliki je prikazan iz primera, kjer se posamezne točke izračunavajo na podlagi vrednosti t, ki navadno teče od vrednosti 0 do 1.

3.7 Robni primeri Bézierovih krivulj

V primeru izrisa Bézierove krivulje ničte stopnje gre za posebnost, pri kateri krivulja ni definirana (slika 3.4). Rešitev je trivialna in vsebuje pri izrisu le množico kontrolnih točk ki jih je definiral uporabnik. Za takšne primere ni potrebe po zapletenem algoritmu, saj bi bil v implementaciji dovolj že en sam pogoj.

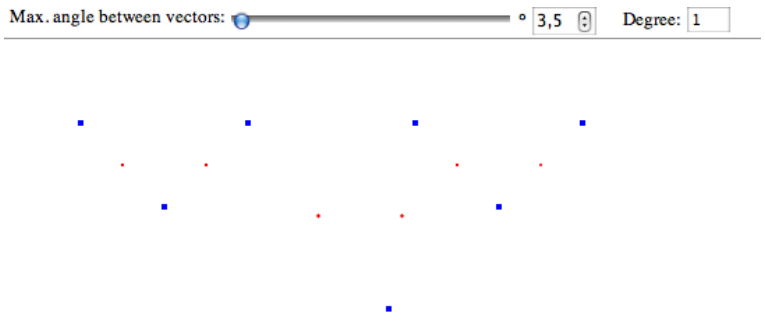
```
1 // Preverimo, ce imamo definiranih dovolj kontrolnih tock in
     stopnjo, vecjo od 0.
```

```
2 if(ctlPoints.length >  stopnja && stopnja > 0){ ...
```

Primer 3.2: Primer omejitve ob primeru izrisa krivulje ničte stopnje

Slika 3.4: Robni primer krivulje ničte stopnje, krivulja ni definirana

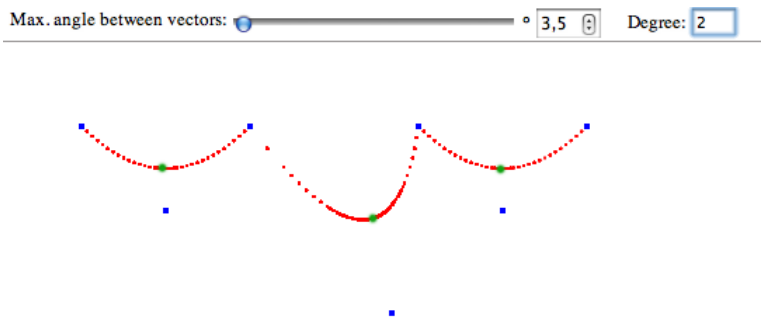
Naslednji primer izrisa krivulje prve stopnje je ravna črta, pri kateri potrebujemo vsaj dve kontrolni točki. Črta tako poteka ravno iz ene proti drugi točki v koordinatnem sistemu. Za takšen primer je enako kot v prejšnjem primeru izris enostaven, pri tem pa za izris ni potrebne kakšne posebne logike. Na sliki 3.5 je potrebno upoštevati lastnost algoritma, ki prikazuje izračunane točke, dejanska krivulja je definirana z linearno interpolacijo med kontrolnimi točkami.



Slika 3.5: Robni primer krivulje prve stopnje, prikaz izračunanih pik, krivulja je definirana z linearnimi povezavami med kontrolnimi točkami.

Na tem primeru se enostavnost izrisovanja konča oziroma bi v kontekstu programa lahko rekli, da slike ni možno izrisati le z mnogimi klici vgrajene funkcije iz programskega jezika Javascript. Pri krivulji druge stopnje so za izris potrebne tri kontrolne točke. Definicijsko območje krivulje se tako prične v prvi točki in poteka po prostoru glede na pozicijo vmesne kontrolne točke in se konča v zadnji kontrolni točki, ki je bila dodana preko vmesnika zadnjikrat. V tem primeru je dejansko že potreben de Casteljau algoritem, preko katerega se izvrši računanje in izris.

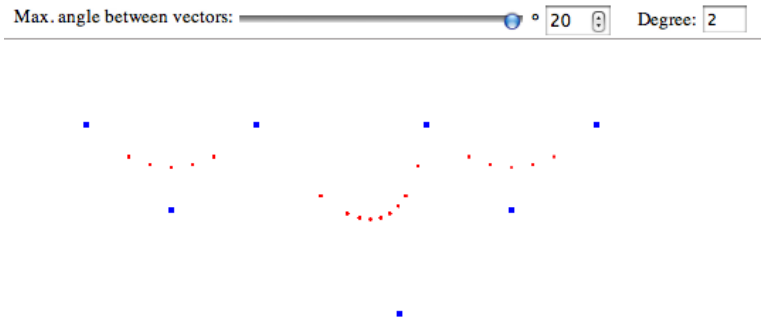
Pri Bézierovi krivulji druge stopnje potrebujemo za izračun točke v 2D prostoru najmanj 3 računske operacije deljenja. Postopek izračuna je dokaj enostaven. Potreben je izračun točk, ki ležijo na polovici razdalj med prvo in drugo ter drugo in tretjo kontrolno točko. Če iz dobljenih točk izračunamo vmesno razdaljo med njima, bi rekli, da dobimo točko, ki dejansko leži na končni krivulji.



Slika 3.6: Primer izrisa krivulje druge stopnje, zaradi narave prikaza algoritma so vidne pike

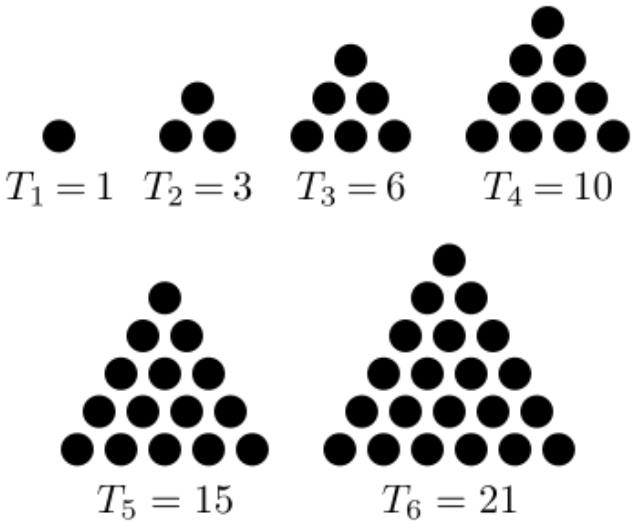
V tem primeru dobimo le eno točko, ki označuje del krivulje, točka je označena za vsak odsek na temenu krivulje, poleg tega če povežemo skrajni točki z novo nastalo točko dobimo približek dejanske krivulje. Ta približek je seveda zelo nenatančen, krivulja pa aproksimira približku dveh ravnih črt. Da bi dobili lepši pogled na krivuljo, moramo postopek razpolavljanja ponavljati mnogokrat - tolikokrat, da novo nastale spremembe pri razpolavljanju niso

več vidne.



Slika 3.7: Prikaz izračunanih koordinat na krivulji po nekaj razpolavljanjih

Program omogoča izris Bézierove krivulje poljubne stopnje, pri tem pa je potrebno upoštevati, da se kompleksnost izrisa povečuje skupaj z naraščajočim številom stopenj. Vsaka nova stopnja pri računanju pomeni nekaj več razpolavljanj, saj je tako krivulja odvisna od večjega števila kontrolnih točk. Temu pojavu rečemo tudi trikotniško število. Na sliki 3.8 je prikazan primer trikotniških števil, s perspektive krivulj je število na sliki njegova stopnja, število izrisanih pik pa število potrebnih deljenj za izračun ene pike na krivulji.



Slika 3.8: Prikaz trikotniških števil, stopnje in števila potrebnih razpolavljanj

Pri tem je potrebno paziti na zalogo vrednosti kontrolnih točk, saj za dovolj veliko stopnjo potrebujemo dovolj definiranih kontrolnih točk. Ob tem je pametno uporabnika opozoriti, da doda nove kontrolne točke. Število kontrolnih točk je definirano kot:

Izrek 3.1 *Za vsako naravno število kontrolnih točk n velja*

$$n \geq stopnja + 1 \tag{3.1}$$

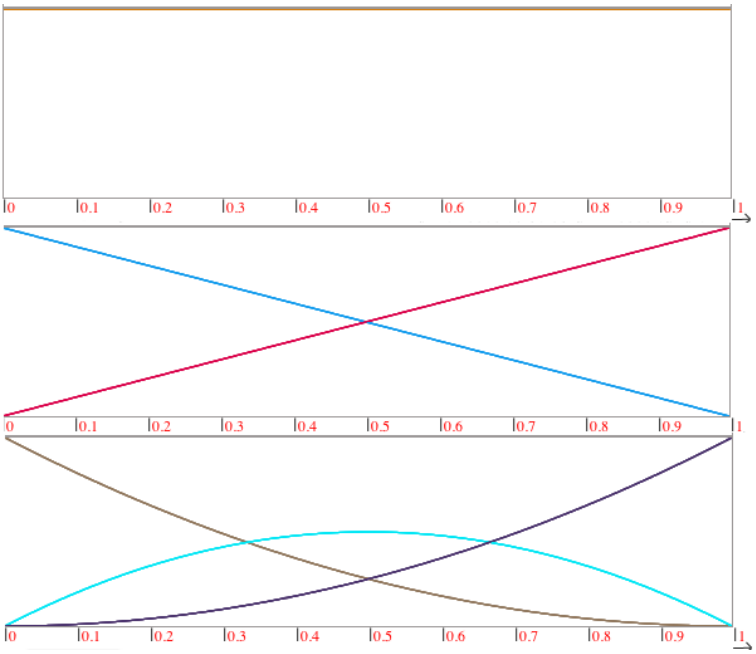
3.8 Bazične funkcije Bézierovih zlepkov

Pri izrisu bazičnih funkcij gre za predstavitev samih komponent, preko katerih je izračunana krivulja. Če se v bazične funkcije vpeljejo kontrolne točke, funkcije pa se nato seštejejo, bi dobili dejanske vrednosti krivulje za vhodne parametre. Bazične funkcije Bézierovih zlepkov so definirane z Bernsteinovimi polinomi stopnje n kot kaže izrek 3.2. Ti polinomi se izračunajo s pomočjo binomskih koeficientov, ti pa so znani kot zelo hitro naraščajoče funkcije.

Izrek 3.2 *Bernsteinovi polinomi stopnje n*

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{3.2}$$

Zaradi tega nastane v programu težava pri računskem delu implementacije. Problem je do neke mere rešen tudi v nalogi. Preizkušeno program deluje do vnesene stopnje 170, pri vrednostih večjih od te stopnje, pa se izpiše napaka. Napaka se sproži pri mejni vrednosti binomskega koeficienta, ki naraste tako visoko, da izračunana vrednost preseže mejno vrednost spremenljivke ki je na voljo za zapis. V programskem jeziku Javascript se tako vrne vrednost*Infinity*, za to vrednost pa program izpiše napako pri računanju.



Slika 3.9: Prikaz bazičnih funkcij za Bézierove krivulje 0., 1. in 2. stopnje

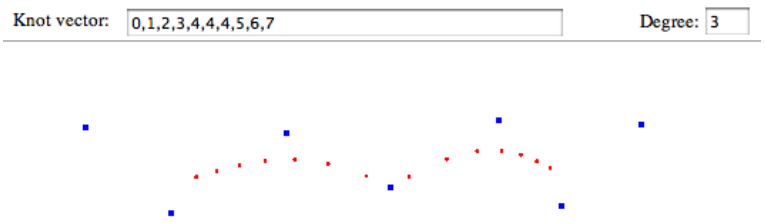
3.9 Postopek implementacije B-zlepkov

Izrisovanje B-zlepkov je nekoliko zahtevnejše od prejšnjih primerov na Bézierovih krivuljah. Kljub vrednostim kontrolnih točk, ki jih uporabnik vnese preko

maske, se tukaj uporabi tudi vektor vozlov, s katerim dobi krivulja preko kontrolnih točk novo odvisnost. Pri izrisu je namreč pomembno, da uporabnik vnese potrebno število vrednosti vozlov, saj se ob nasprotnem primeru krivulja ne izriše v celoti, uporabnika pa bi aplikacija morala obvestiti z rdečim opozorilom: premalo vrednosti vektorja vozlov. Pravilo velja tudi v obratni smeri. Ob prevelikem vektorju vozlov se od uporabnika pričakuje, da vnese v aplikacijo dovolj kontrolnih točk. Ob slednji napaki se bi na aplikaciji prikazalo opozorilo: število vrednosti vozlov je visoko. Ob tem se vektor samodejno prilagodi tako, da prikrito skrajša vektor vozlov in uporabi prvih nekaj vrednosti, odvisno od števila kontrolnih točk in stopnje.

3.10 Vektor vozlov

Vektor vozlov je v nalogi implementiran z vnosnim elementom, v katerega uporabnik vpiše realne vrednosti ki so ločene z vejico. Pri tem je decimalno število predstavljeno z piko. Vektor vozlov nam določa obliko krivulje oziroma množica vrednosti predstavlja vpliv vsake kontrolne točke na krivuljo. Pri tem lahko z večkratnimi vrednostmi določamo večkratnost vozlov, s tem pa nastanejo lomi krivulje kot kaže slika 3.10.



Slika 3.10: Primer večkratnega vozla z vrednostjo 4

3.11 Algoritem za izrisovanje

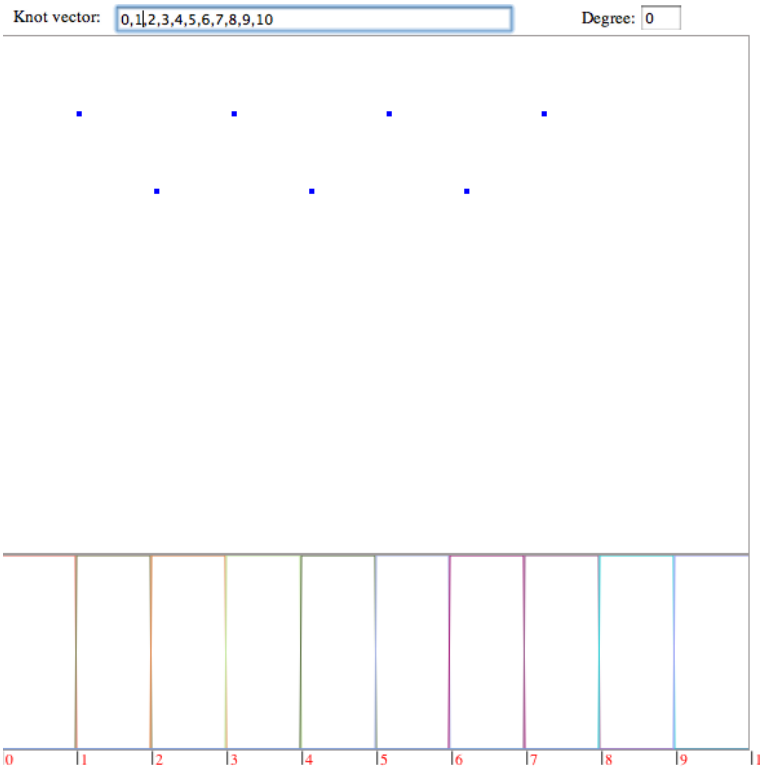
Načinov za izrisovanje B-zlepkov je veliko, v tem delu pa je predstavljen način izrisa z de Boorovim algoritmom, ki je dokaj enostaven za razumevanje. Eden

izmed načinov je tudi direktna implementacija računanja s pomočjo rekurzije, vendar se v tem primeru določeni računski deli izračunavajo mnogokrat, tako da pride do nepotrebnih računskih operacij.

Implementiran algoritem uporabi fiksno število generiranih vrednosti q s korakom, ki je odvisen od končne vrednosti vektorja vozlov. Ta vrednost določa natančnost izračuna točk, ki ležijo na premici. Pri tem večje število pomeni več točk, s tem pa je krivulja bolj natančno interpolirana. Za vsako vrednost t se izračuna večkratnik vozla, pozicije vplivov vozlov ter izhodne vrednosti funkcije. Skozi naraščajočo vrednost t se tako skozi logiko izrišejo vse vrednosti.

3.12 Robni primeri B-zlepkov

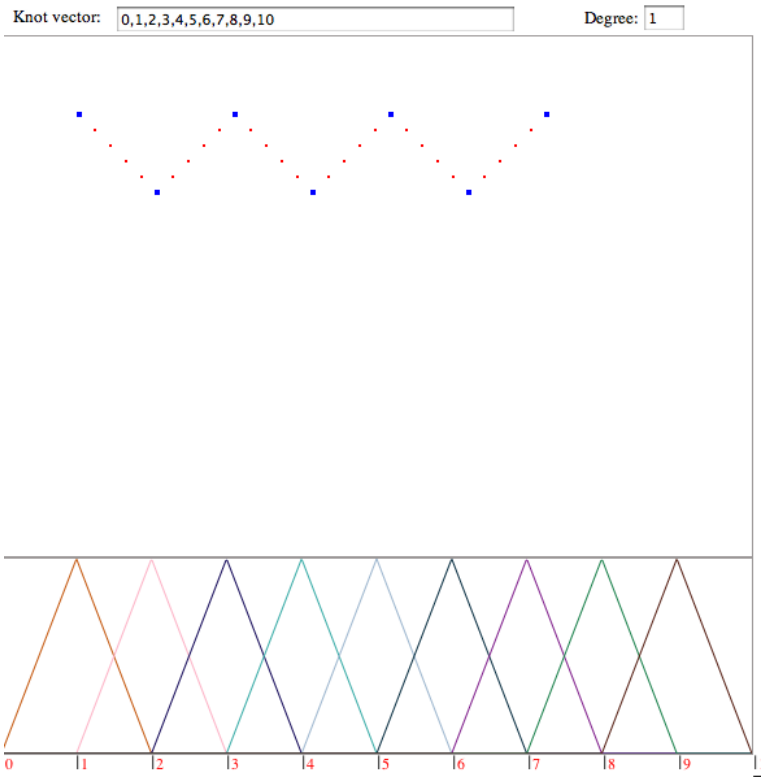
Kot prvi primer B-zlepkov z ničto stopnjo, veljajo podobne lastnosti kot za ničto stopnjo na Bézierovih krivuljah. Za to stopnjo krivulja ni definirana, zaloga izhodnih vrednosti pa je identična množici kontrolnih točk. V programu tako ni videti izrisanih točk, saj pride do neizpolnjenega pogoja, ki omogoča izris že na začetku algoritma. Prikaz takšnega odziva je na sliki 3.11, poleg tega so izrisane bazične funkcije v vrednostih kontrolnih točk.



Slika 3.11: Robni primer klica algoritma B-zlepkov za stopnjo 0

Pri zlepkih prve stopnje gre za nekoliko drugačen primer, pri katerem je razvidno, da krivulja ni naravnega videza, temveč je lomljena na vsaki kontrolni točki. Za razliko od prejšnjega primera velja, da so izračunane točke ne glede na pozicijo kontrolnih točk vedno pozicionirane na daljici, ki povezuje po dve kontrolni točki. Izrisana krivulja je tako identiteta linearne interpolacije med kontrolnimi točkami, podobno kot na primeru Bézierovih krivulj.

Implementiran algoritem ni dovolj optimiziran, kar je vidno s pomočjo množice točk, ki se izrišejo in s tem nakazujejo potek krivulje, kot je razvidno na sliki 3.12. Pravilno izpeljan algoritem je videti na primeru Bézierovih krivulj, na katerem je na intervalu med kontrolnimi točkami izrisana le po ena točka (slika 3.5).



Slika 3.12: Primer B-zlepka prve stopnje in povezovalne funkcije z danim vektorjem vozlov

Pri primeru prve stopnje na sliki 3.12 je videti, da bazične funkcije potekajo linearno med določenimi odseki. V konceptu izrisa to pomeni, da je vpliv vsake kontrolne točke zelo majhen, poleg tega imata na vsakem odseku intervala vpliv največ dve kontrolni točki.

3.13 B-zlepki druge stopnje

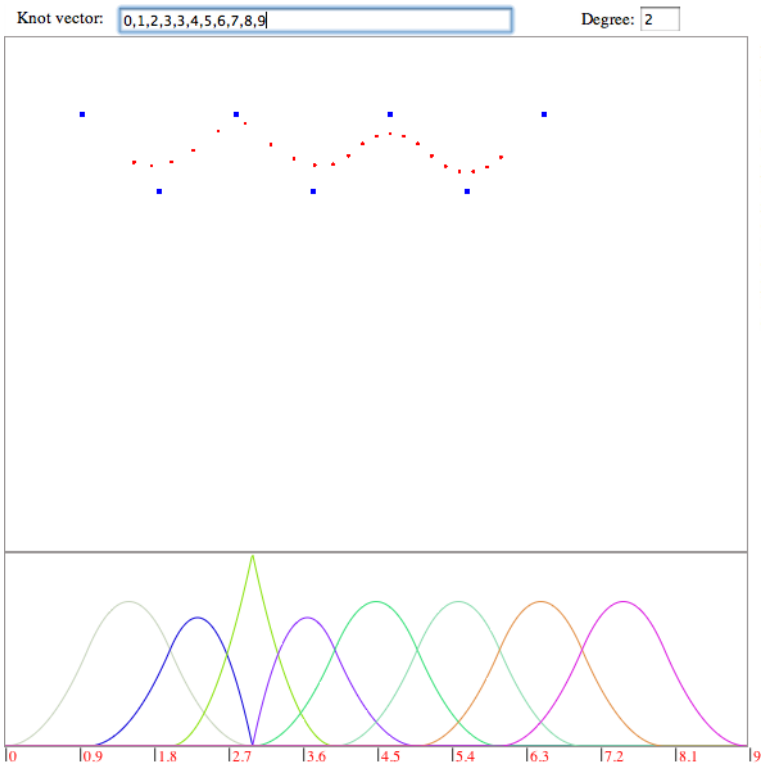
Za kvadratične in večje stopnje pridobi krivulja svojo pravo obliko. Glede na kontrolne točke se pri izračunu upošteva vpliv dodatnih vozlov, ki so indirektno povezani z vrednostmi kontrolnih točk za izračun izhodne vrednosti. Ob primeru krivulje druge stopnje velja, da njen vpliv seže dlje preko intervala kot pri manjših stopnjah. V konceptu implementacije to pomeni dvojni

izračun vpliva po rekurzivni metodi. Ob tem obstajajo primeri, pri katerih je vpliv manjši, predvsem gre za mnogokratnost kontrolnih točk, ki se določijo preko vektorja vozlov. Sledi primer algoritma, ki določa mnogokratnost vozlov.

```
1 //define variable
2 var multiplicity;
3 for(var lastIndex = 0; u >= knots[lastIndex] && lastIndex <
     knots.length-1; lastIndex++);
4 for(var index = lastIndex; knots[index-1] == u; index--);
5
6 multiplicity = lastIndex - index;
```

Primer 3.3: Izsek algoritma za ugotavljanje mnogokratnosti

Za vozle, ki imajo toliko enakih vrednosti, kolikor je vrednost stopnje, velja, da je vrednost funkcije na vplivu intervala določena z vrednostjo ene same kontrolne točke. Za vrednosti vozlov, ki jih definiramo znotraj območja, pa velja, da se ob stopnji enakih vrednostih tvori zlepek dveh krivulj. Krivulja tako na pogled ni več naravna, temveč lomljena.



Slika 3.13: Prikaz multipliciranih vrednosti kontrolnih točk s pomočjo vektorja vozlov

Odlično vlogo ima v tem primeru prikaz bazičnih funkcij, ki prikazuje vpliv vektorja vozlov na podano stopnjo. Lepo je razvidno, da ima ob vrednosti spremenljivke t=3 popolni vpliv tretja kontrolna točka. Točka je tako del krivulje, s premikom točke pa ji na enako lego sledi tudi krivulja.

Pri B-zlepkih druge stopnje je potrebno omeniti še nekoliko hitrejši način izrisovanja s pomočjo vrednosti matrik, ki so izpeljane iz rekurzivne enačbe B-zlepkov. Računanje izhodnih vrednosti je tako enostavnejše in hitrejše, primer pa je uporaben le za določene vrednosti vektorja vozlov, ki je linearen. Metoda je uporabna za specifično uporabo, kjer uporabnik odvečnih odvisnosti, kot je vektor vozlov ne potrebuje.

Izrek 3.3 *Enačba za izračun krivulje druge stopnje po definirani matriki vre-*

dnosti

$$c_i(u) = \frac{1}{2}[u^2 \quad u \quad 1] \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{i-1} \\ r_i \\ r_{i+1} \end{bmatrix} \tag{3.3}$$

Sprva je pričakovano, da bo rezultat specifičnega algoritma precej boljši kot pri splošnem primeru. Po izvedbi kaže, da statistično zaradi nepričakovanih rezultatov pri številu vzorcev n = 100, teorijo ovržemo.

- Vzorčenje z 10 točkami

  Integrated algorithm execution time: 0.47ms.

  Custom algorithm execution time: 0.66ms.

- Vzorčenje s 50 točkami

  Integrated algorithm execution time: 5.37ms.

  Custom algorithm execution time: 7.6ms.

- Vzorčenje s 500 točkami

  Integrated algorithm execution time: 366.95ms.

  Custom algorithm execution time: 663.85ms.

Gledano s stališča algoritma je specifična implementacija enostavnejša za izvedbo ob pogoju, da poznamo enačbe za izračun glede na stopnjo po matriki. Očitno je, da v programskem jeziku aritmetične operacije niso tako nedolžna naloga, saj v tem primeru porabijo veliko sistemskih virov. Kot zanimivost so prikazani rezultati časovnih razlik med izrisom krivulje s pomočjo de Boorovega algoritma ter izrisom s pomočjo računanja rezultata preko predefinirane matrike vrednosti za kvadratične krivulje. Iz rezultatov je mogoče sklepati, da je de Boorov algoritem očitno hitrejši, s povečevanjem števila točk pa se ta rezultat še izboljšuje.

3.14 B-zlepki tretje stopnje

Tako kot pri zlepku kvadratne stopnje, veljajo tudi pri stopnji več podobne lastnosti. Za izris B-zlepka kubične stopnje je potrebno definirati dodatno vrednost vektorja vozlov ob spremembi izrisa iz kvadratične oblike. Za generiranje lomljene krivulje velja podobno, le da je z vrednostmi kontrolnih točk potrebno definirati eno identično vrednost več kot pri primeru krivulje stopnje manj. Kakor je v prejšnjem primeru krivulje druge stopnje omenjena časovna odvisnost algoritmov, je tudi v tem delu implementiran algoritem za izračun krivulje kubične stopnje po preddefinirani matriki.

Izrek 3.4 *Enačba za izračun krivulje tretje stopnje po definirani matriki vrednosti*

$$c_i(u) = \frac{1}{6}[u^3 \quad u^2 \quad u \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{i-1} \\ r_i \\ r_{i+1} \\ r_{i+2} \end{bmatrix} \tag{3.4}$$

S pomočjo programa so izmerjene vrednosti v milisekundah, ki jih program potrebuje, da obdela izračun ene krivulje. Postopek implementacije je spremenjen glede na novo stopnjo, privzeti algoritem pa pri tem ostane enak. Naslednje vrednosti prikazujejo časovno potratnost algoritma po statističnem številu vzorcev n=100.

- Vzorčenje z 10 točkami

  Integrated algorithm execution time: 0.57ms.

  Custom algorithm execution time: 0.84ms.

- Vzorčenje s 50 točkami

  Integrated algorithm execution time: 4.5ms.

  Custom algorithm execution time: 8.3ms.

- Vzorčenje s 500 točkami

Integrated algorithm execution time: 260.34ms.

Custom algorithm execution time: 670.81ms.

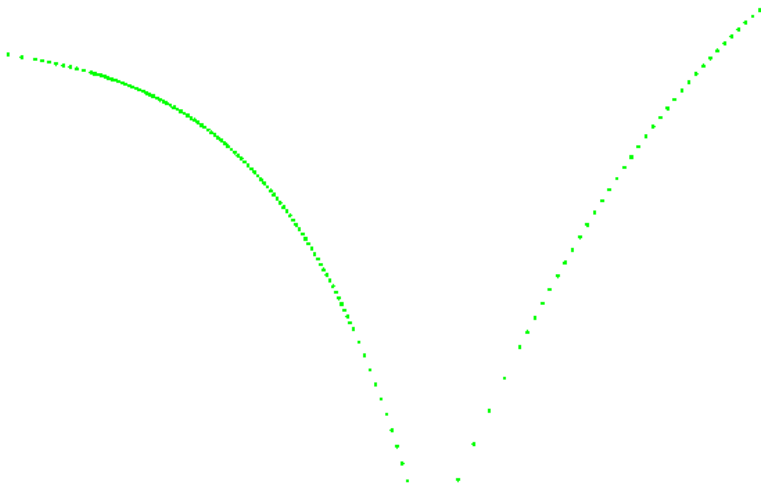
Kakor se vidi razlika med posameznimi rezultati glede na število izračunanih točk na primeru druge stopnje, so razlike na primeru tretje stopnje pri 500 točkah še bolj očitne.

3.15 B-zlepki višjih stopenj

Zlepki višjih stopenj so podobno definirani kakor zlepki kubičnih stopenj, zato jih ni potrebno posebej omenjati glede na posamezno stopnjo. Za vsako stopnjo več veljajo enake razlike kot med krivuljama kvadratne in kubične stopnje.

3.16 Funkcionalnost skaliranja prikazanih krivulj

Pri implementaciji skaliranja prikazane slike za določen faktor se uporablja trivialen premik samih točk, ki določajo krivuljo. Pri tem ostane osnovna podatkovna struktura nedotaknjena, razlika pa se izračuna glede na faktor ob prikazu. Programska koda omogoča faktor povečave od 0 odstotkov naprej, pri tem pomeni faktor 100 odstotkov realno velikost slike. Implementacija skaliranja se navadno opravlja s pomočjo matrik, preko katerih se izračunajo skalirane vrednosti. V primeru naloge se vrednosti ob skaliranju izračunajo tako, da se vse točke direktno pomnožijo s faktorjem skaliranja. Pri tem je potrebno upoštevati miškin kazalec, saj je tudi ta eden izmed parametrov pri kalkulaciji.



Slika 3.14: Skalacija krivulje.

Sam algoritem je nekoliko zapleten, poleg tega bi bilo mogoče ubrati drugo pot in uporabiti drugačne pristope, ki uporabljajo matrike. Glavni pogoj pri tem je, da se povečava izvrši v točko, na katero trenutno kaže miškin kazalec. Pri tem ugotovimo, da se ob premiku kazalca trenutni kontekst prikaza ne izgubi. Za to je potrebno določene parametre v trenutnem kontekstu tudi shraniti v podatkovno strukturo.

V kolikor uporabnik ročno spremeni parameter povečave, se slika samodejno povrne v prvotno velikost. Smiselno je, da se programsko omeji tudi zgornja meja povečave, saj ni mišljeno, da uporabnik povečuje krivuljo v neskončnost. Zaradi takšnega početja bi lahko prišlo do zrušitve programa. Ta del naloge bi lahko bil nekoliko bolj dovršen oziroma modularen. Funkcionalnost je integrirana v sam prikaz, kar pa ni dobra rešitev programske opreme, res pa je, da za takšen način ne potrebujemo posebne podatkovne strukture za shranjevanje vrednosti samih točk na krivulji.

S trenutno implementacijo se krivulja izračunava v celoti ne glede na spremenjeni zlepek, zato je nekaj o tem omenjeno tudi v poglavju optimizacije. Preizkušeno se program ob mnogih zlepkih prične odzivati počasi, odvisno od vrednosti nastavljenega kota, ki drastično vpliva na odzivnost aplikacije.

Za Bézierove krivulje velja omeniti, da se natančnost prikaza spreminja glede na faktor skalacije, zato velja preizkusiti tudi ta del funkcionalnosti programa. Skozi vrtenje miškinega kolesa opazimo, da se počasi pojavljajo nove točke na krivulji, ki bolje interpolirajo sam lik.

Poglavje 4

Optimizacija in učinkovitost

4.1 Optimizacija algoritma za izris

Pri optimizaciji algoritma je potrebno sprva pomisliti na same lastnosti danih podatkov, mask, oziroma v primeru te naloge, krivulje ter njenega prikaza. Ena izmed lastnosti krivulj je oblika, kar pomeni, da bi lahko glede na obliko krivulje optimizirali določen algoritem preko izračuna za izris. V praksi bi tako med tremi kontrolnimi točkami, ki ležijo na isti premici, narisali le eno črto brez kompleksnih operacij, med tem ko bi brez optimizirane kode trošili procesorski čas, rezultat izrisa po povezavi z črtami pa bi bil praktično enak.

Slika 4.1: Razlika v številu pik po optimizaciji (zgoraj) in pred optimizacijo (spodaj)

Zgornji primer stremi k ekstremnosti, saj se v večini primerov določen scenarij ne zgodi. V večini primerov uporabe bi se vmesna kontrolna točka nahajala nekje na naključnih vrednostih x in y koordinatnega sistema. Pri

tem pa je še vedno mogoče izkoristiti omenjeno izboljšavo v primeru, da bolj krive dele vzorčimo večkrat kot pa bolj ravne. Najenostavnejša rešitev pri tem je računanje kotov med vektorji, ki povezujejo dve točki na krivulji.

Postopek izračuna novih točk, ki so na premici, se ponavlja do določene vrednosti kota φ, ki je podan kot parameter algoritma. Vneseni kot je mogoče spreminjati od 0 stopinj naprej in je v implementaciji izpeljan kot inverz kotne funkcije kosinus kot kaže primer iz Bézierovih krivulj.

```
1  //array - parameter mnozice tock
2  function checkAngle(array){
3    //define
4    var angle = 0;
5    var A, B, arccos, fi;
6    for(var i = 0; i < array.length-2; i++) {
7      //razlike med koordinatami dvojic tock
8      A = new Tocka(array[i+1].x-array[i].x,array[i+1].y-array[
         i].y);
9      B = new Tocka(array[i+2].x-array[i+1].x,array[i+2].y-
         array[i+1].y);
10     //izracun kota
11     arccos = ((A.x*B.x)+(A.y*B.y))/((Math.sqrt(Math.pow(A.x
         ,2)+Math.pow(A.y,2)))*(Math.sqrt(Math.pow(B.x,2)+Math.
         pow(B.y,2))));
12     //arccos
13     fi = Math.acos(arccos)*180/Math.PI;
14     // ce kot vecji od trenutnega najvecjega
15     if( fi > angle )
16       angle = fi;
17   }
18   //vrni najvecji kot med posameznimi tockami
19   return angle;
20 }
```

Primer 4.1: Izsek algoritma za optimizacijo izrisa glede na kot med vektorji

De Casteljauov algoritem za izračun Bézierove krivulje je definiran rekurzivno, zato je rekurziven tudi računalniški algoritem v programskem jeziku Javascript. Rekurzija je tipičen način programiranja, kjer se izvede določen

kos kode, ki je povezan v funkcijo, pri tem pa se v izvedbi kliče ponovno enak primerek te funkcije. Tipičen primer takšnega algoritma je izračun faktorele. Takšne funkcije imajo osnovno značilnost, da se lahko izvajajo v neskončnost, kar pomeni, da v programu nebi dobili končne rešitve. Zato se navadno v takšne funkcije določi eden izmed pogojev kot je „kliči funkcijo, dokler je parameter večji od 1“ oziroma v primeru krivulj „kliči funkcijo, dokler je izračunan kot večji do vnesenega“. To še vedno lahko predstavlja težave, če je algoritem napačen ali pa so začetni parametri funkcije napačni. Poleg tega je potreben dodaten pogoj, pri katerem velja, da se funkcija s končnim pogojem kliče v časovno primerljivem času. Tako bi bilo nespametno klicati funkcijo s kotom, ki je enak 0, saj bodo vrednosti kota prehajale proti vrednosti 0, teoretično pa je ne bodo nikoli dosegle.

4.2 Optimiziran način izrisa B-zlepkov

Natanko tako kot je definirana optimizacija krivulje pri Bézieru, je možna izboljšava tudi na področju B-zlepkov. Zelo dobra rešitev je izris točk, ki ležijo na krivulji glede na določen kot med vektorji, vendar v tem delu ni implementiran. Dovolj pomembno je opozoriti na časovno potratnost izrisa, saj je z druge strani potrebno zadostiti potrebam uporabnika ter mu ponuditi odzivno aplikacijo. To je namreč zelo težka naloga, saj razvijalec načeloma s težavo ugiba kdo in s kakšno napravo bo imel dostop do aplikacije.

4.3 Optimizacija s pomočjo odziva zunanjih dejavnikov

S trenutno informacijsko tehnologijo lahko pričakujemo dostop iz zmogljivega osebnega računalnika, počasnejšega prenosnika ali še bolj okrnjene tablice oziroma prenosnega telefona. Na tem področju se odpira nekaj novih možnosti optimizacije, in sicer kakovosti izrisa glede na dano strojno opremo. Razvijalec mora tako nekako pridobiti informacije o napravi, njeni zmogljivosti

in programski opremi. Trivialna rešitev je realizacija z merjenjem časovne zahtevnosti. S programskim jezikom Javascript, ki vsebuje vgrajeno funkcijo Date, se dinamično računa čas glede na zahtevano enoto, podobno kakor je bilo implementirano v delu testiranja učinkovitosti B-zlepkov. Ta enota je v primeru B-zlepkov lahko izračun ene vrednosti točke na krivulji. Glede na časovno odvisnost koraka generiranja vhodnih vrednosti se ta korak prilagaja časovni potratnosti ene enote. Dan je primer začetne vrednosti koraka k, časovnih vrednosti t, ki določa maksimalen čas izračuna krivulje.

```
1  var k = 50; //zacetno 50 tock na krivulji
2  var Ts, Te, Tmax=10; //10ms za izracun celotne krivulje
3  for(...) {
4    Ts = new Date();
5    izracunajTocko();
6    Te = new Date();
7
8    //ce cas izrisa vecji od pricakovanega zmanjsamo vzorcenje
9    if((Te-Ts) > Tmax/k)
10     k*0.9;
11   else //drugace ga povecamo
12     k*1.1;
13 }
```

Primer 4.2: Prototip algoritma optimizacije glede na strojno opremo

Poglavje 5

Sklepne ugotovitve

5.1 Možne izboljšave

Programska oprema je produkt, ki je dovršen do določene mere. Če gre pri tem za komercialne primere, velja načelo konkurenčnosti, zato se programska oprema izvede do te mere, da ponuja čim boljšo rešitev ob najkrajšem času in z najmanjšimi stroški. V primeru izvedene naloge gre za določene funkcionalnosti, ki bi lahko bile izboljšane s tem, da bi uporabniku bolj konkretno nakazale podrobnosti izvrševanja. Konkreten primer izboljšave je prikazan v prejšnjem poglavju z optimizacijo glede na strojno opremo.
Priročno bi bilo imeti enak algoritem, ki izrisuje točke glede na kot tudi na B-zlepkih, ampak ne gre za aplikacijo, katera bi bila uporabljana od kjer koli s kakršno koli napravo, zato je takšen primer omenjen le kot vzorec. Ena izmed izboljšav bi bila možna implementacija kurzorja, ki bi prikazoval indeks na krivulji glede na indeks, ki je z miško označen na prikazu bazičnih funkcij.

5.2 Ugotovitve

Skozi delo pridemo do ugotovitev, da so krivulje zelo širok pojem, saj lahko na njih gledamo iz več različnih vidikov. Krivulje lahko razdelimo na različne tipe, ki se razlikujejo po določenih lastnostih, kar smo ugotovili tudi v izve-

deni nalogi. Večje ugotovitve se bodo pokazale še skozi čas, saj je primarni namen dela pomoč pri izobraževanju. Rezultati bodo vidni v prihodnjem času, ko bodo uporabniki pričeli z uporabo programa. Sklepno ugotovimo, da ima takšen prikaz drugačen pogled na teoretično znanje, zato je veliko bolj primeren za uporabo poleg predavanj ali interpretaciji ob drugih vrstah učil.

Literatura

[1] N. Guid, *Računalniška grafika*, Maribor, 2001, str. 140–166.

[2] H. Prautzsch, W. Boehm, M. Paluszny, *Bézier and B-Spline Techniques*, Springer, 2002.

[3] (2012) S. Galliani, Primer izrisa krivulj s pomočjo Javascript. Dostopno na spletnem naslovu:
http://quico.dyne.org/ kysucix/geox.html

[4] (2012) M. Hoefer, Java bezier curve demo. Dostopno na spletnem naslovu:
http://www.math.ucla.edu/ baker/java/hoefer/Bezier.htm

[5] (2012) "Bezier Curve" and "Curve through points" drawing Demo. Dostopno na spletnem naslovu:
http://jsdraw2d.jsfiction.com/demo/curvesbezier.htm

[6] (2012) De Boor's Algorithm. Dostopno na spletnem naslovu:
http://www.cs.mtu.edu/ shene/COURSES/cs3621/NOTES/spline/de-Boor.html