Eötvös Loránd Tudományegyetem

Informatikai Kar

Algoritmusok és Alkalmazásaik Tanszék

Valós idejű egysugaras puhaárnyék-számító algoritmusok távolságfüggvényekkel definiált felületekhez

Bálint Csaba	Dr. Valasek Gábor	Bán Róbert
Doktorandusz	Egyetemi adjunktus	Prog. inf. MSc

Budapest, 2019

Kivonat

A háromdimenziós színterek valószerű megjelenítésének egyik fontos eleme az árnyékok alkalmazása. Segítségükkel a színteret alkotó geometriák önmagukban vett és egymáshoz viszonyított tulajdonságai is egyértelműsödnek.

A dolgozat célja olyan területtel rendelkező fényforrások által vetett valószerű, puha árnyékok kiszámítását végző algoritmusok konstruálása, amelyek távolságfüggényekkel definiált felületekkel működnek, a reprezentáció speciális tulajdonságait felhasználva a minél hatékonyabb képszintézis megvalósítására.

Számításigény tekintetében a legnagyobb kihívást a térfogattal rendelkező fényforrások jelentik, melyekhez valósidejű megoldást mutatunk be. Fontos eredmény továbbá, hogy a dolgozatban bemutatott puha árnyékot számító adaptív algoritmus az approximált integrál maximális hibájával paraméterezhető.

Tartalomjegyzék

1. fejezet

Bevezetés

Dolgozatom távolságfüggvényekkel definiált felületek által vetett árnyékok számítását vizsgálja.

A távolságfüggvényekkel leírt felületek megjelenítésére Hart sphere trace algoritmusa (Hart [1994]) egyszerre biztosít hatékony és robusztus lehetőséget. Dolgozatom alapötlete azon megfigyelés, hogy ez a reprezentáció nem csak az egyetlen sugárral történő metszést teszi könnyen elvégezhetővé, hanem egy egy pontból kiinduló sugársereg által alkotott kúp és a felületek metszetvizsgálata is egyszerűen megvalósítható. Ezáltal egyszerre több sugár viselkedése is közelíthető.

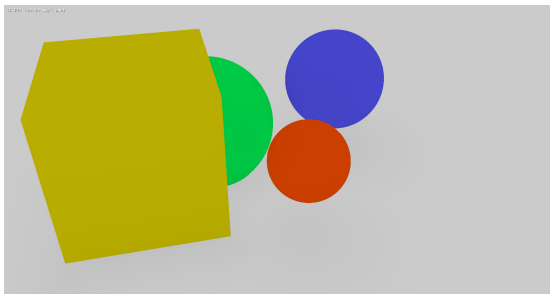
Mivel az implicit függvényekkel definiált felületek sugárkövetése a szokásosnál költségesebb és ez arányos az indított sugarak számával, ezért különösen fontos a grafikai effektusok, például árnyéksugarak hatékony implementációja. Dolgozatomban gömb fényforrás által vetett puha árnyék számítására mutatok olyan algoritmust, amely egyetlen sugár végigkövetésén alapszik a felületi pont és a fényforrás közötti kúp tengelyében.

Legfontosabb eredményünk, hogy az irodalomban eddig ismertetett módszerekkel (Andersson and Barré-Briseboi [2018]; Edward Liu [2018]) szemben algoritmusunk egyetlen kúp végigkövetésével, azaz lényegében egyetlen sugárral jelenít meg valósághű puha árnyékokat. Ez jelentős sebességnövekedéssel jár, azonban az általunk számított árnyékok eltérnek a pontos árnyékoktól, a módszer során alkalmazott felületközelítések miatt. A 6. fejezetben ezen pontatlanságokra is kitérünk. Tesztjeinkben ezek a hibák nem bizonyultak szembetűnőnek. Más egysugaras módszerekhez (Íñigo Quílez [2010]; Aaltonen [2018]) képest az algoritmus sokkal pontosabb eredményt ad, hasonló sebesség mellett.

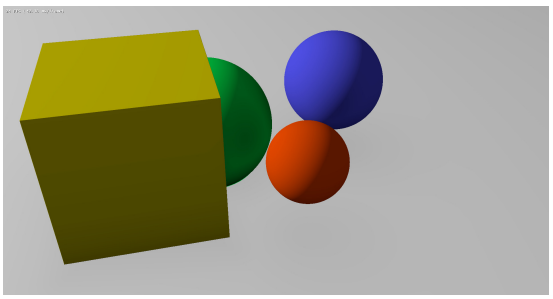
Bemutatunk továbbá egy olyan módszert, amely a fizikai alapú megjelenítés során gyorsítja fel az illumináció számítását. Ennek alapja szintén az egysugaras algoritmus, ami lehetővé teszi, hogy a színtér geometriájától függetlenül közelítsük a számítandó integrált. Ez alkalmazható tetszőleges alakú, gömbbel befoglalható fényforrások által kibocsátott fény számolására is, melyet téglalap alakú fényforrásokkal

szemléltetünk.

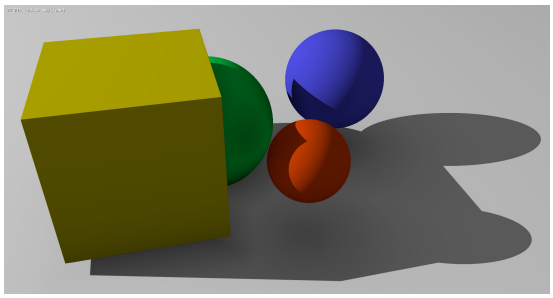
Az algoritmusokat egy kisebb projekt keretében fejlesztett távolságfüggvényekkel definiált felületeket megjelenítő motorban implementáltuk. Az ábrákon látható képek többsége valós időben jelenik meg, ugyanis az implementáció masszívan párhuzamos környezetben valósult meg az NVIDIA Falcor keretrendszerében (Benty *et al.* [2017]), DirectX 12 backend-del.



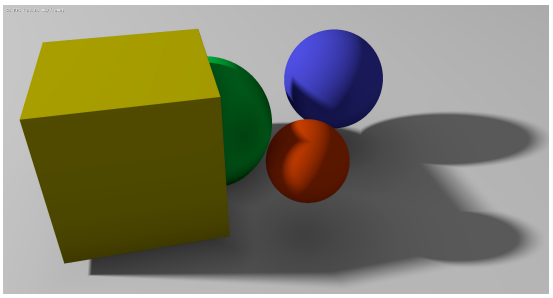
(a) *Árnyalás nélkül*



(b) *Diffúz árnyalás*



(c) *Kemény árnyék*



(d) *Puha árnyék*

1.1. ábra. *Vetett árnyék nélkül az objektumok egymáshoz viszonyított helyzete kétséges. A vetett árnyék egyértelműsíti az objektumok méretét és egymáshoz való viszonyát.*

2. fejezet

Elméleti bevezető

Ebben a fejezetben definiáljuk a feladathoz szükséges matematikát, valamint a felhasznált alapvető algoritmusokat. Tárgyaljuk az implicit függvények – és azok speciális változatának, a távolságfüggvények – által meghatározott felületeket és ezek megjelenítésének problémáját. Értelmezzük továbbá a távolságfüggényeken végezhető műveleteket.

2.1. Implicit függvény által definiált felület

A dolgozatban $f : \mathbb{R}^3 \to \mathbb{R}$ függvények 0-hoz tartozó szintfelülete (nívóhalmaza)

$$\{f = 0\} \coloneqq \left\{\boldsymbol{x} \in \mathbb{R}^3 : f(\boldsymbol{x}) = 0\right\} \subset \mathbb{R}^3$$

definiálja a megjelenítendő felület pontjait. A képernyőn való megjelenítéshez egy $I \in \mathbb{R}^2 \to \mathbb{R}^3$ képfüggvény[1] értékét kell meghatároznunk. Az I függvény a képernyő koordinátáihoz rendeli az ott megjelenítendő színt.

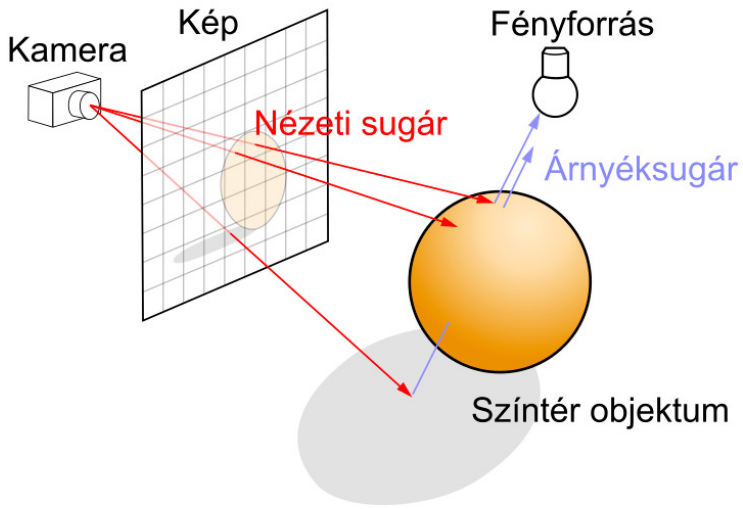
2.2. Sugárkövetés

Az I képfüggvény számolásához a háromdimenziós térbe egy virtuális kamerát helyezünk, és a kamerából sugarakat (félegyeneseket) indítunk. A sugarak a kamera előtt elhelyezett virtuális képernyő megfelelő részén haladnak keresztül. A virtuális kamera sematikus ábrája a 2.1. ábrán[2] látható. A sugarak meghatározásához definiáljuk a kamerafüggvényt: $C \in \mathbb{R}^2 \to \mathbb{R}^3 \times \mathbb{R}^3$. A kamerafüggvény a képernyő (x, y)

[1] Az értelmezési tartomány írja le a teljes képernyőt, például $[0, 1]^2$ vagy $[-1, 1]^2$. A visszatérési értéket most normalizált RGB-színháramsként értjük ($[0, 1]^3$). Az I definiálható lenne különböző színhullámhosszakra, és ekkor az adott hullámhosszhoz tartozó fénymennyiséget határoznánk meg: $I \in \mathbb{R}^2 \times \mathbb{R}^+ \to \mathbb{R}^+$

[2] Forrás: Wikipédia

pontjához visszaadja a rajta átmenő sugarat: $(\boldsymbol{p}, \boldsymbol{v})$ pár, ahol $\boldsymbol{p}$ a sugár kezdőpontja és $\boldsymbol{v}$ a sugár irányvektora. Az irányvektort normalizálva értjük, azaz $\|\boldsymbol{v}\|_2 = 1$.



2.1. ábra. *A virtuális kamera sugárkövetéses képszintézisnél*

A $\boldsymbol{p}$ kezdőpontú $\boldsymbol{v}$ irányvektorú sugár parametrikus függvényére vezessük be az $\boldsymbol{s}_{\boldsymbol{p},\boldsymbol{v}} : \mathbb{R}^+ \to \mathbb{R}^3$ jelölést. Ekkor a képernyő (x, y) koordinátához tartozó sugár az $\boldsymbol{s}_{C(x,y)}$. A sugár a következőképpen számolható:

$$\boldsymbol{s}(t) \coloneqq \boldsymbol{s}_{\boldsymbol{p},\boldsymbol{v}}(t) \coloneqq \boldsymbol{p} + t \cdot \boldsymbol{v}.$$

Ezen a sugáron keressük a felülettel való első – kamerához legközelebbi – metszéspontot, azaz a legkisebb olyan t paramétert, amire $f(\boldsymbol{s}(t)) = 0$:

$$T(\boldsymbol{s}) \coloneqq T_f(\boldsymbol{s}) \coloneqq \inf\left\{t \in \mathbb{R}^+ : f(\boldsymbol{s}(t)) = 0\right\},^3 \quad \left(T : \left(\mathbb{R}^+ \to \mathbb{R}^3\right) \to \mathbb{R}^+\right).$$

A felületi pont térbeli helyét megkapjuk, ha visszahelyettesítjük a kapott t értéket a paramterikus egyenletbe: $\boldsymbol{x} = \boldsymbol{s}(t)$.

A megtalált felületi ponthoz ezután egy színt kell rendelnünk, amit megjeleníthetünk a képernyőn. Ez a szín a jelenet geometriáján – azaz az f függvényen – kívül függ a nézeti iránytól is:

$$S \coloneqq S_{x,y} \coloneqq S_{\boldsymbol{v}_{x,y}} : \mathbb{R}^3 \to \mathbb{R}^3.$$

Az S függvény a felületi pont megvilágítottságát számolja ki a megadott nézeti irányból tekintve. A most definiált függvényekkel az $I \in \mathbb{R}^2 \to \mathbb{R}^3$ képfüggvény a következőképpen adódik:

$$I(x, y) \coloneqq S_{x,y}\left(\boldsymbol{s}_{C(x,y)}\left(T_f\left(\boldsymbol{s}_{C(x,y)}\right)\right)\right).$$

[3] Legyen $\inf \emptyset \coloneqq +\infty$.

Ezen függvények közül a dolgozat az S függvény számolásának egy részével foglalkozik.

2.3. Raymarch

Implicit függvény által definiált felület megjelenítésére alkalmazható a raymarch algoritmus. Feladata az $f \circ \boldsymbol{s} : \mathbb{R}^+ \to \mathbb{R}$ legkisebb paraméterhez tartozó gyökének megkeresése, azaz a korábban definiált $T(\boldsymbol{s})$ meghatározása.

Lényege, hogy egy előre adott $\epsilon > 0$ lépésközzel mintavételezi a sugár mentén az f implicit függvényt. Előjelváltás esetén a gyököt a két minta közt valamilyen numerikus módszerrel pontosítja.

In : $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3, \|\boldsymbol{v}\|_2 = 1$ sugár (kezdőpont és irány),
$\epsilon > 0$ lépésköz,
$f : \mathbb{R}^3 \to \mathbb{R}$ implicit függvény
Out: $t \in \mathbb{R}^+$ a sugáron megtett távolság
begin
  $t \coloneqq 0$
  $f_0 \coloneqq f(\boldsymbol{p})$
  $f_1 \coloneqq f(\boldsymbol{p} + \epsilon \cdot \boldsymbol{v})$
  for $t < t_{max}$ ***and*** $f_0 \cdot f_1 > 0$ **do**
    $t \coloneqq t + \epsilon$
    $f_0 \coloneqq f_1$
    $f_1 \coloneqq f(\boldsymbol{p} + t \cdot \boldsymbol{v})$
  end
  $t \coloneqq \text{Pontosit}\,(f \circ \boldsymbol{s}, [t - \epsilon, t])$
end

Algoritmus 1: *A raymarch algoritmus.*

Az 1. algoritmus működésének feltétele, hogy a gyökök ϵ sugarú környezetében az f függvény folytonos legyen és előjelet váltson. Tehát minél kisebb ϵ, annál valószínűbb, hogy az algoritmus valóban megtalálja a keresett első gyököt, ugyanakkor ezzel együtt nő a számítási idő is. Ugyanígy, ha nagyobb ϵ-t választunk, az algoritmus gyorsabban fut, de nagyobb eséllyel hagy ki gyököt.

Ha f folytonos, az algoritmus akkor találja meg biztosan az objektumot az adott sugáron, ha sugár és az objektum metszete legalább ϵ hosszú. A raymarch algoritmusnál sokkal hatékonyabb módszerek használhatók, ha az f függvényre további megkötéseket teszünk.

2.4. Távolságfüggvények

A távolságfüggvények speciális folytonos implicit függvények, amik megjelenítésére a raymarchnál sokkal gyorsabb algoritmusok léteznek (a sphere-trace algoritmusok). Ráadásul ezek az algoritmusok a távolságfüggvények tulajdonságait kihasználva garantálják, hogy az első gyököt találják meg. (Hart [1994]; Osher and Fedkiw [2003])

2.4.1. Definíció. Legyen (X, d) metrikus tér. Ekkor $x \in X$ és $A \subset X$ *távolsága*:

$$d(x, A) \coloneqq \inf_{a \in A} d(x, a).$$

A dolgozatban az $X = \mathbb{R}^3$ normált térben a $d(x, y) = \|x - y\|_2$ távolsággal számolunk (háromdimenziós tér az euklideszi normával/távolsággal). A példák között előfordul $X = \mathbb{R}^2$ (sík) is.

2.4.2. Definíció. (Hart [1994]) Az $f : \mathbb{R}^n \to \mathbb{R}$ függvény *távolságfüggvény*, ha

$$\forall \boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) = d(\boldsymbol{x}, \{f = 0\}).$$

A megjelenítendő felület az $\{f = 0\}$ halmaz.

2.4.3. Definíció. Az $f : \mathbb{R}^n \to \mathbb{R}$ függvény *előjeles távolságfüggvény*, ha folytonos és $|f|$ távolságfüggvény.

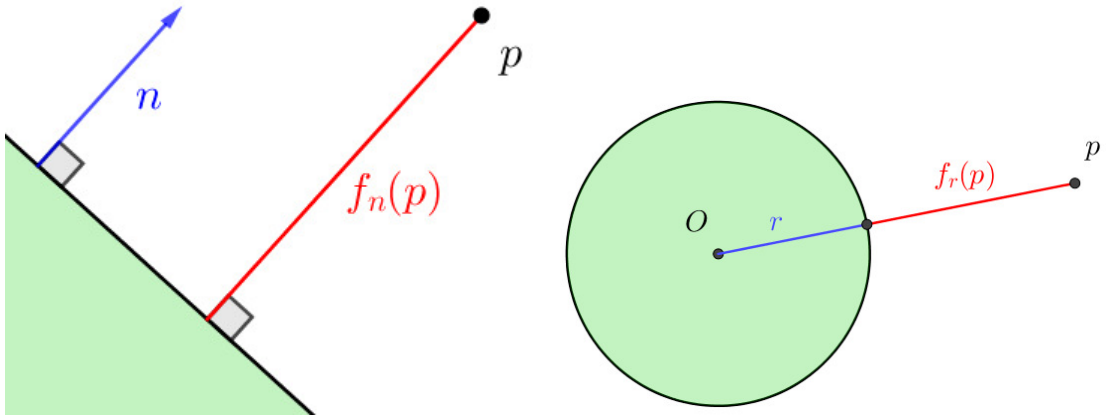
Előjeles távolságfüggvényekkel leírhatók tömör, térfogattal rendelkező objektumok is. A halmaz belseje $\{f < 0\} = \{\boldsymbol{x} \in \mathbb{R}^n \, f(\boldsymbol{x}) < 0\}$, míg a felszíne továbbra is az $\{f = 0\}$ halmaz. Megjegyzendő, hogy a távolságfüggvények is előjeles távolságfüggvények, csak belső rész nélkül.

2.4.4. Definíció. (Bálint and Valasek [2018b]) Az $f : \mathbb{R}^n \to \mathbb{R}$ függvény (előjeles) távolságfüggvény *becslés*, ha létezik $q : \mathbb{R}^n \to [1, K)$ korlátos függvény ($1 \leq K \leq +\infty$) úgy, hogy $f \cdot q$ (előjeles) távolságfüggvény.

A távolságfüggvény becslésekre azért van szükség, mert a legtöbb esetben nem adható (könnyen zárt alakban) távolságfüggvény egy adott halmazra. A távolságfüggvény becslésekre ugyanúgy működnek a távolságfüggvények megjelenítésére használt sphere-trace algoritmusok, mindössze a sebességük csökken (a sebességromlás mértéke a definícióban szereplő K korláttal becsülhető).

2.4.1. Példák

Előjeles távolságfüggvényekre az egyik legegyszerűbb példa a sík. Egy origón áthaladó, $\boldsymbol{n} \in \mathbb{R}^3$ normálissal ($||\boldsymbol{n}||_2 = 1$) rendelkező sík előjeles távolságfüggvénye



(a) *Félsík (egyenes) előjeles távolságfüggvénye*

(b) *Kör előjeles távolságfüggvénye*

2.2. ábra. *Példa előjeles távolságfüggvényekre $\mathbb{R}^2$-ben*

a következő:

$$f_{\boldsymbol{n}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{n} \rangle \,,$$

hiszen a skaláris szorzat éppen az $\boldsymbol{x}$ helyvektorának előjeles vetületét adja az $\boldsymbol{n}$ vektorra. Itt a sík két részre osztja a teret, amiből az egyik az objektum „belseje”. Ha a tér mindkét felét „külső” részként szeretnénk kezelni, vegyük a függvény abszolút értékét. Ugyanez a függvény a síkon az $\boldsymbol{n} \in \mathbb{R}^2$ normálvektorú egyenest (és félsíkot) írja le.

Egy másik egyszerű előjeles távolságfüggvény a gömbé. Az $r > 0$ sugarú origó középpontú gömb előjeles távolságfüggvénye:

$$f_r(\boldsymbol{x}) = \|\boldsymbol{x}\|_2 - r,$$

ugyanis a tér bármely $\boldsymbol{x}$ pontjában a gömb felületének legközelebbi pontja a gömb középpontját és az $\boldsymbol{x}$-et összekötő szakasz és a gömbhéj metszéspontja. Hasonlóan, ha f_r-t a síkon értelmezzük, az origó középpontú r sugarú kör távolságfüggvényét kapjuk.

Az origón áthaladó $\boldsymbol{v} \in \mathbb{R}^3$ irányvektorú ($\|\boldsymbol{v}\|_2 = 1$) egyenes távolságfüggvénye:

$$f_{\boldsymbol{v}}(\boldsymbol{x}) = \|\boldsymbol{x} - \langle \boldsymbol{x}, \boldsymbol{v} \rangle \cdot \boldsymbol{v}\|_2$$

Az origót és tetszőleges $\boldsymbol{0} \neq \boldsymbol{b} \in \mathbb{R}^3$ pontot összekötő szakasz távolságfüggvénye:

$$f_{\boldsymbol{b}}(\boldsymbol{x}) = \left\| \boldsymbol{p} - \operatorname{clamp}\left(\frac{\langle \boldsymbol{b}, \boldsymbol{p} \rangle}{\langle \boldsymbol{b}, \boldsymbol{b} \rangle} \right) \cdot \boldsymbol{b} \right\|_2 , \text{ ahol}$$

$$\text{clamp}(x) = \begin{cases} 0 & \text{ha } x < 0, \\ x & \text{ha } 0 \leq x \leq 1, \\ 1 & \text{ha } 1 < x. \end{cases}$$

2.5. Transzformációk távolságfüggvényekkel

A távolságfüggvények eltolhatóak tetszőleges vektorral úgy, hogy a függvény paraméteréhez a vektor -1-szeresét adjuk. Például a $\boldsymbol{c} \in \mathbb{R}^3$ középpontú gömb előjeles távolságfüggvénye a következő:

$$f_{\boldsymbol{c},r}(\boldsymbol{x}) = f_r(\boldsymbol{x} - \boldsymbol{c}).$$

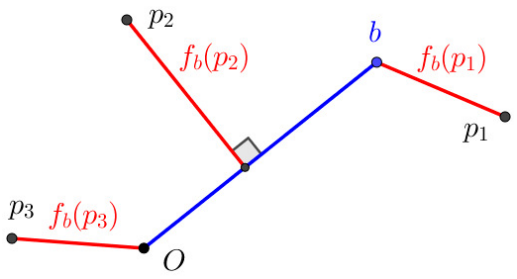
Ha el szeretnénk forgatni a távolságfüggvényt, azt a paraméter ellentétes irányba való forgatásával tehetjük meg. Azaz, ha $R : \mathbb{R}^3 \to \mathbb{R}^3$ forgatás, akkor f távolságfüggvény elforgatása az $f \circ R^{-1}$.

Ugyanígy bármely transzformáció, ami távolság- és szögtartó (egybevágósági transzformáció), alkalmazható így. Több ilyen transzformáció egymás után elvégzése is egy olyan transzformáció, ami rendelkezik ezzel a tulajdonsággal. Tehát ha a transzformációnk $Tr : \mathbb{R}^3 \to \mathbb{R}^3$, akkor f távolságfüggvény (becslés) Tr-rel való transzformáltja $f \circ Tr^{-1}$ is távolságfüggvény (becslés).

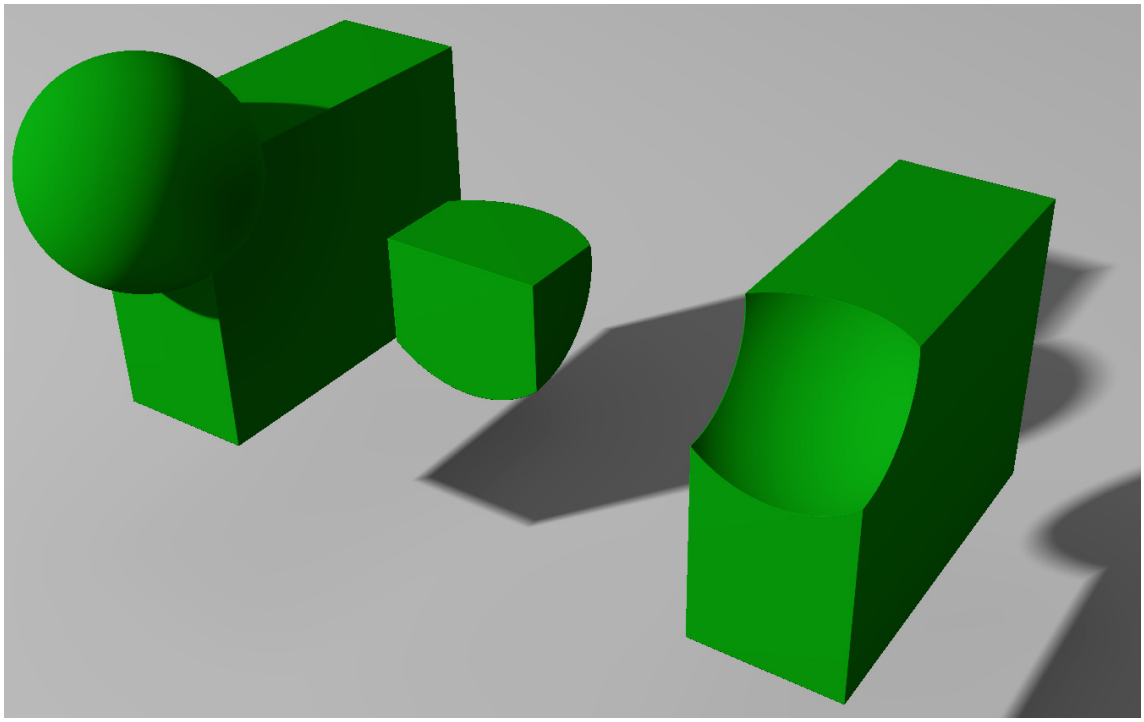
2.6. Műveletek távolságfüggvényekkel

A különböző CSG[4]-műveletek könnyen elvégezhetők távolságfüggvények segítségével (Reiner *et al.* [2011]), melyet a 2.4. ábra demonstrál. Az unió, metszet és különbség halmazműveletek által létrehozott halmazok előjeles távolságfüggvény becsléséről szól a következő tétel.

[4]CSG: Constructive Solid Geometry: A jelenetet egy bináris faként állítjuk össze, ahol a levelek primitív alakzatok (pl. gömb, sík), a belső csúcsok halmazműveletek, az élek pedig affin transzformációk.(Farin [1993])



2.3. ábra. *Pontok távolsága szakasztól*



2.4. ábra. *CSG halmazműveletek rendre: unió, metszet és különbség*

1. Tétel. *Legyen $f, g : \mathbb{R}^3 \to \mathbb{R}$ előjeles távolságfüggvény (becslés), $F := \{f \leq 0\}$ és $G := \{g \leq 0\}$. Ekkor $F \cup G$, $F \cap G$ és $F \setminus G$ halmazok előjeles távolságfüggvény becslései:*

$$h_{F \cup G} = \min(f, g),$$
$$h_{F \cap G} = \max(f, g),$$
$$h_{F \setminus G} = \max(f, -g).$$

Fontos megjegyezni, hogy a tételben a h függvények általában akkor sem pontos előjeles távolságfüggvények, ha f és g az volt.

2.6.1. Ofszet

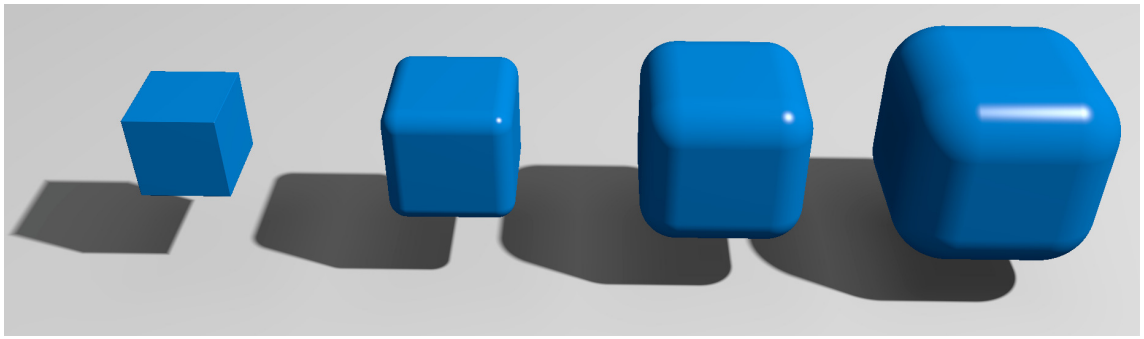
Egy további művelet, aminek a későbbiekben fontos szerepe lesz, az ofszet. Az ofszetelés egy halmaz megadott távolsággal való „bővítését” jelenti.

2.6.1. Definíció. Egy $A \subset \mathbb{R}^3$ halmaz $d \geq 0$ távolsággal való ofszetje:

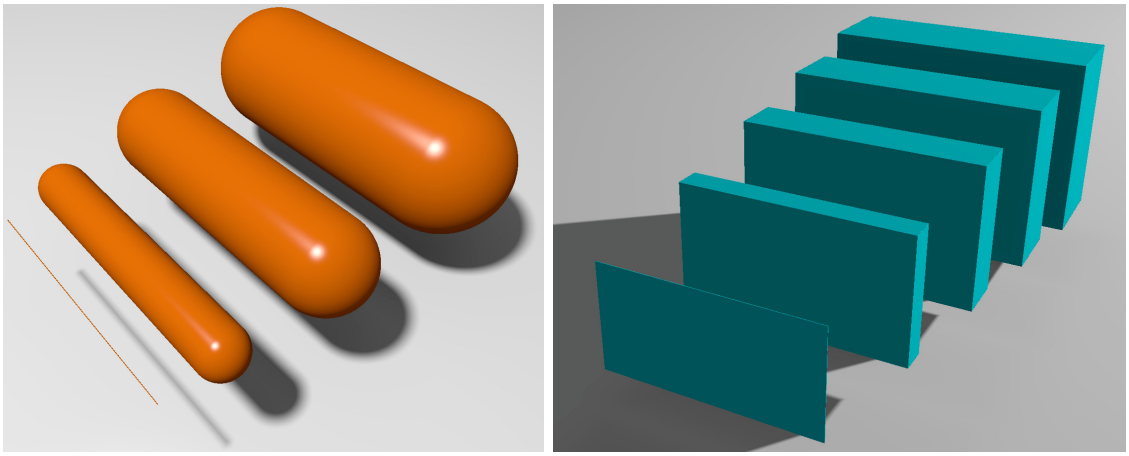
$$S_d(A) := \left\{ \boldsymbol{x} \in \mathbb{R}^3 : d(\boldsymbol{x}, A) \leq d \right\}.$$

Negatív távolsággal való ofszet a halmaz komplementerével definiált:

$$S_{-d}(A) := \mathbb{R}^3 \setminus S_d(\mathbb{R}^3 \setminus A).$$



2.5. ábra. *A bal oldali kocka pozitív ofszetjei sorban növekvő d értékkel*



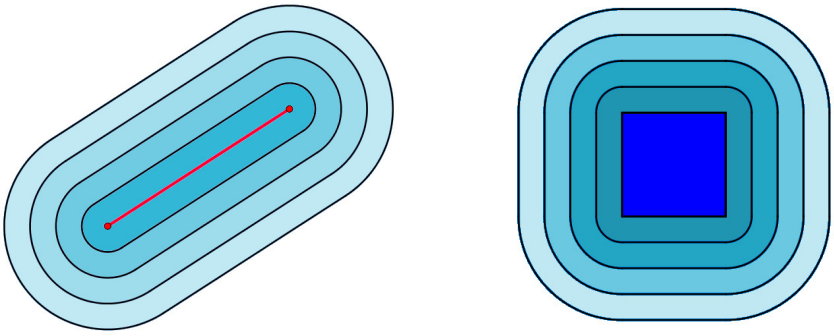
(a) *Szakasz pozitív ofszetjei*

(b) *Téglatest negatív ofszetjei: az eredeti a leghátsó, előtte sorban az egyre kisebb negatív d-vel való ofszetek*

2.6. ábra. *Szakasz és téglatest ofszetjei*

Pozitív d esetén az ofszet a halmazt hizlalja, míg negatív esetben fogyasztja azt. Például a gömb ofszetjei is gömbök, de ez nem általános. A kocka pozitív ofszetjei lekerekített sarkú és élű kockák, míg a negatív ofszetjei kockák (lásd 2.5. ábra). Ugyanakkor egy nem egyenlő élhosszakkal rendelkező téglatest negatív ofszetjei egyre elvékonyodó téglatestek, amik geometriailag nem hasonlóak az eredetihez (lásd 2.6b. ábra). A sík (féltér) ofszetjei szintén síkok. Egy egyenes $d > 0$ ofszetjei az egyenes mint tengely körüli d sugarú végtelen körhengerek. Egy szakasz pozitív ofszetjei félgömbökkel lezárt végű hengerek (kapszulák), ahogy az a 2.6a. ábrán látható.

Egy $f : \mathbb{R}^3 \to \mathbb{R}$ előjeles távolságfüggvény által leírt halmaz $d \in \mathbb{R}$ távolsággal való ofszetjének implicit függvénye az $f - d$ függvény. Sőt, az ofszetelt függvény előjeles távolságfüggvény becslés is, így ez közvetlenül használható lesz megjelenítésre is. Valamint ez a művelet lesz az alapja a cone-tracing módszernek.



(a) *Szakasz pozitív ofszetjei*

(b) *Négyzet pozitív ofszetjei*

2.7. ábra. *Szakasz és négyzet ofszetjei* $\mathbb{R}^2$*-ben*

2.7. Sphere-tracing

A sphere-trace algoritmus sugárkövetési eljárás távolságfüggvények megjelenítésére, ami a raymarch algoritmussal ellentétben fix lépések helyett adaptívan választja a lépésközt, a távolságfüggvény tulajdonságait kihasználva. Az algoritmus nemcsak gyorsabb a raymarch algoritmusnál, hanem pontosabb is, mivel garantálja, hogy mindig a legkisebb gyököt találja meg.

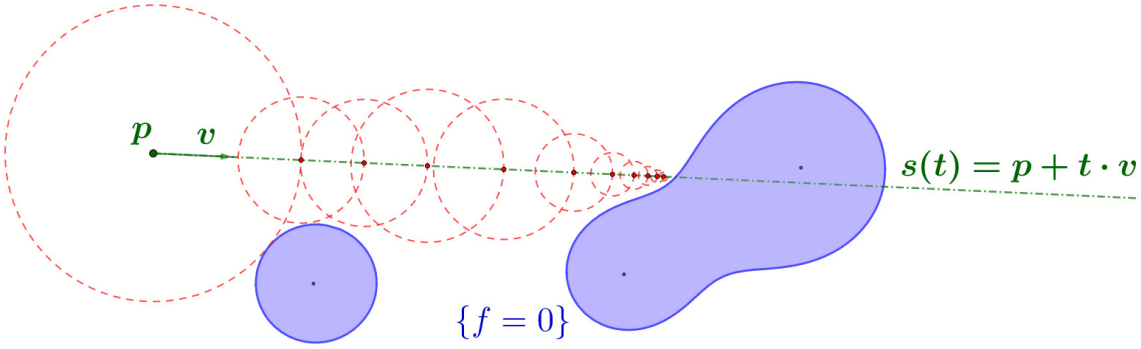
Az algoritmus azon alapszik, hogy ha f távolságfüggvény, akkor az $\boldsymbol{x} \in \mathbb{R}^3$ pont körül az $\boldsymbol{x}$ középpontú $f(\boldsymbol{x})$ sugarú nyílt gömbben nincs felületi pont. Ekkor $f(\boldsymbol{x})$ méretűt mindig léphetünk a sugáron. Az algoritmus a 2. algoritmusvázlatban olvasható. Mivel az algoritmus minden lépésben legfeljebb akkorát lép, hogy ne érjen hozzá a felülethez, a felületen semmiképpen nem fog átlépni, viszont tetszőleges mértékben megközelíti azt.

Az algoritmus megállási feltételei között a gyakorlatban felveszünk egy további elemet: ha a lépés mérete egy meghatározott kis érték alatt van, a keresést termináljuk. Ezzel elkerüljük, hogy az algoritmus fölöslegesen kis lépéseket tegyen a felület közelében. A távolságfüggvény becslésekre a definíció miatt ugyanúgy működik az algoritmus, hiszen azoknál is igaz, hogy az $\boldsymbol{x} \in \mathbb{R}^3$ pont körül az $\boldsymbol{x}$ középpontú $f(\boldsymbol{x})$ sugarú nyílt gömbben nincs felületi pont.

2.8. Cone-tracing

A cone-trace algoritmus azzal egészíti ki a klasszikus sphere-trace algoritmust, hogy a sugár helyett egy egyenes körkúppal vizsgálja a felület metszését. A kúp tengelye az eredeti sugár.

A kúpot négy paraméter írja le, ezek a 2.9. ábrán szemlélhetőek. Az első kettő a tengelyének kezdőpontja $\boldsymbol{p}$ és irányvektora $\boldsymbol{v}$ ($\|\boldsymbol{v}\|_2 = 1$). A harmadik a kúp félnyílásszöge α, aminek szinte kizárólagosan a tangensére van szükség, így ha prog-



2.8. ábra. *Sphere-tracing algoritmus*

In : $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3, \|\boldsymbol{v}\|_2 = 1$ sugár (kezdőpont és irány),
$f : \mathbb{R}^3 \to \mathbb{R}$ távolságfüggvény
Out: $t \in \mathbb{R}^+$ a sugáron megtett távolság
begin
  $t \coloneqq 0, \quad i \coloneqq 0$
  $ft \coloneqq f(\boldsymbol{p} + t\boldsymbol{v})$
  for $ft < t_{max}$ ***and*** $i < i_{max}$ **do**
    $ft \coloneqq f(\boldsymbol{p} + t\boldsymbol{v})$
    $t \coloneqq t + ft$
    $i \coloneqq i + 1$
  end
end

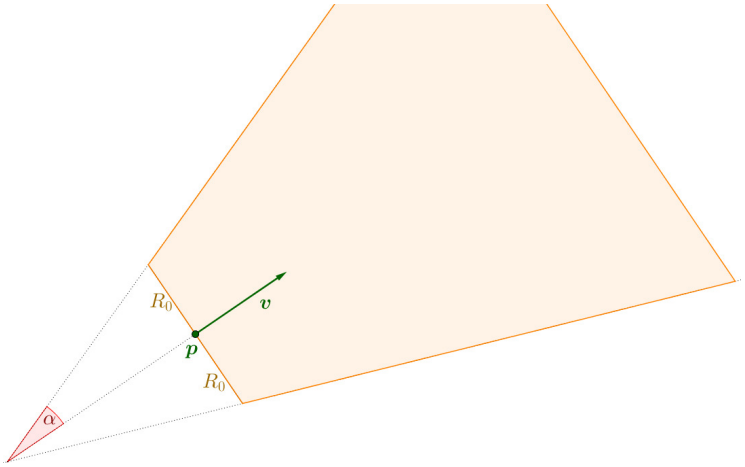
Algoritmus 2: *A klasszikus sphere-trace eljárás.* (Hart [1994])

ramban szerepel, mindig a $\tan\alpha$ értéket tároljuk. Az utolsó paraméter a kúp kezdeti sugara, azaz $t = 0$-ban a kúp körmetszetének sugara. Utóbbi azért hasznos, mert így leírhatunk csonkakúpokat is, vagyis olyan kúpokat, amik csúcsa nem $t = 0$-ban van, sőt hengereket is.

Megjegyzendő továbbá, hogy negatív $\tan\alpha$-nak is van értelme, ekkor a kúp valójában fordítva áll, és a sugáron a csúcsa felé haladunk. Valamint $\tan\alpha = 0$ esetén, ha $R_0 = 0$, visszakapjuk az eredeti sugarat, ha pedig $R_0 \neq 0$, egy végtelen hengert írunk le.

A kúppal való sugárkövetés megvalósítása történhet a raymarch-algoritmus (1. algoritmus) módosításával. A lépéseket a kúp tengelyén végezzük, de a leállási feltételt módosítjuk: akkor állunk meg, ha $f(\boldsymbol{p} + t \cdot \boldsymbol{v}) \geq R_0 + t \cdot \tan\alpha$. Az egyenlőtlenség jobb oldalán a kúp t paraméterbeli sugara áll. Ezzel a módszerrel ugyanazok a problémák, mint az egyszerű raymarch-algoritmussal: a beállított lépésköztől függően az algoritmus túl lassú vagy pontatlan.

A távolságfüggvényt tulajdonságait kihasználva a kúppal való sugárkövetés megvalósítható sphere-tracing segítségével is, az ofszet művelet használatával. A távolságfüggvényt úgy módosítjuk, hogy a sugáron megtett távolság alapján, a kúp su-



2.9. ábra. *A (csonka)kúp paraméterezése, annak keresztmetszetén ábrázolva.* $\boldsymbol{p}$ *a kezdőpont,* $\boldsymbol{v}$ *az irányvektor,* R_0 *a kezdeti sugár és* α *a félnyílásszög.*

garával ofszetelünk:

$$F^*_{\boldsymbol{p},\boldsymbol{v}}(\boldsymbol{x}) \coloneqq f(\boldsymbol{x}) - R_0 - \langle \boldsymbol{v}, \boldsymbol{x} - \boldsymbol{p} \rangle \cdot \tan\alpha$$

A skaláris szorzat az $\boldsymbol{x}$ pontot a tengelyre vetíti, és mivel $\boldsymbol{v}$ egység hosszúságú, ez pontosan a sugár t paramétere.

Annak érdekében, hogy a kapott függvény távolságfüggvény becslés legyen, le kell osztanunk a Lipschitz-konstansával, vagy annak egy felső becslésével (Hart [1994]; Bálint and Valasek [2018a]; Eriksson *et al.* [2004]). Azaz

$$\mathrm{Lip} F^*_{\boldsymbol{p},\boldsymbol{v}} \leq \mathrm{Lip} f + \mathrm{Lip}(\boldsymbol{x} \to \langle \boldsymbol{v}, \boldsymbol{x} - \boldsymbol{p} \rangle \cdot \tan\alpha) \leq 1 + |\tan\alpha|,$$

ugyanis f előjeles távolságfüggvény (becslés), valamint $\|\boldsymbol{v}\|_2 = 1$. Így tehát a megváltoztatott távolságfüggvény a következő:

$$F_{\boldsymbol{p},\boldsymbol{v}}(\boldsymbol{x}) \coloneqq \frac{F^*_{\boldsymbol{p},\boldsymbol{v}}(\boldsymbol{x})}{1 + |\tan\alpha|} = \frac{f(\boldsymbol{x}) - R_0 - \langle \boldsymbol{v}, \boldsymbol{x} - \boldsymbol{p} \rangle \cdot \tan\alpha}{1 + |\tan\alpha|}.$$

A kúppal való sugárkövetés elvégzéséhez tehát a – $\boldsymbol{p}$-től és $\boldsymbol{v}$-től is függő – megváltoztatott $F_{\boldsymbol{p},\boldsymbol{v}}$ függvényen kell végrehajtani az eredeti sphere-tracing algoritmust.

2.9. Távolságfüggvény megjelenítése

A távolságfüggvény által definiált felület megjelenítéséhez az $I \in \mathbb{R}^2 \to \mathbb{R}^3$ képfüggvényt kellene meghatároznunk a virtuális képernyőt leíró paramétertartományon. A képfüggvény a paramétertartományon belül mindenhol értelmezett, viszont nekünk ebből valahogyan diszkrét értékeket kell előállítanunk: véges számú pixel színét kell meghatároznunk. Erre az egyik leggyakrabban használt módszer, hogy az I

függvényt mintavételezzük, például a pixelek középpontjához tartozó paraméterekben. Ekkor viszont elvesztünk minden olyan információt, ami két pixel-középpont között van, azaz kis felbontás esetén vékonyabb objektumok akár el is tűnhetnek az eredményképről.

Ennek javított változata, ha a mintavételezést a cone-tracing segítségével egy olyan kúppal végezzük, ami lefedi a teljes pixelt. Az algoritmus így nem hagyhat ki semmilyen objektumot, hiszen a teljes paraméterteret lefedjük. Ezt a viselkedést demonstrálja, hogy egyetlen dimenziós objektumok is láthatóvá válnak, például a szakasz a 2.6. ábrán.

Felületi pont Amikor az algoritmus leáll és metszést talál egy felülettel, valójában a felülettől egy meghatározott távolságra áll meg. Az így kapott „felületi pont" tehát nincs rajta a felületen, sőt körülötte egy egész gömbben nincs semmi. A gömb sugara megegyezik a pontban a távolságfüggvény értékével, vagy ha távolságfüggvény becslésünk van, akkor a sugár egy alsó becslését ismerjük. Ez a gömb a cone-tracingnek köszönhetően éppen akkora, ami lefedi az éppen kirajzolni kívánt pixelt, így ez jól reprezentálja a pixelen keresztül látható felületdarabot. További előnye a felülettől való távolabbi megállásnak, hogy az innen indított további sugarak – például az árnyaláshoz – nem fognak nagyon lassan indulni.

A dolgozatban ezután felületi pontként a cone-trace algoritmus megállási helyére hivatkozunk, és a köré elképzelt gömb pedig az itt definiált – a felületet nem metsző – gömb. Ezt a gömböt fogjuk használni a puha árnyék egysugaras számításánál is.

3. fejezet

Árnyékok számítása

A vetett árnyékok számításánál figyelembe kell vennünk, hogy az árnyékok széle elmosódik, azaz általában nem húzható egy éles határ a teljes árnyék és a teljesen megvilágított felület között. Ennek oka, hogy a fényt kibocsájtó objektum nem pontszerű, így a különböző irányból érkező fénysugarak egy része akadály nélkül eljuthat egy adott pontra, míg más fénysugarakat kitakarhat egy objektum. Ezt a hatást könnyen ellenőrizhetjük egy lámpával: egy tetszőleges tárgy árnyéka sokkal élesebb, ha közelebb van a felülethez, amire az árnyéka vetül (valamint akkor is élesebb, ha a fényforrás messzebb kerül). Ugyanez a jelenség figyelhető meg a különböző típusú nap- és holdfogyatkozásokkor. Teljes napfogyatkozáskor a megfigyelő a Hold teljes árnyékában van (umbra), míg részleges napfogyatkozáskor a megfigyelő továbbra is látja a Nap egy részét (penumbra).

3.1. Kemény árnyékok számítása

Kemény árnyéknak nevezzük a grafikában az árnyékok olyan módú számolását, amikor az árnyék és a teljes fény között nincs átmenet, azaz egy hirtelen váltás van. Ennek számolásához a felületi pontból a fényforrás irányába egy úgynevezett árnyéksugarat indítunk, amiről azt vizsgáljuk, hogy eléri-e a fényforrást. Ha nem éri el, vagyis beleütközik egy felületbe, a felületi pont árnyékban van, míg ha eléri a fényforrást, akkor nincs árnyékban.

Az implicit felületek megjelenítésénél – és így a távolságfüggvényeknél is – ez egyszerűen kivitelezhető, hiszen alkalmazható a felületi pont megtalálására használt algoritmus. Ez általános implicit függvénynél a raymarch algoritmus (1. algoritmus), míg távolságfüggvények esetén a sphere-trace algoritmus (2. algoritmus).

Az árnyékosság leírható a láthatósági függvénnyel: $V \in \mathbb{R}^3 \to \{0, 1\}$. A felületi pontból a fény irányába mutató vektor ω, ekkor a $V(\omega)$ értéke 0, ha a pont és a fényforrás között van takaró objektum, és 1, ha akadály nélkül elérhető a fényforrás.

3.2. Puha árnyékok

A részleges árnyék számolásához a következő modellt alkalmazzuk. A fényforrásunk egy gömb alakú sugárzó test, amely felszínéről minden irányban azonos mennyiségű fényt bocsájt ki. A kiszámítandó érték a felületi pontba a fényforrásból egyenes úton eljutó fény aránya a potenciális teljes fényhez képest. A különböző visszaverődésekből érkező fénysugarakat nem vesszük figyelembe, a fény ilyen módon való számolása meglehetősen költséges. Az ilyen típusú visszaverődésekkel is foglalkozó fénymodellek a globális illuminációs modellek. A dolgozatban csak a fényforrás felszínéről a felületi pontba közvetlenül eljutó fénysugarakat vesszük figyelembe. A beérkező fény és a lehetséges teljes fény arányát a következő képlet adja:

$$\frac{\int_L V(\omega)\,\mathrm{d}\omega}{\int_L 1\,\mathrm{d}\omega},$$

ahol L azon irányok halmaza, amely irányokban a felületi pontból a fényforrás látszódna, ha nem lenne a pont és a fényforrás között semmilyen takaró objektum. A $V : L \to \{0, 1\}$ a láthatósági függvény, $V(\omega)$ azt adja meg, hogy a felületi pontból látható-e a fényforrás az ω irányban, vagy egy árnyékoló objektum eltakarja.

Az L irányhamaz gömb alakú fényforrás esetén egy kúp csúcsából a kúp belsejébe eső irányok, ahol a kúp csúcsa a felületi pont, a kúpfelület pedig kívülről érinti a gömböt. Ezek az irányok tekinthetők a kúp egy körmetszetén lévő pontokként.

Általánosabban egy nagyobb területre is értelmezhető a beérkező fényarány. Ha a 2.9. alfejezetben tárgyalt módon értelmezzük a felületi pont körüli gömböt, illeszthetjük a kúpot a két gömb – fényforrás és felületi pont – köré úgy, hogy mindkettőt kívülről érintse. Ebben az esetben egy egyszerűsítéssel élünk, csak azokat a sugarakat vizsgáljuk, amik a kúp csúcsán átmennek – természetese csak a felületi pont közeléből indítva. Ekkor az előbbihez hasonlóan ezek az irányok is leírhatóak egy körlap pontjaival.

3.3. Árnyék számolása Monte-Carlo integrállal

A puha árnyék számolásának egy lehetséges módja a Monte-Carlo integrálás (Andersson and Barré-Briseboi [2018]). A módszer kifejezetten számításigényes, viszont jó összehasonlítási alapot nyújt az új algoritmusokhoz.

A módszer integrálok közelítésére használható mintavételezéssel. A legegyszerűbb esetben, amikor az integrál egy $g : [a, b] \to \mathbb{R}^+$ nemnegatív korlátos függvény alatti terület, a mintavételezést egy téglalapon végezzük egyenletes eloszlással[1]. A téglalap alja az x-tengely, vízszintes határai az a és b értékeknél, a felső határa pedig

[1] Alkalmazható más eloszlás is, ekkor az eloszlásnak megfelelően súlyozni kell a mintákat.

a függvény felső korlátjánál van. Egy (x, y) mintánál azt vizsgáljuk, hogy a pont a függvény alatt van-e, azaz $y < g(x)$. A terület méretére, azaz az integrál értékére a közelítés:

$$\int_a^b g \approx T \cdot \frac{k}{n},$$

ahol T a mintavételezett téglalap területe, k a teszten átment minták – függvény alatti pontok – száma és n a teljes mintaszám.

A véletlen pontok választásához kis diszkrepanciájú pszeudorandom sorozatokat érdemes használni, hogy minél kisebb mintaszámmal tudjunk megfelelő pontosságot elérni. Mi erre a feladatra a Halton-sorozatokat használtuk. Az új algoritmusaink validálására a tárgyalt módon implementáltuk a puha árnyékok számítását a Monte-Carlo-integrálás felhasználásával. A referenciaalgoritmusból származó képek a 6. fejezetben láthatók összehasonlításként.

3.3.1. Monte-Carlo implementálása kúpon belül

A puha árnyék számolásánál a kúp egy körmetszetén mintavételezünk és azt vizsgáljuk, hogy a ponthoz tartozó irányban látható-e a fényforrás, azaz kiértékeljük a $V(\omega)$-t. A kérdéses terület itt az, ahol V értéke 1. A kör területével nem kell beszorozni, mert végeredményül csak megvilágítás aránya kell, azaz a területnek is csak a megvilágított rész arányát kell kiszámítani.

A kör területén területén kétféleképpen is lehet numerikus Monte-Carlo-integrálást végrehajtani. Mindkét módszer azon alapszik, hogy polár integrál transzformációt hajtunk végre a $V(\omega)$-n. Ekkor az integrandus megszorzódik a síkbeli polártranszformáció deriváltjának determinánsával, r-el.

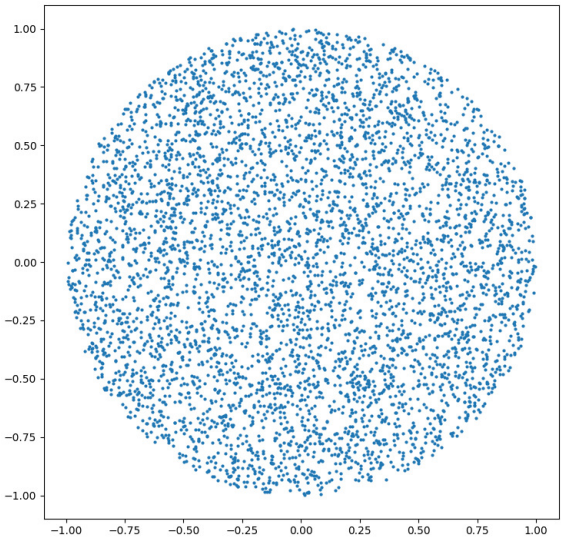
$$\int_L V(\omega)\,\mathrm{d}\omega = \int_0^{2\pi} \int_0^1 V(\varphi, r) \cdot r\,\mathrm{d}r\,\mathrm{d}\varphi$$

Az egyik módszer szerint a egyenletes eloszlású mintavételezés során a mintavételi pontokat annak r polárkoordinátájával súlyozzuk. A másik módszer szerint $r := \sqrt{u}$ helyettesítést végzünk el, mely kiejti az r-es szorzót az integrálból, tehát

$$\int_0^{2\pi} \int_0^1 V(\varphi, r) \cdot r\,\mathrm{d}r\,\mathrm{d}\varphi = \frac{1}{2} \int_0^{2\pi} \int_0^1 V\left(\varphi, \sqrt{u}\right) \mathrm{d}u\,\mathrm{d}\varphi\ .$$

Tehát eme utóbbi esetben véletlen számokat kell választani a $\phi \in [0, 2\pi]$, valamint a $u \in [0, 1]$ intervallumból, majd az utóbbiból gyököt kell vonni. Ekkor a $(\varphi, \sqrt{u})$ polárkoordinátás mintavételezés egyenletes lesz az egységkörön ahogy azt a 3.1. ábra[2] szemlélteti. A két mintavételezés egyformán alkalmas puha árnyék számolására.

[2] Az ábra forrása: https://stackoverflow.com/questions/5837572/generate-a-random-point-within-a-circle-uniformly



3.1. ábra. *Egyenletes mintavételezés az egységkörön*

4. fejezet

Egysugaras árnyékszámítás

A fejezetben bemutatott új algoritmusok a 3.2. alfejezetben tárgyalt puha árnyékot közelítik egy felületi pontban. A felületi pont köré elképzelt – a kirajzolt pixelt lefedő – gömbre (lásd 2.9. alfejezet) és a fényforrás gömbjére könnyű olyan kúpot illeszteni, ami mindkét gömböt kívülről érinti (4.1. ábra). Az algoritmusok ezen a kúpon belül keresnek árnyékolókat a conte-trace-algoritmus segítségével.

4.1. Geometriai interpretáció

Az árnyék vizsgálatához a felületi pont és a fényforrás között elhelyezkedő objektumokat a kúp tengelyére merőleges körmetszetre vetítjük a kúp csúcsából középpontosan (vagy a tengellyel párhuzamosan, ha a két gömb azonos sugarú, és valójában egy hengerünk van). Ez a vetület azonos lesz minden körmetszeten, csak különböző méretekre skálázva, ezért az egyszerűség kedvéért a kört egységnyi sugarúnak vesszük. A tengely képe a vetületen eme kör középpontja. A felületi pontba ténylegesen eljutó fény és a takarás nélküli maximálisan eljutó fény aránya megegyezik a vetületben a nem árnyékolt terület nagyságának és a teljes kör területének arányával.

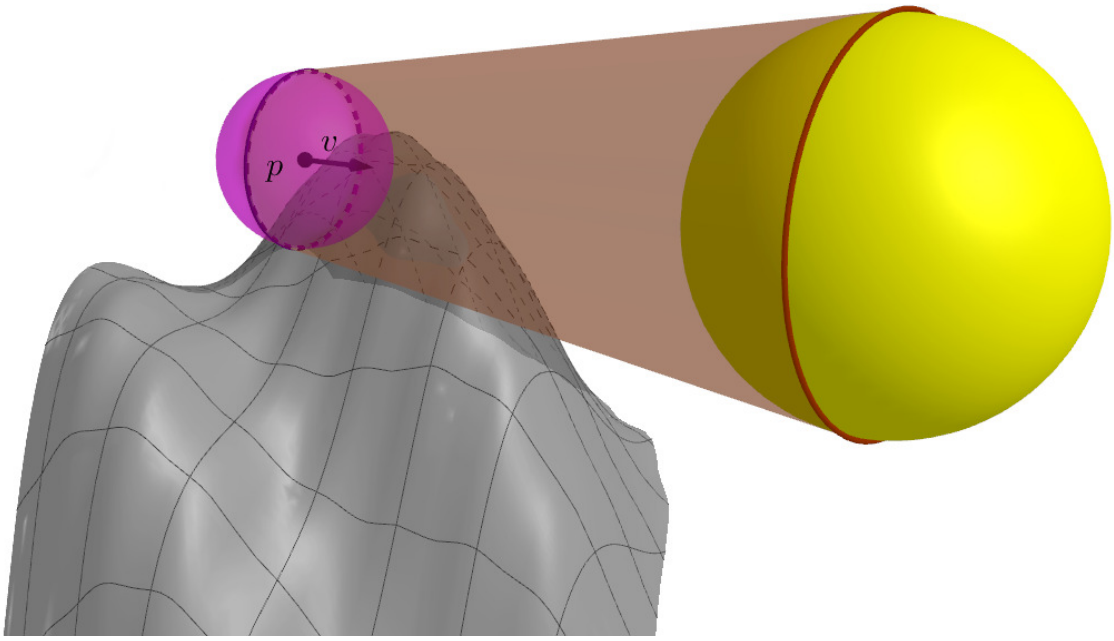
A fényforrás és a felületi pont között a kúp tengelyén vizsgálva a távolságfüggvényt arról kapunk információt, hogy a kúpba mennyire lógnak bele árnyékoló felületek.

4.1.1. Definíció. Egy csonkakúp $t \in \mathbb{R}$ paraméterében egy $h \in \mathbb{R}$ távolságérték relatív mérete:

$$\varrho(h;t) = \frac{h}{R_0 + t \cdot \tan\alpha},$$

ahol a nevezőben lévő érték a kúp t-beli körmetszetének sugara. R_0 a (csonka)kúp kezdeti sugara és α a kúp félnyílásszöge.

A kúp paramétereit lásd a 2.8. alfejezetben a 2.9. ábrán. Ez a relatív távolság felel meg a körvetületen a középponttól mért távolságnak.



4.1. ábra. *A felületi pont gömbjére és a fényforrás gömbjére illesztett csonkakúp*

Az árnyékot vető felületrészek közül a legnagyobb vetületűt keressük, ezzel a vetülettel fogunk számolni. A vetületről feltételezzük, hogy ennek a legkisebb a relatív távolsága a tengelytől, vagyis a vetületen a középponttól vett előjeles távolsága. Ha a kör középpontja egy objektum vetületének belsejében van, akkor az előjeles távolság negatív, ennek megfelelően az objektumot a tengely valahol elmetszette, és a belsejében haladt. Legyen r_{min} a legkisebb relatív távolság:

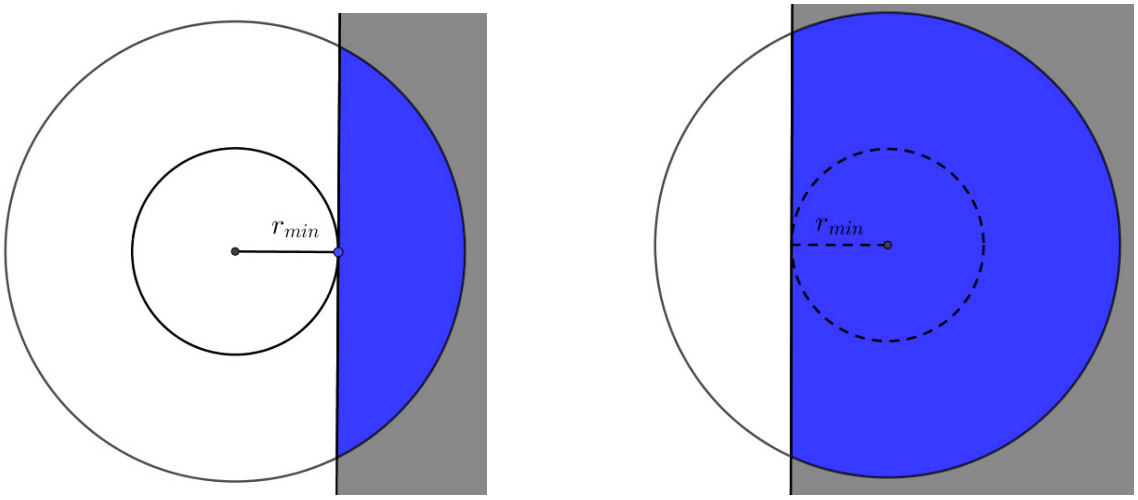
$$r_{min} = \min_{t\in[0,d]} \varrho\left(f\left(\boldsymbol{s}\left(t\right)\right);t\right) = \min_{t\in[0,d]} \frac{f(\boldsymbol{s}(t))}{R_0 + t\cdot\tan\alpha}.$$

Ha $r_{min} \geq 1$, akkor a kúpba nem lóg bele semmilyen árnyékoló, azaz a felületi pontot eléri a teljes fénymennyiség. Ha $r_{min} \leq -1$, a kúpba úgy lóg bele egy felület, hogy az a teljes fényforrást eltakarja a felületi pontból nézve. Míg ha a számolt minimum $r_{min} \in (-1,1)$, a fényforrást csak részlegesen takarja a legjobban belógó árnyékoló. Részleges árnyéknál például az $r_{min} = 0$ érték az jelenti, hogy a tengely éppen érint egy felületet, de nem megy bele egy felület belsejébe sem. Részleges árnyékot okozó felületre látható példa a 4.2. ábrán.

A megvilágítottság becsléséhez bevezetjük a (normalizált) megvilágítottsági függvényt:

$$\tau : [-1,1] \to [0,1]\,.$$

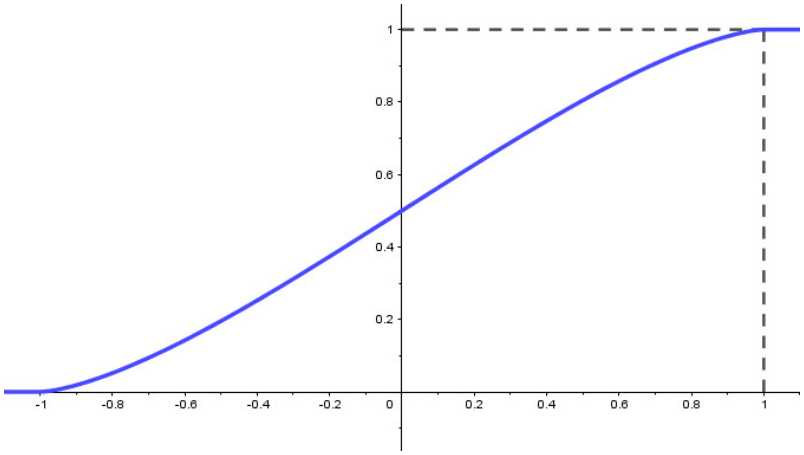
A függvény paramétere az előbb tárgyalt sugár (r_{min}), értéke pedig a megvilágítottság mértéke ($\tau(-1) = 0$ jelenti a teljes árnyékot, és $\tau(1) = 1$ jelenti a teljes fényt). A függvény szigorúan monoton növő, minél messzebb van a belógó felület, annál több fény jut át. A függvényt természetesen kiterjeszthető -1-nél kisebb és 1-nél



(a) *A tengely nem metszi az objektumot,* $r_{min} > 0$

(b) *A tengely átmegy az objektumon,* $r_{min} < 0$

4.2. ábra. *Síkszerű objektum vetülete a vizsgált körön. Az árnyékoltsági arány a sötétkék terület aránya a teljes körben.*

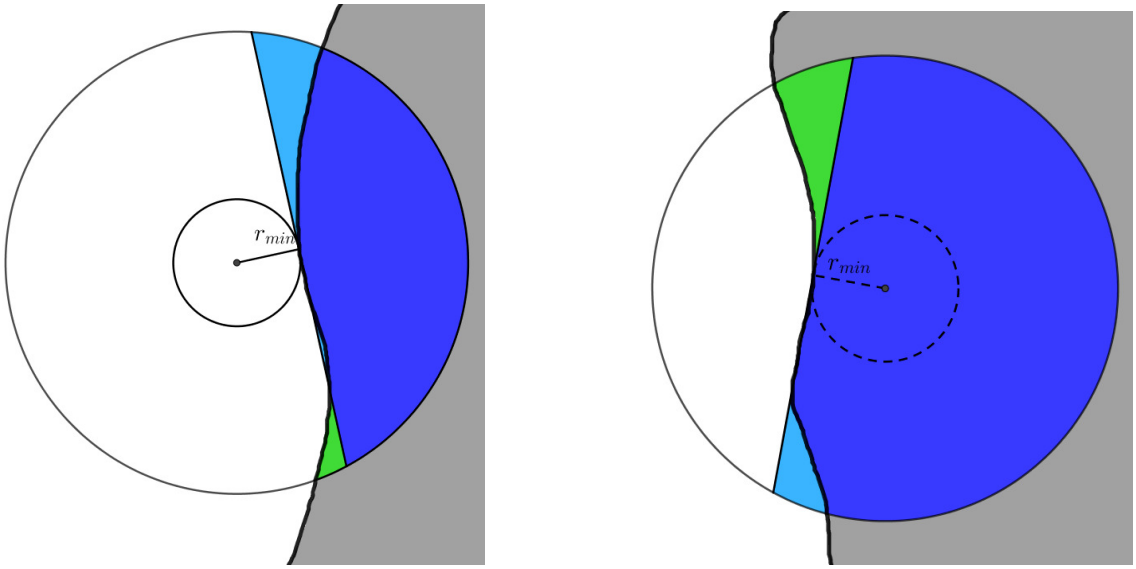


4.3. ábra. *A megvilágítottsági függvény grafikonja.*

nagyobb értékekre értelemszerűen, a szűkebb értelmezésre az invertálhatóság miatt van szükség. A kiterjesztett függvény grafikonja a 4.3 ábrán látható.

$$\hat{\tau}(r) = \begin{cases} 0, & \text{ha } r < -1, \\ \tau(r), & \text{ha } r \in [-1, 1], \\ 1, & \text{ha } 1 < r. \end{cases}$$

A megvilágítottság számolásához feltételezzük, hogy az árnyékot egyedül a legnagyobb árnyékoló okozza. Ez a feltételezés természetesen nem mindig helyes, de a sebesség jelentős romlása nélkül ennél több információ nem nyerhető ki a távolságfüggvényből. Az ebből következő hibákról a 6.5. alfejezetben olvasható és látható további információ.



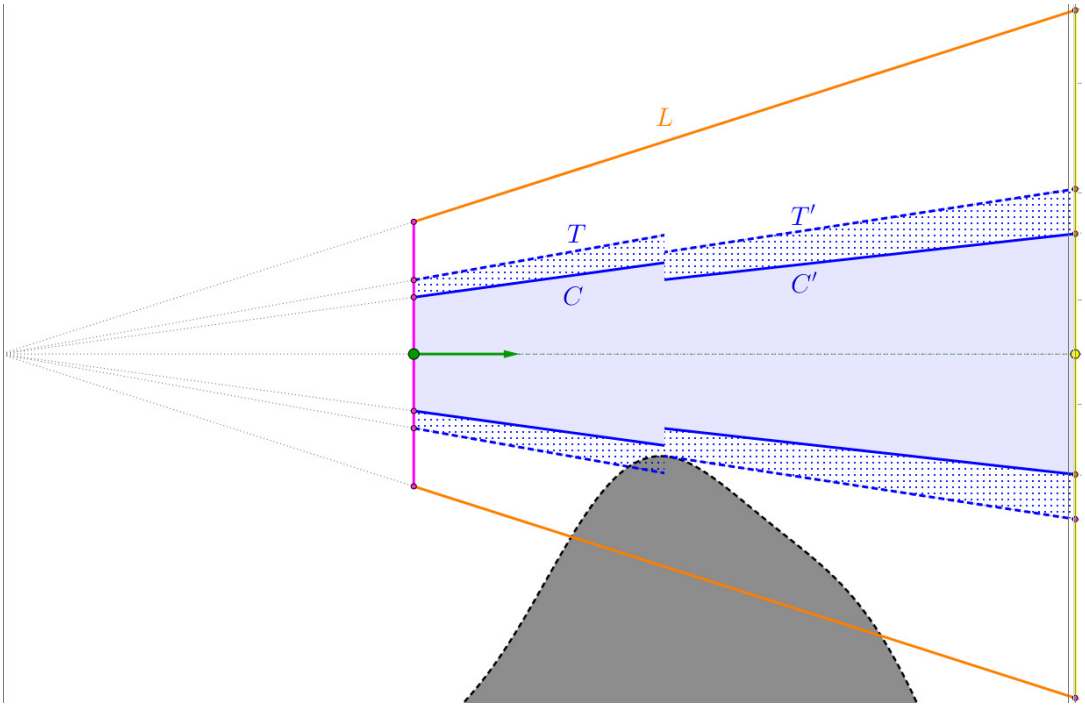
4.4. ábra. *Kevésbé sima felület vetülete a vizsgált körön. A felületet az érintősíkjával közelítjük. A becsült árnyékoltsági arány a sötétkék és világoskék területek együttes aránya a teljes körben. A világoskék terület fölösleges rész, amit nem kellene számolni, míg a világoszöld területet kihagytuk a számolásból, pedig az árnyékvetület része.*

Egy további feltételezésünk, hogy az árnyékot okozó felület (lokálisan) síkszerű. Ez a feltételezés a legtöbb esetben helytáll, hiszen a felületnek csak kis része metszi a kúpot (hiszen a kúp nagyon vékony), és minden sima felület jól közelíthető egy pontban az érintősíkjával. Kevésbé sima felület esetén ez valamennyi hibát okozhat, erre mutat példát a 4.4. ábra. Az érintősík a kúp tengelyével párhuzamos lesz, hiszen ha nem lenne az, akkor a felület nem itt lenne a legközelebb a tengelyhez, azaz a számolt minimum kisebb lenne. Az érintő síkot a körre vetítve tehát egy egyenest kapunk, aminek a kör középpontjától való távolsága $|r_{min}|$, ez két részre osztja a körlapot. Az feltételezésünk miatt (miszerint csak az éppen vizsgált árnyékoló számít) a kör kitakart felén nem megy át a fény, míg a másik felén úgy vesszük, hogy a fény akadály nélkül áthalad. A kör két részének a területe kiszámítható egy paraméteres integrál segítségével:

$$\pi \cdot \tau(r) = \int_{t=-1}^{r} 2\sqrt{1-t^2}\,\mathrm{d}t.$$

Szinuszos helyettesítést végezve (és a kör területével átosztva) a következő eredményt kapjuk (Csaba [2016]):

$$\tau(r) = \frac{1}{2} + \frac{r\sqrt{1-r^2} + \arcsin r}{\pi}.$$



4.5. ábra. *Puhaárnyék-algoritmus adaptív minimumkeresésének egy lépése. Az algoritmus lokális minimumot talált ezért a C és T kúpokon csökkentünk.*

4.2. Az algoritmus vázlata

Az algoritmusnak van egy érzékenységi paramétere (ϵ), ami az a minimális fényarány-különbség, amit az algoritmus minimálisan érzékel, azaz az algoritmus maximális hibája. A paraméter értéke és az elért képminőség között direkt kapcsolat van, valamint az algoritmus futásidejére is közvetlen hatása van. A gyakorlat azt mutatta, hogy általában a paraméternél valójában sokkal nagyobb pontosságot ér el az algoritmus. Az eredmények a 6. fejezetben olvashatók.

A feladat tehát $\tau(r_{min})$ meghatározása. A klasszikus sphere-trace algoritmus nem működik, mivel részleges árnyék úgy is keletkezhet, hogy a sugár áthalad felületeken, és a sphere-trace a metszéspont előtt megállna. A cone-trace algoritmus viszont képes erre, ha a kúp sugarát menet közben állítjuk és megengedünk negatív értékeket is. Elképzelni nehezebb a negatív sugarú – kifordított – kúppal való sugárkövetést, de matematikailag nincs különbség a működésében.

Legyen a két gömbre illesztett kúp az L kúp (*Light*). Ha ezen a kúpon belül veszünk az L-lel közös tengelyű és közös csúccsal rendelkező kúpokat, azok relatív sugara (körmetszetük sugarának relatív hossza) az L kúphoz képest állandó lesz. Tehát, ha például egy csonkakúpot definiáló két körlap sugarait ugyanazon számmal szorozzuk meg, az új alkotóegyenesek ugyanazon pontban metszik egymást mint a régiek. Az algoritmusban kettő az L-hez relatív konstans távolságra lévő kúpot fogunk használni amit T, illetve C-vel jelölünk (*Threshold* és *Cone-trace* nevekből

származtatva). A kúpok relatív sugarát jelölje R^T és R^C. (Természetesen az L relatív sugara $R^L = 1$.) Legyen α^L, α^T és α^C a kúpok félnyílásszöge. Ezek között fennállnak az alábbi lineáris összefüggések:

$$\tan \alpha^T = R^T \cdot \tan \alpha^L \,, \qquad \tan \alpha^C = R^C \cdot \tan \alpha^C.$$

Ezenkívül legyenek a kúpok kezdősugarai ($t = 0$-ban a körmetszet sugara) R_0^L, R_0^T és R_0^C. Ezek között szintén lineáris a kapcsolat:

$$R_0^T = R^T \cdot R_0^L \,, \qquad R_0^C = R^C \cdot R_0^L.$$

A T és C kúpok nyílásszögét fogjuk az algoritmusban addig csökkenteni, amíg R^T-vel el nem érjük az r_{min} értékét. A sugárkövetést a cone-trace algoritmusnak megfelelően a C kúppal végezzük. A C kúp mindig a T kúp belsejében van ($R^C < R^T$), és minden esetben, amikor észleljük, hogy árnyékoló felület kerül a T és C kúpok közötti térbe, csökkentjük R^T értékét az új minimumra. A feltétel:

$$\varrho\left(f\left(\boldsymbol{s}\left(t\right)\right);t\right) \leq R^T \Longleftrightarrow f\left(\boldsymbol{s}\left(t\right)\right) \leq R_0^T + t \cdot \tan \alpha^T.$$

A C kúpot pedig folyamatosan annyival tartjuk T alatt, hogy ϵ mértékű eltérést észrevegyünk, azaz:

$$\tau\left(R^T\right) = \tau\left(R^C\right) + \epsilon \Longrightarrow R^C = \tau^{-1}\left(\tau\left(R^T\right) - \epsilon\right).$$

4.3. Algoritmus

Az egysugaras puhaárnyék-számító algoritmus a korábbiakban leírtaknak megfelelően a 3. algoritmusban látható formát ölti. Az ϵ érzékenységi paraméternek helyes meghatározása fontos az algoritmus sebessége és a végeredmény pontosságának szempontjából. Az erre vonatkozó eredmények a 6. fejezetben találhatók.

4.3.1. Az inverz eltakarás függvény számítása

Az algoritmusnak szüksége van a τ függvény inverzére ahhoz, hogy az elvárt pontosságot elérje. A függvény inverzét, azaz $m \in [0, 1]$-re a $\tau^{-1}(m)$ értéket numerikusan közelítjük. A megoldás szinuszára felírható egyszerűen egy fixponti egyenlet, de sajnos az így kapott iteráció nem konvergál a teljes intervallumon, ugyanis a kapott függvény nem kontrakció. Newton-iteráció sem alkalmazható, mert a két végpontban (-1-ben és 1-ben) τ deriváltja 0. Mivel τ szigorúan monoton növő, használható az intervallumfelezés módszere.

In : $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3, \|\boldsymbol{v}\|_2 = 1$ sugár (kezdőpont és irány),
$f : \mathbb{R}^3 \to \mathbb{R}$ távolságfüggvény,
$R_0^L, R_1^L \in [0, +\infty)$ a kúp kezdeti és végső sugara,
$d \in (0, +\infty)$ fényforrás távolsága $\boldsymbol{p}$-től,
$\epsilon = \frac{1}{256}$ az algoritmus érzékenysége,
$\tau : [-1, 1] \to [0, 1]$ megvilágítottsági függvény
Out: $m \in [0, 1]$ $\boldsymbol{p}$-be jutó fény aránya
begin
  $t \coloneqq 0; \quad m \coloneqq 1$
  $\tan \alpha^L \coloneqq \frac{R_1^L - R_0^L}{d}$
  $R^C \coloneqq \tau^{-1}(1 - \epsilon), \quad R_0^C \coloneqq R^C \cdot R_0^L, \quad \tan \alpha^C \coloneqq R^C \cdot \tan \alpha^L$
  $R^T \coloneqq 1, \quad R_0^T \coloneqq R^T \cdot R_0^L, \quad \tan \alpha^T \coloneqq R^T \cdot \tan \alpha^L$
  while $t < d$ ***and*** $m > \epsilon$ ***and*** $i < i_{max}$ **do**
    $F \coloneqq f(\boldsymbol{p} + t \cdot \boldsymbol{v})$
    if $F < R_0^T + t \cdot \tan \alpha^T$ **then**
      $R^T \coloneqq \frac{F}{R_0^L + t \cdot \tan \alpha^L}$
      $m \coloneqq \tau\left(R^T\right)$
      $R^C \coloneqq \tau^{-1}(m - \epsilon), \quad R_0^C \coloneqq R^C \cdot R_0^L, \quad \tan \alpha^C \coloneqq R^C \cdot \tan \alpha^L$
      $R_0^T \coloneqq R^T \cdot R_0^L, \quad \tan \alpha^T \coloneqq R^T \cdot \tan \alpha^L$
    $t \coloneqq t + \frac{F - t \cdot \tan \alpha^C - R_0^C}{1 + |\tan \alpha^C|}$
    $i \coloneqq i + 1$
  end
end

Algoritmus 3: *Az egysugaras árnyékszámítás alapalgoritmusa.*

4.3.2. A sugárkövetés iránya

Az eddigiekben a számolást a felületi pont irányából végeztük. Mivel az algoritmusban a minimumkeresés két tetszőleges gömb közötti csonkakúpban történik, elegendő iteráció esetében az algoritmus eredménye független a gömbök megadási sorrendjétől. Így a kiszámítandó érték szempontjából érdektelen, hogy a felületi pontot és a fényforrást összekötő csonkakúpon milyen irányban haladunk végig, hiszen az egyetlen kérdés, hogy a kúpba milyen mértékben lógnak bele a környező felületek. Ennek következtében az algoritmus ugyanúgy működik abban az esetben is, ha megcseréljük a haladási irányt. Ez technikailag csak annyit jelent, hogy az algoritmusnak fordítva kell megadni a paramétereit: a sugár a fényforrástól indul – és iránya -1-szerese az eredetinek –, valamint a kúp kezdeti és végső sugara megcserélődik.

Az algoritmus ilyenkor természetesen teljesen más módon számítja az eredményt, de a végeredmény ugyanúgy a kérdéses integrál ϵ pontosságú közelítése. Az algoritmus sebessége a fényforrás, a felületi pont és a színtér geometriájától függ, mely különböző lehet egyik, illetve másik irányban. A két irány összehasonlítására láthatók eredmények a 6. fejezetben.

4.3.3. Kemény árnyék számítása

Az algoritmus képes kemény árnyék számítására is. Ehhez elég, ha a fényforrás sugarát 0-ra állítjuk, így kapva egy pontfényforrást. Ekkor az algoritmusban minden változatlanul történik, mindössze csonka kúp helyett egy kúpot definiálunk.

Valójában még ennél is jobb a helyzet, hiszen továbbra is lesz egy átmenet az árnyék szélén, viszont ez a felületi pont gömbje miatt legfeljebb egy pixel széles a képernyőn. Ez a gyakorlatban azt jelenti, hogy az algoritmus által számolt kemény árnyék olyan, mintha anti-aliasingot alkalmaztunk volna rá, azaz az árnyék határa nem recés. Példa erre az 1.1. ábrán látható.

Megjegyzendő, hogy itt a felületi pont körüli gömb sugara nem állítható a fényforrás sugarával egyszerre 0-ra, ugyanis ekkor az algoritmus nem működőképes. (Ugyanakkor a klasszikus sphere-trace algoritmus pontosan az így meghatározott „0 sugarú kúpot", azaz félegyenest járja be.) Ez természetesen nem okoz gyakorlati problémát, hiszen az előbbi sugár a 2.9. alfejezetben meghatározott módon számolódik, vagyis sosem lesz nulla sugarú.

4.4. Algoritmusvariációk

A puhaárnyék-számító algoritmusnak több változata is készült. Ezek célja a számítás meggyorsítása volt, az eredmény pontosságának megtartása mellett.

4.4.1. Az algoritmus „linearizált" változata

Az algoritmus célja meghatározni a $\tau\left(r_{min}\right)$ értéket egy megadott pontossággal. Az alapalgoritmus τ és τ^{-1} értékeit használja arra, hogy minden lépésben a megengedett legnagyobb hibával számoljon. Ha τ helyett egyszerűen lineárisan állítjuk a maximális hibáért felelős R^T-t, megspóroljuk az összes τ és τ^{-1} kiértékelést, és elég a legvégén egyszer behelyettesíteni r_{min}-t a τ függvénybe. Ekkor természetesen figyelni kell arra, hogy továbbra is megtartsuk a megkövetelt pontosságot. Ehhez tekintsük a τ deriváltjának maximumát. Ez $r = 0$-ban van: $\tau'\left(0\right) = \frac{2}{\pi}$. Ez azt jelenti tehát, hogy τ legmeredekebb részéhez közel a függvényben bekövetkező ϵ mértékű változáshoz $\frac{\epsilon \cdot \pi}{2}$-vel kell megváltoztatni a függvény paraméterét. Mivel a függvény többi részén ennél kisebb a deriváltja – így kevésbé meredek –, ha mindenhol ekkorát lépünk, akkor továbbra is megmarad a kívánt pontosság. A megváltoztatott algoritmus pszeudokódja a 4. algoritmusban olvasható.

In : $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3, \|\boldsymbol{v}\|_2 = 1$ sugár (kezdőpont és irány),
$f : \mathbb{R}^3 \to \mathbb{R}$ távolságfüggvény,
$R_0^L, R_1^L \in [0, +\infty)$ a kúp kezdeti és végső sugara,
$d \in (0, +\infty)$ fényforrás távolsága $\boldsymbol{p}$-től,
$\epsilon = \frac{1}{256}$ az algoritmus érzékenysége,
$\tau : [-1, 1] \to [0, 1]$ megvilágítottsági függvény
Out: $m \in [0, 1]$ $\boldsymbol{p}$-be jutó fény aránya
begin
  $t \coloneqq 0, \quad e \coloneqq \frac{\epsilon \cdot \pi}{2}$
  $\tan \alpha^L \coloneqq \frac{R_1^L - R_0^L}{d}$
  $R^C \coloneqq 1 - e, \quad R_0^C \coloneqq R^C \cdot R_0^L, \quad \tan \alpha^C \coloneqq R^C \cdot \tan \alpha^L$
  $R^T \coloneqq 1, \quad R_0^T \coloneqq R^T \cdot R_0^L, \quad \tan \alpha^T \coloneqq R^T \cdot \tan \alpha^L$
  while $t < d$ ***and*** $R^T > e$ ***and*** $i < i_{max}$ **do**
    $F = f(\boldsymbol{p} + t \cdot \boldsymbol{v})$
    if $F < R_0^T + t \cdot \tan \alpha^T$ **then**
      $R^T \coloneqq \frac{F}{R_0^L + t \cdot \tan \alpha^L}$
      $R^C \coloneqq R^T - e, \quad R_0^C \coloneqq R^C \cdot R_0^L, \quad \tan \alpha^C \coloneqq R^C \cdot \tan \alpha^L$
      $R_0^T \coloneqq R^T \cdot R_0^L, \quad \tan \alpha^T \coloneqq R^T \cdot \tan \alpha^L$
    $t \coloneqq t + \frac{F - t \cdot \tan \alpha^C - R_0^C}{1 + |\tan \alpha^C|}$
    $i \coloneqq i + 1$
  end
  $m = \tau\left(R^T\right)$
end

Algoritmus 4: *Az egysugaras árnyékszámítás „linearizált” algoritmusa.*

4.4.2. Az algoritmus „diszkrét” változata

Az algoritmus ezen változatában a kimeneti értékhalmaz diszkrét, a lehetséges értékek közül a szomszédosak pontosan ϵ távolságra vannak egymástól. Az egyszerűsítés következtében R^T csak ezeken az előre meghatározott szinteken fog végiglépkedni. Itt tehát ahelyett, hogy R^T-t az éppen megtalált új legközelebbi felület relatív távolságára állítanánk, egyszerűen ϵ-nyival lépünk lejjebb (továbbra is a τ változását figyelembe véve), ami éppen az eddigi R^C érték, hiszen az pontosan ennyivel van mindig R^T alatt. Az algoritmus „diszkrét” változatának pszeudokódja az 5. algoritmusban olvasható.

4.4.3. Konvex optimalizáció

A sphere-trace algoritmus jelentősen felgyorsítható konvex optimalizáció használatával. (Hart [1994]) Ha a távolságfüggvényünk egy konvex objektumot ír le (vagy könnyen meghatározhatóan befoglalja egy ilyen halmaz), a sugárkövetés során ezt figyelembe véve a normál $f(\boldsymbol{s}(t))$ értéknél esetenként jóval nagyobbat is léphetünk.

Ugyanez beépíthető a cone-trace algoritmusba is, aminek felhasználásával az ár-

In : $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3, \|\boldsymbol{v}\|_2 = 1$ sugár (kezdőpont és irány),
$f : \mathbb{R}^3 \to \mathbb{R}$ távolságfüggvény,
$R_0^L, R_1^L \in [0, +\infty)$ a kúp kezdeti és végső sugara,
$d \in (0, +\infty)$ fényforrás távolsága $\boldsymbol{p}$-től,
$\epsilon = \frac{1}{256}$ az algoritmus érzékenysége,
$\tau : [-1, 1] \to [0, 1]$ megvilágítottsági függvény
Out: $m \in [0, 1]$ $\boldsymbol{p}$-be jutó fény aránya
begin
  $t \coloneqq 0, \quad m \coloneqq 1$
  $\tan \alpha^L \coloneqq \frac{R_1^L - R_0^L}{d}$
  $R^C \coloneqq \tau^{-1}(1 - \epsilon), \quad R_0^C \coloneqq R^C \cdot R_0^L, \quad \tan \alpha^C \coloneqq R^C \cdot \tan \alpha^L$
  $R^T \coloneqq 1, \quad R_0^T \coloneqq R^T \cdot R_0^L, \quad \tan \alpha^T \coloneqq R^T \cdot \tan \alpha^L$
  while $t < d$ ***and*** $m > \epsilon$ ***and*** $i < i_{max}$ **do**
    $F \coloneqq f(\boldsymbol{p} + t \cdot \boldsymbol{v})$
    if $F < R_0^T + t \cdot \tan \alpha^T$ **then**
      $R^T \coloneqq R^C$
      $m \coloneqq m - \epsilon$
      $R^C \coloneqq \tau^{-1}(m - \epsilon), \quad R_0^C \coloneqq R^C \cdot R_0^L, \quad \tan \alpha^C \coloneqq R^C \cdot \tan \alpha^L$
      $R_0^T \coloneqq R^T \cdot R_0^L, \quad \tan \alpha^T \coloneqq R^T \cdot \tan \alpha^L$
    $t \coloneqq t + \frac{F - t \cdot \tan \alpha^C - R_0^C}{1 + |\tan \alpha^C|}$
    $i \coloneqq i + 1$
  end
end

Algoritmus 5: *Az egysugaras árnyékszámítás diszkrét változata.*

nyékszámító algoritmusunk is jelentősen gyorsul. Ennek az egyik korlátja, hogy csak az objetkumokon kívül ($\{f > 0\}$) működik. A cone-trace algoritmusban az ofszetelés miatt tehát csak akkor működik az optimalizáció, ha a kúp aktuális sugara nem negatív ($R_0 + t \cdot \tan \alpha \geq 0$). Ez az árnyékszámító algoritmusban azt jelenti, hogy amíg a C kúp relatív sugara nem negatív, használhatjuk a konvex-optimalizált lépést ($R^C \geq 0$), utána pedig visszatérünk az eredeti algoritmus szerinti működésre.

A dolgozatnak nem témája a konvex optimalizáció, valamint a konvex-optimalizált cone-tracing sem, ugyanakkor felhasználásra kerültek az algoritmus ezen változatához összehasonlítás céljából. Ezek implementálva vannak a projektben, ezért kerültek felhasználásra és a 6. fejezetben összehasonlításra. Konvex optimalizáció hiányában használható Keinert *et al.* [2014] és Bálint and Valasek [2018a] algoritmusa is.

5. fejezet

Fizikai alapú árnyalás

5.1. Lokális illumináció

Fizikai alapú árnyalásnál az anyagok fényvisszaverődési tulajdonságainak leírására használt egyik gyakori eszköz a kétirányú visszaverődés-eloszlás függvény (BRDF, Pharr *et al.* [2016]). Feladatunk, hogy meghatározzuk az elsődleges sugárral megtalált $\boldsymbol{p}$ felületi pontból a kamera felé szóródó fénymennyiséget. Ez az alábbi képlettel számítható:

$$\int_{\Omega} \underbrace{f_r(\boldsymbol{v}, \boldsymbol{n}, \boldsymbol{l})}_{\text{BRDF}} \cdot \underbrace{V(\boldsymbol{p}, \boldsymbol{l})}_{\text{láthatóság}} \cdot \underbrace{E(\boldsymbol{p}, \boldsymbol{l}) \cos\theta_{\boldsymbol{l}}}_{\text{bejövő radiancia}} \mathrm{d}\boldsymbol{l}. \tag{5.1}$$

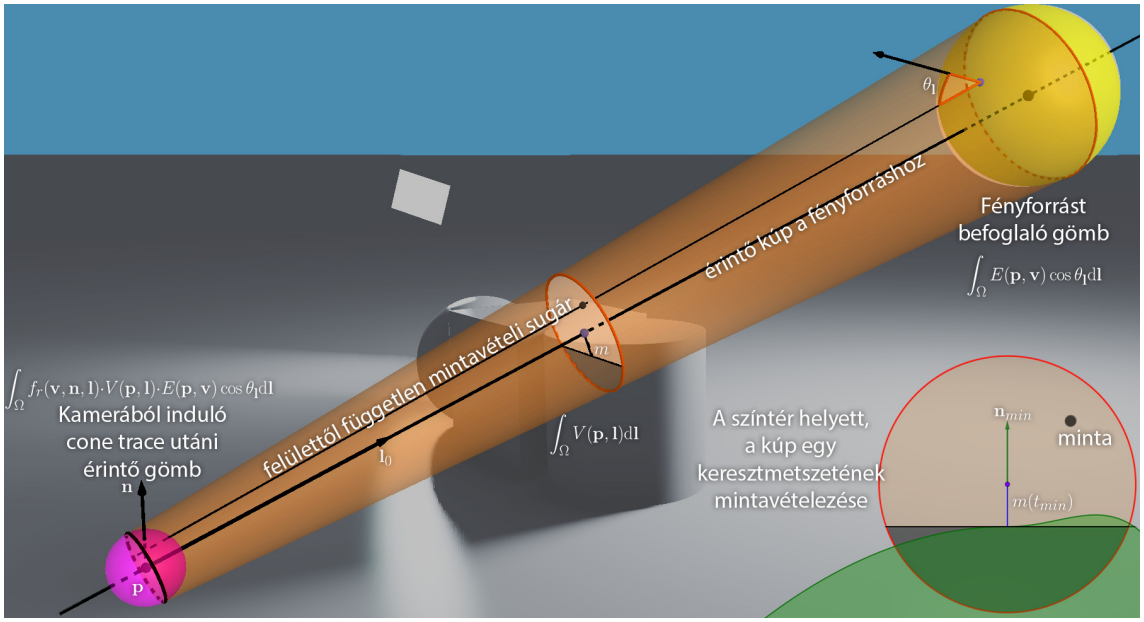
Az integrálást a felületi pont körüli $\boldsymbol{l} \in \Omega$ irányokon végezzük, ami a pont körüli egységsugarú félgömbként értelmezhető. Az $f_r(\boldsymbol{v}, \boldsymbol{n}, \boldsymbol{l})$ a kétirányú visszaverődés-eloszlás függvény, ahol $\boldsymbol{v}$ a kamera felé mutató irány, $\boldsymbol{n}$ pedig a felületi normális a pontban. A $V(\boldsymbol{p}, \boldsymbol{l}) \in \{0, 1\}$ a láthatósági függvény, értéke 1 vagy 0, aszerint, hogy $\boldsymbol{p}$ felületi pontból látszódik-e a fényforrás $\boldsymbol{l}$ irányban. Az $E(\boldsymbol{p}, \boldsymbol{l})$ a fényforrás radianciája, ez homogén fényforrásoknál konstans érték. A $\theta_{\boldsymbol{l}}$ a $-\boldsymbol{l}$ irány és a fényforrás $\boldsymbol{p}$ kezdőpontú, $\boldsymbol{l}$ irányvektorú sugárral való metszéspontjában vett normálvektor által bezárt szög.

5.2. Az integrál közelítése gömb fényforrásra

Az integrál egy durva közelítését kapjuk egy fényforrásra a következő módon:

$$I \approx f_r(\boldsymbol{v}, \boldsymbol{n}, \boldsymbol{l}_0) \cdot \int_{\Omega} V(\boldsymbol{p}, \boldsymbol{l}) \, \mathrm{d}\boldsymbol{l} \cdot \int_{\Omega} E(\boldsymbol{p}, \boldsymbol{l}) \cdot \cos\theta_l \, \mathrm{d}\boldsymbol{l}. \tag{5.2}$$

Ekkor $\boldsymbol{l}_0$ a fényforrás középpontja felé mutató vektor. A többi tagot külön-külön közelíthetjük. A korábbi fejezetekben bemutatott algoritmusok (3. algoritmus és



5.1. ábra. *A* ***p*** *felületi pontból a kamerába eljutó fénymennyiség számolása gömb fényforrásból. Módosított cone trace algoritmussal határozzuk meg a legnagyobb árnyékoló helyét. Az ábra jobb alsó részén látható módon az árnyékoló vetületét közelítve, az integrál láthatósági tagját a mintapontokban már a geometriától függetlenül határozzuk meg.*

változatai) a közelítés láthatósági tagját számolták, Ω-t leszűkítve a gömbfényforrás felé mutató irányokra. A kétirányú visszaverődés-eloszlás függvénnyel kiegészített megjelenítőalgoritmus eredményéről a 6.4. fejezetben láthatók képek.

5.3. Integrál számolása a Monte-Carlo módszerrel

Az 5.1. egyenletben lévő képlet approximálható a 3.3. fejezetben tárgyalt Monte-Carlo integrálással. A mintavételezést a 3.3.1. alfejezetben mutatott módon végezzük a felületi pont körüli érintő gömbre és a fényforrás gömbjére illesztett csonkakúpban. A más irányokból jövő fénysugarakat (pl. felületekről visszatükröződött szóródó fényt) továbbra sem vesszük figyelembe, mivel a globális illumináció számolásához rekurzív sugárkövetésre lenne szükség.

Monte-Carlo integrálás során a képletben a láthatósági tagot minden mintára meg kell határozni. Ez a korábban is tárgyalt módon, egy sugár és a színtér metszését jelenti, ami költséges műveletsor a távolságfüggvény sokszori kiértékelése miatt. Az ilyen módon számolt árnyalást használjuk összehasonlítási alapnak. A következő alfejezetben bemutatunk egy olyan módszert, amivel az integrálás során a geometriától (azaz a távolságfüggvénytől) függetlenül tudjuk meghatározni a láthatóságot.

5.3.1. Láthatósági tag közelítése

A 3. algoritmus meghatározza a csonkakúpba belógó legnagyobb árnyékoló méretét a vizsgált körvetületen. Az algoritmus átalakításával meghatározható ezen árnyékoló térbeli helye is, csak sugárkövetés közben el kell tárolni t paraméter értékét a minimumhelyen (t_{min}). Az árnyékolóról feltesszük, hogy a vizsgált helyen sík- vagy élszerű, azaz a vetületének határa a körön egy egyenes. A láthatósági tag közelítéséhez szükség van az egyenes pontos helyzetére, azaz a középponttól való távolságára és az irányára. Előbbi egyszerűen a távolságfüggvény értéke, utóbbi pedig a távolságfüggvény gradiensével határozható meg. A gradiens vetülete adja az egyenes normálvektorának irányát.

Az árnyékoló vetületét közelítő egyenes helyzetének ismeretében a Monte-Carlo integrálás során vett mintáról könnyen eldönthető, hogy metszi-e a szóban forgó árnyékolót. Ez látható az 5.1. ábra jobb alsó részén. Így tehát az integrálás elvégezhető a színtértől függetlenül.

A módszerrel készült képek a 6.4. fejezetben láthatóak. A közelítés hasonló feltételezéseket használ, mint a korábban tárgyalt egysugaras módszer, így az összehasonlításra használt teljes integrálástól való eltérése is ahhoz hasonló.

5.4. Tetszőleges alakú fényforrás

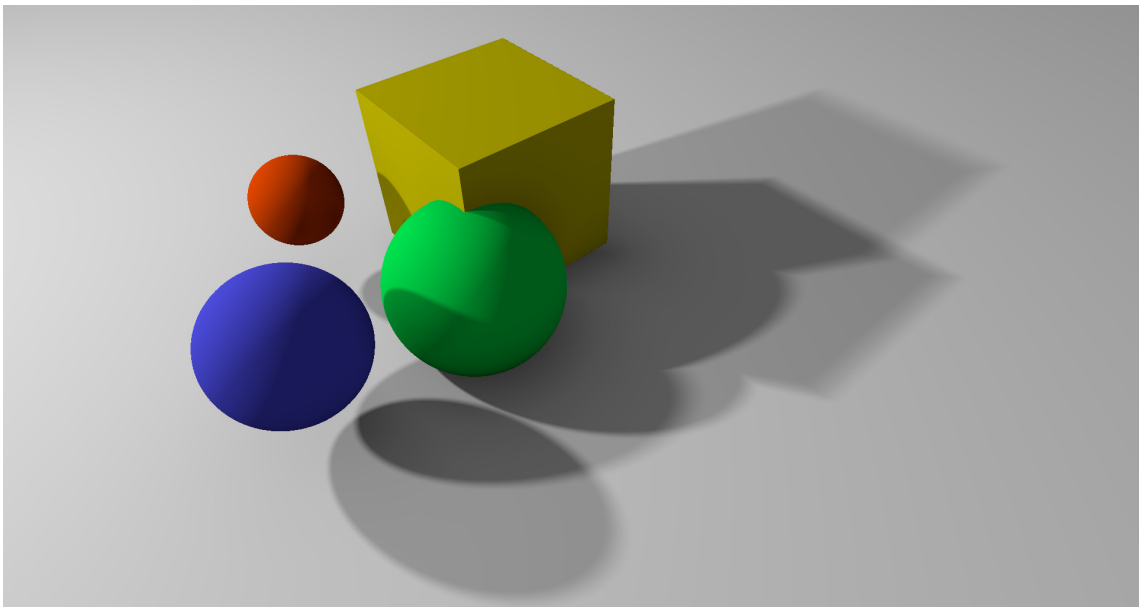
Az előző fejezetben bemutatott módszer alkalmas tetszőleges alakú fényforrásokból származó fény szimulálására is. Ehhez az szükséges, hogy a fényforrás befoglalható legyen egy gömbbe – erre illesztjük az algoritmus által használt csonkakúpot. Valamint a láthatósági tag meghatározásához el kell tudnunk dönteni a kúpban indított sugarakról, hogy eltalálnák-e a fényforrást. A módszer lehetőségeit a 6.15. és a 6.16. ábrákon téglalap alakú fényforrással szemléltetjük.

6. fejezet

Eredmények

Ebben a fejezetben bemutatjuk a született eredményeket. Megvizsgáljuk az algoritmusok paraméterektől való függését. Összehasonlítjuk a különböző algoritmusokat sebesség és minőség szempontjából. A 6.5. alfejezetben kitérek az algoritmusok azon hibáira, amik a közelítésre használt modell feltételezéseiből következnek. A 6.6. alfejezetben az algoritmus működési elvéből adódó pontatlanságok szerepelnek.

Az algoritmusok GPU-n futó shaderprogramok részeként kerültek implementálásra. A futási időket egy Nvidia 1070Ti videokártyával felszerelt asztali gépen mértük, Full HD felbontás mellett.

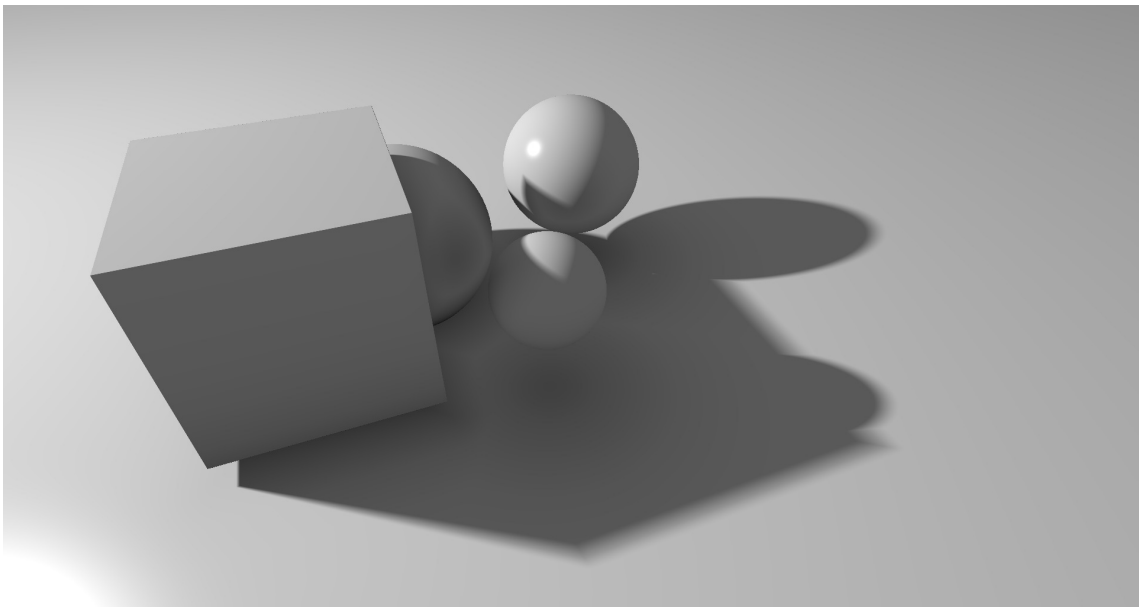


6.1. ábra. *Példa több fényforrásból származó puha árnyékra*

6.1. Összehasonlítás a referenciamódszerrel

Az egysugaras puhaárnyék-számító algoritmusainkat a 3.3. alfejezetben tárgyalt Monte-Carlo-integrálást használó módszerrel hasonlítjuk össze. Utóbbit a fejezetben MC-módszernek vagy MC-algoritmusnak rövidítjük. A referenciaalgoritmus módszere általánosan nem használható valósidejű árnyékszámolásra, azonban pontos eredményt ad, míg a mi algoritmusaink célja a valósidejű renderelés, esetlegesen a minőség kárára.

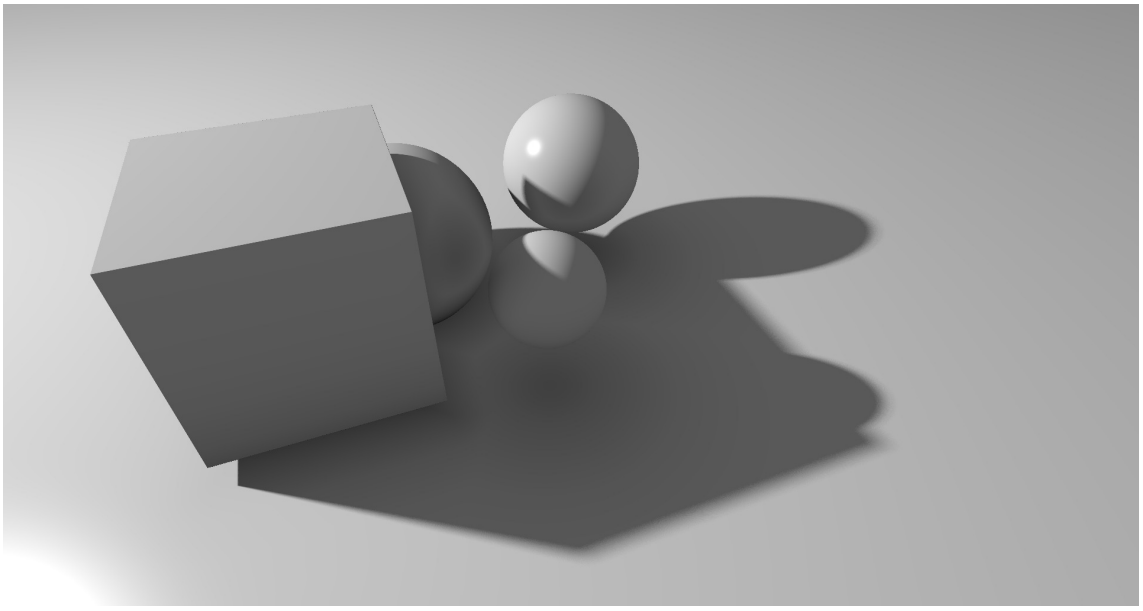
Példaként a 6.2. ábrán az egysugaras algoritmus eredménye látható, ami 1.72 ezredmásodperc alatt készült el, míg az azonos helyről a Monte-Carlo-módszerrel készült kép a 6.3. ábrán látható, amihez 23.46 ms-ra volt szüksége az algoritmusnak, ezalatt minden pixelhez 64 árnyéksugarat használt. Összehasonlításképpen az MC-módszer az adott képhez 1.77 ms alatt három sugarat tudott lőni, ennek eredménye a 6.4. ábrán látható. A képekből általában elmondható, hogy az egysugaras algoritmus szinte azonos eredményt ad a referenciamódszerhez, miközben a rendereléhez felhasznált ideje a másik töredéke.



6.2. ábra. *Példa az egysugaras algoritmus eredményére. Az algoritmus $\epsilon = \frac{1}{10}$ pontossággal számolt 1.72 ms alatt. A fényforrás sugara 0.2 egység. (A kocka oldalhossza 2 egység.)*

Az eme fejezetben felsorakoztatott 6.2., 6.3. és 6.4. ábrák kis sugarú gömb fényforrással készültek, a gömb átmérője 0.2 egység volt. Összehasonlításként, a kocka oldalhossza két egység és a kocka teteje a fényforrás középpontjától körülbelül 9 egységnyire van.

A következő példában a fényforrás sugara egy egység. Ekkor az MC-módszernek jóval több időre van szüksége ahhoz, hogy az eredmény ne legyen szemcsés, míg az



6.3. ábra. *Referenciapélda az MC-módszerrel. Az algoritmus 64 sugarat használt pixelentként és 23.46 ms-ig futott. A fényforrás sugara 0.2 egység. (A kocka oldalhossza 2 egység.)*

egysugaras algoritmus alig fut tovább. Az egysugaras algoritmus eredménye a 6.5. ábrán látható, míg a MC-módszeré a 6.6. ábrán. A referenciaalgoritmus 191.5 ms alatt számolta ki a képet és pixelenként 512 sugarat használt. Az egysugaras algoritmus 2.21 ms-ig futott. Ennél a példánál már nagyobb a vizuális különbség, de a futásidőben is nagy a változás. A MC-módszernek a nagyobb területre szétterülő árnyékok esetén jóval több mintát kell vennie a megfelelő eredményhez, míg az egysugaras algoritmusnak szinte alig változik a futásideje, képminősége.

6.2. Az algoritmusvariációk sebességösszehasonlítása

Az implementált egysugaras algoritmusvariációk futásidejét a 6.1., a 6.2. és a 6.3. táblázatban hasonlítottuk össze. A mérésekre az előző sorrendben a 6.7., a 6.8. és a 6.9. ábrákon beállított jelenteket használtuk. Az értékek egyetlen képkocka teljes elkészítési idejének átlagos értékét mutatják ezredmásodpercben kifejezve. A zárójelben az adott méréshez használt adatok szórása áll. Az első oszlopban a használt ϵ – az algoritmus érzékenységi paramétere – áll. A D jelű oszlopban a sugárkövetés iránya található (A: a felülettől a fény felé, B: fordítva). Továbbá minden sorban kiemelésre került az adott ϵ-hoz a legkisebb futási idő.

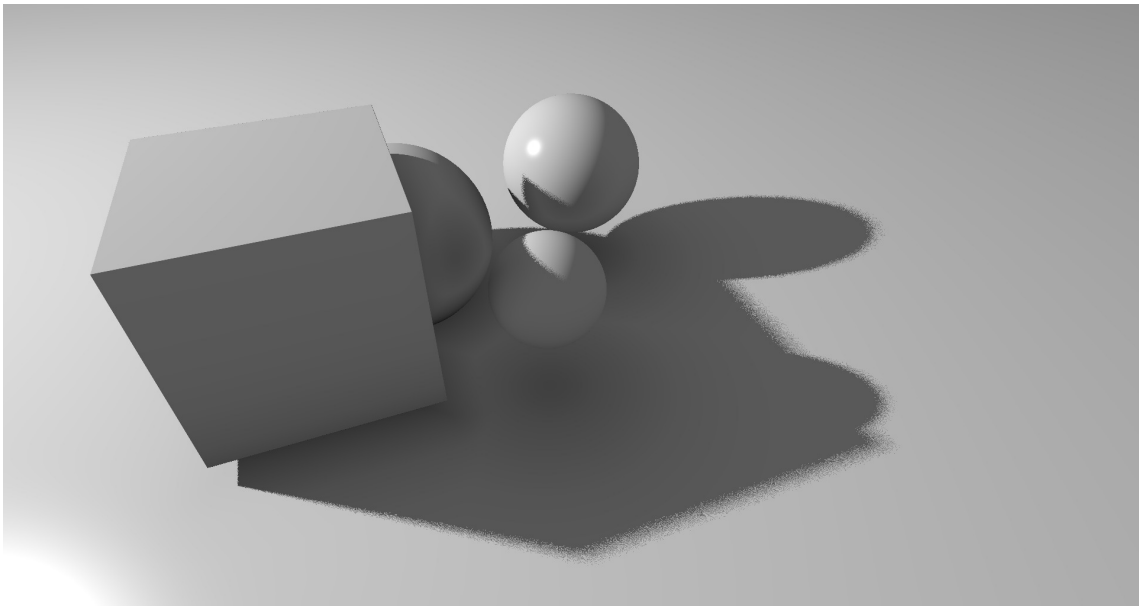
A táblázatokból látható, hogy a linearizált és a konvex-optimalizált algoritmusvariációk a leggyorsabbak. A konvex-optimalizált algoritmus akkor ad jobb eredményt, ha a kép nagy részén teljesen megvilágított felületek jelennek meg, vagy ha

ϵ	D	Alap	Lineáris	Diszkrét	Konvex
$\frac{1}{10}$	A	3.2398 (0.0570)	1.4602 (0.0110)	2.1093 (0.0227)	1.1499 (0.0103)
	B	5.0921 (0.0898)	1.1903 (0.0109)	1.7534 (0.0255)	**1.0872** (0.0087)
$\frac{1}{20}$	A	4.9836 (0.0986)	1.8421 (0.0201)	3.1535 (0.0824)	1.5558 (0.0169)
	B	5.5431 (0.1115)	1.5759 (0.0270)	2.7334 (0.0367)	**1.4943** (0.0209)
$\frac{1}{30}$	A	5.8127 (0.1001)	2.2051 (0.0354)	4.1393 (0.0889)	1.9534 (0.0286)
	B	5.7696 (0.1284)	1.9465 (0.0354)	3.7289 (0.0746)	**1.8854** (0.0215)
$\frac{1}{256}$	A	34.153 (0.7766)	**9.2220** (0.2159)	25.495 (0.6481)	10.233 (0.1947)
	B	37.060 (0.8770)	9.6777 (0.2117)	25.546 (0.6911)	10.547 (0.1927)

6.1. táblázat. *Az algoritmusok futásideje az egyszerű jelenetben. A jelenet vetett árnyék nélküli elkészítésének átlagos ideje 0.7005 ms volt, 0.0009 ms szórással. Az oszlopokban az egysugaras puhaárnyék-számító algoritmus négy változatának futásideje és szórása (ezredmásodperc) áll különböző pontossági paraméterek mellett (ϵ). A sorokban felváltva szerepelnek a felületi pontból a fényforrás felé (A) és fordítva (B) futtatva mért idők.*

ϵ	D	Alap	Lineáris	Diszkrét	Konvex
$\frac{1}{10}$	A	29.429 (0.1244)	27.769 (0.1176)	32.197 (0.1349)	32.188 (0.1697)
	B	26.889 (0.1149)	**25.226** (0.1164)	31.966 (0.1375)	31.870 (0.1849)
$\frac{1}{20}$	A	33.607 (0.1626)	30.304 (0.1185)	40.233 (0.2243)	40.203 (0.2284)
	B	31.387 (0.1161)	**27.717** (0.1173)	40.290 (0.2082)	40.309 (0.2179)
$\frac{1}{30}$	A	38.011 (0.2381)	32.760 (0.1155)	47.382 (0.2497)	47.364 (0.2469)
	B	36.018 (0.1578)	**30.240** (0.1223)	48.028 (0.2074)	47.961 (0.2437)
$\frac{1}{256}$	A	136.24 (1.6938)	**90.70** (1.3146)	171.89 (1.3370)	171.47 (1.6120)
	B	143.22 (2.3168)	92.02 (1.0914)	182.56 (1.7219)	182.48 (1.6443)

6.2. táblázat. *Az algoritmusok futásideje az összetett jelenetben. A jelenet vetett árnyék nélküli elkészítésének átlagos ideje 18.686 ms volt, 0.1293 ms szórással. Az oszlopokban az egysugaras puhaárnyék-számító algoritmus négy változatának futásideje és szórása (ezredmásodperc) áll különböző pontossági paraméterek mellett (ϵ). A sorokban felváltva szerepelnek a felületi pontból a fényforrás felé (A) és fordítva (B) futtatva mért idők.*



6.4. ábra. *Példa a MC-módszer eredményére. A 6.2. ábrán látható egysugaras program futási ideje alatt az MC-módszer három sugárral tud számolni, így eredménye szemcsés lesz.*

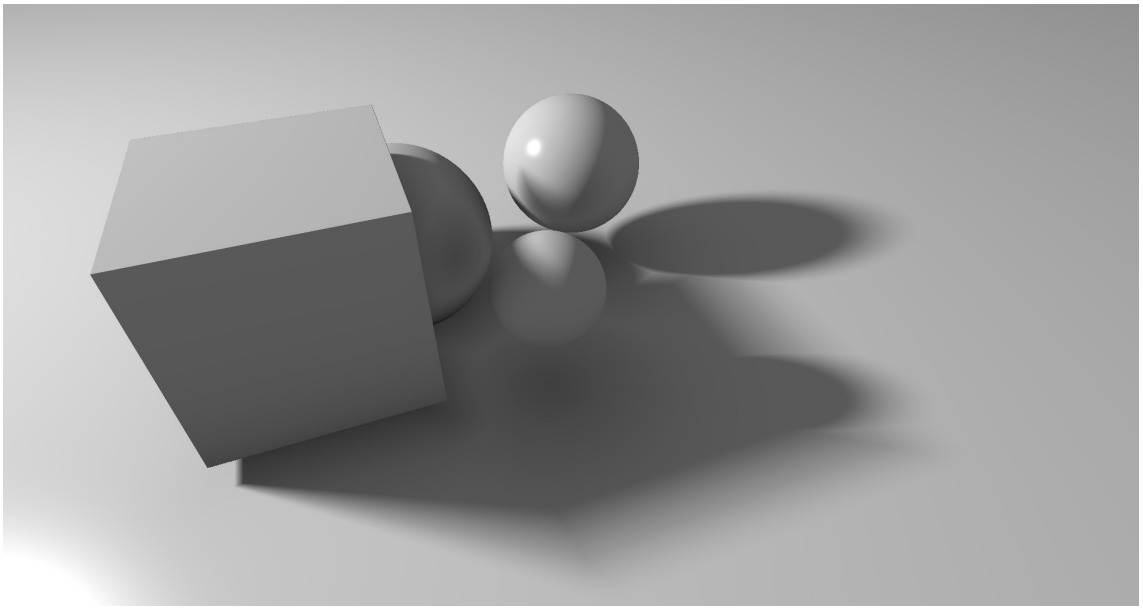
a fénynek nagy távolságot kell megtennie, például ha távolra nézünk és a horizont megjelenik a képen. Ilyen esetben a konvex-optimalizált algoritmus akár háromszor gyorsabb a linearizáltnál. Például a 6.7. ábrán látható jelenetben a kamerát a horizontra fordítva (úgy, hogy a horizont a megjelenített kép teteje alatt van néhány pixellel), $\epsilon = \frac{1}{30}$ esetén a linearizált algoritmusnak 4.55 ms-ra van szüksége, míg a konvex-optimalizációt használó algoritmusnak csak 1.53 ms-ra. A három táblázatban lévő tesztnél ez nem látható, mivel a kamera közvetlenül a közeli objektumokra van fordítva és a fényforrás is viszonylag közel található.

Megjegyzendő továbbá, hogy a mért ϵ-ok közül valódi használatra csak az első három sorban találhatók alkalmazandók. Az $\frac{1}{30}$ alatti ϵ érékek esetén a létrehozott képen nincs vizuális javulás.

6.3. Az algoritmus érzékenysége

Az algoritmusok érzékenységi paramétere (ϵ) az az érték, amilyen mértékű fényarány-különbséget az alkalmazott modell szerint az algoritmus mindenképpen meg tud különböztetni. Ennek megfelelően az $\epsilon = \frac{1}{256}$ érték felel meg a 24-bites színmélység – true color – szürkeárnylatainak. Az algoritmussal végzett tesztek azonban megmutatták, hogy az implementált algoritmus a tőle elvárt pontosságot legalább egy nagyságrenddel túlteljesíti.

Az algoritmus futási sebessége függ ϵ-tól. Minél nagyobb pontosságot várunk el, annál lassabb az algoritmus. Az algoritmus futási sebességének változása meg-



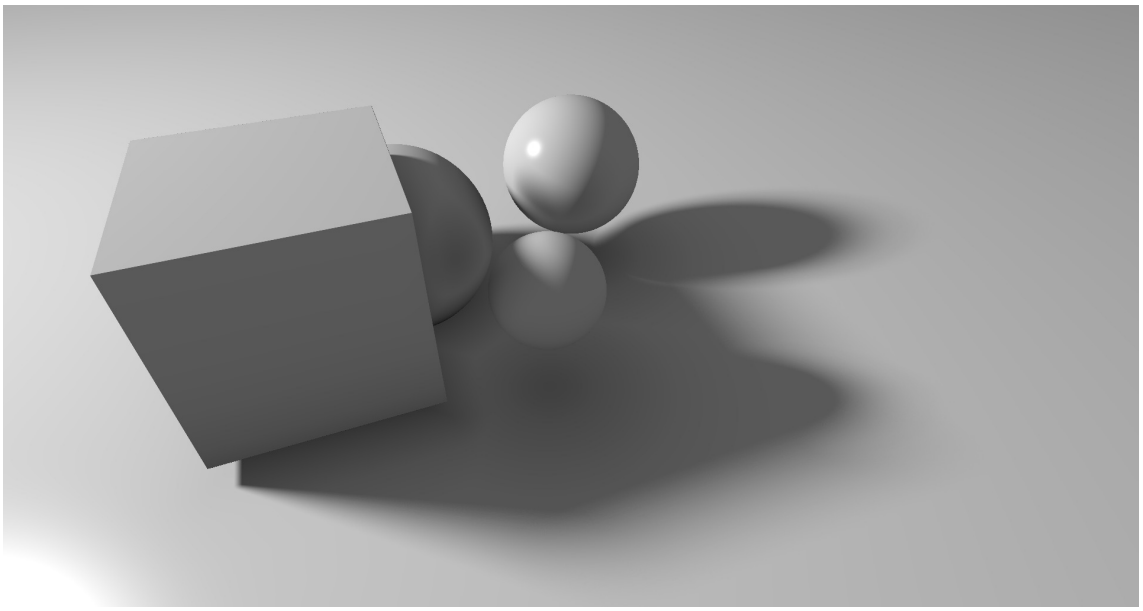
6.5. ábra. *Példa az egysugaras algoritmus eredményére. Az algoritmus $\epsilon = \frac{1}{20}$ pontossággal számolt 2.21 ms alatt. A fényforrás sugara 1 egység. (A kocka oldalhossza 2 egység.)*

figyelhető a 6.1. és a többi hasonló táblázaton. Ezen kívül a 6.10. ábán is látható a különböző ϵ-okhoz tartozó futási idő, az ábrán látható mérési eredményeket a következő bekezdés tárgyalja.

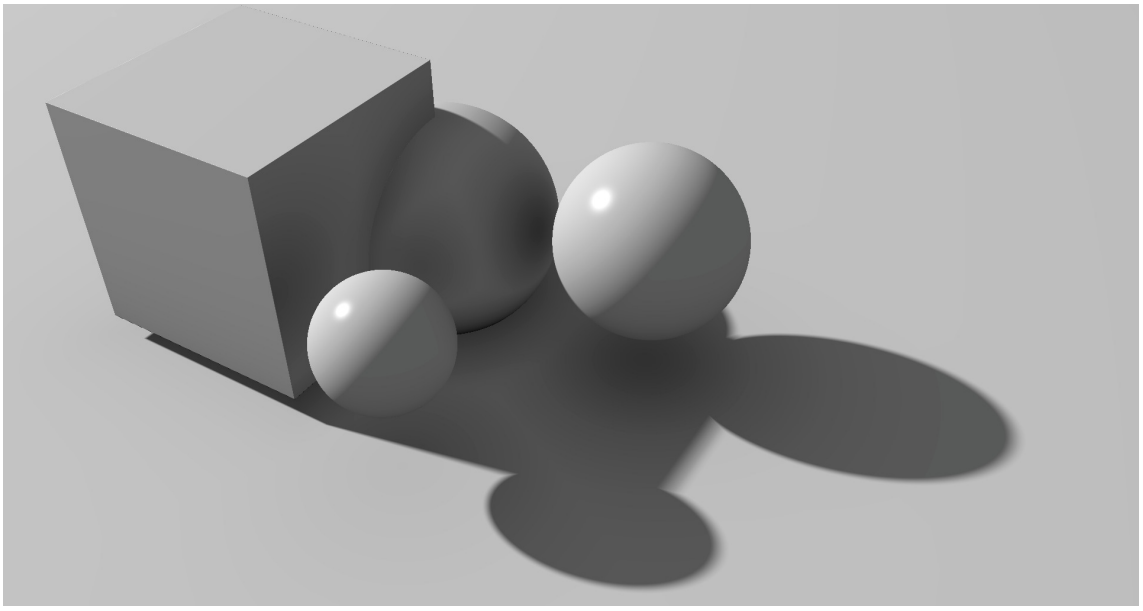
A helyes érzékenységi érték – a maximális ϵ, ami alatt nincs látható javulás – függ a fényforrás méretétől is. Ennek tesztelésére a 6.8. ábrán is látható színteret használtuk egy másik perspektívából. A beállítás a 6.11. ábrán látható. A tesztben a linearizált algoritmust futtattuk különböző méretű fényforrásokra. A fényforrás középpontja a kép bal oldalán lévő henger tetejétől körülbelül 6 egységnyire található, a henger sugara 1 egység. A 6.10. ábrán látható a mérések összesítése.

A használt gömbfényforrás sugarai rendre: 0.25, 0.5, 1, 2 és 4 egység. A különböző fényforrásméretekhez a legnagyobb helyes ϵ-értékek (a mértek közül) sorban: $\frac{1}{16}$, $\frac{1}{16}$, $\frac{1}{32}$, $\frac{1}{32}$ és $\frac{1}{64}$. A helyes ϵ-hoz tartozó futásidők: 34.47, 34.48, 39.42, 40.78 és 54.19 ezredmásodperc. Az alacsony ϵ-értékek oka a közeli és viszonylag nagy fényforrás.

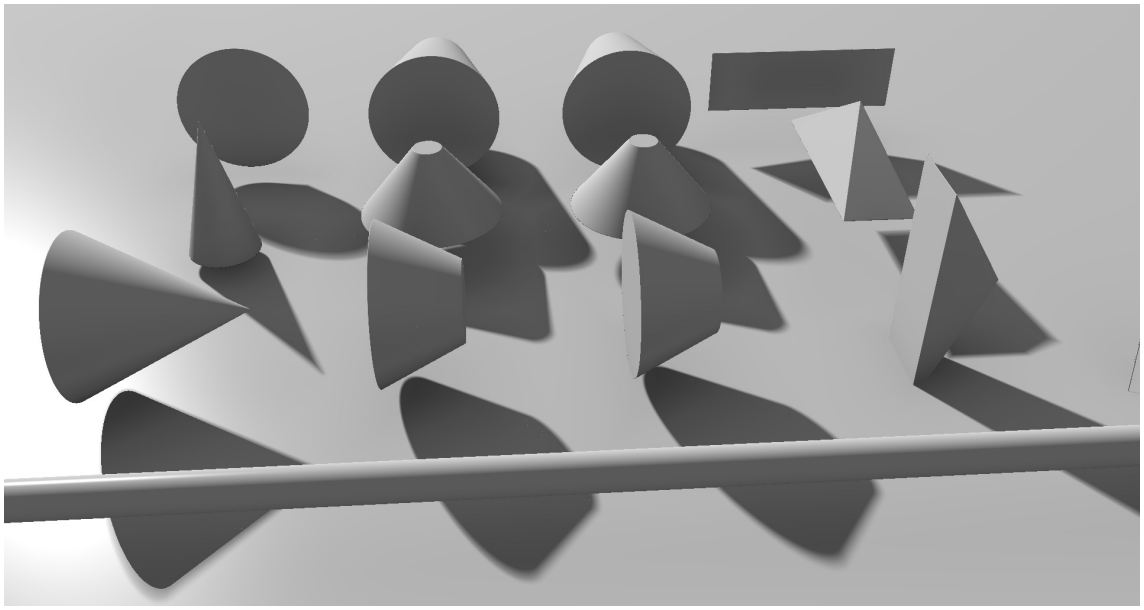
Példaként a legkisebb fényforrással helyesen beállított érzékenységgel a 6.11. ábrán látható képet kaptuk. Ugyanezen fényforrással túl nagy ϵ-ra ($\epsilon = \frac{1}{2}$) példa a 6.12. ábra. A legnagyobb fényforrással ($r = 4$) készült kép helyes érzékenységi paraméterrel a 6.13. ábrán látható, míg $\frac{1}{2}$-es érzékenységgel a 6.14. ábrán.



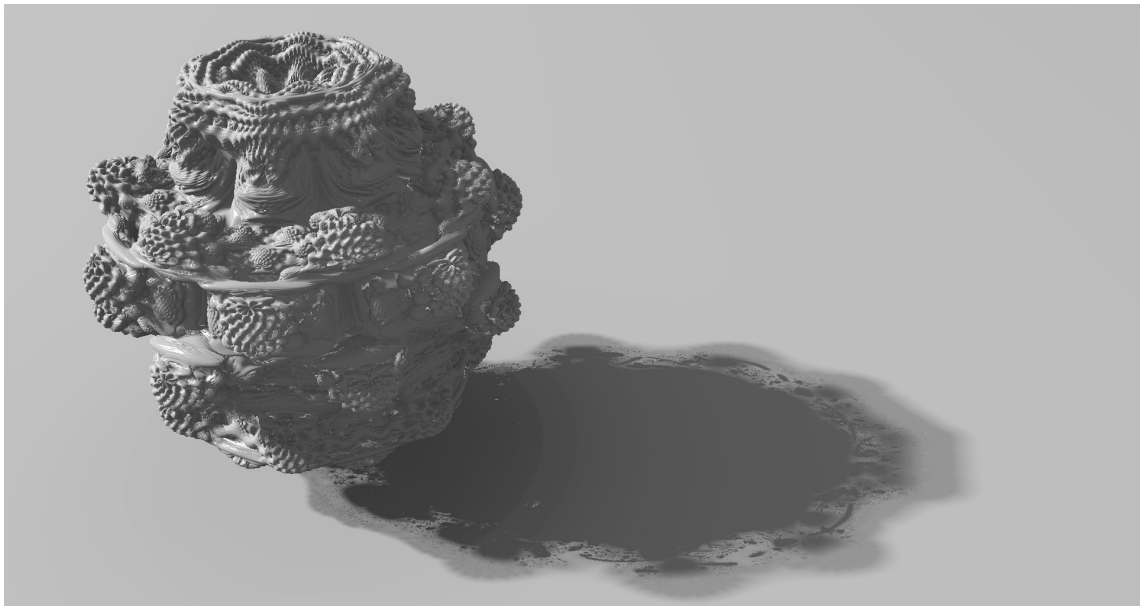
6.6. ábra. *Példa a MC-módszer eredményére. Az algoritmus 512 sugarat használt pixelentként és 191.5 ms-ig futott. A fényforrás sugara 1 egység. (A kocka oldalhossza 2 egység.)*



6.7. ábra. *Az első tesztjelenet. A távolságfüggvény három gömb, egy téglatest és egy sík uniójából áll.*



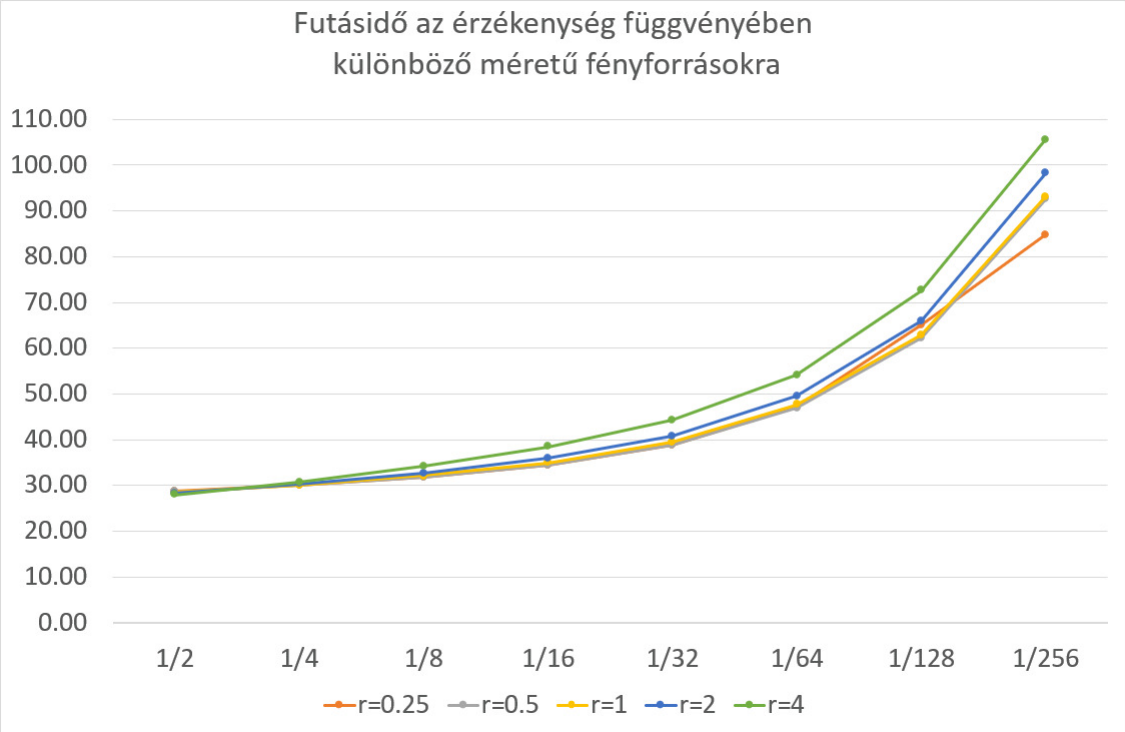
6.8. ábra. *Az második tesztjelenet. A távolságfüggvény 26 különböző méretű és helyzetű objektumból áll. A képen láthatókon kívül is vannak objektumok, ezek szintén növelik a futásidőt, hiszen a távolságfüggvény számolása globális.*



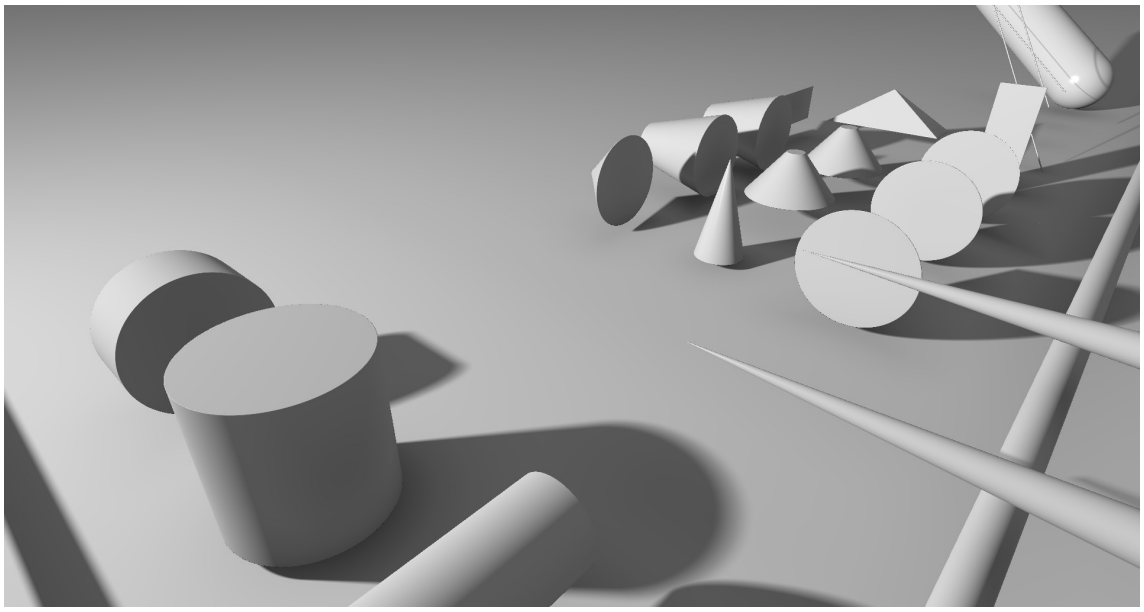
6.9. ábra. *Az harmadik tesztjelenet. A távolságfüggvény a Mandelbulb (Íñigo Quílez [2009]) fraktált tartalmazza.*

ϵ	D	Alap	Lineáris	Diszkrét	Konvex
$\frac{1}{10}$	A	17.149 (0.3686)	11.799 (0.2958)	12.607 (0.2327)	13.413 (0.2833)
	B	13.960 (0.1918)	**9.635** (0.1357)	10.499 (0.2459)	10.987 (0.1624)
$\frac{1}{20}$	A	27.338 (0.5332)	18.266 (0.2982)	20.728 (0.4067)	20.823 (0.4304)
	B	22.382 (0.4129)	**14.726** (0.2060)	17.130 (0.2814)	16.732 (0.2842)
$\frac{1}{30}$	A	36.391 (0.5878)	24.096 (0.4171)	27.884 (0.5314)	27.422 (0.5494)
	B	29.515 (0.4113)	**19.230** (0.4028)	23.201 (0.4057)	21.806 (0.3295)
$\frac{1}{256}$	A	184.96 (3.2495)	128.54 (2.2847)	166.26 (2.8884)	144.63 (3.2016)
	B	150.23 (2.1726)	**99.712** (1.5695)	132.23 (2.1659)	111.80 (1.6595)

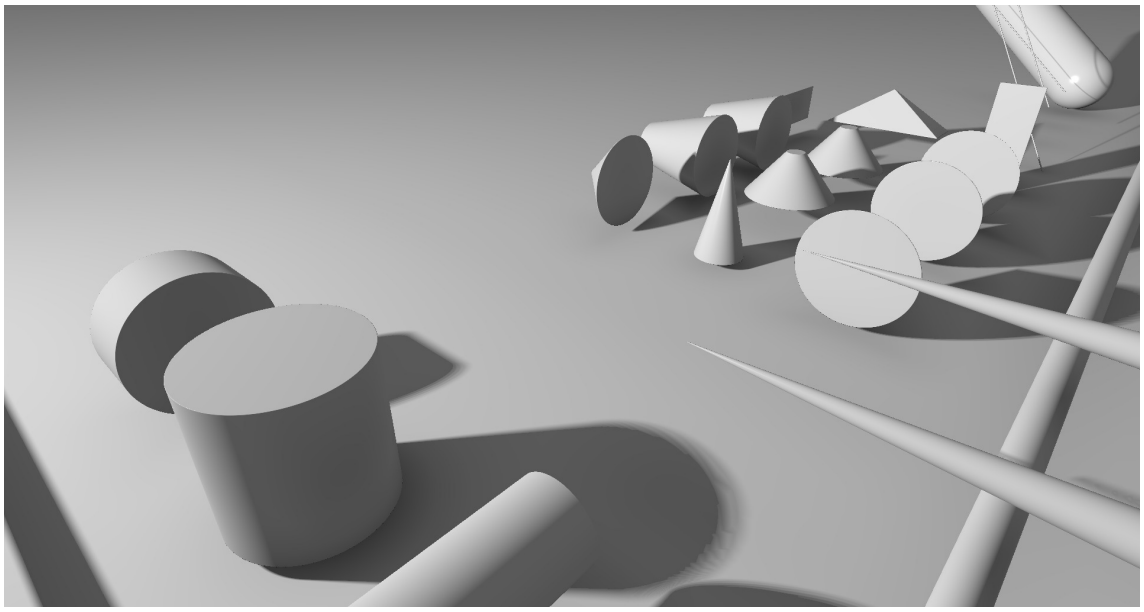
6.3. táblázat. *Az algoritmusok futásideje a Mandelbulb (Íñigo Quílez [2009]) jelenetben. A jelenet vetett árnyék nélküli elkészítésének átlagos ideje 2.4964 ms volt, 0.0465 ms szórással. Az oszlopokban az egysugaras puhaárnyék-számító algoritmus négy változatának futásideje és szórása (ezredmásodperc) áll különböző pontossági paraméterek mellett (ϵ). A sorokban felváltva szerepelnek a felületi pontból a fényforrás felé (A) és fordítva (B) futtatva mért idők.*



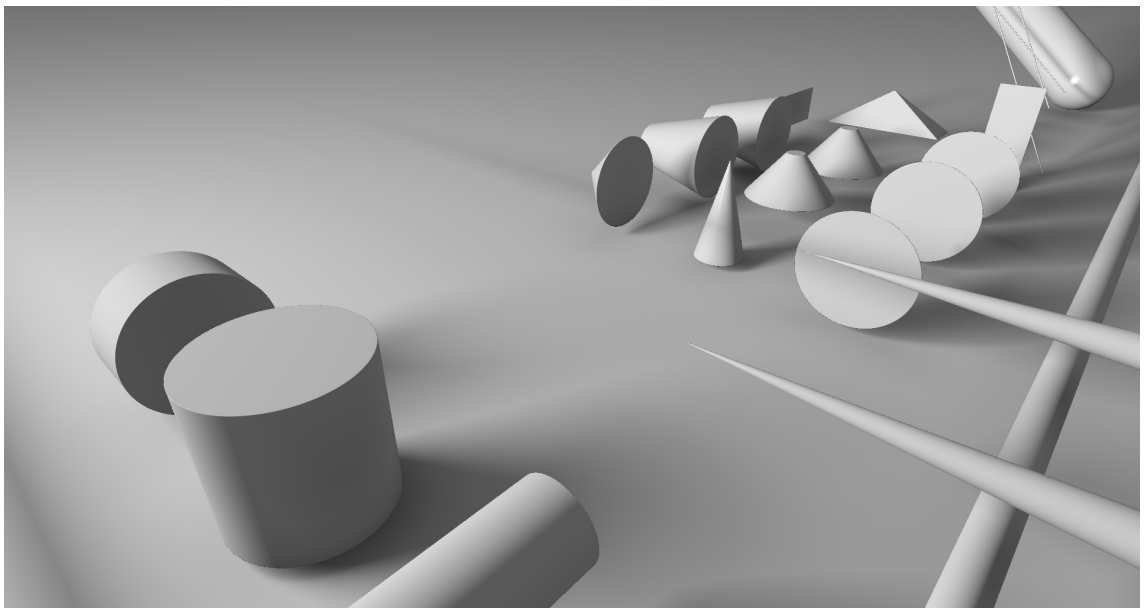
6.10. ábra. *Futásidők az érzékenység és a fényforrás méretének függvényében. A függőleges tengelyen a futási idő áll ezredmásodpercben mérve. Az öt adatsor a különböző fényforrásméretekhez tartozik: r a gömb sugara. Vízszintesen ϵ értékei állnak, csökkenő sorrendben.*



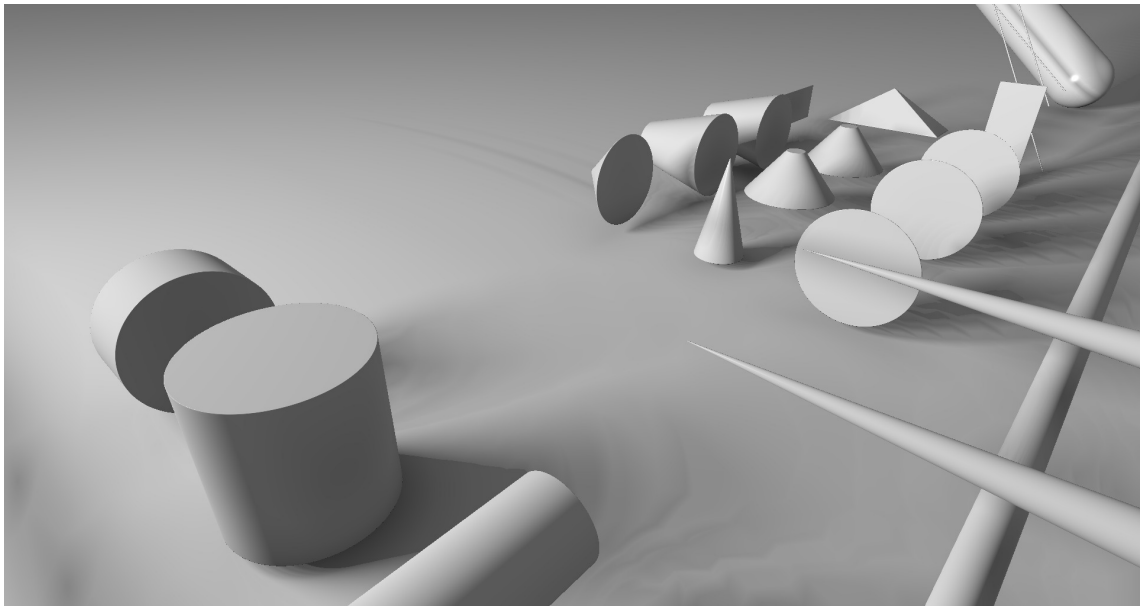
6.11. ábra. *Az összetett jelenet egy másik nézetből. A fényforrás sugara 0.25 egység,* $\epsilon = \frac{1}{16}$*. (A henger sugara 1 egység.)*



6.12. ábra. *A fényforrás sugara 0.25 egység,* $\epsilon = \frac{1}{2}$*. (A henger sugara 1 egység.) A paraméter túl nagy, az árnyék átmenetében sávok jelennek meg.*



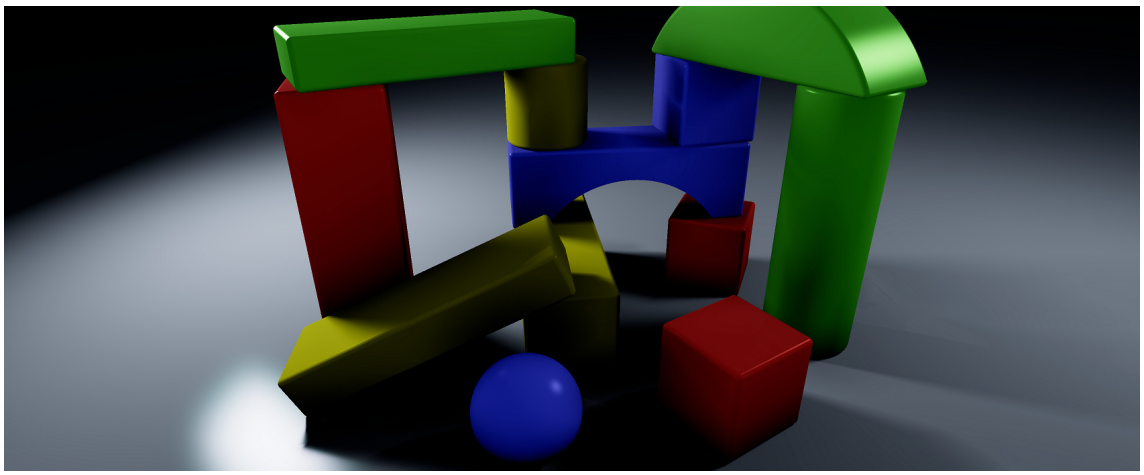
6.13. ábra. *A fényforrás sugara 4 egység, $\epsilon = \frac{1}{64}$. (A henger sugara 1 egység.)*



6.14. ábra. *A fényforrás sugara 4 egység, $\epsilon = \frac{1}{2}$. (A henger sugara 1 egység.) A paraméter túl nagy, az árnyék átmenetében sávok jelennek meg.*

6.4. Fizikai alapú árnyalás

Az 5. fejezetben leírt BRDF alapú fizikai árnyalást megvalósító program egy eredményképe látható a 6.15. ábrán. A képen látható geometriát a kép bal oldalán hátulról egy téglalap alakú fényforrás és jobb oldalról egy gömbfényforrás világítja meg. A kép az 5.3.1. és az 5.4. fejezetekben leírt módszerrel készült.



6.15. ábra. *Fizikai alapú árnyalás egy bonyolultabb színtéren. A bal oldalon hátulról egy téglalap alakú fényforrás – ennek látszódik a spekuláris csillanása az alapsíkon – valamint jobb oldalról egy gömbfényforrás világítja meg az objektumokat.*

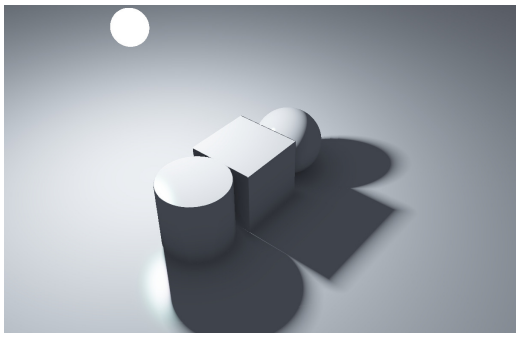
A 6.16. ábrán egy egyszerű színtér – egy gömb, egy kocka és egy henger – látható különböző módszerekkel renderelve. Az első három az 5.2. egyenlet szerinti közelítést használja, míg a többi közvetlenül az 5.1. integrált számolja Monte-Carlo módszerrel.

A 6.16a. kép Íñigo Quílez [2010] és Aaltonen [2018] algoritmusát használja az egyszerűsített integrál láthatósági tagjának meghatározására. A módszer egyszerű sphere tracinget végez a gömb középpontja felé miközben a sugárhoz legközelebbi árnyékolót keresi. Így az átmeneti rész belső felét teljes árnyéknak tekinti, mivel a sugár nem tud átjutni a felületeken.

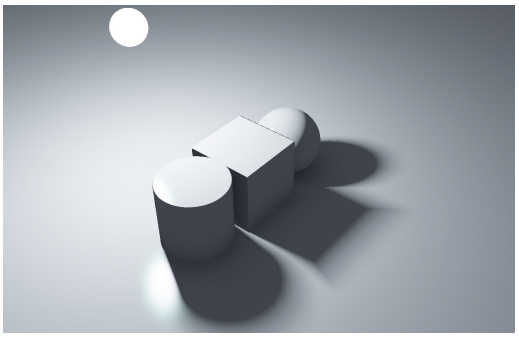
A 6.16b. képen a dolgozatban tárgyalt egysugaras algoritmus eredménye látható. A futási ideje alig haladja meg Íñigo Quílez [2010] és Aaltonen [2018] módszerét, ugyanakkor sokkal pontosabb árnyékot eredményez.

A 6.16c. képhez az integrál láthatósági tagja Monte-Carlo integrálással került meghatározásra. Ezt az eredményt közelíti az előző kettő, összehasonlítva jól látszanak a különbségek. A 6.16a. képen az árnyék túl nagy, míg az egysugaras módszer eredménye sokkal hasonlóbb, ugyanakkor az objektumok között több fény szűrődik át.

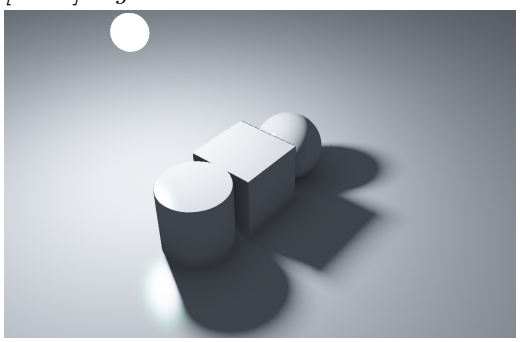
A 6.16d. képen az 5.3.1. alfejezetben leírt módon számoljuk az eredeti integrált, csak a láthatósági tagot közelítjük a geometria nélkül. Ez hasonlítandó össze a 6.16e. képpel, amin a láthatósági tag pontos, a színtér metszésével van meghatározva. Az



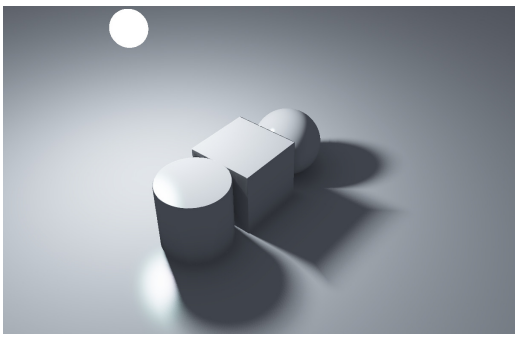
(a) *Íñigo Quílez [2010] és Aaltonen [2018] algoritmusa:* ***1.17 ms***



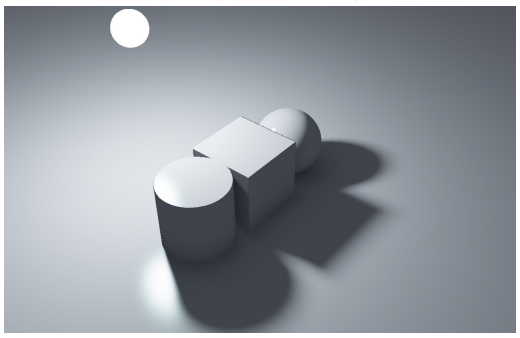
(b) *Egysugaras algoritmus:* ***1.49 ms***



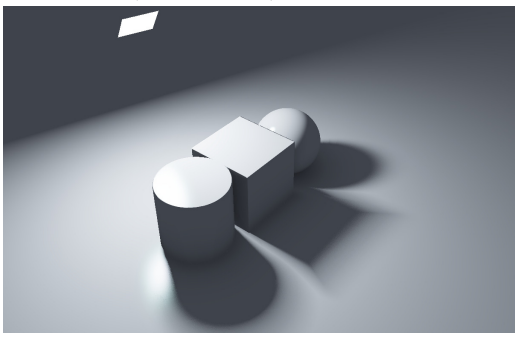
(c) *Monte-Carlo módszer a láthatósági tagra:* ***49.7 ms*** *(150 minta)*



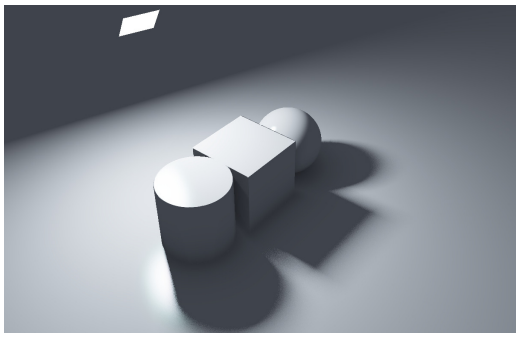
(d) *Teljes MC, a láthatósági tag becsült:* ***41.3 ms*** *(150 minta)*



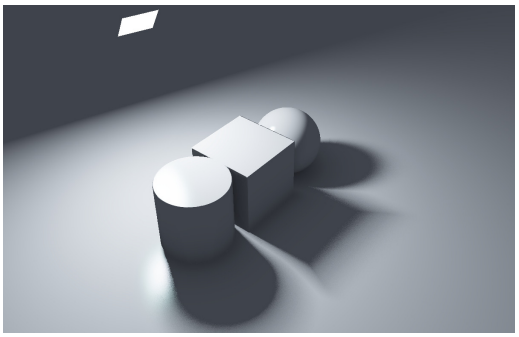
(e) *Teljes MC:* ***74.7 ms*** *(150 minta)*



(f) *Teljes MC, a láthatósági tag becsült:* ***33.6 ms*** *(150 minta)*



(g) *Teljes MC:* ***27.5 ms*** *(50 minta)*



(h) *Teljes MC, a láthatósági tag becsült:* ***12.6 ms*** *(50 minta)*

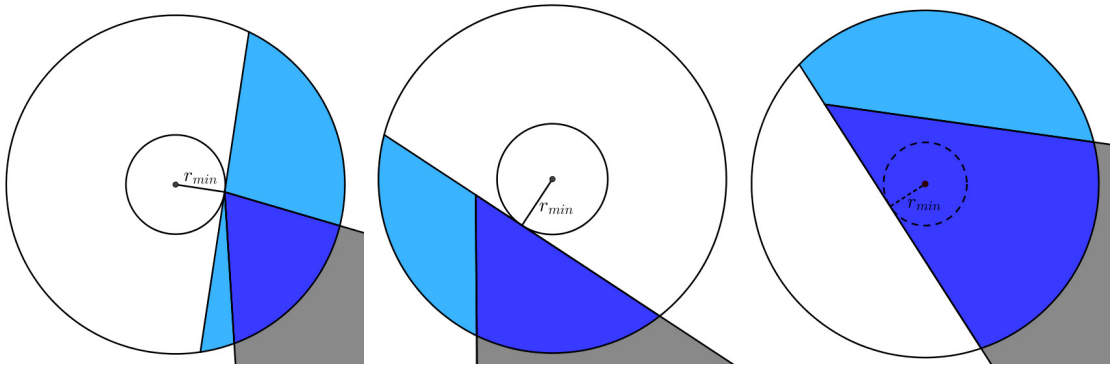
6.16. ábra. *Különböző fizikai alapú renderelési algoritmusok összehasonlítása egy egyszerű jeleneten téglalap- és gömbfényforrásra. Az első három módszer az egyszerűsített 5.2. képlettel számol, míg a többi a teljes 5.1. integrálra alkalmazza a Monte-Carlo módszert.*

egysugaras módszernél tapasztalható fényátszűrődés itt is megmarad (hiszen a láthatósági tagot egyetlen cone trace alapján közelítjük az összes mintához), azonban a sebességbeli különbség itt majdnem kétszeres azonos mintaszám mellett.

Az utolsó három képen az utóbbi két algoritmus (5.3. és 5.3.1. fejezet) eredménye látható egy téglalap alakú fényforrással. A 6.16f. és a 6.16h. képeken a láthatósági tag becsült, a mintaszám 150 és 50; a 6.16g. kép a pontos integrállal számol, 50 mintával. A két alsó kép mintaszáma még nem elégséges, az integrál nem konvergált, a kép szemcsés a penumbrában. Az azonos mintaszám melletti futásidők különbsége itt is szembetűnő, ami a színtértől való függetlenségnek köszönhető.

6.5. A modell hibái

Az algoritmus használ néhány feltételezést, és ahol ezek nem teljesülnek, ott eltér a kapott eredmény a pontos integrál értéktől. Ezek jellemző okai a következő bekezdésekben olvashatók.

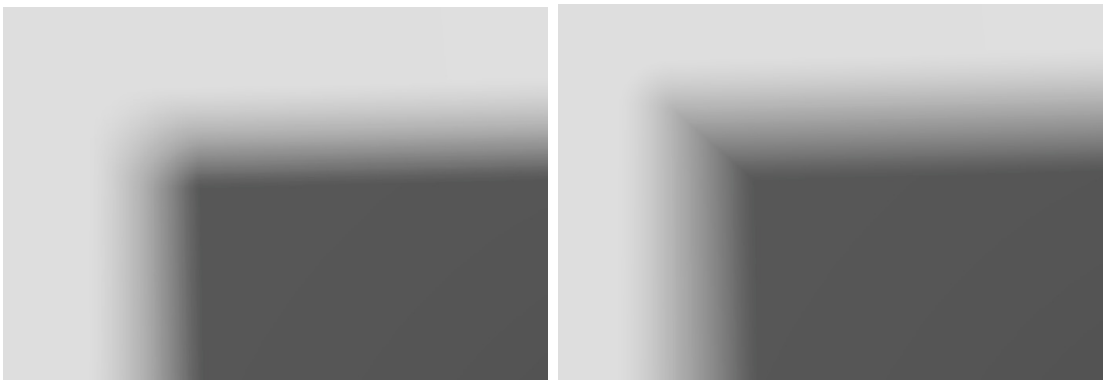


(a) $r_{min} > 0$ **(b)** $r_{min} > 0$ **(c)** $r_{min} < 0$

6.17. ábra. *Hegyes objektumok árnyéka. A világoskék részt az algoritmus beleszámolja az árnyékba, pedig át tudna jutni a fény.*

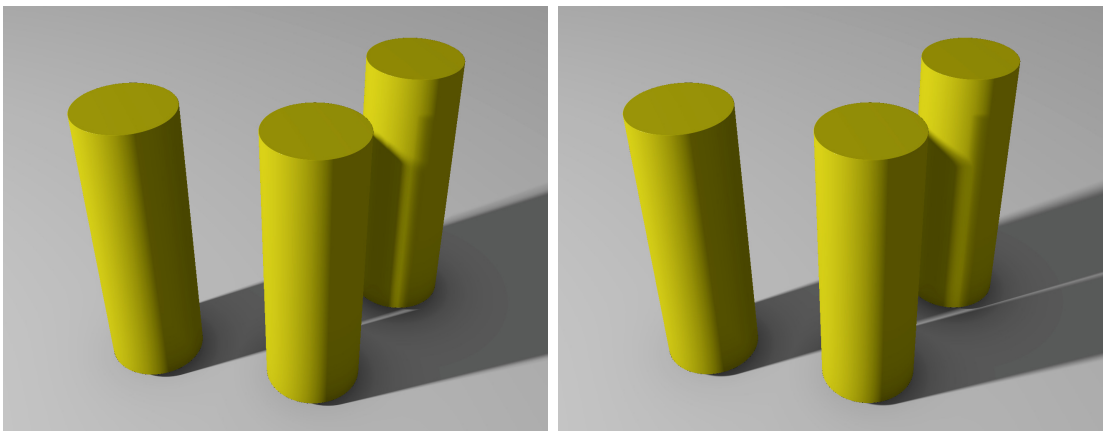
Hegyes objektumok Hegyes objektumok a szükségesnél sötétebb árnyékot vetnek, ugyanis nagyobb kúpmetszetről feltételezzük az algoritmus során, hogy a felület belsejéhez tartozik. A 6.17. ábra példázza az approximált és a tényleges vetület különbségét, a 6.18 ábra pedig egy kocka csúcsának az árnyékán hasonlítja össze a pontos és az approximált módszer eredményét.

Több árnyékoló Ha a kúpba több különböző tárgy is belelóg különböző irányokból, de önmagában egyik sem fedi el teljesen azt, akkor az algoritmus csak az önmagában legnagyobb árnyékot okozó objektum árnyékát veszi figyelembe. Ahogy az a 6.19. ábrán látható, ilyenkor a modell túlbecsüli a fény mennyiségét, azaz „átszivárog” a fény.



(a) *Pontos megoldás – Monte-Carlo* **(b)** *Egysugaras algoritmus*

6.18. ábra. *Hegyes objektum árnyékának összehasonlítása*



(a) *Referencia MC-módszerrel* **(b)** *Egysugaras algoritmus eredménye*

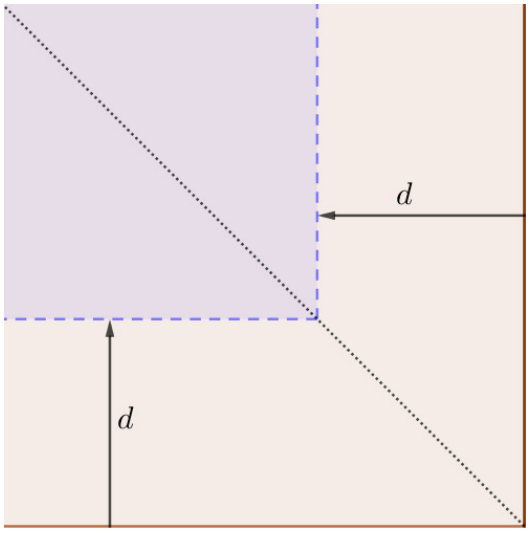
6.19. ábra. *Több árnyékoló éppen teljesen árnyékolhatja a fénysugarat, viszont az egysugaras algoritmus itt hibázhat és túlbecsülheti a fény mennyiségét.*

6.6. Az algoritmus hibái

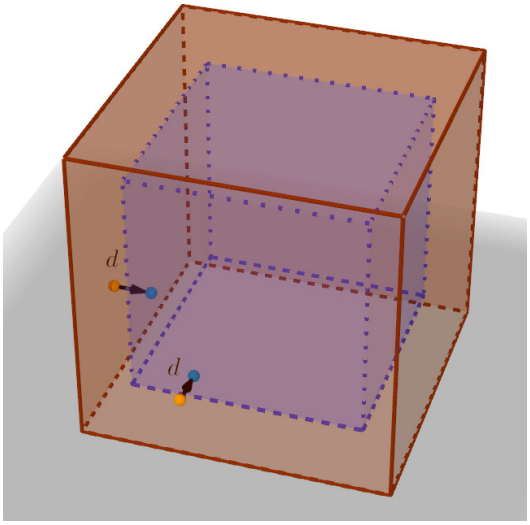
Az algoritmusban használt ofszet művelet további pontatlanságokat eredményez. Ezek az alábbiakban olvashatók.

Ofszet Az árnyék meghatározására a cone-tracing során az ofszet műveletet alkalmazza az algoritmus, ami negatív paraméter esetén zsugorítást jelent. Az élet tartalmazó objektumok negatív ofszetelésnél az él közelében jellemzően nagyobb mértékben fogynak, mint a művelet paramétere, ezt a jelenséget a 6.20. ábra hivatott magyarázni. Ennek következtében, ha egy részleges árnyékot (átmenet) egy él okoz, az átmenet jobban elterül az árnyék belseje felé.

Vékony objektumok Mivel az algoritmus egy cone-trace-t hajt végre, a negatív ofszetnek köszönhetően az árnyéksugár áthaladhat nagyon vékony objektumokon, mint a 6.21. ábra mutatja. Ahhoz, hogy ez megtörténjen az objektumnak vékonyabb-



(a) *Négyzet ofszete ($d < 0$)*



(b) *Kocka ofszete ($d < 0$)*

6.20. ábra. *Negatív ofszet esetén a csúcsok (2d és 3d) és az élek (3d) az ofszet méreténél nagyobb mértékben mozdulnak el.*

nak kell lennie, mint a kúp átmérője a metszés helyén, ami megfelel a fényforrás látszólagos méretének ezen a helyen.

(a) *Referencia árnyék MC-vel*

(b) *Egysugaras algoritmus*

6.21. ábra. *Vékony objektumon keresztülszűrődhet az egysugaras algoritmus során a fény.*

Távolságfüggvény becslés Ez alapvetően nem az algoritmus hibája, hiszen távolságfüggvényekre lett tervezve, és a legtöbb esetben ez nem okoz észrevehető különbséget, de sokszor nincs pontos távolságfüggvényünk. A különböző szilárdtest-műveleteknél (unió, metszet, kivonás), az eredetileg pontos távolságfüggvényekből képzett új függvény – néhány speciális esettől eltekintve – csak távolságfüggvény becslés, emiatt a kérdéses helyeken szintén más lehet az árnyék a pontos távolságfüggvényből számolthoz képest.

7. fejezet

Összefoglalás

Dolgozatomban egy olyan paraméteres algoritmust mutattam be, amellyel valós időben számíthatóak gömbfényforrásokkal megvilágított felületek által vetett puha árnyékok. Az algoritmus működése azon a feltevésen alapszik, hogy a színtérbeli geometriákat távolságfüggvényekkel definiáljuk. A módszer négy változatát dolgoztam ki, amelyek eredményeit egymással és egy bázissal (ground truth) szolgáltató Monte-Carlo módszerrel hasonlítottam össze, több teszt színtéren.

Az elsőként bemutatott naiv algoritmus bizonyult a javasolt változatok közül a leglassabbnak. A diszkretizált változat képminőségében sokkal rosszabbul teljesít azonos paraméterezés mellett, mint az eredeti, így ez nem bizonyult alkalmazhatónak. A linearizált változat az eredetivel azonos képminőség mellett annál jóval gyorsabb, így ez bizonyult a legegyszerűbben használhatónak általános színterekben. A konvex-optimalizált változathoz szükség van a távolságfüggvény konvex-optimalizált kiértékelésére, ezért ez nem alkalmazható minden esetben, viszont ha ez rendelkezésre áll, a linearizált variációnál is jobban teljesít.

Összegezve, tetszőleges színterekben a harmadik módszert, azaz az algoritmus linearizált változatát érdemes használni. Ez képminőségben és sebességben is jól teljesít. Abban a speciális esetben, ha távolságfüggvényeink konvex-optimalizált alakja is rendelkezésre áll, további teljesítménynövekedést hoz a negyedik módszer.

Az algoritmus érzékenységi paramétere közvetlenül befolyásolja az eredmény numerikus pontosságát. Empirikus megfigyelések alapján a fényforrás méretével fordítottan arányosan érdemes ezen érzékenységi paramétert beállítani. További fontos megfigyelés, hogy az $\frac{1}{30}$ alatti értékek már csak vizuálisan nem látható különbségeket eredményeznek.

Az algoritmus felhasználható továbbá fizikai alapú renderelésnél, terület-fényforrásokból származó illumináció meghatározására is. A módszer az integrál kiszámítását teszi függetlenné a színtér geometriájától a legköltségesebben számolható tag – láthatóság – közelítésével.

Továbbfejlesztési lehetőségek A dolgozathoz készített keretrendszer segítségével számtalan további fejlesztésre nyílik lehetőség. Az egyik legérdekesebb irány az algoritmus átdolgozása olyan módon, hogy iteratívan, folyamatosan javított minőségben számítsa a fényforrás hatását.

A dolgozatban bemutatott egysugaras puha árnyékszámító módszerek családja bővíthető lassabb, de annál pontosabb módszerekkel. Itt fontos vizsgálati irány lenne az árnyékolás pontosabb geometriai konfigurációjának számontartása, például a lokális minimumhelyen a felületi normális kiértékelésével az újabban kitakart kör-vetületet pontosabban lehetne becsülni.

Ábrák jegyzéke

Algoritmusok listája

Táblázatok jegyzéke

Irodalomjegyzék

Sebastian Aaltonen. GPU-based clay simulation and ray-tracing tech in Claybook. San Francisco, CA, March 2018. Game Developers Conference.

Andersson and Barré-Briseboi. Shiny pixels and beyond: Real-time raytracing at seed. 2018.

Nir Benty, Kai-Hwa Yao, Tim Foley, Conor Lavelle, and Chris Wyman. The Falcor rendering framework, 07 2017. `https://github.com/NVIDIAGameWorks/Falcor`.

Csaba Bálint and Gábor Valasek. Accelerating Sphere Tracing. In Olga Diamanti and Amir Vaxman, editors, *EG 2018 - Short Papers*. The Eurographics Association, 2018.

Csaba Bálint and Gábor Valasek. Interactive Rendering Framework for Distance Function Representations. In Tibor Tómács, editor, *Annales Mathematicae et Informaticae 48 (2018)*. Líceum University Press, 2018.

Bálint Csaba. Távolságfüggvényekkel definiált felületek interaktív megjelenítése, 2016. XXXIII. Országos Tudományos Diákköri Konferencia.

Ignacio Llamas Edward Liu. Ray tracing in games with NVIDIA RTX. San Francisco, CA, March 2018. Game Developers Conference.

Kenneth Eriksson, Donald Estep, and Claes Johnson. *Applied Mathematics: Body and Soul*, volume 1. Springer-Verlag Berlin Heidelberg, 1 edition, 2004. Volume 1: Derivatives and Geometry in IR3.

Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design (3rd Ed.): A Practical Guide*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

John C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1994.

Íñigo Quílez. Mandelbulb. `http://www.iquilezles.org/www/articles/mandelbulb/mandelbulb.htm`, 2009.

Íñigo Quílez. Penumbra shadows. `https://www.iquilezles.org/www/articles/rmshadows/rmshadows.htm`, 2010.

Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. Enhanced Sphere Tracing. In Andrea Giachetti, editor, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2014.

Stanley Osher and Ronald P. Fedkiw. *Level set methods and dynamic implicit surfaces*. Applied mathematical science. Springer, New York, N.Y., 2003.

Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2016.

Tim Reiner, Gregor Mückl, and Carsten Dachsbacher. Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. *Computers & Graphics*, pages 596–603, 2011.