TDK-dolgozat

Horváth Anna Lili

Geometriai távolságmezők

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

ALGORITMUSOK ÉS ALKALMAZÁSAIK TANSZÉK

Szerző:

Horváth Anna Lili

programtervező informatikus BSc

3. évfolyam

Témavezetők:

Bán Róbert

doktorandusz

dr. Valasek Gábor

egyetemi adjunktus

Budapest, 2023

Tartalomjegyzék

1. Bevezetés **3**

1.1. Távolságmezők 3

1.2. Geometriai távolságmezők 4

1.3. Térbeli geometriai távolságmezők 4

1.4. Mezők szűrése 5

2. Térbeli geometriai távolságmezők **6**

2.1. Távolságfüggvények 6

2.2. Távolságmezők reprezentációja 7

2.3. Mező generálása 7

2.3.1. Mintavételezési pozíció 7

2.3.2. Közelítő geometria meghatározása 8

2.3.3. Mező rendje 8

2.4. Talppont keresése 9

2.5. Helyettesítő geometria első rendű mezőre 10

2.6. Helyettesítő geometria másodrendű mezőre 11

2.6.1. Tórusz reprezentációja 14

2.6.2. Talppontos reprezentáció 17

2.6.3. Görbületek illesztése 22

2.6.4. Algoritmus 31

2.7. Geometriák kirajzolása 32

2.7.1. Mező megjelenítése 32

2.7.2. Másodrendű mező 34

2.8. Mező szűrése általános módszerekkel 40

2.8.1. Trilineáris interpoláció 40

2.9. Tesztelés és mérések 41

2.9.1. Pontosság mérése 42

2.10. Eredmények 43

3. CSG szűrés **51**

3.1. Síkbeli mezők generálása 51

3.1.1. Mintavételezési pozíció 51

3.1.2. A mező rendje 52

3.2. Helyettesítő geometria 52

3.3. Síkbeli elsőrendű mező megjelenítése 54

3.3.1. Bilineáris interpoláció 55

3.4. CSG szűrés 57

3.5. Eredmények 64

4. Összegzés **68**

Irodalomjegyzék **69**

1. fejezet

Bevezetés

A számítógépes grafikában a felületek reprezentációjára számos módszert szoktak alkalmazni. Ezek közül a legjobb implicit megoldás az előjeles távolságfüggvény (SDF) használata. Az SDF értéke a tér minden pontjában a felülettől vett távolság, kívül pozitív belül pedig negatív előjellel. Az SDF értékeket felhasználva sphere tracing algoritmussal megjeleníthető a geometria.

1.1. Távolságmezők

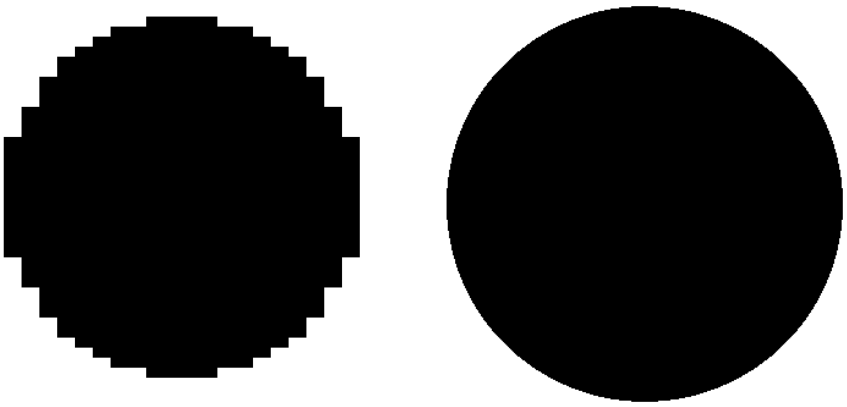
Az alakzatok többsége nem írható le egy implicit képlettel, sokszor akár több felületdarabbal kell közelíteni őket. Ilyen esetekben a távolságfüggvények nagyon bonyolultak lehetnek, a számításigényük megnőhet vagy akár fel sem írhatóak zárt alakban. Erre a problémára kínálnak megoldást a távolságmezők az SDF diszkrét tárolásával.

A távolságmező egy 2 vagy 3 dimenziós rács, amelynek minden cellájában az eredeti távolságfüggvény valamilyen közelítése szerepel. A megjelenítés során minden sík- vagy térbeli pontban a megfelelő cellában található approximációt használjuk az eredeti, bonyolult függvény helyett. A távolságmezőhöz tartozik a tárolt adat mellett egy szűrési módszer is, amellyel visszanyerhetjük az SDF közelítését.

A szűrés egy interpolációs eljárást jelent, amely a mintapontokban rekonsturálja az eredeti távolságfüggvényt, közöttük pedig folytonos eredményt ad. A cellán belül ezek általában végtelenszer folytonosan differenciálhatóak, de a cellahatároknál kisebb, lapoknál például C^0.

A távolságmező egyik legegyszerűbb formája a nulladrendű mező. Itt a mező minden cellájában a cella középpontjában vett SDF értékét tároljuk. Ezekből a skalárokból például

trilineáris szűréssel kapható meg a közelítés.



(a) Szűrés nélkül (b) Bilineáris szűréssel

1.1. ábra. Példa a kör közelítésére nulladrendű mezővel, 100-as felbontáson

1.2. Geometriai távolságmezők

Szakdolgozatomban a geometriai távolságmezőkkel foglalkoztam. A felület egyes részeit valamilyen geometriai objektummal helyettesítettem, aminek a távolságfüggvénye könnyen kiszámítható és kellően közelíti az eredeti felületét.

Ehhez először is minden cellában meg kell keresni a felület legközelebbi pontját (talppontot), majd megtalálni azt a geometriát, ami a talppont körül adott rendben a lehető legjobban közelíti a felületet. Az elsőrendű közelítésnek a normálvektort, a másodrendűnek a normálvektort és a talppontban vett görbületet kell tárolnia. Jó geometria például elsőrendű mezőre a sík, másodrendűre pedig a paraboloid vagy a tórusz.

Ezekre 2 dimezióban született megoldás [1]. Itt az elsőrendű mező a talppontban vett érintőegyenest, a másodrendű pedig a simulókört használja. A végső approximáció a helyettesítő geometriától vett távolságok bilineáris interpolációjával kapható meg.

1.3. Térbeli geometriai távolságmezők

Dolgozatomban a 3 dimenziós geometriai mezők generálására és megjelenítésére dolgoztam ki módszert első- és másodrendben. Az elsőrendű mezőnek a talppontot és a normálvektort kellett tárolnia, a másodrendűnek a normálvektort, főgörbületeket és a főgörbületi irányokat.

Ehhez megfelelő geometriák a sík és a tórusz. Implementáltam a geometriák illesztését, valamint szűrését trilineáris interpolációval.

1.4. Mezők szűrése

A bilineáris és trilineáris szűrés nem használja ki hogy a mező geometriákat tárol. Közvetlen interpoláció helyett a 2 dimenziós elsőrendű mezőhöz kidolgoztam egy módszert, amely a geometriák metszetét és unióját használva ad egy jobb közelítést.

2. fejezet

Térbeli geometriai távolságmezők

Dolgozatom első felében bemutatom a térbeli geometriai távolságmezők generálására kidolgozott módszeremet, valamint a mezők kiértékelését és egy egyszerű szűrési módszert.

2.1. Távolságfüggvények

A távolságmezők egy előjeles távolságfüggvény approximációját adják meg.

1. Definíció. Az $f : \mathbb{R}^N \to \mathbb{R}$ függvény távolságfüggvény, ha

$$f(\boldsymbol{p}) = d(\boldsymbol{p}, f \equiv 0), \quad (\forall \boldsymbol{p} \in \mathbb{R}^N)$$

2. Definíció. Az $f : \mathbb{R}^N \to \mathbb{R}$ előjeles távolságfüggvény, ha folytonos és $|f|$ távolságfüggvény.

3. Definíció. X topologikus tér $S \subseteq X$ részhalmazának ∂S határán azokat a pontokat értjük, amelyek elemei S lezártjának, de nem belső pontjai S-nek.

$$\partial S = \overline{S} \backslash int_X S$$

1. Tétel. *Az f pontosan akkor SDF, ha létezik $\emptyset \neq D \subseteq \mathbb{R}^N$ halmaz amivel*

$$f(\boldsymbol{p}) = \begin{cases} d(\boldsymbol{p}, \partial D) & ha\ \boldsymbol{p} \notin D \\ -d(\boldsymbol{p}, \partial D) & ha\ \boldsymbol{p} \in D \end{cases}$$

2.2. Távolságmezők reprezentációja

A távolságmezőket 3 dimenziós tömbökkel reprezentálom, amelyeknek minden cellájában egy közelítő geometriát tárolok. A távolságmezőt a következő, dimenziótól és rendtől független adatok határozzák meg:

- a felbontása ($res \in \mathbb{N}$, hány cellából álljon a tömb egy dimenzió mentén)
- a befoglaló doboz oldalhossza ($box \in \mathbb{R}$, a közelítendő objektum egy origó középpontú négyzetben/ kockában helyezkedik el, ezen belül generálok mezőt)

Feltehetjük, hogy minden objektum a $[-1,1]^3$ kockának helyezkedik el, ezért a *box* értéket alapértelmezetten 2-nek veszem, az ábrák és mérések ezzel az értékkel készültek.

A távolságmezőket a könnyű GPU-s elérés érdekében 4 csatornás RGBA textúrában tárolom, így a geometriákat valós számokkal kell reprezentálni.

A mező generálása során ezeket a textúrákat compute shaderekkel töltöm fel, majd átmásolom a GPU-ra, ahol kiolvasom az adatot és kiszámítom belőle az approximációt. Ehhez egyes esetekben nem elég a tárolt geometriát kiértékelni, arra még bizonyos megszorítások kellhetnek a geometria állásától függően (például csak egy metszetét kell venni), ezért a távolságmező önmagában még nem ad jó közelítést. A pontossabb eredményhez szűrési módszerre is szükség lehet.

A következőkben a mező generálását mutatom be, majd a tárolt adat értelmezését, végül a szűrési módszereket.

2.3. Mező generálása

A mező generálása dimenziótól függetlenül hasonlóan történik. Először meghatározom a térbeli pozíciót, ahonnan mintavételezni és közelíteni akarom az SDF-et. Ez a fent leírt felbontástól ($res \in \mathbb{N}$) és a befoglaló doboz oldalhosszától ($box \in \mathbb{R}$) függ.

2.3.1. Mintavételezési pozíció

A mintavételezési pozíció a *box* oldalhosszú origo középpontú, $res \times res \times res$ felbontású kocka (i,j,k) indexű cellájának középpontja, azaz $\forall i \in \{0 \dots res-1\} : \forall j \in \{0 \dots res-1\} : \forall k \in \{0 \dots res-1\} :$

$$\boldsymbol{grid}(i,j,k) = -\frac{1}{2}\begin{bmatrix} box \\ box \\ box \end{bmatrix} + \begin{bmatrix} i+0.5 \\ j+0.5 \\ k+0.5 \end{bmatrix} \cdot \frac{box}{res} \tag{2.1}$$

2.3.2. Közelítő geometria meghatározása

Minden cellában egy olyan geometriát szeretnénk tárolni, amelynek SDF-je a mintavételezési pont egy környezetében közelíti az eredeti SDF-et.

Ehhez először megkeresem a görbe vagy felület $\boldsymbol{grid}(i,j,k)$ ponthoz legközelebbi pontját, ez lesz a talppont, majd illesztek egy geometriát, ami adott rendben megfelelően közelíti az objektumot a talppont kis környezetében.

A geometria paremétereit alkalmas reprezentációban eltárolom a mezőben. Fontos, hogy az adott paramtéterekből a geometria távolságfüggvénye könnyen számolható legyen, a geometriákat és a reprezentációt is ennek megfelelően választom.

2.3.3. Mező rendje

A mezőtől azt várjuk, hogy az eredeti felület lokális differenciálgeometriáját adott rendben rekonstruálni tudjuk belőle a mintavételezési pontokhoz tartozó talppontokban. A mező rendje határozza meg, hogy pontosan mik azok az adatok amiket meg szeretnénk kapni.

Az elsőrendű mezőnek a talppontot és a talppontban vett normálvektort kell visszaadnia, a másodrendű mezőnek pedig ezeken kívül a görbületet is tárolni kell.

4. Definíció. Tekintsük az $\boldsymbol{x}$ természetes paraméterezés második deriváltját és legyen $\kappa = |\boldsymbol{x}''|$ a görbe adott pontbeli előjel nélküli görbülete, ami értelmezhető úgy is, mint $\boldsymbol{x}''$ vektornak a főnormálisra vett előjeles merőleges vetületének hossza, azaz $\kappa = |\boldsymbol{x}'' \cdot \boldsymbol{n}|$.

3D-ben egy görbület helyett a két főgörbületet és a a főgörbületi irányokat kell ismerni (ezek definícióját lásd a 2.6-os fejezetben). Mivel a főgörbületi irányok nem umbilikus pontokban biztosan merőlegesek egymásra elég csak az egyik a kettő közül, azt hogy melyiket, a proxy geometriának megfelelően fogom megválasztani.

Megjegyzés. A mintavételezési ponton kívüli pontokban nem elvárás, hogy a differenciálgeometriát pontosan visszakapjuk, csak minél jobb közelítéssel.

2.4. Talppont keresése

A talppont keresésére gradiens módszert használok, amely bármely dimenzióban működik.

5. Definíció. Az $f : \mathbb{R}^N \to \mathbb{R}$ függvény gradiense:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Az $f : \mathbb{R}^N \to \mathbb{R}$ függvény esetén a gradiens azt az irányt adja meg, amerre elmozdulva a függvényérték a legtöbbet változik. Az SDF esetén $\nabla f \in \mathbb{R}^N$ vektor párhuzamos a felület talppontban vett normálvektorával, hiszen egy adott felületi pontból indulva a merőleges irányába elmozdulva nő a legtöbbet a felülettől vett távolság.

Tehát ∇f megadja a talppontban vett normálvektor irányát, $f(p), \boldsymbol{p} \in \mathbb{R}^N$ pedig a talpponttól vett távolságot. Így az $\boldsymbol{p} \in \mathbb{R}^N$ pozícióhoz tartozó talppont:

$$\boldsymbol{fp}(\boldsymbol{p}) = \boldsymbol{p} - f(\boldsymbol{p}) \cdot \frac{\nabla f}{\|\nabla f\|} \tag{2.2}$$

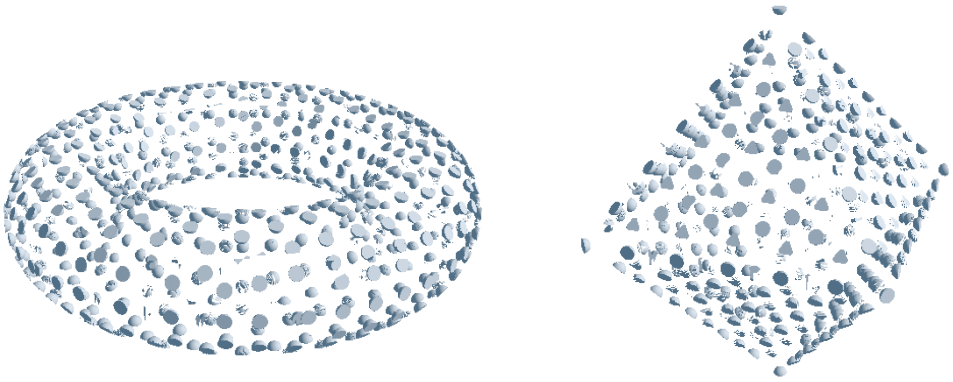
A gradiens egyszerű számolásához centrális differenciákat használok:

$$\partial_x f(\boldsymbol{p}) = \frac{f(\boldsymbol{p}+\varepsilon) - f(\boldsymbol{x}-\varepsilon)}{2\varepsilon}$$

ahol $\varepsilon \in \mathbb{R}$ egy kellően kicsi pozitív szám.

A centrális differencia csak közelíti a deriváltat, ezért lehet, hogy a deriváltvektor nem lesz párhuzamos a nomálvektorral. Ekkor a (2.2) eredményeként kapott pont nem biztos, hogy a felületre fog esni. Az ilyen pontatlanságok kiküszöbölése érdekében a módszert néhányszor megitárlom a talált talppontból is. Minden iterációs lépésben veszem az aktuális pont gradiensét és az alapján lépek a felület irányába. Az iteráció véget ért, ha a pont eléri a felületet.

Minden $\boldsymbol{grid}(i, j, k)$ pozícióhoz külön megkeresve a talppontot megkapjuk a felületi pontokat (2.1 ábra), amelyeknek a környezetében valamilyen geometriával szeretnénk helyettesíteni a felület.



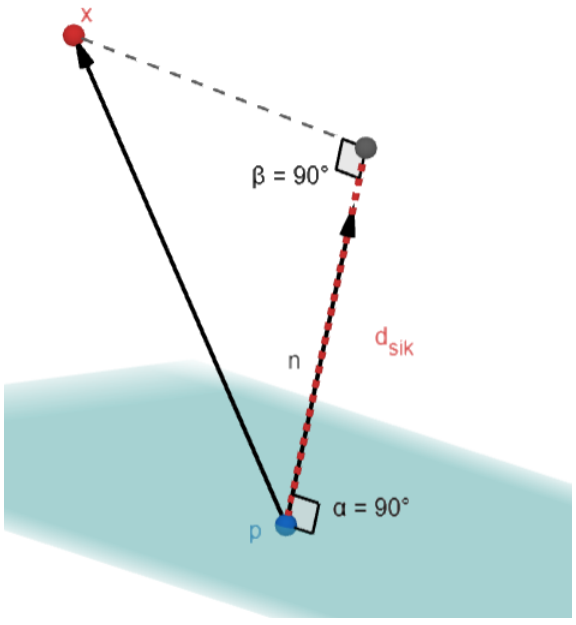
2.1. ábra. Talppontok a tóruszról és az oktaéderről

2.5. Helyettesítő geometria első rendű mezőre

A talppont helye a mező rendjétől és dimenziójától független volt, de a helyettesítő geometria nem az. Minden mezőhöz olyan alakzatot kell választani, amely teljesíti a fent leírt elvárásokat és visszadja a megfelelő adatokat a mintavételezési pozíciókban.

A 3D-s elsőrendű mezőnek a talppontot és a normálvektort kell megadnia. Ehhez a legegyszerűbb geometria az érintősík.

Egy sík megadható egy $\boldsymbol{p}$ pontjával és az ott vett $\boldsymbol{n}$ normálvektorral. A $\boldsymbol{p}$-ből $\boldsymbol{x} \in \mathbb{R}^3$-be mutató vektort normálvektorra vetítve a 2.2 ábrán látható módon megkapjuk a pont távolságát a síktól.



2.2. ábra. A síktól vett távolság számítása

Az $\boldsymbol{x}$ távolsága a síktól:

$$d_{sik} = \langle \boldsymbol{x} - \boldsymbol{p}, \boldsymbol{n} \rangle \tag{2.3}$$

2.6. Helyettesítő geometria másodrendű mezőre

A 3 dimenziós másodrendű mezőnek a talppontot, a normálvektort, a főgörbületeket és a főirányokat kell rekonstuálnia.

Főgörbületek és főirányok

A továbbiakban tegyük fel, hogy $\boldsymbol{x}(u,v)$ egy reguláris parametrikus felület.

Legyen adott egy $u : \mathbb{R} \to \mathbb{R}^2$ reguláris görbe $\boldsymbol{x}(u,v)$ értelmezési tartományában

Ekkor $\boldsymbol{c}(t) = \boldsymbol{x}(\boldsymbol{u}(t)) : \mathbb{R} \to \mathbb{R}^3$ egy reguláris térgörbe lesz, ami az $\boldsymbol{x}(u,v)$ felületen fut.

6. Definíció. Az $\boldsymbol{x}(u,v)$ felület paraméterezése reguláris, ha az u és v szerinti parciális deriváltvektorok linerárisan függetlenek.

Ekkor a felület minden pontjában definiált az

$$\boldsymbol{m}(u,v) = \frac{\boldsymbol{x}_u(u,v) \times \boldsymbol{x}_v(u,v)}{|\boldsymbol{x}_u(u,v) \times \boldsymbol{x}_v(u,v)|}$$

felületi normális irány.

7. Definíció. A felület első főmennyiségei

$$\mathbb{E} = \boldsymbol{x}_u \cdot \boldsymbol{x}_u \quad \mathbb{F} = \boldsymbol{x}_u \cdot \boldsymbol{x}_v \quad \mathbb{G} = \boldsymbol{x}_v \cdot \boldsymbol{x}_v$$

8. Definíció. A felületi görbe íveleme pedig

$$ds^2 = \mathbb{E} \cdot u'^2 + 2\mathbb{F} \cdot u' \cdot v' + \mathbb{G} \cdot v'^2$$

9. Definíció. Legyen a

$$\boldsymbol{c}(s) = \boldsymbol{x}(\boldsymbol{u}(s))$$

felületi görbe ívhossz szerint paraméterezve és legyen a Frenet frame-je (kísérő triédere) $(\boldsymbol{t}(s), \boldsymbol{n}(s), \boldsymbol{b}(s))$

A Frenet-Serret formulák szerint:

$$\kappa \boldsymbol{n} = \boldsymbol{c}''$$

Mivel $\boldsymbol{c}(s)$ felületi görbe is, ezért:

$$\kappa \boldsymbol{n} = \boldsymbol{x}_{uu} \cdot (u')^2 + 2x_{uv} \cdot u' \cdot v' + \boldsymbol{x}_{vv} \cdot (v')^2 + x_u \cdot u'' + x_v \cdot v''$$

Ezt skalárisan szorozva $\boldsymbol{m}$-el:

$$\kappa \boldsymbol{n} \cdot \boldsymbol{m} = \boldsymbol{x}_{uu} \cdot \boldsymbol{m} \cdot (u')^2 + 2x_{uv} \cdot \boldsymbol{m} \cdot u' \cdot v' + \boldsymbol{x}_{vv} \cdot \boldsymbol{m} \cdot (v')^2$$

10. Definíció. A Gauss-féle második főmennyiségek

$$\mathbb{L} = \boldsymbol{x}_{uu} \cdot \boldsymbol{m}, \quad \mathbb{M} = \boldsymbol{x}_{uv} \cdot \boldsymbol{m}, \quad \mathbb{N} = \boldsymbol{x}_{vv} \cdot \boldsymbol{m} \ .$$

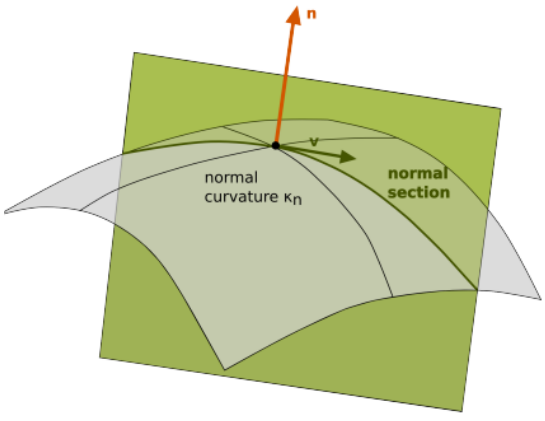
Jelölje ω a felületi normális és a felületi görbe főnormálisa által bezárt szöget. Ekkor a felület második alapformája:

$$\kappa \cdot cos\phi \cdot ds^2 = \mathbb{L} \cdot u'^2 + 2\mathbb{M} \cdot u' \cdot v' + \mathbb{N} \cdot v'^2$$

Leosztva ds^2-el

$$\kappa \cdot cos\phi = \frac{\mathbb{L} \cdot u'^2 + 2\mathbb{M} \cdot u' \cdot v' + \mathbb{N} \cdot v'^2}{\mathbb{E} \cdot u'^2 + 2\mathbb{F} \cdot u' \cdot v' + \mathbb{G} \cdot v'^2}$$

Adott pontban ez csak ϕ-től függ, így ha ezt rögzítsük 0-ra, akkor olyan görbét kapunk amelynek az adott pontbeli simulósíkjai tartalmazzák a felületi normálist. Ezek a normálmetszet görbék, görbületük a normálgörbület (2.3. ábra).



2.3. ábra. A normálmetszet és a normálgörbület. Forrás: [2]

11. Definíció. A normálgörbület:

$$\kappa_0 = \kappa_0(\boldsymbol{t}) = \frac{\mathbb{L}\cdot u'^2 + 2\mathbb{M}\cdot u'\cdot v' + \mathbb{N}\cdot v'^2}{\mathbb{E}\cdot u'^2 + 2\mathbb{F}\cdot u'\cdot v' + \mathbb{G}\cdot v'^2}$$

Legyen $\lambda = \begin{cases} \frac{u'}{v'} & \text{ha } v' > u' \\ \frac{v'}{u'} & \text{ha } u' > v' \end{cases}$ a normálmetszet görbe irányvektorának tangense/kotangense. Ekkor a normálgörbület:

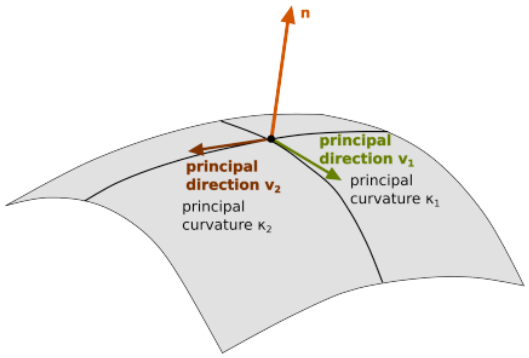
$$\kappa_0(\lambda) = \frac{\mathbb{L} + 2\mathbb{M}\cdot\lambda + \mathbb{N}\cdot\lambda^2}{\mathbb{E} + 2\mathbb{F}\cdot\lambda + \mathbb{G}\cdot\lambda^2}$$

12. Definíció. A normálgörbület szélsőértékeit a következő egyenlet λ_1, λ_2 megoldásainál veszi fel:

$$\begin{vmatrix} \lambda^2 & -\lambda & 1 \\ \mathbb{E} & \mathbb{F} & \mathbb{G} \\ \mathbb{L} & \mathbb{M} & \mathbb{N} \end{vmatrix} = 0$$

13. Definíció. A fenti λ_1, λ_2 tangensekhez tartozó irányok a görbületi főirányok. $\kappa_1 = \kappa(\lambda_1), \kappa_2 = \kappa(\lambda_2)$ a normálgörbületeket pedig főgörbületeknek nevezzük. (2.4. ábra)

Megjegyzés. A továbbiakban az általánosság megszorítása nélkül tegyük fel, hogy $|\kappa_1| \leq |\kappa_2|$



2.4. ábra. Főgörbületek és főgörbületi irányok. Forrás: [2]

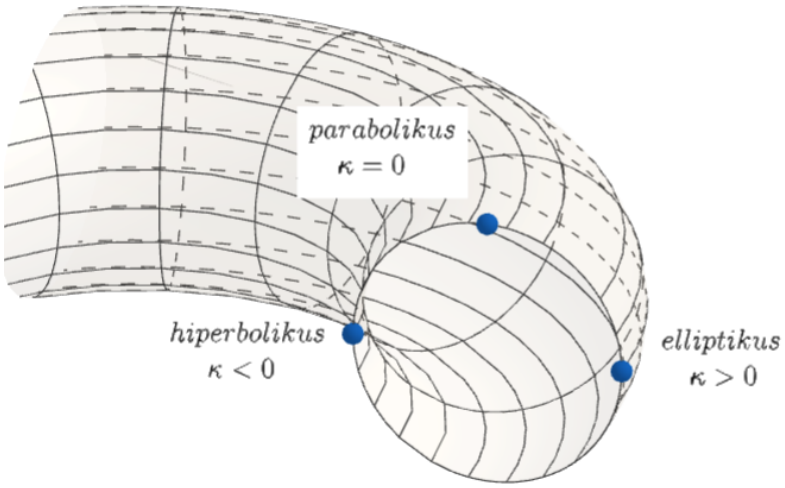
2. Tétel. *A főirányok merőlegesek egymásra.*

3. Tétel. *Egy reguláris felület $\boldsymbol{x}(u,v)$ pontjában rögzített $\boldsymbol{t}$ tangensirányt tartalmazó összes lehetséges simulósíkú görbe esetén a görbe simulókörének sugara $\frac{1}{\kappa}$.*

A helyettesítő geometria

A másodrendű mezőhöz a legalkalmasabb geometria a tórusz, mivel minden említett adatot képes tárolni valamint ismert a távolságfüggvénye zárt alakban, így a megjelenítésnél könnyen számolható.

A főgörbületek lehetnek pozitívak, amikor a normálvektor felé görbül a felület, negatívak amikor a normálvektortól elfele, és 0 amikor sík a felület. A tórusz a főgörbületek bármilyen előjel szerinti kombinációját képes reprezentálni. A tórusz különböző pontjaiban fennálló görbületi viszonyok a 2.5. ábrán láthatóak.

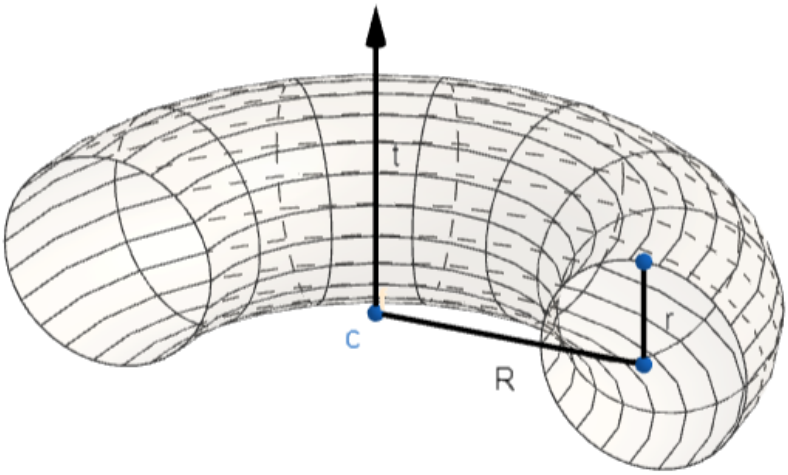


2.5. ábra. A tórusz pontjainak osztályozása görbület szerint

- parabolikus pontok $\kappa_1 = 0 \ \wedge \ \kappa_2 \neq 0$
- hiperbolikus pontok $\kappa_1 < 0 \ \wedge \ \kappa_2 > 0$
- elliptikus pontok $\kappa_1 > 0 \ \wedge \ \kappa_2 > 0$

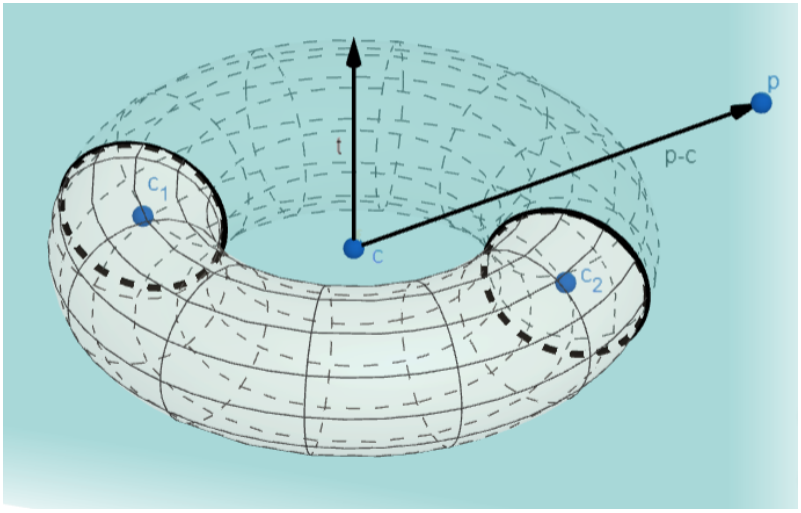
2.6.1. Tórusz reprezentációja

Az általános állású tóruszt definiáló adatok a középpontja ($\boldsymbol{c}$), a tengelye ($\boldsymbol{t}$) és a két sugara (r, R). A tóruszt megadó paraméterek a 2.6. ábráról leolvashatók.



2.6. ábra. Az általános állású tórusz paraméterei

A $\boldsymbol{p} \in \mathbb{R}^3$ pont tórusztól vett távolságának kiszámítása visszavezethető két körtől vett távolságra a síkon. Ehhez először a tóruszt el kell metszeni a tengely és a $\boldsymbol{p} - \boldsymbol{c}$ vektor által kifeszített S síkkal. Ez két kört metsz ki a tóruszból a 2.7. ábrán látható módon.



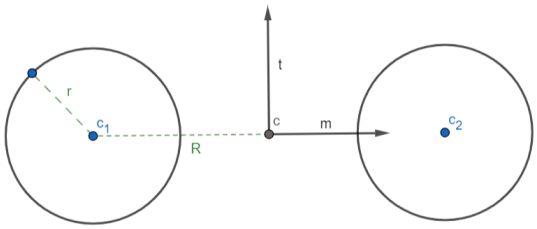
2.7. ábra. A tórusz metszése a síkkal

A körtől vett távolság meghatározásához felírtam egy ortonormált bázist, ami az előbbi síkot feszíti ki. Ennek egyik vektora a tengely, a másik pedig egy a tengelyre merőleges $\boldsymbol{b}$ vektor S síkban. Ez egyszerűen számolható $\boldsymbol{t}$-ből és $\boldsymbol{p} - \boldsymbol{c}$-ből, úgy hogy először felírok egy S-re merőleges $\boldsymbol{m}$ vektort: $\boldsymbol{m} = \boldsymbol{t} \times (\boldsymbol{p} - \boldsymbol{c})$ majd egy $\boldsymbol{t}$-re és $\boldsymbol{m}$-re merőleges vektort: $\boldsymbol{b} = \boldsymbol{m} \times \boldsymbol{t}$. Ekkor $\boldsymbol{t}$ és $\boldsymbol{b}$ vektorok alkotják a keresett ortonormált bázist.

Ezen vektorok segítségével felírható a két metszetkör középpontja ($\boldsymbol{c}_1, \boldsymbol{c}_2$). A középpontok a tórusz középpontjától R távolságra vannak a $\boldsymbol{c}$ és $\boldsymbol{b}$ által meghatározott egyenesen (2.8. ábra):

$$\boldsymbol{c}_1 = \boldsymbol{c} - \boldsymbol{m} \cdot R$$

$$\boldsymbol{c}_2 = \boldsymbol{c} + \boldsymbol{m} \cdot R$$



2.8. ábra. A két metszetkör középpontjai

A két körnek így már ismert a középpontja, a sugara pedig éppen r. A $\boldsymbol{p}$ pont köröktől vett távolsága a kör távolságfüggvényével felírható.

$$d_1 = \|\boldsymbol{p} - \boldsymbol{c}_1\| - r$$

$$d_2 = \|\boldsymbol{p} - \boldsymbol{c}_2\| - r$$

A tórusztól vett távolság a két távolság közül a kisebbik.

$$d = min(d_1, d_2)$$

A tórusztól vett távolság zárt képlettel is felírható[3]. Ebben forgatással az e_2 tengelyű tórusztól vett távolságra vezeti vissza az eredményt. Az e_2 tengelyű esetben a felírt bázist előre ismert: e_1, e_2.

$$d = (\|\boldsymbol{p} - \boldsymbol{c}\|^2 + r^2 + R^2)^2 - 4r^2(\|\boldsymbol{p} - \boldsymbol{c}\|^2 - \langle \boldsymbol{t}, \boldsymbol{p} - \boldsymbol{c} \rangle^2) \tag{2.4}$$

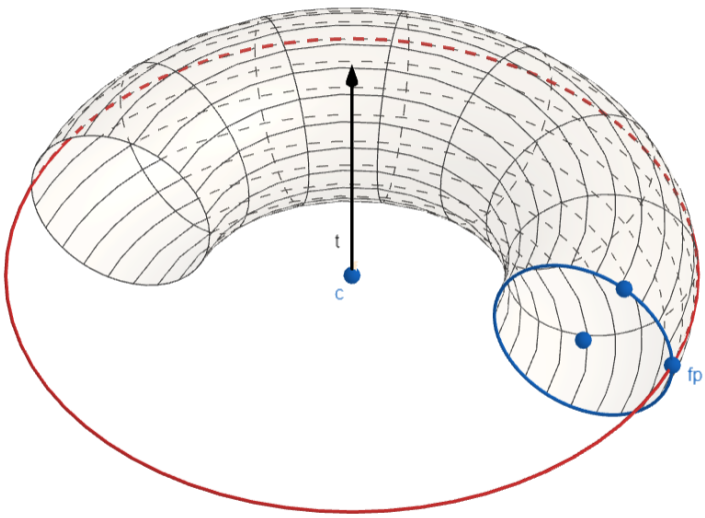
A tórusz távolságfüggvénye ebben a reprezentációban egyszerűen számolható, de a paraméterek nem illeszkednek a távolságmezőhöz. A legalapvetőbb adat, amit a mezőnek vissza kell adnia a talppont, ezért áttértem egy olyan tóruszreprezentációra, amely a legközelebbi ponttal és az ottani görbületértékekkel írja le a geometriát.

Emellett a fent kifejtett megadásnak az is hátránya, hogy a síkot csak egy ∞ sugarú tórusszal tudnánk közelíteni. Ehhez nehéz lenne megfelelően végigvihető számítási módot találni, valamely lépés során jó eséllyel NaN-t kapnánk, ezért jobb lenne egy olyan felírás, amely véges értékekkel tudná reprezentálni a geometriát.

2.6.2. Talppontos reprezentáció

Az eddig használt adatok helyett érdemesebb olyan értékeket tárolni, amelyek illeszkednek a távolságmezőtől elvárt eredményhez.

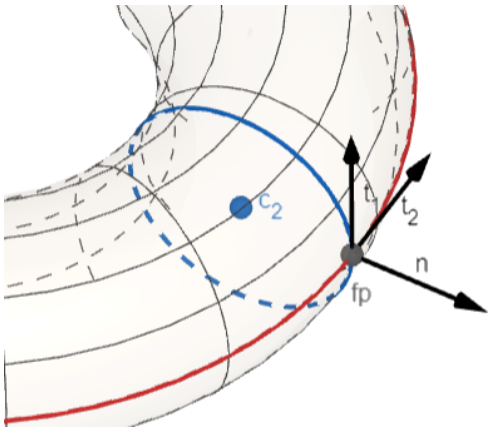
Ehhez a tóruszt a következő paraméterekkel határoztam meg: talppont($\boldsymbol{fp}$), nagyobb főgörbület (κ_r), kisebb főgörbület(κ_R) és a tengely ($\boldsymbol{t}$). A paraméterek a 2.9. ábráról leolvashatóak.



2.9. ábra. A talppontos reprezentáció adatai (a kék kör görbülete κ_r, a pirosé κ_R)

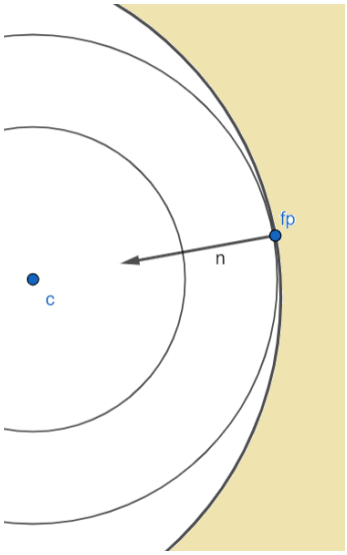
Ebben a reprezentációban a talppont megkeresése után a két főgörbületet kell meghatározni, erre kidolgoztam egy módszert, amit a következő részben fogok ismertetni. A két főgörbület és a talppont meghatároz egy tóruszt, és bármely főgörbület kombináció megfeleltethető egy megfelelő tórusz megfelelő pontjának.

A tengely irányát a kisebb főgörbület irányának választom. A két főgörbületi irány merőleges egymásra (2.10. ábra), ezért ebben a reprezentációban a talppont az illesztett tórusz belső, vagy külső ívén vagy a tetején fog elhelyezkedni. Ez előnyös, hiszen így a talppontban vett normálvektor ($\boldsymbol{n} = \frac{\nabla f(\boldsymbol{fp})}{\|\nabla f(\boldsymbol{fp})\|}$) merőleges a tengelyre, egyszerűen számolható és nem kell külön eltárolni.



2.10. ábra. A talppontban vett főgörbületi irányok és a normálvektor

Megjegyzés. Fontos megjegyezni, hogy a talppontban vett normálvektor nem feltétlenül lesz az illesztett tórusz normálvektora, hiszen például a 2.11. ábrán látható esetben, amikor mindkét görbület pozitív, akkor az illesztett tórusz kívülről fog hozzásimulni a felülethez, $\boldsymbol{n}$ pedig a tórusz belseje felé fog mutatni, míg a tórusz helyes normálvektora pont az ellenkező irányba nézne.A görbületek előjelét mindig $\boldsymbol{n}$-hez képest értelmezem.



2.11. ábra. Példa. A tórusz kívülről érinti a felületet (sárga a felület)

Ebben a reprezentációban új számítási módot kell kitalálni a tárolt geometria távolságfüggvényére. Ez a megadás helyesen kezeli a szélsőséges eseteket, mint például a sík, így ezekre is ki kell térni.

Adottak κ_r és κ_r, ahol $\kappa_r \geq \kappa_R$, $\boldsymbol{fp} \in \mathbb{R}^3$ és $\boldsymbol{t} \in \mathbb{R}^3$.

Emelett szükség lesz még a felület talppontban vett normálvektorára. $\boldsymbol{n}$-ről tudjuk, hogy a talppontból a mintavételezési pontba mutató vektorral párhuzamos és egységhosszú. A legközelebbi mintavételezési pont a felbontás és befoglaló tégalap alapján szá-

molható, így $\boldsymbol{n} = \frac{\boldsymbol{grid}(i,j,k) - \boldsymbol{fp}}{|\boldsymbol{grid}(i,j,k) - \boldsymbol{fp}|}$ ahol $\boldsymbol{grid}(i,j,k)$ a mintavételi pozíció, azaz a mező $\boldsymbol{p}$ pozíciót tartalmazó cellájának középpontja. $\boldsymbol{tex} = \lfloor (\frac{\boldsymbol{p}}{box} + 0.5) \cdot (res - 1) \rfloor$ a cella indexe, ebből $\boldsymbol{grid}(i,j,k)$ a (2.1) képlettel számolható.

Először tekintsük azokat az eseteket, ahol valamelyik görbület 0:

Sík

A sík esetében κ_r és κ_R is 0 értéket vesz fel. Ekkor a talppont és a talppontban vett normálvektor meghatározza a síkot. A sík (2.3)-ban bemutatott távolságfüggvénye használható.

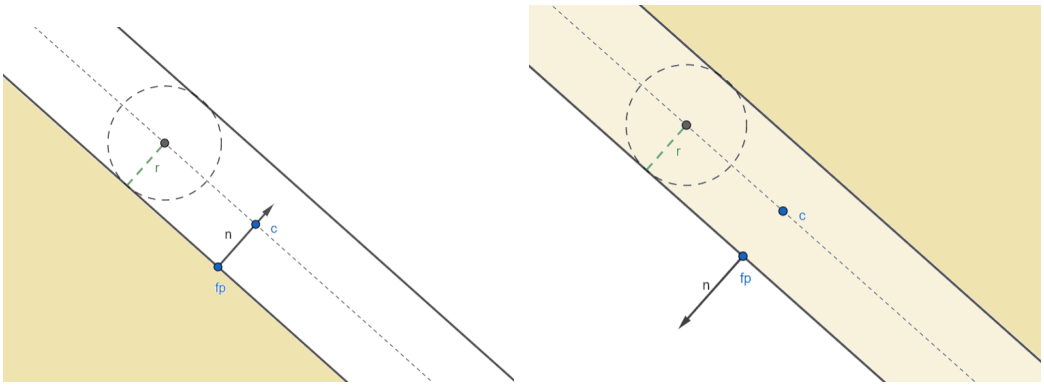
Henger

Ha $\kappa_R = 0$ akkor a közelítő geometria egy végtelen henger. A végtelen henger megadható a sugarával (r), a tengelyegyenesére merőleges vektorral ($\boldsymbol{t}_h$) és a tengelyegyenesének egy pontjával ($\boldsymbol{c}$).

$r = \frac{1}{|\kappa_r|}$

A tengelyegyenes éppen $\boldsymbol{n}$ és $\boldsymbol{t}$ vektorokra merőleges. A tengelyegyenes egy pontja a talppontból és a sugárból megkapható. $\boldsymbol{c}$ a talpponttól r távolságra van, $\boldsymbol{n}$ vagy -$\boldsymbol{n}$ irányban, attól függően milyen a κ_r előjele. (2.12. ábra)

$$C = \begin{cases} \boldsymbol{fp} - r \cdot \boldsymbol{n}, & \text{ha } \kappa_r < 0 \\ \boldsymbol{fp} + r \cdot \boldsymbol{n}, & \text{ha } \kappa_r > 0 \end{cases} \tag{2.5}$$



(a) κ_r pozitív (b) κ_r negatív

2.12. ábra. A henger közéépontjának számítás

Ekkor a $\boldsymbol{p} \in \mathbb{R}^3$ pont hengertől vett távolság:

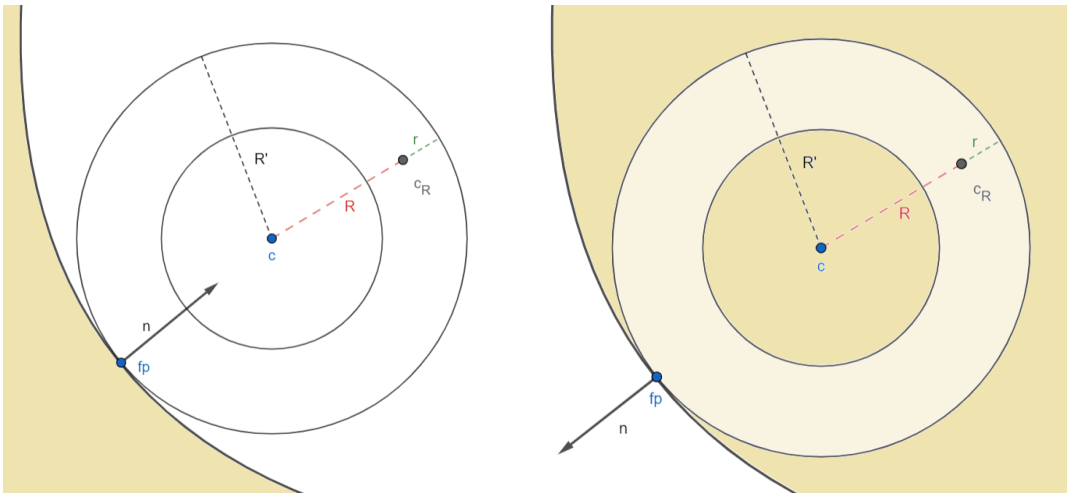
$$d_{henger} = \| \begin{bmatrix} \langle \boldsymbol{p} - \boldsymbol{c}, \boldsymbol{n} \rangle \\ \langle \boldsymbol{p} - \boldsymbol{c}, \boldsymbol{t} \rangle \end{bmatrix} \| - r \tag{2.6}$$

Tórusz

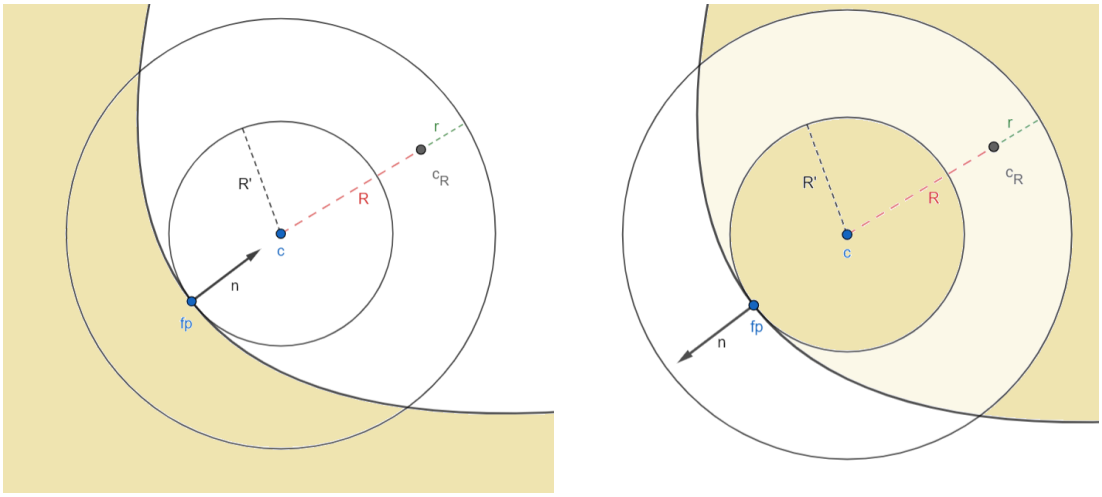
Az utolsó eset, amikor $|\kappa_r| > 0$ és $|\kappa_R| > 0$. Ekkor a paraméterek egy valódi tóruszt írnak le. A tárolt értékekből könnyen kiszámolhatóak az előző részben bemutatott reprezentáció paraméterei, így az ott leírt módon számolható a távolságfüggvény.

A két főgörbület közül a nagyobb megfelel az előző reprezentáció r értékének reciprokának. κ_r előjele azt mutatja meg, hogy a tórusz főköre a normálvektorral megegyező vagy ellentétes irányban helyezkedik el a talpponttól.

Az R értéke a κ_R-ből kiszámolható. (2.8) $R' = \frac{1}{|\kappa_R|}$ a tórusz felületén elhelyezkedő talppont középponttól vett távolságát adja meg, míg R a tórusz főkörének egy pontjának ($\boldsymbol{c}_R$) a középponttól vett távolsága. A két érték között biztosan r lesz a különbség csak azt kell meghatározni, melyik a nagyobb. Ha κ_r és κ_R előjele azonos, az azt jelenti, hogy a talppont a tórusz egy elliptikus pontjában van, mivel a felülettől ugyanabba az irányba görbül mindkét görbület. Ekkor $R < R'$. Ha az előjelek különbözőek, az azt jelenti, hogy a talppont egy hiperbolikus pontban van, azaz $R > R'$. Az előjelek szerinti esetek a 2.13. ábrán láthatók.



(a) κ_r és κ_R azonos előjelű (a talppont elliptikus pontban van)



(b) κ_r és κ_R eltérő előjelű (a talppont hiperbolikus pontban van)

2.13. ábra. R' és R viszonya a különböző előjelpárok esetén

$$r = \frac{1}{|\kappa_r|} \tag{2.7}$$

$$R = \begin{cases} \frac{1}{|\kappa_R|} - r, & \text{ha } \kappa_r \cdot \kappa_R > 0 \\ \frac{1}{|\kappa_R|} + r, & \text{ha } \kappa_r \cdot \kappa_R < 0 \end{cases} \tag{2.8}$$

Végül szükség van a tórusz középpontjára is. A középpont R' távolságra van a talpponttól, $\boldsymbol{n}$ vagy -$\boldsymbol{n}$ irányban a κ_R előjelétől függően. Ha κ_R pozitív $\boldsymbol{c}$ az $\boldsymbol{n}$ irányában van, ha negatív -$\boldsymbol{n}$ irányban.

$$\boldsymbol{c} = \begin{cases} \boldsymbol{fp} + \boldsymbol{n} \cdot \frac{1}{|\kappa_R|}, & \text{ha } \kappa_R > 0 \\ \boldsymbol{fp} - \boldsymbol{n} \cdot \frac{1}{|\kappa_R|}, & \text{ha } \kappa_R < 0 \end{cases} \tag{2.9}$$

A tórusz tengelye mindkét reprezentációban azonos. Így adott r, R, $\boldsymbol{c}$ és $\boldsymbol{t}$ amiből a tórusztól vett távolság az előző részben mutatott módszerekkel számolható. A másodrendű

3 dimenziós mező egy mintája 8 valós számmal tárolható.

2.6.3. Görbületek illesztése

Megmutattam egy alkalmas reprezentációt, amely képes tárolni a mezőtől elvárt adatokat. Ehhez a talppontban vett főgörbületeket és főirányokat kell csak meghatározni. Ezek számolására kétlépéses módszert dolgoztam ki. Először a felületet a talppont lokális környezetében egy harmadfokú polinomiális felülettel közelítem, majd meghatározom a keresett görbületeket Weingarten mátrix segítségével a [4] alapján.

A harmadfokú felület megállapításához először felületi pontokra van szükség a talppont egy környezetéből. Ehhez felírom a felület talppontban vett érintősíkját. A talppont és a talppontban vett normálvektor ismert, így $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ érintősíkot kifeszítő vektorokra van szükség. A 2.14. ábrán szereplő $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, $\boldsymbol{n}$ bázist a [5]-ben leírt algoritmussal számítom ki.

1. Algoritmus Az talppontban vett bázis számítása

if $\boldsymbol{n}_z < -0.9999999$ **then**

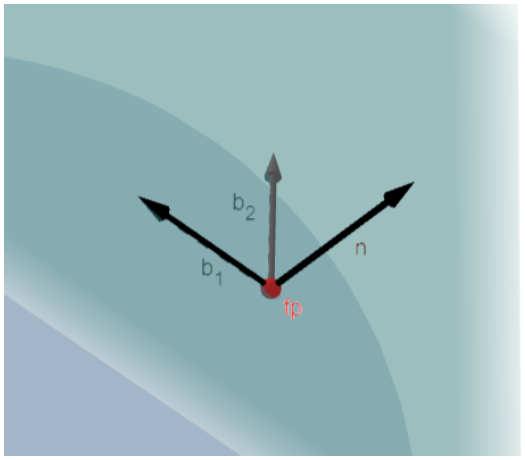
$$\boldsymbol{b}_1 = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \quad \boldsymbol{b}_2 = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}$$

else

$$a = \frac{1}{1+\boldsymbol{n}_z} \quad b = -\boldsymbol{n}_x \cdot \boldsymbol{n}_y \cdot a$$

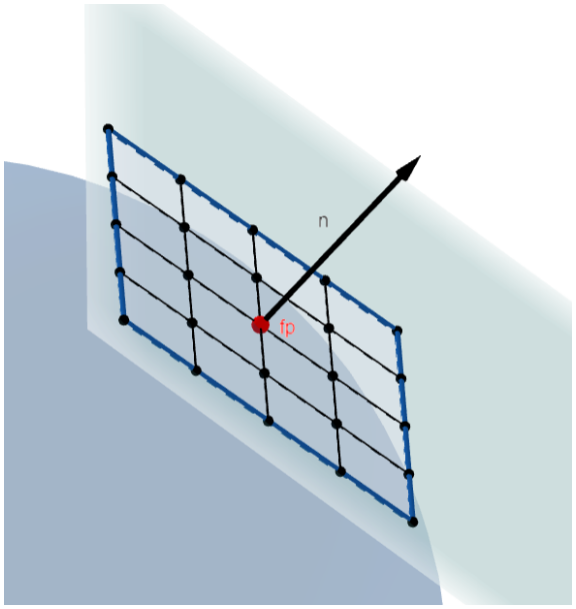
$$\boldsymbol{b}_1 = \begin{bmatrix} 1-\boldsymbol{n}_x \cdot \boldsymbol{n}_x \cdot a \\ b \\ -\boldsymbol{n}_x \end{bmatrix} \quad \boldsymbol{b}_2 = \begin{bmatrix} b \\ 1-\boldsymbol{n}_y \cdot \boldsymbol{n}_y \cdot a \\ -\boldsymbol{n}_y \end{bmatrix}$$

end if



2.14. ábra. A talppontban vett lokális bázis

A kisebb rács felbontása legyen res_f, oldalhossza $size_f$. A felbontás itt azt jelenti, hogy a kis rács $res_f \cdot 2 + 1$ pontot fog tartalmazni mindkét dimenziója mentén, erre példa a 2.15. ábrán található.



2.15. ábra. Az érintősíkon vett finom rács $res_f = 2$ esetben

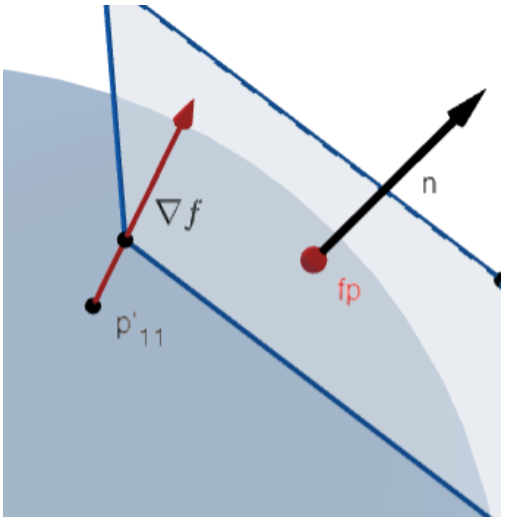
A finomabb rács pontjait (p_f) a globális koordinátarendszerben a következő módon számolom: $\forall i, j \in [-res_f \dots res_f], \quad i, j \in \mathbb{N}$

$$\boldsymbol{p}_f(i,j) = \boldsymbol{fp} + \boldsymbol{b}_1 \cdot i \cdot s + \boldsymbol{b}_2 \cdot j \cdot s \tag{2.10}$$

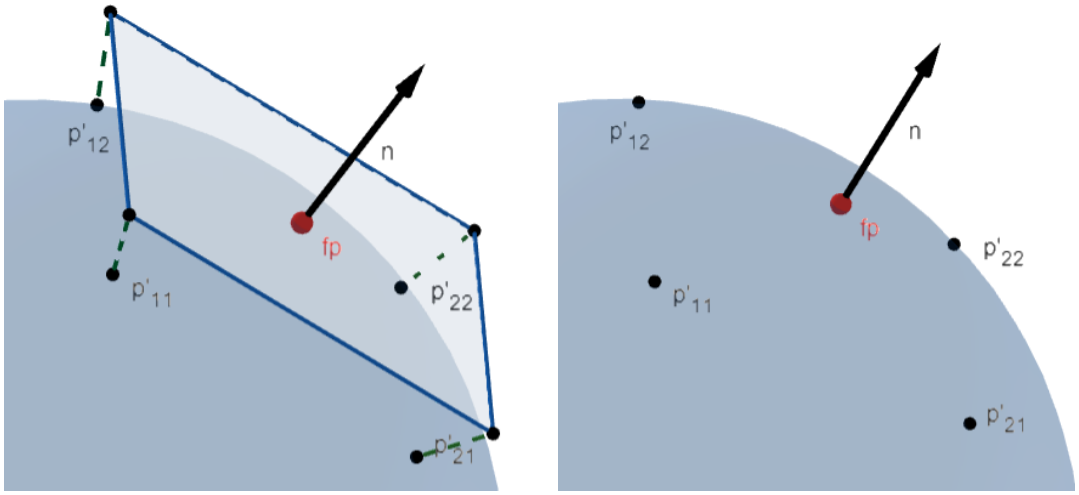
ahol $s = \frac{size_f}{res_f}$ a finom rács két pontja közti távolságot írja le.

A rácspontokhoz ezután felületi pontokat kell hozzárendelni. Ehhez két különböző megközelítést használtam.

- talppont keresése: minden rácsponthoz a gradiens módszerrel megkeresem a felület legközelebbi pontját (2.16. ábra)



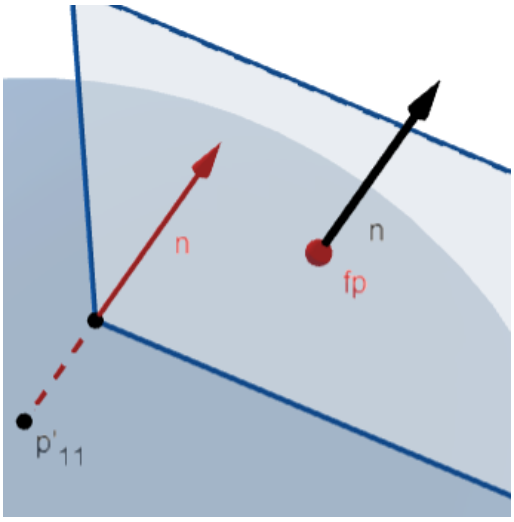
(a) A finom rács egy pontjához tartozó talppont megkeresése



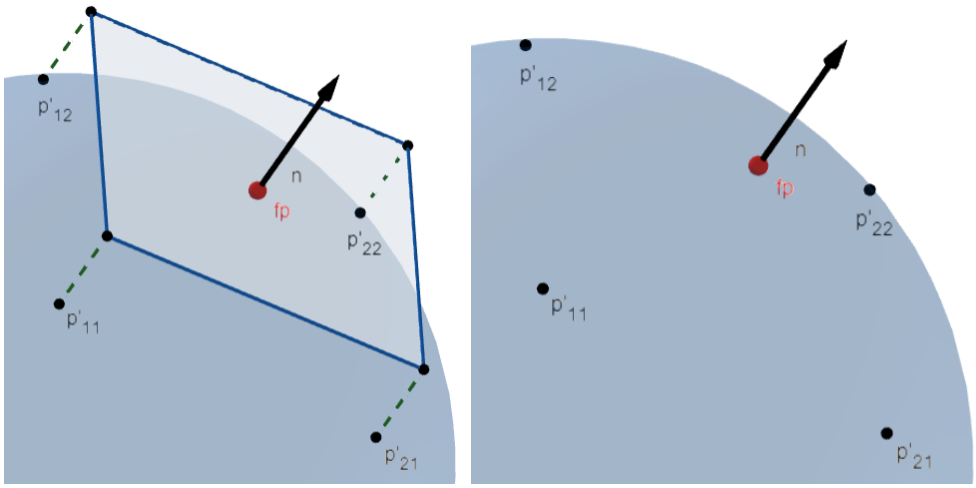
(b) Az összes ponthoz vett talppont

2.16. ábra. A felületi pontok keresése, talppont kereséssel

- merőleges vetítés: minden rácspontot merőlegesen vetítek a felületre sphere tracing algortimus segítségével (2.17. ábra)



(a) A finom rács egy pontjának merőleges vetítése



(b) Az összes pont merőleges vetítése

2.17. ábra. A felületi pontok keresése, merőleges vetítéssel

A két módszer jelentős különbségeket okozott az eredményben, valamint az illesztés megvalósításában, ezt a későbbiekben részletesen kifejtem.

Így megkaptam a felületi pontok halmazát ($\boldsymbol{p}'$), amellyel fel tudom írni a lokális harmadfokú közelítést [4].

Adott ponthalmazra illesztett harmadfokú felület:

$$f(x,y) = \frac{A}{2}x^2 + Bxy + \frac{C}{2}y^2 + Dx^3 + Ex^2y + Fxy^2 + Gy^3 \tag{2.11}$$

A másodfokú tagok együtthatóját a későbbi számolás megkönnyítése miatt kettővel osztva veszem.

Keressük a $\boldsymbol{b} = \begin{bmatrix} A & B & C & D & E & F & G \end{bmatrix}^T$ vektort, amelyre

$$f(x_i, y_i) = \begin{bmatrix} \frac{x_i^2}{2} & x_i y_i & \frac{y_i^2}{2} & x_i^3 & x_i^2 y_i & x_i y_i^2 & y_i^3 \end{bmatrix} \boldsymbol{b} = z_i \tag{2.12}$$

Ehhez a felületi pontokat át kell írni egy lokális koordinátarendszerbe, melynek origója a talppont, x,y tengelye $\boldsymbol{b}_1$ és $\boldsymbol{b}_2$, z tengelye n.

Az $\boldsymbol{p}$ pont lokális koordinátái:

$$\boldsymbol{p} = \begin{bmatrix} \langle \boldsymbol{p}'_x , \boldsymbol{b}_1 \rangle \\ \langle \boldsymbol{p}'_y , \boldsymbol{b}_2 \rangle \\ \langle \boldsymbol{p}'_z , \boldsymbol{n} \rangle \end{bmatrix} \tag{2.13}$$

Így van egy $\boldsymbol{p}$ ponthalmazunk az origó környezetéből és ezekre keressük $\boldsymbol{b}$-t amire teljesül a (2.12) egyenlet.

Ez a számítás nem veszi figyelembe a felületi pontok normálvektorát, amit szintén könnyen ki tudunk számolni a gradiensből. A finom rács felületi pontjaihoz először a gradienssel kiszámoljuk a normálvektort ($\boldsymbol{n}'$), majd átváltjuk a lokális koordinátákra ($\boldsymbol{n}_l$).

$$\boldsymbol{n}' = \frac{\nabla f(\boldsymbol{p}')}{\|\nabla f(\boldsymbol{p}')\|} \tag{2.14}$$

$$\boldsymbol{n}_l = \begin{bmatrix} \langle \boldsymbol{n}'_x, \boldsymbol{b}_1 \rangle \\ \langle \boldsymbol{n}'_y , \boldsymbol{b}_2 \rangle \\ \langle \boldsymbol{n}'_z , \boldsymbol{n} \rangle \end{bmatrix} \tag{2.15}$$

Az $\boldsymbol{n}_l = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$ normálvektort átalakítva $\boldsymbol{n}_l = \begin{bmatrix} -\frac{a}{c} \\ -\frac{b}{c} \\ -1 \end{bmatrix}$ alakba $\partial_x f$-re és $\partial_y f$-re is felírható két egyenlet:

$$\partial_x f(x_i, y_i) = \begin{bmatrix} x_i & y_i & 0 & 3x_i^2 & 2x_i y_i & y_i^2 & 0 \end{bmatrix} \boldsymbol{b} = -\frac{a_i}{c_i} \tag{2.16}$$

$$\partial_y f(x_i, y_i) = \begin{bmatrix} 0 & x_i & y_i & 0 & x_i^2 & 2x_i y_i & 3y_i^2 \end{bmatrix} \boldsymbol{b} = -\frac{b_i}{c_i} \tag{2.17}$$

A (2.12), (2.16) és (2.17) egyenleteket a finom rács minden felületi pontjára felírva és egyesítve azokat egy lineáris egyenletrendszert kapok:

$$A\boldsymbol{b} = \boldsymbol{z} \tag{2.18}$$

Ahol

$$A = \begin{bmatrix} \frac{x_1^2}{2} & x_1y_1 & \frac{y_1^2}{2} & x_1^3 & x_1^2y_1 & x_1y_1^2 & y_1^3 \\ x_1 & y_1 & 0 & 3x_1^2 & 2x_1y_1 & y_1^2 & 0 \\ 0 & x_1 & y_1 & 0 & x_1^2 & 2x_1y_1 & 3y_1^2 \\ . & & & & & & \\ . & & & & & & \\ . & & & & & & \\ \frac{x_1^2}{2} & x_Ny_N & \frac{y_N^2}{2} & x_N^3 & x_N^2y_N & x_Ny_N^2 & y_N^3 \\ x_N & y_N & 0 & 3x_N^2 & 2x_Ny_N & y_N^2 & 0 \\ 0 & x_N & y_N & 0 & x_N^2 & 2x_Ny_N & 3y_N^2 \end{bmatrix} \in \mathbb{R}^{3N\times 7} \tag{2.19}$$

és $N = (2res_f + 1)^2$, továbbá

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ -\frac{a_1}{c_1} \\ -\frac{b_1}{c_1} \\ . \\ . \\ . \\ z_N \\ -\frac{a_N}{c_N} \\ -\frac{b_N}{c_N} \end{bmatrix} \in \mathbb{R}^N \tag{2.20}$$

Ezt az egyenletrendszert pontosan nem biztos, hogy meg lehet oldani, de a legkisebb négyzetek módszerével könnyen kapható közelítő megoldás. Ekkor azt szeretnénk elérni, hogy $\|A\boldsymbol{b} - \boldsymbol{z}\|^2$ minimális legyen. A legkisebb négyzetes hibás megoldás Moore-Penrose féle pszeudoinverzzel számolható:

$$\boldsymbol{b} = A^+\boldsymbol{z} \tag{2.21}$$

ahol

$$A^+ = (A^TA)^{-1}A^T \tag{2.22}$$

A Moore-Penrose pszeudoinverz bármilyen, akár nem invertálható mátrixra is számolható, így A-ra semmilyen különösebb kikötést nem kell tenni.

Miután $\boldsymbol{b}$ megvan A, B és C értékekből a [4]-ban levezetett eredmények alapján felírható a harmadfokú felület Weingarten-mátrixa:

$$W = \begin{bmatrix} A & B \\ B & C \end{bmatrix} \tag{2.23}$$

A Weingarten-mátrix fontos tulajdonsága, hogy sajátértékei a felület főgörbületeit adják meg, a sajátvektorok pedig az adott göbülethez tartozó főirányt. A mátrix két sajátér-

tékét meghatározva tehát megkapjuk κ_r és κ_R értékét, a κ_r sajátértékhez tartozó főirány pedig a $\boldsymbol{t}$ tengely lesz.

A sajátértékek 2×2-es mátrix esetén egy másodfokú egyenlet segítségével számolhatóak:

$$\lambda_{1,2} = (A+C) \pm \sqrt{(A+C)^2 - 4(A \cdot C - B^2)} \tag{2.24}$$

A sajátrétékekből a nagyobbik lesz κ_r a kisebbik κ_R. A nagyobb sajátérétkhez (legyen λ_1) tartozó sajátvektor:

$$\boldsymbol{v}_1 = \begin{bmatrix} -1 \\ \frac{A-\lambda_1}{B} \end{bmatrix} \tag{2.25}$$

A főirány ekkor még a talppontban vett $\begin{bmatrix} 1 \\ 0 \\ f_x \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ f_y \end{bmatrix}$ lokális bázisra vonatkozik. Mivel a számoláshoz használt lokális koordinátarendszer origója a talppont volt, ezért f_x és f_y értéke 0, tehát a bázis egybeesik az x és y tengellyel. Ez alapján a sajátvektort átírva a globális koordinátarendszerbe megkapjuk a tórusz tengelyét.

$$\boldsymbol{t} = \boldsymbol{v}_{1x} \cdot \boldsymbol{b}_1 + \boldsymbol{v}_{1y} \cdot \boldsymbol{b}_2 \tag{2.26}$$

Ezzel minden adatot megkaptunk ami a tórusz megadásához kell, azonban ez a megoldás még nem optimális és nehezen párhuzamosítható a GPU-n a nagy mátrixra elvégzett invertálás miatt.

A pszeudoinverz számolását nem lenne muszáj minden egyes mintavételezési pontban elvégezni, hiszen a felírt (2.18) egyenletrendszer minden helyen hasonlóan néz ki. Az egyenletrendszer bal oldalát előre felírva kiszámolhatjuk a pszeudoinverzt és utána a mező minden cellájában ezzel számolható $\boldsymbol{b}$.

Azonban ha a felületi pontokat a finom rács pontjaiból talppontkereséssel számítjuk ki a $\boldsymbol{p}$ elemeinek x,y koordinátái helyenként változóak lehetnek, akár olyan eset is előfordulhat hogy az összes egybeesik. A pszeudoinverz előzetes számításához szükséges lenne az x és y értékeket előre ismerni, ami a talppontkeresésnél nem megoldható, hiszen a talppontok pozíciója a lokális geometriától függ.

Ha a merőleges vetítést alkalmazzuk, a lokális koordinátarendszerben vett x,y koordináták ugyanazok maradnak, mint a vetítés előtt, így ezek a pozíciók csak res_f és $size_f$ értékektől függenek. Ezeket a mező generálása előtt is ismerjük, így előre felírható egy

általános A_0 mátrix és belőle a pszeudoinverz. A mező generálása során csak a $\boldsymbol{z}$ vektor értékeit kell kiszámolni a lokális geometriák alapján.

Az A_0 mátrixot úgy készítem el, hogy az origó körül veszek egy négyzetet az XY síkban. Ez megfelel a talppontokban felírt lokális koordinátarendszereknek.

2. Algoritmus Az általános A_0 mátrix felírása

for $i = -res_f \dots res_f$ **do**
  for $i = -res_f \dots res_f$ **do**
    $\boldsymbol{p} = i \cdot \boldsymbol{e}_1 \cdot \frac{box}{2res_f+1} + j \cdot \boldsymbol{e}_2 \cdot \frac{box}{2res_f+1}$
    $A_0(3((i+res_f) \cdot res_f + (j+res_f))) = \begin{bmatrix} \frac{p_x^2}{2} & p_x p_y & \frac{y_1^2}{2} & p_x^3 & p_x^2 p_y & p_x p_y^2 & p_y^3 \end{bmatrix}$
    $A_0(3((i+res_f) \cdot res_f + (j+res_f))+1) = \begin{bmatrix} p_x & p_y & 0 & 3p_x^2 & 2p_x p_y & p_y^2 & 0 \end{bmatrix}$
    $A_0(3((i+res_f) \cdot res_f + (j+res_f))+2) = \begin{bmatrix} 0 & p_x & p_y & 0 & p_x^2 & 2p_x p_y & 3p_y^2 \end{bmatrix}$
  end for
end for

Abből a pszeudoinverz: $A_0^+ = (A_0^T A_0)^{-1} A_0^T$

Ezután a mező adott mintavételezési pontjára a talppont megkeresése után a következő algoritmussal felírható a z vektor:

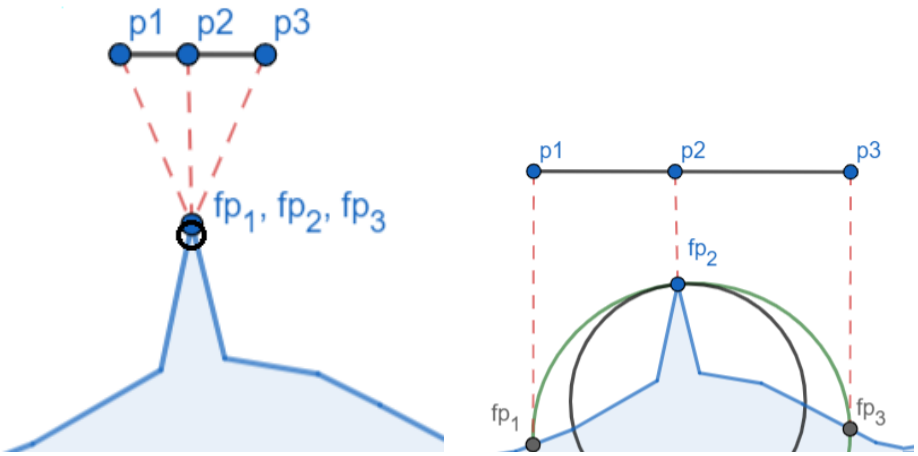
3. Algoritmus A $\boldsymbol{z}$ vektor felírása

for $i = -res_f \dots res_f$ **do**
  for $j = -res_f \dots res_f$ **do**
    $\boldsymbol{p}_f = i \cdot \boldsymbol{b}_1 \cdot \frac{box}{2res_f+1} + j \cdot \boldsymbol{b}_2 \cdot \frac{box}{2res_f+1}$
    $\boldsymbol{p}' =$ Merőleges vetítés $(\boldsymbol{p}_f)$
    $\boldsymbol{n}' = \frac{\nabla f(\boldsymbol{p}')}{\|\nabla f(\boldsymbol{p}')\|}$
    $\boldsymbol{p} =$ Lokális koordinátarendszerbe $(\boldsymbol{p}')$
    $\boldsymbol{n} =$ Lokális koordinátarendszerbe $(\boldsymbol{n}')$
    $\boldsymbol{z}(3((i+res_f) \cdot res_f + (j+res_f))+1) = \boldsymbol{p}_z$
    $\boldsymbol{z}(3((i+res_f) \cdot res_f + (j+res_f))+1) = -\frac{\boldsymbol{n}_x}{\boldsymbol{n}_z}$
    $\boldsymbol{z}(3((i+res_f) \cdot res_f + (j+res_f))+2) = -\frac{\boldsymbol{n}_y}{\boldsymbol{n}_z}$
  end for
end for

A $\boldsymbol{z}$ vektor első 3 eleméből felírva a Weingarten mátrixot a (2.24) és (2.25) alapján számolhatóak a görbületek és a tengely.

Ez a számolás a GPU-n is könnyen kivitelezhető, hiszen csak a pszeudoinverzt linearizálva átmásolom majd utána egyszerű szorzásokkal kiszámolható $\boldsymbol{b}$, így párhuzamosítva a mező generálása jól gyorsítható.

A merőleges vetítés az illesztett geometria folytonosságát tekintve is előnyösebb a talppontkeresésnél.

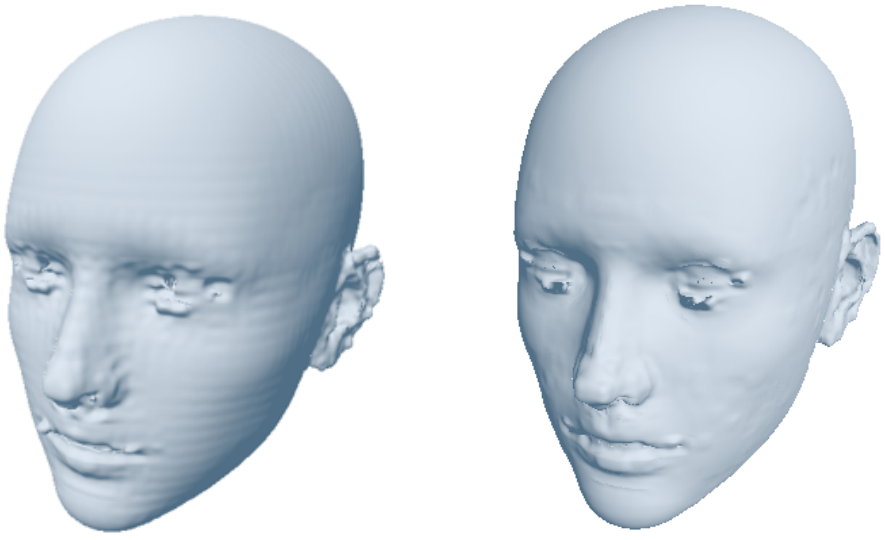


(a) Talppontkeresésnél az egybeeső talppontokra szélsőséges eredményt kapunk

(b) Merőleges vetítésnél átlagosabb a közelítés

2.18. ábra. Példa a talppontkeresés és a merőleges vetítés közti különbségre

Az 2.18-es ábrán látható esetben a talppontkereséssel a cellában egy szélsőségesen kicsi geometriát kapunk, mivel a talppontok egybeesnek. A vetítésnél egy nagyobb felületdarabon becsüljük a görbületeket, ezért az eredmény, a geometria is jobban approximál. A talppontkeresésnél a szomszédos cellák tartalma nagyon eltérő lehet, míg merőleges vetítésnél a görbületi értékek hasonlóbbak így a szűrésnél leírt interpolációk jobban össze tudják mosni a tárolt objektumokat, ezt példázza a 2.19. ábra.



(a) Talppontkereséssel

(b) Merőleges vetítéssel

2.19. ábra. Példa a két módszerrel kapott eredményre trilineáris interpolációnál

Kérdés lehet még a res_f és $size_f$ értéke. Az egyenletrendszer megoldásához a finom

rács pontjainak száma legalább 3 kell legyen. A sűrűbb rács ponotsabb közelítést eredményez, de általánosan jó megoldásnak bizonyul $res_f = 3$ azaz 7×7-es rács használata. A rács méretét minél kisebbnek érdemes választani, mivel a talppontban lokálisan szeretnénk becsülni a görbületet, ha túl nagy rácsot választunk, a talppont körül pontatlan lesz az eredmény. Ennek értékéül a mező egy cellájának oldalhosszának a tizedének választottam.

2.6.4. Algoritmus

Összegezve a másodrendű mező generálásának algoritmusa:

4. Algoritmus Másodrendű mező generálása

Általános A_0 mátrix felírása (**2. algoritmus**)
$A_0^+ = (A_0^T A_0)^{-1} A_0^T$ ▷ Általános pszeudoinverz
for $i = -res \dots res$ **do** ▷ A mező celláiban tárolt adatok számolása
  for $j = -res \dots res$ **do**
    for $k = -res \dots res$ **do**
      $\boldsymbol{grid}(i,j,k) = -\begin{bmatrix} box/2 \\ box/2 \\ box/2 \end{bmatrix} + \begin{bmatrix} i+0.5 \\ j+0.5 \\ k+0.5 \end{bmatrix} \cdot \frac{box}{res}$ ▷ A cella középpontja
      $\boldsymbol{fp} = \boldsymbol{grid}(i,j,k) - f(\boldsymbol{grid}(i,j,k)) \cdot \frac{\nabla f(\boldsymbol{grid}(i,j,k))}{\|\nabla f(\boldsymbol{grid}(i,j,k))\|}$ ▷ Talppont
      $\boldsymbol{n} = \frac{\nabla f(\boldsymbol{fp})}{\|\nabla f(\boldsymbol{fp})\|}$ ▷ Normálvektor a talppontban
      Lokális bázis felírása a talppontban (**1. algoritmus**)
      $\boldsymbol{z}$ vektor kiszámítása (**3. algoritmus**)
      $\boldsymbol{b} = A_0^+ \boldsymbol{z}$ ▷ A harmadfokú felület paraméterei
      $W = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$ ▷ Weingarten-mátrix
      $\lambda_{1,2} = (A+C) \pm \sqrt{(A+C)^2 - 4(A \cdot C - B^2)}$ ▷ Sajátrétékek
      if $\lambda_1 < \lambda_2$ **then** ▷ Görbületek a sajátértékekből
        $\kappa_R = \lambda_1, \quad \kappa_r = \lambda_2$
      else
        $\kappa_R = \lambda_2, \quad \kappa_r = \lambda_1$
      end if
      $\boldsymbol{v} = \begin{bmatrix} -1 \\ \frac{A-\kappa_r}{B} \end{bmatrix}$ ▷ A kisebb görbülethez tartozó főirány
      $\boldsymbol{t} = \boldsymbol{v}_x \cdot \boldsymbol{b}_1 + \boldsymbol{v}_y \cdot \boldsymbol{b}_2$ ▷ Tengely kiszámítása
      $\boldsymbol{field}(i,j,k) = \begin{bmatrix} \boldsymbol{fp} & \kappa_r & \kappa_r & \boldsymbol{t} \end{bmatrix}$
    end for
  end for
end for

2.7. Geometriák kirajzolása

A mezőben az előző részben bemutatott geometriák vannak eltárolva, olyan paraméterekkel amelyekből könnyen rekonstruálható a távolságfüggvényük. A textúrák, amelyek a mezőt tartalmazzák a program futása során átkerülnek a GPU-ra és ott lesznek feldolgozva.

A textúrából a térbeli pozíció alapján kiolvasom a megfelelő adatot. Az a geometria lesz megjelenítve, amelyik a pozíciót is tartalmazó cellában van tárolva.

Egy $res \times res \times res$ méretű tömbből kell olvasni. A térbeli pozícióról feltehető hogy a $[-\frac{box}{2}, \frac{box}{2}]$ intervallumba esik. A cella koordinátáit a következő módon számolom:

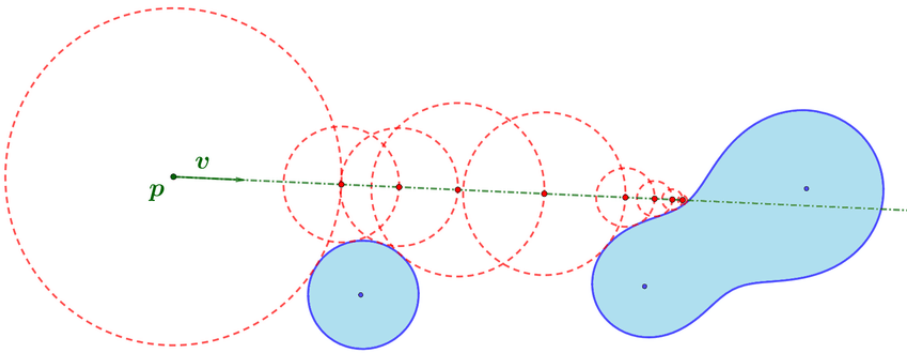
$$\boldsymbol{tex}(\boldsymbol{p}) = \lfloor (\frac{\boldsymbol{p}}{box} + 0.5) \cdot (res - 1) \rfloor \tag{2.27}$$

ahol $\boldsymbol{p} \in \mathbb{R}^3$.

2.7.1. Mező megjelenítése

A 3D-s megjelenítéshez sphere tracing algoritmust használok [6]. A sphere tracing algoritmus lényege, hogy a kamera pozíciójából sugarakat indítok a képernyő pixelein keresztül és azok mentén haladok, amíg bele nem ütköznek a felületbe, vagy túl nem lépnek egy adott határt. Ha a sugár eltalálja a felületet a pixelt színezem, különben fehér marad.

A sugáron való haladást a távolságfüggvény írja le. Egy adott pontban a távolságfüggvény értéke megadja a legközelebbi objektum távolságát (d). A pozíció körül egy d sugarú gömbben biztosan nincs semmi, hiszen akkor nem d lenne a legközelebbi távolság, tehát d-t biztosan előre léphetek a sugár mentén. Ezt folytatom amíg a d érték 0 nem lesz, vagy ki nem lépek a befoglaló dobozból. Az algoritmus működését mutatja a 2.20. ábra.



2.20. ábra. A sphere tracing algoritmus működése

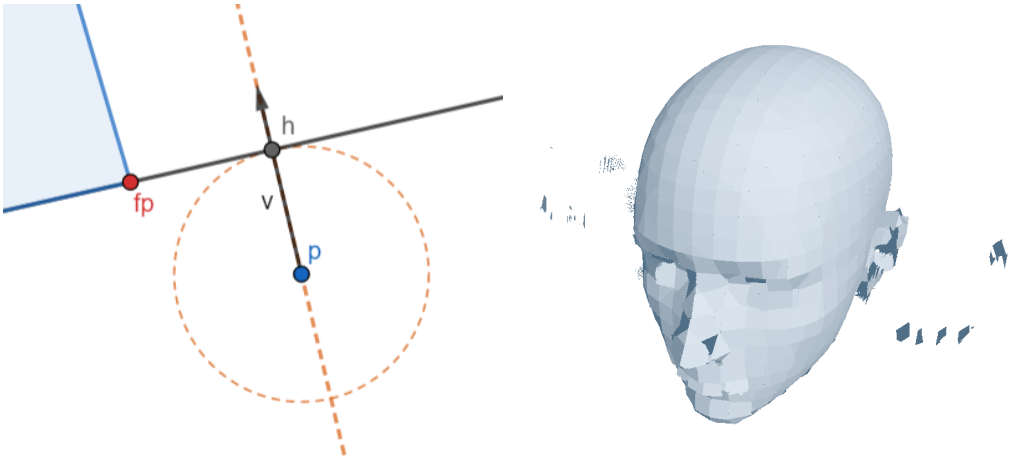
A sugarat a kezdőpozíciója ($\boldsymbol{p}_r$) és irányvektora ($\boldsymbol{v}_r$) definiálja. A sphere tracing algoritmusa az adott sugárral:

5. Algoritmus Sphere tracing algoritmus

Bemenet: Adott r sugár, ami $\boldsymbol{p}_r$ pontból indul $\boldsymbol{v}_r$ irányba és N maximális lépésszám
$t = 0$
$i = 0$
while $i < N$ és $f(\boldsymbol{p}_r + t \cdot \boldsymbol{v}_r) > \varepsilon$ és $t < box$ **do**
    $t += f(\boldsymbol{p}_r + t \cdot \boldsymbol{v}_r)$
    $i += 1$
end while

A távolságfüggvény kiszámításánál használom a mezőt. A (2.27) képlettel a pozícióból előállítom az indexet a textúraolvasáshoz. Elsőrendű mezőnél az olvasott adat leírja a közelítő síkot, ebből (2.3) távolságfüggvény azonnal megadja a távolságot.

Bizonyos esetekben azonban a sík nem ad jó eredményt, mivel csak a talppont lokális környezetében helyes közelítés. Például az olyan felületeknél, amelyeken található valamilyen kiálló rész, előfordulhat, hogy a sugár éppen merőleges lesz a közelítősíkra és az ábrán látható módon hibás találatot kapunk. (2.21. ábra)



(a) A hibás eset (kék az eredeti felület, a fekete egyenes a közelítés)

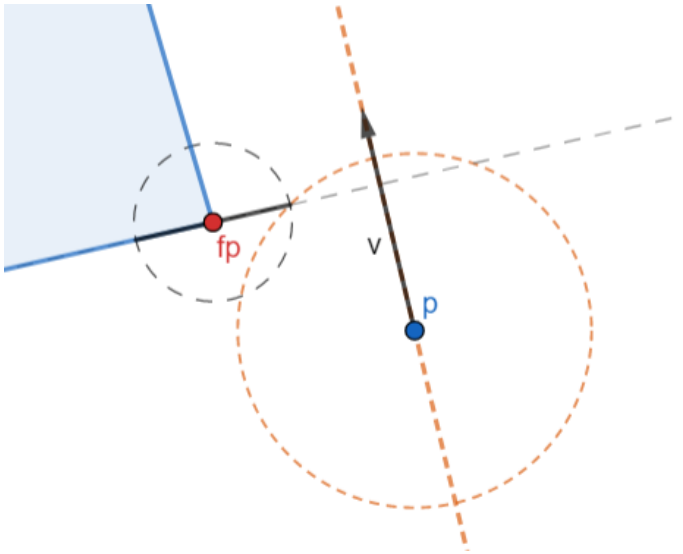
(b) A hibás eset a valóságban

2.21. ábra. Példa a lehetséges hibára

Erre megoldás, ha a síkot lekorlátozzuk a talppont egy kis lokális környezetére egy gömbbel való metszéssel. (2.22. ábra) A gömbbel való metszést megadja, ha a síktól és a gömbtől vett távolság maximumát veszem. A gömb sugarát úgy választom, hogy a mező cellájának oldalhossza legyen, így tényleg csak azt a részt veszem figyelembe a síkból ami az adott talpponthoz tartozik:

$$d = max(d_{egyenes}, d_{gomb})$$

Ahol $d_{gomb} = |\boldsymbol{fp} - \boldsymbol{p}| - \frac{box}{res}$



2.22. ábra. A javított mező (a fekete szakasz az új közelítő geometria)

Másodrendű mező esetében még bonyolultabb a dolog.

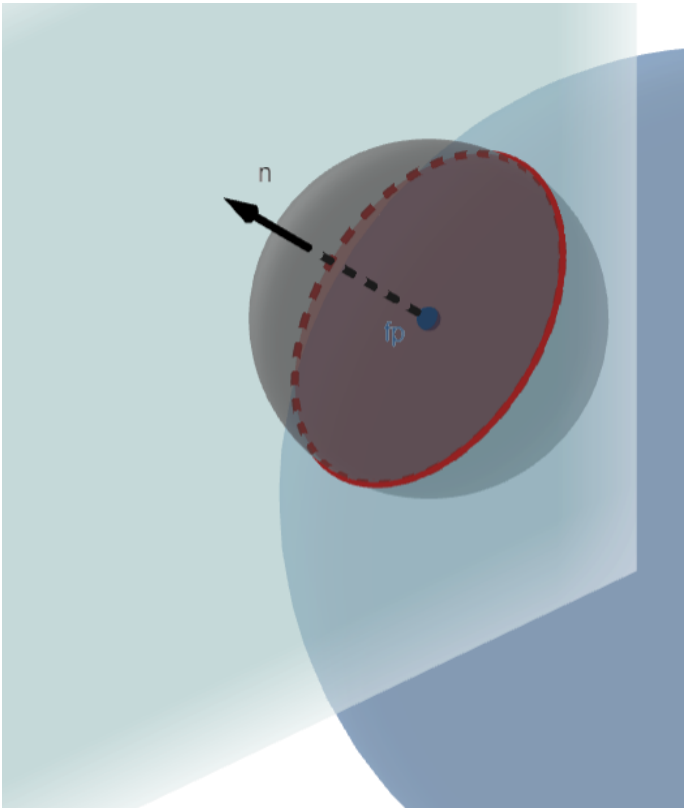
2.7.2. Másodrendű mező

Másodrendű mező esetében lehetséges, hogy valamely görbületi érték 0. Ezekben az esetekben nem a tórusz távolságfüggvényére van szükség, mivel a tórusz paramétereinek számolásakor osztani kell a görbülettel. A megvalósításnál a számítások során gyakran nem nullát, hanem 0 közeli értéket kaptam eredményül. Ekkor is célszerűbb volt bizonyos ε érték alatt nullának venni a görbületet és a következőkben leírt egyéb geometriákat használni, mivel a kis számokkal való osztás numerikus hibát és pontatlan közelítést eredményezett.

Sík

Ha $\kappa_r = 0$ és $\kappa_R = 0$ egy síkot kell kirajzolni. Ezt a talppont és a talppontban vett normálvektor adja meg. $\boldsymbol{n}$ a pozícióból számítható. Ezután a (2.3) függvénnyel számolható a síktól vett távolság, az előző részben említett hibás eset (2.23. ábra) figyelembevételével:

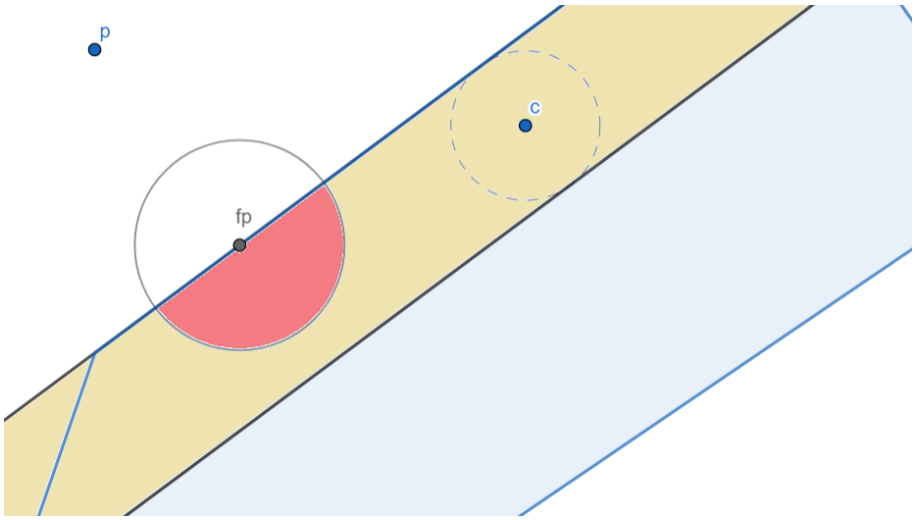
$$d = max(d_{sik}, d_{gomb})$$



2.23. ábra. A gömbbel metszett közelítő sík (piros)

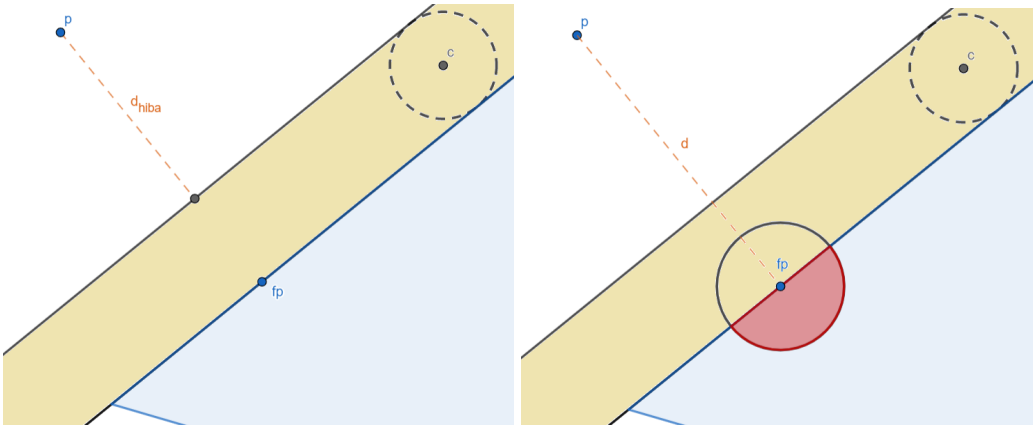
Henger

Ha $\kappa_R = 0$, akkor egy végtelen henger közelíti a felületet. A henger adatai a 2.6.2-ben leírt módon számolhatók a tárolt paraméterekből, majd a (2.6) távolságfüggvény használható. A síknál mutatott esethez hasonlóan a henger is hibás eredményt adhat a felülettelre majdnem merőlegesen haladó sugarak esetében, ezért itt is egy gömbbel le kell korlátozni a geometriát, a 2.24. ábrán látható módon.



2.24. ábra. A gömbbel metszett henger (piros)
A kékkel keretezett rész a közelített felület, a sárga a közelítő henger

Ám még így is bizonyos esetekben hibás közelítést kapunk. Ha κ_r pozítív akkor a henger kívülről simul hozzá a felülethez, a távolságfüggvény eredménye jóval kisebb lesz a felület valós távolságánál. Ekkor nem a henger hanem annak komplementere közelíti a felületet. Az előzőekben leírt esetekhez hasonlóan ezt is le kell korlátozni a gömbbel. (2.25. ábra) Tehát $d = max(-d_{henger}, d_{gomb})$



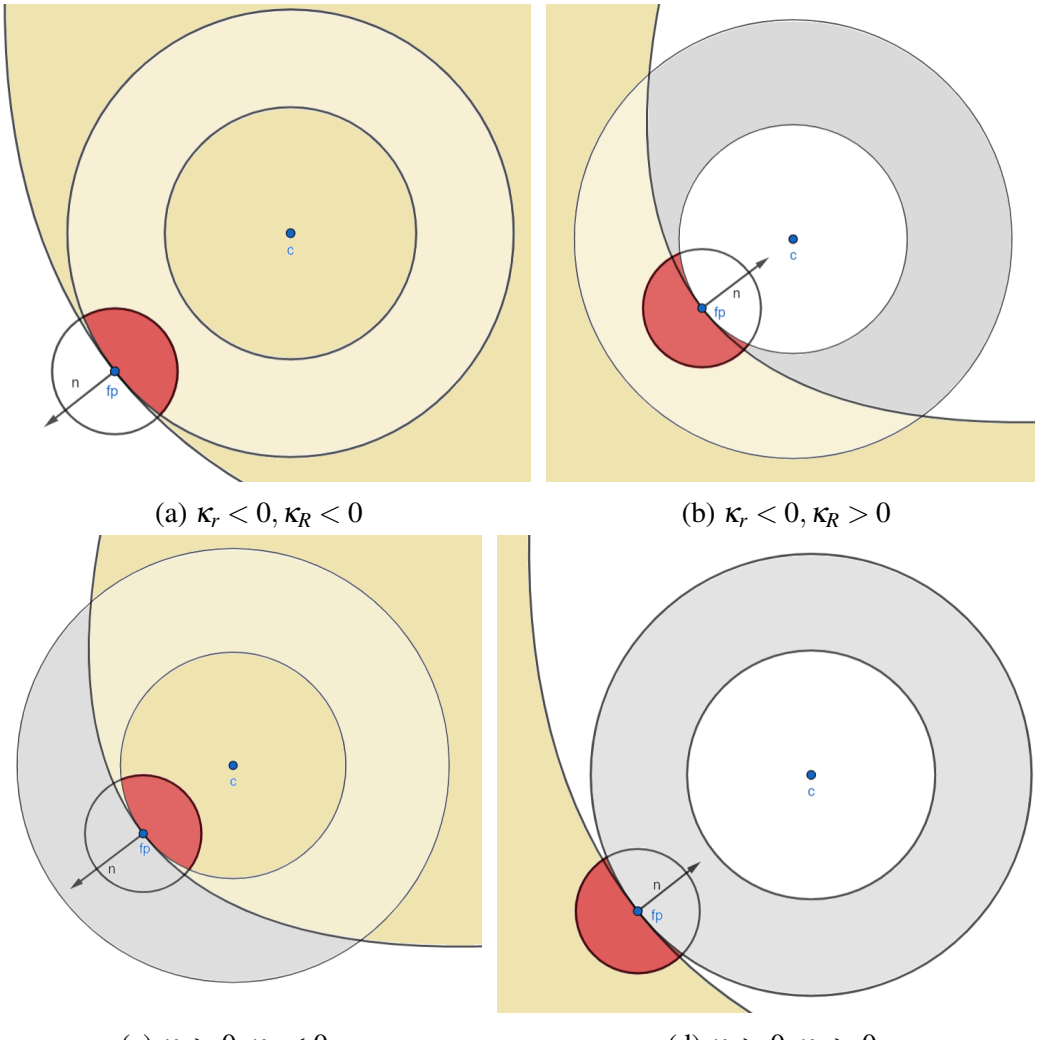
2.25. ábra. A hibás és a korlátozott hengertől(piros) vett javított távolság

Így $\kappa_R = 0$ esetben a távolság $d = \begin{cases} max(d_{henger}, d_{gomb}) & \text{, ha } \kappa_r < 0 \\ max(-d_{henger}, d_{gomb}) & \text{, ha } \kappa_r > 0 \end{cases}$

Tórusz

A $\kappa_r > 0$, $\kappa_R > 0$ esetben használható a tórusz távolságfüggvénye, de a hengernél látotthoz hasonlóan itt is több esetet kell elkülöníteni a görbületek előjele szerint. Ennek oka, hogy a tórusz a legtöbb esetben kívülről érinti vagy körbeöleli a felületet. Ezekben az esetekben a tórusznak csak egy "oldala" közelíti a felületet, a többi részét nem kéne figyelembe venni.

A görbületek előjelei alapján a 2.26. ábrán látható esetek lehetségesek a közelítő tóruszra:



(a) $\kappa_r < 0, \kappa_R < 0$ (b) $\kappa_r < 0, \kappa_R > 0$

(c) $\kappa_r > 0, \kappa_R < 0$ (d) $\kappa_r > 0, \kappa_R > 0$

2.26. ábra. A tórusz lehetséges elhelyezkedései a felülethez (sárga) képest és a valódi közelítés (piros)

- Az (a) esetben a tórusztól vett távolság megfelelő:

$$d = d_{torusz}$$

- A (b) esetben a tórusz egy része a talppont és a pozíció közé esik, ezt viszont nem kéne figyelembe venni. Erre megoldás a korlátozó gömb használata.

$$d = max(d_{torusz}, d_{gomb})$$

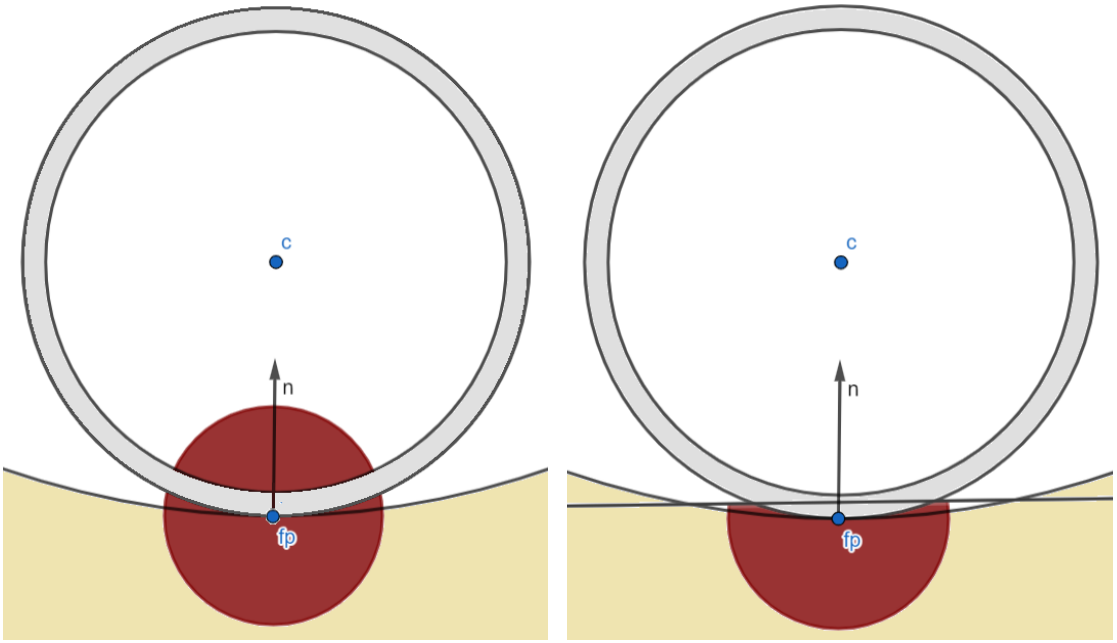
- A (c) és (d) esetben a tórusz kívülről simul hozzá a felülethez, ezért valójában a komplementere adja meg a kívánt közelítést. A komplementer az egész tóruszon kívüli teret magában foglalja, ezért ismét csak le kell korlátozni a talppont körül egy gömbbel.

$$d = max(-d_{torusz}, d_{gomb})$$

  Megjegyzés. A távolságfüggvénnyel megadott geometria komplementerét a távolságfüggvény negáltja adja meg. Ez a függvény a Tételben kifejtett távolságfüggvény tulajdonságot megtartja, de a geometrián belül lesz pozitív, kívül negatív, így pont a komplementert adja meg.

A hengernél és tórusznál is előfordulhat, hogy a korlátozó gömb sugara nagyobb mint a tórusz r sugarának kétszerese, ilyenkor a komplementerrel való metszés egy hibás térrészt is magában foglalhat. (2.27 (a)).

Ezt úgy küszöböltem ki, hogy ezekben az esetekben a gömbbel metszés előtt egy síkkal is elmetszem a geometriát, amelyet $\boldsymbol{n}$ és $\boldsymbol{fp'} = \boldsymbol{fp} + \varepsilon \cdot \boldsymbol{n}$ határoz meg. Így a talppont és a $\boldsymbol{p}$ közötti térrésztől megszabadulok.



(a) A fellépő hiba (b) A félsíkkal metszett javított változat

2.27. ábra. A korlátozó gömbhöz képest túl kicsi sugarú tórusz esete

Összegezés

Összegezve a tórusz kirajzolásának esetei:

- $\kappa_r = 0$ és $\kappa_R = 0$

$$d = max(d_{sik}, d_{gomb}) \tag{2.28}$$

- $\kappa_R = 0$

$$d = \begin{cases} max(d_{henger}, d_{gomb}) & \text{, ha } \kappa_r < 0 \\ max(-d_{henger}, d_{gomb}) & \text{, ha } \kappa_r > 0 \end{cases} \tag{2.29}$$

- $\kappa_r < 0$

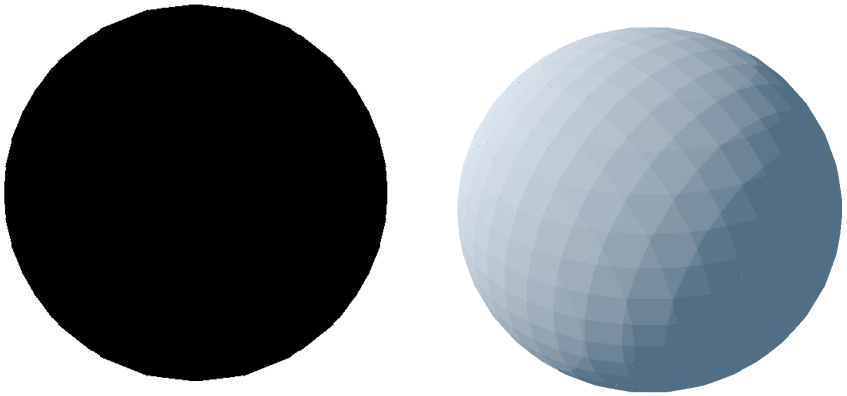
$$d = max(d_{torusz}, d_{gomb}) \tag{2.30}$$

- $\kappa_r > 0$

$$d = max(-d_{torusz}, d_{gomb}) \tag{2.31}$$

2.8. Mező szűrése általános módszerekkel

A mezőt kirajzolva nem biztos hogy folytonos felületet kapunk (2.28. ábra). Ennek oka lehet az, hogy kicsi a mező felbontása a felület részletességéhez képest, vagy az illesztett geometriák nem folytonosan deriválhatóan kapcsolódnak egymáshoz.



2.28. ábra. Példa nem C^1 folytonosan kapcsolódó mezőkre

Ezen problémák kiküszöbölhetőek egy alkalmas szűrési módszerrel. A szűrés lényege, hogy a mező több celláját veszünk adatot és ezeket valahogy összekombináljuk, hogy pontosabb approximációt kapjunk. A legegyszerűbb módszer a sima interpoláció.

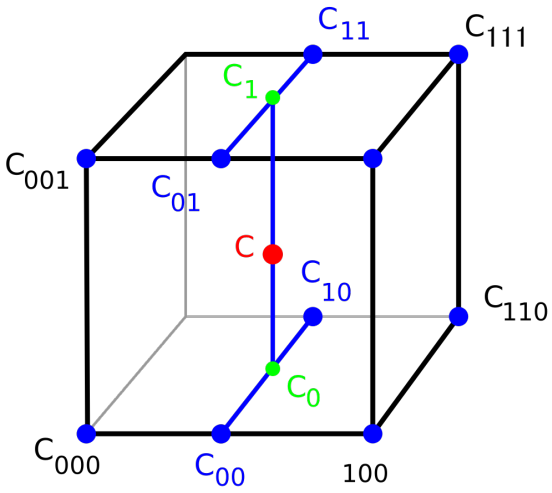
2.8.1. Trilineáris interpoláció

Jó szűrési módszer a trilineáris interpoláció. A pixel pozíciójához először megkeresem a nyolc szomszédos texelt, majd az ezekben található adatokat interpolálom.

$$\boldsymbol{tex}(\boldsymbol{p}) = \lfloor (\frac{\boldsymbol{p}}{box} + 0.5) * (res - 1) \rfloor + \boldsymbol{c} \tag{2.32}$$

ahol, $\boldsymbol{c} \in \{0,1\} \times \{0,1\} \times \{0,1\}$

Az itt kiértékelt d értékeket a 2.29. ábrán látható módon kombinálom. t azt adja meg, hogy a texelek középpontjait összekötő kockán belül hol van $\boldsymbol{p}$.



2.29. ábra. A trilineáris interpoláció számítása

$$d_{12} = (1-t_x)d_1 + t_x d_2 \quad d_{34} = (1-t_x)d_3 + t_x d_4$$
$$d_{56} = (1-t_x)d_5 + t_x d_6 \quad d_{78} = (1-t_x)d_7 + t_x d_8$$
$$d_{1234} = (1-t_y)d_{12} + t_y d_{34}$$
$$d_{5678} = (1-t_y)d_{56} + t_y d_{78}$$
$$d = (1-t_z)d_{1234} + t_z d_{5678}$$

A trilineáris interpolációval C^0 folytonos eredményt kapunk, de a magasabb rendű folytonossághoz a módszer egy módosított változata szükséges. A végső eredményhez ötödfokú Hermite interpolációt [7] használtam, ami biztosítja a C^1 folytonosságot.

Ekkor t paraméter helyett $h_5(t)$ alapján interpoláltam a fent leírt módszerrel.

$$h_5(t) = 6t^5 - 15t^4 + 10t^3$$

2.9. Tesztelés és mérések

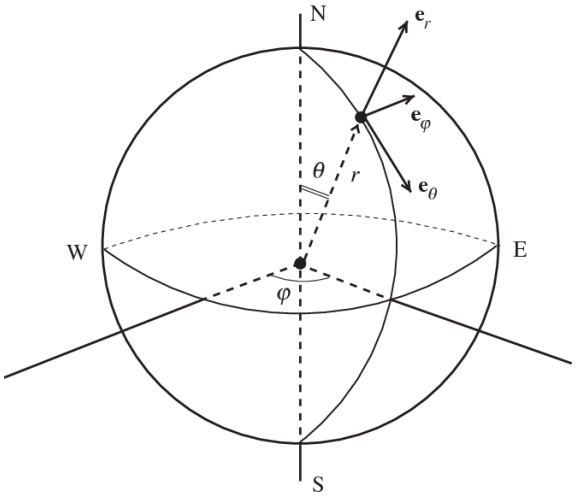
A geometriai mezőket NVIDIA Falcor[8] keretrendszerben implementáltam C++ nyelven.

A kirajzolt eredmény vizsgálatához hasznos ha összehasonlíthatóak az egyes mezőkkel kapott eredmények, a következő részben az ehhez kidolgozott metrikát mutatom be.

2.9.1. Pontosság mérése

A pontosság méréséhez kidolgoztam egy módszert, amellyel a mező által adott eredményt és az eredi SDF-et össze tudom mérni.

Ehhez először egy origó körüli 1 sugarú gömbről veszek pontokat. Feltehetjük, hogy minden geometria ezen a gömbön belül van. A pontos mérés érdekében a gömbről egyenletesen kell venni N darab pontot, ehhez generálok N darab 0 és 1 közötti random számpárt $(rand_1, rand_2)$. Ezeket gömbi koordinátákként értelmezve kapok egy random pontot a gömbön, a 2.30. ábra alapján.



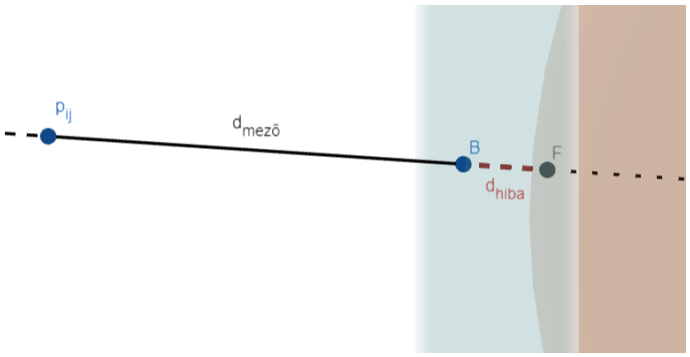
2.30. ábra. Gömbi koordináták megadása

$$\theta = rand_1 \cdot \pi \quad \phi = rand_2 \cdot 2\pi$$

Ez még nem lesz egyenletes, mivel például a gömb tetején $\theta = 0$ foknál, minden ϕ szögre a gömb legfelső pontját kapjuk, míg $\theta = \frac{\pi}{2}$ esetben ezek különböző pontot jelentenek. Tehát a gömb tetejéről és aljáról nagyobb valószínűséggel kapunk pontot. Erre megoldásként átskáláztam a szögeket [9] alapán.

$$\phi = L\theta, \quad \text{ahol } L = \sqrt{N\pi}$$

Az így kapott N darab pontból egy-egy sugarat indítottam a gömb középpontja fele. A sugár mentén sphere tracing algoritmussal lépkedtem, amíg ki nem lépek a gömbből, vagy el nem találom a mező által tárolt felületet. Amikor eltaláltam a felületet, a felületi pontban kiértékeltem az eredeti sdf-et és a két érték különbségét vettem a hibának (2.31. ábra).



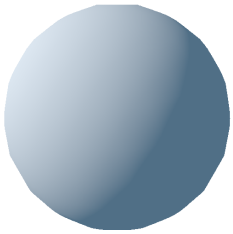
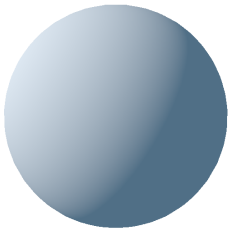
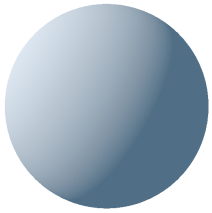
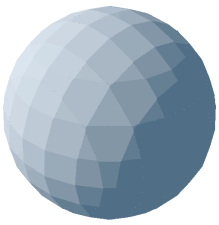
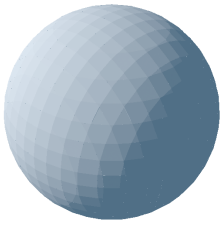
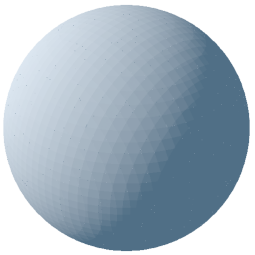
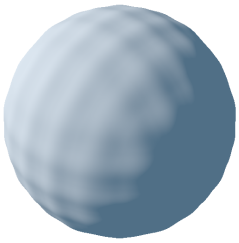
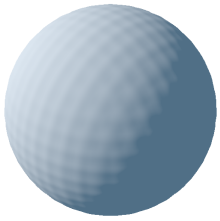
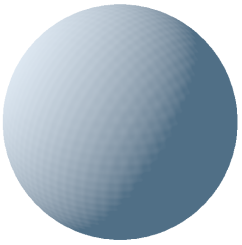
2.31. ábra. A hibaszámítás ábrázolása
(a kék geometria a mezőben tárolt, a sárga az eredeti)

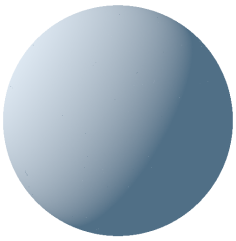
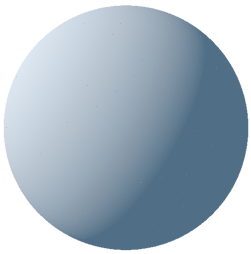
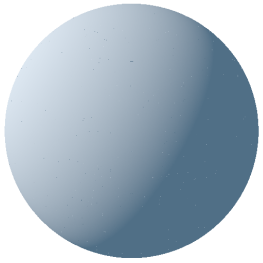
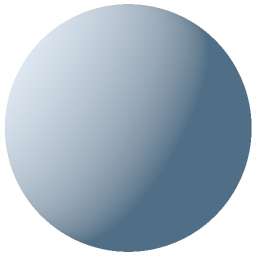
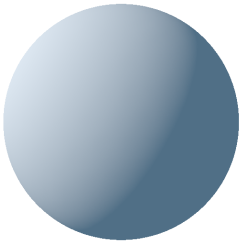
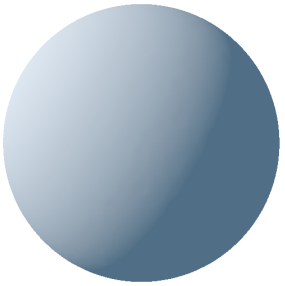
Tehát a hibamérés minden tesztpozícióhoz megadta, hogy az eredeti felülethez képest mennyire van „elcsúszva” a közelítés. A hibákból a maximumot és az átlagot számoltam ki, ezt hasonlítottam össze a különböző mezőkre és szűrésekre.

2.10. Eredmények

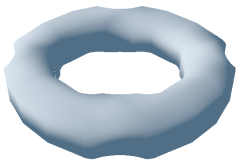
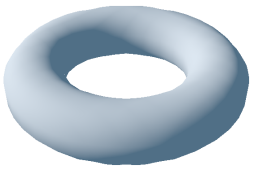
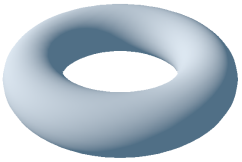
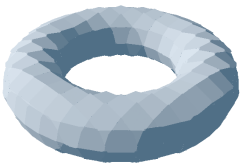
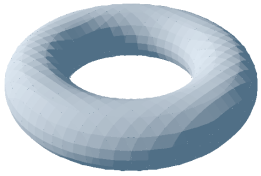
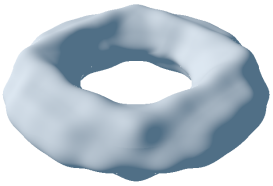
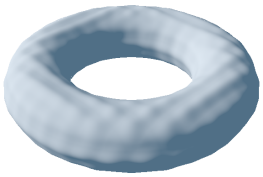
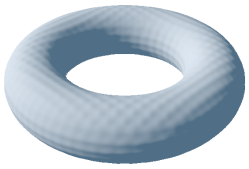
Minden mezőt 16, 32 és 64-es felbontáson értékeltem ki, három SDF-re. A hibamérés-sel kapott eredményeket is összehasonlítottam. A kapott eredmények, interpoláció nélkül és trilineáris interpolációval:

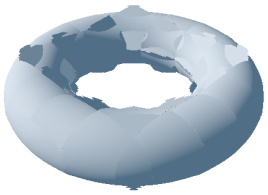
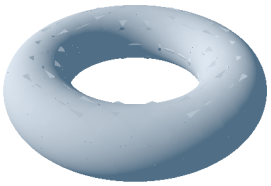
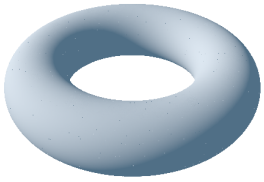
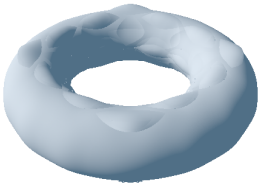
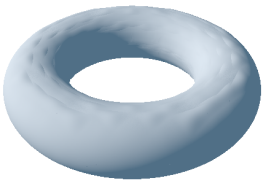
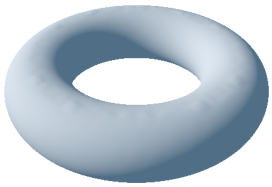
Gömb

Mező		16	32	64
G0	trilineári			
Max hiba		0.008713	0.002313	0.000682
Átlag hiba		0.005532	0.001372	0.000340
G1	nincs szűrés			
Max hiba		0.228969	0.115983	0.058380
Átlag hiba		0.001056	0.000247	0.000095
G1	trilineáris			
Max hiba		0.004507	0.001061	0.000361
Átlag hiba		0.001845	0.000408	0.00098

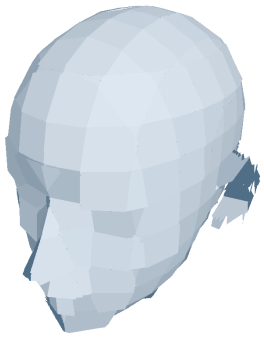
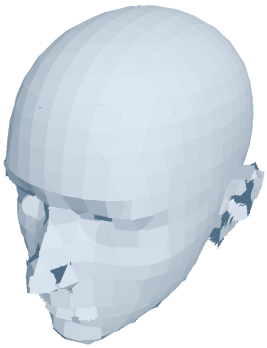
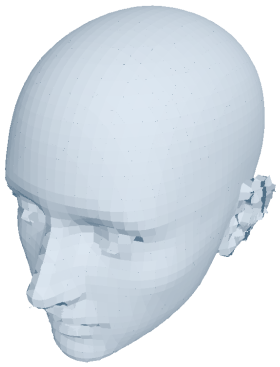
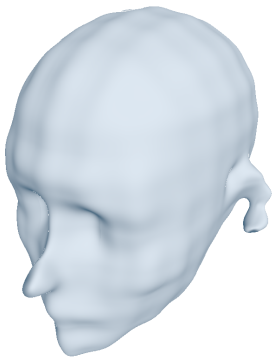
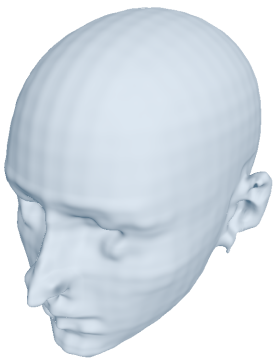
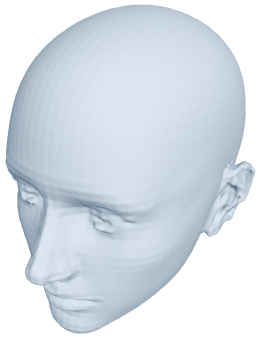
Mező		16	32	64
G2	nincs szűrés			
Max hiba		0.000250	0.000359	0.000329
Átlag hiba		0.000039	0.000041	0.000028
G2	trilineáris			
Max hiba		0.000251	0.000225	0.000322
Átlag hiba		0.000043	0.000031	0.000025

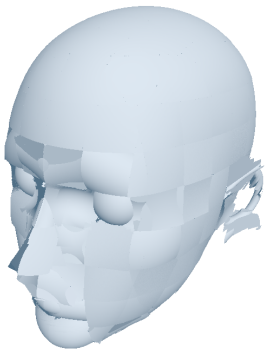
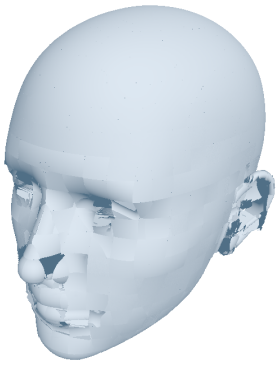
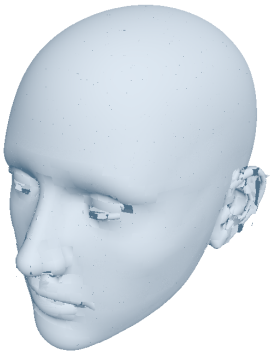
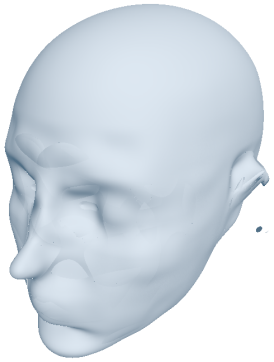
Tórusz

Mező		16	32	64
G0	trilineáris			
Max hiba		0.909611	1.012275	0.943484
Átlag hiba		0.003833	0.000882	0.000245
G1	nincs szűrés			
Max hiba		0.035414	0.709030	0.935121
Átlag hiba		0.001920	0.000372	0.000118
G1	trilineáris			
Max hiba		0.639395	1.034478	1.054804
Átlag hiba		0.002265	0.000635	0.000208

Mező		16	32	64
G2	nincs			
Max hiba		0.826222	0.947209	0.638373
Átlag hiba		0.000438	0.000089	0.000072
G2	trilineáris			
Max hiba		1.030288	0.998169	0.978266
Átlag hiba		0.000400	0.000139	0.00080

Fej

Mező		16	32	64
G0	trilineáris			
Max hiba		0.037631	0.037063	0.042633
Átlag hiba		0.005641	0.002332	0.001379
G1	nincs szűrés			
Max hiba		0.149461	0.101852	0.060539
Átlag hiba		0.002544	0.000944	0.001026
G1	trilineáris			
Max hiba		0.083851	0.054982	0.059297
Átlag hiba		0.003747	0.001130	0.000826

Mező		16	32	64
G2	nincs szűrés			
Max hiba		0.083486	0.095642	0.096772
Átlag hiba		0.002316	0.002336	0.003171
G2	trilineáris			
Max hiba		0.067336	0.089740	0.087617
Átlag hiba		0.001707	0.001575	0.002117

A mérésekből az látható, hogy a mező rendjét növelve egyre jobb eredményt kapunk, viszont nagyobb felbontáson a maximális hiba egyre nagyobb. Ennek az elsődleges oka az lehet, hogy a számítás során sok helyen bejöhetnek numerikus hibák, így nagy felbontáson a hibák miatt romlik a mező pontossága.

A trilineáris interpoláció sem javít minden esetben a mező tartalmán, ezért érdemes lehet térben is kidolgozni a CSG szűréshez hasonló módszert, ami felhasználja a geometriák tulajdonságait.

A mező pontossága mellett a generálás sebessége is fontos tényező, ezért sebességmérést is végeztem. A mérést NVIDIA GeForce GTX 1650 GPU-n végeztem. A méréshez a az előbbiekben látott 3 geometriát használtam és minden mezőt 3 különböző felbontáson mértem.

Gömb

Mező	16	32	64
G0	0.03 ms	0.03 ms	0.03 ms
G1	0.04 ms	0.04 ms	0.04 ms
G2	1.80 ms	1.78 ms	1.78 ms

Tórusz

Mező	16	32	64
G0	0.04 ms	0.03 ms	0.03 ms
G1	0.04 ms	0.04 ms	0.04 ms
G2	1.94 ms	1.94 ms	1.95 ms

Fej

Mező	16	32	64
G0	2.21 ms	2.22 ms	2.22 ms
G1	9.80 ms	9.81 ms	9.90 ms
G2	27.27 ms	26.75 ms	27.08 ms

A várakozásnak megfelelően a magasabb rendű mezők már kisebb felbontáson adnak kellően pontos közelítést. A másodrendű mező, cellánként csak 2 float értékkel tárol több adatot mint az elsőrendű és a generálása se jelentősen lassabb, így alacsony felbontáson hatékonyabb.

3. fejezet

CSG szűrés

Dolgozatom második felében egy a geometriai mezőkhöz jobban illeszkedő szűrési algoritmust fogok bemutatni, amelyet a síkbeli elsőrendű mezőhöz dolgoztam ki.

Először röviden bemutatom a síkbeli távolságmezőket és generálásukat, majd kitérek az új szűrési módszerre.

3.1. Síkbeli mezők generálása

A síkbeli mezők generálása a térbeli esettel analóg módon történik. A mezőt egy 2 dimenziós tömbbel reprezentálom, amelyeknek minden cellájában egy közelítő geometriát tárolok.

A távolságmezőt a ugyanúgy a felbontás ($res \in \mathbb{N}$) és a befoglaló doboz mérete ($box \in \mathbb{R}$) határozza meg. Feltehetjük, hogy minden objektum a $[-1,1]^2$ négyzetben helyezkedik el, ezért a *box* értéket alapártelmezetten 2-nek veszem, az ábrák és mérések ezzel az értékkel készültek.

A mező generálásához először meghatározom a térbeli pozíciót, ahonnan mintavételezni és közelíteni akarom az SDF-et. Ez a fent leírt felbontástól ($res \in \mathbb{N}$) és a befoglaló doboz oldalhosszától ($box \in \mathbb{R}$) függ.

3.1.1. Mintavételezési pozíció

A mintavételezési pozíció a *box* oldalhosszú origó középpontú, $res \times res$ felbontású négyzet i. sorában j. oszlopában található cella középpontja.

$\forall i \in \{0 \dots res-1\} : \forall j \in \{0 \dots res-1\} :$

$$\boldsymbol{grid}(i,j) = -\frac{1}{2}\begin{bmatrix} box \\ box \end{bmatrix} + \begin{bmatrix} i+0.5 \\ j+0.5 \end{bmatrix} \cdot \frac{box}{res} \tag{3.1}$$

3.1.2. A mező rendje

A térbeli esethez hasonlóan itt is minden $\boldsymbol{grid}(i,j)$ mintavételezési pozícióhoz meg kell keresni a geometria legközelebbi pontját a talppontot ($\boldsymbol{fp}$). Ez a 2.4-ben leírt módszerrel megtehető.

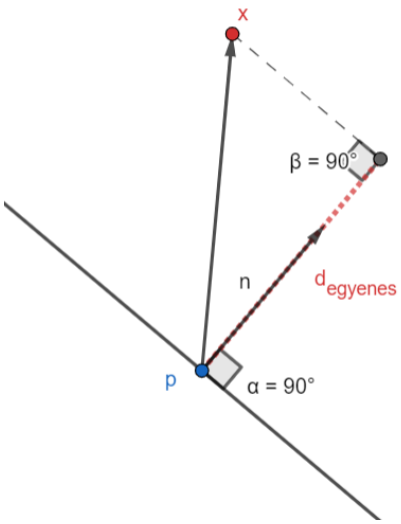
Ezután minden talppontban illesztek egy geometriát, ami adott rendben megfelelően közelíti az objektumot a talppont kis környezetében. A mező rendje szerinti elvárt adatok a térbeli esetnek megfelelőek: Az elsőrendű mezőnek a talppontot és a talppontban vett normálvektort kell visszaadnia, a másodrendű mezőnek pedig ezeken kívül a görbületet is tárolni kell, ami síkban a talppontban kiszámolt görbületi értéket jelenti.

3.2. Helyettesítő geometria

A 2 dimenziós elsőrendű mező helyettesítő geometriájának a talppontot és az ott vett normálvektort kell rekonstruálnia. A legegyszerűbb ennek megfelelő görbe a talppontban vett érintőegyenes, vagy ha a felület proxygeometriáját tekintjük a félsík.

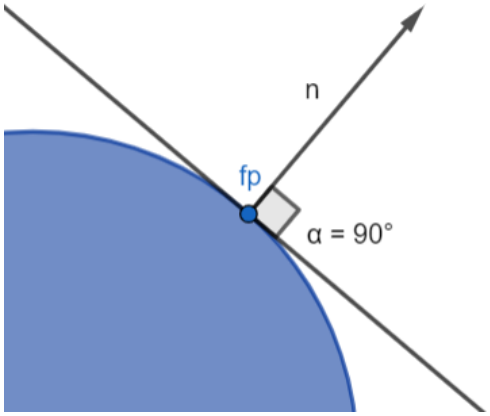
Az egyenes megadásához elég az egyenes egy pontját ($\boldsymbol{p} \in \mathbb{R}^2$) és a normálvektorát ($\boldsymbol{n} \in \mathbb{R}^2$) ismerni. A $\boldsymbol{p}$-ből $\boldsymbol{x} \in \mathbb{R}^2$-be mutató vektort a normálvektorra vetítve megkapjuk a pont távolságát az egyenestől, a 3.1. ábrán látható módon. Így az $\boldsymbol{x}$ távolsága:

$$d_{egyenes} = \langle \boldsymbol{x} - \boldsymbol{p}, \boldsymbol{n} \rangle \tag{3.2}$$



3.1. ábra. Az egyenestől vett távolság számítása

Az $\boldsymbol{fp}$ talppontban vett érintőegyenest definiáló pont maga a talppont, a normálvektor pedig az f távolságfüggvény talppontban vett gradiensének irányába mutató egységhosszú vektor $\boldsymbol{n} = \frac{\nabla f(\boldsymbol{fp})}{\|\nabla f(\boldsymbol{fp})\|}$ (3.2. ábra). Így az elsőrendű mező cellánként 4 valós számmal tárolható.

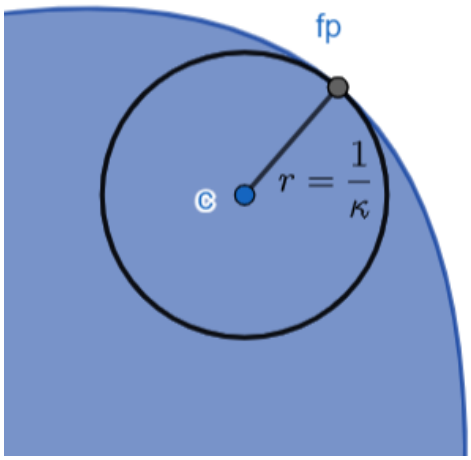


3.2. ábra. Az elsőrendű mezőben tárolt érintőegyenes

A másodrendű mezőnek a talppontban vett görbületet is vissza kell adnia. Erre megfelelő geometria a talppontban vett simulókör [1]. (3.3. ábra)

14. Definíció. Tekintsük az $x(t-\triangle t), x(t), x(t+\triangle t)$ pontok által meghatározott körsereget. A $\triangle t \to 0$ határértékben kapott kör a görbe $x(t)$-beli simulóköre.

4. Tétel. *Egy reguláris görbe $\boldsymbol{x}(t)$ pontjában a görbe simulókörének sugara* $\frac{1}{\kappa}$



3.3. ábra. A másodrendű mezőben tárolt simulókör

3.3. Síkbeli elsőrendű mező megjelenítése

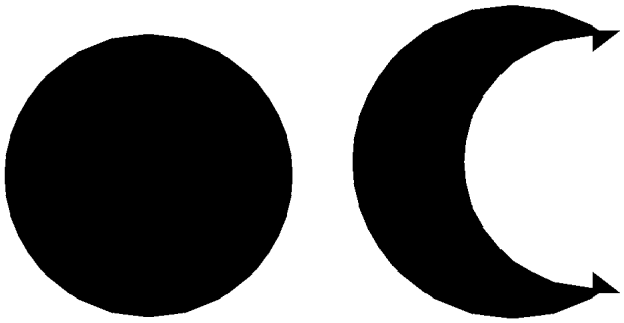
A mezőben az előzőekben leírt érintőegyenes van eltárolva, olyan paraméterekkel, amelyekből könnyen rekonstruálható a távolságfüggvénye. A képernyő minden pixeléhez kiolvasom a mezőből a megfelelő adatot, majd a távolságfüggvény kiértékelése alapján megjelenítem az eredményt.

Síkban a megjelenítés nagyon egyszerű. Veszem a képernyő adott pixelének pozícióját ($\boldsymbol{px} \in [-1,1]^2$), ezt még át kell skálázni a képaránynak (ar) megfelelően: $\boldsymbol{p} = \begin{bmatrix} \boldsymbol{px}_x \\ \boldsymbol{px}_y \cdot ar \end{bmatrix}$ ezzel megvan a pozíció a síkban, amiből a (3.3) képlettel kiszámolható a textúra indexe ($\boldsymbol{tex}$).

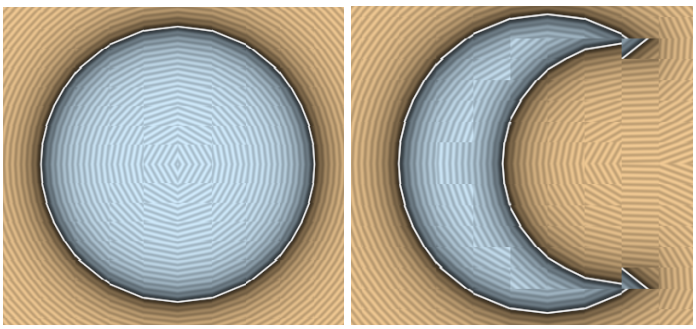
$$\boldsymbol{tex}(\boldsymbol{p}) = \lfloor (\frac{\boldsymbol{p}}{box} + 0.5) \cdot (res - 1) \rfloor \tag{3.3}$$

ahol $\boldsymbol{p} \in \mathbb{R}^2$.

A $\boldsymbol{tex}$ helyről kiolvasott adatok leírják a közelítő egyenest és a távolságfüggvényt használva megkapom a távolságot. Ha ez a távolság kisebb mint 0, akkor ez azt jelenti, hogy belül vagyok a közelítő geometrián, így a pixel színe legyen fekete, különben fehér. A kiértékelt mezőre egy példa látható a 3.4. ábrán.



(a) A kiértékelt mező



(b) A mező tartalma színezve [10] alapján

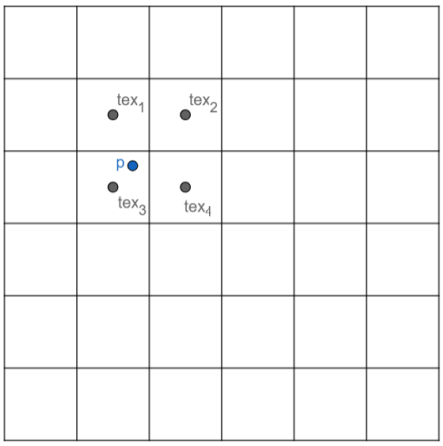
3.4. ábra. Példa kiértékelt távolságmezőre

3.3.1. Bilineáris interpoláció

Két dimenzióban jó szűrési módszer a bilineáris interpoláció. A pixel pozíciójához először megkeresem a négy szomszédos texelt (3.5. ábra), majd az ezekben található adatokat interpolálom.

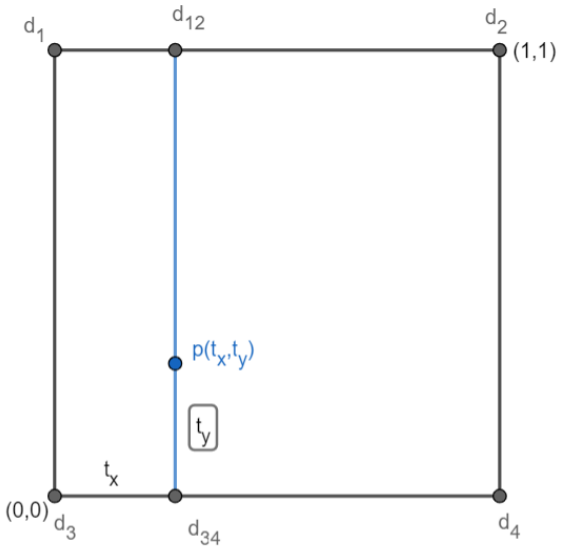
$$\boldsymbol{tex}(\boldsymbol{p}) = \lfloor (\frac{\boldsymbol{p}}{box} + 0.5) \cdot (res - 1) \rfloor + \boldsymbol{c} \tag{3.4}$$

ahol, $\boldsymbol{c} \in \{0,1\} \times \{0,1\}$



3.5. ábra. A ***p*** pont számításához szükséges texelek

Fontos, hogy a textúrában itt paraméterek vannak, ezek interpolációja nem adna helyes eredményt, ezért az előzőekben leírt módon kiszámolt d távolságértékeket kell interpolálni, a 3.6. ábrán szereplő ***t*** paraméter alapján.



3.6. ábra. A bilineáris interpoláció számítása

A négy geometriától vett távolság: d_1, d_2, d_3, d_4

$$d_{12} = (1 - t_x)d_1 + t_x d_2$$

$$d_{34} = (1 - t_x)d_3 + t_x d_3$$

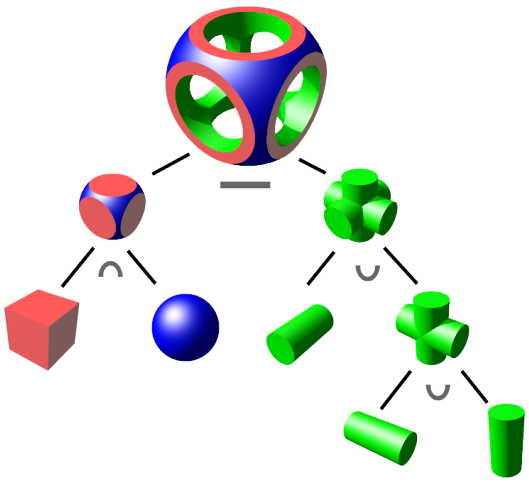
$$d = (1 - t_y)d_{34} + t_y d_{12}$$

3.4. CSG szűrés

Az előbb leírt szűrési módszer jól működik, de nem használja ki, hogy a mező geometriákat tárol. A mezőhöz jobban illeszkedne egy olyan megoldás, amely geometriai műveleteket használ a pontosabb közelítés előállításához a tárolt objektumokból.

Ehhez dolgoztam ki a 2 dimenziós első rendű mezőhöz a CSG szűrést, amely a 4 szomszédos texelben tárolt geometriákból felépít egy CSG fát, amelynek kiértékelésével megkapunk egy új geometriát, ami jobban közelíti a felületet. A kirajzoláshoz az ettől vett távolságot használjuk.

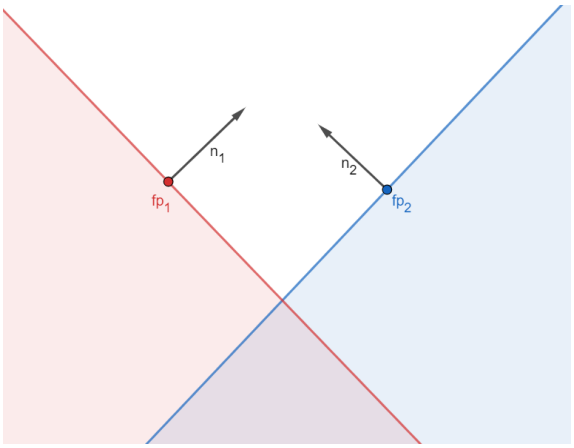
15. Definíció. A CSG fa egy olyan fa gráf, amelynek leveleiben egyszerű primitívek találhatóak, belső csúcsaiban pedig halmazműveletek vagy transzformációk. Ezek a metszet az unió és a különbség. A halmazműveletek a primitívek belsején vannak értelmezve és eredményük a két primitívből kapott halmaz. A fát lentről felfele kell kiértékelni, a gyökerében kapott geometria az eredmény. (3.7. ábra)



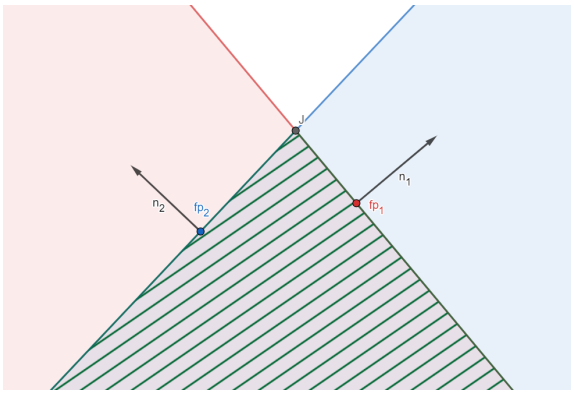
3.7. ábra. Példa CSG fára

A bilineáris interpolációhoz hasonlóan a 4 legközelebbi texelből kell kiolvasni a tárolt adatot. Ezeket a proxy geometriának megfelelően egy félsíkként értelmezem, ezeknek tudom a metszetét vagy unióját venni.

A CSG fa felépítéséhez először azt kell kitalálni, hogy két félsíkon milyen műveletet kell végezni a pontosabb közelítéshez. Ehhez a 3.8. ábrán látható eseteket különítettem el.



(a) Unió

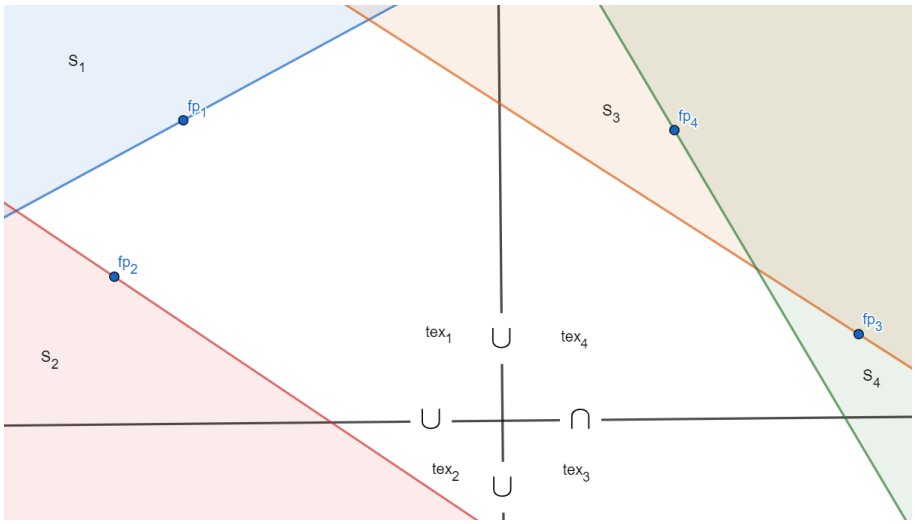


(b) Metszet

3.8. ábra. Az egyértelműen kezelhető esetek

- Metszet: ha mindkét félsíkhoz tartozó talppont a másik félsíkon belülre esik
- Unió: ha mindkét félsíkhoz tartozó talppont a másik félsíkon kívülre esik
- Minden más esetben a közelebbi texel által tárolt félsíkot használom

Ezzel két geometriát már lehet kombinálni, de a teljes CSG szűréshez a 4 legközelebbi texelben tárolt geometria kell. Ezekből a szomszédosakra az előbb leírt módon megállapítom, hogy milyen műveletet kell alkalmazni, így lesz egy négy műveletből és négy geometriából álló táblázat, amiből elő lehet állítani a CSG fát. Ez előre kiszámolható, a mező generálásával párhuzamosan, így a rajzolásnál már csak a fát kell összerakni.

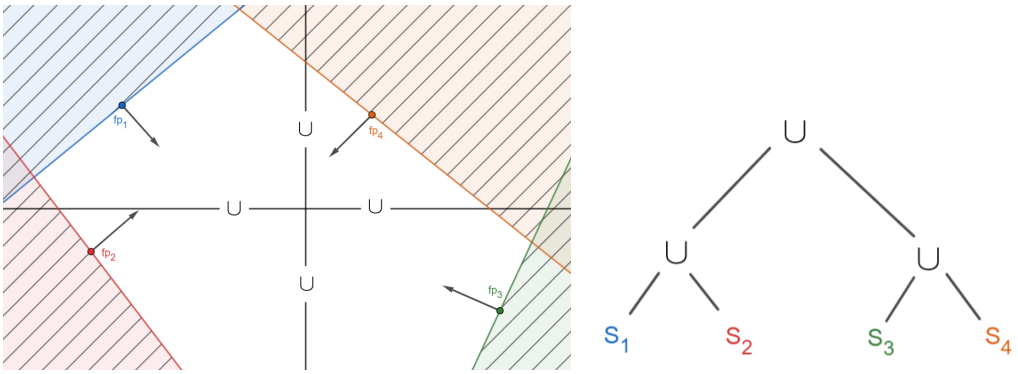


3.9. ábra. Példa a CSG műveletek meghatározására

A 3.9. példán látható a 4 művelet meghatározása. Minden t_i texel tartalmaz egy S_i félsíkot. A texelek határaira felírt műveletek a két szomszédos mezőben tárolt síkra alkalmazandó CSG műveletek, amiket az előzőekben leírt módon határoztam meg.

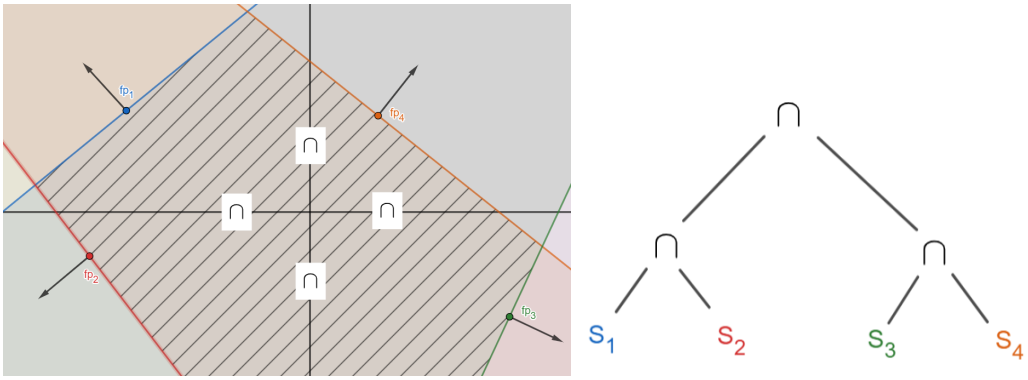
A CSG fa felírásához a metszetek és uniók száma alapján különítek el eseteket. A különböző műveletkombinációkhoz egyértelműen hozzárendelhető egy CSG fa, ezek láthatók lejjebb, mellettük példával a tárolt félsíkokra (a csíkozott rész reprezentálja a szűrt geometriát).

- 4 unió: az uniók sorrendje felcserélhető, így a CSG fa bárhogy felírható (3.10. ábra)



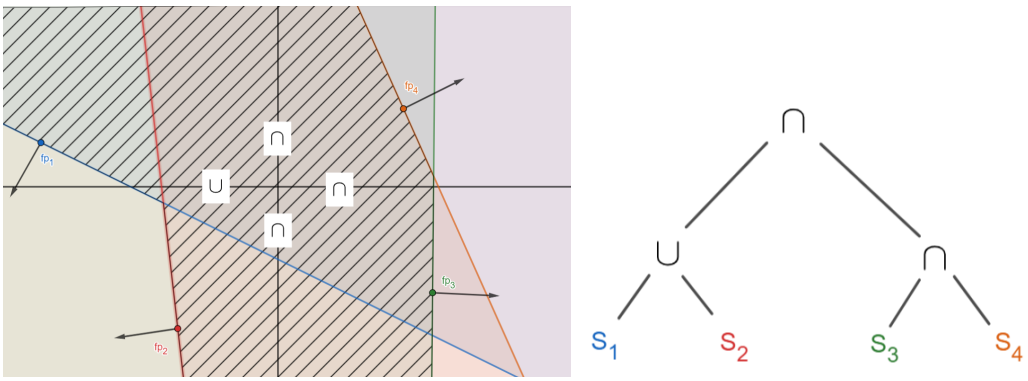
3.10. ábra. 4 unió

- 4 metszet: a metszetek sorrendje is felcserélhető, így a CSG fa bárhogy felírható (3.11. ábra)



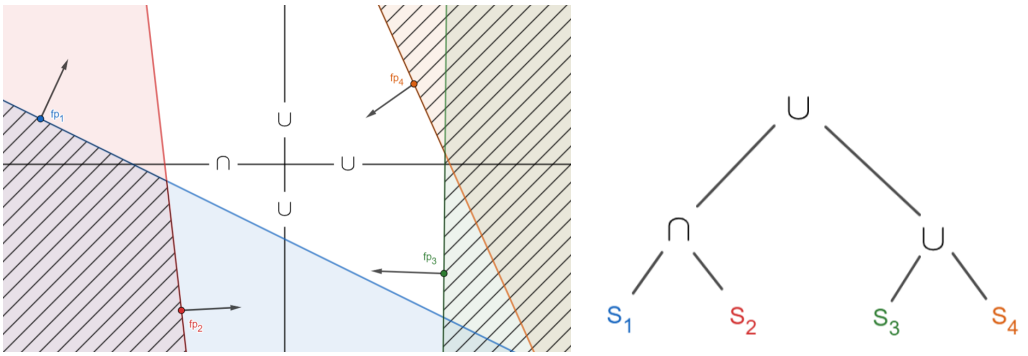
3.11. ábra. 4 metszet

- 3 metszet, 1 unió (3.12. ábra)



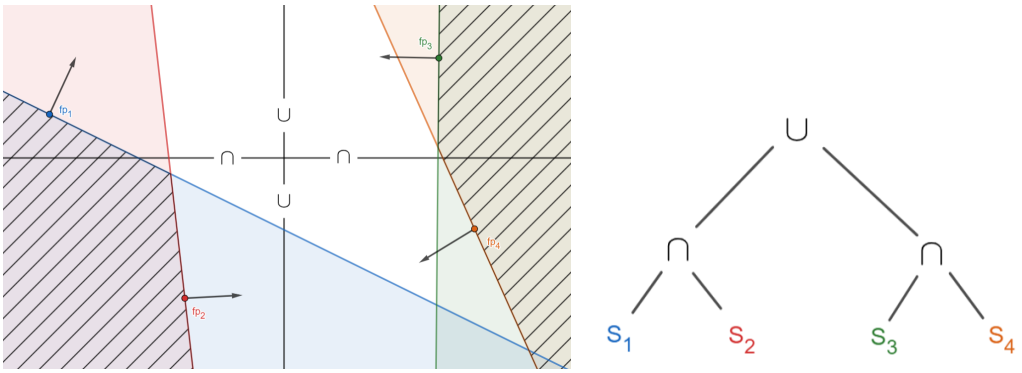
3.12. ábra. 3 metszet, 1 unió

- 3 unió, 1 metszet (3.13. ábra)

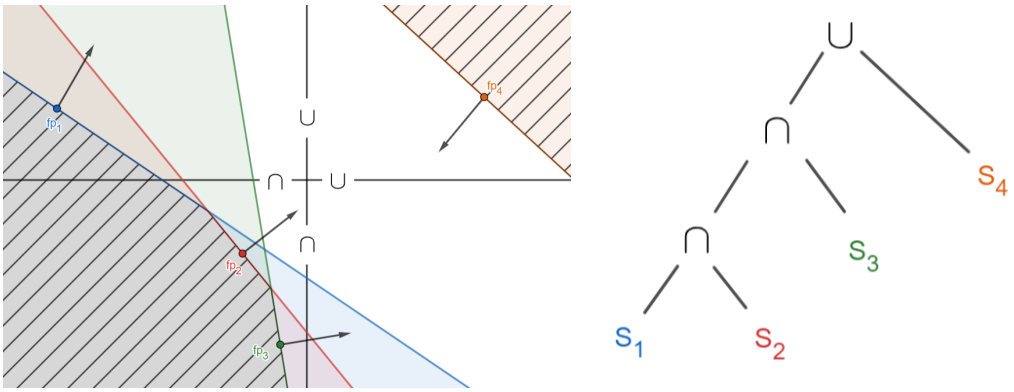


3.13. ábra. 3 unió, 1 metszet

- 2 unió, 2 metszet (3.14. és 3.15. ábra)



3.14. ábra. 2 unió, 2 metszet (1. eset)



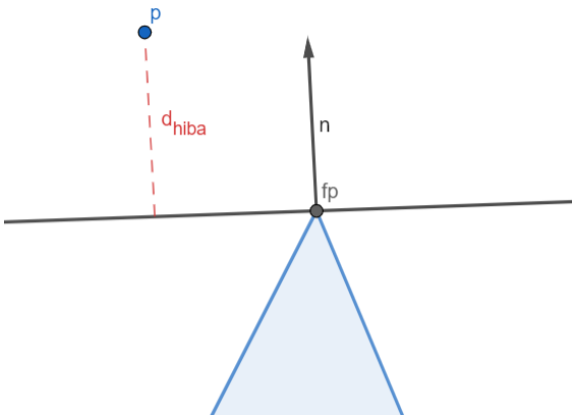
3.15. ábra. 2 unió, 2 metszet (2. eset)

Ennek a módszernek az interpolációhoz képest hátránya, hogy nagyobb felbontás kell ahhoz hogy simának tűnő eredményt kapjunk, viszont képes arra, hogy csúcsokat ábrázoljon. A bilineáris interpoláció lekerekíti a csúcsokat, míg a CSG szűrés képes éles kapcsolódásokat megjeleníteni, emellett általában pontosabb eredményt kaptam, erre látható példa a 3.16. ábrán.

(a) Bilineáris szűrés (b) CSG szűrés

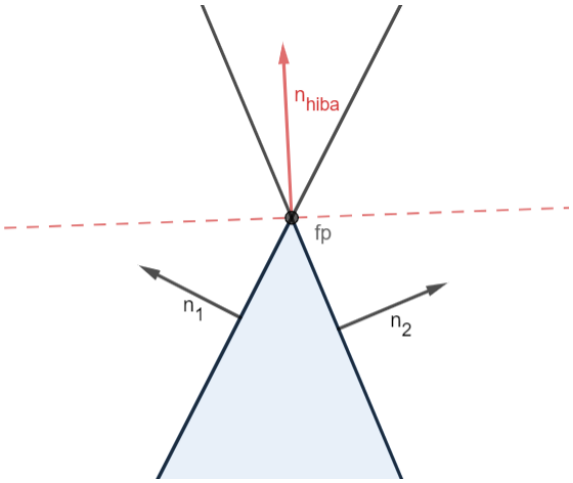
3.16. ábra. Példa a CSG szűrés eredményére

Ez a megoldás még bizonyos esetekben hibás eredményt adhat, főleg olyan alakzatoknál, amiket egyenes szakaszok határolnak. (például csillag) Itt az okozhat problémát, hogy a közelítő egyenes számolásánál a legtöbb talppont a csúcsokba fog esni, ahol a normálvektort nem lehet értelmezni, emiatt a mező alapvetőleg hibás adatot tárol. (3.17. ábra)



3.17. ábra. A csúcsban kapott hibás érintőegyenes

Ennek megoldására átalakítottam az érintőegyenes generálását, hogy azokban az esetekben amikor a csúcspontba esik a talppont, egy egyenes helyett a csúcsot két oldalról határoló egyeneseket tárolom a 3.18. ábrán látható módon.



3.18. ábra. A tárolt adat javítása (a hibás piros egyenes helyett a két fekete egyenest tárolom)

A két egyenes meghatározásához először megkeresem a talppontot, majd a talppont körül felírok egy kisebb rácsot és minden rácspontban meghatározom az SDF normálvektorát. Ezek közül kiválasztom a két leggyakrabban előforduló értéket (n_1, n_2) és ezek valamint a talppont fogják meghatározni a két egyenest.

A leggyakoribb érték megállapításához el kell dönteni, hogy mikor tekintek egyenlőnek két normálvektort. A teljes egyenlőséget vizsgálni nem jó, mivel a pozíciótól függően, vagy numerikus hibák miatt is lehetnek kisebb eltérések.

$$equal(\boldsymbol{n},\boldsymbol{m}) = \langle \boldsymbol{n},\boldsymbol{m} \rangle > cos(\varepsilon) \tag{3.5}$$

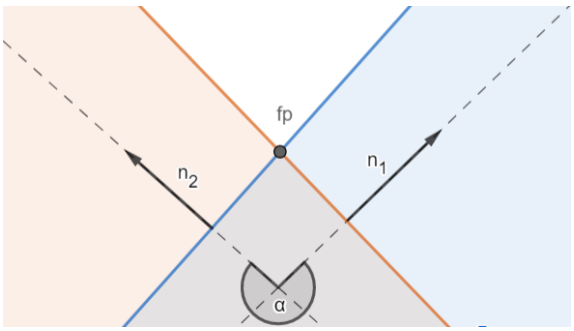
ahol $\varepsilon \in \mathbb{R}$ az egyenlőnek tekintett normálvektorok bezárt szögének maximuma, egy kellően kicsi szám..

Amennyiben az adott talppont nem egy csúcs, akkor az összes normálvektor "egyenlő" lesz, ekkor csak egy egyenes lesz tárolva, viszont a csúcsoknál kettő. A talppont minden egyeneshez azonos lesz.

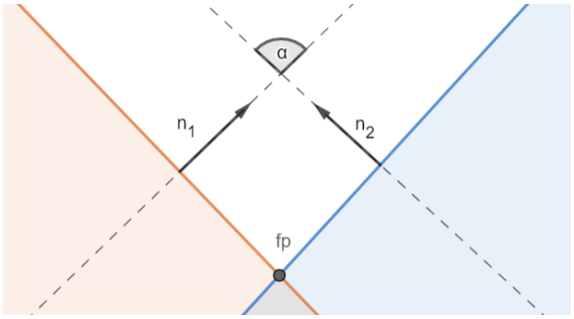
A csúcs kirajzolásnál a távolság a két egyenesből lesz kiszámítva. A csúcs lehet konvex vagy konkáv, eszerint az egyenesek által meghatározott proxy félsíkok unióját vagy metszetét kell használni.

$$d = \begin{cases} max(d_{egyenes}(\boldsymbol{fp},\boldsymbol{n}_1), d_{egyenes}(\boldsymbol{fp},\boldsymbol{n}_2)) & \text{ha } \alpha > 180° \\ min(d_{egyenes}(\boldsymbol{fp},\boldsymbol{n}_1), d_{egyenes}(\boldsymbol{fp},\boldsymbol{n}_2)) & \text{ha } \alpha < 180° \end{cases} \tag{3.6}$$

ahol α az $\boldsymbol{n}_1$ és $\boldsymbol{n}_2$ bezárt szöge. (3.19. ábra)



(a) Konvex eset



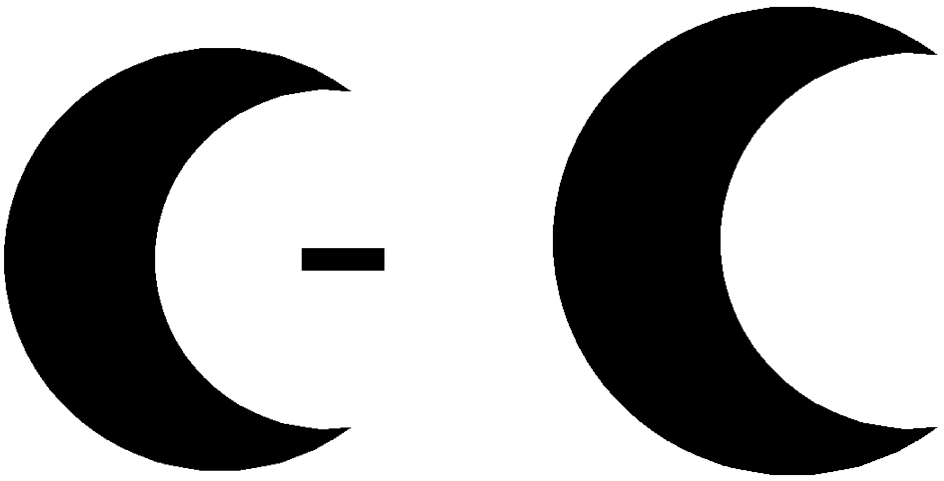
(b) Konkáv eset

3.19. ábra. A két félsík helyzete

A CSG fa elkészítésében nem lesz nagyobb változás, mivel a két objektum közötti

művelet megállapítsára ugyanazok a szabályok használhatóak. A kiértékelés itt is a műveletek darabszáma alapján történik.

A hibás és javított mezőre látható példa a 3.20. ábrán.



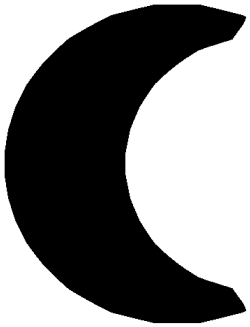
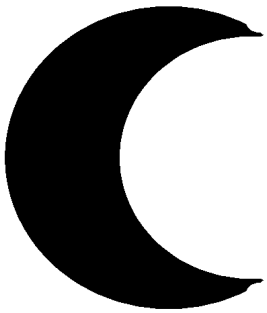
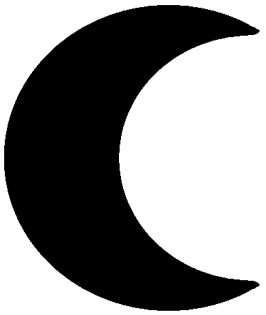
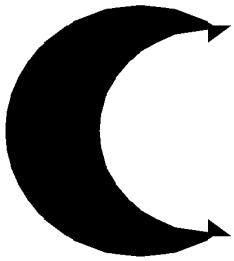
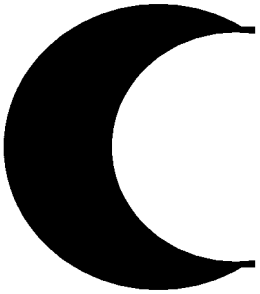
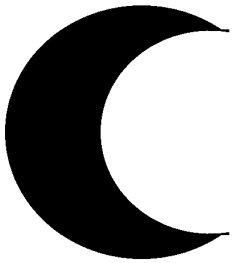
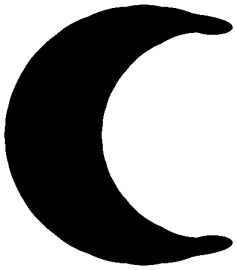
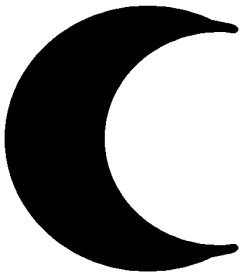
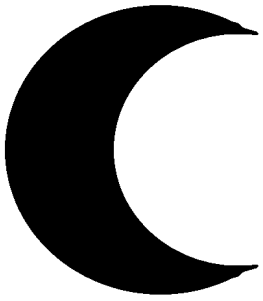
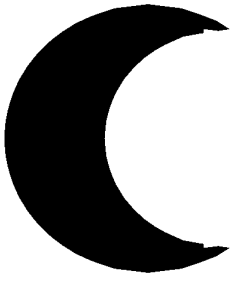
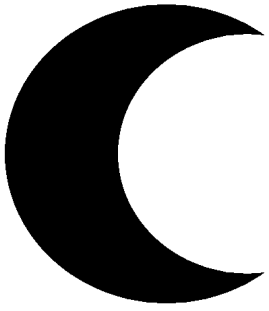
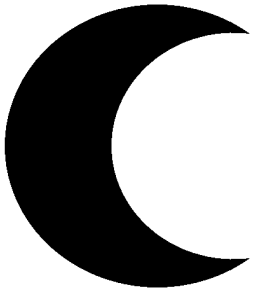
(a) A rossz normálvektor által okozott hiba (b) A javított megoldás

3.20. ábra. A csúcsban vett hibás normálvektor miatt fellépő hiba ás javítása

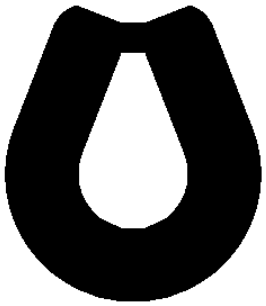
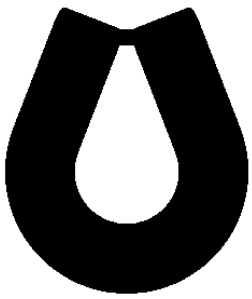
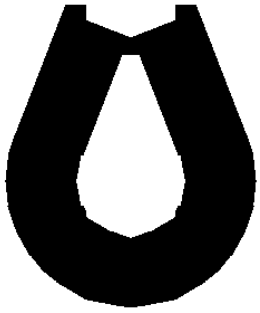
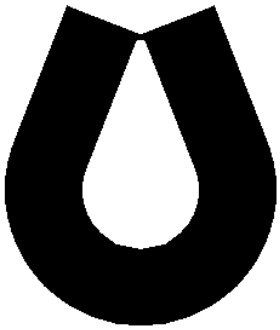
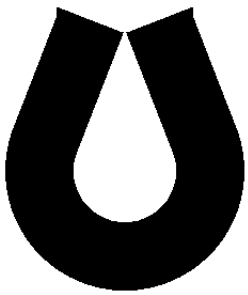
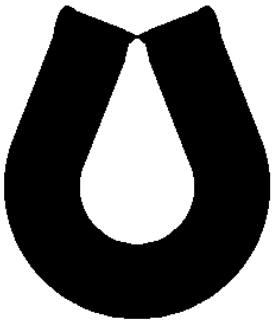
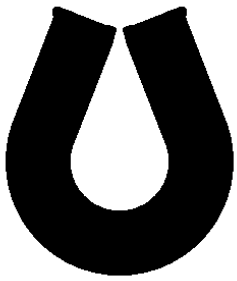
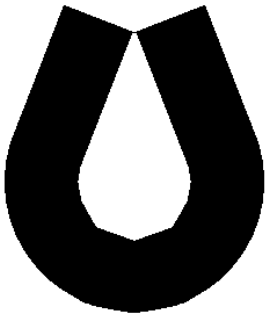
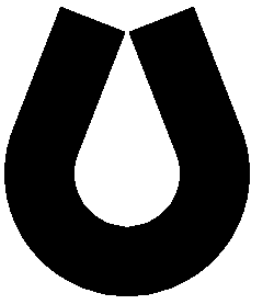
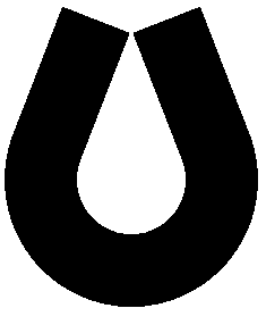
3.5. Eredmények

A mezőt 3 felbontáson értékeltem ki 3 SDF-re, különböző szűrési módszerekkel. Összehasonlításként a síkbeli nulladrendű mezőt is megjelenítettem.

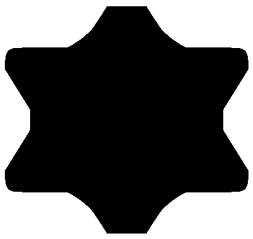
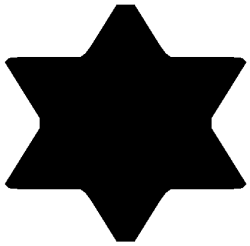
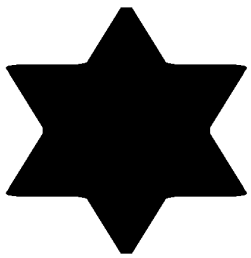
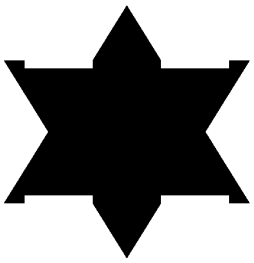
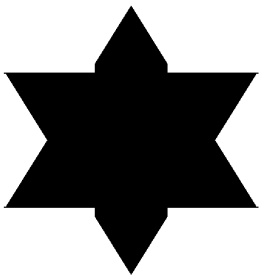
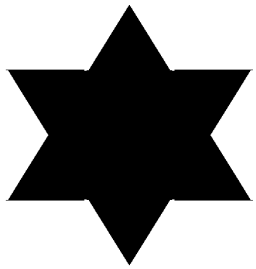
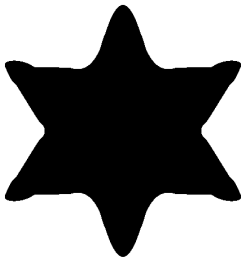
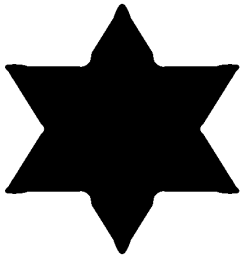
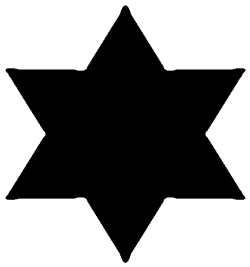
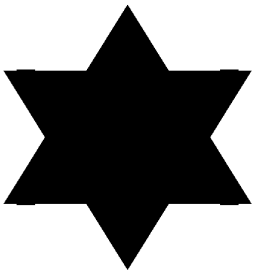
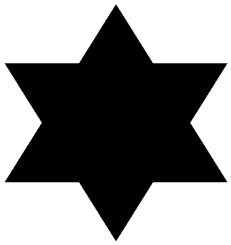
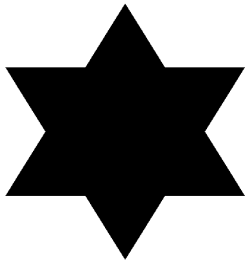
Hold

Mező		32	64	100
G0	trilineáris			
G1	nincs szűrés			
G1	trilineáris			
G1	CSG szűrés			

Patkó

Mező		32	64	100
G0	trilineáris			
G1	nincs szűrés			
G1	trilineáris			
G1	CSG szűrés			

Csillag

Mező		32	64	100
G0	trilineáris			
G1	nincs szűrés			
G1	trilineáris			
G1	CSG szűrés			

A CSG szűrés láthatóan jobb eredményt ad mint egyéb szűrési módszerek, mivel képes az éles csatlakozások megtartására. A geometriai mezőkhöz jobban is illeszkedik a többi szűrésnél, mivel kihasználja, hogy geometriát tárolunk.

4. fejezet

Összegzés

Dolgozatomban bemutattam a geometriai távolságmezőket 2 és 3 dimenzióban, valamint leírtam egy módszert a térbeli másodrendű mező generálására.

Mérésekkel alátámasztottam, hogy a másodrendű mező alacsony felbontáson pontosabb közelítést nyújtott, mint az alacsonyabb rendűek és sebességben sem volt jelentősen rosszabb.

A mezők generálása mellett a síkbeli elsőrendű mezőre kidolgoztam egy szűrési módszert amely kihasználja, hogy félsíkokat tárolunk és képes éles kapcsolódásokat reprezentálni.

A megoldásom továbbfejlesztéseként a későbbiekben szeretném megvalósítani a CSG szűrést síkbeli másodrendű valamint térbeli első és másodrendű mezőkre is.

Irodalomjegyzék

[1] Róbert Bán és Gábor Valasek. „Geometric Distance Fields of Plane Curves”. *Acta Cybernetica* 25.2 (2021), 187–203. old. DOI: 10.14232/actacyb.289248. URL: https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/4192.

[2] Vas Lia. *Surfaces*. Accessed: 2023-05-13. URL: https://liavas.net/courses/math430/files/Surfaces_part3.pdf.

[3] David Eberly. *Fitting 3D Data with a Torus*. Accessed: 2023-05-13. URL: https://www.geometrictools.com/Documentation/TorusFitting.pdf.

[4] Jack Goldfeather és Victoria Interrante. „A Novel Cubic-Order Algorithm for Approximating Principal Direction Vectors”. *ACM Trans. Graph.* 23.1 (2004), 45–63. ISSN: 0730-0301. DOI: 10.1145/966131.966134. URL: https://doi.org/10.1145/966131.966134.

[5] Nelson Max. „Improved accuracy when building an orthonormal basis”. *Journal of Computer Graphics Techniques (JCGT)* 6.1 (2017), 9–16. old. ISSN: 2331-7418. URL: http://jcgt.org/published/0006/01/02/.

[6] John C. Hart. „Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces”. *The Visual Computer* 12.10 (1996), 527–545. old. ISSN: 1432-2315. URL: https://doi.org/10.1007/s003710050084.

[7] Róbert Bán és Gábor Valasek. „Higher Order Algebraic Signed Distance Fields”. *Computer-Aided Design and Applications* 20.5 (2023), 1005–1028. old. URL: https://cad-journal.net/files/vol_20/CAD_20(5)_2023_1005-1028.pdf.

[8] Simon Kallweit és tsai. *The Falcor Rendering Framework*. https://github.com/NVIDIAGameWorks/Falcor. 2022. aug. URL: https://github.com/NVIDIAGameWorks/Falcor.

[9] D. P. Hardin, T. J. Michaels és E. B. Saff. *A Comparison of Popular Point Configurations on* $\mathbb{S}^2$. 2016. arXiv: 1607.04590 [math.NA].

[10] Inigo Quilez. *2D distance functions*. Accessed: 2023-05-15. URL: https://iquilezles.org/articles/distfunctions2d/.

Nyilatkozom, hogy a dolgozat elkészítéséhez nem vettem igénybe mesterséges intelligánciát és azt használó alkalmazást