

LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

**RĪKS BEZJĒ LĪKŅU KONSTRUĒŠANAI UN
MODIFICĒŠANAI**

KVALIFIKĀCIJAS DARBS

Autors: **Elīza Gaile**

Studenta apliecības Nr.: eg17035

Darba vadītājs: doc., Dr. dat. Vineta Arnicāne

RĪGA 2019

ANOTĀCIJA

Kvalifikācijas darbā ir aprakstīta sistēma interaktīvam grafisku objektu rīkam, kas paredzēta kubisku Bezjē līkņu konstruēšanai, modifīcēšanai un dzēšanai, izmantojot datorpeli, datora tastatūru vai teksta failus. Sistēma piedāvā ģenerēt četru dažādu veidu Bezjē līknes, tai skaitā interpolētas līknes un saliktas līknes ar C^2 nepārtrauktību (B-splaini), papildus izmantojot trīs dažādu veidu parametrizācijas metodes. Iespējams arī izvadīt konstruēto Bezjē līkņu kontrolpunktus un/vai mezglu punktus, kā arī citus līkni raksturojošos lielumus. Līkņu vizuālai salīdzināšanai iespējams augšupielādēt fona attēlu un pietuvināt, attālināt grafiskos objektus.

Sistēma implementēta C# programmēšanas valodā, izmantojot Microsoft .NET satvaru un tajā esošo Windows Forms bibliotēku.

Atslēgas vārdi: C#, Microsoft .NET, Windows Forms, Bezjē līknes, interpolācija.

ABSTRACT

TOOL FOR CONSTRUCTION AND MODIFICATION OF BÉZIER CURVES

This qualification paper describes a system for an interactive tool of graphical objects meant for construction, modification and deletion of Bézier curves using a mouse, keyboard or text files. The system allows to create four types of curves, including fitted Bézier curves and composite curves with C^2 continuity (B-splines), and to choose from three different parameterization methods. Additionally, it is possible to output control points, knot points and other descriptive variables of the constructed Bézier curves. It is possible to upload a background image and to zoom in and out of graphical objects for a visual comparison of curves.

System is implemented in C# programming language, using Microsoft .NET framework and Windows Forms library included in the framework.

Keywords: C#, Microsoft .NET, Windows Forms, Bézier curves, interpolation.

SATURA RĀDĪTĀJS

IEVADS	7
Nolūks	7
Darbības sfēra	7
Saistība ar citiem dokumentiem	7
Pārskats	7
APZĪMĒJUMU SARAKSTS	8
1. VISPĀRĒJS APRAKSTS	10
1.1. Esošā stāvokļa apraksts	10
1.2. Pasūtītājs	10
1.3. Produkta perspektīva	10
1.4. Darījumsprasības	10
1.5. Sistēmas lietotāji	10
1.6. Vispārējie ierobežojumi	10
1.7. Pieņēmumi un atkarības	11
2. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA	12
2.1. Funkcionālās prasības	12
2.2. Nefunkcionālās prasības	12
3. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS	13
3.1. Projektējuma matemātiskais pamatojums	13
3.1.1. Kubisku Bezjē līkņu izmantošanas pamatojums	13
3.1.2. Bezjē līkņu pamatteorija	14
3.1.3. Kubiskās Bezjē līknes interpolācija caur četriem punktiem	15
3.1.4. Tuvākās līknes metodes izmantošanas pamatojums un teorija	16
3.1.5. Saliktas līknes izmantošanas pamatojums un teorija	16
3.1.6. B-splainu konstruēšanas un modifīcēšanas metode	17

3.1.7. Parametrizācijas metodes	18
3.1.8. Līkņu modificēšanas metodes	21
3.2. Lietotāja saskarņu projektējums	22
3.2.1. Galvenais logs.....	22
3.2.2. Koordinātu logs	25
3.2.3. Klūdu un citi paziņojumi	27
3.3. Funkciju projektējums	27
3.3.1. Galvenā loga pamatfunkcijas.....	28
3.3.2. Jaunu līkņu pievienošana	29
3.3.3. Līkņu konstruēšana.....	31
3.3.4. Līkņu modificēšana	35
3.3.5. Esošu līkņu funkcijas.....	38
3.3.6. Koordinātu logs	38
3.4. Funkcionālo prasību un projektējuma funkciju atbilstība.....	39
4. TESTĒŠANAS DOKUMENTĀCIJA	41
4.1. Testpiemēru projektējums.....	41
4.2. Testēšanas žurnāls.....	48
5. PROJEKTA ORGANIZĀCIJA	50
5.1. Darbietilpības novērtējums	50
5.1.1. Prognozētā darbietilpība	50
5.1.2. Reālā darbietilpība	51
5.2. Kvalitātes nodrošināšana	51
5.3. Konfigurāciju pārvaldība	52
6. REZULTĀTI UN SECINĀJUMI.....	53
PATEICĪBAS	54
IZMANTOTĀ LITERATŪRA UN AVOTI	55

PIELIKUMI.....	57
1. pielikums. Programmatūras koda frgments.....	57
2. pielikums. Mazāko kvadrātu metodes pierādījums Bezjē līknes interpolācijai	67
3. pielikums. Skalārā reizinājuma $\langle \mathbf{A}\mathbf{x}, \mathbf{x} \rangle$ atvasinājums	71

IEVADS

Nolūks

Kvalifikācijas darba nolūks ir iepazīstināt sistēmas izstrādātājus un lietotājus ar sistēmas prasībām, projektējumu, testēšanas dokumentāciju un projekta organizāciju, funkcijām un rīka funkcionalitātes pielietojumiem, kā arī nodrošināt prasību saskaņošanu turpmākajā izstrādes un uzturēšanas procesā.

Darbības sfēra

Kvalifikācijas darbā aprakstītā sistēma paredzēta iekšējām pasūtītāja uzņēmuma vajadzībām, lai manuāli konstruētu un salīdzinātu līknes apgērba piegrieztnēs. Sistēmas lietotāji izstrādātajā rīkā var konstruēt, modifīcēt un dzēst četru veidu kubiskas Bezjē līknes, kas atšķiras ar konstruēšanas veidu.

Saistība ar citiem dokumentiem

Dokuments tika izstrādāts saskaņā ar programmatūras prasību specifikācijas standartu LVS 68:1996 “Programmatūras prasību specifikācijas ceļvedis” [1], ar programmatūras projektējuma apraksta standartu LVS 72:1996 “Ieteicamā prakse programmatūras projektējuma aprakstīšanai” [2] un standartu LVS 70:1996 “Programmatūras testēšanas dokumentācija” [3].

Pārskats

Kvalifikācijas darba dokuments ir sadalīts sešās galvenajās daļās. Tā sastāvā ir vispārējais apraksts – ietver produktu aprakstu vispārējā līmenī un informāciju par sistēmas lietotājiem, darījumprasībām, arī ierobežojumiem, pieņēmumiem un atkarībām, programmatūras prasību specifikācija – ietver funkcionālās un nefunkcionālās prasības un programmatūras projektējuma apraksts – ietver daļēju funkciju un saskaņu projektējumu. Bez tā, dokumentā ir informācija par programmatūras testēšanu – ietver testpiemēru projektējumu un testēšanas žurnālu ar rezultātiem un projekta organizāciju – apraksta programmatūras izstrādes procesus, iekļauj arī darbietilpības novērtējumu, kvalitātes nodrošināšanu un konfigurāciju pārvaldību. Dokumenta noslēgumā atrodami rezultāti un secinājumi – apraksta izstrādās sistēmas rezultātus un secinājumus, kas radušies pēc kvalifikācijas darba izstrādes. Kvalifikācijas darbā atrodami arī izmantotie avoti un literatūra, pateicības un pielikumi.

APZĪMĒJUMU SARAKSTS

B-splains Splainu veids, kas veidots gludi savienojot Bezjē līknes.

Beigu punkts Bezjē līknes pēdējais kontrolpunkts.

Bezjē līkne Parametriska līkne, ko uzdod izmantojot kontrolpunktus; populārs līkņu uzdošanas veids datorgrafikā [4].

Blakus rokturis Saliktas līknes rokturi sauc par mezgla punkta blakus rokturi, ja tas ir iepriekšējais vai nākamais kontrolpunkts skaitot no kontrolpunkta, kas sakrīt ar minēto mezglu.

Blakus mezgls Iepriekšējais vai nākamais mezgls kādam mezglam saliktā Bezjē līknē.

C² nepārtrauktība Funkcijai piemīt C² nepārtruktība, ja tās pirmās un otrās kārtas atvasinājumi eksistē un ir nepārtrauki. Raksturo vizuāli gludu funkciju.

Galapunkts Bezjē līknes pirmsais vai pēdējais kontrolpunkts.

Git Versiju kontroles sistēma.

ID Identifikators.

Interpolācija Punktu un līkņu konstruēšanas metode, izmantojot galīgu skaitu zināmu punktu.

Kontrolpunkts Elements no punktu kopas, kas nosaka Bezjē līknes formu; n -tās pakāpes Bezjē līknei ir $n + 1$ kontrolpunktai.

Mezgla punkts Punkt, caur kuru tiek interpolēta Bezjē līkne.

NURBS Nevienmērīgi racionāli B-splains, parametrisku līkņu uzdošanas veids.

Pretējais rokturis Saliktas līknes rokturus sauc par pretējiem, ja tie ir iepriekšējais un nākamais kontrolpunkts skaitot no divu secīgu segmentu savienojuma kontrolpunkta (kas ir arī mezglu punkts). Saliktas līknes pirmajam un pēdējam rokturim nav pretējā roktura.

px Pixselis, vismazākais attēla elements.

Rokturis Saliktas Bezjē līknes kontrolpunkts, kas nav segmenta galapunkts.

Sākumpunkts Bezjē līknes pirmsais kontrolpunkts.

Splains Līkne, kas veidota savienojot patvalīgu skaitu citu līkņu.

.txt Teksta failu veids.

Bezjē līknes kontrolpunktai un mezglu punkti tiek skaitīti virzienā no līknes sākumpunkta uz beigu punktu. Segmenti tiek skaitīti virzienā no pirmā atliktā mezglu punkta uz pēdējo.

Šajā dokumentā tiek lietoti sekojoši apzīmējumi:

- vektori tiek apzīmēti ar mazo burtu treknrakstā, piemēram, \mathbf{v} ;
- matricas tiek apzīmētas ar lielo burtu treknrakstā, piemēram, \mathbf{M} ;
- skalāri lielumi tiek apzīmēti ar mazo burtu slīprakstā, piemēram, a ;
- funkcijas tiek apzīmētas ar lielo burtu slīprakstā, piemēram, F ;
- vienkāršības labad, punkts tiek uztverts kā vektors.

1. VISPĀRĒJS APRAKSTS

1.1. Esošā stāvokļa apraksts

Pieejamas vairākas programmatūras (piemēram, *Adobe Illustrator*, *GIMP*, *Inkscape* [5, 6, 7]), kas piedāvā līkņu konstruēšanas un modificēšanas funkcionalitātes, tomēr šīm programmatūrām ir ierobežotas interpolācijas un līkņu daudzveidības iespējas.

1.2. Pasūtītājs

Sistēma aprakstīta un izstrādāta pēc uzņēmuma SIA “FitDex” pasūtījuma studiju kursa “Prakse” (DatZ2033) ietvaros.

1.3. Produkta perspektīva

Kvalifikācijas darbs ir neatkarīga sistēma, kas apvieno pasūtītājam nepieciešamās funkcionalitātes, lai veiktu apģērba piegrieztņu līkņu konstruēšanu, modificēšanu un vizuālu salīdzināšanu.

1.4. Darījumsprasības

Programmatūrai jānodrošina iespēja veikt gludu un apģērbu piegrieztnēm līkņu interpolāciju, ģenerēto līkņu modificēšanu un raksturojošo lielumu ievades un izvades veikšanu ar datorpelī, datora tastatūru vai nepieciešamo datu ielasīšanu no faila. Jānodrošina iespēja vizuāli salīdzināt konstruētās līknes ar līknēm jau esošā piegrieztnē.

1.5. Sistēmas lietotāji

Sistēmai paredzēts viens lietotāju veids. Lietotājs ir saistīts ar pasūtītāja uzņēmumu, ir pazīstams ar izstrādātā rīka specifiku (tai skaitā ar izmantoto Bezjē līkņu specifiku), un izmanto šo rīku, lai konstruētu līknes apģērbu piegrieztnēs. Lietotājs izmanto rīka saskarni un tam ir pieejamas visas programmatūras funkcionalitātes.

1.6. Vispārējie ierobežojumi

Nav.

1.7. Pieņēmumi un atkarības

Lai izmantotu visas programmatūras funkcionalitātes, lietotājam jābūt nodrošinātam ar datoru, ekrānu, datora tastatūru un datorpeli. Sistēma jāizstrādā C# programmēšanas valodā.

2. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA

2.1. Funkcionālās prasības

Programmatūrai jānodrošina pasūtītāja prasības:

1. Līkņu interpolācija caur patvaļīgi uzdotiem punktiem. Punkti var tikt ievadīti manuāli ar tastatūru, atzīmēti ar datorpeli vai ielasīti no faila.
2. Konstruētajām līknēm jābūt gludām (tām jāpiemīt C^2 nepārtrauktībai).
3. Konstruētās līknes jāspēj interaktīvi modifīcēt izmantojot datorpeli vai ievadot interpolējamo punktu izmaiņas ar tastatūru.
4. Jāspēj izvadīt konstruēto līkņu parametrus uz ekrāna un failā, lai nepieciešamības gadījumā varētu rekonstruēt izveidoto līkni.
5. Jānodrošina iespēja vizuāli salīdzināt rīkā konstruētās līknes ar līknēm esošā piegrieztnē.
6. Dažādu nozīmju grafiskajiem objektiem (piemēram, interpolācijas mezglu punktiem un palīgpunktiem) jābūt atšķiramiem ar vizuālo izskatu, piemēram, krāsojumu.
7. Jānodrošina iespēja konstruētās līknes dzēst.

2.2. Nefunkcionālās prasības

Grafisko objektu konstruēšanai, modifīcēšanai, dzēšanai un datu izvadei jānotiek bez pamanāmas aizķeršanās – ne ilgāk kā 0,2 sekunžu laikā [8].

3. PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS

3.1. Projektējuma matemātiskais pamatojums

3.1.1. Kubisku Bezjē līkņu izmantošanas pamatojums

Lai konstruētu divdimensionālus datorgrafiskus attēlus, parasti tiek izmantoti punkti, taisnes un līknes. Atšķirībā no dotiem punktiem un taisnēm, konkrētu līkņu attēlošana datorā nav triviāla, jo nepieciešama precīza matemātiska funkcija, kas šo līknī apraksta. Nepieciešamās līkņu funkcijas iespējams uzdot dažādos veidos – piemēram, ar polinomu un trigonometriskajām funkcijām, kā arī ar dažādām parametriskajām līknēm.

NURBS ir parametrisku līkņu uzdošanas veids, kas salīdzinoši viegli aprakstāms datoram un ir intuitīvi saprotams cilvēkam. NURBS veidotās līknes ir gludas, paredzamas un ērti modifcējamas, turklāt tām piemīt liela ekspresivitāte (atšķirībā no, piemēram, polinomu funkcijām, kur katram argumentam atbilst viena vērība) [9].

Viens no visplašāk izmantotajiem līkņu uzdošanas veidiem datorgrafikā ir Bezjē līknes [4], kas ir NURBS līkņu apakškopa. Lai gan mazāk elastīgas par NURBS līknēm, arī Bezjē līknes ir plaši pielāgojamas un paredzamas, turklāt samazinātās sarežģītības dēļ, tās ir vieglāk implementējamas, prasa mazāk resursus un ir vēl intuitīvākas. Galvenokārt tieši šo īpašību dēļ, Bezjē līknes ir implementētas lielākajā daļā grafisko programmatūru, piemēram, *Adobe Illustrator*, *GIMP*, *Inkscape* un citās.

Bezjē līknī iespējams definēt jebkurai pakāpei. Pirmās kārtas Bezjē līkne ir nogrieznis, savukārt ceturtās un augstāku kārtu Bezjē līknes patēri daudz skaitļošanas resursu. Tādēļ datorgrafikā vispopulārākās ir kvadrātiskās un kubiskās Bezjē līknes. Atceroties, ka līknes pakāpe nosaka arī tās maksimālo ekstrēmu un pārliekuma punktu skaitu, var secināt, ka kubiskās Bezjē līknes piedāvā lielāku pielietojamību nekā kvadrātiskās, turklāt tās ir implementētas C# System.Drawing bibliotēkā. Nemot vērā šos apsvērumus, programmatūra tiek projektēta izmantojot kubiskas Bezjē līknes.

Lai tiktu izpildītas visas funkcionālās prasības, programmatūrā tiek izmantoti četri kubiski Bezjē līkņu konstruēšanas veidi:

1. Uzdodot četrus kontrolpunktus (<4 cPoints>);
2. Uzdodot četrus mezglu punktus (<4 pPoints>);
3. Atrodot tuvāko līknī vairāk nekā četriem uzdotiem mezglu punktiem (<Least Squares>);

4. Veidojot saliktu Bezjē līkni (<Composite>).

3.1.2. Bezjē līkņu pamatteorija

Bezjē līkne ir parametriska funkcija, kas definēta jebkurai pakāpei un n -tās kārtas Bezjē līkne $B(t)$, $t \in [0,1]$ pierakstāma ar formulu

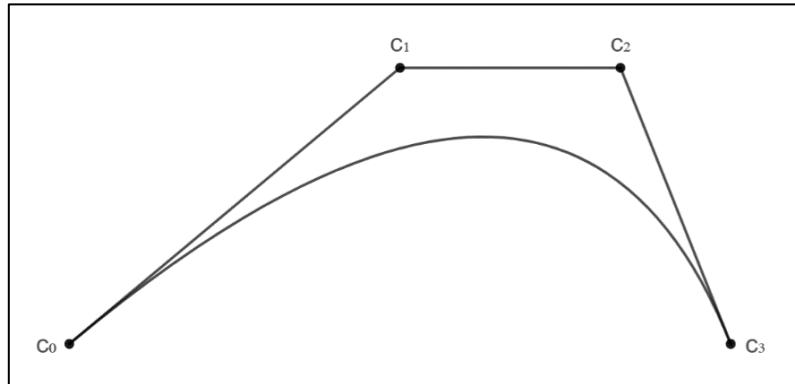
$$B(t) = \sum_{i=0}^n \binom{i}{n} (1-t)^{n-i} t^i \mathbf{c}_i .$$

Kubiska Bezjē līkne ir izsakāma formā

$$B(t) = (1-t)^3 \cdot \mathbf{c}_0 + 3(1-t)^2 t \cdot \mathbf{c}_1 + 3(1-t)t^2 \cdot \mathbf{c}_2 + t^3 \cdot \mathbf{c}_3 .$$

Ar \mathbf{c}_i tiek apzīmēti Bezjē līknes kontrolpunkti. Kontrolpunktji ir mainīgie, kas nosaka, kā līkne tiek izliekta. Katru kontrolpunktu komplekts nosaka tieši vienu līkni. Varam ievērot, ka kontrolpunktji sākas ar indeksu 0.

Trešās pakāpes Bezjē līknei ir četri kontrolpunktji, divus no tiem sauc par galapunktiem (\mathbf{c}_0 , \mathbf{c}_3) un divus par rokturiem (\mathbf{c}_1 , \mathbf{c}_2). Līknes sākumpunkts ir \mathbf{c}_0 un beigu punkts ir \mathbf{c}_3 , līdz ar to šie punkti pieder līknei. Atšķirībā no galapunktiem, patvalīgi izvēlēti rokturu punkti līknei nepieder. Skatīt 3.1. att.



3.1. att. Bezjē līkne un tās kontrolpunktji

Bezjē līknes iespējams reprezentēt arī matricu formā, kas atvieglo dažādu aprēķinu veikšanu. Kubisku Bezjē līkni var izteikt kā trīs matricu \mathbf{T} , \mathbf{M} , \mathbf{C} reizinājumu.

$$\begin{aligned} B(t) &= (1-t)^3 \cdot \mathbf{c}_0 + 3(1-t)^2 t \cdot \mathbf{c}_1 + 3(1-t)t^2 \cdot \mathbf{c}_2 + t^3 \cdot \mathbf{c}_3 = \\ &= (1 - 3t + 3t^2 - t^3)\mathbf{c}_0 + (3t - 6t^2 + 3t^3)\mathbf{c}_1 + (3t^2 - 3t^3)\mathbf{c}_2 + t^3\mathbf{c}_3 = \end{aligned}$$

$$\begin{aligned}
&= (1 - 3t + 3t^2 - t^3)\mathbf{c}_0 + \\
&+ (0 + 3t - 6t^2 + 3t^3)\mathbf{c}_1 + \\
&+ (0 + 0t + 3t^2 - 3t^3)\mathbf{c}_2 + \\
&+ (0 + 0t + 0t^2 + 1t^3)\mathbf{c}_3 = \\
\\
&= (1 \quad t \quad t^2 \quad t^3) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 0 & 0 \\ 1 & 3 & -3 & 1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2 \\ \mathbf{c}_3 \end{pmatrix} = \\
&= \mathbf{T} \cdot \mathbf{M} \cdot \mathbf{C}
\end{aligned}$$

3.1.3. Kubiskās Bezjē līknes interpolācija caur četriem punktiem

Lai veiktu interpolāciju ar Bezjē līkni, t.i., lai uzkonstruētu līkni, kas iet cauri dotiem mezglu punktiem, jāiegūst atbilstošas kontrolpunktu koordinātas. Trešās kārtas gadījumā līkne precīzi var iziet caur ne vairāk kā četriem patvalīgiem mezglu punktiem.

Ja lietotājs vēlas novilkst līkni caur tieši četriem punktiem, iespējams atrast atbilstošos kontrolpunktus. Viens no variantiem, kā atrast šādu kontrolpunktu koordinātas, ir izmantojot mazāko kvadrātu metodi. Šī metode atrod kontrolpunktu koordinātas, minimizējot attālumus starp dotajiem mezglu punktiem un līkni. Zināms, ka trešās pakāpes Bezjē līkni iespējams izvilkst caur jebkādiem četriem mezglu punktiem, tātad mazāko kvadrātu metode atradīs līkni, kurai attālumu kvadrātu summa ir nulle.

Bezjē līkne ir parametriska un, lai aprakstītu punktu uz tās, tiek specificēta t vērtība, savukārt mezglu punkti tiek uzdoti ar koordinātām. Lai mērītu un minimizētu attālumu starp mezgla punktu un līkni, nepieciešams saistīt dotos punktus ar punktiem uz līknes, tas ir, katram mezgla punktam nepieciešams piesaistīt t vērtību. Šo procesu sauc par parametrizāciju (skatīt nodaļu 3.1.7) un tās rezultātā atrasto t vērtību katram mezgla punktam p_i apzīmēsim ar s_i .

No šīm s_i vērtībām tiek izveidota matrica \mathbf{S} , kas interpolācijas aprēķinos aizstāj matricu \mathbf{T} ar simboliskajām t vērtībām [10].

$$\mathbf{S} = \begin{pmatrix} s_0^0 & s_0^1 & s_0^2 & s_0^3 \\ s_1^0 & s_1^1 & s_1^2 & s_1^3 \\ s_2^0 & s_2^1 & s_2^2 & s_2^3 \\ s_3^0 & s_3^1 & s_3^2 & s_3^3 \end{pmatrix}$$

Lai aprēķinātu kontrolpunktus (sakārtotus matricā \mathbf{C}), kuru veidotā līkne interpolē mezglu punktus (sakārtotus matricā \mathbf{P}), jāveic sekojoši aprēķini:

$$\mathbf{C} = \mathbf{M}^{-1}(\mathbf{S}^{-1}\mathbf{S})^T\mathbf{S}^T\mathbf{P}.$$

Aprēķinu izvedumu un mazāko kvadrātu metodes pierādījumu Bezjē līkņu interpolācijai skatīt 2. pielikumā.

3.1.4. Tuvākās līknes metodes izmantošanas pamatojums un teorija

Ja lietotājs vēlas novilkst līkni caur vairāk nekā četriem punktiem, viens no iespējamajiem risinājumiem ir atrast tuvāko līkni dotajiem mezglu punktiem – līkni, kura ir pēc iespējas tuvu visiem punktiem. Tas izdarāms dažādos veidos, bet viens no ērtākajiem un plašāk izmantotajiem veidiem ir mazāko kvadrātu metode. Metodes apraksts sakrīt ar četru punktu interpolāciju (skatīt 3.1.3 nodaļu), tikai n patvaļīgu punktu gadījumā ($n > 4$), attālumu kvadrātu summa nebūs nulle un matrica \mathbf{S} būs formā

$$\mathbf{S} = \begin{pmatrix} s_0^0 & s_0^1 & s_0^2 & s_0^3 \\ s_1^0 & s_1^1 & s_1^2 & s_1^3 \\ \vdots & \vdots & \vdots & \vdots \\ s_{n-1}^0 & s_{n-1}^1 & s_{n-1}^2 & s_{n-1}^3 \end{pmatrix}$$

Tuvākā līkne analītiski neizies cauri visiem dotajiem punktiem, tomēr, ņemot vērā apgērba piegrieztņu modelēšanas specifiku – nepieciešamo precizitāti, iespējamās nobīdes un salīdzinoši mazās virziena izmaiņas līknēm, tā ir noderīga, jo prasa maz resursu un ir lietotājam vienkārši saprotama.

3.1.5. Saliktas līknes izmantošanas pamatojums un teorija

Ja lietotājs vēlas novilkst līkni caur vairāk nekā četriem punktiem, un ir svarīgi, ka konstruētā līkne analītiski iziet caur šiem punktiem, patvaļīgiem mezgliem tas nav izdarāms ar vienu kubisku Bezjē līkni. Tā vietā var izmantot vairākas Bezjē līknes, kuras savā starpā ir savienotas. Katru no šīm savienotajām līknēm sauc par segmentu. Lai gan iespējams katriem četriem punktiem uzdot vienu Bezjē līkni, tādējādi minimizējot segmentu skaitu, šāds sadalījums rada daudz speciālgadījumu (ja mezglu skaits nedalās ar četri), būtiski sarežģī aprēķinus un ierobežo pielāgošanas iespējas. Ērtāks veids, kā ieviest saliktas Bezjē līknes, ir uzdot jaunu līkni katriem

diviem secīgiem mezglu punktiem, kur mezglu punkti ir Bezjē līknes galapunkti. Lietojot šo metodi, nav jāuztraucas par mezglu punktu piederību līknēm.

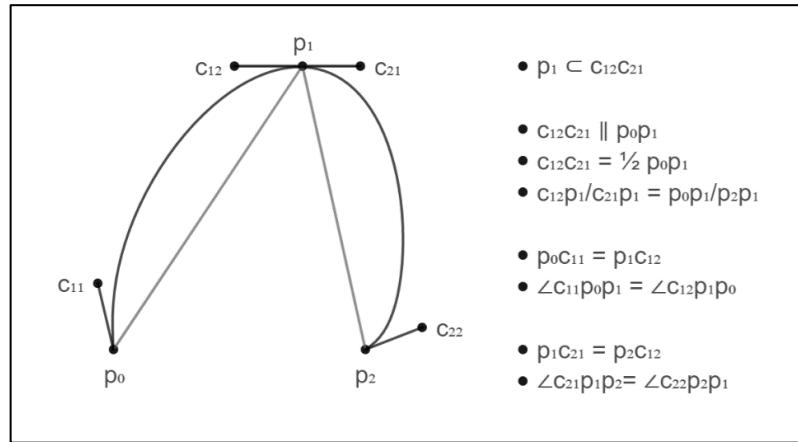
Lai savienotās Bezjē līknes būtu gludas, tām jāpiemīt C^2 nepārtrauktībai. Lai nodrošnātu šo nepārtrauktību, katram mezglu punktam (izņemot pirmo un pēdējo) un tā blakus rokturiem jābūt uz vienas taisnes. Šāda veida saliktas Bezjē līknes sauc par B-splainiem, un tās ir iespējams konstruēt dažādos veidos. Kādā lenķī un cik tālu no mezgla punkta ir blakus rokturi ir maināmi lielumi un raksturo B-splainu veidu.

3.1.6. B-splainu konstruēšanas un modificēšanas metode

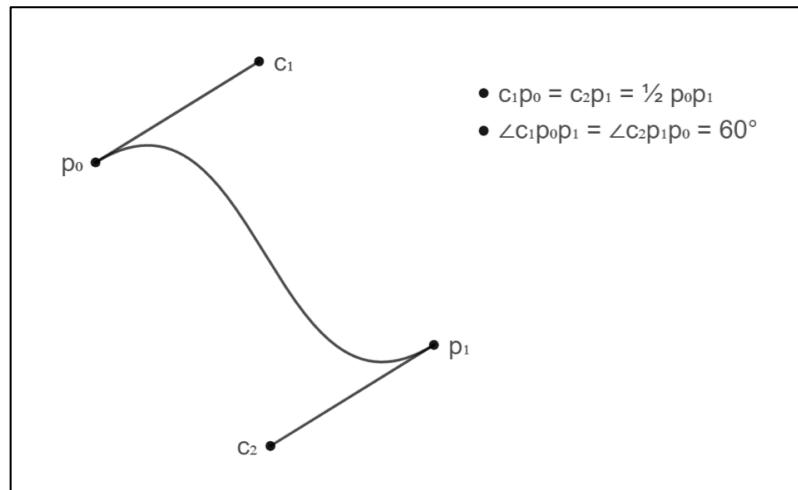
B-splainu veids, kas tiek konstruēts izstrādātajā rīkā, aprakstāms šādi:

- Katrs mezgla punkts (izņemot pirmsākotā un pēdējā) un tā blakus rokturi ir uz vienas taisnes (skatīt 3.2. att.);
- Mezgla punkta blakus rokturu veidotā taisne ir paralēla blakus mezglu veidotajai taisnei (skatīt 3.2. att.);
- Attālums starp blakus rokturiem vienāds ar pusi no attāluma starp blakus mezgliem (skatīt 3.2. att.);
- Attālums starp mezglu un blakus rokturiem proporcionāls attālumam starp mezglu un blakus mezgliem (skatīt 3.2. att.);
- Pirmsākotās B-splains rokturis tiek atliks simetriski otrajam/priekšpēdējam B-līknes rokturim (skatīt 3.2. att.);
- Ja B-splains sastāv no 2 mezglu punktiem, attālums no roktura līdz tam tuvākajam mezgla punktam vienāds ar pusi no attāluma starp mezglu punktiem; lenķis starp mezglu punktu veidoto taisni un taisni, ko veido rokturis un tā tuvākais mezglu punkts, vienāds ar 60° (skatīt 3.3. att.).

Saliktās Bezjē līknes programmatūrā tiek konstruētas kā aprakstīts, jo šis B-splainu veids ir cilvēkam intuitīvs, tā implementācija ir vienkārša un neprasā daudz resursu. Rīka konstruētās saliktās līknes ir gludas un, izņemot retus gadījumus (piemēram, asus pagriezienus), atbilstošas vizuālajām prasībām.



3.2. att. B-splainu konstruēšanas metodes piemērs



3.3. att. B-splainu konstruēšanas piemērs divu mezglu gadījumā

3.1.7. Parametrizācijas metodes

Bezjē līkņu parametrizācija ir mezglu punktu koordinātu sasaiste ar Bezjē līkni $\mathbf{B}(t)$. Parametrizācijas rezultātā katram mezglam p_i tiek atrasta atbilstoša t vērtība, apzīmēta ar s_i . Parametrizācija nepieciešama, lai veiktu Bezjē līkņu interpolāciju. Caur četriem punktiem iespējams izvilkst bezgalīgi daudz kubisku Bezjē līkņu un tieši parametrizācijas metode nosaka, kāda līkne tiks izvilkta. Trīs parametrizācijas metodes, kas tiek izmantotas programmatūrā, ir:

1. Vienmērīga sadalījuma parametrizācijas metode (*uniform method*).

Tiek pieņemts, ka dotie punkti būs vienmērīgi izkārtoti, un atbilstošā t vērtība punktam p_i aprēķināma kā $s_i = \frac{i}{n-1}$, kur n – mezglu skaits. Piemēram, ja tiek doti 5 punkti, attiecīgās s vērtības

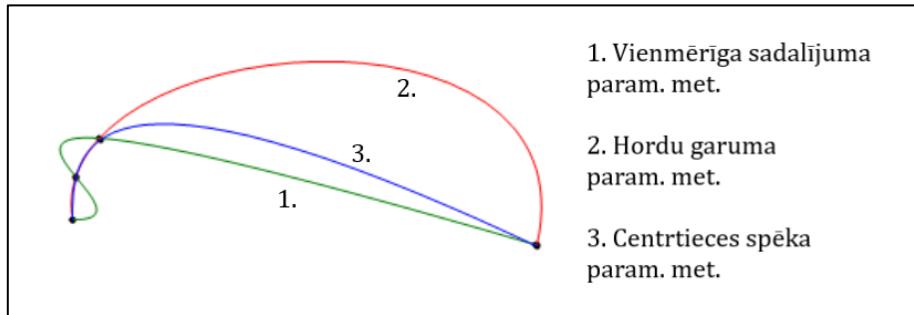
ir 0, 0.25, 0.5, 0.75, 1. Šī ir viena no vienkāršākajām un skaitļošanas resursu taupīgākajām parametrizācijas metodēm, bet tā var dot nevēlamus rezultātus – piemēram, ja mezgli nav vienmērīgos attālumos, līknē var rasties lieki izliekumi, cilpas un asas smailes, skatīt 3.4. att. Lai gan šādi nevēlami rezultāti nav unikāli vienmērīgā sadalījuma metodei, tie atgadās biežāk nekā citām metodēm. [11]

2. Hordu garuma parametrizācijas metode (*chord length method*).

Tiek pieņemts, ka konstruētās līknes punkti būs netālu no lauztās līnijas, kas veidojas savienojot mezglu punktus. Tādā gadījumā līknes garums starp diviem mezgliem būtu tuvs hordas garumam starp tiem un visas interpolētās līknes garums būtu tuvs minētajai lauztajai līnijai. Atbilstošo t vērtību punktam \mathbf{p}_i atrod kā lauztās līnijas garumu no pirmā līdz i -tajam mezglam, mērogotu $[0, 1]$ intervālā. Aprēķināms kā $s_i = s_{i-1} + \frac{|\mathbf{p}_i - \mathbf{p}_{i-1}|}{l}$, kur $s_0 = 0$, ar $|\mathbf{p}_i - \mathbf{p}_{i-1}|$ apzīmē garumu starp dotajiem punktiem, $l = \sum_{i=1}^{n-1} |\mathbf{p}_i - \mathbf{p}_{i-1}|$ – lauztās līnijas garums, n – mezglu skaits. Tai nav apjomīgas skaitļošanas operācijas un visbiežāk līnijas, kas konstruētas izmantojot hordu garuma parametrizāciju, dod vēlamus rezultātus. Kā novērojams arī 3.4. attēlā, bieža nevēlama parādība šai metodei ir lieli izliekumi [12]. Šī un vienmērīgā sadalījuma metode ir vispopulārākās Bezjē līkņu parametrizācijas metodes [13].

3. Centrīces spēka parametrizācijas metode (*centripetal method*).

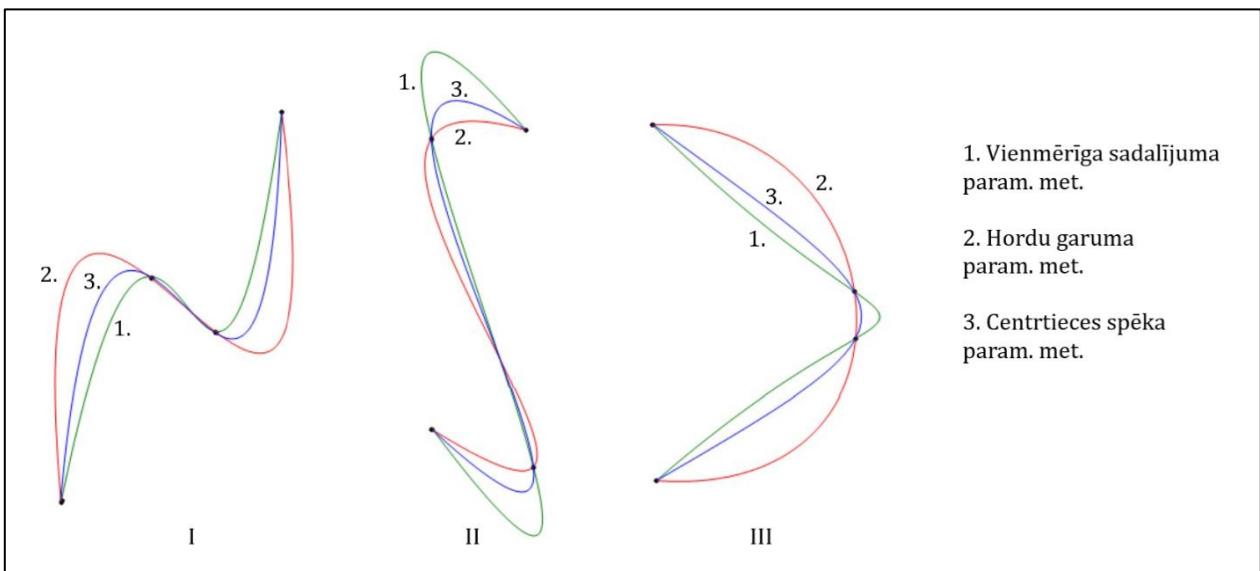
Šī parametrizācijas metode atsaucas uz centrīces spēku. Ja mēs iedomājamies mašīnu, kas brauc pa Bezjē līkni, pagriezienos tai būtu jāsamazina ātrums, lai centrīces spēks nav pārlieku liels un tā neizslīd no trajektorijas – spēkam vajadzētu būt proporcionālam izmaiņām leņķi. Centrīces spēka parametrizācija ir aproksimācija šim modelim. Atbilstošo t vērtību punktam \mathbf{p}_i aprēķina kā $s_i = s_{i-1} + \frac{\sqrt{|\mathbf{p}_i - \mathbf{p}_{i-1}|}}{l}$, kur $s_0 = 0$, ar $|\mathbf{p}_i - \mathbf{p}_{i-1}|$ apzīmē garumu starp dotajiem punktiem, $l = \sum_{i=1}^{n-1} \sqrt{|\mathbf{p}_i - \mathbf{p}_{i-1}|}$, n – mezglu skaits. Var ievērot, ka šī metode ir ļoti līdzīga hordu garuma metodei, bet kvadrātsaknes dēļ garākām hordām tiek samazināta ietekme, savukārt īsākām – palielināta. Kā redzams arī 3.4. attēlā, minētā iemesla dēļ centrīces spēka parametrizācija labāk tiek galā ar asiem pagriezieniem [14].



3.4. att. Parametrizācijas metožu raksturiezīmes

Visām parametrizācijas metodēm ir savas priekšrocības un trūkumi un nav viena labākā metode, it īpaši ņemot vērā, ka līknes labums ir daļēji subjektīvs. Kā redzams 3.5. attēlā, veicot interpolāciju caur dažādiem mezglu punktu izvietojumiem, piemērotākā parametrizācijas metode (tā, kuras veidotā Bezjē līkne ir vistuvākā mezglu punktu veidotajai lauztajai līnijai) var būt gan vienmērīga sadalījuma (I), gan hordu garuma (II), gan centrtieces spēka (III). Šo iemeslu dēļ, izstrādātais rīks ļauj mainīt interpolēto līkņu parametrizācijas metodi.

Eksistē parametrizācijas metodes, kuru konstruētās līknes retāk ir nevēlamas, piemēram, afīnā invarianta leņķa metode (“affine invariant angle method”), taču tās ir sarežģīti implementējamas un prasa daudz skaitļošanas resursu [15].

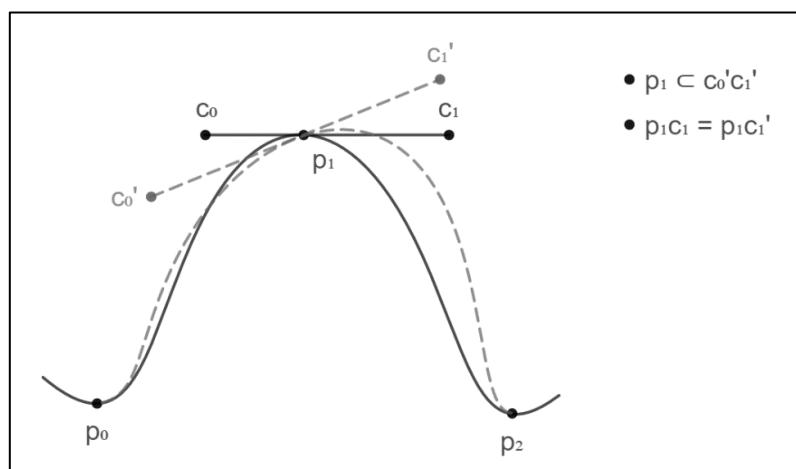


3.5. att. Parametrizācijas metožu salīdzinājums

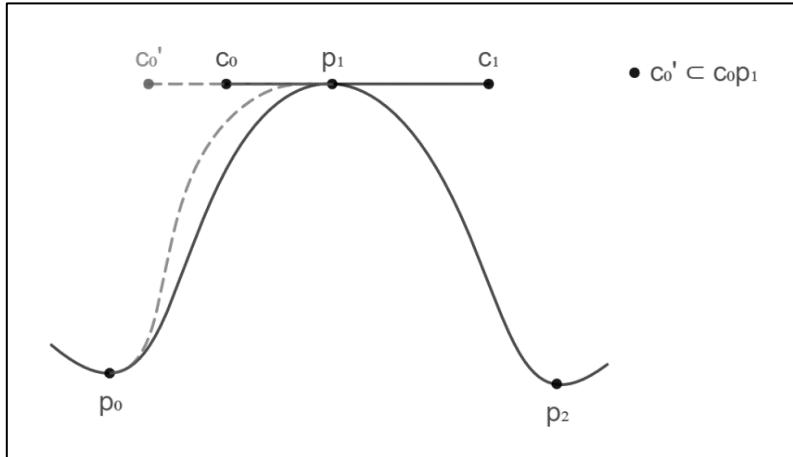
3.1.8. Līkņu modificēšanas metodes

Lai nodrošinātu pilnvērtīgu rīka darbību, konstruētās līknes jāspēj modificēt. Modificēšanas veidi un iespējas atkarīgas no līknes konstruēšanas un modificēšanas veida:

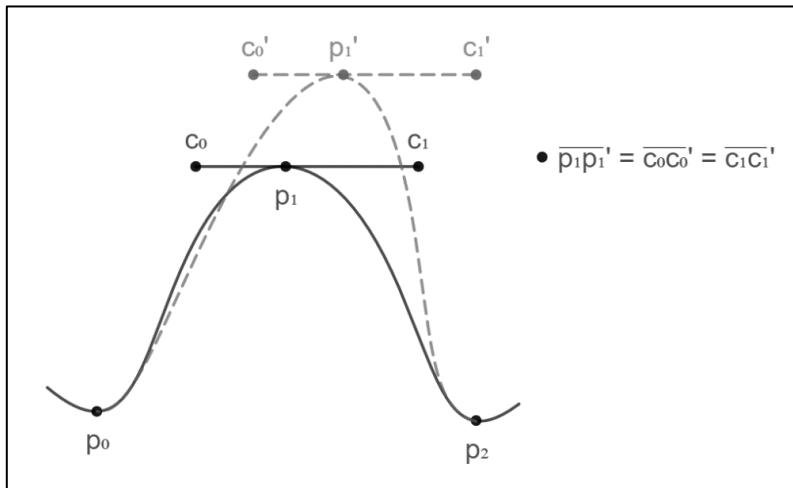
- Līknes `<4 cPoints>` iespējams modificēt, patvaļīgi pārvietojot līknes kontrolpunktus.
- Līknes `<4 pPoints>` un `<Least Squares>` iespējams modificēt, patvaļīgi pārvietojot līknes mezglu punktus.
- Līkņu `<Composite>` iespējams modificēšana iespējama trīs veidos:
 - Patvaļīgi pārvietojot līknes rokturi. Roktura, kurš tiek modificēts, pretējais rokturis arī maina atrašanās vietu, lai abi rokturi un mezgla punkts starp tiem vienmēr atrastos uz vienas taisnes. Šis nosacījums nepieciešams, lai saglabātu līknes C^2 nepārtrauktību. Attālums starp mezglu un pretējo rokturi paliek nemainīgs. Skatīt 3.6. att.
 - Ierobežoti pārvietojot līknes rokturi. Šajā modificēšanas veidā rokturi iespējams pārvietot tikai pa staru, kas atrodas uz taisnes, ko veido roktura tuvākais mezgla punkts un pretējais rokturis. Stara sākumpunkts ir minētais mezgls un pretējais rokturis staram nepieder. Šāds modificēšanas veids nodrošina, ka atrašanās vietu maina tikai izvēlētais rokturis, bet līkne nezaudē C^2 nepārtrauktību. Skatīt 3.7. att.
 - Patvaļīgi pārvietojot mezgla punktu. Lai līknei arī pēc mezgla modificēšanas piemistu C^2 nepārtrauktību, bet pārējie līknes segmenti modifikācijas rezultātā nemainītos, ērtākais veids ir nemainīt mezgla blakus rokturu relatiivo atrašanās vietu pret minēto mezglu. Skatīt 3.8. att.



3.6. att. Saliktas līknes kontrolpunkta c_0 patvaļīga modificēšana



3.7. att. Saliktas līknes kontrolpunkta c_0 ierobežota modificēšana



3.8. att. Saliktas līknes mezgla punkta p_1 modificēšana

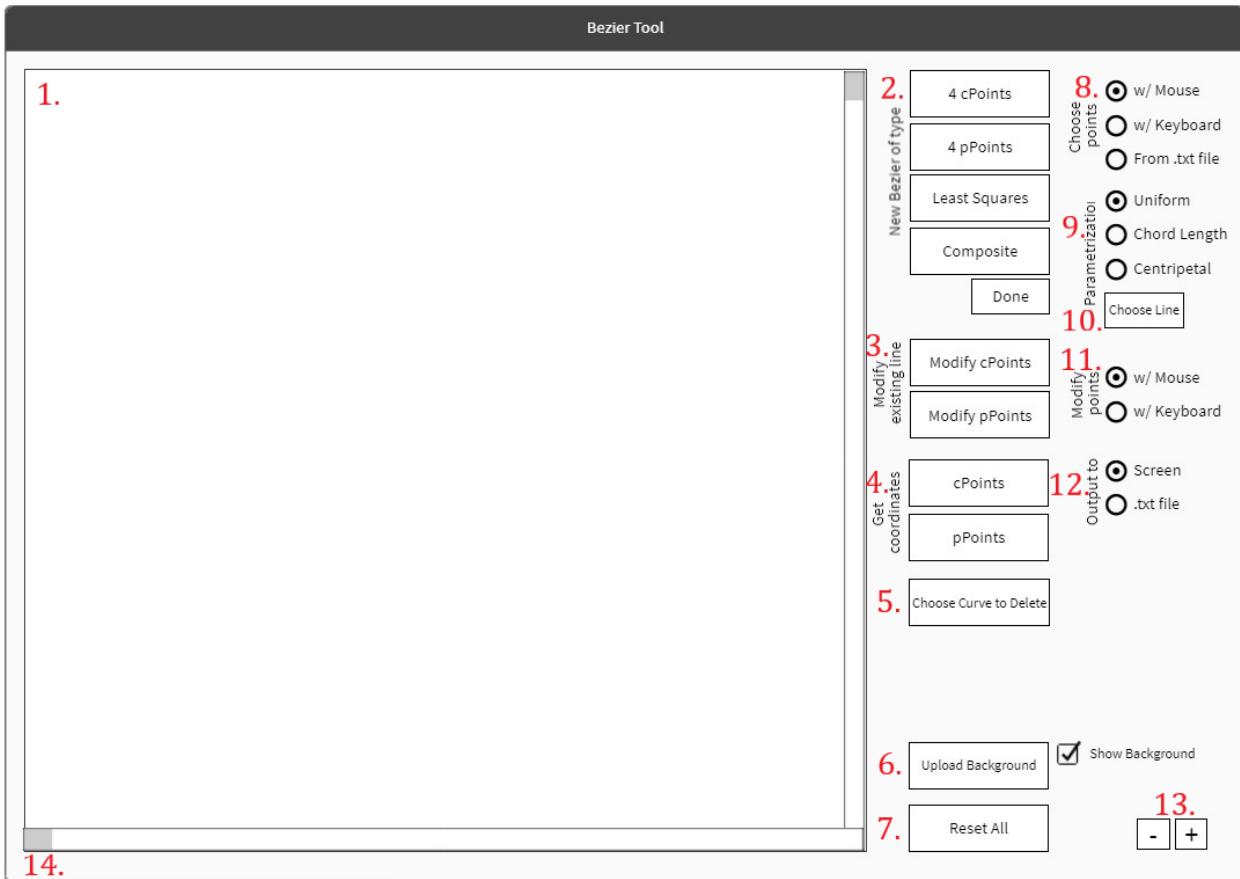
3.2. Lietotāja saskarnu projektējums

Programmatūrai ir divi izvades logi – galvenais logs (skat. 3.9. att.) un koordinātu logs (skat. 3.10. att.). Lielākā daļa funkcionalitāšu noris galvenajā logā un koordinātu logs tiek no tā izsaukts tikai nepieciešamības gadījumā – lai ievadītu vai modificētu punktu koordinātas ar datora tastatūru un punktu koordināšu izvadei uz ekrāna.

3.2.1. Galvenais logs

Galvenais logs paredzēts, visu rīkā izveidoto grafisko objektu attēlošanai un visu funkcionalitāšu darbības sākšanai (uzspiežot uz atbilstošās pogas). Vairākas funkcionalitātes

realizējamas izmantojot tikai galveno logu – jaunu līniju konstruēšana un esošu līniju modifīcēšana ar peli (tai skaitā parametrizācijas izvēle un maiņa), esošu punktu izvēle, līkņu dzēšana, loga inicializācija no jauna, fona attēla redzamības maiņa. Jaunu līkņu konstruēšana no .txt faila, esošu punktu koordinātu izvade uz .txt failu un fona attēla augšupielādēšana arī neizmanto koordinātu logu. Galvenais logs ir uz tā izmēra maiņu reaģējošs, tam piesaistītas funkcijas, kas aprakstītas 3.3.1.-3.3.4. nodaļā.



3.9. att. Galvenā loga saskarne

Galvenajā logā ietvertas vadīkļas:

1. – Lauks, kas paredzēts fona attēla un visu grafisko objektu attēlošanai. Reaģējoši maina izmēru, kad tiek veikta galvenā loga izmēra maiņa, kā arī maina izmēru pēc lietotāja pieprasījuma, spiežot 13. vadīku grupas pogas. Nepieciešamības gadījumā lauka labajā un apakšējā malā kļūst redzamas ritjoslas.

2. – Virsraksts “New Bezier of type”, pie kura atrodas piecas pogas. Pirmo četru pogu “4 cPoints”, “4 pPoints”, “Composite” un “Least Squares” nospiešana izsauc attiecīgi funkcijas

NC02_b4C, NC02_b4P, NC02_bLS un NC02_bC. Pēc šo pogu nospiešanas iespējams pievienot jaunu līkni. Visu līkņu punktu ievades veids atkarīgs no 8. vadīklu grupas radio pogu statusiem, interpolēto līkņu parametrizācijas metode atkarīga no 9. vadīklu grupas radio pogu statusa. Piektā poga “Done” izsauc funkciju NC06_bDC un apzīmē, ka pēdējā konstruētā saliktā līkne ir pabeigta.

3. – Virsraksts “Modify existing curve”, pie kura atrodas divas pogas – “Modify cPoints” un “Modify pPoints”. Pogu nospiešana izsauc attiecīgi funkcijas MC05_bMC un MC06_bMP un pēc šīs darbības iespējams modificēt attiecīgi esošu līkņu kontrolpunktus vai mezglu punktus. Modificēšanas veids atkarīgs no 10. vadīklu grupas radio pogu statusiem. Detalizētāk par modificēšanas iespējām aprakstīts 3.1.8. nodaļā.

4. – Virsraksts “Get coordinates”, pie kura atrodas divas pogas – “cPoints” un “pPoints”. Pogu nospiešana izsauc attiecīgi funkcijas EC02_bOC un EC03_bOP un ļauj izvēlēties līkni, kuras kontrolpunktus vai mezglu punktus izvadīt. Izvades veids atkarīgs no 11. vadīklu grupas radio pogu statusiem.

5. – Poga “Choose Curve to Delete”. Pogas nospiešana izsauc funkciju EC06_bDL un ļauj izvēlēties līkni, ko izdzēst.

6. – Poga “Upload Background” un izvēles rūtiņa “Show Background”. Nospiežot minēto pogu, tiek izsaukta funkcija FM06_bUB, kas ļauj augšupielādēt attēlu. Augšupielādētais attēls tiek attēlots 1. vadīklā un tā redzamību iespējams kontrolēt ar minēto izvēles rūtiņu – mainot tās statusu, tiek izsaukta funkcija FM07_cSB. Veicot loga inicializāciju, izvēles rūtiņa ir neaktīva.

7. – Poga “Reset A11”. Nospiežot šo pogu tiek izsaukta funkcija FM11_bRA, kas inicializē galveno logu no jauna.

8. – Virsraksts “Choose points”, pie kura atrodas grupa ar trīs radio pogām – “w/ Mouse”, “w/ Keyboard” un “From .txt file”. Radio pogu statusi nosaka veidu, kā līknei tiks pievienoti punkti – tos atzīmējot ar datorpeli 1. vadīklā, ievadot punktu koordinātas ar datora tastatūru (tieki izsaukts koordinātu logs) vai ielasot punktu koordinātas no .txt faila. Veicot loga inicializāciju, aktīva ir pirmā no radiopogām.

9. – Virsraksts “Parametrization”, pie kura atrodas grupa ar trīs radio pogām – “Uniform”, “Chord Length”, “Centripetal” un 10. vadīkla. Minētās radio pogas nosaka interpolētu līkņu parametrizācijas metodi, attiecīgi vienmērīga sadalījuma, hordu garuma vai centrtieces spēka. Mainot pirmo divu radio pogu statusu, tiek izsauktas funkcijas MC03_rUC vai MC04_rCC, kas maina izvēlētas līknes parametrizācijas metodi. Detalizētāk par parametrizācijas metodēm aprakstīts 3.1.7. nodaļā. Veicot loga inicializāciju, aktīva ir pirmā no radiopogām.

10. – Poga “Choose Curve”. Nospiežot pogu tiek izsaukta funkcija MC01_bCP, kas ļauj mainīt interpolētu līkņu parametrizācijas metodi, kā arī attēlo izvēlētās līknes parametrizācijas metodi, aktivizējot attiecīgo radio pogu no 9. vadīklu grupas. Veicot loga inicializāciju, aktīva ir pirmā no radiopogām.

11. – Virsraksts “Modify points”, pie kura atrodas grupa ar divām radio pogām – “w/ Mouse” un “w/ Keyboard”. Radio pogu statusi nosaka, kā tiks modificēti līknes punkti – tos pārvietojot ar peli vai ievadot punktu koordinātas ar datora tastatūru (tieka izsaukts koordinātu logs).

12. – Virsraksts “Output to”, pie kura atrodas grupa ar divām radio pogām – “Screen” un “.txt file”. Radio pogu statusi nosaka, vai līknes punkti tiks izvadīti uz ekrāna (izsaucot koordinātu logu) vai saglabāti .txt failā. Veicot loga inicializāciju, aktīva ir pirmā no radiopogām.

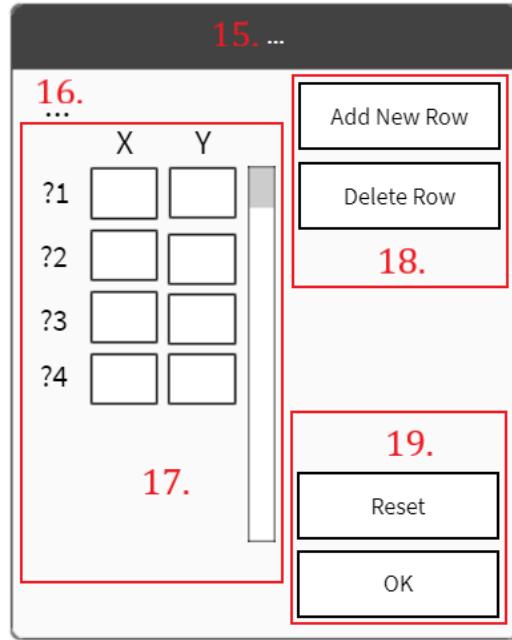
13. – Divas pogas – “-” un “+”. Nospiežot pogas attiecīgi tiek izsauktas funkcijas FM09_bZO un FM10_ZI, kas maina 1. lauka un tajā esošo grafisko objektu izmēru.

14. – Lauks kļūdu paziņojumiem. Paziņojumi aprakstīti 3.2.3 nodaļā.

3.2.2. Koordinātu logs

Koordinātu logs paredzēts punktu koordinātu ievadīšanai vai izvadīšanai. Šis logs tiek izsaukts no galvenā loga, tam ir piesaistītas 3.3.5. nodaļā aprakstītās funkcijas un tā izsaukšanai iespējami trīs mērķi:

1. Koordinātu logs paredzēts punktu ievadei ar datora tastatūru, ja, veicot jaunas līknes pievienošanu (nospiežot kādu no pirmajām četrām 2. vadīklu grupas pogām), radio poga “w/ Keyboard” no 8. vadīklu grupas ir aktīva. Logam atveroties, tiks izsaukta funkcija FC01_IA.
2. Koordinātu logs paredzēts punktu koordinātu modifikācijai ar datora tastatūru, ja, veicot esošas līknes modificēšanu (nospiežot kādu no 3. vadīklu grupas pogām), radio poga “w/ Keyboard” no 11. vadīklu grupas ir aktīva. Logam atveroties, tiks izsaukta funkcija FC02_IM.
3. Koordinātu logs paredzēts punktu koordinātu izvadei uz ekrāna, ja, veicot esošas līknes punktu izvadi (nospiežot kādu no 4. vadīklu grupas pogām), radio poga “Screen” no 12. vadīklu grupas ir aktīva. Logam atveroties, tiks izsaukta funkcija FC03_IO.



3.10. att. Koordinātu loga saskarne

15. – Koordinātu loga virsraksts. Tā saturis atkarīgs no loga izsaukšanas mērķa – “New <curveType> curve”, “Modify <curveType> curve” vai “Output <curveType> curve”; <curveType> vietā ir izvēlētās līknes konstruēšanas veida nosaukums (skatīt 3.1.2. nodaļu).

Bez tā, koordinātu logā vēl ietvertas vadīklas:

16. – Virsraksts. Tā saturis atkarīgs no loga izsaukšanas mērķa – “Input <curveType> <pointType> point coordinates:”, “Modify <curveType> <pointType> point coordinates:” vai “List of <curveType> <pointType> point coordinates:”; <curveType> vietā ir izvēlētās līknes konstruēšanas veida nosaukums (skatīt 3.1.2. nodaļu), <pointType> vietā ir “control” kontrolpunktu gadījumā un “knot” mezglu punktu gadījumā.

17. – Tabula ar vadīklām un ritjosla. Tabulas augšpusē ir virsraksti “X” un “Y”, savukārt pirmā kolonna ir aizpildīta ar virsrakstiem formā “<pointType><i>”; <pointType> vietā ir “C” kontrolpunktu gadījumā un “P” mezglu punktu gadījumā, <i> vietā ir skaitlis, kas apzīmē rindiņu. Blakus šiem virsrakstiem ir teksta lauki, kuros pēc nepieciešamības tiek ievadītas vai izvadītas punktu koordinātas. Inicializējot koordinātu formu līkņu pievienošanai, tiek izveidotas četras teksta lauku rindiņas, veicot punktu izvadi vai modifikāciju, teksta lauku rindiņu skaits sakrīt ar līknes apskatīto punktu skaitu, izņemot saliktu līkņu mezglu punktu modifcēšanas gadījumā, kas tiek veikta pa vienam punktam.

18. – Pogas “Add New Row” un “Delete Row”. Nospiežot pogas tiek attiecīgi izsauktas funkcijas FC05_bAR un FC06_bDR. Šīs pogas ir redzamas un izmantojamas tikai pievienojot <Least Squares> vai <Composite> līknes un tās 15. vadīklu grupas tabulai pievieno jaunas rindas, kas tiek aizpildītas ar nepieciešamajām vadīklām. Tieki ievēroti punktu skaita ierobežojumi.

19. – Pogas “Reset” un “OK”. Nospiežot pogas tiek izsauktas attiecīgi funkcijas FC08_bRI un FC07_bSI. Pogas nav redzamas, ja koordinātu logs izsaukts ar mērķi izvadīt esošas līknes punktu koordinātas. Poga “Reset” nodzēš visus teksta laukus 15. vadīklu grupā, savukārt poga “OK” pievieno jaunu līkni vai modifīcē izvēlēto.

3.2.3. Klūdu un citi paziņojumi

Klūdu un citi paziņojumi tiek izvadīti, ja sistēmas darbībā ir kļūda vai lietotājs veic neatļautas darbības. Paziņojumi var tikt izvadīti atsevišķā tam paredzētā logā (4., 8.-10. paziņojums) vai arī galvenā logā 13. laukā (1.-3., 5.-7. paziņojums). Iespējamie paziņojumi:

1. File upload error!
2. .txt file was not correct!
3. Output error!
4. Do you want to reset this form?
5. Not allowed to move control points of <curveType> curve!
6. Not allowed to move control points of <Composite> curve by keyboard!
7. <curveType> curves does not use parametrization!
8. Maximum count of input points is <pointCount>!
9. <curveType> curves can not have less than <pointCount> knot points!
10. Must fill all values!

3.3. Funkciju projektējums

Esošie pasūtītāja uzņēmuma informāciju tehnoloģiju projekti tiek izstrādāti C# valodā, tāpēc arī kvalifikācijas darbs tiks implementēts C# valodā. Izpētot C# iespējas, par piemērotāko programmēšanas rīku tika atzīts Microsoft .NET satvars un tajā iekļautā Windows Forms bibliotēka. Šie rīki piedāvā plašu vadīklu izvēli, tai skaitā vadīklas grafisku objektu attēlošanai, un saskarņu elementus – piemēram, tekstus, krāsas, darbības ar datorpeli.

3.3.1. Galvenā loga pamatfunkcijas

Galvenā loga pamatfunkcijas iekļauj visas tās funkcijas, kas saistītas ar vadīklu, kurā tiek attēloti visi grafiskie objekti (pbCanva), kā arī funkcijas, kas saistītas tikai ar galveno logu – tā inicializāciju no jauna un reaģēšanu uz izmēra maiņu. Visu funkciju uzskaitē un īsi apraksti atrodami 3.1. tabulā, sarežģītākajām funkcijām pieejami arī detalizētāki apraksti. Klūdu paziņojumu saraksts atrodams 3.2.3. nodaļā.

3.1. tabula – Galvenā loga pamatfunkcijas

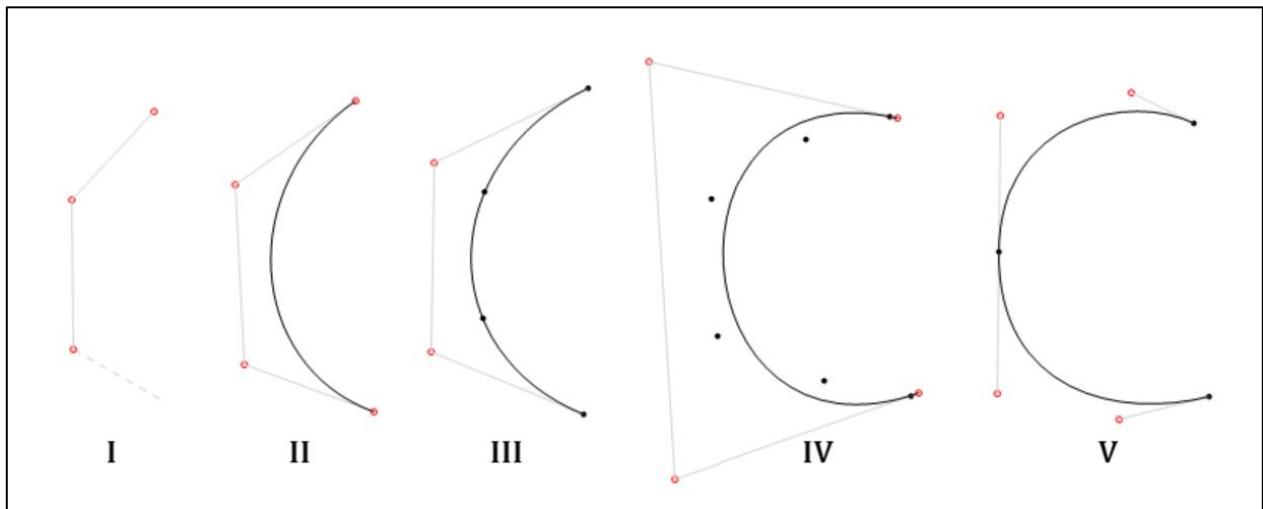
Nosaukums	Funkcija	Apraksts
FM01_pMD	<code>void pbCanva_MouseDown(object sender, MouseEventArgs e)</code>	Izpildās, kad vadīklā pbCanva tiek nospiests peles kursotrs. Īlauj pievienot jaunus vai izvēlēties eksistējošus kontrolpunktus un mezglu punktus.
FM02_pMM	<code>void pbCanva_MouseMove(object sender, MouseEventArgs e)</code>	Izpildās, kad vadīklā pbCanva tiek kustināts peles kursotrs. Īlauj veikt interaktīvu līkņu modificēšanu ar peli un nosaka raustītās līnijas galapunktu, atliekot līknes <4 cPoints> punktus.
FM03_pM	<code>void pbCanva_MouseUp(object sender, MouseEventArgs e)</code>	Izpildās, kad vadīklā pbCanva tiek atlaists datorpeles taustiņš. Pārtrauc līknes modificēšanu ar peli.
FM04_pP	<code>void pbCanva_Paint(object sender, PaintEventArgs e)</code>	Attēlo visus grafiskos objektus vadīklā pbCanva – Bezjē līknes, kontrolpunktus, mezglu punktus, kontrolpunktu veidotos nogriežņus un raustīto līniju atliekot līknes <4 cPoints> punktus.
FM05_fRS	<code>void FormMain_Resize(object sender, EventArgs e)</code>	Izpildās, kad tiek veikta galvenā loga izmēru maiņa. Nodrošina, ka minētais logs ir reaģējošs (uz izmēra izmaiņām).
FM06_bUB	<code>void btnUploadBackground_Click(object sender, EventArgs e)</code>	Augšupielādē un iestata lietotāja izvēlētu fona attēlu vadīklā pbCanva. Klūdas gadījumā izvadīta 1. paziņojumu.
FM07_cSB	<code>void cbShowBackground_CheckStateChanged(object sender, EventArgs e)</code>	Maina augšupielādētā fona attēla redzamību uz pretējo.
FM08_ZC	<code>private void ZoomInOut(bool zoom)</code>	Veic vadīklas pbCanva un visu tajā esošo grafisko objektu pietuvināšanu un attālināšanu.
FM09_bZO	<code>private void btnZoomOut_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla btnZoomOut. Pietuvina vadīklu pbCanva.

Nosaukums	Funkcija	Apraksts
FM10_bZI	private void btnZoomIn_Click (object sender, EventArgs e)	Izpildās, kad tiek nospiesta vadīkla btnZoomIn . Attālina vadīklu pbCanva .
FM11_bRA	void btnResetAll_Click (object sender, EventArgs e)	Inic平izē galveno logu no jauna. Pirms iniciālizācijas izvada 4. paziņojumu.

void pbCanva_Paint(object sender, PaintEventArgs e):

Funkcija attēlo grafisku objektus (skat. 3.11. att.):

- Visas Bezjē līknēs melnā krāsā ar līnijas biezumu 1px.
- Visus kontrolpunktus izņemot <4 pPoints> līkņu (III) galapunktus un <Composite> līkņu (V) kontrolpunktus, kas nav rokturi. Kontrolpunktai tiek attēloti kā sarkanas rīnķa līnijas ar centru kontrolpunktā, rādiusu 2px un līnijas biezumu 1px.
- Visus mezglu punktus kā melnus rīnķus ar centru kontrolpunktā un rādiusu 2px.
- Visus nogriežņus starp katriem diviem secīgiem kontrolpunktēm līknēs <4 cPoints> (I, II), <4 pPoints> (III) un <Least Squares> (IV) un starp katriem diviem secīgiem rokturiem <Composite> līknēs (V) gaiši pelēkā krāsā ar līnijas biezumu 1px.
- Nogriezni starp pēdējo kontrolpunktē nepabeigtā <4 cPoints> līknē (I) un datorpeles kursoora atrašanās vietē ar raustītu līniju biezumā 1px, gaiši pelēkā krāsā.



3.11. att. Grafisko objektu attēlošanas projektējums

3.3.2. Jaunu līkņu pievienošana

Jaunu līkņu pievienošana iekļauj visas tās funkcijas, kas saistītas ar Bezjē līkņu pievienošanu – līknēs kontrolpunktū vai mezglu pievienošanu un līknēs raksturojošo parametru

saglabāšanu, bet neiekļauj funkcijas, kas nepieciešamas konstruēšanas aprēķiniem. Visu funkciju uzskaitē un īsi apraksti atrodami 3.2. tabulā, sarežģītākajām funkcijām pieejami arī detalizētāki apraksti. Klūdu paziņojumu saraksts atrodams 3.2.3. nodaļā.

3.2. tabula – Jaunu līkņu pievienošanas funkcijas

Nosaukums	Funkcija	Apraksts
NC01_NL	<code>void NewCurve(BezierType curveType)</code>	Izveido jaunu līkni.
NC02_b4C	<code>void btnNew4cPoints_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnNew4cPoints</code> . Atļauj veikt līknes <4 cPoints> kontrolpunktu ievadi ar datorpeli, tastatūru vai no .txt faila.
NC03_b4P	<code>void btnNew4pPoints_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnNew4pPoints</code> . Atļauj veikt līknes <4 pPoints> mezglu punktu ievadi ar datorpeli, tastatūru vai no .txt faila.
NC04_bLS	<code>void btnNewLeastSquares_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnNewLeastSquares</code> . Atļauj veikt līknes <Least Squares> mezglu punktu ievadi ar datorpeli, tastatūru vai no .txt faila.
NC05_bC	<code>void btnNewComposite_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnNewComposite</code> . Atļauj veikt līknes <Composite> mezglu punktu ievadi ar datorpeli, tastatūru vai no .txt faila.
NC06_bDC	<code>void btnDoneComposite_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnDoneComposite</code> . Atzīmē, ka pēdējai <Composite> līknei nepieciešams pievienot pēdējos kontrolpunktus.
NC07_AC	<code>void AddcPoint(Point mouseLocation)</code>	Pievieno ar peli ievadīta kontrolpunkta koordinātas izvēlētai līknei.
NC08_AP	<code>void AddpPoint(Point mouseLocation)</code>	Pievieno ar peli ievadīta mezgla punkta koordinātas izvēlētai līknei.
NC09_PF	<code>List<Point> GetPointsFromFile()</code>	Ielasā .txt failu, pārvērš tā tekstu par punktu koordinātām un atgriež sarakstu ar šiem punktiem. Faila atvēršanas klūdas gadījumā izvada 1. paziņojumu, nepareiza faila noformēšanas gadījumā izvada 2. paziņojumu.

3.3.3. Līkņu konstruēšana

Līkņu konstruēšana iekļauj visas tās funkcijas, kas saistītas ar Bezjē līkņu konstruēšanu – līknes kontrolpunktu koordinātu aprēķināšanu, tai skaitā visas interpolācijai nepieciešamās funkcijas, parametrizācijas metodes, saliktu līkņu konstruēšanu. Visu funkciju uzskaitē un īsi apraksti atrodami 3.3. tabulā, sarežģītākajām funkcijām pieejami arī detalizētāki apraksti.

3.3. tabula – Līkņu konstruēšanas funkcijas

Nosaukums	Funkcija	Apraksts
CC01_CI	<code>void AddcPointsInterpolation(int i)</code>	Veic līkņu <4 pPoints> un <Least Squares> interpolāciju izmantojot izvēlētās līknes mezglu punktus. Aprēķina un saglabā interpolētās līknes kontrolpunktus.
CC02_VU	<code>List<double> GetsValuesUniform(List<Point> pList)</code>	Aprēķina un atgriež sarakstu ar s vērtībām mezglu punktiem izvēlētai līknei. Vērtības tiek iegūtas izmantojot vienmērīgā sadalījuma parametrizācijas metodi. Skatīt 3.1.7. nodaļu.
CC03_VCh	<code>List<double> GetsValuesChord(List<Point> pList)</code>	Aprēķina un atgriež sarakstu ar s vērtībām mezglu punktiem izvēlētai līknei. Vērtības tiek iegūtas izmantojot hordu garuma parametrizācijas metodi. Skatīt 3.1.7. nodaļu.
CC04_VCe	<code>List<double> GetsValuesCentripetal(List<Point> pList)</code>	Aprēķina un atgriež sarakstu ar s vērtībām mezglu punktiem izvēlētai līknei. Vērtības tiek iegūtas izmantojot centrīces spēka parametrizācijas metodi. Skatīt 3.1.7. nodaļu.
CC05_AS	<code>double[,] GetArrayS(List<double> sValues)</code>	Aprēķina un atgriež divdimensiju masīvu, kas reprezentē matricu \mathbf{S} . Masīvs tiek aizpildīts, izmantojot parametrizācijas rezultātā iegūtas s vērtības.
CC06_CC	<code>void AddcPointsComposite(int i)</code>	Aprēķina un saglabā kontrolpunktu koordinātas <Composite> līknēm ar vismaz trīs mezglu punktiem.
CC07_OC	<code>void AddOnlycPointsComposite(int i)</code>	Aprēķina un saglabā kontrolpunktu koordinātas <Composite> līknēm ar diviem mezglu punktiem.

Nosaukums	Funkcija	Apraksts
CC08_VF	Point GetVeryFirstHandle (Point firstpPoint, Point oppositeHandle, Point secondpPoint)	Aprēķina un atgriež paša pirmā roktura koordinātas mezgla punktam, kas ir sākumpunkts un pieder <Composite> līknei ar vismaz trīs mezglu punktiem.
CC09_FH	Point GetFirstHandle (Point prevpPoint, Point thispPoint, Point nextpPoint)	Aprēķina un atgriež pirmā roktura koordinātas mezgla punktam, kas nav galapunkts un pieder <Composite> līknei ar vismaz trīs mezglu punktiem.
CC10_LH	Point GetSecondHandle (Point prevpPoint, Point thispPoint, Point nextpPoint)	Aprēķina un atgriež otrā roktura koordinātas mezgla punktam, kas nav galapunkts un pieder <Composite> līknei ar vismaz trīs mezglu punktiem.
CC11_VL	Point GetVeryLastHandle (Point prevpPoint, Point prevHandle, Point lastpPoint)	Aprēķina un atgriež pirmā roktura koordinātas mezgla punktam, kas ir beigu punkts, un pieder <Composite> līknei ar vismaz trīs mezglu punktiem.
CC12_L	double GetLength (Point firstPoint, Point secondPoint)	Aprēķina un atgriež attālumu starp diviem punktiem.

void AddPointsComposite(int i):

Funkcija aprēķina rokturus un saglabā visu kontrolpunktu koordinātas <Composite> līknēm ar trīs vai vairāk mezglu punktiem (skat. 3.2. att.). Apzīmējam saliktās līknes mezglu punktu skaitu ar n . Varam ievērot, ka:

- katrs trešais kontrolpunkts $\mathbf{c}_{3i}, i \in \{0, 1, \dots, n\}$ sakrīt ar līknes i -to mezglu punktu;
- katrs kontrolpunkts formā $\mathbf{c}_{3i-1}, i \in \{1, \dots, n-1\}$ ir pirms rokturis, aprēķināms ar funkciju **GetFirstHandleComposite()**;
- katrs kontrolpunkts formā $\mathbf{c}_{3i+1}, i \in \{1, \dots, n-1\}$ ir otrs rokturis, aprēķināms ar funkciju **GetSecondHandleComposite()**;
- kontrolpunkts \mathbf{c}_1 ir saliktās līknes pats pirms rokturi, aprēķināms ar funkciju **GetVeryFirstHandleComposite()**;
- kontrolpunkts \mathbf{c}_{n-1} ir saliktās līknes pats pēdējais rokturis, aprēķināms ar funkciju **GetVeryLastHandleComposite()**.

void AddOnlycPointsComposite(int i):

Funkcija aprēķina rokturus un saglabā visu kontrolpunktu koordinātas <Composite> līknēm ar diviem mezglu punktiem (skat. 3.3. att.).

Līknes pirmais kontrolpunkts un pēdējais kontrolpunkts sakrīt ar attiecīgi līknes pirmo mezglu \mathbf{p}_0 un otro mezglu punktu \mathbf{p}_1 . Rokturu ($\mathbf{c}_1, \mathbf{c}_2$) veidotais leņķis ar taisni, ko veido mezglu punkti, ir 60° , savukārt attālumi no rokturiem līdz tiem tuvākajiem mezglu punktiem vienādi ar pusi no attāluma starp mezglu punktiem. Tātad rokturu koordinātas aprēķināmas ar formulām:

$$\begin{pmatrix} \mathbf{c}_{ix} \\ \mathbf{c}_{iy} \end{pmatrix} = \begin{pmatrix} \cos 60^\circ & -\sin 60^\circ \\ \sin 60^\circ & \cos 60^\circ \end{pmatrix} \begin{pmatrix} \mathbf{m}_x - \mathbf{p}_{(i-1)x} \\ \mathbf{m}_y - \mathbf{p}_{(i-1)y} \end{pmatrix} + \begin{pmatrix} \mathbf{p}_{(i-1)x} \\ \mathbf{p}_{(i-1)y} \end{pmatrix}$$

$$\mathbf{m} = 0.5 \cdot (\mathbf{p}_0 + \mathbf{p}_1).$$

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.6. nodaļā.

```
Point GetVeryFirstHandle(Point firstpPoint, Point oppositeHandle, Point secondpPoint):
```

Funkcija aprēķina un atgriež pirmā roktura \mathbf{r} koordinātas mezglu punktam, kas ir sākumpunkts un pieder salikta Bezjē līknei ar vismaz 3 mezgliem (skat. 3.2. att.).

Funkcijai tiek padoti divi mezglu punkti – sākumpunkta mezgls (`firstpPoint`), līknes nākamais mezgls (`secondpPoint`), kā arī segmenta otrs rokturis (`nextHandle`). Punkti tiek skaitīti no saliktās līknes sākumpunkta līdz galapunktam.

Aprēķinus var izteikt kā vektoru darbības. Ieviešam sekojošus apzīmējumus:

- \mathbf{v}_{sf} – vektors no `secondpPoint` uz `firstpPoint`;
- \mathbf{v}_{sh} – vektors no `secondpPoint` uz `nextHandle`;
- \mathbf{v}_{hr} – vektors no `nextHandle` uz rokturi \mathbf{r} .

Roktura \mathbf{r} koordinātas aprēķināmas kā $\mathbf{r} = \text{nextHandle} + \mathbf{v}_{hr}$. Vektora \mathbf{v}_{hr} virziens ir tāds pats kā vektoram \mathbf{v}_{sf} , tā garums izsakāms kā \mathbf{v}_{sf} garums mīnus divi garumi \mathbf{v}_{sh} projekcijai uz \mathbf{v}_{sf} .

$$\mathbf{v}_{hr} = \frac{\mathbf{v}_{sf}}{|\mathbf{v}_{sf}|} \cdot \left(|\mathbf{v}_{sf}| - 2 \cdot \left| \text{proj}_{\mathbf{v}_{sf}} \mathbf{v}_{sh} \right| \right)$$

$$\mathbf{v}_{hr} = \mathbf{v}_{sf} \cdot \left(\frac{|\mathbf{v}_{sf}|}{|\mathbf{v}_{sf}|} - \frac{2 \cdot \frac{\mathbf{v}_{sf} \cdot \mathbf{v}_{sh}}{|\mathbf{v}_{sf}|}}{|\mathbf{v}_{sf}|} \right)$$

$$\mathbf{v}_{hr} = \mathbf{v}_{sf} \cdot \left(1 - 2 \frac{\mathbf{v}_{sf} \cdot \mathbf{v}_{sh}}{|\mathbf{v}_{sf}|^2} \right)$$

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.6. nodaļā.

```
Point GetFirstHandle(Point prevpPoint, Point thispPoint, Point nextpPoint):
```

Funkcija aprēķina un atgriež pirmā roktura \mathbf{r} koordinātas mezglu punktam, kas nav galapunkts un pieder salikta Bezjē līknei (skat. 3.2. att.).

Funkcijai tiek padoti trīs mezglu punkti – tas, kuram tiek rēķināts rokturis (`thispPoint`) un tā blakus mezgli (`prevpPoint` un `nextpPoint`). Punkti tiek skaitīti no saliktās līknēs sākumpunkta līdz galapunktam.

Aprēķinus var izteikt kā vektoru darbības. Vektoram \mathbf{v}_1 no `thispPoint` uz pirmo rokturi būs tāds pats virziens, kā vektoram \mathbf{v}_2 no `nextpPoint` uz `prevpPoint`. Apzīmējam attālumu no `prevpPoint` līdz `thispPoint` ar l_1 un attālumu no `thispPoint` līdz `nextpPoint` ar l_2 . Tad vektoru \mathbf{v}_1 var aprēķināt kā pusi no vektora \mathbf{v}_2 reiz $\frac{l_1}{l_1 + l_2}$. Esam ieguvuši, ka pirmā roktura koordinātas izsakāmas ar formulām:

$$\mathbf{v}_1 = 0.5 \cdot \frac{l_1}{l_1 + l_2} \cdot \mathbf{v}_2$$

$$\mathbf{r} = \text{thispPoint} + \mathbf{v}_1.$$

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.6. nodaļā.

```
Point GetSecondHandle(Point prevpPoint, Point thispPoint, Point nextpPoint)
```

Aprēķini ir ļoti līdzīgi aprēķiniem funkcijā `GetFirstHandleComposite()`. Vektoram \mathbf{v}_1 no `thispPoint` uz otro rokturi \mathbf{r} būs tāds pats virziens, kā vektoram \mathbf{v}_2 no `prevpPoint` uz `nextpPoint`. Apzīmējam attālumu no `prevpPoint` līdz `thispPoint` ar l_1 un attālumu no `thispPoint` līdz `nextpPoint` ar l_2 . Tad vektoru \mathbf{v}_1 var aprēķināt kā pusi no vektora \mathbf{v}_2 reiz $\frac{l_2}{l_1 + l_2}$. Esam ieguvuši, ka pirmā roktura \mathbf{r} koordinātas izsakāmas ar formulām:

$$\mathbf{v}_1 = 0.5 \cdot \frac{l_2}{l_1 + l_2} \cdot \mathbf{v}_2$$

$$\mathbf{r} = \text{thispPoint} + \mathbf{v}_1.$$

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.6. nodaļā.

```
Point GetVeryLastHandle(Point prevpPoint, Point prevHandle, Point lastpPoint):
```

Aprēķini ir ļoti līdzīgi aprēķiniem funkcijā `GetVeryFirstHandleComposite()`. Ieviešam sekojošus apzīmējumus:

- \mathbf{v}_{pl} – vektors no `prevpPoint` uz `lastpPoint`;
- \mathbf{v}_{ph} – vektors no `prevpPoint` uz `prevHandle`;
- \mathbf{v}_{hr} – vektors no `prevHandle` uz pēdējo rokturi \mathbf{r} .

Roktura \mathbf{r} koordinātas aprēķināmas kā $\mathbf{r} = \text{prevHandle} + \mathbf{v}_{hr}$. Vektora \mathbf{v}_{hr} virziens ir tāds pats kā vektoram \mathbf{v}_{pl} , turklāt tā garums izsakāms kā \mathbf{v}_{pl} garums mīnus divi garumi \mathbf{v}_{ph} projekcijai uz \mathbf{v}_{pl} .

$$\mathbf{v}_{hr} = \frac{\mathbf{v}_{pl}}{|\mathbf{v}_{pl}|} \cdot \left(|\mathbf{v}_{pl}| - 2 \cdot |\text{proj}_{\mathbf{v}_{pl}} \mathbf{v}_{ph}| \right)$$

$$\mathbf{v}_{hr} = \mathbf{v}_{pl} \cdot \left(\frac{|\mathbf{v}_{pl}|}{|\mathbf{v}_{pl}|} - \frac{2 \cdot \frac{\mathbf{v}_{pl} \cdot \mathbf{v}_{ph}}{|\mathbf{v}_{pl}|}}{|\mathbf{v}_{pl}|} \right)$$

$$\mathbf{v}_{hr} = \mathbf{v}_{pl} \cdot \left(1 - 2 \frac{\mathbf{v}_{pl} \cdot \mathbf{v}_{ph}}{|\mathbf{v}_{pl}|^2} \right)$$

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.6. nodaļā.

3.3.4. Līkņu modificēšana

Līkņu modificēšana iekļauj visas tās funkcijas, kas saistītas ar jau konstruētu Bezjē līkņu modificēšanu, tai skaitā parametrizācijas metodes maiņu, kontrolpunktu un mezglu punktu atrašanās vietu maiņu. Visu funkciju uzskaitē un īsi apraksti atrodami 3.4. tabulā, sarežģītākajām funkcijām pieejami arī detalizētāki apraksti. Klūdu paziņojumu saraksts atrodams 3.2.3. nodaļā.

3.4. tabula – Līkņu modificēšanas funkcijas

Nosaukums	Funkcija	Apraksts
MC01_bCP	<code>void btnChangeParam_Click(object sender, EventArgs e)</code>	Atļauj izvēlēties līkni, kurai veikt līkņu parametrizācijas metodes maiņu. Izpildās, kad tiek nospiesta atbilstošā vadīkla.
MC02_CP	<code>void ChangeParametrization()</code>	Norāda izvēlētās līknes parametrizācijas metodi un atļauj veikt līkņu <code><4 pPoints></code> , <code><Least Squares></code> parametrizācijas metodes maiņu. Ja līknei netiek izmantota parametrizācijas metode, izvada 7. paziņojumu ar attiecīgo līknes veidu.
MC03_rUC	<code>void rbUniform_CheckedChanged(object sender, EventArgs e)</code>	Veic līkņu <code><4 pPoints></code> un <code><Least Squares></code> parametrizācijas metodes maiņu uz/no vienmērīga sadalījuma metodes. Izpildās, kad tiek mainīts atbilstošās radio pogas statuss.

Nosaukums	Funkcija	Apraksts
MC04_rCC	<code>void rbChord_CheckedChanged(object sender, EventArgs e)</code>	Veic līkņu <code><4 pPoints></code> un <code><Least Squares></code> parametrizācijas metodes maiņu uz/no hordu garuma metodes. Izpildās, kad tiek mainīts atbilstošās radio pogas statuss.
MC05_bMC	<code>void btnModifycPoints_Click(object sender, EventArgs e)</code>	Atļauj veikt kontrolpunktu koordinātu modifīcēšanu. Izpildās, kad tiek nospiesta atbilstošā vadīkla.
MC06_bMP	<code>void btnModifypPoints_Click(object sender, EventArgs e)</code>	Atļauj veikt mezglu punktu koordinātu modifīcēšanu. Izpildās, kad tiek nospiesta atbilstošā vadīkla.
MC07_MC	<code>void ModifycPoint(MouseEventArgs e)</code>	Pārbauda, vai izvēlētais kontrolpunkts var tikt modificēts, nosaka tā modifīcēšanas veidu. Ja kontrolpunkts nevar tikt modificēts, izvada 5. paziņojumu ar attiecīgo līknēs veidu <code><4 pPoints></code> un <code><Least Squares></code> līknēm vai 6. paziņojumu, veicot <code><Composite></code> līknēs kontrolpunktu modifīcēšanu ar tastatūru.
MC08_MP	<code>void ModifypPoint()</code>	Pārbauda, vai izvēlētais mezgla punkts var tikt modificēts, nosaka tā modifīcēšanas veidu.
MC09_MH	<code>void ModifyHandleComposite(Point modifyHandle, Point middlePoint, Point oppositeHandle, int opposite)</code>	Saliktais līknēs roktura patvalīgas modifīcēšanas gadījumā aprēķina un saglabā pretējā roktura koordinātas.
MC10_MHS	<code>void ModifyHandleCompositeStraight(Point modifyHandle, Point middlePoint, Point oppositeHandle)</code>	Saliktais līknēs roktura ierobežotas modifīcēšanas gadījumā aprēķina un saglabā roktura koordinātas.
MC11_MPC	<code>void ModifypPointComposite(Point mouseLocation)</code>	Saliktais līknēs mezgla punkta modifīcēšanas gadījumā aprēķina blakus rokturu koordinātas un saglabā visu modificēto punktu koordinātas.

```
void ModifyHandleComposite(Point modifyHandle, Point middlePoint, Point oppositeHandle,
int opposite):
```

Patvalīgi modifīcējot (izmantojot datorpeles kreiso taustiņu) saliktais līknēs roktura `modifyHandle` koordinātas, nepieciešams mainīt arī pretējā roktura `oppositeHandle` koordinātas, lai katrs mezgls un tā blakus rokturi būtu uz vienas taisnes un tādējādi tiktu nodrošināts, ka līknēi piemīt C^2 nepārtrauktība. Tās tiek aprēķinātas tā, lai starp rokturiem esošais mezgla punkts

`middlePoint` pieder taisnei, ko veido minētie rokturi, un attālums no `middlePoint` līdz `oppositeHandle` ir nemainīgs. Skatīt 3.6. att.

Aprēķinus var izteikt kā vektoru darbības. Ieviešam apzīmējumus:

- \mathbf{v}_{mp} – vektors no `modifyHandle` uz `middlePoint`;
- \mathbf{v}_{po} – vektors no `middlePoint` uz `oppositeHandle`;
- \mathbf{v}_{pr} – vektors no `middlePoint` uz jaunajām roktura koordinātām \mathbf{r} .

Vektoram \mathbf{v}_{pr} jābūt tādam pašam virzienam kā \mathbf{v}_{mp} un garumam $|\mathbf{v}_{pr}|$ jābūt vienādam ar $|\mathbf{v}_{po}|$. Varam izmantot formulu:

$$\mathbf{r} = \text{middlePoint} + \frac{\mathbf{v}_{mp}}{|\mathbf{v}_{mp}|} \cdot |\mathbf{v}_{po}|.$$

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.8. nodaļā.

```
void ModifyHandleCompositeStraight(Point modifyHandle, Point middlePoint, Point oppositeHandle):
```

Ierobežoti modificējot (izmantojot datorpeles labo taustiņu) saliktais līknes roktura `modifyHandle` koordinātas, minēto rokturi iespējams pārvietot tikai pa staru, kas pieder taisnei, ko veido pretējais rokturis `oppositeHandle` un mezgla punkts `middlePoint` starp minētajiem rokturiem. Stara sākumpunkts ir `middlePoint`. Skatīt 3.7. att.

Ērts veids, kā to implementēt, ir izmantojot vektoru darbības. Vektoram \mathbf{v}_{pm} no `middlePoint` uz `modifyHandle` jābūt tādam pašam virzienam, kā vektoram \mathbf{v}_{op} no `oppositeHandle` uz `middlePoint`, savukārt attālumam no `middlePoint` līdz jaunajām roktura koordinātām \mathbf{r} jāsakrīt ar $|\mathbf{v}_{pm}|$. Varam izmantot formulu:

$$\mathbf{r} = \text{middlePoint} + \frac{\mathbf{v}_{op}}{|\mathbf{v}_{op}|} \cdot |\mathbf{v}_{pm}|.$$

Implementācijā jāpievērš uzmanība maksimālajam attālumam starp datorpeles kursora atrašanās vietu un mezgla koordinātām.

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.8. nodaļā.

```
void ModifyPointComposite(Point mouseLocation):
```

Funkcija aprēķina un saglabā saliktais līknes mezgla punkta \mathbf{p}_1 un tā blakus rokturu $\mathbf{c}_0, \mathbf{c}_1$ jaunās koordinātas, kad mezgla punkts tiek modificēts. Lai līknei arī pēc modificēšanas piemistu C^2 nepārtrauktība un pārējie līknes segmenti modifikācijas rezultātā nemainītos, ērtākais veids ir nemainīt \mathbf{c}_0 un \mathbf{c}_1 relatīvo atrašanās vietu pret \mathbf{p}_1 . Skatīt 3.8. att.

Aprēķinus var interpretēt kā vektoru darbības. Jaunās rokturu koordinātas $\mathbf{c}'_0, \mathbf{c}'_1$ var aprēķināt izmantojot formulu:

$$\mathbf{c}'_i = \mathbf{c}_i + (\text{mouseLocation} - \mathbf{p}_1).$$

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.8. nodaļā.

3.3.5. Esošu līkņu funkcijas

Esošu līkņu funkcijas iekļauj visas tās funkcijas, kas saistītas ar jau konstruētām Bezjē līknēm – līkņu kontrolpunktu vai mezglu punktu izvadi un līkņu dzēšanu, bet neiekļauj funkcijas, kas saistītas ar līkņu modifīcēšanu. Visu funkciju uzskaitē un īsi apraksti atrodami 3.5. tabulā. Kļūdu paziņojumu saraksts atrodams 3.2.3. nodaļā.

3.5. tabula – Esošu līkņu funkcijas

Nosaukums	Funkcija	Apraksts
EC01_LP	void FindLocalPoint (List<List<Point>> PointsAll, Point MouseLocation)	Atrod, vai peles cursora apkārtnē ir kāds kontrolpunks vai mezgla punkts.
EC02_bOC	void btnOutputcPoints_Click (object sender, EventArgs e)	Atļauj izvēlēties līkni, kurai izvadīt līknes kontrolpunktu koordinātas.
EC03_bOP	void btnOutputpPoints_Click (object sender, EventArgs e)	Atļauj izvēlēties līkni, kurai izvadīt līknes mezglu punktu koordinātas.
EC04_OCF	void OutputcPointsToFile ()	Izvada izvēlētas līknes kontrolpunktu koordinātas .txt failā. Funkcijas kļūdas gadījumā izvada 3. paziņojumu.
EC05_OPF	void OutputpPointsToFile ()	Izvada izvēlētas līknes mezglu punktu koordinātas .txt failā. Funkcijas kļūdas gadījumā izvada 3. paziņojumu.
EC06_bDL	void btnDeleteCurve_Click (object sender, EventArgs e)	Izpildās, kad tiek nospiesta vadīkla btnDeleteCurve . Atļauj izvēlēties līkni, kuru izdzēst.
EC07_DL	void DeleteCurve (int i)	Izdzēš izvēlētu līkni.

3.3.6. Koordinātu logs

Koordinātu logs iekļauj visas tās funkcijas, kas izpildās un ir piesaistītas koordinātu logam – tā inicializācijas veidus, punktu koordinātu ievadi, izvadi un pievienošanu. Visu funkciju uzskaitē un īsi apraksti atrodami 3.5. tabulā. Kļūdu paziņojumu saraksts atrodams 3.2.3. nodaļā.

3.5. tabula – Koordinātu loga funkcijas

Nosaukums	Funkcija	Apraksts
FC01_IA	<code>void InitializeAdd()</code>	Inicializē formu, ja tā izsaukta jaunas līknes pievienošanai izmantojot datora tastatūru.
FC02_IM	<code>void InitializeModify()</code>	Inicializē formu, ja tā izsaukta esošas līknes modifīcēšanai izmantojot datora tastatūru.
FC03_IO	<code>void InitializeOutput()</code>	Inicializē formu, ja tā izsaukta esošas līknes punktu izvadīšanai uz ekrāna.
FC04_AR	<code>void AddRow()</code>	Pievieno jaunu rindu un aizpilda to ar nepieciešamajām vadīklām, lai ievadītu vai izvadītu punktu koordinātas.
FC05_bAR	<code>void btnAddRow_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnAddRow</code> . Pievieno jaunu rindu pēc lietotāja vaicājuma. Ja rindu skaits ir sasniedzis sistēmas uzstādīto ierobežojumu, rindu nepievieno un izvada 8. paziņojumu ar attiecīgo skaitu.
FC06_bDR	<code>void btnDeleteRow_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnDeleteRow</code> . Nodzēš rindu pēc lietotāja vaicājuma. Ja rindu skaits ir vienāds ar četri <Least Squares> līknēm un ar divi <Composite> līknēm, rindu nenodzēš un izvada 9. paziņojumu ar attiecīgo līknes veidu un punktu skaitu.
FC07_bSI	<code>void btnSubmitInput_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnSubmitInput</code> . Pārbauda, vai visas vadīklas koordinātu ievadei ir aizpildītas, pārvērš ievadīto tekstu punktos un tos saglabā. Ja visas vadīklas nav aizpildītas, izvada 10. paziņojumu.
FC08_bRI	<code>void btnResetInput_Click(object sender, EventArgs e)</code>	Izpildās, kad tiek nospiesta vadīkla <code>btnResetInput</code> . Nodzēš tekstu visās vadīklās, kas paredzētas punktu koordinātu ievadei.

3.4. Funkcionālo prasību un projektējuma funkciju atbilstība

Programmatūras funkcijas projektētas ar mērķi, lai tiktu izpildītas visas funkcionālās prasības. Tabulā 3.6. redzams, kuras funkcijas realizē kuras prasības.

3.6. tabula – Funkcionālo prasību un projektējuma funkciju atbilstība

Funkcionālā prasība	Atbilstošās funkcijas
Līkņu interpolācija caur patvalīgi uzdotiem punktiem. Punkti var tikt ievadīti manuāli ar tastatūru, atzīmēti ar datorpeli vai ielasīti no faila.	NC07_AC, NC08_AP, NC09_PF, FC01_IA, kā arī visas funkcijas 3.3.3. nodalā
Konstruētajām līknēm jābūt gludām (tām jāpiemīt C^2 nepārtrauktībai).	CC08_VF, CC09_FH, kā arī vispārīga Bezjē līkņu izmantošana
Konstruētās līknes jāspēj interaktīvi modifīcēt izmantojot datorpeli vai ievadot interpolējamo punktu izmaiņas ar tastatūru.	MC02_CP, MC07_MC, MC08_MP, MC09_MH, MC10_MHS, MC11_MPC, FC07_IM
Jāspēj izvadīt konstruēto līkņu parametrus uz ekrāna un failā, lai nepieciešamības gadījumā varētu rekonstruēt izveidoto līkni.	FC03_IO, EC04_OCF, EC05_OPF
Jānodrošina iespēja vizuāli salīdzināt rīkā konstruētās līknes ar līknēm esošā piegrieztnē.	FM06_bUB FM08_ZC
Dažādu nozīmju grafiskajiem objektiem (piemēram, interpolācijas mezglu punktiem un palīgpunktiem) jābūt atšķiramiem ar vizuālo izskatu, piemēram, krāsojumu.	FM04_pP
Jānodrošina iespēja konstruētās līknes dzēst.	EC07_DL

4. TESTĒŠANAS DOKUMENTĀCIJA

4.1. Testpiemēru projektējums

Testpiemēri (skat. 4.2. tabulu) veidoti apvienojot vairākas programmatūras veicamās darbības (skat. 4.1. tabulu). Testpiemēri projektēti tā, lai pārbaudītu visas testējamās darbības, līdz ar to arī visas programmatūras funkcijas un to mijiedarbību, kā arī kopējo sistēmas darbību.

4.1. tabula – Testējamās darbības

Darbības ID	Iesaistītās funkcijas	Darbība
1	NC01_NL NC02_b4C	Līknes <4 cPoints> konstruēšana.
2	NC01_NL NC03_b4P CC01_CI CC05_AS	Līknes <4 pPoints> konstruēšana.
3	NC01_NL NC04_bLS CC01_CI CC05_AS	Līknes <Least Squares> konstruēšana.
4	NC01_NL NC05_bC CC06_CC CC07_OC CC08_VF CC09_FH CC10_LH CC11_VL CC12_L	Līknes <Composite> konstruēšana.
5	CC02_VU	Vienmērīgā sadalījuma parametrizācijas metodes izvēle.
6	CC03_VCh	Hordu garuma parametrizācijas metodes izvēle.
7	CC04_VCe	Centrīieces spēka parametrizācijas metodes izvēle.
8	MC01_bCP MC02_Cp MC03_rUC MC04_rCC EC01_LP	Parametrizācijas metodes maiņa.
9	FM01_pMD FM02_pMM NC07_AC NC08_AP	Punktu ievade ar datorpeli.
10	FC01_IA FC07_bSI	Punktu ievade ar datora tastatūru.
11	NC09_PF	Punktu ievade no .txt faila.

Darbības ID	Iesaistītās funkcijas	Darbība
12	NC06_bDC	Pogas, kas atzīmē <Composite> līknī kā pabeigtu, nospiešana.
13	MC05_bMC MC07_MC	Kontrolpunktu modificēšana.
14	MC09_MH	Kontrolpunktu modificēšana līknēm <Composite>, izmantojot peles kreiso taustiņu.
15	MC10_MHS	Kontrolpunktu modificēšana līknēm <Composite>, izmantojot peles labo taustiņu.
16	MC06_bMP MC08_MP	Mezglu punktu modificēšana.
17	MC11_MPC	Mezglu punktu modificēšana līknēm <Composite>.
18	FM01_pMD FM02_pMM FM03_pM EC01_LP	Punktu modificēšana ar datorpeli.
19	FC02_IM FC07_bSI	Punktu modificēšana ar datora tastatūru.
20	FM01_pMD EC01_LP EC06_bDL EC07_DL	Līknes dzēšana.
21	FM01_pMD EC01_LP EC02_bOC	Kontrolpunktu izvade.
22	FM01_pMD EC01_LP EC03_bOP	Mezglu punktu izvade.
23	FC03_IO	Punktu izvade uz ekrāna.
24	EC04_OCF EC05_OPF	Punktu izvade .txt failā.
25	FM06_bUB	Fona attēla augšupielāde.
26	FM07_cSB	Fona attēla redzamības maiņa.
27	FM04_pP	Galvenā loga attēlošana.
28	FM05_fRS	Galvenā loga izmēra maiņa.
29	FM08_ZC FM09_bZO FM10_bZI	Grafisko objektu attēlošanas vadīklas izmēra maiņa.
30	FC01_IA FC02_IM FC03_IO FC04_AR FC05_bAR FC06_bDR FC08_bRI	Koordinātu loga attēlošana.
31	FM08_bRA	Logu inicializācija no jauna.

4.2. tabula – Testpiemēri

Test-piemēra ID	Sākuma stāvoklis	Testējamo darbību ID	Testa soli	Sagaidāmais rezultāts
T01		27	1. Palaiž programmu.	Tiek attēlots galvenais logs.
T02	T01	1, 9	1. Aktivizē radio pogu “w/ Mouse” no grupas “Choose points”. 2. Nospiež pogu “4 cPoints”. 3. Ar datorpeli izvēlas četrus patvaļīgus punktus atbilstošajā vadīklā.	Iespējams atlikt tieši četrus punktus. Atliekot punktus, tiek attēlota raustīta līnija starp peles kurSORU un iepriekšējo punktu. TieK attēlota Bezjē līkne, kuras kontrolpunkti ir ievadītie punkti.
T03	T02	13, 18	1. Aktivizē radio pogu “w/ Mouse” no grupas “Modify points”. 2. Nospiež pogu “Modify cPoints”. 3. Peles kurSORU novietu virs kāda kontrolpunkta. 4. Nospiež peles taustiņu. 5. Turot nospiestu taustiņu, pārvieto peles kurSORU.	Pārvietojot kontrolpunktU, Bezjē līkne tiek rekonstruēta, nav pamānāmu aizķeršanos.
T04	T02	21, 23, 30	1. Aktivizē radio pogu “Screen”. 2. Nospiež pogu “cPoints”. 3. Novieto peles kurSORU virs kāda kontrolpunkta. 4. Nospiež peles taustiņu.	Atveras koordinātu logs ar tieši četrām teksta lauku rindāM, kas aizpildītas ar līknes kontrolpunktU koordinātāM.
T05	T02	8	1. Nospiež pogu “Choose curve”. 2. Novieto peles kurSORU virs kāda kontrolpunkta. 3. Nospiež peles taustiņu.	Tiek izvadīts 7. klūdas paziņojums.
T06	T01	2, 5, 10, 30	1. Aktivizē radio pogu “w/ Keyboard” grupā “Choose points”. 2. Aktivizē radio pogu “Uniform”. 3. Nospiež pogu “4 pPoints”. 4. Aizpilda teksta rindas ar patvaļīgiem skaitliem. 5. Nospiež pogu “OK”.	Atveras koordinātu logs ar tieši četrām teksta lauku rindāM. Pēc punktu ievades, tiek attēlota Bezjē līkne, kas iet precīzi caur ievadītajiem punktiem un izmanto vienmērīga sadalījuma parametrizācijas metodi.
T07	T06	13	1. Aktivizē radio pogu “w/ Mouse” no grupas “Modify points”. 2. Nospiež pogu “Modify cPoints”.	Tiek izvadīts 5. klūdas paziņojums.

Test-piemēra ID	Sākuma stāvoklis	Testējamo darbību ID	Testa soli	Sagaidāmais rezultāts
			3. Novieto peles kurSORU virs kāda kontrolpunkta. 4. Nospiež peles taustiņu.	
T08	T06	16, 19, 30	1. Aktivizē radio pogu “w/ Keyboard” no grupas “Modify points”. 2. Nospiež pogu “Modify pPoints”. 3. Novieto peles kurSORU virs kāda mezglu punkta. 4. Nospiež peles taustiņu. 5. Aizpilda teksta rindas ar patvaļīgiem skaitļiem. 6. Nospiež pogu “OK”.	Atveras koordinātu logs ar tieši četrām teksta lauku rindām, kas aizpildītas ar līknes mezglu punktu koordinātām. Pēc punktu modifīcēšanas, Bezjē līkne tiek rekonstruēta.
T09	T06	22, 24	1. Aktivizē radio pogu “.txt file” no grupas “Modify points”. 2. Nospiež pogu “pPoints”. 3. Novieto peles kurSORU virs kāda mezglu punkta. 4. Nospiež peles taustiņu. 5. Nospiež pogu “OK”.	Atveras logs faila saglabāšanas vietas izvēlei datorā. Ja fails ar nosaukumu “points.txt” neeksistē, tāds tiek izveidots. Faila “points.txt” beigās tiek pievienotas rindiņas ar līknes konstruēšanas veidu, parametrizācijas veidu, mezglu punktu koordinātām.
T10	T01	3, 6, 10, 30	1. Aktivizē radio pogu “w/ Keyboard” no grupas “Choose points”. 2. Aktivizē radio pogu “Chord length”. 3. Nospiež pogu “Least Squares”. 4. Patvaļigu skaitu reižu spiež uz pogām “Add Row” un “Delete Row”. 5. Aizpilda teksta rindas ar patvaļīgiem skaitļiem. 6. Nospiež pogu “OK”.	Atveras koordinātu logs ar četrām teksta lauku rindām. Rindu skaitu iespējams samazināt un palielināt noteiktajās robežās. Pēc punktu ievades, tiek attēlota Bezjē līkne, kas iet iespējami tuvu ievadītajiem punktiem un izmanto hordu garuma parametrizācijas metodi.
T11	T01	3, 7, 11	1. Aktivizē radio pogu “From .txt file”. 2. Aktivizē radio pogu “Centripetal”. 3. Nospiež pogu “Least Squares”. 4. Izvēlas .txt failu. 5. Nospiež pogu “Open”.	Atveras logs faila izvēlei no datora. Pēc pareiza faila izvēles, tiek attēlota Bezjē līkne, kas iet iespējami tuvu ievadītajiem punktiem un izmanto centripetalas spēka parametrizācijas metodi.

Test-piemēra ID	Sākuma stāvoklis	Testējamo darbību ID	Testa soli	Sagaidāmais rezultāts
T12	T10	13	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Mouse” no grupas “Modify points”. Nospiež pogu “Modify cPoints”. Novieto peles kurSORU virs kāda kontrolpunkta. Nospiež peles taustiņu. 	Tiek izvadīts 5. klūdas paziņojums.
T13	T10	16, 18	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Mouse” no grupas “Modify points”. Nospiež pogu “Modify pPoints”. Novieto peles kurSORU virs kāda mezglu punkta. Nospiež peles taustiņu. Turot nospiestu taustiņu, pārvieto peles kurSORU. 	Pārvietojot mezglu punktu, Bezjē līkne tiek rekonstruēta, nav pamānāmu aizķeršanos.
T14	T10	8, 5, 6, 7	<ol style="list-style-type: none"> Nospiež pogu “Choose curve”. Novieto peles kurSORU virs kāda kontrolpunkta. Nospiež peles taustiņu. Aktivizē radio pogas “Uniform”, “Chord length”, “Centripetal”. 	Pēc katras parametrizācijas metodes maiņas, Bezjē līkne tiek rekonstruēta izmantojot izvēlēto metodi.
T15	T01	4, 9	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Mouse” no grupas “Choose points”. Nospiež pogu “Composite”. Ar datorpeli izvēlas patvalīgu skaitu punktu atbilstošajā vadīklā. 	Sākot no trešā atliktā punkta, tiek konstruēti un attēloti saliktas Bezjē līknēs segmenti, kuru galapunkti ir ievadītie punkti.
T16	T15	12	<ol style="list-style-type: none"> Nospiež pogu “Done”. 	Saliktajai Bezjē līknei tiek pievienots pēdējais segments.
T17	T01	4, 10, 30	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Keyboard” no grupas “Choose points”. Nospiež pogu “Composite”. Patvalīgu skaitu reižu spiež uz pogām “Add Row” un “Delete Row”. Aizpilda teksta rindas ar patvalīgiem skaitļiem. Nospiež pogu “OK”. 	Atveras koordinātu logs ar četrām teksta lauku rindām. Rindu skaitu iespējams samazināt un palielināt noteiktajās robežās, tiek izvadīti 8. un 10. paziņojumi. Pēc punktu ievades, tiek attēlota pabeigta salikta Bezjē līkne, kas iet precīzi caur ievadītajiem punktiem.

Test-piemēra ID	Sākuma stāvoklis	Testējamo darbību ID	Testa soli	Sagaidāmais rezultāts
T18	T15	13, 19, 30	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Keyboard” no grupas “Modify points”. Nospiež pogu “Modify cPoints”. Novieto peles kurSORU virs kāda kontrolpunkta. Nospiež peles taustiņu. 	Tiek izvadīts 6. klūdas paziņojums.
T19	T15	13, 14, 18	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Mouse” no grupas “Modify points”. Nospiež pogu “Modify cPoints”. Novieto peles kurSORU virs kāda kontrolpunkta. Nospiež peles kreiso taustiņu. Turot nospiestu taustiņu, pārvieto peles kurSORU. 	Pārvietojot kontrolpunktu, pretējais kontrolpunks pārvietojas, nodrošinot kolinearitāti un nemainot attālumu. Saliktās Bezjē līknes divi segmenti tiek rekonstruēti, nav pamanāmu aizķeršanos.
T20	T15	13, 15, 18	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Mouse” no grupas “Modify points”. Nospiež pogu “Modify cPoints”. Novieto peles kurSORU virs kāda kontrolpunkta. Nospiež peles labo taustiņu. Turot nospiestu taustiņu, pārvieto peles kurSORU. 	Kontrolpunktu iespējams pārvietot tikai pa staru, kolineāru pretējam kontrolpunktam. Pretējais kontrolpunks nepārvietojas. Saliktās Bezjē līknes segments tiek rekonstruēts, nav pamanāmu aizķeršanos.
T21	T15	16, 17, 18	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Mouse” no grupas “Modify points”. Nospiež pogu “Modify pPoints”. Novieto peles kurSORU virs kāda mezglu punkta. Nospiež peles labo taustiņu. Turot nospiestu taustiņu, pārvieto peles kurSORU. 	Pārvietojot mezglu punktu, blakus kontrolpunktī pārvietojas nemainot relatīvo atrašanās vietu pret mezglu. Saliktās Bezjē līknes divi segmenti tiek rekonstruēti, nav pamanāmu aizķeršanos.
T22	T15	16, 17, 19, 30	<ol style="list-style-type: none"> Aktivizē radio pogu “w/ Keyboard” no grupas “Choose points”. Nospiež pogu “Modify pPoints”. Novieto peles kurSORU virs kāda mezglu punkta. Nospiež peles taustiņu. Aizpilda teksta rindu ar patvalīgiem skaitļiem. Nospiež pogu “OK”. 	Atveras koordinātu logs ar vienu teksta lauku rindu. Pēc punktu modificēšanas, saliktās Bezjē līknes divi segmenti tiek rekonstruēti.

Test-piemēra ID	Sākuma stāvoklis	Testējamo darbību ID	Testa soli	Sagaidāmais rezultāts
T23	T02, T07, T10, T15	20	1. Nospiež pogu “Choose Curve to Delete”. 2. Novieto peles kurSORU virs kāda kontrolpunkta. 3. Nospiež peles taustiņu.	Uzspiežot uz līknes kontrolpunkta, līkne tiek izdzēsta.
T24	T02	25	1. Nospiež pogu “Upload Background Image”. 2. Izvēlas attēla failu. 3. Nospiež pogu “Open”.	Atveras logs faila izvēlei no datora. Pēc faila izvēles, tiek attēlots fona attēls.
T25	T24	26, 26	1. Divreiz nospiež izvēles rūtiņu “Show Background”.	Fona attēls pazūd un atkal parādās.
T26	T01	28	1. Izmantojot peli, maina galvenā loga izmēru.	Logs ir reaģējošs uz izmēra maiņu.
T27	T23	25, 29	1. Nospiež pogu “Upload Background Image”. 2. Izvēlas attēla failu. 3. Nospiež pogu “Open”. 4. Nospiež pogu “+”. 5. Nospiež pogu “-”.	Atveras logs faila izvēlei no datora. Pēc faila izvēles, tiek attēlots fona attēls. Proporcionāli mainās grafisko objektu vadīklas un tajā esošo objektu izmērs.
T28	T24	31	1. Nospiež pogu “Reset All”.	Tiek izvadīts 4. paziņojums. Logs tiek inicializēts no jauna.

4.2. Testēšanas žurnāls

Programmatūras izstrādes laikā un pēc tās pabeigšanas tika veikta 4.2. tabulā uzskaitīto testpiemēru testēšana. Testēšanas rezultāti pierakstīti 4.3. tabulā. Bezjē līkņu konstruēšanas pareizība tika pārbaudīta, veicot manuālos aprēķinus, no veiktajiem aprēķiniem konstruējot līknes tiešsaistes grafiskajā kalkulatorā *Desmos* [16], un vizuāli salīdzinot rezultātus.

4.3. *tabula – Testēšanas žurnāls*

Testēšanas ID	Datums	Rezultāts
T01	10.04.2019	Veiksmīgs.
T02	10.04.2019	Veiksmīgs.
T03	11.04.2019	Veiksmīgs.
T27	12.04.2019	Veiksmīgs.
T04	12.04.2019	Veiksmīgs.
T06	13.04.2019.	Veiksmīgs.
T09	17.04.2019.	Neveiksmīgs, netiek pievienotas jaunas rindiņas, viss fails tiek pārrakstīts no jauna.
T10	20.04.2019	Neveiksmīgs, rindu skaitu iespējams samazināt ārpus nepieciešamajām robežām.
T11	23.04.2019	Veiksmīgs.
T13	25.04.2019	Veiksmīgs.
T08	25.04.2019	Veiksmīgs.
T05	25.04.2019	Veiksmīgs.
T14	25.04.2019	Neveiksmīgs, līkne netiek rekonstruēta.
T14	25.04.2019	Veiksmīgs.
T24	25.04.2019	Veiksmīgs.
T25	25.04.2019	Veiksmīgs.
T27	25.04.2019	Neveiksmīgs, parametrizācijas metožu radiopogas neatgriežas noklusējuma stāvoklī.
T28	25.04.2019	Veiksmīgs.
T07	26.04.2019	Veiksmīgs.
T12	26.04.2019	Veiksmīgs.
T15	29.04.2019	Neveiksmīgs, rokturi netiek konstruēti atbilstoši projektējumam.
T16	30.04.2019	Neveiksmīgs, rokturi netiek konstruēti atbilstoši projektējumam.
T16	30.04.2019	Veiksmīgs.
T15	02.05.2019	Veiksmīgs.
T16	03.05.2019	Veiksmīgs.
T19	03.05.2019	Veiksmīgs.
T20	03.05.2019	Neveiksmīgs, ja roktura koordinātas sakrīt ar blakus mezgla koordinātam, rīks pārtrauc darbību.
T20	03.05.2019	Veiksmīgs.
T22	07.05.2019	Veiksmīgs.
T21	07.05.2019	Veiksmīgs.

Testēšanas ID	Datums	Rezultāts
T18	07.05.2019	Veiksmīgs.
T23	08.05.2019	Neveiksmīgs, pēc tam pievienojot jaunu līkni, rīks pārtrauc darbību.
T23	12.05.2019	Veiksmīgs.
T27	12.05.2019	Neveiksmīgs, grafisko objektu izmēri maina proporcijas.
T27	12.05.2019	Veiksmīgs.
T26	12.05.2019	Veiksmīgs.
T01	24.05.2019	Veiksmīgs.
T02	24.05.2019	Veiksmīgs.
T03	24.05.2019	Veiksmīgs.
T04	24.05.2019	Veiksmīgs.
T05	24.05.2019	Veiksmīgs.
T06	24.05.2019	Veiksmīgs.
T07	24.05.2019	Veiksmīgs.
T08	24.05.2019	Veiksmīgs.
T09	24.05.2019	Veiksmīgs.
T10	24.05.2019	Veiksmīgs.
T11	24.05.2019	Veiksmīgs.
T12	24.05.2019	Veiksmīgs.
T13	24.05.2019	Veiksmīgs.
T14	24.05.2019	Veiksmīgs.
T15	24.05.2019	Veiksmīgs.
T16	24.05.2019	Veiksmīgs.
T17	24.05.2019	Veiksmīgs.
T18	24.05.2019	Veiksmīgs.
T19	24.05.2019	Veiksmīgs.
T20	24.05.2019	Veiksmīgs.
T21	24.05.2019	Veiksmīgs.
T22	24.05.2019	Veiksmīgs.
T23	24.05.2019	Veiksmīgs.
T24	24.05.2019	Veiksmīgs.
T25	24.05.2019	Veiksmīgs.
T26	24.05.2019	Veiksmīgs.
T27	24.05.2019	Veiksmīgs.
T28	24.05.2019	Veiksmīgs.

5. PROJEKTA ORGANIZĀCIJA

Projekta izstrādi veica viens cilvēks un tika pielietots pakāpeniskā dzīves cikla modeļa paveids. Projekta izstrādi var iedalīt trīs daļās:

1. prasību specifikācija un matemātiskais projektējums,
2. atsevišķu programmatūru izstrāde dažādām funkcionalitātēm,
3. gala produkta izstrāde, testēšana un dokumentēšana.

Pirmajā izstrādes daļā tika veikta visu prasību specifikācija un lielākā daļa programmatūras matemātiskā projektējuma (3.1. nodaļa). Otrajai izstrādes daļai bija seši soli, kuros tika izstrādātas un daļēji testētas sešas atsevišķas, vienkāršas programmatūras, lai pārbaudītu, vai iecerētās metodes var apmierināt funkcionālās prasības. Atšķirībā no klasiskā pakāpeniskā dzīves cikla modeļa, katra soļa izstrādātie modeli uzreiz netika pievienoti kopējai sistēmai. Izstrādes trešajā daļā tika izveidots vispārējs arhitektūras projektējums, kopējā sistēma un tai pievienoti iepriekš izstrādātie modeli, kā arī pievienotas jaunas, iepriekš neizstrādātas, funkcionalitātes. Bez tā, trešajā izstrādes daļā notika gan vienībtestēšana, gan kopēja sistēmas testēšana, kā arī dokumentācijas izveide. Programmatūras projektēšana mazākos apmēros notika arī otrajā un trešajā izstrādes daļā.

5.1. Darbietilpības novērtējums

5.1.1. Prognozētā darbietilpība

Lai prognozētu projekta darbietilpību, ērti sadalīt visus izstrādes procesus vairākās daļās un katras daļas iecerēto ilgumu dienās novērtēt ar trīs punktu metodi (pesimistisko, reālistisko un optimistisko), skatīt 5.1. tabulu. Prognozētais ilgums tiek aprēķināts ar formula $\frac{\text{pes.} + 4 \cdot \text{reāl.} + \text{opt.}}{6}$.

5.1. tabula – Prognozētā darbietilpība

Izstrādes process	Pesimistiskais novērtējums (dienās)	Reālistiskais novērtējums (dienās)	Optimistiskais novērtējums (dienās)	Prognozētais ilgums (dienās)
Prasību specificēšana	4	3	2	3
Matemātiskais projektējums	60	45	40	46.7
Funkcionālo prasību modeļu izstrāde	18	11	7	11.5
Sistēmas arhitektūras un	6	4	2	4

Izstrādes process	Pesimistiskais novērtējums (dienās)	Reālistiskais novērtējums (dienās)	Optimistiskais novērtējums (dienās)	Prognozētais ilgums (dienās)
funkciju projektējums				
Izveidoto modeļu integrācija	7	4	2	4.2
Sistēmas izstrāde	20	17	12	16.7
Sistēmas testēšana	5	3	2	3.2
Dokumentācijas izstrāde	10	7	5	7.2
Kopā	130	94	72	96.3

Kā redzams 5.1. tabulā, sagaidāmais izstrādes ilgums ir 96.3 dienas jeb aptuveni 4.4 personmēneši, pieņemot, ka vienā mēnesī vidēji ir 21.74 darba dienas.

5.1.2. Reālā darbietilpība

Pēc programmatūras izstrādes bija iespējams novērtēt reālo projekta darbietilpību. Izveidotais rīks sastāv no divām saskarnēm un ļauj konstruēt līknes četros veidos, papildus izvēloties no trīs parametrizācijas metodēm un trīs punktu ievades veidiem. Jau konstruētas līknes iespējams modifīcēt, mainot to parametrizācijas metodi, kā arī mainot to kontrolpunktu un/vai mezglu punktu koordinātas ar datorpeli vai ievadot ar datora tastatūru. Konstruētās līknes iespējams arī dzēst, izvadīt to kontrolpunktu un/vai mezglu punktu koordinātas – uz ekrāna vai .txt failā. Tieki piedāvāta iespēja arī pievienot fona attēlu un mainīt tā redzamību. Rīkam kopumā saskaitāmi 31 funkcijpunkt (sakrīt ar darbībām 4.1. tabulā) jeb 4.4 personmēneši [17].

Pirms rīka programmēšanas, bija nepieciešams veikt padziļinātu izpēti par parametriskām, tai skaitā Bezjē, līknēm, kas prasīja vairāk nekā divus personmēnešus. Pēc rīka izstrādes bija nepieciešams izveidot programmatūras dokumentāciju.

Programmatūras funkcionālo rindiņu skaits ir 1488, un, ņemot vērā, ka izstrādātais rīks ir biznesa sistēma ar zinātnisku ievirzi, tā darbietilpība ir 3.5-7.7 personmēneši [17].

Kopumā projekta izstrāde autorei prasīja 5.3 personmēnešus.

5.2. Kvalitātes nodrošināšana

Lai nodrošinātu kvalifikācijas darba kvalitāti, tā izstrādē tika ievērotas vairākas darbības.

Darba dokumentācija tika izstrādāta, balstoties uz standarti LVS 68:1996 “Programmatūras prasību specifikācijas ceļvedis” [1], LVS 72:1996 “Ieteicamā prakse

programmatūras projektējuma aprakstīšanai” [2], un LVS 70:1996 “Programmatūras testēšanas dokumentācija” [3].

Sistēmas pirmkoda izstrādē ir ievērots Microsoft .NET programmēšanas stila standarts [18], uzturot atbilstošu mainīgo un funkciju nosaukumu veidošanas praksi un komentēšanas stilu. Pirmkods rakstīts veidā, kas iespējami palielina tā lasāmību, pirmkodā pēc vajadzības un uzņēmuma politikas ievietoti komentāri, kas apraksta tā darbību. Identifikatori veidoti saturīgi, funkcijas sakārtotas secībā, kas iespējami sakrīt ar to izmantošanas secību.

Visām kvalifikācijas darba funkcijām veikta vienībtestēšana, kā arī veikta sistēmas kopējās darbības testēšana un atķļudošana.

5.3. Konfigurāciju pārvaldība

Kvalifikācijas darba izstrādei tika izmantotas versiju kontroles sistēmas. Pie lielākām izmaiņām un darba dienas beigās tika veikta pirmkoda augšupielāde Git sistēmā. Sistēmas dokumentācijas versiju kontrolei tika izmantots Microsoft Office 365 tiešsaistes dokumentu veidošanas rīks [19], pie apjomīgākām izmaiņām dokumentācijas versiju kontrole (vismaz reizi trīs darba dienās) tika uzturēta arī Git sistēmā. Izstrādes versiju kontroles sistēmas tika izmantotas, lai glabātu un nepieciešamības gadījumā spētu atgriezt veiktās izmaiņas, piemēram, ja pašreizējā sistēmas versijā radusies klūda.

6. REZULTĀTI UN SECINĀJUMI

Kvalifikācijas darba izstrādes rezultātā tika iegūts funkcionējošs rīks, kas ļauj attēlot četros dažādos veidos konstruētas Bezjē līknes, papildus izmantojot trīs dažādas parametrizācijas metodes. Konstruētās līknes iespējams modifcēt, dzēst un izvadīt to kontrolpunktus un/vai mezglu punktus. Līkņu punktu ievade, izvade un modifcēšana veicama izmantojot datorpeli, datora tastatūru un teksta failus. Tika veikta arī rīka testēšana un izstrādāta tā dokumentācija.

Izstrādes laikā autore ieguva vērtīgu pieredzi visos programmatūras izstrādes procesos un veiksmīgi apguva jaunus programmēšanas rīkus – Microsoft Visual Studio vidi, C# valodu, Microsoft .NET satvaru, bibliotēku Windows Forms un Git versiju kontroles sistēmu. Bez tā autore apguva parametrisku līkņu teoriju, pastiprināti Bezjē līknes, to interpolāciju un parametrizāciju.

Pašlaik pasūtītājs sistēmu lieto ražošanā, turklāt ir plānots turpināt tās izstrādi, pievienojot tai papildu funkcionalitātes, piemēram, līknes garuma noteikšanu, grafisko objektu pietuvināšanu un mēroga iestatīšanu u. c.

Izvērtējot padarīto, autore secina, ka rīka implementācijā būtu ērtāk izmantot objektorientētās programmēšanas metodes, it īpaši Bezjē līkņu un citu grafisko objektu implementāciju kā klases.

Pabeidzot kvalifikācijas darbu, autore un pasūtītājs atkārtoti pārliecinājās, ka Bezjē līknes ir piemērotas izvēlētajai sistēmai. Veicot matemātisko projektējumu, tika izvēlēti četri Bezjē līkņu konstruēšanas veidi, trīs parametrizācijas metodes un dažādi līkņu modifcēšanas veidi. Autore secina, ka apvienojot šīs metodes, tiek nosegtas visas līkņu konstruēšanas prasības, turklāt tas tiek izdarīts intuitīvi un patērējot salīdzinoši maz skaitļošanas un atmiņas resursu.

PATEICĪBAS

Autore izsaka pateicību profesorei Dr. mat. Inesei Bulai par spriedumu un aprēķinu pārskatīšanu mazāko kvadrātu metodes izvedumā Bezjē līkņu interpolācijai, kā arī vispārīgiem ieteikumiem matemātiskās daļu noformēšanā.

IZMANTOTĀ LITERATŪRA UN AVOTI

- [1] LVS 68:1996 “Programmatūras prasību specifikācijas ceļvedis”.
- [2] LVS 72:1996 “Ieteicamā prakse programmatūras projektējuma aprakstīšanai”.
- [3] LVS 70:1996 “Programmatūras testēšanas dokumentācija”.
- [4] P. J. Barry, R. N. Goldman, “Three examples of Dual Properties of Bézier Curves”, *Mathematical Methods in Computer Aided Geometric Design*.
- [5] Bezjē līknes *Adobe Illustrator* programmatūrā,
<https://helpx.adobe.com/illustrator/using/drawing-basics.html>, pārbaudīts 25.05.2019.
- [6] Bezjē līknes *GIMP* programmatūrā, https://www.gimp.org/tutorials/Bezier_Selections, pārbaudīts 25.05.2019.
- [7] Bezjē līknes *Inkscape* programmatūrā,
<https://inkscape.org/en/doc/tutorials/advanced/tutorial-advanced.html>, pārbaudīts 25.05.2019.
- [8] C. Kohrs, N. Angenstein, A. Brechmann “Delays in Human-Computer Interaction and Their Effects on Brain Activity”, *PLOS One*.
- [9] D. F. Rogers, *An Introduction to NURBS with Historical Perspective*.
- [10] M. Kamermans, tiešsaistes grāmata par Bezjē līknēm, <https://pomax.github.io/bezierinfo>, pārbaudīts 25.05.2019.
- [11] Dr. C.-K. Shene lekciju materiāli par vienmērīgā sadalījuma parametrizācijas metodi,
<https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/INT-APP/PARA-uniform.html>, pārbaudīts 25.05.2019.
- [12] Dr. C.-K. Shene lekciju materiāli par hordu garuma parametrizacijas metodi,
<https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/INT-APP/PARA-chord-length.html>, pārbaudīts 25.05.2019.

- [13] E. Cohen, C. L. O'Dell "A Data Dependent Parametrization for Spline Approximation", *Mathematical Methods in Computer Aided Geometric Design*.
- [14] Dr. C.-K. Shene lekciju materiāli par centripetālā spēka parametrizacijas metodi, <https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/INT-APP/PARA-centripetal.html>, pārbaudīts 25.05.2019.
- [15] T. A. Foley, G. M. Nielson, "Knot Selection for Parametric Spline Interpolation", *Mathematical Methods in Computer Aided Geometric Design*.
- [16] Tiešsaistes grafiskais kalkulators *Desmos*, <https://www.desmos.com/calculator>, pārbaudīts 25.05.2019.
- [17] Quantitative Software Management, "Performance Benchmark Tables", <https://estudijas.lu.lv/mod/resource/view.php?id=169157>, pārbaudīts 25.05.2019.
- [18] C# programmēšanas stila standarti .NET satvarā <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>, pārbaudīts 25.05.2019
- [19] Tiešsaistes *Microsoft Office 365* rīks, <https://office.live.com/start/Word.aspx>, pārbaudīts 25.05.2019.

PIELIKUMI

1. pielikums Programmatūras koda fragments

```
1 // Finish a <Composite> curve, that has only two control points, but is indicated
2 // as finished.
3 private void AddOnlycPointsComposite(int i)
4 {
5     Point firstcPoint = new Point();
6     Point firstHandle = new Point();
7     Point secondHandle = new Point();
8     Point lastcPoint = new Point();
9
10    // first and last control points are knot points of the curve:
11    firstcPoint = pPointsAll[i][0];
12    lastcPoint = pPointsAll[i][1];
13
14    double sin60 = Math.Sin(Math.PI / 3);
15    double cos60 = Math.Cos(Math.PI / 3);
16
17    // each control point will be the midpoint of firstcPoint-lastcPoint curve
18    // segment, rotated by 60 degrees
19    // first we find oordinates of the midpoint:
20    double xMidpoint = 0.5 * (lastcPoint.X - firstcPoint.X);
21    double yMidpoint = 0.5 * (lastcPoint.Y - firstcPoint.Y);
22
23    // then we rotate the midpoint by 60 degrees:
24    firstHandle.X = Convert.ToInt32(cos60 * xMidpoint - sin60 * yMidpoint +
25        firstcPoint.X);
26    firstHandle.Y = Convert.ToInt32(sin60 * xMidpoint + cos60 * yMidpoint +
27        firstcPoint.Y);
28
29    // for control points of the curve to be on different sides, change the signs
30    // for second handle:
31    secondHandle.X = Convert.ToInt32(cos60 * -xMidpoint - sin60 * -yMidpoint +
32        lastcPoint.X);
33    secondHandle.Y = Convert.ToInt32(sin60 * -xMidpoint + cos60 * -yMidpoint +
34        lastcPoint.Y);
35
36    cPointsAll[i].Add(firstcPoint);
37    cPointsAll[i].Add(firstHandle);
38    cPointsAll[i].Add(secondHandle);
39    cPointsAll[i].Add(lastcPoint);
40
41    pbCanva.Invalidate();
42
43    return;
44 }
45
46 // Add the very first control point that's not a knot point for <Composite> curve
47 // with at least three knot points.
48 private Point GetVeryFirstHandle(Point firstpPoint, Point nextHandle, Point
49     secondpPoint)
50 {
```

```

44     Point veryFirstHandle = new Point();
45
46     // coordinates of the very first handle is calculated from first, third and
47     // fourth control points of the <Composite> curve
48
49     // We can look at these calculations as vector operations.
50     // First, we calculate dot product of vectors secondpPoint-nextHandle and
51     // secondpPoint-firstpPoint:
52     double dotProduct = (nextHandle.X - secondpPoint.X) * (firstpPoint.X -
53         secondpPoint.X) +
54             (nextHandle.Y - secondpPoint.Y) * (firstpPoint.Y -
55                 secondpPoint.Y);
56
57     //We need to find how long the vector v1 from veryFirstHandle to nextHandle
58     // needs to be, so that the middle control points are symmetrical.
59     //The symmetry can be achieved if the vector v1 is parallel to vector v2 from
60     // secondpPoint to firstpPoint
61     //and has the length: length(v2) - 2*length(projection of vector secondpPoint-
62     // nextHandle on to v2). One projection length for each side.
63     //Using projection formula, we get: proportion = |v2| - 2 * dot / |v2| . We
64     // will multiply this proportion by unit vector
65     //parallel to vector v2, which can be expressed as v2 / |v2|. If we devide our
66     // proportion with |v2| from the unit vector,
67     //we get: proportion = 1 - 2 * dot / |v2|^2
68
69     //That means, the length of the vector we will add equals
70     double prop = 1 - 2 * dotProduct / (Math.Pow(GetLength(firstpPoint,
71         secondpPoint), 2));
72
73     //Lastly, to point nextHandle we add vector parallel to vector firstpPoint-
74     // secondpPoint scaled by the proportion:
75     veryFirstHandle.X = Convert.ToInt32(nextHandle.X + prop * (firstpPoint.X -
76         secondpPoint.X));
77     veryFirstHandle.Y = Convert.ToInt32(nextHandle.Y + prop * (firstpPoint.Y -
78         secondpPoint.Y));
79
80     // We have achieved a "symmetrical" point to nextHandle; both of these points
81     // are on the same side of the bezier curve.
82
83     return veryFirstHandle;
84 }
85
86
87 // Calculate coordinates of first handle for <Composite> curves in a way to ensure
88 // C2 continuity.
89 private Point GetFirstHandle(Point prevpPoint, Point thispPoint, Point nextpPoint)
90 {
91     Point firstHandle = new Point();
92     double lengthPrevThis = GetLength(prevpPoint, thispPoint);
93     double lengthThisNext = GetLength(thispPoint, nextpPoint);
94
95     // Distance from first to second handle is half the distance from prevpPoint (
96     // a) to nextpPoint (b).
97     // The proportions of the length of each handle are the same as proportion ab/
98     // bc, where b thispPoint.
99     // Methods of calculations for distances and angles of handles can be
100    // different and there isn't one best method.
101    // I have discovered that this method works nice most of the time and isn't
102    // computationally expensive.
103
104    double proportion = 0.5 * lengthPrevThis / (lengthPrevThis + lengthThisNext);
105
106    firstHandle.X = thispPoint.X + Convert.ToInt32(proportion * (prevpPoint.X -
107        nextpPoint.X));
108    firstHandle.Y = thispPoint.Y + Convert.ToInt32(proportion * (prevpPoint.Y -
109            nextpPoint.Y));

```

```

        nextpPoint.Y));
89
90     return firstHandle;
91 }
92
93
94 // Calculate coordinates of second handle for <Composite> curves in a way to
95 // ensure C2 continuity.
96 private Point GetSecondHandle(Point prevPoint, Point thispPoint, Point nextpPoint
97 )
98 {
99     Point secondHandle = new Point();
100    double lengthPrevThis = GetLength(prevPoint, thispPoint);
101    double lengthThisNext = GetLength(thispPoint, nextpPoint);
102
103    //Calculations are very similar to those in the function GetFirstHandle.
104
105    double proportion = 0.5 * lengthThisNext / (lengthPrevThis + lengthThisNext);
106
107    secondHandle.X = thispPoint.X + Convert.ToInt32(proportion * (nextpPoint.X -
108        prevPoint.X));
109    secondHandle.Y = thispPoint.Y + Convert.ToInt32(proportion * (nextpPoint.Y -
110        prevPoint.Y));
111
112    return secondHandle;
113 }
114
115
116 // Add two last control points of a <Composite> curve that is indicated as
117 // finished and has at least three knot points
118 private Point GetVeryLastHandle(Point prevPoint, Point prevHandle, Point
119     lastpPoint)
120 {
121     Point veryLastHandle = new Point();
122
123     // Coordinates of the very last handle is calculated from first, second and
124     // fourth control points of the <Composite> curve's last segment
125
126     // We can look at these calculations as vector operations.
127     // First we calculate dot product of vectors lastpPoint-prevHandle and
128     // lastpPoint-prevHandle:
129     double dotProduct = (prevHandle.X - lastpPoint.X) * (prevPoint.X - lastpPoint
130         .X) +
131             (prevHandle.Y - lastpPoint.Y) * (prevPoint.Y - lastpPoint
132                 .Y);
133
134     // Calculations are very similar to those in the function GetVeryFirstHandle.
135     // To find how long the vector from prevHandle to veryLastHandle needs to be,
136     // we find the proportion:
137     double proportion = 1 - 2 * dotProduct / (Math.Pow(GetLength(prevPoint,
138         lastpPoint), 2));
139
140     // Lastly, to point prevHandle we add vector parallel to vector prevPoint-
141     // lastpPoint scaled by the proportion:
142     veryLastHandle.X = Convert.ToInt32(proportion * (prevPoint.X - lastpPoint.X)
143         + prevHandle.X);
144     veryLastHandle.Y = Convert.ToInt32(proportion * (prevPoint.Y - lastpPoint.Y)
145         + prevHandle.Y);
146
147     // We have achieved a "symmetrical" point to prevHandle; both of these points
148     // are on the same side of the bezier curve.
149
150     return veryLastHandle;
151 }
152
153

```

```

137
138     // Modify coordinates of a chosen control point.
139     private void ModifyCPoint(MouseEventArgs e)
140     {
141         int i = localPoint.Item1;
142         int j = localPoint.Item2;
143
144         modifyCurveType = allCurves[i];
145
146         if (modifyCurveType == BezierType.pPoints || modifyCurveType == BezierType.
147             LeastSquares)
148         {
149             lblError.Text = "Not allowed to move control points of <" +
150                 modifyCurveType + "> curve!";
151             modifyCurveType = BezierType.Nothing;
152             localPoint = null;
153         }
154
155         else if (modifyCurveType == BezierType.Composite)
156         {
157             if (rbKeyboardModify.Checked == true)
158             {
159                 lblError.Text = "Not allowed to move control points of <Composite>
160                     curve by keyboard!";
161                 localPoint = null;
162                 modifyCurveType = BezierType.Nothing;
163             }
164
165             // every third control point on a <Composite> curve is also a knot point
166             // therefore moved as a knot point
167             else if (j % 3 == 0)
168             {
169                 localPoint = null;
170                 modifyCurveType = BezierType.Nothing;
171             }
172
173             else if (e.Button == MouseButtons.Left)
174             {
175                 movedCurve[i] = MoveType.LeftClick;
176             }
177
178             else if (e.Button == MouseButtons.Right)
179             {
180                 movedCurve[i] = MoveType.RightClick;
181             }
182
183             return;
184         }
185
186         else if (rbKeyboardModify.Checked == true)
187         {
188             // when modifying points by keyboard, initialize the form of coordinates
189             FormCoordinates form_KeyboardAdd = new FormCoordinates(FormType.Modify,
190                 modifyCurveType);
191             form_KeyboardAdd.ShowDialog();
192
193             movedCurve[i] = MoveType.pPoints;
194             modifyCurveType = BezierType.Nothing;
195             localPoint = null;
196         }
197
198         pbCanva.Invalidate();
199     }
200

```

```

197     // Modify coordinates of a chosen knot point.
198     private void ModifyPoint()
199     {
200         int i = localPoint.Item1;
201         modifyCurveType = allCurves[i];
202
203         if (rbKeyboardModify.Checked == true)
204         {
205             // when modifying points by keyboard, initialize the form of coordinates
206             FormCoordinates form_KeyboardAdd = new FormCoordinates(FormType.Modify,
207                         modifyCurveType);
208             form_KeyboardAdd.ShowDialog();
209
210             if (modifyCurveType == BezierType.pPoints || modifyCurveType == BezierType
211                 .LeastSquares)
212             {
213                 AddcPointsInterpolation(i);
214             }
215
216             modifyCurveType = BezierType.Nothing;
217             localPoint = null;
218         }
219
220         pbCanva.Invalidate();
221         return;
222     }
223
224     // To ensure C2 continuity, when dragging a control point of a <Composite> curve
225     // with the left mouse button,
226     // the opposite handle needs to move as well.
227     private void ModifyHandleComposite(Point modifyHandle, Point middlePoint, Point
228                                         oppositeHandle, int opposite)
229     {
230
231         // It doesn't make mathematical sense and makes an error for two control
232         // points in <Composite> curve segment to have the same location.
233         if (middlePoint == modifyHandle)
234         {
235             modifyHandle.X++;
236             modifyHandle.Y++;
237         }
238
239         //We can look at these calculations as vector operations. We want for vector
240         //middle-change to keep its length,
241         //but change its direction so it starts from middle point and is parallel to
242         //moving-middle vector.
243         //To do that, we take unit vector from moving-middle (devide moving-middle
244         //with its length) and multiply that by
245         //middle-change length. Finally, we add that to middle point.
246
247         double proportion = GetLength(middlePoint, oppositeHandle) / GetLength(
248             modifyHandle, middlePoint);
249
250         oppositeHandle.X = Convert.ToInt32(middlePoint.X + proportion * (middlePoint
251                         .X - modifyHandle.X));
252         oppositeHandle.Y = Convert.ToInt32(middlePoint.Y + proportion * (middlePoint
253                         .Y - modifyHandle.Y));
254
255         cPointsAll[localPoint.Item1][opposite] = oppositeHandle;
256
257         return;
258     }

```

```

251
252 // To ensure C2 continuity and make sure no other points move when dragging a
253 // control point of a <Composite> curve with the right mouse button,
254 //the control point can only be moved in a straight line away from the middle
255 // point.
256 private void ModifyHandleCompositeStraight(Point modifyHandle, Point middlepPoint,
257 Point oppositeHandle)
258 {
259     const int maxDistanceToMouse = 100; // maximum distance between mouse location
260     // and control point being dragged; chosen arbitrary
261
262     int i = localPoint.Item1;
263     int j = localPoint.Item2;
264
265     if (GetLength(modifyHandle, cPointsAll[i][j]) > maxDistanceToMouse)
266     {
267         return;
268     }
269
270     Point result = new Point();
271
272     // To move the control point in straight line, we take unit vector from the
273     // middlepPoint to
274     // the place control point was before moving (modifyHandle). It's known that
275     // modifyHandle was on the needed curve.
276     // Than we scale this unit vector by the distance mouse is from the
277     // middlepPoint and at last add this vector to the middlepPoint.
278
279     double prop = GetLength(middlepPoint, modifyHandle) / GetLength(oppositeHandle
280     , middlepPoint);
281
282     result.X = Convert.ToInt32(middlepPoint.X + prop * (middlepPoint.X -
283     oppositeHandle.X));
284     result.Y = Convert.ToInt32(middlepPoint.Y + prop * (middlepPoint.Y -
285     oppositeHandle.Y));
286
287     cPointsAll[i][j] = result;
288
289     return;
290 }
291
292 // Modify coordinates of a chosen knot point of <Composite> curve.
293 public static void ModifyPPointComposite(Point mouseLocation)
294 {
295     int i = localPoint.Item1;
296     int j = localPoint.Item2;
297
298     Point pointOld = new Point();
299     pointOld = pPointsAll[i][j];
300
301     // every knot point of <Composite> curve is also a control point; change both
302     // these point coordinates:
303     pPointsAll[i][j] = mouseLocation;
304     cPointsAll[i][j * 3] = mouseLocation;
305
306     // We can look at these calculations as vector operations.
307     // We want for the adjacent handles of the knot point to stay in the same
308     // position relative to the knot point.
309     // To do that, we take vectors from knot point to control points and add those
310     // vectors to the new knot point coordinates.
311
312     Point newcPoint = new Point();
313
314     // first knot point doesn't have the first handle

```

```

303     if (j != 0)
304     {
305         newcPoint.X = mouseLocation.X - pointOld.X + cPointsAll[i][j * 3 - 1].X;
306         newcPoint.Y = mouseLocation.Y - pointOld.Y + cPointsAll[i][j * 3 - 1].Y;
307         cPointsAll[i][j * 3 - 1] = newcPoint;
308     }
309
310     //last knot point doesn't have the second handle
311     if (j != pPointsAll[i].Count - 1)
312     {
313         newcPoint.X = mouseLocation.X - pointOld.X + cPointsAll[i][j * 3 + 1].X;
314         newcPoint.Y = mouseLocation.Y - pointOld.Y + cPointsAll[i][j * 3 + 1].Y;
315         cPointsAll[i][j * 3 + 1] = newcPoint;
316     }
317
318     return;
319 }
320
321
322 // Change parametrization method and show the method being used now.
323 private void ChangeParametrization()
324 {
325     lblError.Text = "";
326
327     int i = localPoint.Item1;
328     ParamType paramType = parametrization[i];
329
330     if (allCurves[i] == BezierType.cPoints || allCurves[i] == BezierType.Composite
331         )
332     {
333         lblError.Text = "<" + allCurves[i] + "> curves does not use
334             parametrization!";
335         return;
336     }
337
338     // Show the real parametrization type of the selected curve:
339
340     if (paramType == ParamType.Uniform)
341     {
342         rbUniform.Checked = true;
343     }
344
345     else if (paramType == ParamType.Chord)
346     {
347         rbChord.Checked = true;
348     }
349
350     else if (paramType == ParamType.Centripetal)
351     {
352         rbCentripetal.Checked = true;
353     }
354
355     isChangingParam = true;
356 }
357
358 // Find if there is a control or knot point near mouse location
359 private void FindLocalPoint(List<List<Point>> PointsAll, Point MouseLocation)
360 {
361     const int localRadius = 7; // radius of neiborhood, used when selecting a
362         point with mouse; chosen arbitrary
363
364     for (int i = 0; i < PointsAll.Count; i++)
365     {
366         if (PointsAll[i] != null)

```

```

365    {
366        for (int j = 0; j < PointsAll[i].Count; j++)
367        {
368            if (GetLength(MouseLocation, PointsAll[i][j]) < localRadius)
369            {
370                localPoint = new Tuple<int, int>(i, j);
371            }
372        }
373    }
374
375    return;
376}
377
378
379 // Calculate control points for interpolated curves - <4 pPoints> and <Least
380 // Squares>.
381 private void AddcPointsInterpolation(int i)
382 {
383     List<Point> pList = pPointsAll[i];
384
385     // This method of curve fitting uses least squares method, so that distance
386     // errors from given knot points to the Bezier curve
387     // at respective t values is the smallest possible.
388     // To get control point coordinates, we will use formula C = M^1 * ( T^T * T )
389     // ^1 * T^T * P
390     // For more calculation information see documentation.
391
392     // We will represent Bezier curve in its matrix form.
393
394     // Matrix M contains coefficients of an expanded Bezier curve function. We
395     // will use only cubic Bezier curves therefor M always is:
396     var matrix = Matrix<double>.Build;
397     double[,] arrayM4 = new double[4, 4]
398     {
399         { 1, 0, 0, 0 },
400         { -3, 3, 0, 0 },
401         { 3, -6, 3, 0 },
402         { -1, 3, -3, 1 }
403     };
404
405     // Matrix P contains coordinates of all knot points:
406     double[,] arrayP = new double[pList.Count, 2];
407     for (int j = 0; j < pList.Count; j++)
408     {
409         arrayP[j, 0] = pList[j].X;
410         arrayP[j, 1] = pList[j].Y;
411     }
412
413     var matrixP = matrix.DenseOfArray(arrayP);
414     var matrixM4 = matrix.DenseOfArray(arrayM4);
415     var matrixM4Inv = matrixM4.Inverse();
416
417     // Bezier curves are parametric, so we need appropriate t values to tie each
418     // knot point to coordinates of points on the curve.
419     // This parametrization can be done in different ways; we will store the
420     // resulting t values in a list sValues.
421     List<double> sValues = new List<double>();
422
423     if (parametrization[i] == ParamType.Uniform)
424     {
425         sValues = GetsValuesUniform(pList);
426     }
427
428     else if (parametrization[i] == ParamType.Chord)

```

```

424     {
425         sValues = GetsValuesChord(pList);
426     }
427
428     else if (parametrization[i] == ParamType.Centripetal)
429     {
430         sValues = GetsValuesCentripetal(pList);
431     }
432
433     var matrixS = matrix.DenseOfArray(GetArrayS(sValues));
434     var matrixSTranspose = matrixS.Transpose();
435     var matrixSSTr = matrixSTranspose * matrixS;
436     var matrixSSTrInv = matrixSSTr.Inverse();
437
438     var matrixMul1 = matrixM4Inv * matrixSSTrInv;
439     var matrixMul2 = matrixMul1 * matrixSTranspose;
440
441     var matrixC = matrixMul2 * matrixP;
442
443     // if this is the first time calculating control points
444     if (cPointsAll[i] == null)
445     {
446         cPoints = new List<Point>();
447
448         for (int j = 0; j < 4; j++)
449         {
450             Point tmp = new Point(Convert.ToInt32(matrixC[j, 0]), Convert.ToInt32(
451                 matrixC[j, 1]));
452             cPoints.Add(tmp);
453         }
454         cPointsAll[i] = cPoints;
455     }
456
457     // if we are modifying a curve
458     else
459     {
460         for (int j = 0; j < 4; j++)
461         {
462             Point tmp = new Point(Convert.ToInt32(matrixC[j, 0]), Convert.ToInt32(
463                 matrixC[j, 1]));
464             cPointsAll[i][j] = tmp;
465         }
466     }
467
468     return;
469 }
470
471 // Bezier curve parametrization method where t values are equally spaced.
472 private List<double> GetsValuesUniform(List<Point> pList)
473 {
474     List<double> sValues = new List<double>();
475
476     for (int i = 0; i < pList.Count; i++)
477     {
478         double s = (double)i / (pList.Count - 1);
479         sValues.Add(s);
480     }
481     return (sValues);
482 }
483
484 // Bezier curve parametrization method where t values are aligned with distance
485 // along the polygon of control points.
486 private List<double> GetsValuesChord(List<Point> pList)

```

```

486    {
487        // At the first point, we're fixing t = 0, at the last point t = 1. Anywhere
488        // in between t value is equal to the distance
489        // along the polygon scaled to the [0,1] domain.
490
491        List<double> sValues = new List<double>();
492
493        // First we calculate distance along the polygon for each point:
494        List<double> dPoints = new List<double> { 0 };
495        for (int i = 1; i < pList.Count; i++)
496        {
497            double d = dPoints[i - 1] + GetLength(pList[i - 1], pList[i]);
498            dPoints.Add(d);
499        }
500
501        // Then we scale these values to [0, 1] domain:
502        for (int i = 0; i < pList.Count; i++)
503        {
504            double s = dPoints[i] / dPoints[pList.Count - 1];
505            sValues.Add(s);
506        }
507
508        return (sValues);
509    }
510
511    // Bezier curve parametrization method where t values are aligned with square root
512    // of the distance along the polygon.
513    private List<double> GetsValuesCentripetal(List<Point> pList)
514    {
515        // At the first point, we're fixing t = 0, at the last point t = 1. Anywhere
516        // in between t value is equal to the
517        // square root of the distance along the polygon scaled to the [0,1] domain.
518
519        List<double> sValues = new List<double>();
520
521        // First we calculate the square root of distance along the polygon for each
522        // point:
523        List<double> dPoints = new List<double> { 0 };
524        for (int i = 1; i < pList.Count; i++)
525        {
526            double d = dPoints[i - 1] + Math.Sqrt(GetLength(pList[i - 1], pList[i]));
527            dPoints.Add(d);
528        }
529
530        // Then we scale these values to [0, 1] domain:
531        for (int i = 0; i < pList.Count; i++)
532        {
533            double s = dPoints[i] / dPoints[pList.Count - 1];
534            sValues.Add(s);
535        }
536
537        return (sValues);
538    }

```

Mazāko kvadrātu metodes pierādījums Bezjē līknes interpolācijai

Mazāko kvadrātu metode ir populāra regresijas analīzes metode, kas visbiežāk tiek pielietota, lai veiktu līknes piemeklēšanu dotiem punktiem [1]. Līknes piemeklēšana tiek veikta, minimizējot attālumu kvadrātu summu starp dotajiem punktiem un meklēto līkni.

Dots:

- n mezglu punkti ($n \geq 4$), apzīmēti ar p_i , sakārtoti matricā \mathbf{P} .
- katram punktam p_i parametrizācijas rezultātā piemeklēta t vērtība, apzīmēta ar s_i .

Vēlamies caur dotajiem punktiem izvilkst tuvāko trēsās pakāpes Bezjē līkni $B(t)$, tas ir, atrast tādus līknes kontrolpunktus $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$, sakārtotus matricā \mathbf{C} , lai attālumi starp punktiem p_i un līkni, ko raksturo minētie kontrolpunkti, būtu pēc iespējas mazākas.

Bezjē līkne ir parametriska, savukārt mezgli tiek uzdoti ar koordinātām. Lai saistītu mezglu punktus ar līkni, tiek izmantotas parametrizācijas metodes. Arī attālumus d_i meklēsim starp katru punktu un tā piemeklēto s_i vērtību.

$$d_i = |(p_i - B(s_i))| = \sqrt{(p_i - B(s_i))^2}$$

Lai izvairītos no kvadrātsaknēm un moduļiem, apskatīsim attālumu kvadrātus d_i^2 .

$$d_i^2 = (p_i - B(s_i))^2$$

Vēlamies atrast tuvāko Bezjē līkni – tādu, kurai attālumu d_i summa ir pēc iespējas mazāka.

$$\sum_{i=1}^n d_i^2 = \sum_{i=1}^n (p_i - B(s_i))^2$$

Pārrakstam minēto summu kā funkciju \mathbf{D} , kas atkarīga no Bezjē līknes kontrolpunktiem $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3$.

$$D(\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4) = \sum_{i=1}^n (p_i - B(s_i))^2$$

Izmantojam matricu pierakstu Bezjē līknēm.

$$\begin{aligned} D(\mathbf{C}) &= (\mathbf{P} - \mathbf{SMC})^{*2} \\ D(\mathbf{C}) &= (\mathbf{P} - \mathbf{SMC})^T (\mathbf{P} - \mathbf{SMC}) \end{aligned}$$

Funkcijai $D(\mathbf{C})$ vēlamies atrast mazāko vērtību, t. i., funkcijas minimumu. Ekstrēma nepieciešamie nosacījumi pieprasī, lai minimuma punktā funkcijas atvasinājums būtu vienāds ar 0.

Varam ievērot, ka matrica $(\mathbf{P} - \mathbf{SMC})$ ir ar dimensijām $n \times 1$, tātad tas ir vektors. Divu vektoru \mathbf{a} un \mathbf{b} skalārais reizinājums $\langle \mathbf{a}, \mathbf{b} \rangle$ uzrakstāms kā matricu reizinājums $\mathbf{a}^T \cdot \mathbf{b}$. Tāpāt $(\mathbf{P} - \mathbf{SMC})^T(\mathbf{P} - \mathbf{SMC})$ ir vektoru $(\mathbf{P} - \mathbf{SMC})$ skalārais reizinājums pašam ar sevi.

$$D(\mathbf{C}) = \langle \mathbf{P} - \mathbf{SMC}, \mathbf{P} - \mathbf{SMC} \rangle$$

Turklāt, vektoriem reālo skaitļu laukos darbojas distributīvais un komutatīvais likums.

$$D(\mathbf{C}) = \langle \mathbf{P}, \mathbf{P} \rangle - 2\langle \mathbf{P}, \mathbf{SMC} \rangle + \langle \mathbf{SMC}, \mathbf{SMC} \rangle$$

Šo summu var atvasināt pa saskaitājamiem.

1. $\langle \mathbf{P}, \mathbf{P} \rangle$ atvasinājums:

$$\frac{\partial \langle \mathbf{P}, \mathbf{P} \rangle}{\partial \mathbf{C}} = 0, \text{ jo } \mathbf{P} \text{ nesatur } \mathbf{C}.$$

2. $\langle \mathbf{P}, \mathbf{SMC} \rangle$ atvasinājums:

Varam ievērot, ka

$$\langle \mathbf{P}, \mathbf{SMC} \rangle = \mathbf{P}^T \mathbf{SMC} = \langle \mathbf{M}^T \mathbf{S}^T \mathbf{P}, \mathbf{C} \rangle,$$

kā arī vektoriem $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

$$\frac{\partial \langle \mathbf{y}, \mathbf{x} \rangle}{\partial x_i} = \frac{\partial (y_1 x_1 + y_2 x_2 + \dots + y_i x_i + \dots + y_n x_n)}{\partial x_i} = y_i.$$

Izmantojot ievērotās sakarības, iegūstam:

$$\frac{\partial \langle \mathbf{P}, \mathbf{SMC} \rangle}{\partial c_i} = \frac{\partial \langle \mathbf{M}^T \mathbf{S}^T \mathbf{P}, \mathbf{C} \rangle}{\partial c_i} = \mathbf{M}^T \mathbf{S}_i^T \mathbf{P}.$$

3. $\langle \mathbf{SMC}, \mathbf{SMC} \rangle$ atvasinājums:

Varam ievērot, ka

$$\langle \mathbf{SMC}, \mathbf{SMC} \rangle = \mathbf{C}^T \mathbf{M}^T \mathbf{S}^T \mathbf{SMC} = \langle \mathbf{M}^T \mathbf{S}^T \mathbf{SMC}, \mathbf{C} \rangle,$$

kā arī vektoram $\mathbf{x} \in \mathbb{R}^n$ un matricai $\mathbf{A} \subset \mathbb{R}^{n \times n}$

$$\frac{\partial \langle \mathbf{Ax}, \mathbf{x} \rangle}{\partial x_i} = (\mathbf{A}_i + \mathbf{A}_i^T) \mathbf{x}.$$

Pilnais izvedums šajai identitātei atrodams 3. pielikumā.

Izmantojam minēto identitāti un veicam ekvivalentus pārveidojumus.

$$\begin{aligned} \frac{\partial}{\partial \mathbf{c}_i} \langle \mathbf{M}^T \mathbf{S}^T \mathbf{SMC}, \mathbf{C} \rangle &= (\mathbf{M}^T \mathbf{S}_i^T \mathbf{S}_i \mathbf{M} + (\mathbf{M}^T \mathbf{S}_i^T \mathbf{S}_i \mathbf{M})^T) \mathbf{C} \\ &= (\mathbf{M}^T \mathbf{S}_i^T \mathbf{S}_i \mathbf{M} + \mathbf{M}^T \mathbf{S}_i^T \mathbf{S}_i \mathbf{M}) \mathbf{C} \\ &= 2(\mathbf{M}^T \mathbf{S}_i^T \mathbf{S}_i \mathbf{M}) \mathbf{C} \\ &= 2\mathbf{M}^T \mathbf{S}_i^T \mathbf{S}_i \mathbf{MC} \end{aligned}$$

Tātad esam ieguvuši trešā funkcijas saskaitāmā atvasinājumu:

$$\frac{\partial \langle \mathbf{SMC}, \mathbf{SMC} \rangle}{\partial \mathbf{c}_i} = 2\mathbf{M}^T \mathbf{S}_i^T \mathbf{S}_i \mathbf{MC}.$$

Apvienojot saskaitāmo parciālos atvasinājumus, iegūstam

$$\frac{\partial D}{\partial \mathbf{c}_i} = -2\mathbf{M}^T \mathbf{S}_i^T \mathbf{P} + 2\mathbf{M}^T \mathbf{S}_i^T \mathbf{S}_i \mathbf{MC},$$

savukārt apvienojot i -tos locekļus, iegūstam

$$\frac{\partial D}{\partial \mathbf{C}} = -2\mathbf{M}^T \mathbf{S}^T \mathbf{P} + 2\mathbf{M}^T \mathbf{S}^T \mathbf{SMC}.$$

Lai atrastu funkcijas $D(\mathbf{C})$ ekstrēmu, pielīdzinām tās atvasinājumu nullei.

$$\begin{aligned} -2\mathbf{M}^T \mathbf{S}^T \mathbf{P} + 2\mathbf{M}^T \mathbf{S}^T \mathbf{SMC} &= 0 \\ \mathbf{M}^T \mathbf{S}^T \mathbf{P} - \mathbf{M}^T \mathbf{S}^T \mathbf{SMC} &= 0 \\ \mathbf{M}^T \mathbf{S}^T \mathbf{SMC} &= \mathbf{M}^T \mathbf{S}^T \mathbf{P} \\ \mathbf{C} &= (\mathbf{M}^T \mathbf{S}^T \mathbf{SM})^{-1} \mathbf{M}^T \mathbf{S}^T \mathbf{P} \end{aligned}$$

\mathbf{M} - trīsstūra matrica, kurai uz diagonāles ir četri nenualles elementi $\Rightarrow rank(\mathbf{M}) \geq 4$.

$$\left. \begin{aligned} rank(\mathbf{M}) &\geq 4 \\ \mathbf{M} &\subset \mathbb{R}^{4 \times 4} \end{aligned} \right\} \Rightarrow rank(\mathbf{M}) = 4$$

Matricai \mathbf{M} ir pilns rangs, tātad tai eksistē inversā matrica, tāpat arī matricai \mathbf{M}^T eksistē inversā matrica. Tātad \mathbf{M} un \mathbf{M}^T var iznest no iekavām.

$$\begin{aligned} \mathbf{C} &= \mathbf{M}^{-1} (\mathbf{S}^T \mathbf{S})^{-1} (\mathbf{M}^T)^{-1} \mathbf{M}^T \mathbf{S}^T \mathbf{P} \\ \mathbf{C} &= \mathbf{M}^{-1} (\mathbf{S}^T \mathbf{S})^{-1} \mathbf{I} \mathbf{S}^T \mathbf{P} \\ \mathbf{C} &= \mathbf{M}^{-1} (\mathbf{S}^T \mathbf{S})^{-1} \mathbf{S}^T \mathbf{P} \end{aligned}$$

Esam atraduši kritisko punktu. Varam ievērot, ka funkcijai neeksistē maksimums - ja kontrolpunktus novieto patvalīgi tālu no dotajiem mezgli punktiem, funkcijas $D(\mathbf{C})$ vērība tiecas uz bezgalību. Secinām, ka atrastais ekstrēms ir minimums.

Tātad, izmantojot mazāko kvadrātu metodi, punktiem \mathbf{P} tuvāko Bezjē līkni definē kontrolpunkti \mathbf{C} , kas izsakāmi kā

$$\mathbf{C} = \mathbf{M}^{-1} (\mathbf{S}^T \mathbf{S})^{-1} \mathbf{S}^T \mathbf{P}.$$

Izmantotie apzīmējumi:

- \mathbf{A}^T Matricas \mathbf{A} transponētā matrica.
- \mathbf{A}^{-1} Matricas \mathbf{A} inversā matrica.
- \mathbf{A}_i Matricas \mathbf{A} i -tā kolonna.
- \mathbf{A}^{*2} Matrica \mathbf{A} , kur katrs matricas elements ir kāpināts kvadrātā.

[1] S. Boyd, L. Vandenberghe, *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares*.

Skalārā reizinājuma $\langle \mathbf{Ax}, \mathbf{x} \rangle$ atvasinājums

$$\begin{aligned}
\langle \mathbf{Ax}, \mathbf{x} \rangle &= \\
&= \left(\begin{pmatrix} a_{11}x_1 & a_{12}x_2 & \cdots & a_{1n}x_n \\ a_{21}x_1 & a_{22}x_2 & \cdots & a_{2n}x_n \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}x_1 & a_{n2}x_2 & \cdots & a_{nn}x_n \end{pmatrix}, (x_1 \ x_2 \ \cdots \ x_n) \right) = \\
&= x_1(a_{11}x_1 + \cdots + a_{1n}x_n) + x_2(a_{21}x_1 + \cdots + a_{2n}x_n) + \cdots + x_n(a_{n1}x_1 + \cdots + a_{nn}x_n) =
\end{aligned}$$

$$= \sum_{j=1}^n (x_j \sum_{k=1}^n a_{jk}x_k)$$

Atvasinām šo izteiksmi pa saskaitāmajiem pēc x_i .

$$\begin{aligned}
\frac{\partial \langle \mathbf{Ax}, \mathbf{x} \rangle}{\partial x_i} &= \\
&= x_1a_{1i} + x_2a_{2i} + \cdots + x_{i-1}a_{(i-1)i} + \\
&\quad + x'_i(a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n) + x_i(a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n)' + \\
&\quad + x_{i+1}a_{(i+1)i} + \cdots + x_na_{ni} = \\
&= x_1a_{1i} + x_2a_{2i} + \cdots + x_{i-1}a_{(i-1)i} + \\
&\quad + (a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n) + x_ia_{ii} + \\
&\quad + x_{i+1}a_{(i+1)i} + \cdots + x_na_{ni} = \\
&= x_1a_{1i} + x_2a_{2i} + \cdots + x_{i-1}a_{(i-1)i} + x_ia_{ii} + x_{i+1}a_{(i+1)i} + \cdots + x_na_{ni} + \\
&\quad + (a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n) = \\
&= \sum_{j=1}^n a_{ji}x_j + \sum_{k=1}^n a_{ik}x_k = \\
&= (a_{1i} \ a_{2i} \ \cdots \ a_{ni}) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + (a_{i1} \ a_{i2} \ \cdots \ a_{in}) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \\
&= ((a_{1i} \ a_{2i} \ \cdots \ a_{ni}) + (a_{i1} \ a_{i2} \ \cdots \ a_{in})) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \\
&= (\mathbf{A}_i^T + \mathbf{A}_i)\mathbf{x}
\end{aligned}$$

Izmantotie apzīmējumi:

- \mathbf{A}^T Matricas \mathbf{A} transponētā matrica.
- \mathbf{A}_i Matricas \mathbf{A} i -tā kolonna.
- a_{ij} Matricas \mathbf{A} elements, kas atrodas i -tajā rindā un j -tajā kolonnā.

Kvalifikācijas darbs „*Rīks Bezjē līkņu konstruēšanai un modificešanai*” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: *Elīza Gaile* _____.05.2019.

Rekomendēju darbu aizstāvēšanai

Darba vadītājs: *doc., Dr. dat. Vineta Arnicāne* _____.05.2019.

Recenzents: *Dr. sc. ing. Edžus Žeiris*

Darbs iesniegts 27.05.2019.

Kvalifikācijas darbu pārbaudījumu komisijas sekretāre: *Darja Solodovņikova* _____

Darbs aizstāvēts kvalifikācijas darbu pārbaudījuma komisijas sēdē
_____.06.2019. prot. Nr. _____

Komisijas sekretārs(-e): _____