

k-puffer: Hatékony, memóriabarát és Dinamikus*k*-puffer Framework

Andreas A. Vasilakis, Georgios Papaioannou, Ioannis Fudos, Tag, IEEE

Absztrakt- A mélységben rendezett töredékmeghatározás alapvető fontosságú számos képalapú technikánál, amelyek összetett renderelési hatásokat szimulálnak. Ez a feladat idő- és térigénye szempontjából is nagy kihívást jelent a nagy mélységű jelenetek riasztérezésekor. Amikor az alacsony grafikus memóriaigény rendkívül fontos, *kA*-buffer objektíve a legelőnyösebb keretrendszernek tekinthető, amely előnyösen biztosítja a megfelelő mélységi sorrendet az összes generált töredék egy részhalmazán. Bár különféle alternatívákat vezettek be annak érdekében, hogy részben vagy teljesen enyhítsék a kezdeti minőségi műtermékeket, *k*-puffer algoritmus a memória növelése vagy a teljesítmény csökkenése rovására, megfelelő eszközök a memória automatikus és dinamikus kiszámításához legmegfelelőbbé tékekmég mindig hiányoznak. Ennek érdekében bemutatjuk *k*-buffer, egy gyors keretrendszer, amely pontosan szimulálja a viselkedését *k*-puffer egyetlen renderelési lépésben. Két memóriakorlátos adatstruktúra: (i) a *amax-tömb* és (ii) a *max-halom* GPU-n fejlesztettük ki, hogy egyidejűleg karbantartsák a *k*-pixelenkénti legelső töredékek feltárással pixel szinkronizálás töredék selejezését. A memóriabarát stratégiákat tovább vezetik be annak érdekében, hogy dinamikusan (a) csökkentsék az egyes pixelk pazarló memória foglalását alacsony mélységű komplexitású frekvenciákkal, (b) minimalizálják a lefoglalt méretet. *k*-puffer a különböző alkalmazási céloknak és hardveres korlátoknak megfelelően egy egyszerű mélységi hisztogram elemzéssel, és (c) kezeli a helyi GPU gyorsítótárat egy rögzített memória mélység szerinti rendezési mechanizmussal. Végül egy kiterjedt kísérleti értékelést adunk, amely bemutatja munkánk előnyeit az összes korábbihoz képest *k*-puffer változatok a memóriahasználat, a teljesítménnyel költség és a képmínőség tekintetében.

Index feltételek-*k*-puffer, A-puffer, mélység-lehúzás, pixelszinkronizálás, mélységi komplexitás hisztogramja, dinamikus geometria

F

1 IBEVEZETÉS

D EPTH-MEGRENDELVEA töredékek meghatározása az interaktív 3D-s játékok és grafikus alkalmazások számos tetszetős és elfogadható vizuális effektusának kifejlesztésének szokásos lépése. Különféle algoritmusok a fotorealisztikus rendereléstől kezdve, mint például a globális megvilágítás [1], a sorrendtől független átlátszóság az előre, halasztott, térfogati árnyékoláshoz [2], [3], [4] és árnyékoláshoz [5] a térfogat megjelenítéséig és feldolgozásáig. áramlási, molekuláris, haj- és szilárd geometria [6], [7], [8], [9], [10] pontos többfragmentum-feldolgozást igényel interaktív sebességgel.

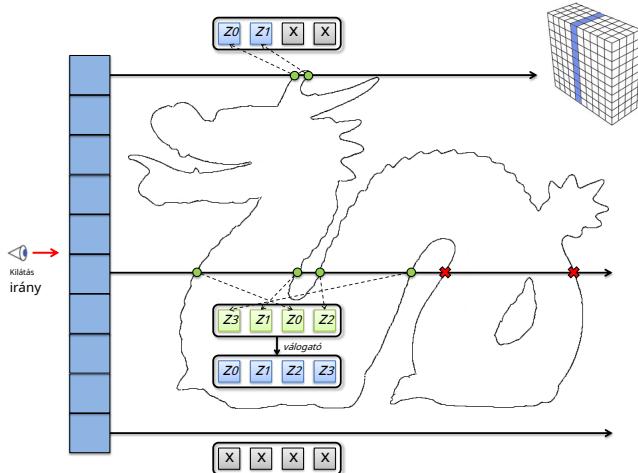
Az elmúlt évtizedben jelentős kutatások folytak a láthatóság-meghatározás problémájának különböző perspektívákból történő kezelésével kapcsolatban, általában tágabb szintre sorolva, hogy az objektumokon/primitíveken (geometriai tér algoritmusok) vagy a generált pixeltöredékeken végeznek-e mélységi rendezést (kép-tér algoritmusok). Az objektum [11] és a nagyobb szemcsésségű, primitív rendezési technikák [12], [13] népszerűsége megnövekedett a kívánatos tulajdonságok egyedi kombinációjának köszönhetően (hibamentes, rendkívül hatékony és könnyen integrálható a szabványos grafikus csővezetékkel). Ezek azonban költséges előfeldolgozási lépést igényelnek a nézetfüggő adatok összeállításához

struktúrák (pl. BSP fa [14]), ami sajnos alkalmatlanná teszi őket dinamikus, rosszabb esetben önmagát metsző geometriából álló jelenetekhez [15].

Ezeket a korlátozásokat elkerülve a GPU-gyorsított pufferek családjába hagyományosan felelős a probléma kezeléséért. tárolása, majd ezt követően válogató, a rendezetlen felületi metszéspontok, nevezetesen töredékek, a geometria pixelszintű mintavételekor jön létre. Az 1. ábra szemléltető példát mutat a töredékgenerálási folyamatra, mint perpixel sugár-felület metszésponti folyamatra. Míg a GPU-gyorsított A-puffer [5] és további változatai, amelyek a rögzített [16] vagy dinamikus [17] lapozott memória kezelést használják ki, a domináns struktúrák több töredék karbantartására.egywagytöbb[18] pixelenként változó hosszúságú linkelt listák, számos alternatívát javasoltak a videomemória túlzott kiosztásának és hozzáférésének költségeinek enyhítésére [19].

kA-buffer [20], valamint annak stencil-útvonalas változata [21] széles körben elfogadott A-puffer közelítések, amelyek képesek rögzíteni a *k*-legközelebb megjelenő töredékekhez a GPU-n pixelenként rögzített méretű vektorok alkalmazásával (lásd még az ötlet szemléltető bemutatását az 1. ábrán). Annak ellenére, hogy az A-puffer megoldásokhoz képest csökkent a memória- és számításgényük, többé-kevésbé mindenkorral vasás-módosítás-írás veszélye(RMWH) akkor keletkezik, amikor a generált töredékeket tetszőleges mélységi sorrendben helyezik be. Ebből a célból rengeteg *k*- a közelmúltban bevezették azokat a puffer-változatokat, amelyek célja a fent említett problémából adódó zavaró pontozott vagy erősen felosztott felületi területek megszüntetése.

- AA Vasilakis és G. Papaaoannou az Athéni Közgazdasági és Üzleti Egyetem Informatikai Tanszékén dolgozik, Görögország. E-mail: {abasilak, gepap}@aueb.gr
- I. Fudos a görögországi Ioannina Egyetem Számítástechnikai és Mérnöki Tanszékének munkatársa. E-mail: fudos@cs.uoi.gr



1. ábra: Egy sor építési folyamatának szemléltetése a4-puffer (kék színnel kiemelve a jobb felső miniatűrön), amikor a sárkány modellt sugározza. Jelentős mennyiségi memória vész kárba azoknál a pixeleknél, amelyek kevesebb, mint 4 töredékek a pixelenként azonos pufferhosszúság előzetes kiosztása miatt.

két [22] vagy többszörös lépés [23], egy kiegészítő A-puffer létrehozása [9], vagy hardveres gyorsítású pixelszinkronizációs mechanizmusok [24], [25] feltárása a további teljesítmény- és memóriaigényekkel, valamint a szükségességgel. speciális hardver.

Fejlesztési és gyártási szempontból az összetett és potenciálisan animált környezetek, ahol nagy az elzáródás, több érdekes pont vagy mélységréteg, nagy kihívást jelenthetnek az optimális pixelenkénti töredék komplexitási határ megtalálása szempontjából (k), hogy megfelelően rögzítse a tervező szándékát korlátozott memóriaköltség mellett. Hagyományosan ezt a feladatot egy iteratív próba és hiba eljárással oldják meg, ahol a felhasználó manuálisan konfigurálja a k , amíg elfogadható vizuális eredményt nem kapunk. Ez a megközelítés azonban bizonyos nézőpontokból szemléltve statikus geometriával működik. Így a felhasználói beavatkozás helyi hatóköre tévesen befolyásolja a jelenet más területeit és megtekintési konfigurációt, ami végül torzítja a kimenetet globális hatókörben.

Ebben a cikkben kiterjesztjük a több töredékes rendereléssel kapcsolatos munkánkat, a k -puffer (**K+B**), először az I3D'14 konferencián mutatták be [26]. Az k -puffer egy hatékony k -puffer keretrendszer, amely kiküszöböli a fent említett teljesítménybeli szűk keresztmetszeteket, memóriaigényeket, képtermékeket és hardverkorlátozásokat.

A többivel ellentében k -puffer alternatívák, amelyek menet közben tárolják és rendezik a generált töredékeket, az A-puffer konstrukcióhoz hasonló gyorsabb stratégiát követünk: k -a legközelebbi töredékeket rendezetlen sorrendben rögzíti, majd egy utólagos rendezési lépés követi, amely átrendezi őket mélységük szerint. Megvizsgálunk egy GPU-gyorsítást *spinlock*stratégia pixel szemaforokon keresztül a valós idejű biztosításhoz

a rendezetlenek szinkronizált felépítések-elülső töredékek (3.1. szakasz). Két korlátos tömb alapú adatstruktúra került bevezetésre a töredékadatokhoz, amelyek lehetővé teszik az alacsony költségű selejtezési tesztet, amely egyidejűleg eldobja a kiugró töredékeket (3.2. fejezet). Végül egy utólagos válogatási folyamatot hajtanak végre, amely a töredékeket a mélységük szerint helyesen rendezi át hibrid megoldással (3.3. szakasz).

Az eredeti algoritmushoz képest ebben a cikkben számos újdonságot és bővítést mutatunk be. Az újonnan bevezetett hozzájárulások a következők:

- Egy új fragmens beillesztési stratégiát vezettünk be, amely enyhíti a fragmentum torlódását a kritikus szakasz elérésekor (3.2.1. szakasz).
- Keretünk könnyen kiterjeszthető a „*legjobb*”, nem feltétlenül áll a legközelebb a nézőhöz, k töredékek (3.2.2. szakasz).
- Egy hisztogram alapú mélységi összetettségi határ (k) becslést használnak annak elkerülésére, hogy a felhasználókat azzal a nem intuitív feladattal delegálják, hogy kézzel állítsák be a krendkívül összetett és potenciálisan dinamikus környezetben (4.2. fejezet). Tudomásunk szerint ez az első k -puffer megvalósítás a szükséges tárhely dinamikus és precíz kiosztásával.
- A túlméretezett helyi GPU gyorsítótárak problémáját a töredékek rendezése során a rögzített maxarray adatszerkezettel történő töredéklevágással oldják meg (4.4. fejezet).
- Egy alapos kísérleti tanulmányt mutatunk be, beleértve a mélységi peeling módszerekkel való összehasonlítást, valamint két közelmúltbeli verseny k -puffer implementációk (5. fejezet).

Az átfogó keretrendszer leírása a shader-szerű pszeudokód és a töredékfeldolgozó folyamat segítségével történik. Kiemeljük a keretrendszerünk jellemzőit és kompromisszumait, rámutatva a megvalósítás részleteire és a könnyű módosításokra, amelyek segítségével el lehet dönten, hogy egy adott környezetben melyik csővezeték-alternatívát alkalmazzuk. A cikk felépítése a következő: A 2. rész részletes áttekintést nyújt a technika állásáról. A 3. rész bemutatja fő keretrendszerünk algoritmikus részleteit. A 4. szakasz leírja, hogy a javasolt folyamat hogyan bővül a pontos memória foglalás és a dinamikus támogatás érdekében megfoghatóra. Az 5. szakasz kiterjedt összehasonlító eredményeket nyújt több több töredékes megjelenítési alternatívához. Végül a 6. rész következtetéseket és jövőbeli kutatási irányokat kínál.

2 RFELAJZOTTWORK

A képernyőtérben való láthatóság meghatározásának problémáját az elmúlt két évtizedben alaposan vizsgálták a számítógépes grafika kutatói. Számos többmenetes képalapú technikát fejlesztettek ki a GPU-n a töredékek láthatósági sorrendjének feloldására [19], amelyek célja, hogy többé-kevésbé minimalizálják a töredékek pontos kibontásához szükséges számítási költségeket és memória foglalást.

$részhalmazs=\{1, \dots, k\}$, $k \leq n$ az összes képpontonként generált töredékből n . Ezeket a technikákat két nagy kategóriába soroljuk a töredékminták száma alapján, amelyeket jelölünk f mostantól az egyes algoritmusok egyetlen iterációs lépésben rögzítik.

Memória-korlátlan Az algoritmusok célja a rögzítés összes töredékek pixelenként egyetlen geometriai menetben ($f = n$). A töredékek pixelenként változó hosszúságú adatstruktúrákban tárolódnak a geometriai megjelenítés során, amit egy utólagos rendezési folyamat követ, amely megfelelően átrendezi a töredékeket mélységük szerint.

Másrészről, *emlékezethet kötött* Az algoritmusok minden töredékinformációt képesek feldolgozni egy többmenetes renderelési folyamatban keresztül, amelyet állandó videomemória-költségvetés mellett hajtanak végre. Az algoritmustól függően minden iteráció egy vagy két geometriai lépést hajt végre, hogy garantált mélységi sorrendű töredékköteget vonjon ki ($f \leq k$). Az utóbbi osztály fő előnye a memória túlcordulásmentes viselkedése a megnövekedett számítási igények rovására.

Memória korlátlan módszerek. Az A-puffer [27] volt az első módszer az összes töredék rögzítésére a valós idejű egyidejű konstrukcióval változó hosszúságú linkelt listák pixelenként egyetlen raszterezési lépésben. Ezt követően a tárolt töredékeket mélységrétekeik szerint utolag válogatják. Az atomi memóriaműveletek közelmúltbeli megjelenésével a grafikus hardvereken Yang et al. bevezetett egy tényleges GPU-gyorsított A-puffer implementációt (**ABLL**) [5]. Bár kezdetben a teljes töredéklista összeállítása és összeállítása során a heves versengés és a véletlenszerű memóriaelérés miatt teljesítménybeli szűk kereszmetszetek szenvedtek [16], jelenleg ez a legkiválóbb módszer több töredék feldolgozására. A fő ok az L2gyorsítótár-integráció a GPU-architektúrák legújabb generációjában, ami az atomi műveletek átviteli sebességének jelentős növekedését eredményezte. Egységes [28] és adaptív [1] csempézési stratégiákat javasoltak továbbá a lehetséges töredéktúlcordulási kockázatok kihasználására.

Másrészt a FreePipe [29], egy teljes CUDA-alapú raszterezési folyamat több töredéket karbantart *fixed méretű* pixelenkénti vektorok. Az összes képpont teljes töredékkivonásának biztosítása érdekében a puffer hosszát pontosan be kell állítani a renderelés előtt. A FreePipe modern OpenGL API-k segítségével valósult meg, így elkerülhető a hagyományos grafikus folyamatról a szoftveres raszterezőre való átállás (**AB sor**) [30]. Nagy teljesítménye ellenére nagy és esetenként szükségtelen mennyiségű grafikus memória lefoglalását igényli.

A linkelt listás és a rögzített tömb technikák korlátainak leküzdésére a [17], [31] két geometriai áteresztő A-puffer variációkat vezetett be. Míg ezek a módszerek önbeállítják a memóriafoglalást, hogy pixelenként változó számról töredéket kezeljenek memóriapazarlás nélkül, csak az S-puffer megvalósítása (**ABsb**) [17] CUDA-mentes, és kihasználja a pixeltér ritkaságát.

A töredékek utólagos szétválogatási szakaszával összefüggésben számos megközelítést javasoltak az előadás szűk kereszmetszete többé-kevésbé enyhítésére a nagy mélységű, összetett jelenetek kezelésekor. A szortírozási folyamat szekvenciális jellegének leküzdésére a mélységi rétegek száma alapján Patney és munkatársai egy CUDA-alapú technikát javasoltak. [32] a párhuzamosítás tartományának kiterjesztése az egyes töredékekre. A tárolt töredékek száma alapján a pixelenkénti rendezési algoritmus megváltoztatásának ötlete jelentős előnyököt mutatott [33]. A közelmúltban egy új, regiszter alapú blokkrendezési algoritmust vezettek be, amely jobban kihasználja a GPU memória hierarchiáját [34]. [35], [36] munkái ihlette Vasilakis és Fudos [18] azt javasolta, hogy egyidejűleg tároljanak töredékeket egynél több pixelenként linkelt listában (**ABLL-BUN**), felgyorsítva a következő válogatási fázist. Lindholm et al. [37] két új komponenst mutatott be a helyi GPU gyorsítótárrak kezelésének javítására. Az előbbi minimalizálja a lefoglalt méretet a gyors gyorsítótárban azáltal, hogy az allokációt a pixelmélység összetettségehez igazítja (itt is megtalálható [38], mint visszafelé történő memóriafoglalás), míg az utóbbi a mélységi leválasztáshoz hasonlóan particionálja a mélységi rendezést [39], és kisebb mennyiséggű lefoglalt memóriát hasznosít újra.

Memóriakorlátos módszerek. Az adatszerkezettől függetlenül a metódusok fent említett osztálya (i) memóriatúlcordulástól szenvédt az összes generált töredék tárolásához szükséges korlátlan puffer miatt, és (ii) a teljesítmény szűk kereszmetszete miatt, amelyek akkor lépnek fel, amikor a pixelenkénti töredékek számát meg kell határozni. postsorted jelentősen megnő.

Valószínűleg a legismertebb többmenetes peeling technika alacsony és állandó tárolási igénye miatt az elől-hátul (**F2B**) mélységi peeling [39], amely úgy működik, hogy a geometriát többszörösen rendereli, mindenkorán egyetlen töredékréteget lehántva. Kettős mélységű peeling (**DUPLA**) [40] felgyorsítja a több töredékes megjelenítést azáltal, hogy minden lépésben rögzíti a legközelebbi és a legtávolabbi töredékeket is. A DUAL-t tovább bővítettük úgy, hogy egységesen fürtözött vödörenként két fragmentumot vontunk ki (**KONTY**) [35]. Enyhíteni *Zfighting* A mélyhámlás problémái miatt a közelmúltban számos megoldást vezettek be [18]. A számos optimalizálás (Z-Batch [41], koherens rétegleválasztás [42], objektumlevágás [18]) ellenére azonban rendszeres időközönként többmenetes renderelésre van szükség az összetett jelenetek végrehajtásához, ami lényegében nem képes interaktívan viselkedni.

k-puffer (KB) [20], [43] csökkenti a számítási költségeket azáltal, hogy rögzíti a k-A nézőhöz legközelebb eső töredékek egyetlen geometriai raszterezési lépésben. Azonban érzékeny az RMWH által okozott zavaró villogó műtermékekre a töredékbillesztési frissítések során. Liu et al. kiterjesztette ezt a munkát több menetes megközelítésre (**KB-Multi**) [23] robosztus renderelési viselkedést ér el az alacsony képkockasebesség kompromisszumával. Sőt, Bavoil és Mayers a legtöbb memóriakonfliktust fellépéssel megszüntette *stencil-útválasztásműveletek* egy multi-

minta élsimító puffer (**KB-SR**) [21]. Végül a *memóriaiveszélytudatosmegoldás* (**KB-MHA**) [44] mélységi hibajavító kódoláson alapuló kódolási sémát vizsgálunk, azonban a gyakorlatban nem garantálják minden esetben a helyes eredményeket. A fent leírt módszerek képmínősége nagymértékben függ a primitív térben végzett durva CPU alapú előválogatástól, ami kiküszöböli a rendellenes töredékek érkezését. Több renderelési iteráció is szükséges az A-puffer kimenet biztosításához, mivel a GPU-n korlátozott számú megjelenítési cél található. A várakozásoknak megfelelően ez jelentős teljesítménycsökkenést eredményez. Ezzel szemben Wang és Xe azt javasolta, hogy a bemeneti jelenetet meghatározott számú rétegű komponensekre particionálják, majd külön-külön rendereljék őket, hogy beleférjenek a korlátozott KB-SR pufferméretbe [45]. Ez a séma azonban nem támogatja az animált jeleneteket, és nem különösen alkalmas rendelésfüggő alkalmazásokhoz.

Több mélységű tesztrendszer (**KB-MDT**₃₂), amelyet mind a CUDA [29], mind az OpenGL [22] API-kban fejlesztettek ki, garantálja a helyes mélységi sorrendet a töredékek menet közbeni rögzítésével és rendezésével. 32-bit atomi egész összehasonlítások. A mélység- és színértékek egyidejű frissítésének képtelensége azonban további költséges geometriai átvizsgálást tesz szükségessé. Szerencsére az 64-bites verzió (**KB-MDT**₆₄) [46] jelenleg megvalósítható modern NVIDIA grafikus kártyákon [47]. A 32 és 64 bites verzióból azonban zajos képek készülhetnek, mivel a lebegő mélységértékek konvertálása során elveszik a pontosság.

A mi módszerünkhez hasonlóan Salvi kiterjesztette az eredetitk-puffer a töredezett verseny elkerülésére hardvertudatos pixelszinkronizálással (**KB-PS**) [24]. Ez a módszer azonban csak a Haswell architektúrán alapuló grafikus kártyákkal kompatibilis.

Végül Yu és mtsai. két linkelt lista alapú megoldást javasolt a pontos kiszámításához k-első töredékek [9]. Az elsőnek az az ötlete, hogy rögzítse az összes töredéket úgy, hogy kezdetben egy A-puffert hoz létre linkelt listákon [5], majd egy lépést, amely kiválasztja és rendezi ak-legközelebbi töredékek (**KB-ABLL**). Ugyanezt a stratégiát követte a korábbi munka [48] is, amely adaptív módon tömöríti a töredékadatokat, hogy közelítse az alap-igazság láthatósági megoldást. Másrészt a második megközelítés közvetlenül kiszámítja a mélység szerint rendezett, pixelenként linkelt listákat, elkerülve a szükségtelen A-puffer konstrukciót (**KB-LL**). Annak ellenére, hogy elmeletileg kevesebb tárhelyet igényel, a töredékek ritkán tárolódnak a memóriában, ami további összefüggő memoriablokkok kiosztását okozza.

Az 1. táblázat összehasonlító áttekintést nyújt az összesrőlk-puffer alternatívák a memóriaigény, a renderelés bonyolultsága, a töredékkivonás pontossága és a rendezési szakasz tekintetében.

3 CÉRCFKERETÖTTÉKINTÉS

Javasoljuk k+-puffer, hatékonyk-puffer megvalósítás a GPU-n, amely mentes a következőktől: (i) geometria

raszterezés előtti rendezés, (ii) korlátlan memóriaigény, (iii) RMW memóriakockázatok, (iv) mélységpontos konverziós műtermékek és (iv) speciális hardverbővítések (azaz pixelszinkronizálás, 64-bit atomi műveletek). A legtöbbsel ellentében k-buffer alternatívákat, amelyek menet közben tárolják és rendezik a generált töredékeket, kezdetben tároljuk ak-a legközelebbi töredékek rendezetlen sorozatban, majd egy utólagos rendezési lépés követi őket, amely átrendezi őket mélységük szerint. Szemátor alapú spin-lock mechanizmus biztosítja a pixelenkénti töredékműveletek atomitását a megosztott memóriában. Megvalósítási részletek is rendelkezésre állnak, hogy könnyen átválthatsson a modern architektúrákon elérhető hardveresen megvalósított pixelszinkronizálási megoldásokra (3.1. fejezet). A távoli töredékek vitájának (elfoglaltságának) enyhítésére egyidejűleg végünkselejtezésellenőrző, hogy hatékonyan dobja-e el azokat a töredékeket, amelyek távolabb vannak az összes jelenleg karbantartott töredéktől.

A GPU-ra két tömb alapú adatstruktúra épül fel a legközelebbi képpontonkénti töredékek pontos tárolására: (i) *max-tömb*, egy tömb, ahol a maximális elem minden az első bejegyzésnél van tárolva, és (ii) *maxheap*, egy teljes bináris fa, amelyben az egyes belső csomópontok értéke nagyobb vagy egyenlő, mint az adott csomópont gyermekeinek értéke. Lineáris összetettsége ellenére az előbbi gyorsabban teljesít, mint az utóbbi, ha a probléma mérete kellően kicsi (3.2. fejezet). Végül egy utólagos szortírozási folyamat helyesen rendezi át a töredékeket mélységük szerint egy hibrid megoldás feltárással (3.3. szakasz).

3.1 Spin-Lock (SL)

Képpontonként bináris szemátorokszinkronizálási mechanizmusként használják a kritikus tárolórész töredék-kizárolagos használatát. Figyelembe véve a zárhoz való egyidejű hozzáférés lehetőségét, ami versenykörülményeket okozhat, egy atomerőmű megvalósítása tesztelés és beállításműveletet vezetik be. Általában a hívó szál megkapja a zárolást, ha a régi érték volt 0. Pörgeti az írást 1 a változóra, amíg ez meg nem történik. Az 1. algoritmus bemutatja a spin-lock stratégia megvalósításának egyik módját, amely a tesztet és a pixel shader-beállítást alkalmazza (a színkódolt vonalak figyelmen kívül hagyása).

1. algoritmus Kölcsönös kizárási (textúra t, Pixel p)

```

1:beginFragmentShaderOrderingINTEL(); 2: . zár beszerzése (PS)
beginInvocationInterlockNV(); 3:mígigaz . zár megszerzése (FSI)
csináld 4: . pörgesse, amíg a zár szabadá válik (SL)
    halImageAtomicExchange(t, p,1)akkor . zár beszerzése (SL)
5:   !lépjön be a kritikus . kizárolagos használat
6:   szakaszba;imageStore(t, p, . zár kioldása (SL)
7:   0); szünet; . centrifugálás leállítása (SL)
8:   vége ha
9:vége közben
10:endInvocationInterlockNV(); . zár kioldása (FSI)
    . ahol 'szöveg' és 'szöveg' A PS és FSI implementációkat a kritikus
    szakasz töredék-kizárolagos használatára határozza meg

```

A perpixel szemátorok megjelenítésére egy 32 bites előjel nélküli egész szám textúra van kijelölve belső pixelformátummal, R 32UI. Először is egy teljes képernyős négyes renderelés

(tiszta passz) végrehajtódi a textúra nullákkal történő inicializálásához. Módszerünket továbbfejlesztik az OpenGL-ek *imageAtomicExchange*(texture lock, *ivec2 P, uint V*) függvény, amely atomosan lecseréli az atomi objektum V értékét a P koordinátán lévő texel argumentumra, és visszaadja az eredeti értékét.

Pixel szinkronizálás (**PS**) egy olyan bővítmény, amelyet az Intel a közmúltban vezetett be az IRIS Pro Graphics-hoz, amely olcsó mechanizmust biztosít a kritikus szakaszok töredéktöközsének elkerülésére, és biztosítja, hogy az RMW memóriaműveletek a beküldési sorrendben történjenek [24]. Ezt a trendet követve egy hasonló kiterjesztést fejlesztettek ki, Fragment Shader Interlock néven (**FSI**) [25], csak Maxwell architektúrájú NVIDIA grafikus kártyák támogatják. Keretrendszerünk továbbfejleszthető PS/FSI használatával anélkül, hogy a javasolt folyamatot minimális implementációs módosításokkal átalakítanánk (lásd az 1. algoritmus színes kódját). Vegye figyelembe, hogy a pixelenkénti szemaforok használatának elkerülése csökkenti a memóriaigényt is.

3.2 Töredéktárolás

Egy geometriai renderelés (**bolti bérlet**) kezdetben a képpontonkénti legközelebbi töredékatok rögzítésére szolgál egy 64 bites lebegőpontos 3D tömbpufferben RG 32F belső formátummal (R a szín és G a mélység) és k hossz. Az 1. ábra a k -puffer, amely legfeljebb 400 töredékek (képernyő mérete: $10 \times 10, k=4$).

A forgásának enyhítésére generált töredékek, amelyek nem tartoznak a legközelebbihez, gyors selejtezési mechanizmust hajtanak végre. Az ötlet az, hogy hatékonyan selejtezzen minden bejövő töredéket $f_{\text{én}} \in \{0, \dots, n-1\}$ amelynek azonos vagy nagyobb mélységi értéke van ($f_{\text{én}}.z$) az összes jelenleg karbantartott töredékből, mielőtt megpróbálná megszerezni a szemafort. Vegye figyelembe, hogy én meghatározza a benyújtási sorrendet. Legyen $a_{\text{én}}[:] = f_{\text{én}}[j], j=0 \dots k-1$ jelöli a tartalmát k -puffer, ha töredéket $f_{\text{én}}$ feldolgozásra került. Kezdetben nem dobunk el egyetlen bejövő töredéket sem, amíg a töredéktároló puffer meg nem telik ($\forall i < k$). Ezután eldobjuk az összes töredéket $f_{\text{én}}$ módon, hogy $f_{\text{én}}.z \geq \max\{a_{i-1}[:].z\}$. Másrészt egy töredék azzal $f_{\text{én}}.z < \max\{a_{i-1}[:].z\}$ töredékét helyettesíti a k -puffer a legnagyobb mélységtől kezdve. Ez a stratégia garantálja, hogy a legközelebbi töredékek minden fennmaradnak, mivel:

$$\max\{a_{n-1}[:].z\} \leq \dots \leq \max\{a_{i-1}[:].z\} \leq \dots \leq \max\{a_{k-1}[:].z\} \quad (1)$$

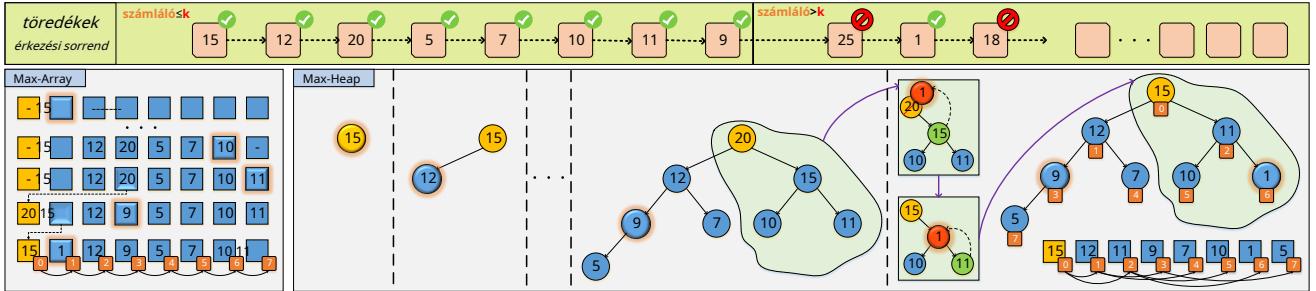
Annak érdekében, hogy a töredékek selejtezését a teljes pixelsor bejárása nélkül érjük el minden bejövő töredéknél, két tömbalapú adatstruktúrát fejlesztettünk ki a GPU-n, amelyek a maximális elemet az első tömb pozícióban tárolják: (i) **max-tömb(K+B-tömb)** és (ii) **maxheap(K+B-Heap)**. Így ez a művelet állandó időben történik. Ennek az ötletnek a megvalósítását a 2. algoritmus mutatja be (a kék színű kód figyelmen kívül hagyása).

A Max-tömb egy olyan tömbnek tekinthető, ahol a legnagyobb mélységtől mindig az első helyen van tárolva, a többi pedig véletlenszerűen helyezkedik el. Amikor egy bejövő töredék szemafort kap, az információt az első üres bejegyzésben tárolja ($O(1)$). Ebben az esetben egy pixelenkénti számláló(A 32 bites előjele nélküli egész textúra belső pixelformátummal R 32UI) indexként használható, és a sikeres beillesztés után növekszik. A képpontonkénti számlálók nullára inicializálódnak a tiszta renderelés során. Ha a tömb megtelt ($\text{számláló} = k$), a legnagyobb mélységtől töredék helyét veszi át. Vegye figyelembe, hogy mivel a selejtező mechanizmus a kritikus szakaszban kívül található, egy további ellenőrzés kötelező a helyes eredmények garantálása érdekében. Ahhoz, hogy a max-tömb konzisztens maradjon egy teljesen kitöltött tömb beszúrása után, megkeressük a legnagyobb mélységtől töredéket ($O(k)$), és cserélje ki az újonnan hozzáadott töredékre (kivéve, ha az utóbbi a legnagyobb). Ez az eljárás költséges atomi műveletek alkalmazása nélkül valósul meg, mivel a fragmentum atomitása garantált.

Ha azonban a probléma mérete növekszik ($k > 16$), A töredékat-adat-információ alternatívaként max-halom adatstruktúrában is karbantartható. A Max-heap egy teljes bináris fa (*alaktulajdonság*), amelyben minden csomópont nagyobb vagy egyenlő, mint az egyes gyermekei (*halom* ingatlan). Max-heap megvalósítható egy egyszerű k -méretű tömb anélkül, hogy helyet foglalna a mutatóknak: Ha a fa gyökere az indexen van 0, majd minden elem az indexnélén $\leq 0, k$ gyermekei vannak az indexknél $2\text{én}+1$ és $2\text{én}+2\text{szülője}$ pedig a címen található $\text{index}_b(i-1, c)$. Mivel az első csomópont tartalmazza a legnagyobb elem, a mag csővezeték, amelyet a max-tömb követ, nem módosul. Mindkét beszúrási művelet egy üres vagy egy teljes kupacba módosítja a halmot, hogy először megfeleljen az alak tulajdonságának, csomópontok hozzáadásával a kupac végéről vagy a kupac gyökér cseréjével ($O(1)$). Ezután a kupac tulajdonság visszaáll a fel- vagy le-halom bejárásával ($O(\log_2 k)$). Mindkét beszúrási funkció pseudokódja megtalálható az eredeti cikkben [26]. A 2. ábra szemlélteti, hogyan működik minden töredéktároló adatstruktúra a $k=8$ számos rendellenes töredéktárolásból épülnek fel és frissítik. Figyelje meg, hogy a rendszer két bejövő töredéket sikeresen eldob, amikor a puffer teljesen megtelt. Megjelenik a max-array és a maxheap csomópontmutatók belső adatreprézentációinak összehasonlítása is.

3.2.1 K felülvizsgálata+B-tömb

Azón a megfigyelésen alapul, hogy a folyamat kezdetben a beillesztést igénylik töredékeket, mielőtt bármilyen selejtezési tesztet végrehajtana, arra törekünk, hogy csökkentsük a töredékek versenyét az első $k-1$ bejövő töredékek. Ez rendkívül hatékony olyan esetekben, amikor k -a puffer elég nagy ahhoz, hogy minden ($n \leq k$) vagy a legtöbb ($k \leq n \leq 2 \cdot k$) a generált töredékekből ($n = \max_p\{f(p)\}$, ahol $f(p)$ a szám



2. ábra: Rendkívüli töredékek tetszőleges sorozatának beillesztési folyamatának áttekintése, amikor (balra) maxarray és (jobbra) max-heap adatstruktúrákkal hasznosulnak. A bejövő töredék minden lépésben világító hatással van kiemelve. Amikor a tömb megtelik, a maximálisan rögzített töredéknél nagyobb értékű (sárga színű) töredékeket hatékonyan eldobja ($f_z=25$ és $f_0=18$).

képpontonként generált töredékek $\text{számap}[x, y]$, többé-kevésbé olyan gyorsan viselkedik, mint az ABSor.

Az egyidejűség minden esetben történő biztosítása érdekében egy pixelenkénti atomszámláló jelzi a következő elérhető címpozíciót a bejövő töredékhez. Teljesítmény szempontjából ezt az ötletet nem javasoljuk a kupacszerkeztnél alkalmazni, mivel költséges halmozanak műveletet kell végrehajtani (a halom minden tulajdonságának biztosítása érdekében), miután a kupac teljesen megtelt. A 2. algoritmusban kékkel szemléltetjük a töredéklejelezési shader felülvizsgálatához szükséges kódfrissítést.

2. algoritmus K+B-tömb-R(Sora, Text, Pixp, Fragf, Intk)

```

1:p.számláló←p.counter+1; 2:          . atomi index növekmény
hp.számláló<kakkor 3:          . fielsőtöredékek
  a[k-p.számláló-2]:=f; 4:          . gyors töredéktár
  más
5:ha fz<a[0].akkor 6:          . töredék selejelezés
  Kölcsönös kizáras(t, p); 7:          . lassú töredéktár (1. alg.)
  vége ha
8:vége ha
. ahol 'szöveg' határozza meg a felülvizsgált töredéklevágási megvalósítást

```

3.2.2 Általánosk+-puffer

Az általánosság elvezetése nélkül a keretrendszerünk módosítható, hogy bármilyen rendezetlen sorozatot rögzítsen a töredékek (nem korlátozva a legközelebbiek). A selejelezés helyett az egyes bejövő töredékek mélységi értékét használnak kritériumként (*f*_{*n*}), akkor eltávolíthatjuk azt a töredékcsoportot, amely a legnagyobb hibát generálja az általános célfüggvényen keresztüli integrációban (*f*_{*n*}) számos egyenlőtlenségi megkötésnek van kitéve $y(x)$. Így a töredékek selejelezési feltétele a 2. algoritmusban általánosítható a következők minimalizálásaként:

$$\min_{y(x) \leq 0, j=1 \dots /} g(f_{\bar{n}}) = h(f_{\bar{n}}) - h(a[0]) \quad (2)$$

$$h(a[0]) = \max\{h(a_{i-1}[:])\}$$

Speciális esetben K+B-Heap, a selejelezési optimalizálás megfogalmazása megegyezik:

$$h(f_{\bar{n}}) = f_{\bar{n}}.z \text{ és } y(p) = k - p.\text{számláló} \quad (3)$$

mivel $a_{\bar{n}}[0]$ minden sikeres fragmens beillesztés után megváltozik, még az elsőtől is $i < k$ azok. Azonban esedékes

arra a döntésünkre, hogy a töredékeket menet közbeni rendezés nélkül tároljuk, olyan alkalmazások, amelyek minden töredékleillesztés után megkövetlik a mélységi sorrend tulajdonságot, mint pl. *Adaptív átlátszóság*[48], nehéz megvalósítani az alapvető adatstruktúra megváltoztatása nélkül.

3.3 Töredékek rendezése

Végül egy rendezési eljárást alkalmaznak a töredékek átrendezésére az egyes pixelekhez a végső kép létrehozása előtt (**megoldani pass**). A rendezetlen töredékek kezdetben egy helyi méretű tömbbe másolódnak/pixelenként a mélységi rendezés végrehajtása előtt, mivel viszonylag gyorsabban lehet írási-olvasási műveleteket végrehajtani a lokális térből, nem pedig a globális grafikus memóriában. Végül a rögzített töredékek száma alapján egy mechanizmus dönti el, hogy melyik rendezési algoritmust alkalmazza az egyes pixelekre [33]. Kvadratikus összetettsége ellenére, *beillesztési rendezés* gyorsabb kis töredékszékencák rendezésére. Amikor a generált töredékek száma növekszik ($\#p>16$), *shell fajta* előnyös. Vegye figyelembe, hogy avisszafelé történő memória foglalás[38] könnyen használható a helyi memória-gyorsítótár tulcsordulási és késleltetési problémáinak kihasználására.

4 MEMORY-TUDATÁBAN EKINYÚJTÁSOK

Célja a GPU memória optimalizálása, amely a *k*-a puffer alapú renderelési keretrendszer lefoglalja a keretpuffereket (a lokális és globális memória tekintetében), keretünket négy új komponenssel bővítettük:

- A grafikus memória foglalás dinamikus és precíz kezeléséhez kezdetben egy folyamatbővítést vezettek be (4.1. fejezet).
- Az értéke/adaptív módon kiszámítható a raszterizált jelenet mélységi eloszlási hisztogramjának elemzésével a megadott alkalmazási célok és a GPU globális memória korlátai alapján (4.2. fejezet).
- A beállított/meghatározotttól függően *k*, a javasolt módszer egy egységes keretrendszernek is tekinthető, amely sikeresen integrál

a Z-puffer funkciói, k -puffer és Abuffer (4.3. fejezet).

- Végül, a helyi GPU-memóriában előforduló túlcordulás és szálcseré-várakozás elkerülése érdekében a K+A B-tömb ötletet is használják a végrehajtáshoz/- mélységi válogatás, állandó mennyiség újrafelhasználása (/) lefoglalt terület (4.4. szakasz).

4.1 Precíz memória foglalás

Mindegyikhez hasonló k -puffer alternatívák hol minden pixelnél ugyanaz, minden K+B-tömb és K+A B-Heap potenciálisan nagy és fel nem használt, előre lefoglalt tárthelykövetelményeket igényel azoknál a pixeleknél, amelyek kevesebb mint töredékek (k -töredéketlenpixel). Például az 1. ábra szemlélteti a pazarlón kiosztott tárolást a 4-puffer egy (felső) képponthoz, amely két töredékből és (alul) egy üres pixelből áll. Vegye figyelembe, hogy az értékek nem igazodik automatikusan a rasszerezzet jelenet alapján, és a felhasználónak előzetesen gondosan kell állítania.

AB ihlettes [17], bevezetünk egy memóriatudatos implementációt két geometriai lépéssel (**K+B-AB_{SB}**). A memória lineárisan szerveződik változó összefüggő régiók minden egyes pixelhez, lehetővé téve minden két javasolt adatstruktúra megvalósítását. A szükséges memóriaterület pontos lefoglalása egy kezdeti geometriai megjelenítés végrehajtásával érhető el (**gróf pass**), amely hardveres okklúziós lekérdezések keresztül összegzi az egyes pixeleket lefedő töredékek számát. Ellentében az AB-vels, ahol minden töredék hozzájárul a pixelenkénti aggregációhoz, a pixel befolyásoló töredékek számát a $\text{kmikor}(p) > k$. minden bejövő töredékhez egy pixelenkénti számláló atomosan növekszik ($p.\text{counterTotal}$). Amikor a számláló értéke eléri a később érkező töredékeket eldobjuk. Ezután a memóriaeltolás keresőtáblája (**utaló passz**) párhuzamos módon kerül kiszámításra, kihasználva a pixeltér ritkaságát. Végül a pixelenkénti számlálók ($p.\text{counter}$) nullára inicializálódnak, hogy irányítsák a következő tárolási fázist.

A memória-gyorsítotárazás, és ezáltal a teljesítmény növelése érdekében a nem üres pixeleket egységes négyzet alakú csempézési stratégia alapján fürtözzük, amely felváltja a kezdeti megvalósítás pixel-oszlop hash funkcióját. Ha további információra van szüksége az algoritmus részleteiről és ennek a lépésnek a shader-megvalósításáról, az olvasók az eredeti AB-ban találhatók a papír [17].

Geometriai rasszerezést alkalmaznak a legjelentősebb töredékek tárolására egy hibrid puffersémában, az egyes pixelekhez kiszámított memóriaeltolásokból kiindulva. Ismerve a töredék számosságát a priori ($p.\text{counterTotal}$), minden képpont hatékonyan kiválaszthatja a töredékeinek max-array vagy max-heap tárolóban való tárolásának leggyorsabb módját. Mivel a max-array szerkezet gyorsabban illeszti be az elemeket, mint a max-heap, amikor a kapacitás nem teljes éskkicsi marad, a következő stratégiát alkalmazzuk: $\text{ha } \text{hf}(p) > \text{kész } > 16$ akkor a max-heap-et választjuk, egyébként a max-array adatstruktúrát használjuk tárolópufferként.

A teljesítmény szempontjából a véletlenszerű memória elérése az összes töredék egyidejű tárolására jelentős szűk keresztsmetszetté válik, szemben az eredeti eglépéses verziókkal, amelyek előnyösek a szekvenciális memóriaterület gyors műveleteiből. Végül, de nem utolsósorban, egy további geometria-megjelenítési lépés szükségessége a tessellációtól függő számítási költséget is növeli.

A hiányzó összetevők a K teljesítéséhez+B-AB_{SB} A pipeline, beleértve az eredetit és a dinamikus változatát (lásd alább), a 3. algoritmusban látható. Vegye figyelembe, hogy az üres() beszúrás és a teljes beszúrás() az absztrakt beszúrási függvények. K+B-tömb és K+ Mindkét függvény B-Heap saját verziója elérhető a forráskódban, amely kiegészítő anyagként szolgál.

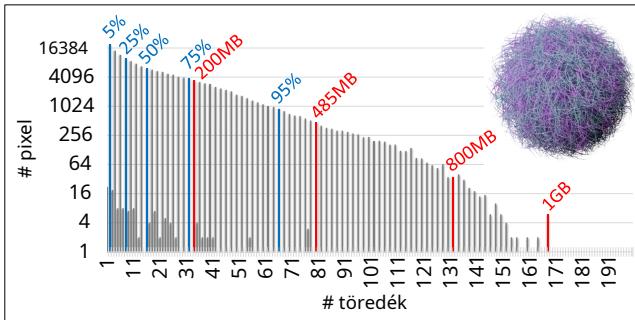
4.2 Maximális rögzített rétegek (k) Előrejelzés

Kézi beállítás/ktetszőleges geometria és megtekintési konfiguráció esetén könnyen a mélység bonyolultságának rossz lefedettségéhez vagy az érték túlbecsüléséhez vezet, a lehetséges kameramozgás vagy az animált/dinamikusan generált geometria miatt. Ez elkerülhetetlenül látható és nézetfüggő műtermékeket vagy rossz memória használatot eredményez.

Itt bemutatunk egy intuitív és automatikus módszert a maximális rétegek előrejelzésére kelfogta a k -puffer, a perpixelmélység komplexitásának hisztogramja alapján. Az ötlet egy további geometriai lépés végrehajtása (vagy könnyen beágyazható a K számláló lépésébe+B-AB_{SB}) a mélységi komplexitás hisztogramjának generálásához a GPU-n, amelyet egy gazdagépoldali folyamat követ, amely minimalizálja a lefoglalt méretet. A különböző alkalmazáscéloknak és GPU-memóriakorlátoknak megfelelően.

Általában a folyamat mögött meghúzódó alapvető tervezési cél, majd a betartandó elvek és szabályok az adott alkalmazástól függnek. Általában meg kell becsülnünk az értéket/kami megfelel a teljesnek k -puffer memória kötött, miközben eléri a kívánt minőségi szintet. Ebben a munkában igazítunk a kívántnak való megfelelést célozva töredék találati aránya R :minőségi célként, más néven robusztussági arány: a kivont fragmentumok teljes száma a generált töredékek teljes számához képest. R :hatékonyan számítható hardveres gyorsítású okklúziós lekérdezések segítségével a számlálási lépésnél. A maximális memória-költségvetés felhasználó által megadott kemény megszorítása M :MB-ban van megadva.

Bár a hisztogram generálása egy eleve szekvenciális művelet, ahol több ezer pixel szavazatokkal járul hozzá a diszjunkt kategóriák csökkentett halmazához, ún. kükák, az elemzett változó (esetünkben töredékminták) az akkumuláció a modern grafikus hardvereken elérhető atomi kiegészítésekkel hajtható végre (lásd **hisztogram átadása** 3. algoritmusban). A rendelkezésre álló atomszámlálók számának jelenlegi hardveres korlátja miatt töredékenkénti mélységeig terjedő bonyolultságot tudunk kezelni. 1024, ami a legtöbb jelenethez több mint megfelelő.



3. ábra: Mélységi komplexitás hisztogramja (log2skála) az Hairball r csillagozás. Figyelje meg a számított kiszámára különböző v a GPU memória és a robusztussági arány értékei.

Adott th a szélességénél és magasságénél határozatának a rend ering képénés a tárolási követelmények a szeléből ctek k-puffer algoritmus, amely általában lehet exp ressed as $x \cdot k + y$ bájt per pixel, megtehetjük kiszámítani a n felső határba döntőre kry költségvetés az emlékeztető alapján $M_{balabbi}$ szerint:

$$M_b = \frac{(x \cdot k + y) \cdot \text{én}_w \cdot \text{én}_h}{1024^2} \quad (4)$$

$$k_b = \frac{1024^2 \cdot M_b / (\text{én}_w \cdot \text{én}_h) - y}{x} \quad (5)$$

Ezután tr elkerülve a generált mélységi komplexitást hisztogram hhátról előre, ami először a CPU-t a GPU-ból ferred a t [49] és normalizálva, bizonyosodunk meg arról, tudunk nekem hogy hány töredéket hagytunk ki eddig, és (1023... k_b) folytassuk a robusztussági arány eléréséig a cél teljesült. Ez az optimalizálás a következőképpen fogalmazható meg:

$$\min_{k \leq k_b} \left\{ \sum_{\text{én}=1023}^k (i - k) \cdot h[\text{én}] \right\} - R_h \quad (6)$$

Mivel nem egyszerű kiszámítani k_b -ben K esete+B-ABsök következetében

$$M_b = \frac{\sum_{p \in \text{px}} \min\{f(p), k_b\} + y \cdot \text{én}_w \cdot \text{én}_h}{1024^2} \quad (7)$$

beállítjuk $k \leq 1024$ és adjunk hozzá egy egyenlőtlenségi megkötést Eq. 6: $\sum_{p \in \text{px}} \min\{f(p), k\} \leq M_b - y \cdot \text{én}_w \cdot \text{én}_h$.

A 3. ábra a mélységi komplexitási hisztogramot szemlélteti a Hairball modell raszterezeit fix nézőpontból. Figyelje meg a kiemeltetka GPU memória és a robusztussági arány különböző értékeihez. Másrészt a 4. ábra bemutatja, hogyan k dinamikusan igazodik a felhasználói korlátokhoz ($M_b=200\text{MB}$, ami azt eredményezi $k_b=32$ és $R_h=95\%$) azokban a jelenetekben, ahol (balra) a virtuális kamera szabadon mozoghat a 3D jelenetben, és (jobbra) sok elemi objektum mozog és kölcsönhatásba lép egymással.

4.3 Egységes töredékpuffer

$k+A$ -buffer egy egységes keretrendszernek is tekinthető, amely sikeresen integrálja a funkcionálitást

3. algoritmus DinamikusK+B-ABSB(Sora,Hist,Pixp, Intk)

```

1:eljárás1EGYÉRTELMI(h,p) 2:
  h[.] := 0 ;
  p.címek s :=0; p.counterTotal: = 0;
  4:folyamat befejezése ure
  . teljes képernyős quad belépő
  . init hisztogram nullára

5:eljárás2 COUNT(p, k)
6:  hap.counterTotal <akkor
7:    p.counterTotal ←(+1);
8:  más
9:    lemezard;
10: vége ha
11:eljárás befejezése
  . geometria át
  . korlátos felhalmozódás

12:eljárás3 HISTOGRAM (h, p)
13:  hap.counterTotal>0 akkor
14:    h[p.counterTotal - 1] ←(+1); . növekedési tartály számlálója
15:  vége ha
16:eljárás befejezése
  . teljes képernyős quad belépő

17:eljárás4 REFERENCIN G(p, k)
18:  p.counterTotal :=mi n/p.counterTotal,k}.kötött számláló e
19:  p.cím :=számítógép pixel offset(p.counterTotal);
20:  p.számláló :=0;
21:eljárás befejezése
  . teljes képernyős quad belépő

22:eljárás5BOLT(a, t, f, p, k)
23:  p.method :=(k <16 vagy p.counterTotal <k) : Tömb ? Halom; . z
24:  hap.számláló <k vagy f <a[0].zakkor . töredék selejtezés
25:    szerez lock(); . lépjön be a kritikus szakaszba (1. alg.)
26:    hap.számláló <k t yúk . a tömb nincs tele
27:    üres beszúrás ( p.counter++,p.módszer);
28:    különben ha f <a[0] . zakkor . töredék selejtezés
29:      beszúrni full(p.módszer);
30:    vége ha
31:    zár feloldása(); . kilépés a kritikus szakaszból (1. alg.)
32:  vége ha
33:eljárás befejezése
  . geometria át
  . töredék selejtezés
  . a tömb nincs tele
  . kilépés a kritikus szakaszból (1. alg.)

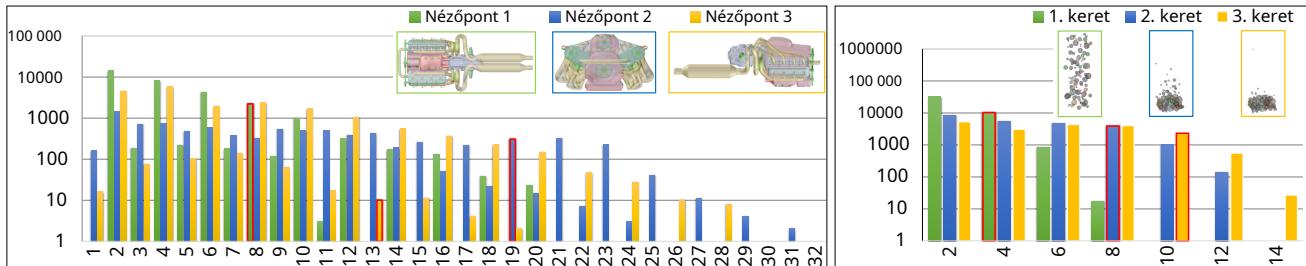
34:eljárás6ELHATÁROZÁS(a, p )
35:  h-mélységi válogatás (a, p,32); . töredékrendezés (4. alg.)
36:eljárás befejezése
  . ahol 'szöveg' és 'szöveg' ill. be kell illeszteni és
  el kell távolítani az átfedő funkciókból, amikor dinamikusk megoldásban
  kihasználva.{←, → atomot jelöl{bolt, növekmény} tevékenységek
  . teljes képernyős quad belépő
  . töredékrendezés (4. alg.)

```

Z-puffer, (többmenetes) k -puffer és A-puffer a felhasználó által meghatározott vagy dinamikusan számított típustól függően k érték. Egyetlen bejegyzés kiosztásával képpontonként ($k=1$), módszerünk biztosítja a nézőhöz legközelebb eső töredék megjelenítését. Ez azonban a hardvermélységi pufferreléshöz képest további memóriaigényekkel és teljesítménycsökkenéssel jár.

Másrészt a memória korlátaitól és a GPU feldolgozási kapacitásától függően/kértéke elég nagyra állítható, hogy elkerülje a töredék túlcsordulását ($k=\max_p\{f(p)\}$). Pontosabban, a keretrendszerünk egy hibrid sémának tekinthető, amely helyesen szimulálja az AB viselkedését Sora(amikor K+B) vagy AB-t használunkSB(amikor K+B-ABSB használt). Teljesítmény szempontjából a maxarray struktúra szemaformentes megvalósítását (3.2.1. fejezet) kell választani, mivel állandó beszúrási bonyolultsága, amikor a tömb nem tele (hiszen $\forall p[x, y] : f(p) \leq k$).

Annak ellenére, hogy a keretrendszerünk nincs korlátozva több renderelési célra és mintára az anti-



4. ábra: Mélységi komplexitás hisztogramjai (\log_{10} skála) fixhez $M_b=200\text{MB}$ és $R_h=0.95$, a motormotor modelljének renderelésekor (balra) különböző látószögekből és (jobbra) egy több objektumból álló animáció három képkockája. Figyeld meg, hogyan akerték alkalmazkodik (balra: {8,19,13} és jobbra: {4,8,10}) előrejelzési folyamatunk segítségével.

aliasing puffer, a korlátozott hardver erőforrások alacsony értéket eredményezhetnek k($\max_p\{\mathbf{f}(p)\}$). Így kifejlesztettük a k -mélységi leválasztási variációt az A-puffer funkcionálisának eléréséhez korlátozott memóriaigények esetén $d\max_p\{\mathbf{f}(p)\}/k$ iterációk megjelenítése. A Sec. 3.2.2, beszűrünk egy további egyenlőtlenségi kényszertyz($f_{\text{én}} = d_{i-1}[0] - f_{\text{én}}$)hogy az előző iteráció legtávolabbi töredékével több lépéses viselkedést kínáljonén –1 mint az iteráció selejtezési kritériumaén.

4.4-k-mélységi válogatás

Míg a szakaszban leírt töredékrendezési stratégia. A 3.3 nem szenved a helyi memória-gyorsítótár túlcordulásától és a kis értékekkel való csökkentett üzem közbeni cserétőlk($k \leq 64$), ez nem igaz, ha dinamikus forgatókönyvekkel dolgozunk (lásd 4.2. fejezet). Bővíve a töredékek rendezésének gondolatát a mélységi hámozással helyi gyorsítótár szinten [37], egy gyorsabb variációt javasolunk, az ún. k -mélységi válogatásK meghosszabbításával+B-tömb növeli a számítási teljesítményt, valamint a maximális támogatott mélységgkomplexitást a helyi memóriában.

Minden iterációban a/-elülső töredékek, hol/ előre meghatározott és tesztjeinkben rögzített32, hatékonyan választják ki a globális töredékmemória bejárásával, és egy/-méretű helyi tömb, követve a szakaszban leírt stratégiát. 3.2.1. Kérjük, vegye figyelembe, hogy ebben az esetben nincs szükség szemaforokra, mivel ez egy szekvenciálisan kezelt probléma. Ezt követően a rögzített töredékeket rendezi és feloldja, ezzel véglegesítve az aktuális iterációt. Az ebben az iterációban rögzített legtávolabbi fragmentum hatékonyan használható a következőben az összes korábban feldolgozott töredék eldobására.

Ennek a megközelítésnek a korlátai (i) képtelenek kezelnı a z-fighting problémákat (a jelenség kiküszöbölésére vonatkozó átfogó áttekintésre utaljuk az olvasókat [18]), és (ii) az a követelmény, hogy a teljes töredéklistát többször ki kell olvasni a globális GPU-ról. memória ($d\mathbf{f}(p)/le$). Másrészt a hurok hamarabb is véget érhet, pl. abban a speciális esetben, amikor az átlátszatlanság küszöbértékét [41] hajtják végre az átlátszó objektumoknál, elkerülve a sok számítást. A teljes k -mélységi hámolasztás kódja a 4. algoritmusban látható (lásd a mellékletet kiegészítő anyagként).

Végül az 5. ábra a teljes k -buffer keretrendszer, kiemelve a komponensek (shaderek) közötti áramlást, amelyet követni kell a megfelelő funkciók végrehajtásához.

5 REREDMÉNYEK

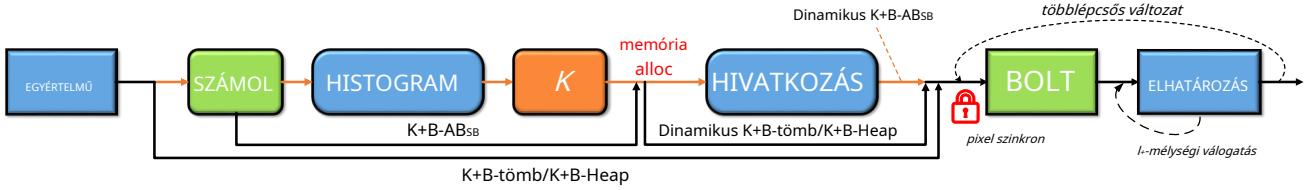
Kísérleti elemzésünket mutatjuk be k -pufferes megközelítés a mélységi hámolasztás sorozatával szemben, k -puffer és A-puffer megvalósítások, amelyek a teljesítményre, a robosztusságra és a memóriaigényekre összpontosítanak különböző tesztelt körülmények között. A teljesítményt ezredmásodpercben (ms), a memóriaigényt pedig megabájtból (MB) mértük. Összehasonlítás céljából a KB-AB két változatát fejlesztettük kiLL, ahol ahelyett, hogy pixelenként linkelt listákat használnánk az A-puffer felépítéséhez, vagy rögzített hosszúságú (**KB-ABsor**) vagy változó hosszúságú (**KB-ABsb**) tömbök minden képponthoz. Kiegészítő anyagként rendelkezésre áll az összes tesztelt módszer shader forráskódja is. minden metódus OpenGL-lel van megvalósítva4.4API és főleg az NVIDIA GTX-en fut780Ti vele3GB memória. Végül megvalósítottuk saját szemafor mechanizmusunkat az INTEL pixelszinkronizálása helyett a KB-PS metódus futtatásához a fenti hardveren.

Az 1. táblázat összehasonlító áttekintést nyújt az összesről k -puffer alternatívák a memóriaigény, a renderelés bonyolultsága és egyéb jellemzők tekintetében.

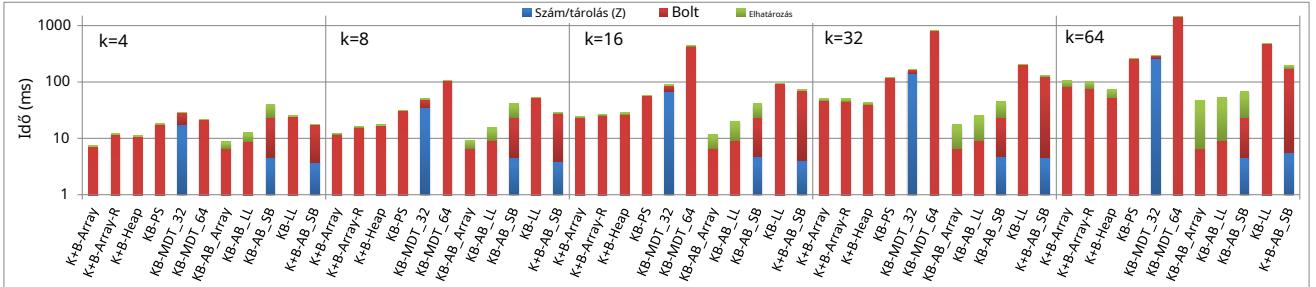
5.1 Teljesítményelemzés

Elvégeztük módszereink teljesítményének kísérleti értékelését a versengő technikákhoz képest, több különböző konfigurációban jelenetgyűjtemény felhasználásával. Célunk, hogy kiterjesszük és frissítsük a jelen cikk eredeti verziójában található teljesítményelemzési következetet [26], ahol a kísérleteket egy régebbi NVIDIA GTX-en végezték.480.

A különböző képfelbontású jelenetek renderelése helyett a 1024x1024nézetablakot és végrehajtott nagyítást, amely különböző képlefedettségi értékeken végzett méréseket eredményezett. Az igazságos összehasonlítás érdekében minden módszert extrém körülmények között tesztelnek mesterségesen generált jelenetek mellett, amelyek bizonyos százalékát lefedik



5. ábra: Diagram *ak+*-puffer csővezeték. minden doboz egy shader programot jelent, kivéve a narancssárgát, amely CPU-ban van megvalósítva. A kék négyzetek képpontonként, teljes képernyős négyes renderelési lépéssel, míg a zöldek minden geometria-raszterezett töredékhez kerülnek végrehajtásra.



6. ábra: Teljesítményértékelés ms-ban (log₁₀skála) az *k*-puffer változatok változó *k*egy jelenetén *n*=128.

képernyőmérét (vagy pixelsűrűség: p_d) és előállítani $r=r/k$ véletlenszerűen előállított töredékek pixelenként, ahol $r \geq 1$.

5.1.1 *k*-puffer Összehasonlítás

Hatása *k*. A 6. ábra azt mutatja be, hogy a számítási idő egy halmaz minden egyes renderelési menete hogyan történik *k*-puffer módszerek, skálák értékének növelésével $k=4, 8, 16, 32, 64$ egy jelenethez, amely abból áll $n=128$ töredékek pixelenként. Megfigyeljük, hogy a mi K+A B változatok mindenkinél jobban teljesítenek, mint a többi memóriakorlátos technika kértékeket. Ahogy az várható volt, K+A B-Heap jobban teljesít, mint a K+B-tömb nagyobb értékekhez *k*. Vegye figyelembe, hogy mikor $k=64$, K+A B-Heap készül $3.5 \times$ és $4 \times$ gyorsabb, mint a KB-PS és KB-MDT jelenlegi megvalósításai 32 , ill. Bár a onepass KB-MDT₆₄ felülműlja azt 32-bites verziót a számára $k=4, A$ fő teljesítményproblémák akkor jelennek meg, amikor a töredékes versenybőzint intenzívebbé válik a nagyobb értékek esetében, *k*, valószínűleg a lassabb miatt 64-bit atomi műveletek.

Módszereink valamivel lassabbak, mint az A-puffer alapúak a modern hardveren végzett gyorsabb atomműveletek miatt (L2gyorsítótár). Vegye figyelembe, hogy a *elhatározás* lépés drágább az előbbi módszereknél, mivel meg kell találnia a legközelebbi *k* töredékeket az összes elfogottból a válogatás előtt. Végül megfigyeljük, hogy a *számol* és *elhatározás* passzol K+B-ABsba KB-AB számítási költsége alacsonyabb, az előbbi által végzett korlátozott műveletek miatt. Azonban a lassú töredéktárolás a globális memóriában teljesítménycsökkenést eredményez, ha a raszterizált töredékek száma jelentősen megnő.

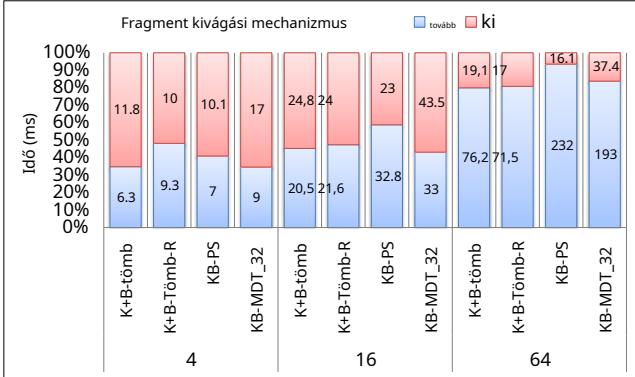
A selejtezés hatása. Ugyanezt a tesztelt konfigurációt követve a 7. ábra szemlélteti, hogy a teljesítmény jelentősen javul a töredékek selejtezése során

mechanizmust használják ki a mi K+B-tömb módszerek. KB-PS és KB-MDT₃₂ hatékonyan beállítható, hogy támogassa ezt a funkciót, ami jelentős lökést ér el, azonban nem elég ahhoz, hogy legyőzze a K+B-tömb teljesítménye. Ahogy az várható volt, a mechanizmus megnyugtat, amikor *knöveli*.

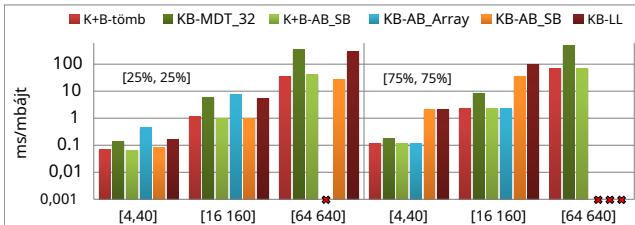
Sajnos a javasolt töredék-elutasítási folyamat több okból is szenned **korlátozások**. Először is, a folyamat kezdetben megköveteli a beillesztést töredékeket, mielőtt bármilyen selejtezési vizsgálatot elkezdene alkalmazni. Másodszor, ez a töredék bejövő mélyiségi sorrendjétől függ; nincs hatással a legrosszabb esetre, amikor a töredékek csökkenő sorrendben érkeznek. Ezenkívül a tényleges töredékek eltávolítása sajnos a pixel shader végrehajtásán belül történik, elkerülve a hardveres gyorsítású korai Z selejtezés kihasználásával járó teljesítménynövekedést.

A memóriakorlátás hatása. A 8. ábra a teljesítményértékelést szemlélteti ms/MB-ban kifejezve egy tesztelt egység esetében *k*-puffer metódus beállítva, amikor a teljesítmény és a memória rendkívül fontos. Építeni *k*-fragmentless pixels, a pixelek befolyásolását legfeljebb $n=10$ -kötredékek. Így definiáljuk f_p mint annak a valószínűsége, hogy egy generált töredéket nem dobunk el. Megfigyeljük, hogy K+B-ABsba sok üres képpontot tartalmazó jelenetek kezelésére előnyös ($p_d=25\%$) és kis számú raszteres töredék ($f_p=25\%$). Amikor a pixel- és töredéksűrűség nő ($p_d=75\%$, $f_p=75\%$), K+B-ABsba jobban teljesít, mint a többi memória-tudatos módszer. Azonban K+B-ABsba viselkedése általában a legrosszabb, mint a korlátos módszerek, mivel elméletileg lassabban teljesít (pl. egy plusz lépés, adatok tárolása a globális memóriában) a kis, nem használt mem-mével együtt.

a korlátos módszerekről. A KB-AB gyors sebessége ellenére britka jeleneteknél a teljesítmény jelentősen csökken, ha a keletkezett töredékek magas szintre robbannak fel. Végül a KB-AB_{Sor}, KB-AB_{SB} és KB-LL nem működik, ha a töredékkiosztás memóriatúlcordulást eredményez ($k=64$).



7. ábra: Teljesítményértékelés k -puffer variánsok a töredéklevágási mechanizmusunk engedélyezésével és anélkül $k=4,16,64$.



8. ábra: A teljesítményértékelés összehasonlítása ms/MB-ban (log₁₀skála) az k -puffer variánsok, amikor egy jelenetről mozog $r=10$ kis számktól a nagy számú generált töredék felé.

A GPU (szinkronizálás) hatása. A 9. ábra (balra) szemlélteti a tesztelt teljesítmény értékelését k -puffermódszer beállítása különböző grafikus hardvereken és szinkronizálási megvalósítások fixhez $k=8$ és változón $n \in \{8, 16, 32\}$. Kezdetben a hardveres gyorsítású PS/FSI-bővítmények felsőbbrendűségét figyeljük meg az SL-megvalósításunkkal összehasonlítva minden NVIDIA GTX-en. 970 és Intel Iris Pro Graphics 5200 kártyák, ill. Figyelje meg, hogy minden szinkronizálási technika lineáris viselkedést tapasztal, amikor magasabb töredékes versenyzésre vált. Érdekesség, hogy K+A B-tömb teljesítménye még a KB-PS-nél is jobb, ha a szinkronizálást hardveresen gyorsítják (Nvidia: $\approx 2x$, Intel: $\approx 1.5x$). Ha összehasonlítkuk a régebbi (GTX480) szemben a modernnel (GTX780 Ti, 970) grafikus kártyák esetén megfigyeljük a KB-MDT fokozott viselkedését és KB-AB_{Sor} az utóbbi kártyák fejlett atomi műveleteit miatt.

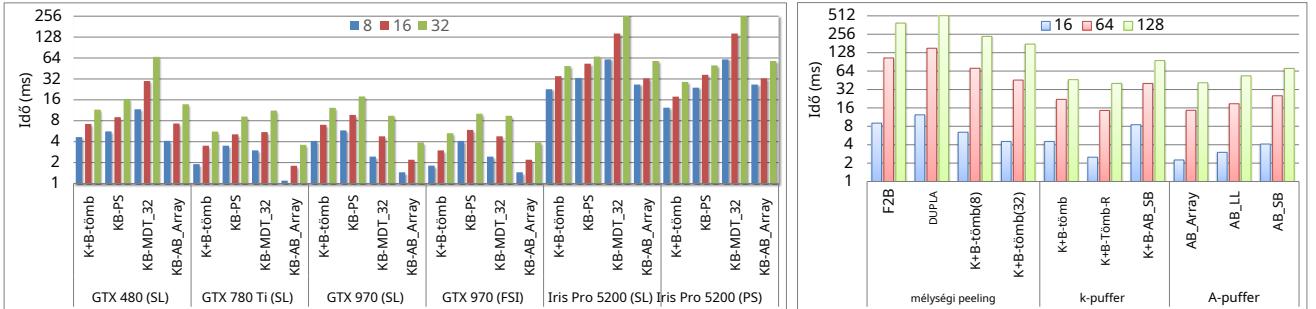
5.1.2 Mélységi hámlasztás/A-puffer összehasonlítása

A 9. ábra (jobbra) szemlélteti módszereink teljesítményének összehasonlítását a mélységi hámlasztással és az A-puffer alternatíváival egy változó mélységű komplexitású jelenethez. Ebben a beállításban a töredék számosságára van állítva, így K+A B módszerek képesek befogni az összes generált töredéket. Kezdetben azt látjuk, hogy a mi többmenetes K-verzióink+B-tömb -val $k=8,32$ jobban teljesít, mint a széles körben használt F2B és DUAL metódusok, az utóbbiak többi renderelési iterációi miatt. Ne feledje, hogy az alacsony geometriai részleteket tartalmazó (kis számú háromszög) jelenetek nagy felbontású kezeléséhez előnyös az F2B használata [18].

Megfigyeljük, hogy a K átdolgozott változata+A B-tömb enyhíti a szükségtelen selejtező mechanizmus terheit, jobban teljesít az összes memóriatudatos Abuffer változatnál, és valamivel rosszabb, mint az AB_{Sor}, az eddig leggyorsabb A-puffer megvalósítás. Vegye figyelembe, hogy $k=4$ a mélyhámlasztás javította a válogatási teljesítményt $3.1 \times$, $n=64$ és $4.64 \times$, $n=128$. Kihagyjuk a K tesztelését+B-Heap óta K+A B-tömb valamivel jobb teljesítményt nyújt, mint az előbbi, és ezt tovább fokozza az állandó idejű beillesztési folyamat egy kitöltetlen tömbön. Másrészt K+B-ABsrosszabb mint az AB_{Sor} minden esetben. A szükségtelen selejtezési költségeken kívül a megnövekedett töredékszinkronizálás jelentősen befolyásolja a teljesítményt.

5.2 Memóriakiosztás An elemzés

Az 1. táblázat az összes rendelkezésre y a memória ble módszerek álló vagy kevésbé szimulált viselkedés tekintetében, hogy több vagy komplexit fogyasztását mutatja be az k -puffer. Mi inimethods figyeljük meg, hogy a mi K+B igényel egy kicsit több tárhely (8 bájt) pixelenként, mint a többi memóriakorlátos módszer (KB, KB-SR, KB-PS, KB-MDT)₃₂, 64, KB-MHA) a számláló és szemafor textúrák további kiosztása miatt. Amikor szélsőséges képernyőfelbontásra váltunk, ez a teher észrevehető. A KB, KB-Multi és KB-MHA metódusoknak több tárhelyre van szükségük, amikor az adatcsomagolást vizsgálják ($A \times k > 1:4k > 2k+2$). K+A B módszerek kevesebb memória erőforrást igényelnek a KB-SR-hez képest ($A \times k > 2:3k > 2k+2$). Ne feledje, hogy a szemafor textúra kiosztása továbbra is elkerülhető, ha a képpont szinkronizálási bővítményt Haswell vagy Maxwell hardveren alkalmazzák. Másrészt a videomemória fogyasztása magas szintre emelkedik, amikor A-puffert készítenek. Vegye figyelembe a KB-AB megnövekedett memóriaigényét sora pixelenkénti maximális memória lefoglalására vonatkozó stratégiája miatt ($p(n=\max(k, K))$). AB_{LL}, KB-LL, KB-ABs kevesebb tárolási erőforrást igényel az által, hogy dinamikusan csak a nem üres pixelekhez osztja ki a tárhelyet ($f(p) \in [1, n]$). Memóriatudatos módszerünk K+B-ABs begyenlőt igényel (amikor $f(p) \leq k$) vagy kevesebb (mikor $f(p) > k$). Érdekes megfigyelés, hogy a K+B-tömb és K+B-ABs amikor kiterjesztették az összes rögzítésére



9. ábra: Teljesítményértékelés ms-ban (log2skála) / (bal) k -puffer-alternatívák különböző grafikus hardvereken és szinkronizálási megvalósításokon, valamint (jobbra) mélység-lehúzás és A-puffer alternatívák a változó intenzitású jeleneteknél.

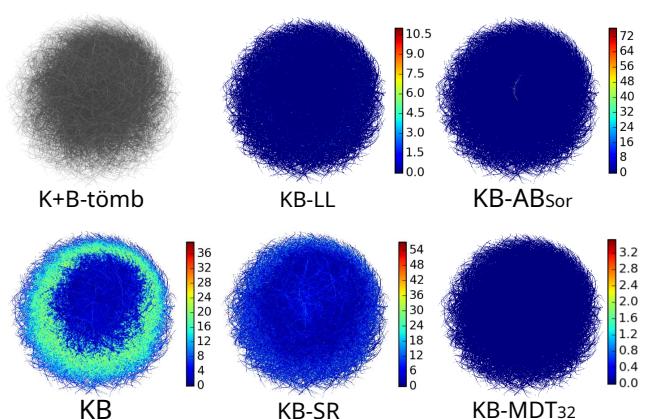
Algoritmus		Teljesítmény	Rendezési igény		Peeling pontosság	memória		
Betűszó	Leírás	A geometria átmegy	a primitíveken	töredékeken	Max k	Műtárgyak	Pixel-kiosztásokonként	Rögzített
KB	A kezdeti k -puffer implementáció [20],[42]	1	✓	BOLT	8; 16	MRT veszélyek, Geom. Interpen.	2k; 4k ; ⚡ ; ⚡ ; ⚡	
KB-Multi	Több passzív k -puffer [23]	1-től k-ig	✓	ELHATÁROZÁS			2k; 4k ; ⚡ ; ⚡ ; ⚡	
KB-SR	Stencil irányítva k -puffer [21]	1	✓	ELHATÁROZÁS	32	Geom. Interpen.	3k ; ⚡ ; ⚡	
KB-PS	k -puffer pixelszinkronizálással [24]	1	✗	BOLT	-		2k ; ⚡	
K+B-tömb	k -puffer max-array használatával	1	✗	ELHATÁROZÁS	-		2k + 2 ; ⚡ ; ⚡	
K+B-Heap	k -puffer max-heap használatával	1	✗	ELHATÁROZÁS	-		2k + 2 ; ⚡ ; ⚡	
KB-MDT ₃₂	Többmélységű tesztiséma (32 bit) [22], [28]	2	✗	BOLT	-		2k ; ⚡	
KB-MDT ₆₄	Többmélységű tesztiséma (64 bit) [45]	1	✗	BOLT	-	Csak tárolható 32 bites szín	2k ; ⚡	
KB-MHA	Memoriaveszély-tudatos k -puffer [43]	1	✓	BOLT	8; 16	MRT veszélyek, Geom. Interpen.	2k; 4k ; ⚡ ; ⚡ ; ⚡	
KB-AB _{Sor}	k -A-puffer alapú puffer (fix méretű tömbök)	1	✗	ELHATÁROZÁS	-	memória túlcordulás kockázatok	2n + 1 ; ⚡ ; ⚡ ; ⚡ ; ⚡	
KB-AB _{LL}	k -A-pufferen alapuló puffer (dinamikus linkelt listák) [9],[47]	1	✗	ELHATÁROZÁS	-		3f + 1 ; ⚡ ; ⚡ ; ⚡	
KB-LL	k -a csatolt listákon alapuló puffer [9]	1	✗	BOLT	-		3f + 6 ; ⚡ ; ⚡ ; ⚡	
KB-AB _{SB}	k -S-pufferen alapuló puffer (változó-függő régiók)	2	✗	ELHATÁROZÁS	-		2f + 2 ; ⚡ ; ⚡	✗
K+B-AB _{SB}	k -S-pufferen alapuló puffer (változó-függő régiók)	2	✗	ELHATÁROZÁS	-		2f + 3 ; ⚡	
$f(p) = \# \text{töredékek pixelnél } p[x,y]$		$n = \max_{x,y} f(p)$			A-ban; B, A jelöli az alapszintű foliákat/memóriát metódus és B az attribútumot használó variációhoz csomagolás			
$f_k(p) = (f(p) < k) ? f(p) : k$		$f_k(p) \leq k$						

1. TÁBLÁZAT: A korábbi átfogó összehasonlítás k -puffer megközelítések és a bevezetett $k+$ -puffer változatok. Intuitív módon több csillag nagyobb memóriaigényt jelez. A töredékek rendezése oszlop jelzi, hogy a mélységrendezés melyik szakaszban történik.

töredékek ($k=n$) ugyanolyan tárhelyet igényelnek, mint az AB_{Sor} és AB_{SB} módszerek, ill. Végül az apró hisztogram kiosztás (4KByte, felbontásfüggetlen) az egyetlen tárolási igény, amikor a dinamikus/folyamat engedélyezve van.

5.3 Képminőség-elemzés

A 10. ábra a KB, KB-SR, KB-MDT képkülönbségeit mutatja, KB-LL és KB-AB_{Sor} módszerek összehasonlítható a Hairball renderelés alapigazságával (180max rétegek) -val $k=16$ (minden tesztelt módszer támogatja). Észrevehető minőségrömlás figyelhető meg az alsó három képen a (bal oldali) KB és (középen) KB-SR módszerek RMW veszélyei, valamint a KB-MDT (jobb) mélysékonverziós műtermékei miatt. A KB-LL következetesen korlátozva van, hogy garantálja a végtelen hurokmentes viselkedést a jelölt fragmentumok ismételt sikertelen beillesztése miatt. (fent, középen) Ez észrevehető töredékvesztést eredményez. A KB-AB memória túlcordulás elkerülése érdekében, kevesebb tárhelyet kell foglalnunk (120rétegek), mint amennyire valójában szükségünk van, ami (felül, jobbra) információvesztést okoz egy kis pixelkészlethez.



10. ábra: Hótérképpel kódolt különbségek a K segítségével előállított kép között+B-tömb több kimenete ellen k -puffer változatok.

A teljességet tekintve KB, KB-SR, KB-MHA és KB-MDT₆₄ csak a tömöríthető töredék jellemzőit kihasználó vizuális effektusok előállítására korlátozódnak.

32 bites kompakt vektorra. Végül, $k+$ -a puffer könnyen adaptálható a többmintás élsimítás támogatására (**MSAA**) a széles körben elfogadott töredéklefedettség-alapú stratégia követésével [5].

6 CÖKÖVKEZTETÉSEK

Bemutattuk $k+$ -puffer, továbbfejlesztett pixelszinkronizált és töredékseljezetés-tudatos k -buffer keretrendszer, amely a GPU-n karbantartott két új, korlátos tömb adatstruktúrára épül. Egy további geometria-megjelenítés is elvégezhető annak érdekében, hogy elkerüljük az érték kézi beállításának fárasztó feladatát. A megfelelő érték előrejelzésével repülés közbeni mélységi komplexitási hisztogram elemzéssel. Egy precíz memóriakiosztási stratégia is beépítésre került a javasolt folyamatba, amely az egyes pixelek mélységi összetettségehez igazítja a tárolóhasználatot. A megvalósítás részleteit és a különféle lépések könnyű alternatív megvalósításait is biztosítottuk, hogy lehetővé tegyük rendszerünk teljes támogatását a különböző GPU-architektúrákon. Egy kiterjedt kísérleti összehasonlítás bebizonyította, hogy keretünk jobb az előzőhöz képest k -puffer megoldások a memória, a teljesítmény és a képminőség tekintetében.

További irányok is feltárhatók a láthatóság meghatározásának problémájának megoldására több töredékes megjelenítési megoldásokban. Míg a töredékek kiugró értékeit elutasító mechanizmusunk kulcsfontosságú a keretrendszerünkben, további kutatásokat kell végezni annak érdekében, hogy enyhítsük annak sorrendfüggő teljesítményét. Ezenkívül a töredékek selejtezésének felgyorsítása elérhető a szomszédos keretek közötti időbeli koherencia kihasználásával [50]. Végül a keretrendszerünk kombinálható egy figyelem alapú részletgazdag menedzserrel, hogy leminősítse a töredéktárolást azokon a területeken, amelyeket a megfigyelő várhatóan észre sem vesz [51].

AISMERTETÉSEK

Köszönjük Anthousis Andreadisnak és az AUJEB Computer Graphics Group többi tagjának hozzájárulásukat és támogatásukat. Andreas Vasilakist az Intel hardveradományai támogatták. Stanford sárkány és szőrgombóc A modellek a Morgan McGuire's Computer Graphics Archive oldaláról töltötték le. Golyókaz animációt a Utah Egyetem 3D Animation Repositoryjából szereztük be. Szeretnénk megköszönni Louis Bavoilnak, hogy biztosította számunkramotoros motorháló. Ez a kutatás az Európai Unió (Európai Szociális Alap – ESZA) és a görög nemzeti alapok társfinanszírozásával valósult meg a Nemzeti Stratégiai Referenciakeret (NSRF) „Oktatás és élethosszig tartó tanulás” Operatív Programján keresztül – Kutatásfinanszírozási Program: ARISTEIA II-GLIDE (támogatás sz.3712). Andreas Vasilakis a megfelelő szerző.

REFERENCIA

- [1] Y. Tokuyoshi, T. Sekine, T. da Silva és T. Kanai, „Adaptive ray-bundle tracing with memory usage előrejelzés: Hatékony globális megvilágítás nagy jelenetekben”, *Számítógépes grafikai fórum*, vol. 32. sz. 7, 315–324., 2013.
- [2] M. Salvi és K. Vaidyanathan, „Multi-layer alfa blending”, in *ACM SIGGRAPH interaktív 3D grafikával és játékokkal foglalkozó szimpózium 18. üléssének anyaga*, ser. I3D '14. New York, NY, USA: ACM, 2014, 151–158.
- [3] KE Hillesland, B. Bilodeau és N. Thibieroz, „Deferred shading for order-independent transparency”, in *Proceedings of Eurographics 2014 Short Papers*, ser. EG '14. The Eurographics Association, 2014, 49–52.
- [4] Szécsi L., Barta P. és Kovács B., „Volumetric transparency with per-pixel fragment lists”, *GPU PRO 3: Fejlett renderelési technikák*, p. 323, 2012.
- [5] JC Yang, J. Hensley, H. Grun és N. Thibieroz, „Real-time concurrent linked list construction on the GPU”, *Számítógépes grafikai fórum*, vol. 29. sz. 4, 1297–1304, 2010.
- [6] R. Carneky, R. Fuchs, S. Mehl, Y. Jang és R. Peikert, „Smart transparency for illustrative visualization of complex flow surfaces”, *IEEE-tranzakciók a vizualizációt és a számítógépes grafikán*, vol. 19, sz. 5., 838–851. o., 2013. május.
- [7] D. Kauker, M. Krone, A. Panagiotidis, G. Reina és T. Ertl, „Rendering molecular surfaces using order-independent transparency”, in *A párhuzamos grafikáról és vizualizációról szóló 13. eurografikai szimpózium anyaga*, ser. EGPGV '13. Aire-la-Ville, Svájc: Eurographics Association, 2013, 33–40.
- [8] J. Parulek és A. Brambilla, „Fast blending schema for molecular surface representation”, *IEEE-tranzakciók a vizualizációt és a számítógépes grafikán*, vol. 19, sz. 12., 2653–2662. o., 2013. dec.
- [9] X. Yu, JC Yang, J. Hensley, T. Harada és J. Yu, „A keretrendszer összetett szórási hatások hajon való megjelenítéséhez” ban ben *Az interaktív 3D-s grafikával és játékokkal foglalkozó 2012-es szimpózium anyaga*, ser. I3D '12. NY, USA: ACM, 2012, 111–118.
- [10] Y.-S. Leung és CCL Wang, „Szilárd anyagok konzervatív mintavételei képtérben”, *IEEE számítógépes grafika és alkalmazások*, vol. 33. sz. 1., 32–43. o., 2013. jan.
- [11] NK Govindaraju, M. Henson, MC Lin és D. Manocha, „Interactive visibility ordering and transparency computations among geometric primitives in complex environments” *Az interaktív 3D grafikával és játékokkal foglalkozó 2005-ös szimpózium anyaga*, ser. I3D '05. NY, USA: ACM, 2005, 49–56.
- [12] E. Sintorn és U. Assarsson, „Valós idejű hozzávetőleges rendezés az önrányelőlástsárt és az átlátszóságért a hajrenderelésben” *Az interaktív 3D-s grafikáról és játékokról szóló 2008-as szimpózium anyaga*, ser. I3D '08. NY, USA: ACM, 2008, 157–162.
- [13] G. Chen, PV Sander, D. Nehab, L. Yang és L. Hu, „Depthpresorted triangle lists”, *ACM Trans. Grafikon.*, vol. 31. sz. 6., 160:1–160:9 o., 2012. nov.
- [14] J. Huang és MB Carter, „Interactive transparency rendering for large cad model”, *IEEE-tranzakciók a vizualizációt és a számítógépes grafikán*, vol. 11, sz. 5., 584–595., 2005. szep.
- [15] J. Rossignac, I. Fudos és AA Vasilakis, „Direct rendering of boolean kombinációk önvágó felületek” *Számítógéppel segített tervezés*, vol. 45, sz. 2., 288–300. o., 2013. febr.
- [16] C. Crassin, „Linked lists of fragment pages”, 2010.
- [17] AA Vasilakis és I. Fudos, „S-buffer: Sparsity-aware multifragment rendering”, in *Proceedings of Eurographics 2012 Short Papers*, ser. EG '12, Cagliari, Szardínia, Olaszország, 2012, 101–104.
- [18] A.-A. Vasilakis és I. Fudos: „Mélyesség elleni küzdelem tudatos módszerei többtöredékes renderinghez”, *IEEE-tranzakciók a vizualizációt és a számítógépes grafikán*, vol. 19, sz. 6., 967–977., 2013.
- [19] M. Maule, JL Comba, RP Torchelsen és R. Bastos, „A raszteres átlátszósági technikák felmérése”, *Számítógépek és grafika*, vol. 35, sz. 6, 1023–1034, 2011.
- [20] L. Bavoil, SP Callahan, A. Lefohn, JLD Comba és CT Silva, „Multi-fragment effects on the GPU using the k -puffer” be *Az interaktív 3D grafikával és játékokkal foglalkozó 2007-es szimpózium anyaga*, ser. I3D '07. NY, USA: ACM, 2007, 97–104.
- [21] L. Bavoil és K. Myers: „Késléltetett renderelés irányított sablon használatával k -puffer”, *ShaderX6: Fejlett renderelési technikák*, 189–198., 2008.
- [22] M. Maule, J. Comba, R. Torchelsen és R. Bastos, „Hybrid transzparencia”, in *A 2013-as interakciós szimpózium anyaga*

- tív 3D-s grafika és játékok**, ser. I3D '13. New York, NY, USA: ACM, 2013, 103–118.
- [23] B. Liu, L.-Y. Wei, Y.-Q. Xu és E. Wu, „Multi-layer deep peeling via fragment sort”, in *Proceedings of 11. IEEE International Conference on Computer-Aided Design and Computer Graphics*, 2009, 452–456.
- [24] M. Salvi, „Advances in real-time rendering in games: Pixel szinkronizálás: Old grafikai problémák megoldása új adatstruktúrákkal”, in *ACM SIGGRAPH 2013 tanfolyamok*, ser. SIGGRÁF '13. New York, NY, USA: ACM, 2013.
- [25] Z. Bolz és M. Heyer, „OpenGL extension: GL NV fragment shader interlock”, 2014.
- [26] AA Vasilakis és I. Fudos, „k+buffer: Fragment synchronized k-buffer”, in *Az ACM SIGGRAPH interaktív 3D grafikával és játékokkal foglalkozó szimpózium 18. ülésének anyaga*, ser. I3D '14. New York, NY, USA: ACM, 2014, 143–150.
- [27] L. Carpenter, „The A-buffer, an antialiased rejected-felület módszer”, in *A 11. Számítógépes grafika és interaktív technikák éves konferenciájának anyaga*, vol. 18, sz. 3. ACM New York, NY, USA, 1984, 103–108.
- [28] N. Thibierge, „Rendeléstől független átláthatóság perpixelhez kötött listák segítségével”, *GPU Pro* 2, 409–431., 2011.
- [29] F. Liu, M.-C. Huang, X.-H. Liu és E.-H. Wu, „FreePipe: programozható párhuzamos renderelő architektúra a hatékony többtöredékes effektusokhoz” *Az interaktív 3D-s grafikáról és játékokról szóló 2010-es szimpózium anyaga*, ser. I3D '10. New York, NY, USA: ACM, 2010, 75–82.
- [30] C. Crassin, „Gyors és pontos egymenetes A-puffer”, 2010.
- [31] M. Maule, J.L. Comba, R. Torchelsen és R. Bastos, „Memoryoptimized order-independent Transparency with dynamic fragment buffer” *Számítógépek és grafika*, vol. 38., 2014. 1–9.
- [32] A. Patney, S. Tzeng és JD Owens, „Fragment-parallel kompozit és szűrő”, *Számítógépes grafikai fórum*, vol. 29. sz. 4, 1251–1258, 2010.
- [33] P. Knowles, G. Leach és F. Zambetta, „Efficient layered fragmens puffertechnikai”, in *OpenGL Insights*, P. Cozzi és C. Riccio, szerk. CRC Press, 2012, 279–292.
- [34] P. Knowles, G. Leach és F. Zambetta, „Gyors válogatás az összetett jelenetek pontos elkészítéséhez”, *A vizuális számítógép*, vol. 30, sz. 6–8., 603–613., 2014.
- [35] F. Liu, M.-C. Huang, X.-H. Liu és E.-H. Wu, „Hatékony mélységi peeling via bucket sort”, in *A 2009-es nagy teljesítményű grafikáról szóló konferencia anyaga*, ser. HPG '09. New York, NY, USA: ACM, 2009, 51–57.
- [36] E. Sintorn és U. Assarsson, „Hair önárynyékolás és átlátszósági mélység rendezése foglaltsági térképekkel”, in *Az interaktív 3D-s grafikáról és játékokról szóló 2009-es szimpózium anyaga*, ser. I3D '09. New York, NY, USA: ACM, 2009, 67–74.
- [37] S. Lindholm, M. Falk, E. Sundin, A. Bock, A. Ynnerman és T. Ropinski: „Hibrid adativizualizáció mélységi komplexitási hisztogram elemzésén alapuló” *Számítógépes grafikai fórum*, 2014.
- [38] P. Knowles, G. Leach és F. Zambetta, „Backwards Memory Allocation and Improved OIT”, in *Proceedings of Pacific Graphics 2013 (rövid tanulmányok)*, ser. PG '13, 2013. október, 59–64.
- [39] C. Everitt, „Interactive order-independent Transparency”, Nvidia Corporation, Tech. Rep., 2001.
- [40] L. Bavoil és K. Myers, „Rendeljen független átlátszóságot kettős mélységű peelinggel”, Nvidia Corp., Tech. Rep., 2008.
- [41] D. Wexler, L. Gritz, E. Enderton és J. Rice, „GPU-gyorsított, kiváló minőségű rejtett felület eltávolítása” *A 2005-ös Grafikai hardver konferencia anyaga*, ser. HWWS '05. New York, NY, USA: ACM, 2005, 7–14.
- [42] N. Carr, R. Měch és G. Miller, „Coherent layer peeling for transparens, nagy mélységű komplexitású jelenetek” c. *A 2008-as Grafikai Hardver Szimpózium előadásai*, ser. GH '08. Aire-la-Ville, Svájc: Eurographs Association, 2008, 33–40.
- [43] SP Callahan, M. Ikits, JLD Comba és CT Silva, „Hardveresen támogatott láthatósági rendezés strukturálalisan kötetrendeléshez”, *IEEE-tranzakciók a vizualizáció és a számítógépes grafikán*, vol. 11, sz. 3., 285–295. o., 2005. május.
- [44] N. Zhang: „Memory-hazard-aware-k-puffer algoritmus a sorrendtől független transparencia megjelenítéshez” *IEEE Trans. a Visualizaton. és Comput. Grafikon.*, vol. 20, sz. 2., 238–248., 2014.
- [45] W. Wang és G. Xie, „Memória-hatékony, egymenetes GPU renderelés több töredékes effektusokról”, *IEEE tranzakció. a vizualizációról és a számítógépes grafikáról*, vol. 19, sz. 8., 1307–1316., 2013.
- [46] C. Kubisch, „Order független transparencia az OpenGL 4.x-ben”, in *GPU technológiai konferencia 2014*, ser. ÁSZF 2014. 14.
- [47] P. Brown, Z. Bolz, C. Crassin és C. Kubisch, „OpenGL-kiterjesztés: GL nv shader atomic int64”, 2014. –
- [48] M. Salvi, J. Montgomery és A. Lefohn, „Adaptive transparency”, in *A 2011-es nagyteljesítményű grafikák szimpóziumának anyaga*. NY, USA: ACM, 2011, 119–126.
- [49] L. Hrabcák és A. Masserann, „Asynchronous puffer transfers”, in *OpenGL Insights*, P. Cozzi és C. Riccio, szerk. CRC Press, 2012, 391–414.
- [50] D. Scherzer, L. Yang, O. Mattausch, D. Nehab, PV Sander, M. Wimmer és E. Eisemann: „Időbeli koherencia módszerek a valós idejű renderingben”, *Számítógépes grafikai fórum*, vol. 31. sz. 8., 2378–2408. o., 2012. dec.
- [51] S. Lee, GJ Kim és S. Choi, „Vizuálisan látogatott objektumok valós idejű nyomón követése virtuális környezetben és alkalmazása LOD-ra”, *IEEE-tranzakciók a vizualizáció és a számítógépes grafikán*, vol. 15, sz. 1. 6–19. o., 2009. jan.

**Andreas-Alexandros Vasilakis** PhD

fokozatát számítógépes grafika szakon szerezte a görögországi Ioannina Egyetem Számítástechnikai és Mérnöki Tanszékén Prof. Ioannis Fudos felügyelete mellett. Ugyanebben az intézményben MSc és BSc diplomát is szerzett 2008-ban, illetve 2006-ban. 2014 márciusa óta csatlakozott az Athéni Közgazdasági és Üzleti Egyetem Informatikai Tanszékének Grafikai Csoportjához, ahol

jelenleg interaktív framebuffer technikákon, valamint inverz világítási és globális megvilágítási problémákon dolgozik.

**Georgios Papaioannou** 1996-ban BSc fokozatot szerzett számítástechnikából és PhD fokozatot számítógépes grafikából és mintafelismerésből 2001-ben, mindenkorától az Athéni Egyetemen, Görögországban. Jelenleg az Athéni Közgazdasági és Üzleti Egyetem Informatikai Tanszékének adjunktusa, és kutatásai a valós idejű számítógépes grafikai algoritmusokra, a fotorealisztikus renderelésre, a virtuális valóságra, a 3D-s mintafelismerésre és a számítástechnikára összpontosítanak.

régészeti. 1997 óta számos nemzeti és uniós finanszírozású kutatás-fejlesztési projektben dolgozott tudományos munkatársként és kutatóvezetőként. Papaioannou professzor az ACM, a SIGGRAPH, az Eurographs Association tagja, és számos konferencia programbizottságának tagja a fenti területeken.

**Ioannis Fudos** 1990-ben a görögországi Pátra Egyetemen szerzett számítástechnikai és informatikus diplomát, 1993-ban és 1995-ben pedig a Purdue Egyetemen szerzett MSc és PhD fokozatot számítástechnikából. A Ioannina Egyetem Számítástechnikai és Mérnöki Tanszékének docense. Kutatási területei közé tartozik az animáció, renderelés, morfizálás, CAD rendszerek, visszafejtés, geometria fordítók, szilárd

modellezés és képkeresés. Jól ismert folyóiratokban és konferenciákon publikált, valamint lektorként szolgált különböző konferenciákon és folyóiratokban. Finanszírozásban részesült az Európai Bizottságtól, a Görögországi Kutatási és Technológiai Fótiitkárságtól, valamint a görög Nemzeti Oktatási és Vallásügyi Minisztériumtól. Tagja az IEEE-nek.

AFÜGGELÉK

Ebben a függelékben analitikus pszeudokódos leírást adunk a k -mélységi szortírozó algoritmus (további részletek a 4.4. szakaszban), a frissített rendszer lényeges összetevője k -puffer pipeline (lásd még: 5. ábra és 3. algoritmus).

4. algoritmus k -mélységi rendezés (tömb a , Pixel p , Ints)

```

1: Tömb [ $s$ ];                                . helyi hosszúságú tömb kiosztás
2: számára ért: =0nak nek  $d.p$ .számláló / secsináld 3:          . összes iteráció
   sz:=0; [ $s$  -1].z :=1.0; előző max z=0.0; számára j:=0
4:  nak nek  $p$ .számlálócsináld           . bejárni a globális memóriát
5:     ha a[j].z < előző max zakkor      . selejtezés feldolgozott töredékei
6:       folytatni;                      .
7:     vége ha
8:   haszám < s -1akkor
9:     sz++ := a[j];                     . beszúrás kitöltetlen tömbbe
10:  különben ha a[j].z < [ $s$  -1].zakkor
11:    max([keresése] := a[j];           . beszúrni a teljes tömbbe
12:  vége ha
13: véget ért
14: előző max z := [ $s$  -1].z; hasz
15: ≤16akkor                                . hibrid válogatás
16:   beillesztés_rendezés( /, sz);
17:   más
18:   shell_sort ( /, sz);
19: vége ha
20: számítási_effektus ( /, sz);           . töredék összetétele
21:véget ért
. ahol find max() a maximális érték indexét adja vissza/

```
