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

2025-05-20

Tartalomjegyzék

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, 2025-05-20

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 alapművelettel, 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 őket kerülni 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 többit 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

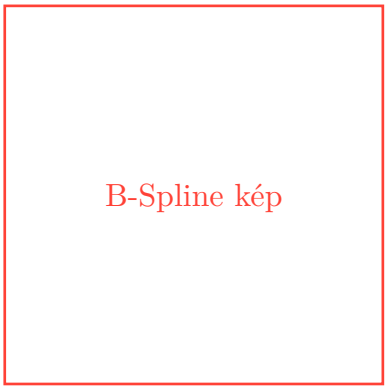
A B-Spline (Basis-Spline) darabonként definiált bázisfüggvényekből áll, melyeknek szegmenseit úgynevezett „csomópontok” (knots) választják el $(t_0, t_1 ... t_m)$. A bázisfüggvényeket A Cox-de Boor képlettel tudjuk kiértékelni:

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

$$B_{i,n}(t) := \frac{t - t_i}{t_{i+n} - t_i} + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}}$$

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 vesztésével 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 pitagoraszi 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 egységhosszúságú normálvektor. Normálvektort egyszerűen kapunk úgy, hogy merőlegest állítunk a deriváltra. De ahhoz, hogy ez egységhosszúságú 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 szélsebességek elemzésére. A „pitagoraszi” alatt pedig a Pitagorasz tételt értjük egy derékszögű háromszög oldalai közti összefüggésre. Összesítve, PH görbe alatt olyan parametrikus görbéket 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) számhármasok, melyekre teljesül, hogy $a^2 + b^2 = c^2$, léteznek $a(t)$, $b(t)$, $c(t)$ pitagoraszi *polinom*hármasok is. Ilyen polinomhármasokat tudunk generálni is, méghozzá ugyanazzal a képlettel, amivel a számhármasokat.

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

$$\begin{aligned} a(t) &= m^2(t) - n^2(t) \\ b(t) &= 2m(t)n(t) \\ c(t) &= m^2(t) + n^2(t) \end{aligned}$$

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 egységhosszúságú 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", $\left(n_x, n_y, n_z, h\right)$ 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 érintősík 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 leírhassuk 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 írnunk néhány azonosságot.

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

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

Így $h(\mathbf{s})$ deriváltja

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

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

$$\frac{d}{d\mathbf{s}}\mathbf{n}\cdot\mathbf{n} = 2\,\mathbf{n}^T\frac{d\mathbf{n}}{d\mathbf{s}} = \frac{d}{d\mathbf{s}}1 = \mathbf{0}$$

$$\Rightarrow \frac{d\mathbf{n}}{d\mathbf{s}}^T\,\mathbf{n} = \mathbf{0}$$

$h\mathbf{n}$ 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

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

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

$$\frac{dh}{d\mathbf{s}}^T = \frac{d\mathbf{n}}{d\mathbf{s}}^T\frac{d\mathbf{n}}{d\mathbf{s}}\cdot\mathbf{r}$$

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

Tehát

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

4.3. Izotróp tér

Az egységhosszúságú 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

$$\mathbf{x}^T \frac{d\mathbf{n}}{d\mathbf{s}} = \frac{dh}{d\mathbf{s}}$$
$$\mathbf{x}^T \frac{d\mathbf{n}}{d\mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{s}} = \frac{dh}{d\mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{s}}$$
$$\underbrace{\left(\mathbf{x}^T \frac{d\mathbf{n}}{d\mathbf{y}} - \frac{dh}{d\mathbf{y}} \right)}_{\mathbf{v}} \frac{d\mathbf{y}}{d\mathbf{s}} = \mathbf{0}$$

Ahol

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

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 rendelendő 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 $\boldsymbol{\gamma}_{i,j}$ a felület u szerinti deriváltját az $\mathbf{a}_{i,j}$ pontban, $\boldsymbol{\delta}_{i,j}$ pedig a v szerinti deriváltat ugyanitt.

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

$$\boldsymbol{\gamma}^*_{0,j} = \mathbf{a}_{1,j} - \mathbf{a}_{0,j} \qquad \boldsymbol{\delta}^*_{i,0} = \mathbf{a}_{i,1} - \mathbf{a}_{i,0}$$
$$\boldsymbol{\gamma}^*_{n,j} = \mathbf{a}_{n,j} - \mathbf{a}_{n-1,j} \qquad \boldsymbol{\delta}^*_{i,n} = \mathbf{a}_{i,n} - \mathbf{a}_{i,n-1}$$

Egyébként a két vektort átlagoljuk

$$\boldsymbol{\gamma}^*_{i,j} = \frac{\mathbf{a}_{i+1,j} - \mathbf{a}_{i-1,j}}{2} \qquad \boldsymbol{\delta}^*_{i,j} = \frac{\mathbf{a}_{i,j+1} - \mathbf{a}_{i,j-1}}{2}$$

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

$$\boldsymbol{\gamma}_{i,j} = \boldsymbol{\gamma}^*_{i,j} - \frac{\mathbf{v} \cdot \boldsymbol{\gamma}^*_{i,j}}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v} \qquad \boldsymbol{\delta}_{i,j} = \boldsymbol{\delta}^*_{i,j} - \frac{\mathbf{v} \cdot \boldsymbol{\delta}^*_{i,j}}{\mathbf{v} \cdot \mathbf{v}} \mathbf{v}$$

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 ($\mathbf{c}_0(u)$, $\mathbf{c}_1(u)$, $\mathbf{d}_0(v)$, $\mathbf{d}_0(v)$), 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

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

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

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

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

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

A határgörbékhez használjunk Hermite interpolációt

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

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

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

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

$$\frac{d\mathbf{y}}{du} = \begin{pmatrix} F_0'(u) & G_0'(u) & F_1'(u) & G_1'(u) \end{pmatrix} \begin{pmatrix} \mathbf{a}_{00} & \boldsymbol{\delta}_{00} & \mathbf{a}_{01} & \boldsymbol{\delta}_{01} \\ \boldsymbol{\gamma}_{00} & \mathbf{0} & \boldsymbol{\gamma}_{01} & \mathbf{0} \\ \mathbf{a}_{10} & \boldsymbol{\delta}_{10} & \mathbf{a}_{11} & \boldsymbol{\delta}_{11} \\ \boldsymbol{\gamma}_{10} & \mathbf{0} & \boldsymbol{\gamma}_{11} & \mathbf{0} \end{pmatrix} \begin{pmatrix} F_0(v) \\ G_0(v) \\ F_1(v) \\ G_1(v) \end{pmatrix}$$
$$\frac{d\mathbf{y}}{dv} = \begin{pmatrix} F_0(u) & G_0(u) & F_1(u) & G_1(u) \end{pmatrix} \begin{pmatrix} \mathbf{a}_{00} & \boldsymbol{\delta}_{00} & \mathbf{a}_{01} & \boldsymbol{\delta}_{01} \\ \boldsymbol{\gamma}_{00} & \mathbf{0} & \boldsymbol{\gamma}_{01} & \mathbf{0} \\ \mathbf{a}_{10} & \boldsymbol{\delta}_{10} & \mathbf{a}_{11} & \boldsymbol{\delta}_{11} \\ \boldsymbol{\gamma}_{10} & \mathbf{0} & \boldsymbol{\gamma}_{11} & \mathbf{0} \end{pmatrix} \begin{pmatrix} F_0'(v) \\ G_0'(v) \\ F_1'(v) \\ G_1'(v) \end{pmatrix}$$

4.6. Visszatranszformálás

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` hátterében 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 inuitív 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 használatatakor 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 értékel 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 használom a C++ template funkcióját.

- Maga az osztály templatelt az együtthatók típusára (`coeff_type`)
- A polinomot kiértékelő `evaluate` függvény templatelt 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 templatelt 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 értendőek. 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). Nagyban 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 másikban.

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 végiglépegessen 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

6. Fejezet

Eredmények