Eötvös Loránd Tudományegyetem

Informatikai kar

Algoritmusok és Alkalmazásaik Tanszék

Diplomamunka

Magas fokszámú görbeszegmensek hatékony optimalizálása

Témavezető:
Valasek Gábor
adjunktus
ELTE-IK

Készítette:
Sipos Ágoston
prog. inf. MSc
ELTE-IK

Budapest, 2018

Tartalomjegyzék

1 Bevezetés

Interpolációs görbék tervezése évtizedek óta fontos téma. A geometriai Hermite-interpoláció [1] egy ismert algoritmus a probléma megoldására. A módszer által adott görbét a végpontokban adott paraméterek a megfelelő rendű folytonosság elérése érdekében megkötik, azonban emellett több szabadságfokkal is rendelkezik az illesztés. Ezek felhasználhatóak arra, hogy a görbéket bizonyos szempontok szerint minél optimálisabbra állítsuk be. Ha az illeszthető görbék fokszámát növeljük, a szabadságfokok száma jóval nagyobb lesz, ami egyrészt felhasználható még optimálisabb görbék illesztésére, másrészt viszont növeli a numerikus eljárások paraméterdimenzióját, ezáltal a műveletigényét, robusztusságukat pedig rontja.

Görbék valamilyen funkcionál szerinti optimalizálását többen vizsgálták. Moreton és Séquin [5]-ben görbék és felületek optimalizálását végezték a görbületük, illetve normálgörbületeik integrálja alapján. Schaback [7]-ben az interpolációs problémák megoldhatóságát vizsgálta polinomokkal, illetve racionális splineokkal.

Egy érdekes új megközelítés szerint érdemes lehet megvizsgálni, hogy a görbék optimális szabadságfok-értékei állnak-e valamilyen folytonos függésben a bemenő kontrolladatokkal. Amennyiben fennállnak a megfelelő feltételek, úgy ezek a paraméterek a közelükben levő végponti adatokkal rendelkező görbék paramétereinek interpolációjával becsülhetők, ezáltal közelítőleg optimális megoldások kaphatók.

Ezen leképezések közelítésére azonban szükséges az optimalizált szabadságfokok kiértékelése a bemeneti konfigurációk szerint egy sűrű rácson. Ahhoz, hogy ez magasabb fokszámú görbék esetén is megvalósítható legyen, hatékony, párhuzamos megvalósítás szükséges.

A dolgozatban megtárgyalom az erre vonatkozó módszert és implementációját, a megoldások közelítésére felállított algoritmust, illetve a kapott eredmények elemzését.

2 Elméleti háttér

2.1. Fogalmak

Egyszerű sima reguláris görbe paraméterezésének nevezem az $\mathbf{f} : I \to \mathbb{R}^3,\ f \in C^1$ leképezést, ha $\mathbf{f}$ az értékkészletén bijekció, és $\mathbf{f}'$ sehol sem nullvektor. A görbe a paraméterezés értékkészlete.

Egy paraméterezés n-edrendben folytonos (C^n-beli), ha a paraméterezése n-szer folytonosan differenciálható. Egy görbe geometriailag n-edrendben folytonos (G^n-beli), ha létezik olyan paraméterezése, amely C^n-beli.

Egy görbe *kísérő triéder*e, vagy lokális *Frenet-bázis*a egy $t \in I$ paraméterértékre az alábbi módon értelmezhető[1]:

$$\begin{aligned} \mathbf{t}(t) &= \frac{\mathbf{f}'(t)}{|\mathbf{f}'(t)|} \\ \mathbf{b}(t) &= \frac{\mathbf{f}'(t) \times \mathbf{f}^{(k)}(t)}{|\mathbf{f}'(t) \times \mathbf{f}^{(k)}(t)|} \\ \mathbf{n}(t) &= \mathbf{b}(t) \times \mathbf{t}(t) \end{aligned} \tag{2.1}$$

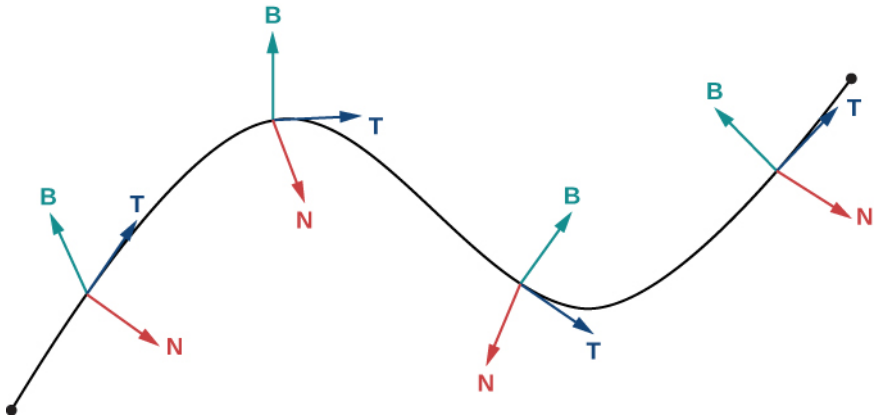
ahol $k = \min\{k \in \mathbb{N}^+ \,|\, \mathbf{f}'(t) \times \mathbf{f}^{(k)}(t) \neq \mathbf{0}\}$[2]

Ez a rendszer $\mathbb{R}^3$-ban bármely $t \in I$-re ortonormált bázist alkot, és $(\mathbf{t}, \mathbf{n}, \mathbf{b})$ sorrendben jobbkéz-rendszert. A 2.1.1 ábrán látható a Frenet-bázis egy görbe néhány pontjában.

Egy $t \in I$ paraméterértékre a görbe görbülete (κ) és torziója (τ) az alábbi mennyi-

[1] Általában $(\mathbf{t}, \mathbf{n}, \mathbf{b})$ sorrendben szokás megadni a lokális bázist, a továbbiakban én is ezt fogom követni. Az ehhez alkalmas definíció megtalálható [8]-ben, az általam alkalmazott megadás az algoritmusok szempontjából célszerűbb.

[2] $\mathbf{a} \times \mathbf{b}$ az $\mathbf{a}$ és $\mathbf{b}$ vektorok vektoriális szorzatát jelöli.



2.1.1. ábra. Frenet-bázis

ségek[3]:

$$\kappa(t) = \frac{\left\langle \mathbf{t}'(t),\ \mathbf{n}(t) \right\rangle}{|\mathbf{f}'(t)|}$$
$$\tau(t) = \frac{\left\langle \mathbf{n}'(t),\ \mathbf{b}(t) \right\rangle}{|\mathbf{f}'(t)|} \tag{2.2}$$

Ezeket a mennyiségeket *geometriai invariáns*oknak is szokás nevezni, mert függetlenek a konkrét paraméterezéstől. ([8])

A görbe Frenet-koordinátáinak nevezzük a deriváltjainak koordinátáit ebben a bázisban, azaz azokat az (x_i), (y_i), (z_i) sorozatokat, melyekre $x_i = \left\langle \mathbf{f}^{(i)}(t), \mathbf{t}(t) \right\rangle$, $y_i = \left\langle \mathbf{f}^{(i)}(t),\ \mathbf{n}(t) \right\rangle$, $z_i = \left\langle \mathbf{f}^{(i)}(t),\ \mathbf{b}(t) \right\rangle$.

A Frenet-koordináták közül $y_1 = 0$, mivel $\mathbf{f}' = \lambda \cdot \mathbf{t}$ miatt $\mathbf{f}' \perp \mathbf{n}$, illetve z_1 és z_2 is 0, mivel az első és második derivált az érintősíkban van, amire $\mathbf{b}$ merőleges.

Belátható ([10]), hogy a görbe $\kappa(t)$ görbülete, és $\tau(t)$ torziója ekkor a következő módon számítható.

$$\kappa = \frac{y_2}{x_1^2}$$
$$\tau = \frac{z_3}{x_1 y_2} \tag{2.3}$$

Ehhez hasonló módon levezethetők az invariáns mennyiségek ívhossz szerinti de-

[3] $\langle \mathbf{a}, \mathbf{b} \rangle$ az $\mathbf{a}$ és $\mathbf{b}$ vektorok skaláris szorzatát jelöli.

riváltjaira vonatkozó képletek is (levezetések: A.1):

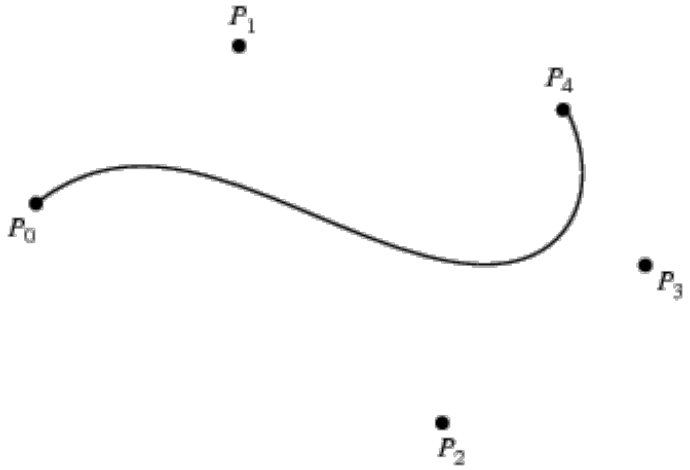
$$\begin{aligned}\kappa' &= \frac{y_3 x_1 - 3x_2^2}{x_1^4} \\ \tau' &= \frac{z_4 y_2 - 2 z_3 y_3}{x_1^2 y_2^2} \\ \kappa'' &= \frac{y_4 x_1^2 y_2 - 3 x_1 x_2 y_2 y_3 + y_2^4 + z_3^2 x_1^2}{x_1^6 y_2}\end{aligned} \tag{2.4}$$

E képletek alkalmazásával pedig olyan formulák hozhatók létre, melyek a geometriai invariánsokból, és az x_i $(i = 1..n)$ Frenet-koordinátákból előállítják a görbe $1..n$-edik deriváltjait az ($\mathbf{t}$, $\mathbf{n}$, $\mathbf{b}$) bázisban

$$\begin{aligned}\mathbf{f}' &= x_1 \mathbf{t} \\ \mathbf{f}'' &= x_2 \mathbf{t} + x_1^2 \kappa \mathbf{n} \\ \mathbf{f}''' &= x_3 \mathbf{t} + (3 x_1 x_2 \kappa + x_1^3 \kappa') \mathbf{n} + x_1^3 \kappa \tau \mathbf{b}\dot{:}\end{aligned} \tag{2.5}$$

Egy görbe Bézier-alakjának nevezzük az alábbi megadását:

Legyenek $\mathbf{b}_0$, $\mathbf{b}_1$, ..., $\mathbf{b}_n \in \mathbb{R}^3$ kontrollpontok, $B_k^n(t) = \binom{n}{k} t^k (1-t)^{n-k}$, $t \in [0,1]$ a k-adik n-edfokú Bernstein-polinom. Ekkor a görbe paraméterezése $\mathbf{f}(t) = \sum_{k=0}^{n} B_k^n(t) \mathbf{b}_k$, $t \in [0,1]$. A görbét ekkor nevezhetjük Bézier-görbének, illetve mondhatjuk úgy is, hogy a görbe a Bernstein-bázisban adott. A Bernstein-bázisban megadás igen elterjedt, mivel az így megadott görbék számítógépes kiértékelésére hatékony és numerikusan stabil algoritmusok léteznek ([2]).



2.1.2. ábra. Bézier-görbe és kontrollpontjai

2.2. Optimalizációs algoritmus

A továbbiakban optimális Geometriai Hermite interpolációs görbékről ([7], [1]) lesz szó. Ennek az interpolációs módszernek az előnye, hogy a szegmensvégpontokban G^n folytonosságot garantál, továbbá szabadságfokokkal rendelkezik, melyekkel jól paraméterezhető a feladatot megoldó görbesereg.

Az interpolálandó geometriai adatok a GH interpolációs feladatban **megkötésként** értelmezhetőek. Általános esetben ezeket az alábbi táblázat foglalja össze

rend (i)	0	1	2	3	..	n
GH adat ($\mathbf{D}_i$)	$\mathbf{p}$	$\mathbf{t}$	$\mathbf{n}$, κ	κ', τ	..	$\kappa^{(n-2)}$, $\tau^{(n-3)}$

ahol $\mathbf{p} \in \mathbb{E}^3$ rekonstruálandó pozíció, $\mathbf{t}, \mathbf{n}, \mathbf{b} \in \mathbb{R}^3$ rendre a Frenet kísérő triéder tangens, normális és binormális egységvektorai, κ, τ pedig a görbület- és torziófüggvény, illetve ezek ívhossz szerinti deriváltjai. Egy konkrét $\mathbf{D}_n$ adatvektort **geometriai konfigurációnak** nevezünk.

A megoldandó feladat a következőképpen foglalható össze:

- A bemenet kontrolladatok egy listája, ahol a kontrolladat egy pozíció, és a hozzá tartozó - folytonossági rendtől függő - egyéb adatok:
  - G^1 folytonos csatlakozás esetén a görbe iránya, azaz az $\mathbf{t}$ bázisvektor ottani értéke
  - G^2 folytonos csatlakozás esetén ezenfelül az $\mathbf{n}$ bázisvektor, és a κ görbületérték
  - G^3 folytonos csatlakozás esetén továbbá a κ' görbületi derivált, és a τ torzió (a $\mathbf{b}$ binormális $\mathbf{t}$ és $\mathbf{n}$ alapján már meghatározható)
- További bemenet egy $C^n(I) \to \mathbb{R}$ funkcionál, amelyre nézve optimalizálást végzünk. Egy egyszerű példa: $\int_I \kappa^2$, azaz a görbület kettes normája.
- Az algoritmus kimenete egy olyan spline görbe, melyre teljesül, hogy a szakaszok határpontjaiban interpolálja a bemenő adatokat, adott n-re G^n folytonos, és a funkcionál értéke rajta minimális.

Ha a feladatot csak a spline egy szakaszára tekintjük, akkor ismert, hogy az n-edrendű két végponti illesztési feladatnak mindig létezik $2n+1$ fokszámú polinomiális görbével megoldása ([7]). Kisebb fokú csak akkor létezik, ha a kontrolladatok teljesítenek bizonyos feltételeket. Magasabb fokú viszont mindig létezik, a dolgozat egyik tárgya lesz, hogy magasabb fokszámmal elérhető-e, hogy a görbe optimálisabb legyen.

Az optimalizálási feladat dimenziója ekkor a következő módon áll elő:

- A görbe paramétereinek száma $2n+1$-fokú esetben $3(2n+2)$, mivel a Bézier-megadásban $2n+2$ kontrollpontot kell beállítani, és mindegyik 3 valós értékkel reprezentálható.
- A kontrolladatok viszont ebből megkötnek $2n+3$ szabadságfokot végpontonként (a végpont pozíciója 3, a további rendek 2-2 adatot hordoznak)
- A fokszám minden emelésével egy újabb kontrollpont keletkezik, ami 3-mal növeli a szabadságfokok számát
- A beállítható szabadságfokok száma tehát $3(2n+2)-2(2n+3)=2n$, továbbá ha $d>2n+1$ fokú görbét illesztünk, akkor ezen felül még $3(d-(2n+1))$ az extra kontrollpontokból.

Ezek a szabadságfokok jól reprezentálhatóak a két végpontban levő $x_1, ..., x_n$ Frenet-koordinátákkal, továbbá magasabb fokú esetben az első és utolsó $n+1$ darab kontrollpont közé eső pontok 3-3 koordinátájával ([6]). Ezekből a 2.5 formulákkal megkapjuk a görbe első n deriváltját a két végpontban $(\mathbf{f}(0), \mathbf{f}'(0), ..., \mathbf{f}^{(n)}(0)$, illetve $\mathbf{f}(1), \mathbf{f}'(1), ..., \mathbf{f}^{(n)}(1))$, amivel egy Hermite-féle interpolációs feladathoz jutottunk.

Valamilyen általános (nem feltétlenül polinomiális) bázisban ($F_i : [0,1] \to \mathbb{R}^3$, $i \in \{0,1,...,d\}$) a Hermite-interpoláció megoldható az alábbi mátrixegyenlettel ([2]):

$$\begin{bmatrix} \mathbf{f}(0) \\ \mathbf{f}'(0) \\ \mathbf{f}''(0) \\ \vdots \\ \mathbf{f}^{(n)}(0) \\ \mathbf{f}(1) \\ \mathbf{f}'(1) \\ \mathbf{f}''(1) \\ \vdots \\ \mathbf{f}^{(n)}(1) \end{bmatrix} = \underbrace{\begin{bmatrix} F_0(0) & F_1(0) & ... & F_d(0) \\ F_0'(0) & F_1'(0) & ... & F_d'(0) \\ F_0''(0) & F_1''(0) & ... & F_d''(0) \\ \vdots & \vdots & \ddots & \vdots \\ F_0^{(n)}(0) & F_1^{(n)}(0) & ... & F_d^{(n)}(0) \\ F_0(1) & F_1(1) & ... & F_d(1) \\ F_0'(1) & F_1'(1) & ... & F_d'(1) \\ F_0''(1) & F_1''(1) & ... & F_d''(1) \\ \vdots & \vdots & \ddots & \vdots \\ F_0^{(n)}(1) & F_1^{(n)}(1) & ... & F_d^{(n)}(1) \end{bmatrix}}_{F \in \mathbb{R}^{(2n+2)\times(d+1)}} \cdot \underbrace{\begin{bmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \vdots \\ \mathbf{p}_d \end{bmatrix}}_{\mathbf{p}} \tag{2.6}$$

Ezt speciálisan a Bernstein-bázisra alkalmazva, egy egyenletet kapunk a $\mathbf{p}_0, ..., \mathbf{p}_d$ kontrollpontokra. Megfigyelhető, hogy az egyenlet mátrixa csak a bázistól függ (a konkrét bemenő adatoktól nem), tehát tárolható, sőt inverze is. A rendszer a $d>2n+1$ esetben (d a szegmens fokszáma, n a folytonosság rendje) alulhatározott lesz, ami épp azt jelenti, hogy további szabadságfokok is vannak. Ekkor a mátrix középső néhány oszlopa nullvektor lesz ([2]), ami épp azt jelenti, hogy ezek a végponti

deriváltak nem függnek a középen levő kontrollpontoktól, azok a korábban írtak szerint extra szabadságfokokként értelmezhetők.

A kontrollpontok birtokában kiértékelhető a görbe tetszőleges tulajdonsága bármilyen paraméterértékre, illetve bármely invariáns mennyiség a 2.3-2.4 formulák szerint. Így az ezekből, és az integrálás operátorból összeállítható célfüggvények kiértékelhetők (numerikus integrálással).

Mivel a szabadságfokok két jól elkülönülő csoportot alkotnak, melyek közül az egyik Frenet-koordinátákat, a másik viszont pozícióadatokat tartalmaz, célszerű - a fejlesztés során szerzett tapasztalatokat is figyelembe véve - a [11] által alkalmazott módszerhez hasonlóan kettéválasztani az optimalizációkat. Ez azt jelenti, hogy először optimalizálok egy alacsonyabb fokú görbét, amely csak a végponti szabadságfokokkal paramétereződik, majd ennek fokszámemelésével ([2]) kapott görbe belső kontrollpontjait optimalizálom tovább.

Az optimalizálás L-BFGS algoritmussal történik ([4]). A célfüggvény deriváltjait numerikusan, szimmetrikus differenciahányadosokkal becslem, azaz:

$$\frac{\partial f}{\partial x_i}(x_i^{(0)}) = \frac{f(..., x_i^{(0)} + \varepsilon, ...) - f(..., x_i^{(0)} - \varepsilon, ...)}{2\varepsilon} \tag{2.7}$$

Az eljárást iterálva, kapjuk kontrolladatok egy sorozatát, és a hozzájuk tartozó célfüggvény-értékeket. Annak érdekében, hogy egy hibás lépés ne rontsa el a végeredményt (például ha az első deriváltnak többszörös gyöke van, ennek közelében instabilitás léphet fel 0 közeli számmal való osztás miatt), az iteráció végén egy minimumkiválasztás megkeresi a kapott legkisebb normájú görbét.

A Newton-jellegű iterációs módszerek kezdőértékre való érzékenysége miatt több kezdőpontból is indítok iterációkat, majd az eredmények között szintén egy minimumkiválasztás dönt.

2.3. Közelítő algoritmusok

Feltételezve, hogy a numerikus optimalizáció eredményre jutott, az optimumként kapott Frenet-koordináták tekinthetők a konfigurációhoz tartozó függvényértéknek egyfajta ($\mathbf{D}_n \times \mathbf{D}_n \to \mathbb{R}^{2n}$ alakú) „optimumleképezésének” eredményeképp. Ennek az optimumleképezésnek - vagy legalábbis egy közelítésének - az ismeretében az optimális szabadságfokok egy numerikus algoritmus végrehajtása nélkül, akár konstans időben is megkaphatóak.

Algoritmus 2.2.1 Optimalizált görbeszegmens illesztése

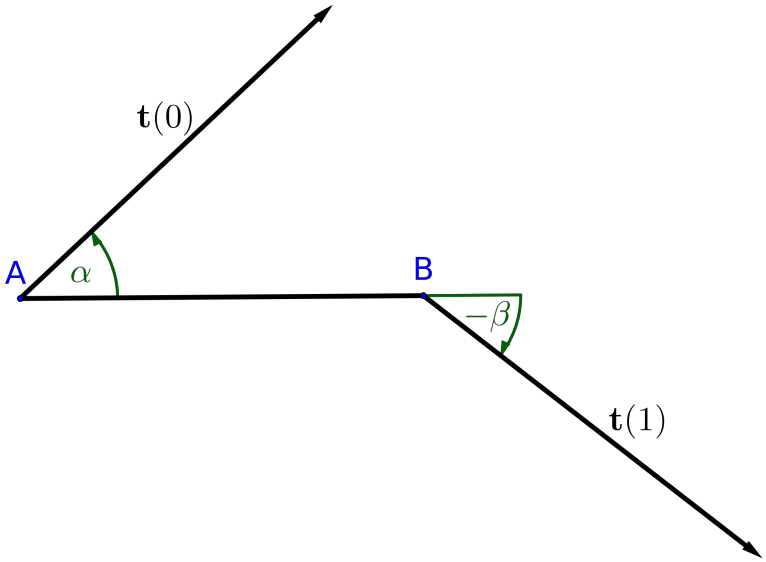
Bemenet: p1, p2 végponti adatvektorok
Kezdőértékek beállítása a végponti szabadságfokokra, és a belső kontrollpontokra
Ciklus
    Végponti szabadságfokok optimalizálása
    Ciklus
        Kontrollpontok generálása a differenciahányados meghatározásához szükséges szabadságfokokra
        Integrálnorma kiértékelése minden generált esetben
        Parciális deriváltak becslése
        Lépésszámítás L-BFGS módszerrel és lépés (A.2)
    Ciklus vége
    Ha kis elmozdulás (és nem első iteráció): kilépés
    Fokszámemelés
    Belső kontrollpontok optimalizálása:
    Ciklus
        Kontrollpontok generálása a differenciahányados meghatározásához szükséges szabadságfokokra
        Integrálnorma kiértékelése minden generált esetben
        Parciális deriváltak becslése
        Lépésszámítás L-BFGS módszerrel és lépés (A.2)
    Ciklus vége
    Ha kis elmozdulás: kilépés
Ciklus vége
Szabadságfokok alapján kontrollpontok generálása
Kimenet: Bézier-görbe kontrollpontokkal

A gyakorlatban ezt a közelítést egy rácson fogom megadni, azaz egy minél sűrűbb felosztáson kiértékelem az optimális paramétereket. Amennyiben ezek adottak, tetszőleges konfigurációra közelíthetőek ez alapján a paraméterek.

Annak érdekében, hogy a rácsok minél kevesebb dimenziósak legyenek, a konfigurációkat normalizáljuk.

Elsőrendű esetben a síkbeli konfigurációkat két szöggel ($\alpha, \beta \in [0, 2\pi]$) paraméterezzük a 2.3.1 ábrán látható módon. Ehhez könnyen belátható, hogy létezik olyan invertálható transzformáció, ami általános bemeteti konfigurációból a kezdőpontot (0,0)-ba, a végpontot (1,0)-ba transzformálja. Ez felbontható egy eltolási, forgatási és izotróp skálázási komponensre. Visszatranszformáláskor az inverz transzformáció skálázási komponensével kell szorozni az x_i Frenet koordinátákat (a tangens és minden derivált ugyanazon a forgatáson esik át, tehát ez a skaláris szorzatokat nem módosítja, a deriváltak hossza viszont skálázódik).

A másodrendű konfigurációkat négy skalár mennyiséggel tudjuk leírni a 2.3.2 ábrán látható módon. A két végpontot, illetve a tangenseket az elsőrendű esethez hasonlóan írjuk le, a normális egyszerűen a tangens $+90°$-os elforgatottja, míg a



2.3.1. ábra. Elsőrendű normalizált konfiguráció

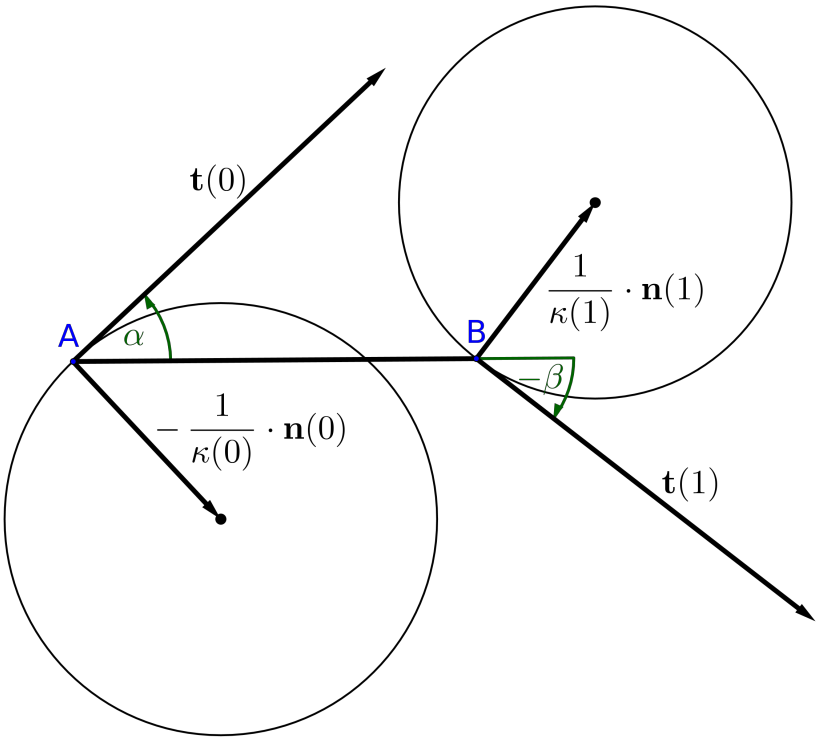
görbület ekkor pozitív és negatív egyaránt lehet (így jellemezhetők azok az esetek, amikor a normális az ellenkező irányba mutat).

Általános konfiguráció normalizálása esetén - az elsőrendű esetnél már leírtakon kívül - arra kell figyelni, hogy a görbület a skálázási komponens inverzével szorzódik (mivel a simulókör sugara skálázódik, de a görbület ennek a reciproka).

Ezek a konfigurációk jól paraméterezhetők tehát az elsőrendű esethez hasonló α és β szögekkel $[0, 2\pi]$-ből, illetve a $\kappa(0)$, $\kappa(1)$ valós görbületi értékekkel. Ezzel a kiértékelési rács 4 dimenziós lesz.

Mivel elsőrendű esetben 2, másodrendűben 4 szabadságfoka van a GH interpolációnak, ezért rendre egy $\mathbb{R}^2 \to \mathbb{R}^2$ illetve egy $\mathbb{R}^4 \to \mathbb{R}^4$ függvényt kell ráccsal közelítenünk.

Az optimumbecslésekhez ezt a rácsot használjuk, legközelebbi szomszéd, bilineáris, köbös spline, illetve szakaszos Hermite spline szerinti közelítésekre a rácspontok közti bemenetek esetén. (Az utóbbi kettő a Matlab *'spline'* illetve *'pchip'* beépített függvénye.) A becslő megadja a becsült szabadságfok-értékeket, illetve egy becsült értéket a görbe funkcionál szerinti értékére.



2.3.2. ábra. Másodrendű normalizált konfiguráció

3 Implementáció

A megvalósítás egy, az Algoritmusok alkalmazásai labor kereteiben készült, görbék kezelésére alkalmas program kibővítésével készült. A kód publikusan elérhető, letöltéséhez, és használatához segítség található a B mellékletben. A görbék kezelése tehát közös munka, az optimalizáció és vizualizáció pedig saját.

A program C++ nyelven íródott, a GPU-n futó kódon GLSL-ben, kisebb részek (kódgenerálás, konstansok előre kiszámítása) scriptnyelveken készültek.

3.1. Görbeműveletek

A görbékkel kapcsolatos könyvtár egy C++ és egy OpenGL implementációból áll. A görbék implementációja jelenleg Bézier-görbékkel történik, de létezik egy általános interfész (mely a görbe és deriváltjainak kiértékelését várja el egy paraméterértékben), melyet használva más implementáció is adható a görbékhez.

A Bézier-implementáció [2] alapján készült. Végrehajthatók benne a leggyakrabban alkalmazott Bézier-görbékkel kapcsolatos műveletek: kiértékelés (Bernstein-együtthatókkal és de Casteljau-algoritmussal), deriváltak kiértékelése, görbék felosztása, fokszámnövelése, stb.

A könyvtár képes továbbá a 2.1-ben leírt geometriai invariánsok kiértékelésére, az ott leírt módszerekkel, azaz a Frenet-koordináták segítségével.

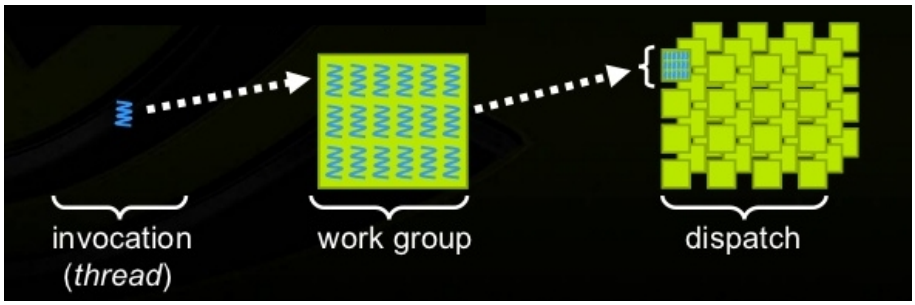
Ezeken kívül a C++ könyvtár tartalmaz még néhány példagörbét, illetve egy egyenletes eloszlású görbék véletlenszerű generálására alkalmas alprogramot (teszteléshez).

Az OpenGL-implementáció GLSL kódokban található, melyek hívhatók például a görbék kiértékelését végző *tesselation shader*ekből, de ugyanúgy az optimalizációt végző kódból is.

3.2. Optimalizáció

Az optimalizáció megvalósítása OpenGL (4.3) Compute Shaderekkel ([9]) történt, ami egy általános célú GPU programozási API. Ez a környezet jól illeszthető a szintén OpenGL-ben elkészült megjelenítőhöz, valamint a korábban leírt, görbéken végezhető függvényekhez, amiket így nem kellett többszörösen implementálni. A compute shaderek kényelmes támogatást nyújtanak a párhuzamos számításokhoz, így az algoritmusok jelentősen gyorsíthatók.

A compute shaderek ún. hierarchikus állapottéren dolgoznak, azaz az indításukkor (*dispatch*) a felhasználó egy vagy több munkacsoportot (*workgroup*) indít el, amik egy-egy feladaton önállóan dolgoznak. A workgroup-ok szálakra (*thread* vagy *invocation*) oszlanak, amik párhuzamosan tudnak dolgozni a feladaton. Egy groupon belül a szálak egymással osztott memóriát használhatnak, illetve szinkronizálhatók egymással. A csoportok és a szálak azonosítására is 3 dimenziós koordináták használhatók, amivel egy absztrakt tér elemeiként hivatkozhatunk a csoportokra, illetve az egy csoporton belüli szálakra.



3.2.1. ábra. OpenGL Compute Shaderek szerkezete
Forrás: `https://www.slideshare.net/Mark_Kilgard/siggraph-2012-nvidia-opengl-for-2012`

A compute shaderek be- és kimenetét a GPU memóriájában allokált bufferekbe (pl. SSBO) kell betölteni, illetve onnan kiolvasni. Ennek megvan az az előnye, hogy például egy compute shader kimenetét egy rajzolási parancs bemeneteként használhatjuk, a felesleges adatmozgatást elkerülve.

Az algoritmus bemenete a szegmensvégpontok a megfelelő rendű folytonossági adatokkal (2.2). Ezt lebegőpontos számokkal, és - a shadernyelvben beépített típusként, míg C++-ban a GLM könyvtárban definiált - 3 dimenziós vektor típussal tárolom. Minden egyes görbeszegmensre egy-egy workgroup-ot indítunk, amik tehát két-két végpontot fognak feldolgozni.

Kimenetként az eredményül kapott görbeszegmensek Bézier-kontrollpontjait adjuk meg, itt is workgroup-onként egy görbe adatait. Ez fokszám+1 darab 3 dimenzi-

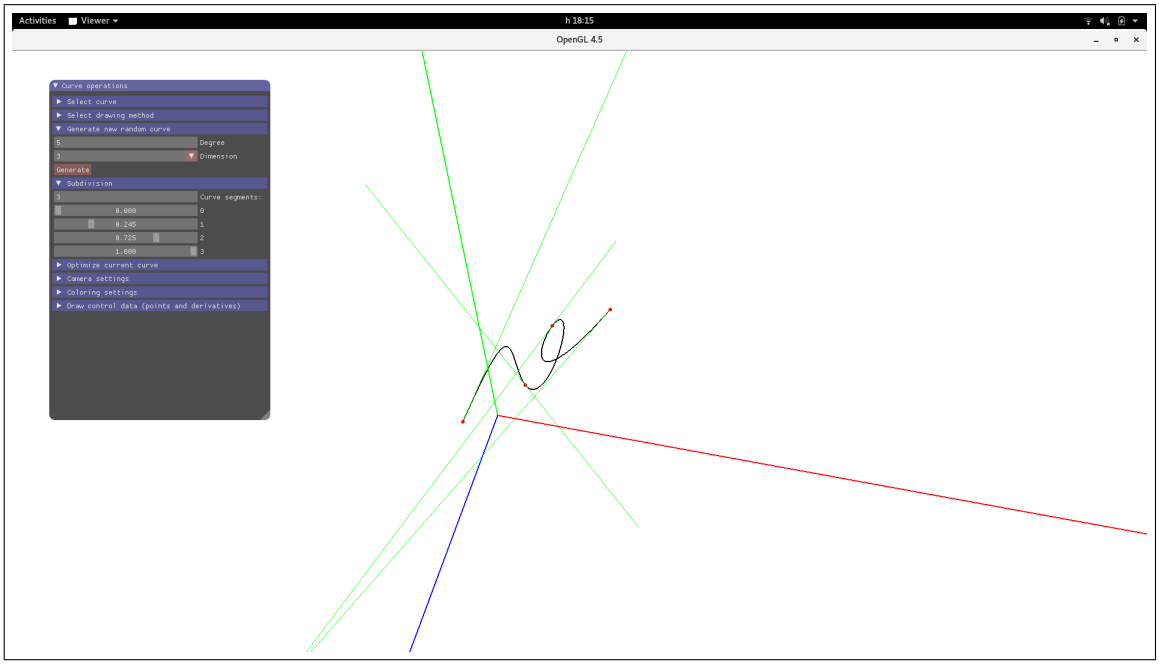
ós vektort jelent. Ezenkívül visszaadja a program az eredménygörbe optimalizálási funkcionál szerinti normáját, hogy összehasonlítható legyen más görbékkel (illetve ellenőrizhető legyen, hogy az optimalizáló helyesen működik-e, C++-ban írt validációs számításokkal).

Az implementáció alapvetően a 2.2 alapján történt. A tárolható adatokat: Hermite-interpoláció egyenletrendszerének mátrixa, kvadratúra-alappontok és együtthatók, a shaderekben globális konstansokban tárolom, miután előzőleg Matlab scriptekkel meghatároztam.

3.3. Kezelőfelület

3.3.1. Grafikus felület

A megjelenítőprogram képes egy térben görbék megjelenítésére, amit egy ablakban szabad kamerával bejárhatunk. A program OpenGL-lel implementált, a görbék kiértékelését *tesselation shader*ek végzik, az ablakkezelés az SDL könyvtárral készült. Ezenkívül egy, az ImGui könyvtár segítségével implementált grafikus interfészen tud a felhasználó inputot bevinni, beállításokat módosítani. A kezelőfelület angol nyelvű.



3.3.1. ábra. A program működés közben

A program egy görbét, vagy az egy görbe felosztásával és optimalizálásával kapott szegmenseket tud megjeleníteni. Láthatók ezenfelül a koordinátatengelyek (tájékozódás céljából, rendre vörös, zöld és kék színnel az X, Y, Z tengelyek), valamint

bizonyos segédadatok (erről a megfelelő beállításnál lesz szó). A végezhető operációk a következők:

- Görbe választása („Select curve”): Kiválaszthatunk egy görbét további operációkra és megjelenítésre. Választhatunk előre tárolt példagörbék vagy véletlenszerűen generált görbe közül (lásd később)
- Rajzolási mód („Select drawing method”): Megválaszthatjuk, hogy szeretnénk-e GPU gyorsított tesszelációt alkalmazni a megjelenítéskor. Alacsonyabb számítási teljesítményű videókártyákon érdemes lehet kikapcsolni, ekkor a C++ program generálja a görbe szakaszokkal való közelítését.
- Görbe generálása („Generate new random curve”): Új véletlenszerű görbe generálására kerül sor. Megadható a görbe fokszáma, illetve, hogy sík vagy térgörbét szeretnénk-e kapni. A görbét nem a Bernstein-, hanem a Legendre-bázisban generálja, mivel az ortogonális, ezért jobban teljesíti az egyenletes eloszlás kritériumait. Ezt [3] által leírt módon konvertálja Bernstein-bázisba.
- Felosztás („Subdivision”): Megadható, hogy hány szegmensre szeretnénk felosztani a görbét, majd a megjelenő csúszkákkal be lehet állítani a felosztási pontok helyét. Ha szeretnénk látni ezen pontok helyét, be kell kapcsolni az utolsó opcióban a kontrolladatok megjelenítését.
- Optimalizáció („Optimize current curve”): Optimalizál egy görbét a megadott felosztási pontokra. Kiválasztható, hogy milyen rendű folytonosságot szeretnénk elérni, milyen fokszámú görbét használnánk, és milyen funkcionál szerint szeretnénk optimalizálni.
- Kamerabeállítások („Camera settings”): Beállítható, hogy milyen kamerával szeretnénk a világot nézni, a szabadon mozgó perspektív kamera helyett választható az XY síkra néző ortografikus projekció is (ez elsősorban síkgörbékhez hasznos).
- Színezési beállítások („Coloring settings”): Az R, G és B színkomponensekhez hozzárendelhetők különböző funkcionálok, amiknek az értékét a görbe színezésére használjuk fel. Alapesetben a színek 0 és 1 közötti lebegőpontos számokkal reprezentálhatók, de a kapott érték egy skálázási paraméterrel módosítható, hogy a színváltozás minél jobban leírja a görbe tulajdonságait.
- Kontrolladatok megjelenítése („Draw control data”): Megjeleníti a felosztási pontokat (piros ponttal), és az ottani derivált irányát és hosszát (zöld vonallal). Optimalizálás előtt az eredeti görbe, utána a kapott görbe adataival dolgozik (ez csak a deriválthosszt módosíthatja).

Szabad kamera esetén a világot a w, a, s, d billentyűkkel lehet bejárni, míg a bal egérgomb lenyomásával, és az egér mozgatásával a kamerát forgatni. Ha az XY síkra

rögzített kamerát választjuk, az egér görgőjével lehet nagyítani/kicsinyíteni a képet, míg a w, a, s, d gombokkal az adott irányba eltolni a nézetet.

3.3.2. Parancssori felület

A program parancssori felületen keresztül is kezelhető, ahol futtathatók nagy számú görbére párhuzamosan az optimalizációs eljárások. A parancssori felület az eredménygörbék paramétereit a képernyőn írja ki, vagy egy szövegfájlba irányítja.

Itt vannak megvalósítva azok az eljárások, amelyek a görbék paramétereit a konfigurációk egy sűrű rácsán kiértékelik. Ezeket szövegfájlba exportálva, a fájlok felhasználásával futtathatók a következőekben leírt közelítő módszerek.

3.4. Közelítő módszerek

A közelítő módszerek Matlab-ban vannak implementálva. A GPU implementációval generált rácsot fájlból beolvasva a 2.3-ban leírt módokon közelíti a bemenő konfigurációk paramétereit. A kapott eredményeket szintén fájlba exportálja, amit ezután össze lehet vetni az ugyanezen konfigurációkra kiszámított numerikus optimalizációkkal.

4 Eredmények

A leggyakrabban használt funkcionál a

$$C : (I \to \mathbb{R}^3) \to \mathbb{R} \qquad C\mathbf{f} = \int_0^1 \langle \mathbf{f}'(x), \mathbf{f}''(x) \rangle dx \tag{4.1}$$

Ez a függvény ívhossz szerinti paraméterezés esetén ismert, hogy konstans 0 (mivel a két derivált ekkor merőleges egymásra). Minimalizálásával közelítően konstans sebességű görbék konstruálhatóak.

Egy másik gyakran használt funkcionál a

$$K\mathbf{f} = \int_0^1 \kappa_{\mathbf{f}}(x) dx \tag{4.2}$$

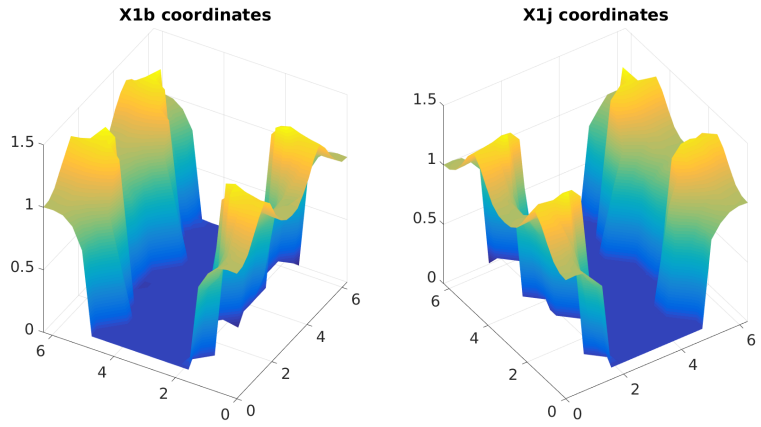
A továbbiakban az ezen két funkcionál szerint optimális polinomiális görbéket fogom vizsgálni első-, és másodrendű illesztési feladatokban.

4.1. Elsőrendű illesztés

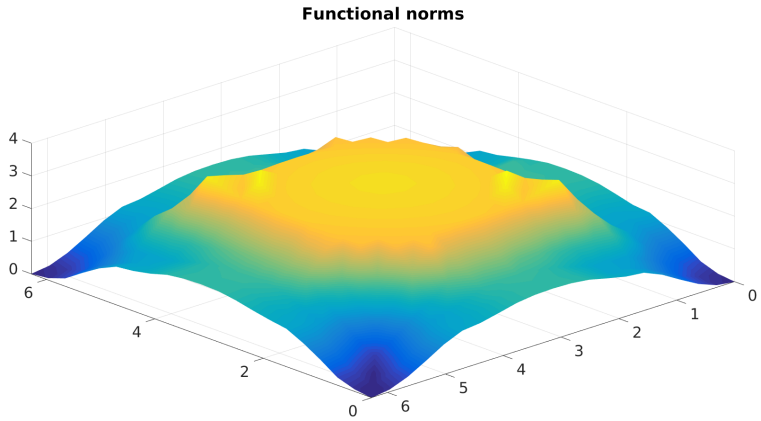
4.1.1. Harmadfokú eset

Elsőrendű esetben egy konfigurációra egy kétparaméteres, harmadfokú görbesereg illeszthető. A szabadságfokok, a $\overrightarrow{x}_1, \overleftarrow{x}_1 > 0$ skalárok a két végpontbeli deriváltak hosszát adják meg, azaz a kezdeti és a végső parametrikus sebességet. Ezáltal az optimumleképezés $\mathbb{R}^2 \to \mathbb{R}^2$ alakú lesz, így jól elemezhető és vizualizálható. 20×20-as rácson kiértékelve a 4.1.1. ábrán látható approximációját kapjuk.

Fontos megjegyezni, hogy itt mindenképpen feltételes optimalizációt kell végezni, mivel az $\overleftarrow{x_1}$ és $\overrightarrow{x_1}$ nem lehetnek negatívak. (Negatív érték esetén nem a pontbeli tangenst, hanem éppen az ellentettjét.) Sőt, ha valamelyik 0, akkor is egy nem reguláris görbét kapunk, ezért alsó korlátként 0.1-et adtunk meg. Ez okozza a nagy lapos területet a két optimumleképezésben. Továbbá azokon a területeken, ahol mindkét



4.1.1. ábra. Optimális $\overrightarrow{x}_1$ és $\overleftarrow{x}_1$ paraméterek az α, β paraméterek szerint.



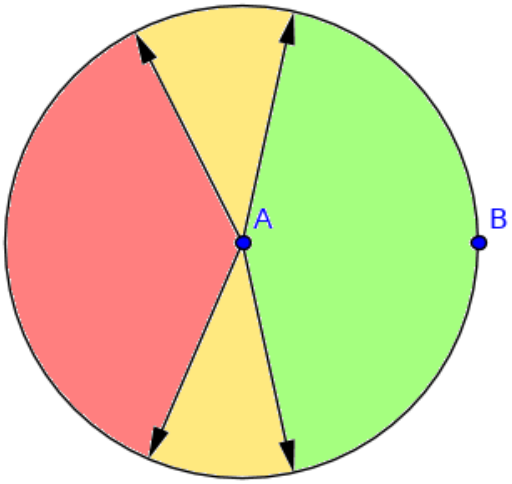
4.1.2. ábra. Optimalizált görbe (4.1) szerinti normája az α, β paraméterek függvényében.

paraméter optimalizálása a korlátba ütközött lesz a legnagyobb a célfüggvény értéke (4.1.2. ábra), hiszen ekkor a numerikus módszer nem konvergált egy reguláris görbéhez.

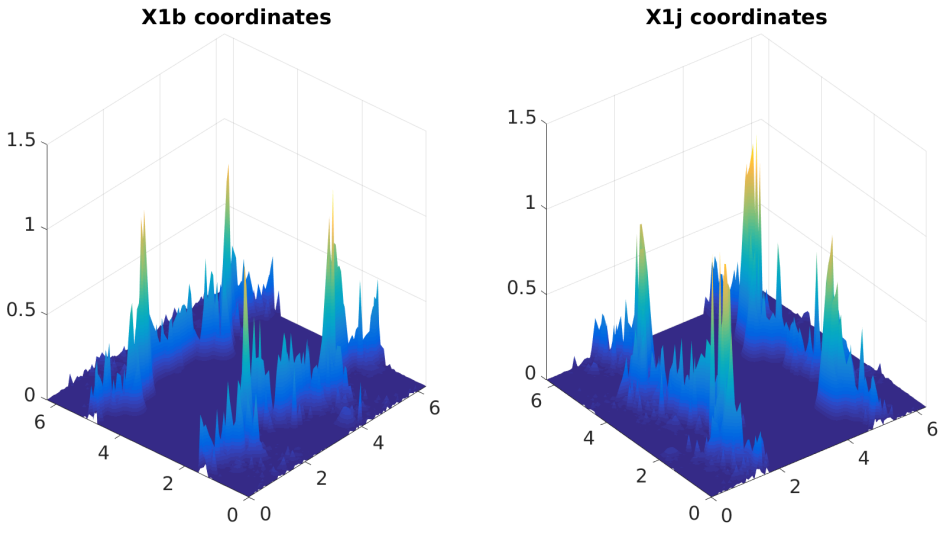
A 4.1.3. ábrán megtekinthető, hogy melyek azok a konfigurációk, melyekre a numerikus optimalizálás a 4.1 funkcionál szerint nemdegenerált eredményhez konvergált. Az A pontból B pontba mutató görbe optimalizálása során $\overrightarrow{x}_1$-ben akkor kapunk optimális reguláris görbét, ha az A-beli tangens az egységkör zölddel jelölt részében található. Azaz amikor a végponti tangens a másik végpont irányához képest ellentétes irányba mutat, nem lesz optimális az illesztés.

Az 4.1.4. ábrán látható, hogy a közelítő algoritmus a tartomány nagy részén pontos, néhány esetben viszont nagy különbség van. Ezek a nagy eltérések éppen a 4.1.3. ábrán sárgával jelölt tartományba esnek. Ezek a konfigurációk tehát speciális figyelmet igényelnek.

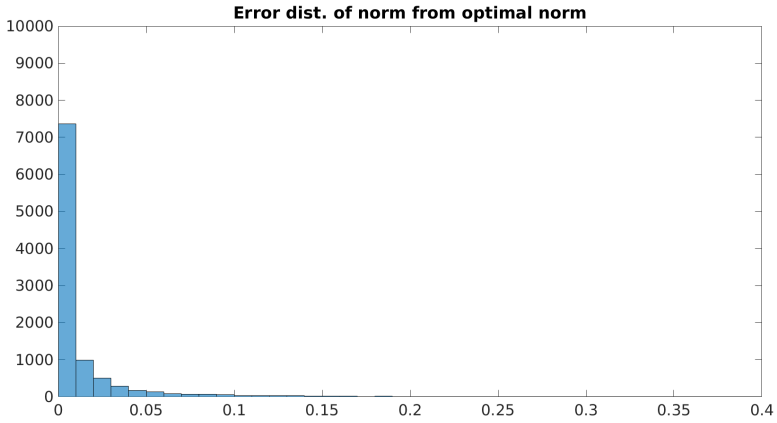
Az alábbi táblázatban láthatók a módszerek átlagos relatív hibaértékei, összevet-



4.1.3. ábra. Tangens-konfigurációk a feltételes optimalizálás konvergenciája szerint. Piros tartomány: nem konvergál, sárga tartomány: az esetek legalább 10%-ában nem konvergál, zöld tartomány: konvergál.



4.1.4. ábra. Becslő algoritmus hibája: a 100×100-as rács, és a felülmintavételezett 20×20-as rács különbsége.



4.1.5. ábra. Szakaszos Hermite-spline szűrés optimalitási hibájának eloszlása.

ve az ugyanazokból a bemenő adatokból kiinduló optimalizációval. (10000 egyenletes eloszlású teszteset generálásából.)

$\overrightarrow{x}_1$ becslésének hiba statisztikái a következők (vastagon szerepel a legjobb):

Módszer	Átlag	Szórás	Medián
Legközelebbi	0.0645	0.1793	0.0018
Bilineáris	0.0658	0.1663	0.0025
Köbös spline	0.1196	0.2404	0.0187
Szakaszos Hermite	**0.0588**	**0.1602**	**0.0017**

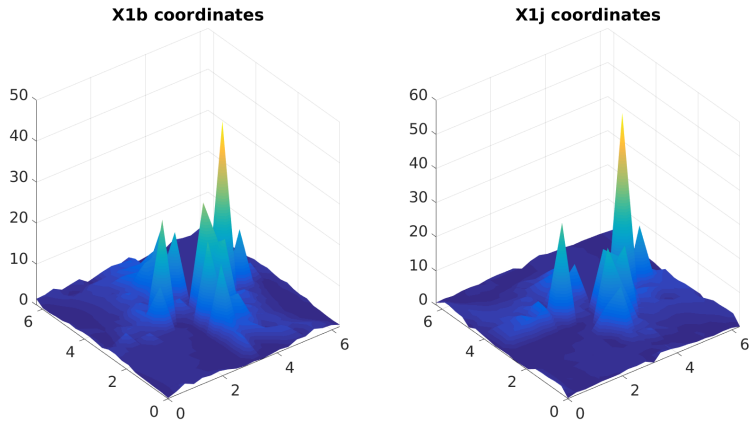
A becsült görbék tényleges célfüggvény-értékének eltérése az optimálisétól:

Módszer	Átlag	Szórás	Medián
Legközelebbi	0.0250	0.0539	0.0086
Bilineáris	0.0146	0.0384	0.0031
Köbös spline	0.0196	0.0471	0.0059
Szakaszos Hermite	**0.0135**	**0.0365**	**0.0022**

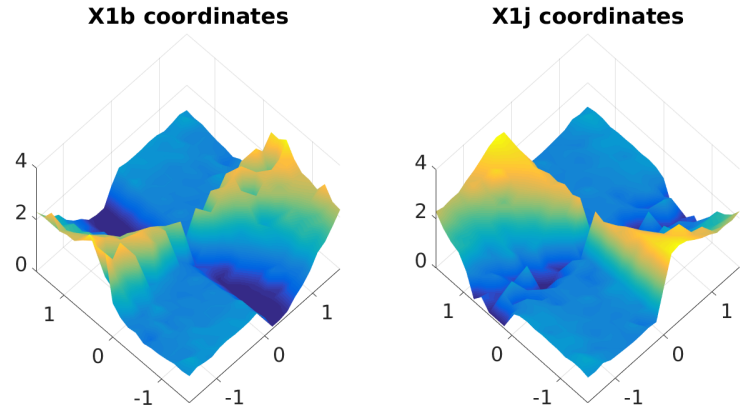
Levonható a következtetés, hogy a szakaszos Hermite-spline approximáció a legpontosabb, de a lineáris nem sokkal marad el, és lényegesen alacsonyabb számításigénye miatt jó alternatíva lehet.

A hibák eloszlása szakaszos Hermite esetén a 4.1.5. ábrán látható. A hiba javarészt kicsiny, kevés kiugró magas érték van. A többi módszer hibájának eloszlása is ehhez hasonló struktúrájú. Ritkán (<0.1%) előfordulnak nagy hibák, melyek az átlagot felhúzzák.

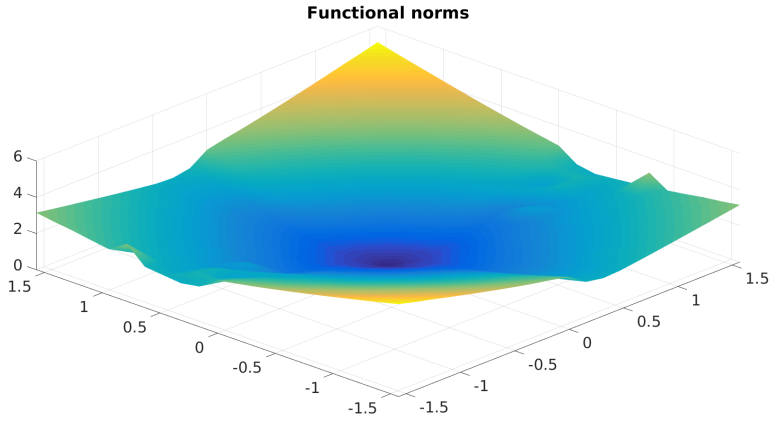
A 4.2 funkcionál alkalmazása esetén az optimumleképezés más struktúrájú lesz. Ennek az optimumleképezésnek a $[0, 2\pi] \times [0, 2\pi]$-n szingularitásai vannak (4.1.6 ábra), ezért célszerű a $[-\frac{\pi}{2}; \frac{\pi}{2}]$-re vett leszűkítését vizsgálni (4.1.7-4.1.8 ábrák).



4.1.6. ábra. Optimumleképezések a (4.2) funkcionál szerint



4.1.7. ábra. Optimumleképezések a (4.2) funkcionál szerint (leszűkítve $[-\frac{\pi}{2};\frac{\pi}{2}]$-re)



4.1.8. ábra. Optimális görbe normája a (4.2) funkcionál szerint (leszűkítve $[-\frac{\pi}{2};\frac{\pi}{2}]$-re)

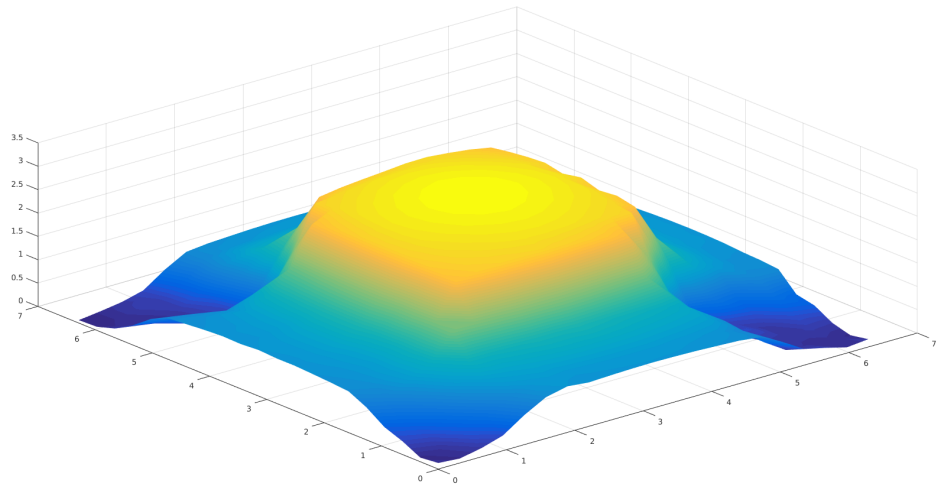
Ezen a tartományon generált tesztesetekre a (4.1)-es funkcionálhoz hasonló pontosságú becslés adható például szakaszos Hermite-approximációval:

Paraméter	Átlag	Szórás	Medián
$\overrightarrow{x}_1$	0.1362	0.1680	0.0729
Célfüggvény	0.0148	0.0376	0.0021

4.1.2. Magasabb fokú eset

Harmadfokúnál nagyobb görbefokszám esetén a köztes (azaz az első kettőn, és az utolsó kettőn kívüli) Bézier-kontrollpontok önmagukban szabadságfoknak tekinthetők, mivel nem kötik őket a végponti $\overrightarrow{x}_1, \overleftarrow{x}_1$ értékek.

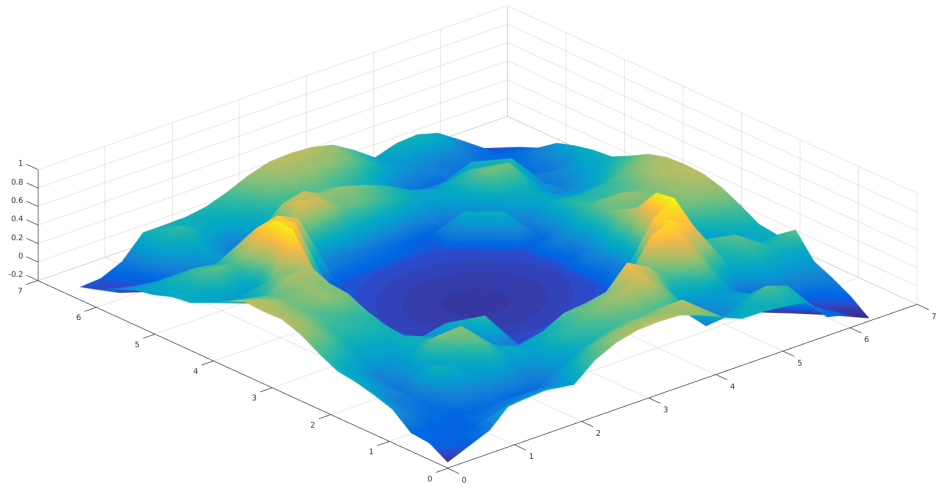
Érdemes először megvizsgálni, hogy milyen eltérések vannak azonos konfigurációra illesztett harmad-, illetve negyed- vagy magasabb fokú görbék célfüggvényértékében. Ezzel jobban megismerhetjük, hogy melyek azok a konfigurációk, ahol a fokszámemelés hasznosabb.



4.1.9. ábra. Optimális negyedfokú görbe normája 4.1 szerint

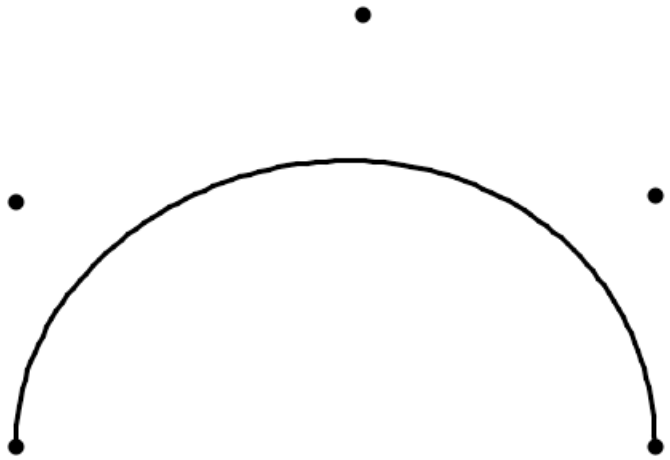
A 4.1.9 ábrán látható az optimális negyedfokú görbe normája 4.1 szerint. Ezt a harmadfokúval összevetve a 4.1.10 ábrán látható a fokszámemelés által elérhető javítás.

Egy példa olyan konfigurációra, ahol nagy jelentősége van a fokszámnak, látható a 4.1.11-4.1.12 ábrákon. A harmad- és negyedfokú Bézier görbék, valamint a kontrollpontjaik láthatók az $\alpha = \frac{\pi}{2}, \beta = -\frac{\pi}{2}$ esetben.



4.1.10. ábra. Negyedfokú görbe javítása a harmadfokúhoz képest

4.1.11. ábra. Harmadfokú optimális görbe. $C\mathbf{f} = 1.78817749$



4.1.12. ábra. Negyedfokú optimális görbe. $C\mathbf{f} = 0.957862$

4.2. Másodrendű illesztés

4.2.1. Ötödfokú eset

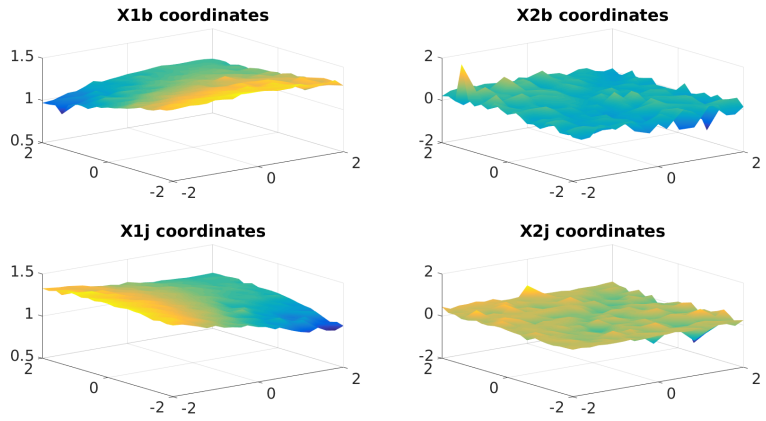
Az elsőrendű esetnél szerzett tapasztalatok alapján az α, β paramétereket a $[-\frac{\pi}{2}; \frac{\pi}{2}]$ intervallumba korlátoztuk. A $\kappa(0)$, $\kappa(1)$-t praktikussági szempontok szerint a $[-2; 2]$ intervallumon mintavételeztük.

Az illesztett görbék ekkor ötödfokúak lesznek, 4 szabadságfokkal ($\overrightarrow{x}_1$, $\overrightarrow{x}_2$, $\overleftarrow{x}_1$, $\overleftarrow{x}_2$), az optimumleképezés pedig $\mathbb{R}^4 \to \mathbb{R}^4$ alakú. Ennek vizualizációja csak metszetekkel lehetséges, például rögzítve a két szöget, megadható a szabadságfokok függése a görbületektől.

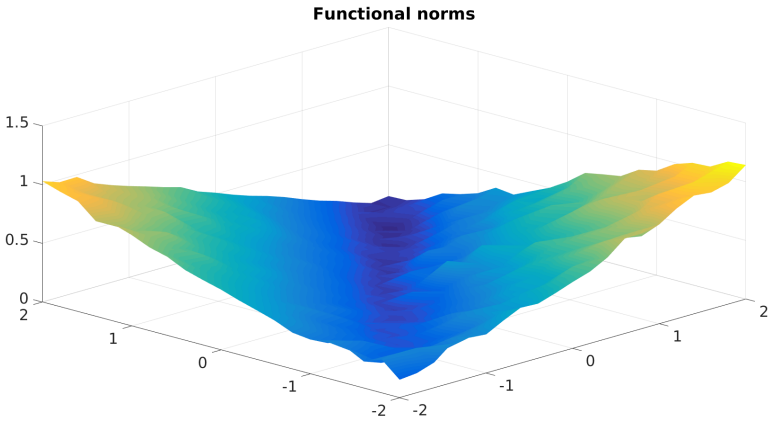
Néhány példa ilyen metszetekre látható a 4.2.1.-4.2.4. ábrákon. Ezek a függvények folytonosan jól közelíthetőnek látszanak, és nincs szingularitásuk.

4 dimenziós lineáris interpolációval becsülhetők ekkor a paraméterek. A módszer relatív hibastatisztikái az alábbi táblázatban láthatók, a hiba eloszlása a 4.2.5. ábrán.

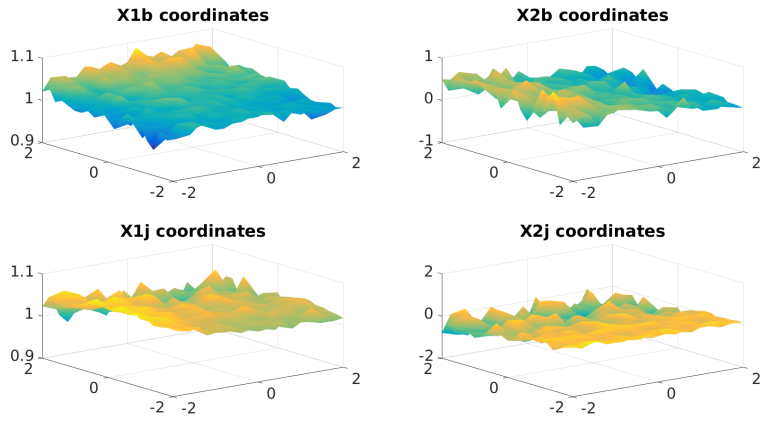
Paraméter	Átlag	Szórás	Medián
$\overrightarrow{x}_1$	0.1148	0.1782	0.0482
$\overrightarrow{x}_2$	0.6481	0.4787	0.5428
Célfüggvény	0.1167	0.1625	0.0524



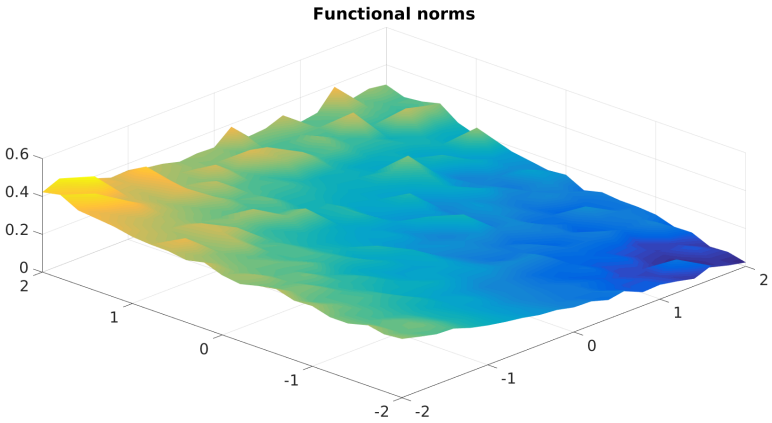
4.2.1. ábra. A négy szabadságfok függése a görbülettől ($\alpha = \frac{3\pi}{10}, \beta = -\frac{3\pi}{10}$)



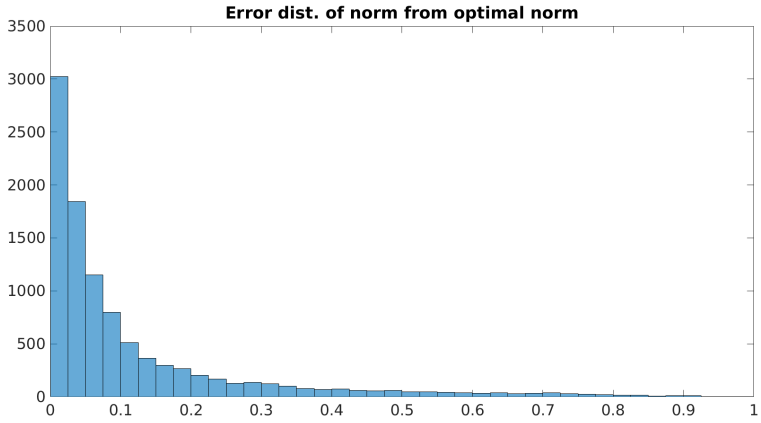
4.2.2. ábra. Optimális görbe (4.1) szerinti normája ($\alpha = \frac{3\pi}{10}, \beta = -\frac{3\pi}{10}$)



4.2.3. ábra. A négy szabadságfok függése a görbülettől ($\alpha = \frac{\pi}{10}, \beta = \frac{\pi}{10}$)



4.2.4. ábra. Optimális görbe (4.1) szerinti normája ($\alpha = \frac{\pi}{10}, \beta = \frac{\pi}{10}$)



4.2.5. ábra. Másodrendű illesztés optimalitási hibájának eloszlása.

Megfigyelhető egyrészt, hogy a pontosság lényegesen romlott az elsőrendű esethez képest, másrészt pedig az x_2 paraméterek becslése lényegesen rosszabb az x_1-ekénél. Ennek oka lehet, hogy az x_2 koordináták optimuma gyakran közel van 0-hoz, ami felerősítheti a relatív hibát.

4.3. Hatékonyság

Az algoritmus C++ prototípus-implementációja az elsőrendű, harmadfokú konfigurációkra a közelítően konstans sebesség funkcionálra egy szálon futva *266* másodperc alatt számítja ki az optimumot egy 20×20-as rácson. Nagyobb rendek, fokszámok, felosztási sűrűség, valamint bonyolultabb funkcionál esetén a műveletigény jelentősen nőtt, akár több órás is lehetett.

Az implementáció GPU-ra való átültetése esetén minden egyes konfigurációra egy-egy workgroup indul. Ekkor ugyanez a műveletigény *2.55* másodperc.

5 Összefoglalás

A dolgozatban megmutattuk, hogy bizonyos funkcionálok esetén van lehetőség konstans időben közelíteni a numerikus optimalizálással kapott görbeértékeket rácson mintavételezett optimumok segítségével.

Elsőrendű geometriai Hermite feladatoknál konstans sebességet leíró funkcionálok esetén a szakaszos Hermite illetve a bilineáris szűrések hibastatisztikái bizonyultak a legjobbnak. A pontosság mellett a hatékonyságot is figyelembe véve a bilineáris módszer megfelelő választás lehet gyakorlati problémákra is.

Görbületfüggő funkcionálok esetén azt tapasztaltuk, hogy a bemeneti geometriai konfigurációknak van olyan összefüggő részhalmaza, amelyen szintén elfogadható hibán belül lehetséges konstans időben közelítően optimális görbét találni.

Az implementáció segítségével jól feltérképezhető, hogy melyek azok a konfigurációk, melyekre a fokszámemelés szignifikáns javulást okozhat az eredménygörbe optimalitásában. Ezáltal segítheti tervezők stratégiai döntését, hogy adott alkalmazáshoz milyen fokszámú görbéket érdemes használni.

További vizsgálat tárgyát képezheti más, nem feltétlen reguláris mintavételezések, illetve nem csak szabályos rácsokon értelmezett szűrések alkalmazása az optimumleképezés mintákkal való közelítésére, például szórt adat approximációs módszerek segítségével.

A Képletek, levezetések

A.1. Magasabb rendű geometriai invariánsok képlete

A 2.3-2.4 képletek az alábbi módon vezethetők le:

A Frenet-koordináták kielégítik az alábbi differenciálegyenleteket ([10]):

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ z_{n+1} \end{bmatrix} = \begin{bmatrix} x'_n \\ y'_n \\ z'_n \end{bmatrix} + x_1 \begin{bmatrix} 0 & -\kappa & 0 \\ \kappa & 0 & -\tau \\ 0 & \tau & 0 \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} \tag{A.1}$$

$n = 1$-re felírva:

$$\begin{bmatrix} x_2 \\ y_2 \\ 0 \end{bmatrix} = \begin{bmatrix} x'_1 \\ 0 \\ 0 \end{bmatrix} + x_1 \begin{bmatrix} 0 & -\kappa & 0 \\ \kappa & 0 & -\tau \\ 0 & \tau & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ 0 \\ 0 \end{bmatrix} \tag{A.2}$$

Az első komponensek egyenlőségéből: $x'_1 = x_2$ (ezt később felhasználjuk)

A második komponensek egyenlőségéből: $y_2 = \kappa x_1^2$, amiből:

$$\kappa = \frac{y_2}{x_1^2} \tag{A.3}$$

$n = 2$ esetén:

$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} x'_2 \\ y'_2 \\ 0 \end{bmatrix} + x_1 \begin{bmatrix} 0 & -\kappa & 0 \\ \kappa & 0 & -\tau \\ 0 & \tau & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 0 \end{bmatrix} = \begin{bmatrix} x'_2 + \dfrac{y_2^2}{x_1} \\ y'_2 + \dfrac{x_2 y_2}{x_1} \\ x_1 \tau y_2 \end{bmatrix} \tag{A.4}$$

$x_2' = x_3 - \frac{y_2^2}{x_1}$

$y_2' = (\kappa x_1^2)' = \kappa' x_1^3 + \kappa 2 x_1 x_1' = \kappa' x_1^3 + 2\frac{y_2 x_2}{x_1}$

Másrészt $y_2' = y_3 - \frac{x_2 y_2}{x_1}$, amiből:

$$\begin{aligned} \kappa' x_1^3 + 2\frac{x_2 y_2}{x_1} &= y_3 - \frac{x_2 y_2}{x_1} \\ \kappa' x_1^3 &= y_3 - 3\frac{x_2 y_2}{x_1} \\ \kappa' &= \frac{y_3 x_1 - 3 x_2 y_2}{x_1^4} \end{aligned} \tag{A.5}$$

Illetve a harmadik sorból:

$$z_3 = \tau x_1 y_2 \Rightarrow \tau = \frac{z_3}{x_1 y_2} \tag{A.6}$$

Az $n = 3$ rendű egyenlet:

$$\begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix} = \begin{bmatrix} x_3' \\ y_3' \\ z_3' \end{bmatrix} + x_1 \begin{bmatrix} 0 & -\kappa & 0 \\ \kappa & 0 & -\tau \\ 0 & \tau & 0 \end{bmatrix} \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_3' - x_1 \kappa y_3 \\ y_3' + x_1 \kappa x_3 - x_1 \tau z_3 \\ z_3' + x_1 \tau y_3 \end{bmatrix} \tag{A.7}$$

A második sor szerint:

$y_3' = (\kappa' x_1^3 + 3 x_1 \kappa x_2)' = \kappa'' x_1^4 + \kappa' 3 x_1^2 x_2 + 3\kappa' x_1^2 x_2 + 3\kappa x_2^2 + 3\kappa x_1 (x_3 - \frac{y_2^2}{x_1}) =$
$= \kappa'' x_1^4 + \kappa' 6 x_1^2 x_2 + \kappa 3 (x_2^2 + x_1 x_3 - y_2^2) = \kappa'' x_1^4 + 3 x_2 \frac{y_3 x_1 - 3 x_2 y_2}{x_1^2} + 3\frac{y_2 x_2^2 + y_2 x_1 x_3 - y_2^3}{x_1^2} =$
$= \kappa'' x_1^4 + 3\frac{x_1 x_2 y_3 + x_1 x_3 y_2 - y_2^3}{x_1^2}$

$y_4 = y_3' + x_1 \kappa x_3 - x_1 \tau z_3 = \kappa'' x_1^4 + 3\frac{x_1 x_2 y_3 + x_1 x_3 y_2 - y_2^3}{x_1^2} + \frac{y_2 x_3}{x_1} - \frac{z_3^2}{y_2}$

Ebből:

$$\begin{aligned} \kappa'' x_1^4 &= y_4 - 3\frac{x_1 x_2 y_3 + x_1 x_3 y_2 - y_2^3}{x_1^2} - \frac{y_2 x_3}{x_1} + \frac{z_3^2}{y_2} \\ \kappa'' x_1^4 &= y_4 - 3\frac{x_1 x_2 y_3 - y_2^3}{x_1^2} + \frac{z_3^2}{y_2} \\ \kappa'' &= \frac{y_4 x_1^2 y_2 - 3 x_1 x_2 y_2 y_3 + y_2^4 + z_3^2 x_1^2}{x_1^6 y_2} \end{aligned} \tag{A.8}$$

Továbbá a harmadik sorból:

$z_4 = z_3' + x_1\tau y_3 = \tau' x_1^2 y_2 + \tau x_2 y_2 + \tau x_1(y_3 - \frac{x_2 y_2}{x_1}) + x_1\tau y_3 = \tau' x_1^2 y_2 + 2\tau x_1 y_3 = \tau' x_1^2 y_2 + 2\frac{z_3 y_3}{y_2}$

Ebből:

$$\tau' = \frac{z_4}{x_1^2 y_2} - 2\frac{z_3 y_3}{x_1^2 y_2^2} \tag{A.9}$$

A.2. Az L-BFGS algoritmus

Az L-BFGS ([4]) iterációs módszer, a Newton-szerű módszerek családjába tartozik. A Newton-módszer optimalizációs változata:

$\Delta x := f''(x_k)^{-1} \cdot f'(x_k)$

$x_{k+1} := x_k - \Delta x$

Az L-BFGS módszer használata esetén nincs szükség a második deriváltat, azaz a Hesse-mátrix meghatározására. Ezáltal a műveletigény jóval kisebb lehet. Ráadásul, ha a másodrendű parciális deriváltakat csak numerikusan tudnánk becsülni, akkor sok esetben robusztusabb eljárást is kapunk a Newtonhoz képest.

Az alábbiakban $x_k \in \mathbb{R}^n$ a k-adik iterációs lépés pozíciója, $g_k \in \mathbb{R}^n$ az ottani gradiens. $m \in \mathbb{Z}^+$ konstans, mellyel szabályozható, hogy hány korábbi lépést vegyen figyelembe a Hesse-mátrix approximációja során.

Algoritmus A.2.1 L-BFGS lépésszámítás

Bemenet: $x_0, ..., x_k \in \mathbb{R}^n$ iterációs pontok, $d_i = f'(x_i)(i = 0..k)$

$y_{k-1} := g_k - g_{k-1}$

$\rho_{k-1} := \dfrac{1}{\langle y_{k-1}, s_{k-1} \rangle}$

$q := g_k$

Ciklus $j := k - 1.. \max(0, k - m)$

$\quad \alpha_j := \rho_j \cdot \langle s_j, q \rangle$

$\quad q := q - \alpha_j \cdot y_j$

Ciklus vége

$H_k^0 := \dfrac{y_{k-1} \cdot s_{k-1}^\top}{\langle y_{k-1}, y_{k-1} \rangle}$

$z := H_k^0 q$

Ciklus $j := \max(0, k - m)..k - 1$

$\quad \beta_j := \rho_j \langle y_j, z \rangle$

$\quad z := z + s_j(\alpha_j - \beta_j)$

Ciklus vége

$\Delta x := z$

$x_{k+1} := x_k - \Delta x$

B Segédlet a program használatához

A program forráskódja (https://github.com/Smillancs/CurveLib/tree/opt_dev) letölthető, fordításához és futtatásához az alábbiak nyújtanak segítséget.

Windows rendszeren

A fordítási függőségek megtalálhatók a kódhoz csatolt *OGLPack* mappában. Fordítóként Visual Studio (legalább 2015-ös) használata javasolt. A projektfájlok CMake-kel generálhatók, a *CMakeLists.txt* alapján. A Viewer nevű projektet kell futtatni a fordítás után, ekkor jelenik meg a 3.3.1-ban leírt felület.

Linux rendszeren

A függőségek a legtöbb disztribúción könnyen telepíthetők csomagkezelőből. Szükséges egy OpenGL-kompatibilis videókártya-driver, továbbá a GLEW, GLU, GLM, Eigen, SDL2 könyvtárak fejlesztéshez szükséges (headereket tartalmazó) változatai. A CMake forrásokból generálhatók pl. makefile-ok. C++14 kompatibilis fordítóval fordítható a kód. Szintén a Viewer projektből létrejött futtathatót kell elindítani.

Ábrák jegyzéke

Irodalomjegyzék

[1] Carl De Boor, Klaus Höllig, and Malcolm Sabin. High accuracy geometric Hermite interpolation. *Computer Aided Geometric Design*, 4(4):269–278, 1987.

[2] Gerald Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2002.

[3] Rida T Farouki. Legendre–Bernstein basis transformations. *Journal of Computational and Applied Mathematics*, 119(1):145–160, 2000.

[4] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[5] Henry P Moreton and Carlo H Séquin. *Functional optimization for fair surface design*, volume 26. ACM, 1992.

[6] Helmut Pottmann. Projectively invariant classes of geometric continuity for CAGD. *Computer Aided Geometric Design*, 6(4):307–321, 1989.

[7] Robert Schaback. Optimal geometric Hermite interpolation of curves. *Mathematical Methods for Curves and Surfaces II*, pages 1–12, 1998.

[8] Ferenc Schipp. *Analízis 3 - Differenciálgeometria (egyetemi jegyzet)*. 2003.

[9] Dave Shreiner, Graham Sellers, John Kessenich, and Bill Licea-Kane. *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. Addison-Wesley, 2013.

[10] Gábor Valasek. Nonlinear geometric models.

[11] Volker Weiss, Laszlo Andor, Gábor Renner, and Tamás Várady. Advanced surface fitting techniques. *Computer Aided Geometric Design*, 19(1):19–42, 2002.