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**

Šajā dokumentā ir aprakstīta grafisku objektu interaktīva modificēšanas sistēma, kas paredzēta kubisku Bezjē līkņu konstruēšanai un modificēšanai izmantojot datorpeli, datora tastatūru vai teksta failus. Sistēma piedāvā ģenerēt četru dažādu veidu Bezjē līknes, tai skaitā interpolētas līknes un saliktas līknes ar C2 nepārtrauktību, 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 augšuplādēt fona attēlu.

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**

This document describes a system for interactive modification of graphical objects, meant for Bezier curve creation and modification using mouse, keyboard or text files. System allows to create four types of curves, including Bezier curve fitting and composite Bezier curves with C2 continuity, and to choose from three different parameterization methods. Constructed curves can be interactively modified using mouse or by changing input data using keyboard. It’s also possible to output control points and/or knot points. For visual comparison of curves, it is possible to upload a background image.

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, Bezier curves, interpolation.

**SATURA RĀDĪTĀJS**

**IEVADS**

Lai konstruētu divdimensionālus datorgrafiskus attēlus, parasti tiek izmantoti punkti, taisnes un līknes. Kamēr punktu un taišņu izveide ir triviāla, līkņu konstruēšana prasa vairāk pūļu, jo datoram nepieciešama matemātiska funkcija, kas to apraksta. Viens no plaši izplatītiem un ērtiem gludu līkņu uzdošanas veidiem datorgrafikā ir Bezjē līknes, it īpaši kubiskas Bezjē līknes, tāpēc tās tika izvēlētas šim projektam. Bezjē līknes tiek izmantotas arī, piemēram, PostScript, SVG, Microsoft Paint un Adobe Illustrator programmās.

Kvalifikācijas darba rezultāts ir Bezjē līkņu rīks, kas ļauj konstruēt un modificēt līknes. Šī programma tiks pielietota iekšējām uzņēmuma vajadzībām, lai manuāli modelētu un salīdzinātu apģērba piegrieznes. Lai gan eksistē vairāki citi rīki, kuros ietvertas līdzīgas funkcionalitātes, izveidotā programma apvieno visas projektam nepieciešamās funkcionalitātes vienā un neietver nevajadzīgas funkcijas. Izstrādais rīks atbalsta četru veidu kubiskas Bezjē līknes, kas atšķiras konstruēšanas veidā:

1. uzdodot četrus kontrolpunktus,
2. uzdodot četrus punktus, caur kuriem jāizvelk līkne (interpolēta Bezjē līkne),
3. uzdodot vairāk kā četrus punktus, caur kuriem pēc iespējas tuvu jāizvelk Bezjē līkne (interpolācija izmantojot mazāko kvadrātu metodi),
4. saliktu Bezjē līkņu konstruēšana, nodrošinot C2 nepārtrauktību katrām divām secīgām līknēm.

**Nolūks**

Šī dokumenta nolūks ir iepazīstināt sistēmas izstrādātājus un lietotājus ar sistēmas prasībām un projektējumu, funkcijām un rīka funkcionalitātes pielietojumiem, kā arī nodrošināt prasību saskaņošanu turpmākajā izstrādes un uzturēšanas procesā.

**Darbības sfēra**

Programmatūra paredzēta manuālai apģērba piegriežņu modelēšanai un salīdzināšanai, izmantojot kubiskās Bezjē līknes. Sistēmas lietotāji var konstruēt un modificēt četru veidu Bezjē līknes, izvēlēties piemērotāko parametrizācijas metodi, konstruētās līknes modificēt un dzēst, izvadīt par tām datus un veikt manuālu salīdzināšanu ar jau esošām piegrieznēm.

**Saistība ar citiem dokumentiem**

Dokuments tika izstrādāts saskaņā ar programmatūras prasību specifikācijas standartu LVS 68:1996 „Programmatūras prasību specifikācijas ceļvedis” un saskaņā ar programmatūras projektējuma apraksta standartu LVS 72:1996 “Ieteicamā prakse programmatūras projektējuma aprakstīšanai”.

**Pārskats**

Kvalifikācijas darba dokuments ir sadalīts piecās galvenajās daļās:

* Vispārējs apraksts – apraksta produktu vispārējā līmenī un iekļauj informāciju par sistēmas lietotājiem, darījumprasībām. Aprakstīti arī ierobežojumi, pieņēmumi un atkarības.
* Programmatūras prasību specifikācija – ietver funkcionālās un nefunkcionālās prasības.
* Programmatūras projektējuma apraksts – ietver daļēju funkciju un saskaņu projektējumu.
* Programmatūras testēšana – ietver testpiemēru projektējumu un testēšanas žurnālu ar rezultātiem.
* Projekta organizācija – apraksta programmatūras izstrādes procesus, iekļauj arī darbietilpības novērtējumu, kvalitātes nodrošināšanu un konfigurāciju pārvaldību.

**APZĪMĒJUMU SARAKSTS**

**PPS**

**PPA**

**GIT**

**C#**

**Microsoft .NET satvars**

**Windows Forms**

**Bezjē līkne**

**Kubiska Bezjē līkne**

**kontrolpunkti**

**mezglu punkti**

**Bezjē līknes galapunkts**

**Sākumpunkts**

**Beigu punkts**

**Rokturis**

**Pretējais rokturis**

**Blakus esošie rokturi var izdomāt kkadu “kaimini” or smth**

**px**

**Bezjē līknes rokturi**

**C2 nepārtrauktība**

**Šajā dokumentā vektori tiek apzīmēti ar mazo bold burtu, piemēram:**

**Matricas tiek apzīmētas ar lielo bold burtu, piemēram:**

**Skalāri lielumi tiek apzīmēti ar mazo italic burtu, piemēram:**

**Vienkāršības labad, punkts tiek uztverts kā vektors.**

**Funkcijas tiek apzīmētas ar lielo burtu italic, piemēram:** *F*

1. **VISPĀRĒJS APRAKSTS**
   1. **Esošā stāvokļa apraksts**

Pieejamas vairākas programmatūras (piemēram, Adobe Illustrator, GIMP, Inkscape), kas piedāvā līkņu konstruēšanas un modificēšanas funkcionalitātes, kā arī piegriežņu modelēšanas programmatūras (piemēram, GRAFIS). Tomēr šīm programmatūrām ir ierobežotas interpolācijas, līkņu konstruēšanas un modificēšanas iespējas, turklāt tās piedāvā daudz lieku funkcionalitāšu, tādējādi apgrūtinot sistēmas paredzēto lietotāju.

* 1. **Pasūtītājs**

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

* 1. **Produkta perspektīva**

Sistēma apvieno visas funkcionalitātes, kas nepieciešamas pasūtītājam, lai veiktu manuālu piegriežņu izstrādi un salīdzināšanu datorā, vienlaikus neapgrūtinot lietotāju ar liekām funkcionalitātēm.

* 1. **Darījumsprasības**

Programmatūrai jānodrošina iespēja manuāli modelēt un salīdzināt apģērba piegrieznes. Apģērbu piegrieznes tiek modelētas izmantojot interpolāciju caur patvaļīgi uzdotiem punktiem, kā arī veicot manuālu līkņu modificēšanu. Rīkam jābūt ērti izmantojamam un konstruēto piegriežņu līknēm jābūt vizuāli pievilcīgām.

* 1. **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 modelētu un salīdzinātu apģērba piegrieznes. Lietotājs izmanto rīka saskarni un tam ir pieejamas visas programmatūras funkcionalitātes.

* 1. **Vispārējie ierobežojumi**

Sistēmas lietošanai jābūt vieglai un intuitīvai.

* 1. **Pieņēmumi un atkarības**

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

1. **PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA**
   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 C2 nepārtrauktībai) un tām jābūt vizuāli pievilcīgām.
3. Konstruētās līknes jāspēj interaktīvi modificēt 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 piegrieznē.
6. Dažādu nozīmju grafiskajiem objektiem (piemēram, interpolācijas mezglu punktiem un palīgpunktiem) jābūt atšķiramiem ar vizuālo izskatu, piemēram, krāsojumu.
7. Jānodrošina iespēja konstruētās līknes dzēst.
   1. **Nefunkcionālās prasības**

Grafisko objektu konstruēšanai, modificēšanai, dzēšanai un datu izvadei jānotiek ne ilgāk kā 0,1 sekunžu laikā. Līkņu interaktīvai modificēšanai ar datorpeli jābūt “bez aizķeršanās”.

1. **PROGRAMMATŪRAS PROJEKTĒJUMA APRAKSTS**
   1. **Projektējuma matemātiskais pamatojums**
      1. **Kubisku Bezjē līkņu izmantošanas pamatojums**

Programmatūras mērķis ir konstruēt divdimensionālus līkņu attēlus datorā, interpolējot patvaļīgi dotus punktus – caur punktiem izvelkot gludas līknes. Atšķirībā no dotiem punktiem un taisnēm, konkrētu līkņu attēlošana datorā nav triviāla – nepieciešama precīza matemātiska funkcija, kas šo līkni 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 modificējamas, turklāt tām piemīt liela elastība (atšķirībā no, piemēram, polinomu funkcijām, kur katram argumentam atbilst viena vērība).

Viens no visplašāk izmantotajiem līkņu uzdošanas veidiem datorgrafikā ir Bezjē līknes, 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, Microsoft Paint, SVG un citās.

Ir definētas visu kārtu Bezjē līknes. Pirmās kārtas Bezjē līkne ir nogrieznis, savukārt ceturtās un augstāku kārtu Bezjē līknes ir intensīvas skaitļošanas ziņā. Tādēļ datorgrafikā vispopulārākās ir kvadrātiskās un kubiskās Bezjē līknes. Atceroties, ka Bezjē līknes kārta nosaka arī tās maksimālo pārliekuma punktu skaitu, var secināt, ka kubiskās Bezjē līknes piedāvā lielāku pielietojamību, turklāt tās ir implementētas C# System.Drawing bibliotēkā. Ņemot vērā šos apsvērumus, programmatūra tiek projektēta izmantojot kubiskās Bezjē līknes.

* + 1. **Bezjē līkņu pamatteorija**

Bezjē līkne ir definētas jebkurai pakāpei un-tās kārtas Bezjē līkne pierakstāma ar formulu:

Kubiska Bezjē līkne ir izsakāma formā:

Ar tiek apzīmēti kontrolpunkti. Kontrolpunkti ir mainīgie, kas nosaka, kā Bezjē līkne tiek izliekta. Katru kontrolpunktu komplekts nosaka tieši vienu līkni. Varam ievērot, ka kontrolpunkti sākas ar indeksu 0.

Trešās pakāpes Bezjē līknei ir četri kontrolpunkti, divus no tiem sauc par galapunktiem (, ) un divus par rokturiem (, ). Līknes sākumpunkts ir un beigu punkts ir , līdz ar to šie punkti pieder līknei. Atšķirībā no galapunktiem, patvaļīgi izvēlēti rokturu punkti līknei nepieder. Skatīt 3.1. att.

*A picture containing sky

Description automatically generated*

3.1. att. **Bezjē līkne un tās kontrolpunkti**

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

Lai veiktu interpolāciju ar Bezjē līkni –, 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 patvaļīgiem mezglu punktiem. Ja tiek doti vairāk nekā četri punkti, caur kuriem jāveic interpolācija ar kubisku Bezjē līkni, var izmantot citas metodes:

1. Atrast tuvāko līkni dotajiem punktiem;
2. Konstruēt saliktu Bezjē līkni.

Šajā programmatūrā Bezjē līknes konstruēšanai tiek izmantoti četri minētie veidi:

1. Uzdodot četrus kontrolpunktus (<4 cPoints>);
2. Uzdodot četrus mezglu punktus (<4 pPoints>);
3. Atrodot tuvāko līkni vairāk nekā četriem uzdotiem mezglu punktiem (<Least Square>);
4. Veidojot saliktu Bezjē līkni (<Composite>).

* + 1. **Kubiskās Bezjē līknes interpolācija caur četriem punktiem**

Ja lietotājs vēlas novilkt līkni caur tieši četriem punktiem, iespējams atrast atbilstošos kontrolpunktus. Viens no variantiem, kā atrast šādu kontrolpunktu koordinātas, ir izmanot mazāko kvadrātu metodi. Šī metode atrod kontrolpunktu koordinātas minimizējot distanci starp dotajiem mezglu punktiem un līkni. Zināms, ka trešās pakāpes Bezjē līkni iespējams izvilkt caur jebkādiem četriem mezglu punktiem, tātad mazāko kvadrātu metode atradīs līkni, kurai distanču kvadrātu summa ir nulle. Bezjē līkne ir parametriska un, lai aprakstītu punktu uz tās, tiek specificēta vērtība, savukārt mezglu punkti tiek uzdoti ar koordinātām. Lai mērītu un minimizētu distanci starp mezglu punktu un līkni, nepieciešams saistīt dotos punktus ar punktiem uz līknes, tas ir, katram mezglu punktam nepieciešams piesaistīt vērtību. Šo procesu sauc par parametrizāciju (skatīt nodaļu 3.1.7) un tās rezultātā atrasto vērtību katram mezgla punktam apzīmēsim ar .

No šīm vērtībām tiek izveidota matrica , kas interpolācijas aprēķinos aizstāj matricu ar simboliskajām vērtībām:

Lai aprēķinātu kontrolpunktus (sakārtotus matricā ), kuru veidotā līkne interpolē mezglu punktus (sakārtotus matricā ), jāveic sekojoši aprēķini:

Pilnu mazāko kvadrātu metodes izvedumu un pierādījumu skatīt pielikumā Nr. 1

* + 1. **Tuvākās līknes metodes izmantošanas pamatojums un teorija**

Ja lietotājs vēlas novilkt 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 patvaļīgu punktu gadījumā (), distanču kvadrātu summa nebūs nulle un matrica būs formā:

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

* + 1. **Saliktu līkņu izmantošanas pamatojums un teorija**

Ja lietotājs vēlas novilkt 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. Lai gan iespējams katriem četriem punktiem uzdot vienu Bezjē līkni, tādējādi minimizējot līkņu 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 C2 nepārtrauktībai. Lai nodrošnātu šo nepārtrauktību, katram mezglu punktam (izņemot pirmo un pēdējo) un tā blakus esošajiem rokturiem jābūt uz vienas taisnes. Šāda veida saliktas Bezjē līknes sauc par B-līknēm, un tās ir iespējams konstruēt dažādos veidos. Kādā leņķī un cik tālu no mezglu punkta ir blakus esošie rokturi ir maināmi lielumi un raksturo B-līknes veidu.

* + 1. **B-līkņu konstruēšanas un modificēšanas metode**

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

* Katrs mezglu punkts (izņemot pirmais un pēdējais) un tā blakus esošie rokturi ir uz vienas taisnes (skatīt 3.2. att.);
* Mezglu punkta blakus esošo rokturu veidotā taisne ir paralēla blakus esošo mezglu veidotajai taisnei (skatīt 3.2. att.);
* Attālums starp blakus esošajiem rokturiem vienāds ar pusi no attāluma starp blakus esošajiem mezgliem (skatīt 3.2. att.);
* Attālums starp mezglu un blakus esošajiem rokturiem proporcionāls attālumam starp mezglu un blakus esošajiem mezgliem (skatīt 3.2. att.);
* Pirmais/pēdējais B-līknes rokturis tiek atlikts simetriski otrajam/priekšpēdējam B-līknes rokturim (skatīt 3.2. att.);
* Ja B-līkne sastāv no 2 mezglu punktiem, attālums no roktura līdz tam tuvākajam mezglu punktam vienāds ar pusi no attāluma starp mezglu punktiem; leņķis starp mezglu punktu veidoto taisni un taisni, ko veido rokturis un tā tuvākais mezglu punkts, vienāds ar (skatīt 3.3. att.).

A close up of a device

Description automatically generated

3.2 att. **B-līkņu konstruēšanas metodes piemērs**

A screenshot of a cell phone

Description automatically generated

3.3. att. **B-līkņu konstruēšanas piemērs divu mezglu gadījumā**

Aprakstītais B-līkņu veids ir cilvēkam intuitīvs, tā implementācija ir vienkārša un neprasa daudz resursu. Rīka konstruētās saliktās Bezjē līknes ir gludas un, izņemot retus gadījumus (piemēram, asus pagriezienus), atbilstošas vizuālajām prasībām.

* + 1. **Parametrizācijas metodes**

Bezjē līkņu parametrizācija ir mezglu punktu koordināšu sasaiste ar Bezjē līkni . Parametrizācijas rezultātā katram mezglam tiek atrasta atbilstoša vērtība, apzīmēta ar . Parametrizācija nepieciešama, lai veiktu Bezjē līkņu interpolāciju. Caur četriem punktiem iespējams izvilkt 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”).**

Tiek pieņemts, ka dotie punkti būs vienmērīgi izkārtoti, un atbilstošā vērtība punktam atroda kā , kur – mezglu skaits. Piemēram, ja tiek doti 5 punkti, attiecīgās 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.

1. **Hordu garuma parametrizācijas metode (“Chord length”).**

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 vērtību punktam atrod kā lauztās līnijas garumu no pirmā līdz -tajam mezglam, mērogotu intervālā. Aprēķināms kā , kur , ar apzīmē garumu starp dotajiem punktiem, – lauztās līnijas garums, – mezglu skaits. Šī ir vispopulārākā Bezjē līkņu parametrizācijas metode. Tai nav apjomīgas skaitļošanas operācijas un visbiežāk līnijas, kas konstruētas izmantojot hordu garuma parametrizāciju, dod vēlamus rezultātus. Bieža nevēlama parādība šai metodei ir lieli izliekumi (skatīt 3.4. att.).

1. **Centrtieces spēka parametrizācijas metode (“Centripetal Method”).**

Šī parametrizācijas metode atsaucas uz centrtieces spēku. Ja mēs iedomājamies mašīnu, kas brauc pa Bezjē līkni, pagriezienos tai būtu jāsamazina ātrums, lai centrtieces spēks nav pārlieku liels un tā neizslīd no trajektorijas – spēkam vajadzētu būt proporcionālam izmaiņām leņķī. Centrtieces spēka parametrizācija ir aproksimācija šim modelim. Atbilstošo vērtību punktam aprēķina kā , kur , ar apzīmē garumu starp dotajiem punktiem, , – mezglu skaits. Var ievērot, ka šī metode ir ļoti līdzīga hordu garuma metodei, bet kvadrātsaknes dēļ garākām hordām tiek samazināta ietekme, savukārt īsākām – palielināta. Šī iemesla dēļ centrtieces spēka parametrizācija labāk tiek galā ar asiem pagriezieniem (skatīt 3.4. att).

A picture containing text

Description automatically generated

3.4. att. **Parametrizācijas metožu salīdzinājums**

Visām parametrizācijas metodēm ir savas priekšrocības un trūkumi un nav viena “labākā metode”, it īpaši ņemot vērā, ka līknes “labums” ir daļēji subjektīvs (skat 3.5. att.). 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.

A picture containing map

Description automatically generated

3.5. att. **Parametrizācijas metožu salīdzinājums**

* + 1. **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 tās 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 mezglu punkts starp tiem vienmēr atrastos uz vienas taisnes. Šis nosacījums nepieciešams, lai saglabātu līknes C2 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 C2 nepārtrauktību. Skatīt 3.7. att.
  + Patvaļīgi pārvietojot mezglu punktu. Lai līknei arī pēc mezgla modificēšanas piemistu C2 nepārtrauktība, bet pārējie līknes segmenti modifikācijas rezultātā nemainītos, ērtākais veids ir nemainīt mezgla blakus esošo rokturu relatīvo atrašanās vietu pret minēto mezglu. Skatīt 3.8. att.

A close up of a map

Description automatically generated

3.6. att. **Saliktas līknes kontrolpunkta patvaļīga modificēšana**

**A screenshot of a cell phone

Description automatically generated**

3.7. att. **Saliktas līknes kontrolpunkta ierobežota modificēšana**

**A close up of a map

Description automatically generated**

3.8. att. **Saliktas līknes mezgla punkta modificēšana**

* 1. **Funkciju projektējums**

Te ļoti iespējams uzrakstīšu ievadtekstu par nodaļu vai varbūt par vispārējo projektējuma arhitektūru(?). Par to, ka tiek taisīts ar formām un ka ir vairākas formas? Vai arī šis iet uzreiz pēc virsraksta “PPA”? Kā arī, jānomaina visi trešā līmeņa virsraksti šajā formā un jēdzīgākiem. Tabulās pelēkajam krāsojumam jābūt pēc iespējas gaišākam un, ja tabula turpinās pa vairakām lapām, jāpieliek virsraksti no jauna.

* + 1. **Galvenā loga pamatfunkcijas**

|  |  |  |
| --- | --- | --- |
| **Nosaukums** | **Funkcija** | **Apraksts** |
| FM01\_MD | void **pbCanva\_MouseDown**(object sender, MouseEventArgs e) | Izpildās, kad vadīklā pbCanva tiek nospiests peles kursors. Ļauj pievienot jaunus vai izvēlēties eksistējošos kontrolpunktus un mezglu punktus. |
| FM02\_MM | void **pbCanva\_MouseMove**(object sender, MouseEventArgs e) | Izpildās, kad vadīklā pbCanva tiek kustināts peles kursors. Ļauj veikt interaktīvu līkņu modificēšanu ar peli un nosaka raustītās līnijas galapunktu, atliekot līknes <4 cPoints> punktus. |
| FM03\_MU | void **pbCanva\_MouseUp**(object sender, MouseEventArgs e) | Izpildās, kad vadīklā pbCanva tiek atlaists datorpeles taustiņš. Pārtrauc līknes modificēšanu ar peli. |
| FM04\_P | void **pbCanva\_Paint**(object sender, PaintEventArgs e) | Attēlo visus grafiskos objektus vadīklā pbCanva – Bezjē līknes, kontrolpunktus, mezglu punktus, kontrolpunktu nogriežņus un raustīto līniju atliekot līknes <4 cPoints> punktus. |
| FM05\_RS | void **FormMain\_Resize**(object sender, EventArgs e) | Izpildās, kad tiek veikta formas FormMain izmēru maiņa. Nodrošina, ka forma ir reaģējoša (uz izmēra izmaiņām). |
| FM06\_UB | void **btnUploadBackground\_Click**(object sender, EventArgs e) | Augšuplādē un iestata lietotāja izvēlētu fona attēlu vadīklai pbCanva. |
| FM07\_SB | void **cbShowBackground\_ CheckStateChanged**(object sender, EventArgs e) | Maina augšuplādētā fona attēla redzamību uz pretējo. |
| FM08\_RA | void **btnResetAll\_Click**(object sender, EventArgs e) | Inicializē formu no jauna. |

void **pbCanva\_Paint**(object sender, PaintEventArgs e):

Funkcija attēlo grafisku objektus:

* Visas Bezjē līknes melnā krāsā ar līnijas biezumu 1px.
* Visus kontrolpunktus izņemot <4 pPoints> līkņu galapunktus un <Composite> līkņu kontrolpunktus, kas nav rokturi. Kontrolpunkti tiek attēloti kā sarkanas riņķa līnijas ar centru kontrolpunktā, rādiusu 2px un līnijas biezumu 1px.
* Visus mezglu punktus kā melnus riņķus ar centru kontrolpunktā un rādiusu 2px.
* Visus nogriežņus starp katriem diviem secīgiem kontrolpunktiem līknēs <4 cPoints>, <4 pPoints> un <Least Squares> un starp katriem diviem secīgiem rokturiem <Composite> līknēs gaiši pelēkā krāsā ar līnijas biezumu 1px.
* Nogriezni starp pēdējo kontrolpunktu nepabeigtā <4 cPoints> līknē un datorpeles kursora atrašanās vietu ar raustītu līniju biezumā 1px, gaiši pelēkā krāsā.
  + 1. **Jaunu līkņu izveides funkcijas**

|  |  |  |
| --- | --- | --- |
| **Nosaukums** | **Funkcija** | **Apraksts** |
| NC01\_NL | void **NewLine**(BezierType lineType) | Izveido jaunu līkni. |
| NC02\_4C | void **btnNew4cPoints\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnNew4cPoints. Atļauj veikt līknes <4 cPoints> kontrolpunktu ievadi ar datorpeli, klaviatūru vai no .txt faila. |
| NC03\_4P | void **btnNew4pPoints\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnNew4pPoints. Atļauj veikt līknes <4 pPoints> mezglu punktu ievadi ar datorpeli, klaviatūru vai no .txt faila. |
| NC04\_LS | void **btnNewLeastSquares\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnNewLeastSquares. Atļauj veikt līknes <Least Squares> mezglu punktu ievadi ar datorpeli, klaviatūru vai no .txt faila. |
| NC05\_C | void **btnNewComposite\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnNewComposite. Atļauj veikt līknes <Composite> mezglu punktu ievadi ar datorpeli, klaviatūru vai no .txt faila. |
| NC06\_DC | void **btnDoneComposite\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnDoneComposite. Atzīmē, ka pēdējai <Composite> līknei nepieciešams pievienot pēdējos kontrolpunktus. |
| NC07\_AC | void **AddcPoint**(Point mouseLocation) | Pievieno ar peli ievadīta kontrolpunkta koordinātes izvēlētai līknei. |
| NC08\_AP | void **AddpPoint**(Point mouseLocation) | Pievieno ar peli ievadīta mezglu punkta koordinātes izvēlētai līknei. |
| NC09\_FF | List<Point> **GetPointsfromFile**() | Ielasa .txt failu, pārvērš tā tekstu par punktu koordinātām un atgriež sarakstu ar šiem punktiem. |

* + 1. **Līkņu konstruēšanas funkcijas**

|  |  |  |
| --- | --- | --- |
| **Nosaukums** | **Funkcija** | **Apraksts** |
| CC01\_CI | void **AddcPointsInterpolation**(int i) | Veic līkņu <4 pPoints> un <Least Squares> interpolāciju izmantojot izvēlētās līknes mezglu punktus. Aprēķina un saglabā interpolētās līknes kontrolpunkus. |
| CC02\_VU | List<double> **GetsValuesUniform**(List<Point> pList) | Aprēķina un atgriež sarakstu ar vērtībām mezglu punktiem izvēlētai līknei. Vērtības tiek iegūtas izmantojot vienmērīgā sadalījuma parametrizācijas metodi. |
| CC03\_VCh | List<double> **GetsValuesChord**(List<Point> pList) | Aprēķina un atgriež sarakstu ar vērtībām mezglu punktiem izvēlētai līknei. Vērtības tiek iegūtas izmantojot hordu garuma parametrizācijas metodi. |
| CC04\_VCe | List<double> **GetsValuesCentripetal**(List<Point> pList) | Aprēķina un atgriež sarakstu ar vērtībām mezglu punktiem izvēlētai līknei. Vērtības tiek iegūtas izmantojot centrtieces spēka parametrizācijas metodi. |
| CC05\_AS | double[,] **GetArrayS**(List<double> sValues) | Aprēķina un atgriež divdimensiju masīvu, kas reprezentē matricu . Masīvs tiek aizpildīts, izmantojot parametrizācijas rezultātā iegūtas vērtības. |
| CC06\_CC | void **AddcPointsComposite**(int i) | Aprēķina un saglabā kontrolpunktu koordinātas <Composite> līknēm ar vismaz trīs mezglu punktiem. |
| CC07\_OC | void **AddOnlycPointsComposite**(int i) | Aprēķina un saglabā kontrolpunktu koordinātas <Composite> līknēm ar diviem mezglu punktiem. |
| CC08\_VF | Point **GetVeryFirstHandle**(Point firstpPoint, Point oppositeHandle, Point secondpPoint) | Aprēķina un atgriež paša pirmā roktura koordinātas mezglu 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 mezglu 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 mezglu 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 mezglu 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 **AddcPointsComposite**(int i):

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

* Katrs trešais kontrolpunkts sakrīt ar līknes -to mezglu punktu;
* Katrs kontrolpunkts formā ir pirmais rokturis, aprēķināms ar funkciju **GetFirstHandleComposite()**;
* Katrs kontrolpunkts formā ir otrais rokturis, aprēķināms ar funkciju **GetSecondHandleComposite()**;
* Kontrolpunkts ir saliktās līknes pats pirmais rokturi, aprēķināms ar funkciju **GetVeryFirstHandleComposite()**;
* Kontrolpunkts 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.

Līknes pirmais kontrolpunkts un pēdējais kontrolpunkts sakrīt ar attiecīgi līknes pirmo mezglu un otro mezglu punktu . Rokturu veidotais leņķis ar taisni, ko veido mezglu punkti, ir , savukārt attālumi no rokturiem līdz tiem tuvākajiem mezglu punktiem vienādi ar pusi no attāluma starp mezglu punktiem. Tātad rokturu koordinātas aprēķināmas ar formulu:

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.6. nodaļā.

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

Funkcija aprēķina un atgriež pirmā roktura koordinātas mezgla punktam, kas ir sākumpunkts un pieder saliktai Bezjē līknei ar vismaz 3 mezgliem.

Funkcijai tiek padoti divi mezglu punkti sākumpunkta mezgls (firstpPoint) un līknes nākamais mezgls (secondpPoint), kā arī nākamā mezgla pirmais rokturis (nextHandle). Punkti tiek skaitīti no saliktās līknes sākumpunkta līdz galapunktam.

Aprēķinus var izteikt kā vektoru darbības. Ieviešam sekojošus apzīmējumus:

* – vektors no secondpPoint uz firstpPoint;
* – vektors no secondpPoint uz nextHandle;
* – vektors no nextHandle uz rokturi .

Roktura koordinātas aprēķināmas kā. Vektora virziens ir tāds pats kā vektoram , tā garums izsakāms kā garums mīnus divi garumi projekcijai uz :

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.6. nodaļā.

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

Funkcija aprēķina un atgriež pirmā roktura koordinātas mezglu punktam, kas nav galapunkts un pieder saliktai Bezjē līknei.

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

Aprēķinus var izteikt kā vektoru darbības. Vektoram no thispPoint uz pirmo rokturi būs tāds pats virziens, kā vektoram no nextpPoint uz prevpPoint. Apzīmējam attālumu no prevpPoint līdz thispPoint ar un attālumu no thispPoint līdz nextpPoint ar . Tad vektoru var aprēķināt kā pusi no vektora reiz . Esam ieguvuši, ka pirmā roktura koordinātas izsakāmas ar formulām:

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 no thispPoint uz otro rokturi būs tāds pats virziens, kā vektoram no prevpPoint uz nextpPoint. Apzīmējam attālumu no prevpPoint līdz thispPoint ar un attālumu no thispPoint līdz nextpPoint ar . Tad vektoru var aprēķināt kā pusi no vektora reiz . Esam ieguvuši, ka pirmā roktura koordinātas izsakāmas ar formulām:

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:

* – vektors no prevpPoint uz lastpPoint;
* – vektors no prevpPoint uz prevHandle;
* – vektors no prevHandle uz pēdējo rokturi .

Roktura koordinātas aprēķināmas kā. Vektora virziens ir tāds pats kā vektoram , turklāt tā garums izsakāms kā garums mīnus divi garumi projekcijai uz :

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.6. nodaļā.

* + 1. **Līkņu modificēšanas funkcijas**

|  |  |  |
| --- | --- | --- |
| **Nosaukums** | **Funkcija** | **Apraksts** |
| CM01\_bCP | void **btnChangeParam\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnChangeParam. Atļauj izvēlēties līkni, kurai veikt līkņu parametrizācijas metodes maiņu. |
| CM02\_CP | void **ChangeParametrization**() | Pārbauda, vai izvēlētajai līknei ir parametrizācijas metode. Atļauj veikt līkņu <4 pPoints> un <Least Squares> parametrizācijas metodes maiņu. |
| CM03\_UC | void **rbUniform\_CheckedChanged**(object sender, EventArgs e) | Izpildās, kad tiek mainīts vadīklas rbUniform statuss. Veic līkņu <4 pPoints> un <Least Squares> parametrizācijas metodes maiņu. |
| CM04\_CC | void **rbChord\_CheckedChanged**(object sender, EventArgs e) | Izpildās, kad tiek mainīts vadīklas rbChord statuss. Veic līkņu <4 pPoints> un <Least Squares> parametrizācijas metodes maiņu. |
| CM05\_bMC | void **btnModifycPoints\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnModifycPoints. Atļauj veikt kontrolpunktu koordināšu modificēšanu. |
| CM06\_bMP | void **btnModifypPoints\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnModifypPoints. Atļauj veikt mezglu punktu koordināšu modificēšanu. |
| CM07\_MC | void **ModifycPoint**(MouseEventArgs e) | Pārbauda, vai izvēlētais kontrolpunkts var tikt modificēts, nosaka tā modificēšanas veidu. |
| CM08\_MP | void **ModifypPoint**() | Pārbauda, vai izvēlētais mezgla punkts var tikt modificēts, nosaka tā modificēšanas veidu. |
| CM09\_MH | void **ModifyHandleComposite**(Point modifyHandle, Point middlepPoint, Point oppositeHandle, int opposite) | Aprēķina un saglabā roktura oppositeHandle koordinātas. Punkta modifyHandle koordinātas tiek patvaļīgi modificētas izmantojot datorpeles kreiso taustiņu, un, lai nodrošinātu C2 nepārtrauktību, nepieciešams mainīt arī pretējā roktura koordinātas. |
| CM10\_MHS | void **ModifyHandleCompositeStraight**(Point modifyHandle, Point middlepPoint, Point oppositeHandle) | Aprēķina un saglabā roktura modifyHandle koordinātas, kad tas tiek modificēts izmantojot datorpeles labo taustiņu tā, lai nodrošinātu C2 nepārtrauktību, bet nemainītu citu punktu koordinātas. |
| CM11\_MPC | void **ModifypPointComposite**(Point mouseLocation) | Saglabā <Composite> līknes mezglu punkta un tā blakus esošo rokturu jaunās koordinātas, kad tās tiek modificētas ar datorpeli. Aprēķina blakus esošo rokturu koordinātas tā, lai nodrošinātu C2 nepārtrauktību. |

void **ModifyHandleComposite**(Point modifyHandle, Point middlepPoint, Point oppositeHandle, int opposite):

Lai nodrošinātu, ka saliktai līknei piemīt C2 nepārtrauktība, katram mezglam un tā blakus esošajiem rokturiem jābūt uz vienas taisnes. Ar datorpeles kreiso taustiņu patvaļīgi modificējot roktura modifyHandle koordinātas, nepieciešams mainīt arī pretējā roktura oppositeHandle koordinātas. Tās tiek aprēķinātas tā, lai starp rokturiem esošais mezglu punkts middlepPoint pieder taisnei, ko veido minētie rokturi, un attālums no middlepPoint līdz oppositeHandle ir nemainīgs.

Aprēķinus var izteikt kā vektoru darbības. Ieviešam apzīmējumus:

* – vektors no modifyHandle uz middlepPoint;
* – vektors no middlepPoint uz oppositeHandle;
* – vektors no middlepPoint uz jaunajām roktura koordinātām .

Vektoram jābūt tādam pašam virzienam kā un garumam jābūt vienādam ar . Varam izmantot formulu:

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.8. nodaļā.

void **ModifyHandleCompositeStraight**(Point modifyHandle, Point middlepPoint, Point oppositeHandle):

Lai modificētu saliktas līknes rokturi modifyHandle un nodrošinātu, ka pārējie līknes punkti nemaina savas koordinātas, iespējams izmantot peles labo taustiņu. Rokturi iespējams pārvietot tikai pa staru, kas pieder taisnei, ko veido pretējais rokturis oppositeHandle un mezglu punkts middlepPoint starp minētajiem rokturiem. Stara sākumpunkts ir middlepPoint.

Ērts veids, kā to implementēt, ir izmantojot vektoru darbības. Vektoram no middlepPoint uz modifyHandle jābūt tādam pašam virzienam, kā vektoram no oppositeHandle uz middlepPoint, savukārt attālumam no middlepPoint līdz jaunajām roktura koordinātām jāsakrīt ar . Varam izmantot formulu:

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

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.8. nodaļā.

void **ModifypPointComposite**(Point mouseLocation):

Funkcija aprēķina un saglabā saliktas līknes mezglu punkta un tā blakus esošo rokturu jaunās koordinātas, kad mezglu punkts tiek modificēts ar peli. Lai līknei arī pēc modificēšanas piemistu C2 nepārtrauktība un pārējie līknes segmenti modifikācijas rezultātā nemainītos, ērtākais veids ir nemainīt un relatīvo atrašanās vietu pret .

Aprēķinus var interpretēt kā vektoru darbības. Jaunās rokturu koordinātas var aprēķināt izmantojot formulu:

Pilnu metodes skaidrojumu un pamatojumu skatīt 3.1.8. nodaļā.

* + 1. **Esošu līkņu funkcijas**

|  |  |  |
| --- | --- | --- |
| **Nosaukums** | **Funkcija** | **Apraksts** |
| EC01\_LP | void **FindLocalPoint**(List<List<Point>> PointsAll, Point MouseLocation) | Atrod, vai peles kursora apkārtnē ir kāds kontrolpunkts vai mezglu punkts. |
| EC02\_OC | void **btnOutputcPoints\_Click**(object sender, EventArgs e) | Atļauj izvēlēties līkni, kurai izvadīt līknes kontrolpunktu koordinātes. |
| EC03\_OP | void **btnOutputpPoints\_Click**(object sender, EventArgs e) | Atļauj izvēlēties līkni, kurai izvadīt līknes mezglu punktu koordinātes. |
| EC04\_OCF | void **OutputcPointsToFile**() | Izvada izvēlētas līknes kontrolpunktu koordinātes .txt failā. |
| EC05\_OPF | void **OutputpPointsToFile**() | Izvada izvēlētas līknes mezglu punktu koordinātes .txt failā. |
| EC06\_bDL | void **btnDeleteLine\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnDeleteLine. Atļauj izvēlēties līkni, kuru izdzēst. |
| EC07\_DL | void **DeleteLine**(int i) | Izdzēš izvēlētu līkni. |

* + 1. **Koordināšu loga funkcijas**

|  |  |  |
| --- | --- | --- |
| **Nosaukums** | **Funkcija** | **Apraksts** |
| FC01\_ | void **InitializeAdd**() | Inicializē formu, ja tā izsaukta jaunas līknes pievienošanai izmantojot datora tastatūru. |
| FC01\_ | void **InitializeModify**() | Inicializē formu, ja tā izsaukta esošas līknes modificēšanai izmantojot datora tastatūru. |
| FC01\_ | void **InitializeOutput**() | Inicializē formu, ja tā izsaukta esošas līknes punktu izvadīšanai uz ekrāna. |
| FC01\_ | void **AddRow**() | Pievieno jaunu rindu un aizpilda to ar nepieciešamajām vadīklām, lai ievadītu vai izvadītu punktu koordinātas. |
| FC01\_ | void **btnAddRow\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnAddRow. Pievieno jaunu rindu pēc lietotāja vaicājuma. |
| FC01\_ | void **btnDeleteRow\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnDeleteRow. Pievieno jaunu rindu pēc lietotāja vaicājuma |
| FC01\_ | void **btnSubmitInput\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnSubmitInput. Pārbauda, vai visas vadīklas koordināšu ievadei ir aizpildītas, pārvērš ievadīto tekstu punktos un tos saglabā. |
| FC01\_ | void **btnResetInput\_Click**(object sender, EventArgs e) | Izpildās, kad tiek nospiesta vadīkla btnResetInput. Nodzēš tekstu visās vadīklās, kas paradzētas punktu koordināšu ievadei. |

* 1. **Saskarņu projektējums**

Kā minēts 3.2. nodaļā, programmatūrai ir divi izvades logi – galvenais logs (skat. 3.9. att.) un koordināšu logs (skat. 3.10. att.).

Galvenai 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 modificēšana ar peli (tai skaitā parametrizācijas izvēle un maiņa), esošu punktu izvēle, līkņu dzēšana, formas inicializācija no jauna, fona attēla redzamības maiņa. Jaunu līkņu konstruēšana no .txt faila, esošu punktu koordinātu izvade uz .txt failu un fona attēla augšuplādēšana arī neizmanto koordināšu logu. Galvenais logs ir uz tā izmēra maiņu reaģējošs, tam piesaistītas funkcijas, kas aprakstītas 3.2.1.-3.2.4. nodaļā.

Galvenajā logā ietverts:

1. – Lauks, kas paredzēts fona attēla un visu grafisko objektu attēlošanai. Reaģējoši maina izmēru, kad tiek veikta galvenā loga izmēra maiņa.

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

3. – df

4. – df

5. – df

6. – df

7. – df

8. – df

9. – df

10. – df

11. – df

12. – df

2. Pogas zem virsraksta “New Bezier of type” (2.) paredzētas jaunu līkņu pievienošanai. Pirmās četras no šīm pogām

**A screenshot of a cell phone

Description automatically generated**

3.9. att. **Nosaukums**

**A screenshot of a cell phone

Description automatically generated**

3.10. att. **Nosaukums**

1. **TESTĒŠANAS DOKUMENTĀCIJA**
   1. **Testpiemēru projektējums**
   2. **Testēšanas žurnāls**
2. **PROJEKTA ORGANIZĀCIJA**
   1. **Darbietilpības novērojums**
   2. **Kvalitātes nodrošināšana**
   3. **Konfigurāciju pārvaldība**

**IZMANTOTĀ LITERATŪRA UN AVOTI**

– 75.lpp pētījums “bezier curve fitting”

<https://pomax.github.io/bezierinfo/> – classic pomax

lietas no seienes: <https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/> :

<https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/Bezier/intro.html>

<https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/INT-APP/PARA-chord-length.html>

<https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/INT-APP/PARA-uniform.html>

myb tā viena grāmata?

<https://en.wikipedia.org/wiki/B%C3%A9zier_curve>

<https://en.wikipedia.org/wiki/Non-uniform_rational_B-spline>

<https://en.wikipedia.org/wiki/B-spline>

**PIELIKUMI**

1. **Programmatūras koda fragments**
2. **Mazāko kvadrātu metodes pierādījums**

https://www.overleaf.com/read/kyrvrzrdfyxb

* 1. **Skalārā reizinājuma atvasinājums**

https://www.overleaf.com/read/brzmwwsqpznd

1. **pielikums**

(Dokumentārā lapa)