

TDK-dolgozat

Kiglics Mátyás

Finom felületi részletek megjelenítése általánosított cone map adatszerkezettel

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

ALGORITMUSOK ÉS ALKALMAZÁSAIK TANSZÉK



Szerző:

Kiglics Mátyás

programtervező informatikus MSc

3. évfolyam

Témavezetők:

Valasek Gábor

egyetemi adjunktus

Bálint Csaba

tanársegéd

Budapest, 2023

Tartalomjegyzék

1. Bevezetés	3
1.1. Makro- és mezogeometria	3
1.2. Mezogeometria megjelenítése	4
1.2.1. Bump mapping és normal mapping	4
1.2.2. Displacement mapping	5
1.2.3. Parallax mapping	6
2. Részletgazdag felületvizualizáció sugárkövetéssel	7
2.1. Per-pixel displacement mapping	7
2.2. Lineáris keresés	8
2.3. Cone step mapping	9
2.3.1. Előfeldolgozás	10
2.3.2. Megjelenítés	11
2.4. Relaxed cone step mapping	12
3. Másodrendű felülettérkép	15
3.1. Konzervatív forgáspárolák	15
3.1.1. Renderelés	15
3.1.2. Parabólák előállítás	16
3.2. Eltolts forgáspárolák	17
3.2.1. Forgásfelület definíció	17
3.3. Előfeldolgozás, forgásfelületek generálása	18
3.3.1. Radiális függvény előállítás	19
3.3.2. Forgásfelület paramétereinek meghatározása	20
3.3.3. Generálás optimalizálása	23
3.4. Renderelés forgásfelületekkel	24
3.4.1. Másodfokú egyenlet numerikus stabilizálása	27

4. Eredmények és összefoglalás	28
4.1. Implementáció és tesztelés	28
4.2. Eredmények	29
4.2.1. Lépésszám vizsgálata	30
4.2.2. Renderelési sebesség vizsgálata	32
4.3. Összefoglalás	34
4.3.1. További kutatási irányok	34
Köszönetnyilvánítás	36
Irodalomjegyzék	36
Ábrajegyzék	39
Algoritmusjegyzék	40

1. fejezet

Bevezetés

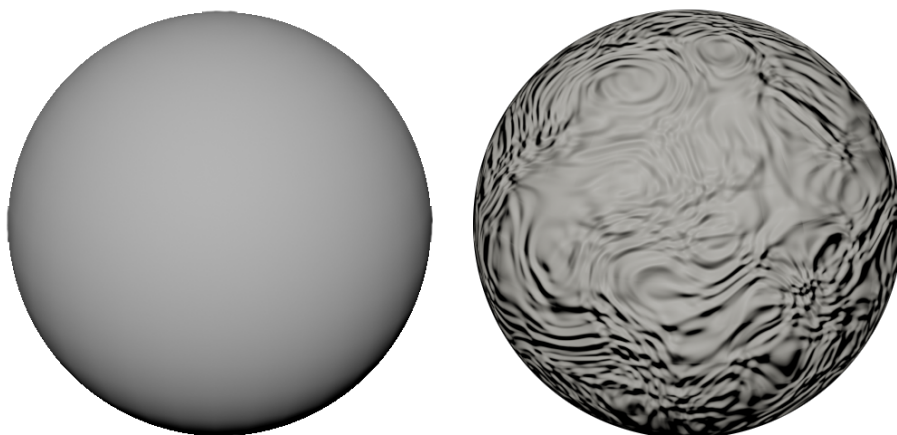
A dolgozat témája egy saját fejlesztésű, finom felületek megjelenítését hatékonyan megvalósító algoritmus ismertetése. Ebben a fejezetben a számítógépes grafika releváns területei kerülnek bemutatásra, majd részletesen kifejtjük a napjainkban is használt módszereket, azok működését, előnyeit. Ezután a 3. fejezetben a saját eljárás teljeskörű ismertetése következik, végül a dolgozat utolsó szakaszában az elvégzett mérések és összehasonlítások találhatók.

1.1. Makro- és mezogeometria

A valós idejű háromdimenziós képszintézisben valósághű színterek megjelenítésekor kiemelten fontos szerepet játszik a hatékonyság. A hardverek rohamos fejlődésével párhuzamosan az előállított képek minőségére vonatkozó elvárások is egyre nagyobbak lettek, ezzel folyamatosan újabb optimalizálási módszerekre igényt teremtve. Különböző részletességű felületek megjelenítésére eltérő módszerek alakultak ki.

A szakirodalomban általában három szintre osztják a geometriákat azok mérete alapján[1]. A legnagyobb kiterjedésű makrogeometriákat legtöbbször csúcspontokkal, valamint azokból összeállított háromszögekkel definiáljuk, ennek megfelelően síklapokkal határolt felületeket jelenítünk meg. Ez évtizedek óta rendkívül hatékonyan megvalósítható erre a célra optimalizált hardverek, a GPU-k használatával.

Gyakran a felszín olyan apró tulajdonságait is modellezzük, amelyeket a geometria módosításával nem lehetséges hatékonyan megjeleníteni, ugyanis pixeleknél kisebb részleteket szimulálnánk. Ilyen mikrogeometriai részlet lehet például a felület érdessége, ame-



1.1. ábra. A bal oldalon egy sima geometria, jobb oldalon ennek a zajjal generált bump map alkalmazásával való megjelenítése. Megfigyelhető, hogy a jobb oldali gömb sziluettje megegyezik az eredetivel.

lyet nem explicit jelenítünk meg, hanem a pixelek színének meghatározásakor vesszük figyelembe a fények hatásainak kiszámításához[2, 3].

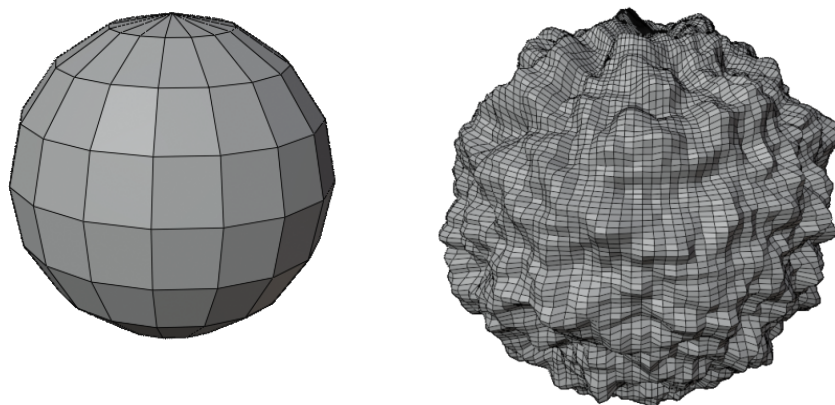
A két szint közötti, néhány pixelt lefedő részleteket nevezzük mezogeometriának. Általában túl összetettek ahhoz, hogy sokszöglapokkal hatékonyan jelenítsük meg őket, ezért számos különböző módszer terjedt el renderelésükre, a dolgozat további részében ilyen eljárásokat elemzünk.

1.2. Mezogeometria megjelenítése

Mezogeometria alatt apró, azonban még látható, néhány pixelt lefedő részleteket értünk. Míg makrogeometriákat legtöbbször triviális módon jelenítünk meg mai hardverekkel, finomabb részletekhez igényektől függően számos technika ismert.

1.2.1. Bump mapping és normal mapping

Blinn[4] bevezetett egy bump mapping-ként ismert eljárást, amellyel a geometria módosítása nélkül jeleníthetők meg részletgazdagnak ható felületek. A háromszögek színének meghatározásakor egy bump map-nek nevezett textúrában tárolt irányok alapján a felület normálisait torzítjuk, ezzel az árnyalással térbeli hatást keltve (1.1. ábra). A normal mapping technika[5] hasonló elven működik, a torzítás helyett azonban a tényleges normálist olvasva egy textúrából, annak színtérbeli irányát a geometria orientációja alapján számítva.

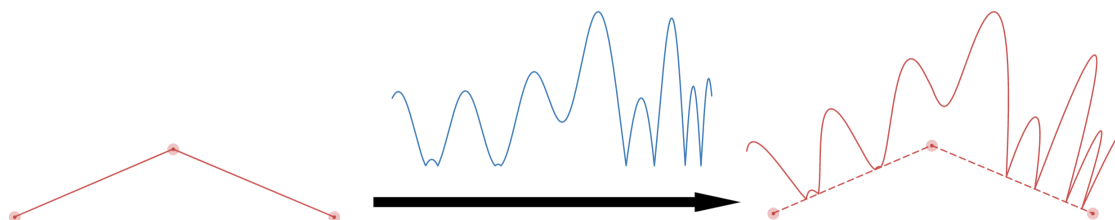


1.2. ábra. A bal oldalon egy egyszerű geometria a síklapok megjelenítésével, jobb oldalon ennek a felbontása, majd magasságtérkép alapú transzformáltja az új vertexekkel.

Felületi normálisok változtatása erőforráskímélő, azonban a geometria körvonalát, árnyékát nem változtatja meg, ezért sokszor nem elegendő a valósághű felületábrázoláshoz.

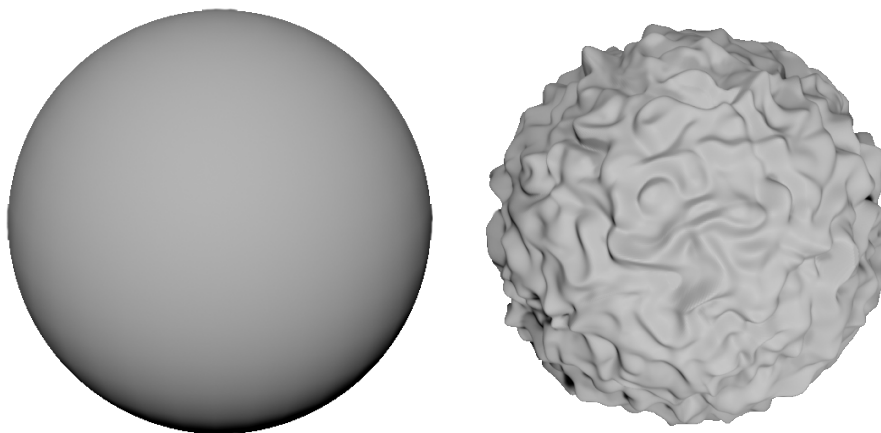
1.2.2. Displacement mapping

Léteznek a normális helyett az ábrázolt felület pontjait transzformáló módszerek is, melyek a tárolt geometria síkjához viszonyítva azok pontjait a normális irányba mentén való elmozdításával definiálják a részletesebb felületet. Az eltolás mértékét egy magasságtérképnek nevezett kétdimenziós, legtöbbször egycsatornás textúrában tároljuk, melynek egyes texeleinek értéke jelenti a távolságot a geometria síkjától (1.3. ábra). A magasságtérképben eltárolható a transzformált pontbeli normálvektor iránya, illetve a szomszédos értékekből is kiszámítható.



1.3. ábra. Geometria és ahhoz rendelt magasságtérképpel a transzformáltja. Az eredeti (piros) felület pontjait eltoljuk az eredeti normális irányába a magasságtérképben (kék) tárolt

A magasságtérképek felhasználhatóak a geometria tényleges módosítására. Ehhez felosztjuk a háromszögeket kisebb egységekre[6], majd az így kapott csúcspontokat transz-



1.4. ábra. A bal oldalon egy sima geometria, jobb oldalon ennek a zajjal generált magasságtérképpel transzformált megjelenítése. A széleken a gömb bizonyos pontjait eldobjuk, azonban a geometrika nem változik.

formáljuk (1.2. ábra). A kapott felület hagyományos módon megjeleníthető, azonban nagy részletességű felbontás használata jelentős extra erőforrást jelent mind memória, mind számítási kapacitás szempontjából.

1.2.3. Parallax mapping

Magasságtérképpel megadott mezogeometriák megjeleníthetők a tárolt vertexek módosítása nélkül. A parallax mapping[7] néven bevezetett módszer a normal mapping-hez hasonló technikával, pixelenkénti árnyalással megvalósítható, azonban tényleges térhatású felületet eredményez (1.4. ábra).

A geometriát megőrző technikák hatékonyan használhatók, azonban alacsony szögből renderelve torzul a térhatás jelensége, valamint árnyékszámítás nehezen megvalósítható a felület részletein. Ezen javíthatnak bizonyos sugárkövető algoritmusokat alkalmazó technikák, melyek számításigénye magasabb, azonban megfelelő gyorsító struktúrák felhasználásával valós idejű megjelenítésre alkalmasak. A következő fejezetben ilyen módszerek kerülnek bemutatásra.

2. fejezet

Részletgazdag felületvizualizáció sugárkövetéssel

Ebben a fejezetben három olyan módszert mutatok be, amely a geometriai komplexitás növelése nélkül teszi lehetővé részletgazdag felületek megjelenítését. Valós idejű vizualizációkban gyakran alkalmazzák ezeket az úgynevezett per-pixel displacement mapping technikákat az előző fejezetben részletezett előnyeik miatt.

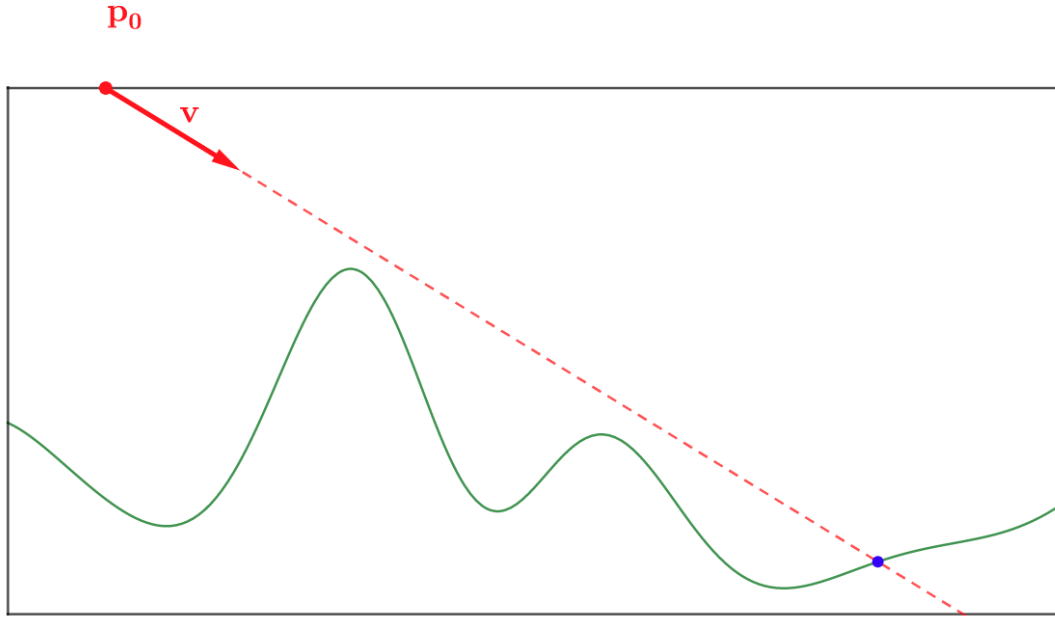
A három algoritmus mindegyike iteratív módon keresi egy sugár és egy felület metszéspontját, azonban lényeges különbség van közöttük teljesítmény és pontosság terén.

2.1. Per-pixel displacement mapping

A technika lényege, hogy a magasságtérképben definiált transzformációt a grafikus pipeline utolsó lépésében, a raszterizáció után hajtjuk végre. Ekkor már a kimenetre, általában a frame bufferbe kerülő egyes pixelek színét határozzuk meg a GPU-n futó speciális programmal, a fragment shaderrel.

Szemléletesen a probléma a geometria háromszögein egy térbeli felület képének megjelenítése, amely megvalósítható valamilyen sugárkövető módszer felhasználásával.

Ha a transzformált felületet a geometria háromszögei alatt értelmezzük, akkor annak az adott pontjából a kamera irányából húzott félegyenes által meghatározott sugárral a tárolt geometria belsejében keressük a legközelebbi metszéspontot a felülettel, ahogy a 2.1. ábrán látható. A metszésponthoz tartozó textúrákoordináta alapján meghatározható a felületi normális a környező magasságértékekből számolva vagy textúrából kiolvasva,



2.1. ábra. A sugárkövetés probléma displacement mapping esetén kétdimenziós formában. A $\mathbf{p}_0$ pontból induló $\mathbf{v}$ irányú piros sugár metszéspontját keressük a zölddel jelölt magasságtérképből definiált felülettel. A felső vízszintes egyenes a transzformálandó geometria síkja.

amennyiben rendelkezésre áll. A normális és a pozíció ismeretében fényhatásokat tudunk számolni részletes, térbeli hatást keltve [8].

Formalizálva, a sugarat $\mathbf{p}_0$ kezdőponttal és $\mathbf{v}$ iránnyal definiáljuk, melyre $\|\mathbf{v}\| = 1$. Ekkor felírhatók a félegyenes pontjai $\mathbf{s}(t) = \mathbf{p}_0 + t\mathbf{v}$ alakban, ahol $t > 0$ valós távolság a kezdőponttól. A magasságmezőt $y = h(u, v)$ explicit függvény írja le, amelynek értelmezési tartománya általában a $[0, 1] \times [0, 1]$ halmaz, valamint értékkészlete a $[0, 1]$ intervallum.

A gyökkeresés problémáját olyan t paraméter kereséseként definiáljuk, amelyre igaz, hogy $y = h(\mathbf{s}(t)_x, \mathbf{s}(t)_y)$, ahol $\mathbf{s}(t)_x$ és $\mathbf{s}(t)_y$ a sugár t -hez tartozó pontjának vízszintes koordinátái.

A metszéspont meghatározására a továbbiakban három ismert módszert mutatok be.

2.2. Lineáris keresés

Egy sugár felülettel való első metszéspontjának közelítése megoldható számos eljárással, ezek közül az egyik legegyszerűbb a lineáris keresés. Ennek során egy δ lépésköz meghatározása után minden iterációban δ nagyságú lépést teszünk a sugár mentén, egészen addig, amíg a felület belsejébe érünk (2.2. ábra, 1. Algoritmus). A felületbe való

belépés detektálása a magasságtérkép az akutális pont alatti texelének kiolvasásával történik. Amennyiben ennek az értéke nagyobb, mint a sugár pontjának magassága, akkor átléptünk a felülethatáron.

1. Algoritmus Lineáris keresés

Function LinearSearch(uv_{in}, uv_{out}) ▷ Textúrankoordináták a magasságmező tetején és alján

```

 $H \leftarrow 1$                                 ▷ Kezdeti magasság
 $H_{int} \leftarrow 0$                             ▷ Metszéspont magassága
for  $i = 0 \dots steps$  do
     $H \leftarrow H - \frac{1}{steps}$ 
     $uv \leftarrow uv_{in} \cdot H + uv_{out} \cdot (1 - H)$     ▷ Sugár pontja alatti texel koordinátái
     $m \leftarrow h(uv)$                                 ▷ Magasságtérkép értéke  $uv$ -ben
    if  $H_{int} = 0$  then
        if  $H \leq m$  then
             $H_{int} \leftarrow H$ 
        end if
    end if
end for
 $uv \leftarrow uv_{in} \cdot H_{int} + uv_{out} \cdot (1 - H_{int})$     ▷ Metszéspont koordinátái
return  $uv$ 

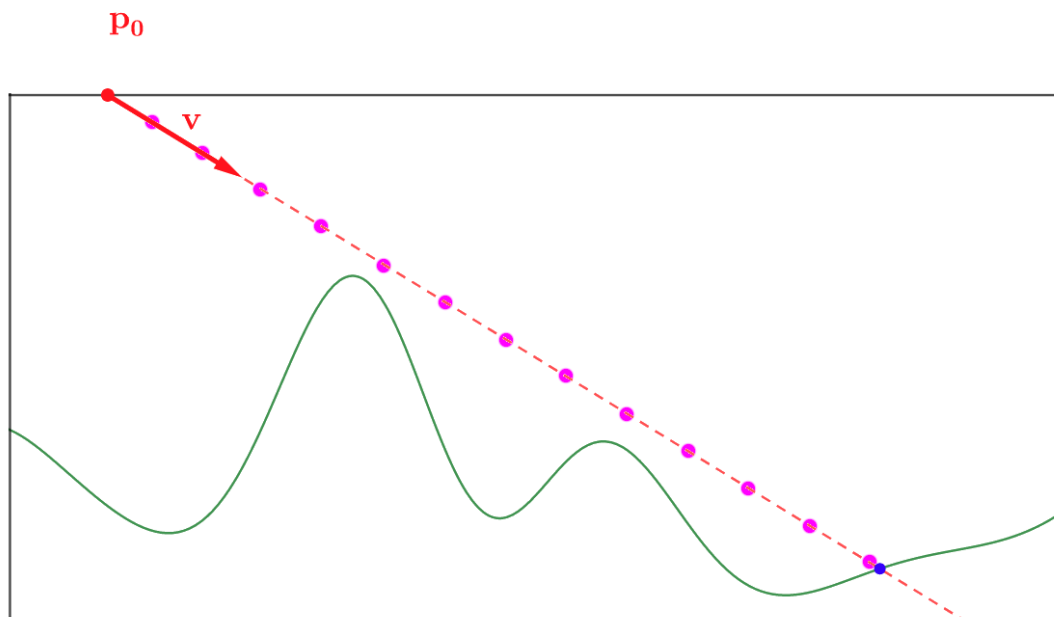
```

A módszer előnye, hogy gyorsan implementálható, megfelelően kicsi δ választásával jól közelíti a metszéspontot, valamint a renderelés nem igényel semmiféle előfeldolgozást vagy információt a magasságtérképről. Ellenben a pontosság feltételéből adódik, hogy sok apró lépést tesz a sugár mentén, így a megjelenítés sokszor kritikusan lelassul. Nagyobb lépésköz mellett a sebesség valamelyest javul, azonban esetenként kisebb felületrészeket átléphet, helytelen képet eredményezve (2.3. ábra).

2.3. Cone step mapping

Lineáris keresés során a módszer általános hatékonyságát rontja az a tényező, hogy egy-egy lépés közben mindössze egyetlen magasságértékből nyert információ áll rendelkezésre. Ezáltal a sugár jelenleg ismert pontjának tetszőlegesen kicsi környezetében lehet felületi pont, amelyet potenciális metszéspontként kezelünk. Nagyon kicsi lépésközzel haladva veszítünk a teljesítményből, míg ellenkező esetben zajos képet eredményeznek a nagy lépések által átlépett felületi pontok.

A Dummer[9] által bemutatott cone step mapping nevű eljárás egy gyorsító adatszerkezet definiál, amely a magasságtérkép fölé a felületet érintő, konzervatív kúpokat helyez,



2.2. ábra. A metszéspont (kék) lineáris kereséssel való közelítése. A piros sugár mentén teszünk egyenlő nagyságú lépéseket (lila pontok), amíg a felületbe (zöld) nem lépünk.

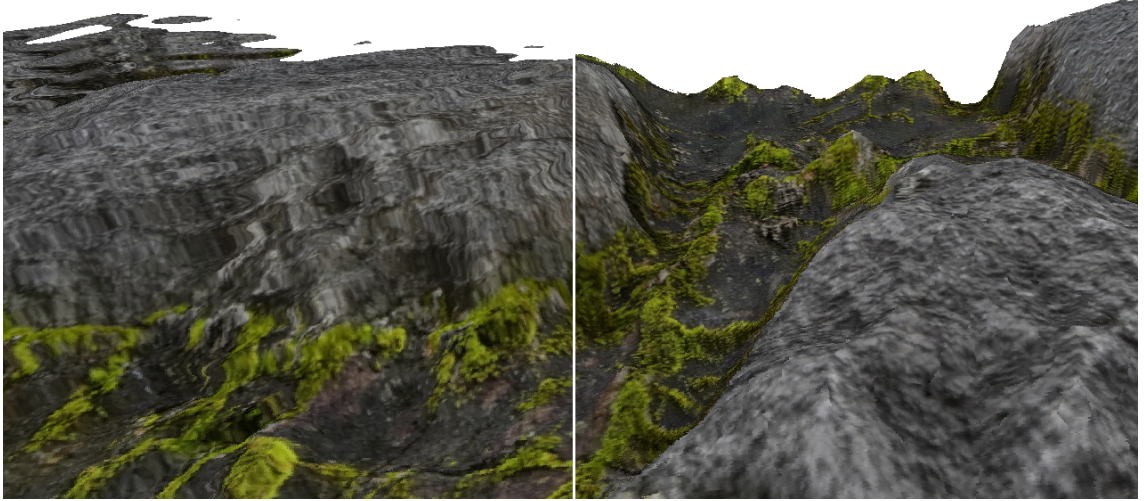
amelyek két részre osztják a teret, ezek egyikén a felület található. A konzervatív tulajdonság ebben az esetben arra utal, hogy a felületet felülről érintő geometriát adunk a kúpoknak, amelyek nem metszhetik azt. A magasságtérkép minden pontjához egy ilyen kúpot rendelve egy cone map-nek nevezett struktúrát kapunk, amelyet egy kétcsatornás textúrában eltárolunk.

2.3.1. Előfeldolgozás

A kúpok generálása erőforrásigényes feladat, ezért ezt a folyamatot a renderelés előtt hajtjuk végre, és az eredményét elmentjük. Ezeknek a közös tulajdonságai, hogy a csúcspontjuk a magasságmező felületén helyezkedik el, a tengelyük függőleges, valamint a nyílásszögük akkora, hogy legalább egy ponton érintsék a felületet (2.4. ábra).

Az algoritmus naiv módon implementálható. A csúcspont és a tengely minden textúrában adott, mindössze a palást nyílásszögét szükséges kiszámítani. Ehhez végigiterálunk minden pontján a magasságtérképnek, és vesszük azt, amelyhez a csúcspontból húzott egyenes meredeksége a legnagyobb, ez a meredekség adja meg a kúp szögét (2.5. ábra).

Látható, hogy $N \times N$ -es textúrákhoz N^4 egyenes meredekségét kell vizsgálni, amely részletes magasságtérképek esetén a GPU-n futó generálást is hamar lelassítja. Halli publikált $\Theta(N^2)$ -es algoritmust[10] a kúpok előállítására, azonban ez rendelkezik egy



2.3. ábra. 60 (bal oldal), valamint 120 lépéses lineáris kereséssel renderelt felületek. A bal oldali ábrán több helyen átléptünk metszéspontokat, ezek jól látható hibákat eredményeznek.

konstans szorzóval, amely magasságtérkép különböző értékeinek számával egyezik meg. Nemrég bemutatott nagyságrendekkel hatékonyabb, $\Theta(N^2 \log N)$ komplexitású eljárást mipmap-ek felhasználásával [11].

2.3.2. Megjelenítés

A sugár-felület metszéspontjának előállításához a korábban generált cone map-ben tárolt adatokat használjuk fel. A lineáris kereséshez hasonlóan a sugarat a geometria síkjából indítjuk, azonban a lépések nagyságát az aktuális pont alatti kúp határozza meg, minden iterációban a sugárnak ezzel való metszéspontjába lépünk (2. Algoritmus). Ezáltal garantáltan nem lépünk a felületbe, azonban általában nagyobb lépésekben tudunk haladni, így hamarabb metszésponthoz jutunk.

2. Algoritmus Cone stepping

Function ConeStep(uv_{in}, uv_{out}) $\triangleright$ Textúrákoordináták a magasságmező tetején és alján

$uvh \leftarrow [uv_{in}, 1]^T$

$l_v \leftarrow \|\mathbf{v}_{xy}\|$

for $i = 0 \dots steps$ **do**

$H \leftarrow h(\mathbf{uvh}_{xy})$

$m \leftarrow \mathbf{uvh}_{xy} - \text{ban tárolt kúp meredeksége}$

$dts \leftarrow \frac{m}{l_v + mv_z}$

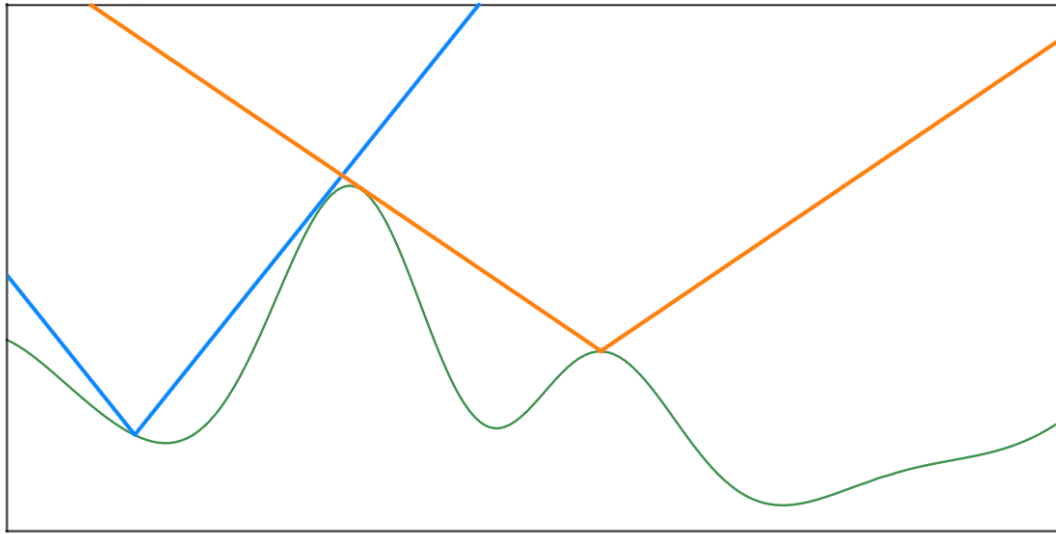
$\triangleright$ Sugár metszése kúppal

$dt \leftarrow dts(\mathbf{uvh}_z - h)$

end for

return uvh_{xy}

$\triangleright$ Metszéspont koordinátái



2.4. ábra. Példák konzervatív kúpok (kék, narancssárga) egy felületre (zöld) illesztve. A kúpok csúcsa illeszkedik a felületre, a palástjuk pedig érinti, a tengelyük pedig merőleges a geometrika síkjára (felső vízszintes egyenes). A felület minden pontjához definiálunk egy-egy ilyen kúpot, ezek közül kettőt jelenítettünk meg.

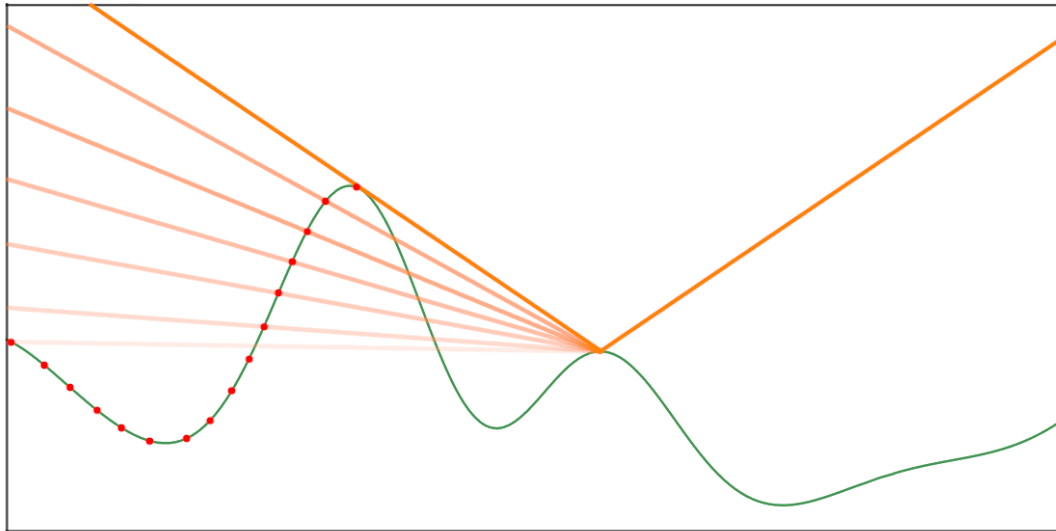
Figyelembe véve, hogy a kúpok forgásszimmetrikusak, valamint azt, hogy egy egyenes mentén keressük a felületet, a cone step mapping esetében is minden számítást elegendő a magasságtérkép megfelelő metszetén két dimenzióban végrehajtani.

Eredményképpen minden lépés egy textúraolvasásból és egy lineáris egyenlet megoldásából áll, amely összemérhető a lineáris keresés lépéseivel, azonban ezen kiértékelések száma jelentősen csökken, számottevően javítva a megjelenítés sebességén.

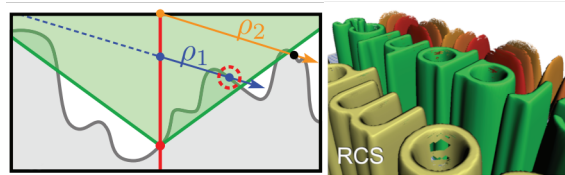
2.4. Relaxed cone step mapping

Az előző módszert gondolta tovább Policarpo és Oliveira[12], lecserélve a konzervatív kúpokat relaxáltra. Ezek nyílásszögére az a feltétel teljesül, hogy a cone step mapping-hez hasonlóan iterálva sugárkövetés során legfeljebb egyszer lépjük át a felület határát, engedélyezve, hogy belépünk abba (2.7. ábra). Fontos megjegyezni, hogy ez a feltétel egyedül a primitív síkjából induló sugarak esetén teljesül, alacsonyabbról indítva előfordulhat több metszéspont is a felülettel (2.6. ábra).

A módszer szükségessé teszi ugyan, hogy a leálláskor kapott pontból esetlegesen visszalépésekkel megkeressük a tényleges metszéspontot, azonban a kúpok nyílásszöge,



2.5. ábra. Egy konzervatív kúp generálásának menete. A zölddel jelölt felület texelein (piros) iterálva keressük a legnagyobb meredekségű érintő szakaszt. Egyre sötétebb egyenesek az aktuális maximumok.

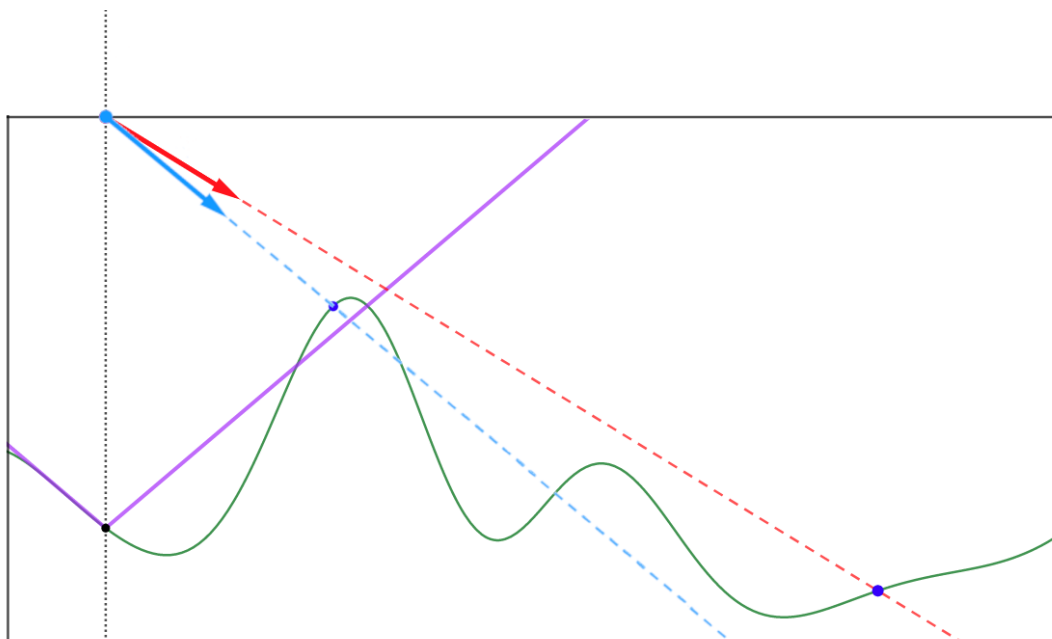


2.6. ábra. Relaxált kúpok metszése nem primitív síkjából indított sugarakkal. A ρ_2 sugár helyes metszéspontot ad, azonban a ρ_1 átlép a felületen, amely nem detektálható hibához vezet. Ilyen sugarak a jobb oldali ábrán látható eredményhez vezet. Forrás: Precomputed Safety Shapes for Efficient and Accurate Height-Field Rendering [13]

ebből kifolyólag az egyes lépések nagysága is, jelentős mértékben növekedhet. Ez általában teljesül, és a renderelés a konzervatív cone stepping-hez képest is gyorsabbá válik[1], jelenleg ez a state-of-the-art módszer teljesítmény szempontjából.

Relaxált kúpok generálása implementáció szempontjából bonyolultabb, mint a konzervatív esetben, futási időben azonban 10 – 20%-kal gyorsabb, mint a naiv módszer.

Fontos tulajdonsága ennek az algoritmusnak, hogy kizárólag a magasságmező tetejéből induló sugarakkal működik megfelelően, az alacsonyabbról indítottal való metszése a kúpoknak sokszor hibás eredményt ad bizonyos részleteire a felületnek. A konzervatív cone step mapping ezzel szemben mindkét esetben helyes megoldást ad, ezért a legtöbb összehasonlítást a későbbiekben a konzervatív módszerrel végzem.



2.7. ábra. Példa relaxált kúpra (lila) a magasságtérkép (zöld) egy pontjában. A késsel jelölt sugár kúpbal való metszéspontjába való lépés során a felület belsejébe kerülhetünk, azonban garantált, hogy legfeljebb egyszer lépjük át a felület határát, így adott esetben visszakereshető a metszéspont. A piros sugárral a konzervatív kúpokhoz hasonlóan iterálunk, a megetett lépések nagysága azonban itt is nagyobb lesz.

3. fejezet

Másodrendű felülettérkép

Ebben a fejezetben részletesen ismertetem a saját munkámat mind elméleti mind megvalósítási szempontból. Először egy egyszerűbb konstrukciót mutatok be konzervatív másodrendű felületeket definiálva, majd ebből kiindulva egy összetett adatszerkezet építek fel.

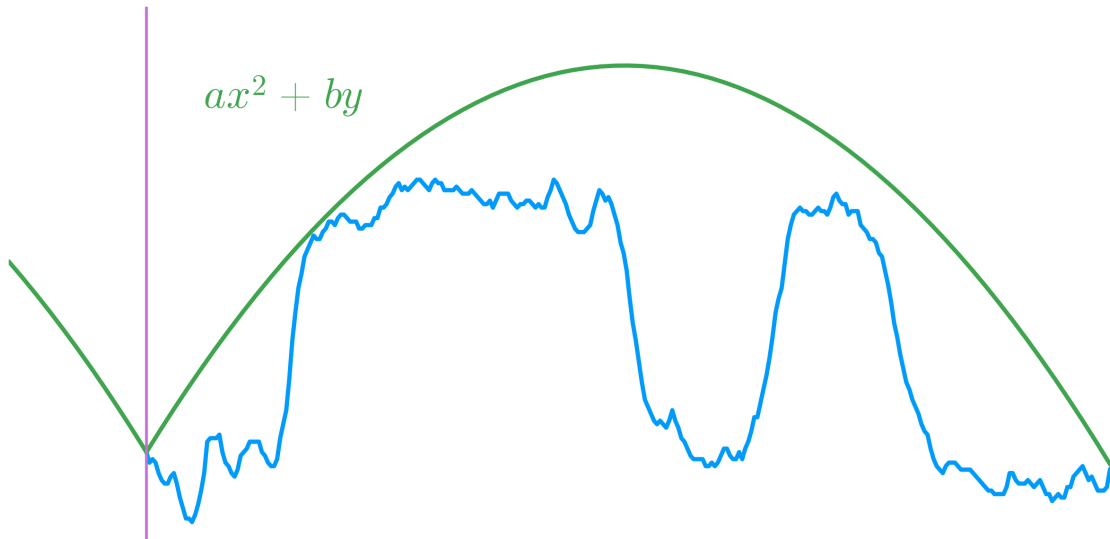
3.1. Konzervatív forgásparabolák

Az általam fejlesztett módszer a cone step mapping általánosításán alapszik. A konzervatív kúpok helyett egy azonos kritériumoknak megfelelő forgásparabolát definiálok a magasságtérkép pontjaihoz, azonban a nyílásszög helyett két, parabolát egyértelműen definiáló paramétert tárolok.

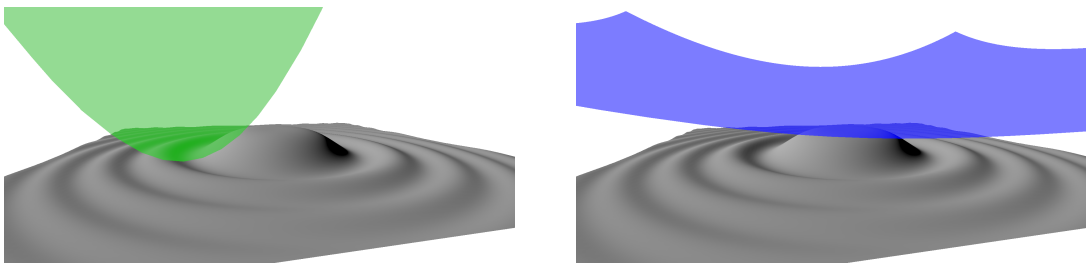
A célom egy iteráción belül nagyobb lépéseket elérni a renderelés közben[14], kihasználva, hogy a forgásparabolák a felülettől távolodva nagyobb üres térrészt fednek le (3.1. ábra).

3.1.1. Renderelés

A forgásparabolák felhasználására a szimmetriát kihasználva a cone stepping-hez hasonlóan megadható két dimenzióra visszavezethető algoritmus. A különbség az egyenes metszésének helyettesítése parabolával, amely egy másodrendű egyenlet megoldását jelenti. Ha létezik gyök, akkor a kisebb fogja az első metszéspontot, tehát a lépés nagyságát adni, ellenkező esetben nem szükséges több lépés, a sugár garantáltan divergál.



Síkbeli példa egy felülethez (kék) generált érintő parabolára (zöld). A függőleges lila egyenes a forgástengelyt jelöli.

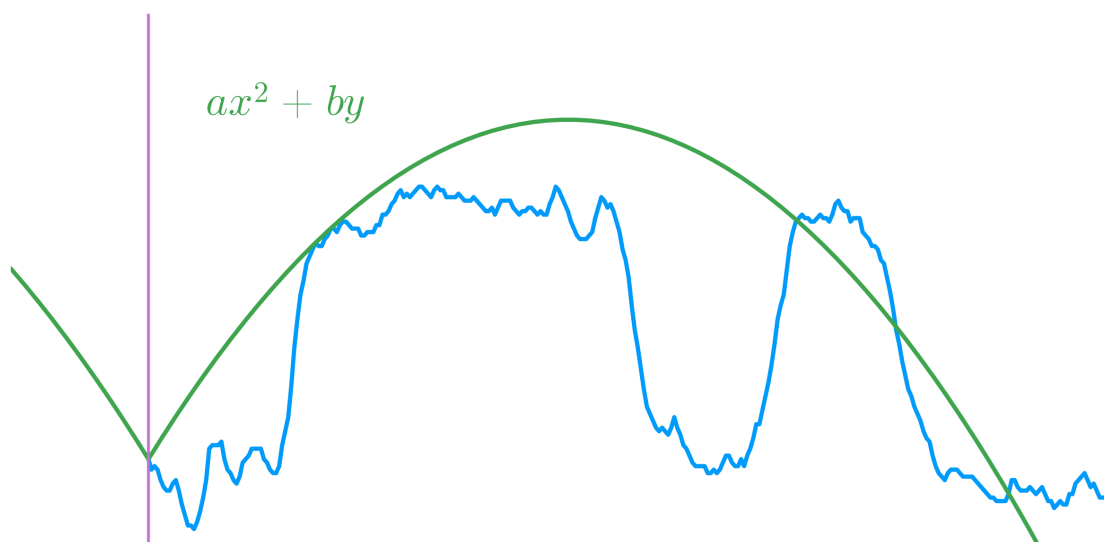


3.1. ábra. Példák konzervatív forgásparabolákra síkban (felső ábra) és három dimenzióban (alsó ábrák).

3.1.2. Parabolák előállítása

A szükséges tulajdonságokkal rendelkező forgásparabolák előállítása egy összetett probléma, ugyanis a kúpokkal szemben több különböző paraméter esetén is érintik a felületet. Emellett a magasságtérkép egészét figyelembe kell vennünk a görbület meghatározásakor, ugyanis felhasználunk konvex valamint nem konvex parabolákat is (3.2. ábra).

A végső forgásparabola kiválasztásakor a legoptimálisabbnak azt tekintettem, amely feletti térfogat a magasságmező felső határáig a legnagyobb.



3.2. ábra. Felületbe metsző parabola, amelyet kizárólag lokális információk alapján generálunk. Ehhez hasonló hibák elkerülése érdekében a teljes magasságtérképet figyelembe vesszük a paraméterek meghatározásához.

3.2. Eltolt forgáspárolák

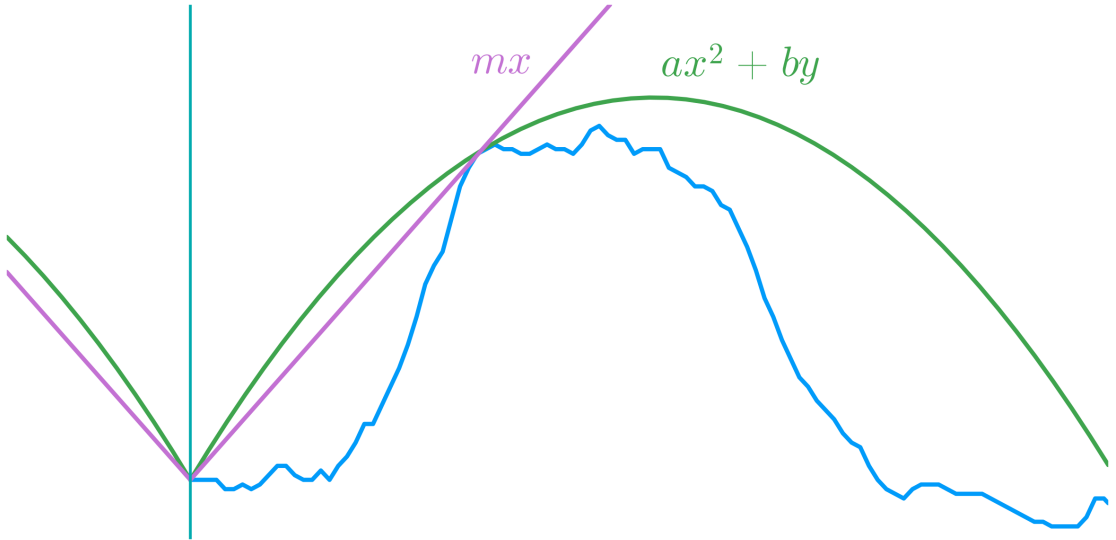
Az eddigiekben bemutatott koncepciót részletesen kidolgoztam, implementáltam és összehasonlítottam létező módszerekkel [15]. A kutatás során két nagyobb nehézség merült fel, az egyik a generálás lépéshez az ‘optimális’ parabola pontos definíciója, amellyel elkerülhető a heurisztika alapján történő kiválasztás. A másik a sugárkövetés konvergenciájának lelassulása a felület közelébe érve, amelyet a párolák geometriája miatt általában nem lehet elkerülni (3.3. ábra).

Ezen dolgozat fő témája a fenti technika továbbgondolása. Két részre osztjuk a forgáspárolákat, egy Dummer által leírt tulajdonságú kúpot számítunk, azonban egy maximális magasságot rendelünk hozzá, amely felett egy forgáspárolát illesztünk hozzá (3.4. ábra). A kúp minimális magasságát egy hiperparaméter határozza meg, amely megegyezik a generált felülettérkép összes pontjára.

Az így módon definiált konzervatív forgáspárolákat hasonló módon tudjuk alkalmazni sugárkövetés konvergencia-sebességének javítására, elkerülve a párolák felület közelében előforduló szűkebb alakjából adódó hátrányokat.

3.2.1. Forgáspárola definíció

A generálás részletezése előtt szükséges formalizálni a felület paramétereit, és az azokkal szembeni megkötéseket. A forgáspárolákat a továbbiakban azoknak egy kétdi-



3.3. ábra. Felület közelében kisebb lépésközt okozhatnak a parabolák a kúpokhoz képest, lassulást eredményezve a renderelésben.

menziós metszetével fogom bemutatni, ugyanis minden szükséges számítás elvégezhető ily módon egyszerűsített geometrián, a generálás és a renderelés során egyaránt.

A felületet három paraméterrel reprezentáljuk, ezek jelölése legyen $a, b, c \in \mathbb{R}$. Az első két érték jelentse egy pont relatív koordinátáit a texel pozíciójához képest, az ebbe a pontba húzott szakasz legyen a kúp palástjának egyik oldala. A harmadik, c paraméter egy parabolát határoz meg, amely átmegy az (a, b) ponton, valamint pontjai kielégítik a

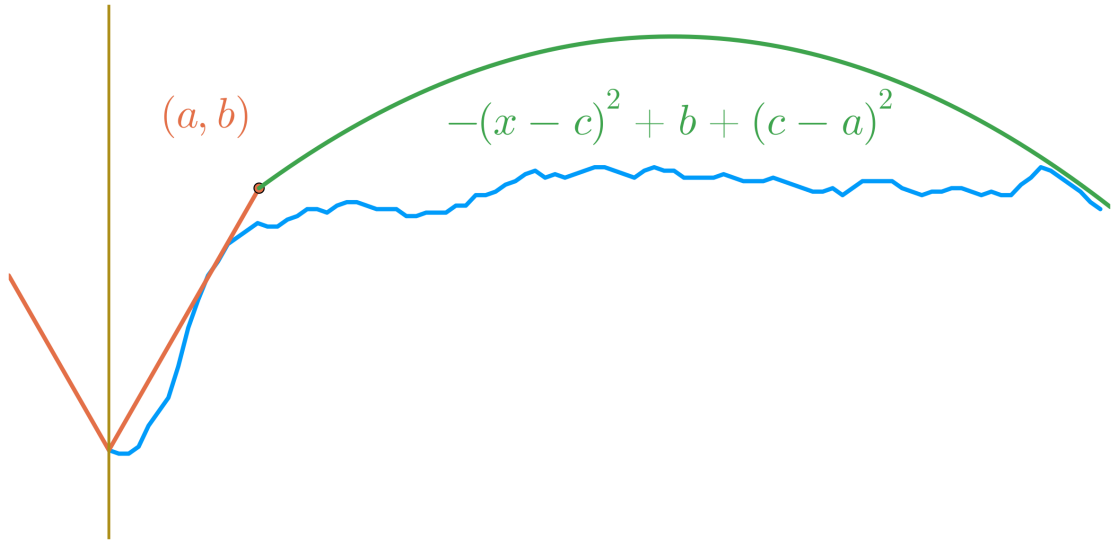
$$y = -(x - c)^2 + b + (c - a)^2 \quad (a, b \geq 0)$$

egyenletet. Ilyen görbék forgatásával kapott felületek láthatók a 3.5. ábrán.

Az egyenlet megválasztásának fő oka, hogy adott a, b paraméterek mellett a c érték változtatása a parabola maximumának helyét módosítja, a növelésével pedig $x > a$ pontban magasabb y értékeket vesz fel a görbe, ezzel leegyszerűsítve a legjobb paraméter meghatározását (3.6. ábra).

3.3. Előfeldolgozás, forgásfelületek generálása

A renderelés előtt a másodrendű felülettérkép előállítására és eltárolására kerítünk sort. Ez a lépés a jelenlegi algoritmus memória- és számításigénye miatt indokolt, azonban egy adott textúrához elegendő egyszer elvégezni, a későbbiekben a felhasználásával drasztikusan javítható a renderelés sebessége.



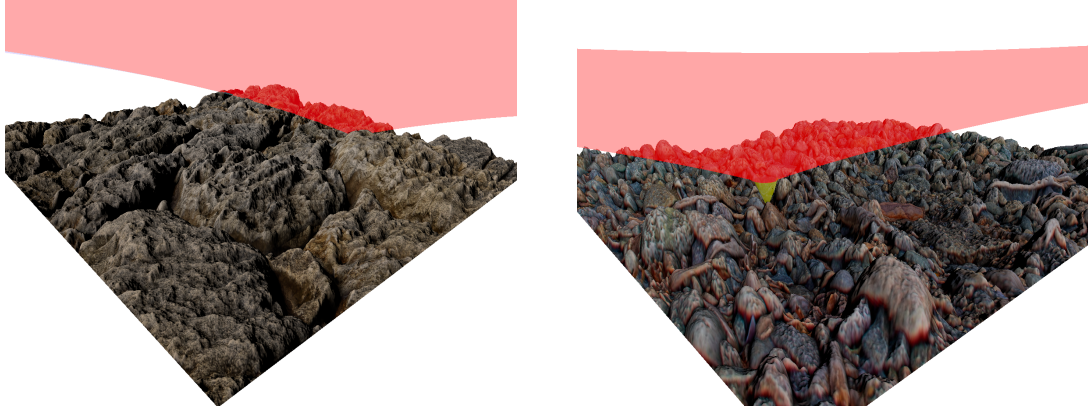
3.4. ábra. Kétdimenziós példa kúpra illesztett parabolára. A kúp és a parabola is oly módon van definiálva sárgával jelölt tengely körül a, b, c paraméterekkel, hogy érintse a felületet (kék görbe).

A cél a magasságtérkép minden texeléhez minél hatékonyabban meghatározni a forgásfelület a, b, c paramétereit. Ezek a számítások egymástól függetlennek tekinthetők, így a texelekre egy jól párhuzamosítható algoritmussal ki tudjuk használni a GPU adta számítási kapacitást.

A két részből előálló felületek a, b valamint c paramétereit két szakaszban számítjuk ki. Először a kúp paramétereit adjuk meg a texel környezetéből vett magasságértékek alapján, majd a rá illesztett parabolát a teljes magasságtérkép figyelembevételével. Naiv implementáció esetén mindkét lépésben egy adott texelre megvizsgálhatnánk a textúra összes értékét, amely sok ismételt memóriaolvasást jelentene. Ehelyett bevezetünk egy transzformációt, amellyel a kétdimenziós magasságtérkép leképezzük egy egydimenziós függvényre.

3.3.1. Radiális függvény előállítása

Forgásszimmetrikus felületek generálása során általában nincs szükség a teljes magasságtérképben tárolt összes adat ismeretére, hanem annak egy megfelelően tömörített reprezentációja is elegendő. A hatékonyabb előfeldolgozás érdekében definiálunk egy olyan függvényt minden texelhez, amely egy pozitív d számhoz hozzárendeli a ponttól vett d távolságú magasságértékek maximumát (3.7. ábra). Formalizálva, legyen tetszőleges uv



3.5. ábra. Példák kúpokhoz illesztett forgáspárolákra. Sárgával jelöltük a kúpot, pirossal a forgáspárolát.

3. Algoritmus Radiális függvény előállítás

Function GenHeightFun($h(x, y)$)

▷ Magasságtérkép

$$B \leftarrow \frac{B}{2}$$

$$M_1 \leftarrow C$$

$$M_2 \leftarrow -B + \sqrt{B^2 - AC}$$

$$x \leftarrow \frac{M_1}{M_2}$$

if $x \in \mathbb{R}$ **then**

return x

▷ A kisebb valós gyöke az $Ax^2 + Bx + C = 0$ egyenletnek

else

return ∞

▷ Nincs valós gyök, a következő lépés nagysága végtelen

end if

texelre

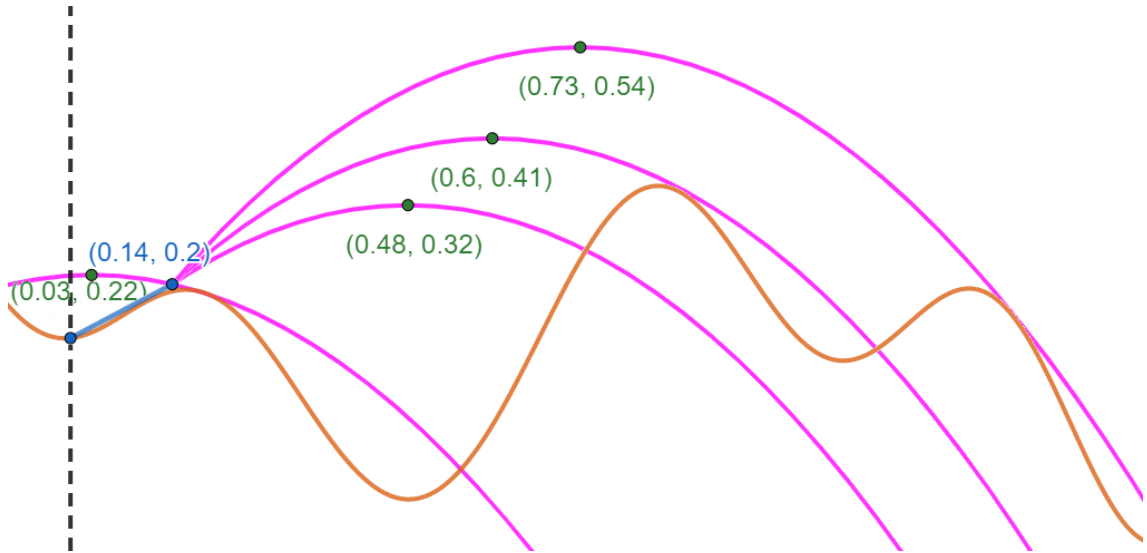
$$m_{uv}(d) = \max_{d \leq \|(x,y)-(u,v)\|_2 < d+1} \{h(x,y) - h(u,v)\} \quad (x,y) \in \mathcal{D}_h, d \in \mathbb{Z}^+$$

ahol $h(x, y) \in [0, 1]$ a magasságfüggvény.

Ilyen leképezéssel elegendő m_{uv} ismerete az a, b és c paraméterek meghatározásához. Ezeket az m_{uv} értékeket az előfeldolgozás első lépésében számítjuk ki minden texelre, majd eltároljuk a GPU memóriájában a későbbi hatékony kiolvasás érdekében.

3.3.2. Forgásfelület paramétereinek meghatározása

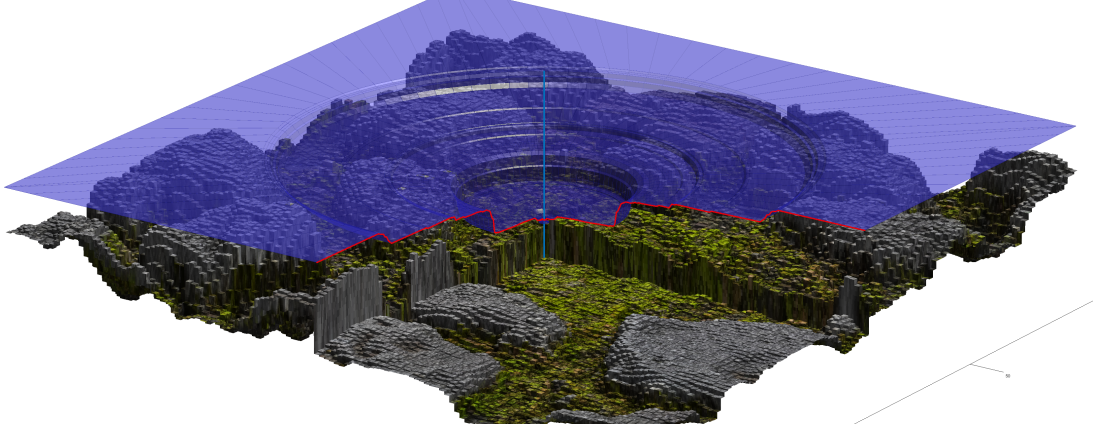
A korábban bevezetett a, b, c paraméterek az m_{uv} függvény felhasználásával kerülnek kiszámításra. Első lépésként bevezetünk egy $\alpha \in [0, 1]$ hiperparamétert, a minimális magasságát a felület alsó részét alkotó kúpnak, a texelhez rendelt magasságérték és a globális



3.6. ábra. Példa a parabola c paraméterének meghatározása. A kézzel jelölt szakasz végpontjából induló, magasságmezőt (narancssárga) érintő görbét keresünk. A zölddel jelölt értékek mutatják néhány vizsgált értékhez (növekvő sorrendben) tartozó parabola (lila) maximumát, amelynek a vízszintes koordinátája c -vel egyenlő. Ebben a példában az a, b, c paraméterek végső értéke rendre 0.14, 0.2 és 0.73 lesz.

maximum különbségének arányaként. Ez minden texelre azonos arány, azonban különböző magasságú pontokra más-más tényleges magasságot jelent. Ezáltal minden esetben megköveteltünk az algoritmustól valamekkora kúpot a forgásparabola alatt.

Az α ismeretében először az a, b paramétereket számítjuk ki. Ezek minden texelre egy abból kiinduló szakasznak a végpontjai, amely elforgatása adja a kúpot. Ehhez végigmegyünk m_{uv} értékein, majd a $(d, m_{uv}(d))$ végpontú szakaszok közül a legnagyobb meredekségűnek a végpontját vesszük addig, amíg $m_{uv}(d)$ kisebb, mint az α paraméter által definiált magasság.



3.7. ábra. Magasságtérkép transzformációja lineáris függvényre. A kék függőleges tengely jelzi a feldolgozott textelt, a piros görbe pedig az adott távolságú magasságértékek maximumát.

4. Algoritmus A forgásfelület paramétereinek előállítása

Function GenParam(α, m_{uv})

```

1:  $a \leftarrow 0; b \leftarrow 0; c \leftarrow 0$ 
2:  $d \leftarrow 1$ 
3:  $h \leftarrow 0$ 
4: while  $h < \alpha \wedge d < \text{size}(m_{uv})$  do                                ▷  $(a, b)$  pont meghatározása
5:    $b \leftarrow \max\{m_{uv}(d), h\}$ 
6:    $a \leftarrow d$ 
7:    $d \leftarrow d + 1$ 
8:    $h \leftarrow d \cdot \frac{b}{a}$                                             ▷ A szakasz meghosszabbításának magassága
9: end while
10: for  $i = d \dots \text{size}(m_{uv})$  do
11:    $x \leftarrow i$ 
12:    $y \leftarrow m_{uv}(i)$ 
13:    $c \leftarrow \max\{c, \frac{x^2 - a^2 - b + y}{2(x - a)}\}$                 ▷ Az  $(x, y)$  ponton átmenő görbe, vagy  $c$  maximuma
14: end for
15: return  $a, b, c$ 

```

Az eredményül kapott a, b paramétereket felhasználásával olyan c valós értéket keresünk, amellyel a

$$y = -(x - c)^2 + b + (c - a)^2 \quad (3.1)$$

függvénygörbe legfeljebb érinti, és nem metszi az m_{uv} függvény grafikonját. Tekintve, hogy a függvény globális maximuma c helyen van, ezért ennek az értékét növelve garantált, hogy a és c között a függvény minden értéke emelkedni fog, ezzel egyszerűvé téve az érintő feltételnek megfelelő paraméter keresését. Elegendő az m_{uv} minden pontjára illeszteni egy adott egyenletű parabolát, majd ezek közül kiválasztani a legnagyobb

globális maximummal rendelkezőt. Azaz, megoldva az

$$m_{uv}(j) = -(j - c_j)^2 + b + (c_j - a)^2 \quad j \in (a + 1, a + 2, \dots) \cap \mathcal{D}_{m_{uv}}$$

egyenleteket c_j -re, megkapjuk a keresett paramétert, mint $c = \max c_j$. A leírt eljárást formalizálja az Algoritmus 4.

A forgásfelület paramétereinek kiszámítása után azokat eltárolhatjuk a memóriában. Ezt egy magasságtérképpel megegyező felbontású, négycsatornás textúra felhasználásával tesszük, amelynek első három csatornájába az a, b, c értékek kerülnek, a negyedikbe pedig a $h(u, v)$ magasságot írjuk. Ezzel létrehoztuk a rendereléshez szükséges adatszerkezetet, az m_{uv} -nek és további segédváltozóknak lefoglalt memóriát felszabadíthatjuk.

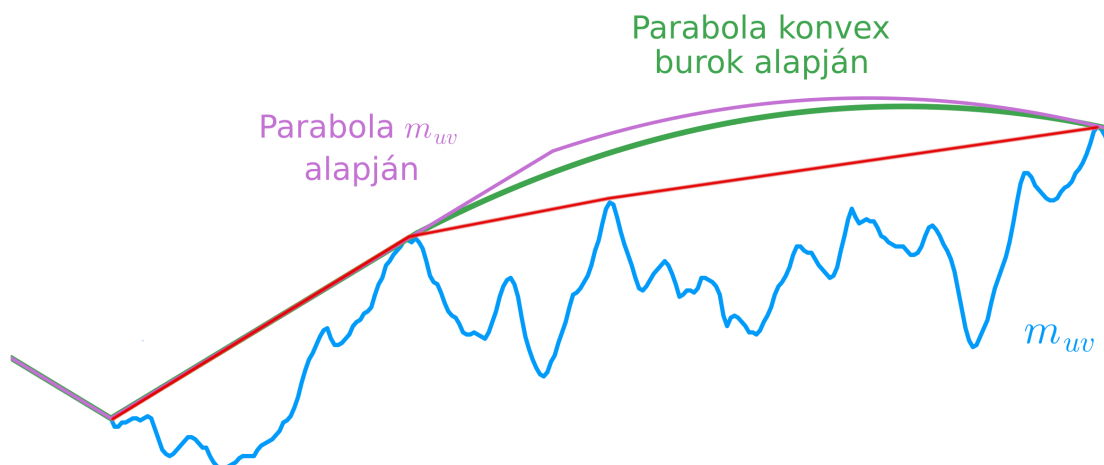
A bemutatott algoritmus komplexitása $N \times N$ -es textúra esetén a 2. fejezetben leírt módszerekhez hasonlóan $\Theta(N^4)$. Ez ugyan jelentősen lekorlátozza a generálható textúrák maximális méretét, azonban a gyakorlatban 2048×2048 -nál nagyobb felbontású magasságtérképeket általában nem használnak.

3.3.3. Generálás optimalizálása

Az előfeldolgozás mind memóriafelhasználás, mind számítási komplexitás szempontjából hatékonyabbá tehető. Az általában szűkös erőforrást, a GPU memóriáját, legjobban az m_{uv} értékeinek tárolása terheli. Erre egy nem ekvivalens eredményt adó (3.8. ábra) optimalizációt tudunk adni. A minden egész távolsághoz tartozó maximális magasság eltárolása helyett az m_{uv} görbéjének felső konvex burkát meghatározva egy általában számtottevően kevesebb ponttal leírható vonalat kapunk. Ez általános esetekben felbontástól függetlenül legfeljebb 15 – 20 csúcsú sokszög. A konvex burok előállítását a pontok számában lineáris [16], így ez a változtatás nem jár a számítási komplexitás emelkedésével.

A konvex burok használata továbbá gyorsabbá teszi az előfeldolgozás második szakaszát, nevezetesen az a, b, c paraméterek keresését. Ekkor ugyanis az m_{uv} minden értékének vizsgálata helyett mindössze a konvex burok csúcsain szükséges végigiterálunk, amellyel számos memóriaelérést meg tudunk spórolni.

Egy másik kritikus szempont a textúra méretében $\Theta(N^4)$ -es futási idő. Ennek csökkentéséhez az m_{uv} olyan előállítására van szükség, amely nem igényli az összes texelen történő iterációt, azonban ennek feltétele egy újabb segédstruktúra bevezetése és kiszámítása. Egy maximum mipmap-et használó technika már létezik erre [11], amelyet jelen



3.8. ábra. Lehetséges különbségek konvex burok (piros) és m_{uv} felhasználásával generált görbékre. Nem minden esetben van különbség a kettő között, valamint előfordulhat olyan eredmény, amikor az eredeti algoritmus generál alacsonyabb görbét.

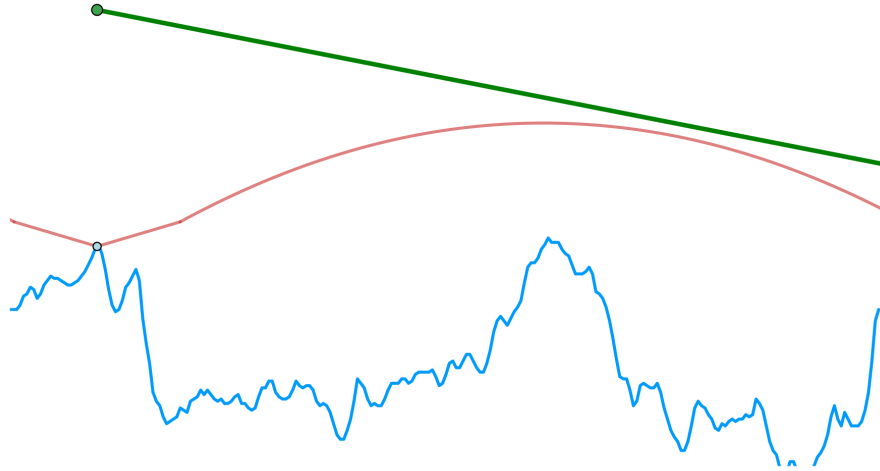
algoritmushoz is lehet adoptálni, azonban ennek implementációja a dolgozat megírásakor még folyamatban van.

3.4. Renderelés forgásfelületekkel

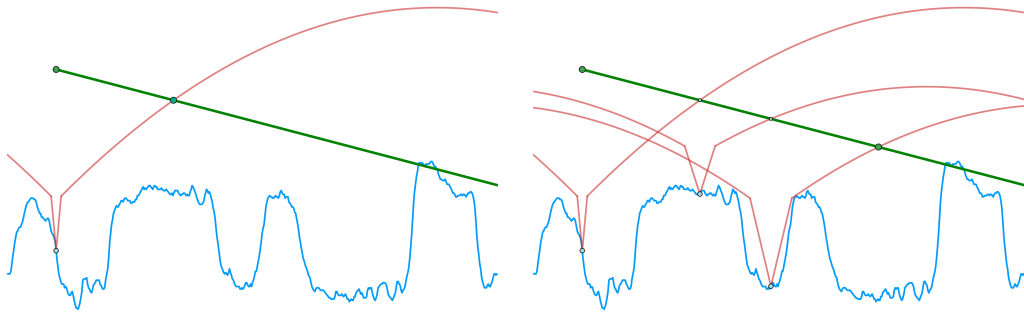
A megjelenítés során a GPU-n tárolt, háromszögekből kirajzolt durva geometria pixelen hajtunk végre olyan számításokat, amelyekkel a felületnek egy magasságtérképben tárolt információ alapján való transzformáltját jelenítjük meg. Ehhez egy sugarat definiálunk minden látható pixelre, amelyet a kamerából érkező irány, valamint a pixel mint kiindulópont definiál. Ezek a sugarak a geometria belseje felé mutatnak, ahol a magasságmezővel való metszéspontot keressük. Ezt a sugárkövetési problémát a korábbi algoritmusokhoz hasonlóan kétdimenziós formában definiáljuk, forgásfelületek szimmetriája miatt, továbbá a síkmetszet két parabolája közül elegendő az egyiket vizsgálni minden esetben, ugyanis feltehetjük, hogy ismerjük a sugár haladási irányának előjelét.

A 3.3. fejezetben definiált gyorsító felülettérképpel a sugárkövetés iterációs számának csökkentését, ezáltal a megjelenítés sebességének javítását szeretnénk elérni. A generált struktúrát úgy választottam meg, hogy a forgástengely valamely pontjából indított sugárral való metszés hatékonyan számítható legyen.

A metszéspont keresésének algoritmusai hasonló a cone step mapping módszeréhez, minden lépésben a jelenlegi pontnak megfelelő texelben tárolt forgásfelületnek a sugárral



A sugárnak nincs metszéspontja a parabolával, ebben az esetben tetszőlegesen nagyot lépünk



3.9. ábra. Sugárkövetési lépések a generált forgásfelületek (piros) felhasználásával. Amennyiben létezik a zölddel jelölt sugárral metszéspontja, úgy a következő iterációban ebből a pontból haladunk tovább.

a legközelebbi közös pontjába lépünk. Ennek létezése a mi esetünkben nem garantált, ekkor egy tetszőlegesen nagy lépést tehetünk (3.9. ábra).

Legyen a sugár $\mathbf{p}_0$ pontból kiinduló, $\mathbf{v}$ irányú, valamint pontjait a $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$ parametrikus egyenlet írja le, ahol $t \geq 0$. Az i . lépésben a paraméter legyen t_i . Minden iterációban kiolvassuk $\mathbf{p}(t_i)$ alatti texelből az a, b, c, h értékeket, ahol h a magasság, majd kiszámítjuk a $t_{i+1} > t_i$ paramétert a sugár és a négy paraméter által meghatározott forgásfelület metszéspontjából.

A metszéspontot külön keressük a két részén a felületnek, először az alsó kúppal vizsgáljuk meg, hogy létezik-e, ezt egy szakasz és egy egyenes közös pontjának kiszámításával tesszük. Legyen $\mathbf{s}(\beta) = \beta \cdot (a, b)$, ahol $\mathbf{s}(\beta)$ pontosan akkor van rajta a szakaszon, ha

$\beta \in [0, 1]$. Jelölje $\mathbf{p}(t)$ és $\mathbf{s}(\beta)$ metszéspontját (x, y) , amelyet a két egyenes egyenletéből triviálisan kapunk, ekkor $\beta = \frac{x}{a}$. Amennyiben $\beta \leq 1$, a kúppal van metszéspontja a sugárnak, így $t_{i+1} = \|\mathbf{p}(t_i) - (x, y)\|$. Ellenkező esetben, azaz ha $\beta > 1$, a parabolával való metszést keressük $\mathbf{p}_0 + t_i \mathbf{v}$ behelyettesítésével a 3.3.2. egyenletbe:

$$\mathbf{p}_{0,y} + t \mathbf{v}_y = -(\mathbf{p}_{0,x} + t \mathbf{v}_x - c)^2 + b + (c - a)^2$$

Az egyenlet t -re másodfokú, amennyiben nincs valós gyök, a sugárkövetést leállítjuk metszéspont nélkül, egyébként a nem feltétlenül különböző gyököket t_1 és t_2 -vel jelölve $t_{i+1} = \min\{t_1, t_2\}$. A kisebb gyök választásának a helyességére elegendő feltétel, hogy $\mathbf{v}_y < 0$ és $\mathbf{p}(t_i)_y > -c^2 + b + (c - a)^2$ definíció szerint teljesül. Utóbbi egyenlőtlenség akkor garantált, ha $\beta > 1$ is igaz, azonban ellenkező esetben a másodfokú egyenlet megoldása szükségtelenné válik. A sugármetszést az Algoritmus 5 formalizálja, példa a 3.9. ábrán látható.

5. Algoritmus Sugárkövetés forgásfelületek metszésével

Function Intersect($\mathbf{p}_0, \mathbf{v}, a_{ij}, b_{ij}, c_{ij}, h_{ij}$) ▷ Bemenet: sugár, forgásfelületek paraméterei

```

1:  $\mathbf{p} \leftarrow \mathbf{p}_0; s \leftarrow 0$ 
2: while  $s < steps \wedge t > \varepsilon \wedge \mathbf{p}$  a magasságmező felett do
3:    $(i, j) \leftarrow \mathbf{p}$  alatti texel koordinátái
4:    $(x, y) \leftarrow$  sugár és  $(a_{ij}, b_{ij})$  pontba húzott egyenes metszéspontja
5:    $\alpha \leftarrow \frac{x}{a}$ 
6:   if  $\alpha \leq 1$  then ▷ A szakaszt metszette a sugár
7:      $t \leftarrow \|\mathbf{p} - (x, y)\|$ 
8:   else ▷ A parabolával való metszést vizsgáljuk
9:      $A \leftarrow \mathbf{v}_x^2$ 
10:     $B \leftarrow \mathbf{v}_y - 2 \cdot c_{ij} \cdot \mathbf{v}_x$ 
11:     $C \leftarrow \mathbf{p}_y - h_{ij} - b_{ij} - a_{ij}^2 + 2 \cdot a_{ij} \cdot c_{ij}$ 
12:     $t \leftarrow solveQuadratic(A, B, C)$ 
13:   end if
14:    $\mathbf{p} \leftarrow \mathbf{p} + t \cdot \mathbf{v}$  ▷ Előrehaladunk a sugár mentén
15:    $s \leftarrow s + 1$  ▷ Megtett lépések számlálása
16: end while
17: return  $t$ 
```

A sugár mentén való iterációt három feltétellel szakíthatjuk meg. Amennyiben a $\mathbf{p}(t_i)_y$ értéke egy előre meghatározott kicsi környezetén belül kerül a pont alatti texel magasságának, akkor metszéspontba juttottunk és a visszaadjuk a kapott koordinátákat. Abban az esetben, ha kilépünk a magasságmező feletti térből az előző feltétel miatti leállás nélkül, akkor metszéspont hiányával terminálunk. Ha az előző két módon nem állunk le egy előre

megadott iterációs szám eléréséig, metszéspont nélkül leállítjuk az algoritmust, elkerülve a lassan konvergáló sugarakból adódó renderelési teljesítmény csökkenését.

3.4.1. Másodfokú egyenlet numerikus stabilizálása

A megjelenítés során a sebességhez hasonlóan kiemelten fontos a numerikus számítások hibájának minimalizálása. Másodrendű függvények zérushelyeinek keresése esetén általánosan nehéz feladat időhatékonyan elérni ezt, azonban a sugárkövetés során számos alkalommal szükség van a 3.3.2.-hez hasonló egyenletek gyökeinek kiszámítására.

Blinn[17] bemutatott egy homogén koordinátákat használó, numerikusan stabil másodfokú egyenletmegoldó módszert, amely azonban több feltétel-kiértékelést igényel a számítás során. Valós idejű rendereléskor a feltételes elágazások használata futási idő szempontjából költséges művelet, ezért a saját követelmények speciális tulajdonságait kihasználva egyszerűsítünk a módszeren.

A 5. Algoritmus jelöléseit használva, a megoldandó egyenlet együtthatóira $A \geq 0$ és $B < 0$ garantáltan teljesül. Az A -ra vonatkozó feltétel triviális, míg B -hez elegendő, hogy $\mathbf{v}_y < 0$ és $\mathbf{v}_x, c_{ij} > 0$, amelyek mindegyike definícióból adódóan igaz. Ezekkel a megkötésekkel, valamint kihasználva, hogy kisebb megoldását keressük az egyenletnek, egyedül azt a feltételt kell megvizsgálni, hogy a kapott érték valós-e. Az egyszerűsített megoldó eljárást az Algoritmus 6 mutatja.

6. Algoritmus Numerikusan stabil másodfokú egyenletmegoldás

Function solveQuadratic(A, B, C) $\triangleright$ A paraméterekre teljesül, hogy $A \geq 0, B < 0, C \in \mathbb{R}$

$$B \leftarrow \frac{B}{2}$$

$$M_1 = C$$

$$M_2 = -B + \sqrt{B^2 - AC}$$

$$x = \frac{M_1}{M_2}$$

if $x \in \mathbb{R}$ **then**

return x

$\triangleright$ A kisebb valós gyöke az $Ax^2 + Bx + C = 0$ egyenletnek

else

return ∞

$\triangleright$ Nincs valós gyök, a következő lépés nagysága végtelen

end if

Ezzel minimalizáltuk az esetlegesen előforduló numerikus pontatlanság okozta átlépett felületmetszések valószínűségét, amelyek általában a felület élesebb részein látható hibaként jelentkeznek.

4. fejezet

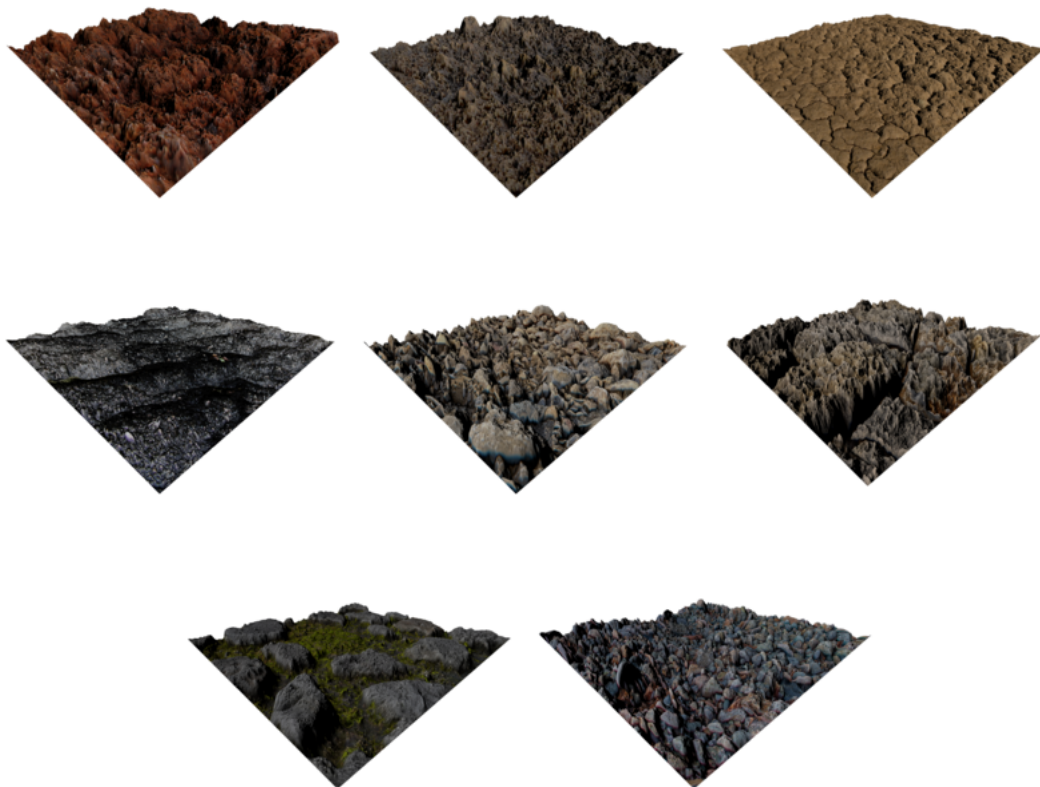
Eredmények és összefoglalás

Ebben a fejezetben a dolgozatban bemutatott algoritmusok implementációjáról, azoknak több szempontból történő összehasonlításáról más módszerekkel, valamint további kutatási irányokról esik szó.

4.1. Implementáció és tesztelés

A korábban kifejtett per-pixel displacement mapping technikák, név szerint lineáris keresés, cone step mapping, relaxed cone step mapping, valamint a saját másodrendű módszer, mindegyike egy közös tesztprogramban implementálásra került. A program az NVIDIA Falcor[18] keretrendszerére épül, amelyben C++ valamint HLSL programnyelveket használtunk. A renderelés pixel shaderekkel történik, míg a gyorsító struktúrák generálása compute shaderekkel megy végbe. A saját algoritmus előfeldolgozásához készült különálló implementáció is általános célú GPU programozásra alkalmas CUDA API felhasználásával, ugyanis a nagyobb felbontáshoz szükséges generálási idő eseténként meghaladta a kernel által engedélyezett szűk időkeretet.

A működés teszteléséhez mind procedurálisan generált, mind fájlból betöltött magasságtérképek felhasználásra kerültek (4.1. ábra). A másodrendű felületeket tároló textúrák 128×128 -tól 2048×2048 -as felbontásig lettek sugárkövetéshez tesztelve. A procedurális textúrákat, amelyek egyszerű kétváltozós függvényekből kerültek generálásra kizárólag az implementáció helyességének ellenőrzésére használtam fel, ugyanis ezek megjelenítése speciális tulajdonságokkal rendelkezik, ezáltal valós alkalmazásokban általában hatékonyabban renderelhetők.



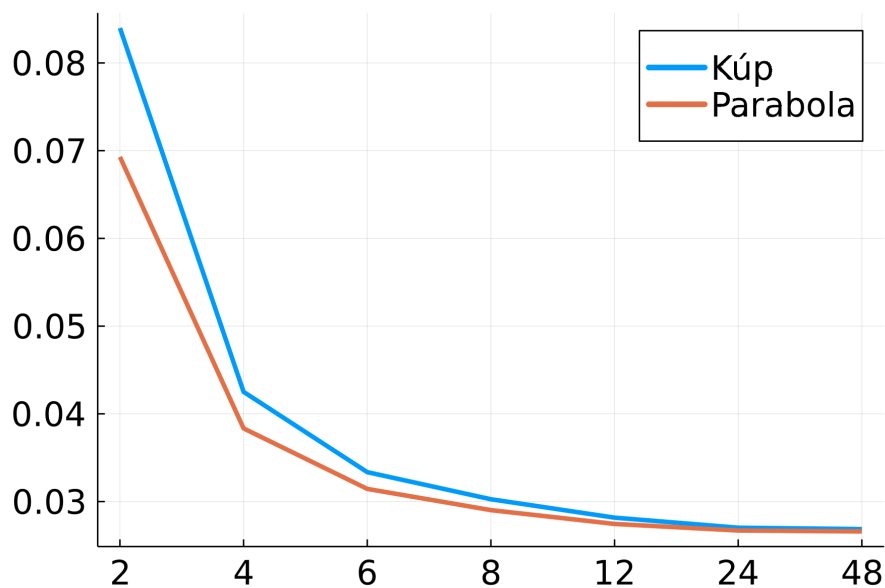
4.1. ábra. Teszteléshez használt magasságmezők textúrázva.

A következőekben bemutatott mérések adatai 1920×1080 pixel felbontás mellett egy NVIDIA GeForce GTX 1060M GPU-n történő futtatásokból származnak. A tesztekhez használt másodrendű felülettérképek mindegyikét az $\alpha = 0.2$ hiperparaméterrel generáltuk, amely előzetes kísérletek alapján általánosan jó választásnak bizonyult.

4.2. Eredmények

A bevezetett algoritmus célja a magasságmezők megjelenítéskor történő sugárkövetés során megtett lépések számának csökkentése, ezáltal gyorsítva egy képkocka előállítását. Ezt az összefüggést a lépésszám és a sebesség között a GPU-n történő memórialvasásra való relatíven magas várakozási idő indokolja. Ebből kifolyólag kevesebb iteráció megtétele egy-egy sugár mentén javíthat a renderelés gyorsaságán, amennyiben az aritmetikai számításokra fordítandó idő nem emelkedik nagy mértékben.

A teljesítményt sugarak konvergenciájának sebessége, valamint egy kép átlagos megjelenítési ideje alapján vizsgáljuk, mindkét szempont alapján közvetlenül a cone step mapping implementációjával hasonlítjuk össze. Ezt a döntést az indokolja, hogy a lineáris ke-



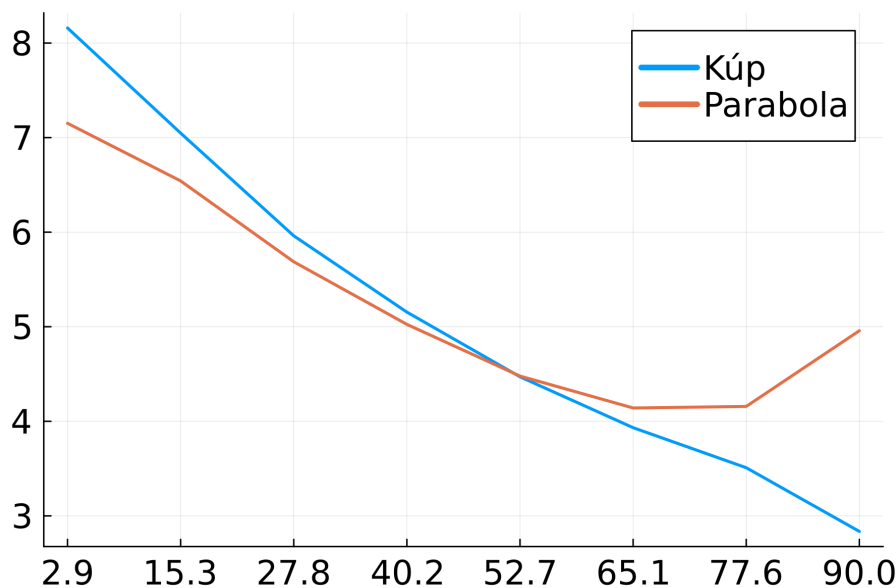
4.2. ábra. Átlagos hiba maximális iterációs szám alapján. Az átlagos eltérés (függőleges tengely) a pontos értéktől mindkét módszer esetén nagymértékben csökken a lépésszám növelésével (vízszintes tengely). 48 iteráció után a hiba elhanyagolhatóan kicsivé válik.

resés egy naiv, gyorsító adatszerkezetet nem használó módszer, így teljesítményben jelentős mértékben elmarad egyéb algoritmusoktól. A relaxed cone step mapping bár hasonló struktúrákat használ fel, alapvető különbséget jelent a működésében, hogy megengedi a felületbe való belépését a sugárnak, így számos esetben visszalépéseket követel meg, amelyet a konzervatív geometriákkal elkerülhetünk. Ezen felül a relaxált kúpok azon tulajdonsága, hogy legfeljebb egyszeri átlépést garantál a felület határán, kizárólag olyan sugarakra teljesül, amelyek a magasságmező tetejéből indulnak ki, tetszőleges irány esetén helytelen eredményt kaphatunk.

4.2.1. Lépésszám vizsgálata

A sugarak mentén való előrehaladás sebességének összehasonlításához megvizsgáltuk megadott maximális iterációs szám eléréséig megtett távolságot, valamint a konvergenciához szükséges lépések számát. A felületeket 8 különböző pozícióból rendereltük egy körív mentén egyenletesen csökkenő beesési szöggel rendelkező, azonos pont felé irányított virtuális kamerák elhelyezésével. A szögek minden esetben 3° és 90° között változnak.

A maximális iterációs szám eléréséig megtett lépések nagyságát a sugár mentén a metszésponttól vett távolságként hasonlítjuk össze, ezzel abszolút különbségeket tudunk mérni a két algoritmus között. Az alapul szolgáló konvergált sugarak pozícióját 200 lépéses lineáris keresséssel, majd a kapott értékből a pontos metszéspont 20 lépéses bináris köze-



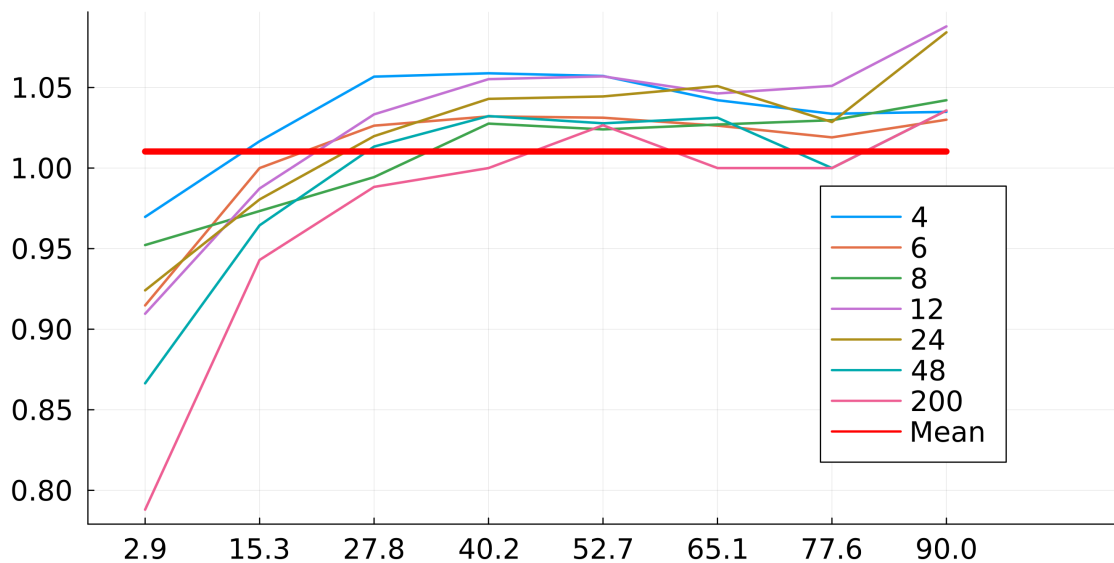
4.3. ábra. Átlagosan megtett lépések száma a konvergenciához (függőleges tengely) a betekintési szög (vízszintes tengely) függvényében. Általános tendencia, hogy nagyobb szög esetén kevesebb lépés szükséges a metszéspont eléréséhez. A saját módszer kis szögekre jobb eredményt ad, azonban 45° felett lassabban áll le a cone step mapping-hez viszonyítva.

lítésével határoztuk meg. Ez az eljárás stabil, kellően pontos eredményt ad, ehhez viszonyított hibát hasonlítottunk össze. Átlagosan a másodrendű felületek használata kisebb hibát jelent a több betekintési szögből mért felületek megjelenítésekor (4.2. ábra).

A hiba mértékének gyorsabb csökkenése következtetni enged arra, hogy az átlagosan metszéspontba jutáshoz szükséges lépések száma is csökken. Ehhez megvizsgáltuk a korábban használt kamerapozíciókból az összes felületet eltaláló sugár esetén az átlagos értéket (4.3. ábra).

Azt az eredményt kapjuk, hogy 45° -ig a saját algoritmus ad jobb eredményt, alacsony szögeknél akár átlagosan 0.5-1 lépéssel hamarabb áll le. Magasabbról indított sugarak esetén azonban ez az arány megfordul, valamint a várakozással ellentétben emelkedni kezd a lépésszám. A jelenség okát a két algoritmus közötti implementációs különbségek okozhatják, ugyanis a lassúnak bizonyuló sugarak többsége egyedül a kúpokat metszik el, amelyeket legalább akkora nyílásszöggel generáltunk, mint a cone step mapping esetében.

Összességében az iterációk száma a vártnak megfelelően változott, a felület közelében azonban még megvizsgálandó, hogy mi okozhat látható mértékű lassulást.



4.4. ábra. Renderelési sebesség összehasonlítása a két módszer között. A vízszintes tengelyen a sugarak beesési szögeit jelöltük, a görbék maximális iterációs szám alapján elkülönítve az általános forgásfelületek átlagos idejét mutatják a cone step mapping arányában. Az 1-nél kisebb érték gyorsabb, az afeletti lassabb renderelést jelent. A vízszintes piros vonal az arányok átlagát jelöli, nagyjából 1% sebességvesztést mutatva. Az ábra alapján az algoritmus alacsony szögekre jól látható gyorsulást eredményez.

4.2.2. Renderelési sebesség vizsgálata

Egy valós időben futó renderelő eljárások esetén az egyik kritikus szempont az egyes képkockák megjelenítésére fordított idő. A bemutatott módszerek mindegyike a mezőfelület pontos renderelésére törekszik a lehető leggyorsabb módon. Az itt bemutatott mérések értelemszerűen nem tartalmazzák az egyszer végrehajtott előfeldolgozást, hanem a tényleges sugárkövetésre vonatkoznak.

A renderelési sebességek összehasonlítására a 4.1. ábrán látható textúrák mindegyikén, a korábban részletezett 8 kameraállásból, 7 különböző maximális iterációs számra vonatkozó méréseket végeztünk a két algoritmussal. Az egyes magasságtérképek közötti különbségek nem relevánsak, ugyanis előfordulhatnak olyan esetek, ahol az egyik algoritmus jellemzőiből adódóan jobban vagy gyengébben teljesít, azonban ezek nem reprezentatív példák, így a továbbiakban az összes textúrára kapott értékek átlagát vesszük.

A bevezetett módszer egyetlen iterációban általában két felülettel való metszéspont kiszámítását végzi, melyek közül egyik másodrendű, ezen lépések aritmetikailag összetettebbek, mint a cone step mapping esetében. Azonos számú lépés megtétele ebből kifelől lassabb renderelést eredményez, azonban a fejezet előző szakaszában bemutatott mérések alapján helyenként a gyorsabb konvergenciával ellensúlyozva javulhat a haté-

konyság.

Az egy kép átlagos kirajzolásának idejére vonatkozó eredményeket ezredmásodperc-ben adjuk meg, amelyek a képernyő maximális frissítési frekvenciájától függetlenül mér-tünk. A 4.4. ábrán látható a két módszer átlagos futási idejének aránya, a cone step mapping-hez viszonyítva. Jelentősebb különbséget az alacsony betekintési szögek esetén figyelhetünk meg, ahol 5-20% közötti javulást eredményez a módszerünk, míg magasabb-ról érkező sugaraknál átlagosan kisebb mértékű lassulás tapasztalható.

Külön méréseket végeztünk 4, 6, 8, 12, 24, 48 valamint 200 maximális iterációval, amelyeknél megfigyelhető, hogy több lépéssel nagyobb mértékben javul a futási idő. Ez a jelenség igazolja az egyes sugár és forgásfelület közötti metszések kiszámításának bo-nyolultságát a kúpok használatához képest. Amint a sugarak számottevően nagyobb része hamarabb konvergál másodrendű felületekkel, azaz 12-24 lépés felett, a sebességben is nagyobb mértékű javulást eredményez. Magasabb szögek esetén a 4.3. ábrán is látható iterációszám-növekedés miatt az átlagos renderelési ideje az algoritmusnak emelkedik.

4.1. táblázat. Átlagos különbség a kúpokkal és a másodrendű felületekkel történő renderelések között ezredmásodpercben. Negatív értékek gyorsabb renderelést jelentenek. A 45°-nál kisebb szögekre az összeg -0.667 , míg az összesre -0.32 , ez egy képkocka átlagos kirajzolásának gyorsulását jelenti. Egy képkocka kirajzolása a mérések során 0.3 és 1.2 ezredmásodperc között változott.

Szög	Cone step és másodrendű különbség (ms)						
	4 iter.	6 iter.	8 iter.	12 iter.	24 iter.	48 iter.	200 iter.
2.9	-0.023	-0.100	-0.047	-0.107	-0.093	-0.123	-0.260
15.3	0.010	0.000	-0.020	-0.010	-0.017	-0.023	-0.043
27.8	0.027	0.017	-0.003	0.020	0.013	0.007	-0.007
40.2	0.023	0.017	0.013	0.027	0.023	0.013	0.000
52.7	0.020	0.013	0.010	0.023	0.020	0.010	0.010
65.1	0.013	0.010	0.010	0.017	0.020	0.010	0.000
77.6	0.010	0.007	0.010	0.017	0.010	0.000	0.000
90.0	0.010	0.010	0.013	0.027	0.027	0.010	0.010
Összesen:	0.090	-0.027	-0.013	0.013	0.003	-0.097	-0.290

Az összes mérés alapján átlagosan 1% a lassulás mértéke a cone step mapping-hez képest, legrosszabb esetben 5-7% a veszteség, míg akár 20%-os gyorsulást is elérhetünk alacsony szögek esetén. Azonban az átlagok különbségét tekintve a másodrendű mód-szer bizonyult hatékonyabbnak, ugyanis ahogy azt a 4.1. Táblázat mutatja, a renderidők különbségének összegei általában negatívak. Ezt értelmezhetjük úgy, hogy ha egyenletes sebességgel körbejárjuk a felületet a tesztelt irányokban, akkor összességében több kép-kockát renderelünk le, mintha az eredeti cone step mapping módszert használtuk volna.

4.3. Összefoglalás

A dolgozatban bevezettünk egy per-pixel displacement mapping módszert, amely a cone step mapping néven ismert gyorsított sugárkövetést általánosítja kúpok helyett másodrendű forgásfelületek alkalmazásával. A magasságtérképből olvasott adatok alapján egy gyorsító adatszerkezetet generálunk, amelyet egy négycsatornás textúrában tárolva, valós idejű sugárkövetés hatékonyságának javítására használunk a generált felületek sugárral való metszésével.

Az implementációt összevetettük a kiinduló algoritmussal, számos felület megjelenítését vizsgáltuk több szempontból. Az eredmények megoszlóak, alacsonyabb szögből a felületre érkező sugaraknál számottevő javulást tapasztalunk a renderelés sebességében, magasabb szögeknél azonban kisebb mértékben, viszont romlott a teljesítmény. Összesítve az eltéréseket a saját módszer átlagos esetben gyorsabbnak bizonyult, ezzel ösztönző hatást gyakorolva további optimalizációs irányok kivizsgálására.

A bemutatott algoritmusokról és az eredményekről publikáció készült az Acta Cybernetica folyóiratba, mely a megírásakor elfogadott státuszban van.

4.3.1. További kutatási irányok

Az bemutatott eredmények alapján az algoritmus a modern módszerekkel összehasonlítható hatékonysággal jeleníti meg a transzformált felületeket, emellett több szempontból további optimalizálásra ad lehetőséget. Ezek között szerepel a geometriára közel merőlegesen érkező sugarak átlagosnál lassabb konvergencia-sebessége, amely a cone step mapping eljárással való hasonlóságok alapján várhatóan számottevő mértékben javítható.

A renderelésnél kevesebb figyelmet fordítottunk az előfeldolgozás idejének minimalizálására, elsősorban a generált felületek geometriájára fektettük a hangsúlyt. Ebből kifolyólag ez a lépés meglehetősen lassú, és elegendő egyszer végrehajtani minden textúrához, helyenként problémába ütközik a futtatása. Ismert azonban a konzervatív kúpok generálására nagyságrendekkel gyorsabb módszer[11], amelyhez hasonló eljárás a mi struktúránkon is alkalmazható. Ennek során mipmap-ekben tárolva a magasságtérkép maximumait, elkerülve, hogy a texteltől távol elhelyezkedő figyelembe vett magasságértékek számát.

Egy további lehetőség a relaxált felületek vizsgálata. Jelenleg a generált felületek nem metszik a magasságmezőt, azonban a relaxed cone step mapping módszer ezt megengedő konstrukciója további gyorsulását eredményezi a sugárkövetésnek. Feltételezhető, hogy

valamilyen hasonló tulajdonsággal rendelkező másodrendű struktúra definiálásával a hatékonyság javítható, azonban ennek részletesebb vizsgálata a jövőbeni kutatás tárgya.

Köszönetnyilvánítás

A Kulturális és Innovációs Minisztérium ÚNKP-22-2 kódszámú Új Nemzeti Kiválóság Programjának a Nemzeti Kutatási, Fejlesztési és Innovációs Alapból finanszírozott szakmai támogatásával készült.



Irodalomjegyzék

- [1] László Szirmay-Kalos és Tamás Umenhoffer. „Displacement Mapping on the GPU - State of the Art”. *Comput. Graph. Forum* 27.6 (2008), 1567–1592. old. DOI: 10.1111/j.1467-8659.2007.01108.x. URL: <https://doi.org/10.1111/j.1467-8659.2007.01108.x>.
- [2] Robert L Cook és Kenneth E. Torrance. „A reflectance model for computer graphics”. *ACM Transactions on Graphics (ToG)* 1.1 (1982), 7–24. old.
- [3] Xiao D He és tsai. „A comprehensive physical model for light reflection”. *ACM SIGGRAPH computer graphics* 25.4 (1991), 175–186. old.
- [4] James F. Blinn. „Simulation of Wrinkled Surfaces”. *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '78*. New York, NY, USA: Association for Computing Machinery, 1978, 286–292. old. ISBN: 9781450379083. DOI: 10.1145/800248.507101.
- [5] Jonathan Cohen, Marc Olano és Dinesh Manocha. „Appearance-preserving simplification”. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 1998, 115–122. old.
- [6] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces*. The University of Utah, 1974.
- [7] Tomomichi Kaneko és tsai. „Detailed shape representation with parallax mapping”. *In Proceedings of the ICAT 2001* (2001. jan.).
- [8] Bui Tuong Phong. „Illumination for computer generated pictures”. *Communications of the ACM* 18.6 (1975), 311–317. old.
- [9] Jonathan Dummer. „Cone step mapping: An iterative ray-heightfield intersection algorithm (2006)”. URL: <http://www.lonesock.net/files/ConeStepMapping.pdf> 2.3 (2006), 4. old.

- [10] Akram Halli és tsai. „Per-pixel displacement mapping using cone tracing”. *International Review on Computers and Software* 3.3 (2008), 1–11. old.
- [11] Gábor Valasek és Róbert Bán. „Quick cone map generation on the GPU”. (2022).
- [12] Fabio Policarpo és Manuel M Oliveira. „Relaxed cone stepping for relief mapping”. *GPU gems* 3 (2007), 409–428. old.
- [13] Lionel Baboud, Elmar Eisemann és Hans-Peter Seidel. „Precomputed safety shapes for efficient and accurate height-field rendering”. *IEEE transactions on visualization and computer graphics* 18.11 (2011), 1811–1823. old.
- [14] Csaba Bálint és Mátyás Kiglics. „A Geometric Method for Accelerated Sphere Tracing of Implicit Surfaces”. *Acta Cybernetica* (2021). DOI: 10 . 14232 / actacyb . 290007. URL: <https://cyber.bibl.u-szeged.hu/index.php/actcybern/article/view/4203>.
- [15] Mátyás Kiglics, Valasek Gábor és Csaba Bálint. „Másodfokú felülettérkép parallax leképezés gyorsítására”. *X. Magyar Számítógépes Grafika és Geometria Konferencia*. Szerk. L Szirmay-Kalos és G Renner. 2022, 85–88. old.
- [16] Alex M Andrew. „Another efficient algorithm for convex hulls in two dimensions”. *Information Processing Letters* 9.5 (1979), 216–219. old.
- [17] H Blinn. „How to solve a quadratic equation?”: *IEEE computer Graphics and Applications* 25.6 (2005), 76–79. old.
- [18] Simon Kallweit és tsai. *The Falcor Rendering Framework*. 2022. márc. URL: <https://github.com/NVIDIAGameWorks/Falcor>.

Ábrák jegyzéke

1.1. Egyszerű geometria és bump map alkalmazása	4
1.2. Egyszerű geometria és felbontás utáni transzformáltja	5
1.3. Geometria és ahhoz rendelt magasságtérképpel transzformáltja	5
1.4. Egyszerű geometria és parallax mapping alkalmazása	6
2.1. A sugárkövetés probléma displacement mapping esetén	8
2.2. Lineáris keresés	10
2.3. Hibásan renderelt felület	11
2.4. Példák konzervatív kúpra	12
2.5. Egy konzervatív kúp generálásának menete	13
2.6. Relaxált kúpokkal való sugármetszés hibája	13
2.7. Példa relaxált kúpra	14
3.1. Konzervatív forgáspárolák	16
3.2. Helytelen parabola generálás	17
3.3. Felület közelében leszűkülő parabola	18
3.4. Kúpra illesztett parabola síkban	19
3.5. Kúpokhoz illesztett forgáspárolák	20
3.6. A parabola c paraméterének meghatározása.	21
3.7. Magasságtérkép transzformációja lineáris függvényre	22
3.8. Különbség generált görbék között	24
3.9. Sugárkövetés forgáspárolákkal	25
4.1. Teszteléshez használt textúrák	29
4.2. Átlagos hiba iterációszám alapján	30
4.3. Átlagosan megtett lépések leállásig	31
4.4. Renderelési sebesség összehasonlítása	32

Algoritmusjegyzék

1.	Lineáris keresés	9
2.	Cone stepping	11
3.	Radiális függvény előállítása	20
4.	A forgásfelület paramétereinek előállítása	22
5.	Sugárkövetés forgásfelületek metszésével	26
6.	Numerikusan stabil másodfokú egyenletmegoldás	27