GÁBOR DÉNES FŐISKOLA

Valós idejű Sugárkövetés

103/2019

Erdős Zoltán

Budapest

2020.

Tartalomjegyzék

1. Bevezetés:

Azért a sugárkövetés témát választottam szakdolgozatnak, mert érdekelt, hogy meg tudom-e csinálni ezt a feladatot. C# nyelven nem találtam forráskódot ebből a témából. Illetve most aktuális ez a téma, pl. a következő Playstation5 támogatni fogja a sugárkövetést hardveresen. Érdekel ez a téma.

Régóta létezik 3D-s megjelenítés. A számítógépek teljesítménye nem képes valós időben fénykép minőségű kép előállítására, ezért közelítő megoldásokat találtak ki. A közelítő megoldások sokkal gyorsabbak, elfutnak kisebb teljesítményű számítógépeken, viszont nem élethű képet adnak eredményül.

Az 1995-ös években az akkori játékok a háromszög csúcsainak adataiból számolták ki egy pixel színét a barycentrikus koordináták segítségével (VertexShader), ez gyors. Majd a 2002-es években lehetett a pixelek színét (PixelShader) egyedileg programozni, ez lassabb, de jelenleg ez is elég gyors már. Itt is még csak korlátozott adatok álltak rendelkezésre egy pixel színének kiszámításához. Nem volt információ (egy pixel színének számításakor) arról, hogy a legközelebbi háromszög, ami a képernyőn megjelenik, az mögött mi van.

Mostanság a 2016-os években annyira megnőtt a számítógépek teljesítménye, hogy lehet alkalmazni a „sugárkövetést”. Így nem csak a legközelebb álló háromszög adatait ismerjük, hanem a mögötte lévőket is el tudjuk érni. A „sugárkövetés”, amit be szeretnék mutatni, ez a megoldás a mai számítógépeken elfogadható sebességgel fut, élethűbb képet lehet elő állítani vele. Van tükröződés és törés, amivel, ha egy pixel színét számolom, akkor a szomszédos testekről visszaverődő fény színét is számításba tudom venni. De még ez a megoldás sem fénykép minőségű, hiszen sugárkövetésnél pontosan egy sugarat indítok tükröződés és törés irányba. Míg a valóságban van egy kis fény szóródás, mivel a felületek, amin pattan vagy törik a fény, nem tükörsima.

A „Globális illumináció” az a megoldás, ami túlmutat a jelenlegi sugárkövetésen, és még élethűbb képet állít elő, de az gépigényesebb mint a sugárkövetés.

2. Klasszikus DirectX9 és OpenGL2:

Ide vehető a 2002-ben megjelent VertexShader és PixelShader. Nem látni a háromszögek mögé, csak a legközelebbi háromszöget látjuk.

3. Újítás: Valós idejű DirectX RayTracing és RadeonRays:

2016-ban, jelentek meg az első valós idejű sugárkövetés megoldások. AMD oldalon, ami platformfüggetlen, a „RadeonRays SDK” jelent meg, ami CPU, OpenCL vagy Vulkan segítségével számol. A Microsoft a „DirectX12 RayTracing SDK”, ami naprakész, viszont (úgy tudom) csak Windows 10-en fut. Az Intelnek és az NVidia-nak is vannak sugárkövetés SDK-juk.

4. Saját Sugárkövetés program:

Ha vannak nagy cégektől sugárkövetés SDK-k, akkor miért írok sajátot (ami butább)? Kíváncsiságból. Azért is írom, hogy megértsem ennek a működését.

A forráskód amit írtam/írok, letölthető a https://github.com/ezszoftver/OpenCLRenderer weboldalról. OpenCL-t használok a párhuzamos számításokhoz. Szabadon felhasználható, módosítható a forráskód, nincs licensz védelem alatt.

4.1 Elmélet röviden:

A sugárkövetés „futószalagja” elméletben így működik:

1. sugarakat indítok a kamerából, amik elmetszhetnek háromszögeket.
2. ha egy sugár-háromszög metszés van, akkor abból az ütközéspontból kiszámolom, hogy mennyi fény éri azt a pontot, majd újabb sugarakat indíthatok tükröződési és törési irányba. Ez a sugár újra elmetszhet egy háromszöget, és kezdődik ez a lépés újra.
3. Kb. 3 ütközés után abba hagyom az ütközés keresést, meg vannak a szín információk, amiből a képernyőn megjelenő pixel színét ki tudom számolni. Azért hagyom abba a további ütközéskeresést, mert, ha valós idejű képet szeretnék előállítani, akkor tovább folytatni gépigényes. Így is jobban megközelítem a valóságot, mint a 2002-es DirectX9/OpenGL2 -vel.

Elméletben ennyi. Vajon mik azok a megoldások, amivel rövid idő alatt el lehet a „futószalagot” végezni? Most ezt mutatom be.

Fontos! Amit leírok megoldásokat, nem a leggyorsabb megoldások, én ezeket ismerem, ezeket tudom bemutatni.

4.2 Gyorsítási lehetőségek:

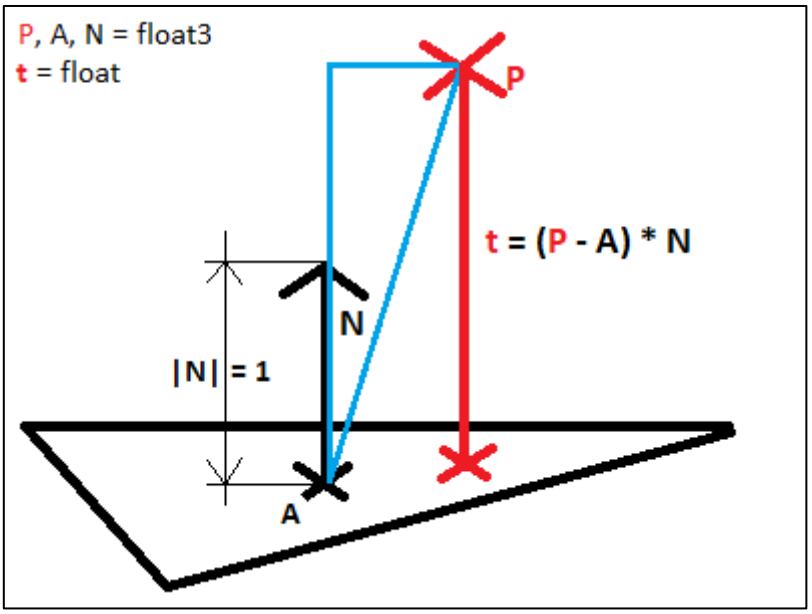
4.2.1 Sugarak-háromszögek ütközésvizsgálata, párhuzamosan:

Először be szeretném mutatni, hogyan lehet egy „sugár-háromszög” metszéspont vizsgálatot elvégezni. Majd bemutatom, hogyan lehet, ha sok sugarunk van, ezt gyorsítani (elöljáróban annyi, hogy ahány sugarunk van, annyi szálat kell indítani OpenCL segítségével, és úgy elvégezni a „sugár-háromszögek” metszéspont vizsgálatot). Az „OpenCL” script kód nem a CPU-n, hanem a videokártyán fut.

A „sugár-háromszög” metszéspont vizsgálat lépései:

- „pont-sík” távolsága
- „sugár-sík” távolsága
- metszéspont kiszámítása
- metszéspont a háromszögen belül van? vizsgálat

pont-sík távolsága:



1. ábra: pont-sík távolsága [saját]

Először „float t” -t kell kiszámolni. Tekintsük úgy most a háromszöget, mintha az egy sík lenne. Egy síkot meghatároz egy „**float3 A**” pont, amit a sík elmetsz, és egy „**float3 N**” irány, hogy merre néz a sík.

Ha a „**float3 (P - A)**” vektort skalárisan össze szorozzuk az 1.0 egységnyi hosszú „**float3 N**” vektorral, akkor megkapjuk „float t” -t.

Két „**float3 a,b**” **egységnyi** hosszú vektorok **skaláris szorzat**ára, igaz ez a képlet:

a * b = cos(alpha)

„(P - A)” és „**N**” vektorok skaláris szorzatának képlete:

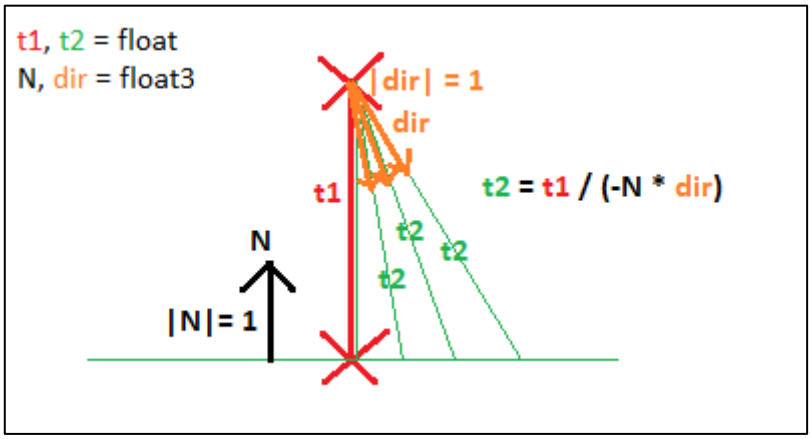
(P - A) * N = cos(alpha) * |P - A|

cos(alpha) eredménye, **[-1.0 .. +1.0]** intervallum béli számot ad eredményűl.

Tehát **|P - A|** hosszt összeszorozzuk egy +1.0 -nál kisebb számmal, így eredményül egy

|P - A| -nál rövidebb, „t” hosszt ad eredményül, ami a sík és a „P” pont távolsága.

sugár-sík távolsága:



2. ábra: sugár-sík távolsága [saját]

Ha meg van a t távolság, akkor, ha figyelembe vesszük azt, hogy a „P” pontnak van iránya is, akkor egy félegyenest kapunk. Ha az irány megegyezik a -N -el, akkor az a legrövidebb távolság. Ahogyan a „t2” -k mutatják a 2. ábrán úgy, ha minél nagyobb a „-N, dir” közötti szög, annál hosszabb „t2” -ket kapunk eredményűl.

metszéspont kiszámítása:

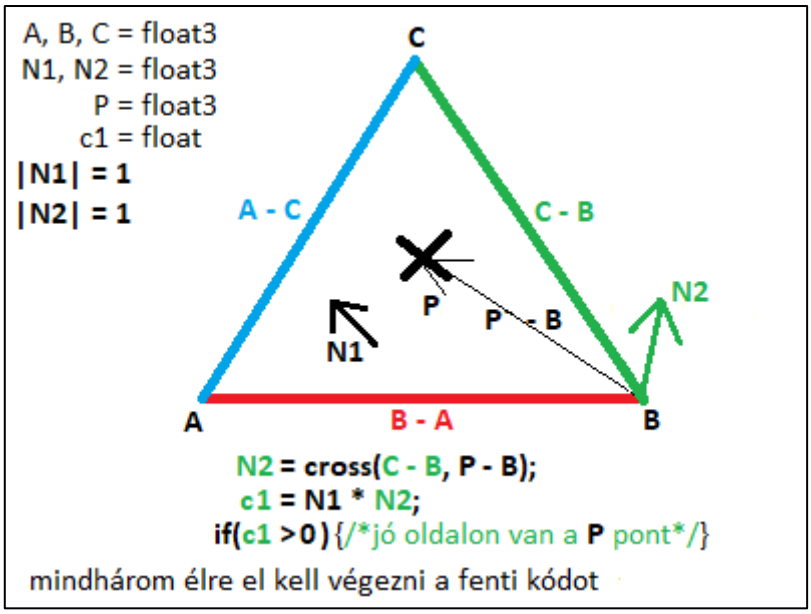
Azt a pontot, hogy a félegyenes hol metszi el a síkot ezzel a képlettel kapjuk meg:

P2 = P + (dir * t2);

„float3 P2” az a pont, amit a sík elmetsz. A „float3 P” és „float3 dir” a félegyenes (sugár) kezdőpontja és iránya.

A „float3 dir” hossza, 1 egység.

metszéspont a háromszögen belül van?:



3. ábra: a metszéspont a háromszögen belül van? [saját]

A harmadik lépés, hogy a síkot metsző pont a háromszögen belül van-e? Én azt a megoldást választottam, hogy vektoriális szorzattal ellenőrzöm ezt. A háromszög mindhárom élére el kell végezni, a „jó oldalon van a pont?” ellenőrzést. Én egy élre mutatom be az ellenőrzést:

Vegyünk B-C szakaszt. Ha vektoriálisan össze szorzom (C - B) és (P - B) vektorokat, akkor a képen látható N2 vektort kapom eredményül. Fontos, hogy a vektoriális szorzat nem kommutatív, vagyis, ha nem ebben a sorrendben, hanem fordítva végzem el a vektoriális szorzatot (cross(P - B, C - B)), akkor -N2 -t kapok. Majd szög ellenőrzéssel (skaláris szorzat) ellenőrzöm, hogy jó oldalon van-e a P pont. Összeszorzom skalárisan a háromszög normal vektorját(N1), az N2-vel. Ha a két normalvektor közötti szög kissebb mint 90fok, akkor a „P” pont, a „C - B” él jó oldalán áll. Mind a három élre igaznak kell lennie, hogy a szög kisebb-e mint 90fok. h Ekkor a „háromszögen belül vagyok-e?” kérdésre a válasz, igaz. A vektoriális szorzatnál figyelembe kell venni azt is, hogy nem mindegy, hogy (C - B) vagy (B - C), mert, ha felcserélem a pontokat, akkor ellenkező irányba fog mutatni a vektor. Én az óramutató járásával ellentétes irányt választottam.

Tehát el tudok mostantól végezni a félegyenes-háromszög ütközésvizsgálatot. Félegyenes alatt sugarat, vagy ray -t is mondhatok, mindhárom szó most ugyanazt jelenti. Amikor egy képet akarok előállítani sugárkövetéssel, akkor minden egyes pixelből sugarat indítok a világba. Tehát nagyon sok sugarat kell indítanom. Két

pixelnek a textúrából, nincs köze egymáshoz, tehát két sugárnak sincs köze egymáshoz. Nincs függőség, vagyis pl. a (0,0) pixelből indított sugár, nem várakozik a (0,1) pixelből indított sugárra, tehát párhuzamosan, külön-külön szálakon lehet elindítani a sugár-háromszög metszésvizsgálatot.
Egy videokártyában kb. 100-2000 darab kis órajelű (500Mhz – 1GHz) processzor van. Ezeket a processzorokat el lehet érni C nyelven, „OpenCL" segítségével. Tehát tudok párhuzamosítani „hardveresen". A CPU abban más a GPU-k tól, hogy magasabb órajelen működnek, kevesebb van belőlük (1 – 16 mag), viszont összetettebb, az egész számítógépet vezérlik. CPU-n lassabban fut egy kép számítás, mint egy videokártyán. Pont azért találták ki a videokártyát, vagyis egy külön hardvert képszámításra, mert az a képszámításra van optimalizálva, gyorsabban kiszámolja a képet.

OpenCL-ben, „Buffer" -ekben vannak tárolva az adatok. Egy „Buffer" osztály, más néven tömb. Meg lehet adni a „Buffer" -nek, hogy milyen típusú adatokat akarok benne tárolni: Buffer<int> egesz_szamok;. Ez a buffer a videokártya memóriájában jön létre. Általában van egy bemeneti Buffer, amivel számol, és van egy kimeneti Buffer, ahova az eredmények kerülnek. Majd a Buffer-t ha kell, vissza lehet másolni az operációs rendszer memóriába, és lehet az eredményekkel tovább számolni.

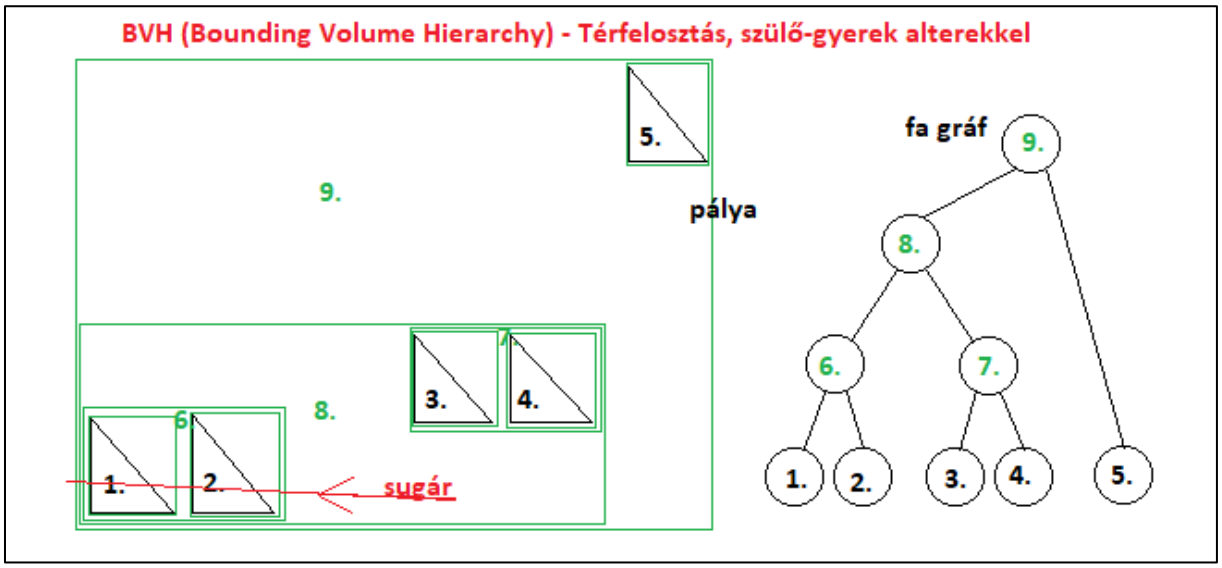
Tehát van sok sugaram, ez egy „Buffer" (Buffer<Ray> rays). Illetve van a háromszögeket tartalmazó „Buffer" (Buffer<Triangle> triangles). Ezek a bemenő bufferek. Kell egy kimeneti buffer is, ami megmondja, hogy egy sugár elmetszette-e a „triangles" bufferben lévő háromszögek valamelyikét. Ezek az adatok az eredmény (Buffer<hit> hits). „hits" Buffer annyi elemet tartalmaz, ahány sugarunk van.
Két párhuzamosan, külön-külön szálon futó Ray, ugyanazt a „triangles" buffert használja, baj ez? Nem, mert csak olvasnak belőle, nem módosítják.
Egy videokártya, kb. 10x gyorsabban kiszámol egy képet a párhuzamosítás miatt, mint egy vele egyenértékű CPU.
A sugarakat párhuzamosítottuk. Most nézzük meg, hogy mit lehet tenni a háromszögekkel, ott is gyorsítani kéne.

4.2.2 Ütközésdetektálás gyorsítás, tér felosztással (BVH):

Eddig egy sugár végig járta, az összes háromszöget, és úgy kereste a hozzá legközelebbi, elmetsző háromszöget. Nem lehet ezen gyorsítani? De lehet, tér felosztással.

Képzeljük el, hogy van a világ. Az legyen mondjuk 1km hosszú, 1km magas, 1km széles). Itt vannak a háromszögek elszórva. Van egy sugarunk, pl. (10,10,10) pontban. Felesleges azokkal a háromszögekkel ütközéstdetektálást végezni, amik nagyon messze, pl. (1km, 1km, 1km) távolságra vannak a Ray-től. Jobb lenne, csak a Ray-hoz közeli háromszögeket vizsgálni. Megoldás, daraboljuk fel a teret, minden altérbe másoljuk bele, a teret elmetsző háromszögeket. Majd, ha jön egy sugár, akkor számoljuk ki hogy a sugár megy tér részeket metsz el, és csak az azokban a tér részekben lévő háromszögekkel végezzünk ütközés keresést.

Én itt, a BVH (Bounding Volume Hierarchy) megoldást választottam. Jelentése: alterek, szülő-gyerek kapcsolatban, vagyis hierarchiában.



4. ábra: BVH (alterek a gyors kereséshez) [saját]

A kép bal oldali része az altereket, világot mutatja jól, míg a kép jobb oldali része, a gráfot, a szülő gyerek kapcsolatokat mutatja jól.

Az 1,2,3,4,5 csomópontok, háromszögek, míg a 6,7,8,9 csomópontok, tér részek (bounding box). A 9. csomópont, a gráf szerint, az a gyökér, az az egész világot magába foglalja.

Vizsgáljunk a sugár szemszögéből, ami a 4. ábrán van: amit elmetsz az a 9, 8, 6 -os alterek, és az 1, 2 -es háromszögek. Felesleges vizsgálni a 3, 4, 5-ös háromszögekkel a metszésvizsgálatot.

Hogyan kapjuk meg az 1 és 2-es háromszögeket? Kezdjük a metsző háromszögek keresést a fa gráf bejárásával.

- A sugár elmetszi a gyökeret? (9) => igen, tehát vizsgáljuk meg a 9. csomópont gyerekeit.
- A sugár elmetszi a 5 alteret? nem, tehát erre nem folytatjuk a keresést.
- A sugár elmetszi a 8-as alteret? igen, vizsgáljuk meg ennek az altérnek/csomópontnak a gyerekeit.
- a sugár elmetszi a 7-es alteret? nem, erre nem keresünk tovább.
- a sugár elmetszi a 6-os alteret? igen, akkor vizsgáljuk ennek az altérnek/csomópontnak a gyerekeit.
- a sugár elmetszi az 1 háromszöget? igen
- a sugár elmetszi a 2-es háromszöget? igen

Levél: Az a csomópont, akinek nincsen gyereke (a fa gráf alja), vagyis egy háromszög van benne.

Ha eljutottunk egy levélig, akkor háromszög-sugár metszést kell vizsgálni.

Ha nem levélben vagyunk, akkor „sugár-altér” (bounding box) metszésvizsgálatot kell csinálni.

Mindkét háromszögre megkapjuk a „t” távolságokat. Nekünk a kisebb értékű „t” (háromszög) kell, számoljuk ki milyen textúra szín és fény éri azt a pontot, és tároljuk el azt a színt.

Ez a térfelosztás azért jó, mert, ha pl. 1millió háromszögből áll egy pálya, akkor az 1millióhoz képest kevés altér vizsgálattal eljutok a „nagy valószínűségű, hogy metsző” háromszögekig.

- Tegyük fel, hogy van 1millió háromszögem. Ha nem lenne térfelosztás, akkor egyesével, minden háromszöget vizsgálni kéne, ez 1 millió vizsgálat.
- De ha BVH segítségével keresünk, akkor kb. 100 vizsgálattal megkapom a metsző háromszögeket. Kevesebb így a háromszög metszéspont keresés vizsgálat.

Hogyan lehet egy háromszögek listából, BVH fát felépíteni?:

Adottak a háromszögek. Első lépésben veszek egy háromszöget, és megkeresem a hozzá legközelebbi másik háromszöget. „Csúcs-csúcs” vizsgálat elég, mert általában a felületek folytonosak, zártak mindig van egy háromszögnek egy olyan csúcsa, ami egy

másik háromszöghöz is tartozik. Ebből a két szomszédos háromszögből, egy csomópontot lehet csinálni. A csomópontot egy „List<Node> nodes” listába teszem és a két háromszöget törlöm a „háromszögek listájából”. Majd veszem a következő háromszöget a háromszögek listájából, és ugyan ezt a „szomszéd keresés” algoritmust futtatom, majd a megszületett csomópontot hozzá fűzöm a „nodes” listához. Addíg keresem egy háromszög szomszédos háromszögeit, amíg vannak háromszögek a „triangles” listában. Előfordulhat, hogy csak egy háromszög maradt a listában, annak nem tudok szomszédot találni, így belőle egy olyan csomópontot hozok létre, aminek csak egy gyereke van.
a „nodes” listában, most sok kicsi altér van. Ezekkel az alterekkel ugyanúgy elvégezzük a szomszéd keresést, ugyan úgy, mint a háromszögeknél. Mindig, az újjonan keletkező csomópontokat bele tesszük egy új „List<Node> out” listába, majd ha az „List<Node> in” listából, ha elfogytak a csomópontok, akkor az „in = out; out = new List<Node>();” (az out lista az in listába kerül, és egy új, üres out Lista jön létre), és kezdődik elölről a szomszédok keresése.
Egyszer eljutunk egy olyan állapothoz, amikor az in Listában egy elem lesz. Az lesz a gyökér elem.
(Amikor létre hozunk egy csomópontot, akkor mindig kiszámoljuk a gyerekei altérből, az aktuális alteret, amiben mindkét gyerek altér benne van).
Így létre jön egy BVH fa.

Ez a gráf addig „jó”, amíg a háromszögek mozdulatlanok. De a számítógépes grafikában a háromszögek mozognak. Pl. animáció. Hogyan lehet egy már felépített fát, amiben, ha elmozdul egy háromszög (valamelyik csúcsa), akkor újra „jóvá” tenni? Lehet ezt párhuzamosan számolni? Igen. Az altereket (bounding boksz-okat) kell újra számolni.
A következő rész egy fa „újra jóvá tételét” mutatja be.

4.2.3 BVH gyors újra építése, ha a csúcsok megváltoztak:

Mi van akkor, ha egy BVH fa háromszögeinek csúcsai transzformálódott? Újra kell építeni az egész fát? Nem.
Két eset lehet:

- Animációkor, a háromszögek szomszédsága megmarad, pl. felemeljük a kezünket. Hagyományos animációkor ez az állítás érvényes.
- Animációkor a háromszögek szomszédsága nem marad meg, pl. levágják egy ember kezét, és messzire eldobják.

Én azt a megoldást választottam itt, hogy mindkét esetben a háromszögek szomszédságán nem változtatok a BVH-n belül.

Első esetben ez nem gond, a háromszögek szomszédsága ugyanúgy megmarad, csak az altereket kell a gyerekektől a szülők felé újra számolni.

Második esetben, „amikor messzire repül a kéz", akkor is csak az altereket számoljuk újra, igaz ilyenkor nem lesz optimális a BVH bejárás, mert több 10 méter is lehet a távolsága a kéz és ember között, pedig biztosan lenne közelebbi háromszög.

Sajnos nem tudok megoldást, amivel gyorsan a háromszögek szomszédságát újra lehetne számolni. De ha a csomópont altereket újra számoljuk, függetlenül attól, hogy nem tökéletes a szomszédság, így is gyorsabb a fa bejárással, a metsző háromszögek keresése, mintha egyesével járnánk végig az összes háromszöget, metszéspontot keresve.

Megoldás: „ne szakadjon le a kéz". Maradjon meg a szomszédsági viszony. De ez nem lehetséges (mindig).

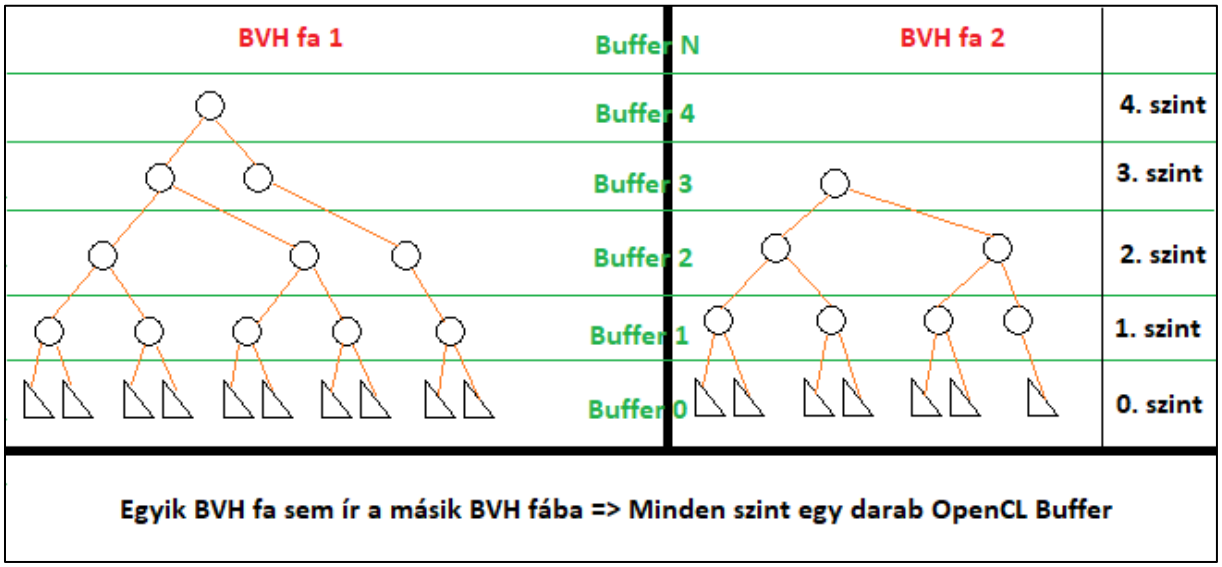
Az altereket hogyan lehet párhuzamosan számolni?:

- tudjuk azt egy fa gráfról, hogy vannak szintjei. Én most megfordítom a szintek sorszámozását. Legyen a 0. szint a levelek, vagyis a háromszögek szintje. az 1. szint, a háromszögek szülői, … az N. szint pedig a gyökér.
- Tudjuk azt is, hogy egy pl. 5. szintet csak akkor tudom párhuzamosan számolni, ha a 4. szint alterei már kiszámításra kerültek.

Tehát a 0. szintű háromszögekből inicializáláskor készítek egy OpenCL „Buffer<Node> level0" buffert, ő egyben in/out buffer és kiszámolom a háromszögek altereit OpenCL-el. Így a 0. szint altereit kiszámolta az OpenCL, párhuzamosan, el lehet kezdeni az 1. szintű Node -k altereinek számolását. Fontos: Egy Node egyszerre altér és háromszög. Onnan tudom hogy egy Node levél (vagyis hogy háromszög), hogy nincs egy gyereke sem.

Minden szint egy „Buffer<Node>”. Ezt előfeldolgozási lépésben ki lehet számolni, hiszen egy BVH fa szintjei mindig ugyan azok maradnak, csak a levelek változnak.

Sorra kiszámolom a 2,3,4,5 … N -edik szintig párhuzamosan a szintek csomópontjainak altereit OpenCL-el, a gyerekek altereiből. Így frissítettem egy BVH fát.



5. ábra: BVH fák szintjei. Egy szint elemei párhuzamosíthatók OpenCL-el [saját]

Sőt ahogy a kép is mutatja, ha sok BVH fa van, legyen mondjuk 2 darab (vagy több), vagyis több objektum van a világban. Amikor készítem pl. 0. szintet, akkor össze lehet fűzni mind a 2 darab 0.ás szintű level-eket, és egy nagy level0 Buffer keletkezik. Ugyanígy level N-ig. Mivel egyik BVH fa sem ír/olvas a másik BVH fa Node -jéből/-jébe, függetlenek egymástól, ezért összefűzhetők, párhuzamosíthatók.

Egy fa szintje, kb. 15-25 lehet. Ha kiszámoljuk, ha 25 szintű a fa, akkor elfér benne (2^25) 33 millió háromszög a levelekben, BVH-nként. És csak 25 darab OpenCL függvényhívást kellett a megfelelő sorrendben meghívni, akkor is, ha több BVH fa van, és újra „jóvá” tettük a fát.

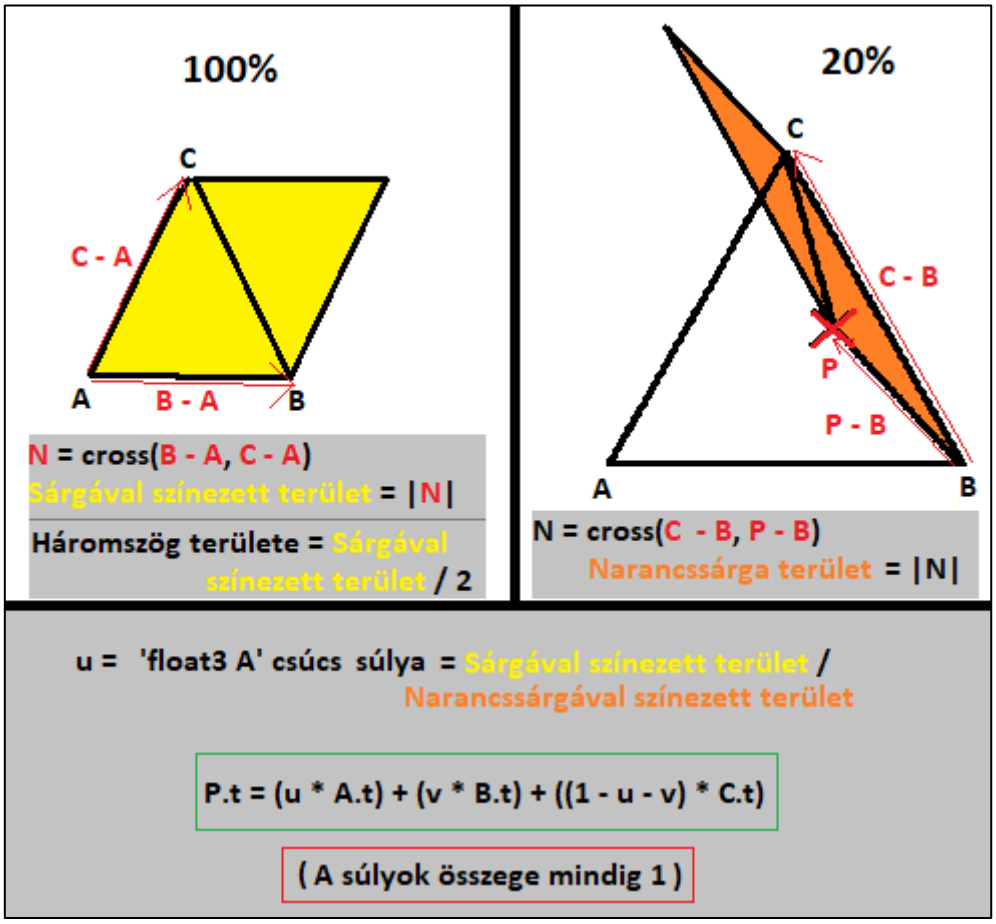
Külömbséget kell tenni animált és nem animált BVH fa között. Az animált BVH fa „Dynamic” típusú, a nem animált BVH fa „Static” típusú. Elég csak a „Dynamic” típusú BVH-knak a altereit frissíteni. Azt hogy egy BVH fa Static

vagy Dynamic lesz-e, azt a programozó dönti el. Egy Static típusú BVH-nak sohasem változnak meg a háromszög csúcsainak pozíciói, így azt elég egyszer, inicializáláskor az altekreket frissíteni.

Statikus BVH-nak számít a mozdulatlan pálya, míg Dynamic BVH-nak számít az animált-, vagy a térben máshová kerülő tárgyak.

Az OpenCL level1,2, .. 25 Bufferekbe csak a Dynamic típusú BVH fákat kell bele tenni.

5. Textúrázás Barycentrikus koordinátákkal:



6. ábra: textúrakoordináta számítása, a P pontban [saját termék]

Ha ismerem az A-, B, és C csúcshoz tartozó textúra koordinátákat, és ki szeretném számolni a P metszéspont textúra koordinátáját, azt hogyan kell? Súlyokkal.

Képzeljük el, hogy ha egy P metszéspont közel van az A csúcshoz, akkor az A csúcs textúra koordinátához közeli értékű lesz a P csúcs textúra koordinátája. Minél messzebb kerül a P pont az A csúcstól, és minél közelebb kerül a B csúcs felé, annál inkább a B

csúcs textúra koordinájához közeli értéket fog a P csúcs textúra koordinátája fel venni. Ugyan ez a C csúccsal.

Súlyokat kéne létre hoznom, amik megmondják [0.0 .. 1.0] intervallumban, hogy milyen közel vagyok egy csúcshoz. Ha nagyon közel vagyok pl. az A csúcshoz, akkor az A csúcs súlya 1.0-hoz közeli szám, és a B és C csúcsok súlya 0.0-hoz közeli szám. Az A csúcs textúra koordinátája fog jobban részt venni a P csúcs textúrakoordinátájának számítása közben, a B és C csúcsok, közel 0%-al fognak részt venni. A súlyok összege 1.0-et kell hogy kiadjon.

Hogyan tudom az A csúcs „float u" súlyát kiszámolni? Ahogyan a kép is mutatja, ha kiszámolom a teljes háromszög területét (A, B, C csúcsok), ez legyen „t1". Majd, ha kiszámolom az A csúccsal szemközti (P, B, C csúcsok) háromszög területét, legyen ez „t2". Majd „float u = t2 / t1". Így egy kisebb számot osztok egy nagyobb számmal, pont az A csúcs súlyát fogom megkapni. A mellékelt ábra az A csúcs súlyának kiszámítását mutatja be.

Számoljuk ki a B csúcsal szemközti kis háromszög területét, ez legyen „t3". Majs a súly: „float v = t3 / t1". A C csúcs súlyát ugyan ezen elven lehet kiszámolni, de felesleges, mivel tudjuk hogy „u + v + w = 1", ebből következik, hogy a C scsúcs súlya: „1 – u - v".

Ismerjük a három súlyt, alkalmazzuk, ezt képletet: „P.t = (u * A.t) + (v * B.t) + ((1 – u - v) * C.t)". Így megkapjuk a P pontban lévő textúra koordinátát. Ugyan ezzel a módszerrel, nem csak textúra koordinátát, hanem normálvektort, vagy pozíciót is lehet számolni.

Hogyan kell kiszámolni egy háromszög területét? Ahogyan a kép is mutatja, egy A, B, C csúcsú háromszög területe egyenlő

„length(cross(B - A, C - A)) / 2.0". „cross()" a vektoriális szorzat, „length()" a vektor hossza. És osztani kell kettővel.

6. Shaderek:

Az én alkalmazásomban, a címben szereplő 5 lépésre osztottam a futószalagot. Ezek OpenCL függvények.

6.1 TriangleShader:

A „TriangleShader" OpenCL függvény feladata, hogy a paraméterül kapott „Buffer<Node> inNodes" buffer háromszögeit frissítse. Ha az aktuális Node egy altér,

(tehát nem háromszög), vagy ha a Node háromszög és „Static” típusú, akkor nincs szükség frissítésre, elég csak ezt a node-t átmásolni a ” Buffer<Node> outNodes” listába.
Különben, ha egy Node háromszög (levél), és „Dynamic” típusú (módosulnak a csúcspontjai a háromszögnek), akkor a háromszög A, B, C csúcsára meg kell hívni egyesével a „VertexShader” opencl fuggvényt, ami transzformálja (a térbe máshova helyezi) az A, B, C csúcsokat. Oda helyezi a csúcsokat a VertexShader, ahova a programozó szeretné. A VertexShader megvalósítása a programozó feladata. Ha ez megtörtént, akkor a TriangleShader újraszámolja az új háromszög területét.

6.2 VertexShader:

A „Vertex Shader”, ahogy fennt írtam, paraméterként egy darab csúcsot kap. Egy csúcs több változót tartalmaz: Csúcs pozíciója, a csúcshoz tartozó textúra koordináta, a csúcshoz tartozó normal vektor. Általában elég csak a csúcs pozícióját és normal vektor-ját transzformálni a „Vertex Shader” -ben, a textúra koordináta változatlan szokott lenni.

6.3 RefitTreeShader:

A „TriangleShader” OpenCL függvényhívás egy olyan „Buffer<Node> outNodes” fát ad eredményűl, amiben a levelek (háromszögek) kiszámításra kerültek (0. szint, valid). De a gráf felsőbb szintjei, vagyis az alterek nem valid-ak, azokat újra kell számolni. A „RefitTreeShader()” opencl függvény hívással lehet az altereket újra számolni. Fontos, hogy a „RefitTreeShader” függvényhívást előzze meg a „TriangleShader()” függvényhívás. Elég csak akkor meghívni a RefitTreeShader() függvényt, ha a világ rendelkezik „Dynamic” típusú objektummal (háromszöggel).
Ahogy feljebb írtam, a gráfokból, szintenként egy-egy „Buffer<Node> inoutLevelN” buffer létrehozható. Tehát „level0”, „level1” „level2” … kb „level25” buffer elég a gráfok tárolásához. A buffereket a megfelelő sorrendben kell meghívni. Először csak a háromszögeket tartalmazó „level0” buffer-t kell paraméterül átadni a „RefitTreeShader()” OpenCL függvénynek. A háromszöget tartalmazó buffer, a fák leveleit tartalmazza, nincsen egy Node csomópontnak egy gyereke sem, nem függ a Node-ben található altér, gyerekeitől, mert nincs gyereke a levélnek. Mivel nincs függőség, ezt a buffert lehet számolni. A „RefitTreeShader(level0)”-nek ezt a buffert átadva, frissíti az altereket (bounding box). Ha a „Buffer<Node> level0” alterei frissítve lettek, akkor a gráfban, az egyel felette lévő szintet (level1) lehet számolni, mert a „level1” alterek, csak az egy

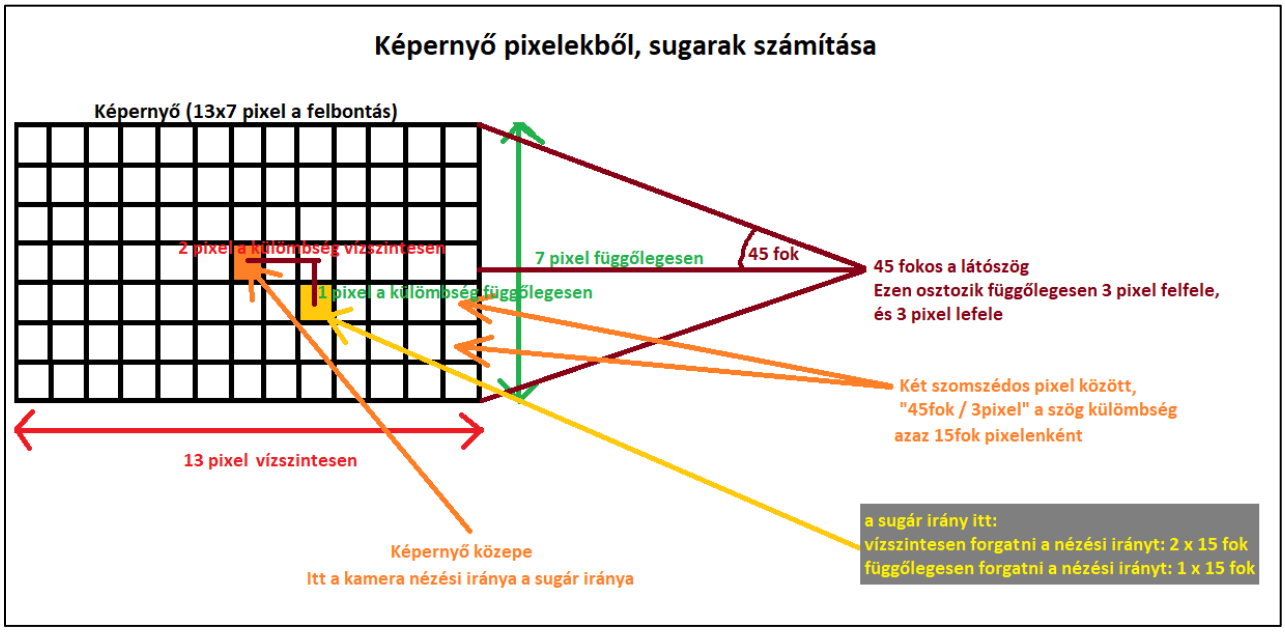
szinttel alatta lévő, (level0) alterektől függ. Hívjuk meg a „RefitTreeShader(level1)” függvényt, aminek paraméterül az 1. es szintet adjuk. Majd így folytatva, hívjuk meg a függvényt, paraméterként megfelelő sorrendben a 2., 3., … N. bufferekkel. Így a gráf Node-jeinek, alterei helyes értékeket tartalmaznak. Fontos, hogy ez a megoldás gyorsabb, mint ha CPU- val végeznénk egyesével a csomópontok altereinek újra számolását. A „RefitTreeShader()” OpenCL függvény párhuzamosan számolja a paraméterül kapott Node-ket, max. kb. 25 függvényhívással, míg CPU-n elvégezve, ez akár több ezer egymás utáni számítás is lehet.

Most világban lévő háromszög adatok validak.

6.4 GenerateCameraRays Shader:

Kamera pozícióból és látószögből sugarakat hoz létre ez az OpenCL függvény.

Ahhoz hogy a monitoron egy pixel színét megkapjuk, szükség van egy kiinduló sugárra, ami úgymond a monitor pixeléből indul, megkeresi a háromszögek között a legközelebbit, kiszámolja a színt, majd tükör-, és törési- irányba újabb sugarat indít, amivel szintén a legközelebbi háromszöget keresi, abból kiszámolja megint a metszéspontot érő fényt, majd újabb sugarakat indít tükröződési- törési- irányba, és így tovább. Úgy gondolom, elég 2x-3x egy képernyő pixeléből kiinduló sugárral új sugarakat generárni, és új színt keresni, mert ez gépigényes feladat még a videokártyának is.



7. ábra: Képernyő pixelekből, sugarak előállítása [saját]

Tehát először is szükség van a monitor pixeleiből, kiinduló sugarakat létre hozni. Erre van a „GenerateCameraRays()” OpenCL függvényhívás.

Paraméterül megkapja a függvény egy felbontást (pl. 640x480), így tudni lehet, hogy mennyi sugarat kell létre hozni. Paraméterül kap még egy kamera pozíció és nézeti irányt. Ezekből az értékekből tudja a függvény, hogy a felbontás közepén lévő pixel-nek mi a kiinduló pontja és iránya. Ha 640x480 a felbontás, akkor a képernyő közepén (320x240. pixel) található sugár kiinduló pontja a kamera pozíciója, a sugár iránya pedig a kamera nézeti iránya. Tehát a képernyő közepén leévő sugár mindig meghatározható, minden egyes sugár létrehozáskor. Minden egyes pixelből ki lehet számolni, hogy hány pixel távolságra van a képernyő középpontjától (középpont: 320x240). Ha tudjuk a távolságot, akkor ha a képernyő közepén lévő sugár nézőpontját ennyivel eltoljuk, akkor lehet generálni bármelyik pixel-ből, sugarat.

Kérdés, hogy két szomszédos pixel között, mennyi a sugár nézeti pont eltolása? A programozó azt megadja, hogy a világot hány fokos látószögű kamerával szeretné nézni. (Az emberi 45fok-os szögben lát, ezt az értéket szokták használni általában a programozók). Ha a felbontás függőlegesen 480 pixelből áll, és tudjuk azt, hogy a képernyő közepe, vagyis a 240. pixelnél 0fok az különbség és és függőlegesen a 0. pixel -hez a 45fok tartozik, és függőlegesen a 480. pixelhez szintén a -45fok tartozik, akkor a függőlegesen a köztes pixelekhez [1 .. 239, 241 .. 479] lehet tudni hogy milyen fok tartozik hozzá.

A képernyő közepén lévő 0fokhoz tartozó kamera nézeti irányt most már tudjuk minden pixelre, hogy mennyivel kell elforgatni függőlegesen.

Ha tudjuk hogy 45fokon osztozik 240 pixel (640x480-as felbontáson), akkor két szomszédos pixel között, 45/240 elforgatás tartozik.

Ebből kiindulva ki lehet számolni a vízszintes [0 .. 640] pixelekhez tartozó elforgatást is.

Ha tudjuk hogy egy pixelhez mekkora szögelfordulások tartoznak, akkor a képernyő közepén lévő sugár irányát elforgatjuk ezekkel a fokokkal, így magkapjuk bármelyik pixel sugarának irányát. Minden képernyőből induló sugár kiindulópontja a kamera pozíciója.

A képernyő sugarakat egy „Buffer<ray> rays” bufferbe el kell tárolni, mert szüksége lesz a „RayShader”-nek ezekre.

A sugarak létrehozása egymástól nem függ, tehát ezeket is lehet párhuzamosan létre hozni. 640x480-as felbontás, 307200 pixelt tartalmaz, ekkora méretűre kell létre hozni a „Buffer<ray> rays” buffert.

Egy 2D-s pixelből, 1D-s tömbbéli indexet, ezzel a képlettel kaphatunk:

int id = (width * y) + x; // (width, height) a képernyő felbontása, (x, y) egy tetszőleges pixel. Így az 1D-s tömbbe tudjuk írni a 2D-s pixelből létrehozott sugarat.

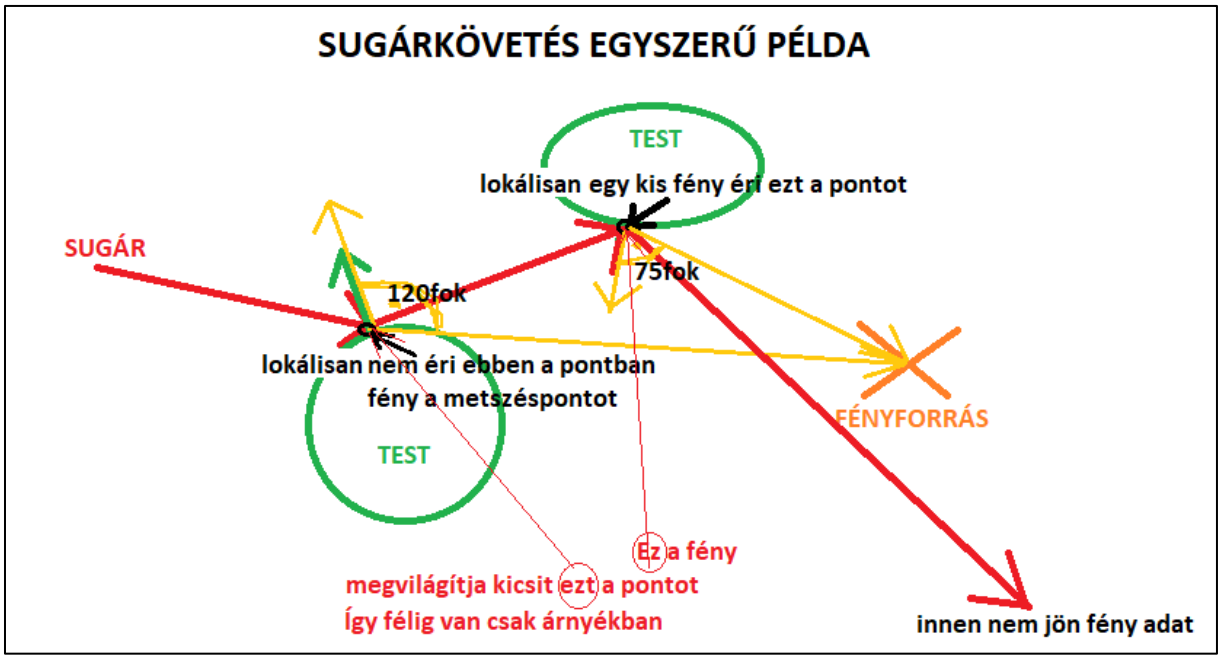
6.5 RayShader:

A ray shader eredménye egy szín. A klasszikus opengl2.0/direct3d9-nél a pixel színének kiszámítását a „pixel/fragment shader” nevű függvény végezte. Most ezt helyettesíti a „RayShader()”.

Régen csak a legközelebbi pixel pozícióját, és a fényforrásokat ismertük. Csak a fényforrásból érkező színnel lehetett egy pixel színét kiszámolni.

Sugárkövetésnél igaz, hogy a képernyőből induló sugarak és háromszögek metszéspontja ugyanúgy a képernyőhöz legközelebbi pontot adja eredményül. Szín számításkor ugyanúgy csak a fényforrások helyzetét vesszük csak számításba.

DE: sugárkövetésnél lehet folytatni a színszámítást.



8. ábra: sugárkövetés példa (csak tükröződés van benne, törés nincs) [saját]

A sugár-háromszög metszéspontban, tükör irányba és/vagy törési irányba újabb sugarakat lehet indítani, ami, ha elmetsz egy másik háromszöget, akkor azzal a metszésponttal szintén ki lehet a számítani a fényforrások helyzetéből a metszéspontot érő színt. Ezt a színinformációt ismerve, az eredeti képernyő pixel színét lehet módosítani. Így lehet például olyat csinálni, hogy:

Első sugár indításkor, és fényforrás pozíció szerint árnyékban van a pont, tehát fény nem éri ezt a pontot. De ha tükröződeési- törési- irányba újabb metszéspontot keresünk, lehet, hogy az újabb metszéspontot éri a fény, ami megvilágítja az eredeti, fényt nem érő pontot.

Tehát a fény most már tud pattogni, és olyan helyre is eljutni több pattogás közben, ami a kalsszikus opengl2.0/direct3d9 pixel shader-rel nem kiszámítható.

Az én megvalósításomban, a „RayShader()”-t is a programozó programozza. A RayShader függvény kap paraméterül egy „Hit *hits” ütközéspontokat tartalmazó listát, és egy sugarakat tartalmazó „Ray *rays” listát. Ezek a listák 2D-sak. Az első dimenzió a sugár hosszát mondja meg. Hányadjára tükröződik/törik egy pixelből indított sugár. A második dimenzióba több sugarat is tehetünk, több irányba folytathatjuk a sugárkövetést. A 2D-s tömb mérete [6][64]. tehát egy képernyőből induló sugár 6 hosszú lehet (első dimenzió), és lépésenként maximum 64 sugarat kezelhetek egyszerre (második dimenzió).

Ez a 2D-s tömb nem a legjobb megoldás, mert az 1. lépésben is lefoglalok a memóriában 64 sugárnak helyet, ami szinte biztos, hogy nincs kihasználva.
Azt az elméletet próbáltam követni, hogy: Ha minden lépésben 2 sugarat indítok (egyet tükröződési, egyet törési irányba), és minden sugár mindig ütközik háromszöggel, akkor a 6. lépésben, 2^6 = 64 különböző sugarat kell kezelnem. Tehát a 2D-s tömb (5. indexű) utolsó rekeszében kihasználom a maximum használható 64 sugarat. De az első vagy második lépésben valószínű, hogy nem kezelek 64 sugarat egyszerre.

A „8. ábra” azt mutatja, hogy egy pixel színének kiszámításához, 3 lépést használtam fel, és lépésenként 1 sugarat indítottam. Hogyis néz ez ki RayShader-ben? Paraméterként megkapom mindig a „hits”- és „rays” 2D-s tömböket. Ezeket így kell használni:

Első lépés: hits[0][0]-ba kerül a képernyő pixeléből kiinduló sugár, ezzel számolok. Ha ütközés van, akkor a „rays[1][0]”-ba helyezem a következő sugarat.

Második lépés: „hits[1][0]”-ban megkapom az előző lépésben, a „rays[1][0]”-ba elhelyezett sugár eredményét. Ha a „hits[1][0]” azt mondja hogy ütközés van, akkor a „rays[2][0]”-ba bele helyezem a következő sugarat.

Harmadik lépés: „hits[2][0]”-ban megkapom az előző lépésben, a „rays[2][0]”-ba elhelyezett sugár eredményét. Most a „hits[2][0]”, a kép alapján azt mondja, hogy nincs ütközés. Itt befejezem a sugár indításokat. Kiszámolom a „hits” 2D-s tömbben lévő ütközési pontokból és normal vektorokból, illetve a fényforrásból érkező színeket, összeadom őket, így kiszámoltam a pixel színét.

Ha a „RayShader”-ben **false** értékkel térek vissza, az azt jelenti, hogy nincs vége a pixel színének számításnak.

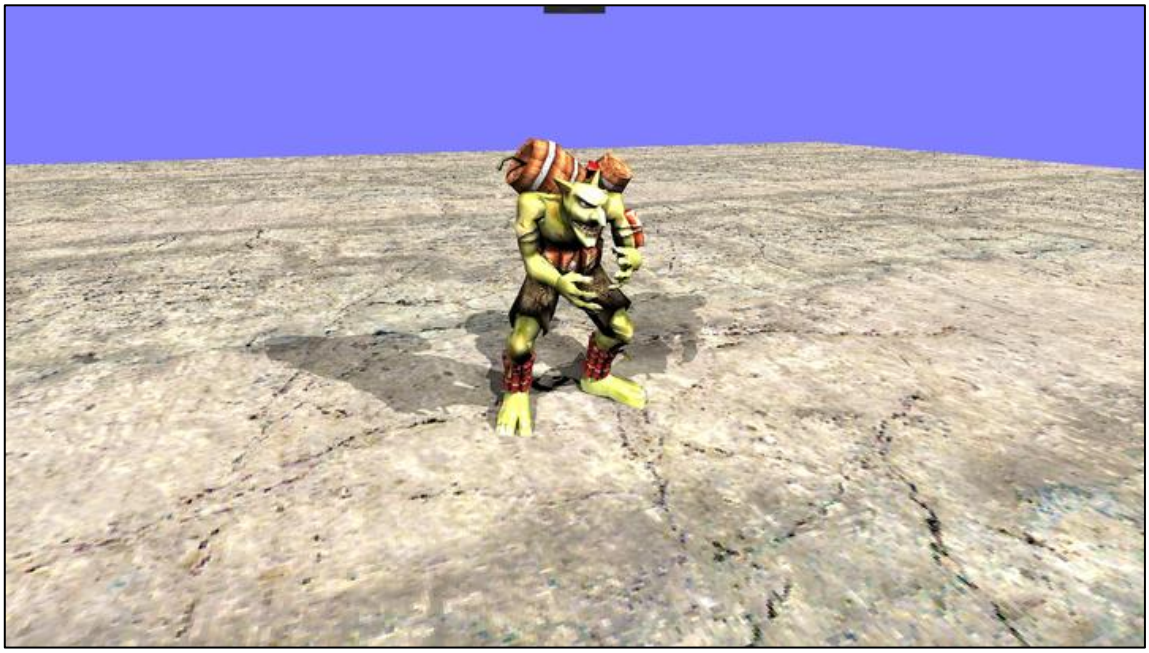
Így is fogalmazhatjuk: „új sugarat helyeztem a „Ray *rays” tömbbe, ha elmetsz háromszöget ez a sugár, akkor kérem a „Hit *hits” tömbben a metszéspontot. Majd hívódjon meg újra a RayShader() függvény”. Tehát a sugárkövetést folytatom.

Ha a „RayShader”-ben **true** értékkel térek vissza, az azt jelenti, hogy vége van a sugárkövetésnek.

Így is fogalmazhatjuk: „nem helyeztem új sugarat a „Ray *rays” tömbbe, kiszámoltam a „Hit *hits” adatokból az eredeti pixel színét, amit beírtam a paraméterként megkapott képernyő-textúrába”. A sugárkövetést befejezem ebben a pixelben.

7. Fények és árnyék számítása RayShader-ben:

Az alábbi képernyőkép (9. ábra) mutatja, hogy hogyan néz ki a RayShader eredménye, aminek a megírása a programozó feladata. Amit lejjebb bemutatok algoritmust, annak eredménye (9. ábra) ez a kép:



9. ábra: diffúz színek és árnyékok, RayShader-ben [saját]

Nulladik lépés: A képernyőképet törlöm. Jelen esetben ez a kék szín.

Első lépés: Megkapom RayShader-ben, a „hits[0][0]”-ban, hogy egy pixelben lévő sugár elmetsz-e egy háromszöget. Ha nem metsz el egy háromszöget sem, akkor befejezem a pixel színének számítását „return true” visszatérési értékkel. Ha van háromszög metszés, akkor az aktuális pixelt feketére színezem, mert ezt a pixelt még nem éri fény. A valóságban is így van ez. Illetve én három fényforrást hozok létre, amik messze vannak, a megvilágítandó objektumtól, így „irány fényforrás” hatást lehet elérni. Kiszámolom a három fényforrás pozícióból és az aktuális pixel metszéspontból, a három sugarat, amit bele helyezek a „rays[1][0]”, „rays[1][1]”, és „rays[1][2]”-be. Majd azt mondom a RayShader-nek, hogy „return false” (nincs vége a szín számításnak, kérem az 1.es rekeszbe tett sugarak metszéspontjait).

Második lépés: a „hits[1][0]”, „hits[1][1]” és „hits[1][2]” megmondja, hogy a három fényforrásból induló sugár metsz-e el háromszöget. Mi a fényforrás szemszögéből, a legközelebbi háromszög metszéspont. (Fontos: most egy pixel színét számolom) Ha nincs metszéspont, akkor „return true”-val befejezem a pixel színének számítását. Nem írok a képernyő textúrába semmit, tehát az a pixel, változatlan marad, ez a fényforrás, ezt a pixelt nem világítja meg (de lehet, hogy pl. a 2. vagy 3. fényforrás megvilágítja?).
Onnan lehet tudni hogy egy pixel metszéspontját éri-e fény, vagyis hogy nincs árnyékban, hogy a fényforrásból indított sugár, ugyan azt a metszéspontot adja eredményül, mint amit a képernyőből indított sugár talált metszéspontot.
Onnan tudom hogy a két pont különbözik-e egymástól, hogy a két pont távolsága túl nagy egymáshoz képest (a nagy távolság árnyékot jelent). Azokat a pixeleket, amiket fény ér, kiszámolom a diffúz színüket, mindhárom fényforrásra, összeadom őket, bele írom az új színt a képernyő textúra pixelébe. Majd (én) „return true”-t mondok, tehát befejezem az aktuális pixel színének számítását.
A fent leírt eljárás, a (9. ábra) képet adja eredményül.

„Hello World” példakód a VertexShader-re:

```
Vertex VertexShader(Vertex in, __global Matrix4x4 *in_Matrices)
{
  Vertex out;

  out = in;

  if (1 == in.numMatrices)
  {
    float3 v1 = Mult_Matrix4x4Float4(in_Matrices[in.matrixId1],
ToFloat4(in.vx, in.vy, in.vz, 1.0f);
    out.vx = v1.x;
    out.vy = v1.y;
    out.vz = v1.z;

    float3 n1 = Mult_Matrix4x4Float4(in_Matrices[in.matrixId1],
ToFloat4(in.nx, in.ny, in.nz, 0.0f);
    float3 n = normalize(n1);
    out.nx = n.x;
    out.ny = n.y;
    out.nz = n.z;
  }
  else if (2 == in.numMatrices)
  {
    ;
  }
  else if (3 == in.numMatrices)
  {
    ;
  }

  return out;
}
```

Ez a kód új pozícióba helyezi az „out.v” -t, és elforgatja az új irányba az „out.n” -t, egy mátrix segítségével.

„Hello World” példakód RayShader-re:

```
bool RayShader(Hits *hits, Rays *rays, __global Material *materials,
__global unsigned char *textureDatas, __global unsigned char *out, int
in_Width, int in_Height, int pixelx, int pixely)
{
  if (hits->id == 0)
  {
    Hit hit = hits->hit[hits->id][0];
    if (hit.isCollision == 0) { return true; }

    Color textureColor = Tex2DDiffuse(materials, textureDatas,
hit1.materialId, hit1.st);

    Color diffuseColor;
    diffuseColor.red   = float)textureColor.red;
    diffuseColor.green = float)textureColor.green;
    diffuseColor.blue  = float)textureColor.blue;
    diffuseColor.alpha = 255;

    WriteTexture(out, in_Width, in_Height, ToFloat2(pixelx, pixely),
diffuseColor);

    return true;
  }

  return true;
}
```

Ez a kód, ha „id == 0”, akkor fut le.

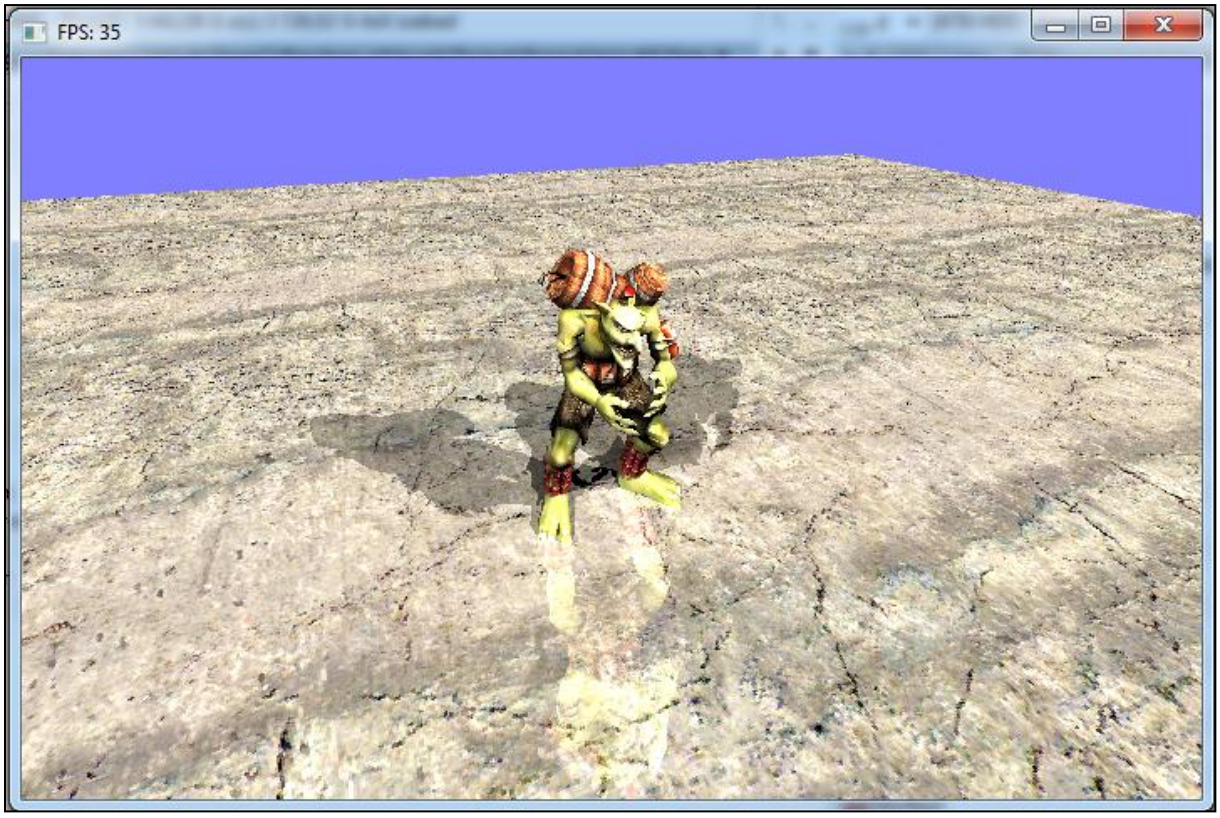
Ha nincs ütközés háromszöggel, akkor „return true”, vége a sugárkövetésnek, nem módosítunk az aktuális pixel színén.

Ha van ütközés háromszöggel, akkor lekérdezzük a a textúra színét, amit elmetsz a sugár, majd ezt a színt bele írjuk a textúrába, és „return true” -val vége a sugárkövetésnek.

Még egy példa, árnyékok és tükröződés számítására, RayShaderben:

Egy példa kódot szeretnék bemutatni, ami a RayShader használatát mutatja be. Létrehozok 3 fényforrást, amik diffúz színt számolnak (árnyékkal), illetve egy tükröződést mutatok be:

A kép:



10. ábra: árnyék és tükröződés példa [saját]

A forráskód:

```
bool RayShader(Hits *hits, Rays *rays, Vector3 camPos, Vector3 camAt,
__global Material *materials, __global unsigned char *textureDatas,
__global unsigned char *out, int in_Width, int in_Height, int pixelx, int
pixely)
{
    float3 light1; /*első fényforrás pozíciója*/
    light1.x = +1000.0f;
    light1.y = +1000.0f;
    light1.z = +1000.0f;

    float3 light2; /*második fényforrás pozíciója*/
```

```
    light2.x = -1000.0f;
    light2.y = +1000.0f;
    light2.z = +1000.0f;

    float3 light3; /*harmadik fényforrás pozíciója*/
    light3.x = 0.0f;
    light3.y = +1000.0f;
    light3.z = +1000.0f;

    float3 cam_pos;
    cam_pos.x = camPos.x;
    cam_pos.y = camPos.y;
    cam_pos.z = camPos.z;

    if (hits->id == 0)
    {
/*első lépésként ha nincs ütközés, akkor rayshader leállítása*/
        Hit hit = hits->hit[hits->id][0];
        if (hit.isCollision == 0) { return true; } /*nincs ütközés, vége*/

/*fényforrások pozíciójából, sugár indítása, egy pixel felé*/
        Ray newRay1; // light1
        newRay1.posx = light1.x;
        newRay1.posy = light1.y;
        newRay1.posz = light1.z;
        float3 dir1 = normalize(hit.pos - light1);
        newRay1.dirx = dir1.x;
        newRay1.diry = dir1.y;
        newRay1.dirz = dir1.z;
        newRay1.length = 5000.0f;

        Ray newRay2; // light 2
        newRay2.posx = light2.x;
        newRay2.posy = light2.y;
        newRay2.posz = light2.z;
        float3 dir2 = normalize(hit.pos - light2);
        newRay2.dirx = dir2.x;
        newRay2.diry = dir2.y;
        newRay2.dirz = dir2.z;
        newRay2.length = 5000.0f;
```

```
        Ray newRay3; // light3
        newRay3.posx = light3.x;
        newRay3.posy = light3.y;
        newRay3.posz = light3.z;
        float3 dir3 = normalize(hit.pos - light3);
        newRay3.dirx = dir3.x;
        newRay3.diry = dir3.y;
        newRay3.dirz = dir3.z;
        newRay3.length = 5000.0f;

/*tükröződés irányba sugár indítása. ez a negyedik sugár ebben az első
lépésben*/
        Ray newRay4; // reflection
        float3 pos = hit.pos + hit.normal * 0.01f;
        newRay4.posx = pos.x;
        newRay4.posy = pos.y;
        newRay4.posz = pos.z;
        float3 dir4 = reflect(normalize(hit.pos - cam_pos), hit.normal);
        newRay4.dirx = dir4.x;
        newRay4.diry = dir4.y;
        newRay4.dirz = dir4.z;
        newRay4.length = 5000.0f;

        rays->id = 1;
        rays->count[rays->id] = 4;
        rays->ray[rays->id][0] = newRay1;
        rays->ray[rays->id][1] = newRay2;
        rays->ray[rays->id][2] = newRay3;
        rays->ray[rays->id][3] = newRay4;

        return false;
    }

/*második lépés*/
    if (hits->id == 1)
    {
/*ha az első lépésben nem volt metszéspont, akkor kilépés*/
        Hit hit1 = hits->hit[0][0];
        if (hit1.isCollision == 0) { return true; }

/*alapértelmezésben a pixel diffúz színének intenzitása, 0*/
```

```
        float diffuseIntensity = 0.0f;

        Hit hit2 = hits->hit[hits->id][0];
        if (hit2.isCollision == 1)
        {
            float length2 = length(light1 - hit2.pos);
            float length1 = length(light1 - hit1.pos);

/*első fényforrás: ha nem vagyunk árnyékban, akkor az aktuális pixel diffúz
színének számítása*/
            if ((length2 + 0.005f) > length1)
            {
                float3 dir = normalize(hit1.pos - light1);
                diffuseIntensity += max(dot(-dir, hit2.normal), 0.0f);// +
max(dot(-dir2, hit.normal), 0.0f);
            }
        }
/*második fényforrás: ha nem vagyunk árnyékban, akkor az aktuális pixel
diffúz színének számítása*/
        Hit hit3 = hits->hit[hits->id][1];
        if (hit3.isCollision == 1)
        {
            {
                float length2 = length(light2 - hit3.pos);
                float length1 = length(light2 - hit1.pos);

                if ((length2 + 0.005f) > length1)
                {
                    float3 dir = normalize(hit1.pos - light2);
                    diffuseIntensity += max(dot(-dir, hit3.normal), 0.0f);
                }
            }
        }

/*harmadik fényforrás: ha nem vagyunk árnyékban, akkor az aktuális pixel
diffúz színének számítása*/
        Hit hit4 = hits->hit[hits->id][2];
        if (hit4.isCollision == 1)
        {
            {
                float length2 = length(light3 - hit4.pos);
```

```
                float length1 = length(light3 - hit1.pos);

                if ((length2 + 0.005f) > length1)
                {
                    float3 dir = normalize(hit1.pos - light3);
                    diffuseIntensity += max(dot(-dir, hit4.normal), 0.0f);
                }
            }
        }



        Color textureColor = Tex2DDiffuse(materials, textureDatas,
hit1.materialId, hit1.st);

/*textúra szín és intenzitás beírása az aktuális pixelbe */
        // diffuse
        Color diffuseColor;
        diffuseColor.red   = (int)(((float)textureColor.red  ) *
diffuseIntensity);
        diffuseColor.green = (int)(((float)textureColor.green) *
diffuseIntensity);
        diffuseColor.blue  = (int)(((float)textureColor.blue ) *
diffuseIntensity);
        diffuseColor.alpha = 255;

        // reflection
        Hit hit0 = hits->hit[0][0];
        Hit hit5 = hits->hit[hits->id][3];
        if (hit5.isCollision == 1 && hit0.objectId == 0)
        {
/*tükröződés: ha van metszéspont, ami a 0. objektumot metszi (vagyis a
talajt), akkor színszámítás*/
            Color reflectionColor = Tex2DDiffuse(materials, textureDatas,
hit5.materialId, hit5.st);

            float reflectionIntensity = diffuseIntensity * 0.25;
            diffuseColor.red   += (int)(((float)reflectionColor.red  ) *
reflectionIntensity);
            diffuseColor.green += (int)(((float)reflectionColor.green) *
reflectionIntensity);
```

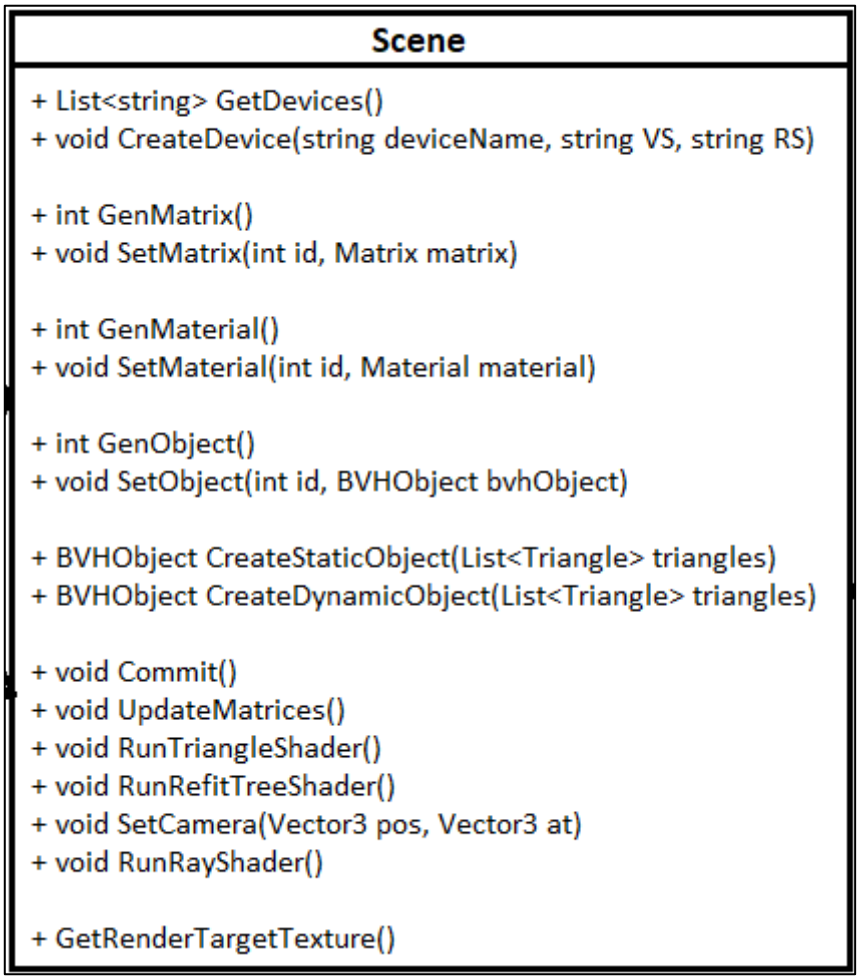
```
            diffuseColor.blue  += (int)(((float)reflectionColor.blue ) *
reflectionIntensity);
            diffuseColor.alpha = 255;
        }

        WriteTexture(out, in_Width, in_Height, ToFloat2(pixelx, pixely),
diffuseColor);

/*vége a sugár követésnek, ebben a pixelben*/
        return true;
    }

    return true;
}
```

8. A ’class Scene’ osztály feladata:



11. ábra: Scene osztály feladatai [saját]

A Scene osztály feladatai:

- A számítógépben lévő OpenCL eszközök kilistázása. Ehhez a "GetDevices()" függvényt lehet használni, ami visszatér egy listával, hogy milyen OpenCL eszközöket talált a rendszer.
- Egy OpenCL eszköz betöltése. Ki kell választani egy OpenCL eszközt a listából, és azon az eszköz fogja számolni a OpenCL függvény hívásokat.
- Matrix létrehozása, eltárolása, amit majd később el lehet érni. Egy OpenCL Bufferben létrejön egy mátrix, amit majd VertexShader-ben el lehet érni.
- Material (textúra) létrehozása, eltárolása, amit majd később el lehet érni OpenCL-el, a RayShader-ben.
- 'BVHObject' objektum (statikus vagy dinamikus) létrehozása háromszög listából, és eltárolása, amit majd később el lehet érni OpenCL-ben a RayShader-ben.
- OpenCL parancsok kiadása (ezeket már bemutattam), amivel a végső kép elkészül, amit a képernyőre lehet rajzolni. Ezek a parancsok:
    - void Commit()
    - void UpdateMatrices()
    - void RunTriangleShader()
    - void RunRefitTreeShader()
    - void SetCamera(Vector3 pos, Vector3 at)
    - void RunRayShader()
- A végső, elkészült kép lekérése.

Az 'OBJLoader' és 'SMDLoader' osztályok feladata, hogy szabványos .obj vagy .smd fájlból, háromszög listatát, mátrixokat hozzanak létre, amivel már 'BVHObject'-et lehet létre hozni. Most ezeknek a fájloknak a betöltését mutatom be.

8.1 Statikus objektum betöltése, 'class OBJLoader' osztály segítségével:

Az `.OBJ` fájl viszonylag elterjedt, minden 3D-s szerkesztőben megtalálható. Ez a fájl nem támogatja az animációt. Az `.OBJ` mellett általában szokott lenni egy `.MTL` fájl is, amely a textúrákat és az anyagok tulajdonságait írja le.

Az alábbi kulcsszavak szerepelnek egy `.OBJ` fájlban:

- Megjegyzés:

```
# ez egy sornyi megjegyzés a fájlban, nem kerül értelmezésre
```

- Egy csúcspont:

```
v 2.963193 1.167374 -0.431719
```

Ha `v` szöveggel kezdődik egy sor, akkor utána három lebegőpontos szám következik, amelyek egy pontot adnak meg a térben. Ezeket gyűjtsük össze egy `List<Vector3> vertices;` listába.

- Egy textúra koordináta:

```
vt 0.875000 0.035000
```

Ha `vt` szöveggel kezdődik egy sor, akkor utána két lebegőpontos szám következik, amelyek egy textúra-koordinátát adnak meg. Ezeket gyűjtsük össze egy `List<Vector2> textcoords;` listába.

- Egy normál vektor:

```
vn 0.704114 0.480790 0.522556
```

Ha `vn` szöveggel kezdődik egy sor, akkor utána három lebegőpontos szám következik, amelyek egy, a felületre merőleges vektort adnak meg. Ezeket is gyűjtsük össze egy `List<Vector3> normals;` listába.

- Háromszögek:

```
f 1/2/3 1821/1924/2 609/712/3
```

Ha `f` szöveggel kezdődik egy sor, akkor utána szóközökkel elválasztva minimum három vertex következik. Az első vertex `1/2/3` jelentése: Az aktuális csúcs az 1. elem a `vertices` listából, a csúcshoz tartozó textúra-koordináta a 2. elem a `textcoords` listából, és a csúcshoz tartozó normálvektor a 3. elem a `normals` listából. Ne felejtsük el, hogy a lista indexelése 0-tól kezdődik, tehát az `1/2/3` azt jelenti, hogy egy vertex így néz ki:

```
class Vertex
{
  public Vector3 v = vertices[0];
  public Vector2 vt = textcoords[1];
  public Vector3 n = normals[2];
}
```

Így van egy vertexünk. Egy háromszöghöz három vertex kell. Ha több vertex szerepel egy sorban, akkor a 4. vertex a 3. és az 1. vertexszel alkot egy háromszöget. Az 5. vertex a 4. és az 1. vertexel alkot egy háromszöget, és így tovább.

- Materiál kijelölése:

```
usemtl Texture_0
```

Ha `usemtl` szöveggel kezdődik egy sor, akkor az utána szereplő materiál-azonosítóval úgymond megmondom, hogy a most következő f-ekre (háromszögekre) ezt a textúrát kell ráhúzni.

- Textúrák és tulajdonságaik:

  `mtllib cottage.mtl`

Ha `mtllib` szöveggel kezdődik egy sor, akkor utána az `.MTL` materiálfájl neve szerepel, amely a textúrákat és azok tulajdonságait írja le. Az `.MTL` fájlban a legfontosabb kulcsszavak az alábbiak:

- `newmtl Texture_0`: Új materiált hoz létre, amelyre a `Texture_0` névvel hivatkozhatunk. Most ez az aktuális materiál.
- `mmap_Kd alma.bmp` vagy `map_Kd alma.bmp`: Az aktuális materiál textúrája az `alma.bmp`. Ez nem mindig szerepel.

`Kd 1.0 0.5 0.0`: Az aktuális materiál színe red: 1.0, green: 0.5, blue: 0.0. Ha nincs textúra, akkor a színt kell használni.

.OBJ fájl kirajzolása OpenGL-el:

```
foreach(Material material in materials)
{
  glBindTexture(GL_TEXTURE_2D, material.texture_id);
  glBegin(GL_TRIANGLES);
  foreach(Vertex vertex in material.vertices)
  {
    Vector3 v = vertex.v;
    Vector2 tc = vertex.tc;
    Vector3 n = vertex.n;
    glTexCoord2f(tc.X, tc.Y);
    glNormal3f(n.X, n.Y, n.Z);
    glVertex3f(v.X, v.Y, v.Z),
  }
  glEnd();
}
```

8.2 Dinamikus objektum betöltése, „class SMDLoader” osztály segítségével:

Az .SMD kiterjesztésű fájlt a Valve Corporation, 1996-ban alapított, videojátékokat fejlesztő amerikai vállalat hozta létre [9]. A Half-Life 1-2 Half-Life tudományos-fantasztikus belső nézetű lövöldözős (First Person Shooter, FPS) számítógépes játék is ezeket használja ahhoz, hogy animált modelleket jelenítsen meg.

Ez az .SMD kiterjesztésű fájl egy szöveges fájl. Két féle .SMD fájl létezik. Az egyikben mozdulatlan geometriát tárolunk (háromszögek), míg a másik típusban csak az animációt tároljuk. Mind a két féle fájlnak a kiterjesztése .SMD.

Miért kell külön tárolni a geometriát a hozzá tartozó animáció(k)tól? Azért, mert ha mi csak például a „futás” animációt szeretnénk betölteni, akkor elég csak ezt az egy animációt betölteni. Ha egy fájlban lenne a geometria és az összes animáció, akkor olyan animációt is betöltene a program, amelyre nincs szükség. Így tudunk válogatni, hogy mit töltsünk be, és mit ne.

Legelőször a geometriát kell betölteni – ezt kötelező. Utána töltjük be az animáció(ka)t. A geometriát tartalmazó fájlt szokás Reference.smd-nek nevezni, míg az animáció(kat)t tartalmazó fájl(oka)t pedig például Anim.smd-nek. Az .SMD kiterjesztésű fájl létrehozható 3D-s szerkesztőprogramokkal, például MilkShape3D-vel, 3DStudioMaxszal vagy Blenderrel (ehhez be kell kapcsolni a pluginját).

Az alábbi kódban példát látunk a `Reference.smd` fájl tartalmára:

```
version 1
nodes
0 "joint1" -1
1 "joint2" 0
2 "joint3" 1
end
skeleton
time 0
0 -0.750000 0.000000 0.500000 1.577046 0.000000 1.570796
1 0.000000 0.000000 40.000790 0.013222 0.000300 3.141593
2 0.000000 0.000000 59.751438 0.000000 -0.000000 0.000000
end
triangles
Kep1.bmp
0 19.250000 -20.500000 20.500000 0.000000 -1.000000 0.000000 0.000000
1.000000
0 19.250000 -20.500000 -20.500000 0.000000 -1.000000 0.000000 0.000000
0.000000
0 60.250000 -20.500000 20.500000 0.000000 -1.000000 0.000000 1.000000
1.000000
Kep1.bmp
0 19.250000 -20.500000 -20.500000 0.000000 -1.000000 0.000000 0.000000
0.000000
0 60.250000 -20.500000 -20.500000 0.000000 -1.000000 0.000000 1.000000
0.000000
0 60.250000 -20.500000 20.500000 0.000000 -1.000000 0.000000 1.000000
1.000000
end
```

Az alábbi kód példa az `Anim.smd` fájl tartalmára:

```
version 1
nodes
0 "joint1" -1
1 "joint2" 0
2 "joint3" 1
end
skeleton
time 0
0 -0.750000 0.000000 0.500000 1.577046 0.000000 1.570796
1 0.000000 0.000000 40.000790 0.013222 0.000300 3.141593
2 0.000000 0.000000 59.751438 0.000000 -0.000000 0.000000
time 1
0 -0.750000 0.000000 0.500000 0.791647 0.000000 1.570796
1 -0.000000 -0.000003 40.000549 0.798618 0.000212 -3.141380
2 0.000000 0.000000 59.751438 0.000000 -0.000000 0.000000
time 2
0 -0.750000 0.000000 0.500000 0.006249 0.000000 1.570796
1 -0.000000 -0.000002 39.999920 1.584001 0.000000 -3.141292
2 0.000000 0.000000 59.751438 0.000000 -0.000000 0.000000
end
```

. SMD fájlok kulcsszavai

Az alábbiakban az . SMD fájlok kulcsszavait mutatom be.

- nodes: Jelentése csomópontok. A csomópontok felsorolása end-ig tart.

```
nodes
0 "joint1" -1
1 "joint2" 0
2 "joint3" 1
end
```

Minden sor egy csomópont. Például 1 "joint2" 0 jelentése: ennek a csomópontnak az azonosítója: 1, neve: joint2, a szülejének azonosítója: 0. Célszerű tömbben (listában) tárolni ezeket az adatokat. Ha az i. azonosítójú elemre vagyunk kíváncsiak, akkor a nodes[i] visszaadja az i. azonosítójú csomópont adatait. Ha a szülő azonosítója −1, azt jelenti, hogy ennek a csomópontnak nincs szülője, tehát ez a csomópont a gyökér. Fel lehet rajzolni fagráfban ezeket a csomópontokat, hiszen szülő–gyerek kapcsolat van a csomópontok között.

```
class Node
{
public:
  string name;
  public int parent_id;
};
List < Node > nodes;
```

- bone: Jelentése csont. Egy modell, például az ember mozgatásához csontvázrendszer van létrehozva. Ha mozgatjuk például a törzsünket, akkor vele mozog a hozzá kapcsolt fejünk, karunk. Tehát a csontok között hierarchia van, szülő–gyerek kapcsolat. Sok csont határoz meg egy csontvázat, angolul skeletont. Egy csont lokális transzformációval forogni tud az X, Y, Z tengelyek mentén, és eltolható a szülejéhez képest. Az alábbi kódrészletek egy csontra és egy csontvázra adnak példát.

```
// egy csont
class Bone
{
  // lokális transzformációk. Eltérés a szülőhöz képest
  public Vector3 translate;
  public Vector3 rotate;
};
// egy csontváz csontok listájából áll
class Skeleton
{
public:
  public List < Bone > bones;
};
```

Illetve egy animáció sok csontvázból áll:

```
// egy animáció
class Animation
{
  public string name; // az aktuális animáció neve (Anim.smd)
  public float fps; // 1 sec alatt, hány skeleton játszódik le
  public List < Skeleton > times;
};
// sok animáció egy listában
List < Animation > animations;
```

- `time`: Jelentése idő.

```
time 0
0 -0.750000 0.000000 0.500000 1.577046 0.000000 1.570796
1 0.000000 0.000000 40.000790 0.013222 0.000300 3.141593
2 0.000000 0.000000 59.751438 0.000000 -0.000000 0.000000
```

Az animációban vannak a csomópontok (csontok) lokális transzformációi. Például a `0 1.0 2.0 3.0 4.0 5.0 6.0` jelentése: a 0. csomópont lokális mátrixa ez:

```
Matrix4 local = Matrix4.Translate(1.0, 2.0, 3.0) *
Matrix4.RotateXYZ(4.0, 5.0, 6.0);
```

Vagyis forgatás a tengelyek mentén 4.0, 5.0, 6.0 radiánnal, majd eltolás 1.0, 2.0, 3.0 egységgel. Ha egy modellben például 25 csomópont van, akkor egy time-ban 25 db transzformáció van felsorolva egymás után. Egy time egy pillanatot ábrázol. Ahhoz hogy ebből animáció legyen, ahhoz sok time kell egymás után. 1 másodperc alatt kb. 4-30 time jeleik meg a képernyőn. Két time között kb. 0,2 másodperc telik el. A két time közötti pillanatnyi eltolást lineáris interpolációval, a forgást előjeles szögelfordulással lehet kiszámolni:

```
times[0].bones[0].translate = new Vector3(1,0,0);
times[0].bones[0].rotate = new Vector(1,0,0);
times[1].bones[0].translate = new Vector(2,0,0);
times[1].bones[0].rotate = new Vector(2,0,0);
```

Ha például `t = 0.2` (t = [0.0 .. 1.0]), akkor a pillanatnyi transzformáció ez:

```
float t = 0.2;
current_skeleton.bones [0].translate = (times[0].bones[0].translate * (1 -
t)) + (times[1].bones[0].translate * t); // lineáris interpoláció
// rotate X
float rotate_x = GetSignedRad(times[0].bones[0].rotate.X,
times[1].bones[0].rotate.X);
current_skeleton.bones[0].rotate.X = start.bones[0].rotate.X + (rotate_x *
dt);
// rotate Y …
// rotate Z …
```

Forgásnál fontos, hogy merre forgatunk. Például ha a kezdő szög 0,1 radián, a vég szög pedig 6,27 radián, akkor -0,2 radiánnal kell forogni az óra járásával megegyező irányba. Itt nem lehet lineáris interpolációval számolni, mert 6,26 radiánnal forogna az óra járásával ellentétes irányba.

- **skeleton:** Jelentése csontváz. `time`-ok vannak benne `end` végjelig:

```
skeleton
time 0
0 -0.750000 0.000000 0.500000 1.577046 0.000000 1.570796
1 0.000000 0.000000 40.000790 0.013222 0.000300 3.141593
2 0.000000 0.000000 59.751438 0.000000 -0.000000 0.000000
time 1
0 -0.750000 0.000000 0.500000 0.791647 0.000000 1.570796
1 -0.000000 -0.000003 40.000549 0.798618 0.000212 -3.141380
2 0.000000 0.000000 59.751438 0.000000 -0.000000 0.000000
time 2
0 -0.750000 0.000000 0.500000 0.006249 0.000000 1.570796
1 -0.000000 -0.000002 39.999920 1.584001 0.000000 -3.141292
2 0.000000 0.000000 59.751438 0.000000 -0.000000 0.000000
end
```

Például egy `time 0` leírja a 0. időben a csomópontok transzformációit, vagyis az aktuális csontváz csontjainak helyét és forgási szögeit. A `Reference.smd` fájlban csak egy `time` szerepel, ez a kezdeti beállása a modellnek. Míg az `Anim.smd` fájlban sok `time` szerepel, mindegyik leírja hogy az i. `time`-ban mi az aktuális csontváz, vagyis az aktuális transzformációk.

Egy csomópontnak úgy tudjuk kiszámolni a globális koordináta-rendszerben a transzformációját a lokális transzformációból, hogy vesszük a lokális transzformációt, majd rekurzívan összeszorozzuk a szülejének a lokális transzformációjával, azután a szülő szülejének a lokális transzformációjával addig, amíg el nem jutunk a gyökérig. A gyökér csomópont transzformációja az egység mátrix.

Itt egy példakód erre:

```
Matrix4 GetMatrix(int bone_id)
{
  if (bone_id == -1) return Matrix4.Identity;

  // lokális transzformáció. Eltérés a szülőhöz képest
  Vector3 r = currBones[bone_id].rotate;
  Vector3 t = currBones[bone_id].translate;
  Matrix4 local = Matrix4.Translate(t.X, t.Y, t.Z) * Matrix4.RotateXYZ(r.X,
r.Y, r.Z);

  // szülő transzformáció kiszámítása, rekurzívan
  Matrix4 parent = GetMatrix(nodes[bone_id].parent_id);

  // visszatérünk az aktuális transzformációval
  return Matrix4.Mult(parent, local);
}
```

- `triangles`**:** A triangle jelentése háromszög poligon. Háromszögek felsorolva `end` végjelig. Egy háromszög így néz ki:

```
texture.bmp
1 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
2 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
3 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
```

Ez azt jelenti, hogy erre a háromszögre a `texture.bmp`-t kell illeszteni. Utána szerepel három sor. Minden sor egy csúcsot ír le. Az első sor ez: `1 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0`. Ez azt jelenti, hogy a háromszög egyik csúcsa [1.0, 2.0, 3.0], normálvektora [4.0, 5.0, 6.0], textúra-koordinátája [7.0, 8.0]. Az első szám az 1-es azt jelenti, hogy ehhez a csúcshoz az első indexű csomópont tartozik. Így már hozzá van rendelve egy csúcs egy transzformációhoz.

Amikor animációt akarunk megjeleníteni, elég csak kiszámítani a pillanatnyi transzformációkat, majd a megfelelő csúcsokra alkalmazni.

Animáció kiszámítása és megjelenítése, OpenGL segítségével:

Végigjárjuk a `materials` listát, kijelöljük az aktuális textúrát, majd a material minden egyes csúcspontját transzformáljuk, majd kirajzoljuk OpenGL segítségével:

```
void Draw()
{
  glPushMatrix();
  for(int i = 0; i < materials.Count; i++)
  {
    Material mat = materials[i];
    glBindTexture(gl.GL_TEXTURE_2D, mat.texture_id);
    glBegin(GL_TRIANGLES);

    for(int j = 0; j < mat.vertices.Count; j++)
    {
      Vertex in = mat.vertices[j];

      Matrix4 T = transform[in.id] * transformInverse[in.id]; //
transzformáció kiszámítása
      Vector3 v = T * Vector4(in.position, 1);
      Vector3 n = T * Vector4(in.normal  , 0); // itt nincs eltolás, csak
forgás van
      Vector3 t = in.texCoord;

      glTexCoord2f(t.X, t.Y);
      glNormal3f(n.X, n.Y, n.zZ);
      glVertex3f(v.X, v.Y, v.Z);
    }
    glEnd();
  }
  glPopMatrix();
}
```

Mit is jelent a kirajzolásnál az alábbi sor?

```
Matrix4 T = transform[in.id] * transformInverse[in.id]; // transzformáció kiszámítása
```

Ez azt jelenti, hogy két listánk van. Egy `List < Matrix4 > transform;` és egy `List < Matrix4 > transformInverse;`.

A `transform` lista a pillanatnyi csontváz transzformációit tartalmazza. Ezt minden képkockán (frame-en) ki kell számítani:

```
for(int i = 0; i < currBones.size(); i++)
{
  transform[i] = GetMatrix(i);
}
```

A `transformInverse` lista pedig a `Reference.smd` fájlban található egyetlen csontváz transzformációinak inverzét tartalmazza. Ez mit is jelent? A `Reference.smd` fájlban található egyetlen csontváz adott, és a csúcspontok is adottak. Nevezzük el a `Reference.smd` fájlban található csontváz i. transzformációját `T`-nek, és a `Reference.smd` fájlban található egyik háromszög egyik csúcsát `out`-nak. Ha elképzeljük, akkor: `Vector3 out = T * ?;` Tehát adott az `out` vektor és a `T` mátrix a `Reference.smd` fájlban. A `?` az ismeretlen, amire szükségünk lesz, ezért ki kell számítani.

Amikor a pillanatnyi animációt akarom megjeleníteni, akkor azt így kell: `Vector3 v = transform[i] * ?;`

Tehát fejezzük ki a ?-et: `Vector3 out = T * ?; // átalakítás (fejezzük ki a ?-et)`

1. egyenlet: `Vector3 ? = Matrix4.Inverse(T) * out;`

Ha megvan a `?`, akkor:

2. egyenlet: `Vector3 v = transform[i] * ?;`

Egybe a két egyenlet: `Vector3 v = transform[i] * inverse(T) * out;` Így `Vector3 v` az aktuális animáció egyik csúcspontja.

Ezért szerepel a `Draw()` metódusban a fenti megoldás.

A `transformInverse` listát elég egyszer kiszámítani, például a `Reference.smd` fájl betöltése után:

```
Matrix4 GetRefMatrix(int bone_id)
{
  if (bone_id == -1) return Matrix4.Identity; // egységmátrix, ha a
gyökérben vagyunk

  // lokális transzformáció. Eltérés a szülőhöz képest
  Vector3 r = referenceSkeleton.bones[boneId].rotate;
  Vector3 t = referenceSkeleton.bones[boneId].translate;
  Matrix4 local = Matrix4.Translate(t.X, t.Y, t.Z) * Matrix4.RotateXYZ(r.X,
r.Y, r.Z);

  // szülő transzformáció kiszámítása, rekurzívan
  Matrix4 parent = GetRefMatrix(nodes[bone_id].parent_id);

  // visszatérünk az aktuális transzformációval
  return Matrix4.Mult(parent, local);
}
for(int i = 0; i < referenceSkeleton.bones.Count; i++)
{
  transformInverse[i] = Matrix4.Inverse(GetRefMatrix(i));
}
```

A `SetTime(float time)` metódusban pedig a pillanatnyi csontvázat kell kiszámolni, amelyet majd később kirajzolunk.

```
// dt = [0.0 .. 1.0]
void CalcNewSkeleton(Skeleton start, Skeleton end, float dt)
{
  for(int i = 0; i < current_skeleton.bones.Count; i++)
  {
    // translate
    current_skeleton.bones[i].translate = (start.bones[i].translate * (1.0f
- dt)) + (end.bones[i].translate * dt);
    // rotate
    float rotate_x = GetSignedRad(start.bones[i].rotate.X,
end.bones[i].rotate.X);
    current_skeleton.bones[i].rotate.X = start.bones[i].rotate.X +
(rotate_x * dt);
    // rotate Y …
    // rotate Z …
  }
}
```

```
// Az "int anim_id"-edik animáció, "float time" másodpercének (pl. 6.5 mp),
a pillanatnyi transzformációjának kiszámítása
void SetTime(int anim_id, float time)
{
  // aktuális animáció lekérdezése
  Animation *anim = animations[anim_id];

  // Get Skeleton
  int start = (int)Math.Floor(time * anim.fps);
  int end = (int)Math.Ceil(time * anim.fps);

  if (start == end)
  {
    CalcNewSkeleton(times[start], times[end], 0.0f);
  }
  else
  {
    float skeletonTime1 = (float)start / anim->fps;
    float skeletonTime2 = (float)end / anim->fps;
    float timeDiff1 = skeletonTime2 - skeletonTime1;
    float timeDiff2 = time - skeletonTime1;
    float dt = timeDiff2 / timeDiff1;
    CalcNewSkeleton(times[start], times[end], dt); // dt = [0.0 .. 1.0]
  }

  // Update Matrices (transforms)
  for(int i = 0; i < currBones.size(); i++)
  {
    transform[i] = GetMatrix(i);
  }
}
```

- Érdekesség: A régi, Half-Life 1 `.SMD` fájlban, a háromszög egy csúcsához pontosan egy transzformáció (csont) tartozott. Az új, Half-Life 2 `.SMD` fájlban a háromszög egy csúcsához 1-3 transzformáció (csont) tartozhat. A csontok súlyozva vannak. A súlyok összege 1,0. Ez az újítás azért van, mert például egy ember könyökénél egy csúcs félig a felkarhoz, félig az alkarhoz tartozik. A súly mondja meg, hogy melyikhez tartozik jobban.

  A Half-Life 2-es `.SMD` fájlban így szerepel egy háromszög:

```
Kep1.bmp
0 19.250000 -20.500000 20.500000 0.000000 -1.000000 0.000000 0.000000
1.000000 1 0 1.0
0 19.250000 -20.500000 -20.500000 0.000000 -1.000000 0.000000 0.000000
0.000000 2 0 0.6 1 0.4
0 60.250000 -20.500000 20.500000 0.000000 -1.000000 0.000000 1.000000
1.000000 3 0 0.4 1 0.3 2 0.3
```

Itt az **1 0 1.0** azt jelenti, hogy egy súlyozott transzformáció tartozik ehhez a csúcshoz. A transzformáció azonosítója a 0, súlya 1.0.

```
Matrix4 T = Matrix4.Mult(GetMatrix(0), 1.0);
```

A **2 0 0.6 1 0.4** azt jelenti, hogy két súlyozott transzformáció tartozik ehhez a csúcshoz. Az első transzformáció azonosítója 0, súlya 0.6. A második transzformáció azonosítója 1, súlya 0.4.

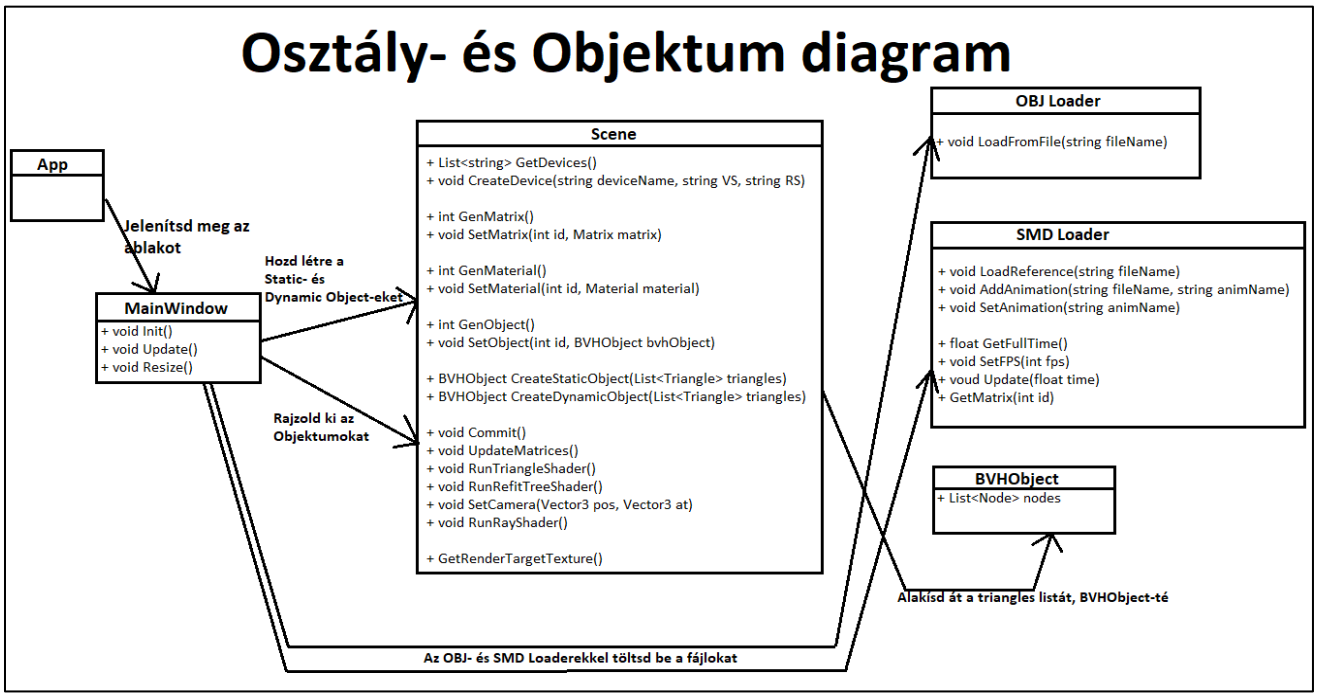
```
Matrix4 T = Matrix4.Mult(GetMatrix(0), 0.6) + Matrix4.Mult(GetMatrix(1),
0.4);
```

A **3 0 0.4 1 0.3 2 0.3** azt jelenti, hogy három súlyozott transzformáció tartozik ehhez a csúcshoz. Az első transzformáció azonosítója 0, súlya 0.4. A második transzformáció azonosítója 1, súlya 0.3. A harmadik transzformáció azonosítója 2, súlya 0.3.

```
Matrix4 T = Matrix4.Mult(GetMatrix(0), 0.4) + Matrix4.Mult(GetMatrix(1),
0.3) + Matrix4.Mult(GetMatrix(2), 0.3);
```

A súlyok összege mindenhol 1.0.

9. Osztály-, Objektum-, és Használati eset diagram:



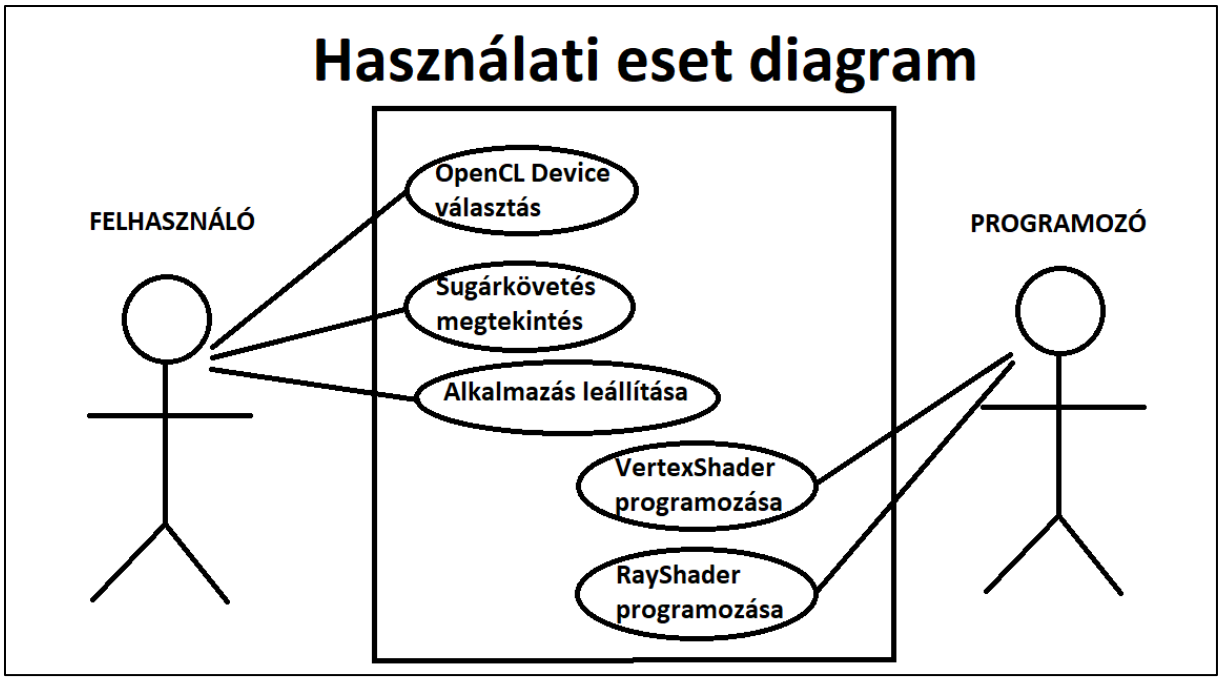
12. ábra: Oszály diagram és Objektum diagram[saját]

Ahogy az objektum diagram mutatja, a „MainWindow” „void Init()” függvénye:

- Létre hozza a „Scene” objektum segítségével az „OpenCL Device” -t.
- Betölti az „OBJLoader” és „SMDLoader” segítségével a „*.obj” és „*.smd” fájlokat.
- A betöltött objektumokat átadja a „Scene” -nek, ami betölti az objektumokat.

A „MainWindow” „void Update()” függvénye:

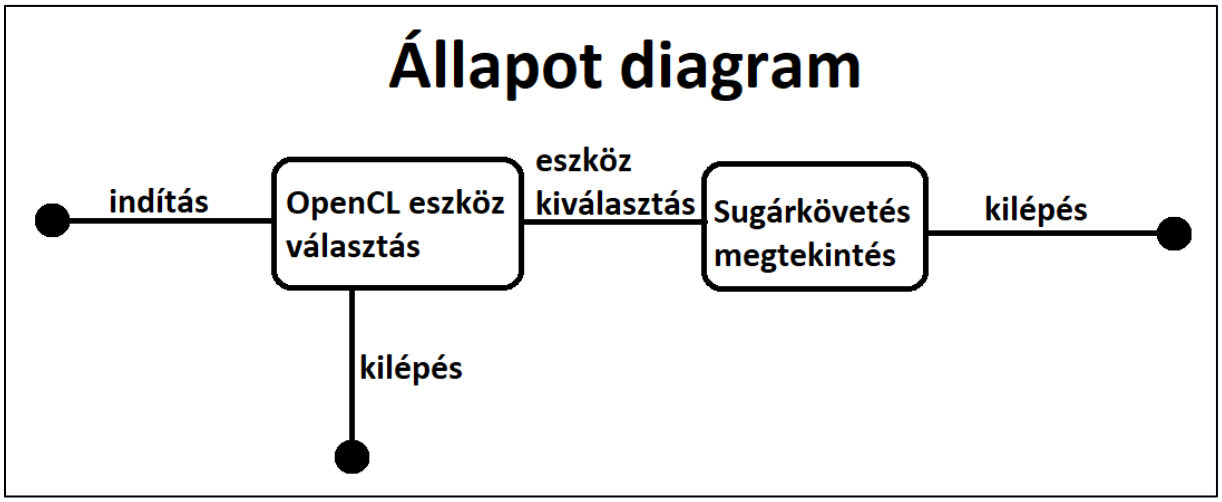
- Frissíti a mátrixokat a „Scene” objektum segítségével
- Meghívja a TriangleShader-t, RefitTreeShader-t.
- A „void SetCamera(…)” -val beállítja a néző pontot, ami a kezdő sugarakat létre hozza.
- Majd meghívja a RayShader-t, ami kiszámolja a végső színt, amit a sugarak elmetszenek.



13. ábra: Használati eset diagram[saját]

Két ember használhatja a programot:

- **Programozó:** Ő írja meg a TriangleShader-t és a RayShader-t.
- **Felhasználó:** Ő választ „OpenCL device” -t, megtekinti a képet, majd bezárja az alkalmazást.



14. ábra: Állapot diagram[saját]

Az alkalmazás állapotai:

- Az alkalmazás indítása után bezárhatjuk, vagy „OpenCL device”-t választahunk.
- Ha device-t választottunk, akkor megtekinthetjük a képet.
- Majd bezárhatjuk az alkalmazást.

10. Továbbfejlesztési lehetőségek:

Több továbbfejlesztési lehetőség is van:

Első: A képernyőből induló első metszéspont kiszámítását, nem csak sugárkövetéssel lehet megkapni (ami gépigényes), hanem „OpenGL2.0” vagy „Direct3D9”-el is kiszámíthatók a metszéspontok. Az OpenCL nem rendelkezik „raszterizáló parancsal”, míg az opengl vagy direct3d rendelkezik ilyennel. A raszterizálás azt jelenti, hogy ha van egy háromszögem, aminek ismerem a három csúcs adatait, akkor a videokártya képes kiszámolni a háromszögen belüli pixelek adatait a három csúcsot figyelembe véve. Erre egy külön áramkör áll rendelkezésre a videokártyában. Úgy tudom, ez az áramkör OpenCL-ben, nem elérhető. Ha opengl-t vagy direct3d-t használnék a pixelekből indított sugarak helyett, az akár 100x gyorsabban eredményül adná az első lépésben, a kamerához legközelebbi metszéspontokat. Igaz ilyenkor shader-ben (GLSL, HLSL), kell ügyeskedni.

Második: Ez az OpenCL-es megoldás, amit bemutattam, elavult. Manapság olyan videokártyákat lehet kapni (pl. Nvidia RTX), amik a háromszög-félegyenes metszéspontszámítást elektronikával (chip-el) oldja meg, azaz 1 utasítást kell csak kiadni.
Az én megoldásom, a háromszög-sugár metszéspont számításához, kb. 20-30 utasítást használ (sík-pont távolság, sík-sugár távolság, háromszögen belül van-e a metszéspont), ez lassú.

Harmadik: Ha már sugárkövetés, kihasználhatnám az előnyeit. Indíthatnék sugarakat tükröződési- törési irányba, a diffúz szín kiszámításához. Most csak a fényforrásokból indítottam sugarat egy pixel színének kiszámításához.

Negyedik: Az alkalmazás, amikor háromszög-sugár metszéspontot keres, igaz hogy egy objektum egy BVH fa, abban gyors a keresés. De ha sok objektumom van, pl. 100 darab, jelen esetben a program egy „for” ciklussal bejárja mind a pl. 100 objektumot, legközelebbi metszéspontot keresve. Ez nem optimális. Az objektumokat is rendezni kéne a térben. Igaz, ezek az objektumok elmozdulhatnak, ilyenkor „frissíteni” kéne az elrendezést.

Ötödik: A Multi GPU. Vagyis, ha több videokártya van egy számítógépben, akkor ha a képernyőn megjelenő pixelekből induló sugarak egyik felét az egyik vga számolja, míg a képernyőn megjelenő pixelekből indított sugarak másik felét egy másik gpu számolja, akkor majdnem 2x-es sebességel gyorsabb lehet az alkalmazás. A TriangleShader-t egy vga-val kell elvégezni, de a pixelek színének számításához lehetne párhuzamosítani több vga-val. Én ezt nem tettem meg, de elméletileg meg lehet tenni.

Hatodik:

Azt, hogy a sugár a háromszögön belül van-e, azt nem csak vektoriális szorzattal, hanem barycentrikus koordinátákkal is ki lehet számolni.

A háromszög súlya, 1.0. Ha a metszésponttal számolt súlyok összege nagyobb mint 1.0, akkor a háromszögen kívül van a metszéspont.

10.1 BVH fa építése, gyorsan:

Ez megoldás lehet arra, hogy ne "for" ciklussal járjuk be az objektumokat. A gyors BVH fát lehet használni úgy, hogy a fa leveleiben az objektumok vannak, de még gyorsabb megoldás az, ha a háromszögeket tenném a BVH fába. Illetve csak akkor építeném újra a BVH fát, ha animáció van (a háromszögek torzulnak).

Egy megoldást szeretnék mutatni "gyors BVH fa építésére". Ez saját elgondolás, biztosan létezik gyorsabb megoldás erre, ennek ellenére be szeretném mutatni a saját megoldásomat. Most pontokkal fogok dolgozni, ezek a pontok lehetnek az objektumok középpontjai, vagy még jobb, ha a háromszögek lista, egy háromszögének, egyik csúcsa, pl. az "triangle.A" csúcs.

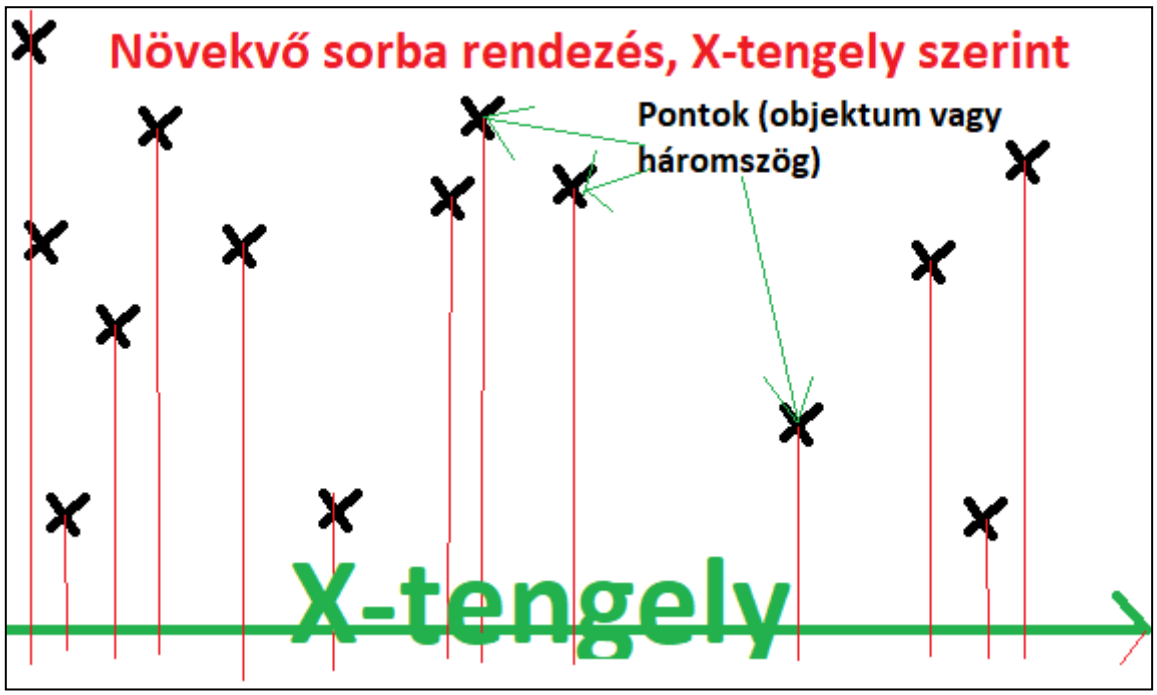
3 lépésből áll a megoldásom:

1. növekvő sorrendbe rendezni valamelyik tengely(X-tengely) szerint a csúcsokat.

2. egy csúcshoz, egy környezeten belül, a legközelebbi szomszédos csúcs megkerése. Így megvannak a levelek, a BVH fa 0. szintje.

3. Alkalmazni a 2. lépést, a további 2., 3. ... N. szintekre addíg, amíg vannak szomszédok. Ha már nincs szomszédja egy befoglaló négyzetnek, az a "root", vagyis a fa gyökere. Így elkészült egy BVH fa.

Most bemutatom a három lépést részletesen:



15. ábra: Valamely koordináta-tengely szerindt, rendezni a pontokat (legyen ez pl. az X tengely) [saját]

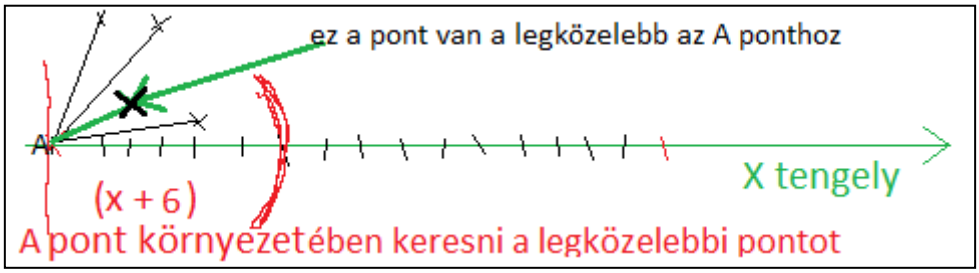
Ahogy a 14.-es ábra mutatja, rendezzük pl. x tengely szerint növekvő sorrendbe a csúcsokat. Így kapunk egy olyan rendezett csúcs listát, amiben majd ha később keresni szeretnénk, akkor használhatom a "bináris keresést".

Bináris keresés: ha van egy növekvő számsorozat, és meg szeretném tudni hogy szerepel-e benne egy X szám, akkor megtehetem, hogy először a számsorozat közepén lévő Y számmal összehasonlítást végzek. Ha az Y szám nagyobb, mint az X szám, az azt jelenti, hogy a számsorozatnak a második felén, az Y számnál nagyobb számok vannak, ezért azon a részen már nem is kell keresnem, elég csak a számsorozat első felében folytatni a keresést. Egy Y szám összehasonlítással, a sorozat felét el lehet dobni. Majd a sorozaton ismételni ezt az eljárást, addíg felezgetjük a sorozatot, mígy egyszer eljutunk egy számhoz. Ez a szám lesz a legközelebb a keresett X számhoz.

Fontos! Bináris keresést csak rendezett szám sorozaton lehet alkalmazni.

Legelőször a sorba rendezést kell elvégezni. Ezt lehet egy szálon vagy több szálon is végezni. A C# "Array.Sort()" metódusa egy szálon rendezi növekvőbe az elemeket. Ha sok csúcs van, akkor érdemes a sorbarendezést több szálon, több CPU-val végezni.

Már most lehetne BVH fát építeni úgy, hogy a számsorozatban lévő szomszédos számok, egy levelet alkossanak, hiszen x tengely szerint, ők egymáshoz a legközelebb vannak.
Viszont Y és Z tengely is létezik. Mi van akkor, ha két szomszédos pont x tengely szerint pl. 0 méterre vannak egymástól, de ha a pl. az Y vagy Z tengelyt vizsgáljuk, akkor pl. 1km távolságra vannak egymástól a koordináták. Akkor ez az X tengely szerinti szomszédos pont mégsem a legközelebbi? Nem, nem a legközelebbi, mert 1km a távolsága a két pontnak, ha az Y és Z dimenziókat is figyelembe vesszük.
Az én megoldásom az, hogy vegyük a pont jobb oldali környezetét, vizsgáljunk meg nem csak a szomszédos pont távolságát, hanem a szomszéd-szomszédját, illetve annak a szomszédját. Ha sok pontot vizsgálok, nagyobb valószínűséggel találom meg a ponthoz, valószínűleg a legközelebbi pontját.

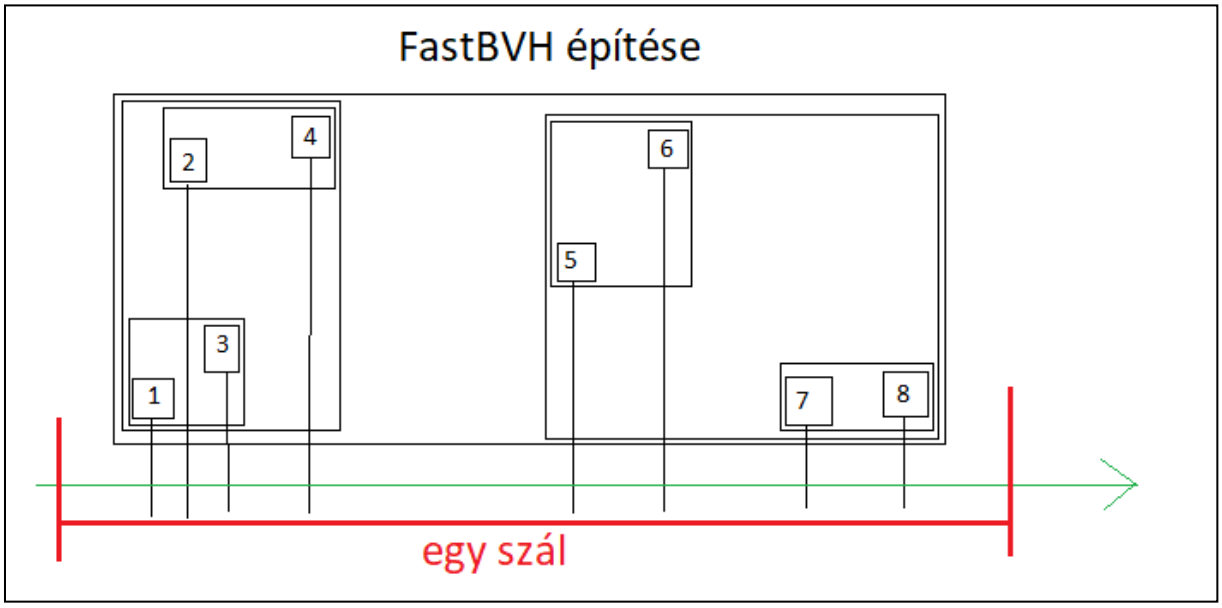


16. ábra: egy pont környezetét vizsgálva, a szomszédja pont kiválasztása [saját]

Ahogyan a 15. ábra mutatja, vizsgáljuk meg az "A" ponttól, jobbra lévő 6 pontot, és keressük meg a legközelebbi pontot. Így közelítőleg jó szomszédot tudunk találni az "A" ponthoz, és nem kellett az egész számsorozatot bejárni, hogy melyik a biztosan legközelebbi pont. Minél nagyobb környezetet vizsgálunk egy pontnál, annál nagyobb a valószínűség, hogy a legközelebbi pontot találtuk meg. De nekünk most a cél a gyors BVH fa építése, ezért ahogy a kép is mutatja, csak 6 pontot vizsgálok. Ezt ki kell tapasztalni programozáskor, hogy mi a jó határ(6 vagy 20) ahhoz, hogy gyors legyen a fa építése, és annak a bejárása. Ha nagyon nem jó szomszédot találunk egy csúcsnak, nagy lesz a távolság két szomszédos pont között, akkor amikor sugarat indítok, akkor több csomópontra fogja azt mondani a bvh fa, hogy ott lehet

háromszög, hiszen nagyobb területű a befoglaló doboz, nagyobb eséllyel mont "igaz" választ egy csomópont arra, hogy van-e sugár-boundingbox metszéspont. Így többet fog számolni az OpenCL, tehát lassabb.

Amikor megtaláltam a számsorozat 0. eleméhez, a szomszédos N pontját, akkor a 0. és N. pontot töröljük a listából, és ez a két ponttal hozzunk létre egy csomópontot. Mivel töröltük a 0. pontot, és az N. pontot a listából, ugyan ezt az eljárást meg lehet ismételni. Megint a 0. indexű ponthoz keressük meg a környezetében lévő szomszédos pontot, és ők is alkossanak egy csomópontot. Ezek a csomópontok kerüljenek bele egy kimeneti listába. Addig ismételjük az eljárást, amíg a bemeneti lista ki nem ürül. Ha egy darab elem marad a bemeneti listában, akkor abból egy olyan csomópontot kell létre hozni, aminek egy levele van.

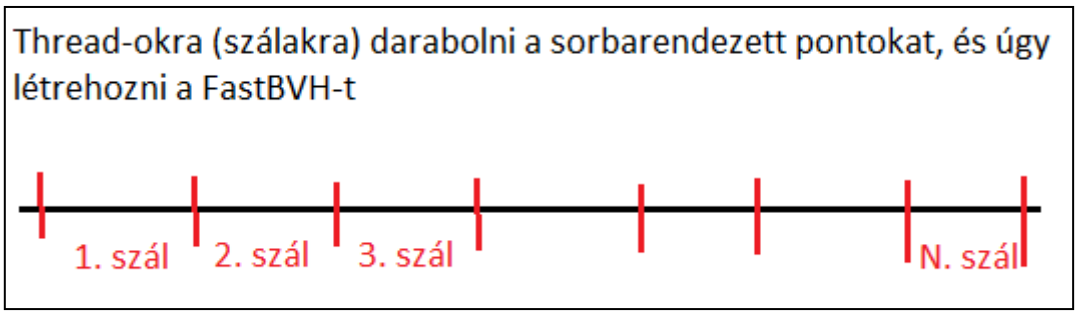


17. ábra: befoglaló négyzetek frissítése, egészen a "root" -ig [saját]

Ahogyan a 16. ábra mutatja, az 1. es csúcs szomszédja a 3. csúcs, pedig X tengely szerint a 2. csúcs közelebb lenne, tehát műlködőképes lehet ez a megoldás.

Majd a kapott kimeneti lista elemeinek számítsuk ki a bounding box-ját, és ismételjük meg a fennti szomszédos keresés eljárást újra, a most már bounding box-okat tartalmazó listában. Mivel a bemeneti listából mindig két elemből, egy elemet készítünk, és azt tesszük a kimeneti listába, így a kimeneti lista minden lépéssel fele akkora lesz, mint a bemeneti lista. Egyszer eljutunk ahhoz az állapothoz, hogy 1 darab bounding box-ot kapunk eredményül. Ő lesz a gyökér "root" elem. A befoglaló négyzetek kiszámítása, amiből nagyon sok van, pont annyi, ahány csomópontja van egy bvh fának, gépigényes. Ha meg van egy szint. pl. a levelek

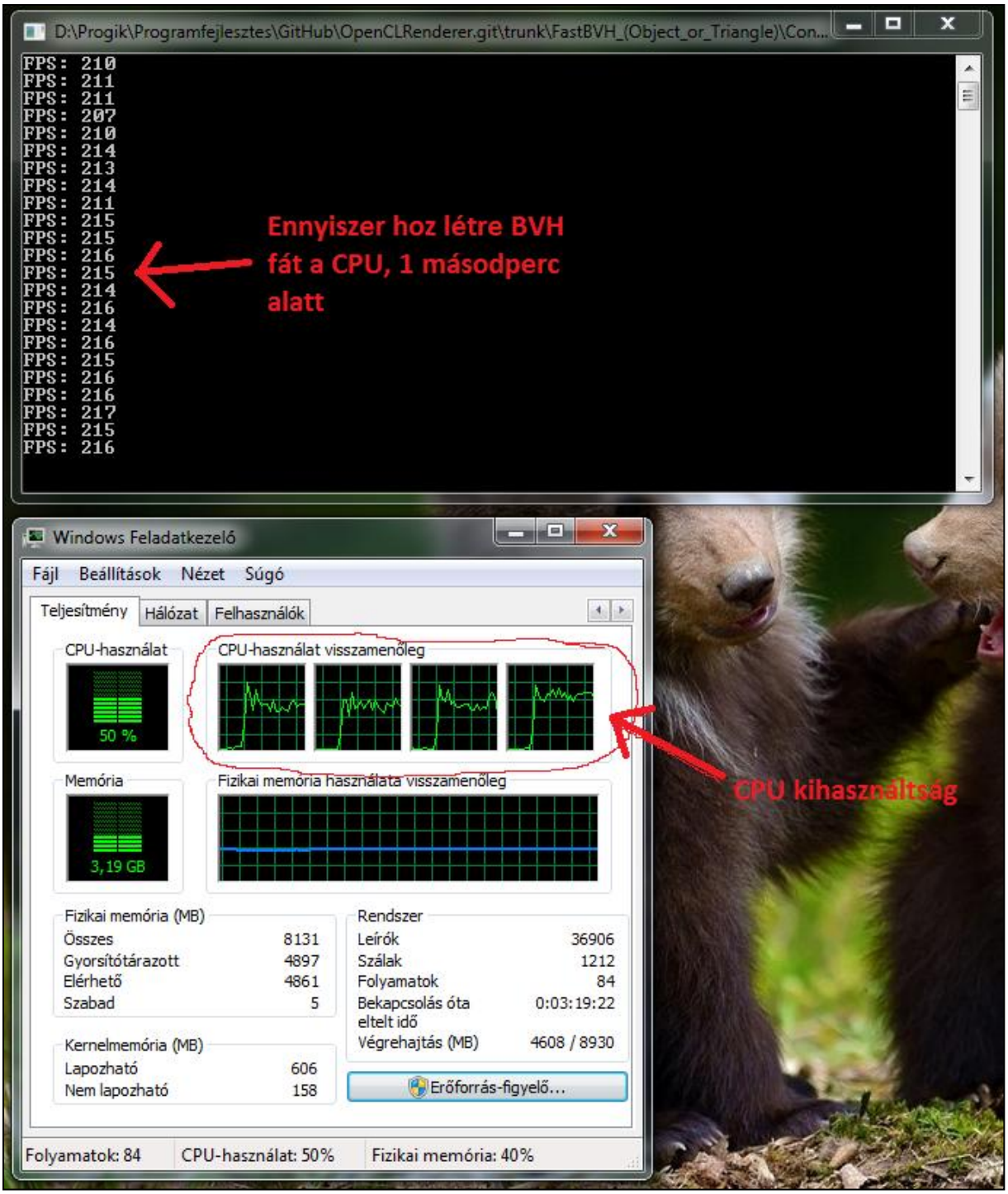
szintje, akkor OpenCL-el a VGA kiszámolhatja párhuzamosan a bounding box-okat, csak az aktuális szint OpenCL Bufferjét kell frissíteni a csomópontokkal.
Így gyorsan BVH fát lehet építeni.
Ez a BVH fa CPU segítségével jött létre. Egy színtér tartalmazhat akár több százezer háromszöget. Ez egy darab szálnak, túl sok idő, míg kiszámolja a BVH fát.
De lehet párhuzamosítani. Ha egy pl. 100.000 pontból álló listát 100 részlistára bontunk, a növekvő sorrendbe helyezés után, akkor a kapott 100 darab részlista, ami darabonként 1000 darab csúcsot ratratlmaz, lehet párhuzamosan, több szálon egyszerre számolni.



18. ábra: több szálra lehet bontani a szomszédok keresését. [saját]

Ahogyan a 17. ábra is mutatja, az 1. szálban lévő elemek nem használják a 2. szálban lévő elemeket, tehát nincs függőség, lehet párhuzamosítani. Manapság 4-16 magos egy CPU, tehát egyszerre akár 16 szál tud számolni, egymástól függetlenül. Vagyis 16x hamarabb befelyezi egy szál a számítást, mint ha egy szálon számolnánánk csak.

Fotos, először növekvő sorrendbe kell rendezni az egy listát, majd utána lehet darabolni a listatát, több részlistára.



19. ábra: teszt, i5 cpu [saját]

Én ezt a megoldást, amit fennt írtam csak félig programoztam le. A sebességre voltam kíváncsi. Nem számoltam OpenCL-el bounding box-okat. Háromszögekkel végeztem a kísérletet.

35.000 háromszöggel, a sorbarendezés 1 szálon történt a C# "Sort()" függvénnyel, a fa egy szintje 64 szálra/tömbbre lett darabolva. így másodpercenként 200x futott le a a bvh fa építés,

egy i5-ös (2. generációs) processzoron, ami 4 magos. Ennek a processzornak a használt piaci ára 10e ft.

Úgy gondolom, hogy ez jó sebesség. Ha azt nézzük, hogy amit megvalósítottam alkalmazást, abban for ciklussal járom be az objektumokat, ezt lehetne helyettesíteni azzal, hogy 1 darab BVH fát építek CPU-val, így benne a keresés jóval gyorsabb, mint ha n. darab BVH fán keresek.

Illetve gyorsítási lehetőség lehet még az, hogy ha megkülömböztetjük a mozdulatlan és az animált háromszögeket. A mozdulatlan háromszögeket tartalmazó BVH fát elég egyszer kiszámolni, míg csak az animált háromszögekre kell csak újra számolni minden frame-ben a BVH fát.

11. Jövőbeli tervek:

Jövőbeli terv lehet az, hogy amit a továbbfejlesztési lehetőségekben leírtam "gyors BVH fa építés"-ét megvalósítom.

Illetve terv lehet még az, hogy jelenleg nem használok a textúráknál "linear interpolation" -t, tehát nagyon pixeles egy textúra, ha közelről mutatja a kamera.

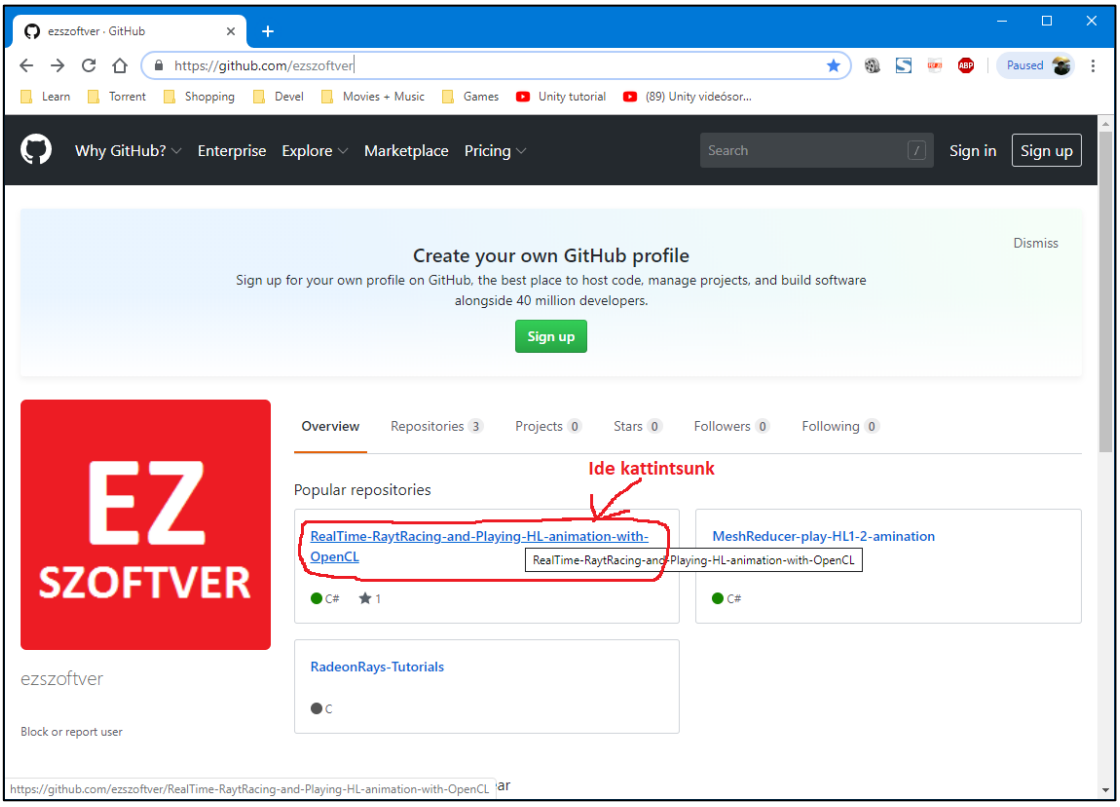
De mivel léteznek professzionális megoldások már, nem biztos hogy tovább foglalkozom ennek a leprogramozásával. Egy év múlva jelenik meg amd oldalon a Playstation 5, ami támogatni fogja a sugárkövetést. Illetve Nvidia oldalon ott van az "RTX", amik léteznek, műlködnek jól.

Az én megoldásomnak előnye lehet, hogy OpenCL és C# létezik sok rendszeren, windows7-en, linux-on, mac os-en, a régi vga-k is támogatják. Nem kell 100e ft-ért RTX-es Nvidia vga-t venni. Igaz, amd oldalon létezik C++ + OpenCL megoldás sugárkövetésre.

12. Felhasználói kézikönyv:

12.1 Letöltés:

Az alkalmazás és forráskód letöltését mutatom be. Először töltsük be ezt a weboldalt: https://github.com/ezszoftver

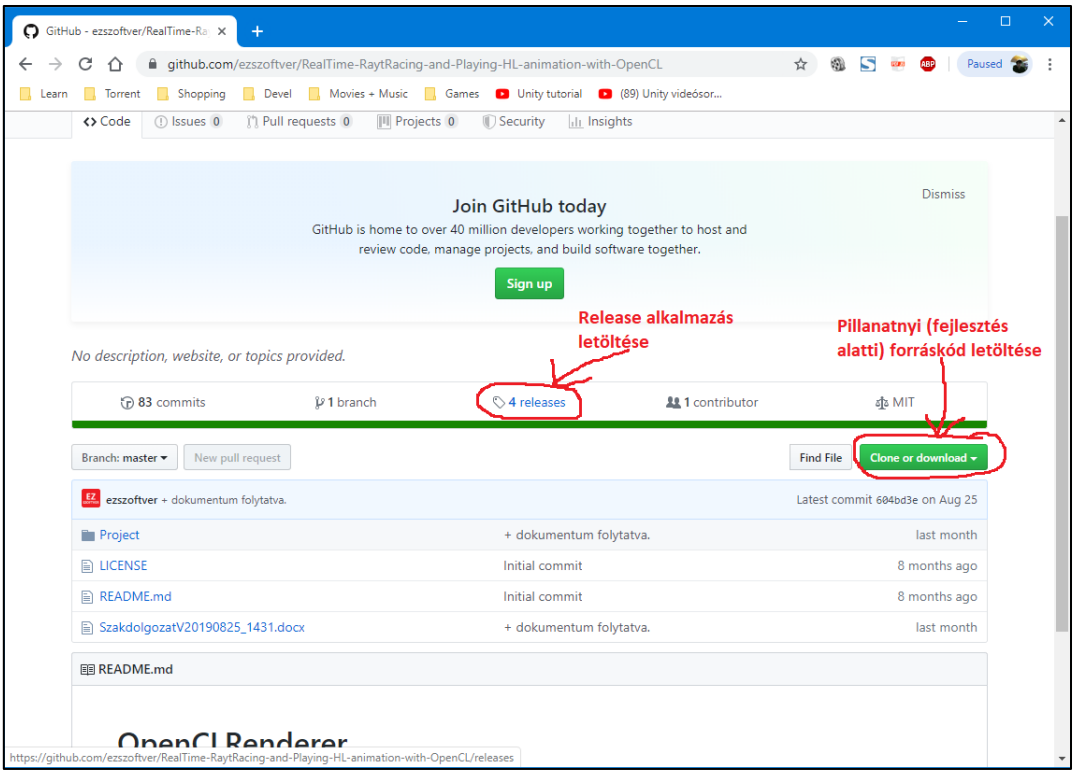


20. ábra: github.com/ezszoftver

Majd kattintsunk a „RealTime-RayTracing” linkre.

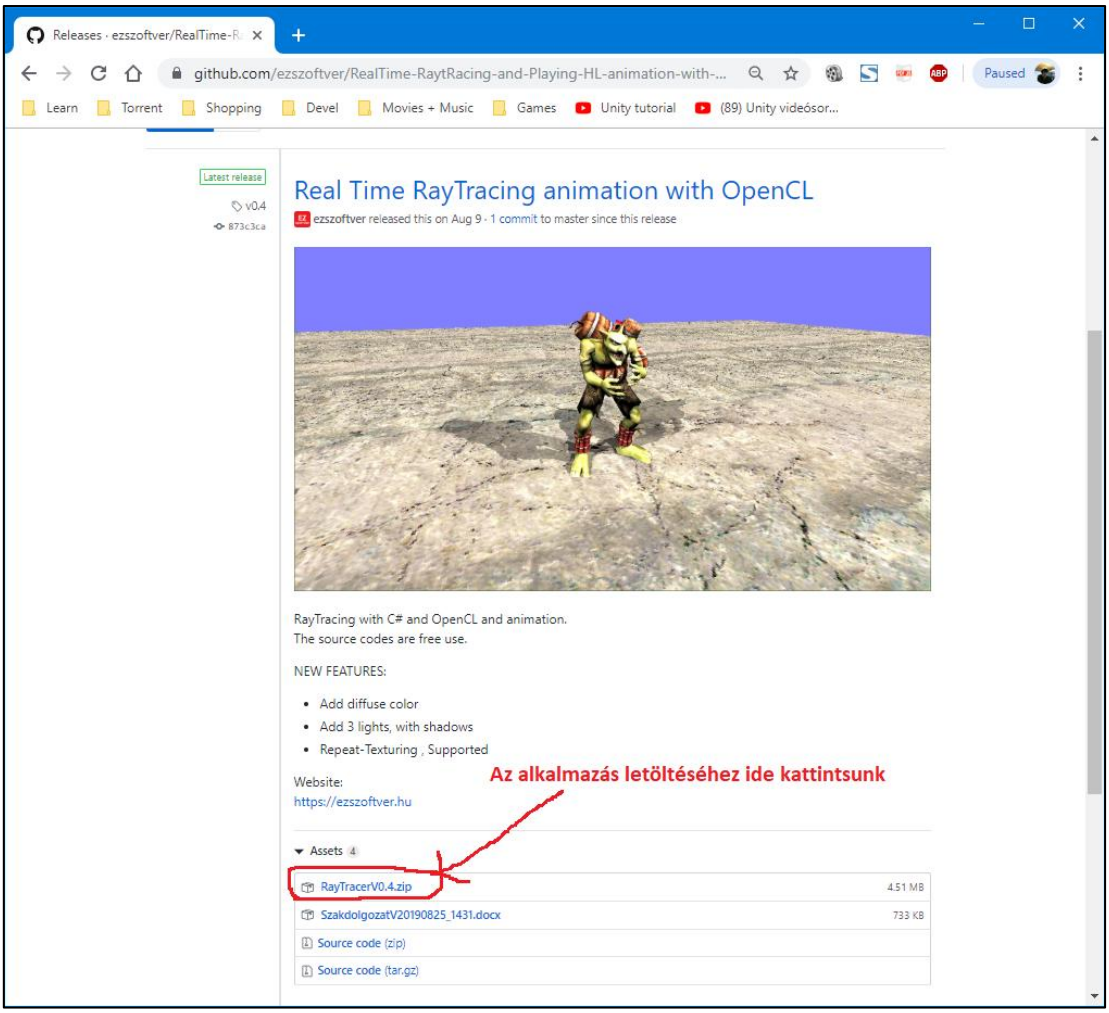
Ha a forráskódot akarjuk letölteni, akkor kattintsunk a „download” gombra. Ez a forráskód a pillanatnyi, nem kiadásra szánt forráskód.

Ha a Kiadásra szánt forráskódot és alkalmazást szeretnénk letölteni, akkor kattintsunk a „Releases” linkre.



21. ábra: forráskód vagy Release letöltése [saját]

Kattintsunk most a „Releases” linkre.



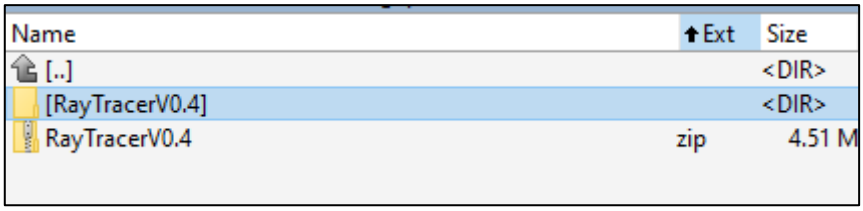
22. ábra: Itt lehet letölteni az alkalmazást [saját]

Majd az alkalmazás letöltéséhez kattintsunk a „RayTracerV0.4.zip” fájlra.

Ezzel letöltöttük az alkalmazást. Itt megtalálható az alkalmazás forráskódja is. Ez nem ugyan az, mint a pillanatnyi fejlesztés alatti forráskód.

12.2 Telepítés:

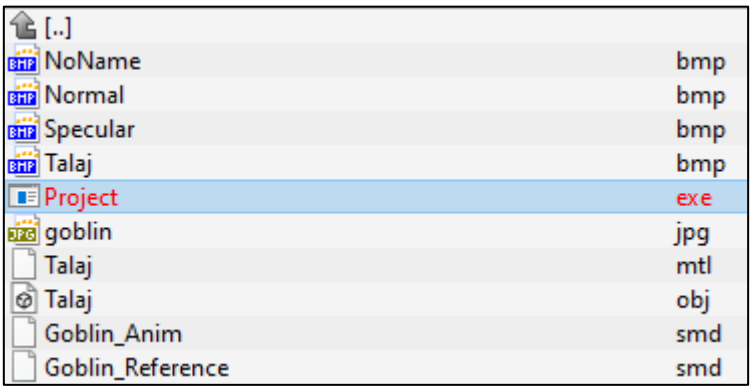
Csomagoljuk ki a letöltött .zip fájlt.



23. ábra: .zip fájl, kicsomagolva [saját]

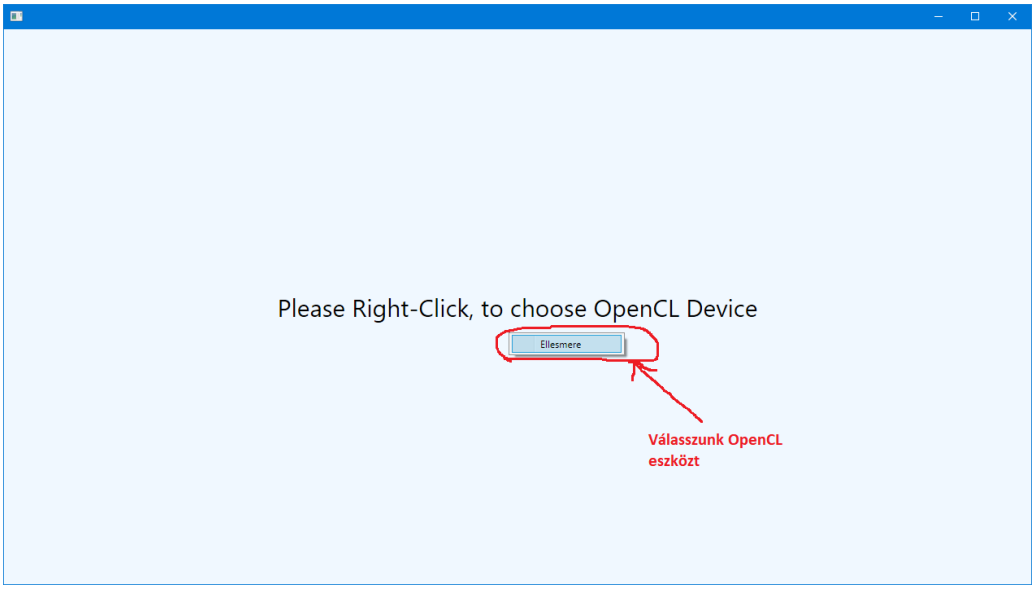
12.3 A program használata:

Indítsuk el a kitömörített mappában lévő „Project.exe” fájlt.



24. ábra: Alkalmazás elindítása [saját]

Az alábbi ablak fogad minket:



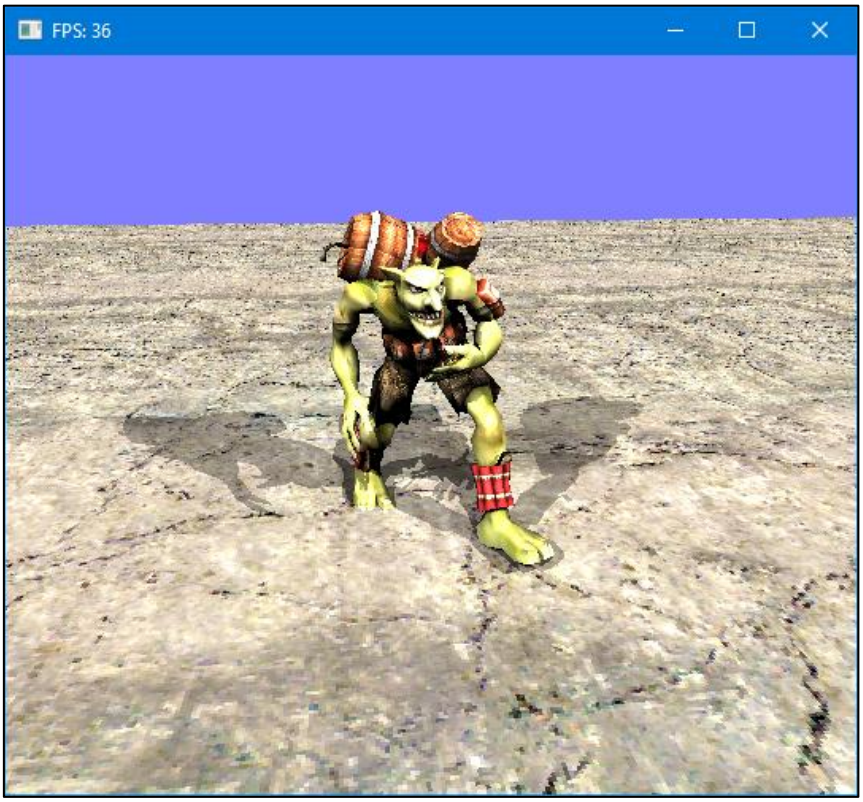
25. ábra: Válassz OpenCL eszközt [saját]

Az angol szöveg jelentése, „kattints jobb egérgombbal, az OpenCL eszköz kiválasztásához”.

Tegyünk így, válasszunk egy eszközt.

Fontos: Jelenleg OpenCL eszköz, csak GPU (vagyis videokártya) lehet. „CPU”, vagy „Accelerator” OpenCL eszköz nem kerül felsorolásra a helyi menüben.

Ha kiválasztottuk az eszközt, akkor elindul a RealTime Sugárkövetés példaprogram. Átméretezhető az ablak. Látható a fejlécben, hogy hány új kép készül egy másodperc alatt (FPS). Az alkalmazás bezárásához kattintsunk a jobb felső sarokban található „X” -re.



26. ábra: futó alkalmazás [saját]

13. Összefoglalás:

Úgy érzem, hogy sikerült a magamnak felállított célokat elérni. Sikerült egy majdnem használható sugárkövetés alkalmazást megírni, bemutatni. Azért majdnem használható, mert nagyon gépigényes ez az alkalmazás, mert for ciklussal járom be a BVH fákat, ez lassú.

Bemutattam hogyan lehet gyorsan sugár-háromszög metszéspontot számolni, animációt megjeleníteni. Az alkalmazásban, a "RayShader"-ben van példa, hogyan lehet árnyékot számolni sugárkövetéssel.

Illetve célom az is, hogy aki nem ismeri a sugárkövetést, ő most már könnyebben boldoguljon vele, értse az elméletet, hogy hogyan műlködik az én megoldásom.

Segítségemre volt az irodalomjegyzékben lévő leírások, amikből el lehet indulni.

Bemutattam a Vertex shadert, a Ray shadert, és azt, hogy hogyan lehet ezt OpenCL nyelven, ezt megvalósítani.

Sajnos nem platformfüggetlen az alkalmazás, mert WPF ablakot használok a kép megjelenítésére, ami csak windows-on érhető el. De ha a WPF ablakot lecseréljük pl. "Windows Forms" ablakozó rendszerre, akkor a forráskód, linuxon is lefordul a "MonoDevelop" alkalmazással. Tehát ha az ablakozó rendszert leszámítjuk, akkor forráskód szinten, platformfüggetlen az alkalmazás. C# és OpenCL API létezik Linux, Mac rendszerek alatt is.

Bemutattam hogyan lehet .OBJ fájlt betölteni, ami a mozdulatlan modell betöltésekor szóba jöhet, megoldásként.

Illetve bemutattam, hogy hogyan lehet .SMD fájlból csontanimációt betölteni, megjeleníteni. Remélem hogy az animáció megjelenítését érthetően írtam le, úgy, hogy aki nem ismeri ezt, ő ebből a leírásból megértette.

A sugárkövetés téma, régóta létezik, de eddig (2019), valós időben nem volt elterjedve. Új a téma, ha valós időben akarunk sugárkövetéssel képet megjeleníteni.

14. Irodalomjegyzék:

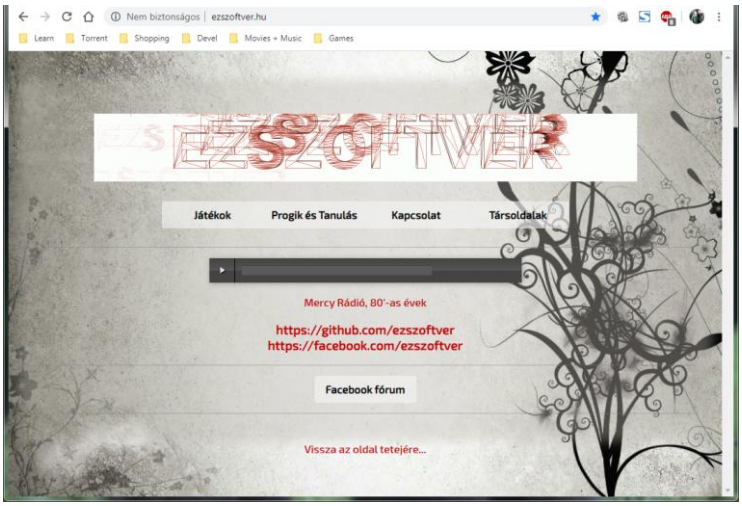
1. Szirmai-Kalos László, Csonka György, Csonka Ferenc: *Háromdimenzós grafika animáció és játékfejlesztés*, Computerbooks, 2005. pp. 486, ISBN: 9636183031.
2. DirectX12 RayTracing tutorials: https://developer.nvidia.com/rtx/raytracing/dxr/DX12-Raytracing-tutorial-Part-1, látogatva: 2020.02.16
3. NVIDIA RTX and DirectX Ry Tracing: https://devblogs.nvidia.com/introduction-nvidia-rtx-directx-ray-tracing/, látogatva: 2020.02.17
4. RadeonRays 2.0 SDK: https://gpuopen.com/gaming-product/radeon-rays/, GitHub: https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays_SDK, látogatva: 2019.09.27
5. RadeonRays 3.0 SDK: https://www.amd.com/en/technologies/sdk-agreement, látogatva: 2020.02.16
6. NVIDIA Vulkan Raytracing Tutorial: https://developer.nvidia.com/rtx/raytracing/vkray, látogatva: 2020.02.16
7. Vulkan, RayTracing tutorial (C++): https://iorange.github.io/, látogatva: 2020.02.16
8. GPUOpen Libraries: https://github.com/GPUOpen-LibrariesAndSDKs, látogatva: 2020.02.16
9. 3D C++ tutorials: http://www.3dcpptutorials.sk/index.php?id=16, látogatva: 2020.02.16
10. Scratchpixel 2.0: https://www.scratchapixel.com/index.php?redirect, látogatva: 2020.02.17
11. Global Illumination in 99 lines: http://www.kevinbeason.com/smallpt/, látogatva: 202002.17
12. RayTracing in One Weekend: http://in1weekend.blogspot.com/2016/01/ray-tracing-in-one-weekend.html, látogatva: 2020.02.17
13. Daily Pathtracer: http://aras-p.info/blog/2018/03/28/Daily-Pathtracer-Part-0-Intro/, látogatva: 2020.02.17
14. Erdős Zoltán honlapja: https://ezszoftver.hu/, látogatva: 2019.09.27.
15. Erdős Zoltán: EZSzoftver lapja a GitHubon: http://github.com/ezszoftver, látogatva: 2019.09.27

16. Erdős Zoltán: RealTime RayTracing, with Animation alkalmazás forráskódja, https://github.com/ezszoftver/RealTime-RaytRacing-and-Playing-HL-animation-with-OpenCL/releases, látogatva: 2019.09.27
17. Erdős Zoltán: 3D-s animációt lejátszó és objektum-poligonszámot csökkentő alkalmazás forráskódja, https://github.com/ezszoftver/MeshReducer-play-HL1-2-amination/releases, látogatva: 2019.09.27

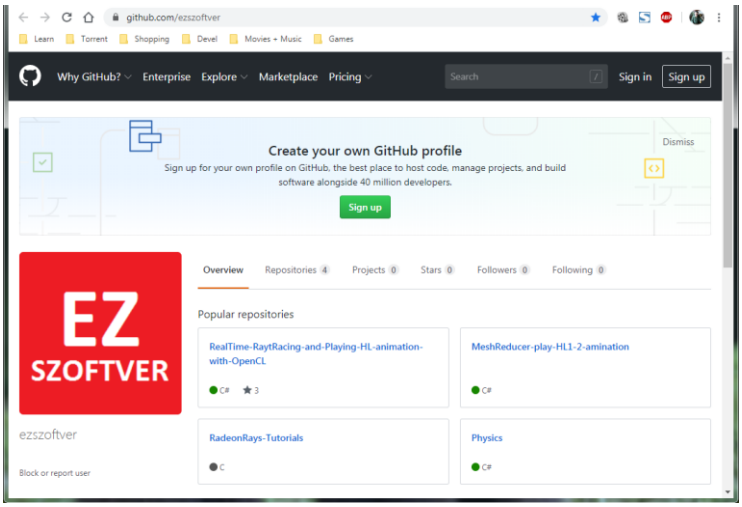
15. Ábrajegyzék:

16. Mellékletek:

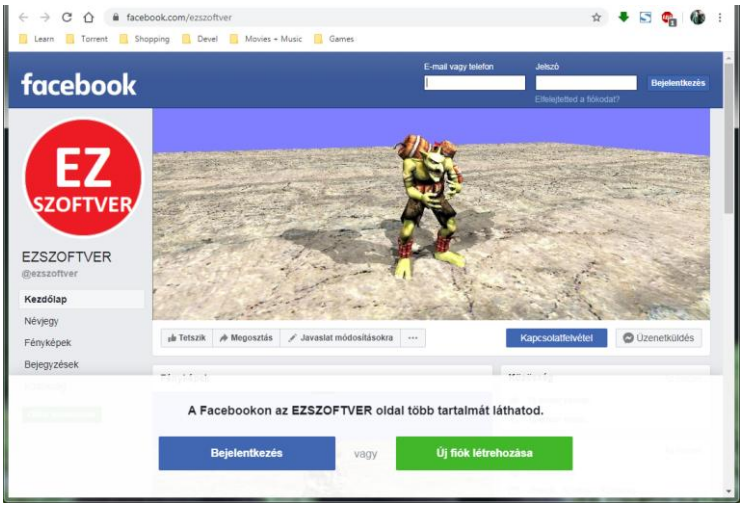
Saját készítésű ingyenes játékaimat és azok forráskódját saját weboldalamon, a https://ezszoftver.hu/-n osztom meg.



Saját készítésű forráskódok: csontanimáció, sugárkövetés + dokumentumok a https://github.com/ezszoftver/ -n osztom meg



Facebook fórum: https://facebook.com/ezszoftver/ -n érhető el.



17. Forráskódok:

17.1 MainWindow.xaml

```
<Window x:Class="Project.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:wf="clr-namespace:System.Windows.Forms;assembly=System.Windows.Forms"
        xmlns:local="clr-namespace:Project"
        mc:Ignorable="d"
        Width="720" Height="480"
        MinWidth="256" MinHeight="256"
        Title=""
        WindowStartupLocation="CenterScreen"
        Closing="Window_Closing"
        >
    <Grid Background="AliceBlue">
        <Label x:Name="labelMessage" Content="Please Right-Click, to choose OpenCL Device" FontSize="25"
Foreground="Black" HorizontalContentAlignment="Center" VerticalContentAlignment="Center"/>
        <Image x:Name="image" Stretch="Fill" RenderOptions.BitmapScalingMode="NearestNeighbor"
SizeChanged="Image_SizeChanged" RenderTransformOrigin="0.5,0.5">
            <Image.RenderTransform>
                <ScaleTransform ScaleY="-1.0"/>
            </Image.RenderTransform>
        </Image>

    </Grid>
</Window>
```

17.2 MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Threading;

using OpenTK;
using OpenTK.Graphics;
using OpenTK.Graphics.OpenGL;

namespace Project
{
   /// <summary>
   /// Interaction logic for MainWindow.xaml
   /// </summary>
   public partial class MainWindow : Window
   {
      public MainWindow()
      {
         InitializeComponent();

         m_Scene = new OpenCLRenderer.Scene();
         List<string> listDevices = m_Scene.GetDevices();
```

```
    m_Scene.Dispose();
    m_Scene = null;

    ContextMenu contextMenu = new ContextMenu();
    foreach (string strDevice in listDevices)
    {
        MenuItem item = new MenuItem();
        item.Header = strDevice;
        item.Click += Item_Click;

        contextMenu.Items.Add(item);
    }
    ContextMenu = contextMenu;
}

private void Item_Click(object sender, RoutedEventArgs e)
{
    labelMessage.IsEnabled = false;

    if (null != m_Timer)
    {
        m_Timer.Stop();
        m_Timer = null;
    }

    if (null != m_Scene)
    {
        m_Scene.Dispose();
        m_Scene = null;
    }

    string strDeviceName = (sender as MenuItem).Header.ToString();
    m_Scene = new OpenCLRenderer.Scene();

    string strVertexShader = File.ReadAllText("VertexShader.txt");
    string strRayShader = File.ReadAllText("RayShader.txt");

    m_Scene.CreateDevice(strDeviceName, @strVertexShader, @strRayShader);

    //Mutex mtxMutex = new Mutex();
```

```
        Random rand = new Random();

        {
            // load from obj file
            string strDirectory = @".\";
            OBJLoader objLoader = new OBJLoader();

            //mtxMutex.WaitOne();
            objLoader.LoadFromFile(@strDirectory, @"Talaj.obj");
            //mtxMutex.ReleaseMutex();

            // convert to triangle list
            //int iMatrixId = m_Scene.GenMatrix();
            //m_Scene.SetMatrix(iMatrixId, Matrix4.Identity);
            int iMatrixId = 0;


            List<OpenCLRenderer.Triangle> triangles = new List<OpenCLRenderer.Triangle>();
            foreach (OBJLoader.Material material in objLoader.materials)
            {
                string strDiffuseTextureName = @strDirectory + @material.texture_filename;
                string strSpecularTextureName = @strDirectory + @"Specular.bmp";
                string strNormalTextureName = @strDirectory + @"Normal.bmp";

                int iMaterialId = m_Scene.GenMaterial();
                m_Scene.SetMaterial(iMaterialId, @strDiffuseTextureName, @strSpecularTextureName,
@strNormalTextureName);

                for (int i = 0; i < material.indices.Count; i += 3)
                {
                    Vector3 vA = objLoader.vertices[material.indices[i + 0].id_vertex];
                    Vector3 vB = objLoader.vertices[material.indices[i + 1].id_vertex];
                    Vector3 vC = objLoader.vertices[material.indices[i + 2].id_vertex];

                    Vector3 nA = objLoader.normals[material.indices[i + 0].id_normal];
                    Vector3 nB = objLoader.normals[material.indices[i + 1].id_normal];
                    Vector3 nC = objLoader.normals[material.indices[i + 2].id_normal];

                    Vector2 tA = objLoader.text_coords[material.indices[i + 0].id_textcoord];
```

```
Vector2 tB = objLoader.text_coords[material.indices[i + 1].id_textcoord];
Vector2 tC = objLoader.text_coords[material.indices[i + 2].id_textcoord];

OpenCLRenderer.Vertex vertexA = new OpenCLRenderer.Vertex();
vertexA.m_Vx = vA.X;
vertexA.m_Vy = vA.Y;
vertexA.m_Vz = vA.Z;
vertexA.m_Nx = nA.X;
vertexA.m_Ny = nA.Y;
vertexA.m_Nz = nA.Z;
vertexA.m_TCx = tA.X;
vertexA.m_TCy = tA.Y;
vertexA.m_iNumMatrices = 1;
vertexA.m_iMatrixId1 = iMatrixId;
vertexA.m_fWeight1 = 1.0f;
vertexA.m_iMatrixId2 = -1;
vertexA.m_fWeight2 = 0.0f;
vertexA.m_iMatrixId3 = -1;
vertexA.m_fWeight3 = 0.0f;

OpenCLRenderer.Vertex vertexB = new OpenCLRenderer.Vertex();
vertexB.m_Vx = vB.X;
vertexB.m_Vy = vB.Y;
vertexB.m_Vz = vB.Z;
vertexB.m_Nx = nB.X;
vertexB.m_Ny = nB.Y;
vertexB.m_Nz = nB.Z;
vertexB.m_TCx = tB.X;
vertexB.m_TCy = tB.Y;
vertexB.m_iNumMatrices = 1;
vertexB.m_iMatrixId1 = iMatrixId;
vertexB.m_fWeight1 = 1.0f;
vertexB.m_iMatrixId2 = -1;
vertexB.m_fWeight2 = 0.0f;
vertexB.m_iMatrixId3 = -1;
vertexB.m_fWeight3 = 0.0f;

OpenCLRenderer.Vertex vertexC = new OpenCLRenderer.Vertex();
vertexC.m_Vx = vC.X;
vertexC.m_Vy = vC.Y;
```

```
                vertexC.m_Vz = vC.Z;
                vertexC.m_Nx = nC.X;
                vertexC.m_Ny = nC.Y;
                vertexC.m_Nz = nC.Z;
                vertexC.m_TCx = tC.X;
                vertexC.m_TCy = tC.Y;
                vertexC.m_iNumMatrices = 1;
                vertexC.m_iMatrixId1 = iMatrixId;
                vertexC.m_fWeight1 = 1.0f;
                vertexC.m_iMatrixId2 = -1;
                vertexC.m_fWeight2 = 0.0f;
                vertexC.m_iMatrixId3 = -1;
                vertexC.m_fWeight3 = 0.0f;

                OpenCLRenderer.Triangle newTriangle = new OpenCLRenderer.Triangle();
                newTriangle.m_A = vertexA;
                newTriangle.m_B = vertexB;
                newTriangle.m_C = vertexC;
                newTriangle.m_iMaterialId = iMaterialId;

                triangles.Add(newTriangle);
            }
        }

        int iId;
        iId = m_Scene.GenObject();
        OpenCLRenderer.BVHObject staticObject = m_Scene.CreateStaticObject(triangles,
Matrix4.CreateScale(7.0f,3.0f,7.0f));
        m_Scene.SetObject(iId, staticObject);
        objLoader.Release();
        triangles.Clear();

        // load from obj file
        strDirectory = @".\";
        smd = new SMDLoader();
        mesh = new Mesh();

        //mtxMutex.WaitOne();

        // HL1
```

```
            smd.LoadReference(@strDirectory, @"Goblin_Reference.smd", mesh);
            smd.AddAnimation(@strDirectory, @"Goblin_Anim.smd", "Anim1", 30.0f);

            // HL2
            //smd.LoadReference(@strDirectory, @"Antlion_guard_reference.smd", mesh);
            //smd.AddAnimation(@strDirectory, @"Antlion_idle.smd", "Anim1", 30.0f);

            smd.SetAnimation("Anim1");

            iMatrixOffset = m_Scene.NumMatrices();
            for (int i = 0; i < mesh.transforms.Count; i++)
            {
               iMatrixId = m_Scene.GenMatrix();
               m_Scene.SetMatrix(iMatrixId, smd.GetMatrix(i) * Matrix4.CreateRotationX(-1.57f) *
Matrix4.CreateRotationY(3.14f));
            }

            //int iMatrixId = m_Scene.GenMatrix();
            //m_Scene.SetMatrix(iMatrixId, Matrix4.CreateRotationX(-1.57f) * Matrix4.CreateRotationY(3.14f));

            //mtxMutex.ReleaseMutex();

            triangles = new List<OpenCLRenderer.Triangle>();
            foreach (Mesh.Material material in mesh.materials)
            {
               string strDiffuseTextureName = @strDirectory + @material.texture_name;
               string strSpecularTextureName = @strDirectory + @"Specular.bmp";
               string strNormalTextureName = @strDirectory + @"Normal.bmp";

               int iMaterialId = m_Scene.GenMaterial();
               m_Scene.SetMaterial(iMaterialId, @strDiffuseTextureName, @strSpecularTextureName,
@strNormalTextureName);

               for (int i = 0; i < material.vertices.Count(); i += 3)
               {
                  // add triangle
                  Mesh.Vertex meshVertexA = material.vertices[i + 0];
                  Mesh.Vertex meshVertexB = material.vertices[i + 1];
                  Mesh.Vertex meshVertexC = material.vertices[i + 2];
```

```
Vector3 vA = meshVertexA.vertex;
Vector3 vB = meshVertexB.vertex;
Vector3 vC = meshVertexC.vertex;

// none normal vector
Vector3 nA = meshVertexA.normal;
Vector3 nB = meshVertexB.normal;
Vector3 nC = meshVertexC.normal;

Vector2 tA = meshVertexA.textcoords;
Vector2 tB = meshVertexB.textcoords;
Vector2 tC = meshVertexC.textcoords;

OpenCLRenderer.Vertex vertexA = new OpenCLRenderer.Vertex();
vertexA.m_Vx = vA.X;
vertexA.m_Vy = vA.Y;
vertexA.m_Vz = vA.Z;
vertexA.m_Nx = nA.X;
vertexA.m_Ny = nA.Y;
vertexA.m_Nz = nA.Z;
vertexA.m_TCx = tA.X;
vertexA.m_TCy = tA.Y;
vertexA.m_iNumMatrices = meshVertexA.matrices.Count;
for(int j = 0; j < vertexA.m_iNumMatrices; j++)
{
   if (j == 0)
   {
      vertexA.m_iMatrixId1 = iMatrixOffset + meshVertexA.matrices[j].matrix_id;
      vertexA.m_fWeight1 = meshVertexA.matrices[j].weight;
   }
   if (j == 1)
   {
      vertexA.m_iMatrixId2 = iMatrixOffset + meshVertexA.matrices[j].matrix_id;
      vertexA.m_fWeight2 = meshVertexA.matrices[j].weight;
   }
   if (j == 2)
   {
      vertexA.m_iMatrixId3 = iMatrixOffset + meshVertexA.matrices[j].matrix_id;
      vertexA.m_fWeight3 = meshVertexA.matrices[j].weight;
   }
```

```
}

OpenCLRenderer.Vertex vertexB = new OpenCLRenderer.Vertex();
vertexB.m_Vx = vB.X;
vertexB.m_Vy = vB.Y;
vertexB.m_Vz = vB.Z;
vertexB.m_Nx = nB.X;
vertexB.m_Ny = nB.Y;
vertexB.m_Nz = nB.Z;
vertexB.m_TCx = tB.X;
vertexB.m_TCy = tB.Y;
vertexB.m_iNumMatrices = meshVertexB.matrices.Count;
for (int j = 0; j < vertexB.m_iNumMatrices; j++)
{
   if (j == 0)
   {
      vertexB.m_iMatrixId1 = iMatrixOffset + meshVertexB.matrices[j].matrix_id;
      vertexB.m_fWeight1 = meshVertexB.matrices[j].weight;
   }
   if (j == 1)
   {
      vertexB.m_iMatrixId2 = iMatrixOffset + meshVertexB.matrices[j].matrix_id;
      vertexB.m_fWeight2 = meshVertexB.matrices[j].weight;
   }
   if (j == 2)
   {
      vertexB.m_iMatrixId3 = iMatrixOffset + meshVertexB.matrices[j].matrix_id;
      vertexB.m_fWeight3 = meshVertexB.matrices[j].weight;
   }
}

OpenCLRenderer.Vertex vertexC = new OpenCLRenderer.Vertex();
vertexC.m_Vx = vC.X;
vertexC.m_Vy = vC.Y;
vertexC.m_Vz = vC.Z;
vertexC.m_Nx = nC.X;
vertexC.m_Ny = nC.Y;
vertexC.m_Nz = nC.Z;
vertexC.m_TCx = tC.X;
vertexC.m_TCy = tC.Y;
```

```
            vertexC.m_iNumMatrices = meshVertexC.matrices.Count;
            for (int j = 0; j < vertexC.m_iNumMatrices; j++)
            {
               if (j == 0)
               {
                  vertexC.m_iMatrixId1 = iMatrixOffset + meshVertexC.matrices[j].matrix_id;
                  vertexC.m_fWeight1 = meshVertexC.matrices[j].weight;
               }
               if (j == 1)
               {
                  vertexC.m_iMatrixId2 = iMatrixOffset + meshVertexC.matrices[j].matrix_id;
                  vertexC.m_fWeight2 = meshVertexC.matrices[j].weight;
               }
               if (j == 2)
               {
                  vertexC.m_iMatrixId3 = iMatrixOffset + meshVertexC.matrices[j].matrix_id;
                  vertexC.m_fWeight3 = meshVertexC.matrices[j].weight;
               }
            }

            OpenCLRenderer.Triangle newTriangle = new OpenCLRenderer.Triangle();
            newTriangle.m_A = vertexA;
            newTriangle.m_B = vertexB;
            newTriangle.m_C = vertexC;
            newTriangle.m_iMaterialId = iMaterialId;

            triangles.Add(newTriangle);
         }
      }

      //int iId;
      iId = m_Scene.GenObject();
      OpenCLRenderer.BVHObject dynamicObject = m_Scene.CreateDynamicObject(triangles);
      m_Scene.SetObject(iId, dynamicObject);
      triangles.Clear();
   }

   m_Scene.Commit();

   Image_SizeChanged(null, null);
```

```
        m_Timer = new DispatcherTimer();
        m_Timer.Tick += Timer_Tick;
        m_Timer.Interval = TimeSpan.FromMilliseconds(0);

        m_ElapsedTime = m_CurrentTime = DateTime.Now;
        m_fSec = 0.0f;
        m_fFullTime = 0.0f;
        m_Timer.Start();
    }

    private void Timer_Tick(object sender, EventArgs e)
    {
        // delta time
        m_ElapsedTime = m_CurrentTime;
        m_CurrentTime = DateTime.Now;
        m_fDeltaTime = (float)(m_CurrentTime - m_ElapsedTime).TotalSeconds;


        // FPS
        FPS++;
        m_fSec += m_fDeltaTime;
        if (m_fSec >= 1.0f)
        {
            this.Title = "FPS: " + FPS;
            m_fSec = 0.0f;
            FPS = 0;
        }

        // smd update
        time += m_fDeltaTime;
        while (time >= smd.GetFullTime())
        {
            time -= smd.GetFullTime();
        }
        smd.SetTime(time, mesh);
        for (int i = 0; i < mesh.transforms.Count; i++)
        {
            m_Scene.SetMatrix(iMatrixOffset + i, mesh.transforms[i] * Matrix4.CreateRotationX(-1.57f) *
Matrix4.CreateRotationY(3.14f));
```

```
        }

        m_Scene.UpdateMatrices();
        m_Scene.RunTriangleShader();
        m_Scene.RunRefitTreeShader();

        m_fFullTime += m_fDeltaTime;
        float fSpeed = 0.5f;
        m_Scene.SetCamera(new Vector3(-45.0f * (float)Math.Cos(m_fFullTime * fSpeed), 30, -45.0f *
(float)Math.Sin(m_fFullTime * fSpeed)), new Vector3(0, 5, 0), new Vector3(0, 1, 0), (float)Math.PI / 4.0f,
1000.0f);
        m_Scene.RunRayShader(127, 127, 255, 255);

        image.Source = m_Scene.GetWriteableBitmap();
    }

    OpenCLRenderer.Scene m_Scene = null;
    DispatcherTimer m_Timer = null;
    int FPS = 0;
    DateTime m_ElapsedTime;
    DateTime m_CurrentTime;
    float m_fDeltaTime;
    float m_fSec;
    float m_fFullTime;

    SMDLoader smd;
    Mesh mesh;
    float time = 0.0f;
    int iMatrixOffset;

    private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
    {
        if (null != m_Timer)
        {
            m_Timer.Stop();
            m_Timer = null;
        }

        if (null != m_Scene)
        {
```

```
            m_Scene.Dispose();
            m_Scene = null;
        }
    }

    private void Image_SizeChanged(object sender, SizeChangedEventArgs e)
    {
        if (null == m_Scene) { return; }

        int iWidth = (int)image.ActualWidth;
        int iHeight = (int)image.ActualHeight;

        if (iWidth == 0 || iHeight == 0) { return; }
        m_Scene.Resize(iWidth, iHeight);
    }


  }
}
```

17.3 Mesh.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using OpenTK;

namespace Project
{
   class Mesh
   {
      public class MatrixIdAndWeight
      {
         public MatrixIdAndWeight(int matrix_id, float weight)
         {
            this.weight = weight;
            this.matrix_id = matrix_id;
         }

         public int matrix_id;
         public float weight;
      }

      public class Vertex
      {
         public Vertex(Vector3 vertex, Vector3 normal, Vector2 textcoords)
         {
            this.vertex = vertex;
            this.normal = normal;
            this.textcoords = textcoords;
            this.matrices = new List<MatrixIdAndWeight>();
         }

         public Vertex(Vertex b)
         {
            vertex = new Vector3(b.vertex.X, b.vertex.Y, b.vertex.Z);
            normal = new Vector3(b.normal.X, b.normal.Y, b.normal.Z);
            textcoords = new Vector2(b.textcoords.X, b.textcoords.Y);
```

```
        matrices = new List<MatrixIdAndWeight>();
        foreach (MatrixIdAndWeight b_matrix in b.matrices)
        {
            MatrixIdAndWeight matrix = new MatrixIdAndWeight(b_matrix.matrix_id, b_matrix.weight);
            matrices.Add(matrix);
        }
    }

    public void AddMatrix(int matrix_id, float weight)
    {
        matrices.Add(new MatrixIdAndWeight(matrix_id, weight));
    }

    public int GetMaxWeightID()
    {
        int matrix_id = 0; float max_weight = 0.0f;

        for (int i = 0; i < matrices.Count; i++)
        {
            if (matrices[i].weight > max_weight)
            {
                max_weight = matrices[i].weight;
                matrix_id = matrices[i].matrix_id;
            }
        }

        return matrix_id;
    }

    public Vector3 vertex;
    public Vector3 normal;
    public Vector2 textcoords;
    public List<MatrixIdAndWeight> matrices;
}

public class Material
{
    public string texture_name;

    public List<Vertex> vertices;
```

```
    public Material(string texture_name)
    {
        vertices = new List<Vertex>();
        this.texture_name = texture_name;
    }
}

public List<Matrix4> inverse_transforms_reference;
public List<Matrix4> transforms;
public List<Material>  materials;
public Vector3 min, max;
public bool is_loaded;

public string mtllib;
public bool is_obj;
public bool is_hl1;
public bool is_hl2;

public Mesh()
{
    is_loaded = false;
    transforms = new List<Matrix4>();
    inverse_transforms_reference = new List<Matrix4>();
    materials = new List<Material>();
    min = new Vector3(0, 0, 0);
    max = new Vector3(0, 0, 0);
    mtllib = "";
    is_obj = false;
    is_hl1 = false;
    is_hl2 = false;
}

public Mesh(Mesh b)
{
    is_loaded = b.is_loaded;
    mtllib = b.mtllib;
    is_obj = b.is_obj;
    is_hl1 = b.is_hl1;
    is_hl2 = b.is_hl2;
```

```
transforms = new List<Matrix4>();
foreach (Matrix4 b_m in b.transforms)
{
   Matrix4 m = new Matrix4(b_m.Row0, b_m.Row1, b_m.Row2, b_m.Row3);
   transforms.Add(m);
}

inverse_transforms_reference = new List<Matrix4>();
foreach (Matrix4 b_m in b.inverse_transforms_reference)
{
   Matrix4 m = new Matrix4(b_m.Row0, b_m.Row1, b_m.Row2, b_m.Row3);
   inverse_transforms_reference.Add(m);
}

materials = new List<Material>();
min = new Vector3(+1000000.0f, +1000000.0f, +1000000.0f);
max = new Vector3(-1000000.0f, -1000000.0f, -1000000.0f);

foreach (Material b_mat in b.materials)
{
   Material mat = new Material(b_mat.texture_name);

   foreach (Vertex b_v in b_mat.vertices)
   {
      Vertex v = new Vertex(b_v);

      // min
      if (v.vertex.X < min.X) { min.X = v.vertex.X; }
      if (v.vertex.Y < min.Y) { min.Y = v.vertex.Y; }
      if (v.vertex.Z < min.Z) { min.Z = v.vertex.Z; }
      // max
      if (max.X < v.vertex.X) { max.X = v.vertex.X; }
      if (max.Y < v.vertex.Y) { max.Y = v.vertex.Y; }
      if (max.Z < v.vertex.Z) { max.Z = v.vertex.Z; }

      mat.vertices.Add(v);
   }

   materials.Add(mat);
```

```
    }
}

public int GetVerticesCount()
{
    int count = 0;

    foreach (Material material in materials)
    {
        foreach (Vertex vertex in material.vertices)
        {
            if (vertex == null) continue;
            count++;
        }
    }

    return count;
}

public void Release(bool is_delete_textures)
{
    foreach (Material material in materials)
    {
        foreach (Vertex vertex in material.vertices)
        {
            if (vertex == null) { continue; }

            if (vertex.matrices != null)
            {
                vertex.matrices.Clear();
                vertex.matrices = null;
            }
        }

        material.vertices.Clear();
        material.vertices = null;
    }

    transforms.Clear();
    transforms = null;
```

```
            if (is_delete_textures) { materials.Clear(); }
        }
    }
}
```

17.4 OBJLoader.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using OpenTK;

namespace Project
{
   class OBJLoader
   {
      public List<Vector3> vertices;
      public List<Vector3> normals;
      public List<Vector2> text_coords;

      private Dictionary<string, UInt32> material_to_id;
      private Dictionary<string, string> material_to_texture;

      public class Vertex
      {
         public Int32 id_vertex;
         public Int32 id_textcoord;
         public Int32 id_normal;

         public Vertex(Int32 id_vertex, Int32 id_textcoord, Int32 id_normal)
         {
            this.id_vertex = id_vertex;
            this.id_textcoord = id_textcoord;
            this.id_normal = id_normal;
         }
      }

      public class Material
      {
         public string texture_filename;
```

```
        public List<Vertex> indices;

        public Material(string texture_filename)
        {
            indices = new List<Vertex>();
            this.texture_filename = texture_filename;
        }

        public void Release()
        {
            indices.Clear();
            indices = null;
        }
    };

    public List<Material> materials;

    public OBJLoader()
    {
        vertices = new List<Vector3>();
        normals = new List<Vector3>();
        text_coords = new List<Vector2>();
        material_to_id = new Dictionary<string, UInt32>();
        material_to_texture = new Dictionary<string, string>();
        materials = new List<Material>();
    }

    public bool LoadFromFile(string directory, string filename)
    {
        string[] lines = File.ReadAllText(directory + filename).Split(new char[] {'\r', '\n'},
StringSplitOptions.RemoveEmptyEntries);

        UInt32 material_id = 0;
        string mtllib = "";

        NumberFormatInfo formatInfo = CultureInfo.CreateSpecificCulture("en-US").NumberFormat;

        foreach (string line in lines) {
            string[] words = line.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
```

```
            switch (words[0]) {
                case ("#"): {
                        break;
                    }
                case ("v"): {
                        Vector3 v = new Vector3(float.Parse(words[1], formatInfo), float.Parse(words[2], formatInfo),
float.Parse(words[3], formatInfo));
                        vertices.Add(v);
                        break;
                    }
                case ("vn"):
                    {
                        Vector3 n = new Vector3(float.Parse(words[1], formatInfo), float.Parse(words[2], formatInfo),
float.Parse(words[3], formatInfo));
                        normals.Add(n);
                        break;
                    }
                case ("vt"): {
                        Vector2 vt = new Vector2(float.Parse(words[1], formatInfo), float.Parse(words[2],
formatInfo));
                        text_coords.Add(vt);
                        break;
                    }
                case ("mtllib"): {
                        mtllib = words[1];
                        string[] mtl_lines = File.ReadAllText(directory + @"/" + words[1]).Split(new char[] {'\r', '\n'},
StringSplitOptions.RemoveEmptyEntries);

                        string material = "";
                        string texture_name = "";
                        bool is_save = true;

                        foreach (string mtl_line in mtl_lines) {
                            string[] mtl_words = mtl_line.Split(new char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries);

                            switch (mtl_words[0]) {
                                case ("#"): {
                                        break;
                                    }
```

```
            case ("newmtl"): {
                if (is_save == false) {
                    texture_name = "NoName.bmp";

                    materials.Add(new Material(texture_name));
                    material_to_texture.Add(material, texture_name);
                    int id = material_to_id.Count;
                    material_to_id.Add(material, (UInt32)id);

                    material = "";
                }

                material = mtl_words[1];
                is_save = false;
                break;
            }
            case ("map_Kd"): {
                texture_name = mtl_words[1];

                materials.Add(new Material(texture_name));
                material_to_texture.Add(material, texture_name);
                int id = material_to_id.Count;
                material_to_id.Add(material, (UInt32)id);

                material = "";
                is_save = true;
                break;
            }
        }
    }

    break;
  }
case ("usemtl"): {
    material_id = material_to_id[words[1]];
    break;
  }
case ("f"): {
    Vertex first = null;
    for (int i = 1; i < words.Count(); i++) {
```

```
                string vertex = words[i];
                string[] vnt = vertex.Split('/');

                Int32 v_id = Int32.Parse(vnt[0]) - 1;
                Int32 t_id = Int32.Parse(vnt[1]) - 1;
                Int32 n_id = Int32.Parse(vnt[2]) - 1;

                if (i <= 3)
                {
                   if (i == 1)
                   {
                      first = new Vertex(v_id, t_id, n_id);
                      materials[(int)material_id].indices.Add(first);
                   }
                   else
                   {
                      materials[(int)material_id].indices.Add(new Vertex(v_id, t_id, n_id));
                   }
                }
                else
                {
                   int id_last = materials[(int)material_id].indices.Count - 1;
                   Vertex second = materials[(int)material_id].indices[id_last];
                   // 1
                   materials[(int)material_id].indices.Add(first);
                   // 2
                   materials[(int)material_id].indices.Add(second);
                   // 3
                   materials[(int)material_id].indices.Add(new Vertex(v_id, t_id, n_id));
                }
             }
             break;
          }
       }
    }

    return true;
}

public void Release()
```

```
        {
            // vertices
            vertices.Clear();
            vertices = null;

            // text coords
            text_coords.Clear();
            text_coords = null;

            normals.Clear();
            normals = null;

            // material to id
            material_to_id.Clear();
            material_to_id = null;
            // material to texture
            material_to_texture.Clear();
            material_to_texture = null;

            // materials
            foreach (Material material in materials) {
                material.Release();
            }
            materials.Clear();
            materials = null;
        }
    }
}
```

17.5 SMDLoader.cs

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Text;

using OpenTK;

namespace Project
{
   class SMDLoader
   {
      // egy csomópont
      public class Node
      {
                  public Node(string name, int parent_id)
         {
            this.parent_id = parent_id;
            this.name = name;
         }

         public string name;
         public int parent_id;
      };
      // csomópontok
      public List<Node> nodes;

      // egy csont
      public class Bone
      {
                  public Bone(Vector3 translate, Vector3 rotate)
         {
            this.translate = translate;
            this.rotate = rotate;
         }

         public Vector3 translate;
         public Vector3 rotate;
```

```
};

// egy csontváz
public class Skeleton
{
   public Skeleton()
   {
      bones = new List<Bone>();
   }

                  public List<Bone> bones;
};

// egy animáció
public class Animation
{
                  public string name;
   public float fps; // 1 sec alatt, hány skeleton játszódik le
   public List<Skeleton> times;

   public Animation()
   {
      times = new List<Skeleton>();
      name = "";
      fps = 0;
   }
};
// animációk
List<Animation> animations;

public Skeleton reference_skeleton;
Skeleton current_skeleton;

private Dictionary<string, int> material_to_id;

public SMDLoader()
{
   nodes = new List<Node>();
   animations = new List<Animation>();
   material_to_id = new Dictionary<string, int>();
```

```
        reference_skeleton = new Skeleton();
        current_skeleton = new Skeleton();
    }

    float ToFloat(string text)
    {
        text = text.Replace(',', '.');
        float value;
        value = float.Parse(text, NumberStyles.Any, CultureInfo.InvariantCulture);
        return value;
    }

    public bool LoadReference(string directory, string filename, Mesh mesh)
    {
        string[] lines = File.ReadAllText(directory + @"/" + filename).Split(new char[] { '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries);

        int mat_id = -1;

        bool is_nodes = false;
        bool is_skeleton_header = false;
        bool is_skeleton = false;
        bool is_triangles_texturename = false;
        bool is_triangles_v = false;
        int is_triangles_vrepeat = 0;

        foreach (string line in lines)
        {
            string[] words = line.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

            if (words[0] == "nodes") { is_nodes = true; continue; }
            if (words[0] == "skeleton") { is_skeleton_header = true; continue; }
            if (words[0] == "triangles") { is_triangles_texturename = true; continue; }

            if (is_nodes)
            {
                if (words[0] == "end") { is_nodes = false; continue; }

                string node_name = "";
                int i = 1;
```

```
            bool words_ok = false;
            while (!words_ok)
            {
                node_name += ((i == 1) ? "" : " ") + words[i];
                if (words[i][words[i].Length - 1] == "")
                {
                    words_ok = true;
                }
                i++;
            }

            nodes.Add(new Node(node_name, int.Parse(words[i])));
        }

        if (is_skeleton_header)
        {
            is_skeleton_header = false;
            is_skeleton = true;
            continue;
        }

        if (is_skeleton)
        {
            if (words[0] == "end") { is_skeleton = false; continue; }

            reference_skeleton.bones.Add(new Bone(new Vector3(ToFloat(words[1]), ToFloat(words[2]),
ToFloat(words[3])), new Vector3(ToFloat(words[4]), ToFloat(words[5]), ToFloat(words[6]))));
        }

        if (is_triangles_texturename)
        {
            if (words[0] == "end") { is_triangles_texturename = false; continue; }

            string texture_name = line;// String.Join(" ", words);

            if (!material_to_id.ContainsKey(texture_name))
            {
                mat_id = material_to_id.Count;
                material_to_id.Add(texture_name, mat_id);
                mesh.materials.Add(new Mesh.Material(texture_name));
```

```
    }
    else
    {
        mat_id = material_to_id[texture_name];
    }

    is_triangles_texturename = false;
    is_triangles_v = true;
    is_triangles_vrepeat = 0;
    continue;
}

if (is_triangles_v)
{
    if (words.Count() == 9) // HL1
    {
        mesh.is_obj = false;
        mesh.is_hl1 = true;
        mesh.is_hl2 = false;

        // matrix
        int matrix_id = int.Parse(words[0]);
        float weight = 1.0f;
        // vertex
        Vector3 v = new Vector3(ToFloat(words[1]), ToFloat(words[2]), ToFloat(words[3]));
        // normal
        Vector3 n = new Vector3(ToFloat(words[4]), ToFloat(words[5]), ToFloat(words[6]));
        // textcoords
        Vector2 t = new Vector2(ToFloat(words[7]), ToFloat(words[8]));

        Mesh.Vertex vertex = new Mesh.Vertex(v, n, t);

        // one matrix
        vertex.AddMatrix(matrix_id, weight);

        // add
        mesh.materials[mat_id].vertices.Add(vertex);
    }
    else // HL2
    {
```

```
            mesh.is_obj = false;
            mesh.is_hl1 = false;
            mesh.is_hl2 = true;

            // vertex
            Vector3 v = new Vector3(ToFloat(words[1]), ToFloat(words[2]), ToFloat(words[3]));
            // normal
            Vector3 n = new Vector3(ToFloat(words[4]), ToFloat(words[5]), ToFloat(words[6]));
            // textcoords
            Vector2 t = new Vector2(ToFloat(words[7]), ToFloat(words[8]));

            Mesh.Vertex vertex = new Mesh.Vertex(v, n, t);

            // many matrix
            int num = int.Parse(words[9]);
            int id = 10;
            for (int i = 0; i < num; i++)
            {
                // matrix
                int matrix_id = int.Parse(words[id++]);
                float weight = ToFloat(words[id++]);

                // add
                vertex.AddMatrix(matrix_id, weight);
            }

            // add
            mesh.materials[mat_id].vertices.Add(vertex);
        }

        is_triangles_vrepeat++;
        if (is_triangles_vrepeat == 3)
        {
            is_triangles_v = false;
            is_triangles_texturename = true;
            continue;
        }

        continue;
    }
```

```
        }

        // min-max
        mesh.min = new Vector3(+1000000.0f, +1000000.0f, +1000000.0f);
        mesh.max = new Vector3(-1000000.0f, -1000000.0f, -1000000.0f);
        foreach (Mesh.Material material in mesh.materials)
        {
           foreach (Mesh.Vertex vertex in material.vertices)
           {
              Vector3 v = vertex.vertex;

              // min
              if (v.X < mesh.min.X) { mesh.min.X = v.X; }
              if (v.Y < mesh.min.Y) { mesh.min.Y = v.Y; }
              if (v.Z < mesh.min.Z) { mesh.min.Z = v.Z; }
              // max
              if (mesh.max.X < v.X) { mesh.max.X = v.X; }
              if (mesh.max.Y < v.Y) { mesh.max.Y = v.Y; }
              if (mesh.max.Z < v.Z) { mesh.max.Z = v.Z; }
           }
        }

        // init current skeleton
        for (int i = 0; i < reference_skeleton.bones.Count(); i++)
        {
           current_skeleton.bones.Add(new Bone(new Vector3(0, 0, 0), new Vector3(0, 0, 0)));
        }

        // calc matrices
        mesh.transforms = new List<Matrix4>();
        for (int i = 0; i < reference_skeleton.bones.Count(); i++) { mesh.transforms.Add(new Matrix4()); }

        mesh.inverse_transforms_reference = new List<Matrix4>();
        for (int i = 0; i < reference_skeleton.bones.Count(); i++) { mesh.inverse_transforms_reference.Add(new
Matrix4()); }

        for (int i = 0; i < reference_skeleton.bones.Count(); i++)
        {
           Matrix4 invert = GetReferenceMatrix(i);
           mesh.inverse_transforms_reference[i] = invert;
```

```
        }

        for (int i = 0; i < mesh.materials.Count; i++)
        {
            Mesh.Material material = mesh.materials[i];

            for (int j = 0; j < material.vertices.Count; j++)
            {
                Mesh.Vertex vertex = material.vertices[j];

                Matrix4 inverseTransform = mesh.inverse_transforms_reference[vertex.matrices[0].matrix_id] *
vertex.matrices[0].weight;
                for (int k = 1; k < vertex.matrices.Count; k++)
                {
                    inverseTransform += mesh.inverse_transforms_reference[vertex.matrices[i].matrix_id] *
vertex.matrices[i].weight;
                }

                inverseTransform.Invert();

                // vertex
                vertex.vertex = new Vector3(new Vector4(vertex.vertex, 1.0f) * inverseTransform);

                // normal
                vertex.normal = new Vector3(new Vector4(vertex.normal, 0.0f) * inverseTransform);
            }
        }

        return true;
    }

    public bool AddAnimation(string directory, string filename, string anim_name, float fps)
    {
        string[] lines = File.ReadAllText(directory + @"/" + filename).Split(new char[] { '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries);

        Animation animation = new Animation();
        animation.name = anim_name;
        animation.fps = fps;
```

```
Skeleton skeleton = null;

bool is_time_or_end = false;
bool is_time = false;

foreach (string line in lines)
{
   string[] words = line.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

   if (words[0] == "skeleton")
   {
      is_time_or_end = true;
      is_time = false;
      continue;
   }

   if (is_time)
   {
      if (words[0] == "time" || words[0] == "end")
      {
         is_time = false;
         is_time_or_end = true;
      }
   }

   if (is_time_or_end) {
      if (skeleton != null)
      {
         animation.times.Add(skeleton);
      }

      if (words[0] == "time")
      {
         skeleton = new Skeleton();

         is_time_or_end = false;
         is_time = true;

         continue;
      }
```

```
            if (words[0] == "end")
            {
                is_time_or_end = false;
                animations.Add(animation);
                break;
            }
        }

        if (is_time)
        {
            skeleton.bones.Add(new Bone(new Vector3(ToFloat(words[1]), ToFloat(words[2]),
ToFloat(words[3])), new Vector3(ToFloat(words[4]), ToFloat(words[5]), ToFloat(words[6]))));
        }
    }

    return true;
}

Animation curr_animation;
public void SetAnimation(string anim_name)
{
    curr_animation = null;

    foreach (Animation animation in animations)
    {
        if (animation.name == anim_name)
        {
            curr_animation = animation;
            return;
        }
    }
}

public void SetFPS(int fps)
{
    curr_animation.fps = fps;
}

public bool IsAnimationSelected()
```

```
{
    if (curr_animation == null) { return false; }
    return true;
}

public float GetFullTime()
{
    return ((float)(curr_animation.times.Count() - 1) / curr_animation.fps);
}

public void SetTime(float time, Mesh mesh)
{
    // Set Skeleton
    int start = (int)Math.Floor((double)(time * curr_animation.fps));
    int end = (int)Math.Ceiling((double)(time * curr_animation.fps));

    if (start == end)
    {
        CalcNewSkeleton(curr_animation.times[start], curr_animation.times[end], 0.0f);
    }
    else
    {
        double skeletonT1 = (double)start / curr_animation.fps;
        double skeletonT2 = (double)end / curr_animation.fps;
        double diff1 = skeletonT2 - skeletonT1;
        double diff2 = time - skeletonT1;
        double dt = diff2 / diff1;
        CalcNewSkeleton(curr_animation.times[start], curr_animation.times[end], (float)dt);
    }

    // Update Matrices
    UpdateMatrices(mesh);
}

private float GetSignedRad(float alfa, float beta)
{
    float difference = beta - alfa;
    while (difference < -(float)Math.PI) difference += 2.0f * (float)Math.PI;
    while (difference > (float)Math.PI) difference -= 2.0f * (float)Math.PI;
    return difference;
```

```
    }

    void CalcNewSkeleton(Skeleton start, Skeleton end, float dt)
    {
       for (int i = 0; i < (int)current_skeleton.bones.Count(); i++)
       {
          // translate
          current_skeleton.bones[i].translate = (start.bones[i].translate * (1.0f - dt)) + (end.bones[i].translate *
dt);
          // rotate
          // X
          float rotate_x = GetSignedRad(start.bones[i].rotate.X, end.bones[i].rotate.X);
          current_skeleton.bones[i].rotate.X = start.bones[i].rotate.X + (rotate_x * dt);
          // Y
          float rotate_y = GetSignedRad(start.bones[i].rotate.Y, end.bones[i].rotate.Y);
          current_skeleton.bones[i].rotate.Y = start.bones[i].rotate.Y + (rotate_y * dt);
          // Z
          float rotate_z = GetSignedRad(start.bones[i].rotate.Z, end.bones[i].rotate.Z);
          current_skeleton.bones[i].rotate.Z = start.bones[i].rotate.Z + (rotate_z * dt);
       }
    }

    void UpdateMatrices(Mesh mesh)
    {
       // mátrixok kiszámítása rekurzívan
       for (int i = 0; i < current_skeleton.bones.Count(); i++)
       {
          mesh.transforms[i] = GetMatrix(i);
       }
    }

    public Matrix4 GetReferenceMatrix(int id)
    {
       // ha a gyökérnél vagyunk, akkor egységmátrix
       if (id == -1) return Matrix4.Identity;

       // változások a szülõhöz képest
       Vector3 tr = reference_skeleton.bones[id].translate; // eltolás
       Vector3 rot = reference_skeleton.bones[id].rotate; // forgatás
       // lokális és a szülõ mátrixok egybe gyúrása
```

```
        Matrix4 local = Matrix4.CreateRotationX(rot.X) * Matrix4.CreateRotationY(rot.Y) *
Matrix4.CreateRotationZ(rot.Z) * Matrix4.CreateTranslation(tr);
        Matrix4 parent_global = GetReferenceMatrix(nodes[id].parent_id);

        return (local * parent_global);
      }

      public Matrix4 GetMatrix(int id)
      {
        // ha a gyökérnél vagyunk, akkor egységmátrix
        if (id == -1) return Matrix4.Identity;

        // változások a szülõhöz képest
        Vector3 tr = current_skeleton.bones[id].translate; // eltolás
        Vector3 rot = current_skeleton.bones[id].rotate; // forgatás
        // lokális és a szülõ mátrixok egybe gyúrása
        Matrix4 local = Matrix4.CreateRotationX(rot.X) * Matrix4.CreateRotationY(rot.Y) *
Matrix4.CreateRotationZ(rot.Z) * Matrix4.CreateTranslation(tr);
        Matrix4 parent_global = GetMatrix(nodes[id].parent_id);

        return (local * parent_global);
      }

      public void Release()
      {
        // nodes
        nodes.Clear();

        ;
      }


    }
}
```

17.6 OpenCLRenderer.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Media.Imaging;
using System.Runtime.InteropServices;

using OpenTK;
using Cloo;
using System.Windows;
using System.Windows.Media;

namespace OpenCLRenderer
{
   struct Float3
   {
      public float m_X;
      public float m_Y;
      public float m_Z;
   }

   struct Vertex
   {
      public float m_Vx;
      public float m_Vy;
      public float m_Vz;
      public float m_Nx;
      public float m_Ny;
      public float m_Nz;
      public float m_TCx;
      public float m_TCy;
      public int m_iNumMatrices;
      public int m_iMatrixId1;
      public int m_iMatrixId2;
```

```
    public int m_iMatrixId3;
    public float m_fWeight1;
    public float m_fWeight2;
    public float m_fWeight3;
}

struct Triangle
{
    public Vertex m_A;
    public Vertex m_B;
    public Vertex m_C;
    public int m_iMaterialId;
    public float m_fNormalX;
    public float m_fNormalY;
    public float m_fNormalZ;
    public float m_fArea;
}

struct BBox
{
    public float minx;
    public float miny;
    public float minz;
    public float maxx;
    public float maxy;
    public float maxz;
}

struct BVHNode
{
    public int m_iId;
    public Triangle m_Triangle;
    public BBox m_BBox;
    public int m_iLeft;
    public int m_iRight;
}

struct Matrix
{
    public float m11;
```

```
    public float m12;
    public float m13;
    public float m14;

    public float m21;
    public float m22;
    public float m23;
    public float m24;

    public float m31;
    public float m32;
    public float m33;
    public float m34;

    public float m41;
    public float m42;
    public float m43;
    public float m44;
}

struct Texture
{
    public ulong m_iOffsetTextreDatas;
    public int m_iWidth;
    public int m_iHeight;
}


struct Material
{
    public Texture m_DiffuseTexture;
    public Texture m_SpecularTexture;
    public Texture m_NormalTexture;
}

struct BVHObject
{
    public int m_iType;
    public List<BVHNode> m_listBVHNodes;
    public List< List<BVHNode> > m_LevelXBVHs;
```

```
}

struct Ray
{
   public float posx;
   public float posy;
   public float posz;
   public float dirx;
   public float diry;
   public float dirz;
   public float length;
}

public static class Win32
{
   [DllImport("kernel32.dll", EntryPoint = "RtlMoveMemory")]
   public static extern void CopyMemory(IntPtr dest, IntPtr source, int Length);
}

class Scene : IDisposable
{

   public Scene()
   {
      // ez azert, hogy ne adjon warning-ot a fordito, mert 1x sem hasznaljuk a Ray strukturat
      Ray ray = new Ray();
      ray.posx = 0;
      ray.posy = 0;
      ray.posz = 0;
      ray.dirx = 0;
      ray.diry = 0;
      ray.dirz = 0;
      ray.length = 0;
   }

   public List<string> GetDevices()
   {
      List<string> ret = new List<string>();

      ComputePlatform[] platforms = ComputePlatform.Platforms.ToArray();
```

```
        foreach (ComputePlatform platform in platforms)
        {
            ComputeContextPropertyList properties = new ComputeContextPropertyList(platform);

            ComputeContext newContext = null;
            try
            {
                newContext = new ComputeContext(ComputeDeviceTypes.Gpu, properties, null, IntPtr.Zero);
            }
            catch (Exception ex)
            {
                ex.ToString();
                continue;
            }

            ComputeDevice[] devices = newContext.Devices.ToArray();

            foreach (ComputeDevice device in devices)
            {
                m_Context = newContext;
                m_Device = device;

                cmdQueue = new ComputeCommandQueue(m_Context, m_Device,
ComputeCommandQueueFlags.None);
                m_Program = new ComputeProgram(m_Context, OpenCLScript.GetText());

                ret.Add(m_Device.Name);
                //try
                //{
                //    m_Program.Build(null, null, null, IntPtr.Zero);
                //
                //    ret.Add(m_Device.Name);
                //}
                //catch (Exception e)
                //{
                //    e.ToString();
                //    string strText = m_Program.GetBuildLog(m_Device);
                //    MessageBox.Show(strText, "Exception");
                //    continue;
```

```
            //}
            //finally
            {
                m_Program.Dispose();
                m_Program = null;
                cmdQueue.Dispose();
                cmdQueue = null;
            }
        }

        newContext.Dispose();
        newContext = null;

        //return ret;
    }

    return ret;
}

public void CreateDevice(string strDeviceName, string @strVertexShader, string @strRayShader)
{
    ComputePlatform[] platforms = ComputePlatform.Platforms.ToArray();

    foreach (ComputePlatform platform in platforms)
    {
        ComputeContextPropertyList properties = new ComputeContextPropertyList(platform);

        ComputeContext newContext = null;
        try
        {
            newContext = new ComputeContext(ComputeDeviceTypes.Gpu, properties, null, IntPtr.Zero);
        }
        catch (Exception ex)
        {
            ex.ToString();
            continue;
        }

        ComputeDevice[] devices = newContext.Devices.ToArray();
```

```
        foreach (ComputeDevice device in devices)
        {
            m_Context = newContext;
            m_Device = device;

            if (m_Device.Name != strDeviceName)
            {
                continue;
            }

            OpenCLScript.SetVertexShader(@strVertexShader);
            OpenCLScript.SetRayShader(@strRayShader);

            cmdQueue = new ComputeCommandQueue(m_Context, m_Device,
ComputeCommandQueueFlags.None);
            m_Program = new ComputeProgram(m_Context, OpenCLScript.GetText());

            try
            {
                m_Program.Build(null, "",  null, IntPtr.Zero);
            }
            catch (Exception e)
            {
                e.ToString();

                string strText = m_Program.GetBuildLog(m_Device);
                MessageBox.Show(strText, "Exception");

                Application.Current.Shutdown();
                return;
            }

            // VertexShader
            kernelTriangleShader = m_Program.CreateKernel("Main_TriangleShader");

            // RefitTree
            KernelRefitTree_LevelX = m_Program.CreateKernel("Main_RefitTree_LevelX");

            // CameraRays
            KernelCameraRays = m_Program.CreateKernel("Main_CameraRays");
```

```
            // RayShader
            KernelRayShader = m_Program.CreateKernel("Main_RayShader");

            // Resize
            Resize(8, 8);

            // ok
            return;
        }

        newContext.Dispose();
    }
}

public int NumMatrices()
{
    //m_mtxMutex.WaitOne();
    int iRet = m_listMatrices.Count();
    //m_mtxMutex.ReleaseMutex();

    return iRet;
}

// Matrix
public int GenMatrix()
{
    //m_mtxMutex.WaitOne();
    int iId = m_listMatrices.Count;

    Matrix newMatrix = new Matrix();

    m_listMatrices.Add(newMatrix);
    //m_mtxMutex.ReleaseMutex();
    return iId;
}

public void SetMatrix(int iId, Matrix4 mMatrix)
{
    //m_mtxMutex.WaitOne();
```

```
        Matrix newMatrix = new Matrix();

        Matrix4 transposedMatrix = new Matrix4(mMatrix.Row0, mMatrix.Row1, mMatrix.Row2,
mMatrix.Row3);
        transposedMatrix.Transpose();

        newMatrix.m11 = transposedMatrix.M11;
        newMatrix.m12 = transposedMatrix.M12;
        newMatrix.m13 = transposedMatrix.M13;
        newMatrix.m14 = transposedMatrix.M14;

        newMatrix.m21 = transposedMatrix.M21;
        newMatrix.m22 = transposedMatrix.M22;
        newMatrix.m23 = transposedMatrix.M23;
        newMatrix.m24 = transposedMatrix.M24;

        newMatrix.m31 = transposedMatrix.M31;
        newMatrix.m32 = transposedMatrix.M32;
        newMatrix.m33 = transposedMatrix.M33;
        newMatrix.m34 = transposedMatrix.M34;

        newMatrix.m41 = transposedMatrix.M41;
        newMatrix.m42 = transposedMatrix.M42;
        newMatrix.m43 = transposedMatrix.M43;
        newMatrix.m44 = transposedMatrix.M44;

        m_listMatrices[iId] = newMatrix;
        //m_mtxMutex.ReleaseMutex();
    }

    // Material
    public int GenMaterial()
    {
        //m_mtxMutex.WaitOne();
        int iId = m_listMaterials.Count;

        Material newMaterial = new Material();

        m_listMaterials.Add(newMaterial);
        //m_mtxMutex.ReleaseMutex();
```

```
        return iId;
    }

    public void SetMaterial(int iId, string @strDiffuseFileName, string @strSpecularFileName, string
@strNormalFileName)
    {
        //m_mtxMutex.WaitOne();
        Material newMaterial = new Material();

        newMaterial.m_DiffuseTexture  = CreateTextureFromFile(@strDiffuseFileName);
        newMaterial.m_NormalTexture = CreateTextureFromFile(@strSpecularFileName);
        newMaterial.m_SpecularTexture = CreateTextureFromFile(@strNormalFileName);

        m_listMaterials[iId] = newMaterial;
        //m_mtxMutex.ReleaseMutex();
    }

    Texture CreateTextureFromFile(string @strFileName)
    {
        Bitmap bitmap = new Bitmap(@strFileName);
        bitmap.RotateFlip(RotateFlipType.RotateNoneFlipY);
        Rectangle rectangle = new Rectangle(0, 0, bitmap.Width, bitmap.Height);
        BitmapData bitmapData = bitmap.LockBits(rectangle, ImageLockMode.ReadOnly,
System.Drawing.Imaging.PixelFormat.Format32bppArgb);

        Texture newTexture = new Texture();
        newTexture.m_iWidth = bitmap.Width;
        newTexture.m_iHeight = bitmap.Height;
        newTexture.m_iOffsetTextreDatas = (uint)m_listTexturesData.Count;
        // copy data
        int iSize = newTexture.m_iWidth * newTexture.m_iHeight * 4;
        byte[] datas = new byte[iSize];
        Marshal.Copy(bitmapData.Scan0, datas, 0, iSize);
        m_listTexturesData.AddRange(datas);

        datas = null;

        bitmap.UnlockBits(bitmapData);

        bitmap.Dispose();
```

```
            bitmap = null;

            return newTexture;
        }

        static float GetDistance_Triangle_Triangle(Triangle tri1, Triangle tri2)
        {
            float fMinDistance = float.MaxValue;

            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_A.m_Vx,
tri1.m_A.m_Vy, tri1.m_A.m_Vz), new Vector3(tri2.m_A.m_Vx, tri2.m_A.m_Vy, tri2.m_A.m_Vz)));
            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_A.m_Vx,
tri1.m_A.m_Vy, tri1.m_A.m_Vz), new Vector3(tri2.m_B.m_Vx, tri2.m_B.m_Vy, tri2.m_B.m_Vz)));
            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_A.m_Vx,
tri1.m_A.m_Vy, tri1.m_A.m_Vz), new Vector3(tri2.m_C.m_Vx, tri2.m_C.m_Vy, tri2.m_C.m_Vz)));

            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_B.m_Vx,
tri1.m_B.m_Vy, tri1.m_B.m_Vz), new Vector3(tri2.m_A.m_Vx, tri2.m_A.m_Vy, tri2.m_A.m_Vz)));
            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_B.m_Vx,
tri1.m_B.m_Vy, tri1.m_B.m_Vz), new Vector3(tri2.m_B.m_Vx, tri2.m_B.m_Vy, tri2.m_B.m_Vz)));
            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_B.m_Vx,
tri1.m_B.m_Vy, tri1.m_B.m_Vz), new Vector3(tri2.m_C.m_Vx, tri2.m_C.m_Vy, tri2.m_C.m_Vz)));

            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_C.m_Vx,
tri1.m_C.m_Vy, tri1.m_C.m_Vz), new Vector3(tri2.m_A.m_Vx, tri2.m_A.m_Vy, tri2.m_A.m_Vz)));
            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_C.m_Vx,
tri1.m_C.m_Vy, tri1.m_C.m_Vz), new Vector3(tri2.m_B.m_Vx, tri2.m_B.m_Vy, tri2.m_B.m_Vz)));
            fMinDistance = Math.Min(fMinDistance, Vector3.Distance(new Vector3(tri1.m_C.m_Vx,
tri1.m_C.m_Vy, tri1.m_C.m_Vz), new Vector3(tri2.m_C.m_Vx, tri2.m_C.m_Vy, tri2.m_C.m_Vz)));

            return fMinDistance;
        }

        static float GetDistance_BBox_BBox(BBox bbox1, BBox bbox2)
        {
            Vector3 center1 = (new Vector3(bbox1.minx, bbox1.miny, bbox1.minz) + new Vector3(bbox1.maxx,
bbox1.maxy, bbox1.maxz)) / 2.0f;
            Vector3 center2 = (new Vector3(bbox2.minx, bbox2.miny, bbox2.minz) + new Vector3(bbox2.maxx,
bbox2.maxy, bbox2.maxz)) / 2.0f;
```

```
    Vector3 halfSize1 = new Vector3(bbox1.maxx, bbox1.maxy, bbox1.maxz) - center1;
    Vector3 halfSize2 = new Vector3(bbox2.maxx, bbox2.maxy, bbox2.maxz) - center2;

    float x = Math.Abs(center2.X - center1.X) - halfSize1.X - halfSize2.X;
    float y = Math.Abs(center2.Y - center1.Y) - halfSize1.Y - halfSize2.Y;
    float z = Math.Abs(center2.Z - center1.Z) - halfSize1.Z - halfSize2.Z;

    float fLength = (float)Math.Sqrt(x * x + y * y + z * z);

    // ha osszeernek, akor negativ tavolsag
    if (x < 0.0f && y < 0.0f && z < 0.0f) { fLength = -fLength; }

    return fLength;
}

static BBox GenBBox(Triangle tri1, Triangle tri2)
{
    float fMinX = float.MaxValue;
    float fMinY = float.MaxValue;
    float fMinZ = float.MaxValue;

    fMinX = Math.Min(fMinX, tri1.m_A.m_Vx);
    fMinX = Math.Min(fMinX, tri1.m_B.m_Vx);
    fMinX = Math.Min(fMinX, tri1.m_C.m_Vx);
    fMinX = Math.Min(fMinX, tri2.m_A.m_Vx);
    fMinX = Math.Min(fMinX, tri2.m_B.m_Vx);
    fMinX = Math.Min(fMinX, tri2.m_C.m_Vx);

    fMinY = Math.Min(fMinY, tri1.m_A.m_Vy);
    fMinY = Math.Min(fMinY, tri1.m_B.m_Vy);
    fMinY = Math.Min(fMinY, tri1.m_C.m_Vy);
    fMinY = Math.Min(fMinY, tri2.m_A.m_Vy);
    fMinY = Math.Min(fMinY, tri2.m_B.m_Vy);
    fMinY = Math.Min(fMinY, tri2.m_C.m_Vy);

    fMinZ = Math.Min(fMinZ, tri1.m_A.m_Vz);
    fMinZ = Math.Min(fMinZ, tri1.m_B.m_Vz);
    fMinZ = Math.Min(fMinZ, tri1.m_C.m_Vz);
    fMinZ = Math.Min(fMinZ, tri2.m_A.m_Vz);
    fMinZ = Math.Min(fMinZ, tri2.m_B.m_Vz);
```

```
    fMinZ = Math.Min(fMinZ, tri2.m_C.m_Vz);

    float fMaxX = float.MinValue;
    float fMaxY = float.MinValue;
    float fMaxZ = float.MinValue;

    fMaxX = Math.Max(fMaxX, tri1.m_A.m_Vx);
    fMaxX = Math.Max(fMaxX, tri1.m_B.m_Vx);
    fMaxX = Math.Max(fMaxX, tri1.m_C.m_Vx);
    fMaxX = Math.Max(fMaxX, tri2.m_A.m_Vx);
    fMaxX = Math.Max(fMaxX, tri2.m_B.m_Vx);
    fMaxX = Math.Max(fMaxX, tri2.m_C.m_Vx);

    fMaxY = Math.Max(fMaxY, tri1.m_A.m_Vy);
    fMaxY = Math.Max(fMaxY, tri1.m_B.m_Vy);
    fMaxY = Math.Max(fMaxY, tri1.m_C.m_Vy);
    fMaxY = Math.Max(fMaxY, tri2.m_A.m_Vy);
    fMaxY = Math.Max(fMaxY, tri2.m_B.m_Vy);
    fMaxY = Math.Max(fMaxY, tri2.m_C.m_Vy);

    fMaxZ = Math.Max(fMaxZ, tri1.m_A.m_Vz);
    fMaxZ = Math.Max(fMaxZ, tri1.m_B.m_Vz);
    fMaxZ = Math.Max(fMaxZ, tri1.m_C.m_Vz);
    fMaxZ = Math.Max(fMaxZ, tri2.m_A.m_Vz);
    fMaxZ = Math.Max(fMaxZ, tri2.m_B.m_Vz);
    fMaxZ = Math.Max(fMaxZ, tri2.m_C.m_Vz);

    BBox bbox = new BBox();
    bbox.minx = fMinX;
    bbox.miny = fMinY;
    bbox.minz = fMinZ;
    bbox.maxx = fMaxX;
    bbox.maxy = fMaxY;
    bbox.maxz = fMaxZ;

    return bbox;
}

static BBox GenBBox(BBox bbox1, BBox bbox2)
{
```

```
float fMinX = float.MaxValue;
float fMinY = float.MaxValue;
float fMinZ = float.MaxValue;

fMinX = Math.Min(fMinX, bbox1.minx);
fMinX = Math.Min(fMinX, bbox1.maxx);
fMinX = Math.Min(fMinX, bbox2.minx);
fMinX = Math.Min(fMinX, bbox2.maxx);

fMinY = Math.Min(fMinY, bbox1.miny);
fMinY = Math.Min(fMinY, bbox1.maxy);
fMinY = Math.Min(fMinY, bbox2.miny);
fMinY = Math.Min(fMinY, bbox2.maxy);

fMinZ = Math.Min(fMinZ, bbox1.minz);
fMinZ = Math.Min(fMinZ, bbox1.maxz);
fMinZ = Math.Min(fMinZ, bbox2.minz);
fMinZ = Math.Min(fMinZ, bbox2.maxz);

float fMaxX = float.MinValue;
float fMaxY = float.MinValue;
float fMaxZ = float.MinValue;

fMaxX = Math.Max(fMaxX, bbox1.minx);
fMaxX = Math.Max(fMaxX, bbox1.maxx);
fMaxX = Math.Max(fMaxX, bbox2.minx);
fMaxX = Math.Max(fMaxX, bbox2.maxx);

fMaxY = Math.Max(fMaxY, bbox1.miny);
fMaxY = Math.Max(fMaxY, bbox1.maxy);
fMaxY = Math.Max(fMaxY, bbox2.miny);
fMaxY = Math.Max(fMaxY, bbox2.maxy);

fMaxZ = Math.Max(fMaxZ, bbox1.minz);
fMaxZ = Math.Max(fMaxZ, bbox1.maxz);
fMaxZ = Math.Max(fMaxZ, bbox2.minz);
fMaxZ = Math.Max(fMaxZ, bbox2.maxz);

BBox bbox = new BBox();
bbox.minx = fMinX;
```

```
            bbox.miny = fMinY;
            bbox.minz = fMinZ;
            bbox.maxx = fMaxX;
            bbox.maxy = fMaxY;
            bbox.maxz = fMaxZ;

            return bbox;
        }

        public int GenObject()
        {
            //m_mtxMutex.WaitOne();

            int iId = m_listObjects.Count;
            BVHObject newObject = new BVHObject();
            m_listObjects.Add(newObject);

            //m_mtxMutex.ReleaseMutex();

            return iId;
        }

        public void SetObject(int iId, BVHObject bvhObject)
        {
            //m_mtxMutex.WaitOne();
            m_listObjects[iId] = bvhObject;
            //m_mtxMutex.ReleaseMutex();
        }

        public BVHObject CreateStaticObject(List<Triangle> triangles, Matrix4 matTransform)
        {
            Matrix4 transposedMatrix = new Matrix4(matTransform.Row0, matTransform.Row1,
matTransform.Row2, matTransform.Row3);
            transposedMatrix.Transpose();

            List<Triangle> newTriangles = new List<Triangle>();

            foreach (Triangle oldTri in triangles)
            {
                Triangle newTri = new Triangle();
```

```
            Vertex vertexA = new Vertex();
            Vector3 AV = new Vector3(transposedMatrix * new Vector4(oldTri.m_A.m_Vx, oldTri.m_A.m_Vy,
oldTri.m_A.m_Vz, 1.0f));
            vertexA.m_Vx = AV.X;
            vertexA.m_Vy = AV.Y;
            vertexA.m_Vz = AV.Z;
            Vector3 AN = new Vector3(transposedMatrix * new Vector4(oldTri.m_A.m_Nx, oldTri.m_A.m_Ny,
oldTri.m_A.m_Nz, 0.0f));
            vertexA.m_Nx = AN.X;
            vertexA.m_Ny = AN.Y;
            vertexA.m_Nz = AN.Z;
            vertexA.m_TCx = oldTri.m_A.m_TCx;
            vertexA.m_TCy = oldTri.m_A.m_TCy;
            vertexA.m_iNumMatrices = 0;
            vertexA.m_iMatrixId1 = -1;
            vertexA.m_fWeight1 = 0.0f;
            vertexA.m_iMatrixId2 = -1;
            vertexA.m_fWeight2 = 0.0f;
            vertexA.m_iMatrixId3 = -1;
            vertexA.m_fWeight3 = 0.0f;

            Vertex vertexB = new Vertex();
            Vector3 BV = new Vector3(transposedMatrix * new Vector4(oldTri.m_B.m_Vx, oldTri.m_B.m_Vy,
oldTri.m_B.m_Vz, 1.0f));
            vertexB.m_Vx = BV.X;
            vertexB.m_Vy = BV.Y;
            vertexB.m_Vz = BV.Z;
            Vector3 BN = new Vector3(transposedMatrix * new Vector4(oldTri.m_B.m_Nx, oldTri.m_B.m_Ny,
oldTri.m_B.m_Nz, 0.0f));
            vertexB.m_Nx = BN.X;
            vertexB.m_Ny = BN.Y;
            vertexB.m_Nz = BN.Z;
            vertexB.m_TCx = oldTri.m_B.m_TCx;
            vertexB.m_TCy = oldTri.m_B.m_TCy;
            vertexB.m_iNumMatrices = 0;
            vertexB.m_iMatrixId1 = -1;
            vertexB.m_fWeight1 = 0.0f;
            vertexB.m_iMatrixId2 = -1;
            vertexB.m_fWeight2 = 0.0f;
```

```
            vertexB.m_iMatrixId3 = -1;
            vertexB.m_fWeight3 = 0.0f;

            Vertex vertexC = new Vertex();
            Vector3 CV = new Vector3(transposedMatrix * new Vector4(oldTri.m_C.m_Vx, oldTri.m_C.m_Vy,
oldTri.m_C.m_Vz, 1.0f));
            vertexC.m_Vx = CV.X;
            vertexC.m_Vy = CV.Y;
            vertexC.m_Vz = CV.Z;
            Vector3 CN = new Vector3(transposedMatrix * new Vector4(oldTri.m_C.m_Nx, oldTri.m_C.m_Ny,
oldTri.m_C.m_Nz, 0.0f));
            vertexC.m_Nx = CN.X;
            vertexC.m_Ny = CN.Y;
            vertexC.m_Nz = CN.Z;
            vertexC.m_TCx = oldTri.m_C.m_TCx;
            vertexC.m_TCy = oldTri.m_C.m_TCy;
            vertexC.m_iNumMatrices = 0;
            vertexC.m_iMatrixId1 = -1;
            vertexC.m_fWeight1 = 0.0f;
            vertexC.m_iMatrixId2 = -1;
            vertexC.m_fWeight2 = 0.0f;
            vertexC.m_iMatrixId3 = -1;
            vertexC.m_fWeight3 = 0.0f;

            newTri.m_A = vertexA;
            newTri.m_B = vertexB;
            newTri.m_C = vertexC;
            newTri.m_iMaterialId = oldTri.m_iMaterialId;

            // normal
            Vector3 normal = Vector3.Cross(BV - AV, CV - AV).Normalized();
            newTri.m_fNormalX = normal.X;
            newTri.m_fNormalY = normal.Y;
            newTri.m_fNormalZ = normal.Z;
            newTri.m_fArea = Vector3.Cross(BV - AV, CV - AV).Length / 2.0f;

            newTriangles.Add(newTri);
        }

        List<BVHNode> newBVH = CreateBVH(newTriangles);
```

```
    newTriangles.Clear();

    //m_mtxMutex.WaitOne();

    BVHObject newObject = new BVHObject();
    newObject.m_iType = (int)BVHObjectType.Static;
    newObject.m_listBVHNodes = newBVH;
    newObject.m_LevelXBVHs = null;

    //m_mtxMutex.ReleaseMutex();

    return newObject;
}

public BVHObject CreateDynamicObject(List<Triangle> triangles)
{
    List<BVHNode> newBVH = CreateBVH(triangles);

    List< List<BVHNode> > levelXBVH = CopyBVHToLevelX(newBVH);

    //m_mtxMutex.WaitOne();

    BVHObject newObject = new BVHObject();
    newObject.m_iType = (int)BVHObjectType.Dynamic;
    newObject.m_listBVHNodes = newBVH;
    newObject.m_LevelXBVHs = levelXBVH;

    //m_mtxMutex.ReleaseMutex();

    return newObject;
}

public static List<BVHNode> CreateBVH(List<Triangle> triangles0)
{
    List<BVHNode> tree = new List<BVHNode>();

    // gyoker a 0. indexen. ez majd a legvegen lesz beallitva
    BVHNode root = new BVHNode();
    tree.Add(root);
```

```
List<Triangle> triangles = new List<Triangle>();
triangles.AddRange(triangles0);

List<BVHNode> outBuffer = new List<BVHNode>();

// elso szint: haromszogek tavolsaga
while (triangles.Count > 1)
{
   Triangle tri1 = triangles[0];

   float fMinDistance = float.MaxValue;
   int id = 0;

   for (int i = 1; i < triangles.Count; i++)
   {
      float fCurrentDistance = GetDistance_Triangle_Triangle(tri1, triangles[i]);

      if (fCurrentDistance < fMinDistance)
      {
         fMinDistance = fCurrentDistance;
         id = i;
      }

      if (fMinDistance < 0.00001f) { break; }
   }

   Triangle tri2 = triangles[id];
   triangles.RemoveAt(id);
   triangles.RemoveAt(0);

   BVHNode leaf1 = new BVHNode();
   leaf1.m_Triangle = tri1;
   leaf1.m_iLeft = -1;
   leaf1.m_iRight = -1;
   leaf1.m_BBox = GenBBox(tri1, tri1);
   BVHNode leaf2 = new BVHNode();
   leaf2.m_Triangle = tri2;
   leaf2.m_iLeft = -1;
   leaf2.m_iRight = -1;
   leaf2.m_BBox = GenBBox(tri2, tri2);
```

```
    leaf1.m_iId = tree.Count;
    tree.Add(leaf1);
    leaf2.m_iId = tree.Count;
    tree.Add(leaf2);

    BVHNode parent = new BVHNode();
    parent.m_iLeft = leaf1.m_iId;
    parent.m_iRight = leaf2.m_iId;

    parent.m_BBox = GenBBox(tri1, tri2);
    outBuffer.Add(parent);
}

if (triangles.Count == 1)
{
    Triangle tri1 = triangles[0];

    triangles.RemoveAt(0);

    BVHNode leaf1 = new BVHNode();
    leaf1.m_Triangle = tri1;
    leaf1.m_iLeft = -1;
    leaf1.m_iRight = -1;
    leaf1.m_BBox = GenBBox(tri1, tri1);
    leaf1.m_iId = tree.Count;
    tree.Add(leaf1);

    BVHNode parent = new BVHNode();
    parent.m_iLeft = leaf1.m_iId;
    parent.m_iRight = -1;

    parent.m_BBox = GenBBox(tri1, tri1);
    outBuffer.Add(parent);
}

// tovabbi bboxok epitese, mig 1-et nem kapunk
while (outBuffer.Count > 1)
{
    List<BVHNode> inBuffer = new List<BVHNode>();
```

```
inBuffer.AddRange(outBuffer);

outBuffer.Clear();

while (inBuffer.Count > 1)
{
   BVHNode node1 = inBuffer[0];

   float fMinDistance = float.MaxValue;
   int id = 0;

   for (int i = 1; i < inBuffer.Count; i++)
   {
      float fCurrentDistance = GetDistance_BBox_BBox(node1.m_BBox, inBuffer[i].m_BBox);

      if (fCurrentDistance < fMinDistance)
      {
         fMinDistance = fCurrentDistance;
         id = i;
      }
   }

   BVHNode node2 = inBuffer[id];
   inBuffer.RemoveAt(id);
   inBuffer.RemoveAt(0);

   node1.m_iId = tree.Count;
   tree.Add(node1);
   node2.m_iId = tree.Count;
   tree.Add(node2);

   BVHNode parent = new BVHNode();
   parent.m_iLeft = node1.m_iId;
   parent.m_iRight = node2.m_iId;

   parent.m_BBox = GenBBox(node1.m_BBox, node2.m_BBox);

   outBuffer.Add(parent);
}
```

```
            if (inBuffer.Count == 1)
            {
                BVHNode node1 = inBuffer[0];

                inBuffer.RemoveAt(0);

                node1.m_iId = tree.Count;
                tree.Add(node1);

                BVHNode parent = new BVHNode();
                parent.m_iLeft = node1.m_iId;
                parent.m_iRight = -1;

                parent.m_BBox = GenBBox(node1.m_BBox, node1.m_BBox);

                outBuffer.Add(parent);
            }
        }

        // root frissitese
        BVHNode goodRoot = outBuffer[0];
        outBuffer.RemoveAt(0);

        goodRoot.m_iId = 0;
        tree[0] = goodRoot;

        return tree;
    }

    public static void Preorder(List<List<BVHNode>> levelXBVH, List<BVHNode> list, BVHNode node, int
iLevel)
    {
        // ha haromszog, akkor nem kell
        if (-1 == node.m_iLeft && -1 == node.m_iRight)
        {
            return;
        }

        // node eltarolasa
        levelXBVH[iLevel].Add(node);
```

```
    // egy szinttel lejjebb megyunk
    if (-1 != node.m_iLeft)
    {
        BVHNode leftNode = list[node.m_iLeft];
        Preorder(levelXBVH, list, leftNode, iLevel - 1);
    }

    if (-1 != node.m_iRight)
    {
        BVHNode rightNode = list[node.m_iRight];
        Preorder(levelXBVH, list, rightNode, iLevel - 1);
    }
}

static int GetMaxLevel(List<BVHNode> list, BVHNode node, int iCurrentLevel)
{
    if (-1 == node.m_iLeft && -1 == node.m_iRight)
    {
        return iCurrentLevel;
    }

    int iLeftLevel = 0;
    if (-1 != node.m_iLeft)
    {
        BVHNode leftNode = list[node.m_iLeft];
        iLeftLevel = GetMaxLevel(list, leftNode, iCurrentLevel + 1);
    }

    int iRightLevel = 0;
    if (-1 != node.m_iRight)
    {
        BVHNode rightNode = list[node.m_iRight];
        iRightLevel = GetMaxLevel(list, rightNode, iCurrentLevel + 1);
    }

    if (iLeftLevel < iRightLevel) { return iRightLevel; }
    else { return iLeftLevel; }
}
```

```
static void LevelXList_Resize<T>(List<List<T>> levelX, int iMaxId)
{
   while (levelX.Count < iMaxId)
   {
      levelX.Add(new List<T>());
   }
}

List<List<BVHNode>> CopyBVHToLevelX(List<BVHNode> newBVH)
{
   List< List<BVHNode> > levelXBVH = new List< List<BVHNode> >();

   BVHNode root = newBVH[0];

   int iMaxLevel = GetMaxLevel(newBVH, root, 0);
   LevelXList_Resize(levelXBVH, iMaxLevel);

   Preorder(levelXBVH, newBVH, root, iMaxLevel - 1);

   return levelXBVH;
}

List<BVHNode> ConvertBVHListToGlobal(List<BVHNode> listBVH, int iOffset)
{
   List<BVHNode> ret = new List<BVHNode>();

   foreach (BVHNode node in listBVH)
   {
      BVHNode globalNode = node;

      globalNode.m_iId += iOffset;
      if (-1 != globalNode.m_iLeft) { globalNode.m_iLeft += iOffset; }
      if (-1 != globalNode.m_iRight) { globalNode.m_iRight += iOffset; }

      ret.Add(globalNode);
   }

   return ret;
}
```

```
    public void Commit()
    {
        //m_mtxMutex.WaitOne();

        List<BVHNode> listAllBVHNodes = new List<BVHNode>();
        List<int> listAllBVHNodesType = new List<int>();
        List< List<BVHNode> > listAllLevelXBVHs = new List< List<BVHNode> >();
        List< List<int> > listAllLevelXBVHsOffsets = new List< List<int> >();
        List<int> listBeginObjects = new List<int>();

        foreach (BVHObject bvhObject in m_listObjects)
        {
            int iOffset = listAllBVHNodes.Count;
            listBeginObjects.Add(iOffset);

            List<BVHNode> listGlobalBVHNode = ConvertBVHListToGlobal(bvhObject.m_listBVHNodes,
iOffset);

            // global bvh nodes
            listAllBVHNodes.AddRange(listGlobalBVHNode);

            // global types
            foreach (BVHNode node in bvhObject.m_listBVHNodes) {
listAllBVHNodesType.Add(bvhObject.m_iType); }

            // Global level N
            if (bvhObject.m_iType == (int)BVHObjectType.Dynamic)
            {
                LevelXList_Resize(listAllLevelXBVHs, bvhObject.m_LevelXBVHs.Count);
                for (int i = 0; i < bvhObject.m_LevelXBVHs.Count; i++)
                {
                    List<BVHNode> listGlobalLevelN = ConvertBVHListToGlobal(bvhObject.m_LevelXBVHs[i],
iOffset);
                    listAllLevelXBVHs[i].AddRange(listGlobalLevelN);

                    // size
                    m_listRefitTree_LevelXSizes.Add(listGlobalLevelN.Count);
                }
            }
        }
```

```
        // bufferek letrehozasa, device-onkent
        m_iNumBeginObjects = listBeginObjects.Count();
        if (null != clInput_BeginObjects) { clInput_BeginObjects.Dispose(); clInput_BeginObjects = null; }
        clInput_BeginObjects = new ComputeBuffer<int>(m_Context, ComputeMemoryFlags.ReadOnly |
ComputeMemoryFlags.CopyHostPointer, listBeginObjects.ToArray());

        // texturak betoltese
        if (null != clInput_Materials) { clInput_Materials.Dispose(); clInput_Materials = null; }
        clInput_Materials = new ComputeBuffer<Material>(m_Context, ComputeMemoryFlags.ReadOnly |
ComputeMemoryFlags.CopyHostPointer, m_listMaterials.ToArray());

        if (null != clInput_TexturesData) { clInput_TexturesData.Dispose(); clInput_TexturesData = null; }
        clInput_TexturesData = new ComputeBuffer<byte>(m_Context, ComputeMemoryFlags.ReadOnly |
ComputeMemoryFlags.CopyHostPointer, m_listTexturesData.ToArray());

        // matrixok betoltese
        if (null != clInput_MatricesData) { clInput_MatricesData.Dispose(); clInput_MatricesData = null; }
        clInput_MatricesData = new ComputeBuffer<Matrix>(m_Context, ComputeMemoryFlags.ReadWrite |
ComputeMemoryFlags.CopyHostPointer, m_listMatrices.ToArray());

        // global bvh nodes, count
        m_iNumBVHNodes = listAllBVHNodes.Count;

        // Global level N
        foreach (ComputeBuffer<BVHNode> clInput_RefitTree_LevelX in listCLInput_RefitTree_LevelX)
        {
            clInput_RefitTree_LevelX.Dispose();
        }
        listCLInput_RefitTree_LevelX.Clear();
        foreach (List<BVHNode> listLevelXBVHs in listAllLevelXBVHs)
        {
            // levels
            ComputeBuffer<BVHNode> clInput_RefitTree_LevelX = new
ComputeBuffer<BVHNode>(m_Context, ComputeMemoryFlags.ReadOnly |
ComputeMemoryFlags.CopyHostPointer, listLevelXBVHs.ToArray());
            listCLInput_RefitTree_LevelX.Add(clInput_RefitTree_LevelX);
        }

        // global types
```

```
        if (null != clInput_AllBVHNodesType) { clInput_AllBVHNodesType.Dispose();
clInput_AllBVHNodesType = null; }
        clInput_AllBVHNodesType = new ComputeBuffer<int>(m_Context, ComputeMemoryFlags.ReadOnly |
ComputeMemoryFlags.CopyHostPointer, listAllBVHNodesType.ToArray());

        // start global nodes
        if (null != clInput_AllBVHNodes) { clInput_AllBVHNodes.Dispose(); clInput_AllBVHNodes = null; }
        clInput_AllBVHNodes = new ComputeBuffer<BVHNode>(m_Context,
ComputeMemoryFlags.ReadOnly | ComputeMemoryFlags.CopyHostPointer, listAllBVHNodes.ToArray());

        // calculating global types
        if (null != clInputOutput_AllBVHNodes) { clInputOutput_AllBVHNodes.Dispose();
clInputOutput_AllBVHNodes = null; }
        clInputOutput_AllBVHNodes = new ComputeBuffer<BVHNode>(m_Context,
ComputeMemoryFlags.ReadWrite | ComputeMemoryFlags.CopyHostPointer, listAllBVHNodes.ToArray());

        ;

        listBeginObjects.Clear();
        listAllBVHNodes.Clear();
        listAllBVHNodesType.Clear();
        // all levelX
        foreach (List<BVHNode> listLevelXBVHs in listAllLevelXBVHs)
        {
           listLevelXBVHs.Clear();
        }
        listAllLevelXBVHs.Clear();
        // all levelX offsets
        foreach (List<int> listLevelXBVHsOffsets in listAllLevelXBVHsOffsets)
        {
           listLevelXBVHsOffsets.Clear();
        }
        listAllLevelXBVHsOffsets.Clear();

        //m_mtxMutex.ReleaseMutex();
    }

    public void UpdateMatrices()
    {
        //m_mtxMutex.WaitOne();
```

```
    ComputeEventList eventList = new ComputeEventList();
    cmdQueue.WriteToBuffer(m_listMatrices.ToArray(), clInput_MatricesData, true, eventList);
    cmdQueue.Finish();
    foreach (ComputeEventBase eventBase in eventList) { eventBase.Dispose(); }
    eventList.Clear();

    //m_mtxMutex.ReleaseMutex();
}

public void RunTriangleShader()
{
    //m_mtxMutex.WaitOne();

    kernelTriangleShader.SetMemoryArgument(0, clInput_AllBVHNodesType);
    kernelTriangleShader.SetMemoryArgument(1, clInput_AllBVHNodes);
    kernelTriangleShader.SetMemoryArgument(2, clInput_MatricesData);
    kernelTriangleShader.SetMemoryArgument(3, clInputOutput_AllBVHNodes);

    int iCount = m_iNumBVHNodes;

    ComputeEventList eventList = new ComputeEventList();
    cmdQueue.Execute(kernelTriangleShader, null, new long[] { iCount }, null, eventList);
    cmdQueue.Finish();
    foreach (ComputeEventBase eventBase in eventList) { eventBase.Dispose(); }
    eventList.Clear();

    //m_mtxMutex.ReleaseMutex();
}

public void RunRefitTreeShader()
{
    //m_mtxMutex.WaitOne();

    ComputeEventList eventList = new ComputeEventList();

    KernelRefitTree_LevelX.SetMemoryArgument(1, clInputOutput_AllBVHNodes);

    for (int i = 0; i < listCLInput_RefitTree_LevelX.Count; i++)
    {
```

```
            int iCount = m_listRefitTree_LevelXSizes[i];
            if (iCount == 0) { continue; }

            ComputeBuffer<BVHNode> clInput_RefitTree_LevelX = listCLInput_RefitTree_LevelX[i];

            KernelRefitTree_LevelX.SetMemoryArgument(0, clInput_RefitTree_LevelX);

            cmdQueue.Execute(KernelRefitTree_LevelX, null, new long[] { iCount }, null, eventList);
        }

        cmdQueue.Finish();
        foreach (ComputeEventBase eventBase in eventList) { eventBase.Dispose(); }
        eventList.Clear();

        //m_mtxMutex.ReleaseMutex();
    }

    private Float3 cameraPos = new Float3();
    private Float3 cameraAt = new Float3();
    public Float3 GetCameraPos() { return cameraPos; }
    public Float3 GetCameraAt() { return cameraAt; }

    public void SetCamera(Vector3 pos, Vector3 at, Vector3 up, float angle, float zfar)
    {
        //m_mtxMutex.WaitOne();

        // pos
        cameraPos.m_X = pos.X;
        cameraPos.m_Y = pos.Y;
        cameraPos.m_Z = pos.Z;

        // at
        cameraAt.m_X = at.X;
        cameraAt.m_Y = at.Y;
        cameraAt.m_Z = at.Z;

        // up
        up = up.Normalized();
        Float3 inUp;
        inUp.m_X = up.X;
```

```
            inUp.m_Y = up.Y;
            inUp.m_Z = up.Z;

            // dir
            Vector3 dir = (at - pos).Normalized();
            Float3 inDir;
            inDir.m_X = dir.X;
            inDir.m_Y = dir.Y;
            inDir.m_Z = dir.Z;

            // right
            Vector3 right = Vector3.Cross(dir, up).Normalized();
            Float3 inRight;
            inRight.m_X = right.X;
            inRight.m_Y = right.Y;
            inRight.m_Z = right.Z;

            KernelCameraRays.SetValueArgument<Float3>(0, cameraPos);
            KernelCameraRays.SetValueArgument<Float3>(1, inUp);
            KernelCameraRays.SetValueArgument<Float3>(2, inDir);
            KernelCameraRays.SetValueArgument<Float3>(3, inRight);
            KernelCameraRays.SetValueArgument<float>(4, angle);
            KernelCameraRays.SetValueArgument<float>(5, zfar);
            KernelCameraRays.SetValueArgument<int>(6, m_iWidth);
            KernelCameraRays.SetValueArgument<int>(7, m_iHeight);
            KernelCameraRays.SetMemoryArgument(8, clInputOutput_Rays);

            ComputeEventList eventList = new ComputeEventList();
            cmdQueue.Execute(KernelCameraRays, null, new long[] { (m_iWidth + 7) / 8 * 8, (m_iHeight + 7) / 8 *
8 }, new long[] { 8, 8 }, eventList);
            cmdQueue.Finish();
            foreach (ComputeEventBase eventBase in eventList) { eventBase.Dispose(); }
            eventList.Clear();

            //m_mtxMutex.ReleaseMutex();
        }

        public void RunRayShader(byte iRed, byte iGreen, byte iBlue, byte iAlpha)
        {
            //m_mtxMutex.WaitOne();
```

```
            KernelRayShader.SetMemoryArgument(0, clInputOutput_Rays);
            KernelRayShader.SetMemoryArgument(1, clInputOutput_AllBVHNodes);
            KernelRayShader.SetMemoryArgument(2, clInput_BeginObjects);
            KernelRayShader.SetValueArgument<int>(3, m_iNumBeginObjects);
            KernelRayShader.SetValueArgument<int>(4, m_iWidth);
            KernelRayShader.SetValueArgument<int>(5, m_iHeight);
            KernelRayShader.SetValueArgument<byte>(6, iRed);
            KernelRayShader.SetValueArgument<byte>(7, iGreen);
            KernelRayShader.SetValueArgument<byte>(8, iBlue);
            KernelRayShader.SetValueArgument<byte>(9, iAlpha);
            KernelRayShader.SetMemoryArgument(10, clInput_Materials);
            KernelRayShader.SetMemoryArgument(11, clInput_TexturesData);
            KernelRayShader.SetMemoryArgument(12, clOutput_TextureBuffer);
            KernelRayShader.SetValueArgument<Float3>(13, cameraPos);
            KernelRayShader.SetValueArgument<Float3>(14, cameraAt);

            ComputeEventList eventList = new ComputeEventList();
            cmdQueue.Execute(KernelRayShader, null, new long[] { (m_iWidth + 7) / 8 * 8, (m_iHeight + 7) / 8 * 8
}, new long[] { 8, 8 }, eventList);
            cmdQueue.Finish();
            foreach (ComputeEventBase eventBase in eventList) { eventBase.Dispose(); }
            eventList.Clear();

            IntPtr source = cmdQueue.Map(clOutput_TextureBuffer, true, ComputeMemoryMappingFlags.Read, 0,
m_iWidth * m_iHeight * 4, eventList);
            cmdQueue.Finish();
            foreach (ComputeEventBase eventBase in eventList) { eventBase.Dispose(); }
            eventList.Clear();

            writeableBitmap.Lock();
            Win32.CopyMemory(writeableBitmap.BackBuffer, source, m_iWidth * m_iHeight * 4);
            writeableBitmap.AddDirtyRect(new Int32Rect(0, 0, m_iWidth, m_iHeight));
            writeableBitmap.Unlock();
            cmdQueue.Unmap(clOutput_TextureBuffer, ref source, null);
            cmdQueue.Finish();

            //m_mtxMutex.ReleaseMutex();
        }
```

```
public WriteableBitmap GetWriteableBitmap()
{
   return writeableBitmap;
}

static BBox GenBBox(Triangle tri)
{
   float fMinX = float.MaxValue;
   float fMinY = float.MaxValue;
   float fMinZ = float.MaxValue;

   fMinX = Math.Min(fMinX, tri.m_A.m_Vx);
   fMinX = Math.Min(fMinX, tri.m_B.m_Vx);
   fMinX = Math.Min(fMinX, tri.m_C.m_Vx);

   fMinY = Math.Min(fMinY, tri.m_A.m_Vy);
   fMinY = Math.Min(fMinY, tri.m_B.m_Vy);
   fMinY = Math.Min(fMinY, tri.m_C.m_Vy);

   fMinZ = Math.Min(fMinZ, tri.m_A.m_Vz);
   fMinZ = Math.Min(fMinZ, tri.m_B.m_Vz);
   fMinZ = Math.Min(fMinZ, tri.m_C.m_Vz);

   float fMaxX = float.MinValue;
   float fMaxY = float.MinValue;
   float fMaxZ = float.MinValue;

   fMaxX = Math.Max(fMaxX, tri.m_A.m_Vx);
   fMaxX = Math.Max(fMaxX, tri.m_B.m_Vx);
   fMaxX = Math.Max(fMaxX, tri.m_C.m_Vx);

   fMaxY = Math.Max(fMaxY, tri.m_A.m_Vy);
   fMaxY = Math.Max(fMaxY, tri.m_B.m_Vy);
   fMaxY = Math.Max(fMaxY, tri.m_C.m_Vy);

   fMaxZ = Math.Max(fMaxZ, tri.m_A.m_Vz);
   fMaxZ = Math.Max(fMaxZ, tri.m_B.m_Vz);
   fMaxZ = Math.Max(fMaxZ, tri.m_C.m_Vz);

   BBox bbox = new BBox();
```

```
            bbox.minx = fMinX;
            bbox.miny = fMinY;
            bbox.minz = fMinZ;
            bbox.maxx = fMaxX;
            bbox.maxy = fMaxY;
            bbox.maxz = fMaxZ;

            return bbox;
        }

        public void Resize(int iWidth, int iHeight)
        {
            //m_mtxMutex.WaitOne();

            m_iWidth = iWidth;
            m_iHeight = iHeight;

            int iSize = iWidth * iHeight;

            // rays
            if (null != clInputOutput_Rays) { clInputOutput_Rays.Dispose(); clInputOutput_Rays = null; }
            clInputOutput_Rays = new ComputeBuffer<Ray>(m_Context, ComputeMemoryFlags.ReadWrite, iSize);

            // texture
            if (null != clOutput_TextureBuffer) { clOutput_TextureBuffer.Dispose(); clOutput_TextureBuffer = null;
}
            clOutput_TextureBuffer = new ComputeBuffer<byte>(m_Context, ComputeMemoryFlags.ReadWrite,
iWidth * iHeight * 4);

            writeableBitmap = new WriteableBitmap(iWidth, iHeight, 1, 1, PixelFormats.Bgra32, null);

            //m_mtxMutex.ReleaseMutex();
        }

        public void Dispose()
        {
            //m_mtxMutex.WaitOne();

            m_listMaterials.Clear();
            m_listTexturesData.Clear();
```

```
        m_listMatrices.Clear();
        m_listObjects.Clear();
        m_listRefitTree_LevelXSizes.Clear();

        // bufferek letrehozasa, device-onkent
        // texturak betoltese
        if (null != clInput_Materials) { clInput_Materials.Dispose(); clInput_Materials = null; }

        if (null != clInput_TexturesData) { clInput_TexturesData.Dispose(); clInput_TexturesData = null; }

        // matrixok betoltese
        if (null != clInput_MatricesData) { clInput_MatricesData.Dispose(); clInput_MatricesData = null; }

        // Global level N
        foreach (ComputeBuffer<BVHNode> clInput_RefitTree_LevelX in listCLInput_RefitTree_LevelX)
        {
           clInput_RefitTree_LevelX.Dispose();
        }
        listCLInput_RefitTree_LevelX.Clear();

        // global types
        if (null != clInput_AllBVHNodesType) { clInput_AllBVHNodesType.Dispose();
clInput_AllBVHNodesType = null; }

        // start global nodes
        if (null != clInput_AllBVHNodes) { clInput_AllBVHNodes.Dispose(); clInput_AllBVHNodes = null; }

        // calculating global types
        if (null != clInputOutput_AllBVHNodes) { clInputOutput_AllBVHNodes.Dispose();
clInputOutput_AllBVHNodes = null; }

        // rendertarget
        if (null != clOutput_TextureBuffer) { clOutput_TextureBuffer.Dispose(); clOutput_TextureBuffer = null;
}

        if (null != kernelTriangleShader  ) { kernelTriangleShader  .Dispose(); kernelTriangleShader   = null; }
        if (null != KernelRefitTree_LevelX) { KernelRefitTree_LevelX.Dispose(); KernelRefitTree_LevelX =
null; }
        if (null != KernelCameraRays      ) { KernelCameraRays      .Dispose(); KernelCameraRays       = null; }
        if (null != KernelRayShader       ) { KernelRayShader       .Dispose(); KernelRayShader        = null; }
```

```
    if (null != cmdQueue          ) { cmdQueue          .Dispose(); cmdQueue          = null; }
    if (null != m_Program         ) { m_Program         .Dispose(); m_Program         = null; }
    if (null != m_Context         ) { m_Context.Dispose();          m_Context         = null; }

    //m_mtxMutex.ReleaseMutex();
}

enum BVHObjectType
{
    Static = 1,
    Dynamic = 2
}

//Mutex m_mtxMutex = new Mutex();

List<Material> m_listMaterials = new List<Material>();
List<byte> m_listTexturesData = new List<byte>();
List<Matrix> m_listMatrices = new List<Matrix>();
List<BVHObject> m_listObjects = new List<BVHObject>();
List<int> m_listRefitTree_LevelXSizes = new List<int>();

int m_iWidth;
int m_iHeight;

// opencl resources
ComputeContext m_Context;
ComputeProgram m_Program;
ComputeDevice m_Device;
ComputeCommandQueue cmdQueue;
ComputeKernel kernelTriangleShader;
ComputeKernel KernelRefitTree_LevelX;
ComputeKernel KernelCameraRays;
ComputeKernel KernelRayShader;

// textures
ComputeBuffer<Material> clInput_Materials = null;
ComputeBuffer<byte> clInput_TexturesData = null;

// matrices
ComputeBuffer<Matrix> clInput_MatricesData = null;
```

```
        int m_iNumBeginObjects;
        ComputeBuffer<int> clInput_BeginObjects = null;
        int m_iNumBVHNodes;
        ComputeBuffer<int> clInput_AllBVHNodesType = null;
        ComputeBuffer<BVHNode> clInput_AllBVHNodes = null;
        ComputeBuffer<BVHNode> clInputOutput_AllBVHNodes = null;
        List<ComputeBuffer<BVHNode>> listCLInput_RefitTree_LevelX = new
List<ComputeBuffer<BVHNode>>();
        ComputeBuffer<Ray> clInputOutput_Rays = null;

        // rendertarget
        ComputeBuffer<byte> clOutput_TextureBuffer = null;
        WriteableBitmap writeableBitmap = null;
    }
}
```

17.7 OpenCLScript.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace OpenCLRenderer
{
   public static class OpenCLScript
   {
      static string m_strVertexShader = @"";
      public static void SetVertexShader(string @strVertexShader)
      {
         m_strVertexShader = strVertexShader;
      }

      static string m_strRayShader = @"";
      public static void SetRayShader(string @strRayShader)
      {
         m_strRayShader = strRayShader;
      }

      public static string GetText()
      {
         return
@"
//#pragma OPENCL EXTENSION cl_khr_fp16 : enable

typedef struct
{
   float m11;
   float m12;
   float m13;
   float m14;

   float m21;
   float m22;
   float m23;
   float m24;
```

```
    float m31;
    float m32;
    float m33;
    float m34;

    float m41;
    float m42;
    float m43;
    float m44;
}
Matrix4x4;

float2 ToFloat2(float x, float y)
{
    float2 ret;
    ret.x = x;
    ret.y = y;
    return ret;
}

float3 ToFloat3(float x, float y, float z)
{
    float3 ret;
    ret.x = x;
    ret.y = y;
    ret.z = z;
    return ret;
}

float4 ToFloat4(float x, float y, float z, float w)
{
    float4 ret;
    ret.x = x;
    ret.y = y;
    ret.z = z;
    ret.w = w;
    return ret;
}
```

```
float4 Mult_Matrix4x4Float3(Matrix4x4 T, float3 v, float w)
{
   float4 ret;

   float4 v1 = ToFloat4(v.x, v.y, v.z, w);

   ret.x = dot(ToFloat4(T.m11, T.m12, T.m13, T.m14), v1);
   ret.y = dot(ToFloat4(T.m21, T.m22, T.m23, T.m24), v1);
   ret.z = dot(ToFloat4(T.m31, T.m32, T.m33, T.m34), v1);
   ret.w = dot(ToFloat4(T.m41, T.m42, T.m43, T.m44), v1);

   return ret;
}

float4 Mult_Matrix4x4Float4(Matrix4x4 T, float4 v)
{
   float4 ret;

   ret.x = dot(ToFloat4(T.m11, T.m12, T.m13, T.m14), v);
   ret.y = dot(ToFloat4(T.m21, T.m22, T.m23, T.m24), v);
   ret.z = dot(ToFloat4(T.m31, T.m32, T.m33, T.m34), v);
   ret.w = dot(ToFloat4(T.m41, T.m42, T.m43, T.m44), v);

   return ret;
}

Matrix4x4 Mult_Matrix4x4Matrix4x4(Matrix4x4 T2, Matrix4x4 T1)
{
   Matrix4x4 ret;

   float4 T1row1 = ToFloat4(T1.m11, T1.m12, T1.m13, T1.m14);
   float4 T1row2 = ToFloat4(T1.m21, T1.m22, T1.m23, T1.m24);
   float4 T1row3 = ToFloat4(T1.m31, T1.m32, T1.m33, T1.m34);
   float4 T1row4 = ToFloat4(T1.m41, T1.m42, T1.m43, T1.m44);

   float4 T2column1 = ToFloat4(T2.m11, T2.m21, T2.m31, T2.m41);
   float4 T2column2 = ToFloat4(T2.m12, T2.m22, T2.m32, T2.m42);
   float4 T2column3 = ToFloat4(T2.m13, T2.m23, T2.m33, T2.m43);
   float4 T2column4 = ToFloat4(T2.m14, T2.m24, T2.m34, T2.m44);
```

```
float m11 = dot(T1row1, T2column1);
float m12 = dot(T1row1, T2column2);
float m13 = dot(T1row1, T2column3);
float m14 = dot(T1row1, T2column4);

float m21 = dot(T1row2, T2column1);
float m22 = dot(T1row2, T2column2);
float m23 = dot(T1row2, T2column3);
float m24 = dot(T1row2, T2column4);

float m31 = dot(T1row3, T2column1);
float m32 = dot(T1row3, T2column2);
float m33 = dot(T1row3, T2column3);
float m34 = dot(T1row3, T2column4);

float m41 = dot(T1row4, T2column1);
float m42 = dot(T1row4, T2column2);
float m43 = dot(T1row4, T2column3);
float m44 = dot(T1row4, T2column4);

ret.m11 = m11;
ret.m12 = m12;
ret.m13 = m13;
ret.m14 = m14;

ret.m21 = m21;
ret.m22 = m22;
ret.m23 = m23;
ret.m24 = m24;

ret.m31 = m31;
ret.m32 = m32;
ret.m33 = m33;
ret.m34 = m34;

ret.m41 = m41;
ret.m42 = m42;
ret.m43 = m43;
ret.m44 = m44;
```

```
    return ret;
}

typedef struct
{
    float posx;
    float posy;
    float posz;
    float dirx;
    float diry;
    float dirz;
    float length;
}
Ray;

typedef struct
{
    float3 pos;
    float3 normal;
    float t;
    int materialId;
    float2 st;
    int isCollision;
    int objectId;
}
Hit;

typedef struct
{
    float vx;
    float vy;
    float vz;
    float nx;
    float ny;
    float nz;
    float tx;
    float ty;
    int numMatrices;
    int matrixId1;
    int matrixId2;
```

```
    int matrixId3;
    float weight1;
    float weight2;
    float weight3;
}
Vertex;

typedef struct
{
    Vertex a;
    Vertex b;
    Vertex c;
    int materialId;
    float normalx;
    float normaly;
    float normalz;
    float area;
}
Triangle;

typedef struct
{
    float minx;
    float miny;
    float minz;
    float maxx;
    float maxy;
    float maxz;
}
BBox;

typedef struct
{
    int id;
    Triangle triangle;
    BBox bbox;
    int left;
    int right;
}
BVHNode;
```

```
#define Static  1
#define Dynamic 2

typedef struct
{
   int type;
}
BVHNodeType;

typedef struct
{
   int offset;
   int count;
}
BVHNodeOffset;

typedef struct
{
   unsigned long offset;
   int width;
   int height;
}
Texture;

typedef struct
{
   Texture diffuseTexture;
   Texture specularTexture;
   Texture normalTexture;
}
Material;

typedef struct
{
   int red;
   int green;
   int blue;
   int alpha;
}
```

```
Color;

BBox GenBBox_Tri(Triangle tri)
{
   float fMinX = +10000000.0f;
   float fMinY = +10000000.0f;
   float fMinZ = +10000000.0f;

   fMinX = min(fMinX, tri.a.vx);
   fMinX = min(fMinX, tri.b.vx);
   fMinX = min(fMinX, tri.c.vx);

   fMinY = min(fMinY, tri.a.vy);
   fMinY = min(fMinY, tri.b.vy);
   fMinY = min(fMinY, tri.c.vy);

   fMinZ = min(fMinZ, tri.a.vz);
   fMinZ = min(fMinZ, tri.b.vz);
   fMinZ = min(fMinZ, tri.c.vz);

   float fMaxX = -10000000.0f;
   float fMaxY = -10000000.0f;
   float fMaxZ = -10000000.0f;

   fMaxX = max(fMaxX, tri.a.vx);
   fMaxX = max(fMaxX, tri.b.vx);
   fMaxX = max(fMaxX, tri.c.vx);

   fMaxY = max(fMaxY, tri.a.vy);
   fMaxY = max(fMaxY, tri.b.vy);
   fMaxY = max(fMaxY, tri.c.vy);

   fMaxZ = max(fMaxZ, tri.a.vz);
   fMaxZ = max(fMaxZ, tri.b.vz);
   fMaxZ = max(fMaxZ, tri.c.vz);

   BBox bbox;
   bbox.minx = fMinX;
   bbox.miny = fMinY;
   bbox.minz = fMinZ;
```

```
    bbox.maxx = fMaxX;
    bbox.maxy = fMaxY;
    bbox.maxz = fMaxZ;

    return bbox;
}

BBox GenBBox_BBoxBBox(BBox bbox1, BBox bbox2)
{
    float fMinX = +10000000.0f;
    float fMinY = +10000000.0f;
    float fMinZ = +10000000.0f;

    fMinX = min(fMinX, bbox1.minx);
    fMinX = min(fMinX, bbox1.maxx);
    fMinX = min(fMinX, bbox2.minx);
    fMinX = min(fMinX, bbox2.maxx);

    fMinY = min(fMinY, bbox1.miny);
    fMinY = min(fMinY, bbox1.maxy);
    fMinY = min(fMinY, bbox2.miny);
    fMinY = min(fMinY, bbox2.maxy);

    fMinZ = min(fMinZ, bbox1.minz);
    fMinZ = min(fMinZ, bbox1.maxz);
    fMinZ = min(fMinZ, bbox2.minz);
    fMinZ = min(fMinZ, bbox2.maxz);

    float fMaxX = -10000000.0f;
    float fMaxY = -10000000.0f;
    float fMaxZ = -10000000.0f;

    fMaxX = max(fMaxX, bbox1.minx);
    fMaxX = max(fMaxX, bbox1.maxx);
    fMaxX = max(fMaxX, bbox2.minx);
    fMaxX = max(fMaxX, bbox2.maxx);

    fMaxY = max(fMaxY, bbox1.miny);
    fMaxY = max(fMaxY, bbox1.maxy);
    fMaxY = max(fMaxY, bbox2.miny);
```

```
    fMaxY = max(fMaxY, bbox2.maxy);

    fMaxZ = max(fMaxZ, bbox1.minz);
    fMaxZ = max(fMaxZ, bbox1.maxz);
    fMaxZ = max(fMaxZ, bbox2.minz);
    fMaxZ = max(fMaxZ, bbox2.maxz);

    BBox bbox;
    bbox.minx = fMinX;
    bbox.miny = fMinY;
    bbox.minz = fMinZ;
    bbox.maxx = fMaxX;
    bbox.maxy = fMaxY;
    bbox.maxz = fMaxZ;

    return bbox;
}

float3 scale4(float4 point, float scale)
{
        float3 ret;
        ret.x = point.x * scale;
        ret.y = point.y * scale;
        ret.z = point.z * scale;
        return ret;
}

float3 scale3(float3 point, float scale)
{
        float3 ret;
        ret.x = point.x * scale;
        ret.y = point.y * scale;
        ret.z = point.z * scale;
        return ret;
}

float2 scale2(float2 point, float scale)
{
        float2 ret;
        ret.x = point.x * scale;
```

```
            ret.y = point.y * scale;
            return ret;
}

float3 reflect(float3 dir, float3 n)
{
   float3 ret = dir - scale3(n, 2.0 * dot(dir, n));
   return ret;
}

typedef struct
{
   float x;
   float y;
   float z;
}
Vector3;

" + @m_strVertexShader + @"

__kernel void Main_TriangleShader(__global BVHNodeType *in_BVHNodeTypes, __global BVHNode
*in_BVHNodes, __global Matrix4x4 *in_Matrices, __global BVHNode *inout_BVHNodes)
{
   int id = get_global_id(0);

   BVHNodeType bvhNodeType = in_BVHNodeTypes[id];
   BVHNode inBVHNode = in_BVHNodes[id];
   BVHNode outBVHNode;

   if (Static == bvhNodeType.type)
   {
      outBVHNode = inBVHNode;
   }
   else if (Dynamic == bvhNodeType.type)
   {
      if (-1 == inBVHNode.left && -1 == inBVHNode.right)
      {
         outBVHNode = inBVHNode;
         outBVHNode.triangle.a = VertexShader(inBVHNode.triangle.a, in_Matrices);
         outBVHNode.triangle.b = VertexShader(inBVHNode.triangle.b, in_Matrices);
```

```
            outBVHNode.triangle.c = VertexShader(inBVHNode.triangle.c, in_Matrices);

            // normal
            float3 va = ToFloat3(outBVHNode.triangle.a.vx, outBVHNode.triangle.a.vy,
outBVHNode.triangle.a.vz);
            float3 vb = ToFloat3(outBVHNode.triangle.b.vx, outBVHNode.triangle.b.vy,
outBVHNode.triangle.b.vz);
            float3 vc = ToFloat3(outBVHNode.triangle.c.vx, outBVHNode.triangle.c.vy,
outBVHNode.triangle.c.vz);
            float3 normal = normalize(cross( vb - va, vc - va ));
            outBVHNode.triangle.normalx = normal.x;
            outBVHNode.triangle.normaly = normal.y;
            outBVHNode.triangle.normalz = normal.z;
            // area
            outBVHNode.triangle.area = length(cross((vb - va), (vc - va))) / 2.0f;

            // level 1
            outBVHNode.bbox = GenBBox_Tri(outBVHNode.triangle);
        }
        else
        {
            outBVHNode = inBVHNode;

            // level 2 - X
            //outBVHNode.bbox.minx = 0;
            //outBVHNode.bbox.miny = 0;
            //outBVHNode.bbox.minz = 0;
            //outBVHNode.bbox.maxx = 0;
            //outBVHNode.bbox.maxy = 0;
            //outBVHNode.bbox.maxz = 0;
        }
    }

    inout_BVHNodes[id] = outBVHNode;
}

__kernel void Main_RefitTree_LevelX(__global BVHNode *in_BVHNodes, __global BVHNode
*inout_allBVHNodes)
{
    int id = get_global_id(0);
```

```
    BVHNode node = in_BVHNodes[id];

    if (-1 != node.left && -1 != node.right)
    {
        BBox bbox1 = inout_allBVHNodes[node.left].bbox;
        BBox bbox2 = inout_allBVHNodes[node.right].bbox;
        inout_allBVHNodes[node.id].bbox = GenBBox_BBoxBBox(bbox1, bbox2);
    }
    else if (-1 != node.left)
    {
        inout_allBVHNodes[node.id].bbox = inout_allBVHNodes[node.left].bbox;
    }
    else if (-1 != node.right)
    {
        inout_allBVHNodes[node.id].bbox = inout_allBVHNodes[node.right].bbox;
    }
}

__kernel void Main_CameraRays(Vector3 in_Pos, Vector3 in_Up, Vector3 in_Dir, Vector3 in_Right, float
in_Angle, float in_ZFar, int in_Width, int in_Height, __global Ray *inout_Rays)
{
    int pixelx = get_global_id(0);
    int pixely = get_global_id(1);

    if (pixelx >= in_Width || pixely >= in_Height)
    {
        return;
    }

    int id = (in_Width * pixely) + pixelx;

    float3 pos   = ToFloat3(in_Pos.x, in_Pos.y, in_Pos.z);
    float3 up    = ToFloat3(in_Up.x, in_Up.y, in_Up.z);
    float3 dir   = ToFloat3(in_Dir.x, in_Dir.y, in_Dir.z);
    float3 right = ToFloat3(in_Right.x, in_Right.y, in_Right.z);

    float stepPerPixel = tan(in_Angle) / ((float)in_Height);

    int movePixelX = pixelx - (in_Width / 2);
```

```
        int movePixelY = pixely - (in_Height / 2);

    float3 moveUp = scale3(up, movePixelY * stepPerPixel);
    float3 moveRight = scale3(right, movePixelX * stepPerPixel);

    float3 dir2 = normalize(dir + moveUp + moveRight);

    Ray ray;
    ray.posx = pos.x;
    ray.posy = pos.y;
    ray.posz = pos.z;
    ray.dirx = dir2.x;
    ray.diry = dir2.y;
    ray.dirz = dir2.z;
    ray.length = in_ZFar;

    inout_Rays[id] = ray;
}

float3 Ray_GetPoint(Ray ray, float t)
{
    return ( ToFloat3(ray.posx, ray.posy, ray.posz) + ToFloat3(ray.dirx * t, ray.diry * t, ray.dirz * t) );
}

Hit Intersect_RayTriangle(Ray ray, Triangle tri)
{

    Hit ret;
    ret.isCollision = 0;
    ret.objectId = -1;

    float3 A      = ToFloat3(tri.a.vx, tri.a.vy, tri.a.vz);
    float3 B      = ToFloat3(tri.b.vx, tri.b.vy, tri.b.vz);
    float3 C      = ToFloat3(tri.c.vx, tri.c.vy, tri.c.vz);
    float3 nA     = ToFloat3(tri.a.nx, tri.a.ny, tri.a.nz);
    float3 nB     = ToFloat3(tri.b.nx, tri.b.ny, tri.b.nz);
    float3 nC     = ToFloat3(tri.c.nx, tri.c.ny, tri.c.nz);
    float2 tA     = ToFloat2(tri.a.tx, tri.a.ty);
    float2 tB     = ToFloat2(tri.b.tx, tri.b.ty);
    float2 tC     = ToFloat2(tri.c.tx, tri.c.ty);
```

```
    float3 normal = ToFloat3(tri.normalx, tri.normaly, tri.normalz);

    float cost = dot(ToFloat3(ray.dirx, ray.diry, ray.dirz), normal);
        if (fabs(cost) <= 0.001f)
            return ret;

        float t = dot(A - ToFloat3(ray.posx, ray.posy, ray.posz), normal) / cost;
        if(t < 0.001f)
            return ret;

        float3 P = Ray_GetPoint(ray, t);

    float3 edge1 = C - B;
    float3 vp1 = P - B;
    float area1 = length(cross(edge1, vp1)) / 2.0f;
    float u = area1 / tri.area;

    float3 edge2 = A - C;
    float3 vp2 = P - C;
    float area2 = length(cross(edge2, vp2)) / 2.0f;
    float v = area2 / tri.area;

    float3 edge3 = B - A;
    float3 vp3 = P - A;
    float area3 = length(cross(edge3, vp3)) / 2.0f;
    float w = area3 / tri.area;

    if ((u + v + w) > 1.001) { return ret; }

    ret.isCollision = 1;
    ret.pos = P;
    ret.normal = normalize((u * nA) + (v * nB) + ((1 - u - v) * nC));
    ret.t = t;
    ret.materialId = tri.materialId;
    ret.st = (u * tA) + (v * tB) + ((1 - u - v) * tC);

    return ret;
}
```

```
int Intersect_RayBBox(Ray ray, BBox bbox)
{
   float3 lb;
   lb.x = bbox.minx;
   lb.y = bbox.miny;
   lb.z = bbox.minz;

   float3 rt;
   rt.x = bbox.maxx;
   rt.y = bbox.maxy;
   rt.z = bbox.maxz;

   float3 dirfrac;
   dirfrac.x = 1.0f / ray.dirx;
   dirfrac.y = 1.0f / ray.diry;
   dirfrac.z = 1.0f / ray.dirz;

   float t1 = (lb.x - ray.posx) * dirfrac.x;
   float t2 = (rt.x - ray.posx) * dirfrac.x;
   float t3 = (lb.y - ray.posy) * dirfrac.y;
   float t4 = (rt.y - ray.posy) * dirfrac.y;
   float t5 = (lb.z - ray.posz) * dirfrac.z;
   float t6 = (rt.z - ray.posz) * dirfrac.z;

   float tmin = max(max(min(t1, t2), min(t3, t4)), min(t5, t6));
   float tmax = min(min(max(t1, t2), max(t3, t4)), max(t5, t6));

   // if tmax < 0, ray (line) is intersecting AABB, but the whole AABB is behind us
   if (tmax < 0.0f)
   {
      return 0;
   }

   // if tmin > tmax, ray doesn't intersect AABB
   if (tmin > tmax)
   {
      return 0;
   }

   return 1;
```

```
}

void WriteTexture(__global unsigned char *texture, int width, int height, float2 pixel, Color color)
{
   int id = (width * (int)pixel.y * 4) + ((int)pixel.x * 4);

   // clip
   int blue  = color.blue;  if (blue  < 0) { blue  = 0; } else if (blue  > 255) { blue  = 255; }
   int green = color.green; if (green < 0) { green = 0; } else if (green > 255) { green = 255; }
   int red   = color.red;   if (red   < 0) { red   = 0; } else if (red   > 255) { red   = 255; }
   int alpha = color.alpha; if (alpha < 0) { alpha = 0; } else if (alpha > 255) { alpha = 255; }

   texture[id + 0] = blue;
   texture[id + 1] = green;
   texture[id + 2] = red;
   texture[id + 3] = alpha;
}

Color ReadTexture(__global unsigned char *texture, int width, int height, float2 pixel)
{
   int id = (width * (int)pixel.y * 4) + ((int)pixel.x * 4);

   Color color;
   color.blue  = texture[id + 0];
   color.green = texture[id + 1];
   color.red   = texture[id + 2];
   color.alpha = texture[id + 3];

   return color;
}

Color ColorBlending(Color color1, Color color2, float t)
{
   float red = ((float)color1.red * (1.0f - t)) + ((float)color2.red * t);
   float green = ((float)color1.green * (1.0f - t)) + ((float)color2.green * t);
   float blue = ((float)color1.blue * (1.0f - t)) + ((float)color2.blue * t);
   float alpha = 255.0f;

   Color ret;
   ret.red = (unsigned char)red;
```

```
    ret.green = (unsigned char)green;
    ret.blue = (unsigned char)blue;
    ret.alpha = (unsigned char)alpha;
    return ret;
}

Color Tex2DDiffuse(__global Material *materials, __global unsigned char *textureDatas, int materialId, float2
st)
{
    Material material = materials[materialId];
    unsigned int offset = material.diffuseTexture.offset;
    int width = material.diffuseTexture.width;
    int height = material.diffuseTexture.height;
    __global unsigned char *texture = &(textureDatas[offset]);

    // repeat on
    while (st.x < 0.0f) { st.x += 1.0f; }
    while (st.y < 0.0f) { st.y += 1.0f; }
    while (st.x > 1.0f) { st.x -= 1.0f; }
    while (st.y > 1.0f) { st.y -= 1.0f; }

    float2 pixel;
    pixel.x = ((float)st.x * (float)width);
    pixel.y = ((float)st.y * (float)height);

    Color ret = ReadTexture(texture, width, height, pixel);
    return ret;
}

typedef struct
{
    int id;
    int count[64];
    Ray ray[6][64];
}
Rays;

typedef struct
{
    int id;
```

```
    int count[64];
    Hit hit[6][64];
}
Hits;

" + @m_strRayShader + @"

__kernel void Main_RayShader(__global Ray *in_Rays, __global BVHNode *in_BVHNodes, __global int
*in_BeginObjects, int in_NumBeginObjects, int in_Width, int in_Height, unsigned char red, unsigned char
green, unsigned char blue, unsigned char alpha, __global Material *materials, __global unsigned char
*textureDatas, __global unsigned char *out_Texture, Vector3 camPos, Vector3 camAt)
{
    int pixelx = get_global_id(0);
    int pixely = get_global_id(1);

    if (pixelx >= in_Width || pixely >= in_Height)
    {
        return;
    }


    int id = (in_Width * pixely) + pixelx;

    Rays rays;
    rays.id = 0;
    rays.count[rays.id] = 1;
    rays.ray[rays.id][0] = in_Rays[id];

    Hits hits;
    hits.id = rays.id;
    hits.count[rays.id] = 1;

    // clear
    Color background;
    background.red = red;
    background.green = green;
    background.blue = blue;
    background.alpha = alpha;
    WriteTexture(out_Texture, in_Width, in_Height, ToFloat2(pixelx, pixely), background);
```

```
bool isEnd = false;

do
{
   for(int ray_id = 0; ray_id < rays.count[rays.id]; ray_id++)
   {
      hits.id = rays.id;
      Ray ray = rays.ray[rays.id][ray_id];
      hits.count[rays.id] = rays.count[rays.id];
      hits.hit[rays.id][ray_id].isCollision = 0;
      hits.hit[rays.id][ray_id].t = 1000000.0f;
      hits.hit[rays.id][ray_id].objectId = -1;

      for (int i = 0; i < in_NumBeginObjects; i++)
      {
         int isSearching = 1;

         int rootId = in_BeginObjects[i];

         int stack[100];
         int top = 0;

         stack[top] = rootId;
         top++;

         while(isSearching == 1)
         {
            top--;
            if (top < 0) { isSearching = 0; continue; }
            BVHNode temp_node = in_BVHNodes[stack[top]];

            if (temp_node.left == -1 && temp_node.right == -1) // ha haromszog
            {
               // haromszog-ray utkozesvizsgalat
               Hit hit = Intersect_RayTriangle(ray, temp_node.triangle);
               hit.objectId = i;

               if (hit.isCollision == 1)
               {
                  if (hit.t < hits.hit[rays.id][ray_id].t)
```

```
                {
                    hits.hit[rays.id][ray_id] = hit;
                }
            }
        }

        if (temp_node.left != -1)
        {
            BVHNode node = in_BVHNodes[temp_node.left];
            if (1 == Intersect_RayBBox(ray, node.bbox))
            {
                stack[top] = temp_node.left;
                top++;
            }
        }
        if (temp_node.right != -1)
        {
            BVHNode node = in_BVHNodes[temp_node.right];
            if (1 == Intersect_RayBBox(ray, node.bbox))
            {
                stack[top] = temp_node.right;
                top++;
            }
        }
      }
    }
  }

  isEnd = RayShader(&hits, &rays, camPos, camAt, materials, textureDatas, out_Texture, in_Width,
in_Height, pixelx, pixely);

  //if(true == RayShader(&hits, &rays, materials, textureDatas, out_Texture, in_Width, in_Height, pixelx,
pixely))
  //{
  //    return;
  //}
 }
 while(isEnd == false);
}
";
```

```
        }
    }
}
```

17.8 VertexShader.txt

```
Vertex VertexShader(Vertex in, __global Matrix4x4 *in_Matrices)
{
   Vertex out;

   out = in;

   if (1 == in.numMatrices)
   {
      float3 v1 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId1], ToFloat4(in.vx, in.vy, in.vz, 1.0f)),
in.weight1);
      out.vx = v1.x;
      out.vy = v1.y;
      out.vz = v1.z;

      float3 n1 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId1], ToFloat4(in.nx, in.ny, in.nz, 0.0f)),
in.weight1);
      float3 n = normalize(n1);
      out.nx = n.x;
      out.ny = n.y;
      out.nz = n.z;
   }
   else if (2 == in.numMatrices)
   {
      float3 v1 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId1], ToFloat4(in.vx, in.vy, in.vz, 1.0f)),
in.weight1);
      float3 v2 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId2], ToFloat4(in.vx, in.vy, in.vz, 1.0f)),
in.weight2);
      out.vx = v1.x + v2.x;
      out.vy = v1.y + v2.y;
      out.vz = v1.z + v2.z;

      float3 n1 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId1], ToFloat4(in.nx, in.ny, in.nz, 0.0f)),
in.weight1);
      float3 n2 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId2], ToFloat4(in.nx, in.ny, in.nz, 0.0f)),
in.weight2);
      float3 n = normalize(n1 + n2);
      out.nx = n.x;
      out.ny = n.y;
      out.nz = n.z;
```

```
    }
    else if (3 == in.numMatrices)
    {
        float3 v1 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId1], ToFloat4(in.vx, in.vy, in.vz, 1.0f)),
in.weight1);
        float3 v2 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId2], ToFloat4(in.vx, in.vy, in.vz, 1.0f)),
in.weight2);
        float3 v3 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId3], ToFloat4(in.vx, in.vy, in.vz, 1.0f)),
in.weight3);
        out.vx = v1.x + v2.x + v3.x;
        out.vy = v1.y + v2.y + v3.y;
        out.vz = v1.z + v2.z + v3.z;

        float3 n1 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId1], ToFloat4(in.nx, in.ny, in.nz, 0.0f)),
in.weight1);
        float3 n2 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId2], ToFloat4(in.nx, in.ny, in.nz, 0.0f)),
in.weight2);
        float3 n3 = scale4(Mult_Matrix4x4Float4(in_Matrices[in.matrixId3], ToFloat4(in.nx, in.ny, in.nz, 0.0f)),
in.weight3);
        float3 n = normalize(n1 + n2 + n3);
        out.nx = n.x;
        out.ny = n.y;
        out.nz = n.z;
    }

    return out;
}
```

17.9 RayShader.txt

```
bool RayShader(Hits *hits, Rays *rays, Vector3 camPos, Vector3 camAt, __global Material *materials, __global unsigned char *textureDatas, __global unsigned char *out, int in_Width, int in_Height, int pixelx, int pixely)
{
    float3 light1;
    light1.x = +1000.0f;
    light1.y = +1000.0f;
    light1.z = +1000.0f;

    float3 light2;
    light2.x = -1000.0f;
    light2.y = +1000.0f;
    light2.z = +1000.0f;

    float3 light3;
    light3.x = 0.0f;
    light3.y = +1000.0f;
    light3.z = +1000.0f;

    float3 cam_pos;
    cam_pos.x = camPos.x;
    cam_pos.y = camPos.y;
    cam_pos.z = camPos.z;

    if (hits->id == 0)
    {
        Hit hit = hits->hit[hits->id][0];
        if (hit.isCollision == 0) { return true; }

        Ray newRay1; // light1
        newRay1.posx = light1.x;
        newRay1.posy = light1.y;
        newRay1.posz = light1.z;
        float3 dir1 = normalize(hit.pos - light1);
        newRay1.dirx = dir1.x;
        newRay1.diry = dir1.y;
        newRay1.dirz = dir1.z;
        newRay1.length = 5000.0f;

        Ray newRay2; // light 2
```

```
    newRay2.posx = light2.x;
    newRay2.posy = light2.y;
    newRay2.posz = light2.z;
    float3 dir2 = normalize(hit.pos - light2);
    newRay2.dirx = dir2.x;
    newRay2.diry = dir2.y;
    newRay2.dirz = dir2.z;
    newRay2.length = 5000.0f;

    Ray newRay3; // light3
    newRay3.posx = light3.x;
    newRay3.posy = light3.y;
    newRay3.posz = light3.z;
    float3 dir3 = normalize(hit.pos - light3);
    newRay3.dirx = dir3.x;
    newRay3.diry = dir3.y;
    newRay3.dirz = dir3.z;
    newRay3.length = 5000.0f;

    Ray newRay4; // reflection
    float3 pos = hit.pos + hit.normal * 0.01f;
    newRay4.posx = pos.x;
    newRay4.posy = pos.y;
    newRay4.posz = pos.z;
    float3 dir4 = reflect(normalize(hit.pos - cam_pos), hit.normal);
    newRay4.dirx = dir4.x;
    newRay4.diry = dir4.y;
    newRay4.dirz = dir4.z;
    newRay4.length = 5000.0f;

    rays->id = 1;
    rays->count[rays->id] = 4;
    rays->ray[rays->id][0] = newRay1;
    rays->ray[rays->id][1] = newRay2;
    rays->ray[rays->id][2] = newRay3;
    rays->ray[rays->id][3] = newRay4;

    return false;
}
```

```
if (hits->id == 1)
{
    Hit hit1 = hits->hit[0][0];
    if (hit1.isCollision == 0) { return true; }

    float diffuseIntensity = 0.0f;

    Hit hit2 = hits->hit[hits->id][0];
    if (hit2.isCollision == 1)
    {
        float length2 = length(light1 - hit2.pos);
        float length1 = length(light1 - hit1.pos);

        if ((length2 + 0.005f) > length1)
        {
            float3 dir = normalize(hit1.pos - light1);
            diffuseIntensity += max(dot(-dir, hit2.normal), 0.0f);// + max(dot(-dir2, hit.normal), 0.0f);
        }
    }

    Hit hit3 = hits->hit[hits->id][1];
    if (hit3.isCollision == 1)
    {
        {
            float length2 = length(light2 - hit3.pos);
            float length1 = length(light2 - hit1.pos);

            if ((length2 + 0.005f) > length1)
            {
                float3 dir = normalize(hit1.pos - light2);
                diffuseIntensity += max(dot(-dir, hit3.normal), 0.0f);
            }
        }
    }

    Hit hit4 = hits->hit[hits->id][2];
    if (hit4.isCollision == 1)
    {
        {
            float length2 = length(light3 - hit4.pos);
```

```
            float length1 = length(light3 - hit1.pos);

            if ((length2 + 0.005f) > length1)
            {
               float3 dir = normalize(hit1.pos - light3);
               diffuseIntensity += max(dot(-dir, hit4.normal), 0.0f);
            }
         }
      }

      Color textureColor = Tex2DDiffuse(materials, textureDatas, hit1.materialId, hit1.st);

      // diffuse
      Color diffuseColor;
      diffuseColor.red   = (int)(((float)textureColor.red  ) * diffuseIntensity);
      diffuseColor.green = (int)(((float)textureColor.green) * diffuseIntensity);
      diffuseColor.blue  = (int)(((float)textureColor.blue ) * diffuseIntensity);
      diffuseColor.alpha = 255;

      // reflection
      Hit hit0 = hits->hit[0][0];
      Hit hit5 = hits->hit[hits->id][3];
      if (hit5.isCollision == 1 && hit0.objectId == 0)
      {
         Color reflectionColor = Tex2DDiffuse(materials, textureDatas, hit5.materialId, hit5.st);

         float reflectionIntensity = diffuseIntensity * 0.25;
         diffuseColor.red   += (int)(((float)reflectionColor.red  ) * reflectionIntensity);
         diffuseColor.green += (int)(((float)reflectionColor.green) * reflectionIntensity);
         diffuseColor.blue  += (int)(((float)reflectionColor.blue ) * reflectionIntensity);
         diffuseColor.alpha = 255;
      }

      WriteTexture(out, in_Width, in_Height, ToFloat2(pixelx, pixely), diffuseColor);

      return true;
   }

   return true;
}
```