



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

NUMERIKUS ANALÍZIS TANSZÉK

# Parametrikus felületek távolságmezőjének generálása és megjelenítése

*Témavezető:*

Bán Róbert

doktorandusz

*Szerző:*

Szente Péter

programtervező informatikus BSc

*Budapest, 2023*

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>4</b>
1.1. Szakirodalmi áttekintés . . . . .	5
<b>2. Felhasználói dokumentáció</b>	<b>6</b>
<b>3. Elméleti háttér</b>	<b>7</b>
3.1. Sphere tracing . . . . .	7
3.2. Előjeles távolságfüggvények . . . . .	8
3.2.1. Egyszerűbb alakzatok . . . . .	8
3.2.2. Parametrikus felületek távolságfüggvénye . . . . .	8
3.3. Bézier görbék . . . . .	10
3.3.1. Bernstein-alak . . . . .	10
3.3.2. Végpont interpoláció . . . . .	10
3.3.3. Pszeudolokális kontroll . . . . .	10
3.3.4. Derivált . . . . .	11
3.4. Bézier-felületek . . . . .	11
3.4.1. Bilineáris interpoláció . . . . .	11
3.4.2. Bernstein-alak . . . . .	11
3.4.3. Parciális deriváltak . . . . .	12
3.5. Legközelebbi pont meghatározása . . . . .	12
3.5.1. Merőlegesség feltétel . . . . .	12
<b>4. Algoritmusok</b>	<b>14</b>
4.1. Tesszelláció . . . . .	14
4.2. Sugárkövetés motivációja . . . . .	14
4.3. Fénymodell . . . . .	15
4.4. Távolságfüggvény rács számítása . . . . .	15
4.5. Sugárkövetés távolságmezőn . . . . .	16

4.6.	Egyszerűbb távolságfüggvény számító algoritmusok . . . . .	16
4.6.1.	Lipschitz módszer . . . . .	17
4.6.2.	Brute force . . . . .	17
4.7.	Gradiens módszerek . . . . .	18
4.7.1.	Gradiens módszer . . . . .	18
4.7.2.	Adam sztochasztikus gradiens módszer . . . . .	18
4.7.3.	AdaMax variáns . . . . .	19
4.8.	Vetített gradiens módszer . . . . .	20
<b>5.</b>	<b>Fejlesztői dokumentáció</b>	<b>21</b>
<b>6.</b>	<b>Szimulációs eredmények</b>	<b>22</b>
6.1.	Gradiens módszerek stabil paraméterezése . . . . .	22
6.1.1.	Rosenbrock-függvény . . . . .	22
6.2.	Globális minimum harmadfokú Bézier-felületen . . . . .	23
6.2.1.	Sarokpontok . . . . .	23
6.2.2.	Oldalak . . . . .	24
6.2.3.	10 pont módszer . . . . .	25
6.3.	AdaMax algoritmus Bézier felületeken . . . . .	25
6.3.1.	Felületgenerálás . . . . .	25
6.3.2.	Mintavételezési pont generálás . . . . .	26
6.3.3.	Referencia értékek . . . . .	26
6.4.	Validáció . . . . .	26
<b>7.</b>	<b>Implementáció</b>	<b>28</b>
7.1.	de Casteljau algoritmus . . . . .	28
7.1.1.	Iterációval . . . . .	28
7.1.2.	Swizzle operátorokkal . . . . .	29
7.1.3.	Mérési eredmények . . . . .	30
7.1.4.	Megjegyzések . . . . .	31
7.2.	Pont-felület távolság és gradiense . . . . .	31
7.2.1.	Analitikus pont-felület távolság . . . . .	31
7.2.2.	Gradiens számításának műveletigénye . . . . .	32
7.2.3.	A távolság gradiense . . . . .	32
7.3.	AdaMax . . . . .	33

<b>8. Eredmények</b>	<b>34</b>
8.1. Generálási módszer hatása a sugárkövetésre . . . . .	34
8.1.1. Lipschitz módszer . . . . .	35
8.1.2. Brute force módszer . . . . .	35
8.1.3. Adam módszer . . . . .	35
8.2. Távolságmező generálás . . . . .	35
8.3. Összehasonlítás a tesszellációval . . . . .	36
<b>9. Összefoglalás</b>	<b>38</b>
<b>Köszönetnyilvánítás</b>	<b>39</b>
<b>Irodalomjegyzék</b>	<b>39</b>
<b>Ábrajegyzék</b>	<b>42</b>

# 1. fejezet

## Bevezetés

A számítógépes modellezés egyik meghatározó eszköze a testek felületének különböző parametrikus felületekkel való leírása. Egyre szélesebb körben elterjedő implicit felületreprezentáció az előjeles távolságfüggvények és diszkretizált változatuk, a távolságmezők. A távolságmezővel reprezentált felület egy speciális sugárkövető eljárással, a sphere tracinggel jeleníthető meg.

Kutatásom célja parametrikus felületekkel ábrázolt objektumok távolságmezőjének generálására GPU segítségével. A létrejövő távolságmezők megjeleníthetők GPU-val gyorsított módon, az említett sphere tracing módszerrel. Ezt a módszert összehasonlítom a parametrikus felület háromszögekkel tesszellált megjelenítésével.

A távolságmezőt felhasználom még geometriai lekérdezések elvégzésére, mint a felületi normális meghatározása, ami a felület árnyalásához elengedhetetlen. Meghatározhatók még láthatósági viszonyok, melyet vetett árnyékok számításánál használok.

A készített programmal kétdimenziós függvények grafikonjai és Bézier felületek jeleníthetők meg. A Bézier felületekre azért esett a választás, mert a kontrollpont-hálójukon keresztül intuitívan manipulálhatók és a folytonosság megtartása mellett összeilleszthetők, így különösen alkalmasak felületek modellezésére. Harmadfokú Bézier felületek távolságmezőjének generálására speciális algoritmust is adok, melyet összehasonlítok a korábban tárgyalt módszerekkel.

## 1.1. Szakirodalmi áttekintés

A téma Hart cikkével [1] kezdődik, melyben bemutatja hogyan használható a sphere trace algoritmus felületek megjelenítésére. A módszer a sugárkövetést gyorsítja fel azzal, hogy a felülettől vett távolsággal lép a sugáron. Ez különösen a felülettől távol eredményez gyorsítást. A módszer előfeltétele, hogy a távolságfüggvényt viszonylag gyorsan ki tudjuk számítani. Ez könnyen megtehető egyszerűbb testek esetében. A módszert általában a GPU pixel shaderében implementálják, így a teljes szintér eltárolható GPU kódként. Emiatt a Demoscene-ekben nagyon sok példát látunk rá. Ilyeneket gyűjt össze például a Shadertoy weboldal.

Ha a felületek távolságfüggvénye nehezen számítható, vagy a jelenet túl összetett, akkor közelítésekre lehet szükség. Ha a távolságfüggvény becslés nem elég jó, akkor a sugárkövetés lelassul. Az alap sugárkövető algoritmus felgyorsítására több módszer is született. [2, 3, 4, 5] A sugárkövetés gyorsítható, ha a távolságfüggvényt egy diszkrét rácson kiértékeljük és az eredményt letároljuk. Ennek a műveletnek nem kell valós idejűnek lennie, minden objektumra előszámítható. A távolságmezőből a távolságfüggvény becslését úgy kapjuk, hogy a szomszédos 8 pontot trilineárisan interpoláljuk. Ha a távolságmezőben pontos értékek vannak, akkor a 8 távolság által reprezentált harmadfokú felület pontosan rekonstruálható. [6]

## 2. fejezet

### Felhasználói dokumentáció

## 3. fejezet

# Elméleti háttér

A sugárkövetéssel történő képszintézisről részletes összefoglaló olvasható Bálint Csaba OTDK dolgozatában [7, 11-16. o.], így azt itt nem részletezem.

### 3.1. Sphere tracing

Adott egy  $d : \mathbb{R}^n \rightarrow \mathbb{R}$  előjeles távolságfüggvény, mely minden bemenetre a pont a felület határától vett előjeles távolságát adja. Ez az érték a felület által határolt térrész belsejében negatív, kívül pedig pozitív. Formálisan  $d$  akkor távolságfüggvény, ha az alábbi teljesül [1]:

$$f(p) = d(p, f^{-1}(0)) \quad (p \in \mathbb{R}^n)$$

A sugárkövetés során adott egy kiinduló pont és egy sugárirány. A cél a félegyenes-felület metszéspont megtalálása. Míg a sugármetszés egyszerűbb matematikai objektumok esetén (pl. gömb, sík) analitikusan elvégezhető, bonyolultabb testek esetén csak óvatosabban közelíthetünk a felülethez a sugáron. (Ray Marching[8])

A sphere trace algoritmusról Hart cikkében [1] részletesen olvashatunk. Röviden összefoglalva a sphere trace egy sugárkövető algoritmus, mely minden lépésben kiértékeli a távolságfüggvényt. Tudjuk, hogy a távolságfüggvénnyel megegyező méretű üres tér van a pont körül, hiszen az a felülethez vett távolság minimumát, vagy annak *alsó közelítését* adja. Ekkor a távolsággal megegyező méretűt léphetünk a sugár mentén. A sugárkövetés a felület közelében lelassul. Ha a távolság epszilonnál kisebb, vagy fix lépésszám után leállítjuk az iterációt.



## 3.2. Előjeles távolságfüggvények

### 3.2.1. Egyszerűbb alakzatok

Inigo Quilez oldalán [9] felsorolja sok egyszerűbb alakzat analitikus távolságfüggvényét. Például egy  $o$  középpontú és  $r$  sugarú gömb előjeles távolságfüggvénye:

$$d_g(p) = \|p - o\|_2 - r$$

A weboldalon további testek távolságfüggvényei is láthatók. Ezekből aztán könnyen építhetünk színteret az alábbi tömörtest-modellezésben is használt műveletekkel: (Legyen az  $A$  testtől vett előjeles távolságfüggvény  $d_A$ , a  $B$  testtől vett pedig  $d_B$ )

- Mivel a távolságfüggvény a legközelebbi távolságot adja, két test uniója a távolságfüggvények minimuma.  $d(A \cup B) = \min(d_A, d_B)$
- Egy test „kifordítható”, ha a távolságfüggvény  $-1$ -szeresét vesszük  $d(\overline{A}) = -d_A$ .
- Két test metszete a távolságfüggvények maximuma, hiszen addig haladhatunk a sugár mentén, amíg mindkét alakzatot el nem találjuk.  $d(A \cap B) = \max(d_A, d_B)$
- Kivonást is elvégezhetünk, ha a test és a kifordított test metszetét vesszük.  $d(A \setminus B) = \max(d_A, -d_B)$

Elvégezhetők még a testeken transzformációk, ami általában a mintavételezési sípontra alkalmazott inverz-transzformációval történik. Ilyenek például a nagyítás, nyújtás, lekerekítés, forgatás, kétdimenziós alakzat kiterjesztése hasábbá vagy forgástestté.

Különösen érdekesek a tükrözés, ami az abszolút érték függvénnyel elvégezhető, illetve az alakzat véges és végtelen ismétlése, melyet a koordinátára alkalmazott moduló operátorral lehet elérni. [9]

### 3.2.2. Parametrikus felületek távolságfüggvénye

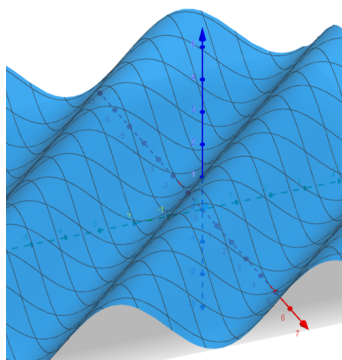
Háromdimenziós euklideszi térben parametrikus egyenlettel megadott felületeket hívjuk így.  $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$  Azért felület, mert a vektor-értékű függvény értelmezési

tartomány beli pontjaihoz háromdimenziós pontokat rendelünk. Modellezéskor így a test felszínét adjuk meg. Például a tórusz parametrikus egyenlete:

$$f(u, v) = \begin{pmatrix} r \cdot \sin(v) \\ (R + r \cdot \cos(v)) \cdot \sin(u) \\ (R + r \cdot \cos(v)) \cdot \cos(u) \end{pmatrix}$$

Ahol  $r$  a generáló kör sugara,  $R$  pedig a forgástengely és a kör középpontjának távolsága.  $u$  és  $v$  szögek, így  $u, v \in [0, 2\pi]$  Parametrikus egyenlettel megadhatók függvények grafikonjai is. Legyen  $h(u, v)$  a függvény, ekkor a grafikon parametrikus egyenlete:

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



3.1. ábra.  $h(u, v) = \sin(u + v) + 1$  grafikon a GeoGebra alkalmazásban.

Ezen felületek távolságfüggvénye analitikusan sok esetben nem meghatározható. Vegyünk egy kétdimenziós példát, az  $y = \sin(x)$  függvényt és egy  $(e, f)$  mintavételezési pontot, amihez a legközelebbi pontot keressük a felületen. ekkor a minimalizálandó függvény:

$$d(x) = \sqrt{(x - e)^2 + (\sin(x) - f)^2}$$

A gyök függvény szigorú monotonitása miatt vizsgáljuk  $d^2$  minimumát. Az elsőrendű szükséges feltétel:

$$\cos(x) \cdot (\sin(x) - f) + x - e = 0$$

Ezt csak néhány speciális esetben tudjuk analitikusan megoldani.

Polinomokkal sem boldogulunk könnyen analitikus módszerekkel. Például ha a polinom fokszáma harmadfokú, akkor a négyzetre emelés és deriválás után egy ötöd-fokú polinom gyökeit kellene meghatározni. Erre már nem létezik megoldóképlet. Ez az eset előkerül a dolgozatomban a Bezier felületek tárgyalásánál.

### 3.3. Bézier görbék

Ennek a résznek alapul G. Farin: *Curves and Surfaces for CAGD* című könyve szolgált. [10] A de Casteljau-algoritmusról és Bézier-görbékről a könyv 45-47., a Bernstein-polinomokról a könyv 57. oldalán olvashatunk. Ezeket itt nem részletezem, csak a szükséges tételeket mutatom be.

#### 3.3.1. Bernstein-alak

Az  $i$ -edik  $n$ -edfokú Bernstein-polinom alakja:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

Legyenek egy  $n$ -edfokú Bézier-görbe kontrollpontjai  $b_0, b_1, \dots, b_n$ . Ekkor a görbe felírható Bernstein-bázisban:

$$b(t) = \sum_{i=0}^n b_i B_i^n(t)$$

Ezt a görbe Bernstein-alakjának is nevezik

#### 3.3.2. Végpont interpoláció

A Bernstein-polinomok tulajdonságaiból adódik, hogy a görbe a végpontjaiban egyenlő a szélső kontrollpontokkal, azaz interpolál.

#### 3.3.3. Pszeudolokális kontroll

A legfontosabb tulajdonsága ezen görbéknek, hogy egy kontrollpont megváltoztatása csak annak környezetét változtatja meg jelentősen. Ennek oka, hogy  $B_i^n(t)$ -nek egyetlen maximuma van, a  $t = i/n$  helyen. [10, 62. o.]

A végpont interpoláció és a pszeudolokális kontroll miatt a görbe különösen alkalmas számítógépes modellezésre, mivel a felület a kontrollponthálójával intuitívan manipulálható.

### 3.3.4. Derivált

A görbe deriváltja: [10, 63. o.]

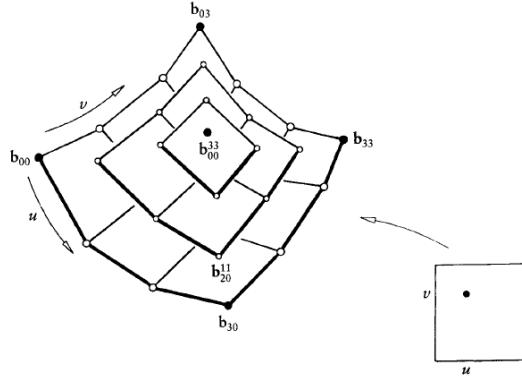
$$b'(t) = n \sum_{i=0}^{n-1} \Delta b_i B_i^{n-1}(t)$$

Ahol  $\Delta b_i = b_{i+1} - b_i$  ( $\Delta$  a jobboldali differenciaoperátor.)

## 3.4. Bézier-felületek

### 3.4.1. Bilineáris interpoláció

Farin a Bézier felületeket a tenzor szorzat felületek bevezetéseként írja le. Egy egyik irányban  $m$ , másik irányban  $n$ -edfokú felületet  $(m+1) \times (n+1)$  kontrollpont definiál. A felület pontjai kiszámíthatók a kontrollpontháló ismételt bilineáris interpolációjával. Ezt a felület pontbeli kiértékelésének nevezzük.



3.2. ábra. Bézier-felület kiértékelése [10, 248. o.]

### 3.4.2. Bernstein-alak

A végeredmény szempontjából lényegtelen, hogy az interpolációkat milyen sorrendben végezzük. Például kiértékelhetjük a kontrollponthálót az egyik paraméter szerint, majd az így keletkezett kontrollpontok által definiált Bézier-görbét a másik paraméter szerint. [10, 251. o.] A felület Bernstein-alakja:

$$b^{m,n}(u, v) = \sum_{i=0}^m \sum_{j=0}^n b_{i,j} B_i^m(u) B_j^n(v)$$

### 3.4.3. Parciális deriváltak

A könnyebb leírás érdekében bevezetjük a parciális differenciaoperátort. Ennek szabályai:  $\Delta^{1,0}b_{i,j} = b_{i+1,j} - b_{i,j}$  és  $\Delta^{0,1}b_{i,j} = b_{i,j+1} - b_{i,j}$

A parciális deriváltat a Bézier-görbe deriváltjára vezetjük vissza:

$$\partial_u b^{m,n}(u, v) = \sum_{j=0}^n \left[ \partial_u \sum_{i=0}^m b_{i,j} B_i^m(u) \right] B_j^n(v)$$

$$\partial_u b^{m,n}(u, v) = m \sum_{j=0}^n \sum_{i=0}^{m-1} \Delta^{1,0} b_{i,j} B_i^{m-1}(u) B_j^n(v)$$

A másik parciális derivált hasonlóan meghatározható:

$$\partial_v b^{m,n}(u, v) = n \sum_{j=0}^{n-1} \sum_{i=0}^m \Delta^{0,1} b_{i,j} B_i^m(u) B_j^{n-1}(v)$$

Magasabbrendű deriváltakat a dolgozatban nem használtam, így azokat itt nem tárgyalom. Az általános alak szintén megtalálható Farin könyvében.

## 3.5. Legközelebbi pont meghatározása

A legközelebbi pont meghatározása nem egyszerű feladat. Egyszerűsítésként egy  $b(x, y)$  függvény grafikonjára próbáljuk meg meghatározni. Ezen belül is vegyünk egy harmadfokú Bézier-felületet.

### 3.5.1. Merőlegesség feltétel

A legközelebbi pont szükséges feltétele: Az  $E$  mintavételezési pontból a felületi pontba húzott egyenes merőleges mindkét parciális deriváltra. Egyik irányra felírva:

$$r(x, y) = (x, y, b(x, y))$$

$$\frac{dr}{dx} = (1, 0, b_x(x, y))$$

Ahol  $b_x$  az első koordináta szerinti parciális derivált. A mintavételezési pont legyen az origó. Ezt megtehetjük, hiszen a felületet átparaméterezhető úgy, hogy a minta-

vételezési pont az origóba essen. Ha két vektor merőleges, a skalárszorzatuk 0.

$$0 = \left\langle r, \frac{dr}{dx} \right\rangle = (1 \ 0 \ b_x(x, y)) \cdot \begin{pmatrix} x \\ y \\ b(x, y) \end{pmatrix} = x + b_x(x, y)b(x, y)$$

$x$  szerint parciálisan integrálva:

$$f(y) = x^2/2 + b(x, y)^2$$

$$x = 0 \rightarrow f(y) = b(0, y)^2$$

$$b(0, y)^2 = x^2/2 + b(x, y)^2$$

A feltétel barátságos alakja ellenére a mi esetünkben (harmadfokú Bézier) ez egy ötödfokú kétismeretlenes egyenlet tetszőleges együtthatókkal.

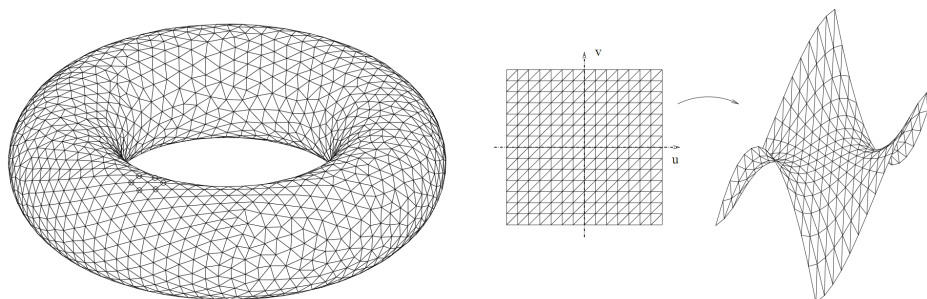
Látjuk, hogy ez a megközelítés nem vezet eredményre. Ugyanezt az egyenletet kapjuk, ha a szükséges feltételt a felületi normális segítségével írjuk fel. Ezt itt nem részletezem. Az algoritmusok résznél a problémára több közelítési módszert is bemutatok.

## 4. fejezet

# Algoritmusok

### 4.1. Tesszelláció

A legelső megjelenítési mód a felület háromszögekkel való közelítése. A Tesszelláció vagy háromszögelés során a cél úgy lefedni háromszögekkel a felületet, hogy annak részletei ne vesszenek el. Ez egyben a leggyakrabban használt modellezési módszer is. A videokártyák hardveresen támogatják háromszögek raszterizációját, így ez a módszer nagyon gyors. A többi módszer helyességét a háromszögekkel tesszellált közelítéssel fogom ellenőrizni. Ha függvények grafikonjait háromszögeljük, általában egyenletes felosztást veszünk a parmétertérben. A függvényértéket a harmadik koordináta reprezentálja.

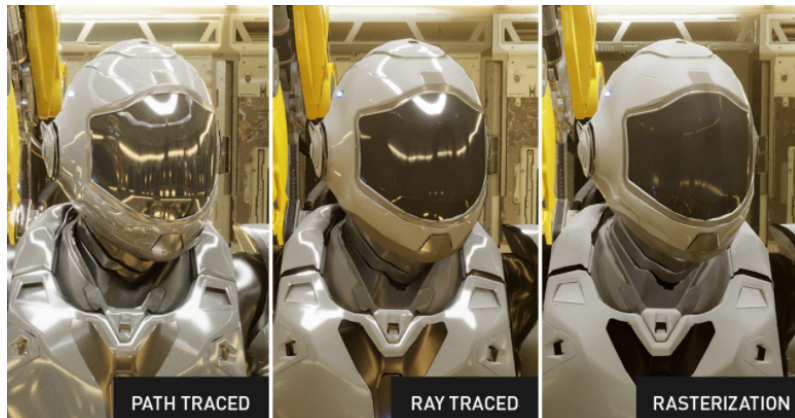


4.1. ábra. Tórusz és függvénygrafikon háromszögelése. Forrás: wikiwand.com

### 4.2. Sugárkövetés motivációja

A sugárkövetés mindenképpen költségesebb művelet, mint a raszterizáció, hiszen visszaverődéseket, törést és árnyéksugarakat is számítunk a jobb eredmény érdekében. Ha a fotorealistikus eredmény a cél, akkor viszont mindenképpen sugárkövetést

használunk és a cél az extra számítási költségek csökkentése, a sugárkövetés felgyorsítása.



4.2. ábra. Megjelenítési módok összehasonlítása. Forrás: [blogs.nvidia.com](https://blogs.nvidia.com)

### 4.3. Fénymodell

Fénymodellnek egy egyszerű Blinn-Phong árnyalást használtam [11] árnyéksugarakkal. Az árnyéksugár számítás ugyanazt az algoritmust használja, mint a sugárkövetés. Ha elmetsszük a felületet a felületi pontból a (pont)fényforrás felé indított sugárral, akkor a felületi pont árnyékban van.

Fontos megemlíteni, hogy a távolságfüggvények segítségével puha árnyékszámító algoritmus is adható. [12]

### 4.4. Távolságfüggvény rács számítása

A távolságfüggvényt egy diszkrét rácson értékelem ki, melyet egy háromdimenziós textúrában tárolok. Mivel csak a mintavételezési pont változik, maga a kiértékelendő (vagy meghatározandó) távolságfüggvény nem, így ki tudom használni a GPU masszívan párhuzamos architektúráját. A textúrát saját GPU compute shaderek segítségével számítom ki. A számítás elvégzésére később több módszert is mutatok.

Fontos megjegyezni, hogy ennek a műveletnek nem kell valós idejűnek lennie. A program indításakor a textúra több frame alatt kiszámítható. Ezután elég a textúrát a fragment shader számára feltölteni a GPU-ra a sugárkövetés előtt.



## 4.5. Sugárkövetés távolságmezőn

A sugárkövetés során adott a kiinduló pont  $P$ , és a sugár iránya  $d$ . Emellett adott egy  $F$  függvény, mely az előbb részletezett textúra olvasást és bilineáris interpolációt elvégzi, majd visszaadja a távolságfüggvény becslését. A távolságfüggvény rácsból az értékeket a textúra bilineáris interpolációjával nyerem ki. Ez a fajta textúra mintavételezés egy hardveresen támogatott művelet a GPU-n. Az alábbi függvény a kiindulási pont és a sugár felülettel vett első metszéspontjának távolságát adja meg.

---

### 1. algoritmus Sugárkövetés távolságmezőn

---

**Funct** DSFBoxTrace( $P, d, F$ )

---

```

1: if  $P$  a dobozon kívül van then
2:   return  $-1$                                 ▷ Távolság nem definiált
3: end if
4: if  $F(P) \leq 0$  then
5:   return  $0$                                 ▷ A doboz oldalán már a felületben vagyunk
6: end if
7:  $depth := 0$ 
8: for  $i \leftarrow 1$  to  $maxSteps$  do                ▷ Maximum lépésszám
9:    $dist := F(P + depth * d)$ 
10:  if  $abs(dist) < \varepsilon$  then
11:    return  $depth$                             ▷ Eltaláltuk a felületet
12:  end if
13:   $depth += dist$ 
14:   $currPos := P + depth * d$ 
15:  if  $currPos$  a dobozon kívül van then
16:    return  $-1$ ;                                ▷ Távolság nem definiált, nincs metszés
17:  end if
18: end for
19: return  $-1$ ;                                ▷ Távolság nem definiált, nincs metszés

```

---

## 4.6. Egyszerűbb távolságfüggvény számító algoritmusok

Ebben a részben két egyszerűbb megközelítést mutatok be, melyek általánosan használhatók előjeles távolságfüggvények generálására.

### 4.6.1. Lipschitz módszer

A módszer elméleti hátterét Bálint Csaba dolgozatából [7, 18. o.] idézem: „... ha  $f \in \mathbb{R}^n \rightarrow \mathbb{R}$  függvény Lipschitz folytonos, akkor

$$\frac{|f|}{Lip(f)}$$

távolságfüggvény becslés. Általában az abszolút érték elhagyható, hogy előjeles távolságfüggvényt kapjunk. Ezzel egy módszert kaptunk arra, hogy egy implicit függvényhez hogyan gyártsunk távolságfüggvényt. Sokszor a  $Lip(f)$ -et csak felülről tudjuk becsülni, vagy nem éri meg kiszámolni. Ilyenkor természetesen  $f$ -et a  $Lip(f)$  felső becslésével osztva egy rosszabb becslést kapunk, amivel sokszor lényegesen lassabb a számolás.”

A módszerrel egy grafikon például megjeleníthető az alábbi módon:

- Egy előre meghatározott felbontású rácson kiértékeljük a függvényt. Ezzel egy magasságtérképet generálunk.
- Megbecsüljük a Lipschitz konstanst a számolt értékekből vagy megadjuk analitikusan, ha ismert.
- A távolságmező minden eleme a pont magasságtérképtől vett távolsága lesz,  $Lip(f)$ -el leosztva. Ezzel távolságfüggvény-becslést kapunk.

### 4.6.2. Brute force

A felületet valamilyen felbontáson kiértékeljük. A legközelebbi pont távolságát eltároljuk a távolságmezőben. A módszer hátránya, hogy a számítási igény négyzetesen nő a felbontás növelésével. (Amellett, hogy a háromdimenziós rács minden pontjára külön ki kell számolni.)

A módszer előnye, hogy a távolságmező (a felbontás limitációja mellett) pontos és maximális lesz. A távolságmező minden pontjára igaz, hogy az értéke nem növelhető, hiszen minden pontra létezik olyan sugár, melyen az ott tárolt távolságot lépve a felület egy ismert pontjába lépünk. Ezen tulajdonság miatt a sugárkövetésnek sokkal kevesebb lépésre van szüksége a felület megtalálására. A bemutatott két módszer közül a brute force a preferált, hiszen a távolságmező előszámítható és akár el is tárolható háttértáron.

## 4.7. Gradiens módszerek

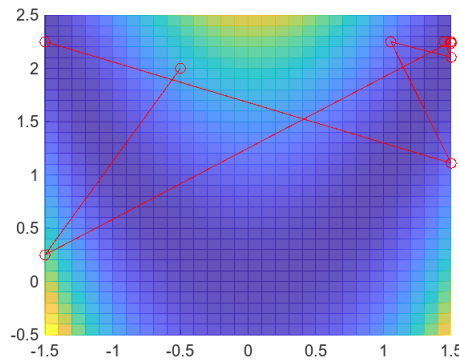
A felülettől vett minimális távolság valójában egy optimalizációs probléma. Optimalizációs problémák megoldásának nagyon széles szakirodalma van. Ilyen problémát kell megoldani többek között neuronhálók tanításánál is. 2020-ban távolságfüggvényekkel való ütközésetektálás javítására is használtak lokális optimalizációt. [13] A távolságfüggvényekhez két, a gépi tanulásban elterjedt algoritmust is implementáltam.

### 4.7.1. Gradiens módszer

A Gradiens módszer egy elsőrendű iteratív optimalizációs algoritmus. A módszer alapötletét egészen Cauchy-ig vezetik vissza [14], így egyáltalán nem újszerű. Az iteráció egy lépése:

$$a_{n+1} = a_n - \alpha \nabla F(a_n)$$

Ahol  $F$  a minimalizálandó függvény,  $\nabla$  a gradiens-operátor és  $\alpha$  a tanulási ráta.  $\alpha$  egy kis konstans, ami a módszer konvergenciáját biztosítja. Ha túl nagyra állítjuk, akkor a módszer nem konvergál.



4.3. ábra. Gradiens módszer divergenciája túl nagy tanulási ráta esetén

A módszer felparaméterezését és alkalmazhatóságát a szimulációs eredményeknél tárgyalom.

### 4.7.2. Adam sztochasztikus gradiens módszer

Az Adam [15] egy sztochasztikus gradiens módszer. Minden lépésben adaptívan változtatja a tanulás paramétereit. Ezekek közül  $m_t$  egy tapasztalati momentum, melyet a korábbi gradiensek exponenciálisan csökkenő súlyozásával kapunk. Hasonlóan

számítandó  $v_t$ , mely a gradiensek négyzetének súlyozott átlaga, avagy a tapasztalati szórás. Legyen  $g_t$  a gradiens, ekkor

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Ahol  $\beta_1$  és  $\beta_2$  a módszer hiperparaméterei. A paraméterek a gyakorlatban a kezdeti értékük felé (általában 0) statisztikai ferdeséget (bias) mutatnak, ezért korrigálni kell őket:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Ezután egy lépés szabálya:

$$a_{n+1} = a_n - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t$$

A hiperparaméterekre a cikk szerzői ezeket az értékeket javasolják:  $\alpha = 0.002$   $\beta_1 = 0.9$   $\beta_2 = 0.999$  és  $\varepsilon = 10^{-6}$ . Ezeket szimulációs eredmények alapján állítottam be az saját alkalmazáshoz. Mivel a GPU-n a dupla pontosságú lebegőpontos értékek számításigényesebbek, így shader környezetben nagyobb  $\varepsilon$ -t kell használni a numerikus stabilitás érdekében.

### 4.7.3. AdaMax variáns

Ezt a módosítást a cikk [15] szerzői a cikk végén írják le. A lényege, hogy a szórás számításánál a kettes norma kicserélhető végtelen normára. Ezután a frissítés szabálya átalakítható max függvényre: (Itt  $v_t$ -t  $u_t$ -re cseréljük a megkülönböztethetőség kedvéért)

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) \|g_t\|_\infty$$

$$u_t = \max(\beta_2 u_{t-1}, |g_t|)$$

Az AdaMax frissítési szabálya ennek felhasználásával:

$$a_{n+1} = a_n - \frac{\alpha}{u_t} \hat{m}_t$$

Végül ezt a variánst implementáltam GPU-n, mivel a számítása egyszerűbb és numerikusan stabilab, illetve mert néhány iterációval jobb eredményt ért el a szimuláció során.

## 4.8. Vetített gradiens módszer

Előfordul, hogy az optimalizációs problémát a paraméterter csak egy kis részén kívánjuk megoldani. Például Bézier-felületeknél általában a  $[0, 1]$  intervallumon. Ebben az esetben nem elég az optimalizáció végén levetíteni az eredményt, ugyanis az a legtöbb esetben nem egyezik meg a tartományon vett minimumhellyel. [16] Az sem jó, ha a lépés megtétele után vetítjük le a pontot a tartományra, ekkor ugyanis a gradiens módszer helytelen adatokkal fog számolni.

Már a gradiens számításakor figyelembe kell venni a korlátokat. Legyen  $g$  a gradiens,  $\pi_C()$  pedig egy függvény, ami minden pontot a hozzá legközelebbi  $C$  tartománybeli pontba visz. A vetített gradiens így számítható:

$$g_p = (a_n - \pi_C(a_n - \varepsilon g)) / \varepsilon$$

Ahol  $\varepsilon$  egy kis szám. Ez úgy értelmezhető, hogy a gradiens irányába megteszünk egy kis lépést, majd azt levetítjük a tartományra. A vetített pont és az előző pont különbségéből megkapjuk a vetített gradienst irányát.

Ezt odaadjuk a gradiens módszerünknek, majd a lépés után ismét levetítjük.

## 5. fejezet

### Fejlesztői dokumentáció

## 6. fejezet

# Szimulációs eredmények

GPU környezetben nehéz algoritmusokat tesztelni és hibákat javítani, a masszívan párhuzamos környezet miatt, ezért az algoritmusokat egy szálon, Matlabban is implementáltam. A vizsgálat középpontjában az algoritmusok stabilitása és konvergenciája állt.

### 6.1. Gradiens módszerek stabil paraméterezése

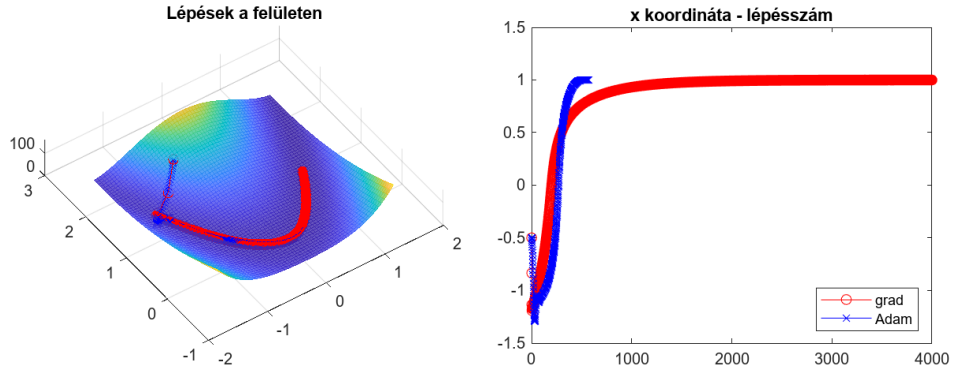
A gradiens módszer egyszerűsége ellenére meglepően jól teljesít Bézier felületeken abban az esetben, ha a tanulási rárát ( $\alpha$ ) a lehető legnagyobb értékre állítjuk. Ekkor azonban nem tudunk garantálni semmiféle konvergenciát. Emiatt olyan hiperparaméter-beállítást keresünk, mely szélsőséges esetben is a lokális minimumhoz konvergál.

#### 6.1.1. Rosenbrock-függvény

A Rosenbrock-függvény egy klasszikus "nehéz" példa, melyet optimalizációs algoritmusok tesztelésére szoktak alkalmazni. Definíciója:

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

Ennek a globális minimuma az  $(a, a^2)$  helyen van. Az  $a$  paraméter értéke általában 1. A  $b$  paraméterrel a probléma „nehézségét” lehet állítani. Minél nagyobb a  $b$  érték, annál nagyobb lesz a gradiensvektor hossza. Én 20-ra állítottam.



6.1. ábra. Stabil paraméterezés a Rosenbrock függvényen

A gradiens módszer paramétere:  $\alpha = 0,005$

Az AdaMax módszer paramétere:  $\alpha = 0,005$ ,  $\beta_1 = 0,9$ ,  $\beta_2 = 0,99$

A 6.1 ábrán pirossal a gradiens módszer lépéseit, késsel az AdaMax módszer lépéseit jelöltem. Az AdaMax módszer 569 lépésben  $10^{-6}$  nagyságrendű hibával konvergál. A gradiens módszer hibája 1000 lépés után  $1/10$ -nél nagyobb, és még 5000 lépés után is  $10^{-4}$  nagyságrendű.

Ebből a példából jól látszik a bonyolultabb módszer előnye, ha megköveteljük a konvergenciát nehezebb példákra is.

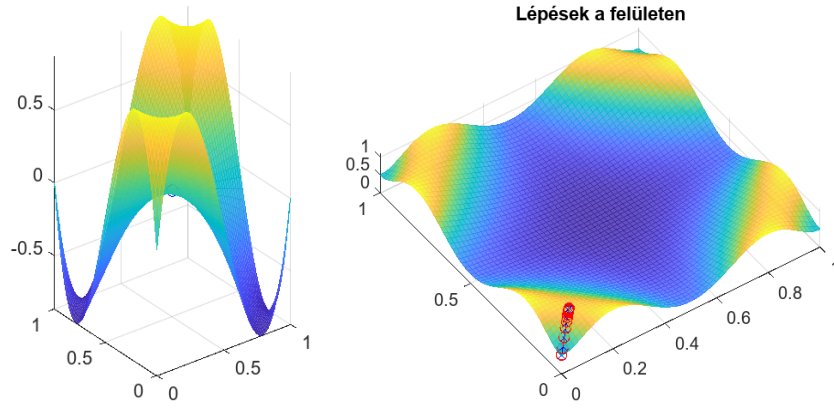
## 6.2. Globális minimum harmadfokú Bézier-felületen

A módszereket Bézier-felületeken is összehasonlítottam. A cél alapvetően a Bézier-felület távolságfüggvényének generálása. Ehhez a globális minimumot kell meghatározni. Ezt úgy kívánjuk elérni, hogy a lokális optimumkereső algoritmust több pontból elindítjuk. Az itt következő példák intuíciót adnak az indítási pontok minimális számára, amit a következő részben szimulációval validálok.

### 6.2.1. Sarokpontok

A 6.2 példában az látszik, hogy lokális optimum közel lehet a felület sarkához ((0, 0), (0, 1), (1, 0), (1, 1) pontok). Ha a felület belsejéhez tartozó kontrolpontok koordinátáit nagyobbra állítjuk, a baloldali ábrán látható lokális szélsőértékek tetszőlegesen közel kerülhetnek a sarokpontokhoz.





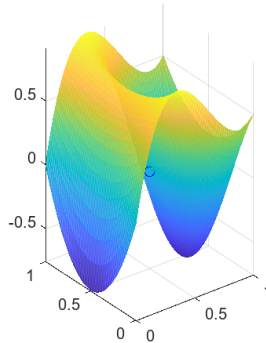
6.2. ábra. Példa sarokpontok szükségességére

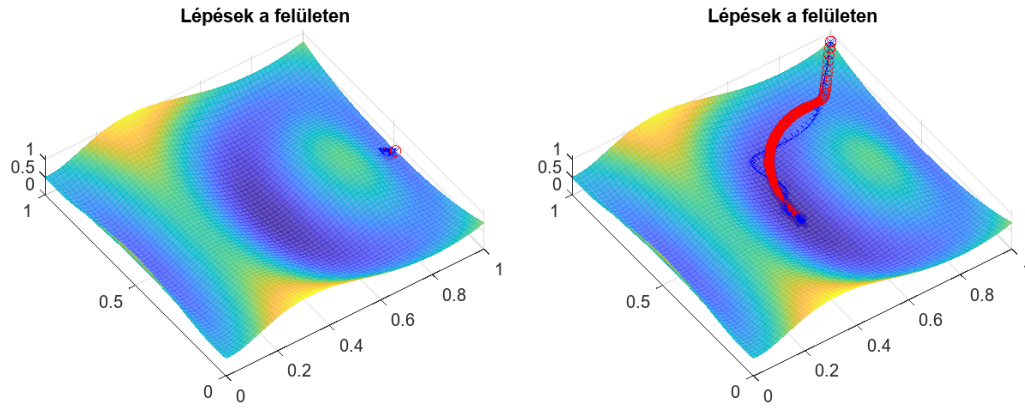
A jobboldali ábrán a felület és az  $(1/2, 1/2, 0)$  pont távolsága látható. Emellett a gradiens módszerek lépései szerepelnek az  $(1/10, 1/10)$  pontból indítva.

Ezen a példán az látszik, hogy a sarokpontokból el kell indítani a keresést, illetve hogy a felület belsejében lévő lokális minimumot nem lehet minden esetben a sarkokból megtalálni. Én ezért az  $(1/2, 1/2)$  pontot is hozzávettem a kiindulási pontokhoz.

### 6.2.2. Oldalak

Az alábbi példán az látszik, hogy a lokális minimum lehet az oldal mentén is, és azt nem feltétlenül találjuk meg a sarokpontból indulva.





6.3. ábra. Példa oldalpontok szükségességére

Emiatt én az oldalak középpontját is hozzávettem a kiindulási pontokhoz.

### 6.2.3. 10 pont módszer

9 indítási pont egy egyenletes  $3 \times 3$ -as felosztás a paramétertérben. Ezzel a felület nagyobb vonásait lefedjük. Azért, hogy a módszer a felület közelében is gyorsan konvergáljon, a mintavételezési pont alatti felületi pontból is elindítom a keresést. Mivel grafikonon vagyunk, ez megegyezik az első két koordinátával.

## 6.3. AdaMax algoritmus Bézier felületeken

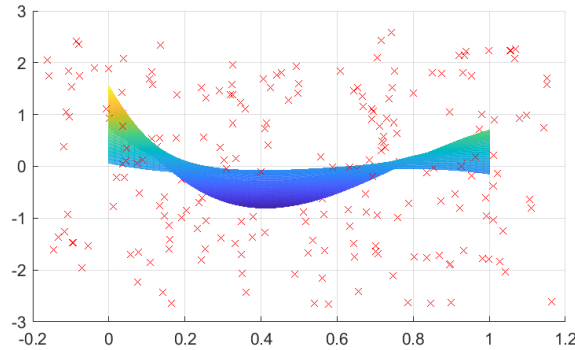
A 10 pont módszer helyességét szimulációval kívánom igazolni. Ehhez nagyszámú mintát generáltam, majd ellenőriztem, hogy a javasolt módszer minden alkalommal megtalálta-e a globális minimumot.

### 6.3.1. Felületgenerálás

A harmadfokú Bézier-felületek generálásához elég a  $4 \times 4$  kontrollpontot megadni. Ehhez egyenletes eloszlással véletlen pontokat vettem a  $[-1, 1]$  intervallumból. Az így keletkező felületek a közepükön meglehetősen laposak voltak. Ennek oka, hogy amíg a 0. és a 3. harmadfokú Bernstein-polinom maximuma az 1 értéket veszi fel, addig a középsők maximuma csak  $4/9$ . Ez különösen látványos a felület közepén, ott ugyanis két Bernstein-polinom szorzata áll melyek maximuma  $(4/9)^2$ . Ezen tulajdonság miatt a középső kontrollpontoknak sokkal kisebb hatása van, mint a szélsőknek, ahol ráadásul interpolál is a felület. Emiatt minden kontrollpont harmadik koordinátáját leosztottam a hozzá tartozó Bernstein-polinom maximumával.

### 6.3.2. Mintavételezési pont generálás

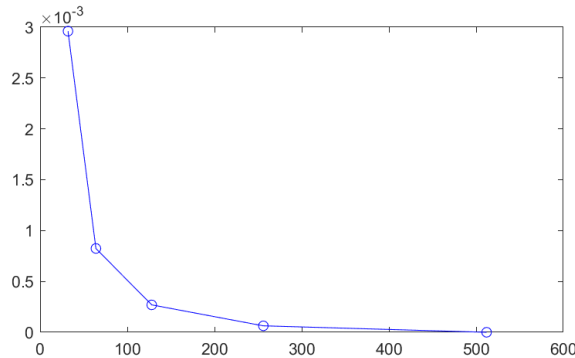
A mintavételezési pontokat a felület bennfoglaló dobozából és annak környezetéből vettem. Ehhez egy egyenletes felosztású térrács pontjait random vektorokkal eltoltam. Az ábrán egy példa merőleges vetülete látható.



6.4. ábra. Felület és pontthalmaz képe

### 6.3.3. Referencia értékek

Referencia értéként kiszámítottam minden mintavételezési pontra a „brute force” módszer eredményét, azaz a távolságot egy  $n \times n$ -es rácson kiértékeltem.



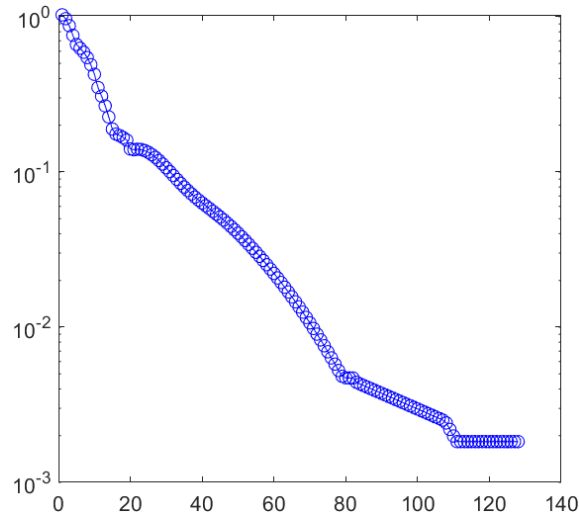
6.5. ábra. Referencia érték maximális hibája a legnagyobb felbontáshoz képest

A tesztek alapján a hiba a műveletigénnyel arányosan csökken, pontosabban a felbontás ( $n$ ) duplázásakor a hiba közel a negyedére csökken. A továbbiakban az  $n = 256$  értéket használom.

## 6.4. Validáció

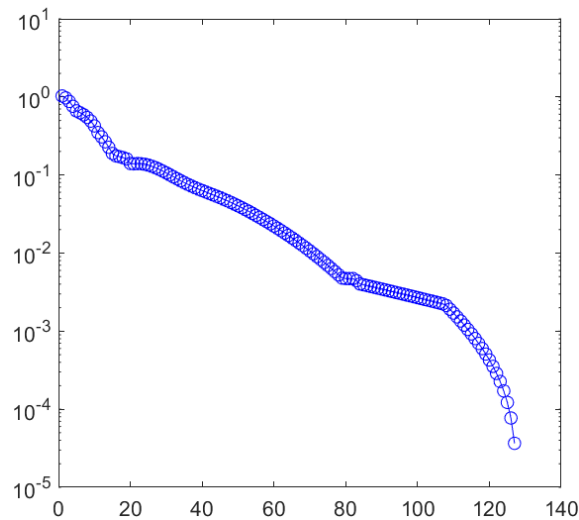
A 10 pontos algoritmust 1000 különböző példára futtattam, majd összehasonlítottam a kapott minimumot a referencia értékkel. Mivel azt szeretnénk, hogy az algo-

ritmus a *legrosszabb* esetben is megadja az optimumot, azért a grafikonon az összes futtatás közül a relatív hiba maximumát ábrázoltam minden egyes lépésszámmra.



6.6. ábra. Abszolút hiba maximuma a lépésszám függvényében, logaritmikus skálán.

A módszer minden futtatás esetén megtalálta az optimumot legalább a referencia érték pontosságával, kevesebb, mint 120 iteráció alatt. A grafikon 110 lépés után azért laposodik el, mert az algoritmus eléri a referencia érték pontosságát. Ha referenciának az utolsó iteráció által adott távolságot vesszük, akkor látható, hogy a módszer tovább konvergál.



6.7. ábra. Abszolút hiba maximuma az iteratív módszerhez viszonyítva.

## 7. fejezet

# Implementáció

Ebben a részben néhány fontos implementációs részletre térek ki. Ennek motivációja, hogy a CPU implementáció profilozásakor kiderült, az algoritmus az idő 60%-át a felület kiértékelésével és a gradiens kiszámításával tölti.

### 7.1. de Casteljau algoritmus

#### 7.1.1. Iterációval

Az algoritmus CPU implementációja két egymásba ágyazott ciklussal működik. Ez a 7.1-es forráskódban látható.

```
1  float deCasteljau_iter(float4 pts, float t, float one_minus_t)
2  {
3      for (int j = 1; j <= 3; j++)
4      {
5          for (int i = 0; i <= 3 - j; i++)
6          {
7              pts[i] = one_minus_t * pts[i] + t * pts[i + 1];
8          }
9      }
10     return pts[0];
11 }
```

7.1. forráskód. de Casteljau iterációval

### 7.1.2. Swizzle operátorokkal

Részben azért esett a választás a köbös Bézier-felületekre, mert a GPU kód támogat műveleteket legfeljebb 4 hosszú vektorokkal. Ez azt jelenti, hogy a felület kontrollponthálójának egy sora belefér egy ilyen vektorba. Az elméleti áttekintésnél láttuk, hogy a felület kiértékelése visszavezethető az egydimenziós problémára, így ezt a függvényt kell vektorizálni. Ezt a szintén hardveresen támogatott ún. swizzle operátorokkal tesszük. Ezek lényege, hogy a művelet elvégzése előtt a bemeneti vektorok elemeit tetszőlegesen átrendezhetjük, akár duplikálhatjuk is. A javított implementáció a 7.2-es forráskódban látható.

```
1  float deCasteljau_swizzle(float4 pts, float t, float one_minus_t)
2  {
3      for (int j = 1; j <= 3; j++)
4      {
5          pts.xyz = one_minus_t * pts.xyz + t * pts.yzw;
6      }
7      return pts[0];
8  }
```

7.2. forráskód. de Casteljau swizzle operátorokkal

A programokat először a Radeon GPU Analyzer-rel hasonlítottam össze. A gépi kódban már egyáltalán nem lesz ciklus, mert a driver kibontja. A ciklusos implementációban 11, a swizzle implementációban 10 vektorművelet szerepelt. Ebből messzemenő következtetést nehéz levonni. A swizzle implementáció ciklusát kézzel is kibontottam. Ez nem okozott jelentős javulást. Végül lecseréltem minden sort egy lerp (lineáris interpoláció) utasításra. Az implementáció a 7.3-as forráskódban látható. Ezt a driver jobban tudja optimalizálni.

```
1  float deCasteljau_lerp(float4 pts, float t)
2  {
3      pts.xyz = lerp(pts.xyz, pts.yzw, t);
4      pts.xy = lerp(pts.xy, pts.yz, t);
5      return lerp(pts.x, pts.y, t);
6  }
```

7.3. forráskód. de Casteljau lerp függvénnyel

### 7.1.3. Mérési eredmények

Az algoritmusokat a „Brute force” távolságmező generáló módszer segítségével hasonlítottam össze, mert ez főként felületkiértékelést végez. Minden esetben egy  $32^3$  méretű textúrát generáltam és kiátlagoltam 512 futtatást. A 7.1 és 7.2 táblázatok a számításhoz szükséges GPU-időt foglalják össze milliszekundumban. Először a Bézier-felületből készített magasságtérképet textúrába írtam, majd a távolságmező generálásakor onnan kiolvastam. A 7.1 táblázat második oszlopában ezek az értékek szerepelnek. A harmadik oszlopban a távolságmező generálásakor számítottam ki a függvényértékeket.

Referenciának kimockoltam a de Casteljau függvényt azzal, hogy csak az utolsó sort hagytam meg. A GPU Analyzer-ből kiderül, hogy ekkor teljesen eltűnik a függvény, mert a driver mindenhol inline-olja. Ezt az értéket kivontam a harmadik oszlopból, így megkaptam, mekkora része a futási időnek a felület kiértékelése.

(ms)	Memóriából olvasva	Számítva	ebből de Casteljau
két ciklusos	51,6	76,97	75,76
swizzle	51,35	2,74	1,53
kibontott	51,33	<b>2,63</b>	1,42
üres	51,41	1,21	0

7.1. táblázat. de Casteljau algoritmus mérése egy  $32^3$  méretű textúrán

A mérésekből kiderül, hogy a ciklusos implementáció nagyjából *50-szer lassabb* GPU-n, mint a többi számításos verzió. A második oszlopban az értékek nagyjából megegyeznek, a futási idők pedig jóval a táblázatban szereplő legjobb idő felett vannak. Ebből arra következtethetünk, hogy ha memóriából olvassuk ki a felületértékeket, akkor a limitáció a memóriaelérés által okozott késleltetés lesz.

A legjobb megoldásokat összehasonlítottam egy nagyobb,  $64^3$  méretű textúrán is. Ennek eredményeit a 7.2 táblázat tartalmazza.

(ms)	Számítva	ebből de Casteljau
swizzle	64,09	49,38
kibontott	62,13	47,42
lerp	<b>52,85</b>	38,14
üres	14,71	0

7.2. táblázat. de Casteljau algoritmus mérése egy  $64^3$  méretű textúrán

A kézzel kibontott ciklus nem okoz lényeges javulást, viszont a lineáris interpoláció nagyjából 25%-al gyorsabb.

#### 7.1.4. Megjegyzések

- A felület kiértékeléséhez a kontrollpont-háló minden sorára meg kell hívni az algoritmust, majd az eredményekből álló vektorra megint. Ez összesen 5 de Casteljau függvényhívás.
- Matematikailag létezik ennél gyorsabb algoritmus, de nem használjuk, mert a numerikus stabilitásra nagyon oda kell figyelni GPU környezetben. (Szűk számbázis miatt.)
- Emellett az is megfigyelhető, hogy a matematikai műveletigény és a driver által generált kód futási ideje között nem feltétlenül intuitív az összefüggés.

## 7.2. Pont-felület távolság és gradiense

### 7.2.1. Analitikus pont-felület távolság

Legyen  $E(e_1, e_2, e_3)$  a mintavételezési pont modell koordináta-rendszerben és  $r(x, y) = (x, y, b(x, y))$  pedig a felület parametrikus egyenlete. Ekkor a távolság egy adott felületi és mintavételezési pontra:

$$D_1(x, y) = \|E - r\|_2 = \sqrt{(e_1 - x)^2 + (e_2 - y)^2 + (e_3 - b(x, y))^2}$$

Ennek egyik parciális deriváltja:

$$\partial_x D_1 = \frac{2(x - e_1) + 2(b(x, y) - e_3)b_x(x, y)}{\sqrt{(e_1 - x)^2 + (e_2 - y)^2 + (e_3 - b(x, y))^2}}$$

Ahol  $b_x(x, y)$  a magasságfüggvény  $x$  szerinti parciális deriváltja. A nevezőben lévő érték a távolság, ami a felület közelében 0 közeli. Mivel a távolságfüggvényt a felülethez közel is kiértékeljük, ez mindenképp numerikusan instabil lesz, sőt, nullával osztást is eredményezhet.

A távolságfüggvényhez a legközelebbi pontot akarjuk meghatározni a felületen. Mivel a négyzetgyök függvény szigorúan monoton nő, a norma négyzetének is ugyan-



ott lesz minimum helye, ahol a normának.

$$D(x, y) = \|E - r\|_2^2 = (e_1 - x)^2 + (e_2 - y)^2 + (e_3 - b(x, y))^2$$

Ennek parciális deriváltjai:

$$\partial_x D = 2(x - e_1) + 2(b(x, y) - e_3)b_x(x, y)$$

$$\partial_y D = 2(y - e_2) + 2(b(x, y) - e_3)b_y(x, y)$$

Ezt sokkal könnyebben és stabilabban tudjuk számolni.

### 7.2.2. Gradiens számításának műveletigénye

Távolságfüggvényeknél gyakori, hogy a gradienst szimmetrikus differencia módszerével határozzuk meg. Ehhez nem kell ismerni a távolságfüggvény képletét, csak ki kell értékelní több helyen. Egy kis  $\varepsilon$  számot választva a gradiens közelítése:

$$b' \approx \frac{1}{2\varepsilon} \cdot \begin{pmatrix} b(x + \varepsilon, y) - b(x - \varepsilon, y) \\ b(x, y + \varepsilon) - b(x, y - \varepsilon) \end{pmatrix}$$

Itt a  $b(x, y)$  függvény kiértékelését négyszer el kell végezni. Emellett egy kivonásra és egy szorzásra is szükség van.  $(1/(2\varepsilon))$  konstans) Ez összesen 122 matematikai művelet.

A parciális derivált számításához csak 12 mátrix elemet kell redukálni, így annak kiértékelése 22 matematikai művelet.  $b$  totális deriváltjának analitikus számításához a két parciális deriváltat kell kiszámolni, ami 44 művelet. Mivel az utóbbi matematikailag hatékonyabb, ezt a verziót implementáltam.

### 7.2.3. A távolság gradiense

A totális derivált kifejezhető vektorokkal:

$$D' = 2 \left[ \begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} \right] + (b(x, y) - e_3) \cdot b'(x, y)$$

Ezt a gradienst használjuk az iterációs módszerben.

### 7.3. AdaMax

A AdaMax algoritmus kódja a pszeudokóddal teljesen ekvivalens. Kizárólag a numerikus stabilitásra kell figyelni. A vetített gradiens számításánál az  $\varepsilon$  érték nem lehet túl kicsi.  $\varepsilon = 10^{-6}$  érték például már a végeredményben is látható numerikus hibákat okoz. Én  $\varepsilon = 10^{-4}$  értéket használtam.

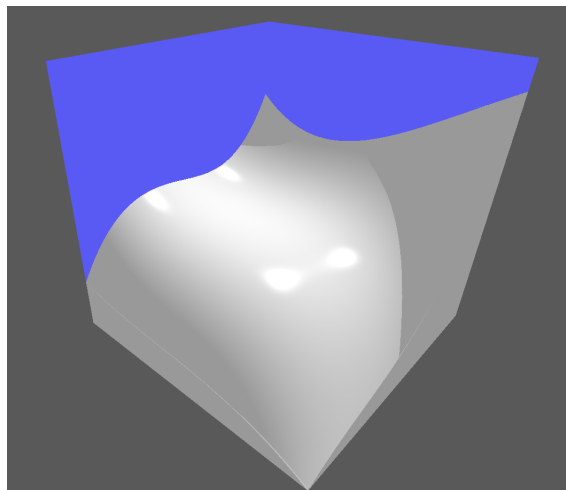
Szintén numerikus hibák és a zéróosztás elkerülése végett a tapasztalati szórás minimumát is  $10^{-4}$ -re állítottam. Ez alapján  $u$  frissítési szabálya:

$$u_t = \max(\beta_2 u_{t-1}, |g_t|, 10^{-4})$$

## 8. fejezet

# Eredmények

Az alábbi mérési eredményeket ugyanazon teszt példákon végeztem el mindhárom módszerre. A használt videokártya egy Nvidia GTX 1060 Ti notebook GPU. A felbontás minden esetben Full HD.



8.1. ábra. Bézier-felület

### 8.1. Generálási módszer hatása a sugárkövetésre

Itt azt hasonlítom össze, hogyan hat a sphere trace algoritmus futási idejére a generálási módszer. A 8.1 táblázat utolsó oszlopa 512 képkocka megjelenítésének átlagos GPU-idejét foglalja össze. A jelenetben minden esetben egy véletlenszerűen generált Bézier-felület és egy pontfényforrás szerepelt, amihez ú.n. kemény árnyékokat számítottam.

Generálási módszer	Szükséges lépésszám	Átlagos GPU idő (ms)
Lipschitz	512 <	1,47
Brute force	< 256	0,95
Adam	< 256	0,96

8.1. táblázat. Generálási módszer hatása a sugárkövetésre

### 8.1.1. Lipschitz módszer

A Lipschitz módszer időkölsége a memóriaigénnyel egyenesen arányos. Az ilyen algoritmusok minden esetben memória-limitáltak. A generált távolságmező olyan rossz minőségű, hogy alacsony látószögből még 512 lépésben sem konvergál a sugárkövetés. A generálás emellett olyan olcsó, hogy akár sugárkövetéskor, a pixel shaderben is kiszámítható. Ezt a módszert a következőkben nem tárgyaljuk.

### 8.1.2. Brute force módszer

A Brute force módszer meglehetősen költséges, azonban a generált távolságmező (a rácspontokban) biztosan optimális, így 256 iteráció minden esetben elég volt.

### 8.1.3. Adam módszer

A tesztek meglepő eredménye, hogy az AdaMax algoritmusban 25-nél nagyobb iterációs szám egy esetben sem javított a képminőségen. Ennek egyik oka, hogy a felület közelében a 10. pont miatt nagyon gyorsan konvergál a módszer. Másik oka, hogy ha a sugárkövetés során beleszaladunk a felületbe egy rossz becslés miatt, akkor a következő távolságkiértékelés negatív lesz, és visszatalálunk a felülethez. A sugárkövetéshez továbbra is elegendő volt 256 iteráció, minden esetben, így kijelenthetjük, hogy a szimulációs eredményeket sikerült megerősíteni.

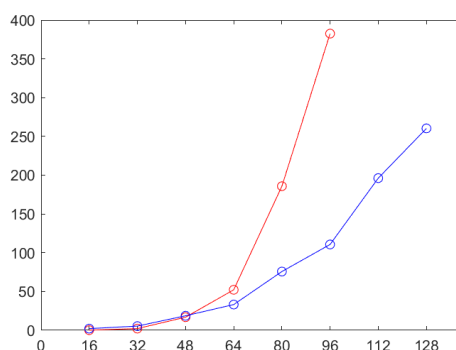
## 8.2. Távolságmező generálás

Mivel a Lipschitz-módszer csak egy gyenge alsó közelítést ad a távolságmezőre, így a Brute force algoritmust hasonlítottam össze az AdaMax algoritmust használó módszerrel. A 8.2 táblázatban az előbb bemutatott teszt példa távolságmezőjének generálásához szükséges átlagos GPU-idők szerepelnek a felbontás függvényében, milliszekundumban.

Felbontás	16	32	48	64	80	96	112	128
<b>Brute force</b>	0.17	2.18	16.87	52.23	185.79	382.57	-	-
<b>AdaMax</b>	1.09	4.36	15.82	28.04	63.49	92.65	163.07	219.13

8.2. táblázat. A generálás ideje a felbontás függvényében, milliszekundumban

Az értékeket a 8.2 grafikonon meg is jelenítettem:

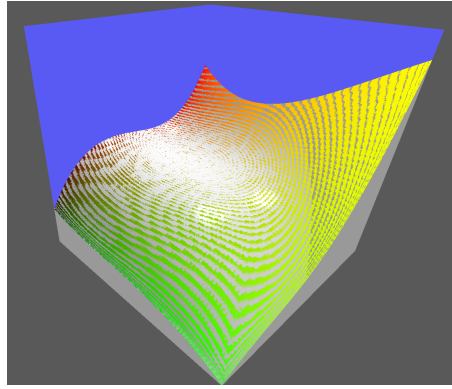


8.2. ábra. A generálás ideje a felbontás függvényében, milliszekundumban

A 8.2 teszt eredményeiből látszik, hogy a két módszer algoritmikus komplexitásából adódó különbségek már kis felbontás esetén ( $64^3$ ) is jelentősek. A komplexitásbeli különbség oka, hogy amíg a Brute force megoldás a függvényt egyre növekvő felbontáson értékeli ki, addig az új módszer fix lépést végez 10 pontból.

### 8.3. Összehasonlítás a tesszellációval

A felületet a sugárkövetés mellett háromszögekkel közelítve is megjeleníthetjük. A két módszer eltéréseit úgy tudjuk vizualizálni, ha sugárkövetés után a fragment shaderben beleírjuk a mélység adatot a mélység bufferbe, és az egyik felület árnyalását valami egyszerűre cseréljük. A 8.3 ábrán a sugárkövetett felületet Blinn-Phong árnyalással, a háromszögekkel tesszelláltat pedig egy egyszerű színskálával jelenítettem meg.



8.3. ábra. Tesszellált és sugárkövetett felület egyszerre kirajzolva

A 8.3 ábrán a két módszerrel kirajzolt felület színe váltakozva jelenik meg. Ennek oka, hogy ha a mélységbuffer értékei helyesek, akkor átlátszatlan felületeknél minden pixel színe a legközelebbi felület színe lesz. A váltakozás oka, hogy amíg felületet tartalmazó voxelekben a sugárkövetés egy bilineáris felületet határoz meg, addig a raszterizációnál a felületet két, a csúcspontokban interpoláló háromszöggel közelítjük.

A raszterizációt felhasználhatjuk arra, hogy az első sugárkövetést megspóroljuk és csak az árnyékokhoz használjunk sugárkövetést. Ezt a technikát használják például az Unreal Engine-ben is puha árnyékok számítására. [17] A 8.3 táblázatban egy, az előzőektől független teszt eredményei láthatók. A második oszlopba egy képkocka kirajzolásához szükséges átlagos GPU-idő került.

Generálási módszer	Átlagos GPU idő (ms)
Lipschitz	0,72
Brute force	0,51
AdaMax	0,50
Háromszögelés + AdaMax	0,24

8.3. táblázat. Sugárkövetés helyettesítése raszterizációval

## 9. fejezet

# Összefoglalás

A dolgozatomban bemutattam és implementáltam a távolságmezők generálásának alapvető módszereit, majd áttekintettem a parametrikus- és Bézier-felületek elméleti hátterét. Fő eredményként bemutattam, hogyan használhatók gradiens módszerek távolságmezők generálásához és szimulációval validáltam az AdaMax algoritmus alkalmazhatóságát köbös Bézier felületek távolságmezőjének generálására. Kitértem a módszer GPU implementációjának részleteire, mellyel nagyságrendekkel gyorsítottam a kiértékelést. Végül mérésekkel igazoltam, hogy az új módszer már kis felbontáson is lényegesen gyorsabb, mint a referencia implementáció.

# Köszönetnyilvánítás

Köszönet illeti a témavezetőmet, Bán Róbertet az elméleti háttér és a program kidolgozásában nyújtott segítségéért.

A Kulturális és Innovációs Minisztérium ÚNKP-22-6 kódszámú Új Nemzeti Kiválóság Programjának a Nemzeti Kutatási, Fejlesztési és Innovációs Alapból finanszírozott szakmai támogatásával készült.



# Irodalomjegyzék

- [1] John C. Hart. „Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces”. *The Visual Computer* 12.10 (1996. dec.), 527–545. old. DOI: 10.1007/s003710050084.
- [2] Benjamin Keinert és tsai. „Enhanced Sphere Tracing”. *Smart Tools and Applications in Graphics*. 2014.
- [3] Csaba Bálint és Gábor Valasek. „Accelerating Sphere Tracing”. *EG 2018 - Short Papers*. Szerk. Olga Diamanti és Amir Vaxman. The Eurographics Association, 2018. DOI: 10.2312/egs.20181037.
- [4] Csaba Bálint és Mátyás Kiglics. „Quadric tracing: A geometric method for accelerated sphere tracing of implicit surfaces”. *Acta Cybernetica* 25 (2021. jan.). DOI: 10.14232/actacyb.290007.
- [5] Róbert Bán és Gábor Valasek. „Automatic Step Size Relaxation in Sphere Tracing”. *Eurographics 2023 - Short Papers*. Szerk. Vahid Babaei és Melina Skouras. The Eurographics Association, 2023. ISBN: 978-3-03868-209-7. DOI: 10.2312/egs.20231014.
- [6] Herman Hansson Söderlund, Alex Evans és Tomas Akenine-Möller. „Ray Tracing of Signed Distance Function Grids”. *Journal of Computer Graphics Techniques (JCGT)* 11.3 (2022. szept.), 94–113. old. ISSN: 2331-7418. URL: <http://jcgt.org/published/0011/03/06/>.
- [7] Bálint Csaba. „Távolságfüggvényekkel definiált felületek interaktív megjelenítése”. *Országos Tudományos Diákköri Konferencia* (2016).
- [8] Morgan McGuire. *Ray Marching*. URL: [https://graphicscodex.courses.nvidia.com/app.html?page=\\_rn\\_rayMrch](https://graphicscodex.courses.nvidia.com/app.html?page=_rn_rayMrch) (elérés dátuma 2023. 05. 12.).
- [9] Inigo Quilez. *Distance functions*. URL: <https://iquilezles.org/articles/distfunctions/> (elérés dátuma 2023. 05. 10.).

- [10] Gerald Farin. *Curves and Surfaces for CAGD: A Practical Guide*. Computer graphics and geometric modeling. Elsevier Science, 2002. ISBN: 9781558607378. URL: <https://books.google.hu/books?id=D0qGMAwSUkEC>.
- [11] James F. Blinn. „Models of light reflection for computer synthesized pictures”. *ACM SIGGRAPH Computer Graphics* 11.2 (1977. júl.), 192–198. old. DOI: 10.1145/965141.563893. URL: <https://doi.org/10.1145/965141.563893>.
- [12] Róbert Bán, Csaba Bálint és Gábor Valasek. „Area Lights in Signed Distance Function Scenes”. 2019. máj. DOI: 10.2312/egs.20191021.
- [13] Miles Macklin és tsai. „Local Optimization for Robust Signed Distance Field Collision”. 3.1 (2020. máj.). DOI: 10.1145/3384538. URL: <https://doi.org/10.1145/3384538>.
- [14] C. Lemaréchal. „Cauchy and the Gradient Method”. *Documenta Mathematica* (2012), 251–254. old. URL: [https://www.math.uni-bielefeld.de/documenta/vol-ismf/40\\_lemarechal-claude.pdf](https://www.math.uni-bielefeld.de/documenta/vol-ismf/40_lemarechal-claude.pdf).
- [15] Diederik P. Kingma és Jimmy Ba. „Adam: A Method for Stochastic Optimization”. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Szerk. Yoshua Bengio és Yann LeCun. 2015. URL: <http://arxiv.org/abs/1412.6980>.
- [16] Niculae Vlad. *Optimizing with constraints: reparametrization and geometry*. URL: <https://vene.ro/blog/mirror-descent.html> (elérés dátuma 2023. 05. 18.).
- [17] *Distance Field Soft Shadows*. URL: <https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/RayTracedDistanceFieldShadowing/> (elérés dátuma 2023. 05. 18.).

# Ábrák jegyzéke

3.1. $h(u, v) = \sin(u + v) + 1$ grafikon a GeoGebra alkalmazásban. . . . .	9
3.2. Bézier-felület kiértékelése [10, 248. o.] . . . . .	11
4.1. Tórusz és függvénygrafikon háromszögelése. Forrás: wikiwand.com . .	14
4.2. Megjelenítési módok összehasonlítása. Forrás: blogs.nvidia.com . . . .	15
4.3. Gradiens módszer divergenciája túl nagy tanulási ráta esetén . . . . .	18
6.1. Stabil paraméterezés a Rosenbrock függvényen . . . . .	23
6.2. Példa sarokpontok szükségességére . . . . .	24
6.3. Példa oldalpontok szükségességére . . . . .	25
6.4. Felület és ponthalmaz képe . . . . .	26
6.5. Referencia érték maximális hibája a legnagyobb felbontáshoz képest .	26
6.6. Abszolút hiba maximuma a lépésszám függvényében, logaritmikus skálán. . . . .	27
6.7. Abszolút hiba maximuma az iteratív módszerhez viszonyítva. . . . .	27
8.1. Bézier-felület . . . . .	34
8.2. A generálás ideje a felbontás függvényében, milliszekundumban . . . .	36
8.3. Tesszellált és sugárkövetett felület egyszerre kirajzolva . . . . .	37