# BONYOLULT GEOMETRIÁJÚ FELÜLETEK MATEMATIKAI REPREZENTÁCIÓJÁNAK ELŐÁLLÍTÁSA MÉRÉSI HIBÁKKAL TERHELT DISZKRÉT ADATOKBÓL


Készítette: Éles András és Glózik Zoltán

Témavezető: Dr Renner Gábor, MTA SZTAKI

Eötvös Loránd Tudományegyetem
Természettudományi Kar
Informatika Tanszékcsoport


Budapest, 1998. június 17.

# Tartalomjegyzék

# 1. fejezet

# Bevezetés

A geometriai modellezés feladata a térbeli testek és szabad formájú felületek matematikai reprezentációjának előállítása. Ennek a matematikai reprezentációnak a használatával lehetőség nyílik a térbeli testek és felületek vizsgálatára, azok jellemzésére különböző természetes szempontok szerint. A tárgyak számítógépen történő ábrázolásának egyik lehetősége lehetne, hogy a tárgyak vagy felületek pontjait tároljuk el, valamilyen sűrű mintavételezéssel. Ehhez azonban nagyon sok memória kellene, a kapott felületet nem tudnánk tetszőleges pontossággal visszakapni, és a felületet nem tudnánk vizsgálni. Egy matematikai reprezentációval adott felület már könnyebben kezelhető, a felület eltárolásához szükséges memóriaméret kisebb és könnyen végrehajthatunk módosításokat a felületen.

Az ipari alkalmazásokban használt szerszámgépek a tárgyakat egy matematikai reprezentáció alapján készítik el. Ez a matematikai reprezentáció a mérnökök és formatervezők munkájának eredményeként áll elő. Gyakran előforduló eset, hogy már valamilyen meglévő tárgyat akarunk legyártatni, és a tárgyról semmilyen terv nem áll rendelkezésre. Például egy autó karosszériájának tervezésekor a formatervezők először valamilyen természetes anyagból előállítják a karosszériát, majd komoly mérnöki munkával kell elkészíteni ugyanennek a felületnek a matematikai reprezentációját, mivel a gyártás csak ezzel végezhető el. Ez úgy történik, hogy egy letapogató berendezéssel valamilyen rendszer szerint mintát vesznek a megformázott karosszériáról, majd kapnak egy részben struktúrált pontfelhőt, ami még mérési hibákat is tartalmaz. Ebből kell előállítani azokat a felület darabokat, amelyeknek nagyon kemény követelményeknek kell megfelelniük. Vizsgálják a felület simaságát, több rendbeli folytonosságát, a fénytörését, szépségét. Ennek a munkafolyamatnak nagyon sok lépése még ma is „kézzel" történik, gyakran próbálgatással. Ennek a tervezési folyamatnak a neve *reverse engineering*. A folyamat a következő lépésekből áll:

1. *A tárgy letapogatása.* Tisztában kell lennünk a mérési hibák mértékével, meg kell határozni a mintavételezés „finomságát". Minél sűrűbben

helyezkednek el a mérési pontok, annál több adatot kell eltárolnunk, amihez több memóriára van szükség és a felület előállítása is több időt vesz igénybe, viszont a felület apróbb eltéréseit is érzékelni tudjuk.

2. *Mérési hibák csökkentése.* A kapott adathalmazon a feldolgozás előtt lefuttathatunk egy kezdeti szűrést. A mérési hiba miatt előfordulhat, hogy egy pont viszonylag távol kerül a többitől. Ezeket a pontokat célszerű eltüntetni a ponthalmazból, vagy módosítani őket úgy, hogy csak kis mértékben térjenek el a környezetüktől. Ha ezek a nem valós adatokat tükröző pontok bennmaradnak a ponthalmazban, akkor később esetleg nagy mértékben torzíthatják az előállított felületet.

3. *Kezdeti paraméterek meghatározása.* Ki kell választani egy megfelelő reprezentációt a felülethez, majd meg kell határozni ennek a reprezentációnak azokat a paramétereit, amelyeket konstansként tekintünk az approximációs lépés során. Ezeknek a paramétereknek a meghatározása automatikusan nehezen megoldható, gyakran a tapasztalatok és a próbálgatás adják a megoldást.

4. *Approximáció.* A választott matematikai reprezentáció szabad paramétereinek megválasztása oly módon, hogy az *valamilyen* szempont szerint a legjobb közelítését adja az eredeti felületnek. Fontos követelmény, hogy az előálló felület pontjai az eredeti felület pontjainak a közelében helyezkedjenek el, és biztosítani kell azt is, hogy az előálló felület megfeleljen a simasági követelményeknek.

5. *Paraméter korrekció.* Az approximáció után kiderülhet, hogy nem megfelelő kötött paramétereket határoztunk meg kezdetben. Ekkor ezek módosítása után ismét el kell végezni az approximációt.

A másik nagy gyakorlati alkalmazás az offset felületek előállítása. Egy adott felülethez létrehozott offset felület minden pontja ugyanakkora, előre rögzített távolságra helyezkedik el a kiinduló felülettől. Az offset felület meghatározására például azért van szükség, mert a szerszámgép marófejének a középpontja a nyersanyagtól valamilyen rögzített távolságra halad. Ekkor a legyártandó felület reprezentációjának ismerete önmagában kevés, abból meg kell határoznunk azt az offset felületet, amelyen a marófej középpontjának végig kell haladnia. Az offset felület gyakran csak a kiinduló felület reprezentációjánál bonyolultabb reprezentációval adható meg, így az analitikus előállítás nem ad megfelelő eredményt. Az offset felületet a korábban leírt lépésekhez hasonlóan meg tudjuk határozni, ha mintát veszünk az eredeti felületről, a kapott pontokat eltoljuk a felületre merőleges irányban, majd approximációt végzünk. Ennek a problémának a megoldása általában könnyebb, mert az eredeti felületnek a reprezentációja már rendelkezésre áll és mérési hibákkal sem kell törődnünk.

A szakdolgozat célja a felületek előállítása során felmerülő részproblémák megoldására használható algoritmusok számítógépes megvalósítása és azok vizsgálata.

# 2. fejezet

# Felületek reprezentációja

A célunk egy olyan reprezentáció használata, amelynek segítségével a bonyolult geometriájú felületek matematikailag „jól” kezelhetők. Ez azonban mást és mást jelenthet, attól függően, hogy milyen célból vizsgáljuk a felületeket. Más reprezentációt kell választanunk akkor, ha az alakzatok gyors megjelenítése a cél, és más reprezentációt választanánk, ha felületek közös pontjait akarnánk meghatározni. Elképzelhető, hogy egy adott feladatot nem is tudunk megoldani az egyik reprezentációval adott felületen, vagy csak nagyon rossz hatékonysággal, míg más reprezentációk esetén jól megoldhatók. Három különböző reprezentációt vizsgálunk meg több szempont szerint, de számunkra csak az egyik lesz igazán megfelelő.

## 2.1. A reprezentációk három csoportja

### Explicit függvény

Ez a legegyszerűbb felület megadási forma, a reprezentáció egy $f\colon \mathbb{R}^2 \mapsto \mathbb{R}$ függvénnyel adott. Pédául egy 5 egység sugarú félgömb ábrázolása a következőképpen történne: $f(x,y) = \sqrt{25 - x^2 - y^2}$.

Kiértékelhetőség szempontjából akkor jó ez a megadási mód, ha a $z$ értéket akarjuk meghatározni az $x$ és $y$ ismeretében. Más esetben egyenletet kell megoldani, ami nem is biztos, hogy egyértelmű. Nehézkes annak a meghatározása is, hogy egy adott $(x,y,z)$ térbeli pont rajta van-e a felületen. Nem adhatunk meg olyan felületeket, amelyek két különböző pontjához ugyanaz az $(x,y)$ pont tartozik. Az explicit megadási módszert a gyakorlatban nem is alkalmazzák geometriai modellezési célokra.

### Implicit függvény

Egy felület implicit függvénnyel történő ábrázolásakor a felület pontjait egy $G(x,y,z) = 0$ alakú egyenlet megoldásai adják. Egy 5 egység sugarú gömb implicit egyenlete a következő lehetne: $x^2 + y^2 + z^2 = 25$.

A felület pontjainak meghatározása szempontjából ez a reprezentáció nagyon rossz, mivel egyenleteket kell megoldani a hiányzó koordináták megtalálásához. De nagyon könnyen ellenőrizhető egy adott térbeli pont felülethez tartozása, ehhez csak a $G$ függvény kiértékelésére van szükség az adott pontban. Ezzel a reprezentációval sokkal többféle felület állítható elő, mint egy explicit függvénnyel.

### Parametrikus felület

Egy felület parametrikus reprezentációja egy $f\colon \mathbb{R}^2 \mapsto \mathbb{R}^3$ függvény, ami a paramétertér pontjait képezi le térbeli pontokra. A felületet a függvény értékkészletében szereplő pontok adják. Egy 5 egység sugarú gömböt a következő parametrikus felület ad meg:

$$u, v \in [0, 2\pi],\ f(u, v) = \begin{bmatrix} 5\cos(v)\cos(u) \\ 5\cos(v)\sin(u) \\ 5\sin(v) \end{bmatrix},$$

ahol $u$ és $v$ a felület paraméterei. Ennek a reprezentációnak az előnye, hogy a paramétertér bejárásával a felület pontjait meg tudjuk határozni, csak egy explicit függvényt kell kiértékelni. Nehéz annak a kérdésnek a megválaszolása, hogy egy adott pont rajta van-e a felületen. Ha már adott egy parametrikus felület, akkor könnyű rajta lineáris transzformációkat és eltolásokat alkalmazni. A gyakorlati alkalmazások 80%-ában ezt a reprezentációt használják, a maradék 20%-ban pedig az implicit megadási formát, az explicit módszert szinte egyáltalán nem használják felületek ábrázolására. Mi is parametrikus felületet szeretnénk előállítani a mért adatokból.

## 2.2. Differenciálgeometriai összefoglaló

Ebben a szakaszban röviden áttekintjük a parametrikus felületekkel kapcsolatos alapvető definíciókat. Felületeket jellemző mennyiségeket vezetünk be. Feltételezzük, hogy az olvasó tisztában van a parametrikus görbékkel kapcsolatos alapvető definíciókkal.

**1. Definíció.** *Az* $r\colon [0,1]^2 \mapsto \mathbb{R}^3$ *függvény* elemi felület, *ha kölcsönösen egyértelmű és folytonos. A* felület *olyan ponthalmaz, amely összerakható elemi felületekből.*

A továbbiakban feltesszük, hogy az $r\colon [0,1]^2 \mapsto \mathbb{R}^3$ elemi felület legalább egyszer folytonosan deriválható és $r_u(u,v) \nparallel r_v(u,v)$. Ez a feltétel biztosítja számunkra, hogy a felület minden pontjában létezik normális vektor, a felületre merőleges irány.

**2. Definíció.** *Legyen* $u, v\colon [0,1] \mapsto [0,1]$ *folytonos függvények. Ekkor a* $g(t) = r(u(t), v(t)), t \in [0,1]$ *függvény az* $r$ *felület egy* felületi görbéje. *Az* $m(t) = r(t, v)$ *és* $n(t) = r(u, t)$ *felületi görbéket* paraméter vonalaknak *nevezzük.*

A felületek ábrázolásának egyik gyakori módja, hogy néhány paramétervonalat jelenítenek meg. Ezzel a felületnek egy drótvázas ábrázolását kapjuk. A felület egy adott $r(u, v)$ pontján átmenő felületi görbék érintői egy síkban vannak, ez a sík az $r$ felület $(u, v)$ pontbeli *érintő síkja.*

**3. Definíció.** *Az* $r$ felület normálisa *egy adott* $(u, v)$ *pontban a felület* $(u, v)$ *pontjához tartozó érintő síkra merőleges egységvektor:*

$$m = \frac{r_u \times r_v}{|r_u \times r_v|}.$$

A felületek pontjaiban egy jellemző tulajdonság, hogy ott mennyire hajlik a felület egy adott felületi görbe mentén.

**4. Definíció.** *Egy felületi görbe adott* $P$ *pontjához tartozó érintő irány és a felület* $P$ *pontjabeli felületi normálisa által meghatározott sík a felületi görbéhez tartozó* $P$ *pontbeli* normálsík, *egy normálsík által a felületből kimetszett görbe a* normálmetszet. *Egy normálmetszethez tartozó görbület a* normálgörbület, *egy adott ponthoz tartozó normálgörbületek minimumát és maximumát* főgörbületeknek, *a hozzájuk tartozó irányokat pedig* főirányoknak *nevezzük.*

**5. Definíció.** Első alapmennyiségek*:*

$$E = r_u^2, \quad F = \langle r_u, r_v \rangle, \quad G = r_v^2.$$

Első alapmátrix*:*

$$\mathcal{G} = \begin{bmatrix} E & F \\ F & G \end{bmatrix}.$$

A $g(t) = r(u(t), v(t))$ felületi görbe ívhossza:

$$s = \int_0^1 |\dot{g}|\, dt = \int_0^1 \sqrt{r_u^2 \dot{u}^2 + 2 \langle r_u, r_v \rangle \dot{u}\dot{v} + r_v^2 \dot{v}^2}\, dt.$$

A felületi görbe ívhossza felírható az első alapmennyiségekkel is a következő módon:

$$s = \int_0^1 \sqrt{E\dot{u}^2 + 2F\dot{u}\dot{v} + G\dot{v}^2}\, dt.$$

Egy $r$ felület felszíne az első alapmennyiségekkel felírva:

$$\int_0^1 \int_0^1 \sqrt{EG - F^2}\, dudv.$$

**6. Definíció.** *Az $r$ kétszer folytonosan deriválható felület $(u,v)$ pontbeli* második alapmennyiségei*:*

$$L = \langle r_{uu}, m\rangle, \quad M = \langle r_{uv}, m\rangle, \quad N = \langle r_{vv}, m\rangle,$$

*ahol $m$ az $r(u,v)$ ponthoz tartozó felületi normális. A* második alapmátrix*:*

$$\mathcal{D} = \begin{bmatrix} L & M \\ M & N \end{bmatrix}.$$

A felületek vizsgálata során két, a főgörbületekkel kapcsolatos mennyiség fontos szerepet fog játszani. Ezek azt mutatják meg, hogy a felület egy adott pontban mennyire görbül.

**7. Definíció.** *Legyenek a felület egy adott $P$ pontjában a főgörbületek $\mathcal{K}_1$ és $\mathcal{K}_2$. Ekkor*

$$\begin{aligned} \mathcal{K} &= \mathcal{K}_1\mathcal{K}_2 && \text{Gauss görbület,} \\ \mathcal{H} &= \tfrac{\mathcal{K}_1+\mathcal{K}_2}{2} && \text{Minkowski görbület.} \end{aligned}$$

A következő tétel megadja a Gauss- és Minkowski-görbületek valamint az első és második alapmennyiségek közötti kapcsolatot adja meg.

**1. Tétel.** *Legyen $g(t) = r(u(t), v(t))$ az $r$ felület egy felületi görbéje. Tegyük fel, hogy a $t_0$ pontban a felületi görbe simulósíkja és a felület érintősíkja nem esik egybe. Legyen $m$ a felület $g(t_0)$-beli normálisa, $n$ pedig a $g$ görbe $t_0$ paraméterértékéhez tartozó főnormálisa. Ekkor a felület e pontbeli görbülete:*

$$\mathcal{K} = \frac{1}{\langle m,n\rangle}\frac{L\dot{u}^2 + 2M\dot{u}\dot{v} + N\dot{v}^2}{E\dot{u}^2 + 2F\dot{u}\dot{v} + G\dot{v}^2} = \frac{1}{\langle m,n\rangle}\frac{c^T\mathcal{D}c}{c^T\mathcal{G}c},$$

*ahol $c = [\dot{u},\ \dot{v}]$.*

Ha a normálgörbületeket akarjuk meghatározni, akkor a felület adott pontbeli érintősíkja és a megfelelő felületi görbe simulósíkja egymásra merőleges, vagyis $\langle m,n\rangle = 1$. Ekkor az $\frac{1}{\langle m,n\rangle}$ együttható elmaradhat a képletből.

**2. Tétel.** *A főgörbületeket az alábbi homogén egyenlet megoldásával kaphatjuk meg:*

$$\begin{vmatrix} L - \mathcal{K}_i E & M - \mathcal{K}_i F \\ M - \mathcal{K}_i F & N - \mathcal{K}_i G \end{vmatrix} = 0,$$

*a főirányokat pedig a*

$$\begin{aligned} (L - \mathcal{K}_i E)\dot{u}_i + (M - \mathcal{K}_i F)\dot{v}_i &= 0 \\ (m - \mathcal{K}_i F)\dot{u}_i + (n - \mathcal{K}_i G)\dot{v}_i &= 0 \end{aligned}$$

*egyenletrendszer határozza meg ($i = 1, 2$).*

A következő tétel azt mondja ki, hogy a Gauss- és Minkowski-görbületek meghatározásához nem szükséges kiszámolnunk a konkrét főgörbületeket, az első és második alapmennyiségek ismerete is elegendő hozzá.

**3. Tétel.**

$$\begin{aligned} \mathcal{K} &= \frac{LN - M^2}{EG - F^2} \\ \mathcal{H} &= \frac{EN - 2FM + GL}{EG - F^2} \end{aligned}$$

## 2.3. Felületek polinomiális reprezentációja

A parametrikus görbék és felületek esetében fontosak azok a reprezentációk, amelyeknél a paramétertér pontjait valamilyen polinom vagy polinomok képezik le a tér pontjaira. Ezeknek a reprezentációknak az előnye a könnyű kezelhetőség és a geomteriai tulajdonságok könnyű meghatározása. A görbék tárolásához elegendő az együtthatóikat megjegyezni és a kiszámításukhoz csak a négy alapművelet szükséges. A legfontosabb polinomiális görbe reprezentációk és tulajdonságaik:

1. *Bézier görbe:* A $c_i \in \mathbb{R}^3$ együtthatókhoz (kontrollpontok) tartozó Bézier görbének a következő polinomot nevezzük:

$$p(t) = \sum_{i=0}^{k} c_i \binom{k}{i} t^i (1-t)^{k-i},$$

   ahol $t$ a görbe paramétere. A Bézier görbe fokszáma a kontrollpontok száma+1 lesz, ami sok kontrollpont esetén nagy lehet és így a görbe pontjainak meghatározása sok számítást igényel. A polinom akárhányszor folytonosan differenciálható. A görbe pontjai a kontrollpontok által meghatározott poligonon belül helyezkednek el. Minden kontrollpont hatással van az egész görbére, viszont az $i$. kontrollpont a $t = \frac{i}{n}$. paramétertérbeli pontban fejti ki a legerősebb hatást. Ezzel a reprezentációval azonban nem tudunk minden görbét megadni, például a gyakorlatban gyakran előforduló gömb és kúp nem adható meg polinomiális Bézier reprezentációban. A görbék általánosításaként a Bézier felületeket is definiálhatjuk.

2. *B-Spline görbe:* Ha $N_i^{(n)}$ $n$. fokú szakaszonként polinomiális függvény, akkor a

$$p(t) = \sum_{i=0}^{k} c_i N_i^{(n)}(t)$$

függvényt B-Spline görbének nevezzük, ahol a $c_i$-k a kontrollpontok, $t$ a paramétertér egy pontja. A B-Spline görbéket több alacsony fokszámú polinomból rakjuk össze, így a görbe szakaszonként polinomiális. A B-Spline reprezentáció előnye a Bézier-vel szemben, hogy nem kell magas fokszámú polinomokkal számolnunk. A polinomiális szakaszok csatlakozási pontjaiban nem tudunk végtelen differenciálhatóságot biztosítani, de a bázisfüggvények fokszámának megfelelő megválasztásával el tudjuk érni a gyakorlat számára elegendő eredményt. Számunkra a B-Spline görbék és felületek lesznek megfelelők, mivel a gyakorlat szempontjából fontos felületek reprezentálására alkalmasak, gyorsan kiszámíthatók, könnyen differenciálhatók. A B-Spline függvényekről a következő szakaszban részletesen lesz szó.

3. *Racionális B-Spline görbék (NURBS):* A

$$p(t) = \frac{\sum_{i=0}^{k} c_i w_i N_i^{(n)}(t)}{\sum_{i=0}^{k} w_i N_i^{(n)}(t)}$$

racionális függvényt NURBS (Non Uniform Rational B-Splines) görbének nevezzük, ahol $t$ a paraméter, $c_i$-k a kontrollpontok és $w_i$-k súlyok. A NURBS görbék a B-Spline-ok általánosításai. Ez a reprezentáció már kör és kúp megadására is alkalmas.

### 2.3.1. B-Spline görbék

Régen a hajókészítés során gyakran felmerülő probléma volt olyan görbéknek a meghatározása, amelyek átmennek bizonyos előre meghatározott pontokon és minimalizálják a hajlításból származó energiát. Ezt a feladatot úgy oldották meg, hogy egy hajlékony rudat fektettek át az előre meghatározott pontokon. A „spline" szó valójában ennek a mechanikus tervezési eszköznek a nevéből származik.

Ebben a szakaszban áttekintjük, hogy mik azok a B-Spline görbék és miért fontosak számunkra. Meg fogjuk vizsgálni a tulajdonságaikat és paraméterezhetőségüket. A B-Spline görbéket fogjuk felhasználni felületek létrehozásához.

A továbbiakban legyen $S\colon [0,1] \mapsto \mathbb{R}^3$ egy parametrikus görbe. Azt szeretnénk elérni, hogy az $S$ görbe minél magasabb rendben folytonos legyen, gyorsan kiszámíthatóak legyenek a pontjai és lehetőleg kevés memória felhasználásával el lehessen tárolni számítógépben.

Azt a megoldást választjuk, hogy több azonos – alacsony – fokszámú polinomból rakjuk össze az $S$ parametrikus görbét. Az elemi polinomok értelmezési tartományainak az uniója a $[0,1]$ intervallum lesz, és mindegyik polinom a $[0,1]$ tartománynak csak egy kis részén lesz nullától különböző. Ezzel elérjük, hogy a görbe adott paraméterpontbeli kiértékelése még nagyon sok polinomból álló görbék esetén is gyorsan megvalósítható lesz. Minden

egyes pontban elég lesz csupán néhány polinom helyettesítési értékét meghatározni, mivel a többi polinom értéke úgyis 0 lenne. Ezekből a polinomokból állnak majd elő a bázisfüggvények. Ezzel a módszerrel a görbe tárolása kevés memória felhasználásával is megvalósítható lesz, mivel csak a referenciapontok eltárolására lesz szükség, az egyes polinomokat nem kell külön megjegyezni.

Az $S$ függvényt a következő alakban szeretnénk előállítani:

$$t \in [0,1],\ S(t) = \sum P_i N_i(t),$$

ahol az $N_i$-k szakaszonként polinomiális függvények és egy kis tartományt kivéve mindenhol nullával egyenlők.

**8. Definíció.** *A $t_0, t_1 \dots t_m, t_i \in \mathbb{R}$ sorozatot* knot-vektornak *nevezzük, ha $t_0 \leq t_1 \leq \dots \leq t_m$. Ha $t_1 - t_0 = t_2 - t_1 = \dots = t_m - t_{m-1}$, akkor* egyenletes knot eloszlásról *beszélünk.*

**9. Definíció.** *Legyen $t_0, \dots, t_m$ egy knot-vektor, ekkor definiáljuk a 0-ad rendű B-Spline bázisfüggvényt a következő módon:*

$$i \in [0, m-1],\ N_i^{(0)}(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{különben} \end{cases}.$$

*Az $0 < n$-edrendű bázisfüggvény definíciója rekurzív:*

$$i \in [0, m-n-1],\ N_i^{(n)}(t) = \frac{t - t_i}{t_{i+n} - t_i} N_i^{(n-1)}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} N_{i+1}^{(n-1)}(t).$$

A bázisfüggvények alsó indexe azt jelzi, hogy melyik knot-vektorbeli pontnál kezdődik a függvény, a felső index pedig a bázisfüggvény fokszáma.

A görbék és felületek vizsgálatánál a B-Spline-ok deriváltjainak meghatározását a bázisfüggvények deriváltjaira kell majd visszavezetnünk. Ezért hasznos a következő tétel:

**4. Tétel.** *A $N_i^{(n)}$ bázisfüggvény deriváltját a következő formula adja:*

$$N_i^{'(n)}(t) = n \left( \frac{N_i^{(n-1)}(t)}{t_{i+n} - t_i} - \frac{N_{i+1}^{(n-1)}(t)}{t_{i+n+1} - t_{i+1}} \right).$$

Az $N_i^{(n)}$ bázisfüggvények a knot-vektor pontjai között végtelenszer differenciálhatóak, hiszen ezeken a helyeken a bázisfüggvény polinom. A differenciálással gond a knot-vektor pontjaiban van, itt ugyanis függ a bázisfüggvény rendjétől a differenciálhatóság mértéke.

**5. Tétel.** *Az $N_i^{(n)}$ bázisfüggvény a $t = t_i$ pontban $n-1$-szer folytonosan differenciálható, ha $t_{i-1} \neq t_i$ és $t_{i+1} \neq t_i$. Ha a $t_i$ több knot-vektorbeli ponttal is megegyezik, akkor a folytonos differenciálhatóság annyival csökken, ahány pont a $t_i$-vel egyenlő.*

**10. Definíció.** *Legyen adott* $P_0$, $P_1$, ..., $P_{m-n-1}$ $m-n$ *db pont és a* $t_0$, $t_1$, ..., $t_m$ *knot-vektor. Ekkor az*

$$\forall t \in [t_n, t_{m-n}]\colon S(t) = \sum_{i=0}^{m-n-1} P_i N_i^{(n)}(t)$$

*függvényt* $n$*-ed fokú* B-Spline függvénynek *nevezzük. A* $P_i$ *pontokat* kontrollpontoknak *nevezzük.*

**Bázisfüggvények és B-Spline görbék tulajdonságai**

1. A B-Spline-hoz tartozó knot-vektorban lévő pontok száma ($m$), a kontrollpontok száma ($k$), és a B-Spline rendje ($n$) között a következő összefüggés áll fenn:

$$m = k + n + 1.$$

2. Az $N_i^{(n)}$ B-Spline bázis függvény a $(t_i, t_{i+n+1})$ intervallumon nullától különböző, mindenhol máshol nullával egyenlő. Teljes indukcióval nagyon könnyen bizonyítható az állítás.

3. $$\forall t \in \mathbb{R}\colon N_i^{(n)}(t) \geq 0.$$

   Ez a bázisfüggvények rekurzív definíciója alapján nyilvánvaló.

4. $$\forall t \in [t_n, t_{m-n}]\colon \sum_{i=0}^{m-n-1} N_i^{(n)}(t) = 1.$$

   **Bizonyítás.** A bázisfüggvények fokszámára vonatkozó teljes indukcióval igazolható az állítás. Tegyük fel, hogy $t \in [t_j, t_{j+1})$.

   (a) $n = 0$ eset. Ekkor

   $$\sum_{i=0}^{m-n-1} N_i^{(n)}(t) = N_j^{(n)}(t) = N_j^{(0)}(t) = 1.$$

   (b) Tegyük fel, hogy $k = n-1$-re fennáll az állítás. Ekkor

$$\sum_{i=0}^{m-n-1} N_i^{(n)}(t) = \sum_{i=j-n}^{j} N_i^{(n)}(t) = \sum_{i=j-n}^{j} \frac{t-t_i}{t_{i+n}-t_i} N_i^{(n-1)}(t) +$$

$$+\frac{t_{i+n+1}-t}{t_{i+n+1}-t_{i+1}} N_{i+1}^{(n-1)}(t) = \sum_{i=j-n+1}^{j} \frac{t-t_i}{t_{i+n}-t_i} N_i^{(n-1)}(t) +$$

$$+ \sum_{i=j-n}^{j-1} \frac{t_{i+n+1}-t}{t_{i+n+1}-t_{i+1}} N_{i+1}^{(n-1)}(t) = \sum_{i=j-n+1}^{j} \frac{t-t_i}{t_{i+n}-t_i} N_i^{(n-1)}(t) +$$

$$+\frac{t_{i+n}-t}{t_{i+n}-t_i} N_i^{(n-1)}(t) = \sum_{i=j-n+1}^{j} N_i^{(n-1)}(t) = 1.$$

Az állítás teljesül az indukciós feltétel miatt. ■

5. A B-Spline görbe pontjai a kontrollpontok konvex burkán belül helyezkednek el. A előző két tulajdonság alapján könnyen látható.

6. Ha a knot-vektor első és utolsó $n+1$ pontját összevonjuk, vagyis $t_0 = t_1 = \cdots = t_n$ és $t_{m-n} = t_{m-n+1} = \cdots = t_m$, akkor

$$N_0^{(n)}(t) = \begin{cases} \left(\frac{t_{n+1}-t}{t_{n+1}-t_0}\right)^n, & t_0 \le t < t_{n+1} \\ 0, & \text{különben} \end{cases}$$

és

$$N_{m-n-1}^{(n)}(t) = \begin{cases} \left(\frac{t}{t_m-t_{m-n-1}}\right)^n, & t_0 \le t < t_{n+1} \\ 0, & \text{különben} \end{cases}.$$

**Bizonyítás.** A bázisfüggvények rekurzív definícióját felhasználva:

$$N_0^{(n)}(t) = \frac{t-t_0}{t_n-t_0} N_0^{(n-1)}(t) + \frac{t_{n+1}-t}{t_{n+1}-t_1} N_1^{(n-1)}(t) =$$

$$= \frac{t_{n+1}-t}{t_{n+1}-t_1} N_1^{(n-1)}(t) = \cdots = \frac{t_{n+1}-t}{t_{n+1}-t_1} \cdots \frac{t_{n+1}-t}{t_{n+1}-t_n} N_n^{(0)}(t) =$$

$$= \left(\frac{t_{n+1}-t}{t_{n+1}-t_0}\right)^n N_n^{(0)}(t).$$

A $N_{m-n-1}^{(n)}$-re vonatkozó állítás ehhez hasonlóan igazolható. ■

7. Ha a knot-vektor két végén összevonunk $n+1$ db csomópontot, akkor a $P_0$ és $P_{m-n-1}$ kontrollpontok illeszkedni fognak a B-Spline görbére, vagyis a görbét ki tudjuk húzni a kontroll poligon szélére. Az előző tulajdonság alapján:

**Bizonyítás.** Nézzük meg a görbe első pontját, amihez a $t_n$ paraméterérték tartozik:

$$\sum_{i=0}^{m-n-1} P_i N_i^{(n)}(t_n) = P_0 N_0^{(n)}(t_n) = P_0 \left(\frac{t_{n+1}-t_n}{t_{n+1}-t_0}\right)^n = \\ = P_0 \left(\frac{t_{n+1}-t_n}{t_{n+1}-t_n}\right)^n = P_0.$$

A levezetés során kihasználtuk a 6. tulajdonságot. ■

Hasonlóképpen belátható, hogy

$$\sum_{i=0}^{m-n-1} P_i N_i^{(n)}(t_{n-m}) = P_{m-n-1}.$$

8. A kontrollpontok hatásköre lokális, ami azt jelenti, hogy egy kontrollpont megváltoztatása nem fogja megváltoztatni az egész görbét, hanem annak csak egy jelentősen kisebb szakaszát. Ennek az az oka, hogy a $P_i$ pont csak az $i$. bázisfüggvénnyel szorzódik meg a B-Spline görbében, és minden bázis függvény csak a paraméter tartománynak egy kisebb tartományán különbözik nullától. Pontosabban az $N_i^{(n)}(t)$ kifejezés csak $t \in (t_i, t_{i+n+1})$ esetén különbözik nullától, a többi paraméterértékhez tartozó görbeszakasz nem fog megváltozni, akárhogy is változtatjuk meg a $P_i$-t.

### 2.3.2. B-Spline felületek

A B-Spline felületeket a B-Spline görbék általánosításával kapjuk. Előnyös tulajdonságaik alapján a CAGD-ben előforduló felületeket ezzel a reprezentációval adják meg.

Kezdetben egy kontrollpont mátrixot adunk meg. Annyi B-Spline görbét hozunk létre, ahány sora van a kontrollpontokat tartalmazó mátrixnak, majd ezeknek a B-Spline görbéknek azonos paraméterértékekkel kiszámolt helyettesítési értékeit használjuk fel egy másik B-Spline görbe kontrollpontjaiként.

**11. Definíció.** *Adott a $P_{ij}$ ($i \in [0,n]$, $j \in [0,m]$) kontrollpont mátrix és $N$, $M$, a két irányhoz tartozó B-Spline bázisfüggvények fokszáma. Legyenek $t_0^u, t_1^u, \ldots, t_{n+N+1}^u$ az $u$ irányhoz, $t_0^v, t_1^v, \ldots, t_{m+M+1}^v$ pedig a $v$ irányhoz tartozó knot-vektorok. Ekkor az*

$$\forall u \in [t_N^u, t_{n+1}^u], \forall v \in [t_M^v, t_{m+1}^v]:$$

$$S(u,v) = \sum_{i=0}^{n} \left( \sum_{j=0}^{m} P_{ij} N_j^{(M)}(v) \right) N_i^{(N)}(u) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij}\ N_i^{(N)}(u) N_j^{(M)}(v)$$

*függvényt* B-Spline felületnek *nevezzük.*

Ha a továbbiakban nem mondunk mást, akkor a jelöléseket az előző definíciónak megfelelően kell érteni. A későbbiekben szükségünk lesz a B-Spline felületek különböző parciális deriváltjaira:

**6. Tétel.**

$$\frac{\partial S(u,v)}{\partial u} = \sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij}\ \frac{\partial N_i^{(N)}(u)}{\partial u} N_j^{(M)}(v)$$

$$\frac{\partial S(u,v)}{\partial v} = \sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij}\ N_i^{(N)}(u) \frac{\partial N_j^{(M)}(v)}{\partial v}$$

**Bizonyítás.** Közvetlenül adódik a B-Spline felület definíciójából. ■

# 3. fejezet

# Felületek közelítése diszkrét adatok alapján

A felületek illesztése során mindig valamilyen méréssel kapott diszkrét adatokból indulunk ki, és azokból szeretnénk olyan parametrikus felületet előállítani, amelyik jól illeszkedik az adatokra. A bemenő adatok lehetnek egy felület térbeli pontjai, vagy egy felület normális vektorai, esetleg offset felület meghatározása esetén a kiinduló felület pontjai. Az is lehet, hogy több felületdarabból rakjuk össze a közelítő felületet, ilyenkor a felületek egymáshoz kapcsolása is nehézségeket okoz.

Ebben a fejezetben azt vizsgáljuk, hogyan tudunk előállítani parametrikus felületeket valamilyen ponthalmazból azzal a feltétellel, hogy az előállított felület a lehető legjobban illeszkedjen az adott pontokra. A bemenő pontokat valamilyen tárgy letapogatásával kaptuk meg. A „legjobb illeszkedés" azonban nem egyértelmű, ezt többféleképpen is definiálhatjuk. Mi valami természetes megoldást szeretnénk keresni. Az sem biztos, hogy az a jó számunkra, ha a felület minden mérési ponton keresztül megy, hiszen mérési hibák is előfordulhatnak és azokat nem akarjuk visszakapni. Nem akarjuk, hogy a felület nagyon „egyenetlen" legyen. Valamilyen kompromisszumot kell kötni a mért pontok közelítése és a felület minősége között.

Tegyük fel, hogy a mérési adatok a $D_{ij}(i \in [0,k], j \in [0,l])$ mátrixban adottak. Ez a gyakorlatban nem biztos, hogy fennáll, sokszor csak egy részben struktúrált ponthalmaz áll rendelkezésre. Sok letapogató rendszer a mérés során egy párhuzamos vonalrendszer mentén vesz mintát. A mintavételezés száma egy vonal mentén nagyságrendileg nagyobb lehet a párhuzamos vonalak számánál, és különböző vonalak esetén ez a szám nem feltétlenül azonos. A párhuzamos vonalak mentén végighaladva, esetleg pontok elhagyásával ez a részben struktúrált ponthalmaz mátrixba rendezhető.

Keressük azt az $S$ felületet, amelyik a $D_{ij}$ pontokat legjobban közelíti. Ehhez azonban azt is meg kell mondanunk, hogy a felület mely pontja felel meg egy adott $D_{ij}$ mért térbeli pontnak. Az optimális megoldás az lenne, ha

a $D_{ij}$ pontnak a legközelebbi felületi pont felelne meg, ennek meghatározásához azonban ki kellene számolnunk egy pont és egy B-Spline felület közti távolságot, ami nem lineáris feladat. E helyett minden egyes mérési adathoz hozzárendelünk egy paraméterértéket ($D_{ij} \to (u_i, v_j)$), és az $S(u_i, v_j)$ felületi pontot feleltetjük meg a $D_{ij}$ mért pontnak. Ez a hozzárendelés később korrigálható a közelítés során, hiszen egyáltalán nem biztos, hogy kezdetben helyes paraméterezést választottunk. A feladat megtalálni azt az $S(u, v)$ felületet, amely minimalizálja

- az eltérések abszolut érték összegét:

$$\sum_{i=0}^{k} \sum_{j=0}^{l} \|D_{ij} - S(u_i, v_j)\| \to \text{minimum}$$

- az eltérések abszolut értékének maximumát:

$$\max_{i \in [0,k], j \in [0,l]} \{ \|D_{ij} - S(u_i, v_j)\| \} \to \text{minimum}$$

- az eltérések abszolut értékének négyzetösszegét:

$$\sum_{i=0}^{k} \sum_{j=0}^{l} \|D_{ij} - S(u_i, v_j)\|^2 \to \text{minimum}$$

A $\|.\|$ jelölés az euklideszi normát jelenti.

Ezek a feladatok nem veszik figyelembe a keletkező felület simaságát, csak a mért pontok közelítését. Mi a négyeztösszegek eltérésének minimalizálását választjuk, ez a túl nagy eltéréseket lecsökkenti és figyelembe vesz minden eltérést, nem csak a legnagyobbat, azonkívül lineáris feladatra vezet.

## 3.1. Legkisebb négyzetek módszere

A mért adatokból B-Spline reprezentációban állítjuk elő a felületet oly módon, hogy az a legkisebb négyzetek értelmében a legjobb közelítése legyen a bemenő adatoknak.

A továbbiakban feltételezzük, hogy adottak a mért adatok ($D_{ij}$, $i \in [0, k]$, $j \in [0, l]$), a B-Spline felület fokszáma a két irányban $(N, M)$, a kontrollpontok száma a két irányban $(n + 1, m + 1)$, a knot-vektorok a két irányban ($t^u_0$, $t^u_1$, ..., $t^u_{n+N+1}$, és $t^v_0$, $t^v_1$, ..., $t^v_{m+M+1}$). A kontrollpontokat jelölje $P_{pq}$ ($p \in [0, n]$, $q \in [0, m]$). A $D_{ij}$ pontnak az $(u_i, v_j)$ kezdeti paraméterek felelnek meg. Ekkor a keresett paraméterek a felület kontrollpontjai ($P$ mátrix elemei) lesznek. Ez lineárisan megoldható feladat lesz.

### 3.1.1. Megoldás négyzetes mátrixszal

Legyen

$$\mathcal{I} = \sum_{i=0}^{k} \sum_{j=0}^{l} \|D_{ij} - S(u_i, v_j)\|^2 .$$

Az $\mathcal{I}$ definíciójában az $S$ felület függ a $P_{pq}$ kontrollpontoktól. A cél az $\mathcal{I} \approx 0$ egyenlet megoldása, a paraméterek a $P_{pq}$ kontrollpontok. Mivel $\mathcal{I} \geq 0$, ezért elegendő megkeresni azokat a kontrollpontokat, amelyekre az $\mathcal{I}$ minimális. Ha megoldjuk a

$$\frac{\partial \mathcal{I}}{\partial P} = \underline{0}$$

egyenletet $P$-re, akkor kapunk egy optimális megoldást. Az egyenlet sorait kifejtve kapjuk:

$$\forall r \in [0,n] \colon \forall s \in [0,m] \colon \frac{\partial \mathcal{I}}{\partial P_{rs}} = \sum_{i=0}^{k} \sum_{j=0}^{l} -2[D_{ij} - S(u_i, v_j)] \frac{\partial S(u_i, v_j)}{\partial P_{rs}} =$$

$$= \sum_{i=0}^{k} \sum_{j=0}^{l} -2[D_{ij} - \sum_{p=0}^{n} \sum_{q=0}^{m} P_{pq}\, N_p^{(N)}(u_i) N_q^{(M)}(v_j)]\; N_r^{(N)}(u_i) N_s^{(M)}(v_j) =$$

$$= 2 \sum_{p=0}^{n} \sum_{q=0}^{m} P_{pq} \sum_{i=0}^{k} \sum_{j=0}^{l} N_p^{(N)}(u_i) N_q^{(M)}(v_j) N_r^{(N)}(u_i) N_s^{(M)}(v_j) -$$

$$-2 \sum_{i=0}^{k} \sum_{j=0}^{l} D_{ij}\; N_r^{(N)}(u_i) N_s^{(M)}(v_j) = 0.$$

Az egyenletet átrendezve a következő feladathoz jutunk:

$$\sum_{p=0}^{n} \sum_{q=0}^{m} P_{pq} \sum_{i=0}^{k} \sum_{j=0}^{l} N_p^{(N)}(u_i) N_q^{(M)}(v_j) N_r^{(N)}(u_i) N_s^{(M)}(v_j) =$$

$$= \sum_{i=0}^{k} \sum_{j=0}^{l} D_{ij}\; N_r^{(N)}(u_i) N_s^{(M)}(v_j).$$

Az egyenletek és a változók száma megegyezik a kontrollpontok számával $((n+1)(m+1))$, így egy négyzetes alapmátrixot kaptunk. Az egyenletet megoldhatjuk például *Householder transzformációval.*

### 3.1.2. Megoldás túlhatározott lineáris egyenletrendszerrel

A problémát más oldalról megközelítve különböző megoldási módszerhez jutunk. Az eredeti feladat:

$$\sum_{i=0}^{k}\sum_{j=0}^{l}\|D_{ij}-S(u_i,v_j)\|^2 =$$

$$\sum_{i=0}^{k}\sum_{j=0}^{l}\left\|D_{ij}-\sum_{p=0}^{n}\sum_{q=0}^{m}P_{pq}\,N_p^{(N)}(u_i)N_q^{(M)}(v_j)\right\|^2 =$$

$$\|\underline{b}-\mathcal{C}\underline{x}\|^2 \to \text{minimum},$$

ahol

$$\mathcal{C}=\begin{bmatrix} N_0^{(N)}(u_0)N_0^{(M)}(v_0), & N_0^{(N)}(u_0)N_1^{(M)}(v_0), & \dots & N_n^{(N)}(u_0)N_m^{(M)}(v_0)\\ N_0^{(N)}(u_0)N_0^{(M)}(v_1), & N_0^{(N)}(u_0)N_1^{(M)}(v_1), & \dots & N_n^{(N)}(u_0)N_m^{(M)}(v_1)\\ \vdots & \vdots & \ddots & \vdots\\ N_0^{(N)}(u_k)N_0^{(M)}(v_l), & N_0^{(N)}(u_k)N_1^{(M)}(v_l), & \dots & N_n^{(N)}(u_k)N_m^{(M)}(v_l)\end{bmatrix}, \tag{3.1}$$

$$\underline{b}=\begin{bmatrix} D_{00}\\ D_{01}\\ \vdots\\ D_{kl}\end{bmatrix}. \tag{3.2}$$

A feladat a $P_{pq}$ paraméterek (kontrollpontok) meghatározása a minimum feltételnek megfelelően. Az $\underline{x}$ vektor és a kontrollpontok között a kapcsolat a következő:

$$\underline{x}=\begin{bmatrix} P_{00}\\ P_{01}\\ \vdots\\ P_{nm}\end{bmatrix}. \tag{3.3}$$

A $\mathcal{C}$ mátrixnak annyi sora van, ahány mért adat $((k+1)(l+1))$, és annyi oszlopa, ahány kontrollpont $((n+1)(m+1))$. Az $\underline{x}$ vektornak $((n+1)(m+1))$ sora van.

Mivel $\|\underline{b}-\mathcal{C}\underline{x}\| \geq 0$, ezért a minimum feladat a következő egyenlettel ekvivalens:

$$\|\underline{b}-\mathcal{C}\underline{x}\| \approx 0,$$

ahol $\mathcal{C} \in \mathbb{R}^{n\times m}$, $\underline{b} \in \mathbb{R}^n$, $\underline{x} \in \mathbb{R}^m$. Ha a $\mathcal{C}$ együttható mátrix négyzetes, vagyis ugyanannyi a kontrollpontok és mért pontok száma, akkor interpolációs problémával állunk szemben. Ha a $|\mathcal{C}| \neq 0$, akkor a lineáris egyenletrendszernek létezik egyértelmű megoldása és olyan B-Spline felületet kapunk, amely átmegy minden mért ponton.

Ha több mért pont van, mint kontrollpont, és általában ez a helyzet, akkor approximációs feladatot kapunk. Ilyenkor a legtöbb esetben nem létezik olyan B-Spline felület a megadott paraméterekkel, amelyik minden ponton átmegy. Ekkor keressük a $\underline{b}$ vektort legjobban közelítő megoldást. Ezt a $\mathcal{C}$ mátrix általános inverze adja meg. Az általános inverzet a *Householder* transzformációval számíthatjuk ki.

### 3.1.3. Householder transzformáció

A *Householder* transzformáció előállítja bármely $A \in \mathbb{R}^{n\times n}$ reguláris mátrix esetén az $A = QR$ felbontást, ahol $Q \in \mathbb{R}^{n\times n}$ szimmetrikus ortogonális mátrix ($Q = Q^T,\ QQ^T = I$), az $R$ pedig felső háromszög mátrix.

Először bevezetjük a *Householder transzformáció* fogalmát. Egy transzformációra van szükségünk, ami stabil és ki tudjuk vele nullázni egy mátrix első oszlopában lévő elemeket, az első sortól eltekintve.

**12. Definíció.** Householder transzformációnak *nevezzük az*

$$A \to Q(u)A$$

*leképezést, ahol* $Q(u) = I - \frac{2uu^T}{\|u\|^2} \in \mathbb{R}^{n\times n}$ *és* $u \in \mathbb{R}^n, u \neq 0$ *tetszőleges vektor.*

A $Q(u)$ mátrix szimmetrikus, mivel

$$Q(u)_{ij} = \delta_{ij} - \frac{2u_iu_j}{\|u\|^2}.$$

A $Q(u)$ mátrix ortogonális, mivel

$$Q(u)Q(u)^T = I - \frac{4uu^T}{\|u\|^2} + \frac{4uu^Tuu^T}{\|u\|^4} = I.$$

A leképezést úgy is fel lehet fogni, mint egy tükrözést az $u$ vektorra merőleges síkra, hiszen egy $x$ vektor képe: $Q(u)x = x - 2\frac{u}{\|u\|}\left(\frac{u^T}{\|u\|}x\right)$.

Az $u$ vektort úgy akarjuk megválasztani, hogy egy rögzített $a$ vektor esetén (ami majd az $A$ mátrix első oszlopa lesz) $Q(u)a = \rho e_1$ teljesüljön ($e_1$ az első elemi bázisvektor), ahol $\rho \in \mathbb{R}$ valamilyen rögzített szám. Legyen $u = a + \tau e_1$ alakú, ahol a $\tau \in \mathbb{R}$ számot később definiáljuk. Legyen $c = \frac{2}{\|u\|^2}$, a rövidebb jelölés miatt. Ezek alapján

$$Q(u)a = a - cu(u^Ta) = a - c(a+\tau e_1)(u^Ta) = a(1 - c(u^Ta)) - \tau ce_1(u^Ta).$$

Ha $1 = c(u^Ta)$ teljesülne, akkor fennállna a $Q(u)a = -\tau e_1$ egyenlőség, vagyis amit akartunk. Tekintsük a következő egyenleteket:

$$\left.\begin{array}{c} 1 = c(u^Ta) = c(\|a\|^2 + \tau e_1^Ta) \\ \|u\|^2 = (a+\tau e_1)^2 = \|a\|^2 + 2\tau e_1^Ta + \tau^2\|e_1\|^2 \end{array}\right\} \Rightarrow \|a\|^2 = \tau^2\|e_1\|^2.$$

Ehhez az szükséges, hogy $\tau = \pm\frac{\|a\|}{\|e_1\|}$ igaz legyen. A $\tau$-t bármelyiknek választhatjuk, de a stabilitás miatt célszerű az $\|u\|$-t nagyobbnak választani, ezért legyen:

$$\tau = \begin{cases} \|a\|, & e_1^T a = a_1 \geq 0 \\ -\|a\|, & e_1^T a = a_1 < 0 \end{cases},$$

mivel $\|e_1\| = 1$. Ezzel a módszerrel tetszőleges $A$ mátrix esetén meg tudunk határozni egy $u = a + \tau e_1$ vektort, amire teljesül:

$$Q(u)A = \begin{bmatrix} -\tau & * & \dots & * \\ 0 & & & \\ \vdots & & & \\ 0 & & & \end{bmatrix}.$$

Hívjuk az első lépésben kapott mátrixot $Q^{(1)}$-nek. Végezzük el a kinullázást arra a mátrixra, amit úgy kapunk, hogy a $Q^{(1)}A$ mátrix első sorát és első oszlopát elhagyjuk. Ekkor kapunk egy $Q^{(2)}$ mátrixot. Ezt az eljárást ismételjük $n-1$-szer, majd képezzük a kapott szimmetrikus ortogonális mátrixokból a következő módosított mátrixokat:

$$\tilde{Q}^{(1)} = Q^{(1)} \in \mathbb{R}^{n\times n},\ \tilde{Q}^{(2)} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & Q^{(2)} & \\ 0 & & & \end{bmatrix} \in \mathbb{R}^{n\times n}, \dots,$$

$$\tilde{Q}^{(n-1)} = \begin{bmatrix} 1 & 0 & \dots \\ 0 & \ddots & \\ \vdots & & Q^{(n-1)} \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

Ezzel kaptunk egy $\tilde{Q} = \tilde{Q}^{(n-1)} \dots \tilde{Q}^{(1)}$ ortogonális mátrixot, amire $QA$ felső háromszög mátrix. Legyen $R = QA$, ekkor $Q^T R = A$ az A mátrix egy $QR$ felbontása. Ezt az eljárást $A \in \mathbb{R}^{m\times n}$ mátrixra is lehet alkalmazni, ahol $m > n$. Ekkor $n$ db $\tilde{Q}_i$ mátrixot kapunk, az $R$ pedig a következő alakú lesz:

$$R = \begin{bmatrix} r_{11} & r_{12} & \dots \\ \vdots & \ddots & * \\ 0 & \dots & r_{nn} \\ & 0 & \\ & & \end{bmatrix}.$$

Legyen most $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $x \in \mathbb{R}^n$. Keressük azt az $\tilde{x}$ vektort, melyre:

$$\tilde{x} \in \mathbb{R}^n,\ \|A\tilde{x} - b\| = \min_{x\in\mathbb{R}^n}\{\ \|Ax - b\|\ \}.$$

Vegyük hozzá az $A$ vektorhoz utolsó oszlopként a $b$ vektort, az eredményt jelölje $\tilde{A}$. Végezzük el az $\tilde{A}$ mátrix $QR$ felbontását ($\tilde{A} = QR$). Ez tetszőleges mátrix esetén elvégezhető. A kapott $R$ felső háromszög mátrix legyen

$$R = \begin{bmatrix} R' & r \\ 0 & c \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{m\times(n+1)}, r \in \mathbb{R}^n, c \in \mathbb{R}$$

alakú. Mivel euklideszi normát használunk, ezért $\forall y \in \mathbb{R}^m$: $\|Qy\|^2 = (Qy)^T Qy = y^T Q^T Qy = y^T y = \|y\|^2$. Ezek alapján:

$$\|Ax - b\|^2 = \left\|Q^T(Ax - b)\right\|^2 = \left\|Q^T \tilde{A} \begin{bmatrix} x \\ -1 \end{bmatrix}\right\|^2 = \left\|R'x - r\right\|^2 + c^2 \geq c^2.$$

Innen következik, hogy az „$\|Ax - b\| \to$ minimum" probléma megoldását visszavezettük az „$\|R'x - r\| \to$ minimum" probléma megoldására, ami már $R$ felső háromszög volta miatt triviális. Az sem okoz problémát, ha az $A$ mátrix nem volt teljes rangú, ilyenkor az $R$ mátrix főátlójában nullák is elő fognak fordulni.

A *Householder transzformáció* használható mért adatokat approximáló B-Spline felületek előállítására, a kapcsolatot az $A$ mátrix és a bázisfüggvények, a $b$ vektor és a mérési adatok, az $x$ vektor és a kontrollpontok között a 3.1, 3.2 és 3.3 egyenletek írják le.

## 3.2. Funkcionálok

Gyakorlati feladatok megoldása során sokszor felmerül az a természetes igény, hogy a mért pontokra illesztett felület ne csak a közelítés feltételének feleljen meg, hanem a lehető legsimább is legyen. Ezt részben az esztétikai elvárásaink, részben a felületek gyakorlati szerepe követeli meg. Gondoljunk például egy autó karosszériaelemére, amelynek fényvisszaverődési tulajdonságait nagymértékben befolyásolja a felület simasága. A repülőgépek szárnyának kialakításakor pedig az aerodinamikai jellemzők optimalizálása érdekében fontos, hogy a felület sima legyen. Ebben a fejezetben olyan módszerekről lesz szó, amelyekkel növelhetjük a felületek simaságát úgy, hogy közben a mért adatoktól sem távolodunk el túlságosan.

A feladat tehát olyan B-Spline felületek előállítása, amelyekre minimális a mért pontoktól vett távolságnégyzetek összege, valamint egy a simaságot leíró funkcionál is. Ez ellentmondást hordoz magában, mert minél jobban megközelítjük a mért pontokat, annál többet veszíthetünk a simaságból. Fordítva is igaz. Minél simább az előállított felület, annál jobban eltávolodhatunk a mért pontoktól. Az ellentmondás feloldása érdekében valamilyen súlyokat kell bevezetnünk, ami azt határozza meg, hogy mennyire fontos számunkra a felület simasága és a pontoktól mért távolság. Ezután

nem külön-külön minimalizáljuk a felületek ezen két alapvető jellemzőjét leíró funkcionált, hanem a súllyal korrigált összegüket egyszerre.

Legyen továbbra is

$$\mathcal{I}(P) = \sum_{i=0}^{k} \sum_{j=0}^{l} \|D_{ij} - S(u_i, v_j)\|^2 ,$$

ahol $P$ a kontrollpontokat tartalmazó mátrix, ami az előállított $S$ felületet befolyásolja. Jelölje $\mathcal{Q}(P)$ a simaságot leíró funkcionált. Ezt többféleképpen is megadhatjuk. Például minimalizálhatjuk a görbületet, a görbületi energiát, a felületi feszültséget. Ezután a minimalizálandó függvény a következő lesz:

$$\mathcal{F}(P) = (1 - \mu)\mathcal{I}(P) + \mu\mathcal{Q}(P),$$

ahol $\mu \in [0, 1]$ a simaság fontosságát meghatározó konstans. A minimum meghatározásához deriválnunk kell az $\mathcal{F}(P)$ függvényt minden kontrollpont szerint. A megoldandó egyenlet:

$$\frac{\partial \mathcal{F}(P)}{\partial P} = (1 - \mu)\frac{\partial \mathcal{I}(P)}{\partial P} + \mu\frac{\partial \mathcal{Q}(P)}{\partial P} = 0.$$

A $\frac{\partial \mathcal{I}(P)}{\partial P}$ értékét kontrollpontonkénti deriválással, a 3.1.2. szakaszban leírtak alapján a következőképpen számolhatjuk ki:

$$\forall r \in [0, n] \colon \forall s \in [0, m] \colon$$

$$\frac{\partial \mathcal{I}(P)}{\partial P_{rs}} = 2 \sum_{p=0}^{n} \sum_{q=0}^{m} P_{pq} \sum_{i=0}^{k} \sum_{j=0}^{l} N_p^{(N)}(u_i) N_q^{(M)}(v_j) N_r^{(N)}(u_i) N_s^{(M)}(v_j) -$$

$$-2 \sum_{i=0}^{k} \sum_{j=0}^{l} D_{ij}\ N_r^{(N)}(u_i) N_s^{(M)}(v_j).$$

A következö szakaszokban megvizsgálunk néhány lineárisan minimalizálható funkcionált, amelyek alkalmasak a felület minőségének jellemzésére.

### 3.2.1. Felszín minimalizálása

Ebben a szakaszban úgy próbáljuk simítani a felületet, hogy a felület pontjaihoz tartozó parciális derivált vektorok hosszát minimalizáljuk. Mivel a felszín is az elsőrendű parciális deriváltaktól függ, ez azt eredményezi, hogy a felszín is kisebb lesz, vagyis a felület simul.

Legyen

$$\mathcal{Q}(P) = \int_{t_N^u}^{t_{n+1}^u} \int_{t_M^v}^{t_{m+1}^v} (\nabla S(u, v))^2 \, du dv =$$

$$= \int_{t_N^u}^{t_{n+1}^u} \int_{t_M^v}^{t_{m+1}^v} (S_u(u, v)^2 + S_v(u, v)^2) \, du dv.$$

A $\mathcal{Q}(P)$ kontrollpontok szerinti deriváltjának kiszámításához szükségünk lesz az $S$ felület parciális deriváltjaira és azok kontrollpontok szerinti deriváltjaira is:

$$\begin{aligned}
S_u(u,v) &= \sum_{p=0}^{n}\sum_{q=0}^{m} P_{pq}\frac{\partial N_p^{(N)}(u)}{\partial u}N_q^{(M)}(v)\\
S_v(u,v) &= \sum_{p=0}^{n}\sum_{q=0}^{m} P_{pq}N_p^{(N)}(u)\frac{\partial N_q^{(M)}(v)}{\partial v}\\
\frac{\partial S_u(u,v)}{\partial P_{rs}} &= \frac{\partial N_r^{(N)}(u)}{\partial u}N_s^{(M)}(v)\\
\frac{\partial S_v(u,v)}{\partial P_{rs}} &= N_r^{(N)}(u)\frac{\partial N_s^{(M)}(v)}{\partial v}.
\end{aligned}$$

Ezek alapján

$$\begin{aligned}
\frac{\partial \mathcal{Q}(P)}{\partial P_{rs}} &= 2\int\int S_u(u,v)\frac{\partial S_u(u,v)}{\partial P_{rs}} + S_v(u,v)\frac{\partial S_v(u,v)}{\partial P_{rs}}\;dudv =\\
&= 2\int\int \sum_{p=0}^{n}\sum_{q=0}^{m}\Bigg(P_{pq}\frac{\partial N_p^{(N)}(u)}{\partial u}N_q^{(M)}(v)\frac{\partial N_r^{(N)}(u)}{\partial u}N_s^{(M)}(v) +\\
&\quad + P_{pq}N_p^{(N)}(u)\frac{\partial N_q^{(M)}(v)}{\partial v}N_r^{(N)}(u)\frac{\partial N_s^{(M)}(v)}{\partial v}\Bigg)\;dudv =\\
&= 2\sum_{p=0}^{n}\sum_{q=0}^{m} P_{pq}\Bigg(\int \frac{\partial N_p^{(N)}(u)}{\partial u}\frac{\partial N_r^{(N)}(u)}{\partial u}\;du\int N_q^{(M)}(v)N_s^{(M)}(v)\;dv +\\
&\quad + \int N_p^{(N)}(u)N_r^{(N)}(u)\;du\int \frac{\partial N_q^{(M)}(v)}{\partial v}\frac{\partial N_s^{(M)}(v)}{\partial v}\;dv\Bigg).
\end{aligned}$$

Ezzel megkaptuk a simításhoz szükséges funkcionál kontrollpontonkénti deriváltját. Most már fel tudjuk írni az eredeti feladat megoldását adó egyen-

leteket:

$$\forall r \in [0,n]\colon \forall s \in [0,m]\colon \frac{\partial \mathcal{F}(P)}{\partial P_{rs}} =$$

$$= 2\sum_{p=0}^{n}\sum_{q=0}^{m} P_{pq}\Bigg((1-\mu)\sum_{i=0}^{k}\sum_{j=0}^{l} N_p^{(N)}(u_i)N_q^{(M)}(v_j)N_r^{(N)}(u_i)N_s^{(M)}(v_j) +$$

$$+\mu\Bigg(\int \frac{\partial N_p^{(N)}(u)}{\partial u}\frac{\partial N_r^{(N)}(u)}{\partial u}\,du \int N_q^{(M)}(v)N_s^{(M)}(v)\,dv +$$

$$+\int N_p^{(N)}(u)N_r^{(N)}(u)\,du \int \frac{\partial N_q^{(M)}(v)}{\partial v}\frac{\partial N_s^{(M)}(v)}{\partial v}\,dv\Bigg)\Bigg) -$$

$$-(1-\mu)2\sum_{i=0}^{k}\sum_{j=0}^{l} D_{ij}\; N_r^{(N)}(u_i)N_s^{(M)}(v_j) = 0.$$

Ez $(n+1)(m+1)$ egyenletet jelent, amelyekben az ismeretlenek (a $P$ mátrix elemei) száma szintén $(n+1)(m+1)$. Az egyenletrendszer megoldható *Householder transzformációval.*

### 3.2.2. Görbületi energia minimalizálása

A görbületi energia minimalizálásával is simább felületet kaphatunk. Legyen a $\mathcal{Q}(P)$ funkcionál a következő:

$$\mathcal{Q}(P) = \int_{t_N^u}^{t_{n+1}^u}\int_{t_M^v}^{t_{m+1}^v} (S_{uu}(u,v)+S_{vv}(u,v))^2 -$$
$$-2(1-\nu)(S_{uu}(u,v)S_{vv}(u,v) - S_{uv}(u,v)^2)\;dudv,$$

ahol a felület $u$ illetve $v$ szerinti deriváltjait így írhatjuk fel:

$$\begin{aligned}
S_{uu}(u,v) &= \sum_{p=0}^{n}\sum_{q=0}^{m} P_{pq}\,\frac{\partial^2 N_p(u)}{\partial u^2}N_q(v)\\
S_{vv}(u,v) &= \sum_{p=0}^{n}\sum_{q=0}^{m} P_{pq}\,N_p(u)\frac{\partial^2 N_q(v)}{\partial v^2}\\
S_{uv}(u,v) &= \sum_{p=0}^{n}\sum_{q=0}^{m} P_{pq}\,\frac{\partial N_p(u)}{\partial u}\frac{\partial N_q(v)}{\partial v}.
\end{aligned}$$

Mi csak a $\nu = 1$ esettel foglalkozunk. Most írjuk fel a funkcionál kontrollpontonkénti deriváltját, amire a minimalizálásnál lesz szükség:

$$\frac{\partial \mathcal{Q}(P)}{\partial P_{rs}} =$$

$$= 2 \int\int (S_{uu}(u,v) + S_{vv}(u,v)) \left( \frac{\partial S_{uu}(u,v)}{\partial P_{rs}} + \frac{\partial S_{vv}(u,v)}{\partial P_{rs}} \right) \, dudv =$$

$$= 2 \int\int \left( \sum_{p=0}^{n} \sum_{q=0}^{m} P_{pq} \frac{\partial^2 N_p(u)}{\partial u^2} N_q(v) + \sum_{p=0}^{n} \sum_{q=0}^{m} P_{pq} N_p(u) \frac{\partial^2 N_q(v)}{\partial v^2} \right)$$
$$\left( \frac{\partial^2 N_r(u)}{\partial u^2} N_s(v) + N_r(u) \frac{\partial^2 N_s(v)}{\partial v^2} \right) \, dudv =$$

$$= 2 \sum_{p=0}^{n} \sum_{q=0}^{m} P_{pq} \Bigg( \int \frac{\partial^2 N_p(u)}{\partial u^2} \frac{\partial^2 N_r(u)}{\partial u^2} \, du \int N_q(v) N_s(v) \, dv +$$
$$+ \int \frac{\partial^2 N_p(u)}{\partial u^2} N_r(u) \, du \int N_q(v) \frac{\partial^2 N_s(v)}{\partial v^2} \, dv +$$
$$+ \int N_p(u) \frac{\partial^2 N_r(u)}{\partial u^2} \, du \int \frac{\partial^2 N_q(v)}{\partial v^2} N_s(v) \, dv +$$
$$+ \int N_p(u) N_r(u) \, du \int \frac{\partial^2 N_q(v)}{\partial v^2} \frac{\partial^2 N_s(v)}{\partial v^2} \, dv \Bigg).$$

Ezután felírhatjuk az eredeti feladatot megoldó egyenleteket:

$$\forall r \in [0,n] \colon \forall s \in [0,m] \colon \frac{\partial \mathcal{F}(P)}{\partial P_{rs}} =$$

$$= 2 \sum_{p=0}^{n} \sum_{q=0}^{m} P_{pq} \Bigg( (1-\mu) \sum_{i=0}^{k} \sum_{j=0}^{l} N_p^{(N)}(u_i) N_q^{(M)}(v_j) N_r^{(N)}(u_i) N_s^{(M)}(v_j) +$$
$$+ \mu \Bigg( \int \frac{\partial^2 N_p(u)}{\partial u^2} \frac{\partial^2 N_r(u)}{\partial u^2} \, du \int N_q(v) N_s(v) \, dv +$$
$$+ \int \frac{\partial^2 N_p(u)}{\partial u^2} N_r(u) \, du \int N_q(v) \frac{\partial^2 N_s(v)}{\partial v^2} \, dv +$$
$$+ \int N_p(u) \frac{\partial^2 N_r(u)}{\partial u^2} \, du \int \frac{\partial^2 N_q(v)}{\partial v^2} N_s(v) \, dv +$$
$$+ \int N_p(u) N_r(u) \, du \int \frac{\partial^2 N_q(v)}{\partial v^2} \frac{\partial^2 N_s(v)}{\partial v^2} \, dv \Bigg) \Bigg) -$$
$$- (1-\mu) 2 \sum_{i=0}^{k} \sum_{j=0}^{l} D_{ij} \; N_r^{(N)}(u_i) N_s^{(M)}(v_j) = 0.$$

### 3.2.3. Kontrollpoligon minimalizálása

Most abból indulunk ki, hogy a felületen ott keletkeznek kiemelkedések vagy süllyedések, ahol egy olyan kontrollpont befolyásolja a felületet, amelyik a szomszédaitól távolabb helyezkedik el. Ezért arra fogunk törekedni, hogy az ilyen kiugró kontrollpontokat közelítsük a szomszédaihoz, vagyis a köztük lévő távolságnégyzeteket minimalizáljuk.

Válasszuk a következő funkcionált a simításhoz:

$$\mathcal{Q}(P) = \sum_{p=0}^{n}\sum_{q=0}^{m} \chi(p-1,q)(P_{p-1q} - P_{pq})^2 + \chi(p+1,q)(P_{p+1q} - P_{pq})^2 + \\ +\chi(p,q-1)(P_{pq-1} - P_{pq})^2 + \chi(p,q+1)(P_{pq+1} - P_{pq})^2,$$

ahol

$$\chi(i,j) = \begin{cases} 1, & \text{ha } i \in [0,n] \text{ és } j \in [0,m] \\ 0, & \text{különben} \end{cases} .$$

A $\mathcal{Q}(P)$ funkcionál $P_{rs}$ kontrollpont szerinti deriváltja:

$$\frac{\partial \mathcal{Q}(P)}{\partial P_{rs}} = \chi(r-1,s)4(P_{rs} - P_{r-1s}) + \chi(r+1,s)4(P_{rs} - P_{r+1s}) + \\ +\chi(r,s-1)4(P_{rs} - P_{rs-1}) + \chi(r,s+1)4(P_{rs} - P_{rs+1}).$$

A megoldandó egyenletek tehát:

$$\forall r \in [0,n] \colon \forall s \in [0,m] \colon \frac{\partial \mathcal{F}(P)}{\partial P_{rs}} = \\ = (1-\mu)2\sum_{p=0}^{n}\sum_{q=0}^{m} P_{pq} \sum_{i=0}^{k}\sum_{j=0}^{l} N_p^{(N)}(u_i) N_q^{(M)}(v_j) N_r^{(N)}(u_i) N_s^{(M)}(v_j) + \\ +\mu\big(\chi(r-1,s)4(P_{rs} - P_{r-1s}) + \chi(r+1,s)4(P_{rs} - P_{r+1s}) + \\ +\chi(r,s-1)4(P_{rs} - P_{rs-1}) + \chi(r,s+1)4(P_{rs} - P_{rs+1})\big) - \\ -(1-\mu)2\sum_{i=0}^{k}\sum_{j=0}^{l} D_{ij}\ N_r^{(N)}(u_i) N_s^{(M)}(v_j) = 0.$$

## 3.3. Paraméter korrekció

Az approximáció alapelve, hogy olyan felületet találjunk a mért pontjainkhoz, amely a lehető legjobban közelíti azokat. A közelítésen a legkisebb négyzetes eltérést értjük. Az első lépésnél azonban még nincs felületünk amihez viszonyíthatnánk, azt sem tudjuk eldönteni, hogy a felület mely paraméterértékeinél vett pontja van legközelebb egy adott mért ponthoz. Az approximációhoz viszont mindenképpen felületi pontokat és mért pontokat

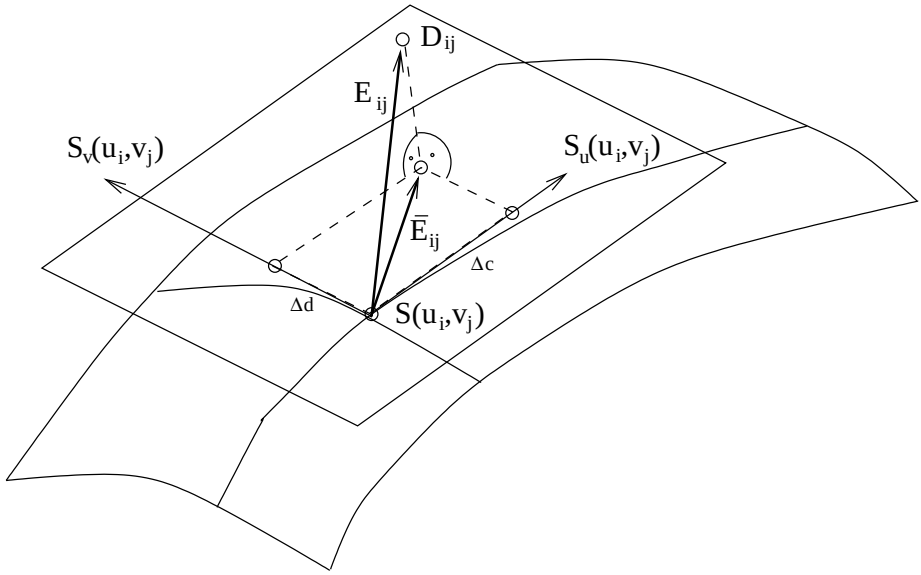
kell egymáshoz rendelnünk. Ezért a közelítés első lépésekor valamilyen kezdeti paraméterezést határozunk meg.

Mi abból indulunk ki, hogy a mért pontjainkat egy mátrixban tároljuk, és a paraméter eloszlás egyenletes. A kezdeti paraméterezés meghatározására más módszereket is használhatunk, például Coons felületre vetíthetjük a mért pontokat és az így kapott felületi ponthoz tartozó paraméterértékeket rendeljük a mért pontokhoz.

Approximáció végzésekor mindenképpen ajánlott a kezdetben megadott paraméterértékeket korrigálni. Nem szabad ugyanis megfeledkeznünk arról, hogy a minimalizáláskor használt hiba vektorok ($E_{ij} = D_{ij} - S(u_i, v_j)$) nem merőlegesek a közelítő felületre ($S(u, v)$).

### 3.3.1. Korrekció vetítéssel

A korrigáláshoz először vetítsük le a $D_{ij}$ pontot az $S(u, v)$ közelítő felület $S(u_i, v_j)$ pontjához tartozó érintősíkra. Legyen ez a $D'_{ij}$.




3.1. ábra. Paraméter korrekció vázlatos rajza.

Az érintősík az $S_u(u_i, v_j)$ és $S_v(u_i, v_j)$ parciális deriváltak segítségével határozható meg. Legyen az $\overline{E}_{ij}$ az $S(u_i, v_j)$-ből a $D'_{ij}$ pontba mutató vektor. Ez a vektor nem más, mint az eredeti $E_{ij}$ hibavetornak az érintősíkon vett vetülete. Most már tudjuk, hogy a felületen milyen irányban és milyen távolságra van az a pont, ami a mért pontunkhoz közelebb esik, mint a ponthoz tartozó paraméterértékek által meghatározott felületi pont. Azt szeretnénk megtudni, hogy ehhez a közelebbi ponthoz milyen paraméterértékek tartoznak.

Legyenek $u_i^*$ és $v_i^*$ a módosított paraméterértékek. $L(u_i)$ valamint $L(v_j)$ az $S(u_i, v_j)$ ponthoz tartozó $u$ illetvet $v$ irányú paramétervonalak hossza. $\omega_u$ és $\omega_v$ az $u$ valamint a $v$ paraméter intervallumok hossza. $\Delta c$ és $\Delta d$ az $E_{ij}$ vektor $S_u(u_i, v_j)$ illetve $S_v(u_i, v_j)$ parciális derivált vektorokra vett vetületének a hosszát jelentik. Ekkor a következő aránypárokat írhatjuk fel:

$$\frac{\Delta c}{L(u_i)} = \frac{u_i^* - u_i}{\omega_u}, \qquad \frac{\Delta d}{L(v_j)} = \frac{v_j^* - v_j}{\omega_v}$$

$$\Downarrow$$

$$u_i^* = u_i + \frac{\Delta c}{L(u_i)}\omega_u, \qquad v_j^* = v_j + \frac{\Delta d}{L(v_j)}\omega_v,$$

ahol

$$\Delta c = \left\langle E_{ij}, \frac{S_u(u_i, v_j)}{\|S_u(u_i, v_j)\|} \right\rangle,$$

$$\Delta d = \left\langle E_{ij}, \frac{S_v(u_i, v_j)}{\|S_v(u_i, v_j)\|} \right\rangle.$$

### 3.3.2. Korrekció első rendű Taylor-polinommal

Vegyük a következő függvényt: $F(u, v) = D_{ij} - S(u, v)$. Ez a közelítő felület pontjait a mért ponttal összekötő vektorokat adja meg. Keressük azokat az $u$, $v$ paramétereket, amelyek mellett az $F(u, v)$ vektor a legrövidebb. Tehát a feladat:

$$\left\| F(u, v)^2 \right\| \to minimum.$$

Az $F$ függvényt közelíteni tudjuk, ha a felületet az $(u_i, v_j)$ pont körül Taylor-sorba fejtjük. Így a következő képletet kapjuk:

$$F(u, v) \approx \tilde{F}(u, v) = D_{ij} - S(u_i, v_j) -$$
$$- S_u(u_i, v_j)(u - u_i) - S_v(u_i, v_j)(v - v_j).$$

A hiba vektor akkor lesz a legrövidebb, ha:

$$\frac{\partial \tilde{F}(u, v)^2}{\partial u} = -2\tilde{F}(u, v)S_u(u_i, v_j) = 0$$

$$\frac{\partial \tilde{F}(u, v)^2}{\partial v} = -2\tilde{F}(u, v)S_v(u_i, v_j) = 0.$$

Kifejtve:

$$\tilde{F}(u,v)S_u(u_i,v_j) =$$
$$= (D_{ij} - S(u_i,v_j) - S_u(u_i,v_j)\Delta u - S_v(u_i,v_j)\Delta v)\, S_u(u_i,v_j) =$$
$$= E_{ij}S_u(u_i,v_j) - S_u(u_i,v_j)^2\Delta u - S_v(u_i,v_j)S_u(u_i,v_j)\Delta v,$$

$$\tilde{F}(u,v)S_v(u_i,v_j) =$$
$$= (D_{ij} - S(u_i,v_j) - S_u(u_i,v_j)\Delta u - S_v(u_i,v_j)\Delta v)\, S_v(u_i,v_j) =$$
$$= E_{ij}S_v(u_i,v_j) - S_u(u_i,v_j)S_v(u_i,v_j)\Delta u - S_v(u_i,v_j)^2\Delta v,$$

ahol

$$\begin{aligned} \Delta u &= u - u_i \\ \Delta v &= v - v_j \\ E_{ij} &= D_{ij} - S(u_i,v_j). \end{aligned}$$

Azokat a $\Delta u$ és $\Delta v$ értékeket kell megkeresni, amelyek kielégítik a fenti egyenletrendszert. Legyen

$$\begin{aligned} a &= S_u(u_i,v_j)^2 \\ b &= S_v(u_i,v_j)S_u(u_i,v_j) \\ c &= S_u(u_i,v_j)S_v(u_i,v_j) \\ d &= S_v(u_i,v_j)^2 \\ e &= E_{ij}S_u(u_i,v_j) \\ f &= E_{ij}S_v(u_i,v_j) \\ x &= \Delta u \\ y &= \Delta v. \end{aligned}$$

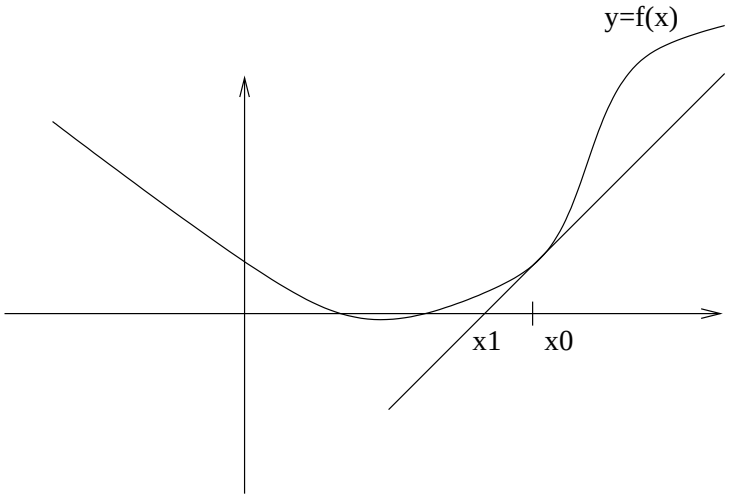
Ebben a jelölésrendszerben felírva az előzőekben kapott egyenleteket és megoldva $x$, $y$-ra, az alábbi eredményre jutunk:

$$\begin{aligned} ax + by = e, &\qquad cx + dy = f \\ &\Downarrow \\ x = \frac{de - fb}{ad - bc}, &\qquad y = \frac{af - ec}{ad - bc}. \end{aligned}$$

Visszahelyettesítve az eredeti egyenlet megoldását kapjuk $\Delta u$, $\Delta v$-re:

$$\Delta u = \frac{S_v(u_i,v_j)^2(E_{ij}S_u(u_i,v_j)) - (E_{ij}S_v(u_i,v_j))(S_v(u_i,v_j)S_u(u_i,v_j))}{S_u(u_i,v_j)^2S_v(u_i,v_j)^2 - (S_v(u_i,v_j)S_u(u_i,v_j))^2}$$

$$\Delta v = \frac{S_u(u_i,v_j)^2(E_{ij}S_v(u_i,v_j)) - (E_{ij}S_u(u_i,v_j))(S_u(u_i,v_j)S_v(u_i,v_j))}{S_u(u_i,v_j)^2S_v(u_i,v_j)^2 - (S_v(u_i,v_j)S_u(u_i,v_j))^2}.$$




3.2. ábra. A Newton-iteráció egy lépése.

Ezek alapján az új paraméterértékek már nagyon egyszerűen kiszámíthatóak:

$$\begin{aligned} u_i^* &= u_i + \Delta u \\ v_j^* &= v_j + \Delta v. \end{aligned}$$

### 3.3.3. Korrekció Newton módszerrel

Az előző módszernél a közelítő felület Taylor-sorba fejtésével kerestük meg azokat a paraméterértékeket, amelyek a közelítő felület azon pontját határozták meg, amelyik a legközelebb helyezkedik el a mért ponthoz. Most egy másik megközelítést vizsgálunk meg. Legyen a minimalizálandó hibavektor az $F(u,v) = D_{ij} - S(u,v)$. Ez a vektor ott lesz a legrövidebb, ahol az $F^2$ függvény $u$ illetve $v$ szerinti deriváltja 0-val egyenlő:

$$\frac{\partial F^2(u,v)}{\partial u} = 0, \qquad \frac{\partial F^2(u,v)}{\partial v} = 0.$$

A zérushely megtalálására használjuk a Newton módszert. A módszer egy lépése során a következő egyenletből kiindulva határozza meg egy adott $x_0$ értéknél a zérushelyhez közelebb eső $x_1$ pontot:

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1}$$

$$\Downarrow$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

A mi esetünkben $f$ függvénynek az $F^2$ függvény $u$ illetve $v$ szerinti deriváltját választjuk, hiszen ezeknek a zérushelyeire vagyunk kíváncsiak. Mivel a Newton módszer alkalmazása során is deriválnunk kell az $f$ függvényt, esetünkben az $F^2$ második deriváltjaira is szükségünk lesz. Jelölje $\Delta u$ az $u$ irányú, $\Delta v$ pedig a $v$ irányú, Newton módszer által meghatározott elmozdulást. Tehát:

$$\frac{\partial F^2(u,v)}{\partial u} = -2(D_{ij} - S(u,v))S_u(u,v)$$
$$\frac{\partial F^2(u,v)}{\partial v} = -2(D_{ij} - S(u,v))S_v(u,v)$$

$$\frac{\partial^2 F^2(u,v)}{\partial u^2} = 2S_u(u,v)S_u(u,v) - 2(D_{ij} - S(u,v))S_{uu}(u,v)$$
$$\frac{\partial^2 F^2(u,v)}{\partial v^2} = 2S_v(u,v)S_v(u,v) - 2(D_{ij} - S(u,v))S_{vv}(u,v)$$

$$\Downarrow$$

$$\Delta u = \frac{-\frac{\partial F^2(u_i,v_j)}{\partial u}}{\frac{\partial^2 F^2(u_i,v_j)}{\partial u^2}} =$$

$$= \frac{-(D_{ij} - S(u_i,v_j))S_u(u_i,v_j)}{(D_{ij} - S(u_i,v_j))S_{uu}(u_i,v_j) - S_u(u_i,v_j)^2}$$

$$\Delta v = \frac{-\frac{\partial F^2(u_i,v_j)}{\partial v}}{\frac{\partial^2 F^2(u_i,v_j)}{\partial v^2}} =$$

$$= \frac{-(D_{ij} - S(u_i,v_j))S_v(u_i,v_j)}{(D_{ij} - S(u_i,v_j))S_{vv}(u_i,v_j) - S_v(u_i,v_j)^2}.$$

Az új paraméterértékek az előző módszer végén látott egyszerű összeadással kiszámíthatók:

$$u_i^* = u_i + \Delta u$$
$$v_j^* = v_j + \Delta v.$$

A bemutatott paraméter korrekciós eljárások mindegyike csupán javít a $D_{ij}$ ponthoz tartozó paraméterértékeken, de nem adja meg azok pontos értékét. Ez azt jelenti, hogy amennyiben ezeket az eljárásokat egymás után többször végrehajtjuk adott $D_{ij}$ pont esetén, akkor a hozzá tartozó hibavektor az $S(u,v)$ közelítő felületre merőleges egyeneshez fog közelíteni.

## 3.4. Mérési pontok vágása

Előfordulhat, hogy a rendelkezésre álló mért pontok közül nem akarjuk az összeset figyelembe venni az approximáció során. Ennek az egyik oka lehet, hogy a mért felületet több felületdarabbal akarjuk közelíteni, amelyek

akár különböző reprezentációban lesznek előállítva. A más felületdarabokhoz tartozó pontokat nem kell figyelembe venni az approximáció során. Az elhagyandó pontok megadására két megoldás jöhet szóba:

1. Azokat a térbeli pontokat adjuk meg, amelyeket nem szeretnénk az approximációs lépésben felhasználni. A pontok nehezen határozhatók meg akkor, ha nincsenek mátrixba rendezve, hanem csak egy pontfelhő a bemenet. Ebben az esetben definiálhatunk térbeli téglatesteket vagy bonyolultabb testeket, és a testek belsejébe eső pontokat tekintjük elhagyhatónak. Tegyük fel, hogy a mért pontok egy mátrixba rendezettek, és a mátrix $n+1 \times m+1$-es. Ekkor megadhatunk poligonokat a $[0,n] \times [0,m]$ téglalapon, és azokat a pontokat hagyjuk el, amelyeknek az indexe valamelyik poligon belsejébe esik.

   Ennél a módszernél már az approximációs lépés megkezdése előtt el tudjuk dönteni, hogy mely pontokat kell kihagyni. Ebben az esetben a megjelenítésnél fordulnak elő nehézségek, mivel a paraméterezés megtartása nélkül nem tudjuk eldönteni, hogy a felületet milyen paraméterhelyeken nem kell kiértékelnünk.

2. Paramétertérbeli pontokat adunk meg, és a hozzájuk tartozó térbeli pontokat hagyjuk el a közelítésből. Ebben az esetben már ismernünk kell valamilyen kezdeti paraméterezést, hogy az approximáció előtt meg tudjuk határozni a fölösleges pontokat, viszont megjelenítésnél könnyen el lehet dönteni egy paramétertérbeli pontról, hogy ott a felületet ki kell-e értékelni. Az elhagyandó pontok megadása itt is történhet a paramétertérben megadott poligonokkal.

Ha el tudjuk dönteni egy mért adatról, hogy az fontos-e, akkor az approximációban a korábbi fejezetben tárgyalt lineáris egyenletrendszerekből egyszerűen el kell hagyni a megfelelő tagokat. Mi a második megoldást fogjuk választani, azaz a paramétertérben adott poligonok határozzák meg az elhagyandó pontokat. A minimalizás problémája a következőképpen módosul:

$$\sum_{i=0}^{k} \sum_{j=0}^{l} \chi(u_i, v_j) \left\| D_{ij} - S(u_i, v_j) \right\|^2 \longrightarrow \text{minimum},$$

ahol

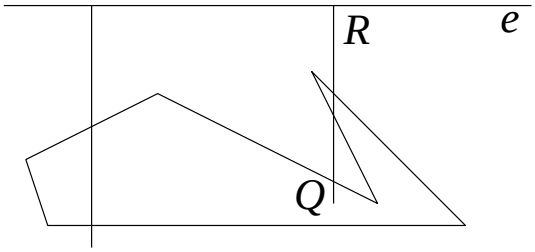
$$\chi(u,v) = \begin{cases} 1, & \text{ha } (u,v) \text{ elhagyandó paraméter} \\ 0, & \text{különben} \end{cases} .$$

Ezután a $\chi$ függvény segítségével már könnyen fel tudjuk írni a megoldandó lineáris egyenletrendszereket. A feladat most már annak meghatározása, hogyan dönthető el hatékonyan egy pontról, hogy belül van-e a paramétertér egy adott poligonján.

### 3.4.1. Poligon belső pontjai

Ebben az részben azt vizsgáljuk meg, hogyan lehet megadni az előző szakaszban szereplő $\chi$ függvényt.

Tegyük fel, hogy adott egy $S$ poligon a síkon pontjaival ($P_0$, $P_1$, ... , $P_n$, $P_n = P_0$), adott a sík egy $Q$ pontja és adott egy $e$ egyenes a poligonon kívül. Legyen $R$ a $Q$-ból $e$-re állított merőleges szakasz talppontja. A $Q$ pontot az $S$ poligonon kívülinek tekintjük, ha páratlan sok olyan $\overline{P_iP_{i+1}}$ szakasz létezik, melyre a $\overline{QR}$ és $\overline{P_iP_{i+1}}$ szakaszok metszik egymást.




3.3. ábra. Poligon belső és külső pontjai.

Tehát elegendő azt megszámolnunk, hogy hány poligon oldallal van közös pontja a $\overline{QR}$ szakasznak. Legyen a vizsgált poligon oldal az $\overline{AB}$ szakasz. Ekkor $Q$, $R$, $A$, $B \in \mathbb{R}^2$, a két egyenes egyenlete pedig:

$$(x - Q)^T n^{(1)} = 0,\ n^{(1)} = \begin{bmatrix} -R_2 + Q_2 \\ R_1 - Q_1 \end{bmatrix},$$
$$(x - A)^T n^{(2)} = 0,\ n^{(2)} = \begin{bmatrix} -B_2 + A_2 \\ B_1 - A_1 \end{bmatrix}.$$

Az $n^{(1)}$ és $n^{(2)}$ vektorok a szakaszok normálvektorai. Két szakasz metszi egymást, ha mindkét szakaszra igaz, hogy végpontjaik a másik szakasznak két különböző oldalán vannak. A $Q$ és az $R$ pont az $\overline{AB}$ szakasz két különböző oldalán van akkor és csak akkor, ha

$$\begin{array}{ccc} (Q - A)^T n^{(2)} \leq 0 & \text{és} & (R - A)^T n^{(2)} \geq 0 \\ & \text{vagy} & \\ (Q - A)^T n^{(2)} \geq 0 & \text{és} & (R - A)^T n^{(2)} \leq 0 \end{array}$$

teljesül. Ugyanezeket a feltételeket kell fogalmazni az $A$ és $B$ pontokra is. Összefoglalva a $\overline{QR}$ és $\overline{AB}$ szakaszok metszik egymást akkor és csak akkor, ha

$$\begin{array}{rcl} ([A - Q]^T n^{(1)})([B - Q]^T n^{(1)}) & \leq & 0 \\ & \text{és} & \\ ([Q - A]^T n^{(2)})([R - A]^T n^{(2)}) & \leq & 0 \end{array}$$

teljesül. Ezzel a számítási módszerrel már könnyedén és hatékonyan tudjuk eldönteni egy paramétertérbeli pontról, hogy egy poligonon belül vagy kívül helyezkedik el.

# 4. fejezet

# Felület approximációs módszerek vizsgálata

Ebben a fejezetben a korábban tárgyalt módszereket vizsgáljuk meg és hasonlítjuk össze őket különböző szempontok szerint. Az algoritmusok bemenete térbeli pontok mátrix formába rendezett halmaza, amelyet mi egy folytonos felület mérésvel kapunk meg. Az algoritmusok kimenete pedig egy B-Spline felület, amit a bemenő pontokon végzett approximációval állítunk elő. A cél az, hogy az eredményül kapott felület minél inkább közelítse meg a bemenetként kapott pontokat, de a felület legyen a lehető legsimább. Ez valójában ellentmondást hordoz, hiszen a legsimább felület a sík lap, ez viszont nagyon kevés ponthalmazt fog jól közelíteni, ezért kompromisszumokra lesz szükségünk.

Az algoritmusokat *Intel Pentium100* processzoron futtattuk, az operációs rendszer *Linux* volt.

## 4.1. Vizsgálati módszerek

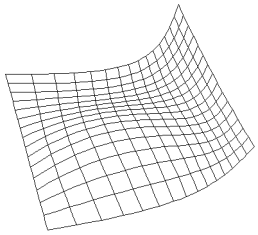
Az algoritmusok egyik legfontosabb vizsgálati szempontja lesz, hogy milyen pontosan kaptuk vissza azt a felületet, amelyet mintavételezve a bemeneti pontokat meghatároztuk. A bemeneti pontok nem minden esetben a kiinduló felület pontjai, azokat véletlenszerű nagyságú és irányú vektorokkal eltoljuk, ezzel egy lokálisan egyáltalán nem sima, de globálisan mégis az eredeti felület alakjához hasonlító ponthalmazt kapunk.

A kiinduló és az eredményül kapott felületet összehasonlíthatjuk numerikusan. Az egyik módszer, hogy a felület és a bemeneti pontok közötti eltérések négyzetének átlagát számoljuk ki. Ha a kiinduló felület egy B-Spline reprezentációval adott felület volt, akkor megvizsgálhatjuk az eredeti- és a kapott kontrollpontok közötti eltérést is.

A másik vizsgálati módszer vizuális, a képernyőn megnézzük a két felületet, és úgy hasonlítjuk össze. A felületek megjelenítésére két módszer terjedt

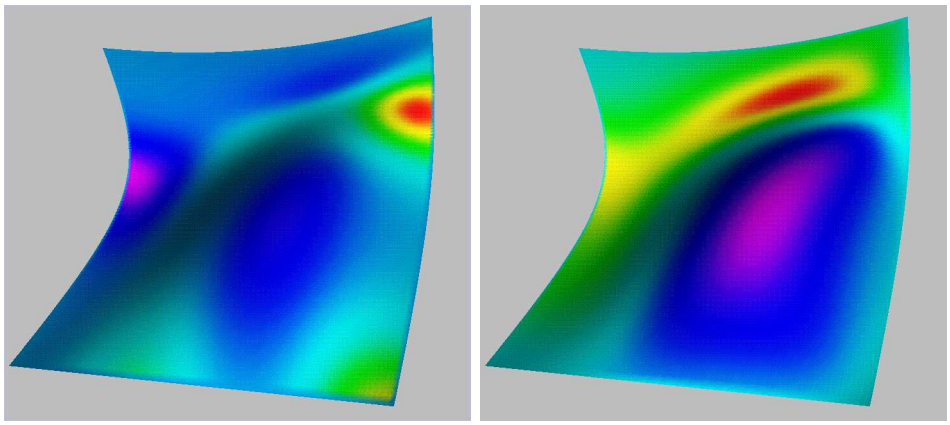
el:

- Paramétervonalas ábrázolás. Ekkor csak a felület paramétervonalait láthatjuk valamilyen sűrűséggel. Ezzel a megjelenítéssel a felület takart részei is látszódnak, viszont nehéz észrevenni a felület kis változásait.



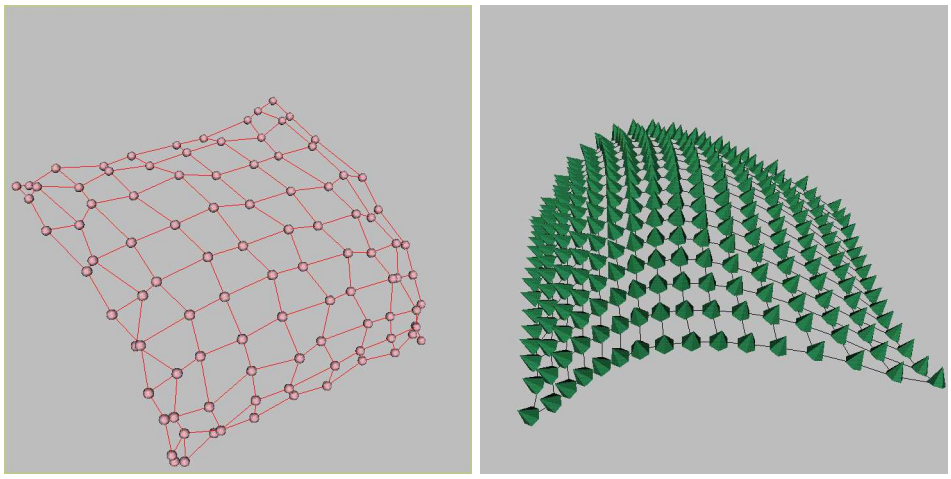
4.1. ábra. Paramétervonalas ábrázolás.

- Árnyékolt ábrázolás. Ezt a módszert használva a teljes felület megjelenik, és a takart részek nem láthatók. A felület színezését fel lehet használni a Gauss- és Minkowski-görbületek jelölésére. Az azonos színű részek azonos görbület értékeket jelentenek. Ez az ábrázolás elsősorban a felület elhajlásának vizsgálatára használható, a felület színváltozása miatt a felületen bekövetkezett kis hirtelen eltérések is jól láthatók.



(a) Gauss görbület (b) Minkowski görbület

4.2. ábra. Árnyékolt ábrázolás.



(a) Kontrollpontok és kontrollpoligon

(b) Normális vektorok és paramétervonalak.

4.3. ábra. Kiegészítő információk megjelenítése.

A két ábrázolást esetleg együtt is alkalmazhatjuk. A felület mellett még meg lehet jeleníteni a felülethez tartozó kontrollpoligont, vagy a felület normálisát is (4.3. ábra).

A kapott eredményen kívül fontos az is, hogy milyen gyors az algoritmus. A sebesség különösen azoknál az algoritmusoknál fontos, amelyek valamilyen iterációt is végeznek.

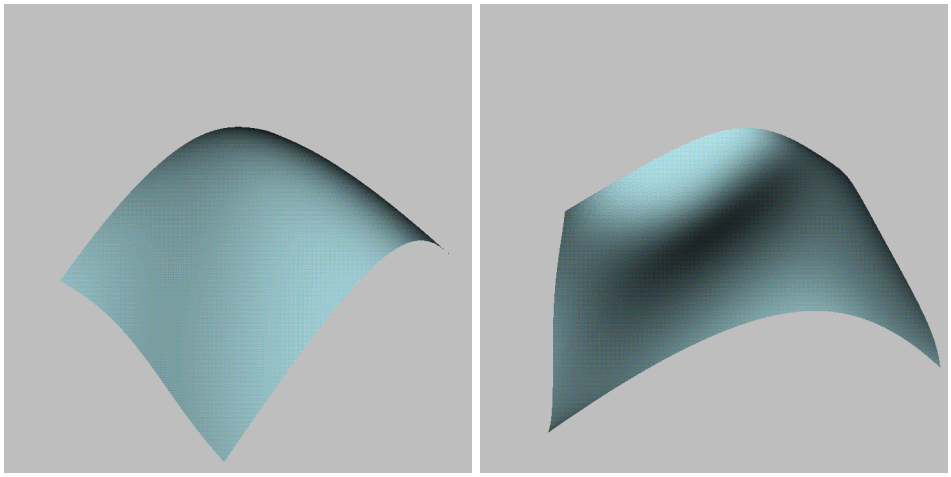
A bemenő adatokat három felületről mintát véve fogjuk megkapni. A három felület B-Spline reprezentációban adott, de a paramétereik különbözők. A felületeket a 4.4. ábrán láthatjuk. A felületek paramétereit a 4.1. táblázat tartalmazza, a knot eloszlás egyenletes.

| | 1. felület | 2. felület | 3. felület |
|---|---|---|---|
| bázisfüggvény fokszáma $u$ irányba | 3 | 4 | 3 |
| bázisfüggvény fokszáma $v$ irányba | 3 | 4 | 3 |
| kontrollpontok száma $u$ irányba | 4 | 5 | 7 |
| kontrollpontok száma $v$ irányba | 4 | 5 | 7 |

4.1. táblázat. A kiinduló felületek paraméterei.

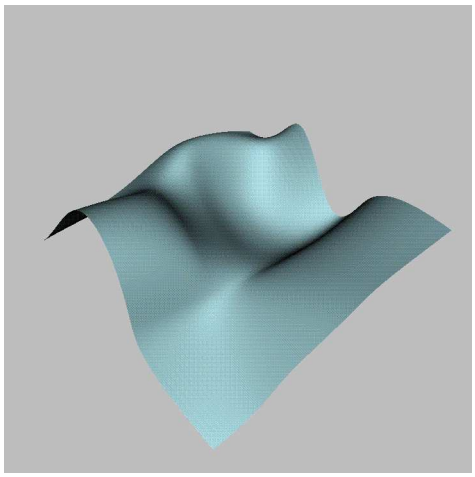
## 4.2. Legkisebb négyzetek módszere

Ebben a szakaszban a legkisebb négyzetek módszerére adott két megoldást hasonlítjuk össze. A bemenő adatokat az előző szakaszban megadott felületek



(a) 1. felület

(b) 2. felület



(c) 3. felület

4.4. ábra. A vizsgálatok során közelített felületek.

| | átlagos hibanégyzet | CPU idő(sec) |
|---|---|---|
| 1. felület | 0.000836745 | 2.40 |
| 2. felület | 0.00076617 | 5.15 |
| 3. felület | 0.000750532 | 10.91 |

4.2. táblázat. Legkisebb négyzetek módszere I.

| | átlagos hibanégyzet | CPU idő(sec) |
|---|---|---|
| 1. felület | 0.000836745 | 1.94 |
| 2. felület | 0.00076617 | 4.33 |
| 3. felület | 0.000750532 | 6.77 |

4.3. táblázat. Legkisebb négyzetek módszere II.

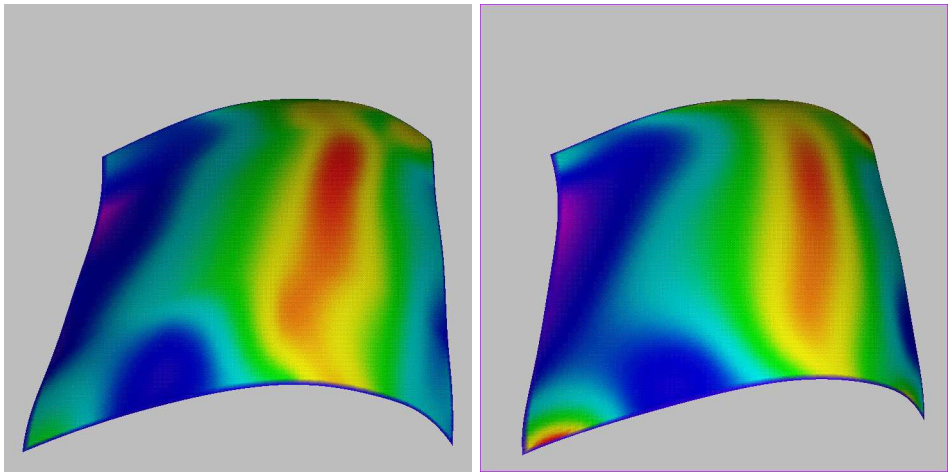
szolgáltatják. A paraméterteret felosztjuk 30×30-as négyzetre és a sarkokhoz tartozó pontokban kiértékeljük a felületet, majd a kapott pontokat eltoljuk véletlenszerű vektorokkal, és ezek a pontok adják a bemenő adatokat. A 4.2. táblázat a 3.1.2. részben tárgyalt algoritmus alkalmazásával kapott megoldások adatait tartalmazza. A kisebb mátrixszal dolgozó algoritmus, amit a 3.1.1. részben adtunk meg, a 4.3. táblázatban szereplő eredményeket adta. Az algoritmust mindhárom felületre alkalmaztuk, a hiba 0.05 volt (a véletlenszerű vektorok abszolut értéke kisebb vagy egyenlő 0.05-el). A felület pontjai benne vannak a $[0,1] \times [0,1] \times [0,1]$ kockában, így a 0.05 5%-os mérési hibát jelent. A közelítő B-Spline paramétereiként ugyanazokat az értékeket használtuk, mint az eredeti felületnél. A knot eloszlás egyenletes. A két megoldás ugyanazt az átlagos hibanégyzetet adja, viszont a futásidők a kisebb mátrixot használó eljárás esetén kisebbek.

Érdemes megvizsgálni, hogy az eljárások milyen eredményt adnak abban az esetben, ha a generált felület nagyobb fokszámú B-Spline, mint a kiinduló felület. A 4.4. táblázat az 1. felületre alkalmazott II. algoritmus eredményeit mutatja, a kontrollpontok száma mind a négy esetben 7 volt mindkét irányban. Ha a fokszám nem ugyanannyi volt, mint a kiinduló B-Spline felület fokszáma, akkor az átlagos hibanégyzet megnőtt, majd a fokszám további növelésével lassan csökkent. A végrehajtási idő nagyobb fokszám esetén megnő, ennek az az oka, hogy a közelítés során sokszor kell kiértékelni a bázisfüggvényeket. A rekurzív kiértékeléshez szükséges idő a fokszám eggyel való növelésével duplájára nő. A 4.5. ábra a 3. és 6. fokú bázisfüggvényekkel kapott felületeket mutatja. Az ábrákon a Minkowski görbület szerinti színezés látható.

A másik lehetőség a paraméterek megváltoztatására, ha több kontrollpontot használunk. A 4.5. táblázatban szereplő értékeket az 1. felület közelítésével kaptuk, a B-Spline fokszáma 3 az $u$ és a $v$ irányban, a knot eloszlás egyenletes. Az I. algoritmust használtuk a közelítéshez. A 4.6. ábra a 16 illetve 256 kontrollponttal előállított felületeket mutatja. Ha több kont-

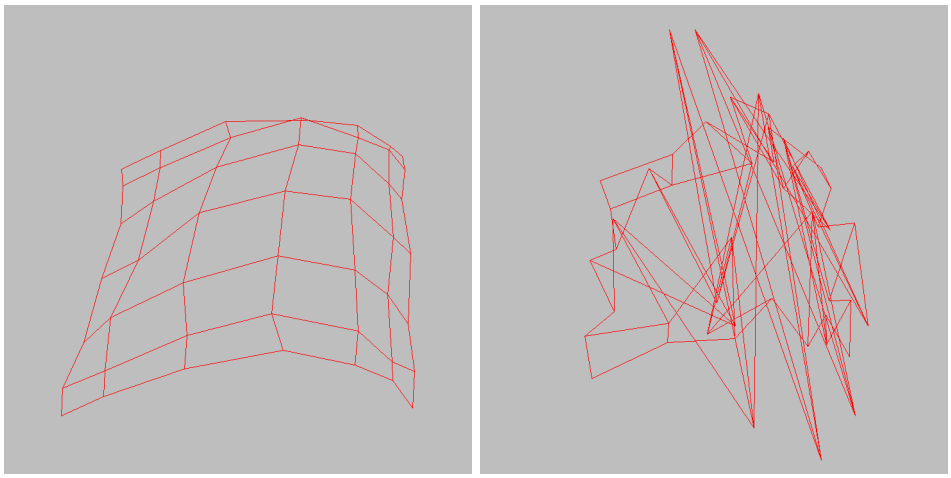
| fokszám(u,v) | átlagos hibanégyzet | CPU idő(sec) |
|---|---|---|
| 3 | 0.00078559 | 7.3 |
| 4 | 0.000785894 | 9.09 |
| 5 | 0.000785884 | 12.77 |
| 6 | 0.000785841 | 17.86 |

4.4. táblázat. Fokszámnövelés.



(a) Fokszám (u,v): 3

(b) Fokszám (u,v): 6



(c) Fokszám (u,v): 3, kontrollpoligon

(d) Fokszám (u,v): 6, kontrollpoligon

4.5. ábra. Fokszámnöveléssel kapott felületek.

| kontrollpontok száma | átlagos hibanégyzet | CPU idő(sec) |
|---|---|---|
| 16 (4×4) | 0.000836745 | 2.40 |
| 64 (8×8) | 0.000795317 | 16.73 |
| 144 (12×12) | 0.000702113 | 74.86 |
| 256 (16×16) | 0.000616019 | 222.98 |

4.5. táblázat. Kontrollpontok számának növelése.

| | hiba nélkül | hibával |
|---|---|---|
| 1. felület | 1.16861e-06 | 0.0188989 |
| 2. felület | 6.03868e-06 | 0.0580196 |
| 3. felület | 5.00293e-07 | 0.0360018 |

4.6. táblázat. Kontrollpontok eltérései.

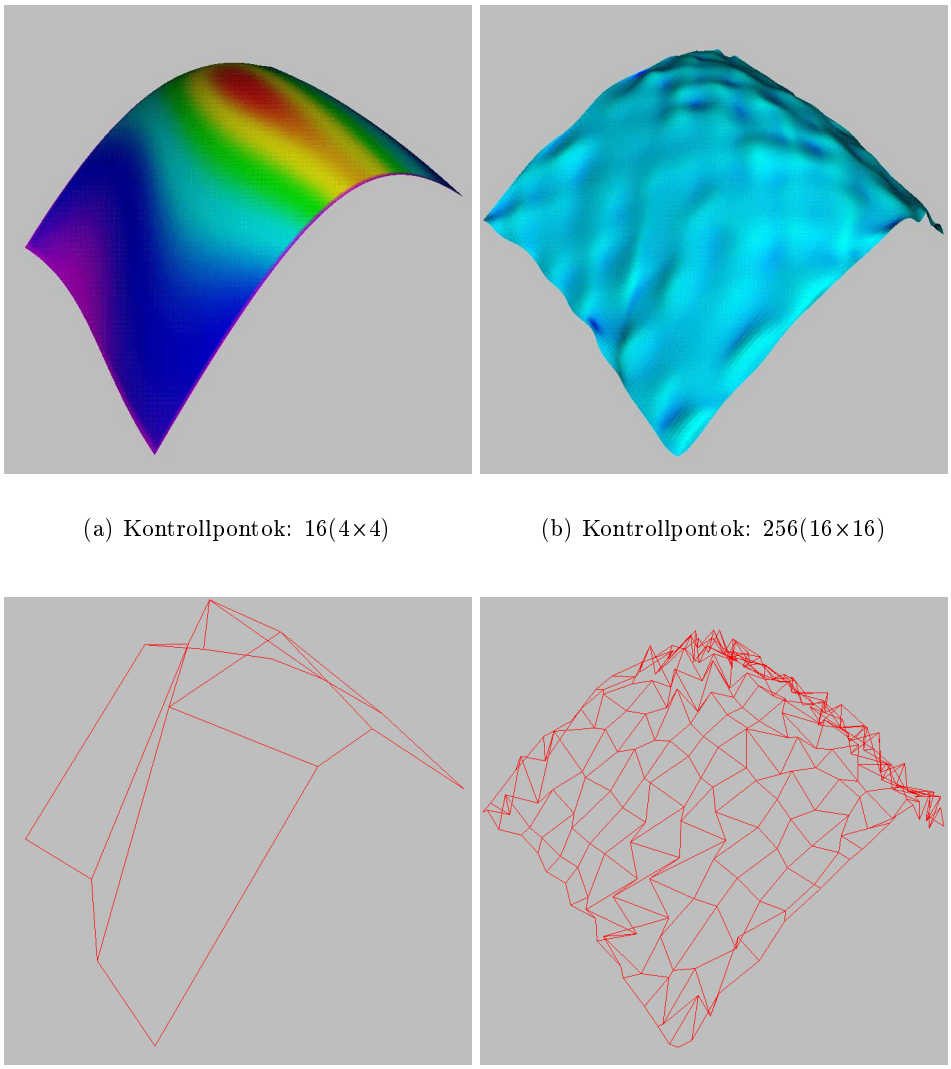
rollpontot használunk, akkor a mérési adatok hibáját is közelíteni fogja a felület, ezért egyenetlenebb felületet kapunk. Ezt a problémát a később tárgyalt funkcionálok meg fogják oldani.

A 4.6. táblázatban szereplő értékek azt mutatják, hogy mennyire pontosan kapjuk vissza a kontrollpontokat a közelítés során. A táblázat sorai a 3 felületre kapott eredményt mutatják, az értékek az eredeti és a számolt kontrollpontok közötti távolságnégyzetek átlagai. Az első oszlop eredményeit úgy kaptuk, hogy az I. algoritmust a felületek hiba nélküli mintavételezésével kapott pontjaira alkalmaztuk. A második oszlop a hibával terhelt adatokra alkalmazott közelítés eredményeit mutatja (a maximális hiba ebben az esetben is 0.05 volt). A közelített felület paraméterei (kontrollpontok, knot eloszlás, fokszám) megegyeznek a kiinduló felület paramétereivel. Ha nem volt hiba a mérések során, akkor az eredmények szerint a kontrollpontokat nagy pontossággal visszakapjuk, a módszer stabil. A 4.7. ábra az 1. felület közelítésével kapott kontrollpoligonokat mutatja a két esetben. A két poligon között szabad szemmel alig kimutatható a különbség.

## 4.3. Funkcionálok

Ahogy azt a 3.2. szakaszban láthattuk, az általunk előállított felületeknek gyakran kell megfelelniük valamilyen simaságot eredményező feltételnek. Bemutattunk három funkcionált, amelyek ilyen simító feltételt fogalmaznak meg. Ebben a szakaszban azokat az eljárásokat vizsgáljuk, amelyek ezeket a funkcionálokat használva simítják az elkészült felületet.

A simaságot vizsgálni nagyon nehéz, mert nem tudjuk adatokkal leírni, hogy egy felület mennyire sima vagy nem. A vizsgálatok éppen ezért nagyrészt vizuálisan történnek. Támpontot az jelent számunkra, hogy a felületeket speciálisan színezzük. Minden pont színe az adott pontban mért Minkowski-görbület értékét reprezentálja. Egy képen belüli azonos színek



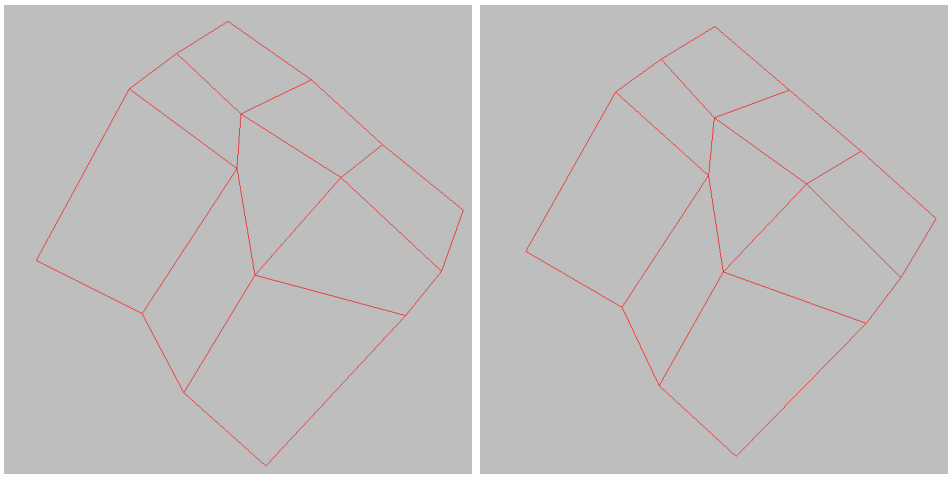
(a) Kontrollpontok: 16(4×4)

(b) Kontrollpontok: 256(16×16)

(c) Kontrollpontok: 16(4×4), kontrollpoligon

(d) Kontrollpontok: 256(16×16), kontrollpoligon

4.6. ábra. Kontrollpontok számának növelésével kapott felületek.



(a) Bemenő adat hibamentes. (b) Bemenő adat hibával terhelt.

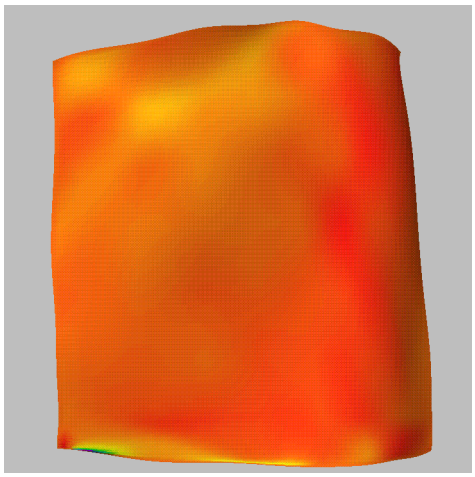
4.7. ábra. 1. felületből kapott kontrollpontok eltérései.

azonos görbületi értéket jelentenek.

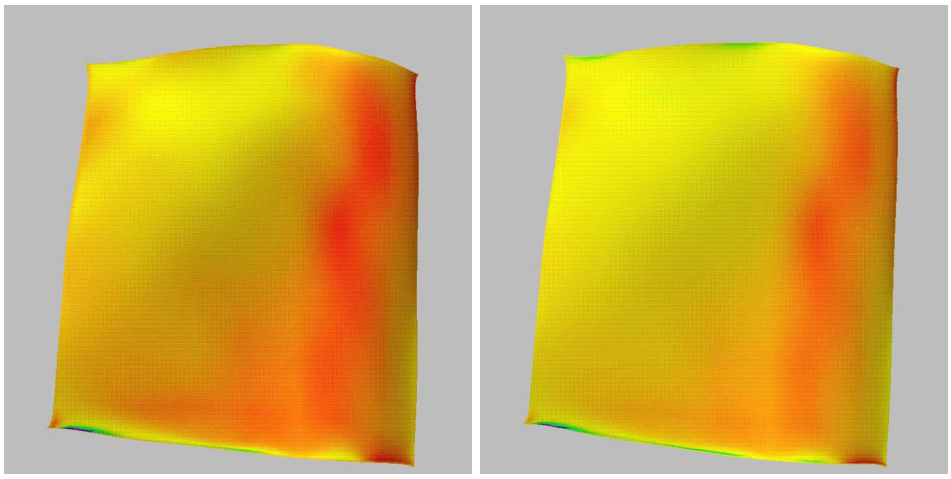
Induljunk ki a 4.8. ábrán látható 2. felületből. Ezen nem végeztünk simítást, ezért a $\mu$ értéke 0. A 4.9., 4.10. és 4.11. ábrákon az eredeti felület különböző eljárással simított képét láthatjuk.

A felület simításakor meg kell határoznunk egy hibahatárt, amelyet még hajlandóak vagyunk elfogadni annak érdekében, hogy egy számunkra jobb, simább felületet kapjunk. Az átlagos távolságnégyzet értékének ezen a hibahatáron belül kell esnie, de legalábbis nem lépheti át nagy mértékben. A program először $\mu = 0.5$-es súllyal kezeli a simaságot leíró funkcionált. Ha az így kapott felület átlagos távolságnégyzetes eltérése a mért adatoktól nem haladja meg a hibahatárt, akkor növeli ezt a fontossági értéket, egyébként csökkenti. $\mu$ értéke 0 és 1 közé kell essen. A következő értéket mindig intervallum felezéssel állapítjuk meg. A program 8 lépésben határozza meg a $\mu$ értékét. Tapasztalataink szerint ez elég ahhoz, hogy nagy pontossággal megközelítse az átlagos távolságnégyzet értéke a hibahatárt.

Nagyon lényeges vizsgálati szempont, hogy a simítás fontosságának mértéke, vagyis a $\mu$ értéke mennyire befolyásolja a távolságnégyzetek változását. Mindhárom funkcionál esetében azt tapasztaltuk, hogy rontja a mért pontok közelítését, ami az elméleti eredményeket is alátámasztja. A 4.7. táblázat adataiból és a rájuk épülő 4.12. grafikonról leolvasható, hogy a görbületi energia minimalizálása a legkevésbé érzékeny a $\mu$ változtatására. A kontrollpont háló simításakor viszont számolni kell azzal, hogy az átlagos távolságnégyzet nagyon megnőhet, ha nagy súllyal ($\mu$) szerepel a simító feltétel.



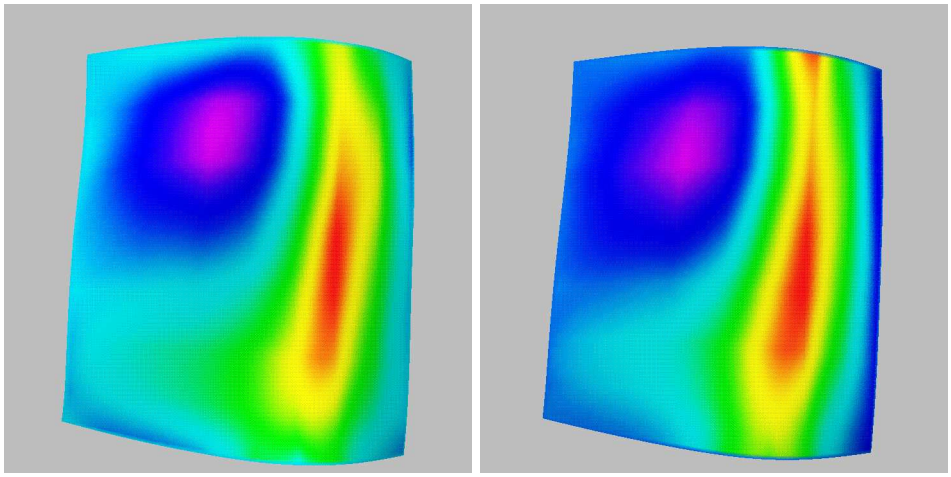
4.8. ábra.
Az eredeti (nem simított) felület. $\mu = 0$, Távolságnégyzet: 0.000782305.



(a) $\mu = 0.664062$, Távolságnégyzet: 0.00159082.

(b) $\mu = 0.804688$, Távolságnégyzet: 0.00330822.

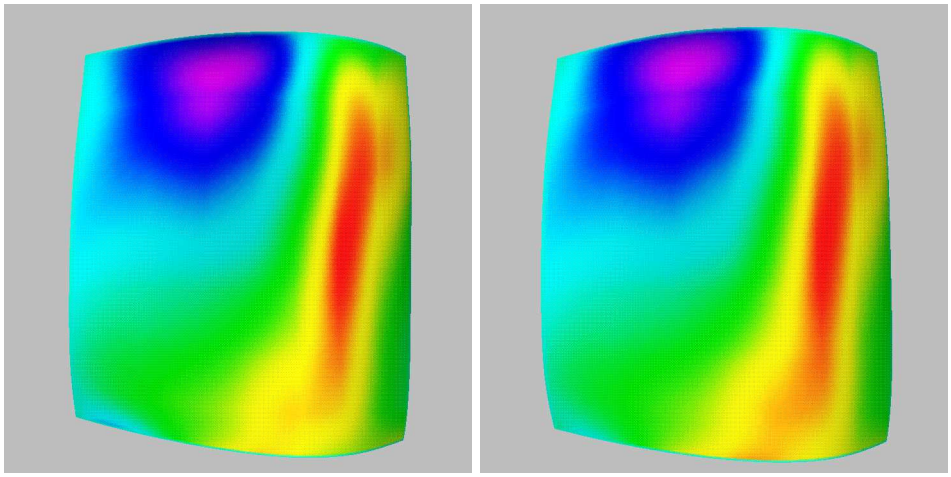
4.9. ábra. Felület minimalizálással simított felület.



(a) $\mu = 0.989197$, Távolságnégyzet: 0.00160427.

(b) $\mu = 0.996765$, Távolságnégyzet: 0.00323295.

4.10. ábra. Görbületi energia minimalizálással simított felület.



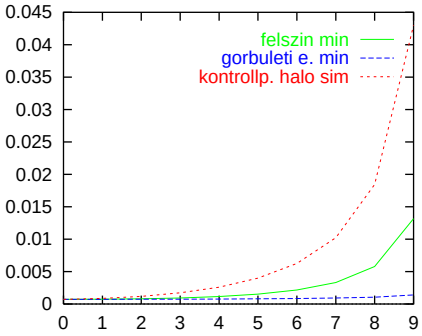
(a) $\mu = 0.414062$, Távolságnégyzet: 0.00162294.

(b) $\mu = 0.570312$, Távolságnégyzet: 0.00315833.

4.11. ábra. Kontrollpoligon minimalizálással kapott felület.

| $\mu$ | Felszín minimalizálása | Görbületi energia minimalizálása | Kontrollpoligon minimalizálása |
|---|---|---|---|
| 0 | 0.000721606 | 0.000721606 | 0.000721606 |
| 0.1 | 0.000739748 | 0.00072448 | 0.000857692 |
| 0.2 | 0.000802553 | 0.000733203 | 0.00117463 |
| 0.3 | 0.000928393 | 0.000748307 | 0.00172296 |
| 0.4 | 0.00114852 | 0.000770995 | 0.00259923 |
| 0.5 | 0.00151978 | 0.000803624 | 0.00398552 |
| 0.6 | 0.00215595 | 0.000850984 | 0.00624844 |
| 0.7 | 0.00331956 | 0.000924157 | 0.0102342 |
| 0.8 | 0.00577739 | 0.00105571 | 0.0184297 |
| 0.9 | 0.0131685 | 0.00139825 | 0.0429893 |
| CPU idő (sec) | 194.67 | 207.31 | 205.24 |

4.7. táblázat.
Átlagos távolságnégyzet változása a 3. felület esetében $\mu$ értékének módosítása mellett.




4.12. ábra.
Átlagos távolságnégyzet változása különböző funkcionálok használata során a 3. Felület esetében.

| Iterációs lépésszám | Vetítéssel | Taylor-sorral | Newton módszerrel |
|---|---|---|---|
| 0. | 0.000745148 | 0.000745148 | 0.000745148 |
| 1. | 0.000387061 | 0.000360401 | 0.000365807 |
| 2. | 0.000365196 | 0.000359109 | 0.000359311 |
| 3. | 0.000360686 | 0.000358385 | 0.000358364 |
| 4. | 0.000358959 | 0.000357719 | 0.000357696 |
| 5. | 0.000357983 | 0.000357096 | 0.000357086 |
| 6. | 0.000357531 | 0.000356799 | 0.000356793 |
| CPU idő (sec) | 545.92 | 153.73 | 167 |

4.8. táblázat.
Átlagos távolságnégyzet változása az 1. felület esetében. Kontrollpontok száma: 10x10

| Iterációs lépésszám | Vetítéssel | Taylor-sorral | Newton módszerrel |
|---|---|---|---|
| 0. | 0.000782305 | 0.000782305 | 0.000782305 |
| 1. | 0.000435505 | 0.000407178 | 0.000416325 |
| 2. | 0.000412292 | 0.000406005 | 0.000406385 |
| 3. | 0.000407838 | 0.000405344 | 0.00040523 |
| 4. | 0.000406018 | 0.00040472 | 0.000404573 |
| 5. | 0.000404926 | 0.000404069 | 0.000403986 |
| 6. | 0.000404394 | 0.000403791 | 0.000403673 |
| CPU idő (sec) | 544.64 | 153.93 | 167.06 |

4.9. táblázat.
Átlagos távolságnégyzet változása a 2. felület esetében. Kontrollpontok száma: 10x10

## 4.4. Paraméter korrekció

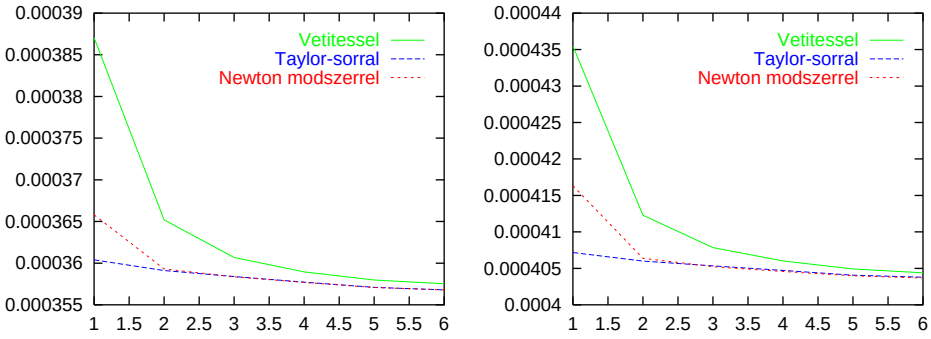
A 3.3.1. szakaszban bemutattuk, hogy a mért pontok és a közelítő felület távolsága, kezdeti paraméterezés esetén nem a megfelelő értéket adja. Módszereket muttatunk arra, hogy hogyan lehet módosítani egy adott mért ponthoz tartozó felületi paraméterértékeket annak érdekében, hogy a kapott távolság a valódi távolságot mutassa. Most ezeket az eljárásokat fogjuk vizsgálni.

Egyazon felületen mindhárom eljárással elvégezzük a paraméter korrekciót. A korrekciós eljárásokat 6 iterációs lépésen keresztül vizsgáljuk. A 0. iterációs lépésben a kezdeti paraméterezéssel kapott értékek szerepelnek. Megfigyelhetjük, hogy mindhárom eljárás ugyanazon értékhez konvergál csak különböző gyorsasággal. Az első eljárás közelít leglasabban a tényleges távolságértékhez. A második és harmadik eljárás közel azonos sebességgel konvergál.

A 4.13. ábrán látható grafikonok a 4.8. és 4.9. táblázatok adatait jelenítik meg. A vízszintes tengely az elvégzett iterációs lépések számát jelenti,

| Iterációs lépésszám | Vetítéssel | Taylor-sorral | Newton módszerrel |
|---|---|---|---|
| 0. | 0.00294967 | 0.00294967 | 0.00294967 |
| 1. | 0.00150967 | 0.00140444 | 0.00148787 |
| 2. | 0.00129486 | 0.00123583 | 0.00125694 |
| 3. | 0.00123009 | 0.00117168 | 0.0011884 |
| 4. | 0.00120066 | 0.00113616 | 0.00115166 |
| 5. | 0.00120114 | 0.00111088 | 0.00112544 |
| 6. | 0.00122973 | 0.00110018 | 0.00111393 |
| CPU idő (sec) | 194.09 | 23.22 | 28.85 |

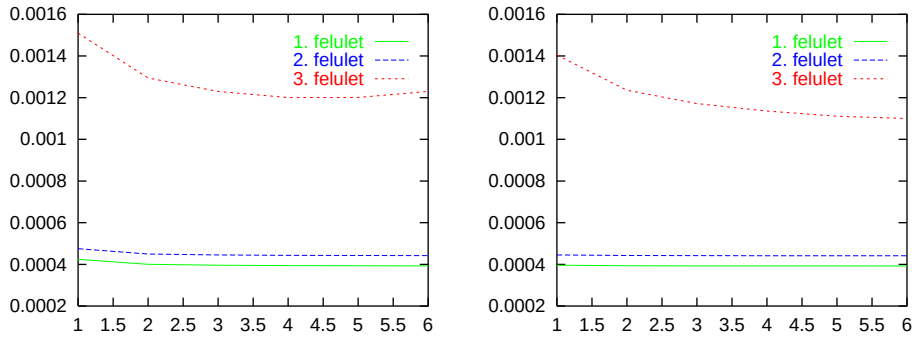
4.10. táblázat.
Átlagos távolságnégyzet változása a 3. felület esetében. Kontrollpontok száma: 5x5




(a) 1. Felület esetében.

(b) 2. Felület esetében.

4.13. ábra.
Átlagos távolságnégyzet változása különböző eljárások használata során.




(a) Vetítéssel. (b) Taylor-sorral.

4.14. ábra. Átlagos távolságnégyzet változása különböző felületek esetén.

míg a függőleges tengely az adott lépés után mért átlagos távolságnégyzetet. A 0. iterációs lépést elhagytuk, mert azonos kezdeti paraméterezésből indult ki mindhárom eljárás, így a kezdeti átlagos távolságnégyzet is megegyezik. Eltérés tehát csak az 1. iterációs lépés elvégzése után figyelhető meg.

A 4.14. ábrán két korrekciós eljárás működését figyelhetjük meg különböző felületek esetében. Jól látható, hogy a vetítéssel módosító algoritmus görbéje az első grafikonon, a 3. felület esetében nem monoton csökkenő, tehát néhány lépés után nem javít, hanem ront a paraméterezésen. A második grafikonon ugyanez nem figyelhető meg, tehát a Taylor-sorral működő eljárás mindig folyamatos javítást eredményezett (megfigyeléseink szerint ugyanez mondható el a Newton módszert alkalmazó algoritmusról is.) Ebből arra következtethetünk, hogy a vetítésses paraméter korrekció hatékonysága erősen függ a vizsgált felülettől.

Lényeges még, hogy milyen gyorsan adják meg az eljárások egy adott kezdeti paraméterezésből kiindulva a mért pontokhoz tartozó optimális paraméterezést. A 4.8., 4.9. és 4.10. táblázatok adatai mutatják, hogy az első eljárás futásidőben is jelentősen lemarad a másik két eljárástól, amelyek közel azonos idő alatt adják meg a végeredményt. A vetítéssel dolgozó eljárás alkalmazásakor minden pont esetén ki kell számolni a paramétervonalak hosszát, ez eredményezi a nagy futásidőket.

## 4.5. Mérési pontok vágása

A 3.4. szakaszban megvizsgáltuk, hogy milyen lehetőségek jöhetnek szóba a mérési pontok vágásának megvalósításához. Mi azt a megoldást választottuk, amikor az elhagyandó pontokat a paramétertérben megadott poligonok határozzák meg. Azt fogjuk megvizsgálni, hogy a pontok kizárása az approximációból hogyan befolyásolja a kontrollpoligont és az eltérésnégyzetek

| fokszám(u,v) | vágás nélkül | vágással |
|---|---|---|
| 3 | 0.00078559 | 0.000779993 |
| 4 | 0.000785894 | 0.000780064 |
| 5 | 0.000785884 | 0.000779822 |
| 6 | 0.000785841 | 0.000779719 |

4.11. táblázat.
Távolságnégyzetek változása a fokszám függvényében, az 1. felület esetén.

| fokszám(u,v) | vágás nélkül | vágással |
|---|---|---|
| 3 | 0.000823847 | 0.000827035 |
| 4 | 0.000822822 | 0.000826087 |
| 5 | 0.0008227 | 0.000825964 |
| 6 | 0.000822412 | 0.000825749 |

4.12. táblázat.
Távolságnégyzetek változása a fokszám függvényében, a 2. felület esetén.

átlagait.

A táblázatok a kiinduló és a közelítő felület pontjai közötti átlagos távolságnégyzeteket hasonlítják össze. A táblázatok első oszlopának eredményeit úgy kaptuk, hogy az összes bemenő pontot figyelembe vettük a közelítés során. A második oszlopban szereplő átlagos távolságnégyzeteket a mérési pontokra alkalmazott vágás utáni közelítés eredményeként kaptuk. A közelítő felületeket a II. algoritmussal határoztuk meg, a vizsgálatokat az 1. és a 2. felületre végeztük el. Vágás esetén az 1. felületből egy téglalap alakú sávot, a 2. felületből pedig egy háromszög alakú területet hagytunk el.

A 4.11. és a 4.12. táblázatok a különböző fokszámú közelítő felületekhez tartozó átlagos hibanégyzeteket hasonlítják össze. Mindegyik közelítő felülethez egyenletes knot eloszlás tartozik, és a kontrollpontok száma 49 (7×7). A 4.11. táblázatban a 3 fokszámú B-Spline felülethez kisebb átlagos hibanégyzet tartozik, mint a magasabb fokszámú felületekhez. Ennek az lehet az oka, hogy az eredeti 1. felület, amiről a mintát vettük, egy 3. fokú B-Spline felület. Az 1. felület esetén, amikor vágtuk a bemenő pontokat, akkor kisebb hibanégyzeteket kaptunk, a 2. felület esetén pedig éppen fordítva, az összes bemenő adat figyelembe vétele esetén volt kisebb a hibanégyzetek átlaga. Vágás esetén néhány kontrollpontra kevesebb megkötés adódik a közelítés során, ezért a vágás után megmaradt pontokat az előállított felület jobban fogja közelíteni, mint vágás nélküli esetben. De amint azt az 1. felület esete mutatja, a megmaradt pontok jobb közelítése nem biztos, hogy ellensúlyozni tudja az elhagyott részekhez tartozó esetleg kisebb hibanégyzeteket, ezért kaphattunk rosszabb eredményt vágás alkalmazása esetén az 1. felületre.

A 4.13. és 4.14. táblázat az azonos fokszámú felületekkel kapott eredményeket mutatja, a kontrollpontok számának függvényében. Az összes közelítő

| kontrollpontok száma | vágás nélkül | vágással |
|---|---|---|
| 16 (4×4) | 0.000812069 | 0.000817699 |
| 25 (5×5) | 0.000806032 | 0.000807511 |
| 36 (6×6) | 0.000797865 | 0.000793167 |
| 49 (7×7) | 0.00078559 | 0.000779993 |

4.13. táblázat.
Távolságnégyzetek változása a kontrollpontok számának függvényében, az 1. felület esetén.

| kontrollpontok száma | vágás nélkül | vágással |
|---|---|---|
| 16 (4×4) | 0.00112126 | 0.00109032 |
| 25 (5×5) | 0.000845686 | 0.000849877 |
| 36 (6×6) | 0.000834935 | 0.000839458 |
| 49 (7×7) | 0.000823847 | 0.000827035 |

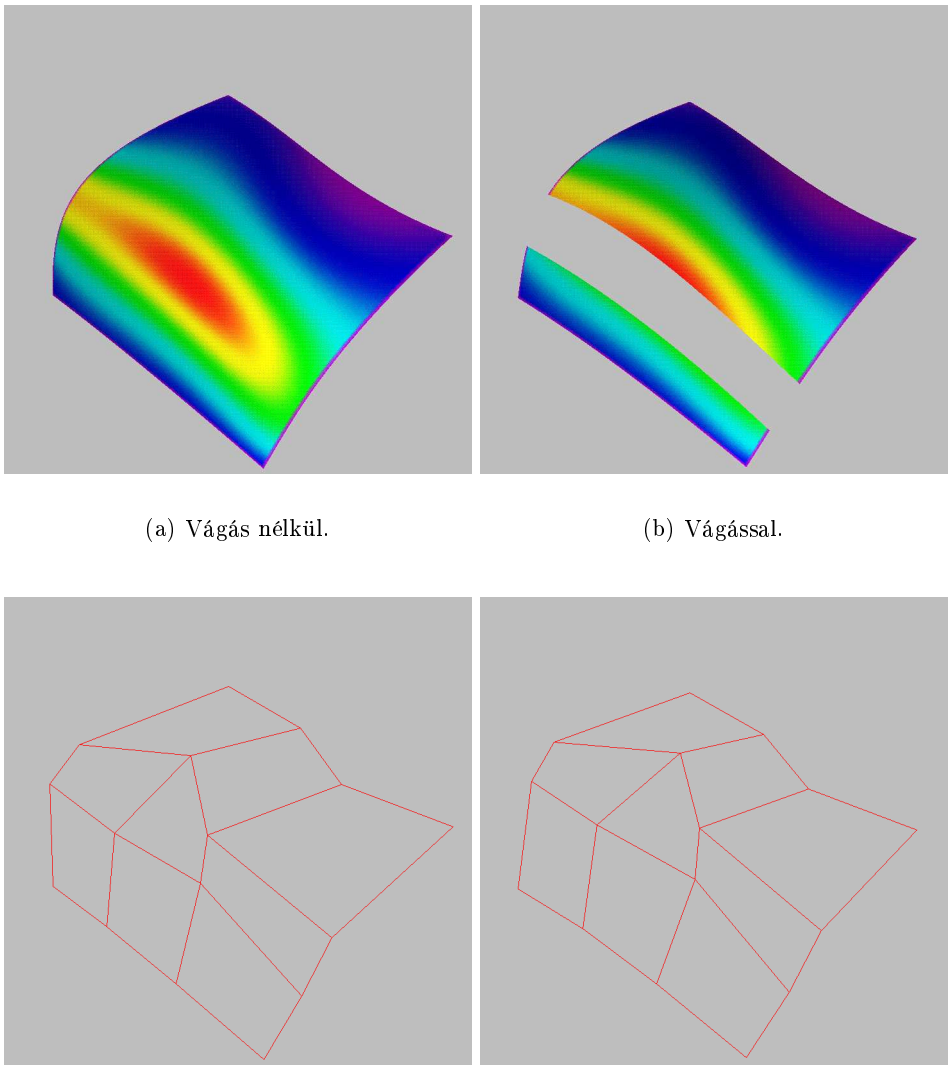
4.14. táblázat.
Távolságnégyzetek változása a kontrollpontok számának függvényében, a 2. felület esetén.

felület 3. fokú B-Spline, egyenletes knot eloszlással. Ha a kontrollpontok számát növeljük, akkor az átlagos hibanégyzet csökken, mivel a felületet jobban hozzá tudjuk igazítani a mért pontokhoz. Az 1. felület esetén (4.13. táblázat) a mérési pontok vágásakor, a 2. felület esetén (4.14. táblázat) pedig az összes pont figyelembe vételekor kaptunk kisebb átlagos távolságnégyzeteket.

A vizsgálati eredményeknek megfelelően az approximáció elvégzéséhez a négyzetes mátrixszot használó algoritmus a jobb, mivel a futásidők rövidebbek és ugyanazt az eredményt adja, mint a másik vizsgált algoritmus.

Ha az approximáció során simító funkcionált is alkalmazunk és annak súlya nagy, akkor használjuk a görbületi energiát minimalizáló funkcionált, mivel így a simított felület a mért pontok közelében marad.

A paraméterkorrekció elvégzéséhez semmiképpen nem ajánljuk a vetítéssel dolgozó módszert annak instabilitása és lassú konvergenciája miatt. A Taylor-soros és a Newton-iterációs módszerek a hatékonyság és az eredmények tekintetében közel azonos eredményt adtak.



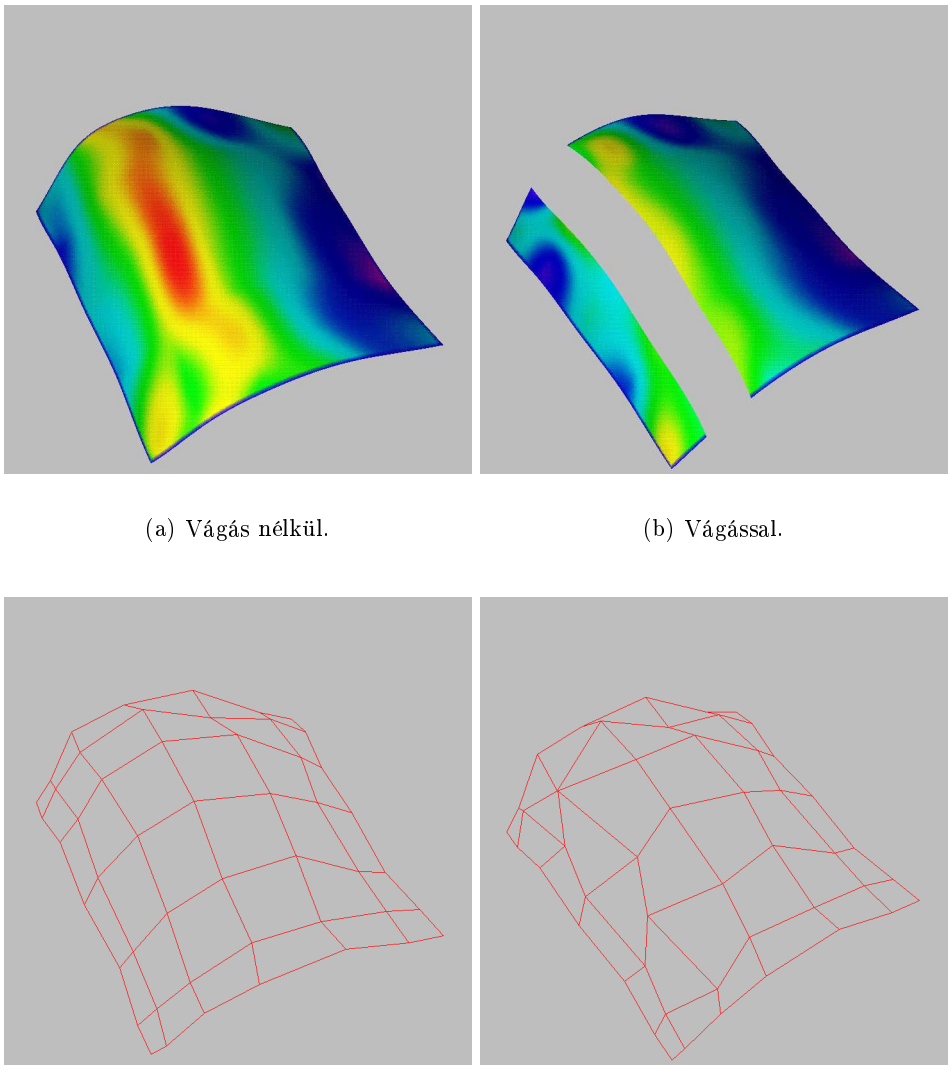
(a) Vágás nélkül.

(b) Vágással.

(c) Kontrollpoligon vágás nélkül.

(d) Kontrollpoligon vágással.

4.15. ábra.
Kontrollpoligon változása a mérési pontok vágásának hatására (kontrollpontok száma: 16 (4×4), fokszám: 3).



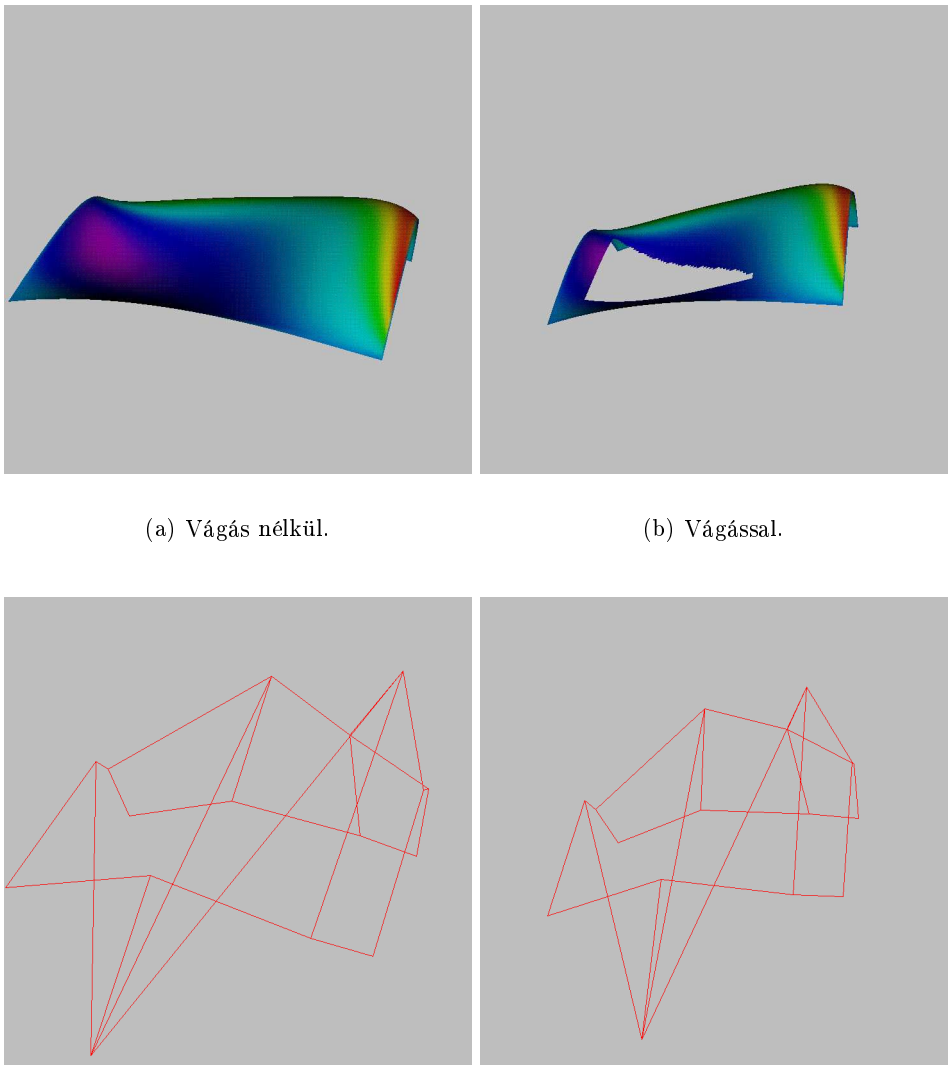
(a) Vágás nélkül.

(b) Vágással.

(c) Kontrollpoligon vágás nélkül.

(d) Kontrollpoligon vágással.

4.16. ábra.
Kontrollpoligon változása a mérési pontok vágásának hatására (kontrollpontok száma: 49 (7×7), fokszám: 3).



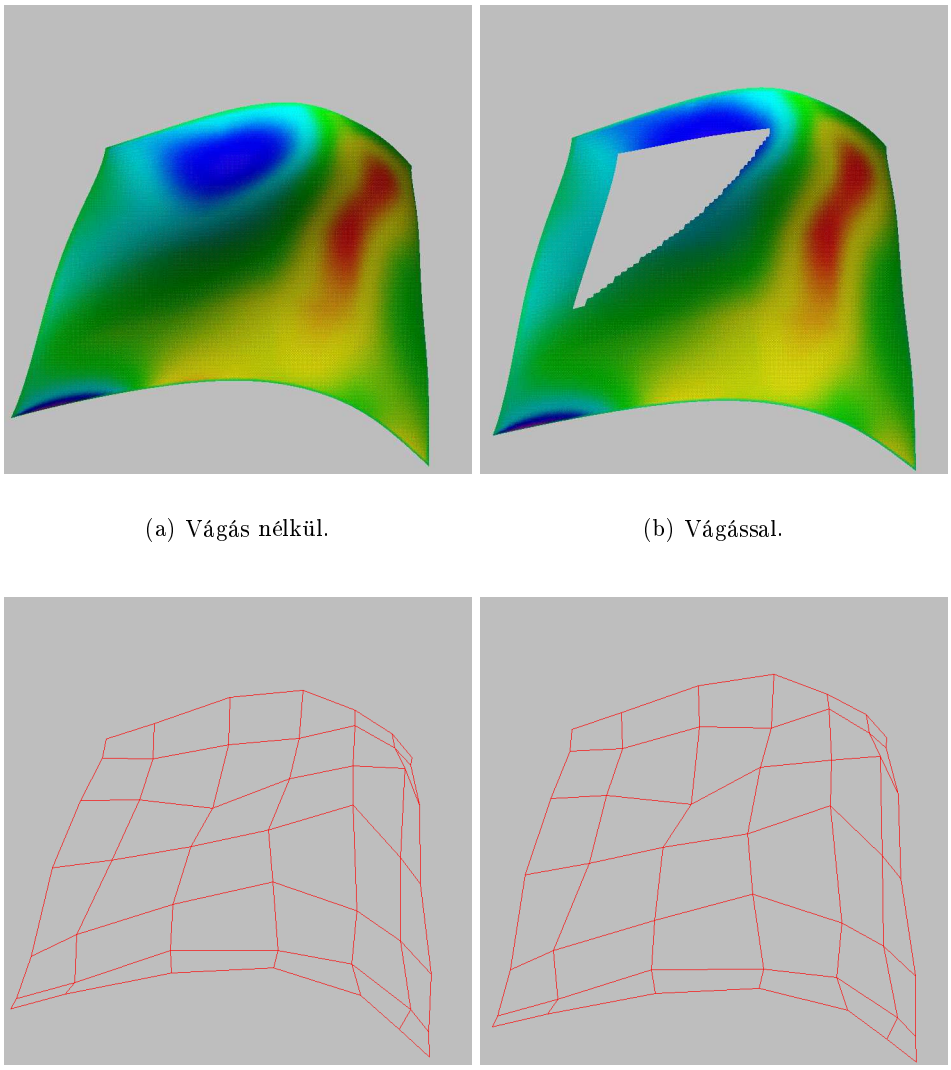
(a) Vágás nélkül.

(b) Vágással.

(c) Kontrollpoligon vágás nélkül.

(d) Kontrollpoligon vágással.

4.17. ábra.
Kontrollpoligon változása a mérési pontok vágásának hatására (kontrollpontok száma: 16 (4×4), fokszám: 3).



(a) Vágás nélkül.

(b) Vágással.

(c) Kontrollpoligon vágás nélkül.

(d) Kontrollpoligon vágással.

4.18. ábra.
Kontrollpoligon változása a mérési pontok vágásának hatására (kontrollpontok száma: 49 (7×7), fokszám: 3).

# 5. fejezet

# Implementáció

Ebben a fejezetben az algoritmusok szemléltetéshez szükséges program modulok megvalósításáról lesz szó. Először áttekintjük a programok implementálásához használt környezetet, majd a rendszer által megvalósított osztályok feladatait vázoljuk. Ezután következik az osztályok és azok member függvényeinek a részletes leírása.

## 5.1. Környezet

A program elkészítéséhez a *UNIX* operációs rendszert választottuk, grafikus környezetként pedig az *X-Window system*-et. A választás egyik oka, hogy a *UNIX* nagy grafikus teljesítményt képes nyújtani és a *Linux* révén mindenki számára ingyenesen elérhető rendszer. Mi is *Linux*-ot használtunk a programozáshoz egy személyi számítógépen.

A térbeli, árnyékolt képek előállításához a Brian Paul által készített *MesaGL* rutinkönyvtárat használtuk, ami az *OpenGL* hálózatról letölthető megvalósítása. Az *OpenGL* egy szabványosított alkalmazói programozói interface számítógépi grafikák előállításához, ami szinte minden platformon elérhető.

Az *OpenGL* csak egy alacsony szintű felületet nyújt, nem teszi lehetővé geometriai alakzatokból összeállított képek könnyű létrehozását. Felhasználtuk a szintén ingyenes *OpenGL*-re épülő *Visualization Toolkit* könyvtárat (a továbbiakban egyszerűen *vtk*), amivel már könnyen összeállíthatunk több különböző geometriai alakzatból álló jeleneteket, és támogatja a felhasználói interakciót is. Számunkra a háromszögekből álló poligonháló megjelenítése volt a legfontosabb, de ezt a *vtk* nagyon jól támogatta.

Az algoritmusok kódolásához a *C++* programozási nyelv mellett döntöttünk, mivel egyszerre nyújtja a sok számítás elvégzéséhez szükséges hatékonyságot, és az objektum-orientált szemlélet világos és könnyen újrafelhasználható kódot eredményez jó programtervezés esetén. A felületek megjelenítéséhez a *Tcl/Tk* interpreteres programozási nyelvet használtuk, mivel ez lehetővé teszi a gyors programfejlesztést és módosítást, és az előre kiszá-

mított felületekből álló jelenetek összeállításához már nem szükséges nagy számítási teljesítmény. A *vtk* jó csatlakozási felületet nyújt mindkét nyelvhez.

A választott alaprendszerek hordozhatósága miatt az elkészült programmodulokat nagyon könnyű átvinni más rendszerekre is.

## 5.2. Osztályok és kapcsolatok

A felületek előállításával kapcsolatos feladatok szempontjából a három legfontosabb osztály a következő:

1. *Base:* Egy tetszőleges knot-vektorhoz tartozó tetszőleges fokú B-Spline bázisfüggvényt reprezentál, le lehet kérdezni a bázisfüggvény paramétereit és a deriváltjait is.

2. *BSplineSurface:* Egy B-Spline felületet reprezentál, két *Base* osztálybeli objektum, a knot-vektorok és a kontrollpontok alapján eltárolja a felületet. Lehetőséget nyújt a felület pontjainak lekérdezésére adott paraméterértékek mellett, ki tudja számítani az $u$ és $v$ irányú deriváltakat adott pontban, a Minkowski- és Gauss-görbületet, a felület normálisát, a felület görbületét, az első és második alapmennyiségeket. Lekérdezhetjük a felület paramétereit.

3. *Approximation:* Ez az osztály valósítja meg a tényleges munkát, a mért adatokból előállítja a felületet. Paraméterként megkapja a mért adatokat, a B-Spline felület fokszámát a két irányhoz, a knot-vektorban lévő pontok számát a két irányhoz, és esetleg egy poligonhalmazt a paramétertérből elhagyandó pontok meghatározásához. Ezek után az osztály képes a következő tevékenységek elvégzésére:

   (a) Legkisebb négyzetek módszere szerint a mért adatokhoz a paramétereknek megfelelő approximáló B-Spline felület létrehozása.

   (b) Paraméter korrekció végrehajtása. Az approximáló felület előállítása után a mért pontokhoz tartozó paramétertérbeli pontokat módosítja oly módon, hogy a mért pont és a neki megfelelő felületi pont közötti egyenes merőleges legyen a felületi ponthoz tartozó érintő síkra. A módosítás után megpróbálja újra közelíteni a mért pontokat a módosított paraméterekkel.

   (c) Funkcionálok kezelése. Az approximáció során nem csak a mért adatoktól való négyzetes eltérést veszi figyelembe az osztály, hanem az eltérések négyzetösszegéhez hozzávesz „valamilyen” súllyal egy funkcionált is, és az így kapott összeget minimalizálja. A funkcionál a felület egyenetlenségét tükrözi, minél kisebb a funkcionál, annál simább a felület.

Az approximáció paramétereként még megadható egy poligonokból álló halmaz, amely azt mutatja, hogy milyen pontokat kell elhagyni a paramétertérből az approximáció során. Ha egy paraméterpár szerepel valamelyik poligonban a halmazból, akkor a neki megfelelő mért adat nem vesz részt a legkisebb négyzetek szerinti közelítésben. Ez lehetővé teszi lyukas felületek közelítését.

A három legfontosabb osztályon kívül még van néhány osztály és adattípus, amelyeket az approximációt végző algoritmusok használnak:

1. *SurfGen:* Ha már előállítottunk egy B-Spline felületet, vagyis már van egy *BSplineSurface* objektumunk, akkor azt egy állományba kell írni, hogy a *vtk* meg tudja jeleníteni. Ezt a feladatot végzi el ez az osztály. Adott távolságonként mintát vesz a felületből és a kapott pontokat a *vtk* számára értelmezhető formában lemezre menti. A felületi pontok mellett elmenti a kontrollpont hálót, és egy adott sűrűségű paramétervonal hálót is. A felületi pontokhoz segédadatként mellékeli a Gauss- és Minkowski-görbületeket, amik a felület megjelenítéséhez szükségesek.

   Az osztály képes olyan formátumú adatok tárolására is, amelyeket később mért adatként fel tud használni approximáláshoz. Ilyenkor a felületen végigmenve mintát vesz és egy véletlenszerű vektorral eltolja a kapott pontokat, így korábban előállított felületekből tesztelési célból mesterségesen mért adatokat tudunk létrehozni.

2. *Householder:* Az approximációs probléma megoldása során felmerülő lineáris egyenletrendszerek megoldását végzi ez az osztály, a *Householder transzformációt* felhasználva.

3. *Exclusion:* Annak eldöntését segíti, hogy egy adott paramétertérbeli ponthoz tartozó mért térbeli pontot figyelembe kell-e venni az approximáció során. Paraméterként egy poligon halmazt kap, és a paramétertér méretét. A megadott poligonoknak a $[0, 1] \times [0, 1]$ négyzetben kell elhelyezkedni, és lineárisan képezi le őket az osztály a paramétertérre.

4. *Polygon:* Egyetlen poligon eltárolásához használt osztály. Le lehet kérdezni, hogy egy pont a poligonon belül vagy kívül helyezkedik el.

5. *Point:* Egy $\mathbb{R}^3$-beli pont eltárolásához használt osztály.

6. *KnotVector:* Valós számokból álló tömb, a knot-vektor pontjainak eltárolásához.

7. *Matrix:* Egy mátrix template, értelmezve vannak rajta a megszokott mátrixműveletek: skalárral való szorzás, mátrix szorzás, mátrix összeadás.

8. *Row:* Egy sorból álló mátrix.
9. *Col:* Egy oszlopból álló mátrix.
10. *Set:* Halmaz típus, az inicializáláskor megadott intervallumba tartozó számokat képes eltárolni. Az *Exclusion* osztály használja.
11. *Functional:* Az approximáció során használt *Q1*, *Q2* és *Q3* funkcionálok ősosztálya. Le lehet kérdezni a funkcionálban szereplő tagok értékét egy adott paraméterpontban.

## 5.3. Referencia

Az osztályok leírása során felmerülő képletekben, ha mást nem mondunk, akkor a változók a következőket jelentik:

| | | |
|---|---|---|
| $P_{ij}$ | = | az $(i, j)$ indexű kontrollpont |
| $k+1$ | = | mért pontok száma az $u$ irányba |
| $l+1$ | = | mért pontok száma a $v$ irányba |
| $n+1$ | = | kontrollpontok száma az $u$ irányba |
| $m+1$ | = | kontrollpontok száma a $v$ irányba |
| $un$ | = | az $u$ irányhoz tartozó bázisfüggvények fokszáma |
| $vn$ | = | az $v$ irányhoz tartozó bázisfüggvények fokszáma |
| $(u_i, v_j)$ | = | a $measured(i, j)$ mért ponthoz tartozó paraméterértékek. |

### Approximation

Header file: `approximation.hh`

ADAT MEMBEREK:

```
const Matrix<Point> &measured;
```

A konstruktor ebben a változóban tárol el egy referenciát a mért adatokat tartalmazó mátrixra.

```
Matrix<Par> parameters;
```

Az aktuális felület paraméterezését tartalmazza, számpárokból álló mátrix, a mérete megegyezik a *measured* mátrix méretével. Az $(i, j)$ indexű mért ponthoz a $parameters(i, j)$ számpár tartozik.

PUBLIC MEMBEREK:

```
Approximation( const Matrix<Point> &measured );
```

*measured:* A mért adatok, amiket közelíteni akarunk.

Inicializálja az objektumot. Az osztály az adatokról nem készít másolatot, csak egy hivatkozást tárol a mátrixra a *measured* adat memberben.

```
double approximate1( uint uK, uint vK, uint un,
       uint vn, BSplineSurface &surface,
       bool default_parameters = true,
       const Exclusion *excl = 0 );
```

*uK:* A kontrollpontok száma az $u$ irányba.
*vK:* A kontrollpontok száma a $v$ irányba.
*un:* A bázisfüggvények fokszáma az $u$ irányba.
*vn:* A bázisfüggvények fokszáma a $v$ irányba.
*surface:* Ebben adja vissza az eredmény felületet.
*default_parameters:* Szükség van-e kezdeti paraméterezés meghatározására.
*excl:* Az érvényben lévő kizárás.

Előállít egy B-Spline felületet a legkisebb négyzetek módszerét alkalmazva. Ha a *default_parameters* hamis, akkor a *parameters* adat memberben szereplő paraméterezést használja fel, ha igaz, akkor pedig a *measured* mátrix elemeinek indexét leképezi a paramétertérre lineárisan, és a kapott paraméterpárt rendeli az indexnek megfelelő ponthoz. Ha az *excl* paramétert nem adjuk meg, akkor az összes mért adatpont részt vesz az approximációban. Ez a művelet egy túlhatározott lineáris egyenletrendszert megoldva határozza meg a legjobb közelítő megoldást, a *Householder* osztályt felhasználva. A megoldandó egyenletrendszer alapmátrixát a *fill_matrix1* és *fill_b1* műveletek számolják ki.

A B-Spline felülethez tartozó knot-vektorok egyenletes eloszlásúak lesznek, a két végpontban $un + 1$ és $vn + 1$ pontok össze lesznek vonva, ezzel az előálló felületet kihúzzuk a kontrollpont háló szélére. A művelet a mért adatoktól vett négyzetes eltérések átlagát adja vissza.

```
void fill_matrix1( Matrix<double> &A,
       const Base &ubase, const Base &vbase,
       const Exclusion *excl = 0 ) const ;
```

*A:* A kitöltendő mátrix.
*ubase:* Az $u$ irányhoz tartozó bázisfüggvény.
*vbase:* A $v$ irányhoz tartozó bázisfüggvény.
*excl:* Az érvényben lévő kizárás.

Az *approximate1* művelet használja a lineáris egyenletrendszer alapmátrixának feltöltésére ezt a membert. Az $A$ mátrix elemei a művelet lefutása után a következők lesznek:

$$A_{r(m+1)+s,p(m+1)+q} = ubase(p, u_r)vbase(q, v_s),$$
$$p \in [0, n],\ q \in [0, m],\ r \in [0, k],\ s \in [0, l].$$

A mátrix csak azokat az itt megadott sorokat fogja tartalmazni, amelyekhez olyan mért adatpont tartozik, amit az *excl* nem zár ki az approximációból.

```
void fill_b1( Matrix<double> &A, uint index,
      const Exclusion *excl ) const;
```

*A:* A lineáris egyenletrendszer alapmátrixa.
*index:* Aktuális koordináta a *measured* mátrixban.
*excl:* Az érvényben lévő kizárás.

A művelet az $A$ mátrix utolsó oszlopát tölti fel az *approximation1* művelet által megoldandó lineáris egyenletrendszernek megfelelően. Az utolsó oszlopba az eredeti egyenletrendszer jobb oldalát kell tenni, ha *Householder transzformációt* szeretnénk alkalmazni. Tehát a művelet az $A$ mátrix utolsó oszlopába a következő vektort teszi:

$$A_{(n+1)(m+1)} = \begin{bmatrix} measured(0,0)[index] \\ measured(0,1)[index] \\ \vdots \\ measured(k,l)[index] \end{bmatrix}.$$

A vektor nem biztos, hogy az összes itt megadott sort tartalmazza, az *excl* paraméternek megfelelően a kizárt pontokhoz tartozó sorok nem kerülnek bele a vektorba. Az *index* azt határozza meg, hogy melyik koordinátát tekintsük a 3 dimenziós mért adatpontokból. A kontrollpontok meghatározásához három lineáris egyenletrendszert kell megoldani, külön egyenletrendszer kell minden koordinátához. A kapott három kontrollpont vektor együtt határozza meg a térbeli kontrollpontokat.

```
double approximate2( uint uK, uint vK, uint un,
      uint vn, BSplineSurface &surface,
      bool default_parameters = true,
      const Exclusion *excl = 0 );
```

*uK:* A kontrollpontok száma az $u$ irányba.
*vK:* A kontrollpontok száma a $v$ irányba.
*un:* A bázisfüggvények fokszáma az $u$ irányba.
*vn:* A bázisfüggvények fokszáma a $v$ irányba.
*surface:* Ebben adja vissza az eredmény felületet.
*default_parameters:* Szükség van-e kezdeti paraméterezés meghatározására.
*excl:* Az érvényben lévő kizárás.

Előállít egy B-Spline felületet a legkisebb négyzetek módszerét alkalmazva. A paraméterek jelentése megegyezik az *approximate1* művelet paramétereivel. Ez a művelet egy egyértelműen meghatározott lineáris egyenlet-

rendszert megoldva határozza meg a legjobb közelítő megoldást, a *Householder* osztályt felhasználva. A megoldandó egyenletrendszer alapmátrixát a *fill_ matrix2* és *fill_ b2* műveletekkel számolja ki.

A knot-vektorok meghatározása ugyanazzal a módszerrel történik, mint az *approximate1* művelet esetében (egyenletes eloszlás, végpontokban összehúzás). A művelet a mért adatoktól vett négyzetes eltérések átlagát adja vissza.

```
void fill_matrix2( Matrix<double> &A,
       const Matrix<double> &base_product,
       const Base &ubase, const Base &vbase,
       const Exclusion *excl ) const;
```

*A:* A lineáris egyenletrendszer alapmátrixa.
*base_ product:* Bázisfüggvények szorzatmátrixa.
*ubase:* Az $u$ irányhoz tartozó bázisfüggvény.
*vbase:* A $v$ irányhoz tartozó bázisfüggvény.
*excl:* Az érvényben lévő kizárás.

Az *approximate2* eljárás által megoldott lineáris egyenletrendszer alapmátrixát tölti fel. A *base_ product* egy segédmátrix az $A$ elemeinek a gyorsabb kiszámításához, a bázisfüggvények szorzatait tartalmazza. A pontos leírása a *base_ product* függvény member ismertetésénél szerepelni fog. Az $A$ mátrix elemei a következők lesznek:

$$A_{r(m+1)+s,p(m+1)+q} = \sum_{i=0}^{k}\sum_{j=0}^{l} N_p^{(un)}(u_i)N_q^{(vn)}(v_j)N_r^{(un)}(u_i)N_s^{(vn)}(v_j),$$
$$p, r \in [0, n],\ q, s \in [0, m].$$

Ha az *excl* kizár egy pontot az approximációból, akkor a neki megfelelő tag nem szerepel az összegben.

```
void fill_b2( Matrix<double> &A,
       const Matrix<double> &bp, uint index,
       const Base &ubase, const Base &vbase,
       const Exclusion *excl = 0 ) const;
```

*A:* A lineáris egyenletrendszer alapmátrixa.
*index:* Aktuális koordináta a *measured* mátrixban.
*excl:* Az érvényben lévő kizárás.

A művelet az $A$ mátrix utolsó oszlopát tölti fel az *approximation2* művelet által megoldandó lineáris egyenletrendszernek megfelelően. Az utolsó oszlopba az eredeti egyenletrendszer jobb oldalát kell tenni, ha *Householder transzformációt* szeretnénk alkalmazni. Tehát a művelet az $A$ mátrix utolsó

oszlopába a következő vektort teszi:

$$A_{(n+1)(m+1)} = \begin{bmatrix} \sum_{i=0}^{k}\sum_{j=0}^{l} measured(i,j)[index]\ N_0^{(un)}(u_i)N_0^{(vn)}(v_j) \\ \sum_{i=0}^{k}\sum_{j=0}^{l} measured(i,j)[index]\ N_0^{(un)}(u_i)N_1^{(vn)}(v_j) \\ \vdots \\ \sum_{i=0}^{k}\sum_{j=0}^{l} measured(i,j)[index]\ N_n^{(un)}(u_i)N_m^{(vn)}(v_j) \end{bmatrix}.$$

Az *excl* paraméternek megfelelően nem biztos, hogy a sorokban szereplő összegek az összes tagot tartalmazzák. Az eredményben a *measured* vektornak csak az *index*-edik koordinátái szerepelnek.

```
double approximate3( uint uK, uint vK, uint un,
       uint vn, BSplineSurface &surface,
       comp_fun act_fun,
       const Exclusion *excl = 0 );
```

*uK:* A kontrollpontok száma az $u$ irányba.
*vK:* A kontrollpontok száma a $v$ irányba.
*un:* A bázisfüggvények fokszáma az $u$ irányba.
*vn:* A bázisfüggvények fokszáma a $v$ irányba.
*surface:* Ebben adja vissza az eredmény felületet.
*act_fun:* A paraméterezés javítását végző függvény.
*excl:* Az érvényben lévő kizárás.

Előállít egy B-Spline felületet a legkisebb négyzetek módszerét alkalmazva. A paraméterek jelentése megegyezik az *approximate1* művelet paramétereivel. Az *approximate2* eljárást alkalmazva létrehoz egy kezdeti felületet, majd minden mért pontra meghívja az *act_fun* member függvényt a paraméterezés javítására. A javított paraméterezéssel újra elvégzi az approximációt. A javítást 6 alkalommal ismétli meg, és minden egyes alkalommal kiírja az átlagos négyzetes eltéréseket. Az utolsó lépésben kialakult átlagos hibanégyzetet adja vissza. Az *act_fun* függvény szignatúrája a következő:

```
void (Approximation::*)( const BSplineSurface &,
       uint, uint );
```

```
void compute_length( const BSplineSurface &surface,
       uint i, uint j,
       double &lu, double &lv ) const;
```

*surface:* A generált B-Spline felület.
*i,j:* A javítandó paraméterérték indexe a *parameters* mátrixban.
*lu,lv:* A felületi ponthoz tartozó paramétervonalak hossza.

A függvény a *correct_parameter* member végrehajtásához szükséges paramétervonal hosszakat számolja ki. Az *lu*-ban adja vissza az $u$ irányhoz tartozó, a $v$-ben a $v$ irányhoz tartozó paramétervonalak hosszát.

```
void correct_parameters( const BSplineSurface
      &surface, const Exclusion *excl,
      comp_fun act_fun );
```

*surface:* A generált B-Spline felület.
*excl:* Az érvényben lévő kizárás.
*act_fun:* A használandó javító függvény.

A member az *excl* által kizárt pontokat kivéve minden pontra meghívja a paraméterként kapott *act_fun* javító függvényt.

```
void correct_parameter( const BSplineSurface
      &surface, uint i, uint j );
```

*surface:* A generált B-Spline felület.
*i,j:* A javítandó paraméterek indexe.

A member a kapott *surface* felülethez tartozó paraméterezést próbálja meg javítani az $(i, j)$ indexű mért pontban. Az aktuális paraméterezést a $parameters(i, j)$ tartalmazza.

A felületet az adott pontban megegyezőnek tekinti a ponthoz tartozó érintősíkkal, és a paraméterértékeket a mért pont és felületi pont érintősíkra eső merőleges vetülete alapján módosítja. Képletben:

$$\begin{aligned} u_i' &= u_i + \frac{r_u}{\|r_u\|}[D_{ij} - r(u_i, v_j)]\frac{measured.n() - 1}{lu}, \\ v_j' &= v_j + \frac{r_v}{\|r_v\|}[D_{ij} - r(u_i, v_j)]\frac{measured.m() - 1}{lv}, \end{aligned}$$

ahol $(u_i, v_j)$ a régi, $(u_i', v_j')$ pedig az új paraméterértékek, $D_{ij} = measured(i, j)$, $r(u_i, v_j) =$ az $(u_i, v_j)$ paraméterekhez tartozó felületi pont, $lu =$ az $r(u_i, v_j)$-hez tartozó $u$ irányú paramétervonal hossza, $lv =$ az $r(u_i, v_j)$-hez tartozó $v$ irányú paramétervonal hossza.

```
void correct_parameter2( const BSplineSurface
      &surface, uint i, uint j );
```

*surface:* A generált B-Spline felület.
*i,j:* A javítandó paraméterek indexe.

A member a kapott *surface* felülethez tartozó paraméterezést próbálja meg javítani az $(i, j)$ indexű mért pontban. Az aktuális paraméterezést a $parameters(i, j)$ tartalmazza.

A felületet Taylor-sorba fejti az $(u_i, v_j)$ pont körül az első deriváltat tartalmazó tagig, és ez alapján határozza meg az új paraméterértékeket:

$$\begin{aligned} u_i' &= u_i + \frac{\langle r_v^2, [D_{ij} - r(u_i, v_j)]r_u\rangle - \langle r_u r_v, [D_{ij} - r(u_i, v_j)]r_v\rangle}{r_u^2 r_v^2 - (r_u r_v)^2}, \\ v_j' &= v_j + \frac{\langle r_u^2, [D_{ij} - r(u_i, v_j)]r_v\rangle - \langle r_u r_v, [D_{ij} - r(u_i, v_j)]r_u\rangle}{r_u^2 r_v^2 - (r_u r_v)^2}, \end{aligned}$$

ahol $(u_i, v_j)$ a régi, $(u'_i, v'_j)$ pedig az új paraméterértékek, $D_{ij} = measured(i,j)$, $r(u_i, v_j) =$ az $(u_i, v_j)$ paraméterekhez tartozó felületi pont.

```
void correct_parameter3( const BSplineSurface
      &surface, uint i, uint j );
```

*surface:* A generált B-Spline felület.
*i,j:* A javítandó paraméterek indexe.

A member a kapott *surface* felülethez tartozó paraméterezést próbálja meg javítani az $(i,j)$ indexű mért pontban. Az aktuális paraméterezést a $parameters(i,j)$ tartalmazza.

A hibavektor deriváltjának zérushelyét próbálja meghatározni oly módon, hogy kiszámolja a hibavektor deriváltjára a *Newton iteráció* egy lépését:

$$\begin{aligned} u'_i &= u_i - \frac{[D_{ij} - r(u_i, v_j)]r_u}{[D_{ij} - r(u_i, v_j)]r_{uu} - r_u^2}, \\ v'_j &= v_j - \frac{[D_{ij} - r(u_i, v_j)]r_v}{[D_{ij} - r(u_i, v_j)]r_{vv} - r_v^2}, \end{aligned}$$

ahol $(u_i, v_j)$ a régi, $(u'_i, v'_j)$ pedig az új paraméterértékek, $D_{ij} = measured(i,j)$, $r(u_i, v_j) =$ az $(u_i, v_j)$ paraméterekhez tartozó felületi pont.

```
double approximate4( uint uK, uint vK, uint un,
      uint vn, BSplineSurface &surface,
      Functional &functional, double max_err,
      bool default_parameters,
      const Exclusion *excl );
```

*uK:* A kontrollpontok száma az $u$ irányba.
*vK:* A kontrollpontok száma a $v$ irányba.
*un:* A bázisfüggvények fokszáma az $u$ irányba.
*vn:* A bázisfüggvények fokszáma a $v$ irányba.
*surface:* Ebben adja vissza az eredmény felületet.
*functional:* A felület minőségét jellemző funkcionál.
*max_err:* Megengedett maximális átlagos eltérés négyzet.
*default_parameters:* Szükség van-e kezdeti paraméterezés meghatározására.
*excl:* Az érvényben lévő kizárás.

Előállít egy B-Spline felületet a legkisebb négyzetek módszerét alkalmazva, a paraméterként kapott funkcionált is felhasználva a felület minőségének javítására. A paraméterek jelentése megegyezik az *approximate1* művelet paramétereivel. A *max_err* paraméter azt határozza meg, hogy mekkora az a maximális átlagos eltérés négyzet, amennyit megengedünk.

Az eljárás a következő kifejezést minimalizálja:

$$\mu \sum_{i,j} \|D_{ij} - surface(u_i, v_j)\| + (1-\mu) functional \longrightarrow \text{minimum}.$$

A cél a $\mu$ paraméterérték meghatározása oly módon, hogy a kapott átlagos hibanégyzet a megengedett maximális átlagos hibanégyzeten belül legyen. Ez a $\mu$ a $[0,1]$ intervallumon belül lesz, az eljárás bináris kereséssel határozza meg a megfelelő értéket a $[0,1]$ intervallumon.

```
double solve_one_step( uint step, BSplineSurface
      &surface, const Matrix<double> base_product,
      const Functional &functional, double u,
      const Exclusion *excl );
```

*step:* Hanyadik approximációs lépés.
*surface:* Ebben adja vissza a generált felületet.
*base_product:* Segéd mátrix a számításhoz.
*functional:* A felület minőségét jellemző funkcionál.
*u:* Az eltérést minimalizáló tag együtthatója.
*excl:* Az érvényben lévő kizárás.

Egy approximációs lépést végez el a megadott $u$ súllyal és *functional* funkcionállal. Az eredményül kapott B-Spline felületet a *surface* változóban adja vissza. A standard output-ra kiírja az átlagos négyzetes eltérést és a használt $1-u$ súlyt. A lineáris egyenletrendszer megoldásához a *Householder* osztályt használja, az egyenletrendszer alapmátrixát a *fill_matrix4* és *fill_b4* member függvények segítségével tölti fel.

A visszatérési érték a négyzetes eltérések átlaga.

```
void fill_matrix4( Matrix<double> &A,
      const Matrix<double> &base_product,
      const Functional &functional, double u,
      const Base &ubase, const Base &vbase,
      const Exclusion *excl ) const;
```

*A:* A feltöltendő mátrix.
*base_product:* Segéd mátrix a számításhoz.
*functional:* A felület minőségét jellemző funkcionál.
*u:* Az eltérést minimalizáló tag együtthatója.
*ubase:* Az $u$ irányhoz tartozó bázisfüggvény.
*vbase:* A $v$ irányhoz tartozó bázisfüggvény.
*excl:* Az érvényben lévő kizárás.

A *solve_ one_ step* műveletben szereplő lineáris egyenletrendszer alapmátrixát tölti fel. A mátrix a következő lesz:

$$A_{r(m+1)+s,p(m+1)+q} = u \sum_{i=0}^{k} \sum_{j=0}^{l} N_p^{(un)}(u_i) N_q^{(vn)}(v_j) N_r^{(un)}(u_i) N_s^{(vn)}(v_j) +$$

$$+(1-u)\mathit{functional.derivate}(r,s,p,q),\ p,r \in [0,n],\ q,s \in [0,m].$$

Az összegből az *excl* paraméternek megfelelő tagok hiányoznak. A *functional* objektum *derivate* művelete egy funkcionál deriváltjának számolja ki egy tagját. A *derivate* művelet paramétereinek a leírása a *Functional* osztálynál szerepel.

```
void fill_b4( Matrix<double> &A,
      const Matrix<double> &bp, uint index,
      double u, const Base &ubase, const Base
      &vbase, const Exclusion *excl ) const;
```

*A:* A lineáris egyenletrendszer alapmátrixa.
*bp:* Segédmátrix, a bázisfüggvények szorzatait tartalmazza.
*index:* Aktuális koordináta a *measured* mátrixban.
*u:* Az eltérést minimalizáló tag együtthatója.
*ubase:* Az $u$ irányhoz tartozó bázisfüggvény.
*vbase:* A $v$ irányhoz tartozó bázisfüggvény.
*excl:* Az érvényben lévő kizárás.

A művelet az $A$ mátrix utolsó oszlopát tölti fel a *solve_ one_ step* művelet által megoldandó lineáris egyenletrendszernek megfelelően. Az utolsó oszlopba az eredeti egyenletrendszer jobb oldalát kell tenni, ha *Householder transzformációt* szeretnénk alkalmazni. Tehát a művelet az $A$ mátrix utolsó oszlopába a következő vektort teszi:

$$A_{(n+1)(m+1)} = u \begin{bmatrix} \sum_{i=0}^{k} \sum_{j=0}^{l} \mathit{measured}(i,j)[\mathit{index}]\ N_0^{(un)}(u_i) N_0^{(vn)}(v_j) \\ \sum_{i=0}^{k} \sum_{j=0}^{l} \mathit{measured}(i,j)[\mathit{index}]\ N_0^{(un)}(u_i) N_1^{(vn)}(v_j) \\ \vdots \\ \sum_{i=0}^{k} \sum_{j=0}^{l} \mathit{measured}(i,j)[\mathit{index}]\ N_n^{(un)}(u_i) N_m^{(vn)}(v_j) \end{bmatrix}.$$

Az *excl* paraméternek megfelelően nem biztos, hogy a sorokban szereplő összegek az összes tagot tartalmazzák. Az eredményben a *measured* mátrixban szereplő térbeli pontoknak csak az *index*-edik koordinátái szerepelnek.

```
void init_parameters( uint un, uint vn );
```

*un:* A mért adatokat tartalmazó mátrix sorainak száma.
*vn:* A mért adatokat tartalmazó mátrix oszlopainak száma.

A művelet a *parameters* adat membert tölti fel a kezdeti paraméterértékekkel. A mért adatokat tartalmazó mátrix elemeihez a paraméterezést

úgy határozza meg, hogy az elemek indexét lineárisan leképezi a paramétertérre. A paramétertér azonban a $[0, measured.n() - 1] \times [0, measured.m() - 1]$ téglalap, így ez a leképezés a következőt jelenti: $parameters(i, j) = (i, j)$.

```
void fill_knotvector( KnotVector &kv, uint n,
      double min, double max ) const;
```

*kv:* A feltöltendő knot-vektor.
*n:* A knot-vektorhoz tartozó bázisfüggvény fokszáma.
*min:* A knot-vektorban szereplő első csomópont értéke.
*max:* A knot-vektorban szereplő utolsó csomópont értéke.

A paraméterként kapott *kv* knot-vektort tölti fel. A knot-vektor első csomópontja a *min*, az utolsó csomópontja a *max* lesz, és $n + 1$ csomópont össze lesz vonva a knot-vektor elején és a végén. A belső pontok közötti távolság egyenlő lesz, az eredmény egy egyenletes eloszlású knot-vektor.

```
void fill_controls( Matrix<Point> &controls,
      Matrix<double> &x,
      uint uK, uint vK, uint index ) const;
```

*controls:* Ebben a mátrixban adja vissza a kontrollpontokat.
*x:* A kontrollpontokat tartalmazó vektor.
*uK:* A kontrollpontok száma az $u$ irányba.
*vK:* A kontrollpontok száma a $v$ irányba.
*index:* Az $x$ vektor a kontrollpontokból csak az *index*-edik koordinátát tartalmazza.

A *controls* mátrixba a megfelelő pozícióba és koordináta helyre átmásolja az $x$ vektorban kapott kontrollpont koordinátákat. Az $x$ vektorban a kontrollpontok folytonosan helyezkednek el. A *controls* mátrix elemeinek csak az *index*-edik koordinátáját változtatja. A *controls* mátrix a következő lesz (csak az *index*-edik koordinátát adjuk meg):

$$controls[index] = \begin{bmatrix} x[0], & \dots & x[vK-1] \\ \vdots & \ddots & \vdots \\ x[(uK-1)vK], & \dots & x[(uK-1)vK+vK-1] \end{bmatrix}.$$

```
double count_diff( const BSplineSurface &surface,
      const Exclusion *excl ) const;
```

*surface:* Egy érvényes B-Spline felület.
*excl:* Érvényben lévő kizárás.

A mért adatok és a nekik megfelelő felületi pontok közötti négyzetes eltérések átlagát adja vissza. Az *excl* paraméter által meghatározott pontokat kihagyja a számításból amikor végigmegy a felületen.

```
double count_matrix_sum( const Matrix<double> &bp,
       const Base &ubase, const Base &vbase,
       uint r, uint s, uint p, uint q,
       const Exclusion *excl ) const;
```

*bp:* Segédmátrix, a bázisfüggvények szorzatát tartalmazza.
*ubase:* Az $u$ irányhoz tartozó bázisfüggvény.
*vbase:* A $v$ irányhoz tartozó bázisfüggvény.
*i,j:* Az alapmátrix aktuális sorához tartozó kontrollpont indexei.
*m,n:* Az alapmátrix aktuális oszlopához tartozó kontrollpont indexei.
*excl:* Az érvényben lévő kizárás.

Az *fill_ matrix2* és *fill_ matrix4* műveletek használják a feladathoz tartozó lineáris egyenletrendszer alapmátrixának meghatározásához. Ez a művelet az alapmátrix egy elemét számolja ki. A paraméterként kapott *bp* mátrix tartalmát a *base_ product* member tárgyalásánál ismertetjük. A függvény a következő értéket adja vissza:

$$\sum_{i=0}^{k}\sum_{j=0}^{l} N_p^{(un)}(u_i)N_q^{(vn)}(v_j)N_r^{(un)}(u_i)N_s^{(vn)}(v_j),\ p,r\in[0,n],\ q,s\in[0,m].$$

Az összegből elmaradnak az *excl* által meghatározott pontok.

```
void fill_base_product( Matrix<double> &A,
       const Base &ubase, const Base &vbase ) const;
```

*A:* A feltöltendő segéd mátrix.
*ubase:* Az $u$ irányhoz tartozó bázisfüggvény.
*vbase:* A $v$ irányhoz tartozó bázisfüggvény.

A member a *count_ matrix_ sum* által használt *base_ product* mátrixot tölti fel. Az eredményt az $A$ mátrixban adja vissza. Az $A$ mátrix tartalma a következő lesz:

$$A_{r(m+1)+s,p(m+1)+q} = N_p^{(un)}(u_r)N_q^{(vn)}(v_s),$$
$$p,r\in[0,n],\ q,s\in[0,m].$$

```
bool solve_problem( const Matrix<double> &A,
       Matrix<double> &x ) const;
```

*A:* A megoldandó egyenletrendszer alapmátrixa és jobboldala.
*x:* Keresett vektor.

Az algoritmus a következő lineáris egyenletrendszert oldja meg a *Householder* osztályt felhasználva:

$$A\begin{bmatrix} x \\ -1 \end{bmatrix} = \underline{0}.$$

A member a paraméterként kapott $A$ mátrixot nem változtatja meg, arról egy lokális másolatot készít mielőtt meghívja a *Householder* objektum megfelelő műveletét.

### Householder

Header file: `householder.hh`

ADAT MEMBEREK:

```
double E;
```

Azt a hibát tartalmazza, amin belül nullának tekint az osztály egy számot.

PUBLIC MEMBEREK:

```
HouseHolder( double E = 1e-16 );
```

*E:* Hibakorlát.

Inicializálja az objektumot.

```
void solve( Matrix<double> &A, Matrix<double> &x );
```

*A:* A megoldandó lineáris egyenletrendszer együtthatói.
*x:* Ebben adja vissza az eredményt.

Megoldja a paraméterekkel adott lineáris egyenletrendszert. Az $A$ mátrixot a megoldás során elrontja. A megoldott egyenlet:

$$A \begin{bmatrix} x \\ -1 \end{bmatrix} = \underline{0}.$$

### SurfGen

Header file: `surfgen.hh`

ADAT MEMBEREK:

```
uint un, vn;
```

A felületből a mintavételezés során az $u$ irányban $un$, a $v$ irányban $vn$ helyen vesz mintát. Összesen $unvn$ pontot fog kiírni a lemezre.

PUBLIC MEMBEREK:

```
SurfGen();
```

Inicializálja az objektumot, az $un$ és a $vn$ 50 lesz alapértelmezés szerint. Inicializálja a véletlenszám generátort is.

```
void set_param( uint un, uint vn );
```

*un, vn:* Az új mintavételezési gyakoriság.

Az *un* és *vn* védett adat membereket állítja be a kapott paraméterértékekre.

```
bool write_surface( const char *name,
      const BSplineSurface &s,
      const Exclusion *excl = 0 );
```

*name:* Az állományok neve, amelyeket létre kell hozni.
*s:* A bemeneti B-Spline felület.
*excl:* Az érvényben lévő kizárás.

A művelet a paraméterként kapott *s* felületről mintát vesz az *un* és *vn* adat membereknek megfelelő gyakorisággal. A paraméter vonalakon halad végig, a befutott paramétervonalak a paramétertérben egyenlő távolságra vannak egymástól, az *excl* által meghatározott pontokat kihagyja. Három állomány jön létre a megadott névvel, de különböző kiterjesztéssel:

1. Paramétervonalakat tartalmazó állomány, ennek a kiterjesztése `.par`.
2. Az egész felületet tartalmazó állomány, minden ponthoz hozzá lesz rendelve a hozzá tartozó Minkowski-görbület. A kiterjesztés `.surf`.
3. Az egész felületet tartalmazó állomány, minden ponthoz hozzá lesz rendelve a hozzá tartozó Gauss-görbület. A kiterjesztés `.gauss`.

A létrehozott állományok a *vtk* függvényeivel olvashatók. A felületet a *showpoly* program képes megjeleníteni.

```
bool write_real( const char *name, double maxerr,
      const BSplineSurface &s );
```

*name:* Az állomány neve, amelyet létre kell hozni.
*maxerr:* A véletlen hibavektorok legnagyobb megengedett mérete.
*s:* A bemeneti B-Spline felület.

Ez a member a *real2spl* program bemeneti formátumának megfelelő állományt képes generálni. Az *un* és *vn* adat membereknek megfelelően végigmegy a felületen, mintát vesz, a kapott térbeli pontokat egy véletlen méretű és irányú vektorral eltolja, majd elteszi egy mátrixba. A mátrixban az elemek a mintavételük sorrendjében szerepelnek. A mátrix mérete $un \times vn$. Az eltoláshoz használt vektorok abszolut értéke kisebb a *maxerr* paraméter értékénél. A létrehozott állomány kiterjesztése `.real`.

```
bool write_controls( const char *name,
      const BSplineSurface &s);
```

*name:* Az állomány neve, amelyet létre kell hozni.

*s:* A bemeneti B-Spline felület.

Az adott *s* felülethez tartozó kontrollpontokat írja ki a *name* nevű állományba, a kiterjesztése `.cp` lesz.

PRIVATE MEMBEREK:

```
bool make_points( vtkPolyData *spolygon,
      vtkPolyData *ppolygon,
      const BSplineSurface &s );
```

*spolygon:* A felület pontjait tartalmazó poligon.
*spolygon:* A paramétervonalak pontjait tartalmazó poligon.

A *write_ surface* member használja ezt a függvényt, a felületről mintát vesz és a kapott pontokat hozzárendeli az *spolygon* és *ppolygon* paraméterekhez. Az egyik a felületet, a másik a paramétervonalakat fogja tartalmazni. A visszatérési érték akkor igaz, ha hiba történt a futás közben.

```
bool make_scalars( vtkPolyData *spolygon,
      const BSplineSurface &s, int curvature );
```

*spolygon:* A poligon, aminek a pontjaihoz értékeket kell rendelni.
*s:* A bemeneti B-Spline felület.
*curvature:* Minkowski- vagy Gauss-görbületet kell számolni.

A paraméterként kapott poligonban szereplő pontokhoz hozzárendeli a görbületeket. Ha a *curvature* paraméter 0, akkor Minkowski-görbületet, ha 1, akkor Gauss-görbületet számol. Igazat ad vissza akkor, ha hiba történt a számítás közben.

```
void make_cells( vtkPolyData *spolygon,
      vtkPolyData *ppolygon, const Exclusion *excl = 0 );
```

*spolygon:* A felület pontjait tartalmazó poligon.
*ppolugon:* A paramétervonalak pontjait tartalmazó poligon.
*excl:* Az érvényben lévő kizárás.

Az *spolygon*-hoz hozzárendel háromszög sávokat, amik a felületet fogják reprezentálni. A *ppolygon*-ban szereplő pontokhoz vonalakat rendel, ezek lesznek a paramétervonalak. Figyelembe veszi az érvényes kizárást.

```
Point mess( const Point &source, double maxerr );
```

*source:* Bemeneti vektor.
*maxerr:* Hibakorlát.

A *source* paraméterben adott vektorhoz hozzáad egy véletlenszerűen generált vektort. A hozzáadott vektor abszolut értéke kisebb, mint *maxerr*, iránya és mérete is véletlenszerű.

### Vector

Header file: `vector.hh`

ADAT MEMBEREK:

```
ELEM x, y, z;
```

A vektor belső reprezentációja a három koordinátának megfelelő érték eltárolásával történik. Az *ELEM* típus a template osztály paramétere.

PUBLIC MEMBEREK:

```
ostream &operator << <>( ostream &s1, const Vector<ELEM> &a );
istream &operator >> <>( istream &s2, Vector<ELEM> &a );
bool operator < <>( const Vector<ELEM> &a,
      const Vector<ELEM> &b );
bool operator == <>( const Vector<ELEM> &a,
      const Vector<ELEM> &b );
Vector<ELEM> operator + <>( const Vector<ELEM> &a,
      const Vector<ELEM> &b );
Vector<ELEM> operator + <>( const Vector<ELEM> &a );
Vector<ELEM> operator - <>( const Vector<ELEM> &a,
      const Vector<ELEM> &b );
Vector<ELEM> operator - <>( const Vector<ELEM> &a );
ELEM operator * <>( const Vector<ELEM> &a,
      const Vector<ELEM> &b );
Vector<ELEM> operator * <>( const ELEM &a,
      const Vector<ELEM> &b );
Vector<ELEM> operator / <>( const Vector<ELEM> &a,
      const ELEM &b );
Vector<ELEM> vector_mul<>( const Vector<ELEM> &a,
      const Vector<ELEM> &b );
Vector &operator += ( const Vector &b );
Vector &operator -= ( const Vector &b );
```

*a,b:* A műveletek bemeneti vektorai.
*s1:* Kimeneti *stream*.
*s2:* Bemeneti *stream*.

Mindegyik függvény a nevének megfelelő műveletet végzi el a vektoron vagy vektorokon.

```
Vector();
Vector( int null );
Vector( const ELEM &x, const ELEM &y, const ELEM &z );
```

*null:* Csak a 0 szám lehet.

*x,y,z:* A hátom koordináta érték.

Ezek a műveletek inicializálják a vektort. Az első művelet semmilyen értéket nem tesz a vektorba. A második művelet paramétere csak a 0 lehet, és a null vektor lesz az incializálás eredménye. A harmadik művelet a megadott értékeket teszi a megfelelő adat memberekbe.

### Set

Header file: `set.hh`

ADAT MEMBEREK:

```
uchar *data;
```

A halmaz belső reprezentációja. A halmaznak egy előjel nélküli karakterekből álló tömb felel meg, amelynek a mérete akkora, hogy a teljes halmaz még éppen belefér. Ha a halmazba betehető elemeket nullával kezdve megsorszámozzuk, akkor neki a *data* tömbben két index felel meg. Az egyik azt adja meg, hogy hanyadik tömbelemben szerepel a neki megfelelő bit, ez az *index div karakter méret* művelettel kapható meg. A másik azt adja meg, hogy a karakteren belül jobbról hanyadik bit felel meg az elemnek. Ezt az *index mod karakter méret* adja meg. Ha az *index*-edik elemnek megfelelő bit a *data* memberben 1, akkor és csak akkor tekinti az elemet a halmazhoz tartozónak.

```
int min, max;
```

A *min* a legkisebb, a *max* pedig a legnagyobb elemet jelenti, amit a halmazban eltárolhatunk.

PUBLIC MEMBEREK:

```
class iterator;
```

A halmaz rendelkezik a *Standard Template Library*-ben megszokott *iterator* osztállyal.

```
ostream &operator << ( ostream &out,
      const Set &set );
istream &operator >> ( istream &in, Set &set );
```

*out:* Kimeneti *stream*.
*in:* Bemeneti *stream*.
*set:* A használt halmaz objektum.

A műveletek a megadott *stream*-re írják ki, illetve olvassák be a halmaz objektumot.

```
Set( int min = 0, int max = 0 );
```

```
Set( const Set &set );
```

*min:* A halmazba betehető legkisebb elem.
*max:* A halmazba betehető legnagyobb elem.
*set:* Egy másik halmaz.

Az első konstruktor egy üres halmazt hoz létre, amelyik a $[min, max]$ intervallumban szereplő elemek tárolására képes.

A második konstruktor a paraméterként kapott *set* halmazt másolja le önmagára.

```
virtual Set();
```

A halmaz destruktora, felszabadítja a *data* számára lefoglalt memóriaterületet.

```
iterator begin();
iterator end();
const iterator begin() const;
const iterator end() const;
```

A halmaz elejére illetve a halmaz végére mutató *iterator* objektumokat adnak vissza.

```
Set &operator = ( const Set &set );
```

*set:* A lemásolandó halmaz.

A művelet lefutása után a paraméterként kapott *set* halmaz és az aktuális objektum ugyanazokat az elemeket fogják tartalmazni, és az ábrázolható tartományuk is meg fog egyezni.

```
void resize( int min, int max );
```

*min:* A halmazba betehető legkisebb elem.
*max:* A halmazba betehető legnagyobb elem.

Megváltoztatja a halmazba tehető elemek tartományát. A művelet eredménye egy üres halmaz lesz.

```
bool in( int elem ) const;
```

*elem:* A keresendő elem.

Igazat ad vissza, ha a megadott *elem* szerepel a halmazban, hamisat egyébként.

```
Set &put( int elem );
```

*elem:* A halmazba betett új elem.

A halmazba beteszi a paraméterként kapott *elem*-et.

```
Set &get( int elem );
```

*elem:* A halmazból kivett elem.

A halmazból kiveszi a megadott elemet, ha nem volt benne, akkor nem történik változás.

```
Set &empty();
```

Kiüríti a halmazt.

```
bool is_empty() const;
```

Visszaadja, hogy a halmaz üres-e.

```
uint size() const;
```

Visszaadja, hogy hány elem van a halmazban.

PRIVATE MEMBEREK:

```
void index( int elem, uint &i, uint &j ) const;
```

*elem:* A lekérdezett elem.
*i,j:* A lekérdezett elem indexeit ezekben adja vissza.

Az $i$ és $j$ változókban visszaadja az *elem data* memberbeli indexét.

### Base

Header file: `bspline.hh`

ADAT MEMBEREK:

```
typedef Array<double> KnotVector;
typedef Vector<double> Point;
typedef Array<Point> Points;
```

Publikus típusok a felületek kezeléséhez.

```
typedef Map<double,uint> Multis;
```

Ez a típus a knot-vektorban lévő multiplicitások kezeléséhez használható. Tartalmazza a knot-vektorban szereplő összes csomópontot, és azt, hogy az adott csomópont hányszor szerepel a knot-vektorban.

```
KnotVector kv;
```

A bázisfüggvényekhez tartozó knot-vektort tartalmazza. Egy *Base* osztály nem csak egy bázisfüggvényt, hanem egy knot-vektorhoz tartozó bázisfüggvény halmazt képes kezelni.

```
Multis m;
```

A *kv* knot-vektorban lévő multiplicitások. A bázisfüggvény deriválhatóságának meghatározásához van szükség a multiplicitásra.

```
uint n;
```

A bázisfüggvény fokszáma.

PUBLIC MEMBEREK:

```
ostream &operator << ( ostream &o, const Base &b );
istream &operator >> ( istream &i, Base &b );
```

*o:* Kimeneti *stream*.
*i:* Bemeneti *stream*.
*b:* A használt *Base* objektum.

Egy *Base* osztálybeli objektumot írnak ki, illetve olvasnak be egy *stream*-ről.

```
Base();
Base( uint n, const KnotVector &kv );
Base( uint n, uint k, double t0 = 0.0, double t1 = 1.0 );
```

*n:* A bázisfüggvény fokszáma.
*kv:* A bázisfüggvényekhez tartozó knot-vektor.
*k:* A knot-vektorban szereplő csomópontok száma.
*t0:* Az első csomópont.
*t1:* Az utolsó csomópont.

Konstruktorok, inicializálják az objektumot. Az első konstruktor nem hoz létre használható bázisfüggvényt, a paramétereit nem definiálja. A második konstruktor egy $n$-ed fokú bázisfüggvényt hoz létre, a hozzá tartozó knot-vektor *kv*. A harmadik konstruktor egy $n$-ed fokú bázisfüggvényt hoz létre, a hozzá tartozó knot-vektor $k$ csomópontot fog tartalmazni és egyenletes eloszlású lesz a $[t0, t1]$ intervallumon.

```
double operator () ( uint i, double t ) const;
```

*i:* A bázisfüggvény indexe.
*t:* A bázisfüggvény paramétere.

Kiértékeli az *i.* bázisfüggvényt a $t$ helyen, és a helyettesítési értéket adja vissza. A paramétereknek a következő feltételeket kell kielégíteniük: $0 \leq i \leq$ knot-vektor mérete $-1-(n+1)$, ahol $n$ a bázisfüggvény fokszáma, $t_0 \leq t \leq t_1$, ahol $t_0$ és $t_1$ a knot-vektorhoz tartozó legkisebb és legnagyobb csomópont. A következő rekurzív képletet használja a számításhoz:

$$N_i^{(n)}(t) = \frac{t - t_i}{t_{i+n} - t_i} N_i^{(n-1)}(t) + \frac{t_{i+n+1} - t}{t_{i+n+1} - t_{i+1}} N_{i+1}^{(n-1)}(t).$$

```
double derivate( uint i, double t, uint k, bool &error ) const;
```

*i:* A bázisfüggvény indexe.
*t:* A bázisfüggvény paramétere.

*k:* Hanyadik derivált.
*error:* Történt-e hiba.

A member kiszámolja az *i.* bázisfüggvény *k.* deriváltját a $t$ helyen. Ha nem deriválható a függvény $k$-szor, akkor az *error* változóba hiba kerül. A deriválhatóság eldöntéséhez az $m$ adat membert használja, ami a knot multiplicitásokat tartalmazza. A paraméterekkel szembeni követelmény: $0 \leq i \leq$ knot-vektor mérete $-1-(n+1)$, ahol $n$ a bázisfüggvény fokszáma, $t_0 \leq t \leq t_1$, ahol $t_0$ és $t_1$ a knot-vektorhoz tartozó legkisebb és legnagyobb csomópont, $k \geq 0$. A derivált meghatározásához a következő rekurzív képletet használja:

$$N_i^{(n)}(t)^k = n \left( \frac{N_i^{(n)}(t)^k}{t_{i+n} - t_i} + \frac{N_{i+1}^{(n)}(t)^k}{t_{i+n+1} - t_{i+1}} \right).$$

```
double min_t() const;
double max_t() const;
```

Visszaadják azt a legkisebb [legnagyobb] paraméterértéket, ahol már [még] $n+1$ bázisfüggvény különbözik nullától ($n$ a bázisfüggvények fokszáma). Képletben:

$$\begin{aligned} min\_\,t() &= kv[n], \\ max\_\,t() &= kv[kv.size() - 1 - n]. \end{aligned}$$

```
uint K() const;
```

Visszaadja a bázisfüggvények számát:

$$K() = kv.size() - (n+1).$$

PRIVATE MEMBEREK:

```
double compute( uint i, uint n, double t ) const;
```

*i:* A bázisfüggvény indexe.
*n:* A bázisfüggvény fokszáma.
*t:* A bázisfüggvény paramétere.

Kiszámolja az *i. n.* fokú bázisfüggvény értékét a $t$ paraméterértékkel. A függvényhívás operátor használja.

```
double compute_derivative( uint i, uint n,
       double t, uint k ) const;
```

*i:* A bázisfüggvény indexe.
*n:* A bázisfüggvény fokszáma.
*t:* A bázisfüggvény paramétere.
*k:* Hanyadik derivált.

Kiszámolja az *i.* *n.* fokú bázisfüggvény *k.* deriváltját az adott *t* helyen. A *derivate* member használja.

```
void update_m( const KnotVector &kv );
```

*kv:* Az aktuális knot-vektor.

Frissíti a *kv* knot-vektornak megfelelően az *m* adat member által eltárolt multiplicitásokat. Mindig meg kell hívni, ha lecserélődik a *kv* védett adat memberben tárolt knot-vektor.

### BSplineSurface

Header file: `bspline.hh`

ADAT MEMBEREK:

```
Base *_ubase;
Base *_vbase;
```

Az *u* és *v* irányhoz tartozó bázisfüggvények. Ezeket az adat membereket az *uset_param* és *vset_param* függvényekkel kell beállítani.

```
Matrix<Point> *points2;
```

A kontrollpontokat tartalmazó mátrix. A *set_points* member állítja.

PUBLIC MEMBEREK:

```
ostream &operator << ( ostream &o, const BSplineSurface &b );
istream &operator >> ( istream &i, BSplineSurface &b );
```

*o:* Kimeneti *stream.*
*i:* Bemeneti *stream.*
*b:* A használt felület.

A paraméterként kapott *b* felületet írják ki, illetve olvassák be a *stream*-ről.

```
BSplineSurface();
BSplineSurface( const BSplineSurface &b );
virtual BSplineSurface();
```

*b:* A lemásolandó objektum.

Az első konstruktor egy olyan felületet hoz létre, amelynek a paramétereit használat előtt még be kell állítani. A második konstruktor önmagára másolja a kapott felületet. A destruktor felszabadítja az objektum által lefoglalt memóriaterületeket.

```
void uset_param( uint n, const KnotVector &kv );
void vset_param( uint n, const KnotVector &kv );
```

*n:* A bázisfüggvény fokszáma.
*kv:* A bázisfüggvényekhez tartozó knot-vektor.

Az $u$ és $v$ irányhoz tartozó bázisfüggvények és knot-vektorok beállítására szolgál. Mielőtt a felület pontjait lekérdezzük, ezeket a műveleteket meg kell hívni. Létrehozzák a megfelelő bázisfüggvény objektumokat az adott *kv* knot-vektorral.

```
void uset_param( uint n, uint k, double t0 = 0.0,
      double t1 = 1.0 );
void vset_param( uint n, uint k, double t0 = 0.0,
      double t1 = 1.0 );
```

Az $u$ és $v$ irányhoz tartozó bázisfüggvények és knot-vektorok beállítására szolgál. Mielőtt a felület pontjait lekérdezzük, ezeket a műveleteket meg kell hívni. Létrehozzák a megfelelő bázisfüggvény objektumokat a megadott $n$ fokszámmal és egy egyenletes eloszlású knot-vektorral, amelynek *t0* a legkisebb és *t1* a legnagyobb csomópontja.

```
void set_points( const Matrix<Point> &p2 );
```

*p2:* A kontrollpontokat tartalmazó mátrix.

A paraméterként kapott *p2* kontrollpont mátrixot lemásolja a *points2* védett adat memberbe. A felület használata előtt a kontrollpontok megadása kötelező.

```
Matrix<Point> &get_points() const;
```

Visszaadja a kontrollpontokat tartalmazó mátrixot.

```
Point operator () ( double u, double v ) const;
```

*u,v:* A paramétertér egy pontja.

Kiértékeli a felületet az adott $(u, v)$ paraméterpontban, és az eredményt visszaadja. A művelet meghívása előtt meg kell adni a kontrollpontokat, az $u$ és $v$ irányú knot-vektorokat és bázisfüggvény paramétereket. A számoláshoz használt képlet:

$$\sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij}\, N_i^{(un)}(u) N_j^{(vn)}(v).$$

```
Point derivate( double u, double v, uint uk, uint vk,
      bool &error ) const;
```

*u,v:* A paramétertér egy pontja.
*uk, vk:* Az $u$ és $v$ változók szerinti deriválások száma.

A függvény a következő kifejezés értékét számolja ki:

$$\frac{\partial^{uk+vk}}{\partial u^{uk}\partial v^{vk}} \sum_{i=0}^{n}\sum_{j=0}^{m} P_{ij}\ N_i^{(un)}(u) N_j^{(vn)}(v).$$

```
bool first_q( double u, double v, double &E, double &F,
      double &G ) const;
```

*u,v:* A paramétertér egy pontja.
*E, F, G:* Az első alapmennyiségek.

A member kiszámolja az első alapmennyiségeket, és azokat az $E$, $F$, $G$ változókban visszaadja. Ha hibatörtént a számítás során, akkor igazat ad vissza. A változók értéke a lefutás után:

$$E = \frac{\partial r(u,v)}{\partial u}^2,\ F = \frac{\partial r(u,v)}{\partial u}\frac{\partial r(u,v)}{\partial v},\ G = \frac{\partial r(u,v)}{\partial v}^2.$$

```
bool second_q( double u, double v, double &L, double &M,
      double &N ) const;
```

*u,v:* A paramétertér egy pontja.
*L, M, N:* A második alapmennyiségek.

A member kiszámolja a második alapmennyiségeket, és azokat az $L$, $M$, $N$ változókban visszaadja. Ha hibatörtént a számítás során, akkor igazat ad vissza. A változók értéke a lefutás után:

$$L = \left\langle \frac{\partial^2 r(u,v)}{\partial u^2},\ m \right\rangle,\ M = \left\langle \frac{\partial^2 r(u,v)}{\partial u \partial v},\ m \right\rangle,\ N = \left\langle \frac{\partial^2 r(u,v)}{\partial v^2},\ m \right\rangle,$$

ahol

$$m = \frac{\frac{\partial r(u,v)}{\partial u} \times \frac{\partial r(u,v)}{\partial v}}{\left\| \frac{\partial r(u,v)}{\partial u} \times \frac{\partial r(u,v)}{\partial v} \right\|},$$

a felület normálisa.

```
Point normal( double u, double v, bool &error ) const;
```

*u,v:* A paramétertér egy pontja.
*error:* Történt-e hiba a számítás során.

A member a felület normálisát számolja ki az $(u, v)$ paraméterekhez tartozó pontban. Ha nem tudja kiszámolni a deriváltakat, akkor az *error* változót igazra állítja. A visszaadott vektor a normális. A használt képlet:

$$m = \frac{\frac{\partial r(u,v)}{\partial u} \times \frac{\partial r(u,v)}{\partial v}}{\left\| \frac{\partial r(u,v)}{\partial u} \times \frac{\partial r(u,v)}{\partial v} \right\|}.$$

```
double curvature( double u, double v, double du,
       double dv, bool &error ) const;
```

*u,v:* A paramétertér egy pontja.
*du, dv:* Irány a paramétertérben.
*error:* Történt-e hiba a számítás során.

A member kiszámolja az $(u, v)$ paraméterértékekhez tartozó felületi pontban azt a normálgörbületet, amelyik a $(du, dv)$ irányhoz tartozik. A hibát az *error* változó jelzi. A normálgörbületeket az első és második alapmennyiségeket használva számítja ki az alábbi képlet segítségével:

$$\mathcal{K} = \frac{L\,du^2 + 2M\,du\,dv + N\,dv^2}{E\,du^2 + 2F\,du\,dv + G\,dv^2},$$

ahol $E$, $F$, $G$ az első-, $L$, $M$, $N$ pedig a második alapmennyiségek.

```
double gaussian( double u, double v, bool &error ) const;
```

*u,v:* A paramétertér egy pontja.
*error:* Történt-e hiba a számítás során.

Kiszámítja az $(u, v)$ paraméterekhez tartozó felületi pontban a Gauss szorzat-görbületet. A hibát az *error* változó jelzi. A képlet:

$$\mathcal{K} = \frac{LN - NM}{EG - F^2},$$

ahol $E$, $F$, $G$ az első-, $L$, $M$, $N$ pedig a második alapmennyiségek.

```
double minkowski( double u, double v, bool &error ) const;
```

*u,v:* A paramétertér egy pontja.
*error:* Történt-e hiba a számítás során.

Kiszámítja az $(u, v)$ paraméterekhez tartozó felületi pontban a Minkowski összeg-görbületet. A hibát az *error* változó jelzi. A képlet:

$$\mathcal{H} = \frac{EN - 2FM + GL}{EG - F^2},$$

ahol $E$, $F$, $G$ az első-, $L$, $M$, $N$ pedig a második alapmennyiségek.

```
const Base &ubase() const;
const Base &vbase() const;
```

Az $u$ és $v$ irányhoz tartozó bázisfüggvényeket adják vissza.

```
double umin_t() const;
double umax_t() const;
double vmin_t() const;
double vmax_t() const;
```

Visszaadják az $u$ és $v$ irányhoz tartozó knot-vektorokban szereplő legkisebb és legnagyobb csomópont értékét.

```
uint uK() const;
uint vK() const;
```

Az $u$ és $v$ irányhoz tartozó kontrollpontok számát adják meg.

# 6. fejezet

# Felhasználói dokumentáció

Ez a fejezet azon felhasználóknak szól, akik még nem találkoztak a programunkkal. Nekik próbálunk néhány tanácsot adni, hogy könnyebben kezelhessék a programot.

## 6.1. Telepítés

Miután az Implementációs részben leírt környezetet feltelepítettük a gépre, a program telepítése rendkívül egyszerű. Másoljuk egy tetszőleges alkönyvtárba a forráskódot, majd a *make all* parancsot kiadva fordítsuk le. Így létrejönnek az adott gépen futtatható állományok. Ha a fordítás nem sikeres, annak az lehet az oka, hogy az adott gépen nem ugyanabban az alkönyvtárban találhatók a fordításhoz szükséges környezeti állományok, mint a mi gépünkön. Ezt a hibát kijavíthatjuk a *Makefile* első sorainak szerkesztésével, amelyek pont az ehhez szükséges információkat tartalmazzák. A forráskódhoz tartozik két állomány a *vtkInclude.tcl* és a *vtkInt.tcl*. Telepítéskor be kell állítanunk a $VTK\_TCL$ környezeti változót úgy, hogy annak a könyvtárnak a nevét tartalmazza, amelyben ezek az állományok találhatók.

## 6.2. Futtatás

A fordításkor létrejött futtatható állományok a következők:

*spl2real:* egy már meglévő B-Spline felületet mér le,
*real2spl:* mért adatokat közelít egy B-Spline felülettel,
*spl2vtk:* egy B-Spline felületet alakít a *vtk* számára megjeleníthető formátumra,
*cmpspl:* két B-Spline felület kontrollpontjait hasonlítja össze.

Van még egy futtatható állomány, *showpoly*. Ezt azonban nem kell lefordítani mivel ez egy $TCL$ forrás kód, amit az interpreter értelmezni tud. Ez fogja az előállított felületet megjeleníteni.

### 6.2.1. spl2real

Ez a program már meglévő B-Spline felületet mér le és mért adatokat készít belőle. A valóságban mikor egy felületet lemérnek nem kapnak pontos értékeket, hiszen a mérőeszközök bizonyos hiba eltéréssel képesek csak mérni. A program első paramétere arra szolgál, hogy ezt a mérésből származó pontatlanságot szimulálni tudjuk. Ha nem adunk meg hiba paramétert, akkor az alapértelmezés szerinti értékkel számol a program. Használata:

```
spl2real [-e <number>] <file>
```

*-e num:* a mérési hiba maximumát határozza meg. Alapérték 1e-3.
*<file>:* annak a file-nak a neve, amelybe a készített B-Spline felületet el szeretnénk tárolni.

### 6.2.2. real2spl

Ezzel a programmal kőzelíthetjük a mért adatokat B-Spline felülettel. A paraméterek segítségével beállíthatjuk, hogy milyen legyen a közelítő felület, van-e olyan rész, amit nem kívánunk megjeleníteni, alkalmazzunk-e paraméter korrekciót és ha igen, akkor milyen módszer szerint, simítani akarjuk-e a felületet és ha igen, akkor milyen módszer szerint.

```
real2spl [-U num] [-V num] [-u num] [-v num] [-n num] [-m num]
      [-r num] [-e excl_file] [-1|-2|-3|-4|-5|-6|-7|-8] <file>
```

*-U num:* Kontrollpontok száma $u$ irányban. Minél több kontrollpontot adunk meg, annál jobban meg fogja közelíteni a generált felület a mért pontokat. Ez nem feltétlenül jó, mert túl nagyra állítva már a hibák is megjelenhetnek a közelítő felületen. Alapérték: 10.
*-V num:* Kontrollpontok száma $v$ irányban. Alapérték: 10.
*-u num:* A *vtk* háromszögeket tartalmazó sávok segítségével jeleníti meg a felületet. *num* az $u$ irányban megjelenített háromszögek száma. Ha túl alacsonyra állítjuk, akkor nem tudja pontosan követni a felületet a megjelenítés. Ha viszont túl nagyra állítjuk, akkor nagyon sokáig fog tartani a felület generálása. Alapérték: 40.
*-v num:* $v$ irányban megjelenített háromszögek száma. Alapérték: 40.
*-n num:* A felületet $u$ irányban leíró B-Spline bázis függvények fokszáma. Egy nagyon bonyolult felületről származó adatokat általában magasabb fokszámú bázis függvényű Spline felülettel jobban tudunk közelíteni. Alapérték: 3.
*-m num:* A felületet $v$ irányban leíró B-Spline bázis függvények fokszáma. Alapérték: 3.

*-e excl_file:* A mérési pontok vágásához szükséges file neve. A file olyan poligonok leírását tartalmazza, amelyek belső pontjait nem kell megjeleníteni, így a közelítésnél sem játszanak szerepet.

*-r number:* A megengedhető legnagyobb átlagos hibanégyzet.

*-1:* Téglalap alakú mátrixot használjon a program. Ez az alapérték.

*-2:* Négyzetes mátrixot használjon a program.

*-3:* Alkalmazzon paraméter korrekciót az 1. eljárás felhasználásával.

*-4:* Alkalmazzon paraméter korrekciót a 2. eljárás felhasználásával.

*-5:* Alkalmazzon paraméter korrekciót a 3. eljárás felhasználásával.

*-6:* Használja a felületek simítására az 1. funkcionált.

*-7:* Használja a felületek simítására a 2. funkcionált.

*-8:* Használja a felületek simítására a 3. funkcionált.

*<file>:* Annak a file-nak a neve, amelyben a mért adatok vannak.

### 6.2.3. spl2vtk

Ezzel a programmal tudjuk az előállított felületet olyan formájúvá alakítani, amit a *vtk* már képes megjeleníteni.

```
spl2vtk [-e <excl>] <file>
```

*-e <excl>:* Annak a file-nak a neve, amely olyan poligonok leírását tartalmazza, amelyek belső pontjait nem kell megjeleníteni.

*<file>:* A megjelenítendő felületet tartalmazó file neve.

### 6.2.4. cmpspl

Ez a program két B-Spline felülethez tartozó kontrollpoligont hasonlít össze. Eredményül azt adja meg, hogy az összetartozó kontrollpontok között mért távolságnégyzeteknek mennyi az átlaga. Ebből adódóan, ha mindkét paraméterében ugyanazt a felületet kapja bemenő adatként, a visszaadott érték 0. Csak olyan B-Spline felületek adhatók meg paraméterként, amelyek ugyanannyi kontrollpontból állnak.

```
cmpspl <spline1> <spline2>
```

*<spline*i*>:* Az összehasonlítandó felületet tartalmazó file-ok nevei.

### 6.2.5. showpoly

Ez a program fogja az előállított felületet megjeleníteni. Futtatásakor egy új ablakban jelenik meg a paraméterként megadott felület. Lehetőségünk van

különböző nézőpontból vizsgálni a felületet. Ezt úgy tehetjük meg, hogy az ablak egy pontján lenyomjuk az egér bal gombját, aminek hatására a felület abba az irányba elfordul. A nézőpont és a felület távolságát is állíthatjuk. Ehhez az egér jobb gombját kell valahol az ablakon lenyomnunk. Ha az egérkurzor az ablak felső felében volt, akkor közeledni fogunk a felülethez. Ennek megfelelően ha az ablak alsó felén nyomtuk le a jobb gombot, a felület távolodni fog. Minél közelebb volt a kurzor az ablak közepéhez, annál lassabban fog közeledni vagy távolodni a felület. A megjelenítő ablak reagál bizonyos billentyű lenyomásokra is:

*w:* A megjelenített felületet egy háromszög háló reprezentálja. Ezeknek a háromszögeknek a számát állíthattuk be a *real2spl* program segítségével. Ebben a megjelenítési módban könnyebben forgathatjuk, nagyíthatjuk a felületet, mivel a megjelenített pontok száma lényegesen kevesebb.

*s:* Ebben a megjelenítési módban a háromszögeknek nem csak a kerülete, de a belseje is meg van jelenítve.

*u:* Lenyomásakor megjelenik az úgynevezett *vtk Interactor*, amelynek segítségével különböző parancsokat adhatunk a programnak. A parancsok segítségével megjeleníthetünk illetve elrejthetünk különböző, a felülethez tartozó segéd elemeket vagy megváltoztathatjuk a már megjelenített elemek jellemzőit. Segéd elem lehet:

*p_ actor:* Kontrollpoligon háló.

*cont_ actor:* Kontrollpontok.

*outline_ actor:* Annak a legkisebb téglatestnek az élei, amely még tartalmazza a felületet és a segéd elemeket.

*cone_ actor:* Kis kúpokat jelent, amelyek a paramétervonalak metszéspontjain helyezkednek el és csúcsuk a felület adott pontjába húzott normális irányába mutatnak.

A következő parancsok megengedettek:

*show* par*:* A paraméterként megadott segédelemet jeleníti meg.

*hide* par*:* A paraméterként megadott segédelemet rejti el.

*show_ all:* Az összes segédelemet megjeleníti az ablakban.

*hide_ all:* Az összes segédelemet elrejti.

*lfile* par*:* A paraméterként megadott file-ban megadott felületet jeleníti meg. A felület színei a Minkowski görbületet jelzik.

*gfile* par*:* A paraméterként megadott file-ban megadott felületet jeleníti meg, de itt a felület színei a Gauss görbületet jelzik.

*e:* Ennek segítségével zárhatjuk le az ablakot és fejezhetjük be a program futását.

`showpoly <file>`

<*file*>*:* A megjelenítendő felületet tartalmazó file neve.

# Táblázatok jegyzéke

# Ábrák jegyzéke

# Irodalomjegyzék


[1] Szőkefalvi-Nagy Gyula, Gehér László, Nagy Péter, *Differenciálgeometria.* Műszaki Könyvkiadó, 1979.

[2] Karvasz Gyula, *Analízis III.* Tankönyvkiadó, 1989.

[3] Biplab Sarkar, Chia-Hsiang Menq, *Parameter optimization in approximating curves and surfaces to measurment data.* Computer Aided Geometric Design, 8(1991):267-290.

[4] Josef Hoschek, Dieter Lasser, *Computer Aided Geometric Design.* B.G. Teubner, Stuttgart, 1989.

[5] Josef Hoschek, Ulrich Dietz, *Smooth B-Spline surface approximation to scattered data.* Reverse Engineering, Editors J. Hoschek, W. Dankwort. B.G. Teubner, Stuttgart, 1996.

[6] Kátai Imre, *Numerikus analízis.* Tankönyvkiadó, 1989.

[7] Stoyan Gisbert, Takó Galina, *Numerikus módszerek I.* ELTE-TypoTEX, 1993.

[8] Wettl Ferenc, Mayer Gyula, Sudár Csaba, *LaTeXkezdőknek és haladóknak.* Panem Kft., 1998.