

BABEŞ-BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
MAGYAR MATEMATIKA ÉS INFORMATIKA INTÉZET

Racionális ciklikus bázisfüggvények zárt görbék és felületek modellezésére



TÉMAVEZETŐ:

DR. RÓTH ÁGOSTON,
EGYETEMI ADJUNKTUS

SZERZŐ:

HADNAGY KINGA,
1. ÉVFOLYAM,
MAGISZTERI KÉPZÉS

Kivonat

Célunk olyan interaktív, OpenGL grafikus függvénykönyvtárra és C++ objektumorientált programozási nyelvre épülő keretrendszer kifejlesztése, mellyel a felhasználó tetszőleges, véges foksámú, zárt (racionális) trigonometrikus görbét és azok tenzor szorzataként előállított felületeket modellezhet kizárólagosan kontrollpont-alapú műveletekkel.

Az említett görbe- és felületosztály egzakt, kontrollpont-alapú reprezentációjához az n -edfokú trigonometrikus polinomok terének frissen bevezetett ciklikus bázisát és annak racionális kiterjesztését használjuk fel.

Kontrollpont-konfigurációk generálása mellett az alkalmazás tetszőleges foksámnövelésre is képes, mely alkalmazása során olyan kontrollpoligonokból (kontrollhálókból) álló sorozatot kapunk, mely az előállítandó görbe (felület) alakjához konvergál egyre finomabb alakváltoztatási lehetőséget biztosítva a modellező számára.

Tartalomjegyzék

1. Bevezető	4
1.1. A számítógépes geometriai tervezés	4
1.2. Görbék és felületek	4
1.3. Zárt görbék és felületek	5
1.4. Racionális görbék és felületek	6
2. Zárt racionális trigonometrikus görbék	9
2.1. Ciklikus bázis	9
2.2. Ciklikus görbék	9
2.3. Trigonometrikus görbék	11
2.4. Racionális trigonometrikus görbék modellezése	13
3. Zárt racionális trigonometrikus felületek	15
3.1. Ciklikus felületek	15
3.2. Trigonometrikus felületek	16
3.3. Racionális trigonometrikus felületek modellezése	18
4. Zárt racionális trigonometrikus görbék és felületek interaktív modellezése	20
4.1. Követelmények	20
4.1.1. Felhasználói követelmények	20
4.1.2. Rendszerkövetelmények	21
4.2. Felhasznált technológiák és függvénykönyvtárak	22
4.2.1. C++	22
4.2.2. OpenGL	22
4.2.3. Qt	26
4.3. Fontosabb osztályok interfésze és hierarchiája	27
4.3.1. GUI csomag	27
4.3.2. Core csomag	28
4.3.3. Cyclic csomag	40
4.3.4. Trigonometric csomag	43

4.3.5. Rational csomag	46
4.4. Alkalmazások és eredmények	46

Ábrák jegyzéke

1. A videokártya csővezetékének állapotai, [9]	23
2. Az (a) és (b) ábrákon egy-egy harmincadfokú ciklikus görbét láthatunk, melyek rendre a (25) hipocikloist illetve a (26) tóruszcsomót állítják elő egzaktul.	47
3. A Bernoulli-féle lemniszkáta egzakt kontrollpont alapú előállítása nyolcadfokú racionális trigonometrikus görbeként. Az ábrán kék színnel tüntettük fel a háromdimenziós őskép alakját és ciklikus reprezentációjához tartozó kontrollpontokat. Az őskép kontrollpontjait a $w = 1$ egyenletű hipersíkra vetítve, megkapjuk a lemniszkátát meghatározó – racionális előállításához szükséges – piros kontrollpontokat.	49
4. A (28) előállítású tórusz egzakt kontrollpont-alapú leírása (2, 3)-fokú ciklikus felülettel.	50

1. Bevezető

Ebben a fejezetben a számítógépes geometriai tervezésben a görbék és felületek leírására használt alapeszközöket ismertetjük, melyekre a dolgozat alapszik.

1.1. A számítógépes geometriai tervezés

A számítógépes geometriai tervezés polinomiális, racionális, (algebrai) trigonometrikus, vagy hiperbolikus bázisfüggvényekkel és kontrollpontok kombinációjával megadott zárt vagy nyílt görbék és felületek leírásával és kezelésével foglalkozik. Ez a terület kapcsolatban áll geometriai modellezéssel és adatillesztéssel. Az idevágó elméleti eredmények építészetben, gépiparban, elektronikai tervezés automatizálásában, gyártási folyamatok tervezésében, szoftver applikációkban, illetve adatpontok interpolációjakor és approximációjakor kerülnek felhasználásra.

1.2. Görbék és felületek

A számítógépes geometriai tervezésben a görbék leírására leggyakrabban a

$$\left\{ \begin{array}{l} \mathbf{g} : [a, b] \rightarrow \mathbb{R}^\delta, \delta \geq 2 \\ \mathbf{g}(u) = \sum_{i=0}^n F_i(u) \mathbf{d}_i \end{array} \right. \quad (1)$$

lineáris kombinációt használják, ahol $\mathbf{d}_i \in \mathbb{R}^\delta$ pontok az úgynevezett kontrollpontok, melyek egy kontrollpoligon csúcsait alkotják. Az (1) előállítás következménye, hogy egy görbe csupán pár kontrollpont segítségével határozható meg, ami előnyös adattárolás és adatátvitel szempontjából. Ráadásul az $\{F_i : [a, b] \rightarrow \mathbb{R}\}_{i=0}^n$ függvényrendszer megfelelő megválasztásával a képlet által eredményezett görbe követi kontrollpoligonjának alakját. A kontrollpoligon tehát egy intuitív modellezési eszköz a tervező számára.

Ha az F_i függvények folytonosak az $[a, b]$ intervallumon, az (1) képlet mindig görbét eredményez. A tervezéshez viszont további feltételek szükségesek. Sima görbék modellezéséhez elengedhetetlen, hogy a függvények folytonosan differenciálhatóak legyenek. Ha $\sum_{i=0}^n F_i(u) \equiv 1, \forall u \in [a, b]$, az (1) görbe alakja zárt a kontrollpontjai affin transzformációira nézve, azaz a transzformált görbe egybeesik a transzformált kontrollpontok által generált görbével. Hogyha

emellett a függvények nemnegatívak is, az (1) görbe a kontrollpontjainak komplex kombinációja, minek következtében a görbe alakja benne van a kontrollpontok konvex burkában. Ha az F_i függvények nemnegatívak és egy összegűek, az $\{F_i : [a, b] \rightarrow \mathbb{R}\}_{i=0}^n$ rendszert normalizálnak, vagy súlyfüggvényrendszernek nevezik. A súlyfüggvények lineáris függetlensége fontos az (1) képlettel leírt interpolációs görbék előállításakor. Ebben az esetben az F_i függvények egy függvénytér bázisát alkotják.

Tervezési célokra használt (1) előállítási görbéknek egy részhalmaza végpontbeli interpolációs feltételeket is teljesít, azaz a görbe keresztülmegy a kontrollpoligon $\mathbf{d}_0$ és $\mathbf{d}_n$ végpontjain. A legismertebb ezen görbék közül a Bézier görbe és annak racionális változata. A spline-görbék egy újabb fontos részhalmazát képezik az (1) görbéknek. Esetükben az $[a, b]$ értelmezési tartományt kisebb részintervallumokra osztjuk, és a bázisfüggvényeket ezeken az intervallumokon szakaszonként úgy értelmezzük, hogy bizonyos rendű folytonosságot is biztosítunk a kapcsolódási pontokban. A legismertebb spline-bázisfüggvények a normalizált B-spline bázisok és azok racionális változata, a NURBS függvények.

A számítógépes geometriai tervezésben felületek leírására leggyakrabban az

$$\begin{cases} \mathbf{s} : [a, b] \times [c, d] \rightarrow \mathbb{R}^3, \\ \mathbf{s}(u, v) = \sum_{i=0}^n \sum_{j=0}^m F_i(u) G_j(v) \mathbf{d}_{i,j} \end{cases} \quad (2)$$

tenzorszorzatot használják, ahol a $\mathbf{d}_{i,j} \in \mathbb{R}^3$ kontrollpontok kontrollhálót alkotnak, míg az $\{F_i : [a, b] \rightarrow \mathbb{R}\}_{i=0}^n$ és $\{G_j : [c, d] \rightarrow \mathbb{R}\}_{j=0}^m$ függvényrendszerek pedig olyan bázisok, melyek kielégítik a görbék esetén részletezett előnyös tervezési tulajdonságokat. Az F_i és G_j leképezések tetszőleges függvényterek bázisfüggvényei lehetnek, gyakorlatban viszont típusaik általában megegyeznek.

A tervezésben általában a NURBS függvényeket alkalmazzák a tenzorszorzattal előállítható felületek modellezésekor. Ilyenkor egy összetett alakzat több felületi foltból áll úgy, hogy a kapcsolódási paramétervonalak mentén egy kívánt folytonossági rend is teljesül.

1.3. Zárt görbék és felületek

A dolgozatban zárt görbék és felületek kerülnek tanulmányozásra. A magas folytonossági rendű, szinguláris pontokkal nem rendelkező zárt görbék és felületek szükségesek a mérnöki alkalmazá-

sokban, például az optikai tervezésben, a járművek reflektorainak, vagy más, precíz optikai komponensek megtervezésében is.

Ahhoz, hogy egy zárt görbét végpontbeli interpolációt teljesítő polinomiális görbeként írjunk le, a $\mathbf{d}_0 = \mathbf{d}_n$ egyenlőségnek kell fennállnia, valamint a $\mathbf{d}_0$ pontnak a görbén kell lennie. Ezek a feltételek viszont csak C^0 -osztályú folytonosságot biztosítanak a $\mathbf{d}_0$ pontban. Ugyanebben a pontban, tetszőleges k -adrendű folytonosság biztosításához, a görbe fokszámát n -ről $(n+k)$ -ra kell növelni, kontrollpontjait pedig az $\mathbf{a}_j = \mathbf{d}_{j \bmod (n+1)}$, $j = 0, 1, \dots, n+k$ alakban kell felvenni. A B-spline reprezentáció esetén a fokszámot nem kell növelni, de a kontrollpontokat ugyanezzel a módszerrel kell meghatározni. Zárt felületmodellezés esetén a folytonossági probléma – redundáns kontrollpontok felvétele mellett – még bonyolultabb geometriai feltételek biztosítását követeli meg.

1.4. Racionális görbék és felületek

Tekintsük az

$$R^{\mathbf{w}} = \{R_i^{\mathbf{w}} : [a, b] \rightarrow \mathbb{R}\}_{i=0}^n \quad (3)$$

$$R_i^{\mathbf{w}}(u) = R_i(u; \mathbf{w}) = \frac{w_i F_i(u)}{\sum_{j=0}^n w_j F_j(u)}, u \in [a, b]$$

függvényrendszert, ahol a $\{\mathbf{w}_i\}_{i=0}^n$ nemnegatív valós skalárok nem azonosan nullák. Ha az $\{F_j : [a, b] \rightarrow \mathbb{R}\}_{j=0}^n$ függvényrendszer normalizált és lineárisan független, akkor a (3) függvényrendszer szintén normalizált és ugyancsak bázist alkot. Ebben az esetben a (3) függvényrendszer a

$$\mathbf{g}^{\mathbf{w}}(u) = \mathbf{g}(u; \mathbf{w}) = \sum_{i=0}^n R_i(u) \mathbf{d}_i = \sum_{i=0}^n \frac{w_i F_i(u)}{\sum_{j=0}^n w_j F_j(u)} \mathbf{d}_i, u \in [a, b] \quad (4)$$

racionális görbe előállítására használhatjuk fel a $\mathbf{d}_i \in \mathbb{R}^\delta$ ($\delta \geq 2$) kontrollpontok segítségével. A w_i skalárokat súlyoknak nevezzük, amelyek pozitív arányosság erejéig egyértelműek, azaz – rögzített $[\mathbf{d}_i]_{i=0}^n$ kontrollpoligon és $\lambda > 0$ skalár mellett – a $\mathbf{w} = [w_i]_{i=0}^n$ és a $\lambda \mathbf{w} = [\lambda w_i]_{i=0}^n$ súlyvektorok ugyanazt a görbét származtatják. Észrevehető, hogyha az összes súly egyenlő, az $\{R_i^{\mathbf{w}}(u)\}_{i=0}^n$ racionális függvényrendszer az eredeti $\{F_i(u)\}_{i=0}^n$ bázisfüggvényekre redukálódik,

minek következtében az (1) előállítású görbék részhalmazát alkotják a (4) leírású racionális görbéknek.

A racionális Bernstein-polinomok és a nem egyenletes racionális B-spline bázisfüggvények által származtatott racionális Bézier, illetve NURBS görbék a legismertebb képviselői a (4) előállítású görbéknek, melyeket napjaink modellező rendszereiben (3D Studio Max, Maya, Blender) széles körben használnak.

Térbeli homogén koordinátákat használva, a (4) görbét a $\delta + 1$ dimenziós térben a

$$\mathbf{g}^z(u) = \sum_{i=0}^n F_i(u) \begin{bmatrix} w_i \mathbf{d}_i \\ w_i \end{bmatrix}, \quad u \in [a, b]$$

görbének (másképpen ösképnek) az origóból a $z = 1$ egyenletű hipersíkra (vagy képsíkra) eső centrális projekciójaként is felfoghatjuk, amennyiben z az utolsó koordinátát jelöli. A $\mathbf{g}^w$ görbe rendelkezik a $\mathbf{g}^z$ görbe azon tulajdonságaival, amik invariánsak a centrális projekcióra nézve. Ilyen például a folytonosság, illeszkedés, kollinearitás és a hullámszcsoökkentés megőrzése.

Az eredeti $\{F_i(u)\}_{i=0}^n$ bázissal definiált görbék alakja csak kontrollpontjainak affin transzformációjára nézve volt invariáns. Ezzel szemben a (4) görbe alakja a kontrollpoligon projektív transzformációjára nézve is invariáns marad. Egy ilyen típusú transzformációt az öskép terében kell végrehajtani, melynek folytán nemcsak a kontrollpontok, hanem a hozzájuk rendelt súlyok is megváltoznak.

A nemnegatív és nem azonosan nulla komponensű $\mathbf{w}^u = [w_i^u]_{i=0}^n$ és $\mathbf{w}^v = [w_j^v]_{j=0}^m$ súlyvektorokat, valamint az $\{F_i(u) : u \in [a, b]\}_{i=0}^n$ és $\{G_j(v) : v \in [c, d]\}_{j=0}^m$ normalizált bázisokat felhasználva, a

$$[\mathbf{d}_{i,j}]_{i=0,j=0}^{n,m} \in \mathcal{M}_{n+1,m+1}(\mathbb{R}^3)$$

kontrollháló az

$$\begin{aligned} \mathbf{s}^{\mathbf{w}^u, \mathbf{w}^v}(u, v) &= \sum_{i=0}^n \sum_{j=0}^m \mathbf{d}_{i,j} R_i^{\mathbf{w}^u}(u) Q_j^{\mathbf{w}^v}(v) \\ &= \sum_{i=0}^n \sum_{j=0}^m \mathbf{d}_{i,j} \frac{w_i^u F_i(u)}{\sum_{k=0}^n w_k^u F_k(u)} \frac{w_j^v G_j(v)}{\sum_{l=0}^m w_l^v G_l(v)} \\ &= \frac{1}{\sum_{k=0}^n w_k^u F_k(u)} \frac{1}{\sum_{l=0}^m w_l^v G_l(v)} \sum_{i=0}^n \sum_{j=0}^m \mathbf{d}_{i,j} (w_i^u w_j^v) F_i(u) G_j(v) \end{aligned}$$

racionális felületet származtatja, ahol

$$\left\{ \begin{array}{l} R_i^{\mathbf{w}^u}(u) = R_i(u; \mathbf{w}^u) = \frac{w_i^u F_i(u)}{\sum_{k=0}^n w_k^u F_k(u)}, \quad u \in [a, b], \quad i = 0, 1, \dots, n, \\ Q_j^{\mathbf{w}^v}(v) = Q_j(v; \mathbf{w}^v) = \frac{w_j^v G_j(v)}{\sum_{l=0}^m w_l^v G_l(v)}, \quad v \in [c, d], \quad j = 0, 1, \dots, m \end{array} \right. \quad (5)$$

az u - és v -irányú racionális bázisfüggvényeket jelölik.

Ha a

$$[\mathbf{d}_{i,j}]_{i=0,j=0}^{n,m} \in \mathcal{M}_{n+1,m+1}(\mathbb{R}^3)$$

kontrollhálózhoz a nemnegatív és nem azonosan nulla elemű

$$W = [w_{i,j}]_{i=0,j=0}^{n,m}$$

súlymátrixot társítjuk, akkor a sokkal általánosabb

$$\mathbf{s}^W(u, v) = \sum_{i=0}^n \sum_{j=0}^m \mathbf{d}_{i,j} \frac{w_{i,j} F_i(u) G_j(v)}{\sum_{k=0}^n \sum_{l=0}^m w_{k,l} F_k(u) G_l(v)}$$

racionális felületet írhatjuk le.

2. Zárt racionális trigonometrikus görbék

Ebben a fejezetben ismertetjük a (racionális) ciklikus bázist valamint az általa származtatott úgynevezett (racionális) ciklikus görbét, amikkel egzaktul trigonometrikus görbék állíthatók elő.

2.1. Ciklikus bázis

Az [1] cikk egy olyan modellezési módszert vezet be, amely a

$$\mathcal{V}_n = \langle 1, \cos(u), \sin(u), \dots, \cos(nu), \sin(nu) : u \in [0, 2\pi] \rangle$$

véges, legfeljebb n -edfokú ($n \geq 1$) trigonometrikus polinomok vektorterének a

$$C = \{C_{i,2n}(u) : u \in [0, 2\pi]\}_{i=0}^{2n} = \left\{ \frac{c_n}{2^n} (1 + \cos(u + i\lambda_n))^n : u \in [0, 2\pi] \right\}_{i=0}^{2n} \quad (6)$$

ciklikus és normalizált bázisát használja zárt görbék és felületek előállításához, ahol a

$$c_n = \frac{2^{2n}}{(2n+1) \binom{2n}{n}}$$

konstans kielégíti a

$$\begin{cases} c_1 = \frac{2}{3}, \\ c_n = \frac{2n}{2n+1} c_{n-1}, n \geq 2 \end{cases} \quad (7)$$

rekurziót, míg $\lambda_n = \frac{2\pi}{2n+1}$ egy rögzített fáziseltolás.

Észrevehető, hogy a (6) bázis 2π -periódusú függvényekből áll, tehát a bázis tulajdonságait bármely 2π -hosszúságú intervallumon tanulmányozhatjuk.

A (6) bázisfüggvényt használva, úgynevezett ciklikus görbéket és felületeket állíthatunk elő, melyeket az alábbi pontokban értelmezünk.

2.2. Ciklikus görbék

2.1. Értelmezés (Ciklikus görbék, [1]). A (6) bázisfüggvények és a

$$[\mathbf{d}_0, \mathbf{d}_1, \dots, \mathbf{d}_{2n}] = [\mathbf{d}_i]_{i=0}^{2n} \in \mathcal{M}_{1,2n+1}(\mathbb{R}^\delta), \delta \geq 2$$

kontrollpoligon által származtatott

$$\mathbf{a}_n(u) = \sum_{i=0}^{2n} C_{i,2n}(u) \mathbf{d}_i, u \in [0, 2\pi] \quad (8)$$

görbét n -edfokú ($n \geq 1$) ciklikus görbének nevezzük.

A ciklikus görbék a következő előnyös tulajdonságokkal rendelkeznek:

- **ciklikus szimmetria** (a görbe alakja nem változik, ha a kontrollpontjait ciklikusan permutáljuk);
- **szingularitás nélküli paraméterezés** (a görbe minden reguláris pontban C^∞ -osztályú, míg a szinguláris pontokban a jobb és bal oldali deriváltak léteznek és szintén nem válnak nullvektorrá);
- **affin transzformációkkal szembeni zártság** (a görbe alakja invariáns a kontrollpoligonjának affin transzformációira – például forgatásra, transzlációra, tükrözésre, skálázásra, nyírásra – nézve);
- **konvex burok tulajdonság** (a görbe alakja a kontrollpontjainak konvex burkában húzódik);
- **pszeudo-lokális változtathatóság** (az $\mathbf{a}_n(\pi - i\lambda_n)$ görbepont kivételével a $\mathbf{d}_i$ kontrollpont változása kihat az egész görbe alakjára, azaz a görbe globálisan változtatható, viszont az n foksám növelése hatásosan csökkenti a kontrollpontoknak a görbe alakjára kifejtett hatását, egyre finomabb alakváltoztatási lehetőségeket biztosítva);
- **hullámszácsökkentő tulajdonság** (ez a tulajdonság akkor teljesül, ha egyetlen hipersík sem metszi a görbét több pontban, mint annak kontrollpoligonját; ebben az esetben a görbe megőrzi síkbeli kontrollpoligonok a konvexitását is, azaz ha egy síkbeli ciklikus görbe kontrollpoligonja konvex, akkor a görbe egy konvex síktartomány határgörbéjét alkotja);
- **fokszámnövelés** (más módszerekkel szemben – rekurzív képletek helyett – a ciklikus görbe n fokszámát kontrollpont-alapú egzakt képletekkel növelhetjük $(n + r)$ -re, mely

során olyan kontrollpoligon-sorozatot kapunk, mely az eredeti, fokszámnövelt ciklikus görbe alakjához konvergál, mikor $r \rightarrow \infty$).

A (8) ciklikus görbe egyszerű trigonometrikus azonosságok segítségével az

$$\mathbf{a}_n = \frac{1}{2n+1} \sum_{i=0}^{2n} \mathbf{d}_i + \frac{2}{(2n+1) \binom{2n}{n}} \sum_{i=0}^{2n} \sum_{k=0}^{n-1} \binom{2n}{k} \cos \left((n-k)u + \frac{2(n-k)i\pi}{2n+1} \right) \mathbf{d}_i$$

alakra hozható, ahol $\frac{1}{2n+1} \sum_{i=0}^{2n} \mathbf{d}_i$ a kontrollpoligon súlypontja. Innen könnyen kifejezhető a ciklikus görbe

$$\frac{d}{du} \mathbf{a}_n(u) = \frac{2}{(2n+1) \binom{2n}{n}} \sum_{i=0}^{2n} \sum_{k=0}^{n-1} \binom{2n}{k} (n-k) \sin \left((n-k)u + \frac{2(n-k)i\pi}{2n+1} \right) \mathbf{d}_i \quad (9)$$

hodográfja. A görbe r -edrendű ($r \geq 1$) deriváltját pedig a

$$\frac{d^r}{du^r} \mathbf{a}_n(u) = \frac{2}{(2n+1) \binom{2n}{n}} \sum_{i=0}^{2n} \sum_{k=0}^{n-1} \binom{2n}{k} (n-k)^r \cos \left((n-k)u + \frac{2(n-k)i\pi}{2n+1} + \frac{r\pi}{2} \right) \mathbf{d}_i \quad (10)$$

kifejezés adja.

2.3. Trigonometrikus görbék

Tekintsük azokat az

$$\begin{cases} \mathbf{x} : [0, 2\pi] \rightarrow \mathbb{R}^\delta, \delta \geq 2 \\ \mathbf{x}(u) = \begin{bmatrix} x_1(u) & x_2(u) & \cdots & x_\delta(u) \end{bmatrix}^T \end{cases} \quad (11)$$

zárt görbék, melyek koordinátafüggvényeit az

$$\begin{aligned} x_l(u) &= x_l \left(u, \{ \alpha_p^l, \psi_p^l \}_{p \in P_l}, \{ \beta_q^l, \varphi_q^l \}_{q \in Q_l} \right) \\ &= \sum_{p \in P_l} \alpha_p^l \cos(pu + \psi_p^l) + \sum_{q \in Q_l} \beta_q^l \sin(qu + \varphi_q^l) \end{aligned} \quad (12)$$

véges fokszámú trigonometrikus polinomok adják, ahol $l = 1, 2, \dots, \delta$, míg $P_l, Q_l \subset \mathbb{N}$ és $\alpha_p^l, \beta_q^l, \psi_p^l, \varphi_q^l \in \mathbb{R}$. Ebben a görbeosztályba tartoznak az ellipszisek, epicikloisok, hipocikloisok, tóruszcsomók, általánosabban pedig minden olyan zárt trigonometrikus görbe, mely a $\mathcal{V}_n$ függvénytérben előállítható. A [2] cikk olyan kontrollpont-konfigurációkat biztosít ciklikus görbék számára, melyek segítségével egzaktul írhatunk le tetszőleges (11) előállítási parametrikus zárt görbék. Az említett cikk eredményei a következő lemmán alapszanak.

2.1. Lemma (A (11) görbék alaptulajdonsága, [2]). Tekintsük a (11) zárt görbét, továbbá az

$$n \geq n_{\min} = \max \{e | e \in \cup_{l=1}^{\delta} (P_l \cup Q_l)\}$$

fokszám, valamint a

$$\mathbf{d}_i = \begin{bmatrix} x_1(-i\lambda_n) & x_2(-i\lambda_n) & \cdots & x_{\delta}(-i\lambda_n) \end{bmatrix}^T, \quad i = 0, 1, \dots, 2n \quad (13)$$

kontrollpontok és a (6) bázis által meghatározott

$$\mathbf{a}_n(u) = \sum_{i=0}^{2n} C_{i,2n}(u) \mathbf{d}_i = \begin{bmatrix} a_1(u) & a_2(u) & \cdots & a_{\delta}(u) \end{bmatrix}^T, \quad u \in [0, 2\pi]$$

ciklikus görbét. Ebben az esetben az $\mathbf{a}_n$ görbe l -edik ($l = 1, 2, \dots, \delta$) koordinátafüggvénye

$$a_l = \frac{1}{\binom{2n}{n}} \left(\sum_{p \in P_l} \alpha_p^l \binom{2n}{n-p} \cos(pu + \psi_p^l) + \sum_{q \in Q_l} \beta_q^l \binom{2n}{n-q} \sin(qu + \varphi_q^l) \right) \quad (14)$$

alakú.

A fordított állítás is igaz, vagyis ha egy ciklikus görbe koordinátafüggvényei (14) alakúak, akkor annak kontrollpontjait éppen a (13) képlet adja.

A lemmából következik, hogy bármely paraméteres görbe fáziseltolás erejéig egyértelmű ciklikus reprezentációval rendelkezik.

2.1. Tétel (A (11) alakú görbék egzakt leírása, [2]). Tekintsük az

$$\tilde{\mathbf{x}}(u) = [\tilde{x}_1(u) \quad \tilde{x}_2(u) \quad \dots \quad \tilde{x}_{\delta}(u)]^T, \quad \delta \geq 2, u \in [0, 2\pi],$$

zárt görbét az

$$\tilde{x}_l(u) = \sum_{p \in P_l} \tilde{\alpha}_p^l \cos(pu + \tilde{\psi}_p^l) + \sum_{q \in Q_l} \tilde{\beta}_q^l \sin(qu + \tilde{\varphi}_q^l), \quad l \in 1, 2, \dots, \delta, \quad (15)$$

koordinátafüggvényekkel, ahol $P_l, Q_l \subset \mathbb{N}, \tilde{\alpha}_p^l, \tilde{\beta}_q^l, \tilde{\psi}_p^l, \tilde{\varphi}_q^l \in \mathbb{R}$. Legyen továbbá

$$n \geq n_{\min} = \max \{e | e \in \cup_{l=1}^{\delta} (P_l \cup Q_l)\},$$

tetszőleges fokszám, illetve $r \in \mathbb{N}$ deriválási rend. Hogyha a [2.1.](#) lemmabeli konstansokat az

$$\begin{aligned}\alpha_p^l(n) &= \frac{\binom{2n}{n}}{\binom{2n}{n-p}} p^r \tilde{\alpha}_p^l, \\ \psi_p^l(n) &\equiv \tilde{\psi}_p^l + \frac{r\pi}{2}, p \in P_l, \\ \beta_q^l(n) &= \frac{\binom{2n}{n}}{\binom{2n}{n-q}} q^r \tilde{\beta}_q^l, \\ \varphi_q^l(n) &\equiv \tilde{\varphi}_q^l + \frac{r\pi}{2}, q \in Q_l,\end{aligned}$$

módon vesszük fel, akkor bármely $n \geq n_{\min}$ esetén a [\(13\)](#) kontrollpontok által generált ciklikus görbe az $\tilde{\mathbf{x}}$ görbe r -edrendű deriváltjával egyezik meg, azaz az $\mathbf{a}_n$ görbe l -edik ($l = 1, 2, \dots, \delta$) koordinátafüggvénye kielégíti az

$$\begin{aligned}a_l(u) &= \frac{d^r}{du^r} \tilde{x}(u) \\ &= \sum_{p \in P_l} p^r \tilde{\alpha}_p^l \cos\left(pu + \tilde{\psi}_p^l + \frac{r\pi}{2}\right) + \sum_{q \in Q_l} q^r \tilde{\beta}_q^l \sin\left(qu + \tilde{\varphi}_q^l + \frac{r\pi}{2}\right)\end{aligned}$$

egyenlőséget minden $u \in [0, 2\pi]$ pontra.

2.4. Racionális trigonometrikus görbék modellezése

2.2. Értelmezés (Racionális trigonometrikus bázis, [\[3\]](#)). A $\mathbf{w} = [w_i]_{i=0}^{2n}$ nemnegatív súlyvektor és a [\(6\)](#) ciklikus bázisfüggvény az

$$R^{\mathbf{w}} = \{R_{i,2n}^{\mathbf{w}} : [0, 2\pi] \rightarrow \mathbb{R}\}_{i=0}^{2n},$$

racionális bázist generálja, ahol

$$R_{i,2n}^{\mathbf{w}}(u) = R_{i,2n}(u; \mathbf{w}) = \frac{w_i C_{i,2n}(u)}{\sum_{j=0}^{2n} w_j C_{j,2n}(u)}, \quad u \in [0, 2\pi]. \quad (16)$$

Ezeket a racionális bázisfüggvényeket használva, a ciklikus görbék racionális változatát értelmezhetjük.

2.3. Értelmezés (Zárt racionális trigonometrikus görbék, [\[3\]](#)). Tekintsük a

$$[\mathbf{d}_i]_{i=0}^{2n} \in \mathcal{M}_{1,2n+1}(\mathbb{R}^\delta), \delta \geq 2$$

kontrollpoligont és a $\mathbf{w} = [w_i]_{i=0}^{2n}$ nemnegatív súlyvektort. A

$$\left\{ \begin{array}{l} \mathbf{g}_n^{\mathbf{w}} : [0, 2\pi] \rightarrow \mathbb{R}^\delta \\ \mathbf{g}_n^{\mathbf{w}}(u) = \mathbf{g}_n^{\mathbf{w}}(u, w) = \sum_{i=0}^{2n} \mathbf{d}_i R_{i,2n}(u; \mathbf{w}) = \sum_{i=0}^{2n} \mathbf{d}_i \frac{w_i C_{i,2n}(u)}{\sum_{j=0}^{2n} w_j C_{j,2n}(u)} \end{array} \right. \quad (17)$$

görbét n -edfokú ($n \geq 1$) zárt racionális trigonometrikus görbének nevezzük.

A ciklikus szimmetriát leszámítva, a (17) görbe öröklí a ciklikus görbék minden előnyös tulajdonságát. Amennyiben a kontrollpontokkal egyszerre a súlyvektor elemeit is ciklikusan permutáljuk, akkor ez a tulajdonság is életben marad.

3. Zárt racionális trigonometrikus felületek

Ebben a fejezetben ismertetjük a (racionális) ciklikus bázis által származtatott úgynevezett (racionális) ciklikus felületeket, amikkel egzaktul trigonometrikus felületek állíthatóak elő.

3.1. Ciklikus felületek

3.1. Értelmezés (Ciklikus felületek, [1]). A (6) képlettel definiált

$$\{C_{i,2n}(u) : u \in [0, 2\pi]\}_{i=0}^{2n}$$

és

$$\{C_{j,2m}(v) : v \in [0, 2\pi]\}_{j=0}^{2m}$$

bázisfüggvények, továbbá a

$$\mathbf{D}_{n,m} = [\mathbf{d}_{i,j}]_{i=0,j=0}^{2n,2m} \in \mathcal{M}_{2n+1,2m+1}(\mathbb{R}^3)$$

kontrollháló által meghatározott

$$\mathbf{s}_{n,m}(u, v) = \sum_{i=0}^{2n} \sum_{j=0}^{2m} \mathbf{d}_{i,j} C_{i,2n}(u) C_{j,2m}(v), \quad (u, v) \in [0, 2\pi] \times [0, 2\pi] \quad (18)$$

tenzorszorzatot (n, m) -edfokú $(n \geq 1, m \geq 1)$ ciklikus felületnek nevezzük.

A hullámzáscsökkentő tulajdonságon kívül a ciklikus felületek öröklik a ciklikus görbék összes előnyös tulajdonságát. Az [1] cikkben igazolva van az is, hogy a ciklikus felületek zártak a deriválásra nézve, vagyis a (18) ciklikus felület $(r_1 + r_2)$ -edrendű $(r_1 \geq 1, r_2 \geq 1)$ vegyes parciális deriváltjai felírhatóak a

$$\mathbf{D}_{p,q} = \left[\mathbf{y} \left(-\frac{2w\pi}{2p+1}, -\frac{2z\pi}{2q+1} \right) \right]_{w=0, z=0}^{2p, 2q}$$

kontrollhálózattal rendelkező $\mathbf{y} : [0, 2\pi] \times [0, 2\pi] \rightarrow \mathbb{R}^3$ ciklikus görbeként, ahol

$$\begin{aligned} \mathbf{y}(u, v) &= \frac{4 \binom{2p}{p} \binom{2q}{q}}{(2n+1)(2m+1) \binom{2n}{n} \binom{2m}{m}} \\ &\cdot \sum_{j=0}^{2m} \left(\sum_{l=0}^{m-1} \frac{\binom{2m}{l}}{\binom{2q}{q-m+l}} (m-l)^{r_2} \cos \left((m-l)v + \frac{2(m-l)j\pi}{2m+1} + \frac{r_2\pi}{2} \right) \right) \\ &\cdot \left(\sum_{i=0}^{2n} \sum_{k=0}^{n-1} \frac{\binom{2n}{k}}{\binom{2p}{p-n+k}} (n-k)^{r_1} \cos \left((n-k)u + \frac{2(n-k)i\pi}{2n+1} + \frac{r_1\pi}{2} \right) \mathbf{d}_{i,j} \right). \end{aligned} \quad (19)$$

3.2. Trigonometrikus felületek

Tekintsük azokat az

$$\mathbf{y} : [0, 2\pi]^\delta \rightarrow \mathbb{R}^{\delta+k}, \delta \geq 2, k \geq 1 \quad (20)$$

$$\mathbf{y}(\mathbf{u}) = \begin{bmatrix} y_1(\mathbf{u}) & y_2(\mathbf{u}) & \cdots & y_{\delta+k}(\mathbf{u}) \end{bmatrix}^T, \mathbf{u} = (u_1, u_2, \dots, u_\delta),$$

zárt felületeket, melyek koordinátafüggvényei

$$\begin{aligned} y_l(\mathbf{u}) &= y_l \left(\left\{ u_j, \{ \alpha_p^{l,w,j}, \psi_p^{l,w,j} \}_{p \in P_{l,w,j}}, \{ \beta_q^{l,w,j}, \varphi_q^{l,w,j} \}_{q \in Q_{l,w,j}} \right\}_{w=1, j=1}^{m_l, \delta} \right) \\ &= \sum_{w=1}^{m_l} \prod_{j=1}^{\delta} \left(\sum_{p \in P_{l,w,j}} \alpha_p^{l,w,j} \cos(pu_j + \psi_p^{l,w,j}) + \sum_{q \in Q_{l,w,j}} \beta_q^{l,w,j} \sin(qu_j + \varphi_q^{l,w,j}) \right) \end{aligned}$$

alakúak, ahol $l = 1, 2, \dots, \delta + k$,

$$\mathbf{u} = \begin{bmatrix} u_1 & u_2 & \cdots & u_\delta \end{bmatrix} \in [0, 2\pi]^\delta,$$

míg $P_{l,w,j}, Q_{l,w,j} \subset \mathbb{N}, m_l \in \mathbb{N} \setminus \{0\}$ és $\alpha_p^{l,w,j}, \beta_q^{l,w,j}, \psi_p^{l,w,j}, \varphi_q^{l,w,j} \in \mathbb{R}$. Ilyen felületek például a (11) trigonometrikus görbék adott tengely körüli forgatásából nyert forgásfelületek, vagy a projektív geometriából ismert egyoldalú Steiner-féle felület, vagy bármilyen más $-u_1, u_2, \dots, u_\delta$ változókban szétválasztható – véges fokszerű trigonometrikus polinomok szorzataként leírt koordinátafüggvényű felület.

A [2] cikkben a szerzők a (20) előállítású felületek (fáziseltolásig egyértelmű) ciklikus reprezentációját írják le. Ezzel kapcsolatosak az alábbi tulajdonságok.

3.1. Lemma ((20) felületek alaptulajdonsága, [2]). *Tekintsük a (20) felületet, az*

$$n_j \geq n_{\min}^j = \max \{e_j | e_j \in \cup_{l=1}^{\delta+k} \cup_{w=1}^{m_l} (P_{l,w,j} \cup Q_{l,w,j})\}, j = 1, 2, \dots, \delta$$

fokszámokat, továbbá a

$$\begin{aligned} [\mathbf{d}_{i_1, i_2, \dots, i_\delta}]_{i_1=0, i_2=0, \dots, i_\delta=0}^{2n_1, 2n_2, \dots, 2n_\delta} &\in \mathcal{M}_{2n_1+1, 2n_2+1, \dots, 2n_\delta+1}(\mathbb{R}^{\delta+k}) \\ \mathbf{d}_{i_1, i_2, \dots, i_\delta} &= \begin{bmatrix} y_1(\mathbf{u}_{i_1, i_2, \dots, i_\delta}^*) & y_2(\mathbf{u}_{i_1, i_2, \dots, i_\delta}^*) & \cdots & y_{\delta+k}(\mathbf{u}_{i_1, i_2, \dots, i_\delta}^*) \end{bmatrix}^T \end{aligned} \quad (21)$$

kontrollhálót, ahol $\mathbf{u}_{i_1, i_2, \dots, i_\delta}^* = \begin{pmatrix} -i_1 \lambda_{n_1} & -i_2 \lambda_{n_2} & \dots & -i_\delta \lambda_{n_\delta} \end{pmatrix} \in \mathbb{R}^\delta$, és az

$$\begin{aligned} \mathbf{s}_{n_1, n_2, \dots, n_\delta}(\mathbf{u}) &= \sum_{i_1=0}^{2n_1} \sum_{i_2=0}^{2n_2} \dots \sum_{i_\delta=0}^{2n_\delta} C_{i_1, 2n_1}(u_1) C_{i_2, 2n_2}(u_2) \dots C_{i_\delta, 2n_\delta}(u_\delta) \mathbf{d}_{i_1, i_2, \dots, i_\delta} \\ &= \left(\prod_{j=1}^{\delta} \frac{C_{n_j}}{2^{n_j}} \right) \sum_{i_1=0}^{2n_1} \sum_{i_2=0}^{2n_2} \dots \sum_{i_\delta=0}^{2n_\delta} (1 + \cos(u_1 + i_1 \lambda_{n_1}))^{n_1} \\ &\quad \cdot (1 + \cos(u_2 + i_2 \lambda_{n_2}))^{n_2} \cdot \dots \cdot (1 + \cos(u_\delta + i_\delta \lambda_{n_\delta}))^{n_\delta} \\ &= \begin{bmatrix} s_1(\mathbf{u}) & s_2(\mathbf{u}) & \dots & s_{\delta+k}(\mathbf{u}) \end{bmatrix}^T, \\ (u_1, u_2, \dots, u_\delta) &= \mathbf{u} \in [0, 2\pi]^\delta \end{aligned}$$

$(n_1, n_2, \dots, n_\delta)$ -edfokú tenzorszorzatot. Ebben az esetben az $\mathbf{s}_{n_1, n_2, \dots, n_\delta}$ felület l -edik $(l = 1, 2, \dots, \delta + k)$ koordinátafüggvénye

$$\begin{aligned} s_l(\mathbf{u}) &= \sum_{w=1}^{m_l} \prod_{j=1}^{\delta} \frac{1}{\binom{2n_j}{n_j}} \left(\sum_{p \in P_{l,w,j}} \alpha_p^{l,w,j} \binom{2n_j}{n_j-p} \cos(pu_j + \psi_p^{l,w,j}) \right. \\ &\quad \left. + \sum_{q \in Q_{l,w,j}} \beta_q^{l,w,j} \binom{2n_j}{n_j-q} \sin(qu_j + \varphi_q^{l,w,j}) \right) \end{aligned} \quad (22)$$

alakú.

A fordított állítás is igaz, vagyis ha egy ciklikus felület koordinátafüggvényei (22) alakúak, akkor kontrollpontjait éppen a (21) képlet adja.

3.1. Tétel (A (20) felületek egzakt leírása, [2]). Legyen sajátosan $\delta = 2$, és tekintsük az

$$\tilde{\mathbf{y}}(u_1, u_2) = \begin{bmatrix} \tilde{y}_1(u_1, u_2) & \tilde{y}_2(u_1, u_2) & \tilde{y}_3(u_1, u_2) \end{bmatrix}^T, \quad (u_1, u_2) \in [0, 2\pi] \times [0, 2\pi], \quad (23)$$

zárt felületet az

$$\tilde{y}_l(u_1, u_1) = \sum_{w=1}^{m_l} \prod_{j=1}^2 \left(\sum_{p \in P_{l,w,j}} \tilde{\alpha}_p^{l,w,j} \cos(pu_j + \tilde{\psi}_p^{l,w,j}) + \sum_{q \in Q_{l,w,j}} \tilde{\beta}_q^{l,w,j} \sin(qu_j + \tilde{\varphi}_q^{l,w,j}) \right)$$

koordinátafüggvényekkel $(l = 1, 2, 3)$, ahol $P_{l,w,j}, Q_{l,w,j} \subset \mathbb{N}$, $m_l \in \mathbb{N} \setminus \{0\}$, $\tilde{\alpha}_p^{l,w,j}, \tilde{\beta}_q^{l,w,j}, \tilde{\psi}_p^{l,w,j}, \tilde{\varphi}_q^{l,w,j} \in \mathbb{R}$. Legyenek továbbá

$$\begin{aligned} n_1 &\geq n_{\min}^1 = \max \{e_1 | e_1 \in \cup_{l=1}^3 \cup_{w=1}^3 (P_{l,w,1} \cup Q_{l,w,1})\}, \\ n_2 &\geq n_{\min}^2 = \max \{e_2 | e_2 \in \cup_{l=1}^3 \cup_{w=1}^3 (P_{l,w,2} \cup Q_{l,w,2})\}, \end{aligned}$$

és $r_1, r_2 \in \mathbb{N}$. *Hogyha a 3.1 lemmabeli konstansokat az*

$$\begin{aligned}\alpha_p^{l,w,j}(n_j) &= \frac{\binom{2n_j}{n_j}}{\binom{2n_j}{n_j-p}} p^{r_j} \tilde{\alpha}_p^{l,w,j}, \\ \psi_p^{l,w,j}(n_j) &\equiv \tilde{\psi}_p^{l,w,j} + \frac{r_j \pi}{2}, \quad p \in P_{l,w,j}, \\ \beta_q^{l,w,j}(n_j) &= \frac{\binom{2n_j}{n_j}}{\binom{2n_j}{n_j-q}} q^{r_j} \tilde{\beta}_q^{l,w,j}, \\ \varphi_q^{l,w,j}(n_j) &\equiv \tilde{\varphi}_q^{l,w,j} + \frac{r_j \pi}{2}, \quad q \in Q_{l,w,j},\end{aligned}$$

alakban választjuk meg minden $j = 1, 2$ és $w = 1, 2, \dots, m_l$ index esetén, akkor bármely $n_1 \geq n_{\min}^1$ és $n_2 \geq n_{\min}^2$ fokszám esetén a

$$\mathbf{d}_{i_1, i_2} = \begin{bmatrix} y_1(-i_1 \lambda_{n_1}, -i_2 \lambda_{n_2}) & y_2(-i_1 \lambda_{n_1}, -i_2 \lambda_{n_2}) & y_3(-i_1 \lambda_{n_1}, -i_2 \lambda_{n_2}) \end{bmatrix}^T$$

kontrollpontok által generált ciklikus felület pontosan az $\tilde{\mathbf{y}}$ felület $(r_1 + r_2)$ -edrendű parciális deriváltjával egyezik meg, azaz az $\mathbf{s}_{n_1, n_2}$ felület l -edik koordinátafüggvénye kielégíti az

$$s_l(u_1, u_1) = \frac{\partial^{r_1+r_2}}{\partial u_1^{r_1} \partial u_1^{r_1}} \sum_{w=1}^{m_l} \prod_{j=1}^2 \left(\sum_{p \in P_{l,w,j}} \tilde{\alpha}_p^{l,w,j} \cos(pu_j + \tilde{\psi}_p^{l,w,j}) + \sum_{q \in Q_{l,w,j}} \tilde{\beta}_q^{l,w,j} \sin(qu_j + \tilde{\varphi}_q^{l,w,j}) \right)$$

azonosságot minden $(u_1, u_2) \in [0, 2\pi] \times [0, 2\pi]$ pontra.

3.3. Racionális trigonometrikus felületek modellezése

A (18) zárt ciklikus felületek racionális kiterjesztésével egy újabb tervezési eszközhöz jutunk zárt racionális trigonometrikus felületek modellezésére.

3.2. Értelmezés (Zárt racionális trigonometrikus felület, [3]). *Tekintsük a nemnegatív és nem azonosan nulla komponensű $\mathbf{w}^u = [w_i^u]_{i=0}^{2n}$ és $\mathbf{w}^v = [w_j^v]_{j=0}^{2m}$ súlyvektorokat, a*

$$\{C_{i,2n}(u) : u \in [0, 2\pi]\}_{i=0}^{2n}$$

és

$$\{C_{j,2m}(v) : v \in [0, 2\pi]\}_{j=0}^{2m}$$

ciklikus bázisokat és a

$$[\mathbf{d}_{i,j}]_{i=0,j=0}^{2n,2m} \in \mathcal{M}_{2n+1,2m+1}(\mathbb{R}^3)$$

kontrollhálót. Az

$$\begin{aligned}
\mathbf{s}^{\mathbf{w}^u, \mathbf{w}^v}(u, v) &= \sum_{i=0}^{2n} \sum_{j=0}^{2m} \mathbf{d}_{i,j} R_{i,2n}^{\mathbf{w}^u}(u) Q_{j,2m}^{\mathbf{w}^v}(v) \\
&= \sum_{i=0}^{2n} \sum_{j=0}^{2m} \mathbf{d}_{i,j} \frac{w_i^u C_{i,2n}(u)}{\sum_{k=0}^{2n} w_k^u C_{k,2n}(u)} \frac{w_j^v C_{j,2m}(v)}{\sum_{l=0}^{2m} w_l^v C_{l,2m}(v)} \\
&= \frac{1}{\sum_{k=0}^{2n} w_k^u C_{k,2n}(u)} \frac{1}{\sum_{l=0}^{2m} w_l^v C_{l,2m}(v)} \sum_{i=0}^{2n} \sum_{j=0}^{2m} \mathbf{d}_{i,j} (w_i^u w_j^v) C_{i,2n}(u) C_{j,2m}(v)
\end{aligned}$$

leképezést (n, m) -edfokú racionális trigonometrikus felületnek nevezzük, ahol $\{R_{i,2n}^{\mathbf{w}^u}(u)\}_{i=0}^{2n}$ és $\{Q_{j,2m}^{\mathbf{w}^v}(v)\}_{j=0}^{2m}$ az u - és v -irányú (16) alakú racionális trigonometrikus bázisfüggvényeket jelölik.

Ha a

$$[\mathbf{d}_{i,j}]_{i=0,j=0}^{2n,2m} \in \mathcal{M}_{2n+1,2m+1}(\mathbb{R}^3)$$

kontrollháléhoz a nemnegatív és nem azonosan nulla elemű

$$W = [w_{i,j}]_{i=0,j=0}^{2n,2m}$$

súlymátrixot társítjuk, akkor a sokkal rugalmasabb tervezési eszközként használható

$$\mathbf{s}^W(u, v) = \sum_{i=0}^{2n} \sum_{j=0}^{2m} \mathbf{d}_{i,j} \frac{w_{i,j} C_{i,2n}(u) C_{j,2m}(v)}{\sum_{k=0}^{2n} \sum_{l=0}^{2m} w_{k,l} C_{k,2n}(u) C_{l,2m}(v)} \quad (24)$$

racionális felületet írhatjuk le.

4. Zárt racionális trigonometrikus görbék és felületek interaktív modellezése

A dolgozat szerzőjének célja egy olyan statikus függvénykönyvtár (*cagd32*) kifejlesztése, amely eszközöket biztosít tetszőleges, véges fokszámú, zárt (racionális) trigonometrikus görbék és azok tenzor szorzataként előállított felületek modellezéséhez kontrollpont-alapú műveletekkel. Emellett egy olyan alkalmazás megvalósítása is a cél, ami szemlélteti a függvénykönyvtár felhasználhatóságát és hatékonyságát.

Ebben a fejezetben a fejlesztés során felhasznált függvénykönyvtárakat, az alkalmazás funkcionális követelményeit, a görbék és felületek kiértékelésében és megjelenítésében szerepet játszó fontosabb osztályok rövid leírását, valamint az előző elméleti fejezetekre épülő numerikus eredményeket (tesztpéldákat) ismertetünk.

4.1. Követelmények

A szoftverfejlesztés első lépése a követelménytervezés, melynek célja, hogy meghatározzuk, milyen szolgáltatásokat várunk el a rendszertől és annak milyen megszorítások mellett kell működnie.

4.1.1. Felhasználói követelmények

A felhasználói követelmények szerint az alkalmazás:

- platformfüggetlenül fordítható;
- intuitív, könnyen tanulható felhasználói felülettel rendelkezik;
- képes ugyanazt a háromdimenziós színteret egyszerre több nézőpontból megjeleníteni, lehetőséget ad nézőpontváltásra, és a kamera interaktív mozgatására;
- nézőpontjai szükség szerint átméretezhetőek és kikapcsolhatóak;
- irányított, pont- és reflektorszerű fényforrásokat tud megjeleníteni, valamint lehetőséget ad ezek ki- és bekapcsolására, interaktív beállítására és transzformációjára;

- textúrákat tud megjeleníteni az éppen ábrázolt felületeken csúcspont és részecske árnyalók alkalmazásával;
- képes (8), (18), (17) és (24) előállítású görbéket és felületeket dinamikusan létrehozni, azok magasabb rendű (parciális) deriváltjait kiértékelni, megjeleníteni és interaktívan módosítani kontrollpont-alapú műveletekkel;
- kontrollpont konfigurációkat biztosít (11) és (23) előállítású (racionális) trigonometrikus görbék és felületek (racionális) ciklikus reprezentációjához;
- jelenítse meg az egyes görbék esetén a hozzájuk tartozó kontrollpoligont, az érintő- és normálvektorokat, felületek esetén pedig a hozzájuk tartozó kontrollhálót, normálvektorokat, valamint az u - és v -irányú izoparametrikus vonalakat;
- legyen képes tetszőleges fokszámnövelésre;
- a beállítások ki- és bekapcsolását valós időben valósítsa meg.

4.1.2. Rendszerkövetelmények

Funkcionális követelmények. A funkcionális követelmények leírják a rendszer funkcióit, szolgáltatásait és működési megszorításait. Ezek a fent említett felhasználói követelményeken kívül a következők:

- az alkalmazás C++ objektum orientált nyelven legyen megírva, ami biztosítja a platformfüggetlenséget;
- a program az OpenGL grafikus függvénykönyvtár segítségével jelenítse meg a háromdimenziós objektumokat;
- a program a C++ nyelvre épülő, platformfüggetlen Qt keretrendszert használja a felhasználói felületek (*graphical user interface*) megjelenítésére.

Nem-funkcionális követelmények. Az alkalmazás:

- legyen platformfüggetlenül fordítható (ehez megfelelő úgy a C++, mint a Qt és OpenGL);

- egy intuitív, könnyen tanulható és testreszabható felhasználói felülettel rendelkezzen (a Qt segít ennek elérésében);
- legyen megbízható;
- legyen képes a háromdimenziós tartalom gyors, és szé megjelenítésére (ezt az OpenGL biztosítja);
- a különféle beállítások könnyű változtatását tegye lehetővé (Qt);
- a görbék és felületek kontrollpont-konfigurációját pontosan és gyorsan generálja;
- hatékonyság.

Az alkalmazás a fenti összes funkcionális és nem funkcionális követelményt igyekszik teljesíteni.

4.2. Felhasznált technológiák és függvénykönyvtárak

4.2.1. C++

Az alkalmazás és a függvénykönyvtár C++ nyelven íródott, ami – ahogyan a [8] könyv is bemutatja – egy általános célú, magas szintű, objektumorientált programozásra lehetőséget adó nyelv. Mivel a manapság használt operációs rendszerek többségére létezik C++ fordító, így az ebben a nyelvben írt programokat platformfüggetlennek (*cross-platform*) nevezhetjük. A platformfüggetlen alkalmazások osztályán belül pedig abba a csoportba sorolhatjuk a C++ alkalmazásokat, melyeket szükséges az összes operációs rendszerre lefordítani, amelyiken használni szeretnénk. A C++ a C nyelvre alapszik, amely a legelső széles körben elterjedt programozási nyelv, és nagyfokú hatékonyságáról ismert. A C++, a C hatékonyságának megőrzése mellett a könnyebben írható és újrahasznosítható kódra törekszik a kivételkezelés, sablonok, operátor túlterhelés, adatfolyamok, osztályokból való származtatás, polimorfizmus, stb. által.

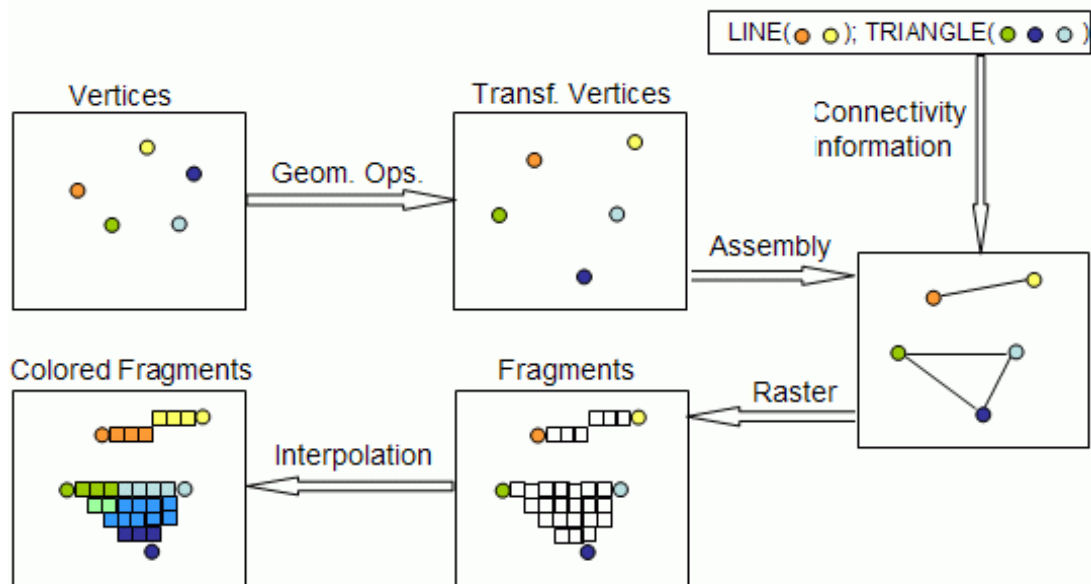
4.2.2. OpenGL

Az OpenGL (Open Graphics Library) a Silicon Graphics amerikai cég által kidolgozott szabvány [6]. Egy olyan API-t valósít meg, amivel két- és háromdimenziós tartalmak jeleníthetők

meg a videokártya segítségével. A számítógépes tervezésben, videojátékok megvalósításában, a tudományos fejlesztésben, és számos grafikai megjelenítést igénylő alkalmazás esetén használják.

Az OpenGL egyik célja, hogy elrejtse az implementáció bonyolultságát egy egyszerűen kezelhető, egységes interfész mögé, ezzel platformfüggetlenítve a videokártya-programozást. Másik célja, hogy elrejtse a különböző hardver platformok változó képességeit, így minden implementáció köteles támogatni a teljes OpenGL utasításkészletet.

Az OpenGL összetett két- és háromdimenziós alakzatokat jelenít meg egyszerű primitívek (például pontok, vonalak és poligonok) segítségével, amelyeket pixelekké konvertál. Egy grafikus csővezeték dolgozza fel a megadott primitíveket, és létrehozza a segítségükkel a raszteres képet. Sokáig a csővezeték minden állapotban egy előre meghatározott függvényt hajtott végre, így csak minimális mértékben volt testreszabható. Az OpenGL 2.0 viszont egy többállapotos csővezetékét kínál, és ez teljes mértékben programozható az [5] könyvben bemutatott **GLSL** (OpenGL Shader Language) használatával. Az alábbi ábra a csővezeték állapotait mutatja be.



1. ábra. A videokártya csővezetékének állapotai, [9]

A programozó felülírhatja a csúcspont transzformációk és a szegmensek textúra és szín beállítási állapotok alapfunkcióit úgynevezett **árnyalók (shader)** használatával. A vertex árnyalók a vertex processzoron, a részecske árnyalók pedig a szegmens processzoron fut-

nak. Mindkét típusú árnyaló esetében az összes alapfunkció felülíródik, tehát minden, az alkalmazáshoz szükséges efektust le kell kódolni. Például tetszőleges textúra koordinátát generálhatunk és alkalmazhatunk pixelenként. A shaderek a GLEW függvénykönyvtárra épülnek, ami biztosítja a megfelelő függvényeket (például *glCreateShader()*, *glShaderSource*, *glCompileShader*, *glAttachShader()*, *glLinkProgram()*, *glUseProgram()*).

Egy bonyolultabb objektum megjelenítéséhez sok OpenGL parancsra van szükség, melyek költséges függvényhívásokat jelentenek. Egy szintér rendereléséért felelős szükséges függvényhívások számát úgy csökkenthetjük le, hogy az egyedi csúcsok koordinátáit és a hozzájuk társított tulajdonságokat (érintő és normálvektorok koordinátáit, színkomponenseket, kód- és textúra-koordinátákat) memóriatakarékosság céljából több különálló, vagy akár vegyes szerkezetű **vertex tömbben** (**vertex arrays**) egyetlen egyszer tároljuk el. A kirajzolás pillanatában az eltárolt adatokat pár függvényhívással egy – a primitívek kapcsolódását leíró – indextömb segítségével érhetjük el. (Például egy n -csúcsú poligon n -darab vertexét egyetlen vertex tömbben tárolhatjuk, és egy függvényhívással hivatkozhatunk rájuk.)

A vertex tömbök kezelésére az OpenGL több függvényt is biztosít.

- A *GLvoid glEnableClientState(GLenum array)* függvény aktiválja a paraméterként megadott típusú tömböt, például *GL_NORMAL_ARRAY* paraméter esetében a normálvektorok tömbjét. Mingid meg kell hívni, mielőtt vertex tömböket használnánk. A párja *GLvoid glDisableClientState(GLenum array)*, amivel a már nem használt tömböt inaktívvá tehetjük. Ha egy tömböt aktiváltunk kötelező inaktíválni is.
- A *GLvoid glVertexPointer(GLint size, GLenum type, GLsize stride, const GLvoid* pointer)* függvény hívásával megtudhatjuk, hol érhetőek el a tárolt adatok, a *pointer* paraméter ugyanis tartalmazni fogja a tömb első vertexének az első koordinátájának memória címét. Hasonló függvények léteznek normál, szín, index, illetve textúra koordinátákat tároló tömbökre is.
- A *GLvoid glArrayElement(GLint i)*-el egyetlen tömbelemre, a *GLvoid glDrawElements(GLenum mode, GLsize count, GLenum type, const GLvoid* indices)* és a *GLvoid glMultiDrawElements(GLenum mode, GLsize* count, GLenum type, const GLvoid** indices GLsize primecount)* függvényekkel a tömbelemek egy listájára, a *GLvoid glDrawAr-*

rays(GLenum mode, GLint first, GLsizei count)-el pedig tömbelemek egy szekvenciájára hivatkozhatunk. Ezeket a dereferenciáló parancsokat geometriák renderelésekor használjuk.

Mivel a geometria adatai a kliens (CPU) oldalon tárolódnak a memóriában, minden egyes megjelenítéskor az adatok a memóriából az OpenGL szerverre (GPU) másolódnak át, ami nagymértékben csökkenti a teljesítményt. A renderelés további gyorsítása érdekében az OpenGL 1.5 verziója a **vertex puffer objektumokat (vertex buffer objects)** vezeti be, melyek lehetővé teszik, hogy az adatokat a szerver oldalán tömbökben tároljuk, egyszeri adatmásolást biztosítva.

A puffer objektumokat kezelő parancsok a GLEW függvénykönyvtárban találhatóak, melyek közül megemlíjtük a legfontosabbakat.

- A *GLvoid glGenBuffers(GLsizei n, GLuint* buffers)* függvénnyel létrehozuk a puffer objektumot azonosítót biztosítva számára.
- A *GLboolean glIsBuffer(GLuint buffer)* adott azonosítóval rendelkező puffer objektum létezését vizsgálja.
- A *GLvoid glBindBuffer(GLenum target, GLuint buffer)* aktiválja a puffer objektumot. Erre szükség van az objektum és adatainak inicializálásakor, amikor ki szeretnénk választani renderelésre, valamint a tárolt adatainak frissítésekor. A függvény zero paraméterrel való hívásakor inaktiváljuk az adott objektumot.
- Aktiválás után helyet foglalhatunk a tárolandó vertex adatoknak a *GLvoid glBufferData(GLenum target, GLsizeiptr size, const GLvoid* data, GLenum usage)* segítségével.
- A videokártya memóriájában létrehozott tömböket indirekt módon módosíthatjuk is a *GLvoid* glMapBuffer(GLenum target, GLenum access)* függvénnyel a szervertől lekért, megfelelő memóriaterületet címző mutató segítségével. A *GLboolean glUnmapBuffer(GLenum target)*-al jelezzük a módosítás végét.
- A *GLvoid glDeleteBuffers(GLsizei n, const GLuint* buffers)* függvényhívással felszabadíthatjuk egy puffer objektum erőforrásait és az azonosítóját, hogyha már nincs szükség rá.

4.2.3. Qt

A Qt [7] nyílt forráskódú, ingyenes, GPL v3 és LGPL v2 licenszű fejlesztői környezetet a Qt Development Frameworks cég fejlesztette ki. Platformfüggetlensége miatt széles körben használják grafikus felhasználói felületeket (GUI) támogató alkalmazások létrehozásához. A Qt C++ standardot használ, amit az úgynevezett **moc** (Meta Object Compiler) és **uic** (User Interface Compiler) speciális előfeldolgozókkal gazdagít. A moc a Qt C++ kiterjesztéseit kezeli. Minden **QObject** makróval rendelkező osztály esetén olyan – úgynevezett "meta-object kódot" tartalmazó – C++ forráskódot generál, amire minden kommunikációra képes osztálynak szüksége van. Az uic a tervezéskor létrehozott (design time) GUI felületek leírását tartalmazó XML fájlok alapján olyan fejlécállományokat generál. Ezek olyan osztályokat tartalmaznak, amik a tervezéskor létrehozott vezérlő elemek dinamikus létrehozásáért felelősek, valamint saját osztályainkat is származtathatjuk belőlük. A Qt nem GUI-specifikus jellemzői közé tartozik az XML feldolgozás mellett, az SQL adatbázis elérés, a hálózati támogatás, a szálkezelés (thread-management), és az egységesített fájlkezelés.

A Qt a felhasználói vezérlő objektumok (úgynevezett **widgetek**) segítségével teszi lehetővé komplex grafikus felületek létrehozását. A megjelenítő motor támogatja az élsimítást (anti-aliasing), valamint a vektorgrafika megjelenítését (SVG formátum), a HTML/CSS alapú stíluskonfigurációt és hardveres-gyorsítást. A Qt és az OpenGL ötvözhető, mivel a beépített *QGLWidget* osztályon keresztül speciális OpenGL funkciók érhetők el, melyek használatával 3D-s alkalmazás készíthető.

A GUI alkalmazások készítésénél egy alapvető probléma a különböző vezérlő objektumok közti kommunikáció megvalósítása, melyre a Qt egy speciális – események bekövetkezésekor kibocsátott jelekre és azokat lekezelő eseménykezelőkre épülő – mechanizmust (**signals and slots mechanism**) biztosít, ami tulajdonképpen a *Megfigyelő* tervezési minta egy implementációja. Egy Qt objektum akár több jelzést is kibocsáthat. Az alap Qt-widgetek már rendelkeznek megfelelő helyzetben kiváltható, beépített jelzésekkel. Ugyanakkor a programozó is létrehozhat új jeleket a **signals** kulcsszó segítségével, amennyiben saját osztályait a *QObject* ősosztályból származtatta. Hasonlóan definiálhatóak saját eseménykezelők is a **slots** kulcsszóval, amelyeket a *QObject::connect(sender, signal, receiver, slot)* metódussal össze is lehet kapcsolni egy nekik megfelelő jelzéssel. A C++ programozásban használt hagyományos callback módszerrel szem-

ben, amiben az objektumok kommunikációját függvénymutatókkal oldják meg, a Qt által nyújtott megoldás típusbiztos, ami azt jelenti, hogy csak megegyező aláírással rendelkező jelzések és eseménykezelők köthetők egymáshoz. A Qt részletesebb bemutatása a [7] leírásban található.

A célkitűzésként megfogalmazott alkalmazás C++ nyelvben íródott és Qt-mechanizmusokat használ. A **Qt Creator** által nyújtott lehetőségekkel szemben választásunk a **Microsoft Visual Studio** környezetre esett, amely VC++ fordítójával szintén kompilálhatóak a Qt függvénykönyvtárai. Az ingyenes Qt Visual Studio Add-in letöltése, a Qt nyílt forráskódú függvénykönyvtárainak telepítése, valamint a Visual Studio-ban a külső függőségek (például az OpenGL Extension Wrangler statikus és dinamikus függvénykönyvtár elérésének) beállítása után, már lehetőségünk van Qt projektet létrehozni és fejleszteni.

4.3. Fontosabb osztályok interfésze és hierarchiája

4.3.1. GUI csomag

A **GUI** (Graphical User Interface) csomag azokat a és Qt-ra alapuló osztályokat tartalmazza, amik az alkalmazás grafikus felületéért felelősek. Ezek a jelzések és csatlakozók mechanizmust alkalmazzák a vizuális felületen belüli és más osztályokkal való kommunikáció során.

GLWidget. A *GLWidget* osztály az a widget, amely különböző görbéket, felületeket és más OpenGL grafikus objektumokat renderel. Ez az osztály a *QGLWidget* osztályból származik, ami a Qt applikációkba beépített OpenGL grafikák megjelenítéséhez kínál funkcionálisokat. A *paintGL()* metódusa – amit a widget minden egyes frissítésekor meghívódik az *updateGL()* függvény által – rendereli az OpenGL színteret. A *resizeGL()* metódus segítségével beállíthatjuk az OpenGL ablak méretét. Ezt a metódust mindig meg kell hívni, amikor a widgetet újraméretezzük. Az *initializeGL()* metódussal pedig az OpenGL renderelési kontextusát állíthatjuk be, és csak egyszer kell meghívni a widget létrehozásakor. A *GLWidget* osztály felülírja ezeket a virtuális függvényeket az alkalmazásnak megfelelően.

Egy OpenGL alapú nézőke, jelen esetben a *GLWidget* rendelkezik egy úgynevezett *kamerával*, ami meghatározza, hogy mit és hogyan látunk ebben a háromdimenziós térben. A *kamerák* két vetítési módban, *perspektivikusan* és *ortografikusan* tudják leképezni a teret a képsíkra. (Példa: hogyha két párhuzamos egyenest vizsgálunk, a *perspektivikus* vetítésnél az egyenesek a

végtelenben összefutnak, míg az *ortografikus* vetítés során mindig párhuzamosak maradnak.) Mindkét módnak megvannak a saját előnyei és hátrányai.

A *GLWidget* fogja a központi helyet elfoglalni fő ablakon.

SideWidget. Szükség van egy olyan eszköztárra is, amellyel váltogathatjuk az alkalmazás szolgáltatásait és más funkciókat is elérhetünk általa. Ezt biztosítja a *SideWidget* osztály, ami a *QWidget*, és *Ui::SideWidget* leszármazottja. Lehetőséget ad a geometria tengelyek körüli elforgatására, a tengelyek menti translációjára, fókuszának növelésére és csökkentésére, és skálázási tényezőjének beállítására. A felhasználó a háromdimenziós színtér nézőpontjait is váltogathatja, be- és kikapcsolhatja, valamint a *GLWidget* kameráját interaktívan mozgathatja. A kezdeti értékek visszaállítására is lehetőség van. Mindezeket a szolgáltatásokat a widget által nyújtott elemek, például gombok, jelölőnégyzetek, sliderek, spinboxok segítségével lehet használni.

A *SideWidget* a fő ablak jobb oldalára van rögzítve.

MainWindow. Az alkalmazás kiindulópontja a fő ablak, ami a program esetében a *MainWindow* nevet viseli, és ami a *QMainWindow* és *Ui::MainWindow* osztályok leszármazottja. A *MainWindow* az a Qt widget, amely összeköti az alkalmazás összes vizuális elemét, és ami legelőször elindul az alkalmazás indításakor. A konstruktora hozza létre a fent említett vizuális elemeket, és elhelyezi őket a *MainWindow* objektumon. Ez az osztály rendeli hozzá a *SideWidget* elemei által kibocsájtott jelekhez a *GLWidget* megfelelő csatlakozó függvényeit a *connect(sender, signal, receiver, slot)* metódussal. A csatlakozó függvények a megfelelő értékváltoztatások után az *updateGL()* függvényhívással frissítik a háromdimenziós színteret.

4.3.2. Core csomag

A következőkben ismertetett csomagok a függvénykönyvtár részei, akár egy másik független alkalmazásban is felhasználhatóak. A **Core** csomag olyan OpenGL alapú osztályokat és adatstruktúrákat tartalmaz, melyek különböző háromdimenziós objektumok megjelenítésében játszanak szerepet. A megjelenítés gyorsítására vertex puffer objektumokat használunk.

Matrix. A *Matrix* sablon osztállyal kollokációs mátrixokat, kontrollpoligonokat, kontrollhálókat, (parciális) deriváltak értékeit tároljuk el. Sor és oszlop mátrix származtatott osztályai is gyakran használatosak a programban.

RealSquareMatrix. Ez az osztály a *Matrix<GLdouble>* leszármazottja. Olyan numerikus módszerek vannak benne implementálva, mint az LU-dekompozíció és a lineáris egyenletrendszer megoldása.

DCoordinate3. A legalapvetőbb adatszerkezet, a három dimenziós Descartes koordináták reprezentációja. A segítségével görbék pontjait, érintő és gyorsulás vektorait, valamint felületek pontjait, a hozzájuk rendelt normál vektorokat, magasabb rendű parciális deriváltakat és kontrollpontokat fogunk meghatározni az alkalmazásban.

GenericCurve3. A *GenericCurve3* bármely típusú görbe bázisosztálya lehet. A görbepontok koordinátáinak és az első illetve másod rendű deriváltak beállítása a származtatott osztály feladata marad. A *GenericCurve3* osztály rendereléshez és a frissítéshez szükséges metódusokat biztosít, ami vertex puffer objektumok segítségével történik.

Az osztály *_derivative* Descartes koordinátákból álló mátrix adattagjában a deriváltak értékei vannak tárolva adott pontokban, a *_vbo_derivative* egészekből álló sormátrix adattagjában pedig a videokártya azon memóriacímei találhatóak, ahol ezek az adatok el vannak tárolva.

A következő metódus végzi a renderelést a vertex koordináta tömbök aktiválása után:

```
1 GLboolean GenericCurve3::RenderDerivatives(GLuint order ,
2                                             GLenum render_mode) const
3 {
4     //ha a görbe pontjait kell renderelni
5     if (!order)
6     {
7         //ha a renderelési mód nem megfelelő , a renderelés nem engedélyezett
8         if (render_mode != GL_LINE_STRIP && render_mode != GL_LINE_LOOP
9             && render_mode != GL_POINTS)
```

```

10      {
11          glBindBuffer(GL_ARRAY_BUFFER, 0);
12          glDisableClientState(GL_VERTEX_ARRAY);
13          return GL_FALSE;
14      }
15      //ha a renderelés módja megfelelő, a renderelés végbemehet
16      glDrawArrays(render_mode, 0, point_count);
17  }
18  else
19      //ha valahányad rendű deriváltat kell renderelni
20      {
21          ...
22          glDrawArrays(render_mode, 0, 2 * point_count);
23      }
24      ...
25 }

```

A következő metódus a vertex puffer objektumok frissítéséért felel. Miután törli a régi puffer objektumokat, újakat hoz létre, és azonosítóikat a *_vbo_derivative* tömbben tárolja. Majd feltölti őket a megfelelő adatokkal.

```

1  GLboolean GenericCurve3::UpdateVertexBufferObjects(GLenum usage_flag)
2  {
3      ...
4      GLuint curve_point_count = _derivative.GetColumnCount();
5      GLfloat *coordinate = 0;
6
7      //a görbe pontjait a magasabb rendű deriváltakhoz hasonlóan kezeljük
8      GLuint curve_point_byte_size = 3 * curve_point_count * sizeof(GLfloat);
9      ...
10     //magasabb rendű deriváltak
11     GLuint higher_order_derivative_byte_size = 2 * curve_point_byte_size;
12     for (GLuint d = 1; d < _derivative.GetRowCount(); ++d)

```

```

13  {
14      glBindBuffer(GL_ARRAY_BUFFER, _vbo_derivative(d));
15      glBufferData(GL_ARRAY_BUFFER, higher_order_derivative_byte_size,
16
17      coordinate = (GLfloat*)glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);
18      ...
19      for (GLuint i = 0; i < curve_point_count; ++i)
20      {
21          DCoordinate3 sum = _derivative(0, i);
22          sum += _derivative(d, i);
23          for (GLint j = 0; j < 3; ++j)
24          {
25              *coordinate = (GLfloat)_derivative(0, i)[j];
26              *(coordinate + 3) = (GLfloat)sum[j];
27              ++coordinate;
28          }
29          coordinate += 3;
30      }
31      ...
32  }
33  glBindBuffer(GL_ARRAY_BUFFER, 0);
34  return GL_TRUE;
35 }

```

LinearCombination3. A *LinearCombination3* absztrakt osztály bármilyen adat pontokkal és súlyfüggvények rendszerével megadható görbe alaposztálya lehet (például a 8 és 4 előállítású görbéké). Van egy *Derivatives* nevű alosztálya, ami egy olyan oszlopmátrix, ami tetszőleges két- és háromdimenziós lineáris kombináció r -ed ($r \geq 0$) rendű deriváltjait tárolja el adott csomópont értékben. A származtatott osztálynak implementálnia kell a

```

1 virtual GLboolean BlendingFunctionValues(GLdouble u,
2                                         RowMatrix<GLfloat>& values) const = 0;

```

```

3 virtual GLboolean CalculateDerivatives(GLuint max_order_of_derivatives,
4                                     GLdouble u, Derivatives& d) const = 0;

```

metódusokat ahhoz, hogy generálni lehessen a görbe alakzatát és, hogy megoldható legyen a görbe interpolációjának problémája. A *virtual GLboolean UpdateDataForInterpolation(const ColumnMatrix<GLdouble> & knot_vector, const ColumnMatrix<DCoordinate3> & data_points_to_interpolate)* LU dekompozícióra alapuló függvényt is érdemes újra deklarálni és implementálni azokban az esetekben, amikor a görbe interpolációs problémájára ez hatékony megoldás.

A görbe képe GenericCurve3* típus lesz, ami a következő metódusban van generálva (a képre mutató pointer a *result* nevet viseli a kódban):

```

1 GenericCurve3* LinearCombination3::GenerateImage
2     (GLuint max_order_of_derivatives,
3     GLuint div_point_count, GLenum usage_flag) const
4 {
5     Derivatives d;
6     ...
7     // beállítjuk a deriváltakat a végpontokban, ha sikerült kiszámolni azokat
8     (*result)._derivative.SetColumn(0, d);
9     (*result)._derivative.SetColumn(div_point_count - 1, d);
10    ...
11    // a közbe eső pontokban is beállítjuk a deriváltakat
12    GLdouble u_step = (_u_max - _u_min) / (div_point_count - 1);
13    GLdouble u = _u_min;
14    for (GLuint i = 1; i < div_point_count - 1; ++i)
15    {
16        u += u_step;
17        d.LoadNullVectors();
18        if (CalculateDerivatives(max_order_of_derivatives, u, d))
19        {
20            (*result)._derivative.SetColumn(i, d);
21        }
22        ...

```

```

23     }
24     return result;
25 }

```

TriangularFace. A *TriangularFace* osztály három vertex indexet tárol, amik a megjeleníteni kívánt felületek háromszögesített hálójának egy háromszögű oldallapját határozzák meg.

Colors4, Lights, Materials. Ezek az osztályok a felületek esztétikus megjelenítéséért felelősek.

Az OpenGL irányított-, pontszerű- és reflektorfényt tud kezelni. Egy szintér legfeljebb 8 különböző fényforrással rendelkezhet, amik olyan tulajdonságokkal rendelkeznek, mint például környező, kiterjedő és spekuláris intenzitások, pozíció, irány, lineáris és kvadratikus csillapítás, stb. Az alábbi metódusok közül az első kettővel a *Lights* osztály a fényforrásokat állítja, az utóbbi kettővel pedig a *Materials* osztály a felület anyagi tulajdonságait.

```

1 GLvoid glLight{i|f}(GLenum light_index, GLenum parameter_name,
2
3 GLvoid glLight{i|f}v(GLenum light_index, GLenum parameter_name,
4
5 void glMaterial{i|f}(GLenum face, GLenum parameter_name,
6
7 void glMaterial{i|f}(GLenum face, GLenum parameter_name,
8

```

TriangulatedMesh3. A *TriangulatedMesh3* osztály Descartes-koordinátákból álló vektorokban eltárolja a geometria vertexeit, normál vektorait, TCoordinate4 típusú vektorban a textúra koordinátáit, TriangularFace típusú vektorban pedig a háromszögű oldallapjait. Vertex puffér objektumok segítségével ezek az adatok a videokártyán vannak eltárolva.

Az osztály legjelentősebb szolgáltatásáért, a renderelésért felelős metódus először a *glEnableClientState()* paranccsal aktiválja a vertex, normál vektor és textúra koordinátavektorok kliens státuszát, amit renderelés után inaktívál, majd a így jár el:

```

1 GLboolean TriangulatedMesh3::Render(GLenum render_mode) const
2 {
3     ...
4     glBindBuffer(GL_ARRAY_BUFFER, _vbo_tex_coordinates);
5     glTexCoordPointer(4, GL_FLOAT, 0, (const GLvoid *)0);
6     //hasonlóan a normál vektorok és a vertexek esetében is a tároló
7     //puffer objektum memóriabeli címére mutató lekérése
8     ...
9     glNormalPointer(GL_FLOAT, 0, (const GLvoid *)0);
10    ...
11    glVertexPointer(3, GL_FLOAT, 0, (const GLvoid *)0);
12    //a puffer tömb elemek aktiválása a háromszög oldallapok
13    //indexelt vertexeivel
14    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, _vbo_indices);
15    glDrawElements(render_mode, 3 * _face.size(), GL_UNSIGNED_INT,
16    ...
17    ...
18    glBindBuffer(GL_ARRAY_BUFFER, 0);
19    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
20
21    return GL_TRUE;
22 }

```

A vertex puffer objektumok módosítására is lehetőséget nyújt az osztály a következő metódusában. Először törölnünk kell a régi puffer objektumokat, majd létre kell hoznunk az újakat. Ha egy létrehozási művelet sikertelen, az összes előzőleg létrehozott puffer objektumot eltöröljük. Hatékonysági okokból minden GLdouble típusú koordinátát GLfloat típusúvá konvertálunk, az adatok pufferbe töltésére pedig – *glMapBuffer* és *glUnmapBuffer* függvényeken kívül – kiegészítő pontereket használunk. Ha csak lehet, több puffer tömböt járunk be egyszerre.

```

1 GLboolean TriangulatedMesh3::UpdateVertexBufferObjects(GLenum usage_flag)
2 {
3     ...

```

```

4  GLuint vertex_byte_size = 3 * _vertex.size() * sizeof(GLfloat);
5  glBindBuffer(GL_ARRAY_BUFFER, _vbo_vertices);
6  glBufferData(GL_ARRAY_BUFFER, vertex_byte_size, 0, _usage_flag);
7  GLfloat *vertex_coordinate = (GLfloat*)glMapBuffer(GL_ARRAY_BUFFER,
8
9  ...
10 GLfloat *normal_coordinate = (GLfloat*)glMapBuffer(GL_ARRAY_BUFFER,
11
12 for (vector<DCoordinate3>::const_iterator vit = _vertex.begin();
13      vit = _normal.begin(); vit != _vertex.end(); ++vit, ++nit)
14 {
15     for (GLint component = 0; component < 3; ++component)
16     {
17         *vertex_coordinate = (GLfloat)(*vit)[component];
18         ++vertex_coordinate;
19         *normal_coordinate = (GLfloat)(*nit)[component];
20         ++normal_coordinate;
21     }
22 }
23
24 GLuint tex_byte_size = 4 * _tex.size() * sizeof(GLfloat);
25 ...
26 GLfloat *tex_coordinate = (GLfloat*)glMapBuffer(GL_ARRAY_BUFFER,
27
28 memcpy(tex_coordinate, &_tex[0][0], tex_byte_size);
29
30 GLuint index_byte_size = 3 * _face.size() * sizeof(GLuint);
31 ...
32 GLuint *element = (GLuint*)glMapBuffer(GL_ELEMENT_ARRAY_BUFFER,
33
34 for (vector<TriangularFace>::const_iterator fit = _face.begin();
35      fit != _face.end(); ++fit)

```

```

36  {
37      for (GLint node = 0; node < 3; ++node)
38      {
39          *element = (*fit)[node];
40          ++element;
41      }
42  }
43
44  glBindBuffer(GL_ARRAY_BUFFER, _vbo_vertices);
45  if (!glUnmapBuffer(GL_ARRAY_BUFFER))
46      return GL_FALSE;
47  //hasonlóan a többi használt puffer objektumra
48  ...
49  glBindBuffer(GL_ARRAY_BUFFER, 0);
50  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
51
52  return GL_TRUE;
53 }

```

TensorProductSurfaces. A TensorProductSurface absztrakt osztály az (2) felületek kiértékeléséhez és rendereléséhez szükséges metódusokat biztosítja. A származtatott osztály feladata implementálni a

```

1 virtual GLboolean UBlendingFunctionValues(GLdouble u_knot,
2                                           RowMatrix<GLdouble>& blending_values) const = 0;
3 virtual GLboolean VBlendingFunctionValues(GLdouble v_knot,
4                                           RowMatrix<GLdouble>& blending_values) const = 0;
5 virtual GLboolean CalculatePartialDerivatives(GLdouble u, GLdouble v,
6                                               PartialDerivatives& pd) const = 0;

```

metódusokat.

A felület megjelenített képe TriangulatedMesh3* típusú lesz, ami a következő metódusban van generálva (a képre mutató pointer a *result* nevet viseli a kódban):

```

1 TriangulatedMesh3* TensorProductSurface3::GenerateImage
2             GLuint u_div_point_count, GLuint v_div_point_count,
3             GLenum usage_flag) const
4 {
5     ...
6     //a vertexek és a háromszögû oldallapok számának kiszámolása:
7     vertex_count = u_div_point_count * v_div_point_count
8     face_count = 2 * (u_div_point_count - 1) * (v_div_point_count - 1)
9
10    //egységes felosztású rács az értelmezési tartományon és az egységnyi
11    //négyzeten
12    GLdouble du = (_u_max - _u_min) / (u_div_point_count - 1);
13    GLdouble dv = (_v_max - _v_min) / (v_div_point_count - 1);
14    GLfloat sdu = 1.0f / (u_div_point_count - 1);
15    GLfloat tdv = 1.0f / (v_div_point_count - 1);
16
17    GLuint current_face = 0;
18    PartialDerivatives pd;
19
20    for (GLuint i = 0; i < u_div_point_count; ++i)
21    {
22        GLdouble u = _u_min + i * du;
23        GLfloat s = i * sdu;
24        for (GLuint j = 0; j < v_div_point_count; ++j)
25        {
26            GLdouble v = _v_min + j * dv;
27            GLfloat t = j * tdv;
28            /*
29             3-2
30             ||
31             0-1
32            */

```

```

33     GLuint index[4];
34     index[0] = i * v_div_point_count + j;
35     index[1] = index[0] + 1;
36     index[2] = index[1] + v_div_point_count;
37     index[3] = index[2] - 1;
38
39     //a felületre vonatkozó adatok kiszámítása
40     CalculatePartialDerivatives(u, v, pd);
41
42     //a felület pontjainak beállítása
43     (*result)._vertex[index[0]] = pd.point;
44     //a felület normálvektora
45     (*result)._normal[index[0]] = pd.diff1u;
46     (*result)._normal[index[0]] ^= pd.diff1v;
47     (*result)._normal[index[0]].normalize();
48     //textúra koordináták
49     (*result)._tex[index[0]].s() = s;
50     (*result)._tex[index[0]].t() = t;
51     //oldallapok
52     if (i < u_div_point_count - 1 && j < v_div_point_count - 1)
53     {
54         (*result)._face[current_face][0] = index[0];
55         (*result)._face[current_face][1] = index[1];
56         (*result)._face[current_face][2] = index[2];
57         ++current_face;
58         (*result)._face[current_face][0] = index[0];
59         (*result)._face[current_face][1] = index[2];
60         (*result)._face[current_face][2] = index[3];
61         ++current_face;
62     }
63 }
64 }

```

```

65  return result;
66 }

```

Emellett az

$$s(u_i, v_j) = \mathbf{d}_{i,j}, i = 0, 1, \dots, n, j = 0, 1, \dots, m,$$

$$[\mathbf{d}_{i,j}]_{i=0,j=0}^{n,m} \in M_{n+1,m+1}(\mathbb{R}^3),$$

$$u_{\min} \leq u_0 \leq u_1 \leq \dots \leq u_n \leq u_{\max},$$

$$v_{\min} \leq v_0 \leq v_1 \leq \dots \leq v_n \leq v_{\max},$$

interpolációs problémára is megoldást nyújt a

```

1  GLboolean  TensorProductSurface3::UpdateDataForInterpolation
2
3      (const RowMatrix<GLdouble>& u_knot_vector,
4       const ColumnMatrix<GLdouble>& v_knot_vector,
5       Matrix<DCoordinate3>& data_points_to_interpolate)
6  {
7      ...
8      // az u-kollokációs mátrix kiszámítása és az LU-dekompozíció
9      // elvégzése rajta
10     RowMatrix<GLdouble> u_blending_values;
11     RealSquareMatrix u_collocation_matrix(row_count);
12     for (GLuint i = 0; i < row_count; ++i)
13     {
14         if (!UBlendingFunctionValues(u_knot_vector(i), u_blending_values))
15             return GL_FALSE;
16         u_collocation_matrix.SetRow(i, u_blending_values);
17     }
18     if (!u_collocation_matrix.PerformLUDecomposition())
19         return GL_FALSE;
20     // hasonlóan v-re
21     ...

```

```

21  Matrix<DCoordinate3> a(row_count, column_count);
22  if (!u_collocation_matrix.SolveLinearSystem
23      (data_points_to_interpolate, a))
24      return GL_FALSE;
25  // hasonlóan v-re
26  ...
27  return GL_TRUE;
28 }

```

metódusával.

4.3.3. Cyclic csomag

CyclicCurve3. Ez az osztály a (8) ciklikus görbék renderelését biztosítja. A *CyclicCurve3* a *LinearCombination3* osztály leszármazottja. Bázisosztályát az *_n* fokszám, *_c_n* normalizációs konstans, *_lambda_n* fázis eltolás, illetve a *_bino* binomiális együtthatókat tároló mátrix adattagokkal egészíti ki, amik a (8) előállításához szükségesek. A *_lambda_n* változót $\frac{2\pi}{2n+1}$ -re, a *_c_n* változót a (??) rekurzió alapján, a *_bino* mátrixot pedig a *_CalculateBinomialCoefficients(2 * _n)* metódushívással inicializáljuk.

A *LinearCombination3* osztály *BlendingFunctionValues* virtuális metódusát a (8) képletet használva implementáltuk.

```

1  GLboolean CyclicCurve3::BlendingFunctionValues
2      (GLdouble u, RowMatrix<GLdouble>& values) const
3  {
4      ...
5      GLuint dimension = 2 * _n + 1;
6      values.ResizeColumns(dimension);
7      for (GLuint i = 0; i < dimension; ++i)
8      {
9          values[i] = _c_n * pow( (1.0 + cos(u + i * _lambda_n)), (int)_n);
10     }
11     return GL_TRUE;
12 }

```

A *LinearCombination3* osztály *CalculateDerivatives* metódusa a (9) és a (10) képletek implementálásával a következő:

```

1  GLboolean CyclicCurve3::CalculateDerivatives
2  (GLuint max_order_of_derivatives, GLdouble u, Derivatives& d) const
3  {
4      ...
5      GLuint dim = 2 * _n + 1;
6      DCoordinate3 centroid;
7      for (GLuint i = 0; i < dim; ++i)
8      {
9          centroid += _data[i];
10     }
11     centroid /= dim;
12     for (GLuint r = 0; r < d.GetRowCount(); ++r)
13     {
14         for (GLuint i = 0; i < dim; ++i)
15         {
16             for (GLuint k = 0; k <= _n-1; ++k)
17             {
18                 d[r] += _bino(dim - 1, k) * std::pow((GLdouble)(_n-k), (GLint)r)
19                 * cos( (_n-k) * (u + i * _lambda_n) + r * PI / 2 ) * _data[i];
20             }
21         }
22         d[r] *= 2.0;
23         d[r] /= dim;
24         d[r] /= _bino(dim - 1, _n);
25         if (r == 0)
26         {
27             d[r] += centroid;
28         }
29     }
30     return GL_TRUE;

```

CyclicSurface3. A *CyclicSurface3* osztály a (18) ciklikus felület reprezentációjáért felelős. A *TensorProductSurface3* osztály leszármazottja, amit az $_n$ és $_m$ foksámokkal, a $_c_n$ és $_c_m$ normalizációs konstansokkal, illetve a $_lambda_n$ és $_lambda_m$ fáziseltolás adattagokkal egészíti ki. A konstruktorban a változókat a (18) képletnek megfelelően inicializáljuk.

A *TensorProductSurface3* osztály *UBlendingFunctionValues* illetve *VBlendingFunctionValues* metódusait hasonlóan implementáltuk, mint a ciklikus görbék esetében. A (18) előállítású ciklikus felület iránymenti parciális deriváltjai a ciklikus görbék (10) deriváltját ismerve könnyen kifejezhetők. Ezt felhasználva írtuk felül az alaposztály *CalculatePartialDerivatives* metódusát.

```

1  GLboolean CyclicSurface3::CalculatePartialDerivatives(GLdouble u,
2                                     GLdouble v, PartialDerivatives& pd) const
3  {
4      ...
5      GLuint dimension1 = 2 * _n + 1;
6      GLuint dimension2 = 2 * _m + 1;
7      RowMatrix<GLdouble> Cu, Cv;
8      UBlendingFunctionValues(u, Cu);
9      VBlendingFunctionValues(v, Cv);
10     ...
11     for (GLuint i = 0; i < dimension1; ++i)
12     {
13         DCoordinate3 aux;
14         for (GLuint j = 0; j < dimension2; ++j)
15         {
16             aux += _data(i, j) * Cv[j];
17         }
18         GLdouble beta = 0.0;
19         for (GLuint k = 0; k <= _n - 1; ++k)
20         {
21             beta += bino(dimension1 - 1, k) * (_n-k) * cos( (_n-k) *

```

```

22         (u + i * _lambda_n) + PI / 2 );
23     }
24     pd.point += aux * Cu[i];
25     pd.diff1u += aux * beta;
26 }
27 pd.diff1u *= 2.0;
28 pd.diff1u /= dimension1;
29 pd.diff1u /= bino(dimension1 - 1, _n);
30 //Hasonlóan számítható ki pd.diff1v
31 ...
32 return GL_TRUE;
33 }

```

4.3.4. Trigonometric csomag

CosineItem, SineItem. Ezek az osztályok a (12) képlet α, ψ, p komponenseket tároló koszinusz, valamint β, φ, q komponenseket tároló szinusz elemeket leíró adatszerkezeteket definiálnak.

TrigonometricCurves3. A *TrigonometricCurve3* osztály *_delta* adattagja a dimenziót, *_C* és *_S* oszlop mátrixai pedig a (12) képletben szereplő koszinusz, illetve szinusz komponenseket tárolják. Az oszlop mátrixokat a *InsertNewCosineItem* és *InsertNewSineItem* metódusok segítségével tölthetjük fel, az *EraseCosineItem* és *EraseSineItem* metódusokkal pedig elemeket törölhetünk ki belőlük. Így tetszőleges (11) típusú trigonometrikus görbét reprezentálhatunk.

A valami trigonometrikus görbe generálására két metódust is biztosít az osztály. Az egyik megoldásban *GenericCurve** típus lesz a görbe képe (a kép a *result* nevet viseli a kódban), és a (12) képlet segítségével határozzuk meg a deriváltakat:

```

1 GenericCurve3* TrigonometricCurve3::GenerateImage
2     (GLuint max_order_of_derivatives, GLuint div_point_count,
3      GLenum usage_flag)
4 {
5     ...

```

```

6  GLdouble u_step = TWO_PI / (div_point_count - 1);
7  for (GLuint i = 1; i < div_point_count ; ++i)
8  {
9      u = i * u_step;
10     //a görbe kontrollpontjainak beállítása
11     for(GLint l = 0; l < _delta; ++l)
12     {
13         for(GLint r = 0; r <= max_order_of_derivatives; ++r)
14         {
15             (*result)._derivative(r,i)[l] = 0;
16             for(GLuint c = 0; c < (int)_C[l].GetColumnCount(); ++c)
17             {
18                 (*result)._derivative(r,i)[l] += pow((double)_C[l][c]._p,
19                 (double)r) * _C[l][c]._alpha * cos(_C[l][c]._p *
20                 u + _C[l][c]._psi + r * PI / 2) /
21                 bc(2 * div_point_count, div_point_count - _C[l][c]._p);
22             }
23             for(GLuint s = 0; s < (int)_S[l].GetColumnCount(); ++s)
24             {
25                 (*result)._derivative(r,i)[l] += pow((double)_S[l][s]._q,
26                 (double)r) * _S[l][s]._beta * sin(_S[l][s]._q *
27                 u + _S[l][s]._phi + r * PI / 2) /
28                 bc(2 * div_point_count, div_point_count - _S[l][s]._q);
29             }
30         }
31     }
32 }
33 return result;
34 },

```

, ahol *bc* RealSquareMatrix típusú, és a *_CalculateBinomialCoefficients(2*div_point_count, bc)* metódushívással inicializáltuk.

Egy másik megoldás a ciklikus reprezentáció létrehozása:

```
1 CyclicCurve3* TrigonometricCurve3::
2     GenerateCyclicRepresentation(GLuint degree)
3 {
4     //ellenőrizzük, hogy a paraméterként megadott foksám ne legyen kisebb,
5     //mint a koszinusz és szinusz elemek p és q
6     //komponenseinek legkisebb értéke
7     ...
8     GLint dimension = 2*degree + 1;
9     CyclicCurve3* result = new CyclicCurve3(degree);
10    GLdouble lambda = TWO_PI / (dimension);
11
12    //a ciklikus görbe kontrollpontjainak beállítása
13    for(GLint i = 0; i < dimension; ++i)
14    {
15        for(GLint l = 0; l < (int)_C.GetRowCount(); ++l)
16        {
17            (*result)[i][l] = 0.0;
18            for(GLuint c = 0; c < (int)_C[l].GetColumnCount(); ++c)
19            {
20                (*result)[i][l] += _C[l][c]._alpha *
21                    cos(_C[l][c]._p * lambda * (-i) +
22                        _C[l][c]._psi) / bc(2 * degree, degree - _C[l][c]._p);
23            }
24            for(GLuint s = 0; s < (int)_S[l].GetColumnCount(); ++s)
25            {
26                (*result)[i][l] += _S[l][s]._beta *
27                    sin(_S[l][s]._q * lambda * (-i) +
28                        _S[l][s]._phi) / bc(2 * degree, degree - _S[l][s]._q);
29            }
30        }
31        (*result)[i] *= bc(2 * degree, degree);
```

```

32 }
33 return result;
34 }

```

TrigonometricSurfaces3.

4.3.5. Rational csomag

Ez a csomag a *RationalCyclicCurve3* és a *RationalCyclicSurface3* osztályokat tartalmazza, amik a (4) előállítású racionális trigonometrikus görbék, és a (24) előállítású racionális trigonometrikus felületek megjelenítéséhez szükségesek. Ezeket az osztályokat a *LinearCombination3* illetve a *TensorProductSurface3* származtatják, és megvalósításuk hasonló a ciklikus görbék és felületekéhez. Ezért ebben a szakaszban nem részletezzük az implementációjukat.

4.4. Alkalmazások és eredmények

A 2.(a) és 2.(b) ábrák ciklikus görbék segítségével egy hipociklois, illetve egy tóruszcsomó kontrollpont-alapú egzakt leírását szemléltetik.

A 2.(a) ábrán feltüntetett hipocikloist az

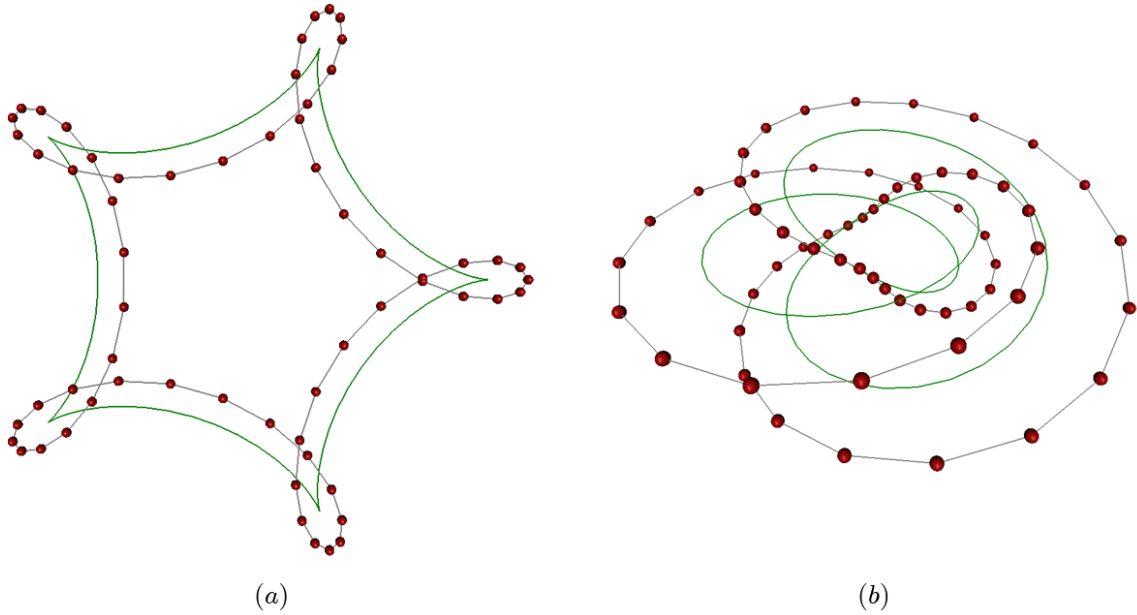
$$\begin{cases} \tilde{x}_1(u) = 4 \cos(u) + \cos(4u) \\ \tilde{x}_2(u) = 4 \sin(u) - \sin(4u) \end{cases} \quad (25)$$

paraméteres képletekkel írhatjuk le, ahol $u \in [0, 2\pi]$. Az ezt reprezentáló ciklikus görbe képét a következő utasításokkal generáltuk a *GLWidget* osztály *initalizeGL* metódusában:

```

1 // koordinátafüggvények felépítése (TrigonometricCurve3* _tc)
2 TrigonometricCurve3* _trigo_curve = new TrigonometricCurve3(3);
3 if (!_trigo_curve)
4     throw Exception("Could not create trigonometric curve!");
5 _trigo_curve->InsertNewCosineItem(0, 4, 0, 1);
6 _trigo_curve->InsertNewCosineItem(0, 1, 0, 4);
7 _trigo_curve->InsertNewSineItem(1, 4, 0, 1);
8 _trigo_curve->InsertNewSineItem(1, -1, 0, 4);

```



2. ábra. Az (a) és (b) ábrákon egy-egy harmincadfokú ciklikus görbét láthatunk, melyek rendre a (25) hipocikloist illetve a (26) tóruszcsomót állítják elő egzaktul.

```

9
10 // görbe alakjának generálása (GenericCurve3* _itc)
11 _trigo_curve_image = _trigo_curve->GenerateImage(max_order_of_derivatives,
12                                                    div_point_count, GL_STATIC_DRAW);
13 if (!_trigo_curve_image)
14     throw Exception("Could not create image of trigonometric curve!");
15 if (!_trigo_curve_image->UpdateVertexBufferObjects(GL_STATIC_DRAW))
16 {
17     // memóriaszivárgás elkerülése
18     delete _trigo_curve, _trigo_curve = 0;
19     delete _trigo_curve_image, _trigo_curve_image = 0;
20     throw Exception("Could not create the geometry in GPU!");
21 }
22
23 //ciklikus reprezentáció és képének generálása
24 //(CyclicCurve3* _cc, GenericCurve3* _icc)

```

```

25 _trigo_cyclic_curve = _trigo_curve->GenerateCyclicRepresentation(degree);
26 if(!_trigo_cyclic_curve)
27     throw Exception("Could not create cyclic representation!");
28 _trigo_cyclic_curve_image = _trigo_cyclic_curve->GenerateImage
29     (max_order_of_derivatives, div_point_count, GL_STATIC_DRAW);
30
31 if(!_trigo_cyclic_curve_image)
32     throw Exception("Could not create image of cyclic curve!");
33 if(!_trigo_cyclic_curve->UpdateVertexBufferObjectsOfData(GL_STATIC_DRAW))
34 {
35     // memóriaszivárgás elkerülése
36     delete _trigo_cyclic_curve, _trigo_cyclic_curve = 0;
37     delete _trigo_cyclic_curve_image, _trigo_cyclic_curve_image = 0;
38     throw Exception("Could not create the geometry in GPU!");
39 }
40 if(!_trigo_cyclic_curve_image->UpdateVertexBufferObjects(GL_STATIC_DRAW))
41     ... // memóriaszivárgás elkerülése

```

Hasonló módon hoztuk létre a 2.(b) ábrán látható

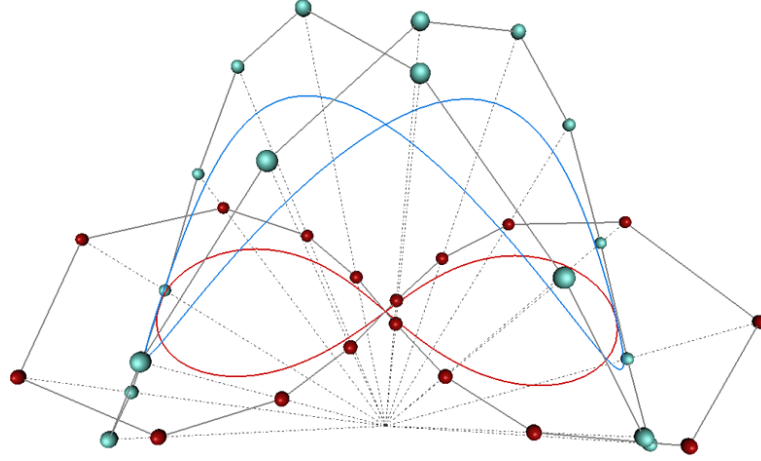
$$\begin{cases} \tilde{x}_1(u) = 2 \cos(3u) + \frac{1}{2} \cos(5u) + \cos(u) \\ \tilde{x}_2(u) = 2 \sin(3u) + \frac{1}{2} \sin(5u) + \sin(u) \\ \tilde{x}_3(u) = \sin(2u) \end{cases} \quad (26)$$

paraméterezésű tóruszcsonót is.

A 3. ábrán a racionális

$$\begin{cases} x_1(u) = \frac{\cos(u)}{1 + \sin^2(u)}, \\ x_2(u) = \frac{\sin(u) \cos(u)}{1 + \sin^2(u)} \end{cases} \quad (27)$$

Bernoulli-féle lemniszkáta egzakt kontrollpont-alapú leírását láthatjuk (4) előállítású racionális trigonometrikus görbe segítségével. Az előállításhoz szükséges súlyokat és kontrollpontokat a



3. ábra. A Bernoulli-féle lemniszkáta egzakt kontrollpont alapú előállítása nyolcadfokú racionális trigonometrikus görbeként. Az ábrán kék színnel tüntettük fel a háromdimenziós öskép alakját és ciklikus reprezentációjához tartozó kontrollpontokat. Az öskép kontrollpontjait a $w = 1$ egyenletű hipersíkra vetítve, megkapjuk a lemniszkátát meghatározó – racionális előállításához szükséges – piros kontrollpontokat.

(27) görbe

$$\begin{cases} \tilde{x}_1(u) = \cos(u), \\ \tilde{x}_2(u) = \frac{1}{2} \sin(2u), \\ \tilde{x}_3(u) = \frac{3}{2} - \frac{1}{2} \cos(2u) \end{cases}$$

alakú ösképének segítségével írhatjuk le, hiszen az egzakt ciklikus reprezentációval rendelkezik, következésképpen kontrollpontjainak meghatározására a *TrigonometricCurve3* osztály *GenerateCyclicRepresentation* metódusát használhatjuk fel. Az öskép terében kapott

$$\mathbf{d}_i^{x_3} = \begin{bmatrix} \tilde{x}_1(-i\lambda_n) & \tilde{x}_2(-i\lambda_n) & \tilde{x}_3(-i\lambda_n) \end{bmatrix}, \quad i = 0, 1, \dots, 2n$$

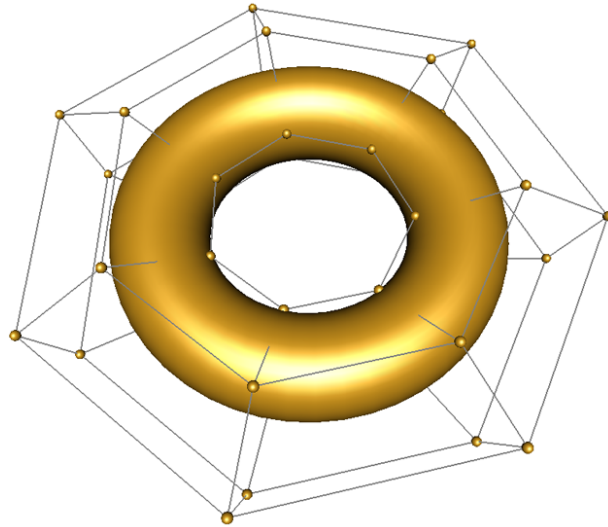
kontrollpontokat az $x_3 = 1$ hipersíkra vetítve, megkapjuk a (27) Bernoulli-féle lemniszkáta racionális trigonometrikus előállításához szükséges

$$\mathbf{d}_i = \begin{bmatrix} \frac{\tilde{x}_1(-i\lambda_n)}{\tilde{x}_3(-i\lambda_n)} & \frac{\tilde{x}_2(-i\lambda_n)}{\tilde{x}_3(-i\lambda_n)} \end{bmatrix}, \quad i = 0, 1, \dots, 2n$$

kontrollpontokat és azokhoz társított

$$w_i = \tilde{x}_3(-i\lambda_n), \quad i = 0, 1, \dots, 2n$$

súlyokat is.



4. ábra. A (28) előállítású tórusz egzakt kontrollpont-alapú leírása (2,3)-fokú ciklikus felülettel.

A 4. ábrán az

$$\begin{cases} y_1(u) = (R + r \sin u) \cos v, \\ y_2(u) = (R + r \sin u) \sin v, \\ y_3(u) = r \cos u \end{cases} \quad (u, v) \in [0, 2\pi] \times [0, 2\pi] \quad (28)$$

tórusz látható, melynek felszínét és kontrollhálóját a *CyclicSurface3* osztályt példányosítva hoztuk létre a következőképpen:

```
1 GLuint n = 2, m = 3, u_dim = 2*n+1, v_dim = 2*m+1;
2
3 // ciklikus felület létrehozása (CyclicSurface3 *_cs)
4 _cs = new CyclicSurface3(n,m);
5 // kontrollpontok beállítása
6 GLdouble lambda_n = 2 * PI / dim1,
7     lambda_m = 2 * PI / dim2, R = 3.0, r = 1.0;
```

```

8
9 for (int i = 0; i < u_dim; ++i)
10 {
11     GLdouble u = -i * lambda_n;
12     for(int j = 0; j < v_dim; ++j)
13     {
14         GLdouble v = -j * lambda_m;
15         (*_cs)(i, j)[0] = (R + r * sin(u) * 1.5) * cos(v) * 4.0 / 3.0;
16         (*_cs)(i, j)[1] = (R + r * sin(u) * 1.5) * sin(v) * 4.0 / 3.0;
17         (*_cs)(i, j)[2] = r * cos(u) * 1.5;
18     }
19 }
20
21 // kontrollháló kialakítása
22 _cs->UpdateVertexBufferObjectsOfData();
23 // a felület felszínének generálása (TriangulatedMesh3 *_i_cs)
24 _i_cs = _cs->GenerateImage(50, 50);
25 _i_cs->UpdateVertexBufferObjects();

```

Hivatkozások

- [1] Á. Róth, I. Juhász, J. Schicho, M. Hoffmann, A cyclic basis for closed curve and surface modelling, Computer Aided Geometric Design 26 (5) (2009) 528-546, doi:10.1016/j.cagd.2009.02.002.
- [2] Á. Róth, I. Juhász, Control point based exact description of a class of closed curves and surfaces, Computer Aided Geometric Design 27 (2) (2009) 179-201, doi:10.1016/j.cagd.2009.11.005.
- [3] JCAM
- [4] Juhász, I., 1995. Számítógépi geometria és grafika, Miskolci Egyetemi Kiadó.
- [5] Rost, R. J., 2006. OpenGL Shading Language, 2nd ed., Addison-Wesley.
- [6] Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis, 2006. OpenGL Programming Guide, 5th edition The Official Guide to Learning OpenGL, Version 2, Addison-Wesley.
- [7] Jasmin Blanchette, Mark Summerfield, 2006. C++ GUI Programming with Qt 4, Trolltech Press.
- [8] Bjarne Stroustrup, The C++ Programming Language, 3rd edition
- [9] <http://www.lighthouse3d.com>