



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Parametrikus görbék és felületek pontos offsetelése

SZAKDOLGOZAT

Készítette
Sandle Nátán

Konzulens
Salvi Péter

2025-05-21

Tartalomjegyzék

1. Bevezetés	1
1.1. CAD/CAM	1
1.2. Parametrikus görbék, felületek	1
1.3. Polinomok, racionális függvények	1
1.4. Kontrollpont-alapú reprezentáció	2
1.4.1. Bézier görbék	2
1.4.2. B-Spline	3
1.4.3. NURBS	3
1.5. Parametrikus sebesség	4
2. Polinomiális PH Görbék	6
2.1. PH síkgörbék	6
2.1.1. Reprezentáció komplex számokkal	7
2.1.2. Interpoláció	7
2.2. PH térgörbék	7
2.2.1. Alapok	7
2.2.2. Reprezentáció kvaterniókkal	7
2.2.3. Interpoláció	7
3. PN felületek	8
4. PN interpoláció C^1 folytonossággal	9
4.1. Feladat	9
4.2. Duális reprezentáció	9
4.3. Izotróp tér	10
4.4. Irányvektorok meghatározása	11
4.5. Coons-patch	12
4.6. Visszatranszformálás, az eredmény fokszáma	13
5. Implementációs részletek	17
5.1. overview	17
5.2. Lineáris algebra	17
5.3. Polinom osztály	17
5.3.1. Változók	17
5.3.2. Struktúra	17
5.3.3. Konstruktív szintaxis	18
5.3.4. Funkcionalitás	18

5.3.4.1. Aritmetikai műveletek	18
5.3.4.2. Kiértékelés	18
5.3.4.3. Derivált	18
5.3.5. Template	19
5.3.6. Racionális függvények	19
5.3.7. Tesztek	20
5.4. grid	20
5.5. range2d	20
5.6. Megjelenítés	20
6. Eredmények	21
6.1. Szabad paraméterek	21
6.2. Végponti deriváltak	22
6.3. Adatpontok előzetes transzformálása	23
6.3.1. Eltolás	23
6.3.2. Skálázás	24
6.3.3. Forgatás	25
6.3.4. Irányvektorok állítása előzetes transzformálás helyett	25
6.3.5. Példa	25
7. Végző	26
7.0.1. Módszer megítélése	26
7.0.2. További kutatás	26

HALLGATÓI NYILATKOZAT

Alulírott *Sandle Nátán*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2025-05-21

Sandle Nátán
hallgató

1. Fejezet

Bevezetés

1.1. CAD/CAM

a

1.2. Parametrikus görbék, felületek

Compu

1.3. Polinomok, racionális függvények

Amikor geometriai alakzatokat szeretnénk szoftveresen reprezentálni, figyelembe kell vennünk a számítógépek technikai limitációit. A reprezentációban megjelenő matematikai kifejezéseket sokszor ki kell értékelnünk, ennek az időigénye és pontossága pedig drasztikus mértékben függ a kifejezés jellegétől.

Az összeadást, kivonást és szorzást nagyon egyszerű algoritmusokkal, akár 1 CPU-ciklus alatt végre tudjuk hajtani, az eredmény pontossága csak a számok mögötti adatszerkezet (általában floating-point) limitációitól függ.

Azokat a függvényeket, amik kifejezhetők véges sok összeadással, kivonással és szorzással, polinomoknak hívjuk. Egy egyváltozós polinom kanonikus alakja

$$P(x) = \sum_{k=0}^n a_k x^k$$

Az osztás egy kissé költségesebb (illetve adott esetben pontatlanabb) művelet. Ha az osztást is megengedjük, az így kifejezhető függvényeket racionális függvényeknek hívjuk. Minden racionális függvény leírható az alábbi alakban

$$R(x) = \frac{A(x)}{B(x)}$$

ahol $A(x)$ és $B(x)$ polinomok. Ez kedvező, mert így egy racionális függvény kiértékelésekor elég csak egyszer osztani.

Sok nevezetes függvényt (például \sqrt{x} , $\sin(x)$, $\ln(x)$) nem lehet kifejezni véges sok alpművelettel, értéküket csak megközelíteni tudjuk. Ezt vagy egy közelítő

polinommal/racionálissal tesszük (pl Taylor-sor, Padé közelítő), vagy ismételt, inkrementálisan közelítő lépéseket hajtunk végre (pl Newton-módszer).

Ebből következik, hogy az ilyen függvények kiértékelése lassabb, pontatlanabb, vagy mindkettő, mint egy alacsony fokú polinom vagy racionális függvény. Így lehetőség szerint el akarjuk kerülni őket egy CAD környezetben.

1.4. Kontrollpont-alapú reprezentáció

Ha egy görbét/felületet meghatározó polinomot a szokásos hatványösszeg alakban írunk le, az együtthatók nem nyújtanak intuitív betekintést a görbe/felület geometriai tulajdonságaiba. A CAD-ben elterjedtek olyan alternatív reprezentációk, melyek.

A kontrollpontok tekinthetők együtthatóknak egy másik bázisban, de léteznek

1.4.1. Bézier görbék

Egy n -ed fokú Bézier görbét $n + 1$ kontrollponttal reprezentálunk. Kiértékelni a De Casteljau algoritmussal tudjuk, ami rekurzív lineáris interpolációra épül. A Béziér kontrollpontok a görbe mögötti polinom együtthatói a Bernstein-bázisban, melynek k -adik eleme

$$b_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

A Béziér görbe $t = 0$ -ban áthalad az első kontrollponton, $t = 1$ -ben az utolsón, a többi pedig közelíti. Az első illetve utolsó kettő kontrollpontot összekötő egyenes érinti a görbét az első illetve utolsó kontrollpontban. Kifejezetten népszerű a harmadfokú Bézier görbe a graphic design területén, hiszen egyszerűen lehet állítani a görbe irányait a végpontokban.



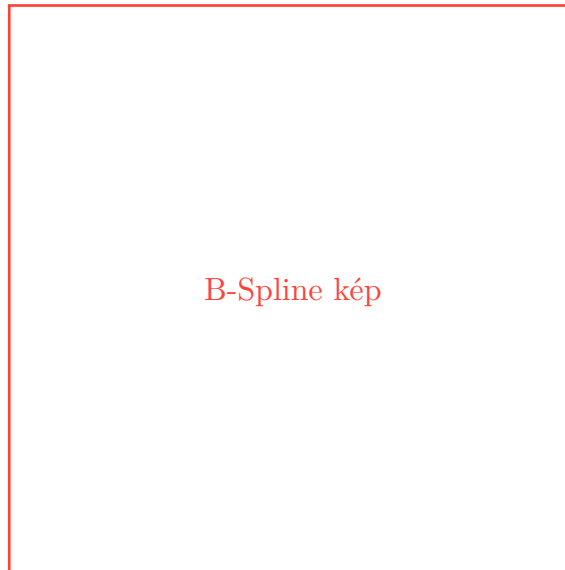
1.4.2. B-Spline

A B-Spline (Basis-Spline) darabonként definiált bázisfüggvényekből áll, melyeknek szegmenseit úgynevezett „csomópontok” (knots) választják el $(t_0, t_1 \dots t_m)$. A bázisfüggvényeket A Cox-de Boor képlettel tudjuk kiértékelni:

$$B_{i,0}(t) := \begin{cases} 1 & \text{ha } t_i \leq t < t_{i+1} \\ 0 & \text{egyébként} \end{cases}$$
$$B_{i,n}(t) := \frac{t - t_i}{t_{i+n} - t_i} B_{i,n-1}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} B_{i+1,n-1}(t)$$

Ebből következik, hogy egy kontrollpont csak a környező $n + 1$ szegmensre hat ki, így lehetőséget ad a lokális kontrollra.

A B-Spline egyik fő előnye, hogy „maximális folytonosságot” biztosít a szegmensek között, n -edfokú spline esetén C^{n-1} -et. Azonban általános esetben az egyik kontrollponton sem megy át, csak közelíti őket. Csomópontok ismétlésével elérhető, hogy a görbe átmenjen egy kontrollponton, ez azonban a folytonosság vesztesével jár. Mivel ez nem okoz gondot az első és utolsó kontrollpontban, ott gyakran megteszik (clamping).



1.4.3. NURBS

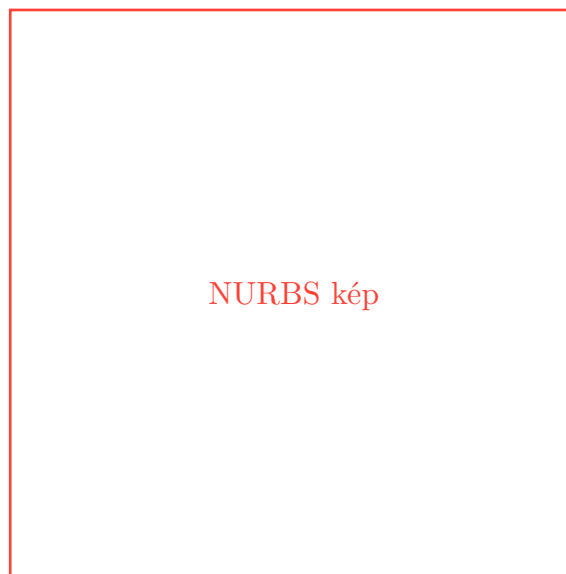
A NURBS valójában nem más, mint a B-Spline általános esete. A rövidítés kifejtése: „Non-Uniform Rational B-Spline”.

A „non-uniform” rész azt jelenti, hogy nem feltétlenül vannak a csomópontok egyenlő távolságra egymástól, így például megengedett a korábban említett csomópont ismétlés is. A kontrollpontok sűrítése a görbe/felület adott szakaszain lehetőséget ad a finomabb részletek lokális szerkesztésére. Az úgynevezett „knot insertion” algoritmussal hozzá tudunk adni egy új csomópontot egy B-Spline-hoz, anélkül, hogy annak az alakját változtatnánk.

A „rational” rész azt jelenti, hogy egyszerű polinomok helyett racionális függvények vannak a háttérben. Ez a gyakorlatban úgy nyilvánul meg, hogy minden kontrollponthoz rendelünk egy súlyt. Számoláskor az adott kontrollponthoz tartozó komponenst beszorozzuk a súllyal, majd végül osztunk a bázisfüggvények súlyozott összegével. Nagyobb súly hatására a görbe nagyobb mértékben fog húzni az adott kontrollpont irányába.

A súlyokat értelmezhetjük a számítógépes grafikában elterjedt projektív geometriával. Elképzelhetjük, hogy a spline egy egyel nagyobb dimenziós térben él, ahol az utolsó koordináta a súly (homogén koordináták). Így az osztás nem más, mint vetítés az eredeti térbe.

A racionális függvényekre való kiterjeszkedés lehetővé teszi a körívek/gömbfelületek pontos leírását.



1.5. Parametrikus sebesség

Egy $\mathbf{r}(t)$ görbe parametrikus sebessége alatt a görbe deriváltjának nagyságát értjük. Ezt a koordinátánként vett deriváltakból a pitagoraszai távolságképlettel tudjuk kiszámolni, ami egy síkgörbe esetén így néz ki:

$$\sigma(t) = |\mathbf{r}'(t)| = \sqrt{x'(t)^2 + y'(t)^2}$$

A parametrikus sebesség ismerete fontos lehet CAD/CAM környezetben, márcsak azért is, mert segítségével további hasznos dolgokat ki tudunk számolni. A korábbi definícióból következik, hogy a parametrikus sebesség integrálásával megkapjuk a görbe hosszát. Az előjeles görbületet ki tudjuk számolni az alábbi képlettel

$$k = \frac{x'y'' - y'x''}{\sigma^3}$$

Számunkra a legfontosabb az adott távolságra lévő „párhuzamos görbe” (más szóval offset) képlete lesz

$$\mathbf{r}_d = \mathbf{r} + d\mathbf{n}$$

ahol \mathbf{n} az \mathbf{r} görbére állított egység-hosszúságú normálvektor. Normálvektort egyszerűen kapunk úgy, hogy merőlegest állítunk a deriváltra. De ahhoz, hogy ez egység-hosszúságú legyen, le kell osztanunk a derivált nagyságával, tehát a parametrikus sebességgel. Így a képlet

$$\mathbf{r}_d = \mathbf{r} + d \frac{\mathbf{r}'_{\perp}}{\sigma}$$

Sajnos a parametrikus sebességgel van egy jelentős probléma a gyakorlati felhasználás terén: a négyzetgyök miatt egy polinomiális/racionális görbe parametrikus sebessége általános esetben nem polinomiális/racionális. Ez nem csak azért okoz gondot, mert költségesebbé teszi a kiértékelést, hanem azért is, mert így nem tudjuk az offsetet kifejezni a szokásos kontrollpont alapú módszerekkel. Így CAD/CAM rendszerekben gyakran pontatlan közelítéseket kell alkalmazni.

2. Fejezet

Polinomiális PH Görbék

A „Pythagorean-hodograph” (PH) görbék olyan speciális polinomiális/racionális görbék, melyeknek a parametrikus sebessége is polinomiális/racionális. A névben a „hodográf” kifejezés valamilyen mozgás sebességének (tehát deriváltjának) a grafikonját jelenti. Ilyen ábrákat használnak például szélesebbégek elemzésére. A „pitagoraszi” alatt pedig a Pitagorasz tételt értjük egy derékszögű háromszög oldalai közti összefüggésre. Összesítve, PH görbe alatt olyan parametrikus görbét értünk, ahol az alábbi kifejezésben

$$x'(t)^2 + y'(t)^2 = \sigma(t)^2$$

minden tag, tehát a parametrikus sebesség is, egy polinom, vagy racionális függvény (még a négyzetre emelés előtt is).

2.1. PH síkgörbék

Ugyanúgy, ahogy léteznek a , b , c pitagoraszi (egész) számhármások, melyekre teljesül, hogy $a^2 + b^2 = c^2$, léteznek $a(t)$, $b(t)$, $c(t)$ pitagoraszi *polinom*hármások is. Ilyen polinomhármásokat tudunk generálni is, még hozzá ugyanazzal a képlettel, amivel a számhármásokat.

Legyen $m(t)$ és $n(t)$ két polinom. Ekkor a polinomhármásunk

$$a(t) = m^2(t) - n^2(t)$$

$$b(t) = 2m(t)n(t)$$

$$c(t) = m^2(t) + n^2(t)$$

A mi parametrikus görbénkben

$$x'(t) = a(t) \qquad y'(t) = b(t) \qquad \sigma(t) = c(t)$$

A görbét magát megkaphatjuk a deriváltak integrálásával (és kiindulópont választásával).

2.1.1. Reprezentáció komplex számokkal

A PH görbék generálásának fent leírt módját elegánsabban ki tudjuk fejezni, ha komplex számokat használunk. Legyen

$$\mathbf{w}(t) = m(t) + in(t)$$

Ekkor

$$\begin{aligned}\mathbf{w}^2(t) &= m(t)^2 - n(t)^2 + 2im(t)n(t) \\ &= x'(t) + iy'(t)\end{aligned}$$

Továbbá

$$\begin{aligned}|\mathbf{w}(t)|^2 &= \mathbf{w}(t)\mathbf{w}^*(t) = m^2(t) + n^2(t) \\ &= \sigma(t)\end{aligned}$$

Végül

$$\mathbf{r}(t) = x(t) + iy(t)$$

2.1.2. Interpoláció

2.2. PH térgörbék

2.2.1. Alapok

2.2.2. Reprezentáció kvaterniókkal

2.2.3. Interpoláció

3. Fejezet

PN felületek

4. Fejezet

PN interpoláció C^1 folytonossággal

4.1. Feladat

4.2. Duális reprezentáció

Egy olyan $\mathbf{x}(\mathbf{s})$ racionális felületet keresünk, melynek egység hosszúságú normálvektorait leíró $\mathbf{n}(\mathbf{s})$ függvény szintén racionális. Kézenfekvő lehet „fordítva gondolkozni”: először konstruálni egy garantáltan racionális $\mathbf{n}(\mathbf{s})$ -t, majd ebből meghatározni $\mathbf{x}(\mathbf{s})$ -t. Felületünket a szokásos (x, y, z) koordináták helyett reprezentálhatjuk az úgynevezett „duális térben”, (n_x, n_y, n_z, h) koordinátákkal. Ezek a koordináták a felület egy pontja helyett a felület egy érintősíkját írják le.

Ha \mathbf{x} a felület egy pontja, \mathbf{n} pedig a felület normálvektora ebben a pontban, az ennek megfelelő pont a duális térben (\mathbf{n}, h) , ahol:

$$\mathbf{x} \cdot \mathbf{n} = h$$

Ha feltételezzük, hogy \mathbf{n} egység hosszúságú, akkor h nem más, mint az érintő sík távolsága az origótól. A $h(\mathbf{s})$ függvényt a felület support függvényének hívjuk.

Ezzel a képlettel már át tudjuk transzformálni az interpolálandó adatpontokat a duális térbe. Ahhoz, hogy a végeredményt leírassuk a „primális” térben, szükségünk lesz az inverzre is, tehát \mathbf{n} -ből és h -ból ki szeretnénk számolni \mathbf{x} -et. Ehhez először fel kell írunk néhány azonosságot.

$\mathbf{x}(\mathbf{s})$ parciális deriváltjai párhuzamosak az érintősíkkal

$$\frac{d\mathbf{x}}{ds}^T \mathbf{n} = \mathbf{0}$$

Így $h(\mathbf{s})$ deriváltja

$$\frac{dh}{ds} = \frac{d}{ds} \mathbf{x}^T \mathbf{n} = \mathbf{x}^T \frac{d\mathbf{n}}{ds}$$

Mivel $\mathbf{n}(\mathbf{s})$ egység hosszúságú, egy gömbfelületet ír le. Parciális deriváltjai merőlegesek rá

$$\begin{aligned}\frac{d}{ds}\mathbf{n} \cdot \mathbf{n} &= 2 \mathbf{n}^T \frac{d\mathbf{n}}{ds} = \frac{d}{ds}1 = 0 \\ \Rightarrow \frac{d\mathbf{n}}{ds}^T \mathbf{n} &= 0\end{aligned}$$

$h\mathbf{n}$ egy pont az érintősíkon, $\frac{d\mathbf{n}}{du}$ és $\frac{d\mathbf{n}}{dv}$ pedig az érintősíkkal párhuzamos vektorok. Így \mathbf{x} -et ki tudjuk fejezni az alábbi módon

$$\mathbf{x} = h\mathbf{n} + \frac{d\mathbf{n}}{ds} \cdot \mathbf{r}$$

Szorozva $\frac{d\mathbf{n}}{ds}^T$ -al

$$\begin{aligned}\frac{dh}{ds}^T &= \frac{d\mathbf{n}}{ds}^T \frac{d\mathbf{n}}{ds} \cdot \mathbf{r} \\ \mathbf{r} &= \left(\frac{d\mathbf{n}}{ds}^T \frac{d\mathbf{n}}{ds} \right)^{-1} \frac{dh}{ds}^T\end{aligned}$$

Tehát

$$\mathbf{x} = h\mathbf{n} + \frac{d\mathbf{n}}{ds} \left(\frac{d\mathbf{n}}{ds}^T \frac{d\mathbf{n}}{ds} \right)^{-1} \frac{dh}{ds}^T$$

4.3. Izotróp tér

Az egység hosszúságú normálvektor előírásával \mathbb{R}^4 -et leszűkítettük \mathcal{B} -re, az úgynevezett Blaschke hengerre. Az interpoláció közben szeretnénk biztosítani, hogy a hengeren maradunk. Ennek érdekében bevezetünk egy új reprezentációt, az izotróp térben. Ezt a reprezentációt úgy állítjuk elő, hogy a $\mathbf{w} = (0, 0, 1, 0)$ pontból az $n_z = 0$ hipersíkba vetítünk

$$\mathbf{y}(\mathbf{b}) = \frac{1}{1 - n_z} \begin{pmatrix} n_x \\ n_y \\ h \end{pmatrix}$$

Ennek az inverze

$$\mathbf{b}(\mathbf{y}) = \frac{1}{1 + y_x^2 + y_y^2} \begin{pmatrix} 2y_x \\ 2y_y \\ -1 + y_x^2 + y_y^2 \\ 2y_z \end{pmatrix}$$

Az izotróp térben szabadon interpolálhatunk a transzformált adatpontok között, majd a felületet visszavetítjük a Blaschke hengerre.

Bárhogy is interpoláljuk az adatpontjainkat az izotróp térben, a visszatranszformált felület érintősíkjai meg fognak egyezni az előírtakkal. Ahhoz viszont, hogy

a konkrét térbeli pozíció is megegyezzen, korlátoznunk kell a felület lehetséges deriváltjait az interpolációs pontokban

$$\begin{aligned}\mathbf{x}^T \frac{d\mathbf{n}}{ds} &= \frac{dh}{ds} \\ \mathbf{x}^T \frac{d\mathbf{n}}{dy} \frac{dy}{ds} &= \frac{dh}{dy} \frac{dy}{ds} \\ \underbrace{\left(\mathbf{x}^T \frac{d\mathbf{n}}{dy} - \frac{dh}{dy} \right)}_{\mathbf{v}} \frac{dy}{ds} &= 0\end{aligned}$$

Ahol

$$\left(\begin{array}{c} \frac{d\mathbf{n}}{dy} \\ \frac{dh}{dy} \end{array} \right) = \frac{d\mathbf{b}}{dy} = \frac{2}{(1 + y_x^2 + y_y^2)^2} \begin{pmatrix} 1 - y_x^2 + y_y^2 & -2y_x y_y & 0 \\ -2y_x y_y & 1 + y_x^2 - y_y^2 & 0 \\ 2y_x & 2y_y & 0 \\ -2y_x y_z & -2y_y y_z & 1 \end{pmatrix}$$

Tehát az izotróp térben kiválasztott kezdeti/végponti deriváltaknak illeszkedniük kell a \mathbf{v} normálvektorú, origót tartalmazó síkra.

4.4. Irányvektorok meghatározása

Jelenleg rendelkezünk egy négyzetrács szerkezetű ponthálózattal, illetve pontonként egy síkkal. Mivel ezekből még nem következnek egyértelműen a pontokhoz rendelő deriváltak, heurisztikát fogunk alkalmazni.

Legyen $\mathbf{a}_{i,j}$ a hálózat egy pontja, ahol i a pont „ u irányban”, j pedig a „ v irányban” vett indexe. Legyen továbbá n és m a legmagasabb i , illetve j index. Jelölje $\gamma_{i,j}$ a felület u szerinti deriváltját az $\mathbf{a}_{i,j}$ pontban, $\delta_{i,j}$ pedig a v szerinti deriváltat ugyanitt.

Ha $\mathbf{a}_{i,j}$ a pontháló szélén van

$$\begin{aligned}\gamma_{0,j}^* &= \mathbf{a}_{1,j} - \mathbf{a}_{0,j} & \delta_{i,0}^* &= \mathbf{a}_{i,1} - \mathbf{a}_{i,0} \\ \gamma_{n,j}^* &= \mathbf{a}_{n,j} - \mathbf{a}_{n-1,j} & \delta_{i,n}^* &= \mathbf{a}_{i,n} - \mathbf{a}_{i,n-1}\end{aligned}$$

Egyébként a két vektort átlagoljuk

$$\gamma_{i,j}^* = \frac{\mathbf{a}_{i+1,j} - \mathbf{a}_{i-1,j}}{2} \quad \delta_{i,j}^* = \frac{\mathbf{a}_{i,j+1} - \mathbf{a}_{i,j-1}}{2}$$

A kapott vektorokat még le kell vetítenünk a \mathbf{v} által meghatározott síkra

$$\gamma_{i,j} = \gamma_{i,j}^* - \frac{\mathbf{v} \cdot \gamma_{i,j}^*}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v} \quad \delta_{i,j} = \delta_{i,j}^* - \frac{\mathbf{v} \cdot \delta_{i,j}^*}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v}$$

4.5. Coons-patch

Vegyünk a ponthálónkból egy kis négyzetet, ennek sarokpontjait nevezzük \mathbf{a}_{00} , \mathbf{a}_{01} , \mathbf{a}_{10} , \mathbf{a}_{11} -nek. Ezek között akarunk interpolálni úgy, hogy a létrejött felületdarab a vele szomszédos felületdarabokra C^1 folytonossággal illeszkedjen. Ehhez egy Coons patch-et fogunk használni.

A Coons patch létrehozásához szükségünk van 4 határgörbére $(\mathbf{c}_0(u), \mathbf{c}_1(u), \mathbf{d}_0(v), \mathbf{d}_1(v))$, ahol

$$\begin{aligned}\mathbf{c}_0(0) &= \mathbf{d}_0(0) = \mathbf{a}_{00} \\ \mathbf{c}_0(1) &= \mathbf{d}_1(0) = \mathbf{a}_{10} \\ \mathbf{c}_1(0) &= \mathbf{d}_0(1) = \mathbf{a}_{01} \\ \mathbf{c}_1(1) &= \mathbf{d}_1(1) = \mathbf{a}_{11}\end{aligned}$$

valamint egy 0 és 1 között interpoláló $F(t)$ függvényre. Az egyszerűség kedvéért a függvény tükörképét is nevezzük meg

$$\begin{aligned}F_0(t) &= 1 - F(t) \\ F_1(t) &= F(t)\end{aligned}$$

A Coons patch három részből áll. Az első kettő interpolál az egymással szemben álló görbék között

$$\begin{aligned}\mathbf{S}_c(u, v) &= F_0(v)\mathbf{c}_0(u) + F_1(v)\mathbf{c}_1(u) \\ \mathbf{S}_d(u, v) &= F_0(u)\mathbf{d}_0(v) + F_1(u)\mathbf{d}_1(v)\end{aligned}$$

A harmadik pedig interpolál a sarokpontok között

$$\begin{aligned}\mathbf{B}(u, v) &= F_0(u)F_0(v)\mathbf{a}_{00} + F_0(u)F_1(v)\mathbf{a}_{01} \\ &\quad + F_1(u)F_0(v)\mathbf{a}_{10} + F_1(u)F_1(v)\mathbf{a}_{11}\end{aligned}$$

Végül

$$\mathbf{y}(u, v) = \mathbf{S}_c(u, v) + \mathbf{S}_d(u, v) - \mathbf{B}(u, v)$$

A képletet értelmezhetjük úgy, hogy \mathbf{S}_c és \mathbf{S}_d összeadásával „kétszer interpoláltunk” a sarokpontok között, ezt kompenzáljuk \mathbf{B} kivonásával.

A Coons patch kifejezhető egy kompaktabb mátrix alakban is

$$\mathbf{y}(u, v) = (F_0(u) \ 1 \ F_1(u)) \cdot \begin{pmatrix} -\mathbf{a}_{00} & \mathbf{d}_{00}(v) & -\mathbf{a}_{01} \\ \mathbf{c}_{00}(u) & 0 & \mathbf{c}_{01}(u) \\ -\mathbf{a}_{10} & \mathbf{d}_{10}(v) & -\mathbf{a}_{11} \end{pmatrix} \cdot \begin{pmatrix} F_0(v) \\ 1 \\ F_1(v) \end{pmatrix}$$

Az interpoláló függvény általában lineáris vagy köbös szokott lenni. Ahhoz, hogy a patch-ek C^1 folytonossággal illeszkedjenek, nekünk köbösre lesz szükségünk

$$F_0(t) = 2t^3 - 3t^2 + 1$$

$$F_1(t) = -2t^3 + 3t^2$$

A határgörbékhez használjunk Hermite interpolációt

$$\begin{aligned} \mathbf{c}_0(u) &= \begin{pmatrix} F_0(u) \\ G_0(u) \\ F_1(u) \\ G_1(u) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{a}_{00} \\ \gamma_{00} \\ \mathbf{a}_{10} \\ \gamma_{10} \end{pmatrix} & \mathbf{d}_0(v) &= \begin{pmatrix} F_0(v) \\ G_0(v) \\ F_1(v) \\ G_1(v) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{a}_{00} \\ \gamma_{00} \\ \mathbf{a}_{10} \\ \gamma_{10} \end{pmatrix} \\ \mathbf{c}_0(u) &= \begin{pmatrix} F_0(u) \\ G_0(u) \\ F_1(u) \\ G_1(u) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{a}_{00} \\ \gamma_{00} \\ \mathbf{a}_{10} \\ \gamma_{10} \end{pmatrix} & \mathbf{d}_0(v) &= \begin{pmatrix} F_0(v) \\ G_0(v) \\ F_1(v) \\ G_1(v) \end{pmatrix} \cdot \begin{pmatrix} \mathbf{a}_{00} \\ \gamma_{00} \\ \mathbf{a}_{10} \\ \gamma_{10} \end{pmatrix} \end{aligned}$$

ahol

$$G_0(t) = t^3 - 2t^2 + t$$

$$G_1(t) = -2t^3 + 3t^2$$

Mivel F_0 és F_1 ugyanaz a Coons patch képletében, mint a határgörbékében, a kettőt összevonva megspórolhatjuk a görbék külön kiszámolását

$$\mathbf{y}(u, v) = (F_0(u) \ G_0(u) \ F_1(u) \ G_1(u)) \begin{pmatrix} \mathbf{a}_{00} & \delta_{00} & \mathbf{a}_{01} & \delta_{01} \\ \gamma_{00} & \mathbf{0} & \gamma_{01} & \mathbf{0} \\ \mathbf{a}_{10} & \delta_{10} & \mathbf{a}_{11} & \delta_{11} \\ \gamma_{10} & \mathbf{0} & \gamma_{11} & \mathbf{0} \end{pmatrix} \begin{pmatrix} F_0(v) \\ G_0(v) \\ F_1(v) \\ G_1(v) \end{pmatrix}$$

Mivel nem szorzunk össze azonos változótól függő függvényeket, a deriváltak is hasonlóan egyszerűek

$$\begin{aligned} \frac{d\mathbf{y}}{du} &= (F'_0(u) \ G'_0(u) \ F'_1(u) \ G'_1(u)) \begin{pmatrix} \mathbf{a}_{00} & \delta_{00} & \mathbf{a}_{01} & \delta_{01} \\ \gamma_{00} & \mathbf{0} & \gamma_{01} & \mathbf{0} \\ \mathbf{a}_{10} & \delta_{10} & \mathbf{a}_{11} & \delta_{11} \\ \gamma_{10} & \mathbf{0} & \gamma_{11} & \mathbf{0} \end{pmatrix} \begin{pmatrix} F_0(v) \\ G_0(v) \\ F_1(v) \\ G_1(v) \end{pmatrix} \\ \frac{d\mathbf{y}}{dv} &= (F_0(u) \ G_0(u) \ F_1(u) \ G_1(u)) \begin{pmatrix} \mathbf{a}_{00} & \delta_{00} & \mathbf{a}_{01} & \delta_{01} \\ \gamma_{00} & \mathbf{0} & \gamma_{01} & \mathbf{0} \\ \mathbf{a}_{10} & \delta_{10} & \mathbf{a}_{11} & \delta_{11} \\ \gamma_{10} & \mathbf{0} & \gamma_{11} & \mathbf{0} \end{pmatrix} \begin{pmatrix} F'_0(v) \\ G'_0(v) \\ F'_1(v) \\ G'_1(v) \end{pmatrix} \end{aligned}$$

4.6. Visszatranszformálás, az eredmény fokszáma

Az \mathbf{y} izotróp térből primális térbe transzformálásához szükséges műveleteket a fejezet korábbi pontjaiban már ismertettük. Viszont gyakorlati megfontolások miatt érdemes ezeket tovább alakítani, hogy a módszert végrehajtó programunk hatékonyabban tudjon futni.

A legfontosabb szempont a transzformálás során előforduló polinomok fokszámainak minimalizálása. Egy kétváltozós polinom tagjainak száma négyzetesen arányos a fokszámmal, a kiértékelés komplexitása köbösen (tagonként $O(n)$), a polinomok közti szorzása pedig már negyedik hatvánnyal (minden tagot minden taggal).

A racionális függvényeket reprezentáló kód jelenlegi formájában nem tudja a közös osztókat tartalmazó számlálót és nevezőt egyszerűsíteni, így azok felhalmozódnak a számolás során. Ez egyrészt egy szükségtelenül magas fokszámú reprezentációt ad a felületről, amit több ideig tart kiértékelni. Erre megoldás lehetne egy kódbeli implementáció a racionális függvények egyszerűsítésére. Másrészt pedig magát a transzformációt is jelentősen lelassítja, ha felesleges műveleteket kell elvégezni. Ez még akkor is így lenne, ha minden lépés után elvégeznénk egy egyszerűsítést, hisz az egyszerűsítésnek is van költsége.

A megvalósítás egy korábbi iterációjában a létrejött felület 56-od fokú lett, és a programnak több órán át tartott kiszámolni (érdekes módon a kiértékelés ehhez képest elenyészően kevés ideig tartott). Szerencsére a felület „valódi” fokszáma ennél általános esetben is sokkal alacsonyabb.

Ez azt jelenti, hogy ha visszatranszformálást először „kézzel” átalakítjuk, észrevesszünk és kihasználunk néhány azonosságot, akkor sokkal hatékonyabb kódot tudunk írni.

Vegyük \mathbf{y} transzformációját a Blaschke-hengerre

$$\mathbf{n} = \frac{1}{1 + y_x^2 + y_y^2} \begin{pmatrix} 2y_x \\ 2y_y \\ -1 + y_x^2 + y_y^2 \end{pmatrix} \quad h = \frac{2y_z}{1 + y_x^2 + y_y^2}$$

Adjunk külön nevet a számlálóknak és nevezőknek

$$\mathbf{n} = \frac{\mathbf{m}}{q} \quad h = \frac{k}{q}$$

Ezzel \mathbf{x} első komponensét le is tudhatjuk

$$\mathbf{x} = h\mathbf{n} + \nabla = \frac{k\mathbf{m}}{q^2} + \nabla$$

A deriváltak

$$\frac{d\mathbf{n}}{d\mathbf{y}} = \frac{2}{q^2} M \quad \frac{dh}{d\mathbf{y}} = \frac{2}{q^2} \mathbf{k}$$

Ahol

$$M = \begin{pmatrix} 1 - y_x^2 + y_y^2 & -2y_x y_y \\ -2y_x y_y & 1 + y_x^2 - y_y^2 \\ 2y_x & 2y_y \end{pmatrix}$$

$$\mathbf{k} = (-2y_x y_z \quad -2y_y y_z \quad 1)$$

Az y_z szerinti derivált oszlopát levágtuk M -ről, hisz az mind 0.

A második komponens

$$\nabla = \frac{d\mathbf{n}}{ds} \left(\frac{d\mathbf{n}^T}{ds} \frac{d\mathbf{n}}{ds} \right)^{-1} \frac{dh}{ds}$$

Tudjuk, hogy

$$\frac{d\mathbf{n}}{ds} = \frac{d\mathbf{n}}{dy} \frac{dy}{ds} = \frac{2}{q^2} M \frac{dy}{ds} \quad \frac{dh}{ds} = \frac{dh}{dy} \frac{dy}{ds} = \frac{2}{q^2} \mathbf{k} \frac{dy}{ds}$$

Így

$$\nabla = \frac{4}{q^4} M \frac{dy}{ds} \left(\frac{4}{q^4} \frac{dy^T}{ds} M^T M \frac{dy}{ds} \right)^{-1} \frac{dy^T}{ds} \mathbf{k}^T$$

Vegyük észre, hogy a $\frac{4}{q^4}$ tagok kiejtik egymást.

M két oszlopa (az u illetve v szerinti deriválthoz tartozó részek) mindig merőleges egymásra. Belátható, hogy

$$M^T M = q^2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Bontsuk szét $\frac{dy}{ds}$ -t

$$\frac{dy}{ds} = \begin{pmatrix} Y_{xy} \\ \mathbf{y}_z \end{pmatrix}$$

És rendezzünk tovább

$$\nabla = \frac{1}{q^2} M Y_{xy} (Y_{xy}^T Y_{xy})^{-1} \begin{pmatrix} Y_{xy} \\ \mathbf{y}_z \end{pmatrix}^T \mathbf{k}^T$$

$$\nabla = \frac{1}{q^2} M (Y_{xy}^{-1})^T \begin{pmatrix} Y_{xy} \\ \mathbf{y}_z \end{pmatrix}^T \mathbf{k}^T$$

$$\nabla = \frac{1}{q^2} M \left(\mathbf{k} \begin{pmatrix} Y_{xy} \\ \mathbf{y}_z \end{pmatrix} Y_{xy}^{-1} \right)^T$$

Bontsuk szét \mathbf{k} -t is

$$\mathbf{k} = (\mathbf{k}_{xy} \ 1)$$

$$\nabla = \frac{1}{q^2} M(\mathbf{k}_{xy} + \mathbf{y}_z Y_{xy}^{-1})^T$$

Y_{xy} inverzének is vegyük külön a számlálóját és nevezőjét

$$Y_{xy}^{-1} = \frac{1}{|Y_{xy}|} Y'_{xy}$$

$$\nabla = \frac{1}{q^2 |Y_{xy}|} M(|Y_{xy}| \mathbf{k}_{xy} + \mathbf{y}_z Y'_{xy})^T$$

Ezzel az egyszerűítéssel csak 17-edfokú az eredmény, a számítás pár másodperc alatt lefut. Valójában még ez az alak sem optimális még, egy CAS rendszer a kapott függvényt le tudja egyszerűsíteni 11-edfokúra.

5. Fejezet

Implementációs részletek

5.1. overview

C++, CMake

5.2. Lineáris algebra

5.3. Polinom osztály

A polinomok szimbolikus reprezentációjához és manipulációjához létrehoztam a `Polynomial` osztályt.

5.3.1. Változók

Ahelyett, hogy egy polinom együtthatóit pozícionálisan határoznánk meg (tehát mondjuk egy `std::vector` n -edik eleme felel meg egy névtelen változó n -edik hatványához tartozó együtthatónak) a polinomunkat egymástól megkülönböztetett szimbolikus változókkal fejezzük ki. A változókat a `Variable` osztály reprezentálja. Ez a háttérben nem több, mint egy (`static`) számláló, ami egyedi azonosítót ad minden változó példánynak.

A változók megkülönböztetése lehetővé teszi, hogy műveleteket végezzünk el különböző paraméterekben értendő polinomok között anélkül, hogy a paraméterek nem kívánt módon „összemosódjanak”. Így például össze tudunk adni/szorozni két egyváltozós polinomot úgy, hogy az eredmény kétváltozós legyen

$$f(u)g(v) = h(u, v)$$

5.3.2. Struktúra

A `Variable` és `Polynomial` osztályok közti lépés a `PowerPermutation` osztály, ami változók hatványpermutációját reprezentálja, például x^2y^3z . A `PowerPermutation` háttérében egy `std::unordered_map` áll, ami változókhoz rendel pozitív egész számokat (`unsigned int`). Így egy `PowerPermutation` példány tetszőlegesen sok változóból állhat. Az egyértelműség érdekében, ha egy változóhoz 0-t rendelnénk, akkor azt az elemet töröljük (mondjuk $x^2y^0 = x^2$). A konstans permutációt (minden hatvány 0) egy üres map jelöli.

A `Polynomial` adatstruktúrája szintén egy `std::unordered_map`, ami `PowerPermutation`-ökhöz rendel együtthatókat. A `PowerPermutation - coeff_type` párokra a kódban `term`-ként (tag) hivatkozok.

5.3.3. Konstrukciós szintaxis

A `Variable`, `PowerPermutation` és `Polynomial` osztályok között definiálva van operator overloading bizonyos aritmetikai operációkra, amik lehetővé teszik a polinomok konstruálását egy intuitív és ismerős szintaxissal. Erre a fontosabb példák

- `operator^(Variable, unsigned int) ⇒ PowerPermutation` (hatványozás)
- `operator*(Variable, Variable) ⇒ PowerPermutation`
- `operator*(PowerPermutation, PowerPermutation) ⇒ PowerPermutation`
- `operator*(coeff_type, PowerPermutation) ⇒ Polynomial`
- `operator+(PowerPermutation, PowerPermutation) ⇒ Polynomial`
- `operator+(Polynomial, Polynomial) ⇒ Polynomial`
- ...

Egy példa polinom-konstrukció

```
Variable x, y;  
Polynomial<double> p = (x^3) - 2.0 * x * (y^2) + 1.0
```

A C++ szintaxisából adódó limitáció, hogy a szorzásjelet nem lehet elhagyni, illetve az operátor precedencia miatt a hatványozást szimbolizáló `^` jel használatakor zárójelezni kell.

5.3.4. Funkcionalitás

A `Polynomial` osztályra az alábbi függvények definiáltak:

5.3.4.1. Aritmetikai műveletek

- összeadás
- kivonás
- szorzás

5.3.4.2. Kiértékelés

Az `evaluate` függvény változó-érték párosok listáját várja (pontosabban `std::initializer_list`) majd tagonként, a `PowerPermutation`-ökbe behelyettesítve értékeli ki. Például

```
double result = p.evaluate<double>({{x, 3.0}, {y, -1.0}});
```

5.3.4.3. Derivált

A `derivative` függvény egy változót vár, majd az adott változó szerinti deriváltat adja vissza. Van egy alternatív verziója, ami változók listáját fogadja el, majd a deriváltakat egy ennek megfelelő `std::vector`-ban adja vissza.

5.3.5. Template

Az osztályban használok a C++ template funkcióját.

- Maga az osztály templatet az együtthatók típusára (`coeff_type`)
- A polinomot kiértékelő `evaluate` függvény templatet a bemeneti paraméter típusára (`input_type`)

Az `evaluate` függvény visszatérési értékét az alábbi template kifejezés írja le

```
using result_type = sum_type<product_type<
                        product_type<input_type>,
                        coeff_type>>;
```

A `sum_type` és `product_type` templatek két típus (vagy egy típus önmagával vett) összegét, illetve szorzatát (`operator+` / `operator*`) fejezik ki, és kódban így néznek ki

```
template <typename T, typename U = T>
using sum_type = decltype(std::declval<T>() + std::declval<U>());

template <typename T, typename U = T>
using product_type = decltype(std::declval<T>() * std::declval<U>());
```

Létrehoztam továbbá egy templatet structot (neve `I`, az „Identity” szót rövidítve) aminek két statikus tagja, `zero` és `one` tárolják azokat az értékeket, amik az adott típus nullája, illetve egységeként értendők. Ezt a structot specializálni kell a használt típusokra ahhoz, hogy bizonyos függvényeket (pl. `evaluate`) használni lehessen.

A templatek használata lehetővé teszi a felhasználást különböző típusokkal (pl float, double, complex, vektor). Nagybán növeli a rugalmasságot, hogy az együtthatók és a bemeneti paraméter külön vannak templatelve. Így például lehet polinomunk ami double típusú paramétert kap, de az együtthatói (így végül a `result_type`-ja is) vektor típusúak.

És mivel az `input_type`-ot csak az `evaluate` meghívásakor kell meghatározni, ugyanazt a `Polynomial` példányt különböző típusú paraméterekkel ki lehet értékelni.

Egy kifejezetten érdekes és hasznos következménye a template-es működésnek az, hogy két polinomot képesek vagyunk komponálni úgy, hogy az egyiket kiértékeljük a másikon.

5.3.6. Racionális függvények

A `Polynomial` mintájára létrehoztam a `Rational` osztályt a racionális függvények reprezentálására. Az osztály 2 `Polynomial`-ból áll, számláló és nevező. A `Polynomial`-hoz hasonlóan definiáltak rajta az aritmetikai műveletek (osztással bővítve), a kiértékelés, és a deriválás.

5.3.7. Tesztek

A `Variable`, `PowerPermutation`, `Polynomial` és `Rational` osztályok működésének ellenőrzéséhez írtam unit-teszteket, a GoogleTest könyvtárral.

5.4. grid

A feldolgozandó adatok és kiszámolt köztes értékek kezelésére létrehoztam a templatelt `grid` osztályt, ami egy $n \times m$ -es kétdimenziós tömböt reprezentál. A `grid`-et a C++23-ban megjelent többdimenziós subscript operátorral (`operator[]`) kényelmesen meg lehet indexelni.

A `grid` egy $n \times m$ nagyságú `std::vector`-t használ a háttérben, iterátora az `std::vector` iterátora.

5.5. range2d

Az egymásba ágyazott ciklusok elkerülése végett létrehoztam a `range2d` osztályt, illetve ennek iterátorát, az `index2d`-t. Az előbbi egy n és m számpár (`size_t`), az utóbbi egy i, j pár, illetve egy harmadik m szám.

Az `index2d` inkrementálás operátora (`operator++`) úgy növeli i -t és j -t, hogy végiglépeghessen egy m hosszú sorokból álló kétdimenziós tömb indexein. A dereferálás operátora (`operator*`) pedig visszaadja i -t és j -t.

Így az alábbi szintaxissal végig lehet iterálni egy $n \times m$ -es tömb indexein

```
for (auto [i, j] : range2d{n, m}) {  
    ...  
}
```

5.6. Megjelenítés

A program a létrejött racionális patcheket kiértékeli $n \times n$ darab (u, v) értékpárban, a pontokat egy `grid`-ben tárolja. Ezután a felületeket exportálja a népszerű *obj* fájlformátumban. A kiírás történhet normálvektorokkal együtt, vagy nélkülük. Lehetséges továbbá több patchet egyszerre, ugyanabba a fájlba írni. Ez esetben a program a patcheket külön group-okba rakja a fájlban belül. Az *obj* fájlok megjelenítésére sok alkalmazás képes, én a ParaView-t használtam.

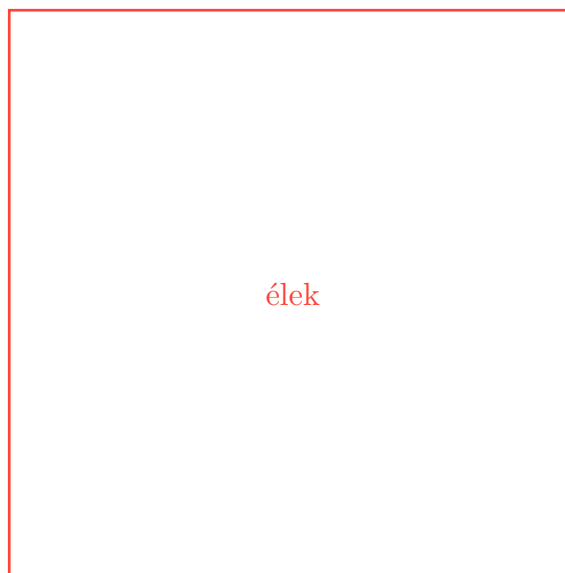
6. Fejezet

Eredmények

6.1. Szabad paraméterek

Már korábban is említettük, hogy a módszer elvi alapjai nem határoznak meg egy egyedi felületet bizonyos adatpontokra. A folyamat során hozhatunk pár szabad döntést, amik befolyásolják az eredményt. Ezek az eredmények mind megfelelnek bizonyos alapvető elvárásoknak (a felület folytonos, interpolál a megadott pontok és normálvektorok között) de adott esetben komolyabb esztétikai hibák és nem kívánt anomáliák jelenhetnek meg.

A legfőbb ezek közül az amikor élek jelennek meg a felületen, az egy vonal mentén hirtelen megfordulnak. Ilyen lehet egyrészt a patchek találkozásánál (maguk a patchek külön-külön simák, de a határokon ellentétes irányba tartanak) de akár a patcheken belül is.





izotróp térben élek nélkül

6.2. Végponti deriváltak

A legkézenfekvőbb szabad paraméter az izotróp térben megszabott végponti deriváltak nagysága. Minél nagyobb a szorzó, a felület annál kaotikusabban viselkedik. Extrém mértékben kitér, sokszor önmagára hajlik. Ezzel szemben a kicsi szorzó hatására a felület „merev” lesz. Az adatpontok környékén lapos, a pont-normálvektor pár által meghatározott síkot közelíti. A lapok találkozásánál pedig szintén élek alakulnak ki.



magas szorzó



alacsony szorzó

Nem muszáj ragaszkodnunk az interpolálandó deriváltak irányát meghatározó heurisztikához sem. A vektorok egy adott síkon belül (melynek normálvektora a folyamat során kiszámolt \mathbf{v}) bármerre mutathatnak. És bár a heurisztikánk az izotróp térben nagyon szép, „egyenest” határgörbékhez vezet, ez a tulajdonság nem feltétlenül marad meg visszatranszformálás után. Előfordulhat, hogy egy patch túlságosan be-, vagy kilóg a sarokpontok közti „keretből”. Az itt használt módszernek egy összetettebb alternatívája lehetne például az, hogy a derivált-heurisztikát még a primális térben alkalmazzuk, a deriváltakat transzformáljuk az izotróp térbe, majd ezeket vetítjük le a megfelelő síkra.

6.3. Adatpontok előzetes transzformálása

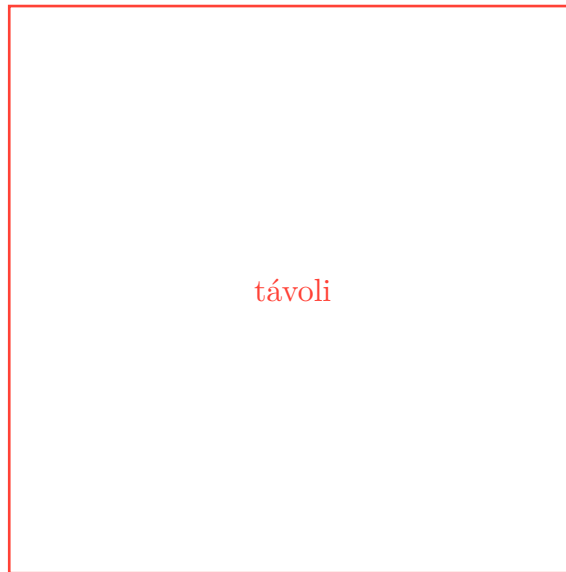
A módszer egy nem kifejezetten szerencsés tulajdonsága az, hogy nem lineáris (illetve pontosabban affin). Ez alatt az értendő, hogy amennyiben az adatpontokat eltoljuk, forgatjuk, vagy skálázzuk, a létrejövő felület nem az eredeti hasonlóan transzformált verziója lesz, attól drasztikus mértékben eltérhet.

Ez egy újabb dimenziót ad a felület igazításához. Adatpontjainkat áttanszformálhatjuk a módszer végrehajtása előtt, majd az eredményt visszatranszformálhatjuk, hogy egy új interpolációt kapjunk. Ezzel a lehetőséggel bizonyos esetben szükséges is élnünk, hogy felületünk megfeleljen az esztétikai elvárásainknak.

6.3.1. Eltolás

Ha az adatpontjaink túl távol vannak az origótól, a patchek között élek alakulnak ki. Ezek az élek látványosan eltérnek a patchen belüli élektől, amiket például a nagy izotróp végponti deriváltak okoznak. Itt a patchek tesznek egy kanyart a határgörbe közelében, és „fordítva” érnek be.

Érdemes ezért az adatpontokat úgy eltolni, hogy az origó a közepükön legyen.

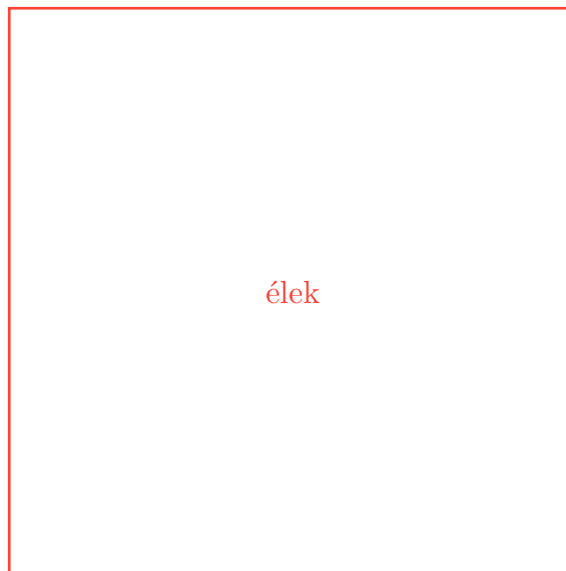


távoli

6.3.2. Skálázás

Még egy egyszerű transzformáció az arányos skálázás, ergo az összes adatpontot beszorozzuk ugyanazzal a számmal (a normálvektorokat békén hagyjuk, azok mindenképp egység hosszúságúak). Egy magasabb szorzó „kifeszíti” a felületet. Az éleket ki tudja simítani, de cserébe a felület szélei „behorpadnak”, a határgörbék egyre inkább kitérnek az origó irányába.

A lefele skálázás szépen kiegyenesíti a határgörbéket, azonban a patchek határai környékén éleket alakíthat ki.



élek



kifeszítve



kicsi skála

6.3.3. Forgatás

6.3.4. Irányvektorok állítása előzetes transzformálás helyett

6.3.5. Példa

7. Fejezet

Végszó

7.0.1. Módszer megítélése

7.0.2. További kutatás