Debreceni Egyetem

Informatika Kar

SHADOW MAPPING
MODERN GPU-KON

Témavezető:

Dr. Schwarcz Tibor

Egyetemi adjunktus

Készítette:

Lehőcz Kornél

Programozó matematikus

Debrecen

2007

Tartalomjegyzék

Bevezetés

Az árnyék sötétség, egy megvilágítatlan terület, a realisztikus számítógépes 3D grafikának egyik elengedhetetlen eleme. Egy kép melyen nincsenek árnyékok, nemcsak nélkülözi a valósághűséget, de a tárgyak térbeli viszonyának felmérését is megnehezíti a szemlélő számára. A háromdimenziós komputergrafikában három alapvető módszer alkalmazott az árnyékok szimulálására: a fénysugárkövetés, a volumetrikus árnyékok [Cro77], és a shadow mapping. Ezek közül a shadow mapping-et tárgyaljuk részletesen. Habár az eredeti módszert Lance Williams 1978-ban publikálta [Wil78], az algoritmusnak számos új, érdekes változata csak nemrégiben jelent meg, komolyabb kutatásnak és fejlődésnek e területen leginkább az elmúlt években lehettünk tanúi. Ennek fő oka a GPU-k (grafikai processzorok) rohamos fejlődése, melyek mostanra értek el olyan szintre, hogy a shadow mapping és annak különféle változatai számítógépes játékokban és más valósidejű alkalmazásokban ideális megoldást jelentenek.

Az algoritmus két lépésből áll:

1. Rajzoljunk egy mélység-puffert (tipikus esetben z-puffert) a fényforrás szemszögéből. A későbbiekben ezt nevezzük shadow map-nek.
2. A jelenet rajzolása során transzformáljuk az aktuális képpontot, melyet árnyalni szeretnénk, a fényforrás terébe. Amennyiben a shadow map-ben azon a ponton egy közelebbi érték szerepel, mint az aktuális pontunk z komponense, akkor a vizsgált pont árnyékban van.

Az algoritmus egyszerűen fogalmazva azt vizsgálja, hogy egy adott pont látható-e a fényforrás szemszögéből. A shadow map mindenhol a fényforráshoz legközelebbi felületet tartalmazza, amennyiben a vizsgált pont mögötte van, árnyékban van. Ezt az algoritmust alkalmazza számos film-produkciókhoz is használt csúcs-minőségű 3D renderelő szoftver, például a PhotoRealistic RenderMan, amellyel olyan filmek készültek, mint a Toy Story. Egyre több 3D-s játékprogram is ezzel a módszerrel rajzolja az árnyékokat.

1. ábra: Shadow map-pel renderelet teáskanna és a hozzá tartozó shadow map z-puffer.

A shadow mapping előnye az egyszerűsége és a sebessége komplex jelenetek esetén. A kirajzolás (a második lépés) színtisztán kép-térben dolgozik, így az időigénye elsősorban a kirajzolt kép felbontásának a függvénye. A shadow mapping nem függ az alkalmazott rajzolási módszertől, nem feltételez például poligon raszterizálással történő rajzolást. Viszonylag könnyen integrálható bármilyen módszert használó renderelőbe, nem szükséges a modellek geometriai adatainak bonyolult feldolgozása, mint a volumetrikus árnyékok esetében. Az algoritmus fényszóró-szerű fényekre a legalkalmasabb, de - bizonyos technikai kihívások leküzdésével - alkalmazható párhuzamos és pontszerű fényforrásokra is. Az algoritmus fő problémái a shadow map-ben használt diszkrét ábrázolásból fakadnak.

A dolgozatban bemutatjuk a módszer problémáit és rájuk létező megoldásokat. Részletes áttekintést kívánunk adni az e területen a közelmúltban történt legfontosabb kutatási fejleményekről, illetve tárgyaljuk a különféle módszerek implementációjánál felmerülő problémákat. A tárgyalást olyan módszerekre korlátozzuk, amelyek hatékonyan megvalósíthatóak modern* GPU-kon valósidőben, interaktív sebességek mellett.

* modern: Hamarosan elavult. A dolgozat írásának idején az nVidia G80 és az ATI R600 számítanak csúcskategóriás GPU-knak.

A shadow mapping algoritmus problémái

A shadow mapping algoritmus legnagyobb problémái a shadow map véges felbontásából fakadnak, mind a korlátozott vízszintes és függőleges felbontásból, mind a z-pufferben tárolt értékek véges pontosságú ábrázolásából. Az egyik probléma a recés árnyékszélek, egy másik komoly gond az önárnyékolás-hiba (shadow acne), mivel a véges felbontás miatt a shadow map-ből kiolvasott érték nem pontosan fogja visszaadni az ottani mélységet, így egy felület gyakran helytelenül vet árnyékot önmagára a fényforrás felé néző oldalán is. A közeli árnyékoknál jelentkező recés széleket szokás perspektivikus aliasing-nak is nevezni. A shadow mapping témát feldolgozó akadémiai cikkekben szintén felmerülő projektív aliasing valójában az önárnyékolás-probléma egy speciális esete, amikor a fényforrással párhuzamos felületeken elnyúlt árnyék-csíkok jelennek meg.

Egy nyilvánvaló módszer e problémák mérséklésére a shadow map felbontásának növelése, ez viszont roppant költséges lehet, mind a számítási igény, mind a memóriahasználat szempontjából. Ne feledjük továbbá, hogy ezzel csupán csökken a hibák mértéke, de a problémák nem szűnnek meg teljesen. Léteznek olyan megoldások, amelyek a shadow map aktuális nézőpont alapján történő betorzításával törekednek a shadow map területének jobb kihasználására, ilyenek például a perspektivikus és a trapéz shadow map-ek. Csökkenthetjük az árnyékszélek recésségét és az önárnyékolás problémát a shadow map filterezésével, de erre is speciális módszerek léteznek, mivel a színes textúrákhoz szánt filterezési módszerek a mélység-puffereknél helytelen eredményt produkálnak.

A shadow mapping fényszóró-szerű fényforrásokra a legalkalmasabb. Pontszerű fényforrásokra, amelyek minden irányba világítanak, több shadow map-et kell alkalmazni, melyek lefedik a teljes környezetet. Az egyik lehetőség a cube mapping, azaz 6 shadow map használata. Ez implementálható talán a legkönnyebben, továbbá a cube map-ekből való olvasásra hardveres támogatás is van a mai GPU-kon. Ennek a módszernek az a hátránya, hogy hatszor kell renderelnünk a jelenetet. Ez nem feltétlenül jelent olyan nagy gondot, mivel egy tipikus jelenetben a legtöbb tárgy csak a kocka egyik oldalához tartozó látómezőbe esik bele. Egy másik lehetőség például a duál-parabola mapping [WH98], melynek segítségével két shadow map

elegendő a tejes környezetet lefedéséhez. A fő probléma ezzel a megközelítéssel az, hogy az új parametrizációval egyenes vonalakból görbék lesznek, a hardver viszont továbbra is egyenes szélű háromszögeket rajzol. Kellően tesszellált objektumok mellett az ebből származó pontatlanság elhanyagolható. [BAS2002]

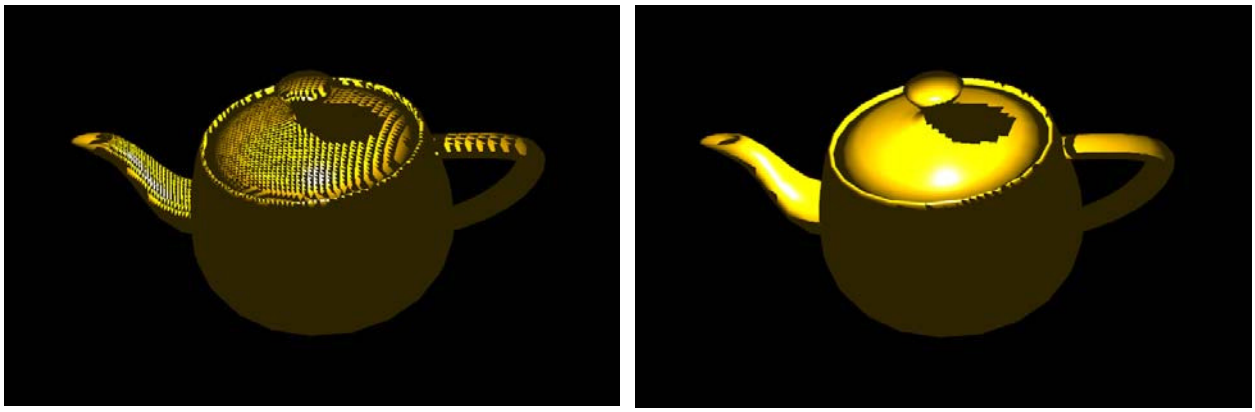
Párhuzamos fényekre a shadow mapping módszer alapvetően ugyanúgy működik, mint fényszórószerű fényekre, azzal a különbséggel, hogy a 4×4-es mátrix, mellyel a fényforrás látómezőjét írjuk le, itt egy párhuzamos vetítést reprezentáló mátrix. Párhuzamos fényt általában a napfény szimulálására szokás használni kültéri jelenetekben. A kihívás ezeknél a kültéri jeleneteknél, az, hogy többnyire hatalmas területet kell lefednünk egy vagy több korlátozott méretű shadow map-pel.

Egy más jellegű érdekes probléma, amely pont és fényszórószerű fényforrások esetén merülhet fel, hogy a z-puffer értékeinek eloszlása a legtöbb 3D-renderelőben nem lineáris, a nézőponthoz közel nagyobb a felbontása, mert a homogén koordinátáknál a w-vel való osztást a z-n is elvégzi. Mivel a shadow map rajzolásakor a fényforrás a nézőpont, hozzá közel nagyobb a z felbontása, viszont az árnyékot vető tárgyak adott esetben távolabb is lehetnek a fényforrástól. Előfordulhat, hogy e tárgyaknak az árnyékát látjuk közelről az aktuális képen, így nem feltétlenül optimális az egyenetlen eloszlású z-puffer. A nem egyenletes eloszlás mellett szól viszont, hogy az ilyen típusú fényforrásoknak általában kicsi a hatósugara, így tőlük távolodva az árnyékok is halványulnak, ezért távolabb kevésbé feltűnőek a pontatlansági hibák.

Felmerül a kérdés, hogy mekkora shadow map szükséges egy adott jelenthez. Tipikusan azt szeretnénk elérni, hogy a képen minden pixelre legalább egy shadow map texel jusson. Abban az esetben, ha a fényforrás pozíciója és látómezeje majdnem megegyezik a kameráéval, körülbelül ugyanakkora shadow map szükséges, mint amekkorában a képet rendereljük. Ez a fajta jelenet a shadow mapping számára a legideálisabb. Más helyzetekben a szükséges méret meghatározása nehezebb, gyakorlatilag attól függ, hogy az aktuális képen egy adott méretben látszó árnyék mekkora részét képezi a shadow map által lefedett területnek. A legrosszabb esetek általában akkor állnak elő, amikor a kamera és a fényforrás egymással szemben vannak, ez a „dueling frusta”, azaz párbajozó látómezők problémaként ismert.

Megoldások az önárnyékolás problémára

Williams az eredeti shadow mapping-et bemutató cikkében a helytelen önárnyékolás probléma megoldására egy eltolás érték (bias) használatát javasolja, azaz adjunk hozzá egy értéket a shadow map-ből kiolvasott mélység értékhez, ami gyakorlatilag hátrébb tolja az árnyékokat. (Feltételezzük, hogy a z-puffer-ben a nulla a legközelebbi értéket reprezentálja. Ellenkező esetben kivonás szükséges.) Amennyiben túl nagy eltolást alkalmazunk, az a probléma jelentkezhet, hogy az árnyék szemmel láthatóan hátrébb kezdődik, mint ahogy annak a valóságban kellene. Ez azt a hatást keltheti, hogy a tárgy lebeg a levegőben, vagy a képbe nem illő fényes csík jelenhet meg bizonyos tárgyak árnyékos oldalán. Nincs képlet a szükséges eltolás meghatározására, ennek mértékét konkrét jelenetekhez kell beállítani, attól függően, hogy az adott jelenetben milyen típusú hibák kevésbé szembetűnők. Nagyobb felbontású shadow map esetén kisebb eltolásra van szükség.



2. ábra: Shadow map-pel renderelet teáskanna eltolás nélkül és eltolás alkalmazásával.

Az eltolást tehetjük a shadow map-ből történő olvasás helyett a shadow map-be való rajzolás elé is. Fontos odafigyelni arra, hogy pont- és fényszórószerű fényforrások esetén a shadow map-ben levő mélység-értékek eloszlása a tipikus implementáció esetén nem lineáris, mivel w-vel osztott értékek kerülnek a pufferbe. Ebben az esetben egy fény-térben történő fix

mértékű eltolás lehet, hogy elegendő a közeli tárgyakhoz, de kevésnek bizonyulhat a távoliaknál. Ehelyett a shadow map-be írt vagy onnan kiolvasott értékhez javasolt adni az eltolást.

Az eltoláshoz egy véletlen-számot is érdemes lehet adni, hogy az esetleges fennmaradó hibák kevésbé legyenek nyilvánvalóak.

Az egyik probléma a z-eltolás megoldással, az hogy egy felületen több önárnyékolási hiba jelentkezhet, amikor a felület a fényforrás szemszögéből élesebb szögben látszik, illetve e felületek miatt nagyobb mértékű eltolás szükséges. Egy kézenfekvő megoldás erre a problémára a meredekséggel skálázott eltolás (slope scale bias), azaz az élesebb szögben látszó felületek esetén egyszerűen nagyobb eltolást használjunk. Az eltolás mértékét az alábbi képlettel szokás meghatározni:

$$m = \max(|\partial z / \partial x|, |\partial z / \partial y|)$$

Sajnos ez a megoldás is eredményezhet hibákat. A nagyon éles szögben látszó felületek mögött, mivel így bizonyos esetekben nagyon nagy lehet az eltolás, az árnyékos területeken fényes csíkok jelenhetnek meg. A gyakorlatban célszerű az össz-eltolás mértékét m 0 és 1 közötti számmal vett szorzata és egy fix eltolás összegeként meghatározni, és ezeket a jelenet sajátosságai szerint beállítani.

Lebegőpontos shadow map

Reeves és társai ’87-es publikációjukban [RSC87] megemlítik, hogy a shadow map-ben lebegőpontos számokat használva 16 bites egészek helyett (mint azt eredetileg Williams tette), csökken az önárnyékolás-hiba mértéke, de nem oldja meg a problémát. Ha megvizsgáljuk a lebegőpontos számok ábrázolását, hogy mely tartományokon milyen pontosságot biztosítanak, láthatjuk, hogy az alap shadow mapping algoritmushoz nem optimálisak. A shadow map által lefedett tartomány második felében a pontossága azonos egy olyan fixpontos ábrázolás pontosságával, mely bitjeinek száma ugyanannyi, mint a lebegőpontos ábrázolásban a mantissza bitjeinek száma. Egy IEEE 754 32 bites lebegőpontos szám esetén ez 23 bit, viszont a rengeteg

GPU által is támogatott 16 bites lebegőpontos formátum esetén csupán 10 bit. Az imént említett cikk szerzői feltehetőleg 32 bites lebegőpontos ábrázolást használtak és ahhoz hasonlították a 16 bites fixpontos z-pufferrel kapott eredményt. Lebegőpontos ábrázolásnál az előjelbitet elveszítjük, amennyiben a nullát tekintjük a legközelebbi pontnak. Lebegőpontos shadow map-nél a z-eltolás mértékét is problémásabb beállítani, mert nagyobb eltolás szükséges a fényforráshoz távolabb eső dolgokhoz (feltételezve, hogy 0 jelenti a legközelebbi pontot). Ha fixpontos ábrázolást alkalmazunk, fontos a shadow map tartományát beállítanunk, azaz, hogy mekkora távolságra vonatkozik. Lebegőpontos ábrázolás esetén kevésbé valószínű, hogy problémát okoz ennek elmulasztása.

Hátoldalak rajzolása

Yulan Wang és Steven Molnar a helytelen önárnyékolás problémára azt javasolja, hogy a shadow map-be csak a fényforrásnak háttal néző felületeket rajzoljuk bele [WM94]. Így a hiba átkerül a tárgyak másik oldalára, amely eleve sötét az amúgy is alkalmazott megvilágítási egyenleteknek (például Phong árnyalási modellnek) köszönhetően, ezáltal elrejti a hibát. Ez a megoldás megköveteli, hogy a jelenetben az összes modell felülete hermetikusan zárt legyen. A módszer jelentős sebességnövekedéssel is járhat, mivel nagyjából megfeleződik a shadow map-be rajzolandó poligonok száma.

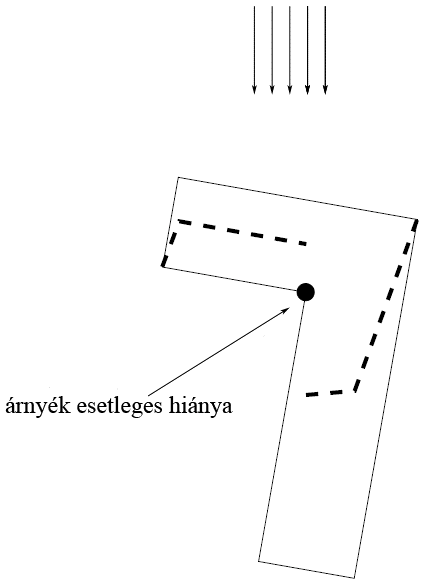
Az a probléma merül fel, ha csak a háttal néző poligonokat rajzoljuk a shadow map-be, hogy a tárgyak túlsó oldalán az élek mentén és a fénnyel majdnem párhuzamos felületeken fényes csíkok jelenhetnek meg. Ennek korrigálására egy negatív eltolást lehetne alkalmazni, emiatt viszont ismét a fényes oldalon jelentkezne helytelen önárnyékolás. Egy további hely, ahol hibás önárnyékolás jelentkezhet, a tárgyak körvonala, de ez nem jelentős, mert a tárgy árnyékos oldala amúgy is körülbelül ott kezdődik. Ez a megoldás, hogy csak a hátoldalakat rendereljük a shadow map-be, tipikus esetben viszonylag kevés hibát eredményez, és gyorsasága miatt ideális módszer lehet valósidejű alkalmazásokhoz, ahol kisebb hibák elfogadhatóak. A módszert egy OpenGL vagy Direct3D alapú shadow map implementációba belerakni roppant egyszerű, csupán a hátsó lap eldobást (backface culling) kell a shadow map renderelésekor megfelelően beállítani.

Egy érdekes optimalizálási lehetőség, amely főleg akkor hatékony, ha csak a háttal néző poligonokat rajzoljuk a shadow map-be az, hogy kihagyhatjuk a shadow map törlését, ha feláldozunk egy bitet a z-puffer pontosságából. Páratlan képkockák rajzolásakor a z-puffernek csak a 0-0,5 tartományát használjuk, páros képkockáknál pedig a 0,5-1,0 tartományt, és invertáljuk a mélység-tesztet.

Midpoint shadow mapping

Létezik egy módszer, amely szinte tökéletesen megszünteti a helytelen önárnyékolás problémáját, ez pedig a midpoint shadow mapping [Woo92], mely Andrew Woo nevéhez fűződik. Az alapötlet az, hogy rajzoljunk egy második shadow map-et is, amely a fényforrás szemszögéből nem a legközelebbi felületet mélységét tartalmazza, hanem az a mögött levő felület mélységét, majd a mélység összehasonlításnál a két shadow map-ben levő értékek átlagát használjuk. Ezzel gyakorlatilag egy virtuális felületet hozunk létre, amely mindig a fényforrás szemszögéből két legközelebb levő felület között húzódik, és ez veti az árnyékokat.

Bizonyos esetekben ez a módszer is eredményezhet kisebb hibákat. A tárgyak fény szemszögéből vett körvonala hibásan önárnyékolhatja magát, akárcsak annál a módszernél, amikor csak a hátoldalakat rendereljük a shadow map-be. Egy kicsit nagyobb gondot jelenthet egy speciális eset, amikor helytelenül nem kerülnek árnyékba bizonyos részek. Ez a tárgyak bemélyedő részének az élén jelentkezhet, amennyiben a virtuális árnyékvető felület közvetlen a bemélyedés mellett az árnyékolt pont mögé esik.



3. ábra: Egy speciális eset, amikor a midpoint shadow mapping hibás eredményt produkál.

Weiskopf és Ertl ennek a módszernek az egy kicsit általánosított változatát nevezte el dual shadow mapping-nek [WE03]. Ők azt vetették fel, hogy a virtuális felület nem csak a két mélység-puffer átlagából jöhet ki, hanem használhatunk ettől eltérő függvényt is, és az alábbi függvényt javasolják:

$$Z_{bias}(Z_1, Z_2) = \min\left(\frac{Z_2 - Z_1}{2}, Z_{offset}\right)$$

Ez lényegében ugyanaz a módszer egy maximum eltolás használatával. Így elkerülhetőek azok az esetek, amikor helytelenül nem kerülnek árnyékba bizonyos részek. Egy további érdekes megfigyelés, hogy amennyiben a shadow map rajzolásakor a háttal néző lapokat eldobjuk, azaz csak a felénk néző felületeket rajzoljuk, akkor hibás önárnyékolás esetek is megszűnnek ezzel a módszerrel. Amennyiben az eredeti midpoint shadow map módszernél tesszük ugyanezt, a hibák csupán máshova helyeződnek át. Fontos megjegyezni, hogy amennyiben hátsó-lap eldobást is alkalmazunk, meg kell követelni a jelenetben szereplő összes objektumtól, hogy a felületük hermetikusan zárt legyen.

Egy valósidejű implementáció szempontjából az a legnagyobb probléma a midpoint és dual shadow mapping-gel, hogy számításigényes, mivel két shadow map-et kell kirajzolnunk. A 3.0-ás shader modellt támogató GPU-kon ez megvalósítható egy menetben, mert itt a pixel shader-ből le tudjuk kérdezni, hogy az aktuálisan renderelt pixel egy felénk néző poligon része-e. Ez alapján két színcsatorna valamelyikébe írunk. További feltételei ennek megvalósíthatóságának, hogy az adott render target textúrára támogassa a GPU a post pixel shader blending-et és a minimum vagy maximum blend módot. Ez a módszer is megköveteli, hogy az objektumok zártak legyenek.

ID-puffer

Az önárnyékolás problémához a z-puffer korlátozott pontossága is hozzájárul. Egy érdekes alternatíva, főleg ha nagyon korlátozott a shadow map pontossága (pl. 8 bites), hogy mélységek helyett objektum ID-ket tárolunk a shadow map-ben, az árnyalás során végzett összehasonlításkor pedig a shadow map-ben levő ID azonosságát vizsgáljuk az éppen kirajzolt objektum ID-jével. Amennyiben a két ID azonos, nincs árnyék az adott ponton. Ezt a módszert Hourcade és Nicolas [HN85] javasolta. E megoldás egy nyilvánvaló limitációja, hogy nem kezeli az objektumokon belül vetett árnyékokat, azaz amikor egy tárgy saját magára vet árnyékot. Kézenfekvő megoldásnak tűnhet, hogy minden egyes háromszögnek külön ID-t adunk, de ez sajnos a háromszögek szélén helytelen recés árnyékhoz vezet, így nem járható út.

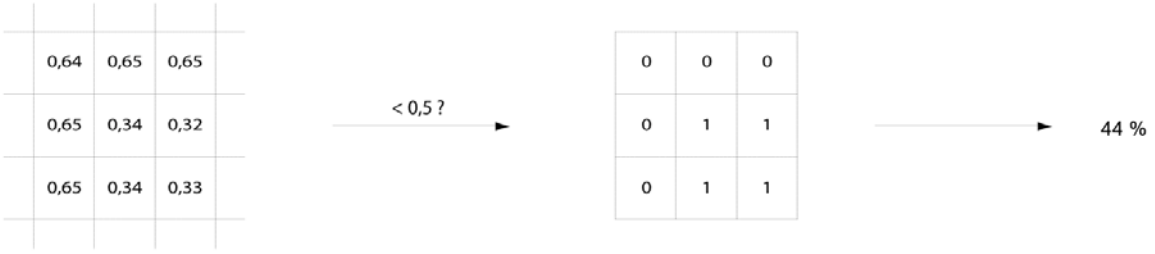
Egy ilyen típusú 8 bites shadow map-ben csak 256 különböző ID tárolható, így egy összetettebb jelenet esetén ID-k újrafelhasználására kényszerülhetünk. Az ID ütközés esélye így is csekély. Amennyiben mégis ütközés történne, az eredmény árnyék hiánya lehet. Egy érdekes lehetőség az ID- és mélység-pufferes módszerek összekombinálása, tárgyak közötti árnyékokra ID-puffert, tárgyakon belül mélység-puffert használni. Ilyenkor minden egyes tárgyhoz hozzá tudjuk igazítani a mélység-puffer tartományát. Ennek a kombinált módszernek akkor lehet például értelme, ha több kilométeres távolságot szeretnénk lefedni a shadow map-pel, de spórolni szeretnénk a memóriával, memória-sávszélességgel. Ezzel a módszerrel egy 16 bites puffer elegendő lehet; 8 bit a mélységre, 8 bit az ID-re.

Filterezés

A shadow map-ek a színes textúráknál megszokott módon történő filterezése, például bilineáris filterezéssel, helytelen eredményt ad, a körvonalak mentén az összeátlagolásból nem kívánt mélység értékek jönnek ki.

Percentage closer filterezés

Az egyik megoldás a percentage closer filterezés [RSC87], melynek a lényege, hogy a mintákon először a mélység-összehasonlítást végezzük el, majd utána megnézzük, hogy hány százalékuk ment át a mélység-teszten. Például ha egy 3×3-as filter-magot használunk, és a 9 minta közül 4 megy át a mélység-teszten, akkor azt mondhatjuk, hogy a pont 44%-ban árnyékban van.



4. ábra: A percentage closer filterezés minden egyes mintán először elvégzi a mélység-tesztet, majd megnézi, hogy a minták hány százaléka ment át a teszten.

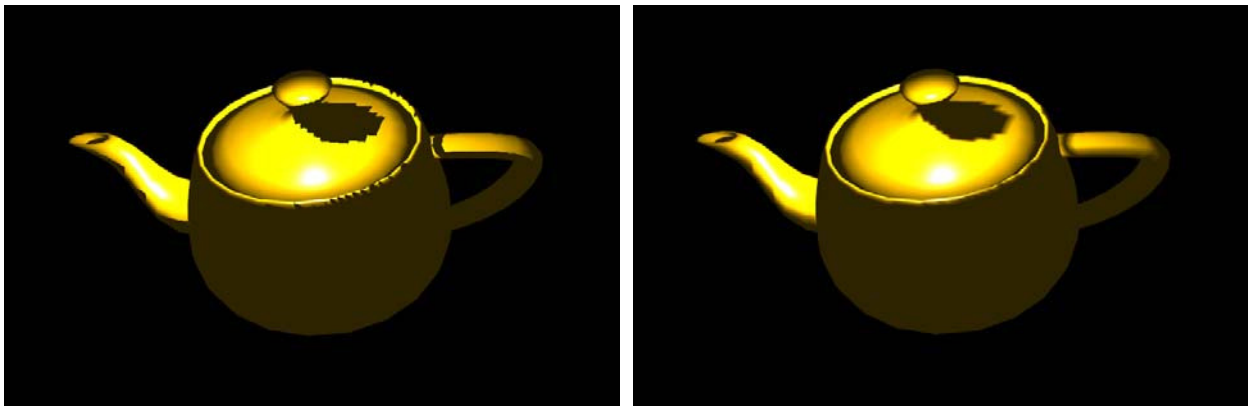
A módszer eredményeképpen lágyabb árnyékszélek keletkeznek, de ez mégis messze van a fizikailag korrekt lágy árnyéktól, amely élesebb, ha az árnyékvető közelebb van az árnyékhoz és lágyabb, ha távolabb van.

Csökkenthetjük a szükséges minták számát Monte Carlo módszerek alkalmazásával [Coo86]. Az aliasing problémáknak az oka nem csupán az alul-mintavételezettség lehet, hanem a mintavétel szabályossága is. Amennyiben véletlenszerű helyeken vesszünk mintát, kevesebb

mintával érhetünk el kinézetre hasonló eredményt. Egy tisztán véletlenszerű eloszlással a minták hajlamosak egyes helyeken összetorlódni, más helyeken pedig nagy hézagokat hagyni. Jó eredményeket Poisson-eloszlású mintákkal érhetünk el, de egy implementáció, mely Poisson-eloszlást produkál, túl számításigényes. Jól közelíthetjük viszont a Poisson-eloszlást „jittering" alkalmazásával, azaz zaj hozzáadásával fix mintavételezési pozíciókhoz. A fix mintavételezési helyeknek használhatunk például egy négyzetrácsot, ez árnyékok esetén jó eredményt produkál.

Egy GPU-n futó implementációban célszerű a véletlen-számokat egy zajt tartalmazó kétdimenziós textúrából venni, ezt a textúrát pedig érdemes a fény terében az x, y koordináta alapján címezni. Így két egymást követő képkockában ugyanazon a helyen ugyanazt a véletlen-számot kapjuk. Olyan megoldásoknál, ahol ez nem teljesül, animált zajt kapunk, mely árnyékok esetén nem előnyös, mert nem kívánt figyelmet terel az árnyékok szélére.

Bilineáris percentage closer filterezésről akkor beszélünk, ha a 4 legközelebbi ponton végezzük el a mélység-tesztet, tehát 2×2-es filter-magot használunk, majd a tesztek eredményein vízszintesen és függőlegesen is lineáris interpolációt végzünk. A percentage closer filtering nagymértékben csökkenti az árnyékok széleinek recésségét, így elsősorban alacsony felbontású shadow map-eknél hoz nagy javulást a képminőségben.



5. ábra: Shadow mapping filterezés nélkül és bilineáris percentage closer filterezéssel.

A bilineáris percentage closer filterezésnél is érdemes lehet véletlenszerű eltolást alkalmazni, de ugyanazzal a módszerrel problémákba ütköznénk, így ezt másképp érdemes

megvalósítani. Ahelyett, hogy mind a négy mintánál külön eltolást használnánk, itt egy közös eltolást érdemes alkalmazni. Ezzel valamelyest elrejthetjük az árnyékszélek szabályszerű recéit és hitelesebb lágy árnyékhatást kelthetünk. Hasonló eredményt érhetünk el ezzel a módszerrel 4 mintával, mint a jitter-elt sima PCF-el 9 mintával.

Az nVidia GPU-iban levő hardveres shadow mapping támogatja a bilineáris percentage closer filterezést, és ennek használata nem jár lassulással. Ezek a GPU-k támogatják a shadow map-ben mip-map-ek használatát is, viszont a mip-map szintek elkészítésének feladata a programozóra hárul, és számításigényes is lehet. Ne feledjük, hogyha a kisebb mip-map szinten a texel-ek értékét az egyel fentebbi mip-map szinten levő értékek összeátlagolásával számolnánk, akkor rossz eredményhez jutnánk. Az egyik járható út egy olyan shader program használata, ami 4 érték közül a legnagyobbat választja ki.

Szórásnégyzet shadow map-ek

Donelly és Lauritzen [DL06] egy olyan ábrázolást kerestek, amelyre működnek a GPU-kban levő színes textúrákra szánt filterezési módok, mint például az anizotropikus szűrés és a trilineáris filterezés. Ahelyett, hogy a shadow map-ben mélység értékeket tárolnának, minden texel-nél a mélység várható értékét és a mélység várható értékének a négyzetét is eltárolják, azaz a mélységek, mint valószínűségi változók, momentumait. Ezzel a kompakt ábrázolással közelítik a mélységek eloszlását minden texel-nél. Ennek az ábrázolásnak a nagy előnye, hogy momentumainak összeátlagolásával közelíthetjük két eloszlás átlagát. Miután a szórásnégyzet shadow map-ből kiolvasunk egy értéket, ki lehet számolni egy felső korlátot arra, hogy az eloszlás mekkora része van távolabb, mint az a pont, melyről épp el akarjunk dönteni, hogy árnyékban van-e. Ezt a Chebychev egyenlőtlenség felhasználásával kapjuk meg.

$$P(x \geq t) \leq p_{\max}(t) \equiv \frac{\sigma^2}{\sigma^2 + (t-\mu)^2}$$

A várható értéket kiolvashatjuk közvetlen a shadow map-ből (egy mélység várható értéke maga a mélység), a szórásnégyzetet pedig a kiszámolhatjuk a közismert képlet alapján:

$$\sigma^2 = E(x^2) - E(x)^2$$

Habár csak egy felső korlátot kapunk arra, hogy az eloszlás mekkora része van távolabb, ez tipikus esetben egy jó közelítése annak, hogy ténylegesen mekkora része van messzebb.

Mivel a momentumok interpolálhatóak, így használhatjuk ugyanazokat a filterezési technikákat, amelyek a színes textúrákhoz rendelkezésre állnak, és így a recés árnyékszélek és interferencia problémák (aliasing) orvosolhatóak. Lehetségessé válik a shadow map elő-filterezése is, így például használhatunk egy szeparálható konvolúciós filtert az árnyékszélek elmosására.

A szórásnégyzet shadow mapping sajnos bizonyos esetekben „fényszivárgás” hibát produkál, tipikusan akkor, amikor több árnyékvető van egymás mögött. Erre a problémára jelenleg még nem létezik megoldás. Egy másik probléma a numerikus stabilitás. Ahhoz, hogy a módszer jól működjön, kétcsatornás 32 bites lebegőpontos textúra használata javasolt, emiatt elég komoly memóriaigénye van, illetve nagy memória-sávszélességet használ.

A shadow map területének jobb kihasználása

A shadow map területének jobb kihasználásának egyik módja, hogy a shadow map-be csak azokat a tárgyakat rajzoljuk, melyek árnyéka az aktuális nézetből potenciálisan látható. Ezekre ráfókuszáljuk a shadow map-et, azaz megkeressük a legkisebb befoglaló-négyzetüket, és ezzel töltjük ki a shadow map-et.

Egy érdekes megoldás az adaptív shadow map [FFGB01], ahol a szokásos shadow map-et egy adaptív, hierarchikus ábrázolással váltják ki, melyet mozgás közben folyamatosan frissítenek. Minden képkocka rajzolásakor az első render-menet annak a felmérése, hogy az adatstruktúra mely részeit szükséges frissíteni. A shadow map legkritikusabb részeit kirenderelik nagyobb felbontásban és beleillesztik a hierarchikus shadow map adatszerkezetbe, a túlmintavételezett részeket eldobják. A hierarchikus adatszerkezetet menedzselése GPU-n kihívást jelent, de Lefohn és társai bemutattak egy implementációt, amely ezt megoldja, bár továbbra is szükséges, hogy a CPU a GPU-ról visszaolvasott adatok függvényében adjon ki a GPU-nak további rajzolási parancsokat. [LSKSO05] Publikációjukból az is kiderül, hogy a jelenlegi hardverek még nem elég gyorsak ahhoz, hogy az adaptív shadow mapping valósidejű alkalmazásokra praktikus legyen.

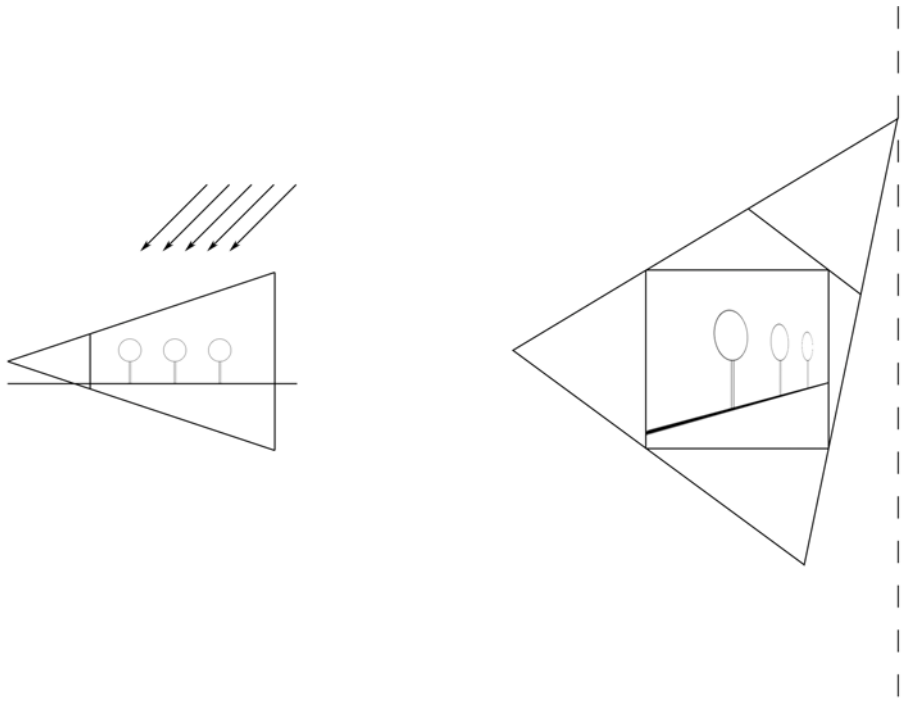
Léteznek megoldások, amelyek úgy csökkentik az árnyékszékszéleknek a főleg közeli tárgyakon feltűnő recésségét, hogy betorzítják a shadow map-et az aktuális nézőpont alapján. Ilyen a perspektivikus shadow map és különböző variációi.

Perspektivikus shadow map-ek

Stamminger és Drettakis megoldása a perspektivikus aliasing problémára az, hogy az aktuális nézőpont perspektivikus transzformációjával torzítják be a shadow map tartalmát [SD02]. Így nagyobb területet kapnak a shadow map-ben azok a részek, amelyeket közelebbről látunk. A shadow map projekció továbbra is leírható egy 4x4-es mátrixszal, így igénybe vehetjük a

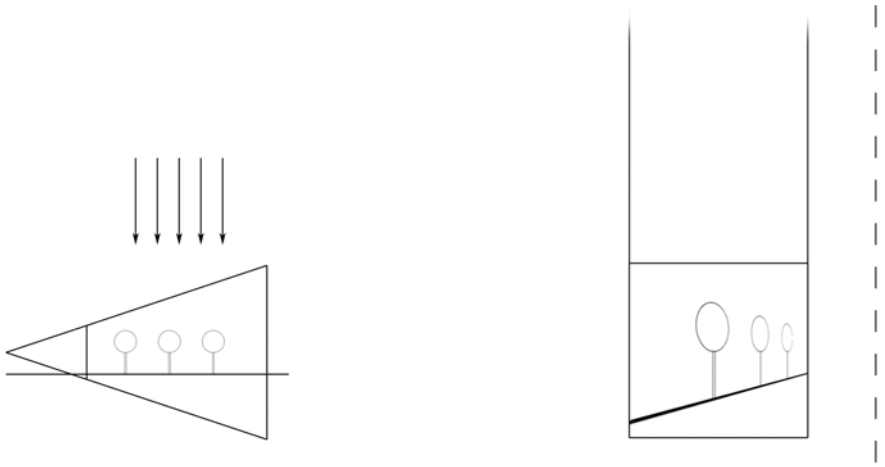
hardveres shadow mapping támogatást. A módszerrel sok esetben nagymértékben javul az árnyékok minősége, és nem igényel extra erőforrásokat.

A shadow map-et az aktuális nézőpont poszt-perspektivikus terében rajzoljuk. A jelenettel együtt a fényforrást is el kell transzformálnunk. A párhuzamos fényeket tekinthetjük fényszórószerű fényeknek a végtelenben, ezek a perspektivikus transzformáció által véges pozícióba kerülhetnek. A párhuzamos fényekből a perspektivikus transzformáció hatására fényszórószerű fények lesznek egy távoli vágósík mögötti, a képernyőre párhuzamos síkon, melynek pontos holléte a közeli és távoli vágósíkok pozíciójának függvénye. (A fény új pozíciójának kiszámításához nekünk azt csupán a projekciós mátrixszal kell megszoroznunk, majd w-vel osztanunk.)



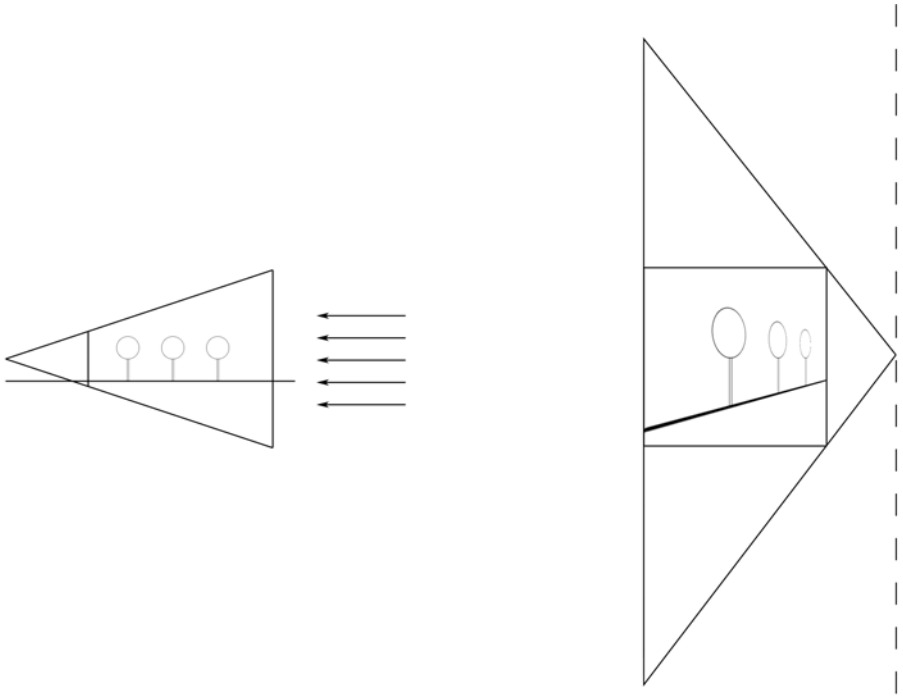
6. ábra: Párhuzamos fényforrások a perspektivikus transzformáció hatására fényszórószerű fényforrássá képződnek le.

Egy párhuzamos fény, amely a kép síkjával párhuzamos, párhuzamos marad a perspektivikus transzformáció után is, azaz egy fényszórószerű fény lesz a végtelenben.

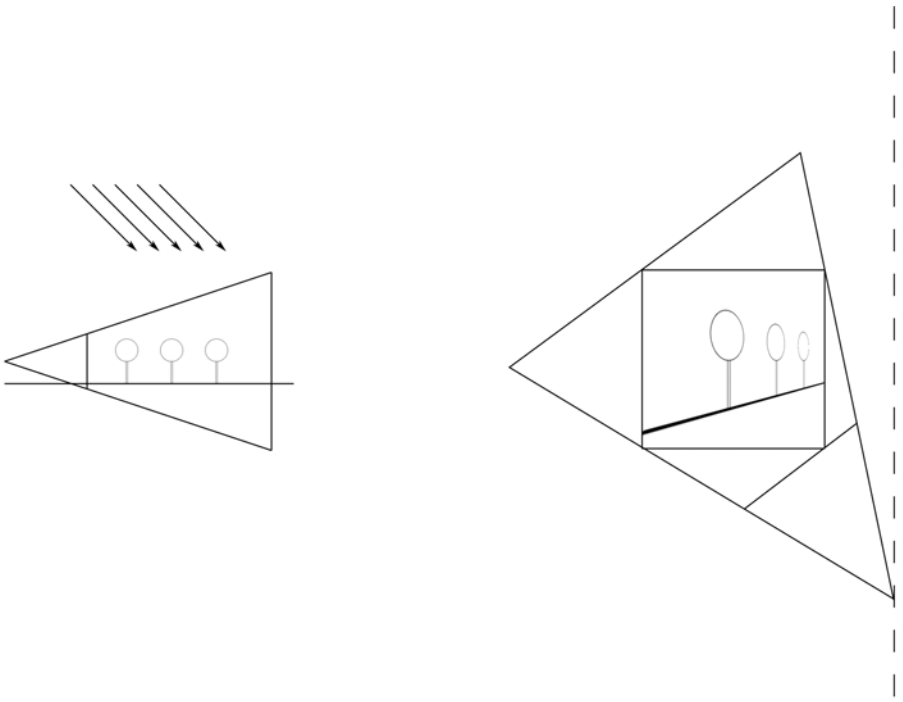


7. ábra: Egy párhuzamos fény, amely párhuzamos a kamera képsíkjával, párhuzamos fény marad a perspektivikus transzformáció után.

Egy párhuzamos fény, amely a kamera síkja mögül világít, egy invertált fényszórószerű fényforrássá képződik le, azaz sugarai nem szét, hanem „össze” tartanak. Ezt az esetet külön kell kezelnünk úgy, hogy a mélység tesztet invertáljuk. Amennyiben egy párhuzamos fényforrás pontosan a kamerával szemben van, az egy szembe levő pontszerű fényforrássá képződik le. Ez egy szélsőséges eset, mivel ez a fényforrás ugyanazt „látja”, mint az eredeti egy sima shadow map esetén.



8. ábra: Egy párhuzamos fény, amely pontosan szembenéz a kamerával, pontosan ugyanazt „látja” perspektivikus transzformáció után is.



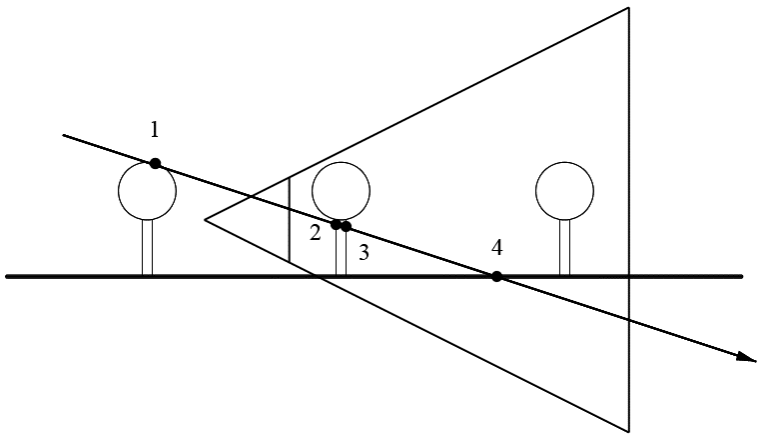
9. ábra: A kamera mögül érkező fény kifordul, azaz fordított sorrendben „látja” a dolgokat.

Fényszórószerű fényekre az eljárás nem különbözik, mivel a párhuzamos fényeket is fényszórószerű fényeknek tekintettük.

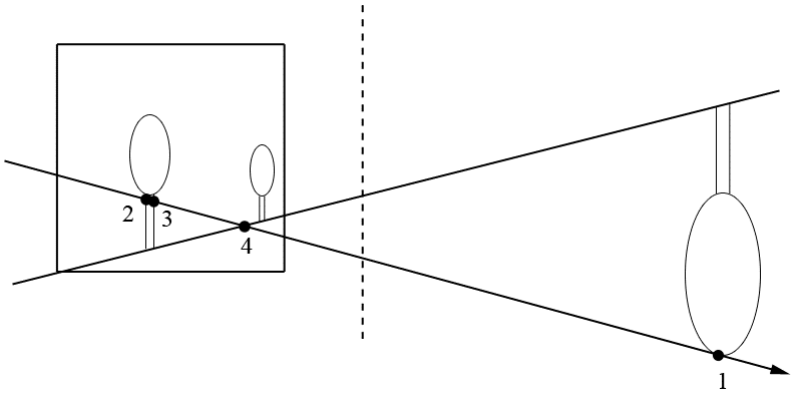
Egy GPU-n futó implementáció esetén a shader azonos a sima shadow mapping-hez használt shader-rel. A különbség a fényforrás mátrixának kiszámításában van, illetve a speciális inverz fényforrás lekezelésében.

A módszer hatékonysága függ a kamera és a fényforrás által bezárt szögtől. Abban az esetben, ha a fény a perspektivikus transzformáció után párhuzamos lesz, a perspektivikus aliasing teljes mértékben megszűnik, ekkor a shadow map-ben az objektumok mérete egyenes arányban lesz a képernyőn elfoglalt méretükkel. Ez az eset akkor áll élő, amikor a fény merőleges a kamera látómezejének irányára. Egy konkrét példa erre az, amikor a nap pont az égbolt tetején van, és a kamera vízszintesen néz. Amikor a kamera és a fényforrás iránya egymáshoz képest a párhuzamoshoz közelít, a parametrizáció a standard shadow map-jéhez degradálódik.

Egy probléma a perspektivikus shadow mapping-gel az, hogy amikor háttal nézünk a fényforrásnak, egyes tárgyak is mögöttünk lehetnek, melyek árnyéka látható kellene, hogy legyen az aktuális képen. Ezeket a perspektivikus transzformáció a végtelen-sík másik oldalára képezi le, ennek során egy sugár mentén levő pontok sorrendje is megváltozik, ezt szemlélteti az alábbi ábra:

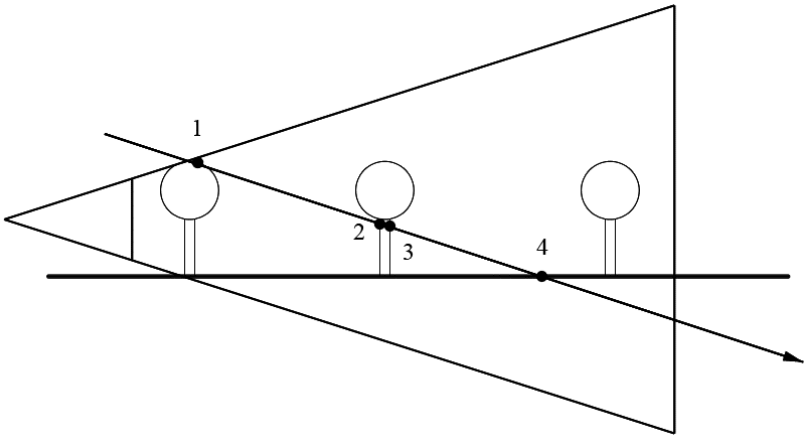


10. ábra: Egy potenciális árnyékvető a kamera mögött.



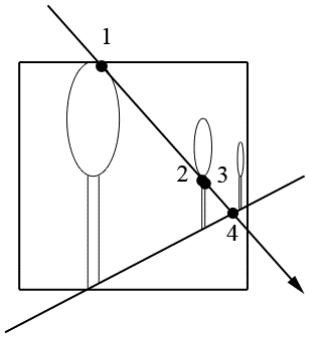
11. ábra: A kamera mögött levő tárgy a perspektivikus transzformáció hatására átkerül a képsík másik oldalára és invertálódik.

Ennek megoldására a kamera „virtuális” hátratolását javasolják, úgy hogy az összes potenciális árnyékvető belekerüljön a transzformált kamera látómezejébe. [SD02]



12. ábra: A vetítés középpontját hátrébb toljuk, hogy minden potenciális árnyékvető korrektül belekerüljön a shadow map-be.

Ez esetben a távoli vágósíkot hátrébb kell tolni úgy, hogy az új látómező csonka gúlája magába foglalja a korábbiét. Ez a kamera-hátratolás csak a shadow map-re vonatkozik, a kamera helyzete, mellyel a végső képet rajzoljuk, változatlan marad.

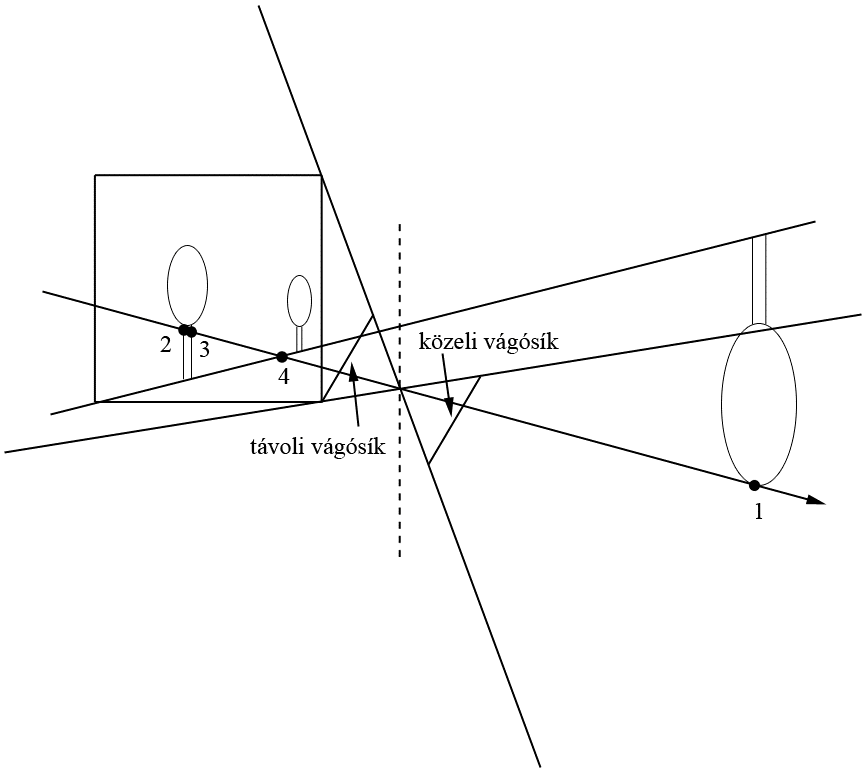


13. ábra: A poszt-perspektivikus tér „virtuális” kamera alkalmazásával, azaz miután hátratoltuk a vetítés középpontját, hogy minden potenciális árnyékvető a kamera síkja elé kerüljön.

A gyakorlatban a virtuális kamera használata gyenge minőségű árnyékokhoz vezet [Koz04]. A valódi kamerához közeli tárgyak kisebb területet kapnak a shadow map-ben. A kamera hátratolásával a parametrizáció a sima shadow map-ekéhez degradálódik. Egy új probléma is felmerül a virtuális kamerával; az, hogy hogyan tudjuk minimalizálni a hátratolás mértéket. Stamminger és társainak megoldása konvex burok számolását és 3D testeken végzett Bool műveleteket tartalmaz. Mindez számításigényes lehet, valamint a shadow mapping elveszti egyszerűségét és eleganciáját. Egy további szépséghibája ennek a megoldásnak, hogy egy animáció során hirtelen változások állnak be az árnyék minőségében, amikor egy objektum kikerül vagy bekerül a potenciális árnyékvetők közé.

Kozlov a virtuális kamera megoldás helyett egy speciális projekciós mátrix alkalmazását javasolja a kamera mögötti potenciális árnyékvetők problémájára. Ha megnézzük az 11. ábrán a poszt-perspektivikus teret és tekintjük olyan fényforrás esetén, mely a kamera mögött van, akkor láthatjuk, hogy a probléma az, hogy a sugárnak a fényforrásból kellene kiindulnia, érintenie az 1-es pontot, kimennie mínusz végtelenbe, bejönnie plusz végtelenből, majd sorra a 2-es, a 3-as és a 4-es pontokon kellene átmennie. Szerencsére poszt-perspektivikus térben lehetőség van olyan

projekciós trükkökre, melyek a szokásos világ-térben nem lehetségesek [Koz04]. Felírható olyan projekciós mátrix, mely megfelel ennek a „képtelen” sugárnak: a közeli vágósíkot egy negatív, a távoli vágósíkot pedig egy pozitív értékre kell állítani.



14. ábra: Egy inverz projekciós mátrix. A fény a távoli vágósík irányába néz.

Legyen

$$| Z_n | = | Z_f | = a$$

Az inverz projekciós mátrixot ugyanúgy állítjuk elő, mint a szokványos projekciós mátrixot:

$$\begin{bmatrix} c & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & Q & 1 \\ 0 & 0 & -QZ_n & 0 \end{bmatrix} \rightarrow \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 \\ 0 & 0 & \frac{1}{2}a & 0 \end{bmatrix}$$

ahol

$$Q = \frac{Z_f}{Z_f - Z_n} = \frac{a}{a-(-a)} = \frac{1}{2}$$

A mátrixot sormátrix formában adtuk meg. Tehát a képlet a transzformált z koordinátákra, amelyek a shadow map-be kerülnek:

$$Z_{psm} = Q \times \left(1 - \frac{Z_n}{Z}\right) = \frac{1}{2} \times \left(1 + \frac{a}{Z}\right)$$

$Z_{psm}(-a) = 0$. Ha Z tart a mínusz végtelenbe, Z_{psm} tart az $\frac{1}{2}$-hez. $Z_{psm}(a)=1$, és ha Z tart a plusz végtelenbe Z_{psm} ismét az $\frac{1}{2}$-hez tart. Ezért a sugár minden ponton a helyes sorrendben halad át, és nincs szükség virtuális hátratolásra, ahhoz, hogy minden potenciális árnyékvető a shadow map-be kerüljön.

Az egyetlen hátránya az inverz projekciós mátrix megoldásnak, hogy nagyobb pontosságra van szükségünk a shadow map-ben. A gyakorlatban, tipikus esetben egy 24 bites fix-pontos mélységpuffer elegendő. [Koz04]

A perspektivikus shadow map-nek egy további hátránya az, hogy a z értékek eloszlása is megváltozik, így az önárnyékolás-probléma kiküszöbölésére egy konstans eltolás alkalmazása alkalmatlannak bizonyulhat. A kamerához közeli árnyékoknál kisebb eltolásra lenne szükség, mint a távolabbiaknál. Egy GPU-n futó implementáció esetén egy lehetséges megoldás erre a shadow map-be írandó értékek linearizálása a pixel shader-ben.

Fénytér perspektivikus shadow map-ek

A perspektivikus shadow mapping úgy javít az árnyékok minőségén, hogy az aktuális nézőpont perspektivikus transzformációjával betorzítja a shadow map-et. A fénytér perspektivikus shadow map kitalálói [WSP04] ebből az alapötletből indultak ki és próbáltak a módszeren javítani. Míg a perspektivikus transzformáció alkalmas a shadow map betorzítására, nem szükségszerű, hogy ez a kamera perspektivikus transzformációja legyen. Mivel a torzítás célja a shadow map pixelei eloszlásának a megváltoztatása, megfelel egy olyan torzítás, amely elsősorban a shadow map síkját befolyásolja és nem a shadow map-re merőleges tengelyt. Ezért fénytérben határoznak meg egy perspektivikus transzformációt. Az általuk javasolt perspektivikus transzformáció képsíkja mindig merőleges a fény sugaraira.

A fény terét párhuzamos fények esetén a következő módon határozzuk meg: A z tengely legyen a fény-vektorral azonos, csak mutasson az ellentétes irányba. Az x és y tengelyek legyenek a z tengelyre és egymásra is merőlegesek, és az y tengely abba az irányba mutasson, amerre a kamera néz. Fényszórószerű fények esetén ezzel azonos módon definiáljuk a fény-teret, de a fény perspektivikus transzformációja után. A fényszórószerű fényeket ezután a párhuzamos fényekkel azonos módon kezelhetjük. A shadow map torzítására használt perspektivikus transzformáció közeli és távoli vágósíkját két az xy síkkal párhuzamos síkként adjuk meg, melyek közrefogják a potenciális árnyékvetőket. A perspektivikus vetítés középpontjának az x koordinátáját úgy határozzuk meg, hogy vesszük a fény terébe eltranszformált kamera pozíciójának x koordinátáját, az y-t pedig a potenciális árnyékvetők minimum és maximum y koordinátájának átlagából vesszük. A torzítás mértékét az határozza meg, hogy a vetítés középpontjának a z koordinátája milyen messze van a közeli vágósíktól. Abban az esetben, ha a fény és a kamera iránya egymásra merőleges, akkor ennek távolsága optimális esetben $n_{opt} = z_n + \sqrt{z_f z_n}$, ahol z_n és z_f a közeli és távoli vágósíkok távolsága [WSP04]. Amennyiben a kamera a fény felé néz, vagy attól eltekint, n-t meg kell növelni úgy, hogy végtelen legyen, amikor a kamera pontosan a fénybe vagy az ellenkező irányba néz.

Ennek a transzformációnak megvan az az előnye az eredeti perspektivikus shadow mapping transzformációjával szemben, hogy nem változtatja meg a fény irányát, így nincs

szükség a mélység teszt invertálgatására. Továbbá nincs szükség virtuális kamerára és az azzal járó bonyolult és költséges jelenet-analizálásra. A shadow map-be kerülő z étékek eloszlása is sokkal egyenletesebb, mint az eredeti perspektivikus shadow mapping esetén.

Trapéz shadow map-ek

A trapéz shadow map megoldás [MT04] hasonlít a perspektivikus shadow map-re. Ez a technika egy trapézzal közelíti azt, hogy hogyan néz ki a kamera látómezejének a csonka gúlája a fény szemszögéből, majd ezt a trapézt kifeszíti, hogy a shadow map teljes felületét kihasználja.

A trapéz megszerkesztésének a menete a következő:

1. Transzformáljuk a kamera látómezejének a csonka gúláját a fény terébe.
2. Számítsuk ki, hogy hol a csonka gúla középvonala.
3. Keressük meg a csonka gúla 2D konvex burkát.
4. Szerkesszük meg a trapéz alját és tetejét úgy, hogy merőlegesek a középvonalra és érintik a konvex burok szélét. A konvex burkot közrefogó két egyenes közül az a teteje, amely közelebb van a közeli vágósík közepéhez. Külön kell kezelni azt az esetet, amikor a csonka gúla vetülete egy négyszög, és ez a négyszög a gúla első vagy hátsó oldala. Ilyenkor a trapéz az ezt a négyszöget befoglaló legkisebb négyzet lesz.

A trapéz négy csúcsából kiszámíthatjuk a szükséges transzformációra vonatkozó mátrixot. Ezt többféleképpen is megtehetjük. Egy kézenfekvő megoldás forgatás, eltolás, nyírás, skálázás és normalizálás transzformációk alkalmazása. Ehhez 8 4x4-es mátrixot kell kiszámolnunk.

1. Első lépésként eltranszformáljuk az origóba a trapéz tetejét:

$$u = \frac{a+b}{2}$$

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -u_x & -u_y & 0 & 1 \end{bmatrix}$$

2. Elforgatjuk a trapézt úgy, hogy a trapéz teteje és alja párhuzamos legyen az x tengellyel:

$$u = \frac{b-a}{|b-a|}$$

$$R = \begin{bmatrix} u_x & u_y & 0 & 0 \\ u_y & -u_x & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3. Legyen a trapéz két oldalának metszéspontja i. Eltoljuk a trapézt úgy, hogy i az origóba kerüljön:

$$u = iT_1R$$

$$T_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -u_x & -u_y & 0 & 1 \end{bmatrix}$$

4. A következő lépésben nyírást alkalmazunk, hogy a trapéz szimmetrikus legyen az y tengelyre:

$$u = \frac{(a+b)T_1 R T_2}{2}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{u_x}{u_y} & 0 & 1 \end{bmatrix}$$

5. A trapézt felskálázzuk, hogy a teteje egységnyi hosszú legyen és 90°-os szöget zárjanak be az oldalai:

$$u = bT_1 R T_2 H$$

$$S_1 = \begin{bmatrix} \frac{1}{u_x} & 0 & 0 & 0 \\ 0 & \frac{1}{u_y} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6. A trapézt négyzetté alakítjuk:

$$N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

7. A négyzetet eltoljuk úgy, hogy a középpontja az origóba kerüljön:

$$u = cT_1RT_2HS_1N$$

$$v = bT_1RT_2HS_1N$$

$$T_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{\frac{u_y}{u_w} + \frac{v_y}{v_w}}{2} & 0 & 1 \end{bmatrix}$$

8. A négyzetet függőlegesen megnyújtjuk, hogy az egység négyzetet kitöltse:

$$u = cT_1RT_2HS_1NT_3$$

$$S_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{u_w}{u_y} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A trapéz transzformációt ezek a mátrixok szorzatából kapjuk:

$$N_T = T_1 R T_2 H S_1 N T_3 S_2$$

Egy másik lehetséges megoldást Paul Heckbert Fundamentals of Texture Mapping and Image Warping című diplomamunkájában [Hec89] találhatunk négyszögből négyzet leképezés címszó alatt.

Az egyik lényeges tulajdonsága a trapéz transzformációnak, hogy a kamerához közeli részek a távolabbi részek rovására nagyobb területet kapnak a shadow map-ből, így, akárcsak a perspektivikus shadow map-pel, segítségével elkerülhetőek a kamerához közeli árnyékok szélein jelentkező recék (perspektivikus aliasing). A trapéz shadow mapping csak ugyanabban az esetben segít, mint amikor a perspektivikus shadow mapping is. Abban az esetben, ha a fényforrás iránya a kamera nézetének irányával párhuzamos vagy ahhoz közeli, az árnyékok minősége a sima shadow map-ekéhez degradálódik. Amennyiben a trapéz teteje nagyon kicsi az aljához képest, a torzítás mértéke nagyobb lehet a kívántnál, azaz ebben az esetben túlmintavételezés lép fel a kamerához közeli árnyékokon, miközben a távolabbi objektumoknak nem jut elég terület a shadow map-en. Ennek korrigálására Martin és Tan egy nagyobb trapézt szerkesztenek, amely magában foglalja a kamera látómezejének csonka gúlája alapján szerkesztett trapézt. Tegyük fel, hogy egy bizonyos távolságon belül szeretnénk jó minőségű árnyékokat, jelölje ezt a távolságot a közeli vágósíkhoz képest δ. Legyen P_L egy pont a látómező középvonalán, a közeli vágósíktól δ távolságra levő pont. A nagyobb trapézt úgy szerkesztik meg, hogy P_L a shadow map 80%-nál levő vonalára képződik le.

A trapéz shadow map transzformációs mátrixának a célja az, hogy az objektumok vertexeinek x és y koordinátáit transzformálja el, viszont ez a transzformáció a z értékeket is befolyásolja, így z-nek más az eloszlása a shadow map különböző részein. Emiatt az önárnyékolás-probléma kiküszöbölésére egy konstans eltolás nem fog megfelelően működni, mivel a szükséges eltolás mértéke függ az x és y-tól. A logikus megoldás erre az, hogy a z-t nem transzformáljuk, azaz a fény terében marad. Egy GPU-s megvalósításban azonban kénytelenek

vagyunk z értékét a pixel shader-ben w-vel visszaszorozni, különben helytelenül lesznek interpolálva a z értékek a háromszögön belül.

Kaszkádolt shadow map-ek

A perspektivikus shadow mapping-nek és a trapéz shadow mapping-nek is az a nagy problémája, hogy amikor a fény irányába nézünk, minősége a sima shadow map-éhez degradálódik. Sajnos olyan újraparametrizációt, amely ebben az esetben is jól működik, legfeljebb olyat lehetséges felírni, mely nem lineáris. Egy ilyen parametrizáció problémás lenne, mivel a hardver lineárisan interpolál a háromszögek raszterizációjakor. Marad az a megoldás, hogy több shadow map-et használunk. Ennek legkézenfekvőbb módja az, hogy az aktuális nézőpont környékét lefedjük egy shadow map-pel, a kétszer ekkora távolságot egy újabb ugyanekkora shadow map-pel, és így tovább, míg el nem érjük a kívánt látótávolságot. Ezt a megoldást nevezik kaszkádolt shadow mapping-nek. A módszer nagyon praktikus kültéri jelenetek rajzolásához.

A shadow mapping megvalósítása GPU-n

Hardveres támogatás shadow mapping-hez

Hardveres támogatás shadow mapping-hez először a Silicon Graphics Infinite Reality szuperszámítógépében jelent meg. Az első PC-khez gyártott videokártya, mely ezt a technikát támogatta az nVidia GeForce3-as volt, mely 2001-ben jelent meg. A módszer OpenGL alatt az SGIX_shadow és SGIX_depth_texture kiterjesztéseken keresztül (később az ARB bizottság szabványosította ezeket a kiterjesztéseket ARB_shadow és ARB_depth_texture néven), Direct3D alatt pedig egy speciális textúraformátum kiválasztásával elérhető. Ezek csak az nVidia GPU-in állnak rendelkezésre, de a technika megvalósítható bármely GPU-n, mely támogat pixel/vertex shader-eket és nagyobb bitmélységű textúra-formátumokat. Gyakorlatilag minden napjainkban eladott új PC-ben levő GPU tudja ezeket. Az ATI GPU-in elérhető két z-puffer textúraformátum, DF16 és DF24, itt ezeket célszerű használni. Az nVidia GPU-iban levő shadow mapping támogatás egyik előnye, hogy „ingyen” van a bilineáris percentage closer filterezés, míg shader programokban megvalósított shadow mapping-nél ez extra pixel shader utasításokat igényel, és így a sebesség rovására megy. A shader-ekben való megvalósítás esetén viszont több a szabadságunk, és egyéb filterezési módszereket is használhatunk.

Érdemes megjegyezni, hogy már korábbi videokártyákon is meg lehetett valósítani a shadow mapping-et az alfa-tesztelés és projektív textúrázás segítségével, bár ezzel a módszerrel csupán 8 bites mélység-puffert használhattunk, mely legfeljebb egy objektumon belül vetett árnyékok megvalósítására elég.

Az újabb ATI GPU-kban (pl. Radeon x1300, x1600, x1900) megjelent egy új shader utasítás, a fetch4, amelynek segítségével jelentősen fel lehet gyorsítani a percentage closer filterezést [Isi06].

A shadow mapping implementációja

A shadow mapping algoritmust Direct3D9 alatt implementáltuk HLSL-ben és C++-ban. HLSL-ben a shader íródott, míg C++-ban a keretrendszer. A programhoz felhasználtuk a DirectX SDK példaprogramjai közt található poligon alapú szövegkiírót. A program megköveteli, hogy a GPU támogassa minimum a 3.0-ás shader modellt.

Egy részlet, amire külön oda kell figyelni egy Direct3D9 implementáció esetén az, hogy ott a textúra-koordináták a texel bal felső sarkát címzik. Emiatt a shadow map-ből való olvasáskor egy fél texel-nyi eltolást kell alkalmazni. Mivel a textúra-koordináták 0 és 1 között fedik le a teljes textúrát, így a textúra méretét is át kell adnunk a shader-nek ahhoz, hogy a fél texel-nyi eltolással tudjuk címezni a shadow map-et.

Forráskódok

A C++ keretprogram terjedelmére való tekintettel a programnak csak a shader részét közöljük nyomtatott formában. A shader az FX keretrendszert használja és önmagában működőképes az FXComposer 1.8-ban.

```
const float2 SHADOW_SIZE;
#define ZMaxValue 130944/16.0
#define ShadowRange 12000.0

#define ShadowRangeMultiplier (ZMaxValue/ShadowRange)



const bool JITTER=false;
float Bias = 0.01;
bool PCF=false;
bool bFlip=false;

const float JITTER_AMOUNT = 1.0;
float4 ClearColor = {0, 0, 0, 0};
```

```
float ClearDepth
<
       string UIWidget = "none";
> = 1.0;


float4 ShadowClearColor = {ZMaxValue, ZMaxValue, ZMaxValue, ZMaxValue};

float4x4 WorldITMatrix : WorldInverseTranspose <string UIWidget="None";>;
float4x4 WorldViewProjectionMatrix : WorldViewProjection <string UIWidget="None";>;
float4x4 ViewProjectionMatrix : ViewProjection <string UIWidget="None";>;
float4x4 WorldMatrix : World <string UIWidget="None";>;
float4x4 ViewIMatrix : ViewInverse <string UIWidget="None";>;
float4x4 WorldViewITMatrix : WorldViewInverseTranspose <string UIWidget="None";>;
float4x4 WorldViewMatrix : WorldView <string UIWidget="None";>;
float4x4 ViewMatrix : View <string UIWidget="None";>;
float4x4 ViewITMatrix : ViewInverseTranspose <string UIWidget="None";>;


float4x4 LightViewMatrix : View
<
    string frustum = "light0";
>;

float4x4 LightProjectionMatrix : Projection
<
       string frustum = "light0";
>;


float3 LightColor : Diffuse = {1.0, 1.0, 1.0};
float3 AmbientLightColor : Ambient = {0.40, 0.395, 0.38};
float3 SurfaceColor : Diffuse = {1.0, 1.0, 1.0};

float DiffuseFactor = 1.0;
float SpecularFactor = 0.05;
float SpecularPower : SpecularPower = 12.0;

texture DiffuseTexture : Diffuse0;

sampler DiffuseSampler = sampler_state
{
       texture = <DiffuseTexture>;
       AddressU  = WRAP;
```

```
	AddressV  = WRAP;
	MIPFILTER = LINEAR;
	MINFILTER = ANISOTROPIC;
	MAGFILTER = LINEAR;
};


texture ShadowMap : RENDERCOLORTARGET;

sampler ShadowMapSampler = sampler_state
{
	texture = <ShadowMap>;
	AddressU  = CLAMP;
	AddressV = CLAMP;
	MipFilter = NONE;
	MinFilter = POINT;
	MagFilter = POINT;
};


texture NoiseTexture : noise;

sampler NoiseSampler = sampler_state
{
    texture = <NoiseTexture>;
    AddressU  = WRAP;
    AddressV  = WRAP;
    MIPFILTER = NONE;
    MINFILTER = LINEAR;
    MAGFILTER = LINEAR;
};


texture ShadDepthTarget : RENDERDEPTHSTENCILTARGET;


struct VertexFormat
{
	float4 Position     : POSITION;
	float4 UV           : TEXCOORD0;
	float4 Normal: NORMAL;
};


struct RenderShadowMapVertexOutput
{
```

```
	float4 Position	: POSITION;
	float Z		: TEXCOORD0;
};


RenderShadowMapVertexOutput RenderShadowMapVS(VertexFormat IN, uniform float4x4
WorldViewProjectionMatrix)
{
	RenderShadowMapVertexOutput OUT;
	float4 Position = float4(IN.Position.xyz, 1.0);
	float4 LightSpacePosition = mul(Position, WorldViewProjectionMatrix);
	OUT.Position = LightSpacePosition;
	OUT.Z = LightSpacePosition.z;
	return OUT;
}

float4 RenderShadowMapPS(RenderShadowMapVertexOutput IN) : COLOR
{
	if(bFlip)
		return float4(-IN.Z*ShadowRangeMultiplier, 1, 1, 1);
	else
		return float4(IN.Z*ShadowRangeMultiplier, 1, 1, 1);
}

struct VertexOutput
{
	float4 HPosition	: POSITION;
	float3 LightVec	: TEXCOORD0;
	float3 WNormal	: TEXCOORD1;
	float3 Wview	: TEXCOORD2;
	float4 LP		: TEXCOORD3;
	float2 UV		: TEXCOORD4;
};


VertexOutput RenderSceneVS(VertexFormat IN, uniform float4x4
ShadowViewProjectionMatrix)
{
	VertexOutput OUT;
	float4 Pos = float4(IN.Position.xyz, 1.0);
	float4 Pw = mul(Pos, WorldMatrix);
	OUT.WNormal = mul(IN.Normal, WorldITMatrix).xyz;

	float4 Pl = mul(Pw, ShadowViewProjectionMatrix);  // "P" in light coords

	OUT.LP = Pl;				// ...for pixel-shader shadow calcs
```

```
	OUT.WView = normalize(ViewIMatrix[3].xyz - Pw.xyz); // world coords
	OUT.HPosition = mul(Pos, WorldViewProjectionMatrix);// screen clipspace coords
	OUT.LightVec = float3(-LightViewMatrix._13, -LightViewMatrix._23, -
LightViewMatrix._33);
	OUT.UV=IN.UV;
	return OUT;
}


float GetShadow(float4 LP, uniform sampler ShadowMapSampler, uniform float ShadowBias)
{
	float shad;

    float2 nuv = float2(.5,-.5)*LP.xy/LP.w + float2(0.5, 0.5);

	nuv+=0.5/SHADOW_SIZE;

	if(JITTER)
	{
		nuv+=JITTER_AMOUNT*(0.5-tex2D(NoiseSampler, LP*64))/SHADOW_SIZE;
	}

    float shadMapDepth = tex2D(ShadowMapSampler,nuv).x;

    float depth = (LP.z/ShadowRange - ShadowBias)*ZMaxValue;
	if(bFlip)
		depth=-depth;


	if(JITTER)
	{
		depth -= (tex2D(NoiseSampler, LP*64).z-0.5)*0.001*ShadowRangeMultiplier;
	}

	if(PCF)
	{
		float2 weight=frac((nuv)*SHADOW_SIZE+0.5);

		float2 pos1=nuv+float2(0.5, 0.5)/SHADOW_SIZE;
		float2 pos2=nuv+float2(-0.5, 0.5)/SHADOW_SIZE;
		float2 pos3=nuv+float2(0.5, -0.5)/SHADOW_SIZE;
		float2 pos4=nuv+float2(-0.5, -0.5)/SHADOW_SIZE;

	float shadMapDepth1 = tex2D(ShadowMapSampler,pos1).x;
	float shadMapDepth2 = tex2D(ShadowMapSampler,pos2).x;
	float shadMapDepth3 = tex2D(ShadowMapSampler,pos3).x;
```

```
	float shadMapDepth4 = tex2D(ShadowMapSampler,pos4).x;

	 float shad1 = shadMapDepth1>=depth;
	 float shad2 = shadMapDepth2>=depth;
	 float shad3 = shadMapDepth3>=depth;
	 float shad4 = shadMapDepth4>=depth;

		float a=shad1*weight.x+shad2*(1-weight.x);
		float b=shad3*weight.x+shad4*(1-weight.x);
		shad=a*weight.y+b*(1-weight.y);

	}

	if(!PCF)
	{
		shad = shadMapDepth>=depth;
	}

	return (shad);
}


struct ClearVertexFormat
{
	float4 Position	: POSITION;
	float2 UV	: TEXCOORD0;
};

struct ClearVertexOutput
{
	float4 Position	: POSITION;
};


ClearVertexOutput ClearVS(ClearVertexFormat IN)
{
	ClearVertexOutput OUT;
	OUT.Position=IN.Position;
	return OUT;
}


float4 ClearPS() : COLOR
{
	return float4(ZMaxValue, ZMaxValue, ZMaxValue, ZMaxValue);
}
```

```
float4 RenderScenePS(VertexOutput IN) : COLOR
{
	float3 N = normalize(IN.WNormal);
	float3 V = normalize(IN.WView);
	float3 L = normalize(IN.LightVec);
	float3 H = normalize(V + L);

	float h_dot_n = dot(H, N);
	float l_dot_n = dot(L, N);

	float4 LightingVector = lit(l_dot_n, h_dot_n, SpecularPower);

	float3 Ambient = SurfaceColor * AmbientLightColor;
	float3 Diffuse = LightingVector.y * SurfaceColor * LightColor;
	float3 Specular = LightingVector.z * SpecularFactor * LightColor;

	float Shadowed = GetShadow(IN.LP, ShadowMapSampler, Bias);

	if(JITTER)
		Shadowed*=(1-(max(((1-LightingVector.y)-0.8), 0)*5));

	float4 res = float4(tex2D(DiffuseSampler, IN.UV) * (Shadowed*Diffuse+Ambient) +
Shadowed*Specular, 1);
	return res;
}


technique main
<
	string Script =	"Pass=ClearShadowMap;"
			"Pass=RenderShadowMap;"
			"Pass=RenderScene;";
>
{
	pass ClearShadowMap
	<
		string Script =	"RenderColorTarget0=ShadowMap;"
				"RenderDepthStencilTarget=ShadDepthTarget;"
				"RenderPort=light0;"
				"Draw=geometry;";
	>
	{
		CullMode = None;
		AlphaBlendEnable=false;
```

```
            ZEnable = false;
            ZWriteEnable = false;

            VertexShader = compile vs_2_0 ClearVS();
            PixelShader = compile ps_2_0 ClearPS();
      }

      pass RenderShadowMap
      {
            AlphaBlendEnable=true;
            SrcBlend=one;
            DestBlend=one;
            BlendOp=min;

            ZEnable = false;
            ZWriteEnable = false;

            VertexShader = compile vs_2_0 RenderShadowMapVS(mul(WorldMatrix,
mul(LightViewMatrix, LightProjectionMatrix)));
            PixelShader = compile ps_2_0 RenderShadowMapPS();
      }

      pass RenderScene
      {
            CullMode = CCW;
            AlphaBlendEnable=false;
            ZEnable = true;
            ZWriteEnable = true;
            ZFunc = LessEqual;

            VertexShader = compile vs_3_0 RenderSceneVS(mul(LightViewMatrix,
LightProjectionMatrix));
            PixelShader = compile ps_3_0 RenderScenePS();
      }
}
```

Konklúzió

Bemutattuk a shadow mapping algoritmust, a problémáit és a rájuk létező különféle megoldásokat. Az áttekintést olyan megoldásokra korlátoztuk, melyek hatékonyan megvalósíthatóak mai modern GPU-kon.

A különféle módszerek áttekintése és ezek egy részének implementálása után az alábbi következtetésekre jutottunk:

- Lebegőpontos ábrázolás használata a shadow map-ben nem optimális.
- A két nagy videó-chip gyártó GPU-ira némileg különböző implementáció az optimális. Az nVidia GPU-in érdemes használni a hardveres shadow mapping támogatást. Az AMD/ATI GPU-in a DF16 textúra-formátumot célszerű használni. Egy olyan implementáció, mely mindkét cég termékein fut lehetséges, de nem optimális. Az AMD/ATI GPU-in az adott jelenet jellegétől függően a bilineáris PCF (percentage closer filterezés) helyett a számítási igénye miatt lehet, hogy egyszerű PCF-et érdemes alkalmazni.
- Megerősítettük azt az állítást, miszerint nagyobb kültéri jelenteket csak több shadow map-pel lehet optimálisan lefedni.

Érdekes kutatási területnek ígérkezik a szórásnégyzet shadow map javítása vagy egy olyan módszer kikísérletezése, amely ugyanazt éri el a szórásnégyzet shadow map problémái nélkül.

Függelék

Szómagyarázat

aliasing: Statisztikában, jelfeldolgozásban, számítógépi grafikában és kapcsolódó területeken, jeleknél, amelyek térben vagy időben folytonosak, mintavételezés szükséges, és a minták halmaza sose csak az eredeti jelre egyedi. A többi jelet, amely ugyanazokat a mintákat eredményezheti (vagy eredményezi) az eredeti jel alias-ainak nevezik.

alpha-teszt: Egy kirajzolandó pixel eldobása vagy megtartása a pixel pipeline-ból érkező alfa érték függvényében.

anizotropikus filterezés: Olyan filterezési mód, mely figyelembe veszi a kirajzolt felületnek a nézőponttal bezárt szögét, több mintát vesz, amikor egy felület élesebb szögben látszik.

bilineáris filterezés: A textúra-filterezés legegyszerűbb formája a bilineáris filterezés. A kirajzolt pixel közepéhez legközelebb eső egyetlen texel mintavételezése helyett, a bilineáris filterezés a négy legközelebbi texel-t mintavételezi, és veszi az értékek súlyozott átlagát.

GPU: Graphics Processing Unit. Grafikai processzor, video-chip-ként is ismert. A GPU elnevezést az nVidia nevű gyártó kezdte el használni, miután programozhatóvá váltak ezek a processzorok.

mip-map: Ugyanannak a textúrának több különböző méretű változata. Egy textúrázott 3D objektum kirajzolása során annak függvényében választjuk ki a használt méretet, hogy az milyen távol van a kamerától, illetve, hogy milyen szögben látszik. A mip betűszó, a latin „Multi in Pavro”-ból származik, melynek jelentése „sok kis helyen”.

önárnyék: Az önárnyék a felület normálvektora és a megvilágítási irány közti szögtől függ, és akkor figyelhetjük meg, ha a felület nem a fény irányába néz. A dolgozat során önárnyékolási hiba néven hivatkozott probléma esetén nem a szó eredeti értelmében vett önárnyékról van szó. A probléma tárgyalását lásd az első fejezetben.

renderelés: Egy magas szintű objektumalapú leírás leképezése egy megjeleníthető grafikai képpé.

shader: Egy GPU-n futó rövid program, amely egy felület árnyalásáért felel.

shadow map: A fényforrás nézőpontjából renderelt mélység-puffer, melyet a shadow mapping során használunk.

texel: Texture element. Egy kétdimenziós textúra legkisebb eleme.

textúra: Egy bitmap alapú kép, mely renderelt 3D képeken kerül rá objektumokra, hogy extra részleteket szolgáltasson.

trilineáris filterezés: A trilineáris filterezés hasonlóan működik a bilineáris filterezéshez, de kezeli a mip-map határokon jelentkező szépséghibákat úgy, hogy bilineáris mintát vesz a két legközelebbi mip-map szintről (összesen pixelenként 8 textúra-mintát). Az eredményül kapott textúrák élesből homályosba mennek át, amint elterülnek a nézőtől. A trilineáris filterezés nem tesz semmit annak érdekében, hogy a textúrák kevésbé látszanak homályosnak.

Köszönetnyilvánítás

Szeretnék köszönetet mondani a témavezetőmnek Schwarcz Tibornak nem mindennapi türelméért, Katsumi Tadamurának segítőkészségéért, és Horváth Zoltánnak, akitől shadow mapping témában folytatott beszélgetéseink során rengeteget tanultam.

Irodalomjegyzék

[Cro77] F. C. Crow: Shadow Algorithms for Computer Graphics, Proceedings of Siggraph 1977, 242.-248. oldal

[Wil78] Lance Williams: Casting Curved Shadows on Curved Surfaces, Proceedings of Siggraph 1978, 270.-274. oldal

[HN85] J. C. Hourcade és A. Nicolas: Algorithms for Antialiased Cast Shadows, Computers & Graphics 9, 259.-265. oldal, 1985

[Coo86] R. L. Cook: Stochastic Sampling in Computer Graphics, ACM Transactions on Graphics 5, 51.-72. Oldal, 1986

[RSC87] W. T. Reeves, D. H. Salesin, R. L. Cook: Rendering Antialiased Shadows with Depth Maps, Proceedings of Siggraph 1987

[Hec89] Paul S. Heckbert: Fundamentals of Texture Mapping and Image Warping, diplomamunka, University of California at Berkeley, 1989

[WPF90] A. Woo, P. Poulin, és A. Fournier: A Survey of Shadow Algorithms, IEEE Computer Graphics and Applications Volume 10, Issue 6, 13.-32. oldal, November 1990

[Woo92] Andrew Woo: The Shadow Depth Map Revisited, Graphics Gems III, 338.-342. oldal, AP Professional, Boston, 1992

[WM94] Yulan Wang, Steven Molnar: Second-Depth Shadow Mapping, UNC-CS Technical Report TR94-019, 1994

[WH98] Wolfgang Heidrich és Hans-Peter Seidel: View Independent Environment Maps, 1998 Siggraph/Eurographics Workshop on Graphics Hardware, 39.-46. oldal

[FFBG01] R. Fernando, S. Fernandez, K. Bala, D. Greenberg: Adaptive Shadow Maps, Proceedings of Siggraph 2001, 387.-390. oldal

[Kil01] Mark J. Kilgard: Shadow Mapping with Today's OpenGL Hardware, CEDEC 2001 prezentáció fóliái

[TQJN01] K. Tadamura, X. Qin, G. Jiao, és E. Nakamae: Rendering optimal solar shadows with plural sunlight depth buffers, The Visual Computer 17, 2, 76.-90. oldal, 2001

[BAS02] Stefan Brabec, Thomas Annen és Hans-Peter Seidel: Shadow Mapping for Hemispherical and Omnidirectional Light Sources, 2002

[BAS02B] Stefan Brabec, Thomas Annen és Hans-Peter Seidel: Practical Shadow Mapping, Journal of Graphics Tools, 7. kötet, 4-es szám, 9.-18. oldal, 2002

[SD02] M. Stamminger, G. Drettakis: Perspective Shadow Maps, Proceedings of Siggraph 2002, 557.-562. oldal

[WE03] D. Weiskopf, T. Ertl: Shadow Mapping Based on Dual Depth Layers, Eurographics 2003

[Koz04] Simon Kozlov: Perspective Shadow Maps: Care and Feeding, GPU Gems, 217.-244. oldal, Addison Wesley, 2004

[MT04] Tobias Martin, Tiow-Seng Tan: Anti-aliasing and Continuity with Trapezoidal Shadow Maps, Eurographics Symposium on Rendering 2004

[WSP04] Michael Wimmer, Daniel Scherzer, és Werner Purgathofer: Light Space Perspective Shadow Maps, Proceedings of Eurographics Symposium on Rendering 2004

[Kin04] Gary King: Shadow Mapping Algorithms, 2004

[LSKSO05] Aaron Lefohn, Shubhabrata Sengupta, Joe Kniss, Robert Strzodka, John D. Owens: Dynamic Adaptive Shadow Maps, Proceedings of Siggraph 2005

[Sch06] Daniel Scherzer: Shadow Mapping of Large Environments, Diplomamunka, Technische Universität Wien, 2006

[DL06] William Donnelly, Andrew Lauritzen: Variance Shadow Maps, 2006 ACM Symposium On Interactive 3D Graphics and Games

[Lau06] Andrew Lauritzen: Variance Shadow Maps, Game Developers Conference 2006 prezentáció fóliái

[Isi06] John R. Isidoro: Shadow Mapping: GPU-based Tips and Techniques, Game Developers Conference 2006 prezentáció fóliái

[LTYM06] D. Brandon Lloyd, David Tuft, Sung-eui Yoon, és Dinesh Manocha: Warping and Partitioning for Low Error Shadow Maps, Eurographics Symposium on Rendering 2006