

Többszínű árnyalás alkalmazása deferred shading esetében

Fridvalszky András,¹ Tóth Balázs¹

¹ Budapesti Műszaki és Gazdaságtudományi Egyetem

Abstract

A népszerű deferred shading alapú képképző eljárások sok előnyük mellett rendelkeznek egy jelentős hátránnyal is. Működési elvükből fakadóan nem tudnak együttműködni a hardveresen támogatott többszínű árnyalással. Ennek a problémának a megoldására több megközelítés is található az irodalomban, azonban általában ezek az eljárások jelentős többlet memória felhasználással és sávszélesség igényel rendelkeznek. Az általunk javasolt új eljárás segítségével korlátok közé szoríthatjuk a többlet memória felhasználást úgy, hogy a végső kép minősége megfelel a hagyományos többszínű árnyalással kapott képekkel. Az általunk javasolt módszer egy új G-Buffer struktúrán alapszik, amely a mintákat képpontként láncolt listákban tárolja. Bemutatjuk azokat a módszereket, amelyek segítségével az új struktúrát létrehozhatjuk és a képképzés során felhasználhatjuk. Az új módszer hatékonyságát egy Vulkan alapú grafikus motor segítségével hasonlítjuk össze a klasszikus megoldásokkal.

1. Áttekintés

A deferred shading alapú grafikus eljárások népszerűek a valós idejű háromdimenziós alkalmazások körében, mivel nagyságrendekkel több fényforrás használatát teszik lehetővé, mint a hagyományos forward shading alapú módszerek. Ennek a megközelítésnek azonban hátránya, hogy a GPU által hardveresen támogatott többszínű árnyalás (MSAA) alkalmazását nem támogatják automatikusan. Erre a problémára több megoldás is létezik, de közös negatívumuk, hogy nagy mértékben növelik a megjelenítő memória használatát, illetve sávszélesség igényét. Emiatt manapság inkább az utófeldolgozás alapú árnyaló eljárásokat részesítik előnyben (pl.: FXAA).

Ezek a technikák, ahelyett, hogy magasabb frekvenciával mintavételeznék a képet, megkeresik és árnyalják az éleket. Ez a módszer jellegéből adódóan sokkal gyorsabb, de nem tud minden esetben tökéletes eredményt nyújtani és könnyen életlen képet, vagy például gyors kamera mozgásnál zavaró hibákat eredményezhet.

1.1. Deferred shading

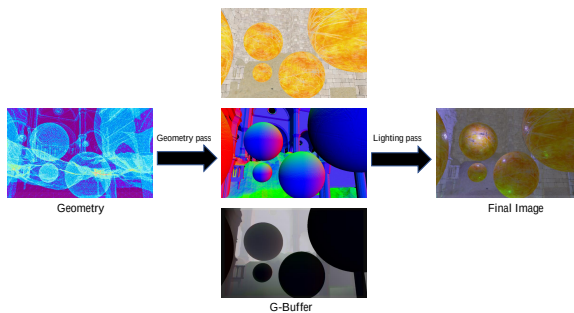
Ezt a technikát először 1988-ban vetették fel, de mind a mai napig elterjedt a számítógépes játékok körében. A célja az, hogy csökkentse, illetve lehetőség szerint megszüntesse a feleslegesen elvégzett munkát az egyes képpontok fényforrá-

sok szerint való árnyalásakor. Ezek a felesleges számítások a kitakart képpontok színezéséből adódnak, amelyek nagyon sok erőforrást igényelhet, például fotorealistikus árnyalás esetén. A problémát deferred shading algoritmus úgy oldja meg, hogy a tényleges árnyalás előtt összegyűjti egy ideiglenes tárolóba (G-Buffer) a majdani színezéshez szükséges adatokat (diffúz szín, normál stb.), minden látható pixelre. Ezt hívják geometry pass-nak. A következő lépésben pedig végrehajtja az árnyalást, amely így, hogy rendelkezésre áll a teljes geometriai információ akár fényforrásonként párhuzamosítható is. Erre lighting pass néven szoktak hivatkozni. Ezzel a megközelítéssel a képképzés bonyolultsága a geometriából adódó képpontok és a fényforrások számának szorzata helyett a látható képpontok és a fényforrások számának összegére csökken.

A deferred shadingnek is több változata van, ezek általában a lighting pass implementációjában térnek el. A legismertebb, amire általában csak simán deferred shading (vagy rendering) néven szoktak hivatkozni, úgy működik, hogy a fényforrásokhoz hozzá rendel egy geometriát, ami a fényforrás "hatóterét" reprezentálja. Ez pontfényforrás esetén egy gömb, aminek a mérete az erősségétől függ, irányfényforrás esetén pedig egy teljes képernyőt lefedő téglalap. A lighting pass során ezek a geometriák kerülnek kirajzolásra. A GPU elvégzi a rasterizálást során azoknak a pixeleknak a kiválasztását, melyek a fényforrás hatni fog. A fragment shader-

ben megtörténik az adott pixelre az árnyalás és az eredményeket egy közös textúrába kell összegezni. Ezen utána el lehet végezni a szükséges utómunkálatokat. A tiled deferred shading nevű változat úgy működik, hogy felbontja a kamera frustumát két (vagy akár három) dimenzióban kisebb darabokra. Egy compute shader a mélység buffer alapján kiválogatja, hogy melyik fényforrás melyik darabokba esik bele és ezeket egy-egy láncolt listába felfűzi. Utána az árnyaláshoz szintén compute shader-t lehet használni, ami minden pixel esetében a megfelelő láncolt listában tárolt fényforrásokon iterál végig.

A deferred lighting nevű változat három lépésből áll. Itt a geometry pass során egy kis méretű G-Buffer készül el, amiben csak a fény összegyűjtéséhez szükséges információk vannak benne (mélység, normál). Ezután következik a fényforrásokból beérkező fény összegyűjtése, hasonlóan a deferred shading-hez, a G-Bufferben lévő adatok alapján. Az utolsó lépés az árnyalás, amihez a teljes geometriát újra ki kell rajzolni, de mivel a teljes beérkező fény minden képpontra ki lett számolva, az árnyalás egyből elvégezhető.



1. ábra. A deferred shading működése.

1.2. Többsimítás élsimítás

A háromszögek raszterizálásakor diszkrét pontokban mintavételezzük a geometriai információkat. A korlátozott mintavételi frekvencia miatt a geometriai primitívek határainál, ahol ugrásszerűen változik az adott mintavételi pontban látható felület fellép az aliasing hatás. Ez a gyakorlatban a recés, lépcsős élekben figyelhető meg melyek zavaróak és elrontják az összképet. A kamera mozgása ezt a jelenséget fel is erősítheti, még látványosabbá téve ezeket a hibákat. Ennek a problémának a megoldására alkalmazhatók az úgynevezett élsimító eljárások (anti-aliasing) amelyek valamilyen módon lokálisan megnövelik a mintavételi frekvenciát a problémás képpontokban.

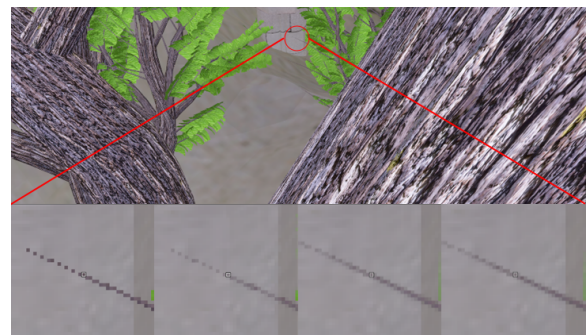
Ezek közül a legegyszerűbb megoldás a kép magasabb felbontásban történő mintavételezése (kirajzolása), majd megfelelő szűrő segítségével történő újra mintavételezése (leskálázása) a cél felbontásra (supersampling). Ez a művelet a probléma gyökerét veszi célba (az alacsony mintavételezési frekvenciát) így az összes vizuális hibát megold-

ja, viszont nagyon nagy az erőforrás igénye ezért gyakorlatban csak ritkán alkalmazzák valós idejű alkalmazások esetén. Ennek egy optimalizált változata a multisample anti-aliasing, amely hasonlóan magasabb felbontáson mintavételezi a képet, de ezt csak ott próbálja megtenni, ahol ténylegesen szükség van rá. Ez az algoritmus a GPU-ban általában hardveresen támogatott.

Ez a megoldás alapvetően úgy működik, hogy minden egyes képponthoz több értéket, úgynevezett mintát tárol el, amelyekhez eltérő képponton belüli pozíció tartozik. Amikor a raszterizáció során egy háromszög metsz egy képpontot, akkor a GPU azt is megvizsgálja, hogy melyik mintákat metszette el. Ezután az árnyalást csak egyszer hajtja végre úgy mintha az a pixel közepén történne, de a kapott eredményt az összes metszett mintához tartozó memóriaterületre kiírja. Ez azt jelenti, hogy minden háromszög-képpont metszés csak egy színezés végrehajtását fogja jelenteni. Ha a teljes kép előállt, akkor az utolsó lépés az egy képponthoz kiírt minták átlagolása és az eredmény kiírása.

Ez általában azt jelenti, hogy egy képpont esetében akkor fogjuk többször végrehajtani az árnyaló algoritmust, ha több háromszög is metszi. Ez két objektum határán is előfordul, ahol általában megjelenik az aliasing így ezeken a helyeken megoldja a problémát, élsimítja a zavaró éleket. Hátránya viszont, hogy ez előfordul egy objektumon belül is, a modellt alkotó olyan háromszögek határán, ahol közel folytonos a felület és itt feleslegesen végez többletmunkát. Ez a többletmunka a látható háromszögek méretétől függően lényegesen kevesebb lehet, mint amennyit a teljes kép túlminiatvételezése jelentett volna. Szintén probléma, hogy csak a geometriai határokkal foglalkozik így az egyéb okokból (például megvilágításból, árnyékok határaiból) adódó aliasingot nem tudja orvosolni.

A 2 ábrán látható, hogy az FXAA nem képes rekonstruálni az apró ág eredeti alakját. Megmarad a darabos struktúra, pusztán kevésbé lesz látható az elmosás hatására. Ezzel szemben az MSAA és a dolgozatomban bemutatott új algoritmus képes megtartani az ág tömörségét a magasabb frekvencián történő mintavételezés segítségével.



2. ábra. Az alulmintavételezés hatása. Balról jobbra: AA nélkül, FXAA, 8x MSAA, javasolt algoritmus.

1.3. Deferred shading többmintás élsimítással

A két technika kombinációja nem lehetetlen feladat, de több problémát is felvet és sok hátránya van. Mindkét módszer önmagában is magas memória használattal és sávszélesség igénytel rendelkezik, amely együtt már érezhetően a teljesítmény rovására megy. Ahhoz, hogy alkalmazhassuk az MSAA-t a teljes G-Buffer-t többszörös mintavételezéssel kell eltárolnunk. Ez az élsimításhoz szükséges minták számával arányosan növeli a méretét. A szokásosan választott 2, 4, illetve 8 minta esetén a G-Buffer memória igénye is 2, 4, illetve 8-szorosa lesz. A mai grafikus kártyák esetében ez a méret növekedés elfogadható, de mobil eszközöknek általában nincsen ekkora grafikus memóriája. További probléma viszont, hogy írni és később olvasni is kell ezt a memóriát, ami sávszélesség problémákhoz vezethet, akár asztali gépek esetében is.

Elteltekintve a memória problémáktól egyéb gondok is felmerülnek. Miután elkészült a G-Buffer végre kell hajtani minden mintára az árnyalást, majd képpontonként átlagolni az eredményeket. Természetesen, ha ezt így tesszük meg, akkor vissza jutunk a supersampling-hez, amit épp elkerülni próbálunk. Erre a problémára megoldást jelent, ha a G-Buffer-ben eltároljuk a lefedettség információt is, ami azt adja meg, hogy egy adott képpontban elhelyezkedő mintákat hány különböző háromszög metszette. Ha ezt csak egyetlen háromszög tette meg, akkor elég egyszer végrehajtani az árnyalást. Ennek a döntésnek az implementálása sem egyszerű, mivel dinamikus elágazást hozhat be a shader kód végrehajtásába, ami teljesítmény veszteséget jelenthet. A stencil buffer használatával megoldható enélkül is, de ez egyéb plusz költségeket fog okozni.

Régebben az is probléma volt, hogy csak modern hardverben voltak meg azok az eszközök melyek segítségével egyáltalán implementálni lehetett a felvázolt megoldást. Ma napság ez már nem gond, de fenti hátrányok továbbra is jelen vannak. Ezért nem is szokás együtt alkalmazni a két módszert, hanem helyette gyorsabb és a deferred shadinggel kompatibilisebb élsimítási technikákat alkalmaznak. Ilyen például az FXAA, MLAA, SMAA is.

Ezek az algoritmusok nem magasabb frekvenciával történő mintavételezést használnak, hanem a rendes felbontású képen végeznek élkeresést majd a megtalált éleket összemossák. Így tehát nem a probléma gyökerét célozzák meg, hanem csak a tünetet próbálják kezelni több-kevesebb sikerrel. Ebből kifolyólag, ha nem elég agresszív az algoritmus, akkor recés élek maradhatnak a képen, ellenkező esetben viszont életlenné válhat. Az eredeti színek is módosulhatnak és minimálisan, de torzíthatják a képet. A gyors mozgás is problémás lehet, mivel az élkeresés sokszor bináris döntést hoz, ami apró, de zavaró ugráláshoz vezethet a képen.

A továbbiakban egy olyan új megoldását mutatunk be, amely lecsökkenti a memória használatot és a sávszélesség igényt, jobb teljesítményhez és nagyobb rugalmassághoz vezetve a klasszikus módszerekhez képest. Az új tech-

nika alapja, hogy nem tárolja feleslegesen az ismételt mintákat, megszünteti a redundáns információt és ezzel javítja a teljesítményt. Mivel a szükséges minták száma képről képre változik a javasolt új módszer adaptívan alkalmazkodik ehhez.

2. Kapcsolódó eredmények

Az alulmintavételezés problémájára több megoldás is fellelhető az irodalomban. Ezek egy része valamilyen utófeldolgozási módszeren alapul. Ilyen például a Reshetov [?] által javasolt morfológiai műveleteken alapuló módszer, amely előre meghatározott mintákat keres a képen és azok alapján összemossa a szomszédos képpontokat. Ennek a megoldásnak továbbfejlesztésére Lottes [?] és Jimenez [?] javasolt változtatásokat.

A Chajdas [?] által javasolt megoldás az egyes képpontok árnyalása során a képponton belüli láthatóság alapján javította az alulmintavételezésből fakadó hibákat.

Másik megközelítése a problémának azon módszerek csoportja, amely a korábbi képek alapján időbeli szűrést alkalmazva csökkenti az alulmintavételezés hatását. Ezen az ötleten alapuló módszert javasolt Marrs [?], amely a túlmintavételezést sugárkövetéssel egészíti ki.

Liktor [?] olyan új struktúrát javasolt a G-Buffernek, amely a minták tulajdonságait hatékonyan képes tárolni. Ezt a struktúrát sztochasztikus képalkotásra és az alulmintavételezés hibáinak csökkentésére egyaránt alkalmazták.

Egy Directx 11-et használó példakódot adott ki az Nvidia amelyben a deferred shading és az MSAA kombinációját vizsgálták. A G-Buffer-t (és a mélység buffert) többmintás textúrák segítségével realizálták. A geometry pass során hagyományos MSAA-t használtak az adatok megszerzésére és tárolására. Az árnyalásra és a minták átlagolására több lehetséges megoldást is bemutattak melyeket az előző fejezetben már felvázoltunk.

3. A javasolt algoritmus

Deferred shading esetén az többszörös mintavételezés alapvető hátránya a minták redundáns tárolásából fakad. A szokásos G-Buffer megvalósítás a pixelenként szükséges adatok tárolására textúrákat alkalmaz. Ennek megfelelően a többszörös mintavételezéshez kellően nagy textúrákat kell választani. Például nyolcszoros túlmintavételezés esetén a szükséges textúrák mérete is nyolcszoros lesz. Ennek a nagy méretű textúrának azonban nagy része felesleges, hiszen a pixelek többségében csak egy-két minta szükséges. Ez a redundáns tárolás további problémákat is okoz. Ellentétben a forward renderinggel - ahol az MSAA sokkal hatékonyabb tud lenni kihasználva azt, hogy a pixelek színének számítását elegendő csak az eltérő mintákra elvégezni, azaz a háromszögek élei mentén kell túlmintavételezni - a deferred shading eljárás esetén az árnyalási lépésben nem áll rendelkezés-

re már geometriai információ. Ennek következtében a szükséges minták meghatározásához ténylegesen túl kell mintavételezni a képet, azaz elveszítjük az MSAA összes előnyét.

Az általunk javasolt új algoritmus ennek a problémának a megoldására új G-Buffer struktúrát alkalmaz, amelyben a G-Bufferet felosztjuk két részre. Az első rész minden egyes képponthoz egy mintát tárol, amely megfelel a klasszikus G-Buffer megvalósításának. Emellett minden egyes blokkban egy mutatót is tárolunk egy olyan lista első elemére, amely a G-Buffer második blokkjában tartalmazza az adott képponthoz tartozó többi mintát.

Az így kialakított struktúra segítségével elkerülhetjük a felesleges redundanciát, azonban megőrizhetünk minden olyan információt, amely az élsimításhoz szükséges. Ennek eredményeként az árnyalási lépésben csak olyan mintákon kell elvégeznünk a számításokat, amelyek valóban hozzájárulnak a végső képhez. A felesleges adatok kiküszöbölésével a G-Buffer számára szükséges memória mérete lényegesen csökkenhet.

A G-Buffer létrehozásának első lépéseként a színtérben található geometriát a szokásos módon raszterizáljuk, a kívánt túlmintavételezésnek megfelelő felbontásban. Fontos azonban, hogy ebben a lépésben célként egy olyan framebuffert alkalmazunk, amely csak egy mélység buffert tartalmaz. A többszínűség élsimításnál szokásos módon a fragmens árnyaló minden egyes háromszög - pixel metszéspontra lefut és eredményül előáll egy vagy több minta. A G-Buffer tartalmát ezek után a szokásos módon állítjuk elő, azonban az előállított mintákat megvizsgáljuk, hogy tartalmazzak-e egy előre meghatározott indexű minta pontot. Amennyiben igen, akkor a minta tartalmát beírjuk a G-Buffer első részébe. Ellenkező esetben az új adatokat a G-Buffer második részében lefoglalt új adatblokkba írjuk egy atomikus számláló segítségével. Az újonnan foglalt adatblokkot az adott képpont láncolt listájához csatoljuk atomi műveletekkel. Ezzel a konstrukcióval a G-Buffer redundancia mentes lesz, egy eset kivételével. Az egyetlen problémás eset, amikor olyan háromszög raszterizálódik először, amely később teljes egészében takarva lesz az adott képpontban. Ebben az esetben feleslegesen kerül be az árnyalásához szükséges információ a G-Bufferbe. Ez a problémát kiküszöbölhetjük úgy, hogy egy z-prepass segítségével kiszűrjük a nem látható fragmenteket.

Az árnyalás során a láncolt listák, vagy akár a teljes G-Buffer egyes elemei egymástól teljesen függetlenül feldolgozhatók, természetesen a fényforrások tulajdonságainak figyelembe vételével. A mi megvalósításunkban az előbbi alkalmaztuk úgy, hogy minden egyes elem esetén az összes fényforrás hatását figyelembe vettük. A végső képpontok színét ezek után a hozzájuk tartozó listák eredményeinek súlyozott átlagaként kapjuk meg, ahol a súlyokat az adott elem által reprezentált minták száma alapján határozzuk meg. Ez a megoldás a fényforrások feldolgozási módja alapján nem optimális, hiszen minden fényforrás a kép összes képpont-

jára hatással van akkor is, ha ez a hatás sok esetben elhanyagolható. Azért választottuk ezt a módszert mégis, mert egyszerűen megvalósítható és a teljesítmény mérések esetén is könnyebben felmérhető a fényforrások számának hatása a módszer teljesítményére. Az általunk megvalósított változat egyszerűen kiegészíthető csempe alapú feldolgozással, amely hatékonyabban tudja kezelni a fényforrások hatását.

4. Implementáció

A javasolt módszer áttekintése után érdemes néhány fontos implementációs részletre is kitérni.

4.1. A G-Buffer

Az általunk készített teszt alkalmazásban fizikai alapú Cook-Torrance árnyalási modellt⁹ alkalmaztunk. Ennek megfelelően az egyes minták árnyalásához szükség van a felület albedójára, normál vektorára, az érdességi, illetve fémességi együtthatójára és az ambiens takarási együtthatóra. Ezen kívül minden mintában szüksége van egy mutatóra is a láncolt lista felépítéséhez.

1. bit	8.	16.	24.	32. bit
	albedo1 (RGB)			metallic1
	albedo2 (RGB)			metallic2
	normal1			
	normal2			
roughness1	roughness2	ao1	ao2	
		pointer1		
		pointer2		

3. ábra. A G-Buffer egy blokkjának felépítése. Minden egyes minta 112 bit helyet igényel.

Annak érdekében, hogy kiküszöböljük a felesleges bitekkel való kiegészítést és az ezzel járó bitműveleteket, két egymást követő blokkot egymásba fűzünk (3. ábra). A hagyományos G-Buffer megvalósítások az ilyen megoldásokat nem teszik lehetővé. Azonos adattartalom mellett minden egyes blokkban 16 bitnyi felesleges adatot kellene tárolnunk mintánként.

1. bit	26.	29.	32. bit
	index	sample count	sample index

1. bit	32.	...	
next pointer	pre-allocated blocks		dynamic blocks

4. ábra. A mutató és a G-Buffer felépítése.

Ahogy azt a 4. ábra felső részén láthatjuk a láncolt listát felépítő mutató három részre tagolódik. A lefedett minták számát 3 bit tartalmazza, további 3 bit adja meg az adott mintának a sorszámát a listán belül. Ez utóbbi annak érdekében szükséges, hogy a mélység bufferből a megfelelő mélység értéket tudjuk kiolvasni. A mutató többi bitjén a láncolt lista következő blokkjának indexét tároljuk.

A G-Buffer teljes felépítését a 4. ábra alsó részén láthatjuk. Fontos kiemelni, hogy az előre lefoglalt blokkok számának meg kell egyeznie a képpontok számával, azonban a dinamikus blokkok száma a rendelkezésre álló memória mérete, a teljesítmény követelmények és a kívánt képminőség alapján tetszőlegesen megválasztható.

4.2. Fényforrások kezelése

Az általunk készített implementáció csak a pontfényforrásokat támogatja, illetve minden egyes minta árnyalása során az összes fényforrás hatását figyelembe vesszük. Ehhez a fényforrásokhoz tartozó adatokat egy uniform bufferben tároljuk és az árnyalási lépésben a buffer minden elemét bejárjuk. Ez a módszer ugyan nem optimális, azonban az egyes ábrák készítése során összehasonlíthatóságát nagy mértékben növeli.

Tekintve, hogy az árnyalási lépés teljesítményére a figyelembe vett fényforrások száma van a legjelentősebb hatással, ezért érdemes a fényforrások térbeli hatását korlátozni. Erre jó lehetőséget ad a csempe alapú deferred shading, amely csempénként képes eltérő számú fényforrást kezelni.

Az általunk javasolt módszer természetes módon képes együttműködni ezekkel a megoldásokkal. Például irányított fényforrások esetén a globális fényforrás listát kell csak lecserélni az árnyalási lépésben, a javasolt módszer többi lépésének változatlanul hagyása mellett.

4.3. Árnyékok és áttetsző anyagok

Az árnyékok és áttetsző anyagok kezelése általános probléma a deferred shadinget alkalmazó rendszerek esetén. Az árnyékok megvalósítására hasonló módszereket alkalmazhatunk, mint a klasszikus forward rendering algoritmusok esetén. A legnépszerűbbek ezek közül az árnyék térképen alapuló megoldások.

Az áttetsző anyagok kezelésére a deferred shading alapú képpalkotás jellegénél fogva alkalmatlan. Az áttetsző objektumokat általában egy külön lépésben adjuk hozzá a képhez úgy, hogy közben figyelembe vesszük a G-Buffer által tartalmazott felületi pontokat is. Az általunk javasolt módszer esetén a klasszikus deferred shading esetén alkalmazható megoldásokat változtatás nélkül tudjuk beépíteni, külön megfontolást nem igényelnek.

5. Eredmények

Az általunk javasolt új algoritmust teljesítmény és memória szükséglet szempontjából vizsgáltuk meg. Annak érdekében, hogy valós képet kapjunk az új módszerről összehasonlíttuk az ábrák készítése nélkül, az FXAA ábrák készítése és a szokásos többszínű ábrák készítése felhasználó megvalósításokkal. Méréseinket több GPU-n is elvégeztük (Nvidia GTX970 és GTX1050).



5. ábra. A méréshez használt színterek.

A teljesítmény mérését három szinten végeztük el, amelyek eltérő geometriai bonyolultságúak. A 5. ábrán balról jobbra az egyes színterek 1 424 145, 8 699 022 és 23 820 168 vertexet tartalmaznak.

Először az összehasonlított módszerek memória igényét vizsgáltuk 1920x1080 képpontos felbontás esetén (1. táblázat). Az általunk javasolt módszer rugalmassága miatt minimális és maximális memória igényt is adhatunk. A minimális memória igényű esetben nem történik ábrák készítése. A maximális memória igény a színtér bonyolultságától függ (6. ábra).

	No AA	MSAA		Proposed algorithm			
		4x	8x	4x		8x	
				Min	Max	Min	Max
G-Buffer	23.73	94.92	189.84	27.69	110.74	27.69	221.48
Z-Buffer	7.91	31.64	63.28	31.64	31.64	63.28	63.28
Total	31.64	126.56	253.12	59.33	142.38	90.97	284.76

1. táblázat. A vizsgált módszerek memória igénye, MByte-ban adva.



6. ábra. Összetett színtér a memória igény vizsgálatához. Nyolc mintás ábrák készítése esetén 115.31 MB memóriát igényel, négy mintás ábrák készítése esetén 75.01 MB-ot. A nyolc mintás ábrák készítése esetén a memória igény kisebb mint a szokásos négy mintás MSAA megvalósítások esetén.

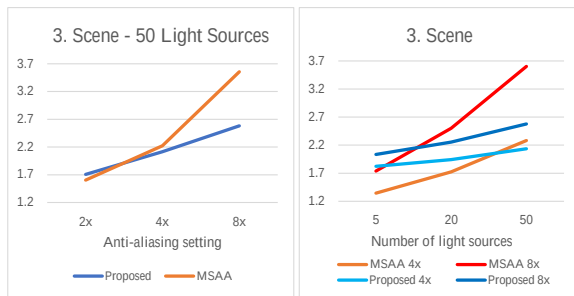
A memória igény mellett az ábrák készítése algoritmusok futási idejét is vizsgáltuk (2. táblázat). Kevésbé összetett színtereken a javasolt algoritmus egyértelműen jobb teljesítményt nyújt úgy, hogy a fényforrások számát növelve ez az előny tovább is nő. Összetett színtereken ez a teljesítmény előny

némileg csökken, de a fényforrások számát növelve egyértelműen jobb eredményt érhetünk el vele, mint a szokásos módszerekkel.

	MSAA	Proposed algorithm	
	4x	4x	8x
1. scene – 5 light sources	5.2648	3.8092	4.3629
1. scene – 50 light sources	21.987	12.5951	14.0772
2. scene – 5 light sources	7.9021	8.1721	8.97702
3. scene – 5 light sources	14.069	19.1354	21.3446
3. scene – 50 light sources	33.293	31.7038	38.6841

2. táblázat. Az élsimító algoritmusok futási ideje (ms).

Az élsimításhoz felhasznált minták számának növelésével a javasolt módszer jobban skálázódik, mint a szokásos megoldások (7. ábra bal oldala). Ennek oka az, hogy az új módszer csak olyan mintákon végez árnyalási műveletet, amely valóban hozzájárul a végső képhez. Nagyon összetett színterek esetén problémát jelenthet a teljesítmény szempontjából a Z-prepass, azonban a fényforrások számának növelésével ez a hátrány eltűnik (7. ábra jobb oldala).



7. ábra. Az élsimításhoz használt minták (bal) és fényforrások (jobb) számának hatása az új módszer és az MSAA relatív számítási idejére. Az ábrán szereplő értékek az élsimítás nélküli számításhoz képesti relatív növekedést jelentik.

6. Összefoglalás

A bemutatott új élsimító algoritmus lehetővé teszi az alul-mintavételezésből fakadó hibák kiküszöbölését deferred shading alkalmazása mellett úgy, hogy a lehető legkevesebb memóriát használja fel. Emellett a javasolt módszerhez nem szükséges a szokásos megoldások összetett árnyalási megoldása. A bemutatott rugalmas adatszerkezet segítségével a G-Buffer redundancia mentesen tárolható, illetve a memória igény és az élsimítás minősége testre szabható.

References

1. Alexander Reshetov. Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 109–116. ACM, 2009.
2. Timothy Lottes. Fxaa. *White paper, Nvidia, Febuary*, 2009.

3. Jorge Jimenez, Jose I Echevarria, Tiago Sousa, and Diego Gutierrez. Smaa: enhanced subpixel morphological antialiasing. In *Computer Graphics Forum*, volume 31, pages 355–364. Wiley Online Library, 2012.
4. Matthäus G Chajdas, Morgan McGuire, and David P Luebke. Subpixel reconstruction antialiasing for deferred shading. In *SI3D*, pages 15–22. Citeseer, 2011.
5. Adam Marrs, Josef Spjut, Holger Gruen, Rahul Sathe, and Morgan McGuire. Adaptive temporal antialiasing. In *Proceedings of the Conference on High-Performance Graphics*, page 1. ACM, 2018.
6. Gábor Liktör and Carsten Dachsbacher. Decoupled deferred shading for hardware rasterization. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 143–150. ACM, 2012.
7. Robert L Cook and Kenneth E Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics (TOG)*, 1(1):7–24, 1982.