Krivulje u računalnoj grafici

Oklen, Mikela

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:897274

Rights / Prava: Attribution 4.0 International/Imenovanje 4.0 međunarodna

Download date / Datum preuzimanja: **2024-05-18**

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Repository / Repozitorij:

Repository of the University of Rijeka, Faculty of Informatics and Digital Technologies - INFORI Repository

zir.nsk.hr

DIGITALNA KNJIŽNICA

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Sveučilišni prijediplomski studij informatike

Mikela Oklen

Krivulje u računalnoj grafici

Završni rad

Mentori:

doc. dr. sc. Vedran Miletić

doc. dr. sc. Sanda Bujačić Babić

Rijeka, rujan 2023.

SAŽETAK

Krivulje u računalnoj grafici savršeno su pomagalo za prikazivanje zaobljenih i 'prirodnih' oblika u vektorskom okruženju. Od svojih prvih pojavljivanja 60-tih godina u automobilskoj industriji, pronađene su još mnoge njihove napredne i korisne uporabe u animaciji, izradi fontova, robotici i drugdje.

U ovom završnom radu bavimo se problemom prikazivanja i crtanja zaobljenih oblika u računalnoj grafici. U modernoj računalnoj grafici realističan prikaz je jako bitan te je crtanje glatkih krivulja jedan od čestih zadataka. Kao jedno od rješenja koriste se Bézierove krivulje koje će biti detaljno opisane u radu. Također, u radu opisujemo Bernsteinove težinske funkcije i polinome koje koristimo za aproksimaciju Bézierove krivulje.

Implementacija algoritma aproksimacije Bézierove krivulje izvedena je u programskom jeziku C++ , a za vizualizaciju korišten je API OpenGL. Literatura koja je korištena kod implementacije algoritma bila je dokumentacija OpenGL-a i popratnih biblioteka.

KLJUČNE RIJEČI

Krivulje, Bézierove krivulje, Bernsteinovi polinomi, De Casteljauov algoritam

Sadržaj:

1. UVOD 1

2. BÉZIEROVE KRIVULJE 2

2.1. APROKSIMACIJA BÉZIEROVIM KRIVULJAMA 3

2.1.1. Bernsteinove težinske funkcije (Bernsteinovi polinomi) 3

2.1.2. Svojstva Bernsteinovih težinskih funkcija 5

2.1.3. Bézierove krivulje 7

De Casteljauov algoritam 7

3. ODABIR OPTIMALNOG ALGORITMA 13

4. PROJEKTNI ZADATAK 14

4.1. GLAVNA IDEJA PROJEKTA 14

4.2. IMPLEMENTACIJA ALGORITMA 14

4.3. PRIKAZIVANJE BÉZIEROVE KRIVULJE 17

4.3.1. Prikaz točaka kontrolnog poligona 19

4.3.2. Prikaz krivulje 20

4.4. REZULTAT PROJEKTA 22

4.5. BUDUĆNOST PROJEKTA 23

5. ZAKLJUČAK 24

6. LITERATURA 25

7. POPIS SLIKA 26

1. UVOD

Jedan od čestih zadataka u računalnoj grafici je određivanje glatke krivulje za zadani niz točaka (1). U većini slučajeva krivulja aproksimira prikaz oblika iz stvarnog svijeta koji inače nema matematički prikaz ili čiji je matematički prikaz nepoznat ili prekompliciran za daljnju obradu (2). Bézierove krivulje nude nam jedno od rješenja ovog problema. Radi se o parametarskim krivuljama koje su dobivene različitim matematičkim metodama koje daju glatku krivulju.

Kada spominjemo glatku krivulju ne mislimo samo na glatkoću koju vidimo na ekranu, glatka krivulja se u matematici kao pojam koristi i kada se želi naglasiti da je krivulja neprekidna i da se u svakoj njenoj točki može postaviti tangenta na tu krivulju, odnosno da je funkcija derivabila u svakoj svojoj točki domene. Glatke krivulje su najoptimalnije za računalnu implementaciju jer smo sigurni da neće dolaziti do fatalnih grešaka prilikom izračuna numeričkih vrijednosti u svim točkama krivulje.

U ovom radu ćemo detaljnije promatrati metode određivanja Bézierovih krivulja, a jednu od metoda kojom smo ih opisali u radu implementirat ćemo u funkcionalni program.

Bézierove krivulje također se koriste u animaciji, dizajnu korisničkog sučelja te kod izglađivanja putanje kursora u sučeljima koje kontroliramo pogledom (2). Animatori koriste Bézierove krivulje u jednom od 12 načela animacije: *„slow in and slow out“*. Načelo sugerira da, kako bi pokret izgledao prirodno, treba mu vremena da ubrza i uspori, odnosno da na početku i kraju pokreta animator dodaje više *frame*-ova[1] kako bi naglasio postupno ubrzavanje i usporavanje radnje animiranog objekta (3).

Ovo načelo animacije također se primjenjuje u robotici gdje se postepeno ubrzavanje i usporavanje implementira kako bi pokret bio što fluidniji. Uzmimo kao primjer industrijsku robotsku ruku: kako bi izbjegli nepotrebno trošenje materijala od kojih je ruka napravljena, njezini pokreti su programirani s posebnom pažnjom da se poštuje ovo pravilo animacije (2).

[1] *frame* – hrv. *okvir* je jedna od mnogih nepokretnih slika koje čine kompletnu pokretnu sliku u animaciji, filmu ili videu.

2. BÉZIEROVE KRIVULJE

Bézierove krivulje ime su dobile po francuskom inžinjeru Pierru Bézieru. Bézier je 1960-tih radio na primjeni interpolacijskih krivulja u sklopu razvoja računalno podržanog oblikovanja (CAD[2]) za potrebe automobilske industrije, odnosno za tvrtku Renault u kojoj je radio. Bézier je svoju metodu patentirao 1962. godine, no nije bio jedini koji je radio na tom problemu (5).

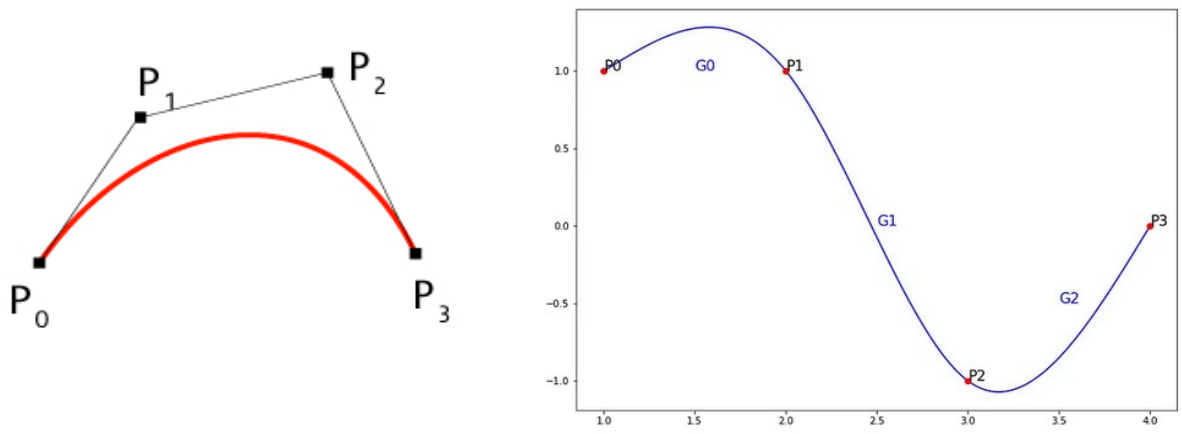
Paul de Casteljau, francuski matematičar i fizičar koji je radio u konkurentskoj tvrtki Citroën, također dolazi do jednog od mogućih rješenja za prikaz krivulja čak tri godine ranije od Béziera, 1959., ali je njegova metoda patentirana tek 1980-ih. Robert Forest je dokazao 1970. godine da se radi o istim krivuljama (1).

Bézier i De Casteljau ovakve su krivulje koristili u dizajniranju krivulja karoserija automobila, no njihova se namjena od 1960-tih do danas proširila na razna područja: od dizajniranja računalnih fontova do animacije (2).

Postoje dvije matematičke metode prikazivanja Bézierovih krivulja: prva se temelji na Bézierovim težinskim funkcijama, a druga na Bernsteinovim težinskim funkcijama (a dobiva se kao rezultat primjene De Casteljauovog algoritma). Bernsteinove težinske funkcije pogodnije su za praktičnu primjenu te smo njih koristili u praktičnom dijelu ovog projekta (1).

Ovisno o tome koji su nam parametri poznati, Bézierove krivulje možemo računati na dva načina: aproksimacijom ili interpolacijom krivulje. Ukoliko su nam poznati vrhovi kontrolnog poligona, tada se radi o aproksimaciji Bézierove krivulje, no ukoliko su nam umjesto vrhova kontrolnog poligona poznati neki drugi parametri krivulje (niz točaka kroz koje krivulja prolazi, funkcijske vrijednosti krivulje i tangenata u pojedinim točkama i sl.), tada se radi o interpolaciji Bézierove krivulje (*vidi sliku 1.*) (1).

[2] CAD - (akronim od engl. *Computer Aided Design:* oblikovanje s pomoću računala), primjena računala za dizajniranje, projektiranje, konstruiranje, vizualizaciju budućega tehničkog objekta, izradu dokumentacije za proizvodnju, planiranje proizvodnje, proračun utroška materijala i drugoga (4).



Slika 1. Aproksimacija i interpolacija Bézierove krivulje

Neovisno kojem načinu računanja pristupamo, rezultat je uvijek ista vrsta krivulje: Bézierova krivulja (1). U ovom radu detaljnije ćemo razmatrati aproksimacijske algoritme crtanja Bézierovih krivulja.

2.1. APROKSIMACIJA BÉZIEROVIM KRIVULJAMA

Aproksimacija je vrlo čest pojam u numeričkoj analizi, a obilježava postupak pronalaženja funkcije koja približno opisuje (aproksimira) zadani konačni skup točaka ili neku drugu funkciju (6). U okviru zadatka završnog rada aproksimirat ćemo vrhove kontrolnog poligona kako bi dobili Bézierovu krivulju.

Aproksimaciju Bézierovih krivulja možemo vršiti pomoću dviju matematičkih metoda: Bézierovog i De Casteljauovog algoritma koji se koristi Bernsteinovim težinskim funkcijama. Pobliže ćemo proučiti ove metode, a projektni zadatak opisan u nastavku ovog rada temeljen je na implementaciji De Casteljauovog algoritma.

2.1.1. Bernsteinove težinske funkcije (Bernsteinovi polinomi)

Polinom je definiran kao matematička funkcija s jednom ili više varijabli koja se može zapisati kao linearna kombinacija umnožaka njihovih potencija, odnosno kao zbroj monoma sastavljenih od umnožaka koeficijenta i kombinacija potencija svake varijable (7).

Polinom u matematičkom zapisu prikazujemo na sljedeći način:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

gdje su $a_0, a_1, a_2, \ldots, a_n$ elementi nekog skupa, a najčešće se radi o skupu realnih brojeva i vrijedi $a_n \neq 0$. a_0 se naziva slobodni koeficijent, a broj a_n vodeći koeficijent. $n \in \mathbb{N}$ određuje stupanj polinoma.

Polinome kraće možemo zapisati:

$$P(x) = \sum_{i=0}^{n} a_n x^i. \qquad \textbf{(1)}$$

Sergei Natanovich Bernstein bio je sovjetski matematičar koji je 1912. godine kratkom objavom u časopisu *Communications* predstavio konstruktivan dokaz Weierstrassovog teorema. U tom članku prvi upotrebljava izraz „Bernsteinovi bazni polinomi“. Godinu kasnije objavljuje novu doktorsku disertaciju koja se temelji na polinomnoj aproksimaciji funkcije.

Definicija 2.1. Neka je $t \in [0,1]$. Tada je i-ti Bernsteinov bazni polinom stupnja n definiran kao

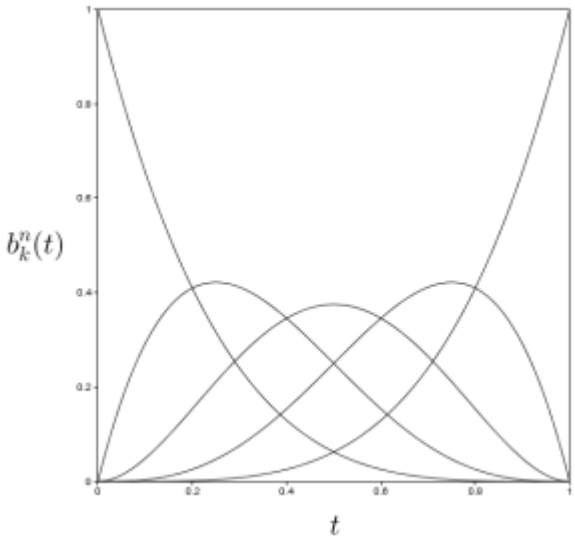
$$b_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0,1,\ldots,n,$$

gdje je $0 \leq t \leq 1$ i gdje je $\binom{n}{i}$ binomni koeficijent, odnosno $\binom{n}{i} = \frac{n!}{i!(n-i)!}$.

Linearnu kombinaciju Bernsteinovih baznih polinoma

$$B_n(t) = \sum_{i=0}^{n} \alpha_i b_i^n(t), \qquad \textbf{(2)}$$

zovemo Bernsteinov polinom stupnja n, gdje su α_i Bernsteinovi koeficijenti.



Slika 2. Bernsteinov polinom stupnja n=4

2.1.2. Svojstva Bernsteinovih težinskih funkcija

U ovome dijelu rada navesti ćemo par ključnih svojstava Bernsteinovih težinskih funkcija, koja ćemo koristiti u daljnjem tekstu.

Nenegativnost

Bernsteinove težinske funkcije su nenegativne na intervalu [0,1]. Za svaki $i = 0,1,\dots,n$ i $t \in [0,1]$ vrijedi $b_i^n(t) \geq 0$.

Zbog toga što je $t \in [0,1]$, vrijedi da je onda $t^i \geq 0$ i $(1-t)^i \geq 0$. Iz toga možemo zaključiti da će $b_i^n(t) \geq 0$ uvijek vrijediti.

Simetričnost

Za Bernsteinove bazne polinome, koji su u intervalu $t \in [0,1]$ vrijedi simetrija

$$b_{n-i}^n(1-t) = b_i^n(t).$$

Dokaz. Polazimo iz osnovne definicije Bernsteinovih polinoma $b_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$, gdje je $i = 0,1,\dots,n$. Iz toga vrijedi:

$$b_{n-i}^n(1-t) = \binom{n}{n-i}(1-t)^{n-1}\big(1-(1-t)\big)^{n-(n-i)}$$

$$= \frac{n!}{(n-i)!\,\big(n-(n-i)\big)!}(1-t)^{n-i}t^i = \frac{n!}{i!\,(n-i)!}t^i(1-t)^{n-i}$$

$$= \binom{n}{i} t^i (1-t)^{n-i} = b_i^n(t).$$

■

Rekurzivnost

Bazne polinome stupnja $n+1$ jednostavno zapisujemo u rekurzivnom obliku pomoću baznih polinoma stupnja n na sljedeći način:

$$b_i^{n+1}(t) = t b_{i-1}^n(t) + (1-t) b_i^n(t)$$

za $i = 0,1,\dots,n+1$, gdje je $b_i^n(t) = 0$ za $i < 0$, odnosno $i > n$ i $b_0^0(t) = 0$.

Dokaz. Vrijedi sljedeće:

$$b_i^{n+1}(t) = \binom{n+1}{i} t^i (1-t)^{n+1-i} = (\binom{n}{i-1} + \binom{n}{i}) t^i (1-t)^{n+1-i}$$

$$= \binom{n}{i-1} t^i (1-t)^{n+1-i} + \binom{n}{i} t^i (1-t)^{n+1-i}$$

$$= t\binom{n}{i-1} t^{i-1} (1-t)^{n+1-i} + (1-t)\binom{n}{i} t^i (1-t)^{n-1}$$

$$= t b_{i-1}^n(t) + (1-t) b_i^n(t).$$

■(8)

Unimodalnost

Poliom $b_i^n(t)$ ima jedan jedini ekstrem gdje je $t = \frac{i}{n}$ za $t \in [0,1]$.

Dokaz. Pronađimo lokalne ekstreme.

$$\frac{d}{dt} b_i^n(t) = \binom{n}{i} i t^{i-1} (1-t)^{n-i} + \binom{n}{i} t^i (-1)(n-i)(1-t)^{n-i-1}$$

$$= \binom{n}{i} \left(i t^{i-1} (1-t)^{n-i} - t^i (n-i)(1-t)^{n-i-1} \right)$$

$$= \binom{n}{i} t^{i-1} \left(i(1-t)^{n-1} - t(n-i)(1-t)^{n-i-1} \right)$$

$$= \binom{n}{i} t^{i-1} (1-t)^{n-i-1} \left(i(1-t) - t(n-i) \right).$$

Izraz $\binom{n}{i} t^{i-1} (1-t)^{n-i-1}$ je jednak 0, kada je $t = 0$ i kada je $i \notin \{0,1\}$, ili kada je $t = 1$ i $i \notin \{n-1, n\}$. Samo onda polinom $b_i^n(t)$ ima vrijednost 0.

Preostaje nam samo da drugi dio dobivenog izraza $(i(1-t) - t(n-i))$ izjednačimo s 0.

$$i(1-t) - t(n-i) = 0 \Leftrightarrow i(1-t) = t(n-i) \Leftrightarrow i - it = tn - it \Leftrightarrow i = nt \Leftrightarrow t = \frac{i}{n}.$$

S time smo dokazali da je maksimum Bernsteinovih polinoma dosegnut u $t = \frac{i}{n}$. ■

2.1.3. Bézierove krivulje

Bézierove krivulje sastavljene su od dviju ili više točaka koje određuju područje na kojem je krivulja definirana i njezin oblik. Prva i posljednja točka označavaju početnu i krajnju točku krivulje, dok sve ostale točke određuju oblik, odnosno zakrivljenost krivulje. Poznajemo linearne, kvadratne, kubične te Bézierove krivulje višeg stupnja. Najčešće upotrebljavamo kvadratne i kubične Bézierove krivulje. Kod kompliciranijih oblika koji zahtjevaju korištenje krivulja višeg stupnja koristimo više krivulja manjeg stupnja zbog jednostavnijeg računanja, lakše kontrole i boljih aproksimacijskih svojstava. Bézierove krivulje definirat ćemo pomoću Bernsteinovih baznih polinoma uz pomoć De Casteljauovog algoritma.

De Casteljauov algoritam

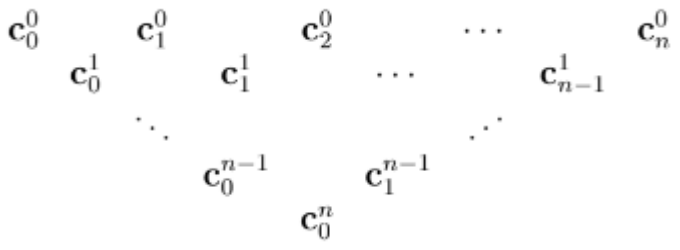
Zadane su točke $c_0, c_1, \dots, c_n \in \mathbb{R}^2$ koje zovemo kontrolne točke i $t \in [0,1]$.

1. Definiramo $c_i^0(t) = c_i$, gdje je $i = 0,1,\dots,n$.
2. Iterativno ponavljamo izračun za svaku sljedeću točku:

   $c_i^k(t) = (1-t)c_i^{k-1}(t) + tc_{i+1}^{k-1}(t)$, gdje je $k = 1,2,\dots,n$ i $i = 0,1,\dots,n-k$. **(3)**

3. Posljednja točka u izračunu je $c_0^n(t)$. Ta točka pripada Bézierovoj krivulji $b(t)$ za parametar t.

Uobičajno sve točke De Casteljauovg algoritma zapisujemo u trokutastu shemu, kako bi nam ovaj rekurzivni proces bio lakše razumljiv.



Slika 3. De Casteljauova shema

Bézierovu krivulju tada zapisujemo kao posebnu kombinaciju Bernsteinovih polinoma i kontrolnih točki.

Definicija 2.2. Neka je zadano $n+1$ kontrolnih točaka $c_0, c_1, \dots, c_n \in \mathbb{R}^2$. Tada je Bézierova krivulja stupnja n

$$b(t) = \sum_{i=0}^{n} c_i b_i^n(t), \qquad t \in [0,1],$$

gdje su $b_i^n(t)$ Bernsteinovi bazni polinomi, a točke c_i kontrolne točke Bézierove krivulje.

Teorem 2.2 Elementi k-tog stupca De Casteljauovog algoritma izražavaju kao:

$$c_i^k(t) = \sum_{l=i}^{i+k} c_l^0(t) b_{l-i}^k(t) = \sum_{l=i}^{i+k} c_l(t) b_{l-i}^k(t), \text{ za svaki } i = 0,1,\dots,n-k.$$

Dokaz 2.2 Dokaz provodimo matematičkom indukcijom. Neka je $k = 0$. Iz toga slijedi da je:

$$c_i^0 = c_i = c_i b_0^0(t) = \sum_{l=i}^{i} c_l b_{l=i}^0(t).$$

Pretpostavimo da za neki $k < n$ vrijedi:

$$c_i^{k+1}(t) = \sum_{l=i}^{i+k} c_l b_{l-i}^k(t)\,, i = 0,1,\dots,n-k.$$

Znamo da se elementi novog stupca u De Casteljauovom algoritmu dobivaju pomoć formule **(3)**. Također, koristimo pretpostavku indukcije i uzimamo u obzir rekurzivnost Bernsteinovih polinoma. Iz svega toga slijedi:

$$c_i^{k+1}(t) = (1-t)\sum_{l=i}^{i+k} c_l b_{l-i}^k(t) + t\sum_{l=i+1}^{i+1+k} c_l b_{l-i-1}^k(t)$$

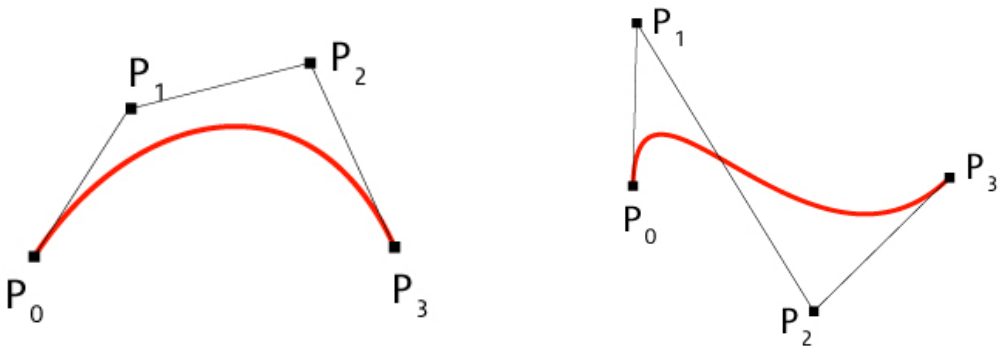
$$= c_j(1-t)b_0^k(t) + \sum_{l=i+1}^{i+k} c_l((1-t)b_{l-i}^k(t) + t b_{l-i-1}^k(t)) + c_{i+1+k} t b_k^k(t)$$

$$= c_j b_0^{k+1}(t) + \sum_{l=i+1}^{i+k} c_l b_{l-i}^{k+1}(t) + c_{i+1+k} b_{k+1}^{k+1}(t) = \sum_{l=i}^{i+1+k} c_l b_{l-i}^{k+1}(t).$$

■

Graf polinoma definiranog na navedeni način naziva se Bézierovom krivuljom i točke c_i nazivaju se kontrolnim točkama. Krivulja je u ovom slučaju ograničena u domeni [0,1], ali definirana je i za t van tog segmenta, no zbog potrebe algoritma koristimo samo navedeni interval.

Poligon koji se formira od točaka $c_0, c_1, \ldots, c_n$ nazivamo kontrolnim poligonom. Oblik krivulje koje crtamo u velikoj mjeri prati pripadajući kontrolni poligon te se zbog toga, osim krivulje, u interaktvnim grafičkim okruženjima prikazuje i njezin pripadajući kontrolni poligon. Kako korisnik pomiče točke kontrolnog poligona, krivulja mijenja svoj oblik na vrlo intuitivan i predvidiv način (*vidi sliku 4.*) (9). Na slici je točka P_2 promjenila poziciju te možemo primjetiti kako se je krivulja promijenila jer oblik krivulje prati pomicanje točke.



Slika 4. Promjena pozicije kontrolne točke

Kao primjer zapisa Bézierove krivulje zapisati ćemo jednadžbu kubične Bézierove krivulje:

$$B(t) = c_0 b_0^3(t) + c_1 b_1^3(t) + c_2 b_2^3(t) + c_3 b_3^3(t),$$

gdje su b_i^3 kubični Bernsteinovi bazni polinomi, a c_i kontrolne točke krivulje $B(t)$. Ovaj zapis lako možemo preoblikovati u:

$$B(t) = c_0(1-t)^3 + c_1(1-t)^2 t + c_2(1-t)t^2 + c_3 t^3.$$

Usporedimo li De Casteljauov algoritam s prethodnom definicijom Bézierovih krivulja zaključujemo da radimo istu stvar, samo je zapis drugačiji. De Casteljauov algoritam je numerički stabilniji, ali je i sporiji. Primjerenost i prikladnost jednog ili drugog zapisa ovisi o svrsi našeg korištenja. (10)

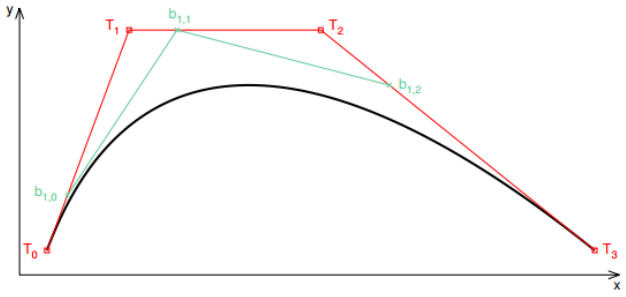
De Casteljauov algoritam određuje točku krivulje za fiksni t. Za svaki fiksni t sve stranice izvornog kontrolnog poligona se podijele novom točkom u omjeru $t : (1-t)$. Novodobivene točke spajamo spojnicama, a zatim se na tim spojnicama ponovno određuje nova točka zadana parametrom t. Postupak se ponavlja sve dok ne ostane samo jedna spojnica i jedna nova točka. Ta posljednja točka je točka krivulje (1). Ovaj postupak prikazan je na slikama broj *5., 6. i 7.*, a detaljno ćemo ga opisati u daljnjem tekstu.

Zadane su nam točke kontrolnog poligona $[T_0, T_1, T_2, T_3]$. Na dužinama između zadanih točaka odredimo poziciju nove točke zadane parametrom t (*vidi sliku 5*).

$$b_{1,0} = (T_1 - T_0) \cdot t + T_0,$$

$$b_{1,1} = (T_2 - T_1) \cdot t + T_1,$$

$$b_{1,2} = (T_3 - T_2) \cdot t + T_2.$$

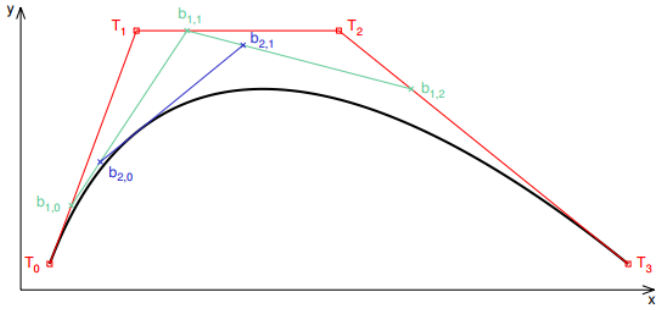


Slika 5. 1. korak rekurzije De Casteljauovog algoritma

Ovim postupkom dobivamo novi skup točaka prema kojima crtamo spojnice te ponavljamo postupak (*vidi sliku 6*):

$$b_{2,0} = (b_{1,1} - b_{1,0}) \cdot t + b_{1,0},$$

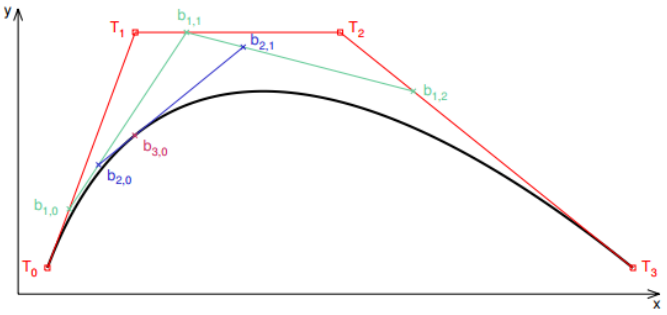
$$b_{2,1} = (b_{1,2} - b_{1,1}) \cdot t + b_{1,1}.$$



Slika 6. 2. korak rekurzije De Casteljauovog algoritma

Nakon izvršene druge iteracije, dobivamo dvije nove točke te na spojnici tih točaka određujemo novu točku zadanu parametrom t (*vidi sliku 7*).

$$b_{3,0} = (b_{2,1} - b_{2,0})\, t + b_{2,0}.$$



Slika 7. 3. korak rekurzije De Casteljauovog algoritma

$b_{3,0}$ izrazit ćemo pomoću zadanih točaka kontrolnog poligona te nakon skraćivanja i uvrštavanja dobivamo:

$$b_{3,0} = (1-t)^3 \cdot T_0 + 3t(1-t)^2 \cdot T_1 + 3t^2(1-t) \cdot T_2 + t^3 \cdot T_3.$$

Točka $b_{3,0}$ pripada krivulji, te ju možemo zapisati i kao:

$$p(t) = (1-t)^3 \cdot T_0 + 3t(1-t)^2 \cdot T_1 + 3t^2(1-t) \cdot T_2 + t^3 \cdot T_3.$$

Upotreba De Casteljauovog algoritma

U nastavku ćemo pokazati korištenje rekurzivnog algoritma na konkretnom primjeru.

Primjer 1: Zadane su nam točke $A = [1,1], B = [2,3], C = [6,4]$ *i* $D = [10,2]$, te je zadan parametar $t \in [0,1]$, a $n = 3$. Korak koji ćemo koristiti neka je 0.2.

Koristimo De Casteljauovu metodu, gdje je: $b(t) = \sum_{i=0}^{n} c_i b_i^n(t)$, te započinjemo s $t = 0$.

$$b(0) = \sum_{i=0}^{4} c_i b_i^4(t)$$

$$= A\binom{3}{0}t^0(1-t)^3 + B\binom{3}{1}t^1(1-t)^2 + C\binom{3}{2}t^2(1-t)^1$$

$$+ D\binom{3}{3}t^3(1-t)^0,$$

$$b(0) = A\,(1-t)^3 + B\,3t(1-t)^2 + C\,3t^2(1-t) + D\,t^3.$$

Primjetimo ako uvrstimo $t = 0$, samo točka A ostaje što znači da je prva točka kontrolnog poligona ujedino i prva točka krivulje. To zapisujemo kao:

$$b(0) = A = [1,1].$$

Pronašli smo prvu točku krivulje, sljedeću tražimo na isti način osim što t povećavamo za korak koji iznosi 0.2. Time je nova vrijednost parametra $t = 0.2$.

$$b(0.2) = A(1-0.2)^3 + 3B0.2(1-0.2)^2 + 3C0.2^2(1-0.2) + D0.2^3$$

$$b(0.2) = 0.512\,A + 0.384\,B + 0.096\,C + 0.008\,D$$

$$b(0.2) = [0.512, 0.512] + [0.768, 1.152] + [0.576, 0.384] + [0.08, 0.016]$$

$$b(0.2) = [1.936, 2.064].$$

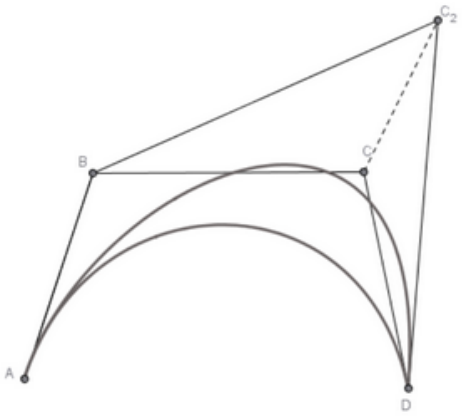
S time zaključujemo drugu iteraciju izračuna te dolazimo do druge točke krivulje. Za svaku sljedeću točku pomičemo korak parametra t i tražimo točku krivulje $b(t)$, dokle god t ne dosegne vrijednost 1.

Kao što smo već spominjali kroz rad, Bézierove krivulje imaju izrazito povoljna svojstva, a jedno od njih je to da se početna i krajnja točka krivulje poklapaju s početnom i krajnjom točkom kontrolnog poligona. Kraće zapisano:

$$B(0) = c_0, \qquad B(1) = c_n.$$

Ovo su jedine kontrolne točke koje su i same dio krivulje. Upravo je zbog ovog svojstva način na koji se crtaju Bézierove krivulje u većini grafičkih okruženja, takav da se odrede prva i posljednja točka kontrolnog poligona, a program proizvoljno određuje minimalno jednu težinsku točku između njih.

Svojstvo koje je nama najzanimljivije naziva se pseudo-lokalna kontrola i ono je od najveće praktične važnosti u oblikovanju Bézierovih krivulja. Pri promjeni bilo koje od kontrolnih točaka c_i, oblik cijele krivulje se mijenja (*vidi sliku 8*), ali promjena oblika najveća je u blizini točke s vrijednošću parametra $t = \frac{i}{n}$. Ovo svojstvo proizlazi iz svojstva unimodalnosti Bernsteinovih baznih polinoma, zato što polinom $b_i^n(t)$ doseže svoj maksimum na intervalu [0,1] na parametru $t = \frac{i}{n}$.



Slika 8. Promjena oblika krivulje

3. ODABIR OPTIMALNOG ALGORITMA

Kod odabira najoptimalnijeg algoritma pazili smo na nekoliko faktora. Primarni faktor koji je utjecao na odabir bila je brzina algoritma, odnosno linearna složenost algoritma koja utječe na brzinu izračunavanja točaka krivulje. Sekundarni, ali ne i manje bitan faktor bila je točnost dobivenih rezultata.

Upravo iz ova dva razloga izabrani algoritam je De Casteljauov algoritam. Daje nam brz i numerički stabilan način izračunavanja Bézierovih krivulja. Iako algoritam koristi faktorijele u svojim izračunima, faktorijela ne smanjuje brzinu izračuna (1).

4. PROJEKTNI ZADATAK

Kao prilog završnom radu napravljena je implementacija algoritma aproksimacije Bézierove krivulje gdje smo koristili De Casteljauov algoritam za crtanje Bézierovih krivulja.

Algoritam je implementiran u jeziku C++, a grafički je prikazan pomoću API-ja[3] OpenGL.

4.1. GLAVNA IDEJA PROJEKTA

Inicijalna ideja ovog projekta, a posljedično tome i ideja za cijeli rad nastala je iz mojeg zanimanja za animaciju i digitalno crtanje. Proučavajući pravila animiranja, bilo je zanimljivo saznati koliko se matematika koristi u pokušajima animatora da prikažu pokrete svojih likova ili objekata koje animiraju prirodnim. Daljnjim čitanjem članaka i knjiga o animiranju pojavile su se Bézierove krivulje kao jedan od često korištenih pojmova.

Moje prvo saznanje o tome kako Bézierove krivulje nastaju bio je animirani GIF koji prikazuje proces crtanja Bézierove krivulje koji me naveo na razmišljanje o tome kako bi se ta animacija mogla implementirati u jednostavan program za crtanje. Nakon proučavanja matematičkih članaka i udžbenika, pronašla sam najpovoljniji algoritam za moju ideju te ga implementirala u program koji izračunava i crta Bézierovu krivulju.

4.2. IMPLEMENTACIJA ALGORITMA

Nakon odabira optimalnog algoritma krenula je implementacija algoritma u programski jezik. Odabran je De Casteljauov algoritam te je implementiran na sljedeći način: kako bi kod programa bio pregledan i razumljiv, sve funkcije vezane uz izračun algoritma i izmjenu krivulje spadaju pod klasu *BezierCurve*.

Klasa, osim što sadržava algoritam, sadrži i pomoćne funkcije kao što su RegisterPoint() i ClearPoints() koje koristimo pri upisivanju novih točaka kontrolnog poligona, odnosno pražnjenje vektora u kojem se nalaze spremljeni podaci o točkama kontrolnog poligona. Funkcije niti primaju parametre, niti ih vraćaju (void funkcije).

[3] API - Aplikacijsko programsko sučelje (eng. application programming interface, API) ili sučelje za programiranje aplikacija je skup određenih pravila i specifikacija koje programeri slijede tako da se mogu služiti uslugama ili resursima operacijskog sustava ili nekog drugog složenog programa kao standardne biblioteke rutina (funkcija, procedura, metoda), struktura podataka, objekata i protokola (11).

```
//function to push back points into the vector
void RegisterPoint(float x, float y)
{
    points.push_back({ x,y });
}

//clearing all of the inputed points
void ClearPoints()
{
    points.clear();
}
```

Slika 9. Projektni kod - unos i brisanje točaka

Funkcija RegisterPoint() poziva se prilikom svakog klika mišem kako bi se spremile koordinate novounesene točke kontrolnog poligona, dok se funkcija ClearPoints() poziva pritiskom tipke *'C'*, kako bi se ispraznio vektor koji sadrži koordinate svih točaka kontrolnog poligona. Funkcije unutar sebe pozivaju funkcije iz <vector> biblioteke kako bi manipulirali vektorom točaka kontrolnog poligona.

Implementacija algoritma smještena je u funkciji klase GetCurve() koja vraća vektor *curvePoints* u koji spremamo izračune koordinata točaka krivulje. Funkcija se poziva u glavnom programu nakon što je korisnik unio 4 ili više točaka kontrolnog poligona.

```
if (curve.points.size() > 3)
{

    std::vector<glm::vec2> curvePoints;
    curvePoints = curve.GetCurve();
    std::cout << "Saved points: " << curvePoints.size() << std::endl;

    //rendering the curve
    RenderLine(curvePoints);

}
```

Slika 10. Projektni kod - nakon unosa pozivamo glavnu funkciju

Osim kontrolnih točaka, drugi parametar De Casteljauovog algoritma je varijabla *t*. U ovom programu izračunavamo ju na način da podijelimo svoju trenutno poziciju sa ukupnom duljinom dužina kontrolnog poligona. Iz tog razloga na samom početku, kao pripremu za algoritam, izračunavamo duljinu dužina, *curveLength*, koju čine točke kontrolnog poligona. To vršimo sljedećom formulom:

$$d = \left|\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}\right|.$$

Uz izračunavanje duljine, inicijaliziramo i varijablu *currentPoint* kako bi algoritam započeo od prve točke kontrolnog poligona. U *while* funkciji kojoj je uvjet *currentPoint* < *curveLength*,

varijablu *currentPoint* povećavamo za korak nakon svake iteracije. Izračun varijable *t* također vršimo u petlji na sljedeći način:

$$t = \frac{currentPoint}{curveLength}.$$

Na ovaj način u programu je osigurano da je *t* uvijek u intervalu [0,1].

Inicijaliziramo varijablu *n* u koju spremamo broj točaka kontrolnog poligona umanjenu za 1. U petlji inicijaliziramo također pomoćnu varijablu *point* veličine *glm::vec2*. Varijablu *point* koristiti ćemo kao spremnik podataka za sumu De Casteljauovog algoritma, te ju stoga inicijaliziramo sa početnim vrijednostima (0,0).

Pomoću *for* petlje prolazimo kroz sve vrijednosti *i* varijable, te izračunavamo svaku točku krivulje. Dobiveni rezultat jedne točke sprema se tijekom izračunavanja u varijablu *point*, koja se na kraju sprema u prethodno inicijalizirani vektor *curvePoints*. Nakon što se izvrši izračun jedne točke prelazimo na izračun sljedeće tako da se varijabli *currentPoint* dodaje korak, te ukoliko i dalje vrijedi uvjet *while* petlje izračunavamo sljedeću točku. Svaku novu točku spremamo u vektor *curvePoints.*

Glavni dio algoritma implementiran je na sljedeći način:

```
while (currentPoint < curveLength)
{
    float t = currentPoint / curveLength;

    int n = points.size() - 1;
    glm::vec2 point(0.0f, 0.0f);

    //the main part of the algorithm

    for (int i = 0; i <= n; i++)
    {
        point = vectorAdd(point, vectorScale(points[i], BinomialCoeff(n, i) * pow(1 - t, n - i) * pow(t, i)));
    }

    curvePoints.push_back(point);
    std::cout << "Calculated point: " << point.x << " : " << point.y << std::endl;
    currentPoint += step;

}
```

Slika 11. Projektni kod - glavni dio algoritma

Koristimo *for* petlju za pomicanje iteracije i kretanje po *i* varijabli, te pomoćne funkcije *vectorAdd()* i *vectorScale()*, funkcije zbrajanja, odnosno skaliranja vektorskih vrijednost varijable *point.* Kada program završi sa *i* iteracijama, dobivenu točku *point* program sprema u vektor *curvePoints* pomoću funkcije *push_back()*.

Kada uvjet *while* petlje više nije istina, odnosno kada stignemo do kraja dužine kontrolnog poligona petlja se prekida, a funkcija *GetCurve()* vraća vektor *curvePoints* kao rezultat algoritma. Dobiveni set podataka koristimo dalje za prikaz Bézierove krivulje.

4.3. PRIKAZIVANJE BÉZIEROVE KRIVULJE

Za grafički prikaz dobivenog seta podataka koristimo se APi-jem OpenGL. Koritimo *glfw* biblioteku za otvaranje prozora za prikaz i za dohvat inputa korisnika, te *glew* biblioteku za prikaz podataka u prozoru (12).

Prozor za prikazivanje postavljamo tako da inicijaliziramo glfw biblioteku pozivom funkcije glfwInit(), te nakon inicijalizacije pozivamo funkciju glfwCreateWindow() koja kao parametre uzima veličinu prozora koji želimo prikazivati, ime prozora, na kojem monitoru se otvara full screen programa i s kojim prozorom dijeli resurse. U našem slučaju parametri funkcije su postavljeni na sljedeći način: (width, height, "Bezier Curve", nullptr, nullptr). Pošto program ne koristi full screen i ne dijeli resurse sa drugim prozorima, posljednja dva parametra postavljena su na nullptr, odnosno ne odabiremo monitor ili drugi prozor. Kako bi naše podatke mogli prikazivati u prozoru, postavljamo ga kao trenutni kontekst funkcijom *glfwMakeContextCurrent()*, odnosno označavamo ga kao trenutno radno mjesto kako bi program znao da crtamo u novostvorenom prozoru.

Slika 12. Prozor za prikazivanje podataka

```
GLFWwindow* window;

if (!glfwInit())
    std::cout << "Error : could not initilize GLFW";

glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
glfwWindowHint(GLFW_SAMPLES, 4);

window = glfwCreateWindow(width, height, "Bezier Curve", nullptr, nullptr);

if (!window)
{

    std::cout << "Error : could not create window";
    glfwTerminate();

}

glfwMakeContextCurrent(window);
```

Slika 13. Projektni kod - inicijalizacija prozora

Nakon stvaranja prozora sve funkcije i radnje koje program izvodi dok je prozor otvoren postavljamo u *while* petlju kojoj je jedini uvjet da je prozor i dalje otvoren i aktivan. Kada zatvorimo prozor *while* petlja se prekida i poziva se funkcija *glfwTerminate()* koja prekida program.

U *while* petlji koju smo prethodno definirali pozivamo i funkciju koja opisuje na koji način će naš program obavljati zadane funkcije. U početku samog projekta korištena je funkcija *glfwPollEvents()* koja odmah obrađuje sve zadatke u redu čekanja. Tako bi za ovaj projekt nakon unosa kontrolnih točaka program izvršio izračun, nacrtao točke i krivulju, ali bi to sve odmah nestalo iz prozora jer funkcija *glfwPollEvents()* ne bi čekala novu naredbu ili unos već samo nastavila dalje s novim zadatkom u redu čekanja.

To nam nije odgovaralo, te umjesto te funkcije koristimo *glfwWaitEvents()* funkciju koja stavlja svoju glavnu nit u stanje mirovanja dok ne postoji barem jedan ili više zadataka u redu čekanja. Kada postoje takvi zadatci u redu čekanja onda se funkcija ponaša isto kao i *glfwPollEvents()*, odnoso obrađuje sve zadatke u redu čekanja, te kada su ti zadatci gotovi vraća se u stanje mirovanja i ponovno čeka nove zadatke. Takav način obrade zadataka na najviše odgovara jer čekamo na korisnikov unos novih kontrolnih točaka.

Podaci koje prikazujemo su točke kontrolnog poligona koje unosi korisnik, a unosi ih klikom na željeno mjesto u prozoru za prikazivanje. Za dohvat pozicije miša koristimo se funkcijom *glfwGetMousePosition()*, poziciju točke spremamo u vektor *points*, a točke spremljene u taj vektor prikazuju se na ekranu odmah po unosu sve dokle god korisnik ne odluči ne unositi nove podatke ili obrisati sve unesene točke.

```
if (glfwGetMouseButton(window, GLFW_MOUSE_BUTTON_LEFT) == GLFW_PRESS && mouseHeld == false)
{
    //pushing in up to 10 points
    if (curve.points.size() < 10)
    {
        //input points
        double xpos, ypos;
        glfwGetCursorPos(window, &xpos, &ypos);
        std::cout << xpos << " : " << ypos << std::endl;
        //pushing new points into the curve vector
        curve.RegisterPoint(xpos, ypos);

    }

    mouseHeld = true;
}
```

Slika 14. Projektni kod - unos i dohvaćanje podataka

Osim kontrolnih točaka, kao glavni dio ovog projekta, prikazujemo i Bézierovu krivulju. Nakon što program završi s kalkulacijom točaka krivulje, taj set podataka prikazuje se na prozoru za prikazivanje zajedno s točkama kontrolnog poligona. Nakon što je korisnik unio 4 ili više točaka kontrolnog poligona program odmah izračunava točke krivulje. Nakon svakog novog unosa točke kontrolnog poligona (max. 10) program ponovno izračunava točke krivulje te prikazuje novu krivulju.

U samom projektu rendereri za prikazivanje točaka kontrolnog poligona i krivulje spremljeni su kao zasebne C++ datoteke koje se povezuju na glavnu datoteku projekta.

4.3.1. Prikaz točaka kontrolnog poligona

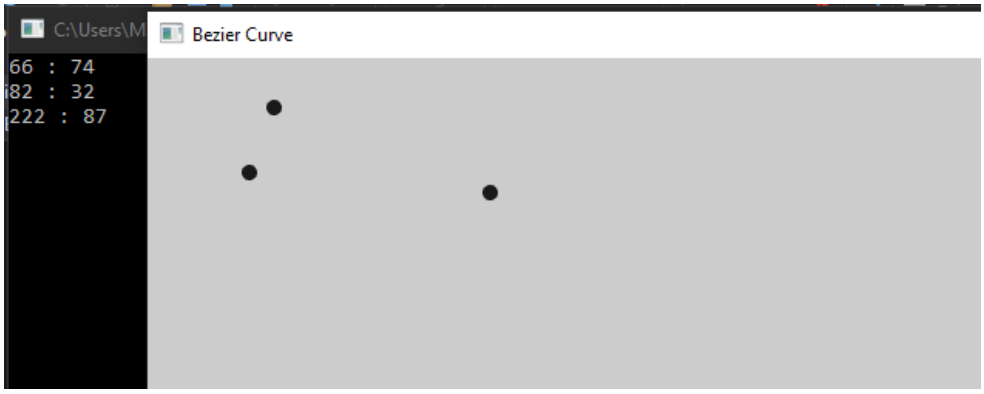
Prikaz točaka kontrolnog poligona implementirano je na način da se pri svakom novootkrivenom unosu točke poziva funkcija *renderPoint()* koja kao svoje parametre zaprima poziciju točke, te mjerilo kojim određujemo veličinu točke koju crtamo.

```
for (glm::vec2& vec : curve.points)
{
    RenderPoint(glm::vec2(vec.x, vec.y), 5);
}
```

Slika 15. Projektni kod – prikaz točaka

Osim funkcije *renderPoint()* za prikaz točaka kontrolnog poligona prije svega moramo inicijalizirati renderer funkcijom *InitPointRendering()*, koja kao parametar zaprima koliko vertexa dajemo rendereru na korištenje. Ovom funkcijom čitamo podatke za shadere, te podatke shadera spremamo u *shader index*, koji pri crtanju dajemo kao parametar kojim program crta. Pri inicijalizaciji, također se pripremaju setovi podataka kako bi program nacrtao točku kao popunjenu kružnicu, a ne samo kao pixel.

Pošto u OpenGL-u ne postoji već određeni način, odnosno funkcija crtanja krugova, moramo dodatno postaviti parametre kako bi funkcija koja inače crta trokute crtala lijepu glatku točku (12).



Slika 16. Prikaz zadanih točki s pripadajućim koordinatama

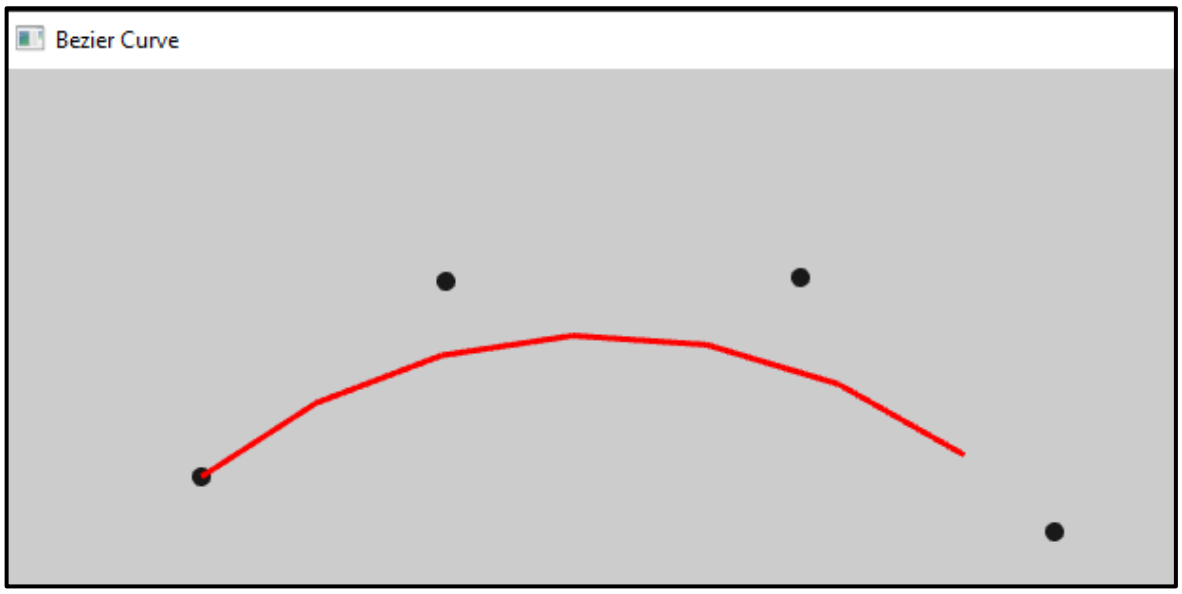
4.3.2. Prikaz krivulje

Prikaz same krivulje vršimo odmah po završetku izračuna točaka krivulje. Kada dobijemo set podataka iz glavnog algoritma poziva se funkcija *renderLine()* koja u pravilu prikazuje dužine, a ne krivulju. To se u ovom slučaju ne vidi prilikom prikazivanja, zbog malog koraka koji smo koristili prilikom izračunavanja točaka krivulje.

```
if (curve.points.size() > 3)
{

    std::vector<glm::vec2> curvePoints;
    curvePoints = curve.GetCurve();
    std::cout << "Saved points: " << curvePoints.size() << std::endl;

    //rendering the curve
    RenderLine(curvePoints);

}
```

Slika 17. Projektni kod – prikaz krivulje

Ono što funkcija *renderLine()* radi je sljedeće: zaprima vektor u kojem su spremljene pozicije točaka krivulje, te između svake dobivene točke crta liniju kako bi ih povezala. Na ekranu se te dužine na daju razaznati jedna od druge upravo zbog toga što je korak koji smo koristili u izračunu algoritma malen, pa su dobivene točke krivulje dovoljno blizu jedna drugoj da se na ekranu prikazuje glatka krivulja. Kada bi koristili veći korak krivulja bi izgledala 'izlomljeno' (*vidi sliku 18*).



Slika 18. Bézierova krivulja s velikim korakom izračuna

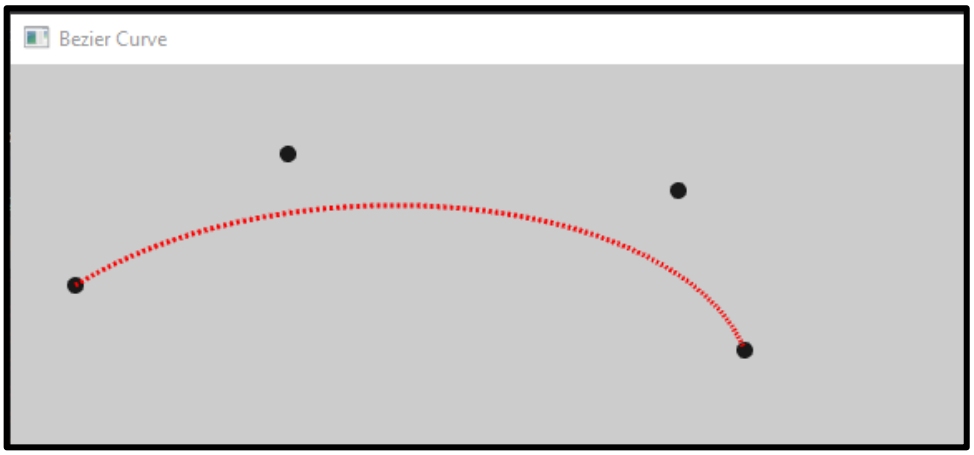
Funkciju koju koristimo za prikazivanje seta podataka dobivenog iz De Casteljauovog algoritma je *glDrawElements()* koja kao svoje parametre zaprima: kako će prikazivati dobiven set podataka, broj elemenata za prikazivanje, tip podataka indexa i pointer na spremljene indekse. Indeksi koje ova funkcija prima kao parametar su indeksi koordinata točaka krivulje, te prema indeksu funkcija crta primitivnu liniju između dvije točke.

```
glDrawElements(GL_LINES, indecies.size(), GL_UNSIGNED_INT, nullptr);
```

Slika 19. Projektni kod - glDrawElements() funkcija

Primijetite kako funkcija kao parametar ne uzima koordinate točaka, već samo indekse koordinata. Niz u koji spremamo indekse točaka definiramo u samoj funkciji *renderLine()* tako što u jednostavnoj *for* petlji spremamo mjesto na kojem se nalazi određena točka.

Funkcija *glDrawElements()* je primitivna funkcija koja crta na način da od dobivenog seta podataka uzima prva dva indeksa [0] i [1] te crta liniju između njih, nakon toga uzima indekse [2] i [3], te tako redom do kraja niza. Kada bi naše koordinate točaka krivulje bile indeksirane po redu tad bi funkcija crtala isprekidanu krivulju (*vidi sliku 20*).



Slika 20. Bézierova krivulja - netočni indexi

Zbog toga kod definicije indeksa koordinata svaku točku osim prve i zadnje, indexiramo dvaput, odnosno naši indeksi pokazuju na točke krivulje na sljedeći način: [0][1][1][2][2][3]...[n-2][n-1][n-1][n].

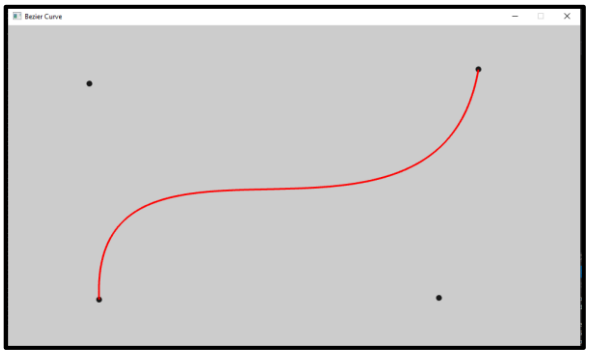
```
for (int i = 0; i < points.size(); i++)
{
    if (i != points.size() - 1) {
        indecies.push_back(i);
        if (i < points.size())
            indecies.push_back(i + 1);
    }
}
```

Slika 21. Projektni kod – definicija indexa

4.4. REZULTAT PROJEKTA

Rezultat ovog projekta je funkcionalan program koji prema zadanim točkama kontrolnog poligona izračunava i crta Bézierovu krivulju. Kontrolne točke korisnik može proizvoljno zadavati klikom miša u prozoru programa, te kada korisnik unese najmanje 4 kontrolne točke program započinje izračun točaka krivulje i crta ih.

Korisniku je omogućeno da unosi do 10 kontrolnih točaka. Također je omogućeno brisanje unesenih točaka pritiskom na tipku '*C*'. Tako korisnik može nebrojno puta crtati nove krivulje.



Slika 22. Bézierova krivulja - finalni produkt

4.5. BUDUĆNOST PROJEKTA

Budućnost ovog projekta bila bi inspirirana svojstvom pseudo-lokalne kontrole o kojem smo više rekli u poglavlju 2.1.3. Prateći to svojstvo u projekt dodala bi se mogućnost pomicanja točaka kontrolnog poligona kako bi korisniku bilo omogućena izmjena pozicije i zakrivljenosti krivulje bez brisanja zadanih kontrolnih točaka i ponovnog unosa. Ideja za postizanje ovog dodatka provodila bi se uz pomoć glfw biblioteke koju program već koristi te bolje korištenje funkcije glfwGetMouseButton().

Još jedna ideja bila bi omogućavanje korisniku da crta više krivulja odjednom, što bi automatski značilo da korisnik može crtati kompleksnije krivulje spajanjem više jednostavnijih krivulja. Ta mogućnost mogla bi se implementirati tako da program prepoznaje više krivulja kao različite objekte kojima bi korisnik mogao manipulirati. Inicijalna ideja za takvo raspoznavanje je da se program postavi tako da korisnik mora prilikom unosa držati 'index' tipku, a program bi imao zadatak da osluškuje koja tipka je bila pritisnuta za vrijeme unosa određenih podataka, te po tom 'indexu' spremiti novi set podataka i korisniku pritiskom na 'index' tipku omogućiti izmjenu istih.

Dodavanjem ovih ideja i poboljšanja ovaj program postao bi program za crtanje. Korisnik bi mogao crtati bilo kakav prirodan oblik uz pomoć jednostavnih krivulja.

U većini programa za crtanje postoji alat za crtanje Bézierovih krivulja. Jedan od takvih programa koji u svojim alatima sadrži Bézierove krivulje je Krita (13).

5. ZAKLJUČAK

Kao dio zadatka završnog rada detaljnije smo se upoznali s Bézierovim krivuljama i algoritmima pomoću kojih crtamo Bézierove krivulje. Definirali smo Bernsteinove težinske funkcije i polinome koje je De Casteljau koristio u svojem algoritmu za izračun Bézierovih krivulja. Predstavili smo svojstva Bernsteinovih polinoma i nama zanimljiva svojstva Bézierovih krivulja.

Projektni zadatak implementacije izvršen je u programskom jeziku C++ uz pomoć API-ja OpenGL, a implementiran je De Casteljauov algoritam koji je izabran kao najoptimalniji. O samom odabiru optimalnog algoritma pričamo više u poglavlju 3. Osim same implementacije u radu opisujemo i sav dodatan sadržaj pomoću kojeg prikazujemo sve setove informacija algoritma: kontrolne točke poligona i dobivenu krivulju. Kao najveći izvor literature za projektni zadatak korištena je dokumentacija OpenGL-a, dokumentacija C++-a, te dokumentacija dodatnih biblioteka koje su korištene u projektnom zadatku.

6. LITERATURA

1. Čupić M, Mihaljević Ž. Interaktivna računalna grafika kroz primjere u OpenGL-u. 2018.

2. Bézier curve. U: Wikipedia [Internet]. 2022 [citirano 06. siječanj 2023.]. Dostupno na: https://en.wikipedia.org/w/index.php?title=B%C3%A9zier_curve&oldid=1128193830

3. Animatron Blog: Latest Video Marketing Tips & News [Internet]. [citirano 08. siječanj 2023.]. 12 Basic Principles of Animation. Dostupno na: https://www.animatron.com/blog/12-basic-principles-of-animation/

4. Leksikografski zavod Miroslav Krleža. Enciklopedija. 2021 [citirano 07. siječanj 2023.]. CAD | Hrvatska enciklopedija. Dostupno na: https://www.enciklopedija.hr/natuknica.aspx?ID=68079

5. Leksikografski zavod Miroslav Krleža. Bézierova krivulja | Hrvatska enciklopedija [Internet]. [citirano 07. siječanj 2023.]. Dostupno na: https://www.enciklopedija.hr/natuknica.aspx?id=68077

6. Aproksimacija. U: Wikipedija [Internet]. 2022 [citirano 02. veljača 2023.]. Dostupno na: https://hr.wikipedia.org/w/index.php?title=Aproksimacija&oldid=6276354

7. Polinom – Wikipedija [Internet]. [citirano 05. ožujak 2023.]. Dostupno na: https://hr.wikipedia.org/wiki/Polinom

8. Farin GE. Curves and surfaces for CAGD: a practical guide. 5th ed. San Francisco, CA: Morgan Kaufmann; 2001. 497 str. (Morgan Kaufmann series in computer graphics and geometric modeling).

9. Floater MS. Lecture 8: Bézier curves. U 2018.

10. Jaklič G. yumpu.com. [citirano 07. lipanj 2023.]. Krivulje in ploskve v računalniško podprtem geometrijskem ... Dostupno na: https://www.yumpu.com/xx/document/read/13045606/krivulje-in-ploskve-v-racunalnisko-podprtem-geometrijskem-

11. API. U: Wikipedija [Internet]. 2022 [citirano 03. veljača 2023.]. Dostupno na: https://hr.wikipedia.org/w/index.php?title=API&oldid=6305486

12. OpenGL Documentation [Internet]. [citirano 14. lipanj 2023.]. Dostupno na: https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/

13. Bezier Curve Tool — Krita Manual 5.0.0 documentation [Internet]. [citirano 03. veljača 2023.]. Dostupno na: https://docs.krita.org/en/reference_manual/tools/path.html

7. POPIS SLIKA

Slika 1. Aproksimacija i interpolacija Bézierove krivulje 3
Slika 2. Bernsteinov polinom stupnja n=4 4
Slika 3. De Casteljauova shema 7
Slika 4. Promjena pozicije kontrolne točke 9
Slika 5. 1. korak rekurzije De Casteljauovog algoritma 10
Slika 6. 2. korak rekurzije De Casteljauovog algoritma 10
Slika 7. 3. korak rekurzije De Casteljauovog algoritma 11
Slika 8. Promjena oblika krivulje 12
Slika 9. Projektni kod - unos i brisanje točaka 15
Slika 10. Projektni kod - nakon unosa pozivamo glavnu funkciju 15
Slika 11. Projektni kod - glavni dio algoritma 16
Slika 12. Prozor za prikazivanje podataka 17
Slika 13. Projektni kod - inicijalizacija prozora 18
Slika 14. Projektni kod - unos i dohvaćanje podataka 19
Slika 15. Projektni kod – prikaz točaka 19
Slika 16. Prikaz zadanih točki s pripadajućim koordinatama 20
Slika 17. Projektni kod – prikaz krivulje 20
Slika 18. Bézierova krivulja s velikim korakom izračuna 21
Slika 19. Projektni kod - glDrawElements() funkcija 21
Slika 20. Bézierova krivulja - netočni indexi 22
Slika 21. Projektni kod – definicija indexa 22
Slika 22. Bézierova krivulja - finalni produkt 22